Tutorial Android Studio: Agregar Notificaciones de firebase

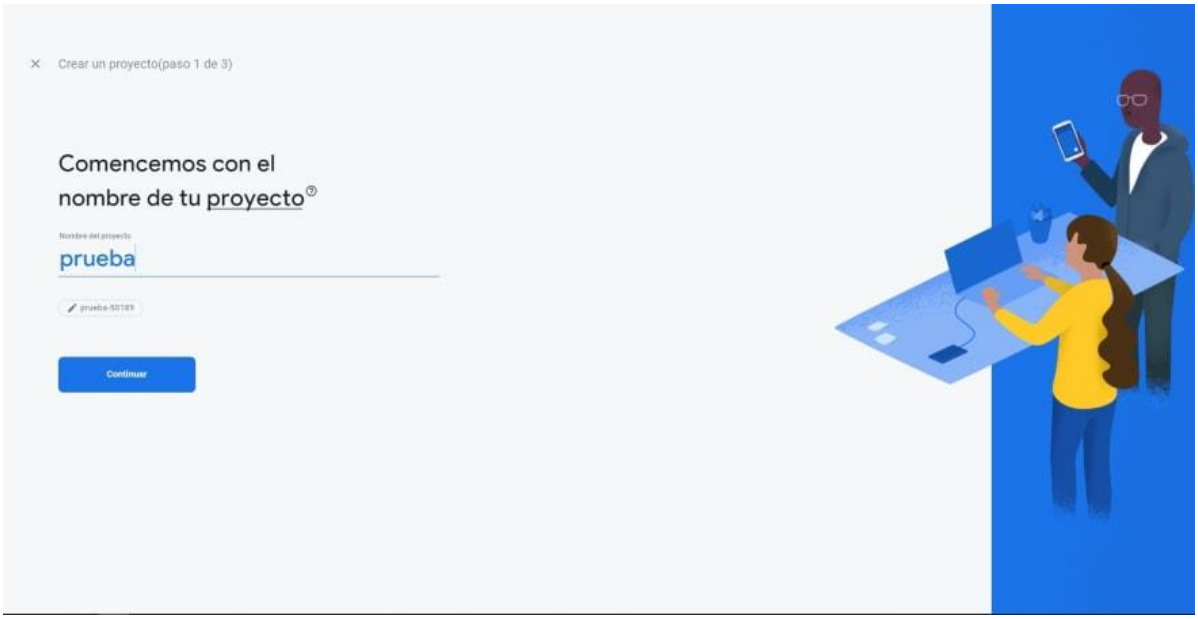
#firebase #android #minitutorial #espanol

Notificaciones en android studio con firebase (2 Part Series)

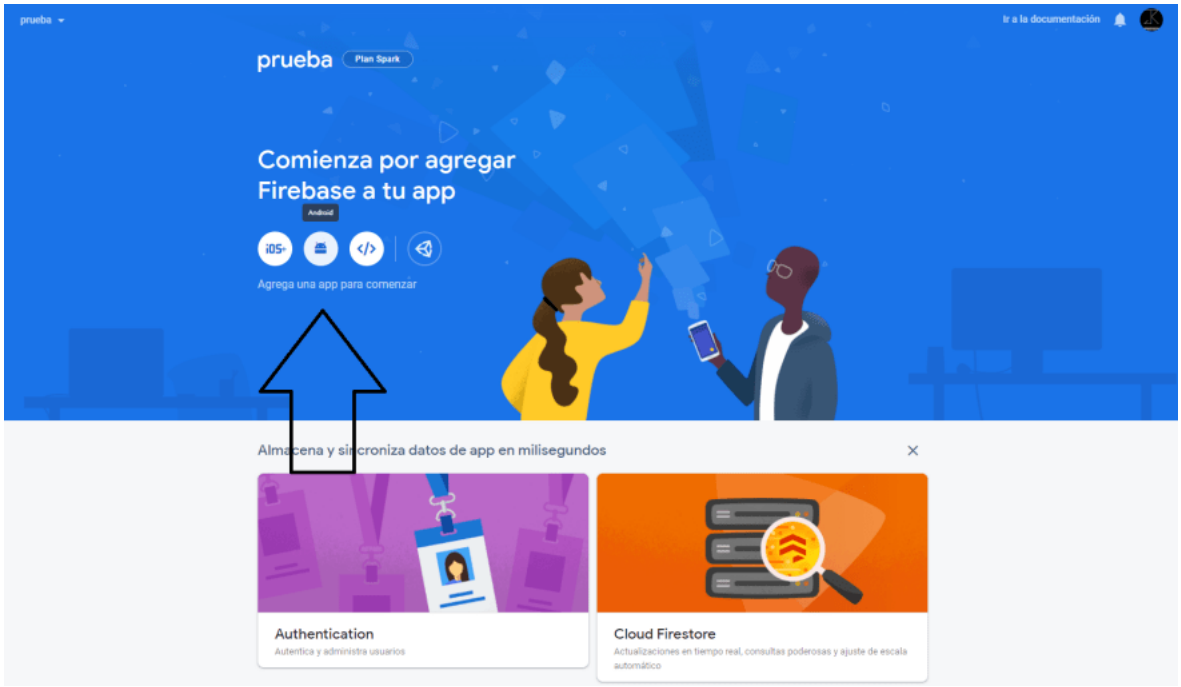
1 Tutorial Android Studio: Agregar Notificaciones de firebase 2 Tutorial Android Studio: Obtener token de mi dispositivo

Firebase

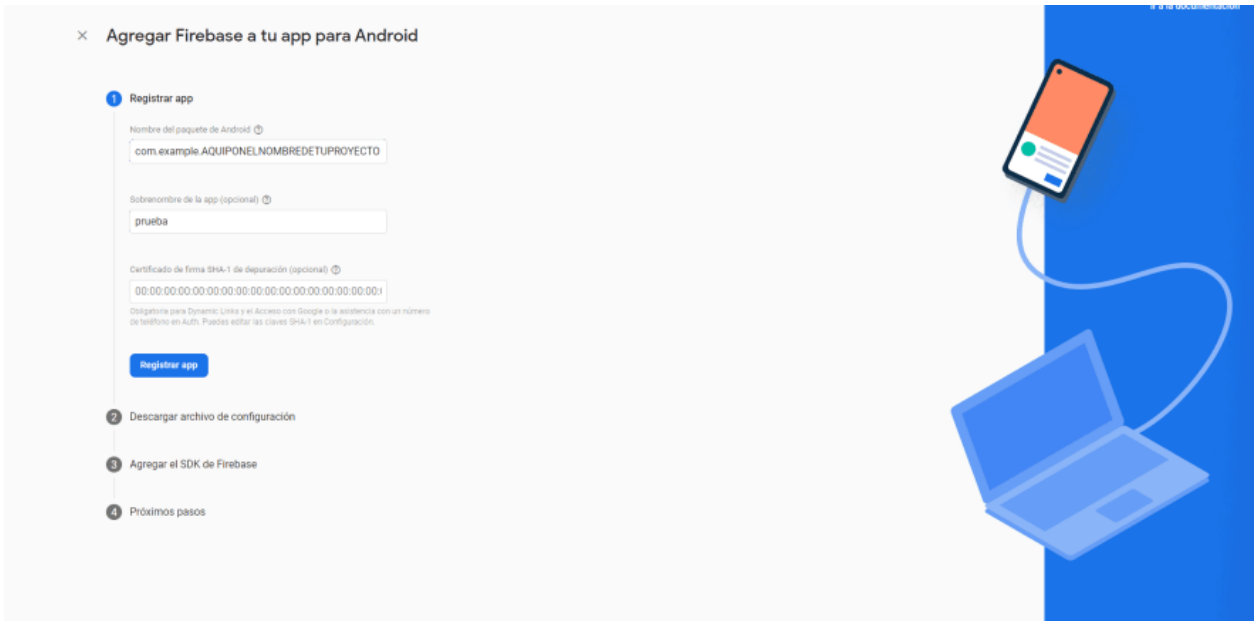
Primero necesitamos un proyecto en firebase para eso iremos a su página web llamada firebase console click aquí y creamos un nuevo proyecto y le asignamos un nombre



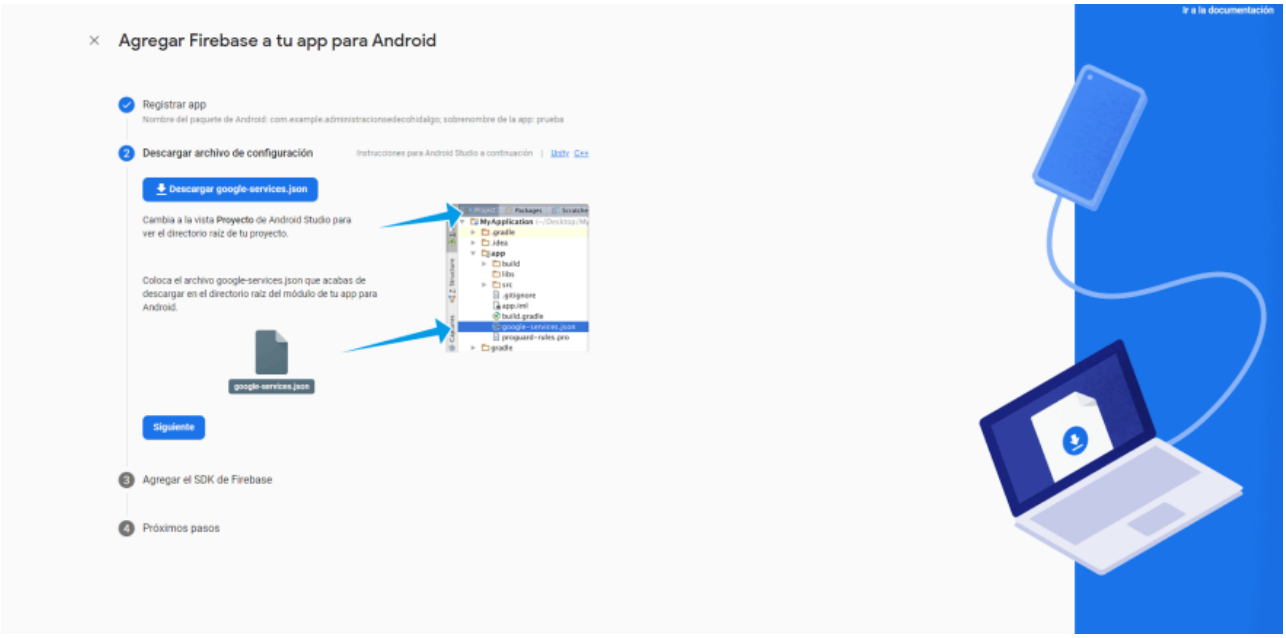
Después agregaremos una app a nuestro proyecto creado tipo android



Después de seleccionar la opción Android debemos de agregar el nombre de nuestro proyecto y si desean pueden poner un nombre a tu proyecto.



Descargamos el archivo google.service.json y lo agregamos en nuestra aplicacion en la carpeta app para eso debemos seleccionar la opcion de ver Project



En la misma carpeta App en el archivo build.gradle agregamos dos apply

```
apply plugin: 'com.android.application'
apply plugin: 'com.google.gms.google-services'
```

Quedando de la siguiente manera



En la sección de dependencias del mismo archivo build_gradle
agregamos lo siguiente:

```
implementation platform('com.google.firebase:firebase-bom:29.0.1')
implementation 'com.google.firebase:firebase-bom:29.0.1'
implementation 'com.google.firebase:firebase-messaging:23.0.0'
implementation 'com.google.firebase:firebase-analytics'
```

Quedando de la siguiente manera



Y le damos en sincronizar.

MyFirebaseMessagingService

Cambiaremos de vista de **Project** a **Android** y creamos una Java Class llamada **MyFirebaseMessagingService** e ingresamos el siguiente código.

```
package com.example.administracionsedecohidalgo;

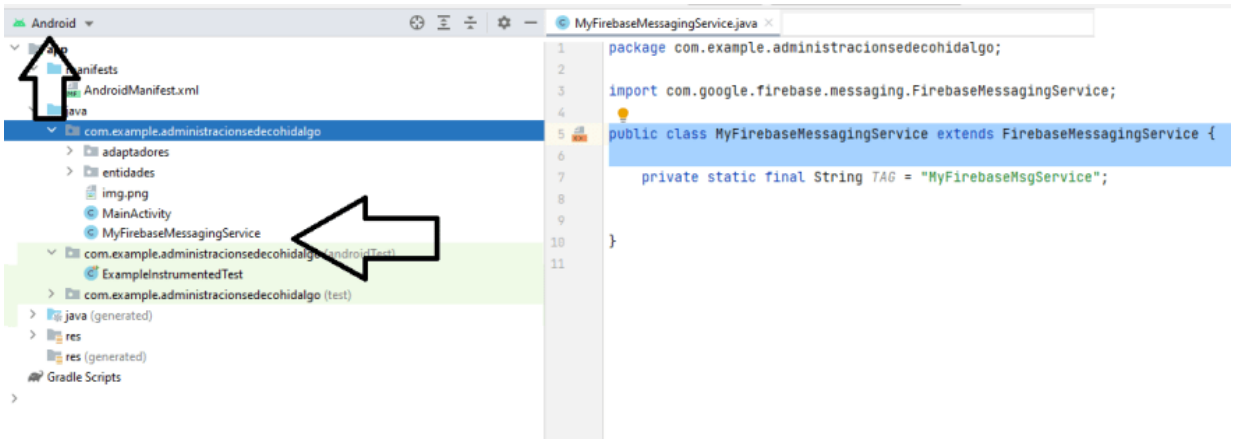
import com.google.firebase.messaging.FirebaseMessagingService;

public class MyFirebaseMessagingService extends
FirebaseMessagingService {

    private static final String TAG = "MyFirebaseMsgService";


}
```

Quedando de la siguiente manera:

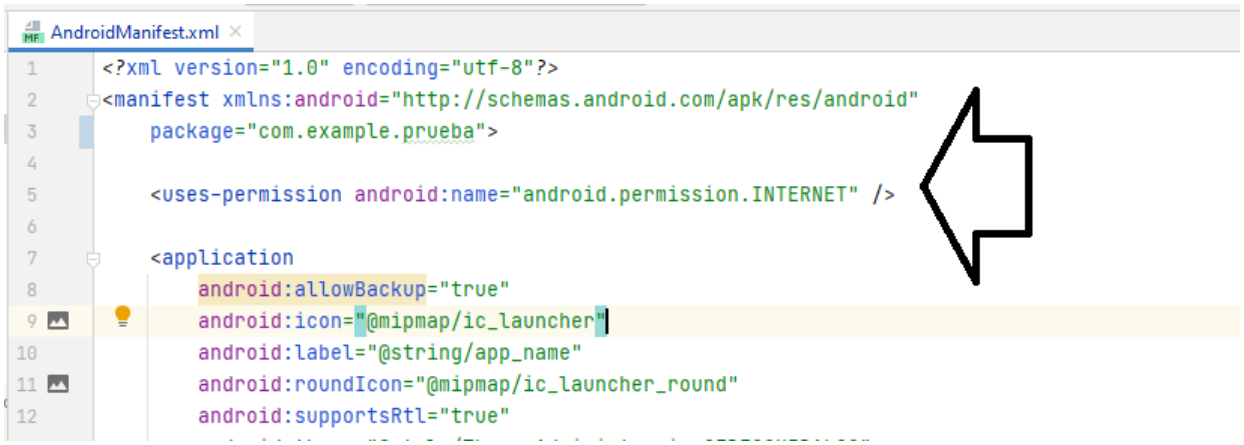


AndroidManifest.xml

En este archivo **AndroidManifest.xml** agregaremos permisos de Internet debajo de donde aparece la etiqueta **package**

```
<uses-permission android:name="android.permission.INTERNET" />
```

Quedando el código de la siguiente manera:



En la etiqueta **<application** agregamos 2 etiquetas Meta-data

```
<meta-data
      android:name="firebase_messaging_auto_init_enabled"
      android:value="false" />
<meta-data
      android:name="firebase_analytics_collection_enabled"
      android:value="false" />
```

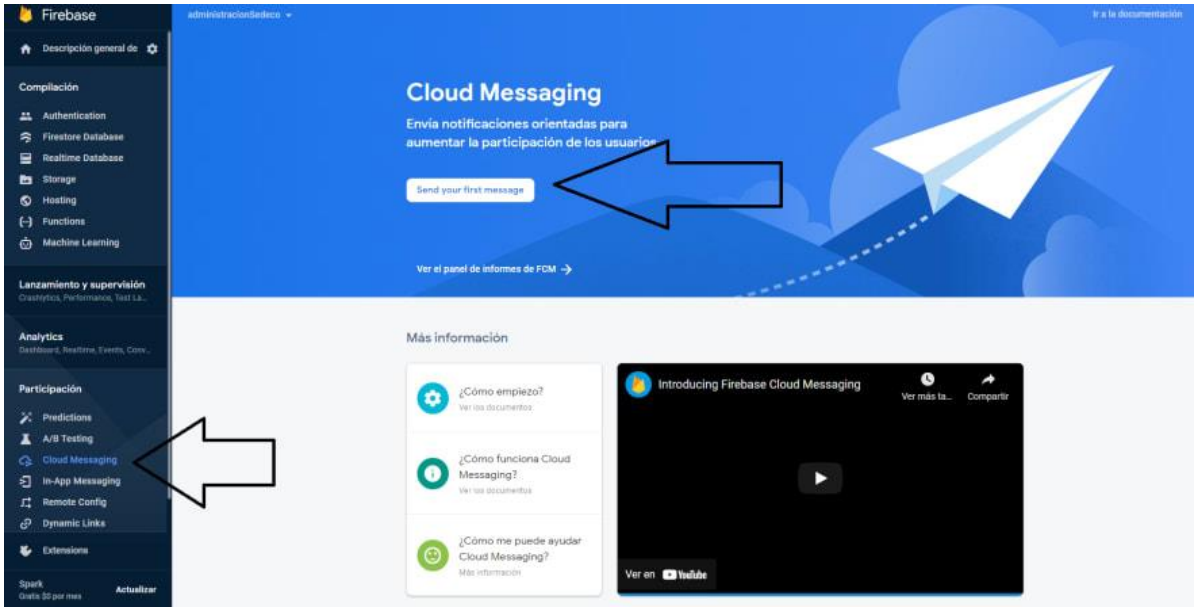


Antes de cerrar la etiqueta agregamos el siguiente código el cual manda llamar la clase MyFirebaseMessagingService que creamos anteriormente:

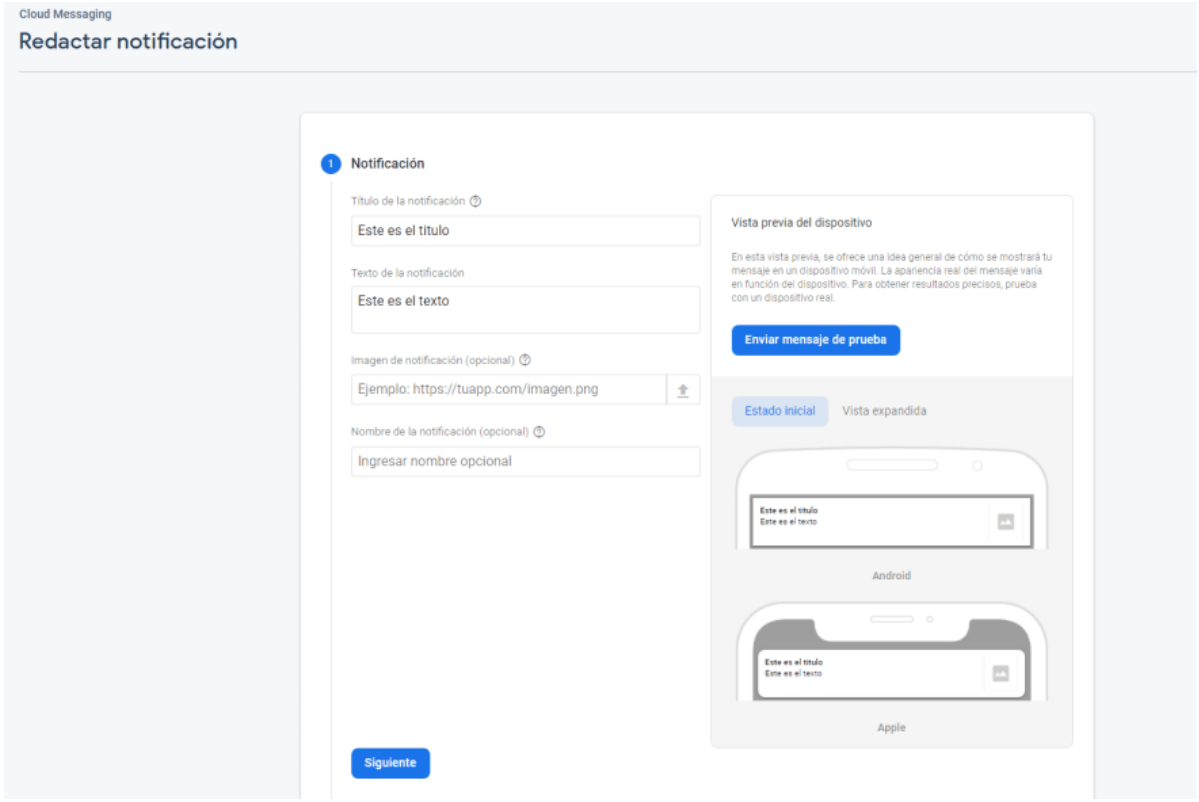
```
        <service
            android:name=".MyFirebaseMessagingService"
            android:exported="false">
            <intent-filter>
                <action
android:name="com.google.firebase.MESSAGING_EVENT" />
            </intent-filter>
        </service>
```



Eso es todo lo que tenemos que agregar en nuestra aplicación de Android studio. Solo nos resta ir a firabase en la sección Cloud Messaging y en la opción **Send your first message**

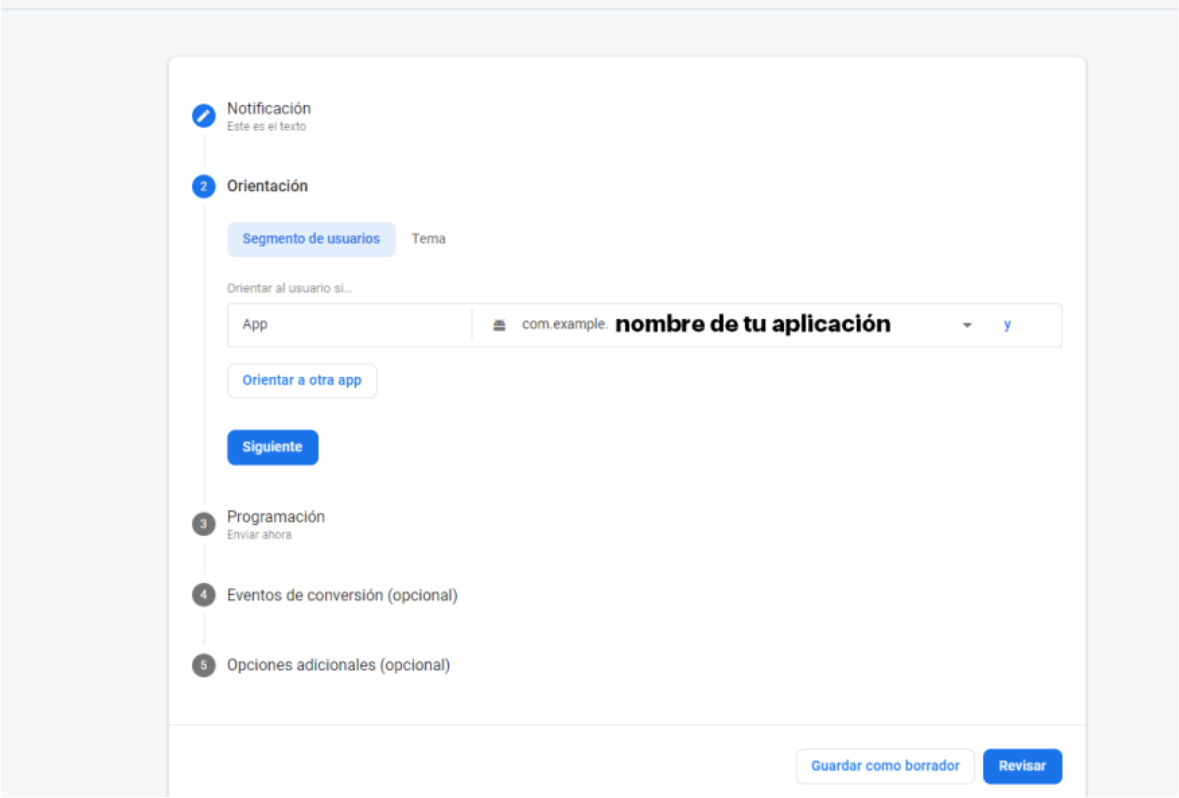


Al darle clic nos mostrará una ventana para redactar la notificación

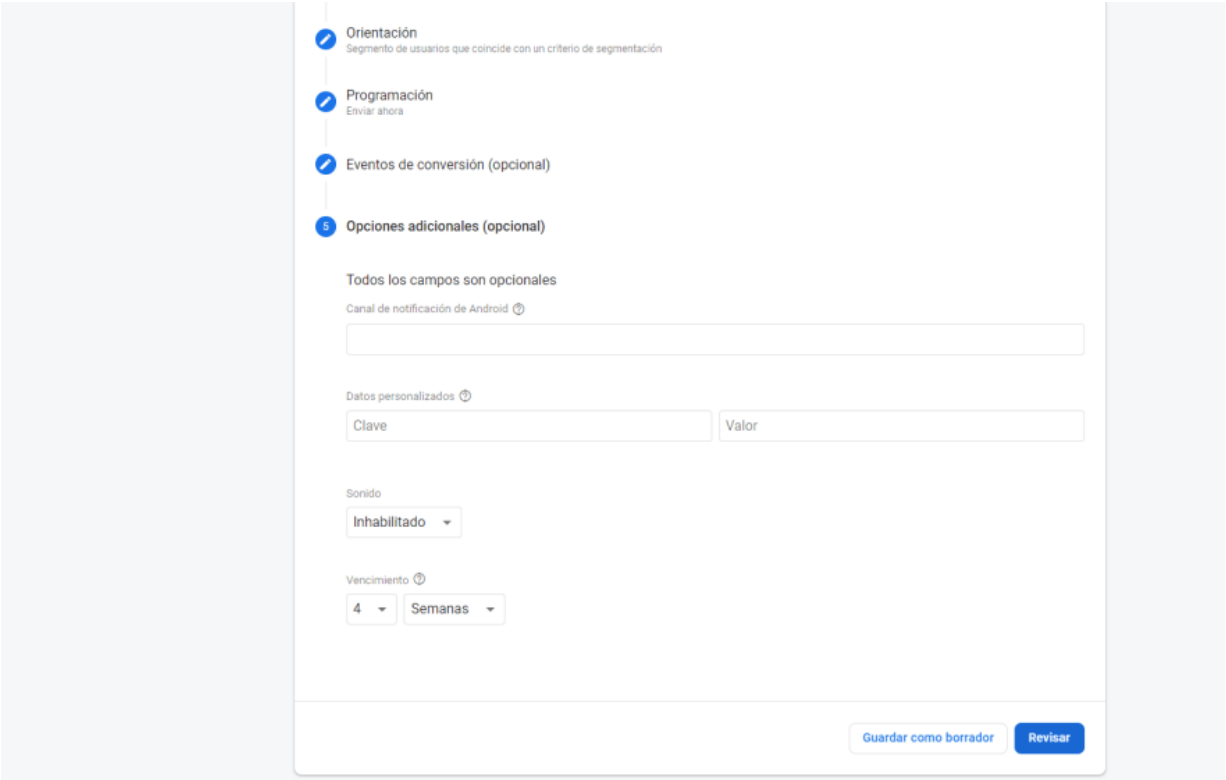


Entonces escribimos un titulo y texto que queramos agregar y le damos en siguiente.

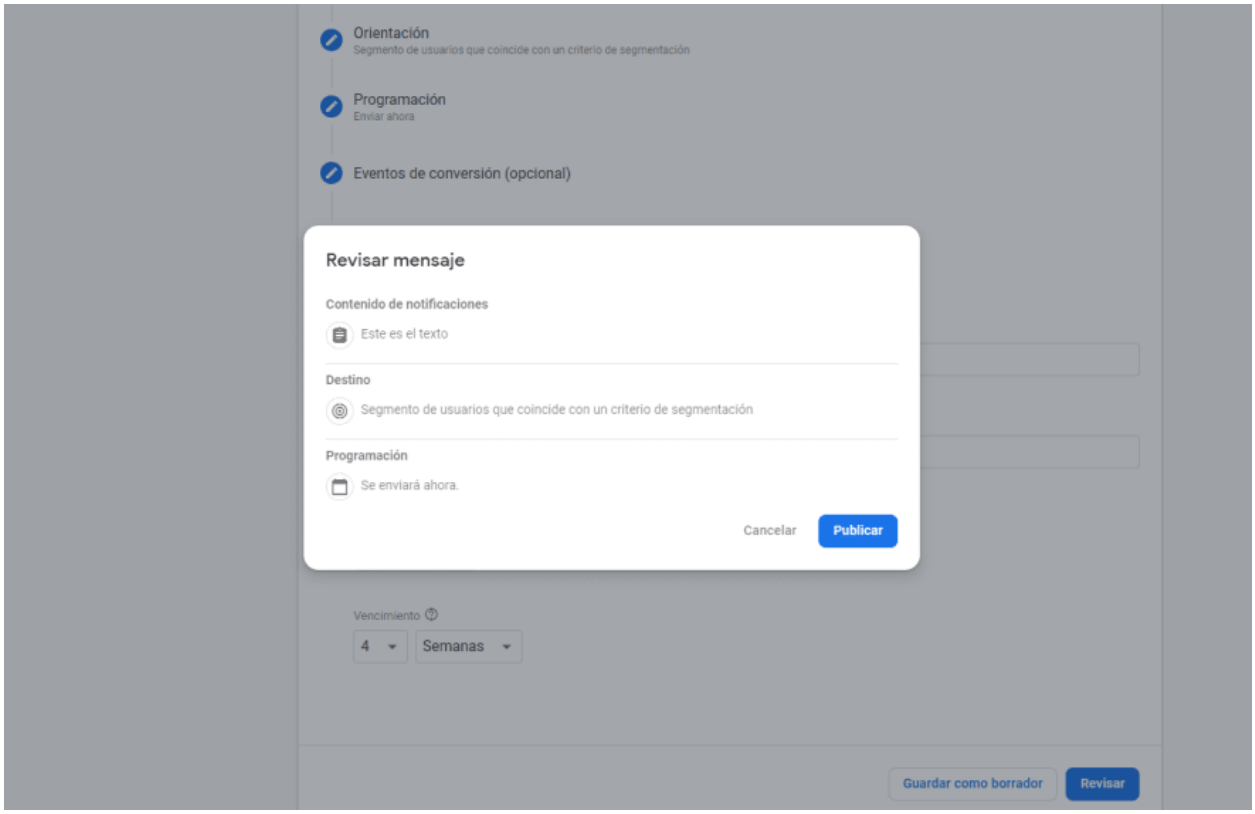
Posteriormente selecciona la aplicación que hemos creado en firabase y le damos en siguiente



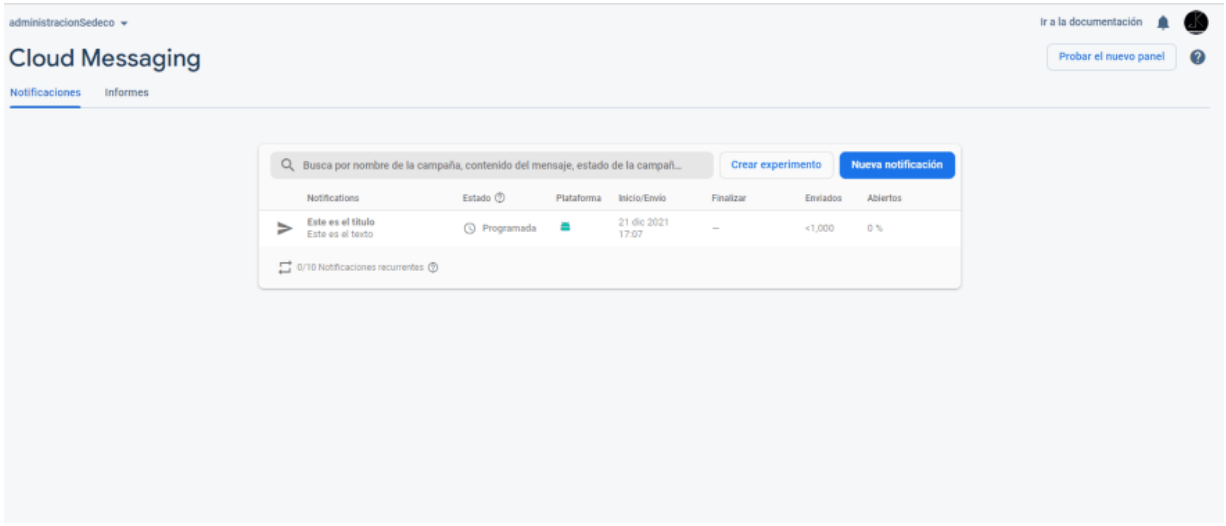
Todos los demás campos de los siguientes pasos los dejaremos como están solo daremos siguiente, siguiente, hasta llegar al ultimo paso



Damos clic en revisar

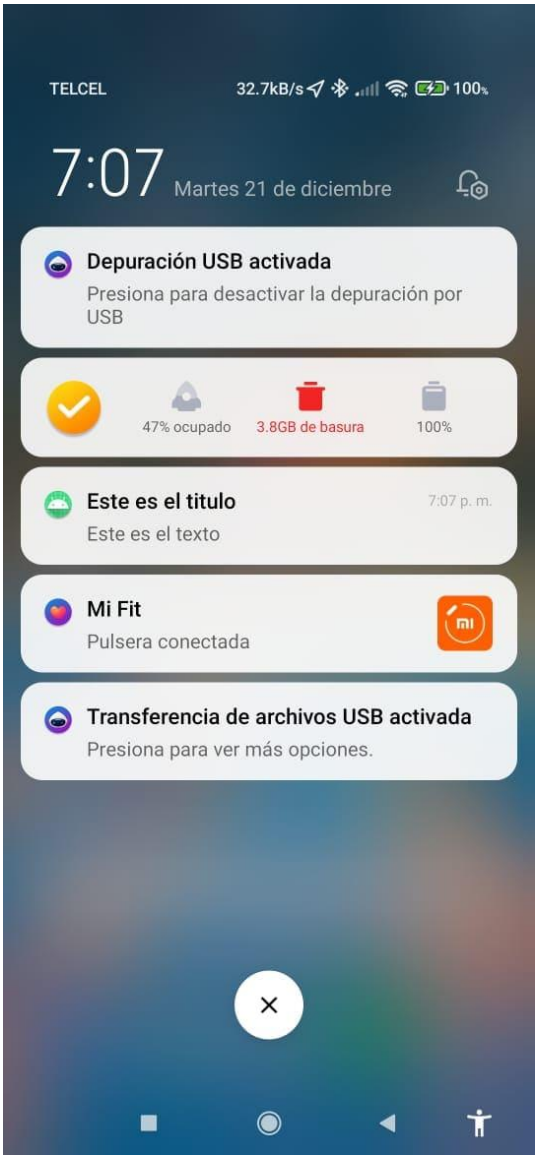


Y en publicar



ESO ES TODO!!

Nos tiene que llegar una notificación en nuestro celular.

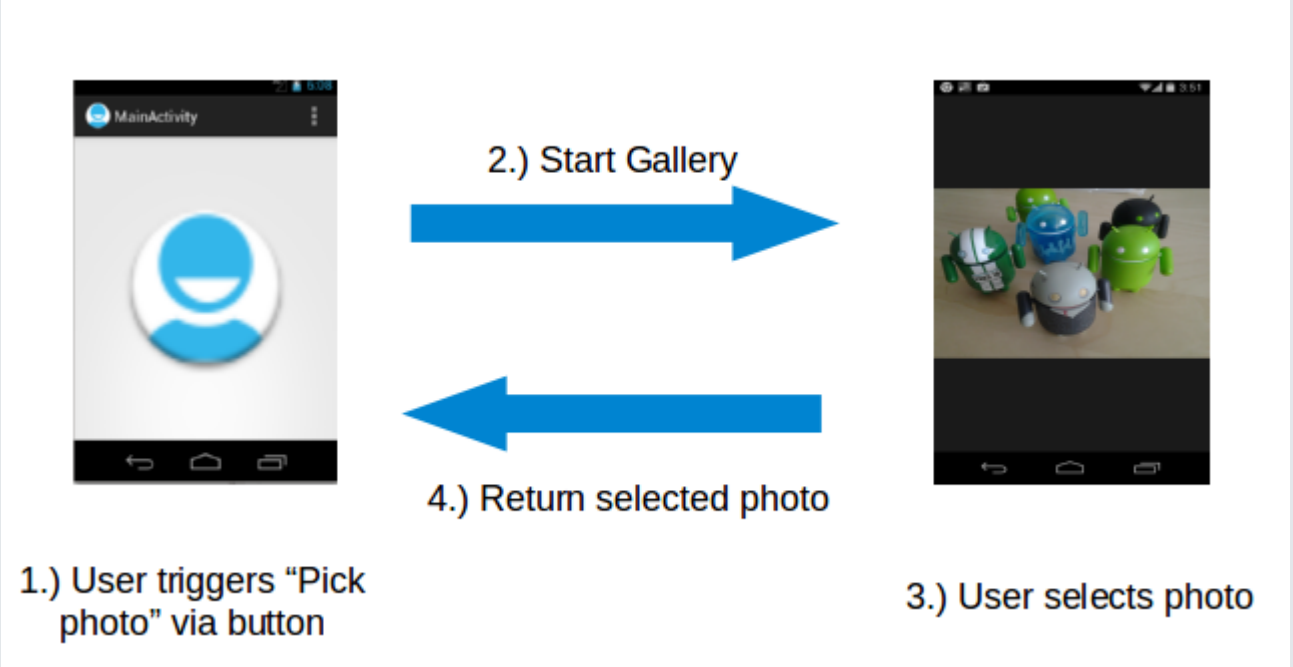

TELCEL
32.7kB/s
100%
7:07
Martes 21 de diciembre
Depuración USB activada
Presiona para desactivar la depuración por USB
47% ocupado
3.8GB de basura
100%
Este es el titulo
7:07 p. m.
Este es el texto
Mi Fit
Pulsera conectada
Transferencia de archivos USB activada
Presiona para ver más opciones.

INTENTS EN ANDROID

INSTRUCTOR: Gustavo Salazar

¿Qué Son Los Intents de Android?

- Los Intents son un tema fundamental para desarrolladores Android. Es imposible construir aplicaciones Android sin entrar en contacto con Intents.



Los Intents

- En Android, la habilidad de enviar mensajes es posible por el objeto Intent. Con la ayuda de los intents, los componentes de Android pueden solicitar funcionalidad de otros componentes Android. Cuando abres la aplicación Instagram en tu teléfono y la usas para tomar una foto, acabas de hacer uso de un intent. Los Intents también ayudan a comunicar entre partes de una app; el movimiento de una pantalla (actividad) a otra es posible mediante intents.
- Míralo de esta forma: todos los componentes (aplicaciones y pantallas) del dispositivo Android están aisladas. La única manera de comunicarse entre ellas es a través de intents.

¿Qué información contiene un Intent?

- Básicamente, portará información que el sistema operativo utilizará para determinar qué componente lanzar, así como cualquier información adicional que la componente pueda necesitar para poder llevar a cabo su acción.

La información principal que podemos encontrar en un Intent es la siguiente:

* **Component Name** (Nombre de componente):

En este campo podremos indicar el nombre del componente que queremos lanzar. Es opcional, pues no siempre conoceremos el componente a lanzar (imagina que queremos lanzar la cámara, pero no sabemos qué actividad utilizar…). Sólo será obligatorio en el caso de un servicio.

* **Data** (Datos): Será una *URI* que haga referencia a los datos que queremos utilizar, así como el tipo de datos que son (*MIME type*).

* **Category** (Categoría): Es una cadena de texto que ofrece información adicional sobre el tipo de componente que debería ser lanzado.

* **Extras** (Extras): Aquí podremos ingresar tantos pares llave-valor como queramos como información adicional que queremos enviar al componente a lanzar.

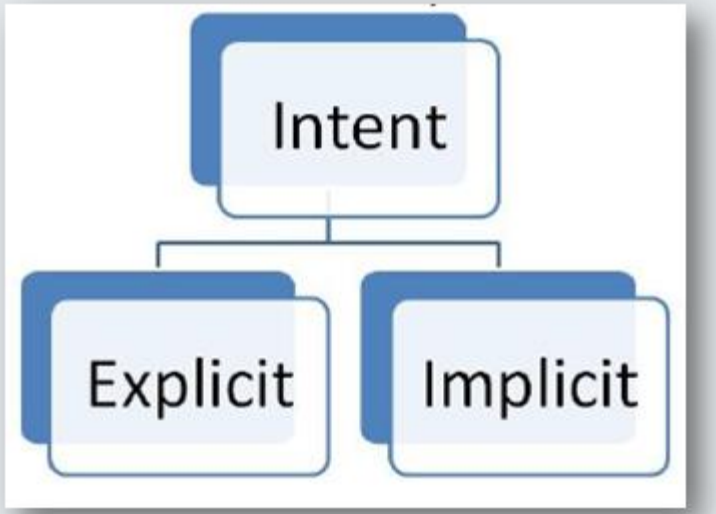
* **Flags** (Banderas): Son metadatos que ayudan a indicar al sistema cómo lanzar, por ejemplo, nuestra actividad.

Constant	Target component	Action
`ACTION_CALL`	activity	Initiate a phone call.
`ACTION_EDIT`	activity	Display data for the user to edit.
`ACTION_MAIN`	activity	Start up as the initial activity of a task, with no data input and no returned output.
`ACTION_SYNC`	activity	Synchronize data on a server with data on the mobile device.
`ACTION_BATTERY_LOW`	broadcast receiver	A warning that the battery is low.

Constant	Meaning
`CATEGORY_BROWSABLE`	The target activity can be safely invoked by the browser to display data referenced by a link — for example, an image or an e-mail message.
`CATEGORY_GADGET`	The activity can be embedded inside of another activity that hosts gadgets.
`CATEGORY_HOME`	The activity displays the home screen, the first screen the user sees when the device is turned on or when the *Home* button is pressed.
`CATEGORY_LAUNCHER`	The activity can be the initial activity of a task and is listed in the top-level application launcher.
`CATEGORY_PREFERENCE`	The target activity is a preference panel.

Tipos de Intents

- Android soporta dos tipos de intents: explícito e implícito. Cuando una aplicación define su componente meta en un intent, ese es un intent explícito. Cuando las aplicaciones no nombran un componente meta, ese es un intent implícito.



Ejemplo de Intent Explícito

- El código de arriba es un ejemplo de un intent explícito. Échale un vistazo nuevamente.

```
Intent numbersIntent = new Intent(MainActivity.this, NumbersActivity.class);

startActivity(numbersIntent);
```

- Aquí, NumbersActivity es el componente meta para nuestro MainActivity. Esto significa que NumbersActivity es el componente definido que será llamado por el sistema Android. Es importante notar (como en el ejemplo de arriba), que los intents explícitos son usados típicamente dentro de una aplicación, porque eso le da al desarrollador mayor control sobre cuál clase será lanzada.

Ejemplo de Intent Implícito

- Aquí hay un intent implícito:

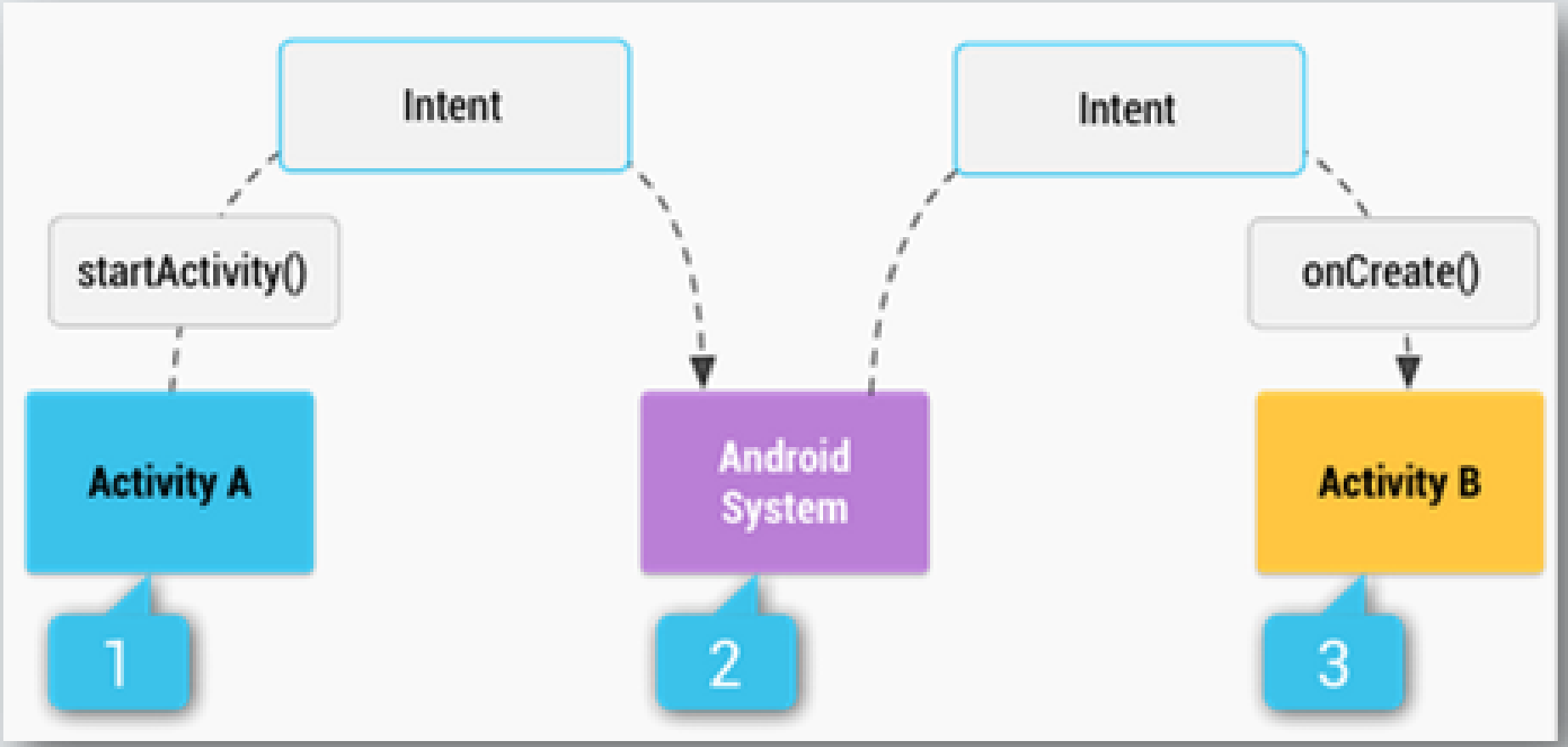
```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("https://www.tutsplus.com"));
startActivity(intent);
```

- Si tienes el código de arriba en tu base de código, tu aplicación puede comenzar un componente de navegador para una cierta URL vía un intent. ¿Pero cómo identifica el sistema Android los componentes que pueden reaccionar a un cierto intent?

Ejemplo de Intent Implícito

Un componente puede ser registrado vía un *filtro intent* para una acción específica. Los filtros intent pueden ser registrados para componentes de manera estática en el **AndroidManifest.xml**. Aquí hay un ejemplo que registra un componente como un visor web:

```
<activity android:name=".BrowserActivity"
        android:label="@string/app_name">
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:scheme="http"/>
  </intent-filter>
</activity>
```


Intent
Intent
startActivity()
onCreate()
Activity A
Android System
Activity B
1
2
3

¿cómo sabe Android qué acciones realiza cada componente de aplicación?

- Aquí es donde juega un papel crucial el fichero **Manifest**. En él, ya hemos comentado que debemos dejar registrados todos los componentes que forman parte de la aplicación. Pero no sólo eso. También deberemos añadir dentro de cada componente en el Manifest el concepto de **Intent Filter**. Este filtro nos permite indicar cuál es la capacidad de dicha componente.
- Concretamente, **un Intent Filter** puede contener los siguientes campos completos**:**

- **Acción**
- **Datos**
- **Categoría**

- Entonces, cuando nosotros lancemos un Intent implícito, **Android buscará en todos los Manifest de todas las aplicaciones instaladas aquellos componentes que cumplan los requisitos de nuestro Intent**: todos y cada uno de ellos.
- Es decir, tendremos que cumplir los requisitos de acción, categoría y datos.

Como crear un Intent

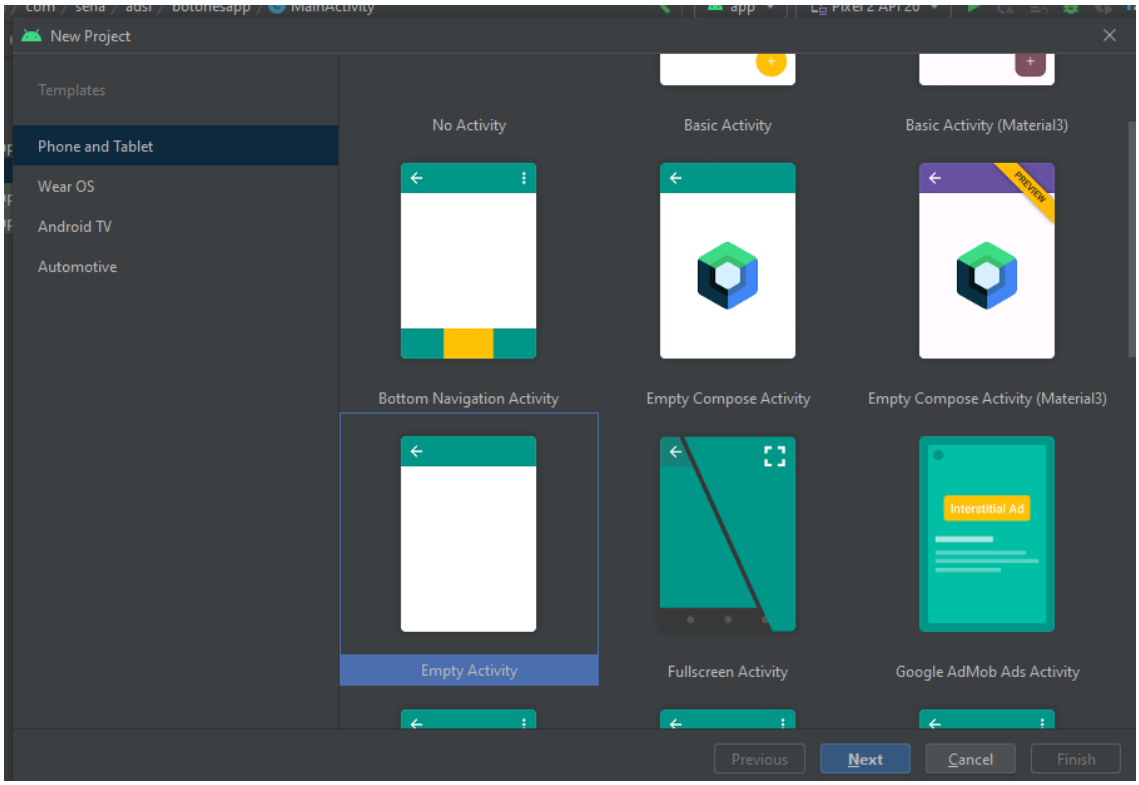
- Un objeto Intent tiene información que el sistema Android usa para determinar qué componente debe iniciar (como el nombre exacto del componente o la categoría que debe recibir la intent), además de información que el componente receptor usa para realizar la acción correctamente (por ejemplo, la acción que debe efectuar y los datos en los que debe actuar).

1. Implementar Un Botón En Android

Crear Proyecto en Android Studio

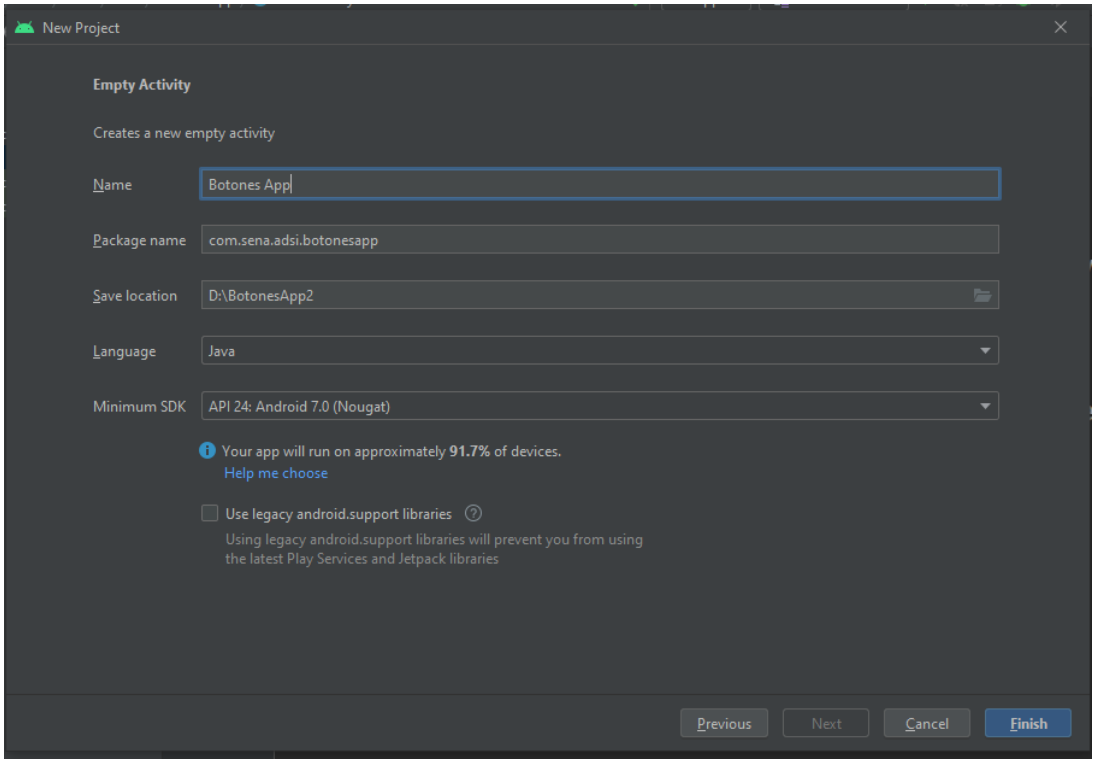
Antes que nada abre Android Studio, ve al menú de herramientas y selecciona **File > New > New Project…** para crear un nuevo proyecto.

En la siguiente venta elige una actividad vacía (**Empty Activity**) cómo contenedor principal para los ejemplos que veremos:



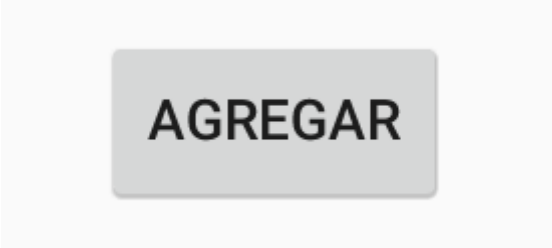
En la configuración del proyecto ten en cuenta los siguientes parámetros:

- **Application Name**: usa el nombre «Botones».
- **Company Domain Name**: Para el dominio elige la entidad que represente tu creación, puede ser *tu_nombre.com*, el *nombre_de_tu_empresa.com* o simplemente *test.com*.
- **Projection Location**: En la ubicación selecciona la carpeta que tengas destinada para guardar tus proyectos. Recuerda no usar espacios en ni caracteres especiales con acentos.



La Clase Button

Un botón es un control con texto o imagen que realiza una acción cuando el usuario lo presiona. La clase Java que lo represente es Button y puedes referirte a él dentro de un layout con la etiqueta `<Button>` respectivamente.



Para agregar un botón a la actividad principal de tu app dirígete a `actividad_botones.xml` y agrega la siguiente definición.

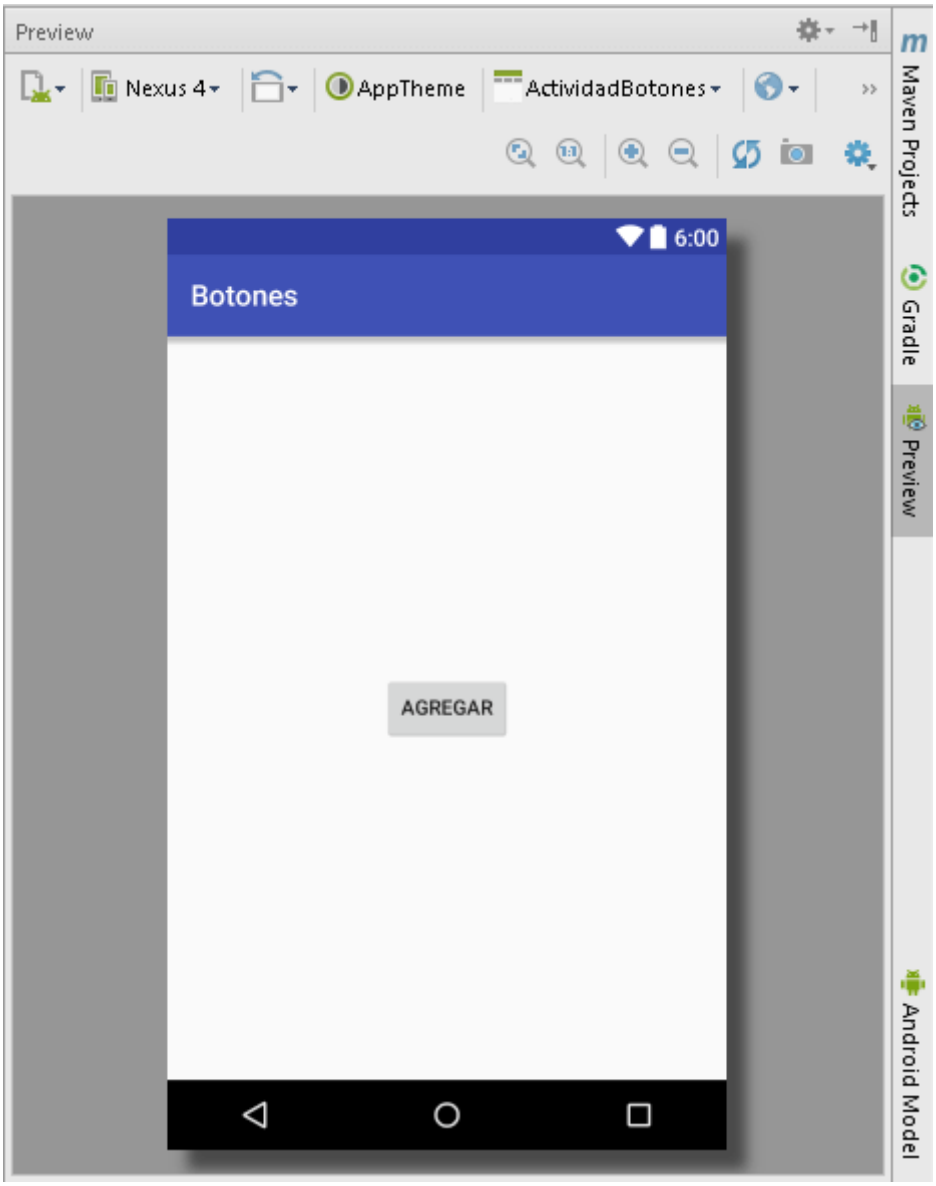
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"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

```
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="AGREGAR" />
</RelativeLayout>
```

Con ello centramos el botón en el relative layout.

El texto que especifica su acción especifícalo en el atributo `android:text`. En el ejemplo anterior se usó la palabra `"AGREGAR"`.

Este código debería mostrar en la ventana **Preview** la siguiente imagen:



Atributos de un Botón

Si quieres cambiar las propiedades de un botón recurre a los atributos que la documentación presenta en formato Java o XML.

Debido a que `Button` extiende de `TextView`, puedes usar todos los atributos de esta clase.

Algunos de los más utilizados son:

Atributo	Descripción
`android:text`	Permite cambiar el texto de un botón
`android:background`	Se usa para cambiar el fondo del botón. Puedes usar un recurso del archivo `colors.xml` o un *drawable*.
`android:enabled`	Determinar si el botón está habilitado ante los eventos del usuario. Usa `true` (valor por defecto) para habilitarlo y `false` en caso contrario.
`android:gravity`	Asigna una posición al texto con respecto a los ejes x o y dependiendo de la orientación deseada. *Por ejemplo*: Si usas `top`, el texto se alineará hacia el borde superior.
`android:id`	Representa al identificador del botón para diferenciar su existencia de otros views.
`android:onClick`	Almacena la referencia de un método que se ejecutará al momento de presionar el botón.
`android:textColor`	Determina el color del texto en el botón
`android:drawable*`	Determina un drawable que será dibujado en la orientación establecida. *Por ejemplo*: Si usas el atributo `android:drawableBottom`, el drawable será dibujado debajo del texto.

Teniendo en cuenta la anterior tabla te invito a ver los resultados que producen usar algunos de esos atributos.

Cambiar texto de un botón

Asigna una cadena directamente al atributo `android:text` para indicar la acción del botón.

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:text="Agregar" />
```

Por defecto el texto del botón estará en mayúsculas, pero si quieres deshabilitar esta característica usa el valor false en el atributo `android:textAllCaps`.

```
<Button
    ...
    android:textAllCaps="false"/>
```

Ahora, si deseas cambiar el texto programáticamente usa el método `setText()`.

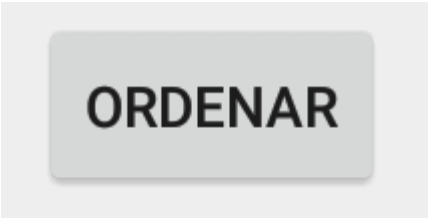
Para ello abre `ActividadBotones.java`. Obtén la instancia del botón con `findViewById()` y luego invoca `setText()` con una secuencia de caracteres como parámetro.

```
public class ActividadBotones extends AppCompatActivity {

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

        Button boton = (Button) findViewById(R.id.button);
        boton.setText("Ordenar");
    }
}
```

Esto produce el siguiente resultado:



Ahora también es posible asignar un recurso string al texto del botón desde XML y Java.

En caso de XML usa la notación de recurso `@string` o `@android:string` (strings del sistema) de la siguiente forma:

```
<Button
    ...
    android:text="@string/texto_agregar"/>
```

En Java solo usa el operador punto para llegar al identificador perteneciente a la clase R:

```
Button boton = (Button) findViewById(R.id.button);
boton.setText(R.string.texto_agregar);
```

Cambiar Color de Fondo

Modificar el fondo de un botón implica el cambio del atributo `android:background` por un recurso de color o drawable.

Ejemplo:

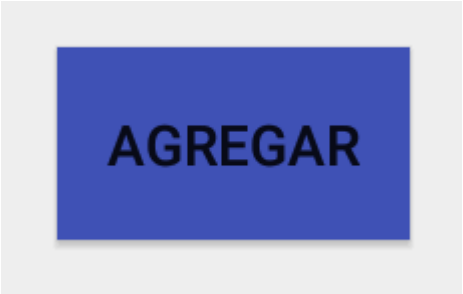
Usar el color primario del proyecto como color de background de un botón

Solución:

Invoca la referencia `@color/colorPrimary` de tu archivo **values/colors.xml**:

```
<Button
    ...
    android:background="@color/colorPrimary"/>
```

El resultado es:



Sin embargo, hacer esto hace perder la reacción de superficie que se tenía antes por el Material Design.

La forma redondeada tampoco se hace presente.

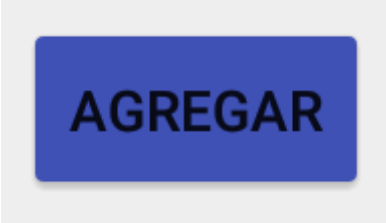
Solo queda la elevación al momento de tocar el botón.

Para conservar el efecto y cambiar el color del botón usa el atributo `app:backgrountTint` de la siguiente forma:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
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"
    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.herprogramacion.botones.ActividadBotones">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        app:backgroundTint="@color/colorPrimary"
        android:text="@string/texto_agregar" />
</RelativeLayout>
```

Así tendrías:



Botón con Texto e Imagen

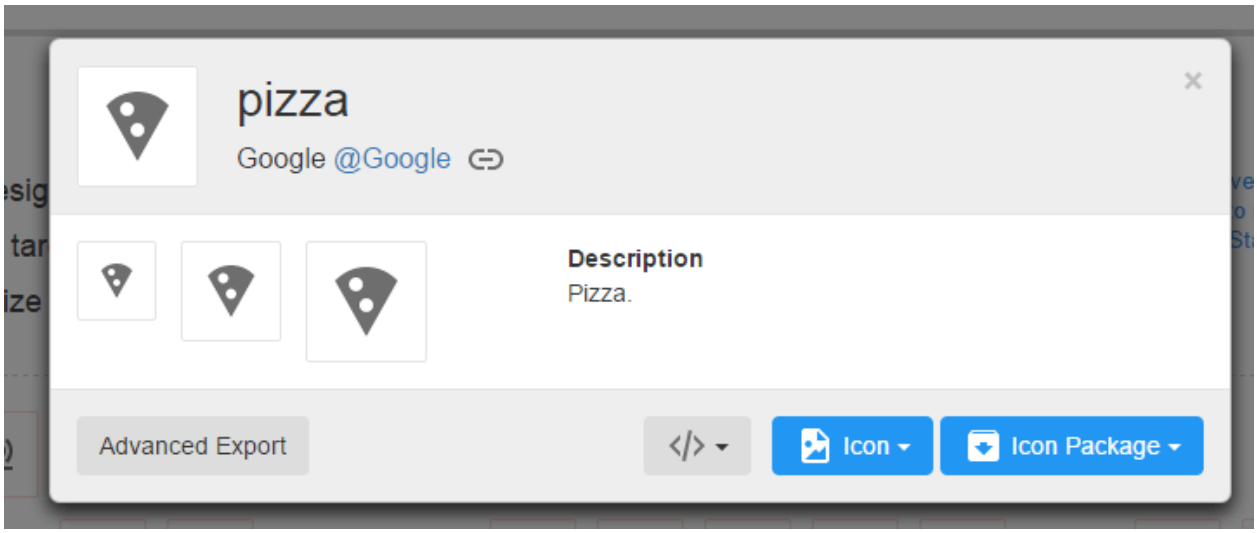
En la sección de atributos viste que existen atributos con la forma `android:drawable*` para alinear una imagen al texto de un botón.

Para demostrar su uso veamos el siguiente ejemplo sencillo.

Alinear a la izquierda del botón ordenar un icono asociado a pizzas.

Solución

Abre el layout `actividad_botones.xml` y agrega un botón centrado en el padre. Cambia el color de texto a blanco (`@android:color/white`), usa un tinte de color rojo (`#ef9a9a`) y alinea a la izquierda el siguiente drawable de pizza:

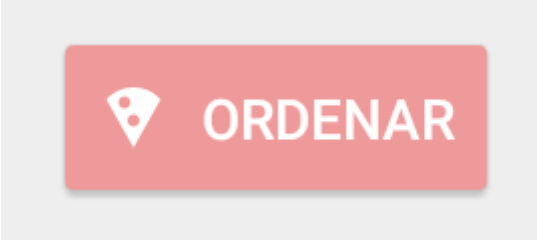


Tu código debería quedar de la siguiente forma:

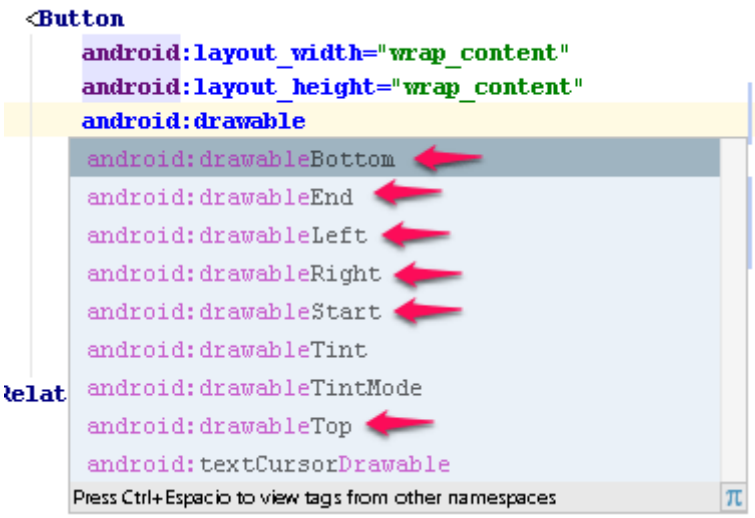
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    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.herprogramacion.botones.ActividadBotones">
```

```
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:drawableLeft="@drawable/ic_pizza"
        android:drawablePadding="8dp"
        android:id="@+id/boton"
        android:textColor="@android:color/white"
        app:backgroundTint="#ef9a9a"
        android:text="Ordenar"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Y el resultado sería:



Como ves, usé `android:drawableLeft` para ubicar la imagen en la izquierda. Si compruebas con las sugerencias de Android tendrás varias posiciones: derecha, abajo y arriba.



La Clase ImageButton

`ImageButton` funciona exactamente cómo `Button`, solo que en lugar de traer un texto en su background, viene una imagen para especificar la acción.

Para cambiar la imagen de un image button usa el atributo `android:src`. Obviamente su valor es un drawable.

El contorno del background se conserva como lo hemos visto hasta ahora, la diferencia está que en el centro se
ubicará la imagen elegida en `src`.

Si quieres que el background por defecto desaparezca, asigna un color transparente o cambia el contenido con un list drawable (esto lo verás más adelante).

Ejemplo:

Cambiar la imagen de un image button con el icono de la app.

Solución

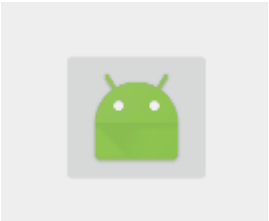
Lo primero es abrir el layout de la actividad y añadir un elemento `<ImageButton>` centrado en el relative layout.

El icono de la aplicación actual se encuentra en la referencia `@mipmap/ic_launcher`. Así que asigna este valor al atributo `andriod:src`.

Adicionalmente puedes el color del sistema `@android:color/transparent` sobre `android:background` para eliminar el contorno. Pero recuerda que esto elimina los efectos del Material Design.

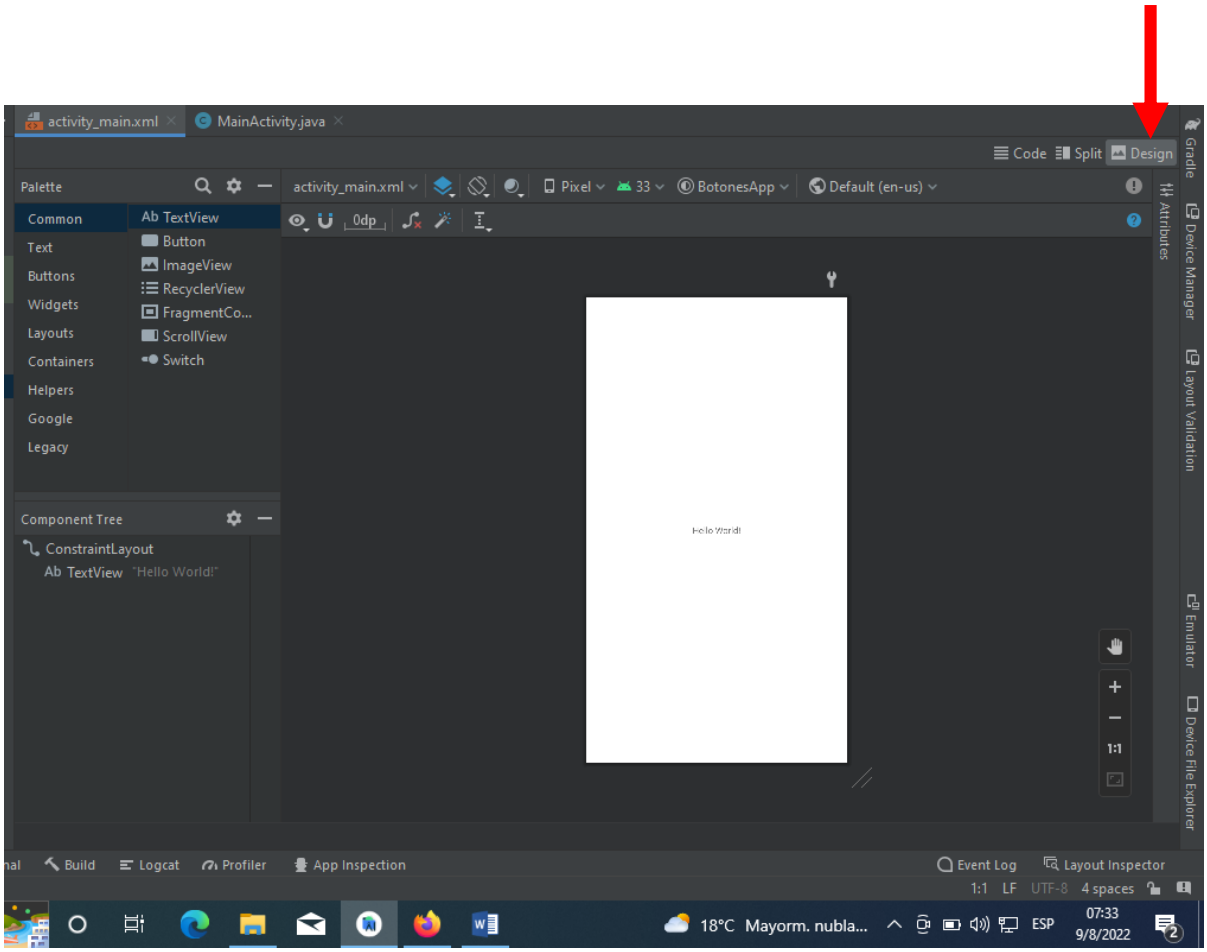
```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/boton"
    android:src="@mipmap/ic_launcher"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />
```

Ejecuta la app y verás la apariencia del botón final:

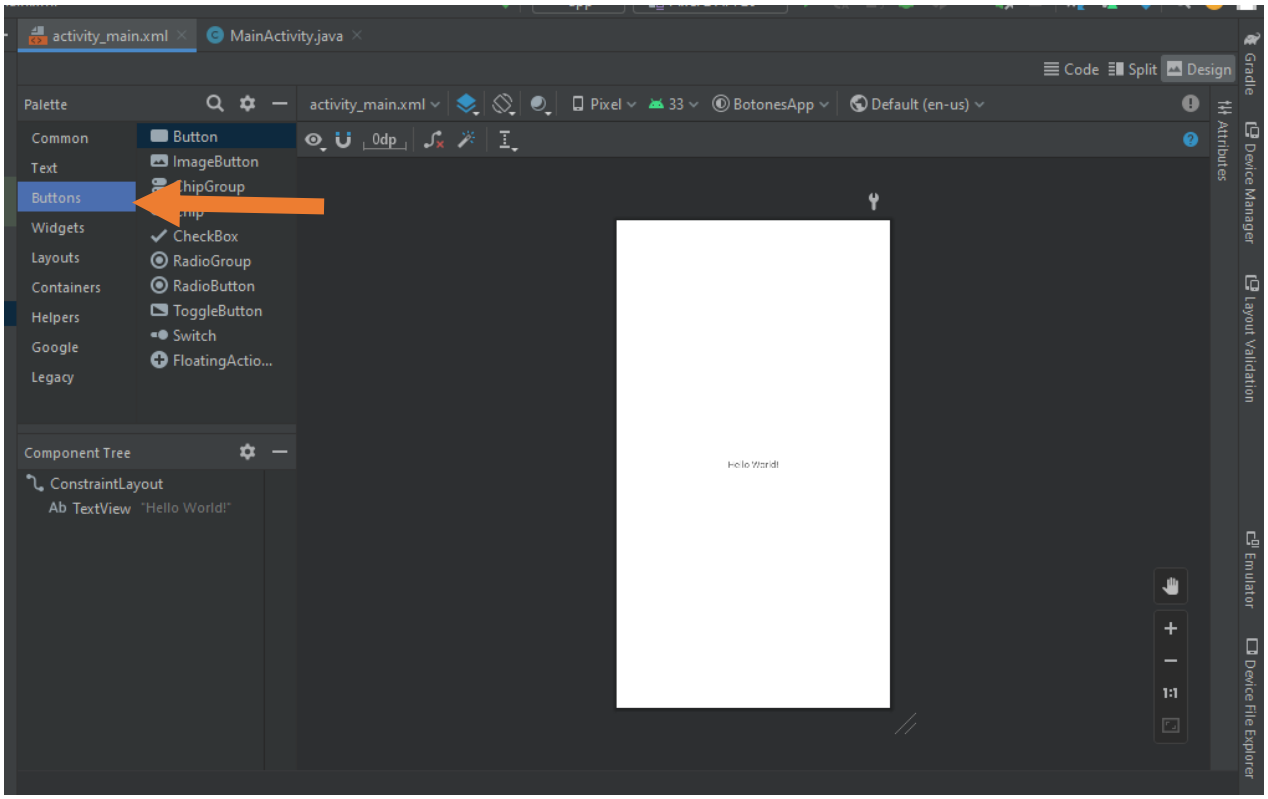


Crear Botón Desde Android Studio

Usa la pestaña **Design** que se ve en la vista de un layout para arrastrar y soltar un botón desde la paleta o cambiar sus atributos.

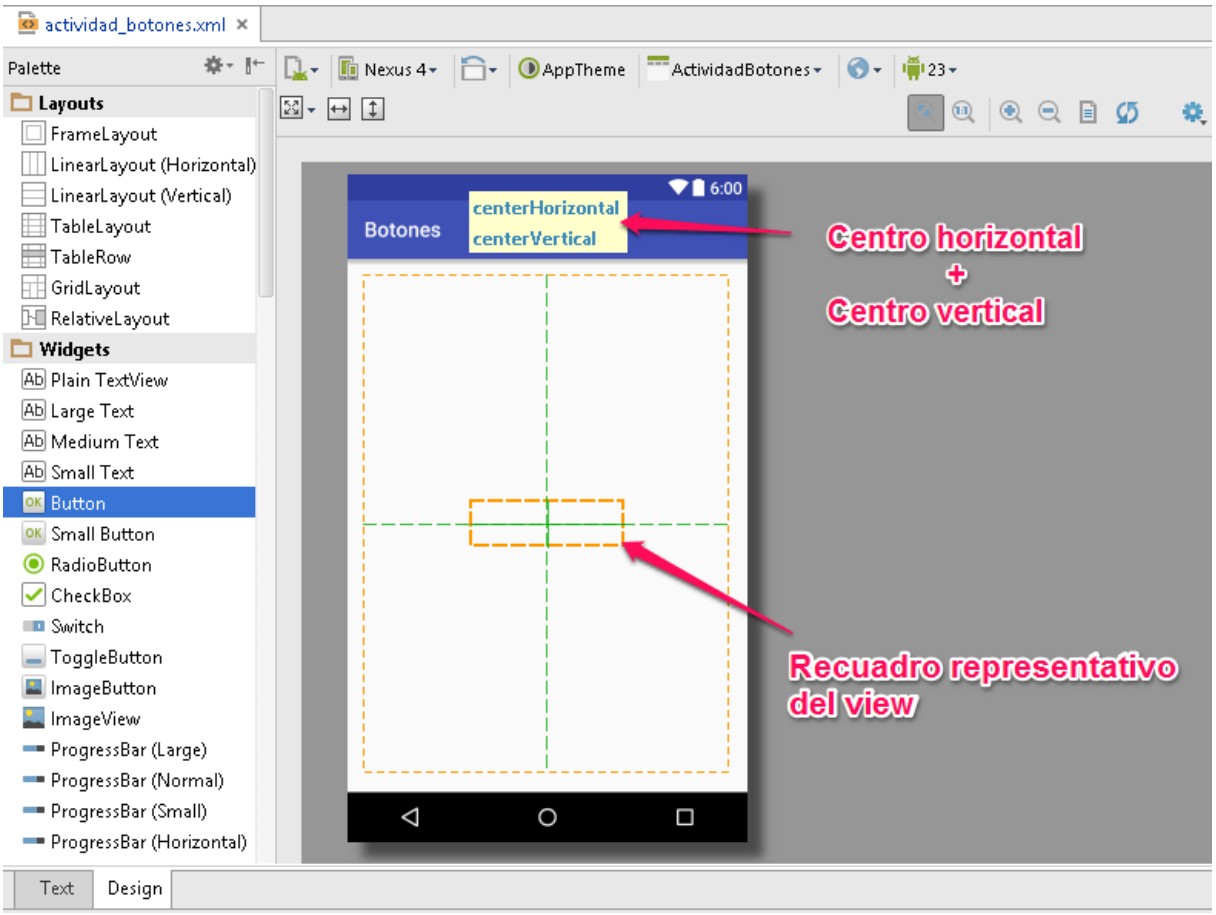


1. Sitúate en la ventana **Palette**. Luego busca el elemento **Button** de la sección **Buttons**

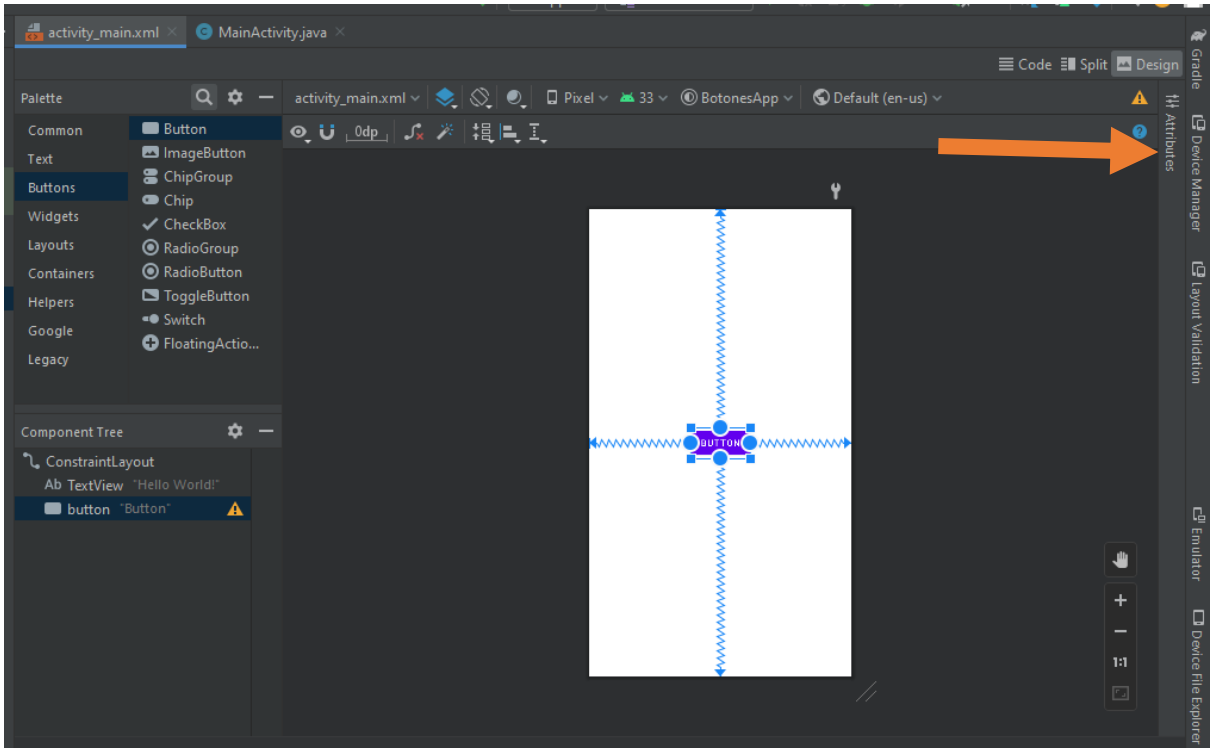


2. Lleva el puntero del mouse hacia el layout y ubícalo según tu objetivo. En mi caso tengo un RelativeLayout, el cual permite orientar mi botón en lugares asociados al contenido del padre y al de otros views.

Por el momento lo ubicaré en el centro absoluto del layout, donde el editor mostrará líneas de guía para saber que valores estoy otorgando:



3. Una vez elegido el lugar donde se ubicará el botón, confirma con un click su inserción. Esto mostrará en la ventana **Atributes** todos los atributos del botón que puedes editar.



Tipos De Botones En Material Design

La documentación sobre botones en el Material Design nos habla de tres tipos:

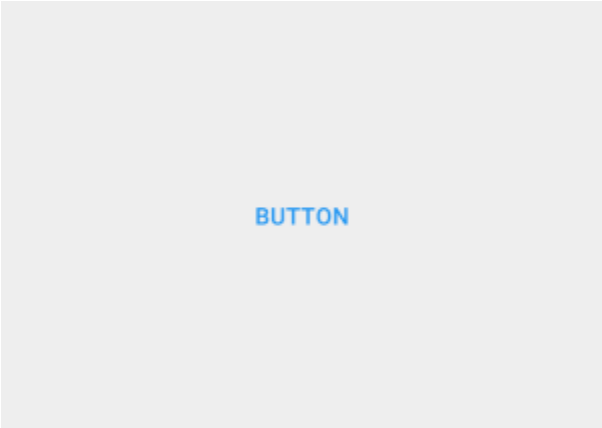
1. Elevated button
2. Filled button
3. Filled tonal button
4. Outlined button
5. Text button
6. Icon button
7. Segmented button
8. Floating action button (FAB)
9. Extended FAB

Los Elevated buttons son los que hemos visto hasta ahora. Tienen backgrounds rectangulares, se elevan ante el click del usuario y su superficie reacciona con un efecto ripple.

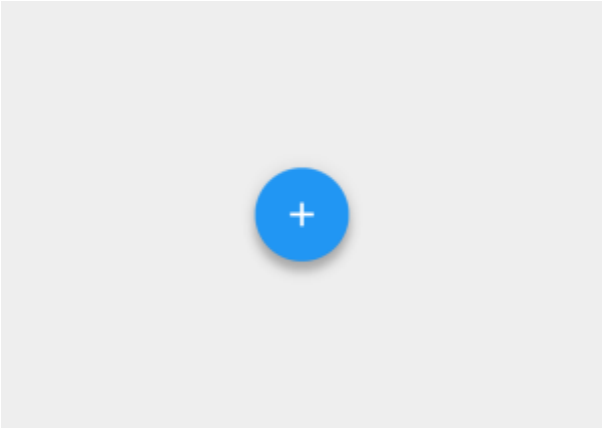
Se usan sobre layouts que tienen gran variedad de contenido para delimitar las secciones o elementos. Además realzan la importancia de realizar una acción relacionada.



Por el contrario, un flat button o botón sin bordes tiene reacción de superficie pero no se eleva al ser presionado. Se usan para acciones en diálogos, toolbars y de forma persistente en steppers.



El Floating action button tiene las mismas interacciones de elevación y ripple que el raised button, solo que este tiene forma circular. Su objetivo es promover una acción principal relacionada a una pieza de material.



Manejar Los Eventos De Un Botón

Es de esperar que un botón dispara un evento al ser clickeado por un usuario, lo que permitirá ejecutar la acción a la que hace referencia dicho view.

Para procesar el evento existen varias formas de proceder.

Usar el Atributo android:onClick

Anteriormente cuando viste la clase `Button` se mencionó la existencia de `android:onClick()` para asignar un método que se ejecute cuando el usuario presione el botón.

Para ello se requiere que el método cumpla con las siguientes condiciones:

1. Que sea público
2. Que sea tipo `void`
3. Que reciba un parámetro del tipo `View`
4. Debe declararse en la actividad que usa el mismo layout

Ejemplo:

Cambiar el texto de un Text View por el mensaje «Botón presionado: [hora_actual]» al presionar un Button.

Solución:

Comienza por añadir un text view al layout. En mi caso lo ubicaré en el centro del relative layout y por debajo pondré el botón:

```
<?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.herprogramacion.botones.ActividadBotones">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/texto_agregar"
        android:layout_below="@+id/textView"
        android:layout_centerHorizontal="true" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="A la espera..."
        android:id="@+id/textView"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Ahora en la actividad `ActividadBotones` crearé un método para los clicks llamado `cambiarMensaje()`.

La idea es obtener el text view como campo para poder cambiar su texto con `setText()` desde el método.

Para obtener la hora usa la clase `Calendar` y para formatear su estructura usa `SimpleDateFormat`.

ActividadBotones.java

```
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.TextView;

import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

public class ActividadBotones extends AppCompatActivity {

    private TextView texto;

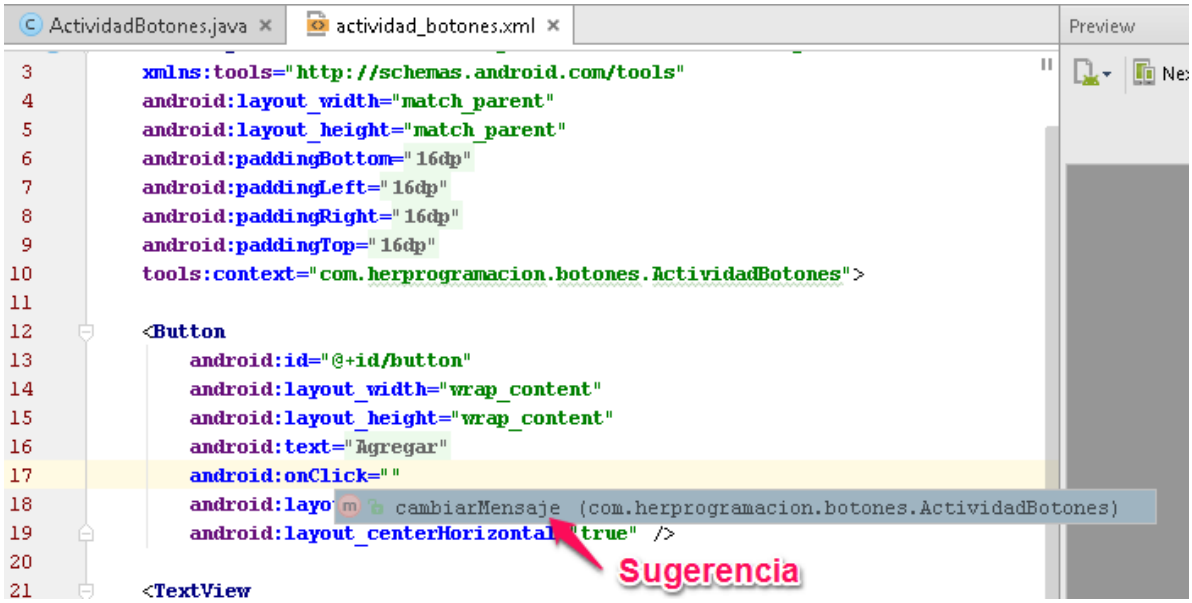
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.actividad_botones);

        texto = (TextView) findViewById(R.id.textView);
    }

    public void cambiarMensaje(View v){
        SimpleDateFormat formato = new SimpleDateFormat("HH:mm:ss");
        Date fechaActual = Calendar.getInstance().getTime();

        String s = formato.format(fechaActual);
        texto.setText(String.format("Botón presionado: %s", s));
    }
}
```

Lo siguiente es dirigirse al layout y asignar el método en el botón con `android:onClick`. Al momento de hacerlo, el motor de Android Studio debe sugerirte que `cambiarMensaje()` está disponible.



El atributo quedaría de la siguiente forma:

```
<Button
    ...
    android:onClick="cambiarMensaje"/>
```

Al ejecutar la app y clickear el botón tendrás el cambio de texto por la hora actual.



Usar Escucha Anónima OnClickListener

Otra forma es crear una instancia anónima de la interfaz `View.OnClickListener` para manejar los eventos del botón.

Esto requiere usar el método `setOnClickListener()` para asignar el listener al botón y luego sobrescribir el controlador `onClick()` con las acciones a ejecutar.

Ejemplo:

Iniciar otra actividad al presionar un botón

Solución

Lo primero es añadir otra actividad al proyecto actual que tienes abierto (recuerda quitar el text view que se tenía antes y la referencia de `android:onClick()`).

Ve al menú de herramientas y presiona **File > New > Activity > Empty Activity**

Nombra a la nueva actividad como **«ActividadNueva»** y confirma.

Lo siguiente es ir `ActividadBotones` y obtener la instancia del botón que tenemos en `onCreate()`.

Luego invoca el método `setOnClickListener()` desde la instancia y como parámetro digita solamente `"new O"`. Esto es suficiente para que Android Studio te recomiende la creación de un nuevo `OnClickListener`.



Al presionar `ENTER` o clickear la sugerencia, Android Studio creará la escucha anónima junto a la implementación del controlador `onClick()`:

```
Button boton = (Button) findViewById(R.id.button);
boton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

    }
});
```

Ahora dentro del controlador crea un nuevo Intent para iniciar la actividad `ActividaNueva`. Seguido invoca `startActivity()` para hacer efectivo el inicio.

```
Button boton = (Button) findViewById(R.id.button);
boton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent i = new Intent(ActividadBotones.this,
ActividadNueva.class);
        startActivity(i);
    }
});
```

Usar OnClickListener Sobre el Contenedor

`OnClickListener` también puede ser implementada sobre el contenedor del botón como lo es una actividad o un fragmento.

Así podrás sobrescribir el controlador `onClick()` para que tenga un alcance global para todos los views que pertenezcan al contenedor. Lo que es de utilidad si se deseas leer clicks sobre varios elementos al tiempo.

Ejemplo:

Cambiar el color del background de una actividad dependiendo de qué botón de 4 disponibles se presione.

Solución:

Pon 4 botones centrados horizontalmente y uno debajo de otro dentro del relative layout. Usa los siguientes colores en sus textos: *Verde azulado*, *Verde*, *Verde claro* y *Lima*.

actividad_botones.xml

```
<?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.herprogramacion.botones.ActividadBotones">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Verde azulado"
        android:layout_centerHorizontal="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Verde"
        android:id="@+id/button2"
        android:layout_below="@+id/button"
        android:layout_centerHorizontal="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Verde claro"
        android:id="@+id/button3"
        android:layout_below="@+id/button2"
        android:layout_centerHorizontal="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Lima"
        android:id="@+id/button4"
```

```
        android:layout_below="@+id/button3"
        android:layout_centerHorizontal="true" />


</RelativeLayout>
```

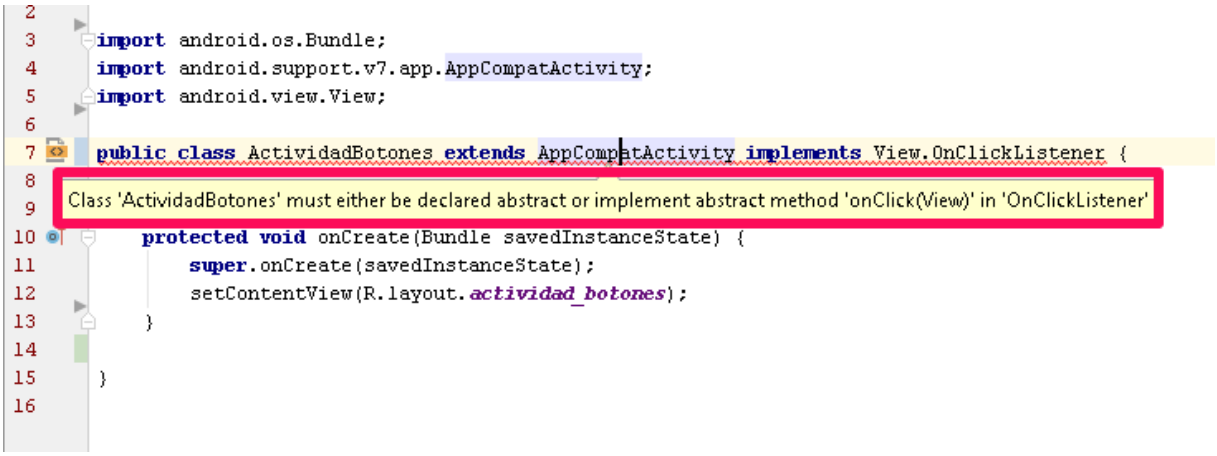
Implementa la escucha `OnClickListener` sobre la `ActividadBotones`:

```
public class ActividadBotones extends AppCompatActivity implements
View.OnClickListener {

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

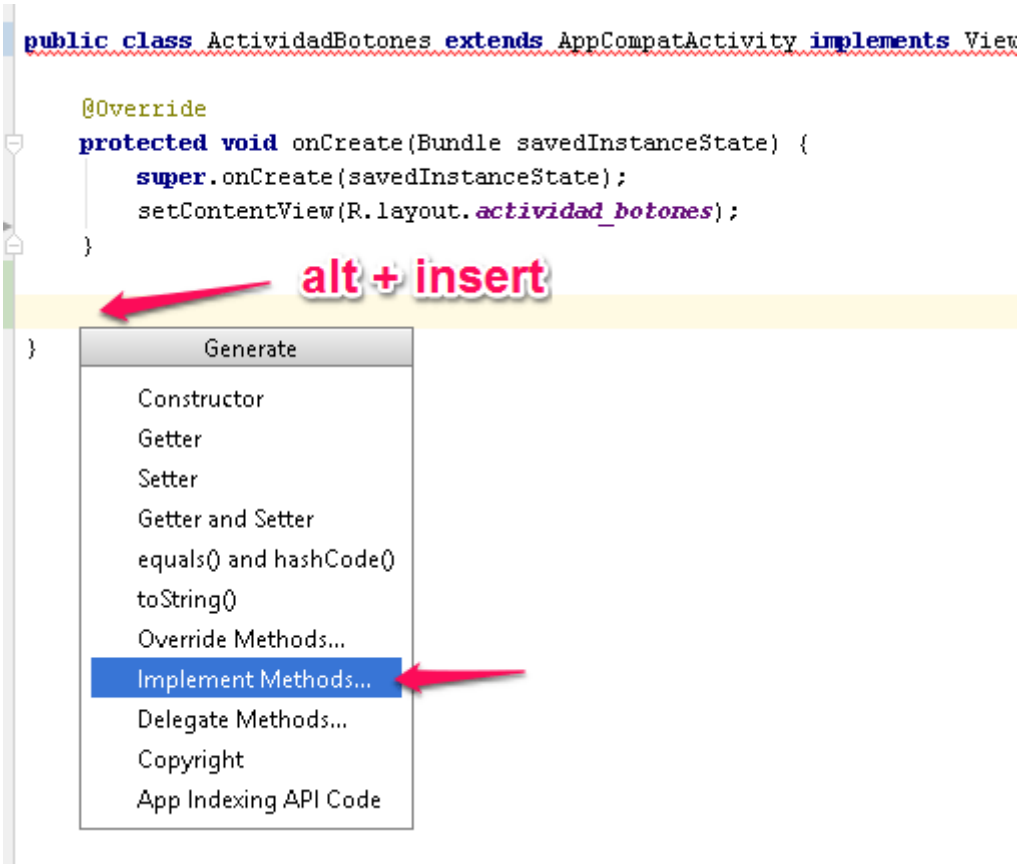
}
```

Si dejas el código como está, Android Studio te informará que hay un error porque no has sobreescrito el controlador.

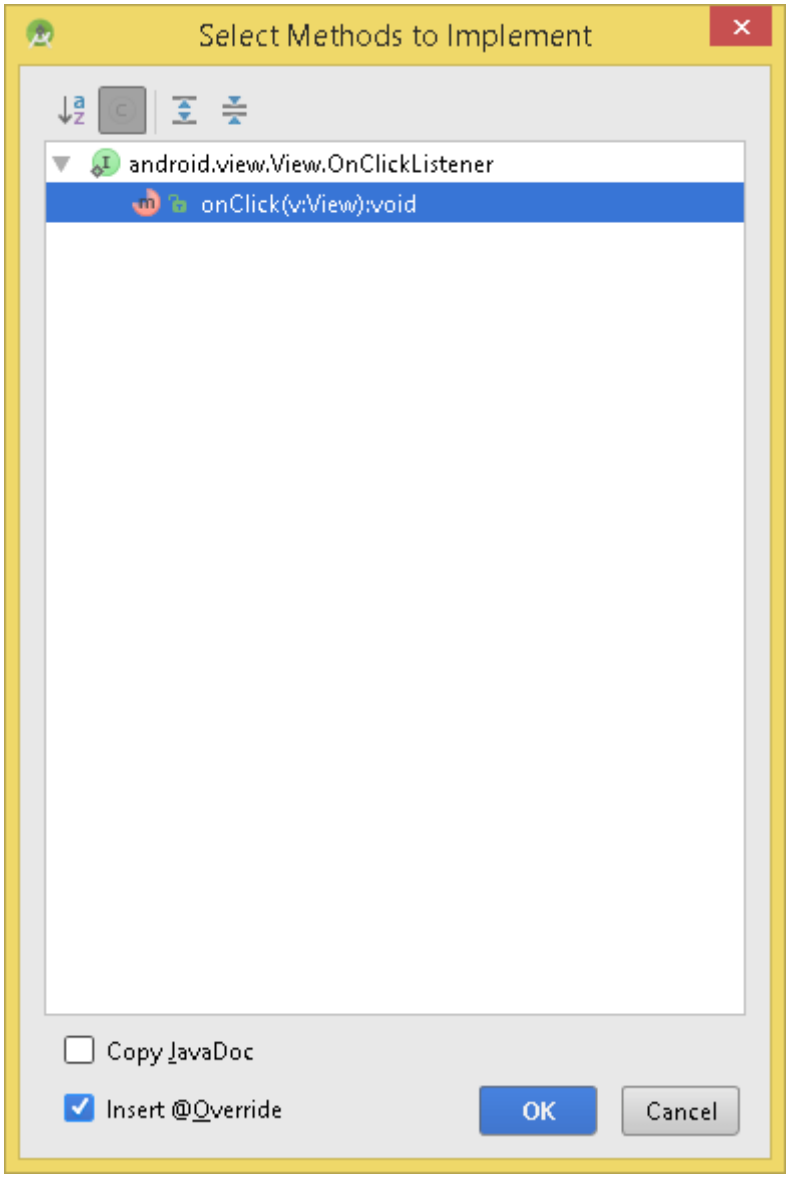


La solución manual sería escribir la firma y cuerpo del método como es requerido. Sin embargo puedes usar la generación de código automático de Android Studio.

Presiona **ALT + insert** dentro del cuerpo de la clase y selecciona
Implement Methods…



Luego selecciona `onClick()` del menú y confirma.



Con ello construirás el cuerpo de forma automática:

```
@Override
public void onClick(View v) {

}
```

Lo que sigue es detectar los eventos de click con múltiples botones.

Una de las formas más sencilla es usar una sentencia `switch`, donde su expresión de validez sea el identificador del view que viene como parámetro de `onClick()`.

El algoritmo sería el siguiente:

- Asignar la escucha a todos los botones con el operador `this`.
- Obtener el view raíz de la jerarquía con `getRootView()`.
- Abrir un `switch` con el id del view clickeado. El id se obtiene con el método `getId()`.

- Elegir el color dependiendo del caso. Convierte una definición hexadecimal en entero con `Color.parseColor()`.
- Cambiar el background del contenedor con el método `setBackgroundColor()`

Veamos:

```
import android.graphics.Color;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;

public class ActividadBotones extends AppCompatActivity implements
View.OnClickListener {

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

        findViewById(R.id.button).setOnClickListener(this);
        findViewById(R.id.button2).setOnClickListener(this);
        findViewById(R.id.button3).setOnClickListener(this);
        findViewById(R.id.button4).setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {

        int color;

        View contenedor = v.getRootView();

        switch (v.getId()) {
            case R.id.button:
                color = Color.parseColor("#80CBC4"); // Verde azulado
                break;
            case R.id.button2:
                color = Color.parseColor("#A5D6A7"); // Verde
                break;
            case R.id.button3:
                color = Color.parseColor("#C5E1A5"); // Verde claro
                break;
            case R.id.button4:
                color = Color.parseColor("#E6EE9C"); // Lima
                break;
            default:
                color = Color.WHITE; // Blano
        }

        contenedor.setBackgroundColor(color);
    }
}
```

Ejecuta la app y prueba clickear los 4 botones para cambiar el background de la actividad.



Personalizar El Estilo De Un Botón

Ya viste algunos atributos que puedes cambiar en los botones para transformar su apariencia. Con eso puedes ir a tu archivo **styles.xml** y crear estilos para estos views.

A continuación te mostraré algunos casos que pueden presentarse a la hora de querer personalizar un botón.

Cambiar Background

Si quieres omitir la forma por defecto del botón y los efectos de reacción, puedes probar usando un recurso drawable del tipo State List.

Este elemento representa múltiples drawables que serán usados en un mismo view simulando cambio de estados.

Esto se ajusta perfectamente a los botones ya que su apariencia varía según la interacción.

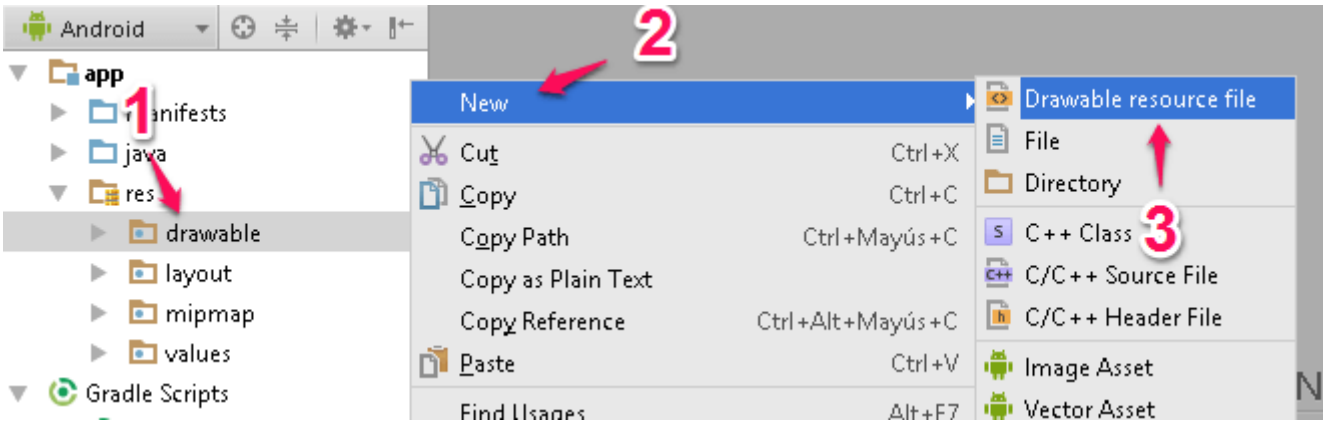
Ejemplo:

Crear background de un botón con diseño plano (Flat Design)

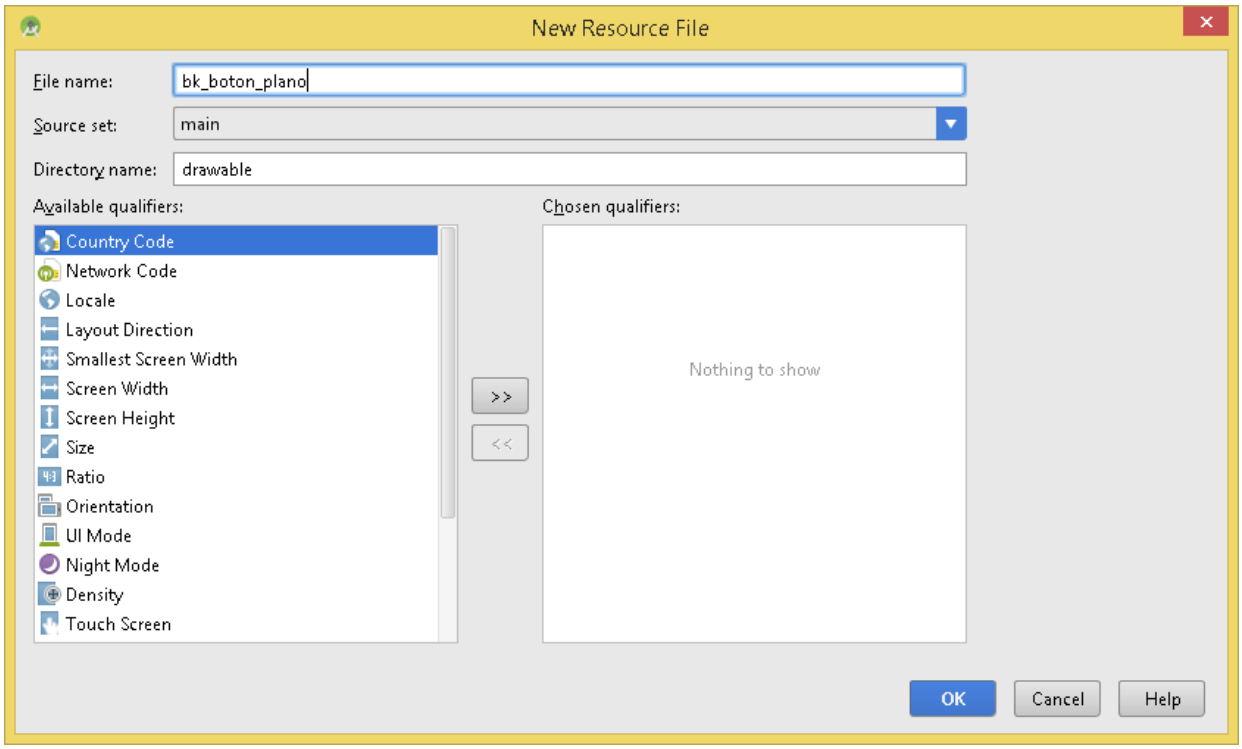
Solución:

Lo primero que harás es crear un nuevo drawable llamado `bk_boton_plano.xml`.

Para ello ve a la carpeta **drawable**, presiona **click derecho** y selecciona **Drawable resource file**.



Confirma el nombre con **OK**:



La definición XML requiere que el nodo principal sea del tipo `<selector>`.

Dentro de este deben existir elementos `<item>` que definan el drawable que asignará a cada estado del botón.

Existen cuatro estados primordiales:

- *normal*
- *enfocado* (`state_focused`)
- *presionado* (`state_pressed`)
- *deshabilitado* (`state_enabled > false`).

Con lo anterior el código quedaría así:

bk_boton_plano.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/boton_presionado"
        android:state_pressed="true" />
    <item android:drawable="@drawable/boton_enfocado"
        android:state_focused="true" />
    <item android:drawable="@drawable/boton_deshabilitado"
        android:state_enabled="false" />
    <item android:drawable="@drawable/boton_normal" />
</selector>
```

Cada drawable de estado se compone de una lista de capas de drawables para simular un borde sombreado inferior.

El estado normal tiene asignado el drawable `boton_normal.xml` el cual tiene dos en tono naranja profundo.

boton_normal.xml

```
<?xml version="1.0" encoding="utf-8"?>

<layer-list
xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Sombra -->
    <item android:top="4dp">
        <shape>
            <solid android:color="#FF7043" />
            <corners android:radius="4dip" />
        </shape>
    </item>
    <!-- Contenido -->
    <item android:bottom="4dp">
        <shape>
            <solid android:color="#FF8A65" />
            <corners android:radius="4dip" />
        </shape>
    </item>
</layer-list>
```

El estado de foco simplemente cambia los colores de ambas capas para visualizar un cambio minimo de contraste.

boton_enfocado.xml

```
<?xml version="1.0" encoding="utf-8"?>

<layer-list
xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Sombra -->
    <item android:top="4dp">
        <shape>
            <solid android:color="#FFCA28" />
            <corners android:radius="4dip" />
        </shape>
    </item>
    <!-- Contenido -->
    <item android:bottom="4dp">
        <shape>
            <solid android:color="#FFD54F" />
            <corners android:radius="4dip" />
        </shape>
    </item>
</layer-list>
```

Y el botón presionado es exactamente lo mismo. Un aumento de opacidad en el tono.

boton_presionado.xml

```
<?xml version="1.0" encoding="utf-8"?>

<layer-list
xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Sombra -->
    <item android:top="4dp">
        <shape>
            <solid android:color="#FFA726" />
            <corners android:radius="4dip" />
        </shape>
    </item>
    <!-- Contenido -->
    <item android:bottom="4dp">
        <shape>
            <solid android:color="#FFB74D" />
            <corners android:radius="4dip" />
        </shape>
    </item>
</layer-list>
```

Con esos drawables listos, ahora puedes ir al layout de la actividad y modificar el atributo `android:background` para asignar la lista. Cambia el color de texto con `android:textColor` a blanco para mejorar el contraste y si deseas desaparecer la animación de elevación, usa `android:stateListAnimator` con el valor de `@null`.

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:textColor="@android:color/white"
    android:background="@drawable/bk_boton_plano"
    android:stateListAnimator="@null"
    android:text="@string/texto_agregar" />
```

Este serían los estados logrados:



Estilo BorderLess o Botón sin Bordes

Ya habías visto que el botón sin bordes se le denomina Flat button en el Material Design.

Para crear dicho acabado puedes aplicar el estilo `Widget.AppCompat.Button.Borderless`.

Por ejemplo…

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Llamar"
    style="@style/Widget.AppCompat.Button.Borderless"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />
```

Produciría el siguiente resultado:



Si deseas que el color del texto sea el mismo del acento, entonces usa `Widget.AppCompat.Button.Borderless.Colored`.



Conclusión

Los botones son unos de los widgets más básicos en Android.

Tienen la función de ayudar al usuario a decidir por sus acciones sobre el contenido y datos de la app.

Saber describir su comportamiento con `OnClickListener` es vital para las interacciones. Recuerda que las acciones se ubican en el controlador `onClick()`.

O puedes usar el editor de diseño en Android Studio para asignar el método en el atributo `android:onClick`.

Usa el estilo correcto según lo hablado en la sección de Material Design. Es decir: *Raised buttons* para acciones estables, *Flat buttons* para confirmaciones o acciones repetitivas y el *Floating action button* para promover acciones principales sobre datos.

WEBGRAFIA

https://www.develou.com/

Intents y filtros de intents | Desarrolladores de Android | Android Developers

Una `Intent` es un objeto de mensajería que puedes usar para solicitar una acción de otro componente de una app. Si bien las intents facilitan la comunicación entre componentes de varias formas, existen tres casos de uso principales:

- **Iniciar una actividad**

  Una `Activity` representa una única pantalla en una aplicación. Puedes iniciar una nueva instancia de una `Activity` pasando una `Intent` a `startActivity()`. La `Intent` describe la actividad que se debe iniciar y contiene los datos necesarios para ello.

  Si deseas recibir un resultado de la actividad cuando finalice, llama a `startActivityForResult()`. La actividad recibe el resultado como un objeto `Intent` separado en la devolución de llamada de `onActivityResult()` de la actividad. Para obtener más información, consulta la guía Actividades.

- **Iniciar un servicio**

  Un `Service` es un componente que realiza operaciones en segundo plano sin una interfaz de usuario. Con Android 5.0 (nivel de API 21) y versiones posteriores, puedes iniciar un servicio con `JobScheduler`. Para obtener más información acerca de `JobScheduler`, consulta su `API-reference documentation`.

  En las versiones anteriores a Android 5.0 (nivel de API 21), puedes iniciar un servicio usando métodos de la clase `Service`. Puedes iniciar un servicio para realizar una operación única (como descargar un archivo) pasando una `Intent` a `startService()`. La `Intent` describe el servicio que se debe iniciar y contiene los datos necesarios para ello.

  Si el servicio está diseñado con una interfaz cliente-servidor, puedes establecer un enlace con el servicio de otro componente pasando una `Intent` a `bindService()`. Para obtener más información, consulta la guía Servicios.

- **Transmitir una emisión**

  Una emisión es un aviso que cualquier aplicación puede recibir. El sistema transmite varias emisiones de eventos, como cuando se inicia el sistema o comienza a cargarse el dispositivo. Puedes transmitir una emisión a otras apps pasando una `Intent` a `sendBroadcast()` o `sendOrderedBroadcast()`.

El resto de esta página explica cómo funcionan las intents y cómo debes usarlas. Para obtener información relacionada, consulta Cómo actualizar con otras apps y Cómo compartir contenido.

Tipos de intents

Existen dos tipos de intents:

- Las **intents explícitas** especifican qué aplicación las administrará, ya sea incluyendo el nombre del paquete de la app de destino o el nombre de clase del componente completamente calificado. Normalmente, el usuario usa una intent explícita para iniciar un componente en su propia aplicación porque conoce el nombre de clase de la actividad o el servicio que desea iniciar. Por ejemplo, puedes utilizarla para iniciar una actividad nueva en respuesta a una acción del usuario o iniciar un servicio para descargar un archivo en segundo plano.
- Las **intents implícitas** no nombran el componente específico, pero, en cambio, declaran una acción general para realizar, lo cual permite que un componente de otra

aplicación la maneje. Por ejemplo, si deseas mostrar al usuario una ubicación en un mapa, puedes usar una intent implícita para solicitar que otra aplicación apta muestre una ubicación específica en un mapa.

La figura 1 muestra cómo se usa una intent para iniciar una actividad. Cuando el objeto `Intent` nombra un componente de actividad específico de forma explícita, el sistema inicia ese componente al instante.

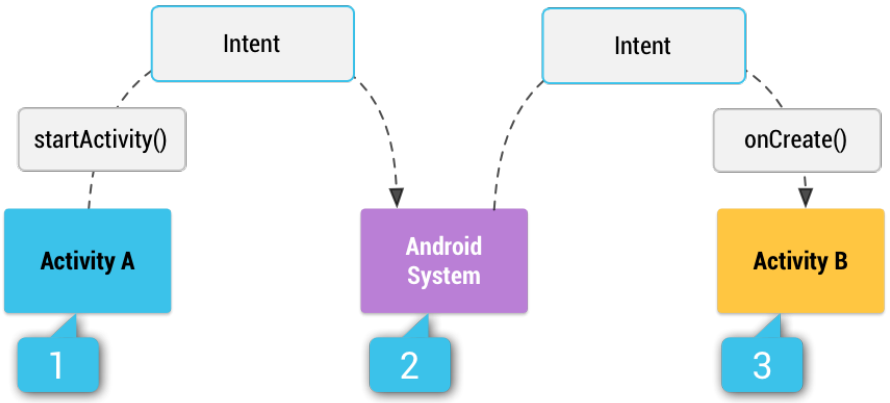


Figura 1: Ilustración de la forma en que se entrega una intent implícita mediante el sistema para iniciar otra actividad. **[1]** La *actividad A* crea una `Intent` con una descripción de acción y la pasa a `startActivity()`. **[2]** El sistema Android busca en todas las apps un filtro de intents que coincida con ella. Cuando se encuentra una coincidencia, **[3]** el sistema inicia la actividad coincidente (*actividad B*) invocando su método `onCreate()` y pasándolo a la `Intent`.

Cuando usas una intent implícita, el sistema Android busca el componente apropiado para iniciar comparando el contenido de la intent con los *filtros de intents* declarados en el archivo de manifiesto de otras aplicaciones en el dispositivo. Si la intent coincide con un filtro de intents, el sistema inicia ese componente y le entrega el objeto `Intent`. Si varios filtros de intents son compatibles, el sistema muestra un cuadro de diálogo para que el usuario pueda elegir la aplicación que se debe usar.

Un filtro de intents es una expresión en el archivo de manifiesto de una aplicación que especifica el tipo de intent que el componente podría recibir. Por ejemplo, declarar un filtro de intents para una actividad permite que otras aplicaciones la inicien directamente con un tipo de intent específico. Asimismo, si *no* declaras ningún filtro de intent para una actividad, esta solo se puede iniciar con una intent explícita.

Advertencia: Para garantizar que tu aplicación sea segura, siempre usa una intent explícita cuando inicies un `Service` y no declares filtros de intents para los servicios. El uso de una intent implícita para iniciar un servicio es un riesgo de seguridad porque no puedes garantizar qué servicio responderá a la intent, y el usuario no puede ver qué servicio se inicia. A partir de Android 5.0 (nivel de API 21), el sistema arroja una excepción si llamas a `bindService()` con una intent implícita.

Cómo crear una intent

Un objeto `Intent` tiene información que el sistema Android usa para determinar qué componente debe iniciar (como el nombre exacto del componente o la categoría que debe recibir la intent), además de información que el componente receptor usa para realizar la acción correctamente (por ejemplo, la acción que debe efectuar y los datos en los que debe actuar).

La información principal que contiene una `Intent` es la siguiente:

Nombre del componente
El nombre del componente que se debe iniciar.

Esto es opcional, pero es la información clave que hace que una intent sea *explícita*, lo que significa que la intent debe enviarse solamente al componente de la aplicación definido en el nombre del componente. Sin un nombre de componente, la intent es

implícita y el sistema decide qué componente debe recibir la intent conforme la información restante que esta contiene (como la acción, los datos y la categoría, que se describen a continuación). Por lo tanto, si necesitas iniciar un componente específico en tu aplicación, debes especificar el nombre del componente.

Nota: Cuando inicies un `Service`, *especifica siempre el nombre del componente*. De lo contrario, no podrás asegurarte de qué servicio responderá a la intent, y el usuario no podrá ver qué servicio se inicia.

Este campo de la `Intent` es un objeto `ComponentName` que puedes especificar con un nombre de clase completamente calificado del componente de destino, incluido el nombre del paquete de la aplicación, como `com.example.ExampleActivity`. Puedes establecer el nombre del componente con `setComponent()`, `setClass()`, `setClassName()` o el constructor `Intent`.

Acción

Una string que especifica la acción genérica que se debe realizar (como *ver* o *elegir*).

En el caso de la intent de una emisión, es la acción que se produjo y que se está registrando. La acción determina cuál es la estructura del resto de la intent, especialmente la información que se incluye en los datos y extras.

Puedes especificar tus propias acciones para que las usen las intents en tu aplicación (o para que las usen otras aplicaciones a fin de invocar componentes en tu aplicación); pero, usualmente, debes especificar acciones constantes definidas por la clase `Intent` u otras clases de marcos de trabajo. Estas son algunas acciones comunes para iniciar una actividad:

`ACTION_VIEW`
: Usa esta acción en una intent con `startActivity()` cuando tengas información que la actividad pueda mostrar al usuario, como una foto para ver en una app de galería o una dirección para ver en una app de mapas.

`ACTION_SEND`
: También se conoce como la intent de *compartir* y debes usarla en una intent con `startActivity()` cuando tengas información que el usuario pueda compartir mediante otra app, como una app de correo electrónico o intercambio social.

Consulta la referencia de la clase `Intent` para conocer más constantes que definen las acciones genéricas. Otras acciones se definen en otras partes del marco de trabajo de Android, como en `Settings` para las acciones que abren pantallas específicas en la aplicación de Configuración del sistema.

Puedes especificar la acción de una intent con `setAction()` o un constructor `Intent`.

Si defines tus propias acciones, asegúrate de incluir el nombre del paquete de la aplicación como prefijo, tal como se muestra en el siguiente ejemplo.

const val ACTION_TIMETRAVEL = "com.example.action.TIMETRAVEL"

static final String ACTION_TIMETRAVEL = "com.example.action.TIMETRAVEL";

Datos

El URI (un objeto `Uri`) que hace referencia a los datos en los que se debe realizar la acción o el tipo de MIME de esos datos. El tipo de datos suministrado está generalmente determinado por la acción de la intent. Por ejemplo, si la acción es `ACTION_EDIT`, los datos deben contener el URI del documento que se debe editar.

Cuando creas una intent, es importante especificar el tipo de datos (su tipo de MIME) además de su URI. Por ejemplo, una actividad que puede mostrar imágenes probablemente no sea capaz de reproducir un archivo de audio aunque los formatos de URI sean similares. Especificar el tipo de MIME de tus datos ayuda al sistema Android a encontrar el mejor componente para recibir tu intent. Sin embargo, el tipo de MIME a veces se puede deducir del URI, especialmente cuando los datos son un URI `content:`. Un URI `content:` indica que los datos se encuentran en el dispositivo y son controlados por

un `ContentProvider`, lo que hace que el sistema pueda ver el tipo de datos de MIME.

Para establecer solamente el URI de datos, llama a `setData()`. Para establecer solo el tipo de MIME, llama a `setType()`. Si es necesario, puedes establecer ambos explícitamente con `setDataAndType()`.

Advertencia: Si deseas establecer tanto el URI como el tipo de MIME, *no* llames a `setData()` ni a `setType()`, ya que cada uno anula el valor del otro. Siempre usa `setDataAndType()` para establecer el URI y el tipo de MIME.

Categoría

Una string que contiene información adicional sobre el tipo de componente que la intent debe manejar. En una intent, se puede incluir la cantidad deseada de descripciones de categorías, pero la mayoría de las intents no requieren una categoría. Estas son algunas categorías comunes:

`CATEGORY_BROWSABLE`
: La actividad de destino permite que la inicie un navegador web para mostrar datos a los que hace referencia un vínculo, como una imagen o un mensaje de correo electrónico.

`CATEGORY_LAUNCHER`
: La actividad es la actividad inicial de una tarea y está enumerada en el selector de la aplicación del sistema.

Para ver la lista completa de categorías, consulta la descripción de la clase `Intent`.

También puedes especificar una categoría con `addCategory()`.

Las propiedades nombradas anteriormente (nombre de componente, acción, datos y categoría) representan las características definitorias de una intent. Mediante la lectura de estas propiedades, el sistema Android puede resolver qué componente de la aplicación debe iniciar. Sin embargo, una intent puede tener información adicional que no afecte cómo se resuelve en un componente de la aplicación. Una intent también puede incluir la siguiente información:

Extras

Pares clave-valor que contienen información adicional necesaria para completar la acción solicitada. Al igual que algunas acciones usan tipos particulares de URI de datos, algunas acciones también usan extras específicos.

Puedes agregar datos adicionales con varios métodos `putExtra()`, cada uno de los cuales acepta dos parámetros: el nombre de la clave y el valor. También puedes crear un objeto `Bundle` con los datos adicionales y, luego, insertar el `Bundle` en la `Intent` con `putExtras()`.

Por ejemplo, al crear una intent para enviar un correo electrónico con `ACTION_SEND`, puedes especificar el destinatario *para* con la clave `EXTRA_EMAIL` y el *asunto* con la clave `EXTRA_SUBJECT`.

La clase `Intent` especifica muchas constantes `EXTRA_*` para tipos de datos estandarizados. Si tienes que declarar tus propias claves extra (para intents que recibe tu aplicación), asegúrate de incluir el nombre del paquete de la aplicación como prefijo, tal como se muestra en el siguiente ejemplo:

```
const val EXTRA_GIGAWATTS = "com.example.EXTRA_GIGAWATTS"
```

```
static final String EXTRA_GIGAWATTS = "com.example.EXTRA_GIGAWATTS";
```

Advertencia: No uses datos `Parcelable` ni `Serializable` para enviar una intent que debería recibir otra app. Si una app intenta acceder a datos de un objeto `Bundle`, pero no tiene acceso a la clase empaquetada o serializada, el sistema arroja un `RuntimeException`.

Indicadores

Los indicadores se definen en la clase `Intent` que funciona como metadatos de la intent. Los indicadores pueden indicar al sistema Android cómo iniciar una actividad (por

ejemplo, a qué tarea debe pertenecer) y cómo tratarla después de que se inicia (por ejemplo, si pertenece a la lista de actividades recientes).

Para obtener más información, consulta el método `setFlags()`.

Ejemplo de una intent explícita

Una intent explícita es una intent que se usa para iniciar un componente específico de la aplicación, como una actividad o un servicio particular en la aplicación. Para crear una intent explícita, define el nombre de componente del objeto `Intent` (todas las otras propiedades de la intent son opcionales).

Por ejemplo, si creaste un servicio en tu app denominado `DownloadService`, diseñado para descargar un archivo de la Web, puedes iniciarlo con el siguiente código:

```
// Executed in an Activity, so 'this' is the
Context

// The fileUrl is a string URL, such as "http://www.example.com/image.png"
val downloadIntent = Intent(this, DownloadService::class.java).apply {
    data =
Uri.parse
(fileUrl)
}
startService(downloadIntent)
```

```
// Executed in an Activity, so 'this' is the
Context

// The fileUrl is a string URL, such as "http://www.example.com/image.png"
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(
Uri.parse
(fileUrl));
startService(downloadIntent);
```

El constructor `Intent(Context, Class)` proporciona `Context` a la aplicación; y el componente, un objeto `Class`. De esta manera, la intent inicia explícitamente la clase `DownloadService` en la app.

Para obtener más información sobre la creación y el inicio de un servicio, consulta la guía Servicios.

Ejemplo de una intent implícita

Una intent implícita especifica una acción que puede invocar cualquier aplicación en el dispositivo que puede realizar la acción. El uso de una intent implícita es útil cuando la aplicación no puede realizar la acción, pero otras aplicaciones probablemente sí, y quieres que el usuario elija qué aplicación usar.

Por ejemplo, si tienes contenido que quieres que el usuario comparta con otras personas, crea una intent con la acción `ACTION_SEND` y agrega extras que especifiquen el contenido para compartir. Cuando llamas a `startActivity()` con esa intent, el usuario puede elegir una aplicación mediante la cual compartir el contenido.

Advertencia: Es posible que un usuario no tenga *ninguna* aplicación para manejar la intent implícita que envías a `startActivity()`. O bien, es posible que no se pueda acceder a una aplicación debido a restricciones de perfil o a la configuración de un administrador. En ese caso, no se produce la llamada y se bloquea la app. Para verificar que una actividad recibirá la intent, llama a `resolveActivity()` en el objeto `Intent`. Si el resultado no es nulo, hay al menos una aplicación que puede administrar la intent y es seguro llamar a `startActivity()`. Si el resultado es nulo, no debes usar la intent y, si es posible, debes inhabilitar la función que emite la intent. El siguiente ejemplo muestra cómo verificar que la intent resuelva una actividad. Este ejemplo no usa un URI, pero se declara el tipo de datos de la intent para

especificar el contenido que tienen los extras.

```
// Create the text message with a string
val sendIntent = Intent().apply {
    action = Intent.ACTION_SEND
    putExtra(Intent.EXTRA_TEXT, textMessage)
    type = "text/plain"
}// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(packageManager) != null) {
    startActivity(sendIntent)
}
```

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

Cuando se llama a `startActivity()`, el sistema examina todas las aplicaciones instaladas para determinar cuáles pueden manejar este tipo de intent (una intent con la acción `ACTION_SEND` y que tiene datos de texto/sin formato). Si solo hay una aplicación que puede manejarla, esta se abre inmediatamente y se le entrega la intent. Si varias actividades aceptan la intent, el sistema muestra un cuadro de diálogo, como el que se muestra en la figura 2, para que el usuario pueda elegir la aplicación que se debe usar.

Obtén más información sobre el lanzamiento de otras apps en Cómo enviar al usuario a otra app.

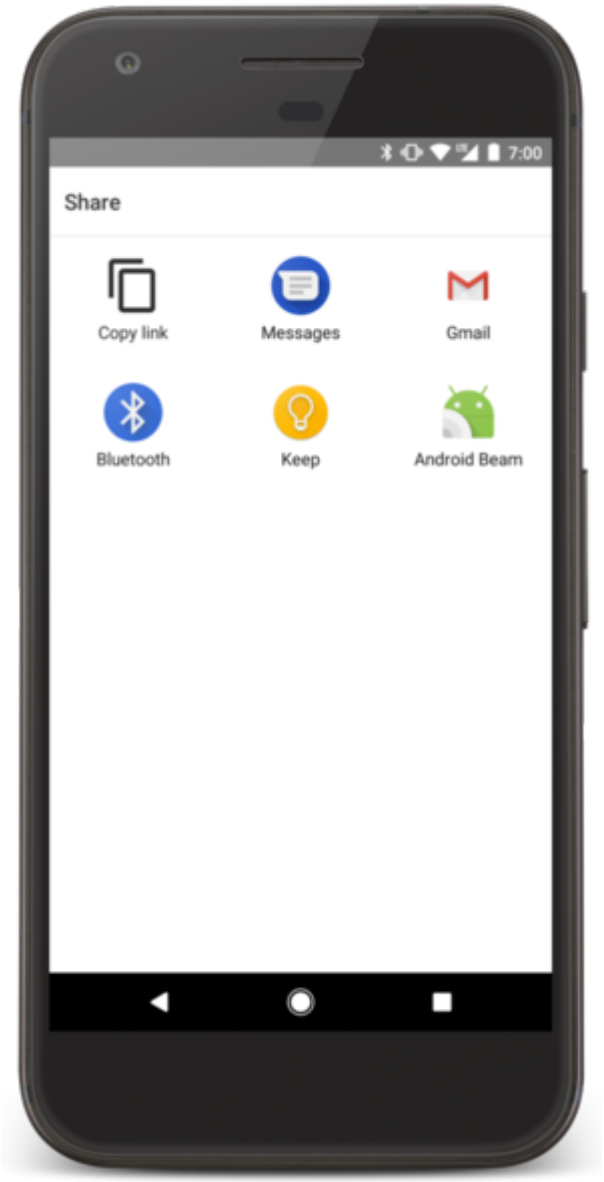


Figura 2: Diálogo de selección

Cómo forzar un selector de apps

Cuando más de una aplicación responde a la intent implícita, el usuario puede seleccionar qué aplicación se debe usar y establecerla como la opción predeterminada para la acción. Esto es útil cuando se realiza una acción para la cual el usuario probablemente desee utilizar la misma aplicación en lo sucesivo, por ejemplo, al abrir una página web (los usuarios suelen preferir un solo navegador web).

No obstante, si varias aplicaciones responden a la intent y es posible que el usuario quiera usar una diferente cada vez, debes mostrar explícitamente un cuadro de diálogo de selección. El cuadro de diálogo de selección le pregunta al usuario qué aplicación desea utilizar para la acción (no puede seleccionar una aplicación predeterminada para la acción). Por ejemplo, cuando tu app realiza la acción de "compartir" con la acción `ACTION_SEND`, es posible que los usuarios deseen concretar la compartición usando una app diferente según la situación en la que estén. Por ello, debes usar siempre el cuadro de diálogo de selección, como se muestra en la figura 2.

Para mostrar el diálogo de selección, crea una `Intent` usando `createChooser()` y transfiérela a `startActivity()`, tal como se muestra en el siguiente ejemplo. Aquí se muestra un diálogo con una lista de apps que responden a la intent transferida al método `createChooser()`, con el texto proporcionado como título del diálogo.

```
val sendIntent = Intent(Intent.ACTION_SEND)
...// Always use string resources for UI text.
// This says something like "Share this photo with"
val title: String = resources.getString(R.string.chooser_title)
```

```
// Create intent to show the chooser dialog
val chooser: Intent = Intent.createChooser(sendIntent, title)// Verify the original intent will resolve to a
if (sendIntent.resolveActivity(packageManager) != null) {
    startActivity(chooser)
}

Intent sendIntent = new Intent(Intent.ACTION_SEND);
...// Always use string resources for UI text.
// This says something like "Share this photo with"
String title = getResources().getString(R.string.chooser_title);
// Create intent to show the chooser dialog
Intent chooser = Intent.createChooser(sendIntent, title);// Verify the original intent will resolve to at lea
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(chooser);
}
```

Cómo recibir una intent implícita

Para informar las intents implícitas que puede recibir tu app, declara uno o más filtros para cada componente de esta con un elemento `<intent-filter>` en el archivo de manifiesto. Cada filtro de intents especifica el tipo de intents que acepta según la acción, los datos y la categoría de la intent. El sistema entregará la intent implícita al componente de la aplicación solo si la intent puede pasar por uno de los filtros.

Nota: Una intent explícita siempre se entrega al destino, independientemente de los filtros de intents que declare el componente.

Un componente de aplicación debe declarar filtros independientes para cada tarea única que puede hacer. Por ejemplo, una actividad de una aplicación de galería de imágenes puede tener dos filtros: uno para ver una imagen y otro para editarla. Cuando comienza la actividad, inspecciona la `Intent` y decide cómo comportarse conforme la información de la `Intent` (como mostrar los controles del editor o no).

Cada filtro de intents está definido por un elemento `<intent-filter>` en el archivo de manifiesto de la app, anidado en el componente correspondiente (como un elemento `<activity>`). En el `<intent-filter>`, puedes especificar el tipo de intents que se aceptará con uno o más de estos tres elementos:

`<action>`
: Declara la acción de la intent aceptada, en el atributo `name`. El valor debe ser el de la string literal de una acción, no la constante de clase.

`<data>`
: Declara el tipo de datos que se acepta, mediante el uso de uno o más atributos que especifican varios aspectos del URI de datos (`scheme`, `host`, `port`, `path`, etc.) y el tipo de MIME.

`<category>`
: Declara la categoría de la intent aceptada, en el atributo `name`. El valor debe ser el de la string literal de una acción, no la constante de clase.

  Nota: Para recibir intents implícitas, *debes incluir* la categoría `CATEGORY_DEFAULT` en el filtro de intents. Los métodos `startActivity()` y `startActivityForResult()` tratan todas las intents como si pertenecieran a la categoría `CATEGORY_DEFAULT`. Si no declaras esta categoría en el filtro de intents, no se aplicará ninguna intent implícita a la actividad.

Por ejemplo, a continuación, se muestra una declaración de actividad con un filtro `de intents para recibir una intent ACTION_SEND` cuando el tipo de datos es un texto:

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
```

```
    </intent-filter>
</activity>
```

Puedes crear un filtro que incluya más de una instancia de `<action>`, `<data>` o `<category>`. Si lo haces, debes asegurarte de que el componente pueda administrar cualquier combinación de esos elementos de filtros.

Cuando quieras administrar varios tipos de intents, pero solo en combinaciones específicas de tipos de acciones, datos y categorías, deberás crear varios filtros de intents.

Una intent implícita se prueba con un filtro comparando la intent con cada uno de los tres elementos. Para que se entregue al componente, la intent debe pasar las tres pruebas. Si no pasa tan solo una de ellas, el sistema Android no la entregará al componente. Sin embargo, como un componente puede tener varios filtros de intents, una intent que no pase por el filtro de un componente quizá pase por otro filtro. Se proporciona más información sobre la manera en que el sistema resuelve las intents en la sección sobre Resolución de intents que aparece más abajo.

Advertencia: El uso de un filtro de intents no es una manera segura de prevenir que otras apps inicien tus componentes. Si bien los filtros de intents restringen un componente para que responda solamente a ciertos tipos de intents implícitas, otra app puede iniciar potencialmente el componente de tu app usando una intent explícita si el desarrollador determina los nombres de los componentes. Si es importante que *solo tu app* pueda iniciar uno de los componentes, no declares filtros de intents en tu manifiesto. En su lugar, establece el atributo `exported` en `"false"` para ese componente.

Para evitar ejecutar accidentalmente el `Service` de otra aplicación, siempre usa una intent explícita cuando inicies tu propio servicio.

Nota: Para todas las actividades, debes declarar tus filtros de intents en el archivo de manifiesto. Sin embargo, los filtros de receptores de emisiones pueden registrarse de forma dinámica llamando a `registerReceiver()`. Puedes anular el registro del receptor con `unregisterReceiver()`. Si lo haces, tu aplicación esperará emisiones específicas solo durante un plazo especificado mientras esté ejecutándose.

Ejemplos de filtros

Para demostrar algunos comportamientos de filtros de intents, se incluye un ejemplo del archivo de manifiesto de una app de intercambio social:

```
<activity android:name="MainActivity">
    <!-- This activity is the main entry, should appear in app launcher -->
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity><activity android:name="ShareActivity">
    <!-- This activity handles "SEND" actions with text data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
    <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <action android:name="android.intent.action.SEND_MULTIPLE"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="application/vnd.google.panorama360+jpg"/>
        <data android:mimeType="image/*"/>
        <data android:mimeType="video/*"/>
    </intent-filter>
```

</activity>

La primera actividad, `MainActivity`, es el punto de entrada principal (la actividad que se abre cuando el usuario lanza inicialmente la app con el ícono de selector):

- La acción `ACTION_MAIN` indica que este es el punto de entrada principal y que no se esperan datos de intents.
- La categoría `CATEGORY_LAUNCHER` indica que el ícono de esta actividad debe colocarse en el selector de la aplicación del sistema. Si el elemento `<activity>` no especifica un ícono con `icon`, el sistema usa el del elemento `<application>`.

Estos dos deben sincronizarse para que la actividad aparezca en el selector de la aplicación.

La segunda actividad, `ShareActivity`, tiene como objetivo facilitar el intercambio de texto y contenido multimedia. Aunque los usuarios pueden ingresar a esta actividad navegando hasta ella desde `MainActivity`, también pueden ingresar a `ShareActivity` directamente desde otra app que emita una intent implícita que coincida con uno de los dos filtros de intents.

Nota: El tipo de MIME, `application/vnd.google.panorama360+jpg`, es un tipo de datos especial que especifica fotos panorámicas que puedes administrar con las API de Google panorama.

Cómo usar una intent pendiente

Un objeto `PendingIntent` es un contenedor de un objeto `Intent`. El propósito principal de un objeto `PendingIntent` es otorgar permiso a una aplicación externa para usar la `Intent` contenida como si se ejecutara desde un proceso de su propia aplicación.

Estos son algunos de los casos de uso principales de una intent pendiente:

- Declarar una intent que se ejecutará cuando el usuario realice una acción con su notificación (el objeto `NotificationManager` del sistema Android ejecuta la `Intent`).
- Declarar una intent que se ejecutará cuando el usuario realice una acción con el widget de tu aplicación (la app de la pantalla principal ejecuta la `Intent`).
- Declarar una intent que se ejecutará en un momento especificado en el futuro (el `AlarmManager` del sistema Android ejecuta la `Intent`).

Como cada objeto `Intent` está diseñado para ser manejado por un tipo específico de componente de la aplicación (`Activity`, `Service` o `BroadcastReceiver`), se debe tener la misma consideración al crear una `PendingIntent`. Cuando use una intent pendiente, la aplicación no la ejecutará con una llamada como `startActivity()`. En cambio, debes declarar el tipo de componente deseado cuando crees la `PendingIntent` llamando al método creador respectivo:

- `PendingIntent.getActivity()` para una `Intent` que inicia una `Activity`.
- `PendingIntent.getService()` para una `Intent` que inicia un `Service`.
- `PendingIntent.getBroadcast()` para una `Intent` que inicia un `BroadcastReceiver`.

A menos que la aplicación esté *recibiendo* intents pendientes de otras aplicaciones, los métodos indicados arriba para crear una `PendingIntent` probablemente sean los únicos métodos `PendingIntent` que necesitarás.

Cada método toma el `Context` actual de la aplicación, la `Intent` que desea encapsular y uno o más indicadores que especifican cómo se debe usar la intent (por ejemplo, si se puede usar más de una vez).

Se proporciona más información sobre el uso de intents pendientes con la documentación para cada caso de uso, como en las guías de Notifications y API de App Widgets.

Resolución de intents

Cuando el sistema recibe una intent implícita para iniciar una actividad, busca la mejor actividad para la intent mediante la comparación de la intent con los filtros de intents

basándose en tres aspectos:

- Acción
- Datos (tanto URI como tipo de datos)
- Categoría

En las siguientes secciones, se describe la manera de hacer coincidir las intents con los componentes apropiados según la forma de declarar el filtro de intents en un archivo de manifiesto de una app.

Prueba de acción

Para especificar las acciones de intents aceptadas, un filtro de intents puede declarar un número de elementos `<action>` de cero en adelante, tal como se muestra en el siguiente ejemplo.

```
<intent-filter>
   <action android:name="android.intent.action.EDIT" />
   <action android:name="android.intent.action.VIEW" />
   ...
</intent-filter>
```

Para pasar el filtro, la acción especificada en la `Intent` debe coincidir con una de las acciones indicadas en el filtro.

Si el filtro no enumera ninguna acción, no hay nada con qué establecer la coincidencia, por lo que ninguna de las intents pasa la prueba. Sin embargo, si una `Intent` no especifica una acción, pasará la prueba (si el filtro contiene, al menos, una acción).

Prueba de categoría

Para especificar las categorías de intents aceptadas, un filtro de intents puede declarar un número de elementos `<category>` de cero en adelante, tal como se muestra en el siguiente ejemplo.

```
<intent-filter>
   <category android:name="android.intent.category.DEFAULT" />
   <category android:name="android.intent.category.BROWSABLE" />
   ...
</intent-filter>
```

Para que una intent pase una prueba de categoría, cada categoría en la `Intent` debe coincidir con la categoría que figura en el filtro. No es necesario que suceda la situación inversa (el filtro de intents puede declarar más categorías de las que se especifican en la `Intent`, y la `Intent` pasará la prueba). Por lo tanto, una intent sin categorías siempre debería pasar esta prueba, independientemente de qué categorías estén declaradas en el filtro.

Nota: Android aplica automáticamente la categoría `CATEGORY_DEFAULT` a todas las intents implícitas que se pasan a `startActivity()` y a `startActivityForResult()`. Por ello, si deseas que tu actividad reciba intents implícitas, debes incluir una categoría para `"android.intent.category.DEFAULT"` en los filtros de intents, tal como se muestra en el ejemplo de `<intent-filter>` anterior.

Prueba de datos

Para especificar los datos de intents que se aceptan, un filtro de intents puede declarar un número de elementos `<data>` de cero en adelante, tal como se muestra en el siguiente ejemplo:

```
<intent-filter>
   <data android:mimeType="video/mpeg" android:scheme="http" ... />
   <data android:mimeType="audio/mpeg" android:scheme="http" ... />
   ...
```

</intent-filter>

Cada elemento <data> puede especificar una estructura de URI y un tipo de datos (tipo de contenido multimedia de MIME). Cada parte del URI es un atributo independiente: `scheme`, `host`, `port` y `path`.

```
<scheme>://<host>:<port>/<path>
```

El siguiente ejemplo muestra valores posibles para estos atributos:

```
content://com.example.project:200/folder/subfolder/etc
```

En este URI, el esquema es `content`; el host, `com.example.project`; el puerto, `200`; y la ruta, `folder/subfolder/etc`.

Cada uno de los atributos es opcional en un elemento <data>, pero existen dependencias lineales:

- Si no se especifica el esquema, se ignora el host.
- Si el host no se especifica, se ignora el puerto.
- Si no se especifican el esquema ni el host, se ignora la ruta de acceso.

Cuando se compara el URI de una intent con una especificación de URI en un filtro, solo se compara con las partes del URI que se incluyen en el filtro. Por ejemplo:

- Si un filtro especifica únicamente un esquema, todos los URI con ese esquema coinciden con el filtro.
- Si un filtro especifica un esquema y una autoridad, pero no una ruta de acceso, todos los URI con el mismo esquema y autoridad pasan el filtro, independientemente de la ruta que tengan.
- Si un filtro especifica un esquema, una autoridad y una ruta de acceso, solo los URI con el mismo esquema, autoridad y ruta pasan el filtro.

Nota: La especificación de una ruta de acceso puede contener un asterisco (*) comodín para requerir solamente una coincidencia parcial del nombre de la ruta.

La prueba de datos compara tanto el URI como el tipo de MIME en la intent con el URI y el tipo de MIME especificados en el filtro. Las reglas son las siguientes:

a. Una intent que no contiene ni un URI ni un tipo de MIME pasa la prueba solo si el filtro no especifica ningún URI ni tipo de MIME.
b. Si una intent contiene un URI, pero no un tipo de MIME (ni explícitamente ni que se pueda inferir del URI), pasa la prueba solo si el URI coincide con el formato de URI del filtro y si el filtro tampoco especifica un tipo de MIME.
c. Si una intent contiene un tipo de MIME, pero no un URI, pasa la prueba solo si el filtro enumera el mismo tipo de MIME y no especifica un formato de URI.
d. Una intent que contiene tanto un URI como un tipo de MIME (explícitamente o que se pueda inferir del URI) pasa la parte de tipo de MIME de la prueba solo si ese tipo coincide con el enumerado en el filtro. Pasa la parte del URI de la prueba si este coincide con un URI del filtro o si tiene un URI `content:` o `file:`, y el filtro no especifica uno. Es decir, se supone que un componente admite datos `content:` y `file:` si el filtro *solo* indica un tipo de MIME.

Nota: Si una intent especifica un tipo de URI o MIME, la prueba de datos fallará si no hay elementos `<data>` en `<intent-filter>`.

Esta última regla (d), refleja la expectativa de que los componentes pueden obtener datos locales de un archivo o un proveedor de contenido. Por lo tanto, sus filtros pueden enumerar solamente un tipo de datos sin nombrar explícitamente los esquemas `content:` o `file:`. El siguiente ejemplo muestra un caso típico en que un elemento <data> indica a Android que el componente puede obtener datos de imagen de un proveedor de contenido y mostrarlos:

<intent-filter>

```
    <data android:mimeType="image/*" />
    ...
</intent-filter>
```

Como la mayoría de los datos son proporcionados por proveedores de contenido, los filtros que especifican un tipo de datos y no especifican un URI son, tal vez, los más comunes.

Otra configuración común es aquella en la que los filtros tienen un esquema y un tipo de datos. Por ejemplo, un elemento `<data>` como el siguiente indica a Android que el componente puede recuperar datos de video de la red para realizar la acción:

```
<intent-filter>
    <data android:scheme="http" android:mimeType="video/*" />
    ...
</intent-filter>
```

Coincidencias de intents

Se establecen coincidencias de las intents con filtros de intents no solo para detectar un componente de destino que se debe activar, sino también para detectar algo sobre el conjunto de componentes del dispositivo. Por ejemplo, la app de página principal completa el selector de la aplicación encontrando todas las actividades con filtros de intents que especifiquen la acción `ACTION_MAIN` y la categoría `CATEGORY_LAUNCHER`. Una coincidencia solo es exitosa si las acciones y categorías de la intent coinciden con el filtro, tal como se describe en la documentación de la clase `IntentFilter`.

Tu aplicación puede usar la coincidencia de intents de forma similar a como lo hace la app de pantalla principal. `PackageManager` tiene un conjunto de métodos `query...()` que muestran todos los componentes que pueden aceptar una intent específica, al igual que una serie similar de métodos `resolve...()` a fin de determinar el mejor componente para responder a la intent. Por ejemplo, `queryIntentActivities()` muestra una lista de todas las actividades que pueden utilizar la intent pasada como un argumento, y `queryIntentServices()` muestra una lista similar de servicios. Ninguno de los métodos activa los componentes; solo enumeran cuáles pueden responder. Existe un método similar, `queryBroadcastReceivers()`, para receptores de emisiones.