

# Módulo 8

Código: 489

## Programación multimedia y dispositivos móviles

Técnico Superior en Desarrollo de  
Aplicaciones Multiplataforma



# UNIDAD 2

## Programación de aplicaciones para dispositivos móviles. Interfaz

Contenidos teóricos



1. Objetivos generales de la unidad
2. Competencias y contribución en la unidad
3. Contenidos conceptuales y procedimentales
4. Evaluación
5. Distribución de los contenidos
6. Sesiones

# 1. Objetivos generales de la unidad

## Objetivos curriculares


1. Generar la estructura de clases necesaria para la aplicación.

2. Analizar y utilizar clases que modelan ventanas, menús, alertas y controles para el desarrollo de aplicaciones gráficas sencillas

3. Utilizar las clases necesarias para el intercambio de mensajes de texto y multimedia.

4. Empaquetar y desplegar aplicaciones desarrolladas en dispositivos móviles reales.

## 2. Competencias y contribución en la unidad



h) Emplear herramientas de desarrollo, lenguajes y componentes visuales, siguiendo las especificaciones y verificando interactividad y usabilidad, para desarrollar interfaces gráficos de usuario en aplicaciones multiplataforma.

	<p>Analizando los diferentes elementos disponibles para la creación de interfaces gráficas y usándolos para la composición de una pantalla con la que el usuario pueda interactuar.</p>	
--	---	--

# 3. Contenidos

## CONTENIDOS CONCEPTUALES

- Framework
- Actividades
- Servicios
- Proveedores de contenido
- Receptores de mensajes
- Intent
- Selector
- Layout
- Fragmentos

## CONTENIDOS PROCEDIMENTALES

- Herramientas y fases de construcción.
- Desarrollo del código.
- Compilación, preverificación, empaquetado y ejecución.
- Depuración.
- Interfaces de usuario. Clases asociadas.
- Contexto gráfico. Imágenes.
- Eventos del teclado.
- Técnicas de animación y sonido.

## 4. Evaluación

a) Se ha generado la estructura de clase necesaria para la aplicación.

c) Se han utilizado las clases necesarias para la conexión y comunicación con dispositivos inalámbricos.

e) Se han empaquetado y desplegado las aplicaciones desarrolladas en dispositivos móviles reales.

b) Se han analizado y utilizado las clases que modelan ventanas, menús, alertas y controles para el desarrollo de aplicaciones gráficas sencillas.

d) Se han utilizado las clases necesarias para el intercambio de mensajes de texto y multimedia.

## 1. INTRODUCCIÓN

Las aplicaciones de Android se escriben en lenguaje de programación Java.

Las herramientas de Android SDK compilan el código, junto con los archivos de recursos y datos, en un APK:

- **Un paquete de Android**, que es un archivo de almacenamiento con el sufijo .apk.
- Un archivo de APK incluye **todos los contenidos de una aplicación** de Android y es el archivo que usan los dispositivos con tecnología Android para instalar la aplicación.



## 1. INTRODUCCIÓN

Una vez instalada en el dispositivo, cada aplicación de Android se aloja en su propia zona de pruebas de seguridad:

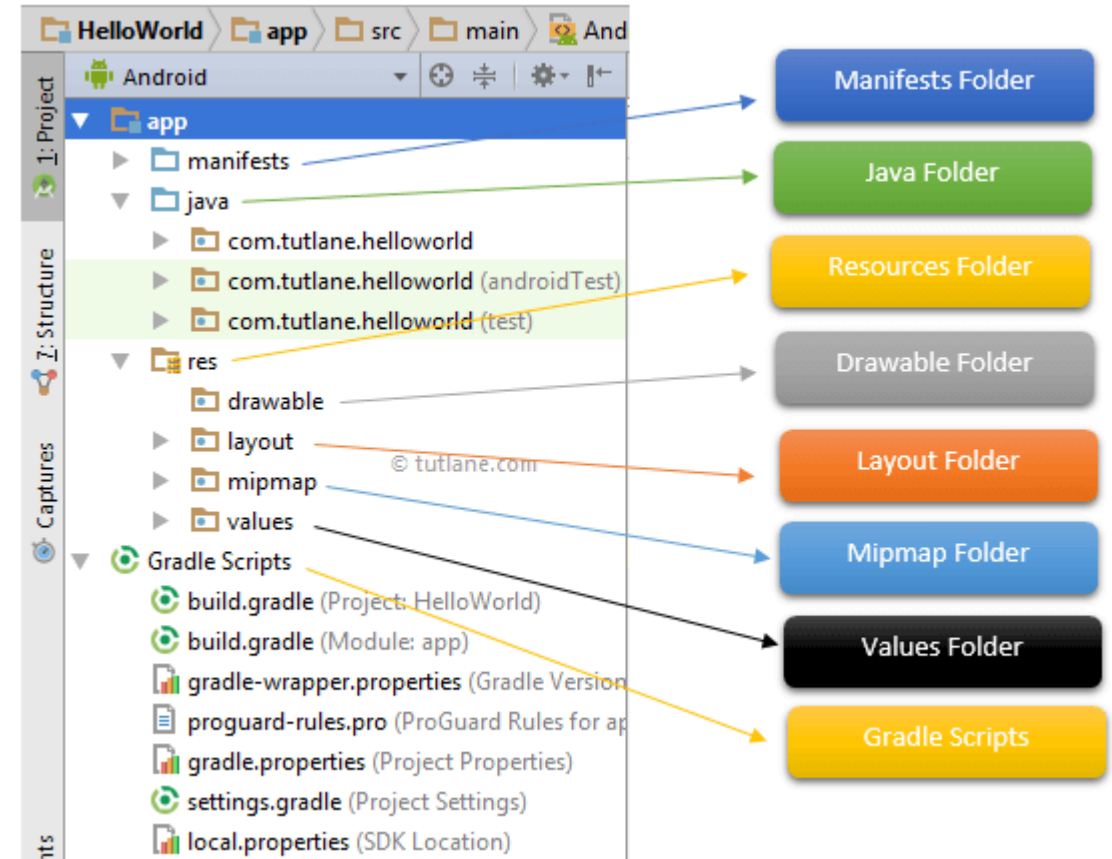
- El sistema operativo Android es un sistema Linux multiusuario en el que cada aplicación es un usuario diferente.
- De forma predeterminada, el sistema le asigna a cada aplicación una ID de usuario de Linux única (solo el sistema utiliza la ID y la aplicación la desconoce). El sistema establece permisos para todos los archivos en una aplicación de modo que solo el ID de usuario asignado a esa aplicación pueda acceder a ellos.
- Cada código de una aplicación se ejecuta de forma independiente en su propio equipo virtual (EV).
- De forma predeterminada, cada aplicación ejecuta su proceso de Linux propio. Android inicia el proceso cuando se requiere la ejecución de alguno de los componentes de la aplicación, luego lo cierra cuando el proceso ya no es necesario o cuando el sistema debe recuperar memoria para otras aplicaciones.

# Sesión 1. Contenidos teóricos

## 2. ARQUITECTURA DE UN PROYECTO

Un proyecto consta de las siguientes partes:

- Los **componentes del framework** central, que definen la aplicación.
- El **archivo de manifiesto**, en el que se declaran los componentes y las funciones necesarias del dispositivo que usará la aplicación.
- Los **recursos independientes del código** de la aplicación, que permiten que la aplicación optimice correctamente su comportamiento para una variedad de configuraciones de dispositivos.



## 2. ARQUITECTURA DE UN PROYECTO

- **La carpeta Java:** contiene el código fuente escrito en java, incluyendo el código de testing con junit. Por defecto, tiene al menos MainActivity.java.
- **El directorio res:** contiene los diferentes recursos importantes que no son código, como imágenes, strings, diseños XML...
  - **Drawable:** Contiene diferentes tipos de imágenes en la aplicación.
  - **Layout:** Contiene los XML que se utilizan para definir la interfaz de usuario de nuestra aplicación.
  - **Mipmap:** Contiene los iconos de la aplicación que se mostrarán en el menú de aplicaciones de Android. Tiene diferentes tamaños dependiendo del de la pantalla (hdpi, mdpi, xhdpi, xxhdpi...)
  - **Values:** Contiene varios ficheros XML con strings, colores, definiciones de estilos...
- **Los Gradle Scripts:** son scripts que ayudan a la construcción automática de nuestro sistema.

## 3. COMPONENTES DE UNA APLICACIÓN

### Actividades

Una actividad representa una pantalla con interfaz de usuario. Por ejemplo, una aplicación de correo electrónico tiene una actividad que muestra una lista de los correos electrónicos nuevos, otra actividad para redactar el correo electrónico y otra actividad para leer correos electrónicos.

Si bien las actividades trabajan juntas para proporcionar una experiencia de usuario consistente en la aplicación de correo electrónico, cada una es independiente de las demás. De esta manera, una aplicación diferente puede inicial cualquiera de estas actividades (si la aplicación de correo electrónico lo permite).

Por ejemplo, una aplicación de cámara puede iniciar la actividad en la aplicación de correo electrónico que redacta el nuevo mensaje para que el usuario comparta una imagen.

## 3. COMPONENTES DE UNA APLICACIÓN

### Servicios

Un servicio es un componente que se ejecuta en segundo plano para realizar operaciones prolongadas o tareas para procesos remotos. Un servicio no proporciona una interfaz de usuario.

Por ejemplo, un servicio podría reproducir música en segundo plano mientras el usuario se encuentra en otra aplicación, o podría capturar datos en la red sin bloquear la interacción del usuario con una actividad.

Otro componente, como una actividad, puede iniciar el servicio y permitir que se ejecute o enlazarse a él para interactuar.

## 3. COMPONENTES DE UNA APLICACIÓN

### Proveedores de contenido

Un proveedor de contenido administra un conjunto compartido de datos de la app. Puedes almacenar los datos en el sistema de archivos, en una base de datos SQLite, en la Web o en cualquier otra ubicación de almacenamiento persistente a la que tu aplicación pueda acceder. A través de este, otras aplicaciones pueden consultar o incluso modificar los datos (si el proveedor de contenido lo permite).

Por ejemplo, el sistema Android proporciona un proveedor de contenido que administra la información de contacto del usuario. De esta manera, cualquier app con los permisos correspondientes puede consultar parte del proveedor de contenido (como `ContactsContract.Data`) para la lectura y escritura de información sobre una persona específica.

Los proveedores de contenido también son útiles para leer y escribir datos privados de tu aplicación y que no se comparten. Por ejemplo, la aplicación de ejemplo Bloc de notas usa un proveedor de contenido para guardar notas.

## 3. COMPONENTES DE UNA APLICACIÓN

### Receptores de mensajes

Un receptor de mensajes es un componente que responde a los anuncios de mensajes en todo el sistema. Muchos mensajes son originados por el sistema; por ejemplo, un mensaje que anuncie que se apagó la pantalla, que la batería tiene poca carga o que se tomó una foto. Las aplicaciones también pueden iniciar mensajes; por ejemplo, para permitir que otras aplicaciones sepan que se descargaron datos al dispositivo y están disponibles para usarlos.

Si bien los receptores de mensajes no exhiben una interfaz de usuario, pueden crear una notificación de la barra de estado para alertar al usuario cuando se produzca un evento de mensaje. Aunque, comúnmente, un receptor de mensajes es simplemente una "puerta de enlace" a otros componentes y está destinado a realizar una cantidad mínima de trabajo. Por ejemplo, podría iniciar un servicio para que realice algunas tareas en función del evento.





## 5. ACTIVACIÓN DE COMPONENTES

Tres de los cuatro tipos de componentes (actividades, servicios y receptores de mensajes) se activan mediante un mensaje asíncrono llamado intent. Las intents enlazan componentes individuales en tiempo de ejecución (son como mensajeros que solicitan una acción de otros componentes), ya sea que el componente le pertenezca a tu aplicación o a otra.

Una intent se crea con un objeto Intent, que define un mensaje para activar un componente específico o un tipo específico de componente; una intent puede ser explícita o implícita, respectivamente.

Para actividades y servicios, una intent define la acción a realizar (por ejemplo, "ver" o "enviar" algo) y puede especificar el URI de los datos en los que debe actuar (entre otras cosas que el componente que se está iniciando podría necesitar saber). Por ejemplo, una intent podría transmitir una solicitud para que una actividad muestre una imagen o abra una página web.

## 5. ACTIVACIÓN DE COMPONENTES

El otro tipo de componente, proveedor de contenido, no se activa mediante intents, sino a través de solicitudes de un ContentResolver. El solucionador de contenido aborda todas las transacciones directas con el proveedor de contenido, de modo que el componente que realiza las transacciones con el proveedor no deba hacerlo y, en su lugar, llame a los métodos del objeto ContentResolver. Esto deja una capa de abstracción entre el proveedor de contenido y el componente que solicita información (por motivos de seguridad).

## 5. ACTIVACIÓN DE COMPONENTES

Existen métodos independientes para activar cada tipo de componente:

- Se puede iniciar una actividad (o asignarle algo nuevo para hacer) al pasar una Intent a `startActivity()` o `startActivityForResult()` (cuando se quiera que la actividad devuelva un resultado).
- Se puede iniciar un servicio (o darle instrucciones nuevas a un servicio en curso) al pasar una Intent a `startService()`. O se puede establecer un enlace con el servicio al pasar una Intent a `bindService()`.
- Se puede iniciar la transmisión de un mensaje pasando un Intent a métodos como `sendBroadcast()`, `sendOrderedBroadcast()`, o `sendStickyBroadcast()`.
- Se puede realizar una consulta a un proveedor de contenido llamando a `query()` en un `ContentResolver`.

## 6. ARCHIVO MANIFEST

Para que el sistema Android pueda iniciar un componente de la app, el sistema debe reconocer la existencia de ese componente leyendo el archivo `AndroidManifest.xml` de la app (el archivo de “manifiesto”). La aplicación debe declarar todos sus componentes en este archivo, que debe encontrarse en la raíz del directorio de proyectos de la aplicación.

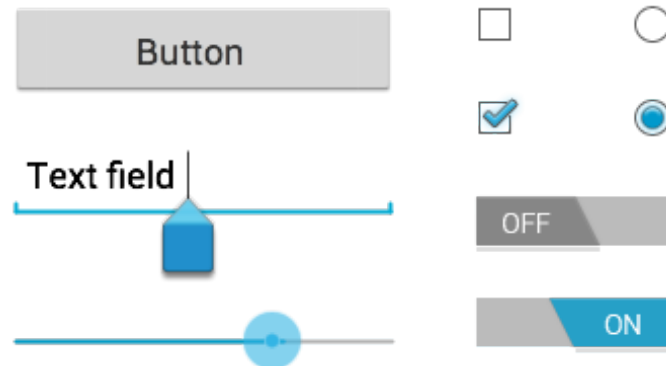
El manifiesto puede hacer ciertas cosas además de declarar los componentes de la aplicación, como por ejemplo:

- Identificar los permisos de usuario que requiere la aplicación, como acceso a Internet o acceso de lectura para los contactos del usuario.
- Declarar el nivel de API mínimo requerido por la aplicación en función de las API que usa la aplicación.
- Declarar características de hardware y software que la aplicación usa o exige, como una cámara, servicios de bluetooth o una pantalla multitáctil.
- Bibliotecas de la API a las que la aplicación necesita estar vinculada (además de las Android framework API), como la biblioteca Google Maps .

## 1. CONTROLES DE ENTRADA

Los controles de entrada son los componentes interactivos de la interfaz de usuario de tu app. Android ofrece una gran variedad de controles que puedes usar en tu IU, como botones, campos de texto, barras de búsqueda, casillas de verificación, botones de zoom, botones de activación o desactivación y muchos más.

Cada control de entrada admite un conjunto específico de eventos de entrada de modo que puedas manipular eventos, como cuando el usuario ingresa texto o toca un botón.



# Sesiones 2 - 4. Contenidos teóricos

## 1. CONTROLES DE ENTRADA

TIPO DE CONTROL	DESCRIPCIÓN	CLASES RELACIONADAS
<a href="#">Botón</a>	Botón que el usuario puede presionar o en el que puede hacer clic para realizar una acción.	<a href="#">Button</a>
<a href="#">Campo de texto</a>	Campo de texto editable.	<a href="#">EditText</a>
<a href="#">Casilla de verificación</a>	Conmutador de selección y deselección que el usuario puede activar o desactivar.	<a href="#">CheckBox</a>
<a href="#">Botón de selección</a>	Similar a las casillas de verificación, excepto porque solo se puede seleccionar una opción en el grupo.	<a href="#">RadioGroup</a> <a href="#">RadioButton</a>
<a href="#">Botón para activar o desactivar</a>	Botón de activación y desactivación con un indicador luminoso.	<a href="#">ToggleButton</a>
<a href="#">Control de número</a>	Una lista desplegable que permite a los usuarios seleccionar un valor de un conjunto.	<a href="#">Spinner</a>
<a href="#">Selectores</a>	Un cuadro de diálogo para que los usuarios seleccionen un valor individual para un conjunto con los botones de arriba y abajo o mediante un gesto de deslizamiento.	<a href="#">DatePicker</a> , <a href="#">TimePicker</a>

## 2. LAYOUTS

Un diseño define la estructura visual para una interfaz de usuario, como la IU para una actividad o widget de una app. Puedes declarar un diseño de dos maneras:

- **Declarar elementos de la IU en XML.** Android proporciona un vocabulario XML simple que coincide con las clases y subclases de vistas, como las que se usan para widgets y diseños.
- **Crear una instancia de elementos del diseño en tiempo de ejecución.** La aplicación puede crear objetos View y ViewGroup (y manipular sus propiedades) programáticamente.

## 2. LAYOUTS

El framework de Android ofrece la flexibilidad de usar uno o ambos de estos métodos para declarar y administrar la IU de tu aplicación. Por ejemplo, se podrían declarar los diseños predeterminados de la aplicación en XML, incluidos los elementos de pantalla que aparecerán en ellos y sus propiedades. Tras esto, se podría agregar código en la aplicación para modificar el estado de los objetos de la pantalla, incluidos los declarados en XML, en tiempo de ejecución.

Al usar vocabulario XML de Android, se pueden crear rápidamente diseños de IU y de los elementos de pantalla que contienen, de la misma manera que creas páginas web en HTML, con una serie de elementos anidados.



## 2. LAYOUTS

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

## 2. LAYOUTS

ID: Identificación exclusiva del objeto. La sintaxis para un ID dentro de una etiqueta XML es la siguiente.

```
android:id="@+id/my_button"
```

Para crear vistas y hacer referencia a ellas desde la aplicación, se puede seguir este patrón común:

1. Definir una vista / un widget en el archivo de diseño y asignarle un ID único:

```
<Button android:id="@+id/my_button"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="@string/my_button_text"/>
```

2. Luego, crear una instancia del objeto View y capturarlo desde el diseño (generalmente en el método onCreate())

```
Button myButton = (Button) findViewById(R.id.my_button);
```

## 1. BUTTON

Elemento de la interfaz que se puede pulsar para realizar una acción.

Definición en el XML del layout:

```
<Button  
    android:id="@+id/button_id"  
    android:layout_height="wrap_content"  
    android:layout_width="wrap_content"  
    android:text="@string/self_destruct" />
```

## 1. BUTTON

Uso dentro de una actividad:

```
public class MyActivity extends Activity {  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        setContentView(R.layout.content_layout_id);  
  
        final Button button = findViewById(R.id.button_id);  
        button.setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) {  
                // Code here executes on main thread after user presses button  
            }  
        });  
    }  
}
```

## 1. BUTTON

Otra manera de utilizar el onClick:

```
<?xml version="1.0" encoding="utf-8"?>
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

En Java escribiríamos:

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

## 2. EDITTEXT

Elemento de la interfaz que nos permite introducir o modificar texto.

Definición en el XML del layout:

```
<EditText  
    android:id="@+id/plain_text_input"  
    android:layout_height="wrap_content"  
    android:layout_width="match_parent"  
    android:inputType="text"/>
```

## 2. EDITTEXT

Se puede utilizar TextWatcher para saber si un texto ha cambiado.

```
field1.addTextChangedListener(new TextWatcher() {  
  
    @Override  
    public void afterTextChanged(Editable s) {}  
  
    @Override  
    public void beforeTextChanged(CharSequence s, int start,  
        int count, int after) {}  
}  
  
@Override  
public void onTextChanged(CharSequence s, int start,  
    int before, int count) {  
    if(s.length() != 0)  
        field2.setText("");  
}  
});
```

## 3. CHECKBOX

Botón con dos estados (checked o unchecked).

Ejemplo de uso en una actividad:

```
public class MyActivity extends Activity {  
    protected void onCreate(Bundle icle) {  
        super.onCreate(icle);  
  
        setContentView(R.layout.content_layout_id);  
  
        final CheckBox checkBox = (CheckBox) findViewById(R.id.checkbox_id);  
        if (checkBox.isChecked()) {  
            checkBox.setChecked(false);  
        }  
    }  
}
```



## 4. RADIOBUTTON

Tienen que ir dentro de un RadioGroup (Definido en el XML).

```
<RadioGroup
    android:id="@+id/radio"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignEnd="@+id/button2"
    android:layout_gravity="center_horizontal">
<RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Pizza"
    android:id="@+id/radioButton"
    android:layout_marginTop="61dp"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:checked="false"
    android:onClick="onRadioButtonClicked"/>
```

## 4. RADIOBUTTON

```
<RadioButton  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Hamburguesa"  
    android:id="@+id/radioButton2"  
    android:layout_below="@+id/radioButton"  
    android:layout_centerHorizontal="true"  
    android:checked="false"  
    android:onClick="onRadioButtonClicked"/>  
</RadioGroup>
```

## 4. RADIOBUTTON

En la actividad se implementaría el siguiente listener:

```
public void onRadioButtonClicked(View view) {  
  
    // Is the button now checked?  
  
    boolean checked = ((RadioButton) view).isChecked();  
  
    // hacemos un case con lo que ocurre cada vez que pulsemos un botón  
  
    switch(view.getId()) {  
        case R.id.radioButton:  
            if (checked)  
                //  
                break;  
        case R.id.radioButton2:  
            if (checked)  
                //  
                break;  
    }  
}
```

## 4. RADIOBUTTON

Otra forma sería:

```
1 r1=(RadioButton)findViewById(R.id.radioButton);
2 r2=(RadioButton)findViewById(R.id.radioButton2);
3 r3=(RadioButton)findViewById(R.id.radioButton3);
4 r4=(RadioButton)findViewById(R.id.radioButton4);
5 r5=(RadioButton)findViewById(R.id.radioButton5);
```

Luego, implementar un método y usarlos por ejemplo con un if:

```
1 if (r1.isChecked()==true) {
2     //código
3 } else
4     if (r2.isChecked()==true) {
5         // código
6     }
7 ...
```

## 5. TOGGLEBUTTON

Botón con un indicador de luz que tiene dos estados: ON y OFF.  
Configuración en el XML:

```
<ToggleButton  
    android:id="@+id/simpleToggleButton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"/>
```

Código de la actividad para leer si el botón está activo:

```
/*Add in Oncreate() funtion after setContentView*/  
ToggleButton simpleToggleButton = (ToggleButton) findViewById(R.id.simpleToggleButton); // initiate a  
Boolean ToggleButtonState = simpleToggleButton.isChecked(); // check current state of a toggle button
```

## 6. TEXTVIEW

Muestra un texto al usuario.

Configuración en el XML (dentro de un layout):

```
<TextView
    android:id="@+id/text_view_id"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="@string/hello" />
```

Código en la actividad para cambiar el texto:

```
public class MainActivity extends Activity {

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final TextView helloTextView = (TextView) findViewById(R.id.text_view_id);
        helloTextView.setText(R.string.user_greeting);
    }
}
```

## 7. IMAGEVIEW

Muestra una imagen de los recursos al usuario.

Configuración en el XML (dentro de un layout):

```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@mipmap/ic_launcher"  
/>
```

## 8. IMAGEBUTTON

Botón que, en vez de texto, muestra una imagen al usuario.

```
<ImageButton  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/button_icon"  
... />
```



## 8. IMAGEBUTTON

Existen diferentes estados, por lo que en el XML podemos definir una imagen para cada estado. Ejemplo de la configuración en el XML:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:state_pressed="true"
        android:drawable="@drawable/button_pressed" /> <!-- pressed -->
  <item android:state_focused="true"
        android:drawable="@drawable/button_focused" /> <!-- focused -->
  <item android:drawable="@drawable/button_normal" /> <!-- default -->
</selector>
```

Se puede cambiar la imagen con el método `ImageView.setImageResource(int)`.

## 9. CREACIÓN DE APLICACIONES MULTIIDIOMA

Para crear aplicaciones multiidioma, hay que establecer recursos alternativos para distintas lenguas o lengua-región.

Por ejemplo:

- Res/values-fr contendrá las cadenas de caracteres traducidas al francés.
- Res/values-en contendrá las cadenas de caracteres traducidas al inglés.

De esta manera, si cambiamos el idioma de nuestro móvil, conseguiremos tener la app en un idioma alternativo.

```
MyProject/  
  res/  
    values/  
      strings.xml  
    values-es/  
      strings.xml  
    values-fr/  
      strings.xml
```

## 9. CREACIÓN DE APLICACIONES MULTIIDIOMA

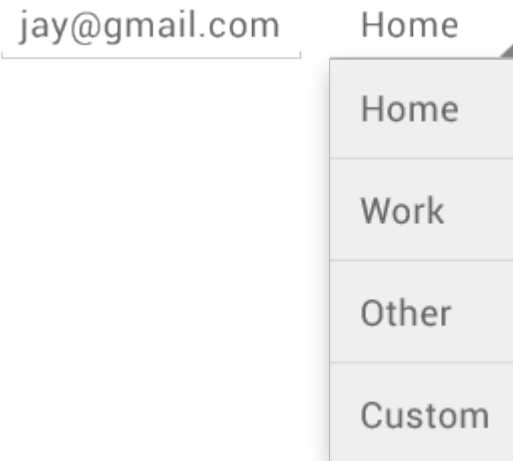
Con la vista como Project, le damos a:

- Res → New... → New Resource Directory
  - En él, ponemos Locale y seleccionamos values-XX (values-en, values-fr...)
- Res → New... → New Resource File
  - Strings.xml

Tras esto, añadimos a los distintos ficheros los distintos valores de la misma string que queramos añadir (mismo ID).

## 1. SPINNER

Los controles de número ofrecen una manera rápida de seleccionar un valor de un conjunto. En el estado predeterminado, un control de número muestra su valor actualmente seleccionado.



## 1. SPINNER

Agregar control al diseño XML:

```
<Spinner  
    android:id="@+id/planets_spinner"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content" />
```

## 1. SPINNER

Cuando el usuario seleccione un elemento del menú, el objeto Spinner recibe un evento en relación con el elemento seleccionado:

```
public class SpinnerActivity extends Activity implements OnItemSelectedListener {  
    ...  
  
    public void onItemSelected(AdapterView<?> parent, View view,  
                               int pos, long id) {  
        // An item was selected. You can retrieve the selected item using  
        // parent.getItemAtPosition(pos)  
    }  
  
    public void onNothingSelected(AdapterView<?> parent) {  
        // Another interface callback  
    }  
}
```

## 2. MATRIZ DE STRINGS

Por ejemplo, si las opciones disponibles para tu control de número son predeterminadas, puedes proporcionarlas con una matriz de strings definida en un archivo de recursos de strings.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string-array name="planets_array">
    <item>Mercury</item>
    <item>Venus</item>
    <item>Earth</item>
    <item>Mars</item>
    <item>Jupiter</item>
    <item>Saturn</item>
    <item>Uranus</item>
    <item>Neptune</item>
  </string-array>
</resources>
```

## 2. MATRIZ DE STRINGS

Con una matriz como la anterior, se puede usar el siguiente código en la Activity o Fragment para suministrar al control de número la matriz mediante una instancia de ArrayAdapter:

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);  
// Create an ArrayAdapter using the string array and a default spinner layout  
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,  
    R.array.planets_array, android.R.layout.simple_spinner_item);  
// Specify the layout to use when the list of choices appears  
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);  
// Apply the adapter to the spinner  
spinner.setAdapter(adapter);
```



## 3. MENÚS

Los menús son un componente común de la interfaz de usuario en muchos tipos de aplicaciones. Para proporcionar una experiencia de usuario conocida y uniforme, se debe usar las Menu API para presentar al usuario acciones y otras opciones en las actividades.

A partir de Android 3.0 (nivel de API 11), los dispositivos con Android ya no tienen que proporcionar un botón Menú dedicado.

Aunque el diseño de la experiencia de usuario para algunos elementos del menú ha cambiado, la semántica para definir un conjunto de acciones y opciones sigue basándose en las API de Menu.

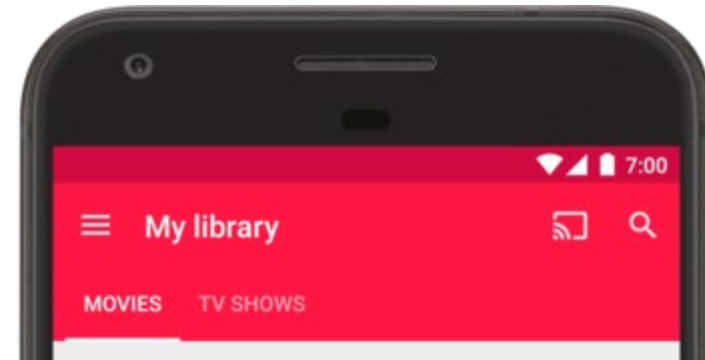
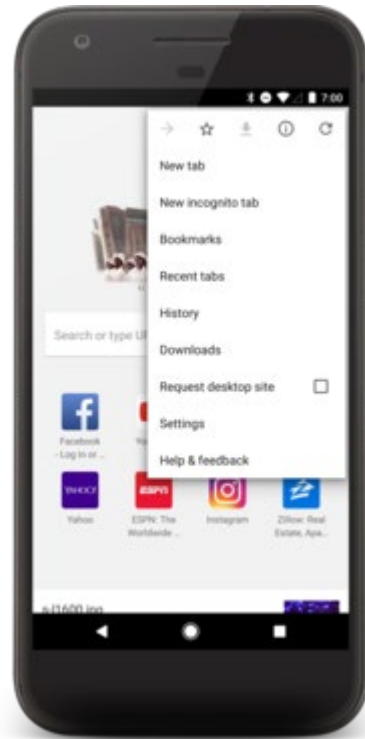
## 3. MENÚS

Tipos:

- **Menú de opciones y barra de app**
  - El menú de opciones es dónde debes incluir las acciones y otras opciones que son relevantes para el contexto de la actividad actual, como "Buscar, "Redactar correo electrónico" y "Ajustes".
- **Menú contextual y modo de acción contextual**
  - Un menú contextual ofrece acciones que afectan un elemento o marco contextual específicos en la IU.
- **Menú emergente**
  - Un PopupMenu es un menú modal anclado a una View. Aparece debajo de la vista anclada si hay espacio o, de lo contrario, sobre esta.

## 3. MENÚS

Menú de opciones y barra de app



# Sesión 7. Contenidos teóricos

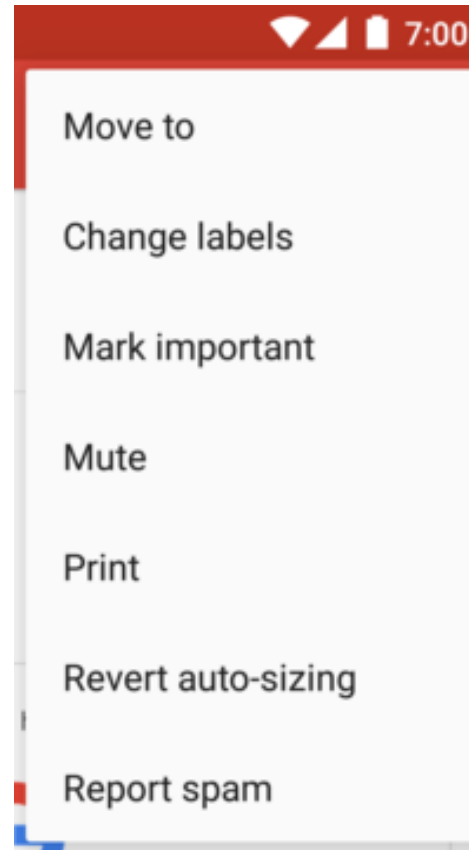
## 3. MENÚS

### Menús contextuales



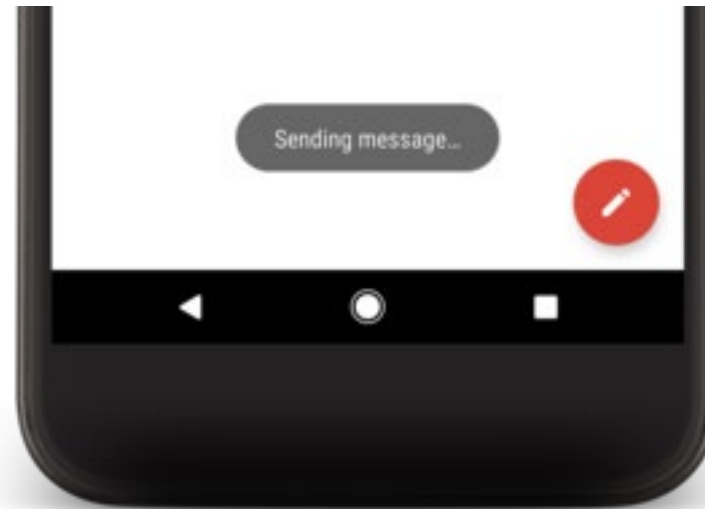
## 3. MENÚS

Menús emergentes



## 4. TOAST

Consiste en un pequeño mensaje pop-up que nos da información sobre algo. Desaparecen automáticamente tras un tiempo.



## 4. TOAST

En código, será de la siguiente manera:

```
Context context = getApplicationContext();  
CharSequence text = "Hello toast!";  
int duration = Toast.LENGTH_SHORT;  
  
Toast toast = Toast.makeText(context, text, duration);  
toast.show();
```

Para posicionarlo:

```
toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);
```

## 4. TOAST

También podemos crear una View propia del Toast en otro diseño XML:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/custom_toast_container"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="8dp"
    android:background="#DAAA"
    >
    <ImageView android:src="@drawable/droid"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="8dp"
        />
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#FFF"
        />
</LinearLayout>
```



## 4. TOAST

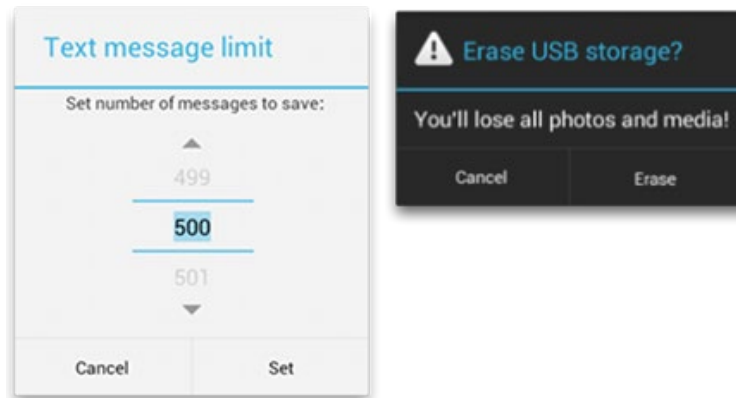
Y después lo lanzaríamos desde la actividad en Java:

```
LayoutInflater inflater = getLayoutInflater();  
View layout = inflater.inflate(R.layout.custom_toast,  
    (ViewGroup) findViewById(R.id.custom_toast_container));  
  
TextView text = (TextView) layout.findViewById(R.id.text);  
text.setText("This is a custom toast");  
  
Toast toast = new Toast(getApplicationContext());  
toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);  
toast.setDuration(Toast.LENGTH_LONG);  
toast.setView(layout);  
toast.show();
```

## 1. CUADROS DE DIÁLOGO

Un diálogo es una ventana pequeña que le indica al usuario que debe tomar una decisión o ingresar información adicional.

Un diálogo no ocupa toda la pantalla y generalmente se usa para eventos modales que requieren que los usuarios realicen alguna acción para poder continuar.



## 1. CUADROS DE DIÁLOGO

Tipos de diálogo:

- **AlertDialog**
  - Un diálogo que puede mostrar un título, hasta tres botones, una lista de elementos seleccionables o un diseño personalizado.
- **DatePickerDialog**
  - Un diálogo con una IU predefinida que le permite al usuario seleccionar una fecha.
- **TimePickerDialog**
  - Un diálogo con una IU predefinida que le permite al usuario seleccionar una hora.

## 1. CUADROS DE DIÁLOGO

Ejemplo de un AlertDialog básico administrado dentro de un DialogFragment:

```
public class FireMissilesDialogFragment extends DialogFragment {  
    @Override  
    public Dialog onCreateDialog(Bundle savedInstanceState) {  
        // Use the Builder class for convenient dialog construction  
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());  
        builder.setMessage(R.string.dialog_fire_missiles)  
            .setPositiveButton(R.string.fire, new DialogInterface.OnClickListener() {  
                public void onClick(DialogInterface dialog, int id) {  
                    // FIRE ZE MISSILES!  
                }  
            })  
            .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {  
                public void onClick(DialogInterface dialog, int id) {  
                    // User cancelled the dialog  
                }  
            });  
        // Create the AlertDialog object and return it  
        return builder.create();  
    }  
}
```

Fire missiles?

CANCEL

FIRE

## 2. FRAGMENTOS

Un Fragment representa un comportamiento o una parte de la interfaz de usuario en una Activity.

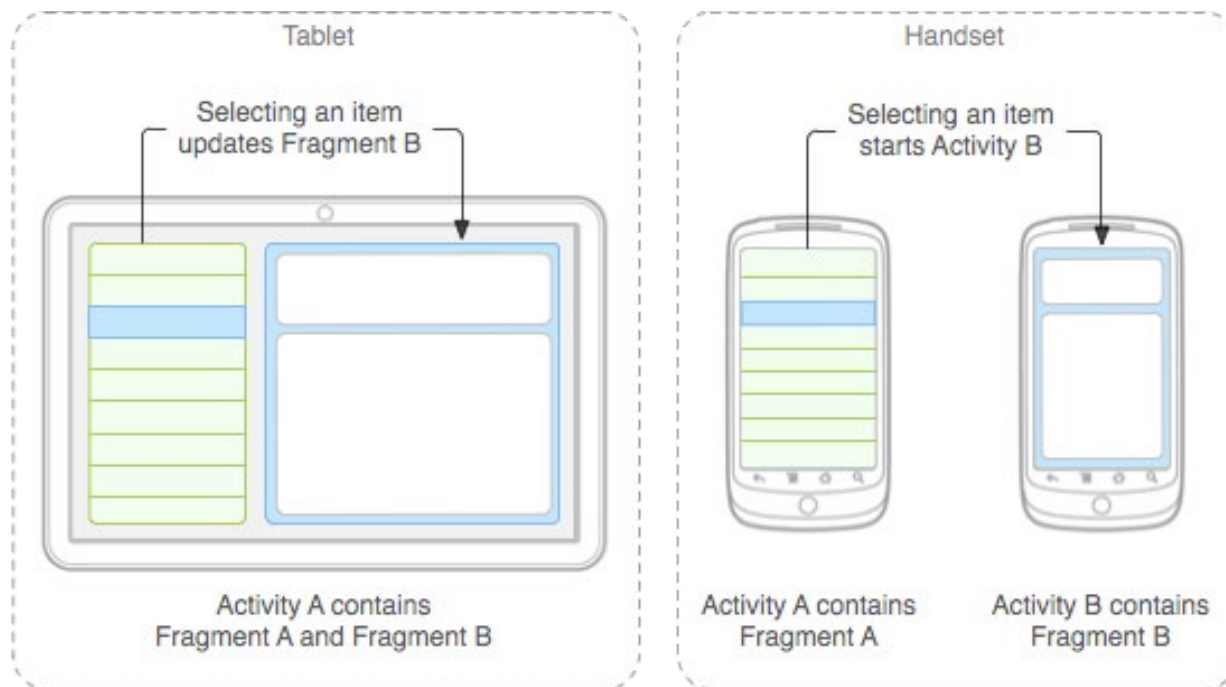
Se pueden combinar múltiples fragmentos en una sola actividad para crear una IU multipanel y volver a usar un fragmento en múltiples actividades.

Se puede pensar en un fragmento como una sección modular de una actividad que tiene su ciclo de vida propio, recibe sus propios eventos de entrada y que se pueden agregar o quitar mientras la actividad se esté ejecutando (algo así como una "subactividad" que puedes volver a usar en diferentes actividades).

## 2. FRAGMENTOS

- Un fragmento siempre debe estar integrado a una actividad y el ciclo de vida del fragmento se ve directamente afectado por el ciclo de vida de la actividad anfitriona.
- Cuando agregas un fragmento como parte del diseño de tu actividad, este se ubica en ViewGroup, dentro de la jerarquía de vistas de la actividad y el fragmento define su propio diseño de vista.
- Se debe diseñar cada fragmento como un componente modular y reutilizable de la actividad.
  - Diseñarlo para volver a utilizarlo.
  - Evitar la manipulación directa de un fragmento desde otro fragmento.

## 2. FRAGMENTOS



## 2. FRAGMENTOS

Agregar a la interfaz de usuario:

```
public static class ExampleFragment extends Fragment {  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
                             Bundle savedInstanceState) {  
        // Inflate the layout for this fragment  
        return inflater.inflate(R.layout.example_fragment, container, false);  
    }  
}
```



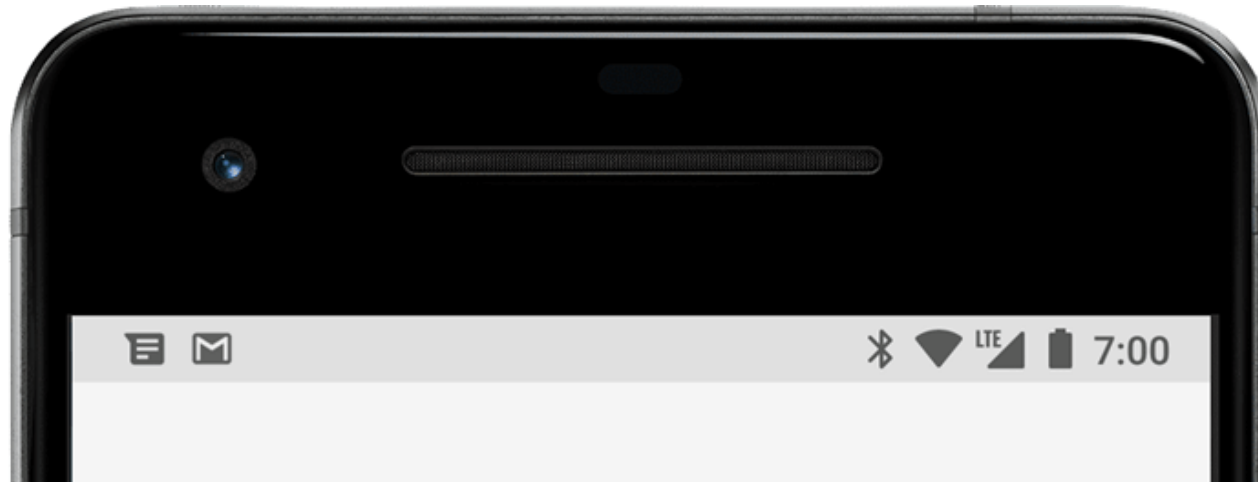
## 2. FRAGMENTOS

Declarar un fragmento en el diseño XML:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.ArticleListFragment"
        android:id="@+id/list"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
        android:id="@+id/viewer"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
</LinearLayout>
```

## 3. NOTIFICACIONES

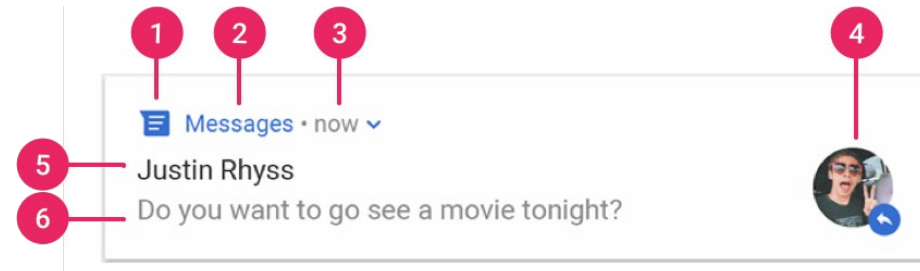
Las notificaciones push son mensajes que recibimos en el dispositivo y han sido emitidos desde un servidor que gestiona el envío de información relevante al usuario. Ofrecen al usuario información en tiempo real sobre las novedades que ocurren en la aplicación y los datos que gestionan.



## 3. NOTIFICACIONES

Anatomía de una notificación

1. Icono pequeño (setSmallIcon())
2. Nombre de la App
3. Hora en la que se recibe (setWhen())
4. Icono grande (setLargeIcon())
5. Título de la notificación (setContentTitle())
6. Texto de la notificación (setContentText())



## 3. NOTIFICACIONES

Tipos:

- Notificaciones en la pantalla de bloqueo (Android 5+)
- Notificaciones heads-up (Android 5+)
- Agrupables (Android 7+)
- Acciones en las notificaciones (Android 7+)
- Notificaciones en la App (Android 8+)
- Canales de notificaciones (Android 8+)
- Importancia de las notificaciones (Android 8+)
- En distintos dispositivos (Wear OS, etc.)

## 3. NOTIFICACIONES

### Notificación básica

- Añadir a build.gradle la siguiente dependencia:

```
dependencies {  
    implementation "com.android.support:support-compat:28.0.0"  
}
```

- Y en el Java pondríamos:

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this, CHANNEL_ID)  
    .setSmallIcon(R.drawable.notification_icon)  
    .setContentTitle(textTitle)  
    .setContentText(textContent)  
    .setPriority(NotificationCompat.PRIORITY_DEFAULT);
```