INDICE

1.1. Introducción

- Un **Layout** es un elemento de la interfaz de usuario (UI).
- En él podemos definir los elementos visuales que componen la pantalla.
- Se puede configurar en archivos **XML** o código **Java** en tiempo de ejecución. Ya hemos visto en la sección anterior las ventajas de los archivos XML.
- Cada archivo XML asociado con una pantalla / layout debe contener un **elemento raíz** y dentro de este se pueden agregar más layouts y objetos secundarios para construir una jerarquía de Vistas (Views) que definirá la pantalla/layout.
- En el siguiente ejemplo, la línea 2 indica el inicio del elemento raíz que cierra en la línea 15.

```
1 <? xml version = "1.0" encoding = "utf-8"?>
2 <LinearLayout  xmlns: android = "http://schemas.android.com/apk/res/android"
3               android: layout_width = "fill_parent"
4               android: layout_height = "fill_parent"
5               android: orientación = "vertical"  >
6     <TextView  android: id = "@ + id / text"
7               android: layout_width = "wrap_content"
8               android: layout_height = "wrap_content"
9               android: text = "Soy un TextView"  />
10    <Botón de  android: id = "@ + id / button"
11            android: layout_width = "wrap_content"
12            android: layout_height = "wrap_content"
13            android: text = "Soy un botón" />
14 />
15 </LinearLayout>
```

- Nota: Indicar que en estos apartados se pueden ver ejemplos utilizando la unidad pixel (*px*). Recordar que debe emplear la unidad *dp's* como ya vimos en el tema 2.1.

1.2. Cargar recurso XML: layout

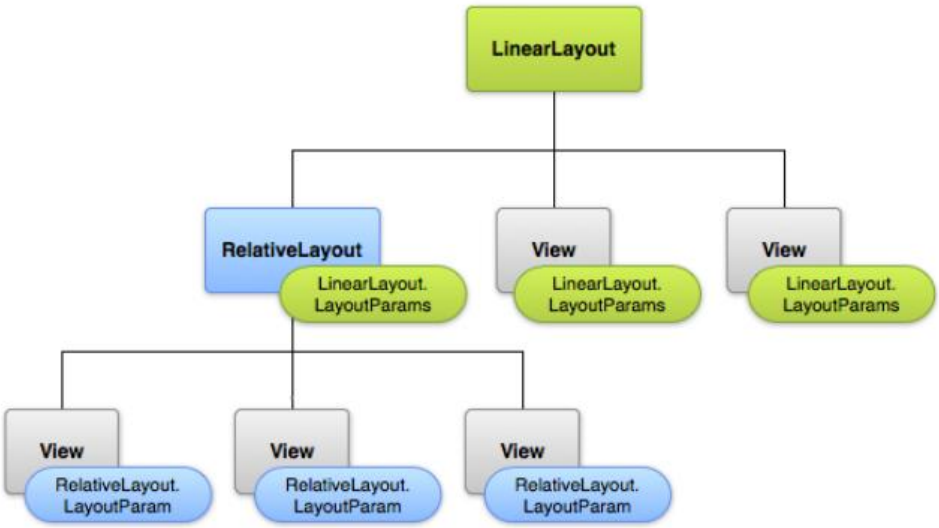
- Como sabemos, cada vez que se compila, cada fichero layout XML se compila a un objeto Vista accesible a través de Java mediante la clase R.
- El layout se cargará cuando se llame al método **onCreate()** de la Activity llamando al método **setContentView()** al que hace referencia el recurso de layout a través de la clase R y del nombre del layout (el nombre del fichero xml, en este caso: *main_layout.xml*)

```
1 public void onCreate(Bundle savedInstanceState) {
2     super.onCreate(savedInstanceState);
3     setContentView(R.layout.main_layout);
4 }
```

- Esto cargará en pantalla el layout con todos los elementos visuales que contiene.
- Un layout tiene **atributos** como cualquier View, como se indica en la sección anterior.
- Un layout extiende la clase ViewGroup como se vio en el apartado anterior.
- **Referencias:**
  - ✓ Layouts: https://developer.android.com/guide/topics/ui/declaring-layout.html
  - ✓ Parámetros del layout: https://developer.android.com/reference/android/view/ViewGroup.LayoutParams.html

1.3. Parámetros de un layout

- Los atributos xml de un layout, que comienzan llamándose **layout_***algo*, definen los atributos que son apropiados para cada ViewGroup donde se encuentra a Vista.



- La imagen muestra cómo los hijos de Views heredan los **LayoutParams** de los contenedores, de los padres.
- Cada vista secundaria debe establecer sus parámetros apropiados en función de su padre, aunque puede establecer parámetros para sus hijos.
- Cada hijo contiene propiedades de tipo que definen su tamaño y posición:

1.3.1.Tamaño

- Definir valores para los atributos: **layout_width** y **layout_height**.
  - ✓ Normalmente se utilizarán estos dos valores:
    - **wrap_content**: ajusta el tamaño al contenido.
    - **match_parent**: aumenta de tamaño al tamaño del contenedor principal.
  - ✓ Otros posibles tipos de valores son:
    - **px**, **dp**, **sp**, **in**, **mm**, **pt**.
    - Las unidades px, dp y sp ya se han visto en un apartado anterior.
- En Java podemos capturar el tamaño de una Vista con:
  - ✓ Para saber el **tamaño deseado** de la View, tenemos **getMeasuredWidth ()** y **getMeasuredHeight ()**.
  - ✓ Para conocer el ancho y alto real de la Vista usaremos **getWidth ()** y **getHeight ()**.

1.3.2.Posición

- Las views son un rectángulo geométricamente hablando.
- Una views se define por:
  - ✓ Una **localización** : expresada por el par de coordenadas: izquierda y arriba
  - ✓ Una **dimensión** : expresada como ancho y alto (ancho y alto)
  - ✓ La unidad para medir la ubicación y el tamaño es el píxel (px).
- En Java para obtener la localización tenemos:
  - ✓ **getLeft()** e **getTop()**
  - ✓ **getRight()** e **getBottom()**
  - ✓ Devuelven la X y la Y relativa a su padre.

1.3.3.Relleno. Padding

- El **Padding** es el relleno/espacio que hace que el contenido se vea con respecto a sus lados.
- El tamaño de **Padding** se expresa en cualquiera de las unidades visualizadas anteriormente (px, pt, dp, sp, mm, in) para los lados izquierdo, superior, derecho e inferior (left, top, right, bottom) de la vista.
  - ✓ Por ejemplo, un padding de 2px en el lado izquierdo indica que el contenido de la vista comenzará 2px a la derecha del borde izquierdo.
  - ✓ El tamaño de la vista lo es todo: más contenido llena el espacio.

- El tamaño del padding de una vista se puede expresar en **XML** como:
  - ✓ android: padding (en este caso, el mismo relleno para los cuatro lados)
  - ✓ android: paddingLeft
  - ✓ android: paddingTop
  - ✓ android: paddingRight
  - ✓ android: paddingBottom
- El tamaño de padding de una view se puede expresar en **Java** con:
  - ✓ setPadding (int, int, int, int)
- El tamaño de relleno de una vista se puede capturar en Java con:
  - ✓ getPaddingLeft(), getPaddingTop(), getPaddingRight() y getPaddingBottom()

1.3.4.Márgenes

- android: layout_margin
- android: layout_marginBottom
- android: layout_marginTop
- android: layout_marginLeft
- android: layout_marginRight

- Nota: A partir de API 17 (Android 4.2), admite tener activities en formato RTL (Right to Left).

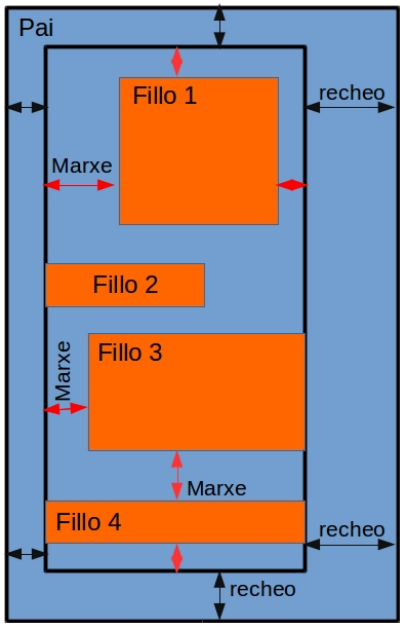
  En este formato, el texto comienza a leerse por la derecha (a diferencia del formato LTR que es el que usamos).

  Por eso aparecen las propiedades:

  - ✓ android: layout_marginEnd
  - ✓ android: layout_marginStart

Es mejor usar estas opciones en lugar de android: *layout_marginRight(=android:layout_marginEnd)* y *android:layout_marginLeft(= android:layout_marginStart)*

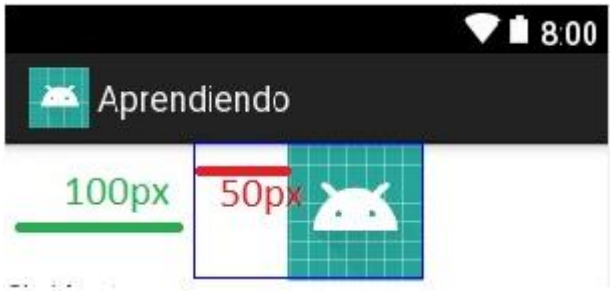
Explicación Gráfica



- La vista **PADRE** tiene un relleno diferente definido para cada lado: los elementos contenidos en el PAI estarán separados del borde principal por la distancia definida en el relleno.
- Cada vista hijo tiene definido un margen (o varios) con respecto a los límites establecidos por el PADRE.
  - ✓ Hijo 1 : A la izquierda y arriba, comienza donde termina el relleno de PAI, pero en la parte superior este elemento tiene sus propios márgenes definidos para esos lados.
  - ✓ Hijo 2 : No tiene margen izquierdo definido.
  - ✓ …
- Por ejemplo, en este caso tenemos el siguiente código layout:

```
1    <ImageView
2        android:id="@+id/imageView"
3        android:layout_width="wrap_content"
4        android:layout_height="wrap_content"
5        android:paddingLeft="50px"
6        android:layout_marginLeft="100px"
7        android:src="@mipmap/ic_launcher" />
```

Que dará este resultado:



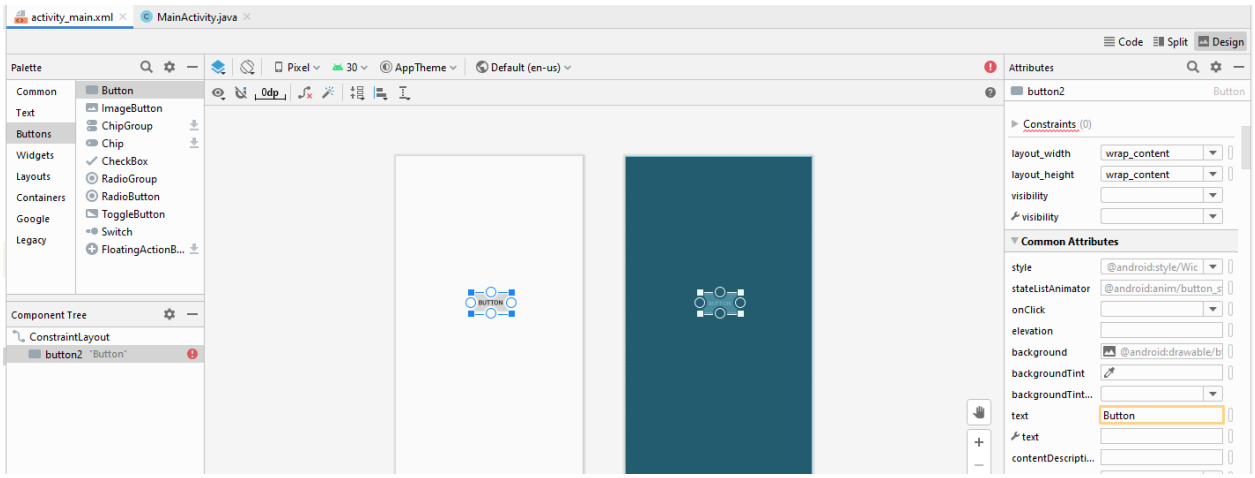
1.4. Construcción de un layout

- Un diseño se puede construir de 2 formas:
  - ✓ **Archivo xml**
  - ✓ **Con un Adapter** : cuando se desconoce el contenido del layout o su contenido es dinámico podemos usar la clase de Java **AdapterView** que en tiempo de ejecución creará el layout y le agregará las vistas indicadas.
- Veremos en las siguientes secciones cómo se construyen los layouts.
- Actualmente, el diseño recomendado para hacer diseños es **ConstraintLayout** .

  Outros son FrameLayout, LinearLayout, RelativeLayout.

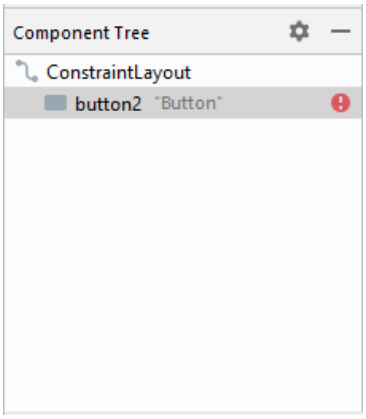
- Cada uno de ellos ordena los elementos visuales en su interior de diferentes formas.
- Cada uno de ellos puede contener diseños del mismo tipo o de un tipo diferente, etc.

1.5. Aspectos generales de los layouts en Android Studio



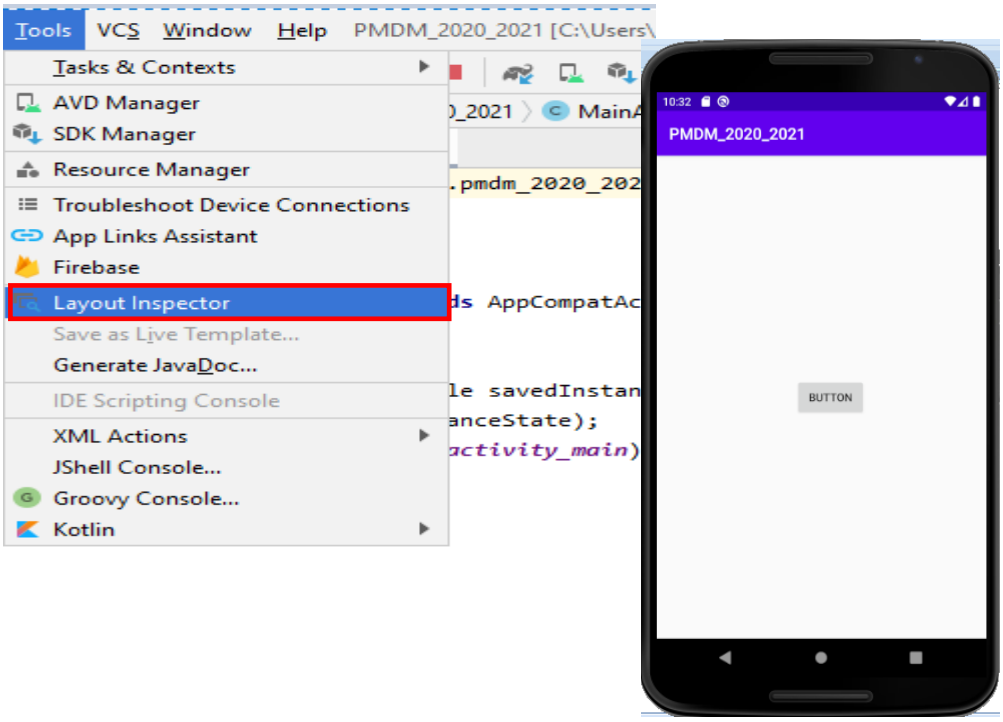
Jerarquía de componentes

- Podemos visualizar a jerarquía dos diferentes views que añadimos en la ventana **Component tree**:

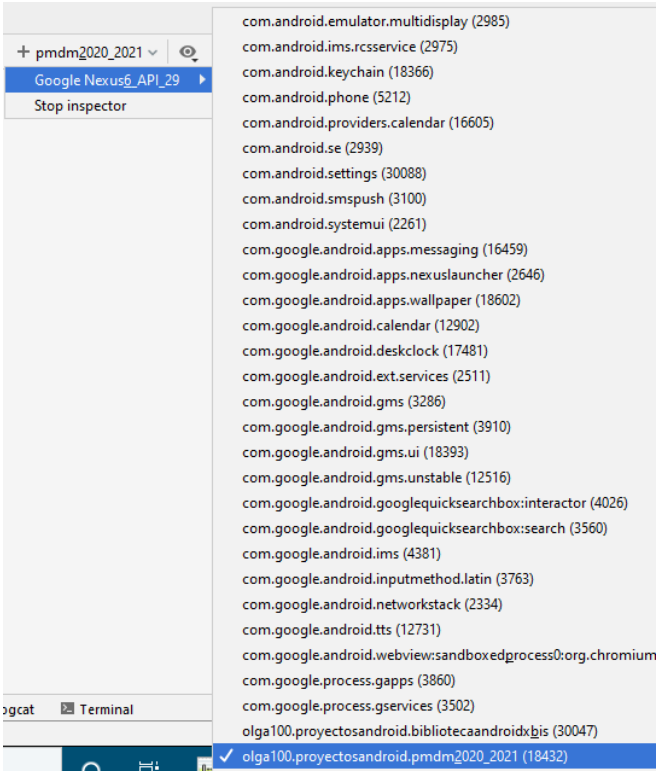


Con la aplicación **ejecutándose** en el emulador, podemos acceder a la misma jerarquía, pero de una manera más completa, permitiendo acceder a todas las propiedades de todos los componentes gráficos de la aplicación:

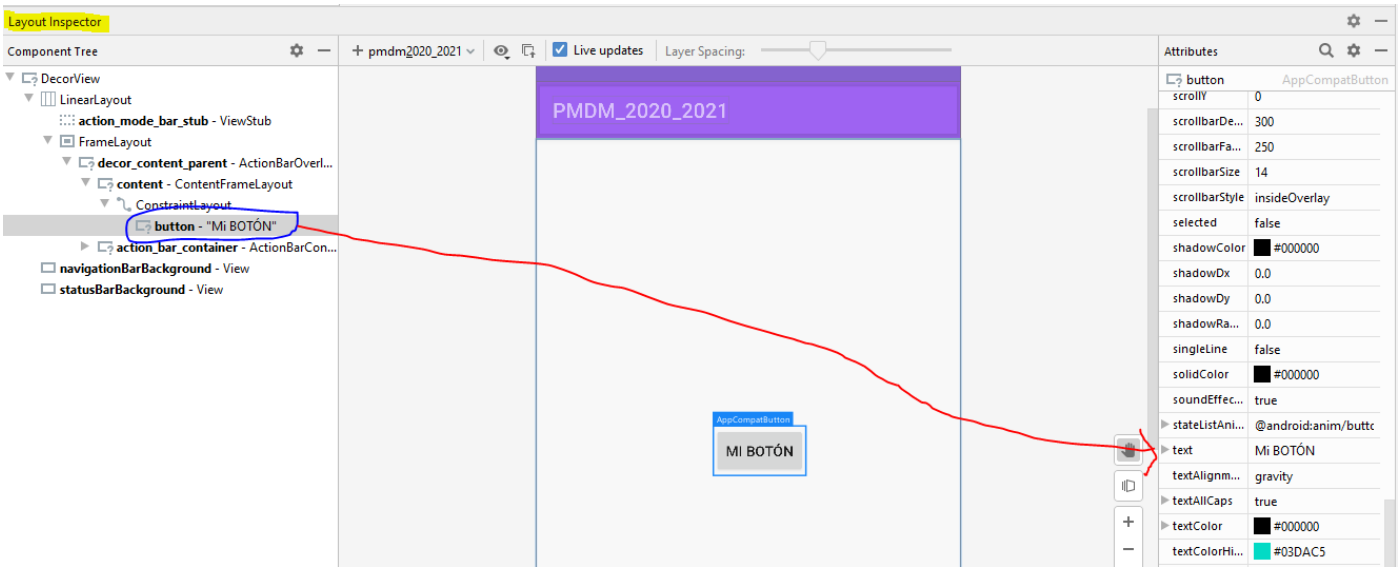
Layout Inspector



Con la aplicación en funcionamento, accedemos a opción de menú **Tools => Layout inspector**.

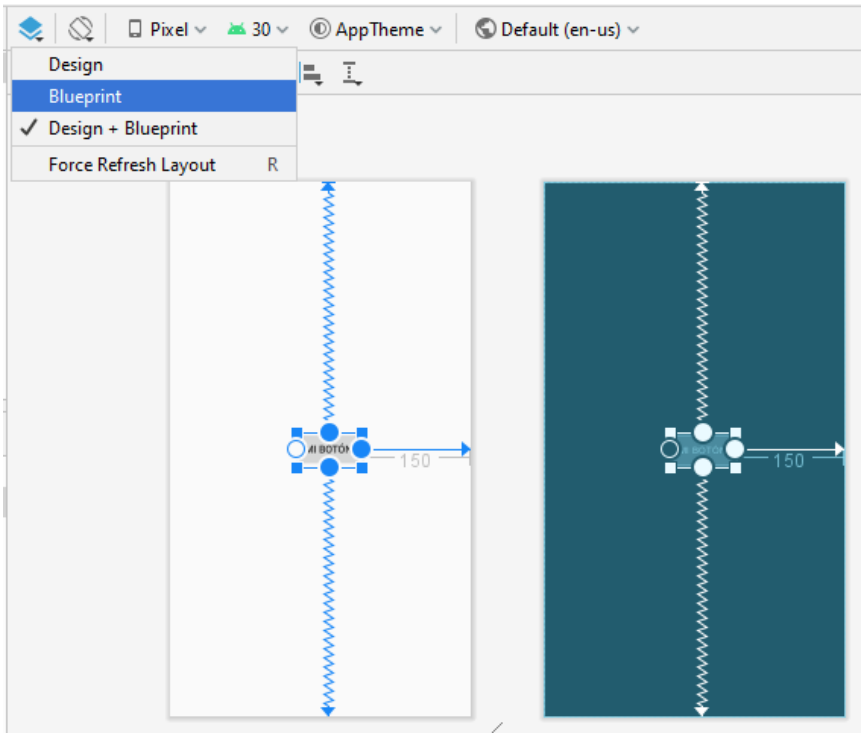


Seleccionamos la aplicación que queremos inspeccionar.



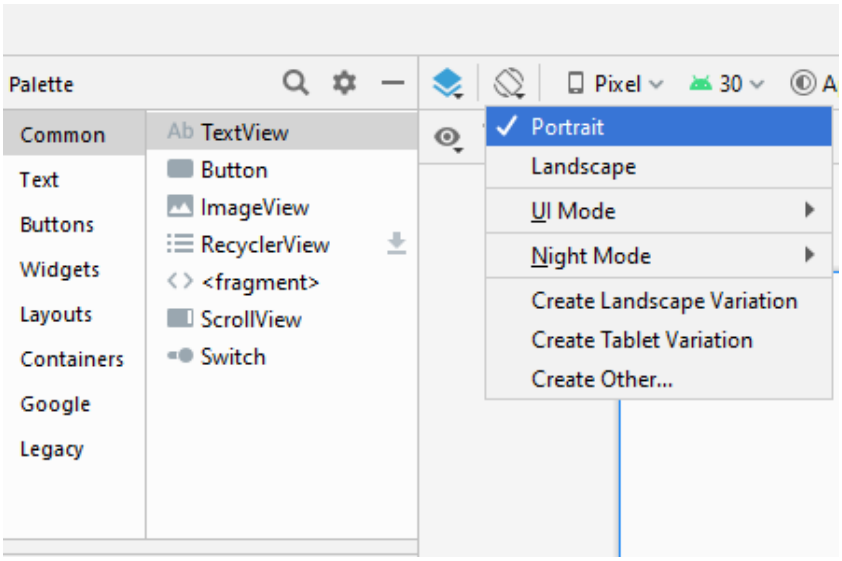
Tenemos acceso a una vista en árbol de todos los componentes gráficos y sus propiedades en ejecución

Modo Diseño



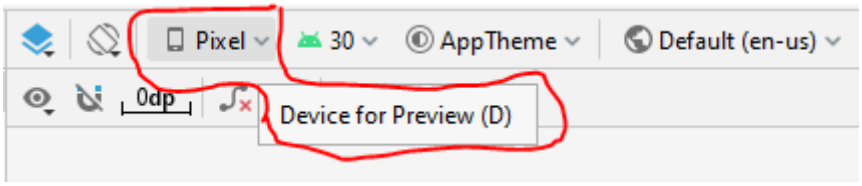
- Podemos ver el diseño de nuestra activity de dos formas distintas:
  - ✓ *Design*: De la forma tradicional, se ve gráficamente cómo se ven los componentes.
  - ✓ *Blueprint*: Se ve el esqueleto del diseño. Las view muestran sus límites, pero no el contenido gráfico.

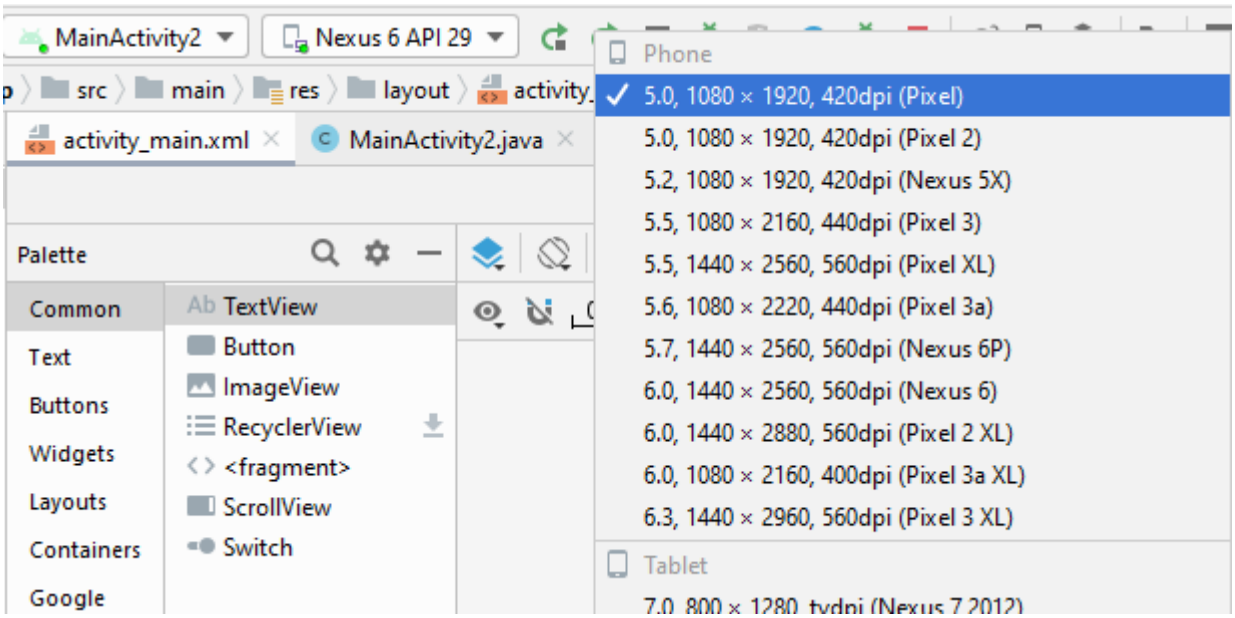
Modo Orientación



- Podemos indicar como está orientado el dispositivo, o activar el modo noche y una serie de variantes.

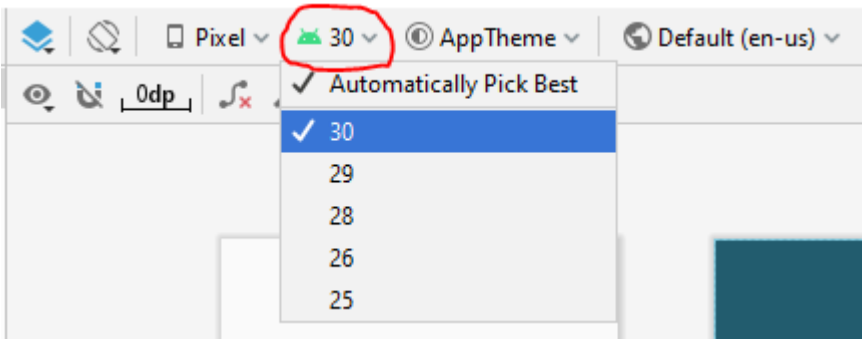
Modo Device





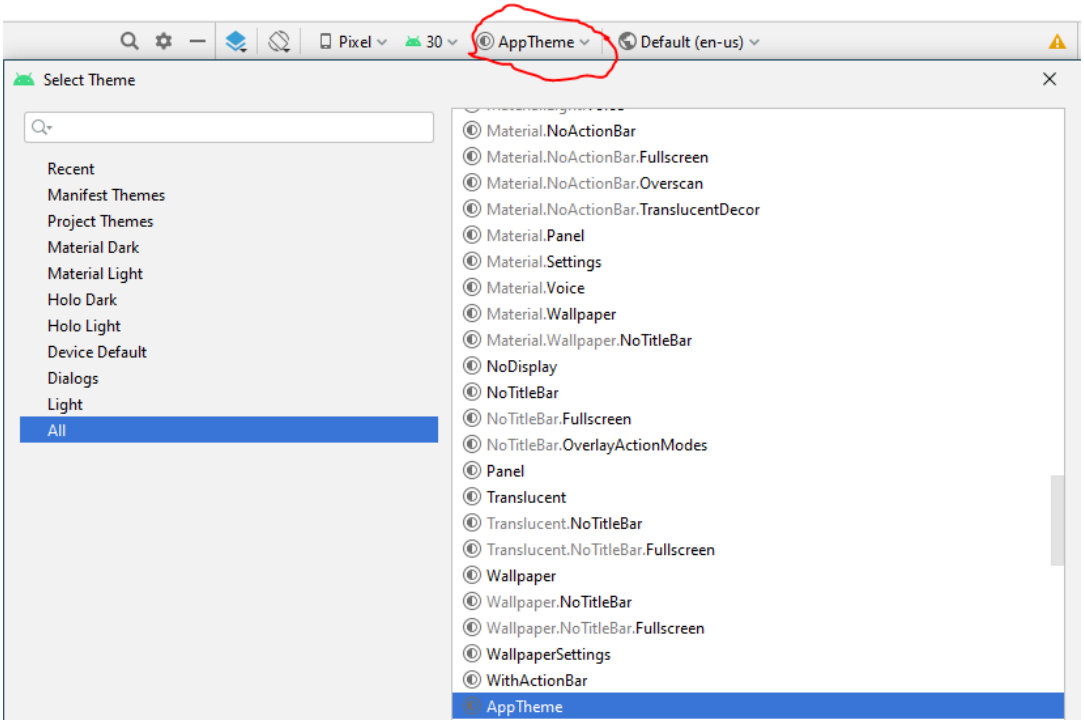
- Podemos visualizar como quedaría nuestro diseño con una resolución determinada.
- Podemos seleccionar una de las preconfiguradas, eligir alguna de los emuladores creados o crear uno nuevo.

Modo API



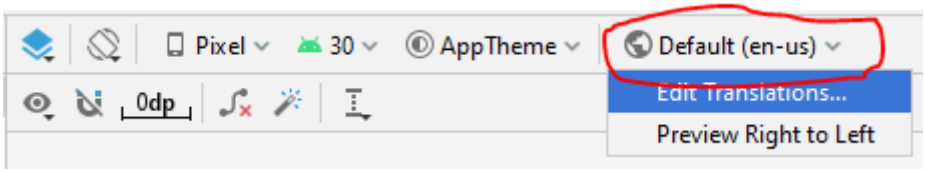
- De las API'S instaladas, podemos seleccionar una de ellas para ver como quedaría el aspecto de la aplicación.

Modo Aspecto



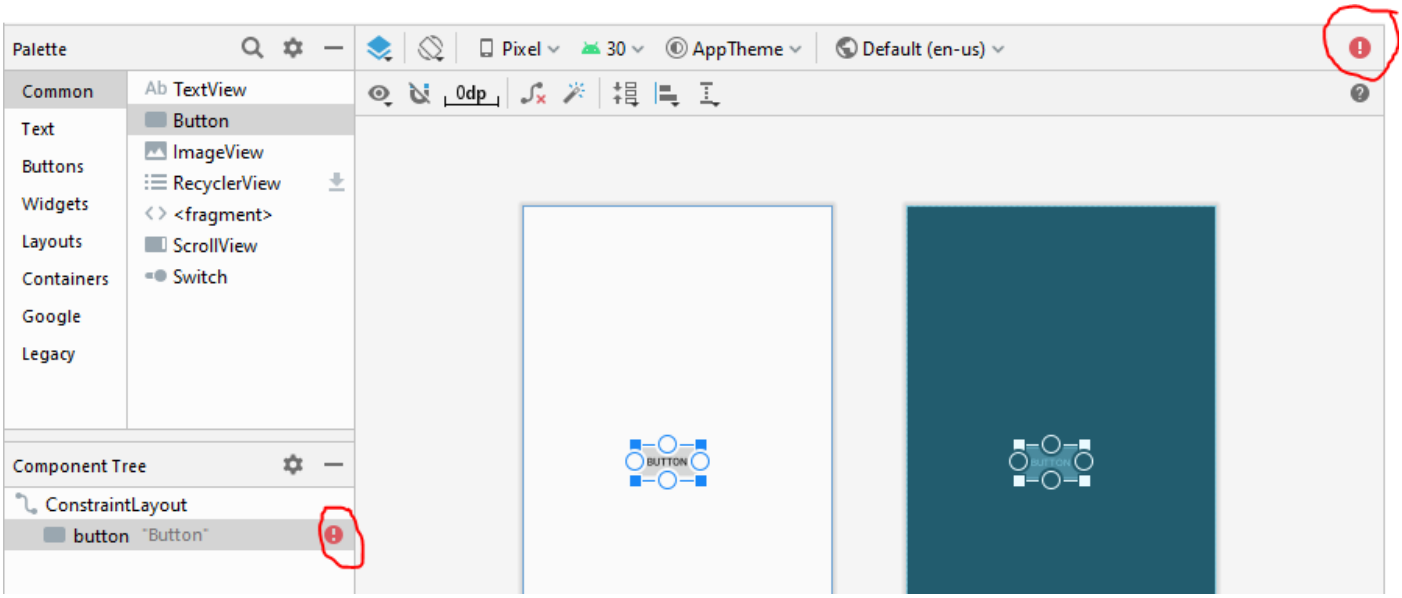
- Podemos cambiar el aspecto de la activity por uno de los predefinidos o por uno de los creados por nosotros.

Modo Idioma



- Si tuviesemos creados archivos de idiomas, podríamos ver el aspecto en diferentes idiomas. Lo veremos en la sección de internacionalización.

Errores o avisos en el diseño



- Si tenemos mal el diseño (atributos mal puestos, valores incorrectos,...) aparecerá un aviso en los puntos indicados en la imagen.
- Pulsando sobre el icono aparecerá la descrición del error.