Controles básicos (II)

En este documento nos vamos a centrar en otros tres componentes básicos imprescindibles en nuestras aplicaciones: Las imágenes (**ImageView**), las etiquetas (**TextView**) y por último los cuadros de texto (**EditText**).

Control ImageView

El control **ImageView** permite mostrar imágenes en la aplicación. La propiedad más interesante es `android:src`, que permite indicar la imagen a mostrar. Nuevamente, lo normal será indicar como origen de la imagen el identificador de un recurso de nuestra carpeta /res/drawable, por ejemplo `android:src="@drawable/unaimagen"`.
Además de esta propiedad, existen algunas otras útiles en algunas ocasiones como las destinadas a establecer el tamaño máximo que puede ocupar la imagen, `android:maxWidth` y `android:maxHeight`, o para indicar cómo debe adaptarse la imagen al tamaño del control, `android:scaleType` (5=CENTER, 6=CENTER_CROP, 7=CENTER_INSIDE, ...). Además, como ya comentamos para el caso de los controles ImageButton, al tratarse de un control de tipo imagen deberíamos establecer siempre la propiedad `android:contentDescription` para ofrecer una breve descripción textual de la imagen, algo que hará nuestra aplicación mucho más accesible.

```
<ImageView android:id="@+id/ImgFoto"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/icon"
    android:contentDescription="@string/imagen_ejemplo" />
```

Si en vez de establecer la imagen a mostrar en el propio layout XML de la actividad quisiéramos establecerla mediante código utilizaríamos el método `setImageResource` (...), pasándole el ID del recurso a utilizar como contenido de la imagen.

```
ImageView img= (ImageView)findViewById(R.id.ImgFoto);
img.setImageResource(R.drawable.icon);
```

En cuanto a posibles eventos, al igual que comentamos para los controles de tipo botón en el apartado anterior, para los componentes **ImageView** también podríamos implementar su evento **onClick**, de forma idéntica a la que ya vimos, aunque en estos casos suele ser menos frecuente la necesidad de capturar este evento

Control TextView

El control **TextView** es otro de los clásicos en la programación de GUIs, las etiquetas de texto, y se utiliza para mostrar un determinado texto al usuario. Al igual que en el caso de los botones, el texto del control se establece mediante la propiedad `android:text`.

A parte de esta propiedad, la naturaleza del control hace que las más interesantes sean las que establecen el formato del texto mostrado, que al igual que en el caso de los botones son las siguientes: `android:background` (color de fondo), `android:textColor` (color del texto), `android:textSize` (tamaño de la fuente) y `android:typeface` (estilo del texto: negrita, cursiva, ...).

```
<TextView android:id="@+id/LblEtiqueta"
    android:layout_width="matchfill_parent"
    android:layout_height="wrap_content"
    android:text="@string/escribe_algoEscribe algo:"
    android:background="#AA44FF"
    android:typeface="monospace" />
```

De igual forma, también podemos manipular estas propiedades desde nuestro código.

Como ejemplo, en el siguiente fragmento recuperamos el texto de una etiqueta con `getText()`, y posteriormente le concatenamos unos números, actualizamos su contenido mediante `setText()` y le cambiamos su color de fondo con `setBackgroundColor()`.

```
final TextView lblEtiqueta = (TextView)findViewById(R.id.LblEtiqueta);
String texto = lblEtiqueta.getText().toString();
texto += "123";
lblEtiqueta.setText(texto);
lblEtiqueta.setBackgroundColor(Color.BLUE);
```

Control EditText

El control **EditText** es el componente de edición de texto que proporciona la plataforma Android. Permite la introducción y edición de texto por parte del usuario, por lo que en tiempo de diseño la propiedad más interesante a establecer, además de su posición/tamaño y formato, es el texto a mostrar, atributo `android:text`.

Por supuesto si no queremos que el cuadro de texto aparezca inicializado con ningún texto, no es necesario incluir esta propiedad en el layout XML. Lo que sí deberemos establecer será la propiedad `android:inputType`.

Esta propiedad indica el tipo de contenido que se va a introducir en el cuadro de texto, como por ejemplo una dirección de correo electrónico (**textEmailAddress**), un número genérico (**number**), un número de teléfono (**phone**), una dirección web (**textUri**), o un texto genérico (**text**).

El valor que establezcamos para esta propiedad tendrá además efecto en el tipo de teclado que mostrará Android para editar dicho campo. Así, por ejemplo, si hemos indicado "**text**" mostrará el teclado completo alfanumérico, si hemos indicado "**phone**" mostrará el teclado numérico del teléfono, etc.


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

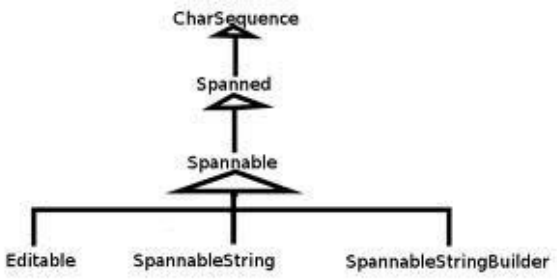
De igual forma, desde nuestro código podremos recuperar y establecer este texto mediante los métodos **`getText`**`()` y **`setText`**`(nuevoTexto)` respectivamente:

```
final EditText txtTexto = (EditText)findViewById(R.id.TxtTexto);
String texto = txtTexto.getText().toString();
txtTexto.setText("Hola mundo!");
```

Un detalle que puede haber pasado desapercibido.
Nota: Hemos tenido que hacer un `toString()` sobre el resultado de `getText()`. La explicación para esto es que el método `getText()` no devuelve directamente una cadena de caracteres (**String**) sino un objeto de tipo **Editable**, que a su vez implementa la interfaz **Spannable**.
Y esto nos lleva a la característica más interesante del control **EditText**, y es que no sólo nos permite editar *texto plano* sino también *texto enriquecido* o con formato.

Interfaz Spanned

Un objeto de tipo **Spanned** es algo así como una cadena de caracteres (de hecho deriva de la interfaz **CharSequence**) en la que podemos insertar otros objetos a modo de marcas o etiquetas (*spans*) asociados a rangos de caracteres. De esta interfaz deriva la interfaz **Spannable**, que permite la modificación de estas marcas, y a su vez de ésta última deriva la interfaz **Editable**, que permite además la modificación del texto.
Aunque en el apartado en el que nos encontramos nos interesamos principalmente por las marcas de formato de texto, en principio podríamos insertar cualquier tipo de objeto.



Existen muchos tipos de *spans* predefinidos en la plataforma que podemos utilizar para dar formato al texto, entre ellos:

- **TypefaceSpan**. Modifica el tipo de fuente.
- **StyleSpan**. Modifica el estilo del texto (negrita, cursiva, ...).
- **ForegroudColorSpan**. Modifica el color del texto.
- **AbsoluteSizeSpan**. Modifica el tamaño de fuente.

De esta forma, para crear un nuevo objeto Editable e insertar una marca de formato podríamos hacer lo siguiente:

```
//Creamos un nuevo objeto de tipo Editable
Editable str = Editable.Factory.getInstance().newEditable("Esto es un simulacro.");

//Marcamos como fuente negrita la palabra "simulacro" (caracteres del 11-19)
StyleSpan style;
style = new StyleSpan(android.graphics.Typeface.BOLD);
str.setSpan(style, 11, 19, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
```

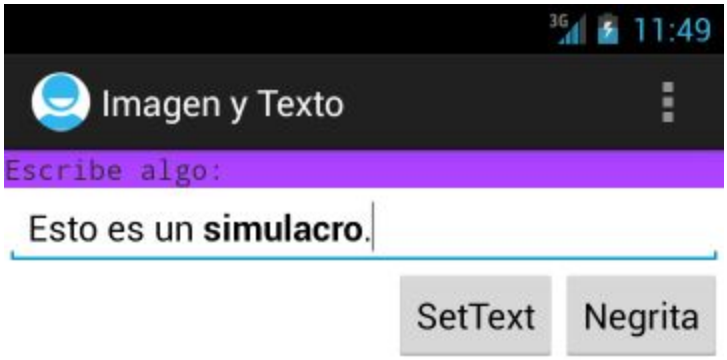
En este ejemplo estamos insertando un span de tipo **StyleSpan** para marcar un fragmento de texto con estilo *negrita*. Para insertarlo utilizamos el método `setSpan()`, que recibe como parámetro el objeto **Span** a insertar, la posición inicial y final del texto a marcar, y un *flag* que indica la forma en la que el span se podrá extender al insertarse nuevo texto.
El resultado sería algo así:
"Esto es un **simulacro.**"

Texto con formato en controles TextView y EditText

Hemos visto cómo crear un objeto Editable y añadir marcas de formato al texto que contiene, pero todo esto no tendría ningún sentido si no pudiéramos visualizarlo. Como podemos imaginar, los controles **TextView** y **EditText** nos van a permitir hacer esto. Veamos qué ocurre si asignamos al nuestro control **EditText** el objeto **Editable** que hemos creado antes:

```
txtTexto.setText(str);
```

Tras ejecutar este código, para lo que hemos insertado un botón "SetText" en la aplicación de ejemplo, veremos cómo efectivamente en el cuadro de texto aparece el mensaje con el formato esperado:



Como podemos ver en la captura anterior, en la aplicación de ejemplo también se ha incluido un botón adicional "Negrita" que se encargará de convertir a estilo negrita un fragmento de texto previamente seleccionado en el cuadro de texto. La intención es aprender los métodos disponibles para determinar el comienzo y el fin de una selección en un control de este tipo.

Para ello utilizaremos los métodos `getSelectionStart()` y `getSelectionEnd()`, que retornan el índice del primer y último carácter seleccionado en el texto.

Sabiendo esto, ya sólo nos queda utilizar el método `setSpan()` para convertir la selección a negrita.

```
Spannable texto = txtTexto.getText();

int inicio = txtTexto.getSelectionStart();
int final = txtTexto.getSelectionEnd();

texto.setSpan(
    new StyleSpan(android.graphics.Typeface.BOLD),
    inicio,
    final,
    Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
```

La función `getText()` retorna un objeto de tipo Editable, pero al hacer un `toString()` estamos perdiendo todo el formato del texto.

La solución a esto último pasa obviamente por recuperar directamente el objeto **Editable** y serializarlo de algún modo, mejor aún si es en un formato estándar. Pues bien, en Android este problema se soluciona a través de la clase **Html**, que dispone de métodos para convertir cualquier objeto **Spanned** en su representación HTML equivalente.

Recuperemos el texto de la ventana anterior y utilizamos el método `Html.toHtml(Spannable)` para convertirlo a formato HTML:

```
//Obtiene el texto del control con etiquetas de formato HTML
String aux2 = Html.toHtml(txtTexto.getText());
```

Haciendo esto, obtendremos una cadena de texto como la siguiente, que ya podríamos por ejemplo almacenar en una base de datos o publicar en cualquier web sin perder el formato de texto establecido:
Esto es un simulacro.

La operación contraria también es posible, es decir, cargar un cuadro de texto de Android (**EditText**) o una etiqueta (**TextView**) a partir de un fragmento de texto en formato HTML.
Para ello podemos utilizar el método `Html.fromHtml(String)` de la siguiente forma:

```
//Asigna texto con formato HTML
txtTexto.setText(
     Html.fromHtml("<p>Esto es un <b>simulacro</b>.</p>"),
     BufferType.SPANNABLE);
```

Desgraciadamente, sólo funciona de forma completa con las opciones de formato más básicas, como negritas, cursivas, subrayado o colores de texto, quedando no soportadas otras sorprendentemente básicas como el tamaño del texto, que aunque sí es correctamente traducido por el método `toHtml()`, es descartado por el método contrario `fromHtml()`. Sí se soporta la inclusión de imágenes.