3. Describir los siguientes tipos de layout en Android, sus propiedades, uso recomendado, y como se disponen los componentes en su interior. Incluir ejemplos tanto de código como de casos de uso:

ConstraintLayout:

Constraintlayout es un “android.view.ViewGroup” que le permite posicionar y dimensionar widgets de manera flexible. Este layout, similar al RelativeLayout nos permitirá establecer relaciones entre todos los elementos y la propia vista padre, permitiendo así ser mucho más flexible que los demás.

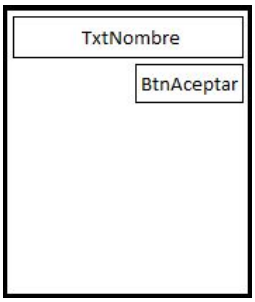
RelativeLayout :

Es un grupo de vistas que muestra vistas secundarias en posiciones relativas. La posición de cada vista se especifica en relación con otros elementos dentro del Layout (como a la izquierda o debajo vista) o en relaciones relativas al padre o área del mismo RelativeLayout (parent RelativeLayout).

```
<RelativeLayout

        xmlns:android="http://schemas.android.com/apk/res/android"

        android:layout_width="match_parent"

        android:layout_height="match_parent" >


        <EditText android:id="@+id/TxtNombre"

              android:layout_width="match_parent"

              android:layout_height="wrap_content"

              android:inputType="text" />


        <Button android:id="@+id/BtnAceptar"

            android:layout_width="wrap_content"

            android:layout_height="wrap_content"

            android:layout_below="@id/TxtNombre"

            android:layout_alignParentRight="true" />
```

```
</RelativeLayout>
```

En el ejemplo, el botón BtnAceptar se colocará debajo del cuadro de texto TxtNombre (android:layout_below=»@id/TxtNombre») y alineado a la derecha del layout padre (android:layout_alignParentRight=»true»), Quedaría algo así:



LinearLayout:

LinearLayout es un grupo de vista que alinea todos los campos secundarios en una única dirección, de manera vertical u horizontal. Puedes especificar la dirección del diseño con el atributo android:orientation.

Un LinearLayout respeta los márgenes entre los campos secundarios y la gravedad (alineación a la derecha, centrada o a la izquierda) de cada campo secundario.

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <EditText android:id="@+id/TxtNombre"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <Button android:id="@+id/BtnAceptar"
        android:layout_width="wrap_content"
        android:layout_height="match_parent" />

</LinearLayout>
```

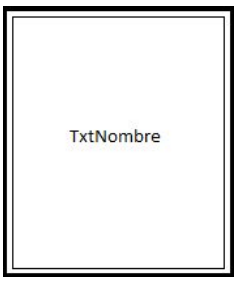
Con el código anterior conseguiríamos un layout como el siguiente:

FrameLayout:

Éste es el más simple de todos los layouts de Android. Un FrameLayout coloca todos sus controles hijos alineados con su esquina superior izquierda, de forma que cada control quedará oculto por el control siguiente (a menos que éste último tenga transparencia). Por ello, suele utilizarse para mostrar un único control en su interior, a modo de contenedor (placeholder) sencillo para un sólo elemento sustituible, por ejemplo una imagen.

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText android:id="@+id/TxtNombre"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:inputType="text" />

</FrameLayout>
```

Con el código anterior conseguimos un layout tan sencillo como el siguiente:



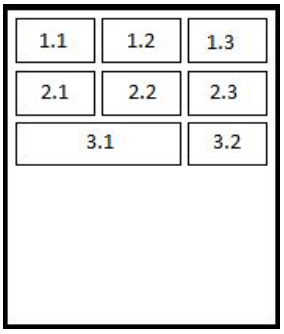
TableLayout :

Un TableLayout permite distribuir sus elementos hijos de forma tabular, definiendo las filas y columnas necesarias, y la posición de cada componente dentro de la tabla.
La estructura de la tabla se define de forma similar a como se hace en HTML, es decir, indicando las filas que compondrán la tabla (objetos TableRow), y dentro de cada fila las columnas necesarias, con la salvedad de que no existe ningún objeto especial para definir una columna (algo así como un TableColumn) sino que directamente insertaremos los controles necesarios dentro del TableRow y cada componente insertado (que puede ser un control sencillo o incluso otro ViewGroup) corresponderá a una columna de la tabla. De esta forma, el número final de filas de la tabla se corresponderá con el número de elementos TableRow insertados, y el número total de columnas quedará determinado por el número de componentes de la fila que más componentes contenga.

```
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

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

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

```
    <TableRow>
        <TextView android:text="Celda 3.1"
                android:layout_span="2" />
        <TextView android:text="Celda 3.2" />
    </TableRow>
</TableLayout>
```

El layout resultante del código anterior sería el siguiente:



GridLayout :

Este tipo de layout fue incluido a partir de la API 14 (Android 4.0) y sus características son similares al TableLayout, ya que se utiliza igualmente para distribuir los diferentes elementos de la interfaz de forma tabular, distribuidos en filas y columnas. La diferencia entre ellos estriba en la forma que tiene el GridLayout de colocar y distribuir sus elementos hijos en el espacio disponible. En este caso, a diferencia del TableLayout indicaremos el número de filas y columnas como propiedades del layout, mediante android:rowCount y android:columnCount. Con estos datos ya no es necesario ningún tipo de elemento para indicar las filas, como hacíamos con el elemento TableRow del TableLayout, sino que los diferentes elementos hijos se irán colocando ordenadamente por filas o columnas (dependiendo de la propiedad android:orientation) hasta completar el número de filas o columnas indicadas en los atributos anteriores.

```
<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>
```

4. Describir cada uno de los directorios, ficheros y subdirectorios de la carpeta "app" de un proyecto Android de Android Studio, Recopilar información de esta página, de Android Developers y/o otras Webs.

Carpeta /app/src/main/java:
Esta carpeta contendrá todo el código fuente de la aplicación, clases auxiliares, etc. Inicialmente, Android Studio creará por nosotros el código básico de la pantalla (actividad o activity) principal de la aplicación, que recordemos que en nuestro caso era MainActivity, y siempre bajo la estructura del paquete java definido durante la creación del proyecto.

Carpeta /app/src/main/res/:
Contiene todos los ficheros de recursos necesarios para el proyecto: imágenes, layouts, cadenas de texto, etc. Los diferentes tipos de recursos se pueden distribuir entre las siguientes subcarpetas:

/res/drawable/ *:*Contiene las imágenes y otros elementos gráficos usados por la aplicación. Para poder definir diferentes recursos dependiendo de la resolución y densidad de la pantalla del dispositivo se suele dividir en varias subcarpetas:

/drawable (recursos independientes de la densidad)

/drawable-ldpi (densidad baja)

/drawable-mdpi (densidad media)

/drawable-hdpi (densidad alta)

/drawable-xhdpi (densidad muy alta)

/drawable-xxhdpi (densidad muy muy alta :

/res/mipmap/ :Contiene los iconos de lanzamiento de la aplicación (el icono que aparecerá en el menú de aplicaciones del dispositivo) para las distintas densidades de pantalla existentes. Al igual que en el caso de las carpetas /drawable, se dividirá en varias subcarpetas dependiendo de la densidad de pantalla:

/mipmap-mdpi

/mipmap-hdpi

/mipmap-xhdpi

…

/res/layout/ :Contiene los ficheros de definición XML de las diferentes pantallas de la interfaz gráfica. Para definir distintos layouts dependiendo de la orientación del dispositivo se puede dividir también en subcarpetas:

/layout (vertical)

/layout-land (horizontal)

/res/anim/ , /res/animator/:Contienen la definición de las animaciones utilizadas por la aplicación.

/res/color/ :Contiene ficheros XML de definición de listas de colores según estado.

/res/menu/ :Contiene la definición XML de los menús de la aplicación.

/res/xml/ :Contiene otros ficheros XML de datos utilizados por la aplicación.

/res/raw/ :Contiene recursos adicionales, normalmente en formato distinto a XML, que no se incluyan en el resto de carpetas de recursos.

/res/values/ :Contiene otros ficheros XML de recursos de la aplicación, como por ejemplo cadenas de texto (strings.xml), estilos (styles.xml), colores (colors.xml), arrays de valores (arrays.xml), tamaños (dimens.xml), etc.

Entre los recursos creados por defecto cabe destacar los **layouts**, en nuestro caso sólo tendremos por ahora el llamado “activity_main.xml”, que contienen la definición de la interfaz gráfica de la pantalla principal de la aplicación.

Fichero /app/src/main/AndroidManifest.xml :
Contiene la definición en XML de muchos de los aspectos principales de la aplicación, como por ejemplo su identificación (nombre, icono, …), sus componentes (pantallas, servicios, …), o los permisos necesarios para su ejecución.

Fichero /app/build.gradle :
Contiene información necesaria para la compilación del proyecto, por ejemplo la versión del SDK de Android utilizada para compilar, la mínima versión de Android que soportará la aplicación, referencias a las librerías externas utilizadas, etc.
En un proyecto pueden existir varios ficheros build.gradle, para definir determinados parámetros a distintos niveles.

Carpeta /app/libs :
Puede contener las librerías java externas (ficheros .jar) que utilice nuestra aplicación. Normalmente no incluiremos directamente aquí ninguna librería, sino que haremos referencia a ellas en el fichero build.gradle descrito en el punto anterior, de forma que entren en el proceso de compilación de nuestra aplicación.

Carpeta /app/build/
Contiene una serie de elementos de código generados automáticamente al compilar el proyecto. Cada vez que compilamos nuestro proyecto, la maquinaria de compilación de Android genera por nosotros una serie de ficheros fuente java dirigidos, entre otras muchas

cosas, al control de los recursos de la aplicación. Importante: dado que estos ficheros se generan automáticamente tras cada compilación del proyecto es importante que no se modifiquen manualmente bajo ninguna circunstancia.

La clase R contendrá en todo momento una serie de constantes con los identificadores (ID) de todos los recursos de la aplicación incluidos en la carpeta /app/src/main/res/, de forma que podamos acceder fácilmente a estos recursos desde nuestro código java a través de dicho dato. Así, por ejemplo, la constante R.layout.activity_main contendrá el ID del layout “activity_main.xml” contenido en la carpeta /app/src/main/res/layout/.