Interfaz de usuario en Android: Layouts

En el artículo anterior del curso, donde desarrollamos una sencilla aplicación Android desde cero, ya hicimos algunos comentarios sobre los *layouts*. Como ya indicamos, los *layouts* son elementos no visuales destinados a controlar la distribución, posición y dimensiones de los controles que se insertan en su interior. Estos componentes extienden a la clase base `ViewGroup`, como muchos otros componentes contenedores, es decir, capaces de contener a otros controles. En el post anterior conocimos la existencia de un tipo concreto de layout, `LinearLayout`, aunque Android nos proporciona algunos otros. Veámos cuántos y cuáles.

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.

Los componentes incluidos en un `FrameLayout` podrán establecer sus propiedades `android:layout_width` y `android:layout_height`, que podrán tomar los valores "`match_parent`" (para que el control hijo tome la dimensión de su layout contenedor) o "`wrap_content`" (para que el control hijo tome la dimensión de su contenido). Veamos un ejemplo:

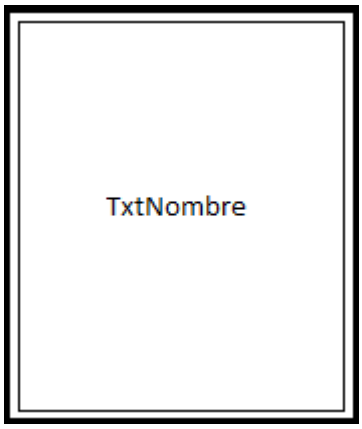
Ejemplo:

```
<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:



LinearLayout

El siguiente tipo de layout en cuanto a nivel de complejidad es el `LinearLayout`. Este layout apila uno tras otro todos sus elementos hijos en sentido horizontal o vertical según se establezca su propiedad `android:orientation`.

Al igual que en un `FrameLayout`, los elementos contenidos en un `LinearLayout` pueden establecer sus propiedades `android:layout_width` y `android:layout_height` para determinar sus dimensiones dentro del layout

```
<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>
```

.

Pero en el caso de un `LinearLayout`, tendremos otro parámetro con el que jugar, la propiedad `android:layout_weight`. Esta propiedad nos va a permitir dar a los elementos contenidos en el layout unas dimensiones proporcionales entre ellas. Esto es más dificil de explicar que de comprender con un ejemplo. Si incluimos en un `LinearLayout` vertical dos cuadros de texto (`EditText`) y a uno de ellos le establecemos un `layout_weight=”1”` y al otro un `layout_weight=”2”`

conseguiremos como efecto que toda la superficie del layout quede ocupada por los dos cuadros de texto y que además el segundo sea el doble (relación entre sus propiedades `weight`) de alto que el primero.

```
<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/TxtDato1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:inputType="text"
        android:layout_weight="1" />

    <EditText android:id="@+id/TxtDato2"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:inputType="text"
        android:layout_weight="2" />

</LinearLayout>
```

Con el código anterior conseguiríamos un layout como el siguiente:

Así pues, a pesar de la simplicidad aparente de este layout resulta ser lo suficiente versátil como para sernos de utilidad en muchas ocasiones.

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.

Por norma general, el ancho de cada columna se corresponderá con el ancho del mayor componente de dicha columna, pero existen una serie de propiedades que nos ayudarán a modificar este comportamiento:

- `android:stretchColumns`. Indicará las columnas que pueden expandir para absorver el espacio libre dejado por las demás columnas a la derecha de la pantalla.
- `android:shrinkColumns`. Indicará las columnas que se pueden contraer para dejar espacio al resto de columnas que se puedan salir por la derecha de la palntalla.
- `android:collapseColumns`. Indicará las columnas de la tabla que se quieren ocultar completamente.

Todas estas propiedades del `TableLayout` pueden recibir una lista de índices de columnas separados por comas (ejemplo: `android:stretchColumns="1,2,3"`) o un asterisco para indicar que debe aplicar a todas las columnas (ejemplo: `android:stretchColumns="*"`).

Otra característica importante es la posibilidad de que una celda determinada pueda ocupar el espacio de varias columnas de la tabla (análogo al atributo `colspan` de HTML). Esto se indicará mediante la propiedad `android:layout_span` del componente concreto que deberá tomar dicho espacio.

Veamos un ejemplo con varios de estos elementos:

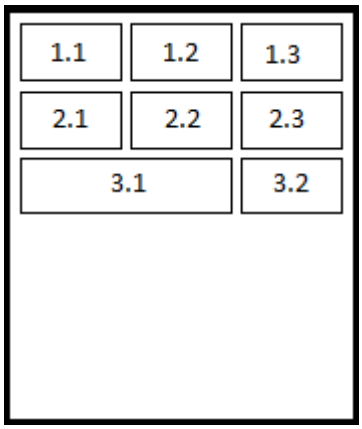
```
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <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. Adicionalmente, igual que en el caso anterior, también tendremos disponibles las propiedades `android:layout_rowSpan` y `android:layout_columnSpan` para conseguir que una celda ocupe el lugar de varias filas o columnas.

Existe también una forma de indicar de forma explícita la fila y columna que debe ocupar un determinado elemento hijo contenido en el `GridLayout`, y se consigue

utilizando los atributos `android:layout_row` y `android:layout_column`. De cualquier forma, salvo para configuraciones complejas del grid no suele ser necesario utilizar estas propiedades.

Con todo esto en cuenta, para conseguir una distribución equivalente a la del ejemplo anterior del `TableLayout`, necesitaríamos escribir un código como el siguiente:

```
<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>
```

RelativeLayout

El último tipo de layout que vamos a ver es el `RelativeLayout`. Este layout permite especificar la posición de cada elemento de forma relativa a su elemento padre o a cualquier otro elemento incluido en el propio layout. De esta forma, al incluir un nuevo elemento X podremos indicar por ejemplo que debe colocarse *debajo del elemento Y* y *alineado a la derecha del layout padre*. Veamos esto en el ejemplo siguiente:

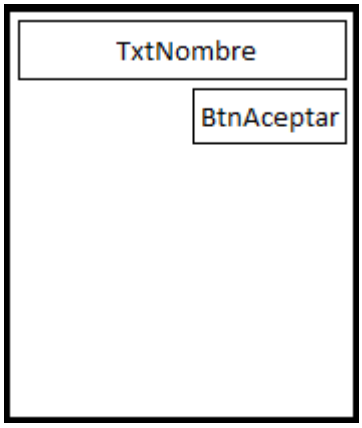
```
<RelativeLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

        <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:



Al igual que estas tres propiedades, en un `RelativeLayout` tendremos un sinfín de propiedades para colocar cada control justo donde queramos. Veamos las principales [creo que sus propios nombres explican perfectamente la función de cada una]:

Posición relativa a otro control:

- android:layout_above
- android:layout_below
- android:layout_toLeftOf
- android:layout_toRightOf
- android:layout_alignLeft
- android:layout_alignRight
- android:layout_alignTop
- android:layout_alignBottom
- android:layout_alignBaseline

Posición relativa al layout padre:

- android:layout_alignParentLeft
- android:layout_alignParentRight
- android:layout_alignParentTop
- android:layout_alignParentBottom
- android:layout_centerHorizontal
- android:layout_centerVertical
- android:layout_centerInParent

Por último indicar que cualquiera de los tipos de layout anteriores poseen otras propiedades comunes como por ejemplo los márgenes exteriores (*margin*) e interiores (*padding*) que pueden establecerse mediante los siguientes atributos:

Opciones de margen exterior:

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

Opciones de margen interior:

- android:padding
- android:paddingBottom
- android:paddingTop
- android:paddingLeft
- android:paddingRight

Existen otros layouts algo más sofisticados a los que dedicaremos artículos específicos un poco más adelante, como por ejemplo el `DrawerLayout` para añadir menús laterales deslizantes.

También en próximos artículos veremos otros elementos comunes que extienden a `ViewGroup`, como por ejemplo las vistas de tipo lista (`ListView`), de tipo grid (`GridView`), y las pestañas o tabs (`TabHost`/`TabWidget`).