Universidad Católica Andrés Bello

Ejercicios Android

Prácticas para la materia Programación en Ambiente Android. Rev 1.0

Recopilado, traducido y adaptado por: José Gregorio Castillo Pacheco
25/05/2018

Contenido

Introducción 7
Capítulo 1: 8
La primera aplicación: “Hello World” 8
Tarea 1. Creación del proyecto 8
Tarea 2. Ejecutar la aplicación 9
Tarea 3. Creación de una interfaz de usuario sencilla 10
Tarea 4. Iniciar otra activity 12
Capítulo 2: 18
2.1 Vistas, ViewGroups, Layouts. 18
Tarea 1. Creación del proyecto 18
Tarea 2: Agreguar vistas a "Hello Toast" en el Editor de diseño 19
Tarea 3: Editar el diseño "Hello Toast" en XML 22
Tarea 4: Agregar controladores onClick para los botones 26
2.2 Usando Layouts 29
Resumen de la aplicación 29
Tarea 1: Cambiar el diseño a RelativeLayout 30
Tarea 2: Cambiar el layout a ConstraintLayout 37
Tarea 3: Crear variantes del diseño 49
2.3 Trabajando con Elementos TextView 51
Resumen de la aplicación 51
Tarea 1: Agregar varios text views 52
Tarea 2: Agregar enlaces Web activos y un ScrollView 58
Tarea 3: Desplazar elementos múltiples 63
Capítulo 3 66
3.1 Creando e Iniciando Activities 66
Tarea 1. Crear el Proyecto TwoActivities 68
Tarea 2. Crear e iniciar la segunda actividad 71
Tarea 3. Enviar datos desde la actividad principal a la segunda actividad 75
Tarea 4. Devolviendo datos a la main activity 81
Resumen 90
Capítulo 4: 92

4.1 Ciclo de vida de un Activity y el Instant State 92
Tarea 1. Agregar Métodos Callbacks del Ciclo de Vida al Proyecto TwoActivities 93
Tarea 2. Guardar y restaurar el estado de instancia (instance state) de la actividad 97
Resumen 101
4.2 Iniciar Activities con Intents Implícitos 102
Resumen de la aplicación 102
Tarea 1. Crear el nuevo proyecto y su layout 102
Tarea 2. Implementando el “open website” 106
Tarea 3. Implementando “open location” 108
Tarea 4. Implementando Share This Text 109
Tarea 5. Recibir intents implícitos 111
Resumen 114
Capítulo 5: 115
Interacción con el Usuario 115
5.1 Uso de teclados, controles de entrada, alertas y selectores 115
Tarea 1. Experimentar con los atributos de un teclado de ingreso de texto 117
Tarea 2. Cambiar el tipo de teclado 122
Tarea 3. Agregar un control de entrada spinner para seleccionar una etiqueta de teléfono 123
Tarea 4. Usar un dialogo para una alerta que requiere de alguna decisión 132
Tarea 5. Usar un picker como entrada de usuario 136
Tarea 6: Usando vistas de imágenes como botones 147
Tarea 7: Usando radio buttons (botones de selección) 153
Resumen 158
5.2 Usar un menú de opciones 159
Tarea 1. Agregar elementos al menú de opciones 162
Tarea 2. Agregar iconos para los elementos del menú 168
Tarea 3. Manejar el elemento seleccionado del menú 170
Resumen 172
5.3 Usando la Barra de Aplicaciones y las Pestañas para Navegación 173
Resumen de la aplicación 174
Tarea 1. Agregar un button Up para navegación ancestral 176
Tarea 2. Usar la navegación por pestañas con vistas por deslizamiento 177

Resumen 185
5.4 Crear un Recycler View 186
Resumen de la aplicación 186
Tarea 1. Crear y configurar un Proyecto nuevo 187
Tarea 2. Crear un dataset 189
Tarea 3. Crear un RecyclerView 190
Tarea 4. Hacer la lista interactiva 198
Tarea 5. Agregar un FAB para insertar elementos 198
Resumen 200
Capítulo 6: 202
Trabajando en el Background 202
6. 1 Crear un AsyncTask 203
Resumen de la aplicación 203
Tarea 1. Configurar el proyecto SimpleAsyncTask 204
Tarea 2. Crear la subclase de AsyncTask 205
Tarea 3. Implementar los pasos finales 209
Resumen 213
6.2 Conectarse a la Internet con AsynTask y AsyncTaskLoader 214
Resumen de la aplicación 214
Tarea 1. Explorar el API de Libros 215
Tarea 2. Crear la aplicación "Who Wrote It?" 216
Resumen 231
6.3 Broadcast Receivers 233
Resumen de la Aplicación 233
Tarea 1. Configurar el proyecto PowerReceiver 234
Tarea 2. Enviar y recibir un Broadcast personalizado 237
Resumen 240
6.4 Notifications 240
Resumen de la aplicación 241
Tarea 1. Crear una notificación básica 242
Tarea 2. Actualizar y cancelar la notificación 246
Tarea 3. Agregar acciones de notificación 249

Resumen 252
6.5 Gestor de Alarmas (Alarm Manager) 252
Resumen de la aplicación 253
Tarea 1. Configure el proyecto Stand Up! y las vistas 253
Tarea 2. Configurar la notificación 255
Tarea 3. Crear la repetición de la alarma 257
Resumen 262
6.6: Job Scheduler 263
Resumen de la Aplicación 263
Tarea 1. Implementar un JobService 264
Tarea 2. Implementar las condiciones del job 266
Resumen 278
Capitulo 7 279
Manejando Datos 279
7.1 Shared Preferences (Preferencias compartidas) 279
Resumen de la aplicación 280
Tarea 1. Explorar la aplicación HelloSharedPrefs 280
Tarea 2. Guardar y restaurar los datos en un archivo shared preferences 281
Resumen 286
7. 2 Agregando Settings a una Aplicación 288
Resumen de la aplicación 289
Tarea 1: Agregar un switch de ajuste para una aplicación 291
Task 2: Usando la plantilla Settings Activity 303
Resumen 315
7.3 Base de Datos SQLite 317
Introducción a SQL 317
SQLite Database 321
Resumen de la Aplicación 333
Tarea 0. Descargar y correr el código de arranque 335
Tarea 1. Extender la SQLiteOpenHelper para crear y popular la base de datos 335
Tarea 2. Creae el modelo de datos para una simple palabra 339
Tarea 3. Implementar el método query() en WordListOpenHelper 340

Tarea 4. Mostrar los datos en el RecyclerView 342
Tarea 5. Agregar nuevas palabaras a la base de datos 343
Tarea 6. Elimine palabras de la base de datos 347
Tarea 7. Actualizar palabras en la base de datos 349
Resumen 352
7.4 Búsqueda de Datos en una Base de Datos SQLite 353
Resumen de la Aplicación 353
Tarea 0. Descargar y ejecutar el código base 353
Tarea 1. Agregar la Búsqueda 354
Resumen 360
7.5 Implementando un Content Provider Minimalista 361
Resumen de la Applicación 363
Tarea 1. Crear el proyecto MinimalistContentProvider 364
Tarea 2. Crear una clase Contract, un URI y datos simulados 366
Tarea 3. Implementar la clase MiniContentProvider 369
Tarea 4. Usar un ContentResolver para obtener los datos 375
Resumen 378
7.6 Agregar un Content Provider a su Base de Datos 380
Resumen de la aplicación 380
Resumen de componentes de la aplicación 381
Resumen de cambios 382
Tarea 1. Descargar y ejecutar el código base 382
Tarea 2. Añadir un clase Contract a la WordListSQLInteractive 382
Tarea 3. Crear un Content Provider 385
Tarea 4. Implementar los métodos del Content Provider 394
Resumen 397
7.7 Compartiendo Contenido con Otras Aplicaciones 398
Resumen de la Aplicación 398
Tarea 1. Hacer que el content provider esté disponible para otras aplicaciones 399
Resumen 401
7.8 Cargar y Mostrar Datos Obtenidos de un Content Provider 402
Resumen de la Aplicación 403

Tarea 1. Crear la aplicación base para el WordListLoader 404
Tarea 2. MainActivity: Agregar un LoaderManager y LoaderCallbacks 408
Tarea 3. WordListAdapter: Implementar setData(), getItemCount(), y onBindViewHolder() 409
Resumen 411
Apéndices: 412
Copiar y renombrar un proyecto 412
1. Copiar el proyecto 412
2. Renombrar y refactorizar los componentes del proyecto 412
3. Actualizar los archivos build.gradle y AndroidManifest.xml 412
Bibliografía 414

Ejercicios Android (UCAB)

Introducción

El presente trabajo constituye la guía de estudios básica que será usada en el semestre para la materia “Desarrollo en Ambiente Android” para estudiantes de Ing de Telecomunicaciones y de Ing Informática de la Universidad Católica Andrés Bello.

Está pensado para que el estudiante pueda ir adquiriendo de forma progresiva los conocimientos básicos que le permitirán contar con las herramientas para el desarrollo de aplicaciones en esta plataforma. Aunque los ejercicios se plantean de la forma más sencilla posible es necesario que el estudiante conozca previamente los conceptos básicos del lenguaje Java, específicamente en lo relacionado con la Programación Orientada a Objetos. De requerir algún refuerzo de conocimientos se recomienda ver la serie de videos del canal Youtube de Mitocode.

El objetivo fundamental de este trabajo es complementar la parte teórica que se imparte en clase y el estudiante, por medio de la práctica, pueda afianzar el conocimiento recibido. A pesar de que son ejemplos sencillos están pensados para que el estudiante que desee profundizar en esta plataforma cuente con los elementos necesarios para que lo haga por cuenta propia.

Por ser la plataforma Android un ecosistema en constante evolución, esta guía se irá adaptando con el tiempo a medida que surgan cambios importantes en la plataforma. Es así que para el momento de redacción del presente documento Google ya había anunciado que se usará oficialmente Kotlin como el lenguaje oficial de desarrollo de en la plataforma Android, por lo que en revisiones futuras de esta guía de seguro se presentarán los ejercicios en ese lenguaje.

La herramienta de desarrollo a utilizar será el Android Studio y se le sugiere al alumno que previamente lo tenga instalado además del JDK de Java necesario para su funcionamiento. En la web encontrará mucha información sobre la instalación de ambas herramientas. Para Java puede consultar la página de descarga oficial de Java, mientras que para el Android Studio lo puede hace de la página de desarrolladores de Android.

Bienvenido al mundo de programación Android y ¡feliz codificación!

Capítulo 1:

La primera aplicación: “Hello World”.

Tarea 1. Creación del proyecto

Este ejercicio está diseñado para que el estudiante:

- Conozca el uso de la herramienta Android Studio,
- Cree un primer proyecto de Android
- Aplique los conocimientos recibidos durante la sesión de teoría
- Ejecute la aplicación en su propio teléfono

Paso a paso:

1. En la ventana **Welcome to Android Studio**, haz clic en **Start a new Android Studio project**.
2. En la pantalla **New Project**, ingresa los siguientes valores:
   - **Nombre de la app**: “My First App”
   - **Dominio de la empresa**: “example.com”
3. Haz clic en **Next**.
4. En la pantalla **Target Android Devices**, conserva los valores predeterminados y haz clic en **Next**.
5. En la pantalla **Add an Activity to Mobile**, selecciona **Empty Activity** y haz clic en **Next**.
6. En la pantalla **Configure Activity**, conserva los valores predeterminados y haz clic en **Finish**.

Luego de un tiempo de procesamiento, Android Studio abrirá el IDE.

Tómate un momento para revisar los archivos más importantes.

Primero, asegúrate de que la ventana **Project** esté abierta (selecciona **View > Tool Windows > Project**) y la vista **Android** esté seleccionada en la lista desplegable de la parte superior de la ventana. Podrás ver los siguientes archivos:

app > java > com.example.myfirstapp > MainActivity.java

Esta es la activity principal (el punto de entrada para tu app). Cuando compilas y ejecutas la app, el sistema inicia una instancia de esta **Activity** y carga su diseño.

app > res > layout > activity_main.xml

Este archivo XML define el diseño para la IU de la activity. Contiene un elemento **TextView** con el texto “Hello world!”.

app > manifests > AndroidManifest.xml

El **archivo de manifiesto** describe las características fundamentales de la app y define cada uno de sus componentes.

Gradle Scripts > build.gradle

Verás dos archivos con este nombre: uno para el proyecto y otro para el módulo de la “app”. Cada módulo tiene su propio archivo **build.gradle,** pero este proyecto por el momento tiene un solo módulo. Trabajarás principalmente con el archivo **build.gradle** del módulo para configurar la forma en que las herramientas de Gradle compilan y crean tu app. Para obtener más información sobre este archivo, visita **Configurar tu compilación**.

Tarea 2. Ejecutar la aplicación

Realizar la ejecución en un dispositivo real:

Configura tu dispositivo de la siguiente manera:

1. Conecta tu dispositivo a tu máquina de desarrollo con un cable USB. Si realizas desarrollos en Windows, es posible que debas **instalar el controlador USB adecuado** para tu dispositivo.
2. Habilita **USB debugging** en **Developer options** de la siguiente manera.

   Primero, debes habilitar las opciones para programador:

   1. Abre la app **Settings**.
   2. (Solo en Android 8.0 o versiones posteriores) Selecciona **System**.
   3. Desplázate hasta abajo y selecciona **About phone**.
   4. Desplázate hasta abajo y presiona **Build number** siete veces.
   5. Cuando regreses a la pantalla anterior, verás **Opciones para programador** cerca de la parte inferior.

   Abre **Developer options** y desplázate hacia abajo hasta encontrar y habilitar **USB debugging**.

Ejecuta la app en tu dispositivo de la siguiente manera:

1. En Android Studio, haz clic en el módulo **app** de la ventana **Project** y selecciona **Run > Run** (o haz clic en **Run** en la barra de herramientas).
2. En la ventana **Select Deployment Target**, selecciona tu dispositivo y haz clic en **OK**.

Realizar la ejecución en un emulador

Ejecuta la app en un emulador de la siguiente manera:

1. En Android Studio, haz clic en el módulo **app** de la ventana **Project** y selecciona **Run > Run** (o haz clic en **Run** en la barra de herramientas).
2. En la ventana **Select Deployment Target**, haz clic en **Create New Virtual Device**.
3. En la pantalla **Select Hardware**, selecciona un teléfono, como Pixel, y haz clic en **Next**.
4. En la pantalla **System Image**, selecciona la versión con el nivel de API más alto. Si no tienes instalada esa versión, se mostrará un vínculo **Download**en el que debes hacer clic para completar la descarga.
5. Haz clic en **Next**.

6. En la pantalla **Android Virtual Device (AVD)**, deja los ajustes tal como están y haz clic en **Finish**.
7. En el diálogo **Select Deployment Target**, selecciona el dispositivo que acabas de crear y haz clic en **OK**.

Android Studio instala la app en el emulador y la ejecuta.

Tarea 3. Creación de una interfaz de usuario sencilla

Abrir el editor de diseño

Para comenzar, configura tu lugar de trabajo de la siguiente manera:

1. En la ventana Project de Android Studio, abre **app > res > layout > activity_main.xml**.
2. Para hacer más espacio para el editor de diseño, oculta la ventana **Project** seleccionando **View > Tool Windows > Project** (o haciendo clic en **Project**  en el lado izquierdo de Android Studio).
3. Si en tu editor se muestra el código fuente XML, haz clic en la pestaña **Design** en la parte inferior de la ventana.
4. Haz clic en **Select Design Surface**  y selecciona **Blueprint**.
5. Haz clic en **Show**  en la barra de herramientas y asegúrate de que **Show Constraints** esté seleccionado.
6. Asegúrate de que Autoconnect esté desactivada. En la información sobre la herramienta de la barra de herramientas debe aparecer la opción **Turn On Autoconnect**  (ya que ahora está desactivada).
7. Haz clic en **Default Margins**  en la barra de herramientas y selecciona **16** (puedes ajustar el margen para cada vista posteriormente).
8. Haz clic en **Device in Editor**  en la barra de herramientas y selecciona **Pixel XL**.

Agregar un cuadro de texto

1. Primero debes quitar lo que ya está en el diseño. Para eso, haz clic en **TextView** en la ventana **Component Tree** y luego presiona Suprimir.
2. Desde la ventana **Palette**, a la izquierda, haz clic en **Text** en el subpanel izquierdo y luego arrastra **Plain Text** hacia el editor de diseño y suéltalo cerca de la parte superior del diseño. Este es un widget **EditText** que acepta entradas de texto sin formato.
3. Haz clic en la vista del editor de diseño. Ahora podrás ver los controladores para cambiar el tamaño en cada esquina (cuadrados) y los anclajes de restricción en cada lado (círculos).

Si deseas un mayor control, puedes acercar la vista en el editor mediante los botones de la barra de herramientas.

4. Haz clic sobre el anclaje del lado superior y, sin soltar el botón, arrástralo hacia arriba hasta que se acople a la parte superior del diseño y suéltalo. Eso es una restricción; especifica que la vista debe estar a 16 dp de la parte superior del diseño (ya que configuraste los márgenes predeterminados en 16 dp).

5. Del mismo modo, crea una restricción desde el lado izquierdo de la vista hasta el lado izquierdo del diseño.

Agregar un botón

1. Desde la ventana **Palette**, haz clic en **Widgets** en el subpanel izquierdo, arrastra **Button** hacia el editor de diseño y suéltalo cerca del lado derecho.

2. Crea una restricción desde el lado izquierdo del botón hasta el lado derecho del cuadro de texto.

3. Para restringir las vistas en una alineación horizontal, debes crear una restricción entre las líneas de base del texto. Para eso, haz clic en el botón y luego en **Edit Baseline**  , que se muestra justo debajo de la vista seleccionada en el editor de diseño. El anclaje de línea de base aparece dentro del botón. Haz clic sobre este anclaje y, sin soltar el botón, arrástralo hacia el anclaje de línea de base que se muestra en el cuadro de texto.

Cambiar las strings de IU

Para generar una vista previa de la IU, haz clic en **Select Design Surface**  en la barra de herramientas y selecciona **Design**. Ten en cuenta que la entrada de texto se completa previamente con “Name” y el botón lleva la etiqueta “Button”. Cambiarás estas strings.

1. Abre la ventana **Project** y luego abre **app > res > values > strings.xml**.

   Este es un archivo de **recursos de strings** en el que debes especificar todas tus strings de IU. Esto te permite controlar todas las strings de IU en una única ubicación, lo que permite encontrarlas, actualizarlas y localizarlas (si se compara con fijar strings en el código de tu diseño o tu app) más fácilmente.

2. Haz clic en **Open editor** en la parte superior de la ventana del editor. Con esto se abre **Translations Editor**, que proporciona una interfaz simple para agregar y editar tus strings predeterminadas, y ayuda a mantener organizadas todas tus strings traducidas.

   Haz clic en **Add Key**  a fin de crear una string nueva como “texto indicado” para el cuadro de texto.

   1. Escribe “edit_message” como nombre para la clave.

   2. Escribe “Enter a message” como valor.

   3. Haz clic en **OK**.

3. Agrega otra clave llamada “button_send” con el valor “Send”.

Ahora podrás configurar estas strings para cada vista. Para eso, regresa al archivo de diseño haciendo clic en **activity_main.xml** en la barra de pestañas y agrega las strings de la siguiente manera:

1. Haz clic en el cuadro de texto del diseño y, si la ventana **Attributes** no está visible a la derecha, haz clic en **Attributes** en la barra lateral derecha.
2. Ubica la propiedad **text** (actualmente fijada en "Name") y borra el valor.
3. Ubica la propiedad **hint** y haz clic en **Pick a Resource** a la derecha del cuadro de texto. Dentro del diálogo que se muestra, haz doble clic en **edit_message** en la lista.
4. Ahora haz clic en el botón en el diseño, ubica la propiedad **text**, haz clic en **Pick a Resource** y selecciona **button_send**.

Hacer que el tamaño del cuadro de texto sea flexible

Antes de continuar, haz clic en **Show** en la barra de herramientas y selecciona **Blueprint**.

1. Selecciona ambas vistas (haz clic con el botón primario en una, mantén presionado Shift y haz clic con el mismo botón en la otra) y luego haz clic con el botón secundario en cualquiera de las vistas y selecciona **Chain > Create Horizontal Chain**.

   Una **cadena** es una restricción bidireccional entre dos o más vistas que te permite organizar las vistas encadenadas de forma simultánea.
2. Selecciona el botón y abre la ventana **Attributes**. Usando el inspector de vistas de la parte superior de la ventana **Attributes**, fija el margen derecho en 16.
3. Ahora, haz clic en el cuadro de texto para ver sus atributos. Haz clic dos veces en el indicador de ancho para fijarlo en **Match Constraints**,

   "Match constraints" significa que el ancho se expande para cumplir con la definición de las limitaciones horizontales y los márgenes. Por lo tanto, el cuadro de texto se estirará para llenar el espacio horizontal (luego de considerar el botón y todos los márgenes).

Ejecutar la app

Si tu app ya está instalada en el dispositivo de **la lección anterior**, simplemente haz clic en **Apply Changes** en la barra de herramientas para actualizar la app con el nuevo diseño. También puedes hacer clic en **Run** para instalar y ejecutar la app.

Tarea 4. Iniciar otra activity

Después de completar la lección anterior, tendrás una app que mostrará una activity (una única pantalla) con un campo de texto y un botón. En esta lección, agregarás a MainActivity un fragmento de código que iniciará una nueva activity para mostrar el mensaje cuando el usuario toque **Send**.

Responder al botón Send

1. En el archivo **app > java > com.example.myfirstapp > MainActivity.java**, agrega el código auxiliar del método `sendMessage()` como se muestra a continuación:

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    /** Called when the user taps the Send button */
    public void sendMessage(View view) {
        // Do something in response to button
    }
}
```

   Es posible que veas un error, ya que Android Studio no puede resolver la clase **`View`** que se usa como argumento del método. Por lo tanto, haz clic para posicionar tu cursor en la declaración **`View`** y luego realiza una corrección rápida presionando Alt + Enter (en Mac, Option + Enter). (Si aparece un menú, selecciona **Import class**).

2. Regresa al archivo **activity_main.xml** para llamar a este método desde el botón:

   1. Haz clic para seleccionar el botón en el editor de diseño.
   2. En la ventana **Attributes**, encuentra la propiedad **onClick** y selecciona **sendMessage [MainActivity]** en la lista desplegable.

Ahora, cuando se presione el botón, el sistema llamará al método **`sendMessage().`**

Toma nota de los detalles de este método que se requieren para que el sistema lo reconozca como compatible con el atributo **`android:onClick`**. En concreto, el método debe declarar lo siguiente:

- acceso público;
- un valor de retorno vacío;
- una **`View`** como único parámetro (es el objeto **`View`** en el que se hizo clic).

Luego, completarás este método para leer el contenido del campo de texto y proporcionar dicho texto a otra activity.

Crear una intent

Un `Intent` es un objeto que proporciona enlace en tiempo de ejecución entre componentes separados, como dos activity. El `Intent` representa la “intención de hacer algo” de una app. Puedes usar las intents para varias tareas, pero en esta lección el intent inicia otra activity.

En `MainActivity.java`, agrega la constante `EXTRA_MESSAGE` y el código `sendMessage()`, como se muestra a continuación:

```
public class MainActivity extends AppCompatActivity {
    public static final String EXTRA_MESSAGE = "com.example.myfirstapp.MESSAGE";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    /** Called when the user taps the Send button */
    public void sendMessage(View view) {
        Intent intent = new Intent(this, DisplayMessageActivity.class);
        EditText editText = (EditText) findViewById(R.id.editText);
        String message = editText.getText().toString();
        intent.putExtra(EXTRA_MESSAGE, message);
        startActivity(intent);
    }
}
```

Android Studio nuevamente mostrará los errores **Cannot resolve symbol**. Presiona Alt + Enter (en Mac, Option + Return). Tus importaciones deben tener el siguiente aspecto final:

```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
```

Queda un error para `DisplayMessageActivity`, pero no hay inconveniente; podrás corregirlo en la sección siguiente.

Esto es lo que sucede en `sendMessage()`:

- El constructor de `Intent` toma dos parámetros:
  - un elemento `Context` como su primer parámetro (se usa `this` porque la clase `Activity` es una subclase de `Context`);
  - la `Class` del componente de la app a la cual el sistema debe entregar la `Intent` (en este caso, la activity que debe iniciarse).
- El método `putExtra()` agrega el valor de `EditText` a la intent. Una `Intent` puede llevar tipos de datos como pares clave-valor denominados *extras*. Tu clave es una constante pública `EXTRA_MESSAGE` porque la activity siguiente usa la clave para obtener el valor de texto. Se recomienda definir claves para los extras de intents usando el nombre del paquete de tu app como prefijo. Esto garantiza que las claves sean únicas, en el caso de que tu app interactúe con otras apps.
- El método `startActivity()` inicia una instancia de la `DisplayMessageActivity` especificada por la `Intent`. Ahora deberás crear esa clase.

Crear la segunda activity

1. En la ventana **Project**, haz clic con el botón secundario en la carpeta **app** y selecciona **New > Activity > Empty Activity**.
2. En la ventana **Configure Activity**, ingresa “DisplayMessageActivity” en **Activity Name** y haz clic en **Finish** (deja todas las demás propiedades en sus valores predeterminados).

Android Studio realiza tres acciones automáticamente:

- Crea el archivo `DisplayMessageActivity.java`.
- Crea el archivo de diseño `activity_display_message.xml` correspondiente.
- Agrega el elemento `<activity>` obligatorio en `AndroidManifest.xml`.

Si ejecutas la app y tocas el botón en la primera activity, la segunda se iniciará, pero estará vacía. Esto sucede porque la segunda activity usa el diseño vacío proporcionado por la plantilla.

Agregar una vista de texto

La activity nueva incluye un archivo de diseño en blanco, por lo que ahora agregarás una vista de texto en la que aparecerá el mensaje.

1. Abre el archivo **app > res > layout > activity_display_message.xml**.
2. Haz clic en **Turn On Autoconnect**  en la barra de herramientas (luego debe habilitarse, como se muestra en la figura 1).
3. En la ventana **Pallete**, haz clic en **Text** y luego arrastra un **TextView** hacia el diseño; ubícalo cerca de la parte superior del diseño, en un punto cercano al centro para que se acople a la línea vertical que aparece. La opción “Autoconnect” agrega restricciones a la derecha y a la izquierda para ubicar la vista en el centro horizontal.
4. Crea una restricción más desde la parte superior de la vista de texto hasta la parte superior del diseño.

Como alternativa, puedes realizar algunas modificaciones en el estilo de texto expandiendo **textAppearance** en la ventana **Attributes** y cambiar atributos como **textSize** y **textColor**.

Mostrar el mensaje

A continuación, modificarás la segunda activity para que muestre el mensaje que pasó la primera.

1. En `DisplayMessageActivity.java`, agrega el siguiente código al método `onCreate()`:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_display_message);

    // Get the Intent that started this activity and extract the string
```

```
    Intent intent = getIntent();
    String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);

    // Capture the layout's TextView and set the string as its text
    TextView textView = findViewById(R.id.textView);
    textView.setText(message);
}
```

2. Presiona Alt + Enter (en Mac, Option + Return) para importar las clases faltantes. Tus importaciones deben tener el siguiente aspecto final:

```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;
```

Agregar navegación superior

En cada pantalla de tu app que no sea la entrada principal (todas las pantallas que no sean la de inicio) se debe proporcionar navegación. De esta forma, el usuario puede regresar a la pantalla primaria lógica en la jerarquía de la app presionando el botón Up en la barra de la app.

Lo único que debes hacer es declarar la activity primaria lógica en el archivo `AndroidManifest.xml`. Para ello, abre el archivo en **app > manifests > AndroidManifest.xml**, ubica la etiqueta `<activity>` para `DisplayMessageActivity` y reemplázala por lo siguiente:

```
<activity android:name=".DisplayMessageActivity"
          android:parentActivityName=".MainActivity" >
    <!-- The meta-data tag is required if you support API level 15 and lower -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
</activity>
```

El sistema de Android agregará automáticamente el botón Up en la barra de la app.

Ejecutar la app

Ahora ejecuta nuevamente la app haciendo clic en **Apply Changes** en la barra de herramientas. Cuando se abra, escribe un mensaje en el campo de texto y toca **Send** para que el mensaje aparezca en la segunda activity.

Eso es todo. ¡Has compilado tu primera app de Android!

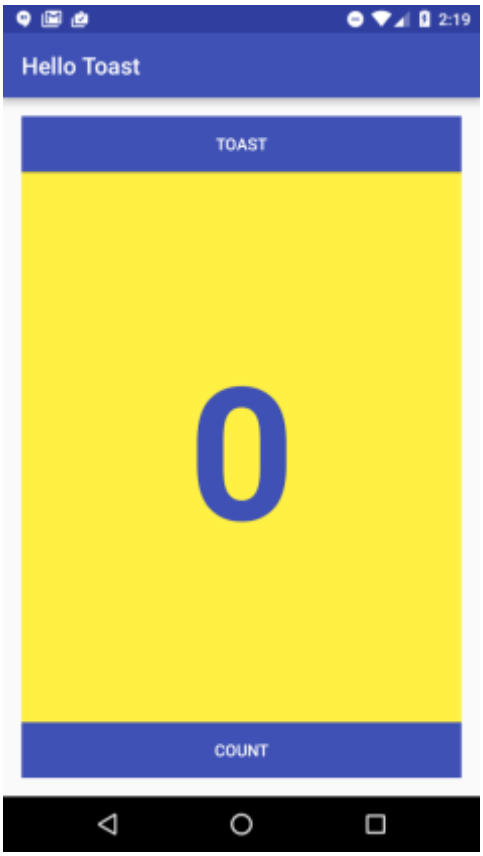
Capítulo 2:

2.1 Vistas, ViewGroups, Layouts.

HelloToast.

La aplicación "Hello Toast" constará de dos botones y una vista de texto. Cuando hace clic en el primer botón, se mostrará un mensaje corto o *toast* en la pantalla. Al hacer clic en el segundo botón aumentará un contador de clics; el recuento total de clics del mouse se mostrará en la vista de texto.

Así es como se verá la aplicación finalizada:



Tarea 1. Creación del proyecto

- Inicie Android Studio y cree un nuevo proyecto con los siguientes parámetros:

Atributo	Valor
Application Name	Hello Toast
Company Name	com.example.android

Phone and Tablet Minimum SDK	API15: Android 4.0.3 IceCreamSandwich
Template	Empty Activity
Generate Layout file box	Seleccionado
Backwards Compatibility box	Seleccionado

- Seleccione **Run > Run app** o haga clic en el icono **Run** en la barra de herramientas para compilar y ejecutar la aplicación en el emulador o su dispositivo.

Tarea 2: Agreguar vistas a "Hello Toast" en el Editor de diseño

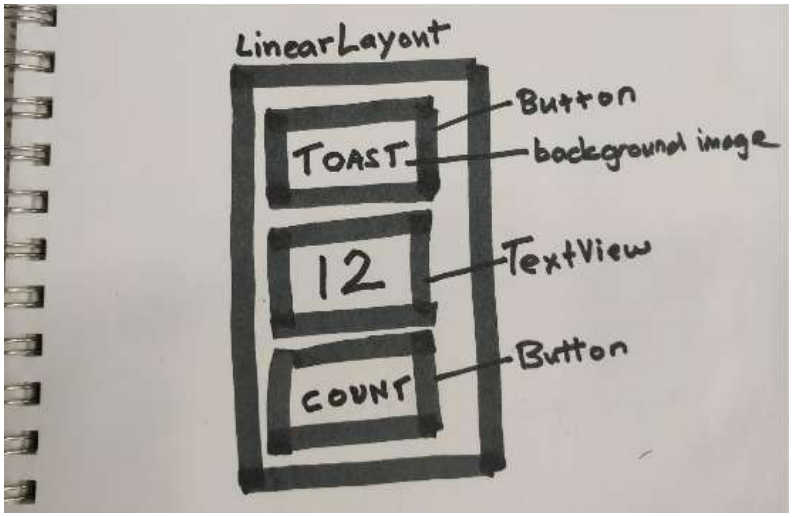
En esta tarea, creará y configurará una interfaz de usuario para la aplicación "Hello Toast" organizando los componentes de la interfaz de usuario de la vista en la pantalla.

¿Por qué?: cada aplicación debe comenzar con la experiencia del usuario, incluso si la implementación inicial es muy básica.

Las **Views** utilizadas para Hello Toast son:

- **TextView**: una vista que muestra texto.
- **Button**: un botón con una etiqueta que generalmente se asocia con un manejador de clics.
- **LinearLayout**: una vista que actúa como un contenedor para organizar otra vista. Este tipo de vista extiende la clase **ViewGroup** y también se llama grupo de vista. LinearLayout es un grupo de vista básico que organiza su colección de vistas en una fila horizontal o vertical.

Aquí hay un bosquejo de la UI que construirá en este ejercicio. Los bocetos de IU simples pueden ser muy útiles para decidir qué vistas usar y cómo organizarlas, especialmente cuando los diseños se vuelven más sofisticados.



2.1 Cambiar el grupo de vista a LinearLayout

La raíz de la jerarquía de vista es un **view group**, que como lo implica el nombre, es una vista que contiene otras vistas.

Un diseño LinearLayout es uno de los diseños más comunes. Es simple, rápido y siempre es un buen punto de partida. Cambie el grupo de vista a Vertical, LinearLayout de la siguiente manera:

1. En el panel **Component Tree** (recuerde lo visto en clase), busque la vista superior o raíz directamente debajo de la pantalla del dispositivo.
2. Haga clic en la pestaña **Text** para cambiar a la vista de código del diseño.
3. En la segunda línea del código, cambie el grupo de vista raíz a LinearLayout. La segunda línea de código ahora se ve así:

```
<LinearLayout xmlns:android=http://schemas.android.com/apk/res/android…
```

4. Asegúrese de que la etiqueta de cierre al final del código haya cambiado a `</LinearLayout>`. Si no ha cambiado automáticamente, cámbielo manualmente.
5. El `android: layout_height` se define como parte de la plantilla. La orientación de diseño predeterminada es una fila horizontal. Para cambiar el diseño para que sea vertical, agregue el siguiente código dentro de LinearLayout, debajo de `android: layout_height.`

```
android:orientation="vertical"
```

6. Desde la barra de menú, selecciona: **Code > Reformat Code…** Puede decir "No se han cambiado líneas: el código ya está formateado correctamente". (o en inglés: "No lines changed: code is already properly formatted")
7. Ejecute el código para asegurarse de que todavía funciona.
8. Regrese al Diseño.
9. Verifique en el panel **Component Tree** que el elemento superior ahora es un LinearLayout con su atributo de orientación establecido en "vertical".

Verificación del código: dependiendo de su versión de Android Studio, su código se verá más o menos así:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="hellotoast.android.example.com.hellotoast.MainActivity">


    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="Hello World!" />
</LinearLayout>
```

2.2 Agregar vistas al diseño lineal en el Editor de diseño

En esta tarea, eliminará el **TextView** actual (para la práctica) y agregará un nuevo **TextView** y dos botones al LinearLayout, tal como se muestra en el boceto de la UI para esta tarea. Consulte el diagrama de la UI anterior, si es necesario.

Agregar elementos de UI

1. Haga clic en la pestaña **Design** para mostrar el diseño del dispositivo virtual.
2. Haga clic en el **TextView** cuyo valor de texto es "Hola mundo" en el diseño de dispositivo virtual o en el panel **Component Tree**.
3. Presione la tecla Eliminar para eliminar ese TextView.
4. Desde el panel de **Palette**, arrastre y suelte un elemento Button, un TextView y otro elemento Button, en ese orden, uno debajo del otro en el diseño del dispositivo virtual.

Ajustar los elementos de UI

1. Para identificar cada vista de forma única dentro de una actividad, cada vista necesita un ID único. Y para que sea de utilidad, los botones necesitan etiquetas y el TextView debe mostrar algo de texto. Haga doble clic en cada elemento en el ComponentTree para ver sus propiedades y cambie los strings del texto y los ID de la siguiente manera:

Elemento	Text	ID
Button superior	Toast	button_toast
TextView	0	show_count
Button inferior	Count	button_count

2. Ejecute la aplicación

Solución al diseño:

Debería haber tres Vistas en su pantalla. No coincidirán con la imagen siguiente, pero mientras tengas tres Vistas en un diseño vertical, ¡estás bien!

Tarea 3: Editar el diseño "Hello Toast" en XML

En esta práctica, editará el código XML para el diseño de la UI de la aplicación Hello Toast. También editará las propiedades de las vistas que ha creado.

¿Por qué?: si bien el Editor de diseños es una herramienta poderosa, algunos cambios son más fáciles de realizar directamente en el código fuente XML. Es una preferencia personal utilizar el LayoutEditor gráfico o editar el archivo XML directamente.

1. Abra **res/layout/activity_main.xml** en el modo Texto.
2. En la barra de menú, seleccione **Code > Reformat Code**
3. Examine el código creado por el Editor de diseño.

3.1 Examinar las propiedades del LinearLayout

Se requiere que el LinearLayout tenga estas propiedades:

- layout_width
- layout_height
- orientation

El layout_width y el layout_height pueden tomar uno de estos tres valores:

- El atributo **match_parent** expande la vista para llenar su elemento principal por ancho o alto. Cuando el LinearLayout es la vista raíz, se expande al tamaño de la vista principal.
- El atributo **wrap_content** reduce las dimensiones de la vista lo suficientemente grandes como para incluir su contenido. (Si no hay contenido, la vista se vuelve invisible.)
- Use un número fijo de dp (device independent pixels) para especificar un tamaño fijo, ajustado para el tamaño de pantalla del dispositivo. Por ejemplo, "16dp" significa 16 píxeles independientes del dispositivo.

La orientación puede ser:

- **horizontal**: las vistas se organizan de izquierda a derecha.
- **vertical**: las vistas se organizan de arriba a abajo.

Cambie el LinearLayout de "Hello Toast" de la siguiente manera:

Propiedad	Valor
layout_width	match_parent (to llenar la pantalla)
layout_height	match_parent (to llenar la pantalla)
orientation	Vertical

3.2 Crear recursos para los strings

En lugar de “cablear” los strings en el código XML, una mejor práctica es usar recursos para los strings, los cuales representan tales strings.

¿Por qué?: tener los strings en un archivo separado facilita su administración, especialmente si usa esos strings más de una vez. Además, los recursos para los strings son obligatorios para traducir y localizar una aplicación, ya que creará un archivo de recursos para strings para cada idioma.

1. Coloque el cursor sobre la palabra "Toast".
2. Presione **Alt-Enter**.
3. Seleccione **Extract string resources**.
4. Establezca el **Resource name** en `button_label_toast` y haga clic en **OK**. (Si comete un error, deshaga el cambio con Ctrl-Z).

Esto crea un recurso para el estring en el archivo values/res/string.xml, y el string en el código se reemplaza con una referencia al recurso,

```
@string/button_label_toast
```

5. Extraiga y nombre las cadenas restantes de las vistas de la siguiente manera:

View	Valor del Recurso / String	Nombre del Recurso
Button	Hello Toast!	button_label_toast
TextView	0	count_initial_value
Button	Count	button_label_count

6. En la vista Proyecto, navegue a **values/strings.xml** para encontrar sus strings. Ahora puedes editar todos tus strings en un solo lugar.

3.3 Establecer colores y fondos

Los estilos y colores son propiedades adicionales que se pueden extraer en los recursos. Todas las vistas pueden tener fondos que pueden ser colores o imágenes.

¿Por qué?: la extracción de estilos y colores hace que sea fácil usarlos de manera consistente en toda la aplicación, y es sencillo cambiarlos en todos los elementos de la interfaz de usuario.

Experimenta con los siguientes cambios:

1) Cambie el tamaño del texto del TextView show_count. "sp" significa pixel independiente de escala, y como dp, es una unidad que se escala con la densidad de pantalla y la preferencia de tamaño de fuente del usuario. Se recomienda utilizar esta unidad cuando se especifiquen los tamaños de fuente, para que se ajusten tanto para la densidad de la pantalla como para las preferencias del usuario.

   ```
   android:textSize="160sp"
   ```

2) Extraiga el tamaño de texto del TextView como recurso de dimensión denominado `count_text_size`, de la siguiente manera:
   a) Haga clic en la pestaña **Text** para mostrar el código XML, si aún no lo ha hecho.
   b) Coloque el cursor en "160sp".
   c) Presione **Alt-Enter**.
   d) Clic **Extract dimension resource**.
   e) Establezca el **Resource name** para count_text_size y haga clic en **OK**. (Si comete un error, puede deshacer el cambio con Ctrl-Z).
   f) En la vista Proyecto, vaya a **values/dimens.xml** para encontrar sus dimensiones. El archivo dimens.xml se aplica a todos los dispositivos.
3) Cambie el `textStyle` del TextView `show_count` a negrita:

   ```
   android:textStyle="bold"
   ```

4) Cambie el color del texto del TextView `show_count` al color primario del tema. Mire el archivo de recursos colors.xml para ver cómo están definidos.

   El `ColorPrimary` es uno de los colores base predefinidos del tema y se usa para la barra de la aplicación. Por ejemplo, en una aplicación de producción, puede personalizar esto para que se ajuste a su marca. Usar los colores base para otros elementos de la interfaz de usuario crea una interfaz de usuario uniforme.

   ```
   android:textColor="@color/colorPrimary"
   ```

5) Cambie el color de ambos botones al color primario del tema.

   ```
   android:background="@color/colorPrimary"
   ```

6) Cambie el color del texto en ambos botones a blanco. El blanco es uno de los colores proporcionados como recurso de la plataforma Android (Android Platform Resource).

```
android:textColor="@android:color/white"
```

7) Agregue un color de fondo al TextView.

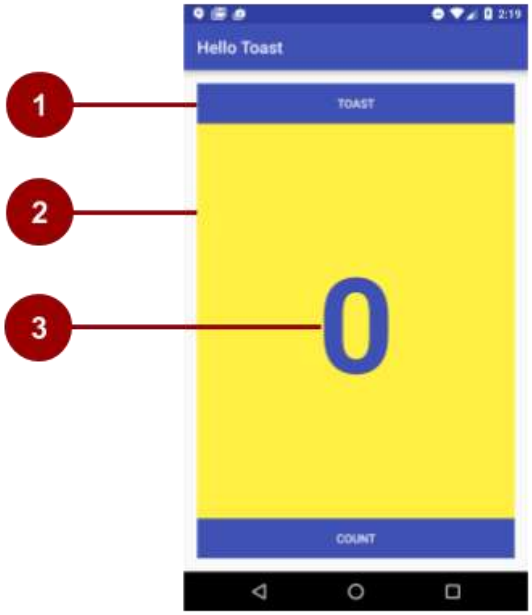
```
android:background="#FFF043"
```

8) En el Layout Editor (pestaña Texto), coloque el cursor del mouse sobre este color y presione **Alt-Enter**
9) Seleccione **Choose color**, lo cual muestra el selector de color, elija el color que desee o deje el amarillo actual, luego haga clic en **Choose**.
10) Abra **values/colors.xml**. Observe que **colorPrimary** que utilizó anteriormente está definido aquí.
11) Usando los colores en **values/colors.xml** como un ejemplo, agregue un recurso llamado `x|` para su color de fondo, y luego úselo para establecer el fondo del TextView.

```
<color name="myBackgroundColor">#FFF043</color>
```

3.4 Gravity y peso

Al especificar las propiedades de gravity y weight, obtiene un control adicional sobre la organización de vistas y el contenido en un linearLayout.



1. El atributo `android:layout_gravity` especifica cómo se alinea una vista dentro de su Vista padre. Debido a que las vistas coinciden con sus padres en ancho, no es necesario establecer esto explícitamente. Puede centrar una vista que es estrecha horizontalmente en su elemento padre: `android:layout_gravity = "center_horizontal"`
2. El atributo `android:layout_weight` indica cuanto del espacio adicional en el LinearLayout se asignará a las vistas que tienen este conjunto de parámetros. Si solo una vista tiene este atributo, obtiene todo el espacio extra de la pantalla. Para vistas múltiples, el espacio es prorrateado. Por ejemplo, si los botones tienen un peso de 1 y el textView 2, que suman un total de 4, los botones obtienen ¼ del espacio cada uno y la mitad el textView.

3. El atributo `android:gravity` especifica la alineación del contenido de una View dentro de la misma View. El contador es centrado en su view con: `android:gravity = "center"`

Haga lo siguiente:

1. Centre el texto en el TextView `show_count` horizontal y verticalmente: `android:gravity = "center"`
2. Haga que el TextView `show_count` se ajuste al tamaño de la pantalla:

```
android:layout_weight="2"
```

Tarea 4: Agregar controladores onClick para los botones

En esta tarea, agregará métodos a su MainActivity que se ejecutarán cuando el usuario haga clic en cada botón.

¿Por qué?: las aplicaciones interactivas deben responder a la entrada del usuario.

Para conectar una acción del usuario en una view al código de la aplicación, se debe hacer dos cosas:

- Escribir un método que realiza una acción específica cuando un usuario hace clic en un botón de la pantalla.
- Asociar ese método a la view, de manera que este método es llamado cuando el usuario interactúa con la vista.

4.1 Agregar una propiedad onClick a un botón

Un manejador de clic es un método que se invoca cuando el usuario hace clic en un elemento de la interfaz de usuario. En Android, se puede especificar el nombre del método del controlador de clic para cada vista en el archivo XML del layout con la propiedad `android:onClick.`

1. Abrir **res/layout/activity_main.xml**.
2. Agregue la siguiente propiedad al botón `button_toast`.

```
android:onClick="showToast"
```

3. Agregue el siguiente atributo al botón `button_count`.

```
android:onClick="countUp"
```

4. Dentro de activity_main.xml, coloque el cursor del mouse sobre cada uno de estos nombres de método.
5. Presione **Alt-Enter** y seleccione **Create onClick event handler**.
6. Elija la **MainACtivity** y haga clic en **OK**.

Esto crea plantillas de método para los métodos onClick en el archivo MainActivity.java.

4.2 Muestra un toast cuando se hace clic en el botón Toast

Un toast proporciona comentarios simples sobre una operación en una pequeña ventana emergente. Solo llena la cantidad de espacio requerido para el mensaje y la actividad actual

permanece visible e interactiva. Los toasts ofrecen otra forma de probar la interactividad de tu aplicación.

En `MainActivity.java`, agregue código al método `showToast()` para mostrar un mensaje tipo toast.

Para crear una instancia de Toast, llame al método factory `makeText()` de la clase Toast, proporcionando un contexto (ver a continuación), el mensaje que se mostrará y la duración de la visualización. Se mostrará el toast llamando al método `show()`. Este es un patrón común en el que puede reutilizar el código que va a escribir.

1. Obtenga el contexto de la aplicación.

   Mostrar un mensaje de Toast requiere de un contexto. El contexto de una aplicación contiene información global sobre el entorno de la aplicación. Dado que se muestra un toats sobre la UI visible, el sistema necesita información sobre la actividad actual.

   ```
   Contexto context = getApplicationContext ();
   ```

   Cuando ya se está en el contexto de la actividad cuyo contexto se necesita, también puede usar `this` como acceso directo al contexto.

2. La duración del string de un toast puede ser corta o larga, y se especifica cuál usando una constante Toast.

   - `Toast.LENGTH_LONG`
   - `Toast.LENGTH_SHORT`

   Las duraciones actuales son de aproximadamente 3,5 seg para la toast larga y 2 seg para la toast corta.

3. Cree una instancia de la clase Toast con el contexto, el mensaje y la duración.
   - El contexto es el contexto de la aplicación que obtuvimos antes.
   - El mensaje es el string que se desea mostrar.
   - La duración es una de las constantes predefinidas Toast.LENGTH_LONG o Toast.LENGTH_SHORT.

   ```
   Toast toast = Toast.makeText(context, "Hello Toast", Toast.LENGTH_LONG);
   ```

4. Extraiga el string "Hello Toast" en un recurso de string y llámelo `toast_message`.
   a) Coloque el cursor sobre el string "Hello Toast!".
   b) Presione **Alt-Enter**.
   c) Seleccione **Extract string resources**.
   d) Establezca el **Resource name** como `toast_message` y haga clic en **OK**.

   Esto almacenará "Hello World" como un recurso de string llamado `toast_message` en el archivo de recursos de string res/values/string.xml. El parámetro string en la llamada al método se reemplaza con una referencia al recurso.

- `R.` identifica al parámetro como un recurso.
- `string` hace referencia al nombre del archivo XML donde el recurso está definido.
- `toast_message` es el nombre del recurso.

```
Toast toast = Toast.makeText(context, R.string.toast_message, duration);
```

5. Mostrar el toast.

```
toast.show();
```

6. Ejecute su aplicación y verifique que el toast se muestra cuando se toca el botón Toast.

4.3 Aumentar el conteo en el TextView cuando se hace clic en el botón Contar

Para mostrar el recuento actual en la vista de texto:

- Mantener un registro del conteo mientras cambia.
- Enviar el recuento actualizado al TextView para mostrarlo en la interfaz de usuario.

Implemente esto de la siguiente manera:

1. En el archivo MainActivity.java, agregue una variable miembro privada denominada `mCount` para rastrear el recuento e iniciarlo en 0.
2. En MainActivity.java, agregue una variable miembro privada llamada `mShowCount` para obtener la referencia del TextView `show_count`.
3. En el método `countUp()`, aumente el valor de la variable de conteo cada vez que haga clic en el botón.
4. Obtenga una referencia al TextView usando el ID que estableció en el archivo layout.

   Las Views, así como los strings y las dimensiones, son recursos que pueden tener una identificación. La llamada al método `findViewById()` toma el ID de una vista como su parámetro y devuelve la view. Como el método devuelve una View, debe convertir el resultado al tipo de vista que espera (hacer casting), en este caso a (TextView).

   Para obtener este recurso solo una vez, use una variable miembro y configúrelo en `onCreate()`.

```
mShowCount = (TextView) findViewById(R.id.show_count);
```

5. Establezca el texto en el TextView al valor de la variable de conteo.

```
if (mShowCount != null)
        mShowCount.setText(Integer.toString(mCount));
```

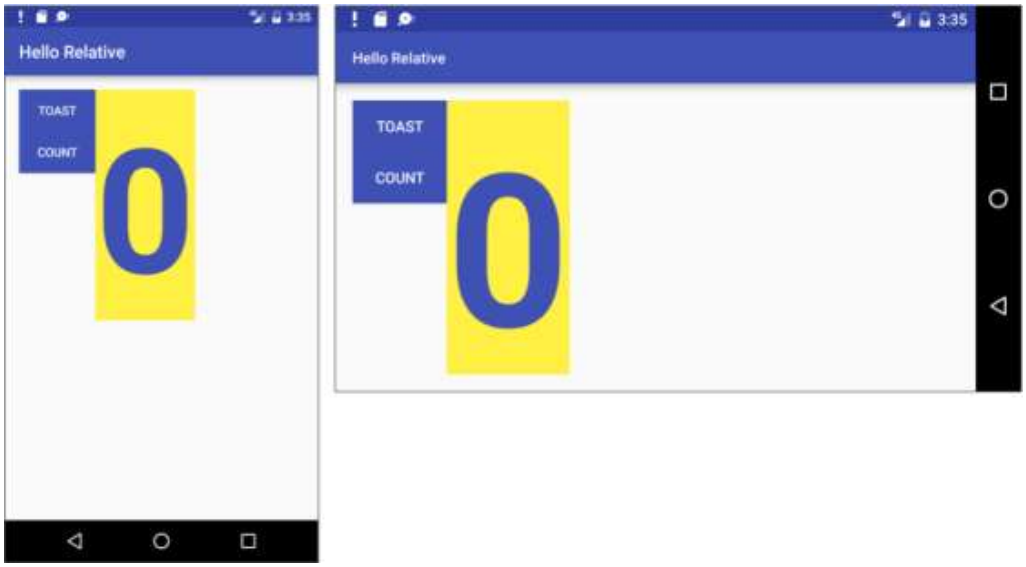
6. Ejecute su aplicación para verificar que el recuento aumenta cuando se presiona el botón Count.

2.2 Usando Layouts

Resumen de la aplicación

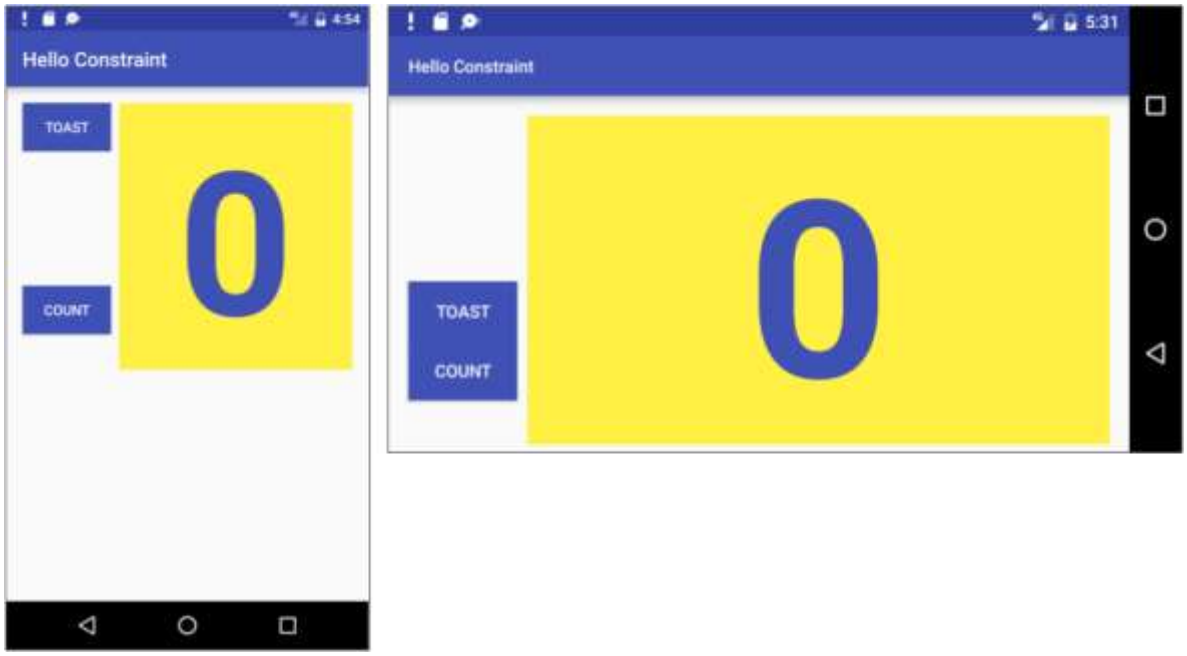
La aplicación Hello Toast en la práctica previa utiliza LinearLayout para organizar las vistas en el diseño de la actividad, como se muestra en la figura a continuación.

Para practicar el uso del editor de diseño, copie la aplicación Hello Toast y llamará a la nueva copia Hello Relative, para experimentar con RelativeLayout. Utilizará el editor de diseño para organizar las vistas en un diseño de UI diferente, como se muestra a continuación.



Finalmente, hará otra copia de la aplicación y la llamará Hello Restraint, y reemplazará LinearLayout por Restricción. ConstraintLayout ofrece más ayudas visuales y funciones de posicionamiento en el editor de diseño. Creará un layout de UI completamente diferente y

también variantes de layout para orientación horizontal y pantallas más grandes, como se muestra a continuación.



Tarea 1: Cambiar el diseño a RelativeLayout

Un **RelativeLayout** es una agrupación de vistas en la que cada vista se posiciona y alinea con respecto a otras vistas dentro del grupo. En esta tarea, investigará el uso del RelativeLayout.

1.1 Copiar y refactorizar la aplicación Hello Toast

1 Copie la carpeta del proyecto HelloToast, cambie el nombre a HelloRelative y vuelva a configurarlo. (Consulte el **Apéndice** para obtener instrucciones sobre cómo copiar un proyecto).

2 Después de refactorizar, cambie el valor <string name = "app_name"> en el archivo **strings.xml** (dentro de **app > res > values**) a Hello Relative (con un espacio) como el nombre de la aplicación.

1.2 Cambiar LinearLayout a RelativeLayout

1. Abra el archivo de diseño **activity_main.xml** y haga clic en la pestaña **Text** en la parte inferior del panel de edición para ver el código XML.
2. Cambie la etiqueta <LinearLayout en la parte superior a **<RelativeLayout** de manera que la sentencia luzca como se muestra aqui:

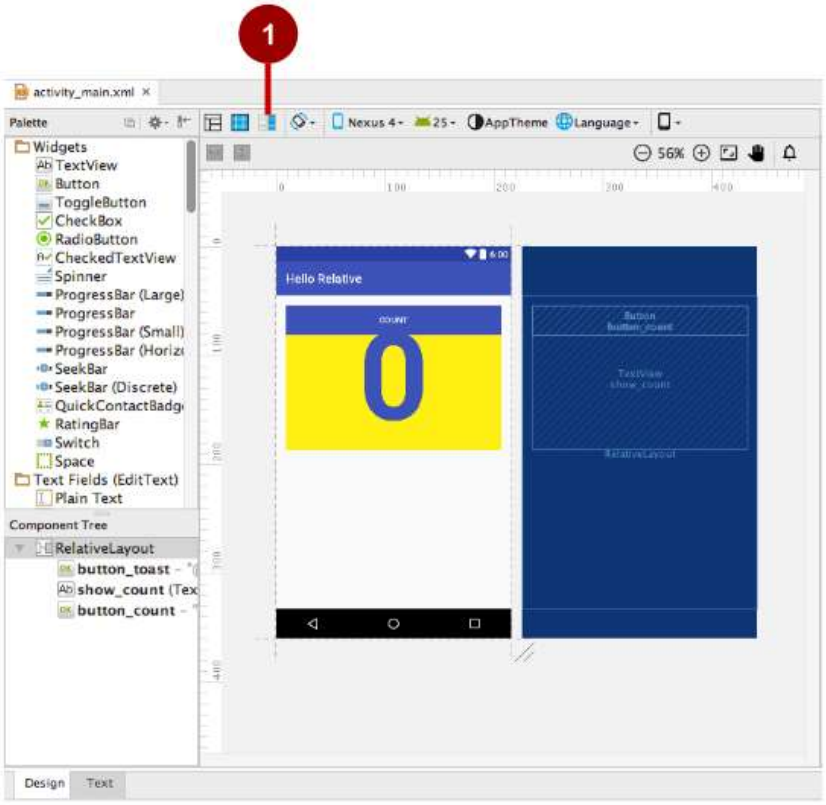
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

3. Desplácese hacia abajo para asegurarse de que la etiqueta de cierre `</LinearLayout>` también haya cambiado a `</ RelativeLayout>`; si no lo ha hecho, cámbielo manualmente.

1.3 Reorganizar vistas con la pestaña Design

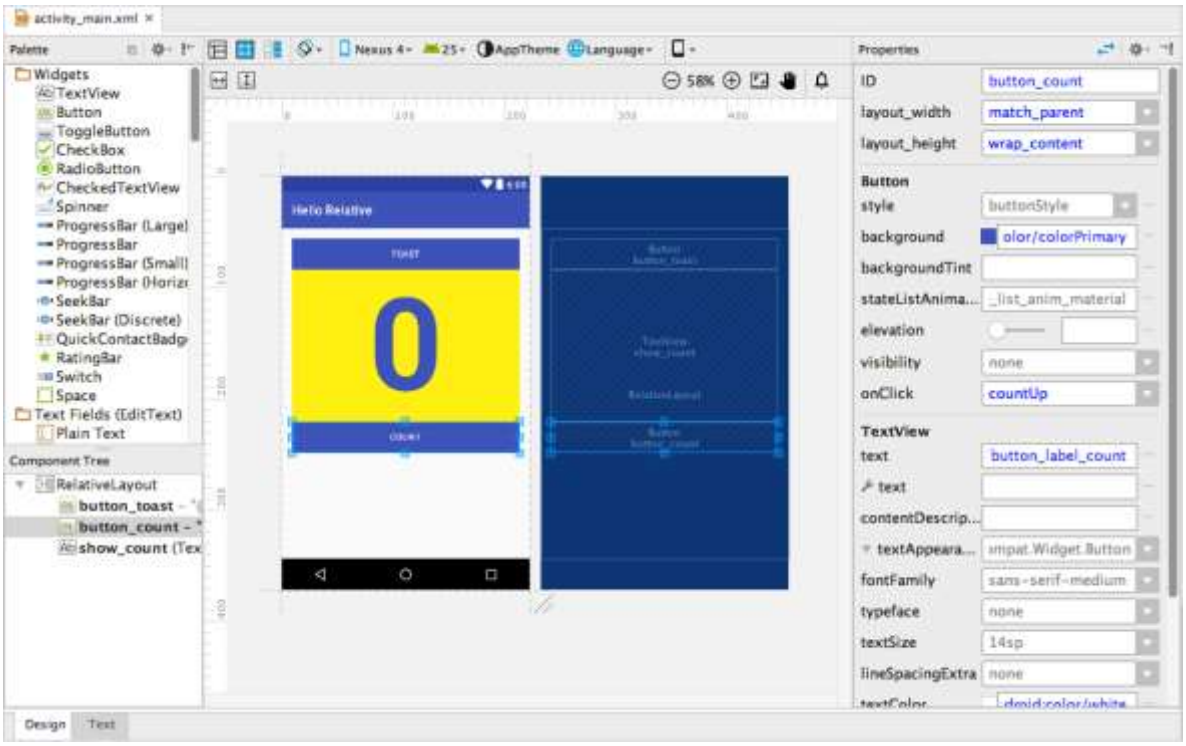
1 Haga clic en la pestaña **Design** en la parte inferior del panel de edición.

2 El panel de edición ahora debe verse como se muestra en la figura a continuación, con el layout del diseño y su plano. Si solo ve un layout del diseño o solo el plano, haga clic en el botón Mostrar diseño + plano (n. ° 1 en la figura siguiente):



3 Con el cambio a `RelativeLayout`, el editor de diseño cambia además alguno de los atributos de la vista. Por ejemplo:
- La vista `button_count` para el botón `COUNT` se superpone a la vista `button_toast` para el botón `TOAST`, por lo que no puede ver el botón `TOAST`. Sin embargo, en la vista plano, puede ver que los dos botones ocupan el mismo espacio.
- La parte superior de la vista `show_count` (que muestra 0) también se superpone a los botones `COUNT` y `TOAST`

4 Arrastre la vista `button_count` (para el botón `COUNT`) a un área debajo de la vista `show_count` (que muestra 0), y luego arrástrela hacia la parte inferior de la vista `show_count` hasta que encaje en su lugar como se muestra a continuación. También arrastre la vista `show_count` para que la parte superior de la vista encaje en la parte inferior de la vista `button_toast` para el botón `TOAST`.



Consejo: cuando selecciona una vista en el layout, sus propiedades aparecen en el panel Properties en el lado derecho del panel de edición. Estas propiedades corresponden a los atributos XML en el código XML para el layout, que examinará en el siguiente paso.

1.4 Examinar el código XML en la pestaña Text

Sigua estos pasos para ver cómo se ve la aplicación:

1. Ejecute la aplicación. La aplicación funciona de la misma manera que antes. La única diferencia es que el diseño usa un RelativeLayout para posicionar los elementos, en lugar de un LinearLayout. En la próxima tarea, cambiará el diseño de la UI.
2. Cambia la orientación del dispositivo o emulador a horizontal. Tenga en cuenta que la vista `button_count` desaparece porque el diseño de la pantalla no se adapta a la orientación horizontal. Arreglará este problema en una tarea posterior en esta práctica.
3. Haga clic en la pestaña **Text** en la parte inferior del panel de edición.
4. Examine los cambios en el código XML en el panel de edición como resultado de cambiar `LinearLayout` a `RelativeLayout`. Comience por examinar el segundo botón (`button_count`):

```
<Button
    android:id="@+id/button_count"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:background="@color/colorPrimary"
    android:onClick="countUp"
    android:text="@string/button_label_count"
```

```
    android:textColor="@android:color/white"
    android:layout_below="@+id/show_count"
    android:layout_centerHorizontal="true" />
```

Dos nuevos atributos XML se agregaron automáticamente por el editor de diseño después de mover el botón (`button_count`) en el diseño:

```
android:layout_below="@+id/show_count"
android:layout_centerHorizontal="true"
```

El atributo `android:layout_below` coloca la vista `button_count` directamente debajo de la vista `show_count`. Este atributo es uno de varios atributos para posicionar vistas dentro de un `RelativeLayout`; - simplemente coloca vistas en relación con el código XML para la vista `show_count`, que también movió en el editor de diseño, ahora está en una posición debajo de los dos botones en la vista de Text. Esto se debe al cambio de `LinearLayout` a `RelativeLayout`. La vista `show_count` ahora también incluye los siguientes atributos, como resultado de mover la vista en el editor de diseño:

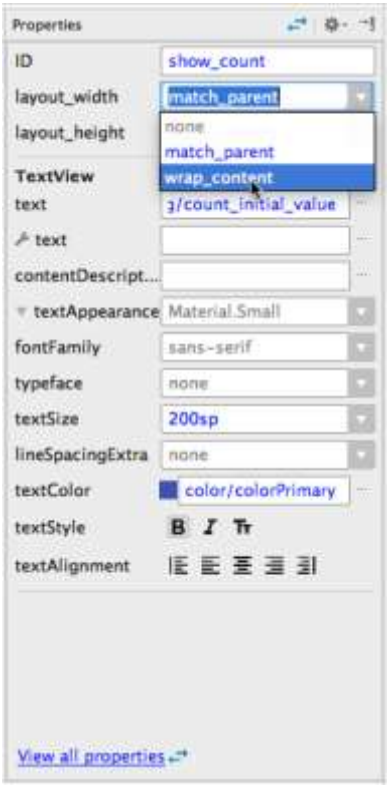
```
android:layout_below="@+id/button_toast"
android:layout_alignParentLeft="true"
android:layout_alignParentStart="true"
```

El `android:layout_alignParentLeft` alinea la vista al lado izquierdo del view group padre `RelativeLayout`. Si bien este atributo en sí mismo es suficiente para alinear la vista hacia el lado izquierdo, es posible que desee que la vista se alinee al lado derecho *si* la aplicación se está ejecutando en un dispositivo que usa un idioma de derecha a izquierda. Por lo tanto, el atributo `android:layout_alignParentStart` hace que el borde de "inicio" de esta vista coincida con el borde de inicio del elemento principal. El *inicio* es el borde izquierdo de la pantalla si la preferencia es de izquierda a derecha, o es el borde derecho de la pantalla si la preferencia es de derecha a izquierda.

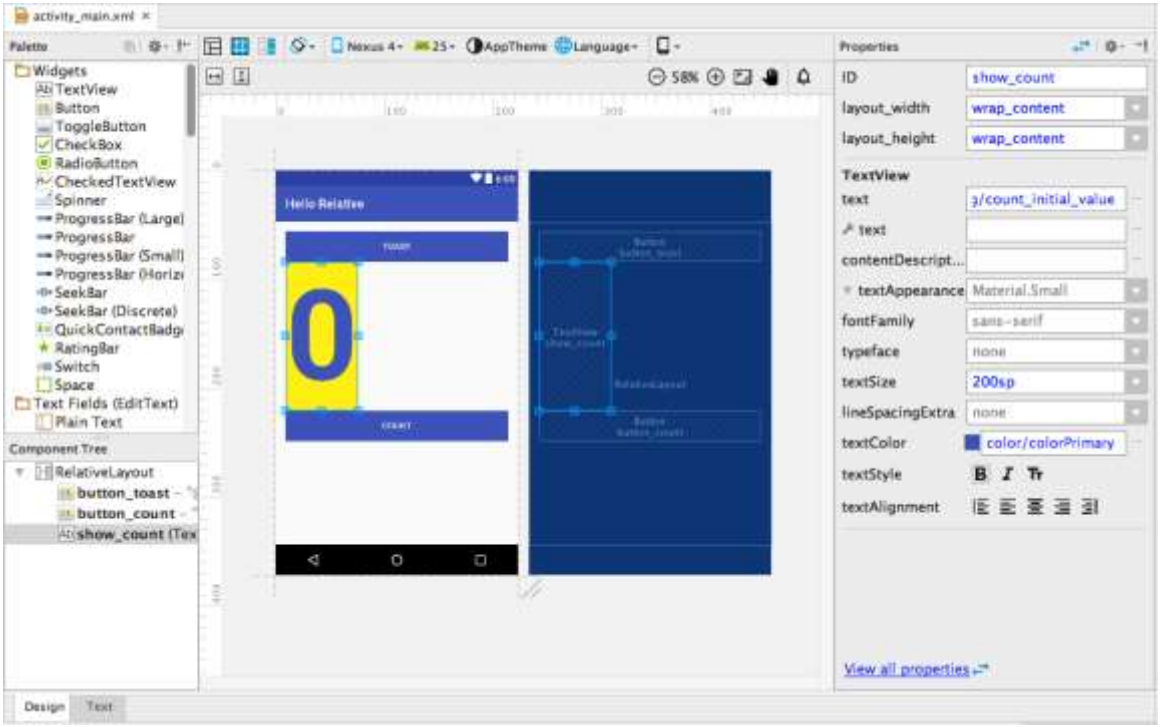
1.5 Reorganizar elementos en el RelativeLayout

1 Para experimentar más con `RelativeLayout`, seleccione nuevamente el layout activity_main.xml para editarlo (si no está ya seleccionado) y haga clic en la pestaña **Design** en la parte inferior del panel de edición.

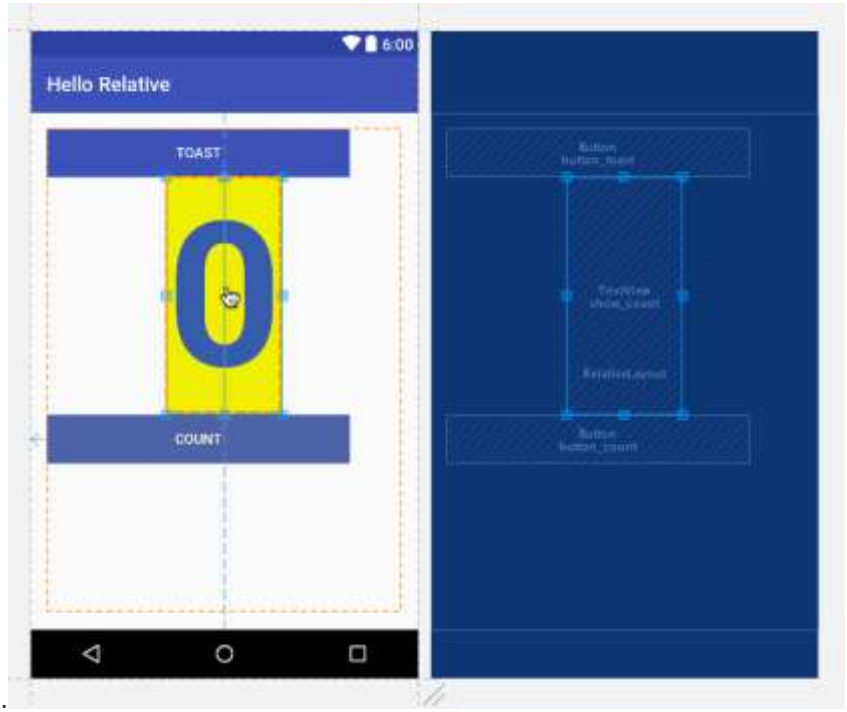
2 Seleccione la vista `show_count` en el diseño o el Component Tree, y en el panel Properties en el lado derecho de la ventana cambie la propiedad `layout_width` para **`wrap_content`** como se muestra en la figura siguiente.



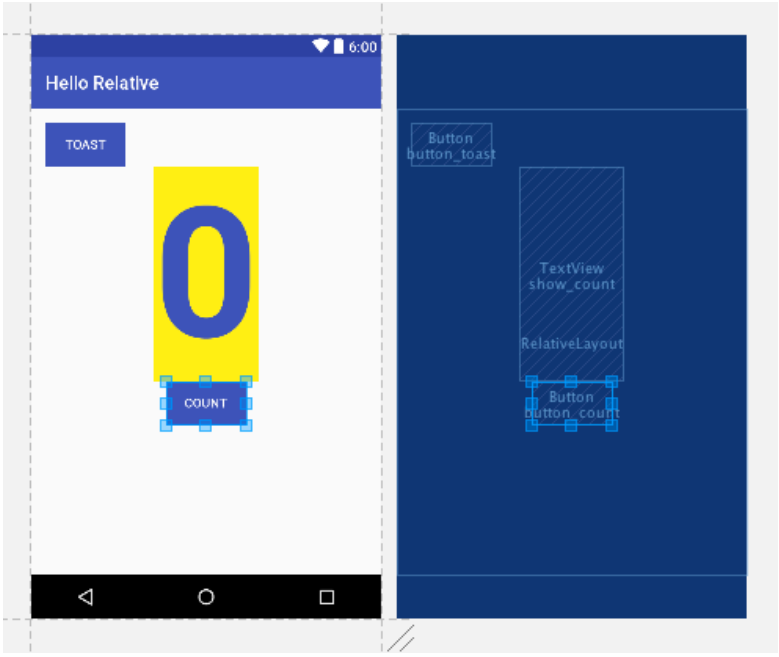
El editor de diseño muestra una vista más delgada de `show_count` alineada al lado izquierdo de la vista principal, como se muestra en la figura a continuación.



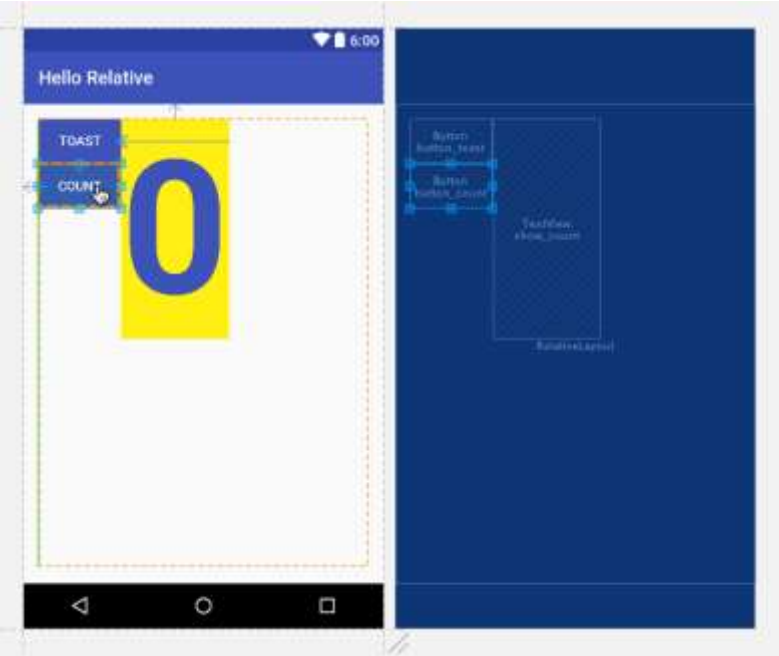
3 Arrastre la vista `show_count` horizontalmente al centro del diseño. A medida que arrastra la vista, aparece una guía central - el centro de la vista debe encajar en su lugar con la guía como se muestra.



4 Seleccione la vista `button_toast` y cambie su `layout_width` por `wrap_content` en el panel Propiedades, y luego cambie el `layout_width` de la vista `button_count` por `wrap_content`. El diseño ahora debería verse como la figura a continuación.



5 Arrastre la vista `button_count` hasta justo debajo de la vista `button_toast` para que encaje en la parte inferior de la vista `button_toast`, y arrastre la vista `show_count` hacia arriba junto al borde derecho de la `button_toastview` para que encaje en el borde derecho del botón. El diseño ahora debe verse como la figura a continuación:



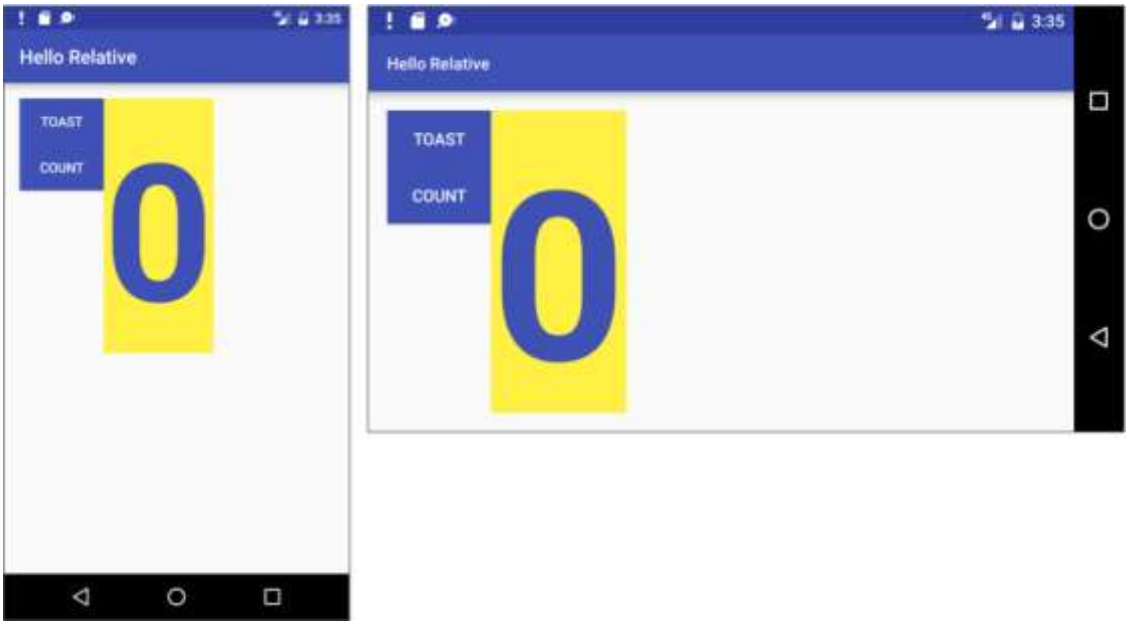
6. Haga clic en la pestaña Texto, en la parte inferior del panel de edición, y examine los cambios al código XML como resultado de mover las vistas en el diseño:
   - La vista `show_count` usa ahora los siguientes atributos para posicionarse a la *derecha* (toRightOf) y al *final* (toEndOf) de la vista `button_toast`:

```
android:layout_toRightOf="@+id/button_toast"
android:layout_toEndOf="@+id/button_toast"
```

   - La vista `button_count` ahora usa el siguiente atributo para ubicarlo *debajo* (below) de la vista `button_toast`:

```
android:layout_below="@+id/button_toast"
```

7 Ejecute la aplicación. La aplicación funciona de la misma manera que antes (ya que no cambiamos ningún código de Java). Sin embargo, el diseño es diferente, como se muestra en la figura a continuación. Cambie la orientación del dispositivo o del emulador a horizontal para ver si el nuevo diseño funciona para ambas orientaciones.



Consejo: Para aprender más acerca de cómo posicionar las vistas en un RelativeLayout, vea "Positioning Views" en el tópico "Relative Layout" de la Guía del API.

Tarea 2: Cambiar el layout a ConstraintLayout

ConstraintLayout es un view group disponible en la biblioteca Constraint Layout, que se incluye con Android Studio 2.2 y versiones posteriores. El diseño basado en restricciones permite al desarrollador crear diseños complejos sin tener que anidar grupos de vistas, lo que puede mejorar el rendimiento de la aplicación. Está integrado en el editor de diseño, de modo que se puede acceder a las herramientas de restricción desde la pestaña **Design** sin tener que editar el XML manualmente.

En esta tarea, copiará y refactorizará la aplicación Hello Toast para crear la aplicación Hello Constraint. A continuación, cambiará la raíz del view group de `LinearLayout` en el diseño principal para que sea `ConstraintLayout`. Después de cambiar el grupo de vista raíz, reorganizará las vistas en el diseño principal para tener restricciones que rijan su apariencia.

2.1 Copiar y refactorizar la aplicación Hello Toast

1. Copie la carpeta del proyecto HelloToast, cambie el nombre a HelloRelative y vuelva a configurarlo. (Consulte el **Apéndice** para obtener instrucciones sobre cómo copiar un proyecto).
2. Después de refactorizar, cambie el valor <string name = "app_name"> en el archivo **strings.xml** (dentro de **app > res > values**) a Hello Relative (con un espacio) como el nombre de la aplicación.

2.2 Agregar ConstraintLayout al proyecto

Asegúrese que el ConstraintLayout está disponible en el proyecto

1 En el Android Studio, seleccione **Tools > Android > SDK Manager**.
2 En el panel izquierdo, haga clic en **Android SDK**.
3 En el panel derecho, haga clic en la pestaña **SDK Tools** en la parte superior del panel.
4 Expanda **Support Repository**y observe si **ConstraintLayout for Android** y **Solver for Constraint Layout** están seleccionados.
- Si aparece “Installed” en la columna Status, está listo, hagla clic en **Cancel**.
- Si aparece “Not Installed” o “Update”:
  a. Haga click en cada uno de los checkbox al lado de **ConstraintLayout for Android** y **Solver for Constraint Layout**
  b. Haga clic en uno de los siguientes botones al pie de la ventana:
    - **Apply** para comenzar a instalar los componentes y permanecer en el SDK Manager para hacer otros cambios.
    - **OK** para instalar los componentes.
  c. Después de instalar los componentes (y hacer otros cambios si fuese necesario), haga clic en **Finish** para terminar de usar el SDK Manager.

2.3 Convertir un layout en ConstraintLayout

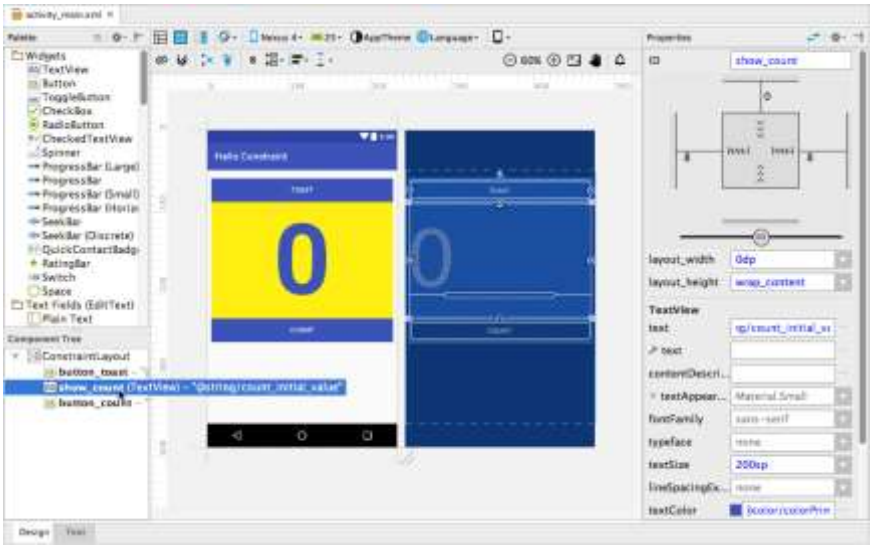
Android Studio tiene un convertidor integrado para ayudarlo a convertir un layout a `ConstraintLayout`. Siga los siguientes pasos:

1. Abra el archivo de layout (**activity_main.xml**) en Android Studio y haga clic en la pestaña **Design** en la parte inferior de la ventana del editor.
2. En la ventana Component Tree, haga clic con el botón derecho en **LinearLayout** y luego seleccione **Convert layout to ConstraintLayout** en el menú contextual.
3. El convertidor muestra una alerta con dos casillas de verificación ya marcadas. No las desmarque - asegúrese de que ambas opciones permanezcan *marcadas*:
   a. Flatten Layout Hierarchy: Esta opción This option removes all other nested layouts in the hierarchy. Esta opción elimina todos los demás diseños anidados en la jerarquía. El resultado es un diseño único y plano, que puede ser más eficiente para estos fines.
   b. Don't flatten layouts referenced from other files: Si un diseño particular define un `android:id attribute` al que se hace referencia en su código Java, es posible que no desee aplanar ese diseño porque es posible que su código ya no funcione. Sin embargo, en HelloConstraint, no tiene un `android:id attribute` para un layout, solo para las vistas.
4. En la alerta Add Project Dependency, haga clic en **OK** para agregar la librería contrsint-layout. Android Studio copia automáticamente la dependencia adecuada al archivo **build.gradle (Module: app)** del proyecto y sincroniza el cambio por usted. El editor de diseño vuelve a aparecer con `ConstraintLayout` como raíz de ese view group.

Nota: SI el editor de layout tiene un problema con el cambio, se verá una advertencia indicando Rendering Problems. Haga clic en **build** en el mensage `Tip: Try to build the project.` Esto resincronizará el archivo **build.gradle (Module: app)** del proyecto con la nueva dependencia.

5. El panel del Component Tree del editor de layout ahora muestra `ConstraintLayout` como la raíz del view group para el diseño con las otras vistas debajo de él, como se muestra en la figura a continuación. Haga clic en la vista **show_count** en el panel Component Tree.

La vista `show_count` también se selecciona en el blueprint, y sus propiedades aparecen en el panel Properties en el lado derecho.



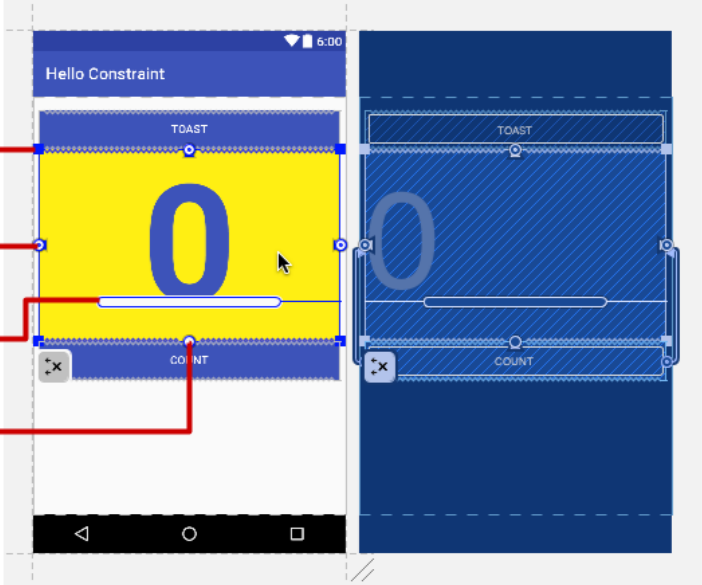
2.4 Convertir un layout en ConstraintLayout

El editor de diseño ofrece más funciones en la pestaña **Design** cuando utiliza un `ConstraintLayout`, que incluye más herramientas de diseño visual y una segunda fila de iconos para más herramientas.

El diseño visual y el blueprint ofrecen controladores para definir restricciones. Un *constraint* es una conexión o alineación con otra vista, con el diseño principal o con una línea invisible. Siga estos pasos para explorar las restricciones que Android Studio creó al convertir `LinearLayout` a `ConstraintLayout`:

1. Haga clic en la vista `show_count` en el panel Component Tree.
2. Desplace el cursor sobre la vista `show_count` en el layout, como se muestra en la figura siguiente.

Cada restricción aparece como una línea que se extiende desde un identificador circular. Cada vista tiene un identificador circular de restricción en el medio de cada lado. Después de seleccionar una vista en el panel Component Tree o hacer clic en el layout, la vista también muestra los controladores de cambio de tamaño en cada esquina.



En la figura superior:

1. **Resizing handle.**
2. **Constraint line and handle**. En la figura, la restricción alinea el lado izquierdo de la vista show_count con el lado izquierdo del botón button_toast.
3. **Baseline handle**. El identificador de línea de base alinea la línea base de texto de una vista con la línea base de texto de otra vista.
4. **Constraint handle** sin una llínea de restricción.

El editor de diseño también ofrece una fila de botones que le permiten configurar el aspecto del diseño:

En la figura superior:

1. **Design**, **Blueprint**, y **Both**: Hacer clic en el icono **Design** (primer icocno) para mostrar una vista previa en color de su layout. Hacer clicken el icono **Blueprint** (del centro) para mostrar solo los contornos para cada vista. Se pueden ver *ambas vistas* de lado a lado haciendo ckic en el tercer icono.
2. **Screen orientation**: Haga clic para rotar el dispositivo entre horizontal y vertical.
3. **Device type and size**: Permite seleccionar el tipo de dispositivo (teléfono/tablet, Android TV, o Android Wear) y la configuración de la pantalla (tamaño y densidad).
4. **API version**: Selecciona la version del API de Android en la que se prevee el diseño.
5. **App theme**: Selecciona el tema de UI para aplicar a la prevista.
6. **Language**: Selecciona el idioma a mostrar en los string de la UI. Esta lista muestra solo los idiomas disponibles en los recursos de string.
7. **Layout Variants**: Se intercambia a uno de los layout alernativos de este archive, o crea uno Nuevo.

Consejo: para aprender más acerca de cómo usar el editor de layout, ver Build a UI with Layout Editor. Para aprender más acerca de cómo construir un diseño con ConstraintLayout ver Build a Responsive UI with ConstraintLayout.

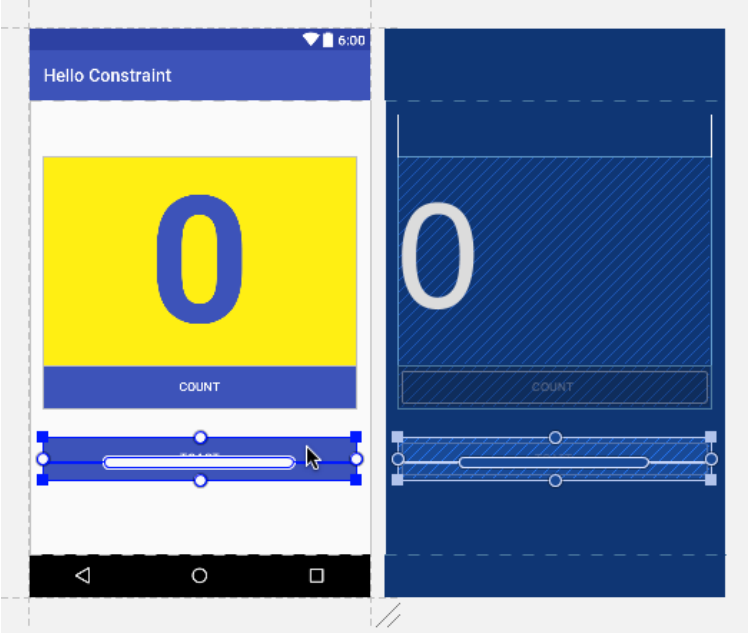
2.5 Eliminar restricciones

Android Studio infiere automáticamente las restricciones para los elementos de diseño al convertir un layout para usar ConstraintLayout. Sin embargo, las conjeturas pueden no ser lo que se desea. Siga estos pasos para borrar las restricciones con el fin de posicionar libremente los elementos en el diseño:

1 Hacer clic derecho **ConstraintLayout** en el panel Component Tree y seleccionar **Clear All Constraints**.

Consejo: También se puede eliminar una sola línea de restricción colocando el cursor sobre el controlador de restricción hasta que aparezca un círculo rojo y luego hacer clic en el controlador. El comando **Clear All Constraints** es más rápido para eliminar todas las restricciones.

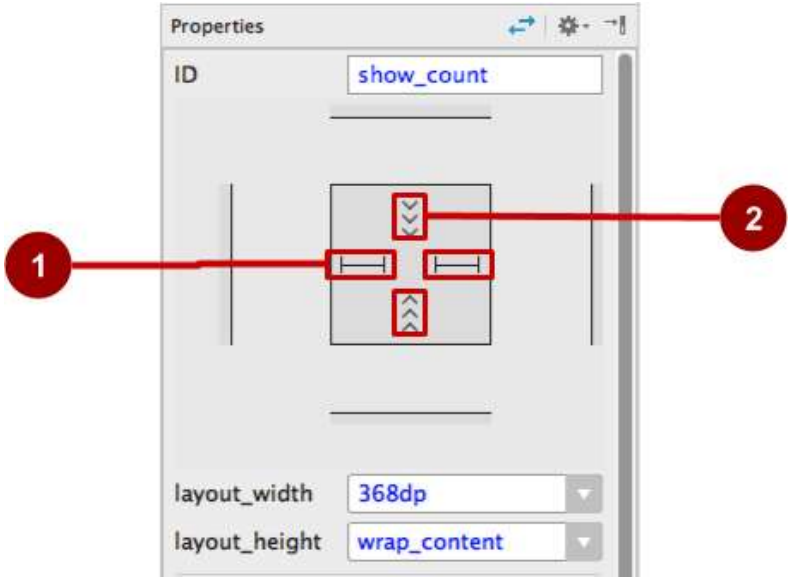
2 Con las restricciones eliminadas, ahora puede mover las vistas en el diseño libremente. Arrastre la vista `button_toast` hacia abajo a cualquier posición debajo de la vista `button_count`, de modo que la vista amarilla `show_count` esté en la parte superior, como se muestra en la figura a continuación.



2.6 Cambiar el tamaño de una vista

El editor de diseño ofrece controladores de cambio de tamaño en las cuatro esquinas de una vista para cambiar el tamaño de la vista rápidamente. Puede arrastrar los controladores en cada esquina de la vista para cambiar su tamaño, pero al hacerlo, está cableando las dimensiones de ancho y alto, lo que debe evitar para la mayoría de las vistas porque las dimensiones de vista cableadas no se pueden adaptar a diferentes tamaños de pantalla y contenido.

En su lugar, utilice el panel Properties en el lado derecho del editor de diseño para seleccionar un modo de tamaño que no utilice dimensiones cableadas. El panel Properties incluye un panel de tamaño cuadrado en la parte superior. Los símbolos dentro del cuadrado representan las configuraciones de altura y ancho de la siguiente manera:

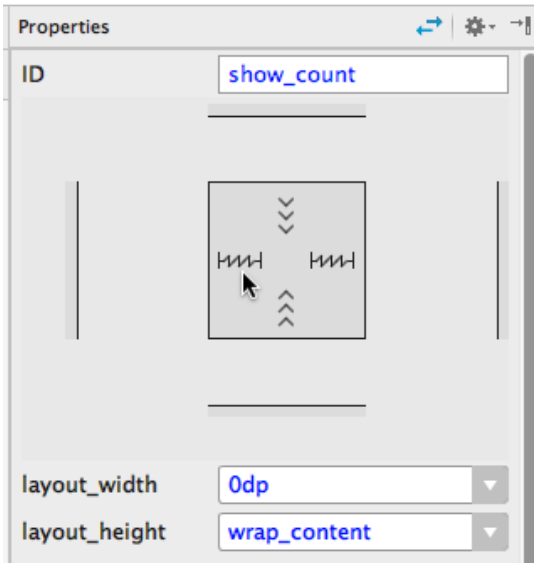


En la anterior figura:

1. **Horizontal view size control**. El control de tamaño horizontal, que aparece en dos segmentos en los lados izquierdo y derecho del cuadrado, especifica el `layout_width`. Las líneas rectas indican que la dimensión es fija y se establece en la propiedad `layout_width` debajo del cuadrado.
2. **Vertical view size control**. El control de tamaño vertical, que aparece en dos segmentos en los lados superior e inferior del cuadrado, especifica la propiedad `layout_height`. Los ángulos indican que este control de tamaño se establece en `wrap_content`, lo que significa que la vista se

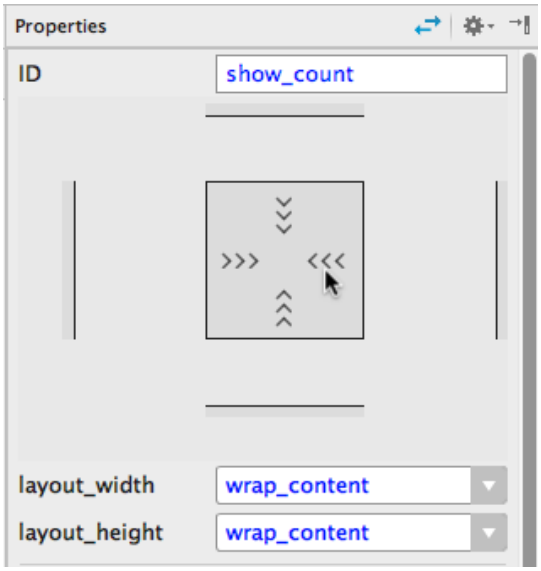
Siga los siguientes pasos para redimensionar la vista `show_count`:

1 Haga clic en la vista `show_count` en el panel Component Tree.
2 Haga clic en el control de tamaño de vista horizontal en el panel Properties. Las líneas rectas cambian a bobinas de resorte, como se muestra en la figura siguiente, que representa "cualquier tamaño". La propiedad `layout_width` se establece en cero porque no hay una dimensión establecida, pero la vista puede expandirse tanto como sea posible para cumplir con las restricciones y la configuración de margen.

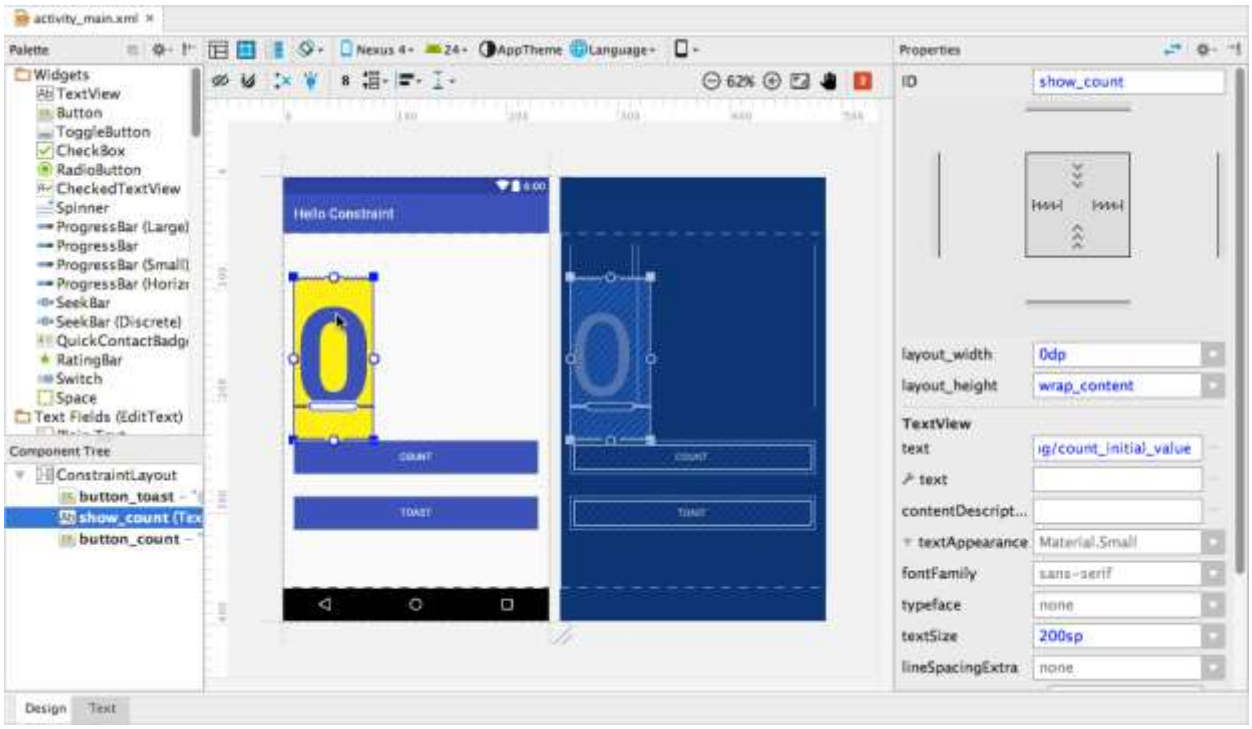


Utilizará esta configuración para anclar el tamaño de la vista a las restricciones, pero primero, continúe experimentando con la configuración.

1 Haga clic en el control de tamaño horizontal de la vista nuevamente (lado izquierdo o derecho), solo para ver qué otras opciones tiene. Las bobinas de resorte cambian a ángulos, como se muestra en la figura a continuación, lo que indica que el `layout_width` está establecido en `wrap_content`



2 Haga clic en el control de tamaño horizontal de la vista nuevamente, y volverá a las líneas rectas, lo que indica una dimensión fija. Haga clic de nuevo para que las líneas cambien a bobinas de resorte, como se muestra en la figura a continuación, que representa "cualquier tamaño"

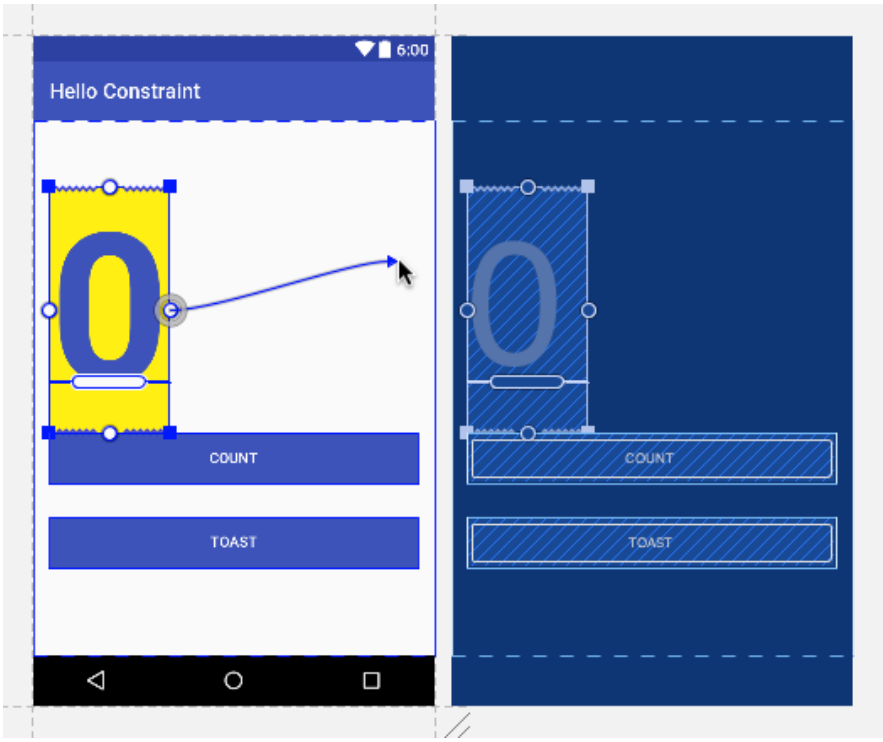


2.7 Agregando restricciones a las vistas

Agregará una restricción a la vista `show_count` para que se extienda hasta el borde derecho del diseño y otra restricción para que la vista se ubique justo debajo del borde superior del diseño. Como la vista se configuró en "cualquier tamaño" en el paso anterior, la vista se expandirá según sea necesario para que coincida con las restricciones.

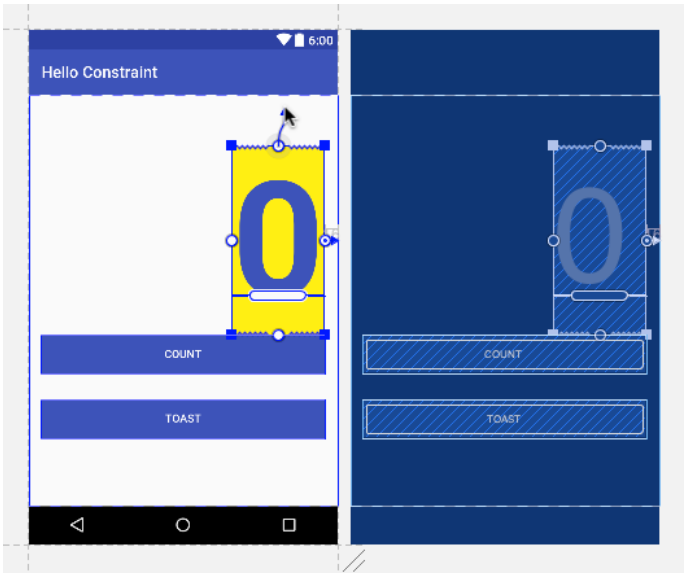
También moverá los dos botones a su posición en el lado izquierdo de la vista `show_count`, limitará el botón `button_toast` a los bordes superior e izquierdo del diseño y restringirá el botón `button_count` para que su línea base de texto coincida con la línea base de texto de la vista `show_count`.

1 Para crear una restricción del lado derecho para la vista `show_count`, haga clic en la vista en el diseño y luego desplace el cursor sobre la vista para ver sus controles de restricción. Haga clic y mantenga presionado el controlador de restricción en el lado derecho de la vista y arrastre la línea de restricción que aparece en el borde derecho del diseño, como se muestra en la figura siguiente.



A medida que suelta el click-and-hold, se crea la restricción y la vista `show_count` salta al borde derecho del diseño.

2 Haga clic y mantenga presionado el controlador de restricción en la parte superior de la vista y arrastre la línea de restricción que aparece hasta el borde superior del diseño debajo de la barra de la aplicación, como se muestra en la figura a continuación.

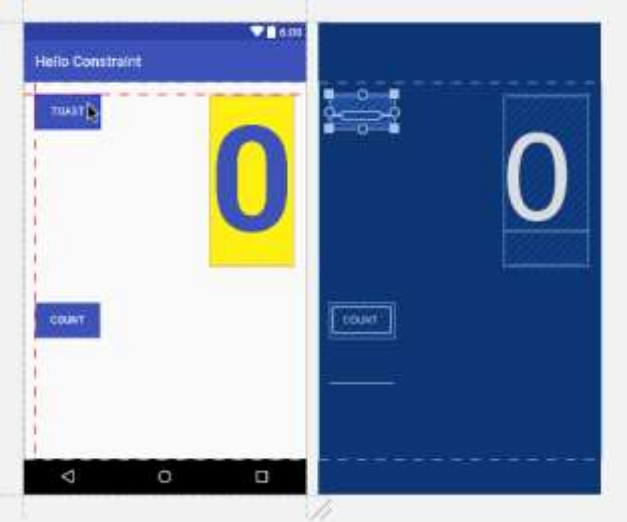


Esto restringe la vista al borde superior. Después de arrastrar la restricción, la vista `show_count` salta al borde superior derecho del diseño, porque está anclado a los bordes superior y derecho.

3 Haga clic en la vista `button_toast` y use el panel Properties como se mostró anteriormente para cambiar el tamaño de la vista a `wrap_content` tanto para `layout_width` y `layout_height`. También cambie el tamaño de la vista `button_count` a `wrap_content` para `layout_width` y `layout_height`.

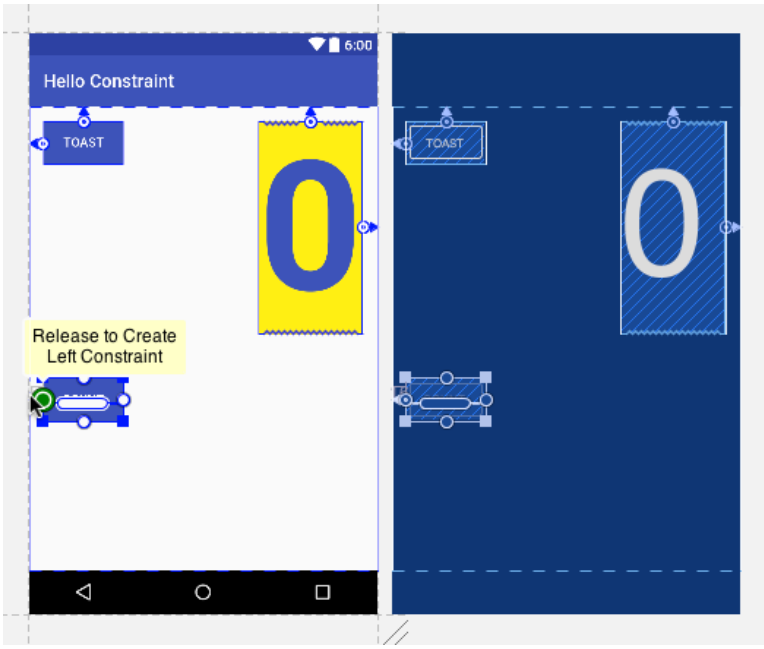
Siempre utilizar `wrap_content` para los botones, de modo que si el texto del botón está localizado en un idioma diferente, el botón aparecerá más ancho o más delgado para acomodar la palabra en el idioma diferente.

4 Arrastre la vista `button_toast` a su posición en el lado izquierdo de la vista `show_count` como se muestra en la figura a continuación. Las guías aparecen para que pueda alinear la vista en posición con los márgenes superior e izquierdo.

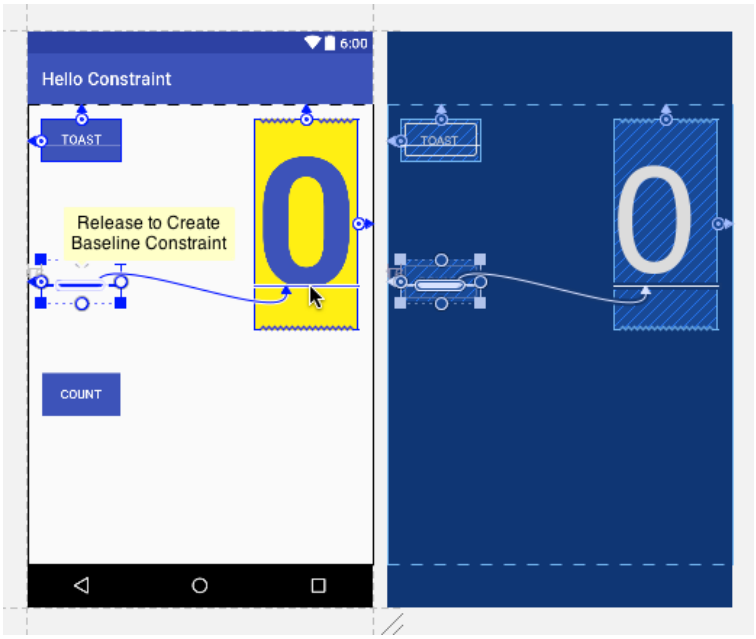


5 Seleccione la vista `button_toast` en el diseño, haga clic en el identificador de restricción que aparece en la parte superior de la vista y arrástrelo al borde superior del diseño debajo de la barra de la aplicación como se muestra en la figura a continuación. Luego, haga clic en el identificador de restricción que aparece en el lado izquierdo de la vista y arrástrelo al borde izquierdo del diseño.

6. Seleccione la vista `button_count`, haga clic en el identificador de restricción que aparece en el lado izquierdo de la vista y arrástrelo al borde izquierdo del diseño.



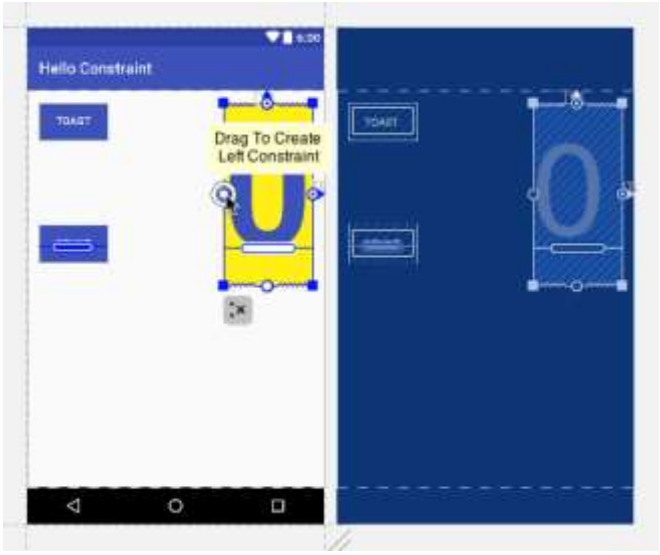
7. Para crear una restricción de línea de base entre la línea de base de texto de la vista `button_count` y la línea de base de texto de `show_countview`, seleccione la vista `button_count`, y luego desplace el cursor sobre el asa de la línea de base de la vista durante dos segundos hasta que la manija parpadee en blanco. A continuación, haga clic y arrastre la línea de restricción que aparece a la línea base de la vista `show_count`, como se muestra en la figura siguiente.



Ahora tiene un diseño en el que cada vista está configurada en dimensiones no específicas y restringidas al diseño. El texto de un botón está alineado con la línea de base de un TextView, de modo que si mueve el TextView, el botón se mueve con él.

Consejo: si una vista no tiene al menos *dos* restricciones, esta aparecerá en la parte superior del diseño.

8. Aunque la vista `show_count` ya tiene dos restricciones, puede agregarle otra restricción. Arrastre una línea de restricción desde el controlador de restricciones en el lado izquierdo de la vista al lado derecho de la vista `button_count`, como se muestra en las figuras a continuación.

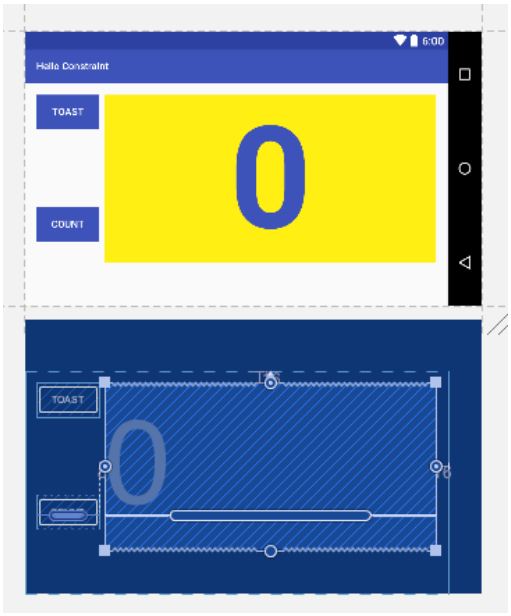


9. Ejecute la aplicación. El diseño se ajusta a sus restricciones.

Tarea 3: Crear variantes del diseño

Puede crear variantes de su diseño para orientación horizontal y pantallas más grandes. Creará una versión alternativa del diseño de la tarea anterior para optimizarla para la orientación horizontal:

1. Abra su archivo de diseño para la aplicación HelloConstraint, y asegúrese de estar viendo el editor de diseño (haga clic en la pestaña **Design** en la parte inferior de la ventana).
2. Haga clic en el icono **Layout Variant** en la segunda fila de iconos (consulte la figura en Tarea 2, paso 4) y elija **Create Landscape Variation**. La pestaña "land/activity_main.xml" aparecerá mostrando el diseño para la orientación lanscape (horizontal), como se muestra en la figura a continuación:



Puede cambiar el diseño de la versión landscape (horizontal) sin cambiar la orientación portrait original (vertical), aprovechando así la pantalla más amplia.

3. En Proyecto: vista de Android en el panel izquierdo del Android Studio, busque dentro del directorio **res > layout** y verá que el Android Studio creó automáticamente la variante, llamada **activity_main.xml (land)**.
4. La pestaña "land/activity_main.xml" aún debe estar abierta en el editor de diseño; de lo contrario, haga doble clic en el archivo **activity_main.xml** (land) en el directorio de **layout**.
5. Haga clic en la pestaña **Text** para ver el diseño en XML. Encuentre la restricción para la vista `button_toast` que alinea su borde superior con la vista principal:

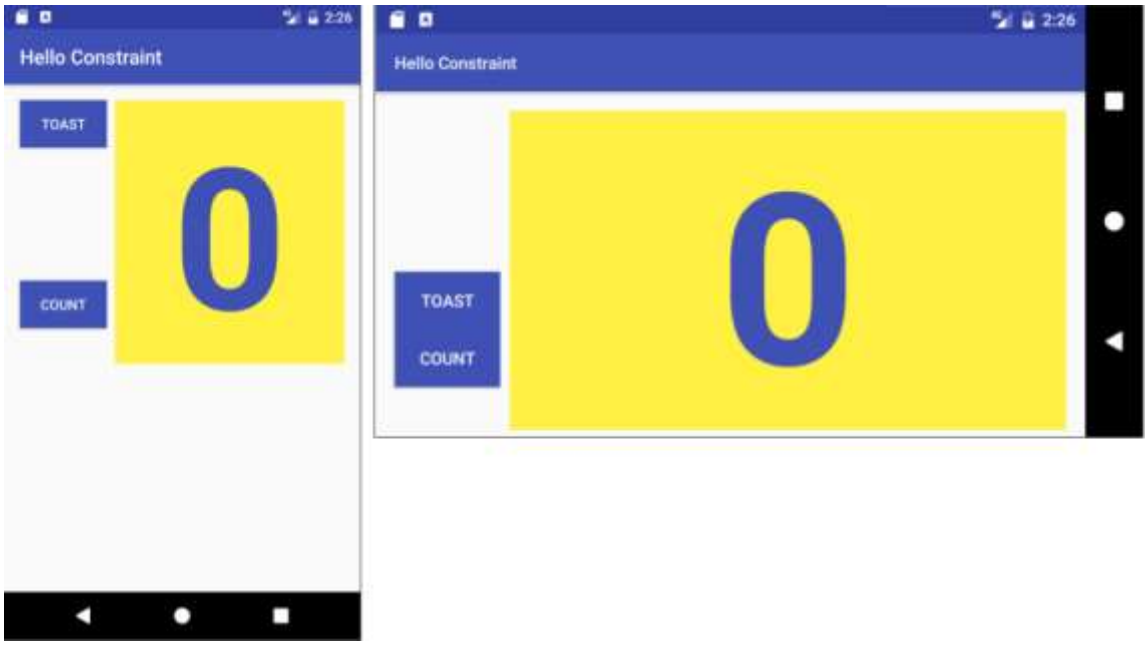
```
<Botón
android: id = "@ + id / button_toast"
...
aplicación: layout_constraintTop_toTopOf = "parent"
...
```

6. Cambie esta restricción para que el borde inferior de la vista `button_toast` esté alineado con el borde superior de la vista `button_count`.

Sugerencia: si la restricción para alinear la parte superior de una vista con otra vista es `app:layout_constraintTop_toTopOf`, ¿cuál cree que es la restricción para alinear la parte inferior de una vista con la parte superior de otra vista? Respuesta:

```
app:layout_constraintBottom_toTopOf = "@ id / button_count"
```

7. Ejecute la aplicación y cambie al modo horizontal para ver el diseño diferente. Los diseños deberían aparecer como se muestra a continuación.



2.3 Trabajando con Elementos TextView

La clase **TextView** es una subclase de la clase **View** que muestra texto en la pantalla. Se puede controlar cómo aparecerá el texto con los atributos TextView en el archivo XML de layout. Esta práctica muestra cómo trabajar con múltiples elementos TextView, incluido uno en el que el usuario puede desplazar su contenido verticalmente.

Si tiene más información de la que cabe en la pantalla del dispositivo, puede crear una vista de desplazamiento (*scrolling view*) para que el usuario pueda desplazarse verticalmente deslizando hacia arriba o hacia abajo, u horizontalmente deslizando hacia la derecha o hacia la izquierda.

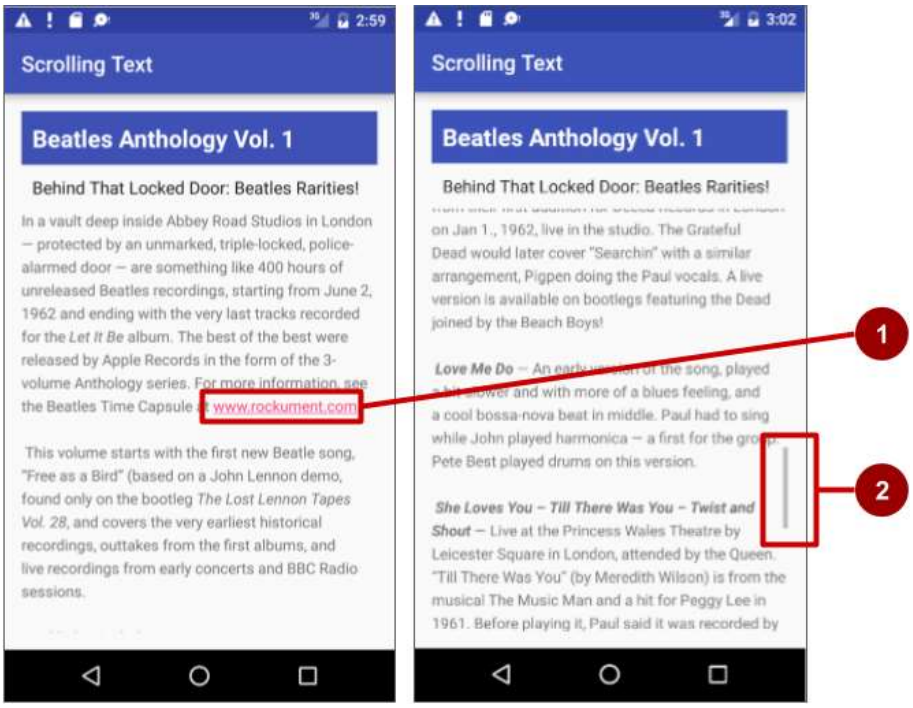
Normalmente, utilizará una vista de desplazamiento para presentar noticias, artículos o cualquier texto extenso que no se ajuste completamente a la pantalla del dispositivo. También puede usar una vista desplazable para permitir a los usuarios ingresar múltiples líneas de texto, o combinar elementos de la UI (como un campo de texto y un botón) dentro de una vista desplazable.

La clase **ScrollView** proporciona el diseño de la vista de desplazamiento. ScrollView es una subclase de **FrameLayout**, y los desarrolladores deben colocar solo una vista como elemento secundario dentro de ella, donde la vista secundaria contiene todo el contenido para desplazarse. Esta vista secundaria puede ser un grupo de visualización (como un administrador de diseño como LinearLayout) con una jerarquía compleja de objetos. Tenga en cuenta que los diseños complejos pueden sufrir problemas de rendimiento con vistas secundarias, como imágenes. Una buena opción para una vista dentro de un ScrollView es un LinearLayout que esté organizado en una orientación vertical, presentando elementos de nivel superior por los que el usuario puede desplazarse.

Con ScrollView, todas las vistas están en la memoria y en la jerarquía de vistas, incluso si no se muestran en la pantalla. Esto hace que ScrollView sea ideal para desplazar páginas de texto de forma libre sin problemas, porque el texto ya está en la memoria. Sin embargo, ScrollView puede consumir mucha memoria, lo que puede afectar el rendimiento del resto de la aplicación. Para mostrar una lista larga de elementos a los que los usuarios pueden agregar, eliminar o editar, considere usar RecyclerView, que se describe en una práctica separada.

Resumen de la aplicación

La aplicación Scrolling Text muestra el componente de UI ScrollView. ScrollView es un view group que en este ejemplo contiene un TextView. Está app muestra una página de texto bastante larga, en este caso, una revisión de un álbum de música, de manera que el usuario pueda desplazarse verticalmente para leer la revisión deslizando hacia arriba y hacia abajo. Una barra de desplazamiento aparece en el margen derecho. La aplicación muestra cómo puede usar texto formateado con etiquetas HTML mínimas para configurar texto en negrita o cursiva, y con caracteres de nueva línea para separar párrafos. También puede incluir enlaces web activos en el texto.

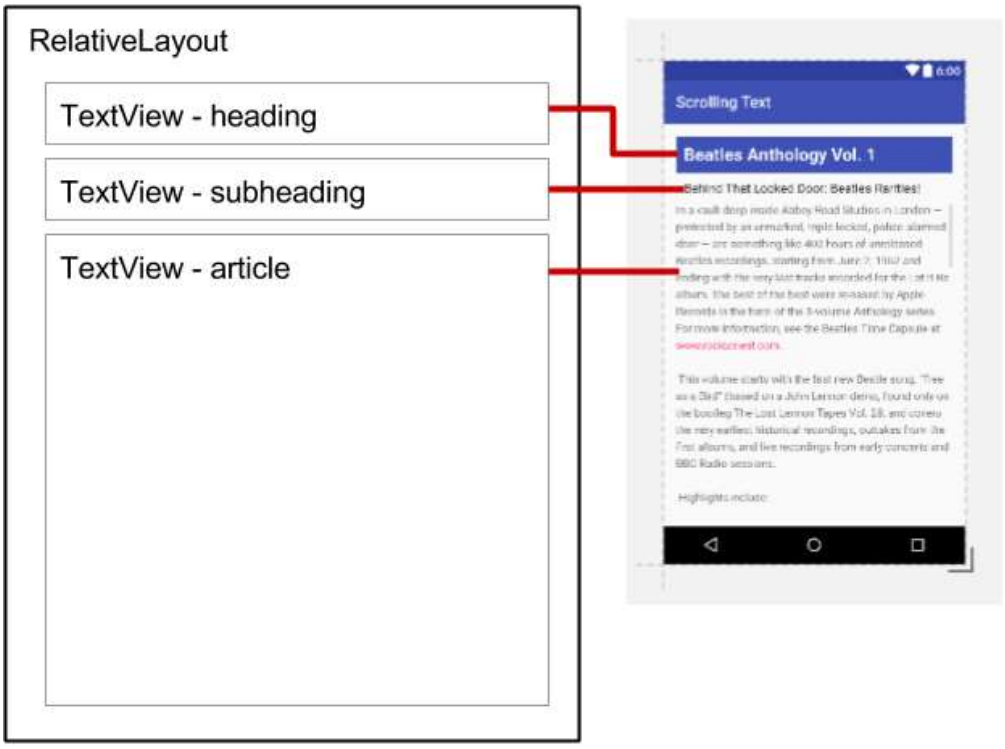


En la figura anterior, aparece lo siguiente:

1. Un enlace web activo incrustado en texto de forma libre
2. La barra de desplazamiento que aparece cuando se desplaza el texto

Tarea 1: Agregar varios text views

En esta práctica, creará un proyecto Android para la aplicación Scrolling Text, para ello agregará TextViews al diseño para el título del artículo y un subtítulo, y cambiará el elemento TextView "Hello World" existente para mostrar un artículo extenso. La siguiente figura es un diagrama del diseño.



Realizará todos estos cambios en el código XML y en el archivo strings.xml. Editará el código XML para el diseño en el panel Text, el cual se muestra al hacer clic en la pestaña **Text**, en lugar de hacer clic en la pestaña **Design** para el panel Diseño. Algunos cambios en los elementos y atributos de la UI son más fáciles de realizar directamente en el panel de texto utilizando el código fuente XML.

1.1 Crear el Proyecto y los elementos TextView

1. Crear un Proyecto Nuevo en el Android Studio con los siguientes parámetros:

Atributo	**Valor**
Application Name	Scrolling Text
Company Name	android.example.com (o el que usted decida)
Phone and Tablet Minimum SDK	API18: Android 4.3 Jelly Bean
Template	Empty Activity
Generate Layout File checkbox	Checked

2. En la carpeta **app > res > layout**, abra el archivo **activity_main.xml**, y haga clic en la pestaña **Text** tab para ver el código XML si este no ha sido seleccionado.

   En el tope, o *root*, de la vista de jerarquía está un view group tal como un ConstraintLayout. Cambie este view group a RelativeLayout. La segunda línea de código ahora luce como se muestra:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

RelativeLayout permite posicionar sus Views hijas relativas una a otra o relativas al propio padre RelativeLayout. El TextView por defecto “Hello Word” ha sido creado por la plantilla del Empty Layout el cual es una View hija dentro del view group RelativeLayout. Para más información acerca del uso del RelativeLayout, ver Relative Layout API Guide.

3. Agregue un elemento TextView encima del TextView “Hello World”. Una vez agregado configúrelo con los siguientes atributos:

Atributos del TextView #1	**Valor**
android:id	"@+id/article_heading"
layout_width	"match_parent"
layout_height	"wrap_content"
android:background	"@color/colorPrimary"
android:textColor	"@android:color/white"
android:padding	"10dp"
android:textAppearance	"@android:style/TextAppearance.Large"
android:textStyle	"bold"
android:text	"Article Title"

Consejo: Los atributos para darle estilo al texto y el fondo los puede encontrar en la documentación en la documentación de la claseTextView.

4. Extraer el recurso de string para el atributo de texto `android:text` “Article Title” en el TextView para crear una entrada para ello en **strings.xml**.

   Coloque el cursor sobre el string cableado, presione Alt-Enter (Opción-Intro en la Mac) y seleccione **Extract string resource**. Luego edite el nombre del recurso para el valor del string y coloque **article_title**.

   Consejo: Los recursos de string están descritos con detalle en String Resources documentation.

5. Extraer los recursos de dimensiones para el atributo `android:padding` que está cableado en “10dp” en el TextView para crear una entrada en **dimens.xml**.

   Coloque el cursor en el string cableado, presione Alt-Enter (Option-Enter en Mac), y seleccione **Extract dimension resource**. Luego edite el Resource name y coloque **padding_regular**.

6. Agregue otro elemento TextView sobre el TextView “Hello World” y por debajo del TextView recientemente creado. Agregue los siguientes atributos a este Nuevo TextView:

Atributos TextView #2	Valor
android:id	"@+id/article_subheading"
layout_width	"match_parent"
layout_height	"wrap_content"
android:layout_below	"@id/article_heading"
android:padding	"@dimen/padding_regular"
android:textAppearance	"@android:style/TextAppearance"
android:text	"Article Subtitle"

7. Note que debido a que ya extrayó el recurso de dimension para el string "`10dp`" en **paddin_regular** del previo TextView creado, puede utilizar "`@dimen/padding_regular`" para el atributo `android:padding` del TextView.

8. Extraiga el recurso de string para el string cableado de los atributos del `android:text` "Article Subtitle" en el TextView como **article_subtitle**.

9. En el elemento TextView "Hello World", remueva los atributos `layout_costraint`, si están presentes.

10. Agregue los siguientes atributos de TextView al element TextView "Hello World", y cambie el atributo `android:text`:

Atributos TextView	Valor
android:id	"@+id/article"
android:lineSpacingExtra	"5sp"
android:layout_below	"@id/article_subheading"
android:text	Change to "Article text"

11. Extraiga el recurso de string para "Article text" en **article_text** y extraiga el recurso de dimensión para "`5sp`" en **line_spacing**.

12. Reformatee y alinee el código seleccionando **Code > Reformat Code**. Es una buena práctica reformatear y alinear su código para que sea más fácil de entender para usted y para otros

1.2 Agregar el texto al artículo

En una aplicación real que accede a artículos de revistas o periódicos, los artículos que aparecen probablemente provienen de una fuente en línea a través de un content provider o están previamente guardados en una base de datos en el dispositivo.

Para esta práctica, creará el artículo como un único string largo en el recurso **strings.xml**.

1. En la carpeta **app > res > values**, abra **strings.xml**.

2. Ingrese los valores para los strings `article_title` y `article_subtitle` con un título inventado y un subtítulo para el artículo que está agregando. Los valores de string para cada uno deben ser texto de una sola línea sin etiquetas HTML ni líneas múltiples.

3. Ingresar o copiar y peguar el texto para el string `article_text`. **Sugerencia**: de la página https://es.lipsum.com/ puede copiar un texto con la cantidad de caracteres que desee.

   Use el texto suministrado por el string `article_text` en el archivo **strings.xml** con su propio texto genérico. Puede copiar y pegar la misma oración una y otra vez, siempre que el resultado sea una larga sección de texto que no se ajuste por completo a la pantalla. Tenga en cuenta lo siguiente (consulte la figura a continuación para ver un ejemplo):

   a. Al ingresar o pegar texto en el archivo **strings.xml**, las líneas de texto no se ajustan a la siguiente línea, sino que se extienden más allá del margen derecho. Este es el comportamiento correcto: cada nueva línea de texto que comienza en el margen izquierdo representa un párrafo completo.

   b. Ingrese **\n** para representar un fin de línea, y otro **\n** para representar una línea en blanco.

   ¿Por qué? Necesita agregar caracteres “end-of-line” para mantener los párrafos se fundan entre sí.

   Consejo: Si desea ver el texto completo en strings.xml, puede presionar Return para ingresar los finales de la línea, o formatear primero el texto en un editor de texto con terminaciones de línea.

   c. Si tiene un apóstrofe (') en su texto, debe “escaparlo” precediéndolo con una barra diagonal inversa (\'). Si tiene una comilla doble en su texto, también debe “escaparla” (\ "). También debe “escapar” cualquier otro carácter que no sea ASCII. Consulte la sección "Formatting and Styling" de String Resources para obtener más detalles.

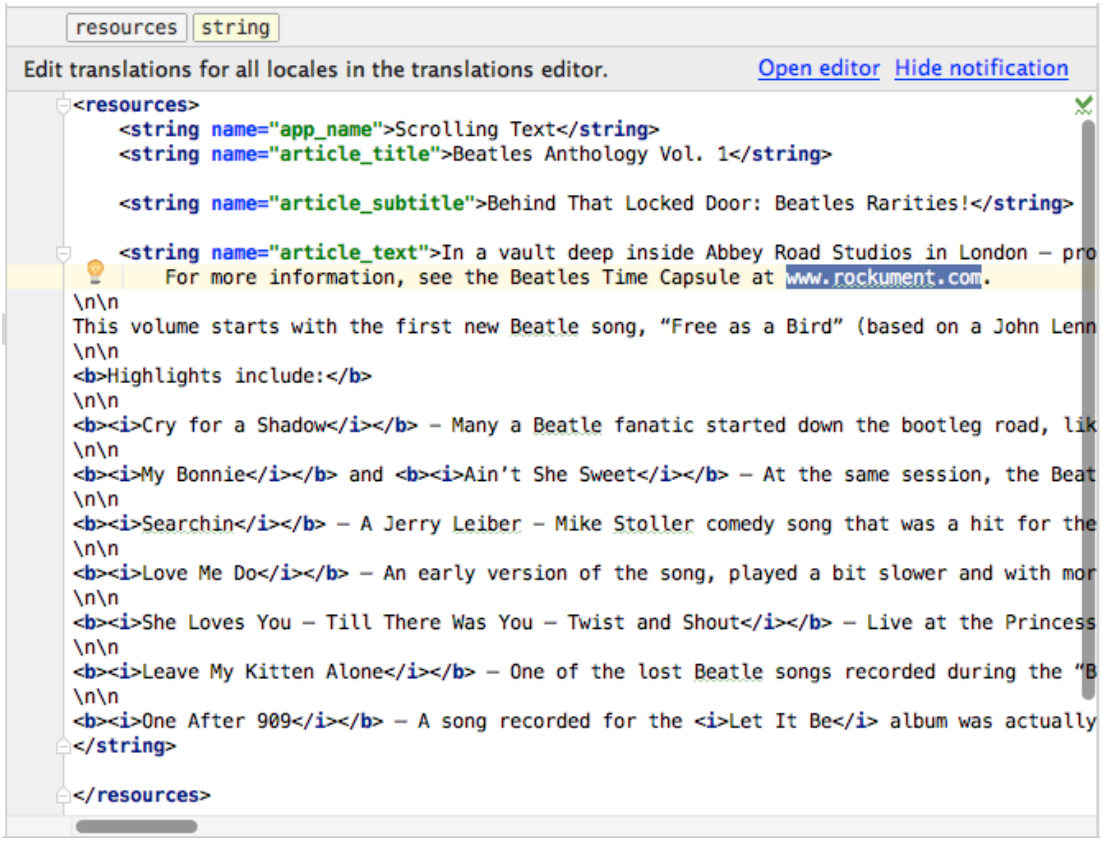
   d. Ingrese las etiqueta HTML **** y **** alrededor de las palabras que deben aparecer en negrillas.

   e. Ingrese las etiqueta HTML **** y **** alrededor de las palabras que deben aparecer en *itálicas*. Note, sin embargo, que si usa apóstrofes curvos en una frase itálica, los puede remplazar por apóstrofes rectos.

   f. Puede combiner negrillas e itálicas simplemnte combinando las etiquetas, tal como **** palabras… **<\i><\b>**. Otras etiquetas HTML serán ignoradas.

   g. Adjunte todo el texto dentro de `<string name="article_text"> </string>` en el archivo strings.xml.

   h. Incluya un enlace web para prueba, tal como **www.google.com** (el ejemplo mostrado usa www.rockument.com). *No use* ninguna otra etiqueta HTML except la de negrillas e itálicas ya que serán ignoradas y serán presentadas como texto, lo cual no es lo que se desea.



4. Ejecute la aplicación.

   Aparece el artículo, e incluso puede desplazarlo, pero el desplazamiento no es uniforme y no hay barra de desplazamiento porque aún no ha incluido un ScrollView (lo que hará en la próxima tarea). Tenga en cuenta también que tocar un enlace web no hace nada actualmente. También eso será arreglado en la siguiente tarea.

Dependiendo de la versión de Android Studio, el archivo de diseño activity_main.xml lucirá parecido al presentado a continuación:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.android.scrollingtext.MainActivity">

    <TextView
        android:id="@+id/article_heading"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
        android:background="@color/colorPrimary"
        android:textColor="@android:color/holo_orange_light"
        android:textColorHighlight="@color/colorAccent"
        android:padding="10dp"
        android:textAppearance="@android:style/TextAppearance.Large"
        android:textStyle="bold"
        android:text="@string/article_title"/>

    <TextView
        android:id="@+id/article_subheading"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/article_heading"
        android:padding="10dp"
        android:textAppearance="@android:style/TextAppearance"
        android:text="@string/article_subtitle"/>

    <TextView
        android:id="@+id/article"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/article_subheading"
        android:lineSpacingExtra="5sp"
        android:text="@string/article_text"/>

</RelativeLayout>
```

Tarea 2: Agregar enlaces Web activos y un ScrollView

En la tarea anterior, creó la aplicación de Scrolling Text con TextViews para un título del artículo, subtítulo y el texto extenso del artículo. También incluyó un enlace web, pero el enlace aún no está activo. Ahora agregará el código para que esté activo.

Además, el TextView por sí solo no puede permitir a los usuarios desplazarse por el texto del artículo para verlo todo. Agregará un nuevo grupo de vista llamado ScrollView al diseño XML que hará que TextView se pueda desplazar.

2.1 Add the autoLink attribute for active web links

Agregar el atributo `android:autoLink="web"` al TextView del artículo. El código XML para este TextView lucirá ahora así:

```
<TextView
   android:id="@+id/article"
   ...
   android:autoLink="web"
   ... />
```

2.2 Add a ScrollView to the layout

Para hacer que una vista (tal como un TextView) sea desplazable, se debe incrustar tal vista *dentro* de un ScrollView.

1. Agregue un ScrollView entre el TextView article_subheading y el TextView `article`. Al ingresar **<ScrollView**, Android Studio agrega automáticamente `</ScrollView>` al final y presenta los atributos `android:layout_width` y `android:layout_height` con sugerencias. Elija **wrap_content** de las sugerencias para ambos atributos. El código ahora debería verse así:

```
<TextView
   android:id="@+id/article_subheading"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:layout_below="@id/article_heading"
   android:padding="10dp"
   android:textAppearance="@android:style/TextAppearance"
   android:text="@string/article_subtitle"/>

<ScrollView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_below="@id/article_subheading"></ScrollView>
<TextView
   android:id="@+id/article"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_below="@id/article_subheading"
   android:lineSpacingExtra="5sp"
   android:autoLink="web"
   android:text="@string/article_text"/>
```

2. Mueva el código de finalización `</ScrollView>` *después* del TextView `article` para que los atributos TextView del artículo estén dentro del elemento XML del ScrollView.

3. Elimine el siguiente atributo del TextView `article`, porque el ScrollView se colocará debajo del elemento article_subheading, y este atributo para TextView entraría en conflicto con el ScrollView:

   ```
   android:layout_below="@id/article_subheading"
   ```

   El diseño debería lucir de la manera siguiente:

4. Seleccione **Code > Reformat Code** para reformatear el código XML de manera que el TextView `article` aparezca ahora sangrado dentro dentro del código `<Scrollview`.

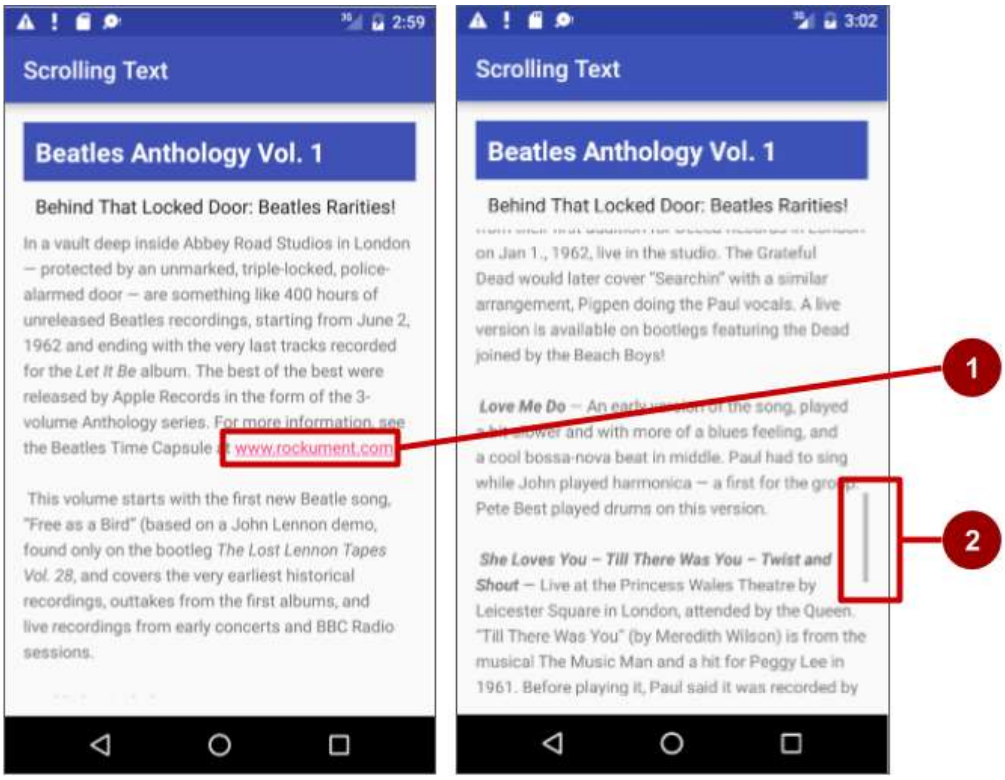
5. Ejecute la aplicación.

Deslize el dedo hacia arriba y hacia abajo para desplazarte por el artículo. La barra de desplazamiento aparece en el margen derecho mientras se desplaza.

Toque el enlace web para ir a la página web. El atributo `android:autoLink` convierte cualquier URL reconocible en el TextView (tal como www.rockument.com) en un enlace web.

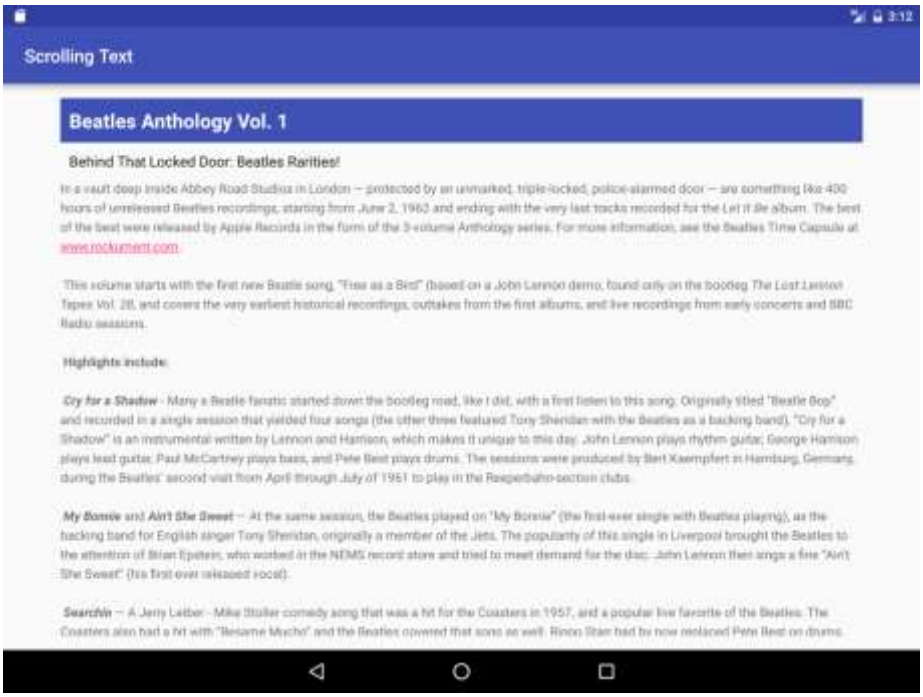
6. Rote su dispositivo o emulador mientras ejecuta la aplicación. Observe cómo la vista de desplazamiento se amplía para usar la pantalla completa y aún así se desplaza correctamente.

7. Ejecute la aplicación en una talet o emulador de tablet. Observe cómo la vista de desplazamiento se amplía para usar la pantalla completa y aún así se desplaza correctamente.



En la figura anterior, lo siguiente aparecerá:

1. Un enlace web activo dentro del texto mostrado.
2. La barra de desplazamiento que aparece cuando se desplaza por el texto.



Dependiendo de la versión de Android Studio, el archivo de diseño activity_main.xml lucirá algo parecido a como se muestra a continuación:

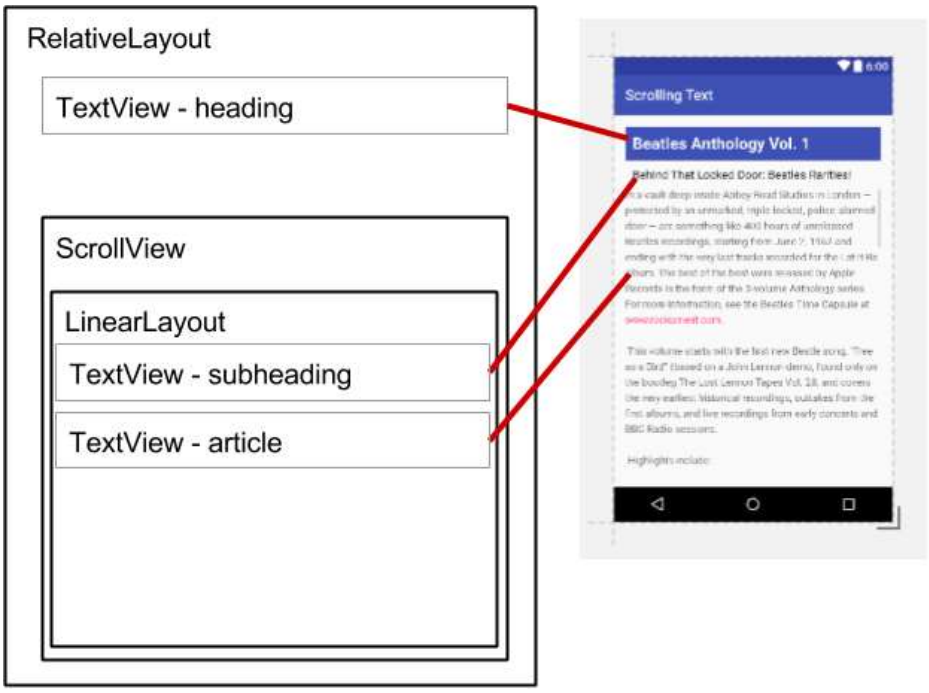
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.android.scrollingtext.MainActivity">

    <TextView
        android:id="@+id/article_heading"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/colorPrimary"
        android:textColor="@android:color/white"
        android:paddingTop="10dp"
        android:paddingBottom="10dp"
        android:paddingLeft="10dp"
        android:paddingRight="10dp"
        android:textAppearance="@android:style/TextAppearance.Large"
        android:textStyle="bold"
        android:text="@string/article_title"/>

    <TextView
        android:id="@+id/article_subheading"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/article_heading"
        android:paddingTop="10dp"
        android:paddingBottom="10dp"
        android:paddingLeft="10dp"
        android:paddingRight="10dp"
        android:textAppearance="@android:style/TextAppearance"
        android:text="@string/article_subtitle"/>

    <ScrollView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/article_subheading">

        <TextView
            android:id="@+id/article"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:lineSpacingExtra="5sp"
            android:autoLink="web"
            android:text="@string/article_text"/>
```

```
    </ScrollView>
</RelativeLayout>
```

Tarea 3: Desplazar elementos múltiples

Como se indicó anteriormente, el grupo de vistas ScrollView puede contener solo una vista secundaria (como el artículo TextView creado); sin embargo, esa Vista puede ser otro grupo de vistas que contenga Vistas, como LinearLayout. Puede anidar un grupo de vistas como LinearLayout dentro del grupo de vistas ScrollView, desplazando así todo lo que está dentro de LinearLayout.

Por ejemplo, si desea que el subtítulo del artículo se desplace junto con el artículo, agregue un LinearLayout dentro del ScrollView y mueva el subtítulo, junto con el artículo, al LinearLayout. El grupo de vistas LinearLayout se convierte en la View única hija en el ScrollView, como se muestra en la figura siguiente, y el usuario puede desplazar todo el grupo de vistas: el subtítulo y el artículo.



3.1 Add a LinearLayout to the ScrollView

1. En su computadora, haga una copia de la carpeta del proyecto ScrollingText del Android Studio, y renombrelo como **ScrollingText2**. Para copiar y renombrar un proyecto, siga las instrucciones de “Copiar y renombrar un proyecto” del Apéndice.

2. Abra en Android Studio el proyecto **ScrollingText2**, y abra el archivo **activity_main.xml** para cambiar el código XML de diseño.

3. Agregue un LinearLayout sobre el TextView `artículo` en el ScrollView. Mientras ingrsa **<LinearLayout**, Android Studio agrega automáticamente `</LinearLayout>` al final y

presenta los atributos `android:layout_width` y `android:layout_height` con sugerencias. Elija **match_parent** y **wrap_content** de las sugerencias para su ancho y alto, respectivamente. El código ahora debería verse así:

```
<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="wrap_content"></LinearLayout>
```

Se utiliza `match_parent` para que coincida con el ancho del grupo de vista principal y `wrap_content` para que el grupo de vista sea lo suficientemente grande como para incluir su contenido y relleno (padding) .

4. Mueva el código de finalización `</LinearLayout>` después del TextView `article` pero antes del cierre del `</ScrollView>` para que el LinearLayout incluya el TextView `article` y se encuentre completamente dentro de ScrollView.

5. Agregue el atributo `android:orientation="vertical"` al LinearLayout para establecer la orientación de LinearLayout en vertical. El LinearLayout dentro del ScrollView ahora debería verse así (elija **Code > Reformat Code** para sangrar los grupos de vista correctamente):

```
<ScrollView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_below="@id/article_subheading">

   <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="vertical">

      <TextView
         android:id="@+id/article"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:autoLink="web"
         android:lineSpacingExtra="5sp"
         android:text="@string/article_text" />

   </LinearLayout>
</ScrollView>
```

6. Mueva el TextView `article_subheading` a una posición dentro del LinearLayout sobre el TextView article.

7. Elimine el atributo `android:layout_below="@id/article_heading"` del TextView `article_subheading`. Como este TextView se encuentra ahora dentro del LinearLayout, este atributo entraría en conflicto con los atributos del LinearLayout.

8. Cambie el atributo de diseño del ScrollView `android:layout_below="@id/article_subheading"` a `android:layout_below="@id/article_heading"`. Ahora que el subtítulo es parte del LinearLayout, ScrollView debe colocarse debajo del encabezado, no en el subtítulo.

9. Ejecute la aplicación.

   Deslice el dedo hacia arriba y hacia abajo para desplazarse por el artículo, y ahora note que el subtítulo también se desliza en conjunto con el artículo, mientras el encabezado permanece en su mismo lugar.

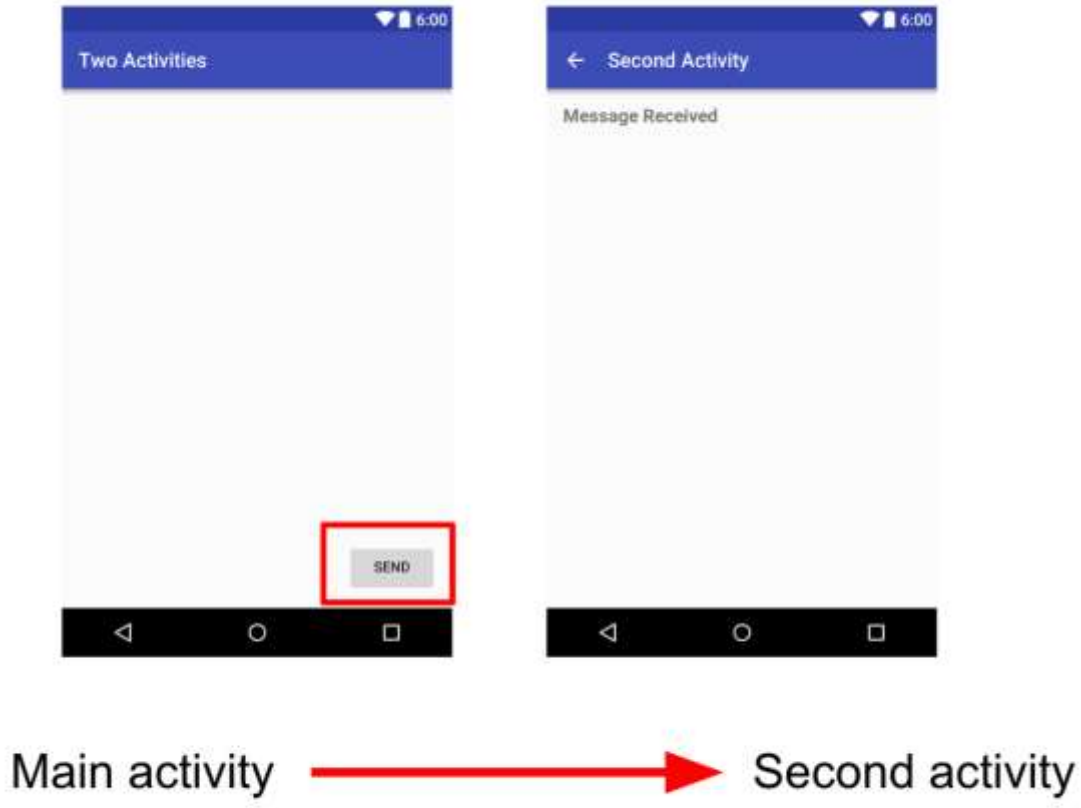
Capítulo 3

3.1 Creando e Iniciando Activities

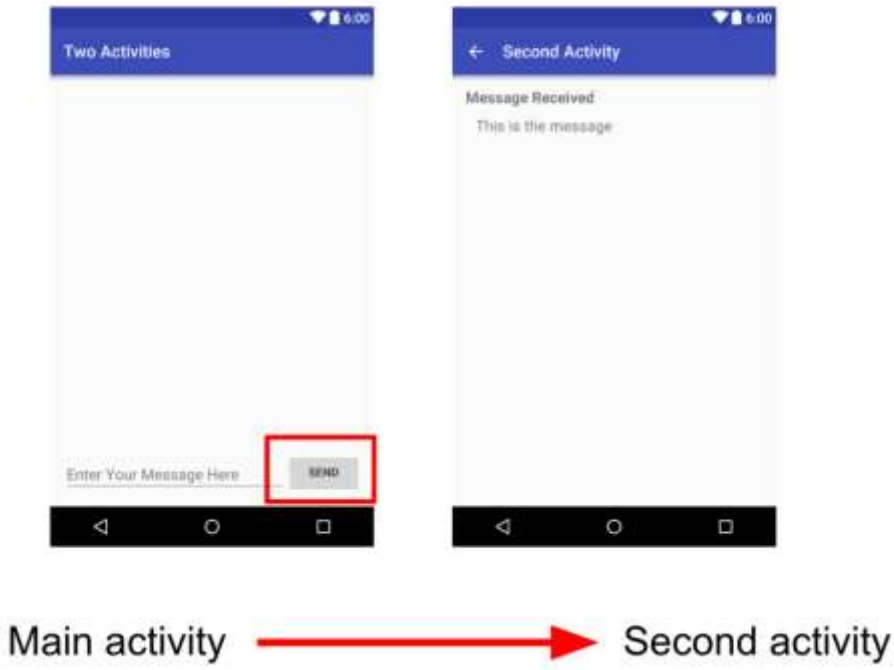
Two Activites

En este capítulo, creará una aplicación llamada TwoActivities que, como era de esperar, contiene dos actividades. Esta aplicación se construirá en tres etapas.

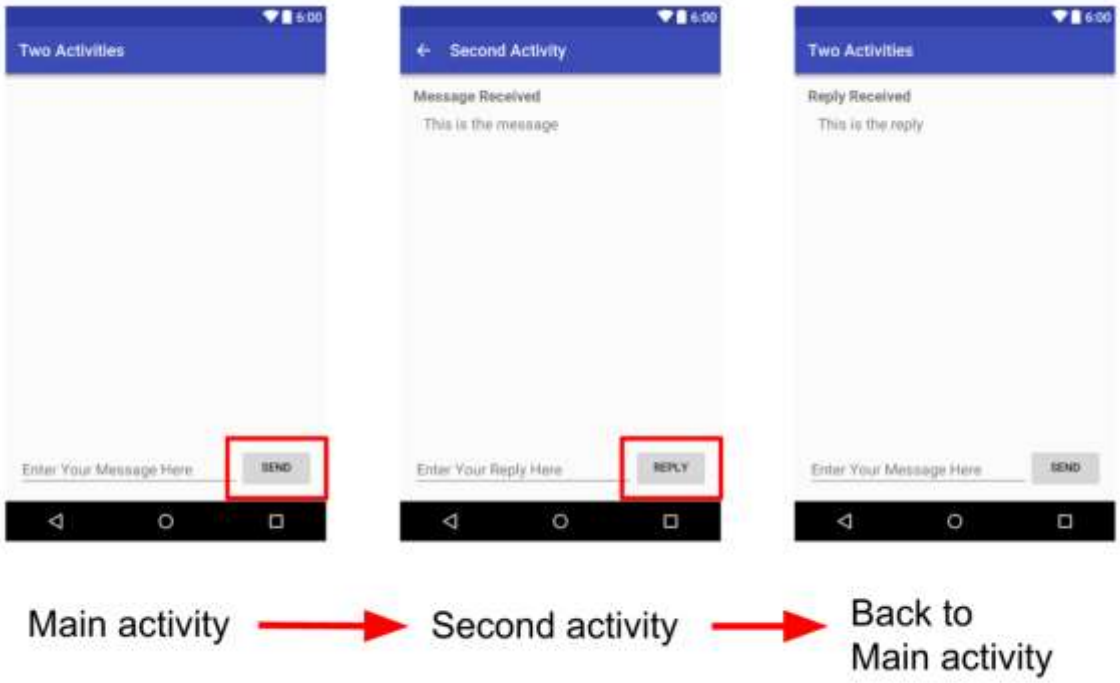
En la primera etapa, se va a crear una aplicación cuya actividad principal contiene solo un botón (Send). Cuando el usuario hace clic en este botón, la actividad principal tiene un intent para iniciar la segunda actividad.



En la segunda etapa, agregará una vista EditText a la actividad principal. El usuario ingresa un mensaje y hace clic en Send. La actividad principal utiliza el intent para iniciar la segunda actividad y enviar el mensaje del usuario a esa actividad. La segunda actividad muestra el mensaje que recibió.



En la etapa final de la aplicación TwoActivities, se va a agregar una vista EditText y un botón Replay a la segunda actividad. El usuario ahora puede escribir un mensaje de respuesta y hacer clic en Replay, y esta respuesta se mostrará en la actividad principal. En este punto, se usa un intent para pasar el mensaje de respuesta de la segunda actividad a la actividad principal.



Tarea 1. Crear el Proyecto TwoActivities

En esta tarea, configurará el proyecto inicial con una actividad principal, definirá el diseño y definirá un método de esqueleto para el evento del botón onClick.

1.1 Crear el proyecto TwoActivities

1. Inicie Android Studio y cree un nuevo proyecto de Android Studio.
2. Llame a su aplicación "TwoActivities" y cambie el dominio de la compañía a "android.example.com". Elija el SDK mínimo que utilizó en los proyectos anteriores.
3. Elija **Empty Activity** para la plantilla del proyecto. Haga clic en **Next**.
4. Acepte el nombre de actividad predeterminado (MainActivity). Asegúrese de que el cuadro **Generate Layout file** esté marcado. Haga clic en **Finish**.

1.2 Definir el diseño (layout) para la actividad principal

1. Abra `res/layout/activity_main.xml` en el Editor de diseño, haga clic en la pestaña Text en la parte inferior de la pantalla y cambie el grupo de vista raíz a RelativeLayout, como lo hizo en ejercicios anteriores.
2. Haga clic en la pestaña Design en la parte inferior de la pantalla y elimine el TextView que dice "Hello World".
3. Agregue un Button al diseño en cualquier posición.
4. Cambie al Editor XML (haga clic en la pestaña Text) y modifique estos atributos del Button:

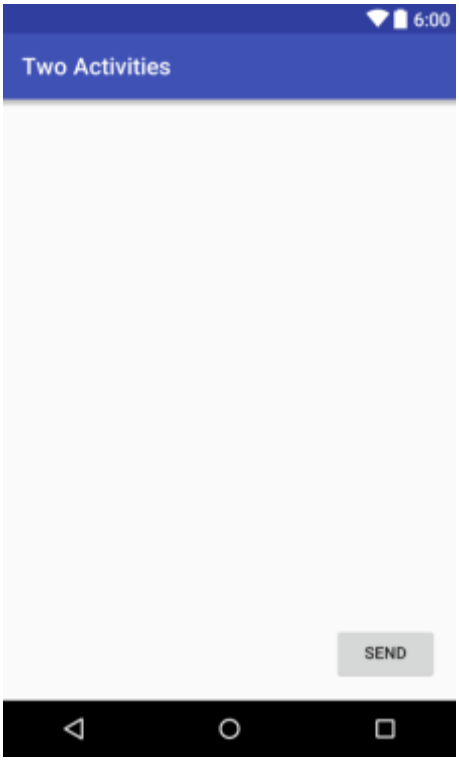
Atributo	**Valor**
android:id	"@+id/button_main"
android:layout_width	wrap_content
android:layout_height	wrap_content
android:layout_alignParentRight	"true"
android:layout_alignParentBottom	"true"
android:layout_alignParentEnd	"true"
android:text	"Send"
android:onClick	"launchSecondActivity"

Esto puede generar un error que " Method launchSecondActivity is missing in MainActivity." Por favor ignora este error por ahora. Se tratará en la siguiente tarea.

5. Coloque el cursor sobre la palabra "`Send`".
6. Presione **Alt-Enter** (**Opción-Enter** en Mac) y seleccione **Extract string resources**.
7. Cambie el **Resource name** en `button_main` y haga clic en **OK**.

   Esto crea un recurso de string en el archivo values/res/string.xml, y el string en su código se reemplaza por una referencia a ese recurso de string.

8. Elija **Code>Reformat Code** para formatear el código XML, si fuese necesario.
9. Obtenga una vista previa del diseño de la actividad principal utilizando el Editor de diseño (Layout Editor). El diseño debería verse así:



Código de la Solución: Dependiendo de su version del Android Studio, el código lucirá parecido como se muestra a continuación:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context="com.example.android.twoactivities.MainActivity">
    <Button
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/button_main"
       android:id="@+id/button_main"
       android:layout_alignParentBottom="true"
       android:layout_alignParentRight="true"
       android:layout_alignParentEnd="true"
       android:onClick="launchSecondActivity"/>
</RelativeLayout>
```

1.3 Definir la acción del botón

En esta tarea, implementará el método onClick que definió en el diseño.

1. En el Editor XML, coloque el cursor sobre la palabra "`launchSecondActivity`".
2. Presione **Alt-Enter** (**Opción-Enter** en Mac) y seleccione **Create 'launchSecondActivity(View)' in 'MainActivity**.

   Se abren los archivos de MainActivity.java y el Android Studio genera la plantilla de un método para el controlador onClick.

3. Dentro de `launchSecondActivity`, agregue una declaración de log que diga “Button clicked!"

   ```
   Log.d (LOG_TAG, "Button clicked!");
   ```

   LOG_TAG se mostrará en rojo. Las definiciones para esa variable se agregarán en un paso posterior.

4. Coloque el cursor sobre la palabra "Log" y presione **Alt-Enter** (**Option-Enter** en Mac). El Android Studio agrega una declaración de importación para `android.util.Log`.
5. En la parte superior de la clase, agregue una constante para la variable LOG_TAG:

   ```
   private static final String LOG_TAG = MainActivity.class.getSimpleName();
   ```

   Esta constante usa el nombre de la clase como la etiqueta.

6. Ejecute su aplicación. Al hacer clic en el botón "Send", verá el mensaje "Botón presionado" en el Monitor de Android (logcat). Si hay demasiados resultados en el monitor, escriba MainActivity en la caja de búsqueda y el registro solo mostrará líneas que coincidan con esa etiqueta.

Código de la solución:

```
package com.example.android.twoactivities;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;

public class MainActivity extends AppCompatActivity {
   private static final String LOG_TAG = MainActivity.class.getSimpleName();

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
    }

    public void launchSecondActivity(View view) {
        Log.d(LOG_TAG, "Button clicked!");
    }
}
```

Tarea 2. Crear e iniciar la segunda actividad

Cada actividad nueva que agregue al proyecto tiene su propio diseño y archivos Java, separados de los de la actividad principal. También tienen sus propios elementos de `<actividad>` en el manifiesto de Android. Al igual que con la actividad principal, las nuevas actividades que cree en el Android Studio también son extensiones de la clase AppCompatActivity.

Todas las actividades de la aplicación están desconectadas entre sí. Sin embargo, puede definir una actividad como padre de otra actividad en el archivo AndroidManifest.xml. Esta relación padre-hijo permite a Android agregar sugerencias de navegación, como “flechas hacia la izquierda” en la barra de título de cada actividad.

Las actividades se comunican entre sí (tanto en la misma aplicación como en diferentes aplicaciones) usando los *intents*. Hay dos tipos de intenciones, explícitas e implícitas. Un intent explícito es aquel en el que se conoce el objetivo de ese intent, es decir, ya se conoce el nombre completo de la clase de esa actividad específica. Un intent implícito es aquel en la que no tiene el nombre del componente de destino, pero tiene una acción general que va realizar. Aprenderás acerca de los intents implícitos en una práctica más adelante.

En esta tarea, agregará una segunda actividad a la aplicación, con su propio diseño. Debe Modifica el manifest de Android para definir la actividad principal como el padre de la segunda actividad. Luego, modificará el método de evento `onClick` en la actividad principal para incluir un intent que inicie la segunda actividad cuando haga clic en el botón.

2.1 Crear la segunda actividad

1. Haga clic en la carpeta **app** del proyecto y elija **File > New > Activity > Empty Activity**.
2. Cambie el nombre de la nueva actividad como "SecondActivity". Asegúrese de que esté activado **Generate Layout File** y que el nombre del diseño se complete como `activity_second`.
3. Haga clic en **Finish**. El Android Studio agregará un nuevo diseño de actividad (`activity_second`) y un nuevo archivo Java (SecondActivity) al proyecto para la nueva actividad. También actualizará el manifest de Android para incluir esta nueva actividad.

2.2 Modificar el manifest de Android

1. Abrir el archivo `manifests/AndroidManifest.xml`.
2. Buscar el elemento `<activity>` que el Android Studio creó para la segunda actividad.

```
<activity android:name=".SecondActivity"> </activity>
```

3. Agregar los siguientes atributos al elemento `<activity>`:

Atributo	Valor
android:label	"Second Activity"
android:parentActivityName	".MainActivity"

El atributo `label` agrega el título de la actividad a la barra de acciones.

El atributo `parentActivityName` indica que la actividad principal es el padre de la segunda actividad. Esta relación de la actividad principal se utiliza para la navegación "de avance" dentro de la aplicación. Al definir este atributo, la barra de acciones para la segunda actividad aparecerá con una flecha hacia la izquierda para permitir al usuario navegar "en avance" hacia la actividad principal.

4. Coloque el cursor en " Second Activity" y presione **Alt-Enter** (**Opción-Enter** en la Mac).
5. Elija **Extract string resource**, nombre al recurso como activity2_name " y haga clic en **OK**. Android Studio agregará un recurso de string para la etiqueta de la actividad.
6. Agregue un elemento `<meta-data>` dentro del elemento `<activity>` para la segunda actividad. Use estos atributos:

Atributo	Valor
android:name	"android.support.PARENT_ACTIVITY"
android:value	"com.example.android.twoactivities.MainActivity"

El elemento `<meta-data>`. En este caso, estos atributos logran lo mismo que el atributo `android:parentActivityName`: definen una relación entre dos actividades con el objetivo de navegar en avance. Estos atributos son necesarios para las versiones anteriores de Android. `android:parentActivityName` solo está disponible para niveles API 16 y superiores.

Código de la solución:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.android.twoactivities">

   <application
       android:allowBackup="true"
       android:icon="@mipmap/ic_launcher"
       android:label="@string/app_name"
       android:supportsRtl="true"
       android:theme="@style/AppTheme">
```

```
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".SecondActivity"
            android:label="@string/activity2_name"
            android:parentActivityName=".MainActivity">
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.example.android.twoactivities.MainActivity" />
        </activity>
    </application>
</manifest>
```

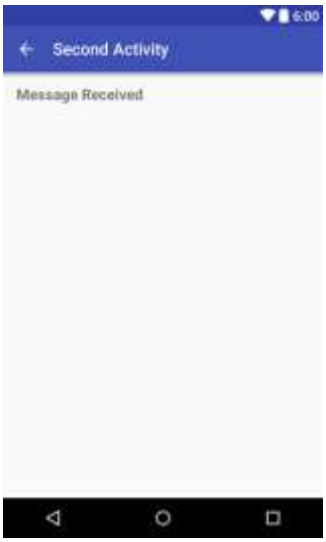
2.3 Definir el diseño (layout) de la segunda actividad

1. Abra el archivo `res/layout/activity_second.xml` y cambie la raíz del view group a RelativeLayout, como lo hizo en ejercicios anteriores.
2. Agregue un TextView ("Plain TextView" en el Editor de diseño). Agregue estos atributos al TextView:

Atributo	**Valor**
android:id	"@+id/text_header"
android:layout_width	wrap_content
android:layout_height	wrap_content
android:layout_marginBottom	"@dimen/activity_vertical_margin"
android:text	"Message Received"
android:textAppearance	"?android:attr/textAppearanceMedium"
android:textStyle	"bold"

El valor de textAppearance es un atributo especial de un tema de Android que define los estilos de fuente básicos para fuentes pequeñas, medianas y grandes. Aprenderá más sobre los temas en una práctica más adelante.

3. Extraiga el string " Message Received" en un recurso llamado `text_header`.
4. Obtenga una vista previa del diseño en el Editor de diseño. El diseño debería verse así:



Código de la solución: Dependiendo de su version del Android Studio, el código lucirá parecido como se muestra a continuación:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context=".SecondActivity">

   <TextView
       android:id="@+id/text_header"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_marginBottom="@dimen/activity_vertical_margin"
       android:text="@string/text_header"
       android:textAppearance="?android:attr/textAppearanceMedium"
       android:textStyle="bold" />
</RelativeLayout>
```

2.4 Agregar un intent a la actividad principal

En esta tarea, agregará un intento explícito a la actividad principal. Esta intención se usa para activar la segunda actividad cuando se hace clic en el botón Enviar.

1. Abra el archivo Java de la `MainActivity` (`java/com.example.android.twoactivities/MainActivity`).
2. Cree un nuevo intent en el método `launchSecondActivity()`.

El constructor de la clase Intent toma dos argumentos para un intent explícito: un objeto **Context** de la aplicación y el componente específico que recibirá ese intent. Aquí debe usar `this` como el context, y `SecondActivity.class` como la clase específica.

```
Intent intent = new Intent (this, SecondActivity.class);
```

3. Coloque el cursor en el Intent y presione **Alt-Enter** (**Opción-Enter** en Mac) para agregar una importación para la clase Intent.
4. Llame al método `startActivity()` con el nuevo intent como argumento.

```
startActivity (intent);
```

5. Ejecute la aplicación.

   Cuando hace clic en el botón Send, la actividad principal envía el intent y el sistema Android inicia la segunda actividad. Esa segunda actividad aparece en la pantalla. Para volver a la actividad principal, haga clic en el botón Atrás de Android en la parte inferior izquierda de la pantalla, o puede usar la flecha izquierda en la parte superior de la segunda actividad.

Tarea 3. Enviar datos desde la actividad principal a la segunda actividad

En la última tarea, agregó un intent explícito a la actividad principal lo que activó la segunda actividad. También puede usar intents para enviar datos de una actividad a otra.

En esta tarea, modificará el intent explícito en la actividad principal para incluir datos adicionales (en este caso, un string ingresado por el usuario) en los extras del intent. Luego, modificará la segunda actividad para recuperar los datos de los extras del intent y mostrarlos en pantalla.

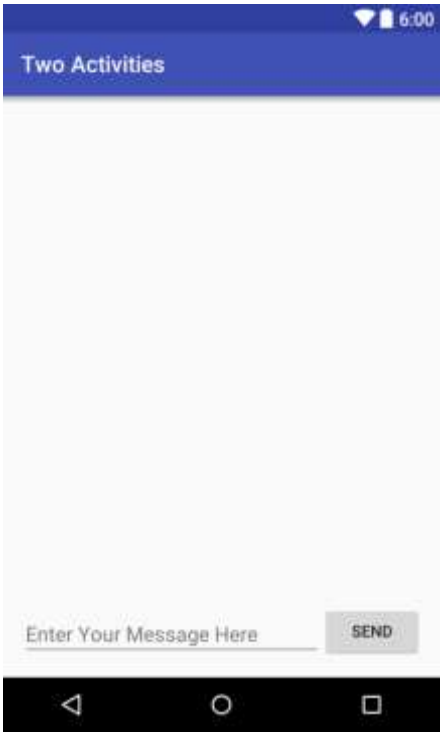
3.1 Agregar un EditText al diseño de la actividad principal

1. Abra res/layout/activity_main.xml.
2. Agregue una vista EditText (Plain Text en el Editor de diseño). Otorgue a EditText estos atributos:

Atributo	**Valor**
android:id	"@+id/editText_main"
android:layout_width	match_parent
android:layout_height	wrap_content
android:layout_toLeftOf	"@+id/button_main"
android:layout_toStartOf	"@+id/button_main"
android:layout_alignParentBottom	"true"
android:hint	"Enter Your Message Here"

3. Elimine el atributo `Android:text`.
4. Extraiga el string "Enter Your Message Here" en un recurso llamado `editText_main`.

El Nuevo diseño de la actividad principal lucirá así:



Código de la solución: Dependiendo de su version del Android Studio, el código lucirá parecido como se muestra a continuación:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context="com.example.android.twoactivities.MainActivity">

   <Button
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/button_main"
       android:id="@+id/button_main"
       android:layout_alignParentBottom="true"
       android:layout_alignParentRight="true"
       android:layout_alignParentEnd="true"
       android:onClick="launchSecondActivity"/>
```

```
    <EditText
        android:id="@+id/editText_main"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_toLeftOf="@+id/button_main"
        android:layout_toStartOf="@+id/button_main"
        android:hint="@string/editText_main" />
</RelativeLayout>
```

3.2 Agregue un string a los extras del intent de la actividad principal

El objeto intent puede pasar datos a la actividad de destino de dos maneras: en el campo de data, o en los extras del intent. La data del intent es un URI que indica los datos específicos sobre los que se actuará. Si la información que desea pasar a una actividad a través de un intent no es una URI, o si tiene más de una pieza de información que desea enviar, puede poner tal información adicional en los *extras* del intent.

Los *extras* del intent son pares clave/valor en un paquete (Bundle). Un bundle es una colección de datos, almacenados como pares clave/valor. Para pasar información de una actividad a otra, se colocan claves y valores en el paquete extra del intent de la actividad de envío, y luego los vuelven a sacar en la actividad de recepción.

1. Abra `java/com.example.android.twoactivities/MainActivity`.
2. Agregue una constante pública en la parte superior de la clase para definir la clave para el extra del intent:

```
public static final String EXTRA_MESSAGE =
                        "com.example.android.twoactivities.extra.MESSAGE";
```

3. Agregue una variable privada en la parte superior de la clase para definir el objeto EditText. Importe la clase EditText.

```
private EditText mMessageEditText;
```

4. En el método onCreate (), use findViewByID para obtener una referencia a la instancia de EditText y asígnesela a la variable privada:

```
mMessageEditText = (EditText)findViewById(R.id.editText_main);
```

5. En el método launchSecondActivity (), justo debajo del nuevo intent, obtenga el texto de EditText como un string:

```
String message = mMessageEditText.getText().ToString();
```

6. Agregue ese string al intent como un extra usando la constante EXTRA_MESSAGE como la clave y el string como el valor:

```
intent.putExtra(EXTRA_MESSAGE, message);
```

Código de la solución:

```
package com.example.android.twoactivities;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {
   private static final String LOG_TAG = MainActivity.class.getSimpleName();

   public static final String EXTRA_MESSAGE =
                           "com.example.android.twoactivities.extra.MESSAGE";

   private EditText mMessageEditText;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);

       mMessageEditText = (EditText) findViewById(R.id.editText_main);
   }

   public void launchSecondActivity(View view) {
       Log.d(LOG_TAG, "Button clicked!");

       Intent intent = new Intent(this, SecondActivity.class);
       String message = mMessageEditText.getText().toString();

       intent.putExtra(EXTRA_MESSAGE, message);
       startActivity(intent);
   }
}
```

3.3 Agregar un TextView para el mensaje de la segunda activity

1. Abrir el archivo `res/layout/activity_second.xml`.
2. Agregar un Segundo TextView. Darle a este TextView los siguientes atributos:

Atributo	Valor
android:id	"@+id/text_message"
android:layout_width	wrap_content
android:layout_height	wrap_content
android:layout_below	"@+id/text_header"
android:layout_marginLeft	"@dimen/activity_horizontal_margin"
android:layout_marginStart	"@dimen/activity_horizontal_margin"
android:textAppearance	"?android:attr/textAppearanceMedium"

1. Borrar el atributo `android:text` (si existe)
2. El nuevo diseño de la segunda actividad luce semejante al que se hizo en la tarea anterior, debido a que el nuevo TextView no contiene (por ahora) ningún texto y por lo tanto no aparecerá en la pantalla.

Código de la solución: Dependiendo de su version del Android Studio, el código lucirá parecido como se muestra a continuación:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context="com.example.android.twoactivities.SecondActivity">

   <TextView
       android:id="@+id/text_header"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/text_header"
       android:layout_marginBottom="@dimen/activity_vertical_margin"
       android:textAppearance="?android:attr/textAppearanceMedium"
       android:textStyle="bold"/>

   <TextView
       android:id="@+id/text_message"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_below="@+id/text_header"
       android:layout_marginLeft="@dimen/activity_horizontal_margin"
       android:layout_marginStart="@dimen/activity_horizontal_margin"
       android:textAppearance="?android:attr/textAppearanceMedium" />
```

```
</RelativeLayout>
```

3.4 Modificar la segunda actividad para obtener los extras y mostrar el mensaje

1. Abra `java/com.example.android.twoactivities/SecondActivity`.
2. En el método `onCreate()`, obtenga el intent que activó esta actividad:

```
Intent intent = getIntent();
```

3. Obtenga el string que contiene el mensaje de los extras del intent utilizando la variable estática `MainActivity.EXTRA_MESSAGE` como la clave:

```
String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
```

4. Use el método `findViewByID` para obtener una referencia del TextView para el mensaje del diseño (puede que necesite importar la clase TextView):

```
TextView textView = (TextView)findViewById(R.id.text_message);
```

5. Establezca el texto de ese TextView con el string del extra del intent:

```
textView.setText(mensaje);
```

6. Ejecute la aplicación. Cuando escribe un mensaje en la actividad principal y hace clic en Send, se inicia la segunda actividad y muestra ese mensaje.

Código de la solución:

```
package com.example.android.twoactivities;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

public class SecondActivity extends AppCompatActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_second);

       Intent intent = getIntent();
       String message =
               intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
```

```
        TextView textView = (TextView) findViewById(R.id.text_messag
        textView.setText(message);
    }
}
```

Tarea 4. Devolviendo datos a la main activity

Ahora que tiene una aplicación que inicia una nueva actividad y le envía datos, el paso final es devolver los datos de la segunda actividad a la actividad principal. También usará intents y extras de intent para esta tarea.

4.1 Agregar un EditText y un Button al diseño de la segunda activity

1. Copie el EditText y el Button del archivo de diseño de la actividad principal y péguelos en el segundo diseño.
2. En el archivo activity_second.xml, modifique los valores de los atributos para las vistas Button y EditText. Use estos valores:

Antiguo atributo (Button)	**Nuevo atributo (Button)**
android:id="@+id/button_main"	android:id="@+id/button_second"
android:onClick= "launchSecondActivity"	android:onClick="returnReply"
android:text= "@string/button_main"	android:text= "@string/button_second"
Antiguo atributo (EditText)	**Nuevo atributo (EditText)**
android:id="@+id/editText_main"	android:id="@+id/editText_second"
android:layout_toLeftOf= "@+id/button_main"	android:layout_toLeftOf= "@+id/button_second"
android:layout_toStartOf= "@+id/button_main"	android:layout_toStartOf= "@+id/button_second"
android:hint= "@string/editText_main"	android:hint= "@string/editText_second"

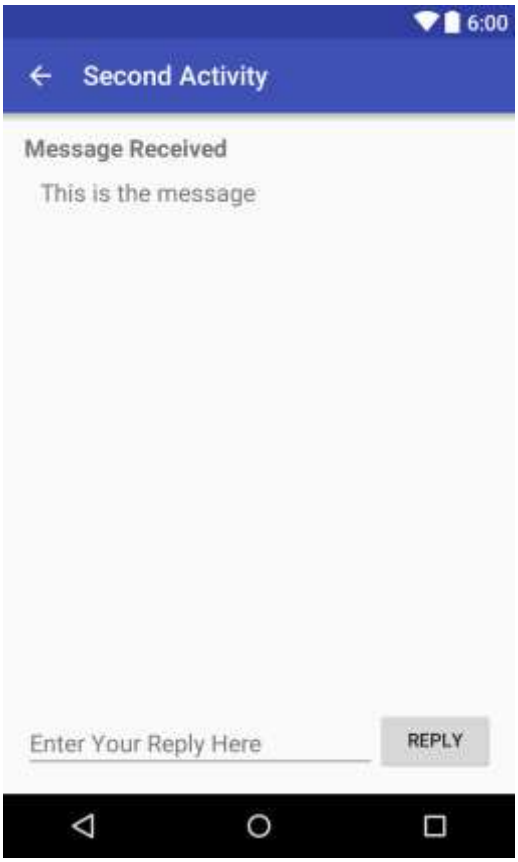
3. Abra el archivo `res/values/strings.xml` y agregue los recursos de string para el texto del botón y la ayuda del EditText

```
<string name="button_second">Reply</string>
<string name="editText_second">Enter Your Reply Here</string>
```

4. En el editor de diseño XML, coloque el cursor en "`returnReply`", presione **Alt-Enter** (**Option-Enter** en la Mac) y seleccione **Create 'returnReply(View)' in 'SecondActivity'**.

   El archivo SecondActivity.java se abre y Android Studio genera una plantilla de método para el controlador onClick. Ahora usted Implementará este método en la siguiente tarea.

   El nuevo diseño para la segunda actividad se debe ver así:



Código de la solución: Dependiendo de su version del Android Studio, el código lucirá parecido como se muestra a continuación:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context="com.example.android.twoactivities.SecondActivity">

   <TextView
       android:id="@+id/text_header"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/text_header"
       android:layout_marginBottom="@dimen/activity_vertical_margin"
       android:textAppearance="?android:attr/textAppearanceMedium"
       android:textStyle="bold"/>

   <TextView
       android:id="@+id/text_message"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/text_header"
        android:layout_marginLeft="@dimen/activity_horizontal_margin"
        android:layout_marginStart="@dimen/activity_horizontal_margin"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_second"
        android:id="@+id/button_second"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:onClick="returnReply"/>

    <EditText
        android:id="@+id/editText_second"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_toLeftOf="@+id/button_second"
        android:layout_toStartOf="@+id/button_second"
        android:hint="@string/editText_second" />

</RelativeLayout>
```

4.2 Crear un intent de respuesta en la segunda activity

1. Abra el archivo `java/com.example.android.twoactivities/SecondActivity`.
2. En la parte superior de la clase, agregue una constante pública para definir la clave para el extra del intent:

```
public static final String EXTRA_REPLY =
    "com.example.android.twoactivities.extra.REPLY";
```

3. Agregue una variable privada en la parte superior de la clase la cual contenga al objeto EditText

```
private EditText mReply;
```

4. En el método onCreate(), use findViewByID() para obtener una referencia a la instancia de EditText y asígnela a esa variable privada:

```
mReply = (EditText) findViewById(R.id.editText_second);
```

5. En el método `returnReply()`, obtenga el texto de EditText como un string:

```
String reply = mReply.getText().toString();
```

6. Cree un nuevo intent para la respuesta

Nota: No reuse el objeto intent que recibió del requerimiento original. Cree un nuevo intent para la respuesta.

```
Intent replyIntent = new Intent();
```

7. Agregue el string de respuesta de EditText al nuevo intent como un extra de intent. Como los extras son pares clave/valor, aquí la clave es EXTRA_REPLY y el valor es la respuesta:

```
replyIntent.putExtra(EXTRA_REPLY, reply);
```

8. Configure el resultado como RESULT_OK para indicar que la respuesta fue exitosa. Los códigos de resultado (incluyendo RESULT_OK y RESULT_CANCELLED) están definidos por la clase Activity.

```
setResult(RESULT_OK,replyIntent);
```

9. Llame al método finish() para cerrar la actividad y retornar a la actividad principal.

```
finish();
```

Código de la solución:

```
package com.example.android.twoactivities;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

public class SecondActivity extends AppCompatActivity {
   public static final String EXTRA_REPLY =
           "com.example.android.twoactivities.extra.REPLY";

   private EditText mReply;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_second);
```

```
        mReply = (EditText) findViewById(R.id.editText_second);

        Intent intent = getIntent();
        String message =
                intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
        TextView textView = (TextView) findViewById(R.id.text_message);
        textView.setText(message);
    }

    public void returnReply(View view) {
        String reply = mReply.getText().toString();

        Intent replyIntent = new Intent();
        replyIntent.putExtra(EXTRA_REPLY, reply);
        setResult(RESULT_OK, replyIntent);
        finish();
    }
}
```

4.3 Agregar TextViews a la main activity para mostrar la respuesta

La actividad principal necesita una forma de mostrar la respuesta enviada desde la segunda actividad. En esta tarea, agregará TextViews al diseño de la actividad principal para mostrar esa respuesta. Para hacerlo más fácil, copiará los TextViews que usó en la segunda actividad.

1. Copie los dos TextViews para la visualización de mensajes del archivo de diseño de la segunda actividad y péguelos en el diseño principal sobre las vistas existentes de EditText y Button.
2. Modifique los valores de los atributo para los nuevos TextViews. Use estos valores:

Antiguo atributo (TextView del header)	**Nuevo atributo (TextView del header)**
android:id="@+id/text_header"	android:id="@+id/text_header_reply"
android:text="@string/text_header"	android:text= "@string/text_header_reply"
Antiguo atributo (TextView del mensaje)	**Nuevo atributo (TextView del mensaje)**
android:id="@+id/text_message"	android:id="@+id/text_message_reply"
android:layout_below= "@+id/text_header"	android:layout_below= "@+id/text_header_reply"

3. Agregue el atributo `android:visibility` a cada una de las TextViews para que sean inicialmente invisibles. (Tenerlos visibles en la pantalla, pero sin ningún contenido, puede ser confuso para el usuario). Hará que estos TextViews sean visibles después de que los datos de respuesta pasen de la segunda actividad.

```
android:visibility="invisible"
```

4. Abra el archivo `res/values/strings.xml` y agregue un recurso de string para el encabezado de la respuesta:

```
<string name="text_header_reply">Reply Received</string>
```

El diseño de la actividad principal tiene el mismo aspecto que en la tarea anterior, aunque ha agregado dos nuevos TextViews al diseño. Sin embargo, dado que configura TextViews como invisible, no aparecen en la pantalla.

Código de la solución: dependiendo de su versión de Android Studio, su código tendrá el siguiente aspecto.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context="com.example.android.twoactivities.MainActivity">
   <TextView
       android:id="@+id/text_header_reply"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/text_header_reply"
       android:visibility="invisible"
       android:layout_marginBottom="@dimen/activity_vertical_margin"
       android:textAppearance="?android:attr/textAppearanceMedium"
       android:textStyle="bold"/>

   <TextView
       android:id="@+id/text_message_reply"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_below="@+id/text_header_reply"
       android:visibility="invisible"
       android:layout_marginLeft="@dimen/activity_horizontal_margin"
       android:layout_marginStart="@dimen/activity_horizontal_margin"
       android:textAppearance="?android:attr/textAppearanceMedium" />

   <Button
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/button_main"
       android:id="@+id/button_main"
       android:layout_alignParentBottom="true"
       android:layout_alignParentRight="true"
       android:layout_alignParentEnd="true"
       android:onClick="launchSecondActivity"/>

   <EditText
```

```
        android:id="@+id/editText_main"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_toLeftOf="@+id/button_main"
        android:layout_toStartOf="@+id/button_main"
        android:hint="@string/editText_main" />
</RelativeLayout>
```

4.4 Obtener la respuesta del extra del intent y mostrarlo

Cuando se utiliza un intent explícito para comenzar otra actividad, es posible que no esperes recuperar ninguna información; solo estás activando esa actividad. En ese caso, se usa startActivity() para comenzar la nueva actividad, como lo hizo anteriormente en esta lección. Sin embargo, si desea recuperar datos de la activity activada, deberá iniciarlos con startActivityFromResult().

En esta tarea, modificará la aplicación para iniciar la segunda actividad y esperará un resultado, extraer los datos devueltos del intent y mostrar esos datos en los TextViews que creó en la última tarea.

1. Abra el archivo `java/com.example.android.twoactivities/MainActivity`.
2. Agregue una constante pública en la parte superior de la clase para definir la clave para en tipo particular de respuesta que le interese:

```
public static final int TEXT_REQUEST = 1;
```

3. Agregue dos variables privadas: una para contener el encabezado y el TextView de la respuesta:

```
private TextView mReplyHeadTextView;
private TextView mReplyTextView;
```

4. En el método onCreate(), use findViewByID() para obtener referencias desde el diseño al encabezado y al TextView de la respuesta. Asigna esas instancias de vista a las variables privadas:

```
mReplyHeadTextView = (TextView) findViewById(R.id.text_header_reply);
mReplyTextView = (TextView) findViewById(R.id.text_message_reply);
```

5. En el método `launchSecondActivity()`, modifique la llamada al `startActivity()` para que sea `startActivityForResult()`, e incluya la clave `TEXT_REQUEST` como argumento:

```
startActivityForResult(intent, TEXT_REQUEST);
```

6. Cree el método de callback `onActivityResult()` con esta firma:

```
public void onActivityResult(int requestCode, int resultCode,Intent data) {}
```

7. Dentro del onActivityResult(), haga una llamada a `super.onActivityResult()`:

```
super.onActivityResult(requestCode, resultCode, data);
```

8. Agregue el código para probar tanto el `TEXT_REQUEST` (para procesar el resultado correcto del intent, en caso de que hubiesen multiples requerimientos) y el `RESULT_CODE` (para asegurarse que el requerimiento fue exitoso):

```
if (requestCode == TEXT_REQUEST) {
    if (resultCode == RESULT_OK) {
    }
}
```

9. Dentro del bloque if interno, obtenga el extra del intent de la respuesta (`data`). Aqui la clave para el extra es la constante EXTRA_REPLY del SecondActivity:

```
String reply = data.getStringExtra(SecondActivity.EXTRA_REPLY);
```

10. Coloque la visibilidad del header de respuesta a true:

```
mReplyHeadTextView.setVisibility(View.VISIBLE);
```

11. Coloque el texto del TextView de respuesta al valor de `reply`, y hágalo visible:

```
mReplyTextView.setText(reply);
mReplyTextView.setVisibility(View.VISIBLE);
```

12. Ejecute la aplicación

    Ahora, cuando envía un mensaje a la segunda actividad y obtiene una respuesta, la actividad principal se actualiza para mostrar la respuesta.

Código de la solución:

```
package com.example.android.twoactivities;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
   private static final String LOG_TAG = MainActivity.class.getSimpleName();

   public static final String EXTRA_MESSAGE =
        "com.example.android.twoactivities.extra.MESSAGE";

   public static final int TEXT_REQUEST = 1;

   private EditText mMessageEditText;
   private TextView mReplyHeadTextView;
   private TextView mReplyTextView;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);

        mMessageEditText = (EditText) findViewById(R.id.editText_main);
        mReplyHeadTextView = (TextView) findViewById(R.id.text_header_reply);
        mReplyTextView = (TextView) findViewById(R.id.text_message_reply);
    }

    public void launchSecondActivity(View view) {
        Log.d(LOG_TAG, "Button clicked!");

        Intent intent = new Intent(this, SecondActivity.class);
        String message = mMessageEditText.getText().toString();

        intent.putExtra(EXTRA_MESSAGE, message);
        startActivityForResult(intent, TEXT_REQUEST);
    }

    public void onActivityResult(int requestCode, int resultCode,
                                 Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (requestCode == TEXT_REQUEST) {
            if (resultCode == RESULT_OK) {
                String reply =
                    data.getStringExtra(SecondActivity.EXTRA_REPLY);

                mReplyHeadTextView.setVisibility(View.VISIBLE);
                mReplyTextView.setText(reply);
                mReplyTextView.setVisibility(View.VISIBLE);
            }
        }
    }
}
```

Resumen

En esta práctica usted ha aprendido que:

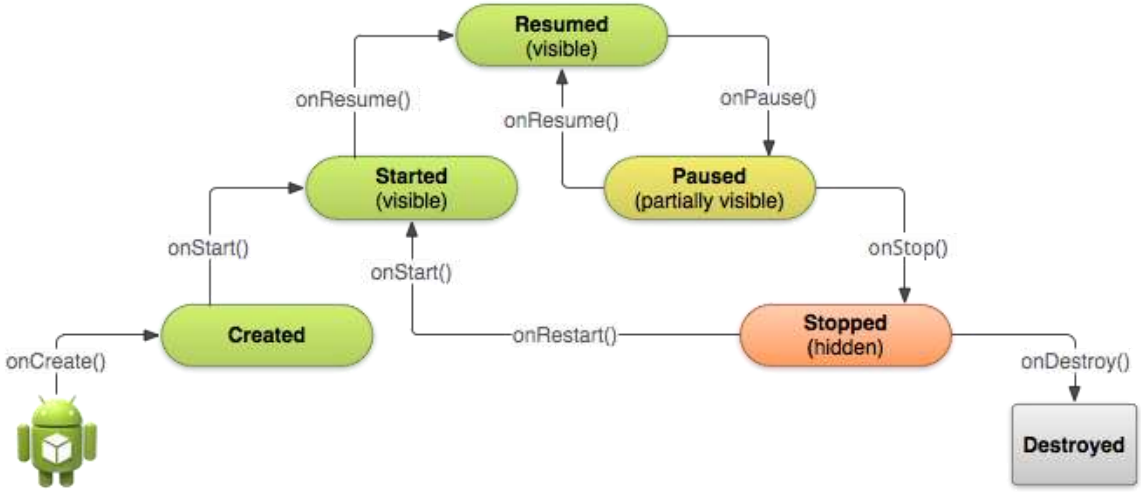
- Una actividad es un componente de aplicación que proporciona una sola pantalla centrada en una sola tarea de usuario.
    - Cada actividad tiene su propio archivo de diseño de UI (layout).
    - Puede asignar a sus actividades una relación padre/hijo para habilitar la navegación "hacia arriba" dentro de su aplicación.
- Para implementar una actividad en su aplicación, haga lo siguiente:
    - Crear una clase Java de actividad.
    - Implementar una interfaz de usuario para esa actividad.
    - Declarar esa nueva actividad en el manifiesto de la aplicación.
    - Cuando se crea un nuevo proyecto para la aplicación, o cuando agrega una nueva actividad a su aplicación, en Android Studio (con File>New>Activity), se le proporciona un código de plantilla para cada una de estas tareas.

- Los Intents le permiten solicitar una acción de otro componente en su aplicación, por ejemplo, para iniciar una actividad desde otra. Los intents pueden ser explícitos o implícitos.
    - Con intents explícitos, se indica el componente objetivo específico para recibir los datos.
    - Con intents implícitos, se especifica la funcionalidad que desea pero no el componente de destino.
    - Los intents pueden incluir datos sobre los cuales realizar una acción (como un URI) o información adicional como *extras* de intent.
    - Los extras de intent son pares de clave/valor en un paquete que se envían junto con el intent.
- Las vistas pueden hacerse visibles o invisibles con el atributo `android:visibility`

Capítulo 4:

4.1 Ciclo de vida de un Activity y el Instant State

En esta práctica, aprenderá sobre el ciclo de vida de la actividad. El ciclo de vida de la actividad es el conjunto de estados en los que puede estar una actividad durante toda su vida, desde el momento en que se crea inicialmente hasta cuando se destruye y el sistema recupera los recursos de dicha actividad. A medida que el usuario navega entre las actividades de su aplicación (así como dentro y fuera de su aplicación), esas actividades pasan en cada transición entre diferentes estados en el ciclo de vida de la actividad.



Cada etapa en el ciclo de vida de una actividad tiene un método callback correspondiente (onCreate(), onStart(), onPause(), y así sucesivamente). Cuando una actividad cambia de estado, se invoca el método callback asociado. Ya ha visto uno de estos métodos: onCreate(). Al anular cualquiera de los métodos callback del ciclo de vida en sus clases de actividad, puede cambiar el comportamiento predeterminado de cómo se comporta su actividad en respuesta a diferentes acciones del usuario o del sistema.

Los cambios en el estado de una actividad también pueden ocurrir en respuesta a los cambios de configuración del dispositivo, como girar el dispositivo de vertical a horizontal. Estos cambios de configuración provocan que la actividad se destruya y recree completamente en su estado predeterminado, lo que puede causar la pérdida de información que el usuario ha introducido en esa actividad. Es importante desarrollar su aplicación para evitar que esto ocurra y así evitar la confusión del usuario. Más adelante en esta práctica, experimentaremos con los cambios de configuración y aprenderemos a preservar el estado de sus actividades en respuesta a los cambios en la configuración del dispositivo u otros eventos del ciclo de vida de la actividad.

En esta práctica, agregará instrucciones de log a la aplicación TwoActivities y observará los cambios del ciclo de vida a medida que usa la aplicación de varias maneras. Luego comenzará a trabajar con estos cambios y explorará cómo manejar la entrada del usuario en estas condiciones.

Tarea 1. Agregar Métodos Callbacks del Ciclo de Vida al Proyecto TwoActivities

En esta tarea se implementarán todos los métodos callback del ciclo de vida de la actividad para imprimir mensajes al logcat cuando se invoquen dichos métodos. Estos mensajes del log le permitirán ver cuándo cambia el estado del ciclo de vida de la actividad y cómo esos cambios afectan a su aplicación mientras se ejecuta.

1.1 (Opcional) Copiar el proyecto TwoActivities

Para las tareas de esta práctica, se modificará el proyecto existente de TwoActivities que se completó en la última práctica. Si prefiere mantener intacto el proyecto anterior de TwoActivities, siga los pasos del Apéndice para hacer una copia del proyecto.

1.2 Implementando los callbacks en la MainActivity

1 Abra java/com.example.android.twoactivities/MainActivity.
2 En el método onCreate(), agregue las siguientes sentencias para el log:

```
Log.d(LOG_TAG, "-------");
Log.d(LOG_TAG, "onCreate");
```

3 Añada un nuevo método callback onStart(), con una sentencia en el log para ese evento:

```
@Override
public void onStart(){
    super.onStart();
    Log.d(LOG_TAG, "onStart");
}
```

SUGERENCIA: Seleccione **Code>Override Methods** en Android Studio. Aparece un cuadro de diálogo con todos los métodos posibles que puede sustituir en su clase. Al escoger uno o más métodos callbacks de la lista inserta una plantilla completa para esos métodos, incluyendo la llamada requerida a la superclase.

4 Utilice el método onStart() como plantilla para implementar los otros calbacks del ciclo de vida:
- onPause()
- onRestart()
- onResume()
- onStop()
- onDestroy()

Todos los métodos callback tienen las mismas firmas (excepto el nombre). Si copia y pega onStart() para crear estos otros métodos callback, no olvide actualizar el contenido para llamar al método correcto en la superclase, y para hacer lod del método correcto.

5 Haga Build y ejecute la aplicación

Código de la solución (no está la clase completa):

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);

   Log.d(LOG_TAG, "-------");
   Log.d(LOG_TAG, "onCreate");

   mMessageEditText = (EditText) findViewById(R.id.editText_main);
   mReplyHeadTextView = (TextView) findViewById(R.id.text_header_reply);
   mReplyTextView = (TextView) findViewById(R.id.text_message_reply);
}

@Override
public void onStart(){
   super.onStart();
   Log.d(LOG_TAG, "onStart");
}

@Override
public void onRestart() {
   super.onRestart();
   Log.d(LOG_TAG, "onRestart");
}

@Override
public void onResume() {
   super.onResume();
   Log.d(LOG_TAG, "onResume");
}

@Override
public void onPause() {
   super.onPause();
   Log.d(LOG_TAG, "onPause");
}

@Override
public void onStop() {
   super.onStop();
   Log.d(LOG_TAG, "onStop");
}

@Override
public void onDestroy() {
   super.onDestroy();
   Log.d(LOG_TAG, "onDestroy");
}
```

1.4 Implementando los callback del ciclo de vida en la SecondActivity

Ahora que ha implementado los métodos callback del ciclo de vida para MainActivity, haga lo mismo con SecondActivity.

1 Abra java/com.example.android.twoactivities/SecondActivity.
2 En la parte superior de la clase, agregue una constante para la variable LOG_TAG:

```
private static final String LOG_TAG =
     SecondActivity.class.getSimpleName();
```

3 Añada los callback del ciclo de vida y las declaraciones del log a la segunda actividad. (También puede copiar y pegar los métodos de devolución de llamada desde MainActivity)
4 Agregue una sentencia de log al método returnReply(), justo antes del método finish():

```
Log.d(LOG_TAG, "End SecondActivity");
```

Código de la solución (no está la clase completa):

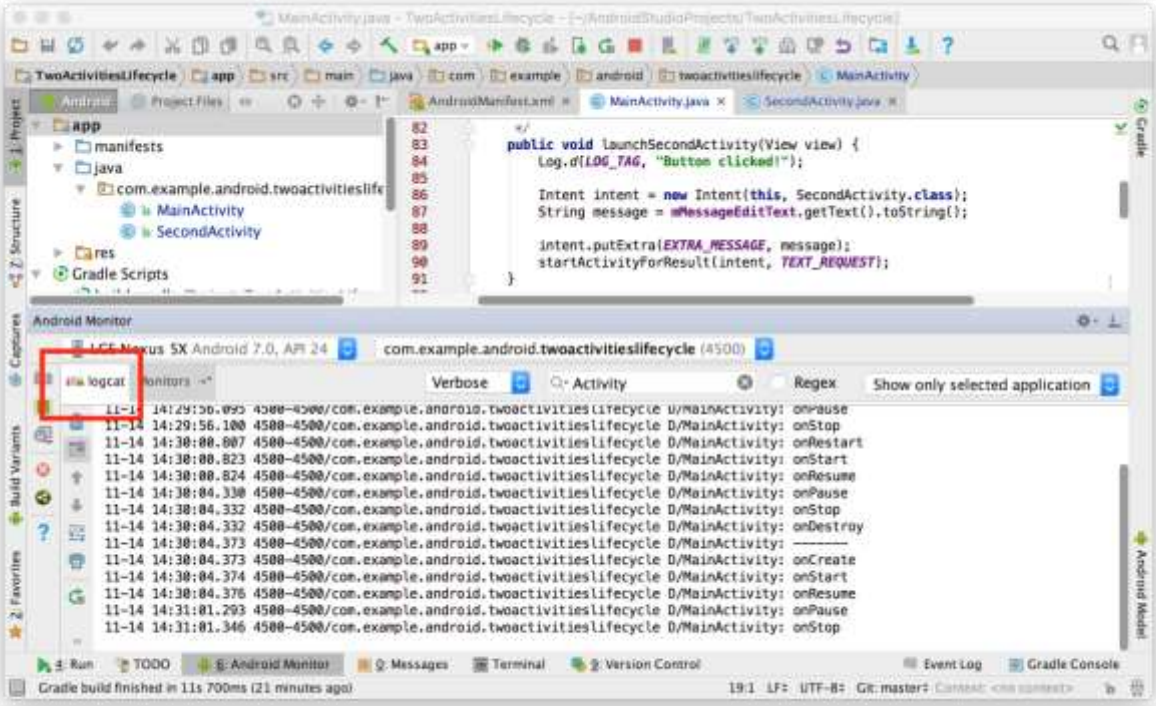
```
private static final String LOG_TAG =
SecondActivity.class.getSimpleName();

public void returnReply(View view) {
   String reply = mReply.getText().toString();

   Intent replyIntent = new Intent();
   replyIntent.putExtra(EXTRA_REPLY, reply);
   setResult(RESULT_OK, replyIntent);

   Log.d(LOG_TAG, "End SecondActivity");
   finish();
}

@Override
protected void onStart() {
   super.onStart();
   Log.d(LOG_TAG, "onStart");
}

@Override
public void onRestart() {
   super.onRestart();
   Log.d(LOG_TAG, "onRestart");
}

@Override
public void onResume() {
   super.onResume();
   Log.d(LOG_TAG, "onResume");
}

@Override
public void onPause() {
   super.onPause();
   Log.d(LOG_TAG, "onPause");
```

```
}

@Override
public void onStop() {
   super.onStop();
   Log.d(LOG_TAG, "onStop");
}

@Override
public void onDestroy() {
   super.onDestroy();
   Log.d(LOG_TAG, "onDestroy");
}
```

1.4 Observar el log a medida que la aplicación se está ejecutando

1 Ejecute la aplicación.
2 Haga clic en **Android Monitor** en la parte inferior del Android Studio para abrir el Android Monitor.
3 Seleccione la pestaña Logcat.
4 Escriba "Activity" en la caja de búsqueda del Android Monitor.

El Android logcat puede ser muy largo y desordenado. Debido a que la variable LOG_TAG en cada clase contiene bien sea las palabras MainActivity o SecondActivity, esta palabra clave le permitirá filtrar el log sólo para las cosas que le interesan.



5 Experimente con la aplicación y tenga en cuenta que los eventos del ciclo de vida ocurren en respuesta a diferentes acciones. En particular, pruebe estas cosas:

- Usar la aplicación normalmente (enviar un mensaje, responder con otro mensaje.)
- Utilice el botón Atrás para retroceder desde la segunda actividad hasta la actividad principal.
- Utilice la flecha izquierda de la barra de acciones para retroceder desde la segunda actividad hasta la actividad principal.
- Gire el dispositivo tanto en la actividad principal como en la segunda actividad en diferentes momentos de la aplicación y observe lo que sucede en el log y en la pantalla. **SUGERENCIA**: Si estás ejecutando la aplicación en un emulador, puedes simular la rotación con Ctrl-F11 o Ctrl-Fn-F11.
- Presione el botón de vista general (el botón cuadrado a la derecha de Inicio) y cierre la aplicación (toque la X).
- Vuelve a la pantalla de inicio y reinicia la aplicación.

Tarea 2. Guardar y restaurar el estado de instancia (instance state) de la actividad

Dependiendo de los recursos del sistema y del comportamiento de los usuarios, las actividades de su aplicación pueden ser destruidas y reconstruidas con mucha más frecuencia de lo que usted piensa. Puede que haya notado este conjunto de actividades en la última sección cuando rotó el dispositivo o emulador. Rotar el dispositivo es un ejemplo de un *cambio de configuración* del dispositivo. Aunque la rotación es la más común, todos los cambios de configuración hacen que la actividad actual sea destruida y recreada como si fuera nueva. Si no se tiene en cuenta este comportamiento en el código, al ocurrir un cambio de configuración, el diseño de la actividad puede volver a su apariencia predeterminada y valores iniciales, y el usuario puede perder su lugar, sus datos, o el estado del progreso en la aplicación.

El estado de cada actividad se almacena como un conjunto de pares clave/valor en un objeto Bundle llamado *estado de instancia de la actividad* (activity instance state). El sistema guarda la información predeterminada del estado en el bundle del estado de instancia justo antes de que se detenga la actividad, y lo transfiere a la nueva instancia de actividad para que se restaure.

Para evitar la pérdida de datos en las actividades cuando se destruyen y se recrean inesperadamente, se necesita implementar el método onSaveInstanceState(). El sistema llama a este método en la actividad (entre onPause() y onStop()) cuando existe la posibilidad que la actividad pueda ser destruida y recreada.

Los datos que se guardan en el instance state son específicos sólo para la instancia de esta actividad específica durante la sesión actual de la aplicación. Al detener y reiniciar una nueva sesión de la aplicación, se pierde el instance state de la actividad y las actividades volverán a su apariencia predeterminada. Si es necesario guardar los datos del usuario entre sesiones de la aplicación, debe utilizar share preference o una base de datos. De ambas formas se tratará en una práctica más adelante.

2.1 Guardar el instance state con onSaveInstanceState()

Es posible que haya notado que la rotación del dispositivo no afecta en absoluto el estado de la segunda actividad. Esto se debe a que el diseño y el estado de la segunda actividad se generan

a partir del layout y el intent que lo activó. Incluso si la actividad es recreada, el intent todavía está ahí y los datos en ese intent todavía se usan cada vez que es llamado el onCreate() de la segunda actividad.

Además, se puede notar que en ambas actividades, cualquier texto que haya escrito en el EditText del mensaje o de la respuesta se conserva incluso cuando se gira el dispositivo. Esto se debe a que la información de estado de algunas de las vistas en su diseño son guardadas automáticamente a través de cambios de configuración, y el valor actual de un EditText es uno de esos casos.

Los únicos estados de actividad que le interesan son las TextView del encabezado y el de respuesta de la actividad principal. Ambas TextView son invisibles por defecto; sólo aparecen una vez que se envía un mensaje a la actividad principal desde la segunda actividad.

En esta tarea se añadirá el código para preservar el estado de instancia de estas dos TextView usando onSaveInstanceState().

1. Abra java/com.example.android.twoactivities/MainActivity.
2. Agregue la plantilla de implementación del onSaveInstanceActivity() a la actividad, o use **Code > Override Methods** para insertar un override de la plantilla.

```
@Override
public void onSaveInstanceState(Bundle outState) {
     super.onSaveInstanceState(outState);
}
```

3. Compruebe si el encabezado está visible actualmente, y si es así, ponga ese estado de visibilidad en el paquete de estados con el método putBoolean() y la clave "reply_visible".

```
if (mReplyHeadTextView.getVisibility() == View.VISIBLE) {
    outState.putBoolean("reply_visible", true);
}
```

   Recuerde que el encabezado y el texto de la respuesta están marcados como invisibles hasta que haya una respuesta de la segunda actividad. Si la cabecera es visible, entonces hay datos de respuesta que necesitan ser guardados. Sólo estamos interesados en ese estado de visibilidad -- el texto real del encabezado no necesita ser guardado, porque ese texto nunca cambia.

4. Dentro de la misma comprobación, añada el texto de respuesta en el bundle.

```
outState.putString("reply_text", mReplyTextView.getText().toString());
```

Si el encabezado es visible, puede asumir que el mensaje de respuesta también es visible. No es necesario comprobar o guardar el estado de visibilidad actual del mensaje de respuesta. Sólo el texto real del mensaje entra en el bundle con la clave "reply_text".

Sólo guardamos el estado de aquellas vistas que puedan cambiar después de crear la actividad.

Las otras vistas de la aplicación (el EditText, el Button) pueden ser recreadas desde el diseño por defecto en cualquier momento.

Nota: El sistema guardará el estado de algunas vistas, como el contenido de EditText.

Código de la solución (no está la clase completa):

```
@Override
public void onSaveInstanceState(Bundle outState) {
   super.onSaveInstanceState(outState);

   // Si el encabezado es visible, tenemos un mensaje que necesita ser
guardado.
   // De otra manera, seguiremos usando el default layout.
   if (mReplyHeadTextView.getVisibility() == View.VISIBLE) {
       outState.putBoolean("reply_visible", true);
       outState.putString("reply_text",
mReplyTextView.getText().toString());
   }
}
```

2.2 Restaurar el instance state de la actividad con onCreate()

Una vez que haya guardado el estado de instancia de la actividad, también deberá restaurarlo cuando se vuelva a crear la actividad. Puede hacer esto ya sea en onCreate(), o implementando la llamada onRestoreInstanceState(), que se llama luego de onStart() después de crear la actividad.

La mayoría de las veces el mejor lugar para restaurar el estado de la actividad es en onCreate(), para asegurar que la interfaz de usuario incluyendo el estado esté disponible tan pronto como sea posible. A veces es conveniente hacerlo en onRestoreInstanceState() después que se haya hecho toda la inicialización, o para permitir que las subclases decidan si utilizan su implementación predeterminada.

1. En el método onCreate(), agregue una prueba para asegurarse que el bundle no es nulo.

```
if (savedInstanceState != null) {
}
```

   Cuando se crea la actividad, el sistema pasa el bundle de estado a onCreate() como un único argumento. La primera vez que onCreate() es llamado y se inicia la aplicación, el bundle es nulo: no hay un estado existente la primera vez que se inicia la aplicación. Las llamadas subsiguientes a onCreate() tienen un bundle poblado con cualquier dato que se haya almacenado en onSaveInstanceState().

2. Dentro de la prueba, obtenga la actual visibilidad (true o false) del bundle con la clave “reply_visible”

```
if (savedInstanceState != null) {
    boolean isVisible =
    savedInstanceState.getBoolean("reply_visible");
}
```

3. Agregue una prueba por debajo de la línea previa para la variable IsVisible.

```
if (isVisible) {
}
```

Si hay una clave reply_visible en el bundle de estado (y por lo tanto isVisible es verdadero), se necesitará restaurar el estado.

4. Dentro de la prueba isVisible, haga visible el encabezado.

```
mReplyHeadTextView.setVisibility(View.VISIBLE);
```

5. Obtenga el texto del mensaje de respuesta del bundle con la clave "reply_text", y configure el TextView de la respuesta para mostrar ese string.

```
mReplyTextView.setText(savedInstanceState.getString("reply_text"));
```

6. Haga que el TextView de respuesta sea visible también:

```
mReplyTextView.setVisibility(View.VISIBLE);
```

7. Ejecute la aplicación. Pruebe rotando el dispositivo o el emulador para asegurarse que el mensaje (si hay alguno) permanece en la pantalla después que la actividad es recreada.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);

   Log.d(LOG_TAG, "-------");
   Log.d(LOG_TAG, "onCreate");

   // Initialize all the view variables.
   mMessageEditText = (EditText) findViewById(R.id.editText_main);
   mReplyHeadTextView = (TextView) findViewById(R.id.text_header_reply);
   mReplyTextView = (TextView) findViewById(R.id.text_message_reply);

   // Restore the saved state. See onSaveInstanceState() for what gets
saved.
   if (savedInstanceState != null) {
       boolean isVisible =
savedInstanceState.getBoolean("reply_visible");
       // Show both the header and the message views. If isVisible is
       // false or missing from the bundle, use the default layout.
```

```
        if (isVisible) {
            mReplyHeadTextView.setVisibility(View.VISIBLE);


mReplyTextView.setText(savedInstanceState.getString("reply_text"));
            mReplyTextView.setVisibility(View.VISIBLE);
        }
    }
}
```

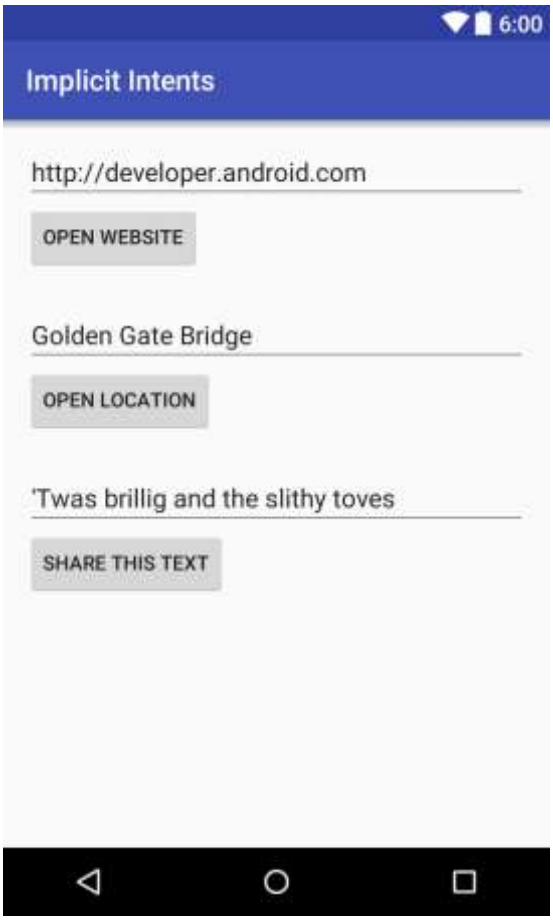
Resumen

- El ciclo de vida de la actividad es un conjunto de estados a través de los cuales una actividad migra, comenzando cuando se crea por primera vez y terminando cuando el sistema Android recupera los recursos usados de esa actividad.
- A medida que el usuario navega entre las actividades y dentro y fuera de la aplicación, cada actividad se mueve entre diferentes estados del ciclo de vida de la actividad.
- Cada estado del ciclo de vida de la actividad tiene un método callback correspondiente al que puede sobreescribir de la clase Activity. Esos son los métodos del ciclo de vida:
    - onCrear()
    - onStart()
    - onPausa()
    - onRestart()
    - onResume()
    - onStop()
    - onDestroy()
- Al sobreescribir un método callback del ciclo de vida le permite agregar un comportamiento que ocurre cuando la actividad pase a ese estado.
- Puede añadir plantillas de métodos sobreescritos a sus clases en Android Studio usando **Code > Override**.
- Los cambios en la configuración del dispositivo, como la rotación, hacen que la actividad sea destruida y recreada como si fuera nueva.
- Una parte del estado de la actividad se conserva en un cambio de configuración, incluyendo los valores actuales de los EditTexts. Para el resto, se deben grabar explícitamente los datos por uno mismo.
- Guardar el estado de instancia de la actividad en el método onSaveInstanceState().
- Los datos del estado de instancia se almacenan como pares simples clave/valor en un paquete (Bundle). Se utilizan los métodos del Bundle para poner los datos en el y volver a sacarlos.
- Restaurar el estado de instancia en onCreate(), lo cual es la forma preferida, o usando enRestoreInstanceState().

4.2 Iniciar Activities con Intents Implícitos

Resumen de la aplicación

En esta sección crearemos una nueva aplicación con una actividad y tres opciones de acción: abrir un sitio web, abrir una ubicación en un mapa y compartir un fragmento de texto. Todos los campos de texto son editables (EditText), pero contienen valores por defecto.



Tarea 1. Crear el nuevo proyecto y su layout

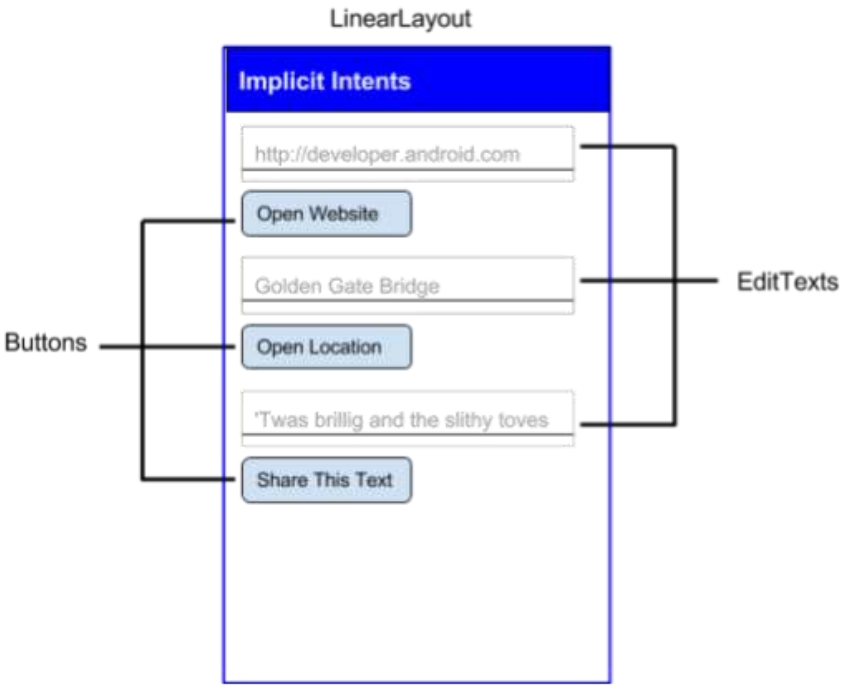
Para este ejercicio creará un nuevo proyecto y aplicación llamado “Implicit Intents” con un nuevo diseño.

1.1 Crear el proyecto

1. Inicie el Android Studio y cree un nuevo proyecto. Llame a la aplicación "Implicit Intents".
2. Seleccione **Empty Activity** para el modelo de proyecto. Haga clic en **Next**.
3. Acepte el nombre de actividad por defecto (MainActivity). Asegúrese de que la casilla **Generate Layout file** esté marcada. Haga clic en **Finish**.

1.2 Cree el layout

En esta tarea creará el layout de la aplicación. Use un LinearLayout, tres Buttons y tres EditText, de la siguiente manera:



1. Edite res/values/strings.xml para incluir estos recursos de strings:

```
<string name="edittext_uri">http://developer.android.com</string>
<string name="button_uri">Open Website</string>

<string name="edittext_loc">Golden Gate Bridge</string>
<string name="button_loc">Open Location</string>

<string name="edittext_share">\'Twas brillig and the slithy toves</string>
<string name="button_share">Share This Text</string>
```

2. Cambie el diseño a LinearLayout. Añada el atributo android:orientation y asigne el valor en "vertical".

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
```

```
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.android.implicitintents.MainActivity"
    android:orientation="vertical">
```

3. Remueva el TextView “Hello World”
4. Añada un EditText y un Button al diseño de la función Open Website. Utilice estos valores para los atributos:

Atributo (EditText)	**Valor (EditText)**
android:id	"@+id/website_edittext"
android:layout_width	"match_parent"
android:layout_height	"wrap_content"
android:text	"@string/edittext_uri"
Atributo (Button)	**Valor (Button)**
android:id	"@+id/open_website_button"
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:layout_marginBottom	"24dp"
android:text	"@string/button_uri"
android:onClick	"openWebsite"

5. Añada un segundo EditText y otro Button para la función Open Location.
6. Utilice los mismos atributos que en el paso anterior, pero modifíquelos tal como se indica a continuación:

Atributo (EditText)	**Valor (EditText)**
android:id	"@+id/location_edittext"
android:text	"@string/edittext_loc"
Atributo (Button)	**Valor (Button)**
android:id	"@+id/open_location_button"
android:text	"@string/button_loc"
android:onClick	"openLocation"

7. Añada un tercer EditText y otro Button para la función Share This. Realice estas modificaciones:

Atributo (EditText)	**Valor (EditText)**
android:id	"@+id/share_edittext"

android:text	"@string/edittext_share"
Atributo (Button)	**Valor (Button)**
android:id	"@+id/share_text_button"
android:text	"@string/button_share"
android:onClick	"shareText"

Código de la solución: Dependiendo de la versión del Android Studio, el código debe lucir parecido al siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context="com.example.android.implicitintents.MainActivity"
   android:orientation="vertical">

   <EditText
       android:id="@+id/website_edittext"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:text="@string/edittext_uri" />

   <Button
       android:id="@+id/open_website_button"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_marginBottom="24dp"
       android:onClick="openWebsite"
       android:text="@string/button_uri" />

   <EditText
       android:id="@+id/location_edittext"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:text="@string/edittext_loc" />

   <Button
       android:id="@+id/open_location_button"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_marginBottom="24dp"
       android:onClick="openLocation"
       android:text="@string/button_loc" />

   <EditText
```

```
        android:id="@+id/share_edittext"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/edittext_share" />

    <Button
        android:id="@+id/share_text_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="24dp"
        android:onClick="shareText"
        android:text="@string/button_share" />
</LinearLayout>
```

Tarea 2. Implementando el “open website”

En esta tarea, implementará el método manejador de hacer clic en el primer botón del layout ("Open Website"). Esta acción utiliza un intent implícito para enviar la URI dada a una actividad que pueda manejar tal intent (como lo es un navegador web).

2.1 definir el método openWebsite

1. Abra el archivo MainActivity.java
2. Agregue una variable privada en l aparte superior de la clase que contenga al objeto EditText para el URI del sitio web.

```
private EditText mWebsiteEditText;
```

3. En el método onCreate(), utilice findViewById() para obtener una referencia a la instancia EditText y asignarla a esa variable privada:

```
mWebsiteEditText = (EditText) findViewById(R.id.website_edittext);
```

4. Cree un Nuevo método llamado openWebsite(), con esta firma:

```
public void openWebsite(View view) { }
```

5. Obtenga el valor del string del EditText:

```
String url = mWebsiteEditText.getText().toString();
```

6. Codifique tal String en un objeto Uri:

```
Uri webpage = Uri.parse(url);
```

7. Cree un nuevo Intent con Intent.ACTION_VIEW como la acción y la URI como los datos:

```
Intent intent = new Intent(Intent.ACTION_VIEW, webpage);
```

El constructor de este intent es diferente del que se usó para crear un intent explícito. En el constructor anterior, se especificó el contexto actual y un componente determinado (la clase de la actividad) para enviar el intent. En este constructor se especifica una acción y los datos para esa acción. Las acciones son definidas por la clase de la intención y pueden incluir ACTION_VIEW (para ver los datos dados), ACTION_EDIT (para editar los datos dados), o ACTION_DIAL (para marcar un número de teléfono). En este caso la acción es ACTION_VIEW porque queremos abrir y ver la página web especificada por la URI en la variable webpage.

8. Utilice el resolveActivity() y el Android package manager para encontrar una actividad que pueda gestionar el intent implícito. Compruebe que la solicitud se haya resuelto correctamente.

```
if (intent.resolveActivity(getPackageManager()) != null) {
}
```

Esto ejecuta una solicitud que haga coincidir la acción del intent y los datos con los fitros del intent con alguna de las aplicaciones instaladas en el dispositivo para asegurarse de que hay al menos una actividad que puede manejar tal requerimiento.

9. Dentro de la sentencia if, haga una llamada al método startActivity() para enviar el intent.

```
startActivity(intent);
```

10. Agregur un bloque else para imprimir un mensaje en el log si el intent no puede ser resuelto.

```
} else {
   Log.d("ImplicitIntents", "Can't handle this!");
}
```

Código de la solución (no está la clase completa):

```
public void openWebsite(View view) {
   // Get the URL text.
   String url = mWebsiteEditText.getText().toString();

   // Parse the URI and create the intent.
   Uri webpage = Uri.parse(url);
   Intent intent = new Intent(Intent.ACTION_VIEW, webpage);

   // Find an activity to hand the intent and start that activity.
   if (intent.resolveActivity(getPackageManager()) != null) {
       startActivity(intent);
   } else {
       Log.d("ImplicitIntents", "Can't handle this intent!");
   }
}
```

Tarea 3. Implementando “open location”

En esta tarea implementará el método on-click handler para el segundo botón en la UI ("Open Location"). Este método es casi idéntico al método openWebsite(). La diferencia es el uso de un geo URI para indicar la ubicación en un mapa. Puede usar una geo URI con latitud y longitud, o usar un string de consulta para una ubicación general. En este ejemplo usaremos el último.

3.1 Defina el método openLocation

1. Abra el archive MainActivity (java/com.example.android.implicitintents/MainActivity).
2. Agregue una variable privada en la parte superior de la clase que contenga al objeto EditText que tiene la ubicación URI.

```
private EditText mLocationEditText;
```

3. En el método onCreate(), utilice findViewById() para obtener una referencia a la instancia EditText y asignarla a esa variable privada:

```
mLocationEditText = (EditText) findViewById(R.id.location_edittext);
```

4. Cree un Nuevo método llamado openLocation() el cual responderá al método onClick() del botón OpenLocation. Use la misma firma del método openWebsite().
5. Obtenga el valor del string de la variable mLocationEditText:

```
String loc = mLocationEditText.getText().toString();
```

6. Codifique tal string en un objeto Uri con una consulta de búsqueda geográfica:

```
Uri addressUri = Uri.parse("geo:0,0?q=" + loc);
```

7. Cree un Nuevo Intent con Intent.ACTION_VIEW como la acción y addressUri como los datos.

```
Intent intent = new Intent(Intent.ACTION_VIEW, addressUri);
```

8. Resuelva el intent y asegúrese que este se ejecuta exitosamente. Si es así, ejecuta startActivity(), de lo contrario muestra un mensaje de error en el log.

```
if (intent.resolveActivity(getPackageManager()) != null) {
   startActivity(intent);
} else {
   Log.d("ImplicitIntents", "Can't handle this intent!");
}
```

Código de la solución (no está la clase completa):

```
public void openLocation(View view) {
```

```
    // Get the string indicating a location. Input is not validated; it is
    // passed to the location handler intact.
    String loc = mLocationEditText.getText().toString();

    // Parse the location and create the intent.
    Uri addressUri = Uri.parse("geo:0,0?q=" + loc);
    Intent intent = new Intent(Intent.ACTION_VIEW, addressUri);

    // Find an activity to handle the intent, and start that activity.
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    } else {
        Log.d("ImplicitIntents", "Can't handle this intent!");
    }
}
```

Tarea 4. Implementando Share This Text

Las acciones de compartir son una forma fácil para que los usuarios compartan elementos de la aplicación con redes sociales y otras aplicaciones. Aunque podría crear una acción de compartición en su propia aplicación utilizando intents implícitos, Android proporciona la clase helper ShareCompat.IntentBuilder para facilitar la implementación del copartir. Puede utilizar ShareCompat.IntentBuilder para crear un intent y mostrar un selector que permita al usuario elegir la aplicación de destino para compartir.

En esta tarea final implementaremos compartir un trozo de texto en una edición de texto con la clase ShareCompat.IntentBuilder.

4.1 Implementando el método shareText

1. Abra el archivo MainActivity.java
2. Agregue una variable privada en la parte superior de la clase que contenga al objeto EditText que tiene la ubicación URI.

```
private EditText mShareTextEditText;
```

3. En el método onCreate(), utilice findViewById() para obtener una referencia a la instancia EditText y asignarla a esa variable privada:

```
mShareTextEditText = (EditText) findViewById(R.id.share_edittext);
```

4. Cree un nuevo método llamado shareThis() el cual responderá al método onClick() del botón Share This Text. Use la misma firma del método openWebsite().
5. Obtenga el valor del string de la variable `mShareTextEditText`:

```
String txt = mShareTextEditText.getText().toString();
```

6. Defina el tipo MIME del texto a compartir:

```
String mimeType = "text/plain";
```

7. Llame a ShareCompat.IntentBuilder con estos métodos:

```
ShareCompat.IntentBuilder
       .from(this)
       .setType(mimeType)
       .setChooserTitle("Share this text with: ")
       .setText(txt)
       .startChooser();
```

Este formato, con todos los métodos del constructor encadenados en una sola sentencia, es una forma fácil de crear y lanzar el intent. Puede añadir cualquiera de los métodos adicionales a esta lista.

Esta llamada a ShareCompat.IntentBuilder utiliza estos métodos:

Método	**Descripción**
from()	La actividad que lanza este intent compartido (this).
setType()	El tipo MIME del element a ser compartido.
setChooserTitle()	El título que aparece en el selector de aplicaciones del sistema.
setText()	El texto actua ha ser compartido.
startChooser()	Muestra eñ selector de aplicaiones del Sistema y envía el intent.

Código de la solución (no está la clase completa):

```
public void shareText(View view) {
    String txt = mShareTextEditText.getText().toString();
    String mimeType = "text/plain";

    ShareCompat.IntentBuilder
            .from(this)
            .setType(mimeType)
            .setChooserTitle("Share this text with: ")
            .setText(txt)
            .startChooser();
}
```

Tarea 5. Recibir intents implícitos

Hasta ahora, has creado aplicaciones que usan intents explícitos e implícitos para lanzar una actividad de otras aplicaciones. En esta tarea veremos el problema desde el otro lado: permitir que una actividad en nuestra aplicación responda a intents implícitos enviados desde otra aplicación.

Las actividades de nuetra aplicación siempre se pueden activar desde dentro o fuera de ella con intents explícitos. Para permitir que una actividad reciba intents implícitos, definiremos un *intent filter* en el manifest para indicar cuales intents implícitos nuestra actividad está interesa en manejar.

Para hacer coincidir nuestro requerimiento con una aplicación específica instalada en el dispositivo, el sistema Android hace coincidir nuestro intent implícito con una actividad cuyos intent filters indican que pueden realizar esa acción. Si hay varias aplicaciones instaladas que coinciden, al usuario se le presenta un selector de aplicaciones que le permite seleccionar la aplicación que desea usar para manejar ese intent.

Cuando una aplicación en el dispositivo envía un intent implícito, el sistema Android hace coincidir la acción y los datos de esa intención con las actividades disponibles que incluyan los intent filters correctos. Si los intent filters de nuestra actividad coinciden con el intent, nuestra actividad puede o bien manejar el intent por sí misma (si es la única actividad que coincide), o (si hay varias coincidencias) aparece un selector de aplicaciones que permite al usuario elegir qué aplicación prefiere para ejecutar tal acción.

En esta tarea crearemos una aplicación muy simple que recibe intents implícitos para abrir la URI de una página web. Cuando se activa por un intent implícito, esa aplicación muestra la URI solicitada como un string en un TextView.

5.1 Crear el proyecto y el layout

1. Inicie el Android Studio y cree un nuevo proyecto.
2. Llame a la aplicación " Implicit Intents Receiver ".
3. Seleccione **Empty Activity** para el modelo del proyecto.
4. Acepte el nombre de la actividad por defecto (MainActivity). Haga clic en **Next**.
5. Asegúrese de que la casilla **Generate Layout file** esté marcada. Haga clic en **Finish**.
6. Abria el archivo `res/layout/activity_main.xml`.
7. Cambie los atributos del TextView existente ("Hola Mundo"):

Atributo	Valor
android:id	"@+id/text_uri_message"
android:layout_width	wrap_content
android:layout_height	wrap_content
android:textSize	"18sp"
android:textStyle	"bold"

8. Elimine el atributo `android:text`. No debe haber ningún texto por defecto en este TextView, debido a que allí agregará el URI desde el intent en onCreate().

5.2 Modifique el manifest del Android para agregar el intent filter

1. Abra el archivo `manifests/AndroidManifest.xml`.
2. Note que la main activity ya tiene su intent filter:

```
<intent-filter>
   <action android:name="android.intent.action.MAIN" />
   <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

Este intent filter, el cual forma parte del manifest predeterminado del proyecto, indica que esta actividad es el principal punto de entrada para la aplicación (tiene un intent de acción "android.intent.action.MAIN"), y que esta actividad debería aparecer como un elemento de primer nivel en el lanzador (su categoría es "`android.intent.category.LAUNCHER`").

3. Agregue una segunda etiqueta `<intent-filter>` dentro de `<activity>` e incluya estos elementos:

```
<action android:name="android.intent.action.VIEW" />
<category android:name="android.intent.category.DEFAULT" />
<category android:name="android.intent.category.BROWSABLE" />
<data android:scheme="http" android:host="developer.android.com" />
```

Estas líneas definen un intent filter para esta actividad, esto es, la clase de intents que la actividad puede manejar. Este intent filter declara estos elementos:

Tipo de Filtro	Valor	Coincidencia
action	"android.intent.action.VIEW"	Todos los intents con acciones de vista.
category	"android.intent.category.DEFAULT"	Todos los intents implícitos. Esta categoría debe ser incluida en nuestra actividad para recibir **cualquier** intent implícito.
category	"android.intent.category.BROWSABLE"	Requerimientos de enlaces navegables desde páginas web, correo electrónico u otras fuentes.
data	android:scheme="http" android:host="developer.android.com"	URIs que contienen un esquema http Y un host de nombre developer.android.com.

Tenga en cuenta que el filtro data tiene una restricción tanto en el tipo de enlaces que aceptará como en el nombre de host para esos URI. Si prefiere que su receptor pueda aceptar cualquier enlace, puede dejar el elemento `<data>` totalmente fuera.

Código de la solución

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.android.implicitintentsreceiver">

   <application
       android:allowBackup="true"
       android:icon="@mipmap/ic_launcher"
       android:label="@string/app_name"
       android:supportsRtl="true"
       android:theme="@style/AppTheme">

       <activity android:name=".MainActivity">
           <intent-filter>
               <action android:name="android.intent.action.MAIN" />
               <category android:name="android.intent.category.LAUNCHER" />
           </intent-filter>

           <intent-filter>
               <action android:name="android.intent.action.VIEW" />
               <category android:name="android.intent.category.DEFAULT" />
               <category android:name="android.intent.category.BROWSABLE" />
               <data android:scheme="http"
android:host="developer.android.com" />
           </intent-filter>
       </activity>
   </application>
</manifest>
```

5.3 Procese el intent

En el método onCreate() de la actividad, se procesa el intent entrante para cualquier data o extras que incluya. En este caso, el intent implícito entrante tiene la URI almacenada en los datos del intent.

1. Abra `MainActivity.java`.
2. En el método onCreate(), obtenga el intent entrante que usará para activar la activity:

```
Intent intent = getIntent();
```

3. Obtenga la data del intent. Los datos del Intent son siempre un objeto URI:

```
Uri uri = intent.getData();
```

4. Asegúrese que la variable `uri` no sea null. Si la verificación pasa, cree un string a partir del objeto URI:

```
if (uri != null) {
   String uri_string = "URI: " + uri.toString();
```

```
}
```

Dentro del mismo bloque if, obtenga el TextView para el mensaje:

```
TextView textView = (TextView) findViewById(R.id.text_uri_message);
```

5. Además dentro del mismo bloque if, configure el texto de ese TextView al del URI:

```
textView.setText(uri_string);
```

6. Ejecute la aplicación del receptor.

   Ejecutar la aplicación por sí sola muestra una actividad en blanco sin texto. Esto se debe a que la actividad fue activada desde el lanzador del sistema, y no con el intent de otra aplicación.

7. Ejecute la aplicación ImplicitIntents y haga clic en Open Website con la URI predeterminada.

   Aparece un selector de aplicaciones que pregunta si desea utilizar el navegador predeterminado o la aplicación ImplicitIntentsReceiver. Elija "Sólo una vez" para la aplicación del receptor. Se inicia la aplicación ImplicitIntentsReceiver y el mensaje muestra la URI de la solicitud original.

8. Pulse el botón Atrás e introduzca una URI diferente. Haga clic en Open Website.

   La aplicación del receptor tiene un intent filter muy restrictivo que sólo coincide con el protocolo URI exacto (http) y el host (developer.android.com). Cualquier otra URI se abrirá en el navegador web predeterminado.

Resumen

- Los intents implícitos le permiten activar una actividad si conoce la acción, pero no la aplicación o actividad específica que la manejará.
- Las actividades que pueden recibir intents implícitos deben definir intent filters en el manifest del Android que coincidan con una o más acciones y categorías de intent.
- El sistema Android iguala el contenido de una intent implícito y los intent filters de todas las actividades disponibles para determinar qué actividad activar. Si hay más de una actividad disponible, el sistema proporciona un selector para que el usuario pueda seleccionar una.
- La clase ShareCompat.IntentBuilder facilita la creación de intents implícitos para compartir datos en medios sociales o correo electrónico.

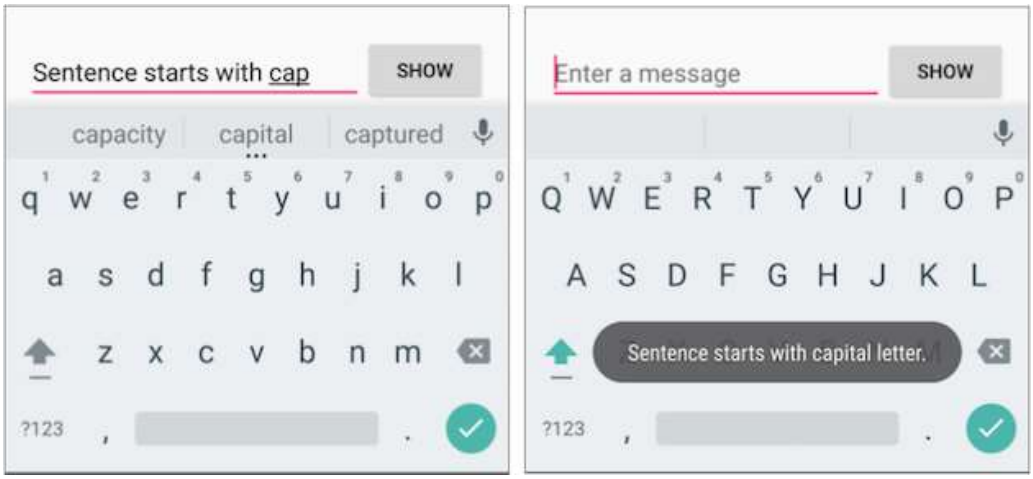
Capítulo 5:

Interacción con el Usuario

5.1 Uso de teclados, controles de entrada, alertas y selectores

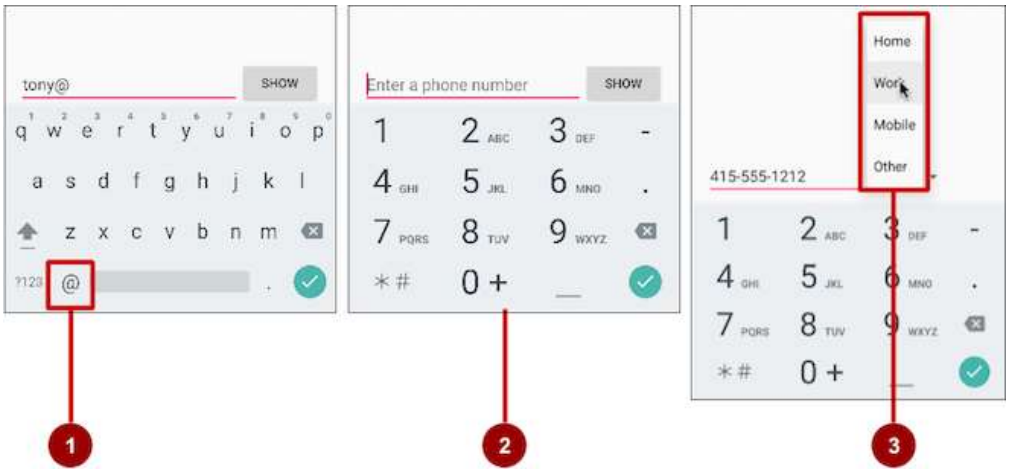
Vista general de la aplicación

En esta práctica, creará y construirá una nueva aplicación llamada **Keyboard Samples** para experimentar con el atributo `android:inputType` para el elemento EditText UI. Cambiará el teclado para que sugiera correcciones ortográficas y ponga en mayúsculas cada nueva oración, como se muestra en el lado izquierdo de la figura de abajo. Para mantener la aplicación simple, mostrará el texto ingresado en un mensaje toast, tal como se muestra en el lado derecho de la figura siguiente.



También cambiará el teclado a uno que ofrezca el símbolo "@" en un lugar destacado para introducir direcciones de correo electrónico, y a un teclado telefónico para introducir números de teléfono, como se muestra en el lado izquierdo y en el centro de la figura de abajo. Como reto, implementará un listener para la tecla de acción en el teclado con el fin de enviar un intent implícito a otra aplicación para marcar el número de teléfono.

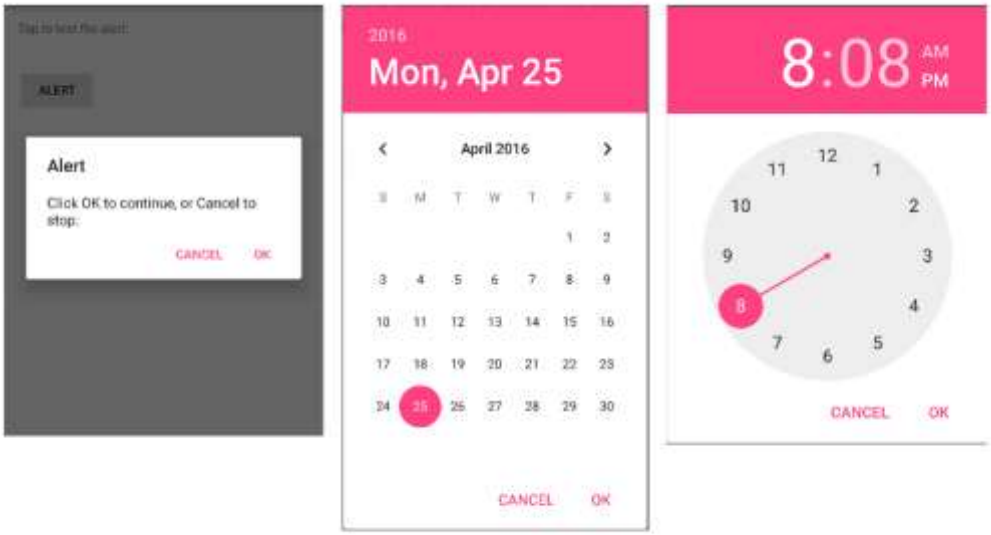
A continuación, copiará la aplicación para crear un Phone Number Spinner que ofrece un control de entrada tipo spinner para seleccionar la etiqueta (Hogar, Trabajo, Otro, Personalizado) para el número de teléfono, tal como se muestra en el lado derecho de la figura de abajo.



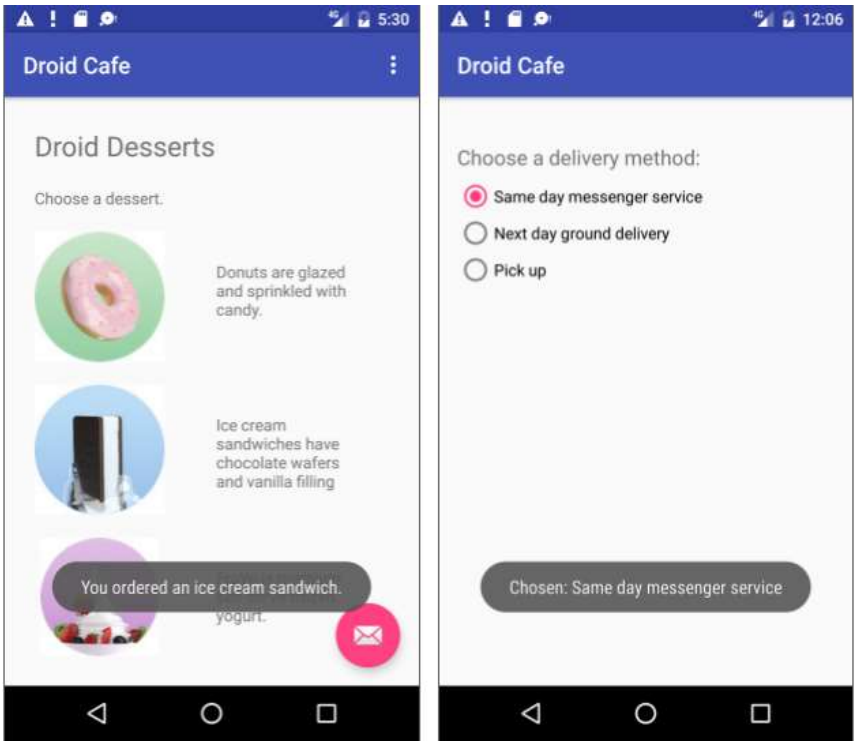
La figura anterior muestra lo siguiente:

1. El teclado del correo electrónico con el símbolo "@" en un lugar fácil de encontrar
2. El teclado del teléfono
3. El spinner

También creará una Muestra de Alerta (Alert Sample) para experimentar con un diálogo de alerta, muestrado en el lado izquierdo de la figura siguiente, y Seleccionadores de Fecha y Hora (Date Time Pickers) para experimentar con un selector de fecha y un selector de hora, que se muestran en el centro y en el lado derecho de la figura de abajo, y usar las selecciones de hora y fecha en la aplicación.



Las últimas tareas consisten en crear una aplicación a partir de la plantilla de la Actividad Básica que permita al usuario pulsar en los botones de imagen para iniciar una actividad, como se muestra en el lado izquierdo de la figura de abajo, y elegir una opción de entrega única entre las botones de opción tipo radio para un pedido de alimentos, como se muestra en el lado derecho de la figura siguiente.



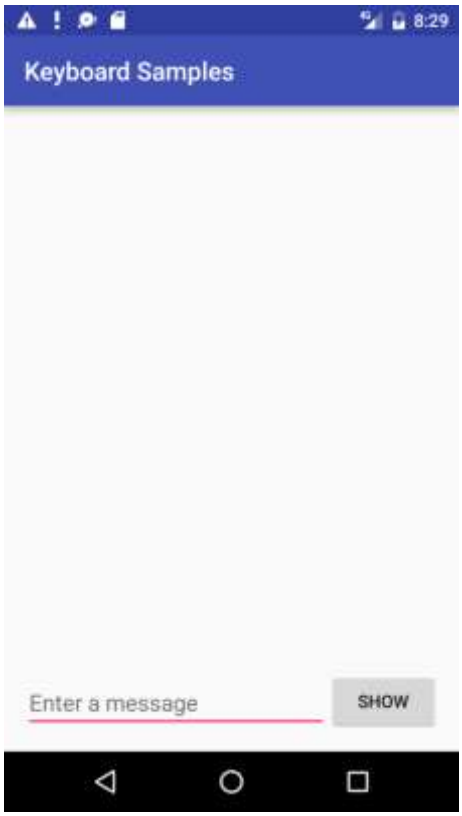
Tarea 1. Experimentar con los atributos de un teclado de ingreso de texto

Al tocar un campo de texto EditText editable, el cursor se coloca en el campo de texto y muestra automáticamente el teclado en pantalla. Cambiará los atributos del campo de entrada de texto para que el teclado sugiera correcciones ortográficas mientras escribe, e inicie automáticamente cada nueva oración con mayúsculas. Por ejemplo:

- `android:inputType="textCapSentences"`: Configura el teclado para que aparezcan mayúsculas al principio de las oraciones.
- `android:inputType="textAutoCorrect"`: Configura el teclado para mostrar las correcciones ortográficas automáticamente a medida que se introducen los caracteres.
- `android:inputType="textMultiLine"`: Permite que la tecla Return del teclado finalice las líneas y cree nuevas líneas en blanco sin cerrar el teclado.
- `android:inputType="textPassword"`: Coloca caracteres de punto para ocultar la contraseña introducida por el usuario.

1.1 Crear el layout principal y método showText

Añadirá un botón y cambiará el elemento TextView por un elemento EditText para que el usuario pueda introducir texto. El diseño de la aplicación se verá como la figura siguiente.



1. Cree un nuevo proyecto llamado **Keyboard Samples** (Muestras de teclado) y seleccione la plantilla Empty Activity.
2. Abra el archivo de diseño activity_main.xml. En el Editor de diseño, haga clic en la pestaña Text en la parte inferior de la pantalla y cambie el grupo de vista raíz a RelativeLayout, como ha hecho en ejercicios anteriores.
3. Añada un botón encima del elemento TextView existente con los siguientes atributos:

Atributo del Button	**Nuevo valor**
android:id	"@+id/button_main"
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:layout_alignParentBottom	"true"
android:layout_alignParentRight	"true"
android:onClick	"showText"
android:text	"Show"

4. Extraiga el recurso de string para el valor del atributo `android:text` para crear una entrada en strings.xml: Sitúe el cursor en "**Show**", pulse Alt-Enter (Opción-Enter en el Mac) y seleccione **Extract string resources**. Luego cambie el Nombre del recurso para el valor de string **show**.

   Extraer recursos de string hace que el proyecto para la aplicación sea más flexible al quere cambiar strings. Las asignaciones de recursos de string se almacenan en el archivo strings.xml (en **app > res > values**). Se puede editar este archivo para cambiar las asignaciones de strings de modo que la aplicación pueda localizarse en otro idioma. Por ejemplo, el valor "Show" para el recurso llamado `show` podría cambiarse a "Montrer" para la versión francesa de la aplicación.

5. Cambie el elemento TextView existente de la siguiente manera:
   a. Elimine el atributo `android:text` que especificó "Hello World!".
   b. Si el elemento TextView incluye algún atributo de layout-constraint, elimínelo.
   c. Cambie la etiqueta TextView a una etiqueta EditText, y asegúrese de que la etiqueta final sea />.
   d. Agregue o cambie los siguientes atributos:

EditText atributo	TextView valor anterior	EditText nuevo valor
android:id		"@+id/editText_main"
android:layout_width	"wrap_content"	"match_parent"
android:layout_height	"wrap_content"	"wrap_content"
android:layout_alignParentBottom		"true"
android:layout_toLeftOf		"@id/button_main"
android:hint		"Enter a message"

   Ya ha aprendido acerca de los atributos `android:layout_toLeftOf` y `android:layout_alignParentBottom` en una lección anterior. Estos atributos relacionados con la presentación funcionan con el grupo de vistas RelativeLayout para posicionar las vistas hijo relativas entre sí o con el padre. El atributo `android:hint` establece el texto que aparecerá en el campo que proporciona una pista para que el usuario proporcione información, como "Enter a message".

6. Extraiga el recurso de string para el valor del atributo `android:hint` "Enter a message" al nombre del recurso **enter**. Dependiendo de su versión del Android Studio, el archivo de layout activity_main.xml tendrá un aspecto parecido al siguiente:

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
```

```
tools:context="com.example.android.keyboardsamples.MainActivity">

<Button
android:id="@+id/button_main"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentBottom="true"
android:layout_alignParentRight="true"
android:onClick="showText"
android:text="@string/show" />

<EditText
android:id="@+id/editText_main"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_alignParentBottom="true"
android:layout_toLeftOf="@id/button_main"
android:hint="@string/enter" />

</RelativeLayout>
```

7. Abra MainActivity.java e introduzca el siguiente método showText, el cual recupera la información introducida en el elemento EditText y la muestra en un mensaje toast:

```
 public void showText(View view) {
   EditText editText = (EditText) findViewById(R.id.editText_main);
   if (editText != null) {
      String showString = editText.getText().toString();
      Toast.makeText(this, showString, Toast.LENGTH_SHORT).show();
   }
}
```

8. Abra strings.xml (en **app > res > values**), y edite el valor `app_name` y coloque "**Keyboard Samples**" (asegúrese de incluir un espacio entre "Keyboard" y "Samples").
9. Ejecute la aplicación y examine cómo funciona el teclado.

Pulsando sobre el botón **Show** se muestra el mensaje toast de la entrada de texto.

Para cerrar el teclado en pantalla, pulse en la flecha que apunta hacia abajo en la fila inferior de iconos.

En la disposición estándar del teclado, aparece un icono con una marca de verificación en un círculo verde, como se muestra a continuación, en la esquina inferior derecha del teclado. Esto se conoce como tecla Return (o Enter) y se utiliza para introducir una nueva línea:

Con los atributos predeterminados para el elemento EditText, al pulsar la tecla Intro se añade otra línea de texto. En la siguiente sección, cambiará el teclado para que escriba las frases en mayúsculas mientras escribe. Como resultado de establecer el atributo `android:inputType`, el

atributo por defecto de la tecla Return cambia para abandonar el foco del elemento EditText y cerrar el teclado.

1.2 Configurar el teclado para comenzar las oraciones en las mayúsculas

1. Agregue el atributo `android:inputType` al elemento EditText usando el valor `textCapSentences` para poner el teclado en mayúsculas al principio de una oración, de manera que los usuarios puedan comenzar automáticamente una oración con mayúsculas:

```
android:inputType="textCapSentences"
```

2. Ejecute la aplicación

Las mayúsculas aparecerán en el teclado al principio de las oraciones. Al tocar la tecla Intro del teclado, el teclado se cierra y la entrada de texto finaliza. Aún puede pulsar en el campo de entrada de texto para agregar más texto o editarlo. Toque **Show** para mostrar el texto en un mensaje toast.

Para detalles acerca del atributo `android:inputType`, vea Specifying the Input Method Type (en https://developer.android.com/training/keyboard-input/style.html).

1.3 Configurar el teclado para ocultar el password when es introducido

1. Cambie el elemento EditText para utilizar el valor `textPassword` para el atributo `android:inputType`.

```
android:inputType="textPassword"
```

2. Cambie el `android:hint` para "**Enter your password**".
3. Ejecute la aplicación.

Los caracteres que el usuario introduce se convierten en puntos para ocultar el password introducido. Para más ayuda, vea Text Fields (en https://developer.android.com/training/keyboard-input/style.html).

Código de la solución:

```
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

/**
 * Displays the entered text in a toast message.
 * This app is designed to show different keyboards when entering text.
 */
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

    }

    /**
     * Shows the text entered into editText_main in a Toast message.
     * @param view  View with editText_main.
     */
    public void showText(View view) {
        // Associate editText with the editText_main EditText element.
        EditText editText = (EditText) findViewById(R.id.editText_main);
        if (editText != null) {
            // Assign showString to the text that was entered.
            String showString = editText.getText().toString();
            // Make the Toast message with showString.
            Toast.makeText(this, showString, Toast.LENGTH_SHORT).show();
        }
    }
}
```

Tarea 2. Cambiar el tipo de teclado

Cada campo de texto espera un cierto tipo de entrada de texto, tal como una dirección de correo electrónico, un número de teléfono, una contraseña o simplemente texto. Es importante especificar el tipo de entrada para cada campo de texto en la aplicación para que el sistema muestre el método de entrada más apropiado, como por ejemplo:

- El teclado estándar en pantalla para texto sin formato
- El teclado para una dirección de correo electrónico que incluye el símbolo "@" en un lugar destacado.
- El teclado del teléfono para un número de teléfono

2.1 Usando el teclado de correo electrónico

Modifique el elemento EditText de la actividad principal para que muestre un teclado de correo electrónico en lugar de un teclado estándar:

En el elemento EditText del archivo de diseño activity_main.xml, cambie el atributo `android:inputType` por el siguiente:

```
android:inputType="textEmailAddress"
```

1. Cambie el atributo `android:hint` para “**Enter an email address**”.
2. Extraiga el recurso de string para el valor del `android:hint` en `enter_mail`
3. Ejecute la app. Al tocar el campo, aparece el teclado de correo electrónico en pantalla con el símbolo "@" situado junto a la tecla espaciadora.

2.2 Usando un teclado de teléfono

Modifique el elemento EditText de la actividad principal para que muestre un teclado de teléfono en lugar de un teclado estándar:

1. En el elemento EditText del archivo de diseño activity_main.xml, cambie el atributo `android:inputType` por el siguiente:

```
android:inputType="phone"
```

2. Cambie el atributo `android:hint` para "**Enter an email address**".
3. Extraiga el recurso de string para el valor del `android:hint` en `enter_phone`
4. Ejecute la aplicación

Al tocar el campo ahora aparece el teclado del teléfono en pantalla en lugar del teclado estándar.

Nota: Cuando ejecute la aplicación en el emulador, el campo seguirá aceptando texto en lugar de números si escribe en el teclado de la computadora. Sin embargo, cuando se ejecuta en el dispositivo, el campo sólo acepta los números del teclado.

Tarea 3. Agregar un control de entrada spinner para seleccionar una etiqueta de teléfono

Los controles de entrada son los componentes interactivos de la interfaz de usuario de la aplicación. Android proporciona una amplia variedad de controles que puede utilizar en su interfaz de usuario, como botones, barras de búsqueda, casillas de verificación, botones de zoom, botones de conmutación, giradores (spinners) y muchos más. (Para obtener más información sobre los controles de entrada, consulte Input Controls.

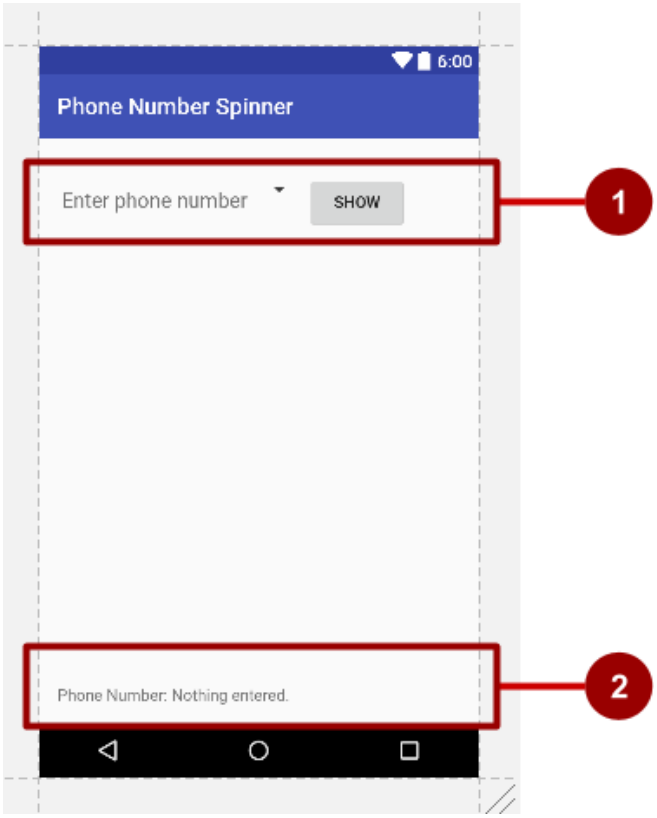
Un spinner proporciona una manera rápida de seleccionar un valor de un conjunto. Al tocar el spinner aparece una lista desplegable con todos los valores disponibles, de la cual el usuario puede seleccionar uno. Si proporciona sólo dos o tres opciones, es posible que desee utilizar botones de selección (radio buttons) para las opciones, si tiene espacio en el diseño para ellas; sin embargo, con más de tres opciones, un spinner funciona muy bien ya que se desplaza según sea necesario para mostrar los elementos y ocupa poco espacio en el diseño.

Para obtener más información acerca de los spinner, consulte Spinners.

Para proporcionar una forma de seleccionar una etiqueta de un número de teléfono (como Casa, Trabajo, Móvil y Otro), puede agregar un spinner al layout para que aparezca justo al lado del campo de número de teléfono.

3.1 Copie el proyecto KeyboardSamples y modifique el layout

Use la siguiente figura como una guía para el layout de la actividad principal:



En la figura anterior:

1. El primer LinearLayout con una vista EditText, un icono de spinner y el botón **Show**.
2. El Segundo LinearLayout con dos TextViews.

Siga los siguientes pasos:

1. Copie la carpeta del proyecto **KeyboardSamples**, renómbrela como **PhoneNumberSpinner**, y *refactorice* para asignar el nuevo nombre a lo largo de todo el proyecto. (Consulte el Apéndice para obtener instrucciones sobre cómo copiar un proyecto).
2. Después de refactorizar, cambie el valor `<string name="app_name">` en el archivo strings.xml (dentro de **app > res > values**) a **Phone Number Spinner** (con los espacios) como el nombre de la aplicación.
3. Abra el archivo de diseño activity_main.xml.
4. Incluya los elementos EditText y Button existentes de la lección anterior dentro de un LinearLayout con una orientación horizontal, colocando el elemento EditText encima del Button:

```
<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:layout_marginTop="@dimen/activity_vertical_margin"
   android:orientation="horizontal">

   <EditText
```

```
      …
   <Button
      …
</LinearLayout>
```

5. Haga los siguientes cambios a los elementos EditText y Button:
    a. Remueva los siguientes atributos del elemento EditText:

    - `android:layout_toLeftOf`
    - `android:layout_alignParentBottom`

    b. Remueva los siguientes atributos del elemento Button:

    - `android:layout_alignParentRight`
    - `android:layout_alignParentBottom`

    c. Cambie otros tres atributos del elemento EditText como sigue:

Atributo de EditText	Valor
android:layout_width	"wrap_content"
android:inputType	"phone"
android:hint	"Enter phone number"

6. Agregue un elemento Spinner entre los elemento EditText y Button:

```
<Spinner
   android:id="@+id/label_spinner"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content">
</Spinner>
```

El elemento Spinner proporciona la lista desplegable. En la siguiente tarea se agregará el código que llenará la lista del spinner con sus valores. El código del layout para los elementos EditText, Spinner y Button dentro de LinearLayout debería tener el siguiente aspecto:

```
<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:layout_marginTop="@dimen/activity_vertical_margin"
   android:orientation="horizontal">

   <EditText
      android:id="@+id/editText_main"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:inputType="phone"
```

```
      android:hint="Enter phone number" />

   <Spinner
      android:id="@+id/label_spinner"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content">
   </Spinner>

   <Button
      android:id="@+id/button_main"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:onClick="showText"
      android:text="Show" />

</LinearLayout>
```

7. Añada otro LinearLayout debajo del LinearLayout que acaba de crear, con una orientación horizontal para contener dos elementos TextView uno al lado del otro - una descripción de texto y un campo de texto para mostrar el número de teléfono y la etiqueta del teléfono - y alinee el LinearLayout con la parte inferior del padre (consulte la figura anterior):

```
<LinearLayout
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:orientation="horizontal"
   android:layout_alignParentBottom="true">

   <TextView
      …
   <TextView
      …
</LinearLayout>
```

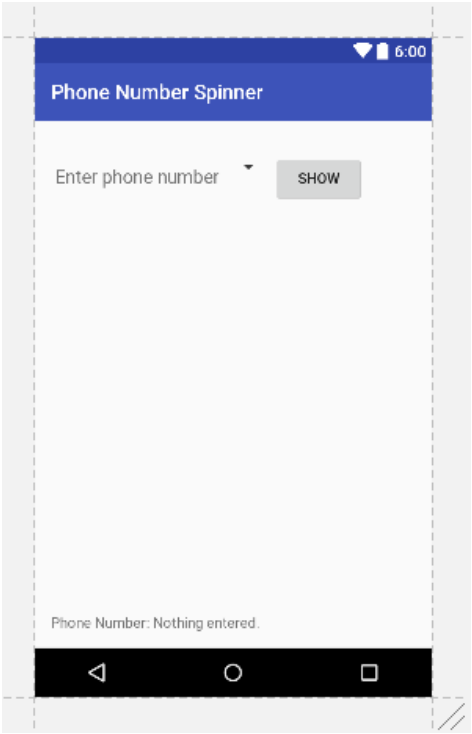
8. Agregue los siguientes atributos a los elementos TextView dentro del LInearLayout:

Atributo del TextView	**Valor**
android:id	"@+id/title_phonelabel"
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:text	"Phone Number: "

Atributo del TextView	**Valor**
android:id	"@+id/text_phonelabel"
android:layout_width	"wrap_content"

android:layout_height	"wrap_content"
android:text	"Nothing entered."

9. Verifique su diseño haciendo clic en la pestaña Preview del lado derecho de la ventana layout



Ahora debería tener una pantalla (consulte la figura anterior) con un campo de entrada de teléfono en la parte superior izquierda, el símbolo del spinner junto al campo y el botón **Show** a la derecha. En la parte inferior debe aparece el texto "Phone Number:" seguido de "Nothing entered."

10. Extraiga los string en recursos de string: Coloque el cursos en el string cableado, presione Alt-Enter (Option-Enter en Mac), y seleccione **Extract string resources**. Luego edite el nombre del recurso con el valor del string. Extraiga como se muestra a continuación:

Elemento	**String**	**String resource**
EditText	"Enter phone number"	"@string/hint_phonenumber"
Button	"Show"	"@string/show_button"
TextView	"Phone Number: "	"@string/phonenumber_label"
TextView	"Nothing entered."	"@string/nothing_entered"

3.2 Agregar el código para activar el spinner y su listener

Las opciones para este spinner de etiquetas de teléfono son strings estáticas bien definidas ("Home", "Work", etc), por lo que puede utilizar un arreglo de texto definido en strings.xml para mantener los valores de la misma.

Para activar el spinner y su listener, implemente la interfaz `AdapterView.OnItemSelectedListener`, que requiere también añadir los métodos callback `onItemSelected()` y `onNothingSelected()`.

1. Abra el archivo strings.xml para definir los valores seleccionables (**Home**, **Work**, **Mobile** y **Other**) del spinner como un arreglo de string llamado `labels_array`:

```
<string-array name="labels_array">
        <item>Home</item>
        <item>Work</item>
        <item>Mobile</item>
        <item>Other</item>
</string-array>
```

2. Para definir el callback de selección del spinner, cambie la clase `MainActivity` para que implemente la interface `AdapterView.OnItemSelectedListener` como se muestra a continuación:

```
public class MainActivity extends AppCompatActivity implements
            AdapterView.OnItemSelectedListener {
```

   A medida que escribe **AdapterView**, en la declaración anterior, Android Studio importa automáticamente el widget `AdapterView`. La razón por la que necesita el `AdapterView` es porque necesita un adaptador -específicamente un ArrayAdapter- para asignar el arreglo al spinner. Un *adaptador* conecta los datos (en este caso, el arreglo con los elementos del spinner) con la vista del spinner. Aprenderá más sobre este patrón usando un adaptador para conectar datos en otra lección. Esta línea debería aparecer en su bloque de declaraciones de importación:

```
import android.widget.AdapterView;
```

   Después de escribir **OnItemSelectedListener** en la sentencia anterior, espere unos segundos para que aparezca una señal de *bombillo rojo* en el margen izquierdo.

3. Haga clic en el bombillo y seleccione **Implement methods**. Los métodos `onItemSelected()` y `onNothingSelected()`, que son requeridos por el `OnItemSelectedListener`, ya deben estar resaltados, y la opción "Insert @Override" debe estar marcada. Haga clic en **OK**.

   Este paso añade automáticamente los métodos callback vacíos `onItemSelected()` y `onNothingSelected()` a la parte inferior de la clase `MainActivity`. Ambos métodos utilizan el parámetro `AdapterView<?>`. El simbolo `<?>` es un comodín de tipo Java, lo que permite que el

método sea lo suficientemente flexible como para aceptar cualquier tipo de `AdapterView` como argumento.

4. Instancie un objeto spinner en el método `onCreate()` usando el elemento Spinner en el layout (`label_spinner`), y establezca su oyente (`spinner.setOnItemSelectedListener`) en el método `onCreate()`. Añada el código al método `onCreate()`:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   ...
   // Create the spinner.
   Spinner spinner = (Spinner) findViewById(R.id.label_spinner);
   if (spinner != null) {
            spinner.setOnItemSelectedListener(this);
   }
   ...
```

5. Siguiendo con la edición del método `onCreate()`, añada una sentencia que cree el ArrayAdapter con el arreglo de strings (`labels_array`) usando el diseño simple del spinner suministrado por Android para cada elemento (`layout.simple_spinner_item`):

```
...
// Create ArrayAdapter using the string array and default spinner layout.
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
            R.array.labels_array, android.R.layout.simple_spinner_item);
...
```

   El layout del elemento `simple_spinner_item` utilizado en este paso, y el layout del elemento `simple_spinner_dropdown_item` utilizado en el siguiente paso, son las disposiciones predefinidas por defecto proporcionadas por Android en la clase R.layout. Debería utilizar estos layouts a menos que desee definir sus propios layouts para los elementos y la apariencia del spinner.

6. Especifique el diseño para que las opciones del spinner sean `simple_spinner_dropdown_item` y a continuación, aplique el adaptador al spinner:

```
   ...
   // Specify the layout to use when the list of choices appears.
   adapter.setDropDownViewResource
                       (android.R.layout.simple_spinner_dropdown_item);
   // Apply the adapter to the spinner.
   if (spinner != null) {
            spinner.setAdapter(adapter);
   }
   ...
```

3.3 Agregar el código para que responda a la selección del usuario

Cuando el usuario selecciona un elemento en el spinner, el objeto Spinner recibe un evento `on-item-selected`. Para manejar este evento, ya se ha implementado en el paso anterior la interfaz `AdapterView.OnItemSelectedListener`, añadiendo métodos callback vacíos `onItemSelected()` y `onNothingSelected()`.

En este paso primero declarará `mSpinnerLabel` como el string que contiene el elemento seleccionado. Luego rellenará el código del método `onItemSelected()` para recuperar el elemento seleccionado en el spinner, usando `getItemAtPosition()`, y lo asignará a `mSpinnerLabel`:

1. Declare el string `mSpinnerLabel` al comienzo de la definición de la clase `MainActivity`:

```
public class MainActivity extends AppCompatActivity implements
              AdapterView.OnItemSelectedListener {
 private String mSpinnerLabel = "";
 ...
}
```

2. Complete el código del método callback `onItemSelected()`, como se muestra abajo, para recuperar el ítem seleccionado por el usuario usando el `getItemAtPosition`, y asígnelo al `mSpinnerLabel`. También puede añadir una llamada al método `showText()` que ya ha sido añadido a la versión anterior de la aplicación:

```
public void onItemSelected(AdapterView<?> adapterView, View view, int
              i, long l) {
   mSpinnerLabel = adapterView.getItemAtPosition(i).toString();
   showText(view);
}
```

   Sugerencia: Al añadir el método showText() al método anterior `onItemSelected()`, se ha habilitado el listener de selección del spinner para que muestre la selección junto con el número de teléfono, de modo que ya no necesita el botón **Show** que llamaba al método `showText()`.

3. Añada código al método callback vacío `onNothingSelected()`, como se muestra a continuación, para mostrar un mensaje logcat si no se ha seleccionado nada:

```
public void onNothingSelected(AdapterView<?> adapterView) {
    Log.d(TAG, "onNothingSelected: ");
}
```

La variable TAG en la sentencia anterior está en rojo debido a que no ha sido definida.

4. Extraiga el recurso de string para "`onNothingSelected: `" para `nothing_selected`.
5. Haga clic en **TAG**, luego haga clic en el bombillo rojo y seleccione **Create constant field 'TAG'** en el menú emergente. Android Studio añade lo siguiente en la declaración de la clase MainActivity:

```
private static final String TAG = ;
```

6. Agregar `MainActivity.class.getSimpleName()` para usar el nombre simple de la clase para TAG:

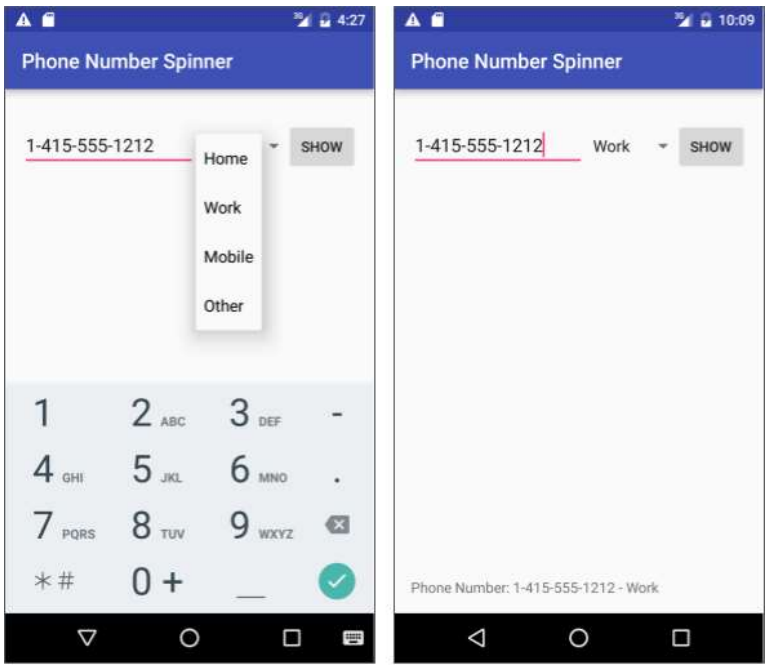
```
private static final String TAG = MainActivity.class.getSimpleName();
```

7. Cambiar la sentencia `String showString` en el método `showText` para que muestre tanto el string introducido como el elemento seleccionado del spinner (mSpinnerLabel):

```
String showString = (editText.getText().toString() + " - " + mSpinnerLabel);
```

8. Ejecute la aplicación,

El spinner aparece junto al campo de entrada del teléfono y muestra la primera opción (Home). Al tocar el spinner este revela todas las opciones, tal como se muestra en el lado izquierdo de la figura de abajo. Después de introducir un número de teléfono y elegir un elemento del spinner, aparece un mensaje en la parte inferior de la pantalla con el número de teléfono y el elemento seleccionado del spinner, como se muestra en la parte derecha de la figura de abajo. (También puede tocar en el botón **Show** para mostrar tanto el número de teléfono como el elemento del spinner, pero como esto es redundante, ahora puede quitar el botón **Show**.



Tarea 4. Usar un dialogo para una alerta que requiere de alguna decisión

Puede proporcionar un cuadro de diálogo para una alerta que requiera que los usuarios tomen una decisión. Un *dialog* es una ventana que aparece en la parte superior de la pantalla o la llena, interrumpiendo el flujo de la actividad.

Por ejemplo, un diálogo de alerta puede requerir que el usuario haga clic en **Continue** después de leerlo, o darle al usuario la opción de aceptar una acción haciendo clic en un botón positivo (como **OK** o **Accept**), o en desacuerdo haciendo clic en un botón negativo (como **Cancel**). En Android, se utiliza la subclase AlertDialog de la clase Dialog para mostrar un cuadro de diálogo estándar para una alerta.

Sugerencia: Utilice los diálogos con moderación ya que interrumpen el flujo de trabajo del usuario. Lea la guía de diseño de Dialogs para conocer las mejores prácticas de diseño y Dialogs en la documentación del desarrollador de Android para ver ejemplos de código.

En esta práctica, utilizará un botón para activar un cuadro de diálogo de alerta estándar. En una aplicación del mundo real, es posible que desencadene un cuadro de diálogo de alerta basado en alguna condición o en que el usuario toque algo.

4.1 Crear un proyecto nuevo con un layout para mostrar un diálogo de alerta.

En este ejercicio, creará una alerta con los botones OK y Cancel, la cual se activará cuando el usuario haga clic en uno de esos botones.

1. Cree un nuevo proyecto llamado **Alert Sample** basado en la plantilla Empty Activity (Actividad vacía).
2. Abra el archivo de layout activity_main.xml. En el Editor de diseño, haga clic en la pestaña Text en la parte inferior de la pantalla y cambie el grupo de vista raíz como RelativeLayout, como lo ha hecho en ejercicios anteriores.
3. Si el elemento TextView incluye algún atributo layout-constraint, elimínelo.
4. Haga los siguientes cambios en el TextView:

Atributo TextView	Valor
android:id	"@+id/top_message"
android:text	"Tap to test the alert:"

5. Extraiga la cadena `android:text` de arriba en el recurso `tap_test` para que sea más fácil de traducir.
6. Agregar un botón con los siguientes atributos:

Atributo del Button	Valor
android:id	"@+button1"
android:layout_width	wrap_content
android:layout_height	wrap_content
android:layout_below	"@id/top_message"

android:layout_marginTop	"36dp"
android:text	"Alert"
android:onClick	"onClickShowAlert"

7. Extraiga el string `android:text` de arriba en el recurso `alert_button` para que sea más fácil de traducir.
8. Extraiga el valor de dimensión para `android:layout_marginTop` de la misma manera: Coloque el cursor en "`36dp`", pulse Alt-Enter (Opción-Enter en el Mac), y seleccione **Extract dimension resource**. A continuación, edite el Nombre del recurso para que el valor sea **button_top_margin**.

   Las asignaciones de recursos de dimensión se almacenan en el archivo dimens.xml (en **app > res > values > dimens**). Puede editar este archivo para cambiar las asignaciones de modo que la aplicación pueda cambiar de acuerdo a diferentes tamaños de pantalla.

4.2 Agregar un dialogo de alerta a la actividad prinicipal

El patrón de diseño *builder* facilita la creación de un objeto a partir de una clase que tiene muchos atributos obligatorios y opcionales y que, por lo tanto, necesitaría muchos parámetros para construirse. Sin este patrón, usted tendría que crear constructores para combinaciones de atributos requeridos y opcionales; con este patrón, el código es más fácil de leer y mantener. Para obtener más información sobre el patrón de diseño del constructor, consulte Patrón builder.

La clase builder suele ser un miembro estático de la clase que construye (es decir, una clase interna). Puede utilizar AlertDialog.Builder para crear un diálogo de alerta estándar, utilizando `setTitle` para darle el título, `setMessage` para definir el mensaje, y `setPositiveButton` y `setNegativeButton` para definir los botones.

Para hacer la alerta, se necesita instanciar un objeto de AlertDialog.Builder. Para ello debe añadir el método `onClickShowAlert()`, lo que hace convierte que este objeto sea la primera prioridad.

Nota: Para que este ejemplo sea fácil de entender, el diálogo de alerta se crea en el método `onClickShowAlert()`. Esto ocurre sólo si se llama al método `onClickShowAlert()`, que es lo que ocurre cuando el usuario hace clic en el botón. Esto significa que la aplicación sólo construirá un nuevo cuadro de diálogo cuando se haga clic en el botón, lo que es útil si el cuadro de diálogo se ve sólo en raras ocasiones (cuando el usuario toma una determinada ruta a través de la aplicación). Sin embargo, si el diálogo debe aparecer a menudo, es posible que sea mejor construir el objeto diálogo sólo una vez en el método `onCreate()`, y luego invocarlo en el método `onClickShowAlert()`.

1. Agregue el método `onClickShowAlert()` a la clase `MainActivity.java` como sigue:

```
public void onClickShowAlert(View view) {
  AlertDialog.Builder myAlertBuilder = new
                AlertDialog.Builder(MainActivity.this);
```

Nota: Si AlertDialog.Builder no es reconocido al introducirlo, haga clic en el icono del bombillo rojo y elija la versión de la biblioteca de soporte (**android.support.v7.app.AlertDialog**) para importarla a la actividad.

2. Configure el título y el mensaje para el diálogo de alerta dentro de `onClickShowAlert()` después del código del paso anterior:

```
...
// Set the dialog title.
myAlertBuilder.setTitle("Alert");
// Set the dialog message.
myAlertBuilder.setMessage("Click OK to continue, or Cancel to stop:");
...
```

3. Extraiga el título y el mensaje en los recursos de string. Las líneas de código anteriores deberían ser ahora así:

```
...
// Set the dialog title.
myAlertBuilder.setTitle(R.string.alert_title);
// Set the dialog message.
myAlertBuilder.setMessage(R.string.alert_message);
...
```

4. Añada el botón **OK** a la alerta con `setPositiveButton()` y usando el `onClickListener()`:

```
...
// Add the buttons.
myAlertBuilder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
     public void onClick(DialogInterface dialog, int which) {
          // User clicked OK button.
          Toast.makeText(getApplicationContext(), "Pressed OK",
                  Toast.LENGTH_SHORT).show();
     }
});
...
```

Los botones positivos (**OK**) y negativos (**Cancel**) se configuran utilizando los métodos `setPositiveButton()` y `setNegativeButton()`. Después de que el usuario toque el botón **OK** en la alerta, puede tomar la selección del usuario y usarla en su código. En este ejemplo, se mostrará un mensaje toast si se hace clic en el botón **OK**.

5. Extraiga el recurso de string para "`OK`" y para "`Pressed OK`". La declaración debería ser ahora:

```
...
```

```
// Add the buttons.
myAlertBuilder.setPositiveButton(R.string.ok, new
                      DialogInterface.OnClickListener() {
     public void onClick(DialogInterface dialog, int which) {
         // User clicked OK button.
         Toast.makeText(getApplicationContext(), R.string.pressed_ok,
                      Toast.LENGTH_SHORT).show();
     }
});
...
```

6. Añada el botón **Cancel** a la alerta con `setNegativeButton()` y `onClickListener()`, muestre un mensaje tost si se hace clic en el botón, y luego cancele el diálogo:

```
...
myAlertBuilder.setNegativeButton("Cancel", new
                       DialogInterface.OnClickListener() {
     public void onClick(DialogInterface dialog, int which) {
          // User cancelled the dialog.
          Toast.makeText(getApplicationContext(), "Pressed Cancel",
                       Toast.LENGTH_SHORT).show();
     }
});
...
```

7. Extraiga el recurso de string para "`Cancel`" y "`Pressed Cancel`". La declaración debería ser ahora:

```
...
myAlertBuilder.setNegativeButton(R.string.cancel, new
                       DialogInterface.OnClickListener() {
     public void onClick(DialogInterface dialog, int which) {
         // User cancelled the dialog.
         Toast.makeText(getApplicationContext(), R.string.pressed_cancel,
                       Toast.LENGTH_SHORT).show();
     }
});
...
```

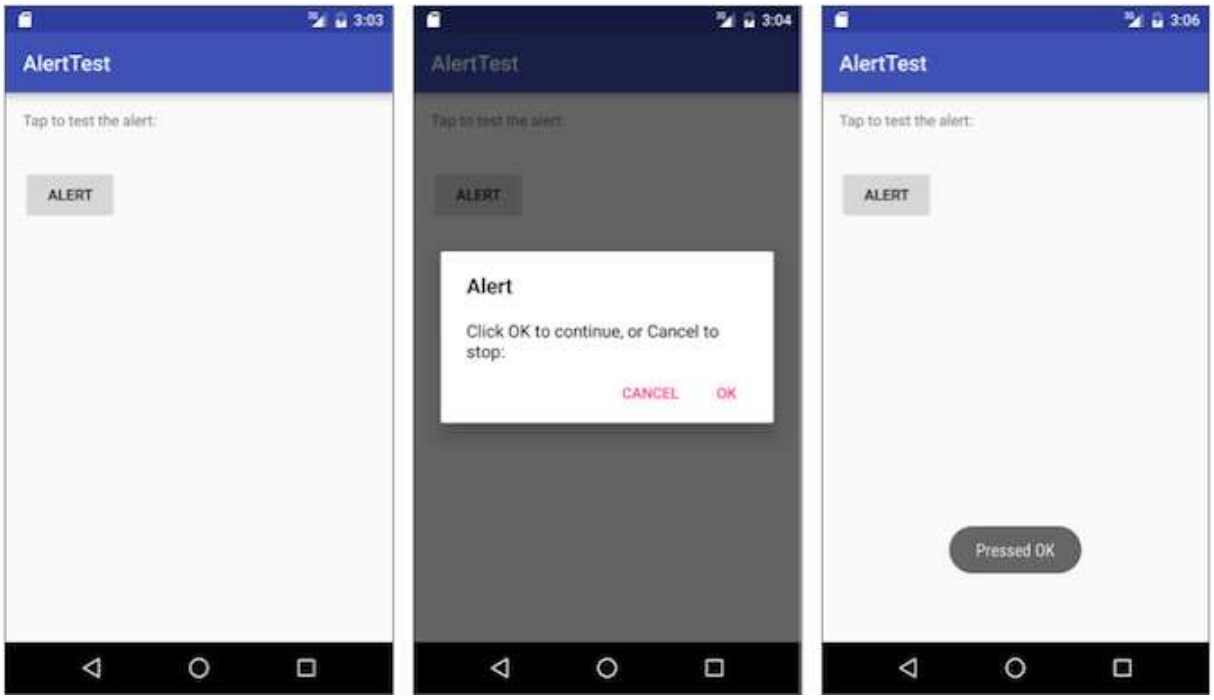
8. Añada `show()`, el cual crea y luego muestra el diálogo de alerta, al final de `onClickShowAlert()`:

```
  ...
  // Create and show the AlertDialog.
  myAlertBuilder.show();
}
```

Sugerencia: Para obtener más información sobre onClickListener y otros listeners, consulte User Interface: Input Events

9. Ejecute la aplicación

Debería poder tocar el botón **Alert**, que se muestra en el lado izquierdo de la figura de abajo, para ver el diálogo de alerta, que se muestra en el centro de la figura de abajo. El cuadro de diálogo muestra los botones **OK** y **Cancel**, y aparece un mensaje toast que muestra cuál presionó, como se muestra en el lado derecho de la figura de abajo.



Tarea 5. Usar un picker como entrada de usuario

Android proporciona diálogos listos para usar, llamados *pickers*, para elegir una hora o una fecha. Puede utilizarlos para asegurarse de que los usuarios eligan una hora o fecha válida que esté también formateada correctamente y ajustada a la hora y fecha locales del usuario. Cada picker proporciona controles para seleccionar cada parte de la hora (hora, minuto, AM/PM) o fecha (mes, día, año). Puede leer todo sobre la configuración de los responsables de picking en Pickers.

En esta tarea creará un nuevo proyecto y agregará un picker de fecha y un picker de hora. También aprenderá a usar los fragmentos. Un fragmento es un comportamiento o una porción de la interfaz de usuario dentro de una actividad. Es como una mini-actividad dentro de la actividad principal, con su propio ciclo de vida, y se utiliza para construir un picker. Todo el trabajo está ya listo para el desarrolador. Para obtener más información acerca de los fragmentos, consulte Fragments en la Guía de la API.

Una ventaja del uso de fragmentos para los pickers es que se pueden aislar las secciones de código para gestionar la fecha y la hora para varias localizaciones que muestran la fecha y la hora de diferentes maneras. La mejor práctica para mostrar un picker es usar una instancia de DialogFragment, que es una subclase de Fragment. Un DialogFragment muestra una ventana de

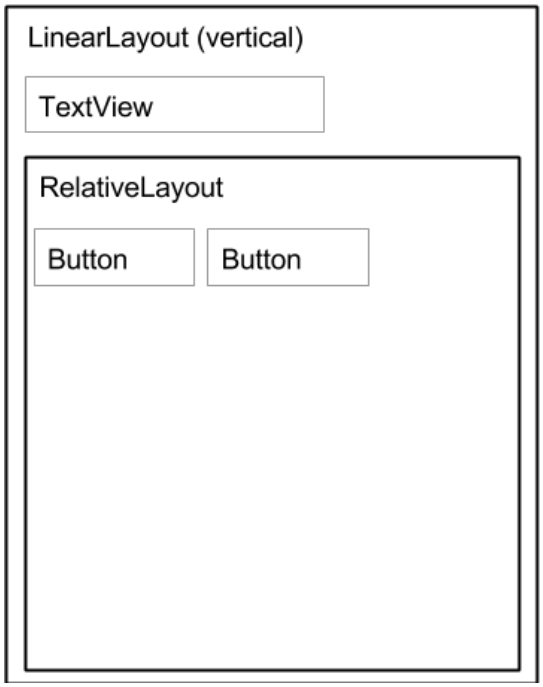
diálogo flotando sobre la ventana de la actividad. En este ejercicio, agregará un fragmento para cada diálogo del picker y utilizará DialogFragment para administrar el ciclo de vida del diálogo.

Sugerencia: Otra ventaja de utilizar fragmentos para los pickers es que puede implementar diferentes configuraciones de layout, como un cuadro de diálogo básico en pantallas del tamaño de un teléfono o una parte incrustada de un diseño en pantallas grandes.

5.1 Crear el diseño de la actividad principal

Para iniciar esta tarea, creará el layout de la actividad principal para proporcionar botones para acceder a los pickers de fecha y hora. Consulte el código de layout XML a continuación:

1. Inicie un nuevo proyecto llamado **Date Time Pickers** usando la plantilla Empty Activity (Actividad vacía).
2. Abra el archivo activity_main.xml para editar el código de diseño.
3. Cambie el layout a LinearLayout y añada `android:orientation="vertical"` para orientar el diseño verticalmente. No se preocupe por la apariencia del diseño todavía. El objetivo es utilizar un diseño que integre un RelativeLayout dentro del LinearLayout:



4. Cambie el texto del primer elemento TextView a "**Choose the date and time:**" (Elija la fecha y la hora:) y extraiga el texto al recurso de string `choose_datetime`.

Atributo del TextView	Valor anterior	Valor nuevo
android:text	"Hello World"	"@string/choose_datetime"

5. Agregur el atributo `android:textSize` e introduzca un tamaño de texto de **20sp**. Extraiga el `android:textSize` dimension en `text_size`.

Atributo del TextView	Valor anterior	Valor nuevo
android:textSize		"@dimen/text_size"

6. Añada un RelativeLayout hijo dentro de LinearLayout para contener los elementos Button, y acepte el ancho y alto por defecto que es "`match parent`".
7. Añada el primer elemento Button dentro del RelativeLayout con los siguientes atributos:

Atributos del Primer Button	Valor
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:id	"@+id/button_date"
android:layout_marginTop	"12dp"
android:text	"Date"
android:onClick	"showDatePickerDialog"

   No se preocupe si la referencia `showDatePickerDialog` está en rojo. El método no se ha definido todavía, sino que se definirá más tarde.

8. Extraiga el string "`Date`" en el recurso de string `date_button`.
9. Extraiga la dimensión "`12dp`" para `android:layout_marginTop` del `button_top_margin`.
10. Añada el segundo elemento Button dentro del RelativeLayout hijo con los siguientes atributos:

Atributos del Segundo Button	Valor
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:id	"@+id/button_time"
android:layout_marginTop	"@dimen/button_top_margin"
android:layout_alignBottom	"@id/button_date"
android:layout_toRightOf	"@id/button_date"
android:text	"Time"
android:onClick	"showTimePickerDialog"

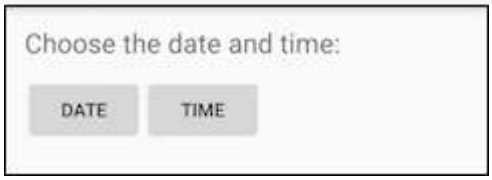
    La referencia `showTimePickerDialog` está en rojo. El método no ha sido definido todavía - se definirá más tarde.

11. Extraiga el string "`Time`" en el recurso de string `time_button`.
12. Si aún no lo ha hecho, haga clic en la pestaña **Preview** para mostrar una vista previa de la presentación. Debería verse como el código y la figura de abajo.

Código de solución para el layout principal:

Dependiendo de su versión de Android Studio, su código tendrá un aspecto parecido al siguiente.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.android.DateTimePickers.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="@dimen/text_size"
        android:text="@string/choose_datetime"/>

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/button_date"
            android:layout_marginTop="@dimen/button_top_margin"
            android:text="@string/date_button"
            android:onClick="showDatePickerDialog"/>

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/button_time"
            android:layout_marginTop="@dimen/button_top_margin"
            android:layout_alignBottom="@id/button_date"
            android:layout_toRightOf="@id/button_date"
            android:text="@string/time_button"
            android:onClick="showTimePickerDialog"/>

    </RelativeLayout>
</LinearLayout>
```



5.2 Crear un nuevo fragmento para el picker de fecha

En este ejercicio, agregará un fragmento para el picker de fechas. Un fragmento es como una mini-actividad dentro de la actividad principal, con su propio ciclo de vida.

1. Expanda **app > java > com.example.android.DateTimePickers** y seleccione **MainActivity**.
2. Seleccione **File > New > Fragment > Fragment (Blank)** y nombre el fragmento DatePickerFragment. Desmarque las tres opciones de la casilla de verificación de modo que *no cree* un layout XML, *no* incluya los métodos factory de los fragmentos y *no* incluya los callbacks de la interfaz. No es necesario crear un layout para un picker estándar. Haga clic en **Finish** para crear el fragmento.
3. Abra el `DatePickerFragment` y edite la definición de la clase DatePickerFragment para que extienda a DialogFragment e implemente `DatePickerDialog.OnDateSetListener` para crear un picker de fecha estándar con un listener. Consulte Picker para obtener más información sobre extender DialogFragment para un picker de fecha:

```
public class DatePickerFragment extends DialogFragment
                implements DatePickerDialog.OnDateSetListener {
```

   Mientras escribe **DialogFragment** y **DatePickerDialog.OnDateSetListener**, Android Studio añade automáticamente lo siguiente en el bloque `import` de la parte superior:

```
import android.app.DatePickerDialog;
import android.support.v4.app.DialogFragment;
```

   Además, un icono de bombillo rojo aparece en el margen izquierdo después de unos segundos.

4. Haga clic en el icono del bombill rojo y seleccione **Implement methods** en el menú emergente. Aparece un diálogo con `onDateSet()` ya seleccionado y la opción "Insert @Override" seleccionada. Haga clic en **OK** para crear el método vacío `onDateSet()`. Este método se llamará cuando el usuario fije la fecha. Después de añadir el método `onDateSet()` vacío, Android Studio añade automáticamente lo siguiente en el bloque `import` de la parte superior:

```
import android.widget.DatePicker;
```

   Los parámetros del método `onDateSet()` deben ser `int year`, `int month` e `int dayOfMonth`. Cambie el parámetro `dayOfMonth` por `day` para mayor brevedad:

```
public void onDateSet(DatePicker view, int year, int month, int day)
```

5. Elimine el constructor público vacío del DatePickerFragment.
6. Reemplace `onCreateView()` con `onCreateDialog()` que devuelve un `Dialog`, y el método `onCreateDialog()` anótelo con `@NonNull` para indicar que el valor de retorno `Dialog` no puede ser nulo - cualquier intento de referirse al valor de retorno `Dialog` se debe chequear si es nulo.

```
  @NonNull
  @Override
  public Dialog onCreateDialog(Bundle savedInstanceState) {
      ...
  }
```

7. Añada el siguiente código al `onCreateDialog()` para inicializar el `year`, `month` y `day` desde el `Calendar`, y devuelva el diálogo y estos valores a la actividad principal. A medida que introduce `Calendar`, especifique que la importación sea `java.util.Calendar`.

```
  @NonNull
  @Override
  public Dialog onCreateDialog(Bundle savedInstanceState) {
     // Use the current date as the default date in the picker.
     final Calendar c = Calendar.getInstance();
     int year = c.get(Calendar.YEAR);
     int month = c.get(Calendar.MONTH);
     int day = c.get(Calendar.DAY_OF_MONTH);

     // Create a new instance of DatePickerDialog and return it.
     return new DatePickerDialog(getActivity(), this, year, month, day);
  }
```

Código de solución para el DatePickerFragment:

```
public class DatePickerFragment extends DialogFragment
        implements DatePickerDialog.OnDateSetListener {

    @NonNull
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current date as the default date in the picker.
        final Calendar c = Calendar.getInstance();
        int year = c.get(Calendar.YEAR);
        int month = c.get(Calendar.MONTH);
        int day = c.get(Calendar.DAY_OF_MONTH);

        // Create a new instance of DatePickerDialog and return it.
        return new DatePickerDialog(getActivity(), this, year, month, day);
    }

    public void onDateSet(DatePicker view, int year, int month, int day) {
        // Do something with the date chosen by the user.
    }

}
```

5.5 Crear un Nuevo fragment para el picker de hora

Agregue un fragmento al proyecto DateTimePickers para el selector de hora:

1 Seleccione nuevamente el **MainActivity**
2 Seleccione **File > New > Fragment > Fragment (Blank)** y nombre el fragmento TimePickerFragment. Desmarque las tres opciones para no crear un layout XML, *no* incluya los métodos fragment factory y *no* incluya las interfaces callback. Haga clic en **Finish** para crear el fragmento.
3 Abra el TimePickerFragment y siga los mismos procedimientos que con el DatePickerFragment, implementando el método vacío `onTimeSet()`, reemplazando `onCreateView()` con `onCreateDialog()`, y eliminando el constructor público vacío del TimePickerFragment. TimePickerFragment realiza las mismas tareas que DatePickerFragment, pero con valores de tiempo:
   - Extiende DialogFragment e implementa `TimePickerDialog.OnTimeSetListener` para crear un selector de tiempo estándar con un listener. Vea Picker para más información sobre la extensión de DialogFragment para un selector de hora.
   - Utiliza el método `onCreateDialog()` para inicializar `hour` y `minute` del `Calendar`, y devuelve el diálogo y estos valores a la actividad principal utilizando el formato de 24 horas. A manera que escribe `Calendar`, especifique que la importación sea `java.util.Calendar`.
   - También define el método vacío `onTimeSet()` para que sea agreguado el código para usar el `hourOfDay` y `minute` que el usuario seleccione. Este método será llamado cuando el usuario configure la hora:

```
public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
    // Do something with the time chosen by the user.
}
```

Nota: Al realizar los cambios, Android Studio añade automáticamente lo siguiente en el bloque de importación de la parte superior:

```
import android.support.v4.app.DialogFragment;

import android.app.TimePickerDialog;

import android.widget.TimePicker;

import java.util.Calendar;
```

Código para la solución del TimePickerFragment:

```
public class TimePickerFragment extends DialogFragment
        implements TimePickerDialog.OnTimeSetListener {

    @NonNull
```

```
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current time as the default values for the picker.
        final Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        int minute = c.get(Calendar.MINUTE);

        // Create a new instance of TimePickerDialog and return it.
        return new TimePickerDialog(getActivity(), this, hour, minute,
                DateFormat.is24HourFormat(getActivity()));
    }

    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
        // Do something with the time chosen by the user.
    }
}
```

5.4 Modificar la actividad principal

Aunque gran parte del código en la actividad principal permanece igual, necesitará agregar métodos que creen instancias del FragmentManager para administrar cada fragmento y mostrar cada selector.

1. Cree el recurso de string en strings.xml:

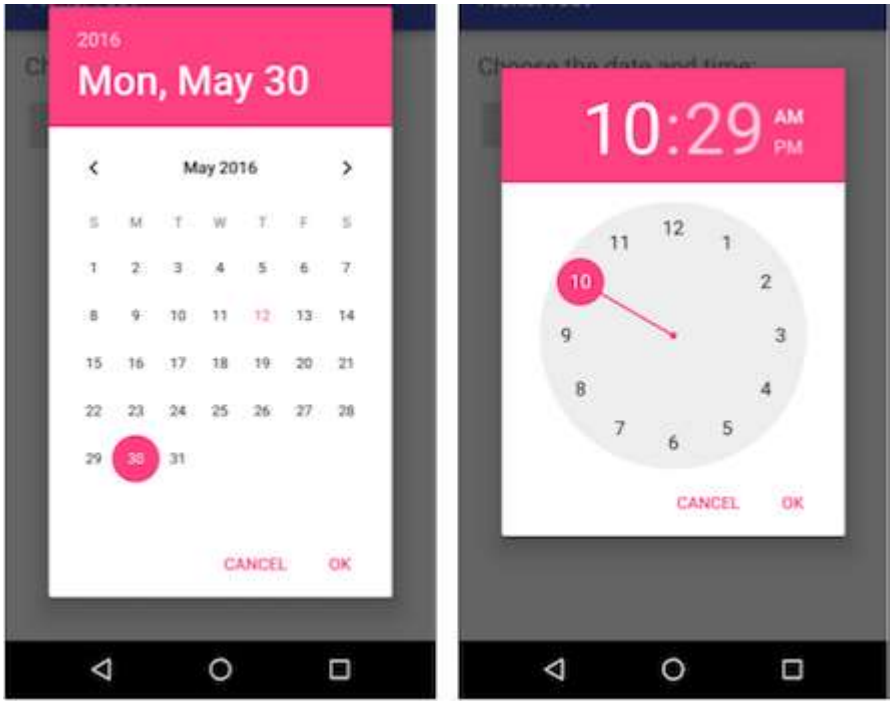
```
<string name="date_picker">datePicker</string>
<string name="time_picker">timePicker</string>
```

2. Abra **MainActivity**
3. Añada los métodos `showDatePickerDialog()` y `showTimePickerDialog()`, haciendo referencia al código de abajo. Crea una instancia del FragmentManager para administrar el fragment automáticamente, y para mostrar el picker. Para obtener más información sobre los fragments, véase Fragments.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void showDatePickerDialog(View v) {
        DialogFragment newFragment = new DatePickerFragment();
        newFragment.show(getSupportFragmentManager(),
                    getString(R.string.date_picker));
    }

    public void showTimePickerDialog(View view) {
        DialogFragment newFragment = new TimePickerFragment();
        newFragment.show(getSupportFragmentManager(),
                    getString(R.string.time_picker));
```

```
    }
}
```

4. Ejecute la aplicación. Deb ever los pickers de fecha y hora después de tocar los botones.



5.5 Usando la fecha y hora seleccionadas

En este ejercicio pasará la fecha y la hora al MainActivity, y convertirá la fecha y la hora en strings que puede mostrar en un mensaje toast.

1. Abra **MainActivity** y agregue la firma del método `processDatePickerResult()` que toma `year`, `month` y `day` como argumentos:

```
public void processDatePickerResult(int year, int month, int day) {
}
```

2. Añada el siguiente código al método `processDatePickerResult()` para convertir `year`, `month` y `day` en cadenas separadas:

```
String month_string = Integer.toString(month+1);
String day_string = Integer.toString(day);
String year_string = Integer.toString(year);
```

Sugerencia: El número entero del mes devuelto por el picker de la fecha comienza a contar desde 0 para enero, por lo que necesita agregarle 1 para comenzar a mostrar los meses a partir de 1.

3. Añada lo siguiente después del código anterior para concatenar los tres strings e incluir marcas de barra oblicua para el formato de fecha de Español:

```
String dateMessage = (day_string + "/" +
                           month_string + "/" + year_string);
```

4. Añada lo siguiente después de la declaración anterior para mostrar un mensaje toast:

```
Toast.makeText(this, "Date: " + dateMessage,
                           Toast.LENGTH_SHORT).show();
```

5. Extraiga el string "Date:" en un recurso de string llamado `date`. Esto reemplaza automáticamente el string cableado con `getString(R.string.date)`. El código para el método `processDatePickerResult()` debería verse así:

```
public void processDatePickerResult(int year, int month, int day) {
   String month_string = Integer.toString(month + 1);
   String day_string = Integer.toString(day);
   String year_string = Integer.toString(year);
   // Assign the concatenated strings to dateMessage.
   String dateMessage = (day_string + "/" +
                           month_string + "/" + year_string);
   Toast.makeText(this, getString(R.string.date) + dateMessage,
                                Toast.LENGTH_SHORT).show();
}
```

6. Abra el DatePickerFragment, y agregue lo siguiente al método `onDateSet()` para invocar el método `processDatePickerResult()` en `MainActivity` y pase el `year`, `month` y `day`:

```
public void onDateSet(DatePicker view, int year, int month, int day) {
   // Set the activity to the Main Activity.
   MainActivity activity = (MainActivity) getActivity();
   // Invoke Main Activity's processDatePickerResult() method.
   activity.processDatePickerResult(year, month, day);
}
```

Se utiliza `getActivity()` que, cuando es utilizado en un fragment, devuelve la actividad a la que el fragment está asociado actualmente. Necesita esto porque no puede llamar a un método en `MainActivity` sin el contexto del `MainActivity` (tendría que usar un intent en su lugar, como aprendió en una lección anterior). La actividad hereda el contexto, por lo que puede utilizarlo como contexto para llamar al método (como en `activity.processDatePickerResult`).

7. El TimePickerFragment utiliza la misma lógica. Abra **MainActivity** y añada la firma del método `processTimePickerResult()` que toma como argumentos `hourOfDay` y el `minute`:

```
public void processTimePickerResult(int hourOfDay, int minute) {
```

```
}
```

8. Añada el siguiente código al método `processTimePickerResult()` para convertir `hourOfDay` y `minute` en strings separados:

```
String hour_string = Integer.toString(hourOfDay);
String minute_string = Integer.toString(minute);
```

9. Añada lo siguiente después del código anterior para concatenar las cadenas e incluya dos puntos para el formato de hora:

```
String timeMessage = (hour_string + ":" + minute_string);
```

10. Añada lo siguiente después de la declaración anterior para mostrar un mensaje toast:

```
Toast.makeText(this, "Time: " + timeMessage,
                            Toast.LENGTH_SHORT).show();
```

11. Extraiga el string cableado "`Time: `" en un recurso de string llamado `time`. Esto reemplaza automáticamente el string cableado con `getString(R.string.time)`. El código para el método `processDatePickerResult()` debería verse así:

```
public void processTimePickerResult(int hourOfDay, int minute) {
   // Convert time elements into strings.
   String hour_string = Integer.toString(hourOfDay);
   String minute_string = Integer.toString(minute);
   // Assign the concatenated strings to timeMessage.
   String timeMessage = (hour_string + ":" + minute_string);
   Toast.makeText(this, getString(R.string.time) + timeMessage,
                                  Toast.LENGTH_SHORT).show();
}
```

12. Abra el TimePickerFragment y añada lo siguiente al método `onTimeSet()` para invocar el método `processTimePickerResult()` en `MainActivity` y pásele la `hourOfDay` y el `minute`:

```
public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
   // Set the activity to the Main Activity.
   MainActivity activity = (MainActivity) getActivity();
   // Invoke Main Activity's processTimePickerResult() method.
   activity.processTimePickerResult(hourOfDay, minute);
}
```

13. Ahora puede ejecutar la aplicación. Después de seleccionar la fecha u hora, estas aparecerán en un toast en la parte inferior, como se muestra en la siguiente figura

Tarea 6: Usando vistas de imágenes como botones

Puede hacer que una vista se comporte como un botón, añadiendo el atributo `android:onClick` en el layout XML. Por ejemplo, puede hacer que una imagen actúe como un botón añadiendo `android:onClick` a un ImageView.

Sugerencia: Si utiliza varias imágenes en las que se puede hacer clic, organícelas en un viewgroup de modo que estén agrupadas.

En esta tarea creará un prototipo de una aplicación para pedir postres en una cafetería. Después de iniciar un nuevo proyecto basado en la plantilla de Basic Activity, modificará el TextView "Hello World" con el texto apropiado, y añadirá imágenes para usar en los botones "Add to order".

6.1 Iniciar el nuevo Proyecto

1. Inicie un nuevo proyecto de Android Studio con el nombre de aplicación **Droid Cafe**. Seleccione la plantilla **Basic Activity**, acepte la configuración predeterminada y haga clic en **Finish**. El proyecto abre con dos layouts en la carpeta **res > layout**: activity_main.xml y content_main.xml.
2. Abra el archivo content_main.xml. En el Editor de diseño, haga clic en la pestaña Text en la parte inferior de la pantalla y cambie el grupo raíz a RelativeLayout, como ha hecho en ejercicios anteriores.
3. Abra el content_main.xml. Si el elemento TextView incluye algún atributo constrain-layout (restricción de diseño), elimínelo. Extraiga el string "Hello World" en el TextView para que utilice el nombre de recurso `intro_text`.
4. Abra strings.xml y redefina el recurso `intro_text` para utilizar un texto más descriptivo, como "Droid Desserts":

```
        <string id="intro_text">Postresroid </string>
```

5. Cambie el TextView en el layout para utilizar un tamaño de texto más grande: `24sp` y un padding (relleno) de `10dp`, luego añada el atributo `android:id` con el `id` como `textintro`.
6. Extraiga el recurso de dimensión para el atributo `android:padding` al nombre del recurso `padding_regular`, y el atributo `android:textSize` al nombre del recurso `text_heading`. Estos nombres de recurso se utilizarán en pasos posteriores.
7. Añada otro TextView bajo el TextView `textintro` con los siguientes atributos:

Atributos del TextView	Valor
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:padding	"@dimen/padding_regular"
android:id	"@+id/choose_dessert"
android:layout_below	"@id/textintro"
android:text	"Choose a dessert."

8. Extraiga el recurso de string para el atributo `android:text` al nombre del recurso `choose_a_dessert`.

6.2 Agregar las imágenes

1. Las imágenes llamadas donut_circle.jpg, froyo_circle.jpg, y icecream_circle.jpg se proporcionan con las aplicaciones iniciales en el archivo **Cap5_Tarea6_imagenes.rar**, que encontrará en la sección **Archivo** del **M7 de la materia**. Para copiar las imágenes en el proyecto, realice las operaciones siguientes:
   a. Cierre el proyecto.
   b. Copie los archivos de imagen en la carpeta **drawable** de su proyecto. Encuentre la carpeta **drawable** en un proyecto utilizando la siguiente ruta: *nombre_de_proyecto* **> app > src > main > res > drawable**
   c. Reabrir el proyecto.
2. Vuelva a abrir el archivo content_main.xml y añada un ImageView para la imagen de la donut al layout en la vista `choose_dessert`, utilizando los siguientes atributos:

Atributos del ImageView para donut	Valor
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:padding	"@dimen/padding_regular"
android:id	"@+id/donut"
android:layout_below	"@id/choose_dessert"
android:contentDescription	"Donuts are glazed and sprinkled with candy."
android:src	"@drawable/donut_circle"

3. Extraiga el valor del atributo `android:contentDescription` al recurso de string `donuts`. Utilizará este recurso de string en el siguiente paso.
4. Añada un TextView que aparecerá junto a la imagen de la donut como descripción, con los siguientes atributos:

Atributos del TextView	**Valor**
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:padding	"35dp"
android:layout_below	"@+id/choose_dessert"
android:layout_toRightOf	"@id/donut"
android:text	"@string/donuts"

5. Extraiga el recurso para la dimensión del atributo `android:padding` con el nombre `padding_wide`. Este nombre de recurso se utilizará en los pasos siguientes.
6. Añada un segundo ImageView al layout para el ice cream sandwich, utilizando los siguientes atributos:

Atributos para el ImageView del ice_cream	**Valor**
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:padding	"@dimen/padding_regular"
android:id	"@+id/ice_cream"
android:layout_below	"@id/donut"
android:contentDescription	"Ice cream sandwiches have chocolate wafers and vanilla filling."
android:src	"@drawable/icecream_circle"

7. Extraiga el valor del atributo `android:contentDescription` en el recurso de string con el nombre `ice_cream_sandwiches`.
8. Añada un TextView que aparecerá junto al ice cream sandwich como descripción, con los siguientes atributos:

Atributos del TextView	**Valor**
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"

android:padding	"@dimen/padding_wide"
android:layout_below	"@+id/donut"
android:layout_toRightOf	"@id/ice_cream"
android:text	"@string/ice_cream_sandwiches"

9. Agregue un tercer ImageView al layout para el froyo, usando los siguientes atributos:

Atrubutos para el ImageView del froyo	**Valor**
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:padding	"@dimen/padding_regular"
android:id	"@+id/froyo"
android:layout_below	"@id/ice_cream"
android:contentDescription	"FroYo is premium self-serve frozen yogurt."
android:src	"@drawable/froyo_circle"

10. Extraiga el valor del atributo `android:contentDescription` en el recurso de string con el nombre `froyo`.
11. Añada un TextView que aparecerá junto al froyo como descripción, con los siguientes atributos:

TextView attribute	**Value**
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:padding	"@dimen/padding_wide"
android:layout_below	"@+id/ice_cream"
android:layout_toRightOf	"@id/froyo"
android:text	"@string/froyo"

6.3 Agregar métodos onClick a las vistas de las imágenes

Se puede añadir el atributo `android:onClick` a cualquier View para que responda a un clic como lo hace un botón. En este paso añadirá el atributo `android:onClick` a las imágenes en el layout content_main.xml. También se necesita añadir un método para que el atributo `android:onClick` lo pueda llamar. El método, para esta tarea, muestra un mensaje toast que indica qué imagen fue tocada. (En una tarea posterior, modificará el método para iniciar otra actividad llamada OrderActivity).

1. Añada los siguientes recursos de string al archivo strings.xml para que los strings se muestren en el mensaje toast:

```
<string name="donut_order_message">You ordered a donut.</string>
<string name="ice_cream_order_message">You ordered an ice cream sandwich.</string>
<string name="froyo_order_message">You ordered a FroYo.</string>
```

2. Agregue el siguiente método `displayToast()` para mostrar un mensaje toast:

```
public void displayToast(String message) {
   Toast.makeText(getApplicationContext(), message, Toast.LENGTH_SHORT).show();
}
```

3. Añada el siguiente método `showFoodOrder`() al final del **MainActivity** (antes de cerrar la llave). Para esta tarea, se usará el método `displayToast`() para mostrar un mensaje toast.

```
/**
* Displays a toast message for the food order
* and starts the OrderActivity activity.
* @param message   Message to display.
*/
public void showFoodOrder(String message) {
   displayToast(message);
}
```

> **Consejo**: Las primeras cuatro líneas son un comentario en formato Javadoc, lo que hace que el código sea más fácil de entender y también ayuda a generar la documentación del código si utiliza Javadoc. Es una buena práctica añadir un comentario de este tipo a cada nuevo método que se cree. Para obtener más información acerca de cómo escribir comentarios, consulte Cómo escribir comentarios de documentación para la herramienta Javadoc.

Aunque podría haber añadido este método en cualquier posición dentro de **MainActivity**, es una buena práctica poner los métodos propios *por debajo* de los métodos proporcionados por la plantilla en la **MainActivity**.

4. Agregue los siguientes métodos a l final de **MainActivity** (puede agregarlos antes de `showFoodOrder()`)

```
/**
* Shows a message that the donut image was clicked.
*/
public void showDonutOrder(View view) {
   showFoodOrder(getString(R.string.donut_order_message));
}

/**
* Shows a message that the ice cream sandwich image was clicked.
```

```
*/
public void showIceCreamOrder(View view) {
   showFoodOrder(getString(R.string.ice_cream_order_message));
}

/**
* Shows a message that the froyo image was clicked.
*/
public void showFroyoOrder(View view) {
   showFoodOrder(getString(R.string.froyo_order_message));
}
```

5. Añada el atributo `android:onClick` a las tres imaViews en el content_main.xml:

```
<ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:id="@+id/donut"
        android:layout_below="@id/choose_dessert"
        android:contentDescription="@string/donut"
        android:src="@drawable/donut_circle"
        android:onClick="showDonutOrder"/>
. . .
<ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:id="@+id/ice_cream"
        android:layout_below="@id/donut"
        android:contentDescription="@string/ice_cream_sandwich"
        android:src="@drawable/icecream_circle"
        android:onClick="showIceCreamOrder"/>
. . .
<ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:id="@+id/froyo"
        android:layout_below="@id/ice_cream"
        android:contentDescription="@string/froyo"
        android:src="@drawable/froyo_circle"
        android:onClick="showFroyoOrder"/>
```

6. Ejecute la aplicación.

   Haciendo clic en las imágenes de la donut, el ice cream sandwich o el froyo, mostrará un mensaje toast relacionado a la orden, tal como se muestra en el siguiente figura.

Tarea 7: Usando radio buttons (botones de selección)

Los botones de selección (radio buttons) son controles de entrada útiles para seleccionar una única opción de un conjunto de opciones. Se debe utilizar los radio buttons si desea que el usuario vea todas las opciones disponibles una al lado de la otra. Si no es necesario mostrar todas las opciones una al lado de la otra, puede usar un spinner.

Más adelante, en esta práctica, agregará otra actividad y un layout de pantalla para configurar las opciones de entrega de un pedido de alimentos, y utilizará botones de selección para tales opciones de entrega.

Para obtener una descripción general y más ejemplos de código de los radio buttons, consulte Radio Buttons.

7.1 Agregar otra activity

Como ya ha aprendido en una lección anterior, una *actividad* representa una sola pantalla en tu aplicación en la que el usuario puede realizar una única tarea enfocada. Ya tiene una actividad, MainActivity.java. Ahora añadirá otra actividad para establecer las opciones de entrega de un pedido y utilizará un intent explícito para iniciar la segunda actividad.

1. Haga clic con el botón derecho en la columna izquierda de la carpeta **com.example.android.droidcafe** y seleccione **New > Activity > Empty ACtivity**. Edite el

Nombre de la actividad para que sea **OrderActivity** y el Nombre del layout como **activity_order**. Deje las otras opciones como están y haga clic en **Finish**.

La clase `OrderActivity` debe aparecer listada por debajo de `MainActivity` en la carpeta **java** y el archivo `activity_order.xml` debe aparecer en la carpeta **layout**. La plantilla Empty Activity agregó todos estos archivos.

2. Abra el archivo activity_order.xml. En el Editor de diseño, haga clic en la pestaña Text en la parte inferior de la pantalla y cambie el grupo de vista raíz a RelativeLayout, tal como lo ha hecho en ejercicios anteriores.
3. Abra el **MainActivity**. Cambie el método `showFoodOrder()` para hacer un intent explícito para iniciar la OrderActivity:

```
public void showFoodOrder(String message) {
        displayToast(message);
        Intent intent = new Intent(this, OrderActivity.class);
        startActivity(intent);
}
```

4. Ejecute la aplicación. Al hacer clic en un botón de imagen ahora se inicia la segunda actividad, que es una pantalla en blanco. (El mensaje toast aparece brevemente en la pantalla en blanco.)

7.2 Agregar el layout para los radio buttons

Para crear cada una de las opcines de los radio button, debe crear elementos RadioButton en el archivo layout activity_order.xml, que está vinculado a la OrderActivity. Después de editar el archivo layout, la disposición de los radio buttons en la OrderActivity se parecerá a la siguiente figura, dependiendo de la versión de Android Studio.

Dado que las selecciones de los radio buttons son mutuamente excluyentes, debe agruparlos dentro de un RadioGrup. Al agruparlos de esta manera, el sistema Android garantiza que sólo se pueda seleccionar un único radio button a la vez.

Nota: El orden en el que se listan los elementos RadioButton determina el orden en el que aparecen en la pantalla.

1. Abra el archivo activity_order.xml y añada un elemento TextView con el `id` como `order_intro_text` y los siguientes atributos:

Atributos del TextView	Valor
android:id	"@+id/order_intro_text"
android:layout_width	"match_parent"
android:layout_height	"wrap_content"
android:layout_marginTop	"24dp"
android:layout_marginBottom	"6dp"
android:textSize	"18sp"
android:text	"Choose a delivery method:"

2. Extraiga el recurso de string para `"Choose a delivery method:"` para que sea `choose_delivery_method`.
3. Extraiga los recursos de dimensión para los valores del margen:
    a. `"24dp"` para `text_margin_top`
    b. `"6dp"` para `text_margin_bottom`
    c. `"18sp"` para `intro_text_size`
4. Agregue un RadioGrupo al layout por debajo del TextView que acaba de agregar:

```
<RadioGroup
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:orientation="vertical"
   android:layout_below="@id/order_intro_text">
</RadioGroup>
```

5. Agregue los siguientes tres elementos de RadioButton dentro del RadioGroup, usando los siguientes atributos. La entrada `"onRadioButtonClick"` para el atributo `onClick` se resaltará hasta que añada ese método en la siguiente tarea.

Atributos del RadioButton #1	Valor
android:id	"@+id/sameday"
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:text	"Same day messenger service"
android:onClick	"onRadioButtonClicked"

Atributos del RadioButton #2	Valor

android:id	"@+id/nextday"
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:text	"Next day ground delivery"
android:onClick	"onRadioButtonClicked"

Atributos RadioButton #3	**Valor**
android:id	"@+id/pickup"
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:text	"Pick up"
android:onClick	"onRadioButtonClicked"

6. Extraiga los tres recursos de string para los atributos de `android:text` con los siguientes nombres, para que los strings puedan traducirse fácilmente:
   - same_day_messenger_service
   - next_day_ground_delivery
   - pick_up

7.3 Agregar el manejador del clic del radio button

El atributo `android:onClick` de cada elemento radio button especifica el método `onRadioButtonClick()` que va a gestionar el evento clic. Por lo tanto, debe añadir un nuevo método `onRadioButtonClick()` en la clase OrderActivity.

Normalmente, la aplicación mostrará algún mensaje con respecto al tipo de entrega elegido. Esto se logrará con un mensaje toast creando un método llamado `displayToast()` en OrderActivity.

En el método `onRadioButtonClick()` usará un bloque `switch case` para comprobar si se ha hecho clic en un radio button. Al final del bloque `switch case`, agregará una sentencia `default` la cual mostrará un mensaje `log` si no se ha seleccionado ninguno de los radio button.

1. Abra el archivo strings.xml y cree los siguientes recursos de string:
   a. Un recurso denominado `chosen` para el string `"Chosen: "` (incluya el espacio después de los dos puntos y las comillas).
   b. Un recurso llamado `nothing_clicked` para el string "`onRadioButtonClick: Nothing clicked`".
2. Abra **OrderActivity** y añada la siguiente sentencia para definir el `TAG_ACTIVITY` del mensaje `log`:

```
private static final String TAG_ACTIVITY = OrderActivity.class.getSimpleName();
```

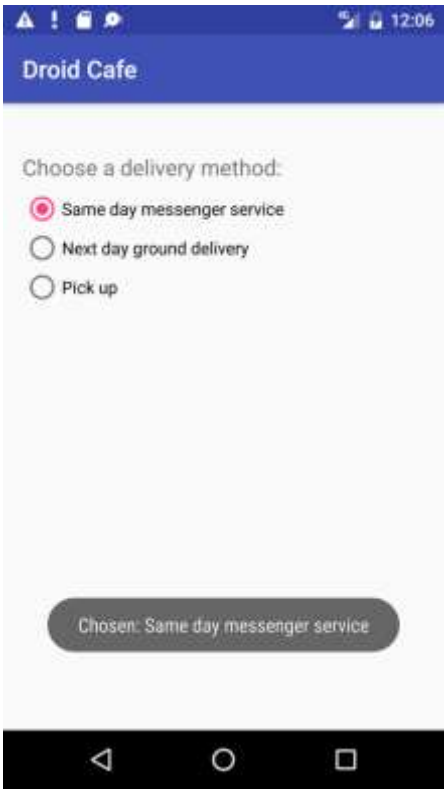
3. Agregue el método `displayToast()` a OrderActivity

```
public void displayToast(String message) {
   Toast.makeText(getApplicationContext(), message,
                        Toast.LENGTH_SHORT).show();
}
```

4. Añada el siguiente método `onRadioButtonClick()`, que comprueba si un radio button ha sido marcado, además usa un bloque `switch case` para determinar qué elemento de radio button fue seleccionado, con el fin de establecer el `message` apropiado para ese elemento que vaa usar con `displayToast()`:

```
public void onRadioButtonClicked(View view) {
   // Is the button now checked?
   boolean checked = ((RadioButton) view).isChecked();
   // Check which radio button was clicked
   switch(view.getId()) {
      case R.id.sameday:
         if (checked)
            // Same day service
            displayToast(getString(R.string.chosen) +
                        getString(R.string.same_day_messenger_service));
         break;
      case R.id.nextday:
         if (checked)
            // Next day delivery
            displayToast(getString(R.string.chosen) +
                        getString(R.string.next_day_ground_delivery));
         break;
      case R.id.pickup:
         if (checked)
            // Pick up
               displayToast(getString(R.string.chosen) +
                        getString(R.string.pick_up));
         break;
      default:
         Log.d(TAG_ACTIVITY, getString(R.string.nothing_clicked));
         break;
   }
}
```

5. Ejecute la aplicación. Toque una imagen para ver la actividad OrderActivity, la cual muestra las opciones de entrega. Marque una opción de entrega y verá un mensaje toast en la parte inferior de la pantalla con la opción seleccionada, como se muestra en la figura siguiente.



Resumen

En esta práctica, ha aprendido a:

- Configurar atributos del layout XML para controlar el teclado de un elemento EditText:
  - Utilice el valor `textAutoCorrect` para el atributo `androd:inputType` para cambiar el teclado de modo que sugiera correcciones ortográficas.
  - Utilice el valor `textCapSentences` para el atributo `android:inputType` para iniciar cada nueva oración con una letra mayúscula.
  - Utilice el valor `textPassword` para el atributo `android:inputType` para ocultar una contraseña al introducirla.
  - Utilice el valor `textEmailAddress` para el atributo `android:inputType` para mostrar un teclado de correo electrónico en lugar de un teclado estándar.
  - Utilice el valor `phone` para el atributo `android:inputType` para mostrar un teclado de teléfono en lugar de un teclado estándar.
- Utilice un control de entrada Spinner para proporcionar un menú desplegable y escriba el código para controlarlo:
  - Utilice un adaptador ArrayAdapter para asignar una serie de valores de texto como los elementos del menú spinner.
  - Implemente la interfaz `AdapterView.OnItemSelectedListener`, la cual requiere también añadir los métodos callback `onItemSelected()` y `onNothingSelected()` para activar el spinner y su listener.
  - Utilice el método callback `onItemSelected()` para recuperar el elemento seleccionado en el menú spinner utilizando `getItemAtPosition`.

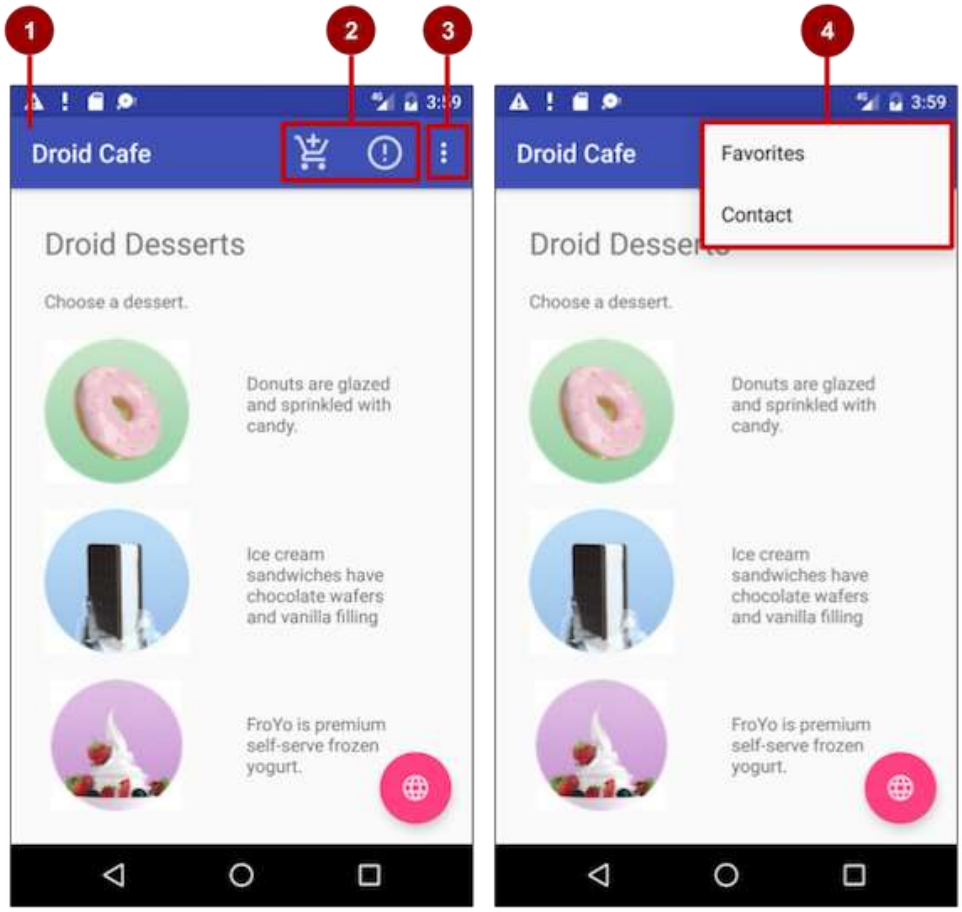
- Use `AlertDialog.Builder`, una subclase de AlertDialog, para crear un cuadro de diálogo de alerta estándar, usando `setTitle` para asignar el título, `setMessage` para definir el mensaje, y `setPositiveButton` y `setNegativeButton` para definir sus botones.
- Utilice los pickers de fecha y hora estándar:
    - Agregue un fragment para un picker de fecha, y extienda la clase DialogFragment para implementar `DatePickerDialog.OnDateSetListener` para un picker de fecha estándar con un listener.
    - Agregue un fragment para un picker de hora, y extienda la clase DialogFragment para implementar `TimePickerDialog.OnTimeSetListener` para un selector de hora estándar con un listener.
    - Implemente los métodos `onDateSet()`, `onTimeSet()`, y `onCreateDialog()`.
    - Utilice los métodos `onFinishDateDialog()` y `onFinishTimeDialog()` para recuperar la fecha y hora seleccionadas.
- Usar imágenes en un proyecto:
    - Copie una imagen en el proyecto y defina un elemento ImageView para utilizarla.
    - Añada el atributo `android:onClick` para hacer que los elementos ImageView tengan comportamiento de botón. Puede hacer que cualquier View se comporte como un botón con el atributo `android:onClick`.
- Usar radio button:
    - Cree una segunda operación.
    - Agregue los elementos del RadioButton dentro de un RadioGrupo en la segunda actividad.
    - Cree los manejadores de los radio button.
    - Inicie la segunda actividad desde un clic en una imagen.

5.2 Usar un menú de opciones

La *app bar* (también llamada *action bar*) es un espacio dedicado en la parte superior de la pantalla de cada activity. Cuando se crea una actividad a partir de una plantilla (como la Basic Activity Template), se incluye automáticamente una app bar para la actividad en un grupo de vistas de raíz CoordinatorLayout en la parte superior de la jerarquía de vistas.

El *menú de opciones (options menu)* en la barra de aplicaciones (app bar) proporciona una forma de navegación a otras actividades en la aplicación, o las opciones principales que afectan el uso de la aplicación en sí, pero no las que realizan una acción en un elemento de la pantalla. Por ejemplo, el menú de opciones podría proporcionarle al usuario opciones para navegar a otras actividades, como realizar un pedido o para realizar acciones que tengan un impacto global en la aplicación, como cambiar la configuración o la información del perfil.

En esta práctica, aprenderá cómo configurar la barra de la aplicación (app bar) y el menú de opciones (options menu) en una aplicación (como la que se muestra en la figura a continuación).



En la figura anterior:

1. **App Bar**. La barra de la aplicación incluye el título de la aplicación, el menú de opciones y el botón de despliegue.
2. **Iconos de acción de menú de opciones**. Los dos primeros elementos del menú de opciones aparecen como iconos en la barra de la aplicación.
3. **Botón de despliegue (Overflow button)**. El botón de despliegue (tres puntos verticales) abre un menú que muestra más elementos del menú de opciones.
4. **Opciones del menú de despliegue**. Después de hacer clic en el botón de despliegue, aparecen más elementos del menú de opciones en el menú de desbordamiento.

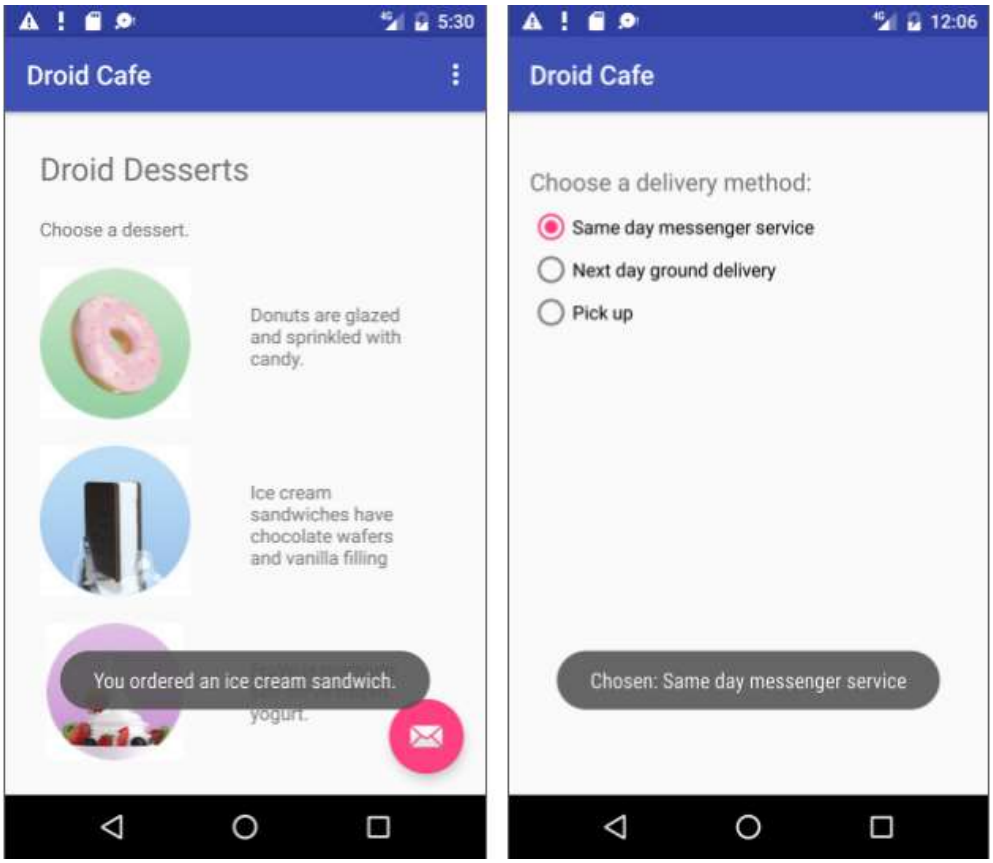
Los elementos del menú de opciones aparecen en las opciones del menú de despliegue (ver la figura anterior). Sin embargo, puede colocar algunos elementos como iconos, tantos como puedan entrar, en la barra de la aplicación. El uso de la barra de aplicaciones para el menú de opciones hace que su aplicación sea coherente con otras aplicaciones de Android, lo que permite a los usuarios comprender rápidamente cómo operar su aplicación y tener una gran experiencia.

Sugerencia: Para proporcionar una experiencia de usuario coherente y familiar, debe usar las API de menú para presentar las acciones del usuario y otras opciones en sus actividades. Ver Menus para más detalles.

Resumen de la aplicación

En la práctica anterior, creó una aplicación llamada Droid Cafe, que se muestra en la figura a continuación, utilizando la plantilla de Basic Activity. Esta plantilla también proporciona un esqueleto de un menú de opciones en la barra de la aplicación en la parte superior de la pantalla. Aprenderá cómo:

- Configura la barra de aplicaciones.
- Modificar el menú de opciones.
- Agregar iconos para algunos de los elementos del menú.
- Muestrar el icono del elemento de menú en la barra de la aplicación en lugar del menú de despliegue.
- Muestrar el elemento en el menú de despliegue, según el tamaño y la orientación de la pantalla.



Para este ejercicio, está utilizando Toolbar de la biblioteca de soporte de v7 appcompat como barra de aplicaciones. Hay otras formas de implementar una barra de aplicaciones. Por ejemplo, algunos temas configuran un ActionBar como una barra de aplicaciones de forma predeterminada. Pero el uso de la Toolbar, appcompat hace que sea más fácil configurar una barra de aplicaciones que funcione en la mayoría de los dispositivos, y también brinda más espacio para personalizar la barra de aplicaciones más adelante a medida que se desarrolla la aplicación.

Para obtener más información sobre las consideraciones de diseño para usar la barra de aplicaciones, consulte App Bar en la especificación de Material Design.

Para comenzar el proyecto, si no desea perder su proyecto, haga una copia del proyecto anterior como se indica en el Apéndice.

Tarea 1. Agregar elementos al menú de opciones

Debe abrir el proyecto Droid Cafe de la práctica anterior, o su copia, y agregará los elementos del menú al menú de opciones en la barra de la aplicación en la parte superior de la pantalla.

1.1 Examine el código del app bar

1. Abra el proyecto Droid Cafe de la práctica anterior, o su copia. El proyecto incluye los siguientes archivos de layout en la carpeta **res > layout**:
   a. **activity_main.xml**: el layout principal de MainActivity, la primera pantalla que el usuario ve.
   b. **content_main.xml**: el layout para el contenido de la pantalla MainActivity, que (como verá en breve) está *incluido* en activity_main.xml.
   c. **activity_order.xml**: el layout del OrderActivity, que ha agregado en la práctica anterior.
2. Abra content_main.xml. En la práctica anterior, agregó los TextViews y los ImageViews en el viewGroup raiz (el cual fue cambiado a RelativeLayout).

   El *comportamiento* del diseño para el `RelativeLayout` se configura en `@string/appbar_scrolling_view_behavior`, el cual controla el comportamiento de desplazamiento de la pantalla en relación con la barra de aplicaciones en la parte superior. Haga clic con el botón derecho (Control-clic) en este recurso de string y seleccione **Go to > Declaration** para ver el valor real del recurso de string, que se define en un archivo llamado "values.xml". Este archivo es generado por Android Studio, no visible en el Proyecto: Android view y no debe ser editado. El valor real de `@string/appbar_scrolling_view_behavior` en values.xml es "`android.support.design.widget.AppBarLayout$ScrollingViewBehavior`".

   Para obtener más información sobre el comportamiento de desplazamiento, consulte la entrada en el blog Android Design Support Library en el Blog de desarrolladores de Android. Para las prácticas de diseño que implican menús de desplazamiento, véase Técnicas de desplazamiento en la especificación del Material Design.

3. Abra activity_main.xml para ver la layout principal, el cual utiliza un layout CoordinatorLayout con un layout de AppBarLayout incrustado. Las etiquetas CoordinatorLayout y AppBarLayout requieren nombres completamente cualificados que especifiquen `android.support.design`, la cual es la librería Android Design Support Library.

   AppBarLayout es un LinearLayout vertical que utiliza la clase Toolbar de la biblioteca de soporte, en lugar de la ActionBar nativa, para implementar una barra de aplicaciones. La app bar es una sección en la parte superior de la pantalla que puede mostrar el título de la actividad, la navegación y otros elementos interactivos. El ActionBar nativo se comporta de manera diferente según la versión de Android que se ejecute en el dispositivo. Por esta razón, si se está agregando un menú de opciones, se debe usar la Toolbar de la biblioteca de soporte de appcompat v7 como una barra de aplicaciones. Usando la Toolbar hace que sea más fácil configurar una barra de aplicaciones que funcione en la mayoría de los dispositivos, y también le da espacio para personalizar su barra de aplicaciones más adelante a medida que se desarrolla su aplicación. La barra de herramientas incluye las funciones más recientes y funciona para cualquier dispositivo que pueda usar la biblioteca de soporte.

La barra de herramientas dentro de este diseño tiene el id `toolbar`, y también es especificado, al igual que AppBarLayout, con un nombre completamente calificado (`android.support.v7.widget`):

```
<android.support.design.widget.AppBarLayout
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:theme="@style/AppTheme.AppBarOverlay">

   <android.support.v7.widget.Toolbar
      android:id="@+id/toolbar"
      android:layout_width="match_parent"
      android:layout_height="?attr/actionBarSize"
      android:background="?attr/colorPrimary"
      app:popupTheme="@style/AppTheme.PopupOverlay" />

</android.support.design.widget.AppBarLayout>
```

Para obtener más detalles sobre la clase AppBarLayout, consulte AppBarLayout en la Referencia del desarrollador de Android. Para obtener más información sobre las barras de herramientas, consulte Toolbar en la Referencia del desarrollador de Android.

Sugerencia: el layout activity_main.xml también usa una declaración `include layout` para incluir el layout completo definido en content_main.xml. Esta separación de las definiciones de diseño facilita el cambio del *contenido* del diseño, apartando el layout de la definición de la barra de herramientas y el layout del coordinador. Esto es considerado una mejor práctica para separar el contenido (que podría necesitar ser traducido) del formato de su diseño.

4. Ejecute la aplicación. Observe la barra en la parte superior de la pantalla que muestra el nombre de la aplicación (Droid Cafe). También muestra el *botón de acción del despliegue* (*action overflow*, con tres puntos verticales) en el lado derecho. Toque el botón de despliegue para ver el menú de opciones, que en este momento tiene solo una opción de menú, **Settings**
5. Examine el archivo AndroidManifest.xml. La actividad `.MainActivity` está configurada para usar el tema `NoActionBar`:

```
android:theme="@style/AppTheme.NoActionBar"
```

El tema `NoActionBar` se define en el archivo styles.xml (expanda **app> res> values> styles.xml** para verlo). Los estilos están cubiertos en otra lección, pero puede ver que el tema `NoActionBar` configura el atributo `windowActionBar` como `false` (sin barra de acciones de ventana) y el atributo `windowNoTitle` como `true` (sin título).

La razón por la que se establecen estos valores es porque está definiendo la app bar en su layout (activity_main.xml) con AppBarLayout, en lugar de usar un ActionBar. El uso de uno de los temas NoActionBar evita que la aplicación use la clase nativa ActionBar para proporcionar la barra de la aplicación (app bar). La clase nativa ActionBar se comporta de manera diferente según la versión del sistema Android que esté usando el dispositivo. Por el contrario, las características más recientes se agregan a la versión de la librería de

soporte de Toolbar y están disponibles en cualquier dispositivo que pueda usar la biblioteca de soporte. Por este motivo, se debe usar la clase Toolbar de la biblioteca de soporte para implementar las app bar de las activities de la aplicación en lugar del ActionBar. El uso de la clase ToolBar de la biblioteca de soporte asegura que la aplicación tenga un comportamiento consistente en la mayoría de los dispositivos.

6. Observe que la MainActivity, extiende a `AppCompatActivity` y comienza con el método `onCreate()`:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    ...
```

Después de configurar el contenido de la vista en el layout activity_main.xml, el método `onCreate()` configura a `toolbar` para que sea la Toolbar definida en el layout activity_main.xml. Luego llama al método `setSupportActionBar()` de la actividad y le pasa `toolbar`, configurando al `toolbar` definido en activity_main.xml como la app bar para la actividad.

Para conocer las mejores prácticas sobre cómo agregar la app bar a una aplicación, consulte Adding the App Bar en Best Practices for User Interface.

1.2 Agregue más elementos de menú al menú de opciones

Ahora debe agregar los siguientes elementos al menú de opciones de la aplicación Droid Cafe:

- **Order**: Ir a la pantalla de la Order Activity para ver el pedido de comida.
- **Status**: verifica el estado de un pedido de comida.
- **Favorites**: muestra los alimentos favoritos.
- **Contact**: ponerse en contacto con el restaurante. Como no necesita el elemento **Settings** existente, lo cambiará por **Contact**.

Android proporciona un formato XML estándar para definir elementos de menú. En lugar de crear un menú en el código de la actividad, se puede definir un menú con todos sus elementos en un recurso de menú XML. A continuación, puede inflar el recurso de menú (cargarlo como un objeto Menu) en la activity o fragment:

1. Eche un vistazo al menu_main.xml (expanda **res > menu** en la vista Proyecto). Allí se definen los elementos del menú con las etiquetas `<item> </item>` dentro del bloque `<menu> </menu>`. El único elemento de menú proporcionado desde la plantilla es `action_settings` (la opción **Settings**), la cual es definida como:

```
<item
```

```
android:id="@+id/action_settings"
android:orderInCategory="100"
android:title="@string/action_settings"
app:showAsAction="never" />
```

En Android Studio, el atributo `android:title` muestra el string de valor "Settings" aunque el string esté definido como un recurso. Android Studio muestra el valor para que pueda dar un vistazo de cuál es el valor sin tener que abrir el archivo de recursos strings.xml. Si hace clic en ese string, este cambia para mostrar el recurso de string `"@string/action_settings"`.

2. Cambie los siguientes atributos del elemento `action_settings` para convertirlo en el elemento `action_contact` (no cambie el atributo `android:orderInCategory` existente):

Atributo	Valor
android:id	"@+id/action_contact"
android:title	"Contact"
app:showAsAction	"never"

3. Extraiga el string cableado "`Contact`" en el recurso de string action_contact.
4. Agregue un nuevo elemento de menú usando la etiqueta `<item> </item>` dentro del bloque `<menu> </menu>` y otorgue al elemento los siguientes atributos:

Atributo	Valor
android:id	"@+id/action_order"
android:orderInCategory	"10"
android:title	"Order"
app:showAsAction	"never"

El atributo `android:orderInCategory` especifica el orden en que los elementos aparecerán en el menú, con el número más bajo apareciendo más arriba en el menú. El elemento Contact se le da len 100, que es un número grande para especificar que se muestre en la parte inferior en lugar de en la parte superior. Establezca el elemento Order en 10, lo cual lo coloca arriba del Contact, y deja mucho espacio en el menú para más artículos.

5. Extraiga el string cableado "`Order`" en el recurso de string `action_order`.
6. Agregue dos elementos de menú más de la misma manera con los siguientes atributos:

Atributos del elemento Status	Valor

android:id	"@+id/action_status"
android:orderInCategory	"20"
android:title	"Status"
app:showAsAction	"never"

Atributos del elemento Favorites	**Valor**
android:id	"@+id/action_favorites"
android:orderInCategory	"40"
android:title	"Favorites"
app:showAsAction	"never"

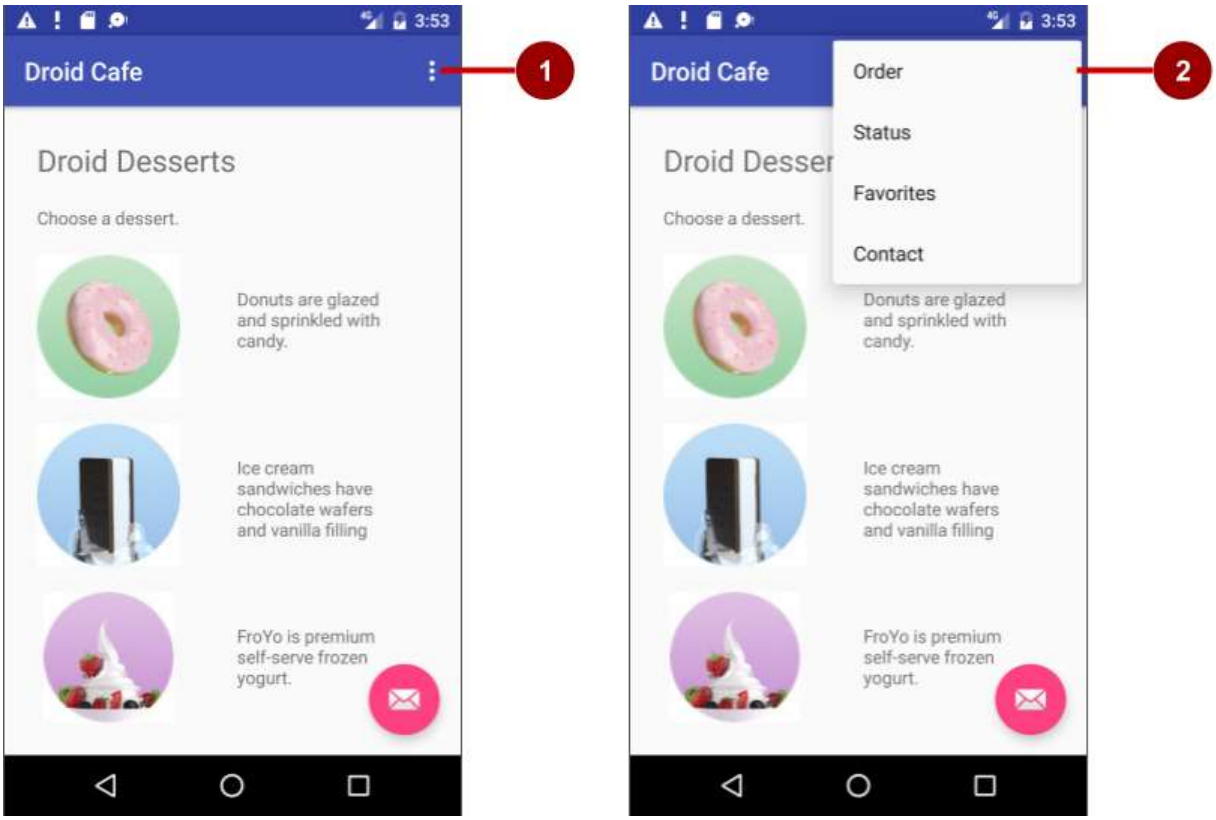
7. Extraiga “`Status`” en el recurso `action_status` y “`Favorites`” en el recurso `action_favorites`.
8. Mostrará ahora un mensaje toast con un mensaje de acción dependiendo cuál ha sido el element de menu seleccionado. Agregur los siguientes nombres y valores de string en strings.xml para los mensajes:

```
<string name="action_order_message">You selected Order.</string>
<string name="action_status_message">You selected Status.</string>
<string name="action_favorites_message">You selected Favorites.</string>
<string name="action_contact_message">You selected Contact.</string>
```

9. Abra la MainActivity y cambie la sentencia `if` en el método `onOptionsItemSelected()` reemplazando el id `action_settings` con el nuevo id `action_order`:

```
if (id == R.id.action_order)
```

Ejecute la aplicación y toque el icono desplegable (los tres puntos), como es mostrado en el lado izquierdo de la siguiente figura, para ver el menú de opciones, tal como se muestra en el lado derecho de la figura. Pronto agregará métodos callback para responder al elemento seleccionado en este menú.



En la figura anterior:

1. Toque el icono desplegable en la app bar para ver las opciones del menú.
2. Las opciones se despliegan por debajo de la app bar.

Observe el orden de los elementos en el menú de opciones. Se usó el atributo `android:orderInCategory` para especificar la prioridad de los elementos del menú: El elemento Order es 10, seguido de Status (20) y Favorites (40), y Contact es el último (100). La siguiente tabla muestra la prioridad de los elementos del menú:

Notice the order of items in the options menu. You used the android:orderInCategory attribute to specify the priority of the menu items in the menu: The Order item is 10, followed by Status (20) and Favorites (40), and Contact is last (100). The following table shows the priority of items in the menu:

Elemento del Menu	Atributo orderInCategory
Order	10
Status	20
Favorites	40
Contact	100

Tarea 2. Agregar iconos para los elementos del menú

Siempre que sea posible, se desea mostrar las acciones más utilizadas utilizando iconos en la app bar para que el usuario pueda hacer clic en ellas sin tener que hacer clic primero en el icono desplegable. En esta tarea, se agregarán iconos para algunos de los elementos del menú y se mostrará algunos de los elementos del menú como iconos en la app bar en la parte superior de la pantalla.

En este ejemplo, vamos a suponer que las acciones de **Order** y **Status** se consideran las más utilizadas. **Favorites** se usa ocasionalmente, y **Contact** es el menos utilizado. Se puede establecer iconos para estas acciones y especificar lo siguiente:

- **Order** y **Status** se deben mostrar siempre en la barra de la aplicación.
- **Favorites** se debe mostrar en la barra de la aplicación si existe espacio; si no, debería aparecer en el menú de desplegable.
- **Contact** no debería aparecer en la barra de la aplicación; el debe aparecer *solamente* en el menú de desplegable.

2.1 Agregar iconos para los elementos del menú

Para especificar iconos a las acciones, primero debe agregar los iconos como elementos de imagen a la carpeta **drawable**.

1. Expanda **res** en la vista Project y haga clic con el botón derecho (o Control-clic) en **drawable**.
2. Elija **New > Image Asset**. Aparecerá el cuadro de diálogo Configure Image Asset.
3. Elija **Action Bar and Tab Items** en el menú desplegable.
4. Cambie **ic_action_name** por **ic_order_white** (para la acción **Order**). La pantalla Configure Image Asset debe verse de la siguiente manera (consulte Create App Icons with Image Asset Studio para obtener una descripción completa).

5. Haga clic en la imagen clipart (el logotipo de Android junto a "Clipart:") para seleccionar una imagen clipart como icono. Aparece una página de iconos. Haga clic en el ícono que desea usar para la acción **Order** (por ejemplo, el ícono del carrito de compra puede ser apropiado).
6. Elija **HOLO_DARK** en el menú desplegable del Tema. Esto configura al icono para que sea blanco sobre un fondo oscuro (o negro). Haga clic en **Next**.
7. Haga clic en **Finish** en el diálogo Confirm Icon Path.
8. Repita los pasos anteriores para los iconos **Status** y **Favorites**, nombrandolos **ic_status_white** y **ic_favorites_white** respectivamente. Tal vez desearía usar el icono circled-I para el **Status** (usado tipicamente para Información) y el icono corazón para **Favotites**.

2.2 Mostrar los elementos del menú como iconos en la app bar

Para mostrar los elementos del menú como iconos en la barra de aplicaciones, utilice el atributo `app:showAsAction` en el menu_main.xml. Los siguientes valores para el atributo especifican si la acción debe aparecer o no en la barra de aplicaciones como un icono:

- `"siempre"`: Siempre aparece en la barra de aplicaciones. (Si no hay suficiente espacio, puede solaparse con otros iconos de menú).
- `"ifRoom"`: Aparece en la barra de aplicaciones sólo si hay espacio.
- `"never"`: Nunca aparece en la barra de la aplicación; su texto aparece en el menú desplegable.

Siga estos pasos para mostrar algunos de los elementos del menú como iconos:

1. Vuelva a abrir el menu_main.xml y agregue los siguientes atributos a los elementos **Order**, **Status** y **Favorites** para que siempre aparezcan los dos primeros (**Order** y **Status**) y el elemento **Favorites** sólo aparezca si hay espacio para ello:

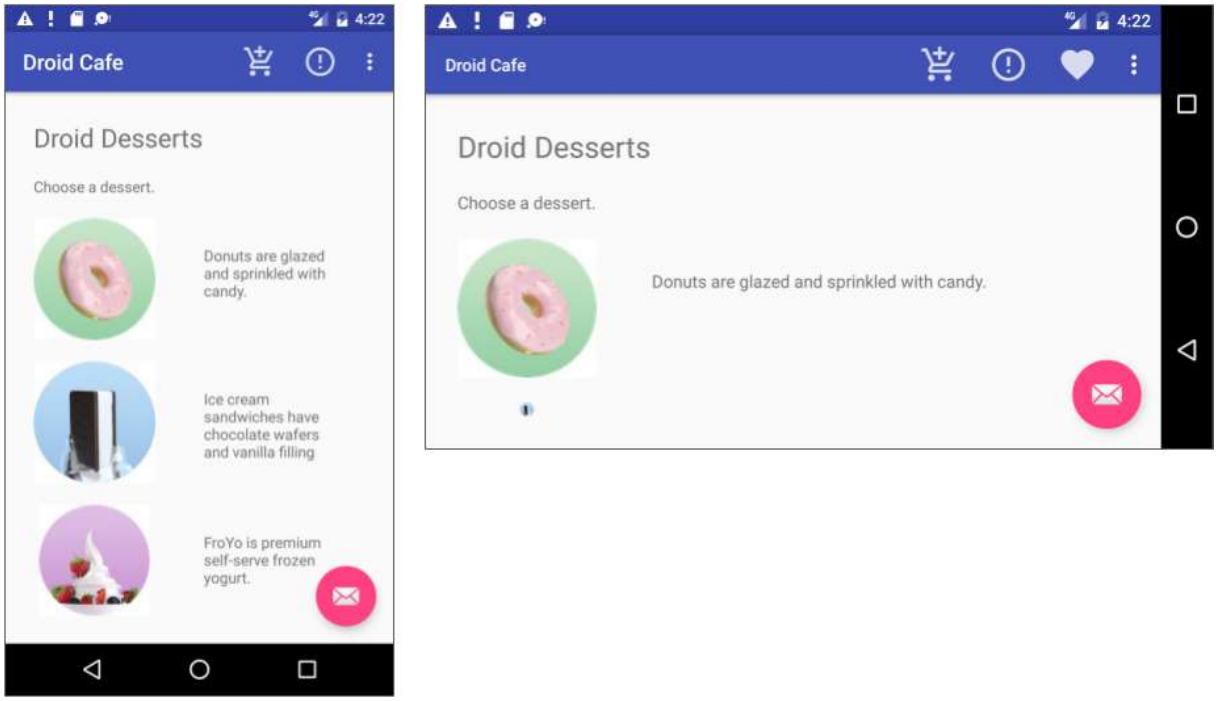
Atributos del elemento Order	**Valor anterior**	**Valor nuevo**
android:icon		"@drawable/ic_order_white"
app:showAsAction	"never"	"always"

Atributos del elemento Status	**Valor anterior**	**Valor nuevo**
android:icon		"@drawable/ic_status_white"
app:showAsAction	"never"	"always"

Atributos del elemento Favorites	**Valor anterior**	**Valor nuevo**
android:icon		"@drawable/ic_favorites_white"
app:showAsAction	"never"	"ifRoom"

2. Ejecuta la aplicación. Ahora se deben ver al menos dos iconos en la barra de aplicaciones: el ícono de **Order** y el de **Status** como se muestra en la figura siguiente. Si su dispositivo o el emulador se muestran en orientación vertical, las opciones de **Favorites** y **Contact** aparecen en el menú desplegable.
3. Gire su dispositivo a la orientación horizontal, o si está ejecutando en el emulador, haga clic en los iconos **Rotate Left** o **Rotate Right** para girar la pantalla a la orientación horizontal. Debería ver los tres íconos en la barra de la aplicación para **Order**, **Status** y **Favorites**.

Consejo: ¿Cuántos botones de acción caben en la app bar? Depende de la orientación y del tamaño de la pantalla del dispositivo. Siempre aparecen menos botones en una orientación vertical, como se muestra en el lado izquierdo de la figura de abajo, en comparación con una orientación horizontal como se muestra en el lado derecho de la figura de abajo. Los botones de acción no pueden ocupar más de la mitad del ancho de la barra de aplicaciones principal.



Tarea 3. Manejar el elemento seleccionado del menú

En esta tarea, se agregará un método para mostrar un mensaje sobre qué elemento del menú se ha tocado y se usará el método `onOptionsItemSelected()` para determinar en qué elemento del menú se hizo el toque.

3.1 Crea un método para mostrar la opción del menú

1. Abra el MainActivity.
2. Si aún no ha agregado el siguiente método (en la lección anterior) para mostrar un mensaje toast, agréguelo ahora:

```
public void displayToast(String message) {
   Toast.makeText(getApplicationContext(), message,
                          Toast.LENGTH_SHORT).show();
}
```

El método `displayToast()` obtiene el `message` del string apropiado (tal como `action_contact_message`).

3.2 Utilice el controlador de eventos onOptionsItemSelected

El método `onOptionsItemSelected()` maneja las selecciones hechas en el menú de opciones. Se agregará un bloque `switch` para determinar qué elemento del menú se seleccionó y qué mensaje crear para cada elemento seleccionado. (En lugar de crear un mensaje para cada elemento, podría implementar un manejador de eventos para cada elemento que realice una acción, como iniciar otra actividad, como se muestra más adelante en esta lección).

1. Busque el método `onOptionsItemSelected()`. La sentencia `if` del método, proporcionada por la plantilla, determina si se ha hecho clic en un determinado elemento de menú, utilizando el `id` del elemento de menú (`action_order` en el ejemplo siguiente):

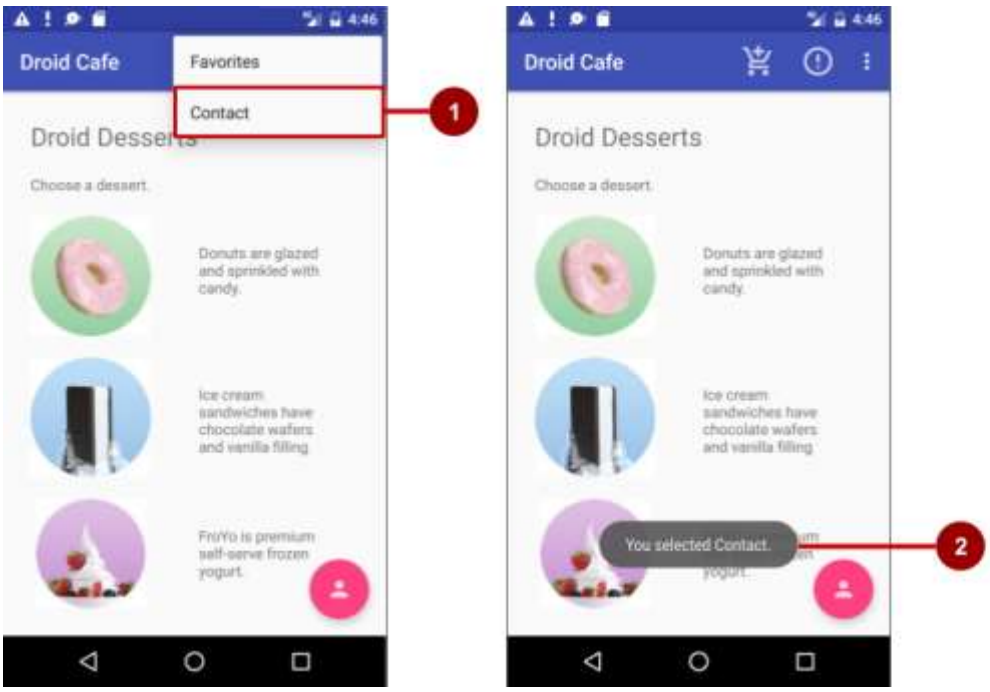
```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    if (id == R.id.action_order) {
        return true;
    }
    return super.onOptionsItemSelected(item);
}
```

2. Reemplace la sentencia if y la asignación del id con el siguiente bloque switch el cual configura el mensaje apropiado basado en el id del element de menú:

```
  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_order:
            displayToast(getString(R.string.action_order_message));
            return true;
        case R.id.action_status:
            displayToast(getString(R.string.action_status_message));
            return true;
        case R.id.action_favorites:
            displayToast(getString(R.string.action_favorites_message));
            return true;
        case R.id.action_contact:
            displayToast(getString(R.string.action_contact_message));
            return true;
        default:
            // Do nothing
    }
```

```
    return super.onOptionsItemSelected(item);
}
```

3. Ejecute la aplicación. Ahora debería ver un mensaje toast diferente en la pantalla, como se muestra en el lado derecho de la figura a continuación, según el elemento de menú que se haya elijido.



En la figura anterior:

1. Seleccionar el elemento **Contact** en el menú de opciones.
2. El mensaje toast que aparece.

Finalizará la aplicación DroidCafe en la próxima lección.

Resumen

En esta práctica, ha aprendido a hacer lo siguiente:

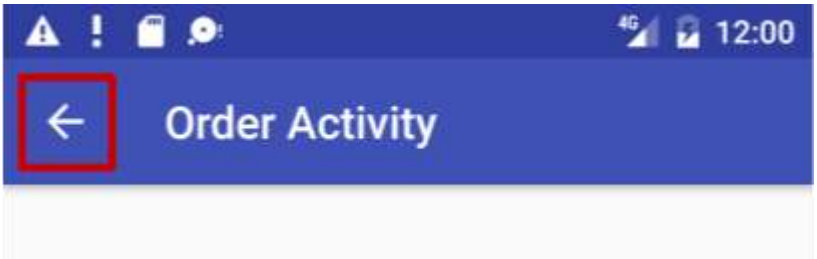
- Configura un menú de opciones en la barra de aplicaciones:
    - Usar la plantilla Basic Activity para configurar automáticamente el menú de opciones y un botón de acción flotante.
    - Usar `@string/appbar_scrolling_view_behavior` para proporcionar el comportamiento de desplazamiento estándar del menú de opciones de la barra de aplicaciones.
    - Usar un grupo de vistas CoordinatorLayout con la clase AppBarLayout para crear un menú de opciones en la barra de aplicaciones.

    - Usar una declaración `include layout` en el archivo layout XML para incluir un layout completo definido en otro archivo XML.
    - Usar el tema `NoActionBar` para evitar que la aplicación use los atributos nativos de la clase ActionBar en la app bar, para configurar el atributo `windowActionBar` en `false` (sin barra de acciones de la ventana) y el atributo `windowNoTitle` para `true` (sin título).
    - Usar el método `onCreate()` de una actividad para llamar al método `setSupportActionBar()` de la actividad para configurar la barra de herramientas definida en el layout como la barra de la aplicación para la actividad.
    - Definir un menú y todos sus elementos en un recurso de menú XML, y luego inflar el recurso de menú en una actividad o fragment para cargarlo como un objeto Menu.
    - Usar el atributo `android:orderInCategory` para especificar el orden en que los elementos del menú aparecen en el menú, con el número más bajo indicando al que aparece más arriba en el menú.
    - Usar el atributo `app:showAsAction` para mostrar los elementos del menú como iconos en la barra de la aplicación.
    - Agregar controladores de eventos para los elementos del menú de opciones y usar el método `onOptionsItemSelected()` para recuperar el elemento seleccionado del menú de opciones.
- Usar íconos en un proyecto:
    - Agregar iconos a un proyecto y usarlos para mostrar los elementos del menú en la barra de la aplicación.

5.3 Usando la Barra de Aplicaciones y las Pestañas para Navegación

En las primeras etapas del desarrollo de una aplicación, se debe determinar las rutas que los usuarios deben tomar a través de la aplicación para hacer algo, como hacer un pedido o navegar por el contenido. Cada ruta permite a los usuarios navegar a través, dentro y fuera de las diferentes tareas y piezas de contenido dentro de la aplicación.

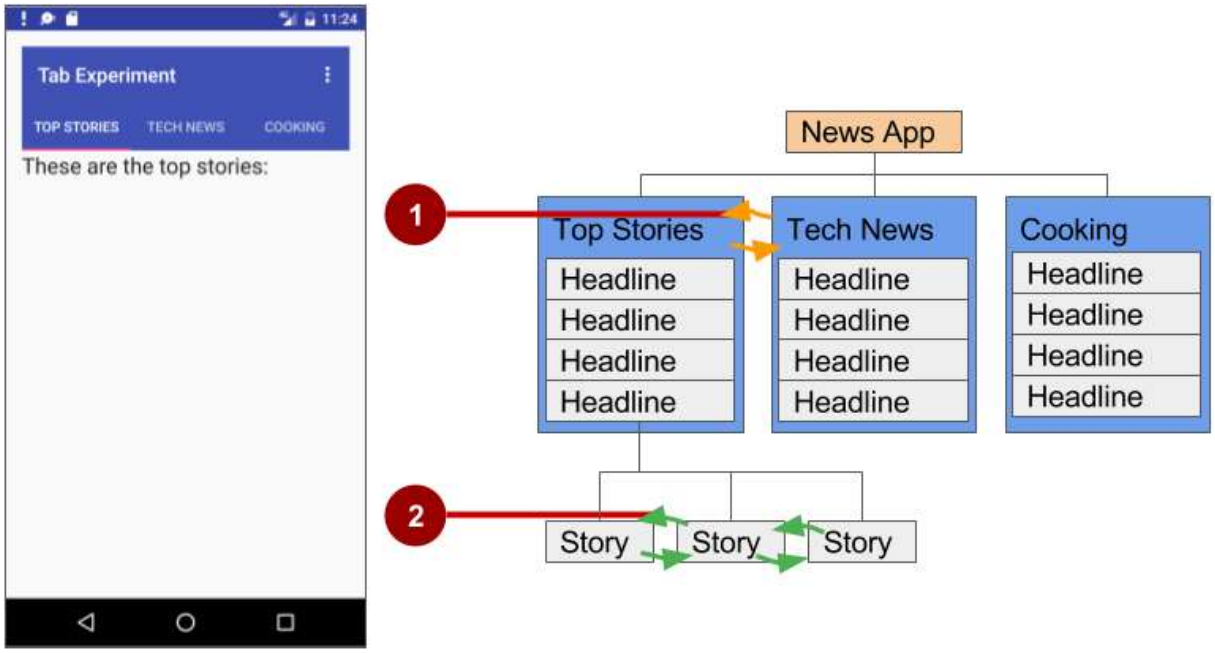
En este práctica, aprenderá a añadir un botón **Up** (una flecha hacia la izquierda) en la barra de aplicaciones de la aplicación, tal como se muestra a continuación, para navegar desde una pantalla secundaria *hasta* la pantalla principal.



El botón **Up** siempre se utiliza para navegar a la pantalla superior en la jerarquía. Se diferencia del botón **Back** (el triángulo en la parte inferior de la pantalla), en que este último proporciona navegación a cualquier pantalla que el usuario haya visto anteriormente.

Esta práctica también introduce la *navegación por pestañas* (*tab navegation*), en la que aparecen pestañas en la parte superior de una pantalla, lo que permite navegar por otras pantallas. La navegación por pestañas es una solución muy popular para la navegación lateral de una pantalla secundaria a otra pantalla secundaria que es un hermano, como se muestra en el siguiente diagrama. Las pestañas proporcionan navegación hacia y desde las pantallas de los hermanos Top Stories, Tech News y Cooking sin tener que navegar hasta el padre. Las pestañas también pueden proporcionar navegación de ida y vuelta a las historias, que son pantallas de hermanos bajo el padre de Top Stories.

Las pestañas son más apropiadas para cuatro o menos pantallas hermanas. El usuario puede tocar en una pestaña para ver una pantalla diferente, o deslizar el dedo hacia la izquierda o hacia la derecha para también ver una pantalla diferente.



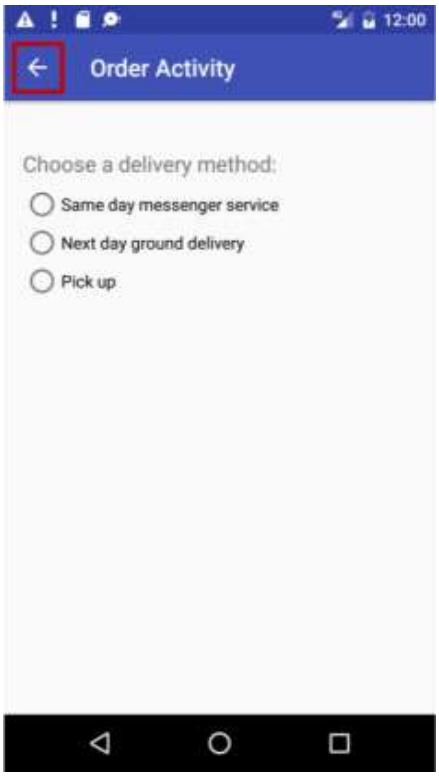
En la figura anterior:

1. Navegación lateral desde una categoría de pantalla a otra
2. Navegación lateral desde una historia de pantalla a otra.

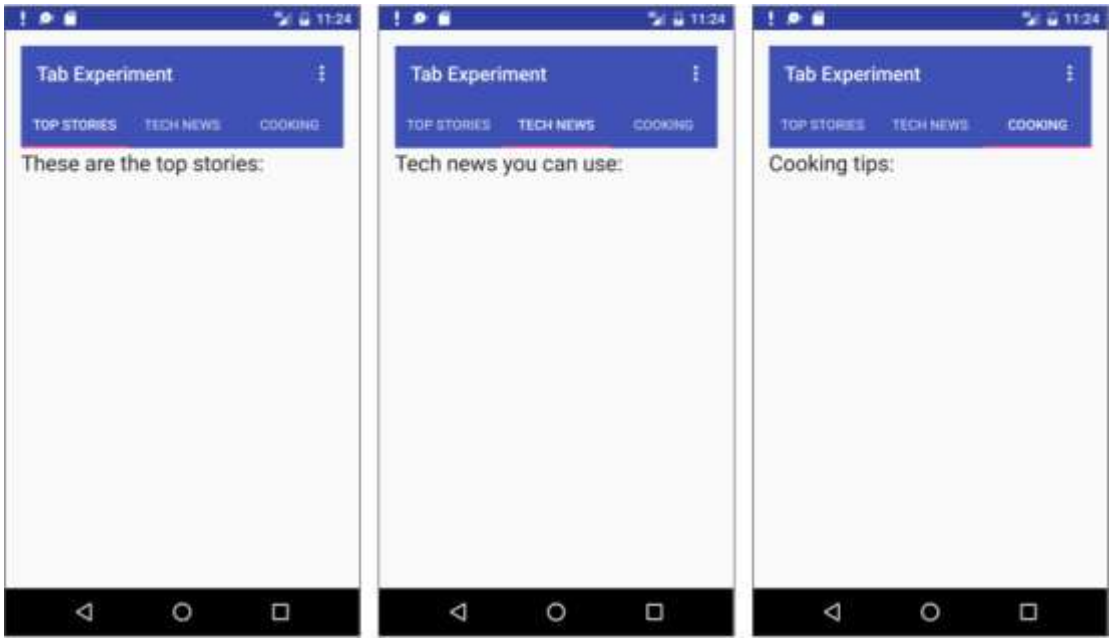
Resumen de la aplicación

En la práctica anterior creamos una aplicación llamada Droid Cafe en tres partes, usando la plantilla Basic Activity. Esta plantilla además proporciona una barra de aplicaciones en la parte superior de la pantalla. Aquí aprenderá a añadir un botón **Up** (flecha hacia la izquierda) a la barra de aplicaciones para la navegación hacia el padre, desde la segunda actividad (OrderActivity) hacia la actividad principal (MainActivity). Con esto se completará la aplicación Droid Cafe.

Puede continuar el proyecto desde donde lo dejó en la práctica anterior o hacer una copia, para mantener el anterior como referencia.



La segunda aplicación que creará para la navegación por pestañas mostrará tres pestañas debajo de la barra de aplicaciones para navegar a las pantallas hermanas. A medida que el usuario toca una pestaña, la pantalla muestra el contenido de una pantalla dependiendo de la pestaña que se haya tocado. El usuario también puede deslizar hacia la izquierda y hacia la derecha para visitar el contenido de las pantallas. La clase ViewPager maneja automáticamente las vistas Swipping (de deslizamiento).

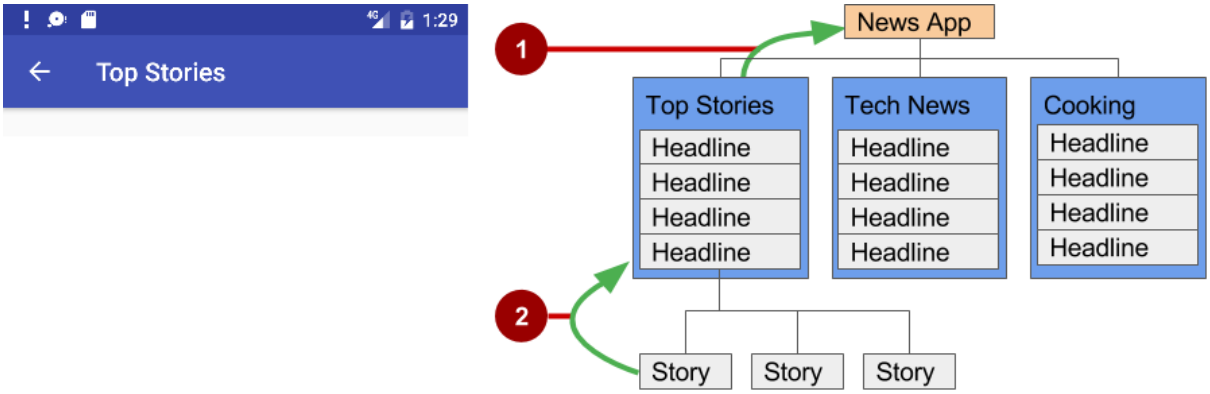


Tarea 1. Agregar un button Up para navegación ancestral

La aplicación debería facilitar a los usuarios una forma de retornar a la pantalla principal de la aplicación, que es la actividad principal. Una forma de hacerlo es proporcionar un botón **Up** en la barra de aplicaciones para todas las actividades que son hijas de la actividad principal.

El botón **Up** proporciona una navegación "*ascendente*" ancestral, que permite al usuario *subir de* una página secundaria a la página principal. El botón **Up** es la flecha hacia la izquierda en el lado izquierdo de la barra de aplicaciones, como se muestra en el lado izquierdo de la figura de abajo.

Cuando el usuario toca el botón **Up**, la aplicación navega a la actividad padre. El diagrama a la derecha de la siguiente figura muestra cómo se utiliza el botón Arriba para navegar dentro de una aplicación basándose en las relaciones jerárquicas entre pantallas.



En la anterior figura:

1. Navegando desde los hermanos de primer nivel hasta el padre.
2. Naavegando desde los hermanos de segundo nivel a los de primer nivel actuando como nivel padre.

Consejo: El botón **Back** (el triángulo en la parte inferior de la pantalla) difiere del botón **Up**. El botón **Back** proporciona navegación a cualquier pantalla que haya visto anteriormente. Si tiene varias pantallas secundarias por las que el usuario puede navegar utilizando un patrón de navegación lateral (como se describe en la siguiente sección), el botón **Back** enviará al usuario de vuelta a la pantalla secundaria anterior, no a la pantalla principal. Se utiliza el botón **Up** si desea proporcionar navegación ancestral desde una pantalla secundaria hasta la pantalla principal. Para obtener más información acerca de la navegación Up, consulte Proporcionar navegación Up.

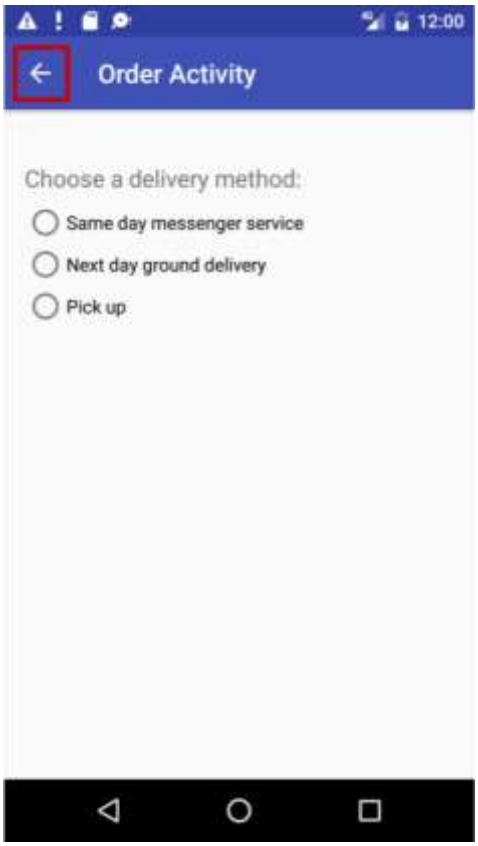
Como aprendió anteriormente, al agregar actividades a una aplicación, puede agregar la navegación con el botón **Up** a una actividad secundaria como OrderActivity declarando al padre de la actividad como MainActivity en el archivo AndroidManifest.xml. También puede definir el atributo `android:label` para un título de la pantalla de actividad, como "`Order Activity`":

1. SI no tiene la aplicación Droid Café abierta de la práctica anterior, descargue el proyecto de Android Studio DroidCafe Part3 y renombrelo como **DroidCafe**.
2. Abra el proyecto **DroidCafe**.
3. Abra el AndroidManifest.xml.
4. Cambie el elemento OrderActivity por lo siguiente:

```
<activity android:name=".OrderActivity"
  android:label="Order Activity"
  android:parentActivityName="com.example.android.
                                        droidcafe.MainActivity">
  <meta-data
     android:name="android.support.PARENT_ACTIVITY"
     android:value=".MainActivity"/>
</activity>
```

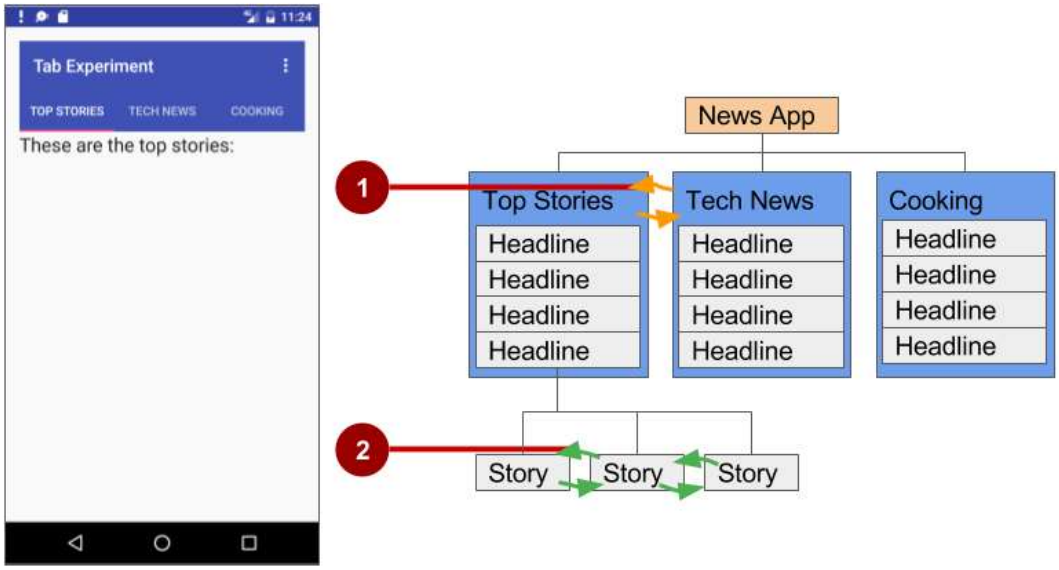
5. Extraiga el valor de android:label "`Order Activity`" a un recurso de string llamado `title_activity_order`.
6. Ejecute la aplicación.

La pantalla Order Activity ahora incluye el butón **Up** (resaltado en la figura siguiente) en la barra de la aplicación para retornar a la actividad padre.



Tarea 2. Usar la navegación por pestañas con vistas por deslizamiento

Al usar la navegación lateral, se permite al usuario pasar de un hermano a otro (en un mismo nivel de una jerarquía de varios niveles). Por ejemplo, si la aplicación proporciona varias categorías de historias (como Top Stories, Tech News y Cooking, como se muestra en la figura de abajo), puede proporcionar a sus usuarios la capacidad de navegar de una categoría a la siguiente, sin tener que navegar de nuevo hasta la pantalla principal. Otro ejemplo de navegación lateral es la posibilidad de deslizar el dedo hacia la izquierda o hacia la derecha en una conversación de Gmail para ver una más reciente o una más antigua en la misma Bandeja de entrada.



En la figura anterior:

1. Navegación lateral de una pantalla de categoría a otra
2. Navegación lateral de un pantalla de historias a otra

Se puede implementar la navegación lateral con pestañas que representan cada pantalla. Las pestañas aparecen en la parte superior de la pantalla, como se muestra en el lado izquierdo de la figura anterior, con el fin de facilitar la navegación a otras pantallas. La navegación por pestañas es una solución muy popular para la navegación lateral de una pantalla secundaria a otra pantalla secundaria que es *hermana*: es decir, que está en la misma posición en la jerarquía y compartiendo la misma pantalla principal. La navegación por pestañas a menudo se combina con la capacidad de deslizar pantallas secundarias de izquierda a derecha y de derecha a izquierda.

La clase principal que se utiliza para mostrar pestañas es TabLayout en la Biblioteca de Soporte ade Diseño de Android. Ella proporciona una disposición horizontal para mostrar las pestañas. Se pueden mostrar las pestañas debajo de la barra de aplicaciones y utilizar la clase PagerAdapter para rellenar las "páginas" de las pantallas dentro de un ViewPager. El ViewPager es un gestor de layout que permite al usuario hojear a la izquierda y a la derecha a través de las pantallas. Este es un patrón común para presentar diferentes pantallas de contenido dentro de una actividad: se utiliza un *adapter* para llenar la pantalla de contenido de manera de mostrarla en la actividad y un *layout manager* que cambie el contenido de las pantallas según la pestaña seleccionada.

Se puede proporcionar una implementación de un *PagerAdapater* para generar las pantallas que muestren la vista. El ViewPager se utiliza a menudo en conjunto con un Fragment. Mediante el uso de fragmentos, se tiene una forma conveniente de gestionar el ciclo de vida de cada "página" de la pantalla.

Para utilizar las clases de la Biblioteca de Soporte de Android, se debe añadir `com.android.support:design:xx.xx.x` (donde `xx.xx.x` es la versión más reciente) al archivo **build.gradle (Module: app)**.

Los siguientes son adaptadores estándar para usar fragmentos con el ViewPager:

- FragmentPagerAdapter: Diseñado para navegar entre pantallas hermanas (páginas) las cuales representan un pequeño número fijo de pantallas.

- FragmentStatePagerAdapter: Diseñado para la paginación a través de una colección de pantallas (páginas) en las cuales el número de pantallas es indeterminado. Este destruye los fragmentos a medida que el usuario navega a otras pantallas, minimizando el uso dela memoria. La aplicación de esta práctica usa FragmentStatePagerAdapter.

2.1 Crear el layout para la navegación por pestañas

1. Cree un nuevo Proyecto usando la plantilla Empty Activity. Nombre la aplicación **Tab Experiment**
2. Edite el archivo **build.gradle (Module: app)**, y agregue las siguientes líneas (si no han sido ya agregadas) a la sección de dependencias:

```
compile 'com.android.support:design:25.0.1'
compile 'com.android.support:support-v4:25.0.1'
```

   Si el Android Studio sugiere una versión con un número mayor, edite las líneas anteriores para actualizar la versión. Además, si el Android Studio sugiere una nueva versión de `compileSdkVersion`, `buildToolsVersion`, y/o `targetSdkVersion` edítelas para actualizar la versión.

3. Para usar un Toolbar en vez de un action bar y el título de la aplicación, agregue las siguientes sentencias al archivo **res > values > styles.xml** para ocultar el action bar y le título de la aplicacación:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
 ...
 <item name="windowActionBar">false</item>
 <item name="windowNoTitle">true</item>
</style>
```

4. Abra el archivo activity_main.xml. En el Editor de diseño, haga clic en la ficha Text en la parte inferior de la pantalla y cambie el grupo de vista raíz a RelativeLayout, como ha hecho en ejercicios anteriores.
5. En el activity_main.xml, elimine el TextView suministrado por la plantilla y agregue una Toolbar, un TabLayout y un ViewPager dentro del layout raíz.

   A medida que escribe el atributo `app:popupTheme` para la `Toolbar` como se muestra a continuación, `app` estará en rojo si no agregó la siguiente declaración al RelativeLayout:

```
<RelativeLayout xmlns:app="http://schemas.android.com/apk/res-auto"
```

   Se puede hacer clic en `app` y pulsar **Alt-Return** (**Opción-Return)**, y Android Studio añadirá automáticamente la declaración.

Dependiendo de su versión de Android Studio, el código de diseño tendrá un aspecto parecido al siguiente:

Código de la solución:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.android.tabexperiment.MainActivity">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:background="?attr/colorPrimary"
        android:minHeight="?attr/actionBarSize"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
        app:popupTheme="@style/ThemeOverlay.AppCompat.Light"/>

    <android.support.design.widget.TabLayout
        android:id="@+id/tab_layout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/toolbar"
        android:background="?attr/colorPrimary"
        android:minHeight="?attr/actionBarSize"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"/>

   <android.support.v4.view.ViewPager
        android:id="@+id/pager"
        android:layout_width="match_parent"
        android:layout_height="fill_parent"
        android:layout_below="@id/tab_layout"/>

</RelativeLayout>
```

2.2 Crear un layout y una clase para cada fragment

1. Añada un fragment por cada pantalla con pestañas: **TabFragment1**, **TabFragment2** y **TabFragment3**. Para añadir cada fragmento haga lo siguiente:
    a. Haga clic en **com.example.android.tabexperiment** en la vista del proyecto.
    b. Seleccione **Archivo > Nuevo > Fragmento > Fragmento (en blanco)**.
    c. Nombre el fragmento **TabFragment1**.

d. Marque que la opción "Create layout XML?". y cambie el Fragment Layout Name en el archivo XML a **tab_fragment1**.
e. *Desmarque* las opciones "Include fragment factory methods?". y "include interface callbacks?". No se necesitarán estos métodos.
f. Haga clic en **Finish**.
g. Repita los pasos anteriores, usando **TabFragment2** y **TabFragment3** para el Paso c, y **tab_fragment2** y **tab_fragment3** para el Paso d.

Cada fragmento (**TabFragment1**, **TabFragment2** y **TabFragment3**) es creado con su definición de clase para extender `Fragment`. Además, cada fragmento *infla* el layout asociado a la pantalla (`tab_fragment1`, `tab_fragment2`, y `tab_fragment3`), utilizando el conocido patrón de diseño de inflado de recursos que aprendió en un capítulo anterior con el menú de opciones.

Por ejemplo, **TabFragment1** lucirá así:

```
public class TabFragment1 extends Fragment {
          @Override
          public View onCreateView(LayoutInflater inflater,
                    ViewGroup container, Bundle savedInstanceState) {
          return inflater.inflate(R.layout.tab_fragment1, container, false);
          }
}
```

Android Studio automáticamente incluye las siguientes sentencias de import:

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
```

2. Edite cada archivo XML de layout de fragment (**tab_fragment1**, **tab_fragment2** y **tab_fragment3**):
    a. Cambie la etiqueta raíz a RelativeLayout.
    b. Añada un TextView con un texto como " These are the top stories ".
    c. Configure la apariencia del texto con `android:textAppearance="?android:attr/textAppearanceLarge"`.
    d. Repita los pasos anteriores para cada archivo XML de layout de fragments, introduciendo un texto diferente para el TextView del paso b.
3. Examine cada archivo XML de layout de fragments. Por ejemplo, **tab_fragment1** debería verse así:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
```

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="These are the top stories:"
        android:textAppearance="?android:attr/textAppearanceLarge"/>

</RelativeLayout>
```

4. En el archivo XML de layout del fragment **tab_fragment1**, extraiga el string de "`These are the top stories:`" en el recurso de string `tab_1`. Haga lo mismo para las cadenas en **tab_fragment2** y **tab_fragment3**.

2.3 Añadir un PageAdapter

El patrón adapter-layout manager le permite proporcionar diferentes contenidos de pantallas dentro de una actividad: utilice un *adapter* para rellenar el contenido de la pantalla con lo que se mostrará en la actividad y un *layout manager* que cambie el contenido de la pantalla según la pestaña seleccionada.

1. Agregue una nueva clase `PagerAdapter` a la aplicación que extiende FragmentStatePagerAdapter y defina el número de pestañas (`mNumOfTabs`):

```
public class PagerAdapter extends FragmentStatePagerAdapter {
    int mNumOfTabs;
    public PagerAdapter(FragmentManager fm, int NumOfTabs) {
             super(fm);
             this.mNumOfTabs = NumOfTabs;
    }
}
```

Al introducir el código anterior, Android Studio importa automáticamente:

```
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentStatePagerAdapter;
```

Si el FragmentManager en el código anterior está en rojo, aparecerá un icono de bombillo rojo cuando haga clic en él. Haga clic en el icono del bombillo y seleccione **Importar class**. Aparecen las opciones de importación. Seleccione la siguiente opción de importación:

FragmentManager (android.support.v4)

Seleccionando los imports siguientes:

```
import android.support.v4.app.FragmentManager;
```

Además, el Android Studio subraya la definición de la clase PagerAdapter y, si hace clic en **PagerAdapter**, muestra un icono de bombillo rojo. Haga clic en el icono y elija **Implement methods** y, a continuación, haga clic en **OK** para implementar los métodos `getItem()` y `getCount()` ya seleccionados.

2. Cambie el método `getItem()` recién agregado con lo siguiente, el cual usa un `bloque switch` para devolver el fragmento a mostrar basándose en la pestaña en la que se hace clic

```
@Override
public Fragment getItem(int position) {
        switch (position) {
            case 0:
                return new TabFragment1();
            case 1:
                return new TabFragment2();
            case 2:
                return new TabFragment3();
            default:
                return null;
        }
}
```

3. Cambie el nuevo método getCount() con el siguiente para devolver el número de pestañas:

```
@Override
  public int getCount() {
      return mNumOfTabs;
  }
```

2.4 Inflar el Toolbar y el TabLayout

Dado que está usando pestañas que caben debajo de la barra de aplicaciones, se ha configurado la barra de aplicaciones (app bar) y la Toolbar en el layout activity_main.xml en el primer paso de esta tarea. Ahora necesita inflar la Barra de herramientas (usando el mismo método descrito en un capítulo anterior sobre el menú de opciones), y crear una instancia de TabLayout para posicionar las pestañas.

Since you are using tabs that fit underneath the app bar, you have set up the app bar and Toolbar in the activity_main.xml layout in the first step of this task. Now you need to inflate the Toolbar (using the same method described in a previous chapter about the options menu), and create an instance of TabLayout to position the tabs.

1. Inflar el `Toolbar` en el método `onCreate()` en MainActivity.java:

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
   ...
   Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
   setSupportActionBar(toolbar);

   // Create an instance of the tab layout from the view.
   ...
}
```

En el código anterior, la `Toolbar` está en rojo, y debería aparecer un icono de bombillo rojo. Haga clic en el icono y seleccione **Importar class**. Aparecerán las opciones de importación. Seleccione **Toolbar (android.support.v7.widget.Toolbar)**, y la siguiente declaración de importación aparecerá en su código:

```
import android.support.v7.widget.Toolbar;
```

2. Abra **strings.xml**, y cree los siguientes recursos de string:

```
<string name="tab_label1">Top Stories</string>
<string name="tab_label2">Tech News</string>
<string name="tab_label3">Cooking</string>
```

3. Al final del método `onCreate()`, cree una instancia del tab layout desde el elemento `tab_layout` y configure el texto de cada pestaña usando addTab():

```
...
// Create an instance of the tab layout from the view.
TabLayout tabLayout = (TabLayout) findViewById(R.id.tab_layout);
// Set the text for each tab.
tabLayout.addTab(tabLayout.newTab().setText(R.string.tab_label1));
tabLayout.addTab(tabLayout.newTab().setText(R.string.tab_label2));
tabLayout.addTab(tabLayout.newTab().setText(R.string.tab_label3));
// Set the tabs to fill the entire layout.
tabLayout.setTabGravity(TabLayout.GRAVITY_FILL);

// Use PagerAdapter to manage page views in fragments.
...
```

2.5 Usar el PagerAdapter para administrar las vistas de las pantallas

1. Debajo del código que agregó al método `onCreate()` en la tarea anterior, agregue el siguiente código para usar el `PagerAdapter` para administrar las vistas de la pantalla (página) en los fragmentos:

```
...
```

```
// Using PagerAdapter to manage page views in fragments.
// Each page is represented by its own fragment.
// This is another example of the adapter pattern.
final ViewPager viewPager = (ViewPager) findViewById(R.id.pager);
final PagerAdapter adapter = new PagerAdapter
                (getSupportFragmentManager(), tabLayout.getTabCount());
viewPager.setAdapter(adapter);
// Setting a listener for clicks.
...
```

2. Al final del método `onCreate()`, configure un listener (TabLayoutOnPageChangeListener) para detectar si se hace clic en una pestaña, y cree el método `onTabSelected()` para configurar el `ViewPager` en la pantalla con la pestañas apropiada. El código debe tener el siguiente aspecto:

```
...
// Setting a listener for clicks.
viewPager.addOnPageChangeListener(new
                TabLayout.TabLayoutOnPageChangeListener(tabLayout));
tabLayout.addOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {
   @Override
   public void onTabSelected(TabLayout.Tab tab) {
      viewPager.setCurrentItem(tab.getPosition());
   }

   @Override
   public void onTabUnselected(TabLayout.Tab tab) {

   }

   @Override
      public void onTabReselected(TabLayout.Tab tab) {
      }
});
}
```

3. Ejecute la aplicación. Toque cada pestaña para ver cada “página” (pantalla). Además debe ser capaz de deslizar hacia la derecha y hacia la izquierda para visitar diferentes “páginas”.

Resumen

- Agregar un botón de navegación Up a una actividad hija declarando la actividad padre en el archivo AndroidManifest.xml
- Configurado la navegación up:
    - Las Pestañas son una Buena solución para la “navegación lateral” entre vistas hermanas.
    - La clase primaria usada para las pestañas es TabLayout en la librería de soporte de diseño.

- Se debe usar un patron adapter cuando se rellenan las pestañas (páginas) con datos.
- Un ViewPager es un layout manager que permite al usario deslizar las páginas de datos hacia la izquierda o la derecha.
- El ViewPager se usa freccuentemente en conjunto con fragments.
- Existen dos adaptadores estándar para usar el ViewPager: el FragmentPagerAdapter y el FragmentStatePagerAdapter.

5.4 Crear un Recycler View

Mostrar y manipular una lista desplegable de elementos de datos similares, como hizo en la práctica de desplazamientos de vista, es una característica común de las aplicaciones. Por ejemplo, contactos, listas de reproducción, fotos, diccionarios, listas de la compra, un índice de documentos o una lista de juegos guardados son ejemplos de listas desplegables.

Anteriormente en esta clase, se usó el ScrollView para realizar el desplazamiento de otras Views. ScrollView es fácil de usar, pero no se recomienda para su uso en producción, especialmente para largas listas de elementos desplazables.

RecyclerView es una subclase de ViewGroup y es una forma más eficiente de mostrar listas desplegables. En lugar de crear una vista para cada elemento, sea visible o no, RecyclerView crea un número limitado de elementos de la lista y los reutiliza para hacer visible el contenido.

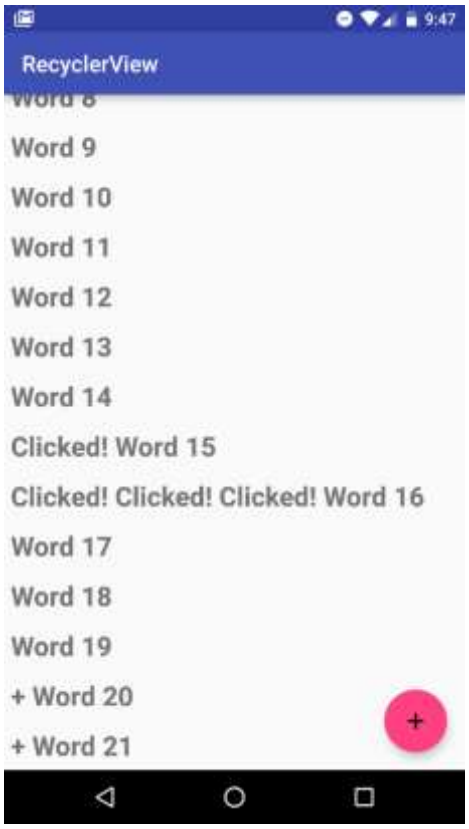
En estas prácticas se usará un RecyclerView para:

- Mostrar una lista desplegable de elementos.
- Agregar un *handler* de clics a cada elemento.
- Añadir elementos a la lista utilizando un floating action button (FAB) (butón de acción flotante), que se ve como el botón rosa de la imagen de abajo. Los botones de acción flotante se pueden utilizar para acciones comunes o para una acción promovida, es decir, una acción que desea que el usuario realice.

En esta práctica se creará una aplicación que usará un RecyclerView para mostrar una lista de elementos como una lista desplegable y asociar el comportamiento de un clic a los elementos de la lista.Además se utilizará un botón de acción flotante para agregar elementos al RecyclerView.

Resumen de la aplicación

La aplicación “RecyclerView” mostrará un larga lista de palabras.



- Tocando un elemento este quedará marcado que recibió un clic
- Tocando el botón de acción flotante agregará un elemento a la lista.
- No existirá una entrada de palabras por el usuario en esta aplicación.

Tarea 1. Crear y configurar un Proyecto nuevo

En esta tarea se creará y configurará un nuevo proyecto para la aplicación RecyclerView.

1.1 Crear el proyecto

1. Iniciar el Android Studio y crear un Nuevo Proyecto con los siguientes parámetros:

Atributo	Valor
Application Name	RecyclerView
Company Name	com.example.android o un dominio propio
Phone and Tablet Minimum SDK	API15: Android 4.0.3 IceCreamSandwich
Template	Empty Activity
Generate Layout file box	Checked

2. Ejecute la aplicación. Deb ever el título "RecyclerView" y "Hello World" en la vista en blanco.

1.2. Agregar las librerías de soporte al archivo build

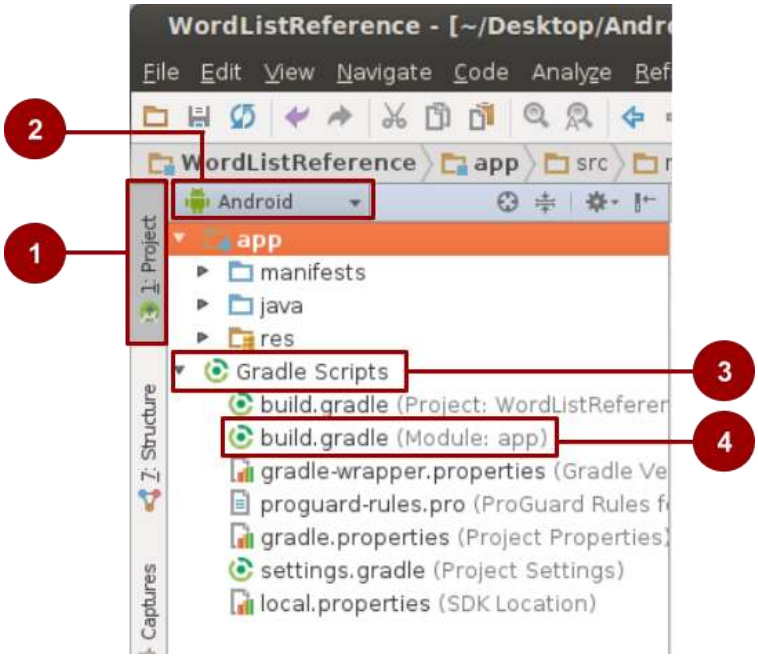
Para utilizar el RecyclerView y el botón de acción flotante (FAB), necesita añadir las respectivas Android Support Libraries a su build.

¿Por qué?: Las bibliotecas de soporte para Android proporcionan versiones compatibles con versiones anteriores de las API de Android, componentes de interfaz de usuario adicionales y un conjunto de utilidades. La clase RecyclerView se encuentra en el paquete Android Support; para utilizarla se deben incluir dos dependencias en el proceso de compilación de Gradle.

Siga estos pasos y consulte la captura de pantalla:

In order to use the RecyclerView and the floating action button (FAB), you need to add the respective Android Support Libraries to your build.

1. En Android Studio, en su proyecto nuevo, asegúrese de que está en el panel **Project** (1) y en la vista **Android** (2).
2. En la jerarquía de archivos, encuentre la carpeta **Gradle Scripts** (3).
3. Expanda **Gradle Scripts**, si es necesario, y abra el archivo **build.gradle (Module: app)** (4).



4. Hacia el final del archivo **build.gradle (Module: app)**, busque la sección de dependencias.
5. Añada estas dos dependencias como las dos últimas líneas dentro de la sección de dependencias:

```
compile 'com.android.support:recyclerview-v7:23.1.1'
compile 'com.android.support:design:23.1.1'
```

- Probablemente exista una línea como esta, pero el número puede ser diferente: `compile 'com.android.support:appcompat-v7:23.1.1''`.
- Añada sus líneas debajo de esa línea.
- **Haga coincidir el número de versión de sus líneas con el número de versión de esa línea existente.**
- Asegúrese de que los números de versión de todas las bibliotecas son los mismos y coinciden con la versión `compiledSdkVersion` en la parte superior del archivo. Si estos no coinciden, obtendrá un error de tiempo de compilación.

6. Si se le solicita, sincronize la aplicación ahora.
7. Ejecute su aplicación. Debería ver la misma aplicación "RecyclerView" mostrando "Hello World". Si obtiene errores de gradle, sincronice el proyecto. No es necesario instalar plugins adicionales.

Solución:

Este es un ejemplo de la sección de dependencias del archivo build.gradle. Su archivo puede ser ligeramente diferente y sus entradas pueden tener un número de versión diferente.

```
dependencies {
   compile fileTree(dir: 'libs', include: ['*.jar'])
   testCompile 'junit:junit:4.12'
   compile 'com.android.support:appcompat-v7:23.1.1'
   compile 'com.android.support:recyclerview-v7:23.1.1'
   compile 'com.android.support:design:23.1.1'
}
```

Tarea 2. Crear un dataset

Antes de poder visualizar cualquier cosa, se necesitan datos para mostrar. En una aplicación más sofisticada, los datos pueden provenir de un almacenamiento interno (un archivo, una base de datos SQLite, preferencias guardadas), de otra aplicación (Contactos, Fotos), o de Internet (almacenamiento en la nube, Google Sheets, o cualquier fuente de datos con un API). Para este ejercicio, simulará los datos creándolos en la main activity dentro del método `onCreate()`.

¿Por qué?: El almacenamiento y la recuperación de datos es un tema que se tratará en el capítulo sobre almacenamiento de datos. Luego tendrá la oportunidad de ampliar la aplicación para utilizar datos reales en esa lección posterior.

2.1 Agregar el código para crear los datos

En esta tarea creará dinámicamente una lista enlazada de veinte strings de palabras que terminan en un número que se irá incrementando, tal como ["Word 1", "Word 2", "Word 3", ….].

Se debe utilizar una LinkedList para esta práctica. Consulte el código de la solución si necesita ayuda.

1. Abra el archive MainActivity.java
2. Agregue una variable member private llamada mWordList que contenga la linkedList.
3. Agregue una variable de tipo entero llamada mCount usada como contador para rastrear el número de cada palabra.
4. Agregue el código que complete la lista mWordList con las palabras. Para ello concatene el string "Word" con el valor de mCount, y luego proceda a aumentar la cuenta.
5. Dado que aún no puede mostrar las palabras para la prueba, agregue una declaración de log que verifique que las palabras se agreguen correctamente a la linkedList.
6. Ejecute su aplicación para asegurarse de que no hay errores.

La UI de la palicación no ha cambiado, pero debe ver una lista de mensajes log en el logcat, tales como: `android.example.com.wordlist D/WordList: Word 1`

Solución:

Variables de la clase:

```
private final LinkedList<String> mWordList = new LinkedList<>();
private int mCount = 0;
```

En el método onCreate() de la MainActivity:

```
for (int i = 0; i < 20; i++) {
   mWordList.addLast("Word " + mCount++);
   Log.d("WordList", mWordList.getLast());
}
```

Tarea 3. Crear un RecyclerView

En esta práctica, mostrará los datos en un RecyclerView. Dado que la creación de un RecyclerView completamente operativo involucra varias partes, asegúrese de corregir inmediatamente cualquier error que vea en el Android Studio.

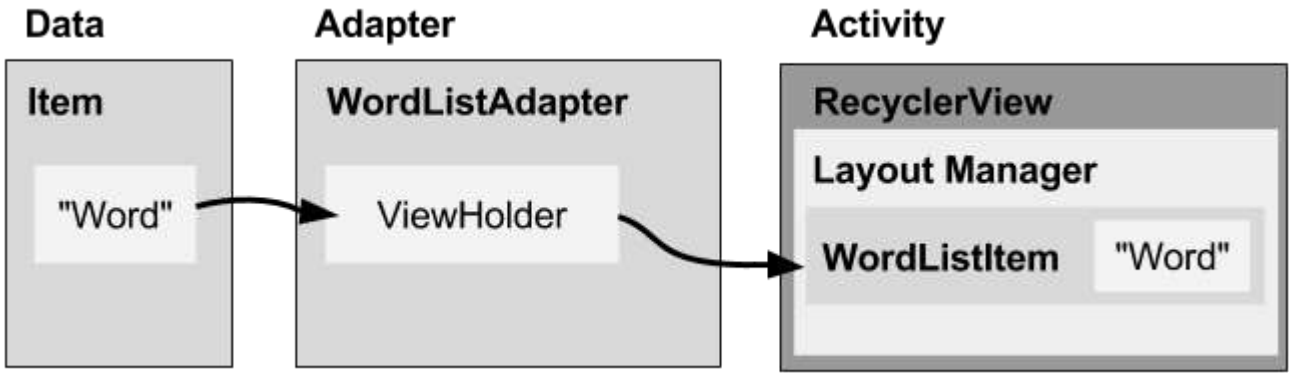
Para mostrar los datos en un RecyclerView, necesitará de las siguientes partes:

- **Datos.** Usará la mWordList.
- **Un RecyclerView.** La lista desplegable que contiene los elementos de esa lista.
- **Un layout para uno de los elementos de los datos.** Todos los elementos lucirán igual.
- **Un layout manager.** El layout manager maneja la organzación (layout) de los components de la Interfaz en una vista. Ya ha utilizado el LinearLayout en una práctica anterior en la que el sistema Android se encarga del layout por usted. El RecyclerView requiere de un

layout manager explícito para administrar la disposición de los elementos de la lista que contiene. Esta disposición puede ser vertical, horizontal o una cuadrícula. Usará un gestor de linear layout vertical proporcionado por Android.

- **Un adapter.** El adapter conecta los datos al RecyclerView. Prepara los datos en un contenedor de vista. Ud creará un adapter que insertará y actualizará las palabras generadas en las vistas.
- **Un view holder.** Dentro del adaptador, creará una clase ViewHolder que contiene la información que mostrará un elemento del layout del elemento.

El siguiente diagram muestra la relación entre los datos, el adapter, el view holder y el layout manager.



Resumen de los pasos de implementación

Para implementar estas piezas, necesitará hacer lo siguiente:

1. Crear el layout XML para la aplicación "RecyclerView" (activity_main.xml).
2. Crear el layout XML utilizado para presentar un elemento de la lista, la cual es WordListItem (wordlist_item.xml).
3. Crear un adapter (WordListAdapter) con un view holder (WordViewHolder). Implementar el método que tomará los datos, los coloca en un view holder y permite que el layout manager los muestre.
4. En el método onCreate de la MainActivity, creear un RecyclerView e inicializarlo con el adapter y un layout manager estándar. Esto lo vamos a hacer paso a paso.

3.1. Crear el layout principal en el activity_main.xml

En las aplicaciones anteriores, se usaba LinearLayout para organizar las vistas. Para acomodar el RecyclerView y el botón de acción flotante que se agregará más tarde, se necesita usar un grupo de vista diferente llamado un coordinador de layout. CoordinatorLayout es más flexible que el LinearLayout para organizar las vistas. Por ejemplo, vistas como el botón de acción flotante pueden superponerse a otras vistas.

En el main_activity.xml, reemplace el código creado por la Empty Activity con el código para un CoordinatorLayout, y luego agregue un RecyclerView:

1. Abra el build.gradle (Module:app) y verifique que la dependencia del recycler view existe.

```
compile 'com.android.support:recyclerview-v7:24.1.1'
```

2. Abra el activity_main.xml.

3. Seleccione todo el código en el activity_main.xml y reemplacelo con este otro código:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
</android.support.design.widget.CoordinatorLayout>
```

4. Inspeccione el código y note lo siguiente:

    - Las propiedades especificadas para este grupo de vistas son las mismas que para un LinearLayout, ya que algunas propiedades básicas, como layout_width y layout_height, son requeridas para todas las vistas y grupos de vistas.

    - Debido a que el CoordinatorLayout está en la biblioteca de soporte, debe especificar la ruta completa de la biblioteca de soporte. Esto mismo se debe hacer también para el RecyclerView.

5. Agregue el código del RecyclerView code dentro del CoordinatorLayout:

    - Se necesita especificar la ruta complete, debido a que el RecyclerView es parte de la librería de soporte.

```
<android.support.v7.widget.RecyclerView>
</android.support.v7.widget.RecyclerView>
```

6. Suministre al RecyclerView las siguientes propiedades:

Atributo	**Valor**
android:id	"@+id/recyclerview"
android:layout_width	match_parent
android:layout_height	match_parent

7. Ejecute la aplicación y asegúrese que no hay errores mostrados en el logcat. Sólo verá una pantalla en blanco, debido a que aún no ha puesto ningún elemento en el RecyclerView.

Solución:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerview"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </android.support.v7.widget.RecyclerView>

</android.support.design.widget.CoordinatorLayout>
```

3.2. Crear el layout para un element de la lista

El adapter necesita el layout de un elemento de la lista. Todos los elementos utilizarán el mismo layout. Se necesita especificar ese layout de elemento de lista en un archivo de recursos de layouts separado, ya que será utilizado por el adapter, independientemente del RecyclerView.

Cree un layout simple de elemento de palabra utilizando un LinearLayout vertical con un TextView:

1. Haga clic en el botón derecho del mouse en la carpeta **app/res/layout** y seleccione **New > Layout resource file**
2. Nombre al archive como wordlist_item y haga clic en **OK**.
3. En el mode Text, cambie el LinearLayout que fue creado con el archivo para que coincida con los siguientes atributos. A medida que avance extraiga los recursos.

Attribute	Value
android:layout_width	"match_parent"
android:layout_height	"wrap_content"
android:orientation	"vertical"
android:padding	"6dp"

4. Agregue un TextView para la palabra (word) al LinearLayout:

Attribute	Value
android:id	"@+id/word"
android:layout_width	"match_parent"
android:layout_height	"wrap_content"

android:textSize	"24sp"
android:textStyle	"bold"

3.3 Crear un estilo con los atributos del TextView

Puede utilizar estilos para permitir que los elementos compartan grupos de atributos de visualización. Una manera fácil de crear un estilo es extraer el estilo de un elemento de UI que ya ha creado. Para extraer la información de estilo para la vista de texto de la palabra:

1. Mientras tiene abierto wordlist_item.xml, pase el ratón por encima de la sección TextView que acaba de crear y haga clic con el botón derecho del ratón sobre **Refactor > Extract > Style**.
2. En el diálogo **Extract Android Style**,
    - Nombre su estilo **word_title**.
    - Deje todas las casillas marcadas.
    - Marque la casilla de verificación `**Use Style Where Possible**'.
    - Haga clic en **OK**.
3. Cuando se le solicite, aplique el estilo a **Whole Project**.
4. Busque y examine el estilo word_title en `values/styles.xml`.
5. Regrese a wordlist_item.xml. El text view ahora hace referencia al estilo en lugar de usar propiedades de estilo individuales.
6. Ejecute la aplicación. Dado que ha eliminado la vista de texto predeterminada "Hello World", debería ver el título "RecyclerView" y una vista en blanco.

Solución:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:padding="6dp">

    <TextView
        android:id="@+id/word"
        style="@style/word_title" />
</LinearLayout>
```

3.4 Crear un adapter con un view holder

Android utiliza adaptadores (de la clase Adapter) para conectar los datos con sus vistas. Hay muchos tipos diferentes de adapters disponibles. También puede escribir sus propios adapters personalizados. En esta tarea creará un adapter que asociará su lista de palabras con las vistas de elementos de la lista de palabras.

Para conectar datos con vistas, el adapter necesita conocer las vistas en las que colocará los datos. Por lo tanto, el adapter contiene un view holder (de la clase ViewHolder) la cual describe la vista de un elemento y su posición dentro del RecyclerView.

En esta tarea se construirá un adapter con un view holder que hace un puente entre los datos en su lista de palabras y el RecyclerView que la muestra:

1. Haga clic con el botón derecho en `java/com.android.example.recyclerview` y seleccione **New > Java Class**.

2. Nombre a la clase WordListAdapter.

3. Déle al WordListAdapter la siguiente firma:

```
public class WordListAdapter extends
    RecyclerView.Adapter<WordListAdapter.WordViewHolder> {}
```

   WordListAdapter extiende un adapter genérico del RecyclerView para usar un view holder específico para su aplicación y definido dentro del WordListAdapter. WordViewHolder muestra un error, porque aún no lo ha definido.

4. Haga clic en la declaración de la clase (WordListAdapter) y luego haga clic en el bombillo rojo en el lado izquierdo del panel. Seleccione **Implement methods**. Aparecerá una ventana de diálogo en la que se le pedirá que seleccione los métodos que desea implementar. Seleccione los tres métodos y haga clic en **OK**.

   Esto crea una plantilla vacía para todos los métodos que debe implementar. Observe como tanto en onCreateViewHolder como en onBindViewHolder hacen referencia al WordViewHolder, que aún no ha sido implementado.

3.5 Crear el view holder

1. Dentro de la clase WordListAdapter, agregue una nueva clase interna de WordViewHolder con esta firma:

```
class WordViewHolder extends RecyclerView.ViewHolder {}
```

2. Verá un error acerca de un constructor por defecto que falta. Puede ver detalles sobre los errores colocando el cursor sobre el código fuente subrayado en rojo o sobre cualquier línea horizontal roja en el margen derecho del panel de archivos abiertos.

3. Agregue las siguientes variables a la clase interna WordViewHolder para el text view y el adapter:

```
public final TextView wordItemView;
final WordListAdapter mAdapter;
```

4. En la clase interna WordViewHolder, agregue un constructor que inicialice el text view del view holder desde los recursos XML y configure su adapter:

```
public WordViewHolder(View itemView, WordListAdapter adapter) {
   super(itemView);
   wordItemView = (TextView) itemView.findViewById(R.id.word);
   this.mAdapter = adapter;
}
```

5. Ejecute la aplicación para asegurarse de que no tiene errores. Aún así, sólo verá una vista en blanco. Tome nota de la advertencia en el logcat: **E/RecyclerView: No adapter attached; skipping layout.**

3.6 Almacenando los datos en el adapter

1. Para mantener sus datos en el adapter, cree una linked list privada de strings en WordListAdapter y llámela mWordList.

```
private final LinkedList<String> mWordList;
```

2. Ahora puede completar el método getItemCount() para que devuelva el tamaño de mWordList.

```
@Override
public int getItemCount() {
   return mWordList.size();
}
```

A continuación, WordListAdapter necesita de un constructor que inicialice la lista de palabras. Para crear una vista para un elemento de la lista, el WordListAdapter necesita inflar el XML para un elemento de lista. Se utilizará un a layout inflater para ese trabajo. Un LayoutInflator lee una descripción XML de disposición y la convierte en las vistas correspondientes.

3. Cree una variable miembro para el *inflador* en WordListAdapter.

```
private LayoutInflater mInflater;
```

4. Implemente el constructor de WordListAdapter. Ell constructor necesita tener un parámetro de context, y una linked list de palabras con los datos de la aplicación. El método necesita instanciar un layout inflater para `mInflater` e inicializar mWordList con os datos recibidos.

```
public WordListAdapter(Context context, LinkedList<String> wordList) {
   mInflater = LayoutInflater.from(context);
```

```
    this.mWordList = wordList;
}
```

5. Complete el método onCreateViewHolder() con el siguiente código. El método onCreateViewHolder es similar al método onCreate. El Infla el layout del elemento y devuelve un view holder con layout y el adapter.

```
@Override
public WordViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    View mItemView = mInflater.inflate(R.layout.wordlist_item, parent, false);
    return new WordViewHolder(mItemView, this);
}
```

6. Complete el método onBindViewHolder con el siguiente código. El método onBindViewHolder conecta sus datos con el view holder.

```
@Override
public void onBindViewHolder(WordViewHolder holder, int position) {
    String mCurrent = mWordList.get(position);
    holder.wordItemView.setText(mCurrent);
}
```

7. Ejecute la aplicación para asegurarse de que no hay errores. Todavía verá la advertencia " E/RecyclerView: No adapter attached; skipping layout ". Esto será arreglado en la siguiente tarea.

3.7. Cree el RecyclerView en la Main Activity

Ahora que ya tiene un adapter con un view holder, puede finalmente crear un RecyclerView y conectar todas las piezas para mostrar los datos.

1. Abra el MainActivity.java

2. Agregue las variable member al MainActivity para el RecyclerView y el adapter.

```
private RecyclerView mRecyclerView;
private WordListAdapter mAdapter;
```

3. En el método onCreate de la MainActivity, agregur el siguiente código el cual crea el RecyclerView y los conecta con un adapter y los datos. Lea los comentarios del código!!, Note que debe inserter este código después de la inicialización de mWordList.

```
// Get a handle to the RecyclerView.
mRecyclerView = (RecyclerView) findViewById(R.id.recyclerview);
// Create an adapter and supply the data to be displayed.
mAdapter = new WordListAdapter(this, mWordList);
// Connect the adapter with the RecyclerView.
mRecyclerView.setAdapter(mAdapter);
```

```
// Give the RecyclerView a default layout manager.
mRecyclerView.setLayoutManager(new LinearLayoutManager(this));
```

4. Ejecute la aplicación. Debe observer una lista de palabras y podrá recorrer la lista.

Tarea 4. Hacer la lista interactiva

Mirar las listas de objetos es interesante, pero es mucho más divertido y útil si el usuario puede interactuar con ellos.

Para ver cómo el RecyclerView puede responder a las entradas del usuario, se debe adjuntar programáticamente un manejador de clic a cada elemento. Cuando se toca el elemento, se ejecuta el manejador de clics y el texto de ese elemento cambiará.

4.1. Hacer que los elementos respondan alos clics

1. Abra el archivo WordListAdapter.java.

2. Cambie la firma de la clase WordViewHolder para que implemente a View.onClickListener.

```
class WordViewHolder extends RecyclerView.ViewHolder implements
View.OnClickListener
```

3. Haga clic en el encabezado de la clase y en el bombillo rojo para implementar los métodos requeridos, que en este caso es sólo el método onClick().

4. Añada el siguiente código al cuerpo del método onClick().

```
// Get the position of the item that was clicked.
int mPosition = getLayoutPosition();
// Use that to access the affected item in mWordList.
String element = mWordList.get(mPosition);
// Change the word in the mWordList.
mWordList.set(mPosition, "Clicked! " + element);
// Notify the adapter, that the data has changed so it can
// update the RecyclerView to display the data.
mAdapter.notifyDataSetChanged();
```

5. Conecte el onClickListener con la vista. Añada este código al constructor de WordViewHolder (debajo de la línea "this.mAdapter = adapter"):

```
itemView.setOnClickListener(this);
```

6. Ejecute la aplicación. Haga clic en los elementos para ver como cambia el texto.

Tarea 5. Agregar un FAB para insertar elementos

Existen múltiples maneras de agregar comportamientos adicionales a la lista y a los elementos de la lista. Una forma es usar un botón de acción flotante (FAB). Por ejemplo, en Gmail, el FAB se

utiliza para redactar un nuevo mensaje de correo electrónico. En esta tarea implementará un FAB para añadir un elemento a la lista de palabras.

¿Por qué? Ya ha visto que puede modificar el contenido de los elementos de la lista. La lista de elementos que muestra un RecyclerView puede modificarse dinámicamente, no es sólo una lista estática de elementos.

Para esta práctica, se generará una nueva palabra que será insertada en la lista. En una aplicación más útil, se podría obtener datos de los usuarios.

5.1. Agregar un Butón de Acción Flotante (FAB)

El FAB es un control estándar de la Especificación de Material Design y forma parte de la Biblioteca de Soporte de Diseño Android. Luego encontrará más información en el capítulo sobre Material Design. Estos controles de UI tienen propiedades predefinidas. Para crear un FAB para su aplicación, añada el siguiente código dentro del layout coordinador del activity_main.xml.

```
<android.support.design.widget.FloatingActionButton
   android:id="@+id/fab"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_gravity="bottom|end"
   android:layout_margin="16dp"
   android:clickable="true"
   android:src="@drawable/ic_add_24dp" />
```

Note lo siguiente:

- **<code>@+id/fab </code>** Es la personalización para dar el id “fab” al FAB.
- **android:layout_gravity="bottom|end".** El FAB se coloca comúnmente en la parte inferior y al final del flujo de lectura/escritura.
- **android:src="@drawable/ic_add_black_24dp".** Es marcado rojo por el Android Studio debido a que el recurso no ha sido declarado.

Android proporciona una biblioteca de iconos para los iconos estándar de Android. ic_add_black_24dp es uno de los iconos estándar. Tiene que agregarlo a sus recursos drawable para usarlo.

1. Haga clic con el botón secundario en la carpeta drawable.
2. Seleccione **New > Vector Asset**
3. Asegúrese de que el **Asset Type** es **Material Icon**.
4. Haga clic en el botón del icono junto a **Icon**.
5. En la sección Contet encuentre el signo **+**. El nombre del recurso es **ic_add_black_24dp**.
6. Deje todo lo demás sin marcar y haga clic en **Next**.

7. Haga clic en **Finish**.

8. Ejecute la aplicación.

Nota: Debido a que este es un dibujo vectorial, se almacena como un archivo XML. Los dibujos vectoriales se escalan automáticamente, por lo que no es necesario tener alrededor un mapa de bits para cada resolución de pantalla. Obtenga más información Android Vector Asset Studio.

5.2. Agregar el comportamiento al FAB

En esta tarea se agregará un onClick listener al FAB para que haga lo siguiente:

- Agrega una palabra al final de la lista de palabras.
- Notifica al adapter que los datos han cambiado.
- Se desplaza hasta el elemento insertado.
- En el archivo MainActivity.java, al final del método onCreate(), agregue el código siguiente:

```
// Add a floating action click handler for creating new entries.
FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
       int wordListSize = mWordList.size();
       // Add a new word to the end of the wordList.
       mWordList.addLast("+ Word " + wordListSize);
       // Notify the adapter, that the data has changed so it can
       // update the RecyclerView to display the data.
       mRecyclerView.getAdapter().notifyItemInserted(wordListSize);
       // Scroll to the bottom.
       mRecyclerView.smoothScrollToPosition(wordListSize);
    }
});
```

- Ejecute la aplicación. Para verificar la aplicación haga lo siguiente:

    1. Desplácese por la lista de palabras.
    2. Haga clic en los elementos.
    3. Agregue los elementos haciendo clic en el FAB.
    4. ¿Qué pasa si se rota la pantalla? En una lección posterior aprenderá cómo conservar el estado de una aplicación cuando se gira la pantalla.

Resumen

- El RecyclerView es una forma eficiente de mostrar una lista desplegable de elementos.

- Para utilizar un RecyclerView, asocie los datos al Adapter/ViewHolder que cree y al layout manager de su elección.
- Se pueden crear clic listeners para detectar clics en un RecyclerView.
- Las bibliotecas de soporte para Android contienen versiones compatibles con versiones anteriores del frameword de Android.
- Las bibliotecas de soporte para Android contienen una serie de utilidades de gran utilidad para las aplicaciones.
- Las dependencias del compilación se añaden al archivo build.gradle (Module app).
- Los layouts se pueden especificar como un archivo de recursos.
- Un LayoutInflater lee un archivo de recursos de layout y crea los objetos View a partir de ese archivo.
- Un botón de acción flotante (FAB) puede modificar dinámicamente los elementos de un RecyclerView.

Capítulo 6:

Trabajando en el Background

Hay dos maneras de realizar el procesamiento en el background en Android: usando la clase `AsyncTask`, o usando el framework `Loader`, que incluye una clase `AsyncTaskLoader` que usa `AsyncTask`. En la mayoría de las situaciones se elegirá el framework `Loader`, pero es importante saber cómo funciona `AsyncTask` para que se pueda hacer una buena elección.

En este capítulo se verá por qué es importante procesar algunas tareas en el background, fuera del hilo de la UI. Veremos cómo usar `AsyncTask`, cuándo no usar `AsyncTask` y los conceptos básicos del uso de los loaders (cargadores).

El hilo UI

Cuando se inicia una aplicación Android, se crea el hilo principal, que a menudo se denomina hilo de UI. El hilo de la UI envía eventos a los widgets de la interfaz de usuario (UI) apropiada, y es donde la aplicación interactúa con los componentes del kit de herramientas de la UI de Android (componentes de los paquetes android.widget y android.view).

El modelo de hilo de Android tiene dos reglas:

1. No bloquee el hilo UI.

El hilo de UI debe prestar atención a mostrar la UI y mantener la aplicación sensible a las entradas de los usuarios. Si todo sucediera en el hilo de la UI, las operaciones largas como el acceso a la red o las consultas a la base de datos podrían bloquear toda la UI. Desde la perspectiva del usuario, la aplicación parece colgarse. Peor aún, si el hilo de la UI estuviera bloqueado durante más de unos pocos segundos (actualmente unos 5 segundos), al usuario se le presentaría el diálogo "application not responding" (ANR). El usuario podría decidir salir de la aplicación y desinstalarla.

Para asegurarse de que la aplicación no bloquea el hilo de la UI:

- Completar todo el trabajo en menos de 16 ms para cada pantalla de UI.
- No ejecutar tareas asincrónicas y otras tareas de larga duración en el hilo de la UI. En su lugar, implementar tareas en un hilo de background utilizando AsyncTask (para tareas cortas o interrumpibles) o AsyncTaskLoader (para tareas de alta prioridad, o tareas que necesitan informar al usuario o a la UI).

2. Realice el trabajo de la UI sólo en el hilo UI

No utilice un hilo de fondo para manipular la UI, debido a que el toolkit (kit de herramientas) de la UI de Android no es seguro para el trabajo entre hilos (se dice, no es thread-safe).

6. 1 Crear un AsyncTask

Un hilo es una ruta de ejecución independiente en un programa que se está ejecutando. Cuando un programa Android es iniciado, el sistema Runtime del Android crea un hilo llamado "Main". A medida que se ejecuta el programa, cada línea de código se ejecuta en serie, línea por línea. Este hilo principal es la forma en que la aplicación interactúa con los componentes del Android UI Toolkit, y es por eso que el hilo principal a veces se llama "hilo UI". Sin embargo, a veces una aplicación necesita realizar un trabajo que hace un uso intensivo de los recursos, como descargar archivos, consultas a bases de datos, reproducir medios o realizar análisis complejos. Este tipo de trabajo intensivo puede bloquear el hilo UI si todo el código se ejecuta en serie en un único hilo. Cuando la aplicación está realizando un trabajo intensivo de uso de recursos, no responde al usuario ni dibuja en la pantalla porque está esperando a que se haga ese trabajo. Esto puede producir un rendimiento deficiente, lo que afecta negativamente la experiencia del usuario. Los usuarios pueden frustrarse y desinstalar su aplicación Android si el rendimiento de la aplicación es lento.

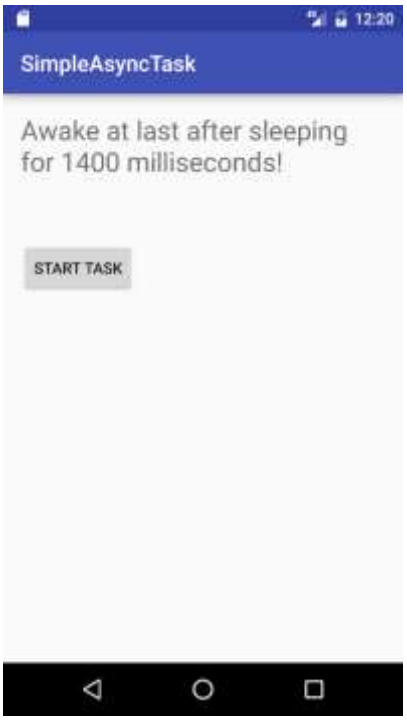
Para mantener la experiencia del usuario (UX) funcionando sin problemas y respondiendo rápidamente a los gestos del usuario, el Android Framework proporciona una clase de ayuda llamada AsyncTask que procesa el trabajo desde el hilo de la interfaz de usuario. Una AsyncTask es una clase abstracta de Java que proporciona una forma de mover este procesamiento intensivo a un subproceso separado, permitiendo así que el subproceso de la interfaz de usuario pueda seguir respondiendo. Dado que el hilo separado no está sincronizada con el hilo que lo llama, se denomina hilo asíncrono. Un AsyncTask también contiene un callback que le permite mostrar los resultados del cálculo en el hilo de la UI.

En esta práctica, aprenderá a añadir una tarea en segundo plano a una aplicación Android utilizando un AsyncTask.

Resumen de la aplicación

Usted construirá una aplicación que tiene un TextView y un botón. Cuando el usuario haga clic en el botón, la aplicación se pone a dormir durante un tiempo aleatorio y, a continuación, muestra un mensaje en el TextView cuando se despierta.

Así es como se verá la aplicación una vez terminada:



Tarea 1. Configurar el proyecto SimpleAsyncTask

La UI de SimpleAsyncTask es muy sencilla. Sólo contiene un botón que inicia el AsyncTask, y un TextView que muestra el estado de la aplicación.

1.1 Crear el layout

1. Cree un nuevo proyecto llamado SimpleAsyncTask utilizando la plantilla de Empty Activity. (Acepte los valores predeterminados para las demás opciones.)

2. Abra el archivo de diseño activity_main.xml.

    a. Cambie la vista raíz a LinearLayout.

    b. En el elemento "Hello World" TextView, elimine los atributos `layout_constraint`, si están presentes.

    c. Añada los siguientes elementos esenciales de la UI al layout:

View	Atributos	Vallores
LinearLayout	android:orientation	vertical
TextView	android:text android:id	I am ready to start work! @+id/textView1
Button	android:text android:onClick	Start Task startTask

Nota: Puede establecer la altura y el ancho de la presentación de cada vista como desee, siempre que las vistas permanezcan en la pantalla independientemente del tamaño de la pantalla (utilizando `wrap_content` se asegura este caso).

3. El atributo onClick del botón se resaltará en amarillo, ya que el método `startTask()` aún no está implementado en el MainActivity. Sitúe el cursor en el texto resaltado, pulse **Alt + Enter** (**Opción + Enter** en un Mac) y seleccione **Create 'startTask(View) in 'MainActivity'** para crear la plantilla del método en el MainActivity.

Dependiendo dela version del Android Studio, el archivo layout de activity_main.xml lucirá como el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:orientation="vertical">

   <TextView
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/ready_to_start"
       android:id = "@+id/textView1"
       android:textSize="24sp"/>

   <Button
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/start_task"
       android:id="@+id/button"
       android:layout_marginTop="56dp"
       android:onClick="startTask" />
</LinearLayout>
```

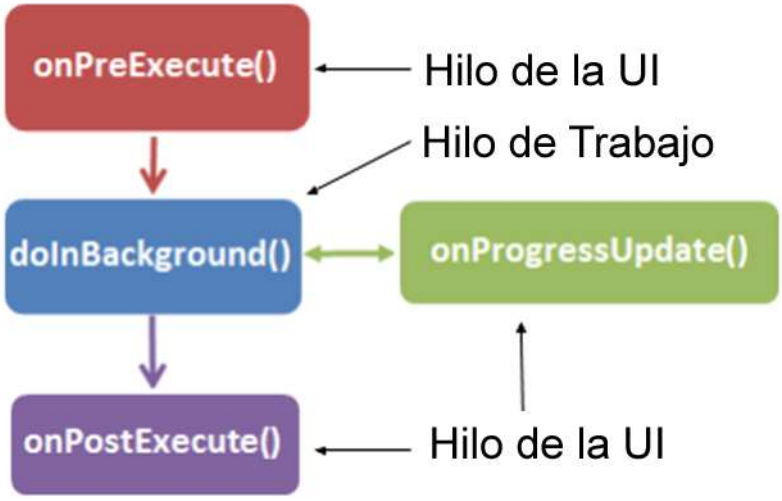
Tarea 2. Crear la subclase de AsyncTask

Como AsyncTask es una clase abstracta, se necesitas extenderla para poder usarla. En este ejemplo, el AsyncTask ejecutará una tarea en segundo plano muy simple: sólo hará que se duerma durante una cantidad de tiempo aleatoria. En una aplicación real, la tarea de background podría realizar todo tipo de trabajo, desde consultar una base de datos, conectarse a Internet, hasta calcular el siguiente movimiento de Go para poder vencer al actual campeón de Go.

Un AsyncTask tiene los siguientes métodos para realizar los trabajos fuera del hilo principal:

- `onPreExecute()`: Este método se ejecuta en hilo de la UI y se utiliza para configurar su tarea (como mostrar una barra de progreso).

- `doInBackground()`: Aquí es donde se implementa el código para ejecutar el trabajo que se debe realizar en el hilo separado.
- `onProgressUpdate()`: Se invoca en el hilo de la UI y se utiliza para actualizar el progreso en la UI (como rellenar una barra de progreso).
- `onPostExecute()`: De nuevo en el hilo de la UI, se utiliza para actualizar los resultados en la UI una vez que AsyncTask ha terminado de cargar.



Nota: Un trabajo en segundo plano o hilo de trabajo es cualquier hilo que no sea el principal o el hilo de la UI.

Al crear un AsyncTask, es posible que se deba proporcionarle información sobre el trabajo que debe realizar, si debe informar de su progreso y cómo hacerlo, y en qué forma debe devolver el resultado.

En este ejercicio se usará una subclase de AsyncTask para definir el trabajo que se ejecutará en un hilo diferente al hilo de la UI, lo que evitará cualquier problema de rendimiento.

Al crear un AsyncTask, se puede configur utilizando estos parámetros:

- Params: El tipo de datos de los parámetros enviados a la tarea al ejecutar el método sobreescrito `doInBackground()`.
- Progress: El tipo de datos de las unidades de progreso publicadas utilizando el método sobreescrito `onProgressUpdated()`.
- Result: El tipo de datos del resultado entregado por el método sobrescrito `onPostExecute()`.

Por ejemplo, un AsyncTask con la siguiente declaración de clase tomaría un String como parámetro en `doInBackground()` (para ser usado en una consulta, por ejemplo), un Integer para el `onProgressUpdate()` (porcentaje del trabajo completado), y un Bitmap para el resultado en el `onPostExecute()` (el resultado de la consulta):

```
public class MyAsyncTask extends AsyncTask <String, Integer, Bitmap>{}
```

2.1 Extendiendo el AsyncTask

En la primera implementación de AsyncTask, la subclase de AsyncTask será muy simple. No requiere ningún parámetro de consulta ni tampoco publica su progreso. Sólo se utilizarán los métodos doInBackground() y `onPostExecute()`.

1. Cree una nueva clase Java llamada SimpleAsyncTask que extienda a AsyncTask y que tome tres parámetros de tipo genérico:
    - Void para los parámetros, ya que este AsyncTask no requiere ninguna entrada.
    - Vacío para el tipo de progreso, ya que el progreso no se publica.
    - Un string como para el tipo de resultado, ya que actualizará la TextView con un string cuando la ejecución del AsyncTask haya finalizado.

    ```
    public class SimpleAsyncTask extends AsyncTask <Void, Void, String>{}
    ```

    Nota: La declaración de clase se subrayará en rojo, ya que el método requerido `doInBackground()` aún no se ha implementado.

    El AsyncTask necesitará actualizar el TextView una vez que haya terminado de dormir. El constructor necesitará entonces incluir el TextView, para que pueda ser actualizado en el `onPostExecute()`.

2. Defina una variable member mTextView.
3. Implemente un constructor para la AsyncTask que tome un TextView y asígnele al mTextView el recibido en el constructor:

```
public SimpleAsyncTask(TextView tv) {
    mTextView = tv;
 }
```

2.2 Implementar doInBackground()

1. Añada el método `doInBackground()` requerido. Coloque el cursor en la declaración de clase resaltada, pulse **Alt + Enter** (**Opción + Enter** en un Mac) y seleccione Implementar los métodos. Seleccione `doInBackground()` y haga clic en **OK**:

```
@Override
protected String doInBackground(Void... voids) {
   return null;
}
```

2. Implemente doInBackground() para que:
    - Genere un valor entero aleatorio entre 0 y 10
    - Multiplique ese número por 200
    - Coloque dormer el hilo actual. (Use `Thread.sleep()`) en un bloque try/catch.
    - Retorne the String "Awake at last after xx milliseconds" (donde xx es el número de milisegundos que durmió la aplicación)

```
@Override
protected String doInBackground(Void... voids) {

   // Generate a random number between 0 and 10
   Random r = new Random();
   int n = r.nextInt(11);

   // Make the task take long enough that we have
   // time to rotate the phone while it is running
   int s = n * 200;

   // Sleep for the random amount of time
   try {
       Thread.sleep(s);
   } catch (InterruptedException e) {
       e.printStackTrace();
   }

   // Return a String result
   return "Awake at last after sleeping for " + s + " milliseconds!";
}
```

2.3 Implemente onPostExecute()

Cuando el método `doInBackground()` se complete, el valor de retorno se pasa automáticamente al callback `onPostExecute()`.

1. Implemente `onPostExecute()` para que tome un argumento String (esto es lo que se define en el tercer parámetro del AsyncTask y lo que el método `doInBackground()` devuelve) y muestre ese string en el TextView:

```
protected void onPostExecute(String result) {
   mTextView.setText(result);
}
```

Nota: Puede actualizar la UI en el `onPostExecute()` porque se ejecuta en el hilo principal (UI). No se puede llamar `mTextView.setText()` en el `doInBackground()`, porque ese método se ejecuta en un hilo separado.

Tarea 3. Implementar los pasos finales

3.1 Implemente los métodos que inician el AsyncTask

La aplicación tiene ahora un AsyncTask que realiza el trabajo en segundo plano (o lo haría si no llamara a sleep() como el trabajo simulado.) Ahora se puede implementar el método que es llamado cuando se hace clic en el botón Start Task, para activar la tarea en background.

1. En el archivo MainActivity.java, agregue una variable miembro para almacenar el TextView.

```
private TextView mTextView;
```

2. En el método `onCreate()`, inicialice `mTextView` como un TextView en la UI.
3. Añada el código al método `startTask()` para crear una instancia de `SimpleAsyncTask`, pasando el TextView `mTextView` al constructor.
4. Llame al `execute()` en esa instancia de `SimpleAsyncTask`.

Nota: El método `execute()` es donde se pasan los parámetros (separados por comas) que luego el sistema pasa a `doInBackground()`. Como este AsyncTask no tiene parámetros, lo dejará en blanco.

5. Actualize el TextView para que muestre el texto "Napping…"

```
public void startTask (View view) {
   // Put a message in the text view
   mTextView.setText("Napping... ");

   // Start the AsyncTask.
   // The AsyncTask has a callback that will update the text view.
   new SimpleAsyncTask(mTextView).execute();
}
```

Código de la solución para MainActivity:

```
package android.example.com.simpleasynctask;

import android.os.Bundle;
```

```
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

   // The TextView where we will show results
   TextView mTextView;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
       //  Initialize mTextView
       mTextView = (TextView) findViewById(R.id.textView1);

   }

   public void startTask (View view) {
       // Put a message in the text view
       mTextView.setText("Napping... ");

       // Start the AsyncTask.
       // The AsyncTask has a callback that will update the text view.
       new SimpleAsyncTask(mTextView).execute();
   }
}
```

3.2 Implemente onSaveInstanceState()

1. Ejecute la Aplicación y haga clic en el botón **Start Task**. ¿Cuánto tiempo la aplicación está de siesta?

2. Haga clic de nuevo en el botón **Start Task** y, mientras la aplicación está durmiendo, gire el dispositivo. Si la tarea en segundo plano se completa antes de que pueda rotar el teléfono, inténtelo de nuevo. Alternativamente, puede actualizar el código y hacerlo dormir durante un período de tiempo más largo.

Nota: Notará que cuando se gira el dispositivo, el TextView se restablece a su contenido inicial y el AsyncTask no parece capaz de actualizar el TextView.

There are several things going on here:

- When you rotate the device, the system restarts the app, calling onDestroy() and then onCreate(), which restarts the activity lifecycle. Since the AsyncTasks are no longer connected to the lifecycle of your app, and cannot reconnect to the activity.
- The AsyncTasks will continue running to completion in the background, consuming system resources, but never showing the results in the UI, which gets reset in onCreate(). It will never be able to update the TextView that was passed to it, since

that particular TextView has also been destroyed. Eventually, the system run out of resources, and will fail.

- Even without the AsyncTask, the rotation of the device resets all of the UI elements to their default state, which for the TextView implies a particular string that you set in the activity_main.xml file.

For these reasons, AsyncTasks are not well suited to tasks which may be interrupted by the destruction of the Activity. In use cases where this is critical you can use a different type of class called a Loader, which you will implement in a later practical.

In order to prevent the TextView from resetting to the initial string, you need to save its state. You've already learned how to maintain the state of views in a previous practical, using the SavedInstanceState class.

You will now implement onSaveInstanceState() to preserve the content of your TextView when the activity is spontaneously destroyed.

Note: Not all uses of AsyncTask require you to handle the state of the views on rotation. This app uses a TextView to display the results of the app, so preserving the state is useful. In other cases, such as uploading a file, you may not need any persistent information in the UI, so retaining the state is not critical.

Aquí están pasando varias cosas:

- Al rotar el dispositivo, el sistema reinicia la aplicación, llamando a `onDestroy()` y luego a `onCreate()`, que reinicia el ciclo de vida de la actividad. Dado que las AsyncTasks no están conectadas al ciclo de vida de la aplicación no pueden volverse a conectar a la actividad.
- Los AsyncTasks continuarán ejecutándose hasta completarse en segundo plano, consumiendo recursos del sistema, pero nunca mostrando los resultados en la interfaz de usuario, el cual se reinicia en `onCreate()`. Nunca podrá actualizar el TextView que le fue pasado, ya que ese TextView en particular también ha sido destruido. Eventualmente, el sistema se queda sin recursos, y fallará.
- Incluso sin AsyncTask, la rotación del dispositivo resetea todos los elementos de la interfaz de usuario a su estado por defecto, lo que para el TextView implica un string particular que se configura en el archivo activity_main.xml.

Por estas razones, los AsyncTasks no son adecuados para tareas que pueden ser interrumpidas por la destrucción de la Activity. En los casos de uso en los que esto es crítico, puedes usar un tipo diferente de clase llamada Loader, la cual implementará en una práctica posterior.

Para evitar que el TextView se restablezca al string inicial, es necesario guardar su estado. Ya ha aprendido a mantener el estado de las vistas en una práctica anterior, usando la clase SavedInstanceState.

Ahora implementará `onSaveInstanceState()` para preservar el contenido de su TextView cuando la actividad sea destruida espontáneamente.

Nota: No todos los usos de AsyncTask requieren que maneje el estado de las vistas durante la rotación. Esta aplicación utiliza un TextView para mostrar los resultados de la aplicación, por lo que es útil preservar el estado. En otros casos, como al subir un archivo, es posible que no necesite información persistente en la interfaz de usuario, por lo que conservar el estado no es crítico.

3. Sobreescriba el método `onSaveInstanceState()` en la MainActivity para preservar el texto dentro del TextView cuando la actividad sea destruida:

```
outState.putString(TEXT_STATE, mTextView.getText().toString());
```

4. Recupere el valor del TextView cuando la actividad sea restaurada en el método `OnCreate()`.

```
// Restore TextView if there is a savedInstanceState
if(savedInstanceState!=null){
  mTextView.setText(savedInstanceState.getString(TEXT_STATE));
}
```

Código de la solución para la MainActivity:

```
package android.example.com.simpleasynctask;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.TextView;

/**
 * The SimpleAsyncTask app contains a button that launches an AsyncTask
 * which sleeps in the asynchronous thread for a random amount of time.
 */
public class MainActivity extends AppCompatActivity {

    //Key for saving the state of the TextView
    private static final String TEXT_STATE = "currentText";

    // The TextView where we will show results
    private TextView mTextView = null;

    /**
     * Initializes the activity.
     * @param savedInstanceState The current state data
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //  Initialize mTextView
```

```
        mTextView = (TextView) findViewById(R.id.textView1);

        // Restore TextView if there is a savedInstanceState
        if(savedInstanceState!=null){
           mTextView.setText(savedInstanceState.getString(TEXT_STATE));
        }
    }

    /**`
     * Handles the onCLick for the "Start Task" button. Launches the AsyncTask
     * which performs work off of the UI thread.
     *
     * @param view The view (Button) that was clicked.
     */
    public void startTask (View view) {
        // Put a message in the text view
        mTextView.setText(R.string.napping);

        // Start the AsyncTask.
        // The AsyncTask has a callback that will update the textview.
        new SimpleAsyncTask(mTextView).execute();
    }

    /**
     * Saves the contents of the TextView to restore on configuration change.
     * @param outState The bundle in which the state of the activity is saved
when it is spontaneously destroyed.
     */
    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        // Save the state of the TextView
        outState.putString(TEXT_STATE, mTextView.getText().toString());
    }
}
```

Resumen

- Evite el trabajo intensivo con el uso de recursos en el hilo de la UI, ya que puede hacer que la UI sea lenta o errática.
    - Cualquier código que no involucre trabajar en la UI o responda a una entrada de usuario debe ser removido del hilo de la UI a otro hillo separado.
- Un AsyncTask es una clase abstracta de Java la cual mueve el procesamiento intensive a un hilo separado.
    - AsyncTask debe ser extendido para poder ser usado.
    - AsyncTask tiene 4 métodos muy útiles: `onPreExecute()`, `doInBackground()`, `onPostExecute()` y `onProgressUpdate()`.

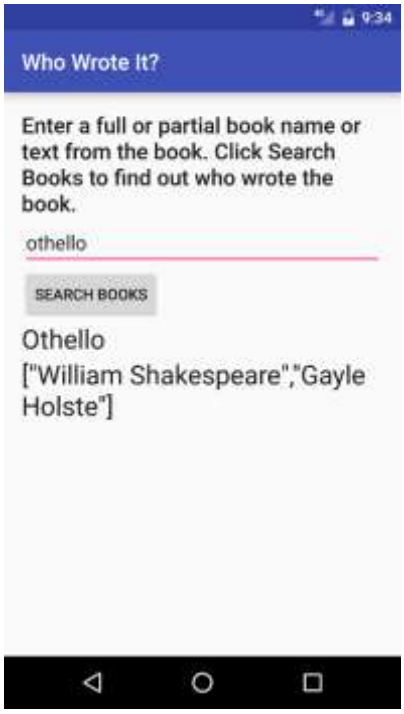
- `doInBackground()` es el único método que se ejecuta en un hilo de trabajo separado.
    - No se debe hacer llamados a métodos de la UI en métodos de la AsyncTask.
    - Los otros métodos de la AsyncTask son ejecutados en el hilo de la UI y permiten llamar a métodos de los componentes de la UI.
- Al rotar un dispositivo Android se destruye y se recrea una Actividad. Esto puede disacionar la UI del hilo en segundo plano, el cual continuará trabajando.

6.2 Conectarse a la Internet con AsynTask y AsyncTaskLoader

En esta práctica usará un AsyncTask para iniciar una tarea en segundo plano que obtiene datos de Internet usando una simple API REST. Utilizará el Google API Explorer para aprender a consultar la API de Búsqueda de Libros, implementará esta consulta en un hilo de trabajado mediante AsyncTask y mostrará el resultado en la UI. A continuación, volverá a implementar la misma tarea de segundo plano pero utilizando un AsyncTaskLoader, el cual será más eficiente al momento de actualizar la UI, gestionar los problemas de rendimiento y mejorar la UX en general.

Resumen de la aplicación

Construirá una aplicación que contenga un campo EditText y un Button. El usuario introduce el nombre de un libro en el campo EdtText y hace clic en el botón. El botón ejecuta un AsyncTask el cual consulta la Google Book Search API para encontrar el autor y el título del libro que el usuario está buscando. Los resultados son recuperados y son mostrados en un campo TextView debajo del botón. Una vez que la aplicación esté funcionando, modificará la aplicación para usar un AsyncTaskLoader en lugar de la clase AsyncTask.



Tarea 1. Explorar el API de Libros

En esta práctica, utilizará la API de la Búsqueda de libros de Google (Google Books API) para buscar información sobre un libro, tal como el autor o los autores y el título. La Google Books API proporciona acceso programático al servicio de Búsqueda de libros de Google mediante un API REST. Se trata del mismo servicio que se utiliza entre bastidores cuando se ejecuta manualmente una búsqueda en Google Books. Puede utilizar el Google API Explorer y el Google Book Search en su navegador para comprobar que su aplicación Android está obteniendo los resultados esperados.

1.1 Enviar un requerimiento al Books API

1. Vaya al Google APIs Explorer (encuentrelo en https://developers.google.com/apis-explorer/).
2. Haga clic en **Books API**.
3. Encuentre (**Ctrl-F** or **Cmd-F**) **books.volumes.list** y haga clic en el nombre de esa función. Debe ver una página web con la lista de los parámetros que usa la función del Books API para realizar la búsqueda de libros.
4. En el campo `q` introduzca el nombre completo o parcial de un libro. El parámetro `q` es el único campo requerido.
5. Use los campos `maxResults` y `printType` para limitar los resultados a 10 coincidencias con libros que fueron impresos. El campo maxResults toma un valor entero que limita la cantidad de resultados por query (consulta). El campo `printType` toma uno de tres strings como parámetros: `all`, el cual no limita el resultado por tipo; `books`, el cual retorna solamente libros impresos; y `magazines` el cual solo retorna revistas.
6. Asegúrese que el switch "Authorize requests using OAuth 2.0" en la parte superior de la forma esté apagado. Haga clic en **Execute without OAuth** en la parte inferior de la forma.
7. Desplácese hacia abajo para ver el Requerimiento (Request) y la Respuesta (Response).

El campo `Request` es un ejemplo de un Uniform Resource Identifier (URI). Un URI es un string que nombra o localiza un recurso en particular. Las URL son un cierto tipo de URI para identificar y localizar un recurso web. Para la API de Books, el request es una URL que contiene la búsqueda como parámetro (siguiente el parámetro `q`). Observe el campo `key` de la API después del campo query. Por razones de seguridad, cuando se accede a una API pública, normalmente se necesita obtener una clave del API e incluirla en el Request. Sin embargo, esta API específica no requiere una clave, por lo que puede omitir esa parte del URI del Request en la aplicación.

1.2 Analize el Response del Books API

Hacia la parte inferior de la página se puede ver la Respuesta (el Response) a la consulta. La respuesta utiliza el formato JSON, que es un formato común para las respuestas de consulta de una API. En la página web del Explorador de la API, el código JSON está formateado para que sea legible por el ser humano. En la aplicación, la respuesta JSON será devuelta desde el servicio del API como un simple string, y necesitará analizar esa cadena para extraer la información que necesita.

1. En la sección Response, encuentre el valor de la clave “title”. Es de hacer notar que este resultado tiene una única clave y valor.

2. Encuentre el valor para la clave “authors”. Note que esta puede tener un arreglo de valores.

3. En esta práctica, solamente retornará el título y los autores del primer elemento.

Tarea 2. Crear la aplicación "Who Wrote It?"

Ahora que está familiarizado con el método de la Books API que estará usando, es momento de configurar el layout de la aplicación.

2.1 Crear el proyecto y la interfaz de usuario

1. Cree un Proyecto de aplicación llamado **Who Wrote It?** Con una Activity, usando la plantilla Empty Activity.

2. Agregue los siguientes elementos de UI en el archivo XML, utilizando un LinearLayout vertical como vista raíz, la vista que contendrá todas las demás vistas dentro del archivo XML de layout. Asegúrese de que LinearLayout utiliza `android:orientation="vertical"`:

View	Atributos	Valores
TextView	android:layout_width android:layout_height android:id android:text android:textAppearance	wrap_content wrap_content @+id/instructions @string/instructions @style/TextAppearance.AppCompat.Title
EditText	android:layout_width android:layout_height android:id android:inputType android:hint	match_parent wrap_content @+id/bookInput text @string/input_hint
Button	android:layout_width android:layout_height android:id android:text android:onClick	wrap_content wrap_content @+id/searchButton @string/button_text searchBooks
TextView	android:layout_width android:layout_height android:id android:textAppearance	wrap_content wrap_content @+id/titleText @style/TextAppearance.AppCompat.Headline
TextView	android:layout_width android:layout_height android:id android:textAppearance	wrap_content wrap_content @+id/authorText @style/TextAppearance.AppCompat.Headline

3. En el archivo strings.xml, agregue estos recursos de string:

```
<string name="instructions">Enter a book name, or part of a
<string name="instructions">Enter a book name, or part of a
book name, or just some text from a book to find
the full book title and who wrote the book!</string>
<string name="button_text">Search Books</string>
```

4. Cree un método llamado `searchBooks()` en MainActivity.java para manejar la acción onClick del botón. Como con todos los métodos onClick, éste toma una `View` como parámetro.

2.2 Configure la Main Activity

Para consultar la Books API, necesita obtener la entrada del usuario con un EditText.

1. En el archivo MainActivity.java, cree las variables member para EditText, el autor TextView y el título TextView.

2. Inicialice estas variables en el método `onCreate()`.

3. En el método `searchBooks()`, obtenga el texto del widget EditText y conviértalo en un string, asignándolo a una variable de tipo string.

```
String queryString = mBookInput.getText().toString();
```

Nota: `mBookInput.getText()` retorna un tipo de datos "Editable" el cual necesita ser convertido a String.

2.3 Crear un AsyncTask vacío

Ahora está listo para conectarse a Internet y utilizar la Book Search REST API. La conectividad a la red puede ser a veces lenta o experimentar retrasos. Esto puede hacer que la aplicación se comporte de forma errática o que se vuelva lenta, por lo que no se debe realizar una conexión de red en el hilo de la UI. Si intenta hacer una conexión de red en el hilo de la UI, el sistema de Runtime del Android puede lanzar una NetworkOnMainThreadException para advertirle que eso es una mala idea.

Utilice un AsyncTask para realizer las conecciones a la red:

1. Cree una nueva clase Java llamada `FetchBook` en **app/java** que extienda `AsyncTask`. Un AsyncTask requiere tres argumentos:
    - Los parámetros de entrada.
    - El indicador de progreso.
    - El tipo de resultado.

Los parámetros de tipo genérico para la tarea serán `<String, Void, String>` esto debido a que el AsyncTask toma un `String` como el primer parámetro (el query), `Void` ya que no habrá actualización de progreso, y `String` debido a que retorna un string como resultado (la respuesta JSON).

2. Implemente el método requerido, `doInBackground()`, colocando el cursor sobre el texto subrayado en rojo, pulsando **Alt + Enter** (**Opt + Enter** en un Mac) y seleccionando **Implement methods**. Seleccione **doInBackground()** y haga clic en **OK**. Asegúrese de que los parámetros y los tipos de retorno son los correctos (toma un arreglo de String y devuelve un string).

    a. Haga clic en el menu **Code** y seleccione **Override methods** (o presione **Ctrl + O**). Seleccione el método **onPostExecute()**. El método `onPostExecute()` toma un `String` como parámetro y retorna `void`.

3. Para mostrar los resultados en las TextViews, debe tener acceso a esas TextViews dentro del AsyncTask. Para ello cree variables member en el FetchBook AsyncTask para las dos TextViews que muestran los resultados e inicialícelas en un constructor. Usará este constructor en su MainActivity para pasar las TextViews al AsyncTask.

Código de la solución para FetchBook:

```
public class FetchBook extends AsyncTask<String,Void,String>{
        private TextView mTitleText;
        private TextView mAuthorText;

   public FetchBook(TextView mTitleText, TextView mAuthorText) {
       this.mTitleText = mTitleText;
       this.mAuthorText = mAuthorText;
   }

   @Override
   protected String doInBackground(String... params) {
       return null;
   }

   @Override
   protected void onPostExecute(String s) {
       super.onPostExecute(s);
   }
}
```

2.4 Crear la clase NetworkUtils y construer el URI

En este paso, abrirá una conexión a Internet y consultará la Books API. Esta sección tiene bastante código, así que recuerde visitar la documentación del desarrollador para Connecting to the Network si se quedas atascado. Escribirá el código para conectarse a Internet en una clase helper llamada NetworkUtils.

1. Cree una nueva clase Java llamada NetworkUtils haciendo clic en **File > New > Java Class** y completando únicamente el campo "Nombre".

2. Cree una variable final `LOG_TAG` para ser usada en toda la clase NetworkUtils para el logging:

```
private static final String LOG_TAG = NetworkUtils.class.getSimpleName();
```

3. Cree un método estático nuevo llamado `getBookInfo()` el cual toma un `String` como parámetro (el cual será el término de búsqueda) y devuelve un `String` (la respuesta String en JSON de la API que examinó anteriormente).

```
static String getBookInfo(String queryString){}
```

4. Cree las siguientes dos variables locales en `getBookInfo()` que se necesitarán más adelante para ayudar a conectar y leer los datos entrantes.

```
HttpURLConnection urlConnection = null;
BufferedReader reader = null;
```

5. Cree otra variable local al final de `getBookInfo()` para contener la respuesta cruda de la consulta y devolverla:

```
String bookJSONString = null;
return bookJSONString;
```

Si recuerda el request de la página web del Books API, notará que todas las solicitudes comienzan con la misma URI. Para especificar el tipo de recurso, añada parámetros de consulta a la URI base. Es práctica común separar todos estos parámetros de consulta en constantes, y combinarlos usando un Uri.Builder para que puedan ser reutilizados por diferentes URI's. La clase Uri tiene un método conveniente, `Uri.buildUpon()` que devuelve un URI.Builder el cual podemos usar.

Para esta aplicación, limitará el número y tipo de resultados devueltos para aumentar la velocidad de la consulta. Para restringir la consulta, sólo buscará libros que estén impresos.

6. Cree las siguientes constants miembros en la clase NetworkUtils:

```
private static final String BOOK_BASE_URL =
"https://www.googleapis.com/books/v1/volumes?"; // Base URI for the Books API
private static final String QUERY_PARAM = "q"; // Parameter for the search string
private static final String MAX_RESULTS = "maxResults"; // Parameter that limits
                                                        // search results
private static final String PRINT_TYPE = "printType";   // Parameter to filter by
                                                        // print type
```

7. Cree la plantilla de un bloque try/catch/finally en `getBookInfo()`. Aquí es donde realizará el request HTTP. El código para construir el URI y emitir la consulta irá en el bloque try. El bloque catch se utiliza para tratar cualquier problema que surja al realizar la solicitud HTTP y, el bloque

finally sirve para cerrar la conexión de la red una vez que haya terminado de recibir los datos JSON y devolver el resultado.

```
try {
...
} catch (Exception ex) {
...
} finally {
   return bookJSONString;
}
```

8. Construya el request URI en el bloque try:

```
//Build up your query URI, limiting results to 10 items and printed books
Uri builtURI = Uri.parse(BOOK_BASE_URL).buildUpon()
       .appendQueryParameter(QUERY_PARAM, queryString)
       .appendQueryParameter(MAX_RESULTS, "10")
       .appendQueryParameter(PRINT_TYPE, "books")
       .build();
```

9. Convierta el URI en un URL:

```
URL requestURL = new URL(builtURI.toString());
```

2.5 Hacer el Request

Es bastante común hacer una solicitud a un API a través de Internet. Dado que es probable que vuelva a utilizar esta funcionalidad, es posible que desee crear una clase de utilidad con esta funcionalidad o desarrollar una subclase útil para su propia conveniencia. Este request API utiliza la clase HttpURLConnection en combinación con un InputStream y un StringBuffer para obtener la respuesta JSON desde la web. Si en algún momento el proceso falla el InputStream o el StringBuffer están vacíos, o devuelve nulo, significa que la consulta falló.

1. En el bloque try del método `getBookInfo()`, abra la conexión URL y haga el request:

```
urlConnection = (HttpURLConnection) requestURL.openConnection();
urlConnection.setRequestMethod("GET");
urlConnection.connect();
```

2. Lea el response usando un InputStream y un StringBuffer, luego conviertalo en un String:

```
InputStream inputStream = urlConnection.getInputStream();
StringBuffer buffer = new StringBuffer();
if (inputStream == null) {
   // Nothing to do.
   return null;
```

```
}
reader = new BufferedReader(new InputStreamReader(inputStream));
String line;
while ((line = reader.readLine()) != null) {
   /* Since it's JSON, adding a newline isn't necessary (it won't affect
      parsing) but it does make debugging a *lot* easier if you print out the
      completed buffer for debugging. */
   buffer.append(line + "\n");
}
if (buffer.length() == 0) {
   // Stream was empty.  No point in parsing.
   return null;
}
bookJSONString = buffer.toString();
```

3. Cierre el bloque try y registre la excepción en el bloque catch.

```
catch (IOException e) {
   e.printStackTrace();
   return null;
}
```

4. Cierre tanto la urlConnection como las variables del reader en el bloque finally:

```
finally {
   if (urlConnection != null) {
       urlConnection.disconnect();
   }
   if (reader != null) {
       try {
           reader.close();
       } catch (IOException e) {
           e.printStackTrace();
       }
   }
}
```

Nota: Cada vez que la conexión falla, este código devuelve nulo. Esto significa que `onPostExecute()` tendrá que comprobar que su parámetro de entrada no sea un string nulo y hacerle saber al usuario que la conexión falló. Esta estrategia de manejo de errores es simplista, ya que el usuario no tiene idea del por qué falló la conexión. Una mejor solución para una aplicación de producción sería manejar cada punto de falla de manera diferente para que el usuario pueda obtener la retroalimentación apropiada.

5. Registre en el Log el valor de la variable `bookJSONString` antes de retornarla. Esto es todo con el método `getBookInfo()`.

```
Log.d(LOG_TAG, bookJSONString);
```

6. En el método `doInBackground()` del AsyncTask, llame al método `getBookInfo()`, pasando el término de búsqueda que obtuvo del argumento params pasado por el sistema (es el primer valor en el arreglo de params). Devuelva el resultado de este método en el método `doInBackground()`:

```
return NetworkUtils.getBookInfo(params[0]);
```

7. Ahora que el AsyncTask está configurado, necesita lanzarlo desde la MainActivity usando el método `execute()`. Agregue el siguiente código al método `searchBooks()` en el archivo MainActivity.java para iniciar el AsyncTask:

```
new FetchBook(mTitleText, mAuthorText).execute(mQueryString);
```

8. Ejecute la aplicación. Realize una búsqueda. La aplicación fallará. Observe los Logs para ver que está causando el error. Debe ver una línea como esta:

```
Caused by: java.lang.SecurityException: Permission denied (missing INTERNET
permission?)
```

Este error indica que no se ha incluido el permiso para acceder a Internet en el archivo AndroidManifest.xml. La conexión a Internet introduce nuevos problemas de seguridad, por lo que sus aplicaciones no tienen conectividad de forma predeterminada. Debe agregar esos permisos manualmente en forma de una etiqueta `<uses-permission>` en el AndroidManifest.xml.

2.6 Agregar los permisos de Internet

1. Abra el archivo **AndroidManifest.xml**.

2. Todos los permisos de la aplicación deben estar en el archivo AndroidManifest.xml fuera de la etiqueta <application>. Debe asegurarse de seguir el orden en el que se definen las etiquetas en AndroidManifest.xml.

3. Añada las siguientes etiquetas xml fuera de la etiqueta <application>:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

4. Construya la aplicación y ejecutela nuevamente. Al realizar una consulta debe resultar ahora en un string JSON que ha sido impreso en el Log.

2.7 Analizar el string JSON

Ahora que tiene la respuesta correcta a su consulta, debe analizar los resultados para extraer la información que desea mostrar en la UI. Afortunadamente, Java tiene clases existentes que ayudan en el análisis y manejo de datos de tipo JSON. Este proceso, así como la actualización de la UI, tendrá lugar en el método `onPostExecute()`.

Existe la posibilidad de que el método `doInBackground()` no devuelva la cadena JSON esperada. Por ejemplo, el try catch puede fallar y arrojar una excepción, el tiempo de espera de la red se haya excedido u otros errores no manejados puedan ocurrir. En esos casos, los métodos Java JSON fallarán al analizar los datos y arrojarán excepciones. Esta es la razón por la que tiene que hacer el análisis en el bloque try, y el bloque catch debe manejar el caso donde se devuelven datos incorrectos o incompletos.

Para analizar los datos JSON y manejar posibles excepciones, haga lo siguiente:

1. En el método `onPostExecute()`, agregue un bloque try/catch debajo a la llamada a super.

2. Use las clase propias de Java (`JSONObject` y `JSONArray`) para obtener el arreglo JSON de los items del resultado en el bloque try.

```
JSONObject jsonObject = new JSONObject(s);
JSONArray itemsArray = jsonObject.getJSONArray("items");
```

3. Haga una iteración a través de los `ítemsArray`, revisando cada libro para ver el título y la información del autor. Si ambos no son nulos, salga del bucle y actualice la UI; de lo contrario, continúe buscando en la lista. De esta manera, sólo se mostrarán las entradas con un título y autores.

```
//Iterate through the results
for(int i = 0; i<itemsArray.length(); i++){
   JSONObject book = itemsArray.getJSONObject(i); //Get the current item
   String title=null;
   String authors=null;
   JSONObject volumeInfo = book.getJSONObject("volumeInfo");

   try {
       title = volumeInfo.getString("title");
       authors = volumeInfo.getString("authors");
   } catch (Exception e){
       e.printStackTrace();
   }

   //If both a title and author exist, update the TextViews and return
   if (title != null && authors != null){
       mTitleText.setText(title);
       mAuthorText.setText(authors);
       return;
   }
}
```

4. Si no hay resultados que cumplan los criterios de tener un autor y un título válidos, y el bucle se ha detenido, ajuste el TextView título para que diga "No Results Found", y limpie el TextView de los autores.

5. En el bloque catch, imprima el error en el log, ponga el TextView título en "No Results Found", y limpie el TextView de los autores.

Código de la solución:

```
//Method for handling the results on the UI thread
@Override
protected void onPostExecute(String s) {
    super.onPostExecute(s);
    try {
       JSONObject jsonObject = new JSONObject(s);
       JSONArray itemsArray = jsonObject.getJSONArray("items");
       for(int i = 0; i<itemsArray.length(); i++){
           JSONObject book = itemsArray.getJSONObject(i);
           String title=null;
           String authors=null;
           JSONObject volumeInfo = book.getJSONObject("volumeInfo");

           try {
               title = volumeInfo.getString("title");
               authors = volumeInfo.getString("authors");
           } catch (Exception e){
               e.printStackTrace();
           }


           if (title != null && authors != null){
               mTitleText.setText(title);
               mAuthorText.setText(authors);
               return;
           }
       }

       mTitleText.setText("No Results Found");
       mAuthorText.setText("");

   } catch (Exception e){
       mTitleText.setText("No Results Found");
       mAuthorText.setText("");
       e.printStackTrace();
   }
}
```

Tarea 4. Implementar las Mejores Prácticas para la UI

Ahora tiene una aplicación funcional que utiliza la Books API para ejecutar una búsqueda de libros. Sin embargo, hay algunas cosas que no se comportan como se espera:

- Cuando el usuario hace clic en **Search Books**, el teclado no desaparece, y no hay ninguna indicación para el usuario de que la consulta se está ejecutando realmente.
- Si no hay conexión de red, o el campo de búsqueda está vacío, la aplicación sigue intentando consultar la API y falla sin actualizar correctamente la UI.

- Si gira la pantalla durante una consulta, AsyncTask se desconecta de la Actividad y no puede actualizar la UI con los resultados.

Procederemos a corregir estos problemas a continuación.

3.1 Ocultar el Teclado y Actualizar el TextView

La experiencia de búsqueda del usuario no es intuitiva. Cuando el botón es pulsado, el teclado permanece visible y no hay forma de saber si la consulta está en curso. Una solución es ocultar programáticamente el teclado y actualizar uno de los TextViews de resultado para que diga "Loading…" mientras se realiza la consulta. Para utilizar esta solución, puede hacer lo siguiente:

1. Agregue el siguiente código al método `searchBooks()` para ocultar el teclado cuando el botón sea presionado:

```
InputMethodManager inputManager = (InputMethodManager)
                    getSystemService(Context.INPUT_METHOD_SERVICE);
inputManager.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(),
                    InputMethodManager.HIDE_NOT_ALWAYS);
```

2. Añada una línea de código debajo de la llamada para ejecutar la tarea FetchBook para que cambie el TextView del título a "Loading..." y limpie el TextView autor.

3. Extraiga sus recursos de String.

3.2 Gestionar el estado de la red y el campo de búsqueda vacío

Cada vez que su aplicación utiliza la red, tiene que manejar la posibilidad de que una conexión de red no esté disponible. Antes de intentar conectarse a la red usando el AsyncTask o el AsyncTaskLoader, la aplicación debe comprobar el estado de la conexión de red.

1. Modifique el método `searchBooks()` para comprobar tanto la conexión de red como si hay algún texto en el campo de búsqueda antes de ejecutar la tarea FetchBook.

2. Actualice la UI en caso de que no haya conexión a Internet o no haya texto en el campo de búsqueda. Muestr la causa del error en el TextView.

Código de la solución:

```
public void searchBooks(View, view) {

   String queryString = mBookInput.getText().toString();

   InputMethodManager inputManager = (InputMethodManager)
           getSystemService(Context.INPUT_METHOD_SERVICE);
   inputManager.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(),
           InputMethodManager.HIDE_NOT_ALWAYS);

   ConnectivityManager connMgr = (ConnectivityManager)
           getSystemService(Context.CONNECTIVITY_SERVICE);
   NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
```

```
   if (networkInfo != null && networkInfo.isConnected() &&
queryString.length()!=0) {
       new FetchBook(mTitleText, mAuthorText).execute(queryString);
       mAuthorText.setText("");
       mTitleText.setText(R.string.loading);
   } else {
       if (queryString.length() == 0) {
           mAuthorText.setText("");
           mTitleText.setText("Please enter a search term");
       } else {
           mAuthorText.setText("");
           mTitleText.setText("Please check your network connection and try
again.");
       }
   }
}
```

Tarea 4. Migrar al AsyncTaskLoader

Cuando se utiliza AsyncTask, no puede actualizar la interfaz de usuario si se produce un cambio de configuración mientras se ejecuta la tarea en segundo plano. Para hacer frente a esta situación, el SDK de Android proporciona un conjunto de clases denominadas loaders (cargadores), diseñadas específicamente para cargar datos en la interfaz de usuario de forma asíncrona. Si utiliza un loader, no tiene que preocuparse de que el loader pierda la capacidad de actualizar la interfaz de usuario en la actividad que lo creó inicialmente. El framework Loader hace el trabajo por usted reasociando el cargador con la Actividad apropiada cuando el dispositivo cambia su configuración. Esto significa que si gira el dispositivo mientras la tarea aún está en ejecución, los resultados se mostrarán correctamente en la Actividad una vez que se devuelvan los datos.

En esta tarea se usará un loader específico llamado AsyncTaskLoader. Un AsyncTaskLoader es una subclase abstracta de Loader y utiliza un AsyncTask para cargar datos eficientemente en segundo plano.

Nota: Cuando se utiliza un AsyncTask, se implementa el método `onPostExecute()` *en el* AsyncTask para mostrar los resultados en la pantalla. Cuando se utiliza un AsyncTaskLoader, se definen métodos callback *en la Actividad* para mostrar los resultados.

Los cargadores proporcionan una gran cantidad de funcionalidad adicional más allá de simplemente ejecutar tareas y volver a conectarse a la actividad. Por ejemplo, puede conectar un cargador a una fuente de datos y hacer que actualice automáticamente los elementos de la UI cuando cambien los datos subyacentes. Los cargadores también pueden programarse para reanudar la carga si se interrumpen.

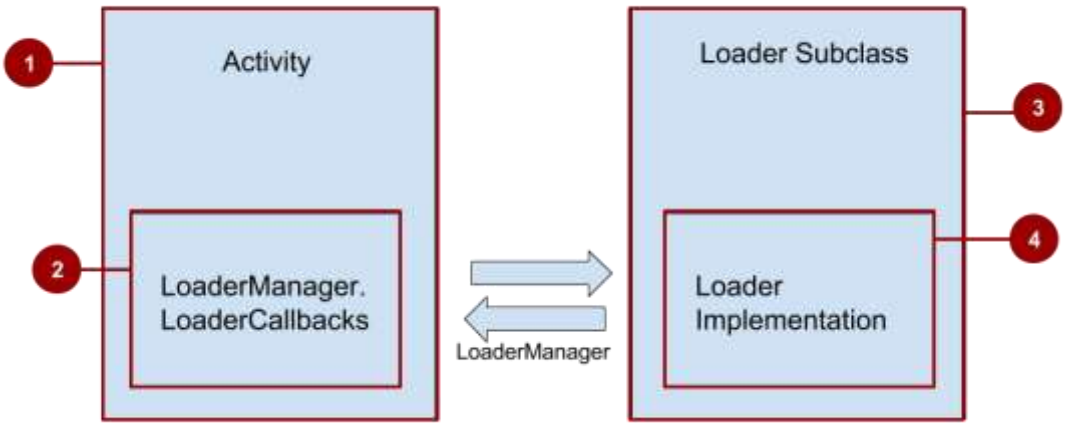
Entonces, ¿por qué usar un AsyncTask si un AsyncTaskLoader es mucho más útil? La respuesta es que depende de la situación. Si es probable que la tarea en segundo plano termine antes de que ocurran cambios de configuración, y no es crucial que actualice la interfaz de usuario, un AsyncTask puede ser suficiente. El framework Loader utiliza un AsyncTask entre bastidores para hacer funcionar su magia.

Una buena regla empírica es usar un AsyncTaskLoader en lugar de un AsyncTask si el usuario puede rotar la pantalla mientras el trabajo se está ejecutando, o cuando es crítico actualizar la interfaz de usuario cuando el trabajo termine.

En este ejercicio aprenderá a usar un AsyncTaskLoader en lugar de un AsyncTask para ejecutar su consulta de API de Libros. Aprenderá más sobre los usos de otros cargadores en una lección posterior.

La implementación de un cargador requiere los siguientes componentes:

- Una clase que extienda la clase Loader (en este caso, AsyncTaskLoader).
- Un Activity que implemente la clase LoaderManager.LoaderCallbacks.
- Una instancia del LoaderManager.



1. La Activity.
2. El LoaderManager.LoaderCallbacks.
3. La subclase Loader.
4. La implementación del Loader.

El LoaderManager mueve automáticamente el cargador a través de su ciclo de vida dependiendo del estado de los datos y de la Activity. Por ejemplo, el LoaderManager llama a `onStartLoading()` cuando el cargador está inicializado y destruye el cargador cuando la Activity es destruida.

El LoaderManager.LoaderCallbacks es un conjunto de métodos en la Activity que son llamados por el LoaderManager cuando el cargador está siendo creado, cuando los datos han terminado de cargarse, y cuando el cargador es reseteado. El LoaderCallbacks puede tomar los resultados de la tarea y pasarlos de nuevo a la UI de la Activity.

La subclase Loader contiene los detalles de la carga de los datos, normalmente sobreescribiendo a `onStartLoading()`. También puede contener características adicionales como la observación de la fuente de datos para cambios y el almacenamiento de datos en caché localmente.

La subclase Loader de esta aplicación implementa los métodos callback del ciclo de vida del Loader tales como `onStartLoading()`, `onStopLoading()` y `onReset()`. La subclase Loader también contiene el método `forceLoad()` que inicia la carga de los datos. Este método no se llama automáticamente cuando se inicia el cargador porque normalmente se requiere alguna configuración antes de realizar una carga. La implementación más simple llamaría al `forceLoad()` en `onStartLoading()` lo que resulta en una carga cada vez que el LoaderManager arranca el cargador.

4.1 Crear un AsyncTaskLoader

1. Copie el proyecto **WhoWroteIt**, para preservar los resultados de la práctica anterior. Cambie el nombre del proyecto copiado a **WhoWroteItLoader**.

2. Cree una nueva clase en el directorio Java llamada BookLoader.

3. Haga que la clase BookLoader extienda a AsyncTaskLoader con el tipo parametrizado <String>.

4. Asegúrese de importar el cargador desde la v4 Support Library.

5. Implemente el método requerido (`loadInBackground()`). Note la similitud entre este método y el método inicial `doInBackground()` de AsyncTask.

6. Cree el constructor de la nueva clase. En Android Studio, es probable que la declaración de clase siga estando subrayada en rojo porque el constructor no coincide con la implementación de la superclase. Con el cursor en la línea de declaración de clase, pulse **Alt + Enter** (**Opción + Enter** en un Mac) y seleccione **Create constructor matching super**. Esto creará un constructor con el contexto como parámetro.

Definir onStartLoading()

1. Pulse **Ctrl + O** para abrir el menú para sobreescribir métodos y seleccione en **onStartLoading**. Este método es llamado por el sistema al iniciar el cargador.

2. El cargador no comenzará a cargar los datos hasta que llame al método `forceLoad()`. Dentro de la plantilla del método `onStartLoading()`, llame al `forceLoad()` para iniciar el método `loadInBackground()` una vez creado el Loader.

Definir loadInBackground()

1. Cree una variable member llamada `mQueryString` que contendrá la consulta de tipo `String`, y modifique el constructor para que tome un `String` como argumento y se lo asigne a la variable `mQueryString`.

2. En el método `loadInBackground()`, llame al método `getBookInfo()` pasandole un `mQueryString`, y devuelva el resultado para descargar la información de la Books API:

```
@Override
public String loadInBackground() {
   return NetworkUtils.getBookInfo(mQueryString);
}
```

4.2 Modifcar el MainActivity

Ahora debe implementar las Loader Callbacks en la MainActivity para manejar los resultados del método `loadInBackground()` del AsyncTaskLoader.

1. Agregue la implementación de `LoaderManager.LoaderCallbacks` a la declaración de la clase MainActivity, parametrizada con el tipo `String`:

```
public class MainActivity extends AppCompatActivity
                implements LoaderManager.LoaderCallbacks<String>{
```

2. Implemente todos los métodos requeridos: `onCreateLoader()`, `onLoadFinished()`, `onLoaderReset()`. Coloque el cursor en la línea de la firma de la clase y pulse **Alt + Enter** (**Option + Enter** en una Mac). Asegúrese que todos los métodos estén seleccionados.

Nota: Si las importaciones para `Loader` y `LoaderManager` en la MainActivity no coinciden con la importación para `AsyncTaskLoader` de la clase `BookLoader`, tendrá algunos errores de tipo en los callback. Asegúrese de que todas las importaciones procedan de la Android Support Library.

Los loaders utilizan la clase Bundle para pasar información desde la actividad que llamada a los LoaderCallbacks. Se puede agregar datos primitivos a un bundle con el método `putType()` apropiado.

Para iniciar un cargador (loader), tiene dos opciones:

- `initLoader()`: Este método crea un nuevo cargador si no existe ya, y pasa en los argumentos un Bundle. Si ya existe un cargador, la Activity que hace la llamada se vuelve a asociar con él sin actualizar el Bundle.
- `restartLoader()`: Este método es el mismo que initLoader() excepto que si encuentra un cargador existente, lo destruye y lo recrea con el nuevo Bundle.

Ambos métodos están definidos en el LoaderManager, el cual gestiona todas las instancias del Loader utilizadas en una Activity (o Fragment). Cada Activity tiene exactamente una instancia del LoaderManager que es responsable del ciclo de vida de los Loaders que maneja.

Actualmente, el FetchBook del AsyncTask se activa cuando el usuario pulsa el botón. Cada vez que pulse el botón, querrá poner en marcha su cargador con un nuevo Bundle. Para ello, debe editar el método onClick para el botón.

1. En el método `searchBooks()`, que es el método onClick para el botón, reemplace la llamada para ejecutar la tarea FetchBook con una llamada al `restartLoader()`, pasando el string de consulta que obtuvo del EditText en el Bundle:

```
Bundle queryBundle = new Bundle();
queryBundle.putString("queryString", queryString);
getSupportLoaderManager().restartLoader(0, queryBundle,this);
```

El método `restartLoader()` toma tres argumentos:

   - Un loader `id` (muy útil si se implementa más de un loader en la actividad).
   - Un `Bundle` (aqui es donde va la data necesaria para el loader).
   - La instancia de LoaderCallbacks que impementó en la actividad. Si desea que el loader entregue los resultados a la MainActivity, se especifica este como el tercer argumeto.

2. Examine los métodos sobreescritos en la clase LoaderCallbacks. Estos métodos son:

   - `onCreateLoader()`: Llamado cuando se instancia el Loader.
   - `onLoadFinished()`: Llamado cuando finalizan las tareas del loader. Aquí es donde se agrega el código que actualiza la UI con los resultados.
   - `onLoaderReset()`: Libera cualquier recurso remanente.

Sólo definirá los dos primeros métodos, ya que el modelo de datos actual es un simple string que no necesita cuidado extra cuando se reinicia el cargador.

Implementar onCreateLoader()

1. En `onCreateLoader()`, se retorna una instancia de la clase BookLoader, pasando el `queryString` obtenido desde los argumentos Bundle:

```
return new BookLoader(this, args.getString("queryString"));
```

Implementar onLoadFinished()

1. Actualize `onLoadFinished()` para procesar el resultado, que es la respuesta en String JSON del BooksAPI.

   a. Copie el código del `onPostExecute()` de la clase FetchBook en `onLoadFinished()` en la MainActivity, excluyendo la llamada al `super.onPostExecute()`.

   b. Reemplace el argumento del constructor de JSONObject con el pasado en los datos `String`.

2. Ejecute su aplicación.

   Usted debe tener la misma funcionalidad que antes, pero ahora en un Loader! Una cosa sigue sin funcionar. Cuando se gira el dispositivo, se pierden los datos de la View. Esto se debe a que cuando la Activity es creada (o recreada), la Activity no sabe que hay un cargador corriendo. Se necesita un método `initLoader()` en el `onCreate()` de la MainActivity para volver a conectarse al cargador.

3. Agregue el siguiente código al `onCreate()` para reconectarse al Loader si este ya existe:

```
if(getSupportLoaderManager().getLoader(0)!=null){
   getSupportLoaderManager().initLoader(0,null,this);
}
```

Nota: Si el cargador existe, inicialícelo. Sólo desea reasociar el loader a la Activity si ya se ha ejecutado una consulta. En el estado inicial de la aplicación, no se carga ningún dato por lo que no hay ninguno que preservar.

4. Ejecute nuevamente la aplicación y rote el dispositivo. El LoaderManager ahora preserva los datos a través de las configuraciones del dispositivo!

5. Remueva la clase FetchBook ya que no se está usando.

Resumen

- Las tareas que se conectan a la red, o requieren un tiempo extra para su procesamiento, no deben ser ejecutadas en el hilo de la UI.
    - El Runtime de Android está predeterminado, por lo general, en StrictMode con lo cual se generará una excepción si se intenta hacer una conección a la red o tener acceso a archivos en el hilo de la UI.
- El Google API Explorer es una herramienta que ayuda a explorar numerosas APIs de Google de forma interactiva.
    - La Books Search API es un conjunto de APIs RESTful para accede al Google Books de forma programática.
    - Un request API al Google Books se hace bajo la forma de un URL.
    - Como respuesta a ese requerimiento al API es retornado un string JSON.
- Utilice `getText()` para recuperar el texto de una vista EditText. Se puede convertir en un simple string usando toString().
- La clase URI tiene un método conveniente, `Uri.buildUpon()` el cual devuelve un URI.Builder que puede ser usado para construir una string URI.
- Un AsyncTask es una clase que le permite ejecutar tareas en segundo plano, asincrónicamente, en lugar de en el hilo de la UI.

    - Un AsyncTask puede ser iniciado con `execute()`.
    - Un AsyncTask no podrá actualizar la UI si la Activity que está controlando termina (como en un cambio de configuración en el dispositivo).
    - Una AsyncTask debe ser extendida para ser usada. La subclase debe sobreescribir al método doInBackground(Params) y la mayoría de las veces también debe sobreescribir un segundo método onPostExecute(Result).

- Siempre que un AsyncTask es ejecutado, pasa por los siguientes 4 pasos:

1. onPreExecute(). Invocado en el hilo de la UI antes que la tarea sea ejecutada. Este paso es normalmente usado para configurar la tarea.

2. doInBackground(Params). Invocado en el hilo de segundo plano inmediantamente después que finaliza la ejecución de onPreExecute(). Este paso es usado para los cómputos de segundo plano que puedan tomar mucho tiempo.

3. onProgressUpdate(Progress). Invocado en el hilo de la UI después de que el doInBackground es llamado para que haga publishProgress(Progress).

4. onPostExecute(Result). Invocada en el hilo de la UI una vez finalizado el cálculo en segundo plano. El resultado del tal cálculo se pasa a este método como un parámetro.

- AsyncTaskLoader es el equivalente Loader de un AsyncTask. El proporciona un método, `loadInBackground()`, que se ejecuta en un hilo separado y cuyos resultados se envían automáticamente al hilo de la UI (al callback `onLoadFinished()` del LoaderManager).
- Debe configurar los permisos de red en el archivo de manifiesto de Android para conectarse a Internet:

```
<uses-permission android:name="android.permission.INTERNET">
```

- Utilize las clases JSON integradas en (`JSONObject` y `JSONArray`) para crear y analizar strings JSON.
- Un Loader permite la carga asíncrona de datos en una actividad.
    - Un Loader poder ser usado para restablecer la comunicación con la UI cuando una Activity es finalizada antes de que finalice la tarea (por ejemplo, mediante la rotación del dispositivo).
    - Un AsyncTaskLoader es un Loader que utiliza una clase helper AsyncTask detrás de bastidores para hacer el trabajo en segundo plano fuera del hilo principal.
    - Los Loaders son administrados por un LoaderManager; uno o más Loaders pueden ser asignados y administrados por un solo LoadManager.
    - El LoaderManager permite asociar una Activity recién creada con un Loader usando un `getSupportLoaderManager().initLoader()`.

6.3 Broadcast Receivers

Algunos eventos que pueden ocurrir en el sistema Android pueden afectar a la funcionalidad de las aplicaciones instaladas en el dispositivo. Por ejemplo, si el sistema ha terminado de arrancar, es posible que desee que su aplicación meteorológica actualice su información. El marco de trabajo de Android maneja esto mediante el envío de emisiones (broadcasts) desde el sistema que contienen Intents que están destinados a ser recibidos utilizando BroadcastReceivers. Un BroadcastReceiver es la clase base para el código que recibirá los Intents enviadas por `sendBroadcast()`. Hay dos clases principales de emisiones que se pueden recibir:

- Las emisiones normales (enviadas con Context.sendBroadcast()) son completamente asíncronas. Todos los receptores (receivers) de la emisión se ejecutan en un orden indefinido, a menudo al mismo tiempo. Esto es más eficiente, pero implica que los receptores no pueden usar el resultado o abortar las APIs incluidas aquí.

- Las emisiones ordenadas (enviadas con Context.sendOrderedBroadcast) son entregadas a un receptor a la vez. A medida que cada receptor se ejecuta por turnos, puede propagar un resultado al siguiente receptor, o puede abortar completamente la transmisión para que no se pase a otros receptores. Los receptores de orden que se ejecutan se pueden controlar con el atributo android:priority del intent-filter correspondiente; los receptores con la misma prioridad se ejecutarán en un orden arbitrario.

Incluso en el caso de emisiones normales, el sistema puede, en algunas situaciones, revertir la entrega de emisión un receptor a la vez. En particular, para los receptores que puedan requerir la creación de un proceso, sólo uno se ejecutará a la vez para evitar sobrecargar el sistema con nuevos procesos. En esta situación, sin embargo, la semántica no ordenada se mantiene: estos receptores todavía no pueden devolver resultados o abortar su emisión.

Además, se pueden crear Intents con acciones personalizadas y transmitirlos por usted mismo desde su propia aplicación utilizando el método `sendBroadcast()`. La transmisión será recibida por todas las solicitudes con un BroadcastReceiver registrado para tal acción. Para obtener más información sobre los broadcast Intents y BroadcastReceivers, visite la Intent documentation.

Es útil notar que mientras que la clase Intent se usa para enviar y recibir transmisiones, el mecanismo de transmisión del Intent es completamente separado de los Intents que se usan para iniciar Actividades.

En esta práctica, se creará una aplicación que responda a un cambio en el estado de carga del dispositivo, además de enviar y recibir una Broadcast Intent personalizada.

Resumen de la Aplicación

La aplicación PowerReceiver registrará un BroadcastReceiver que mostrará un mensaje Toast cuando el dispositivo esté conectado o desconectado de la alimentación. También enviará y recibirá un Broadcast Intent personalizado para mostrar un mensaje Toast diferente.

Tarea 1. Configurar el proyecto PowerReceiver

1.1 Crear el Proyecto

1. Cree un nuevo proyecto llamado **PowerReceiver**, acepte las opciones predeterminadas y utilice la plantilla Empty.

2. Cree un nuevo Broadcast Receiver. Seleccione el nombre del paquete en el Android Project View y vaya a **File > New > Other > Broadcast Receiver**.

3. Llame a la clase como **CustomReceiver** y asegúrese de que las opciones "Exported" (Exportado) y "Enabled" (Habilitado) estén marcadas.

   Nota: La función "Exported" permite que su aplicación responda a emisiones externas, mientras que "Enabled" permite que el sistema la instancie.

4. Navegue hasta el archivo manifest de Android. Tenga en cuenta que Android Studio genera automáticamente una etiqueta `<receiver>` con las opciones elegidas como atributos. BroadcastReceivers también puede registrarse programáticamente, pero lo más fácil es definirlos en el manifest.

1.2 Registre el Receiver para las emisiones del sistema

Para recibir cualquier emisión (broadcast), primero debe determinar cuáles broadcast intents le interesan. En la Intent documentation, bajo "Standard Broadcast Actions", puede encontrar algunas de las broadcast intents más comunes enviadas por el sistema. En esta aplicación, usted estará interesado en dos emisiones en particular: `ACTION_POWER_CONNECTED` y `ACTION_POWER_DISCONNECTED`. Los BroadcastReceivers se registran para la emisión de la misma

manera que registró las Activities para los Intents implícitos: utiliza un intent filter. Ya aprendió sobre intent implícitos en una práctica anterior.

1. En el archivo **AndroidManifest.xml**, agregue el siguiente código entre las etiquetas <receiver> para registrar el Receiver para los Intents del sistema:

```
<intent-filter>
   <action android:name="android.intent.action.ACTION_POWER_CONNECTED"/>
   <action android:name="android.intent.action.ACTION_POWER_DISCONNECTED"/>
</intent-filter>
```

1.3 Implementar onReceive() en el BroadcastReceiver

Una vez que el BroadcastReceiver intercepta una emisión para la que está registrado, el Intent es entregado al método `onReceive()` del receptor, junto con el contexto en el que se está ejecutando el receptor.

1. Navegue hasta el archivo CustomReceiver y elimine la implementación predeterminada dentro del método `onReceive()`.

2. Obtenga la acción a partir del intent y guárdela en una variable `String` llamada `intentAction`:

```
@Override
public void onReceive(Context context, Intent intent) {
   String intentAction = intent.getAction();
}
```

3. Cree una sentencia switch con el string `intentAction`, de modo que su aplicación pueda mostrar un mensaje toast diferente para cada acción específica para el que el receptor esté registrado:

```
switch (intentAction){
   case Intent.ACTION_POWER_CONNECTED:
       break;
   case Intent.ACTION_POWER_DISCONNECTED:
       break;
}
```

4. Inicialice una variable `String` llamada `toastMessage` antes de la sentencia `switch`, y haga que su valor sea `null` para que pueda ser asignado dependiendo de la broadcast action que reciba.

5. Asigne al `toastMessage` el valor "Power connected!" si la acción es `ACTION_POWER_CONNECTED`, y "Power disconnected!" si la acción es `ACTION_POWER_DISCONNECTED`. Extraiga los recursos de string.

6. Muestr un mensaje toast de corta duración después de la sentencia switch

```
Toast.makeText(context, toastMessage, Toast.LENGTH_SHORT).show();
```

7. Ejecute la aplicación. Una vez instalado, desenchufe el dispositivo. Puede que tome un momento la primera vez, pero seguro que se muestra un toast cada vez que conectas o desconectas el dispositivo.

   Nota: Si utiliza un emulador, puede cambiar el estado de la conexión de alimentación seleccionando el icono de elipses (…) del menú, seleccionando **Battery** en la barra de la izquierda e intercambiar utilizando la selección de **Charger connection**.

1.4 Restringir el Broadcast Receiver

Los Broadcast Receivers están siempre activos, y por lo tanto su aplicación ni siquiera necesita estar ejecutándose para que su método `onReceive()` sea llamado.

1. Adelante, pruebe lo siguiente: cierre la aplicación, y conecte o desconecte el dispositivo.

   El mensaje toast continua mostrandose!

   Hay mucha responsabilidad en usted, como desarrollador, para no abrumar a su usuario con notificaciones o funciones no deseadas cada vez que se produce un broadcast. En este ejemplo, si aparece un mensaje Toast cada vez que se cambia el estado de encendido, el usuario podría molestarse rápidamente. Para limitar esto, agregarás algún código para asegurarte de que el receptor sólo esté activo cuando la aplicación sea mostrada.

   La clase PackageManager es responsable de habilitar y deshabilitar un componente android en particular (tal como un service, activity o un broadcast occurs). Esto se logra usando el método `setComponentEnabledSetting()` el cual toma tres argumentos:

   - El ComponentName (un identificador del componente que se desea habilitar o desabilitar).
   - Una de las constante de la clase PackageManager que representan el estado habilitado de un componente. En esta application se usará `PackageManager.COMPONENT_ENABLED_STATE_ENABLED` y `PackageManager.COMPONENT_ENABLED_STATE_DISABLED`. Vea la referencia del PackageManager para que conozca las demás constantes.
   - Una bandera constante opcional que le indica al sistema que no mate la aplicación cuando cambie el estado del componente: `PackageManager.DONT_KILL_APP`.

2. Para que el broadcast receiver sólo esté activo cuando se muestre la aplicación, habilítelo en `onStart()` y deshabilítelo en `onStop()`.

3. Cree dos variables member: una de tipo `PackageManager` y una de tipo `ComponentName`.

4. Inicialice ambas en el método `onCreate()`.

Instancie el PackageManager with `getPackageManager()`. El constructor del ComponentName toma el context de la aplicación y el nombre de la clase del componente:

```
mReceiverComponentName = new ComponentName(this, CustomReceiver.class);
mPackageManager = getPackageManager();
```

5. Sobreescriba tanto `onStart()` como el `onStop()`:

```
@Override
protected void onStart() {
   super.onStart();
}
@Override
protected void onStop() {
   super.onStop();
}
```

6. LLame a `setComponentEnabledSetting()` en el PackageManager en `onStart()`. Pase el combre del Component, en la constante `PackageManager.COMPONENT_ENABLED_STATE_ENABLED` constant, y la bandera `DONT_KILL_APPX`:

```
mPackageManager.setComponentEnabledSetting
(mReceiverComponentName, PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
PackageManager.DONT_KILL_APP);
```

7. En `onStop()`, use el PackageManager para desabilitar el CustomReceiver, usando la constante `PackageManager.COMPONENT_ENABLED_STATE_DISABLED`:

```
mPackageManager.setComponentEnabledSetting
(mReceiverComponentName, PackageManager.COMPONENT_ENABLED_STATE_DISABLED,
PackageManager.DONT_KILL_APP);
```

Tarea 2. Enviar y recibir un Broadcast personalizado

Además de responder a los broadcast del sistema, una aplicación también puede enviar y recibir Broadcast Intents personalizadas. Un broadcast intent personalizada es exactamente la misma que la del sistema, excepto que se le debe definir su propia Intent action (un string único) y se entrega utilizando el método `sendBroadcast()`. En esta tarea, se agregará un botón a la actividad que envía un Broadcast Intent personalizado, el cual será registrado por su receptor y mostrado en un mensaje Toast.

2.1 Definir el string del Broadcast Action personalizado

Tanto el emisor como el receptor de un broadcast personalizado deben acordar un string de acción único para el Broadcast Intent. Es una práctica común crear strings de acción únicos añadiendo el Action Name al nombre del paquete.

1. Cree una constante `String` tanto en la clase MainActivity como en la CustomReceiver para ser usadas como la Broadcast Intent Action (esta es el string del custom action):

```
private static final String ACTION_CUSTOM_BROADCAST =
"com.example.android.powerreceiver.ACTION_CUSTOM_BROADCAST";
```

2.2 Agregar un botón "Send Custom Broadcast"

1. En el archivo activity_main.xml, agregue una vista Button con los siguientes atributos:

Atributo	Valor
android:id	"@+id/sendBroadcast"
android:layout_width	wrap_content
android:layout_height	wrap_content
android:text	"Send Custom Broadcast"
android:layout_margin	"8dp"
android:onClick	"sendCustomBroadcast"

2. Extraiga los recursos de string.

3. Cree la plantilla para el método `sendCustomBroadcast()`: Haga clic en el nombre resaltado en Amarillo del método onClick(). Presione **Alt + Enter (Option + Enter** para usuarios Mac**)** y seleccione '**Create 'sendCustomBroadcast(View)' in 'MainActivity'**.

2.3 Implementar sendCustomBroadcast()

Dado que este broadcast está destinado a ser utilizado únicamente por esta aplicación, se debe utilizar LocalBroadcastManager para gestionar los broadcast de la aplicación. LocalBroadcastManager es una clase que le permite registrarse y enviar transmisiones de Intents a objetos locales dentro de la propia aplicación. Al mantener las transmisiones locales, los datos de la aplicación no se compartirán con otras aplicaciones Android, manteniendo la información más segura y mejorando la eficiencia del sistema.

1. En el método `sendCustomBroadcast()` de MainActivity, cree un nuevo Intent, con el action string personalizado como argumento.

```
Intent customBroadcastIntent = new Intent(ACTION_CUSTOM_BROADCAST);
```

2. Envíe el broadcast usando la clase LocalBroadcastManager:

```
LocalBroadcastManager.getInstance(this).sendBroadcast(customBroadcastIntent);
```

2.4 Registrar el Broadcast Personalizado

Para los bradcasts del sistema, se registró el receptor en el archivo AndroidManifest.xml. También es posible registrar programáticamente al receptor para acciones específicas. Para los broadcasts enviados a través del LocalBroadcastManager, no se permiten registros estáticos en el manifiesto.

Si registra el broadcast receiver de forma programada, también debe anular el registro del receptor cuando ya no sea necesario. En esta aplicación, el receptor sólo tendrá que responder al broadcast personalizado cuando se esté ejecutando, por lo que podemos registrar la acción en `onCreate()` y desregistrarla en `onDestroy()`.

1. Cree una variable member en MainActivity para el Receiver e iniciela:

```
private CustomReceiver mReceiver = new CustomReceiver();
```

2. En `onCreate()`, obtenga una instancia del LocalBroadcastManager y register el receptor con el intent action personalizado:

```
LocalBroadcastManager.getInstance(this)
.registerReceiver(mReceiver, new IntentFilter(ACTION_CUSTOM_BROADCAST));
```

3. Sobreescriba el método `onDestroy()` y desregistre el receptor del `LocalBroadcastManager`:

```
@Override
protected void onDestroy() {
   LocalBroadcastManager.getInstance(this).unregisterReceiver(mReceiver);
   super.onDestroy();
}
```

2.5 Responder al Broadcast Personalizado

1. En `onReceive()` en la clase CustomReceiver, agregue una sentencia case para el Intent Action personalizado.

2. Modifique el mensaje toast message para que sea "Custom Broadcast Received", extraigalo en strings.xml y llamelo `custom_broadcast_toast` (presione **Alt + Enter** o **Option + Enter** en una Mac y seleccione **extract string resource)**:

```
case ACTION_CUSTOM_BROADCAST:
   toastMessage = context.getString(R.string.custom_broadcast_toast);
   break;
```

Nota: Los Broadcast Receivers que están registrados programáticamente no se ven afectados por la activación o desactivación realizada por la clase PackageManager, la cual está destinada a los componentes listados en el archivo Android Manifest. La activación o desactivación de dichos receptores se realiza registrándolos o desregistrándolos,

respectivamente. En este caso, al apagar la tecla "Receiver Enabled" (Receptor habilitado) se detendrán los mensajes de alimentación conectada o desconectada, pero no los mensajes Toast del Broadcast Intent personalizado.

¡Eso es todo! La aplicación ahora entrega broadcast intents personalizados y puede recibir tanto transmisiones del sistema como transmisiones personalizadas.

Resumen

- Los Broadcast Receivers son uno de los componentes fundamentales de una aplicación android.
- Los Broadcast Receivers pueden recibir contenidos emitidos tanto por el sistema como por la aplicación.
- El mecanismo de transmisión de un Intent está completamente separado de los Intents que se utilizan para iniciar las Activities.
- Se necesita extender la clase BroadcastReceiver e implementar onReceive() para procesar la Intent entrante asociada con la emisión.
- Se puede registrar un broadcast receiver en el archivo de manifiesto de Android o de forma programada.
- Se utiliza el LocalBroadcastManager para registrar y enviar Broadcasts que son privadas para una aplicación.
- LocalBroadcastManager es más eficiente y seguro que los broadcast del sistema.
- En el caso de broadcasts enviados a través del LocalBroadcastManager, sólo se puede registrar su interés por acciones específicas de forma programada.
- Una práctica común para crear nombres únicos de intent actions para los broadcasts es anteponer el Action Name al nombre del paquete.

6.4 Notifications

Hasta ahora, las aplicaciones que hemos creado usaban elementos de la UI que sólo son visibles cuando la aplicación se está ejecutando. La única excepción a esto es el BroadcastReceiver que implementamos que mostraba un mensaje Toast cuando el dispositivo estaba conectado o desconectado de la alimentación. Hay muchas veces que se quiere mostrar la información del usuario incluso cuando la aplicación no se está ejecutando. Por ejemplo, se puede hacer saber que hay un nuevo contenido disponible, o actualizar la puntuación del equipo favorito. El marco de trabajo de Android proporciona un mecanismo para que la aplicación notifique a los usuarios incluso cuando la aplicación no esté en primer plano: el Notification framework.

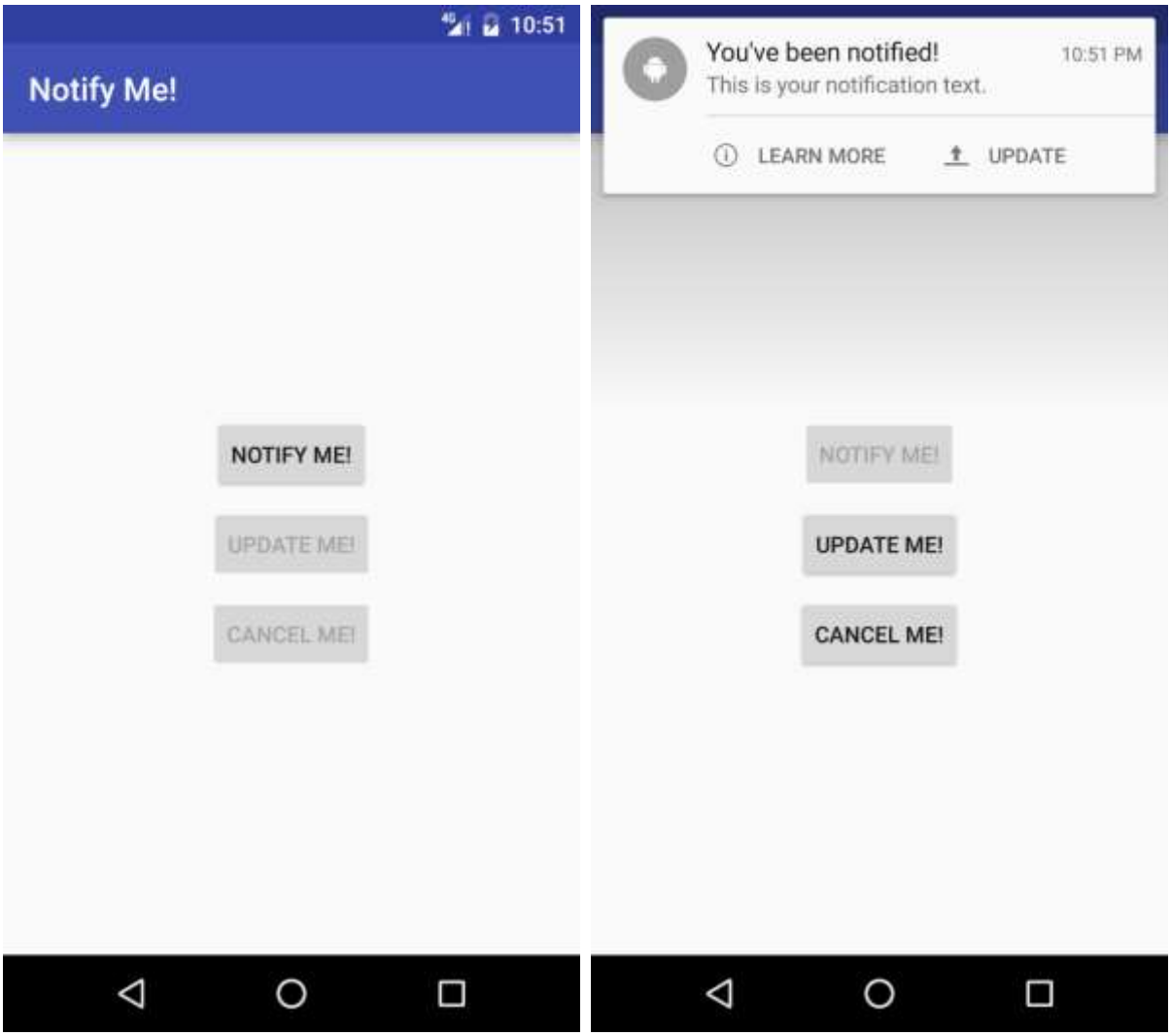
Una Notification es un mensaje que se puede mostrar al usuario fuera de la UI normal de la aplicación. Cuando Android emite una notificación, aparecerá primero como un icono en el **área de notificación del dispositivo**. Para ver los detalles específicos de la notificación, el usuario

abre el **notification drawer**. Tanto el área de notificación como el notification drawer son áreas controladas por el sistema que el usuario puede ver en cualquier momento.

En esta práctica creará una aplicación que dispara una notificación cuando se pulsa un botón y proporciona la capacidad de actualizar la notificación o cancelarla.

Resumen de la aplicación

Notify Me! es una aplicación que puede mostrar, actualizar y cancelar una notificación. También experimenta con estilos de notificación, acciones y prioridades.



Tarea 1. Crear una notificación básica

1.1 Crear el proyecto

1. Cree un Nuevo Proyecto llamado “Notify Me!”, acepte las opciones por defecto y use una plantilla vacía.
2. En el archivo activity_main.xml, cambie el element raiz de la vista por un vertical LinearLayout con su atributo gravity configurado como "center".
3. Agregur un botón con los siguientes atributos para reemplazar el TextView predeterminado:

Atributo	Valor
android:id	"@+id/notify"
android:layout_width	"wrap_content"

android:layout_height	"wrap_content"
android:text	"Notify Me!"
android:layout_margin	"4dp"

4. Cree un método vacío `sendNotification()`. El método no debe tomar ningún argumento y devolver void:

```
public void sendNotification() {}
```

5. Cree una variable miembro para el Botón Notify.

6. Inicialize el botón en `onCreate()` y cree un onClickListener para ello:

```
mNotifyButton = (Button) findViewById(R.id.notify);
mNotifyButton.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View view) {
   }
});
```

7. Llame al `sendNotification()` desde el método onClick.

1.2 Construir la primera notificación

Las notificaciones se crean utilizando la clase NotificationCompat.Builder , la cual permite definir el contenido y el comportamiento de la Notification. Una notificación debe contener los siguientes elementos:

- Un título, establecido por `setContentTitle()`.
- Texto detallado, establecido por `setContentText()`.
- Un icono, definido por `setSmallIcon()`.

Una Notificación de Android es desplegada por el NotificationManager . Si se necesita actualizar o cancelar la notificación en el futuro, se debe asociar un ID de notificación a tal Notificación.

Creación del icono de notificación

1. Vaya a **File > New > Image Asset**.
2. En el menú desplegable **Icon Type**, seleccione **Notification Icons** .
3. Haga clic en el icono junto al elemento **Clip Art** para seleccionar un icono material que utilizará como icono para la notificación. En este ejemplo, puede utilizar el icono de Android.
4. Cambie el nombre del recurso `ic_android` y haga clic en **Next and Finish**. Esto creará un número de archivos drawables con diferentes resoluciones para diferentes API´s.

5. Cree una variable miembro en MainActivity para almacenar el NotificationManager:

```
private NotificationManager mNotifyManager;
```

6. Cree una variable constante para el ID de la notificación. Dado que sólo habrá una notificación activa a la vez, podemos utilizar el mismo ID para todas las notificaciones:

```
private static final int NOTIFICATION_ID = 0;
```

7. Instancie el NotificationManager en el onCreate usando `getSystemService()`:

```
mNotifyManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
```

8. Cree e instancie el Notification Builder en el método `sendNotification()`:

```
NotificationCompat.Builder notifyBuilder = new NotificationCompat.Builder(this)
```

Nota: Asegúrese que la clase NotificationCompat es importada desde la libría de soporte v4.

9. Establezca el título de la notificación en "You've been notified!".

10. Establezca el texto de notificación en "This is your notification text."

11. Establezca el icono de Notificación con el icono de Android que ha añadido.

```
NotificationCompat.Builder notifyBuilder = new NotificationCompat.Builder(this)
   .setContentTitle("You've been notified!")
   .setContentText("This is your notification text.")
   .setSmallIcon(R.drawable.ic_android);
```

12. Llame al método `Call notify()` en el NotificationManager al final del método `sendNotification()`, pasando el Id de notificación y la notificación:

```
Notification myNotification = notifyBuilder.build();
mNotifyManager.notify(NOTIFICATION_ID, myNotification);
```

13. Ejecute su aplicación. El botón "Notify me!" ahora emite una notificación (busque el icono en la barra de estado), pero le faltan algunas características esenciales: no hay sonido o vibración de notificación, y al hacer clic en la notificación no hace nada. Ahora añadiremos alguna funcionalidad adicional a esa notificación.

1.3 Agregar un content intent

Para mejorar la notificación, se agregarán algunas características adicionales, disponibles a través de la clase NotificationCompat.Builder:

- Un content intent, el cual es lanzado cuando se toca la notificación y es configurado por `setContentIntent()`.
- Una prioridad, la cual determina como el sistema mostrará la notificación con respect a otras notificaciones, esto es configurado por `setPriority()`.
- Las opciones predeterminadas, como sonidos, vibración y luces LED (si están disponibles), y se configuran en `setDefaults()`.

Al tocar una notificación se inicia un Intent. Los Content Intents para notificaciones son muy similares a los Intens que ya ha estado usando a lo largo de este curso. Pueden ser intents explícitos para lanzar una actividad, intents implícitos para realizar una acción, o broadcast intents para notificar al sistema de una ocurrencia o evento personalizado. La principal diferencia con un Intent en una notificación es que debe estar envuelta en un PendingIntent, el cual permite que la notificación realice la acción incluso si la aplicación no se está ejecutando. Un PendingIntent es entregado a un componente externo (por ejemplo, el NotificationManager) lo cual permite que la aplicación externa utilize los permisos de su aplicación para ejecutar un código predefinido. En efecto, autoriza a la notificación para que envíe el Intent en nombre de la aplicación.

Para este ejemplo, el content intent de la notificación (es decir, la intención que se lanza cuando se pulsa la notificación) iniciará la MainActivity de la aplicación (si ya se está en la aplicación, esto no tendrá ningún efecto).

1. Cree un intent explicito en el método `sendNotification()` para lanzar la clase MainActivity:

```
Intent notificationIntent = new Intent(this, MainActivity.class);
```

2. Obtenga un `PendingIntent` usando `getActivity()`, pasando la constante de ID de notificación al requestCode y usando la bandera FLAG_UPDATE_CURRENT:

```
PendingIntent notificationPendingIntent = PendingIntent.getActivity(this,
    NOTIFICATION_ID, notificationIntent, PendingIntent.FLAG_UPDATE_CURRENT);
```

3. Agreguege el PendingIntent a la Notification usando `setContentIntent()` en el NotificationCompat.Builder:

```
.setContentIntent(notificationPendingIntent)
```

4. Ejecute la aplicación. Haga clic en el botón “Notify Me!” para enviar la notificación. Salga de la aplicación. Ahora vea la notificación y haga clic en ella. Note que la aplicación se abrirá de nuevo en el MainActivity.

1.4 Agregar la prioridad y los valores predeterminados a la notificación

Cuando su usuario hace clic en el botón " Notify Me!", la notificación se emite pero lo único que el usuario ve es el icono en la barra de notificación. Para captar la atención del usuario, los valores predeterminados de la notificación y la prioridad deben estar configurados correctamente.

La prioridad es un valor entero desde PRIORITY_MIN (-2) a PRIORITY_MAX (2) que representa la importancia de la notificación para el usuario. Las notificaciones con mayor prioridad se clasificarán por encima de las de menor prioridad en el notification drawer. Las notificaciones de prioridad HIGH o MAX se entregarán como notificaciones "Heads - Up", las cuales se despliegan en la parte superior de la pantalla activa del usuario.

1. Añada la siguiente línea al Notification Builder para establecer la prioridad de la notificación en HIGH:

```
.setPriority(NotificationCompat.PRIORITY_HIGH)
```

2. Las opciones predeterminadas en el Builder se utilizan para configurar los sonidos, la vibración y el patrón de colores del LED para la notificación (si el dispositivo del usuario tiene un indicador LED). En este ejemplo, se utilizarán las opciones predeterminadas añadiendo la siguiente línea al Builder:

```
.setDefaults(NotificationCompat.DEFAULT_ALL)
```

3. Se necesitará salir de la aplicación y reiniciarla para ver los cambios.

Nota: La notificación de alta prioridad no se desplegará delante de la pantalla activa a menos que se hayan establecido tanto la prioridad como los valores predeterminados. La prioridad por sí sola no es suficiente.

Tarea 2. Actualizar y cancelar la notificación

Después de emitir una notificación, es útil poder actualizar o cancelar el aviso si la información cambia o deja de ser relevante.

En esta tarea, se indicará cómo actualizar y cancelar una notificación.

2.1 Add update and cancel buttons

1. En el archivo layout, cree dos copias del botón "Notify Me!
2. Cambie el atributo de texto en las copias como "Update Me! y “Cancel Me!”.
3. Cambie los id's a "update" y "cancel", respectivamente.
4. Añada una variable miembro para cada uno de los nuevos botones e inicialícelos en el onCreate().
5. Cree dos métodos en MainActivity que no tomen ningún parámetro y devuelvan void:

```
public void updateNotification() {}
public void cancelNotification() {}
```

6. Cree los métodos onClick Listeners para los nuevos botones y llame al método `updateNotification()` en el método onClick del botón "update" y `cancelNotification()` en el método onClick del botón "cancel".

2.2 Implementar los métodos de cancelar y actualizar las notificaciones

Cancelar la Notificación

Cancelar una notificación es sencillo: llame el método `cancel()` en el NotificationManager, pasando el ID de notificación:

```
mNotifyManager.cancel(NOTIFICATION_ID);
```

Actualizar la Notificación

La actualización de una notificación es más compleja. Las notificaciones Android vienen con estilos alternativos que pueden ayudar a condensar la información o representarla de forma más eficiente. Por ejemplo, la aplicación de Gmail utiliza notificaciones "InboxStyle" si hay más de un mensaje no leído, condensando la información en una sola notificación.

En este ejemplo, actualizará su notificación para utilizar la notificación BigPictureStyle, que le permite incluir una imagen en su notificación.

1. Descargue la imagen Cap6_4_starter_image.jpg desde la sección “Archivos/Ejercicios” del M7 para utilizarla en su notificación y renómbrela como mascot_1.
2. Colóquela en la carpeta res/drawable.
3. En el método `updateNotification()`, convierta el drawable en un mapa de bits:

```
Bitmap androidImage = BitmapFactory
    .decodeResource(getResources(),R.drawable.mascot_1);
```

4. Copie el Intent y el PendingIntent que creó en `sendNotification()` en el `updateNotification()`, ya que se utilizará el mismo PendingIntent como Content Intent.
5. Copie el código del NotificationCompat.Builder del `sendNotification()` en el `updateNotification()`, para tener las mismas opciones básicas de notificación en la notificación actualizada.
6. Cambie el estilo de la notificación en el mismo NotificationCompat.Builder, configurando la imagen y el "Big Content Title":

```
.setStyle(new NotificationCompat.BigPictureStyle()
   .bigPicture(androidImage)
   .setBigContentTitle("Notification Updated!"));
```

Note: La BigPictureStyle es una subclase de NotificationCompat.Style la cual suministra layouts alternativos para las notificaciones. Vea la documentación para las otras subclases definidas.

7. Cambie la prioridad del Builder al valor predeterminado, de modo que no reciba otra notificación de alerta cuando esta sea actualizada (las notificaciones *heads up* sólo se pueden mostrar en el estilo predeterminado).

```
.setPriority(NotificationCompat.PRIORITY_DEFAULT)
```

8. Llame al método `notify()` en el NotificationManager, pasando el mismo ID de notificación como se hizo anteriormente.

```
mNotifyManager.notify(NOTIFICATION_ID, notifyBuilder.build());
```

9. Ejecute la aplicación. Después de hacer clic en actualizar, vuelva a comprobar la notificación. Ahora tiene una imagen y el título actualizado!. Puede volver al estilo de notificación normal presionando en el estilo extendido.

2.3 Cambiar el estado de los botones

En esta aplicación, el usuario puede confundirse porque el estado de la notificación no es rastreado dentro de la actividad. Por ejemplo, el usuario puede tocar "Cancel Me!" cuando no se ha mostrado ninguna notificación. Se puede solucionar habilitando y deshabilitando los distintos botones en función del estado de la notificación. Cuando la aplicación se ejecuta por primera vez, el botón "Notify Me!" debería ser el único habilitado, ya que aún no hay notificación que actualizar o cancelar. Después de enviar una notificación, los botones de cancelación y actualización deben estar habilitados, y el botón de notificación debe estar deshabilitado, ya que la notificación ya ha sido entregada. Después de que se actualice la notificación, los botones de actualización y notificación deben desactivarse, dejando sólo activado el botón de cancelación. Por último, si se cancela la notificación, los botones deben volver a la condición inicial, siendo el botón de notificación el único habilitado.

Aquí está el código de cambio de estado habilitado para cada método:

onCreate():

```
mNotifyButton.setEnabled(true);
mUpdateButton.setEnabled(false);
mCancelButton.setEnabled(false);
```

sendNotification():

```
mNotifyButton.setEnabled(false);
mUpdateButton.setEnabled(true);
mCancelButton.setEnabled(true);
```

updateNotification():

```
mNotifyButton.setEnabled(false);
mUpdateButton.setEnabled(false);
mCancelButton.setEnabled(true);
```

cancelNotification():

```
mNotifyButton.setEnabled(true);
mUpdateButton.setEnabled(false);
mCancelButton.setEnabled(false);
```

Tarea 3. Agregar acciones de notificación

A veces, una notificación requiere una interacción inmediata: apagar una alarma, responder un mensaje de texto, etc. Cuando se producen estos tipos de notificaciones, el usuario puede tocar su notificación para responder al evento. Android entonces carga la Actividad apropiada en la aplicación para que el usuario responda. Para evitar la apertura de la aplicación, el framework de notificación permite incrustar una acción de notificación directamente en la propia notificación. Esto permite al usuario actuar sobre la notificación sin abrir la aplicación.

Los componentes necesarios para una acción son:

- Un icono, a ser colocado en la notificación.
- Un string de etiqueta, colocado al lado del icono.
- Un PendingIntent, ha ser enviado cuando se haga clic en la acción de la notificación.

Para este ejemplo, se agregarán dos acciones a la notificación. Primero se añadirá una acción "Learn More" con un intent implícito el cual lanza una página web, luego una acción "Update" con un broadcast intent el cual actualiza la notificación sin lanzar la aplicación.

3.1 Implementar la acción "Learn More"

Como primer ejemplo de acciones de notificación, se implementará una que lance un intent implícito para abrir un sitio web.

1. Cree una variable String miembro que contenga la URL de la guía Material Design para notificaciones: https://developer.android.com/design/patterns/notifications.html.
2. Cree un intent implícito que abra la URL guardada en el método `sendNotification()` antes de crear la notificación.
3. Cree un PendingIntent a partir del intent implícito, utilizando la bandera FLAG_ONE_SHOT para que el PendingIntent no pueda ser reutilizado:

```
Intent learnMoreIntent = new Intent(Intent.ACTION_VIEW, Uri
    .parse(NOTIFICATION_GUIDE_URL));
```

```
PendingIntent learnMorePendingIntent = PendingIntent.getActivity
    (this,NOTIFICATION_ID,learnMoreIntent,PendingIntent.FLAG_ONE_SHOT);
```

4. Agregur este icono usasndo el Image Asset Studio, y llámelo ic_learn_more: 

5. Agregue la siguiente línea de código al buider tanto en `sendNotification()` como en `updateNotification()` para agregar la acción tanto a la notificación original como a la actualizada:

```
.addAction(R.drawable.ic_learn_more,"Learn More", learnMorePendingIntent);
```

6. Ejecute la aplicación. La aplicación mostrará ahora un icono al que se le puede hacer clic que lo llevará a la web!

3.2 Implementar la acción "Update"

Ha visto que una acción de notificación utiliza un PendingIntent para responder a la interacción con el usuario. En el último paso, agregó una acción que utiliza un PendingIntent creado mediante el método `getActivity()`. También puede crear un PendingIntent que entregue un broadcast intent llamando a `getBroadcast()` en la clase PendingIntent. Los broadcast intents son muy útiles en las notificaciones, ya que un broadcast receiver puede registrar su interés en el intent y responder en consecuencia, sin necesidad de iniciar una actividad específica.

Ahora se implementará un Broadcast Receiver que llamará al método `updateNotification()` cuando se pulse la acción "Update" en la notificación. Este es un patrón común para añadir funcionalidad a una notificación que ya existe en la aplicación, por lo que el usuario no necesita lanzar ninguna aplicación para realizar la acción.

1. Haga una subclase de BroadcastReceiver como una clase interna en la MainActivity y sobrescriba el método `onReceive()`. No olvide de incluir un constructor vacío:

```
public class NotificationReceiver extends BroadcastReceiver {

   public NotificationReceiver() {
   }

   @Override
   public void onReceive(Context context, Intent intent) {
   }
}
```

2. En el método `onReceive()`, llame a `updateNotification()`.

3. Cree una variable miembro constante en MainActivity para representar la acción de notificación de actualización para el BroadcastIntent. Esté pendiente de que comienza con el nombre de su paquete para asegurarse de que es único:

```
private static final String ACTION_UPDATE_NOTIFICATION =
    "com.example.android.notifyme.ACTION_UPDATE_NOTIFICATION";
```

4. Cree una variable miembro para el receiver e inicialicela usando el constructor por defecto.

```
private NotificationReceiver mReceiver = new NotificationReceiver();
```

5. En el método `onCreate()`, registre el Broadcast Receiver para recibir el intent ACTION_UPDATE_NOTIFICATION:

```
registerReceiver(mReceiver,new IntentFilter(ACTION_UPDATE_NOTIFICATION));
```

6. Override the onDestroy() method of your Activity to unregister your receiver:

```
@Override
protected void onDestroy() {
unregisterReceiver(mReceiver);
   super.onDestroy();
}
```

Nota: En este ejemplo está registrando el Broadcast Receiver programáticamente debido a que el receptor está definido como una clase interna. Cuando los receptores se definen de esta manera, no pueden registrarse en el Manifiesto Android ya que son dinámicos y tienen la posibilidad de cambiar durante la vida de la aplicación.

Puede parecer que el broadcast enviado por la notificación sólo concierne a su aplicación y debe ser entregada con un LocalBroadcastManager. Sin embargo, el uso de PendingIntents delega la responsabilidad de entregar la notificación al Android Framework. Debido a que el tiempo de ejecución de Android está manejando la transmisión, LocalBroadcastManager no puede ser utilizado.

Crear la Acción Update

1. Cree un broadcast Intent en el método `sendNotification()` usando la acción personalizada update.

2. Obtenga un PendingIntent usando `getBroadcast()`:

```
Intent updateIntent = new Intent(ACTION_UPDATE_NOTIFICATION);
PendingIntent updatePendingIntent = PendingIntent.getBroadcast
      (this, NOTIFICATION_ID, updateIntent, PendingIntent.FLAG_ONE_SHOT);
```

3. Cree este icono usando el Image Asset Studio, llámelo ic_update. 

4. Agregue la acción al builder en el método `sendNotification()`, dándole el título "Update":

```
.addAction(R.drawable.ic_update, "Update", updatePendingIntent)
```

5. Ejecute la aplicación. Ahora puede actualizar la notificación sin abrir la aplicación!.

Resumen

- Una notificación es un mensaje que se puede mostrar al usuario fuera de la UI normal de la aplicación.
- Las Notifications proporcionan una forma de interacción con el usuario de la aplicación, aún cuando está no esté en ejecución.
- Cuando Android emite una notificación, aparecerá primero como un icono en el área de notificación del dispositivo.
- La UI y las acciones para una notificación se especifican usando el NotificationCompat.Builder.
- Para crear una notificación utilice `NotificationCompat.Builder.build()`.
- Para emitir la notificación, pase el objeto Notification al runtime de Android con `NotificationManager.notify()`.
- Para actualizar o cancelar una notificación, se debe asociar un ID de notificación a la notificación.
- Un Intent puede formar parte de una notificación (Explícito, Implícito o Broadcast).
- Los Intents de una notificación deben estar "envueltas" en un PendingIntent, que en realidad no es un Intent. Un PendingIntent es una implementación del patrón decorator.
- Los componentes necesarios de una notificación son: un icono pequeño (`setSmallIcon()`), el título (`setContentTitle()`) y un texto de detalle (`setContentText()`).
- Algunos componentes opcionales de una notificación son: intent, estilos ampliados, prioridad, etc. Para más información, véase NotificationCompat.Builder.

6.5 Gestor de Alarmas (Alarm Manager)

En prácticas anteriores, ha aprendido a hacer que la aplicación responda a la interacción del usuario pulsando un botón o una notificación. También has aprendido a hacer que la aplicación responda a los eventos del sistema usando BroadcastReceivers. ¿Pero qué pasa si la aplicación necesita tomar acción en un momento específico, como es el caso de una notificación en el calendario? En ese caso, se debe utilizar AlarmManager, una clase que permite iniciar y repetir un PendingIntent a una hora e intervalo específicos.

En esta práctica, va a crear un temporizador que le recordará que debe ponerse de pie si ha estado sentado durante demasiado tiempo.

Resumen de la aplicación

Stand Up! es una aplicación que le ayuda a mantenerse saludable recordándole que se debe levantar y caminar cada quince minutos. Para este fin se utiliza una notificación para avisarle cuando han pasado quince minutos. La aplicación incluye un botón toggle que puede activar y desactivar la alarma.

Tarea 1. Configure el proyecto Stand Up! y las vistas

1.1 Crear el layout del proyecto Stand Up!

1. Cree un nuevo proyecto llamado "Stand Up!", acepte las opciones por defecto y use la plantilla empty activity.
2. Abra el archivo layout activity_main.xml.
    a. Cambie el root view para RelativeLayout.
    b. Remueva completamente el TextView "Hello World" y agregue los siguientes elementos:

TextView	Atributo	Valor
	android:layout_width	"wrap_content"
	android:layout_height	"wrap_content"

	android:layout_above	"@+id/alarmToggle"
	android:layout_centerHorizontal	"true"
	android:layout_margin	"8dp"
	android:text	"Stand Up Alarm"
	android:textAppearance	"@style/TextAppearance.AppCompat.Headline"
ToggleButton	**Atributo**	**Valor**
	android:id	"@+id/alarmToggle"
	android:layout_width	"wrap_content"
	android:layout_height	"wrap_content"
	android:layout_centerHorizontal	"true"
	android:layout_centerVertical	"true"

1.2 Configure le método setOnCheckedChangeListener()

La aplicación Stand Up! incluye un botón toggle que se utiliza para configurar y cancelar la alarma, así como para representar visiblemente el estado actual de la alarma. Para configurar la alarma cuando el botón esté encendido, utilizará el método `onCheckedChangeListener()`:

1. En el método onCreate() del MainActivity, encuentre el botón alarmToggle por su id.
2. Llame al `setOnCheckedChangeListener()` en la instancia del botón toggle, y empiece a escribir "`new OnCheckedChangeListener`". Android Studio completará automáticamente el método por usted, incluyendo la sobreescritura del método `onCheckedChanged()` que es requerido. Este método tiene dos parámetros: el CompoundButton en el que se ha hecho clic (en este caso es el botón Alarm Toggle), y un booleano que representa el estado actual del Botón Toggle (es decir, si el botón está ahora activado o desactivado).

```
alarmToggle.setOnCheckedChangeListener(
   new CompoundButton.OnCheckedChangeListener() {

   @Override
   public void onCheckedChanged(CompoundButton compoundButton,
        boolean isChecked) {
   }
});
```

3. Es útil para el usuario tener alguna retroalimentación aparte del botón toggle que se está activando y desactivando para indicar que la alarma fue efectivamente activada (todavía no se ha implementado la alarma, esto se hará en otra sección). Configure un bloque if/else usando el parámetro booleano en el método `onCheckedChanged()` que entregue un mensaje toast para decirle al usuario si la Alarma fue activada o desactivada. No olvide extraer tus recursos de cadena.

```
String toastMessage;
if(isChecked){
   //Set the toast message for the "on" case
   toastMessage = getString(R.string.alarm_on_toast);
} else {
   //Set the toast message for the "off" case
   toastMessage = getString(R.string.alarm_off_toast);
}

//Show a toast to say the alarm is turned on or off
Toast.makeText(MainActivity.this, toastMessage, Toast.LENGTH_SHORT)
       .show();
```

Tarea 2. Configurar la notificación

El siguiente paso es crear la notificación que recordará al usuario que debe ponerse de pie cada quince minutos. Por ahora, la notificación se entregará inmediatamente cuando el toggle se active.

2.1 Crear la notificación

En este paso, creará un método `deliverNotification()` que publicará el recordatorio para que el usuario se levante y camine.

1. Cree una variable miembro en MainActivity llamada mNotificationManager de tipo NotificationManager.
2. Inicialícela en `onCreate()` llamando al método `getSystemService()`:

```
mNotificationManager =
               (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
```

3. Cree un método en MainActivity llamado `deliverNotification()` que tome el Contexto como argumento y que no devuelva nada.

```
private void deliverNotification(Context context) {}
```

4. Cree una constante miembro en MainActivity llamada NOTIFICATION_ID e inicialícela en 0. La aplicación sólo tendrá una notificación a la vez, por lo que utilizará el mismo ID de notificación para todas las notificaciones.

Nota: Los ID de notificación se utilizan para distinguir las notificaciones dentro de su aplicación. El NotificationManager sólo podrá cancelar las notificaciones enviadas desde la aplicación para que pueda utilizar el mismo ID en diferentes aplicaciones.

Content Intent de Notificación

1. Cree un Intent en el método onCreate() que será usado por el content intent de la notificación:

```
Intent contentIntent = new Intent(context, MainActivity.class);
```

2. Cree un PendingIntent a partir del content Intent justo debajo de la definición de `contentIntent` usando el método `getActivity()`, pasando el ID de notificación y usando el flag FLAG_UPDATE_CURRENT:

3. Create a PendingIntent from content Intent right below the definition of contentIntent using the getActivity() method, passing in the notification ID and using the FLAG_UPDATE_CURRENT flag:

```
PendingIntent contentPendingIntent = PendingIntent.getActivity
   (context, NOTIFICATION_ID, contentIntent, PendingIntent.FLAG_UPDATE_CURRENT);
```

Nota: Los flags de PendingIntent le indican al sistema cómo manejar la situación cuando se crean varias instancias de la misma PendingIntent (lo que significa que contienen la misma intent). La bandera FLAG_UPDATE_CURRENT indica al sistema que utilice el Intent antiguo pero que sustituya los extras data. Puesto que no tienes ningún extra en este Intent, se reutiliza el mismo PendingIntent una y otra vez.

Título y texto de la Notificación

1. Cree un recurso de string en el archivo strings.xml y llámelo notification_title. Hágalo igual a "Alerta para levantarse".

2. Cree un recurso de string en el archivo strings.xml y llámelo notification_text. Hágalo igual a "Ya debe levantarse y caminar alrededor!".

Icono de la Notificación

1. Añada un activo de imagen para utilizarlo como icono de notificación (utilice Image Asset Studio). Seleccione cualquier icono que considere apropiado para esta alarma:

Construya la notificación

1. Use el NotificationCompat.Builder para construir una notificación en el método `deliverNotification()` usando el título, texto, ícono y content intent de la anterior notificación.

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(context)
   .setSmallIcon(R.drawable.ic_stand_up)
   .setContentTitle(context.getString(R.string.notification_title))
   .setContentText(context.getString(R.string.notification_text))
   .setContentIntent(contentPendingIntent)
```

2. Coloque la prioridad de la Notification en PRIORITY_HIGH:

```
.setPriority(NotificationCompat.PRIORITY_HIGH)
```

3. Añada una opción al builder para configurar el setAutoCancel en true, y otra opción para usar el patrón de la luz, sonido y vibración predeterminado:

```
.setAutoCancel(true)
.setDefaults(NotificationCompat.DEFAULT_ALL);
```

Entregar la notificación

1. Use el NotificationManager para entregar la notificación:

```
mNotificationManager.notify(NOTIFICATION_ID, builder.build());
```

2. Llame al método `deliverNotification()` cuando el botón toggle de alarma sea encendido, pasando el context de la actividad:

3. Llame al método `cancelAll()` en el NotificationManager si el toggle es apagado de manera de quitar la notificación.

```
if(isChecked){
   deliverNotification(MainActivity.this);
   //Set the toast message for the "on" case
   toastMessage = getString(R.string.alarm_on_toast);
} else {
   //Cancel notification if the alarm is turned off
   mNotificationManager.cancelAll();

   //Set the toast message for the "off" case
   toastMessage = getString(R.string.alarm_off_toast);
}
```

4. Ejecute la aplicación, y verifique que la notificación es entregada con todas las opciones deseadas.

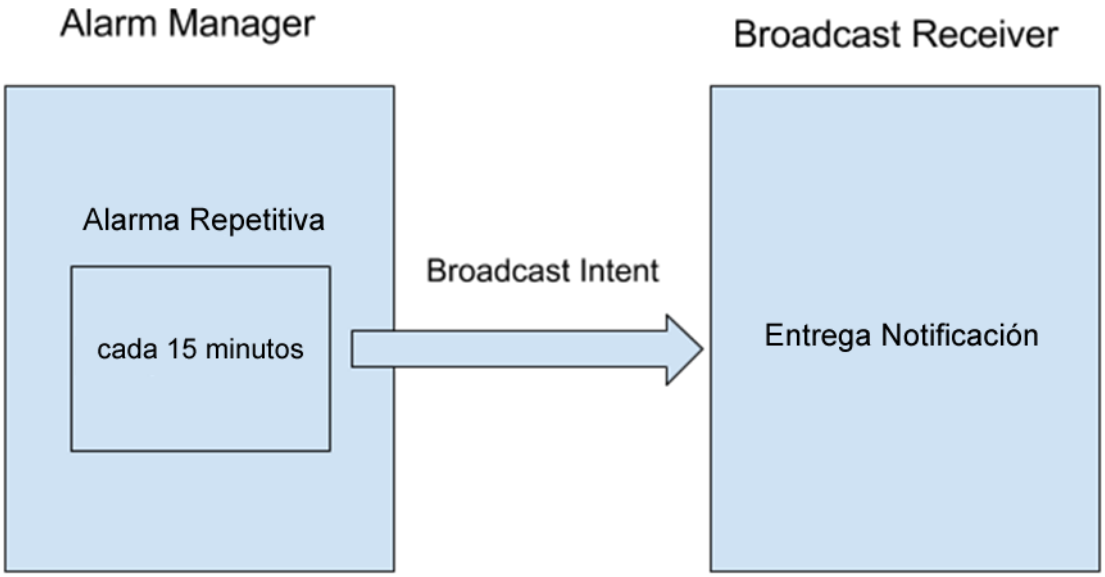
En este punto no hay ninguna alarma configurada en absolute: la notificación es entregada inmediatamente cuando el toggle de la alarma sea encendido. En la siguiente sección se implementará el AlarmManager para programar y entregar la notificación cada 15 minutos.

Tarea 3. Crear la repetición de la alarma

Ahora que la aplicación puede enviar una notificación, es el momento de implementar el componente principal de la aplicación: el AlarmManager. Esta es la clase que será responsable de entregar periódicamente el recordatorio de ponerse de pie. AlarmManager tiene muchos tipos de alarmas incorporadas, tanto únicas como periódicas, exactas e inexactas. Para obtener más información sobre los diferentes tipos de alarmas, consulte Scheduling Repeating Alarms.

Al igual que las notificaciones, AlarmManager utiliza un PendingIntent que se entrega con las opciones especificadas. Debido a esto, puede entregar el Intent incluso cuando la aplicación ya no se está ejecutando. En esta aplicación, el PendingIntent entregará un Intent de broadcast con una acción "Notify" personalizada.

El intent de broadcast será recibido por un broadcast receiver el cual tomará la acción apropiada (entregar la notificación).



El AlarmManager puede disparar eventos únicos o recurrentes los cuales se producen incluso cuando el dispositivo está en reposo profundo o cuando la aplicación no se está ejecutando. Los eventos pueden programarse con una selección de currentTimeMillis() cuando se utiliza la versión en tiempo real (RTC) o bien con elapsedRealtime() cuando se utiliza la versión en tiempo real transcurrido (ELAPSED_REALTIME), y entregar un PendingIntent cuando ocurra el seleccionado. Para obtener más información sobre los diferentes relojes disponibles e información sobre cómo controlar la temporización de los eventos, consulte SystemClock Developer Reference.

3.1 Configurar el intent de broadcast pendiente

El AlarmManager es responsable de entregar el PendingIntent en un intervalo especificado. Esta PendingIntent entregará un intent para broadcast que hará saber a la aplicación que ha llegado el momento de actualizar el tiempo restante de la notificación.

1. Cree una variable miembro como constante tipo String en el MainActivity para utilizarla como acción de un broadcast intent el cual enviará la notificación:

```
private static final String ACTION_NOTIFY =
    "com.example.android.standup.ACTION_NOTIFY";
```

Nota: usar el nombre completo del paquete como parte del string del Intent, asegura que el Broadcast será único y que no puede ser usado accidentalmente por otras aplicaciones con acciones similares.

2. Cree en el método `onCreate()` un Intent llamado notifyIntent con el string personalizado como su acción:

```
Intent notifyIntent = new Intent(ACTION_NOTIFY);
```

3. Cree el PendigIntent notificador usando el contexto, la variable NOTIFICATION_ID, el nuevo intent de notificación y la constante FLAG_UPDATE_CURRENT del PendingIntent:

```
PendingIntent notifyPendingIntent = PendingIntent.getBroadcast
   (this, NOTIFICATION_ID, notifyIntent, PendingIntent.FLAG_UPDATE_CURRENT);
```

3.2 Configure la alarma repetitiva

Ahora usará el AlarmManager para entregar este broadcast Intent cada 15 minutos. Para esta tarea, el tipo apropiado de alarma es una alarma inexacta y repetitiva que utiliza el tiempo transcurrido y despertará al dispositivo si está dormido. El reloj en tiempo real no es relevante aquí, ya que queremos entregar la notificación cada quince minutos.

1. Inicialice en el método `onCreate()` el AlarmManager llamando a `getSystemService()`:

```
AlarmManager alarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);
```

2. En el método `onCheckedChanged()`, llame a `setInexactRepeating()` en la instancia del alarm manager cuando el usuario haga clic en la Alarma "ON" (El segundo parámetro es `true`). Usará la alarma `setInexactRepeating()` ya que es más eficiente en cuanto a recursos usar tiempos inexactos (ya que el sistema puede agrupar alarmas de diferentes aplicaciones) y es aceptable que su alarma se desvíe un poco del intervalo exacto de repetición de 15 minutos. El método `setInexactRepeating()` toma 4 argumentos:

3. El alarm type (tipo de alarma). En este caso se utilizará el tiempo transcurrido desde el tiempo de arranque, ya que sólo el tiempo relativo es importante. Además, también se desea despertar el dispositivo si está dormido, por lo que el tipo de alarma es ELAPSED_REALTIME_WAKEUP.

4. El tiempo de disparo en milisegundos. Para ello, utilice el tiempo transcurrido actual, más 15 minutos. Para obtener el tiempo transcurrido actual, puede llamar a `SystemClock.elapsedRealtime()`. A continuación, puede utilizar una constante incorporada del AlarmManager para añadir 15 minutos al tiempo transcurrido: `AlarmManager.INTERVAL_FIFTEEN_MINUTES`.

5. El intervalo de tiempo en milisegundos. Se desea que la notificación se publique cada 15 minutos. Para ello, uede volver a utilizar la constante AlarmManager.INTERVAL_FIFTEEN_MINUTES.

6. El PendingIntent a entregar. Ya se ha creado el PendingIntent en la tarea anterior.

```
long triggerTime = SystemClock.elapsedRealtime()
       + AlarmManager.INTERVAL_FIFTEEN_MINUTES;

long repeatInterval = AlarmManager.INTERVAL_FIFTEEN_MINUTES;

//If the Toggle is turned on, set the repeating alarm with a 15 minute
interval
alarmManager.setInexactRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP,
       triggerTime, repeatInterval, notifyPendingIntent);
```

Nota: Dado que se está accediendo al AlarmManager y la instancia notifyPendingIntent desde una clase interna anónima, Android Studio puede hacer que estas instancias sean final. Si no lo hace, tiene que declararlos final usted mismo.

7. Remueva la llamada al `deliverNotification()` en el método `onCheckedChanged()`.

8. SI el toggle de la alarma esta apagado (haciendo click en el toggle en el estado ON), cancele la alarma llamando `cancel()` en el AlarmManager, pasando el pending intent usado para crear la alarma.

```
alarmManager.cancel(notifyPendingIntent);
```

   Mantenga la llamada al `cancelAll()` en el NotificationManager, ya que al desactivar el toggle se debería eliminar cualquier notificación existente.

El AlarmManager comenzará a entregar su intención de transmisión a partir de quince minutos desde el momento en que se configuró la Alarm, y luego con intervalos de cada quince minutos a partir de ese momento. La aplicación debe ser capaz de responder a estos intents mediante la entrega de la notificación. En el siguiente paso se creará una subclase de BroadcastReceiver para recibir los bradcast intents y entregar la notificación.

3.3 Crear el Broadcast Receiver

El Broadcast Receiver es responsable de recibir los broadcast intents del AlarmManager y reaccionar apropiadamente.

1. En Android Studio, haga clic en **File > New > Other > Broadcast Receiver**.

2. Introduzca AlarmReceiver como el nombre, asegúrese de que el checkbox Exported no esté marcado (para asegurarse de que otras aplicaciones no puedan invocar este Broacast Receiver). También puede cambiar esta configuración en el AndroidManifest configurando el atributo `android:exported` a `false`. Android Studio creará la subclase de BroadcastReceiver con el método requerido (`onReceive()`), además añadirá el receptor al AndroidManifest. Se necesita añadir un Intent Filter a la etiqueta `<receiverr>` en el AndroidManifest para seleccionar los apropiados Broadcast Intents entrantes.

3. En el Android Manifest, cree un `<intent-filter>` abriendo y cerrrando una etiqueta entre las etiquetas `<receiver>`. Cree un elemento `<action>` en el intent filter cone el atributo

`android:name` colocado como la acción de string personalizada ACTION_NOTIFY que ya fue creada:

```
<intent-filter>
    <action android:name="com.example.android.standup.ACTION_NOTIFY" />
</intent-filter>
```

4. Corte y pegue el método `deliverNotification()` al método `onReceive()` en el BroadcastReceiver y llámelo desde `onReceive()`. El notification manager y el ID de notificación no se han inicializado en la clase BroadcastReceiver, por lo que se resaltarán en rojo.

5. Copie la variable NOTIFICATION_ID de MainActivity en la clase BroadcastReceiver.

6. Inicialice el NotificationManager al principio del método `onReceive()`. Tiene que llamar a getSystemService() desde el Contexto pasado:

```
NotificationManager notificationManager = (NotificationManager)
   context.getSystemService(Context.NOTIFICATION_SERVICE);
```

7. Remueva la línea que indica la UnsupportedOperationException.

8. Ejecute la aplicación. Si no desea esperar quince minutos para ver la notificación, puede cambiar el tiempo de activación a `SystemClock.elapsedRealtime()` para ver la notificación inmediatamente. También puede cambiar el intervalo a un tiempo más corto para asegurarse de que la alarma repetitiva está funcionando.

Ahora se tiene una aplicación que se puede programar y que realiza una operación repetitiva, incluso si la aplicación ya no se está ejecutando. Ahora, salga de la aplicación completamente, la notificación seguirá siendo entregada. Falta un último componente que aseguraría una experiencia de usuario adecuada: si se sale de la aplicación, el botón toggle se reajustará al estado de apagado, incluso si la alarma ya se ha configurado. Para solucionar este problema, deberá comprobar el estado de la alarma cada vez que se ejecute la aplicación.

3.5 Verificar el estado de la alarma

Para rastrear el estado de la alarma, se necesitará una variable booleana que sea `true` si la alarma ya existe, y `false` en caso contrario. Para establecer este booleano, puede llamar a `PendingIntent.getBroadcast()` con el indicador FLAG_NO_CREATE del PendingIntent. En este caso, el PendingIntent es devuelto si ya existe, de lo contrario la llamada devuelve `null`. Esto es extremadamente útil para comprobar si la alarma ya ha sido configurada.

Nota: Al crear un PendingIntent, el sistema utiliza el método Intent.filterEquals() para determinar si ya existe un PendingIntent con el mismo Intent. Esto significa que para tener dos PendingIntent distintos, los Intents contenidos tienen que diferir en una de acción, datos, tipo, clase o categorías. Los extras de intención no se incluyen en la comparación.

El indicador PendienteIntención determina lo que sucede cuando ya existe un Pendiente cuya Intención coincide con la que está intentando crear. En el caso de la bandera NO_CREATE, devolverá null a menos que ya exista una PendienteIntención con una Intención coincidente.

1. Cree un booleano que sea verdadero si PendingIntent no es nulo, y falso de lo contrario, usando esta estrategia. Use este booleano para establecer correctamente el estado del ToggleButton cuando se inicie la aplicación. Este código tiene que venir antes de que se haya creado el PendingIntent, de lo contrario siempre volverá a ser verdadero:

```
boolean alarmUp = (PendingIntent.getBroadcast(this, NOTIFICATION_ID,
                   notifyIntent, PendingIntent.FLAG_NO_CREATE) != null);
```

2. Coloque el estado verificado del toggle justo después de haber definido el booleano alarmUp:

```
alarmToggle.setChecked(alarmUp);
```

Esto asegura que el toggle siempre estará encendido si la Alarm está activada, y apagado en caso contrario. Eso es todo, ahora se tiene una alarma programada repetitiva para recordarle que se pongas de pie cada quince minutos.

3. Ejecute la aplicación. Encienda la alarma. Salga de la aplicación. Abra la aplicación de nuevo. El botón de alarma mostrará que la alarma está activada.

Resumen:

- AlarmManager le permite programar tareas basándose en el reloj en tiempo real o en el tiempo transcurrido desde el inicio.
- AlarmManager proporciona una variedad de tipos de alarmas, tanto periódicas como de una sola vez, con opciones para despertar su dispositivo si está dormido.
- AlarmManager está diseñado para situaciones en las que es fundamental una sincronización precisa (por ejemplo, un evento de calendario). De lo contrario, considere el framework Job Scheduler para una programación más eficiente de los recursos.
- Utilice la versión de temporización inexacta del AlarmManager siempre que sea posible, para minimizar la carga causada por múltiples dispositivos de usuarios o múltiples aplicaciones que realizan una tarea exactamente al mismo tiempo.
- El AlarmManager utiliza los PendingIntents para realizar las operaciones, de modo que usted puede programar emisiones (broadcast), servicios y actividades utilizando el PendingIntent apropiado.

6.6: Job Scheduler

Hemos visto que se puede activar eventos basados en el reloj en tiempo real o en el tiempo transcurrido desde el inicio utilizando la clase AlarmManager. Sin embargo, la mayoría de las tareas no requieren una hora exacta, sino que deben programarse en base a una combinación de requisitos del sistema y del usuario. Por ejemplo, una aplicación de noticias podría querer actualizar las noticias por la mañana, pero podría esperar hasta que el dispositivo se esté cargando y conectado a wifi para actualizar las noticias, para preservar los datos del usuario y los recursos del sistema.

La clase JobScheduler está pensada para este tipo de programación; permite establecer las condiciones o parámetros de ejecución de la tarea. Dadas estas condiciones, el JobScheduler calcula el mejor momento para programar la ejecución del trabajo. Algunos ejemplos de estos parámetros son: persistencia del trabajo a través de reinicios, el intervalo en el que debe ejecutarse el trabajo, si el dispositivo está conectado o no, o si el dispositivo está inactivo o no.

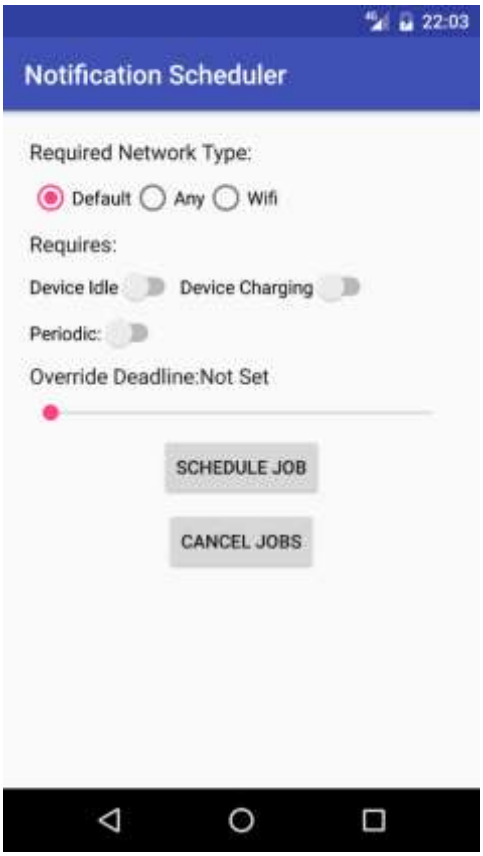
La tarea a ejecutar se implementa como una subclase de JobService y se ejecuta según las restricciones especificadas.

JobScheduler sólo está disponible en dispositivos que ejecutan API 21+ y actualmente no está disponible en la support library. Para compatibilidad con versiones anteriores, utilice GcmNetworkManager (que pronto será FirebaseJobDispatcher).

En esta práctica, se creará una aplicación que programe una notificación para que se contabilice cuando se cumplan los parámetros establecidos por el usuario y se cumplan los requisitos del sistema.

Resumen de la Aplicación

Para esta práctica se creará una aplicación llamada "Notification Scheduler". La aplicación mostrará el framework JobScheduler permitiendo al usuario seleccionar restricciones y programar (schedule) un trabajo. Cuando se ejecuta ese trabajo, publicará una notificación (en esta aplicación, la notificación es efectivamente su "trabajo").



Para utilizar el JobScheduler, necesita dos partes adicionales: JobService y JobInfo. Un objeto JobInfo contiene el conjunto de condiciones que desencadenarán la ejecución del job. Un JobService es la implementación del job que debe ejecutarse en esas condiciones.

Tarea 1. Implementar un JobService

Para empezar, se debe crear un servicio que se ejecutará en el momento determinado por las condiciones. El sistema ejecuta automáticamente el JobService, y las únicas partes que se necesita implementar son:

onStartJob() callback

- llamado cuando el sistema determina que el trabajo deba ser ejecutado. El job ha ser realizado debe ser implementado en este método.

> **Nota:** `onStartJob()` es ejecutado en el hilo principal, y por lo tanto cualquier tarea de larga duración debe ser descargada a un hilo diferente. En este caso, simplemente se está contabilizando un aviso, que se puede hacer de forma segura en el hilo principal.

- retorna un booleano indicando si el job necesita continuar en un hilo separado. Si es true, el trabajo se descarga a un hilo diferente, y la aplicación debe llamar a `jobFinished()` explícitamente en ese hilo para indicar que el job se ha completado. Si el valor de retorno es falso, el framework sabe que el job está terminado al final de `onStartJob()` y automáticamente llamará a `jobFinished()` en su nombre.

onStopJob() callback

- llamado si las condiciones no se completan, significando que el job deba ser parado.
- retorna un boolean que determina que hacer si el job no se ha terminado. Si el valor de retorno es true, el job será reprogramado, de lo contrario se cancelará.

1.1 Crear el Proyecto y el NotificationJobService

Verificar que el minimum SDK que está usando es el API 21. Anterior al API 21, JobScheduler no trabajará, ya que le faltan algunas de las APIs requeridas.

1. Use la plantilla vacía y cree un Nuevo Proyecto llamado "Notification Scheduler".
2. Cree una nueva clase Java llamada NotificationJobService que extienda a JobService.
3. Agregue los métodos requeridos: `onStartJob()` and `onStopJob()`.
4. En el archovo AndroidManfiest.xml, register el JobService con los siguientes permisos dentro de la etiqueta `<applicaction>`:

```
<service
   android:name=".NotificationJobService"
   android:permission="android.permission.BIND_JOB_SERVICE"/>
```

1.2 Implementar onStartJob()

1. Agregue un ícono para la notificación "Job Running".
2. En el método `onStartJob()`, cree un PendingIntent para iniciar la MainActivity de la aplicación para que sea usadacomo el content intent de la notificación.
3. En el método `onStartJob()`, construya y entregue una notificación con los siguientes atributos:

Atributo	Título
Content Title	"Job Service"
Content Text	"Your Job is running!"
Content Intent	contentPendingIntent
Small Icon	R.drawable.ic_job_running
Priority	NotificationCompat.PRIORITY_HIGH
Defaults	NotificationCompat.DEFAULT_ALL
AutoCancel	true

4. Asegúrese que `onStartJob()` retorna false, debido a que todo el trabajo es completado en ese callback.

5. Haga que el `onStopJob()` retorna true, para que el job sea reprogramado si ha fallado.

```
@Override
public boolean onStartJob(JobParameters jobParameters) {
   //Set up the notification content intent to launch the app when clicked
   PendingIntent contentPendingIntent = PendingIntent.getActivity
           (this, 0, new Intent(this, MainActivity.class),
            PendingIntent.FLAG_UPDATE_CURRENT);

   NotificationManager manager =
       (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

   NotificationCompat.Builder builder = new NotificationCompat.Builder(this)
           .setContentTitle(getString(R.string.job_service))
           .setContentText(getString(R.string.job_running))
           .setContentIntent(contentPendingIntent)
           .setSmallIcon(R.drawable.ic_job_running)
           .setPriority(NotificationCompat.PRIORITY_HIGH)
           .setDefaults(NotificationCompat.DEFAULT_ALL)
           .setAutoCancel(true);

   manager.notify(0, builder.build());

   return false;
}
```

Tarea 2. Implementar las condiciones del job

Ahora que se tiene el JobService, es el momento de identificar los criterios para ejecutar el job. Para ello, utilice el componente JobInfo. Se creará una serie de condiciones parametrizadas para ejecutar un job utilizando una variedad de tipos de conectividad de red y el estado del dispositivo.

Para empezar, crearemos un grupo de botones de opción para determinar el tipo de red requerido para este job.

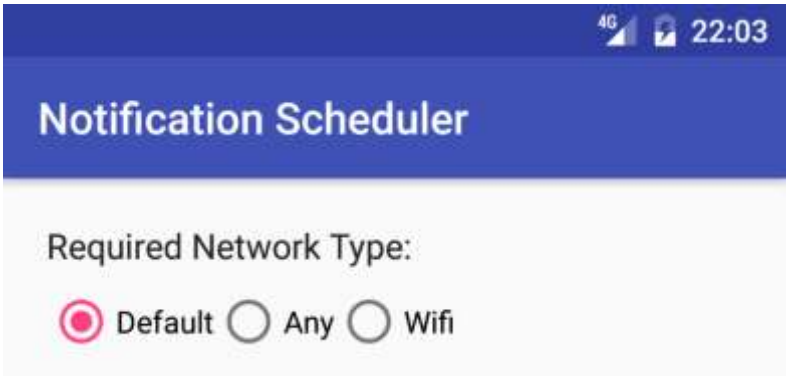
2.1 Implementar las restricciones de red

Una de las condiciones posibles para ejecutar un Job es el estado de la conectividad de red del dispositivo. Sólo se puede limitar la ejecución de JobService cuando se cumplen determinadas condiciones de red. Las opciones son:

- NETWORK_TYPE_NONE: el trabajo se ejecutará con o sin conexión de red. Este es el valor por defecto.
- NETWORK_TYPE_ANY: el job funcionará mientras haya una red (celular, wifi) disponible.
- NETWORK_TYPE_UNMETERED: el trabajo se ejecutará mientras el dispositivo esté conectado a un wifi que no utilice un HotSpot.

Crear el layout de la aplicación

Cree el layout de la aplicación para mostrar los botones para que el usuario escoja el criterio de red.



1. En el archivo activity_main.xml, cambie el element raiz de la vista a ***vertical*** Linear Layout.
2. Cambie el TextView para que tenda los siguientes atributos:

Atributo	**Valor**
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:text	"Network Type Required: "
android:textAppearance	"@style/TextAppearance.AppCompat.Subhead"
android:layout_margin	"4dp"

3. Agregue un elemento RadioGroup debajo del TextView con los siguientes atributos:

Atributo	**Valor**
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:orientation	"horizontal"
android:id	"@+id/networkOptions"
android:layout_margin	"4dp"

Nota: El uso de un grupo de radio (radio group) asegura que sólo uno de radio buttons pueda ser seleccionado a la vez. Para obtener más información sobre los Radio Buttons, consulte esta guía.

4. Agregue tres RadioButtons al RadioGroup con los atributos alto y ancho del layout en "wrap_content" y estos otros atributos:

RadioButton 1	
android:text	"None"
android:id	"@+id/noNetwork"
android:checked	true
RadioButton 2	
android:text	"Any"
android:id	"@+id/anyNetwork"
RadioButton 3	
android:text	"Wifi"
android:id	"@+id/wifiNetwork"

5. Añada dos botones debajo del grupo de botones de selección con el alto y el ancho ajustadas a "wrap content" con los siguientes atributos:

Button 1	
android:text	"Schedule Job"
android:onClick	"scheduleJob"
android:layout_gravity	"center_horizontal"
android:layout_margin	"4dp"
Button 2	
android:text	"Cancel Jobs"
android:onClick	"cancelJobs"
android:layout_gravity	"center_horizontal"
android:layout_margin	"4dp"

6. Agregue las plantillas de método de los dos métodos onClick en la MainActivity.

Obtener la opción de red seleccionada

1. En `scheduleJob()`, busque el RadioGroup por id y guárdelo en una variable de instancia llamada networkOptions.

2. Obtenga el id de la red seleccionada y guárdelo en una variable entera:

```
int selectedNetworkID = networkOptions.getCheckedRadioButtonId();
```

3. Cree una variable entera de opción de red seleccionada y configúrela igual a la opción de red predeterminada (red no requerida):

```
int selectedNetworkOption = JobInfo.NETWORK_TYPE_NONE;
```

4. Cree una sentencia switch con el id de la red seleccionada y agregur un case para cada una de las posibilidades del id:

```
switch(selectedNetworkID){
    case R.id.noNetwork:
        break;
    case R.id.anyNetwork:
        break;
    case R.id.wifiNetwork:
        break;
}
```

5. Asigne a la opción de red seleccionada la constante de red apropiada JobInfo, dependiendo del case:

```
switch(selectedNetworkID){
   case R.id.noNetwork:
       selectedNetworkOption = JobInfo.NETWORK_TYPE_NONE;
       break;
   case R.id.anyNetwork:
       selectedNetworkOption = JobInfo.NETWORK_TYPE_ANY;
       break;
   case R.id.wifiNetwork:
       selectedNetworkOption = JobInfo.NETWORK_TYPE_UNMETERED;
       break;
}
```

Crear el JobScheduler y el objecto JobInfo

1. En MainActivity, cree una variable de miembro para el JobScheduler, e inicialícela como `scheduleJob()` usando `getSystemService()`:

```
mScheduler = (JobScheduler) getSystemService(JOB_SCHEDULER_SERVICE);
```

2. Cree una constant miembro para el JOB_ID y defínala igual a 0.

3. Cree un objecto JobInfo.Builder en `scheduleJob()`. El constructor de la clase JobInfo.Builder toma dos parámetros:

    - El JOB_ID.
    - El ComponentName del JobService ya creado. Un ComponentName es usado para identificar al JobService con el objeto JobInfo.

```
ComponentName serviceName = new ComponentName(getPackageName(),
NotificationJobService.class.getName());
JobInfo.Builder builder = new JobInfo.Builder(JOB_ID, serviceName)
```

4. Llame al `setRequiredNetworkType()` en el objeto JobInfo.Builder, pasando la opción de red seleccionada:

```
.setRequiredNetworkType(selectedNetworkOption);
```

5. Llame al método `schedule()` del objeto JobScheduler, pasando el objeto JobInfo construido con el método build():

```
JobInfo myJobInfo = builder.build();
mScheduler.schedule(myJobInfo);
```

6. Muestre un mensaje Toast, permitiendo al usuario conocer el job que fue programado.

7. En el método `cancelJobs()` verifique si el objeto JobScheduler es null, y si no, llame al `cancelAll()` allí para remover todos los jons pendientes, resetee el JobScheduler para que sea null y muestre un mensaje Toast que le permita al usuario saber que el job fue cancelado:

```
if (mScheduler!=null){
   mScheduler.cancelAll();
   mScheduler = null;
   Toast.makeText(this, "Jobs Canceled", Toast.LENGTH_SHORT).show();
}
```

8. Ejecute la aplicación. Ahora puede configurar tareas que tienen restricciones de red y ver cuánto tiempo tardan en ejecutarse. En este caso, la tarea es entregar una notificación. Para rechazar la notificación deslícela hacia un lado o bien pulse sobre ella para abrirla.

Puede haber notado que si no cambia la restricción de red a “Any” o “Wifi”, la aplicación You may notice that if you do not change the network constraint to either "Any" or "Wifi", la aplicación se colgará con la siguiente excepción:

```
java.lang.IllegalArgumentException:
   You're trying to build a job with no constraints, this is not allowed.
```

Esto se debe a que la condición "No Network Required" es la predeterminada y no cuenta como una restricción. El JobScheduler necesita al menos una restricción para programar correctamente el JobService. En la siguiente sección se creará un condicional que es verdadero cuando al menos una restricción está definida, y falso de lo contrario. A continuación, programará la tarea si es verdadera y mostrará un Toast para indicar al usuario que establezca una restricción si no lo es.

2.2 Verificación de las restricciones

JobScheduler requiere que se establezca al menos una restricción. En esta tarea creará un booleano que rastreará si se ha cumplido tal requisito, para que se pueda notificar al usuario que establezca al menos una restricción, si aún no lo ha hecho. A medida que cree opciones adicionales en los pasos siguientes, necesitará modificar este booleano para que siempre sea verdadero si se establece al menos una restricción, y falso en caso contrario.

1. Cree una variable booleana llamada constraintSet que sea verdadera si la opción de red seleccionada no es la predeterminada JobInfo.NETWORK_TYPE_NONE:

```
boolean constraintSet = selectedNetworkOption != JobInfo.NETWORK_TYPE_NONE;
```

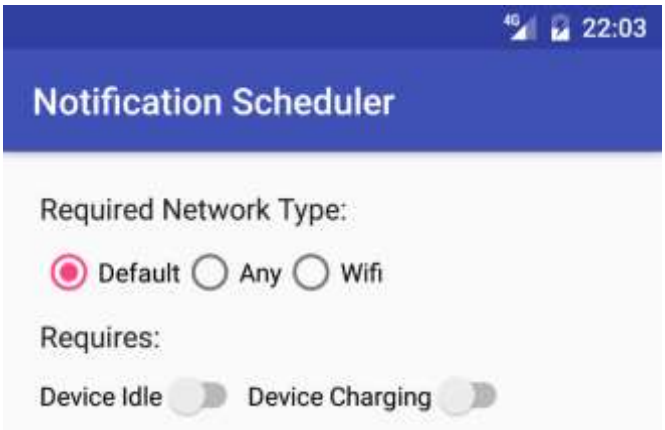
2. Cree un bloque if/else usando el booleano constraintSet.
3. Mueva el código que programa la tarea y muestra el mensaje Toast en el bloque if.
4. Si `constraintSet` es `false`, muestre un mensaje Toast al usuario para que configure al menos una restricción. No olvide de extraer los recursos de string:

```
if(constraintSet) {
   //Schedule the job and notify the user
   JobInfo myJobInfo = builder.build();
   mScheduler.schedule(myJobInfo);
   Toast.makeText(this, R.string.job_scheduled, Toast.LENGTH_SHORT).show();
} else {
   Toast.makeText(this, R.string.no_constraint_toast,
Toast.LENGTH_SHORT).show();
}
```

2.3 Implementar las restricciones Device Idle y Device Charging

JobScheduler incluye la capacidad de esperar hasta que el dispositivo se esté cargando, o en un estado inactivo (la pantalla está apagada, y la CPU se ha ido a dormir) para ejecutar su JobService. Ahora agregará cambios a su aplicación para alternar estas restricciones en su JobService.

Agregar element de UI para las nuevas restricciones



1. En el archivo activity_main.xml copie el TextView de la etiqueta de tipo de red y peguela por debajo del RadioGroup.

2. Cambie el atributo `android:text` para "Requires:".

3. Por debajo de ese textView inserte un Linearlayout con 4dp de margen.

4. Cree dos views Switch como hijos del LineraLayput horizontal con el alto y el ancho colocados en “wrap_content” y llos siguientes atributos:

Switch 1	
android:text	"Device Idle"
android:id	"@+id/idleSwitch"
Switch 2	
android:text	"Device Charging"
android:id	"@+id/chargingSwitch"

Agregar el código para las nuevas restricciones

1. En MainActivity, cree las variables de miembro, `mDeviceIdle` y `mDeviceCharging`, para los switches e inicialícelos en `onCreate()`.

2. En el método `scheduleJob()`, agregue las siguientes llamadas para establecer las restricciones en el JobScheduler según la selección del usuario en los switches:

```
builder.setRequiresDeviceIdle(mDeviceIdle.isChecked());
builder.setRequiresCharging(mDeviceCharging.isChecked());
```

3. Actualize el código que configure a `constraintSet` para considerara estas nuevas restricciones:

```
boolean constraintSet = (selectedNetworkOption != JobInfo.NETWORK_TYPE_NONE)
   || mDeviceChargingSwitch.isChecked() || mDeviceIdleSwitch.isChecked();
```

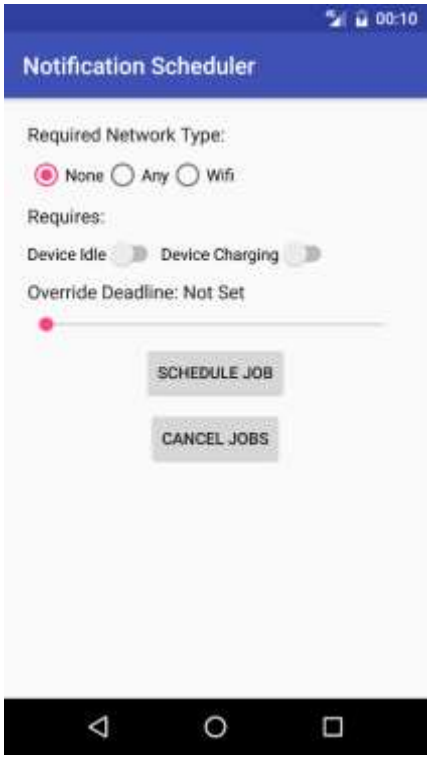
4. Ejecute la aplicación, ahora con las restricciones adicionales. Prueba diferentes combinanciones de switches para ver cuando las notificaciones son enviadas (lo que indica que el job se ejecutó). Puede probar la restricción de “estado cargando” en un emulador abriendo el menú (el icono de tres puntos al lado del emulador del dispositivo), ir al panel **Battery** y apagar el Battery Status. No hay una forma manual de colocar el emulador en modo inactivo (idle) al momento de escribir esta práctica.

Espere hasta que el dispositivo esté inactivo y enchufado es un patrón común para las tareas que requieren mucha batería, como descargar o cargar archivos grandes.

2.4 Implementar la restricción Override Deadline (Límite de Anulación)

Hasta este punto, no hay forma de saber con precisión cuándo el framework ejecutará su tarea. El sistema tiene en cuenta la gestión eficaz de los recursos, que puede retrasar la tarea en función del estado del dispositivo, y no garantiza que la tarea se ejecute a tiempo. Por ejemplo, una aplicación de noticias puede querer descargar las últimas noticias sólo cuando hay wifi disponible y el dispositivo esté conectado y cargando; pero un usuario puede olvidar inadvertidamente habilitar su wifi o cargar su dispositivo. Si no añade un parámetro de tiempo a su trabajo programado, el usuario se sentirá decepcionado cuando se despierte con las noticias de ayer. Por esta razón, la API de JobScheduler incluye la capacidad de establecer un límite que anula las restricciones anteriores.

Agregar la nueva UI para la configuración del límite para ejecutar la tarea



En este paso usará un nuevo componente de UI, una Seekbar, para permitir al usuario configurar el límte entre 0 y 100 segundos para ejecutar la tarea.

El usuario establece el valor arrastrando la SeekBar.

1. Cree un LinearLayout horizontal debajo del LinearLayout que contiene los switches, el cual contendrá las etiquetas de la SeekBar.

2. La SeekBar tendrá dos etiquetas: una estática como la etiqueta para el RadioGroup de botones, y otra dinámica que se actualizará con el valor de la SeekBar. Añada dos TextViews al LinearLayout con los siguientes atributos:

TextView 1	

android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:text	"Override Deadline: "
android:id	"@+id/seekBarLabel"
android:textAppearance	"@style/TextAppearance.AppCompat.Subhead"
TextView 2	
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:text	"Not Set"
android:id	"@+id/seekBarProgress"
android:textAppearance	"@style/TextAppearance.AppCompat.Subhead"

3. Agregue una view SeekBar por debajo del LinearLayout con los siguientes atributos:

Atributo	**Valor**
android:layout_width	"match_parent"
android:layout_height	"wrap_content"
android:id	"@+id/seekBar"
android:layout_margin	"4dp"

Escribir el código para agregar el límite

1. En la MainActivity, cree una variable miembro para la SeekBar e inicialicela en el `onCreate()`:

```
mSeekBar = (SeekBar) findViewById(R.id.seekBar);
```

2. Cree las variables final para ambos TextViews (ya que serán accedidos desde una clase interna) e inicialicelas en el `onCreate()`:

```
final TextView label = (TextView) findViewById(R.id.seekBarLabel);
final TextView seekBarProgress = (TextView) findViewById(R.id.seekBarProgress);
```

3. En `onCreate()`, llame al `setOnSeekBarChangeListener()` en la SeekBar, pasando un nuevo OnSeekBarChangeListener (Android Studio debe generar los métodos requeridos)

```
mSeekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
   @Override
   public void onProgressChanged(SeekBar seekBar, int i, boolean b) {}

   @Override
   public void onStartTrackingTouch(SeekBar seekBar) {}

   @Override
   public void onStopTrackingTouch(SeekBar seekBar) {}
});
```

4. El segundo argumento de `onProgressChanged()` es el valor actual de la SeekBar. En la llamada `onProgressChanged()`, compruebe si el valor entero es mayor que 0 (lo que significa que un valor ha sido fijado por el usuario), y si lo es, fije la etiqueta de progreso SeekBar al valor entero, seguido por "s" para indicar segundos:

```
if (i > 0){
    mSeekBarProgress.setText(String.valueOf(i) + " s");
}
Otherwise, set the TextView to read "Not Set":
else {
    mSeekBarProgress.setText("Not Set");
}
```

5. El límite de anulación sólo se debe establecer si el valor entero de la SeekBar es mayor que 0. En el método `scheduleJob()`, cree un número entero para almacenar el progreso de la SeekBar y una variable booleana que sea verdadera si la SeekBar tiene un valor entero mayor que 0:

```
int seekBarInteger = mSeekBar.getProgress();
boolean seekBarSet = seekBarInteger > 0;
```

6. Si el booleano es verdadero, llame a `setOverrideDeadline()` en el JobInfo.Builder, pasando el valor entero de la SeekBar multiplicado por 1000 (el parámetro está en milisegundos, usted quiere que el usuario establezca el límite de anulación en segundos):

```
if (seekBarSet) {
      builder.setOverrideDeadline(seekBarInteger * 1000);
}
```

7. Modifique el booleano `constraintSet` para incluir el valor de `seekBarSet` como una posible restricción:

```
boolean constraintSet = selectedNetworkOption != JobInfo.NETWORK_TYPE_NONE
   || mDeviceChargingSwitch.isChecked() || mDeviceIdleSwitch.isChecked()
   || seekBarSet;
```

8. Ejecute la aplicación. El usuario puede ahora configurar un límite en segundos para que el JobService deba ser ejecutado!

2.5 Implementar una restricción Períodica

JobScheduler también permite programar una tarea repetitiva, al igual que el AlarmManager. Esta opción tiene algunas salvedades:

- No se garantiza que la tarea se ejecute en un período determinado (es posible que no se cumplan las demás condiciones o que no haya suficientes recursos del sistema).
- El uso de esta restricción evita que también se establezca un límite de anulación o una latencia mínima (), ya que estas opciones no tienen sentido para tareas repetitivas. Consulte la documentación JobInfo.Builder para obtener más información.

Agregar el Switch Periodic al layout

Se añadirá un Switch para que el usuario pueda cambiar entre ejecutar el trabajo una o varias veces a intervalos periódicos.

1. En activity_main.xml, agregue una view Switch entre los dos LinearLayout horizontals. Use los siguientes atributos:

Atributo	Valor
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:text	"Periodic"
android:id	"@+id/periodicSwitch"
android:layout_margin	"4dp"

2. Cree una variable miembro para el switch e inicialicela en el `onCreate()`:

```
mPeriodicSwitch = (Switch) findViewById(R.id.periodicSwitch);
```

Escribir el código para usar el Switch Periodic

El límite de anulación y las restricciones periódicas son mutuamente excluyentes. Se usará el switch para alternar la funcionalidad y la etiqueta de la SeekBar para representar el límite de anulación o el intervalo periódico.

1. Llame a `setOnCheckedChangeListener()` en el switch periódico, pasando un nuevo `OnCheckedChangeListener`.
2. Si está checked, coloque la etiqueta en "Periodic Interval: ", de lo contrario en "Override Deadline: ":

```
mPeriodicSwitch.setOnCheckedChangeListener(
   new CompoundButton.OnCheckedChangeListener() {
   @Override
   public void onCheckedChanged(CompoundButton compoundButton, boolean
isChecked) {
       if (isChecked){
           label.setText(R.string.periodic_interval);
       } else {
           label.setText(R.string.override_deadline);
       }
   }
});
```

Lo único que queda por hacer es implementar la lógica en el método `scheduleJob()` para definir correctamente las restricciones en el objeto JobInfo.

Si la opción periódica está **activada** (**on**):

- Si la SeekBar tiene un valor **distinto de cero**, establezca la restricción llamando a `setPeriodic()` en el objeto JobInfo.Builder.
- Si la SeekBar tiene un **valor de 0**, muestre un mensaje Toast pidiendo al usuario que establezca un intervalo periódico con la SeekBar.

Si la opción periódica está **apagada** (**off**)**:**

- Si la SeekBar tiene un valor **distinto de cero**, el usuario debe establecer un límite de anulación. Aplique el límite de anulación usando la opción `setOverrideDeadline()`**.**
- Si la SeekBar tiene un **valor de 0**, el usuario simplemente no ha especificado un límite de anulación ni una tarea períodica, de manera que no hay que agregar nada al objeto JobInfo.Builder.
- Reemplace el código que configure el límite de anulación al JobInfo.Builder en `scheduleJob()` con el siguiente código para implementar esta lógica:

```
if (mPeriodicSwitch.isChecked()){
   if (seekBarSet){
       builder.setPeriodic(seekBarInteger * 1000);
   } else {
       Toast.makeText(MainActivity.this,
           "Please set a periodic interval", Toast.LENGTH_SHORT).show();
   }
} else {
   if (seekBarSet){
       builder.setOverrideDeadline(seekBarInteger * 1000);
   }
}
```

Resumen

- JobScheduler proporciona un marco flexible para realizar servicios en segundo plano de forma inteligente.
- JobScheduler sólo está disponible en dispositivos que ejecutan el API 21+.
- Para utilizar el JobScheduler, necesita dos partes: JobService y JobInfo.
- JobInfo es un conjunto de condiciones que desencadenarán la ejecución del job.
- JobService implementa el job para ejecutarse en las condiciones especificadas por JobInfo.
- Sólo tiene que implementar los métodos callback `onStartJob()` y `onStopJob()` en el JobService.
- La implementación del job ocurre (o es iniciado) en `onStartJob()`.
- `onStartJob()` devuelve un booleano que indica si el servicio necesita procesar el job en un hilo separado.
- Si `onStartJob()` devuelve `true`, debe llamar explícitamente a `jobFinished()`. Si `onStartJob()` devuelve `false`, el tiempo de ejecución llamará a `jobFinished()` por usted.
- JobService se procesa en el hilo principal, así que evite cálculos o I/O largos.
- JobScheduler agrupa las tareas para maximizar la eficiencia de los recursos del sistema, lo que significa que usted no tiene el control exacto de cuándo se ejecutará.

Capitulo 7

Manejando Datos

Android suministra diversas opciones para guardar de forma persistente los datos de una aplicación. La solución que se escoga depende de necesidades específicas, tales como si los datos serán privados para la aplicación a accequibles desde otras aplicaciones (y el usuario) y caunto espacio requieren esos datos.

Las opciones de almacenamiento de datos son las siguientes:

- Shared preferences—Almacenan datos primitivos en pares clave-valor. Se tratará en un ejercicio más adelante.
- Internal storage—Almacena los datos privados en la memoria del dispositivo.
- External storage—Almacena datos públicos en un almacen de datos compartidos externo.
- SQLite databases—Alamacena los datos estructurados en una base de datos privada.
- Network connection—Alamacena los datos en la web con un proveedor de red.
- Cloud Backup—Respaldo de los datos del usuari y la aplicaciónn enla nube.
- Content providers—Almacena los datos de forma privada y luego los hace disponible de forma pública. También se verá en un próximo ejercicio.
- Firebase realtime database—Almacena y sincroniza los datos en una base de datos NoSQL en la nube. Los datos son sincronizados entre todos los clients en tiempo real y permanencen disponibles cuando la aplicación queda fuera de línea.

7.1 Shared Preferences (Preferencias compartidas)

Las shared preferences permiten leer y escribir pequeñas cantidades de datos primitivos (como pares clave/valor) en un archivo en el almacenamiento del dispositivo. La clase SharedPreference proporciona APIs para manejar un archivo de preferencias y para leer, escribir y administrar estos datos. El archivo de shared preferences en sí mismo es administrado por el framework Android, y accesible a (compartido con) todos los componentes de la aplicación. Sin embargo, estos datos no se comparten ni son accesibles por ninguna otra aplicación.

Los datos que son guardados en las shared preferences son diferentes de los datos guardados del estado de la actividad en la que se trabajó en un capítulo anterior. Los datos del estado de instancia de una actividad se conservan en todas las instancias de la actividad en la misma sesión del usuario. Las shared preferences persisten en todas las sesiones de usuario, incluso si la aplicación se elimina y se reinicia o si el dispositivo se reinicia.

Se utilizan las shared preferences sólo cuando necesite guardar una pequeña cantidad de datos como simples pares clave/valor. Para gestionar grandes cantidades de datos de aplicaciones

persistentes, se utilizarán otros métodos como las bases de datos SQL, de las que trataremos en un capítulo posterior.

Resumen de la aplicación

La aplicación HelloSharedPrefs es otra variación de la aplicación HelloToast que creó en la Lección 1. Incluye botones para incrementar el número, cambiar el color de fondo y restablecer tanto el número como el color a sus valores predeterminados. La aplicación también utiliza temas y estilos para definir los botones.

Se comenzará con una aplicación de inicio para esta práctica y se agregarán shared preferences al código de la actividad principal. También se agregará un botón de reset que establece tanto el recuento como el color de fondo en el valor predeterminado y borra el archivo de preferencias.

Tarea 1. Explorar la aplicación HelloSharedPrefs

El proyecto completo de la aplicación de arranque para esta práctica está disponible en la carpeta “Archivos/Aplicaciones de Arranque/HelloSharedPrefs-Start.rar” del M7. En esta tarea cargará el proyecto en Android Studio y explorará algunas de las funciones clave de la aplicación.

1.1 Abrir y Ejecutar el Proyecto HelloSharedPrefs

1. Descargue la aplicación HelloSharedPrefs-Start y descomprima el archivo.
2. Abra la aplicación en el Android Studio.
3. Build y ejecute el proyecto en el Android Studio. Pruebe lo siguiente:
    - Haga clic en el botón Count para incrementar el número en el text view principal.

- Haga clic en cualquiera de los botones de color para cambiar el color de fondo del text view principal.
- Rote el dispositivo y note que tanto el fondo como la cuenta son preservados.
- Haga clic en el botón Reset para tener la cuenta y el color de vuelta a los valores predeterminados.

4. Force la salida de la aplicación usando uno de estos métodos:
    - En el Android Studio, seleccione Run > Stop 'app' o haga clic en el ícono de Stop  en la toolbar.
    - En el dispositivo, haga clic en el botón “Recientes” (el boton cuadrado en la esquina inferior derecha). Deslize la tarjeta de la aplicación HelloSharedPrefs para salir, o haga clic en la X en la esquina superior derecha. Si sale de la aplicación de esta manera, espere unos segundos antes de comenzarla nuevamente para permitirle al sistema realizar la limpieza correspondiente.
5. Vuelva a ejecutar la aplicación.

La aplicación se reinicia con la apariencia por predeterminada – la cuenta en 0 y el clor de fondo en gris.

1.2 Explore el código de la Activity

1. Abra el archivo MainActivity (java/com.example.android.simplecalc/MainActivity).
2. Examine el código y observe lo siguiente:
    - La cuenta (mCount) está definida mediante un número entero. El método de manejo de clics countUp() incrementa este valor y actualiza la vista de texto principal.
    - El color (mColor) es también un entero que es definido inicialmente como gris en el archivo de recursos colors.xml como default_background.
    - El método de manejo del clic changeBackground() obtiene el color de fondo del botón en el que se hizo clic y luego establece el color de fondo de la vista de texto principal.
    - Tanto el número como los enteros de color se guardan en el instance state bundle en el método onSaveInstanceState(), y se restauran en onCreate(). Las claves bundle para la cuenta y el color se definen mediante variables privadas (COUNT_KEY) y (COLOR_KEY).

Tarea 2. Guardar y restaurar los datos en un archivo shared preferences

En esta tarea se guardará el estado de la aplicación en un archivo shared preferences y volverá a leer los datos cuando se reinicie la aplicación. Dado que los datos de estado que está guardando en las shared preferences (la cuenta y el color) son los mismos datos que conserva en el instance

state, no tiene que hacerlo dos veces: puede reemplazar el instance state por el estado del shared preferences.

2.1 Inicialize las preferencias

1. Añada dos variables miembro a la clase MainActivity para mantener el nombre del archivo shared preferences y una referencia a un objeto SharedPreferences.

```
private SharedPreferences mPreferences;
private String sharedPrefFile = "com.example.android.hellosharedprefs";
```

Se puede nombrar el archive shared preferences como se desee, pero por convención se usa el mismo nombre del paquete de la aplicación

2. En el método onCreate, inicialize las preferencias compartidas. Asegúrese que inserta este código antes de la sentencia `if`

```
mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);
```

El métdo getSharedPreferences() abre el archivo con el nombre suministrado (sharedPrefFile) en el modo MODE_PRIVATE.

Nota: Las versiones anteriores de Android tenían otros modos que le permitían crear un archivo de shared preferences legible o escribible en todo el mundo. Estos modos fueron obsoletos en API 17, y ahora son fuertemente desaconsejados por razones de seguridad. Si necesita compartir datos con otras aplicaciones, utilice un service o un content provider.

Código de la Solución (Main Activity - partial)

```
public class MainActivity extends AppCompatActivity {
   private int mCount = 0;
   private TextView mShowCount;
   private int mColor;

   private SharedPreferences mPreferences;
   private String sharedPrefFile = "com.example.android.hellosharedprefs";

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);

       mShowCount = (TextView) findViewById(R.id.textview);
       mColor = ContextCompat.getColor(this, R.color.default_background);
       mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);

       // …
    }
}
```

2.2 Guardar las preferencias en onPause()

Guardar las preferencias es muy parecido a guardar el instance state: ambas operaciones llevan los datos a un objeto Bundle como un par clave/valor. Sin embargo, para las shared preferences, guarde los datos en el callback onPause() del ciclo de vida y necesitará un objeto shared editor (SharedPreferences.Editor) para escribir en el objeto shared preferences.

1. Haga clic en la última línea de la clase MainActivity, justo antes del corchete de cierre.
2. Seleccione Code > Generate y, a continuación, seleccione Override Methods.
3. Escriba "onPause", seleccione la firma de método para el método onPause() y haga clic en OK.

   Se añade una plantilla del método onPause() al punto de inserción.
4. Obtenga un editor para el objeto SharedPreferences:

```
SharedPreferences.Editor preferencesEditor = mPreferences.edit();
```

Se necesita un editor de shared preferences para escribir en el objeto shared preferences. Añada esta línea a onPause() después de la llamada a super.onPause().

5. Utilice el método putInt() para poner los enteros mCount y mColor en las shared preferences con las claves apropiadas:

```
preferencesEditor.putInt(COUNT_KEY, mCount);
preferencesEditor.putInt(COLOR_KEY, mColor);
```

La clase SharedPreferences.Editor incluye multiples métodos `put` para diferentes tipos de datos, incluyendo `putInt()` y `putString()`.

6. Llame al método apply() para guarder las preferencias:

```
preferencesEditor.apply();
```

El método `apply()` guarda las preferencias asincrónicamente, fuera del hilo de UI. El editor de shared preferences también tiene un método `commit()` para guardar las preferencias de forma sincronizada. El método `commit()` está desaconsejado ya que puede bloquear otras operaciones.

7. Elimine completamente el método onSaveInstanceState(). Dado que el instance state de la actividad contiene los mismos datos que las shared preferences, puede reemplazar el instance state por completo.

Código de la Solución

```
@Override
protected void onPause(){
   super.onPause();

   SharedPreferences.Editor preferencesEditor = mPreferences.edit();
   preferencesEditor.putInt(COUNT_KEY, mCount);
   preferencesEditor.putInt(COLOR_KEY, mColor);
   preferencesEditor.apply();
}
```

2.3 Resturar las preferencias en onCreate()

Al igual que con el instance state, la aplicación lee cualquier shared preferences guardada en el método onCreate(). Nuevamente, ya que las shared preferences contienen los mismos datos que el instance state, podemos reemplazar el estado con las preferencias también aquí. Cada vez que se llama a onCreate() -- cuando se inicia la aplicación o al cambiar la configuración -- las shared preferences se usan para restaurar el estado de la vista.

1. Localize la parte del método onCreate() que prueba si el argumento savedInstanceState es null y restaura el instance state:

```
if (savedInstanceState != null) {
   mCount = savedInstanceState.getInt(COUNT_KEY);
   if (mCount != 0) {
       mShowCountTextView.setText(String.format("%s", mCount));
   }
   mColor = savedInstanceState.getInt(COLOR_KEY);
   mShowCountTextView.setBackgroundColor(mColor);
}
```

2. Elimine todo el bloque.
3. En el método onCreate(), en el mismo lugar donde se encontraba el código del instance state guardado, obtenga el conteo de las preferencias con la clave COUNT_KEY y asígnelo a la variable mCount.

```
mCount = mPreferences.getInt(COUNT_KEY, 0);
```

   Cuando usted lee datos de las preferencias no necesita obtener un editor de shared preferences. Utilice cualquiera de los métodos get de un objeto shared preferences para recuperar los datos de preferencias.

   Tenga en cuenta que el método getInt() toma dos argumentos: uno para la clave, y el otro para el valor por defecto si no se puede encontrar la clave. En este caso, el valor por defecto es 0, que es el mismo que el valor inicial de mCount.

4. Actualize el valor del text view principal con la nueva cuenta.

```
mShowCountTextView.setText(String.format("%s", mCount));
```

5. Obtenga el color de las preferencias con la clave COLOR_KEY y asignela a la variable mColor.

```
mColor = mPreferences.getInt(COLOR_KEY, mColor);
```

Como antes, el segundo argumento para getInt() es el valor por defecto a usar en caso de que la clave no exista en las shared preferences. En este caso puede reutilizar el valor de mColor, que acaba de ser inicializado más arriba al color de fondo por defecto en el método.

6. Actualize el color de fondo para el text view principal.

```
mShowCountTextView.setBackgroundColor(mColor);
```

7. Ejecute la aplicación. Haga clic en el botón de cuenta y cambie el color de fondo para actualizar el instance state y las preferencias.
8. Rote el dispositivo o el emulador para verificar que la cuenta y el color son salvados a través de los cambios de configuración.
9. Force la salida de la aplicación usando uno de los métodos siguientes:
    - En el Android Studio, seleccione Run > Stop 'app.'
    - En el dispositivo, haga clic en el botón “Recientes” (el boton cuadrado en la esquina inferior derecha). Deslize la tarjeta de HelloSharedPrefs para salir, o haga clic en la X en la esquina superior derecha.
10. Reinicie la aplicación. La aplicación se reinicia y carga las preferencias manteniendo el estado.

SCódigo de la Solución (Main Activity - onCreate())

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);

   // Initialize views, color, preferences
   mShowCountTextView = (TextView) findViewById(R.id.count_textview);
   mColor = ContextCompat.getColor(this, R.color.default_background);
   mPreferences = getSharedPreferences(mSharedPrefFile, MODE_PRIVATE);

   // Restore preferences
   mCount = mPreferences.getInt(COUNT_KEY, 0);
   mShowCountTextView.setText(String.format("%s", mCount));
   mColor = mPreferences.getInt(COLOR_KEY, mColor);
```

```
    mShowCountTextView.setBackgroundColor(mColor);
}
```

2.4 Resetear las preferencias con el manejador de clic reset()

El botón de reset de la aplicación restablece tanto la cuenta como el color de la actividad a sus valores predeterminados. Puesto que las preferencias mantienen el estado de la actividad, es importante borrar también las preferencias al mismo tiempo.

1. En el método manejador del clic reset(), después que se han restablecido el color y la cuenta, obtenga un editor del objeto SharedPreferences:

```
SharedPreferences.Editor preferencesEditor = mPreferences.edit();
```

2. Elimine todas las shared preferences:

```
preferencesEditor.clear();
```

3. Aplique los cambios:

```
preferencesEditor.apply();
```

Código de la Solución (método reset()) :

```
public void reset(View view) {
    // Reset count
    mCount = 0;
    mShowCountTextView.setText(String.format("%s", mCount));

    // Reset color
    mColor = ContextCompat.getColor(this, R.color.default_background);
    mShowCountTextView.setBackgroundColor(mColor);

    // Clear preferences
    SharedPreferences.Editor preferencesEditor = mPreferences.edit();
    preferencesEditor.clear();
    preferencesEditor.apply();

}
```

Resumen

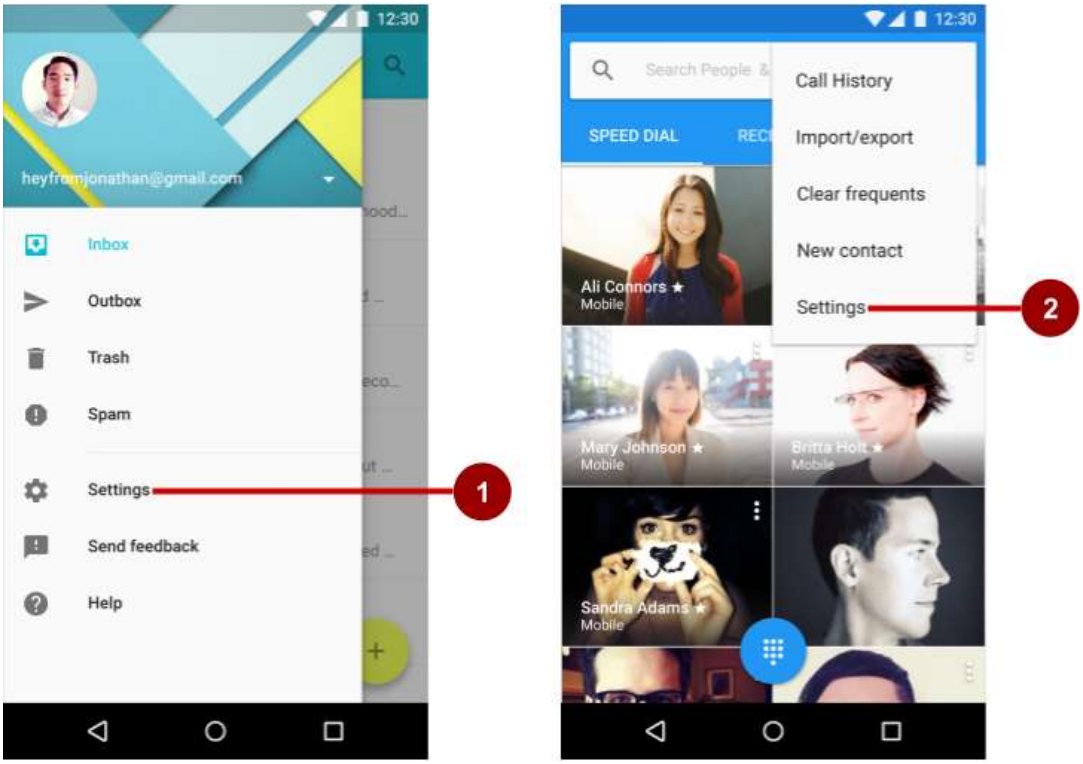
- La clase SharedPreferences permite a una aplicación guardar pequeñas cantidades de datos primitivos en pares clave-valor.

- Las shared preferences persisten a través de diferentes sesiones de usuario de la misma palicación.
- Para escribir las shared preferences, obtenga un objeto SharedPreferences.Editor.
- Use uno de los varios métodos put* del objeto SharedPreferences.Editor, tales como putInt() o putString(), para poner los datos en las shared preferences con una clave y un valor.
- Use uno de los varios métodos get* del objeto SharedPreferences, tales como getInt() o getString(), para obtener los datos desde las shared preferences con una clave.
- Use el método clear() del objeto SharedPreferences.Editor para remover todos los datos almacenados en las preferencias.
- Use el método apply() del objeto SharedPreferences.Editor para guarder los cambios al archivo de preferencias.

7. 2 Agregando Settings a una Aplicación

Las aplicaciones a menudo incluyen configuraciones que permiten a los usuarios modificar las características y comportamientos de la aplicación. Por ejemplo, algunas aplicaciones permiten a los usuarios establecer la ubicación de sus hogares, las unidades predeterminadas para las mediciones, las opciones de cena y otros ajustes que se aplican a toda la aplicación. Por lo general, el acceso a los ajustes es poco frecuente, ya que una vez que un usuario cambia un ajuste, como la ubicación de inicio, rara vez necesita volver atrás y cambiarlo de nuevo.

Los usuarios esperan navegar a la configuración de la aplicación tocando **Settings** (Configuración) en la navegación lateral, tal como un panel de navegación como se muestra en el lado izquierdo de la figura de abajo, o en el menú de opciones en la barra de aplicaciones, que se muestra en el lado derecho de la figura de abajo.



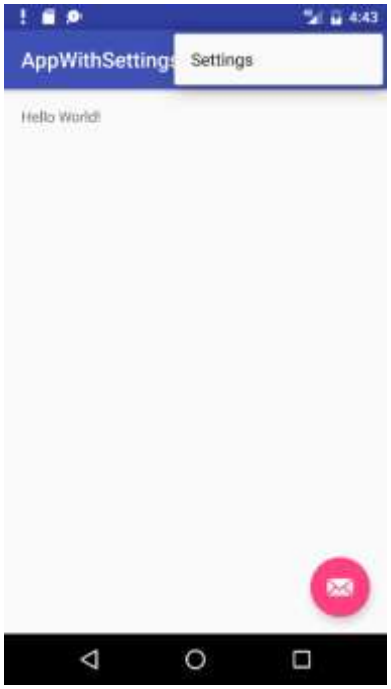
En la figura anterior:

1. **Settings** en la navegación lateral (un panel de navegación)
2. **Settings** en las opciones del menu de la barra de aplicaciones

En esta práctica se agregará una actividad de configuración a la aplicación. Los usuarios estarán en capacidad de navegar a la configuración de la aplicación tocando **Settings**, el cual estará localizado en el menú de opciones de la app bar.

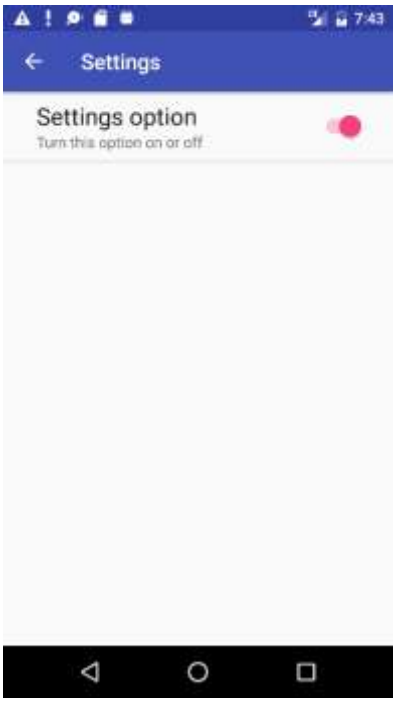
Resumen de la aplicación

Android Studio proporciona un acceso directo para configurar un menú de opciones con **Settings**. Si inicia un proyecto de Android Studio para un smartphone o tableta utilizando la plantilla Basic Activity, la nueva aplicación incluye la configuración que se muestra a continuación:



La plantilla incluye además un floating action button en la esquina inferior izquierda de la pantalla con un ícono de sobre. Puede ignorar este botón para esta práctica ya que no será usado.

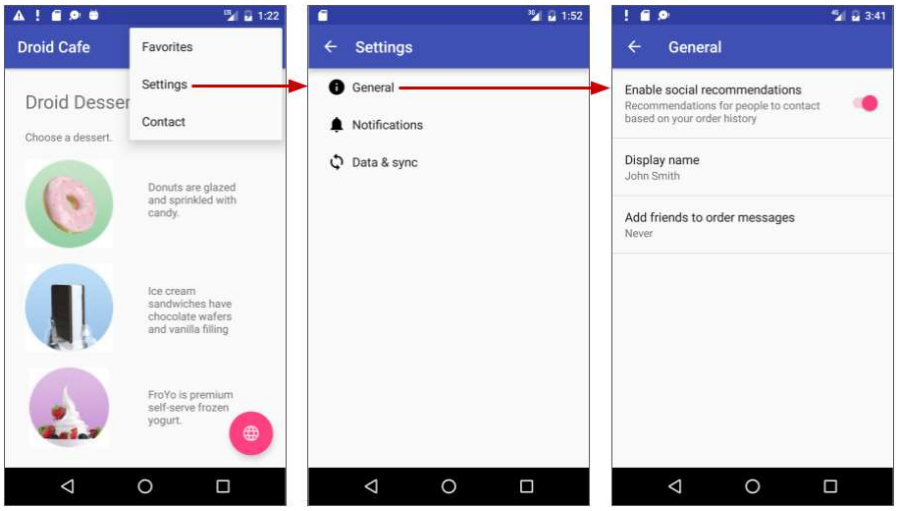
Se comenzará creando una aplicación llamada AppWithSettings usando la plantilla Basic Activity, y se agregará la actividad de configuración que suministra un switch toggle que permite al usuario prenderla o apagarla.



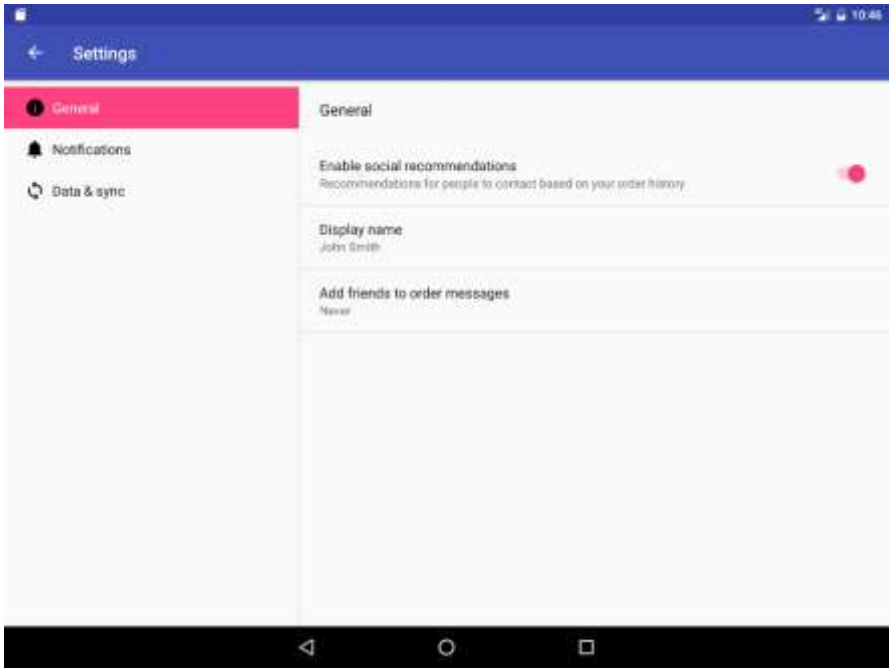
Añadirá código para leer la configuración y realizar una acción basada en su valor. En aras de la simplicidad, la acción será mostrar un mensaje toast con el valor configurado.

En la segunda tarea, añadirá la plantilla Settings Activity estándar proporcionada por Android Studio a la aplicación DroidCafe que creó en una lección anterior. La plantilla Settings Activity se completa previamente con configuraciones que ud puede personalizar para una aplicación y proporciona un diseño diferente para teléfonos inteligentes y tabletas:

- *Smartphones*: Una pantalla principal de Settings con un enlace de encabezado para un grupo de configuraciones, tales como General para austes generales, como se muestra abajo.



- *Tablets*: una pantalla master/detail con un enlace de encabezado por cada grupo del lado izquierdo (master) y el grupo de ajustes en el lado derecho (detail), como se muestra en la figura abajo.



Todo lo que necesita hacer para personalizar la plantilla es cambiar las cabeceras, los títulos de los ajustes, las descripciones de los ajustes y los valores de los ajustes, y escribir el código que normalmente escribiría para utilizar los valores de los ajustes.

La aplicación Droid Cafe se creó en una lección anterior a partir de la plantilla Basic Activity, que proporciona un menú de opciones en la barra de aplicaciones para colocar la opción **Settings**. Se personalizará la plantilla Setting Activity suministrada cambiando el título, la descripción, los valores y los valores predeterminados en una única configuración. Se agregará código para leer el valor del ajuste después de que el usuario lo cambie, y mostrará ese valor.

Tarea 1: Agregar un switch de ajuste para una aplicación

En esta tarea, se realizará lo siguiente:

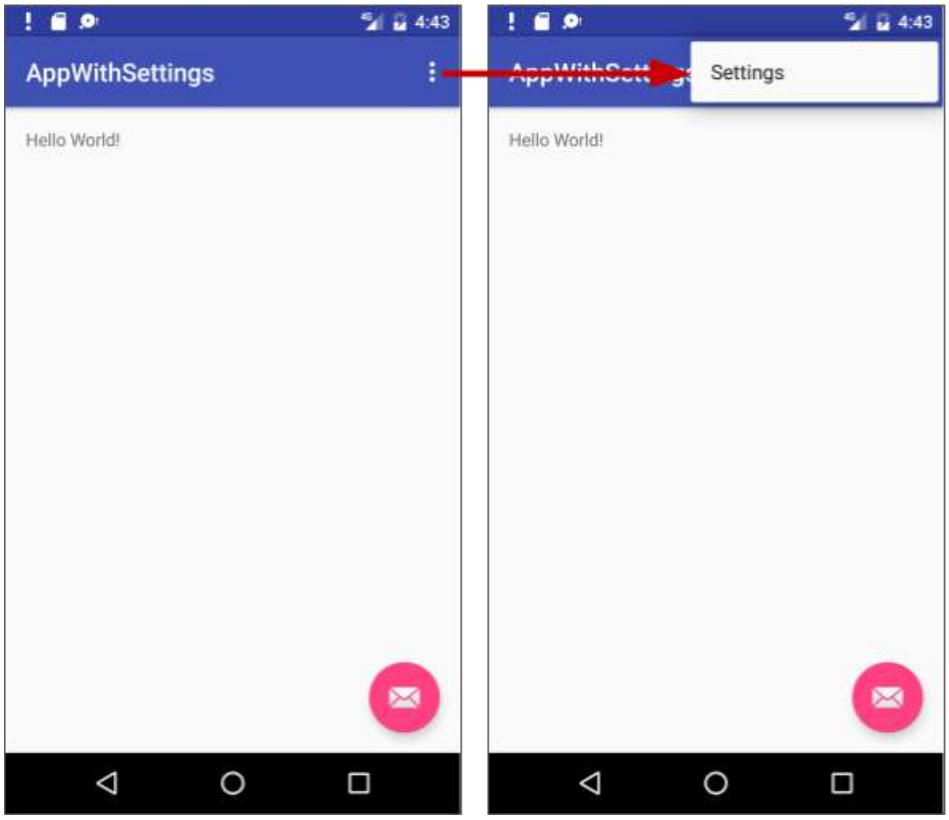
- Crear un proyecto nuevo basado en la plantilla Basic Activity (la cual suministra un menú de opciones).
- Agregar una configuración de un switch con atributos en un archivo XML.
- Añadir una actividad para la configuración y un fragmento para una configuración específica. Usará la versión PreferenceFragmentCompat de PreferenceFragment para mantener la compatibilidad con AppCompatActivity. También agregará la biblioteca android.support:preference-v7.
- Conectar el elemento **Settings** en las opciones del menu para la actividad de configuración.

1.1 Crear el proyecto y agregar el directorio xml y el archivo de recursos

1. En el Android Studio, cree un proyecto nuevo con los siguientes parámetros:

Atributo	Valor
Application Name	AppWithSettings
Company Name	android.example.com (or your own domain)
Phone and Tablet Minimum SDK	API15: Android 4.0.3 IceCreamSandwich
Use a Fragment?	Leave unchecked
Template	Basic Activity

2. Ejecute la aplicación, y toque el ícono de desborde en la barra de la aplicación para ver las opciones del menu, tal como se muestra en la figura siguiente. El único elemento en el menu de opciones es **Settings**.



3. Cree un nuevo directorio de recursos para contener el archivo XML que tendrá la configuración:

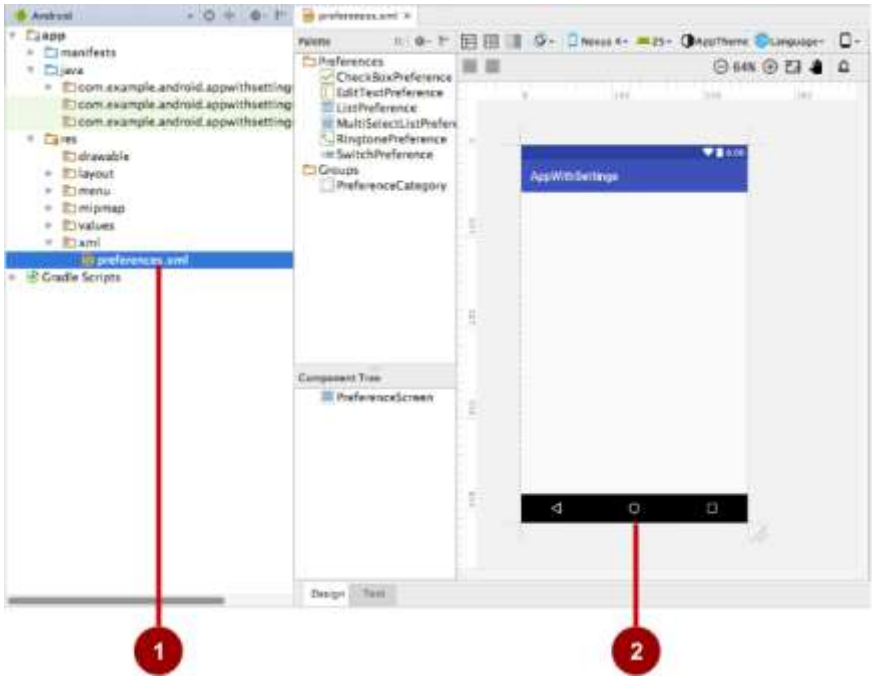
    a. Seleccione el directorio **res** en la vista Project: Android y seleccione **File > New > Android Resource Directory**. El diálogo New Resource Directory aparece.

b. En el menú Resource seleccione **xml**. El nombre del directorio automáticamente cambia a xml.

c. Haga clic en **OK**.

4. El directorio **xml** aparece en la vista Project: Android dentro del diretorio **res**. Seleccione el directorio **xml** y seleccione **File > New > XML resource file** (o haga clic derecho en el directorio **xml** y escoja **New > XML resource file**).

5. Ingrese el nombre del archive XML, **preferences**, en el campo File name y haga clic en **OK**. El archivo preferences.xml aparecerá dentro del directorio xml y aparece el editor de layout, como se muestra a continuación.



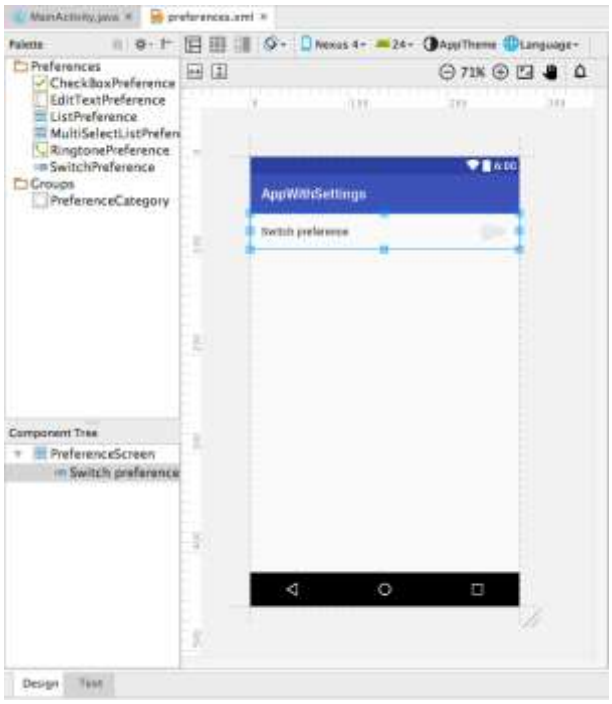
En la figura anterior:

1. El archive de preferences.xml dentro del directorio xml.

2. El editor de layout mostrando el contenido de preferences.xml.

1.2 Agregar las preferencias y atributos XML para la configuración

1. Arrastre un **SwitchPreference** desde la paleta en el lado izquierdo al tope del layout, como se muestra a continuación.



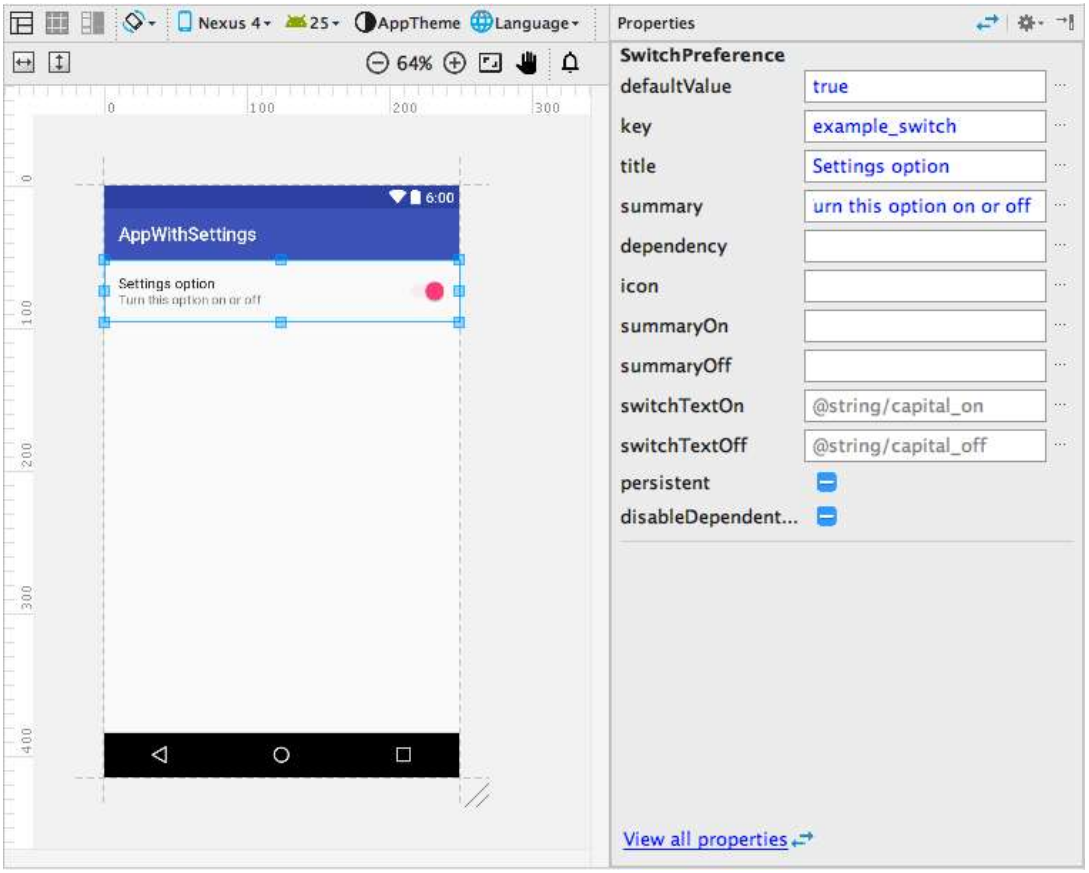
2. Cambie los valores en el panel Properties en ell lado derecho del editor de layout como se muestra (refierace a la figura siguiente):

    a. defaultValue: **true**

    b. key: **example_switch**

    c. title: **Settings option**

    d. summary: **Turn this option on or off**



3. Haga clic en la pestaña Text en la base del editor de layout para editar el código XML:

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
   <SwitchPreference
      android:defaultValue="true"
      android:title="Settings option"
      android:key="example_switch"
      android:summary="Turn this option on or off" />
</PreferenceScreen>
```

Los valores de Propiedades que ingresó representan los atributos XML:

- `android:defaultValue`: El valor predeterminado del ajuste cuando la aplicación comienza por vez primera.
- `android:title`: El título del ajuste. Para un SwitchPreference, el título aparece en la izquierda del switch.
- `android:key`: La clave a usar par almacenar el valor de ajuste. Cada ajuste tiene un correspondiente par clave-valor que el sistema usa para guarder el valor del ajuste en un archivo predeterminado SharedPreferences en la configuración de la aplicación.
- `android:summary`: El texto resumen aparece por debajo del ajuste.

4. Extraiga los recursos de string para los valores de los atributos `android:title` y `android:summary` en `@string/switch_title` y `@string/switch_summary`.

5. Cambie `<SwitchPreference` en el código a `<android.support.v7.preference.SwitchPreferenceCompat`:

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
   <android.support.v7.preference.SwitchPreferenceCompat
      ... />
</PreferenceScreen>
```

Para utilizar la versión `PreferenceFragmentCompat` de PreferenceFragment, también debe utilizar el android.support.versión v7 de SwitchPreference (`SwitchPreferenceCompat`).

La línea `SwitchPreferenceCompat` de arriba puede mostrar un icono de un bombillo amarillo con una advertencia, pero puede ignorarla.

6. Abra el archivo **styles.xml** y agregue la siguiente declaración `preferenceTheme` al `AppTheme`:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
   ...
   <item name="preferenceTheme">@style/PreferenceThemeOverlay</item>
</style>
```

Para poder usar la versión `PreferenceFragmentCompat` de PreferenceFragment, debe además declarer `preferenceTheme` con el estilo `PreferenceThemeOverlay` al tema de la app.

7. Abra el archive **build.gradle (Module: app)** y agregue lo siguiente a la sección de dependencias:

```
dependencies {
   ...
   compile 'com.android.support:preference-v7:25.0.1'
}
```

Lo anterior agrega la librería de soporte support:preference-v7 para poder usar la versión `PreferenceFragmentCompatversion` del `PreferenceFragment`.

1.3 Agregar una actividad para los ajustes y el fragment para una configuración específica

1. Para crear una actividad Settings que suministre la UI para los ajustes, agregue una Empty Activity a la aplicación:

    a. Seleccione **app** en el tope de la vista Project: Android.

    b. Seleccione **New > Activity > Empty Activity**.

c. Nombre la actividad como **SettingsActivity**.

d. Deseleccione la opción para generar un archivo layout (no necesitará ninguno).

e. Deje marcada la opción Backwards Compatibility (AppCompat).

f. El nombre del paquete ya debería estar puesto como **com.example.android.*projectname***y el Target Source Set debería estar colocado en **main**. Si no es así, realice estas selecciones en los menús drop-down.

g. Haga clic en **Finish**.

El resultado es la siguiente definición de la clase SettingsActivity:

```
public class SettingsActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

2. Agregue un fragment en blanco para un grupo de ajustes similares (*sin* un layout, métodos de factoría o los callback de la interface) a la aplicación, para cambiarlos a la pantalla de la actividad Settings cuando se necesite:

    a. Seleccione **app** en el tope de la vista Project: Android, nuevamente.

    b. Seleccione **New > Fragment > Fragment (Blank)**.

    c. Nombre al fragment **SettingsFragment**.

    d. Deseleccione la opción para generar un archive layout (no necesitará ninguno).

    e. Deseleccione la opción para incluir los métodos de factoría del fragment.

    f. Deseleccione la opción para incluir los callback de la interface.

    g. Haga clic en **Finish**.

    El resultado será la definición de la clase SettingsFragment:

```
public class SettingsFragment extends Fragment {

    public SettingsFragment() {
        // Required empty public constructor
    }

    @Override
```

```
        public View onCreateView(LayoutInflater inflater,
                                 ViewGroup container, Bundle savedInstanceState) {
            TextView textView = new TextView(getActivity());
            textView.setText(R.string.hello_blank_fragment);
            return textView;
        }
    }
```

3. Cambie la definición de la clase SettingsFragment para que extienda a `PreferenceFragmentCompat`:

```
public class SettingsFragment extends PreferenceFragmentCompat {
...
}
```

Se usa una subclase especializada de Fragment para visualizar una lista de ajustes. La mejor práctica es utilizar una Activity regular que contenga un PreferenceFragment que muestre la configuración de la aplicación. Fragments como PreferenceFragment proporcionan una arquitectura más flexible para la aplicación, en comparación con el uso de actividades solas. Un fragment es como una sección modular de una actividad: tiene su propio ciclo de vida y recibe sus propios eventos de entrada, y puede agregar o quitar un fragment mientras se ejecuta la actividad.

Utilice la versión PreferenceFragmentCompat de PreferenceFragment con una actividad que extienda a AppCompatActivity. Para ampliar el fragment, puede que tenga que añadir la siguiente declaración de importación:

```
import android.support.v7.preference.PreferenceFragmentCompat;
```

4. Reemplace totalmente el método `onCreateView()` en el fragment con este metodo onCreate():

```
    @Override
    public void onCreatePreferences(Bundle savedInstanceState,
        String rootKey) {
    }
```

La razón por la que reemplaza `onCreateView()` con `onCreatePreferences()` en SettingsFragment es porque se agregará este fragmento a la actividad de configuración existente para mostrar las preferencias, en lugar de mostrar una pantalla de fragmentos separada. Añadirlo a la actividad existente facilita la adición o eliminación de un fragmento mientras la actividad se está ejecutando. El fragmento de preferencia se enraíza en la PreferenceScreen utilizando rootKey.

También puede quitar con seguridad el constructor vacío del fragmento, ya que el fragmento no se muestra por sí solo:

```
public SettingsFragment() {
        // Required empty public constructor
      }
```

5. Al final del método `onCreatePreferences()` en SettingsFragment, necesita asociar con este fragmento el recurso **preferences.xml** de settings que acaba de crear. Agregue una llamada a setPreferencesFromResource() pasando el id del archivo XML (`R.xml.preferences`) y el `rootKey` para identificar la raíz de la preferencia en `PreferenceScreen`:

```
setPreferencesFromResource(R.xml.preferences, rootKey);
```

El método `onCreatePreferences()` debe lucir como se muestra:

```
@Override
public void onCreatePreferences(Bundle savedInstanceState,
                                String rootKey) {
    setPreferencesFromResource(R.xml.preferences, rootKey);
}
```

6. Agregue el siguiente código al final del método `onCreate()` de SettingsActivity para que el fragment sea mostrado en el contenido principal:

```
getSupportFragmentManager().beginTransaction()
                .replace(android.R.id.content, new SettingsFragment())
                .commit();
```

El código anterior en un patron típico usado para agregar un fragment a una actividad para que el fragment aparezca en el contenido principal de la actividad. Use:

- `getFragmentManager()` si la clase extiende Activity y el fragment extiende `PreferenceFragment`.
- `getSupportFragmentManager()` si la clase extiende a AppCompatActivity y el fragment extendie `PreferenceFragmentCompat`.

Todo el método `onCreate()` en SettingsActivity debe lucir como a continuación:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getSupportFragmentManager().beginTransaction()
                .replace(android.R.id.content, new SettingsFragment())
                .commit();
```

```
}
```

1.4 Conectar el menu de Settings a la actividad de configuración

Use un `intent` para iniciar la SettingsActivity desde la MainActivity.

1. Encuentre el blque `if` en el método `onOptionsItemSelected()` de la MainActivity, el cual maneja el toque de **Settings** en el menú de opciones:

```
if (id == R.id.action_settings) {
   return true;
}
```

2. Agregue un `intent` al bloque `if` para iniciar la SettingsActivity:

```
if (id == R.id.action_settings) {
   Intent intent = new Intent(this, SettingsActivity.class);
   startActivity(intent);
   return true;
}
```

3. Agregar el botón-**UP** de navegación a la `SettingsActivity` editando su declaración en el archivo **AndroidManifest.xml** para definir a `MainActivity` como el padre de la actividad.

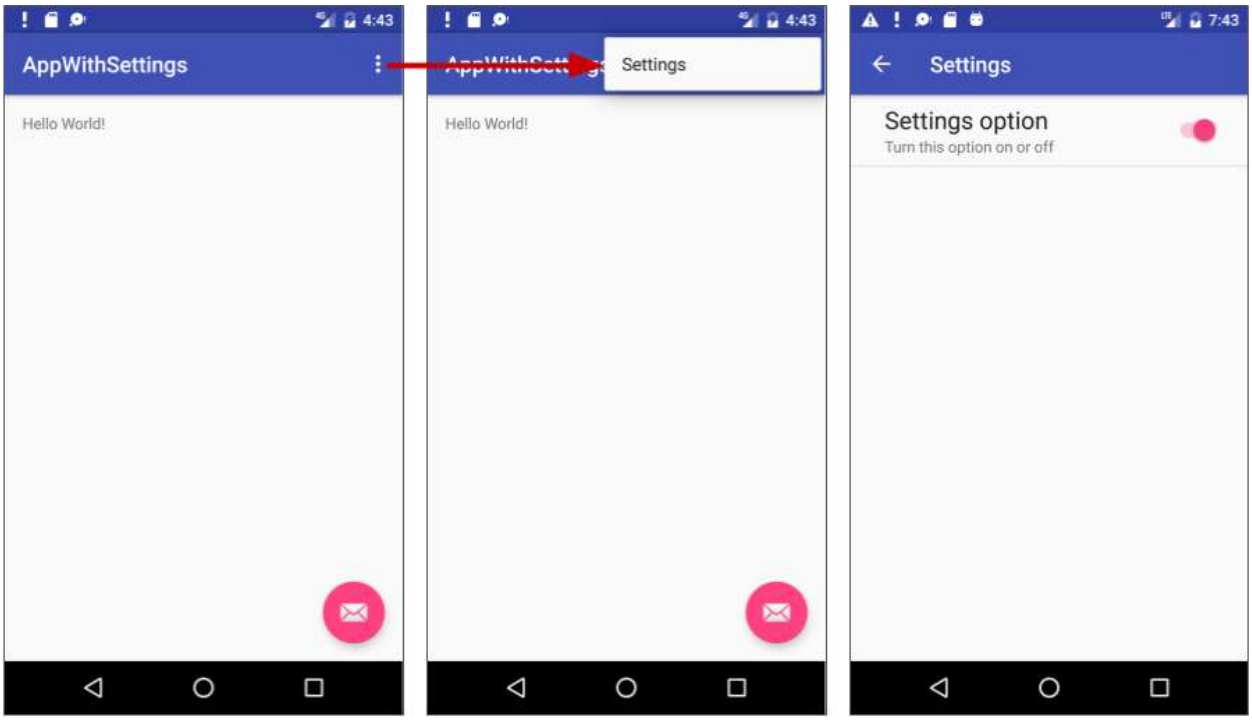
    a. Encuentre la declaración de `SettingsActivity` en el AndroidManifest.xml:

```
<activity android:name=".SettingsActivity"></activity>
```

    b. Cambie la declaración dela forma siguiente:

```
<activity android:name=".SettingsActivity"
    android:label="Settings"
    android:parentActivityName=".MainActivity">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity"/>
</activity>
```

4. Ejecute la aplicación. Toque el icono de desbordamiento para el menú de opciones (como se muestra en el lado izquierdo de la figura de abajo), y toque **Settings** para ver la actividad de configuración (como se muestra en el centro de la figura de abajo). Toque el botón **Up** en la barra de aplicaciones de la actividad de configuración, que se muestra en la parte derecha de la figura de abajo, para volver a la actividad principal.



1.5 Guardar los valores predeterminados en las shared preferences

Aunque el valor predeterminado para la configuración del switch ya se ha establecido en el atributo `android:defaultValueattribute` (en el paso 1.2 de esta tarea), la aplicación debe guardar el valor predeterminado en el archivo SharedPreferences para cada configuración cuando el usuario abra la aplicación por primera vez. Siga estos pasos para establecer el valor predeterminado para el toggle switch:

1. En MainActivity, agregue lo siguiente al final del método onCreate():

```
protected void onCreate(Bundle savedInstanceState) {
   ...
   PreferenceManager.setDefaultValues(this, R.xml.preferences, false);
}
```

El código superior asegura que las configuraciones sean apropiadamente inicializadas con sus valores predeterminados. El método `setDefaultValues()` toma tres argumentos:

- El context de la aplicación, tal como `this`.
- El ID del recurso ID (`preferences`) del archive XML de recursos con uno o más ajustes.
- Un boolean que indica si los valores predeterminados deben establecerse más de una vez. Si es `false`, el sistema fija los valores predeterminados sólo si nunca se ha llamado este método en el pasado. Siempre y cuando configure este tercer argumento como `false`, puede llamar con seguridad a este método cada vez que se inicie la actividad principal sin sustituir los valores de configuración guardados por el usuario.

Sin embargo, si lo coloca en `true`, el método sustituirá cualquier valor anterior con los valores predeterminados.

1.6 Leer los valores de configuración modificados desde las shared preferences

Cuando se inicia la aplicación, el método `onCreate()` de la MainActivity puede leer los valores de configuración que se han cambiado y utilizar los valores modificados en lugar de los valores predeterminados.

Cada ajuste se identifica mediante un par clave-valor. El sistema Android utiliza este par clave-valor cuando guarda o recupera la configuración de un archivo SharedPreferences para la aplicación. Cuando el usuario modifica una opción, el sistema actualiza el valor correspondiente en el archivo SharedPreferences. Para usar el valor de la configuración, la aplicación puede usar la clave para obtener la configuración desde archivo SharedPreferences.

Siga los siguientes pasos para agregar el código:

1. Antes de agregar el código para leer el valor de configuración, cree una variable estática de tipo string en SettingsActivity para mantener la clave del valor:

```
public class SettingsActivity extends AppCompatActivity {
   public static final String KEY_PREF_EXAMPLE_SWITCH = "example_switch";
    ...
}
```

2. En el método `onCreate()` de la MainActivity agregue lo siguiente al final del método:

```
protected void onCreate(Bundle savedInstanceState) {
...
SharedPreferences sharedPref =
                PreferenceManager.getDefaultSharedPreferences(this);
Boolean switchPref = sharedPref.getBoolean
                (SettingsActivity.KEY_PREF_EXAMPLE_SWITCH, false);
}
```

   El anterior fragmento de código usa:

   - `PreferenceManager.getDefaultSharedPreferences(this)` para obtener las configuración como un objeto SharedPreferences (`sharedPref`).
   - `getBoolean()` para obtener el valor Boolean de la configuración que usa la clave definida en SettingsActivity (`KEY_PREF_EXAMPLE_SWITCH`) y asignada a `switchPref`. SI no hay un valor para la clave, el método `getBoolean()` coloca el valor (`switchPref`) en `false`. Para otros valores tales como strings, enteros, o números de punto flotante, puede usar los métodos `getString()`, `getInt()`, o `getFloat()` respectivamente..

3. Agregur un método `Toast.makeText()` en `onCreate()` para que sea mostrado el valor del ajuste `switchPref` en un toast:

```
Toast.makeText(this, switchPref.toString(), Toast.LENGTH_SHORT).show();
```

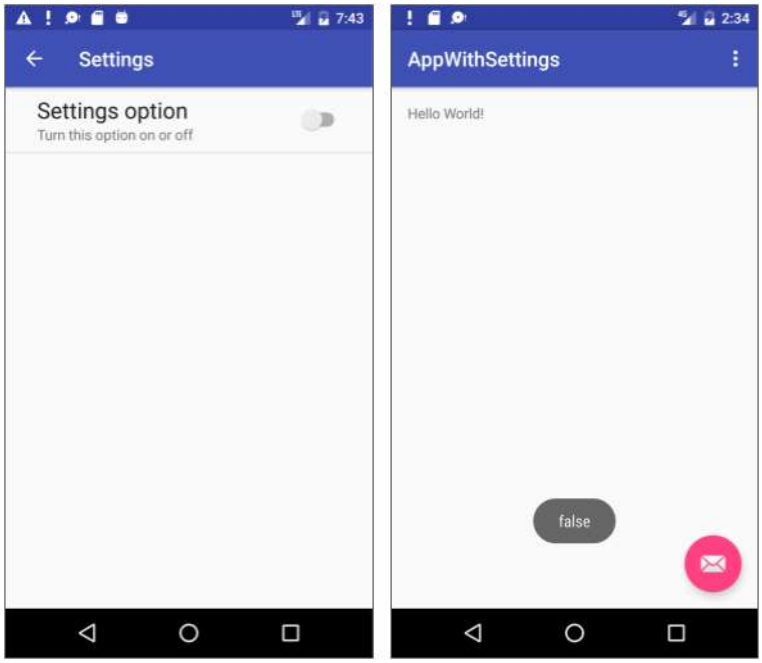
4. Ejecute la application y siga los siguientes pasos:

    a. Toque **Settings** para ver la actividad de configuración.

    b. Toque el switchpara cambiarlo de on a off, como se muestra en el lado izquierdo de la figura inferior.

    c. Toque el botón **Up** en la actividad de configuración para retornar a la actividad principal. El mensaje toast debería aparecer en la ventana principal con el valor configurado, tal como es mostrado en el lado izquierdo de la figura de abajo.

    d. Repita estos pasos para ver el cambio del mensaje toast a medida que cambia el ajuste.



Siempre que MainActivity se inicie o reinicie, el método `onCreate()` debe leer los valores de configuración para poder utilizarlos en la aplicación. El método `Toast.makeText()` debería ser reemplazado por un método que inicializa la configuración.

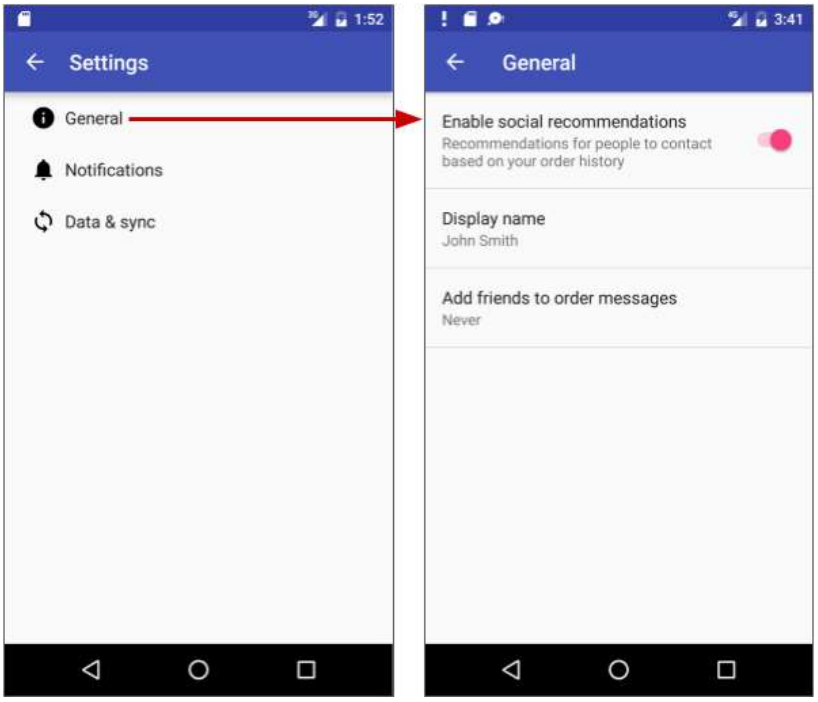
Ahora tiene una actividad de configuración de trabajo en la aplicación.

Task 2: Usando la plantilla Settings Activity

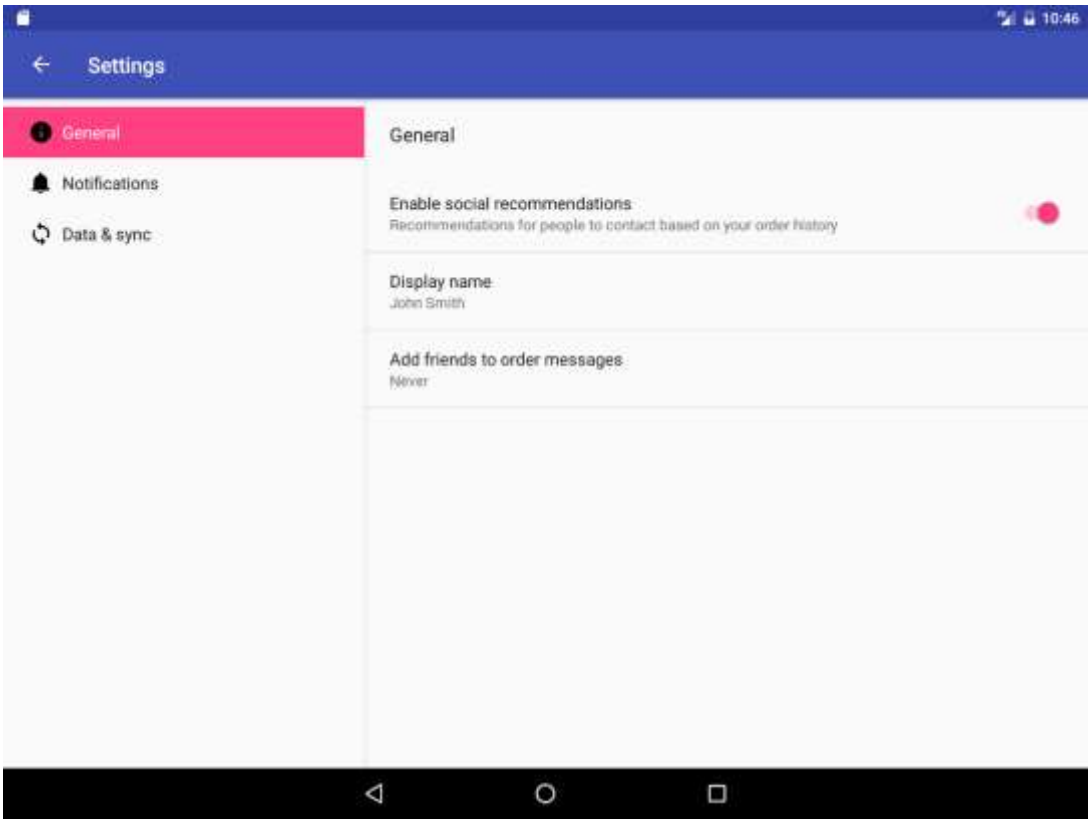
Si se necesita crear varias subpantallas de configuración y se desea aprovechar las pantallas del tamaño de una tableta y mantener la compatibilidad con versiones anteriores de Android para tabletas, Android Studio proporciona un acceso directo: la plantilla Settings Activity.

En la tarea anterior aprendimos a usar una actividad de configuración vacía y un fragmento en blanco para añadir una configuración a una aplicación. La tarea 2 le mostrará ahora cómo utilizar la plantilla Settings Activity suministrada con Android Studio para:

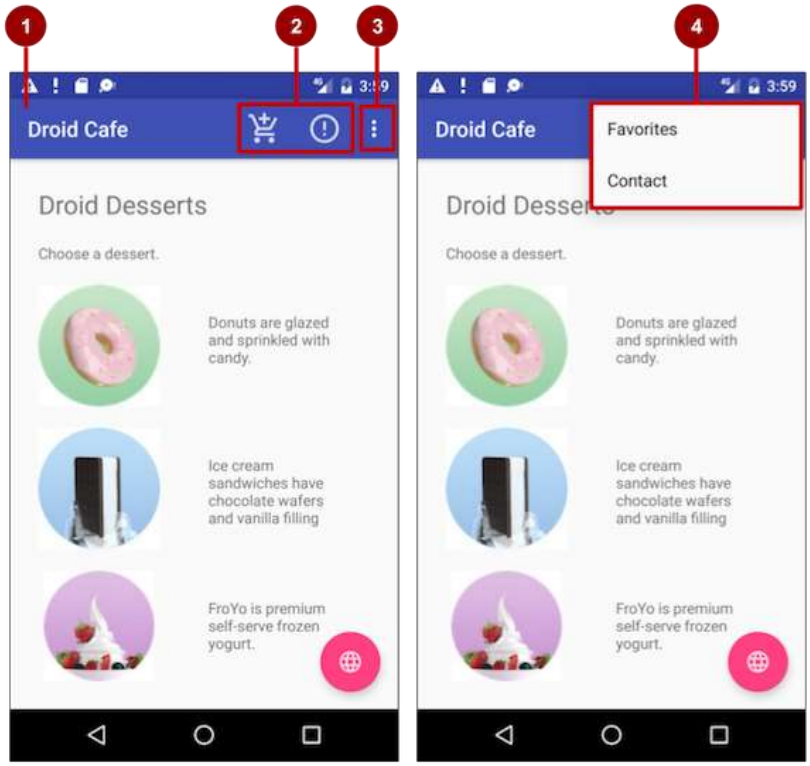
- Dividir multiples ajustes en grupos.
- Personalizar las configuraciones y sus valores.
- Mostrar una pantalla Settings con un encabezado de enlace para cada grupo de ajustes, tales como General para ajustes generales, tal como se muestra en la figura siguiente.



- Mostrar un pantalla layout maestro/detale con un encabezado de enlace por cada grupo en la izquierda (maestro), y el grupo de ajustes en la derecha (detalle), como se muestra a continuación.



En una práctica anterior se creó una aplicación llamada Droid Cafe usando la plantilla Basic Activity, que proporciona un menú de opciones en la app bar como se muestra a continuación.



En la figura anterior:

1. **App bar**.
2. **Icono de acción del menú de opciones**.
3. **Botón de desborde**.
4. **Opciones del menú de desborde**.

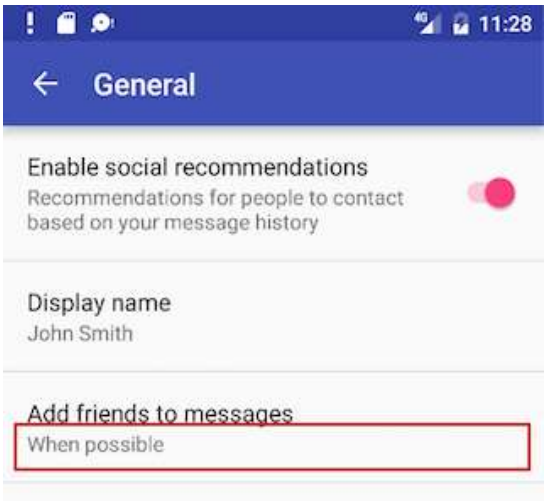
Proyecto de Android Studio: Para comenzar el Proyecto desde donde lo dejó en la práctica anterior, lo puede descargar de M7 desde la carpeta "Archivos/Aplicaciones de Arranque/**DroidCafe.rar**". Luego descomprimalo y lo tiene listo para seguir.

2.1 Explorar la plantilla Settings Activity

Para incluir la plantilla Settings Activity en su proyecto de aplicación en Android Studio, siga los siguientes pasos:

1. Copie la carpeta del proyecto **DroidCafe** y renombrela como **DroidCafeWithSettings**, y haga las refactorizaciones (Vea el Appendice para las instrucciones de como copiar el proyecto). Ejecute la aplicación para asegurarse que funciona apropiadamente.
2. Seleccione **app** en el tope de la vista Project: Android, y seleccione **New > Activity > Settings Activity**.
3. En el díalogo que aparece, acepte el Activity Name (**SettingsActivity** es el nombre sugerido) y el Title (**Settings**).
4. Haga clic en lost res puntos al final del campo Hierarchical Parent y seleccione **MainActivity** como la actividad padre, de manera que el botón **Up** en la Settings Activity devuelva al usuario a la MainActivity. Seleccionando la actividad padre automaticamente actualize el archivo AndroidManifest.xml para que soporte la nvegación del botón **Up**.
5. Haga clic en **Finish**.

La plantilla Settings Activity no sólo proporciona layouts para pantallas del tamaño de un smartphone y de una tableta, sino que también ofrece la función de escuchar un cambio de configuración y cambiar el resumen para reflejar el cambio de configuración. Por ejemplo, si cambia la opción " Add friends to messages " (las opciones son **Always**, **When possible**, o **Never**), la opción que elija aparecerá en el resumen debajo de la opción:



En general, no es necesario cambiar el código de la plantilla Settings Activity para personalizar la activity para los austes que desea en la aplicación. Se pueden personalizar los títulos de las opciones, los resúmenes, los posibles valores y los valores predeterminados sin cambiar el código de plantilla, e incluso agregar más opciones a los grupos que se proporcionan.

Utilice el código de la plantilla Settings Activity tal cual está. Para que funcione para la aplicación, agregue código a la MainActivity para establecer los valores de configuración predeterminados, y para *leer* y *usar* los valores de configuración, como se mostrará más adelante en esta tarea.

La plantilla Settings Activity crea lo siguiente para usted:

- Archivos XML en el directorio **res > xml**, al cual puede agregar o personalizar para los ajustes que implementará.
    - pref_data_sync.xml: PreferenceScreen layout para ajustes "Data & sync".
    - pref_general.xml: PreferenceScreen layout para ajustes "General".
    - pref_headers.xml: Layout de encabezado de Settings en la pantalla principal
    - pref_notification.xml: PreferenceScreen layout para ajustes "Notifications".

Los layouts XML anteriores utilizan varias subclases de la clase Preference en lugar de objetos View , y las subclases directas proporcionan contenedores para layouts con múltiples configuraciones. Por ejemplo, PreferenceScreen representa una Preference de nivel superior que es la raíz de una jerarquía de Preferencias. Los archivos anteriores utilizan PreferenceScreen en la parte superior de cada pantalla de configuración. Otras subclases de Preference para ajustes proporcionan la UI apropiada para que los usuarios cambien el ajuste. Por ejemplo:

- CheckBoxPreference: Un checkbox para un ajuste que pueda ser habilitado o deshabilitado.
- ListPreference: Un díalogo con una lista de radio buttons.
- SwitchPreference: Una opción de dos-estados intercambiables (tal como on/off o true/false).

- EditTextPreference: Un dialogocon un widget EditText.
- RingtonePreference: Un díalogo con ringtones del dispositivo.

Sugerencia: Se pueden editar estos archivos XML para cambiar los ajustes predeterminados por aquellos que se necesiten en la aplicación.

- Recursos de string en el archivo strings.xml file en el directorio **res > values**, los cuales se pueden personalizar por los ajustes que se deseen.

  Todas los strings utilizados en la Settings Activity, como son los títulos de las configuraciones, los arreglos de string para las listas y las descripciones de los ajustes, son definidos como recursos de string al final de este archivo. Estos están marcados como comentarios así: `<!-- Strings related to Settings -->` y `<!-- Example General settings -->`.

  Sugerencia: Se pueden editar estos strings para personalizar los ajustes que necesite para su aplicación.

- SettingsActivity en el directorio **java > com.example.android.*projectname***, el cual puede dejarlo tal como está.

  Esta es la activity que muestra los ajustes. `SettingsActivity` extiende a `AppCompatPreferenceActivity` para mantener compatibilidad con versions anteriores de Android.

- AppCompatPreferenceActivity en el directorio **java>com.example.android.*projectname***, el cual puede dejarlo tal como está.

  Esta activity es una clase helper que SettingsActivity usa para mantener compatibilidad con versions previas de Android.

2.2 Agregar el elemento de menú Settings y conectarse a la activity

Tal como se ha visto en prácticas anteriores, se puede editar el archive **menu_main.xml** en el directorio **res > menu** de las opciones de menú para agregar o remover elementos de menú.

1. Editar el archivo **menu_main.xml** para agregar otro elemento de menú llamado Settings con un nuevo recurso con id `action_settings`:

```
<item
    android:id="@+id/action_settings"
    android:orderInCategory="50"
    android:title="Settings"
    app:showAsAction="never" />
```

   Especifique "`never`" para el atributo `app:showAsAction` de manera que **Settings** aparezca solo en el menu de desborde opciones y no en la barra de la aplicación, debido a que no será usado de forma regular.

Especifique "50" para el atributo `android:orderInCategory` de manera que **Settings** aparezca por debajo de **Favorites** (colocado en "40") pero por sobre **Contact** (colocado en "100").

2. Extraiga el recurso de string para "`Settings`" en el atributo `android:title` al recurso de string de nombre settings.

3. En la MainActivity, encuentre el bloque `switch-case` en el método `onOptionsItemSelected()` el cual maneja el toque sobre elementos del menú de opciones:

```
public boolean onOptionsItemSelected(MenuItem item) {
   switch (item.getItemId()) {
      case R.id.action_order:
         displayToast(getString(R.string.action_order_message));
         return true;
      case R.id.action_status:
         displayToast(getString(R.string.action_status_message));
         return true;
      case R.id.action_favorites:
                displayToast(getString(R.string.action_favorites_message));
         return true;
      case R.id.action_contact:
         displayToast(getString(R.string.action_contact_message));
         return true;
         }

   return super.onOptionsItemSelected(item);
}
```

4. Use un `intent` para iniciar la SettingsActivity desde la MainActivity. Agregue el `intent` al final del bloque `switch case`:

```
      ...
      case R.id.action_settings:
         Intent intent = new Intent(this, SettingsActivity.class);
```

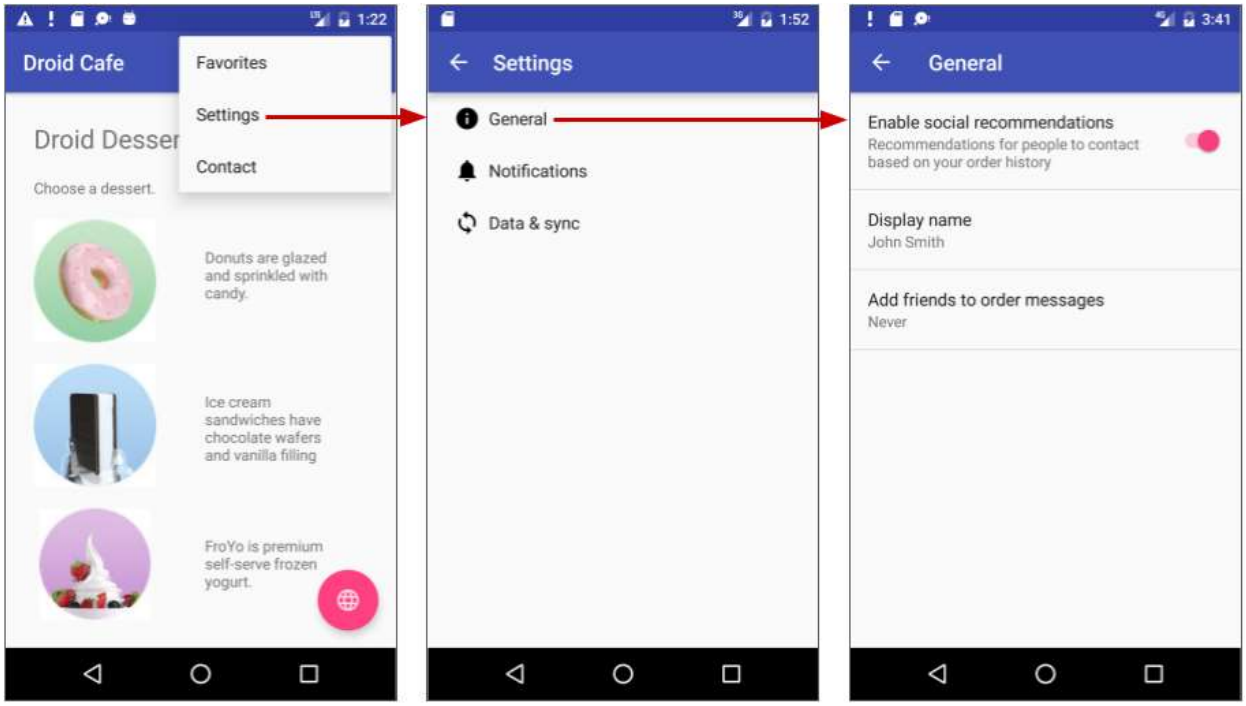
```
            startActivity(intent);
            return true;
            }

    return super.onOptionsItemSelected(item);
}
```

5. Ejecute la aplicaciónusando un teléfono o un emulador de teléfono para que vea como la plantilla Settings Activity maneja el tamaño de la pantalla, y siga los siguientes pasos:

    a. Toque el ícono de desborde del menu de opciones, y toque **Settings** para ver la activity de ajustes, tal como es mostrado en el lado izquierdo de la figura siguiente.

    b. Toque cada uno de los encabezados de ajuste (**General**, **Notifications**, y **Data & sync**), tal como se muestra en el centro de la figura de abajo, para ver el grupo de ajustes en cada pantalla hija de la pantalla Settings, mostrada en el lado derecho de la la figura.

    c. Toque el botón **Up** en la activity de ajustes para regresar a la activity principal.



2.3 Personalizar las configuraciones suministradas por la plantilla

Para personalizar la configuración proporcionada por la plantilla Settings Activity, edite el recurso de string y el arreglo de string en el archivo strings.xml y los atributos de layout por cada ajuste en los archivos del directorio **xml**. En este paso cambiará los ajustes de "Data & sync".

1. Abra el archivo **strings.xml** en el directorio **res > values** y desplazece sobre el contenido hasta el comentario `<!-- Example settings for Data & Sync -->`

```
<!-- Example settings for Data & Sync -->
    <string name="pref_header_data_sync">Data &amp; sync</string>

    <string name="pref_title_sync_frequency">Sync frequency</string>
    <string-array name="pref_sync_frequency_titles">
        <item>15 minutes</item>
        <item>30 minutes</item>
        <item>1 hour</item>
        <item>3 hours</item>
        <item>6 hours</item>
        <item>Never</item>
    </string-array>
    <string-array name="pref_sync_frequency_values">
        <item>15</item>
        <item>30</item>
        <item>60</item>
        <item>180</item>
        <item>360</item>
        <item>-1</item>
    </string-array>
          ...
```

2. Edite el recurso de string `pref_header_data_sync` el cual está colocado como `Data & sync` (el simbolo `&` es el código HTML para un ampersand). Cambie ese valor por **Account** (sin comillas).

3. Refactorize el nombre del recurso siguiendo los siguientes pasos (la aplicación continuará trabajando sin refactorizar los nombres, pero al hacerlo hace que el código sea más fácil de comprender):

    a. Control-clic (o clic-derecho) en el recurso de nombre **pref_header_data_sync** y seleccione **Refactor > Rename**.

    b. Cambie el nombre a **pref_header_account**, haga clic en la opción “search in comments and strings”, y haga clic en **Refactor**.

4. Edite el recurso de string `pref_title_sync_frequency` (el cual configure la frecuencia de Sync) por **Market**.

5. **Refactor > Rename** el nombre del recurso por **pref_title_account** como se hizo previamente.

6. **Refactor > Rename** el nombre del recurso del arreglo de strings `pref_sync_frequency_titles` por **pref_market_titles**.

7. Cambie cada valor en el arreglo de strings `pref_market_titles` (15 minutes, 30 minutes, 1 hour, etc.) para que sean los títulos de mercados, tales como **United States**, **Canada**, etc., más que frecuencias:

```
<string-array name="pref_market_titles">
```

```
        <item>United States</item>
        <item>Canada</item>
        <item>United Kingdom</item>
        <item>India</item>
        <item>Japan</item>
        <item>Other</item>
</string-array>
```

8. **Refactor > Rename** el nombre del recurso del arreglo de strings `pref_sync_frequency_values` por **pref_market_values**.

9. Cambie cada valor en el arreglo de strings `pref_market_values` (15, 30, 60, etc.) para que sean los valores de los mercados: abreviaciones tales como **US**, **CA**, etc.:

```
<string-array name="pref_market_values">
        <item>US</item>
        <item>CA</item>
        <item>UK</item>
        <item>IN</item>
        <item>JA</item>
        <item>-1</item>
    </string-array>
```

10. Descienda hasta el recurso de string `pref_title_system_sync_settings` y edite el recurso (el cual está colocado como System sync settings) por **Account settings**.

11. **Refactor > Rename** el nombre del recurso del arreglo de strings `pref_title_system_sync_settings` por **pref_title_account_settings**.

12. Abra el archivo **pref_data_sync.xml**. La `ListPreference` en este layout define los ajustes que han cambiado. Note que los recursos de string para los atributos `android:entries`, `android:entryValues` y `android:title` muestran los valores cambiados por los que suministró en los pasos anteriores:

```
<ListPreference
        android:defaultValue="180"
        android:entries="@array/pref_market_titles"
        android:entryValues="@array/pref_market_values"
        android:key="sync_frequency"
        android:negativeButtonText="@null"
        android:positiveButtonText="@null"
        android:title="@string/pref_title_account" />
```

13. Cambie el atributo `android:defaultValue`:

```
android:defaultValue="US"
```

Debido a que la clave para la preferencia de ajuste ("`sync_frequency`") está “cableada” en otra parte del código Java, no cambie el atributo `android:key`; utilice "`sync_frequency`" como clave de este ajuste en este ejemplo. Si se está personalizando a fondo la configuración para una aplicación del mundo real, debe pasar tiempo cambiando las claves “cableadas” en todo el código.

Nota: ¿Por qué no usar un recurso de string para la clave? Debido a que los recursos de string pueden adaptarse para diferentes idiomas usando archivos XML en varios idiomas, y el string de la clave puede ser traducida inadvertidamente junto con otros strings, lo que causaría que la aplicación se colgara.

2.4 Agregar el código para colocar los valores predeterminados de los ajustes

Encuentre el método `onCreate()` en la MainActivity, y agregue las siguientes sentencias `PreferenceManager.setDefaultValues` al final del método:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   ...
   PreferenceManager.setDefaultValues(this, R.xml.pref_general, false);
   PreferenceManager.setDefaultValues(this, R.xml.pref_notification, false);
   PreferenceManager.setDefaultValues(this, R.xml.pref_data_sync, false);
}
```

Los valores predeterminados ya están especificados en el archivo XML con el atributo `android:defaultValue`, pero las instrucciones anteriores garantizan que el archivo Shared Preferences se inicialice correctamente con los valores predeterminados. El método `setDefaultValues()` toma tres argumentos:

- El context de la aplicación, tal como `this`.
- El ID del recurso para el archivo XML del layout de ajustes, el cual incluye los valores predeterminados por el atributo `android:defaultValue`.
- Un boolean que indica si los valores predeterminados deben establecerse más de una vez. Si es `false`, el sistema fija los valores predeterminados sólo si nunca se ha llamado este método en el pasado. Siempre y cuando configure este tercer argumento como `false`, puede llamar con seguridad a este método cada vez que se inicie la actividad principal sin sustituir los valores de configuración guardados por el usuario. Sin embargo, si lo coloca en `true`, el método sustituirá cualquier valor anterior con los valores predeterminados.

2.5 Agregar el código para leer los valores de configuración

1. Añada el siguiente código al final del método `onCreate()` de MainActivity. Puede agregarlo inmediatamente después del código que agregó en el paso anterior para establecer los valores predeterminados de la configuración:

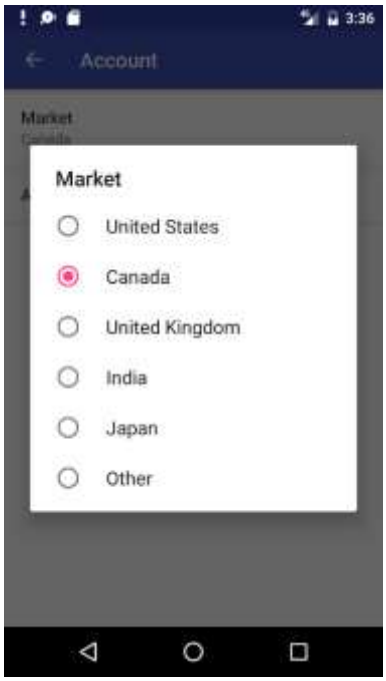
```
...
```

```
SharedPreferences sharedPref =
            PreferenceManager.getDefaultSharedPreferences(this);
String marketPref = sharedPref.getString("sync_frequency", "-1");
Toast.makeText(this, marketPref, Toast.LENGTH_SHORT).show();
}
```

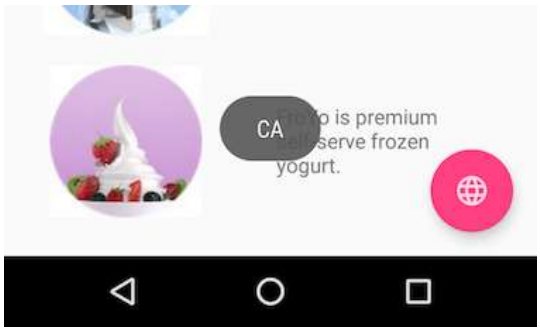
Como ya aprendió en la tarea anterior, se usa `PreferenceManager.getDefaultSharedPreferences(this)` para obtener la configuración como un objeto SharedPreferences (`sharedPref`). A continuación, se utiliza `getString()` para obtener el valor string del parámetro que utiliza la clave (`sync_frequency`) y asignarlo a `marketPref`. Si no hay valor para la clave, el método `getString()` establece el valor de configuración de `marketPref` en -1, el cual es el valor de `Other` en el arreglo `pref_market_values`.

2. Ejecute la aplicación, nuevamente usando un teléfono o el emulador. Cuando la pantalla principal de la aplicación aparezca, verá un mensaje toast en la base de la pantalla. La primera vez que se ejecute la app, debe ver "`-1`" mostrado en el toast debido a que aún no ha cambiado el ajuste.

3. Toque **Settings** en el menu de opciones y toque **Account** la pantalla de Settings. Seleccione **Canada** bajo "Market" como se muestra:

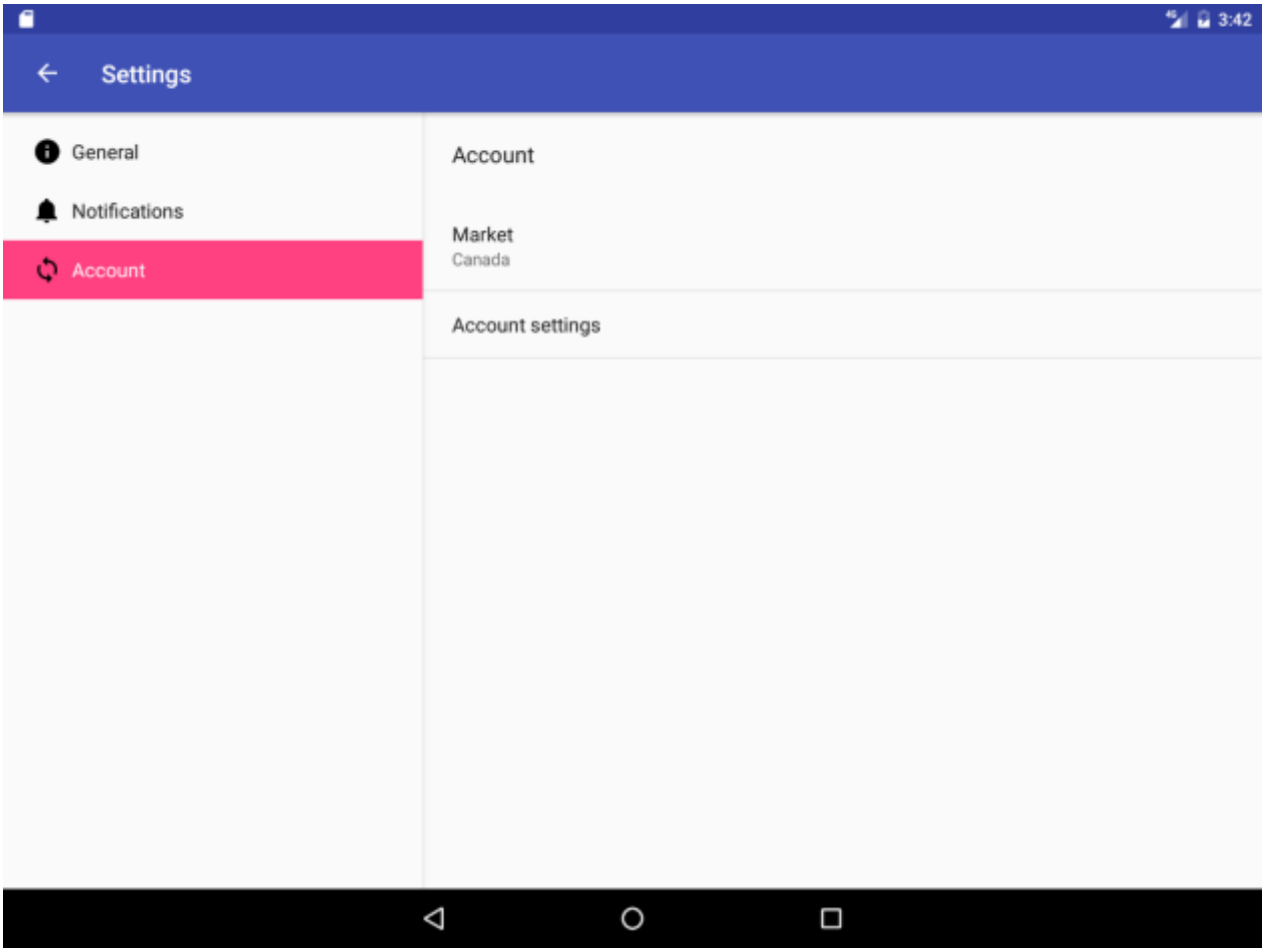


4. Toque el botón **Up** en la app bar para regresar a la pantalla Settingsy toquelo otra vez para retornar a la pantalla principal. Debería ver un mensaje toats con "CA" (por Canada):



Ya ha integrado exitósamente la Settings Activity con la aplicación Droid Café!!.

5. Ahora ejecute la aplicación en una tablet o en un emulador de tablet. Ya que la tablet tiene una pantalla físicamente más grande, el runtime de Android aprovechará el espacio extra. En una tablet, las configuraciones y detalles son mostrados en la misma pantalla hacienda más fácil para los usuarios administrar sus ajustes.



Resumen

En esta práctica ha aprendido a:

- Agregar un ajuste con un switch commutable (`SwitchPreference`) con atributos en un archivo XML de preferencias y configurado sus atributos:
    - `android:defaultValue`: El valor predeterminado del ajuste.
    - `android:title`: El título del ajuste.
    - `android:key`: La clave del ajuste.
    - `android:summary`: El resumen del ajuste.
- Agregar una actividad de configuración para ver los ajustes y un fragment que extiende `PreferenceFragment` por cada ajuste específico.
    - Usar `getFragmentManager()` para agregar el fragment a la actividad de ajustes
    - Usar `addPreferencesFromResource()` en cada fragment para cargar los archivos de preferencias apropiado para ese fragment.
- Usar un `intent` para conectar los elementos **Settings** en las opciones de menú para la actividad de ajustes.
- Colocar los valores predeterminados de los ajustes usando `PreferenceManager.setDefaultValues()`.
- Leer los valores de configuración desde SharedPreferences usando `PreferenceManager.getDefaultSharedPreferences()` y obteniendo cada valor de ajuste usando .getString, .getBoolean, etc.

7.3 Base de Datos SQLite

Introducción a SQL

A continuación se presentará un breve refrescamiento de lo que son base de datos SQL y puede ser usado como una referencia rápida solamente, ya que existen varios cursos en línea muy completos.

Bases de datos SQL

- Almacenan datos en tablas de filas y columnas.
- La intersección de una fila y una columna se denomina campo.
- Los campos contienen datos, referencias a otros campos o referencias a otras tablas.
- Las filas se identifican mediante IDs únicos.
- Las columnas se identifican por nombres que son únicos por tabla.

Piense en ello como una hoja de cálculo con filas, columnas y celdas, donde las celdas pueden contener datos, referencias a otras celdas y enlaces a otras hojas.

SQLite

SQLite es una biblioteca de software que implementa el motor de base de datos SQL que es:

- autónoma (no requiere ningún otro componente)
- sin servidor (no requiere servidor backend)
- cero configuración (no necesita ser configurado para su aplicación)
- transaccional (los cambios dentro de una sola transacción en SQLite ocurren completamente o no ocurren en absoluto)

SQLite es el motor de bases de datos más utilizado del mundo. El código fuente de SQLite es de dominio público.

Para detalles de la base de datos SQLite, vea el SQLite website.

Tabla de Ejemplo

SQLite almacena los datos en tablas.

Asumimos lo siguiente:

- Una base de datos DATABASE_NAME
- Una tabla WORD_LIST_TABLE
- Columans para _id, word y description

Después de inserter las palabras "alpha" y "beta", donde alpha tiene dos definiciones, la tabla podría lucir como esto:

DATABASE_NAME

WORD_LIST_TABLE		
_id	**word**	**definition**
1	"alpha"	"first letter"
2	"beta"	"second letter"
3	"alpha"	"particle"

Se puede encontrar lo que hay en una fila específica usando el _id, o puede recuperar filas formulando consultas que seleccionen filas de la tabla especificando restricciones. Se utiliza el lenguaje de consultas SQL que se describe a continuación para crear consultas (queries).

Transacciones

Una transacción es una secuencia de operaciones realizadas como una única unidad lógica de trabajo. Una unidad lógica de trabajo debe exhibir cuatro propiedades, llamadas propiedades de atomicidad (atomicity), consistencia (consistency), aislamiento (isolation) y durabilidad (durability) (de allí se conozcan como **ACID**), para calificar como una transacción.

Todos los cambios dentro de una sola transacción en SQLite ocurren completamente o no ocurren en absoluto, incluso si el acto de escribir el cambio en el disco es interrumpido por

- un fallo del programa,
- un fallo del sistema operativo, o
- un corte de corriente

Ejemplo de transacciones:

- Transferir dinero de una cuenta de ahorros a una cuenta corriente.
- Introducir un término y una definición en el diccionario.
- Asignar una lista de cambios a la rama maestra.

ACID

- **Atomicity (atomicidad).** O bien se realizan todas las modificaciones de datos, o bien no se realiza ninguna de ellas.
- **Consistency.** Una vez completada, una transacción debe dejar todos los datos en un estado consistente.
- **Isolation.** Las modificaciones hechas por transacciones concurrentes deben ser aisladas de las modificaciones hechas por cualquier otra transacción concurrente. Una transacción

reconoce los datos o bien en el estado en el que se encontraban antes de que otra transacción concurrente los haya modificado, o reconoce los datos después de que la segunda transacción haya finalizado, pero no reconoce un estado intermedio.

- **Durability.** Después de que una transacción se ha completado, sus efectos están permanentemente en el sistema. Las modificaciones persisten incluso en caso de fallo del sistema. After a transaction has completed, its effects are permanently in place in the system. The modifications persist even in the event of a system failure.

Más acerca de transacciones en este enlace.

Query language (lenguaje de consulta)

Se usará un lenguaje especial de consulta SQL para interactuar con la base de datos. Las consultas pueden ser muy complejas, sin embargo sus operaciones básicas son:

- Insertar filas
- Eliminar filas
- Actualizar valores en una fila
- Obtener filas que cumplen determinados criterios

En Android, el objeto base de datos proporciona métodos convenientes para insertar, eliminar y actualizar la base de datos. Sólo necesita entender SQL para recuperar datos.

Descripción completa del lenguaje query.

Estructura del Query

Uan consulta SQL es altamente estructurada y contiene las siguientes partes básicas:

- **SELECT** word, description **FROM** WORD_LIST_TABLE **WHERE** word="alpha"

Versión genérica de la consulta ejemplo:

- **SELECT** columnas **FROM** tabla **WHERE** columna="valor"

Partes:

- **SELECT** columnas—selecciona las columnas a retornar. Se usa ***** para devolver todas las columnas.
- **FROM** tabla—especifica la tablade la cual se obtienen los resultados.
- **WHERE**— palabra clave que indica las condiciones que deben cumplirse.
- columna="valor"—la condición ha ser cumplida.
    - operadores comunes: =, LIKE, <, >
- **AND**, **OR**—connecta multiple conditiones usando operadores lógicos.

- **ORDER BY**—es omitida para un orden predeterminado, o se especifica **ASC** para ascendente **DESC** para descendente.
- **LIMIT** es una palabra clave muy útil si se desea obtener un número limitado de resultados.

Consultas de ejemplo:

1	SELECT * FROM WORD_LIST_TABLE	Obtiene toda la tabla.
2	SELECT word, definition FROM WORD_LIST_TABLE WHERE _id > 2	Retorna `[["alpha", "particle"]]`
3	SELECT _id FROM WORD_LIST_TABLE WHERE word="alpha" AND definition LIKE "%art%"	Retorna el id de la palabra alpha con el substring “art” en la definición `[["3"]]`
4	SELECT * FROM WORD_LIST_TABLE ORDER BY word DESC LIMIT 1	Sort in reverse and get the first item. This gives you the last item per sort order. Sorting is by the first column, in this case, the _id. `[["3", "alpha", "particle"]]`
5	SELECT * FROM WORD_LIST_TABLE LIMIT 2,1	Returna 1 elemento en la posición 2. La cuenta de la posición comienza en 1 (no en cero!). Retorna `[["2", "beta", "second letter"]]`

Se puede practicar creando y consultando bases de datos en este sitio Fiddle website y este otro W3Schools.

Consultas en el Android SQLite

Se puede enviar consultas a la base de datos SQLite del sistema Android como consultas directas (raw queries) o como parámetros.

- rawQuery(String sql, String[] selectionArgs) ejecuta el SQL sumiistrado y retorna un Cursor del conjunto de resultados.

La siguiente tabla muestra cómo se verían las dos primeras consultas de arriba como raw query.

1	String query = "SELECT * FROM WORD_LIST_TABLE"; rawQuery(query, null);
2	query = "SELECT word, definition FROM WORD_LIST_TABLE WHERE _id> ? "; String[] selectionArgs = new String[]{"2"} rawQuery(query, selectionArgs) ;

- query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit) consulta la table dada, retorna un Cursor sobre el result set.

Aquí hay un query mostrando como llenar los argumentos:

```
SELECT * FROM WORD_LIST_TABLE
```

```
WHERE word="alpha"
ORDER BY word ASC
LIMIT 2,1;
```

Retorna:

```
[["alpha", "particle"]]
String table = "WORD_LIST_TABLE"
String[] columns = new String[]{"*"};
String selection = "word = ?"
String[] selectionArgs = new String[]{"alpha"};
String groupBy = null;
String having = null;
String orderBy = "word ASC"
String limit = "2,1"

query(table, columns, selection, selectionArgs, groupBy, having, orderBy, limit);
```

Note que en el código real, no crearía variables para los valores null. Ver SQLiteDatabase documentation para versions de este método con diferentes parámetros.

Cursors

Las consultas siempre retornan un objeto Cursor. Un Cursor es un objeto interfaz que suministra un acceso aleatorio de lectura-escritura al result set retornado por la consulta a la base de datos. Apunta al primer elemento del resultado de la consulta.

Un cursor es un apuntador a una fila de los datos estructurados. Se puede decir que es como un apuntador de las filas de la tabla.

La clase Cursor suministra métodos para mover el cursor a través de esa estructura y métodos para obtener los datos de las columnas de cada fila.

Cuando un método devuelve un objeto Cursor, se itera sobre el resultado, se extraen los datos, se hace algo con ellos y, finalmente, se cierra el cursor para liberar la memoria.

SQLite Database

Este capítulo trata sobre las clases SQLiteDatabase y SQLiteOpenHelper del framework Android. El capítulo asume que usted está familiarizado con las bases de datos SQL en general, y con la construcción básica de consultas SQL. Revise el capítulo de Introducción a SQL si necesita una breve actualización.

De las muchas opciones de almacenamiento discutidas, el uso de una base de datos SQLite es una de las más versátiles y fáciles de implementar.

- Una base de datos SQLite es una buena solución de almacenamiento cuando tiene datos estructurados que necesita almacenar de forma persistente y acceder, buscar y cambiar con frecuencia.

- Puede utilizar la base de datos como el almacenamiento principal para los datos de usuarios o aplicaciones, o puede utilizarla para almacenar en caché y poner a disposición los datos obtenidos de la nube.
- Si puede representar sus datos como filas y columnas, considere una base de datos SQLite.
- Los Content Providers, que serán presentados en un capítulo posterior, trabajan excelentemente con bases de datos SQLite.

Cuando utiliza una base de datos SQLite, representada como un objeto SQLiteDatabase, todas las interacciones con la base de datos se realizan a través de una instancia de la clase SQLiteOpenHelper que ejecuta sus peticiones y administra la base de datos por usted. La aplicación sólo debe interactuar con el SQLiteOpenHelper, que se describirá a continuación.

Hay dos tipos de datos asociados con el uso de bases de datos SQLite en particular, Cursor y ContentValues.

Cursor

La SQLiteDatabase siempre presenta los resultados como un Cursor en un formato de tabla que se asemeja al de una base de datos SQL.

Puede pensar en los datos como una matriz de filas. Un cursor es un puntero en una fila de esos datos estructurados. La clase Cursor proporciona métodos para mover el cursor a través de la estructura de datos, y métodos para obtener los datos de los campos de cada fila.

La clase Cursor tiene un número de subclases que implementan cursores para tipos específicos de datos.

- SQLiteCursor expone los resultados de una consulta en una base de datos SQLiteDatabase. SQLiteCursor no está sincronizado internamente, por lo que un código que utilice un SQLiteCursor en múltiples hilos debería realizar su propia sincronización cuando utilice el SQLiteCursor.
- MatrixCursor es un todoterreno, una implementación de cursor mutable respaldada por un arreglo de objetos que expande automáticamente la capacidad interna según sea necesario.

Algunas operaciones comunes del cursor son:

- getCount() devuelve el número de filas en el cursor.
- getColumnNames() devuelve un arreglo de strings que contiene los nombres de todas las columnas del result set en el orden en que fueron listadas en el resultado.
- getPosition() devuelve la posición actual del cursor en un grupo de filas.
- Hay getters disponibles para tipos de datos específicos, tales como getString(int column) y getInt(int column).
- Operaciones tales como moveToFirst() y moveToNext() mueven el cursor.

- close() libera todos los recursos e invalida por complete el cursor. Recuerde llamar a close para liberar los recursos!

Procesando cursores

Cuando una llamada de método devuelve un cursor, se itera sobre el resultado, se extraen los datos, se hace algo con los datos y, finalmente, se debe cerrar el cursor para liberar la memoria. Si no lo hace, la aplicación se puede bloquear cuando se queda sin memoria.

El cursor comienza antes de la primera línea de resultados, de modo que en la primera iteración se desplaza el cursor al primer resultado, si existe. Si el cursor está vacío o ya se ha procesado la última línea, el loop finaliza. No olvide cerrar el cursor una vez que haya terminado con él. (Esto no se puede repetir demasiado a menudo.)

```
// Perform a query and store the result in a Cursor
Cursor cursor = db.rawQuery(...);
try {
    while (cursor.moveToNext()) {
        // Do something with the data
     }
} finally {
    cursor.close();
}
```

Cuando se utiliza una base de datos SQL, puede implementar la clase SQLiteOpenHelper para devolver el cursor a la activity de llamada o al adapter, o puede convertir los datos a un formato que sea más adecuado para el adapter. La ventaja de esto último es que la gestión del cursor (y su cierre posterior) es manejado por el open helper, y la interfaz de usuario queda independiente de lo que sucede en el backend. Vea la práctica Base de Datos SQLite para un ejemplo de implementación.

ContentValues

De forma similar como los extras almacenan los datos, una instancia de ContentValues almacena los datos como pares clave-valor, donde la clave es el nombre de la columna y el valor es el valor de la celda. Una instancia de ContentValues representa una fila de una tabla.

El método insert() de la base de datos requiere que los valores para llenar una fila sean pasdos como una instancia de ContentValues.

```
ContentValues values = new ContentValues();
// Insert one row. Use a loop to insert multiple rows.
values.put(KEY_WORD, "Android");
values.put(KEY_DEFINITION, "Mobile operating system.");

db.insert(WORD_LIST_TABLE, null, values);
```

Implementando una base de datos SQLite

Para implementar una base de datos en una aplicación Android, se necesita hacer lo siguiente.

1. (Recomendado) Crear un modelo de datos.
2. Hacer una subclase de SQLiteOpenHelper
    a. Usar constantes para los nombres de las tablas y las consultas de creación de las bases de datos
    b. Implementar onCreate para crear la base de datos SQLiteDatabase con tablas para los datos
    c. Implementar onUpgrade()
    d. Implementar métodos opcionales
3. Implementar los métodos query(), insert(), delete(), update(), count() de SQLiteOpenHelper.
4. En la MainActivity, crear una instancia de SQLiteOpenHelper.
5. Llamar los métodos de SQLiteOpenHelper para trabajar con la base de datos.

Advertencias:

- Cuando implemente los métodos, siempre ponga las operaciones de la base de datos en bloques try/catch.
- Las aplicaciones de ejemplo no validan los datos del usuario. Cuando escriba una aplicación para su publicación, asegúrese siempre de que los datos de usuario son los que espera para evitar la inyección de datos erróneos o la ejecución de comandos SQL maliciosos en su base de datos.

Modelo de Datos

Es una Buena práctica crear una clase que represente los datos con métodos getters y setters.

Para una base de datos SQLite, una instancia de esta clase podría representar un registro, y para una base de datos simple, una fila en una tabla.

```
public class WordItem {
    private int mId;
    private String mWord;
    private String mDefinition;
    // Getters and setters and more
}
```

Subclase SQLiteOpenHelper

Cualquier open helper que se cree debe extender SQLiteOpenHelper.

```
public class WordListOpenHelper extends SQLiteOpenHelper {

    public WordListOpenHelper(Context context) {
       super(context, DATABASE_NAME, null, DATABASE_VERSION);
       Log.d(TAG, "Construct WordListOpenHelper");
    }
}
```

Definir constantes para los nombres de las tablas

Aunque no es obligatorio, es habitual declarar los nombres de tablas, columnas y filas como constantes. Esto hace que su código sea mucho más legible, facilita el cambio de nombres, y las consultas terminarán pareciéndose mucho más a SQL. Se puede hacer esto en la clase open helper, o en una clase pública separada; aprenderá más sobre esto en el capítulo sobre content providers.

```
  private static final int DATABASE_VERSION = 1;
  // has to be 1 first time or app will crash
  private static final String WORD_LIST_TABLE = "word_entries";
  private static final String DATABASE_NAME = "wordlist";

  // Column names...
  public static final String KEY_ID = "_id";
  public static final String KEY_WORD = "word";

  // ... and a string array of columns.
  private static final String[] COLUMNS = {KEY_ID, KEY_WORD};
```

Definir la consulta de creación de la base de datos

Se necesita un query que cree una tabla para crear una base de datos. Esto también se define habitualmente como una constante de string. Este ejemplo básico crea una tabla con una columna de un id con auto-incremento y una columna para contener palabras.

```
    private static final String WORD_LIST_TABLE_CREATE =
            "CREATE TABLE " + WORD_LIST_TABLE + " (" +
             KEY_ID + " INTEGER PRIMARY KEY, " +
             // will auto-increment if no value passed
             KEY_WORD + " TEXT );";
```

Implementar onCreate() y crear la base de datos

El método onCreate es llamado unicamente si no existe la base de datos. Se crean las tablas y opcionalmente se agregan los datos iniciales.

```
@Override
public void onCreate(SQLiteDatabase db) { // Creates new database
   db.execSQL(WORD_LIST_TABLE_CREATE); // Create the tables
   fillDatabaseWithData(db); // Add initial data
   // Cannot initialize mWritableDB and mReadableDB here, because
   // this creates an infinite loop of on Create()
   // being repeatedly called.
}
```

Implementar onUpgrade()

Este método es requerido.

Si la base de datos actúa sólo como caché para los datos que también se almacenan en línea, puede eliminar las tablas y volver a crearlas una vez finalizada la actualización.

Nota: Si la base de datos es el almacenamiento principal, debe conservar los datos del usuario antes de hacer esto, ya que esta operación destruye todos los datos.

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // SAVE USER DATA FIRST!!!
        Log.w(WordListOpenHelper.class.getName(),
                "Upgrading database from version " + oldVersion + " to "
                        + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS " + WORD_LIST_TABLE);
        onCreate(db);
}
```

Métodos opcionales

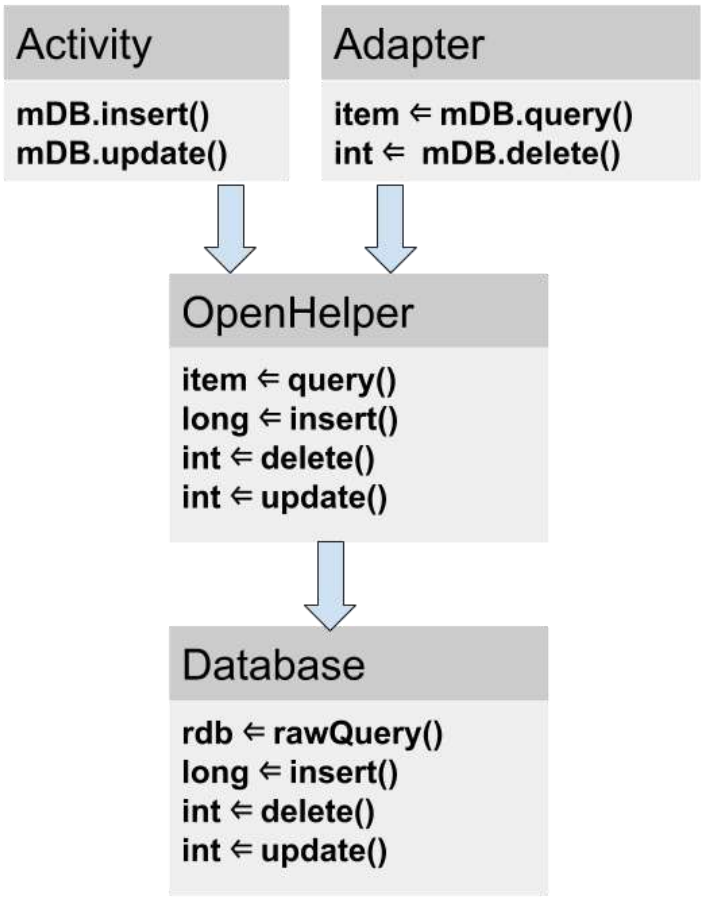
La clase open helper suministra métodos adicionales que pueden ser sobreescritos a medida que sean necesarios.

- onDowngrade()—La implementación predeterminada rechaza los downgrades.
- onConfigure()—llamado antes del onCreate. Utilizar esta opción sólo para llamar a los métodos que configuran los parámetros de la conexión de la base de datos.
- onOpen()—Cualquier otro trabajo que no sea la configuración que se necesite hacer después de abrir la base de datos.

Operaciones de la Database

Aunque puede hacer lo que quiera en los métodos a llamar en el open helper y hacer que devuelvan lo que quiera a la actividad de llamada, es una buena idea usar los métodos estandarizados de query(), insert(), delete(), update(), count() que coincidan con la API de la base de datos y los content providers. Usar este formato hará más fácil de añadir un content provider o un loader en el futuro, y hace más fácil que otras personas entiendan el código.

El siguiente diagrama muestra cómo deben diseñarse las diferentes API para lograr consistencia y claridad.



query()

El método query que implementa en la clase open helper puede tomar y devolver cualquier tipo de datos que la interfaz de usuario necesite.

Dado que el open helper proporciona métodos prácticos para insertar, eliminar y actualizar filas, no es necesario que el método de consulta sea genérico y soporte estas operaciones.

En general, el método query debería permitir sólo consultas que sean necesarias para la aplicación y no ser de uso general.

La base de datos proporciona dos métodos para enviar consultas: SQLiteDatabase.rawQuery() y SQLiteDatabase.query(), con varias opciones para los argumentos.

SQLiteDatabase.rawQuery()

El método query del open helper puede construir un query SQL y enviarla como un rawQuery a la base de datos el que devuelva un cursor. Si los datos son suministrados por la aplicación, y bajo control total del desarrollador, se puede utilizar rawQuery().

```
rawQuery(String sql, String[] selectionArgs)
```

- El primer parámetro para db.rawquery() es un string de query SQLite.
- El segundo parámetro contiene los argumentos.

```
cursor = mReadableDB.rawQuery(queryString, selectionArgs);
```

SQLiteDatabase.query()

Si está procesando datos suministrados por el usuario, incluso después de la validación, es más seguro construir una consulta y utilizar una versión del método SQLiteDatabase.query() para la base de datos. Los argumentos son los que se esperan en SQL y están documentados en SQLiteDatabase documentation..

```
Cursor query (boolean distinct,  String table, String[] columns, String selection,
                String[] selectionArgs, String groupBy, String having,
                String orderBy,String limit)
```

Aquí hay un ejemplo básico:

```
String[] columns = new String[]{KEY_WORD};
String where =  KEY_WORD + " LIKE ?";
searchString = "%" + searchString + "%";
String[] whereArgs = new String[]{searchString};
cursor = mReadableDB.query(WORD_LIST_TABLE, columns, where, whereArgs, null, null,
null);
```

Ejemplo de un query() del open helper completo

```
public WordItem query(int position) {
   String query = "SELECT  * FROM " + WORD_LIST_TABLE +
```

```
            " ORDER BY " + KEY_WORD + " ASC " +
            "LIMIT " + position + ",1";

    Cursor cursor = null;
    WordItem entry = new WordItem();

    try {
        if (mReadableDB == null) {mReadableDB = getReadableDatabase();}
        cursor = mReadableDB.rawQuery(query, null);
        cursor.moveToFirst();
        entry.setId(cursor.getInt(cursor.getColumnIndex(KEY_ID)));
        entry.setWord(cursor.getString(cursor.getColumnIndex(KEY_WORD)));
    } catch (Exception e) {
        Log.d(TAG, "EXCEPTION! " + e);
    } finally {
        // Must close cursor and db now that we are done with it.
        cursor.close();
        return entry;
    }
}
```

insert()

El método insert() del open helper llama al SQLiteDatabase.insert(), que es un método conveniente de SQLiteDatabase para insertar una fila en la base de datos. (Es un método de conveniencia, porque no tiene que escribir la consulta SQL por usted mismo).

Formato

```
long insert(String table, String nullColumnHack, ContentValues values)
```

- El primer argumento es el nombre de la tabla.
- El segundo argumento es un String llamado nullColumnHack. Es una forma de solución que le permite insertar filas vacías. Véase the documentation for insert(). Use `null`.
- El tercer argumento debe ser un contenedor ContentValues con los valores para llenar la fila. Este ejemplo solo tiene una columna; para tablas con multiples columnas, se deben agregar los valores de cada columna en este contenedor.
- El método de la base de datos devuelve el id del Nuevo element insertado y debe pasar eso en la aplicación.

Ejemplo

```
newId = mWritableDB.insert(WORD_LIST_TABLE, null, values);
```

delete()

El método delete del open helper llama al método delete() de la base de datos, el cual es un método de conveniencia de manera tal que no hay necesidad de escribir el query SQL completo.

Formato

```
int delete (String table, String whereClause, String[] whereArgs)
```

- El primer argumento es el nombre de la tabla.
- El segundo argumento es una cláusula WHERE.
- El tercer argumento son los argumentos de la cláusula WHERE.

Se puede borrar utilizando cualquier criterio y el método devuelve el número de posiciones que se borraron realmente, el cual el open helper también debería devolver.

Ejemplo

```
deleted = mWritableDB.delete(WORD_LIST_TABLE,
                    KEY_ID + " =? ", new String[]{String.valueOf(id)});
```

update()

El método update del open heper llama al método update() de la base de datos, el cual es un método conveniente para que no se tenga que escribir la consulta SQL completa. Los argumentos son familiares a los métodos anteriores, y el onUpdate devuelve el número de filas actualizadas.

Formato

```
int update(String table, ContentValues values, String whereClause,
           String[] whereArgs)
```

- El primer argumento es el nombre de la tabla.
- El segundo argumento debe ser un ContentValues con nuevos valores para la fila.
- El tercer argumento es una cláusula WHERE.
- El cuarto argumento son los argumentos de la cláusula WHERE.

Ejemplo

```
ContentValues values = new ContentValues();
values.put(KEY_WORD, word);
```

```
mNumberOfRowsUpdated = mWritableDB.update(WORD_LIST_TABLE,
values, // new values to insert
KEY_ID + " = ?",
new String[]{String.valueOf(id)});
```

count()

El método count() devuelve el número de entradas en la base de datos. Si está usando un RecyclerView.Adapter, tiene que implementar getItemCount(), el cual necesita obtener el número de filas del open helper el que a su vez necesita obtenerlo de la base de datos.

En el adapter

```
@Override
public int getItemCount() {
    return (int) mDB.count();
}
```

En el open helper

```
public long count(){
    if (mReadableDB == null) {mReadableDB = getReadableDatabase();}
    return DatabaseUtils.queryNumEntries(mReadableDB, WORD_LIST_TABLE);
}
```

queryNumEntries() es un método de la clase pública DatabaseUtils, la cual proporciona muchos métodos prácticos para trabajar con cursores, bases de datos y también proveedores de contenido.

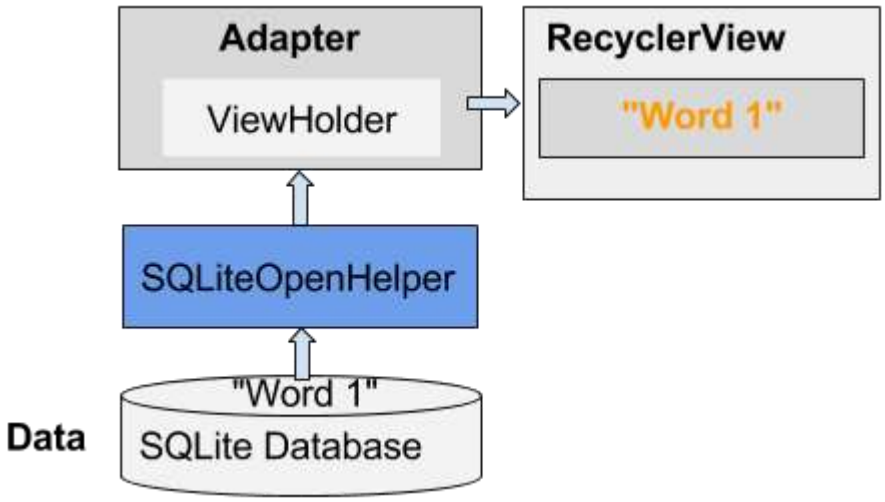
Instanciar el Open Helper

Para obtener un handler de la base de datos, en la MainActivity, en el método onCreate, llame:

```
mDB = new WordListOpenHelper(this);
```

Working with the database

Es un patron común combiner una SQLiteDatabase de backend con un RecyclerView para mostrar los datos.



Por ejemplo:

- Presionando el botón FAB se puede iniciar una actividad que obtenga información del usuario y la almacene en la base de datos como un elemento nuevo o actualizado.
- Deslizar un elemento puede eliminarlo después de que el usuario confirme la eliminación.

Transacciones

Use transacciones

- cuando se realizan varias operaciones que deban completarse para mantener la coherencia de la base de datos, por ejemplo, la actualización de los precios de los artículos relacionados en un evento de venta.
- para agrupar varias operaciones independientes para mejorar el rendimiento, como inserciones en masa.

Las transacciones pueden anidarse, y la clase SQLiteDatabase proporciona métodos adicionales para gestionar las transacciones anidadas. Ver SQLiteDatabase references documentation..

Expresión de una transacción

```
db.beginTransaction();
try {
  ...
  db.setTransactionSuccessful();
} finally {
  db.endTransaction();
}
```

Respaldando las bases de datos

Es una buena idea hacer respaldo de las base de datos de las aplicaciones.

Se puede hacer respaldo utilizando las opciones de Cloud Backup que se describen en el capítulo Opciones de almacenamiento.

Entrega de una base de datos con la aplicación

A veces es posible que se desee incluir una base de datos completa con la aplicación. Hay varias maneras de hacerlo, y hay compromisos para cada una.

- Incluir los comandos SQL con la aplicación y hacer que cree la base de datos e inserte los datos en el primer uso. Esto es básicamente lo que usted hará en la práctica para el almacenamiento de datos. Si la cantidad de datos que se quiere poner en la base de datos es pequeña, sólo como un ejemplo para que el usuario pueda ver algo, se puede usar este método.
- Enviar los datos con el APK como recurso, y construir la base de datos cuando el usuario abra la aplicación por primera vez. Esto es similar al primer método, pero en lugar de definir los datos en el código, se ponen en un recurso, por ejemplo, en formato CSV. A continuación, se puede leer los datos con un input stream y añadirlos a la base de datos.
- Construir y pre-poblar la base de datos SQLite e incluirla en el APK. Con este método se escribirá una aplicación que crea y rellena una base de datos. Se puede hacer esto en el emulador. A continuación, copie el archivo en el que está almacenada la base de datos (el directorio "/data/data/YOUR_PACKAGE/databases/") e inclúyalo como un activo en su aplicación. Cuando se inicie la aplicación por primera vez, se copia el archivo de base de datos en el directorio "/data/data/YOUR_PACKAGE/databases/".

La clase SQLiteAssetHelper , que se puede descargar desde Github, extiende a SQLiteOpenHelper para ayudar a hacerlo. Y en este post de Stackoverflow discute este tema con más detalle.

Tenga en cuenta que para usar una base de datos más grande, el rellenar la base de datos debe hacerse en segundo plano, y la aplicación no debe fallar si aún no existe una base de datos, o si la base de datos está vacía.

Resumen de la Aplicación

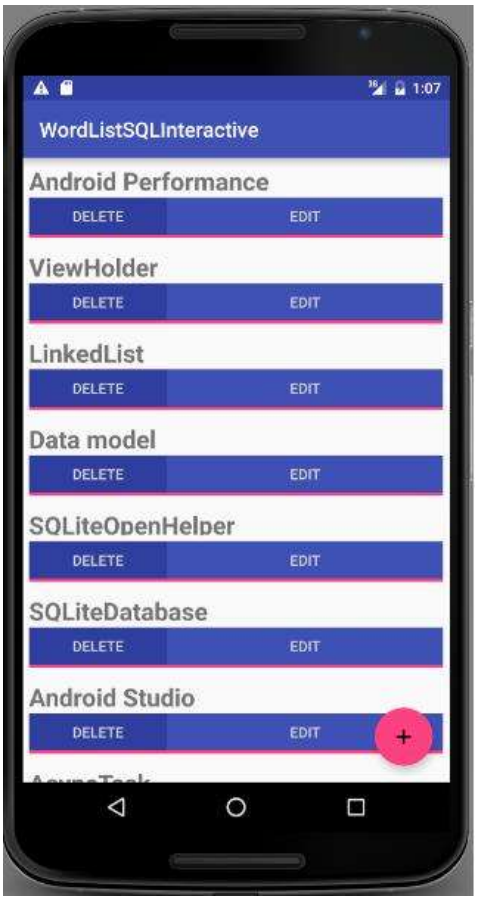
En esta práctica, creará una base de datos SQLite para un conjunto de datos, mostrará los datos obtenidos en un RecyclerView y le pondrá la funcionalidad para agregar, eliminar y editar los datos del RecyclerView y almacenarlos en la base de datos.

Nota: Una base de datos que almacena los datos de forma persistente y los abstrae en un modelo de datos es suficiente para pequeñas aplicaciones Android con una complejidad mínima. En capítulos posteriores, aprenderá a diseñar la aplicación utilizando loaders y content providers para separar aún más los datos de la UI. Estas clases ayudarán a mover el trabajo fuera del hilo de la UI para ayudar a hacer que la experiencia del usuario sea lo más suave y natural posible. Además de mejorar la experiencia del usuario al eliminar un posible problema de rendimiento, mejoran la capacidad para ampliar y mantener una aplicación.

Importante: En esta práctica, el SQLiteOpenHelper ejecuta operaciones de base de datos en el hilo principal. En una aplicación de producción, donde las operaciones de la base de datos pueden llevar bastante tiempo, estas operaciones se realizarían en un hilo de background, por ejemplo, usando un loader como AsyncTaskLoader y CursorLoader.

A partir de una aplicación esqueleto, le agregará la funcionalidad para:

- Mostrar las palabras de una base de datos SQLite en un RecyclerView.
- Que cada palabra puede ser editada o borrada.
- Poder añadir nuevas palabras y almacenarlas en la base de datos.



La minima version de SDK es el API15: Android 4.0.3 IceCreamSandwich y el *target* SDK es la actual version de Android (version 27 al momento de recopilar este libro).

Tarea 0. Descargar y correr el código de arranque

Con el fin de ahorrarle trabajo, en particular al escribir actividades no relacionadas con la base de datos y el código de la interfaz de usuario, necesita obtener el código inicial para esta práctica.

1. Descargar el código de inicio WordListSqlStarterCode desde M7.
2. Abrir la aplicación en Android Studio.
3. Ejecutar la aplicación. Deberá ver la UI como se muestra en el previo screenshot. Todas las palabras mostradas deben ser "placeholder". Al hacer clic en los botones no debe hacer nada.

Tarea 1. Extender la SQLiteOpenHelper para crear y popular la base de datos

Las aplicaciones Android pueden utilizar bases de datos SQLite estándar para almacenar datos. Esta práctica no enseña SQLite, pero muestra cómo usarlo en una aplicación Android. Para obtener más información sobre cómo aprender sobre SQLite, consulte el apartado Introducción en este mismo capítulo.

SQLOpenHelper es una clase de utilidad en el SDK de Android para interactuar con un objeto SQLite database . Incluye los métodos onCreate() y onUpdate() que debe implementar, y loñs métodos de conveniencia insert, delete, update y query para todas las interacciones de la base de datos.

La clase SQLOpenHelper se encarga de abrir la base de datos si existe, crearla si no existe y actualizarla según sea necesario.

Nota: Puede tener más de una base de datos por aplicación, y más de un open helper gestionándolas. Sin embargo, considere la posibilidad de crear varias tablas en la misma base de datos en lugar de utilizar varias bases de datos por el rendimiento y la simplicidad arquitectónica.

1.1 Crear un esqueleto de la clase WordListOpenHelper

El primer paso para añadir una base de datos al código es siempre crear una subclase de SQLiteOpenHelper e implementar sus métodos.

1. Crear una nueva clase Java **WordListOpenHelper** con la siguiente firma.

```
public class WordListOpenHelper extends SQLiteOpenHelper {}
```

2. En el editor de código, colóquese sobre el error y haga clic en el bombillo rojo y seleccione **Implement methods**. Asegúrese que ambos métodos están resaltados y haga clic en **OK**.
3. Agregue el constructor que falta para la clase WordListOpenHelper. (Más Adelante se agregarán las constantes indefinidas.)

```
public WordListOpenHelper(Context context) {
```

```
   super(context, DATABASE_NAME, null, DATABASE_VERSION);
}
```

1.2. Agregar las constantes de base de datos a WordListOpenHelper

1. AL tope de la clase WordListOpenHelper, defina las constantes para las tablas, filas y columnas tal como se muestra en el código inferior. Esto debe hacer desaparecer todos los errores.

```
// It's a good idea to always define a log tag like this.
private static final String TAG = WordListOpenHelper.class.getSimpleName();

// has to be 1 first time or app will crash
private static final int DATABASE_VERSION = 1;
private static final String WORD_LIST_TABLE = "word_entries";
private static final String DATABASE_NAME = "wordlist";

// Column names...
public static final String KEY_ID = "_id";
public static final String KEY_WORD = "word";

// ... and a string array of columns.
private static final String[] COLUMNS = { KEY_ID, KEY_WORD };
```

2. Ejecute el código para asegurarse que ya no hay más errores.

1.3. Construir la consulta SQL y el código para crear la base de datos

Las consultas SQL pueden llegar a ser bastante complejas. Es una buena práctica construir las consultas por separado del código que las utiliza. Esto aumenta la legibilidad del código y ayuda con la depuración.

Continúe añadiendo código a WordListOpenHelper.java:

1. Por debajo de las constantes, añada el siguiente código para construir el query. Refiérase a la Introducción si necesita ayuda para entender esta consulta.

```
// Build the SQL query that creates the table.
private static final String WORD_LIST_TABLE_CREATE =
       "CREATE TABLE " + WORD_LIST_TABLE + " (" +
               KEY_ID + " INTEGER PRIMARY KEY, " +
               // id will auto-increment if no value passed
               KEY_WORD + " TEXT );";
```

2. Añada variables de instancia para referenciar las variables de bases de datos writable y readable. Almacenar estas referencias le ahorra el trabajo de obtener una referencia de base de datos cada vez que necesite leer o escribir.

```
private SQLiteDatabase mWritableDB;
private SQLiteDatabase mReadableDB;
```

3. En el método onCreate, agregue código para crear una base de datos y la tabla (La clase helper no crea otra base de datos, si una ya existe).

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL(WORD_LIST_TABLE_CREATE);
}
```

4. Corriga el error renombrando el argumento SQLiteDatabase del método como db.

1.4 Crear la base de datos en el onCreate de la MainActivity

Para crear la base de datos, cree una instancia de la clase WordListOpenHelper que recién escribió.

1. Abra MainActivity.java y agregue una variable de instancia para el open helper:

```
private WordListOpenHelper mDB;
```

2. En el onCreate, inicialice mDBcon una instancia de WordListOpenHelper. Esto llama al onCreate de la WordListOpenHelper, la cual crea la base de datos.

```
mDB = new WordListOpenHelper(this);
```

3. Agregue un breakpoint, ejecute la aplicación con el depurador (debugger), y verifique que la mDB es una instancia de WordListOpenHelper.

1.5 Agregar datos a la base de datos

La lista de palabras para la aplicación puede provenir de muchas fuentes. Puede ser completamente creada por el usuario, o descargado de Internet, o generado desde un archivo que es parte del APK. Para esta práctica, se alimentará la base de datos con una pequeña cantidad de datos cableados.

Tenga en cuenta que la adquisición, creación y formateo de datos es un tema completamente separado que no se cubre en este curso.

1. Abra WordListOpenHelper.java.

2. En el onCreate, después de agregar la base de datos, agregue una función para llamarla

```
fillDatabaseWithData(db);
```

A continuación, implemente el método `fillDatabaseWithData()` en WordListOpenHelper.

3. Implemente la plantilla del método.

```
private void fillDatabaseWithData(SQLiteDatabase db){}
```

4. Dentro del método, declare un string de palabras para ser usados como datos simulados.

```
String[] words = {"Android", "Adapter", "ListView", "AsyncTask",
    "Android Studio", "SQLiteDatabase", "SQLOpenHelper",
    "Data model", "ViewHolder","Android Performance",
    "OnClickListener"};
```

5. Cree un contenedor para los datos. El método insert al que llamará a continuación requiere que los valores llenen una fila como instancia de ContentValues. Un ContentValues almacena los datos de una fila como pares clave/valor, donde la clave es el nombre de la columna y el valor es el valor a establecer.

```
// Create a container for the data.
ContentValues values = new ContentValues();
```

6. Añada la clave/valor de la primera fila a `values` y, a continuación, inserte esa fila en la base de datos. Repita para todas las palabras de su selección de palabras.

    - db.insert es un método de conveniencia de SQLiteDatabase para inserter una fila en la base de datos. (Es un método de conveniencia, ya que usted no necesita escribir el query SQL.)
    - El primer argumentode `db.insert` es el nombre de la table, WORD_LIST_TABLE.
    - El Segundo argumento es un String nullColumnHack. Es una forma de solución SQL que le permite insertar filas vacías. Véase the documentation for insert(). Use `null` para este argumento.
    - El tercer argumento debe ser un contenedor ContentValues con los valores a llenar la fila. Este ejemplo solo tiene la columna “words” representada por la constante KEY_WORD declarada anteriormente; para tablas con múltiples columnas, agregue los valores de cada columna al contenedor.

```
for (int i=0; i < words.length; i++) {
     // Put column/value pairs into the container.
     // put() overrides existing values.
     values.put(KEY_WORD, words[i]);
     db.insert(WORD_LIST_TABLE, null, values);
 }
```

7. Antes de ejecutar y probar la aplicación, debe borrar los datos de su base de datos SQLite y eliminar la base de datos. Entonces podrá ejecutar la aplicación y recrearla para que la base de datos se inicialice con los datos de la semilla. Puede desinstalar la aplicación del

dispositivo, o puede borrar todos los datos de la aplicación desde **Settings > Apps > WordList > Storage > Clear Data** en el emulador de Android o dispositivo físico.

8. Ejecutar la aplicación. No apreciará mingún cambio en la intefaz de usuario.
    - Compruebe los logs y asegúrese de que no hay errores antes de continuar. Si encuentra errores, lea los mensajes del logcat cuidadosamente y utilice recursos, como Stack Overflow, si se queda atascado.
    - También puede verificar en ajustes, que la aplicación está usando el almacenamiento.

Tarea 2. Creae el modelo de datos para una simple palabra

Un modelo de datos es una clase que encapsula una estructura de datos compleja y proporciona una API para acceder y manipular los datos de esa estructura. Necesita un modelo de datos para pasar los datos recuperados de la base de datos a la UI.

Para esta práctica, el modelo de datos sólo contiene la palabra y su id. Mientras que el id único será generado por la base de datos, necesita una forma de pasar el id a la interfaz de usuario. Esto identificará la palabra que el usuario está cambiando.

2.1. Create a data model for your word data

1. Cree una nueva clase y llamela WordItem.
2. Agregur las siguientes variables a la clase.

```
private int mId;
private String mWord;
```

3. Agregue un cosntructor vacío.
4. Agregue los getters y setters para el id y word.
5. Ejecute la aplicación. No verá ningún cambio en la UI, pero tampoco deben haber errores.

Solución:

```
public class WordItem {

   private int mId;
   private String mWord;

   public WordItem() {}

   public int getId() {return this.mId;}

   public String getWord() {return this.mWord;}

   public void setId(int id) {this.mId = id;}
```

```
    public void setWord(String word) {this.mWord = word;}
}
```

Tarea 3. Implementar el método query() en WordListOpenHelper

El método query() recupera las filas de la base de datos seleccionadas por una consulta SQL.

Para este ejemplo, para poder mostrar las palabras en el RecyclerView, necesitamos obtenerlas de la base de datos, una a la vez, según sea necesario. La palabra necesaria se identifica por su posición en la view.

Como tal, el método de consulta tiene un parámetro para la posición solicitada y devuelve un WordItem.

3.1. Implementar el método query()

1. Cree un método query que tome un argumento `int position` y retorne un `WordItem`.

```
public WordItem query(int position) {
}
```

2. Construya un String query que retorne solamente la enésima fila del resultado. Use LIMIT con `position` como la fila y 1 como la cantidad de filas.

```
 String query = "SELECT  * FROM " + WORD_LIST_TABLE +
                " ORDER BY " + KEY_WORD + " ASC " +
                "LIMIT " + position + ",1";
```

3. Instancie una variable `Cursor` en `null` para contener el resultado de la base de datos.

```
Cursor cursor = null;
```

La SQLiteDatabase siempre presenta los resultados como un Cursor en un formato de tabla que se asemeja a una base de datos SQL.

Un cursor es un puntero en una fila de datos estructurados. Puedes pensar en ello como una serie de filas. La clase Cursor proporciona métodos para mover el cursor a través de esa estructura, y métodos para obtener los datos de las columnas de cada fila.

4. Instancie un WordItem con nombre `entry`.

```
WordItem entry = new WordItem();
```

5. Agregue un bloque `try/catch/finally`.

```
try {} catch (Exception e) {} finally {}
```

6. Dentro del bloque try,

    a. Obtenga una base de datos readable si no existe aún.

```
if (mReadableDB == null) {
    mReadableDB = getReadableDatabase();
}
```

    b. Envíe un raw query a la base de datos y almacene el resultado en un cursor.

```
cursor = mReadableDB.rawQuery(query, null);
```

    El método query del open helper puede construir un string para el query SQL y enviarlo como rawQuery a la base de datos para que devuelva un cursor. Si los datos son suministrados por la aplicación, y bajo su control total, puede usar un raw query().

    c. Mueva el cursor al primer elemento.

```
cursor.moveToFirst();
```

    d. Establezca el id y la word del `entry` WordItem con los valores devueltos por el cursor.

```
entry.setId(cursor.getInt(cursor.getColumnIndex(KEY_ID)));
entry.setWord(cursor.getString(cursor.getColumnIndex(KEY_WORD)));
```

7. En el bloque catch, haga log de la exception.

```
Log.d(TAG, “EXCEPTION! “ + e);
```

8. En el bloque finally, cierre el `cursor` y retorne el WordItem `entry`.

```
cursor.close();
return entry;
```

Solución:

```
public WordItem query(int position) {
    String query = "SELECT  * FROM " + WORD_LIST_TABLE +
            " ORDER BY " + KEY_WORD + " ASC " +
            "LIMIT " + position + ",1";
```

```
    Cursor cursor = null;
    WordItem entry = new WordItem();

    try {
        if (mReadableDB == null) {
            mReadableDB = getReadableDatabase();
        }
        cursor = mReadableDB.rawQuery(query, null);
        cursor.moveToFirst();
        entry.setId(cursor.getInt(cursor.getColumnIndex(KEY_ID)));
        entry.setWord(cursor.getString(cursor.getColumnIndex(KEY_WORD)));
    } catch (Exception e) {
        Log.d(TAG, "QUERY EXCEPTION! " + e.getMessage());
    } finally {
        cursor.close();
        return entry;
    }
}
```

3.2. El método onUpgrade

Cada SQLiteOpenHelper debe implementar el método onUpgrade(), el cual determina lo que sucede si el número de versión de la base de datos cambia. Esto puede suceder si tiene usuarios existentes de la aplicación que utilizan una versión anterior de la base de datos. Este método se activa cuando se abre una base de datos por primera vez. La acción predeterminada habitual es eliminar la base de datos actual y volver a crearla.

Importante: Aunque está bien eliminar la tabla en una aplicación de ejemplo, en una aplicación de producción necesita migrar cuidadosamente los valiosos datos del usuario.

Puede usar el código siguiente para implementar el método onUpgrade() para este ejemplo.

Código de plantilla de onUpgrade():

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
   Log.w(WordListOpenHelper.class.getName(),
           "Upgrading database from version " + oldVersion + " to "
                   + newVersion + ", which will destroy all old data");
   db.execSQL("DROP TABLE IF EXISTS " + WORD_LIST_TABLE);
   onCreate(db);
}
```

Tarea 4. Mostrar los datos en el RecyclerView

Ahora que tiene una base de datos, con datos. A continuación, actualizará el WordListAdapter y la MainActivity para obtener y mostrar estos datos.

4.1. Update WordListAdapter to display WordItems

1. Abra WordListAdapter.
2. En el método `onBindViewHolder` reemplace el código que muestra los datos simulados con el código que obtiene un element de la base de datos y muestrelo. Notará un error en `mDB`.

```
WordItem current = mDB.query(position);
holder.wordItemView.setText(current.getWord());
```

3. Declare `mDB` as an instance variable.

```
WordListOpenHelper mDB;
```

4. Para obtene el valor de mDB, cambie el constructor del WordListAdapter y agregue un segundo parámetro de tipo WordListOpenHelper
5. Asigne el valor del parámetro a mDB. EL cosntructor debe lucir de la manera siguiente:

```
public WordListAdapter(Context context, WordListOpenHelper db) {
        mInflater = LayoutInflater.from(context);
        mContext = context;
        mDB = db;
}
```

Esto generará un error en MainActivity, debido que agregó un argumento al constructor del WordListAdapter.

6. Abra el MainActivity y agregue el argumento mDB.

```
mAdapter = new WordListAdapter (this, mDB);
```

7. Ejecute la apliación. Deb ever todas las palabras de la base de datos.

Tarea 5. Agregar nuevas palabaras a la base de datos

Cuando el usuario hace clic en el botón FAB, se abre una actividad que le permite introducir una palabra que se añade a la base de datos cuando hace clic en guardar.

El código de inicio le proporciona el listener del clic y la actividad EditWordActivity iniciada haciendo clic en el FAB. Agregará el código específico de la base de datos y unirá las piezas juntas, de abajo hacia arriba, como acaba de hacer con el método de consulta.

5.1. Escribir el método insert()

En el WordListOpenHelper:

1. Cree el método insert() con la siguiente firma. El usuario ingresa una palabra y el método retorn el id de la nueva entrada. El id generado puede ser grande, de manera que el insert devuelve un número de tipo long.

```
public long insert(String word){}
```

2. Declare una variable para el id. Si la operación insert falla, el método devuelve 0.

```
long newId = 0;
```

3. Como anteriormente, cree un ContentValues llamado values para los datos directos.

```
ContentValues values = new ContentValues();
values.put(KEY_WORD, word);
```

4. Coloque la operación de la base de datos en un bloque try/catch.

```
try {} catch (Exception e) {}
```

5. Obtenga una variable writable de la database si esta no existe todavía.

```
if (mWritableDB == null) {
    mWritableDB = getWritableDatabase();
}
```

6. Inserte la fila.

```
newId = mWritableDB.insert(WORD_LIST_TABLE, null, values);
```

7. Log la exception.

```
Log.d(TAG, "INSERT EXCEPTION! " + e.getMessage());
```

8. Devuelva el id.

```
return newId;
```

Solución:

```
public long insert(String word){
    long newId = 0;
    ContentValues values = new ContentValues();
```

```
    values.put(KEY_WORD, word);
    try {
        if (mWritableDB == null) {
            mWritableDB = getWritableDatabase();
        }
        newId = mWritableDB.insert(WORD_LIST_TABLE, null, values);
    } catch (Exception e) {
        Log.d(TAG, "INSERT EXCEPTION! " + e.getMessage());
    }
    return newId;
}
```

5.2. Obtenga la palabra a insertar por el usuario y actualice la base de datos

El código de inicio viene con una EditWordActivity que obtiene una palabra del usuario y la devuelve a la actividad principal. En MainActivity, sólo tiene que rellenar el método onActivityResult().

1. Compruebe, para asegurarse, de que el resultado es de la actividad correcta y obtenga la palabra que el usuario introdujo de los extras.

```
if (requestCode == WORD_EDIT) {
   if (resultCode == RESULT_OK) {
        String word = data.getStringExtra(EditWordActivity.EXTRA_REPLY);
```

2. Si la palabra no está vacía, compruebe si se ha pasado un id con los extras. Si no hay un id, inserte una nueva palabra. En la siguiente tarea, actualizará la palabra existente si se pasa un id.

```
if (!TextUtils.isEmpty(word)) {
    int id = data.getIntExtra(WordListAdapter.EXTRA_ID, -99);
    if (id == WORD_ADD) {
        mDB.insert(word);
}
```

3. Para actualizar la UI, notifique al adapter que los datso subyacentes han cambiado.

```
mAdapter.notifyDataSetChanged();
```

4. Si la palabra está vacía porque el usuario no ha introducido nada, haga un toast para que lo sepa. Y no olvide cerrar todos los paréntesis y llaves.

```
} else {
    Toast.makeText(
        getApplicationContext(),
        R.string.empty_not_saved,
        Toast.LENGTH_LONG).show();
```

```
        }
    }
}
```

Solución:

```
if (requestCode == WORD_EDIT) {
    if (resultCode == RESULT_OK) {
        String word = data.getStringExtra(EditWordActivity.EXTRA_REPLY);
        // Update the database
        if (!TextUtils.isEmpty(word)) {
            int id = data.getIntExtra(WordListAdapter.EXTRA_ID, -99);
            if (id == WORD_ADD) {
                mDB.insert(word);
            }
            // Update the UI
            mAdapter.notifyDataSetChanged();
        } else {
            Toast.makeText(
                    getApplicationContext(),
                    R.string.empty_not_saved,
                    Toast.LENGTH_LONG).show();
        }
    }
}
```

5.3. Implementar el getItemCount()

Para que los nuevos elementos se muestren correctamente, getItemCount en WordListAdapter tiene que devolver el número real de entradas en la base de datos en lugar del número de palabras en la lista inicial de palabras.

1. Cambie el getItemCount por el código siguiente, el cual generará un error.

```
return (int) mDB.count();
```

2. Abra WordListOpenHelper e implemente count() para que retorne el número de entradas en la base de datos.

```
public long count(){
    if (mReadableDB == null) {
        mReadableDB = getReadableDatabase();
    }
    return DatabaseUtils.queryNumEntries(mReadableDB, WORD_LIST_TABLE);
}
```

3. Ejecute la apicación y agregue algunas palabras.

Tarea 6. Elimine palabras de la base de datos

Paraimplementar la funcionalidad de eliminar necesitará:

- Implementar el método delete() en WordListOpenHelper
- Añadir un manejador de clics al botón DELETE en WordListAdapter

6.1. Escribir el método delete()

Usará el método delete() en SQLiteDatabase para eliminar una entrada en la base de datos.

Agregar un método delete a la WordListOpenHelper que:

1. Cree la plantilla del método delete(), el cual toma un argumento int para el id del elemto a borrar, y devuelve el número de filas borradas.

```
public int delete(int id) {}
```

2. Declare una variable para contener el resultado.

```
int deleted = 0;
```

3. Al igual que en el insert, agregue un bloque try.

```
try {} catch (Exception e) {}
```

4. Obtenga una writable de la base de datos, si fuese necesario.

```
if (mWritableDB == null) {
    mWritableDB = getWritableDatabase();
}
```

5. Llame a delete en el WORD_LIST_TABLE, seleccionando por el KEY_ID y pasando el valor del id como argumento. El "?" es un marcador de posición que se llena con el string. Esta es una forma más segura de crear consultas.

```
deleted = mWritableDB.delete(WORD_LIST_TABLE,
    KEY_ID + " = ? ", new String[]{String.valueOf(id)});
```

6. Imprima un mensaje log para las excepciones.

```
Log.d (TAG, "DELETE EXCEPTION! " + e.getMessage());
```

7. Devuelva e número de filas eliminadas.

```
return deleted;
```

Solución:

```
public int delete(int id) {
    int deleted = 0;
    try {
        if (mWritableDB == null) {
           mWritableDB = getWritableDatabase();
    }
        deleted = mWritableDB.delete(WORD_LIST_TABLE, //table name
                KEY_ID + " =? ", new String[]{String.valueOf(id)});
    } catch (Exception e) {
        Log.d (TAG, "DELETE EXCEPTION! " + e.getMessage());
    }
    return deleted;
}
```

6.2. Agregar un manejador del clic para el botón DELETE

Ahora puede añadir un manejador de clics al botón DELETE el cual llama al método delete() que acaba de escribir.

Eche un vistazo a la clase MyButtonOnClickListener en el código inicial. La clase MyButtonOnClickListener implementa un receptor de clics que almacena el id y la palabra que necesita para hacer cambios en la base de datos.

Cada v.iew holder, cuando se adjunta (bound) al RecyclerView en el método onBindViewHolder de WordListAdapter, también necesita adjuntar un oyente de clic al botón DELETE, pasando el id y el word al constructor MyButtonOnClickListener. Estos valores son utilizados por el manejador onClick para eliminar el elemento correcto y notificar al adaptador qué elemento ha sido eliminado.

Tenga en cuenta que no puede utilizar el argumento `position` pasado a onBindViewHolder, ya que puede estar desactualizado en el momento en que se llama al manejador de clics. Debe guardar una referencia del view holder y obtener la posición con getAdapterPosition().

Solución:

```
// Keep a reference to the view holder for the click listener
final WordViewHolder h = holder; // needs to be final for use in callback

    // Attach a click listener to the DELETE button.
    holder.delete_button.setOnClickListener(
        new MyButtonOnClickListener(current.getId(), null)  {

            @Override
            public void onClick(View v ) {
```

```
                int deleted = mDB.delete(id);
                if (deleted >= 0)
                    notifyItemRemoved(h.getAdapterPosition());
            }
        });
```

Tarea 7. Actualizar palabras en la base de datos

Para actualizar palabras existentes deberá:

- Agregar un método update() a WordListOpenHelper.
- Agregar un manejador de clic al botón EDITAR de su vista.

7.1. Escriba el método update()

Se usa el método update() en SQLiteDatabase para actualizar una entradad existente en la n¿base de datos.

1. Agregar un método a la WordListOpenHelper que:
    - Tome un id entero y un String word somo sus argumentos y retorne un entero.

```
public int update(int id, String word)
```

    - Inicialice int mNumberOfRowsUpdated a -1.

```
int mNumberOfRowsUpdated = -1;
```

2. Dentro del bloque try, haga los siguientes pasos:
3. Obtenga un writable SQLiteDatabase db si aún no existe.

```
if (mWritableDB == null) {
    mWritableDB = getWritableDatabase();
}
```

4. Cree una nueva instancia de ContentValues y agreguele la palabra con KEY_WORD.

```
ContentValues values = new ContentValues();
values.put(KEY_WORD, word);
```

5. Llame a db.update usando los siguientes argumentos:

```
mNumberOfRowsUpdated = db.update(WORD_LIST_TABLE,
       values, // new values to insert
```

```
        // selection criteria for row (the _id column)
        KEY_ID + " = ?",
        //selection args; value of id
        new String[]{String.valueOf(id)});
```

6. En el bloque catch, imprima un mensaje log si cualquier excepción es encontrada.

```
Log.d (TAG, "UPDATE EXCEPTION: " + e.getMessage());
```

7. Devuelva el número de filas actualizadas, las cuales deben ser -1 (falla), 0 (nada actaulizado), o 1 (éxito).

```
return mNumberOfRowsUpdated;
```

Solución:

```
public int update(int id, String word) {
    int mNumberOfRowsUpdated = -1;
    try {
        if (mWritableDB == null) {
            mWritableDB = getWritableDatabase();
    }
        ContentValues values = new ContentValues();
        values.put(KEY_WORD, word);
        mNumberOfRowsUpdated = mWritableDB.update(WORD_LIST_TABLE,
                values,
                KEY_ID + " = ?",
                new String[]{String.valueOf(id)});
    } catch (Exception e) {
        Log.d (TAG, "UPDATE EXCEPTION! " + e.getMessage());
    }
    return mNumberOfRowsUpdated;
}
```

7.2. Agregar un listener al clic para el botón EDIT

Y aquí está el código para el listener del clic Edit cuando se enlaza la View en el método onBindViewHolder de WordListAdapter. Este listener no tiene ninguna base de datos específica. Inicia la Actividad EditWordActivity usando un Intent y le pasa el id, la posición y la palabra actuales en los Extras.

Si obtiene un error en la constante EXTRA_POSITION, se añáde el valor "POSITION",

Solución:

```
// Attach a click listener to the EDIT button.
holder.edit_button.setOnClickListener(new MyButtonOnClickListener(
        current.getId(), current.getWord()) {

    @Override
    public void onClick(View v) {
        Intent intent = new Intent(mContext, EditWordActivity.class);

        intent.putExtra(EXTRA_ID, id);
        intent.putExtra(EXTRA_POSITION, h.getAdapterPosition());
        intent.putExtra(EXTRA_WORD, word);

        // Start an empty edit activity.
        ((Activity) mContext).startActivityForResult(
                intent, MainActivity.WORD_EDIT);
    }
});
```

7.3. Agregar actualización al onActivityResult

Tal como se implementó, al hacer clic en editar se inicia una actividad que muestra al usuario la palabra actual, y éste puede editarla. Para hacer que la actualización se lleve a cabo,

1. Agregue estas líneas de código al método onActivityResult en la MainActivity.

```
else if (id >= 0) {
    mDB.update(id, word);
}
```

2. Ejecute la aplicación y juegue con ella!

7.4. Consideraciones de diseño y error

- Los métodos que escribió para agregar, actualizar y borrar entradas en la base de datos asumen que su entrada es válida. Esto es aceptable para el código de ejemplo porque el propósito de este código de ejemplo es enseñarle la funcionalidad básica de una base de datos SQLite, y por lo tanto no se considera cada caso de borde, no se prueba cada valor, y se asume que todo el mundo se comporta bien. Si se tratara de una aplicación de producción, tendría mayores consideraciones de seguridad y habría que probar la validez del contenido hasta que sepa que no es malicioso.

- En una aplicación de producción, se deben capturar excepciones específicas y manejarlas apropiadamente.

- Ha comprobado el correcto funcionamiento de la aplicación ejecutándola. Para una aplicación de producción con datos reales, necesitará realizar pruebas más exhaustivas, por ejemplo, utilizando pruebas de unidad e interfaz.

- Para esta práctica, se creó el esquema y las tablas de la base de datos desde la clase SQLiteOpenHelper. Esto es suficiente para un ejemplo simple, como este. Para una aplicación más compleja, es mejor separar las definiciones de esquema del resto del código en una clase helper que no pueda ser instanciada. Aprenderá cómo hacerlo en el capítulo sobre content providers.
- Como se mencionó anteriormente, algunas operaciones de la base de datos pueden ser largas y deben realizarse en un hilo de background. Utilice AsyncTask para operaciones que requieren mucho tiempo. Utilice loaders para cargar grandes cantidades de datos.

Resumen

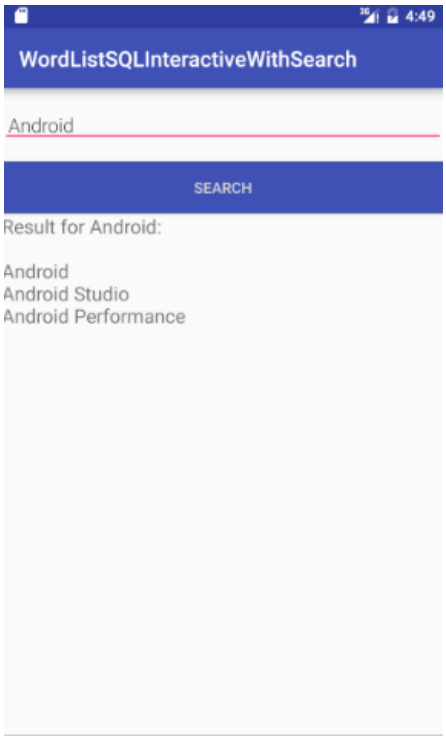
En este capítulo, aprendió como

- Usar una SQLiteDatabase para almacenar datos de usuario de forma persistente.
- Trabajar con un SQLiteOpenHelper para administrar las bases de datos.
- Obtener y mostrar datos desde una base de datos
- Editar datos en la interfaz de usuario y reflejar tales cambios en la base de datos

7.4 Búsqueda de Datos en una Base de Datos SQLite

Resumen de la Aplicación

Usted hará una copia de la aplicación WordListSQLInteractive terminada (o WordListSqlStarterCode si no la renombró; de la práctica anterior), la llamará WordListSQLInteractiveWithSearch, y agregará una actividad que permita a los usuarios buscar palabras parciales y completas en la base de datos. Por ejemplo, al introducir "Android" se mostrarán todas las entradas que contengan la subcadena "Android".



Tarea 0. Descargar y ejecutar el código base

Con el fin de ahorrarle trabajo, esta práctica se basará en una aplicación que ya ha creado. En un entorno de producción, construir sobre código de aplicación existente es una tarea común del desarrollador para añadir características o solucionar problemas.

1. Crear el proyecto

1. Descargue la alicación WordListSQL finished.

   Puede usar su propia aplicación, o descargar la aplicación base. You can use your own app, or download the base app. Siempre y cuando la aplicación utilice una base de datos SQLite, puede utilizar estas instrucciones para ampliarla.

2. Cargue una copia de la aplicación en el Android Studio. Refiérase al Apéndice para información de cómo copiar un proyecto.

3. Renombre el paquete usando Refactor > Rename.

4. Cambie el nombre del paquete en el archivo build.gradle file.

5. Ejecute la aplicación para asegurarse que funciona correctamente.

Tarea 1. Agregar la Búsqueda

1.1. Agregar un Menú de Opciones con un element Search

Utilize el código de OptionsMenuSample de una práctica anterior si necesita un ejemplo de cómo realizar esto

1. En su Proyecto, cree un directorio de Recursos Android y llámelo menu con “menu” como el tipo de recurso (**res > menu**).

2. Agregue un archive de recursos main_menu.xml a res > menu.

3. Cree un menu con un `item` **Search**. Como referencia use el segmento de código para los valores.

```
<menu
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app = "http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   tools:context="com.android.example.wordlistsqlsearchable.MainActivity">

   <item
       android:id="@+id/action_search"
       android:title="Search..."
       android:orderInCategory="1"
       app:showAsAction="never" />
</menu>
```

4. En la MainAcvitiy, infle ele menu sobreescribiendo onCreateOptionsMenu.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
   getMenuInflater().inflate(R.menu.menu_main, menu);
   return true;
}
Override onOptionsItemSelected method. Switch on action_search, and just return
true.
@Override
public boolean onOptionsItemSelected(MenuItem item) {
   switch (item.getItemId()) {
       case R.id.action_search:
           return true;
   }
   return super.onOptionsItemSelected(item);
}
```

5. Ejecute la aplicación. Debe ever los puntos del menu de opciones. Cuando hace clic en él, debería ver un elemento de menú para la búsqueda, sin embargo no hace nada.

1.2. Crear el layout para la activity de búsqueda

Este layout es similar al activity_edit_word, por lo que puede aprovechar el código existente y copiarlo.

1. Cree una copia del activity_edit_word y llámelo activity_search.xml.
2. En activity_search.xml, cambie el id y el string para que sean representativos de la búsqueda.
3. Cambie el método onClick para que el botón showResult.
4. Agregue un TextView con un id de search_result, de al menos 300dp de altura y el tamaño del font en 18sp.
5. Ejecuta la aplicación. No debería notar ninguna diferencia.

1.3. Agregar una Activity para búsqueda

1. Cree un nueva activity, SearchActivity. Si la crea usando **New > Android > Activity** entonces **NO** genere el archive de layout debido a que ya lo creó en la tarea anterior.
2. Agregue una variable miembro private TextView llamada mTextView.
3. Agregur una variable miembro private EditText llamada mEditWordView.
4. Agregur una variable miembro private WordListOpenHelper llamada mDB.
5. En onCreate, inicialice mDB con un new WordListOpenHelper(this).
6. En onCreate, inicialice mTextView y mEditWordView para sus respectivas vistas.

```
public class SearchActivity extends AppCompatActivity {

   private static final String TAG = EditWordActivity.class.getSimpleName();

   private TextView mTextView;
   private EditText mEditWordView;
   private WordListOpenHelper mDB;

   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_search);

       mEditWordView = ((EditText) findViewById(R.id.search_word));
       mTextView = ((TextView) findViewById(R.id.search_result));
       mDB = new WordListOpenHelper(this);
   }
}
```

7. Agregue la activity al AndroidManifest.

```
<activity
    android:name="com.android.example.wordlistsqlsearchable.SearchActivity">
</activity>
```

1.4. Inicie la SearchActivity desde el menu

1. Para iniciar SearchActivity cuando se seleccione el elemento de menú, inserte un código para iniciar SearchActivity en la declaración del switch en el método onOptionSelected() en la MainActivity.

```
Intent intent = new Intent(getBaseContext(), SearchActivity.class);
startActivity(intent);
```

2. Construya y ejecute la aplicación para asegurarse que la SearchActivity es iniciada cuenado el elemento de menú "Search" es seleccionado en el OptionsMenu.
3. Ingrese un string de búsqueda y presione "Search". La aplicación debe fallar.
4. Encuentre por qué la aplicación ha fallado, luego vaya a la siguiente tarea.

1.5. Implementar el manejador de onClick para el botón Search en la SearchActivity

La aplicación ha fallado, debido a que el manejador de onClick para el botón Search en el código XML no existe aún. De manera que construirá showResult a continuación.

Cuando el botón Search es presionado, necesariamente deben suceder varias cosas:

- El manejador de envento llama a `public void showResult(View view)` en SearchActivity.
- La aplicación debe tener el valor actual del `mEditWordView`, el cual es el string de búsqueda.
- Debe imprimir "Result for" y la palabra en el `mTextView`.
- Debe llamar a la función de búsqueda (no escrita aún) en `mDB` (`mDB.search(word)`) y recibir un cursor de base de datos SQLite. Esta función de búsqueda se implementará en la siguiente tarea.
- Procesará el cursor y agregará el resultado al `mTextView`.

1. En la SearchActivity, cree la función showResult. Es `public`, toma un argumento `View` y no retorna nada.
2. Crea una variable `String` llamada `word` e inicialicela con el contenido de la entrada del edit text `mEditWordView`.
3. Muestre el término de búsqueda en el TextView de resultados de búsqueda; algo como esto

```
“Search term: “ + word
```

4. Busque en la base de datos y obtenga el cursor.

```
Cursor cursor = mDB.search(word);
```

5. Para procesar el curso, necesita hacer lo siguiente:
    a. Asegurarse que el cursor no sea null.
    b. Mover el cursor a la primera entrada.
    c. Iteratar sobre el cursor procesando la entrada actual, luego avanzar el cursor.
    d. Extraer la palabra.
    e. Mostrar la palabra en el text view.
6. Cerrar el cursor.
7. Si no se encontraron resultados, el usuario ve una pantalla en blanco sin resultados. Esto debe manejarse en un aplicación en producción.
8. Verifique las anotacione del código para detalles adicionales.

```
    public void showResult(View view){
        String word = mEditWordView.getText().toString();
        mTextView.setText("Result for " + word + ":\n\n");

        // Search for the word in the database.
        Cursor cursor = mDB.search(word);

        // Only process a non-null cursor with rows.
        if (cursor != null & cursor.getCount() > 0) {
            // You must move the cursor to the first item.
            cursor.moveToFirst();
            int index;
            String result;
            // Iterate over the cursor, while there are entries.
            do {
                // Don't guess at the column index.
                // Get the index for the named column.
                index = cursor.getColumnIndex(WordListOpenHelper.KEY_WORD);
                // Get the value from the column for the current cursor.
                result = cursor.getString(index);
                // Add result to what's already in the text view.
                mTextView.append(result + "\n");
            } while (cursor.moveToNext()); // Returns true or false
            cursor.close();
        } // You should add some handling of null case. Right now, nothing
happens.
```

```
    }
```

La aplicación no se ejecutará sin, al menos, una plantilla del método search(). Android Studio creará tal plantilla por usted. En el bombillo rojo, seleccione crear método.

9. Abra WordListOpenHelper.
10. Implemente una plantilla para search, con un parámetro String parameter, que retorne un cursor null.
11. Ejecute la aplicación y corrija cualquier error que pueda tener. Note que la mayoría del código en showResult() no se ha realizado aún.

1.6. Implementar el método search en WordListOpenHelper

El paso final es implementar la búsqueda real de la base de datos.

Dentro del método search(), necesita construir una consulta con el string de búsqueda y enviar la consulta a la base de datos.

Una forma más segura de hacerlo es utilizando parámetros para cada parte de la consulta.

POR QUÉ: En la práctica anterior, para la consulta en WordListOpenHelper, se podía construir el string de consulta directamente y enviarla como rawQuery(), porque se tenía control total sobre el contenido de la consulta. Tan pronto como esté manejando las entradas de usuario, debe asumir que podrían ser maliciosas.

Importante: Por razones de seguridad, ¡siempre debe validar las entradas de usuario antes de crear la consulta!

El query SQL para buscar todas las entradas de la lista de palabras que coincidan con una subcadena tiene esta forma:

```
SELECT * FROM WORD_LIST_TABLE WHERE KEY_WORD LIKE %searchString%;
```

La forma parametrizada del método query que llamará luce asi:

```
Cursor query (String table, // The table to query
              String[] columns, // The columns to return
              String selection, // WHERE statement
              String[] selectionArgs, // Arguments to WHERE
              String groupBy, // Grouping filter. Not used.
              String having, // Additional condition filter. Not used.
              String orderBy) // Ordering. Setting to null uses default.
```

Vea SQLite Database Android y la documentación para varios métodos query().

Para la consulta en el método search(), necesitar asignar solamente los primeros cuatro argumentos.

1. La tabla está ya efinida como la constante WORD_LIST_TABLE.
2. En search(), cree una variable para las columnas. Solamente necesita el valor de la columna KEY_WORD.

```
String[] columns = new String[]{KEY_WORD};
```

3. Agregue el simbolo % al parámetro searchString.

```
searchString = "%" + searchString + "%";
```

4. Cree la clausula su where. Omita “WHERE” ya que es implícita. Use un signo de interrogación para el argumento LIKE. Asegúrese de tener el correcto espaciamiento.

```
String where = KEY_WORD + " LIKE ?";
```

5. Especifique el argumento a la causula where, el cual es searchString.

```
String[] whereArgs = new String[]{searchString};
```

6. Agregue una variable Cursor cursor e inicialicela como null.
7. En un bloque try/catch.
    a. Obtenga una readable database, si mReadable no está definida aún.
    b. Consulte la base de datos usando la forma anterior de query. Coloque en null los parámetros no usados.
    c. Maneje la excepción. Lo puede hacer con un log.
8. Devuelva el cursor.
9. Ejecute la aplicación y haga una búsqueda de algunos strings.

Aqui la solución complete del método:

```
public Cursor search (String searchString) {
   String[] columns = new String[]{KEY_WORD};
   searchString = "%" + searchString + "%";
   String where = KEY_WORD + " LIKE ?";
   String[]whereArgs = new String[]{searchString};

   Cursor cursor = null;

   try {
       if (mReadableDB == null) {mReadableDB = getReadableDatabase();}
```

```
        cursor = mReadableDB.query(WORD_LIST_TABLE, columns, where, whereArgs,
null, null, null);

    } catch (Exception e) {
        Log.d(TAG, "SEARCH EXCEPTION! " + e);
    }

    return cursor;
}
```

Resumen

- Un menu de opciones puede ser una efectiva UI para realizer búsquedas en una base de datos SQLite
- Una activity separada para manejar la UX de búsqueda puede ayudar a obtener el foco del usuario
- En una aplicación en producción, las consultas a SQLite deben ser manejadas con mucho cuidado para evitar corrupción de los datos o problemas de seguridad
- Las búsquedas SQLite pueden ser construidas dinámicamente usando las entrdas del usuario como parámetros del query.
- El método query() busca en una base de datos palabras coincidentes.
- El método query() devuelve un cursor de la base de datos el cual puede recorrer el result set de forma eficiente
- EL cursor puede ser usado par amostrar los resultados al usuario.

7.5 Implementando un Content Provider Minimalista

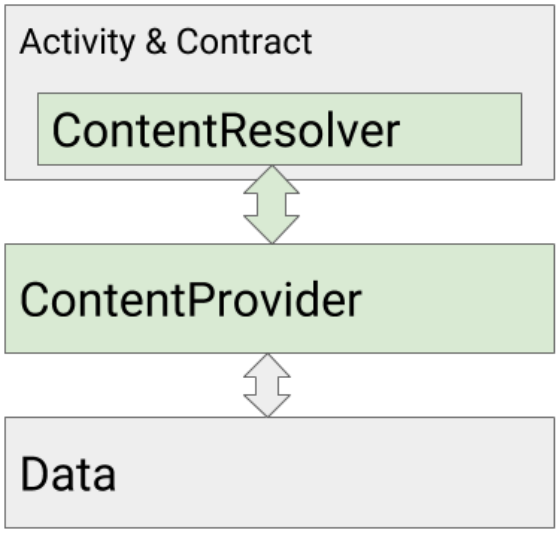
Un content provider (proveedor de contenido) es un componente que gestiona de forma segura el acceso a un repositorio compartido de datos. Proporciona una interfaz coherente para que las aplicaciones accedan a los datos compartidos. Las aplicaciones no acceden directamente al proveedor, sino que utilizan un objeto content resolver el cual proporciona una interfaz para ello y gestiona la conexión con el content provider.

Los proveedores de contenido son útiles porque:

- Las aplicaciones no pueden compartir datos en Android, excepto a través de content providers.
- Los content providers permiten que múltiples aplicaciones accedan, usen y modifiquen de forma segura una única fuente de datos. Ejemplos: Contactos, resultados de los juegos, diccionario de corrección ortográfica.
- Se puede especificar niveles de control de acceso (permisos) para el content provider.
- Se puede almacenar datos independientemente de la aplicación. Tener un content provider le permite cambiar la forma en que se almacenan los datos sin necesidad de cambiar la interfaz de usuario. Por ejemplo, puede construir un prototipo utilizando datos ficticios y, a continuación, utilizar una base de datos SQL para la aplicación real. Incluso podría almacenar algunos de sus datos en la nube y algunos datos localmente, y la interfaz de usuario seguiría siendo la misma para su aplicación.
- Esta arquitectura separa los datos de la interfaz de usuario de manera que los equipos de desarrollo puedan trabajar de forma independiente en la aplicación orientada al cliente y en los componentes del lado del servidor de su aplicación. Para aplicaciones más grandes y complejas es muy común que la interfaz de usuario y los servicios de datos sean desarrollados por diferentes equipos. Incluso pueden ser aplicaciones separadas. Ni siquiera es necesario que una aplicación con un content provider tenga una interfaz de usuario.
- Se puede utilizar un CursorLoader y otras clases que esperan interactuar con un content provider.

Nota: Si la aplicación no comparte datos con otras aplicaciones, entonces la aplicación no requiere de un content provider. Sin embargo, debido a que el content provider separa limpiamente la implementación del backend de la interfaz de usuario, también puede ser útil para la arquitectura de aplicaciones más complejas.

El siguiente diagrama resume las partes de la arquitectura del proveedor de contenido.



Data: La aplicación que crea el content provider es la propietaria de los datos y especifica qué permisos tienen otras aplicaciones para trabajar con los datos.

Los datos se almacenan a menudo en una base de datos SQLite, pero esto no es obligatorio. Normalmente, los datos se ponen a disposición del content provider en forma de tablas, similares a las tablas de base de datos, en las que cada fila representa una entrada y cada columna representa un atributo para esa entrada. Por ejemplo, cada fila en una base de datos de contactos contiene una entrada y esa entrada puede tener columnas para direcciones de correo electrónico y números de teléfono.

ContentProvider: El content provider proporciona una interfaz pública y segura para los datos, de modo que otras aplicaciones puedan acceder a los datos con los permisos apropiados.

ContentResolver: Utilizado por la Actividad para consultar al content provider. El content resolver devuelve los datos como un objeto Cursor que puede ser usado, por ejemplo, por un adapter, para mostrar los datos.

Contract Class (Clase de contrato, no se muestra): El contrato es una clase pública que expone información importante sobre el content provider a otras aplicaciones. Esto usualmente incluye los URIs para acceder a los datos, constantes importantes y la estructura de los datos que serán devueltos.

Las aplicaciones envían solicitudes al content provider utilizando un URI (Uniform Resource Identifiers or URIs). Un content URI para el content provider tiene esta forma general:

scheme://authority/path-to-data/dataset-name

- scheme (para el content URI, esto siempre es `content://`)
- authority (representa el dominio, y para los content providers habitualmente termina en `.provider`)
- path (representa la ruta a los datos)

- ID (identifica de manera única el conjunto de datos a buscar; tal como el nombre de un archivo o el nombre de una tabla)

El siguiente URI podría ser usado para requerir todas las entradas en la tabla “words”:

```
content://com.android.example.wordcontentprovider.provider/words
```

El diseño de esquemas URI es un tema en sí mismo y no está cubierto en este curso.

Content Resolver: El objeto ContentResolver proporciona los métodos query(), insert(), update(), y delete() para acceder a los datos de un content provider y maneja toda la interacción con el content provider por el desarrollador. En la mayoría de los casos, se puede utilizar el content resolver predeterminado proporcionado por el sistema Android.

En esta práctica, usted construirá un content provider básico desde cero. Creará y procesará datos simulados para que pueda centrarse en la comprensión de la arquitectura del content provider. Asimismo, la interfaz de usuario para visualizar los datos es mínima. En la siguiente práctica, usted agregará un content provider a la aplicación WordList, usando esta aplicación minimalista como su plantilla.

Resumen de la Application

- Esta aplicación genera datos simulados y los almacena en una lista enlazada llamada "words".
- La aplicación solicita datos a través de un content resolver y los muestra. La interfaz de usuario consiste en una actividad con una vista de texto y dos botones. El botón "Lista all words" muestra todas las palabras, y el botón "List first word" muestra la primera palabra en la vista de texto.
- El content provider abstrae y gestiona la interacción entre la fuente de datos y la interfaz del usuario.
- El Contract define URIs y constantes públicas.

Nota: La versión mínima del SDK es API15: Android 4.0.3 IceCreamSandwich y el *target* SDK es la versión actual de Android (versión 27 al momento de recopilar este libro).

Tarea 1. Crear el proyecto MinimalistContentProvider

1.1. Crear un Proyecto con las restricciones suministradas

Cree una aplicación con una actividad que muestre una vista de texto y dos botones. Un botón muestra la primera palabra de nuestros datos (la lista), y el otro botón mostrará todas las palabras. Ambos botones llaman a onClickDisplayEntries() cuando se hace clic en ellos. Por ahora, este método usará un switch-case para mostrar una sentencia que indica que se ha hecho clic en un botón en particular. Utilice la siguiente tabla como guía para configurar el proyecto.

App name	MinimalistContentProvider
One Activity	Empty Activity template Name: MainActivity private static final String TAG = MainActivity.class.getSimpleName(); public void onClickDisplayEntries (View view){Log.d (TAG, "Yay, I was clicked!");}
TextView	@+id/textview android:text="response"
Button	@+id/button_display_all android:text="List all words" android:onClick="onClickDisplayEntries"
Button	@+id/button_display_first android:text="List first word" android:onClick="onClickDisplayEntries"

1.2. Completar la configuración básica

Complete la configuración básica de la interfaz de usuario:

1. En la MainActivity, cree una variable member para el text view e inicialicela en el método onCreate().
2. En el onClickDisplayEntries(), use una sentencia switch-case para verificar que botón fue presionado. Use el id del view para distinguir los botones. Imprima una sentencia log para cada caso.
3. En el onClickDisplayEntries(), al final añada algo de texto al textview.
4. Como siempre, extraiga los recursos de string.
5. Ejecute la aplicación.

Su MainActivity debería ser similar a esta solución.

Solución:

```
package android.example.com.minimalistcontentprovider;

[... imports]

public class MainActivity extends AppCompatActivity {

   private static final String TAG = MainActivity.class.getSimpleName();

   private TextView mTextView;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mTextView = (TextView) findViewById(R.id.textview);
    }

    public void onClickDisplayEntries(View view) {
        Log.d (TAG, "Yay, I was clicked!");

        switch (view.getId()) {
            case R.id.button_display_all:
                Log.d (TAG, "Yay, " + R.id.button_display_all + " was clicked!");
                break;
            case R.id.button_display_first:
                Log.d (TAG, "Yay, " + R.id.button_display_first + " was clicked!");
                break;
            default:
                Log.d (TAG, "Error. This should never happen.");
        }
        mTextView.append("Thus we go! \n");
    }
}
```

Tarea 2. Crear una clase Contract, un URI y datos simulados

El contract contiene información acerca de los datos que las aplicaciones necesitan para construer los queries.

- Contract es una clase pública e incluye información importante para otras aplicaciones que quieran conectarse a este content provider y acceder a los datos.
- El URI muestra cómo construir URIs para acceder a los datos. El esquema URI se comporta como una API para acceder a los datos. Es muy similar al diseño de llamadas REST para CRUD. Otras aplicaciones utilizarán estas Content URIs.

2.1. Crear la clase Contract

1. Cree una nueva clase public **Java Contract** con las siguiente firma. Además debe ser final.

```
public final class Contract {}
```

2. Para evitar que alguien instancie accidentalmente la clase Contrato, dale un constructor privado vacío.

```
private Contract() {}
```

2.2. Crear el URI

Un content URI para los content providers tiene esta forma general:

```
scheme://authority/path/id
```

- **scheme** siempre es `content://` para los content URIs.
- **authority** representa el dominio y para los content providers habitualmente termina en `.provider`
- **path** es la ruta a los datos
- **id** identifica de forma única los grupos de datos a buscar

El siguiente URI podría ser usado para requerir todas las entradas en la tabla “words”:

```
content://com.android.example.wordcontentprovider.provider/words
```

La URI para acceder al content provider se define en el Contract de manera que esté disponible para cualquier aplicación que desee consultar a este content provider. Normalmente, esto se hace definiendo constantes para AUTHORITY, CONTENT_PATH y CONTENT_URI.

1. En la clase Contract, cree una constante para AUTHORITY. Para que Authority sea única, utilice el nombre del paquete extendido con "provider".

   ```
   public static final String AUTHORITY =
                   “com.android.example.minimalistcontentprovider.provider";
   ```

2. Cree una constante para CONTENT_PATH. La ruta del content identifica los datos. Debe utilizar algo descriptivo, por ejemplo, el nombre de una tabla o archivo, o el tipo de datos, como "words".

   ```
   public static final String CONTENT_PATH = "words";
   ```

3. Cree una constante para CONTENT_URI. Es en el estilo `content:// del` URI que apunta a un conjunto de datos. Si tiene múltiples "contenedores de datos" en el backend, debería crear un content URI para cada uno.

   Uri es una clase helper para construir y manipular URIs. Puesto que es un string que nunca cambia para todas las instancias de la clase Contract, puede inicializarla estáticamente.

   ```
   public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY +
   "/" + CONTENT_PATH);
   ```

4. Cree una constante de conveniencia para ALL_ITEMS. Este es el nombre del data set que utilizará al recuperar todas las palabras. El valor es -2 debido a que es el primer valor más bajo no devuelto por una llamada al método.

```
static final int ALL_ITEMS = -2;
```

5. Cree una constante de conveniencia para WORD_ID. Este es el id que usará al recuperar una única palabra.

```
static final String WORD_ID = "id";
```

2.3. Agregar el tipo MIME

Los content providers proporcionan contenido, y es necesario especificar qué tipo de contenido proporcionan. Las aplicaciones necesitan conocer la estructura y el formato de los datos devueltos, para poder manejarlos correctamente.

Los tipos MIME son de la forma tipo/subtipo, como text/html para páginas HTML. Para un content provider, se debe definir un tipo MIME específico por proveedor para el tipo de datos que son devueltos por el content provider. El tipo de MIME específicos por proveedor de Android es siempre:

- `vnd.android.cursor.item` para un elemento de dato (un registro)
- `vnd.android.cursor.dir` para un grupo de datos (múltiple registros).

El subtipo puede ser cualquier cosa, pero es una buena práctica hacerlo informativo. Por Ejemplo:

- vnd—un tipo MIME de un proveedor
- com.example—the domain
- provider—esto es para un content provider
- words—el nombre de la tabla

Leer Implementing ContentProvider MIME types para detalles.

1. Declare el tipo MIME para un elemento de dato (un registro).

```
static final String SINGLE_RECORD_MIME_TYPE =
                    "vnd.android.cursor.item/vnd.com.example.provider.words";
```

2. Declare el tipo MIME para múltiples registros.

```
static final String MULTIPLE_RECORD_MIME_TYPE =
                     "vnd.android.cursor.dir/vnd.com.example.provider.words";
```

2.4. Crear los datos simulados

El content provider siempre presenta los resultados como un Cursor en un formato de tabla que se asemeja a una base de datos SQL. Esto es independiente de cómo se almacenan realmente los datos. Esta aplicación utiliza una serie de palabras.

En strings.xml, agregue una lista corta de palabras:

```
<string-array name="words">
   <item>Android</item>
   <item>Activity</item>
   <item>ContentProvider</item>
   <item>ContentResolver</item>
</string-array>
```

Tarea 3. Implementar la clase MiniContentProvider

3.1. Crear la clase MiniContentProvider

1. Cree una **clase Java** MiniContentProvider extendiendo a ContentProvider. (Para esta práctica, **no** use la opción del menú Create Class > Other > Content Provider.)
2. Implemente los métodos (**Code > Implement methods**).
3. Agregue una etiqueta log.
4. Añada una variable miembro para los datos simulados.

```
public String[] mData;
```

5. En onCreate(), inicialice `mData` con el arreglo de palabras y retorne `true`.

```
@Override
public boolean onCreate() {
   Context context = getContext();
   mData = context.getResources().getStringArray(R.array.words);
   return true;
}
```

6. Agregue un mensaje de logging apropiado a los métodos insert, delete, y update methods. Para esta práctica, estos métodos no serán implementados.

```
Log.e(TAG, "Not implemented: update uri: " + uri.toString());
```

3.2. Publique el content provider agregándolo al Android manifest

Para acceder al content provider, esta aplicación y otras aplicaciones necesitan saber que él existe. Añada una declaración para el content provider al Android manifest dentro de una etiqueta <provider>.

La declaración contiene el nombre del content provider y las authorities (su identificador único).

1. En el AndroidManifest, dentro de la etiqueta application, después de la etiqueta de cierre de activity, agregue:

```
<provider
    android:name=".MiniContentProvider"
android:authorities="com.android.example.minimalistcontentprovider.provider" />
```

2. Ejecute su código para asegurarse que compila.

3.3. Configurar la coincidencia de URI

Un ContentProvider necesita responder a las solicitudes de datos hechas por las aplicaciones utilizando una serie de URI diferentes. Para tomar la acción apropiada dependiendo de un particular request URI, el proveedor de contenido necesita analizar la URI para ver si coincide. UriMatcher es una clase helper que se puede utilizar para procesar los esquemas URI aceptados para un content provider dado.

Pasos básicos para usar UriMatcher:

- Crear una instancia de UriMatcher.
- Añadir al UriMatcher cada URI que el content provider reconozca.
- Asignar a cada URI una constante numérica. Tener una constante numérica para cada URI es conveniente cuando se está procesando URIs entrates debido a que se puede usar una sentencia switch/case sobre esos valores numéricos para trabajar a través de los URIs.

Realice los siguientes cambios en la clase MiniContentProvider.

1. En la clase MiniContentProvider, cree una variavle private static variable para un nuevo UriMatcher.

   El argumento del constructor especifica el valor a devolver si no hay coincidencia. Como mejor práctica, use UriMatcher.NO_MATCH.

```
private static UriMatcher sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
```

2. Cree su propio método para inicializar el URI matcher.

```
Private void initializeUriMatching(){}
```

3. Llame al initializeUriMatching en el onCreate() de la clase MiniContentProvider.

4. En el método initializeUriMatching(), agregue al matcher las URIs que el content provider acepta y asígnele un código entero. Estos son los URIs basados en las authorities y las rutas de contenido especificadas en el contrato.

   El símbolo # hace coincidir un string de caracteres numéricos a cualquier longitud. En esta aplicación, se refiere al índice de la palabra en arreglo de string. En una aplicación en producción, este podría ser el id de una entrada en una base de datos. Asigne a esta URI un valor numérico de 1.

```
sUriMatcher.addURI(Contract.AUTHORITY, Contract.CONTENT_PATH + "/#", 1);
```

5. El segundo URI es el que especificó en el contrato para devolver todos los artículos. Asígnele un valor numérico de 0

```
sUriMatcher.addURI(Contract.AUTHORITY, Contract.CONTENT_PATH, 0);
```

Note que si la aplicación es más compleja y usa mas URIs, use constantes nombradas para los códigos, como se muestra en UriMatcher documentation.

Solución:

```
private void initializeUriMatching(){
   sUriMatcher.addURI(Contract.AUTHORITY, Contract.CONTENT_PATH + "/#", 1);
   sUriMatcher.addURI(Contract.AUTHORITY, Contract.CONTENT_PATH, 0);
}
```

3.4. Implementar el método getType()

El método getType() del content provider devuelve el tipo MIME de cada uno de los URIs especificados.

A menos que esté haciendo algo especial en el código, esta implementación de método va a ser muy similar para cualquier content provider. El hace lo siguiente:

1. Hace coincidir el URI.
2. Activa el código retornado.
3. Devuelve el tipo MIME apropiado.

Aprenda más en UriMatcher documentation.

Solución:

```
public String getType(Uri uri) {
```

```
        switch (sUriMatcher.match(uri)) {
            case 0:
                return Contract.MULTIPLE_RECORD_MIME_TYPE;
            case 1:
                return Contract.SINGLE_RECORD_MIME_TYPE;
            default:
                // Alternatively, throw an exception.
                return null;
        }
    }
```

3.5 Implementar el método query()

El propósito del método query() es hacer coincidir el URI, convertirlo al mecanismo interno de acceso a datos (por ejemplo, una consulta SQlite), ejecutar el código interno de acceso a datos y devolver el resultado en un objeto Cursor .

El método query()

El método query tiene la siguiente firma:

```
public Cursor query(Uri uri, String[] projection, String selection,String[]
selectionArgs, String sortOrder){}
```

Los argumentos de este método representan las partes de una consulta SQL. Incluso si se está utilizando otro tipo de mecanismo de almacenamiento de datos, se debe aceptar una consulta con este estilo y manejar los argumentos adecuadamente. (En la siguiente tarea construirá una consulta en la MainActivity para ver cómo se usan los argumentos.) El método devuelve un Cursor de cualquier tipo.

uri	El URI completo. Este no puede ser null.
projection	Indica a cuáles columnas/atributos acceder
selection	Indica a cuáles filas/registros del objeto acceder
selectionArgs	Los parámetros enlazados al argumento de selección anterior. Por razones de seguridad, los argumentos se procesan por separado.
sortOrder	Si están ordenados, y si es así, si es ascendente o descendente . Si esto es nulo, se aplica la clasificación por defecto o no se aplica ninguna clasificación.

Analize el método query()

1. Identificar los siguientes pasos de procesamiento en el código del método query() mostrados la sección solución.

El procesamiento de la consulta consiste de estos pasos:

a. Hacer coincidir el URI.

b. Activar el código retornado.

c. Procesar los argumentos y construer una consulta apropiada para el backend.

d. Obtener los datos y (si es necesario) dejarlos en el Cursor.

e. Devolver el cursor.

2. Identificar porciones de código que necesariamente deben ser diferentes en una aplicación en producción.

   La implementación de la consulta para una aplicación básica toma algunos atajos.

   - El manejo de errores es mínima.
   - Debido a que la aplicación está usando datos simulados, el Cursor puede ser populado directamente.
   - Debido a que el esquema UR es simple, este método es relativamente corto.

3. Identificar al menos una decision de diseño que facilite la comprensión y mantenimiento del código.

   - Analizar la consulta y ejecutarla para popular un cursor están separados en dos métodos.
   - El código contiene más comentarios que código ejecutable.

4. Agegar el código a la aplicación.

Nota: Obtendrá un error en el método populateCursor(), estó será atendido en el siguiente paso.

Código de la solución con Anotaciones para el método query() en MiniContentProvider.java

```
@Nullable
@Override
public Cursor query(Uri uri, String[] projection, String selection, String[]
selectionArgs, String sortOrder) {
   int id = -1;
   switch (sUriMatcher.match(uri)) {
       case 0:
           // Matches URI to get all of the entries.
           id = Contract.ALL_ITEMS;

           // Look at the remaining arguments
           // to see whether there are constraints.
           // In this example, we only support getting
           //a specific entry by id. Not full search.
```

```
            // For a real-life app, you need error-catching code;
            // here we assume that the
            // value we need is actually in selectionArgs and valid.
            if (selection != null){
                id = parseInt(selectionArgs[0]);
            }
            break;

        case 1:
            // The URI ends in a numeric value, which represents an id.
            // Parse the URI to extract the value of the last,
            // numeric part of the path,
            // and set the id to that value.
            id = parseInt(uri.getLastPathSegment());
            // With a database, you would then use this value and
            // the path to build a query.
            break;

        case UriMatcher.NO_MATCH:
            // You should do some error handling here.
            Log.d(TAG, "NO MATCH FOR THIS URI IN SCHEME.");
            id = -1;
            break;
        default:

            // You should do some error handling here.
            Log.d(TAG, "INVALID URI - URI NOT RECOGNIZED.");
            id = -1;
    }
    Log.d(TAG, "query: " + id);
    return populateCursor(id);
}
```

3.6. Implementar el método populateCursor()

Una vez que el método query() ha identificado el URI, llama al populateCursor() con el último segmento de la ruta, el cual es el id (índice) de la palabra a recuperar. El método populateCursor() separa el query coincidente de la obtención de los datos y la creación del cursor resultante. Esta es una buena práctica ya que en una aplicación real, el método query() puede llegar a ser muy grande.

El método query debe devolver un tipo Cursor, por lo que el método populateCursor() tiene que crear, rellenar y devolver un cursor.

- SI los datos están almacenados en una base de datos SQLite, la ejecución del query debe retornar un Cursor.
- SI no se está usando un método e almcenamiento de datos que retorne un cursor, tal como archivos o datos simuados, se puede usar un MatrixCursor para mantener los datos a devolver. Un MatrixCursor es un cursor de propósitos generales dentro de un arreglo de

objetos que crece según se necesite. Para crear un MatrixCursor, debe sumiistrarlo con un arreglo de string con los nombres de las columnas.

El método populateCursor() hace lo siguiente:

1. Recibe el id extraido del URI.
2. Crea un MatrixCursor para almacenar los datos recibidos (debido a que los datos simulados no están en un cursor).
3. Crea y ejecuta un query. Para esta aplicación, esto obtiene el string que se encuentra en el id de index del arreglo de string. En una aplicación más realista, esto debería ejecutar una consulta a la base de datos.
4. Agrega el resultado al cursor.
5. Devuelve el cursor.

```
private Cursor populateCursor(int id) {
   MatrixCursor cursor = new MatrixCursor(new String[] { Contract.CONTENT_PATH
});
   // If there is a valid query, execute it and add the result to the cursor.
   if (id == Contract.ALL_ITEMS) {
       for (int i = 0; i < mData.length; i++) {
           String word = mData[i];
           cursor.addRow(new Object[]{word});
       }
   } else if (id >= 0) {
       // Execute the query to get the requested word.
       String word = mData[id];
       // Add the result to the cursor.
       cursor.addRow(new Object[]{word});
   }
   return cursor;
}
```

Tarea 4. Usar un ContentResolver para obtener los datos

Con el content provider en su lugar, el método onClickDisplayEntries() en la MainActivity puede expandirse para consultar y mostrar los datos en la UI. Esto requiere los siguientes pasos:

1. Crear la consulta en estilo SQL, dependiendo de que botón se haya presionado.
2. Usar un content resolver para interactuar con el content provider para ejecutar el query y devolver un Cursor.
3. Procesar los resultados en el Cursor.

4.1. Obtener el content resolver

El content resolver interactúa con el content provider por usted.

El content resolver espera un Content URI analizado en conjunto con los paramétros del query que asiste para la obtención de los datos

No tiene que crear su propio content resolver. Puede usar el es proporcionado por el framework Android en el contexto de la aplicación llamando a getContentResolver().

1. En la MainActivity, elimine todo el código dentro de onClickDisplayEntries().
2. Añada este código al onClickDisplayEntries() en la MainActivity.

```
Cursor cursor = getContentResolver().query(Uri.parse(queryUri), projection,
selectionClause, selectionArgs, sortOrder);
```

Nota: los argumentos del getContentResolver.query() son identicos a los parámetros del ContentProvider.query().

A continuación debe definer los argumentos del getContentResolver.query().

4.2. Definir los argumentos de la consulta

Para que el getContentResolver.query() trabaje, se necesita declarer y asignar valores a todos sus argumentos.

1. **URI:** Declara el ContentURI que define el content provider y la tabla. Obtiene la información para el URI correcto desde el contracto.

```
String queryUri = Contract.CONTENT_URI.toString();
```

2. **Projection:** Un arreglo de string con los nombres de las columnas a devolver. Si se coloca en null, devuelve todas las columnas. Cuando sólo hay una columna, como en el caso de este ejemplo, la parametrización explícita es opcional, pero puede ser útil a efectos de documentación.

```
// Only get words.
String[] projection = new String[]{Contract.CONTENT_PATH};
```

3. **selectionClause:** Cláusula del argumento para los criterios de selección, es decir, las líneas a devolver. Formateado como una cláusula SQL WHERE (excluyendo la palabra clave "WHERE"). Pasar null devuelve todas las filas para la URI dada. Ya que esto variará dependiendo de qué botón fue presionado, declarelo ahora y configúrelo más tarde.

```
String selectionClause;
```

4. **selectionArgs:** Valores de argumentos para los criterios de selección. Si incluye `?s` en el string de selección, son reemplazados por valores del selectionArgs, en el orden en que aparecen.

IMPORTANTE: Es una buena práctica de seguridad siempre separar la selección y los selectionArgs.

```
String selectionArgs[];
```

5. **sortOrder:** La manera en el que se ordenan los resultados. Formateado como una cláusula SQL ORDER BY (excluyendo la palabra clave ORDER BY). Usualmente ASC o DESC; null solicita la ordenación predeterminada, la cual podría ser desordenado.

```
// For this example, accept the order returned by the response.
String sortOrder = null;
```

4.3. Decidir sobre el criterio de selection

Los valores de selectionClause y selectionArgs dependen del botón que se haya pulsado en la UI.

- Para mostrar todas las palabras, establezca ambos argumentos en null.
- Para obtener la primera palabra, busque la palabra con el ID de 0. (Esto asume que los ID de palabra comienzan en 0 y se crean en orden. Usted lo sabe, porque la información está expuesta en el contrato. Para un proveedor de contenido diferente, puede que no conozca los ids, y puede que tenga que buscar de una manera diferente).

1. Sustituya el bloque switch existente por el siguiente código en onClickDisplayEntries, antes de obtener el content resolver.

```
switch (view.getId()) {
   case R.id.button_display_all:
       selectionClause = null;
       selectionArgs = null;
       break;
   case R.id.button_display_first:
       selectionClause = Contract.WORD_ID + " = ?";
       selectionArgs = new String[] {"0"};
       break;
   default:
       selectionClause = null;
       selectionArgs = null;
}
```

4.4. Procesar the Cursor

Después de obtener el content resolver, debe procesar el resultado del Cursor.

- Si hay datos, mostrarlos en el text view.
- Si no hay datos, comunicar los errores.

1. Examine el siguiente código y asegúrese que lo entiende completamente.

```
if (cursor != null) {
   if (cursor.getCount() > 0) {
       cursor.moveToFirst();
       int columnIndex = cursor.getColumnIndex(projection[0]);
       do {
           String word = cursor.getString(columnIndex);
           mTextView.append(word + "\n");
       } while (cursor.moveToNext());
   } else {
       Log.d(TAG, "onClickDisplayEntries " + "No data returned.");
       mTextView.append("No data returned." + "\n");
   }
   cursor.close();
} else {
   Log.d(TAG, "onClickDisplayEntries " + "Cursor is null.");
   mTextView.append("Cursor is null." + "\n");
}
```

2. Inserte este código al final de onClickDisplayEntry().

3. Ejecute la aplicación.

4. Haga clic en los botones para ver los datos obtenidos en el text view.

Resumen

EN este capítulo ha aprendido

- Los content providers son abstracciones de datos de alto nivel que gestionan el acceso a un repositorio compartido.
- Los content providers están destinados principalmente a ser utilizados por aplicaciones que no sean las suyas.
- Los content providers (lado del servidor) se les accede a través de los content resolvers (lado de la aplicación).
- Un Contrat es una clase pública que expone información importante sobre un content provider.
- Los contratos pueden ser útiles más allá de los content providers,
- Un content provider necesita definir un conjunto de content URIs para que las aplicaciones puedan acceder a los datos a través del content provider.
- El content URI consta de varios componentes: "`content://`", una content authority única (normalmente un nombre de paquete totalmente cualificado) y la ruta del content.
- Utilizar un content resolver para solicitar datos a un content provider y mostrarlos al usuario.

- Si la aplicación no comparte datos con otras aplicaciones, entonces la aplicación no requiere de un content provider.
- Los content provider deben implementar el método getType() que devuelve el tipo MIME por cada tipo del content.
- Los content providers deben estar "publicados" en el Android manifest utilizando la etiqueta `<provider>`
- Los content provider deben inspeccionar la URI entrante para determinar las coincidencias de patrones del URI con el fin de acceder a cualquier dato.
- Debe agregar patrones URI de destino a su content provider. La clase UriMatcher es una clase útil para este propósito.
- La esencia de un content provider se implementa en el método query().
- La firma del método query() en un content resolver (solicitante de datos) debe coincidir con la firma del método query() en un content provider (fuente de datos).
- El método query() devuelve un objeto cursor con un estilo de base de datos independientemente de si los datos provienen de una relacional o no.

7.6 Agregar un Content Provider a su Base de Datos

Los content providers en aplicaciones del mundo real son más complejos que la versión básica que se construyó en la práctica anterior.

En el mundo real:

- El backend es una base de datos, un Sistema de archivos u otra opción de persistencia.
- El front-end muestra los datos en una UI agradable y permite a los usuarios manipularlos.

Raramente construirá una aplicación desde cero. Más a menudo depurará, refactorizará o extenderá una aplicación existente.

En esta práctica, usted tomará la aplicación WordListSQL la refactorizará y la extenderá para usar un content provider como una capa entre la base de datos SQL y el RecyclerView.

Esto se puede hacer de dos maneras.

- Refactorizar y extender la aplicación the WordListSQL. Esto involucra cambiar la arquitectura de la aplicación y refactorizar el código.
- Comenzar desde cero y reusar el código de WordListSQL y la MinimalistContentProvider.

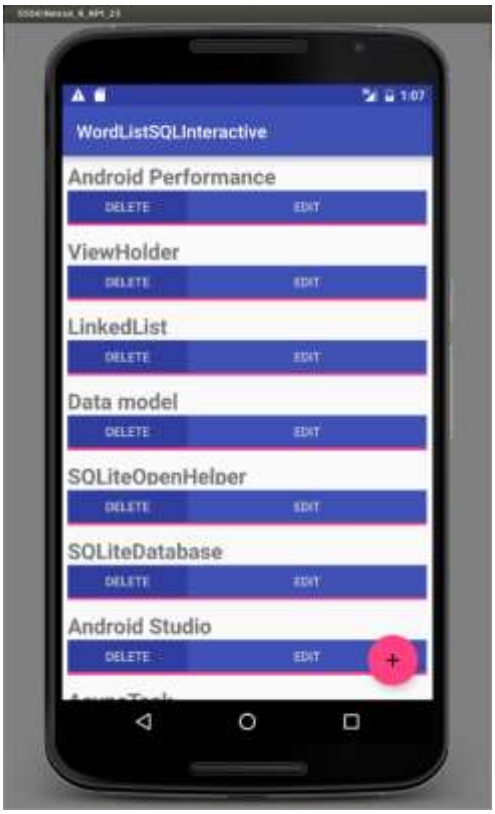
Esta practica demostrará como refactorizar la aplicación existente WordListSQL, debido a que se asemeja más a lo que encontrará en su trabajo.

Resumen de la aplicación

La aplicación completada WordListSQLWithContentProvider tendrá las siguientes catacterísticas:

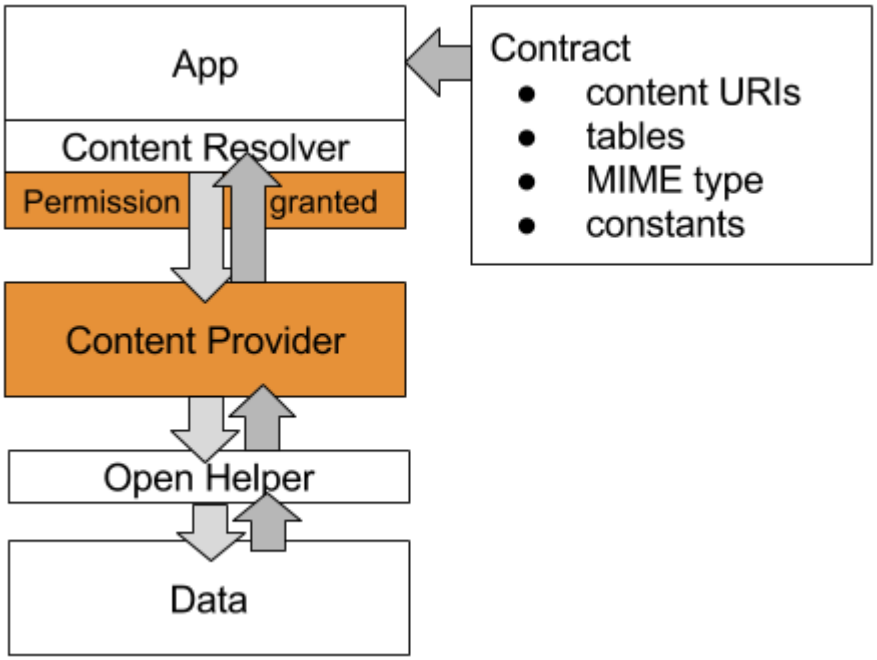
- Un content provider que pueda insertar, eliminar, actualizar y consultar la base de datos.
- Un contrato y permisos configurados que permitan a otras aplicaciones tener acceso a este content provider.
- Un content resolver que interactúe con el content provider to insertar, eliminar, actualizar y consultar datos.
- Una interfaz de usuario y funcionalidad sin cambios.

La aplicación lucirá igual que la el final de la práctica de datos almacenados.

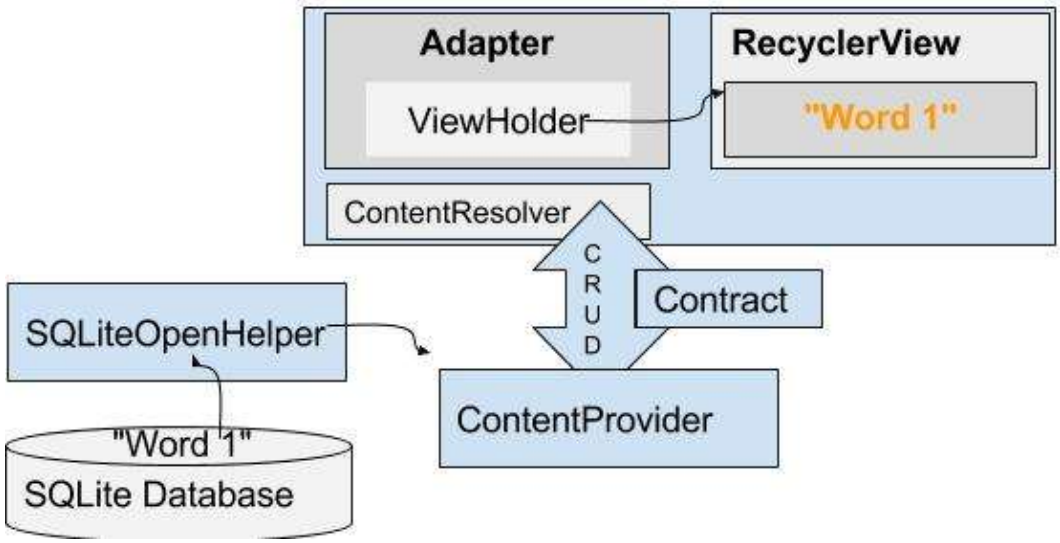


Resumen de componentes de la aplicación

El siguiente diagrama muestra una visión general de los componentes de una aplicación que utiliza una base de datos SQLiteDatabase con un content provider. La única diferencia con la aplicación MinimalistContentProvider es que el content provider obtiene los datos de una base de datos a través de un open helper.



El siguiente diagrama muestra la arquitectura de la aplicación WorldListSQLInteractive con un content provider añadido; ésta es la aplicación WordListSQLWithContentProvider que construirá en esta práctica.



Resumen de cambios

Este es un resumen de los cambios que realizará a WordListInteractive para agregar un content provider.

- Nuevas Clases: Contract, ContentProvider, ContentResolver
- Clases que cambian: WordListOpenHelper, MainActivity, WordListAdapter
- Clases que no deberían cambiar: WordItem, MyButtonOnClickListener, ViewHolder

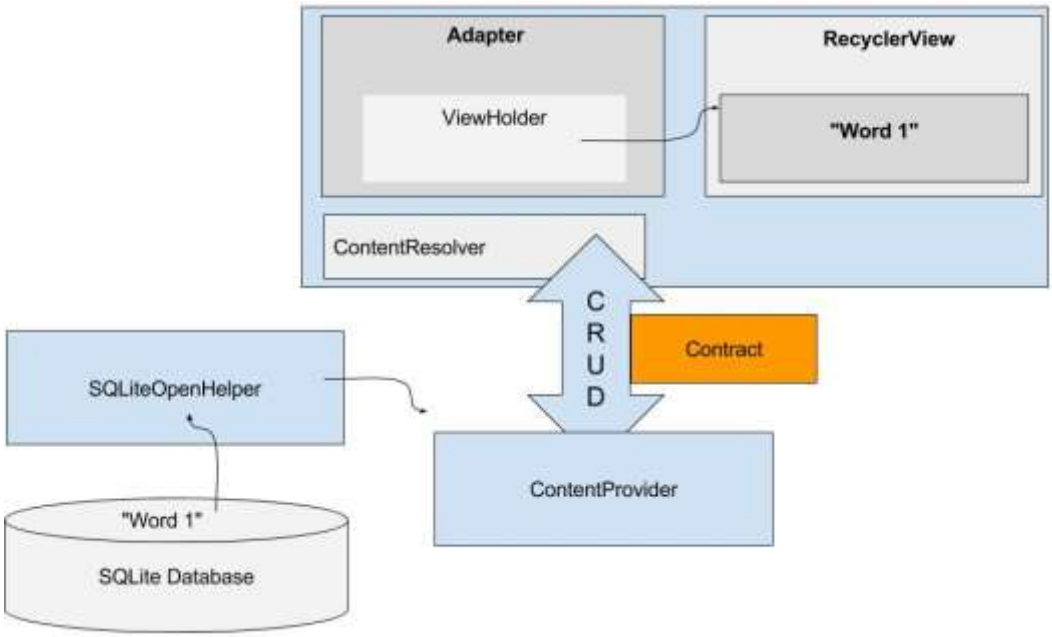
Tarea 1. Descargar y ejecutar el código base

Esta práctica se basa en las aplicaciones WordListSQLInteractive y MinimalistContentProvider que creó anteriormente. Extenderá una copia de WordListSQLInteractive. Puede empezar desde su propio código o descargar las aplicaciones.

- WordListSQLInteractive
- MinimalistContentProvider
- Haga una copia de la WordListSQLInteractive y carguela en el Android Studio.
- Cambie el nombre del paquete a wordlistsqlwithcontentprovider.

Tarea 2. Añadir un clase Contract a la WordListSQLInteractive

Comenzará creando una clase Contract que define unas constantes públicas database, constantes URI y los tipos MIME. Estas constantes serán usadas en otras clases



2.1 Añadir una clase Contract

1. Estudie la siguiente documentación [Define a Schema and Contract documentation](#).
2. Agregue una nueva clase public y final al proyecto y llámela Contract.

   Esta clase Contract contiene toda la información que otra aplicación necesita para usar el content provider de la suya. Puede nombrar la clase como desee, pero habitualmente es llamada "Contract".

```
public final class Contract {}
```

3. Para prevenir que la clase Contract sea instanciada, añada un constructor vacío private.

   Este es un patron estándar para clase que son usadas para mantener meta información y constantes para una aplicación.

```
private Contract() {}
```

2.2 Mueva las constantes database dentro de Contract

Mueva las constantes de la base de datos que otra aplicación necesitaría conocer fuera de la WordListOpenHelper dentro de la contract y hágalas public.

1. Mueva DATABASE_NAME y hacerla public.

```
public static final String DATABASE_NAME = "wordlist";
```

Crear una clase interna abstracta estática para cada tabla con los nombres de las columnas. Esta clase interna implementa comúnmente la interface BaseColumns . Al implementar la interface BaseColumns, la clase puede heredar un campo clave primario llamado _ID que algunas clases de Android, como los adaptadores de cursor, esperan que exista. Estas clases internas no son requeridas, pero pueden ayudar a que su base de datos funcione bien con el framework Android.

2. Cree una clase interna WordList que implemente a BaseColumns.

```
public static abstract class WordList implements BaseColumns {
}
```

3. Mueva el nombre de tabla `WORD_LIST_TABLE`, así como los nombres de columna `KEY_ID` y `KEY_WORD` desde WordListOpenHelper a la clase WordList en Contract y hágalas public.

4. Retorne a la WorldListOpenHelper y espere que Android Studio importe las constantes desde la Contract; o importelas manualmente, si no lo tiene configurado para auto-imports.

   Use **File > Settings > Editor > General > Auto Import** en Windows/Linux o **Android Studio > Preferences >Editor >General > Auto Import** en Mac para configurar los imports automáticos.

2.3 Definir las Constantes URI

1. Declare el esquema URI para el content provider.

   Usando la Contract en MinimalistContentProvider como un ejemplo, declare AUTHORITY, CONTENT_PATH. Añada CONTENT_PATH_URI para que devuelva todos los elementos y ROW_COUNT_URI la cual retorna el número de entradas. En la AUTHORITY, use el nombre de su aplicación.

```
public static final int ALL_ITEMS = -2;
public static final String COUNT = "count";

public static final String AUTHORITY =
       "com.android.example.wordlistsqlwithcontentprovider.provider";

public static final String CONTENT_PATH = "words";

public static final Uri CONTENT_URI =
       Uri.parse("content://" + AUTHORITY + "/" + CONTENT_PATH);
public static final Uri ROW_COUNT_URI =
       Uri.parse("content://" + AUTHORITY + "/" + CONTENT_PATH + "/" +
COUNT);
```

2.4 Declarar los tipos MIME

Los tipos MIME describen el tipo y formato de los datos. Los tipos MIME son usados para procesar los datos apropiadamente. Tipos MIME comunes incluyen a text/html para páginas web y

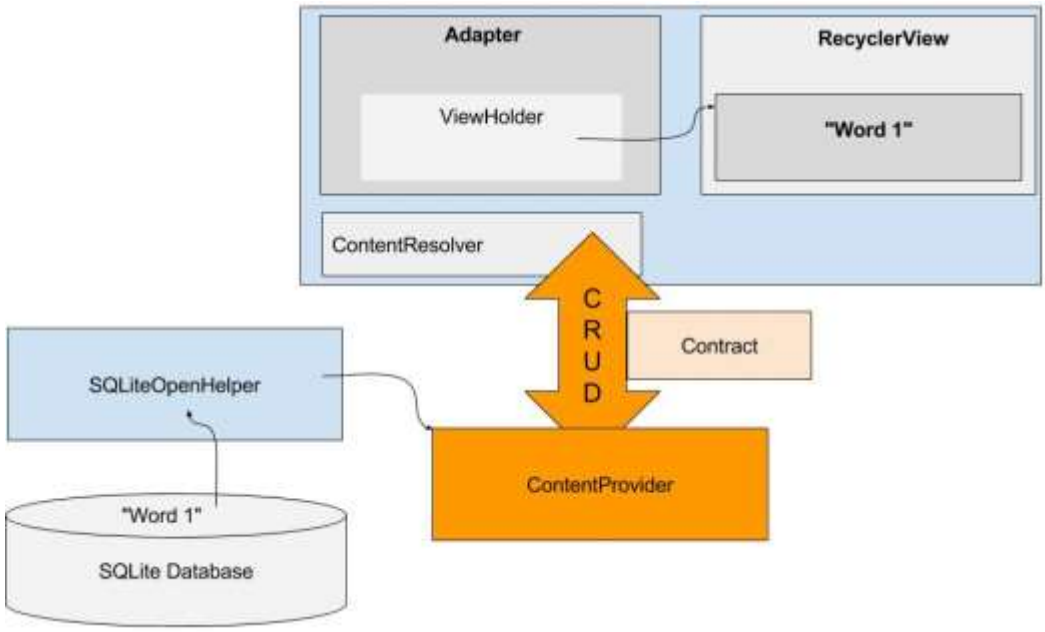
application/json. Lea más acerca de MIME types for content providers en la documentación Android.

1. Declare los tipos MIME types para una respuesta de uno y múltiples registros:

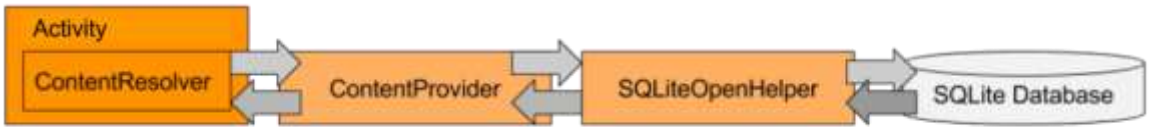
```
static final String SINGLE_RECORD_MIME_TYPE =
   "vnd.android.cursor.item/vnd.com.example.provider.words";
static final String MULTIPLE_RECORDS_MIME_TYPE =
   "vnd.android.cursor.item/vnd.com.example.provider.words";
```

2. Ejecute la aplicación. Debe correr y lucir exactamente igual como antes de cambiarla.

Tarea 3. Crear un Content Provider



En esta tarea creará un content provider, implementará su método query y lo conectará con WordListAdapter y WordListOpenHelper. En lugar de consultar al Helper de WordListOpen, el WordListAdapter usará un content resolver para consultar al content provider, que a su vez consultará al WordListOpenHelper que consultará la base de datos.



3.1 Crear una clase WordListContentProvider

1. Cree una nueva clase que extienda a ContentProvider y llámela WordListContentProvider.
2. En Android Studio, haga clic en el bombillo rojo, seleccione "Implement methods", y haga clic en **OK** para implementar los métodos listados.

3. Specifique un TAG para log.
4. Declare un UriMatcher.

   Este content provider usa un UriMatcher, una clase de utilidad que mapea los URIs a números, de manera que pueda hacer switch en ellos.

```
private static UriMatcher sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
```

5. Declare una variable miembro WordListOpenHelper, mDB.

```
private WordListOpenHelper mDB;
```

6. Declare los códigos para el URI matcher como constantes.

   Esto coloca los códigos en un solo lugar y hace que sean fáciles de cambiar. Utilice decenas, de modo que insertar códigos adicionales sea sencillo.

```
private static final int URI_ALL_ITEMS_CODE = 10;
private static final int URI_ONE_ITEM_CODE = 20;
private static final int URI_COUNT_CODE = 30;
```

7. Cambie el método onCreate() para
   - initialice mDB con un WordListOpenHelper,
   - llame el método initializeUriMatching() que creará a continuación,
   - y retorne true.
   - @Override

```
public boolean onCreate() {
   mDB = new WordListOpenHelper(getContext());
   initializeUriMatching();
   return true;
}
```

8. Cree un método private void llamado initializeUriMatching().
9. En initializeUriMatching(), añada las URIs al matcher para obtener todos los elementos, un elemento, y la cuenta.

   Refierase al Contract y use el método initializeUriMatching() en la aplicación MinimalistContentProver como una plantilla.

Solución:

```
private void initializeUriMatching(){
   sUriMatcher.addURI(Contract.AUTHORITY, Contract.CONTENT_PATH,
                      URI_ALL_ITEMS_CODE);
   sUriMatcher.addURI(Contract.AUTHORITY, Contract.CONTENT_PATH + "/#",
                      URI_ONE_ITEM_CODE);
   sUriMatcher.addURI(Contract.AUTHORITY, Contract.CONTENT_PATH + "/" +
                      Contract.COUNT, URI_COUNT_CODE );
}
```

3.2 Implementar WordListContentProvider.query()

Use el MiniContentProvider como una plantilla para implementar el método query().

1. Modifique el WordListContentProvider.query().
2. Use una sentencia Switch para los códigos devueltos por el sUriMatcher.
3. Para URI_ALL_ITEMS_CODE, URI_ONE_ITEM_CODE, URI_COUNT_CODE, llame al correspondiente en WordListOpenHelper (mDB).

Observe cómo la asignación de los resultados de mDB.query() a un cursor genera un error, porque WordListOpenHelper.query() devuelve un WordItem.

Observe cómo la asignación de los resultados de mDB.count() a un cursor genera un error, porque WordListOpenHelper.count() devuelve un long.

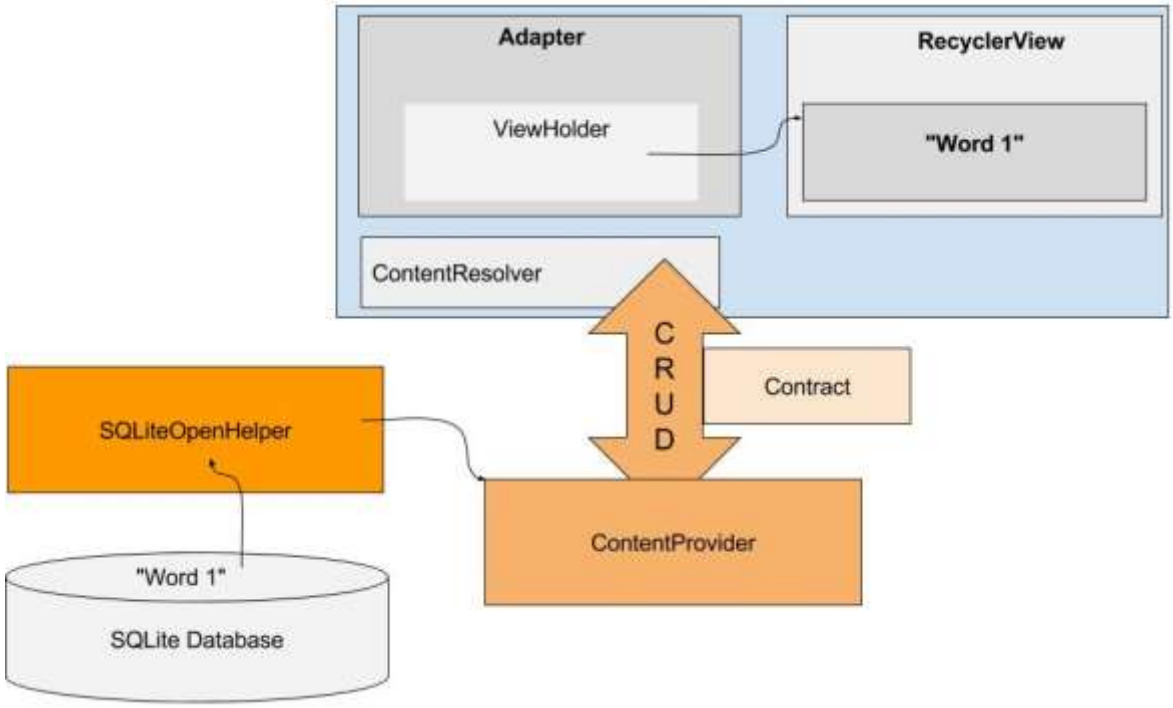
Más continuación repará esos errores.

Solución:

```
@Nullable
@Override
public Cursor query(Uri uri, String[] projection, String selection,
                    String[] selectionArgs, String sortOrder) {

   Cursor cursor = null;

   switch (sUriMatcher.match(uri)) {
       case URI_ALL_ITEMS_CODE:
           cursor = mDB.query(ALL_ITEMS);
           break;

       case URI_ONE_ITEM_CODE:
           cursor = mDB.query(parseInt(uri.getLastPathSegment()));
           break;

       case URI_COUNT_CODE:
           cursor = mDB.count();
           break;

       case UriMatcher.NO_MATCH:
```

```
            // You should do some error handling here.
            Log.d(TAG, "NO MATCH FOR THIS URI IN SCHEME: " + uri);
            break;
        default:
            // You should do some error handling here.
            Log.d(TAG, "INVALID URI - URI NOT RECOGNIZED: "  + uri);
    }
    return cursor;
}
```

3.3 Reparar WordListOpenHelper.query() para que devuelva un Cursor y administre el retornar todos los elementos



Dado que el content provider trabaja con cursores, puede simplificar el método WordListOpenHelper.query() para que devuelva un cursor.

1. Añada el código con un query para devolver todos los elemetos de la base de datos para manejar el `cursor = mDB.query(ALL_ITEMS)` desde la sentencia switch anterior.

2. Simplifique WordListOpenHelper.query() para que devuelva un cursor.

   Esto repara el error en WordListContentProvider.query().

   Sin embargo, esto hace fallar a WordListAdapter.OnBindViewHolder(), el cual espera un WordItem desde WordListOpenHelper.

Para resolver esto, WordListAdapter.onBindViewHolder() necesita usar un content resolver en lugar de llamar a la base de datos directamente, lo que hará después de arreglar WordListContentProvider.count().

Nota: Este tipo de errores en cascada y correcciones es típico para trabajar con aplicaciones de la vida real. Si una aplicación con la que estás trabajando está bien diseñada, puedes corregir los errores uno por uno.

Solución:

```
/**
 * Queries the database for an entry at a given position.
 *
 * @param position The Nth row in the table.
 * @return a WordItem with the requested database entry.
 */
public Cursor query(int position) {
    String query;
    if (position != ALL_ITEMS) {
        position++; // Because database starts counting at 1.
        query = "SELECT " + KEY_ID + "," + KEY_WORD + " FROM "
                + WORD_LIST_TABLE
                +" WHERE " + KEY_ID + "=" + position + ";";
    } else {
        query = "SELECT  * FROM " + WORD_LIST_TABLE
                + " ORDER BY " + KEY_WORD + " ASC ";
    }

    Cursor cursor = null;
    try {
        if (mReadableDB == null) {
            mReadableDB = this.getReadableDatabase();
        }
        cursor = mReadableDB.rawQuery(query, null);
    } catch (Exception e) {
        Log.d(TAG, "QUERY EXCEPTION! " + e);
    } finally {
        return cursor;
    }
}
```

3.4 Reparar WordListOpenHelper.count() para que retorne un Cursor

Dado que el content provider o trabaja con cursores, también debe cambiar el método WordListOpenHelper.count() para que devuelva un cursor.

Utilice un MatrixCursor, que es un cursor de filas y columnas variables.

1. Cree un MatrixCursor usando Contract.CONTENT_PATH.
2. Dentro de un bloque try, obtenga la cuenta y agrégela como una fila al cursor.

3. Devuelva el cursor.

Solución:

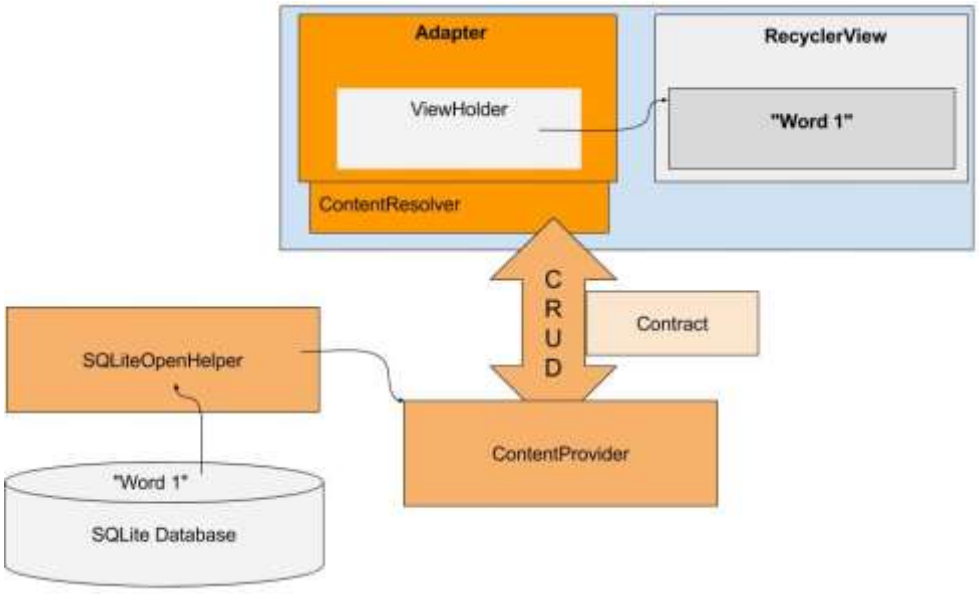
```
public Cursor count(){
   MatrixCursor cursor = new MatrixCursor(new String[] {Contract.CONTENT_PATH});
   try {
       if (mReadableDB == null) {
           mReadableDB = getReadableDatabase();
   }
       int count =  (int) DatabaseUtils.queryNumEntries(mReadableDB,
                                                        WORD_LIST_TABLE);
       cursor.addRow(new Object[]{count});
   } catch (Exception e) {
       Log.d(TAG, "EXCEPTION " + e);
   }
   return cursor;
}
```

Esto repara los errores en WordListContentProvider.count(), pero hace fallar al WordListAdapter.getItemCount(), el cual espera un long desde WordListOpenHelper.

En WordListAdapter.onBindViewHolder(), en vez de llamar directamente a la base de datos, deberá usar content resolvers, lo cual hará a continuación.

3.5 Reparar WordListAdapter.onBindViewHolder() para usar un content resolver

A continuación, reparará WordListAdapter.onBindViewHolder() para usar un content resolver en vez de llamar directamente a WordListOpenHelper.



1. En WordListAdapter, borre la variable mDB variable, debido a que no va a referenciar más la base de datos. Esto mostrará errores en el Android Studio lo cual lo guiará para los siguientes cambios.

2. En el constructor, elimine la asignación a mDB.

3. Refactor > Change la firma del constructor y remueva el parámetro db.

4. Añada variables de instancias para los parámetros del query debido a que los usará más de una vez.

   El content resolver toma un parámetro query, el cual debe construir. El query es similarmente estructurado a una consulta SQL, pero en vez de una selección, usa un URI. Los parámetros query som muy similares a los queries SQL.

```
private String queryUri = Contract.CONTENT_URI.toString(); // base uri
private static final String[] projection = new String[]
{Contract.CONTENT_PATH}; //table
private String selectionClause = null;
private String selectionArgs[] = null;
private String sortOrder = "ASC";
```

5. En onBindViewholder(), elimine las dos primeras líneas de codigo.

   - ~~WordItem current - mDB.query(position)~~;
   - ~~holder.wordItemView.setText(current.getWord())~~;

6. Defina una variable String vacía y llámela word.

7. Defina una variable entera llamada id y asignele -1.

8. Cree un content resolver con los parámetros query especificados y almacene los resultados en un Cursor llamado cursor. (Vea la MainActivity de la aplicación MinimalistContentProvider para un ejemplo.)

```
String word = "";
int id = -1;

Cursor cursor = mContext.getContentResolver().query(Uri.parse(
                    queryUri), null, null, null, sortOrder);
```

9. En vez de sólo obtener un WordItem enviado, WordListAdapter.onBindViewHolder() tiene que hacer el trabajo extra de extraer la palabra desde el cursor devuelto por el content resolver.

   - Si el cursor devuelto contiene datos, extraiga la palabra y configure el texto del view holder.
   - Extraiga el id, debido a que lo necesitará para los listeners de clic.

- Cierre el cursor. Recuerde que no cerró el cursor en WordListOpenHelper.query(), debido a que lo retornó.
- Maneje el caso en que no haya datos en el cursor.
- Implemente cualquier recurso de string referenciado.

```
if (cursor != null) {
      if (cursor.moveToPosition(position)) {
            int indexWord =
cursor.getColumnIndex(Contract.WordList.KEY_WORD);
            word = cursor.getString(indexWord);
            holder.wordItemView.setText(word);
            int indexId = cursor.getColumnIndex(Contract.WordList.KEY_ID);
            id = cursor.getInt(indexId);
       } else {
            holder.wordItemView.setText(R.string.error_no_word);
       }

       cursor.close();

} else {
       Log.e (TAG, "onBindViewHolder: Cursor is null.");
}
```

10. Repare los parámetros de los listeners de clic de los dos botones:
    - current.getId() ⇒ id
    - current.getWord() ⇒ word

    El listener de clic actualizado para el botón DELETE luce así:

```
@Override
public void onClick(View v) {
selectionArgs = new String[]{Integer.toString(id)};
int deleted = mContext.getContentResolver().delete(
    Contract.CONTENT_URI, Contract.CONTENT_PATH,selectionArgs);
 if (deleted > 0) {
     // Need both calls
     notifyItemRemoved(h.getAdapterPosition());
     notifyItemRangeChanged(
             h.getAdapterPosition(), getItemCount());
 } else {
      Log.d (TAG, mContext.getString(R.string.not_deleted) + deleted);
 }
}
```

11. Reemplace la llamada a mDB.delete(id) en el callback del botón DELETE button con una llamada al content resolver call para eliminar.

```
selectionArgs = new String[]{Integer.toString(id)};
int deleted = mContext.getContentResolver().delete(
               Contract.CONTENT_URI, Contract.CONTENT_PATH, selectionArgs);
```

3.6 Cambie WordListAdapter.getItemCount() para que use un content resolver

En lugar de solicitar la cuenta desde la base de datos, getItemCount() tiene que conectarse al content resolver y solicitar la cuenta. En el Contract, se definió un URI para obtener esta cuenta:

```
public static final String COUNT = "count";
public static final Uri ROW_COUNT_URI =
       Uri.parse("content://" + AUTHORITY + "/" + CONTENT_PATH + "/" + COUNT
```

Cambie WordListAdaptergetItemCount() para:

- Usar un query content resolver para obtener el element count
- Use la ROW_COUNT_URI en su query
- La count es un tipo entero y es el primer elementodel Cursor devuelto
- Extraiga count desde el cursor y devuelvalo
- De otra manera devuelva -1
- Cierre el cursor

Use el código que recién escribio para onBindViewHolder como una guía.

Solución:

```
@Override
public int getItemCount() {
    Cursor cursor = mContext.getContentResolver().query(
                    Contract.ROW_COUNT_URI, new String[] {"count(*) AS count"},
                    selectionClause, selectionArgs, sortOrder);
     try {
         cursor.moveToFirst();
         int count = cursor.getInt(0);
         cursor.close();
         return count;
     } catch (Exception e){
         Log.d(TAG, "EXCEPTION getItemCount: " + e);
         return -1;
     }
```

```
}
```

3.7 Añadir el content provider al Android Manifest

1. Ejecute la aplicación.
2. Examine el logcat para las causas (muy comunes) de error.
3. Añada el content provider al Android Manifest dentro de la etiqueta <application>.

```
<provider
   android:name=".WordListContentProvider"
android:authorities="com.android.example.wordlistsqlwithcontentprovider.prov
ider">
</provider>
```

4. Ejecute la aplicación.

Su aplicación debería funcionar y ser completamente funcional. Si no es así, compare su código con el código de la solución suministrado y utilice el debugger (depurador) y el loggin para encontrar el problema.

3.8 ¿Qué sigue?

- Ya ha implementado un content provider y su método query query().
- Ha seguido los errorespara actaulizar los métodos en las clases WordListOpenHelper y WordListAdapter paratrabajar con el content provider.
- Cuando ejecuta la aplicación, para consultas, la llamada al método va a través del content provider.
- Para operaciones de insertar, eliminar y actualizar, la aplicaión aún sigue llamando al WordListOpenHelper.

Con la infraestructura que ha construido, implementar los métodos remanentes será un trabajo mucho menor.

Tarea 4. Implementar los métodos del Content Provider

4.1 getType()

El método getType() es llamado por otras aplicaciones que quieren usar este content provider, para descubrir qué tipo de datos devuelve la aplicación.

Utilice una instrucción switch para devolver los tipos MIME apropiados.

- Los tipos MIME están listados en el contract.
- SINGLE_RECORD_MIME_TYPE es parar URI_ALL_ITEMS_CODE

- MULTIPLE_RECORDS_MIME_TYPE es para URI_ONE_ITEM_CODE

Solución:

```
@Nullable
@Override
public String getType(Uri uri) {
   switch (sUriMatcher.match(uri)) {
       case URI_ALL_ITEMS_CODE:
           return MULTIPLE_RECORDS_MIME_TYPE;
       case URI_ONE_ITEM_CODE:
           return SINGLE_RECORD_MIME_TYPE;
       default:
           return null;
   }
}
```

4.2 Llamar el content provider para insertar insertar y actualizar palabras en MainActivity

Para arreglar las operaciones de inserción en MainActivity().onActivityResult necesita llamar al content provider en lugar de a la base de datos para insertar y actualizar palabras.

1. En MainActivity, elimine las declaraciones de mDB y su instanciación.

En OnActivityResult()

Insertar:

1. Si la longitud de la palabra es diferente de cero, cree una variable ContentValues llamada "values" y añada la palabra introducida por el usuario usando la cadena "word" como clave.

2. Reemplazar mDB.insert(word); con una petición de inserción al content resolver

Actualizar:

1. Reemplace mDB.update(id, word); con una petición de actualización al content resolver.

Trozo de Solución:

```
// Update the database
if (word.length() != 0) {
   ContentValues values = new ContentValues();
   values.put(Contract.WordList.KEY_WORD, word);
   int id = data.getIntExtra(WordListAdapter.EXTRA_ID, -99);

   if (id == WORD_ADD) {
      getContentResolver().insert(Contract.CONTENT_URI, values);
   } else if (id >=0) {
       String[] selectionArgs = {Integer.toString(id)};
```

```
       getContentResolver().update(Contract.CONTENT_URI, values,
Contract.WordList.KEY_ID, selectionArgs
);
   }
   // Update the UI
   mAdapter.notifyDataSetChanged();
```

4.3 Implementar insert() en el content provider

El método insert() en el content provider es muy sencillo. Haga lo siguiente

1. Llame al método insert() del OpenHelper insert(),
2. convierta el id long devuelto en un content URI del elemento insertado,
3. y devuelva ese URI.

Android Studio reporta un error para el parámetro `values`, el cual reparará en siguientes pasos.

Solución:

```
public Uri insert(Uri uri, ContentValues values) {
   long id = mDB.insert(values);
   return Uri.parse(CONTENT_URI + "/" + id);
}
```

4.4 Reparar el insert() en WordListOpenHelper

Android Studio reporta un error para el parámetro `values`.

1. Abra el WordListOpenHelper. El método insert() está escrito para que tome un parámetro String.
2. Cambie el parámetro para que sea de tipo ContentValues.
3. Elimine la declaración y asignación de values en el cuerpo del método.

4.5 Implementar update() en el content provider

Repare el método update de la misma manera que hizo para el método insert.

1. En el WordListContentProvider, implemente update(), lo cual es una línea de código que pasa el id y la word como argumentos.

```
return mDB.update(parseInt(selectionArgs[0]),
values.getAsString(Contract.WordList.KEY_WORD));
```

2. No necesita hacer ningún cambio para actualizar en el WordListOpenHelper.

4.6 Implementar delete() en el content provider

En WordListContentProvider, implemente el método delete() llamando al método delete() en WordListOpenHelper con el id de la palabra a borrar.

```
return mDB.delete(parseInt(selectionArgs[0]));
```

4.7 Ejecute la app

Listo. Esto es todo. Ejecute la aplicación y asegúrese que trabaja.

Y si la aplicación sigue sin funcionar, debería corregir cualquier problema. Necesitará el código de trabajo en una práctica posterior. En esa lección escribirá una aplicación que utiliza este content provider para cargar los datos de la lista de palabras en su interfaz de usuario.

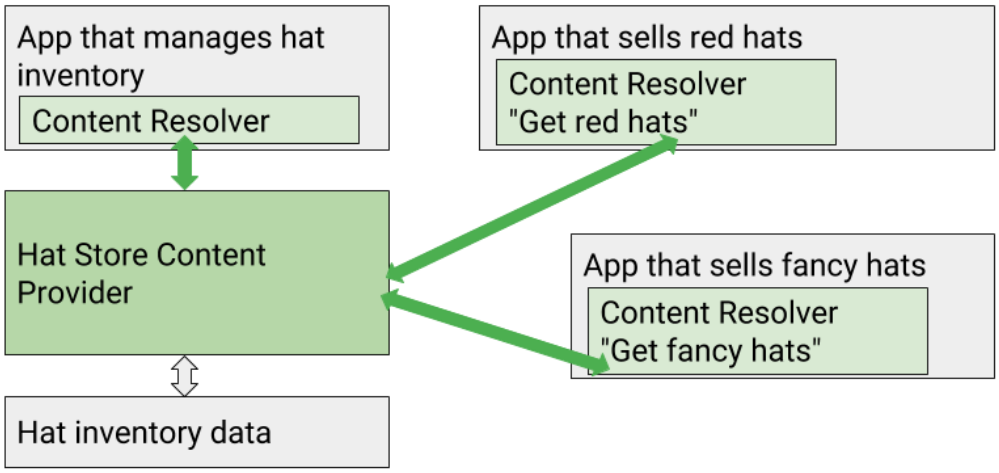
Resumen

- En producción, la mayoría de los desarrolladores de aplicaciones suelen refactorizar las aplicaciones para dar cabida a un content provider.
- Durante la refactorización, los desarrolladores suelen experimentar cambios en cascada y errores.
- Necesita separar la UI de la base de datos utilizando un content provider y un content resolver.
- La UI no debe cambiar durante la refactorización de una base de datos incrustada a una fuente de datos externa.
- La clase Contract define las constantes comunes para todos los componentes de la aplicación reformada.
- La clase Contract localiza todas las constantes comunes para facilitar el mantenimiento.
- Al refactorizar, es muy útil tener diagramas de las clases de acceso a la base de datos.
- Se debe prestar especial atención al diseño de las URI de acceso a la base de datos que otras aplicaciones necesitan para acceder a los datos.
- Todos los accesos a su base de datos deben ser cambiados para usar un Content Resolver en lugar de acceder directamente a una clase helper (por ejemplo: WordListOpenHelper).
- Si los datos subyacentes han cambiado, es importante indicar a la UI que se actualice utilizando notifyDataSetChanged().

7.7 Compartiendo Contenido con Otras Aplicaciones

Para proteger los datos de aplicaciones y usuarios, las aplicaciones no pueden compartir datos con otras aplicaciones directamente. Sin embargo, las aplicaciones pueden poner los datos a disposición de otras aplicaciones utilizando un content provider. Las aplicaciones cliente pueden utilizar un content resolver para acceder a los datos a través de la interfaz pública del content provider.

El siguiente diagrama muestra cómo un mayorista de sombreros puede utilizar a un content provider para compartir información sobre su inventario con aplicaciones que venden sombreros.

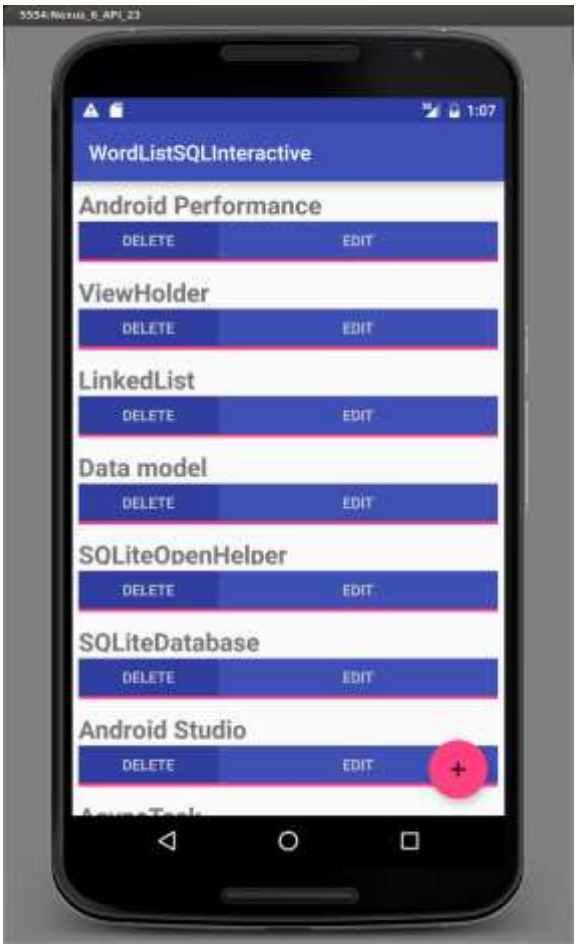


En esta práctica modificará la WordListSQLWithContentProvider para permitir que otras aplicaciones accedan a los datos de su content provider. Luego creará una segunda aplicación, WordListClient, que no tiene datos propios, sino que obtiene datos del content provider de WordListSQLWithContentProvider.

Resumen de la Aplicación

Se usarán dos aplicaciones en esta práctica.

- La aplicación existente WordListSQLWithContentProvider que construyó en la práctica anterior.
- Una aplicación nueva llamada WordListClient que consultará el content provider de WordListSQLWithContentProvider. La UI para esta aplicación será igual a la de WordListInteractive.



Tarea 1. Hacer que el content provider esté disponible para otras aplicaciones

De forma predeterminada, las aplicaciones no pueden tener acceso a los datos de otra saplicaciones.

Para que el content provider esté disponible para otras aplicaciones, debe especificar los permisos en el AndroidManifest de la aplicación. Esto es cierto para cualquier aplicación que tenga un content provider. Cada content provider necesita los permisos especificados en su AndroidManifest.

Los permisos no son cubiertos en detalle en estas prácticas. Si desea puede ampliar la información en Implementing Content Provider Permissions.

1.1. Modificar la WordListWithContentProvider para permitir acceso a otras aplicaciones

1. Abra la WordListSQLWithContentProvider en el Android Studio.
2. Abra el archivo AndroidManifest.xml.
3. Añada una sentencia export dentro de <provider>.

```
android:exported="true"
```

4. En el nivel superior, dentro de la etiqueta <manifest> añada un permiso apar el content provider.

   Es una buena práctica usar el nombre único del paquete para mantener el permiso único.

```
<permission
android:name="com.android.example.wordlistsqlwithcontentprovider.PERMISSION"
/>
```

5. Ejecute la aplicación para asegurarse que no tiene errores, y dejela instalada en el dispositivo.

   Para que otra aplicación pueda acceder al content provider de WordListWithContentProvider, la aplicación con el content provider tiene que estar instalada en el dispositivo. No es necesario que esté funcionando.

Ahora tiene un content provider en su dispositivo al que otra aplicación puede acceder. A continuación, va a crear una nueva aplicación, WordListClient, que obtiene palabras del content provider y las muestra.

1.2. Crear la aplicación WordListClient

En lugar de crear una aplicación cliente desde cero, creará WordListClient a partir de una copia de WordListSQLWithContentProvider. Conservará la interfaz de usuario y el adapter para mostrar los datos. Eliminará el content provider y la base de datos y, en su lugar, obtendrá datos del content provider de WordListSQLWithContentProvider.

1. Cree una copia de la **carpeta** WordListSQLWithContentProvider y renombrela como WordListClient.
2. Abra la aplicación copiada en la carpeta WordListClient en el Android Studio.
3. Renombre el paquete (Refactor > Rename) como wordlistclient.
4. Abra el archivo build.gradle(Module:app) y cambie el id de la aplicación por wordlistclient.
5. En strings.xml, cambie el nombre de la aplicación a WordListClient.
6. En el Android Manifest de WordListClient, remueva la declaración <provider> ya que no abrá un proveedor en WordListClient.
7. En el nivel superior, dentro de la etiqueta <manifest> tag, Agregue un permiso <uses-permission> para usar el content provider de WordListSQLWithContentProvider.

```
<uses-permission android:name =
"com.android.example.wordlistsqlwithcontentprovider.PERMISSION"/>
```

8. Elimine la clase WordListContentProvider, debido a que la aplicación acederá al content provider de WordListSQLWithContentProvider.

9. Elimine la clase WordListOpenHelper, debido a que la aplicación no necesita una base de datos o un open helper propios.

10. Observe la MainActivity y el WordListAdapter. Note que el código para inserter, eliminar y actualizarpalabras permanence sin cambios llamando al content provider de WordListSQLWithContentProvider usando los URIs especificados en la clase Contract.

11. Ejecute WordListClient.

    - A pesar de no tener datos propios, WordListClient muestra los datos, que son los datos que obtiene del content provider de la aplicación WordListSQLWithContentProvider.

12. Inserte una cuanta palabras en WordListClient.

13. Inicie Start WordListSQLWithContentProvider.

    - Note que la palabra que usted insertó con WordListClient, se muestra también en WordListSQLWithContentProvider, porque comparten una fuente de datos.

    - Borre la palabra en WordListSQLWithContentProvider y observe que la palabra también se borra de la pantalla de WordListClient. (Es posible que tenga que desplazarse para que aparezca el cambio).

14. A medida que interactúa con la otra aplicación, los cambios realizados por una aplicación se reflejan en la otra.

En el ejemplo de la tienda de sombreros, el propietario del almacén puede actualizar el inventario de sombreros con nuevos sombreros rojos o de fantasía, y las aplicaciones de tienda podrán mostrar inmediatamente estos nuevos sombreros a sus clientes. Y si la tienda de sombreros rojos vende todos los sombreros rojos de fantasía, la tienda de sombreros de fantasía sabrá que el inventario de sombreros rojos de fantasía está agotado.

Resumen

- LA aplicación puede permitir a otras aplicaciones interactuar con su content provider y obtener o editar datos.

- Se puede crear una aplicación cliente que en lugar de proporcionar sus propios datos, los obtiene del content provider de otra aplicación.

- Se puede separar la gestión de los datos de la visualización de los datos. Es decir, puede tener una aplicación que administre el content provider y muchas aplicaciones cliente que utilicen los datos proporcionados por ese content provider.

7.8 Cargar y Mostrar Datos Obtenidos de un Content Provider

En esta práctica aprenderá a cargar los datos proporcionados por el content provider de otra aplicación en segundo plano y mostrarlos al usuario, cuando esté listo.

Pedirle a un ContentProvider los datos que desea mostrar puede tomar tiempo. Si solicita datos al content provider de una Activity (y los ejecuta en el hilo de la UI), es posible que la aplicación se bloquee lo suficiente como para causar un retraso visible para el usuario, e incluso que el sistema emita un mensaje "Application Not Responding" (Aplicación sin respuesta). Por lo tanto, debe cargar los datos en un hilo separado, en el background, y visualizar los resultados una vez finalizada la carga.

Para ejecutar una consulta en un hilo separado, se utiliza un loader (cargador) que se ejecuta asincrónicamente en background y se vuelve a conectar a la Activity cuando termine. Específicamente, CursorLoader ejecuta una consulta en el background y la vuelve a ejecutar automáticamente cuando los datos asociados con la consulta cambian.

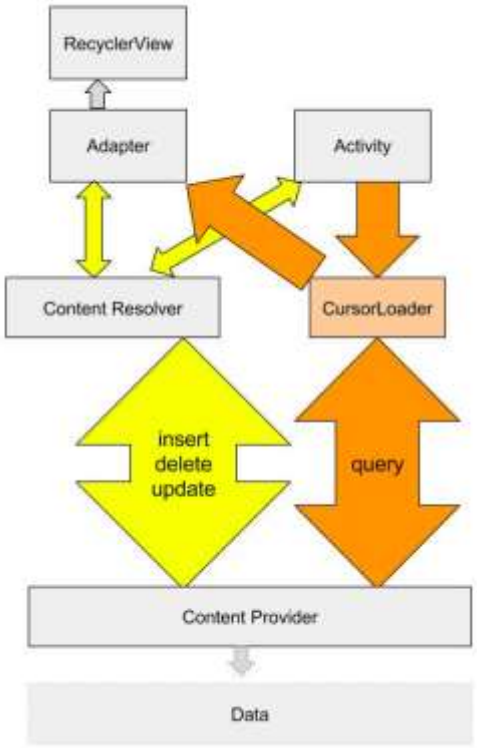
Usted ha utilizado un AsyncTaskLoader en una práctica anterior. CursorLoader extiende AsyncTaskLoader para trabajar con content providers.

A un nivel alto, se necesitan las siguientes piezas para utilizar un loader para visualizar datos de un content provider:

- Una Activity o fragment.
- Una instancia del LoaderManager en la Activity.
- Un CursorLoader para cargar los datos respaldados por el ContentProvider.
- Una implementación de LoaderManager.LoaderCallbacks, una interfaz callback abstracta para que el cliente interactúe con el LoaderManager.
- Una forma de mostrar los datos del loader, comúnmente usando un adapter. Por ejemplo, se pueden mostrar los datos en un RecyclerView.

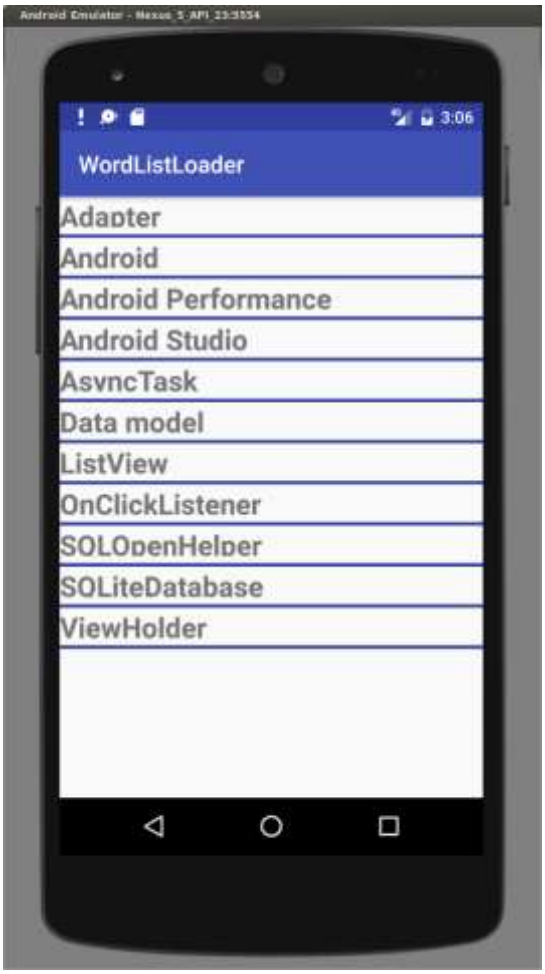
El siguiente diagram muestra la arquitectura completa de una aplicación con un loader.

- El loader realiza la consulta de los elementos en segundo plano. Si los datos cambian, automáticamente obtiene un nuevo conjunto de datos para el loader.
- Las operaciones de inserción, eliminación y actualización no utilizan el loader. Sin embargo, después de que los datos cambian debido a una operación de inserción, eliminación o actualización, el loaderr obtiene los datos actualizados y notifica al adapter.



Resumen de la Aplicación

Usando WordListClient de la práctica anterior como fuente para parte del código, creará una nueva aplicación, WordListLoader la cual carga y muestra datos del content provider de WordListSQLWithContentProvider. La siguiente captura de pantalla muestra cómo la aplicación terminada mostrará las palabras.



IMPORTANTE:

- Debe instalar una aplicación WordListWithContentProvider que content provider para que haya un content provider disponible para WordListLoader.

- Utilize la aplicación WordListClient que construyó en la práctica anterior como referencia y para reutilizar el código.

Tarea 1. Crear la aplicación base para el WordListLoader

En esta tarea creará un proyecto y partes de la aplicación que no son específicas a los loaders. Necesita la aplicación WordListClient cargada en Android Studio, para poder copiar código de ella.

1.1 Crear un proyecto a con las clases Contract y WordListItem y archivos layout

1. Inicie el Android Studio y cargue la aplicación terminada WordListClient de la práctica anterior.

2. Cree un nuevo proyecto con la plantilla Empty Activity y llámelo WordListLoader.

3. Agregue los permisos para el content provider de WordListSQLWithContentProvider's al Android Manifest.

```
<uses-permission android:name =
 "com.android.example.wordlistsqlwithcontentprovider.PERMISSION"/>
```

4. Cree una nueva clase Java y llámela Contract.
5. Copie la clase Contract de WordListClient en la nueva clase Contract de WordListLoader. Asegúrese de no copiar el nombre del paquete.
6. En el WordListLoader, cree una nueva clase Java y llámela WordListItem.
7. Copie la clase WordItem del WordListClient en la nueva clase WordItem del WordListLoader.
8. Copie el layout del recycler view de la actity_main.xml del WordListClient al WordListLoader. Remueva el botón de acción flotante (FAB).
9. Cree un Nuevo layout para WordListItem, wordlist_item.xml.
10. Usando wordlist_item.xml del WordListClient como referencia, cree un LinearLayout con un único TextView.
    - El id del TextView debe ser android:id="@+id/word".
    - Resuelva los strings, dimensions y estilos que esté reusando. Note que puede hacer copy/paste de archivos entre los proyectos. Sobrescriba el archivo XML existente en WordListLoader.
    - Cambie el app_name a WordListLoader en strings.xml.
11. En este punto, no debería ver errores en el Android Studio.

1.2 Agregur un RecyclerView a la MainActivity

Para mostrar los datos, agregue un RecyclerView a la MainActivity. Puede hacerlo por usted mismo o reutilizar el código de WordListClient.

1. Añada el RecyclerView y el Coordinator Layout de la support library al archivo build.gradle.

```
compile 'com.android.support:recyclerview-v7:24.1.1'
compile 'com.android.support:design:24.1.1'
```

2. Importe las versions de la support library del RecyclerView y el LinearLayoutManager en la MainActivity.

```
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
```

3. Cree un TAG para la MainActivity.
4. Cree una variable private mRecyclerView para el RecyclerView.

5. Cree una variable private mWordListAdapter para el adapter. Este permanecerá rojo, hasta que cree la clase adapter.

6. En el onCreate() de la MainActivity, cree un RecyclerView, cree un WordListAdapter, configure el adapter en el Recyclerview, y y adjunte un LinearLayoutManger. Vea el WordListClient para una muestra del código.

```
// Create recycler view.
mRecyclerView = (RecyclerView) findViewById(R.id.recyclerview);
// Create an adapter and supply the data to be displayed.
mAdapter = new WordListAdapter(this);
// Connect the adapter with the recycler view.
mRecyclerView.setAdapter(mAdapter);
// Give the recycler view a default layout manager.
mRecyclerView.setLayoutManager(new LinearLayoutManager(this));
```

7. Si hace build de la aplicación ahora, solamente WordListAdapter debe estar en rojo. La aplicación todavía no se puede ejecutar.

1.3 Cree el WordListAdapter

Utilice el WorldListAdapter de WordListClient y los código de abajo como referencia para crear este adapter. Si necesita un refrescamiento, vuelva a visitar el capítulo del RecyclerView de este curso.

1. Cree una nueva clase Java WordListAdapter que extienda a Recyclerview.Adapter.

```
public class WordListAdapter
extends RecyclerView.Adapter<WordListAdapter.WordViewHolder>   {}
```

Usando el WordListAdapter como referencia, añada lo siguiente:

2. Agregue una clase interna WordViewHolder con un TextView, llamado wordItemView e inflela desde el text view con el id "word".

```
class WordViewHolder extends RecyclerView.ViewHolder {
   public final TextView wordItemView;

   public WordViewHolder(View itemView) {
       super(itemView);
       wordItemView = (TextView) itemView.findViewById(word);
   }
}
```

3. Añada un TAG para mensajes log.

```
private static final String TAG = WordListAdapter.class.getSimpleName();
```

4. Añada variables miembro para el LayoutInflator y el context.

```
private final LayoutInflater mInflater;
private Context mContext;
```

5. Implemente el constructor del WordListAdapter.

```
public WordListAdapter(Context context) {
   mInflater = LayoutInflater.from(context);
   this.mContext = context;
}
```

6. Implemente (o copie) el método onCreateViewHolder para inflar una vista wordlist_item.

```
@Override
public WordViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
   View mItemView = mInflater.inflate(R.layout.wordlist_item, parent,
                                      false);
   return new WordViewHolder(mItemView);
}
```

7. Presione Alt-Enter en el enbazado de la clase adapter y "choose implement methods" para crear las plantillas de los métodos getItemCount() y onBindViewHolder().

8. En este punto, ya no deben haber líneas o palabras en rojo en el código.

9. Ejecute la aplicación y debe mostrar una activida en blanco tal como se muestra en la siguiente figura, debido a que aún no ha cargado ningún dato. Se agregarán datos en la siguiente tarea.

Tarea 2. MainActivity: Agregar un LoaderManager y LoaderCallbacks

Cuando se utiliza un loader para cargar los datos, se utiliza un loader manager para ocuparse de los detalles de la ejecución del loader.

El LoaderManager es una clase de conveniencia que administra todos los loaders. Sólo necesita un loader manager por actividad. Por ejemplo, el loader manager se encarga de registrar un observador con el content provider, que recibe las callbacks cuando cambian los datos en el content provider.

2.1 Agregar el Loader Manager

1. Abra MainActivity.java
2. Extienda la firma de la clase para implementar LoaderManager.LoaderCallbacks. Importe la support library version.

```
public class MainActivity extends AppCompatActivity implements
LoaderManager.LoaderCallbacks<Cursor>
```

3. Implemente las plantillas de los métodos onCreateLoader(), onLoadFinished(), y onLoaderReset().
4. En el onCreate(), cree un LoaderManager desde la support library y registre a loader con él.
    - El primer argumento es una etiqueta numérica; debido a que solo necesita un loader, no importa que número seleccione.
    - No está pasando ningunos datos, de manera que el Segundo argumento es null.
    - Y enlazará el loader a la actual MainActivity (this).

```
getSupportLoaderManager().initLoader(0, null, this);
```

2.2 Implementar el onCreateLoader()

El LoaderManager llama al método onCreateLoader() paracrear el loader, si no existe aún.

Cree un cargador proporcionándole un contexto y la URI desde la que va a cargar los datos (en este caso, el content provider de WordListSQLWithContentProvider, la URI especificada en el contrato).

1. En onCreateLoader(), cree una queryUri y una proyección. Utilice la misma URI que utiliza el content resolver para consultar al content provider. Puede encontrarlo en el Contrat y es usado en WordListClient.
2. Cree y devuelva un nuevo CursorLoader a partir de estos argumentos. Importe el CursorLoader desde la support library.

```
@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
   String queryUri = Contract.CONTENT_URI.toString();
   String[] projection = new String[] {Contract.CONTENT_PATH};
   return new CursorLoader(this, Uri.parse(queryUri), projection, null,
null, null);
}
```

2.3 Implementar el onLoadFinished()

Cuando la carga ha finalizado, se necesita enviar los datos al adapter.

1. Llamar a setData() en onLoadFinished(). El código se volverá rojo, y usted lo implementará en la siguiente tarea. El argumento setData() es el cursor con "data" devuelto por el loader cuando termina de cargar.

```
@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
   mAdapter.setData(data);
}
```

2.4 Implementar el onLoaderReset()

Al resetear el loader, permita que el adapter conozca que los datos no están disponibles pasando null a setData().

```
@Override
public void onLoaderReset(Loader<Cursor> loader) {
   mAdapter.setData(null);
}
```

Tarea 3. WordListAdapter: Implementar setData(), getItemCount(), y onBindViewHolder()

Como tareas finales, necesita implementar el método setData() mencionado anteriormente, e implementar onBindViewHolder() para trabajar con el loader y mostrar los datos. Así es como sucede esto:

- Cuando el loader termina de cargar datos nuevos o modificados en el background, se ejecuta el método onLoadFinished() que implementó en MainActivity.
- onLoadFinished() llama WordListAdapter.setData(), que actualiza la variable mCursor en el adapterr con los últimos datos del loader y notifica al adapter que los datos han cambiado.
- El adaptador actualiza la interfaz de usuario con los nuevos datos, vinculando las vistas a los datos en onBindViewHolder().

3.1. Implementar el setData()

Necesita una forma de configurar y almacenar la última versión cargada de los datos con el adapter. Para esta aplicación, el loader devuelve los datos como un cursor, por lo que necesita crear una variable miembro mCursor de tipo Cursor que siempre contendrá el último conjunto de datos.

El método setData() es llamado por el loader cuando termina de cargar o es reseteado, y necesita actualizar mCursor.

1. Cree una variable miembro private de tipo Cursor. Llámala "mCursor" e inicialízela en null.
2. Implemente el método public setData(). Este toma un argumento Cursor y no devuelve nada.
3. En el cuerpo, asigne a mCursor el valor del argumento pasado en Cursor y llame al notifyDataSetChanged(), para que el adapter actualice la pantalla.

```
public void setData(Cursor cursor) {
   mCursor = cursor;
   notifyDataSetChanged();
}
```

3.2. Implementar el getItemCount()

En vez de 0, getItemCount() necesita devolver el número de elementos en mCursor. Si mCursor is null, devuelva -1.

```
@Override
public int getItemCount() {
   if (mCursor != null) {
       return mCursor.getCount();
   } else {
       return -1;
   }
}
```

3.3 Implementar el onBindViewHolder()

En WordListClient, el método onBindViewHolder() utiliza un content resolver para obtener datos del content provider de WordListSQLWithContentProvider. En esta aplicación, onBindViewHolder() utiliza los datos proporcionados por el loader y almacenados en mCursor.

En onBindViewHolder, maneje las siguientes situaciones.

1. Si mCursor es null, no haga nada, pero muestre un mensaje en el log. En una aplicación real, también necesitaría notificar al usuario de una manera significativa.

2. Si mCursor no es null pero no contiene ninguna palabra, ponga el texto de TextView en ERROR: NO WORD. De nuevo, en una aplicación real, usted manejaría esto dependiendo del tipo de aplicación que tenga.

3. De lo contrario, obtenga el índice de la columna para la columna "word" (no puede asumir que la columna está en una ubicación fija en la fila del cursor), y usando el índice, recupere la palabra. Ajuste el texto del text view a la palabra.

```
@Override
public void onBindViewHolder(WordViewHolder holder, int position) {

   String word = "";

   if (mCursor != null) {
       if (mCursor.moveToPosition(position)) {
           int indexWord =
mCursor.getColumnIndex(Contract.WordList.KEY_WORD);
           word = mCursor.getString(indexWord);
           holder.wordItemView.setText(word);
       } else {
           holder.wordItemView.setText(R.string.error_no_word);
       }
   } else {
       Log.e (TAG, "onBindViewHolder: Cursor is null.");
   }
}
```

3.4 Ejecute y pruebe la aplicación

La aplicación WordListLoader debería funcionar exactamente igual que la aplicación WordListClient para mostrar una lista de palabras. Para probar su aplicación, haga lo siguiente.

1. Asegúrese de que WordListSQLWithContentProvder esté instalado en el dispositivo, para que su aplicación tenga un proveedor de contenido desde el que cargar. De lo contrario, su aplicación mostrará una vista de texto en blanco.

2. Ejecute WordListLoader. Debería ver la misma lista de palabras que en WordListSQLWithContentProvider.

Resumen

- En este capítulo, aprendió a utilizar un loader para cargar datos de un content provider que no forma parte de su aplicación.

Apéndices:

Copiar y renombrar un proyecto

Para algunos ejercicios, deberá hacer una copia de un proyecto antes de hacer nuevos cambios. También puede copiar un proyecto para usar parte de su código en un nuevo proyecto. En cualquier caso, puede copiar el proyecto existente (**ExistingProject**), y luego renombrar y refactorizar los componentes del nuevo proyecto (**NewProject**) para usar el nombre del nuevo proyecto.

1. Copiar el proyecto

1. En el sistema de archivos de su computadora (no en Android Studio), haga una copia del directorio **ExistingProject**.
2. Cambie el nombre del directorio copiado a **NewProject**.

2. Renombrar y refactorizar los componentes del proyecto

El antiguo nombre del proyecto, **ExistingProject**, aún aparece en todos los paquetes y archivos en la nueva copia de su proyecto. Cambie el archivo y las referencias del paquete en su aplicación al nuevo nombre, de la siguiente manera:

1. Abra el Android Studio
2. Clic en **Open an existing Android Studio project**.
3. Navegue hasta el directorio **NewProject**, selecciónelo y haga clic en **OK.**
4. Seleccione **Build > Clean Project** to eliminar los archivos auto generados.
5. Haga clic en **1: pestaña Project** y seleccione **Android** del menu drop-down menu para ver los archivos en la vista Project.
6. Expanda **app > java.**
7. Clic derecho en **com.example.android.existingproject** y seleccione **Refactor > Rename**. Esto abrirá el dialogo Rename.
8. Cambie el **existingproject** a **newproject**.
9. Marque **Search in comments and strings** y **Search for text occurrences** y haga clic en **Refactor**.
10. El panel **Find Refactoring Preview** apraecerá, mostrando el código a ser refectaorizar.
11. Haga click en **Do Refactor**.
12. Expanda **res > values** y haga doble-clic en el archivo **strings.xml**.
13. Cambie el string **name="app_name"** al **New Project**.

3. Actualizar los archivos build.gradle y AndroidManifest.xml

Cada aplicación que es creada debe tener una ID de aplicación única, tal como se define en el archivo build.gradle de la aplicación. Aunque los pasos anteriores deberían haber cambiado el archivo build.gradle, debe ser verificado para asegurarse, y también sincronizar el proyecto con el archivo gradle:

1. Expanda **Gradle Scripts** y haga doble-clic en **build.gradle (Module: app)**.
2. Bajo **defaultConfig**, verifique que el valor de la clave applicationID ha sido cambiada a "**com.example.android.newproject**". SI eso no ha pasado, cámbielo manualmente.

3. Haga clic en **Sync Now** en la esquina superior derecha de la ventana del Android Studio.

   Ayuda: Puede también seleccionar **Tools > Android > Sync Project** con el archive Dradle para sincronizar sus archivos gradle.

Además, algunas aplicaciones incluyen el nombre de la aplicación en forma legible (como "New Proyect " en lugar de newproyect) como una etiqueta en el archivo AndroidManifest.xml.

1. Expanda **app > manifests** y haga doble clic en **AndroidManifest.xml**.
2. Busque la siguiente declaración y, si es necesario, cambie la etiqueta del recurso de string para el nuevo nombre de la aplicación:

```
android:label="@string/app_name"
```

Bibliografía

- https://developer.android.com/guide/, consultado desde marzo 2018 a mayo 2018
- https://legacy.gitbook.com/@google-developer-training, consultado desde marzo 2018 a mayo 2018
- https://developers.google.com/training/android/, consultado desde marzo 2018 a mayo 2018
- https://developers.google.com/training/courses/android-fundamentals, consultado desde marzo 2018 a mayo 2018