

Layout y contenedores

Estos elementos son los padres que contendrán todos los componentes de nuestra aplicación. Son elementos no visuales destinados a controlar la distribución, posición y dimensiones de los controles que se insertan en su interior.

Se pueden manejar de dos formas:

- Declarando los ficheros en XML: como hasta ahora: así se separa la interface de usuario de la lógica de tu programa, además de que las descripciones de tu interfaz son externas al código, pudiendo cambiar el diseño sin tocar el código. Esto nos proporciona FLEXIBILIDAD.
- Instanciando objetos programáticamente en tiempo de ejecución.

a.- Declarados en los ficheros en XML

En este documento vamos a ver los más esenciales

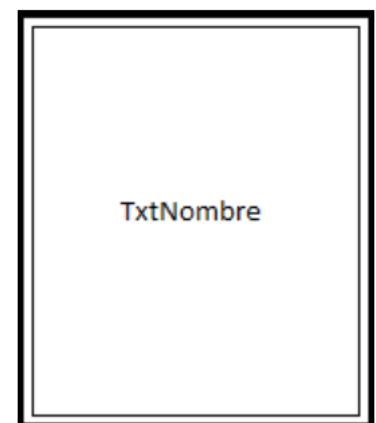
FrameLayout

El más simple. 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, por ejemplo una imagen.

Propiedades `:android:layout_width` y `android:layout_height`, que podrán tomar los valores `"match_parent"` (hijo tome la dimensión de su layout contenedor) o `"wrap_content"` (para que el control hijo tome la dimensión de su contenido).

Ejemplo:

```
1 <FrameLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3   android:layout_width="match_parent"
4   android:layout_height="match_parent">
5
6   <EditText android:id="@+id/TxtNombre"
7       android:layout_width="match_parent"
8       android:layout_height="match_parent"
9       android:inputType="text" />
10
11 </FrameLayout>
```



LinearLayout

Este layout apila uno tras otro todos sus elementos hijos en sentido horizontal o vertical según su propiedad `android:orientation`.

Pueden establecer sus propiedades `android:layout_width` y `android:layout_height` para determinar sus dimensiones dentro del layout.

Además la propiedad `android:layout_weight`. va a permitir dar a los elementos contenidos en el layout unas dimensiones proporcionales entre ellas.

<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>

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.

Indica 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 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.

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`. columnas que pueden expandir para absorber el espacio libre a la derecha.
- `android:shrinkColumns`. columnas que se pueden contraer para dejar espacio al resto de columnas.
- `android:collapseColumns`. las columnas de la tabla que se quieren ocultar.

Pueden recibir una lista de índices (ejemplo: `android:stretchColumns="1, 2, 3"`) o un asterisco.

Una celda determinada pueda ocupar el espacio de varias columnas de la tabla:
android:layout_span del componente concreto que deberá tomar dicho espacio.

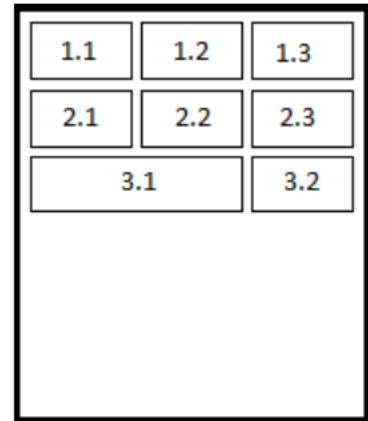
Veamos un ejemplo con varios de estos elementos:

```
<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>
```



1.1	1.2	1.3
2.1	2.2	2.3
3.1		3.2

GridLayout

Características son similares al TableLayout, ya que distribuye en filas y columnas. Pero GridLayout indica el número de filas y columnas como propiedades del layout, mediante android:rowCount y android:columnCount. Los diferentes elementos hijos se irán colocando ordenadamente por filas o columnas (dependiendo de la propiedad android:orientation). 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.

Permite indicar de forma explícita la fila y columna que debe ocupar un hijo contenido utilizando los atributos android:layout_row y android:layout_column.

```

<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" />

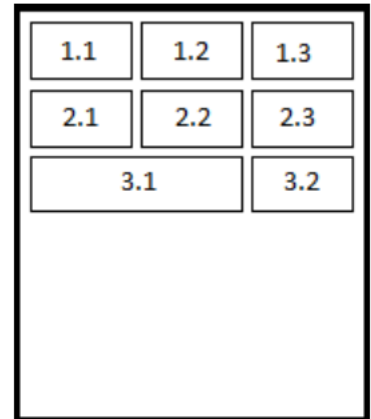
    <TextView android:text="Celda 2.1" />
    <TextView android:text="Celda 2.2" />
    <TextView android:text="Celda 2.3" />

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

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

</GridLayout>

```



RelativeLayout

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.*

```

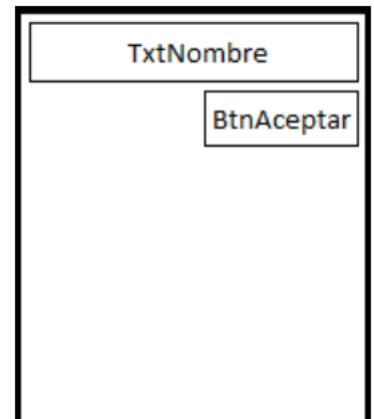
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <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>

```



Más propiedades de RelativeLayout:

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

Propiedades comunes a los Layout vistos referentes a 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

b.- Trabajando programáticamente con contenedores.

Recorrer es obtener una referencia a cada uno de los widgets que están en un Layout que se puede hacer fácilmente con un for.

El método **getChildAt (int index)** de la clase `GroupView` devuelve una referencia al widget con identificador `index` dentro del contenedor. `Index` representa el elemento insertado en esa posición. Puedes pensar que el Layout es un array dinámico con widgets dentro y gracias al índice accedes a cada uno de los elementos.

El método **getChildCount()** devuelve un número entero indicando cuantos elementos hay en el contenedor.

Con esto obtenemos el siguiente código:

```
public void Recorrer() {  
    View v;  
    GridLayout g = (GridLayout) findViewById(R.id.grid1);  
    for (int i = 0; i < g.getChildCount(); i++) {  
        v = g.getChildAt(i);  
        System.out.println("objeto:" + v.toString());  
    }  
}
```

Primero se hace una referencia al Layout (en el ejemplo un GridLayout).

Con el bucle for se visitan todos los hijos de 0 a getChildCount()-1

En cada iteración se obtiene una referencia al i-ésimo widget hijo (v)

b.1.- Diferenciando tipos

Cada contenedor puede tener distintos tipos de widgets, con lo cual tendremos que saber qué tipo es se puede usar el método getClass de la clase View, invocando al método **getSimpleName** para conocer el nombre de la clase a la que pertenece. Luego podemos hacer un cast a un objeto de una clase determinada para obtener sus propiedades. Ejemplo

```
Button b;
if (v.getClass().getSimpleName().equals("Button")) {
    b=(Button)v;
    b.setOnClickListener(...);
    //o cualquier otro método o propiedad de la clase Button
}
```

b.2.- Añadiendo elementos al contenedor

Puedes añadir los elementos que quieras teniendo en cuenta las siguientes consideraciones:

- Cada widget que incluyas tiene que tener definidos sus parámetros de layout. Los Widgets tienen un método llamado **setLayoutParams** para establecer las propiedades en tiempo de ejecución.
- Cada widgets tienen que tener un identificador. En XML es con la propiedad id y en código se hace con un entero. Para ello y para no duplicar los id, desde la Api 17 proporciona el método *generateViewId* que devuelve un id único.

La clase GroupView incorpora el método **addView(View v, int index)** para incorporar todos los elementos que desees. Siguiendo ejemplo incorpora 18 botones en un GridLayout:

```
public void añadeHijos(){
    GridLayout g = (GridLayout) findViewById(R.id.grid1);
    Button b;
    for(int i=0; i < 18; i++){
        b = new Button(this);
        b.setLayoutParams (new ViewGroup.LayoutParams(
            ViewGroup.LayoutParams.WRAP_CONTENT,
            ViewGroup.LayoutParams.WRAP_CONTENT));
        b.setText("bnt"+i);
        b.setId(View.generateViewId());
        g.addView(b,i);
    }
}
```

```
}
```

Por cada iteración se producen las siguientes iteraciones:

1. Se instancia el boton: `b = new Button(this);`
2. Se establecen los parámetros de tamaño para el Layout, es decir, se ajusta al contenido del boton en altura y anchura
`b.setLayoutParams (new ViewGroup.LayoutParams(
ViewGroup.LayoutParams.WRAP_CONTENT,
ViewGroup.LayoutParams.WRAP_CONTENT));`
3. Se establece el texto del botón: `b.setText("bnt"+i);`
4. Se establece el identificador único para cada uno de los botones:
`b.setId(View.generateViewId());`
5. Por último se añade al contenedor: `g.addView(b,i);`

Si cada uno de los botones contenidos responde a eventos de distinta acciones y formas, necesitas un objeto Listener que escuche por todos (o por cierto elementos individuales). Por ejemplo, imagina que quieres que cada botón del ejemplo anterior responda a un evento para hacer una acción determinada. Podríamos programar el siguiente código:

```
public class MyActivity extends Activity implements View.OnClickListener{  
    ...  
    protected void onCreate(Bundle savedInstanceState){  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_my);  
        añadeHijos();  
    }  
    public void añadeHijos(){  
        GridLayout g = (GridLayout)findViewById(R.id.grid1);  
        Button b;  
        for(int i=0; i < 18; i++){  
            b = new Button(this);  
            b.setLayoutParams (new ViewGroup.LayoutParams(  
                ViewGroup.LayoutParams.WRAP_CONTENT,  
                ViewGroup.LayoutParams.WRAP_CONTENT));  
            b.setText("bnt"+i);  
            b.setId(View.generateViewId());  
            b.setOnClickListener(this);  
            g.addView(b,i);  
        }  
    }  
}
```

```

    }
}
public void onClick(View v){
    if (v.getClass().getSimpleName().equals("Button")){
        Button b = (Button) v;
        accion(b);
    }
}
public void accion(Button b){
    //programa aquí tu acción con el botón b
}
}

```

Observa como en cada iteración se establece el listener a la propia actividad, para que responda mediante el método `onClick`. Finalmente, cuando se presiona uno de los botones del contenedor del `GridLayout`, se invoca al método `accion` pasando como parámetro el botón pulsado para consultas posteriores.