INTERFACE DE USUARIO: COMPONENTES BÁSICOS

Programación multimedia
y dispositivos móviles
2º DAM

ÍNDICE DE CONTENIDOS

ÍNDICE DE CONTENIDOS

1. TEXTVIEW

Sirve para mostrar un texto en pantalla.Para trabajar con TextView en modo texto, hay que añadir en el layout correspondiente la etiqueta:

```
<TextView
        Parámetros >
</TextView>
```

Parámetros más importantes:

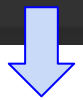
- **android:text** → define cuál será el texto que se mostrará en pantalla.
- **android:background** → establece el color del fondo.
- **android:textColor** → asigna el color del texto.
- **android:textSize** → determina el tamaño del texto (medida sp)
- **android:textStyle** → elige el estilo del texto (normal, cursiva, negrita)
- **android:typeface** → fija el tipo de letra que podemos usar (sans, monospace, serif)

1. TEXTVIEW

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView"
    android:textColor="@color/colorPrimary"
    android:text="@string/texto">
</TextView>
```

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView2"
    android:textColor="@color/colorPrimary"
    android:text="@string/texto"/>
```

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/textView"
    android:textColor="@color/colorPrimary"
    android:text="@string/texto"
    android:textStyle="bold|normal"
    />
```



Con | podemos indicar dos tipos a la vez.

1. TEXTVIEW

Para utilizar un textView desde java, primero tendremos que instanciarlo buscando su ID:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView">
</TextView>
```

```
TextView textView = findViewById(R.id.textView);
```

Una vez instanciado, podemos utilizar sus propiedades desde java.

```
TextView textView = findViewById(R.id.textView);

textView.setText(R.string.texto);
textView.setTextColor(ContextCompat.getColor( context: this, R.color.colorPrimary));
```

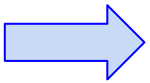
1. TEXTVIEW

Desde java podemos añadir texto al TextView mediante el método append.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView"
    android:textColor="@color/colorPrimary"
    android:text="@string/texto">
</TextView>
```

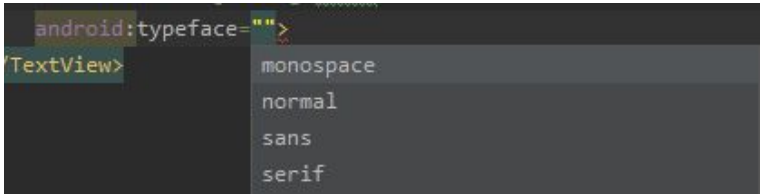
```
TextView textView = findViewById(R.id.textView);

textView.append("\n" + R.string.app_name);
```

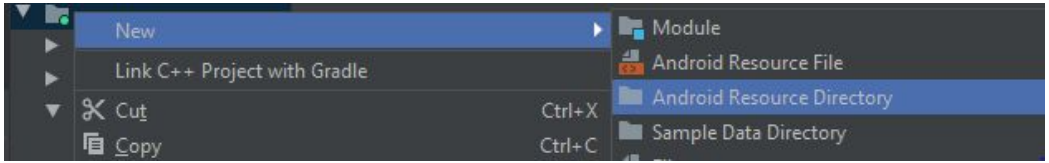


1. TEXTVIEW

Las fuentes que por defecto nos permite usar Android son muy limitadas, pero podemos añadir cualquier fuente siempre que la pongamos como asset.

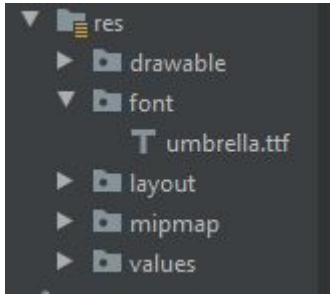


Desde app, creamos un nuevo directorio Android, y seleccionamos font.



1. TEXTVIEW

Añadimos la fuente a la carpeta creada, y ya podremos utilizarla.




```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView"
    android:textColor="@color/colorPrimary"
    android:text="@string/texto"
    android:fontFamily="@font/umbrella">
</TextView>
```

Desde Java, nos permite utilizar el método getFont a partir de la versión OREO (API 26), por lo que si nuestro minSdkVersion es inferior, nos pedirá realizar la comprobación necesaria para utilizar el método.

```
if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.O) {
    Typeface miFuente = getResources().getFont(R.font.umbrella);
    textView.setTypeface(miFuente);
}
```

Si ejecutamos esto en un emulador API 24, se verá así:

2. BUTTON

La clase Button va a permitir crear botones en la interfaz gráfica. Esta clase hereda de TextView, por lo que tiene toda su funcionalidad.

El evento que se suele utilizar en este tipo de objetos es pulsarlo, y su evento se denomina OnClickListener.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/b1"
    android:textColor="@color/colorPrimary"
    android:text="@string/pulsa"
    android:onClick="pulsarBoton">
</Button>
```

```
public void pulsarBoton(View view) {

    //instrucciones que se ejecutaran al pulsar un botón

}
```

2. BUTTON

Otro forma de manejar los botones (más profesional) es sin indicar en XML el evento, y hacerlo directamente desde java.

```xml
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/b1"
    android:textColor="@color/colorPrimary"
    android:text="@string/pulsa">
</Button>
```

```java
Button button = findViewById(R.id.b1);

button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //instrucciones a realizar
    }
});
```

2. BUTTON

Otro forma de manejar los botones (más profesional) es sin indicar en XML el evento, y hacerlo directamente desde java.

```xml
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/b1"
    android:textColor="@color/colorPrimary"
    android:text="@string/pulsa">
</Button>
```

```java
Button button = findViewById(R.id.b1);

button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //instrucciones a realizar
    }
});
```

2. BUTTON

Por último, la forma más profesional cuando se tienen varios componentes con el evento OnClickListener en una misma actividad, es hacer que sea la propia Activity sea la que maneje los botones que existen en ella.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/aceptar"
    android:textColor="@color/colorPrimary"
    android:text="@string/enviar">
</Button>

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/cancelar"
    android:textColor="@color/colorPrimary"
    android:text="@string/cancelar">
</Button>
```

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button b1 = findViewById(R.id.aceptar);
        Button b2 = findViewById(R.id.cancelar);

        b1.setOnClickListener(this);
        b2.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()){
            case R.id.aceptar:
                //instrucciones a realizar al pulsar el botón aceptar.
                break;
            case R.id.cancelar:
                //instrucciones a realizar al pulsar el botón cancelar.
                break;
            default:
                break;
        }
    }
}
```

3. EDITTEXT

Objeto para leer entradas de texto, también hereda de TextView, por lo que tiene todas sus propiedades.

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/editText"
    android:hint="@string/nombre"
    />
```

Con Java, podemos obtener el texto introducido con el método getText()

```
EditText editText = findViewById(R.id.editText);

String nombre = editText.getText().toString();
```

Mediante el atributo inputType podemos indicar el tipo de contenido que puede introducir el usuario:

- **textUri** → texto que se usará como URI
- **textEmailAddress** → texto que se usará como dirección de correo
- **textPersonName** → nombre de una persona
- **textPassword** → contraseña
- **textVisiblePassword** → contraseña que se mostrará

3. EDITTEXT

- **number** → entrada numérica (enteros)
- **numberDecimal** → entrada numérica (decimal)
- **numberPassword** → contraseña con entrada numérica
- **phone** → entrada de un número de teléfono
- **date** → texto tratado como fecha
- **time** → texto tratado como hora
- **textMultiLine** → permite multilínea
- **textCapSentences** → pone en mayúsculas la primera letra de cada frase
- **textCapWords** → pone en mayúsculas la primera letra de cada palabra
- **textNoSuggestions** → desactiva la escritura predictiva

3. EDITTEXT

Otros atributos:

- **android:hint** → pone un texto predefinido (aparece en gris y desaparecerá al escribir texto)
- **android:lines** → limita el número de líneas
- **android:digits** → limita el número de dígitos (numérico)
- **android:maxLength** → limita el número de caracteres

3. EDITTEXT

El evento para manejar el EditText es TextWatcher, y se usa el método addTextChangedListener.

```
EditText editText = findViewById(R.id.editText);

editText.addTextChangedListener(new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
        //nos responde cuando el texto está cambiando
    }

    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
        //nos responde antes de que el texto cambie
    }

    @Override
    public void afterTextChanged(Editable s) {
        //nos responde despues de que el text cambie
    }
});
```

4. IMAGEVIEW

Componente para visualizar imágenes en la pantalla.

Aparte de los atributos para el tamaño (layout_width y layout_height), podemos indicar la imagen en XML si está en los recursos (@drawable) o en los recursos propios de android (@android:drawable) con el atributo **android:src**.

Con el atributo **android:contentDescription** podemos establecer un texto que se vea si no es posible cargar la imagen.

```
<ImageView
    android:id="@+id/imagen"
    android:layout_width="300dp"
    android:layout_height="300dp"
    android:src="@drawable/logo"
    android:contentDescription="@string/imagen"/>
```

```
ImageView imagen = findViewById(R.id.imagen);

imagen.setImageDrawable(ContextCompat.getDrawable( context: this, R.drawable.logo));
```

5. IMAGEBUTTON

Botón que permite colocar una imagen en vez de un texto. Esta clase hereda de ImageView.

```xml
<ImageButton
    android:id="@+id/imagen"
    android:layout_width="300dp"
    android:layout_height="300dp"
    android:src="@drawable/logo"/>
```

```java
public class MainActivity extends AppCompatActivity implements View.OnClickListener {


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ImageButton imageButton = findViewById(R.id.imagen);

        imageButton.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.imagen:
                //instrucciones a ejecutar al pulsar la imagen
                break;
            default:
                break;
        }
    }
}
```

6. CHECKBOX

Se utiliza para marcar o desmarcar opciones. La posición de partida se especifica en el atributo checked:

- android:checked:”true” → marcado por defecto
- android:checked:”false” → desmarcado por defecto.

El evento para manejar desde Java el cambio de valor de un CheckBox es OnClickListener, aunque es más profesional utilizar OnCheckedChangeListener.

```
<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/check"
    android:text="@string/prueba"
    android:checked="false" />
```

6. CHECKBOX

```
CheckBox checkBox = findViewById(R.id.check);

checkBox.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (((CheckBox) v).isChecked()) {
            //instrucciones a ejecutar cuando se marca
        } else {
            //instrucciones a ejecutar cuando se desmarca
        }
    }
});
```

```
CheckBox checkBox = findViewById(R.id.check);

checkBox.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            //instrucciones a ejecutar cuando se marca
        } else {
            //instrucciones a ejecutar cuando se desmarca
        }
    }
});
```

Más profesional

7. RADIOBUTTON

Es un control de selección para elegir una opción de entre un conjunto. Es un control de exclusión mutua, es decir, la selección de uno implica el cambio automático del que previamente estaba seleccionado. Se agrupan mediante un RadioGroup, de manera que, de los que estén incluidos en este grupo, sólo uno de ellos puede estar marcado.

```
<RadioGroup
    android:id="@+id/group1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/opcion1"
        android:id="@+id/radio1"
        android:checked="true"/>

    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/opcion2"
        android:id="@+id/radio2"/>

</RadioGroup>
```

7. RADIOBUTTON

Declarar radioGroup y radioButtons en Java:

```
RadioGroup group = findViewById(R.id.group1);
RadioButton r1 = findViewById(R.id.radio1);
RadioButton r2 = findViewById(R.id.radio2);
```

Conocer que radioButton está marcado:

```
group.getCheckedRadioButtonId();
```

Desmarcar todos los radioButton:

```
group.clearCheck();
```

Para marcar un radioButton, se puede hacer desde Java o desde XML.

```
group.check(R.id.radio1);
```

```
<RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Opción 1"
    android:id="@+id/radio1"
    android:checked="true"/>
```

7. RADIOBUTTON

Por último, el evento para manejar el control desde Java del cambio de botón pulsado es OnClick en cada botón, aunque lo ideal es manejarlo desde el grupo, con el evento OnCheckedChangeListener.

```java
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    TextView textView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        textView = findViewById(R.id.textView);
        RadioButton b1 = findViewById(R.id.radio1);
        RadioButton b2 = findViewById(R.id.radio2);

        b1.setOnClickListener(this);
        b2.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.radio1:
                textView.setText(R.string.opcion1);
                break;
            case R.id.radio2:
                textView.setText(R.string.opcion2);
                break;
            default:
                break;
        }
    }
}
```

```java
RadioGroup group = findViewById(R.id.group1);

group.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(RadioGroup group, int checkedId) {

    }
});
```

Se guarda el id del radioButton que se ha pulsado

7. RADIOBUTTON

```
textView = findViewById(R.id.textView);
RadioGroup group = findViewById(R.id.group1);

group.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(RadioGroup group, int checkedId) {
        switch (checkedId) {
            case R.id.radio1:
                textView.setText(R.string.opcion1);
                break;
            case R.id.radio2:
                textView.setText(R.string.opcion2);
                break;
            default:
                break;
        }
    }
});
```

8. TOGGLEBUTTON

Tipo de botón en el que se establece un texto para cada estado (encendido y apagado) y además se ve un diseño diferente (en el modo ON añade una línea del color ***colorAccent***). El diseño se puede modificar con los estilos.

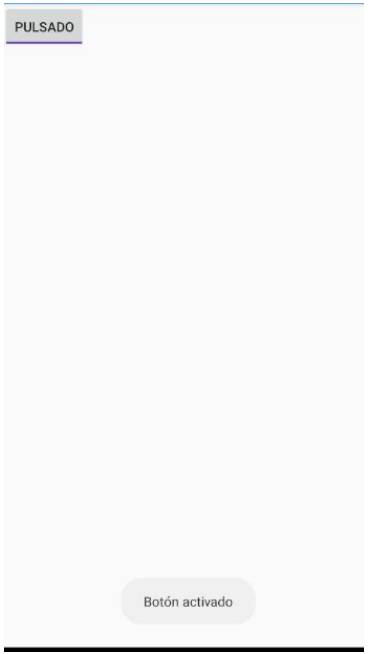
```
<ToggleButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="PULSADO"
    android:textOff="NO PULSADO"/>
```

8. TOGGLEBUTTON

El evento de este botón es OnCheckedChangeListener.

```
ToggleButton toggle = findViewById(R.id.toggle);

toggle.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked == true) {
            Toast.makeText(getApplicationContext(), R.string.activ, Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(getApplicationContext(), R.string.desactiv, Toast.LENGTH_LONG).show();
        }
    }
});
```



9. SWITCH

Es un control muy parecido a ToggleButton (heredan de la misma clase), y también tiene dos estados:

Encendido → marcado

Apagado → desmarcado

La diferencia con ToggleButton es que simula un botón con control deslizante.

```
<Switch
    android:id="@+id/sw1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

OFF:  Por defecto (si no se indica estado)

ON:  Coge el color colorAccent

```
<Switch
    android:id="@+id/sw1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"/>
```

9. SWITCH

El evento para manejar Switch desde Java es OnCheckedChangeListener.

```
Switch pulsador = findViewById(R.id.sw1);

pulsador.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            //instrucciones cuando se enciende
        } else {
            //instrucciones cuando se apaga
        }
    }
});
```

10. PROGRESSBAR

Control que se suele utilizar con las tareas asíncronas, que sirve para informar al usuario de la evolución de un proceso. De esta forma, se rellena el tiempo de espera entre la acción del usuario y la respuesta de la aplicación.

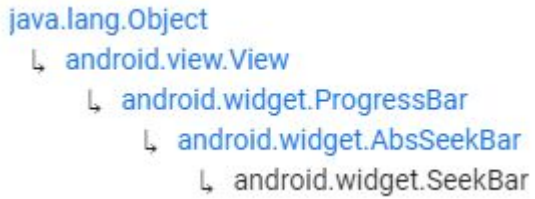
Existen dos formas de diseño (circular o barra).

```
<ProgressBar
    android:id="@+id/p1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="@style/Widget.AppCompat.ProgressBar"/>
```

```
<ProgressBar
    android:id="@+id/p2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:progress="10"
    style="@style/Widget.AppCompat.ProgressBar.Horizontal"/>
```

11. SEEKBAR

Este control pertenece al tipo de controles de selección en los que el usuario elige un valor numérico en un rango de valores predefinidos en la aplicación.



11. SEEKBAR

Atributos:

- **android:max** → define el valor máximo
- **android:min** → define el valor mínimo
- **android:progress** → fija el punto de partida del control
- **android:thumb** → personaliza la imagen del elemento deslizable
- **android:rotation** → permite girar el control y ponerlo verticalmente
- **android:progressDrawable** → personaliza el aspecto de la barra de desplazamiento

11. SEEKBAR

```
<SeekBar
    android:id="@+id/seek"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:max="20"
    android:min="10"
    android:progress="15"
    android:thumb="@android:drawable/btn_star"
    android:progressDrawable="@android:color/holo_orange_light"/>
```

11. SEEKBAR

El evento de este componente es OnSeekBarChangeListener que, al igual que EditText, tiene 3 métodos para manejar.

```
SeekBar barra = findViewById(R.id.seek);

barra.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
    @Override
    public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
        //instrucciones a ejecutar por cambio de valor
    }

    @Override
    public void onStartTrackingTouch(SeekBar seekBar) {
        //instrucciones a ejecutar para inicio de cambio
    }

    @Override
    public void onStopTrackingTouch(SeekBar seekBar) {
        //instrucciones a ejecutar para final de cambio
    }
});
```

11. SEEKBAR

También tiene atributos para cambiar su diseño:

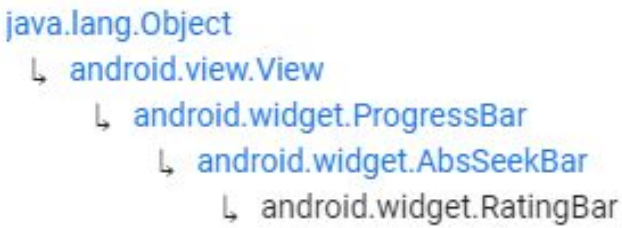
- **android:progressTint** → color de la barra hasta la selección
- **android:thumbTint** → color del elemento para seleccionar
- **android:progressBackgroundTint** → color de la barra después de la selección

Si se quiere mostrar cada una de las marcas, hay que añadir el estilo SeekBar.Discrete

```
<SeekBar
    android:id="@+id/seek"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:max="20"
    android:min="10"
    android:progress="15"
    android:thumbTint="@color/colorAccent"
    android:progressTint="@color/colorPrimaryDark"
    android:progressBackgroundTint="@color/colorPrimary"
    style="@style/Widget.AppCompat.SeekBar.Discrete" />
```

12. RATINGBAR

Es otro control de selección, pero como su propio nombre indica, está diseñado para dar una puntuación.



12. RATINGBAR

Atributos:

- **android:numStars** → número de estrellas
- **android:rating** → puntuación por defecto
- **android:stepSize** → valor de los incrementos

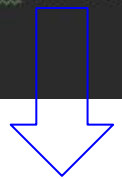
```
<RatingBar
    android:id="@+id/rating"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:numStars="7"
    android:rating="1"
    android:stepSize="0.5"/>
```

12. RATINGBAR

El evento de este componente es OnRatingBarChangeListener.

```
RatingBar rating = findViewById(R.id.rating);

rating.setOnRatingBarChangeListener(new RatingBar.OnRatingBarChangeListener() {
    @Override
    public void onRatingChanged(RatingBar ratingBar, float rating, boolean fromUser) {
        //instrucciones que se ejecutan cuando se cambia el valor
    }
});
```



El nuevo valor seleccionado

13. DUDAS Y PREGUNTAS