Interfaz de usuario en Android: Controles básicos (III)

Tras hablar de varios de los controles indispensables en cualquier aplicación Android, como son los botones y los cuadros de texto, en este artículo vamos a ver cómo utilizar otros dos tipos de controles básicos en muchas aplicaciones, los *checkboxes* y los *radio buttons*.

Control CheckBox [API]

Un control *checkbox* se suele utilizar para marcar o desmarcar opciones en una aplicación, y en Android está representado por la clase del mismo nombre, `CheckBox`. La forma de definirlo en nuestra interfaz y los métodos disponibles para manipularlos desde nuestro código son análogos a los ya comentados para el control `ToggleButton`.

De esta forma, para definir un control de este tipo en nuestro *layout* podemos utilizar el código siguiente, que define un *checkbox* con el texto "Márcame":

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

En cuanto a la personalización del control podemos decir que éste extiende [indirectamente] del control `TextView`, por lo que todas las opciones de formato ya comentadas en artículos anteriores son válidas también para este control. Además, podremos utilizar la propiedad **`android:checked`** para inicializar el estado del control a marcado (**`true`**) o desmarcado (**`false`**). Si no establecemos esta propiedad el control aparecerá por defecto en estado desmarcado.

En el código de la aplicación podremos hacer uso de los métodos `isChecked()` para conocer el estado del control, y `setChecked(estado)` para establecer un estado concreto para el control.

```
1     if (checkBox.isChecked()) {
2         checkBox.setChecked(false);
3     }
```

En cuanto a los posibles eventos que puede lanzar este control, `onClick` vuelve a ser el más interesante ya que nos indicará cuándo se ha pulsado sobre el checkbox. Dentro de este evento consultaremos normalmente el estado del control con `isChecked()` como acabamos de ver.

```
cbMarcame = (CheckBox)findViewById(R.id.ChkMarcame);
cbMarcame.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        boolean isChecked = ((CheckBox)view).isChecked();

        if (isChecked) {
            cbMarcame.setText("Checkbox marcado!");
        }
        else {
            cbMarcame.setText("Checkbox desmarcado!");
        }
    }
});
```

Otro evento que podríamos utilizar es `onCheckedChanged`, que nos informa de que ha cambiado el estado del control. Para implementar las acciones de este evento podríamos utilizar la siguiente lógica, donde tras capturar el evento, y dependiendo del nuevo estado del control (variable `isChecked` recibida como parámetro), haremos una acción u otra:

```
cbMarcame = (CheckBox)findViewById(R.id.ChkMarcame);

cbMarcame.setOnCheckedChangeListener(new
CheckBox.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean
isChecked) {
        if (isChecked) {
            cbMarcame.setText("Checkbox marcado!");
        }
        else {
            cbMarcame.setText("Checkbox desmarcado!");
        }
```

```
    }
});
```

Control RadioButton [API]

Al igual que los controles *checkbox*, un *radio button* puede estar marcado o desmarcado, pero en este caso suelen utilizarse dentro de un grupo de opciones donde una, y sólo una, de ellas debe estar marcada obligatoriamente, es decir, que si se marca una de las opciones se desmarcará automáticamente la que estuviera activa anteriormente. En Android, un grupo de botones *radio button* se define mediante un elemento **`RadioGroup`**, que a su vez contendrá todos los elementos **`RadioButton`** necesarios. Veamos un ejemplo de cómo definir un grupo de dos controles *radiobutton* en nuestra interfaz:

```
<RadioGroup
android:id="@+id/GrbGrupo1"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent"
>

<RadioButton
android:id="@+id/RbOpcion1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/opcion_1" />

<RadioButton
android:id="@+id/RbOpcion2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/opcion_2" />

</RadioGroup>
```

En primer lugar vemos cómo podemos definir el grupo de controles indicando su **orientación (vertical u horizontal)** al igual que ocurría por ejemplo con un `LinearLayout`. Tras esto, se añaden todos los objetos `RadioButton` necesarios indicando su ID mediante la propiedad `android:id` y su texto mediante `android:text`.

Una vez definida la interfaz podremos manipular el control desde nuestro código java haciendo uso de los diferentes métodos del control `RadioGroup`, los más importantes: **`check(id)`** para marcar una opción determinada mediante su ID, **`clearCheck()`** para desmarcar todas las opciones, y **`getCheckedRadioButtonId()`** que como su nombre indica devolverá el ID de la opción marcada (o el valor -1 si no hay ninguna marcada). Veamos un ejemplo:

```
RadioGroup rg = (RadioGroup)findViewById(R.id.GrbGrupo1);
rg.clearCheck();
rg.check(R.id.RbOpcion1);
int idSeleccionado = rg.getCheckedRadioButtonId();
```

En cuanto a los eventos lanzados, recurriremos nuevamente al evento `onClick` para saber cuándo se pulsa cada uno de los botones del grupo. Normalmente utilizaremos un mismo *listener* para todos los radiobutton del grupo, por lo que lo definiremos de forma independiente y después lo asignaremos a todos los botones.

```
private TextView lblMensaje;
private RadioButton rbOpcion1;
private RadioButton rbOpcion2;
……..
……..

lblMensaje = (TextView)findViewById(R.id.LblSeleccion);
rbOpcion1 = (RadioButton)findViewById(R.id.RbOpcion1);
rbOpcion2 = (RadioButton)findViewById(R.id.RbOpcion2);

View.OnClickListener list = new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String opcion = "";
        switch(view.getId()) {
            case R.id.RbOpcion1:
                opcion = "opción 1";
                break;
            case R.id.RbOpcion2:
                opcion = "opción 2";
                break;
        }

        lblMensaje.setText("ID opción seleccionada: " + opcion);
    }
};
```

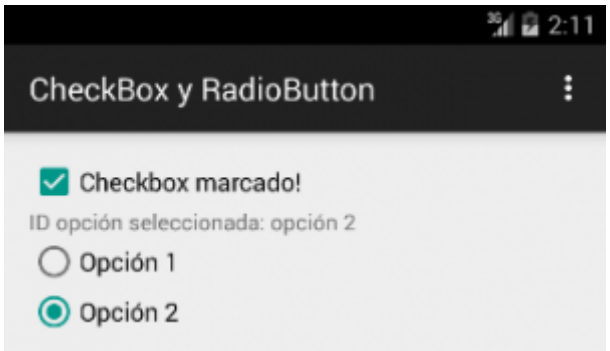
Al igual que en el caso de los checkboxes, también podremos utilizar el evento **onCheckedChange**, que nos informará de los cambios en el elemento seleccionado dentro de un grupo. La diferencia aquí es que este evento está asociado al `RadioGroup`, y no a los diferentes `RadioButton` del grupo. Veamos cómo tratar este evento haciendo por ejemplo que una etiqueta de texto cambie de valor al seleccionar cada opción:

```
rgOpciones = (RadioGroup)findViewById(R.id.GrbGrupo1);
rgOpciones.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener() {
    public void onCheckedChanged(RadioGroup group, int checkedId) {

        String opcion = "";
        switch(checkedId) {
            case R.id.RbOpcion1:
                opcion = "opción 1";
                break;
            case R.id.RbOpcion2:
                opcion = "opción 2";
                break;
        }

        lblMensaje.setText("ID opción seleccionada: " + opcion);
    }
});
```

Veamos finalmente una imagen del aspecto de estos dos nuevos tipos de controles básicos que hemos comentado en este artículo:



Puedes consultar y/o descargar el código completo de los ejemplos desarrollados en este artículo accediendo a la página del curso en GitHub.

Enlaces de interés:

- Switches en Material Design
- Switches en Guía de diseño Android
- CheckBoxes y RadioButton en Guía de desarrollo Android.