Elementos de la Interfaz de Usuario

Elementos de la IU

Al momento de escribir este documento, hay aproximadamente un poco más 3.4 millones de aplicaciones en Google play store. Estas son muchas aplicaciones y muchos desarrolladores con quienes competir.

Si vas a publicar tu aplicación en la tienda, debemos al menos asegurarnos de que la aplicación pueda funcionar cara a cara con las aplicaciones hechas profesionalmente. Necesitas un poco de brillo.
Google ha publicado un conjunto de directrices para el enfoque de la interfaz de usuario; se llama diseño de materiales, y puedes leer más sobre esto en su sitio web https://material.io.
Es un gran tema y no tenemos la intención de cubrirlo aquí. Pero en este tema, discutiremos tres cosas que pueden ayudarte a comenzar y orientarte en la dirección de una investigación más a fondo. Estos son los temas y colores, la AppBar y los Fragments.

Temas y colores

Cuando AS4 crea un proyecto con una actividad vacía, hace bastantes cosas para ti, y hemos visto algunas de esas posibilidades en las últimas dos aplicaciones de ejemplo en las que hemos trabajado.

En esta parte del curso, nos centraremos un poco en la estética. No haremos una inmersión profunda en el diseño de la IU porque es un área grande, y está más allá del alcance de este curso (aunque se tienen algunas ideas en otros acetatos), y en mi experiencia; No hemos sido muy detallados con la IU.

Pero veremos algunas cosas rápidas y fáciles para que nuestras aplicaciones se vean decentes. Creamos un nuevo proyecto con una actividad vacía y le asignamos el nombre StylesAndThemes; deja el factor de forma predeterminado en “Teléfono y tabletas”.

Nota. Un estilo es una colección de atributos que especifica la apariencia y el formato de un objeto de vista individual; un estilo se refiere a la altura, el color, la fuente, etc. Un tema, por otro lado, es un estilo aplicado a toda la actividad o aplicación

Nunca nos hemos equivocado con estos colores en nuestras aplicaciones anteriores; simplemente dejamos que AS4 decida por nosotros cómo se vería nuestra aplicación (la figura 10.1 muestra el tema predeterminado para una aplicación), al menos en lo que respecta

al color. Si deseas hacer un poco de branding y darle cierta identidad a tu aplicación, podemos comenzar por personalizar el color y el tema.

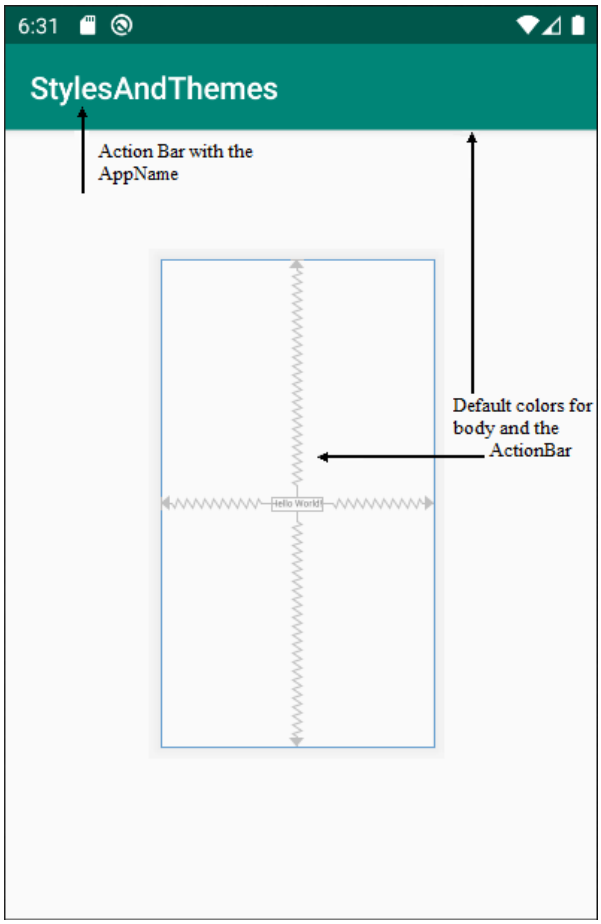


Figura 10.1: activity_main.

Colores

Si bien es posible especificar el color para cada parte de la aplicación, puede resultar tedioso y lento. Una forma más fácil sería trabajar con temas. El tema general de una aplicación está controlado por el AndroidManifest (puedes abrir el archivo de manifiesto desde la ventana de la herramienta del proyecto, app ➤ manifests ➤ AndroidManifest).

Android hace un uso intensivo de XML, como probablemente ya sepas. Además, la práctica de referenciar valores, ya sea de cadena, color, estilo u otra cosa, es muy común: la encontrarás en todas partes. Veamos dos entradas en el archivo del manifiesto (consulta el Listado 10.1).

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"                    (1)
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">                    (2)
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Listado 10.1: AndroidManifest.

(1) La notación @string significa que estamos haciendo referencia a este valor desde app/res/values/strings.xml. Esta es la forma preferida de definir cadenas en tu aplicación. Escribir las cadenas en un recurso nos brinda la capacidad de administrar los recursos de cadena desde una ubicación central; también facilita la facilidad de cambio y la localización. Los recursos de cadenas y estilos se pueden abrir desde la ventana de la herramienta del proyecto (consulta la figura 10.2).

app/res/values/strings.xml
<resources>
<string name = "app_name"> StylesAndThemes </ string>
</ resources>

(2) La notación @style significa que estamos haciendo referencia a esta entrada desde el archivo app/res/values/styles.xml. Dentro de este archivo, debe haber una definición para AppTheme.

En el Listado 10.2, se define el valor de AppTheme al que se hace referencia desde el archivo del manifiesto. En primer lugar, no fue construido desde cero; hereda del tema DarkActionBar, pero nos permite personalizar un par de colores.

Hay tres colores definidos en los estilos, pero puedes agregar más si lo deseas. Simplemente trabajaremos con estos tres por ahora.

Nota. En versiones anteriores de Android Studio, es posible que hayas necesitado crear el archivo /res/styles.xml. En AS4, cuando creamos la actividad vacía, el archivo de recursos de estilos se generó automáticamente.

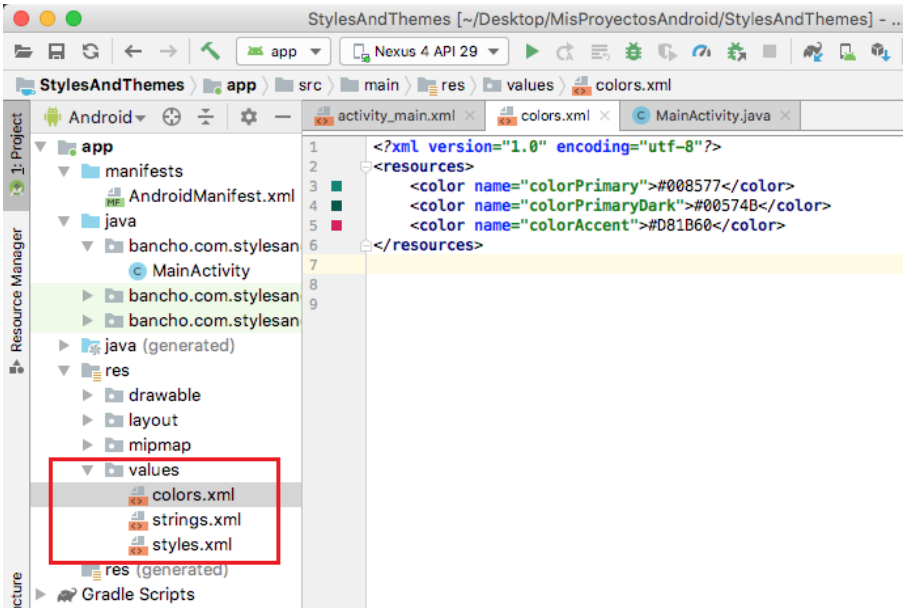


Figura 10.2: colors.xml.

```xml
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

</resources>
```

Listado 10.2: /app/res/values/styles.xml.

colorPrimary, colorPrimaryDark y colorAccent no están (realmente) definidos en styles.xml; en su lugar, encontramos otra indirección que nos refiere a otro archivo de recursos. A veces puede resultar molesto, pero estas indirecciones son necesarias en nombre de la capacidad de administración. Entonces, necesitas acostumbrarte.

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#008577</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>
</resources>
```

Listado 10.3: /app/res/values/colors.xml.

Si abres colors.xml, finalmente podemos ver los valores hexadecimales de los colores.

AS4 te muestra los colores en la cuneta del editor; el color cambia inmediatamente cuando cambias los valores hexadecimales (figura 10.3). Si deseas cambiar el tono de la aplicación, puedes comenzar por hacer cambios en este archivo.

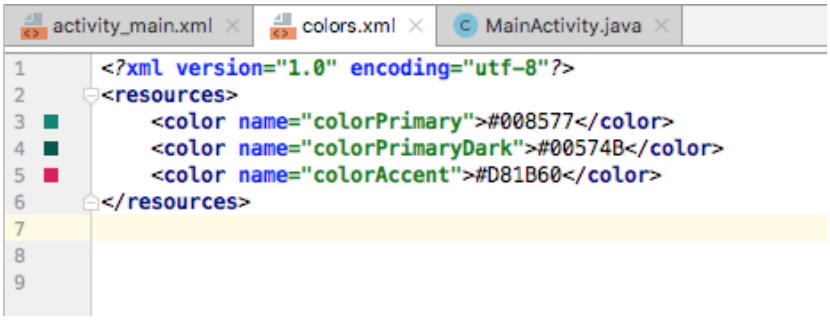


Figura 10.3: colors.xml en el editor principal.

Si necesitas ayuda con los valores hexadecimales de los colores, hay muchos recursos web para eso; colorhexa.com es uno de estos sitios (www.color.hexa.com), que te muestra colores y gradientes relacionados de colores específicos, por lo que es bueno utilizarlo cuando desees trabajar con valores hexadecimales de color. Sin embargo, la maquinación de colores (color scheming) es una gran área, y existen bastantes principios y pautas involucradas. Otro buen recurso para los colores es Materialpalette (www.materialpalette.com, que se muestra en la figura 10.4).

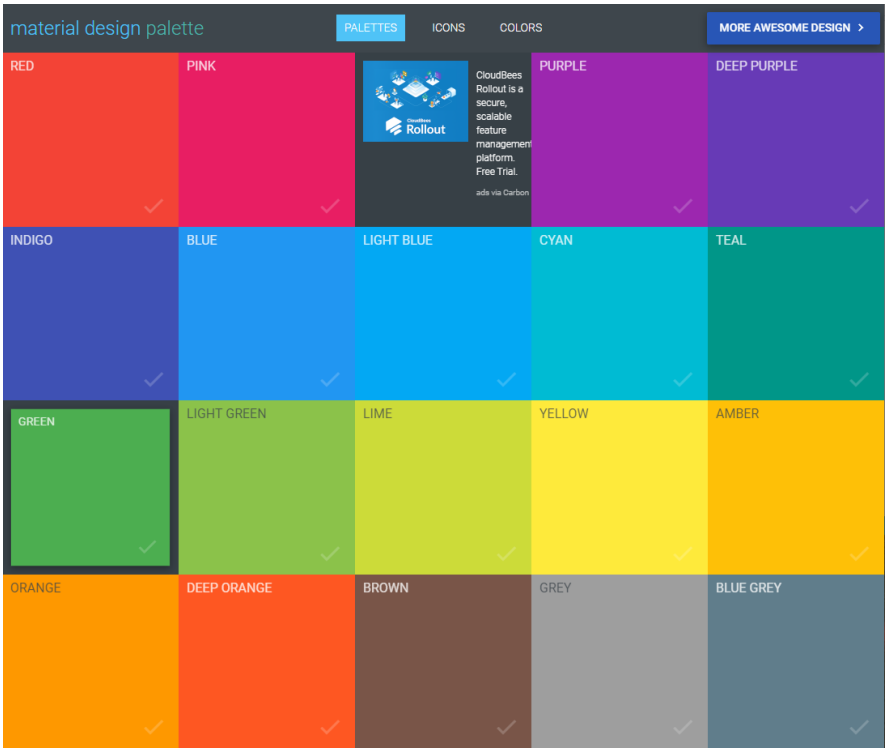


Figura 10.4: Materialpalette.com.

Materialpalette está orientada al diseño de Android, específicamente al diseño de materiales. La idea básica es elegir dos colores y el sitio crea una paleta para ti. Ahora podemos simplemente copiar los valores hexadecimales de los colores primario, primario oscuro, acento y primario claro.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#009688</color>
    <color name="colorPrimaryDark">#00796B</color>
    <color name="colorAccent">#CDDC39</color>
    <color name="colorPrimaryLight">#B2DFDB</color>
</resources>
```

Listado 10.4: colors.xml personalizados.

Cambia el editor principal a la pestaña activity_main para ver el nuevo aspecto de nuestra aplicación (figura 10.5).

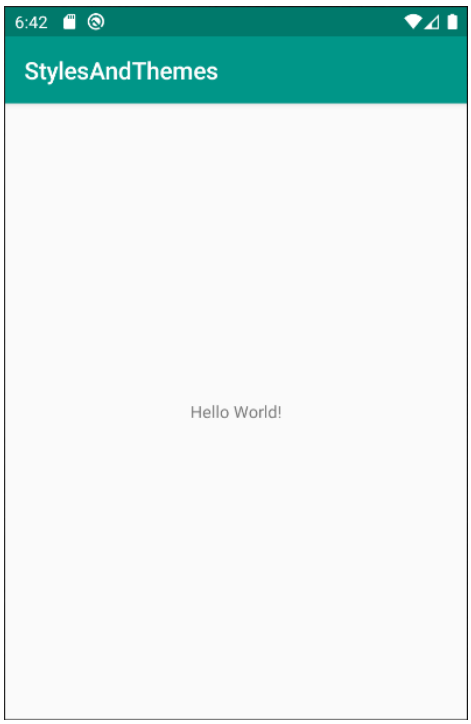


Figura 10.5. activity_main con colores personalizados.

Temas

El aspecto y la sensación de Android han evolucionado a lo largo de los años. A medida que aparecieron los dispositivos más nuevos y las versiones más nuevas de Android, esto también dio paso a un nuevo aspecto para las aplicaciones.

La figura 10.6 muestra algunas instantáneas de los temas a lo largo de los años.

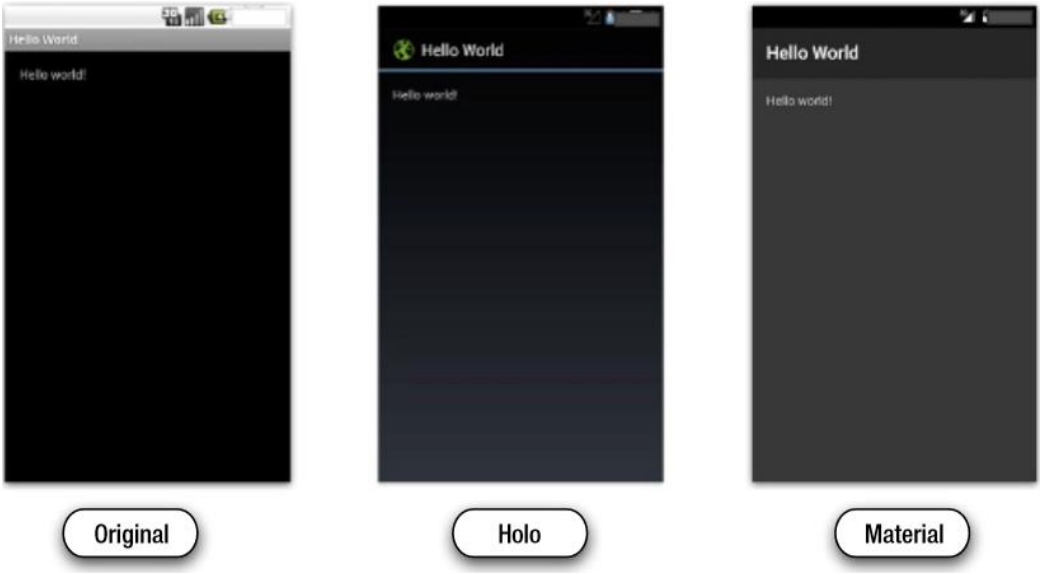


Figura 10.6: Varios temas de Android.

La Tabla 10.1 enumera algunos de los temas más importantes e importantes de Android.

El tema predeterminado para los últimos proyectos que hemos creado es Theme.AppCompat.Light. DarkActionBar. Es un tema decente, y si tus necesidades son bastante simples, puede que no sea necesario seguir trabajando en este tema. Pero si deseas ajustar la apariencia de tu aplicación, puedes comenzar a probar varios temas. Puedes hacerlo editando el tema principal en styles.xml (como se muestra en la figura 10.7).

Tema	Descripción
Theme.Light	Esto fue utilizado por las versiones de Android 2 y siguientes (API 10 y siguientes); por ejemplo, Gingerbread.
Theme.Holo.Light	Android 3 (API 11 y superior).
Theme.Holo.DarkActionBar	API 14 en adelante.
Theme.AppCompat	API 7.
android.Theme.Material	API 21 (Lollipop) en adelante.

Tabla 10.1: Temas Android.

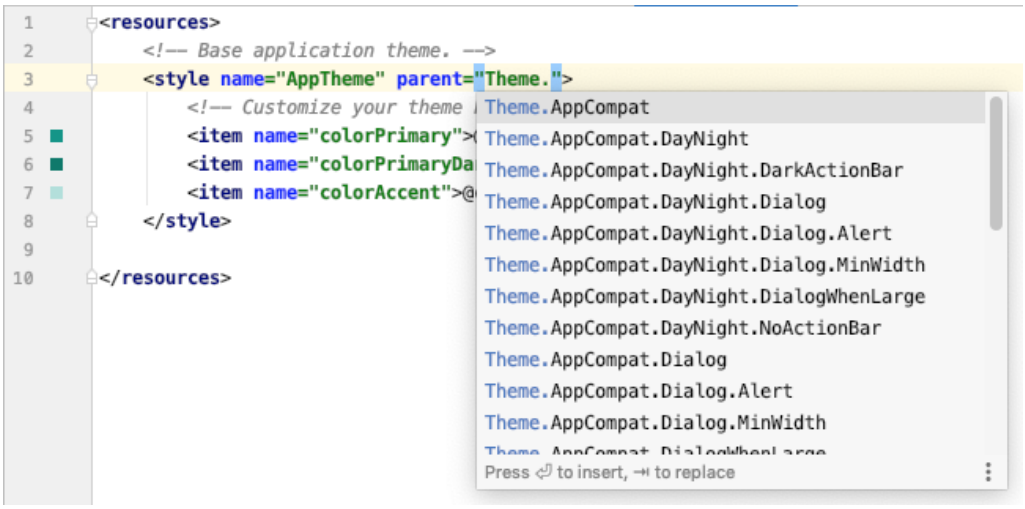


Figura 10.7: Editar temas.

Intenta cambiar el tema un par de veces y luego vuelva a activity_main (modo de diseño) para que puedas explorar los diversos temas de Android.

Los menús AppBar

Los menús son muy importantes en el diseño de la interfaz de usuario, y son herramientas indispensables en el arsenal del programador. Los sistemas de menús permiten al usuario acceder a la funcionalidad de una aplicación.

Tradicionalmente, los sistemas de menú se organizan jerárquicamente y en grupos de introducción. El sistema de menú de Android, en algún punto en el tiempo, se ha comportado exactamente así: agrupado y jerárquico. Pero eso fue en el pasado. El enfoque de Android a los menús ha cambiado drásticamente a lo largo de su vida.

Antes de la API 11 (Honeycomb), los menús de Android dependían de los botones de hardware (como los que se ven en la figura 10.8). Los desarrolladores pueden confiar en que los botones de inicio (más algunos otros, como el botón de opción) siempre estarán allí. Y creamos nuestras aplicaciones según esas suposiciones, porque esas suposiciones eran razonables en ese momento.

Por supuesto, los tiempos han cambiado. Las resoluciones de pantalla han aumentado drásticamente y los botones de hardware han desaparecido. Afortunadamente, el enfoque de Android para los menús también ha cambiado y se ha mantenido actualizado con el estado de las capacidades de hardware.

Figura 10.8: API 10 (Gingerbread).

Se ha agregado un nuevo tipo de sistema de menú a Android a partir de la API 11. Las aplicaciones que están compiladas con un SDK mínimo de 11 pueden usar la Barra de acciones (ver un ejemplo en la figura 10.9).

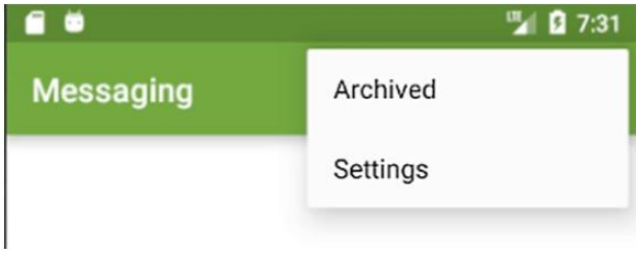


Figura 10.9: Barra de acciones.

ActionBar es un área dedicada en la parte superior de la pantalla y es persistente en toda la aplicación. Es muy parecido a la barra de menú principal de AS4 si lo piensas. En general, puedes usar la barra de acciones para mostrar las características más importantes de tu aplicación y hacer que sean accesibles de una manera predecible; por ejemplo, poner un widget de búsqueda permanente en la parte superior, y así sucesivamente.

Crea una apariencia más limpia al eliminar el desorden en tus menús; si no todos los elementos en el menú se pueden acomodar en la pantalla, la barra de acciones mostrará un icono de desbordamiento: esta es la elipsis vertical, tres puntos dispuestos verticalmente, que

siempre se encuentra en el extremo derecho de la barra. También muestra el nombre de la aplicación, por lo que refuerza la identidad de marca de la aplicación.

Hoy en día, el ActionBar ha quedado un poco pasado de moda y ha sido eclipsado por la barra de herramientas, el nuevo chico en el bloque. La barra de herramientas es un poco más versátil porque no está recortada permanentemente en la parte superior de la pantalla, puedes colocarla en cualquier lugar que desees y tiene algunas capacidades más.

El ActionBar, sin embargo, sigue siendo una solución viable para sistemas de menús simples. De hecho, nada te impide usar tanto la Actionbar como la Toolbar; solo trabaja con las mejores herramientas que tiene.

Aplicación de demostración

Crea un nuevo proyecto con los siguientes detalles (Tabla 10.2).

Application name	**ActionBar**
Project location	Usa el factor de forma predeterminado.
Form factor	Solo teléfono y tableta.
Minimum SDK	API 26 Oreo
Type of activity	Vacío
Activity name	MainActivity (predeterminado)
Layout name	activity_main (predeterminado)

Tabla 7-2: Detalles del proyecto de ActionBar.

Para usar ActionBar, el SDK mínimo debe establecerse en API 11; esto hace que Honeycomb sea un punto de corte. Si un proyecto tiene un SDK mínimo de 11 o superior, significa que puede manejar ActionBar.

Después de crear el proyecto, asegúrate de que el tema esté configurado en “AppTheme”. Después de eso, crea una carpeta de menú debajo de la carpeta **res**. En la ventana de la herramienta del proyecto, haz clic con el botón derecho en la carpeta res Nuevo ➤ Directorio de recursos de Android (consulta la figura 10.10).

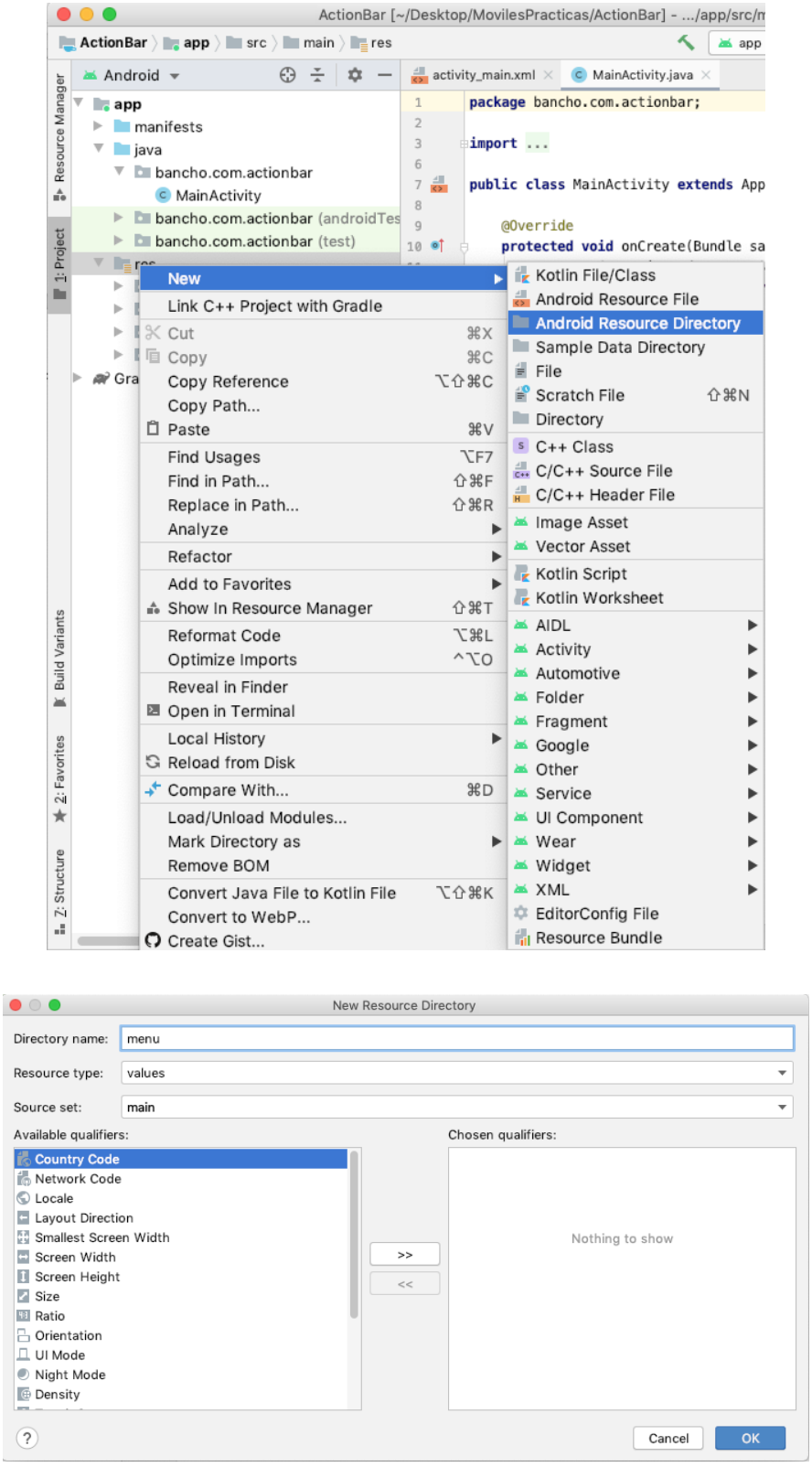


Figura 10.10: Crea un nuevo directorio de recursos.

Crea un archivo de menú debajo del directorio de menú recién creado.

Haz clic con el botón derecho en la carpeta del menú, Nuevo ➤ Archivo de recursos del menú (consulta la figura 10.11).

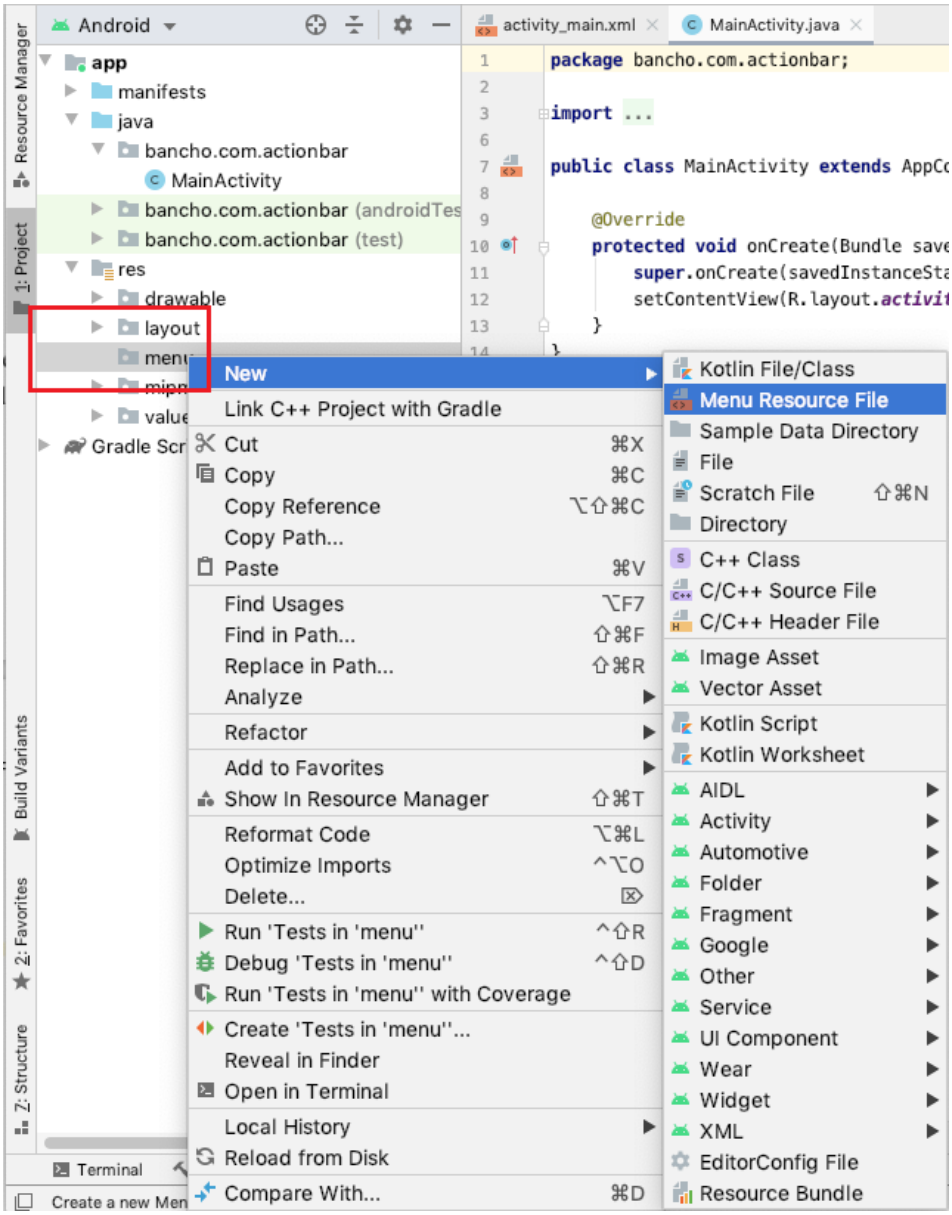


Figura 10.11: Crea una carpeta de menú.

Nombra al nuevo archivo de recursos como “main_menu”, como se muestra en la figura 10.12.

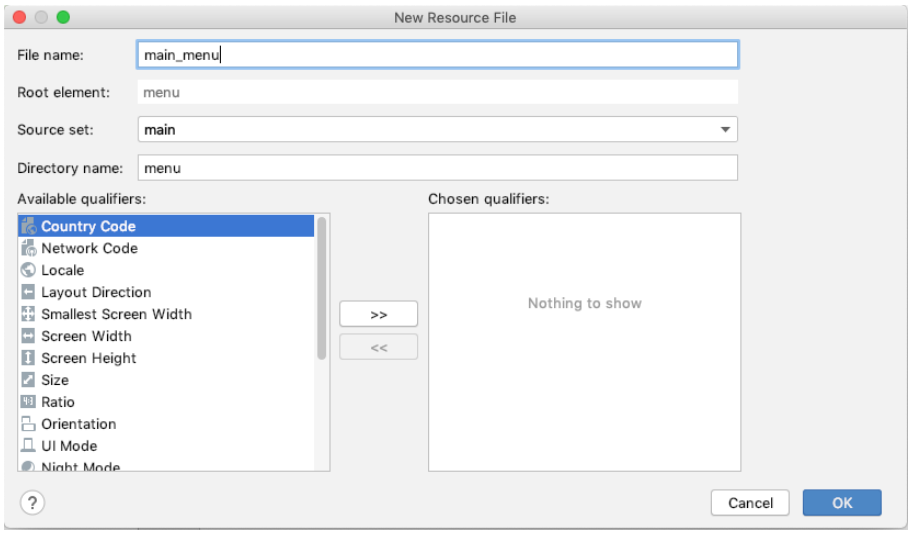


Figura 10.12: Menu resource file.

Haz doble clic en el archivo de recursos main_menu desde la ventana de la herramienta del proyecto. Agreguemos algunos elementos de menú como se muestra en el Listado 10.5. El archivo main_menu.xml se creará en la carpeta /app/res/values/menu/main_menu.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menuFile"
        android:title="@string/menuFile"/>
    <item android:id="@+id/menuEdit"
        android:title="@string/menuEdit"/>
    <item android:id="@+id/menuHelp"
        android:title="@string/menuHelp"/>
    <item android:id="@+id/menuExit"
        android:title="@string/menuExit"/>

</menu>
```

Listado 10.5: main_menu.xml.

(1) La notación @id + significa que nos gustaría crear una identificación para este artículo (item); este es el mismo ID que usaremos más tarde cuando lo referenciamos desde nuestro programa utilizando el método findViewById.

(2) El atributo de título es el que se mostrará en el menú. Podríamos escribir aquí solo una cadena simple, pero eso equivaldría a codificar ese valor en este archivo. Esa es generalmente una mala idea. Las notaciones de @string significan que estamos haciendo referencia al título del archivo /app/res/values/strings.xml.

La primera vez que escribes título, AS4 lo mostrará como un error porque el recurso de cadena aún no se ha creado en strings.xml. Selecciona el valor del atributo de título

(@string/menuFile) y usa la solución rápida (Alt + Intro en Windows/Linux | Opción + Intro en macOS) para que puedas crear el archivo de recursos. Ver la figura 10.13.

Cambia a MainActivity para que podamos agregar el main_menu recién creado a la AppBar. Para hacer esto, necesitamos anular el método onCreateOptions de MainActivity. Puedes usar la función de anulación de métodos de AS4 para hacer esto, desde la barra de menú principal, Código → Sobrescribir métodos (Code → Override Methods).

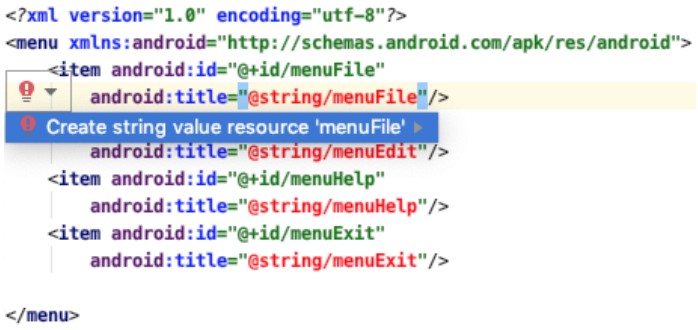


Figura 10.13: Crear un nuevo elemento de recurso de String.

```
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.MenuInflater;
import android.view.Menu;

public class MainActivity extends AppCompatActivity {

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu){
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.main_menu,menu);
        return true;
    }
}
```

Si ejecutas la aplicación en el emulador, verás algo que se parece a la figura 10.14.

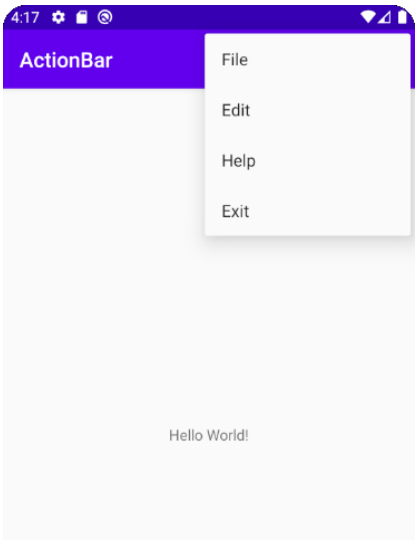


Figura 10.14: Elementos de menú en ActionBar (cuando se hace clic en el desbordamiento de acción).

Usemos algunas imágenes en el menú. AS4 viene con muchas imágenes que puedes usar para una amplia gama de aplicaciones. Antes de que podamos usar cualquier imagen, debemos agregarla a nuestra carpeta de recursos. Puedes usar recursos rasterizados (mapas de bits) o vectores para las imágenes. En este ejemplo, usaremos activos vectoriales (vector assets).

En la ventana de la herramienta del proyecto, haz clic con el botón derecho en /app/res ➤ New ➤ Vector asset (figura 10.15).

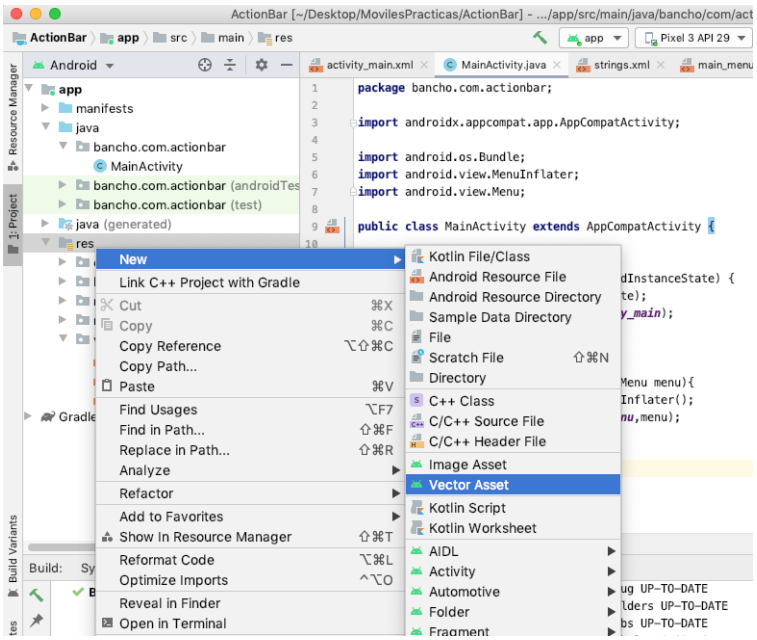


Figura 10.15: Agregar un elemento vectorial al proyecto.

Haz clic en el ícono para encontrar las imágenes que se ajusten a tus necesidades (figura 10.16).

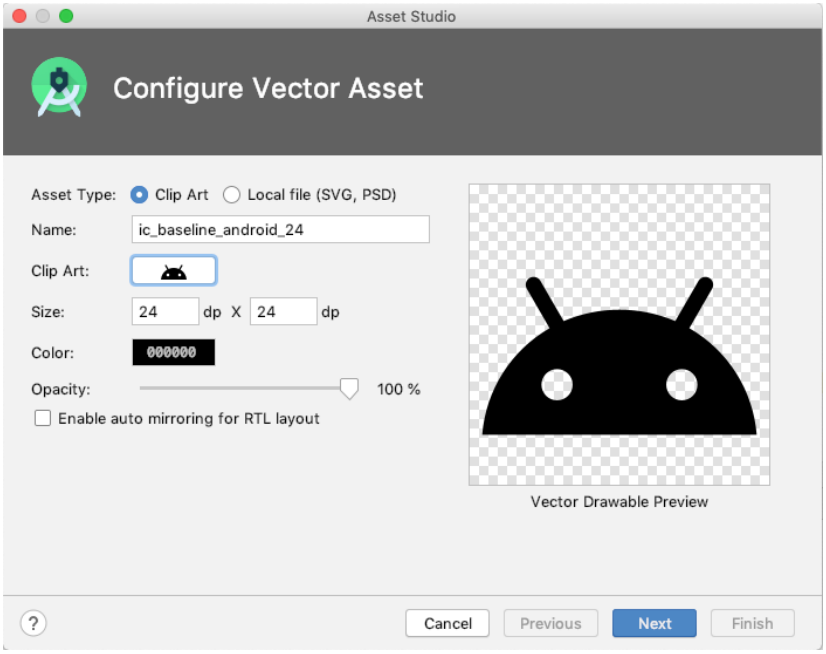


Figura 10.16: Configurar el diálogo de elementos del vector.

Para usar las imágenes en la aplicación, debemos asociarlas con cada elemento en el archivo de recursos main_menu. Ver el Listado 10.6 sobre cómo hacer esto.

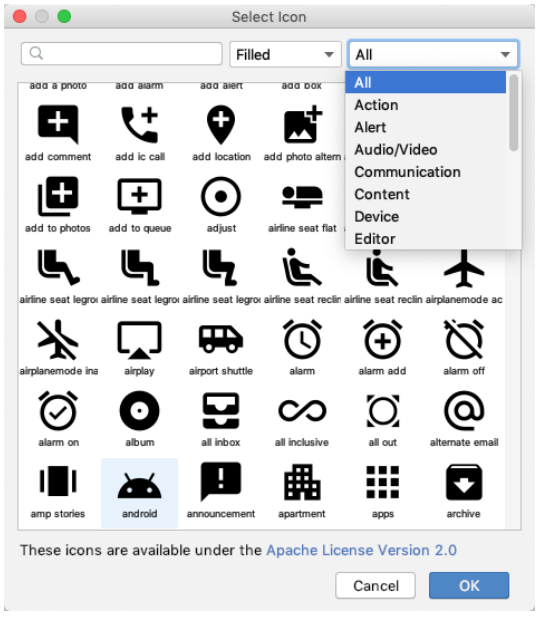
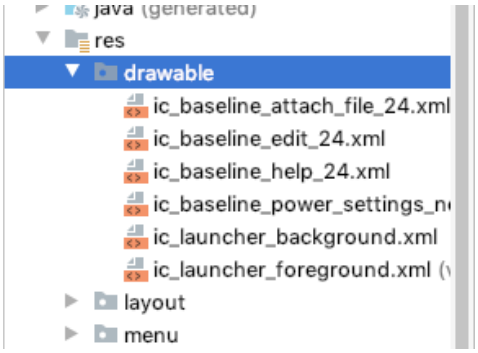


Figura 10.17: Seleccionar diálogo de icono.




```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/menuFile"
        android:title="@string/menuFile"
        android:icon="@drawable/ic_baseline_attach_file_24" 1
        app:showAsAction="ifRoom"/> 2
    <item android:id="@+id/menuEdit"
        android:title="@string/menuEdit"
        android:icon="@drawable/ic_baseline_edit_24"
        app:showAsAction="always"/> 3
    <item android:id="@+id/menuHelp"
        android:title="@string/menuHelp"
        android:icon="@drawable/ic_baseline_help_24"
        app:showAsAction="ifRoom"/>
    <item android:id="@+id/menuExit"
        android:title="@string/menuExit"
        android:icon="@drawable/ic_baseline_power_settings_new_24"
        app:showAsAction="always"/>

</menu>
```

Listado 10.6: Nuevo main_menu.xml.

(1) Los vector assets que agregamos se guardaron en /app/res/drawable.

(2) Si el atributo showAsAction es ifRoom, el ícono se mostrará solo si hay suficiente espacio en la barra de acciones; de lo contrario, los usuarios solo lo verán cuando hagan clic en el botón de desbordamiento de acción.

(3) Si el atributo showAsAction está establecido en always, el icono siempre estará visible para el usuario. Ten cuidado de usar esto con moderación; si todos tus iconos se especifican como “siempre”, el tiempo de ejecución decide cuáles de tus iconos se mostrarán, y es posible que no sean todos visibles.

Para manejar los eventos de cada elemento del menú, podemos usar el atributo android:onClick en cada elemento o anular el método onOptionsItemSelected en

MainActivity. Si deseas ir a la ruta onClick, agrega el atributo onClick a un elemento en main_menu, como el siguiente:

```
<item android:id="+id/menuEdit"
      android:title="@string/menuEdit"
      android:icon="@drawable/ic_baseline_edit_24"
      app:showAsACtion="always"
      android:onClick="mnuEdit"/>
```

Luego, en MainActivity, implementa el método mnuEdit:

```
public void mnuEdit(MenuItem item) {
  Toast.makeText(this, "Edit", Toast.LENGTH_SHORT).show();
}
```

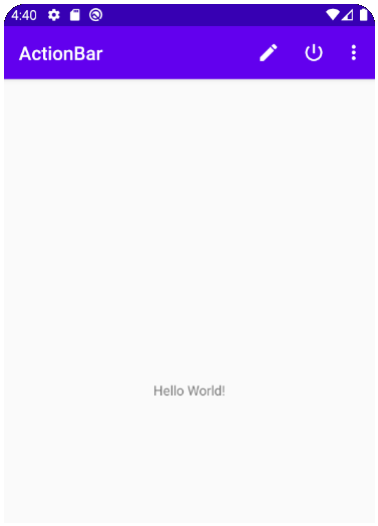


Figura 10.18: Iconos en el ActionBar.

La otra forma de manejar eventos para elementos de menú es anular onOptionItemSelected en MainActivity.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch(item.getItemId()){ 1 2
        case R.id.menuFile:
            showMessage( msg: "File");
            break;
        case R.id.menuEdit:
            showMessage( msg: "Edit");
            break;
        case R.id.menuHelp:
            showMessage( msg: "Help");
            break;
        case R.id.menuExit:
            showMessage( msg: "Exit");
            break;
        default:
            showMessage( msg: "Default");
    }
    return true;
}
```

(1) getItemId devuelve el elemento del menú en el que se hizo clic. Usaremos esto para enrutar la lógica del programa dentro de la estructura del interruptor.

(2) Estamos comparando el valor de getItemId con cada elemento del menú.

```
package bancho.com.actionbar;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.main_menu, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch(item.getItemId()){
            case R.id.menuFile:
                showMessage("File");
                break;
            case R.id.menuEdit:
                showMessage("Edit");
                break;
            case R.id.menuHelp:
                showMessage("Help");
                break;
            case R.id.menuExit:
                showMessage("Exit");
                break;
            default:
                showMessage("Default");
        }
        return true;
    }

    private void showMessage(String msg) {
        Toast.makeText(this, msg, Toast.LENGTH_SHORT).show();
    }
}
```

Listado 10.7: Código completo para MainActivity.

Fragmentos

En los primeros días de Android, cuando solo funcionaba con teléfonos y no había pantallas de alta resolución, las actividades eran suficientes como una forma de componer la interfaz de usuario e interactuar con el usuario. Luego vinieron las tabletas y las pantallas de alta resolución; se volvió cada vez más difícil crear aplicaciones que se puedan ejecutar (bien) en teléfonos y tabletas.

Los desarrolladores se enfrentaron con decisiones difíciles. O bien creaban las aplicaciones eligiendo el hardware menos capaz como objetivo, lo que lo hace como el enfoque de denominador menos común; o creaban la aplicación para adaptarla a un rango de factores de forma eliminando y agregando elementos de la IU en respuesta a la capacidad del dispositivo (que resultó ser muy difícil de hacer manualmente).

La solución de Android a este problema fue Fragmentos; esto se introdujo en algún momento en 2011 cuando se lanzó la API 11 (Honeycomb).

Los fragmentos son un concepto bastante avanzado, y los programadores principiantes pueden abordarlo con inquietud, pero el concepto básico que lo sustenta es bastante simple. Si pensamos en una actividad como unidad de composición para nuestra IU, piensa en un fragmento como una miniactividad, es una unidad de composición más pequeña.

Por lo general, mostrarás (y ocultarás) fragmentos durante el tiempo de ejecución en respuesta a algo que hizo un usuario; por ejemplo, inclinar el dispositivo, cambiar de orientación vertical a horizontal y así tener más espacio en la pantalla disponible. Incluso puedes usar fragmentos como estrategia para adaptarte a los factores de forma del dispositivo; cuando la aplicación se ejecuta en una pantalla más pequeña, solo se mostrarán algunos de los fragmentos.

Un fragmento, como una actividad, se compone de dos partes: un programa Java y un archivo de diseño. La idea es casi la misma: definir los elementos de la interfaz de usuario en un archivo XML y luego ‘inflar’ el archivo XML en el programa Java para que todos los objetos de vista en el XML se conviertan en un objeto de Java. Después de eso, podemos hacer referencia a cada objeto de vista en el XML utilizando R.class.

Una vez que hayamos envuelto nuestro cerebro en ese concepto, piensa en un fragmento como un objeto de vista común que podemos arrastrar y soltar en el archivo de diseño principal, excepto, por supuesto, los fragmentos no son vistas comunes (pero son vistas).

El siguiente flujo de trabajo resume los pasos sobre cómo comenzar a usar Fragmentos. Los exploraremos con más detalle en el proyecto de demostración.

1. Crear una actividad.

2. Crear una clase Fragment (archivo Java) y un recurso de diseño de fragmento (archivo XML).

3. En el recurso de diseño de fragmento, compón la interfaz de usuario arrastrando y soltando elementos de vista en ella, como la forma en que lo hacemos en un archivo de recursos de actividad.

4. En la clase Fragment, anula el método onCreateView e infla el archivo XML.

5. Para agregar el fragmento a la actividad estáticamente, agrega un elemento fragmento a activity_main y asocia este elemento a Fragment class.

6. Para agregar el fragmento durante el tiempo de ejecución;

    a. En el archivo de diseño activity_main, inserta un objeto ViewGroup que actuará como marcador de posición para el fragmento.
    b. En MainActivity.java, crea una instancia de la clase Fragment.
    c. Obtener un objeto FragmentManager; el método getManager() de la clase Activity debería hacer eso.
    d. Obtener un objeto FragmentTransaction llamando al método beginTransaction() del administrador de fragmentos.
    e. Agrega el fragmento a la actividad llamando al método add() del objeto transacción.

Nota. El objeto de transacción es lo que usarás para administrar la disponibilidad y visibilidad de los fragmentos. Utiliza los métodos para agregar y eliminar para adjuntar y separar fragmentos a/desde la actividad.

El tema de los fragmentos es grande, pero trataremos de cubrir al menos las técnicas básicas de composición tanto en el momento del diseño como durante el tiempo de ejecución.

Configuración del proyecto

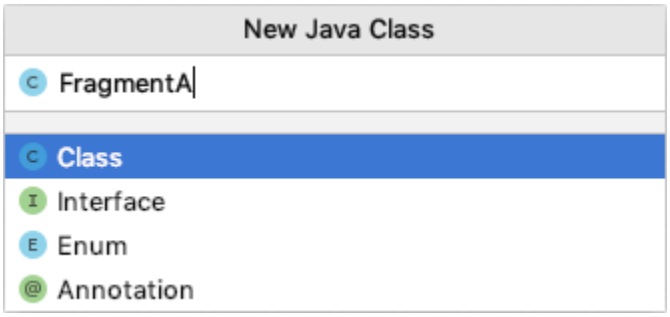
Creamos un proyecto para demostrar el uso de los fragmentos:

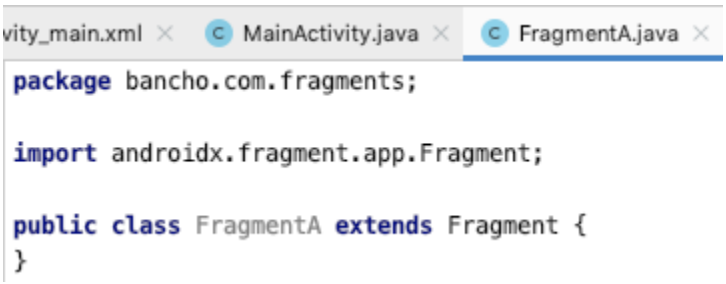
1. Crea un nuevo proyecto con los siguientes detalles (Tabla 10.3).

Application name	Fragments
Project location	Usa el predeterminado.
Form factor	Solo teléfono y tableta.
Minimum SDK	API 26 Oreo
Type of activity	Vacío
Activity name	MainActivity (predeterminado)
Layout name	activity_main (predeterminado)

Tabla 10.3: Fragments App, detalles del proyecto.

2. Crea una nueva clase, desde la barra de menú principal, Archivo ➤ Nueva ➤ Clase Java. Llámala FragmentA y extiende la clase Fragment. Asegúrate de que esté en el mismo paquete que la clase MainActivity.





Nota. Al escribir el fragmento de código se extiende Fragment, AS4 sugerirá dos posibles paquetes para él. Uno es androidx.fragment.app.Fragment (este es el que queremos), y el otro es android.support.v4.app.Fragment (no lo necesitamos). El último paquete es solo en caso de que intente ejecutar esta aplicación en versiones de Android debajo de la API 11, pero el min SDK de nuestra aplicación es la API 26, por lo que no necesitamos la biblioteca de soporte para v4.

3. Crea un nuevo archivo de diseño; este será el archivo de diseño para la clase Fragment. Puedes hacer esto desde la ventana de la herramienta del proyecto. Haz clic con el botón derecho en la carpeta res, Nuevo ➤ Archivo de recursos de diseño. Nombra el nuevo archivo fragment_a, deja el elemento raíz predeterminado y asegúrate de que esté en el directorio "layout" (ver las figuras 10.19 y 10.20).

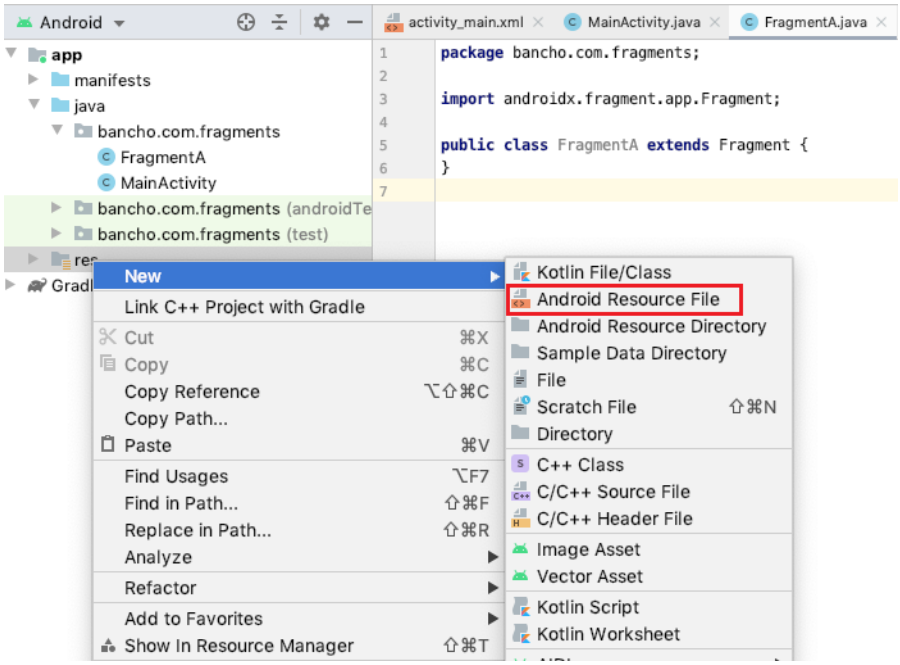


Figura 10.19. Crear un nuevo archivo de recursos de diseño.

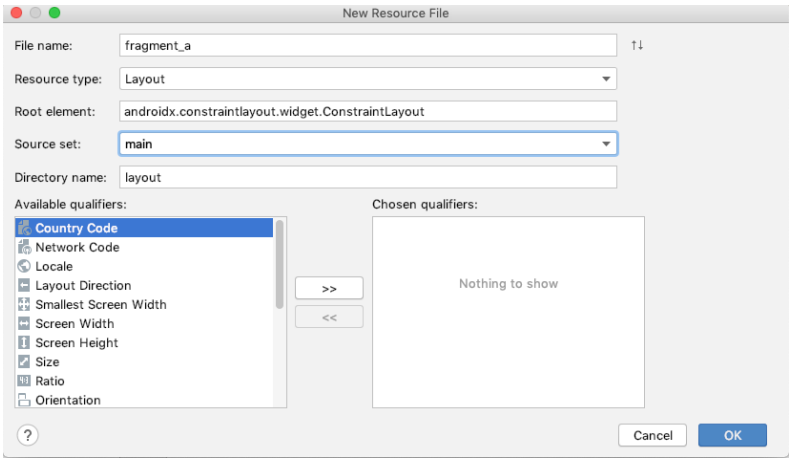


Figura 10.20: Nuevo diálogo de archivo de recursos de diseño para fragment_a.

4. Pon un objeto TextView en fragment_a. Un archivo de recursos de fragmento no es muy diferente del archivo de recursos de nuestra actividad principal. Ambos son grupos de vistas, y como tales, están destinados a contener otros objetos de vista.

Entonces, todas las técnicas que hemos aprendido sobre cómo componer objetos de vista en un archivo de recursos de diseño también se aplican a los fragmentos. El fragmento usa un diseño de restricción (igual que activity_main), por lo que puedes usar el inspector de restricciones y el inspector de atributos para personalizar el aspecto del fragmento. La apariencia de TextView se ha modificado un poco en este ejemplo. El Listado 10.8 muestra el código XML para el recurso de fragmento, y la figura 10.21 muestra cómo se ve en el modo de diseño.

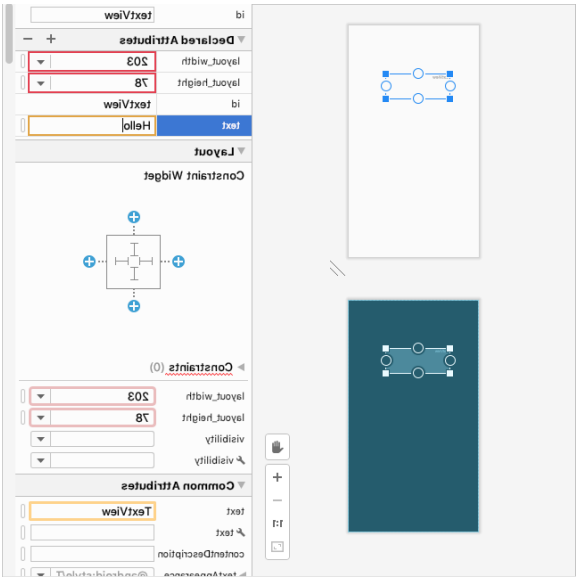


Figura 10.21: Archivo de diseño fragment_a.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView"
        android:layout_width="203dp"
        android:layout_height="78dp"
        android:layout_marginTop="94dp"
        android:layout_gravity="center"
        android:text="Hello"
        android:textSize="18sp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Listado 10.8: /app/res/layout/fragment_a.

5. Asocia el archivo de diseño de fragmento con la clase Fragment. En FragmentA.java, anularemos onCreateViewMethod e inflaremos el archivo de recursos de fragmento.

```
public class FragmentA extends Fragment {

    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, Bundle savedInstanceState){
        View v = inflater.inflate(R.layout.fragment_a, container,  attachToRoot: false);
        return v;
    }
}
```

Listado 10.9: El Método onCreateView de FragmentA.

Al igual que el método setContentView de Activity, inflate lee el archivo de recursos XML (primer parámetro), crea los objetos de Java reales para que podamos referenciarlos más adelante en la clase R. y luego adjunta los objetos Java creados al lugar donde está el fragmento. embedded (segundo parámetro): en este caso, el objeto contenedor es nuestra actividad. La última instrucción en la devolución de llamada es simplemente devolver el objeto View que ha creado el inflador.

6. Pondremos el archivo de recursos de fragmentos dentro del archivo de diseño activity_main, al igual que en otro objeto de vista (por ejemplo, TextView o Button). Abre activity_main en modo de diseño.
   Desde la paleta, ve a Layouts ➤ <fragment> (figura 10.22). O si no aparece, búscalo en la paleta.

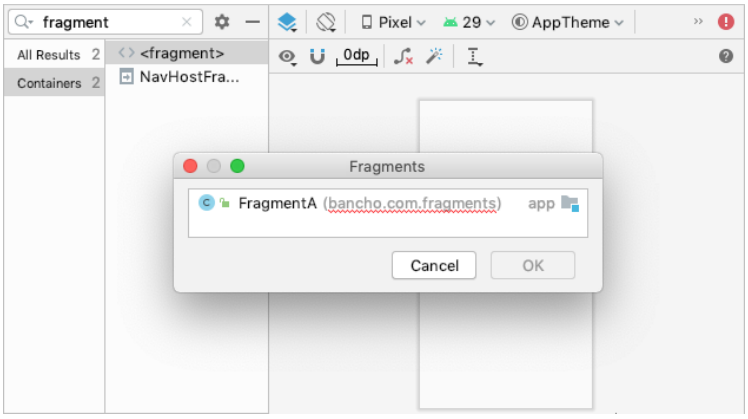


Figura 10.22: Incrustar el archivo de diseño de fragmento en activity_main.

7. Elige el Fragmento que creamos desde la ventana de diálogo (figura 10.23).

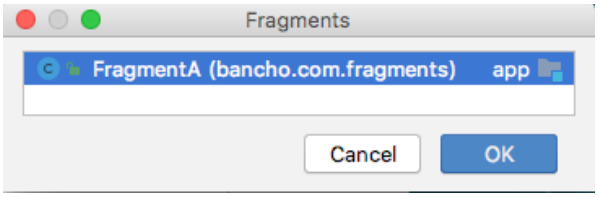


Figura 7.23. Elegir el fragmento.

8. El fragmento es un objeto ViewGroup, lo que significa que es solo otro objeto de vista. Puedes moverlo en activity_main como cualquier otro widget. Moverlo a su posición aproximada y usar las herramientas en el inspector de restricciones para arreglar su posición (figura 10.24).

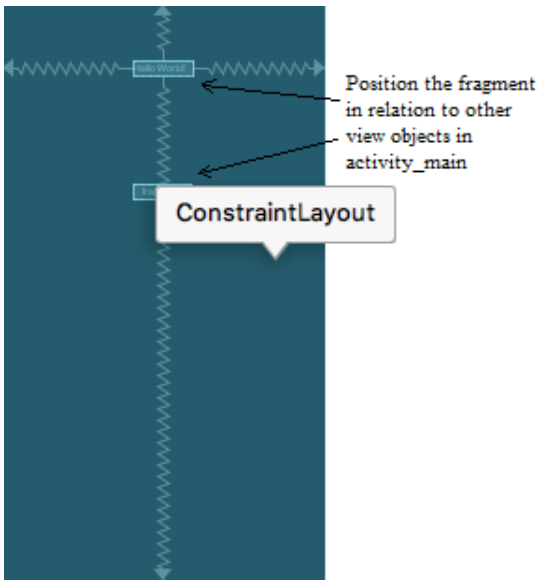


Figura 10.24: activity_main con el archivo de diseño de fragmento.

Si ejecutas la aplicación, deberías ver algo como la figura 10.25.

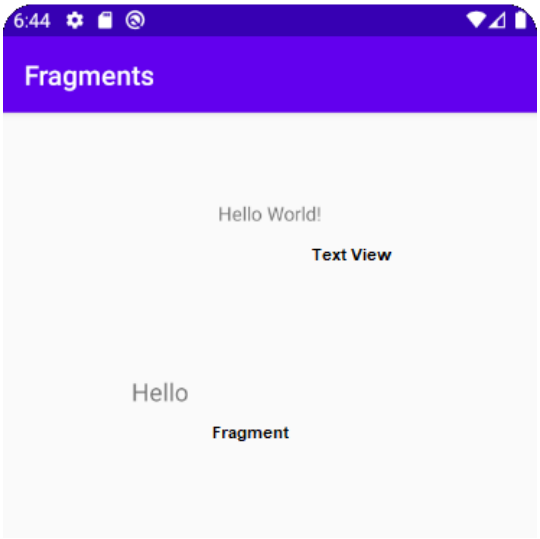


Figura 10.25: Fragments app, ejecutándose.

Los listados completos de códigos para FragmentA, activity_main.xml y MainActivity se muestran en los listados 10.10, 10.11 y 10.12, respectivamente.
El listado completo del código para fragment_a se muestra en el listado 10.10.

```java
package bancho.com.fragments;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;

public class FragmentA extends Fragment {

    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, Bundle savedInstanceState){
        View v = inflater.inflate(R.layout.fragment_a, container,  attachToRoot: false);
        return v;
    }
}
```

Listado 10.10: Listado completo del código para FragmentA.java.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout 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"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="bancho.com.fragments.MainActivity">

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.101"/>

    <fragment
        android:id="@+id/fragment"
        android:name="bancho.com.fragments.FragmentA"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="15dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/textView2"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Listado 10.11: Listado completo para activity_main.

```java
package bancho.com.fragments;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Listado 10.12: Listado de Código para MainActivity.java.

No tuvimos que hacer nada en nuestro MainActivity.java porque el fragmento se agregó de forma estática o declarativa. Lo que hicimos fue simplemente insertar el fragmento en activity_main, ya que era simplemente otro objeto View.

Adición de fragmentos mediante programación

Si bien podemos construir IU con fragmentos durante el tiempo de diseño, al agregar fragmentos en el tiempo de ejecución, nuestra aplicación es más responsiva. Puedes ocultar o mostrar fragmentos según el tamaño de pantalla del dispositivo o su orientación actual. Para agregar fragmentos en tiempo de ejecución, necesitaremos los objetos FragmentManager y FragmentTransaction.

Un objeto de transacción de fragmento es el responsable de agregar y eliminar fragmentos de una actividad, y para obtener una transacción de fragmento, necesitamos un administrador de fragmentos.
Vamos a crear un nuevo proyecto para este ejercicio para que puedas guardar el proyecto anterior como referencia. Ver la Tabla 10.4 para los detalles del proyecto.

Application name	**Fragments2**
Project location	Usa el predeterminado.
Form factor	Solo teléfono y tableta.
Minimum SDK	API 26 Oreo
Type of activity	Vacío
Activity name	MainActivity (predeterminado)
Layout name	activity_main (predeterminado)

Tabla 10.4: Fragments2 App, detalles del proyecto.

Este proyecto usará los mismos archivos que usamos en el proyecto anterior (Fragmentos). Antes de que podamos continuar, debes hacer lo siguiente.

Crea una clase fragment, FragmentA.java (igual que en el proyecto anterior).
Crea un archivo de diseño para el fragmento, fragment_a (igual que en el proyecto anterior), pero no agregues el diseño de fragmento en activity_main. En cambio, agregaremos el fragmento desde MainActivity.java

Los códigos para fragment_a.xml, FragmentA.java y activity_main.xml se muestran en los listados 10.13, 10.14 y 10.15, respectivamente.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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"

    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView"
        android:layout_width="203dp"
        android:layout_height="78dp"
        android:layout_marginTop="94dp"
        android:layout_gravity="center"
        android:text="Hello"
        android:textSize="18sp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Listado 10.13: fragment_a.

```java
package bancho.com.fragments2;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;

public class FragmentA extends Fragment {

    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, Bundle savedInstanceState){
        View v = inflater.inflate(R.layout.fragment_a, container, attachToRoot: false);
        return v;
    }
}
```

Listado 10.14: FragmentA.java.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout 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"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="bancho.com.fragments.MainActivity">

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.101"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Listado 10.15: activity_main.

Ahora que la configuración está lista, podemos comenzar a agregar el fragmento a la Actividad en tiempo de ejecución. Estos códigos residirán en MainActivity.java.

```java
package bancho.com.fragments2;

import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentManager;
import androidx.fragment.app.FragmentTransaction;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

        FragmentA f = new FragmentA();                          (1)
        FragmentManager fm = getSupportFragmentManager();       (2)
        FragmentTransaction ft = fm.beginTransaction();         (3)
        ft.add(R.id.frag_placeholder, f, tag: "");              (4)
        ft.commit();                                            (5)
    }
}
```

Listado 10.16: MainActivity.java.

(1) Crear una instancia de FragmentA.

(2) Obtener el administrador de fragmentos para esta actividad (MainActivity).

(3) Cuando un administrador de fragmentos comienza una transacción, esa llamada devolverá un objeto TransactionManager.

(4) Ahora podemos agregar nuestro fragmento durante el tiempo de ejecución utilizando el método add. Este método tiene tres parámetros, pero solo los dos primeros son importantes para nosotros. El primer parámetro es un ID de recurso de vista (no hemos creado eso todavía, lo crearemos en el Listado 10.17), y el segundo parámetro es la instancia de nuestra clase de fragmento (FragmentA).

(5) Deberíamos llamar al método commit() para que el fragmento sea visible en la actividad.

Importante. Al agregar un fragmento durante el tiempo de ejecución, el diseño activity_main DEBE tener un objeto de vista dentro que actuará como un marcador de posición para el fragmento. En nuestro ejemplo, esta es una vista de diseño de marco (id:frag_placeholder).

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout 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"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="bancho.com.fragments.MainActivity">

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="245dp"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/frag_placeholder" />

    <FrameLayout
        android:id="@+id/frag_placeholder"
        android:layout_width="368dp"
        android:layout_height="0dp"
        android:layout_marginBottom="8dp"
        android:layout_marginTop="8dp"
        app:layout_constraintBottom_toTopOf="@+id/textView2"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent">

    </FrameLayout>

</androidx.constraintlayout.widget.ConstraintLayout>
```

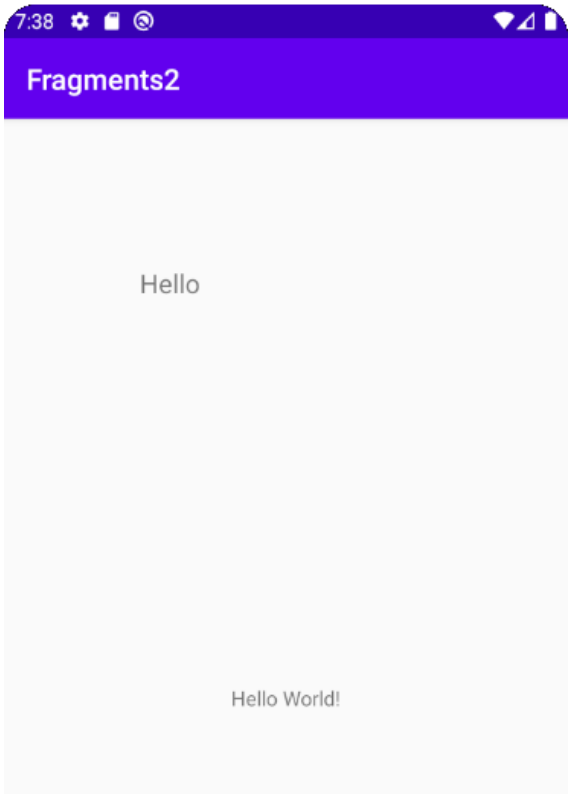
Listado 10.17: activity_main.xml.

(1) Esto se convertirá en un marcador de posición para nuestro fragmento. El contenido de nuestros fragmentos no se puede ver en modo de diseño porque el diseño del marco no contiene nada. Puedes ver el contenido del fragmento en tiempo de ejecución.

(2) Este es el ID que usaremos cuando el objeto de transacción de fragmento agregue el fragmento.

Esto concluye el tema sobre elementos de la IU. Apenas arañamos la superficie del diseño de la interfaz de usuario, pero agregar temas y colores, usar AppBar y el uso juicioso de Fragments debería agregar algo de brillo a tus aplicaciones.

Nota importante: Chequen algunas opciones en la web si hubiera algún detalle en la compilación, puede ser que ya se hayan obsoletizado algunos elementos de Android.


7:38
Fragments2
Hello
Hello World!