CENTRO INTEGRADO de FORMACIÓN PROFESIONAL
AVILÉS

PROGRAMACIÓN MULTIMEDIA Y DISPOSITIVOS MÓVILES

- Android Studio-

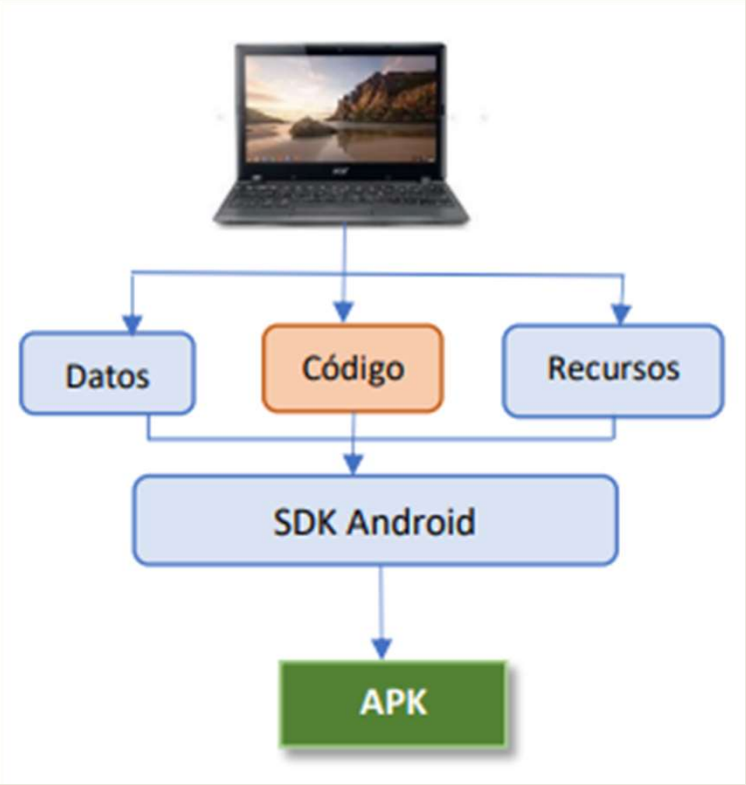
PROFESOR: PATRICIA SÁNCHEZ FORMOSO
patriciasfo@educastur.org
[CURSO 2024 - 2025]

INTRODUCCIÓN (1)

- Android es un sistema operativo multiusuario basado en Linux, donde **cada aplicación es considerada propiedad de un usuario distinto**. El **sistema asigna un identificador de usuario diferente a cada aplicación**, por lo que los archivos incluidos en cada una tendrán permisos sólo para ese usuario y solo él podrá acceder a ellos.
- Esto hace que **cada aplicación tenga su propio sistema de seguridad, con usuarios diferentes para cada una de ellas**, permisos propios para cada usuario con máquinas virtuales propias y un proceso Linux propio.
- Android utiliza el **principio de menor privilegio**, dando los permisos justos a cada aplicación.
- De todos modos, una aplicación Android, puede solicitar permisos para acceder a datos (agenda), recursos como: SD, Bluethooth, o funcionalidades como SMS, Cámara, Linterna. Los permisos asociados a la aplicación se conceden en la instalación de esta o en el momento de usarlos, y los otorga el usuario.

INTRODUCCIÓN (2)

- **Kotlin** es el lenguaje en el que se escriben las aplicaciones Android, apoyado por **XML**. El **SDK** (o kit de desarrollo) de Android compila el código Kotlin, datos y recursos, empaquetando todo en un fichero **APK**.



COMPONENTES (1)

- Los componentes de una aplicación en Android son bloques de código mediante los cuales se comunica el sistema y la aplicación. Cada uno de los componentes tiene una identidad propia y cumple un papel específico.
- Hay 4 tipos distintos de componentes, y cada uno de ellos tiene un ciclo de vida diferente y un fin específico:
  - *Activity: es el principal componente de una aplicación Android. Se encarga de gestionar gran parte de las interacciones con el usuario. Representa una pantalla independiente con una interfaz de usuario. Una aplicación puede tener varias actividades, aunque sean independientes, colaboran entre sí en el funcionamiento de la aplicación.*
  - *Service: son aplicaciones que se ejecutan de fondo para hacer operaciones de larga duración o trabajo en procesos remotos. Un servicio no tiene interfaz de usuario. Una actividad puede iniciar el servicio y permitir que se ejecute o enlazarse con él para desarrollar su cometido.*

COMPONENTES (2)

- *Content Provider: se encarga de gestionar un conjunto de datos de una aplicación para compartir. A través del proveedor de contenido, otras aplicaciones pueden consultar o incluso modificar información (si el proveedor lo permite).*
- *Broadcast Receiver: es una utilidad de Android que permite responder a anuncios broadcast (difusión) del sistema. Frecuentemente, un receptor de mensajes es simplemente un enlace con otros componentes realizando una cantidad mínima de trabajo.*

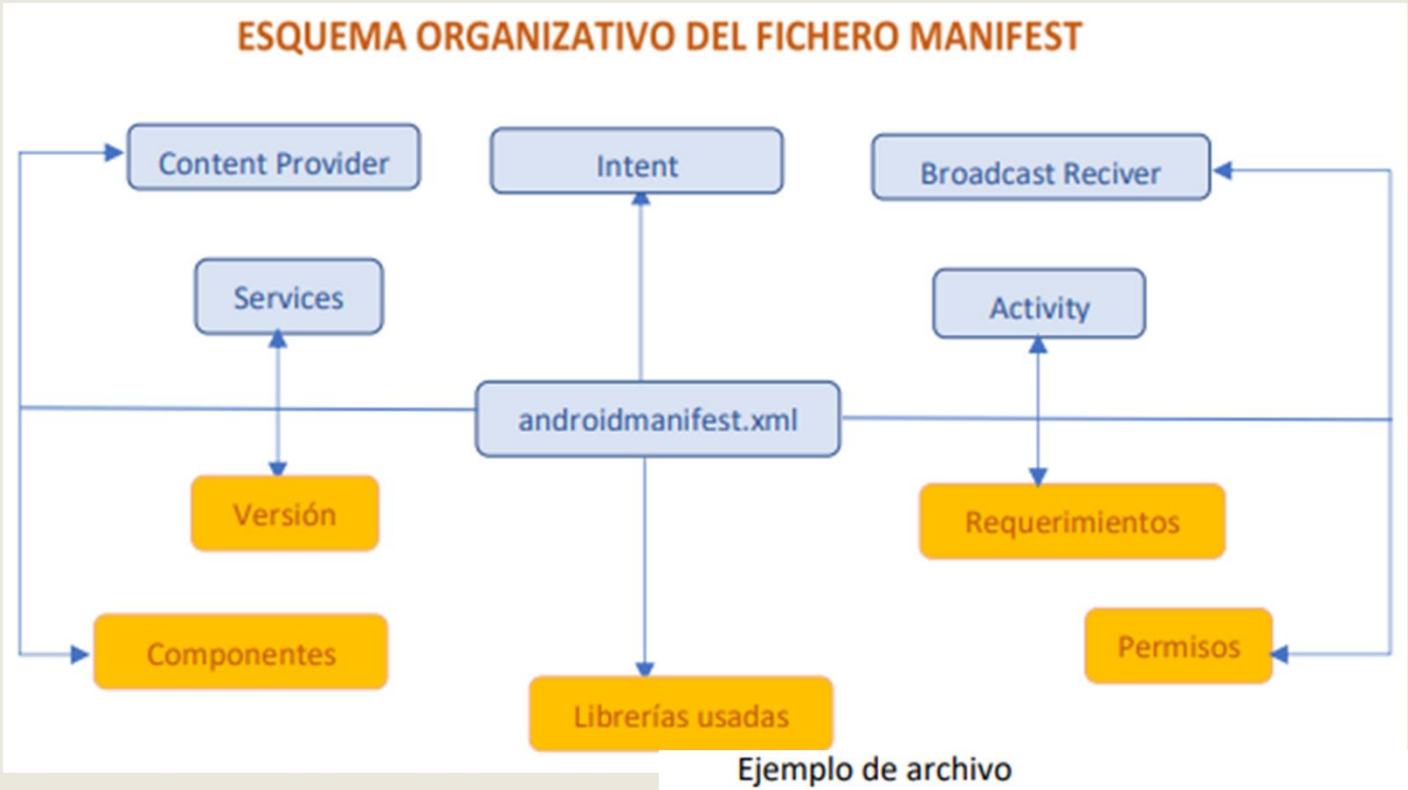
Aplicación Android esta compuesta por uno o más componentes de aplicación

- ✓ Actividades
- ✓ Servicios
- ✓ Proveedores de Contenidos
- ✓ Emisores de Notificaciones

ANDROID MANIFEST (1)

- Para incluir algunos de los anteriores componentes en las aplicaciones será necesario que el sistema reconozca la existencia del componente, y eso lo hace leyendo el fichero **AndroidManifest.xml** de la aplicación.
- Este archivo proporciona información esencial sobre la aplicación al sistema Android, información que el sistema debe tener para poder ejecutar el código de la APP. Todas las aplicaciones tienen este archivo. La gestión de este archivo es fácil a través del uso de un IDE. Una aplicación Android debe tener declarados todos sus componentes en este archivo.
- La principal tarea del Manifest es informar al sistema de los componentes de la aplicación. Otras funciones de este archivo serán registrar los permisos asociados a la aplicación, librerías que utiliza, declarar las necesidades del software/hardware, así como el nivel de API mínimo que requiere la aplicación.

ANDROID MANIFEST (2)



Ejemplo de archivo

```
<?xml versión="1.0" encoding="utf-9"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="es.aplicacion.ejemplo">
    <application
    android:icon="@mipmap/ic_launcher" android:label="@string/app_name" android:theme="@style/AppTheme">
        <activity android:name=".Fragmentos">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

ANDROID MANIFEST (3)

- Las etiquetas que es necesario conocer inicialmente son las siguientes:
  - *manifest: engloba a las demás etiquetas. Define el espacio de nombres, el nombre del paquete y los atributos del mismo.*
  - *application: contiene los metadatos de la aplicación (título, icono, tema, …), etiquetas de actividades, servicios, proveedores de contenidos y receptores de broadcast.*
  - *uses-sdk: indica las versiones del SDK sobre las que podrá ejecutarse, el nivel mínimo de API y el utilizado para su desarrollo.*
  - *uses-permission: declara los permisos que la aplicación necesita para operar. Se presentan al usuario durante la instalación para que los acepte o deniegue. Algunos de los permisos que podemos declarar aquí son:*
    - INTERNET: conexión a internet
    - READ_CONTACTS/ WRITE_CONTACTS: leer / escribir en la lista de contactos
    - SEND_SMS: enviar SMS

ANDROID MANIFEST (4)

- ACCESS_COARSE_LOCATION: localización mediante telefonía
- ACCESS_FINE_LOCATION: localización mediante GPS
- BLUETHOOTH: permite el uso de Bluethooth

– ***permission:*** *define un permiso que se requiere para que otras aplicaciones puedan acceder a partes restringidas de la aplicación.*

– ***instrumentation:*** *permite definir un test de ejecución para las actividades y servicios.*

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"></uses-permission>
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

ANDROID MANIFEST (5)

- El documento en sí empieza con un nodo **<manifest>** en el que **se incluirá todo lo demás.**
- Después pueden ir las siguientes etiquetas, que son las únicas legales, **no se pueden crear nuevos tipos de etiquetas:**

<action>
<activity>
<activity-alias>
<application>
<category>
<data>
<grant-uri-permission>
<instrumentation>
<intent-filter>
<manifest>
<meta-data>
<permission>
<permission-group>
<permission-tree>
<provider>
<receiver>
<service>
<supports-screens>
<uses-configuration>
<uses-feature>
<uses-library>
<uses-permission>
<uses-sdk>

ANDROID MANIFEST (6)

- Sólo los elementos **<manifest>** y **<application>** son **obligatorios** y sólo pueden aparecer **una vez** en el manifest.
- El **elemento raíz** es **<manifest>**:
  - *Siempre tiene que existir.*
  - *Incluye a todos los demás elementos del archivo XML.*
  - *Atributos del elemento raíz:*
    - ***xmlns***: especifica un namespace que será utilizado en el resto del archivo manifest.
    - **package**: especifica el nombre del paquete raíz de nuestra aplicación. A partir de él se buscan todas las clases de la aplicación.
    - **android:versionCode**: especifica la versión de la aplicación. Tiene que ser un **número** que se **incrementa** cada vez que se publica una **nueva versión** de la aplicación. Es usado por Google Play para seguir nuestra aplicación.
    - **android:versionName**: especifica la versión de la aplicación. Es un **texto** y es lo que se va a **mostrar a los usuarios de Google Play**. Puede ser cualquier string que nos guste.

ANDROID MANIFEST (7)

- Todos los **valores** de los elementos se establecen **a través de atributos:**
  - *Casi todos son opcionales.*
  - *Empiezan por el prefijo* ***android:*** *o* ***tools:*** *porque así está definido en el espacio de nombres o namespace del <manifest>.*
    - Son namespaces estándar y siempre usaremos el mismo.
  - *Si se pueden especificar* ***varios valores, el elemento se repite*** *en lugar de enumerar varios valores dentro de un solo elemento.*
- **<intent-filter>** con "*android.intent.action.MAIN*" y "*android.intent.category.LAUNCHER*" anuncia una **activity** que inicia una aplicación, es decir, **que debería ser mostrada en el lanzador de la aplicación**.
  - *El icono y la etiqueta asociados al <intent-filter> serán los mostrados en el lanzador.*

ANDROID MANIFEST (8)

- Elemento **<application>:** describe los componentes de la aplicación.
  - *Atributos:*
    - **android:icon**: especifica el **icono** que se mostrará en Google Play para nuestra app y en el lanzador de dicha app en el teléfono. El **valor** de este atributo debe ser una **referencia** a un **recurso «mipmap» o «drawable»** que contenga una imagen y será el valor por defecto para todos los componentes de la app que incluyamos.
    - **android:label**: es el **string** que se mostrará en el lanzador de la app en el teléfono y en la barra de título. El **valor** de este atributo debe ser una **referencia** a un **recurso «string»** ubicado **en res/values/strings.xml**. Este valor también será el valor por defecto para todos los componentes de la app que incluyamos.
    - **android:debuggable**: para especificar si la app se puede **depurar o no**. Puede tomar los **valores true o false**. Por defecto, toma el valor false. Lo normal es poner este atributo a true durante el desarrollo y false justo antes de publicarla.
    - **android:allowBackup**: puede tomar los valores true o false y por defecto es true. Habilita a la infraestructura de Android a realizar un backup, a que guarde la aplicación y sus datos.

ANDROID MANIFEST (9)

- Elemento **<activity>** incluida en la aplicación.
  - *Atributos:*
    - **android:name**: especifica el **nombre** de la clase Activity. Se puede indicar de 2 formas:
      - ***Relativo al atributo package****: sólo se pone el nombre de la clase precedido de un «.». Ejemplo:* android:name=".MainActivity"
      - ***Con la ruta completa****, independientemente de si se especificó o no el paquete. Ejemplo: android:name="com.example.holamundo*.MainActivity"
    - **android:label**: es el **string** que se mostrará en la barra de título de la actividad, si la tiene. Si no tiene barra de título no se verá. Si la actividad es el punto de entrada a la aplicación, esta etiqueta será la que se muestre en el lanzador.
    - **android:screenOrientation**: es la orientación que se va a mostrar en la pantalla del dispositivo cuando se ejecute esta actividad. Puede tomar los valores: **portrait** (vertical) **o landscape** (horizontal). **Si no se especifica, el dispositivo escoge la orientación**.
      - *La acción por defecto es que la app rote la vista de acuerdo a su orientación.*
      - *Si el dispositivo cambia de orientación, la activity será destruida y creada de nuevo -> puede perder el estado actual de la actividad*

ANDROID MANIFEST (10)

- **android:configChanges**: Android está configurado para que cada vez que haya un cambio en la configuración del dispositivo durante la ejecución, la actividad se destruye y se crea una nueva. Se reinicia, por lo que se puede perder el estado actual de la actividad. Para intentar evitar este reinicio, se utiliza esta propiedad. Los valores más habituales que puede tomar son:
  - *keyboard -> cambia el tipo de teclado*
  - *keyboardHidden -> cambia al aparecer o desaparecer el teclado*
  - *orientation -> cambia la orientación del dispositivo*
  - *screenSize -> cambia el tamaño de la pantalla*

  Ejemplo*: android:configChanges=«keyboard|orientation|screenSize»*

  Usarse como último recurso. La mejor solución para gestionar los cambios de configuración en tiempo de ejecución se puede consultar aquí.

ANDROID MANIFEST (11)

- Los **componentes** deben **anunciar sus capacidades**, qué tipos de intents pueden aceptar mediante los **<intent-filter>**.
  - Hay que definir la action: el tipo de acción que se quiere realizar. Opcionalmente, se indican los datos sobre los que se quiere realizar la acción.
  - Ejemplos:

Capacidad de abrir un navegador

```
<intent-filter>
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:scheme="http"/>
</intent-filter>
```

Capacidad de enviar por email texto plano

```
<intent-filter>
    <action android:name="android.intent.action.SEND" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="text/plain" />
</intent-filter>
```

RECURSOS (1)

- Los recursos hacen referencia a los datos que utiliza la aplicación, entendiendo como tal las imágenes, textos, estilos. Estos junto con el código, configuran una aplicación.
- Estos recursos **pueden estar predeterminados por la aplicación** que se haya programado y se usarán siempre sin importar el tipo o la configuración del dispositivo sobre el que se vaya a ejecutar.
- O bien, pueden crearse recursos alternativos, que son los que se diseñan para usar con una configuración específica del dispositivo (tablets, pantallas grandes, etc.). Se debe tener en cuenta siempre la responsividad de la aplicación.
- Los recursos pueden estar gestionados por ficheros XML: deben tener asignado un identificador (id) y se alojan en la/s subcarpeta/s de la **carpeta de recursos res**. Los nombres de estas subcarpetas son limitados y predefinidos por Android.

RECURSOS (2)

- Los recursos más frecuentes, así como el tipo de recurso que pueden contener son:
  - *res/drawable: recursos dibujables, ficheros de bitmaps o de imágenes escalables.*
  - *res/layout: definidos como ficheros XML. Engloban los elementos visuales que definen el interfaz de nuestra aplicación.*
  - *res/animator: permite crear animaciones sencillas sobre uno o varios gráficos, como rotaciones, fading, movimiento o estiramiento. Cada animación se define en un fichero XML.*
  - *res/mipmap: se trata de un contenido similar al drawable, el directorio mipmap alberga elemento bitmap aunque se suele utilizar para especificar el icono de la aplicación.*
  - *res/values: se trata de ficheros XML, donde se configuran diferentes aspectos de la aplicación, como es el color, dimensiones, cadenas de texto, estilos, etc.*
  - *res/xml: almacena ficheros xml que son utilizados por Android para dar soporte a tareas múltiples, como la que permite configurar la búsqueda.*

RECURSOS (3)

- *res/raw: almacena ficheros raw (audio y vídeo). Hay que destacar que este directorio tiene algunas limitaciones en cuanto al nombre de archivos que puede tener (mayúsculas, números, caracteres especiales), por lo que, si se necesita usar el nombre real del archivo, entonces se deben colocar los archivos raw en el directorio asset de Android.*

¿Cómo referenciar los recursos?

RECURSOS (4)

¿Cómo acceder a los recursos?

- Todos los recursos tienen asignado un identificador (Android lo hace de manera automática), por tanto, queda registrado el tipo de recurso y su identificador en el archivo de la **clase R** de la aplicación. Esto permite que se puede acceder a él a través del código Kotlin o XML.
- Acceso a recursos en XML: se debe escribir el nombre del recurso y la dirección del archivo donde éste se encuentre.

```
<TextView
 android:id="@+id/texto"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:background="@color/colorPrimario"
 android:text="@string/titulo"
 style="@style/EstiloTitulo"/>
```

RECURSOS (5)

- Acceso a recursos en Kotlin: se debe localizar el recurso a través de la clase R, al igual que en XML, indicando el directorio (o fichero XML) y nombre del recurso.

  Ejemplos:

  1. Localizar un botón a través de su identificador y guardarlo en una variable:

     *val botonEnviar = **findViewById<Button>(R.id.botonEnviar)***

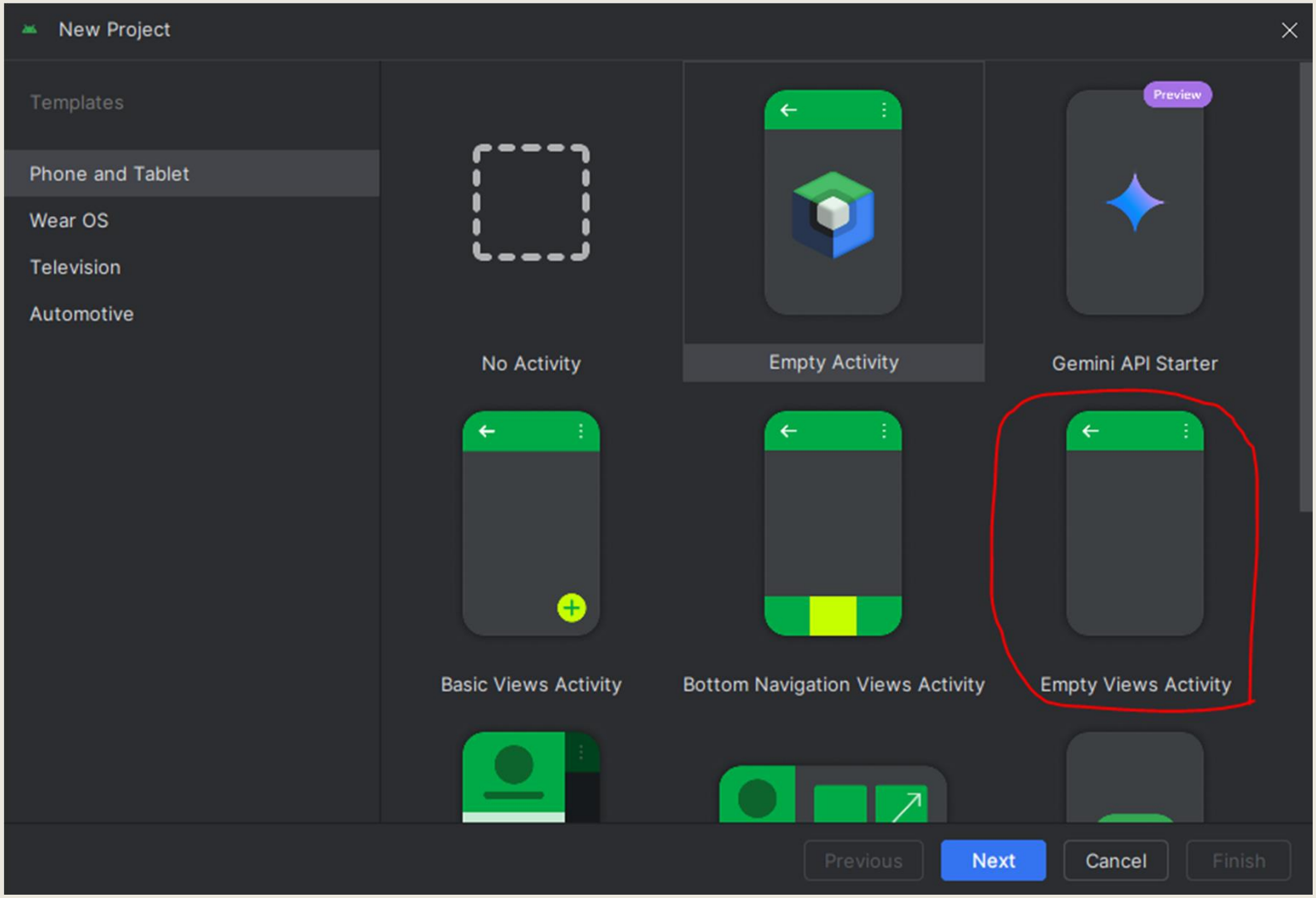
  2. Recuperar un string del fichero de recursos strings.xml

     *Toast.makeText(this, **R.string.login_incorrecto**, Toast.LENGTH_SHORT).show()*

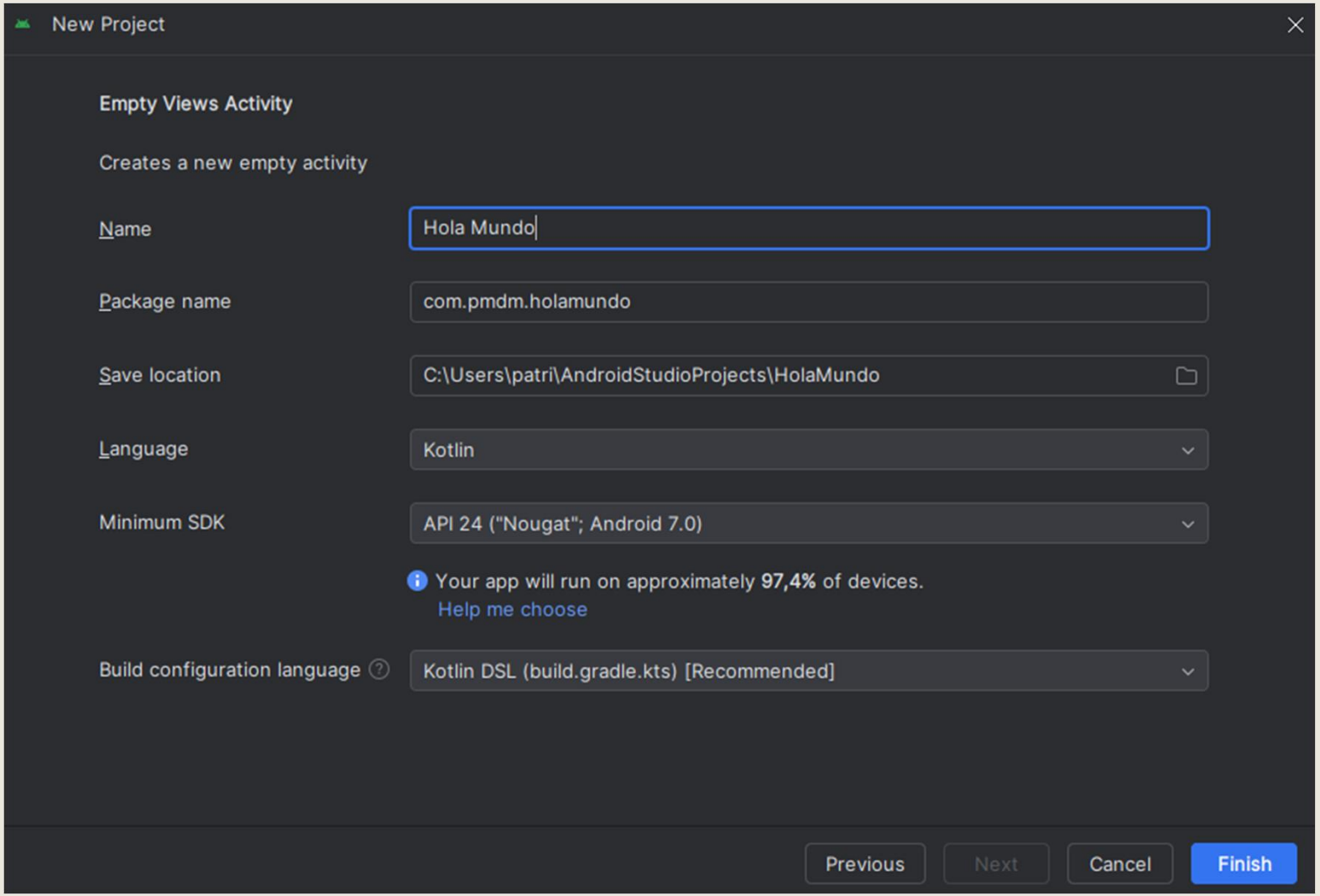
LA CLASE R

- La clase R se trata de una clase que contiene variables estáticas en las que se identifica cada tipo de recurso. Android lee el fichero XML, carga todas las estructuras solicitadas en memoria y mantiene el fichero R como referencia directa a los recursos cargados. Así cada atributo tiene una dirección de memoria asociada referenciada a un recurso específico.
- Es muy importante asegurarse de que no existan enlaces rotos, que no exista ningún error en los ficheros XML.
- Nunca se debe modificar el archivo R.java de manera manual. Este archivo es generado automáticamente a través de la herramienta AAPT (Android Asset Packaging Tool) cuando se compila el proyecto, por tanto, todos los cambios que se realicen de forma manual se anularán tras cada compilación.

ANDROID STUDIO (1)



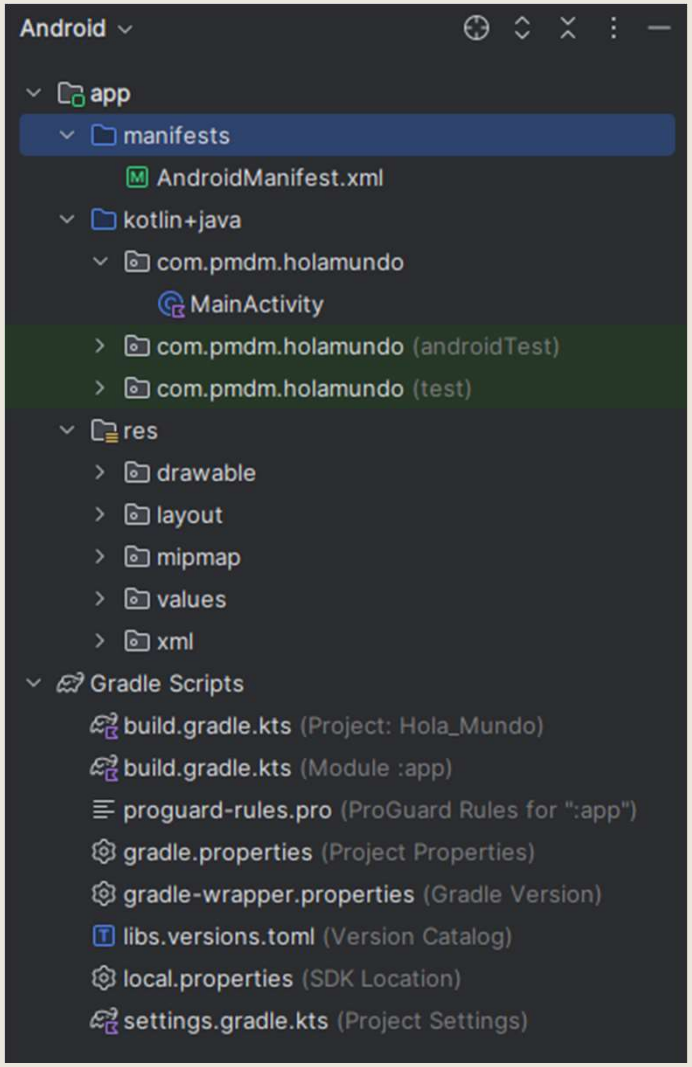
ANDROID STUDIO (2)



ANDROID STUDIO (3)

ELEMENTOS DE UN PROYECTO ANDROID

- Se muestran 2 carpetas principales:
  - *app*
  - *Gradle Scripts*
- La carpeta app contiene 3 subcarpetas:
  - ***manifest:*** *contiene el fichero AndroidManifest.xml.*
  - ***kotlin+java:*** *contiene los diferentes ficheros escritos en kotlin.*
  - ***res:*** *contiene los recursos de la aplicación y se divide en una estructura de directorios.*
- La carpeta *Gradle Scripts* contiene la información necesaria para la compilación del proyecto: versión del SDK de Android utilizada para compilar, la mínima versión de Android que soportará la aplicación, referencia a librerías externas, etc. El contenido de esta carpeta se modificará en ocasiones cuando incorporemos librerías externas o de terceras partes.



ANDROID STUDIO (4)

CONFIGURACIÓN DEL DISPOSITIVO PARA PROBAR LAS APPS

- Para probar el código se necesita un dispositivo dónde instalar al APK y visionar la aplicación. Para ello se puede optar por dos posibilidades:
  1. *Hacerlo con un emulador (el que nos proporciona Android o uno externo)*
  2. *Utilizar un dispositivo real.*
- Siempre es recomendable la segunda opción, ya que, además de ser más rápida y consumir menos recursos, se obtendrá una visión más real del resultado de la aplicación que se está desarrollando.
- Esta 2ª opción tiene la desventaja de que difícilmente se podrá ver el resultado en dispositivos variados en cuanto a dimensiones, versión de Android o características.
- Por tanto, la mejor opción es una combinación de las dos. Probar inicialmente en un dispositivo real, y según se vaya avanzando en el desarrollo hacerlo en otros dispositivos emulados para ver cómo responde en ellos.

ANDROID STUDIO (5)

- Para conectar un dispositivo físico a Android Studio es necesario que el SO lo reconozca y no de conflictos.
- **En el dispositivo móvil tienen que estar habilitadas las opciones de desarrollador**. Para esto, cada marca tiene su sistema, pero por lo general se suelen habilitar dentro del apartado 'Información del teléfono'.
- Si se tiene esta opción habilitada, Android se comunicará con el dispositivo móvil a través del ADB (Android Debug Bridge). Es muy importante que el sistema no muestre errores en el ADB Interface (necesario para el enlace USB). Esto suele ocurrir en versiones anteriores de Windows, cuando no están correctamente instalados los controladores ADB pertenecientes al modelo de móvil (en muchos casos, los generales de Google suelen ser suficientes). En caso de necesidad, se puede recurrir al Asistente de Conexión (Menú Tools -> App links Assistant).
- El emulador es sencillo de configurar ya que cuenta con un sencillo gestor (AVD Manager). Cuenta con algunos dispositivos preconfigurados y es posible realizar alguna configuración totalmente a medida.

ANDROID STUDIO (6)

LOS PERMISOS EN ANDROID

- Los permisos son declarados en el fichero **AndroidManifest.xml**, de tal manera que los usuarios finales son capaces de saber qué va a poder y qué no va a poder hacer la aplicación. Tanto Play Store como Android son responsables de mostrar al usuario dichos permisos, incluso antes de descargar la aplicación en nuestro dispositivo, debiendo el usuario decidir el aceptar o no los permisos, lo que, en la mayoría de los casos (de no aceptarlos) supondría la ausencia de parte o toda la funcionalidad de la aplicación.
- **A partir de la API 23 Marshmallow existe un nuevo administrador de permisos**, una medida de seguridad que nos dará más control y privacidad, que podemos revisar los permisos de las aplicaciones de una forma muy sencilla e intuitiva.

  Al instalar una aplicación, nos pedirá autorización por cada permiso que vaya a necesitar, de tal manera que podemos rechazarlo en un momento determinado y autorizarlo con posterioridad si así lo deseamos. Podemos gestionarlos además manualmente por aplicaciones o por tipo de permiso, de modo que las aplicaciones podrán tener permisos permanentes pero revocables.

ANDROID STUDIO (7)

Android nos informará mediante una notificación cuando una aplicación tenga algún permiso permanente y sensible en el momento de ejecutarla.

- Por ello, **a la hora de codificar aplicaciones no es suficiente con declarar los permisos más críticos en el AndroidManifest.xml, sino que también se debe solicitar acceso en el momento de su uso**.
- Si alguno de los permisos que se van a utilizar en al app aparece en la lista antes indicada, se ha de **incluir un código que compruebe si el permiso ha sido concedido**, lo que se hará **mediante el método ContextCompat.checkSelfPermission()**.
- Ejemplo: cómo comprobar si la actividad tiene permiso para utilizar la cámara:

  int permissionCheck = ContextCompat.checkSelfPermission(thisActivity, Manifest.permission.CAMERA);

ANDROID STUDIO (8)

- Si tiene el permiso autorizado, el método muestra **PackageManager.PERMISSION_GRANTED**, y esta puede continuar con la operación. Si, por el contrario no tiene permiso, el método muestra **PERMISSION_DENIED** y la aplicación debe solicitar permiso al usuario mediante el **método requestPermissions.** Después, se utilizará el **método onRequestPermissionsResult** para bifurcar el código, según este o no permitida la funcionalidad.

```
@Override
public void onRequestPermissionsResult(int requestCode, String permissions[], int[] grantResults){
    switch (requestCode){
        case MY_PERMISSIONS_CAMERA:{
            if(grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED){
                // Código de permiso aceptado
            }else{
                // Código de permiso denegado
            }
            return;
        }
    }

}
```

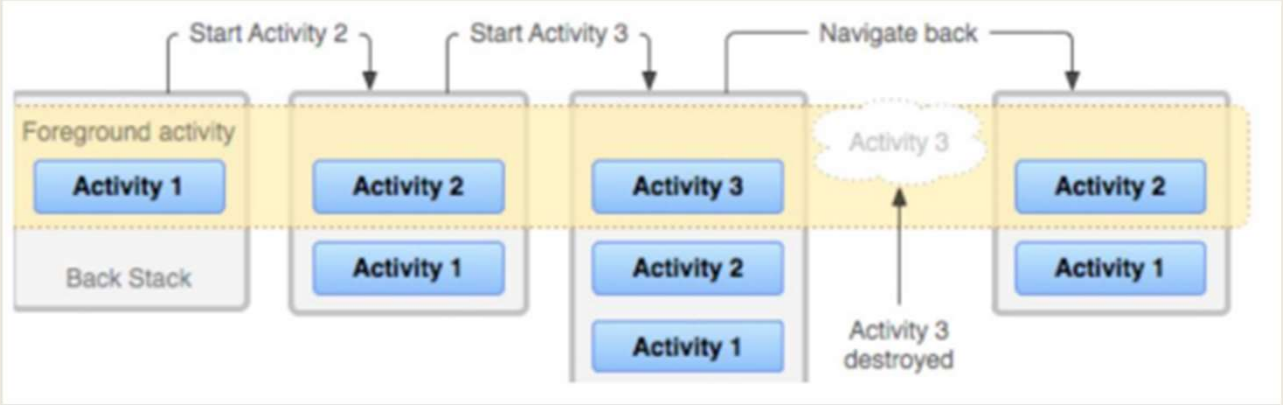
ANDROID STUDIO (9)

LAS ACTIVIDADES: ACTIVITY

- Es el principal componente de una aplicación Android. Se encarga de gestionar gran parte de las interacciones con el usuario. Representa una pantalla independiente con una interfaz de usuario. Una aplicación puede tener varias actividades, y aunque sean independientes, colaboran entre sí en el funcionamiento de la aplicación.
- Es cada una de las pantallas que ve el usuario y en ella se declaran los métodos (funciones) que se pueden hacer.
- Desde una pantalla principal, se podrán realizar desplazamientos de una a otra actividad y, conforme se va "navegando" par las diferentes pantallas de la aplicación, el dispositivo va recordando la secuencia visitada, de tal manera que, si se retrocede utilizando los botones del dispositivo o la lógica de navegación de la aplicación, irán apareciendo pantallas en sentido inverso a como se habían versionado.
- Android almacena la posición de cada una de las Actividades (activity, pantallas) en una pila (stack) también llamada Back Stack, en este caso de tipo LIFO, lo que significa que la última en entrar será la primera en salir.

ANDROID STUDIO (10)

- El sistema de funcionamiento de esta pila es muy simple. Cuando la actividad en curso inicia otra, esta nueva actividad obtiene el foco y es empujada a la parte superior de la pila, la actividad anterior permanece en la pila, pero detenida. El conjunto de actividades que tienen una relación lógica y que se encuentran en la pila se denomina tarea (task).
- Si el usuario regresa (back), la actividad que tenía el foco se elimina, y la actividad previa que se encontraba por debajo de ella en la pila se restaura y vuelve a coger el foco. Si se continúa pulsando hacia atrás, se irán eliminando sucesivamente tareas hasta llegar a la pantalla de inicio. Cuando todas las actividades se han eliminado de la pila, la tarea deja de existir y desaparece de la memoria.



ANDROID STUDIO (11)

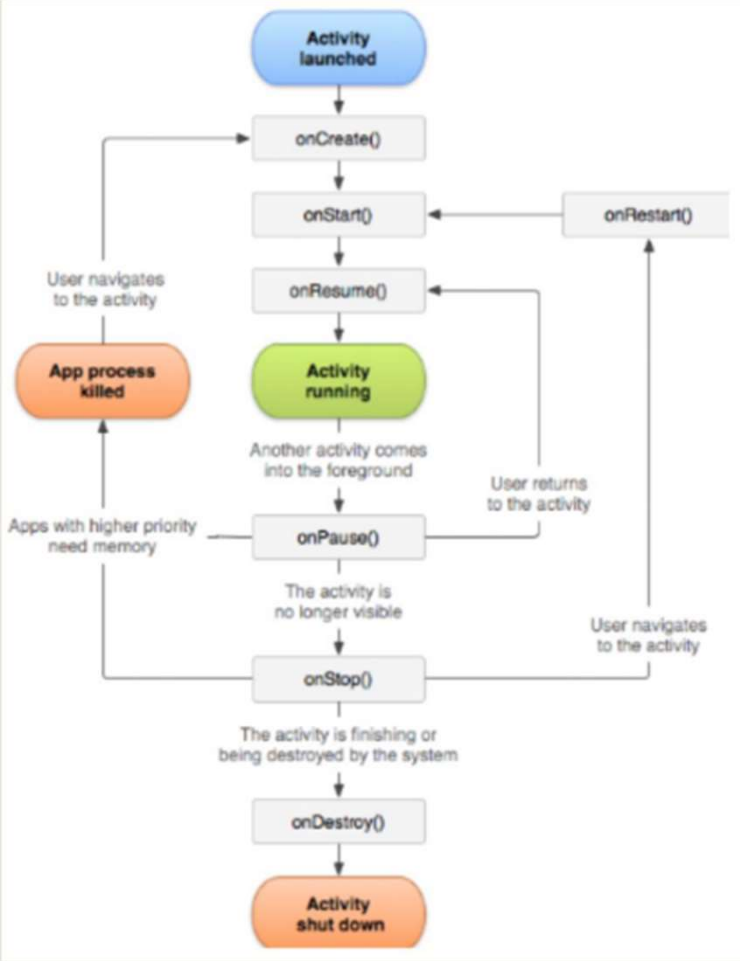
- Una tarea puede moverse a un segundo plano cuando el usuario comienza una nueva tarea o va a la pantalla Inicio. Sí está en segundo plano, todas las actividades de la tarea se paran, pero el Back Stack de esta tarea se conserva. Si una tarea vuelve a situarse en primer plano, esta se vuelve a posicionar en el lugar donde se había dejado.
- Ciclo de vida de una aplicación
  - *Una actividad puede estar en diferentes periodos de funcionamiento a los se denominan estado, y que van a depender del flujo de la aplicación, de la interacción con el usuario y de las necesidades de los recursos del sistema.*
  - *Al conjunto de estados en los que se puede encontrar una actividad se denomina ciclo de vida.*
  - *Los cambios de estados pueden ser controlados a través de métodos llamados por Android tipo callback:*
    - onCreate(): es llamado cuando la actividad es invocada por primera vez. Es el momento en que se crean las vistas (el interfaz de usuario), se hace una reserva de memoria para los datos, se obtienen las referencias e instancias de componentes mediante findViewById(), se suelen hacer las consultas iniciales en la fuentes de datos y se guardan las variables del entorno.

ANDROID STUDIO (12)

- onStart(): sucede al anterior. La actividad se hace visible para el usuario, pero aún no se puede interactuar con ella. Suele ser el lugar donde ubicar instrucciones asociadas al interfaz de usuario.
- onResume(): es llamado cuando la actividad puede interactuar con el usuario. Sucede tras el método anterior y precede al método *onPause()*. En este período se suelen activar sensores como cámara, actualizar información,…
- onPause(): sucede cuando la actividad pasa a segundo plano (back), bien sea porque está en un proceso de destrucción o porque otra actividad pasa a ocupar el primer plano. Es el momento cuando se detienen procesos, animaciones, etc. En este periodo la actividad es parcialmente visible, puede ocurrir que vuelva al primer plano (*onResume()*) o que sea parada (*onStop()*).
- onStop(): es invocado cuando la actividad ya no es visible al usuario y otra actividad ha pasado a primer plano. Se pausan animaciones, determinados servicios, se detiene (GPS) y se minimizan los recursos consumidos por la actividad, aunque la actividad se encuentra todavía en memoria. Si se quiere reactivar se llama al método *onRestart()* volviendo a primer plano, y se llama al método *onDestroy()* si queremos eliminarla definitivamente.
- onRestart(): es llamado cuando una actividad vuelve a primer plano, siempre seguido de un *onStart()*.

ANDROID STUDIO (13)

- **onDestroy()**: llamado cuando la actividad es definitivamente destruida y eliminada de memoria. En su interior se suelen limpiar y liberar recursos, por ejemplo, la cámara.



ANDROID STUDIO (14)

- Las activities tiene dos partes:
  - La parte lógica: la que se desarrolla con kotlin y es la parte que dará respuesta a cada una de las interacciones del usuario.
  - La parte visual: los layouts, que se verán más adelante
- La **Activity** es, básicamente, una clase kotlin (.kt) del tipo **AppCompactActivity** que tiene los métodos necesarios para poder comunicarnos con Android. Por defecto, siempre tendrá la función **onCreate**:
  - *Prepara un Bundle (o paquete de almacenaje) donde guardar las instancias y estados de la aplicación para ser recuperados en caso de necesidad.*
  - *Crea nuestra actividad y le asigna un layout.*
- **Recomendación**: añadir un Log informativo a la aplicación
  - *import android.util.Log*
  - *Ejemplo:*

```
Log.println(Log.DEBUG, null, "Email: $email y Contraseña: $password")
```

ANDROID STUDIO (15)

- Layout

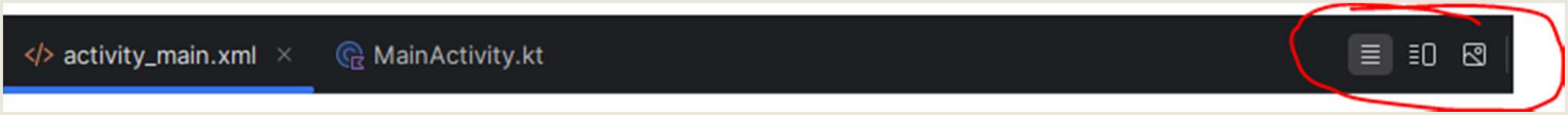
El **Layout** es la segunda parte de la actividad, la interfaz. Es donde se agregarán los componentes: los botones, las imágenes, etc.

Son archivos muy simples y se pueden completar arrastrando los componentes o picando el código en XML.

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/btn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignParentEnd="true"
        android:layout_marginTop="95dp"
        android:layout_marginEnd="366dp"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</RelativeLayout>
```

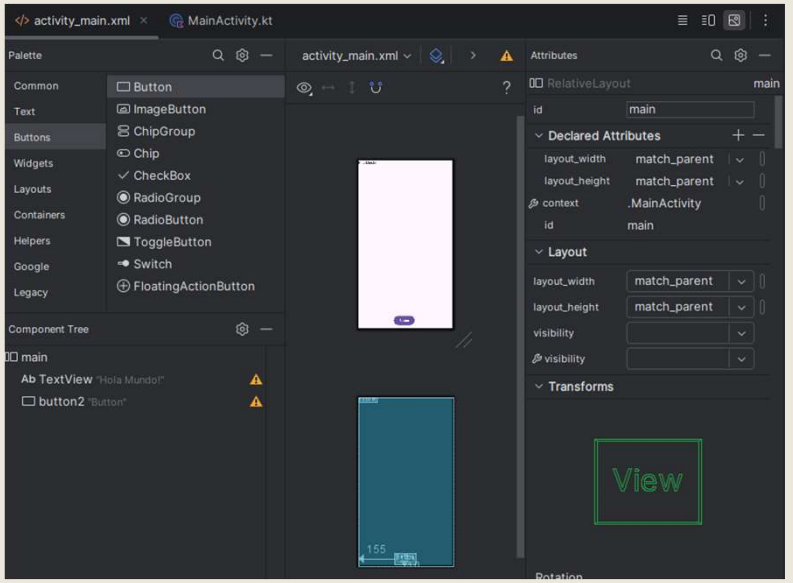
ANDROID STUDIO (16)

La parte más visual del layout: **Split** y **Design**.



Desde aquí, podemos manipular los componentes que están dentro de la vista, e incluso añadir nuevos.

Para ello vamos a coger un botón de la caja de componentes que está en la esquina superior izquierda. Simplemente pulsamos en el botón y lo arrastramos a la parte de la vista que queramos, por ejemplo abajo del todo.



Aparecerán una serie de campos en el lateral derecho:

son los **atributos** del botón.

Los podemos editar desde aquí o desde el XML.

La forma más óptima para dejar nuestras vistas

impolutas es modificar los atributos desde el XML.

ANDROID STUDIO (17)

Los atributos básicos que debemos editar son:

- ID: es el identificador único del componente. Con él, podremos definir en nuestra activity dicho componente para poder trabajar con él.

```
android:id="@+id/tvSaludo"
```

  Lo primero que se hace es llamar a Android, accediendo a la propiedad ID. Acto seguido, con el @+id estamos creando una referencia del componente en el archivo R.java de android y para acabar le damos el nombre que queramos, en este caso, al TextView que tiene el texto 'Hola Mundo!', lo llamaremos tvSaludo.

  La línea del ID como norma general en las prácticas del buen uso, suele ir la primera.

- layout_width: hace referencia a la anchura del componente, con este campo (que es obligatorio), podemos fijar la anchura a través de dos opciones. "Wrap_content" que añadirá una anchura lo suficientemente grande para incluir todo el contenido (en este caso el texto "BUTTON") y "Match_parent" que cogerá todo el espacio que su padre (en este caso el RelativeLayout) le permita.
- layout_height: permite asignarle la altura a cada componente. Sigue el mismo comportamiento que el "layout_width".

ANDROID STUDIO (18)

COMPONENTES BÁSICOS

- **TextView**: muestra un texto no editable en la pantalla.
- **EditText**: para introducir o escribir un texto.

Más atributos:

- **android:layout_below = "@+id/textView"**: indica que dicho componente estará justo debajo del que tenga el ID “textView”. Este atributo aplica sólo al RelativeLayout y los que hereden de él.
- **android:layout_above**: igual que android:layout_below pero esta vez el componente estará justo encima del que tenga el ID.
- **android:layout_centerHorizontal = "true"** : pondrá siempre el componente en el centro de la vista en el eje horizontal.
- **android:ems = "10":** añade una anchura al componente suficiente para que se vean X caracteres en la pantalla.

ANDROID STUDIO (19)

- **android:inputType = "textPersonName"**: permite que al hacer click en un editText se muestre el teclado adecuado al tipo de información. Otro ej: *textWebEmailAddress*
- **android:text = "TextView"**: indica el texto del componente.
- **android:textAppearance = "@style/TextAppearance.AppCompat.Title"**: indica al componente que su apariencia está controlada por un estilo.
- **android:hint = "Nombre"**: por defecto, en el editText se muestra un texto. Ese texto cuando hagamos click sobre él en el editText se esconderá y podremos escribir lo que queramos gracias a esta propiedad.

CONECTANDO LA VISTA AL CÓDIGO

Simplemente tenemos que usar el id que le asignamos a cada uno de los componentes.

Por ejemplo: al pulsar el botón ‘Enviar’ se valide que se haya indicado un nombre en el TextView.

ANDROID STUDIO (20)

El evento que se ejecuta cuando hacemos click en un botón es **setOnClickListener.**

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_main)
        val btnEnviar = findViewById<Button>(R.id.btnEnviar)
        btnEnviar.setOnClickListener{
            checkValue( ctx: this, findViewById<EditText>(R.id.etName))
        }
    }
}

fun checkValue(ctx: AppCompatActivity, etNombre: EditText){
    if(etNombre.text.toString().isEmpty()){
        Toast.makeText(ctx,  text: "El nombre no puede estar vacío", Toast.LENGTH_SHORT).show()
    }else{
        //Iremos a otra pantalla
    }
}
```

ANDROID STUDIO (21)

Lo primero que tenemos que hacer es obtener el botón (componente visual) a través de su ID utilizando el método **findViewById**.

Luego se llama al evento del click del botón (**setOnClickListener**) que ejecuta el método para validar el nombre introducido en el EditText.

Si el nombre está vacío, lanza un **Toast** indicándolo y si no, mostrará otra pantalla (que haremos a continuación).

Toast

Un toast es un aviso que proporciona información simple sobre una acción en una pequeña ventana emergente.

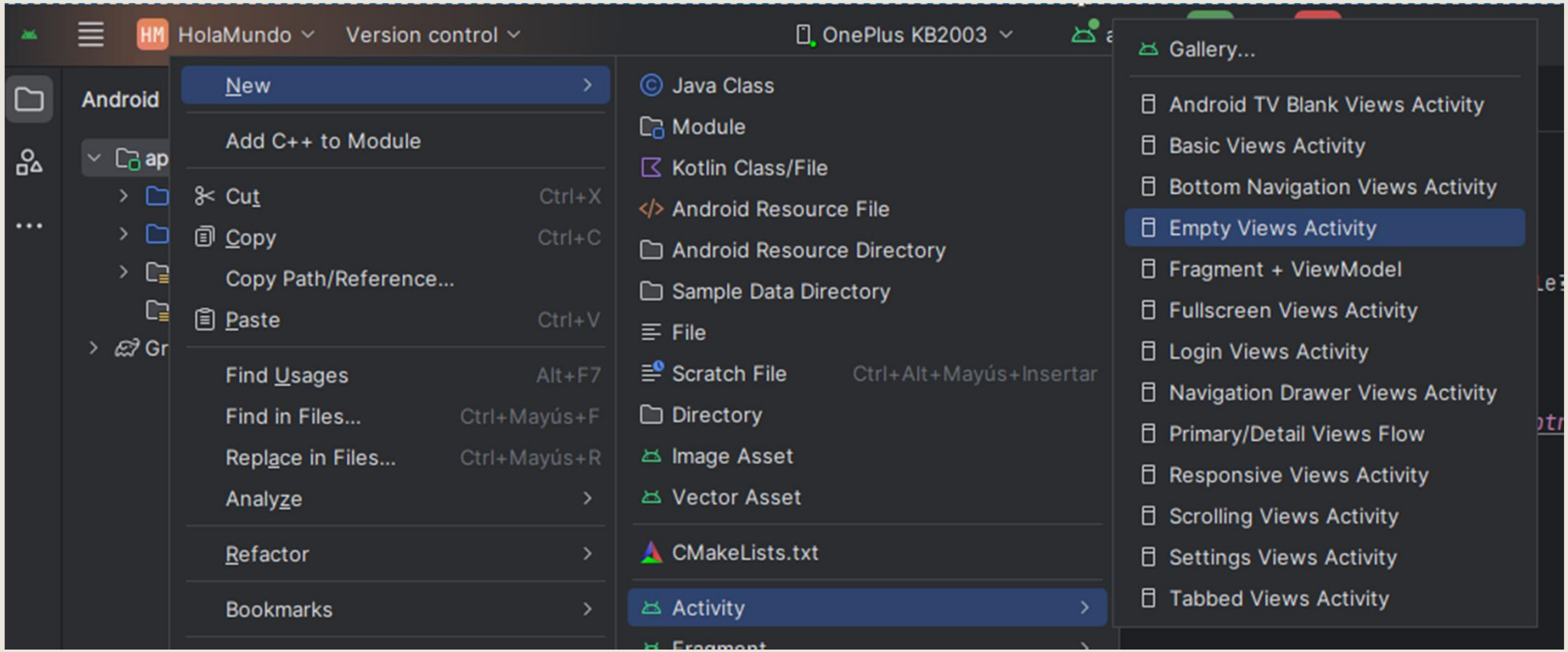
Solo ocupa la cantidad de espacio necesario para el mensaje, y la actividad en curso permanece visible y admite la interacción.

Los avisos desaparecen automáticamente después de un tiempo de espera.

Para más información, pincha aquí.

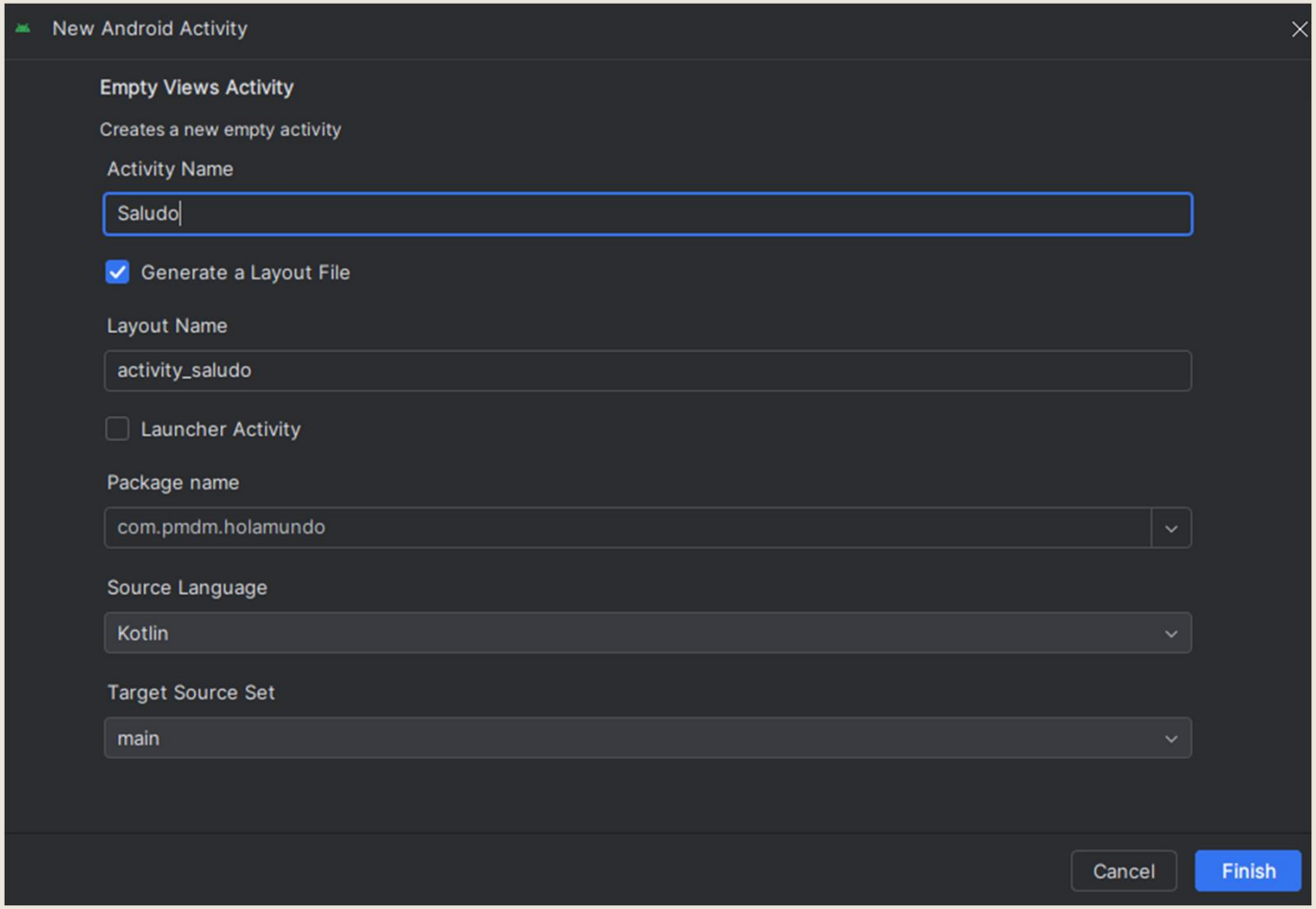
ANDROID STUDIO (22)

Creación de una segunda actividad



ANDROID STUDIO (23)

Creación de una segunda actividad



ANDROID STUDIO (24)

Vista:

- Lo primero será cambiar el ConstrantLayout por un RelativeLayout como en la anterior otra vista.
- Le pondremos un fondo de color azul oscuro y en el centro habrá un texto dándonos la bienvenida.
- En la parte inferior un botón para volver a la otra pantalla.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Saludo"
    android:background="@color/dark_blue">
```

ANDROID STUDIO (25)

- **android:background= "@color/dark_blue"**: da error porque el color dark_blue está creado por nosotros y aún no está definido en el archivo colors. Este archivo es un fichero de recursos sito en **app/res/values/colors.xml**.

  Podríamos haber definido directamente el código hexadecimal del color en el componente, pero no es lo correcto. Se añade en el fichero, y así sólo es necesario modificarlo aquí si algún día lo queremos cambiar.

- **android:text="@string/bienvenida"**: al igual que con los colores, tenemos que utilizar un fichero de recursos.

  Para todos los textos que añadamos en nuestra app.

  Este fichero está en **app/res/values/strings.xml**.

```
<resources>
    <string name="app_name">HolaMundo</string>
    <string name="bienvenida">Bienvenido/a %s</string>
    <string name="volver">Volver</string>
</resources>
```

```
<resources>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <color name="dark_blue">#dc3157ff</color>
</resources>
```

```
<TextView
    android:id="@+id/tvSaludo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true"
    android:layout_marginStart="172dp"
    android:layout_marginTop="319dp"
    android:text="@string/bienvenida"
    android:textColor="@color/white"
    android:layout_centerInParent="true"
    android:textSize="30sp"/>
```

ANDROID STUDIO (26)

```xml
<resources>
    <string name="app_name">HolaMundo</string>
    <string name="bienvenida">Bienvenido/a %s</string>
    <string name="volver">Volver</string>
</resources>
```

- El %s del string "bienvenida" es un parámetro para que Android entienda que ese %s será sustituido por un texto que aún desconocemos.

Lógica

Para enlazar las activities, hay que crear un **Intent:** es un objeto que contiene instrucciones con las cuales Android se comunica.

```kotlin
fun checkValue(ctx: AppCompatActivity){
    val etNombre = ctx.findViewById<EditText>(R.id.etName)
    if (etNombre.text.toString().isEmpty()){
        Toast.makeText(ctx, text: "El nombre no puede estar vacio", Toast.LENGTH_SHORT).show()
    }else{
        //Iremos a otra pantalla
        val intent = Intent(ctx, Saludo::class.java)
        intent.putExtra( name: "nombre", etNombre.text.toString())
        startActivity(ctx, intent, options: null)
    }
}
```

ANDROID STUDIO (27)

- **val intent = Intent(ctx, Saludo::class.java)**

Se crea el intent: hay que pasar el contexto y luego la pantalla a la que queremos ir, en este caso **Saludo**.

- **Intent.putExtra("nombre", etNombre.text.toString())**

Al intent se le puede añadir información "*extra*". Estos extras no son mas que datos para recuperar en otra parte de la aplicación.

Para poder recuperarlos hay que ponerles una clave a dichos datos. En este caso, la clave es "nombre" y el valor que almacena, el texto que se introduzca en el EditText.

Con esto hacemos que cuando llegue el intent a la otra pantalla, solo tenga que buscar si hay algún valor con la clave **nombre**, y nos devolverá el nombre.

- **startActivity(ctx, intent, null)**

Lanzará la actividad declarada en el intent que le pasamos.

ANDROID STUDIO (28)

Ahora ya sólo queda recuperar el extra en la nueva pantalla y pintarlo.

```
class Saludo : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_saludo)
        getAndShowName( ctx: this)
        findViewById<Button>(R.id.btnVolver).setOnClickListener{
            onBackPressedDispatcher.onBackPressed()
        }
    }

    fun getAndShowName(ctx: AppCompatActivity){
        val parametros = intent.extras
        val nombre = parametros?.getString( key: "nombre")
        val saludo = ctx.findViewById<TextView>(R.id.tvSaludo)
        saludo.text = getString(R.string.bienvenida, nombre)
    }
}
```

ANDROID STUDIO (29)

- val parametros = intent.extras

  val nombre = parametros?.getString("nombre")

Obtiene del bundle (un objeto dentro del intent que almacena la información) el valor cuya clave es nombre. El bundle sólo permite llamadas seguras (**?**) o no nulas (**!!**).

- saludo.text = getString(R.string.*bienvenida*, nombre)

Al TextView se le pone el texto deseado.

Si el mensaje tuviese más de un parámetro se indicarían todos una a continuación del otro. Por ejemplo: **getString(R.string.login, usuario, password)** y en el fichero ***strings.xml*** habría una entrada <string name="login"> Login con usuario %s y contraseña %s</string>

- findViewById<Button>(R.id.*btnVolver*).setOnClickListener{
    onBackPressedDispatcher.onBackPressed()

  }

Al botón ***Volver*** se le añade otro listener para que al hacer click ejecute la función **onBackPressedDispatcher.onBackPressed()** que hace que vuelva a la actividad anterior.

ANDROID STUDIO (30)

TIPOS DE LAYOUTS

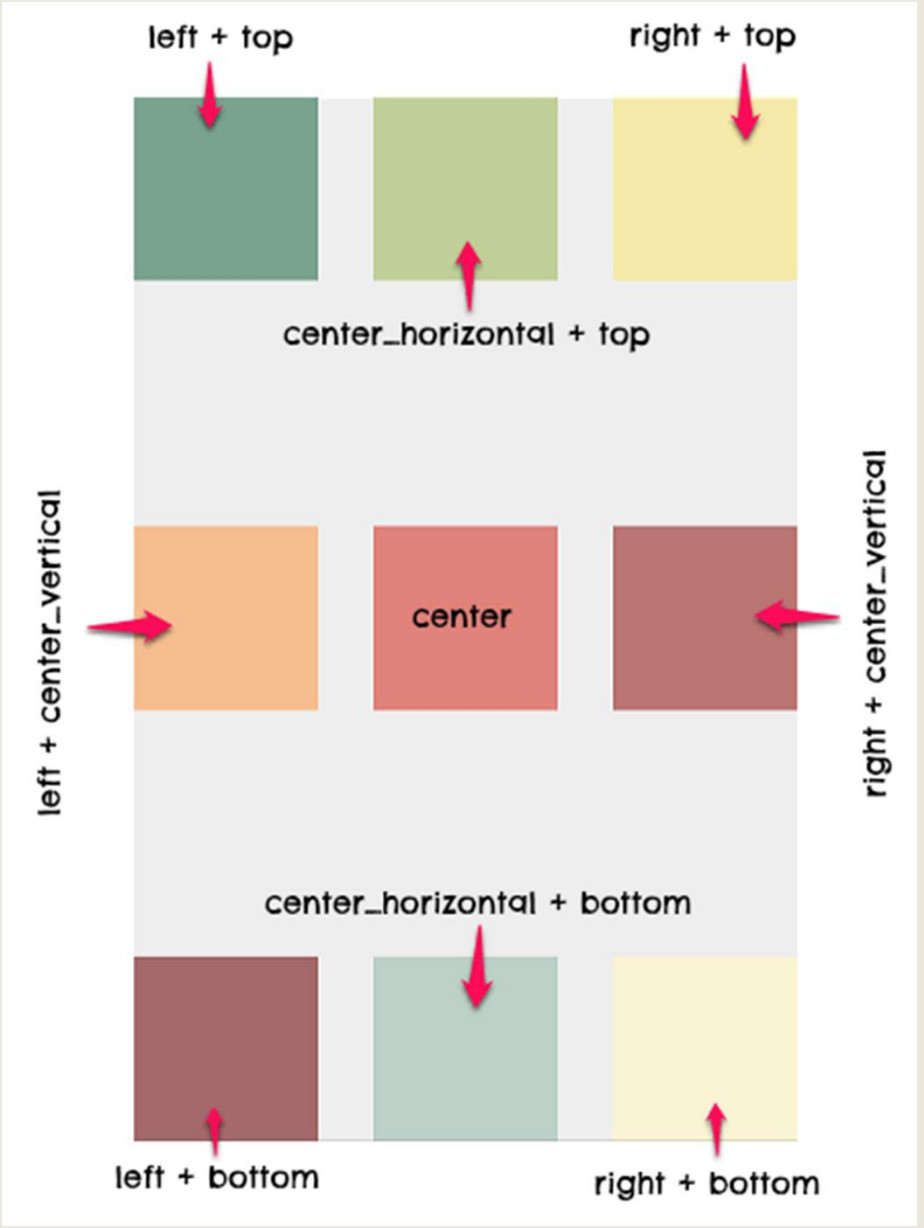
- **FrameLayout**: es uno de los diseños más simples para organizar los controles de vista. Están diseñados para bloquear un área de la pantalla. Suele ser para añadir un solo hijo o para vistas muy sencillas.
- **LinearLayout**: es el más sencillo y los elementos se irán apilando horizontal o verticalmente en función del parámetro **orientation**.
- **TableLayout**: coloca los elementos siguiendo una distribución de tabla.
- **GridLayout**: alinea los elementos hijos en una cuadrícula. Nace con el fin de evitar anidar LinearLayouts para crear diseños complejos.
- **RelativeLayout**: los elementos se posicionan en función de los que tienen al lado. Ha sido sustituido por el **ConstraintLayout**.

ANDROID STUDIO (31)

FRAME LAYOUT

- Se pueden añadir varios hijos con el fin de superponerlos, donde el último hijo agregado, es el que se muestra en la parte superior y el resto se ponen por debajo en forma de pila.
- Para alinear cada elemento dentro del layout se usa el parámetro **android:layout_gravity**. Este parámetro se basa en las posiciones comunes de una vista dentro del layout. Se describe con las constantes de orientación:
  - *top* - *bottom* - *center*
  - *left* - *center_horizontal*
  - *right* - *center_vertical*

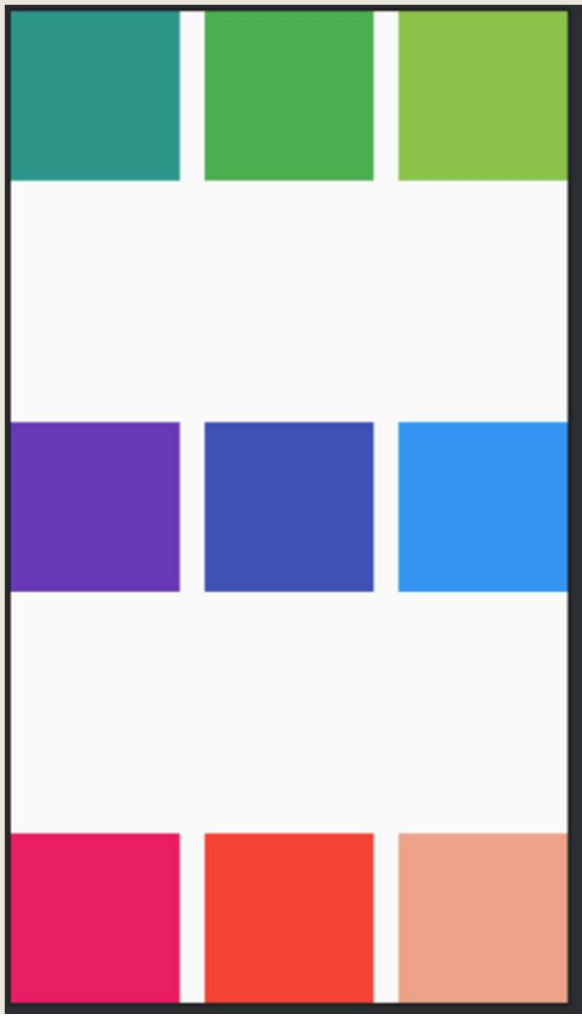
  Es posible crear variaciones combinadas y se representaría: *android:layout_gravity="right|bottom"*



ANDROID STUDIO (32)

FRAME LAYOUT

- Ejercicio 1
  - Crea un nuevo proyecto de nombre Layouts
  - Renombrar el layout a frame_layout.xml
  - Crear un diseño con 3 vistas dentro de un frame layout
    - Una imagen de fondo que recubra todo el layout (https://www.freepik.es/)
    - Una imagen centrada en ambas dimensiones para resaltar una estadística
    - Un botón alineado en la parte inferior
- Ejercicio 2
  - Conseguir la siguiente vista utilizando un FrameLayout y los colores que cada uno quiera.
    - Para representar los cuadrados utilizar ***View*** con tamaño fijo en dp (https://tyris-software.com/que-son-y-para-que-sirven-los-dp-en-android/).
- Ejercicio 3
  - Enlazar las vistas de tal forma que al pulsar el botón del ejercicio 1 se cargue la vista del ejercicio 2.



ANDROID STUDIO (33)

FRAME LAYOUT

- Solución Ejercicio1

frame_layout.xml × MainActivity.kt

```xml
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <ImageView
        android:id="@+id/imagen_background"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_gravity="center|top"
        android:contentDescription="@string/desc_imagen_fondo"
        android:scaleType="centerCrop"
        android:visibility="visible"
        app:srcCompat="@drawable/background_frame_layout"
        tools:visibility="visible" />
    <Button
        android:id="@+id/boton_saltar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal|bottom"
        android:text="@string/saltar" />
    <ImageView
        android:id="@+id/imagen_estadistica"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        app:srcCompat="@drawable/captura"
        android:contentDescription="@string/desc_imagen_estadistica"/>
</FrameLayout>
```

ANDROID STUDIO (34)

FRAME LAYOUT

- Solución Ejercicio2

frame_layout.xml | frame_layout2.xml | MainActivity.kt

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/main2"
    tools:context=".MainActivity">
    <View
        android:id="@+id/topleft"
        android:layout_width="125dp"
        android:layout_height="125dp"
        android:layout_gravity="top|start"
        android:foreground="@color/design_default_color_secondary_variant" />
    <View
        android:id="@+id/centertop"
        android:layout_width="125dp"
        android:layout_height="125dp"
        android:layout_gravity="center|top"
        android:foreground="#008f39" />
    <View
        android:id="@+id/righttop"
        android:layout_width="125dp"
        android:layout_height="125dp"
        android:layout_gravity="end|top"
        android:foreground="@android:color/holo_green_dark" />
    <View
        android:id="@+id/leftcenter"
        android:layout_width="125dp"
        android:layout_height="125dp"
        android:layout_gravity="start|center_vertical"
        android:foreground="?attr/elevationOverlayColor" />
    <View
        android:id="@+id/center"
        android:layout_width="125dp"
        android:layout_height="125dp"
        android:layout_gravity="center"
        android:foreground="@color/design_default_color_primary" />
    <View
        android:id="@+id/centerright"
        android:layout_width="125dp"
        android:layout_height="125dp"
        android:layout_gravity="center_vertical|end"
        android:foreground="@color/material_dynamic_primary70" />
    <View
        android:id="@+id/startbotom"
        android:layout_width="125dp"
        android:layout_height="125dp"
        android:layout_gravity="bottom|start"
        android:foreground="@android:color/holo_red_light" />
    <View
        android:id="@+id/centerbotom"
        android:layout_width="125dp"
        android:layout_height="125dp"
        android:layout_gravity="bottom|center_horizontal"
        android:foreground="@color/design_default_color_error" />
    <View
        android:id="@+id/endbotom"
        android:layout_width="125dp"
        android:layout_height="125dp"
        android:layout_gravity="bottom|end"
        android:foreground="?attr/colorTertiaryFixedDim" />
</FrameLayout>
```

ANDROID STUDIO (35)

FRAME LAYOUT

- Solución Ejercicio3

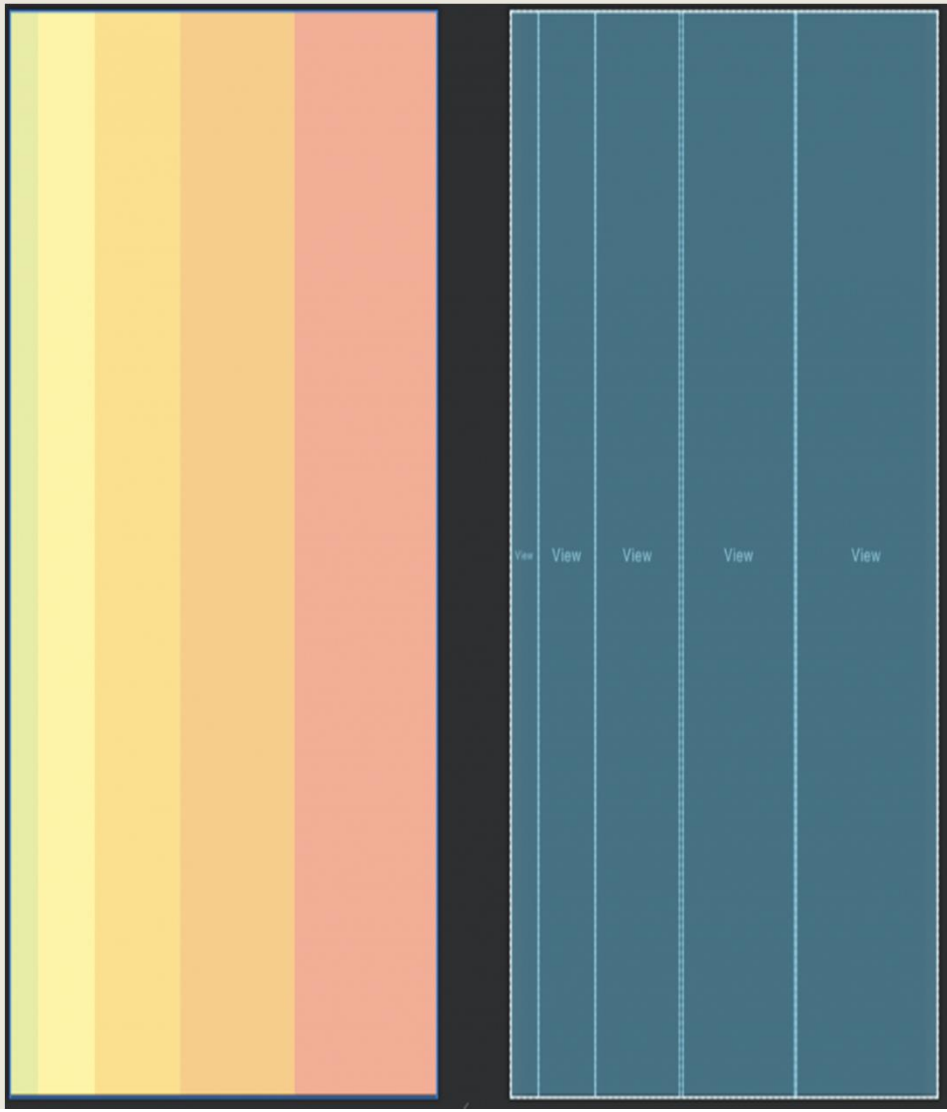
```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.frame_layout)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
            insets
        }
        val boton = findViewById<Button>(R.id.boton_saltar)
        boton.setOnClickListener {
            val intent = Intent( packageContext: this, ColorsActivity::class.java)
            startActivity(intent)
        }
    }
```

```
class ColorsActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_colors)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
            insets
        }
    }
}
```

ANDROID STUDIO (36)

LINEAR LAYOUT

- Para poder definir si las queremos apilar vertical u horizontalmente usaremos el atributo **android:orientation = "horizontal"** ó **"vertical"**
- Este layout nos permite trabajar con **pesos**: porcentajes que nos permiten decirles que ocupen un ancho relativo al porcentaje de la pantalla. **android:layout_weight**
  - Si orientation = horizontal, para que funcione el layout-width = 0dp
  - Si orientation = vertical, para que funcione el layout-height = 0dp
- Se pueden anidar todo tipo de LinearLayouts.
  - Anidar con pesos no es recomendable porque necesita muchos recursos para pintar la vista.



```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    tools:context=".LinearLayoutActivity">

    <View
        android:id="@+id/view1"
        android:layout_width="20dp"
        android:layout_height="match_parent"
        android:background="#8DC64B" />

    <View
        android:id="@+id/view2"
        android:layout_width="50dp"
        android:layout_height="match_parent"
        android:background="#FFEB3B" />

    <View
        android:id="@+id/view3"
        android:layout_width="80dp"
        android:layout_height="match_parent"
        android:background="#FAC41D" />

    <View
        android:id="@+id/view4"
        android:layout_width="110dp"
        android:layout_height="match_parent"
        android:background="#FF9800" />

    <View
        android:id="@+id/view5"
        android:layout_width="140dp"
        android:layout_height="match_parent"
        android:background="#FF5722" />

</LinearLayout>
```

Solución al LinearLayout especificando el ancho de cada vista. -> esta NO es la solución ideal

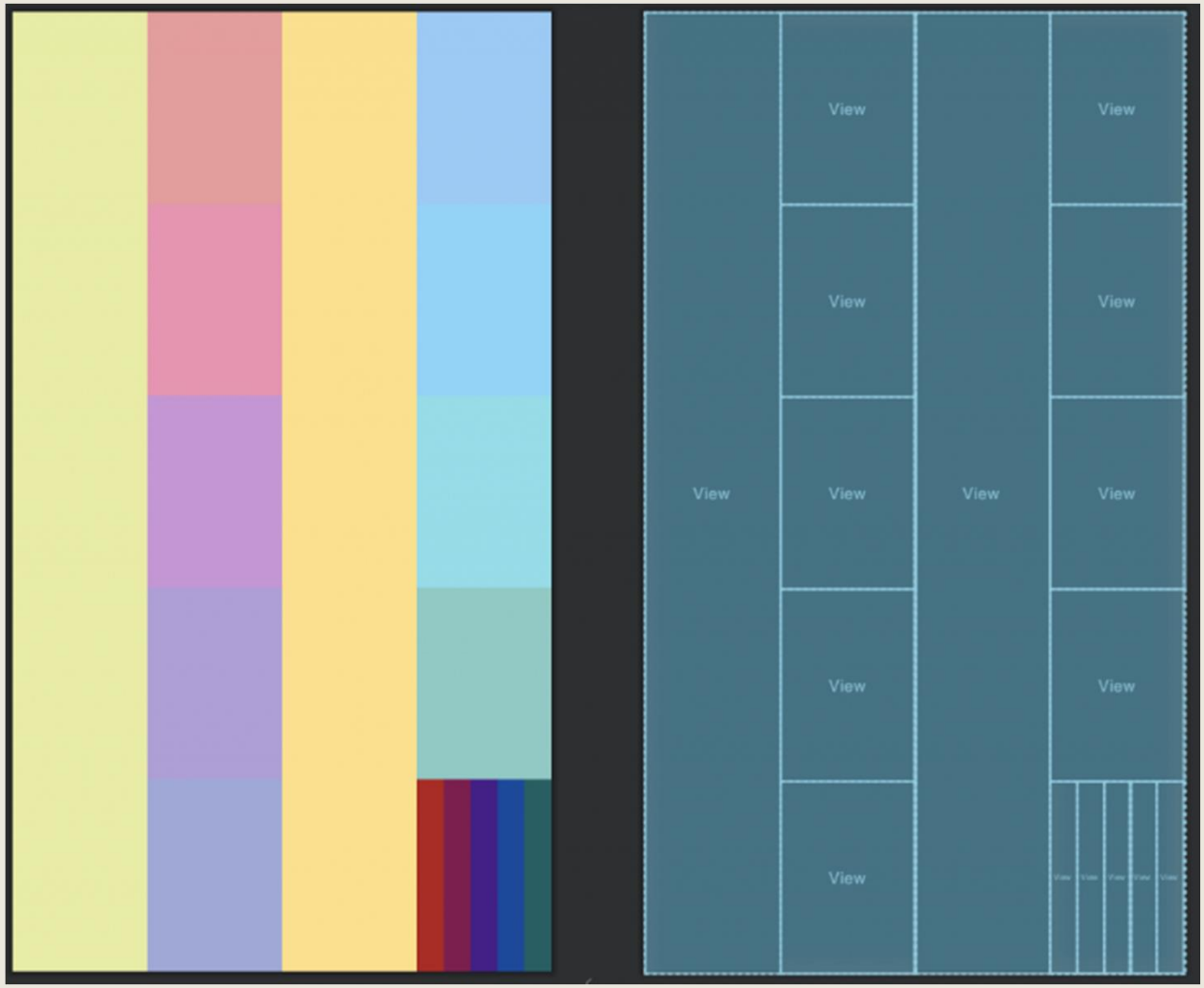
```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    tools:context=".LinearLayoutActivity">

    <View
        android:id="@+id/view1"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:background="#8DC64B" />

    <View
        android:id="@+id/view2"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="2"
        android:background="#FFEB3B" />

    <View
        android:id="@+id/view3"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="3"
        android:background="#FAC41D" />

    <View
        android:id="@+id/view4"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="4"
        android:background="#FF9800" />

    <View
        android:id="@+id/view5"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="5"
        android:background="#FF5722" />

</LinearLayout>
```

Solución al LinearLayout especificando pesos -> esta es la mejor solución.

ANDROID STUDIO (37)

LINEAR LAYOUT

- Ejercicio 1
  - Dibuja la siguiente vista



ANDROID STUDIO (38)

LINEAR LAYOUT

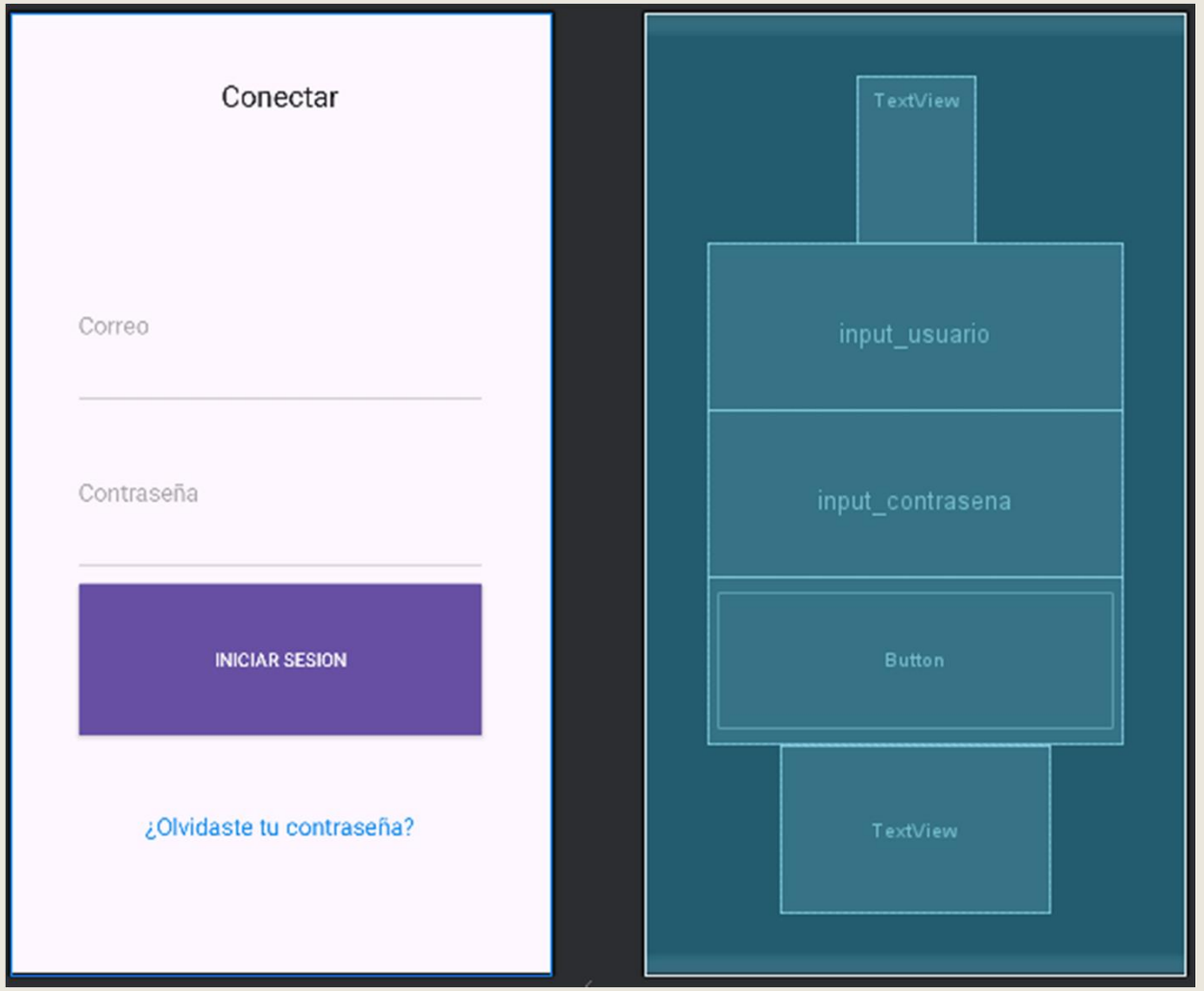
- Solución Ejercicio 1

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">
    <View
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="2"
        android:background="#E6EE9C" />
    <LinearLayout
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="2"
        android:orientation="vertical">
        <View
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:background="#EF9A9A" />
        <View
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:background="#F48FB1" />
        <View
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:background="#CE93D8" />
        <View
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:background="#B39DDB" />
        <View
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:background="#9FA8DA" />
    </LinearLayout>
    <View
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="2"
        android:background="#FFE082" />
    <LinearLayout
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="2"
        android:orientation="vertical">
        <View
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:background="#90CAF9" />
        <View
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:background="#81D4FA" />
        <View
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:background="#80DEEA" />
        <View
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:background="#80CBC4" />
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:orientation="horizontal">
        <View
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:background="#B71C1C" />
        <View
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:background="#880E4F" />
        <View
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:background="#4A148C" />
        <View
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:background="#0D47A1" />
            <View
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:background="#006064" />
        </LinearLayout>
    </LinearLayout>
</LinearLayout>
```

ANDROID STUDIO (39)

LINEAR LAYOUT

- Ejercicio 2
  - Dibuja la siguiente vista



ANDROID STUDIO (40)

LINEAR LAYOUT

- Solución Ejercicio 2

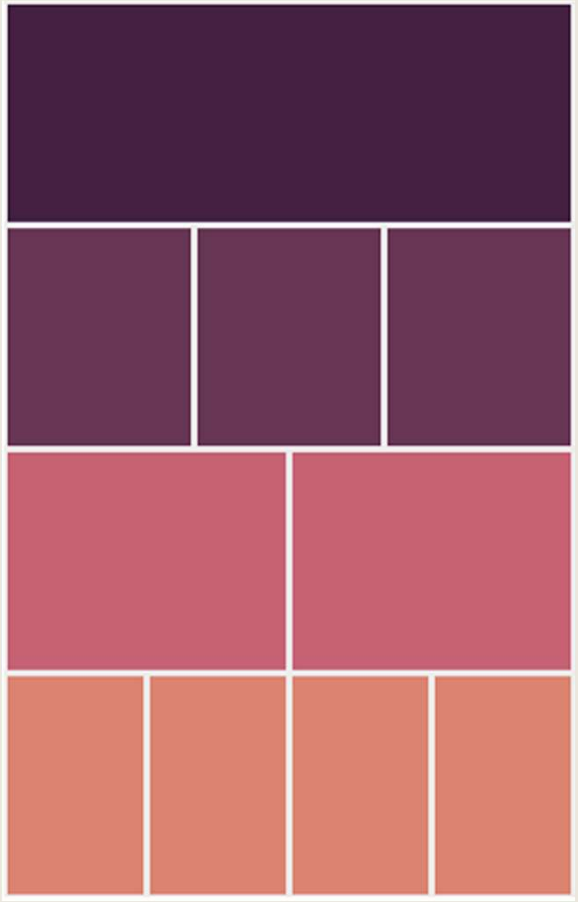
*ems: el ancho de la letra M en el tipo de fuente utilizada

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:id="@+id/main"
   android:orientation="vertical"
   android:padding="48dp">
   <TextView
      android:id="@+id/texto_conectar "
      android:layout_width="wrap_content "
      android:layout_height="0dp "
      android:layout_gravity="center_horizontal "
      android:layout_weight="1 "
      android:text="Conectar "
      android:textAppearance="?android:attr/textAppearanceLarge" />
   <EditText
      android:id="@+id/input_usuario "
      android:layout_width="match_parent "
      android:layout_height="0dp "
      android:layout_gravity="center_horizontal|center_vertical "
      android:layout_weight="1 "
      android:hint="Correo "
      android:inputType="textEmailAddress"/>
<EditText
      android:id="@+id/input_contrasena "
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:layout_gravity="center_horizontal"
      android:layout_weight="1"
      android:ems="10"
      android:hint="Contraseña"
      android:inputType="textPassword" />
<Button
      android:id="@+id/boton_iniciar_sesion"
      style="@style/Widget.AppCompat.Button.Small"
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:layout_gravity="center_horizontal"
            android:layout_weight="1"
      android:text="Iniciar Sesion" />
<TextView
      android:id="@+id/texto_olvidaste_contrasena"
      android:layout_width="wrap_content"
      android:layout_height="0dp"
      android:layout_gravity="center_horizontal"
      android:layout_weight="1"
      android:gravity="center_vertical"
      android:text="¿Olvidaste tu contraseña?"
android:textAppearance="?android:attr/textAppearanceMedium"
      android:textColor="#0E8AEE" />
```

ANDROID STUDIO (41)

TABLE LAYOUT

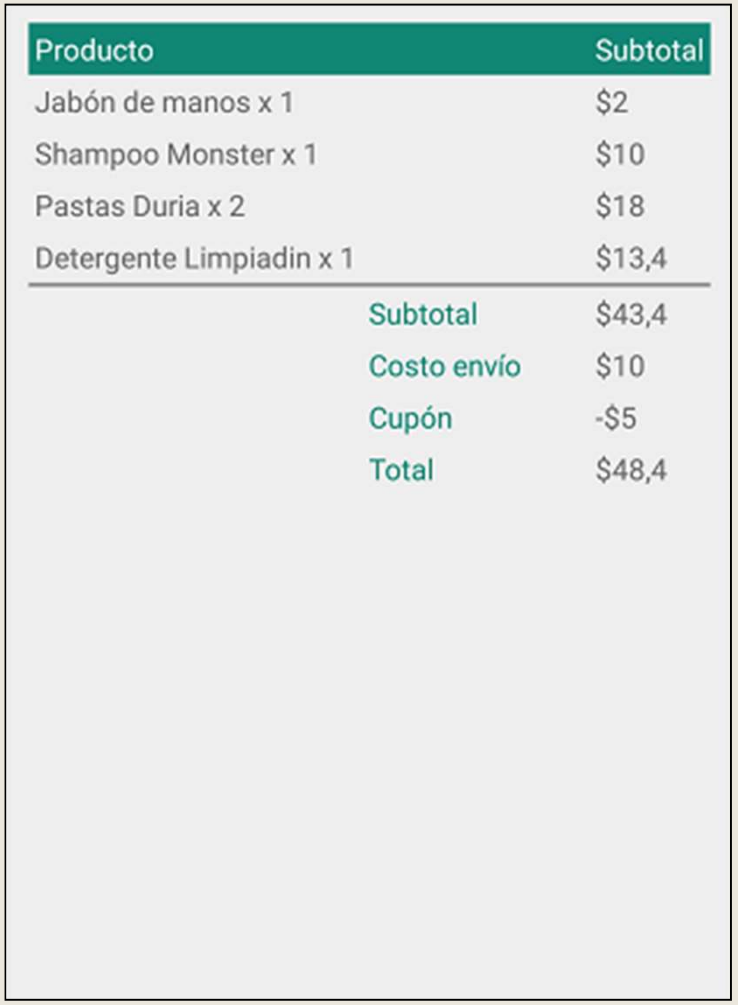
- Para crear las filas se usa el componente **TableRow** dentro del *TableLayout*. Cada celda es declarada como un view de cualquier tipo (imagen, texto, otro group view, etc.) dentro de la fila.
- Las vistas dentro de un TableRow tienen los parámetros:
  - *android:layout_column para asignar la columna a la que pertenece cada celda en su interior.*
  - *android:layout_span para combinar columnas.*
  - *Incluso puedes usar el parámetro **weight** para declarar pesos de las celdas.*
- El ancho de cada columna es definido tomando como referencia la celda más ancha. Pero se puede definir el comportamiento del ancho de las celdas con los siguientes atributos (de TableLayout):
  - **android:collapseColumns**: colapsa o invisibiliza la columna seleccionada.
  - **android:stretchColumns**: permite rellenar el espacio vacío que queda en el *TableLayout*, expandiendo la columna seleccionada.



ANDROID STUDIO (42)

TABLE LAYOUT

- Ejercicio 1
  - Dibuja la siguiente vista



Producto		Subtotal
Jabón de manos x 1		$2
Shampoo Monster x 1		$10
Pastas Duria x 2		$18
Detergente Limpiadin x 1		$13,4
	Subtotal	$43,4
	Costo envío	$10
	Cupón	-$5
	Total	$48,4

ANDROID STUDIO (43)

TABLE LAYOUT

- Solución Ejercicio 1

```xml
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:stretchColumns="1"
    tools:context=".table_layout">

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="#128675">

        <TextView
            android:layout_column="0"
            android:padding="3dp"
            android:text="Producto"
            android:textColor="@android:color/white" />

        <TextView
            android:layout_column="2"
            android:padding="3dp"
            android:text="Subtotal"
            android:textColor="@android:color/white" />
    </TableRow>
```

```xml
<TableRow
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_column="0"
        android:padding="3dp"
        android:text="Jabón de manos x 1" />

    <TextView
        android:layout_column="2"
        android:gravity="start"
        android:padding="3dp"
        android:text="$2" />
</TableRow>

<TableRow
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_column="0"
        android:padding="3dp"
        android:text="Shampoo Monster x 1" />

    <TextView
        android:layout_column="2"
        android:gravity="start"
        android:padding="3dp"
        android:text="$10" />
</TableRow>
```

```xml
<TableRow
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="0"
        android:padding="3dp"
        android:text="Pastas Duria x 2" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="2"
        android:gravity="start"
        android:padding="3dp"
        android:text="$18" />
</TableRow>
```

```xml
<TableRow
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="0"
        android:padding="3dp"
        android:text="Detergente Limpiadin x 1" />

    <TextView
        android:id="@+id/textView4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="2"
        android:gravity="start"
        android:padding="3dp"
        android:text="$13.4" />
</TableRow>
```

ANDROID STUDIO (44)

TABLE LAYOUT

- Continuación

```
<View
    android:layout_height="2dp"
    android:background="#FF909090" />

<TableRow
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_column="1"
        android:padding="3dp"
        android:text="Subtotal"
        android:textColor="#128675" />

    <TextView
        android:id="@+id/textView7"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="2"
        android:layout_gravity="start"
        android:gravity="end"
        android:padding="3dp"
        android:text="$43.4" />
</TableRow>
```

```
<TableRow
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="1"
        android:padding="3dp"
        android:text="Costo envío"
        android:textColor="#128675" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="2"
        android:layout_gravity="start"
        android:gravity="end"
        android:padding="3dp"
        android:text="$10" />
</TableRow>
```

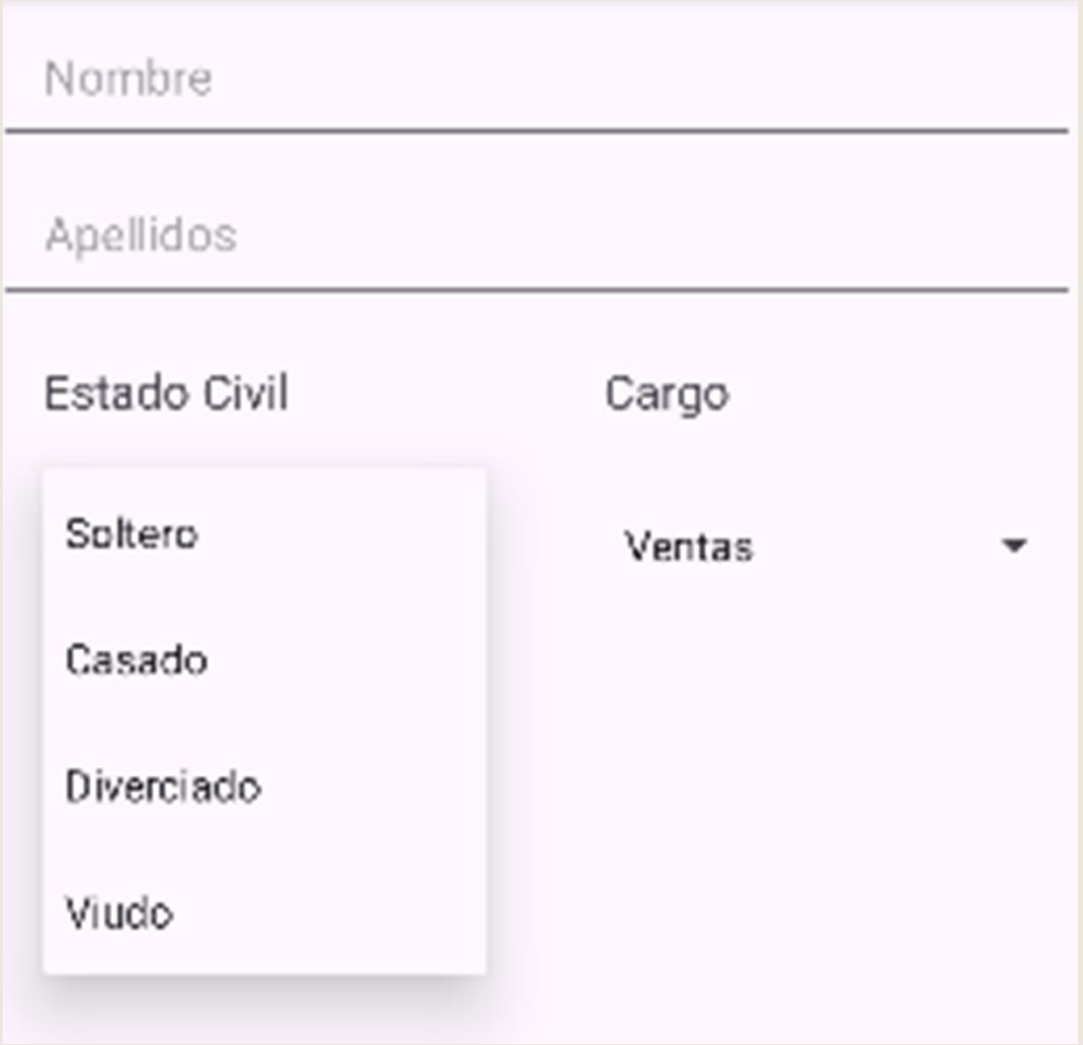
```
<TableRow
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView8"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="1"
        android:padding="3dp"
        android:text="Cupón"
        android:textColor="#128675" />

    <TextView
        android:id="@+id/textView9"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="2"
        android:layout_gravity="start"
        android:gravity="end"
        android:padding="3dp"
        android:text="-$5" />
</TableRow>
```

```
    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:id="@+id/textView5"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_column="1"
            android:padding="3dp"
            android:text="Total"
            android:textColor="#128675" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_column="2"
            android:layout_gravity="start"
            android:gravity="end"
            android:padding="3dp"
            android:text="$48.4" />
    </TableRow>
</TableLayout>
```

ANDROID STUDIO (45)

RELATIVE LAYOUT

- Ejercicio
  - *Dibuja la siguiente vista*



ANDROID STUDIO (46)

RELATIVE LAYOUT

- Solución Ejercicio

```
<string-array name="lista_estado_civil">
    <item>Soltero</item>
    <item>Casado</item>
    <item>Diverciado</item>
    <item>Viudo</item>
</string-array>
<string-array name="lista_cargo">
    <item>Ventas</item>
    <item>Informática</item>
    <item>Jardineria</item>
</string-array>
```

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".relative_layout">
```

```
    <EditText
        android:id="@+id/input_nombre"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:autofillHints="Nombre"
        android:ems="10"
        android:hint="Nombre"
        android:inputType="textPersonName"
        android:padding="18dp" />
    <EditText
        android:id="@+id/input_apellidos"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/input_nombre"
        android:layout_alignParentStart="true"
        android:autofillHints="Apellidos"
        android:ems="10"
        android:hint="Apellidos"
        android:inputType="textPersonName"
        android:padding="18dp" />
    <TextView
        android:id="@+id/texto_estado_civil«
        android:layout_width="203dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/input_apellidos"
        android:layout_alignParentStart="true"
        android:padding="18dp"
        android:text="Estado Civil"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium" />
```

```
    <Spinner
        android:id="@+id/spinner_estado_civil"
        android:layout_width="205dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/texto_estado_civil"
        android:layout_alignParentStart="true"
        android:entries="@array/lista_estado_civil"
        android:padding="18dp" />
    <TextView
        android:id="@+id/texto_cargo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/input_apellidos"
        android:layout_alignStart="@id/spinner_cargo"
        android:layout_alignParentEnd="true"
        android:padding="18dp"
        android:text="Cargo"
        android:textAppearance=
            "@style/TextAppearance.AppCompat.Medium" />
    <Spinner
        android:id="@+id/spinner_cargo"
        android:layout_width="198dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/texto_cargo"
        android:layout_alignParentEnd="true"
        android:entries="@array/lista_cargo"
        android:padding="18dp" />
</RelativeLayout>
```

ANDROID STUDIO (47)

CONSTRAINT LAYOUT

- Para definir la posición de una vista en Constraint Layout, hay que agregar una restricción horizontal y una vertical.
- Cada restricción representa una conexión o alineación con otra vista, el diseño de nivel superior o una guía invisible.
- Cada restricción define la posición de la vista a lo largo del eje vertical u horizontal.
- Cada vista debe tener un mínimo de una restricción para cada eje, aunque a menudo se necesitan más.
- Cuando se suelta una vista en el editor de diseño, ésta queda donde la dejas, incluso si no tiene restricciones. Esto solo sirve para facilitar la edición.
- Si una vista no tiene restricciones al ejecutarla en un dispositivo, se dibuja en la posición [0,0] (la esquina superior izquierda).

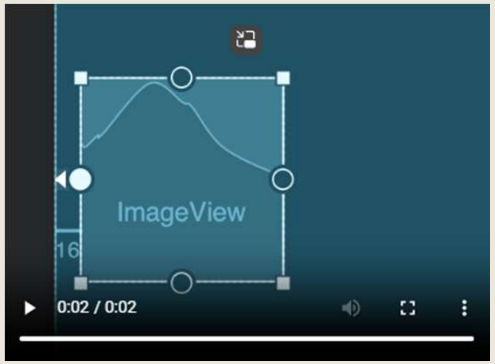
ANDROID STUDIO (48)

CONSTRAINT LAYOUT

- **Agregar o quitar una restricción.** Realizar una de las siguientes opciones:

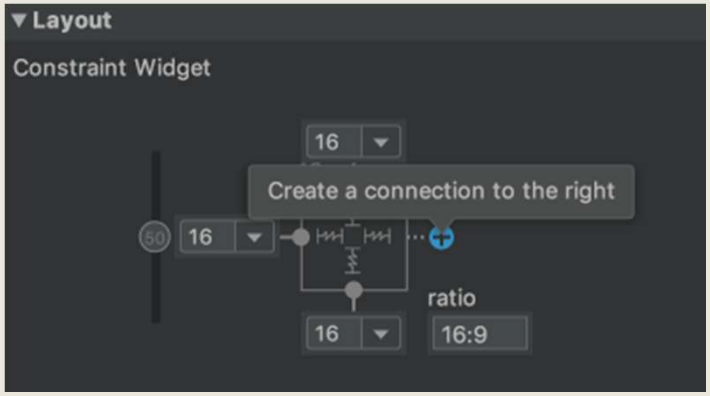
1. Haz clic en un controlador de restricción y arrástralo hasta un punto de anclaje disponible. Ese punto puede ser el borde de otra vista, el borde del diseño o una guía. A medida que arrastras el controlador de restricciones, el editor de diseño muestra posibles anclas de conexión y superposiciones azules.

   Ejemplo: e*l lado izquierdo de una vista está restringido al lado izquierdo del elemento superior.*



2. Haz clic en uno de los botones ⊕ de **Create a connection** en la sección **Layout** de la ventana **Attributes**.

   Cuando se crea la restricción, el editor le otorga un margen predeterminado para separar las dos vistas.



ANDROID STUDIO (49)

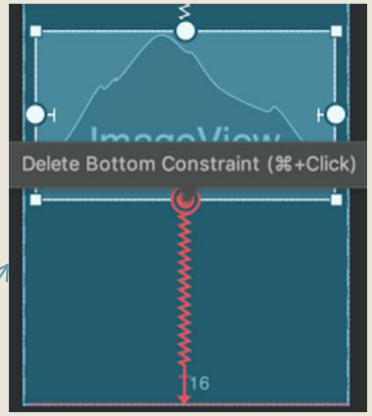
CONSTRAINT LAYOUT

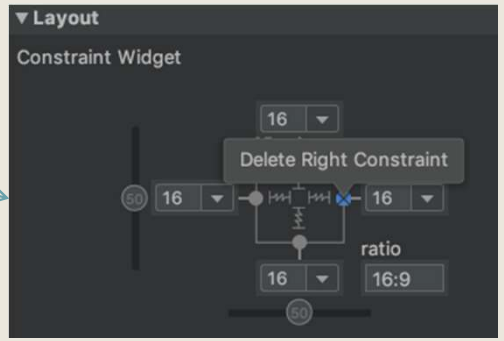
Resumen al crear restricciones:

- Cada vista debe tener al menos dos restricciones: una horizontal y otra vertical.
- Puedes crear restricciones solo entre un controlador de restricción y un punto de anclaje que compartan el mismo plano.
- Cada identificador de restricción se puede usar para una sola restricción, pero es posible crear múltiples restricciones desde diferentes vistas en el mismo punto de anclaje.

Para borrar una restricción, puedes hacerlo de una de las siguientes maneras:

1. Haz clic en una restricción para seleccionarla y, luego, en Borrar.
2. Presiona Ctrl y haz clic en un ancla de restricción. La restricción se vuelve roja para indicar que puedes hacer clic para borrarla.
3. En la sección **Layout** de la ventana **Attributes**, haz clic en un anclaje de restricción.

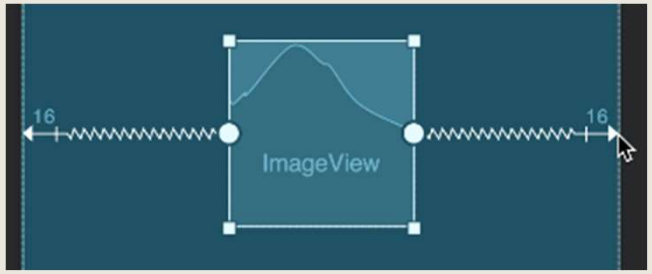




ANDROID STUDIO (50)

CONSTRAINT LAYOUT

- Si se agregan restricciones opuestas en una vista, éstas se enrollan como un resorte para indicar las fuerzas opuestas.



  - El efecto es más visible cuando el tamaño de la vista se define como "Fixed" o "Wrap content", en cuyo caso la vista se centra entre las restricciones.
  - Si se quiere que la vista se estire para ajustarse a las restricciones, hay que cambiar el tamaño a "Match constraints".
  - Si se quiere mantener el tamaño actual, pero mover la vista para que no esté centrada, hay que **ajustar el sesgo de restricciones**.

ANDROID STUDIO (51)

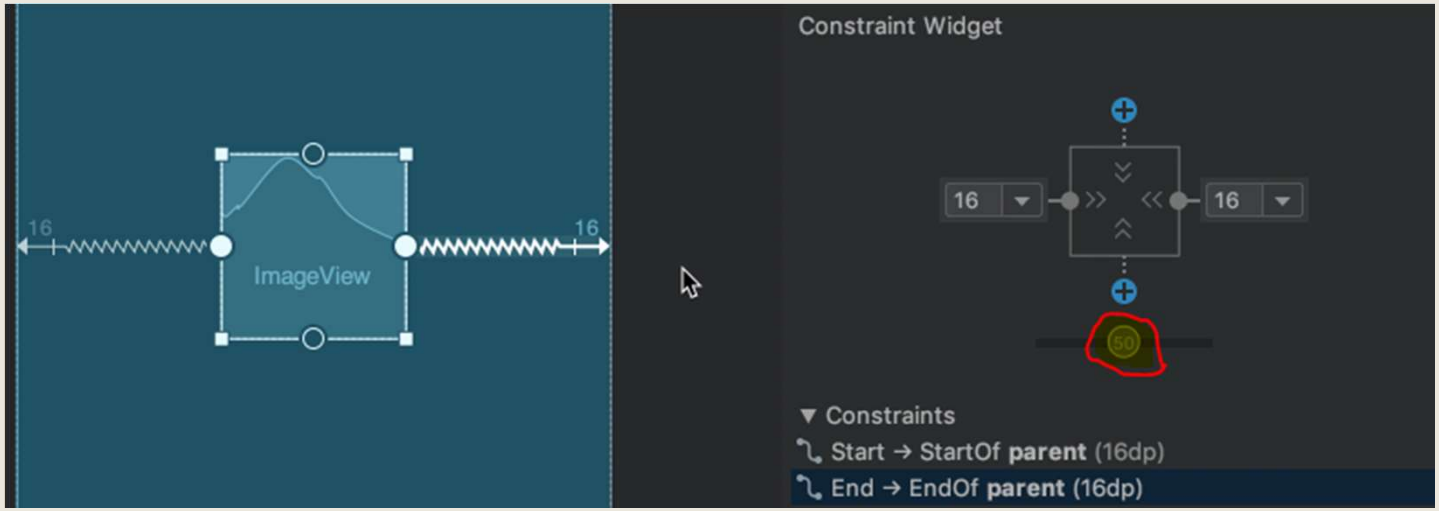
CONSTRAINT LAYOUT

- Cómo ajustar el sesgo de restricciones

Cuando se agrega una restricción a ambos lados de una vista, y el tamaño de la misma dimensión es "Fixed" o "Wrap content", la vista se centra entre las dos restricciones con un sesgo del 50% de forma predeterminada.

Para ajustar el sesgo, hay que arrastrar el control deslizante en la ventana **Attributes** o arrastrar la vista.

Si se quiere que la vista se estire para ajustarse a las restricciones, hay que cambiar el tamaño a "Match constraints".



ANDROID STUDIO (52)

CONSTRAINT LAYOUT

Se pueden usar restricciones para conseguir diferentes comportamientos de diseños:

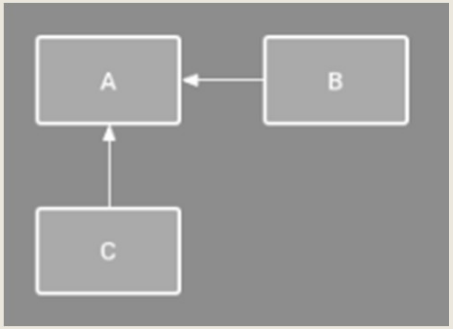
- **Posición superior**
  - Restringe el lado de una vista al borde correspondiente del diseño.
  - *Ejemplo*: una restricción horizontal del elemento superior

    El lado izquierdo de la vista está conectado al borde izquierdo del diseño superior.

    Se puede definir la distancia desde el borde con margen.

- **Posición de orden**
  - Define el orden de aparición de dos vistas, ya sea de manera horizontal o vertical.
  - *Ejemplo*: restricción horizontal y vertical

    La vista B está restringida a mostrarse siempre a la derecha de A, y C está restringida a aparecer debajo de A. Sin embargo, estas restricciones no implican alineación, por lo que B aún puede moverse hacia arriba y hacia abajo.

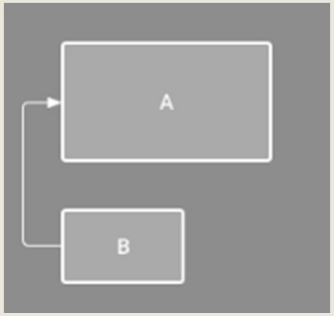


ANDROID STUDIO (53)

CONSTRAINT LAYOUT

- Alineación
  - Alinea el borde de una vista con el mismo borde de otra vista.
  - *Ejemplo 1*: una restricción de alineación horizontal

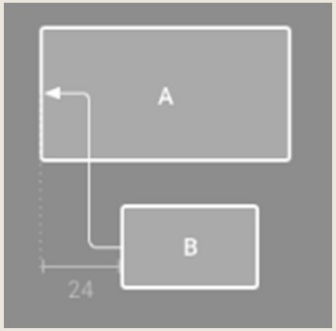
    El lado izquierdo de B está alineado con el lado izquierdo de A. Si se desea alinear los centros de las vistas, hay que crear una restricción en ambos lados. Se puede compensar la alineación arrastrando la vista desde la restricción hacia adentro.



  - *Ejemplo 2*: una restricción de alineación horizontal de desplazamiento

    Se muestra B con una alineación de desplazamiento de 24 dp.

    El desplazamiento está definido por el margen de la vista restringida.



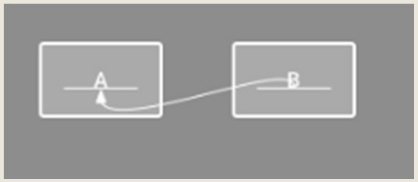
También se pueden seleccionar todas las vistas que se quieren alinear y, luego, hacer clic en **Align** en la barra de herramientas para seleccionar el tipo de alineación.

ANDROID STUDIO (54)

CONSTRAINT LAYOUT

- **Alineación de línea de base**
  - Alinea la línea de base de texto de una vista con la línea de base de texto de otra.
  - Ejemplo: una restricción de alineación de línea de base

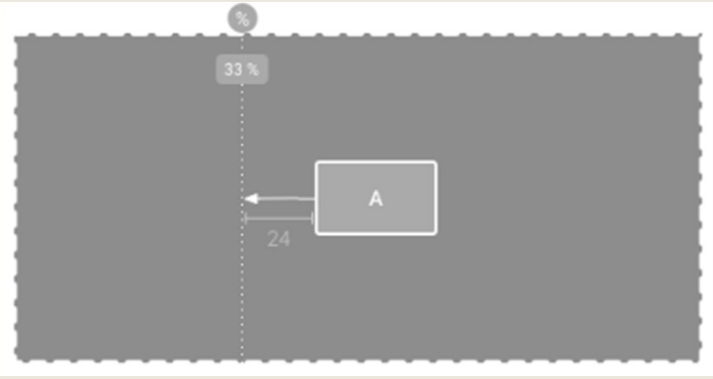
    La primera línea de B está alineada con el texto de A.



Para crear una restricción de línea de base, haz clic con el botón derecho en la vista de texto que deseas restringir y, luego, haz clic en **Show Baseline**. Luego, haz clic en la línea de base de texto y arrástrala hasta otra línea de base.

- **Cómo aplicar una restricción a una guía**

Puedes agregar una guía vertical u horizontal que te permita restringir las vistas y que sea invisible para los usuarios de tu app. Puedes colocar la guía dentro del diseño según las unidades de dp o un porcentaje en relación con el borde del diseño.
Para crear una guía, haz clic en **Lineamientos** en la barra de herramientas y, luego, en **Agregar guía vertical** o **Agregar guía horizontal**.



ANDROID STUDIO (55)

CONSTRAINT LAYOUT

- **Cómo aplicar una restricción a una guía**

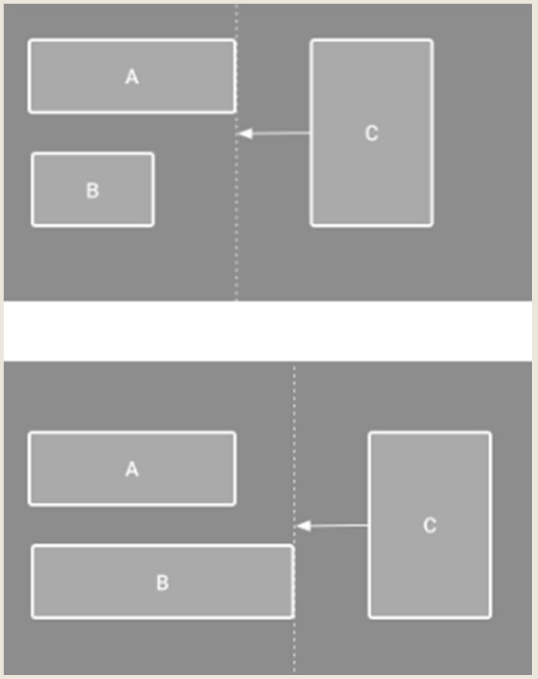
Arrastra la línea punteada para cambiar la posición y haz clic en el círculo que está en el borde de la guía para activar o desactivar el modo de medición.

- **Cómo aplicar una restricción a una barrera**

Al igual que con las guías, una barrera es una línea invisible respecto de la cual puedes restringir vistas, excepto que una barrera no define su propia posición. En su lugar, la posición de la barrera se mueve en función de la posición de las vistas que contiene. Esto es útil si deseas restringir una vista a un conjunto de vistas en lugar de a una vista específica.

Ejemplo: La vista C está restringida a una barrera, que se desplaza según la posición y el tamaño de las vistas A y B.

La vista C está limitada al lado derecho de una barrera. La barrera se establece en el "extremo" (o el lado derecho, en un diseño de izquierda a derecha) de la vista A y la vista B. La barrera se mueve dependiendo de si el lado derecho de la vista A o de la vista B es el lado más a la derecha.



ANDROID STUDIO (56)

CONSTRAINT LAYOUT

- Cómo aplicar una restricción a una barrera

Al igual que con las guías, una barrera es una línea invisible respecto de la cual puedes restringir vistas, excepto que una barrera no define su propia posición. En su lugar, la posición de la barrera se mueve en función de la posición de las vistas que contiene. Esto es útil si deseas restringir una vista a un conjunto de vistas en lugar de a una vista específica.

Pasos para crear una barrera:

1. Haz clic en **Lineamientos** en la barra de herramientas y, luego, en **Agregar barrera vertical** o **Agregar barrera horizontal**.
2. En la ventana **Component Tree**, selecciona las vistas que desees dentro de la barrera y arrástralas al componente de la barrera.
3. Selecciona la barrera en **Component Tree**, abre la ventana **Attributes** y, luego, configura **barrierDirection**.

Ahora puedes crear una restricción desde otra vista hasta la barrera.

ANDROID STUDIO (57)

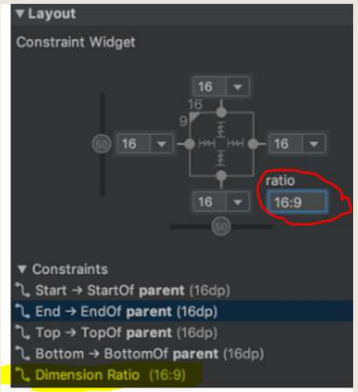
CONSTRAINT LAYOUT

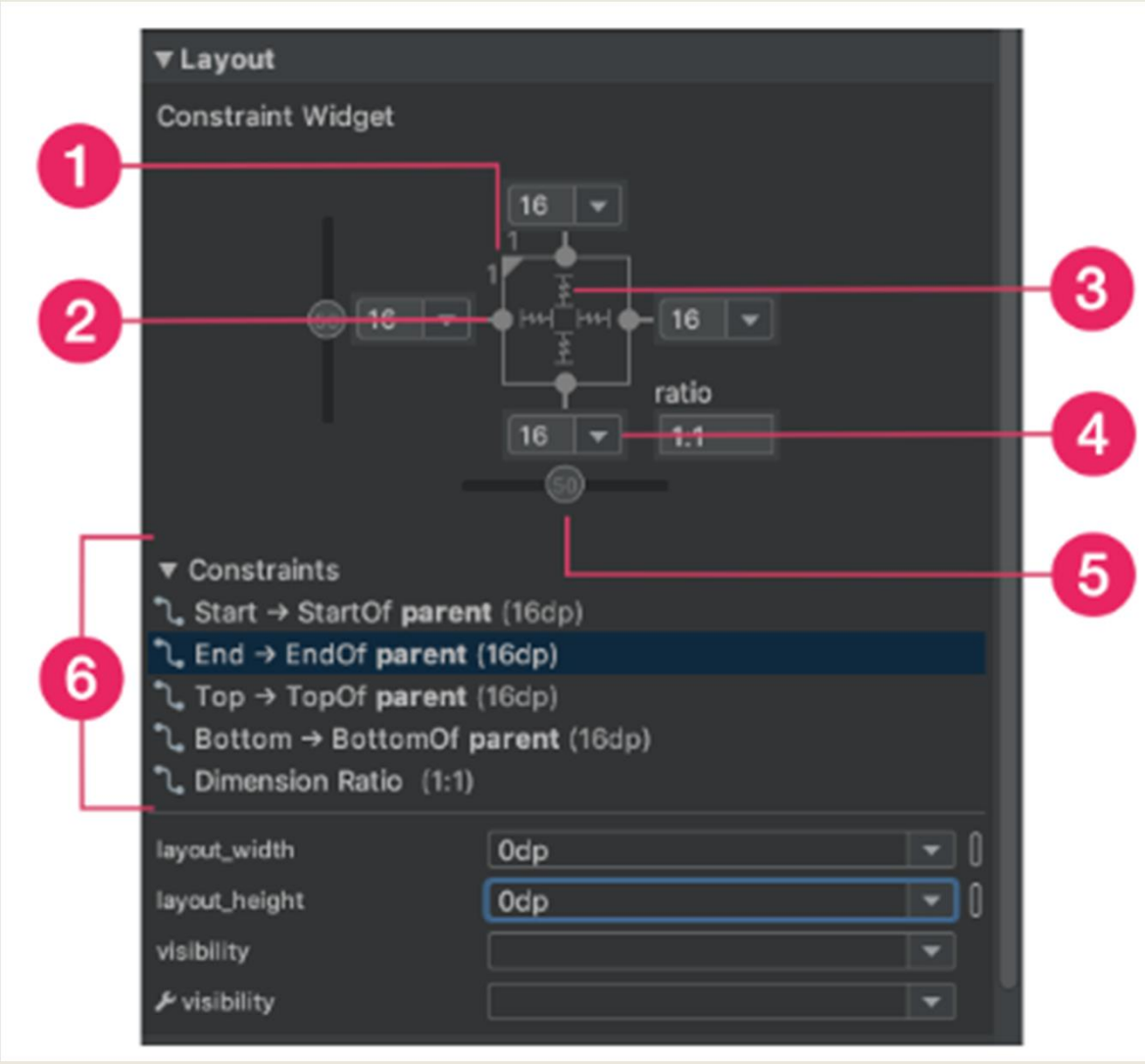
- Cómo ajustar el tamaño de la vista

Esto solo está disponible para vistas en un diseño de restricción.

1 Proporción de tamaño

Se puede establecer el tamaño de la vista en una proporción, como 16:9, si al menos una de las dimensiones de la vista está configurada en "Match Constraints" (0dp). Luego, simplemente hay que ingresar la proporción **width:height** en la entrada que aparece.





ANDROID STUDIO (58)

CONSTRAINT LAYOUT

- Cómo ajustar el tamaño de la vista

2 **Borrar restricciones:** ya visto anteriormente

3 **Modo de altura o ancho**

Haciendo clic en los símbolos se puede cambiar la forma en que se calcula la altura y el ancho y alternar entre las opciones de configuración. Estos símbolos representan el modo de tamaño de la siguiente manera:

- **Fixed:** especifica una dimensión específica en el siguiente cuadro de texto o cambia el tamaño de la vista en el editor.
- **Wrap Content:** la vista se expande solo lo necesario para ajustarse a su contenido.
    - **layout_constrainedWidth:** establece este valor en ***true*** para permitir que la dimensión horizontal cambie para respetar las restricciones. De forma predeterminada, un widget configurado en *WRAP_CONTENT* no está limitado por restricciones.

ANDROID STUDIO (59)

CONSTRAINT LAYOUT

- Cómo ajustar el tamaño de la vista

- **Match Constraints**: la vista se expande tanto como sea posible para cumplir con las restricciones de cada lado, después de considerar los márgenes de la vista. Sin embargo, se puede modificar ese comportamiento con los siguientes atributos y valores. Estos atributos solo tienen efecto cuando se establece que el ancho de la vista coincida con las restricciones:
  - **layout_constraintWidth_min:** toma una dimensión en ***dp*** para el ancho **mínimo** de la vista.
  - **layout_constraintWidth_max:** toma una dimensión en ***dp*** para el ancho **máximo** de la vista.

  Si la dimensión determinada tiene una sola restricción, la vista se expande para ajustarse a su contenido. El uso de este modo en el ancho o la altura también permite **establecer una proporción de tamaño**.

NOTA: *No se puede usar match_parent para ninguna vista en un ConstraintLayout. En su lugar, hay que usar "match constraints" (0dp).*

ANDROID STUDIO (60)

CONSTRAINT LAYOUT

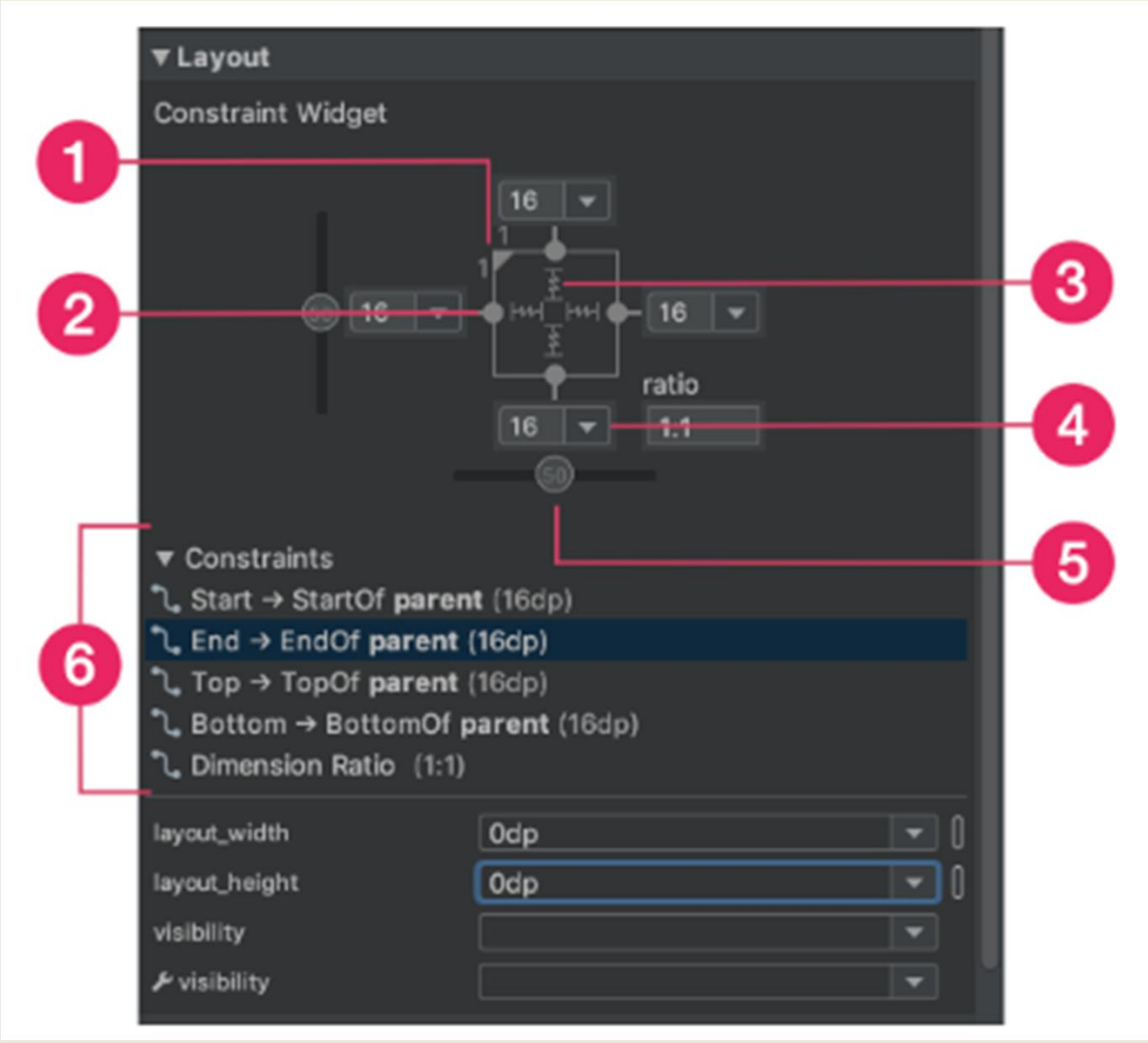
- Cómo ajustar el tamaño de la vista

4 Márgenes

Todos los márgenes son factores de 8 dp, lo que ayudará a alinear las vistas con las recomendaciones de cuadrícula cuadrada de 8 dp de Material Design.

5 Sesgo de restricciones

6 Lista de restricciones



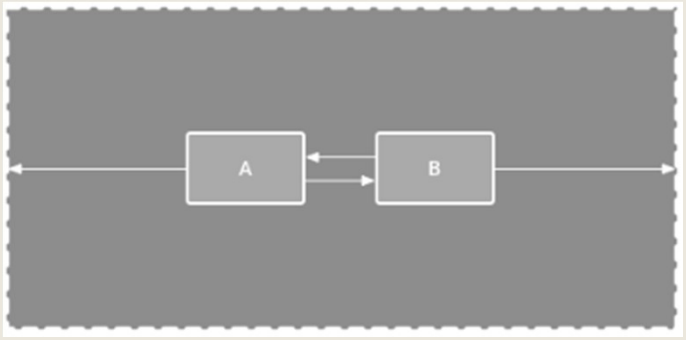
ANDROID STUDIO (61)

CONSTRAINT LAYOUT

- Cómo controlar grupos lineales con una cadena

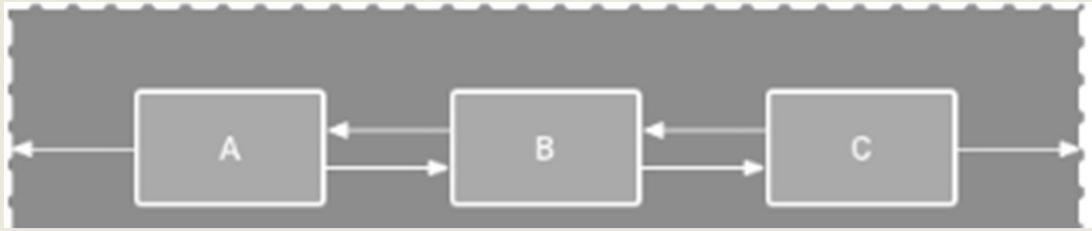
Una cadena es un grupo de vistas que están vinculadas entre sí con restricciones de posición bidireccionales. Dentro de una cadena, se pueden distribuir las vistas de manera horizontal o vertical.

Ejemplo: cadena horizontal con dos vistas



Las cadenas se pueden diseñar de una de las siguientes maneras:

1. **Spread:** las vistas se distribuyen uniformemente después de restar los márgenes. Es el valor predeterminado.



ANDROID STUDIO (62)

CONSTRAINT LAYOUT

- Cómo controlar grupos lineales con una cadena

2. **Distribuido en el interior:** la primera y la última vista se fijan a las restricciones de cada extremo de la cadena y el resto se distribuye de manera uniforme.

3. **Ponderada:** cuando la cadena está configurada en **spread** o **spread inside**, puedes completar el espacio restante configurando una o más vistas para que "coincidan con las restricciones" (0dp).

De forma predeterminada, el espacio se distribuye de manera uniforme entre cada vista configurada para "coincidir con las restricciones", pero puedes asignar un peso de importancia a cada vista con los atributos **layout_constraintHorizontal_weight** y **layout_constraintVertical_weight.** Funciona de la misma manera que *layout_weight* en un diseño lineal: la vista con el valor de ponderación más alto obtiene la mayor cantidad de espacio, y las vistas que tienen el mismo peso obtienen la misma cantidad de espacio.

ANDROID STUDIO (63)

CONSTRAINT LAYOUT

- Cómo controlar grupos lineales con una cadena

4. **Packed:** las vistas se agrupan después de restar los márgenes. Puedes ajustar el sesgo de toda la cadena (izquierda/derecha o arriba/abajo) cambiando el sesgo de la vista de extremo de la cadena.

La vista de "extremo" de la cadena (la vista más a la izquierda en una cadena horizontal [en un diseño de izquierda a derecha] y la vista más arriba en una cadena vertical) define el estilo de la cadena en XML.

Puedes seleccionar cualquier vista en la cadena y hacer clic en el botón de la cadena que aparece debajo de la vista para alternar entre **Spread**, **Spread inside** y **Packed**.

Creación de una cadena

1. Selecciona todas las vistas que se incluirán en la cadena.

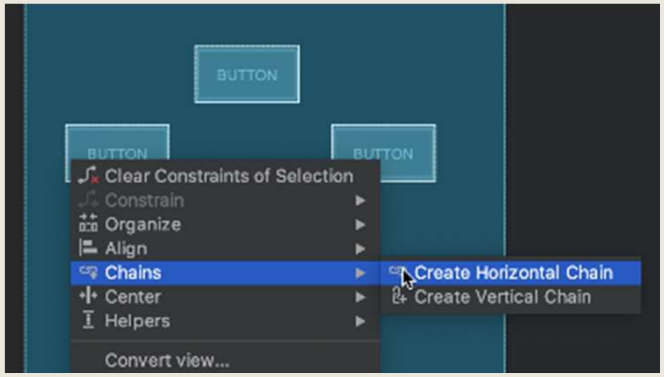
ANDROID STUDIO (64)

CONSTRAINT LAYOUT

- Cómo controlar grupos lineales con una cadena

Creación de una cadea

2. Clic con el botón derecho en una de las vistas
3. Selecciona Cadena
4. Selecciona Centrar Horizontalmente o Centrar verticalmente



Aspectos a tener en cuenta cuando se usan cadenas:

- Una vista puede formar parte de una cadena horizontal y vertical, por lo que puedes crear diseños de cuadrícula flexibles.
- La cadena funciona correctamente solo si cada extremo está restringido a otro objeto en el mismo eje.
- Aunque la orientación de una cadena es vertical u horizontal, cuando se usa no se alinean las vistas en esa dirección. Para lograr la posición adecuada de cada vista en la cadena, hay que incluir otras restricciones, como las restricciones de alineación.