Layout Managers

Android ofrece una colección de clases que son contenedoras de otras clases, llamadas layout managers. Cada una de estas clases maneja la posición y el tamaño de las views que contiene, de forma diferente.

Tipos de layout managers

- LinearLayout
- TableLayout
- RelativeLayout
- FrameLayout

LinearLayout

Organiza sus objetos hijos (views) horizontal o verticalmente, según el valor de la propiedad "orientation".

Definicion en archivo xml de layout:

```
<?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" >

    <!-- views hijos -->

</LinearLayout>
```

Las Views se apilan una debajo de la otra:

Parámetros de las Views contenidas

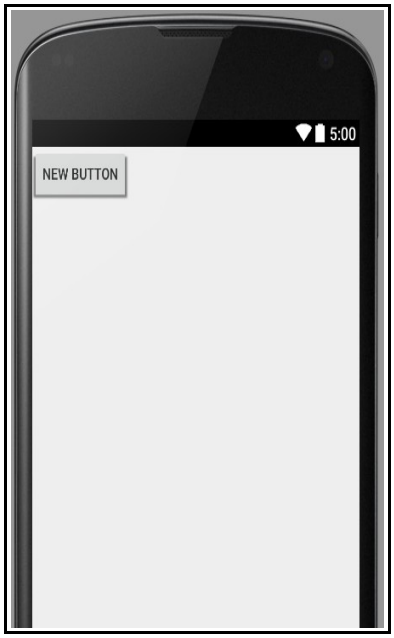
Los siguientes parámetros permiten ubicar con mas precisión las views y contenedores en la pantalla:

- layout_width
- layout_height
- layout_weight
- gravity
- layout_gravity

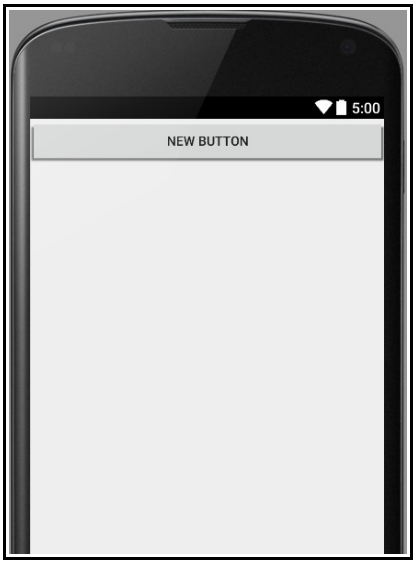
layout_width: Determina el ancho de la view (o contenedor), puede tomar dos valores posibles:

- **match_parent:** La view se extiende hasta llenar el espacio que ofrece el contenedor de la cual es hija.
- **wrap_content:** La view solo ocupa el espacio mínimo que necesita dentro del contenedor de la cual es hija.
- También puede setearse el ancho en px o dp.

Ejemplo:




```
<?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="vertical" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button" />

</LinearLayout>
```




```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res
/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/button1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Button" />

</LinearLayout>
```

layout_height: Determina el alto de la view (o contenedor), puede tomar dos valores posibles:

- **match_parent:** La view se extiende hasta llenar el espacio que ofrece el contenedor de la cual es hija.
- **wrap_content:** La view solo ocupa el espacio mínimo que necesita dentro del contenedor de la cual es hija.
- También puede setearse el ancho en px o dp.

layout_weight: (de 0.0 a 1.0) indica la proporción del espacio libre que queda en el contenedor, que es asignada a una view, por defecto es cero.

Ejemplo, con weight 0.0 – 0.0 – 0.0 (izquierda), y con weigth 0.75 – 0.25 – 0.0 (Derecha)

Puede observarse en el xml, que el atributo layout_weight es seteado en 0.75 , 0.25 y 0.0 para los tres EditText que contiene el LinearLayout

```
<?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="vertical" >

    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_weight="0.75"
        android:layout_height="wrap_content" />


    <EditText
        android:id="@+id/editText2"
        android:layout_width="match_parent"
         android:layout_weight="0.25"
        android:layout_height="wrap_content" />

    <EditText
        android:id="@+id/editText3"
        android:layout_width="match_parent"
        android:layout_weight="0.0"
        android:layout_height="wrap_content" />

</LinearLayout>
```

gravity: Es un parámetro usado por la View para alinear algo de su contenido (por ejemplo texto en un TextView o EditText). Los valores posibles son:

- **left** (a la izquierda (moviéndose en forma horizontal))
- **rigth** (a la derecha (moviéndose en forma horizontal))
- **center** (centrado horizontal y vertical)
- **top** (arriba (moviéndose en forma vertical))
- **botton** (abajo (moviéndose en forma vertical))
- **center_vertical** (centrado (moviéndose en forma vertical))
- **center_horizontal** (centrado (moviéndose en forma horizontal))

Ejemplo con gravity seteado en alguno de los valores que se detallaron anteriormente:


```
<EditText
    android:id="@+id/editText1"
    android:layout_width="match_parent"
    android:layout_weight="0.75"
    android:text="hola"
    android:gravity="bottom"
    android:layout_height="wrap_content" />

<EditText
    android:id="@+id/editText2"
    android:layout_width="match_parent"
   android:layout_weight="0.25"
   android:text="hola"
   android:gravity="center"
    android:layout_height="wrap_content" />

<EditText
    android:id="@+id/editText3"
    android:layout_width="match_parent"
    android:layout_weight="0.0"
    android:text="hola"
    android:gravity="right"
    android:layout_height="wrap_content" />
```

layout_gravity: Es igual que el parámetro gravity, pero este parámetro es utilizado por el contenedor de la view, para ubicarla dentro de sí con las condiciones que aclara el parámetro.

Ejemplo:

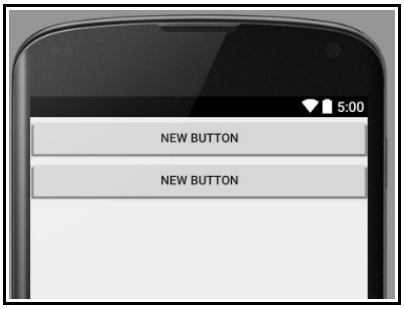

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/a
ndroid"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="Button" />

</LinearLayout>
```

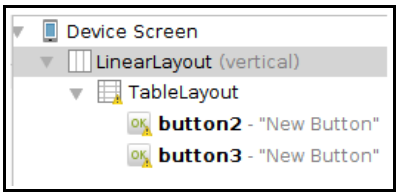
NOTA: Puede observarse que los atributos están definidos en el objeto y no en el Layout.

TableLayout

Administra las Views que contiene en forma de filas y columnas.

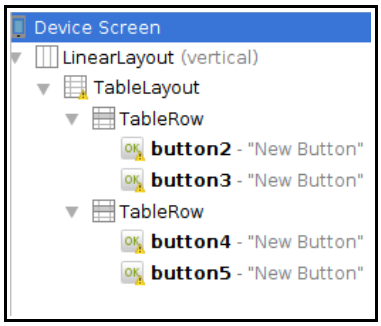
Se considerará una fila, a toda view dentro del TableLayout, por ejemplo, aquí podemos ver dos botones definidos como filas de una TableLayout:






```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/androi
d"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TableLayout
        android:id="@+id/tableLayout1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <Button
            android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button" />

        <Button
            android:id="@+id/button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button" />

    </TableLayout>

</LinearLayout>
```

Para agregar mas de una columna a la tabla, se utiliza el contenedor TableRow, dentro del cual colocaremos las Views que queramos que sean las columnas.



XML utilizando TableRow

```
<TableLayout
    android:id="@+id/tableLayout1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >

    <TableRow
        android:id="@+id/tableRow1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >

        <Button
            android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button" />

        <Button
            android:id="@+id/button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button" />

    </TableRow>

    <TableRow
        android:id="@+id/tableRow2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <Button
            android:id="@+id/button3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button" />

        <Button
            android:id="@+id/button4"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button" />
    </TableRow>

</TableLayout>
```

Algunos parámetros de TableLayout

- stretchColumns
- shrinkColumns

stretchColumns : estira el contenido de las columnas en forma horizontal, hasta completar el objeto que las contiene, las columnas que estira se aclaran en el parámetro con números, comenzando de cero, y separados por coma.

Ejemplo :

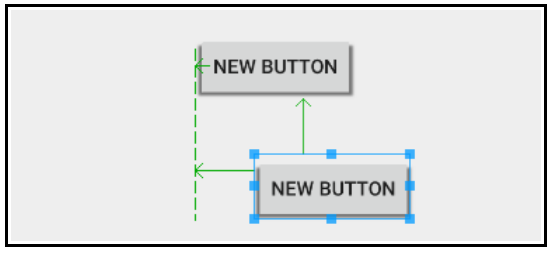

```
<TableLayout
android:id ="@+id/tableLayout1"
 android:layout_width ="match_parent"
 android:stretchColumns ="0"
 android:layout_height ="wrap_content" >
```

En este ejemplo, aclaramos en el parámetro stretchColumns, que la columna 0 (la primera) sea estirada, en el caso de querer hacer ésto en más de una columna, los números de columna irán separados por coma.

shrinkColumns : reduce el contenido de las columnas en forma horizontal, (lo opuesto a stretchColumns) , se aclaran las columnas del mismo modo.

RelativeLayout

En este contenedor, la posición de cada View, se describe relativa a alguna otra View o contenedor. Es muy fácil posicionar los objetos mediante el editor gráfico del IDE, el cual generará automáticamente el archivo XML.



FrameLayout

Este layout, es utilizado para mostrar una sola view dinámicamente, dentro del contenedor, cargando muchos items en el FrameLayout, y seteando por programa, cual es el que está visible.

Ejemplo:

```
<?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="vertical" >

    <FrameLayout
        android:id="@+id/frameLayout1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >


        <ImageView
            android:id="@+id/imageView1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/android" />


        <ImageView
            android:id="@+id/imageView2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/crawler" />

    </FrameLayout>

</LinearLayout>
```

En este ejemplo, colocamos 2 ImageView dentro del FrameLayout, por lo que una imagen quedó sobre la otra y solo se observa una sola. (si la imagen tiene transparencia, pueden verse parte de las 2).Hasta el momento ambas Views están “visibles” en el FrameLayout.

Implementaremos un código Java el cual escuchará el evento de click de ambas imágenes e intercambiará los estados de visibilidad de ambas, para que aparezcan en forma alternada cada vez que se hace un click sobre la imagen.

La propiedad que seteamos a la view ImageView, es “Visibility” mediante el método “setVisibility()” el cual puede tomar los valores:

- **View.GONE :** el objeto no se ve.
- **View.VISIBLE :** el objeto se ve.

Código java de la activity:

```
public class Pantalla1Activity extends Activity {

       ImageView img1;
       ImageView img2;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.layout_frame);

        img1 = (ImageView) findViewById(R.id.imageView1);
        img2 = (ImageView) findViewById(R.id.imageView2);

        img1.setOnClickListener(new OnClickListener() {

                    @Override
                    public void onClick(View v) {
                           Pantalla1Activity.this.img1.setVisibility(View.GONE);
                           Pantalla1Activity.this.img2.setVisibility(View.VISIBLE);
                    }
             });
        img2.setOnClickListener(new OnClickListener() {

                    @Override
                    public void onClick(View v) {
                           Pantalla1Activity.this.img1.setVisibility(View.VISIBLE);
                           Pantalla1Activity.this.img2.setVisibility(View.GONE);

                    }
             });
    }
}
```

NOTA : FrameLayout no fuerza al programador a tener seteado como visible, solo una View, sino que irá apilando las views una encima de otra.

Este ejemplo pretende demostrar que el FrameLayout, apila las Views, dando la posibilidad, de colocar una sobre otra, opción que no puede realizarse con ningún otro Layout.

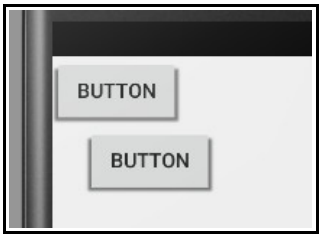
Espaciado de objetos en pantalla

Existen dos tipos de espaciado entre los objetos: Padding y Margin.

Mientras que "Padding" es un espacio situado entre el borde de la View y su contenido, "Margin" se sitúa entre el borde de la View y los bordes de los elementos que lo rodean o del layout que lo contiene.
Ambos atributos se setean en el objeto Form View que se quiere modificar.

Ejemplo de Padding a la izquierda:

Ejemplo de Margin a la izquierda:



Valores de padding posibles y su definición como atributo en el xml:

- **android:padding:** Setea el espaciado para los 4 lados del objeto.
- **android:paddingLeft:** Setea el espaciado para el lado izquierdo.
- **android:paddingRight:** Setea el espaciado para el lado derecho.
- **android:paddingTop:** Setea el espaciado para el lado superior.
- **android:paddingBotton:** Setea el espaciado para el lado inferior.

Ejemplo:

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingRight="20dp"
    android:text="Button" />
```

Valores de margin posibles y su definición como atributo en el xml:

- **android:layout_margin:** Setea el margen para los 4 lados del objeto.
- **android:layout_marginLeft:** Setea el margen para el lado izquierdo.
- **android:layout_margin Right:** Setea el margen para el lado derecho.
- **android:layout_margin Top:** Setea el margen para el lado superior.
- **android:layout_margin Botton:** Setea el margen para el lado inferior.

Ejemplo:

```
<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="20dp"
    android:text="Button" />
```