

Diseño y lógica

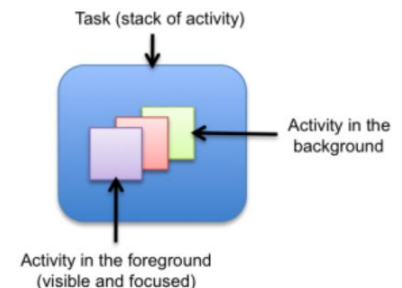
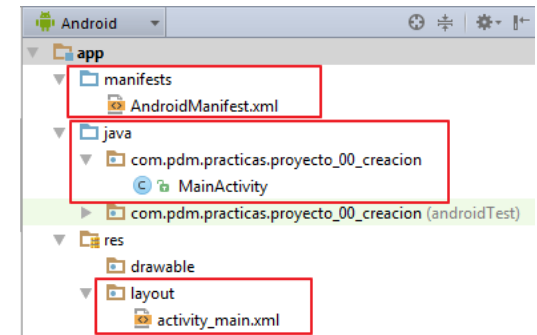
Componentes de una aplicación



Activity



- Una **Activity** (actividad) es un componente de la aplicación que **proporciona una pantalla con la que el usuario puede interactuar para hacer una determinada acción**.
- Para cada actividad hay que desarrollar, por tanto, la interfaz gráfica de usuario, el código java que implemente las acciones y definirla en el fichero AndroidManifest.
- Una aplicación puede tener varias actividades. Existe por tanto una **pila de "activities"**.



Desarrollo de UI



- La estructura de la UI y los objetos que la componen se pueden declarar en un **archivo XML** (diseño declarativo) o mediante **código Java/Kotlin** (diseño programático).

¿Cómo crear una etiqueta con XML?

```
<TextView android:id="@+id/textview"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:text="Hola mundo"/>
```

Tecleando el código xml
o utilizando el asistente
gráfico y guardando en
el correspondiente
/res/layout/fichero.xml

¿Cómo crear una etiqueta dinámicamente?

```
...  
TextView tv = new TextView(this);  
tv.setText("Hola mundo");  
setContentView(tv);  
...
```

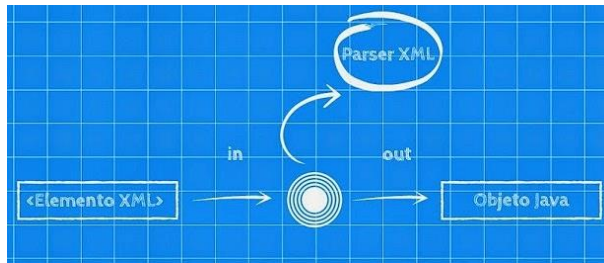
Tecleando el código java
y guardando en el
correspondiente
/src/actividad.java

- Si se sabe a priori todos los elementos que va a tener la pantalla, **normalmente se utilizan los ficheros XML** y el código de la aplicación podrá modificar las propiedades de los objetos durante la ejecución.
- Cuando haya elementos que no se conocen en tiempo de compilación, hay que crearlos dinámicamente con código Java lo que puede resultar complejo y poco eficiente.
- Recuerda que un principio importante en el diseño de *software* es que **conviene separar todo lo posible el diseño, los datos y la lógica de la aplicación** (patrón de diseño MVC).

Ventajas del diseño declarativo de UI



- Los proyectos en Android tienen la capacidad de implementar el poder de XML para reducir la complejidad de diseño de interfaz, permitiéndonos declarar la estructura en un simple archivo definido por elementos muy intuitivos.

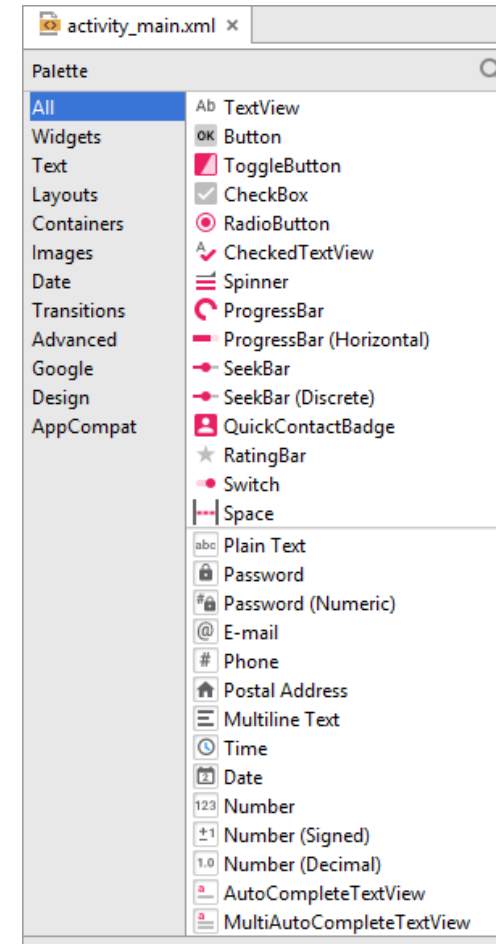


- Estos archivos luego son analizados con una **librería de Parsing XML** para generar automáticamente todo el código Java necesario a partir de una estructura predefinida.
- La idea es usar el estilo declarativo para crear la forma de la interfaz y luego usar el estilo programático para implementar los eventos sobre la interfaz. La combinación de ambas formas desata un poder de flexibilidad increíble.
- La ventaja de declarar el interfaz de usuario en un archivo XML es que permite separar mejor la presentación de la aplicación respecto al código que controla su comportamiento; además, al tener la descripción del interfaz de manera externa, se puede modificar y adaptar sin modificar el código fuente y recompilar y también se pueden crear diferentes estructuras del interfaz en función de la orientación de la pantalla, su tamaño o idioma del sistema.

Carpeta res/layout



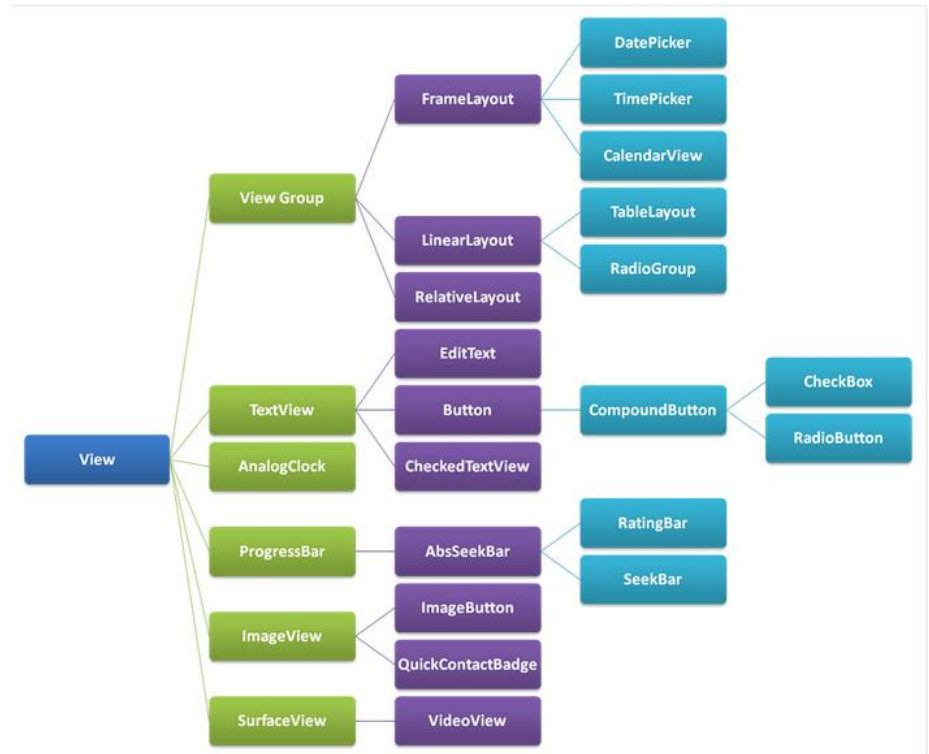
- La interfaz de usuario se define en los archivos XML del directorio **res/layout**. Cada pantalla distinta tendrá un código XML diferente.
- Un **layout** define la estructura visual de una interfaz de usuario.
- De inicio, Android pone a nuestra disposición la **paleta** con una gran cantidad de controles, como cuadros de texto, botones, listas desplegables o imágenes, aunque también existe la posibilidad de extender la funcionalidad de estos controles básicos o crear nuestros propios controles personalizados.
- Cada control tiene propiedades (atributos del elemento en el fichero XML) que pueden fijarse.
- (Nota: en muchos textos encontrarás los términos *View*, *Vista*, *Control*, *Widget* o *Elemento* como sinónimos).



View



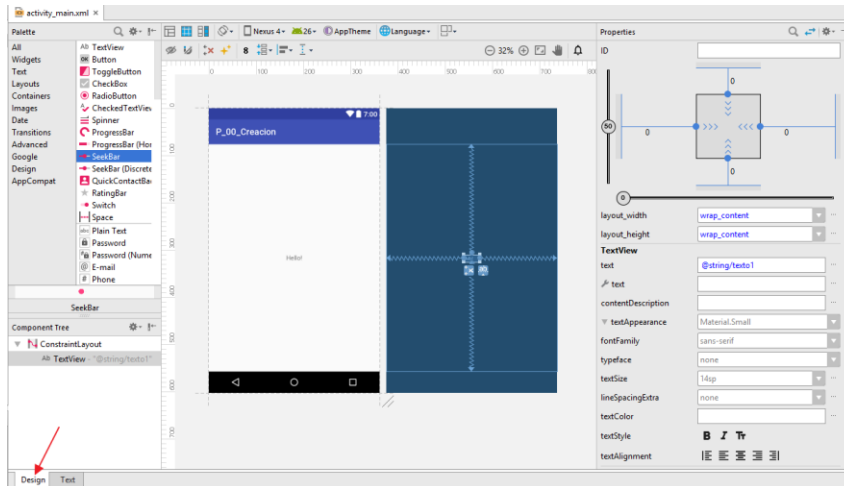
- La interfaz de usuario está basada en una **jerarquía** de clases descendientes de la clase *View* o de su subclase *ViewGroup*.
- Los objetos *View* son los componentes básicos con los que se construye la interfaz gráfica. La mayoría son los objetos visibles con los que el usuario puede interactuar (un botón, una imagen, una etiqueta de texto como en el utilizado en la plantilla,...).
- Los *ViewGroup* no son visibles (como por ejemplo, los ***Layout***) ya que están destinados a controlar la distribución, posición y dimensiones de los objetos visibles que se insertan en su interior.



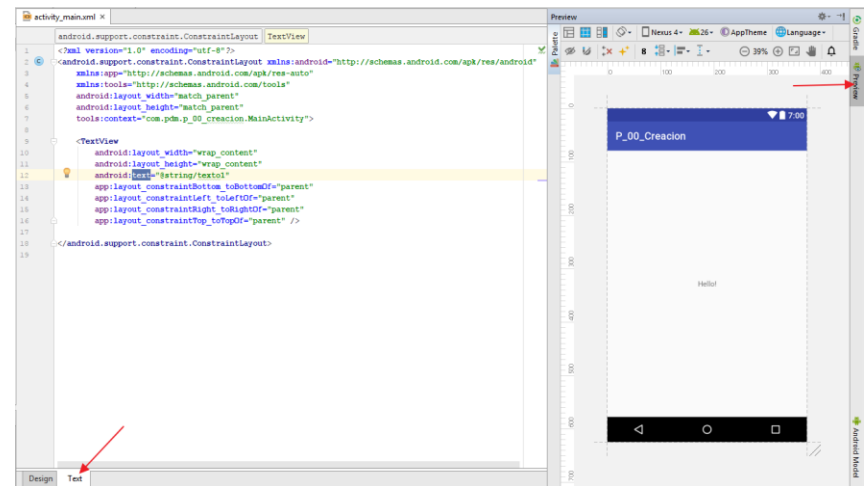
Herramienta diseñador de AS



Modo diseño



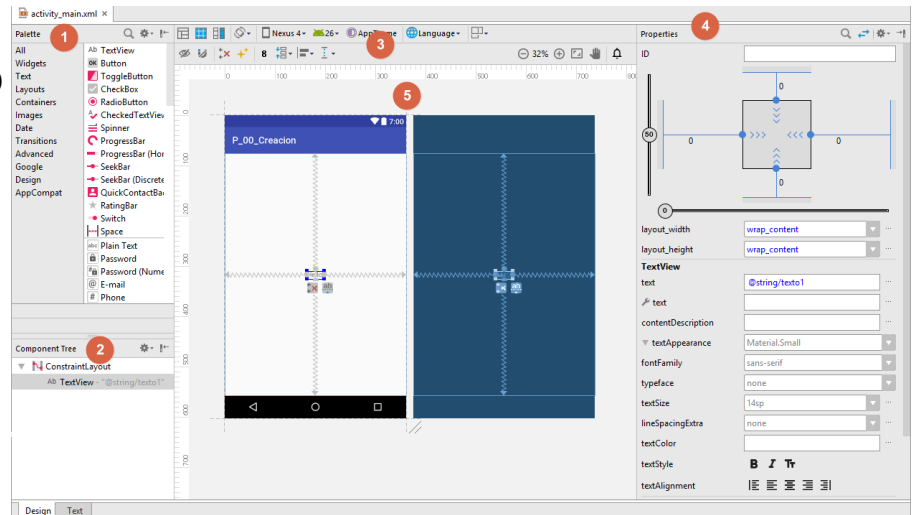
Modo texto



Modo diseño



1. Paleta: conjunto de vistas proporcionado por el SDK agrupados en categorías. Se agregan al diseño arrastrando desde la paleta.
2. Árbol de componentes: las interfaces de usuario se construyen utilizando una estructura jerárquica
3. Barra de herramientas: permite acceder rápidamente a una amplia gama de opciones.
4. Propiedades de la vista seleccionada: para ajustar el comportamiento y la apariencia de esa vista
5. Editor de diseño: Muestra el diseño

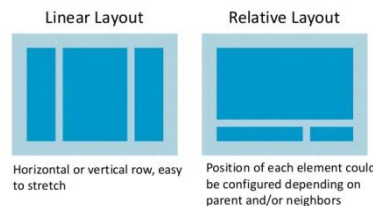


Elemento *layout*



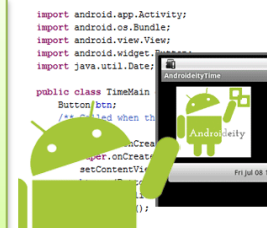
- Entre los diferentes tipos de vistas que tiene Android, hay uno que tiene especial relevancia: el **layout** (contenedor). Se trata de una vista que puede contener a otras vistas y contenedores.
- Un *layout* es un objeto que **representa el espacio contenedor de todas las vistas dentro de la actividad**. En él se define la estructura y el orden de todos los elementos para que el usuario pueda interactuar con la interfaz.
- Son elementos no visuales (extienden a la clase base *ViewGroup*) destinados a controlar la distribución, posición y dimensiones de los controles que se insertan en su interior.
- Según el tipo de *layout* que empleemos los elementos que insertemos se comportarán de una manera u otra y tendrán distintos atributos.

Common ViewGroups

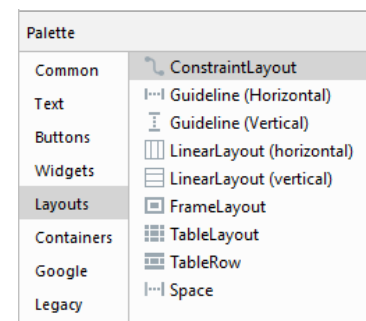


27

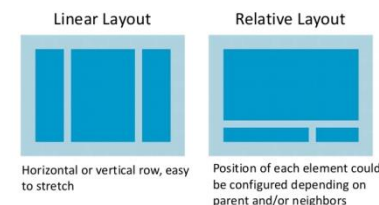
Tipos de layout



- **ConstraintLayout:** Desde AS 2.2, permite crear fácilmente diseños grandes y complejos.
- **LinearLayout:** diseño que sitúa los elementos que contiene en línea, uno a continuación del otro (sin superponerlos), ya sea vertical u horizontalmente. Las vistas se muestran en el orden en que se añaden al diseño.
- **FrameLayout:** el diseño más sencillo. No ordena las vistas que se incluyen en él, simplemente las coloca una sobre otra (hasta 9), en el orden en que se añaden. Por este motivo, normalmente sólo contendrá una vista y se utiliza principalmente para mostrar imágenes o lienzo gráfico, aunque también es la mejor opción cuando queremos que unas vistas se superpongan a otras.
- **TableLayout:** en este diseño las vistas se colocan en forma de tabla, esto es, repartidas en filas y columnas. El contenido del TableLayout son uno o más TableRow (que es otro diseño), ordenados verticalmente, cada uno conteniendo una fila de la tabla. Cada **TableRow** se comporta, básicamente, como un **LinearLayout** horizontal (con restricciones), conteniendo las vistas de cada fila. A pesar de esta estructuración, el conjunto total se comporta como una tabla real, con filas y columnas correctamente alineadas.
- **RelativeLayout:** este es uno de los diseño más versátil del que se disponía hasta el año 2017 (por tanto, en antiguos proyectos era el más utilizado). En él cada vista se puede situar más o menos donde queramos. No usa posiciones absolutas (coordenadas), sino relativas a otras vistas ya incluidas en el diseño. O sea, que podemos indicar que una vista se coloque debajo o a la izquierda de otra, que dos vistas se alineen verticalmente, que se muestren centrada una respecto a otra, etc. Nos permite crear interfaces complejas sin necesidad de anidar numerosos diseños más básicos.
- **GridLayout:** Este tipo fue incluido a partir de la API 14 (Android 4.0) y sus características son similares al TableLayout, ya que se utiliza igualmente para distribuir los diferentes elementos de la interfaz de forma tabular, distribuidos en filas y columnas. La diferencia entre ellos estriba en la forma que tiene el GridLayout de colocar y distribuir sus elementos hijos en el espacio disponible.



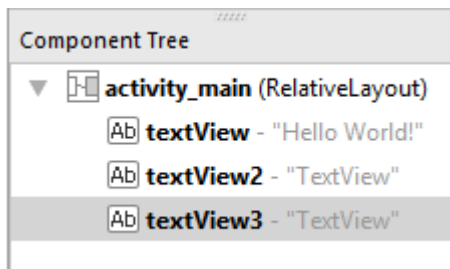
Common ViewGroups



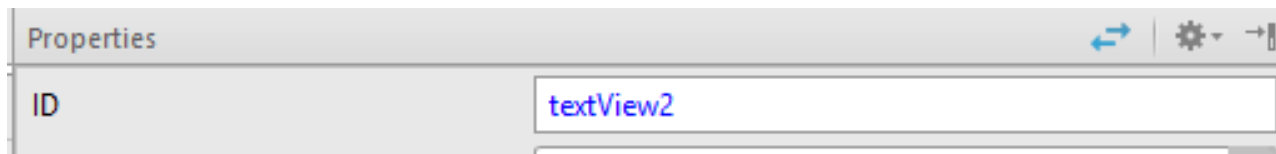
id : Propiedad fundamental de los objetos View



- **id:** ID del objeto, **con el que se identifica más tarde en el código**. En la vista textual, el valor de su atributo se escribe el nombre precedido de "@+id/" (**android:id="@+id/textView2"**) y en la vista gráfica con el nombre asignado simplemente. Esto tendrá como efecto que al compilarse el proyecto se genere automáticamente una nueva constante en la clase R para dicho control y que sea accesible desde código Java. El carácter @ significa que se trata de un identificador de recurso (es decir se definirá en el fichero *R.java*). El carácter + significa que el recurso ha de ser creado en este momento. También existen ciertos identificadores que ya han sido definidos en el sistema. Por ejemplo, más adelante utilizaremos "@android:id/list" para crear un ListView.



- Nota1: observa que el asistente de creación del proyecto, no ha incluido este atributo. La razón es que en el código java no hace ningún uso de la vista TextView, por lo tanto lo ha omitido.
- Nota2: Normalmente al lanzar sobre el layout nuevos objetos, Android Studio asigna un identificador cuyo contenido es el tipo de control (textView, button,etc.) seguido de un nº autoincrementado



Vistas y compilación



- Cada *View* declarada en un fichero XML de la carpeta `/res/layout` al compilar se convierte en un elemento que podrá cargarse desde código a través del fichero **R.java**.
- Ejemplo: `R.layout.activity_main`


A screenshot of an IDE window showing three tabs: 'MainActivity.java', 'activity_main.xml', and 'R.java'. The 'R.java' tab is active, displaying the following code:

```
public static final int abc_select_dialog_material=0x7f040017;  
public static final int activity_main=0x7f040018;  
public static final int notification_media_action=0x7f040019;
```

A red arrow points from the right side of the image to the 'activity_main' resource ID in the second line of code. Above the code, a yellow banner states: 'Files under the build folder are generated and should not be edited.'

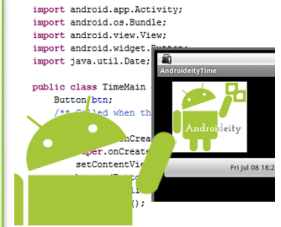
Algunas propiedades visuales



- **layout:width, layout:height:** Permiten ajustar el ancho y alto de la vista. Se puede indicar una dimensión concreta, por ejemplo 200dp, aunque lo habitual es utilizar uno de los valores:
 - *wrap_content*: ajusta el tamaño a las dimensiones necesarias para representar el contenido.
 - *match_parent*: ajusta el tamaño al máximo posible según el *Layout* padre que la contiene.
- **layout:margin, layout:margin_bottom, layout_margin_left, layout:margin_right, layout:margin_top:** Establecen un margen exterior a la vista.
- **padding, paddingBottom, paddingTop, paddingLeft, paddingRight:** Establece un margen interior en la vista.
- **gravity:** Centra o justifica la vista dentro del *layout*.
- **visibility:** Permite hacer invisible una vista. Valores posibles:
 - *visible*: la vista es visible
 - *invisible*: la vista es invisible pero ocupa lugar
 - *gone*: la vista es invisible pero no ocupa lugar
- **background:** Permite establecer una imagen o un color de fondo.
- **style:** Permite aplicar un estilo a la vista.
- El editor de layout solo muestra las propiedades más comunes, pero podemos ver todas [All Attributes](#)
- Algunas propiedades tienen la marca . Esto indica que hay un diálogo de configuración disponible para ayudar a seleccionar un valor de propiedad adecuado. Las propiedades para las cuales se dispone de un número finito de opciones válidas presentarán un menú desplegable.



Algunas propiedades de comportamiento



- **clickable**: Indica si la vista reacciona ante eventos de tipo onClick (se pulsa sobre la vista).
- **onClick**: Nombre del método que será invocado cuando ocurra un evento onClick.
- **longClickable**: Indica si la vista reacciona a eventos de tipo pulsación larga (más de un segundo).
- **focusable**: Indica si la vista puede tomar el foco.
- **focusableInTouchMode**: Establece que cuando el dispositivo tenga capacidades de pantalla táctil y se pulsa sobre la vista esta tomará el foco. Hay que diferenciarlo de clickable. Por ejemplo, nos suele interesar que un botón pueda recibir evento *onClick* pero no que coja el foco.
- Nota: Fijar estas propiedades muchas veces no es recomendable, ya que estaríamos mezclando diseño y lógica.

Properties	
capitalize	
clickable	<input type="checkbox"/>
contentDescription	
contextClickable	<input type="checkbox"/>
digits	
drawableTint	
drawableTintMode	
editable	<input type="checkbox"/>
elegantTextHeight	<input type="checkbox"/>
elevation	
ellipsize	
enabled	<input type="checkbox"/>
focusable	<input type="checkbox"/>
focusableInTouchMode	<input type="checkbox"/>

Algunas propiedades para texto



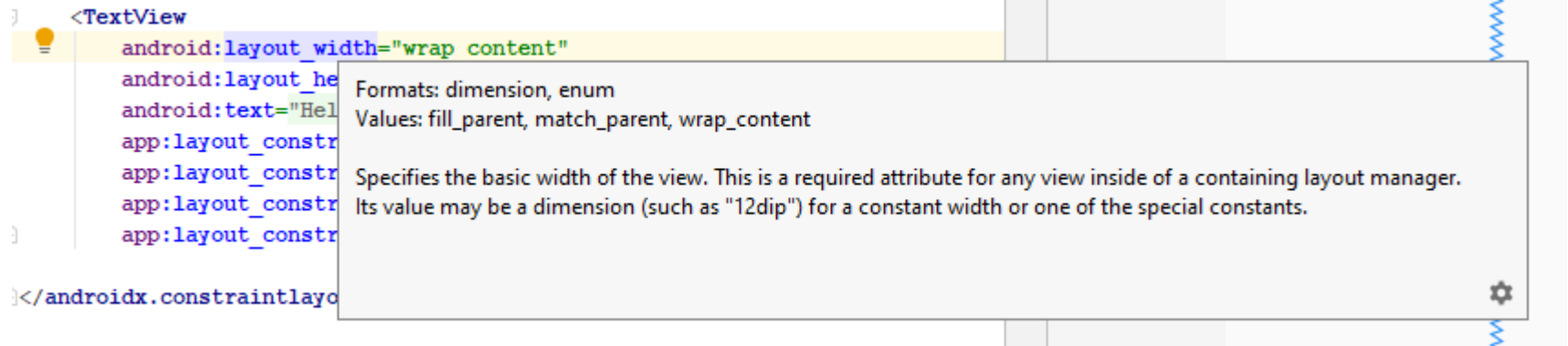
- **text:** Texto que se mostrará
- **textSize:** Tamaño del texto
- **textStyle:** Estilo del texto (negrita ó itálica)
- **textAppearance:** Permite definir conjuntamente el tipo de fuente, tamaño del texto, color,...
- **textColor:** Color del texto
- **textColorLink:** Color del texto para hipervínculos.
- **textColorHighlight:** Color del texto cuando es seleccionado
- **text_scale_x:** Deforma el texto con un factor de escala horizontal.
- **textColorHint:** Color del texto de indicación.
- **typeface:** Tipo de fuente usada en el texto
- **gravity:** Cómo el texto es alineado dentro de la vista
- **width, height:** Hace que el texto tenga exactamente el ancho o alto especificado
- **hint:** Texto que se mostrará, normalmente dentro de un EditText antes de que el usuario haya tecleado, aunque en otro color para indicar algún tipo de instrucciones. Por ejemplo "Introduzca aquí su nombre".

text	@string/texto1	
textAllCaps		
textColor	@android:color/secondary_text_material_light	
textColorHighlight	@android:color/highlighted_text_material	
textColorHint	@android:color/hint_foreground_material_light	
textColorLink	#D81B60	
textCursorDrawable	@android:drawable/text_cursor_material	
textIsSelectable		
textScaleX		
textSize	14sp	
▼ textStyle	normal	
normal	<input type="checkbox"/> false	
bold	<input type="checkbox"/> false	
italic	<input type="checkbox"/> false	
▼ theme		
textAppearance	@android:style/TextAppearance.Material	
textEditNoPasteWindowLayout	@android:layout/text_edit_no_paste_window	
textEditPasteWindowLayout	@android:layout/text_edit_paste_window	
textEditSideNoPasteWindowLayout	@android:layout/text_edit_side_no_paste_window	
textEditSidePasteWindowLayout	@android:layout/text_edit_side_paste_window	
textEditSuggestionItemLayout	@android:layout/text_edit_suggestion_item_material	
textSelectHandle	@android:drawable/text_select_handle_middle_r	
textSelectHandleLeft	@android:drawable/text_select_handle_left_mate	
textSelectHandleRight	@android:drawable/text_select_handle_right_ma	
theme		

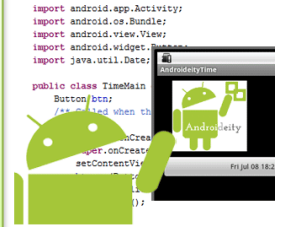
Información sobre un atributo



- Seleccionado el atributo en modo texto, al pulsar Ctrl+Q se obtiene la referencia (a veces!):

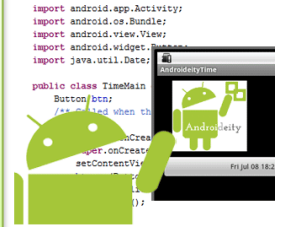


Unidades de medidas



- **dp** – píxeles independientes de la densidad: es una unidad abstracta que se basa en la densidad de píxeles de la pantalla. Sirve para garantizar que las cosas tengan las mismas medidas físicas en cualquier pantalla. Como regla, hay que tener en cuenta que 160dp equivaldrá a una pulgada (o, lo que es lo mismo, 63dp serán un centímetro) en cualquier pantalla. **RECOMENDADA**.
- **sp** – píxeles independientes de la escala: es igual que la unidad dp, pero escalada según el tamaño de fuente escogido por el usuario. Es la unidad **recomendada para definir el tamaño de los textos** que se muestran en pantalla.
- **pt** – puntos: representan 1/72 partes de una pulgada (o, más o menos, 1/28 de un centímetro) en cualquier pantalla.
- **px** – píxeles: no se recomienda utilizar esta unidad, porque las pantallas tienen diferentes densidades y tamaños, por lo que los elementos dimensionados en píxeles se pueden ver de formas muy diferentes.
- **mm** – milímetros: medida directa del elemento en pantalla, en unidades más internacionales.
- **in** – pulgadas: medida directa del elemento en pantalla, en unidades anglosajonas.

Clase Activity



- La lógica de las diferentes pantallas de una aplicación Android se definen mediante objetos de tipo Activity con ficheros escritos en Java (o el nuevo Kotlin).
- Normalmente las aplicaciones tienen una actividad fijada como punto de entrada (MainActivity). AS al crear el proyecto nos ha proporcionado una plantilla con la definición de una nueva clase **MainActivity** que extiende en este caso de un tipo especial de [Activity](#) llamado **AppCompatActivity** (que permite la utilización de la *Action Bar* en dispositivos antiguos).
- La clase Activity representa una ventana de Android y tiene todos los métodos necesarios para crear y mostrar los objetos que se han dispuesto en el archivo xml del diseño.
- El único método que tiene (por ahora) es el método **onCreate()**, llamado cuando se crea por primera vez la actividad.
- En dicho método onCreate() nos encontramos (por ahora) además de la llamada a su implementación en la clase padre, que llama al método **setContentView(R.layout.activity_main)**.

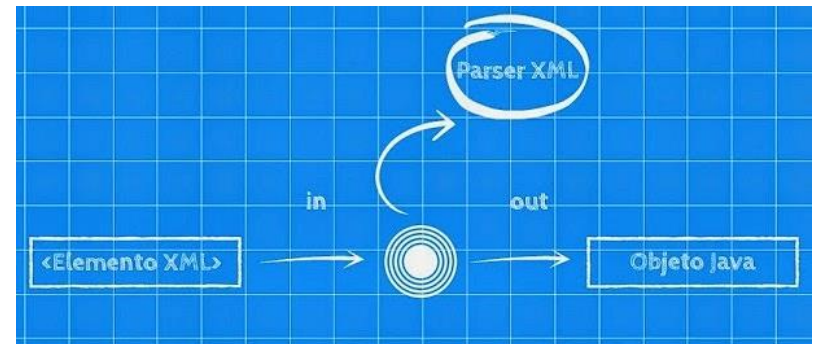
```
MainActivity.java x
MainActivity
1 package com.pdm.p_00_creacion;
2
3 import ...
4
5
6 public class MainActivity extends AppCompatActivity {
7
8     @Override
9     protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.activity_main);
12     }
13 }
```

- Con esta llamada estaremos indicando que se debe establecer como interfaz gráfica de esta actividad la definida en el recurso R.layout.activity_main, que no es más que la que hemos especificado en el fichero /src/main/res/layout/activity_main.xml. Vemos la utilidad de las diferentes constantes de recursos creadas automáticamente en la clase R al compilar el proyecto.

Accediendo a las vistas del layout desde el código

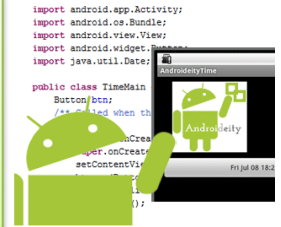


- Muchas veces aparece el término **inflate** (inflar) en la programación Android. Inflar significa "parser" uno a uno los elementos de un archivo layout XML, para generar el código Java necesario y luego agregar los objetos a la memoria.
- Quien se encarga de este trabajo es el método **setContentView()** de la clase Activity. Dicho método analiza el archivo XML, traduce a objetos cada componente, le asigna los atributos, establece contenedores y todas las relaciones padre e hijo necesarias. Este método lo usaremos en la sección onCreate() de cada actividad.
- Para acceder a un determinado elemento del layout, utilizamos el método **findViewById()** que recibe como parámetro una ruta al identificador de la vista en el fichero R.java:
`findViewById(R.id.idDelElemento)`
- Supongamos que nuestro layout tenemos un TextView identificado por el atributo `id="@+id/textView"`, en el código Java, una vez cargado el layout en el método onCreate(), podemos referenciarlo.



```
public class MainActivity extends AppCompatActivity {  
  
    TextView tv1;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        tv1 = (TextView) findViewById(R.id.textview);  
        tv1.setText("Mi mamá me mima");  
    }  
  
    @Override  
    public boolean onCreateOptionsMenu(Menu menu) {  
        // Inflate the menu; this adds items to the action
```

Declaración de la actividad en AndroidManifest.xml



<manifest>	
→ <uses-sdk>	Atributos: minSDKVersion, maxSDKVersion, targetSDKVersion
→ <uses-configuration>	Atributos: reqFiveWayNav, reqHardKeyboard, reqKeyboardType, reqNavigation, reqTouchScreen
→ <uses-feature>	Atributos: glEsVersion, name, required
→ <supports-screens>	Atributos: smallScreens, normalScreens, largeScreens, anyDensity
→ <application>	Atributos: allowCreateUserData, allowTaskReparenting, backupAgent, debuggable, description, enabled, hasCode, icon, killAfterRestore, label, manageSpaceActivity, name, permission, persistent, process, restoreAnyVersion, taskAffinity, theme
→ <activity>	Atributos: allowTaskReparenting, alwaysRetainTaskState, clearTaskOnLaunch, configChanges, enabled, excludeFromRecents, exported, finishOnTaskLaunch, icon, label, launchMode, multiprocess, name, noHistory, permission, process, screenOrientation, stateNotNeeded, taskAffinity, theme, windowSoftInputMode
→ <intent-filter>	Atributos: icon, label, priority
→ <meta-data>	Atributos: name, resource, value
→ <service>	Atributos: enable, exported, icon, label, name, permission, process
→ <intent-filter>	Atributos: icon, label, priority
→ <meta-data>	Atributos: name, resource, value
→ <provider>	Atributos: authorities, enable, exported, grantUriPermissions, icon, initOrder, label, multiprocess, name, permission, process, readPermission, syncable, writePermission
→ <grant-uri-permission>	Atributos: path, pathPattern, pathPrefix
→ <meta-data>	Atributos: name, resource, value
→ <receiver>	Atributos: enabled, exported, icon, label, name, permission, process
→ <intent-filter>	Atributos: icon, label, priority
→ <meta-data>	Atributos: name, resource, value
→ <uses-permission>	Atributos: name
→ <permission>	Atributos: description, icon, label, name, permissionGroup, protectionLevel
→ <instrumentation>	Atributos: functionalTest, handleProfiling, icon, label, name, targetPackage

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3    package="com.pdm.p_00_creacion">
4
5    <application
6      android:allowBackup="true"
7      android:icon="@mipmap/ic_launcher"
8      android:label="P_00_Creacion"
9      android:roundIcon="@mipmap/ic_launcher_round"
10     android:supportRtl="true"
11     android:theme="@style/AppTheme">
12     <activity android:name=".MainActivity">
13       <intent-filter>
14         <action android:name="android.intent.action.MAIN" />
15
16         <category android:name="android.intent.category.LAUNCHER" />
17       </intent-filter>
18     </activity>
19   </application>
20
21 </manifest>
  
```