Tema 3: Programación avanzada y comunicaciones

¿Qué aprenderás?

- Mejorar la experiencia del usuario utilizando componentes gráficos avanzados.
- Crear aplicaciones que utilizan clases para establecer comunicaciones a través de Internet con servidores de datos externos.
- Valorar que opción de persistencia de datos es la más indicada para una aplicación.

¿Sabías que…?

- Los hilos se utilizan para comunicar las aplicaciones Android con servidores remotos.
- Android proporciona una base de datos relacional llamada SQLite que puede ser utilizada en cualquier aplicación.

3.1. Introducción

En esta unidad se tratarán aspectos más complejos de la programación en Android, como el uso de varias actividades, las comunicaciones, la persistencia y el uso de bases de datos.

Una aplicación Android puede estar formada por varias actividades. Estas actividades son capaces de comportarse de manera independientes o bien relacionarse con el resto de actividades intercambiando parámetros entre ellas. Este enfoque se traduce en una programación más ordenada ya que cada actividad puede tener una finalidad concreta y determinada.

La combinación de componentes gráficos avanzados permite dotar de una mayor interactividad a una aplicación, mejorando la experiencia de usuario.

3.1.1. Iniciar una segunda Actividad

Como se vio en el capítulo anterior, una Actividad provee al usuario una pantalla con la que puede interactuar. Por su parte, un Intent es un objeto que proporciona un vínculo en tiempo de ejecución entre dos componentes diferentes de Android.

La Actividad principal puede emitir la intención de iniciar una segunda Actividad y además pasarle información mediante el método *putExtra()*, pasando el nombre del parámetro y su valor, de la siguiente manera:

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void botonIniciarSegundaActividad(View v) {
        EditText editText = findViewById(R.id.etCampoNombre);
        String nombre = editText.getText().toString();

        Intent intent = new Intent(this, SegundaActividad.class);
        intent.putExtra("mensaje", nombre);
        startActivity(intent);
    }
}
```

Es necesario crear la segunda Actividad, añadiéndola al proyecto. Si utilizamos la opción de Android Studio para crear una nueva Actividad en blanco (*Empty Activity*), también nos creará un layout en blanco. Esta segunda Actividad puede recibir los parámetros enviados desde la primera mediante el método *getStringExtra()* indicando el nombre del parámetro que se quiere recibir tal y como realiza el siguiente código:

```
public class SegundaActividad extends AppCompatActivity {

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

        Intent intent = getIntent();
        String nombre= intent.getStringExtra("mensaje");

        TextView textView = findViewById(R.id.etMensaje);
        textView.setText("Hola " + nombre);

    }
}
```

De esto modo podemos construir una aplicación con varias pantallas, cada pantalla siendo una Actividad diferente.

3.1.2. Componentes gráficos basados en listas

Añadir componentes gráficos más avanzados permite dotar de mayor interactividad a una aplicación. Con un funcionamiento parecido al del Spinner visto en el capítulo anterior, el ListView consiste en una lista visible de elementos en donde podemos personalizar el tipo de datos que se mostrarán. El ejemplo típico de uso de estos componentes es una lista de contactos, una lista de clientes o una lista de productos.

En primer lugar, se añade el componente de tipo lista (ListView o RecycleView) al layout utilizando el modo de diseño de layouts en Android Studio.

A continuación, se programa mediante un adaptador el tipo de datos que mostrará el listado. Es posible también definir un layout específico para indicar al componente cómo debe mostrar los datos.

El método setAdapter() asigna el adaptador al componente de tipo lista y únicamente habrá que programar la respuesta cuando se produzca un clic sobre un elemento de la lista.

El siguiente vídeo nos muestra cómo definir y utilizar una lista desplegable.

4. Listas desplegables (I)

5. Listas 2.1 (parte 1). Listas desplegables (ListView)

En la segunda parte del video, se cargan los datos del adaptador en el layout que se ha creado para obtener una lista de la compra que muestra los productos como una serie de imágenes (bitmaps) y mediante dos botones es posible añadir o quitar elementos en la lista.

5. Listas 2.1 (parte 2). Listas desplegables interactivas (ListView)

5. Listas 2.2 (parte 1)

5. Listas 2.2 (parte 2).

A partir de la versión 5.0 de Android (Lollipop), la API incluye un nuevo componente tipo lista llamado RecycleView. Este componente viene a sustituir el ListView que en Android Studio ha pasado a la categoría Legacy, es decir, componentes que existen por compatibilidad con versiones anteriores. El RecyclerView es una versión mejorada del ListView tradicional si bien su funcionamiento es muy similar.

Recordemos a modo de sumario que tanto el RecycleView como el ListView funcionan de manera similar. En primer lugar, se añade el componente de tipo lista (ListView o RecycleView) al layout utilizando el modo de diseño de layouts en Android Studio.

A continuación, se programa mediante un ArrayAdapter que leerá la fuente de datos y los mostrará en el listado. Se puede entonces definir un layout específico para indicar al componente cómo debe mostrar los datos. Finalmente, se capturan los eventos sobre la lista, por lo general para saber qué elemento de la lista ha sido seleccionado y qué respuesta damos a esa selección.

3.2. Comunicaciones en Android

Los dispositivos Android incorporan diversas opciones de conectividad y localización de la ubicación. En el archivo AndroidManifest.xml se deben indicar los permisos para que la aplicación pueda utilizar el dispositivo y establecer una comunicación:

```
<uses-permission android:name="android.permission.INTERNET" />
```

El protocolo para transferir datos hacia y desde Internet es HTTPS, versión segura del protocolo HTTP. Este protocolo encapsula los datos que se envían y reciben de manera segura utilizando el protocolo SSL.

A la hora de programar aplicaciones Android que utilizan las diferentes opciones de comunicaciones, hay que tener en cuenta la estabilidad de la aplicación en los momentos de conexión, envío y recepción de datos, y desconexión. Para ello se utiliza un modelo de hilos: las tareas de conexión y desconexión, descarga y subida de datos se realizan en segundo plano con el objetivo de no bloquear la interfaz de usuario de la aplicación.

Además de Thread, la clase estándar del lenguaje Java para el manejo de hilos, el SDK de Android proporciona la clase AsyncTask: se trata de una interfaz más sencilla que facilita la ejecución de tareas en segundo plano dentro del contexto del ciclo de vida de la aplicación. Esta clase pone a nuestra disposición los siguientes métodos:

- El método *onPreExecute()* se ejecuta antes de iniciar la tarea.
- El método *doInBackground()* se encarga de realizar una tarea en segundo plano.
- El método *onProgressUpdate()* permite mostrar una barra de progreso.
- El método *onPostExecute()* se ejecuta una vez que la tarea se segundo plano ha finalizado.

Vamos a ver un pequeño ejemplo de código que utiliza la clase AsyncTask para conectarse a un servidor remoto.

```
public class MainActivity extends AppCompatActivity {
    EditText mensaje;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mensaje = findViewById(R.id.etCampoNombre);

        // AsyncTask que se se conecta a un servidor
        new MiTarea().execute("https://www.google.es");
    }

    private class MiTarea extends AsyncTask<String, Integer, String> {

        @Override
        protected String doInBackground(String... direccion) {
            String servidor = direccion[0];
            String resultado = new String();
            HttpURLConnection conexion = null;
            try {
                URL url = new URL(servidor);
                conexion = (HttpURLConnection) url.openConnection();
                conexion.setRequestMethod("GET");
                conexion.connect();
                resultado = "¡Conectado!";
            } catch (Exception e) {
                resultado = "error";
            } finally {
                return resultado; // se lo pasa a postExecute()
            }
        }

        @Override
        protected void onPostExecute(String resultado) {
            super.onPostExecute(resultado);
            mensaje.setText(resultado.toString());
        }
    }

}
```

En este ejemplo, se utiliza la clase AsyncTask para conectarse al servidor con URL https://www.google.es. El método *doInBackground()* realiza la conexión mediante la clase HttpURLConnection. El resultado de la conexión se pasa al método *onPostExecute()* que se ejecuta antes de finalizar el hilo.

3.3. Persistencia y bases de datos en Android

La persistencia de una aplicación, ya sea en Android o en cualquier entorno, es un concepto que hace referencia al modo en que los datos utilizados por la aplicación perduran en el tiempo incluso cuando la ejecución de la aplicación ha finalizado. En otras palabras, la persistencia está relacionada con el almacenamiento de los datos en un medio no volátil que posteriormente permita recuperar los datos guardados.

La persistencia en Android se puede conseguir de diversas maneras:

1. Es posible guardar los datos en un archivo como por ejemplo un archivo de texto, en la memoria interna del dispositivo.
2. Un archivo también puede guardarse en memoria externa (por ejemplo, una tarjeta de memoria extraíble).
3. Android pone a disposición de las aplicaciones un sistema de Preferencias Compartidas (Shared Preferences) al que cualquier aplicación puede tener acceso. Este sistema, parecido al Registro utilizado en Windows, implementa además modos de acceso de manera que algunos datos pueden ser accesibles por todas las aplicaciones o bien únicamente por una aplicación en concreto.
4. SQLite es una base de datos relacional muy ligera disponible en el sistema operativo Android. Esta base de datos nos permite almacenar información para después poder recuperarla mediante consultas SQL.

SQLite es un sistema gestor de bases de datos contenido en una biblioteca escrita en lenguaje C, que pasa a formar parte de la aplicación. Los datos se guaran en un único archivo sobre el que se ejecutan las sentencias SQL. Para programar aplicaciones que utilizan SQLite se utiliza el patrón de diseño MVC (Modelo Vista Controlador).

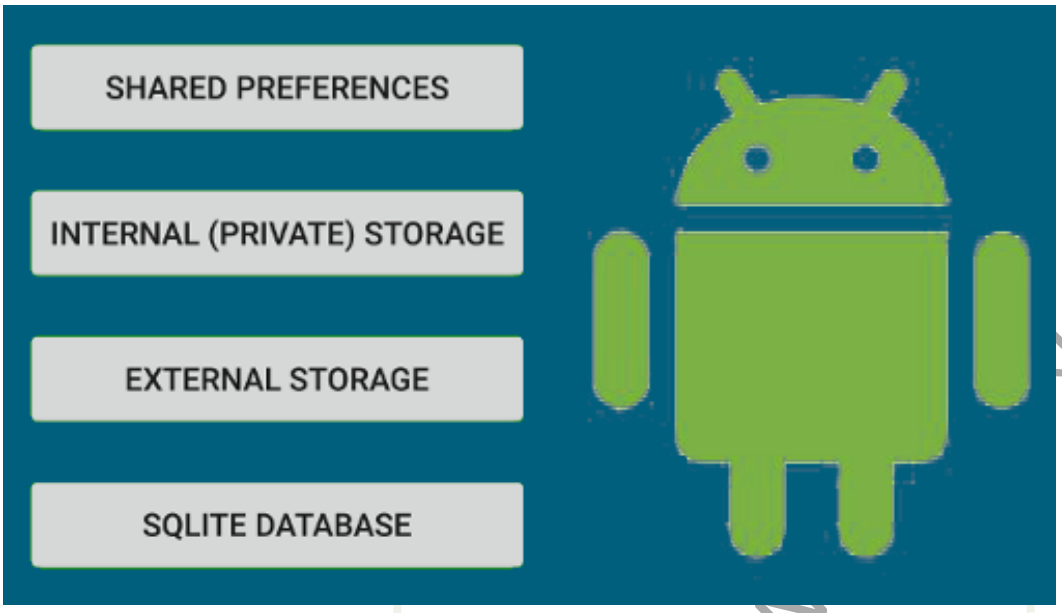

Figura: Opciones de persistencia en Android

Recursos y enlaces

- Guías para desarrolladores de Android

- Concurrency in Java

Conceptos clave

- **Actividad**: Una Actividad representa una pantalla individual con una interfaz de usuario.
- **Hilos:** Los hilos son pedazos de código de un programa que pueden ejecutarse a la vez que otros trozos de código. Están relacionados con el concepto de multitarea.
- **Intent**: Un objeto que se utiliza para solicitar una acción determinada de otro componente de una aplicación. Se utilizan para iniciar una Actividad o un Servicio.
- **Persistencia:** Es la capacidad que tienen los datos de una aplicación para guardarse y volver a ser utilizados en otro momento. Las variables contienen valores que desaparecen al final de la ejecución de una aplicación a menos que se utilice un mecanismo de persistencia que permita salvaguardar los datos. Entre los mecanismos de persistencia más habituales se encuentran los archivos y las bases de datos.
- **Servicio:** Componente que permite mantener la ejecución de la aplicación en segundo plano.

Test de autoevaluación

1. ¿Qué objeto se utiliza como un vínculo entre dos componentes Android y permite que una Actividad principal pase información a una segunda Actividad?
   a) Activity.
   b) Intent.
   c) Message.
   d) Ningua de las respuestas anteriores es correcta.

2. ¿Qué método de la clase AsyncTask se encarga de realizar una tarea en segundo plano?
   a) onPreExecute().
   b) onProgressUpdate().
   c) onPostExecute().
   d) doInBackground().

3. ¿En qué lugar puede guardar sus datos una aplicación Android?
   a) En un archivo en memoria interna o externa.
   b) En las Preferencias Compartidas.
   c) En una base de datos SQLite.
   d) Todas las respuestas anteriores son correctas.

4. ¿De qué tipo son las sentencias SQL que podemos ejecutar sobre una base de datos SQLite?
   a) Select.
   b) Insert.
   c) Update y Delete.
   d) Todas las respuestas anteriores son correctas.

Ponlo en práctica

Actividad 1

Crea un nuevo proyecto en Android Studio que utilice la clase AsyncTask para conectarse a un servidor remoto y descargar un fichero en formato JSON que nos indique la IP de nuestro dispositivo. La dirección del servidor es la siguiente: http://ip.jsontest.com y la estructura del fichero JSON únicamente contiene la etiquetas "ip" y su valor:

```
{"ip": "2.152.163.232"}
```

Solucionarios

Test de autoevaluación

1. ¿Qué objeto se utiliza como un vínculo entre dos componentes Android y permite que una Actividad principal pase información a una segunda Actividad?
   a) Activity.
   b) **Intent.**
   c) Message.
   d) Ningua de las respuestas anteriores es correcta.

2. ¿Qué método de la clase AsyncTask se encarga de realizar una tarea en segundo plano?
   a) onPreExecute().
   b) onProgressUpdate().
   c) onPostExecute().
   d) **doInBackground().**

3. ¿En qué lugar puede guardar sus datos una aplicación Android?
   a) En un archivo en memoria interna o externa.
   b) En las Preferencias Compartidas.
   c) En una base de datos SQLite.
   d) **Todas las respuestas anteriores son correctas.**

4. ¿De qué tipo son las sentencias SQL que podemos ejecutar sobre una base de datos SQLite?
   a) Select.
   b) Insert.
   c) Update y Delete.
   d) **Todas las respuestas anteriores son correctas.**

Ponlo en práctica

Actividad 1

Crea un nuevo proyecto en Android Studio que utilice la clase AsyncTask para conectarse a un servidor remoto y descargar un fichero en formato JSON que nos indique la IP de nuestro dispositivo. La dirección del servidor es la siguiente: http://ip.jsontest.com y la estructura del fichero JSON únicamente contiene la etiquetas "ip" y su valor:

```
{"ip": "2.152.163.232"}
```

Solución:

La aplicación debe utilizar una clase que implemente AsyncTask. El método doInBackground() establecerá la conexión y la descarga del fichero en formato JSON; el método onPostExcute() parseará el fichero en formato JSON y mostrará la información en un campo de texto (TextView). Una vez creado el proyecto y el layout que puede contener únicamente un TextView, codificamos la clase AsyncTask tal y como se muestra a continuación.

```
public class MainActivity extends AppCompatActivity {
    TextView mensaje;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mensaje = findViewById(R.id.etMensaje);
        new MiTarea().execute("http://ip.jsontest.com");
    }

    private class MiTarea extends AsyncTask<String, Void, JSONObject> {
        @Override
        protected JSONObject doInBackground(String... servidor) {
            JSONObject resultado = new JSONObject();
            HttpURLConnection conexion = null;
            try {
                URL url = new URL(servidor[0]); // establece conexión
                conexion = (HttpURLConnection) url.openConnection();
                conexion.setRequestMethod("GET");
                conexion.connect();
                InputStream stream = conexion.getInputStream();
                BufferedReader reader =
                        new BufferedReader(new InputStreamReader(stream));
                StringBuffer buffer = new StringBuffer();
                String line = "";
                while ((line = reader.readLine()) != null) {
                    buffer.append(line + "\n");
                }
                resultado = new JSONObject(buffer.toString());
            } catch (Exception e) {
                resultado = new JSONObject().put("ip", "error");
            } finally {
                return resultado; // pasa el resultado a postExecute()
            }
        }

        @Override
        protected void onPostExecute(JSONObject resultado) {
            super.onPostExecute(resultado);
            try {
                String s = resultado.getString("ip");
                mensaje.setText(s);
            } catch (Exception e) {
            }
        }
    }
}
```