MEDAC
Instituto Oficial de Formación Profesional

PROGRAMACIÓN MULTIMEDIA Y DISPOSITIVOS MÓVILES
TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Programación en Android III: botones y listas

05

ÍNDICE

/ 1. Introducción y contextualización práctica 3

/ 2. Etiquetas y cajas de texto 4

/ 3. Botones 4

/ 4. Caso práctico 1: "¿Cuál es la forma correcta de implementar funcionalidades?" 5

/ 5. Imágenes y listas desplegables 6

/ 6. Checkbox y Radiobutton 6

/ 7. Listas optimizadas I: creación de los componentes 7

/ 8. Caso práctico 2: "Diseño óptimo de interfaces" 8

/ 9. Listas optimizadas II: crear la lista 8

/ 10. Resumen y resolución del caso práctico de la unidad 9

/ 11. Bibliografía 10

OBJETIVOS

Trabajar con interfaz gráfica.

Conocer diferentes elementos gráficos básicos.

Conocer y aplicar el concepto de evento.

Crear aplicaciones básicas.

/ 1. Introducción y contextualización práctica

En esta unidad, vamos a ver una serie de elementos gráficos básicos para el desarrollo de cualquier aplicación Android.

Estudiaremos cómo podemos brindarles **funcionamiento** a dichos elementos, obtener el texto que se escriba, que un botón funcione cuando lo pulsemos, etc.

También veremos el concepto de evento, algo básico si queremos hacer que, por ejemplo, un botón realice cierta funcionalidad cuando lo pulsemos.

Por último, aprenderemos cómo podemos implementar una lista de elementos de forma correcta, lo que se conoce como una **«lista optimizada»**.

Todos estos conceptos los repasaremos en diversos temas relacionados con este, así que no te preocupes, porque tenemos mucho tiempo por delante.

Escucha el siguiente audio en el que planteamos el caso práctico que iremos resolviendo a lo largo de esta unidad:

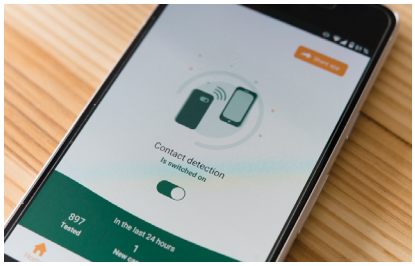

Fig. 1. Aplicación móvil.

Audio Intro. "Listos para comenzar"
https://bit.ly/2Oo1gfY

/ 2. Etiquetas y cajas de texto

La funcionalidad más básica que podremos implementar en una pantalla de nuestra aplicación móvil es la de mostrar texto e introducir texto.

Para esto tenemos los siguientes elementos gráficos:

- **TextView:** Este elemento se encarga de mostrar texto en la pantalla mediante una etiqueta.
- **EditText:** Este elemento representa una caja de texto en la que podemos escribir.

Los elementos *TextView* (o etiquetas de texto) se utilizan principalmente para mostrar algo en la pantalla. Las funcionalidades que nos van a proporcionar son la de obtener y modificar el texto que muestran.

Los elementos *EditText* (o cajas de texto) se utilizan principalmente para poder introducir datos en nuestra aplicación, es decir, escribiremos el texto que deseemos y podremos obtenerlo o modificarlo. Existen varios tipos de *EditText* ya formateados que nos permitirán introducir únicamente un texto, un email, un teléfono, números, contraseñas, etc.

Para poder utilizar cualquier elemento de estos dos **deberemos crearlos en la interfaz gráfica** y proporcionarles un id, lo cual es totalmente imprescindible para poder ligarlos a código y trabajar con ellos. Una vez hecho esto, debemos crear un objeto de la clase *TextView* o *EditText*, según necesitemos, y ligarlo al elemento gráfico mediante el método *findViewById*.

Tras ligarlo, podremos trabajar con él con normalidad. Os recomendamos llamar al id y al objeto de la misma forma, para así tener el código más claro.

```
public class MainActivity extends AppCompatActivity {
    // Creo el objeto de TextView
    private TextView tsaludo;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Ligo los elementos gráficos
        tsaludo = findViewById(R.id.tSaludo);

        // Trabajamos con el elemento normalmente
        tsaludo.setText("Hola");
        System.out.println(tsaludo.getText());
    }
}
```

Fig. 2. Uso de una etiqueta.

Audio 1. "Atributos genéricos a los elementos"
https://bit.ly/32jFtyj

/ 3. Botones

Los botones son un elemento gráfico imprescindible en nuestras aplicaciones móviles, ya que gracias a ellos podremos proporcionar interacción entre el usuario y la aplicación.

La clase encargada de implementar los botones es la clase *Button*, de la cual hay varias variantes:

- **ImageButton:** Es un botón con una imagen en lugar de texto.
- **Switch:** Es un tipo de botón que podrá estar activado o desactivado.
- **FloatingActionButton:** Es un botón flotante que podrá estar en las esquinas de la pantalla.

Los botones mostrarán normalmente un texto, el cual deberá ir en el fichero de *strings.xml* y, entre otras cosas, podremos cambiar y obtener el texto mostrado, cambiar su estilo a uno definido en el fichero *styles.xml*, etc.

Para poder hacer que un botón realice una acción (evento) cuando hacemos clic sobre él, debemos seguir los siguientes pasos:

- **Dar** un **id** al botón y **crear** un **objeto** de tipo *Button* ligándolo con el método *findViewById* a su identificador.
- **Hacer** que **la clase** de la actividad implemente la interfaz *View.OnClickListener*.
- **Implementar el método** *onClick* y, usando un switch a la variable del tipo View con el método *getId()*, podremos implementar el código de cada uno de los botones que tengamos.
- Por último, **asignar** *Listener* al botón con el método *setOnClickListener*.

Podremos hacer esto con todos los botones que implementemos.

```
// Creo el objeto de TextView
private Button bEjemplo1, bEjemplo2;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // Ligo los elementos gráficos
    bEjemplo1 = findViewById(R.id.bEjemplo1);
    bEjemplo2 = findViewById(R.id.bEjemplo2);

    // Trabajamos con el elemento normalmente
    bEjemplo1.setOnClickListener(this);
    bEjemplo2.setOnClickListener(this);
}

@Override
public void onClick(View v) {
    switch(v.getId())
    {
        case R.id.bEjemplo1:
            System.out.println("Botón 1");
            break;
    }
}
```

Fig. 3. Ejemplo uso de dos botones.

Vídeo 1. "Ejemplo completo de botones"
https://bit.ly/3j52Ama

/ 4. Caso práctico 1: "¿Cuál es la forma correcta de implementar funcionalidades?"

Planteamiento: Pilar y José ya saben cómo pueden hacer que un botón concreto realice una acción determinada. «Esto es más fácil de lo que pensaba», comenta Pilar. No obstante, como todo en el mundo de la programación, esta no es la única forma de hacer las cosas.

José, que está intentando descubrir una segunda forma, le comenta a Pilar que no sabe si encontrará una forma más correcta de proporcionar al botón la funcionalidad requerida.

Nudo: ¿Crees que hay una forma más óptima que otra de hacer que un elemento gráfico realice su funcionalidad?

Desenlace: Sabemos de sobra que, cuando estamos programando casi cualquier elemento, existirán distintas formas de realizar la misma tarea. Algunas veces será más óptimo, otras menos y otras no tendrá variación, por lo que no importaría la forma en la que se implemente.

Fig. 4. Desarrollando aplicaciones de forma eficiente.

En el caso que tenemos entre manos, sí que hay una forma «más correcta» de dar una funcionalidad a un elemento gráfico, sea un botón, una lista desplegable, etc.

Esta forma consiste en hacer que la actividad en la que nos encontramos implemente el *Listener* de la funcionalidad que queremos proporcionar, implementando, además, dicha funcionalidad en la propia clase.

A efectos de funcionalidad, esta forma de implementar es igual que la opción de implementar los eventos en cada elemento gráfico. No obstante, nos ofrece la ventaja de tener el código ordenado, de forma que podremos identificar la funcionalidad de cada elemento mucho más rápido, además de tener que escribir mucho menos código.

/ 5. Imágenes y listas desplegables

Para mostrar imágenes en una pantalla de nuestra interfaz gráfica podremos **utilizar el elemento** *ImageView*.

Con *ImageView* podremos **mostrar** de una forma muy sencilla **una imagen** que esté dentro de nuestro proyecto. Al agregar este elemento nos aparecerá una ventana con la que podremos elegir qué imagen queremos mostrar.

Debemos tener cuidado con el tamaño de la imagen (que no sea excesivo), ya que se mostrará con su tamaño original.

Las imágenes que utilizaremos se guardarán en la carpeta *res/drawable* y, para cambiarlas, utilizaremos la propiedad *app:srcCompact="@drawable/unaimagen"*.

Otro elemento que vamos a usar mucho son las **listas desplegables**. En la asignatura de programación vimos que teníamos un elemento gráfico llamado *JComboBox* que nos permitía mostrar una lista de elementos para poder elegir de entre uno de ellos. El equivalente en Android será la clase Spinner, que nos va a permitir realizar exactamente la misma funcionalidad que los *JComboBox* en Java.

Los elementos que queramos mostrar en un *Spinner* deberemos colocarlos dentro del fichero *strings.xml* como un *array* y, mediante la propiedad entries, podremos asignarle su valor como *@array/valores*.

```
    <string name="app_name">My Application</string>

    <array name="valores">
        <item>Valor 1</item>
        <item>Valor 2</item>
        <item>Valor 3</item>
    </array>
</resources>
```

Fig. 5. Ejemplo de array en el fichero strings.xml

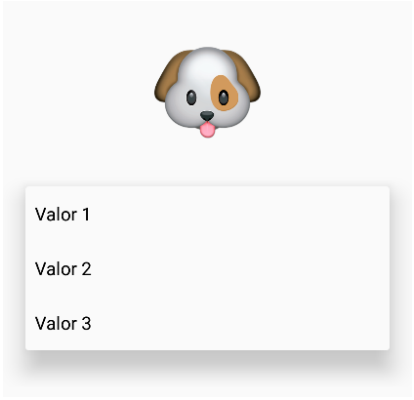

Fig. 6. Ejemplo de imagen y lista desplegable.

/ 6. Checkbox y Radiobutton

Dos de los elementos que no podían faltar en cualquier interfaz gráfica son los **CheckBox** y los **RadioButton**.

Estos elementos funcionan exactamente igual que cuando los estudiamos en la asignatura de programación. Los **CheckBox podrán ser activados o no**, y podremos tener todos los que necesitemos, mientras que los **RadioButton se van a utilizar en grupos**, para que podamos activar un único elemento entre ellos. Los grupos son los *RadioGroup*, e igualmente, podremos tener tantos como necesitamos, pudiendo activar un único *RadioButton* en cada uno. Los grupos los podremos poner tanto en posición vertical como en horizontal.

Podremos saber si los *CheckBox* **están activados o no mediante el método isChecked()**, que nos devolverá 'verdadero' en caso de que esté activado o 'falso' en el caso contrario. También podremos cambiar su activación mediante el método *setChecked()*, al que le pasaremos 'verdadero' o 'falso', según queramos activar o desactivar el elemento.

Sin embargo, para saber qué **RadioButton está activo** dentro de un grupo, tendremos que **implementar la interfaz RadioGroup.OnCheckedChangeListener** con la cual, mediante su método *onCheckedChanged*, podremos realizar una opción específica cuando se seleccione un *RadioButton* dentro de un grupo.

Deberemos utilizar el método *setOnCheckedChangeListener* del grupo para asignar la funcionalidad al mismo.

Al método *onCheckedChanged* le llegarán dos parámetros: el primero será el id del grupo que hayamos cambiado y el segundo será el id del *RadioButton* que ha cambiado.

```
@Override
public void onCheckedChanged(RadioGroup group, int checkedId) {
    switch(group.getId())
    {
        case R.id.grupo1:
            switch(checkedId)
            {
                case R.id.rb1:
                    System.out.println("Cambiado RadioButton1");
                    break;
                case R.id.rb2:
                    System.out.println("Cambiado RadioButton2");
                    break;
            }
            break;
    }
}
```

Fig. 7. Método onCheckedChanged.

/ 7. Listas optimizadas I: creación de los componentes

Las **listas** son elementos que nos van a permitir **mostrar varios elementos en pantalla**. Concretamente, es el componente *RecyclerView* el que nos va a permitir mostrar en pantalla colecciones grandes de datos, teniendo en consideración el trato eficiente de la memoria del dispositivo. El *RecyclerView* no va a hacer prácticamente nada por sí mismo, sino que se va a sustentar sobre otros componentes complementarios para determinar cómo acceder a los datos y cómo mostrarlos. Los más importantes serán los siguientes:

- **RecyclerView.Adapter:** Este representa un adaptador que gestionará cómo se mostrarán los elementos en pantalla.
- **RecyclerView.ViewHolder:** Esta clase se encargará de la gestión de la memoria del dispositivo a la hora de mostrar la lista optimizada. Nosotros únicamente le tendremos que indicar qué elementos habrá.
- **ItemDecoration:** Con estos elementos podremos añadir efectos a nuestras listas como, por ejemplo, una separación más grande entre los elementos.

Lo primero que tendremos que hacer es crear un elemento en la carpeta layout, que será la interfaz de los elementos de nuestra lista optimizada. Esto será una pantalla normal y corriente.

Creamos una clase que representará lo que queramos mostrar en nuestra lista. Esta clase tendrá los métodos básicos. El siguiente paso será crear un adaptador mediante una clase Java a la que llamaremos AdaptadorEjemplo. En el adaptador tendremos que crear los siguientes elementos:

Elemento	Funcionalidad
Crear clase estática interna.	Hereda de RecyclerView.ViewHolder que representará un elemento de la lista.
Método onCreateViewHolder	Obtendrá la interfaz gráfica de los elementos de la lista.
Método onBindViewHolder	Mostrará los datos de los elementos de la lista.
Método getItemCount	Devolverá los elementos que tendrá la lista.

Tabla 1. Métodos de un adaptador básico.

ElementoLista.java

Código 1. "Clase de un elemento de la lista

AdaptadorEjemplo.java

Código 2. "Adaptador de la lista completo"

/ 8. Caso práctico 2: "Diseño óptimo de interfaces"

Planteamiento: Una vez que Pilar y José han realizado un par de ejemplos de interfaces gráficas, comprueban que no es tan complicado como pensaban en un principio y que, debido a los conocimientos que adquirieron en asignaturas anteriores de programación, el proceso de adaptación va bastante rápido. Pilar le comenta a José que las interfaces las tiene controladas, pero que siempre se le queda pequeña la pantalla a la hora de intentar poner todos los elementos gráficos que ella considera convenientes.

Nudo: ¿Qué piensas al respecto? ¿Crees que Pilar hace bien poniendo demasiados elementos gráficos en las interfaces gráficas de las pantallas? ¿Cómo podría optimizarlo?

Desenlace: Es muy normal que estemos acostumbrados a realizar interfaces gráficas en aplicaciones de escritorio donde «no tenemos límite» a la hora de poner elementos. Se trata de una cuestión que puede generarnos algún dolor de cabeza cuando empecemos a diseñar interfaces en dispositivos móviles.

Debemos tener en cuenta que los dispositivos móviles están muy acotados en lo que respecta a sus características y una de ellas es el tamaño de la pantalla, por lo que el número de elementos que podremos colocar se verá muy limitado.

Por este motivo, cuando vayamos a diseñar una pantalla, siempre tendremos que analizar exhaustivamente los elementos que necesitaremos, ya que si alguno de ellos no es verdaderamente imprescindible, podemos eliminarlo sin perder funcionalidad o, en su caso, lo podremos sustituir por otro más compacto.

Fig. 8. Diseño óptimo.

/ 9. Listas optimizadas II: crear la lista

Una vez que hemos:

- **Creado la interfaz gráfica** del elemento de la lista.
- **Creado una clase** para representar el elemento de la lista.
- **Creado un adaptador** completo con el holder interno para mostrar los elementos de la lista.

Podremos crear la lista optimizada y agregar todos los elementos que queramos.

Para esto, debemos seguir los siguientes pasos en el proyecto donde queramos mostrar nuestra lista optimizada:

1. **Crear un objeto** de tipo *RecyclerView* e integrarlo con su equivalente elemento gráfico.
2. A continuación, **crear** tantos **elementos** como queramos en nuestra lista.

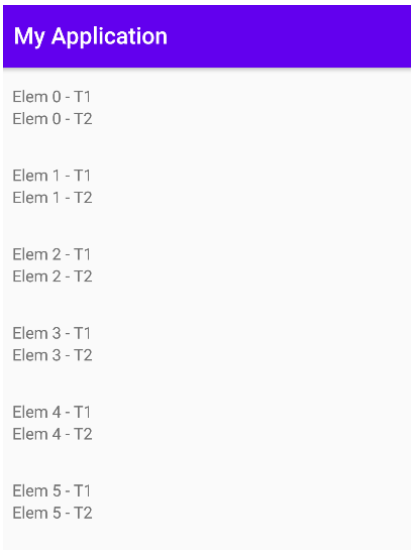

Fig. 9. Ejemplo de lista optimizada.

3. No se nos puede olvidar **crear un adaptador** de elementos para la lista; en nuestro caso, podremos crear un LinearLayoutManager para que los elementos se muestren en forma de lista normal y corriente.

4. **Creamos el adaptador** al que le pasaremos un array con todos los elementos que queramos mostrar.

Como podemos observar, cuando se ejecute nuestra aplicación con la lista, se irán mostrando los elementos que únicamente quedan en pantalla, y tendremos que hacer scroll para poder ver los demás. Esto se debe a que estamos usando una lista optimizada en memoria. Únicamente estarán los elementos que estén en pantalla y cuando vayamos mostrando elementos nuevos esa memoria se irá reutilizando; de esta manera, no se gastará memoria de forma indebida.

Si quisiéramos mostrar los elementos en forma de matriz de tres elementos, podremos usar un GridLayoutManager.

Vídeo 2. "Ejemplo de una lista optimizada paso a paso"
https://bit.ly/2Ots8Lu

/ 10. Resumen y resolución del caso práctico de la unidad

A lo largo de esta unidad hemos visto algunos de los **elementos gráficos** más básicos que cualquier aplicación Android debe tener, como son las etiquetas, las cajas de texto, los botones, los *checkbox* y *radiobutton* y las listas de elementos.

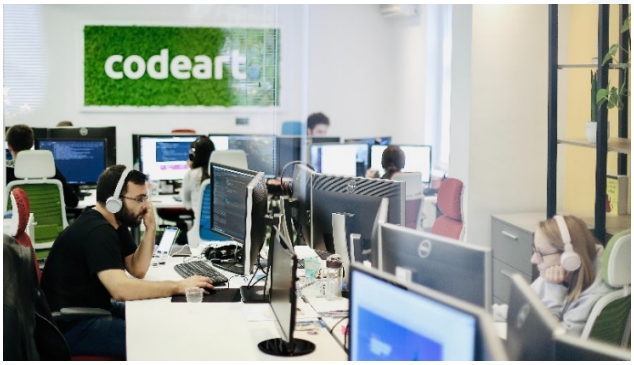

Fig. 10. Equipo de programadores.

Hemos estudiado también el **concepto de evento** y cómo permiten que los elementos gráficos realicen una funcionalidad (por ejemplo, que un botón haga algo cuando lo pulsamos).

La segunda parte del tema ha estado relacionada con la forma en la que podemos **mostrar imágenes** en una pantalla.

Por último, hemos visto cómo podemos crear una l**ista de elementos** de forma correcta, mediante las conocidas como listas optimizadas, sin que la memoria del dispositivo móvil se vea afectada en exceso.

Resolución del caso práctico de la unidad

Uno de los mayores cambios a la hora de pasar de programar aplicaciones de escritorio a aplicaciones móviles es la forma de **crear las interfaces gráficas.**

Debido a las restricciones de potencia de microprocesador, tamaño de pantalla y uso de batería, principalmente, debemos tener especial cuidado con su diseño, ya que un mal diseño puede hacer, por ejemplo, que la batería se vea afectada o que el tamaño de la pantalla sea insuficiente para la cantidad de elementos que queremos mostrar.

Android Studio nos proporciona, en el asistente de creación de un proyecto, una serie de plantillas a la hora de crear las interfaces, las cuales, si sabemos utilizar, nos facilitarán mucho la tarea.

/ 11. Bibliografía

Controles de selección: RecyclerView. (24 de febrero de 2015). [Entrada de Blog]. Recuperado de: https://www.sgoliver.net/blog/controles-de-seleccion-v-recyclerview/

Interfaz de usuario en Android: Controles básicos (I). (19 de agosto de 2010). [Entrada de Blog]. Recuperado de: https://www.sgoliver.net/blog/interfaz-de-usuario-en-android-controles-+basicos-i/

Interfaz de usuario en Android: Controles básicos (III). (27 de agosto de 2010). [Entrada de Blog]. Recuperado de: https://www.sgoliver.net/blog/interfaz-de-usuario-en-android-controles-basicos-iii/

Interfaz de usuario en Android: Controles de selección (I). (7 de septiembre de 2010). [Entrada de Blog]. Recuperado de: https://www.sgoliver.net/blog/interfaz-de-usuario-en-android-+controles-de-seleccion-i/

Interfaz de usuario en Android: Controles de selección (IV). (11 de septiembre de 2010). [Entrada de Blog]. Recuperado