
	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD COMPETENCIA					

Tema 2

Elementos de la interfaz de usuario

Índice

1.	Componentes de una aplicación Android.	1
2.	Activity.	2
3.	Log.	4
4.	<i>Toast</i>	4
5.	Layouts	5
5.1.	LinearLayout	5
5.2.	ConstraintLayout	7
6.	Acceso a elementos de la interfaz de usuario	11
7.	Componentes estándar de la interfaz gráfica de Android	12
8.	Eventos.	15
9.	Radiogroup/Radiobutton	17
10.	Estructura de una aplicación.	19
10.1.	Archivo AndroidManifest.xml	19
10.2.	Carpeta java	19
10.3.	Recursos.	20
10.3.1.	Strings	20
10.3.2.	Dimensiones.	21
10.3.3.	Colores	22
10.3.4.	Otros recursos	22
10.3.5.	Temas claro y tema oscuro	23
10.3.6.	Clase R.	25
10.4.	Assets.	25
11.	Apéndice I: RelativeLayout	26

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

1. Componentes de una aplicación Android.

Toda aplicación Android está formada por alguno de los siguientes cuatro componentes¹. Cada uno de ellos tiene un propósito diferente y un ciclo de vida propio.

- **Activity (actividad):** representa una pantalla individual de la interfaz de usuario de una aplicación. Una aplicación puede contener, y de hecho normalmente contendrá, varias Activities.
- **Servicios:** es el mismo concepto que ya conocéis pero llevado al mundo de dispositivos móviles. Esencialmente se encargan de realizar tareas en segundo plano y se encargan de comunicar los distintos procesos. No tienen interfaz gráfica y no interaccionan con el usuario.
- **Content providers (proveedores de contenido):** mecanismo para almacenar y compartir información entre aplicaciones.
- **Broadcast Receiver (receptores de emisiones):** componente encargado de gestionar eventos y mensajes que suceden en el sistema.

Ejemplo

El programa de envío de SMSs dispone de varias Activities: lista de contactos, escritura del mensaje, mostrar mensajes, ...

Está pendiente a la llegada de nuevos SMS a través del BroadcastReceiver que le informa que ha llegado un mensaje nuevo.

Al llegar un mensaje un servicio en segundo plano se encarga de descargar dicho mensaje, almacenarlo y notificarlo.

Se puede acceder a la lista de contactos gracias a que la aplicación Contactos dispone de un ContentProvider donde almacena los datos de los contactos que pueden mostrar luego otras aplicaciones

Actividad

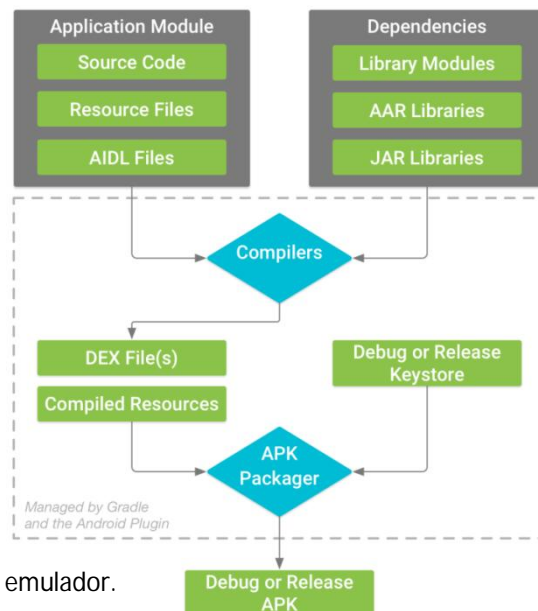
Indica otros Content Providers que consideres que están presentes en prácticamente todos los dispositivos Android.

Iremos profundizando en estos elementos a lo largo del curso.

También es importante conocer el proceso de construcción y ejecución de una aplicación. Este proceso se puede ver en el siguiente esquema de Google².

- Primero se crea, por un lado, un archivo DEX con el código fuente compilado junto con las dependencias que posee y por otro se compilan todos los recursos que utilice la aplicación: sonidos, imágenes, ...
- Después el empaquetador de APKs crea, uniendo los elementos anteriores, el paquete Android con extensión apk.
- Por último el paquete creado, en el paso anterior, se debe firmar antes de poderse ejecutar ya sea en un dispositivo o en el emulador.

Android Studio este se encarga de realizar todos los pasos anteriores. Mientras estamos desarrollando una aplicación Android Studio firma, de forma automática, el paquete APK, con un certificado de prueba. Si queremos crear una versión para distribuir a través de Google Play se deberá firmar con un firma digital única³.



¹ <https://developer.android.com/guide/components/fundamentals.html?hl=es#Components>

² <https://developer.android.com/studio/build?hl=es>

³ <https://developer.android.com/studio/publish/app-signing?hl=es#studio>

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

2. Activity⁴

Una *Activity* es una pantalla de una aplicación que con la que los usuarios pueden interactuar. Por norma general una aplicación constará de varios *activities* conectados entre sí. En una aplicación normal existirá como mínimo la pantalla principal que suele llamarse (si no se modifica su nombre, lo que no es recomendado) *MainActivity.java*.

Cada activity que creamos debe estar reflejada en el archivo *AndroidManifest.xml* que veremos más adelante.

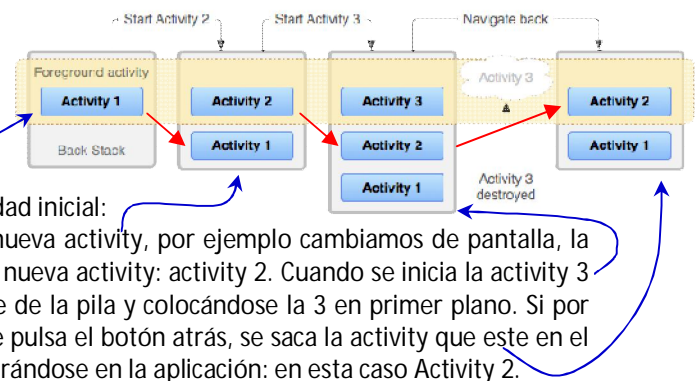
Por ejemplo en un juego podría haber las siguientes *activities*: el menú de inicio, las opciones, la pantalla de juego, los records, etc.

Se debe pensar además en cada *activity* como una tarea o proceso que se almacena en la pila de actividades⁵ o *back stack* con una estructura *LIFO* (*Last In First Out*). El ejemplo típico de una estructura *LIFO* es una pila de platos. El último que se coloca en la cima es el primero que se coge. Así cada vez que se pasa de una *activity* a otra, la que se suspende o abandona pasa a dicha pila.

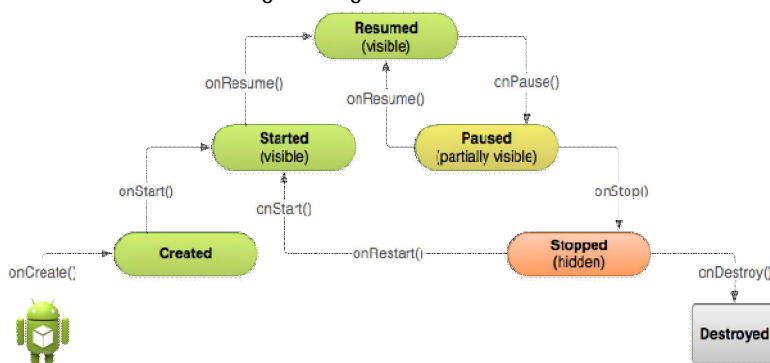
En la imagen se puede ver dos zonas: la zona amarilla representa la *activity* que se está mostrando por pantalla mientras que la zona gris es la pila de actividades que inicialmente está vacía.

Al arrancar la aplicación se muestra por pantalla la actividad inicial:

la Activity 1 estando la pila vacía. Cuando se inicia una nueva actividad, por ejemplo cambiamos de pantalla, la activity actual pasa a la pila y se sitúa en primer plano la nueva actividad: activity 2. Cuando se inicia la actividad 3 pasa exactamente lo mismo pasando la actividad 2 al tope de la pila y colocándose la 3 en primer plano. Si por cualquier motivo se destruye la actividad 3, por ejemplo se pulsa el botón atrás, se saca la actividad que este en el tope de la pila situándose en primer plano, es decir, mostrándose en la aplicación: en esta caso Activity 2.

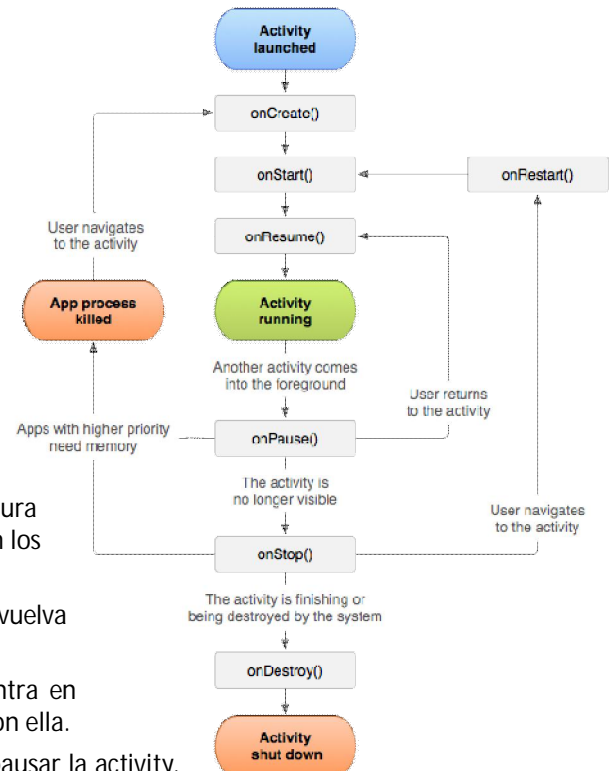


Una actividad sigue el siguiente ciclo de vida^{6 7}:



Se pueden ver los siguientes métodos callbacks que son métodos que responden a eventos de cambio de estado.

- *onCreate*: se desarrolla la inicialización del *Activity*: se restaura un estado anterior si lo hay, se inicializan variables, se inician los componentes de la interfaz gráfica, ...
- *onStart*: se ejecuta *onStart* justo antes que la actividad se vuelva visible. Se pueden realizar preparativos finales.
- *onResume*: tras ejecutar *onResume* la aplicación se encuentra en ejecución siendo visible y pudiendo, el usuario interactuar con ella.
- *onPause*: se ejecuta siempre después de *onResume* y tras pausar la activity.




⁴ <https://developer.android.com/guide/components/activities.html?hl=es>

⁵ <https://developer.android.com/guide/components/activities/tasks-and-back-stack?hl=es>

⁶ <https://developer.android.com/guide/components/activities/activity-lifecycle?hl=es>

⁷ <https://developer.android.com/guide/components/activities/intro-activities?hl=es#java>

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Esta aun puede verse en primer plano pero tapada parcialmente por otra *activity* que ha sido llamada (típico en diálogos o alertas). Hay que tener en cuenta que en situaciones de muy poca memoria Android puede eliminar la *activity* de la memoria.

- *onStop*: se ejecuta este método cuando la *activity* se encuentra parada en segundo plano no siendo visible. El sistema puede cerrarla si se requiere por requerimientos de memoria. Es en este método donde se deben realizar operaciones de guardado de datos, acceso a base de datos, ... y no en *onPause*.
- *onRestart*: este método es invocado cuando la actividad pasa de estar detenida a iniciarse restaurando el estado que tenía en el momento en que se detuvo.
- *onDestroy*: se ejecuta antes de que la *activity* sea destruida. Suele usarse para liberar recursos.


Se debe tener en cuenta que estos métodos son controlados tanto por el programador como por Android por lo que pueden ser ejecutados en cualquier momento.

El código fuente siguiente representa la clase *MainActivity.java* con sus métodos *callback* sobrescritos:

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) { // La actividad se está creando.
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    @Override
    protected void onStart() { // La actividad está a punto de hacerse visible.
        super.onStart();
    }
    @Override
    protected void onResume() { // La actividad se ha hecho visible, se ha reanudado.
        super.onResume();
    }
    @Override
    protected void onPause() { // Otra actividad está tomando el foco y esta actividad va a ser "pausada".
        super.onPause();
    }
    @Override
    protected void onStop() { // La actividad ya no es visible y ahora está "parada"
        super.onStop();
    }
    @Override
    protected void onRestart() { // La actividad, que estaba parada, está a punto de iniciarse.
        super.onRestart();
    }
    @Override
    protected void onDestroy() { // La actividad está a punto de ser destruida.
        super.onDestroy();
    }
}
```

El método *onCreate* tiene como parámetro un objeto del tipo *Bundle* que veremos más adelante. Dicho parámetro nos va a servir para intercambiar información entre distintas *activities*. Se debe llama, mediante *super*, a su clase padre pasando los mismos parámetros para que la inicialización sea correcta.

Luego se ejecuta el método *setContentView(R.layout.activity_main)* que se encarga de inicializar la interfaz de usuario de la *activity* a partir del identificador de un archivo XML de recursos (en este caso *R.layout.activity_main* que apunta al archivo *res/layout/activity_main.xml*). Se debe ejecutar antes acceder a los componentes de la interfaz gráfica del *Activity*, sino se generará una *NullPointerException*.

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

3. Log⁸

Log es una clase que proporciona una serie de métodos estáticos para visualizar mensajes de depuración en el *logcat*.

Existen distintos métodos según el tipo de información que se desea mostrar. Estos métodos toman dos parámetros: una etiqueta del error, que suele mostrar el lugar donde se ejecuta el método, y un mensaje asociado al mismo.

Es habitual definir la constante *TAG* dentro de la clase para utilizarlo con estos métodos:

```
private static final String TAG = "MainActivity";
```

Y luego usar dicha constante en la llamada a los métodos de *Log*.

```
Log.i(TAG, "Fallo en la aplicación");
```

Tipo de log	Método	Tipo de log	Método
Verbose	Log.v	Warning	Log.w
Debug	Log.d	Error	Log.e
Information	Log.i	WTF	Log.wtf

Ejercicio 1

Añade los siguientes mensajes de log en sus correspondientes métodos:

- mensaje "método onCreate" de tipo Verbose dentro del método onCreate.
- mensaje "método onStart" de tipo Debug dentro del método onStart.
- mensaje "método onResume" de tipo Info dentro del método onResume.
- mensaje "método onPause" de tipo Error dentro del método onPause.
- mensaje "método onStop" de tipo Info dentro del método onStop.
- mensaje "método onRestart" de tipo Warning dentro del método onRestart.
- mensaje "método onDestroy" de tipo Verbose dentro del método onDestroy.

Ejecuta la aplicación, sal de la aplicación, vuelve a ella, gira la pantalla, muestra la lista de aplicaciones,... Observa el logcat ¿Qué sucede?.

Filtra los mensajes para que solo muestren los de tipo Info. O los que tengan una cadena de texto.

4. Toast^{9 10}

Un Toast permite mostrar un mensaje por pantalla durante unos segundos y que no interfiere con las acciones que esté realizando el usuario ya que no se puede interacción con él.

Para ver cómo funciona podemos introducir el siguiente en el método onCreate:

```
Toast toast=Toast.makeText(getApplicationContext(), "Prueba de Toast", Toast.LENGTH_LONG);
toast.show();
```

En el ejemplo anterior se define el objeto *toast* usando para ello el método estático *makeTest* y los tres parámetros requeridos: el contexto de la actividad, el mensaje que se desea mostrar y el tiempo que se mostrará el *toast* por pantalla. Este tiempo puede tomar uno de los valores siguientes: *LENGTH_LONG* o *LENGTH_SHORT*.

Luego, mediante el método *show*, se muestra el *Toast* por pantalla.

Si únicamente se quiere mostrar el *Toast* por pantalla y no reutilizar el objeto se puede usar directamente:

```
Toast.makeText(getApplicationContext(), "Prueba de Toast", Toast.LENGTH_LONG).show();
```


Si se desea que aparezca en otra posición de la pantalla se puede usar el método *setGravity*. Este método toma como parámetros una posición, o una combinación de estas, y un desplazamiento en el eje X y en el eje Y. Se ha de ejecutar antes del método *show*.

```
toast.setGravity(Gravity.CENTER|Gravity.LEFT, 0, 10);
```

⁸ <https://developer.android.com/reference/android/util/Log.html>

⁹ <https://developer.android.com/guide/topics/ui/notifiers/toasts.html#java>

¹⁰ <https://developer.android.com/reference/android/widget/Toast.html>

	RAMA:	Informática	CICLO:	Desenvolvemto de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

5. Layouts¹¹

Son contenedores que no se visualizan y que controlan la manera en que se colocan y distribuyen los componentes que se insertan en su interior. Un *layout* puede tener otros *layouts* anidados dentro de él.

Entre otros existen los siguientes layouts:

- **ConstraintLayout**: utiliza para su funcionamiento "*constraints*" que permiten anclar unos componentes a otros, a bordes de elemento padre, ...
- **LinearLayout**: apila los componentes contenidos en él en una sola fila o columna dependiendo de su dirección.
- **FrameLayout**: suele usarse cuando se desea que un solo componente ocupe todo el *layout* (puede modificarse este comportamiento con la propiedad *layout_gravity*).
- **TableLayout**: en un *layout* que coloca los componentes que posea en filas y columnas imitando una tabla.
- **TableRow**: coloca sus componentes que posea de forma horizontal. Se debe usar como un hijo de **TableLayout**.
- **RelativeLayout**: organiza los componentes que están situados dentro de él de forma relativa a sus hermanos o a su padre. Ha sido sustituido por **ConstraintLayout** manteniéndose por compatibilidad.

5.1. LinearLayout¹²

Es un *view group* que va apilando todos los componentes que se insertan en él en una única dirección ya sea esta horizontal o vertical. La dirección se establece mediante el atributo XML *orientation*.

- **LinearLayout** vertical los componentes que posea se apilan uno encima de otro, en filas.
- **LinearLayout** horizontal los componentes que posea se distribuyen de forma horizontal, en columnas, de izquierda a derecha.

Todos los componentes de *Android*: botones, *textviews*, ... así como los *Layouts* deben definir, de forma obligatoria, las propiedades ancho (*layout_width*) y alto (*layout_height*).

Hay tres posibles valores posibles:

- **match_parent**: el componente ocupará todo el espacio que tenga disponible.
- **wrap_content**: el componente ocupará solo el espacio necesario para mostrarse.
- Especificar una medida exacta en **Pixels** (no suele ser recomendable) o en **DPs**.

Además a cada componente dentro de un **LinearLayout** se puede asignar un peso, mediante el atributo *Layout_weight*, que indica el espacio que ocupa en relación al resto de componentes del **LinearLayout**. Se suman todos los pesos y se reparten el espacio proporcionalmente al valor del peso. Si no se desea que utilice sus pesos para gestionar su tamaño hay que poner *Layout_weight* a 0 en todos los elementos del *layout*.

Para el peso funcione de forma correcta se ha de indicar que escale su tamaño de forma automática:


- **LinearLayout** horizontal: la propiedad *layout_width* se establece a **0dp**.
- **LinearLayout** vertical: la propiedad *layout_height* se establece a **0dp**.

Para ver las propiedades disponibles se puede consultar los enlaces:

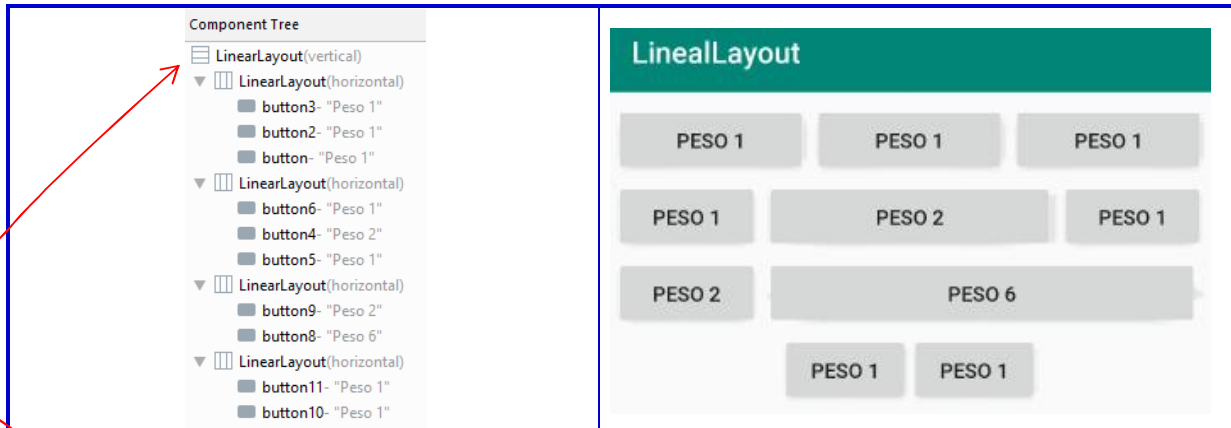
- <https://developer.android.com/reference/android/widget/LinearLayout.html>
- <https://developer.android.com/reference/android/widget/LinearLayout.LayoutParams.html>

¹¹ <https://developer.android.com/guide/topics/ui/declaring-layout>

¹² <https://developer.android.com/guide/topics/ui/layout/linear>

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Podemos ver el ejemplo de uso de *LinearLayout vertical* con cuatro *LinearLayout* horizontales dentro de él.



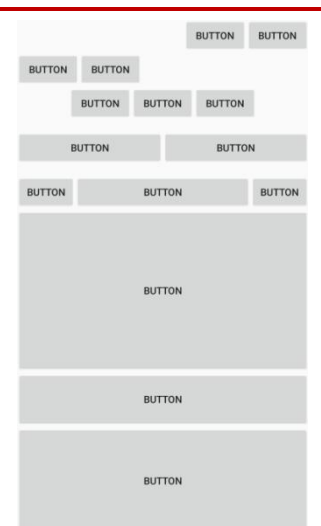
- Es un *LinearLayout* vertical que contiene cuatro filas que contienen cada una un *LinearLayout* horizontal.
- Los *LinearLayout* horizontal de nuestro ejemplo son cada una de las filas del *LinearLayout* vertical con:
 - La primera fila se compone de tres botones con idéntico peso por lo que el espacio disponible se lo reparten de forma equitativa (recordar poner el ancho a 0dp)
 - En la segunda fila están definidos tres botones: dos con peso 1 y uno con peso 2. Tenemos un espacio dividido en cuatro partes (suma de los pesos) por lo que el primer y el último botón tendrán como ancho una parte del total mientras que el botón del medio tendrá dos partes del total.
 - En la tercera fila se definen un botón con peso 2 y otro con peso 6. En este caso se divide el ancho en 8 partes. El primer botón tendrá dos partes del total y el último 6 partes del total, es decir, este último botón ocupará el triple de espacio que el primero.
 - En la última fila definimos dos botones con idéntico peso pero el *LinearLayout* horizontal se ha definido con las propiedades *layout_width* a *wrap_content* para que solo ocupe el espacio necesario y se centre de forma horizontal con la propiedad *gravity* a *center_horizontal*.


Ejercicio 2

Intenta imitar la siguiente distribución de elementos usando lineal layouts (da igual el color de los botones).

Establécela a partir de un *layout* vertical (cambiando las siguientes propiedades) y con *layouts* horizontales como filas

- *Divider*: establece el elemento que separa los elementos que tenga en su interior. Se puede establecer mediante un atributo Android (por ejemplo *dividerVertical*) o especificado una imagen (por ejemplo *line.png*).
- *Show dividers*: indica si se muestran o no los divisores. En nuestro caso llegaría con *middle* para separar los elementos que pongamos.



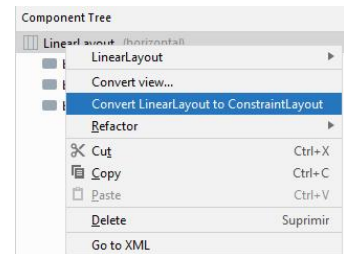
	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

5.2. ConstraintLayout^{13 14}

ConstraintLayout proporciona la creación de una interfaz compleja con un diseño de *layouts* planos, sin otros *layouts* anidados, basándose en el uso *constraints* entre los componentes y su *layout* padre.

Una de sus ventajas principales es que está diseñado y optimizado para ser usado desde el editor de *layouts* de *Android Studio* sin tener que editar, de forma directa, el fichero *XML* correspondiente.

Además, *Android Studio*, permite convertir otros *layouts* a *ConstraintLayout*. Basta con pulsar con el botón derecho sobre un *layout* en el *Component tree* de la pestaña *Design* y escoger la opción *Convert*.

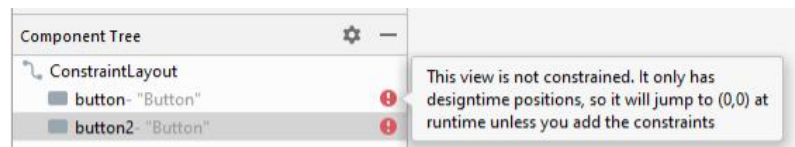


Cada componente que se inserte en un *ConstraintLayout* debe tener definidos, por lo menos, un *constraint* horizontal y otro vertical, que representan una restricción con otro componente presente en el *layout* o con el propio contenedor padre. Estos dos *constraints* posicionan el componente en el eje X y en el eje Y pudiendo necesitar más restricciones para un correcto posicionamiento.

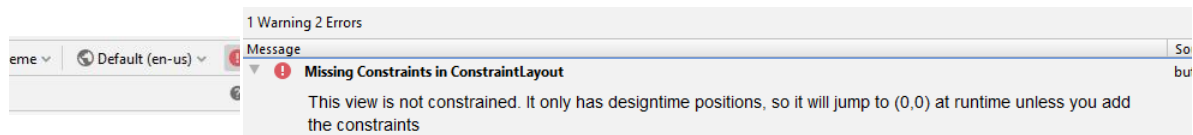
Tomando como referencia la coordenada 0:0, que es la zona superior izquierda del *layout*, si en un componente se omite alguna restricción los valores de sus coordenadas X y/o Y serán:

- No hay restricción en el eje X: el valor de la coordenada X será 0 (se sitúa a la izquierda).
- No hay restricción en el eje Y: el valor de la coordenada Y será 0 (se sitúa en la parte superior).
- No tiene restricciones: se sitúa en la coordenada 0:0 (zona superior izquierda)

El propio editor de *layouts* de *Android Studio* mostrará en el *Component Tree*, como errores, la omisión de alguna de las restricciones obligatorias (aunque la aplicación pueda ejecutarse sin problemas).

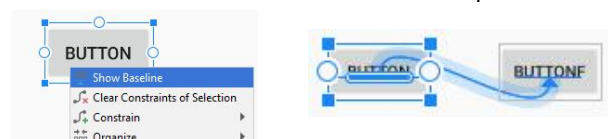
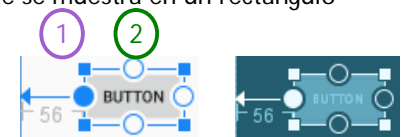


Pulsando en la exclamación superior se muestran los avisos y errores en una nueva ventana.




Cuando en un *layout* de tipo *ConstraintLayout* se selecciona un componente este se muestra en un rectángulo con los siguientes elementos:

1. **Vértices** → rectángulos que permiten redimensionar el componente (creando un tamaño fijo en el fichero *XML* que no suele ser recomendado).
2. **Aristas** → círculos huecos que representan nodos que permiten la creación de *Constraints*. Un círculo hueco indica que aún no se ha creado un *Constraints* en ese nodo mientras que un círculo relleno indica que ya está creado. Seleccionado un círculo con un *Constraints* creado, en el editor de *layouts*, y pulsado suprimir el *Constraint* se elimina. Sucede lo mismo pulsado este círculo en las propiedades.
3. Con la opción *show baseline*, que aparece pulsando con el botón derecho sobre un componente, permite crear *Constraints* con la línea base del texto, lo que normalmente los alinea verticalmente.



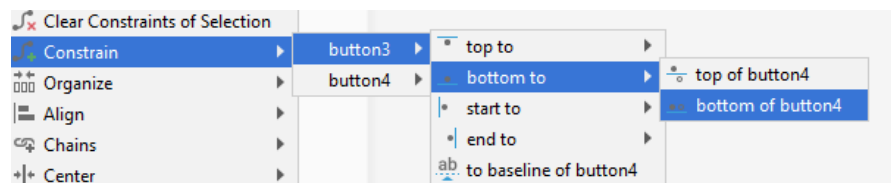
¹³ <https://developer.android.com/training/constraint-layout/>

¹⁴ <https://developer.android.com/studio/write/layout-editor?hl=es>

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Para crear un *constraint* se puede:

- Arrastrar un nodo hacia el nodo de otro componente, en el mismo plano, o hacia los bordes del *contenedor padre* y automáticamente se creara el *constraint* con unos márgenes por defecto.
- Pulsar con el botón derecho en un componente y seleccionar la opción: *Constraint*. Dependiendo de tenemos uno o más componentes seleccionados nos salen diversas opciones.

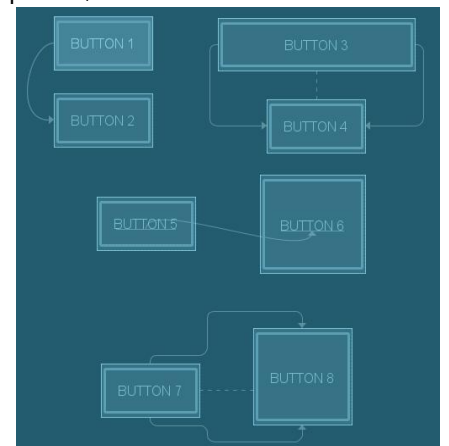


Se deben seguir las siguientes reglas:

- **Todo componente debe tener como mínimo dos *constraints*: uno horizontal y otro vertical.**
- Solo se pueden crear *constraint* entre nodos de la misma orientación. Los nodos situados en un plano horizontal solo se pueden enlazar con nodos horizontales y los verticales con los verticales. Las líneas base solo se pueden enlazar con líneas base.
- De cada nodo solo puede salir un enlace pero pueden llegar varios.

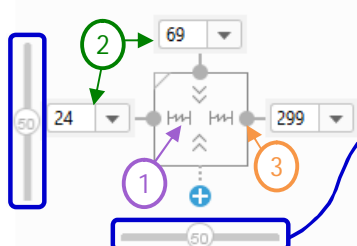
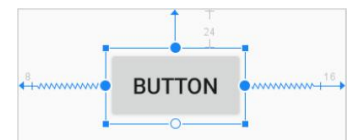
Los siguientes ejemplos sirven como ejemplo de su funcionamiento, teniendo en cuenta son ejemplos incompletos puesto que faltan, a cada componente, *Constraints*.

- En el primer grupo de botones el botón 1 está en la misma línea, por la izquierda, con el botón 2.
- En el segundo grupo de botones se ha enlazado ambos lados del botón 3 con sus respectivos lados del botón 4. Con esto se consigue que el botón 3 este centrado horizontalmente con el botón 4.
- Tanto en el tercer grupo como en el cuarto se han conseguido que los botones estén alineados verticalmente. En el tercer grupo se ha enlazado la línea base del botón 5 sobre la del botón 6.
- En el cuarto se ha alineado la parte superior e inferior del botón 8 con sus respectivos nodos en el botón 7. De este modo se consigue el alineamiento vertical.
- Estos alineamientos se pueden alterar añadiendo márgenes a los componentes. Por ejemplo, el botón 8 está desplazado del centro vertical del botón 7, debido a que se le ha añadido un margen superior, desplazándose hacia abajo.




Cuando se añade dos *Constraints* a un componente apuntado en direcciones opuestas se crean un enlace con una línea ondulada que indica que cada *constraints* "tira" hacia su lado. Esto provoca que el componente se centre en el espacio libre de márgenes (los márgenes aparecen como líneas lisas en los extremos del *constraint*). Mientras que si solo lo hay en una dirección la línea es lisa ya que es el único *constraint* que tira del componente.

En este ejemplo se puede ver que se han creado tres *constraints* sobre un botón. El *constraint* vertical se muestra como una línea lisa, porque no tiene un contrario, frente a los horizontales que se muestran con una línea ondulada. Además puede verse el margen de 8dps que posee a la izquierda y de 16dps a la derecha representado con una línea lisa en el extremo externo de la línea ondulada.






Cuando se añaden *constraints* en sentidos opuestos el componente se centra en el *layout*. La posición del componente se puede modificar mediante la *herramienta Bías*, siendo esta horizontal y vertical.

Desplazando las barras se puede mover el componente de manera horizontal y/o vertical conservando los *constraints* definidos sobre él.

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				


Además, desde la zona anterior, se pueden realizar las siguientes operaciones:


- Se puede cambiar la forma en que el **ancho y el alto** son calculados. Se tienen las siguientes posibilidades:
 -  Fixed su tamaño es fijo estando especificado en *DPs*.
 -  Wrap Content el componente usa el tamaño que necesita para representar su contenido.
 -  Match Constraints (es lo mismo que *Odp*) el componente usa el tamaño que necesita (excluyendo los márgenes) para cumplir los *constraints* definidos sobre él. Esta opción sustituye a *match_parent* en estos *layouts*. Nos permite también, mediante un triángulo en la esquina superior derecha, establecer un ratio en el tamaño del componente, por ejemplo 16:9. <https://constraintlayout.com/basics/dimensions.html>
- Se pueden establecer los **márgenes** para las direcciones con *constraints*.
- Pulsando en los círculos grises (azules cuando se posiciona el ratón en él) se pueden **eliminar el constraint** definido en esa dirección. Si tienen un símbolo + es que ese nodo no tiene un *constraints* definido y se puede crear pulsando en él.


En la barra de herramientas del editor de *layouts* además podemos encontrar las siguientes herramientas.





 Hide/Show Constraint permite mostrar u ocultar elementos del *layout* actual.

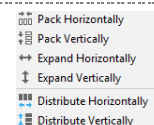
 Cuando autoconnect está habilitado y se añade un componente al *layout* se crean automáticamente dos o más *Constraint* sobre el componente y el *layout* padre (nunca sobre otros componentes).


 Default margins permite establecer el margen por defecto, cuando se crea un *constraint*, de cada componente nuevo que se añade al *layout*.

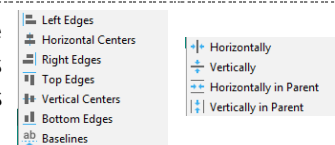
 Clear Constraints permite borrar todos los *constraint* de todos los componentes del *layout*.

 Infer Constraints permite crear de forma automática los *constraints* que faltan en el *layout* actual. Puede ser necesario realizar ajustes sobre estos *constraints* creados para garantizar una correcta visualización ante diferentes tamaños de pantalla y orientaciones.


 Pack permite distribuir los elementos seleccionados por el *layout*. Las opciones disponibles varían si se tiene un único componente seleccionado o varios.



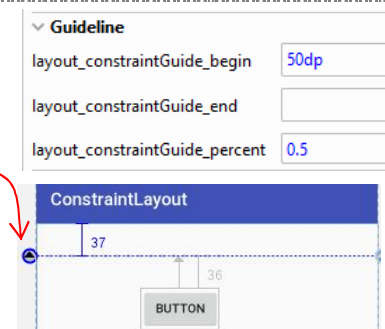
 Align permite alinear, según distintos criterios, los componentes que se han seleccionado (se pueden obtener las mismas opciones pulsando con el botón derecho del ratón sobre el *layout*). Las opciones disponibles varían si se tiene un único componente o varios.




El botón *Guidelines* permite añadir guías verticales y horizontales para facilitar la colocación de elementos en el *layout*. Se puede especificar su posición tanto en *DPs* (desde el principio o el final del *layout*) como con un porcentaje de la pantalla mediante un valor entre 0 y 1. Se puede cambiar el tipo con el siguiente icono.

 En el ejemplo siguiente tenemos una guía horizontal situada a 37dp de la parte superior del *layout* con un botón conectado a ella. Tiene la ventaja de permitir tanto alinear componentes, de manera cómoda, como desplazarlos, de manera simultánea, siguiendo el desplazamiento de la guía.

<https://constraintlayout.com/basics/guidelines.html>

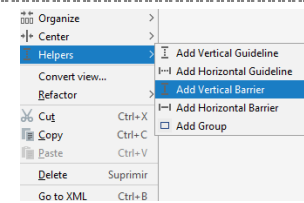


	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

I

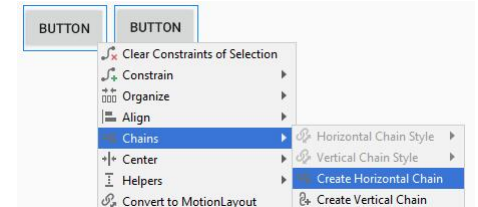
Las barreras permiten crear una barrera virtual en el componente mas extremo del lado que se le haya indicado.

En la siguiente página se encuentra una explicación detallada de las mismas: <https://constraintlayout.com/basics/barriers.html>

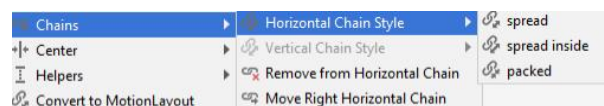


Cuando se selecciona más de un componente se puede crear cadenas (*chains*) sobre ellos. Estos son un tipo de *constrain* que permite especificar cómo se reparte el espacio entre ellos. Se crea con la siguiente opción del menú contextual: Chains -> Create Horizontal/Vertical Chain.

Cuando se crean las cadenas se visualiza una línea como forma de cadena entre ellos:



Una vez creado una cadena se puede cambiar su tipo en el menú anterior. Como en el ejemplo anterior creamos una cadena horizontal se habilita la opción para cambiar el tipo de cadena horizontal.

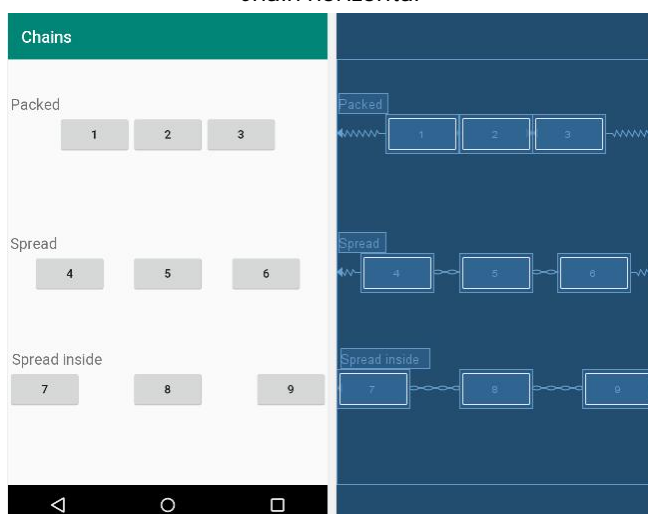


Si no apareciese esta opción este valor se puede cambiar con las propiedades, que están situadas dentro de la propiedad *Layout_Constraint*: *layout_constraintHorizontal_chainStyle* y *layout_constraintVertical_chainStyle*.

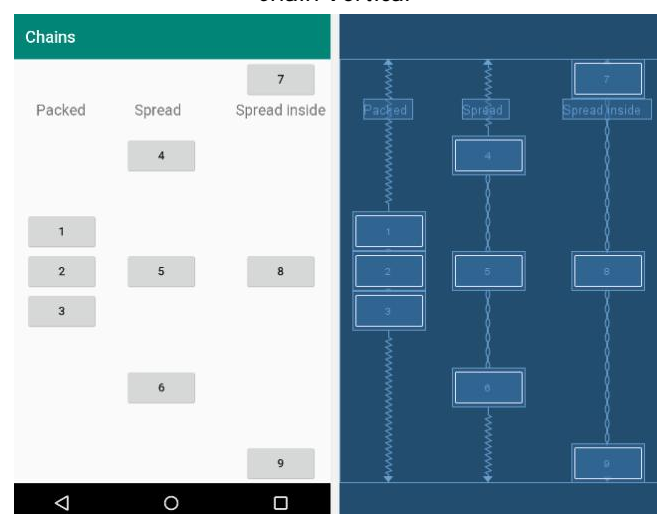
Las opciones disposiciones son:

- *Packed*: los componentes se centran. Mediante los márgenes se puede conseguir la separación deseada. Existe la posibilidad de desplazarlos del centro usando las propiedades *Horizontal_bias* y *Vertical_bias* situadas dentro de la propiedad *Layout_Constraints* con un valor entre 0 (izquierda) y 1 (derecha).
- *Spread*: los componentes se distribuyen uniformemente.
- *Spread inside*: los componentes de los lados se pegan a los extremos mientras que los del interior se reparten el espacio de forma uniforme.

Chain horizontal




Chain Vertical



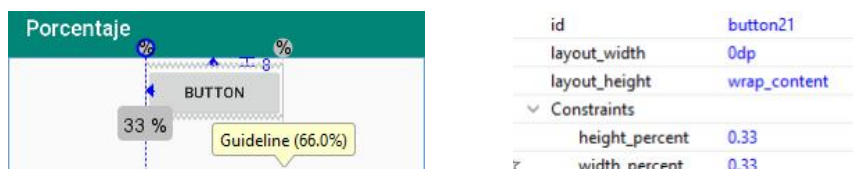
Jugando con los pesos se pueden distribuir los elementos de forma similar a un *linear layout*. Las propiedades que se usarán son *Horizontal_weight* y *Vertical_weight* situadas dentro de la propiedad *Constraints*.

https://constraintlayout.com/basics/create_chains.html

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

También se puede crear un diseño en el que se use un porcentaje del tamaño del componente padre como tamaño de un componente. Las propiedades que se usarán son *layout_constraintWidth_percent* y *layout_constraintHeight_percent* situadas dentro de la propiedad *layout_constraints*. Estas se pueden combinar con líneas guía definidas también con porcentajes.

En el ejemplo queremos que un botón ocupe el tercio central de la pantalla. Para ello se ha definido un botón con una anchura del 33% de la pantalla y se le ha hecho un *Constraints* con una línea guía situada al 33% de la pantalla. Se puede comprobar que el final del botón coincide con una línea guía situada al 66% de la pantalla.



Para ver más información sobre *ConstraintLayout* se puede consultar la siguiente guía:

- <https://developer.android.com/training/constraint-layout/index.html>

Para ver sus propiedades se puede consultar los enlaces:

- <https://developer.android.com/reference/androidx/constraintlayout/widget/ConstraintLayout>
- <https://developer.android.com/reference/android/support/constraint/ConstraintLayout.LayoutParams.html>

Ejercicio 3

Repita el ejercicio 3 pero usando un *ConstraintLayout* en vez de un *LinearLayout*.

En este caso no hacen falta los *dividers*.

6. Acceso a elementos de la interfaz de usuario

Normalmente los elementos de la interfaz de usuario presentes en un *activity* se definen en un fichero *XML* que define el *layout*.

Para acceder a esos elementos se usará la clase *R* junto con el valor de la propiedad *id* del elemento al que queramos acceder.

Por ejemplo:

```
TextView txtMensaje = (TextView) findViewById(R.id.miTextView);
```


En el objeto *txtMensaje* tenemos el componente de tipo *TextView* identificado con el *id*, indicado en la propiedad *id*, *miTextView* (*R.id.miTextView*).

Los demás componentes de la interfaz de usuario se pueden obtener de forma análoga:

```
Button btnAceptar= (Button) findViewById(R.id.botonAceptar);
EditText editNombre = (EditText) findViewById(R.id.editTextNombre);
```

A partir de la versión 3 de Android Studio ya no es necesario realizar el casting y se puede omitir el texto gris que se encuentra entre paréntesis. Por lo que las dos sentencias anteriores quedarían:

```
Button btnAceptar= findViewById(R.id.botonAceptar);
EditText editNombre = findViewById(R.id.editTextNombre);
```

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

7. Componentes estándar de la interfaz gráfica de Android

En la siguiente lista veremos los componentes, *Views*, más comunes de *Android* y sus principales propiedades teniendo en cuenta que cada tipo puede tener propiedades propias. Cada componente deberá tener especificado, en un principio, el atributo *id*. Este atributo es un entero único que nos permite identificar el componente.

Todos los elementos de la interfaz de usuario heredan de:

- *View*¹⁵: Es un objeto que se dibuja en pantalla y con el que usuario puede interactuar.
- *ViewGroup*¹⁶: es un conjunto de *Views* y *ViewGroups* que se combinan para crear una interfaz de usuario.

Esto implica que todos los objetos de nuestra interfaz no solo tendrán los métodos y propiedades del propio componente sino que además tendrán los heredados de *View* y/o los heredados de *ViewGroup*.

Todos los componentes de *Android* (así como los *Layouts*) deben definir, de forma obligatoria, las propiedades ancho (*layout_width*) y alto (*layout_height*). Hay tres posibles valores posibles:


- *match_parent* o *match_constraint*: el componente ocupará todo el espacio que tenga disponible su contenedor o todo el espacio que le deje disponible las restricciones que tenga definidas.
- *wrap_content*: el componente ocupará solo el espacio necesario para mostrarse.
- Especificar una medida exacta en DPs o Pixels (no suele ser recomendable).

Vamos a ver algunas de las propiedades comunes a todos los componentes teniendo en cuenta que los valores asignados a estas propiedades se pueden modificar tanto desde el fichero *XML* que representa *Layout*, desde el editor de propiedades del editor gráfico del *Layout* o mediante código:

- *text*: en de que el componente muestre un texto su valor se establece en este atributo.
- *textColor*: se utiliza para establecer el color del texto.
- *textSize*: se utiliza para establecer el tamaño del texto.
- *textStyle*: se utiliza para establecer el estilo del texto: negrita, cursiva, ...
- *textAlignment*: alineación del texto en el componente.
- *typeface*: fuente de texto utilizada.
- *hint*: sirve para establecer un texto de sugerencia. Este valor se borra al introducir un texto.
- *background*: sirve para establecer el color de fondo del componente.
- *gravity*: especifica cómo se sitúan los elementos internos de un componente dentro del propio componente.
- *padding*: sirve para definir el relleno dentro del componente.
- *margin*: sirve para definir la separación con otros elementos.
- *alpha*: transparencia del componente.
- *style*: se utiliza para especificar el estilo a usar en el componente. Son un ámbito muy amplio mediante los cuales se pueden cambiar totalmente el aspecto de los componentes mediante *XML*. Existen estilos predefinidos y otros que puede definir el programador. En general no se verán.
- *visibility*: permite forzar a que un componente se muestre o se oculte. Sus posibles valores son:
 - *View.VISIBLE*: el elemento es visible.
 - *View.INVISIBLE*: el elemento es invisible pero ocupa su espacio en el *layout* como si fuese visible. Este componente afecta al resto de componentes.
 - *View.GONE*: el elemento es invisible y no ocupa espacio en el *layout*. Este componente no afecta al resto de componentes.
- Todos los atributos tienen sus correspondientes getters y setters para obtener y modificar sus valores mediante código.

¹⁵ <https://developer.android.com/reference/android/view/View.html>

¹⁶ <https://developer.android.com/reference/android/view/ViewGroup.html>

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

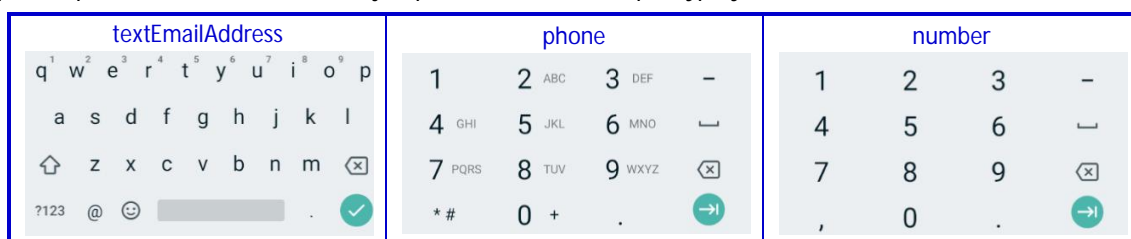
- Hay grupos de componentes que pueden tener las mismas propiedades. Por ejemplo *toggle button* y *check box* tienen la propiedad marcar/desmarcar teniendo métodos como *setChecked(boolean)* o *isChecked()* para marcar/desmarcar o comprobar si están marcados.
- Fijar los *listeners* para distintos eventos: *onClick*, *onLongClick*, *onFocusChange*, *onCheckedChange*, ... Algunos elementos pueden tener sus propios *listeners*. Es importante resaltar que la mayoría de los *listeners* se programan mediante código Java. En XML prácticamente se limita al método *onClick*. El hecho de que no exista el evento en XML no implica que no exista en el componente. Se verán en el punto siguiente del tema.
- Por defecto todos los que tengan elementos en color como pueden ser Switch, CheckBox, ToggleButton, FloatingActionButton, SeekBar este color es el color *colorSecondary* definido en el fichero *themes.xml*.

Veremos ahora los distintos componentes y algunas de sus propiedades, *listeners* o métodos java específicos teniendo en cuenta que aunque se definan en un componente se pueden utilizar en otros:

Ab TextView • TextView: Es un campo de texto, una etiqueta, no editable.

<https://material.io/design/components/text-fields.html>

Ab Plain Text • EditText: Es un campo de texto editable. Existe la posibilidad de establecer mediante el atributo *InputType* el tipo de entrada que acepta y con ello el teclado que se muestra. También se puede indicar que ocupe varias líneas. Estos son ejemplos de valores de *InputType* y de sus teclados asociados:











- Button** • Button: Permite ser pulsado y generar un evento en respuesta. Se puede usar, para detectar la pulsación del mismo, el método *setOnClickListener*. <https://material.io/design/components/buttons.html>
- *icon*: añade un icono a la derecha de texto.
 - *iconTint*: establece el color del icono anterior.
 - *onClick*: método que se ejecuta al pulsar el botón. No se puede usar a la vez que *setOnClickListener*.
 - *strokeColor*: color del borde.
 - *strokeWidth*: ancho del borde.
 - *cornerRadius*: permite crear botones redondeados indicándole el radio del borde en DPs.
 - *drawableStart*, *drawableEnd*, *drawableTop* y *drawableBottom*: permite colocar una imagen antes, después, encima y debajo respectivamente del texto del botón. No es compatible con el atributo *icon*.
 - *Rotation*: y sus variantes. Permiten rotar el componentes en cualquiera de los tres ejes: x, y, z.

ImageButton • ImageButton: Es un botón con una imagen la cual se especifica en su propiedad *src*. Su comportamiento es idéntico al de un botón.


- + FloatingActionButton** • FloatingActionButton (FAB): En un botón flotante con una imagen. En versiones anteriores de Android Studio se requiere de la biblioteca *design* que se añade al arrastrar el componente al layout. <https://material.io/design/components/buttons-floating-action-button.html>
- *srcCompat*: imagen que muestra el botón.
 - *backgroundTint*: color del botón.
 - *fabCustomSize*: permite cambiar su tamaño.


- ToggleButton** • ToggleButton: Es un botón de dos estados, pulsado o no, con un indicador visible. El cambio de colores se realizara mediante estilos. Se usa *setOnCheckedChangeListener* para capturar su cambio de estado.
- *checked*: indica si el botón se encuentra pulsado o no.
 - *textON*: texto que muestra el botón cuanto está pulsado.
 - *textOff*: texto que muestra el botón cuanto no está pulsado.


	RAMA:	Informática	CICLO:	Desenvolvemto de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

-  **Switch** • Switch: Es un botón de dos estados para seleccionar entre dos opciones. Se usa el método `setOnCheckedChangeListener` para capturar su cambio de estado.
 - o checked: indica si se encuentra pulsado o no.
 - o textON: texto que muestra el botón cuando está pulsado.
 - o textOff: texto que muestra el botón cuando no está pulsado.
 - o showText: muestra el texto especificado en textOn o en textOff o no.
 - o thumbTint: color del elemento gráfico del botón.
 - o trackTint: color del camino de fondo del botón.
-  **CheckBox** • CheckBox: Es un conmutador de selección con dos valores posibles: seleccionado y deseleccionado. Se debe de usar cuando se presenten opciones seleccionables no excluyentes: se puede marcar más de uno de forma simultánea. Se usa el método `setOnCheckedChangeListener` para capturar su cambio de estado. <https://material.io/design/components/selection-controls.html>
 - o checked: indica si se encuentra pulsado o no.
 - o button: permite cambiar la imagen que representa el CheckBox.
 - o buttonTint: permite cambiar el color del componente cuando está seleccionado.
 - o style: al igual que la propiedad `button` permite cambiar la imagen que representa el CheckBox.
-  **RadioButton** • RadioButton: Es un conmutador de selección con dos valores posibles: seleccionado y deseleccionado. Se debe de usar cuando se presenten opciones seleccionables excluyentes: solo se puede seleccionar uno de las RadioButton dentro del mismo RadioGroup. Se verán en un punto 9 del tema.
 - o checked: indica si se encuentra pulsado o no.
 - o button: permite cambiar la imagen que representa el RadioButton.
 - o buttonTint: permite cambiar el color del componente cuando está seleccionado.
 - o style: al igual que la propiedad `button` permite cambiar la imagen que representa el RadioButton.
-  **RadioGroup** • RadioGroup: Permite agrupar varios RadioButtons en su interior. Se verán en un punto 9 del tema. Se usa el método `setOnCheckedChangeListener` para capturar su cambio de estado.
 - o checkedButton: permite establecer si inicialmente hay un RadioButton seleccionado.
 - o Orientation: permite mostrar los RadioButtons del RadioGroup en orientación horizontal o vertical.
 - o Tiene el método `clearCheck` que nos permite desmarcar todos los RadioButtons del RadioGroup.
-  **RatingBar** • Rating bar: Permite establecer un valor de forma visual en un principio mediante estrellas. Para capturar el cambio de estado del `RatingBar` se usa el método `setOnRatingBarChangeListener`.
 - o numStars: numero de estrellas que muestra.
 - o rating: número de estrellas que se muestran marcadas.
 - o stepSize: especifica el tamaño mínimo de selección de estrellas: una estrella, media estrella, ...
 - o progressTint: color de las estrellas marcadas.
 - o inIndicador: solo muestra las estrellas no pudiendo cambiar su valor.
 - o style: permite elegir un estilo diferente de RatingBar.
-  **ImageView** • ImageView: Permite mostrar una imagen.
 - o src: indica la imagen que se muestra.
 - o scaleType: permite indicar como se coloca y escala la imagen dentro de `ImageView`.
-  **WebView** • WebView: permite mostrar una página web. Esta puede ser de Internet o indicada mediante una cadena de texto. Solo veremos la carga básica de una página. Para poder acceder a Internet se necesita el siguiente permiso que se ha de añadir al fichero `AndroidManifest.xml` fuera del elemento `application` (los permisos se verán en el tema 5): `<uses-permission android:name="android.permission.INTERNET" />`
 La página a cargar se debe especificar mediante código.



```
WebView webView=findViewById(R.id.miwebView); // Se obtiene el objeto WebView
webView.loadUrl("http://example.com"); // Se especifica la URL de la página a cargar.
webView.loadData(miHtml, "text/html", "UTF-8"); // Se carga la pagina presente en la cadena miHtml
```

	RAMA:	Informática	CICLO:	Desenvolvemeto de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				


 **Chip** • Chip y ChipGroup: Chip es un elemento compacto que representa una entrada, un atributo o una acción. Mientras que ChipGroup es una agrupación de chips. Se verán más adelante.


 **SeekBar** • SeekBar: Permite desplazar un selector para introducir valores. Para capturar el cambio de estado del SeekBar se usa el método `setOnSeekBarChangeListener`. <https://material.io/design/components/sliders.html>

- o progress: el valor del progreso que muestra.
- o max: el valor máximo del progreso que puede mostrar.
- o progressTint: color de la línea de selección.
- o progressBackgroundTint: color de la parte no seleccionada de la línea.
- o thumb: cambia la imagen que muestra la posición seleccionada.
- o progressDrawable: imagen que se muestra en vez de la línea de selección. Puede ser un color.
- o style: permite elegir un estilo diferente de RatingBar.

 **ProgressBar** • ProgressBar: es un elemento visual que indica el progreso de una operación. Tiene las mismas propiedades que SeekBar pero si un selector de valor.

- o style: permite elegir entre un ProgressBar horizontal o circular.
- o Indeterminate: la animación se produce de forma continua. Se puede utilizar para indicar al usuario que se está realizando una tarea en segundo plano que no se está mostrando por pantalla.

 **ScrollView** • ScrollView: Es un contenedor que permite realizar un desplazamiento vertical del *layout* si los componentes insertados en él ocupan más espacio que el tamaño del contenedor.

 **HorizontalScrollView** • HorizontalScrollView: Es un contenedor que permite realizar un desplazamiento horizontal del *layout* si los componentes insertados en él ocupan más espacio que el tamaño del contenedor.

 **NestedScrollView** • NestedScrollView: Permite crear un contenedor con *scroll* dentro de otro contenedor con *scroll*. Es compatible con *CoordinatorLayout*: <http://www.sgoliver.net/blog/animaciones-basicas-coordinatorlayout/>.

Estos componentes, a partir de la versión 8 de *Android*, disponen de la propiedad *tooltipText* que muestra un *tooltip* ante una pulsación larga del componente.

```
Button but=findViewById(R.id.button2);
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
    but.setTooltipText("Resalizar una llamada");
}
```

8. Eventos


Android es capaz de capturar y manejar diversos eventos sobre los componentes. Estos eventos son capturados y pasados a la clase encargada de recogerlos.

A continuación gestionaremos el evento de pulsar un botón. Un escuchador de eventos es una interfaz de la clase *View*¹⁷, y de sus descendientes como *Button* y casi todos los componentes, del cual tendremos que implementar un método *callback* para realizar la acción asociada al evento. Cada escuchador tiene un único método *callback* asociado.

Los más habituales son:

Evento	Método	Objeto anónimo	Método Callback
Se pulsa un componente.	<code>setOnClickListener</code>	<code>View.OnClickListener</code>	<code>onClick()</code>
Se pulsa de forma prolongada un componente.	<code>setOnLongClickListener</code>	<code>View.OnLongClickListener</code>	<code>onLongClick()</code>
Se añade texto en un editText	<code>addTextChangedListener</code>	<code>TextWatcher</code>	<code>onTextChanged()</code>
Se consigue o se pierde el foco de un componente.	<code>setOnFocusChangeListener</code>	<code>View.OnFocusChangeListener</code>	<code>onKey()</code>

¹⁷ <https://developer.android.com/reference/android/view/View.html> para ver todas la interfaces disponibles.

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Cuerpo de los escuchadores indicados en la tabla superior:

```

boton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // acciones a realizar al realizar una pulsación normal
    }
});

editText.addTextChangedListener(new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence cs, int i, int i1, int i2) {
        // acciones a realizar antes de añadir el texto
    }
    @Override
    public void onTextChanged(CharSequence cs, int i, int i1, int i2) {
        // acciones a realizar al añadir el texto
    }
    @Override
    public void afterTextChanged(Editable editable) {
        // acciones a realizar después de añadir el texto
    }
});

editText.setOnFocusChangeListener(new View.OnFocusChangeListener() {
    @Override
    public void onFocusChange(View v, boolean hasFocus) {
        // acciones a realizar al cambiar el foco
    }
});

boton.setOnLongClickListener(new View.OnLongClickListener() {
    @Override
    public boolean onLongClick(View v) {
        // acciones a realizar en una pulsación larga.
        return true;
    }
});

```

Si se devuelve true indica que se ha gestionado la pulsación. En cambio si se devuelve false se indica que no se ha gestionado por lo que se ejecutará además el siguiente *listener*, de haberlo, en este caso el de una pulsación normal.

Según los ejemplos anteriores para añadir un nuevo *listener* al botón para que recoja una pulsación sobre él usaríamos el siguiente código donde el código que queremos que se ejecute se implementará en el método `onClick`:

```

// Gestionamos el evento mediante un método callback
btnAceptar.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        textView.setText("Hola, bienvenido a la programación en Android");
    }
});

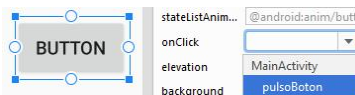
```

Por supuesto se podría hacer también usando alguna otra estructura vista el curso pasado. Por ejemplo que la clase *MainActivity* implemente la interfaz *View.OnClickListener* y luego sobrescribiendo el método `onClick` dentro de la misma.

Otra posibilidad es la utilización del atributo `onClick` del componente, en dicho atributo se ha de especificar el nombre del método que se va a ejecutar al pulsar el botón: debe ser público, que no devuelve nada y tener como único parámetro un *view*, el componente sobre el que se pulsa.

Ejemplo

En las siguientes imágenes se puede observar cómo se especifica, en la propiedad `onClick`, en un botón el método `pulsoBoton` que se implementa en *MainActivity.java*.



```

public void pulsoBoton(View view){
    Log.i("MainActivity", "Pulso botón");
}

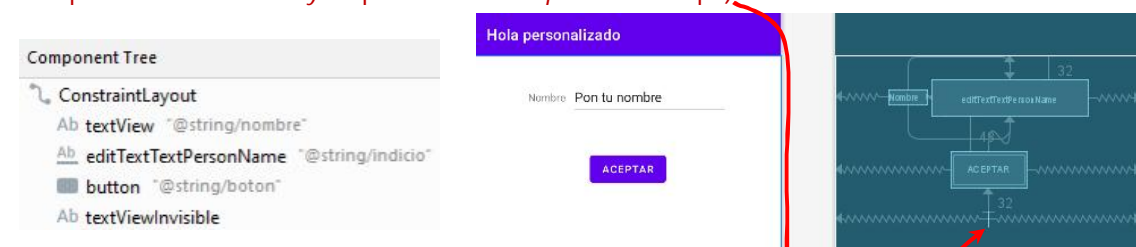
```


Ejercicio 4

Lo primero es crear un proyecto que llamaremos *HolaPersonalizado*.

La *activity* principal tendrá dos *textView*, un *editText* y un botón que al ser pulsado mostrará el contenido del *editText* en el segundo *textView* situado debajo del botón. Además al pulsar el botón, si la cadena contenida en el *editText* contiene una *a* esta se deberá mostrar mediante un *Toast*.

Juega con el *editor de Layout* para conseguir realizar una interfaz parecida (no es importante que sea exacta ni que coincidan los colores) a la de la siguiente imagen (observa que el segundo *textView* no se ve en la previsualización del *layout* pero si en el *blueprint* o cianotipo):



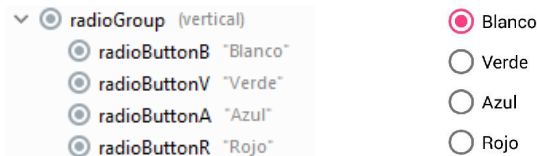
	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

9. Radiogroup/Radiobutton¹⁸

Son elementos de selección mutuamente excluyentes. Los *radiobutton*, también pueden ser *checkboxes*, se agrupan mediante un *Radiogroup* (dentro del fichero XML los *radiobuttons* se definen dentro del *radiogroup*).

Ejemplo

Tenemos un *RadioGroup* con 4 *RadioButtons* que nos permiten seleccionar un color. Crea un *activity* con un *RadioGroup* similar al de las siguientes imágenes e inserta el siguiente código java.



```
RadioGroup radioGroup = (RadioGroup) findViewById(R.id.radioGroup);
radioGroup.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(RadioGroup group, int checkedId) {
        RadioButton rbBlanco = (RadioButton) findViewById(R.id.radioButtonB);
        RadioButton rbVerde = (RadioButton) findViewById(R.id.radioButtonV);
        RadioButton rbAzul = (RadioButton) findViewById(R.id.radioButtonA);
        RadioButton rbRojo = (RadioButton) findViewById(R.id.radioButtonR);
        if (rbBlanco.isChecked()) getWindow().getDecorView().setBackgroundColor(Color.WHITE);
        else if (rbVerde.isChecked()) getWindow().getDecorView().setBackgroundColor(Color.GREEN);
        else if (rbAzul.isChecked()) getWindow().getDecorView().setBackgroundColor(Color.BLUE);
        else if (rbRojo.isChecked()) getWindow().getDecorView().setBackgroundColor(Color.RED);
    }
});
```

El método *onCheckedChanged* anterior es equivalente al siguiente. En esta versión el *id* del elemento pulsado se obtiene del segundo parámetro del método: *checkedId* y no mediante el método *findViewById*.

```
public void onCheckedChanged(RadioGroup radioGroup, int checkedId) {
    int color=Color.RED;
    if (checkedId== R.id.radioButtonB) color=Color.WHITE;
    else if (checkedId == R.id.radioButtonV) color=Color.GREEN;
    else if (checkedId == R.id.radioButtonA) color=Color.BLUE;
    getWindow().getDecorView().setBackgroundColor(color);
}
```

Otra opción sería utilizar la propiedad *onClick* de cada *RadioButtom*. Para ello creamos un método, puede ser uno distinto por cada *radioButtom*, donde colocaremos el código que se ejecutará cuando se pulse en un *radioButtom*. En este caso en la propiedad *onClick* situariamos el método: *onRadioButtonClicked*:

```
public void onRadioButtonClicked(View view) {
    boolean checked = ((RadioButton) view).isChecked(); // Se comprueba si el botón esta pulsado
    switch(view.getId()) { // Se escoge el botón pulsado
        case R.id.radioButtonB:
            if (checked)
                // código
            break;
        case R.id.radioButtonV:
            if (checked)
                // código
            break;
    }
}
```

¹⁸ <https://developer.android.com/guide/topics/ui/controls/radiobutton#java>

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Además también se puede usar la propiedad *tag* para especificar el color correspondiente. Por ejemplo, el XML de unos de los *radioButtons* verde podría quedar así::

```
<RadioButton
    android:id="@+id/radioButtonV"
    android:text="Verde"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:tag="#ff00ff00"/>
```

De esta forma se puede establecer el siguiente código en *onCheckedChange* mucho más corto y cómodo:

```
RadioButton rb = (RadioButton) findViewById(id);
getWindow().getDecorView().setBackgroundColor(Color.parseColor(rb.getTag().toString()));
```

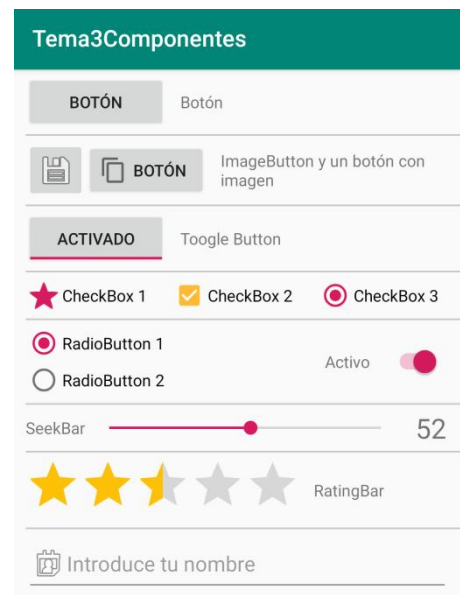
Con este último método hay que tener cuidado porque en un *RadioGroup* se pueden meter elementos que no son *RadioButtons*, por ejemplo un *CheckBox*, con lo que se puede producir un error de *casting*.


Ejercicio 5

Crea una nueva aplicación denominada Componentes dónde probaremos el comportamiento de alguno de los componentes que hemos visto.

Inserta los componentes presentes en la imagen lateral intentando imitar su disposición sin importar los colores de los botones ni de la ActionBar (barra del título).

- Prueba, en uno de los botones, a cambiar la propiedad *layout_width* entre los valores *match_constraint* y *wrap_content*. ¿Entiendes la diferencia?
- Haz que al activar el *toggle button* uno de los *checkboxes* se desactive y al desactivar el *toggle button* el *checkbox* se vuelva a activar
- Al desplazar la *seekBar* se muestre su valor en el campo de texto adyacente.
- Al activar el botón *switch* se muestre, en una etiqueta, el texto *activo* y al desactivarlo de *desactivo*.
- Al pulsar el botón superior el valor se reinicien los valores de los elementos: el *toggle button*, los *checkboxes*, los *radio button* y el *switch* pasen a estar no marcados. El *seekbar* pase a tener un valor de 0. El *rating bar* pase a tener 0 estrellas seleccionadas y se borre el texto que pueda tener el *editText*.
- Al pulsar el botón con imagen el texto adyacente se sustituye por un contador cuyo valor se incrementa cada vez que se pulsa el botón.
- Si el segundo *checkbox* está marcado al pulsar el botón el valor del contador decrece.
- Al pulsar un *RadioButton* se muestre un *Toast* informando del *RadioButton* pulsado

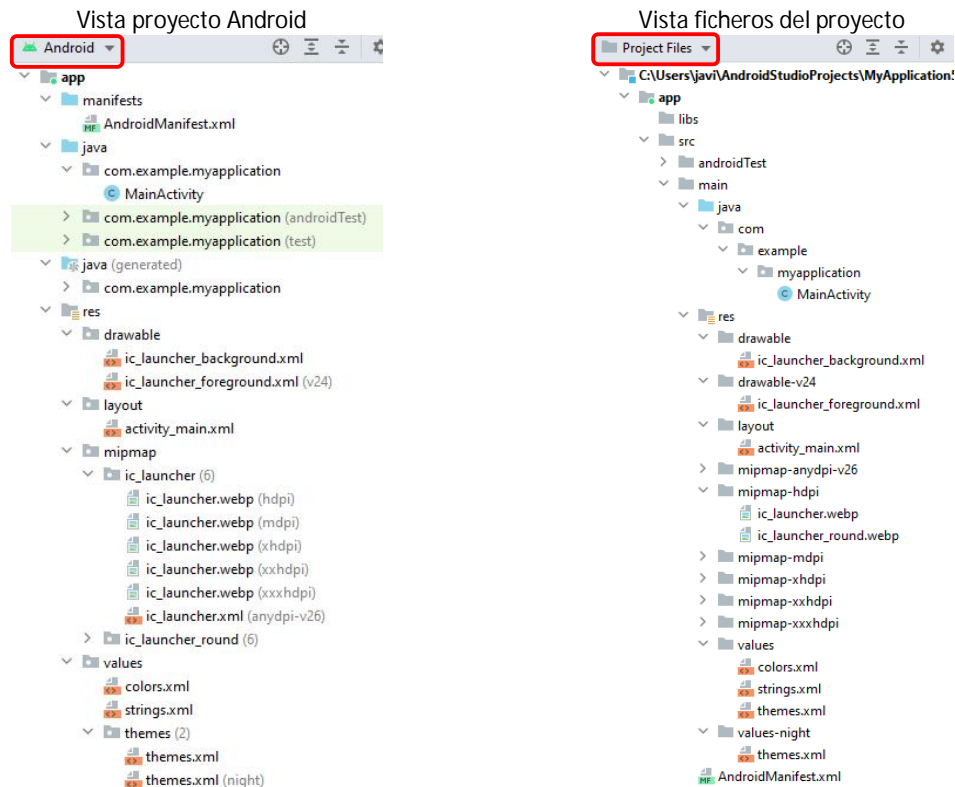


	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

10. Estructura de una aplicación.

La estructura de un proyecto Android es una representación de la estructura real del proyecto que se almacena en disco (se puede intercambiar entre las distintas vistas pulsado sobre el desplegable marcado en rojo).

Los tres directorios principales que se puede observar son el directorio *manifests*, el directorio *java* y el directorio *res*.



10.1. Archivo AndroidManifest.xml¹⁹

En este archivo, que se encuentra en la carpeta *manifests*, se introduce información sobre las características de una aplicación: versión *Android* usada, nombre de la aplicación, icono, *activities* y *activity* principal, gestión de permisos²⁰ para uso de elementos como el almacenamiento en el dispositivo, elementos *hardware*, geolocalización, acceso a la red, ...

Un punto importante del manifiesto es el *package*. A la hora de subir una aplicación al *Play Store* de Google esta puede llamarse igual a otra que ya exista pero el *package* debe ser único.


Iremos definiendo distintos elementos a medida que los vayamos usando.

10.2. Carpeta java

Es en esta carpeta, dentro del paquete definido en la creación del proyecto, donde se codificarán las clases java que utilizará nuestra aplicación. Dentro de este paquete se puede y se debe crear subpaquetes para una correcta organización del código.

¹⁹ <https://developer.android.com/guide/topics/manifest/manifest-intro.html>

²⁰ <https://developer.android.com/reference/android/Manifest.permission.html>

	RAMA:	Informática	CICLO:	Desenvolvememto de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

10.3. Recursos²¹

Los recursos son aquellos elementos que necesita nuestra aplicación pero que no es código propiamente dicho: textos, ficheros *xml* (con *layouts*, menús, propiedades, ...), imágenes, ficheros de sonido, fuentes,

Para este tipo de elementos el proyecto Android dispone de la carpeta *res* que a su vez tiene contiene subcarpetas diferentes según el tipo de recurso (dentro de estas subcarpetas no se pueden crear otras).

Actividad

Consultar la información disponible sobre recursos en las siguientes direcciones:

<https://developer.android.com/guide/topics/resources/available-resources.html>

<https://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources>

<https://developer.android.com/guide/topics/resources/more-resources.html?hl=es>

Para acceder a estos recursos, dependiendo del lugar desde el que se acceda, deberemos referenciarlos con una de las dos sintaxis siguientes:

- Desde un fichero java: `R.tipoDeRecurso.nombreDelRecurso`
- Desde un fichero XML: `@tipoDeRecurso/nombreDelRecurso`

Veremos cómo acceder a los distintos recursos en los puntos siguientes.

10.3.1. Strings

El fichero *strings.xml*, como todos los ficheros de valores que veremos, está situado en la carpeta */res/values* y nos permite almacenar cadenas de caracteres que se utilizan en la aplicación: etiquetas de los botones, los elementos menú, ... Por defecto siempre contiene el nombre de la aplicación: *app_name*.

Su nombre es arbitrario pudiéndose cambiar si se desea.

Un ejemplo de fichero *strings.xml*:

```
<resources>
  <string name="app_name">Mi primera aplicación</string>
  <string name="boton">Aceptar</string>
</resources>
```

Un cadena de texto se define con el elemento *String* donde su id es el valor indicado por el atributo *name* mientras que su valor el dato es valor del *element* (es el texto que va entre los *tags string*).

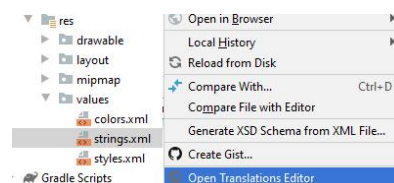
Usando estos ficheros *XML* Android permite realizar programas multiidoma²², creando un archivo *strings.xml* diferente por cada idioma.

Para aplicar esta localización se suele utilizar el editor de traducciones que proporciona *Android Studio* aunque también se puede realizar modificando directamente los ficheros *XML* con las traducciones. Para acceder se debe pulsar con el botón derecho sobre el fichero *strings.xml* seleccionando la opción *Open Translations Editor*.

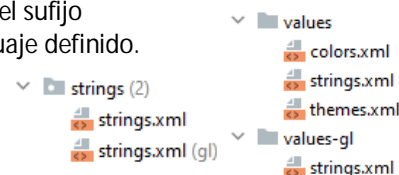
En la pestaña del editor de traducciones se puede:

- Botón +: añadir nuevas cadenas de texto para traducir.
- Botón con un globo terráqueo: añadir nuevos idiomas.
- Untranslatable: la cadena no se traduce siendo idéntica para todos los lenguajes.
- Si la columna key aparece en rojo es que hay un error, normalmente indica que falta alguna traducción.

Al añadir un nuevo idioma se crea automáticamente el fichero *XML* correspondiente. Internamente se crean carpetas *values* diferentes para cada idioma añadiéndole, al nombre de la carpeta, el sufijo del idioma. Es en estas carpetas donde están los recursos de tipo cadena para lenguaje definido.




Key	Resource Folder	Untranslatable	Default Value	Galician (gl)
app_name	app/src/main/res	<input checked="" type="checkbox"/>	My Application	
hello_world	app/src/main/res	<input type="checkbox"/>	¡Hola mundo!	Ola Mundo!



²¹ <https://developer.android.com/guide/topics/resources/accessing-resources.html>

²² <https://developer.android.com/guide/topics/resources/multilingual-support.html?hl=es>

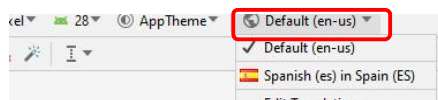
	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Para probar la internalización deberemos modificar el fichero *activity_main.xml* para que haga uso de las nuevas cadenas definidas en los ficheros *strings.xml*. Se puede realizar editando el fichero XML o en Java.

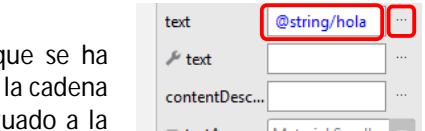
- Editor de *layouts* o fichero *XML*: esta modificación se puede realizar tanto mediante la ventana de diseño de *activities* como mediante el fichero *XML* asociado.

Sustituiremos el texto definido en el *textview* por la cadena *hola* que se ha definido y traducido en el fichero *strings.xml*. Para cambiar el valor de la cadena se puede escribir el valor a mano o usar el icono con tres puntos situado a la derecha de la propiedad *text* que mostrará todas las cadenas definidas en el fichero *strings.xml*.

Como se puede ver, a la derecha, el texto mostrado en el *textview*, atributo *text*, ha cambiado su valor a la cadena de texto con clave *hola* definido en el fichero *Strings.xml*:



Para comprobar su funcionamiento podemos cambiar el idioma en el dispositivo o mediante la opción del editor de layouts marcada a la izquierda.



```
<TextView
    android:id="@+id/textView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentStart="true"
    android:text="@string/hola"/>
```

- Mediante código java: se obtiene el objeto *textView* con el id adecuado y se establece su valor mediante código:

```
TextView textView=(TextView)findViewById(R.id.textView2);
textView.setText(getString(R.string.hola));
```

Para obtener una cadena definida en el fichero *strings.xml* se utiliza el método *getString* junto con el id del recurso. Recordar que el formato a usar en el *id* para acceder a los recursos desde código java es:

R.tipoDeRecurso.nombreDelRecurso pero que en ficheros XML es @tipoDeRecurso/nombreDelRecurso.

Android tratará de cargar los recursos que más se adaptan a nuestro dispositivo y si no hay ninguno, carga los recursos por defecto. Por eso en el ejemplo anterior se toma por defecto la carpeta *values* en inglés pero cuando el móvil está configurado en español la que se carga la carpeta *values-es*.

Además de las cadenas que definamos nosotros Android proporciona una serie de recursos comunes a la plataforma Android. Se diferencian de los recursos definidos por nosotros en que se le antepone la palabra *android* en su definición: *getString(android.R.string.cancel)*;

Actividad

Añadir los idiomas español y gallego y añade, traduciéndolas, algunas cadenas de texto. Añade al layout algún botón y asígnale estas cadenas. Prueba a cambiar de idioma en el editor de *layouts*. ¿Qué sucede?

10.3.2. Dimensiones

Los valores de las dimensiones nos permiten establecer tamaños de componentes, márgenes, *padding*s, ... se establecen en un fichero *XML* dentro del directorio *values*, por ejemplo *dimens.xml*, usando para ello elemento *dimen*. Cuando se va a utilizar el mismo valor en diferentes localizaciones es recomendable definirlo así para garantizar la consistencia y reusabilidad del mismo.

El siguiente ejemplo muestra la definición de un nuevo alto para un *textview* donde *name* representa el id de la dimensión.

```
<dimen name="alto_textview">150dp</dimen>
```


Cuando queremos establecer las dimensiones de un componente se puede realizar, como en el caso de las cadenas de texto, mediante el editor de *layouts* (y por tanto en el fichero *XML*) o mediante código java.

- Editor de *layouts* o fichero *XML*: se edita la propiedad *height* y se le asigna el valor @dimen/alto_textview con lo que en el fichero XML queda:

```
android:height="@dimen/alto_textview".
```

- Código java:

```
textView.setHeight((int) getResources().getDimension(R.dimen.alto_textview));
```

	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

10.3.3. Colores²³

El fichero *colors.xml* permite definir colores, usando el elemento *color*, que pueden ser utilizados en los distintos componentes que formen la aplicación.

```
<color name="mi_color">#ff0000</color>
```

Como en los casos anteriores se puede obtener y utilizar su valor tanto con el editor de *layouts* (y en su correspondiente fichero *XML*) como desde el código java.

- Editor de *layouts* o fichero *XML*: se edita la propiedad *background* y se le asigna el valor *@color/color_textview* con lo que en el fichero *XML* queda:

```
android:background="@color/mi_color"
```

- Código java: como en varias versiones de Android el método *getColor* esta deprecado usaremos una de las siguientes opciones:
 - Usar métodos distintos según la versión de Android usada.

```
int color;
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) { // Api nivel 23 o superior
    color = getColor(R.color.mi_color); // Si la API usada va ser siempre superior al nivel 23 podemos usar esta
} else {
    color = getApplicationContext().getResources().getColor(R.color.mi_color);
}
```

- Usar la librería de retrocompatibilidad *Support V4 Library* la cual da soporte a versiones antiguas de la API de Android.

```
int color = ContextCompat.getColor(getApplicationContext(), R.color.mi_color);
```

Una vez obtenido el color, establecemos el fondo del *textView*:

```
textView.setBackgroundColor(color);
```

Por defecto *colors.xml* define una serie de colores que son usados en el fichero *Themes.xml* que es el que asigna a los distintos elementos de la aplicación:

10.3.4. Otros recursos

- *bool*: es un valor booleano definido en formato *XML*.

<pre><resources> <bool name="activado">true</bool> <bool name="borrar">false</bool> </resources></pre>	<pre>boolean activ = getResources().getBoolean(R.bool.activado);</pre>
--	--


- *integer*: Es un entero definido en *XML*.

<pre><resources> <integer name="max_valor">25</integer> </resources></pre>	<pre>int max = getResources().getInteger(R.integer.max_valor);</pre>
--	--

- *integer-array*: Es un array de enteros definido en *XML*.

<pre><resources> <integer-array name="pares"> <item>4</item> <item>8</item> <item>16</item> <item>32</item> </integer-array> </resources></pre>	<pre>Int[] pares = getResources().getIntArray(R.array.pares);</pre>
---	---

²³ <https://material.io/design/color/the-color-system.html#color-usage-and-palettes>

	RAMA:	Informática	CICLO:	Desenvolvemto de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

- String-array: Es un array de cadenas definidas en XML.

<pre><resources> <string-array name="provincias"> <item>La Coruña</item> <item>Lugo</item> <item>Orense</item> <item>Pontevedra</item> </string-array> </resources></pre>	<pre>String[] pro=getResources().getStringArray(R.array.provincias);</pre>
---	--

- array: Es un array de otros recursos definidos en XML.

<pre><resources> <array name="icons"> <item>@drawable/home</item> <item>@drawable/settings</item> <item>@drawable/logout</item> </array> <array name="colors"> <item>#FFFF0000</item> <item>#FF00FF00</item> <item>#FF0000FF</item> </array> </resources></pre>	<pre>Resources res = getResources(); TypedArray icons = res.obtainTypedArray(R.array.icons); Drawable drawable = icons.getDrawable(0); TypedArray colors = res.obtainTypedArray(R.array.colors); int color = colors.getColor(0,0);</pre>
---	--

10.3.5. Temas claro y tema oscuro²⁴

Android utiliza Material como un sistema adaptable de directrices, componentes y herramientas para el diseño de interfaces de usuario.

Android proporciona, usando temas, la capacidad de personalizar todos los elementos de la interfaz. Usa dos colores temáticos para expresar diferentes partes de la interfaz: un color primario y un color secundario.

Por defecto existen dos temas: un tema claro y un tema oscuro. El núcleo de cualquier tema oscuro es una paleta de colores que utiliza colores de fondo oscuros y colores de primer plano claros. Los temas de Material Dark hacen uso del Sistema de Color de Material, con el fin de proporcionar los valores predeterminados del tema oscuro para los colores de la paleta neutra, como colorBackground y colorSurface.

Según la especificación de Material Dark Theme, las superficies grandes no deberían utilizar colores de fondo brillantes porque pueden emitir demasiado brillo. Cuando sea aplicable, los temas de Material proporcionarán este comportamiento de forma predeterminada, por ejemplo, haciendo que en el tema Light se utilicen, por defecto, los colores primarios y en el tema oscuro los Surface.


Sin embargo, puede haber algunos casos en los que se necesite aplicar un cambio de color primario o Surface en alguna UI personalizada. Por comodidad, los temas de Material proporcionan un atributo colorPrimarySurface, que apunta a colorPrimary en el tema Light y a colorSurface en el tema Dark. También hay un atributo colorOnPrimarySurface correspondiente que se puede utilizar para los elementos de primer plano como el texto y la iconografía en la parte superior de un fondo colorPrimarySurface.

Por ejemplo creando la entrada `<item name="colorSurface">@color/purple_200</item>` se modificará el color de fondo de la ActionBar (barra de título) en el modo oscuro pero no en el claro.

Algunos de los beneficios de un tema oscuro incluyen la conservación de la batería para los dispositivos con pantallas OLED, la reducción de la fatiga visual y hacer más fácil el uso en entornos con poca luz.

Los ficheros con los temas están compuestos de atributos con un valor de un color asociado. Para modificar los colores de un tema simplemente hay que modificar los valores de esos atributos.

²⁴ <https://material.io/develop/android/theming/theming-overview>

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Los atributos presentes por defecto son:

Nombre del atributo	Descripción
colorPrimary	El color que se muestra con más frecuencia en las pantallas y componentes de su aplicación. Este color debe pasar las directrices de accesibilidad para el texto/iconografía cuando se dibuja sobre la superficie o el color de fondo.
colorPrimaryVariant	Una variación tonal del color primario.
colorOnPrimary	Un color que pasa las directrices de accesibilidad para el texto/iconografía cuando se dibuja sobre el color primario.
colorSecondary	El color secundario de la aplicación, normalmente un complemento acentuado del color primario.
colorSecondaryVariant	Una variación tonal del color secundario.
colorOnSecondary	Un color que pasa las directrices de accesibilidad para el texto/iconografía cuando se dibuja sobre el color secundario.

Cambiando estos seis atributos de color, puede cambiar fácilmente el estilo de todos los componentes a los que se aplica su tema.

Existen además una serie de colores adicionales, además de los vistos anteriormente, que aseguran combinaciones de colores accesibles. Estos atributos de color adicionales son los siguientes:

Nombre del atributo	Descripción
android:colorBackground	El color de fondo aparece detrás del contenido desplazable.
colorOnBackground	Un color que pasa las directrices de accesibilidad para el texto/iconografía cuando se dibuja sobre el color de fondo.
colorSurface	Los colores de la superficie afectan a las superficies de los componentes, como cards, sheets y los menús.
colorOnSurface	Un color que pasa las directrices de accesibilidad para el texto/iconografía cuando se dibuja sobre el color de la superficie.
colorError	Color que indica los estados de error, para componentes como los campos de texto
colorOnError	Un color que pasa las directrices de accesibilidad para el texto/iconografía cuando se dibuja sobre el color de error.

Existen también otros atributos, que no pertenecen a Material, y que permiten modificar otros elementos de la interfaz. Por ejemplo: `android:navigationBarColor` permite modificar el color de la barra de navegación. Un ejemplo de su utilización sería:

```
<item name="android:navigationBarColor" >@color/purple_700</item>
```


Mediante los ficheros de temas también se pueden personalizar el aspecto los componentes de la interfaz. Podremos crear un estilo para aplicárselo a una serie de componentes, sin tener que modificarlos uno a uno.

Por ejemplo cambiando el color de botón y el tamaño de la fuente:

```
<style name="Button" parent="Widget.MaterialComponents.Button">
  <item name="backgroundTint">@color/teal_200</item>
  <item name="android:textSize">24dp</item>
</style>
```

El siguiente ejemplo nos permite personalizar el color primario para cambiarle el color de selección a un `ToggleButton`.

```
<style name="Toggle" >
  <item name="colorSecondary">#f00</item>
</style>
```


	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

10.3.6. Clase R²⁵

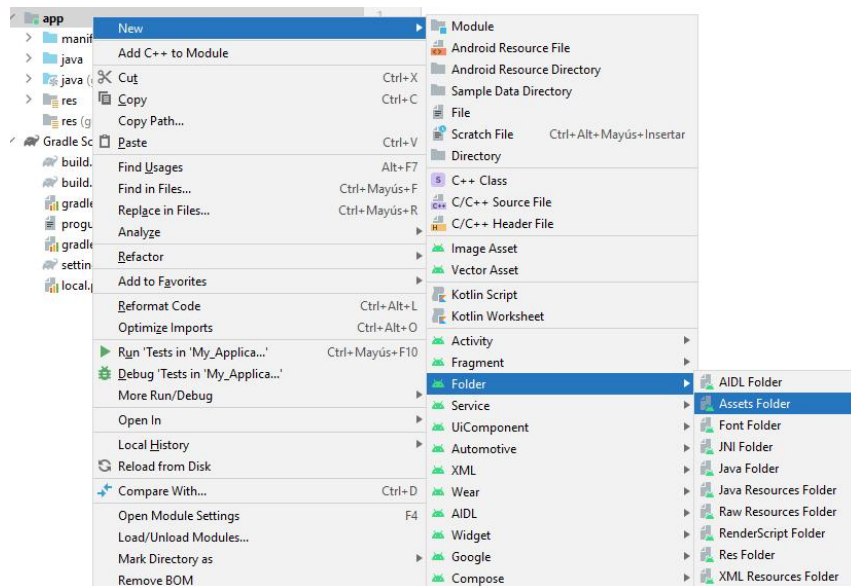
Es una clase que genera de forma automática *Android Studio*, y que no debe modificarse a mano, en el que se asigna un identificador único (entero estático) a cada recurso que exista en el directorio *res*. Cuando se añade un nuevo recurso automáticamente se crea su identificador en la clase *R*.

Estos son los identificadores que se usaran para acceder a los recursos tanto desde los ficheros de código java como desde los ficheros *XML*.

10.4. Assets

Existe una carpeta denominada *assets* que también sirve para guardar recursos. La diferencia con la carpeta *res* estriba principalmente en que a estos recursos, al estar fuera de la carpeta *res*, no tienen identificador por lo que no podemos usar los métodos anteriores para acceder a ellos sino que estos recursos son accesibles directamente por su ruta dentro del directorio *Assets*.

Se puede ver este directorio como un sistema de archivos donde guardar cualquier tipo de fichero. Para crear el directorio *Assets* podemos usar el siguiente asistente:




Imaginemos que queremos cargar una fuente almacenada en el directorio *fonts* de la carpeta *assets* para cambiar la fuente de un *textview* de nuestra *activity*. Podríamos usar el siguiente código para ello.

```
Typeface fuente=Typeface.createFromAsset(getApplicationContext().getAssets(),"fonts/fuente.ttf");
textView.setTypeface(fuente);
```

Una de las ventajas del directorio *assets*, es que, a diferencia²⁶ del directorio *res*, podemos crear una estructura de directorios para organizar los recursos. Por ejemplo podemos crear una carpeta *fonts* para almacenar las fuentes que usemos en nuestra aplicación o la carpeta de imágenes y dentro de ella una carpeta con las imágenes de cada *activity*.

²⁵ <https://developer.android.com/guide/topics/resources/accessing-resources.html>

²⁶ <https://stackoverflow.com/questions/5583608/difference-between-res-and-assets-directories>

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

11. Apéndice I: RelativeLayout

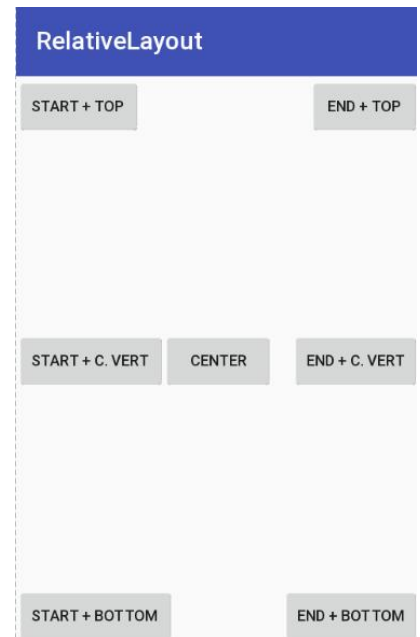
RelativeLayout organiza los componentes que están situados dentro de él de forma relativa a sus hermanos o a su padre. Permite construir *layouts* que no poseen otros *layouts* dentro de él, lo que permite mejorar la eficiencia.

Por ejemplo podemos situar componentes que estén situados en la esquina superior derecha del *layout* o debajo de otro componente.

Veremos a través de los siguientes ejemplos una muestra de su funcionamiento.

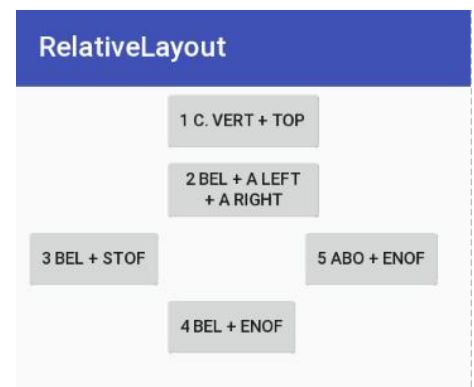
En este ejemplo colocaremos componentes en relación con el *layout*, usaremos para ello las siguientes propiedades:


- `layout_alignParentTop`: coloca el borde superior del componente en el borde superior del padre.
- `layout_alignParentBottom`: coloca el borde inferior del componente en el borde inferior del padre.
- `layout_alignParentStart`: coloca el borde de comienzo del componente en el borde de comienzo del padre.
- `layout_alignParentEnd`: coloca el borde de fin del componente en el borde de fin del padre.
- `layout_alignParentLeft`: coloca el borde izquierdo del componente en el borde izquierdo del padre.
- `layout_alignParentRight`: coloca el borde derecho del componente en el borde derecho del padre.
- `layout_centerHorizontal`: centra el componente de forma horizontal en el padre.
- `layout_centerVertical`: centra el componente de forma vertical en el padre.
- `layout_centerInParent`: centra el componente de forma horizontal y vertical en el padre.



Las siguientes propiedades nos permiten situar componentes en relación a otro componente.

- `layout_above`: coloca el borde inferior del componente sobre el componente especificado.
- `layout_below`: coloca el borde superior del componente debajo del componente especificado.
- `layout_alignBaseline`: coloca la base del componente con la base del componente especificado.
- `layout_alignTop`: coloca el borde superior del componente en el borde superior del componente especificado.
- `layout_alignBottom`: coloca el borde inferior del componente en el borde inferior del componente especificado.
- `layout_alignStart`: coloca el borde de comienzo del componente en el borde de comienzo del componente especificado.
- `layout_alignEnd`: coloca el borde de fin del componente en el borde de fin inferior del componente especificado.
- `layout_alignLeft`: coloca el borde izquierdo del componente en el borde izquierdo del componente especificado.
- `layout_alignRight`: coloca el borde derecho del componente en el borde derecho del componente especificado.
- `layout_toStartOf`: coloca el borde de fin del componente en el borde de comienzo del componente especificado.
- `layout_toEndOf`: coloca el borde de comienzo del componente en el borde de fin del componente especificado.



	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

- `layout_toLeftOf`: coloca el borde derecho del componente en el borde izquierdo del componente especificado.
- `layout_toRightOf`: coloca el borde izquierdo del componente en el borde derecho del componente especificado.

En el ejemplo anterior se han definido los botones en el orden especificado en su texto:

1. Este botón se ha definido centrado horizontalmente y alineado en la parte superior del padre.
2. En este caso este botón se ha definido debajo y alineado a la izquierda y derecha del botón 1. Como el texto de este botón es más largo que el botón 1 se reparte en más de una línea.
3. El botón 3 se ha definido debajo del botón 2 y se ha definido que el final del botón coincida con el principio del botón 2.
4. Aquí el botón 4 se ha definido debajo y que su principio coincida con el final del botón 3.
5. Por último el botón 5 se ha definido sobre el botón 4 y que su comienzo coincida con el final de este.

Para ver más información sobre *RelativeLayout* se puede consultar la siguiente guía proporcionada por Google:

- <https://developer.android.com/guide/topics/ui/layout/relative.html>

Para ver sus propiedades se puede consultar los enlaces:

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