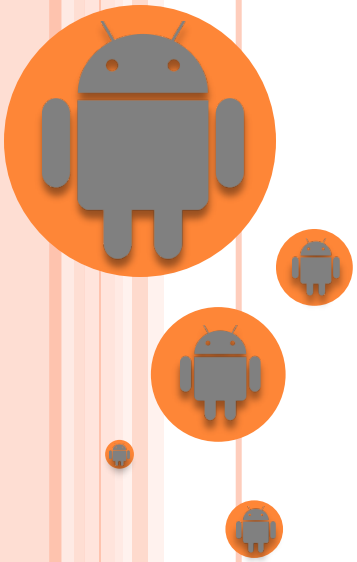
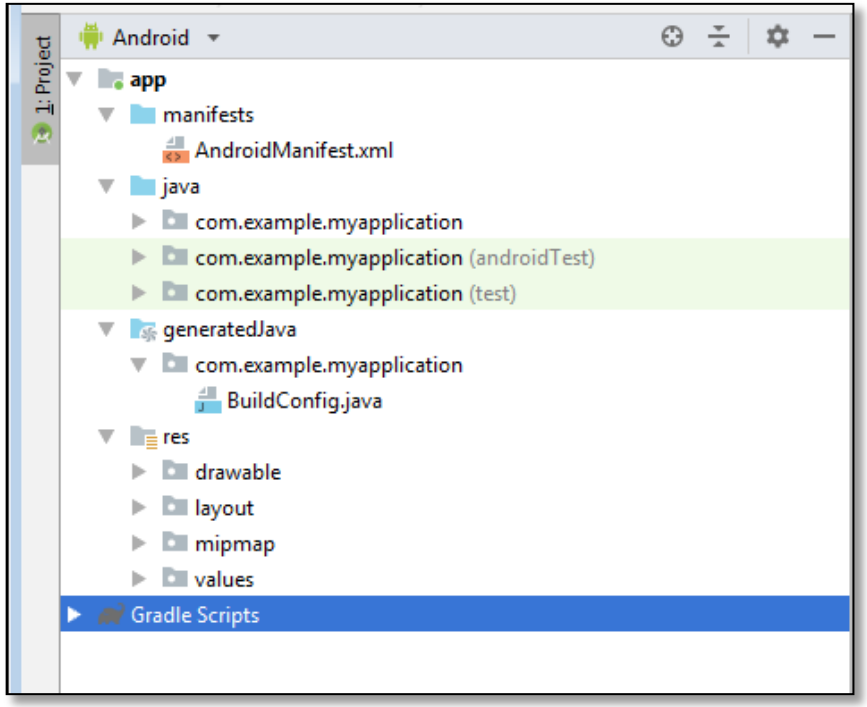


Estructura y componentes básicos de una aplicación

Estructura básica de una aplicación

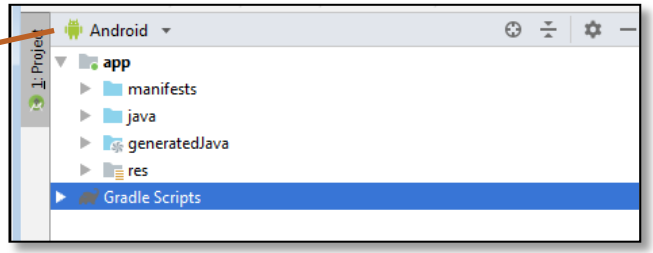
- Android Studio nos genera automáticamente la estructura necesaria.
- Esta estructura será común a cualquier aplicación.



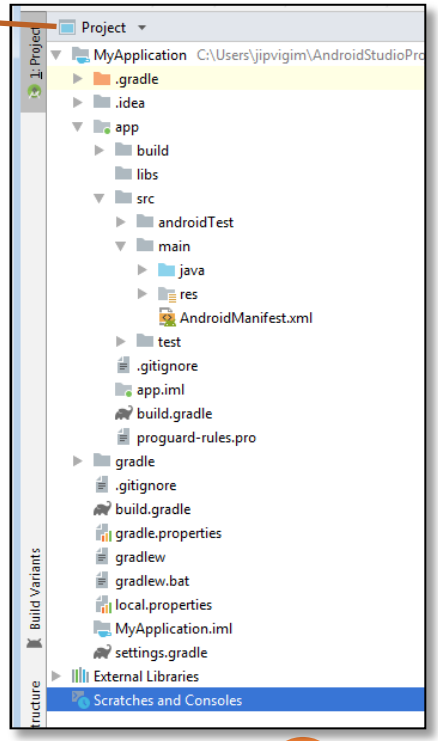
Estructura básica de una aplicación

- Android Studio permite distintas visualizaciones

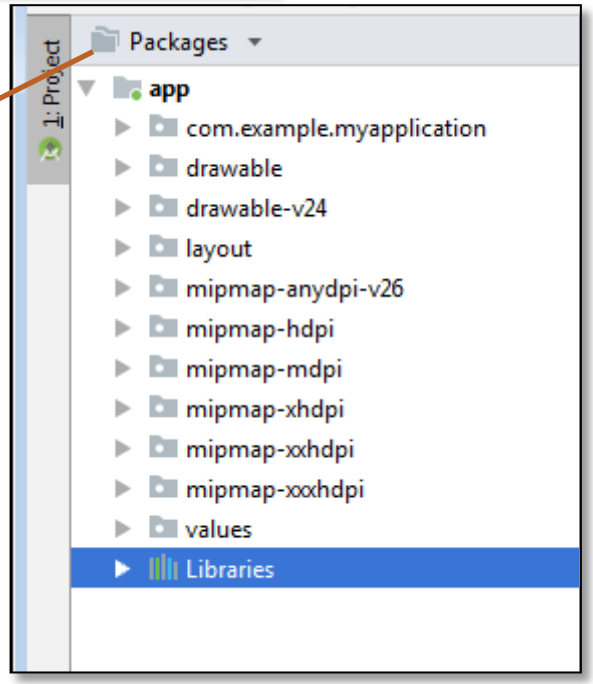
Android



Project

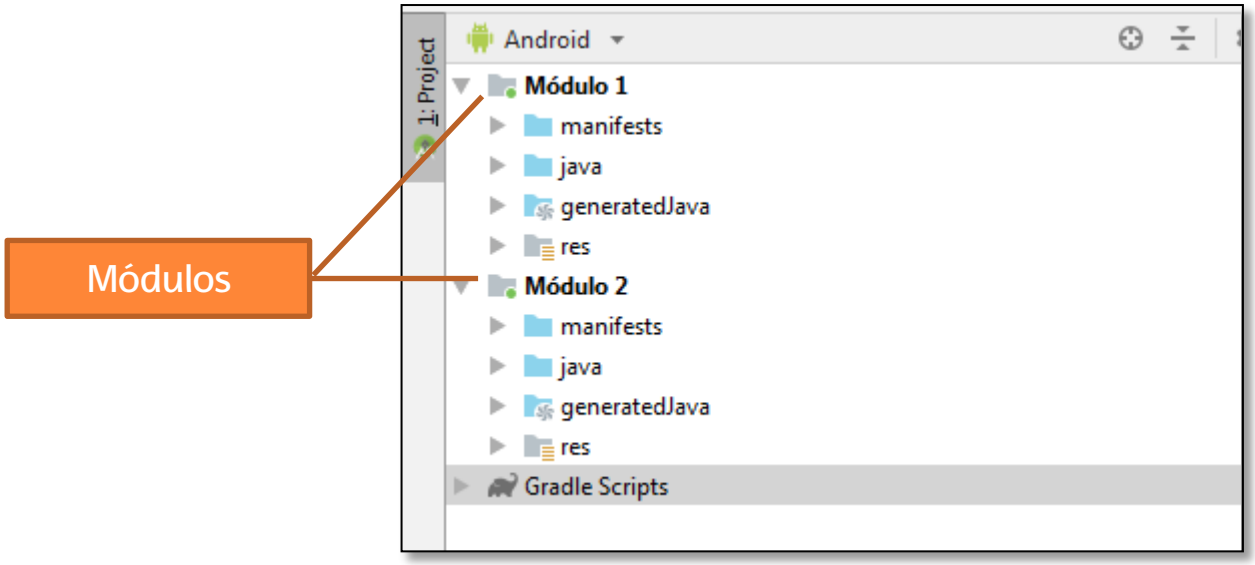


Packages



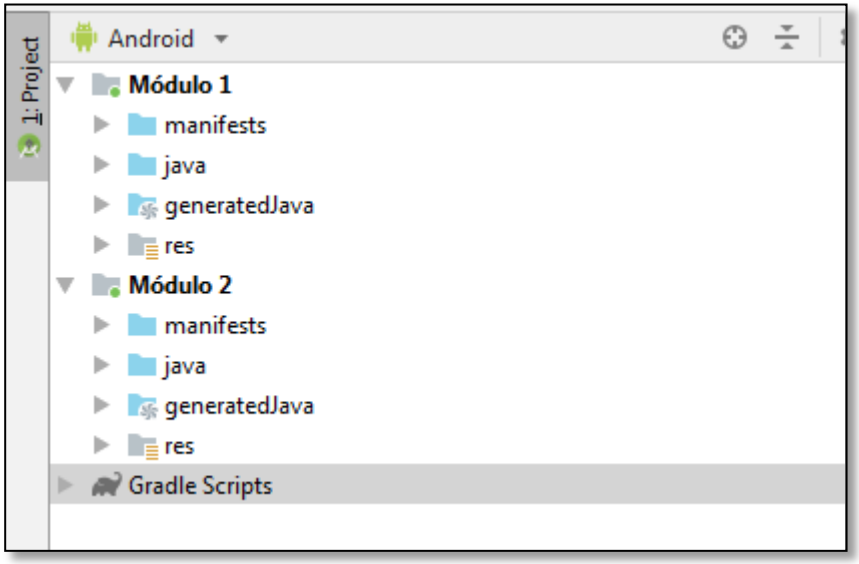
Estructura básica de una aplicación

- La visualización *Project* es la más parecida a la forma de trabajar en Eclipse
- La visualización *Android* es la más habitual
- El proyecto puede contener distintos *módulos* (*apps*).



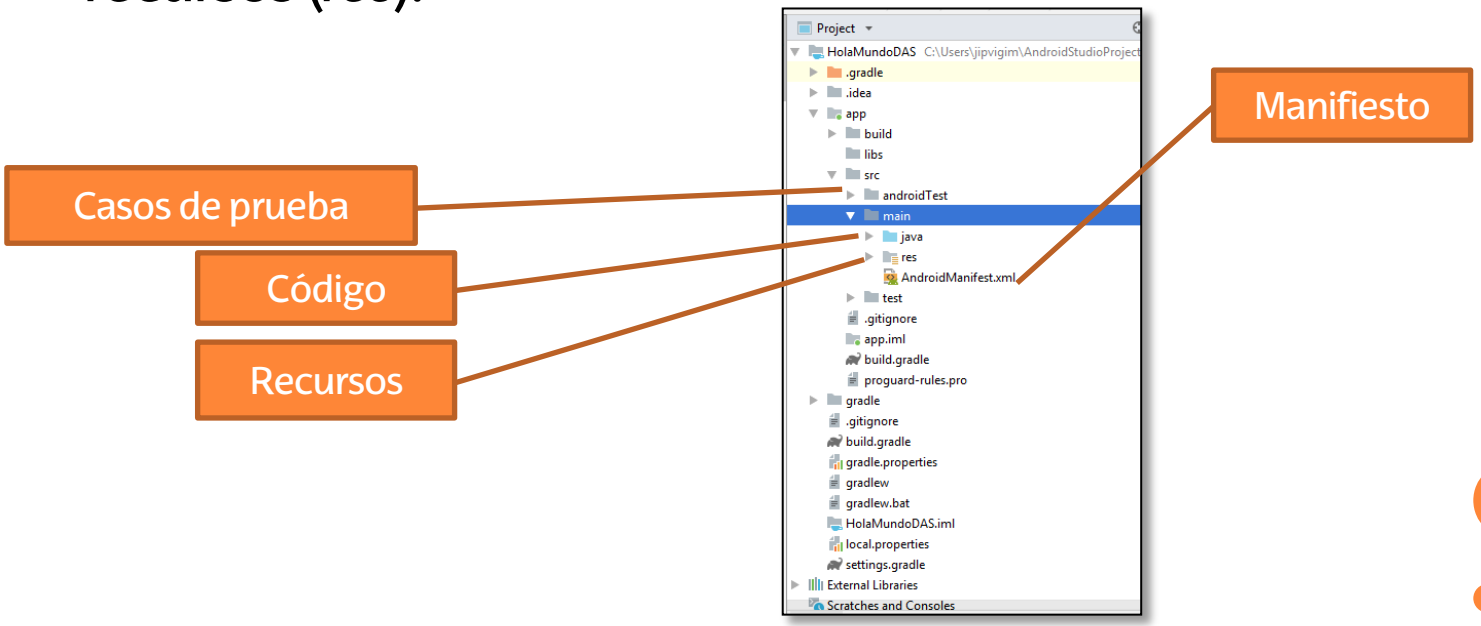
Estructura básica de una aplicación

- En general, sólo se usan módulos dentro del mismo proyecto si comparten código (librerías)
- Cada *módulo* contiene todos los elementos necesarios para esa aplicación.



Estructura básica de una aplicación

- La carpeta *src*:
  - Contiene los casos de prueba (*androidTest*), el manifiesto (AndroidManifest.xml) y el resto de elementos necesarios (*main*).
  - En el directorio *main* encontramos el código (*java*) y los recursos (*res*).



Estructura básica de una aplicación

- Directorio *java*:
  - Se pueden definir paquetes, clases auxiliares, etc. como en cualquier programa Java.
- Directorio *res*
  - Contiene todo aquello que no es código, pero que es necesario para el proyecto: imágenes, vídeos, etc.
  - Contiene una serie de subdirectorios para agrupar los distintos tipos de elementos.
  - Sólo existirán los subdirectorios que nuestro proyecto necesite

Estructura básica de una aplicación

- Algunos subdirectorios de *res* (I)
  - *drawable*: las imágenes de la aplicación.
  - *layout*: contiene los ficheros XML que definen la interfaz gráfica. Se definen carpetas distintas para distinguir la interfaz vertical y la horizontal:
    - *layout*
    - *layout-land*
  - *raw*: contiene ficheros externos con información que no esté en formato XML.

Estructura básica de una aplicación

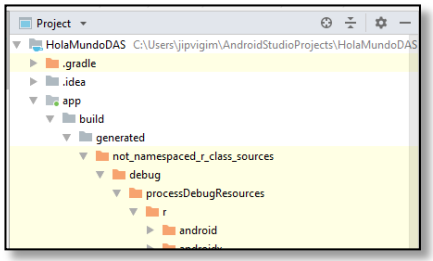
- Algunas subcarpetas de *res* (II)
  - *menu*: contiene la definición de los menús de la aplicación (en XML).
  - *values*: Contiene ficheros XML con información para la aplicación, como los *strings de texto* (diferentes idiomas), los *estilos*, etc.

Estructura básica de una aplicación

- Los subdirectorios de *res* se pueden "especializar" mediante el uso de sufijos
  - *Drawable-ldpi*: elementos gráficos para pantallas de baja densidad
  - *Drawable-mdpi*: elementos gráficos para pantallas de densidad media
  - *values-sw600dp*: elementos a utilizar en pantallas con un mínimo de 600 pixeles de anchura
  - *values-v4:* elementos a utilizar cuando la versión de la API sea 4 (Android 1.6) o superior
  - .....
- Más información sobre los recursos (res) en: http://developer.android.com/guide/topics/resources/providing-resources.html

Estructura básica de una aplicación

- La carpeta *libs*
  - Contiene las librerías auxiliares que se usen en nuestra aplicación.
- La carpeta *build*
  - Contiene ficheros generados automáticamente.
  - Son ficheros para el control de los recursos.
  - NO hay que modificarlos.



Estructura básica de una aplicación

- El fichero *AndroidManifest.xml*
  - Contiene la configuración de la aplicación en formato XML.
  - Se indican todos los elementos de una aplicación, y se crea (y se añaden elementos) automáticamente.
    - Nombre, versión , icono
    - Pantallas, mensajes
    - Permisos necesarios para la ejecución
    - ...

Estructura básica de una aplicación

- La carpeta *assets*
  - No existe por defecto, hay que crearla (botón derecho sobre el módulo)
    - New → Folder → Assets Folder
  - Permite almacenar recursos extra para la aplicación
    - Ficheros de configuración.
    - Ficheros de datos.
  - La diferencia entre ponerlos en *res/raw* o *assets* es:
    - Si están en *res/raw* → tendrán un ID, se podrá acceder a ellos por ID.
    - Si están en *assets* → Hay que acceder a ellos a través de su ruta (como un fichero "normal") y usando la clase *AssetManager*.

Si nos importa la estructura de almacenamiento de los ficheros, habrá que usar *assets*

Estructura básica de una aplicación

- Localización del fichero *.apk*:

  Directorio_proyecto/Directorio_módulo/build/outputs/apk/debug

  - Modulo-debug.apk

Estructura básica de una aplicación

- Ejercicio 1:
  - Localizar el fichero *.apk* generado en laboratorio 00.
  - Pasarlo al dispositivo real (vía Dropbox, correo, etc.) e instalarlo
    - Recordad activar la instalación desde otros orígenes
  - Visualizar qué recursos se han creado automáticamente en el directorio *res* (en la visualización Project)
    - ¿Qué significan los sufijos de los directorios *mipmap*? ¿Qué hay en ellos?
    - ¿Qué hay en el directorio *layout*?

Estructura básica de una aplicación

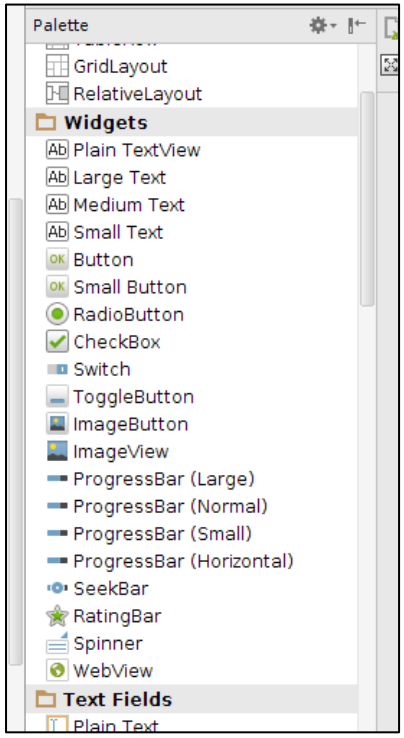
- Ejercicio 2:
  - Modificar la app, de tal forma que visualice en pantalla el texto "estrecho", si la anchura de la pantalla es w < 400 dp, y el texto "ancho", si es mayor.
    - El texto "Hello world!" a modificar está en el fichero activity_main.xml de la carpeta *layout*

Componentes de una aplicación

- **Activity**
  - Son el componente principal de las aplicaciones
  - Son "similares" a las ventanas de las aplicaciones tradicionales
  - Permiten la interacción de los usuarios
  - Se le asigna una interfaz gráfica
    - *setContentView (View)*
  - Toda actividad debería implementar
    - *onCreate(Bundle)* → Se ejecuta al crear la actividad. Normalmente el *setContentView* se ejecutará aquí

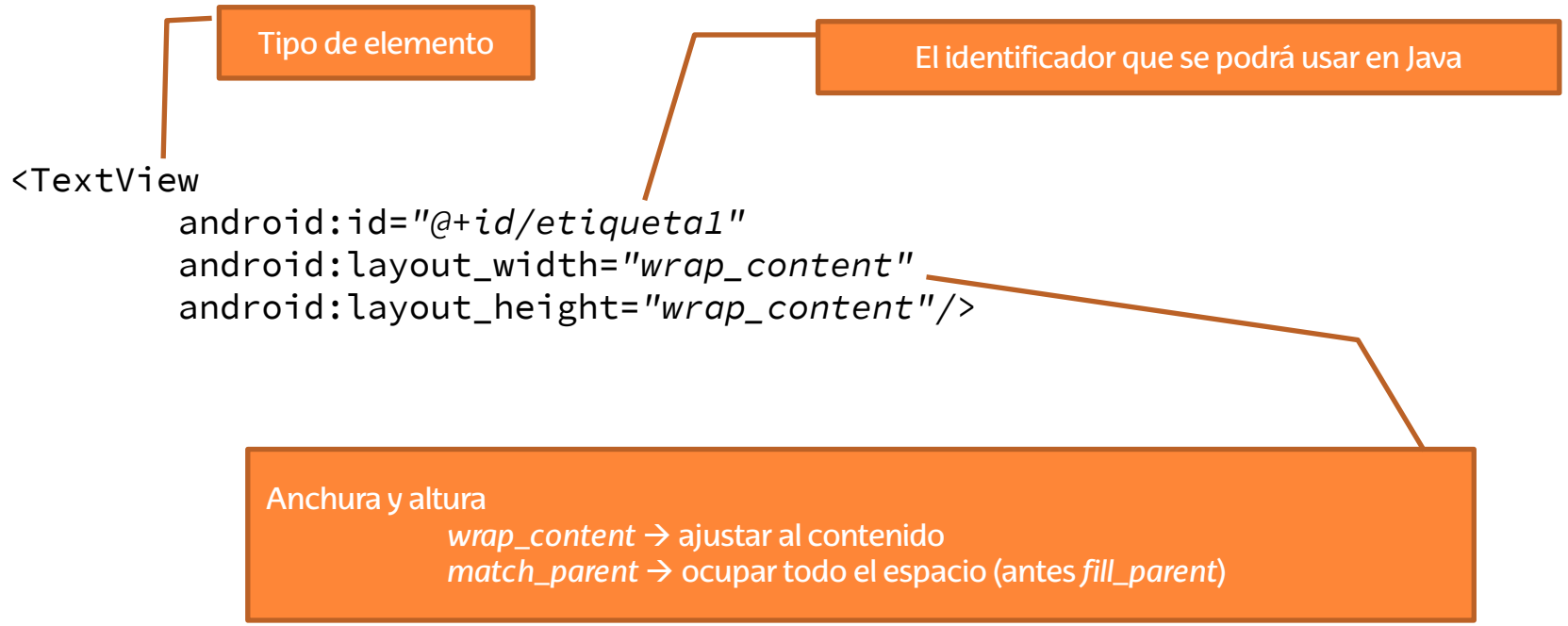
COMPONENTES DE UNA APLICACIÓN

- ***View***
  - Son los componentes básicos para construir la *interfaz gráfica* de la aplicación
  - Incluye cuadros de texto, botones, desplegables, etc.



Componentes de una aplicación

- La interfaz gráfica se define mediante *ficheros XML* en la carpeta *res/layout*



COMPONENTES DE UNA APLICACIÓN

- También se le puede asignar un tamaño predeterminado

píxeles de densidad (antes dip)

```
<TextView
        android:id="@+id/etiqueta1"
        android:layout_width="100dp"
        android:layout_height="100dp"/>
```

- Y un valor predeterminado

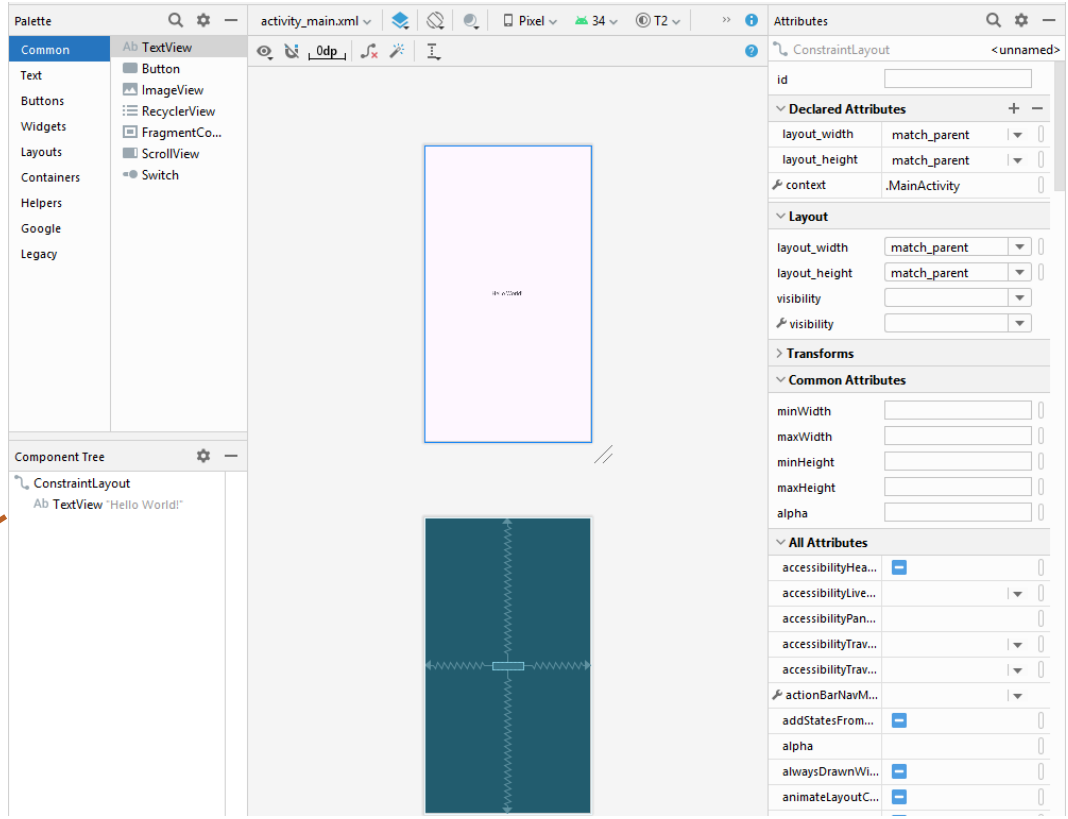
Se pueden usar dp, px, mm,y otras unidades

```
<TextView
        android:id="@+id/etiqueta1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text = "Este es el texto de la etiqueta"/>
```

Es mejor parametrizar los textos a través del fichero *strings.xml* en *res/values*

COMPONENTES DE UNA APLICACIÓN

- Android Studio permite diseñar la interfaz gráficamente.



Elementos gráficos

Propiedades

Jerarquía de componenetes

Vista gráfica: design y blueprint

COMPONENTES DE UNA APLICACIÓN

- Uso del fichero *strings.xml*

```
<resources>
    <string name="eltexto">Este es el texto de la etiqueta</string>
</resources>
```

El identificador

```
<TextView
        android:id="@+id/etiqueta1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text = "@string/eltexto"/>
```

El identificador

Componentes de una aplicación

- Para que la interfaz gráfica contenga más de un elemento de tipo *View*, hay que usar un elemento *Layout*
- Un *Layout* es un *contenedor* de elementos de tipo *View*
- Existen distintos tipos
  - LinearLayout
  - TableLayout
  - RelativeLayout
  - FrameLayout
  - ...

Componentes de una aplicación

- Definición de un layout

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/Tabla"
    android:layout_width="match_parent"
……
```

Es el mismo xmlns para todos los layouts

COMPONENTES DE UNA APLICACIÓN

- *LinearLayout*
  - Muestra sus elementos de forma secuencial (horizontal o vertical)

```
<LinearLayout  xmlns:android="http://...
        android:layout_height="match_parent"
        android:layout_width="match_parent"
        android:orientation ="vertical">
   <AnalogClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
   <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Un checkBox"/>
   <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Un botón"/>
   <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Un texto cualquiera"/>
</LinearLayout>
```

Indicar la orientación

COMPONENTES DE UNA APLICACIÓN

- *TableLayout*
  - Muestra los elementos en forma de tabla

```
<TableLayout xmlns:android=”http://...
      ……..
  <TableRow>
   <AnalogClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
   <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Un checkBox"/>
  </TableRow>
  <TableRow>
   <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Un botón"/>
   <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Un texto cualquiera"/>
  </TableRow>
</TableLayout>
```

Con TableRow creamos filas

COMPONENTES DE UNA APLICACIÓN

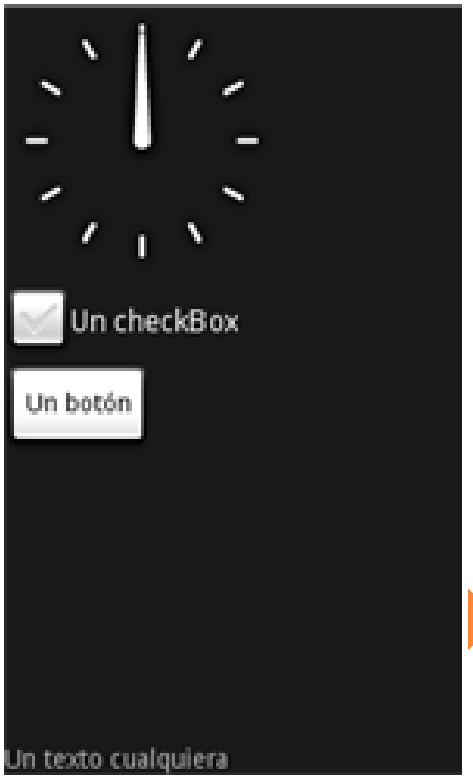
- *RelativeLayout*
  - Muestra los elementos en función de la posición de otros

```
<RelativeLayout
      ……….
  <AnalogClock
      android:id="@+id/AnalogClock01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"/>
  <CheckBox
      android:id="@+id/CheckBox01"
     ……
      android:layout_below="@+id/AnalogClock01"
      android:text="Un checkBox"/>
  <Button
      android:id="@+id/Button01"
    ……
      android:text="Un botón"
      android:layout_below="@+id/CheckBox01"/>
  <TextView
      android:id="@+id/TextView01"
      ……
      android:layout_alignParentBottom="true"
      android:text="Un texto cualquiera"/>
</RelativeLayout>
```

Indicamos la posición respecto al padre (layout)

Debajo del reloj

En el fondo



COMPONENTES DE UNA APLICACIÓN

- *FrameLayout*
  - Muestra los elementos uno encima de otro (visibilidad)

```
<FrameLayout
……
   <AnalogClock
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
   <CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un checkBox"/>
   <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un botón"
    android:visibility="invisible"/>
   <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un texto cualquiera"
    android:visibility="invisible"/>
</FrameLayout>
```

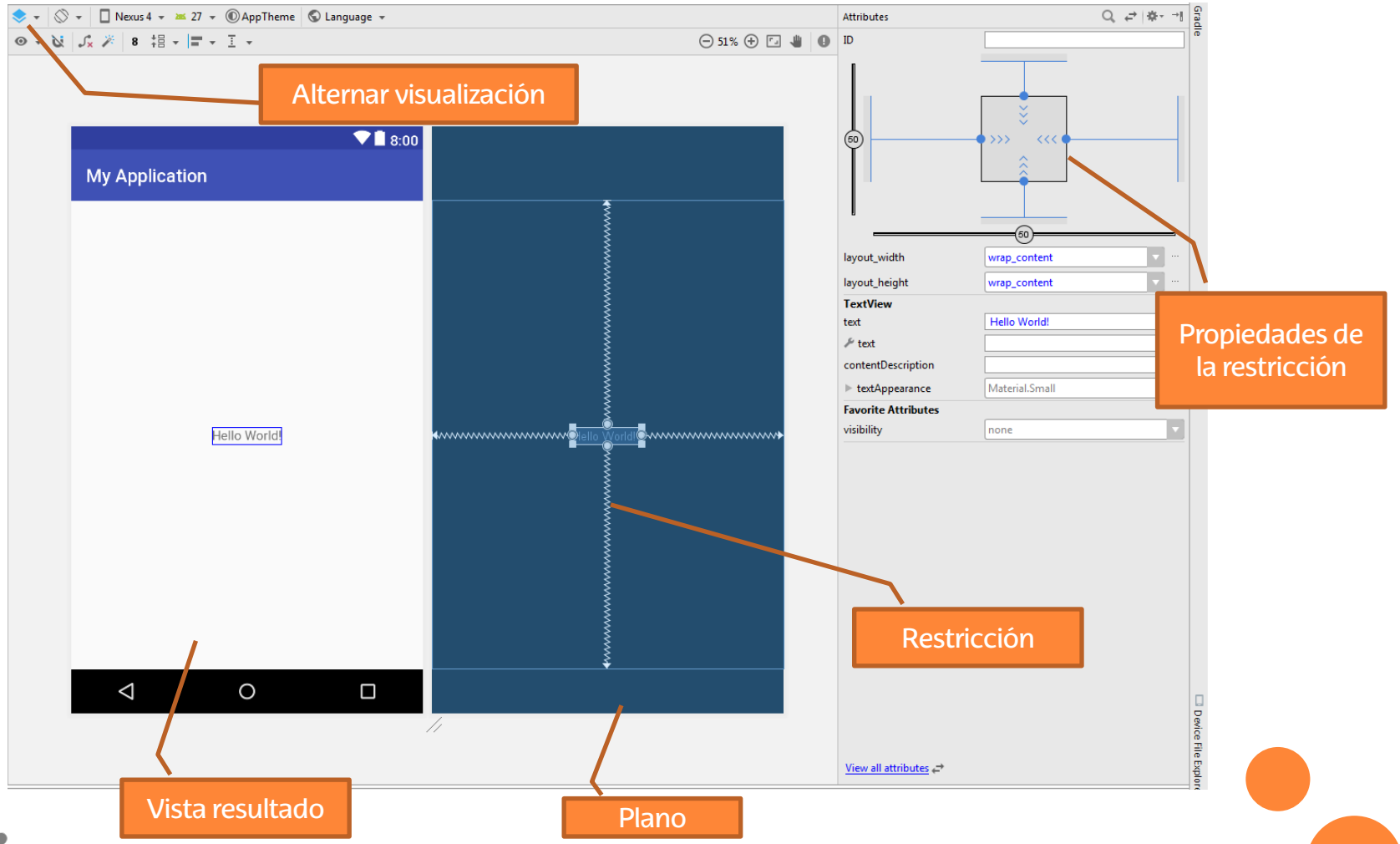
Invisible

Invisible

CONSTRAINT LAYOUT

- Layout por defecto desde Android Studio 2.2
- Permite definir interfaces complejas en base a restricciones
  - Necesita menos memoria en comparación a la combinación de distintos layouts para conseguir el mismo aspecto
- Pensado para trabajar principalmente con el editor gráfico

CONSTRAINT LAYOUT



CONSTRAINT LAYOUT

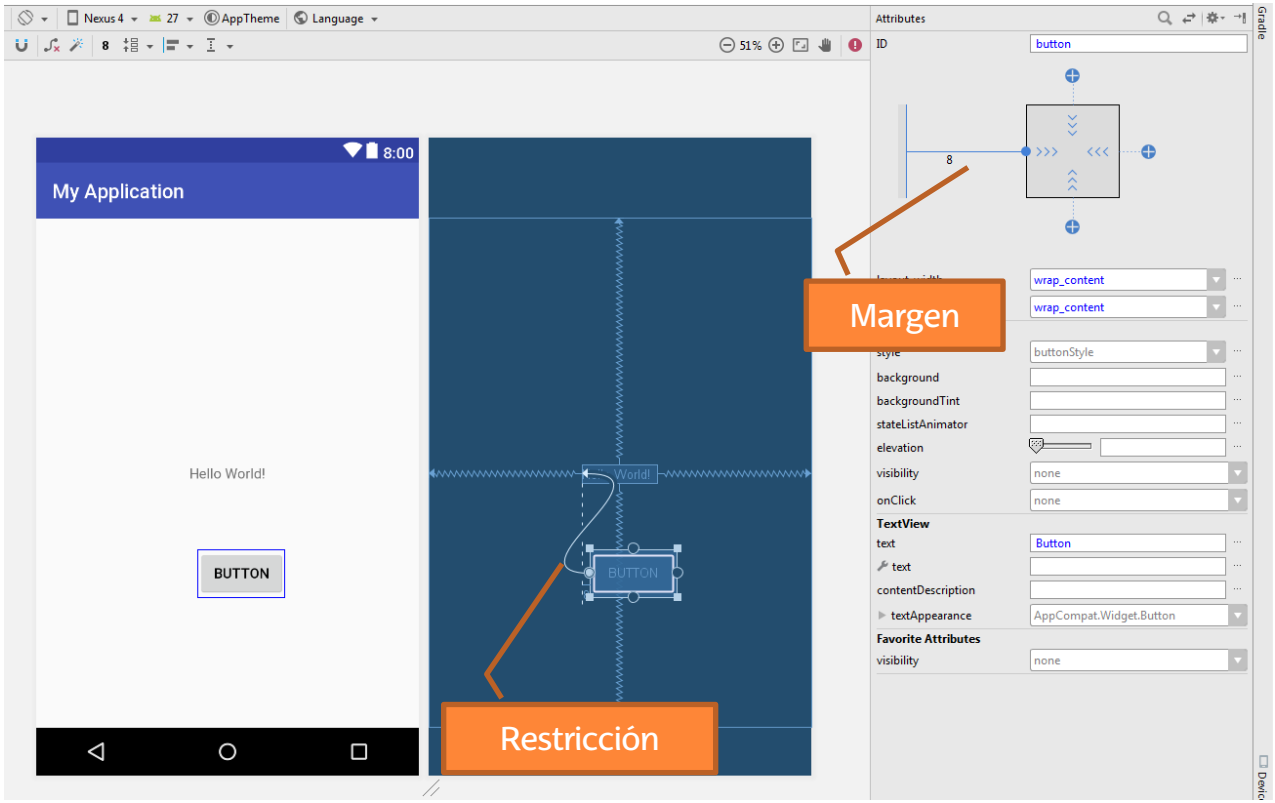
- En el XML

```
<TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```

Restricciones

CONSTRAINT LAYOUT

- Al añadir elementos se definen nuevas restricciones uniendo los puntos



CONSTRAINT LAYOUT

- En el XML

```
<Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:text="Button"
        app:layout_constraintStart_toStartOf="@+id/textView"
        tools:layout_editor_absoluteY="332dp" />
```

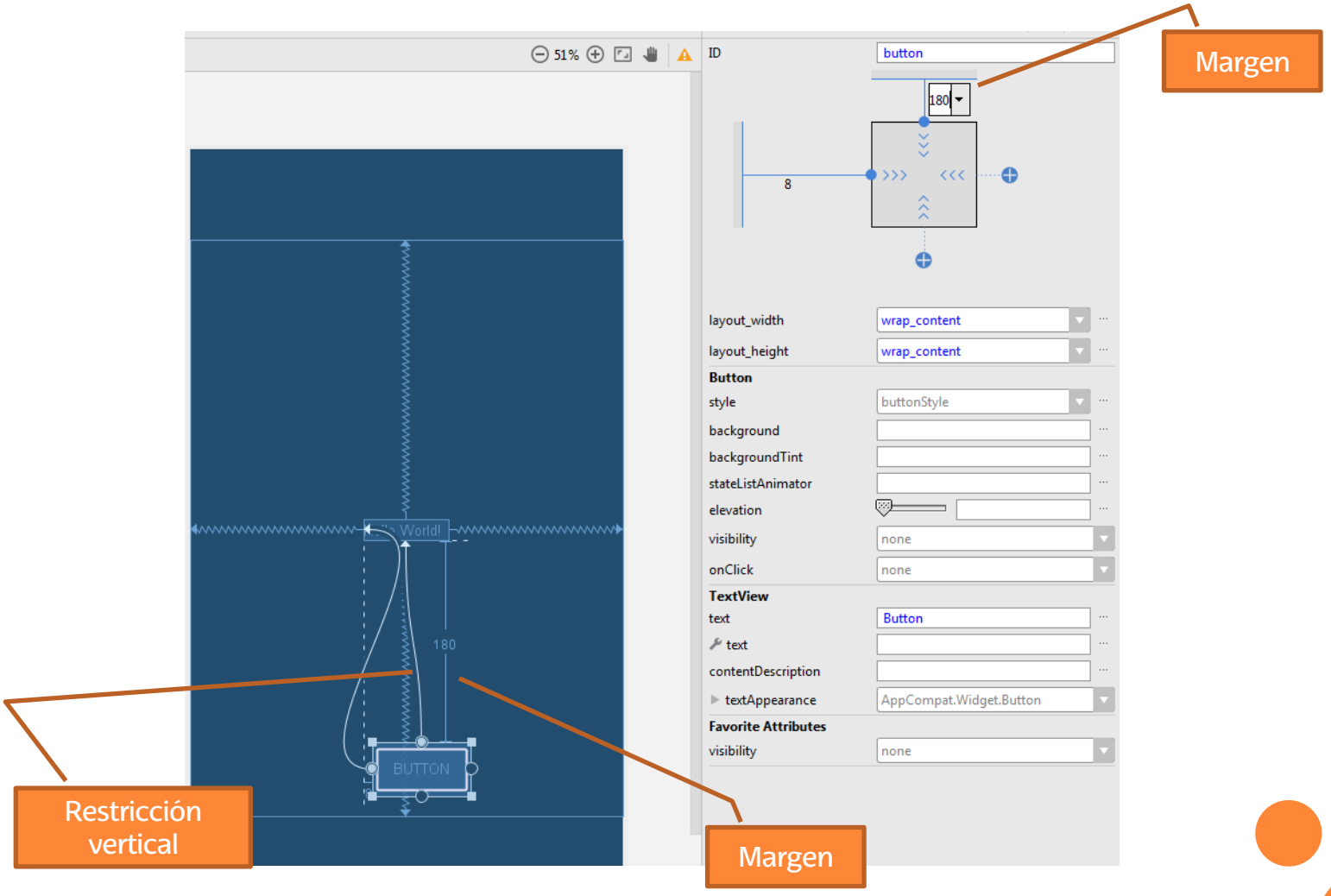
Margen

Restricción

¿Y el posicionamiento vertical?

CONSTRAINT LAYOUT

CONSTRAINT LAYOUT



CONSTRAINT LAYOUT

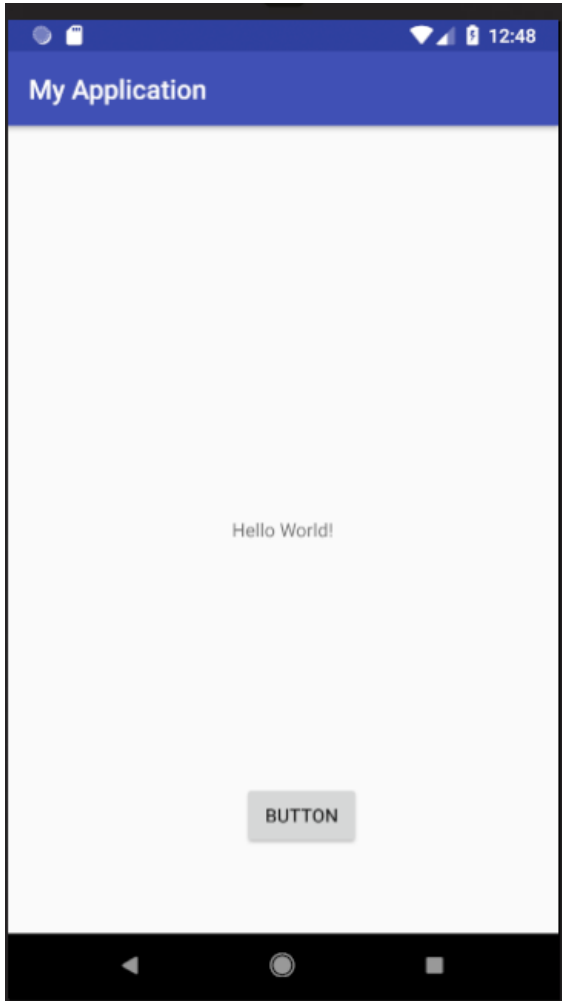
- En el XML

```
<<Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="180dp"
        android:text="Button"
        app:layout_constraintStart_toStartOf="@+id/textView"
        app:layout_constraintTop_toBottomOf="@+id/textView" />
```

Margen vertical

Restricción vertical

CONSTRAINT LAYOUT



COMPONENTES DE UNA APLICACIÓN

- Para acceder desde la actividad a los elementos de tipo *View*, se usa la clase *R* de la carpeta *gen*:
  - Para establecer la vista (desde java):
    - *setContentView(R.layout.nombredelfichero.xml);*
  - Para acceder a los controles (desde java):
    - Se usan las clases contenidas en *android.widget*
    - *Button miButton = (Button) findViewById(R.id.elboton);*

Clase Button

Siempre hay que hacer el casting

Identificador definido en el XML

CONSTRAINT LAYOUT

Cuidado con los valores numéricos
dp vs px

- Se puede trabajar con las interfaces de manera dinámica

```
ConstraintLayout cl=(ConstraintLayout) findViewById(R.id.constxml) ;

        TextView etq= new TextView (this);
        Button boton= (Button) findViewById(R.id.btn);

        ConstraintSet cs = new ConstraintSet();
        cs.clone(cl);

        cs.centerHorizontally(etq.getId(),cl.getId());
        cs.centerVertically(etq.getId(),cl.getId());
        cs.connect(boton.getId(), ConstraintSet.START,
etq.getId(),ConstraintSet.START,8);
        cs.connect(boton.getId(),ConstraintSet.TOP,etq.getId(),
ConstraintSet.BOTTOM,180);
        cs.applyTo(cl);
```

Nueva etiqueta

Botón existente

Coger las restricciones existentes

Centrado

Restricción lateral

Restricción vertical

Aplicar

Componentes de una aplicación

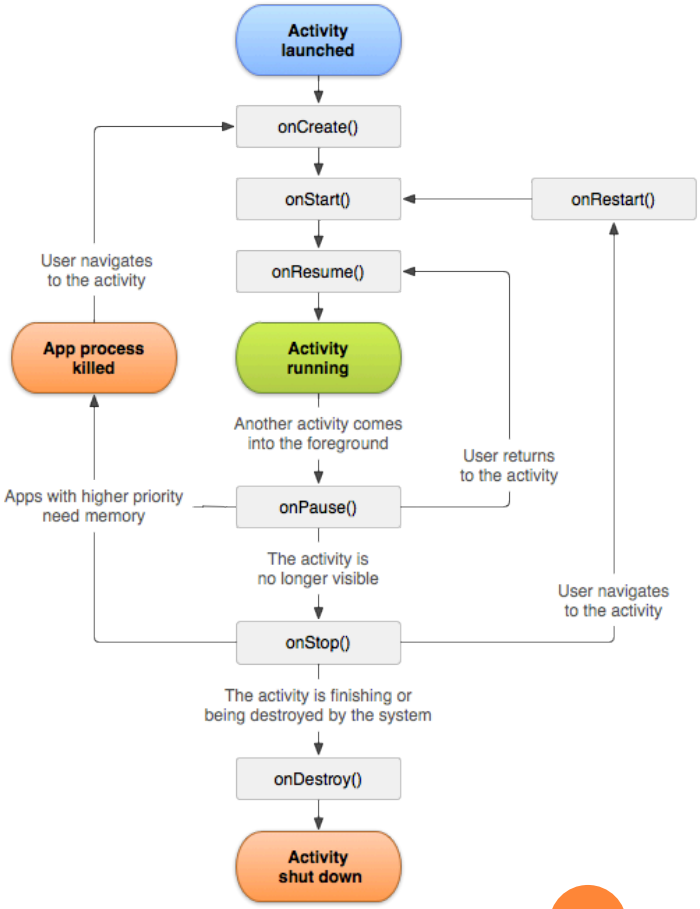
- Ejercicio 3: Cread una *app* con varios *TextViews* (cada uno con su correspondiente texto) mediante la interfaz gráfica.
  - Mirad cómo queda el layout en modo texto
  - Quitad el texto del fichero XML del layout y usad el fichero *strings.xml*.

CONSTRAINT LAYOUT

- Ejercicio 4: Cread una app cuya interfaz tenga:
  - Una caja de texto que ocupe toda la interfaz a lo ancho
  - Un botón, centrado y debajo de la imagen
  - Una etiqueta de texto debajo del botón, que aparecerá rotada en diagonal tras pulsar el botón.

Componentes de una aplicación

- Ciclo de vida de las actividades
  - Hay que tenerlo en cuenta
    - ¿Llamadas?
    - Otros procesos
    - Falta de memoria
  - El botón "back"
    - Destruye la actividad
  - El botón "home"
    - La pone en pausa
    - La pasa a Stop



Extraído de http://developer.android.com/reference/android/app/Activity.html

Componentes de una aplicación

- Al alcanzar un estado, la actividad lanza el método correspondiente
  - *onCreate (Bundle)*
  - *onStart()*
  - *onResume()*
  - *onRestart()*
  - *onPause()*
  - *onStop()*
  - *onDestroy()*

En todos ellos lo primero que hay que hacer es llamar al mismo método de la clase padre *super.on....*

Componentes de una aplicación

- Al alcanzar un estado, la actividad lanza el método correspondiente
  - *onCreate(Bundle)*
    - Primer método que se llama al crear una actividad.
    - Sirve para inicializar los elementos gráficos de la Interfaz de Usuario (UI).
    - El objeto *Bundle* contiene los elementos de estado guardados previamente mediante *savedInstanceState*, que se pueden usar para recrear dicho estado en la UI.
  - *onStart()*
    - Se llama justo antes de presentar la actividad en pantalla.
    - Se suele usar para inicializar contenidos de la actividad:
      - Animaciones, contenidos de audio, ...

Componentes de una aplicación

- Al alcanzar un estado, la actividad lanza el método correspondiente
  - *onResume()*
    - Se ejecuta cuando se trae una actividad a primer plano.
    - Es un buen sitio para actualizar elementos de UI:
      - Reiniciar animaciones, una reproducción de audio/video, o inicializar cualquier componente que se hubiera liberado durante *onPause()*.
  - *onRestart()*
    - Si paramos una actividad y volvemos a inicializarla, se llama a este método.
    - Siempre le sigue el método *onStart()*.

Componentes de una aplicación

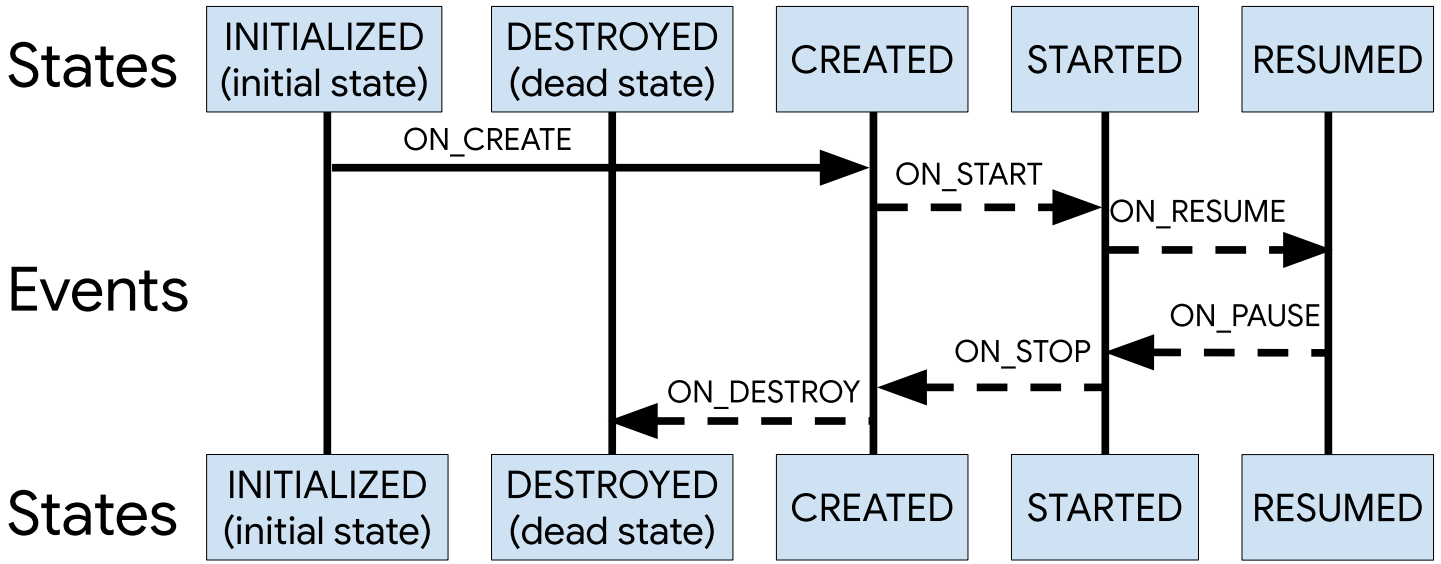
- Al alcanzar un estado, la actividad lanza el método correspondiente
  - *onPause()*
    - Se llama antes de que la aplicación se envíe a un segundo plano.
    - Aquí se suelen parar las animaciones o audios/videos asociados con los elementos de UI.
    - Si la aplicación vuelve al primer plano, le sigue el método *onResume()*; si no, *onStop()*.
  - *onStop()*
    - Se llama justo después de que la aplicación pase a un segundo plano.
    - Es un buen sitio para guardar datos en disco.

Componentes de una aplicación

- Al alcanzar un estado, la actividad lanza el método correspondiente
  - *onDestroy()*
    - Es el último método que se llama antes de que la actividad se destruya completamente.
    - Si hay hilos en background u otros recursos de larga duración, se suelen parar en este método.
    - Una forma de destruir la actividad es llamar a *finish()*, pero el sistema operativo puede hacerlo llamar también si hay un problema de memoria (por ejemplo).

COMPONENTES DE UNA APLICACIÓN

- Diagrama de estados y ciclo de vida



- Extraído de: https://developer.android.com/topic/libraries/architecture/lifecycle

Componentes de una aplicación

- Ejercicio 5:
  - Cread una app donde la actividad principal tenga los métodos de ciclo de vida sobreescritos:
    - *onCreate(Bundle)*, *onStart()*, *onResume()* etc.,
  - Cada método debe escribir su nombre en el log.
    - Utilizar el comando Log.X();
    - *Ver siguiente diapositiva para más información.*
  - Jugad con la *app* y mirad por qué estados pasa la actividad.
    - Girad el móvil, dadle al "back", al "home", etc.

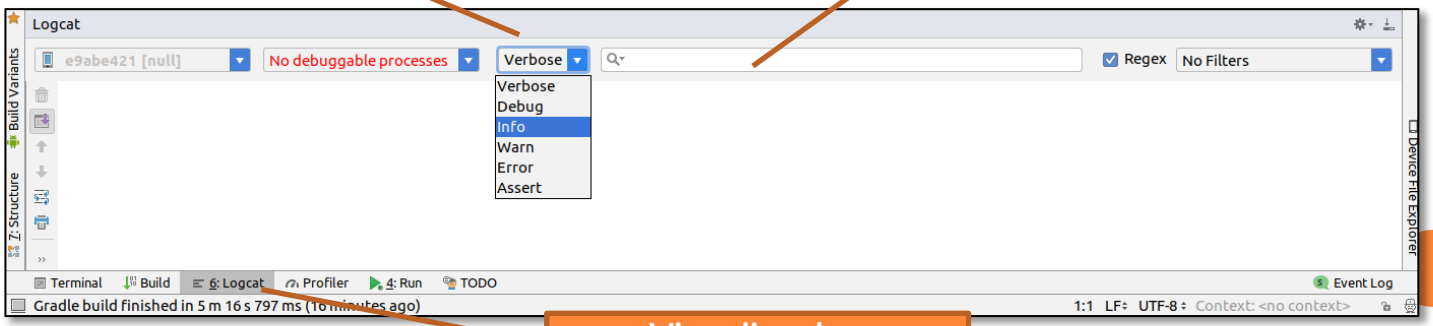
COMPONENTES DE UNA APLICACIÓN

- Ejercicio 5:
  - Para enviar un mensaje al Log de la aplicación: Log
    - Ref.: https://developer.android.com/reference/android/util/Log
  - El Log se visualiza en LogCat
    - Parte inferior de Android Studio
  - Ejemplo: Log.i(*"etiqueta"*, "Paso por onCreate");

Tipo de mensaje:
i (info), e (error), w (warn), etc.

Etiqueta . Útil para hacer búsquedas y filtrados

Texto a escribir



Visualizar log

Componentes de una aplicación

- Android finaliza procesos y actividades cuando necesita liberar RAM
- La probabilidad de que se finalice una actividad depende su estado

Probabilidad de que finalice	Estado del proceso	Estado de la actividad
Menor	Primer plano (en foco o por estar en él)	Created Started Resumed
Más	Segundo plano (foco perdido)	Paused
Mayor	Segundo plano (no visible)	Stopped
	Vacío	Destroyed

Extraído de https://developer.android.com/guide/components/activities/activity-lifecycle

Componentes de una aplicación

- Se puede controlar cuándo el sistema operativo mata una actividad:
  - Por temas de memoria (en los modos *onPause* y *onStop*)
  - Por reorientación (horizontal / vertical)
  - Conviene almacenar datos de la interfaz
    - Datos de un formulario
    - Momento de ejecución de un vídeo
    - ...

Componentes de una aplicación

- *onSaveInstanceState (Bundle)*
  - Se ejecuta antes de destruir la actividad
  - Android llama onSaveInstanceState si:
    - Si es el usuario el que reinicia la actividad, el objeto de la clase Bundle también mantendrá lo guardado.
    - Si Android destruye la actividad por un cambio de configuración o por necesidad de recursos.
  - Si la app se cierra por un fallo de sistema o por una llamada a finish(), no pasa por onSaveinstanceState.
- El *Bundle* almacenado se pasa como parámetro al método *onCreate(Bundle)*
- Se puede usar *onRestoreInstanceState (Bundle)*
  - Se ejecuta después de *onStart()*

COMPONENTES DE UNA APLICACIÓN

- En el Bundle se almacenan pares (nombre,valor)

```
protected void onSaveInstanceState(Bundle savedInstanceState) {
            super.onSaveInstanceState(savedInstanceState);
             savedInstanceState.putString(“variable1", valor1);
             savedInstanceState.putInt(“variable2", valor2);
      }

protected void onRestoreInstanceState(Bundle savedInstanceState) {
            super.onRestoreInstanceState(savedInstanceState);
            val1 = savedInstanceState.getString("variable1");
            var2 = savedInstanceState.getInt(“variable2");
      }
```

Extraído de https://developer.android.com/guide/components/activities/activity-lifecycle

COMPONENTES DE UNA APLICACIÓN

- Si queremos restaurar el estado en el método *onCreate()*
  - Hay que comprobar si tiene algún valor almacenado o no

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (savedInstanceState!= null)
                { valor= savedInstanceState.getInt(“variable");
        }
}
```