

DESARROLLO DE APLICACIONES PARA DISPOSITIVOS MÓVILES

Contenido

2.1. Arquitectura de android.....	3
2.1.1. Los componentes de una aplicación.....	3
2.1.2. Comunicación entre componentes.....	4
2.1.3. El fichero de manifiesto (AndroidManifest.xml).....	4
2.1.4. La pila de software de android.....	5
2.2 profundizando en el desarrollo: widgets para actividades.....	6
2.2.1. Los controles de entrada.....	6
2.2.2 LAS API's de Android.....	8
2.2.3. La clase TextView.....	10
2.2.4. La clase Button.....	11
2.2.5. Las clases RadioGroup y RadioButton.....	12
2.2.6. La clase Checkbox.....	14
2.2.7. Las clases ToggleButton y Switch.....	16
2.2.8. Las clases ImageView e ImageButton.....	16
2.2.9. La clase EditText.....	19
2.3. Internacionalización de aplicaciones mediante el archivo string.xml.....	21
2.3.1. Creación de aplicaciones multiidioma.....	22
2.3.2. Probando la internacionalización en el emulador.....	25
2.4. Layouts y contenedores.....	26
2.4.1. Propiedades de los layouts.....	26
2.4.2. Relative layouts.....	28
2.4.3. Constraint layouts.....	29
2.4.4. Linear layouts.....	30
2.4.5. Layouts tabulares.....	31
2.4.6. Trabajando programáticamente con contenedores.....	33
2.4.6.1. Recorrido del contenedor.....	33
2.4.6.2. Diferenciando tipos.....	34
2.4.6.3. Añadiendo elementos al contenedor.....	34
2.5. Los dialogos y los fragmentos.....	37
2.6. Los widgets de selección.....	41
2.6.1. Seleccionando multiples elementos.....	43
2.6.2. Los spinners.....	45
2.6.3. Las selecciones personalizables.....	47

2.6.4. Los selectores de fecha/hora.....	51
2.7. Construcción de menús.	56
2.7.1. Dotando de acción a los elementos de menú.....	60
2.7.2. Menús contextuales	61
2.7.2.1. Creación del menú contextual.....	62
2.7.2.2. Creación del menú action bar contextual.....	63
2.8. EL ACTIONBAR	65
2.9. OTROS WIDGETS PARA TU IU	66
2.10. MÁS FUNCIONES DE CALLBACK.....	66
2.11. ESTILOS, TEMAS Y MATERIAL DESIGN	67
2.12. OTRAS PLANTILLAS PARA ACTIVIDADES.....	69

2.1. Arquitectura de android

En el caso práctico de la primera unidad de trabajo creaste tu primera actividad. Esta actividad la definíamos como una tarea simple que el dispositivo móvil realiza. Además de actividades hay otros tres tipos de componentes que pueden formar parte de tu App.

Todos estos componentes se programan en Java con las siguientes reglas:

- El Sistema Operativo Android es un sistema multi-usuario en el que cada App es un usuario diferente.
- Por defecto, el sistema asigna un identificador a cada usuario (User ID) que es sólo conocido por la aplicación y el sistema operativo. El sistema establece permisos para todos los ficheros de la App de tal manera que sólo la App puede acceder a ellos.
- Cada proceso tiene su propia Máquina Virtual (VM), de tal manera que el código de cada aplicación se ejecuta aislado de otras aplicaciones, evitando así potenciales problemas de seguridad.

Pese a estas restricciones hay formas para intercambiar información entre las distintas aplicaciones. Además, para que una App pueda acceder a los recursos del dispositivo, por ejemplo, contactos, cámara, bluetooth, etc. se le deben dar permisos en el momento de la instalación.

2.1.1. Los componentes de una aplicación

Cada componente de una aplicación es un punto de entrada a través del cual el sistema Android puede comunicarse con tu App. Cada tipo de componente tiene un propósito y un ciclo de vida propio que dicta cuando un componente se crea y se destruye.

Hay cuatro tipos de componentes:

LAS ACTIVIDADES: Son las que conoces hasta ahora. Como viste en la UT1, una actividad representa una pantalla simple con una interfaz de usuario. Por ejemplo, una App para gestión de pedidos puede tener una actividad para mostrar el listado de clientes, otra actividad para mostrar un producto determinado y una tercera actividad para mostrar los datos de un pedido en concreto. Además de ser programas independientes, cada actividad puede ser ejecutada por otra App (si tu App lo permite). Por ejemplo, la cámara podría activar la actividad de mostrar un producto determinado para poder compartir la foto y ponerle foto al producto. Para poder crear una actividad debes crear una subclase de la clase **Activity**, o en el caso de aplicaciones con soporte de compatibilidad hacia atrás (Backwards compatibility) la clase **AppCompatActivity**. Esta clase Activity tiene su propio ciclo de vida, es decir, recuerda que no tiene la función “main” típica de Java, sino que se van invocando en cadena a una serie de métodos. La figura a la derecha, que ya has visto con otra forma en la UT1, es la típica que se pone en todas las bibliografías, pero que ilustra perfectamente el ciclo de vida de la actividad.

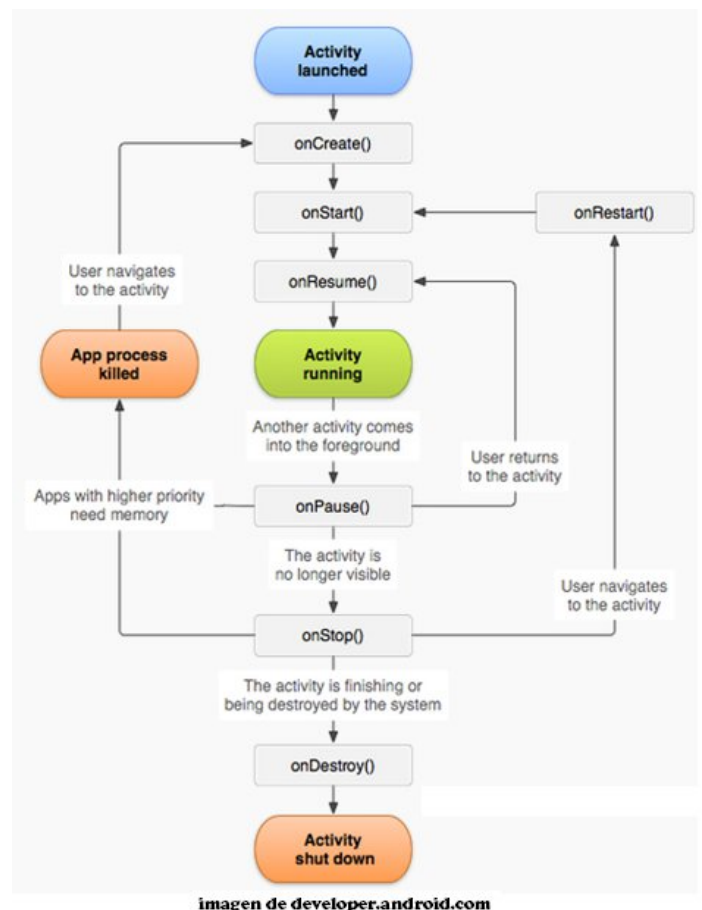


imagen de developer.android.com

```
java.lang.Object
└─ android.content.Context
    └─ android.content.ContextWrapper
        └─ android.view.ContextThemeWrapper
            └─ android.app.Activity
                └─ android.support.v4.app.FragmentActivity
                    └─ android.support.v7.app.AppCompatActivity
```

Fíjate en la imagen anterior, y a partir de ahora, verás que la primera función `onCreate()` vamos a introducir bastante código en las próximas secciones.

LOS SERVICIOS: Un servicio, en inglés *service*, es un programa que no tiene interfaz de usuario y que se ejecuta en *background*. Por ejemplo, un servicio podría estar conectándose a la red para obtener los precios de los productos de tu App en tiempo real. Para poder crear un servicio debes crear una subclase de la clase **Service**.

LOS PROVEEDORES DE CONTENIDOS: Un proveedor de contenido manipula un conjunto compartido de datos de una aplicación. Gracias a este proveedor de contenido una App publica esta información para otras Apps de tal manera que estas pueden acceder al contenido e incluso, si tienen permisos, modificarlas. Por ejemplo, Android tiene un *content provider* para manejar la información de los contactos del usuario. De esta manera, nuestra App de gestión de pedidos, podría actualizar la información de los clientes o de los proveedores directamente en la agenda de contactos. Para poder crear una *ContentProvider* debes crear una subclase de la clase **ContentProvider**.

RECEPTORES DE BROADCAST: Utilizando la nomenclatura de redes, un *Broadcast Receiver* es un programa que recibe un mensaje de multidifusión de alguna App. Por ejemplo, una cámara que acaba de capturar una foto, una App que publica que ha cambiado el precio de algún producto, etc.

2.1.2. Comunicación entre componentes

Cuando quieras que tu aplicación haga fotos con la cámara del dispositivo móvil, no es necesario que programes una aplicación que haga fotos. En su lugar, puedes utilizar mensajes a Apps ya creadas para que hagan esas cosas por ti. Se trata de no reinventar la rueda una y otra vez, y en este aspecto Android nos proporciona su sistema de comunicación entre componentes.

Cuando un componente intenta hacer una llamada a otro componente, por ejemplo, una actividad invoca a otra actividad, se crea un mensaje del componente origen al componente destino, declarando la intención de solicitar algún servicio. Esta intención, en inglés, ***intent***, es la forma asíncrona que tienen los componentes de comunicarse en tiempo de ejecución. La comunicación a través de intents están en contraposición a la comunicación a través de las llamadas a las APIs que, pudiendo estas ser síncronas o asíncronas se enlazan en tiempo de compilación.

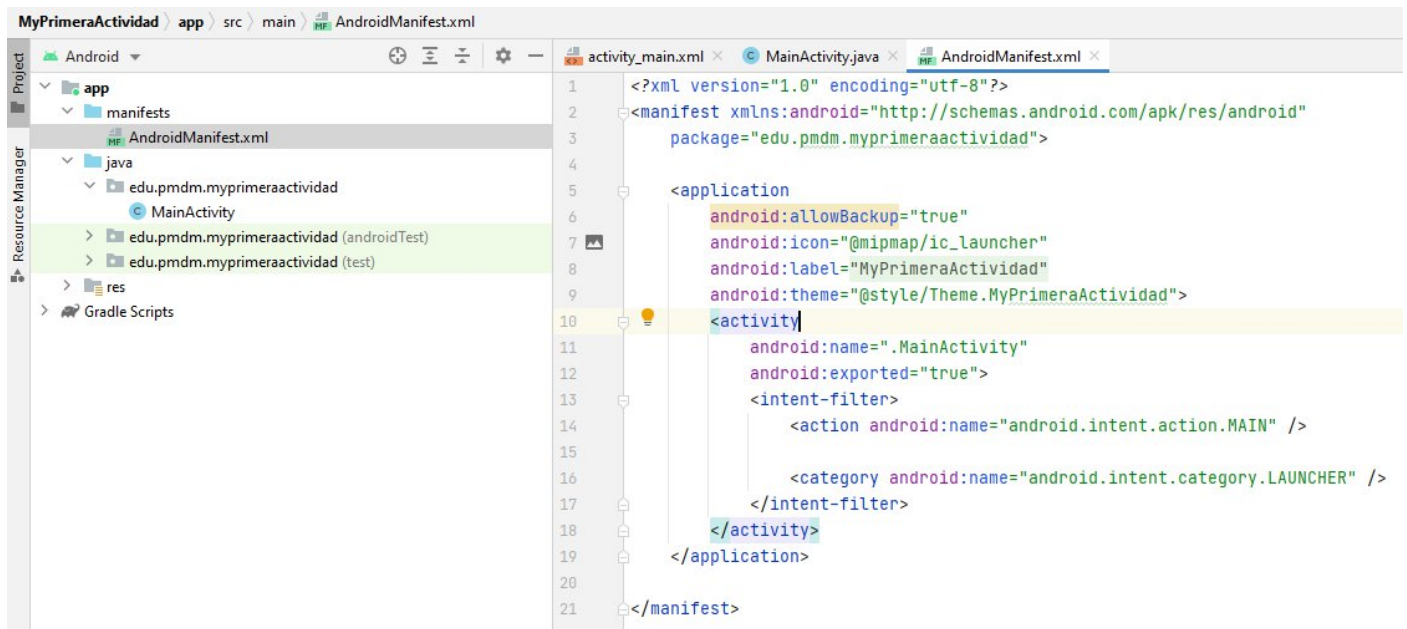
El intent activa el componente invocado, enlazando los componentes. Para actividades y servicios, el intent define la acción a realizar. Para los receptores de broadcast, el intent simplemente define el anuncio a realizar (por ejemplo, baja batería). El componente content provider, no es activado por intents.

2.1.3. El fichero de manifiesto (AndroidManifest.xml)

Antes de ejecutar una App, el sistema operativo lee un fichero xml llamado el *Fichero de manifiesto* o *Android Manifest File* que se encuentra ubicado en el directorio raíz de tu proyecto. En este fichero se declaran todos los componentes que forman parte de tu App. Además en este fichero se declaran:

- Los permisos que requiere una App.
- Mínimo nivel de API de Android requerido por tu App.
- Características hardware y software que necesita tu App (p.ej. bluetooth, cámara).
- Librerías que necesitan ser enlazadas a nuestra App. (p.ej. librerías de google maps).
- Otras cosas....

El formato del fichero de manifiesto es muy sencillo:



Como puedes ver en el ejemplo de la U.T.1, tu fichero AndroidManifest declara la actividad que creaste mediante el asistente de Android Studio mediante el uso de la etiqueta `<activity>`.

El atributo `android:name` establece el nombre de la actividad y el atributo `android:label` la etiqueta que se mostrará. Se debe declarar, de esta manera todos los componentes que vaya a usar tu App:

- `<activity>` componentes de tipo actividad
- `<service>` componentes de tipo servicio
- `<receiver>` componentes de tipo receptores de broadcast
- `<provider>` componentes de tipo proveedor.

Como hemos visto antes, se pueden utilizar intents para iniciar actividades, servicios y receptores de broadcast. Aunque es posible iniciar un componente simplemente escribiendo su nombre y utilizando la clase `Intent`, si lo declaramos en el archivo de manifiesto, el sistema podría seleccionar nuestra aplicación como una posible App que fuera capaz de responder a ese `Intent`.

2.1.4. La pila de software de android.

La pila de software es un reflejo gráfico de cómo organiza Android sus diferentes partes software. La primera capa, o capa de Aplicación, ilustra las Apps que se ejecutan en el SSOO. La segunda capa, la Application Framework es la que garantiza que las aplicaciones que se crean para Android siguen el esquema que dicta Android y no otro. La tercera capa, de librerías, ilustra todos los paquetes de librerías de las que podemos hacer uso para elaborar nuestra App. La

última, muestra lo que es el propio núcleo de Linux, con sus controladores hardware incluidos.

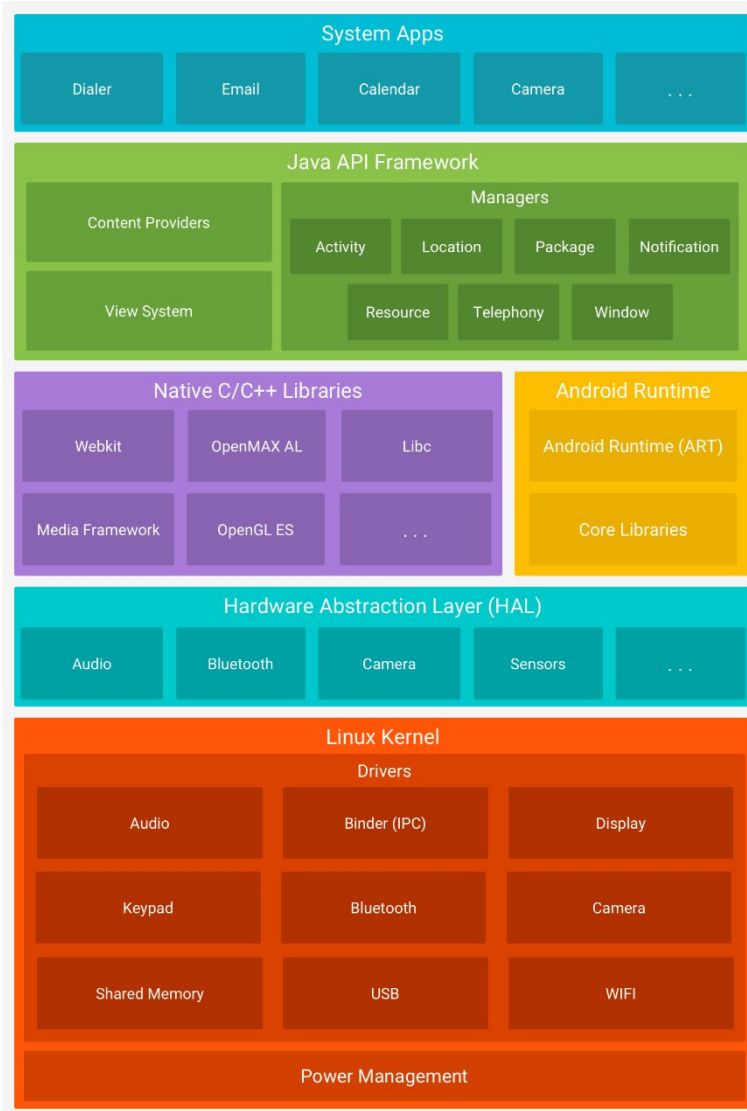


Figura: Android software stack (<https://developer.android.com/guide/platform>)

2.2 profundizando en el desarrollo: widgets para actividades

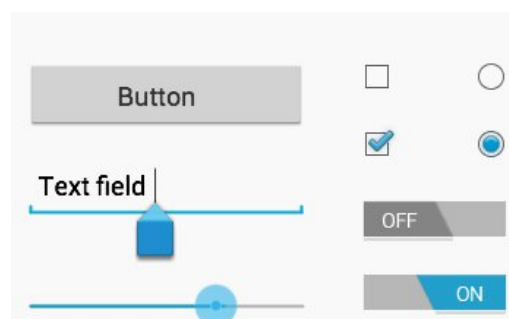
En esta sección se presentan los controles más populares en las Apps de Android. Los controles o widgets que vas a estudiar a continuación tienen muchísimas variantes, así que ármate de paciencia y procura probar todas y cada una de las características de cada Widget.

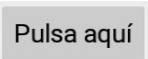







2.2.1. Los controles de entrada

En la unidad de trabajo 1 ya trabajaste con algunos tipos de controles de entrada, entre ellos los controles de tipo texto y de botón. Es el momento de profundizar en los controles de entrada de información para nuestra App.

¿Control o Widget? Digamos que Control es el término usado en los comienzos de la programación visual y que heredamos todos aquellos que empezamos allá por los años 90 con los primeros lenguajes visuales. Se refiere al componente visual mediante el cual el usuario de alguna manera proporciona información a nuestra App, ya sea a través de texto, de números, de casillas que se pueden marcar o desmarcar y de eventos controlados al tocar la pantalla. Android los llama widget, aunque como todo en informática tiene a tener varios nombres, es válido tanto uno como otro.

Controles Típicos: Aunque hay muchos tipos de widgets, aquí te mostramos los más típicos:



Tipo de Control	Descripción	Clases	Imagen
Botón	Un botón que puede ser “apretado” por el usuario para realizar una acción	Button	
Texto	El texto puede ser no editable (etiqueta o TextView*), o editable (Campo de Texto o EditText). Este último, se puede filtrar tipo y longitud del valor que introduce el usuario.	TextView, EditText, AutoCompleteTextView	
Casilla de verificación	Un campo que puede ser encendido o apagado por el usuario. Se presentan al usuario como opciones no mutuamente excluyentes entre sí	CheckBox	
Botón de Opción	Igual que los checkboxes, pero redondos y representan opciones mutuamente excluyentes	RadioButton, RadioGroup	
Switches	Representan una opción booleana, con valores de encendido/apagado	ToggleButton	
Spinners	En otros entornos llamados “Combo boxes” o listas desplegables. Permiten al usuario seleccionar un valor de entre varios	Spinner	
Selectores	Diálogos de usuario que permiten al usuario seleccionar un valor (generalmente fecha/hora) mediante flechas o con gestos con la mano.	DatePicker, TimePicker	
Imágenes	Se pueden utilizar imágenes para producir acciones al igual que con los botones.	ImageView, ImageButton	

*TextView se puede configurar para ser editable.

Otros tipos de controles importantes para tu aplicación:

Layouts: Son agrupaciones de controles para organizarlos en algún tipo de formación.

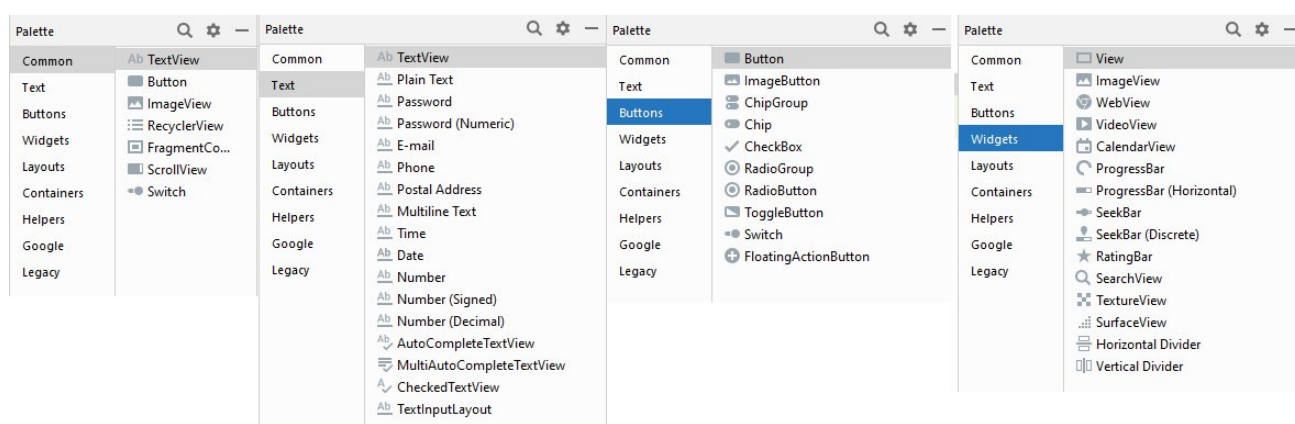
Tabuladores o Tabs: Son un tipo de contenedores que organizan la información por diversos criterios.

Menús y Submenús: Permiten seleccionar al usuario varias opciones (agrupadas) para producir diferentes acciones.

Diálogos de alertas: Permite al usuario responder a una pregunta espontánea de la aplicación.

Barras de Desplazamiento: Permite al usuario desplazar una parte de la interfaz de usuario para examinar una parte que no estaba visible.

ACTIVIDAD: EXAMINA el cuadro “Palette” dentro de Android estudio y localiza los posibles tipos de controles que has visto hasta ahora:



Casi todos estos widget tienen un componente común: Se pueden dibujar en una actividad. Todo lo que se puede dibujar en una actividad y por tanto “Ver” en una actividad hereda de la clase View de Android (<https://developer.android.com/reference/android/view/View>) .

Una vista es cualquier elemento representable en pantalla, con sus dimensiones de alto y ancho y una cantidad de métodos relaciones muy interesante que irás conociendo a lo largo del texto.

¿Recuerdas la funciones findViewById que has utilizado para localizar un componente en una actividad? Pues este método nos es único de la actividad. También está disponible en un tipo de vistas especiales llamados “contenedores de vistas”, Containers o ViewGroup.

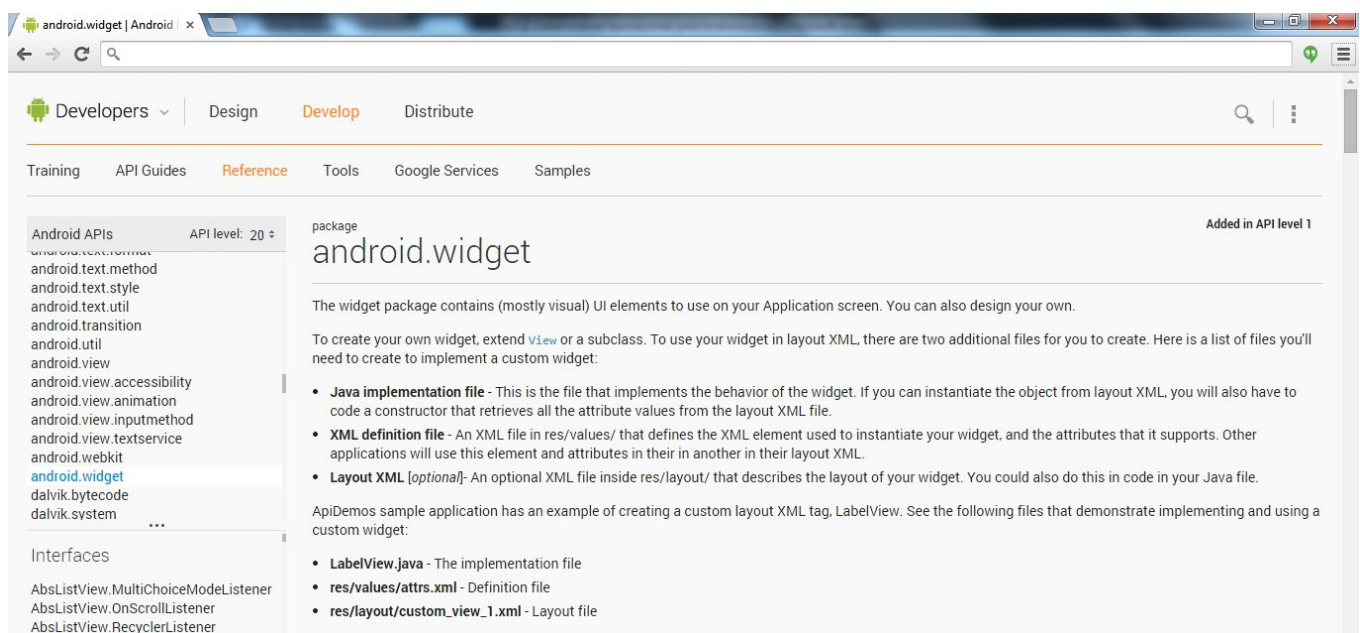
2.2.2 LAS API's de Android

A partir de ahora, todo lo que irás estudiando a continuación, tendrá como base lo expuesto en el propio manual de referencia para programadores de Android. Como no puede ser de otra manera, es la fuente oficial, y cualquier otra fuente, incluido este texto, debe ser secundario. Tu principal guía de referencia, donde viene todo (o al menos prácticamente todo) documentado, será la documentación de las API que puedes encontrar en los siguientes enlaces:

Guía de la API de Android	http://developer.android.com/guide/index.html
Tutoriales de entrenamiento	http://developer.android.com/training/index.html
Referencias de Paquetes	http://developer.android.com/reference/packages.html

La documentación de las APIs además de ser sencilla, es muy versátil, pudiendo seleccionar en todo momento la versión de la API que quieras consultar. Por ejemplo, para ver la documentación sobre los widgets que explicaremos a continuación puedes entrar en el enlace “android.widget” desde la referencia de los paquetes:

<http://developer.android.com/reference/android/widget/package-summary.html>

The screenshot shows the official Android developer website's API reference for the 'android.widget' package. The page is titled 'package android.widget' and notes it was 'Added in API level 1'. It explains that the package contains visual UI elements. To create a custom widget, one must extend 'View' or a subclass and create three files: a Java implementation file, an XML definition file, and an optional XML layout file. Examples of these files are listed at the bottom: 'LabelView.java', 'res/values/attrs.xml', and 'res/layout/custom_view_1.xml'. On the left, a sidebar lists various Android APIs, with 'android.widget' highlighted. The top navigation bar includes links for Developers, Design, Develop, and Distribute, along with a search icon.

2.2.3. La clase TextView

Es el widget más simple es la etiqueta, representada la clase TextView. Se utiliza para formar un texto que normalmente sirve para identificar visualmente otro componente que se sitúa justo al lado. Por ejemplo, el campo Posición en la Activity de la Tarea 1, tiene al lado una etiqueta con el texto “Posición”:



En Java, se crea una etiqueta con una instancia de la clase TextView, aunque en Android lo más normal es crearlas a través de los ficheros XML para la disposición (Layout) de los controles de la actividad, con la propiedad android:text para establecer el texto de la propia etiqueta.

Hay muchas propiedades de la clase TextView para formatear el texto de la etiqueta. Por ejemplo:

android: typeface permite cambiar el tipo de fuente (normal, sans, serif, monospace) para la etiqueta.

android:textStyle permite seleccionar el estilo de la letra (cursiva, negrita, o cursiva y negrita)

android:textColor permite seleccionar el color de la etiqueta en formato RGB hexadecimal.

Android:textSize permite especificar el tamaño de la fuente, por ejemplo 20dp (20 density-independent pixels)

Veamos un ejemplo de uso de propiedades del campo TextView:

```
<TextView
android:id="@+id/txtHelloWorld"
android:text="@string/hello_world"
android:typeface="monospace"
android:textStyle="italic"
android:textColor="#FF0000"
android:textSize="20dp"
android:layout_width="wrap_content"
android:layout_height="wrap_content"/>
```



Puedes ver todas las propiedades de la clase TextView y sus características en la API de Android:

<http://developer.android.com/reference/android/widget/TextView.html>

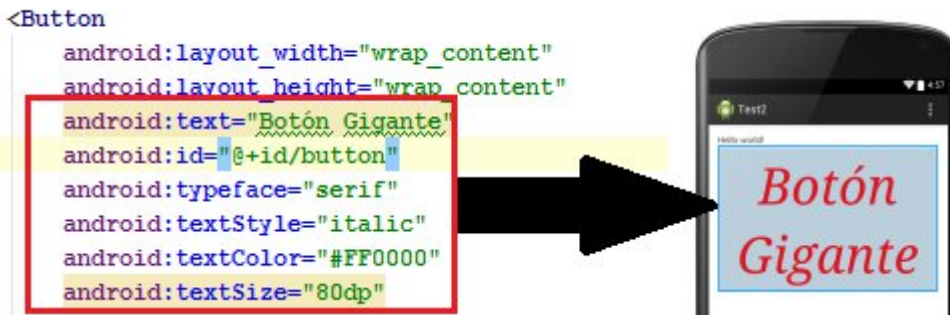
Pregunta: ¿No puedo cambiar en un TextView al tipo de fuente que yo quiera? Si, por supuesto, pero debes tener en cuenta que las fuentes incluidas por defecto en Android son Droid Sans (sans), Droid Sans Mono (monospace) y Droid Serif (serif). Cualquier otra opción debe ser cargada. Por ejemplo, para poder usar una fuente Arial, debes cargar el fichero arial.ttf en el directorio de Assets (herramientas) y cargarlo desde código:

```
TextView txtHelloWorld=findViewById(R.id.txtHelloWorld);
Typeface type = Typeface.createFromAsset(getAssets(),"arial.ttf");
txtHelloWorld.setTypeface(type);
```

Cuidado: Este tipo de ficheros son muy “pesados” y recuerda que estás programando un dispositivo con recursos limitados. Además, estas fuentes, en muchos casos requieren licencia para su uso.

2.2.4. La clase Button

Ya has trabado en el primer capítulo con la clase Button, curiosamente resulta que la clase Button es una subclase de la clase TextView, así que las propiedades que vimos en el punto anterior también son válidas para la clase Button:



También has visto que para responder a un evento de tipo Click, hay que implementar un Listener. Mediante una clase que implemente el tipo de Listener View.OnClickListener, será posible responder al evento de haber pulsado el botón. A partir de la versión 1.6 de Android, también es posible declarar la respuesta a este evento Click, en el fichero XML de la actividad, mediante la propiedad android:onClick, ahora ya **depreciado**. Se puede indicar el nombre de cualquier método público que reciba un parámetro de tipo View y que retorne void. Por ejemplo:

Supon que defines un método en tu Activity llamado "MeHicieronClick":

```
public void MeHicieronClick(View v){  
    System.out.println("DEBUG: Me hicieron Click!");  
}
```

Verás que al escribir en el fichero XML la propiedad onClick, el propio entorno de desarrollo te ofrece como posibilidad el método que has creado:



Una tercera posibilidad para responder al evento de Click, es en lugar de usar la propiedad del código XML android:onClick, codificar clases Anónimas (anonymous –inner classes), es decir, en el propio código de establecimiento del Listener onClick del botón, crear una clase sin nombre como argumento del método setOnClickListener de la siguiente forma:

```
Button btnGigante = (Button) findViewById(R.id.button);  
btnGigante.setOnClickListener(new View.OnClickListener() {  
    public void onClick(View v) {  
        System.out.println("Me hicieron click!");  
    }  
});
```

De igual manera, se puede definir un listener de forma sencilla utilizando una *expresión lambda*:

```
Button btnGigante = (Button) findViewById(R.id.button);  
btnGigante.setOnClickListener( view -> {System.out.println("Me hicieron click!");} );
```

Fíjate en la parte del código que está en gris, es la parte donde a través del operador new, se define un objeto de una clase anónima (sin nombre) derivada de la clase View.OnClickListener en tiempo de ejecución. Esta clase anónima tan sólo tiene un método llamado onClick que es el que responde al evento.

ACTIVIDAD: La documentación sobre cómo gestionar eventos es muy extensa. Te recomendamos que si tienes dudas y quieres refrescar los contenidos del módulo de programación visites estos enlaces de la documentación de java:

Cómo gestionar eventos: <http://docs.oracle.com/javase/tutorial/uiswing/events/generalrules.html>

Cómo programar escuchadores de eventos: <http://docs.oracle.com/javase/tutorial/uiswing/events/>

Expresiones lambda: <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>

Un botón es totalmente personalizable, por ejemplo, se pueden hacer con el cosas sencillas como deshabilitar el click durante un periodo de tiempo (hacerlo no clickable) o cosas más avanzadas como personalizar la forma del botón para que no rompa el estilo de la interfaz de tu App. Puedes leer más documentación sobre todas estas acciones en:

<http://developer.android.com/reference/android/widget/Button.html>

ACTIVIDAD:

Prueba la herramienta online <http://angrytools.com/android/button/> para personalizar de forma sencilla los botones de tu App, de paso, podrás observar cómo cambian las propiedades de los botones con los diferentes posibles parámetros

2.2.5. Las clases RadioGroup y RadioButton

En ocasiones habrás visto que ciertas Apps permiten al usuario la selección de un conjunto de opciones excluyentes entre sí, es decir, tan sólo puedes seleccionar una de ellas a la vez. Este tipo de acciones se implementan con el uso de las clases RadioGroup y RadioButton. Por decirlo de forma sencilla, los botones de tipo radio (RadioButton) se agrupan en un conjunto llamado RadioGroup. De esta manera, sólo una opción del grupo puede estar activada a la vez. Para crearlo, tan sólo tienes que añadir desde tu pantalla de diseño de la actividad, un RadioGroup y después arrastrar dentro del RadioGroup los RadioButton que quieras que se excluyan mutuamente.

Tanto los RadioGroup como los RadioButton los puedes encontrar en la sección “Buttons” de la paleta.



Puedes probar a crear en modo diseño un RadioGroup para seleccionar tus equipos de futbol favoritos. El código que genera el entorno de desarrollo es muy fácil de entender y de reproducir:

```
<RadioGroup
    android:id="@+id/radioGroup"
    ...
    <!--Propiedades del radio group-->
    ... >

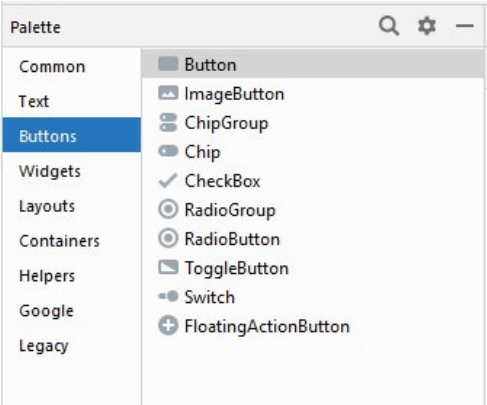
<RadioButton
    android:text="Talavera C.F."
    android:id="@+id/rbTalavera"
    android:checked="false" />

<RadioButton
    android:text="Gimástico de Alcazar"
    android:id="@+id/rbAlcazaar"
    android:checked="false" />

<RadioButton
    android:text="Albacete Balompié"
    android:id="@+id/rbAlbacete"
    android:checked="false" />

<RadioButton
    android:text="Otros equipos (Atlético de Madrid,R.Madrid, Barsa)"
    android:id="@+id/rbOtros"
    android:checked="false" />

</RadioGroup>
```



RadioGroup es un “contenedor” de RadioButtons. Establece las propiedades de anchura a `match_parent` para que se los objetos se adapten al tamaño de la pantalla y el alto al valor `wrap_content` para que se adapte al alto del contenido.

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
```

Habrás visto que el contenedor RadioGroup lleva dentro cada RadioButton. Como propiedades importantes de los RadioButton destacar, el id, que todo widget debe llevar, el texto y si está activado o no (`checked="true"` o `checked="false"`)

Desde código Java también podemos establecer estas propiedades, además, podemos utilizar, entre otros, los siguientes métodos:

Clase	Método	Acción
RadioGroup	<code>void check(int id)</code>	Activa el RadioButton con identificador id
RadioGroup	<code>void clearCheck()</code>	“Limpia”, es decir, quita la selección dejando todos los RadioButton desactivados
RadioGroup	<code>int getCheckedRadioButtonId()</code>	Retorno el Id del RadioButton seleccionado
RadioButton	<code>void toggle()</code>	Cambia el estado de activación del RadioButton

Eventos para RadioGroup y RadioButton

El Click en un RadioGroup y en un RadioButton, se pueden capturar de forma idéntica a un botón, pero además:

Se pueden capturar los eventos de cambio de selección en un RadioGroup mediante el método:

```
void setOnCheckedChangeListener (RadioGroup.OnCheckedChangeListener listener)
```

Se hace igual que con el botón desde código, pero no a través de propiedades en el archivo XML de la actividad.

CASO PRÁCTICO: Crear una App llamada “RadioButtonsExample” que de al usuario 4 opciones para elegir su equipo de futbol favorito. La aplicación responderá con mensajes a cada elección del usuario.

Podemos capturar el cambio en la selección por parte del usuario de la siguiente manera:

Paso 1. Implementar la interfaz `RadioGroup.OnCheckedChangeListener`

```
public class MainActivity extends AppCompatActivity implements
    RadioGroup.OnCheckedChangeListener{
    ...
}
```

Paso 2. Registrar el listener

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    RadioGroup r=(RadioGroup) findViewById(R.id.radioGroup);
    r.setOnCheckedChangeListener(this);
}
```

Vemos cómo se invoca el método `setOnCheckedChangeListener` recibiendo como objeto *this*, es decir, la propia instancia en ejecución que implementa la interfaz `OnCheckedChangeListener`

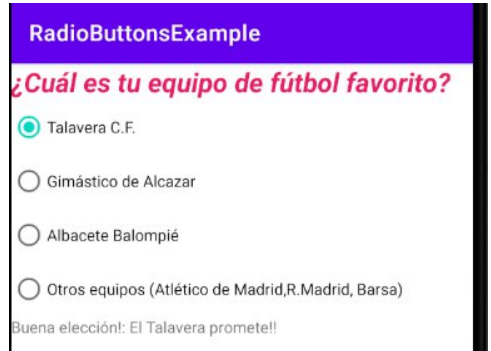
Paso 3. Codificarlo

```
public void onCheckedChanged(RadioGroup group, int checkedId) {
    TextView t=(TextView) findViewById(R.id.txtEstado);
    switch (checkedId) {
        case R.id.rbTalavera: //Talavera
            t.setText("Buena elección!: El Talavera promete!!");
            break;
        case R.id.rbAlcazar: //Alcazar
            t.setText("Gran equipo la gimnástica!!");
            break;
        case R.id.rbAlbacete: //Albacete
            t.setText("El Albacete no es el mismo desde que se fué Iniesta");
            break;
        case R.id.rbOtros: //Otros
            t.setText("El dinero no lo es todo....");
            break;
    }
}
```

CAUTION: Desde la versión 8.0 del Plugin Gradle de Android, los `R.id.*` no son constantes, por lo que no se pueden utilizar en una sentencia `switch case`. Tienes que usar sentencias `if`:

“Resource IDs will be non-final by default in Android Gradle Plugin version 8.0, avoid using them in switch case statements [NonConstantResourceId]”

```
public void onCheckedChanged(RadioGroup group, int checkedId) {
    TextView t=(TextView) findViewById(R.id.txtEstado);
    if (checkedId == R.id.rbTalavera) { //Talavera
        t.setText("Buena elección!: El Talavera promete!!");
    } else if (checkedId == R.id.rbAlcazar) { //Alcazar
        t.setText("Gran equipo la gimnástica!!");
    } else if (checkedId == R.id.rbAlbacete) { //Albacete
        t.setText("El Albacete no es el mismo desde que se fué Iniesta");
    } else if (checkedId == R.id.rbOtros) { //Otros
        t.setText("El dinero no lo es todo....");
    }
}
```



ACTIVIDAD: Mira la extensa documentación en estos enlaces la documentación de `RadioGroup` y `RadioButton`:

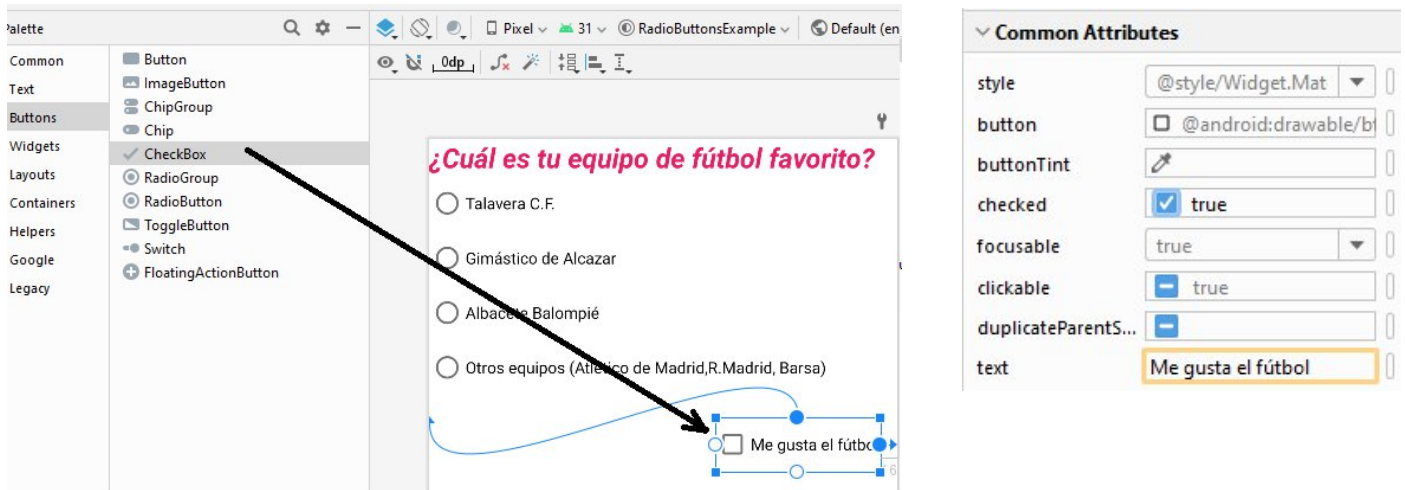
`RadioButton`: <http://developer.android.com/reference/android/widget/RadioButton.html>

`RadioGroup`: <http://developer.android.com/reference/android/widget/RadioGroup.html>

2.2.6. La clase Checkbox

Seguro que alguna vez has tenido que marcar una casilla del tipo “He leído y comprendido las condiciones de licencia de uso del software XXX” para poder disfrutar de algún programa, videojuego o sistema operativo. El componente que se ha utilizado para poder marcar que has leído las condiciones de licencia (aunque no las hayas leído), es un Checkbox o casilla de verificación. Tiene dos estados, marcado (checked) o desmarcado (unchecked); en programación: `setChecked(true)` o `setChecked(false)`.

En Android Studio, incluir un checkbox en tu actividad es de lo más sencillo: Arrastra desde la paleta, ponle identificador y escribe el texto que quieras que Aparezca.



Además del método `setChecked` para establecer el estado, tiene otro método para consultarlo, `isChecked` que devuelve cierto (`true`), si el usuario marcó la casilla y falso (`false`) en caso contrario. Otro tercer método para alterar el estado es `toggle()`, que cambia el estado que tenga en ese momento la checkbox. `CheckBox` también es un descendiente de `TextView`, por lo que podrás poner todas las características de texto de la misma manera que lo haces con `TextView`.

El evento de cambio de estado se puede capturar de manera idéntica a cómo describimos en la sección anterior el cambio de estado de un `RadioGroup`, mediante el `onCheckedChangeListener` que obliga a implementar la interfaz `CheckBox.OnCheckedChangeListener`.

Caso Práctico: Añadir una casilla a la aplicación del apartado anterior para que el usuario pueda seleccionar si le gusta o no el fútbol:

Primero: Agregar a la lista de interfaces que implementa la actividad `CheckBox.OnCheckedChangeListener`.

```
public class MainActivity extends AppCompatActivity implements
    RadioGroup.OnCheckedChangeListener, CheckBox.OnCheckedChangeListener{
    ...
}
```

Segundo: Registramos el listener

```
CheckBox miChk=(CheckBox)findViewById(R.id.chkFutbol);
miChk.setOnCheckedChangeListener(this);
```

Tercero: Crear el método que hace de listener

```
public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
    TextView t=(TextView)findViewById(R.id.txtEstado);
    if(isChecked)
        t.setText("Te gusta el fútbol!!");
    else
        t.setText("No te gusta el fútbol!?!?!");
}
```


¿y qué pasa si..., como es el caso de esta App, tengo que implementar dos interfaces, una para RadioGroup y otra para Checkbox? ¿No habrá conflictos entre los dos métodos que hay que implementar? ¿Cómo sabe Android a qué método invocar cuando se produzca un evento de cambio de estado si los dos métodos se llaman `OnCheckedChangeListener`?

La respuesta es que **no**. Gracias al polimorfismo estático, Java detecta qué método hay que ejecutar dependiendo de los parámetros de entrada.

ACTIVIDAD: DESCARGA EL CÓDIGO DE LA APP `RadioButtonsExample` de los recursos del tema: **"ficheros_tema2/RadioButtonsExample.zip" y examínalo con detalle.**

2.2.7. Las clases `ToggleButton` y `Switch`

Los `ToggleButton` y `Switch` son widgets muy parecidos a `CheckBox`. También son widgets con dos estados, encendido y apagado (on y off). La única diferencia con un `CheckBox` es el aspecto gráfico, que es diferente. El `ToggleButton` no tiene un texto asociado y el `Switch` si, pero en ambos se puede cambiar el título de los dos estados mediante la propiedad `android:textOn` y `android:textOff` en el fichero XML.

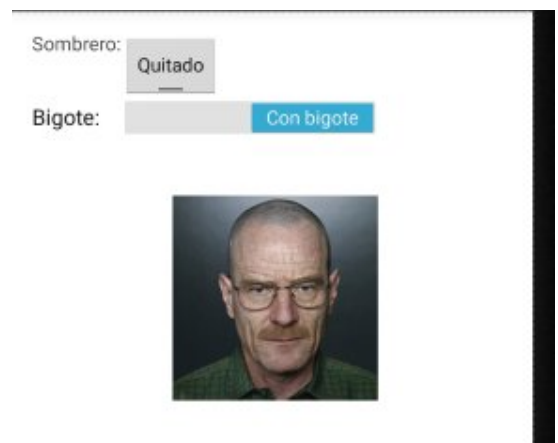


(imagen de <http://developer.android.com/guide/topics/ui/controls/togglebutton.html>)

Para mantener activado o desactivado cualquiera de los dos componentes se utiliza la propiedad `Checked`, que funciona exactamente igual que en una `CheckBox`. En resumen, para obtener un resultado gráfico como el de la imagen, tenemos el siguiente código en XML:

```
<ToggleButton
    android:text="Sombrero"
    android:id="@+id/toggleButton"
    android:textOn="Puesto"
    android:textOff="Quitado"
    android:checked="false"
/>

<Switch
    android:text="Bigote:"
    android:checked="true"
    android:id="@+id/switch1"
    android:textOn="Con bigote"
    android:textOff="Sin bigote"
/>
```

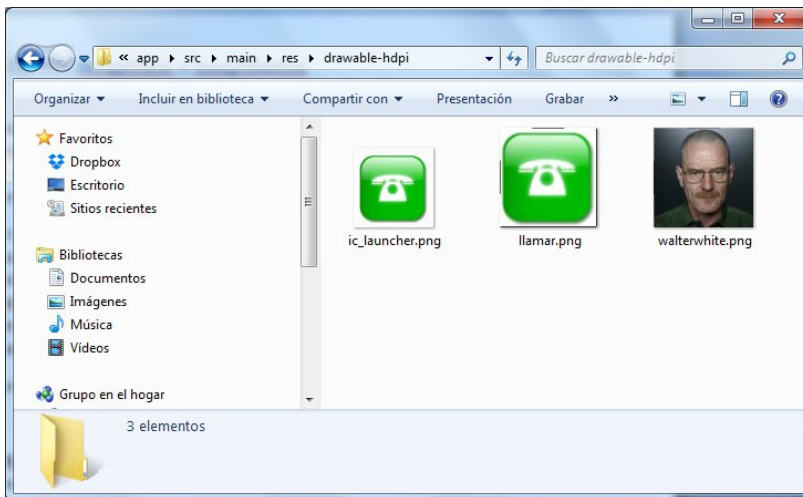


De igual manera se puede capturar el evento `OnCheckedChangeListener` como en `RadioGroup` y `CheckBox`.

2.2.8. Las clases ImageView e ImageButton

Estas clases son las análogas de TextView y Button, pero con imágenes. Cada widget toma un atributo android:src en el fichero XML de la actividad para especificar qué imagen usar mediante una referencia a un **recurso dibujable** (@drawable). Son análogas en el sentido de que tienen la misma funcionalidad, pero en lugar de mostrar texto, muestran una imagen.

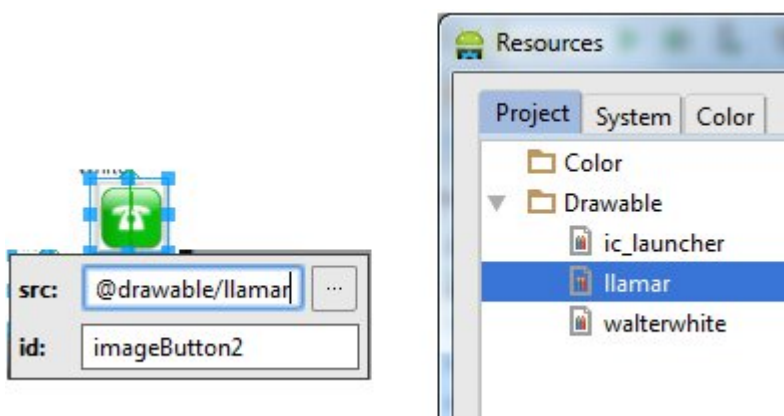
Tanto la clase ImageView como la clase ImageButton tienen una propiedad llamada android:src, que referencia al identificador del recurso de la imagen dibujable. Por tanto, antes de añadir un ImageView o un ImageButton, debemos añadir a las carpetas “src->main->res->drawable*****” de nuestro proyecto los ficheros de las imágenes que queremos utilizar. Ten en cuenta que Android recomienda usar ficheros con formato png, aunque también admite bmp y jpgs. Cada imagen debe tener un identificador para poder ser referenciadas. Este identificador será un número entero accesible a través de la clase R, mediante la propiedad **R.drawable.identificador**.



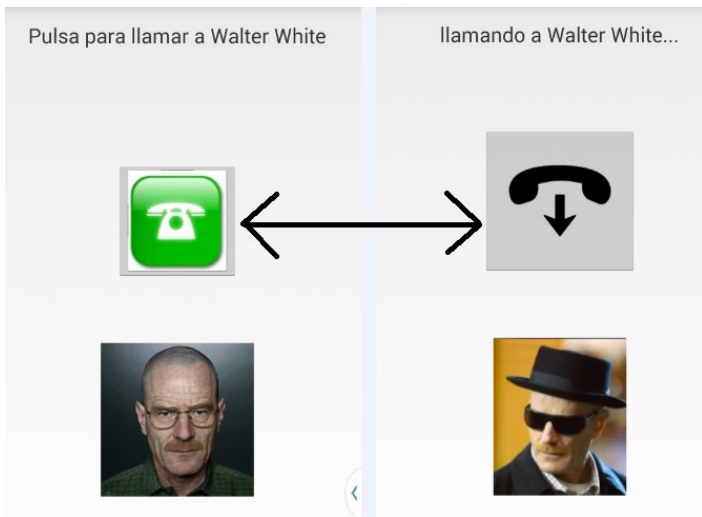
Tienes que tener en cuenta que hay varias carpetas “drawable”, cada una con los diferentes tipos de pantalla para los que tu App puede funcionar. Así por ejemplo, la carpeta drawable-hdpi (high) es una densidad de pantalla de aproximadamente 240dpi (puntos por pulgada). Lee más en:

http://developer.android.com/guide/practices/screens_support.html

De esta manera, para añadir un botón con imagen a tu App, tan solo tienes que arrastrar desde la paleta, poner un id, y escoger el recurso que acabas de copiar.



CASO PRÁCTICO: Crea una App que muestre un botón con imagen y una imagen simple. Al pulsar el botón se cambiará la imagen tanto del ImageView como del ImageButton. Debe quedarte más o menos así:



Para cambiar la imagen de un ImageView o de un ImageButton puedes ejecutar el siguiente código:

```
ImageButton b=findViewById(R.id.imageButton2);
ImageView ww=findViewById(R.id.imageView);

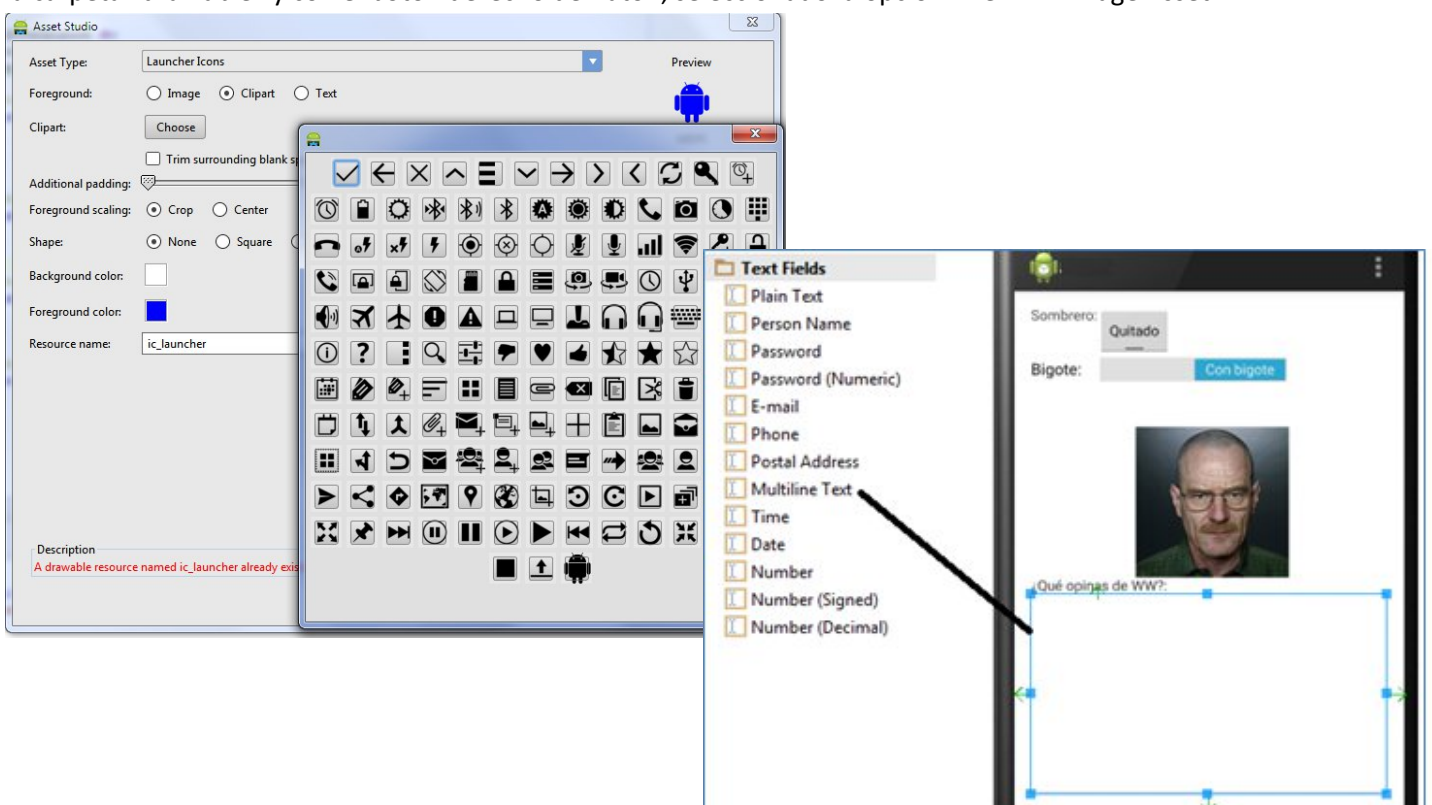
ww.setImageResource(R.drawable.walterwhitesombrero);
b.setImageResource(R.drawable.colgar);
```

ACTIVIDAD: Puedes utilizar la siguiente utilidad online (Android Asset Studio) para generar imágenes para todas las densidades de pantalla.

<http://romannurik.github.io/AndroidAssetStudio/nine-patches.html>

ACTIVIDAD: DESCARGA EL CÓDIGO DE LA APP ImageViews de los recursos del tema: “ficheros_tema 2/ImageViews.zip” y examínala con detalle.

Puedes escoger de la galería de ClipArt de Android Studio Image Asset multitud de recursos dibujables pulsando en la carpeta “drawable” y con el botón derecho del ratón, seleccionando la opción “New”->“Image Asset”:



2.2.9. La clase EditText

La clase EditText representa el widget más popular junto con las etiquetas y los botones. Es una subclase de la clase TextView y permite, por ejemplo, seleccionar el tipo de teclado a la hora de introducir contenido en la caja de texto:

android:inputType, que permite seleccionar entre varios tipos de teclado que se mostrarán cuando se esté introduciendo texto: "text", para textos normales, "textEmailAddress" para incluir el carácter @, "textUri" para incluir el carácter /, "number" para mostrar un teclado numérico normal y "phone" para obtener el teclado para marcación telefónica:

android:inputType también te permite controlar ciertos comportamientos, como por ejemplo que al terminar de escribir una palabra se autocorrija con el diccionario o que los caracteres aparezcan convertidos en puntos (bueno para introducir passwords).



android:inputType="phone"

Android Studio incorpora en la paleta de widgets un montón de campos de texto distintos para que no tengas que enredar mucho con el XML:

ACTIVIDAD: Puedes leer más sobre el comportamiento de la entrada de datos sobre campos de texto en el manual de Android:

<http://developer.android.com/guide/topics/ui/controls/text.html>

Asociar una acción de finalización al campo de texto: Es posible añadir un botón al teclado que aparece cuando se está editando un texto para que el usuario indique que se ha finalizado la edición, y, de esta manera desencadenar una acción. Por ejemplo "actionSend" (enviar) o "actionSearch" (buscar). Si no se especifica esta propiedad, por defecto se establece "actionNone" (ninguna).

Para hacer esto se utiliza la propiedad android:imeOptions en el fichero XML. Por ejemplo, para agregar un botón llamado "Enviar" al campo de texto:

android:imeOptions="actionSend"

Para responder a estos eventos, se puede programar un evento EditorAction mediante el método setOnEditorActionListener() de la siguiente forma:

```
EditText editText = (EditText) findViewById(R.id.editText);
editText.setOnEditorActionListener(new TextView.OnEditorActionListener() {
    public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {
        boolean manejada=false;
        if (actionId == EditorInfo.IME_ACTION_SEND) {
            System.out.println("acción enviada!");
            manejada = true;
        }
        return manejada;
    }
});
```

El método onEditorAction recibe como parámetros el campo de texto TextView t (recuerda que EditText es una subclase de TextView), el identificador de la acción (se puede consultar con la colección de enteros EditorInfo (IME_ACTION_SEND representa la acción de enviar) y el evento de tecla KeyEvent que en estos casos será null. Hay que devolver si hemos manejado o no el tipo de acción que se nos ha comunicado, en este caso, retornaremos cierto si se invocó el método para la acción de terminar (IME_ACTION_SEND)

Capturando los cambios en el texto: Existe una forma de controlar programáticamente cualquier interacción del usuario con un campo de texto. Se puede, por ejemplo, programar un método que responda al cambio del texto en una determinada posición o realizar algo justo antes de que el usuario se ponga a cambiar el texto, o incluso después de cambiar el texto. Para hacer este tipo de programación, se requiere un objeto que implemente la interfaz `TextWatcher`, y registrarla en el campo de texto mediante el método:

```
addTextChangedListener(TextWatcher watcher);
```

La interfaz `TextWatcher` tiene tres métodos:

- `public void beforeTextChanged(CharSequence s, int start, int count, int after)`

Este método te avisa de que en la cadena `s`, los `count` caracteres empezando en `start`, están a punto de ser reemplazados por nuevo texto con longitud `after`

- `public void onTextChanged(CharSequence s, int start, int before, int count)`

Este método te avisa de que en la cadena `s`, se han reemplazado `count` caracteres comenzando en `start` y han reemplazado `before` número caracteres

- `public void afterTextChanged(Editable s)`

Se ejecuta después de haberse cambiado el texto. Este método te avisa de que, en algún punto de la cadena `s`, el texto ha cambiado.

Prueba este ejemplo para examinar cómo se ejecutan los distintos métodos informándonos de cuándo se están cambiando los caracteres del campo de texto:

```
editText.addTextChangedListener(new TextWatcher() {
    //Este método te avisa de que en la cadena s,
    //se han reemplazado count caracteres
    //comenzando en start y han reemplazado before número caracteres
    public void onTextChanged(CharSequence s, int start,
                               int before, int count){
        Log.d(TAG, "Se han reemplazado de '"+s+"' "+count
                + " caracteres a partir de "+start+
                " que antes ocupaban "+before+" posiciones");
    }

    //Este método te avisa de que en la cadena s,
    // los count caracteres empezando en start,
    // están a punto de ser reemplazados por nuevo texto con longitud after
    public void beforeTextChanged(CharSequence s, int start,
                                   int count, int after){
        Log.d(TAG, "Se van a reemplazar de '"+s+"' "+count+
                " caracteres a partir de "+start+
                " por texto con longitud "+after);
    }

    //Este método te avisa de que, en algún
    //punto de la cadena s, el texto ha cambiado.
    public void afterTextChanged(Editable s){
        Log.d(TAG, "Tu texto tiene "+s.length()+" caracteres");
    }
});
```

ACTIVIDAD: PUEDES DESCARGAR EL CÓDIGO DE LA APP Test4 de los recursos del tema: “ficheros_tema 2/TextWatcher.zip”

2.3. Internacionalización de aplicaciones mediante el archivo string.xml

¿Qué son los recursos (resources)?

Los recursos son los datos, imágenes, sonidos, animaciones, layouts y otros datos estáticos que tu App puede utilizar. Una App puede incluir diversos conjuntos de recursos que tu App puede utilizar para funcionar con diversas configuraciones.

Para acceder a la carpeta de recursos de tu proyecto tiene la constante **R**. y para acceder a la carpeta de recursos del propio Android puedes usar la constante R del paquete Android: **android.R**

Es una muy buena práctica mantener todas las cadenas de caracteres con texto para etiquetas, botones y en general cualquier elemento de la interfaz de usuario, fuera del código fuente de la App. Hasta este momento, siempre hemos intentado en todos los ejemplos que para ponerle un texto a un control utilices lo que hemos llamado recurso. Este recurso, generalmente está almacenado en un archivo XML llamado res/values/strings.xml que depende del proyecto.

Este fichero deberá contener todas las cadenas de caracteres en el idioma que consideres que será el de uso mayoritario. Esto es lo que se llamará el RECURSO POR DEFECTO (Default resource).

Esto es también aplicable al resto de recursos de tu App, por ejemplo los ficheros gráficos “dibujables (drawable)” ubicados en res/drawable, animaciones, layouts, etc. cuando deseas que tu dispositivo funcione con varias configuraciones hardware, por ejemplo, densidad y tamaño de pantalla.

Ser muy disciplinado con esta buena práctica te permitirá, de forma muy fácil, internacionalizar tu App, es decir, seleccionar el idioma del usuario dependiendo de la zona geográfica en la que se encuentre o del idioma que tenga personalizado en su dispositivo móvil.

Incluso si no vas a internacionalizar tu App, es mucho más fácil corregir los errores (faltas ortográficas y errores tipográficos) si tienes todos tus textos almacenados fuera del código.

Ejemplo de strings.xml

```
<resources>
    <string name="voldemort">Aquel cuyo nombre no debe ser nombrado</string>
    <string name="spiderman">El hombre araña</string>
</resources>
```

El parámetro name identifica al recurso y entre las etiquetas va el texto que describe el recurso.

Desde el código, puedes referenciar estos valores desde el diseño de una actividad mediante @string/recurso, por ejemplo @string/voldemort o desde código invocando a la función getString (IDRecurso) pasándole como parámetro del identificador del recurso. Por ejemplo:

```
String s=getString(R.string.voldemort);
```

O, si no estás dentro de un activity:


```
String s= getResources().getString(R.string.voldemort);
```

2.3.1. Creación de aplicaciones multiidioma

Para poder utilizar distintos idiomas en tu App, además del recurso por defecto, necesitas RECURSOS ALTERNATIVOS (alternative resources). Una Aplicación puede especificar múltiples directorios /res/<cualificadores> donde un cualificador especifica un lenguaje o una combinación de lenguaje-región.

Cuando programes....

Crea un conjunto de recursos por defecto, y diferentes alternativas para ejecutarse con diferentes localizaciones

Cuando un usuario ejecuta tu App

Android selecciona qué recursos cargar dependiendo de su configuración

Por ejemplo, supón que el lenguaje principal de tu App es el español. Pues en la carpeta res/values tendrás todos los archivos con las cadenas de caracteres en idioma español, y puedes generar además las siguientes carpetas:

res/values-fr -> Contendrá las cadenas de caracteres traducidas al francés.

res/values-en -> Contendrá las cadenas de caracteres traducidas al inglés.

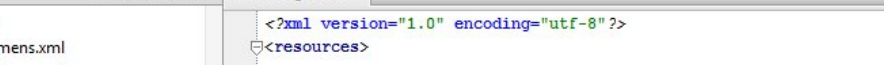
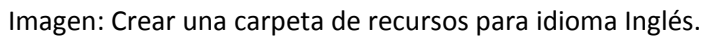
De esta manera, si el dispositivo está configurado para usar inglés, Android cargará el archivo correspondiente a los valores en inglés.

CONSEJO: Diseña tu App para funcionar con cualquier dispositivo. El dispositivo puede tener una configuración software o hardware que puede que no conozcas, o simplemente que no esperas. Prográmala para que “falle” elegantemente sin importar en qué dispositivo se está ejecutando

CASO PRÁCTICO: Crea un App con una actividad con capacidad de ejecución con idioma Español y con idiomas alternativos en Inglés y Alemán. Consistirá en un simple formulario para rellenar datos personales para el registro de un usuario. El formulario tendrá esta pinta en los tres idiomas:

The image displays three side-by-side screenshots of an Android application titled 'Internacionalización', 'Internacionalization', and 'Internacjonalization' respectively. Each screen shows a registration form with the following fields: 'Nombre:' (Spanish), 'Vorname:' (German), 'Name:' (English); 'Apellidos:' (Spanish), 'Nachname:' (German), 'Surname:' (English); 'Email:'; and 'Clave:' (Spanish), 'Passwort:' (German), 'Password:' (English). The status bars at the top show different times: 14:13, 13:03, and 2:15.

New Android resource file



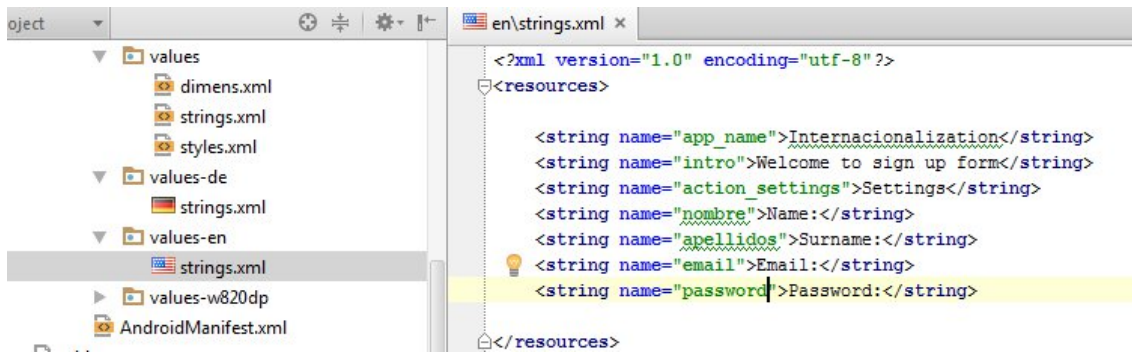
The screenshot shows the IDE's interface with the 'strings.xml' file open in the editor. The file is located in the 'resources' directory. The XML content is as follows:

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

    <string name="app_name">Internacionalización</string>
    <string name="intro">Bienvenido al formulario de registro!</string>
    <string name="action_settings">Configuración</string>
    <string name="nombre">Nombre:</string>
    <string name="apellidos">Apellidos:</string>
    <string name="email">Email:</string>
    <string name="password">Clave:</string>

</resources>
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Internacionalización</string>
    <string name="intro">Bienvenido al formulario de registro!</string>
    <string name="action_settings">Configuración</string>
    <string name="nombre">Nombre:</string>
    <string name="apellidos">Apellidos:</string>
    <string name="email">Email:</string>
    <string name="password">Clave:</string>
</resources>
```

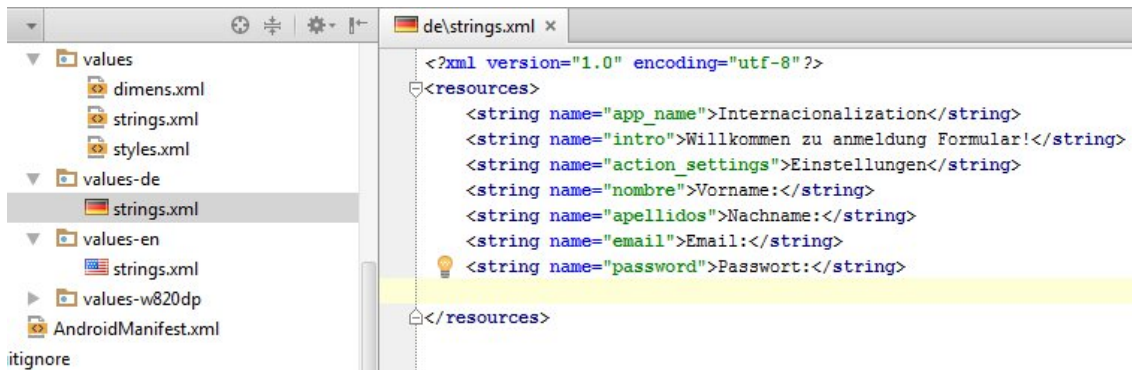


strings.xml de la carpeta res/values-en ->Cadenas de caracteres en Inglés

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Internacionalization</string>
    <string name="intro">Welcome to sign up form</string>
    <string name="action_settings">Settings</string>
    <string name="nombre">Name:</string>
    <string name="apellidos">Surname:</string>
    <string name="email">Email:</string>
    <string name="password">Password:</string>
</resources>

```



strings.xml de la carpeta res/values-de -> Cadenas de caracteres en Alemán

```

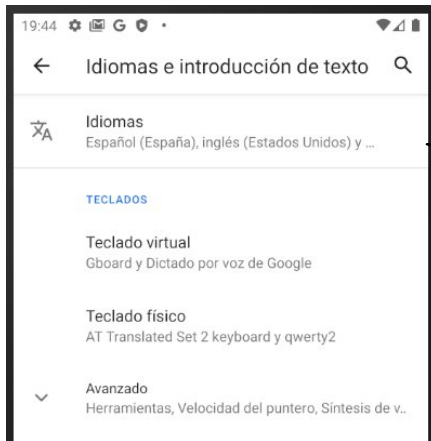
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Internacionalization</string>
    <string name="intro">Willkommen zu anmeldung Formular!</string>
    <string name="action_settings">Einstellungen</string>
    <string name="nombre">Vorname:</string>
    <string name="apellidos">Nachname:</string>
    <string name="email">Email:</string>
    <string name="password">Passwort:</string>
</resources>

```

ACTIVIDAD: Puedes aprender más sobre cómo soportar varios sistemas (pantalla, idioma y plataforma) en la dirección: <http://developer.android.com/training/basics/supporting-devices/index.html>

2.3.2. Probando la internacionalización en el emulador

Se puede cambiar la configuración del emulador mediante la pestaña de aplicaciones:



O directamente desde la Shell del sistema operativo:

```

CA: Administrador: C:\Windows\system32\cmd.exe - adb -e shell

D:\Android Studio Tools\sdk\platform-tools>adb -e shell
root@generic_x86:/ # setprop persist.sys.language de;
setprop persist.sys.language de;
root@generic_x86:/ # setprop persist.sys.country DE;
setprop persist.sys.country DE;
root@generic_x86:/ # stop;sleep 5;start
stop;sleep 5;start
  
```

Si te vas a la Shell del sistema operativo y posicionado en la carpeta sdk->platform-tools de la carpeta donde has instalado Android y ejecutas Android Debug Bridge mediante el comando `adb -e shell` tendrás acceso a la consola del emulador.

Para cambiar de lenguaje y de país, escribe en la consola:

```

setprop persist.sys.language <lenguaje>;
setprop persist.sys.country <país>;
stop;sleep 5;start
  
```

donde `lenguaje` es el código del lenguaje que quieras probar y `país` el país donde el usuario se supone que está. Por ejemplo, para Alemania sería:

```

setprop persist.sys.language de;
setprop persist.sys.country DE;
stop;sleep 5;start
  
```

ACTIVIDAD: DESCARGA EL CÓDIGO DE LA APP Internacionalización de los recursos del tema: “`ficheros_tema2/internacionalizacion.zip`” y examínala con detalle

2.4. Layouts y contenedores

Los contenedores o *containers*, permiten organizar un conjunto o colección de widgets en una estructura específica definida como tú quieras. Por ejemplo, si quieres organizar un conjunto de campos para formar un formulario necesitarás un contenedor. Siempre que necesites utilizar múltiples widgets necesitarás un contenedor para alojarlos y de esta manera tener un elemento de referencia o *raíz* (*root*).

En el módulo de programación estudiaste cómo Java organiza los controles en un contenedor y usaba gestores de distribución de esos controles (layouts managers) para poder usarlos (FlowLayout, BoxLayout, etc.). Esta misma filosofía la usa Android para la disposición y organización de los widgets en la pantalla.

Los layouts se pueden manejar de dos formas:

- Declarando los ficheros en XML: Como lo has hecho hasta con las vistas de diseño en Android studio.

- Instanciando objetos programáticamente en tiempo de ejecución.

La ventaja de declarar los elementos de tu layout a través de la definición en XML es que se separa la interfaz de usuario (IU) de la lógica de tu programa que controla el comportamiento de los elementos de la IU. Haciéndolo de este modo las descripciones de tu interfaz son externas al código de tu aplicación, pudiendo cambiar el diseño de tu interfaz sin tener que tocar código y sin tener que recompilar. Esto te proporciona **FLEXIBILIDAD**. Nos centraremos por tanto, en explicarte cómo declarar los elementos de los layout en XML.

Los `LinearLayout`, `ConstraintLayout` y `TableLayout` son contenedores que gestionan la distribución de los controles de una determinada manera en una pantalla cuando se trabaja con Android.

En otras palabras, el contenedor agrupa controles (también llamados **hijos**) y el layout los ordena/distribuye dentro del contenedor. De esta manera se crean organizaciones jerárquicas de controles, donde el elemento **padre** es el contenedor y los componentes que hay dentro son los **hijos**.

Hasta ahora en los ejemplos que has visto, siempre has utilizado el `ConstraintLayout`, tremendamente flexible y que crea Android Studio por defecto. Este `ConstraintLayout` utiliza un modelo basado en reglas (*rule based model*), mientras que `LinearLayout` ofrece un modelo de cajas, *box model* y `TableLayout` que proporciona un modelo de celdas (*grid model*).

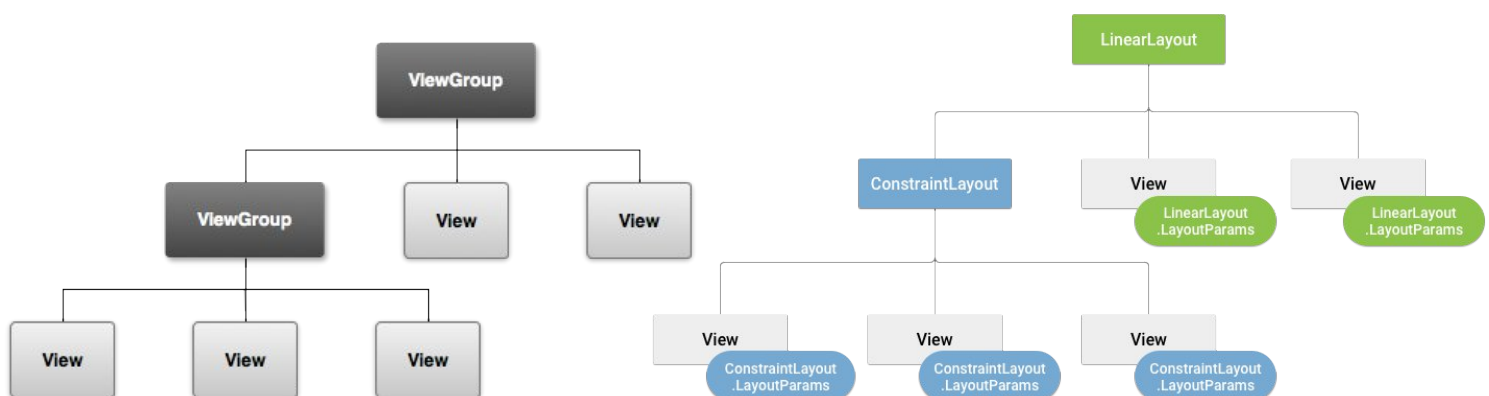
Lo bueno de los contenedores es que puedes anidarlos unos dentro de otros para hacer más flexible y potente tu interfaz de usuario.

2.4.1. Propiedades de los layouts

Los Layout son subclases de una clase más abstracta llamada `ViewGroup`, que a su vez, es una subclase de `View`. Todos los widgets son Views, y por tanto implementan métodos y propiedades comunes para todos los elementos gráficos de tu IU.

```
java.lang.Object
↳ android.view.View
    ↳ android.view.ViewGroup
        ↳ androidx.constraintlayout.widget.ConstraintLayout
```

Ejemplo de la documentación, jerarquía de clases para `ConstraintLayout`:



Cada clase `ViewGroup` implementa una clase anidada que hereda de `ViewGroup.LayoutParams`. Esta clase implementa propiedades importantes que definen la posición y el tamaño de los elementos hijos. Como puedes ver en la siguiente imagen, cada descendiente de un Layout define estas propiedades.

Por ejemplo, si consultas la documentación de la clase `ViewGroup.LayoutParams`, en <http://developer.android.com/reference/android/view/ViewGroup.LayoutParams.html>, todos los widgets dentro de un Layout van a tener las propiedades `layout_width` y `layout_height` que permiten especificar cómo se va a rellenar el espacio del contenedor. Imagina por ejemplo que tienes un layout con dos widgets de texto, que por su tamaño inicial no rellenan todo el espacio del contenedor:

- Se puede indicar en `layout_width` y `layout_height` una dimensión determinada, por ejemplo 150dip para indicar que el widget debe tener exactamente ese tamaño. (Cuidado, porque este tipo de parametrizaciones hace que tu aplicación sea dependiente del tamaño de la pantalla).
- Se puede indicar `wrap_content`, que significa que el widget debería ocupar exactamente su espacio natural, a no ser que sea demasiado grande para entrar en el contenedor, en cuyo caso se haría ajuste automático de línea mediante word-wrap.
- Se puede indicar `match_parent`, en cuyo caso, el widget se haría tan grande como el propio contenedor.

Summary

XML Attributes

Attribute Name	Related Method	Description
<code>android:layout_height</code>		Specifies the basic height of the view.
<code>android:layout_width</code>		Specifies the basic width of the view.

Constants

int	<code>FILL_PARENT</code>	Special value for the height or width requested by a View.
int	<code>MATCH_PARENT</code>	Special value for the height or width requested by a View.
int	<code>WRAP_CONTENT</code>	Special value for the height or width requested by a View.

Imagen: Atributos de un `ViewGroup.LayoutParams`

Cada control se agrupa dentro de un contenedor a cierta distancia del resto. Esto se puede controlar a través de las propiedades de márgenes, `android:layout_marginXXXX`, donde XXXX puede ser Top, Left, End, Right, etc... de manera muy similar a como lo haces con las hojas de estilos css y el modelo de cajas.

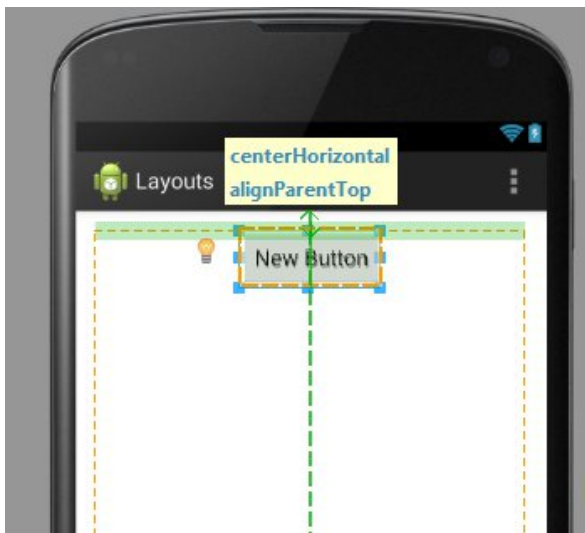
En los próximos apartados podrás leer más sobre los diferentes tipos de layouts y sus propiedades. Mira con atención los videos de las demos de los siguientes apartados.

2.4.2. Relative layouts

El layout relativo permite a los hijos (elementos contenidos) especificar su posición con relación al padre (contenedor) o uno de sus hermanos. De esta manera se pueden alinear elementos al borde, o poner uno debajo de otro, centrado en la pantalla, a la izquierda, etc. Por ejemplo, si en el fichero de Layout escribes:

```
android:layout_alignParentTop="true"
```

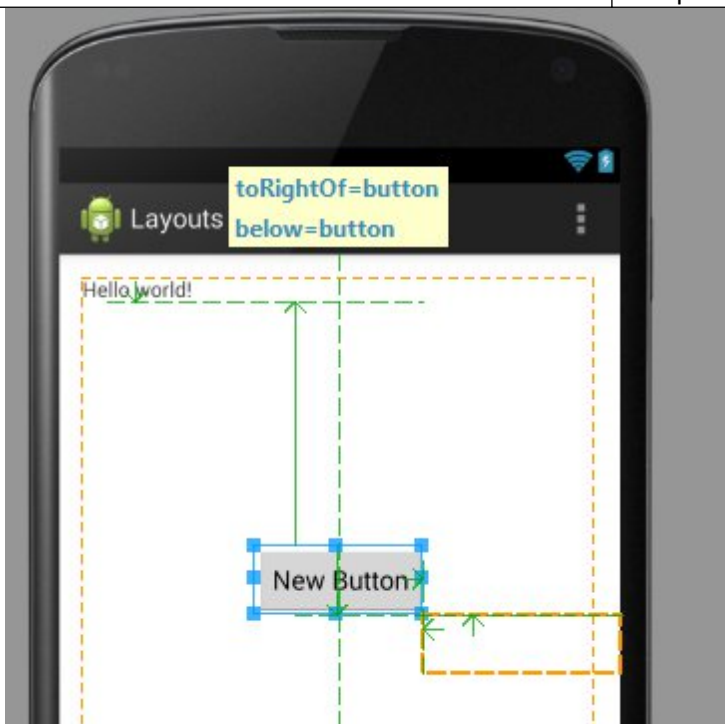
en un elemento hijo, harás que ese elemento se alinee en el borde de arriba para cuadrar con el borde de arriba del padre:



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

Tienes otras muchas propiedades:

android:layout_centerVertical="true"	Centra el hijo verticalmente con su padre
android:layout_below="@+id/xxxxx"	Coloca el elemento debajo de xxxxx
android:layout_toRightOf="@+id/xxxxx"	Coloca el elemento a la derecha de xxxxx
android:layout_alignParentLeft="true"	Coloca el hijo alineado a la borde izquierdo del padre



Tienes todas las opciones posibles documentadas en:

<http://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html>

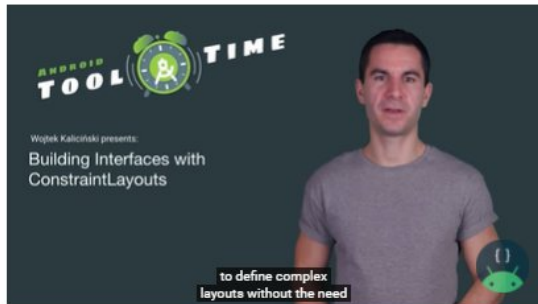
2.4.3. Constraint layouts

Desde que Google liberó la librería JetPack el Layout que incorporan las aplicaciones de Android por defecto es el ConstraintLayout. Este layout te permite crear diseños grandes y complejos con una jerarquía sin grupos de vistas anidadas.

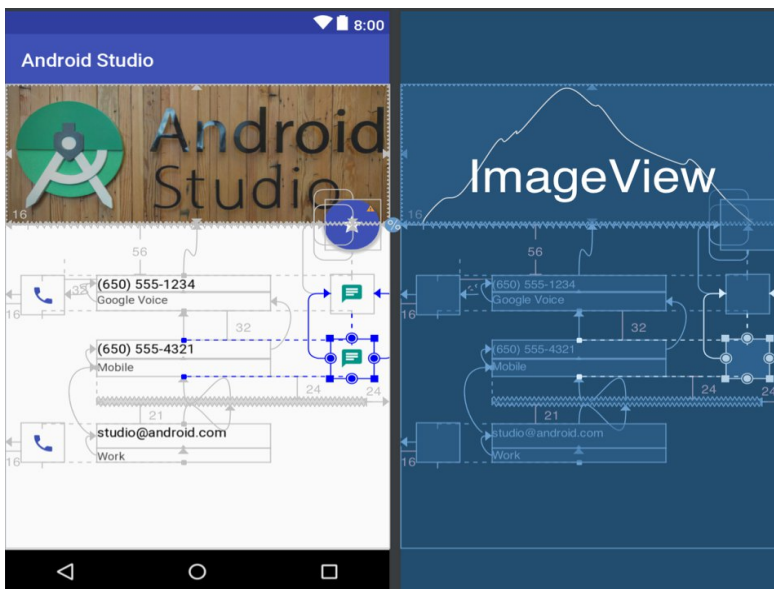
Tiene la misma filosofía que RelativeLayout puesto que todas las vistas se posicionan con relación a otras vistas, pero es mucho más sencillo e intuitivo de utilizar puesto que está integrado completamente con el entorno integrado de

desarrollo. Puedes crear tu diseño con ConstraintLayout arrastrando y soltando componentes, en lugar de editando el XML.

Mira este video, donde Wojtek Kalicinski, de Google, te enseña a construir interfaces usando ConstraintLayout: <https://youtu.be/XamMbnzI5vE>

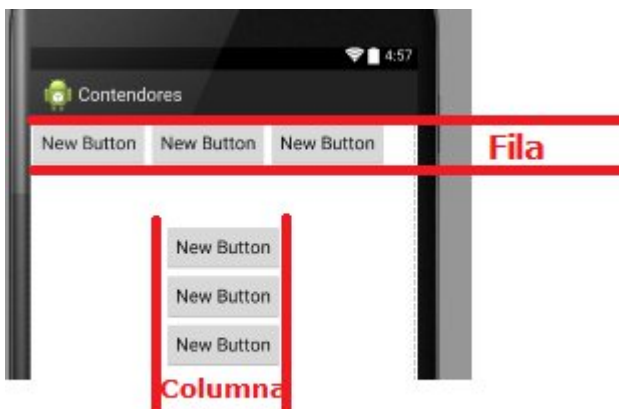


Si construyes todas tus actividades con ConstraintLayout podrás generar una aplicación con una interfaz de usuario reactiva, es decir una aplicación que, independientemente de dónde se ejecute, se adapte a cualquier variación de tamaño o resolución de pantalla:



2.4.4. Linear layouts

El Layout lineal, es un modelo de cajas, es decir alinea los controles horizontalmente en una fila o verticalmente en una columna.



Para indicar si los controles irán agrupados en fila o en columna se utiliza la propiedad *orientation*:

`android:orientation="vertical"` si se quiere agrupar los controles en columna

`android:orientation="horizontal"` si se quiere agrupar los controles en una fila

Además de estas propiedades y de *android:layout_width* y *android:layout_height* tenemos la propiedad de *peso* `android:layout_weight`. La propiedad *peso* permite otorgar preferencia a la hora de rellenar el espacio del contenedor en caso de que queramos que todos los controles ocupen el espacio entero del contenedor. Con la propiedad *peso*, indicaremos la proporción de espacio que deben ocupar los widgets.

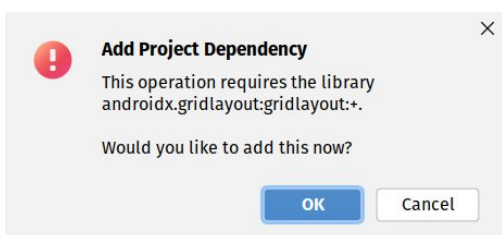
LA GRAVEDAD: La otra propiedad importante de un Layout lineal es la propiedad *layout gravity*. Por defecto todo se alinea de arriba abajo y de izquierda a derecha. Si queremos romper esta regla, debemos usar la gravedad. Por ejemplo, en la imagen anterior, los tres controles con *peso*, están alineados a la izquierda. Si quisiéramos alinear alguno a la derecha, por ejemplo, el de *peso* 1, deberíamos establecer su propiedad

`android:layout_gravity="right"`



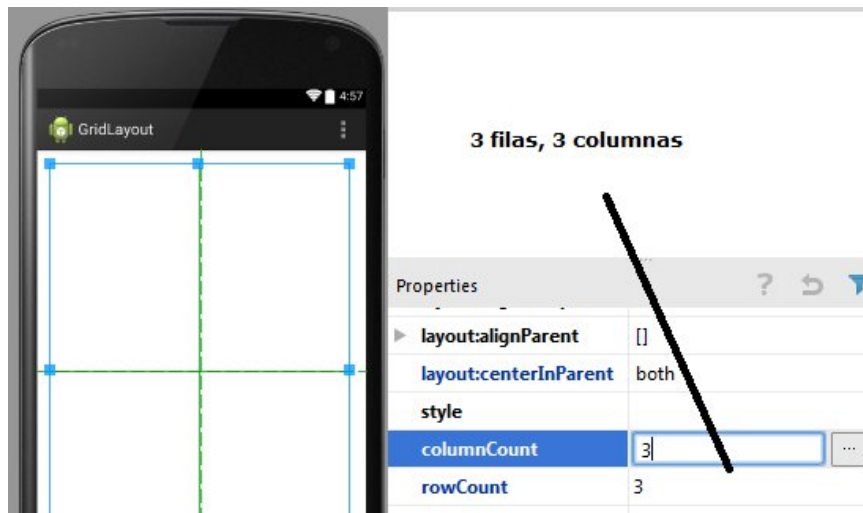
2.4.5. Layouts tabulares

El último grupo de Layouts son los Layouts tabulares. Estos layouts crean una distribución basada en filas y columnas tal y como lo haces con una tabla html o si lo prefieres, como en una hoja de cálculo. De estos tenemos el `GridLayout` y el `TableLayout`. Ambos son bastante similares así que nos centraremos en el `GridLayout` y después, puedes experimentar por tu cuenta el `TableLayout`. Busca el `GridLayout` dentro de los componentes “Legacy” y arrástralo a tu `ConstraintLayout`. Añade la dependencia que te sugiere Android Studio y comienza a trabajar con el componente:

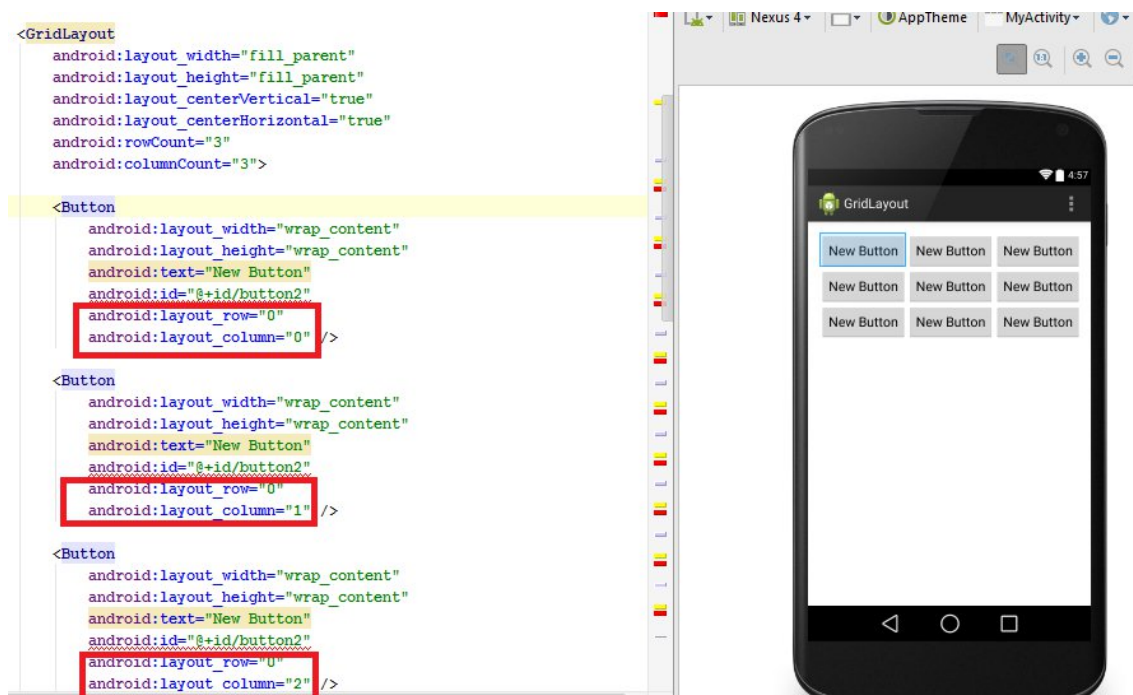


Primero, debes especificar el número de filas y columnas que va a tener tu Grid:

Esto se hace con la propiedad *android:rowCount* y *android:columnCount*



Una vez especificadas las filas y columnas, es posible incluir widgets dentro de cada fila y cada columna de forma muy sencilla. Tan sólo hay que especificar, en cada widget, a qué columna (`android:layout_row`) y a qué fila va a pertenecer (`android:layout_column`). Aunque en la imagen siguiente hemos proporcionado valores a estas dos propiedades para todos los componentes que hemos agregado a la grid, en realidad no es necesario rellenarlas para todos los componentes, puesto que por defecto se alinearán de izquierda a derecha automáticamente si no indicas su posición exacta en la grid.



Como puedes ver en el ejemplo, la primera fila es la fila 0 y la primera columna es la columna 0. Si quieres modificar el espacio entre los controles puedes utilizar la propiedad `layout_margin`, al igual que lo haces en las hojas de estilo CSS.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="20dp"
    android:text="New Button"
    android:id="@+id/button2"
    android:layout_row="0"
    android:layout_column="0" />
```



Se puede dejar una posición vacía en el Grid simplemente no incluyendo ningún control en esa posición:



Al igual que en HTML se utiliza el concepto de *row spanning* y *col spanning* (span) para hacer que un elemento ocupe más de una celda, Android permite igualmente hacerlo con los componentes en un LayoutGrid. Tan sólo hay que jugar con las propiedades `android:layout_columnSpan` y `android:layout_rowSpan` y activarlas asignando el valor "fill" a la propiedad `android:layout_gravity` del componente. Por ejemplo, queremos hacer que el primer botón de la segunda fila llene el espacio que corresponde al hueco que hemos dejado, podemos "expandirlo" mediante las propiedades mencionadas

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="New Button"
    android:id="@+id/button2"
    android:layout_columnSpan="2"
    android:layout_gravity="fill"
    android:layout_row="1"
    android:layout_column="0" />
```



2.4.6. Trabajando programáticamente con contenedores

La otra posibilidad es trabajar con Layouts desde el código fuente. De hecho, necesitarás hacerlo cuando quieras:

- Recorrer todos los hijos del Layout.
- Añadir o eliminar elementos hijos de un Layout en tiempo de ejecución.
- Responder a eventos de los widgets hijos de un Layout.

2.4.6.1. Recorrido del contenedor

El recorrido de los widgets de un Layout es la operación más básica que puedes hacer. Recorrer significa obtener una referencia a cada uno de los widgets que están en un Layout. Se puede hacer fácilmente con un bucle for.

¿Te acuerdas de que cada Layout es una subclase de la clase ViewGroup? ViewGroup es una clase abstracta que representa una agrupación de objetos de la clase View. A su vez cada widget es una subclase de la clase View, por tanto, examinando la documentación de la clase ViewGroup podemos hacernos una idea de los métodos que hay que utilizar para poder acceder a un widget dentro de un contenedor, y de esta manera, programar un bucle que los recorra todos. Es así de sencillo:

El método `getChildAt(int index)` de la clase ViewGroup devuelve una referencia al widget con identificador `index` dentro del contenedor. Index representa el elemento insertado en una determinada posición. Puedes imaginarte que el Layout es un array dinámico con widgets dentro y que el gracias al índice accedes a uno de los elementos de la colección.

Por otro lado, ViewGroup tiene otro método llamado `getChildCount()` que devuelve un número entero indicando cuantos elementos hay en el contenedor.

Ya puedes programar el siguiente pedazo de código:

```
public void Recorrer() {
    View v;
    GridLayout g = (GridLayout) findViewById(R.id.grid1);
    for (int i = 0; i < g.getChildCount(); i++) {
        v=g.getChildAt(i);
        Log.d(TAG, "objeto:"+ v.toString());
    }
}
```

Lo primero que hay que hacer es obtener una referencia al Layout, del tipo que sea, en el ejemplo de recorrido se ha utilizado un GridLayout, pero podría haber sido cualquier tipo de Layout.

Con un bucle for se visitan todos los hijos desde el hijo 0 hasta `getChildCount()-1`.

En cada iteración se obtiene una referencia al i-ésimo widget hijo (v).

2.4.6.2. Diferenciando tipos.

Como sabes, un contenedor puede tener distintos tipos de widgets. Para conocer qué tipo de widget es cada uno de ellos (qué tipo de clase es), puedes utilizar el método `getClass` de la clase View, e invocar a su método `getSimpleName` para conocer el nombre de la clase a la que pertenece el objeto View. Luego puedes hacer un *cast* a un objeto de una clase determinada para tratar sus propiedades específicas. Por ejemplo:

```
Button b;
if(v.getClass().getSimpleName().equals("Button")) {
    b=(Button)v;
    b.setOnClickListener(...); //o cualquier otro método/prop.de la clase Button
}
```

2.4.6.3. Añadiendo elementos al contenedor

Puedes añadir los elementos que quieras al contenedor, teniendo en cuenta las siguientes consideraciones:

- Cada widget que incluyas debe tener definido sus parámetros de Layout, de lo contrario, encontrarás resultados desagradables. Los widgets tienen un método llamado `setLayoutParams` para establecer las propiedades en tiempo de ejecución.
- Cada widget debe tener un identificador. Cuando lo haces con XML se indica la propiedad *id* mediante un recurso, mientras que en código fuente se hace con un entero. La cuestión se traduce en qué identificador puedes ponerle a un objeto creado en tiempo de ejecución. Para que no haya conflictos y dupliquemos estos id, la clase View, a partir de la Api 17 de android proporciona un método llamado *generateViewId* que devuelve un id único no utilizado hasta ese momento.

La clase ViewGroup incorpora el método `addView(View v,int index)` para incorporar todos los elementos que desees. De esta manera, el siguiente ejemplo de código, inserta 18 botones en un GridLayout sin necesidad de especificarlo en código fuente:


```

public void añadeHijos(){
    GridLayout g = (GridLayout) findViewById(R.id.grid1);
    Button b;
    for(int i=0;i<18;i++) {
        b = new Button(this);
        b.setLayoutParams(new ViewGroup.LayoutParams(
            ViewGroup.LayoutParams.WRAP_CONTENT,
            ViewGroup.LayoutParams.WRAP_CONTENT));
        b.setText("btn" + i);
        b.setId(View.generateViewId());
        g.addView(b,i);
    }
}

```

Por cada iteración se producen las siguientes operaciones:

En primer lugar se instancia el botón:

```
b = new Button(this);
```

A continuación se establecen los parámetros de tamaño para el Layout, en concreto se va a ajusta al contenido del botón, tanto en altura con el anchura:

```

b.setLayoutParams(new ViewGroup.LayoutParams(
    ViewGroup.LayoutParams.WRAP_CONTENT,
    ViewGroup.LayoutParams.WRAP_CONTENT));

```

Se establece el texto del botón y el identificador: `b.setText("btn" + i);`

Observa como se establece el identificador único para cada uno de los 18 botones mediante la línea:

```
b.setId(View.generateViewId());
```

Para responder a eventos generales de los widgets hijos, necesitas un objeto Listener que escuche por todos (o por ciertos elementos individuales). Por ejemplo, imagina que quieres que cada botón del GridLayout de los ejemplos anteriores, responda a un evento para hacer una acción determinada. Podríamos programar el siguiente código:

```

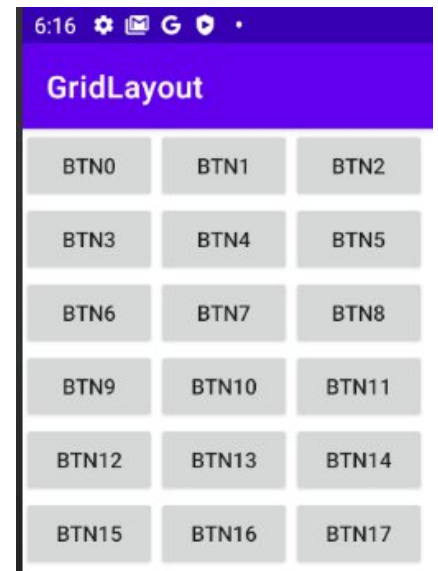
public class MainActivity extends AppCompatActivity implements View.OnClickListener {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        añadeHijos();
    }
    public void añadeHijos() {
        GridLayout g = (GridLayout) findViewById(R.id.grid1);
        Button b;
        for (int i = 0; i < 18; i++) {
            b = new Button(this);
            b.setLayoutParams(new ViewGroup.LayoutParams(
                ViewGroup.LayoutParams.WRAP_CONTENT,
                ViewGroup.LayoutParams.WRAP_CONTENT));

            b.setId(View.generateViewId());
            b.setOnClickListener(this);
            g.addView(b, i);
        }
    }

    public void onClick(View v) {
        if (v.getClass().getSimpleName().equals("Button")) {
            Button b = (Button) v;
            acción(b);
        }
    }

    public void acción(Button b) {
        //programa aquí tu acción con el botón b
    }
}

```



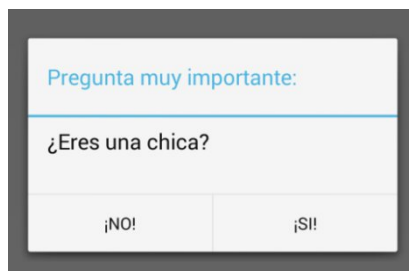
Observa como en cada iteración se establece el listener a la propia actividad, para que responda mediante el método `onClick`. Finalmente, cuando se presiona uno de los botones del contenedor `GridLayout`, se invoca al método *acción* pasando como parámetro el botón pulsado para consultas posteriores



CASO PRÁCTICO: Realiza una aplicación que programáticamente maneje controles dentro de un `GridLayout`. El programa deberá crear 18 botones automáticamente con diferentes colores, al pulsar cada uno de ellos, se volverá blanco. El último botón será un botón de `RESET`, de tal manera que cuando se pulse, se vuelva a la configuración original.

2.5. Los diálogos y los fragmentos

Los diálogos son ventanas que aparecen en la pantalla de tu dispositivo móvil de manera espontánea para hacer al usuario alguna pregunta sobre alguna decisión que deba tomar para poder proceder con el funcionamiento normal de tu App.



Los fragmentos son pequeñas partes de una actividad. Estas partes tienen su propio ciclo de vida y dependen del ciclo de vida de la actividad. Estudiaremos más sobre fragmentos en la próxima unidad, de momento nos centramos en este tipo especial de fragmento llamado diálogo.

Hay muchos tipos de diálogos, tienes diálogos con listas de elementos, diálogos con botones de tipo `Radio` o `checkboxes`, incluso puedes utilizar un fichero de recursos XML para definir un `Layout` en el que diseñes tu diálogo personalizado.

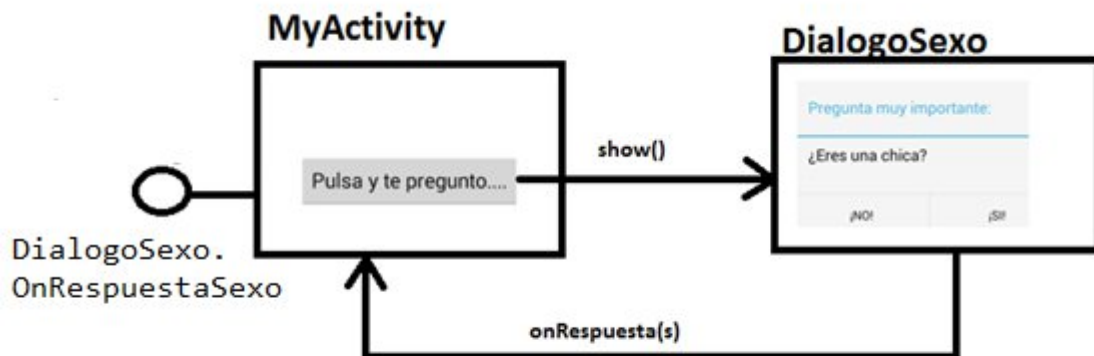
ACTIVIDAD: Puedes encontrar más documentación sobre cómo personalizar tus diálogos en:

<http://developer.android.com/guide/topics/ui/dialogs.html#CustomLayout>

Android recomienda usar la clase `DialogFragment` para contener diálogos. Un fragmento de una actividad es una parte de la App que controla una parte de la interfaz de usuario de la Activity. Los fragmentos tienen su propio ciclo de vida, reciben sus propios eventos y se pueden eliminar o añadir a la Activity mientras esta se ejecuta. Un `DialogFragment` es un tipo especial de fragmento usado para crear diálogos.

El caso práctico que te vamos a plantear a continuación es un escenario que te surgirá muchas veces. Imagina que tienes una App que en base a la actuación de un usuario necesitas enseñarle un cuadro de diálogo para hacerle una

pregunta. En base a la respuesta a esta pregunta, la aplicación tomará un curso de acción u otro. Si lo haces con las recomendaciones de Android y programas el diálogo como un DialogFragment, además, te servirá para futuras ocasiones. Por ejemplo, supón que tienes una Actividad con un botón, al pulsar el botón aparecerá un cuadro de diálogo preguntándote por tu sexo. Al terminar el diálogo, este retornará devolviendo a la actividad el resultado mediante un *callback*. Observa el siguiente esquema:



Nuestra App esta vez tendrá dos clases, una con nuestra actividad principal “MyActivity” y otra con una clase llamada DialogoSexo que extenderá la clase DialogFragment.

El código de la actividad principal es muy sencillo:

```
public class MainActivity extends AppCompatActivity
    implements DialogoSexo.OnRespuestaSexo {

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

    public void Click(View v){
        DialogoSexo ds=new DialogoSexo();
        ds.show(getSupportFragmentManager(),"Mi diálogo");
    }

    @Override
    public void onRespuesta(String s) {
        Toast.makeText(getApplicationContext(),s,Toast.LENGTH_LONG ).show();
    }
}
```

Programamos una función que responda al click del botón y creamos un objeto de la clase DialogoSexo, invocando al método show de un Dialog, que recibe como parámetros un objeto llamado FragmentManager que se obtiene con la llamada al método *getFragmentManager()* de la Activity, y una etiqueta con el texto identificador del diálogo.

Además, hay que recibir la respuesta que nos envía el diálogo, esto lo haremos registrando una función de callback llamada *onRespuesta()*, método que implementamos a través de una interfaz llamada *RespuestaDialogo* y que programaremos nosotros en la clase DialogoSexo. Este método onRespuesta crea un pequeño mensaje a modo de notificación que le aparece al usuario en el dispositivo móvil a través de la clase Toast. Crear una notificación es muy fácil con la clase Toast, cuya documentación puedes consultar en <http://developer.android.com/guide/topics/ui/notifiers/toasts.html>

Por otro lado, la clase DialogoSexo, tiene el siguiente código:

```

public class DialogoSexo extends DialogFragment {
    OnRespuestaSexo respuesta;

    /* Este método es llamado al hacer el show() de la clase DialogFragment() */
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {

        // Usamos la clase Builder para construir el diálogo
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        //Escribimos el título
        builder.setTitle("Pregunta muy importante:");
        //Escribimos la pregunta
        builder.setMessage("¿Eres una chica?");
        //añadimos el botón de Si y su acción asociada
        builder.setPositiveButton(";SI!", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                respuesta.onRespuesta("Es una chica!");
            }
        });
        //añadimos el botón de No y su acción asociada
        builder.setNegativeButton(";NO!", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                respuesta.onRespuesta("Es un chico!");
            }
        });
        // Crear el AlertDialog y devolverlo
        return builder.create();
    }

    //interfaz para la comunicación entre la Actividad y el Fragmento
    public interface OnRespuestaSexo {
        public void onRespuesta(String s);
    }

    //Se invoca cuando el fragmento se añade a la actividad
    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        respuesta=(OnRespuestaSexo) context;
    }
}

```

Desgranamos paso a paso el código fuente:

Los diálogos se crean mediante la clase `AlertDialog.Builder`, estableciendo el título con `setTitle()`, el mensaje principal `setMessage()`, y estableciendo los botones de sí o no con su correspondiente respuesta a través de una función de callback de la interfaz `DialogInterface.OnClickListener`. Todo esto se hace reprogramando el método `onCreateDialog` de la clase `DialogFragment`, método que se invoca cuando la actividad principal ejecuta el `show` de nuestra clase `DialogoSexo`.

Cuando el usuario pulsa uno de los botones del diálogo (sí o no), se desencadena la llamada a la función de *callback* `onRespuesta` que se ha programado en la actividad principal. Para poder hacer esto, se recurre a un ingenioso mecanismo:

- Cuando el diálogo se crea, el fragmento se une "Attach" a la actividad principal. En ese momento el método `onAttach()` se ejecuta y podemos quedarnos con una referencia a la actividad.

```
respuesta=(OnRespuestaSexo) context;
```

- Gracias a esa referencia a la actividad, podremos invocar al método `onRespuesta` de la interfaz que obligamos a la actividad principal a implementar.

```

builder.setPositiveButton(";SI!", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        respuesta.onRespuesta("Es una chica!");
    }
});

```

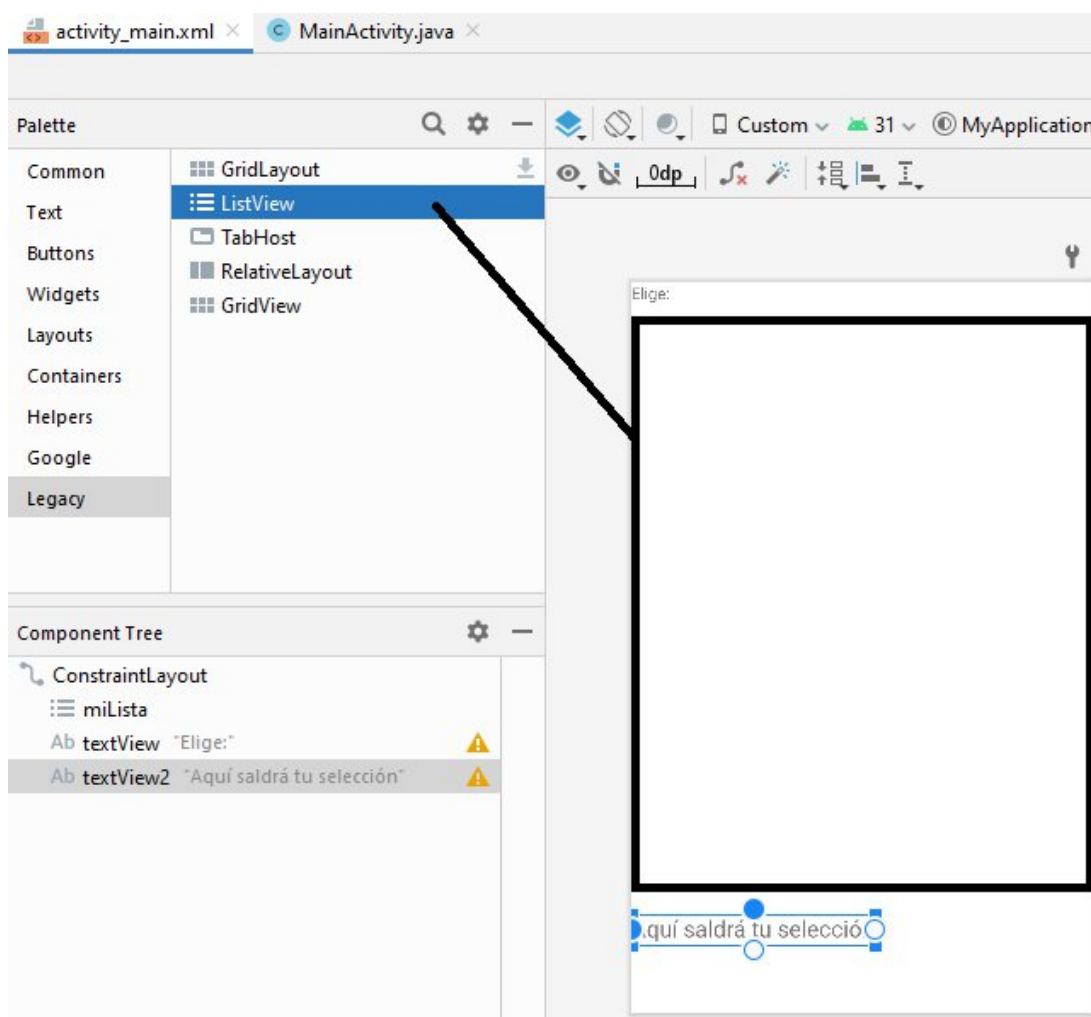
La clase Builder de AlertDialog, tiene un montón de métodos que te serán de utilidad para crear tu diálogo. Por ejemplo, además de los que has visto en el código de ejemplo (setTitle, setMessage, setPositiveButton, setNegativeButton) tienes métodos como **setItems()** para coger de un array una lista de elementos a seleccionar, **setSingleChoiceItems()** si quieres mostrar una lista con RadioButtons y **setMultiChoiceItems()** si quieres mostrar una lista de elementos con Checkboxes. Esta clase, con estos métodos te da toda la libertad del mundo para construir casi cualquier diálogo, no obstante, siempre puedes construirte tu propio recurso de layout personalizado e inflar el diálogo a partir de este recurso personalizado.

ACTIVIDAD: Puedes descargar el código de este ejemplo de los recursos del tema: [ficheros_tema2/DialogosConRespuesta.zip](#)

2.6. Los widgets de selección

Los widgets de selección permiten al usuario elegir un valor de una lista de posibles valores. De esta manera nos evitaremos tener que hacer miles de comprobaciones a la hora de validar la imprevisible entrada del usuario.

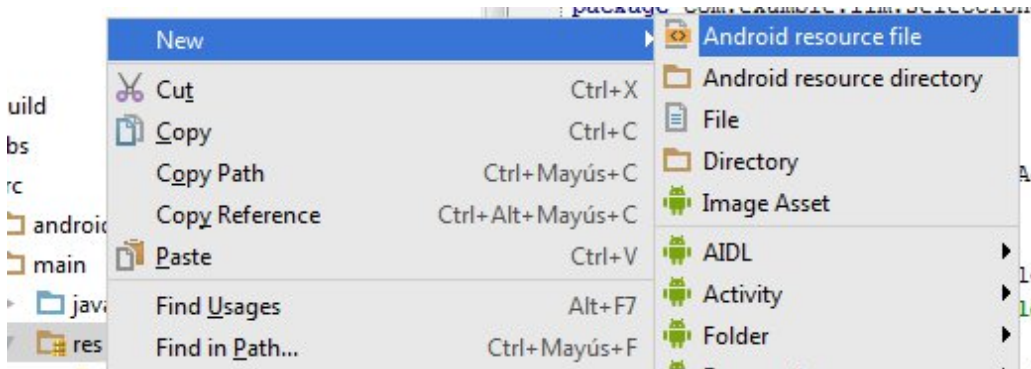
El widget de selección más típico de Android es el ListView. Aunque ya forma parte de los componentes Legacy de Android y sustituido en su mayor parte por los RecyclerView, usaremos esta clase para explicarte los conceptos básicos, y así luego, con otros selectores, puedas aplicar de forma más rápida los conceptos que aprenderás en este apartado. ListView también es una subclase de la clase ViewGroup por lo que también es un contenedor, pero con una peculiaridad: **Los adaptadores**. Los adaptadores son objetos que actúan de puentes entre un widget y sus datos. Determina, entre otras cosas, el tipo de datos que va a contener el widget. Por ejemplo, para crear una actividad con una lista de elementos seleccionables, crea un nuevo proyecto en Android y desde la paleta de componentes, inserta un ListView y dos etiquetas para que quede una interfaz de usuario como en la imagen:



Para poder llenar tu lista de contenido necesitarás dos cosas:

- Un adaptador
- Un nuevo recurso a modo de layout file con la definición de un TextView que se usará como elemento simple de tu ListView.

Para hacerlo, tan sólo tienes que declarar en el código fuente un objeto de esta clase y rellenarlo con los elementos de un array. Estos elementos, a través de un adaptador, serán canalizados hasta los elementos individuales que contenga la lista. Aquí te mostramos cómo hacerlo. Primero, inserta un nuevo fichero de recursos con el nombre "fila.xml":



A continuación, abre el fichero fila.xml e inserta el código para crear un TextView con el formato que desees, por ejemplo, inserta este código:

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textStyle="italic"
    android:textSize="20dp"
    android:textColor="#FF0000"
/>
```

Después, programa la clase MainActivity de la siguiente manera:

```
public class MainActivity extends AppCompatActivity
    implements AdapterView.OnItemClickListener{

    ListView miLista;

    String [] provincias= new String[]{"Ciudad Real","Toledo","Guadalajara",
        "Cuenca","Albacete","Talavera"}; //Datos

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        miLista=findViewById(R.id.miLista); //IU

        ArrayAdapter<String> miAdaptador=new ArrayAdapter<String>(
            this, R.layout.fila,provincias); //Adaptador

        //enfuchar adaptador a la vista
        miLista.setAdapter(miAdaptador);
        miLista.setOnItemClickListener(this);
    }

    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        TextView txtResultado=findViewById(R.id.txtResultado);
        txtResultado.setText("Se ha pulsado "+ provincias[position]);
    }
}
```



Fíjate, para crear el adaptador, el constructor de la clase ArrayAdapter recibe tres parámetros:

- La referencia a la propia actividad (this)
- La referencia al nuevo fichero de recurso que has creado (R.layout.fila)
- Los elementos del array con el texto que se escribirá en cada TextView que forme la lista de selección.

Para saber cuál fue la selección del usuario necesitas registrar el evento con `setOnItemClickListener(this)`, y recibirlo con el método `onItemClickListener` de la interfaz `OnItemClickListener`.

Fíjate en el método `onItemClick`, que nos pasa como parámetro el componente padre donde se hizo la selección (`AdapterView<?>`), la View (el widget) que se pulsó, la posición que ocupa en lista y el identificador de la fila que se selección

```
public void onItemClick(AdapterView<?> parent, View view, int position, long id)
```

Este método es el método *callback* que se invoca cuando se hace click en alguno de los elementos de la lista.

También puedes elegir el elemento que se ha seleccionado con el método *getItemAtPosition* de la siguiente forma:

```
String s=parent.getItemAtPosition(position).toString();
```

Otra alternativa sería, como también te pasan la vista que se ha pulsado, convertir la vista a `TextView` y obtenerlo con el método `getText`:

```
String s=((TextView)view).getText().toString();
```

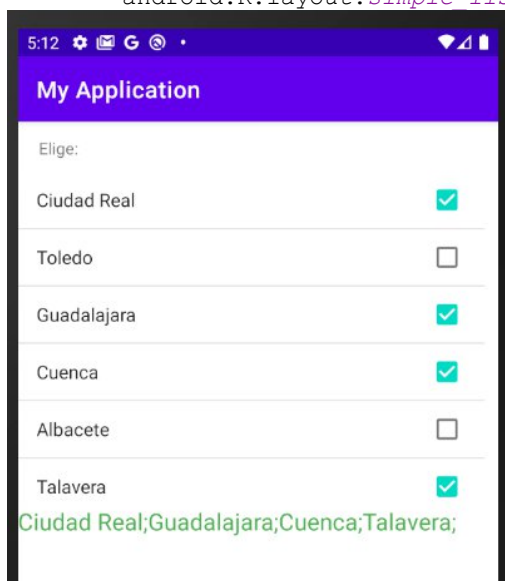
ACTIVIDAD: Puedes descargar el código de los recursos del tema: [ficheros_tema2/ListasSimples.zip](#)

2.6.1. Seleccionando multiples elementos.

Un `ListView` se puede configurar para poder escoger selección múltiple. Esto se puede seleccionar mediante el atributo XML `android:choiceMode="multipleChoice"` o simplemente invocando el método `setChoiceMode(CHOICE_MODE_MULTIPLE)` desde código fuente.

Además, deberás utilizar como Layout para la selección el predefinido por Android para la selección múltiples, llamado `android.R.layout.simple_list_item_multiple_choice`, de tal manera que al crear el adaptador, debes indicarlo como segundo parámetro:

```
ArrayAdapter<String> miAdaptador=new ArrayAdapter<String> (this,  
    android.R.layout.simple_list_item_multiple_choice, provincias);
```



Los métodos para capturar los eventos de selección funcionan exactamente igual que un ListView de selección simple. Eso sí, si queremos explorar todos los elementos seleccionados, por ejemplo mediante un bucle, podemos usar el siguiente código:

Previamente:

```
miLista=findViewById(R.id.miLista); //IU
miLista.setChoiceMode(AbsListView.CHOICE_MODE_MULTIPLE);
miLista.setAdapter(miAdaptador);
```

En el método onItemClick:

```
@Override
public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
    TextView txtResultado=findViewById(R.id.txtResultado);
    String seleccionado=new String();
    SparseBooleanArray checked = miLista.getCheckedItemPositions();
    if(checked!=null) {
        for (int i = 0; i < checked.size(); i++)
            if (checked.valueAt(i)) {
                seleccionado = seleccionado +
                    miLista.getItemAtPosition(checked.keyAt(i)).toString()
                    + ";";
            }
    }
    txtResultado.setText(seleccionado);
}
```

Esta fracción de código responde al evento de hacer una selección en uno de los elementos de la ListView. Utiliza un array booleano disperso (SparseBooleanArray) para obtener de la lista los elementos que están seleccionados (checked):

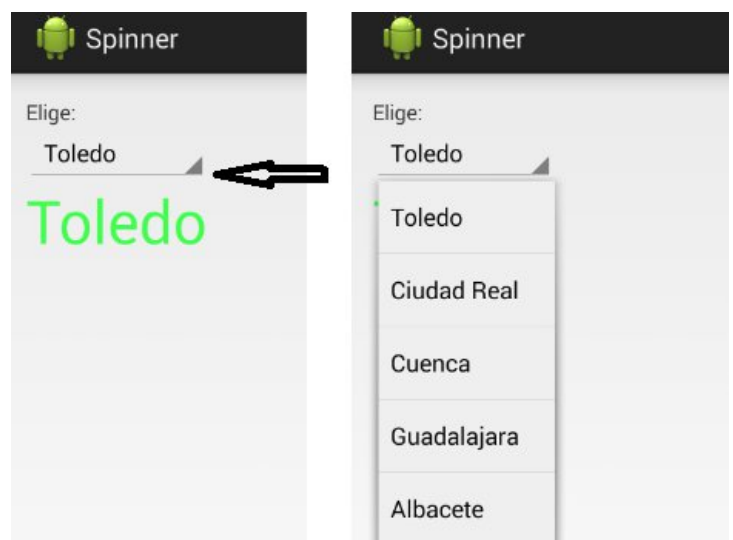
```
SparseBooleanArray checked = miLista.getCheckedItemPositions();
```

El método getCheckedItemPositions de la ListView devuelve este tipo de array para que puedas consultar qué valores se han seleccionado. Puedes consultar toda la documentación del SparseBooleanArray en <http://developer.android.com/reference/android/util/SparseBooleanArray.html> pero en resumen, puedes ver que el método *valueAt(i)* devuelve un booleano indicando si el elemento *i* del array fue seleccionado o no. Gracias al método *keyAt(i)* de este array, devuelve un número indicando la posición de un elemento seleccionado dentro de la ListView. De esta manera se puede hacer un bucle desde 0 hasta el tamaño del array y extraer los valores múltiples seleccionados.

ACTIVIDAD: Descárgate el código completo aquí [ficheros_tema2/SeleccionMultiple.zip](#)

2.6.2. Los spinners.

Los Spinners en Android son los equivalentes a los ComboBox de otros sistemas, es decir, listas desplegables. Funcionan prácticamente igual que los ListView, es decir, hay que utilizar un adaptador para enchufar los datos y capturar los eventos de selección, en el caso de los Spinners a través de *setOnItemSelectedListener()* de la interfaz *OnItemSelectedListener* de la clase *Spinner*.



El adaptador se programa prácticamente igual:

```
adaptador = new ArrayAdapter<String>(this,
    android.R.layout.simple_spinner_item, elementos);
adaptador.setDropDownViewResource(
    android.R.layout.simple_spinner_dropdown_item);
```

En este caso hay dos layouts, uno para mostrar el elemento seleccionado `android.R.layout.simple_spinner_item` que se pasa al constructor del adaptador como segundo parámetro (al igual que en el `ListView`) y otro para mostrar los elementos desplegables `android.R.layout.simple_spinner_dropdown_item`, que se especifica a través del método `setDropDownViewResource()`:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    String[] elementos = {"Toledo", "Ciudad Real",
        "Cuenca", "Guadalajara", "Albacete"};

    ArrayAdapter<String> adaptador;

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Spinner sp = (Spinner) findViewById(R.id.spinner);
    adaptador = new ArrayAdapter<String>(this,
        android.R.layout.simple_spinner_item, elementos);
    adaptador.setDropDownViewResource(
        android.R.layout.simple_spinner_dropdown_item);
    sp.setAdapter(adaptador);
    sp.setOnItemSelectedListener(this);
}
```

Para capturar el evento de selección de un elemento hay que implementar dos métodos, puesto que la interfaz `OnItemSelectedListener` exige los dos:

```
public class MainActivity extends AppCompatActivity
    implements AdapterView.OnItemSelectedListener {

    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id)
    {
        //Callback cuando se selecciona un elemento del Spinner
        TextView txtResultado = findViewById(R.id.txtResultado);
        Spinner sp = (Spinner) findViewById(R.id.spinner);

        txtResultado.setText("Se ha seleccionado "+sp.getSelectedItem().toString());
    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {
        //Callback cuando se no se selecciona un elemento del Spinner
        TextView txtResultado = findViewById(R.id.txtResultado);
        txtResultado.setText("No se ha seleccionado nada");
    }

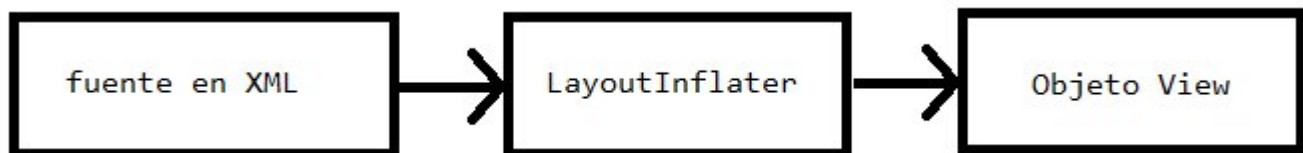
    ...
}
```

ACTIVIDAD: Puedes descargar el código de los recursos del tema: ficheros_tema2/Spinner.zip

2.6.3. Las selecciones personalizables.

Cualquier tipo de control de selección es personalizable. Es decir, podemos definir la estructura de cada fila de la lista de selección como nosotros queramos. Para poder integrarlo con un Spinner, o una ListView, hay que reescribir el funcionamiento del adaptador (`ArrayAdapter<T>`) de manera que cuando el selector demande los datos de las filas, devuelva una vista que sea la que nosotros queramos en cada momento. Es esencial entender dos conceptos:

- El inflador: Se pueden crear vistas (`View`) en tiempo de ejecución a partir de un fichero de recursos XML. De esta manera, se puede crear un widget personalizado e “inflarlo” (crearlo) habiendo definido su estructura mediante un Layout definido en XML. Al inflar un recurso XML se retorna una `View` con todos los elementos que este fichero incluye.



- Se puede hacer “override”, es decir, redefinir métodos de una clase padre. En concreto, si estamos personalizando un Spinner, hay que reprogramar dos métodos del adaptador, `getDropDownView(int position, View convertView, ViewGroup parent)` y `getView(int position, View convertView, ViewGroup parent)` que retornan la vista personalizada de una fila determinada. Si estamos personalizando un ListView tan sólo hay que reprogramar `getView()`. Cuando el Spinner o la ListView invoquen a estos métodos nosotros debemos pasarle la fila personalizada de la posición que nos indique el parámetro posición.

View	<code>getDropDownView(int position, View convertView, ViewGroup parent)</code> Gets a View that displays in the drop down popup the data at the specified position in the data set.
View	<code>getView(int position, View convertView, ViewGroup parent)</code> Get a View that displays the data at the specified position in the data set.

En el siguiente caso práctico, queremos mostrar una ListView personalizada, de tal manera que nos muestre las ciudades de Castilla-La Mancha, pero esta vez con un formato personalizado:

La App consta esta vez de dos ficheros XML, el propio de la actividad, con una ListView y una caja de texto donde se escribirá el resultado de la elección, y el fichero XML con el diseño de una fila de la listView. El fichero de la fila, llamado `mi_fila_personalizada` tiene el siguiente código:

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

    <androidx.cardview.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:cardBackgroundColor="#FFF"
        app:cardCornerRadius="10dp"
        app:cardElevation="3dp"
        app:cardPreventCornerOverlap="true"
        app:cardUseCompatPadding="true"
        app:contentPadding="10dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent">
```

```

app:layout_constraintTop_toTopOf="parent">

<androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/txtCiudad"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView"
        android:textSize="24sp"
        android:textStyle="bold|italic"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

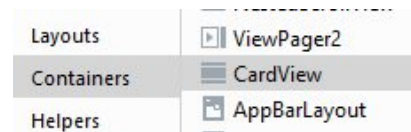
    <ImageView
        android:id="@+id/imgCiudad"
        android:layout_width="200dp"
        android:layout_height="150dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:srcCompat="@tools:sample/avatars" />

    <TextView
        android:id="@+id/txtDescripcion"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/txtCiudad" />

    </androidx.constraintlayout.widget.ConstraintLayout>
</androidx.cardview.widget.CardView>
</LinearLayout>

```

El widget CardView es un contenedor que crea una vista de tipo tarjetas de visita. Puedes utilizar este componente para dar diseños elegantes a tu aplicación:



En el fichero java de la actividad, vamos a crear una subclase de la clase ArrayAdapter, con el método getView reescrito. Esta clase se apoyará en tres arrays de datos para crear objetos de tipo Ciudad. Cada objeto ciudad tiene los datos necesarios para componer una fila de la ListView:

```

public class Ciudad{
    String nombre;
    String descripcion;
    int imagen;

    public Ciudad(String nombre, String descripcion, int imagen) {
        this.nombre = nombre;
        this.descripcion = descripcion;
        this.imagen = imagen;
    }
}

```

Para crear los objetos de este tipo, usaremos los siguientes arrays:

```

ListView miLista;

private String [] provincias= new String[]{"Ciudad Real","Toledo","Guadalajara",
                                           "Cuenca","Albacete","Talavera"}; //Datos
private String [] descripciones= new String[]{"Qué gran ciudad para tapas","La ciudad
imperial","Bonita ciudad de compras",
                                              "Curiosas casas colgantes","El nueva york de la mancha","Agua y sol"}; //Datos
private int imagenes[]=new int[]{R.drawable.ciudadreal,
                                  R.drawable.toledo,R.drawable.guadalajara,
                                  R.drawable.cuenca,R.drawable.albacete,R.drawable.talavera};

```

De esta manera, en el método onCreate de la actividad podremos componer objetos de tipo Ciudad para insertarlos en un ArrayList y mandárselo al nuevo adaptador personalizado:

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    ArrayList<Ciudad> ciudades=new ArrayList<Ciudad>();

    for(int i=0;i< provincias.length;i++){
        Ciudad c=new Ciudad(provincias[i],descripciones[i],imagenes[i]);
        ciudades.add(c);
    }

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    miLista=findViewById(R.id.miLista); //IU -> se va a "inflar" una fila
                                     // por cada ciudad a través de mi_fila_personalizada.xml

    MiAdaptadorPersonalizado adapter=
        new MiAdaptadorPersonalizado(this,R.layout.mi_fila_personalizada,ciudades);

    //enfuchar adaptador a la vista
    miLista.setAdapter(adapter);
    miLista.setOnItemClickListener(this);
}
```

Finalmente la clase MiAdaptadorPersonalizado hereda el comportamiento de ArrayAdapter sobrescribiendo el método getView(). Observa cómo el constructor se queda con el contexto para poder crear una vista dinámicamente a través del LayoutInflater, el recurso a inflar (mi_fila_personalizada.xml) en mResource y la lista de ciudades:

```
public class MiAdaptadorPersonalizado extends ArrayAdapter<Ciudad>{

    private int mResource;
    private ArrayList<Ciudad> misCiudades;

    public MiAdaptadorPersonalizado(@NonNull Context context, int resource,
        @NonNull List<Ciudad> objects) {
        super(context, resource, objects);
        mResource=resource;
        misCiudades=(ArrayList<Ciudad>) objects;
    }

    @NonNull
    @Override
    public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {
        return crearFila(position,convertView,parent);
    }

    private View crearFila(int position, @Nullable View convertView, @NonNull ViewGroup parent){
        //Este método es invocado tantas veces como "filas" se pinten en la actividad

        LayoutInflater miInflador= getLayoutInflater();
        View miFila=miInflador.inflate(mResource,parent,false);

        TextView txtDescripcion=miFila.findViewById(R.id.txtDescripcion);
        TextView txtCiudad=miFila.findViewById(R.id.txtCiudad);
        ImageView imgCiudad=miFila.findViewById(R.id.imgCiudad);

        txtDescripcion.setText(misCiudades.get(position).descripcion);
        txtCiudad.setText(misCiudades.get(position).nombre);
        imgCiudad.setImageResource(misCiudades.get(position).imagen);
        Log.d("RDT","creada la fila"+position);
        return miFila;
    }
}
```

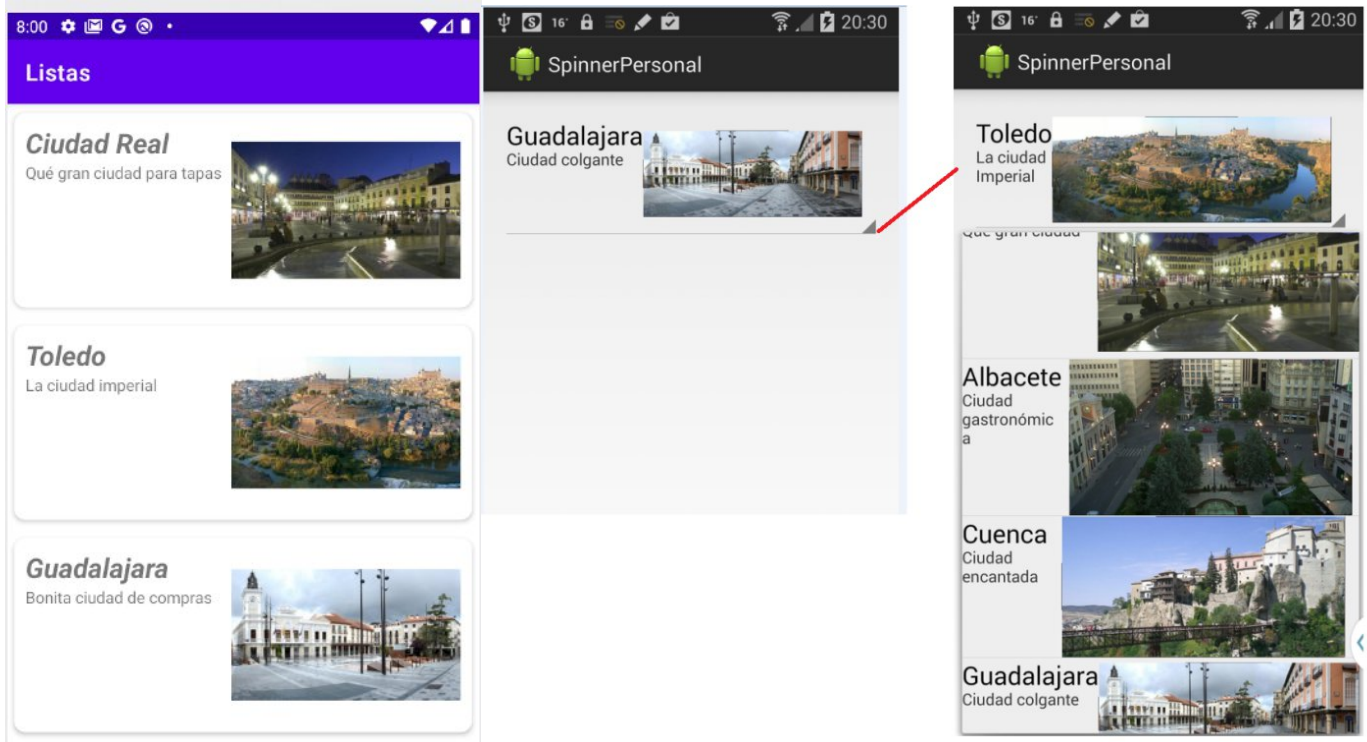
Observa cómo el método reescrito invoca a una tercera función crearFilaPersonalizada() que “**infla**” la vista del fichero xml *mi_fila_personalizada.xml* (miembro mResource) para crear un objeto View con los datos de los arrays y lo retorna. Fijate como mResource ha sido inicializado en la creación del adaptador personalizado.


```
MiAdaptadorPersonalizado adapter=
    new MiAdaptadorPersonalizado(this,R.layout.mi_fila_personalizada,ciudades);
```

Si lo que creas es un Spinner, acuérdate de reescribir en el adaptador el método get:

```
@Override
public View getDropDownView(int position, @Nullable View convertView,
    @NonNull ViewGroup parent) {
    return crearFila(position,convertView,parent);
}
```

De esta forma, quedan resultados tan espectaculares como estos:



ACTIVIDAD: Puedes descargarte el código de aquí: [ficheros_tema2/ListasPersonalizadas.zip](#)

Una de las ventajas de utilizar un adaptador es que si, en algún momento el conjunto de datos cambia, tan sólo es necesario invocar a su método `notifyDataSetChanged()` y automáticamente, el adaptador redibujará todos los elementos que necesite.

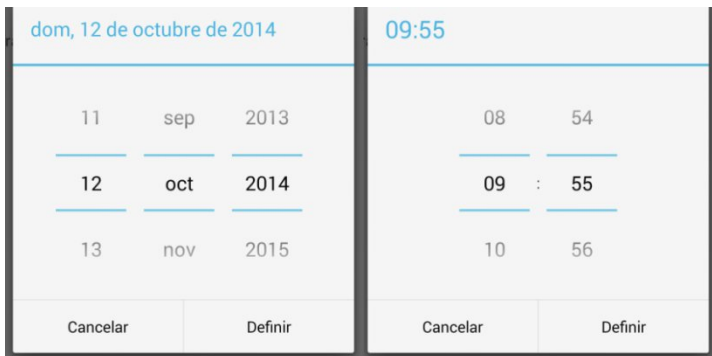
`void`

`notifyDataSetChanged()`

Notifies the attached observers that the underlying data has been changed and any View reflecting the data set should refresh itself.

2.6.4. Los selectores de fecha/hora

En inglés se llaman *Pickers*, *DatePicker* o *TimePicker*. El uso de estos selectores te evitará muchas comprobaciones en código sobre si el usuario escribió correctamente el formato de hora/fecha o si se ajustan a las configuraciones locales del dispositivo móvil o si es una fecha válida.



Al igual que los Diálogos, y aunque se pueden usar directamente en un Layout, Android también recomienda utilizar la clase `DialogFragment` para implementarlos. Puedes releer el punto 2.5 si quieres repasar lo que era un `DialogFragment` porque en el siguiente caso práctico los vamos a volver a utilizar.


El caso práctico consiste en un sencillo formulario que muestra al usuario la posibilidad de introducir su fecha de nacimiento y una hora a través de un campo de texto (*EditText*). Estos campos de fecha y hora estarán inhabilitados para forzar al usuario a pulsar un botón para poder seleccionar de un *Picker* tanto la fecha como la hora. Al pulsar en cada respectivo botón se le mostrará un diálogo con su *Picker* correspondiente:


DatePicker

nombre

fecha de nacimiento

hora





Para cada diálogo creamos una clase que herede de `DialogFragment` y que poseerá una interfaz para comunicarse con la actividad principal, devolviendo esta un objeto `Fecha` cuando se haya seleccionado tanto la fecha como la hora. Para la devolución utilizaremos la clase `GregorianCalendar`, que sustituye a la clase `Date` de Java al quedar esta depreciada (*deprecated*) en la API 1. Para la creación del diálogo se puede usar un Layout XML que contenga un *Picker* o directamente utilizar las clases `DatePickerDialog` y `TimePickerDialog` que directamente construyen un *Dialog* con un *Picker*. En nuestro ejemplo usaremos esta segunda opción. Además, al igual que en el caso práctico del punto 2.5, al crear el fragmento se ejecutará el método `onAttach`, momento que aprovecharemos para quedarnos con una referencia a la actividad y así poder invocar al método `onResultadoFecha/onResultadoHora` de la interfaz y pasar el resultado de la elección del usuario a la actividad principal.

La clase `DialogoFecha` consistirá en:

```

public class DialogoFecha extends DialogFragment
    implements DatePickerDialog.OnDateSetListener{

    OnFechaSeleccionada f;
    @Override
    public void onAttach(Context c) {
        f=(OnFechaSeleccionada) getActivity();
        super.onAttach(c);
    }

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {

        Calendar c=Calendar.getInstance();
        int año=c.get(Calendar.YEAR);
        int mes=c.get(Calendar.MONTH);
        int dia=c.get(Calendar.DAY_OF_MONTH);

        return new DatePickerDialog(getActivity(), this, año, mes, dia);
    }

    @Override
    public void onDateSet(DatePicker view, int year, int month, int dayOfMonth) {
        GregorianCalendar g=new GregorianCalendar(year, month, dayOfMonth);
        f.onResultadoFecha(g);
    }

    public interface OnFechaSeleccionada{
        public void onResultadoFecha(GregorianCalendar fecha);
    }
}

```

Y la clase DialogoHora es muy similar:

```

public class DialogoHora extends DialogFragment
    implements TimePickerDialog.OnTimeSetListener{

    OnHoraSeleccionada f;
    @Override
    public void onAttach(Context c) {
        f=(OnHoraSeleccionada) getActivity();
        super.onAttach(c);
    }

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        Calendar c=Calendar.getInstance();
        int hora=c.get(Calendar.HOUR);
        int minutos=c.get(Calendar.MINUTE);

        return new TimePickerDialog(getActivity(), this, hora, minutos, true);
    }

    @Override
    public void onTimeSet(TimePicker timePicker, int hora, int minutos) {
        GregorianCalendar g=new GregorianCalendar();
        g.set(Calendar.HOUR, hora);
        g.set(Calendar.MINUTE, minutos);
        f.onResultadoHora(g);
    }

    public interface OnHoraSeleccionada{
        public void onResultadoHora(GregorianCalendar hora);
    }
}

```

Al crear un objeto de estos tipos `DialogoFecha`/`DialogoHora` e invocar a su método `show()`, se ejecutará el método `onCreateDialog()` que debe retornar el diálogo. Para crear el diálogo usamos los constructores:

```
new DatePickerDialog (getActivity(), this, año, mes, dia);  
new TimePickerDialog (getActivity(), this, hora, minutos, true);
```

A estos constructores, les pasamos los valores de fecha y hora actuales (año,mes,día) y (hora,minutos) obtenidos la clase `Calendar`. La clase `Calendar` nos proporciona acceso a la fecha y hora actual del dispositivo móvil. Para formar una fecha y una hora a partir de la elección del usuario usamos los métodos de *callback* `on<X>Set` de las interfaces `OnXSetListener`, donde `<X>` puede ser `Time` o `Date`. Estos métodos nos avisan de que el usuario ha definido una nueva fecha y hora en el diálogo y ha retornado con éxito de la selección. En ese momento, se construye un objeto `GregorianCalendar` con los valores seleccionados por el usuario (estos se reciben en los parámetros de la función `On<X>Set`) y se invoca a la función `onResultadoHora`/`onResultadoFecha` de la interfaz correspondiente y que debe implementar la actividad. Estos dos últimos métodos reciben como parámetro un objeto `GregorianCalendar`, subclase de `Calendar`.

ACTIVIDAD: Puedes obtener más información sobre estas dos clases en:

<http://developer.android.com/reference/java/util/Calendar.html>

<http://developer.android.com/reference/java/util/GregorianCalendar.html>

Por último, la actividad principal, tendrá el siguiente código:

```
public class MainActivity extends AppCompatActivity  
    implements DialogoFecha.OnFechaSeleccionada, DialogoHora.OnHoraSeleccionada{  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        ImageButton fecha=findViewById(R.id.imageButton);  
        fecha.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View view) {  
                DialogoFecha d=new DialogoFecha();  
                d.show(getSupportFragmentManager(),"Mi diálogo de fecha");  
            }  
        });  
  
        ImageButton hora=findViewById(R.id.imageButton2);  
        hora.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View view) {  
                DialogoHora d=new DialogoHora();  
                d.show(getSupportFragmentManager(),"Mi diálogo de hora");  
            }  
        });  
    }  
}
```

```

@Override
public void onResultadoFecha(GregorianCalendar fecha) {
    EditText t=findViewById(R.id.fecha);
    t.setText(fecha.get(GregorianCalendar.DAY_OF_MONTH)+"/"+
        (fecha.get(GregorianCalendar.MONTH)+1)+
        "/" + fecha.get(GregorianCalendar.YEAR));
}

@Override
public void onResultadoHora(GregorianCalendar hora) {
    EditText t=findViewById(R.id.hora);
    t.setText(hora.get(GregorianCalendar.HOUR)+":"+
        hora.get(GregorianCalendar.MINUTE));
}
}

```

ACTIVIDAD: Puedes descargarte el código del ejemplo de los recursos del tema: [ficheros_tema2/DatePicker.zip](#)

2.7. Construcción de menús.

Crear elementos de menús en Android es muy sencillo. Tan sólo tienes que definir un archivo de recursos XML dentro de la carpeta “Menu” del proyecto. Con sólo dos etiquetas (*menu* e *item*) puedes hacer cualquier organización jerárquica de menús. La etiqueta <menu> describe un grupo de elementos (ítem) que será cada entrada del menú.

Los menús se suelen colocar en una barra de app para mostrar las acciones más comunes del usuario.

Hay varios tipos de menús:

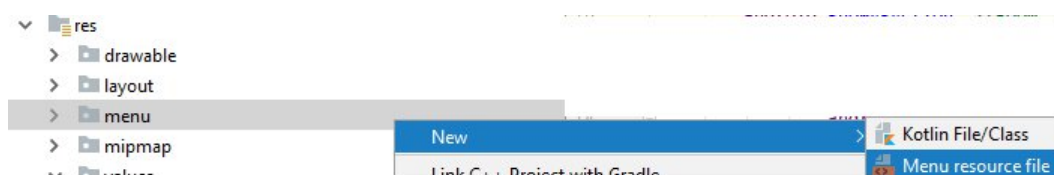
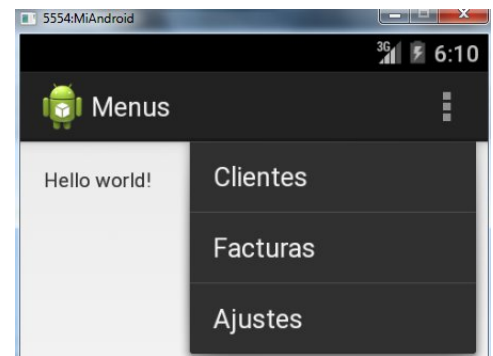
Menús de Opciones o Ajustes (settings): Son los menús que salen en la barra de acción de tu App (o AppBar) y se controlan a través de un método callback que genera automática Android Studio y que se llama *onOptionsItemSelected*. Este método es invocado cuando el usuario selecciona uno de los elementos de este menú.

Menús contextuales: Son los menús flotantes que aparecen cuando un usuario hace un click de manera prolongada en un elemento de la interfaz.

Menús de Pop-up: Visualizan elementos de menú en una lista vertical. Estos menús aparecen anclados al elemento de la IU que provocó su aparición.

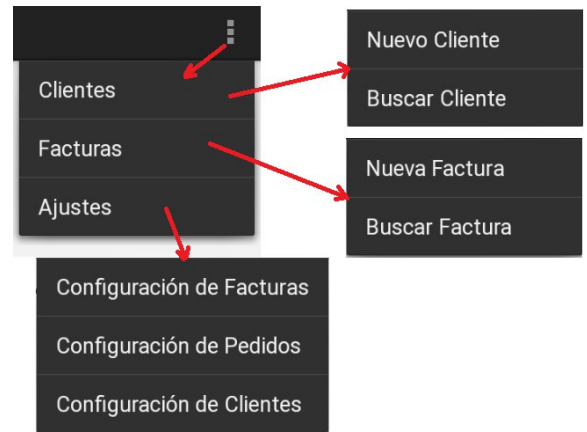
Submenús: Aparecen al seleccionar una opción de menú, reemplazando el menú principal por las opciones del submenú.

Para utilizar un recurso de menú, primero crea la carpeta *menu* dentro de tu proyecto y añade un recurso de tipo “menu”.




Puedes crear un recurso, por ejemplo, **mimenu.xml** y sobrescribir la función `onCreateOptionsMenu` para “inflar” este recurso.

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



Por ejemplo, imagina que quieres hacer una App típica de gestión comercial. Puedes tener como menú de opciones de tu App el siguiente menú:

Los menús en la barra de acción, a partir de android 3.0, aparecen también en la parte superior derecha al presionar el botón de *Overflow* (icono  en la barra de acción), en versiones anteriores aparecen en la esquina inferior izquierda al pulsar la tecla de menú del dispositivo.

El código en XML para este menú es:

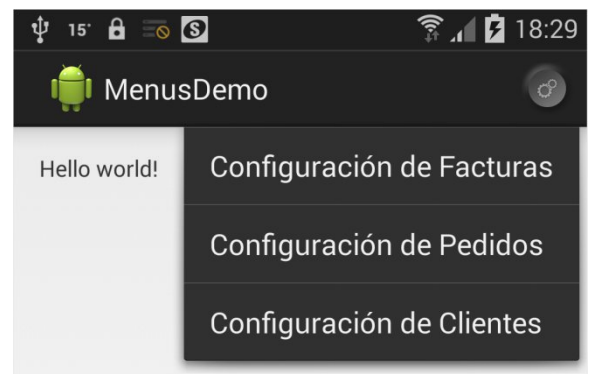
```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools"
      android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      tools:context=".MainActivity" >
    <item android:id="@+id/ajustes"
          android:title="Ajustes"
          android:orderInCategory="100"
          android:icon="@drawable/new_york"
          app:showAsAction="ifRoom">
        <!-- submenú -->
        <menu>
            <item android:id="@+id/confFacturas"
                  android:title="Configuración de Facturas"
                  android:orderInCategory="1"/>
            <item android:id="@+id/confPedidos"
                  android:title="Configuración de Pedidos"
                  android:orderInCategory="2"/>
            <item android:id="@+id/confClientes"
                  android:title="Configuración de Clientes"
                  android:orderInCategory="3"/>
        </menu>
    </item>
    <item android:id="@+id/Clientes"
          android:title="Clientes"
          android:orderInCategory="1">
        <!-- submenú -->
        <menu>
            <item android:id="@+id/NuevoCliente"
                  android:title="Nuevo Cliente"
                  android:orderInCategory="1" />
            <item android:id="@+id/BuscarCliente"
                  android:title="Buscar Cliente"
                  android:orderInCategory="2" />
        </menu>
    </item>
    <item android:id="@+id/Facturas"
          android:title="Facturas"
          android:orderInCategory="2">
        <!-- submenú -->
        <menu>
            <item android:id="@+id/NuevaFactura"
                  android:title="Nueva Factura"
                  android:orderInCategory="1" />
            <item android:id="@+id/BuscarFactura"
                  android:title="Buscar Factura"
                  android:orderInCategory="2" />
        </menu>
    </item>
</menu>
```

Observa cómo cuando dentro de una pareja de etiquetas `<menu>` `</menu>` incluyes un `<item>` se crea una opción y si anidas de nuevo una pareja `<menu>` `</menu>` se crea un submenú.

Las opciones de cada <item> son las siguientes:

- **android:id** Identificador para el elemento de menú. Gracias al *id* puedes luego diferenciar en el código en qué elemento se hizo "Click"
- **android:title** Texto que aparece en el elemento de menú.
- **android:orderInCategory** Orden en el que aparece el elemento de menú dentro de su agrupación.
- **android:icon** Icono para el elemento de menú. Los iconos no aparecerán si tu aplicación se ejecuta en Android 3.0 o mayor. Tan sólo se mostrarán si forman parte de tu ActionBar.
- **android:showAsAction, o app:showAsAction (para AppCompatActivity)** Indica si el elemento aparecerá como un elemento en la barra de Acción. El valor *never*, hará el elemento de menú no se muestre en la barra de acción. Si se especifica *always*, siempre aparecerá y si se especifica *ifRoom* solo aparecerá si hay espacio suficiente en la barra de acción. Por ejemplo, si quisiéramos cierta parte de nuestro menú (por ejemplo, el submenú de configuración) apareciera vinculado a la barra de acción, por ejemplo, mediante un icono, tendríamos que usar las propiedades:

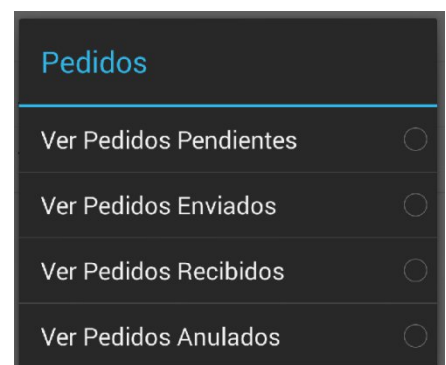
```
<item android:id="@+id/action_settings"
      android:title="Ajustes"
      android:icon="@drawable/sys_settings"
      android:showAsAction="ifRoom"
      android:orderInCategory="100"
      >
  <menu> <!-- submenú -->
    <item android:id="@+id/confFacturas"
          android:title="Configuración de
Facturas"
          android:orderInCategory="1"/>
    <item android:id="@+id/confPedidos"
          android:title="Configuración de Pedidos"
          android:orderInCategory="2"/>
    <item android:id="@+id/confClientes"
          android:title="Configuración de Clientes"
          android:orderInCategory="3"/>
  </menu>
</item>
```



También puedes diseñar menús con opciones seleccionables utilizando la opción <group android:checkableBehavior="modo"> siendo modo *single* (sólo un elemento es seleccionable, tipo RadioButton), *all* (todos los elementos son seleccionables, tipo CheckBox) y *none* (ningún elemento seleccionable).

Por ejemplo:

```
<menu>
  <group android:checkableBehavior="single">
    <item android:id="@+id/PedPend"
          android:title="Ver Pedidos Pendientes"
          android:orderInCategory="1" />
    <item android:id="@+id/PedEnviados"
          android:title="Ver Pedidos Enviados"
          android:orderInCategory="2" />
    <item android:id="@+id/PedRecibidos"
          android:title="Ver Pedidos Recibidos"
          android:orderInCategory="2" />
    <item android:id="@+id/PedAnulados"
          android:title="Ver Pedidos Anulados"
          android:orderInCategory="2" />
  </group>
</menu>
```



2.7.1. Dotando de acción a los elementos de menú

Para responder al click en un elemento de menú, hay que modificar el método `onOptionsItemSelected`:

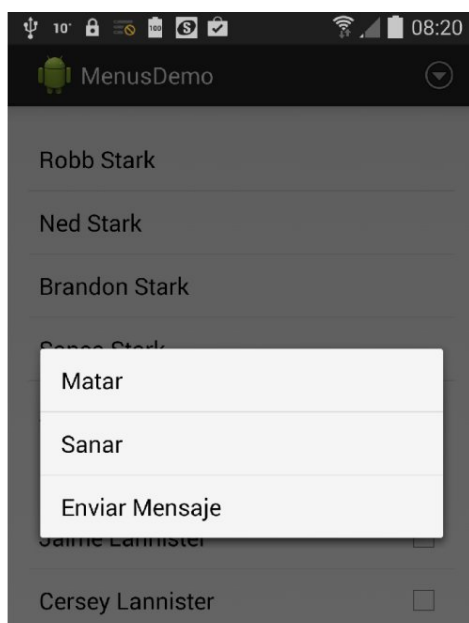
```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();
    if (id == R.id.BuscarCliente) {
        Toast.makeText(getApplicationContext(), "Se ha pulsado Buscar
        Cliente", Toast.LENGTH_LONG).show();
        return true;
    } else if (id == R.id.Clientes) {
        Toast.makeText(getApplicationContext(),
        "Se ha pulsado Cliente", Toast.LENGTH_LONG).show();
        return true;
    } else if (id == R.id.Facturas) {
        Toast.makeText(getApplicationContext(),
        "Se ha pulsado Facturas", Toast.LENGTH_LONG).show();
        return true;
    } else{
        Toast.makeText(getApplicationContext(), "Se ha pulsado otro elemento
        de menu("+item.getTitle()+")", Toast.LENGTH_LONG).show();
        return true;
    }

    return super.onOptionsItemSelected(item);
}
```

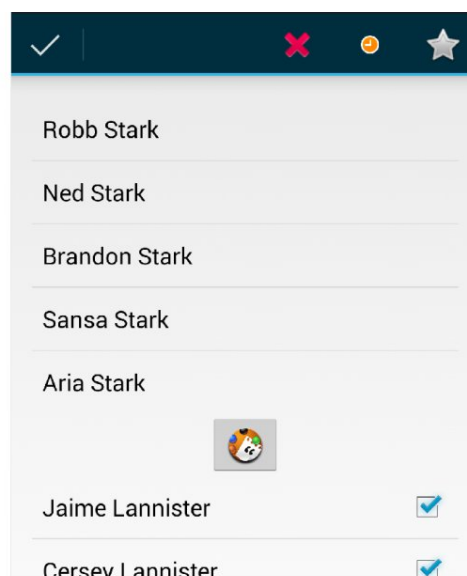
Este método recibe como parámetro el elemento seleccionado `MenuItem`, y a partir de ahí podemos discriminar qué elemento se seleccionó. Por ejemplo, con el método `getItemId()` del objeto `item` nos quedamos con el identificador y después podemos comparar con un evaluador múltiple `switch` qué elemento de menú se pulso. El método exige que se retorne el valor `true` si se procesó correctamente el click del menú.

2.7.2. Menús contextuales

Un menú contextual ofrece opciones que afectan a un elemento de la interfaz, por ejemplo, un elemento de una lista o una imagen. Hay dos formas de hacer menús contextuales:



Menú contextual

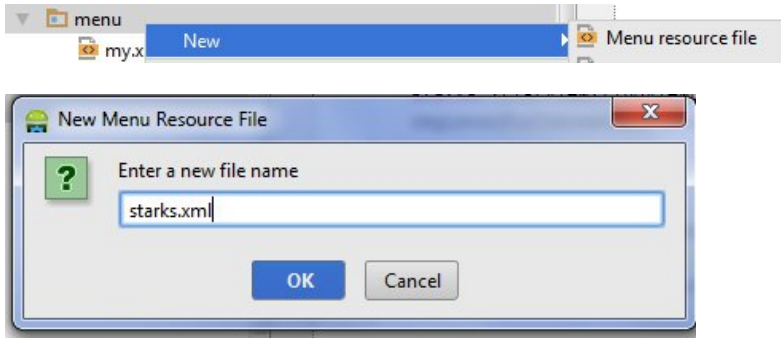


Menú en Contextual Action Bar (CAB)

El primero se utiliza cuando el usuario realiza una pulsación larga (long click) en un elemento de la interfaz y el segundo visualiza una barra de acción contextual, en inglés Contextual ActionBar (CAB). Esta CAB se puede programar para que aparezca también cuando se produce un click largo en un elemento de la IU o, por ejemplo, cuando se selecciona uno o varios elementos de una lista.

A continuación, mostramos en un caso práctico los dos tipos de menús. Primero el menú flotante:

¿Te acuerdas de lo que era Inflar un View? Consistía en, a partir de la definición de un fichero de recursos xml crear una vista. Pues eso mismo es lo que hay que hacer, definir un recurso de menú, para luego “inflarlo”.



Después hay que registrar el componente al que se le va a asociar el menú contextual, mediante el método `registerForContextMenu()`. Después hay que crear (override) el método `onCreateContextMenu()` de Actividad e inflar el menú que hayas creado en xml.

Después, para responder a la selección del elemento del menú, se programa el método `onContextItemSelected()`.

Te mostramos a continuación el siguiente caso práctico. Vamos a crear una App para jugar con los personajes de una famosa serie de televisión. En esta aplicación tenemos dos listas de personajes. A la primera lista (con los personajes de la familia Stark) le vamos a vincular un menú contextual flotante y a la segunda lista (con los personajes de la familia Lannister) le vamos a asociar un menú contextual de Acción, tal y como puedes ver en la primera figura de esta sección.

2.7.2.1. Creación del menú contextual

Este es el fichero de menú xml que vamos a crear:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:icon="@android:drawable/ic_delete"
        android:title="Matar"
        android:id="@+id/matar"></item>
    <item android:icon="@android:drawable/ic_menu_edit"
        android:title="Sanar"
        android:id="@+id/sanar"></item>
    <item android:icon="@android:drawable/sym_call_incoming"
        android:title="Enviar Mensaje"
        android:id="@+id/enviarmensaje"></item>
</menu>
```

Primero hay que registrar la lista como la asociada al menú contextual, esto lo puedes hacer con la función `registerForContextMenu(lista)` en el método `onCreate` de la actividad:

```
//Creamos lista de starks para el menú contextual
starks=(ListView)findViewById(R.id.listaStarks);
registerForContextMenu(starks);
```

Después, tienes que implementar en la actividad el método `onCreateContextMenu()` para inflar el menú desde el archivo de recursos xml:

```

@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo)
{
    MenuInflater m=getMenuInflater();
    m.inflate(R.menu.starks,menu);
    super.onCreateContextMenu(menu, v, menuInfo);
}

```

Finalmente, tienes que implementar el método *onContextItemSelected()* para responder a los eventos del menú contextual. Este método recibirá como parámetro el elemento de menú pulsado *item*. A través de este parámetro puedes obtener, con el método *getMenuInfo()* de *item* puedes obtener un objeto *AdapterContextMenuInfo* con el que poder conocer sobre qué elemento de la *listView* fue pulsado el menú contextual.

```

@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterView.AdapterContextMenuInfo info =
        (AdapterView.AdapterContextMenuInfo) item.getMenuInfo();
    int itemId = item.getItemId();
    if (itemId == R.id.matar) {
        Toast.makeText(getApplicationContext(), "Hemos matado a " +
            starks.getItemAtPosition(info.position), Toast.LENGTH_LONG).show();
        return true;
    } else if (itemId == R.id.sanar) {
        Toast.makeText(getApplicationContext(), "Hemos sanado a " +
            starks.getItemAtPosition(info.position), Toast.LENGTH_LONG).show();
        return true;
    } else if (itemId == R.id.enviarmensaje) {
        Toast.makeText(getApplicationContext(), "Le hemos enviado un mensaje a " +
            starks.getItemAtPosition(info.position), Toast.LENGTH_LONG).show();
        return true;
    }
    Toast.makeText(getApplicationContext(), "Le hemos hecho otra cosa a " +
        starks.getItemAtPosition(info.position), Toast.LENGTH_LONG).show();
    return true;
}

```

2.7.2.2. Creación del menú action bar contextual

Crear el menú contextual en la barra de acción, exige un poco más de código, pero el fundamento es el mismo. Primero, crear el menú. Ten en cuenta que en la barra de acción si se muestran los iconos, al contrario que en un menú normal, por tanto, la adición de la propiedad *android:icon* toma especial importancia:

Primero creamos el menu en el fichero *lannisters.xml*

```

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

    <item android:icon="@android:drawable/ic_delete"
        android:title="Aniquilar"
        android:id="@+id/aniquilar"
    ></item>

    <item android:icon="@android:drawable/presence_away"
        android:title="Encerrar"
        android:id="@+id/encerrar"
    ></item>

    <item android:icon="@android:drawable/btn_star"
        android:title="Salvar"
        android:id="@+id/salvar"
    ></item>

</menu>

```

Después, dentro del método *onCreate* de la actividad, creamos la segunda lista de elementos a través de un *ArrayAdapter<String>* de tal manera que sean todos seleccionables. Observa la última instrucción que registra el listener de Click en los elementos de la lista (hay que implementar en la actividad la interfaz *ListView.OnItemClickListener*, para que cuando se seleccione en una de las opciones de la lista, aparezca la Action Bar Contextual.

```
//creamos lista de lannisters (seleccionable múltiple) para el
//Action Mode Context Menu
ListView listaLannisters=(ListView) findViewById(R.id.listaLannisters);

ArrayAdapter<String> adaptador=new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_multiple_choice,
    getResources().getStringArray(R.array.lannisters));

listaLannisters.setAdapter(adaptador);
listaLannisters.setOnItemClickListener(this);
```

El gran truco para mostrar la barra de acción contextual es la creación de dos objetos, uno de tipo *ActionMode*, que representa el modo contextual en la barra de acción y otro de tipo *ActionMode.Callback*, que es la interfaz que se debe programar para responder a sus eventos y que tiene su propio ciclo de vida también:

```
public void onItemClick(AdapterView<?> p, View v, int position, long id){
    mActionMode = MainActivity.this.startActionMode(mActionModeCallback);
    v.setSelected(true);
}
```

```

//Menú ActionMode
ActionMode mActionMode;

private ActionMode.Callback mActionModeCallback = new ActionMode.Callback() {
    // Called when the action mode is created; startActionMode() was called
    @Override
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
        // Inflate a menu resource providing context menu items
        MenuInflater inflater = mode.getMenuInflater();
        inflater.inflate(R.menu.lannisters, menu);
        return true;
    }
    // Called each time the action mode is shown. Always called after
    // onCreateActionMode, but
    // may be called multiple times if the mode is invalidated.
    @Override
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
        return false; // Return false if nothing is done
    }
    // Se llama a este método cuando se ha pulsado en la lista de los lannisters
    @Override
    public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
        int itemId = item.getItemId();
        if (itemId == R.id.aniquilar) { //hay que crear un Aniquilar() para
            //recorrer todos los elementos seleccionado (checked) en la listView
            Toast.makeText(getApplicationContext(), "Hemos aniquilado a algún
                Lannister", Toast.LENGTH_LONG).show();
            return true;
        } else if (itemId == R.id.encerrar) {
            Toast.makeText(getApplicationContext(), "Hemos encerrado a algún
                Lannister", Toast.LENGTH_LONG).show();
            return true;
        } else if (itemId == R.id.salvar) {
            Toast.makeText(getApplicationContext(), "Hemos salvado a algún Lannister",
                Toast.LENGTH_LONG).show();
            return true;
        } return false;
    }

    // Called when the user exits the action mode
    @Override
    public void onDestroyActionMode(ActionMode mode) {
        mActionMode = null;
    }
};

```

En este código tienes que fijarte en la creación de la interfaz: El método `onCreateActionMode()` infla el menú `lannisters.xml` adjuntándose a la barra de acción. El método `onActionItemClicked()` discrimina el elemento del menú que se seleccionó y se actúa en consecuencia.

ACTIVIDAD: Descarga el código de esta aplicación desde los recursos del tema: `ficheros_tema2/MenuDemos.zip`

2.8. EL ACTIONBAR

Ya has visto en la última sección la utilidad de la Action Bar, aunque aplicado a menús contextuales. La barra de acción es una de las principales características de tu App. Está visible en todo momento mientras tu aplicación está visible y te da la posibilidad de dotar a tu aplicación de una identidad (hasta ahora lo hemos hecho con un icono), pero también te permite situar iconos para acciones importantes para tu aplicación (por ejemplo, buscar o crear algo importante) y permite que el usuario realice una navegación consistente por toda la aplicación, por ejemplo, a través de pestañas o *tabs*.



A partir del API de nivel 11, se incluye en todas las actividades que utilicen el tema Theme.Holo en el diseño de la actividad (Este es el tema por defecto). Si no quieres incluir un ActionBar en tu App, tan sólo tienes que cambiar el tema de la actividad a Theme.Holo.NoActionBar en el fichero styles.xml:

```
<!-- Base application theme. -->
<style name="AppTheme" parent="android:Theme.Holo.NoActionBar">
    <!-- Customize your theme here. -->
</style>
```

No obstante, no recomendamos que prives a tus usuarios de las bondades de la ActionBar, por tanto, a partir de ahora, te pondremos ejemplos de cómo utilizarla.

ACTIVIDAD: Puedes utilizar y descargar iconos típicos de Android en <http://developer.android.com/design/style/iconography.html>

Para añadir la barra de acción tan sólo hay que modificar el método *onCreateOptionsMenu()* e “inflar” el archivo de recursos de menú xml que hayas creado. Cuando vayas a crear el menú recuerda que, al igual que en un menú contextual, puedes solicitar que aparezca directamente en la barra de acción alguno de los elementos del menú con la opción *showAsAction="ifRoom"* o *showAsAction="always"*.

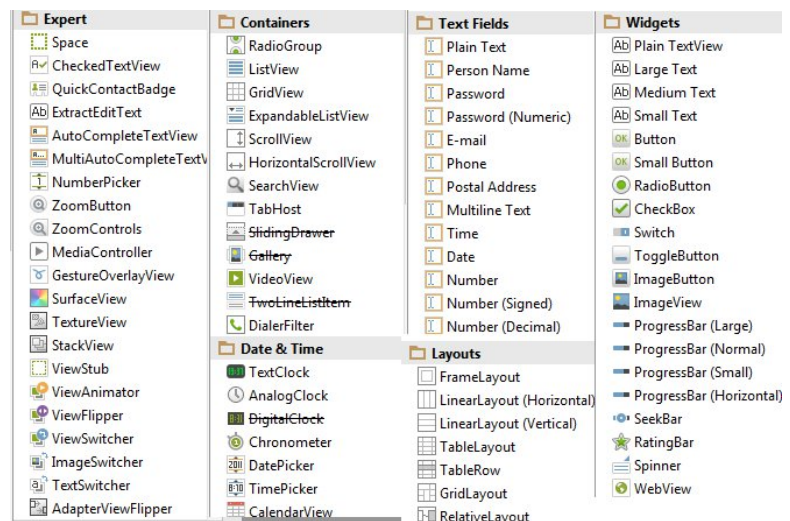
```
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu items for use in the action bar
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu_action_bar, menu);
    return super.onCreateOptionsMenu(menu);
}
```

A continuación, para responder a los eventos de click de los elementos que has colocado en la Action Bar debes escribir tu código en el método *onOptionsItemSelected()*.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle presses on the action bar items
    int itemId = item.getItemId();
    if (itemId == R.id.borrar) {
        Toast.makeText(getApplicationContext(),
            "Se ha pulsado borrar", Toast.LENGTH_LONG).show();
        return true;
    } else if (itemId == R.id.edit) {
        Toast.makeText(getApplicationContext(),
            "Se ha pulsado borrar", Toast.LENGTH_LONG).show();
        return true;
    } else if (itemId == R.id.llamar) {
        Toast.makeText(getApplicationContext(),
            "Se ha pulsado borrar", Toast.LENGTH_LONG).show();
        return true;
    }
    return super.onOptionsItemSelected(item);
}
```

2.9. OTROS WIDGETS PARA TU IU

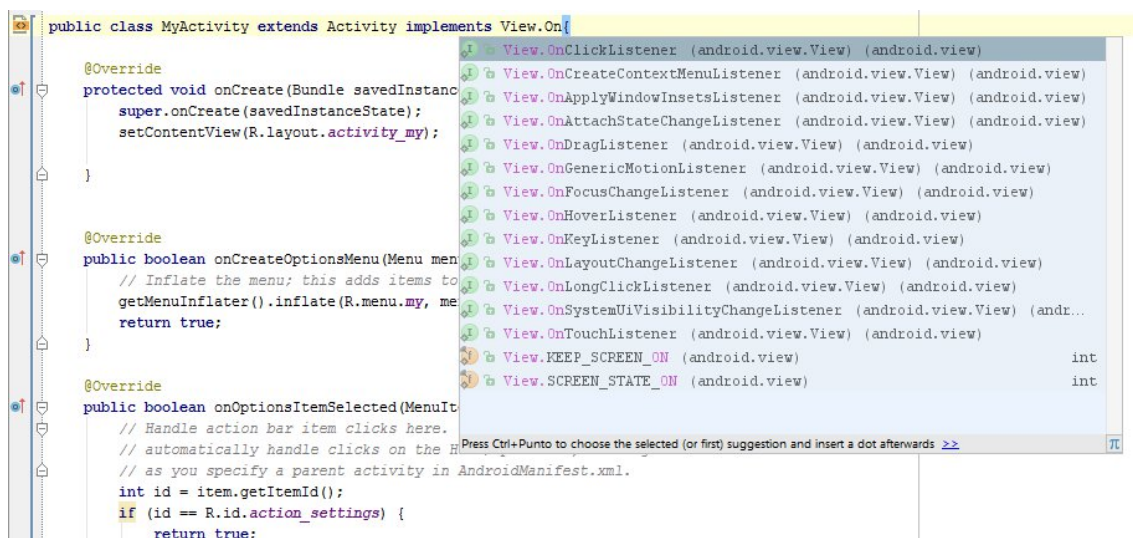
Existen para tu uso y disfrute un montón de widgets y componentes de interfaz de usuario que pueden ayudarte a crear una App muy profesional. Échale un ojo por ejemplo a las barras de desplazamiento, webviews, Menús Pop-up y el uso de acciones drag-and-drop. Herramientas como el progress-bar pueden dar a tu App un toque profesional que le encantará al usuario cuando programes en tu aplicación procesos que mueven la barra de progreso a toda velocidad. Calendarios, botones de zoom, controladores de medios son widgets que facilitarán tu vida y que facilitarán el éxito de tu



App. Diferentes tipos de campos de textos, de barras de ítems, etc. Todo la experiencia de los programadores de Google y de Linux puesta a tu disposición totalmente gratis y agrupados en carpetas en la paleta de widgets para que puedas utilizarlos de forma sencilla y millones de páginas de documentación y código para que no tengas que aportar mucho esfuerzo en tu aprendizaje. Prueba también los controles de tipo Tab a través de la “Split Action Bar” para que tu App pueda manejar varias actividades cada una con su correspondiente Tab.

2.10. MÁS FUNCIONES DE CALLBACK.

Ya conoces las funciones de callback más básicas OnClick, OnLongClick, OnItemSelectedListener, etc. Todas ellas implementadas mediante interfaces:



Tienes una larga lista de interfaces con sus funciones de callback respectivas para programar en tu dispositivo Android. Es tu deber como programador, familiarizarte con todas ellas y probablemente no encuentres un texto de menos de 500 páginas que te cuente todas y cada una de ellas. Como este texto no pretende ser una guía de referencia sino una guía para ayudarte a comenzar a programar en Android, te redigirimos a la documentación oficial de Android para aprender todas estas interfaces y funciones de callback.

2.11. ESTILOS, TEMAS Y MATERIAL DESIGN

Un estilo es una colección de propiedades que especifican cómo se dibujan una vista (View) o ventana. Un estilo puede especificar propiedades como aletura, márgenes, padding, color de fuente o tamaño de fuente. Un estilo se define en un fichero XML que está separado del Layout.

En Android los estilos comparten una filosofía similar a las hojas CSS en diseño web. Te permiten separar el diseño del contenido, por ejemplo, usando un estilo puedes coger un diseño XML como este:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="@string/hola" />
```

Y convertirlo en

```
<TextView
    style="@style/CodeFont"
    android:text="@string/hola" />
```

Todos los atributos referentes al estilo se han quitado de la definición XML e introducidos en una definición de estilo llamada CodeFont, que se aplica con atributo style. Para crear este estilo y otros más, puedes crear un fichero XML en la carpeta res/values de tu proyecto con el siguiente contenido:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CodeFont" parent="@android:style/TextAppearance.Medium">
        <item name="android:layout_width">fill_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

Un tema (theme) es un estilo aplicado a toda una actividad o aplicación, en lugar de a un widget en particular. Cuando un estilo se aplica a un tema, cada widget de la actividad tomará los valores de este estilo. Para aplicar un tema a una aplicación, hay que modificar el archivo de manifiesto AndroidManifest.xml:

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

Si solo quieres aplicarlo a una actividad, establece la propiedad android:theme en la definición de la actividad:

```
<activity android:theme="@android:style/Theme.Dialog">
```

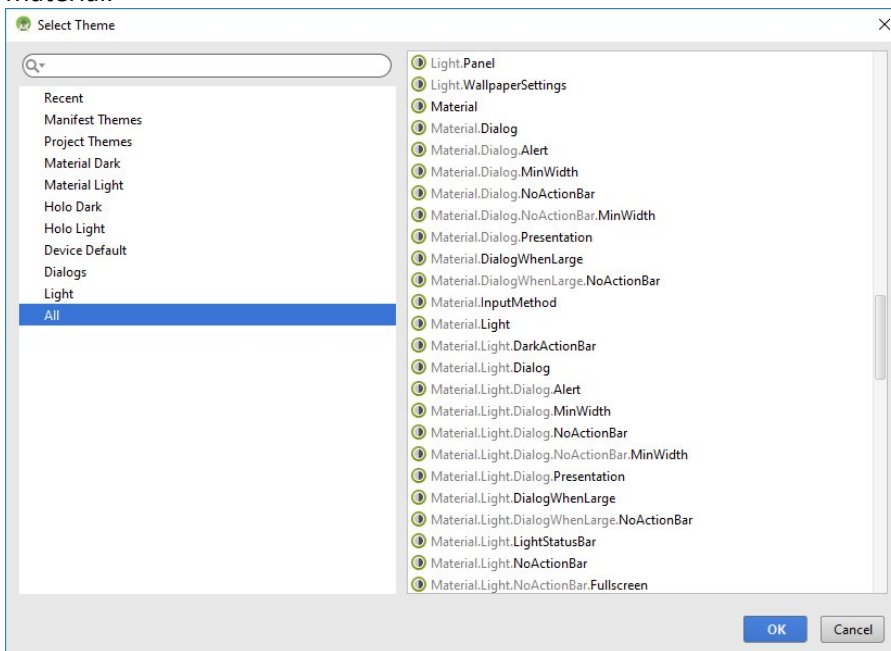
Android tiene temas predefinidos, por ejemplo, si quieres que todo el fondo de tu actividad sea transparente, puedes usar el tema Translucent:

```
<activity android:theme="@android:style/Theme.Translucent">
```

A su vez, también puedes personalizar un tema añadiendo como padre (propiedad parent) el tema que quieras personalizar:

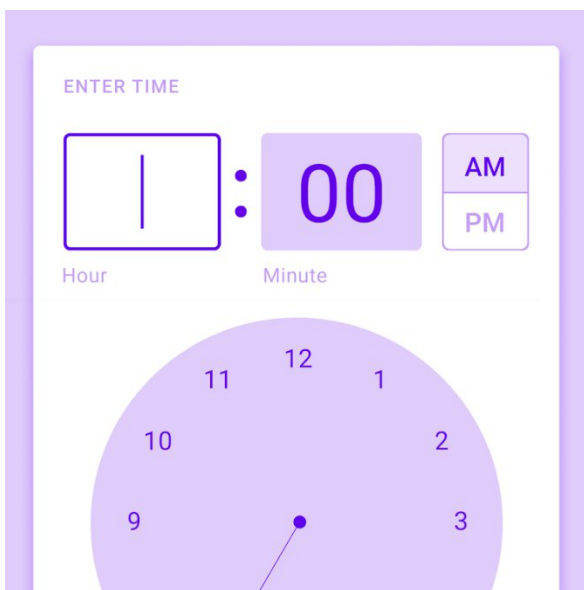
```
<color name="custom_theme_color">#b0b0ff</color>
<style name="CustomTheme" parent="android:Theme.Light">
    <item name="android:windowBackground">@color/custom_theme_color</item>
    <item name="android:colorBackground">@color/custom_theme_color</item>
</style>
```

Con cada nivel de API, Android suele incorporar nuevos temas, por ejemplo, en la versión 21 incorporó los temas Material.



Con los temas Material, se incorporó una nueva filosofía, un concepto de diseño llamado Material Design:

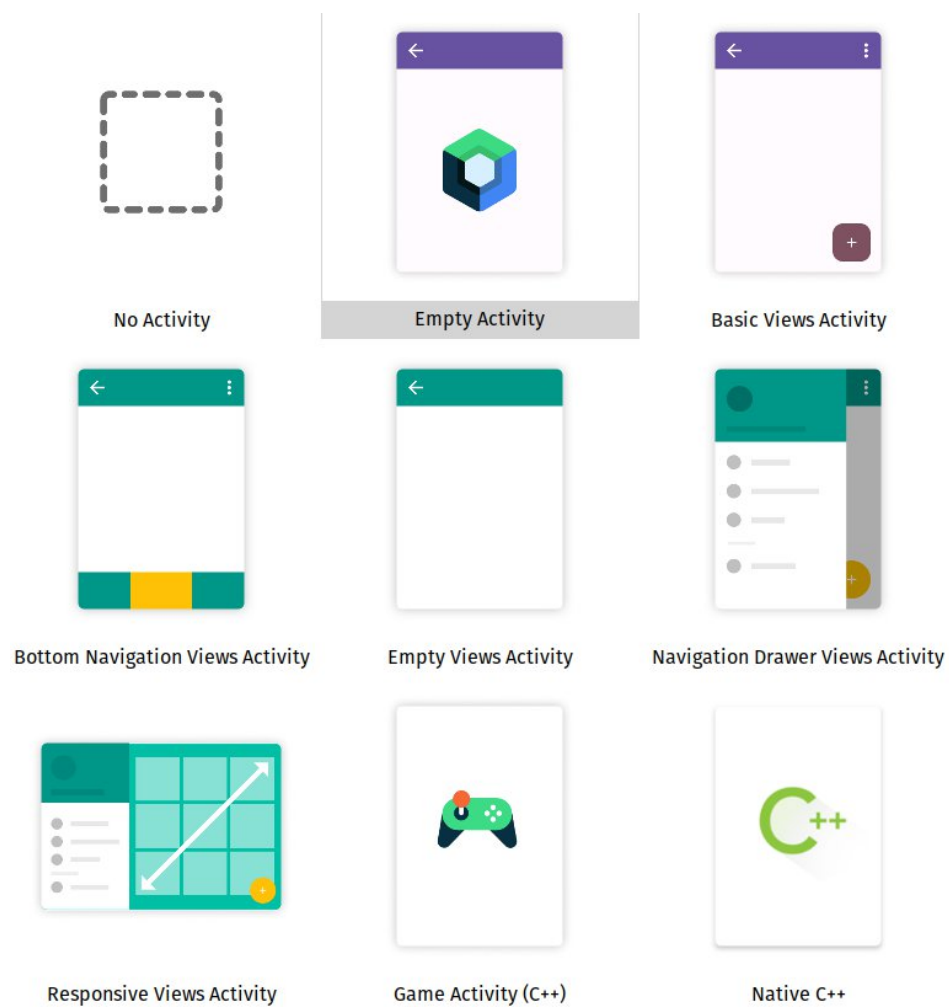
Material Design se escapa del objetivo del curso, pero es importante que lo conozcas. Es un conjunto de pautas enfocadas al diseño utilizado en Android, pero también en la web y en cualquier plataforma. En concreto para diseñar aplicaciones en Android.



Echa un ojo a esta filosofía propuesta por google y a su documentación en <https://developer.android.com/training/material/get-started.html?hl=es>

2.12. OTRAS PLANTILLAS PARA ACTIVIDADES

Puedes usar cualquiera de las plantillas que te proporciona Android Studio al crear un nuevo proyecto, por ejemplo “Navigation Drawer Activity” para realizar una app con un panel desplegable a la derecha estilo **GMAIL**.



Prueba con estas plantillas para elegir las actividades que más se adapten a la función de las apps que quieras crear.