

## 5.1. Introducción

Después de haber trabajado con los distintos tipos de layouts, en este capítulo se van a estudiar los diferentes controles que Android pone a disposición del programador para crear elegantes y funcionales interfaces. Se agruparán por la tipología del control, comenzando por los más básicos, para ver después algunos de los más complejos. Son numerosos, aun sin incluir librerías de terceras partes, por lo que difícilmente podrían ser incluidas todas sus propiedades en este libro.

## 5.2. TextView

Este control, de fácil construcción, quizás sea uno de los más usados, tiene la única función de mostrar un texto en pantalla. Sus parámetros más importantes, además de los comunes a otros objetos View, son:

- *android:text*. Define cuál será el texto que se mostrará en pantalla.
- *android:background*. Establece el color de fondo.
- *android:textColor*. Asigna el color del texto.
- *android:textSize*. Determina el tamaño del texto (es aconsejable utilizar unidades de medida sp).
- *android:textStyle*. Elige el estilo del texto: normal, negrita (bold) o cursiva (italic).
- *android:typeface*. Fija el tipo de letra por el que podemos optar: sans, monospace o serif.

Además de estos, disponemos de muchos otros, que seguramente se utilicen en menos ocasiones, como *android:textAppearance*, *android:shadowColor*, *android:shadowRadius*...

Para Incluirlo en el fichero de la vista (XML) de la aplicación tan solo deberá ser añadido dentro de un objeto ViewGroup (layout) de la siguiente manera:

```
<TextView  
    android:id="@+id/texto"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Texto para mostrar"  
/>
```

Para manipular este objeto de texto desde Java, primero deberá asociarse a la vista, para lo que es necesario importar la librería del widget (esto suele hacerlo de manera automática Android Studio) que da acceso a los objetos incluidos en el interface. En este caso, se necesitará *import android.widget.TextView;*

### TEN EN CUENTA

- ✓ Si Android Studio deja librerías sin importar, puede hacerse de forma manual, posicionando el cursor sobre los objetos no reconocidos y pulsando Alt+Intro.

A continuación, se hará de la siguiente manera:  
que como ya anteriormente, se

```
TextView miTexto = (TextView) findViewById(R.id.texto);
miTexto.setText("Nuevo texto para mostrar");
```

Donde miTexto es el nombre que adquirirá esta instancia en Java y R.id.texto es el nombre con el que lo identificamos en el fichero XML de la vista asociada a los atributos (android:id="@+id/texto"). Una vez hecho esto, se pueden obtener (get) o modificar (set) los atributos de texto. Y así, si se quisiera cambiar el contenido de la etiqueta de texto, tan solo se lo escribiría:

- `miTexto.setTextColor(Color.argb(255, 255, 0, 0));` tres dígitos en decimal con transparencia.
  - `miTexto.setTextColor(Color.parseColor("#ff0000"));` nos permite obtener el color de una cadena.
- Por último, con el tiempo, es aconsejable acostumbrarse, por su utilidad, a incluir los parámetros de los atributos en los recursos, lo que puede hacerse también con el color. En este caso, se procedería de la siguiente forma:
- ```
miTexto.setTextColor(getResources().getColor(R.color.micolor));
```

Previamente, se debe modificar el fichero res/values/color.xml y añadirle una nueva línea, definiendo este color personalizado.

```
<color name="micolor">#ff0000</color>
```

El utilizar una u otra forma de aplicar el color, de las vistas anteriormente, dependerá de la propia práctica y, aunque parezcan suficientes, si se busca en la página de Android Developer, se podrán encontrar algunas más.

## 5.2.2. Añadir nuevo texto

Como se ha visto anteriormente, se puede cambiar el contenido de la etiqueta de texto tan solo con setText, pero si lo que se desea es añadir más texto al ya existente, se hará mediante el comando append(), incluyendo como parámetro el texto que se va a añadir. Si previamente se desea hacer un salto de línea, se debe incorporar el modificado \n. Un ejemplo podría ser:

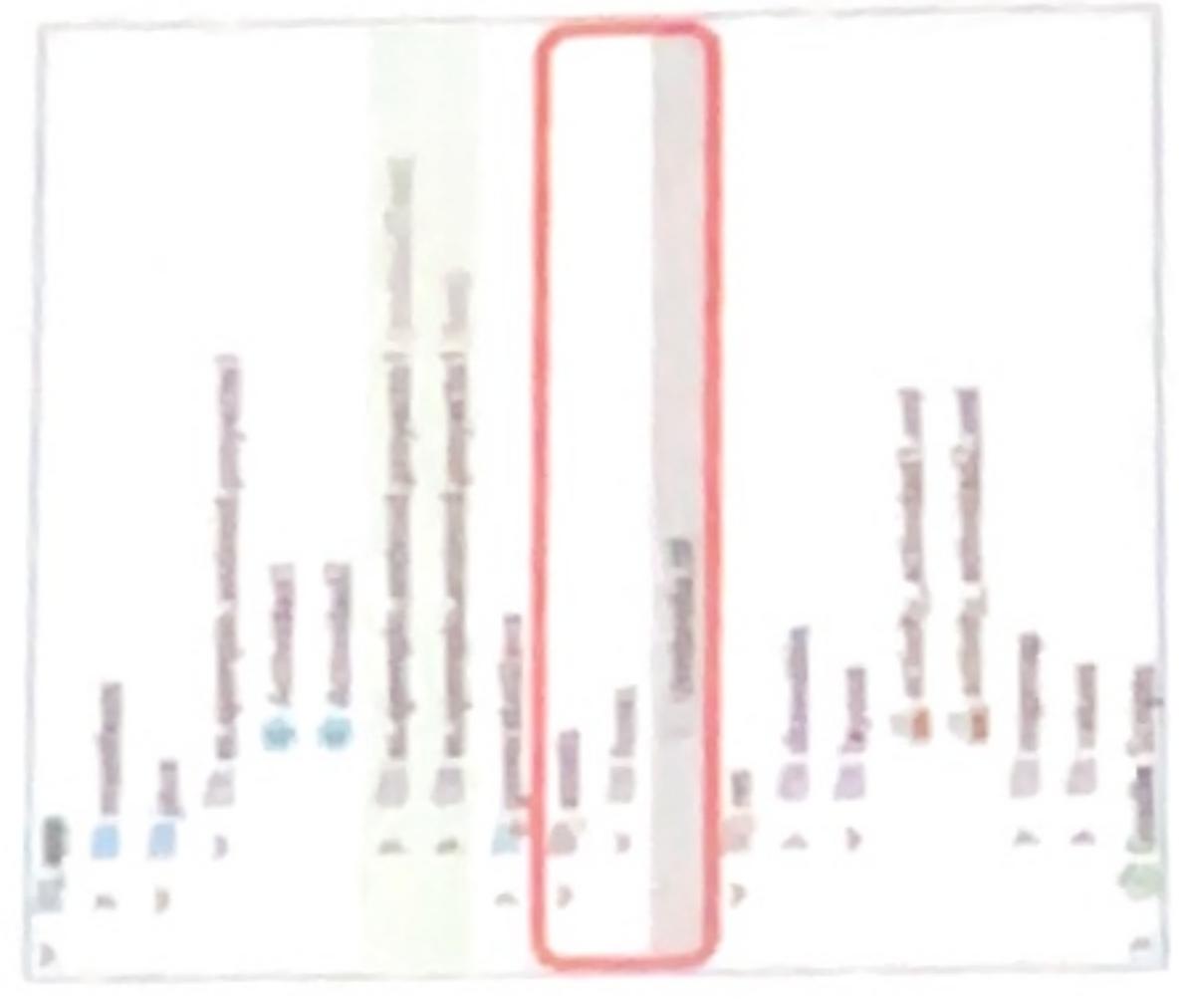
```
miTexto.append("\n Nueva linea de texto");
```

## 5.2.1. Cambiar las propiedades del texto

Puede ser manipulado un gran número de propiedades de este objeto desde Java. De modo similar a lo visto anteriormente, se deben proceder desde Java, utilizando la librería android.graphics.Typeface. Las fuentes que se desean utilizar (que deberán ser true type) han de ser incluidas previamente en el directorio assets/fonts.

Posteriormente, se instanciará la fuente (mediante la librería antes citada) y se aplicará al texto. El resultado podría quedar así.

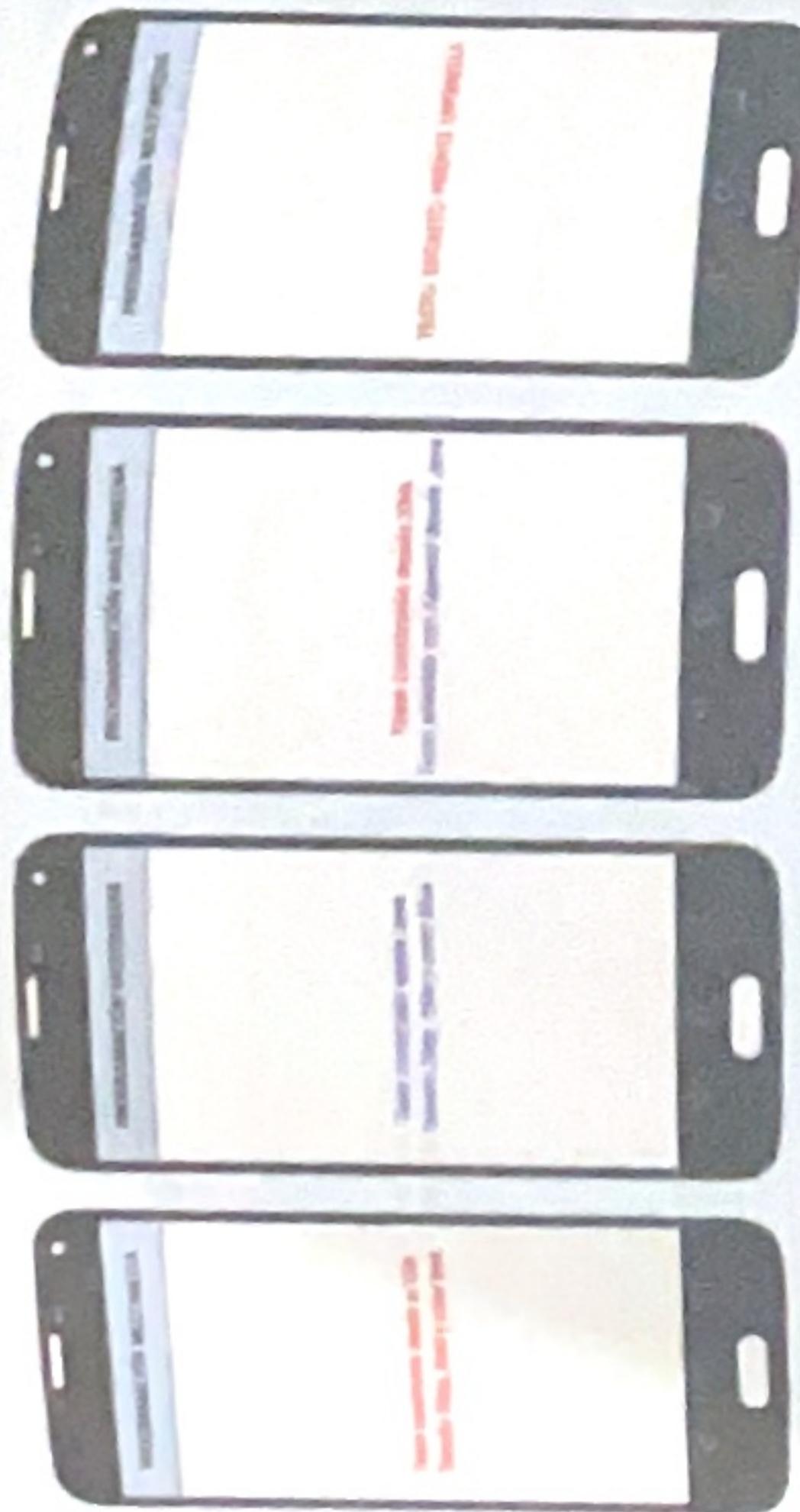
- ```
Typeface miFuente =
Typeface.createFromAsset(getAssets(), "fonts/umbrella.ttf");
miTexto.setTypeface(miFuente);
```
- `miTexto.setTextColor(0xffff0000);` directamente en hexadecimal incluyendo la transparencia (alfa).
  - `miTexto.setTextColor(Color.RED);` como constante con la librería android.
  - `miTexto.setTextColor(Color.rgb(255, 00, 00));` tres dígitos en decimal (rango 0-255).



**Figura 5.1**  
Directorio de fuentes en recursos.

que como ya anteriormente, se hará de la siguiente manera:  
que como ya anteriormente, se

Realiza las siguientes actividades de texto. Para el posicionamiento en pantalla, recuerda que puedes utilizar los mismos atributos estudiados en los layouts.



Android permite variedad de animaciones (que después se verán), pero todas ellas tienen la misma forma de estructurarse desde Java. El proceso de construcción es el siguiente:

```
Animation miAnimacion = AnimationUtils.loadAnimation(this, R.anim.animacion);  
  
miAnimacion.setRepeatMode(Animation.RESTART);  
miAnimacion.setRepeatCount(20);  
  
miTexto.startAnimation(miAnimacion);
```

Donde R.anim.animacion indica la ruta y el fichero que contiene la secuencia de animación. Luego se ha de definir el modo de repetición (lo normal es que se reinicie al llegar al final) y el número de repeticiones.

Si se desarrolla para API 25 o superior se puede crear directamente la carpeta res/recursos) y acceder a la fuente desde XML mediante android:fontFamily="@font/fuente" o desde Java con Typeface miFuente = getResources().getFont(R.font.umbrella). En este caso se deberán evitar las mayúsculas.

#### RECUERDA

- ✓ Existen varias alternativas para instanciar los objetos de texto, una sería la vista anteriormente, TextView [miTexto= [TextView] findViewById(R.id.texto)]. También podría declararse primero el objeto [TextView mitexto,] y, posteriormente, las instancias [miTexto= [TextView] findViewById(R.id.mitexto)].
- ✓ Sea cual sea la forma utilizada, se ha de tener en cuenta la visibilidad que tenga cada uno de los objetos declarado (public, private, protected).

En cuanto al tipo de animaciones que pueden ser utilizadas, las más frecuentes son:

- Translación (translate).
- Rotación (rotate).
- Escalado (scale).
- Aparición (alpha).

Según la utilizada, se tendrán que definir unos u otros parámetros (no necesariamente hay que utilizarlos todos). A modo de ejemplo: si se quisiera realizar un desplazamiento del texto por la pantalla (translate), deberán definirse el punto de origen, punto final (tanto en coordenadas X como Y), duración (en milisegundos) y la forma que se desea que tenga el movimiento (interpolación). Este último apartado es el más complejo, pero también el más efectista, ya que permite numerosas alternativas:

- linear\_interpolator
- accelerate\_interpolator
- decelerate\_interpolator
- accelerate\_decelerate\_interpolator
- anticipate\_interpolator
- bounce\_interpolator
- overshoot\_interpolator

Android permite aplicar efectos de animación a los objetos incluidos en el interface, para lo que a nuestra disposición dos librerías que lo facilitan y que, como en otras ocasiones, ser importadas:

```
<?xml version="1.0" encoding="utf-8"?>  
<translate  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:interpolator="#android:anim/accelerate_interpolator"  
    android:fromXDelta="-100%p"  
    android:toXDelta="100%p"  
    android:duration="4000"  
/>
```

El efecto de texto que se desea aplicar se puede definir previamente en un fichero situado en el directorio res/anim. Este directorio no suele aparecer por defecto y posiblemente deba ser creado haciendo click con el botón derecho sobre el directorio res.

## 5.2.4. Animación del texto

Un ejemplo de fichero R.anim.animacion para una transición podría ser:

```
<?xml version="1.0" encoding="utf-8"?>
<rotate xmlns:android="http://schemas.android.com/apk/res/android"
    android:values="@+id/anim"
    android:duration="2000"
    android:interpolator="@+id/linear_interpolator"
    android:pivotX="50%"
    android:pivotY="50%"
    android:toDegrees="360" />
```

Para crear un efecto de aparición-desaparición, podría usarse lo siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<alpha xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="3000"
    android:fromAlpha="0.0"
    android:interpolator="@+id/anim/linear_interpolator"
    android:toAlpha="1.0" />
```

Android también permite crear secuencias de animación, que se pueden definir en un fichero XML en el que se ha estado trabajando hasta ahora. El aspecto que debemos es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android
    android:interpolator="@+id/anim/linear_interpolator"
    <rotate
        />
    <alpha
        />
</set>
```

Entre las correspondientes etiquetas se incluyen los parámetros deseados, así como animaciones como se deseé.

Por último, en el apartado animaciones también es posible definir animaciones directas desde Java (aunque es más complejo). Para ello, se deberá instanciar la clase Animation (esta modalidad de animación que se va a emplear (AlphaAnimation, ScaleAnimation, TranslateAnimation, RotateAnimation). Posteriormente, se configura esta animación y, después, al texto. Un ejemplo podría ser el siguiente:

```
AnimationSet animacion = new AnimationSet(true);
animacion.add(animacion_aparicion);
animacion.setDuration(2000);
animacion.setFillAfter(true);
animacion.setRepeatCount(20);
animacion.setStartDuration(1, animation);
```

En la página oficial de Android pueden consultarse los parámetros que se deben para cada tipo de animaciones.



## Actividad propuesta 5.2

En la plataforma de Editorial Síntesis encontrarás diferentes ficheros de animaciones, incorpóralos a tu proyecto y observa el efecto de cada uno. También puedes modificar los atributos hasta que te acabes familiarizando con los efectos de animación.

### 5.3. 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. Al igual que el caso anterior, puede ser creado directamente el objeto desde la vista en el fichero XML, procediendo de la siguiente forma.

```
<Button
    android:id="@+id/miBoton"
    android:text="BOTÓN"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:onClick="pulsado"
/>
```

Para manejar los eventos de este objeto (habitualmente el pulsarlo), puede hacerse de dos maneras, la primera desde XML, definiendo un evento onClick, al que luego accederá un método desde el fichero Java.

El método que responderá en Java debe venir definido de la siguiente forma:

```
public void pulsado (View view) {
    // Código que se ejecutará
}
```

La segunda manera, y más profesional, es mediante un manejador de eventos desde código Java. Para ello, se debe poner un identificador al botón en la vista (en este caso, miBoton), y luego instanciarlo y asociarle un manejador de eventos (escuchador). Sería de la siguiente forma:

```
Button miBotonJava = (Button) findViewById(R.id.miBoton);
miBotonJava.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Código que se ejecutará
    }
});
```

Algo más avanzado, para lo cual se ha de implementar el escuchador en la clase principal de la aplicación, para que sea el que responda a los eventos.

```
public class ClasePrincipal extends AppCompatActivity implements View.OnClickListener {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.vista_principal);
        Button miBotonJava = (Button) findViewById(R.id.miBoton);
        miBotonJava.setOnClickListener(this);
    }
    @Override
    public void onClick(View view) {
        // Código que se ejecutará
    }
}
```

En este caso, puede asociarse el manejador de eventos a más de un botón.

```
public class ClasePrincipal extends AppCompatActivity implements View.OnClickListener
```

```
    {
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.vista_principal);
        }
    }
}
```

```
    Button miBoton1Java = (Button) findViewById(R.id.miBoton1);
    Button miBoton2Java = (Button) findViewById(R.id.miBoton2);
    miBoton1Java.setOnClickListener(this);
    miBoton2Java.setOnClickListener(this);
}
```

```
    @Override
    public void onClick(View view) {
        switch (view.getId()) {
            case R.id.miBoton1:
                // Código que se ejecutará al pulsar el botón 1
                break;
            case R.id.miBoton2:
                // Código que se ejecutará al pulsar el botón 2
                break;
        }
    }
}
```

No se debe olvidar que deben estar previamente diseñados e identificados los tres componentes necesarios para esta funcionalidad.

- import android.view.View;
- import android.view.View.OnClickListener;
- import android.widget.Button;

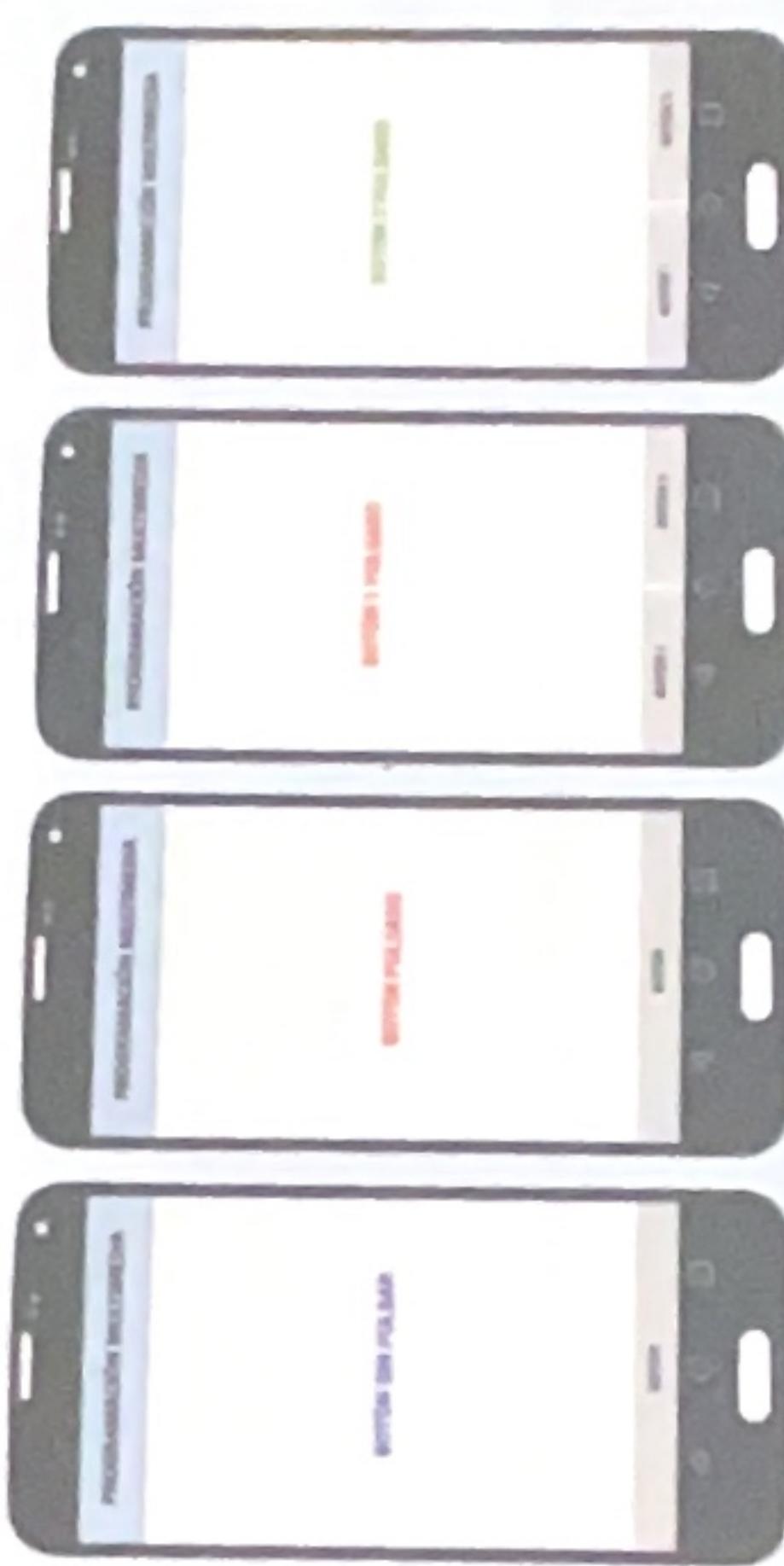
### Investiga

Todas las vistas permiten registrar eventos de distinto tipo, entre los que tenemos los siguientes. Investiga y practica con cada uno de ellos.

```
setOnClickListener
setOnTouchListener
setOnDragListener
setOnFocusChangeListener
setOnTouchListener
```

### Actividad propuesta 5.3

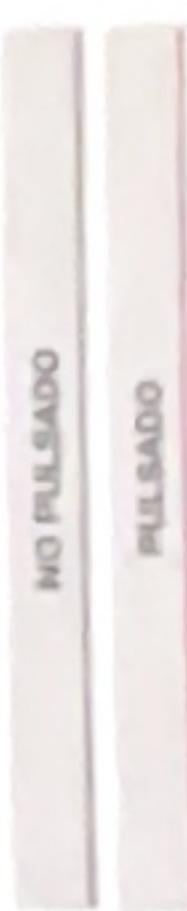
Realiza las siguientes actividades con botones. El objetivo es que se modifique el contenido de una etiqueta de texto tras pulsar un botón, en el primer caso, mediante un método que responde a un evento onClick, y el segundo caso, con un escuchador en Java.



### 5.4. ToggleButton

Es una variante del anterior, en el que puede personalizarse el estilo, tanto para cuando se encuentre pulsado como para cuando no lo esté.

En este ejemplo, variará el texto según la opción en la que se encuentre (si se ejecuta, se observará que, además del texto, también cambia el aspecto del botón).



```
<ToggleButton
    android:id="@+id/miToggle"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textOff="No PULSADO"
    android:textOn="PULSADO" />
```

Figura 5.2  
Ejemplo de ToggleButton.

- ✓ El escuchador de eventos personalizado podrá ser del tipo `OnCheckedChangeListener`. Y escuchará el evento cambio de estado (`onCheckedChanged`).
- ✓ También puede personalizarse este botón definiendo el atributo `buttonMode`, incluso creando un recurso XML que personalizado.

## 5.5. ImageButton

Es otro objeto de interfaz que hereda de `ImageView` (se estudiará más tarde), sin embargo, a estos no es posible ponerle texto. Tanto para crear botones ilustrados con una imagen (que debe previamente ser incluido en el archivo `res/drawable`), como para manejarlos de los manejadores de eventos estudiados anterior, puede utilizarse cualquiera de los manejadores de eventos estudiados.

```
<ImageButton
    android:id="@+id/miBoton"
    android:src="@drawable/imagen"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:onClick="pulsado" />
```

## 5.6. EditText

Es el objeto más simple que nos ofrece Android para leer entradas de texto, todas las propiedades de `TextView`.

```
<EditText
    android:id="@+id/miTexto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
```

Para manipular este objeto desde Java, se necesita el widget incluido en la clase `EditText`, y se deberá proceder de la siguiente manera:

```
EditText textoLeido = (EditText) findViewById(R.id.miTexto);
String cadenaTexto = textoLeido.getText().toString();
```

Por defecto, `EditText` lee texto plano y asocia el teclado textual del dispositivo. Dispone de numerosas opciones, gracias al modificador `inputType`, según la aplicación desarrollada.

- `textUri`. Texto que se usará como URI.
- `textEmailAddress`. Texto que se usará como dirección de correo.
- `textPersonName`. Nombre de una persona.



TOMA NOTA

En caso de querer utilizar más de una opción, se utilizará el separador |.

- `textPassword`. Contraseña.
- `textVisiblePassword`. Contraseña que se mostrará.
- `number`. 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.

Otras interesantes opciones, que posiblemente se usen en más de una ocasión, son:

- `android_hint`. Pone un texto predefinido (que aparecerá en gris).
- `android_lines`. Limita el número de líneas.
- `android_digits`. Limita el número de dígitos (numérico).
- `android_maxLength`. Limita el número de caracteres.

Investiga

Algo más compleja sería la opción `android:imeOptions`, que define el botón de opción del teclado entre las siguientes posibilidades. Investiga y practica con cada uno de ellos.

```
actionNone: al botón muestra return
actionNext: si igualmente
actionSearch: anterior
actionSend: enviar
actionDone: no especificada
```

Un ejemplo de todo lo anterior podría ser el siguiente:

```
<EditText
    android:id="@+id/correo"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Avenida de Madrid"
    android:imeOptions="actionSearch"
    android:inputType="textPostalAddress|textEmailAddress" />
```

de varios métodos que responden a los cambios en documentos.

```

textoleido.addListener(new TextWatcher() {
    @Override
    public void onTextChanged(CharSequence s, int start, int count, int after) {
        // Nos responde antes de que el texto cambie
    }
    @Override
    public void afterTextChanged(Editable s) {
        // Nos responde después de que el texto cambie
    }
})
}

```

Por último, para manejar los eventos del botón de acción declarado en el atributo android:actionOptions se deberá usar el escuchador OnEditorActionListener junto a su controlador onEditorAction(). Un ejemplo para implementar esto sería así:

```

textoleido.setOnEditorActionListener(new TextView.OnEditorActionListener() {
    @Override
    public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {
        if (actionId == EditorInfo.IME_ACTION_SEARCH) {
            // Código que se ejecutara al pulsar el botón de acción
            return true;
        }
        return false;
    }
})
}

```

Este objeto es una variante del anterior, al que se añaden sugerencias de autocompletado. Para ello, tiene una importante propiedad android:completionThreshold, que define el número de caracteres mínimo que se han de escribir para que la aplicación ofrezca una sugerencia (en este ejemplo, cuatro caracteres).

```

    <AutoCompleteTextView
        android:id="@+id/miTexto"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:completionHint="Ejige una opción"
        android:completionThreshold="4" />
}

```

Los elementos que se sugerirán deben ser definidos en el fichero Java en un array de string, que se le debe crear un adaptador del tipo ArrayAdapter (más adelante se estudiarán los adaptadores en profundidad), cuya misión es adaptar los datos del array al AutoCompleteTextView. Este adaptador se pasará como argumento en el constructor, junto al contexto de la actividad y forma de mostrarse (en este caso, en forma de lista desplegable).

Por último, se pasa el adaptador a la instancia del texto definido en el XML. Como en casos anteriores, deberán importarse las librerías necesarias, que aquí serán android.widget.ArrayAdapter y android.widget.AutoCompleteTextView. El código final Java quedaría así:

```

String[] opciones = {"Opción 1", "Opción 2", "Opción 3", "Opción 4"};
AutoCompleteTextView textoleido = (AutoCompleteTextView) findViewById(R.id.miTexto);
ArrayAdapter<String> adaptador = new ArrayAdapter<String>(this,
    android.R.layout.simple_dropdown_item_1line, opciones);
textoleido.setAdapter(adaptador);
}

```



Figura 5.3  
Autocompletado con tres caracteres.

## 5.7. AutoCompleteTextView

Implementa un EditText en tu proyecto y modifica los parámetros para obtener los siguientes resultados:



## Actividad propuesta 5.4

Implementa un EditText en tu proyecto y modifica los parámetros para obtener los siguientes resultados:

La diferencia entre este objeto y el anterior es que este permite ofrecer sugerencias para cada bloque de texto introducido, marcando mediante un delimitador (token) el elemento a partir del cual volverá a hacerlos la sugerencia. En el ejemplo siguiente se utiliza la coma como elemento delimitador.

```

MultiAutoCompleteTextView textoleido = (MultiAutoCompleteTextView) findViewById(R.id.miTexto);
ArrayAdapter<String> adaptador = new ArrayAdapter<String>(this,
    android.R.layout.simple_dropdown_item_1line, opciones);
textoleido.setTokenizer(new MultiAutoCompleteTextView.CommaTokenizer());
textoleido.setAdapter(adaptador);
}

```

## 5.9. Spinner

El widget Spinner de Android muestra una lista desplegable para seleccionar, y es equivalente a ComboBox de otros lenguajes de programación.

Para realizar un escuchador que responda a los eventos de cambio, se utiliza el selector OnItemSelectedListener, que puede responder a dos eventos, onItemSelected y onNothingSelected. Este devolverá el contenido del array, que se encuentra en la posición tocada del adaptador.

```
<Spinner  
    android:id="@+id/miSpinner"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"/>  
  
En Java este objeto depende de la librería android.widget.Spinner, y para instanciarse, asignarle un array de valores y un adaptador (del tipo simple).  
  
Spinner spinner = (Spinner) findViewById(R.id.miSpinner);  
String[] valores = {"Valor1", "Valor2", "Valor3", "Valor4", "Valor5"};  
spinner.setAdapter(new ArrayAdapter<String>(this,  
    android.R.layout.simple_spinner_item, valores));
```

En Java este objeto depende de la librería android.widget.Spinner, y para instanciarse, asignarle un array de valores y un adaptador (del tipo simple).

```
Spinner spinner = (Spinner) findViewById(R.id.miSpinner);  
String[] valores = {"Valor1", "Valor2", "Valor3", "Valor4", "Valor5"};  
spinner.setAdapter(new ArrayAdapter<String>(this,  
    android.R.layout.simple_spinner_item, valores));
```

Como ya se vio anteriormente, puede ser implementado el escuchador en la clase principal.

## 10. CheckBox

También puede añadirse la lista de valores directamente en el XML dentro del control CheckBox se suele utilizar para marcar o desmarcar opciones en una aplicación. En valores.xml.

```
<CheckBox  
    android:id="@+id/miCheckBox"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Pulsa Opción"  
    android:checked="false" />  
  
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <string-array name="valores">  
        <item>Valor1</item>  
        <item>Valor2</item>  
        <item>Valor3</item>  
        <item>Valor4</item>  
        <item>Valor5</item>  
    </string-array>
```

**Figura 5.4** Panel de Spins-ulsado.

El adaptador en Java se ha de aplicar de la siguiente manera:

```
Spinner spinner = (Spinner) findViewById(R.id.miSpinner);  
ArrayAdapter<CharSequence> adapter = new ArrayAdapter<CharSequence>(this,  
    R.layout.valores, android.R.layout.simple_spinner_item);  
spinner.setAdapter(adapter);
```

Para leer el elemento seleccionado, se escribirá el siguiente código:

```
String value = spinner.getSelectedItem().toString();
```

Y al igual que en el CheckBox, puede asociarle mediante android:onClick un método uno a uno) que lo controle desde Java o bien asociarle un manejador de eventos al grupo en un conjunto.

```
CheckBox checkBox = (CheckBox) findViewById(R.id.miCheckbox);
checkBox.setOnCheckedChangeListener(new CheckBox.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            //Código para pulsado
        } else {
            //Código para no pulsado
        }
    }
});
```

## 5.11. RadioButton

Es un control de selección para elegir una opción de entre un conjunto de control de exclusión mutua, es decir, la selección de uno implica el cambio que previamente estaba seleccionado. Pueden agruparse mediante RadioGroup que, de los que están incluidos en este grupo, solo uno de ellos permanecerá marcado.

```
<RadioGroup
    android:id="@+id/grupo"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_weight="1"
    android:orientation="vertical">
    <RadioButton
        android:id="@+id/radio1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Opción 1" />
    <RadioButton
        android:id="@+id/radio2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Opción 2" />
</RadioGroup>
```

**Figura 5.5**  
Ejemplo de RadioButton

Para manejar los eventos, debe procederse de una manera similar que en anteriores ocasiones.

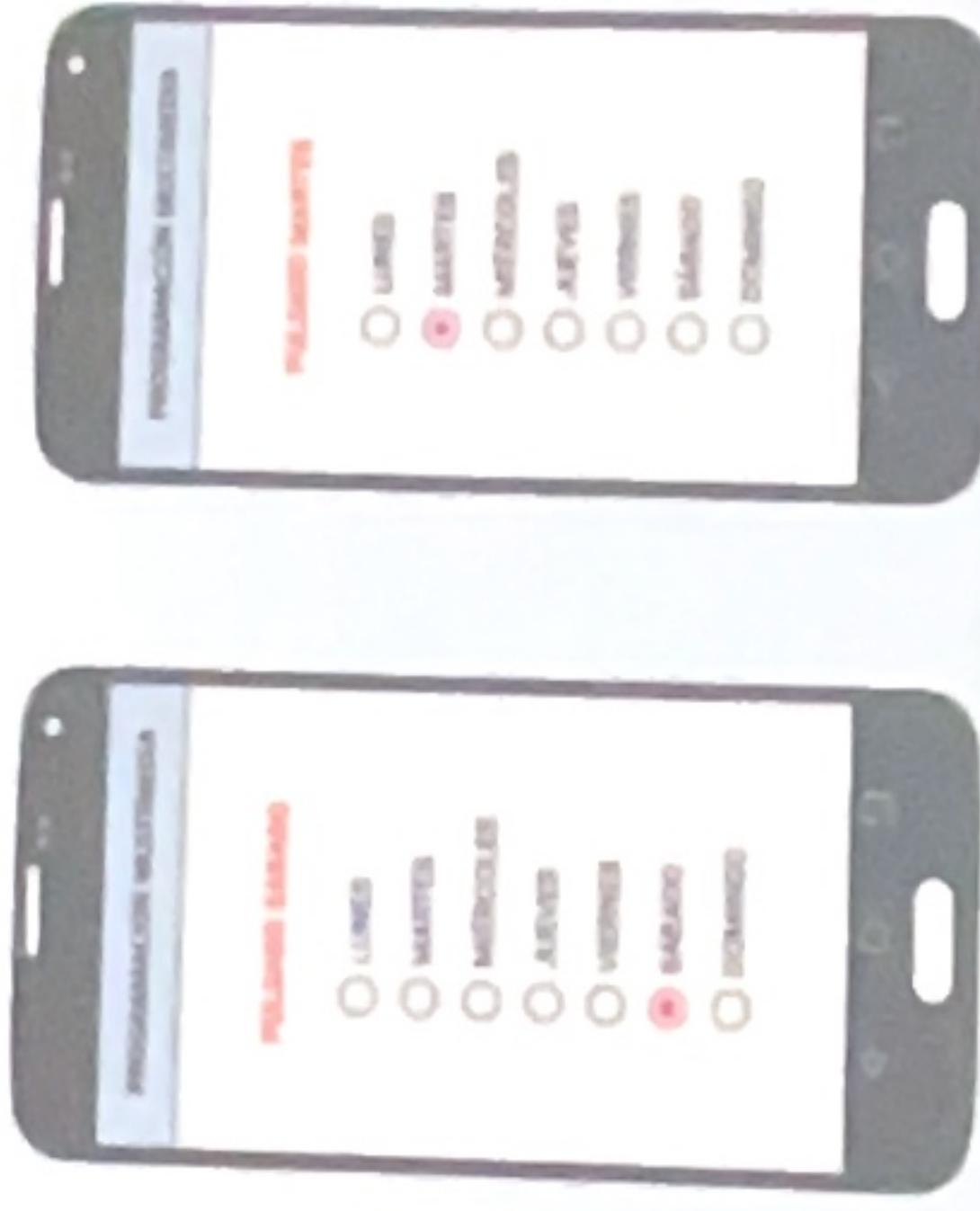
```
RadioGroup miGrupo = (RadioGroup) findViewById(R.id.grupo);
miGrupo.clearCheck();
int idMarcado = miGrupo.getCheckedRadioButtonId();
switch pulsador = (Switch) findViewById(R.id.miSwitch);
pulsador.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) { //código para pulsado
        } else { //código para no pulsado
        }
    }
});
```

Puede desmarcarse (clearCheck) o marcarse (check), y localizar el elemento que ocurre en este ejemplo:

```
RadioGroup miGrupo = (RadioGroup) findViewById(R.id.grupo);
miGrupo.check(R.id.radio1);
int idMarcado = miGrupo.getCheckedRadioButtonId();
```

## Actividad propuesta 5.5

Realiza un proyecto con los RadioButton que aparecen en la imagen y que respondan al evento checked cambiando una etiqueta de texto.



## 5.12. Switch

Este objeto puede utilizarse en el diseño del interfaz y consta de dos estados, encendido (marcado) o apagado (desmarcado), bastante parecido al ToggleButton (ambos subclase de CompoundButton), pero lo diferencia en que este simula un botón con control deslizante.

**Opción 1**

**Opción 2**

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

**Opción 1**

**Opción 2**

**Ejemplo de RadioButton**

Para manejar los eventos, debe procederse de una manera similar que en anteriores ocasiones.

```
switch pulsador = (Switch) findViewById(R.id.miSwitch);
pulsador.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) { //código para pulsado
        } else { //código para no pulsado
        }
    }
});
```

Para manejar los eventos, debe procederse de una manera similar que en anteriores ocasiones.



Los tres elementos básicos que hemos estudiado, CheckBox, RadioButton y ProgressBar, pueden mejorar mucho su aspecto y más importante aún personalizarlos interactuando con los ficheros de estilo en los recursos. Tras esto, y prácticamente cada uno de ellos,



Este objeto dispone de un estilo propio que nos permite ver cada una de las fracciones de avance con style="@style/Widget.AppCompat.SeekBar.Discrete", que nos dejará el siguiente aspecto:

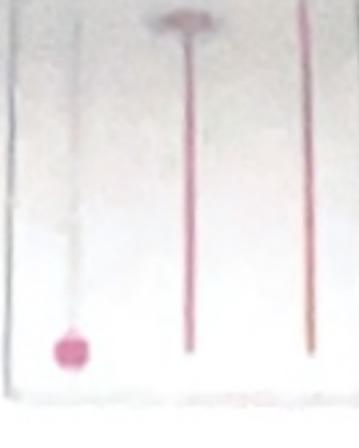
## 5.13. SeekBar

Este control pertenece al tipo de controles de selección en los que el número crece entre un rango de valores predefinidos en la aplicación. Para ello dispone de interesantes atributos:

- android: max. Define el valor máximo del control.
- android: min. Define el valor mínimo del control.
- 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.

`<SeekBar  
    android:id="@+id/miSeekBar"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:min="10"  
    android:progress="50"/>`

Figura 5.7  
Ejemplo de SeekBar.



Este objeto posee un escuchador que nos permite manejar tres eventos: empezamos a mover el control (onStartTrackingTouch), durante el movimiento (onProgressChanged) y al dejar de actuar sobre él (onStopTrackingTouch).

```
SeekBar miControl = (SeekBar) findViewById(R.id.miSeekBar);  
miControl.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {  
    public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {  
        // Código para cambio de valor  
    }  
    public void onStartTrackingTouch(SeekBar seekBar) {  
        // Código para inicio de cambio  
    }  
    public void onStopTrackingTouch(SeekBar seekBar) {  
        // Código para final de cambio  
    }  
});
```

Este control, al igual que todos los demás, tiene multitud de posibilidades especialmente en su aspecto estético.

## Investiga



Figura 5.8  
Ejemplo de SeekBar con fracción de avance.

Además de poder crear tu estilo propio de barra en los recursos, puedes modificar el color tanto solo con los siguientes atributos:

Figura 5.9  
Ejemplo de SeekBar con estilo de color.



## 5.14. RatingBar

Es otro control de selección, pero aquí, como su nombre indica, está diseñado para dar una puntuación. Los atributos de este control son múltiples y sería conveniente consultar la página oficial de desarrolladores de Android. En el siguiente ejemplo, se fijan el número de estrella y el valor de los incrementos. En algunos dispositivos, el número de estrellas debe ajustarse a los márgenes establecidos para este control.



Figura 5.7  
Ejemplo de RatingBar.



Figura 5.10  
Ejemplo de RatingBar.

El manejador de eventos que debe ser utilizado será de la siguiente forma:

```
RatingBar miControl = (RatingBar) findViewById(R.id.miRating);  
miControl.setOnRatingBarChangeListener(new RatingBar.OnRatingBarChangeListener() {  
    @Override  
    public void onRatingChanged(RatingBar ratingBar, float rating, boolean fromUser) {  
        // Código para cambio de valor  
    }  
});
```

## 5.15. ProgressBar

Un interesante control que suele utilizarse con las tareas asíncronas y que sirve para informar al usuario de la evolución de un proceso, rellenando el tiempo de espera entre la acción del

Este control, al igual que todos los demás, tiene multitud de posibilidades especialmente en su aspecto estético.