

Caso práctico



Una vez que **Maria y Juan** han realizado un amplio repaso del mundo de las tecnologías móviles, **Ada** quiere que empiecen a realizar sus primeras aplicaciones. Por ahora, lo único que han hecho ha sido generar un ejemplo y han visto los ficheros más importantes que se han generado. Han podido probar la aplicación sencilla tanto en un emulador como en un dispositivo real, pero aún no se consideran "autores" de ese resultado. Ellos quieren comenzar a escribir el código de sus propias aplicaciones, aunque sean muy básicas.

Caso práctico

Maria y Juan ya se han familiarizado con el proceso de generación de una aplicación Android, llega el momento de empezar a escribir código propio.

Dado que ellos ya tienen conocimientos de programación, no se trata de volver a aprender lo que era un bucle, una clase o un objeto.

Ada les aconseja que empiecen por lo que es más específico de esta tecnología: las **interfaces de usuario**. Es entonces cuando les pregunta si recuerdan la unidad sobre interfaces que estudiaron en el módulo de Programación. Los conceptos que aparecían en esa unidad van a volver a ser utilizados ahora:

- ✓ componentes gráficos;
- ✓ pantalla;
- ✓ contenedor;
- ✓ evento;
- ✓ listener;
- ✓ manejador de eventos;
- ✓ etc...



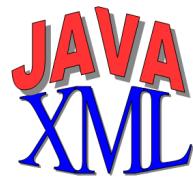
La interfaz de usuario se refiere a todo aquello que el usuario puede ver y con lo que puede interactuar en cualquier aplicación. Android incluye componentes prediseñados que podemos utilizar en nuestras pantallas tales como objetos de tipo **layout** (contenedores para organizar la estructura de la interfaz) y **views** (controles con los que el usuario podrá interactuar). También se incluyen otros módulos especiales como **dialogs**, **notifications** y **menus** como veremos en esta unidad.

Como vimos en la anterior unidad, los ficheros generados en una aplicación Android están escritos en Java y **XML**. Para desarrollar las interfaces de usuario necesarias en cualquier aplicación, trabajaremos indistintamente con ambos lenguajes:

✓ **Programación Java imperativa**: sigue la misma filosofía de la programación de interfaces Java aunque se añaden nuevas clases específicas de Android.

✓ **Definición XML declarativa**: mediante ficheros **XML** se define la estructura que tendrá cada pantalla y los componentes que la integran. La implementación de la lógica seguirá realizándose en Java.

Ambos lenguajes son necesarios para el desarrollo de cualquier aplicación Android, por ello vamos a estudiar los componentes imprescindibles necesarios para implementar de forma fácil interfaces de usuario que sean intuitivas, uniformes y de fácil manejo para el usuario final pero que a su vez sean capaces de sorprenderle también. Tenemos que pensar que la calidad de la interfaz de usuario es un factor importante para que nuestra aplicación tenga éxito o no.



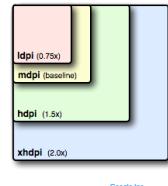
1.1.- La pantalla de Android

Vamos a destacar una de las características principales que debemos tener en cuenta antes de comenzar a desarrollar la interfaz de usuario: **la pantalla**.

Como desarrolladores de aplicaciones Android, nuestro objetivo es que las aplicaciones se puedan instalar en el máximo número posible de terminales, tanto smartphones como tablets. Como sabemos, cada dispositivo tiene un tamaño distinto de pantalla y eso nos va a condicionar mucho a la hora de elaborar nuestras aplicaciones.

Pero el tamaño de la pantalla no es lo único que hay que tener en cuenta, Android también considera las diferentes **densidades de la pantalla**. Veamos algunos conceptos a tener en cuenta para diseñar las interfaces de usuario de una aplicación Android.

- ✓ **Tamaño:** longitud de la pantalla en diagonal, se mide en pulgadas. Android hace cuatro grupos (**small, normal, large y extra large**).
- ✓ **Densidad:** cantidad de píxeles en un área física de la pantalla. Se mide en puntos por pulgadas (**DPI**). Android establece cuatro grupos según la densidad (**low o LDPI, medium o MDPI, high o HDPI y extra high o XHDPI**).
- ✓ **Resolución:** cantidad de píxeles de la pantalla en horizontal y vertical. Recuerda que las aplicaciones Android no trabajan directamente con la resolución, sino que tienen en cuenta el tamaño y la densidad.
- ✓ **Píxeles independientes de la densidad (DP):** ésta es la unidad de pixel virtual que se utiliza en la definición de un diseño de interfaz de usuario para expresar las dimensiones del diseño o la posición de manera independiente a la densidad. Un pixel independiente de la densidad representa un pixel físico en una pantalla de 160 DPI (densidad media). En tiempo de ejecución, el diseño de la interfaz se adapta de forma transparente al usuario escalando al tamaño adecuado según la pantalla que tenga el dispositivo.



Google Inc.

En la siguiente tabla se muestran algunos de los tipos de pantalla más comunes teniendo en cuenta ambas características. Estas pantallas las podremos representar con el emulador y así probar mejor nuestras aplicaciones.

Tipos de pantalla más comunes según tamaño y densidad

Tipos de pantalla		DENSIDAD			
		LDPI (120)	MDPI (160)	HDPI (240)	XHDPI (320)
TAMAÑO	Pequeña	QVGA (240x320)		480x640	
	Normal	WQVGA400 (240x400) WQVGA (240x432)	HVGA (320x480)	WVGA800 (480x800) WVGA854 (480x854)	640x960 600x1024
	Grande	WVGA800 (480x800) WVGA854 (480x854)		WVGA800 (480x800) WVGA854 (480x854)	
	Extragrande	1024x600		1536x1152 1920x1152	2048x1536 2560x1536 1920x1200 2560x1600

Para saber más

Si deseas profundizar más sobre los diferentes tipos de pantallas en Android puedes consultar la referencia de Android.

[Tipos de pantallas en Android.](#)

Autoevaluación

Rellena la siguiente frase con las palabras adecuadas.

En dispositivos que tienen Android (móviles, tablets, ...) nos podemos encontrar pantallas con diferentes tamaños y píxeles. Tomemos una imagen que se muestra correctamente en un dispositivo con 160 píxeles, cuando se visualiza en un dispositivo XHDPI con píxeles el sistema escalará la imagen y probablemente se mostrará borrosa. Para solucionar este problema la medida de nuestras imágenes será el píxel de la densidad (DP) como unidad de medida.

Caso práctico



Maria y Juan ya tienen claro que programar para un dispositivo móvil no va a ser exactamente lo mismo que programar para un ordenador convencional, debido a las limitaciones y características especiales que aquellos aparatos tienen. Ahora quieren saber cómo pueden comenzar a realizar el diseño de las pantallas de sus aplicaciones.

¿Qué tipo de elementos u objetos pueden insertar?

¿Cómo lo pueden hacer?

¿Es necesario tener alguna plantilla u objeto que contengan a éstos elementos?

Para el desarrollo de cualquier interfaz de usuario, Android utiliza la clase **View** y **ViewGroup** que hereda de la clase anterior **View**. Los objetos **ViewGroup** también son conocidos como **Layout**. Un objeto **View** es aquél que dibuja algo en la pantalla con el que el usuario va a interactuar (botón, caja de texto, imagen, etc.) Un **ViewGroup** o **Layout** es un objeto que contiene los objetos de tipo **View** de nuestra pantalla. También es posible que un **Layout** contenga otro objeto **Layout**.

Para declarar el **Layout** se puede instanciar objetos de tipo **View** mediante código, pero la forma más fácil y efectiva es hacerlo con un fichero **XML** ya que ofrecen una estructura legible parecida a un **HTML**.

Aquí tienes un ejemplo de un fichero **XML** que define un objeto de tipo **Layout** que contiene a su vez dos objetos de tipo **View** (un **TextView** y un **Button**).

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/text" >
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Este es el objeto de tipo TextView" />
    <Button android:id="@+id/button" >
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Botón" />
</LinearLayout>
```

Como puedes comprobar se trata de una jerarquía de vistas donde **LinearLayout** es el nodo raíz. A la hora de definir una interfaz hay que evitar crear anidamientos, es decir, que la jerarquía de vistas tenga varios niveles. Cuando se muestra una interfaz el sistema operativo le pide al nodo raíz las siguientes operaciones:

1. Que indique su tamaño (**medición**). Su tamaño dependerá del tamaño de las vistas que contiene. Así que se recorre la jerarquía de vistas de arriba hacia abajo para conocer el tamaño de cada **ViewGroup** o **view** que contiene. Se realiza llamando al método **onMeasure()** de cada vista.
2. Que indique su posición (**diseño**). De nuevo se realiza otro recorrido preguntando a las vistas que contiene qué posición ocupan en base al tamaño que hayan establecido en el primer paso. Se realiza llamando al método **onLayout()** de cada vista.
3. Que se dibuje (**dibujo**). Finalmente se realiza un último recorrido de forma que por cada vista se crea un objeto **Canvas** que se encarga de dibujar el componente según su tamaño y posición. Se realiza llamando al método **onDraw()** de cada vista.

Cuando se concluyen las tres fases se muestra la interfaz al usuario.

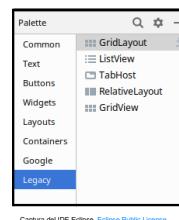
Si se anidan muchas vistas o se crea una interfaz compleja se necesitará más tiempo y recursos de cómputo para dibujar la interfaz.

Según la forma de organizar los distintos objetos **View** en la pantalla, utilizaremos un tipo de **Layout** u otro. Algunos de los tipos de **Layout** más utilizados son:

- ✓ **FrameLayout**: todas las vistas se colocan en la esquina superior izquierda y ocupan todo el espacio de la pantalla.
- ✓ **LinearLayout**: las vistas se colocan una detrás de otra de forma vertical u horizontal.
- ✓ **TableLayout**: las vistas se colocan en celdas dentro de filas y columnas.
- ✓ **ScrollView**: automáticamente se visualiza una barra de desplazamiento si los objetos **View** no pueden verse en su totalidad en la pantalla.
- ✓ **ConstraintLayout**: las vistas se colocan mediante restricciones con respecto a otras vistas o bien con respecto al padre. Por tanto, su posicionamiento es relativo.

Es posible que encuentres documentación de otros **Layout** que han quedado en desuso con la aparición de **ConstraintLayout**. En las actualizaciones de Android Studio estos **Layout** se han añadido a la sección **Legacy** del panel **Palette** en el Editor de diseño como muestra la imagen.

Todos los ficheros **XML** que definen los layouts que necesite nuestra aplicación Android se guardarán en el directorio **/res/layout**.



Captura del IDE Eclipse. [Eclipse Public License](#)

Autoevaluación

Un layout sólo puede contener objetos de tipo **View**.

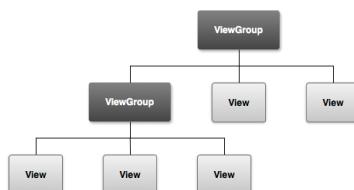
Verdadero Falso

Falso

La afirmación es falsa. Los layouts pueden contener además otros objetos de tipo layout.

Reflexiona

Se quiere dibujar una interfaz con la jerarquía de vistas que muestra la imagen. ¿Cuántas veces se ejecutaría el método **onMeasure()**?



[Google Inc.](#)

Mostrar retroalimentación

Se llamaría al menos siete veces. Mínimo el número de elementos **View** o **ViewGroup** que tiene la interfaz.

2.1.- Atributos genéricos

¿Cómo podemos definir el tamaño, la orientación, alineación y otras características de una pantalla?

Como hemos comentado, utilizaremos ficheros XML para crear los layouts y organizar nuestras pantallas. Los ficheros XML están formados por etiquetas y atributos que nos permitirán realizar pantallas de forma muy fácil.

Antes de comenzar a ver cada uno de los layouts y las vistas que pueden contener nuestras pantallas, veamos qué atributos comunes tienen estos objetos y para qué sirven.

Los atributos de View y ViewGroup permiten modificar las propiedades de cada objeto de forma personalizada. Para ello, debemos conocer qué atributos podemos modificar según el tipo de objeto al que haga referencia el atributo. Estos atributos se definen en la clase LayoutParams de cada ViewGroup. Éstos siguen el formato android:nombre_atributo. Algunos ejemplos serían android:layout_width, android:layout_height, android:paddingLeft, etc...

Vamos a ver algunos de los atributos más comunes que pueden aparecer en los objetos de tipo GroupView o layouts. Debemos recordar que la clase GroupView hereda de la clase View, por tanto estos atributos también estarán presentes cuando veamos los objetos View o vistas.

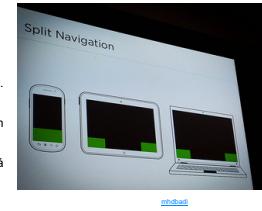
- ✓ **ID:** cualquier objeto de tipo View debe tener un identificador asociado que lo identifique de forma única. En los ficheros XML a este atributo se le asigna un tipo de dato String pero cuando la aplicación sea compilada, el ID será referenciado como un integer. La sintaxis para este atributo será:

```
android:id="@+id/texto1"
```

- ✓ **Height y Width:** se utilizan para definir la altura y la anchura del objeto tanto de un layout como en una vista o View. Los valores podemos asignarlos directamente usando las dimensiones que queremos definir aunque no suele ser la práctica habitual. Es más recomendable utilizar los valores WRAP_CONTENT y MATCH_PARENT. Desde la versión API 8 el valor MATCH_PARENT ha sustituido al valor FILL_PARENT aunque éste puede seguir usándose sin problemas. Con el valor WRAP_CONTENT, el objeto sólo cogerá el espacio necesario para representarse con respecto al objeto contendido o padre y con el valor MATCH_PARENT cogerá el espacio entero del padre.

- ✓ **Padding y Margins:** ambos se utilizan para crear espacio alrededor de los objetos. Esto le da una mejor apariencia a nuestra interfaz de usuario. Con el atributo margin establecemos un espacio fuera del objeto con respecto a los otros objetos de alrededor, mientras que el atributo padding establece un espacio dentro del objeto, por ejemplo en el caso de tener un Button, la distancia desde el texto del botón a las líneas del mismo.

- ✓ **Gravity:** este atributo se utiliza para alinear los objetos. Por defecto se alinea a la izquierda, si deseamos poner alguna otra alineación debemos utilizarlo.



Autoevaluación

El valor MATCH_PARENT y WRAP_CONTENT realizan la misma acción en los atributos Height y width.

Verdadero Falso

Falso

La afirmación es falsa. El valor MATCH_PARENT indica al objeto que abarque todo el espacio del padre mientras que el valor WRAP_CONTENT indica al objeto que coja el espacio necesario para mostrarse.

2.2.- Acceder a las vistas en código

Llegados a este punto, ¿crees que podemos mejorarnuestras aplicaciones y hacerlas más accesibles y atractivas a los usuarios con independencia del dispositivo que utilicen? Seguro que sí. Veámoslo.

Para finalizar este apartado es interesante saber cómo una aplicación puede acceder a estos recursos. Es cierto que lo hace de forma transparente al usuario, pero como desarrolladores debemos saber cómo se referencia a estos ficheros.

Como ya se ha comentado, en la programación Android podemos utilizar el lenguaje Java o bien XML y de momento para todo el diseño de la interfaz de usuario estamos utilizando el segundo por ser más aconsejable en esta tarea. No obstante no debemos dejar de lado que la lógica de la aplicación se implementará mediante código Java y por tanto también será necesario acceder a los diferentes objetos y recursos creados mediante XML.

Cuando se genera un proyecto con Android Studio automáticamente se crea el fichero R.java donde se identifican todos los objetos y recursos creados. **Este fichero nunca debe ser modificado manualmente**. Para acceder a los recursos lo haremos de dos formas:

- ✓ **Mediante código Java:** será necesario seguir la siguiente sintaxis:

```
[<nombre_paquete>.R.<tipo_recurso>.<nombre_recurso>]
```

Donde el nombre del paquete no es necesario si nos referimos a recursos que están dentro del mismo. Ejemplos:

```
setContentView(R.layout.activity_main);
ImageView imageView = (ImageView) findViewById(R.id.myimageview);
imageView.setImageResource(R.drawable.myimage);
```



- ✓ **Mediante XML:** será necesario seguir la siguiente sintaxis:

```
@[<nombre_paquete>]<tipo_recurso><nombre_recurso>
```

Donde el nombre del paquete no es necesario si nos referimos a recursos que están dentro del mismo. Ejemplos:

```
android:paddingBottom="@dimen/activity_vertical_margin"
android:text="@string/hello_world"
```

Debes conocer

A continuación os mostramos dos vídeos que explican de forma detallada cómo crear una interfaz de usuario en Android Studio. Fíjate que puedes hacerlo directamente de forma gráfica o bien por medio de código XML.

Curso Android. Crear interfaz de usuario I.

<http://www.youtube.com/embed/gjwVzPddKKU>



Curso Android. Crear interfaz de usuario II.

<http://www.youtube.com/embed/nG2qrzmxUh8>



Autoevaluación

Si queremos acceder al fichero R.java desde un fichero XML utilizamos el método `findViewById`.

- Verdadero Falso

Falso

La afirmación es falsa porque este método se utiliza si queremos acceder al fichero desde código Java.

2.3.- FrameLayout

Este tipo de layout organiza todos los objetos de tipo View en la esquina superior izquierda uno encima de otro. En este tipo de contenedor se hace necesario utilizar el atributo android:visibility de los objetos View que lo componen para definir la transparencia de aquellos elementos que no queramos mostrar en un momento determinado de la aplicación.

Aquí vemos un ejemplo donde se insertan tres objetos View como son un TextView, un Button y un ProgressBar o barra de progreso y sólo dejamos visible el botón. Durante la ejecución se podrá cambiar el estado del atributo android:visibility de los demás objetos View para que se muestren por pantalla.

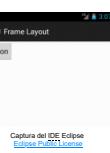
```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/Texto"
        android:visibility="invisible"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/Boton" />

    <ProgressBar
        android:id="@+id/progressBar1"
        style="?android:attr/progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="invisible" />

</FrameLayout>
```



2.4.- LinearLayout

Este tipo de layout dispone los elementos View o vistas de forma consecutiva en la pantalla. Puede utilizarse de forma vertical u horizontal mediante el atributo android:orientation. Las etiquetas que se utilizan para abrir y cerrar el layout son <LinearLayout> y </LinearLayout>

En el siguiente ejemplo puedes ver algunos de los atributos más utilizados en este tipo de layout. Recuerda que los valores de textos están introducidos mediante el fichero strings.xml de la carpeta /res/values/, aquí tan sólo se referencia al contenido de ese tipo de recurso.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/para" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/mensaje" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/cambia" />
</LinearLayout>
```



Este ejemplo de LinearLayout tiene asignada la anchura y altura del objeto padre, en este caso, la pantalla. Se define un espacio interior izquierdo y derecho de 16dp y la orientación es vertical.

En este layout se organizan tres objetos View o vistas de tipo EditText y un objeto Button. Fíjate que el botón se alinea a la derecha con el atributo de la vista: layout_gravity. Por ahora nos centraremos en los atributos del layout, ya que cada objeto View tiene su propia lista de atributos que veremos más adelante.

En este tipo de layout es interesante ver cómo algunas vistas tienen más espacio reservado que otras. Por defecto, si no se indica lo contrario, el sistema reparte todo el espacio del layout entre las vistas que lo contienen. Si nos fijamos en el tercer objeto de tipo EditText, comprobaremos que tiene más espacio reservado que los dos anteriores. Esto se consigue con el atributo android:layout_weight que según el número que le asignamos tendrá más o menos peso a la hora de repartir el espacio disponible. Si no indicamos nada cogerá el espacio necesario, si indicamos algún valor 1, 2, etc., repartirá el espacio total del layout dejando más espacio para las vistas que tengan el valor más alto y menos para el valor más bajo.

Para que el peso se ajuste correctamente y se repartan el espacio sobrante de la pantalla, es importante que el atributo android:layout_width sea igual a 0dp en un diseño horizontal. Si se trata de un diseño vertical el atributo android:layout_height debe ser igual a 0dp.

Debes conocer

Es muy recomendable que revises todos los atributos específicos de este tipo de layout, puedes consultar la guía de referencia Android donde encontrarás la lista completa de parámetros que podrás utilizar.

[Parámetros del LinearLayout](#)

Ejercicio Resuelto

Crea una aplicación que contenga tres botones que estén alineados horizontalmente y cuyo texto sea el siguiente:

1. Botón 1
2. Soy el botón 2
3. Soy el último botón

Comprueba cómo el sistema operativo reparte el ancho de la ventana en base a la longitud del texto de los diferentes botones.

A continuación, asigna el siguiente peso a los botones:

1. 50%
2. 30%
3. 20%

Comprueba el comportamiento de la interfaz cuando se asigna el peso a cada botón.

[Mostrar retroalimentación](#)

```
<?xml version="1.0" encoding="utf-8"?><br />
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"><br />
    android:layout_width="match_parent"<br />
    android:layout_height="match_parent"<br />
    android:orientation="horizontal"<br />
    android:gravity="center_vertical"<br />
    android:layout_margin="0dip"><br />
    <Button><br />
        android:layout_height="wrap_content"<br />
        android:width="100dip"<br />
        style="@+id/button_bar_buttonStyle"<br />
        android:layout_weight="50"<br />
        android:gravity="end"<br />
        android:text="@string/fifty_weight"/>
    <Button><br />
        android:layout_height="wrap_content"<br />
        android:width="0dip"<br />
        style="@+id/button_bar_buttonStyle"<br />
        android:layout_weight="30"<br />
        android:gravity="end"<br />
        android:text="@string/thirty_weight"/>
    <Button><br />
        android:width="0dip"<br />
        style="@+id/button_bar_buttonStyle"<br />
        android:height="wrap_content"<br />
        android:text="@string/twenty_weight"<br />
        android:layout_weight="20"<br />
        android:gravity="end"/>
</LinearLayout>
```

2.5.- TableLayout

Este tipo de layout dispone las vistas en forma de tabla, es decir mediante filas y columnas se organizan las vistas que queremos incorporar en el layout. Las etiquetas que emplearemos para abrir y cerrar el layout son <TableLayout> y </TableLayout>. Además para definir cada una de las filas de nuestro layout utilizaremos las etiquetas <TableRow> y </TableRow> dentro de las cuales se insertaran los elementos que deseemos considerando cada uno de ellos como una columna de la tabla.

Veamos un ejemplo de este tipo de layout tanto en la definición del fichero XML como el aspecto que obtendríamos en el dispositivo.

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp">
    <TableRow>
        <EditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="@string/para" />
        <EditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="@string/asunto" />
    </TableRow>
    <TableRow>
        <EditText
            android:layout_width="match_parent"
            android:layout_height="200dp"
            android:gravity="top"
            android:hint="@string/mensaje" />
        <Button
            android:layout_width="100dp"
            android:layout_height="wrap_content"
            android:layout_gravity="right"
            android:text="@string/enviar" />
    </TableRow>
</TableLayout>
```



En este ejemplo se ha utilizado este tipo de layout para organizar las cuatro vistas en forma de tabla. En la primera fila se sitúan dos objetos View de tipo EditText, que formarán a su vez dos columnas, una por cada objeto. En la segunda fila se sitúa otro EditText al cual se le ha dado una altura de 200dp y en la siguiente columna el objeto Button. En cuanto a los textos de cada objeto tipo View al igual que el caso anterior, se hace referencia al fichero strings.xml donde deben estar definidos.

Hay que tener en cuenta que el ancho de cada columna viene definido por el máximo ancho de los elementos que hay en cada columna. Esto podemos modificarlo utilizando los atributos (como android:stretchColumn, android:shrinkColumns y android:collapseColumns) y recuerda que en la documentación oficial de Android puedes encontrar todos los atributos específicos de este layout.

Autoevaluación

El atributo utilizado para indicar al objeto que tome más espacio que el resto de objetos contenidos en el mismo layout es...

- android:strecthColumn
- android:layout_height
- android:layout_weight
- android:layout_gravity

No has acertado, es incorrecto.

Fallaste, no es correcto.

¡Muy bien! Es la única opción correcta. Indica al objeto el peso que tiene en el layout con respecto al resto de objetos.

No es correcto.

Solución

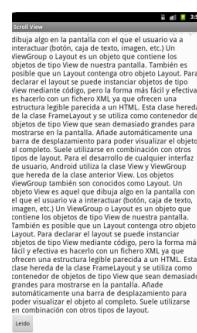
1. Incorrecto
2. Incorrecto
3. Opción correcta
4. Incorrecto

2.6.- ScrollView

Esta clase hereda de la clase FrameLayout y se utiliza como contenedor de objetos de tipo View que sean demasiado grandes para mostrarse en la pantalla. Añade automáticamente una barra de desplazamiento para poder visualizar el objeto al completo. Suele utilizarse en combinación con otros tipos de layout.

Vamos a comprobar su funcionamiento con un ejemplo en el que se ha utilizado dentro del ScrollView un LinearLayout para organizar el objeto TextView que contiene todo el texto que sobrepasa verticalmente la pantalla (el texto se guarda en el fichero de recursos strings.xml) y un objeto Button que deseamos visualizar cuando el usuario haya llegado hasta el final del texto. La utilización del atributo android:fillViewport define si queremos que el objeto ScrollView ocupe la pantalla completa incluso si el contenido del texto no sobrepasa los límites de la pantalla verticalmente.

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/scrollView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:fillViewport="true"
    android:paddingRight="5dp">
    <LinearLayout
        android:id="@+id/linearLayout1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <TextView
            android:id="@+id/textView1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textAppearance=
                "@android:attr/textAppearanceMedium"
            android:text="@string/contenido"
            android:layout_weight="2"/>
        <Button
            android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Leido"/>
    </LinearLayout>
</ScrollView>
```



Para saber más

Puedes aprender algo más sobre este tipo de contenedor y su utilización visitando esta página:

[ScrollView y HorizontalScrollView.](#)

Autoevaluación

ScrollView suele utilizarse en combinación con otros tipos de layout.

Verdadero Falso

Verdadero

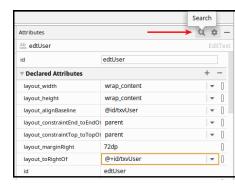
Es cierto, este tipo de contenedor utiliza otros layouts para organizar su contenido.

2.7.- ConstraintLayout

Este **Layout** permite construir UI complejas sin crear anidamientos ya que todas las vistas se encuentran en un mismo nivel. Antes de este **Layout** las interfaces complejas se construían anidando diferentes ViewGroup. Un ejemplo típico era usar LinearLayout dentro de un RelativeLayout para intentar centrar los botones horizontalmente. Este **Layout** ofrece atributos que permiten crear interfaces de forma más sencilla que RelativeLayout al que ha sustituido, de forma que se relacionan las vistas mediante restricciones o *constraints*.

Este **Layout** muestra la información de forma más visual y permite construir la interfaz de una forma sencilla utilizando el **Editor de diseño** de Android Studio. Ya no se tiene que editar el código XML sino que se realiza de forma gráfica a través de la vista **Design**. Para añadir un botón al diseño se realizarán los siguientes pasos:

- 1.- Se arrastra el widget Button desde el panel **Palette** hasta la vista de diseño.
- 2.- Todas las vistas del diseño aparecen en el panel **Component Tree** que muestra la jerarquía de vistas en nuestro diseño.
- 3.- Se selecciona el componente en el diseño y a continuación en el panel **Attributes** aparecen los atributos que se han definido pudiendo modificarlos o bien definir un atributo nuevo. Es recomendable utilizar el buscador que hay en la parte superior como muestra la imagen.



Captura del IDE Eclipse. Eclipse Public License

En este componente es importante el icono Design y Blueprint de la barra de herramientas de Android Studio. Recuerda que en este desplegable puedes escoger cómo quieres ver tu diseño en el editor:

- ✓ La vista **Design** muestra una vista real de la interfaz.
- ✓ La vista **Blueprint** marca el contorno de cada vista y en el caso de ConstraintLayout las relaciones que se han definido en una vista respecto a otra, propiedades como el margen, si se ha definido una guía,... Esta vista es importante en este **Layout** porque nos ofrece una "radiografía" de nuestro diseño.
- ✓ Finalmente puedes optar por tener ambas vistas (**Design + Blueprint**) una al lado de la otra.

A continuación veremos qué hacen las distintas acciones de la barra de herramientas de este **Layout**, de izquierda a derecha:



Captura del IDE Eclipse. Eclipse Public License

1. **View options:** esta opción indica qué restricciones se quieren ver y en qué vista **Design** o **Blueprint**. A veces interesa ver todas las restricciones en las dos vistas seleccionando la opción **Show All Constraints** y los márgenes con la opción **Show Margins**. Pero a veces esta información puede no ser necesaria en la vista de diseño y seleccionar la opción **Hide Unselected Views**.
2. **Autoconnect:** el icono del imán en la barra de herramientas activa la opción **Autoconnect**. Si esta opción está activa y desplazas una vista hacia los lados de la ventana principal creará automáticamente la restricción con el padre en base a la posición de la vista y su proximidad a ambos lados del padre. Por defecto esta opción está deshabilitada y sólo crea restricciones con el padre, no con otras vistas.
3. **Default Margin:** esta opción permite crear un margen al valor indicado según se añade las restricciones a una vista. Un ejemplo de su utilidad es cuando se quiere que todas las vistas de la izquierda tengan un margen de 16dp con respecto al padre. Se inicializa el margen por defecto a 16dp y cuando se conecta la vista con el padre se crea automáticamente el margen de 16dp.
4. **Clear All Constraints:** si el diseño actual no te convence o bien simplemente quieres empezar de nuevo haz clic en este icono y se eliminarán TODAS las restricciones.
5. **Infer Constraint:** cuando pulas el icono de la varita mágica Android Studio agregará todas las restricciones que faltan en tu diseño en base a una serie de algoritmos y probabilidades. Puede ser una fantástica idea que Android Studio implemente automáticamente las conexiones, pero puedes pasar que no se parezca en nada a lo que tú querías hacer. Una opción intermedia es añadir una vista, pulsar este icono y modificar posteriormente el diseño en vez de esperar que Android Studio acierte con tu diseño.
6. **Pack:** esta opción se activa cuando se seleccionan varias vistas mediante la tecla **Ctrl**. Permite realizar una serie de acciones (empaquetar, expandir o distribuir) de forma conjunta sobre las vistas seleccionadas.
7. **Align:** esta opción se activa cuando se seleccionan varias vistas mediante la tecla **Ctrl**. En este menú se encuentran todas las posibles alineaciones de componentes. Por ejemplo, podemos seleccionar varias imágenes y en la barra de herramientas escoger la opción **Align > Verticals Center**. De esta forma se agrega una nueva restricción que hace que las imágenes se alineen en la parte inferior y la parte superior con la imagen vecina.
8. **Tools:** en esta opción se encuentra una serie de herramientas que son el potencial de este **Layout**: líneas guía, las barreras, los grupos y las capas.

Debes conocer

Realiza el siguiente tutorial que te permitirá afianzar los contenidos que se han visto sobre la barra de herramientas de ConstraintLayout.

[Comenzar con ConstraintLayout](#)

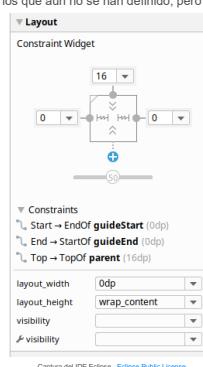
Autoevaluación

Una de las ventajas de ConstraintLayout es que en el **Editor** aparecen nuevas herramientas que permiten establecer las restricciones en el editor sin tener que añadir código XML.

Verdadero Falso

Verdadero

La afirmación es correcta. Al igual que otros layout aparecen en el panel **Attributes** los atributos declarados y los que aún no se han definido, pero no tienen un widget específico dentro del panel como muestra la siguiente imagen.



Captura del IDE Eclipse. Eclipse Public License

2.7.1.- Posición y tamaño de las vistas

Para posicionar una vista en ConstraintLayout se debe añadir una restricción (*constraint*) con respecto a otra vista o bien con respecto al nodo raíz y debe cumplir las siguientes reglas:

1. Cada vista debe tener al menos dos restricciones: una para el **eje horizontal** (comienzo y final) y otra restricción en el **eje vertical** (arriba, abajo y línea base).
2. Al crear una restricción se debe realizar entre puntos de anclaje que comparten el mismo plano, es decir, un punto de anclaje al comienzo de una vista sólo se puede unir a otro anclaje que esté al comienzo o final de otra vista.
3. Cada punto de anclaje sólo puede utilizarse para crear una sola restricción, pero puede ser el destino de varias restricciones siempre y cuando sean de vistas diferentes.



A continuación vamos a analizar los elementos que aparecen en la imagen:

1. **Guías cuadradas:** aparece en las esquinas de `editUser` y se usan para ajustar el tamaño de la vista en `dp`.
2. **Guía base:** es el rectángulo con orillas redondas que se encuentra dentro de `editUser` y se usa para alinear el contenido de la vista con la línea base de `txvUser`. Mediante este atributo se establece que `editUser` se posiciona a la misma altura que `txvUser` en el eje y:

```
app:layout_constraintBaseline_toBaselineOf="@+id/txvUser">br />
```

3. **Restricciones:** son los puntos de anclaje y se representan como círculos. Cuando hay una restricción entre dos vistas puede ser:

✓ Una **línea en zig-zag**. Este caso ocurre cuando hay dos restricciones que actúan como fuerzas opuestas que separan el componente y hay espacio sobrante a ambos lados del componente porque bien el ancho o el alto tiene asignado el valor `wrap_content`. Es lo que sucede con el componente `editUser`. A la izquierda hay una restricción que indica que la vista comienza donde finaliza `txvUser` y a la derecha hay otra restricción que indica que finaliza en el padre o parent. Si este componente no ocupa todo el ancho de la ventana, ¿dónde se coloca finalmente la vista? Lo que hace el sistema es centrar la vista en el espacio disponible.

```
app:layout_constraintStart_toEndOf="@+id/txvUser"
app:layout_constraintEnd_toEndOf="parent"
```

- ✓ Una **línea continua**. En el caso de `txvUser` no hay dos restricciones opuestas, sino que se sitúa al inicio del parent y en la parte de arriba del parent.

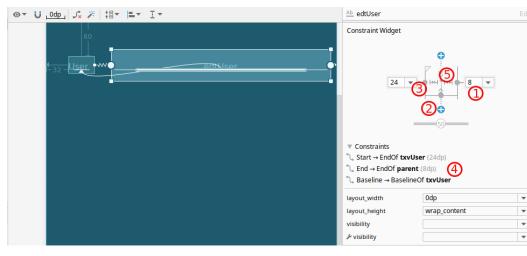
```
app:layout_constraintStart_toStartOf="parent">br />
app:layout_constraintTop_toTopOf="parent"
```

- ✓ **Márgenes.** Aparecen señalados mediante líneas continuas junto con el valor del margen en `dp`. La vista `bxvUser` tiene un margen superior de `60dp` y un margen inicial de `30dp`.

```
android:layout_marginStart="32dp">br />
android:layout_marginTop="64dp"
```

Si seleccionamos el componente `editUser` aparece el **widget Attributes** y se pueden realizar las siguientes operaciones:

1. **Modificar los márgenes** que se han aplicado. Mediante el desplegable se puede seleccionar valores por defecto o bien escribir directamente el valor del margen.
2. **Crear conexiones** con respecto al parent.
3. **Eliminar una restricción** ya creada pulsando sobre el círculo o punto de anclaje.
4. **Seleccionar las restricciones** que se han creado en la pestana **Design del Editor**.
5. **Modificar el tamaño de los componentes.** Estos símbolos representan cómo se calcula el tamaño de la vista. Si haces clic en el símbolo cambiarás entre los diferentes tipos de configuración:
 - Fixed: se ha especificado un tamaño fijo en el cuadro de texto en `dp`.
 - Wrap Content: el tamaño de la vista será el tamaño necesario para poder mostrar el contenido.
 - Match Constraints: la vista se expande tanto como sea posible para cumplir con las restricciones de cada lado después de aplicar los márgenes de la vista. Para que se cumpla esta restricción se debe asignar el valor `0dp` a `layout_width` si se quiere que se expanda horizontalmente o a `layout_height` para que se expanda verticalmente.



Captura del IDE Eclipse. [Eclipse Public License](#)

Ejercicio Resuelto

Puedes descargar el proyecto que se ha utilizado en este apartado en el caso que quieras resolver alguna duda en el siguiente enlace.

[Ejemplo_Constraint_layout_login \(zip - 0.46 MB\)](#)

Debes conocer

Consulta la sección **Cómo crear una IU receptiva con ConstraintLayout** de la documentación oficial, donde aprenderás las siguientes operaciones:

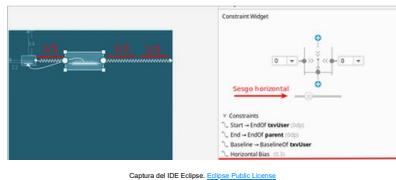
1. Crear una restricción.
2. Crear dos restricciones opuestas.
3. Eliminar una restricción.
4. Aplicar un margen a una vista.
5. Alinear una vista con otra.
6. Establecer el tamaño como una relación

[Cómo crear una interfaz de usuario con ConstraintLayout.](#)

2.7.2.- Sesgo, guías, barreras y cadenas

Seguimos con el ejercicio planteado ConstraintLayoutLogin. Como ya se comentó el componente `editUser` no ocupa todo el ancho de la pantalla, con lo cual se centra en el espacio disponible si tiene el atributo `android:layout_width="wrap_content"`. Este **Layout** ofrece un parámetro `_bias` que permite desplazar una vista en términos de porcentajes (0% a 100%) o fracciones (0 a 1) en el eje horizontal o vertical con respecto a los anclajes que tiene el componente, es lo que se conoce como **sesgo**.

Si se mueve el sesgo horizontal en `editUser` al valor 30%, lo que hace es que la vista se desplaza hacia la izquierda dejando 1/3 del espacio disponible a la izquierda y 2/3 a la derecha. Este comportamiento se utiliza para garantizar que una vista tenga la misma posición independientemente de la densidad y tamaño de la pantalla.



La **guía** es una herramienta que se utiliza en la vista de diseño para poder alinear las vistas en base a un elemento pero que no se verá en el diseño de nuestra aplicación. Esta guía evita añadir en todos las vistas un margen concreto sino que se añade la restricción que la vista empieza al principio de la guía:

 app:layout_constraintStart_toStartOf="@+id/guideline"

En la parte superior del diseño aparece el icono de la guía y se puede alternar entre los tipos haciendo clic repetidamente en el ícono. Por defecto se indica un desplazamiento fijo en `dp` desde el borde inicial del layout `ConstraintLayout`, a continuación el desplazamiento fijo contando desde el final de la pantalla y finalmente el porcentaje del desplazamiento en base al ancho de la pantalla.

La **barrera** es una vista "invisible", pero que su efecto se muestra en tiempo de ejecución. Al igual que pasa con la guía, las vistas se limitan a la barrera, de forma que si una vista crece (porque el texto a mostrar aumenta ya que hay un cambio de idioma en el dispositivo), la barrera ajustará su tamaño a la altura o anchura más grande de los elementos referenciados sin que se solape ninguna vista. Las barreras pueden ser verticales u horizontales y pueden crearse en la parte superior, inferior, izquierda o derecha de las vistas referenciadas.

Es curioso ver el código de las barreras en XML. En primer lugar se debe establecer la dirección de la barrera. En el ejemplo `ConstraintLayoutLogin` la barrera se posiciona al final de `txvUser` o `txvPassword` dependiendo de cuál sea el más grande. A continuación se necesita un atributo para definir los ID de las múltiples vistas a las que hace referencia la barrera.

```
<android.support.constraint.Barrier><br />
    android:id="@+id/barrier"<br />
    android.layout_width="wrap_content"<br />
    android.layout_height="wrap_content"<br />
    app:constraint_referenced_ids="editUser,editPassword"<br />
```

En el siguiente video se muestra cómo crear una barrera:

Crear una restricción "barrera" en el layout
ConstraintLayout

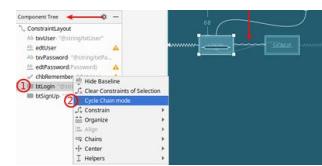
<http://www.youtube.com/embed/ZZ31wlny3C4>

PlayLogos_Banner_ConstraintLayout_Betabeers
Barrera en layout alternativo

Finalmente, las **cadenas** son un tipo específico de restricción que permite compartir espacio entre las vistas que pertenecen a la cadena y controlar cómo se divide el espacio disponible entre ellas. Esta restricción se utiliza mucho para centrar dos botones con respecto al eje horizontal. Para crear una cadena se siguen los siguientes pasos:

1. Añadir dos botones `btnLogin` y `btnSignUp`.
2. Seleccionar los botones pulsando la tecla `Ctrl` -> opción **Chains** del menú contextual -> **Create Horizontal Chain**.

Se puede comprobar como una cadena une las dos vistas.



Si se quiere explorar algunos de los modos de cadena, hay que hacer clic en una de las vistas de la cadena en el panel **Component Tree**, mostrar el menú contextual y seleccionar la opción **Cicle Chain Mode**. Se puede simplificar en tres modos:

1. **Empaquetado (packed)**: los elementos se agrupan en el centro del espacio disponible. En esta opción se puede utilizar márgenes para separar las vistas ligeramente. Con esta opción y utilizando las `_bias` se pueden centrar todas las vistas aplicando un sesgo de 0.5 o bien desplazar la cadena hacia un lado.
2. **Distribución (spread)**: los elementos se extienden por el espacio disponible, como se muestra arriba.
3. **Separación en el interior (spread_inside)**: similar a la opción anterior, pero las vistas finales de la cadena no se extienden.

No existe un atributo específico en XML que se asigne a todas las vistas sino que se establece el atributo `app:layout_constraintHorizontal_chainStyle="spread"` en una de las vistas de la cadena.

Debes conocer

En el caso que tengas dudas te mostramos a continuación dos vídeos en inglés que explican de forma detallada cómo crear los contenidos estudiados en este apartado.

ConstraintLayout Tutorial Part 3. -
GONE MARGIN, CHAINS and GUIDELINES

http://www.youtube.com/embed/_NBjbOI2Rkw

ConstraintLayout Tutorial Part 3 ...



Código en Fase Resumen textual alternativo

ConstraintLayout Tutorial Part 4. -
BARRIERS AND GROUPS

<http://www.youtube.com/embed/NgZ5cgLC7m4>

ConstraintLayout Tutorial Part 4 ...



Código en Fase Resumen textual alternativo

Caso práctico



El diseño de la pantalla está terminado, pero de nada sirve tener una pantalla si al pulsar en un botón, éste no hace nada.

Ada se reúne con el equipo de desarrolladores para explicar qué funcionalidad se requiere conseguir mediante la gestión de eventos del usuario y además, en este caso, será mucho mejor utilizar la programación en código Java para su implementación.

Como has podido comprobar hasta ahora el diseño de la interfaz de usuario se ha realizado mediante XML, ahora comenzaremos a trabajar con Java para proporcionar los mecanismos apropiados para esa interacción con el usuario. Comencemos por ver algunos conceptos necesarios:

- ✓ **Eventos:** son las acciones que los usuarios realizan sobre el dispositivo móvil o sobre algún objeto de la pantalla. (Ej.: pulsar un botón, arrastrar un elemento por la pantalla, etc.)
- ✓ **Escuchadores (Listeners):** son los componentes que gestionan los eventos. Normalmente existe un listener por evento. Se representan por clases abstractas donde el desarrollador debe implementar el método que recibirá el evento.
- ✓ **Métodos que registran los escuchadores:** son los métodos que permiten registrar los listeners para que puedan ser utilizados. Puedes encontrarlos en la clase android.view.View. (Ej.: el método setOnClickListener() registra el listener OnClickListener que capturará los eventos cuando se pulsa sobre un elemento de la pantalla).

Para registrar los escuchadores podemos utilizar varias técnicas según la documentación oficial, pero vamos a centrarnos en las siguientes que seguiremos con un ejemplo muy sencillo.

- ✓ **Implementación anónima:** creamos un objeto anónimo del escuchador con la implementación del evento dentro y su registro se hace desde la clase de la actividad.

```
// Registraremos el escuchador
Button boton = (Button) findViewById(R.id.button1);
boton.setOnClickListener(miEscuchador);

//Implementamos el evento dentro del Listener
private OnClickListener miEscuchador = new OnClickListener() {
    public void onClick(View v) {
        TextView texto = (TextView) findViewById(R.id.textView1);
        texto.setText(R.string.despedida);
    }
};
```



Tim Ellis

- ✓ **Implementación en la misma clase:** en este caso la clase de la actividad implementaría la interfaz del escuchador y lo registramos desde aquí. Esta opción es la más recomendada, puesto que tiene menos gasto de memoria.

```
public class MainActivity extends Activity implements OnClickListener
...
// Registraremos el escuchador
Button boton = (Button) findViewById(R.id.button1);
boton.setOnClickListener(this);
}
//Implementamos el evento
public void onClick(View v) {
    TextView texto = (TextView) findViewById(R.id.textView1);
    texto.setText(R.string.despedida);
}
```

Recomendación

En los siguientes enlaces puedes descargar los proyectos de ejemplo de cada caso (implementación anónima e implementación en la misma clase). Puedes estudiar su código con lo que se ha explicado anteriormente.

[Ejemplo de implementación anónima](#) (zip - 26.57 MB).

[Ejemplo de implementación en la misma clase.](#) (zip - 26.48 MB)

Debes conocer

Existen muchos tipos de eventos, algunos de ellos comunes a varios tipos de objetos, otros más específicos de cada uno. Es fundamental consultar la página oficial de Android.

[Tipos de eventos en Android](#)

Autoevaluación

Un escuchador o listener es una acción que realiza el usuario en su dispositivo móvil.

Verdadero Falso

Falso

La afirmación es falsa porque la definición se corresponde a un evento. El listener es el componente que gestiona esa acción o evento.

Caso práctico



Juan ya ha comprobado las diferentes formas de organizar los elementos que contienen cada una de las pantallas que puede diseñar. Ha insertado algunos objetos de tipo View en algunos Layouts como son el TextView, Button o EditText, pero quiere saber cuáles son todos los objetos View que puede utilizar para desarrollar sus aplicaciones Android.

¿Qué objetos de tipo View se pueden utilizar en las aplicaciones Android?

Para diseñar una pantalla o interfaz de usuario es fundamental incorporar elementos para que el usuario pueda interactuar con la aplicación que creamos. Estos objetos, también llamados controles, vienen definidos en la clase View y los atributos de los que dispone para personalizarlos, dependerá de cada uno de ellos, aunque como vimos anteriormente hay muchos atributos comunes a todos.

Si estamos diseñando la interfaz de usuario mediante un fichero XML, es fácil e intuitivo organizar en el layout elegido como contenedor los distintos botones, etiquetas, cuadros de texto y otros controles que necesitemos. Como ya vimos al comienzo del apartado anterior, Android Studio incorpora una forma muy sencilla de insertar los objetos View sobre un Layout, utilizando la interfaz gráfica (arrastrar y soltar los objetos View desde la paleta al lienzo) o bien mediante código XML escribiendo las etiquetas y atributos apropiados de cada objeto.

Podemos organizar los distintos objetos View en tres grupos:

- ✓ **Controles básicos:** son controles que aparecen en la mayoría de las pantallas de una aplicación debido a su gran utilización. Aquí encontramos los típicos botones, imágenes, etiquetas, cuadros de texto, etc.
- ✓ **Controles de selección:** permiten seleccionar una opción de un listado de opciones. Aquí encontramos por ejemplo listas fijas o desplegables y tablas.
- ✓ **Controles personalizados:** podemos crear nuestros propios controles para ofrecer un diseño más apropiado a nuestras aplicaciones de los que nos ofrecen los controles convencionales.



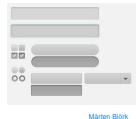
Macmillan

4.1.- Controles básicos

Llegados a este punto, nos planteamos algunas cuestiones, como las siguientes:

- ✓ ¿Cómo puedes utilizar las aplicaciones instaladas en tu dispositivo móvil?
- ✓ ¿Cómo rellenes un formulario con tus datos personales?
- ✓ ¿Utilizas algún botón para confirmar alguna acción como "enviar"?

Pues bien, todas estas acciones las podemos llevar a cabo porque las diferentes pantallas por las que navegamos en nuestra aplicación disponen de elementos o controles predeterminados que nos permiten interactuar con el dispositivo. Veremos a continuación cuáles son.



TextView
EditText
Button
Switch
CheckBox
RadioButton

Debes conocer

Puedes realizar el siguiente ejemplo que te explica cómo crear controles del tipo CheckBox y RadioButton y manejar estos eventos de estado para recoger la información que el usuario ha introducido:

[Hermosa Programación](#)

ImageButton
ImageView
FloatingActionButton

Hasta ahora hemos visto los diferentes tipos de controles más utilizados, pero si te fijas en la paleta que incorpora Android Studio puedes insertar en tus aplicaciones muchos controles más como por ejemplo: TextClock, CalendarView, ProgressBar, RatingBar, etc. Puedes probarlos en tus aplicaciones y consultar toda la información de cada una de las clases en la documentación oficial de Android.

Recomendación

En el siguiente vídeo te mostramos cómo crear una interfaz con controles básicos y manejar los eventos que ocurren en ellos.

[Crear Controles básicos y manejar sus eventos](#)

<http://www.youtube.com/embed/RTVqU2z1ipg>

[Ver en YouTube](#)

Para saber más

En ocasiones necesitaremos dotar de alguna funcionalidad extra o dar una imagen más personal y original a nuestra aplicación. Por tanto, Android nos permite crear nuestros propios controles personalizados de varias formas. En este enlace puedes ver un ejemplo de cada una de ellas.

[Cómo crear un control personalizado.](#)

Autoevaluación

Si queremos incluir un control u objeto TextView para introducir una contraseña utilizamos en el atributo android:inputType el valor:

- "text"
- "textPassword"
- "number"
- "phone"

No es correcto.

¡Has acertado! Se trata de la opción correcta, ya que se introducen los caracteres en forma de puntos.

Incorrecto.

No es la opción correcta.

Solución

1. Incorrecto
2. Opción correcta
3. Incorrecto
4. Incorrecto

Autoevaluación

Indica qué controles heredan de la clase Button.

- CheckBox
-
- RadioButton
- ToggleButton

ImageView

Mostrar retroalimentación

Solución

- 1. Correcto
- 2. Correcto
- 3. Correcto
- 4. Incorrecto

4.2.- Controles de selección

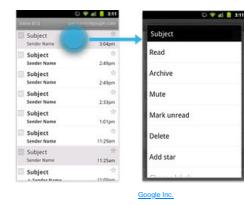
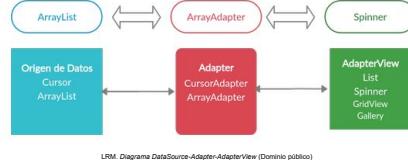
¿Podemos utilizar listas de opciones donde poder seleccionar aquella que nos interese?

Efectivamente, Android incorpora controles para la selección de opciones. Su implementación no es tan sencilla como los controles básicos que hemos visto, ya que para utilizar este tipo de controles necesitamos apoyarnos en un elemento imprescindible como son los adaptadores. Estos controles heredan de la clase AdapterView. Veremos en profundidad el Spinner (lista desplegable) y RecyclerView ya que es una versión más avanzada y flexible de ListView. Aunque hay que mencionar el control GridView y Gallery como otras posibles alternativas.

Un adaptador es un objeto que tiene dos funciones:

1. Proporcionar los datos necesarios a la vista. Estos datos pueden ser de cualquier tipo Arrays, objetos List y objetos Cursor. Android dispone de varios tipos de adaptadores en función de los datos que gestiona, pero aquí nos centraremos en el más sencillo llamado ArrayAdapter.
2. Crear la vista o objeto View con los datos que se representarán en el control de selección o AdapterView vinculado al adaptador.

Diagrama de funcionamiento en los controles de selección



Lo más importante es que en un control de selección se pueden mostrar muchos datos pero se consumen pocos recursos ya que el adaptador sólo crea los objetos View que se visualizan en pantalla o que están a punto de moverse en la pantalla con lo cual se ahorra memoria. El número de las vistas que se muestran es siempre constante y no se eliminan sino que se reutilizan, lo único que hace el adaptador es modificar el objeto View con la información del elemento a visualizar.

Autoevaluación

Los adaptadores son necesarios cuando queremos implementar un control de selección.

Verdadero Falso

Verdadero

Es cierto, proporciona al control los datos necesarios así como la forma en la que se muestran.

4.2.1.- ArrayAdapter

Ya se ha explicado que un adaptador se encarga de poblar de datos a la vista. En el ejemplo que realizaremos tendremos un Spinner que mostrará un listado de ciudades. Cada ciudad estará identificada por su código postal y nombre. Utilizaremos ArrayAdapter, que tendrá una lista de elementos del tipo ciudad.

Lo primero que haremos será definir el **modelo** o clase Java:

```
public class City {  
  
    private String code;  
    private String name;  
  
    public City(int code, String name) {  
        this.code = code;  
        this.name = name;  
    }  
  
    @NonNull  
    @Override  
    public String toString() {  
        return this.name;  
    }  
}
```



Es importante implementar el método `toString()` ya que de forma predeterminada ArrayAdapter crea una vista para cada elemento de la lista llamando a este método y colocando el contenido en un TextView de un layout que se pasa como parámetro al construir el objeto ArrayAdapter, como muestra el siguiente ejemplo:

```
ArrayAdapter<String> adapter = new ArrayAdapter<City>(this,  
        android.R.layout.simple_spinner_item, listCity);
```

Al crear un objeto de la clase ArrayAdapter se ha pasado tres parámetros:

1. El contexto, es decir la Activity que contiene el control **Spinner**.
2. El **Layout** que se usará para dibujar cada elemento del Spinner (en este caso `android.R.layout.simple_spinner_item` que ya viene definido por Android y contiene un TextView con un **id** llamado `text1`. Es este componente lo que se actualiza con el valor que devuelve el método `toString()` de la clase modelo **City**.
3. Como tercer parámetro, la **colección de datos** que hemos tenido que crear previamente

Para inicializar el ArrayAdapter se crea una lista de ciudades llamada `listCity`:

```
ArrayList<City> listCity = new ArrayList<>();<br />listCity.add(new City(1,"Sevilla"));<br />listCity.add(new City(2,"Málaga"));<br />listCity.add(new City(3,"Huelva"));<br />listCity.add(new City(4,"Jaén"));<br />listCity.add(new City(5,"Córdoba"));<br />listCity.add(new City(6,"Almería"));
```

Sólo queda definir un **Layout** o diseño, ya que cuando se pincha sobre el control Spinner aparece una lista debajo y hay que definir el diseño de la misma mediante el método `setDropDownViewResource(int)` del Adapter. En el ejemplo utilizaremos de nuevo un diseño y definido en Android: `simple_spinner_dropdown_item`:

```
arrayAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
```

A continuación, se debe identificar el Spinner que se ha creado en el diseño:

```
spCity=findViewById(R.id.spCity);
```

Finalmente, unimos el Spinner que ya hemos identificado con el Adapter que hemos creado y definido, utilizando el método `setAdapter()` como mostramos a continuación:

```
spCity.setAdapter(arrayAdapter);
```

4.2.2.- Eventos

En este apartado queremos escuchar el evento de cambio de elemento seleccionado en el control Spinner. Para ello hay que crear una instancia nueva de la clase anónima AdapterView.OnItemSelectedListener que implementa la interfaz e implementar el método:

```
public void onItemSelected(AdapterView<?> adapterView, View view, int i, long id)
```

A continuación explicamos los parámetros de este método:

- ✓ AdapterView<?> adapterView: hace referencia al control donde ocurrió el evento. El signo de interrogación dentro de los corchetes angulares indica que es un tipo genérico, ya que el componente sobre el que ocurrió el evento puede ser del tipo ListView, GridView o spinner. En nuestro ejemplo se trata de un Spinner.
- ✓ View view: devuelve el objeto view que contiene el TextView que muestra el nombre de la ciudad.
- ✓ int position: este entero hace referencia a la posición del elemento en la lista que ha sido seleccionado.
- ✓ long id: es el id de la fila del elemento que se seleccionó.

El siguiente código ilustra cómo mostrar un mensaje emergente Toast mostrando los datos del elemento seleccionado:

```
Toast.makeText(MainActivity.this, "Elemento seleccionado " + parent.getAdapter().getItem(position).toString(), Toast.LENGTH_SHORT).show();
```

Ejercicio Resuelto

Intenta realizar el ejercicio que se ha planteado en este apartado. Ten en cuenta que en el diseño de la interfaz se ha realizado las siguientes acciones:

1. Se ha personalizado el título con una fuente externa.
2. En cada EditText se muestra un texto de ayuda al usuario
3. Todos los controles TextView y ~~EditText~~ están alineados verticalmente.
4. Se ha añadido un botón flotante o FloatingActionButton

[Mostrar retroalimentación](#)

Puedes descargar el ejercicio completo en el siguiente enlace:

[Ejercicio_SpinnerCity \(zip - 720.07 KB\)](#).

4.3.- Recycler View

¿Sabes que el componente RecyclerView es la versión mejorada de ListView?

Al usar las listas era frecuente tener que personalizar el Adapter extendiendo de la clase BaseAdapter. Al manejar gran cantidad de datos se vio que las listas eran inefficientes ya que cada vez que se hacia un desplazamiento en la lista, se creaba de nuevo cada objeto View con todos los controles que tuviera.

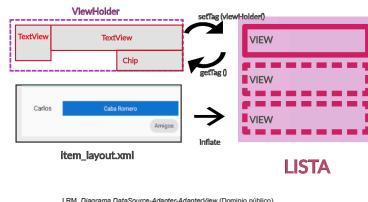
Para agilizar la carga de los datos en el Adapter se creó el patrón ViewHolder que consiste en crear una clase que representara a la vista. Esta clase tiene como atributos los objetos View que se tengan que actualizar en cada fila.

¿Qué se conseguía con esto? Al iniciar el listado se comprueba si la vista que se recicla (parámetro convertView del método getView()) del adapter es nula. Si es así, significa que no se ha creado ningún objeto View para esa fila y por tanto:

1. Se crea un objeto View inflando el fichero XML que representa a cada elemento de la fila.
2. Se crea un objeto ViewHolder.
3. Se inicializan los atributos del objeto ViewHolder al resultado de buscar cada componente en el objeto View mediante el método findViewById().
4. Se almacena el objeto ViewHolder dentro del objeto View mediante el método setTag(Object object).
5. Se devuelve el objeto View.

Si la vista a reciclar no es nula, se obtiene el objeto ViewHolder mediante el método getTag() de la vista. Con este truco se evitaba llamar continuamente al método findViewById() para actualizar los controles de la vista.

Diagrama que muestra el patrón ViewHolder



LRM_Diagrama DataSource-Adapter-AdapterView (Dominio público)

Para saber más

Puedes aprender en la siguiente página web cómo se personaliza la clase Adapter y cómo se define la clase ViewHolder. Comprueba como en el método getView() del Adapter se construye el objeto View inflando una vista que se ha definido en un Layout. Además por cada fila se crea un objeto ViewHolder que se guarda en el atributo Tag de la vista y se actualiza con la información del elemento de la lista.

[Ejemplo patrón ViewHolder](#)

Este patrón no fue adoptado por Android hasta la aparición del componente RecyclerView que nos obliga a usar la clase RecyclerView.ViewHolder. Y añade una gran ventaja: RecyclerView reutiliza eficientemente las vistas sin tener que realizar por nuestra parte ninguna comprobación.

Otra de las ventajas es que la información se puede mostrar de diferentes formas y se puede cambiar su diseño en tiempo de ejecución. El componente ListView sólo permite listar los elementos en formato vertical mientras que RecyclerView es mucho más flexible, ya que puede tener los siguientes Manager:

1. LinearLayoutManager: permite definir listas verticales y también horizontales.
2. StaggeredLayoutManager: permite definir listas escalonadas. Su mejor ejemplo es la aplicación Pinterest.
3. GridLayoutManager: en este diseño los elementos se visualizan en cuadrículas. Su mejor ejemplo son las aplicaciones Galería que muestran las imágenes en este formato.

Otra clase que se puede implementar es RecyclerView.ItemDecorator que permite a los desarrolladores decorar las filas del RecyclerView agregando bordes o líneas divisorias. También se puede agregar animaciones utilizando la clase RecyclerView.ItemAnimator de forma más fácil e intuitiva que con ListView.

Ya se han comentado las múltiples ventajas de este componente, pero sí que hay un aspecto que complica la tarea a los desarrolladores, y es la implementación de los eventos clics dentro de un elemento. Con ListView es simple, ya que implementa la interfaz AdapterView.OnItemTouchListener, pero en el caso de RecyclerView es nuestra tarea implementarlo y veremos en los próximos apartados cómo realizarlo.

4.3.1.- Adapter

La clase RecyclerView.Adapter es un Adapter específico de RecyclerView y se encarga de manejar la colección de datos y vincularla con la vista. Esta vista será un objeto de la clase RecyclerView.ViewHolder que como ya se ha explicado contiene las referencias de los componentes visuales o View de cada elemento de la lista. A continuación se explicarán los pasos para crear un Adapter:

- 1.- Crear la clase ViewHolder, que permite obtener referencias de los componentes visuales o View de cada elemento de la lista. Es dentro del constructor de esta clase donde se inicializa los componentes visuales que pertenecen a cada fila del RecyclerView
- 2.- Crear el constructor de RecyclerView que reciba el conjunto de datos. Se suele utilizar un ArrayList de objetos
- 3.- Crear los métodos (añadir, editar o eliminar elementos) para gestionar el conjunto de datos
- 4.- Sobrescribir el método onCreateViewHolder() que infla el Layout que representa a nuestros elementos, y devuelve una instancia de la clase ViewHolder que se ha definido previamente. Recuerda que puedes utilizar la opción Code -> Override Methods... Si introduces los primeros caracteres del método, aparece el componente de texto **Search for** y filtra el listado en base a los caracteres introducidos.



Página | Licencia Pública

En herencia sabemos que un método se ha modificado porque aparece la anotación override en la declaración del método.

- 1.- Sobrescribir el método onBindViewHolder() que enlaza cada dato con cada ViewHolder.
- 2.- Y no menos importante, sobreescibir el método getItemCount() que devuelve el número de elementos a mostrar en el RecyclerView.

Debes conocer

En el siguiente enlace se explica cómo crear un listado con el componente RecyclerView. Intenta realizar el ejercicio siguiendo los pasos de la página.

[Ejemplo guiado](#)

En el caso de no poder haberlo realizado puedes descargar el código en el siguiente enlace a GitHub.

[Ejemplo guiado ya resuelto](#)

Para saber más

Si quieras ampliar tus conocimientos sobre el componente RecyclerView, como por ejemplo, personalizar la decoración de los elementos de la lista, consulta la siguiente guía:

[Using the RecyclerView](#)

4.3.2.- Eventos

Para implementar un evento dentro de cada vista de un RecyclerView hay que crear un listener que se encargue de controlar este comportamiento en el adapter. Hay tres posibles soluciones que planteamos a continuación:

1. Que sea el propio RecyclerView el encargado de implementar la interfaz View.OnClickListener
2. Que sea la clase ViewHolder quien implemente la interfaz View.OnClickListener
3. Que la clase que implementa la interfaz View.OnClickListener esté en la Activity a través de un listener y el RecyclerView tenga que llamar al listener de la Activity.

La opción a escoger dependerá de la complejidad de nuestra aplicación. Las dos primeras opciones se explican en el video de esta sección, con lo cual nos centraremos en explicar cómo implementar la tercera opción.

Lo primero que hay que hacer es pasar el oyente como parámetro al constructor y luego asignarlo cuando enlazo los datos a la vista en el método onBindViewHolder(). Dentro de la clase del Adapter hay que declarar una interfaz que especifique el comportamiento del oyente. En el siguiente ejemplo, mostramos un listado de notas o **Note** con lo cual por lo que el clic devolverá un elemento de ese tipo:

```
public interface OnItemClickListener {  
    void onItemClick(Note note);  
}
```



http://www.iconarchive.com/icon.php

El constructor recibirá un objeto que implementa esta interfaz, junto con la lista que contendrá el Adapter:

```
private final List list;  
private final OnItemClickListener listener;  
public ContentAdapter(List list, OnItemClickListener listener) {  
    this.list = list;  
    this.listener = listener;  
}
```

Ahora, en el método en onBindViewHolder(), viewHolder recibirá el listener del constructor mediante un método de enlace que crearemos llamado bind() en el método de enlace personalizado:

```
@Override public void onBindViewHolder(ViewHolder holder, int position) {  
    holder.bind(list.get(position), listener);  
}
```

Este método bind() tendría el siguiente código:

```
public void bind(final Note note, final OnItemClickListener listener) {  
    ...  
    itemView.setOnClickListener(new View.OnClickListener() {  
        @Override public void onClick(View v) {  
            listener.onItemClick(note);  
        }  
    });  
}<br />
```

A continuación, desde la actividad se inicia todos los componentes y se muestra un mensaje indicando la nota seleccionada:

```
//Se identifica el RecyclerView  
rvnote = findViewById(R.id.rvnote);  
rvnote.setLayoutManager(new LinearLayoutManager(this, LinearLayoutManager.VERTICAL, false));  
listener = new NoteAdapter.OnItemClickListener() {  
    @Override  
    public void onItemClick(Note note) {  
        Toast.makeText(ListNoteActivity.this, "Nota seleccionada: " + note.toString(),  
        Toast.LENGTH_LONG).show();  
    }  
};  
adaptnote = new NoteAdapter(list, listener);  
rvnote.setAdapter(adaptnote);
```

Debes conocer

Como ya se ha indicado en este video se explica cómo el **Adapter** o bien la clase **ViewHolder** pueden implementar el evento clic cuando se pulsa sobre un elemento de la lista.

Asignar eventos a un RecyclerView

<http://www.youtube.com/embed/ZZ91wlmy3C4>

Programación y más - Eventos en cada elemento de un RecyclerView
[Ir a la descripción del video](#)

Ejercicio Resuelto

Intenta realizar el ejercicio que hemos planteado en este apartado.

1. Crear el modelo de la clase nota o **Note**.
2. Crear la clase **Adapter** e implementa los métodos necesarios para manejar los datos y crear la vista.
3. Crear la clase **ViewHolder** con las vistas que creas conveniente por ejemplo un título y un contenido.
4. Implementar el evento clic dentro de unos de los elementos de la lista.

[Mostrar retroalimentación](#)

¡Seguro que lo has conseguido! Recuerda que siempre que esté bien implementado la solución que has adoptado puede ser diferente a la que te proponemos. Todo depende de la complejidad y la arquitectura de la aplicación.

[PersonalNote: ejercicio que implementa eventos en RecyclerView](#) (zip - 10.31 MB).

Caso práctico



Una vez que **Juan** y **María** han terminado de insertar algunos controles de tipo View, la primera pantalla o Activity ya está lista. Pero les surgen ciertas dudas que deciden preguntar a **Ada**:

Juan: Si en lugar de utilizar esta aplicación con un dispositivo móvil en posición vertical el usuario lo visualiza en posición horizontal, ¿cómo se vería el diseño? ¿Podría modificarse para visualizar mejor los objetos en esa nueva posición?

Ada: Por supuesto que sí, **Juan**, de hecho Android incorpora las herramientas necesarias para ello. Es lo que se llaman "recursos" y se guardarán en ficheros en una carpeta aparte para que, dependiendo del caso, la aplicación utilice unos u otros.

María: ¿Y podemos hacer lo mismo con el idioma? Seguro que aquel que tenga instalado el usuario, ¿se podría mostrar el texto en su propio idioma?

Ada: Claro. Es sencillo, consultad la aplicación de Hola Mundo. Recordad que el texto que sale en la pantalla lo modificasteis en un fichero distinto a la plantilla de la interfaz?

Juan: Sí, era un fichero XML llamado strings en la carpeta /res/values/

Ada: Pues eso es un ejemplo de cómo se crean los recursos necesarios para personalizar nuestra aplicación y hacerla más portable a cualquier dispositivo. Os explicaré cuáles son los recursos más utilizados.

¿Es importante que una aplicación funcione en el mayor número de dispositivos móviles posibles?

Nuestro objetivo como desarrolladores de aplicaciones Android es precisamente que éstas funcionen en un gran número de ellos. Debemos saber la gran variedad de dispositivos que existen en cuanto a características de pantalla, diferentes idiomas o incluso las diferentes versiones API de Android que usan. Es por tanto fundamental tener en cuenta cuantas más características mejor para poder crear recursos alternativos y que la aplicación pueda mostrarse de la mejor forma posible.

¿Qué tipo de recursos podemos identificar?

En la carpeta res/ de cualquier proyecto Android podemos ver diferentes subcarpetas donde se almacenan los ficheros que contienen estos recursos. Cuando compilamos nuestro proyecto se genera automáticamente la clase R.java que contiene un ID por cada recurso definido en el directorio res/. Este ID es un número entero estático que nos permite acceder al recurso desde un fichero Java. Para cada tipo de recurso, hay una subclase R que indicamos en la siguiente tabla:



Morten Rand-Hendriksen

Recursos usados en una aplicación

Recurso XML	Recurso Java	Funcionalidad
res/anim/	R.anim	Contiene las animaciones del proyecto.
res/drawable/	R.drawable	Contiene diferentes subcarpetas que almacenan los gráficos o imágenes dependiendo de la densidad de la pantalla del dispositivo (hdpi, mdpi, ldpi, etc.).
res/font/	R.font	Contiene las fuentes personalizadas del proyecto (ficheros .ttf, .ttc, .otf o .xml)
res/layout/	R.layout	Almacena las plantillas en formato XML de nuestras pantallas.
res/menu/	R.menu	Contiene los ficheros de menú de las actividades y fragmentos.
res/mipmap/	R.mipmap	Contiene el ícono de la aplicación. Se debe identificar el ícono de la pantalla por cada tipo de densidad y colocar el ícono en diferentes carpetas mipmap.
res/raw/	R.raw	Contiene recursos en formato binario, entre los más comunes vídeos y sonido.
res/values/		En este directorio se almacenan varios recursos entre los más comunes el color, el estilo, las cadenas de caracteres.
res/xml/	R.xml	Contiene otros ficheros XML. En esta carpeta se suelen guardar los ficheros que hacen referencia a las preferencias.

En el directorio res/values/ se almacenan varios recursos:

1. strings.xml: los textos que se muestran en los objetos objetos View.
2. dimens.xml: almacena medidas expresadas en dp que se aplican a ciertos atributos de las vistas como margin, padding...
3. arrays.xml: para matrices de recursos.
4. styles.xml: guarda los estilos creados.
5. colors.xml: guarda los colores que se utilizan en toda la aplicación.

Si se quiere crear subcarpetas para organizar ficheros no se puede utilizar la carpeta res/, se podría crear la carpeta assets/ en el mismo que nivel que res/. Además se pueden utilizar nombres de archivos que contengan mayúsculas o signos, lo que no está permitido dentro de res/. A estos recursos no se les asigna un ID en el fichero R.java, son leídos mediante la clase AssetManager como un flujo de bytes.

Android distingue entre dos tipos de recursos:

- ✓ **Recursos por defecto:** son aquellos que toma la aplicación cuando no se define ninguna configuración especial o no encuentra un recurso alternativo.
- ✓ **Recursos alternativos:** son aquellos que se guardan en una carpeta específica mediante un sufijo para indicar cuándo deben ser utilizados.

Debes conocer

En el siguiente video puedes ver un tutorial donde se explican los diferentes tipos de recursos alternativos y se dan algunos ejemplos sobre su utilización.

Recursos alternativos de un proyecto Android

<https://www.youtube.com/embed/jwzK7M2WkQ>

Resumen textual alternativo.

En el siguiente enlace puedes consultar los posibles sufijos a utilizar en tu proyecto si quieres añadir diferentes recursos alternativos. Recuerda que se pueden utilizar sufijos concatenados. En los siguientes apartados veremos algunos de estos recursos más utilizados.

[Tipos de recursos y sufijos](#)

5.1.- Según la orientación de la pantalla

¿Qué ocurre cuando ponemos nuestro dispositivo móvil en posición horizontal?

Este simple gesto es muy habitual y debe tenerse muy en cuenta a la hora de diseñar las interfaces de usuario de las aplicaciones, porque a veces no resulta conveniente utilizar la misma plantilla si nuestro dispositivo está en posición vertical u horizontal.

Veamos esto con un ejemplo. Supongamos que tenemos una interfaz gráfica como la que se muestra en la imagen de la izquierda que podría estar hecho con un LinearLayout y cuatro botones en orientación vertical. Si giramos nuestro emulador verímos el resultado, tal y como se muestra en la imagen de la derecha.

La mejor solución sería crear un recurso alternativo de la siguiente forma:

- ✓ **Paso 1:** crear una carpeta llamada `/res/layout-land/`. Puedes hacerlo pinchando el botón derecho del ratón sobre la carpeta `/res/` y elegir New - Android Resource Directory. En el tipo de recurso seleccionamos Layout y entre los posibles atributos (Available Qualifiers) seleccionamos la orientación (Orientation) y elegimos la opción horizontal (Landscape). Si nos fijamos en el nombre del directorio que se va a crear es el mismo que el original añadiendo el sufijo `-land`, de esta manera cuando el dispositivo se encuentre en posición horizontal, automáticamente cogerá de esta carpeta la plantilla a mostrar.
- ✓ **Paso 2:** crear el fichero XML correspondiente a la nueva plantilla alternativa que debe mostrarse cuando el dispositivo esté en horizontal dentro de la carpeta `res/layout-land/`. El nombre del fichero debe ser el mismo de la plantilla que se toma por defecto en la carpeta `/res/layout/`. Para crear dicho fichero nos situamos en la carpeta recién creada y con el botón derecho del ratón seleccionamos New - Layout resource file. A continuación ponemos el mismo nombre del fichero original y ya podremos editar con otro tipo de layout diferente el contenido de la nueva plantilla. (NOTA: Una buena opción suele ser copiar y pegar el fichero original y modificar el tipo de layout por el que más se ajuste cambiando los atributos que sean necesarios).

Ahora, cuando la aplicación detecta que el dispositivo está orientado horizontalmente se cargaría la plantilla alternativa. Podría quedar algo así utilizando para este nuevo caso un TableLayout.



Captura del IDE Eclipse. Eclipse Public License

¡Vamos, inténtalo tú ahora!

Seguro que con los conocimientos de los diferentes layouts y los distintos tipos de controles no te costará sacar el código y probarlo en tu equipo.

5.2.- Según la configuración del idioma

¿Has pensado en crear una aplicación que la puedan utilizar distintos tipos de personas según su idioma?

Esto es fácil de implementar con los recursos alternativos de Android.

Una buena práctica es abstraer los textos de la aplicación en sí. Por eso cuando insertamos cualquier objeto View que utiliza algún atributo que muestra un texto, por ejemplo android:text, podemos indicar que tome el valor de un fichero externo situado en la carpeta /res/values/. Concretamente nos estamos refiriendo al fichero strings.xml que ya has utilizado anteriormente. Si hacemos esto nos será muy fácil crear un recurso alternativo para otro idioma añadiendo el sufijo correspondiente a la carpeta /values/.

Vamos a verlo utilizando el ejemplo del caso anterior. Supongamos que abrimos la aplicación anterior en un dispositivo cuya configuración tiene como idioma el inglés. Si no tenemos creado el recurso alternativo apropiado, se cogerá por defecto el fichero strings.xml de la carpeta /res/values/.

Probablemente estás utilizando un emulador configurado en inglés. Puedes cambiar su configuración pulsando en el ícono Settings que puedes encontrar en la pantalla inicial. Despues selecciona Language & input y ya podemos elegir el idioma español.

Partiendo de que tenemos configurado el emulador en el idioma español, cuando ejecutaremos la aplicación, ésta mostrará los textos por defecto de su carpeta de recursos. Pero fíjate cómo teniendo antes configurado el emulador con el idioma inglés, también nos mostraba los textos en español. Vamos a crear la carpeta de recursos alternativos para el idioma inglés.

✓ **Paso 1:** crear una carpeta llamada /res/values-en/. El nombre de la carpeta debe coincidir con el recurso por defecto añadiendo el sufijo -en. Para ello podemos proceder de la misma forma que en el apartado anterior. Desde el botón derecho del ratón sobre la carpeta /res/values/ elegimos New -> Android Resource Directory. En el tipo de recurso seleccionamos Values y entre los posibles atributos (Available Qualifiers) seleccionamos la localización (Locale) y elegimos el idioma (Language) que deseamos. Si nos fijamos en el nombre del directorio que se va a crear es el mismo que el original añadiendo el sufijo del idioma -en (Inglés), de esta manera cuando el dispositivo tenga la configuración en este idioma accederá a los recursos almacenados en dicha carpeta, como por ejemplo el fichero strings.xml que guardaría los textos de la aplicación traducidos a inglés.

✓ **Paso 2:** crear el fichero strings.xml dentro de esta nueva carpeta y modificar los valores de los textos que queremos traducir. Para ello es buena idea copiar el fichero origen, pegarlo en esta nueva carpeta y a partir de ahí sustituir las cadenas de texto que queremos, por ejemplo las siguientes líneas:

```
<string name="Button1">One</string>
<string name="Button2">Two</string>
<string name="Button3">Three</string>
<string name="Button4">Four</string>
```

NOTA: los identificadores de los recursos no se deben cambiar. Se mantienen los mismos que aparecen en la plantilla para que puedan ser reconocidos de la misma forma. Sólo cambiariamos los textos One, Two, Three y Four que corresponden a los valores que se mostrarán dentro de los botones.

Ya está todo listo para que puedas probar la aplicación. Si dejas la configuración en español deberían aparecer los valores de los textos del recurso por defecto y si cambias de nuevo la configuración del emulador a inglés la aplicación detecta el recurso alternativo de este idioma y cogerá los valores de la carpeta correspondiente.

¡Ánimo, pruébalo!

Y recuerda cambiar también el nombre de la aplicación, es conveniente modificar todos los valores de texto, no sólo los que aparecen en los objetos View.



Reflexiona

¿Has pensado qué ocurriría si además de mostrar los textos en inglés el usuario cambia la orientación de su dispositivo?

5.3.- Según las características de la pantalla

Pensemos ahora en la cantidad de dispositivos que hay en el mercado, los diferentes fabricantes que ofrecen modelos con distintos tamaños y densidades de pantalla.

¿Qué ocurre por ejemplo con las imágenes cuando las vemos en pantallas con características diferentes? ¿Cómo se muestran los botones?

En cuanto a los recursos alternativos referentes a las pantallas podemos encontrarnos en la documentación oficial de , bastantes aspectos a tener en cuenta pero vamos a centrarnos en los recursos que se refieren al tamaño de la pantalla que como ya vimos puede ser **pequeña, normal, grande y extra grande** y por otro lado en las diferentes **densidades de las pantallas** que pueden ser **ldpi, mdpi, hdpi y xhdpi**.

En la carpeta /res/ encontramos la carpeta mipmap donde por defecto se almacena el fichero ic_launcher.png que corresponde al icono de la aplicación y está adaptada para que pueda verse en distintos dispositivos con diferentes densidades. De la misma forma, en la carpeta drawable se almacenan los recursos gráficos de la aplicación y es recomendable que éstas se puedan ver correctamente dependiendo de las distintas características de los dispositivos. Para ello se podrán crear las subcarpetas necesarias con los sufijos apropiados para que la aplicación acceda a cada una de ellas dependiendo de las características del dispositivo.

En cuanto al tamaño, podemos personalizar los elementos que forman nuestro layout para que se visualicen lo mejor posible y almacenar estas alternativas añadiendo el sufijo correspondiente a la carpeta /res/layout/. Podemos encontrar entonces la carpeta layout-small, layout-normal, layout-large y layout-xlarge. En estas guardaremos con el mismo nombre de la plantilla que queremos personalizar según el tamaño y modificaremos el código necesario para mejorar la apariencia.



Autoevaluación

Si queremos crear recursos alternativos para el idioma francés utilizamos la siguiente carpeta de recursos:

- /res/drawable/
- /res/values/
- /res/layout/
- /res/language/

Incorrecta.

La opción correcta es /res/values/ donde añadiremos el sufijo -fr quedando la carpeta /res/values-fr/ e incorporando el fichero strings.xml con los valores adecuados. En /res/drawable/ se añaden las imágenes, en /res/layout/ los diseños de pantalla y la carpeta /res/language/ no es ningún recurso definido por Android.

No es correcta.

No es la opción correcta.

Solución

1. Incorrecto
2. Opción correcta
3. Incorrecto
4. Incorrecto

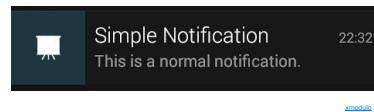
5.4.- Imágenes de una aplicación

El diseño de nuestra aplicación es un factor importante y el ícono de nuestra aplicación es la primera imagen que el usuario encontrará en el lanzador de aplicaciones o **Launcher**. Los iconos también deben ser compatibles con todos los dispositivos y tendrán diferentes tamaños según el lugar que ocupen. Android Studio contiene la herramienta **Image Asset** que permite crear los siguientes tipos de íconos:

- ✓ **Iconos de selector o Launcher icon:** este ícono representa nuestra aplicación en la pantalla de inicio del dispositivo, en Google Play Store o en cualquier otro lugar. Estos íconos se almacenan en la carpeta res/mipmap-density.
- ✓ **Iconos de pestañas y barras de acciones:** son los que se muestran en la barra de acción y representan una acción individual. Los íconos de pestañas se usan en las aplicaciones que tienen múltiples pestañas o **Tabs**, estos íconos se almacenan en la carpeta res/drawable-density.
- ✓ **Iconos de notificaciones:** identifica que una notificación pertenece a una aplicación en base al ícono que se muestra al usuario. Estos íconos se almacenan en la carpeta res/drawable-density.



Ashuoj Hung



Debes conocer

Consulta la documentación oficial donde puedes ver cómo se crean los diferentes íconos de una aplicación. Comprueba cómo **Image Asset** ayuda a crear íconos de nuestra aplicación para todas las densidades de pantalla y los almacena en las carpetas de densidad respectivas.

[Cómo crear íconos de apps con Image Asset Studio](#)

Android Studio incluye una herramienta llamada **Vector Asset** que se usa para agregar algunos íconos de material predefinidos junto con su propio Gráfico vectorial escalable ([SVG](#)) y ficheros Adobe Photoshop ([.PSD](#)) como archivos vectoriales en su aplicación de Android. Las imágenes vectoriales no son compatibles con Android 4.4 (API nivel 20) y versiones inferiores.

Esta herramienta crea recursos Vector Drawable. Para que una aplicación los soporte se debe agregar la siguiente línea en el archivo build.gradle de nivel de módulo:

```
android {  
    defaultConfig {  
        vectorDrawables.useSupportL.library = true  
        ...  
    }  
}
```

Para saber más

Consulta la documentación oficial donde puedes ver cómo crear gráficos vectoriales y posteriormente agregarlos en un diseño de nuestra aplicación.

[Cómo agregar gráficos vectoriales de varias densidades](#)

Autoevaluación

El ícono de una aplicación no es necesario que se sea compatible con diferentes resoluciones de pantalla, ya que el sistema adaptará el ícono por nosotros.

Verdadero Falso

Falso

El ícono de la aplicación se muestra en diferentes lugares de la aplicación, con lo cual debe tener un tamaño adecuado al lugar donde se muestre.

Caso práctico

Maria está investigando cómo mejorar la apariencia de los elementos que se han insertado en la interfaz de usuario que ha creado. Encuentra información respecto al uso de **temas** para que la aplicación guarde una apariencia homogénea. Pero también ha visto algo sobre los **estilos**.

- ✓ ¿Qué diferencia hay entre ambos conceptos?
- ✓ ¿Cuándo aplicar un tema o un estilo?
- ✓ ¿Dónde se guardan los estilos?

Ada va a orientar un poco a **Maria** sobre este tema.



La apariencia de una aplicación es un detalle muy importante a tener en cuenta para que tenga éxito o no. Hay que cuidar, por ejemplo, los colores que se utilizan como fondo o para el tipo de fuente, el tamaño, etc.

Un **estilo** y un **tema** son dos conceptos muy similares, ya que ambos recopilan las características de apariencia común a los objetos a los que se van a aplicar. La única diferencia es que un estilo se puede asociar a objetos de tipo View mientras que un tema se asocia a objetos de tipo Activity.

Los estilos se almacenan en la carpeta de recursos /res/values/ en el fichero de nombre style.xml. Existen estilos ya predefinidos y el desarrollador puede crear otros nuevos heredando de un estilo padre y creando los ítems correspondientes a los atributos XML que queremos personalizar.



Viaje Everett

```
<resources>
    <style name="EstiloNuevo" parent="@android:style/TextAppearance.Medium">
        <item name="android:textColor">#00FF00</item>
        <!-- Escribir el resto de ítems que se desean personalizar -->
    </style>
</resources>
```

Para aplicar un estilo a un objeto de tipo View se utiliza el atributo style y se asigna el valor del recurso. Por ejemplo, si tenemos definido un estilo llamado EstiloNuevo y queremos aplicarlo a un TextView escribiremos lo siguiente:

```
<TextView style="@style/EstiloNuevo"
    android:text="@string/Saludo" >
```

Los temas son estilos que se aplican a una actividad completa en lugar de cada elemento.

Sólo se aplicarán aquellas características que soporte cada uno de los elementos. Como veremos más adelante, una aplicación tendrá varias pantallas o Activity, por tanto también podemos aplicar el tema a toda la aplicación. En ambos casos tenemos que editar el fichero de manifestio AndroidManifest.xml con las siguientes líneas dependiendo de dónde queramos aplicar el tema:

- ✓ En la etiqueta <application>:

```
<application android:theme="@style/TemaNuevo" >
```

- ✓ En la etiqueta <activity>:

```
<activity android:theme="@style/TemaNuevo" >
```

Debes conocer

En el siguiente video se explica cómo crear y aplicar tanto estilos como temas. También la posibilidad de heredar de otros estilos ya creados. Es fundamental que le eches un vistazo.

Estilos y temas en Android

<http://www.youtube.com/embed/10ugleF1NM>

[Resumen textual alternativo.](#)

También es interesante ver qué estilos y temas hay predefinidos para poder utilizarlos y heredar de ellos.

[Lista de estilos y temas predefinidos en Android](#)

7.- Barra de acción

La barra de acción o ActionBar es el elemento encargado de presentar el nombre de la aplicación o actividad en la que nos encontramos, los botones de las acciones disponibles y el menú de desbordamiento u Overflow representado por el icono de tres puntos verticales. Este componente apareció en Android 3.0 y es un componente nativo de Android de forma que se muestra de forma diferente según la versión del dispositivo Android.

Para poder mostrar nuestra aplicación con el mismo diseño, independientemente de la versión de Android, Google creó el componente Toolbar o barra de herramientas que se encuentra en la librería de soporte y que no depende de la versión de Android.

Este componente hereda de ViewGroup, con lo cual puede contener otros componentes y aunque proporciona la misma funcionalidad que ActionBar es más flexible. Por ejemplo, se podría utilizar este componente para añadir un título dentro de una tarjeta CardView.

Lo primero que hay que hacer es deshabilitar la barra de acción aplicando un tema NoActionBar en el elemento <application> del fichero manifest:

```
android:theme="@+id/appBarLayout"
```

Al establecer el tema dentro de este elemento ninguna de las actividades que contenga la aplicación mostrará la barra de acción, con lo cual habrá que añadir el widget Toolbar en todos los diseños de las actividades con el siguiente código:

```
<androidx.appcompat.widget.Toolbar  
    android:id="@+id/toolbar"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:background="@color/colorPrimary"  
    android:elevation="4dp"  
    android:minHeight="?attr/actionBarSize"  
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"  
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light"/>
```

Ya en el código Java, identificamos la barra de herramientas y establecemos la barra de herramientas como la barra de acción de nuestra aplicación:

```
@Override protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);  
    setSupportActionBar(toolbar);  
}
```

A veces nos interesa que esta barra de herramientas se anime en base al desplazamiento o scroll que ocurre en la vista del contenido, por ejemplo con un RecyclerView. Esto se consigue con la clase CoordinatorLayout que se encarga de coordinar automáticamente todos los hijos: AppBarLayout, RecyclerView y el botón flotante. En este caso la barra Toolbar está dentro de un Layout especial llamado AppBarLayout que es el encargado de sincronizarse con el contenido que se desplaza mediante el atributo layout_behavior.

```
app:layout_behavior="@string/appbar_scrolling_view_behavior"
```

Realmente es bastante sencillo, sólo tienes que utilizar el esquema que te planteamos en el siguiente archivo:

[plantillaCoordinatorLayout.xml](#) (xml - 1.71 KB).

Para saber más

El siguiente enlace muestra una serie de ejemplos que explican las diferentes animaciones que se pueden tener en un CoordinatorLayout y que dependen de los elementos hijos que tenga.

¡Es muy interesante! Verás cómo interactúan los diferentes elementos de la interfaz en base a los atributos que se definen en los componentes.

[Componentes CoordinatorLayout](#)

Autoevaluación

Indica qué elemento no forma parte de la ActionBar

- Barra de estado.
- Título de la aplicación.
- Botones de acción.
- Menú desbordamiento

Lo tenías claro, ¡¡ enhorabuena !!

Incorrecto.

No es correcto.

No es la opción correcta.

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto
4. Incorrecto

7.1.- Creando un menú principal con submenús I

¿Un menú es una opción importante dentro de una aplicación?

Claro que sí, ya que presenta una serie de opciones que permite realizar una serie de acciones en una aplicación Android.

Las acciones más importantes se mostrarán con un ícono en la barra de acción pero el espacio es limitado, con lo cual el resto de opciones estarán en el **menú desbordamiento o ampliado**. Para codificar los menús vamos a utilizar le lenguaje **XML**, al igual que hicimos en la elaboración de los layouts. No obstante se podría hacer mediante código Java. El fichero estará localizado en la carpeta de recursos `/res/menu/` y si Android Studio no nos lo crea por defecto podremos hacerlo nosotros pulsando en el botón derecho sobre la carpeta `/res/` y a continuación elegimos New - Android resource file y elegimos como tipo de recurso (Resource type) **menu** y le damos un nombre al fichero **XML** que tendrá el menú.

Para cada opción del menú se debe añadir un elemento `<item>` que puede tener los siguientes atributos: un `ID` (`android:id`), un ícono (`android:icon`) o un título (`android:title`). Con el uso de la `ActionBar` debemos tener en cuenta el atributo `app:showAsAction` que puede tomar algunos de estos valores:

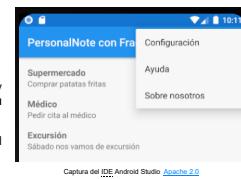
- ✓ "never": nunca se mostrará el botón en la barra.
- ✓ "ifRoom": se muestra el botón en la barra si hay sitio suficiente para ello.
- ✓ "always": se muestra siempre el botón en la barra de acción.
- ✓ "withText": se muestra también el texto en el botón.

Es posible combinar varios valores de la siguiente forma:

```
<code><app:showAsAction></code>="ifRoom|withText"
```

Siguiendo con el ejemplo anterior, si deseas cambiar el menú principal para que se visualice en la barra de acción puedes añadir las líneas de código que se destacan con negrita en el siguiente ejemplo:

```
xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/action_order"
        android:icon="@drawable/ic_action_order"
        android:title="@string/action_order"
        app:showAsAction="always"></item>
    <item
        android:id="@+id/action_share"
        android:icon="@drawable/ic_action_share"
        android:title="@string/action_share"
        app:showAsAction="always"></item>
    <item
        android:id="@+id/action_settings"
        android:title="@string/menu_settings"
        android:orderInCategory="98"/>
    <item
        android:id="@+id/action_help"
        android:title="@string/menu_help"
        android:orderInCategory="99"/>
    <item
        android:id="@+id/action_aboutus"
        android:title="@string/menu_aboutus"
        android:orderInCategory="100"/>
</menu>
```



Para saber más

Para saber más sobre el recurso menú de Android puedes visitar la página oficial, donde encontrarás todos los atributos de dicho recurso. Si deseas insertar íconos en las opciones de menú puedes utilizar algunos que vienen por defecto en los recursos drawable del sistema y utilizar el atributo `android:icon`.

[Recurso menú de Android](#)

7.2.- Creando un menú principal con submenús II

Cuando ejecutemos la aplicación veremos los botones en la barra de acción y el funcionamiento seguirá siendo el mismo. Se puede tener un menú con un submenú dentro de una de las opciones, sería similar al siguiente código:

```
<?xml version="1.0" encoding="utf-8"?><br /><menu xmlns:android="http://schemas.android.com/apk/res/android">  
    android:id="@+id/action_settings"  
    android:title="@string/menu_settings"  
    android.orderInCategory="98">  
        <menu>  
            <item android:id="@+id/action_modo_noche"  
                android:title="@string/menu_modo_noche"/>  
        </menu>  
        .....  
</menu>
```

Si te fijas bien, verás que hay que editar el fichero strings.xml para poner los recursos de tipo texto que aparecerán en cada una de las opciones del menú. Por ejemplo las siguientes:

```
<string name="action_order">Ordenar por string</string>  
<string name="menu_abouts">Sobre nosotrosstring</string>  
<string name="menu_help">Ayudastring</string>  
<string name="menu_settings">Configuraciónstring</string>  
<string name="action_share">Compartirstring</string>
```

El siguiente paso será llamar a este menú desde nuestra clase Java donde queremos que se visualice, en nuestro caso en la actividad principal. Primero hay que obtener una referencia al objeto Inflater con el método getMenuInflater() para después generar el menú con el método inflate() pasando el identificador.

```
public boolean onCreateOptionsMenu(Menu menu) {<br />    getMenuInflater().inflate(R.menu.menu, menu);<br />    return true;<br />}<br />
```

Aunque ya puedes comprobar que se visualiza el menú en tu emulador, faltaría proporcionar la funcionalidad a los botones, lo que conseguiremos sobre escribiendo el método onOptionsItemSelected() al que se le pasa el objeto de tipo MenuItem que almacena la opción pulsada.

```
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.action_order:  
            Toast.makeText(this,  
                "Has pulsado la opción ordenar alfabéticamente", Toast.LENGTH_SHORT).show();  
            return true;  
        case R.id.action_share:  
            Toast.makeText(this,  
                "Has pulsado la opción compartir", Toast.LENGTH_SHORT).show();  
            return true;  
        .....  
        default:  
            return super.onOptionsItemSelected(item);  
    }  
}
```

Al lanzar la aplicación veremos que pulsando sobre el botón menú nos aparecerán las opciones que hemos creado, incluyendo el submenú que tiene la primera opción.

Autoevaluación

Para mostrar mostrar el menú que se ha definido en un recurso menu hay que llamar al método inflate() de la clase MenInflater

Verdadero Falso

Verdadero

Caso práctico



Juan y María están trabajando juntos en la gestión de eventos cuando a Juan le surge una duda:

- ¿Existe en Android alguna forma de mostrar mensajes de información al usuario?
- ¿Te refieres a mensajes que se muestran en ventanas como suele hacerse en otros lenguajes?
- Sí, eso es, normalmente suelen ser de información, confirmación, etc.
- He leído sobre este tema en Internet y he encontrado información sobre algunos tipos de notificaciones al usuario.

¿Cómo podemos mostrar un mensaje al usuario cuando éste realiza una acción?

¿Existe la posibilidad de mostrar en pequeñas ventanas algunos mensajes?

¿Pueden esos mensajes incorporar algunos botones?

Android dispone de varias técnicas para notificar mensajes al usuario, pero nos centraremos en la utilización de los siguientes objetos para este propósito:

✓ El objeto **Toast**: permite mostrar mensajes al usuario en pantalla y se ocultan automáticamente en un breve espacio de tiempo.

✓ El objeto **AlertDialog**: permite mostrar información al usuario para confirmar o solicitar información. Se utilizan ventanas de diálogo donde se muestra el mensaje y uno, dos o incluso tres botones.



8.1.- El objeto Toast

¿Cómo mostrar un mensaje breve al usuario?

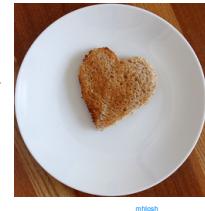
Podemos emplear el **objeto Toast**. Este objeto utiliza el método `makeText()` al que se le pasan tres parámetros:

- ✓ El **contexto** de la aplicación que se puede recuperar con el método `getApplicationContext()`.
- ✓ El **mensaje de texto**, que no debe ser muy extenso.
- ✓ La **duración** que puede tomar los valores `LENGTH_SHORT` o `LENGTH_LONG` dependiendo de si queremos que sea corta o larga. Debemos tener en cuenta que este tipo de mensaje no dura más de 5 segundos por tanto el texto a mostrar debe ser breve para que el usuario pueda leerlo.

Por último utilizamos el método `show()` para mostrar el objeto `Toast` en pantalla.

Vamos a ver un ejemplo:

```
Toast.makeText(getApplicationContext(),"Este es el texto a mostrar", Toast.LENGTH_SHORT).show();
```



Por defecto el mensaje nos aparecerá en la parte inferior de la pantalla, pero si queremos cambiar su alineación podemos utilizar el método `setGravity()` al cual le indicamos mediante alguna de las constantes de la clase `Gravity` qué posición debe ocupar (CENTER, BOTTOM, LEFT, etc.). También se pueden combinar algunas de ellas.

```
Context context = getApplicationContext();
Toast toast = Toast.makeText(context, text, duration);
toast.setGravity(Gravity.CENTER(Gravity.RIGHT 0, 0));
toast.show();
```

8.2.- El objeto AlertDialog

¿Y para pedir confirmación al usuario, cómo lo hacemos?

Con el objeto AlertDialog podemos mostrar un mensaje en forma ventana de diálogo y esperar a la confirmación por parte del usuario. También es útil para solicitar determinada información al usuario.

La clase AlertDialog define el estilo y la estructura, pero debemos apoyarnos en la clase DialogFragment, ya que dispone de los controles necesarios para construir la ventana de diálogo. Por tanto debemos asegurarnos que disponemos de la librería android-support-v4.jar instalada en nuestro proyecto para que la aplicación funcione en dispositivos que tengan instalada una versión inferior a Android 3.0 (API 11).

Podemos implementar varias tipos de diálogo dependiendo de la finalidad que busquemos:

- ✓ **Alerta:** la ventana está formada por un título (opcional), un mensaje y un único botón de Aceptar.
- ✓ **Confirmación:** la ventana está formada por un título (opcional), un mensaje y al menos dos botones (Aceptar y Cancelar).
- ✓ **Selección:** la ventana está formada por un título y una serie de opciones a seleccionar.
- ✓ **Personalizados:** la ventana está formada por un layout personalizado (XML) y un botón de Aceptar.



Para implementar una ventana de diálogo podemos seguir los siguientes pasos:

- 1.- Crear una clase nueva que extienda de la clase DialogFragment y sobreescribir el método onCreateDialog() que construirá el diálogo.
- 2.- Crear un objeto de tipo AlertDialog.Builder cuyos métodos nos servirán para asignar las propiedades a la ventana de diálogo. Algunos de estos métodos son los siguientes:
 - 2.1.- setTitle(): establece el título de la ventana. Se puede prescindir de él si no es necesario.
 - 2.2.- setMessage(): establece el texto del mensaje en el área reservada para el contenido.
 - 2.3.- setPositiveButton(): establece un botón de Aceptar o de OK en la parte inferior.
 - 2.4.- setNegativeButton(): establece un botón de Cancelar en el parte inferior.

Aquí vemos un ejemplo de implementación de mensaje de alerta o información al usuario:

```
public class EjemploAlertDialog extends DialogFragment {  
    @Override  
    public Dialog onCreateDialog(Bundle savedInstanceState) {  
        //Creamos el objeto AlertDialog.Builder  
        AlertDialog.Builder builder =  
            new AlertDialog.Builder(getActivity());  
        // Asignamos las propiedades que se mostrarán.  
        builder.setTitle("Importante")  
            .setMessage("Debes leer este mensaje.")  
            .setPositiveButton("Aceptar",  
                new DialogInterface.OnClickListener() {  
                    public void onClick(DialogInterface dialog, int id) {  
                        //Acciones a realizar cuando pulsamos el botón.  
                        dialog.cancel();  
                    }  
                });  
        return builder.create();  
    }  
}
```

Cuando queramos que se abra la ventana de diálogo anterior desde nuestra aplicación al pulsar un botón, por ejemplo, tendremos que utilizar el método getSupportFragmentManager() como en el siguiente código:

```
boton.setOnClickListener(new View.OnClickListener() {  
    public void onClick(View v) {  
        FragmentManager fragmentManager = getSupportFragmentManager();  
        DialogoAlerta dialogo = new DialogoAlerta();  
        dialogo.show(fragmentManager, "tagAlerta");  
    }  
});
```

Debes conocer

Intenta implementar algunos ejemplos más sobre otro tipo de ventanas de diálogo (confirmación, selección, etc.). Puedes consultar el siguiente videotutorial para ver más ejemplos.

Crear un AlertDialog - Tutorial Android Studio

<http://www.youtube.com/embed/ljNa5LyjeE>

[Resumen textual alternativo](#)

Autoevaluación

Si queremos mostrar al usuario un mensaje en la pantalla con dos botones “Aceptar” y “Cancelar” utilizaremos un objeto:

- AlertDialog** de tipo Alerta.
- AlertDialog** de tipo Confirmación.
- Toast**.
- Toast** con dos botones.

Incorrecto.

Esta es la única respuesta correcta, pues se utiliza para aceptar o cancelar una petición por parte del usuario. El objeto Toast no pide nunca confirmación al usuario, y el AlertDialog de tipo alerta sólo lleva un botón de Aceptar cuando el usuario ha leído el mensaje.

No es correcto.

No es la opción correcta.

Solución

- 1. Incorrecto
- 2. Opción correcta
- 3. Incorrecto
- 4. Incorrecto

8.3.- El objeto Snackbar



¿Has oido hablar de la utilidad de Snackbar?

El widget Snackbar proporciona un comentario breve respecto a una acción y se muestra en la parte inferior de la pantalla. La ventaja de este componente es que nos ofrece la capacidad de realizar una acción. El ejemplo más común es la acción **UNDO**, deshacer una acción que se acaba de realizar o bien **RETRY**, reintentar en el caso que la acción haya fallado.

Al igual que un mensaje Toast, los Snackbars desaparecen automáticamente después de un tiempo de espera, pero al contrario que el primero, Snackbar se puede desactivar, por ejemplo, cuando el usuario pulsa en otra parte de la pantalla o bien cuando se descarta desplazando el Snackbar hacia un lateral. Para crear un mensaje Snackbar utilizamos el método make().

```
public void simpleSnackbar(View view){  
    Snackbar.make(view, getString(R.string.message), Snackbar.LENGTH_SHORT).show(); }
```

El método make() acepta tres parámetros:

1. view: es el diseño raíz de la actividad.
2. R.string.message: este es el mensaje que debe aparecer en el Snackbar. Al utilizar un recurso se puede personalizar e internacionalizar.
3. Snackbar.LENGTH_LONG: se indica límite de tiempo que se mostrará la barra del Snackbar.

Observa cómo se encadena la llamada del método show() para mostrar Snackbar en la pantalla una vez ha finalizado el método make(). Esta técnica se llama **Method chaining** y evita utilizar variables intermedias en la obtención de un resultado.

Se debe utilizar como diseño raíz un CoordinatorLayout en el caso que nuestro diseño tenga el botón FloatingActionButton. De esta forma el botón se desplaza automáticamente hacia arriba para mostrar el mensaje en la parte inferior de la pantalla y cuando desaparece el mensaje, el botón vuelve a su posición original.

Para saber más

Para saber definir una acción dentro del componente Snackbar puedes visitar la siguiente página, que te guía sobre cómo hacerlo. Intenta realizar el ejemplo por ti mismo.

¡Seguro que puedes!

[Añadir una acción en una notificación Snackbar](#)

Caso práctico



Ada está muy satisfecha con lo que han conseguido **Maria** y **Juan**. Ahora ha llegado el momento de dar un paso más. Los ha reunido para ampliar los conocimientos sobre Android. Les va a explicar qué son los Intents o intenciones y qué relación guardan con las actividades o pantallas que ya conocen.

Comienza por mostrarles una aplicación sencilla que consta de una pantalla o Activity principal desde donde el usuario puede ir accediendo a otras pantallas interactuando con los elementos que se muestran en ellas.

En el apartado de eventos ya hemos visto cómo un usuario puede interactuar con los elementos que incorpora una determinada Activity. ¿Pero cómo podríamos hacer que un usuario pueda accionar determinados servicios (como llamar por teléfono o enviar mensajes) o bien navegar entre distintas pantallas o Activity? En el desarrollo de cualquier aplicación es necesario emplear más de un objeto Activity, por tanto la comunicación entre ellas es importante. De todo esto se encarga la clase Intent.

A través de un objeto de la clase Intent solicitamos una acción abstracta a otro componente de nuestra aplicación. La acción más frecuente es iniciar una actividad aunque también se puede iniciar servicios, y receptores de transmisión.

Las intenciones se pueden clasificar en dos tipos:

✓ **intents explícitos.** Al pulsar un botón debe lanzarse una nueva Activity o pantalla hablaremos de intents explícitos porque indicamos la clase que queremos lanzar.

✓ **intents implícitos.** Si sólo conocemos lo que queremos realizar pero no qué clase es la encargada de lanzarlo

De momento nos vamos a centrar en el primer tipo de intents puesto que nos interesa hacer aplicaciones donde el usuario pueda navegar entre diferentes pantallas.



Angelo González

9.1.- Creando una nueva actividad

Para crear una nueva actividad en nuestra aplicación debemos crear una nueva clase Java que se almacenará en la carpeta /src/. Este fichero utilizará como recurso una nueva plantilla que se guarda con formato XML en la carpeta /res/layout/. Por tanto debemos generar dos ficheros nuevos en nuestro proyecto además de los ya creados por defecto desde el inicio por Android Studio.

Para comenzar vamos a partir de un nuevo proyecto que vamos a crear con el IDE Android Studio al que podemos llamar HelloWorld y a partir de éste crearemos la primera actividad (pantalla) que enviará un mensaje a una segunda actividad. Una vez hayas creado el proyecto y tengas los ficheros que genera de forma automática para la única actividad que tiene la aplicación, podemos crear una segunda actividad con los siguientes pasos:

1. Desde la carpeta raíz o paquete donde se encuentra el fichero MainActivity.java de nuestra aplicación pulsamos con el botón derecho del ratón y elegimos New – Activity - Empty Activity y ponemos como nombre de la actividad (clase java) SecondActivity. Vemos cómo automáticamente se renombra la plantilla XML que va asociada a dicha clase java (activity_second).
2. Pulsamos en el botón Finish y vemos cómo ahora tenemos dos clases java y dos ficheros layout en nuestra aplicación.
3. Editamos de forma gráfica (pestaña diseño) el layout de la nueva actividad y añadimos un objeto TextView que muestre por pantalla el texto "Hasta Luego" totalmente centrado vertical y horizontalmente. Recuerda utilizar el recurso de tipo String para almacenar el valor.

Ahora mismo tenemos dos actividades o pantallas en nuestra aplicación, pero todavía no se comunican entre ellas. Pasemos a ver cómo se hace.



9.2.- Comunicar varias actividades

¿Como comunicar las diferentes actividades de nuestra aplicación?

Una vez que nuestra aplicación está compuesta por varias actividades, vamos a ver cómo se puede acceder desde la actividad principal al resto, que consideraremos secundarias.

Siguiendo con el ejemplo anterior necesitaremos modificar el layout principal para insertar un botón que nos sirva de enlace con la segunda actividad. Para eso hacemos lo siguiente:

1. Abrir el fichero /res/layout/activity_main.xml correspondiente a la plantilla de la pantalla principal.
- 2.- Añadir un objeto de tipo Button debajo del TextView. Utiliza el recurso strings.xml para el texto del botón.

En este momento podemos utilizar el evento onClick del objeto Button para indicar la acción deseada. Recordemos cómo se puede gestionar este evento desde Java e implementamos el código de la siguiente forma (hay que recordar importar las clases que necesitaremos para dicho código, puedes utilizar Alt + Intro):

```
package com.example.usuario.holamundo;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.view.View.OnClickListener;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity implements OnClickListener{

    protected static final int REQUESTCODE_SECOND_ACTIVITY = 10;
    private Button btSend;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btSend=findViewById(R.id.btSend);
        btSend.setOnClickListener(this);
    }

    public void onClick(View v){
        sendMessage();
    }

    public void sendMessage(){
        Intent intent = new Intent(this,SecondActivity.class);
        startActivityForResult(intent, REQUESTCODE_SECOND_ACTIVITY);
    }
}
```



Cuando declaramos el objeto Intent pasamos el propio contexto de la actividad (this) y el nombre de la clase java que debe lanzarse y finalmente se inicia la segunda actividad mediante el método startActivityForResult() (en lugar de startActivity()) indicando que queremos obtener el resultado obtenido por la segunda actividad en forma de otro objeto Intent. Se pasa por parámetro un identificador REQUESTCODE_SECOND_ACTIVITY que nos ayude a saber qué actividad hemos iniciado y de la cual podemos obtener una respuesta. Hay que tener en cuenta que desde una actividad principal se puede llamar a varias actividades secundarias, así que necesitaremos identificar posteriormente el resultado de su ejecución.

Cuando se añade una nueva actividad o pantalla a una aplicación debemos prestar atención al fichero AndroidManifest.xml y comprobar que se ha registrado correctamente un elemento nuevo de tipo <activity> como se muestra a continuación:

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

En caso de que no tuviésemos la actividad registrada en el fichero de manifiesto no será reconocida por la aplicación y por tanto nos dará error. Si nos fijamos bien en el atributo android:name, simplemente ponemos el símbolo de puntuación seguido del nombre de la nueva actividad.

Cuando ejecutamos la aplicación comprobamos que al pulsar sobre el botón se abre la nueva actividad en cuyo layout se puede ver el TextView que dice "Hasta Luego".

Autoevaluación

Es necesario que una actividad nueva esté registrada en el fichero de manifiesto si queremos integrarla en la aplicación.

Verdadero Falso

Verdadero

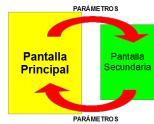
Es verdadero, porque en caso contrario la aplicación no tendrá constancia del nuevo objeto.

9.3.- Pasar parámetros de la actividad principal a otra secundaria

Lo que pretendemos realizar en las siguientes secciones es pasar ciertos parámetros entre dos actividades.

La primera pasará los parámetros a través del objeto Intent y la segunda informará a la primera actividad cuando finalice también mediante parámetros siguiendo el esquema de la imagen de la derecha.

Vamos a ver cómo la primera actividad puede enviar información a la segunda. Mostraremos las dos opciones:

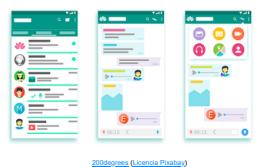


A través de los métodos que nos ofrece el objeto Intent.

Creando nosotros un objeto Bundle.

9.4.- Obtener resultado de una actividad secundaria

Ya se ha comentado que la segunda actividad se inicializó con el método `startActivityForResult()` y debe devolver el resultado de su ejecución a la primera actividad en forma de Intent. Esto se realiza mediante el siguiente código:



El primer parámetro `resultCode` indica si la actividad secundaria se ha ejecutado bien o no, pudiendo tomar los valores `RESULT_CANCELED` o `RESULT_OK` y el parámetro `data` contiene el Intent a devolver. Este Intent también puede tener parámetros que la primera actividad necesite.

Para finalizar la actividad secundaria y que el control siga en la actividad principal debemos utilizar el método:

```
finish();
```

Vamos a seguir con el ejemplo planteado HelloWorld y añadiremos un botón en la actividad secundaria para que al pulsar volvamos de nuevo a la actividad principal. Para ello además de dibujarlo en el fichero layout le asignaremos la funcionalidad tal y como hicimos en la pantalla principal, añadiendo el código siguiente en el fichero java de la actividad secundaria y diciendo que nuestra clase implemente el interface `OnClickListener`:

```
btReturn=findViewById(R.id.btReturn);
btReturn.setOnClickListener(this);
```

Por último implementamos el método `onClick()` con el siguiente código:

```
public void onClick(View v) {
    Intent data = new Intent();
    data.putExtra("return", "Venimos de la segunda pantalla");
    setResult(RESULT_OK, data);
    finish();
}
```

Volvemos a la actividad principal, ya que necesitamos sobrescribir el método `onActivityResult()` en la clase `MainActivity` que se ejecuta automáticamente cuando la segunda actividad finaliza:

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if(requestCode == REQUEST_CODE_SECOND_ACTIVITY){
        if(resultCode == RESULT_OK){
            String valor = (String) data.getExtras().get("return");
            Toast toast =
                Toast.makeText(getApplicationContext(),
                valor, Toast.LENGTH_SHORT);
            toast.show();
        }
    }
}
```

Recomendación

En el siguiente enlace puedes descargar el proyecto de ejemplo en su totalidad.

[Aplicación de ejemplo sobre el paso de parámetros entre actividades \(zip - 0.76 MB\)](#).

Autoevaluación

El método utilizado para enviar datos a la otra actividad es `getExtras(clave)`.

Verdadero Falso

Falso

Es falso porque este método recoge los valores enviados desde una actividad. El método que envía los datos es `putExtra(clave, valor)`.

Ejercicio Propuesto

Realiza una aplicación que contenga una actividad principal y dos actividades secundarias con el siguiente diseño:

- ✓ La actividad principal consistirá básicamente en:
 - un objeto `EditText` donde el usuario pueda introducir su nombre;
 - dos objetos `Button` para poder abrir cada una de las actividades secundarias;
 - un objeto `TextView` donde se mostrará el contenido del parámetro pasado de cada una de las actividades secundarias.
- ✓ Las actividades secundarias estarán formadas cada una de ellas por:
 - un objeto `TextView` donde aparecerá el nombre del usuario que se le habrá pasado como parámetro desde la actividad principal;
 - un objeto `EditText` para introducir una palabra;
 - un objeto `Button` para poder regresar a la actividad principal.

El funcionamiento de la aplicación será el siguiente:

- 1.- El usuario introduce su nombre en la pantalla principal y puede pulsar uno de los dos botones (**Pantalla 1** o **Pantalla2**).
- 2.- A cada una de las dos pantallas se debe pasar como parámetro su nombre para que se escriba como texto en el objeto `TextView`.
- 3.- Además, en cada pantalla el usuario podrá introducir una palabra de forma que al pulsar al botón "**Volver**" ésta se pasará a la pantalla principal como parámetro.
- 4.- Al regresar a la pantalla principal, se debe visualizar la palabra pasada como parámetro en el objeto `TextView`.

[Mostrar retroalimentación](#)

Te puedes basar en el proyecto HelloWorld que hemos hecho anteriormente. Sólo tienes que tener en cuenta los objetos `View` que debes implementar en cada pantalla y gestionar el paso de parámetros tal y como se ha hecho en el proyecto de ejemplo.
¡Animo que te servirá para practicar y entender mejor este apartado!

10.- Creando un menú contextual

Para construir un menú contextual seguiremos los mismos pasos que en el menú principal pero con ciertas particularidades que tenemos que tener en cuenta. Si tenemos varios elementos en la pantalla, ¿podemos abrir diferentes menús contextuales dependiendo del objeto en el que estamos haciendo una pulsación larga? ¿Se guardarán los menús como recursos al igual que velamos en el menú principal?

Vamos a definir un menú contextual para el objeto de tipo TextView que tenemos en la actividad principal del proyecto HolaMundo.

Para ello primero creamos el fichero XML en la carpeta /res/menu/ como cualquier otro menú, pulsando el botón derecho y seleccionando New - Menu resource file. Ponemos un nombre al fichero (mcontextualetiqueta.xml) y le damos la siguiente estructura:

```
<?xml version="1.0" encoding="utf-8"?><br /><menu xmlns:android="http://schemas.android.com/apk/res/android" ><br />    <item android:id="@+id/opCtx1" android:title="@string/opCtx1"></item><br />    <item android:id="@+id/opCtx2" android:title="@string/opCtx2"></item><br /></menu>
```



Hay que recordar editar el fichero strings.xml para completar los textos de las opciones que queremos que aparezcan en el menú.

Dentro del fichero .java de la clase principal registramos en el método OnCreate() este menú mediante registerForContextMenu() donde pasamos como parámetro el identificador del elemento al que queremos aplicar el menú contextual. En nuestro caso el menú contextual lo haremos sobre la etiqueta cuyo identificador es textView2.

```
TextView etiqueta = (TextView)findViewById(R.id.textView2);  
registerForContextMenu(etiqueta);
```

Y añadimos el siguiente método para poder mostrar el menú:

```
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuItemInfo menuInfo){  
    super.onCreateContextMenu(menu, v, menuInfo);  
  
    MenuInflater inflater = getMenuInflater();  
    inflater.inflate(R.menu.mcontextualetiqueta, menu);  
}
```

Y para asignar la funcionalidad correspondiente a la pulsación de la opción elegida utilizaremos el método:

```
public boolean onContextItemSelected(MenuItem item){  
  
    switch (item.getItemId()) {  
        case R.id.opCtx1:  
            Toast.makeText(getApplicationContext(),  
                "Opción 1 del menú contextual", Toast.LENGTH_SHORT).show();  
            return true;  
        case R.id.opCtx2:  
            Toast.makeText(getApplicationContext(),  
                "Opción 2 del menú contextual", Toast.LENGTH_SHORT).show();  
            return true;  
        default:  
            return super.onContextItemSelected(item);  
    }  
}
```

Recomendación

En el siguiente enlace puedes descargar el proyecto del ejemplo en su totalidad. Puedes estudiar su código con lo que se ha explicado anteriormente.

[Aplicación de ejemplo sobre la utilización de menús.](#) (zip - 26.73 MB)

Ejercicio Resuelto

- ✓ Crea un nuevo objeto de tipo TextView en el layout de la actividad principal.
- ✓ Crea otro menú contextual que se abra si hacemos una pulsación larga sobre este nuevo elemento.
- ✓ Modifica el código del proyecto descargado anteriormente para poder visualizar ambos menús contextuales según pulsemos en una etiqueta o en la otra.

[Mostrar retroalimentación](#)

Después de añadir el nuevo TextView y creado el fichero de menú correspondiente con las opciones a mostrar:

- 1.- Registra el nuevo menú en la clase.
- 2.- Para mostrar el menú que corresponda al elemento pulsado puedes utilizar este código cambiando los identificadores que tú hayas puesto. Fíjate que debemos comparar el identificador del objeto que se ha pulsado para mostrar el menú correspondiente.

```
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuItemInfo menuInfo){  
    super.onCreateContextMenu(menu, v, menuInfo);  
    MenuInflater inflater = getMenuInflater();  
  
    if(v.getId() == R.id.textView1)  
        inflater.inflate(R.menu.mcontextualetiqueta1, menu);  
    else if(v.getId() == R.id.textView2)  
    {  
        inflater.inflate(R.menu.mcontextualetiqueta2, menu);  
    }  
}
```

Autoevaluación

Tanto si estamos implementando un menú principal como si es contextual, si queremos programar la funcionalidad de sus opciones utilizamos el método (o los métodos):

- [onContextItemSelected\(\)](#)
- [onCreateContextMenu\(\)](#)
- [onOptionsItemSelected\(\)](#)
- [onClickItemSelected\(\)](#)

[Mostrar retroalimentación](#)

Solución

1. Correcto
2. Incorrecto
3. Correcto
4. Incorrecto

Caso práctico



Ada les propone un nuevo reto a María y Juan, y es realizar la aplicación que han desarrollado para que sea visible en dispositivos con pantallas más grandes como Tablets.

María y Juan se miran y piensan en el trabajo que les espera al tener que diseñar de nuevo las interfaces para este tipo de pantallas. Ada les tranquiliza y les dice que no se preocupen, que podrán reutilizar las interfaces ya creadas, pero que se tendrán que adaptar al nuevo tipo de pantallas mediante los fragmentos o Fragments.

Ada les muestra una aplicación que consta de varios fragmentos y cómo la misma aplicación se visualiza de una forma en el móvil y de otra en una tablet.

Los fragmentos surgen con la versión 3.0 de Android, ya que aparecen en el mercado diferentes dispositivos con diferentes tamaños de pantalla, de forma que el diseño de la interfaz se tenía que adaptar para aprovechar al máximo el espacio de pantalla en cualquiera de las orientaciones (Landscape y Portrait).

Un fragmento o Fragment es una porción de la interfaz de usuario que está dentro de una actividad, de hecho tiene su propio diseño (recurso Layout) y tiene su propio ciclo de vida aunque está ligado al ciclo de vida de la Activity de la que depende. Por ejemplo, cuando la actividad está pausada, también lo están todos sus fragmentos, y cuando la actividad se destruye, todos los fragmentos que dependen de esa actividad se destruyen.

Los fragmentos pueden ser de dos tipos:

- ✓ **Estáticos:** se declara directamente en el fichero XML o Layout de la Activity. Este fragmento una vez que está visible no podrá ser eliminado ni sustituido por otro.
- ✓ **Dinámicos:** se crea en código Java y se suelen añadir dentro de un ViewGroup normalmente FrameLayout. Este Viewgroup será el contenedor de los fragmentos que se añaden y eliminan en tiempo de ejecución.

Nosotros utilizaremos fragmentos dinámicos, con lo cual la estructura de nuestras aplicaciones cambia, ya no hay que ir cambiando entre distintas actividades con el consiguiente gasto de memoria sino que dentro de una sola Activity se pueden lanzar varios Fragments que interactúan entre ellos a través de su Activity, es lo que llamamos **transacciones de fragmentos**.

Dentro de nuestra aplicación crearemos una clase que hereda de Fragment y que implementará al menos estos tres métodos:

- ✓ **onCreate():** este método lo llama el sistema operativo para crear el fragmento. Se deben inicializar los componentes del fragmento de forma que se conservarán si el fragmento se detiene y se reanuda.
- ✓ **onCreateView():** este método se llama para dibujar la interfaz gráfica del fragmento, de forma que será visible en la Activity. Este método devuelve un objeto View resultado de inflar el Layout del fragmento. Puede ser un valor nulo en el caso de que el fragmento no tenga UI.
- ✓ **onPause():** si la actividad que lo contiene entra en pausa, se llama automáticamente a este método. Se deben guardar los valores que se deban mostrar al usuario cuando la actividad se reanude de nuevo, mostrando la interfaz en el mismo estado que el usuario la dejó.

Recuerda que es el sistema operativo quien se encarga del control de los ciclos de vida, es decir, de llamar a los métodos del ciclo de vida de los componentes cuando corresponda. Cuando se llama al método onStart() de la Activity se llama a continuación al método onStart() de todos sus fragmentos. Hasta que no terminan todos los métodos onStart() de los fragmentos no finaliza el método onStart() de la Activity.

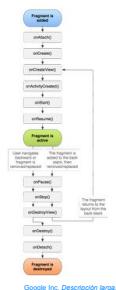
Esta secuencia de llamadas que el sistema realiza (métodos callback) hay que tenerlas en cuenta y como desarrolladores no implementar operaciones largas en el tiempo que bloquee el proceso de nuestra aplicación.

Ejercicio Resuelto

El uso de fragmentos estáticos es más frecuente en diseños para tablets donde una parte de la interfaz interesa que quede fija. Si quieres ver cómo se implementan, visualiza el fichero activity_main.xml y verás que se añade el fragmento estático mediante el siguiente código:

```
<strong><fragment</strong>
    android:id="@+id/fragment"
    android:name="com.example.staticfragment.StaticFragment"<strong></strong>
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginTop="10dp"/>
```

[Descarga el ejemplo StaticFragment](#) (zip - 14.19 MB).



Google Inc. Propiedad de Google Inc.

11.1.- Transacciones de fragmentos

Ya sabemos que las actividades se organizan en una pila (la **pila de actividades**) en el orden en que se abre cada actividad. Pero ahora son los fragmentos los que se añaden y eliminan mediante transacciones que se guardan en una pila interna que gestiona cada actividad. Como estas transacciones se guardan en la Activity que se encuentra en la pila de actividades, cuando el usuario presiona el botón **Atrás**, el sistema operativo que gestiona la pila de actividades, deshace la última transacción de fragmentos de la actividad que se encuentra en la cima la pila de actividades.

Si hay un giro de pantalla sabemos que la Activity se destruye ¿pero qué pasa con la pila de fragmentos?:

El sistema Android restaura automáticamente esa pila de fragmentos de forma que no se pierde, y si creamos un nuevo fragmento se posicionará encima de los que ya estaban guardados.

Ya tenemos clara la teoría así que vamos a crear nuestro primer proyecto con fragmentos. Para ello partimos del ejercicio PersonalNote que hemos renombrado como PersonalNoteFragment y que usaremos de base para entender los fragmentos.



Material gratis ([Licencia Pública](#))

Si tienes duda de cómo copiar o renombrar un proyecto consulta el siguiente enlace:

[Cómo copiar y renombrar un proyecto.](#)

Resumimos los cambios que hemos realizado:

- ✓ Se crea una clase MainActivity con su correspondiente Layout. Hemos copiado toda la estructura que contiene la barra de acción y el botón flotante y lo hemos colocado en este Layout. Además hemos añadido un FrameLayout que será el contenedor de nuestros fragmentos.
- ✓ La clase ListNoteActivity pasa a llamarse ListNoteFragment y hereda de la clase Fragment. Se ha creado el método onCreateView() que infla la vista y el código que estaba en onCreate() lo hemos pasado al método onViewCreated().
- ✓ Ahora, cuando el fragmento necesite referenciar a la Activity que lo contiene, se utilizará el método getActivity().

Ejercicio Resuelto

Descárgate el código del proyecto en el siguiente enlace que hemos transformado para usar Fragments:

[Ejercicio PersonalNoteFragment \(zip - 1.12 MB\)](#)

Si ejecutas el código no aparecerá nada, ya que no se ha añadido el fragmento a la actividad. La clase encargada de agregar, quitar o reemplazar fragmentos es FragmentManager. Esta clase contiene y gestiona todos los fragmentos que estén presentes en nuestra aplicación, así como si son visibles o no. El proceso es similar a las transacciones de bases de datos donde cualquier operación debe ejecutarse en una transacción mediante un objeto FragmentTransaction. Para añadir el fragmento NoteListFragment a nuestra Activity se tienen que realizar las siguientes operaciones:

- 1.- Crear una transacción.
- 2.- Añadir, reemplazar o eliminar un fragmento a través de los métodos add(), replace() y remove().
- 3.- Confirmar la transacción o hacer un commit.

Mostramos a continuación el código que implementa estas operaciones:

```
FragmentManager fm=getSupportFragmentManager();
ft.replace(R.id.fragmentContainer,new ListNoteFragment(),ListNoteFragment.TAG);
ft.addToBackStack(null);<br />
ft.commit();
```

Para saber más

Te proponemos que realices el ejemplo que proponen los desarrolladores de Google a través de su [plataforma Codelabs](#), que contiene pequeños tutoriales que explican cómo implementar ciertas características en pequeños proyectos o aplicaciones. El tutorial que te proponemos explica paso a paso cómo crear fragmentos dinámicos:

[Proyecto para practicar con fragmentos en Codelabs](#)

Autoevaluación

Indica las acciones que se pueden realizar en una transacción de fragmentos.

Añadir.

Editar.

Reemplazar.

Eliminar.

[Mostrar retroalimentación](#)

Solución

1. Correcto
2. Incorrecto
3. Correcto
4. Correcto

11.2.- Comunicación con una Activity



En este apartado veremos cómo un Fragment se comunica con la Activity que lo contiene, bien porque envía o recibe información que necesita. En el proyecto que estamos realizando vamos a añadir un nuevo fragmento llamado ViewNoteFragment que mostrará la información de la nota seleccionada en NoteListFragment. Para ello este último fragmento le pasará un objeto Note a ViewNoteFragment a través de la Activity que contiene a ambos.

La forma más fácil de comunicar el fragmento con su actividad es mediante interfaces. La idea es básicamente definir una interfaz dentro de un fragmento y dejar que la actividad implemente esa interfaz:

1.- Primero se define una interfaz que describa la acción que se desea.

```
interface ListNoteFragmentCallback {  
    void onNoteView(Note note);  
}
```

2.- Implementar la interfaz definida en la actividad.

```
public class MainActivity extends AppCompatActivity <br />    implements ListNoteFragment.ListNoteFragmentCallback
```

3.- Adquirir instancia de la interfaz desde el Fragment.

```
private ListNoteFragmentCallback callback;  
  
<br />@Override  
public void onAttach(Context context) {  
    super.onAttach(context);  
    try {  
        callback = (ListNoteFragmentCallback) context;  
    } catch (ClassCastException e) { <br />        throw new ClassCastException(context.toString() + " must implement  
        ListNoteFragmentCallback");  
    }  
}
```

4.- Invocar la instancia de interfaz cuando sea necesario.

```
listener = new NoteAdapter.OnItemClickListener() {  
    @Override  
    public void onItemClick(Note note) {  
        callback.onNoteView(note);  
    }  
};
```

Autoevaluación

Para comunicar un fragmento con su actividad, el fragmento debe implementar la interfaz que se ha declarado en la actividad que lo contiene.

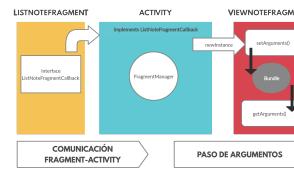
Verdadero Falso

Falso

La interfaz se define dentro del fragmento y es la actividad quien implementa esa interfaz:

11.3.- Pasar argumentos a un fragmento

Ya hemos comunicado el primer fragmento con la Activity. A continuación, es el segundo fragmento el que tiene que recibir el objeto Note como un argumento.



Debemos asegurarnos que los argumentos se pasan al fragmento justo cuando se usa un método estático newInstance() para crear el fragmento, poniendo los parámetros que se desee en un objeto Bundle que estará disponible al crear una nueva instancia:

```
public static Fragment newInstance(Bundle bundle) {  
    ViewNoteFragment fragment = new ViewNoteFragment();  
    if (bundle != null) {  
        <strong>        fragment.setArguments(bundle);</strong>  
    }  
    return fragment;  
}
```

Más adelante en el fragmento se puede acceder al objeto Bundle mediante el método getArguments():

```
@Override  
public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {  
    super.onViewCreated(view, savedInstanceState);  
    tvTitle = view.findViewById(R.id.tvTitle);  
    tvContent = view.findViewById(R.id.tvContent);  
    if (getArguments() != null) {  
        <strong>        Note note = getArguments().getParcelable(Note.TAG);</strong>  
        tvTitle.setText(note.getTitle());  
        tvContent.setText(note.getContent());  
    }  
}
```

Finalmente la actividad inicia el fragmento mediante el método newInstance() con un parámetro de tipo Bundle que contiene el objeto Note:

```
@Override  
public void onNoteView(Note note) {  
    FragmentManager fm = getSupportFragmentManager();  
    viewNoteFragment = (ViewNoteFragment) fm.findFragmentByTag(ViewNoteFragment.TAG);  
    if (viewNoteFragment == null) {  
        Bundle bundle = null;  
        if (note != null){  
            bundle= new Bundle();  
            bundle.putParcelable(Note.TAG.note);  
        }  
        <strong>        viewNoteFragment = (ViewNoteFragment) viewNoteFragment.newInstance(bundle);</strong>  
    }  
    FragmentTransaction ft = fm.beginTransaction();  
    ft.replace(R.id.fragmentContainer, viewNoteFragment, ViewNoteFragment.TAG);  
    ft.addToBackStack(null);  
    ft.commit();  
}
```

Para que un objeto de la clase Note se pueda pasar como argumento debe implementar la interfaz Parcelable, añadiendo implements Parcelable en la declaración de la clase. Cuando se añade esa línea automáticamente Android Studio implementa por nosotros el código pulsando Alt+Intro → Implement Method. A continuación nos da un error en Note que se soluciona de la misma forma Alt+Intro → Add Parcelable Implementation. Si quieres conocer el sentido de los métodos creados puedes consultar el siguiente enlace:

[Implementar la interface Parcelable en una clase en Android](#)

Ejercicio Resuelto

Puedes descargar el ejercicio entero en el siguiente enlace:

[Proyecto PersonalNoteFragment con todo el proceso realizado en los apartados \(zip - 1.23 MB\).](#)

Anexo I.- Librerías de diseño

Es importante indicar que cuando desarrollamos una aplicación se intenta que se admitan múltiples versiones de API, por eso se utilizan las **librerías de soporte**, de forma que podemos proporcionar funciones más recientes a versiones anteriores de Android. Estas librerías nos ofrecen clases y funciones adicionales que no están disponibles en la API estándar del Framework de nuestro proyecto, y es una forma sencilla de dar soporte a más dispositivos.

Si compruebas por ejemplo el proyecto HelloWorld podrás ver que las clases importadas comienzan con android.support.*, estas clases son de la librería de compatibilidad o soporte. Esta librería sólo llega hasta la versión 27.



```
package com.example.usuario.holamundo;  
  
import android.content.Intent;  
import android.support.v7.app.AppCompatActivity;  
.....  
public class MainActivity extends AppCompatActivity implements OnClickListener {
```

Con el lanzamiento de Android 9.0 (API nivel 28), hay una nueva versión de la biblioteca de compatibilidad denominada AndroidX que contiene la biblioteca de compatibilidad existente e incluye los últimos componentes de **Jetpack**. Si compruebas el proyecto ExampleFragments podrás comprobar que la clase que se ha importado comienza por androidx.appcompat.*.

```
package com.example.examplefragment;  
import androidx.appcompat.app.AppCompatActivity;  
import android.os.Bundle;  
public class MainActivity extends AppCompatActivity {
```

Es importante tener claro que no se puede trabajar con ambas librerías. Actualmente cuando creas un nuevo proyecto en Android Studio se utiliza AndroidX, así lo recomienda Google, pero algunos ejemplos de este módulo se realizan utilizando la librería de soporte.

De cualquier forma siempre se puede migrar un proyecto a AndroidX. Enumeraremos los pasos a realizar:

- 1.- Modificar el valor de compileSdk al menos a la versión 28 en el fichero build.gradle del módulo app.
- 2.- Tener al menos la versión com.android.tools.build:gradle:3.2.0 en el fichero build.grade del proyecto. Recuerda que se puede modificar este valor mediante **File -> Project Structure -> Project** y modificar el valor de Android Gradle Plugin Version.



Aparecerán una serie de mensajes en Build indicando que se están descargando los paquetes necesarios.

Si estas dos configuraciones se cumplen se puede realizar la opción **Refactor -> Refactor This -> Migrate to AndroidX**.

Una vez que ha finalizado la refactorización se puede comprobar que en el archivo gradle.properties se ha configurado a true los siguientes complementos:

- ✓ android.useAndroidX=true: que indica que se utilizará la biblioteca AndroidX de forma que cuando se añade una clase o un objeto View buscará en esta librería, no en la librería de soporte.
- ✓ android.enableJetifier=true: este complemento migra automáticamente las bibliotecas de terceros existentes a AndroidX.

Condiciones y términos de uso de los materiales

Materiales desarrollados inicialmente por el Ministerio de Educación, Cultura y Deporte y actualizados por el profesorado de la Junta de Andalucía bajo licencia Creative Commons BY-NC-SA.



Antes de cualquier uso leer detenidamente el siguiente [Aviso legal](#)

Historial de actualizaciones

Versión: 03.00.01

Fecha de actualización: 15/11/20

Actualización de materiales y correcciones menores.

Versión: 03.00.00

Fecha de actualización: 29/05/20

Autoría: Lourdes Rodríguez Morón

Ubicación: 2.8

Mejora (tipo 2): No se explica correctamente el uso de los fragment.

Ubicación: apartado 9.3

Mejora (tipo 2): Esta apartado ya está obsoleto, Android recomienda el uso de Toolbar en lugar de lo que se explica en esta sección.

Ubicación: apartado 3

Mejora (tipo 2): El apartado de los Adapter para el uso de controles de selección no queda claro. Los alumnos suelen preguntar a menudo por ello.

Ubicación: apartado 2.8

Mejora (tipo 2): Todo este apartado ya no funciona en android. La creación de pestañas ya no se realiza de esta forma.

Ubicación: apartado 2

Mejora (tipo 3): Muchos de los layout que se explican están obsoletos y los no se dice nada de los nuevos que recomienda Android, ej constraint. No se explica nada de desarrollo de interfaces mediante el modo gráfico, muy más fácil y de uso más habitual.

Ubicación: 3.2

Mejora (tipo 2): El componente ListView está obsoleto (legacy).

Ubicación: 2.4, 2.6

Mejora (tipo 2): Actualizar los layout. Algunos de estos layout están obsoletos (legacy). Actualizar otros nuevos como ConstraintLayout.

Ubicación: 7.1.- El objeto Toast.

Mejora (tipo 1): En el ejemplo para mostrar el toast aparece:

Toast miToast = Toast.makeText(getApplicationContext(),

"Este es el texto a mostrar",

Toast.LENGTH_SHORT);

miToast.show();

Debe aparecer en su lugar:

Tenemos dos maneras de mostrar el Toast, definiéndolo primero y mostrándolo después:

Toast miToast = Toast.makeText(getApplicationContext());

Ubicación: Nodos del mapa

Mejora (Mapa conceptual): Modificación en base a los nuevos contenidos de la unidad

Ubicación: Índice de contenidos

Mejora (Orientaciones del alumnado): Modificación en base a los nuevos contenidos de la unidad

Versión: 02.00.00

Fecha de actualización: 06/02/14

Autoría: Victor Gil Rodriguez

Actualización de los contenidos a Android.

Versión: 01.00.00

Fecha de actualización: 06/02/14

Versión inicial de los materiales.

