Capítulo 9
Tareas asíncronas y servicios

1. Ejecutar acciones como tarea de fondo

En cualquier aplicación, el confort y la experiencia de usuario son aspectos esenciales: una aplicación debe, en smartphones y tabletas, reaccionar inmediatamente a cada solicitud del usuario. Para garantizar una respuesta óptima, la plataforma Android introduce una regla: de cualquier aplicación que no reaccione a una solicitud del usuario en un plazo de 10 segundos se dice que no responde. En tal situación, se produce un error ANR, de *Application Not Responding* («la aplicación no responde»), y el sistema puede detener la aplicación.

Para aquellas operaciones que pueden potencialmente tomar cierto tiempo (y cuya respuesta, por lo tanto, no es inmediata), se recomienda de forma especial realizar procesamientos asíncronos: la operación se ejecuta como tarea de fondo, y el usuario puede observar el progreso de la operación.

Si bien es posible implementar dicho mecanismo utilizando las tipicas clases de gestión de threads de Java, la plataforma provee una clase abstracta, `android.os.AsyncTask`, que se encarga de la mayor parte de la implementación de una solución asíncrona, aliviando el trabajo del desarrollador.

```
AsyncTask<Params, Progress, Result>
```

Los tipos genéricos `Params`, `Progress`, `Result` representan, como se detalla aquí arriba, los tipos de datos que se pasan como parámetro o devueltos por los métodos expuestos por la clase.

Para simplificar el diseño de una operación de tarea de fondo, `AsyncTask` separa el procesamiento en varias fases; cada una está representada por un método.

- `void onPreExecute()`: este método se ejecuta cuando se lanza la tarea asíncrona. Se ejecuta en el thread principal, lo que permite manipular los componentes de la interfaz de usuario. No hay que invocar este método directamente, sino invocar el método `execute`, que lanza el procesamiento.
- `Result doInBackground(Params... params)`: este método es abstracto, debe sobrecargarse obligatoriamente, y se ejecuta en un thread como tarea de fondo cuando termina el método `onPreExecute`. En este método debe realizarse el procesamiento, y no permite llevar a cabo ninguna operación sobre los componentes de la interfaz. `doInBackground` recibe como parámetro un conjunto de datos de tipo genérico `Params` y debe devolver un objeto de tipo genérico `Result`.
- `void onPostExecute(Result result)`: este método se invoca tras el método `doInBackground`. Recibe como parámetro el objeto de tipo genérico `Result` devuelto por `doInBackground`. Este método se ejecuta por el thread principal y permite manipular componentes de la interfaz de usuario.
- `void onProgressUpdate(Progress... values)`: este método se ejecuta en el thread principal cuando se invoca el método `publishProgress` desdel método `doInBackground`. Su cometido es, principalmente, gestionar la visualización de una ventana emergente de progreso.

Observación

La noción de thread principal/thread de tarea de fondo no es banal: en efecto, es imposible manipular cualquier objeto que forme parte de la interfaz de usuario en un thread de tarea de fondo: se produce una excepción. Si bien, en el marco de la clase AsyncTask, esto no supone un problema, los métodos onPreExecute, onPostExecute y onProgressUpdate se ejecutan en el thread principal, lo que puede plantear problemas en ciertos casos. Para resolver esta limitación, la clase Activity expone el método runOnUIThread, que, como su propio nombre indica, fuerza la ejecución en el thread principal. runOnUIThread recibe como parámetro una instancia de la interfaz Runnable. En el capítulo Uso de Bluetooth Low Energy se muestra un ejemplo, en la sección Conectar un objeto.

En el proyecto LocDVD, la inserción de los DVD de ejemplo es una tarea que puede, potencialmente, llevar cierto tiempo: aquí hay tan solo unos pocos DVD, pero podríamos imaginar un archivo más extenso, que tomara tiempo en cargarse. Por ello resulta conveniente transformar este procesamiento en un procesamiento asíncrono.

- Edite el archivo `MainActivity.java` que contiene el método que realiza la inserción de los DVD de ejemplo.
- En la clase `MainActivity`, defina una clase `AsyncReadEmbeddedData` que extienda de la clase `AsyncTask<String, Integer, Boolean>`.

```
class AsyncReadEmbeddedData extends AsyncTask<String, Integer, Boolean> {
}
```

- Hay que sobrecargar los métodos `onPreExecute`, `doInBackground`, `onProgressUpdate` y `onPostExecute`:

```
class AsyncReadEmbeddedData extends AsyncTask<String, Integer, Boolean> {

    @Override
    protected void onPreExecute()  {
    }


    @Override
    protected Boolean doInBackground(String... params) {
        return false;
    }

    @Override
    protected void onProgressUpdate(Integer... values) {
```

```
    }

    @Override
    protected void onPostExecute(Boolean result)  {
    }

};
```

- El método `doInBackground` parte del código del método `readEmbeddedData` que realiza la lectura e inserción de los DVD de ejemplo. El nombre del archivo, en vez de indicarse directamente en el cuerpo del método, se pasa como parámetro de `doInBackground`.

El cuerpo de `doInBackground` es, de momento, el siguiente:

```
@Override
protected Boolean doInBackground(String... params) {
    String dataFile = params[0];
    InputStreamReader reader = null;
    InputStream file=null;
    BufferedReader bufferedReader=null;
    try {
        file = getAssets().open(dataFile);
        reader = new InputStreamReader(file);
        bufferedReader = new BufferedReader(reader);
        String line= null;
        while((line=bufferedReader.readLine())!=null) {
            String [] data = line.split("\\|");
            if(data!=null && data.length==4) {
                DVD dvd = new DVD();
                dvd.titulo = data[0];
                dvd.anyo = Integer.decode(data[1]);
                dvd.actores = data[2].split(",");
                dvd.resumen = data[3];
                dvd.insert(MainActivity.this);
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }  finally {
        if(bufferedReader!=null) {
            try {
                bufferedReader.close();
                reader.close();
                SharedPreferences sharedPreferences =
```

```
getSharedPreferences("com.ejemplo.locDVD.prefs",

Context.MODE_PRIVATE);
                SharedPreferences.Editor editor =
sharedPreferences.edit();
                editor.putBoolean("embeddedDataInserted", true);
                editor.commit();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    return false;
}
```

- Para informar al usuario del avance de la inicialización, hay que publicar con cada inserción el número de DVD insertados en la base de datos. Para ello, hay que implementar un contador e invocar, con cada iteración del bucle `while`, el método `publishProgress`:

```
try {

    int counter = 0;

    file = getAssets().open(dataFile);
    reader = new InputStreamReader(file);
    bufferedReader = new BufferedReader(reader);
    String line= null;
    while((line=bufferedReader.readLine())!=null) {
        String [] data = line.split("\\|");
        if(data!=null && data.length==4) {
            DVD dvd = new DVD();
            dvd.titulo = data[0];
            dvd.anyo = Integer.decode(data[1]);
            dvd.actores = data[2].split(",");
            dvd.resumen = data[3];
            dvd.insert(MainActivity.this);

            publishProgress(++counter);

        }
    }
} catch (IOException e) {
```

```
        e.printStackTrace();
    }
```

- `doInBackground` debe devolver verdadero si la inserción se ha producido correctamente, y falso en caso contrario. En el marco de la aplicación, resulta interesante además agregar un temporizador entre la inserción de dos DVD, para visualizar más fácilmente el procesamiento como tarea de fondo. A continuación se muestra una versión completa del método:

```
@Override
protected Boolean doInBackground(String... params) {
    boolean result = false;
    String dataFile = params[0];
    InputStreamReader reader = null;
    InputStream file=null;
    BufferedReader bufferedReader=null;
    try {
        int counter = 0;
        file = getAssets().open(dataFile);
        reader = new InputStreamReader(file);
        bufferedReader = new BufferedReader(reader);
        String line= null;
        while((line=bufferedReader.readLine())!=null) {
            String [] data = line.split("\\|");
            if(data!=null && data.length==4) {
                DVD dvd = new DVD();
                dvd.titulo = data[0];
                dvd.anyo = Integer.decode(data[1]);
                dvd.actores = data[2].split(",");
                dvd.resumen = data[3];
                dvd.insert(MainActivity.this);
                publishProgress(++counter);
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if(bufferedReader!=null) {
            try {
                 bufferedReader.close();
```