Ejercicio 3:

-ConstraintLayout:

Nos permitirá simplificar las interfaces en anidamiento, para hacerlas lo más complejas posibles a nivel de diseño. Es similar a RelativeLayout en que todas las vistas se establecen según las relaciones entre las vistas de hermanos y el diseño de los padres, pero es más flexible que RelativeLayout y más fácil de usar con el Editor de diseño de Android Studio.
Con un estilo muy visual, podremos desde Android Studio gestionar todas las relaciones que establezcamos, de un modo muy sencillo, con su herramienta de drag-and-drop, en lugar de necesitar utilizar el fichero XML.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
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:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/appe"
      android:textSize="30sp"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintRight_toRightOf="parent"
      app:layout_constraintTop_toTopOf="parent" />
```

</androidx.constraintlayout.widget.ConstraintLayout>

-RelativeLayout:

Los relative layout son uno de los tipos mas comunes de layouts utilizados por los diseñadores de la interfaz de usuario de Android. Los relative layouts pueden ser definidos dentro de los recursos layout XML o por medio de programación en el código de la aplicación de Java. Los relative layouts funcionan como da a entender su nombre: organiza los controles relativos unos a otros, o el mismo control parent.

Significa que los controles child, tales como ImageView, TextView y los controles Button, pueden ser ubicados arriba, abajo, a la izquierda o la derecha de unos u otros. Los controles child, además, pueden ser ubicados en relación al parent (que es el contenedor del relative layout) incluyendo los controles de ubicación alineados arriba, abajo, a la izquierda o derecha del layout.

El control de ubicación child del relative layout está definido usando reglas. Estas reglas definen como se mostraran los controles dentro del relative layout.

Digamos que queremos diseñar una pantalla con un control EditText y un Button. Queremos que el Button esté a la derecha del control EditText. Por lo tanto, podríamos definir un relative layout con dos controles child: el EditText y el Button. El control EditText podría tener una regla que dice: alinear éste control en el lado izquierdo del control parent (es decir, el layout) y a la izquierda de un segundo control Button. Mientras tanto, el control Button podría tener un regla que dice: alinear este control al lado derecho del control parent (es decir, el layout).

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
```

```
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent">
    <EditText
        android:id="@+id/EditText01"
        android:hint="Enter some text..."
        android:layout_alignParentLeft="true"
        android:layout_width="fill_parent"
        android:layout_toLeftOf="@+id/Button01"
        android:layout_height="wrap_content"></EditText>
    <Button
        android:id="@+id/Button01"
        android:text="Press Here!"
        android:layout_width="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_height="wrap_content"></Button>
</RelativeLayout>
```

LinearLayout

Es uno de los diseños básicos con que contamos en Android para crear las interfaces de usuario de nuestras aplicaciones. A diferencia de FrameLayout, que coloca cada vista que se incluye en él sobre la anterior, LinearLayout coloca las vistas una a continuación de otra, sin superponerlas nunca. Es decir, las alinea.
Esta alineación puede ser vertical u horizontal. O sea, que LinearLayout sirve para colocar vistas en una misma fila o columna, pero no ambas cosas a la vez.
Las vistas se colocan en el mismo orden en se agregan al diseño o en el que aparecen en el archivo XML.
De forma predeterminada, LinearLayout tiende a establecer el mismo tamaño para cada vista que contiene, repartiendo el espacio disponible de forma equitativa entre todas ellas. Sin embargo, algunas propiedades

de las vistas y ciertos parámetros que el propio LinearLayout añade permiten variar este comportamiento.

Algunas de sus propiedades:

- android:orientation (setOrientation): la principal propiedad de LinearLayout. Determina si las vistas se alinean verticalmente u horizontalmente. Puede tomar dos valores: horizontal o vertical. De forma predeterminada, la orientación es vertical.
- android:baselineAligned (setBaselineAligned): de forma predeterminada, LinearLayout alinea las vistas que contiene en función de su línea base (línea invisible sobre la que apoya el texto). Esto permite que un TextView y un Button queden alineados verticalmente de forma natural en un diseño horizontal. Sin embargo, este tipo de alineación puede dar problemas en algunos casos. Establecer esta propiedad a false desactiva este alineamiento.

- android:gravity (setGravity): permite definir la gravedad del propio LinearLayout, o sea, cómo se alinea todo su contenido. Afecta al conjunto de las vistas que se incluyen en él, pero no al contenido de las propias vistas. Los valores que puede tomar son los habituales para la gravedad.

-FrameLayout:

El FrameLayout, el diseño más sencillo de todos los disponibles, pero también el más eficiente.

FrameLayout no realiza ninguna distribución de las vistas, simplemente las coloca unas encima de otras. Esto le evita tener que relacionar los tamaños de unas vistas con los de las demás, por lo que se ahorra recorridos del árbol de vistas, tardando menos en mostrar su contenido.

android:layout_gravity no es exclusivo de FrameLayout ya que otros diseños básicos lo permiten. Los valores disponibles para el parámetro y el resultado obtenido con cada uno de ellos son comunes para todos estos diseños.

La gravedad permite indicar cómo se debe alinear la vista sobre la que se aplica, tanto horizontal como verticalmente, en relación a los límites del área disponible para el contenido del FrameLayout. Este área incluye todo el espacio que el FrameLayout ocupa en la actividad menos el padding que se haya establecido.
al atributo android:layout_gravity se le asigna una cadena de texto que contiene uno o más de estos valores. Si hay más de uno se separan mediante |. Para cada uno de estos valores existe una constante de igual nombre que se puede utilizar para asignar la gravedad mediante código Java. De forma similar, si en Java usa más de una constante, se agregan mediante el operador lógico |:

- top y bottom: alinean el borde superior/inferior de la vista con el del área útil del diseño, sin alterar el tamaño de la vista.
- left y right: alinean el borde izquierdo/derecho de la vista con el del área útil del diseño, sin alterar el tamaño de la vista.
- center_horizontal y center_vertical: centran la vista horizontal o verticalmente en el área útil del diseño sin alterar su tamaño.
- center: centra la vista vertical y horizontalmente, de forma simultánea, en el área útil del diseño sin alterar su tamaño.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Sin gravedad"
      android:textSize="23sp"
      android:background="#080" />
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="right"
      android:background="#080"
      android:textSize="23sp"
      android:layout_gravity="right" />
   <TextView
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="bottom"
        android:background="#080"
        android:textSize="23sp"
        android:layout_gravity="bottom" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="center_vertical"
        android:background="#080"
        android:textSize="23sp"
        android:layout_gravity="center_vertical" />

    </FrameLayout>
```

-TableLayout:

Con TableLayout añadimos una nueva dimensión, pudiendo disponer las vistas tanto verticalmente como horizontalmente, en forma de tabla pero es bastante menos potente que GridLayout.

Lo primero que es necesario destacar de TableLayout es que no tiene, estrictamente hablando, una estructura de tabla. En realidad, TableLayout es una especialización de LinearLayout. En concreto, de un LinearLayout vertical. Aunque con un comportamiento particularizado.
Al igual que en un LinearLayout, en un TableLayout se pueden incluir todo tipo de vistas. Sin embargo, hay una vista específica que sólo se puede utilizar en TableLayout y que es la que aporta la funcionalidad necesaria para crear la tabla: TableRow.

TableRow es, a su vez, otra especialización de LinearLayout. Esta vez de un LinearLayout horizontal. Así que resulta evidente que es en realidad un grupo de LinearLayout horizontales dentro de un LinearLayout vertical.

Esta organización tampoco resulta tan extraña. De hecho, es muy similar a la de las tablas en HTML, con las que este diseño comparte otras características.
Resulta evidente que cada TableRow representa una fila de la tabla y que las vistas que contengan harán las veces de columnas. En concreto, cada vista que se añade a un TableRow va a parar a una columna diferente.

Lo que diferencia a TableLayout de una estructura similar creada con varios LinearLayout es el tratamiento global que le da a las vistas que se incluyen en todos los TableRow.
Propiedades del diseño

TableLayout hereda directamente de LinearLayout, por lo que dispone de todas sus propiedades. Aunque algunas, como android:orientation, carecen de efecto si se intentan usar aquí.
Por su parte, TableLayout tiene algunas propiedades exclusivas:

- android:collapseColumns (setColumnCollapsed): permite ocultar columnas, como si no estuvieran en el diseño. El espacio que dejan lo ocupan el resto de columnas.

- android:shrinkColumns (setColumnShrinkable): permite marcar una o más columnas como encogibles, de forma que su anchura se pueda reducir para adaptarse al tamaño del contenedor del TableLayout.

- android:stretchColumns (setColumnStretchable): esta propiedad viene a ser la opuesta de la anterior, dado que permite que una columna se expanda aumentando su anchura hasta que todo el contenido del TableLayout ocupe la totalidad de la anchura de su contenedor.

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/Tabla"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:stretchColumns="1" >

   <TableRow
      android:id="@+id/Cabecera"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content" >

      <TextView
         android:id="@+id/ColumnaAnio"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:padding="5px"
         android:text="Año"
         android:textColor="#005500"
         android:textSize="26sp" />

      <TextView
         android:id="@+id/ColumnaCiudad"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_gravity="center_horizontal"
         android:padding="5px"
         android:text="Ciudad"
         android:textColor="#005500"
         android:textSize="26sp" />

      <TextView
         android:id="@+id/ColumnaOro"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:padding="5px"
         android:text="Oro"
         android:textColor="#005500"
         android:textSize="26sp" />

      <TextView
```

```
        android:id="@+id/ColumnaPlata"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="5px"
        android:text="Plata"
        android:textColor="#005500"
        android:textSize="26sp" />

    <TextView
        android:id="@+id/ColumnaBronce"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="5px"
        android:text="Bronce"
        android:textColor="#005500"
        android:textSize="26sp" />
</TableRow>

<TableRow
    android:id="@+id/SeparadorCabecera"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >

    <FrameLayout
        android:id="@+id/LineaCabecera"
        android:layout_width="fill_parent"
        android:layout_height="2px"
        android:layout_span="6"
        android:background="#FFFFFF" >
    </FrameLayout>
</TableRow>

<TableRow
    android:id="@+id/Fila1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >

    <TextView
        android:id="@+id/Anio1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
```

```
        android:text="1900"
        android:textSize="20sp" />

    <TextView
        android:id="@+id/Cuidad1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="París"
        android:textSize="20sp" />

    <TextView
        android:id="@+id/Oro1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="1"
        android:textSize="20sp" />
</TableRow>

<TableRow
    android:id="@+id/Fila2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >

    <TextView
        android:id="@+id/Anio2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="1976"
        android:textSize="20sp" />

    <TextView
        android:id="@+id/Cuidad2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="Montreal"
        android:textSize="20sp" />

    <TextView
```

```
        android:id="@+id/Plata2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="3"
        android:gravity="center"
        android:text="2"
        android:textSize="20sp" />
</TableRow>



<TableRow
    android:id="@+id/SeparadorTotales"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >

    <FrameLayout
        android:id="@+id/LineaTotales"
        android:layout_width="fill_parent"
        android:layout_height="2px"
        android:layout_span="5"
        android:background="#FFFFFF" >
    </FrameLayout>
</TableRow>

<TableRow
    android:id="@+id/Totales"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" >

    <TextView
        android:id="@+id/TextoTotal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_span="2"
        android:gravity="right"
        android:text="Total"
        android:textColor="#0000CC"
        android:textSize="22sp" />

    <TextView
        android:id="@+id/OroTotal"
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:text="17"
            android:textSize="22sp" />

        <TextView
            android:id="@+id/PlataTotal"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:text="19"
            android:textSize="22sp" />

        <TextView
            android:id="@+id/BronceTotal"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:text="6"
            android:textSize="22sp" />
    </TableRow>

 </TableLayout>
```

-GridLayout:

El grid se compone de un conjunto de líneas infinitamente delgadas que separan el área de visualización en celdas. En toda la API, las líneas de cuadrícula se referencian mediante índices de cuadrícula. Una cuadrícula con N columnas tiene N + 1 índices de cuadrícula que van de 0 a N inclusive. Independientemente de cómo se configure GridLayout, el índice de cuadrícula 0 se fija al borde inicial del contenedor y el índice de cuadrícula N se fija a su borde posterior (después de tener en cuenta el padding).

GridLayout te brinda la posibilidad de crear diseños basados en cuadrículas con una sola vista raíz.

Los hijos ocupan una o más celdas contiguas, según lo definido por sus parámetros de disposición rowSpec y columnSpec. Cada especificación define el conjunto de filas o columnas que se van a ocupar; Y cómo los hijos deben ser alineados dentro del grupo resultante de células.

Aunque las celdas normalmente no se solapan en un GridLayout, GridLayout no impide que los hijos estén definidos para ocupar la misma celda o grupo de celdas. En este caso, sin embargo, no hay garantía de que los hijos no se superpongan una vez finalizada la operación de disposición.

android:alignmentMode – Cuando se establece en alignMargins, hace que la alineación tenga lugar entre el límite exterior de una vista, según lo definido por sus márgenes.

android:columnCount – El número máximo de columnas para crear al posicionar automáticamente a los hijos.

android:columnOrderPreserved –
Cuando se establece en verdadero, obliga a que los límites de la columna aparezcan en el mismo orden que los índices de columna.

android:orientation –
La propiedad de orientación no se utiliza durante el diseño.

android:rowCount – El número máximo de filas para crear al posicionar automáticamente a los hijos.

android:rowOrderPreserved –Cuando se establece en verdadero, obliga a los límites de fila a aparecer en el mismo orden que los índices de fila.

android:useDefaultMargins – Cuando se establece en verdadero, le dice a GridLayout que use márgenes predeterminados cuando no se especifique ninguno en los parámetros de diseño de una vista.

```
<GridLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:rowCount="2"
    android:columnCount="3"
    android:orientation="horizontal" >

        <TextView android:text="Celda 1.1" />
        <TextView android:text="Celda 1.2" />
        <TextView android:text="Celda 1.3" />

        <TextView android:text="Celda 2.1" />
        <TextView android:text="Celda 2.2" />
        <TextView android:text="Celda 2.3" />

        <TextView android:text="Celda 3.1"
              android:layout_columnSpan="2" />

        <TextView android:text="Celda 3.2" />

</GridLayout>
```

Ejercicio 4:

AndroidManifest.xml:

Este fichero describe la aplicación Android. Se define su nombre, paquete, icono, estilos, etc. Se indican las actividades, las intenciones, los servicios y los proveedores de contenido de la aplicación. También

se declaran los permisos que requerirá la aplicación. Se indica el paquete Java, la versión de la aplicación, etc.

java:

Carpeta que contiene el código fuente de la aplicación. Los ficheros Java se almacenan en carpetas según el nombre de su paquete.

MainActivity:

Clase con el código de la actividad inicial.

ExampleInstrumentTest:

Clase pensada para insertar código de testeo de la aplicación.

ExampleUnitTest:

Clase para insertar test unitarios sobre otras aplicaciones.

res:

Carpeta que contiene los recursos usados por la aplicación.

drawable:

En esta carpeta se almacenan los ficheros de imágenes (JPG o PNG) y descriptores de imágenes en XML.

mipmap:

En una carpeta guardaremos el icono de la aplicación.

El fichero ic_launcher_round.png es similar pero para cuando se quieren usar iconos redondos.

layout:

Contiene ficheros XML con vistas de la aplicación. Las vistas nos permitirán configurar las diferentes pantallas que compondrán la interfaz de usuario de la aplicación. Se utiliza un formato similar al HTML usado para diseñar páginas web.

menú:

Ficheros XML con los menús de cada actividad. En el proyecto no hay ningún menú por lo que no se muestra esta carpeta.

values:

También utilizaremos ficheros XML para indicar valores usados en la aplicación, de esta manera podremos cambiarlos desde estos ficheros sin necesidad de ir al código fuente. En colors.xml se definen los tres colores primarios de la aplicación. En dimens.xml, se pueden definir dimensiones como el margen por defecto o el ancho de los botones. En el fichero strings.xml, tendrás que definir todas las cadenas de caracteres de tu aplicación. Creando recursos alternativos resultará muy sencillo traducir una aplicación a otro idioma. Finalmente en styles.xml, podrás definir los estilos y temas de tu aplicación.

anim:

Contiene ficheros XML con animaciones de vistas (Tween).

animator:

Contiene ficheros XML con animaciones de propiedades.

xml:

Otros ficheros XML requeridos por la aplicación.

raw:

Ficheros adicionales que no se encuentran en formato XML.

Gradle Scripts:

En esta carpeta se almacenan una serie de ficheros Gradle que permiten compilar y construir la aplicación. Algunos hacen referencia al módulo app y el resto son para configurar todo el proyecto.