

Lo que se verá ahora en el curso es lo siguiente.

- ✚ Layouts
- ✚ Componentes de interface de usuario
- ✚ Programación de eventos de interface de usuario

## Layouts

Los layouts facilitan la organización de los distintos elementos que componen una interfaz. Sirven de contenedor a los componentes de una vista. Todos los layouts Android heredan de la clase **ViewGroup**.

### 1. FrameLayout

El FrameLayout es el contenedor más sencillo, representa un espacio que muestra un objeto a su elección.

Cualquier elemento añadido a un FrameLayout se ubica en la esquina superior izquierda del layout. Puede cambiar esta posición mediante el atributo **android:gravity**.

Tiene la posibilidad de añadir varios elementos en un mismo FrameLayout y de modificar la visibilidad de estos elementos para mostrar u ocultar varios elementos en un mismo lugar.

### 2. LinearLayout

El LinearLayout permite alinear elementos (en el orden de las declaraciones) en una dirección (vertical u horizontal).

Puede definir los atributos siguientes:

- Orientación del layout.
- Gravedad de los elementos.
- Peso de los elementos.

#### Orientación

En la creación de un LinearLayout, debe detallar su orientación (horizontal o vertical) mediante el uso del atributo **android:orientation**.



La orientación por defecto es la horizontal.

#### Posicionamiento de un elemento

Para definir el posicionamiento de un elemento en un LinearLayout, hay dos atributos disponibles:

- **layout\_gravity**: especifica el posicionamiento de un elemento en su contenedor.
- **gravity**: especifica el posicionamiento del contenido de un elemento (por ejemplo, se puede especificar la posición de un texto en un botón).

A continuación se muestra un ejemplo con ambos atributos:

```

<?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:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="ANDROID: LAYOUT_GRAVITY=RIGHT" />

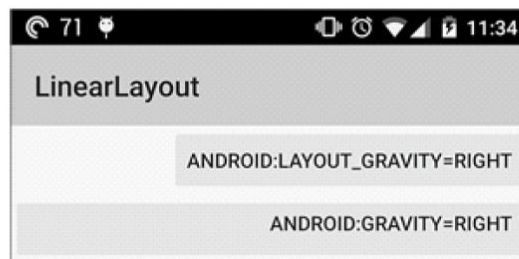
    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="right"
        android:text="ANDROID: LAYOUT_GRAVITY=RIGHT" />

</LinearLayout>

```

Esta vista se compone de:

- un Layout, definido con una orientación vertical.
- un Botón A, que tiene una anchura igual a la de su contenido. Este botón se ubica a la derecha del layout (**layout\_gravity**).
- un Botón B, que tiene un texto ubicado a la derecha del botón (**gravity**).



### Peso de un elemento

El peso sirve para indicar a un elemento el espacio que ocupa. Cuanto mayor sea el peso de un elemento, más se podrá extender un componente y ocupar el espacio disponible. El tamaño del espacio que desea alargar debe ser de **0dp**.



El valor por defecto del peso de un elemento es 0.

En este ejemplo, el peso de dos botones es idéntico para que cada uno de ellos tenga el 50 % del ancho disponible.

```

<?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="horizontal"
    android:weightSum="2">

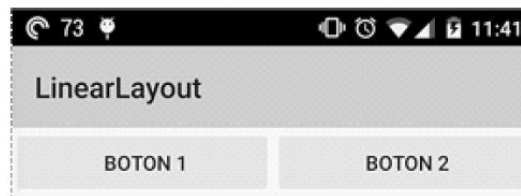
    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="BOTON 1" />

    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="BOTON 2" />

</LinearLayout>

```

Con lo que obtenemos:



El ejemplo siguiente representa dos botones alineados de forma vertical. Sin embargo, el botón 1 es el doble de grande que el botón 2. Puede observar que el peso del botón 1 es también el doble que el del botón 2.

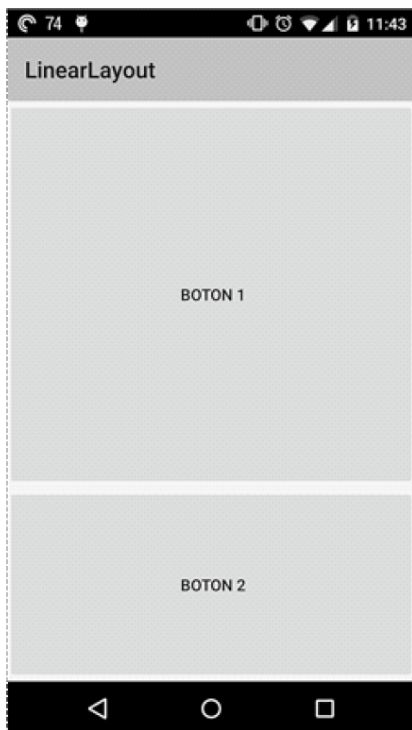
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <Button
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="2"
        android:text="BOTON 1" />

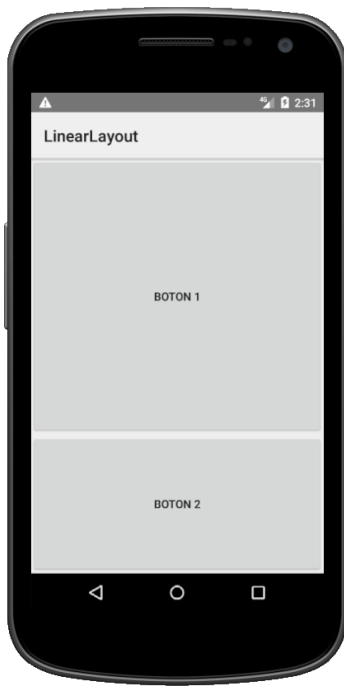
    <Button
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:text="BOTON 2" />

</LinearLayout>
```

Con lo que obtenemos:



Ver programa de ejemplo: Ch5\_LinearLayout



### 3. TableLayout

El TableLayout posiciona todos sus hijos en filas y columnas, en forma de tabla (esta tabla hereda de LinearLayout). Ofrece la posibilidad de dejar celdas vacías, pero no la de extenderse en varias filas o en varias columnas.

Para crear filas en un TableLayout, utilice el elemento **TableRow**. Cada elemento de tipo TableRow representa un elemento con 0 o más columnas en su interior.

A continuación tenemos un ejemplo de una vista que tiene dos filas, cada una de ellas compuesta por dos elementos (dos columnas):

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <TableRow>
        <TextView
            android:padding="3dp"
            android:text="Fila 1 - Columna 1" />
        <TextView
            android:padding="3dp"
            android:text="Fila 1 - Columna 2" />
    </TableRow>

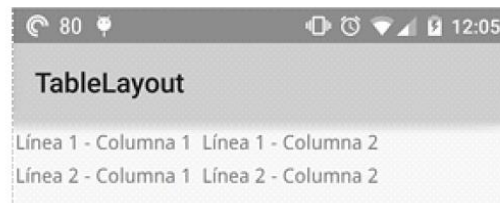
    <TableRow>
        <TextView
            android:padding="3dp"
            android:text="Fila 2 - Columna 1" />
```

```

        <TextView
            android:padding="3dp"
            android:text="Fila 2 - Columna 2" />
    </TableRow>
</TableLayout>

```

Con lo que obtenemos:



El atributo **padding** permite especificar el margen interno de un elemento (el espacio entre los bordes del elemento y su contenido). En cambio, el atributo **margin**, permite especificar el margen externo de un elemento.

Ver programa de ejemplo: Ch5\_TableLayout



## 4. RelativeLayout

Este layout permite ubicar unos elementos en función de los otros. Un elemento puede posicionarse en función de su contenedor o en función de otro elemento de la vista.

### Posicionamiento relativo al contenedor

Puede posicionar un elemento en función de los bordes del contenedor: para ello, existen diversos atributos que permiten posicionar un elemento en función de sus cuatro bordes (arriba, abajo, derecha e izquierda).

Para ello, utilice uno o varios de los atributos siguientes con un valor booleano true/false (verdadero/falso).

- **android:layout\_alignParentTop**: alinear el borde superior del elemento con el borde superior del contenedor.
- **android:layout\_alignParentBottom**: alinear el borde inferior del elemento con el borde inferior del contenedor.
- **android:layout\_alignParentLeft**: alinear el borde izquierdo del elemento con el borde izquierdo del contenedor.
- **android:layout\_alignParentRight**: alinear el borde derecho del elemento con el borde derecho del contenedor.
- **android:layout\_alignParentEnd**: alinear el final del elemento con el final del contenedor.
- **android:layout\_alignParentStart**: alinear el comienzo del elemento con el comienzo del contenedor.



Puede combinar varios valores de posicionamiento.

### Posicionamiento relativo a otros elementos

Dispone de varias opciones de posicionamiento distintas. Para utilizarlas, añada el atributo deseado indicando el valor del identificador del elemento relativo.

- **android:layout\_above**: alinear el borde superior del elemento con el borde superior del contenedor.
- **android:layout\_below**: alinear el borde inferior del elemento con el borde inferior del contenedor.
- **android:layout\_toLeftOf**: alinear el borde izquierdo del elemento con el borde izquierdo del contenedor.
- **android:layout\_toRightOf**: ubicar un elemento a la derecha del elemento referenciado.
- **android:layout\_alignTop**: indica que el extremo superior de este elemento está alineado con el extremo superior del elemento referenciado.
- **android:layout\_alignBottom**: indica que el extremo inferior de este elemento está alineado con el extremo inferior del elemento referenciado.
- **android:layout\_alignLeft**: indica que el extremo izquierdo de este elemento está alineado con el extremo izquierdo del elemento referenciado.
- **android:layout\_alignRight**: indica que el extremo derecho de este elemento está alineado con el extremo derecho del elemento referenciado.
- **android:layout\_alignBaseline**: indica que las líneas de base de este elemento están alineadas con las del elemento referenciado.

A continuación se muestra un ejemplo de formulario de entrada de datos con este layout:

```

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

    <EditText
        android:id="@+id/direccionEmail"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:hint="Dirección email" />

    <EditText
        android:id="@+id/contrasena"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_below="@id/direccionEmail"
        android:hint="Contraseña" />

    <Button



---


        android:id="@+id/aceptar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignRight="@id/contrasena"
        android:layout_below="@id/contrasena"
        android:text="Aceptar" />

    <Button
        android:id="@+id/cancelar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignTop="@id/aceptar"

        android:layout_toLeftOf="@id/aceptar"
        android:text="Cancelar" />

</RelativeLayout>

```



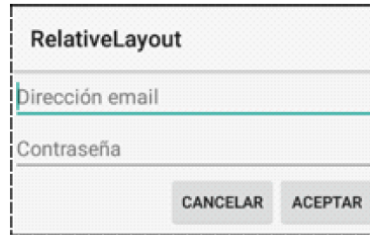
El primer campo está alineado con el extremo superior y derecho del contenedor.

El segundo campo de entrada está alineado por su extremo derecho con el extremo derecho del contenedor y por debajo del elemento anterior.

El primer botón se sitúa debajo del segundo campo, alineado a su derecha.

El segundo botón está alineado por arriba con el primer botón y se encuentra justo a su izquierda.

Con lo que obtenemos:



➤ Puede combinar varios valores de posicionamiento.

También puede referirse a un elemento que se declarará posteriormente en la vista. Para ello, hay que declarar el identificador del siguiente elemento en la referencia (@+id) y reutilizarlo en la declaración del elemento.

Por ejemplo, queremos posicionar un campo de texto (**EditText**) tras un botón cuando se ha declarado antes en el archivo. Para poder asociar el campo de texto al botón y, en particular, a su identificador, hay que declarar el identificador del botón en la referencia del campo de texto (**android:layout\_below**) y asignarla al botón posteriormente (**android:id**).

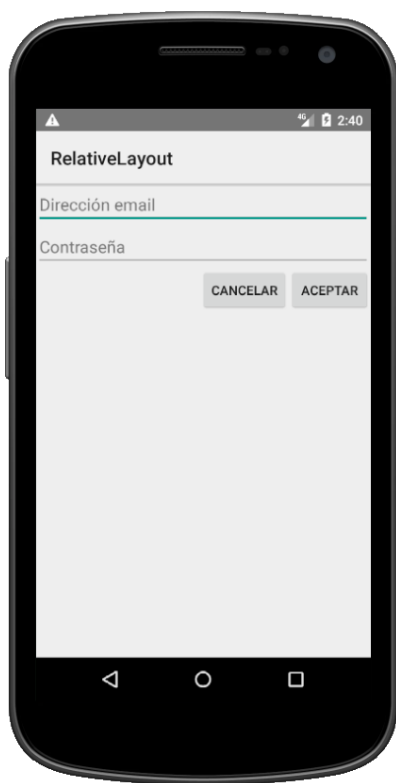
```
<EditText android:id="@+id/login"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/connect" />
```

```
<Button android:id="@id/connect"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/connect" />
```

Puede centrar elementos en un RelativeLayout mediante las opciones:

- **android:layout\_centerHorizontal**: permite centrar un elemento horizontalmente en el contenedor padre.
- **android:layout\_centerVertical**: permite centrar un elemento verticalmente en el contenedor padre.
- **android:layout\_centerInParent**: permite centrar un elemento (horizontal y verticalmente) en el contenedor padre.

Ver programa de ejemplo: Ch5\_RelativeLayout



## 5. GridLayout

El GridLayout (disponible desde la API 14) permite dividir la pantalla en filas y columnas, con la posibilidad de dejar celdas vacías o bien agrupar celdas de la misma fila o columna.

El elemento **spacer** permite dejar una celda vacía y, de este modo, tener espacios vacíos en una interfaz gráfica.

A continuación se muestra una interfaz que contiene dos botones repartidos en dos filas.

```

<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="2"
    android:rowCount="2">

    <Space
        android:layout_width="100dp"
        android:layout_column="0"
        android:layout_row="0" />

    <Button
        android:layout_column="1"
        android:layout_row="0"
        android:text="BOTON 1" />

    <Button
        android:layout_columnSpan="2"
        android:layout_gravity="fill_horizontal"
        android:text="BOTON 2" />

</GridLayout>

```

En primer lugar, en la declaración del GridLayout, se indica el número de columnas (columnCount) y el número de filas (rowCount) que lo componen.

- El primer elemento se corresponde con un espacio vacío, donde se especifica el ancho (width) y su posición (columna y fila).
- A continuación, se posiciona un botón en la fila 0 y la columna 1.
- Por último, se extiende un botón en las dos columnas (atributo ColumnSpan), donde se indica también el comportamiento del botón (aquí fill\_horizontal, para que el botón ocupe todo el espacio disponible).

A continuación se muestra el resultado de la ejecución del ejemplo:



Ver programa de ejemplo: Ch5\_GridLayout



## 🚦 Programas Interfaces de usuario

Componentes de interface de usuario.

### 1. Etiqueta de texto

El elemento que sirve para mostrar un texto en su interfaz es un **TextView**.

A continuación tenemos un ejemplo de uso de un TextView:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/text" />
```

La especificación del texto que se desea mostrar se realiza mediante el atributo **android:text**.

La etiqueta de texto tiene muchos atributos que puede usar en su aplicación (tamaño, color y fuente del texto, posicionamiento, número de filas, etc.).

## 2. Campo de edición de texto

El elemento llamado **EditText** permite al usuario introducir un texto mediante el teclado.

A continuación se muestra un ejemplo de uso:

```
<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/editHint" />
```


El atributo **android:hint** permite indicar al usuario el tipo de texto esperado.

Puede mejorar la experiencia del usuario mostrando un teclado específico en función del tipo de campo. Esto es posible gracias al atributo **android:inputType**.

Este atributo puede tomar los siguientes valores:

- **text** (valor por defecto): teclado normal.
- **textCapCharacters**: teclado todo en mayúsculas.
- **textCapWords**: primera letra automáticamente en mayúsculas.
- **textAutoCorrect**: activa la corrección automática.
- **textMultiLine**: texto en varias líneas.
- **textNoSuggestions**: sin sugerencias de corrección.
- **textUri**: permite introducir una URL web.
- **textEmailAddress**: dirección de correo electrónico.
- **textEmailSubject**: asunto de correo electrónico.
- **textShortMessage**: activa el acceso directo a smiley en el teclado.
- **textPersonName**: permite introducir el nombre de una persona (muestra speech to text en la parte inferior izquierda).
- **textPostalAddress**: permite introducir una dirección postal (muestra speech to text en la parte inferior izquierda del teclado).
- **textPassword**: entrada de una contraseña.
- **textVisiblePassword**: entrada de una contraseña visible.
- **number/numberSigned/numberDecimal/phone/datetime/date/time**: teclado numérico.

Esta lista no es exhaustiva.

 Puede utilizar varios valores separándolos con |.

### 3. Botón

Usar el elemento **Button** es muy sencillo, como puede ver en el siguiente ejemplo:

```
<Button android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/btn" />
```

### 4. Checkbox

El elemento **Checkbox** representa una simple casilla de selección, como las que se pueden activar en los formularios web:

```
<CheckBox android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:checked="true"
          android:text="@string/checkbox" />
```

Puede definir el estado inicial de una checkbox con el atributo **android:checked** (true si activa, false en caso contrario).

Puede conocer el cambio de estado del checkbox con un listener. Para ello, utilice el método **setOnCheckedChangeListener**, e implemente a continuación una nueva instancia de la clase **OnCheckedChangeListener**, lo que supondrá la implementación del método **onCheckedChanged**. Este método se invocará cada vez que cambie el estado del checkbox.

```
checkboxbox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked) {

        }
});
```

El método **onCheckedChanged** recibe dos variables:

- El botón cuyo estado ha cambiado.
- El nuevo estado del botón.

## 5. Imagen

Para añadir imágenes en una aplicación fácilmente, podemos utilizar el elemento **ImageView**.

```
<ImageView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_launcher"
    android:contentDescription="@string/app_name"/>
```

El atributo **android:src** permite especificar la imagen que se desea mostrar.

El atributo **android:contentDescription** permite proporcionar una breve descripción de la imagen mostrada (utilizado para accesibilidad).

Ver programa: android-imagen-texto



## Programación de eventos en botones y CheckBox.

El clic es una acción indispensable en la gestión de la interacción del usuario con su aplicación.

La gestión del clic puede hacerse de dos formas distintas:

- Gestionar el clic en los botones, de forma separada.
- Hacer que su actividad implemente la interfaz **onClickListener**.

Para ilustrar mejor este concepto, a continuación tenemos el ejemplo de una interfaz que dispone de dos botones.

### Gestionar el clic en los botones de forma separada

```
Button btn1 = (Button) findViewById(R.id.btn1);
btn1.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        // Gestión del clic en el botón 1
    }
});
```

```
Button btn2 = (Button) findViewById(R.id.btn2);

btn2.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        // Gestión del clic en el botón 2
    }
});
```

Se han declarado, en la vista, dos botones inicializados mediante el método **findViewById**.

La llamada al método **setOnClickListener** permite añadir un listener al clic de los botones.

Este método recibe como parámetro un nuevo listener que permite sobrecargar el método **onClick**. Este listener se crea mediante el constructor de la clase **onClickListener**.

Todo el tratamiento realizado en el clic debe implementarse en el método **onClick**, que recibirá como parámetro la vista que ha recibido dicho clic.

### La actividad implementa la interfaz onClickListener



```

public class ClickListenerMethod2Activity extends Activity
implements OnClickListener {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Button btn1 = (Button) findViewById(R.id.btn1);
        btn1.setOnClickListener(this);

        Button btn2 = (Button) findViewById(R.id.btn2);
        btn2.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.btn1:
                // Gestión del clic en el botón 1
                break;
            case R.id.btn2:
                // Gestión del clic en el botón 2
                break;
            default:
                break;
        }
    }
}

```

El primer paso consiste en hacer que la actividad implemente la clase **OnClickListener**, lo que nos permite sobrecargar el método **onClick**.

En este método, se comprueba cuál es el identificador del botón pulsado y, en función de su valor, se ejecuta la acción correspondiente.

El último paso consiste en obtener los dos botones y especificar que la instancia de nuestra clase es la gestora del evento clic.

Ver programa: android-imagen-texto



Ver programa: [android-check-radio](#)

