Capítulo 5
Acceso a datos

Práctica 5.1: Reconstrucción de una actividad

El primer modo de almacenar datos que vamos a ver puede que, incluso, ni siquiera deba ser categorizado como tal, dado que es el propio sistema quien hace la mayor parte del trabajo.

Cuando Android detecta una necesidad de memoria que debe atender y no tiene suficiente, podría eliminar actividades que estén ocultas. Si más adelante el usuario vuelve a ellas, las volverá a crear, y éstas tendrán que ser capaces de reconstruirse en el estado original, para que el usuario no note que fueron eliminadas temporalmente.

La destrucción y reconstrucción de las actividades ocurre también en situaciones menos límite. En concreto, cuando el usuario gira el teléfono, Android también reinicia la actividad. El objetivo no es otro que darla la oportunidad de reconstruirse utilizando un layout nuevo, específico de la nueva orientación.

En el guardado y recuperación del estado de la actividad entran en juego dos métodos:

- **onSaveInstanceState(Bundle outInstance):** es un método que debemos sobreescribir en la actividad, y que será invocado automáticamente por Android antes de destruirnos. El parámetro es un bundle, donde podremos guardar las parejas de atributo-valor que consideremos oportuno con lo mínimo imprescindible para reconstruir nuestro estado.
- **onCreate(Bundle savedInstanceState):** es el método al que Android invoca al iniciar una actividad. El parámetro será null cuando se haga una creación "fresca". Sin embargo, si se está reconstruyendo la actividad a partir de un estado anterior, el parámetro contendrá una copia del bundle que rellenamos en la llamada a onSaveInstanceState().

Vamos a enriquecer la comprobación del ciclo de vida con este nuevo método invocado automáticamente.

- Haz una copia de la práctica del ciclo de vida (práctica 3.8) y llámala CicloDeVidaSaveInstance.
- Modifica el código del método onCreate para que muestre el contenido del parámetro:

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_ciclo_de_vida);
        android.util.Log.i(TAG,
                "onCreate(" + savedInstanceState + ")");
}
```

- Añade un método onSaveInstance(Bundle):

```
protected void onSaveInstanceState(Bundle outInstance) {
        super.onSaveInstanceState(outInstance);
        android.util.Log.i(TAG, "onSaveInstanceState");
}
```

- Lanza la aplicación y observa el logcat. Prueba a rotar el móvil con la aplicación lanzada, a volver al "escritorio" y relanzarla, o a cerrarla con el botón Volver para comprobar cuándo se invoca al método onSaveInstance().

Android garantiza que se llamará a **onSavedInstance**() antes que a **onStop**(), pero el orden relativo entre **onSavedInstance**() y **onPause**() es arbitrario.

Práctica 5.2: Púlsame inmune a rotaciones

Para hacer uso del Bundle, vamos a recuperar la aplicación Púlsame de la práctica 2.10 y a guardar su estado cuando sea necesario para que, al girar el móvil, el usuario no note la destrucción y recreación de la actividad.

- Haz una copia del proyecto PulsameLayout y llámalo PulsameEstable.
- Ejecútala. Pulsa varias veces sobre el botón. Comprueba que, efectivamente, el contador aumenta.
- Rota el móvil. Verás que la cuenta se reinicia.
- Vamos a meter la persistencia en la aplicación. Lo único que tenemos que mantener para reconstruir el estado es el valor de la variable _numVeces.
- En el código en Java, añade un nuevo atributo estático que guardará la constante con el nombre que usaremos como atributo para el bundle:

```
private final static String STATE_NUM_VECES = "numVeces";
```

- Ahora, sobreescribe el método **onSaveInstanceState**() e introduce en el bundle del parámetro el valor del atributo _numVeces.

```
@Override
public void onSaveInstanceState(Bundle outInstance) {

        // Llamamos al método de la superclase.
        super.onSaveInstanceState(outInstance);

        // Guardamos el único valor que nos interesa.
        outInstance.putInt(STATE_NUM_VECES, _numVeces);

} // onSaveInstanceState
```

En el método onCreate(), vamos a tener que recuperar ese mismo valor, y actualizar la etiqueta del botón en función a él cuando estemos reconstruyendo la actividad. En el

método botonPulsado() del evento de pulsación del botón ya tenemos código con el que establecemos la etiqueta. Para no repetirlo, lo refactorizamos creando un nuevo método que cambie la etiqueta en función del valor de _numVeces:

```
private String getEtiquetaBoton() {

        android.content.res.Resources res = getResources();
        String numPulsados;
        numPulsados = res.getQuantityString(
                                R.plurals.numPulsaciones,
                                _numVeces, _numVeces);

        return numPulsados;

} // getEtiquetaBoton
```

- Modifica el método del evento botonPulsado() para que utilice el nuevo método:

```
public void botonPulsado(View v) {

        // Incrementamos el contador...
        ++_numVeces;

        // ... y actualizamos la etiqueta del botón.
        Button b = (Button) v;
        b.setText(getEtiquetaBoton());

} // botonPulsado
```

- Ahora, modificamos el método onCreate(). Diferenciaremos si estamos realizando una inicialización nueva, o a partir de datos guardados previamente dependiendo de si el parámetro Bundle que recibimos es o no null:

```
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_pulsame);

        if (savedInstanceState != null) {
                _numVeces = savedInstanceState.getInt(
                                        STATE_NUM_VECES);

                if (_numVeces != 0) {
                        Button b = (Button) findViewById(R.id.boton);
                        b.setText(getEtiquetaBoton());
                }
        } // if (savedInstanceState != null)
```

} // onCreate

- Ejecuta de nuevo el programa. Pulsa varias veces el botón y rota el móvil. Comprueba que, ahora sí, se mantiene el contador correctamente.

Práctica 5.3: Mejorando el juego Adivina mi número

El juego Adivina mi número que hicimos en la práctica 3.15 tampoco reaccionaba bien si Android se veía obligado a eliminar temporalmente la actividad, por ejemplo ante la rotación del dispositivo.

En esta práctica el código que se muestra es únicamente informativo, dado que cada uno puede haber implementado el juego de una manera diferente.

- Haz una copia del proyecto con el juego Adivina mi número. Llámalo **AdivinaMiNumeroEstable**.
- Lánzalo. Haz un par de intentos de adivinar el número.
- Escribe un tercer número. En lugar de dar a aceptar, pulsa "volver" para cerrar el teclado en pantalla, y rota el móvil[1].
- La partida se ha reiniciado. Sin embargo, en el cuadro de texto se mantiene el último número que has escrito. Esto nos indica que la superclase android.app.Activity guarda por sí misma el estado de los controles en los que típicamente el usuario introduce datos (como ocurre con los EditText) en su método onSaveInstanceState(), y establece esos mismos valores en su método onCreate(). Ésa es una de las razones por las que resulta importante llamar siempre a la superclase en los métodos del ciclo de vida de las actividades.
- Para esta aplicación, necesitamos reconstruir el estado de nuestras variables donde guardamos el número que hay que adivinar (asumiremos que se llama _numElegido) y el número de intentos hechos (asumiremos _numIntentos). Reconstruir el atributo con el objeto generador de números aleatorios no es importante. También necesitamos saber el estado de la aplicación, para actualizar el interfaz. En concreto, si la partida ha terminado tendremos que ocultar algunos controles y mostrar el botón de jugar otra partida. Además, si la partida no ha terminado, necesitamos establecer el texto de la etiqueta superior para indicar si el último número era mayor o menor.
- Para conseguir todo esto, vamos a seguir algunos atajos. Vamos a guardar:
  - ✓ Último número elegido (el que se está usando en la partida en curso). Si la partida ha terminado, la variable _numElegido la pondremos a -1, lo que nos servirá de marca en el caso de que tengamos que reiniciar la actividad.
  - ✓ Número de intentos que llevamos.
  - ✓ Texto que estamos mostrando en la etiqueta superior. Podríamos guardar el último número introducido y configurar la etiqueta manualmente, pero guardando el texto nos ahorramos trabajo.

---

1 Si estás usando un AVD y estás escribiendo directamente en el teclado del ordenador, no pulses "Volver", pues cerraría la aplicación.

- Modifica la aplicación para que cuando se detecte el final de la partida se ponga _numElegido en -1.
- Define tres constantes estáticas de cadena con los nombres que utilizarás en el bundle de mantenimiento de estado para los tres datos que vamos a guardar:

```
private static final String STATE_NUM_ELEGIDO = "numElegido";

private static final String STATE_NUM_INTENTOS="numIntentos";

private static final String STATE_MENSAJE = "mensajeActual";
```

- Implementa el método onSaveInstanceState() en el que guardes esostres valores.
- Modifica el método onCreate(). Al principio, tendrás que mantener la configuración del GUI y establecimiento de los listeners. La inicialización de la partida (para elegir un número aleatorio) tendrás que hacerla únicamente si no hemos recibido bundle (estamos iniciando una actividad desde cero). En otro caso, tendrás que establecer las variables _numElegido y _numIntentos. Además, si la partida está terminada, tendrás que ocultar los controles de introducción del número y mostrar el botón de jugar de nuevo, y si la partida está en marcha tendrás que establecer el texto de la etiqueta del mensaje superior y del número de intentos. El resultado debería ser un código similar al siguiente:

```
if (savedInstanceState == null) {
        // Estamos iniciándonos desde cero.
        // Comenzamos una partida.
        reiniciaPartida();
}
else {
        // Tenemos que reconstruirnos desde el bundle.
        _numElegido = savedInstanceState.getInt(
                                        STATE_NUM_ELEGIDO);
        _numIntentos = savedInstanceState.getInt(
                                        STATE_NUM_INTENTOS);
        if (_numElegido == -1)
                // La partida está terminada.
                partidaAcabada();
        else {
                // La partida está a mitad. Ponemos el texto de la
                // etiqueta superior.
                TextView tv =
                tv = (TextView)findViewById(R.id.etiquetaSuperior);
                tv.setText(
                savedInstanceState.getString(STATE_MENSAJE));
                // Actualizamos la etiqueta del número de intentos
                actualizarIntentos();
        }
}
```

Práctica 5.4: Almacenando preferencias

El estado de la aplicación que se almacena gracias al bundle recibido en el método onSaveInstanceState() lo gestiona el propio sistema y su tiempo de vida es corto; únicamente sirve para reconstruir la actividad a su estado anterior si fue Android quién la destruyó. Pero si la actividad se eliminó porque, por ejemplo, el usuario pulsó "volver", entonces la creación de nuestra actividad partirá de cero.

Si queremos que algo de nuestro estado perdure, necesitaremos preocuparnos nosotros mismos de guardarlo. Un ejemplo habitual es el almacenamiento de preferencias de la aplicación, que son leídas durante la inicialización.

Android nos facilita la tarea a través de la clase **SharedPreferences**. En esencia, es similar a un bundle, que se lee y escribe en un fichero XML cuya localización, lectura y escritura es gestionada automáticamente por el sistema.

Cada aplicación tiene un directorio particular para guardar sus ficheros de preferencias. Además, es posible tener más de un fichero, diferenciándolos por nombre. Para conseguir uno de estos "almacenes de preferencias" basta con:

```
SharedPreferences preferencias;

preferencias = getSharedPreferences(<nombre>, <tipoAcceso>);
```

El tipo de acceso será uno de los siguientes:

- Context.MODE_PRIVATE: el fichero será accesible únicamente para la aplicación.
- Context.MODE_WORLD_READABLE: el fichero podrá ser leído por el resto de aplicaciones.
- Context.MODE_WORLD_WRITABLE: el fichero podrá ser escrito por el resto de aplicaciones.

Puedes combinar los dos últimos concatenándolos con la OR a nivel de bits (|).

Una vez conseguido el objeto, obtener los atributos que contiene es similar a como lo haríamos con un bundle:

```
int v = preferencias.getInt(<atributo>, <valorPorDefecto>);
```

El valor por defecto será el devuelto en el caso de que no exista en las preferencias el atributo solicitado.

La escritura de las preferencias requiere la ayuda de una clase adicional:

```
SharedPreferences.Editor editor;
editor = preferencias.edit();
```

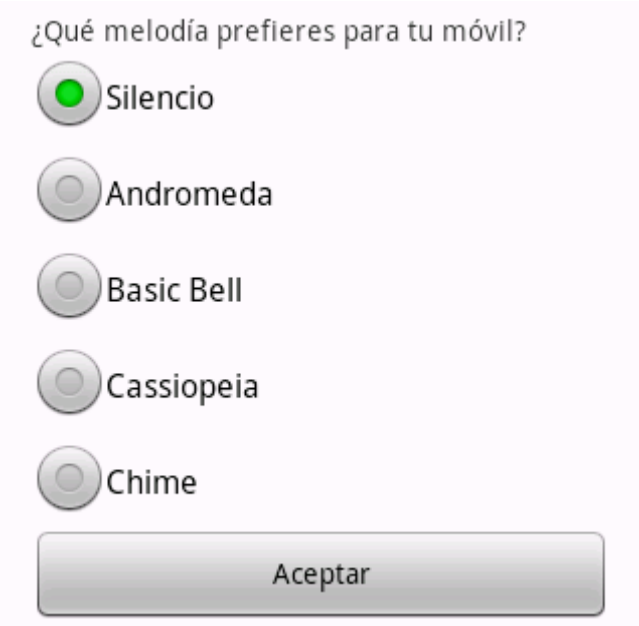
```
editor.putInt(<atributo>, <valor>);
...

editor.commit();
```

Naturalmente, aunque en ambos ejemplos se ha utilizado un entero como tipo de datos, tenemos disponibles muchos otros.

Por conveniencia, la clase **Activity** proporciona, además, un método **getPreferences(<modoApertura>)** que devuelve un objeto de preferencias asociado a su propio nombre. La intención es que se utilice ese método para conseguir un objeto en el que guardar las preferencias particulares de la actividad, y que se use el método con nombre cuando se estén guardando datos que vayan a ser leídos desde cualquier parte de la aplicación.

Para esta práctica vamos a hacer una actividad nueva que simulará un cuadro de diálogo de selección de opciones. Mostrará varios botones de radio, y el usuario deberá elegir una de las opciones que ofrecen. Tendrá un botón de Aceptar que, al ser pulsado, guardará la selección, de modo que la próxima vez que se lance la actividad aparecerá seleccionada.



- Puedes partir de cero creando el proyecto o, si lo prefieres, comenzar basándote en la práctica 4.2. En ese caso:

  - ✓ Haz una copia del proyecto original y renómbralo a Preferencias.
  - ✓ Cambia el nombre del paquete de dm2e.radiobutton a dm2e.preferencias. Puedes usar la refactorización proporcionada por Android Studio; asegúrate de que también lo cambia en el fichero de manifiesto (o hazlo tú manualmente si Android Studio no lo hace).
  - ✓ Cambia el nombre de la aplicación en el fichero strings.xml.

- ✓ Modifica el layout para quitar la etiqueta inferior con el mensaje de respuesta original, añade un ScrollView externo, y modifica los textos de los controles. Cambia también el identificador de cada botón de radio.

- Ejecuta la aplicación. Comprueba que, gracias a la implementación predefinida del método onSaveInstanceState(), se mantiene automáticamente la última selección cuando se rota el teléfono. Prueba a cambiar la selección y rotar varias veces para estar seguro.

- Sin embargo, si seleccionas cualquier opción (diferente a la de partida) y cierras el programa con el botón "Volver", la última selección se olvida y al lanzar el programa de nuevo comenzará con el primer botón de radio seleccionado.

- Define un atributo estático constante con el nombre del atributo que utilizaremos para guardar la última selección del usuario.

```
private final static String PREFERENCIA_MELODIA = "melodia";
```

- En el evento del botón de aceptar mete el código necesario para guardaren el fichero de preferencias particular de la actividad el último valor seleccionado. Por simplicidad, vamos a guardar directamente el identificador del control que está marcado. Normalmente, dado que este valor se utilizaría en otros lugares de la aplicación, declararíamos untipo enumerado para cada alternativa y guardaríamos su valor:

```
public void onAceptar(View v) {
        SharedPreferences preferencias;

        preferencias = getPreferences(Context.MODE_PRIVATE);
        SharedPreferences.Editor editor;
        editor = preferencias.edit();

        RadioGroup rg;
        rg = (RadioGroup) findViewById(R.id.preferenciasMelodia);

        editor.putInt(PREFERENCIA_MELODIA,
                                rg.getCheckedRadioButtonId());

        editor.commit();

        finish();

} // onAceptar
```

En el código del método onCreate() añade código para leer de las preferencias el botón de radio seleccionado la última vez que se aceptó el cuadro de diálogo. Si no se lee ninguno, elegiremos por defecto el primer botón de radio.

```
public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        SharedPreferences preferencias;
        preferencias = getPreferences(Context.MODE_PRIVATE);

        int id = preferencias.getInt(PREFERENCIA_MELODIA,
                R.id.silencio);

        RadioButton rb;
        rb = (RadioButton) findViewById(id);

        rb.setChecked(true);

} // onCreate
```

- Ejecuta la aplicación en un AVD. Comprueba que tras pulsar aceptar, se consigue que, efectivamente, en ejecuciones posteriores se recupere la selección, y que, sin embargo, al pulsar cancelar no ocurre lo mismo.
- Usando adb, abre un shell con el AVD en el que hayas ejecutado el programa[2].
- En el shell, muevete por la jerarquía de directorios hasta llegar a /data/data/dm2e.preferencias/shared_prefs.
- Comprueba la existencia del fichero MainActivity.xml (el nombre de la actividad).
- Haz un cat del fichero para ver la estructura del XML. Comprobarás que dentro está la entrada para nuestro atributo. El valor es el identificador del botón de radio.
- Si utilizas ls -l, podrás ver los permisos del fichero. Comprueba que el usuario y el grupo tienen acceso de lectura y de escritura. Ambos son el usuario particular que Android ha asociado a nuestra aplicación (app_xx).
- Vuelve al código java y sustituye las dos apariciones de MODE_PRIVATE por MODE_WORLD_READABLE. Vuelve a ejecutar el programa, modifica la selección y vuelve a mirar los permisos del fichero. Verás que el resto del mundo tiene ahora acceso de lectura.
- Por último, utiliza MODE_WORLD_WRITABLE. Repite el proceso y comprueba que ahora los permisos permiten al resto del mundo escribir.

Práctica 5.5: Leyendo un fichero de los recursos

El directorio res/ de los .apk contiene ficheros que son automáticamente gestionados por Android. Durante la construcción de la aplicación, se genera el conocido fichero R.java, con identificadores a todos los recursos definidos, y el programador puede utilizarlos a través de diferentes llamadas al API de Android. Además de encargarse de su carga y procesamiento, también es capaz de escoger la versión del recurso que más se

2 Esta prueba no podrás hacerla si has ejecutado la aplicación en un dispositivo físico, dado que el shell no tiene priviliegios de administrador.

adapte a la configuración particular del dispositivo, como hemos visto con los recursos de tipo cadena.

Dentro de res/ podemos crear un directorio particular, res/raw/ (crudo), donde meter ficheros propios que no serán interpretados por Android (no son cadenas, ni imágenes, ni layouts) pero que podremos abrir muy fácilmente utilizando un identificador de recurso como en el resto de los casos. Además, si tenemos varios directorios raw con calificadores (por ejemplo raw-en/) el sistema se encargará de proporcionarnos el fichero que está en el directorio más apto para la configuración del sistema.

Para abrir uno de estos ficheros se utiliza un código como el siguiente:

```
//Abrir el fichero de la carpeta raw
InputStream is = getResources().openRawResource(R.raw.<id>);

//Abrir el fichero de la carpeta assets
InputStream stream = getAssets().open(inFile);
```

Fíjate que esto nos proporciona automáticamente un objeto de la clase InputStream, que es estándar de Java y que representa un flujo de bytes para ser consumidos. Ese stream podremos "envolverlo" en clases adicionales (como BufferReader o Scanner) para que nos facilite la lectura, al igual que haríamos con una aplicación en Java tradicional.

En esta práctica vamos a hacer una actividad que se parecerá a la de la práctica 5.4, mostrando un conjunto de botones de radio con opciones de melodías. En este caso, sin embargo, los títulos de las melodías las leeremos de un fichero situado en res/raw. Para eso, abriremos el fichero y leeremos línea por línea, y por cada una añadiremos un nuevo botón de radio a la actividad a través de programación.

Ten en cuenta que no pretendemos imitar el funcionamiento de la práctica 5.4. En concreto, no vamos a intentar guardar la selección a modo de preferencias, dado que al generar dinámicamente los botones de radio no tendremos sus identificadores y necesitaríamos algún otro mecanismo para guardar la opción elegida.

- Crea un nuevo proyecto y denomínalo CargaResRaw.

Configura el layout de la actividad para que tenga un ScrollView como raíz con un LinearLayout dentro, que, a su vez, posea un RadioGroup vacío y el botón de aceptar.

```
<ScrollView
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" >
        <LinearLayout
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
```

```
            android:orientation="vertical">
            <RadioGroup
                android:id="@+id/rgOpciones"
                android:layout_width="wrap_content"
                android:layout_height="fill_parent"
                android:orientation="vertical">
            </RadioGroup>
            <Button
                android:id="@+id/bAceptar"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="@android:string/ok"
                android:onClick="onAceptar"/>
        </LinearLayout>
</ScrollView>
```

- En el Package Explorer de Android Studio, pulsa con el botón derecho sobre el directorio res/ y crea un nuevo directorio (New - Folder) y llámalo raw.

- Pulsa con el botón derecho sobre el nuevo directorio, y crea un nuevo fichero (New - File) y llámalo melodias.txt.
- Abre el fichero recién creado, y mete varias líneas de texto con lo que serían las melodías disponibles.

```
Silencio
Andrómeda
Basic Bell
Cassiopeia
Chime
A cricket chirps
Crossing walk
Cuisine
Down hill
Emotive sensation
Eridani
Faint
Happy synth
Illuminator
Lira
```

- Abre el fichero de código fuente en Java de la actividad. En el evento onCreate() vamos a añadir el código para abrir el fichero, leer línea por línea, y por cada una añadir un nuevo botón de radio con el nombre de la melodía como texto. Para eso, crea un objeto nuevo de la clase RadioButton, establece el texto, y añádelo al radio group de la actividad que has metido en el layout. Cuando hayas metido todos los botones, marca como seleccionado el primero de ellos.

```
public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_carga_res_raw);

                RadioGroup rg;
                rg = (RadioGroup) findViewById(R.id.rgOpciones);
                InputStream is;
                is = getResources().openRawResource(R.raw.melodias);
                Scanner sc = new Scanner(is);
                while (sc.hasNextLine()) {
                        String melodia = sc.nextLine();
                        RadioButton rb;
                        rb = new RadioButton(this);
                        rb.setText(melodia);
                        rg.addView(rb);
                }
                ((RadioButton)rg.getChildAt(0)).setChecked(true);
} // onCreate
```

- Ejecuta el programa. Verás que se muestra un botón de radio por cada línea que hayas metido en el fichero.
- Para comprobar que Android aplica también los cualificadores a la elección del fichero, crea un nuevo directorio res/raw-en, copia el fichero original sobre él, y modifícalo. Para probar que, efectivamente, Android leerá del nuevo fichero es suficiente con que cambies una de las líneas (por ejemplo, sustituye **Silencio por Silent).**
- Ejecuta de nuevo la aplicación, asegurándote de que tienes la configuración del AVD con el idioma en inglés. Comprueba que ahora el fichero que se abre es el nuevo.

Ten en cuenta que esta aplicación es sólo un ejemplo para probar la carga desde ficheros de recursos _crudos_. La utilidad que pretende tener (hacer las veces de preferencias del teléfono para elegir una melodía) requeriría todavía bastante más trabajo de programación.

Además, el uso de ficheros en res/raw tiene la ventaja de ser gestionado automáticamente por Android y sus identificadores, pero eso significa que no resulta sencillo averiguar qué ficheros tenemos, ni podemos tener subdirectorios. Si queremos hacer este tipo de tareas (y que nuestros ficheros sigan estando empaquetados en el .apk) deberíamos utilizar el directorio assets/, aunque entonces quedamos fuera de los identificadores del fichero R.java.

Práctica 5.6: Leyendo y escribiendo en ficheros externos

El uso de los ficheros en el directorio res/raw o assets/ del .apk tienen el problema de que son de sólo lectura. Si queremos utilizar ficheros para guardar información de cualquier tipo y no nos sirven las preferencias que vimos en la práctica 6.4, tendremos que utilizar otro mecanismo.

En concreto, para escribir en ficheros particulares de las actividades, éstas nos proporcionan el método:

FileOutputStream openFileOutput(String nombre, int tipoAcceso);

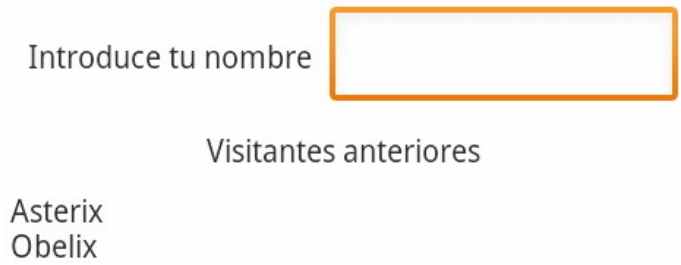
que nos devuelve una instancia de una clase tradicional de Java que representa un stream de un fichero. Como antes, podremos utilizar clases auxiliares para recubrirlo y que nos facilite la escritura.

El parámetro llamado tipoAcceso es equivalente al que tenía el método getSharedPreferences() que vimos en la práctica 6.4. Además, se añadela posibilidad de utilizar Context.MODE_APPEND, en cuyo caso todo lo que escribamos se añadirá por el final.

Para la lectura, podemos utilizar el método simétrico:

FileInputStream openFileInput(String nombre);

Para ponerlos en práctica, vamos a realizar una aplicación que muestre un "libro de visitas". En la parte superior mostrará un cuadro de texto en el que el usuario podrá escribir su nombre, que se irá acumulando a los visitantes. Cada nuevo nombre quedará registrado en el fichero de visitantes, y cuando se arranque el programa se mostrarán todos los que se han registrado anteriormente.



- Crea un proyecto nuevo y llámalo LibroDeVisitas.
- Configura el layout de la actividad para que tenga el aspecto de la _gura. Tendrá tres etiquetas, y un cuadro de texto.

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:padding="10dp">
<TextView
android:id="@+id/tvNombre"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentLeft="true"
android:layout_alignBaseline="@+id/etNombre"
android:layout_marginRight="10dp"
```

```
android:text="@string/introduceNombre"
tools:context=".LibroVisitas" />
<EditText
android:id="@+id/etNombre"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_toRightOf="@+id/tvNombre"
android:layout_alignParentRight="true"
android:layout_alignParentTop="true"
android:inputType="text"/>
<TextView
android:id="@+id/tvVisitantes"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerHorizontal="true"
android:layout_below="@id/etNombre"
android:text="@string/visitantesAnteriores"/>
<ScrollView
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:layout_below="@id/tvVisitantes">
<TextView
android:id="@+id/etVisitantes"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:gravity="top"/>
</ScrollView>
</RelativeLayout>
```

Para crear el borde naranja del EditText, creamos un fichero en la carpeta drawable (borde.xml) y escribimos el siguiente código.

```xml
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
android:shape="rectangle">
    <solid android:color="#ffffff"/>
    <!-- grosor del borde y color-->
    <stroke android:width="3dp"
        android:color="#ffaa00"/>
    <!-- espacio estre el borde y el texto-->
    <padding android:left="10dp"
        android:top="2dp"
        android:right="10dp"
        android:bottom="2dp"/>
    <!-- esquinas redondeadas-->
    <corners android:radius="25px"/>
</shape>
```

En la etiqueta del EditText añadimos la siguiente línea.

```xml
android:background="@drawable/borde"
```

Añade al fichero strings.xml las cadenas correspondientes.

- En el método onCreate() de la actividad, busca el cuadro de texto y registra un listener onNuevoNombre() para que se llame cuando el usuario pulse intro en él. Además, al terminar deberá llamar a un método actualizarVisitantes() que lea del fichero los nombres registrados y los añada a la etiqueta de las visitas.

```
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_libro_visitas);
EditText et;
et = (EditText) findViewById(R.id.etNombre);
et.setOnKeyListener(
new android.view.View.OnKeyListener() {
public boolean onKey(View v,
int keyCode,
android.view.KeyEvent event) {
// Han pulsado una tecla...
if ((event.getAction() ==
android.view.KeyEvent.ACTION_DOWN) &&
(keyCode ==
android.view.KeyEvent.KEYCODE_ENTER)) {
// Ha sido "intro"
onNuevoNombre();
return true;
}
return false;
}
});
// Actualizamos la lista de visitantes a partir
// de los datos del fichero.
actualizarVisitantes();
} // onCreate
```

- Define una constante nueva donde mantendremos el nombre del fichero con los visitantes.

```
private final static String NOMBRE_FICHERO = "visitantes.txt";
```

- En el método onNuevoNombre, abre dicho fichero para escritura por el final, y añade el nombre leído del cuadro de texto:

```
protected void onNuevoNombre() {
EditText et;
et = (EditText) findViewById(R.id.etNombre);
String nuevoNombre;
nuevoNombre = et.getText().toString();
if (nuevoNombre.trim().equals("")) {
```

```
muestraMensaje(R.string.errorNombre);
return;
}
try {
FileOutputStream fos;
fos = openFileOutput(NOMBRE_FICHERO,
Context.MODE_PRIVATE |
Context.MODE_APPEND);
java.io.OutputStreamWriter out;
out = new OutputStreamWriter(fos);
out.write(nuevoNombre + "\n");
out.close();
muestraMensaje(R.string.bienvenido);
actualizarVisitantes();
}
catch (Exception e) {
muestraMensaje(R.string.errorRegistro);
}
InputMethodManager imm;
imm = (InputMethodManager)getSystemService(
Context.INPUT_METHOD_SERVICE);
imm.hideSoftInputFromWindow(et.getWindowToken(), 0);
et.requestFocus();
et.setText("");
} // onNuevoNombre
```

- El método actualizarVisitantes() tendrá que hacer la labor inversa. Leerá el fichero completamente, y añadirá cada nombre en la etiqueta de los visitantes:

```
protected void actualizarVisitantes() {
StringBuffer strBuf = new StringBuffer();
TextView tv;
tv = (TextView) findViewById(R.id.etVisitantes);
try {
String visitante;
FileInputStream fis;
fis = openFileInput(NOMBRE_FICHERO);
Scanner scanner = new Scanner(fis);
if (scanner.hasNextLine()) {
visitante = scanner.nextLine();
strBuf.append(visitante);
}
while (scanner.hasNextLine()) {
visitante = scanner.nextLine();
strBuf.append("\n" + visitante);
} // while
scanner.close();
```

```
tv.setText(strBuf.toString());
} // try
catch (Exception e) {
// Algo fue mal :-(
muestraMensaje(R.string.errorVisitantes);
} // try-catch
} // actualizarVisitantes
```

- Sólo nos falta el método auxiliar que muestra un mensaje en un toast:

```
protected void muestraMensaje(int id) {
Toast t;
String mensaje;
mensaje = getResources().getString(id);
t = Toast.makeText(this, mensaje, Toast.LENGTH_LONG);
t.show();
} // muestraMensaje
```

- Añade la definición de todas las cadenas utilizadas.
- Prueba la aplicación en un AVD e introduce varios nombres.
- Abre y cierra la aplicación varias veces, y constata que los nombres anteriores se mantienen.
- Abre una sesión de shell en el AVD.

```
$ ./avd shell
```

- Navega por los directorios hasta llegar al directorio de la aplicación, /data/ data/dm2e.librodevisitas/files
- Mira el contenido y verás el fichero visitantes.txt. Comprueba los permisos de acceso, coherentes con el MODE_PRIVATE usado, y mira el contenido para ver los visitantes.

Con este método de guardar ficheros, la ruta también es predefinida. En concreto, todos los ficheros creados así se almacenan en el directorio local de la aplicación. En ninguno de los dos métodos (de apertura para leer y para escribir) podemos utilizar símbolos de ruta en el nombre (/).