Práctica 3
Aplicación de Notas: implementación inicial y modelado estático y dinámico

Grado en Ingeniería Informática - Ingeniería del Software
Dpto. de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

1. Introducción y objetivos

Para realizar la presente práctica es necesario:

- Haber completado la práctica 1 de la asignatura.
- Tener un conocimiento básico de Modelio. La herramienta Modelio se presentó en la práctica 2 de la asignatura.
- Además deberemos tener claros los conceptos básicos sobre desarrollo Android que se explican en el Apéndice de la práctica 1.

Los objetivos de esta práctica son los siguientes:

1. Conocer un código de ejemplo para la creación de una aplicación de gestión de notas para dispositivos Android. Dicho código podrá utilizarse como base para realizar el trabajo de la asignatura.
2. Utilizar Modelio [1] para construir diagramas UML (tanto estáticos como dinámicos) a partir del código estudiado. Estos diagramas también podrán ser utilizados en el trabajo de la asignatura.

2. Actividades a realizar en la práctica

2.1. Introducción a la aplicación Notepad (versión 1)

En primer lugar, vas a descargar de Moodle el fichero «`NotepadCodeLab.zip`». El contenido de este fichero son varios proyectos Android, que se deben descomprimir en un directorio en tu cuenta. A continuación vamos a proponer una serie de modificaciones en el proyecto `Notepadv1` para ir familiarizándonos con el código de la aplicación `Notepad`.

Vas a construir una aplicación que permite crear notas pero no permite editarlas. Practicaremos:

- Los fundamentos de `ListActivity` para crear y manejar opciones de menú.
- Cómo usar el objeto `SQLiteDatabase` para almacenar notas.
- Cómo asociar información de un cursor a una `ListView`.
- Fundamentos del diseño de ventanas: diseñar una `ListView`, añadir `items` en el menú de una actividad, y veremos cómo la actividad maneja el menú.

Paso 1 Abre el proyecto `Notepadv1` tal y como aprendiste en la práctica 1.

Paso 2 Estudiar la clase `NotesDbAdapter`. Esta clase encapsula el acceso a la clase `SQLiteDatabase` que gestionará nuestras notas.

La clase declara constantes que serán los nombres de los campos de la base de datos. También se declara un `String` para crear la base de datos.

Estudiar el constructor de la clase. Este toma un `Context` que permitirá la comunicación con Android. La clase `Activity` implementa la clase `Context`, normalmente cuando necesitemos un `Context` lo pasaremos desde la Actividad usando `this`.

Estudiar los métodos `open()`, `close()`, `createNote()` y `deleteNote()`. Fijarse en el uso que hacen de un objeto de la clase `DatabaseHelper`. Esta clase es nuestra implementación de la clase `SQLiteOpenHelper`.

Estudiar el método `fetchAllNotes()`, en concreto los parámetros de la `query` al objeto `DatabaseHelper`. Fíjate ahora en el método que recupera una sola nota `fetchNote`.

Paso 3 Abre el fichero `activity_notepadv1.xml` en `res/layout`. Este contiene los elementos gráficos que aparecerán en nuestra Actividad, que será la pantalla que nos muestre la aplicación y contendrá la lista de notas. En él se define un `LinearLayout` como aprendiste en la práctica 1. Revisa el enunciado de la práctica 1 que explicaba este

fichero XML e identifica ahora los elementos que en él aparecen.

Paso 4 Vamos a crear en el *layout* un objeto `ListView` y otro `TextView`. Estos objetos serán vistas alternativas en nuestra aplicación. Cuando haya notas que mostrar aparecerá el objeto `ListView`, cuando no haya notas aparecerá el `TextView`. Sustituye el código que aparece en `activity_notepadv1.xml` por este:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <ListView android:id="@+id/list"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <TextView android:id="@+id/empty"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/no_notes"/>

</LinearLayout>
```

En la práctica 1 aprendiste el significado de los campos que aparecen en los objetos `ListView` y `TextView`, repásalos.

Paso 5 Para crear la lista de notas en la `ListView` necesitamos definir una Vista para cada fila:

1. Crea un nuevo fichero en `res/layout` que se llame `notes_row.xml`.
2. Añade en ese fichero lo siguiente:

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <TextView android:id="@+id/text1"
       xmlns:android="http://schemas.android.com/apk/res/android"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"/>
   ```

   Esta es la Vista que se utilizará para cada fila con nombre de nota, sólo tiene un campo de texto.
3. Guarda el fichero

Paso 6 Abre y estudia la clase `Notepadv1`. En los pasos siguientes modificaremos esta clase para convertirla en una Lista y mostrar las notas, también nos permitirá añadir nuevas notas.

Al definir el layout de la actividad `Notepadv1` se ha incluido un objeto de tipo `ListView`, que tiene funcionalidad para las acciones que se pueden hacer con una lista, por ejemplo: mostrar un número arbitrario de elementos de la lista en filas de la pantalla, moviéndose a través de los elementos de la lista, y permitiendo que sean seleccionados.

El atributo `mNoteNumber` lo utilizaremos para crear nombres de notas numerados.

También hay tres métodos redefinidos: `onCreate()`, `onCreateOptionsMenu()` y `onOptionsItemSelected()`:

- `onCreate()` se llama cuando se inicia la actividad (vendría a ser algo similar al método "main" de un programa). Lo utilizaremos para establecer los recursos y el estado de la actividad cuando se está ejecutando.
- `onCreateOptionsMenu()` se utiliza para rellenar el menú de la actividad cuando el usuario pulsa la tecla de menú del dispositivo Android, y tiene una lista de opciones (como "Crear nota").
- `onOptionsItemSelected()` se utiliza para controlar los eventos generados por el menú (por ejemplo, cuando el usuario selecciona la opción "Crear nota").

Paso 7 Añade un atributo privado de tipo `ListView` para acceder a la lista de notas que se mostrarán en la interfaz.

```
private ListView mList;
```

Tienes que importar `ListView`.

Paso 8 Rellena el método `onCreate()`.

Aquí pondremos el título de la Actividad (que se muestra en la parte superior de la pantalla), utilizaremos el layout `activity_notepadv1` que hemos creado en XML, crearemos la instancia `NotesDbAdapter` que tendrá acceso a los datos de notas y rellenaremos la lista con los títulos disponibles:

1. En el método `onCreate()`, llamar a `super.onCreate()` con el parámetro `savedInstanceState`.
2. Llamar a `setContentView()` y pasar como parámetro `R.layout.activity_notepadv1`.
3. En la parte inicial de la clase, crear un nuevo atributo privado llamado `mDbHelper` de la clase `NotesDbAdapter`.
4. En el método `onCreate()`, crear una instancia de `NotesDbAdapter` y asignarla al

atributo `mDbHelper` (`this` será un parámetro para este constructor)

5. Invoca al método `open()` sobre `mDbHelper` para abrir (o crear) la base de datos.

6. Inicializa el atributo `mList` recuperando el objeto `ListView` con el método `findViewById` y el identificador de este objeto gráfico, que se ha generado de forma automática en la clase `R`.

```
mList = (ListView)findViewById(R.id.list);
```

7. Por último, invoca al método `fillData()`, que recibirá los datos y llenará el ListView- todavía no hemos definido este método.

Paso 9 Rellena el método `onCreateOptionsMenu()`.

Vamos a crear el botón "AddItem" que estará accesible cuando presionemos el botón del menú en el dispositivo Android.

1. En el fichero `strings.xml` (en `res/values`), añadir una nueva cadena llamada "menu_insert" con valor "AddItem":

```
<string name="menu_insert">Add Item</string>
```

   A continuación guarda el fichero y vuelve a Notepadv1.

2. Crear una constante para el menú en la parte inicial de la clase:

```
public static final int INSERT_ID = Menu.FIRST;
```

3. En el método `onCreateOptionsMenu()` cambiar la llamada a super para capturar el valor de retorno y devolverlo.

4. Por último añade el item de menú con `menu.add()`. El método tendrá ahora este aspecto:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    boolean result = super.onCreateOptionsMenu(menu);
    menu.add(Menu.NONE, INSERT_ID, Menu.NONE, R.string.menu_insert);
    return result;
}
```

   Los argumentos de `menu.add()` indican: un identificador de grupo para este menú (ninguno, en este caso), un ID único (definido anteriormente), el orden del elemento (cero indica que no hay preferencia), y el recurso de la cadena a utilizar para el elemento.

Paso 10 Rellena el método `onOptionsItemSelected()`.

Este método manejará el ítem de menú que acabamos de añadir "Add Note". Cuando el ítem es pulsado, el método `onOptionsItemSelected()` es llamado con el identificador "INSERT_ID". Esto lo podemos detectar y tomar las acciones necesarias.

El método quedará como sigue:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case INSERT_ID:
        createNote();
        return true;
    }

    return super.onOptionsItemSelected(item);
}
```

Paso 11 Crea el método `createNote()`.

En esta primera versión `createNote()` no va a ser muy útil. Simplemente, creará una nueva nota con un título asignado utilizando un contador ("Nota 1", "Nota 2"...), el cuerpo de la nota estará vacío. En este momento no tenemos ninguna manera de editar el contenido de una nota, así que por ahora tendremos que conformarnos con crearlas con valores por defecto.

El método quedará como sigue:

```
private void createNote() {
    String noteName = "Note " + mNoteNumber++;
    mDbHelper.createNote(noteName, "Body text");
    fillData();
}
```

Paso 12 Crea el método `fillData()`.

Este método utiliza la clase `SimpleCursorAdapter`, que tiene un cursor de base de datos y lo une a los campos de nuestro `layout`. Estos campos definen los elementos de la lista (en este caso utilizamos el campo "text1" en `notes_row.xml`), así que esto nos permite rellenar fácilmente la lista con las entradas de la base de datos.

Para conseguir esto asociaremos el campo "title" devuelto por el `Cursor` con el campo "text1", que es un `TextView` que creamos anteriormente. Lo haremos con dos vectores: "from" y "to". Estudia el código del método:

```
private void fillData() {
```

```
        // Get all of the notes from the database and create the item list
        Cursor c = mDbHelper.fetchAllNotes();
        startManagingCursor(c);

        String[] from = new String[] { NotesDbAdapter.KEY_TITLE };
        int[] to = new int[] { R.id.text1 };

        // Now create an array adapter and set it to display using our row
        SimpleCursorAdapter notes =
            new SimpleCursorAdapter(this, R.layout.notes_row, c, from, to);
         mList.setAdapter(notes);
    }
```

Paso 13 Ejecuta el proyecto tal y como aprendiste en la práctica 1.

2.2. Creación de diagramas de clase y secuencia para la aplicación Notepad (versión 1)

Utilizando la herramienta CASE Modelio, construye el diagrama de clases correspondiente a la aplicación que acabas de modificar. En dicho diagrama deberían aparecer reflejadas las clases que se han ido modificando o explicado a través de los pasos descritos en la sección 2.1: *Notepadv1*, *AppCompatActivity*, *ListView*, *NotesDbAdapter*, *Context*, *SQLiteDatabase*, *DatabaseHelper* y *SQLiteOpenHelper*. Además, se añadirán los atributos, métodos y relaciones entre clases que faciliten la comprensión del diagrama de clases y sean relevantes.

Una vez que hayas creado el diagrama de clases, crea los diagramas de secuencia correspondientes a los siguientes escenarios:

1. Inicialización de la aplicación: secuencia de acciones desencadenadas por un usuario al abrir la aplicación.
2. Creación de una nota por parte del usuario.
3. Actualización de una nota. No se puede realizar desde la interfaz de usuario, así que solo se modelará la secuencia desde la clase *Notepadv1*.
4. Borrado de una nota. No se puede realizar desde la interfaz de usuario, así que solo se modelará la secuencia desde la clase *Notepadv1*.
5. Búsqueda de una nota. No se puede realizar desde la interfaz de usuario, así que solo se modelará la secuencia desde la clase *Notepadv1*.
6. Búsqueda de todas las notas. No se puede realizar desde la interfaz de usuario, así

que solo se modelará la secuencia desde la clase *Notepadv1*.

Los diagramas que has creado con Modelio son los diagramas que reflejan el desarrollo final de la aplicación Notepad. Compáralos con los diagramas de Análisis que propusiste en la práctica anterior para la aplicación de Notas. En clase de teoría, durante las sesiones de Diseño, aprenderás cómo los diagramas de Análisis evolucionan hacia diagramas de Diseño, que serán los que se utilizan finalmente para implementar un sistema.

2.3. Aplicación Notepad (versión 2)

En esta segunda versión se añadirá una segunda actividad para que el usuario cree y edite notas. También se permitirá eliminar notas a través del menú contextual. La actividad nueva asumirá la responsabilidad de crear notas nuevas recopilando la entrada del usuario y agrupando la información que devolverá a través de un objeto `Bundle` que se facilita a través del objeto `Intent` que lanza esta segunda actividad.

Con esta segunda versión practicaremos:

- La construcción de actividades nuevas.
- La invocación a otra actividad de forma asíncrona utilizando `startActivityForResult`.
- El paso de datos entre actividades con objetos `Bundle`.
- La creación de un layout más avanzado.
- La creación de un menú contextual.

Paso 1 Abre el proyecto `Notepadv2` y comprueba lo siguiente:

- Hay nuevas cadenas en el fichero `strings.xml`.
- Se han definido nuevas constantes en la clase `Notepadv2`, incluido un atributo `mNotesCursor` para referenciar al cursor.
- El método `fillData` incluye nuevos comentarios y utiliza el atributo `mNotesCursor`.
- Hay varios métodos redefinidos (`onCreateContextMenu`, `onContextItemSelected`, y `onActivityResult`) que se irán completando en los siguientes pasos.

Paso 2 Los menús contextuales se utilizan para realizar acciones sobre elementos específicos de la interfaz de usuario. Cuando se asocia un objeto `View` a un menú contextual, el menú se despliega después de realizar un clic largo sobre esa componente de la interfaz.

Crearemos un menú contextual que permita borrar y editar notas de forma individual:

1. Asociaremos el menú contextual al objeto `ListView` como última acción dentro del método `onCreate`.

   ```
   registerForContextMenu(mList);
   ```

2. Rellenaremos el método `onCreateContextMenu`:

   ```
   public void onCreateContextMenu(ContextMenu menu, View v,
                                   ContextMenu.ContextMenuInfo menuInfo) {
       super.onCreateContextMenu(menu, v, menuInfo);
       menu.add(Menu.NONE, DELETE_ID, Menu.NONE, R.string.menu_delete);
       menu.add(Menu.NONE, EDIT_ID, Menu.NONE, R.string.menu_edit);
   }
   ```

Paso 3 Ya que hemos asociado el objeto de tipo `ListView` con un menú contextual, deberemos definir el comportamiento asociado al mismo cuando se selecciona una opción de este menú. De momento, gestionaremos la eliminación de una nota de la siguiente forma:

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    switch(item.getItemId()) {
       case DELETE_ID:
           AdapterView.AdapterContextMenuInfo info =
              (AdapterView.AdapterContextMenuInfo) item.getMenuInfo();
           mDbHelper.deleteNote(info.id);
           fillData();
           return true;
    }
    return super.onContextItemSelected(item);
}
```

El atributo `id` del objeto `info` de tipo `AdapterContextMenuInfo` nos permite conocer el identificador interno del elemento seleccionado en en la lista. Pasaremos este `id` al método `deleteNote` de nuestro `NotesDbAdapter` para que borre la nota.

Paso 4 Para crear notas vamos a abrir una actividad nueva que facilite la edición de la nota al usuario. La apertura de otras actividades la realizaremos a través de un objeto de tipo `Intent`. Añadiremos en el cuerpo del método `createNote` las siguientes instrucciones:

```
Intent i = new Intent(this, NoteEdit.class);
startActivityForResult(i, ACTIVITY_CREATE);
```

Cómo se puede ver, el objeto `Intent` abre una actividad `NoteEdit` (que se definirá posteriormente) utilizando la actividad `Notepadv2` como contexto.

El método `startActivityForResult` lanza el `Intent`, pero además se asegura de que se llame al método `onActivityResult` cuando finalice la actividad `NoteEdit`.

Si no quisiésemos que notificaciones al finalizar una actividad, podríamos haber utilizado el método `startActivity`.

Paso 5 También abriremos la actividad `NoteEdit` cuando el usuario seleccione la opción de editar en el menú contextual. Para ello, realizaremos lo siguiente:

1. Implementaremos el comportamiento asociado en el método `onContextItemSelected`:

   ```
   case EDIT_ID:
      info = (AdapterView.AdapterContextMenuInfo) item.getMenuInfo();
      editNote(info.position, info.id);
      return true;
   ```

   Como se puede observar, estamos interesados en recuperar la posición de la lista sobre la que hizo clic el usuario (`info.position`), y el identificador interno asociado (`info.id`).

2. Definiremos el método `editNote` de la siguiente forma:

   ```
   private void editNote(int position, long id) {
       Cursor c = mNotesCursor;
       c.moveToPosition(position);
       Intent i = new Intent(this, NoteEdit.class);
       i.putExtra(NotesDbAdapter.KEY_ROWID, id);
       i.putExtra(NotesDbAdapter.KEY_TITLE, c.getString(
               c.getColumnIndexOrThrow(NotesDbAdapter.KEY_TITLE)));
       i.putExtra(NotesDbAdapter.KEY_BODY, c.getString(
               c.getColumnIndexOrThrow(NotesDbAdapter.KEY_BODY)));
       startActivityForResult(i, ACTIVITY_EDIT);
   }
   ```

   El método `putExtra` permite añadir elementos extra al objeto `Bundle` que se utiliza para pasar información en la apertura de la nueva actividad a través del objeto `Intent`.

   Nosotros queremos pasar el titulo, cuerpo e identificador de la fila de la tabla que

almacena las notas en la base de datos. Esta información se obtiene del objeto de tipo cursor una vez que nos hemos movido a la posición seleccionada por el usuario con el método `moveToPosition`.

Paso 6 Los métodos `createNote` y `editNote` abren la actividad `NoteEdit` de forma asíncrona. Por tanto, debemos gestionar los callback a través del método `onActivityResult` rellenando el cuerpo del método de la siguiente forma:

```
super.onActivityResult(requestCode, resultCode, intent);
Bundle extras = intent.getExtras();

switch(requestCode) {
case ACTIVITY_CREATE:
    String title = extras.getString(NotesDbAdapter.KEY_TITLE);
    String body = extras.getString(NotesDbAdapter.KEY_BODY);
    mDbHelper.createNote(title, body);
    fillData();
    break;
case ACTIVITY_EDIT:
    Long mRowId = extras.getLong(NotesDbAdapter.KEY_ROWID);
    if (mRowId != null) {
        String editTitle = extras.getString(NotesDbAdapter.KEY_TITLE);
        String editBody = extras.getString(NotesDbAdapter.KEY_BODY);
        mDbHelper.updateNote(mRowId, editTitle, editBody);
    }
    fillData();
    break;
}
```

Algunas notas sobre esta implementación:

- El parámetro `requestCode` identifica el objeto `Intent` con el que se lanzó la actividad `NoteEdit` (`ACTIVITY_CREATE` o `ACTIVITY_EDIT`).
- El parámetro `resultCode` informa sobre la finalización de la actividad (0 si todo fue bien, distinto a 0 en otros casos).
- El parámetro `intent` es creado por la actividad `NoteEdit` para devolver resultados. Los campos extras se utilizan para pasar información.
- En el caso de un callback de `NoteEdit` para editar los campos de una nota nueva, debemos crear la nota en la base de datos.
- En el caso de un callback de `NoteEdit` para modificar los campos de una nota existente, debemos actualizar la nota en la base de datos.

- Como último paso se invoca al método `fillData` para actualizar la lista que se muestra al usuario.

Paso 7 Vas a crear ahora la actividad `NoteEdit` como un `BlankActivity` tal como aprendiste en la práctica 1. Una vez vez creada, definiremos el método `onCreate` de la siguiente forma:

1. Estableceremos como layout el fichero `note_edit.xml`:

   ```
   setContentView(R.layout.note_edit);
   ```

2. Cambiaremos el título de la actividad a la cadena "Edit Note":

   ```
   setTitle(R.string.edit_note);
   ```

3. Recuperaremos los componentes `EditText` y `Button` que necesitamos para actualizar los atributos `mTitleText` y `mBodyText` (que se deberían definir dentro de la clase `NoteEdit`):

   ```
   mTitleText = (EditText) findViewById(R.id.title);
   mBodyText = (EditText) findViewById(R.id.body);
   Button confirmButton = (Button) findViewById(R.id.confirm);
   ```

4. Añadiremos el código para inicializar `title`, `body` y `mRowId` (se debería definir como atributo privado dentro de la clase `NoteEdit`). Estos valores se pasan en los extras del objeto `Bundle` asociado al `Intent` que ha abierto la actividad.

   ```
   mRowId = null;
   Bundle extras = getIntent().getExtras();
   if (extras != null) {
     String title = extras.getString(NotesDbAdapter.KEY_TITLE);
     String body = extras.getString(NotesDbAdapter.KEY_BODY);
     mRowId = extras.getLong(NotesDbAdapter.KEY_ROWID);

     if (title != null) {
         mTitleText.setText(title);
     }
     if (body != null) {
         mBodyText.setText(body);
     }
   }
   ```

5. Crearemos un objeto que escuche los eventos del botón de confirmación:

   ```
   confirmButton.setOnClickListener(new View.OnClickListener() {
       public void onClick(View view) {
   ```

```
    }
});
```

En el siguiente paso se define el método `onClick` con el comportamiento asociado a la pulsación de este botón.

Paso 8 Vamos a definir el método `onClick` para poder cerrar la actividad `NoteEdit` y pasar la información editada a la actividad `Notepadv2` que abrió `NoteEdit` de la siguiente forma:

1. Crearemos un objeto `Bundle` y pondremos el título y el cuerpo de según las claves definidas en la clase `NotesDbAdapter`:

   ```
   Bundle bundle = new Bundle();
   bundle.putString(NotesDbAdapter.KEY_TITLE, mTitleText.getText().toString());
   bundle.putString(NotesDbAdapter.KEY_BODY, mBodyText.getText().toString());
   if (mRowId != null) {
       bundle.putLong(NotesDbAdapter.KEY_ROWID, mRowId);
   }
   ```

2. Crearemos un objeto `Intent`, integraremos la información recopilada en el objeto `Bundle`, y finalizaremos la actividad:

   ```
   Intent mIntent = new Intent();
   mIntent.putExtras(bundle);
   setResult(RESULT_OK, mIntent);
   finish();
   ```

Ahora ya puedes ejecutar el proyecto.

2.4. Aplicación Notepad (versión 3)

En esta tercera versión aprenderemos a manejar los eventos en el ciclo de vida de las actividades. Utilizaremos estos eventos para almacenar y recuperar los datos de la aplicación en el momento oportuno. También veremos cómo mantener el estado de la aplicación. Esta versión será la que tomaremos como punto de partida para la práctica 4 y el trabajo de la asignatura.

Paso 1 Abre el proyecto `Notepadv3`. El punto de partida de este proyecto es la versión final de `Notepadv2`. Esta aplicación tiene algunos problemas: si se pulsa el botón de retroceso durante la edición, se provoca un error; si ocurre algo inesperado durante la edición, se pierde el texto introducido.

Para evitar esto, trasladaremos la mayor parte de la funcionalidad relacionada con la creación y la edición de la nota a la actividad `NoteEdit` de la siguiente forma:

- Eliminaremos el código de `NoteEdit` que recoge el título y el cuerpo del objeto `Bundle`:

  ```
  String title = extras.getString(NotesDbAdapter.KEY_TITLE);
  String body = extras.getString(NotesDbAdapter.KEY_BODY);
  ```

  En su lugar, utilizaremos la clase `DBHelper` para acceder a las notas en la base de datos. La única información que será pasada `NoteEdit`, y solo para la edición, será `mRowId`.

- También eliminaremos la actualización de mTitleText y mBodyText.

  ```
  if (title != null) {
      mTitleText.setText(title);
  }
  if (body != null) {
      mBodyText.setText(body);
  }
  ```

Paso 2 Crea un atributo para guardar una referencia a `NotesDbAdapter` en la clase `NoteEdit`:

```
private NotesDbAdapter mDbHelper;
```

e inicialízalo dentro del método onCreate justo después de la llamada a super:

```
mDbHelper = new NotesDbAdapter(this);
mDbHelper.open();
```

Paso 3 En `NoteEdit` necesitamos chequear el `savedInstanceState` para actualizar el valor de `mRowId` en el caso de que la nota que está editándose contenga un estado almacenado en el `Bundle`, el cual deberíamos recuperar. Esto ocurre cuando nuestra actividad pierde el foco y se reinicia. Para ello se debe reemplazar el código:

```
mRowId = null;
Bundle extras = getIntent().getExtras();
if (extras != null) {
    mRowId = extras.getLong(NotesDbAdapter.KEY_ROWID);
}
```

por este:

```
mRowId = (savedInstanceState == null) ? null :
    (Long) savedInstanceState.getSerializable(NotesDbAdapter.KEY_ROWID);
if (mRowId == null) {
    Bundle extras = getIntent().getExtras();
    mRowId = (extras != null) ? extras.getLong(NotesDbAdapter.KEY_ROWID)
                              : null;
}
```

Paso 4 Ahora tenemos que recuperar el título y el cuerpo en base a `mRowId` con el método `populateFields()`, el cual deberá ser invocado antes de `confirmButton.setOnClickListener()`.

El método `populateFields` se definirá de la siguiente forma dentro de la clase `NoteEdit`:

```
private void populateFields() {
    if (mRowId != null) {
        Cursor note = mDbHelper.fetchNote(mRowId);
        startManagingCursor(note);
        mTitleText.setText(note.getString(
            note.getColumnIndexOrThrow(NotesDbAdapter.KEY_TITLE)));
        mBodyText.setText(note.getString(
                note.getColumnIndexOrThrow(NotesDbAdapter.KEY_BODY)));
    }
}
```

Se utiliza el método `NotesDbAdapter.fetchNote()` para recuperar la nota a editar. A continuación se invoca a `startManagingCursor` para gestionar el ciclo de vida del cursos, y reservar/liberar recursos de forma automática para este cursor. Finalmente, se actualiza `mTitleText` y `mBodyText` con los valores recuperados de la base de datos.

Paso 5 Evitaremos también la creación del objeto `Bundle` dentro del método `onClick` ya que no se necesita enviar información extra a la actividad llamante. Tampoco hace falta devolver un objeto `Intent`. Así que la versión simplificada será:

```
public void onClick(View view) {
    setResult(RESULT_OK);
    finish();
}
```

En los siguientes pasos nos preocuparemos de guardar las actualizaciones o las notas nuevas utilizando los métodos del ciclo de vida de la actividad.

Paso 6 Dentro de la clase `NoteEdit` redefiniremos los métodos `onSaveInstanceState`, `onPause` y `onResume`. Junto con el método `onCreate` estos métodos definen la acciones a realizar dentro del ciclo de vida de una actividad cuando se reciben eventos.

Se llama a `onSaveInstanceState` cada vez que se para la actividad y puede ser eliminada antes de ser retomada (`resumed`). Esto significa que deberíamos guardar cualquier estado necesario para reinicializar la actividad cuando se reinicie (`restarted`). Es la contrapartida al método `onCreate`. De hecho, el parámetro `savedInstanceState` es el mismo objeto `Bundle` que se devuelve dentro del método `onSaveInstanceState`.

Los métodos `onPause` y `onResume` son también complementarios. Se llama a `onPause` siempre que se finaliza la actividad, incluso aunque nosotros hayamos sido los responsables (con una invocación a `finish`). Usaremos este método para guardar el estado actual en la base de datos. También es conveniente liberar recursos cuando se invoca a este método. Por otro lado, el método `onResume` llamará a `populateFields` para volver a leer la nota de la base de datos y actualizar la interfaz.

El código de estos métodos será el siguiente:

```
@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    saveState();
    outState.putSerializable(NotesDbAdapter.KEY_ROWID, mRowId);
}

@Override
protected void onPause() {
    super.onPause();
    saveState();
}

@Override
protected void onResume() {
    super.onResume();
    populateFields();
}
```

Se puede observar que el método `saveState` se invoca tanto desde `onSaveInstanceState` como desde `onPause` para asegurarnos de que los datos se guardan. Se hace esto porque no hay garantía de que se invoque a `onSaveInstanceState`, y porque cuando se llama, se realiza antes de la invocación a `onPause`.

Paso 7 El método `saveState` se definirá de la siguiente forma:

```
    private void saveState() {
       String title = mTitleText.getText().toString();
       String body = mBodyText.getText().toString();

       if (mRowId == null) {
           long id = mDbHelper.createNote(title, body);
           if (id > 0) {
               mRowId = id;
           }
       } else {
           mDbHelper.updateNote(mRowId, title, body);
       }
   }
```

Observa que se captura el valor devuelto por el método `createNote` y se almacena en `mRowId` por si posteriormente se actualiza la nota antes de finalizar la actividad. El método `saveState` puede ser invocado porque se ha perdido el foco en la actividad `NoteEdit` durante la creación de una nota.

Paso 8 Ahora eliminaremos el código sobrante de la clase `Notepadv3` en su método `onActivityResult`. Ya no es necesario solicitar la inserción o actualización de filas de la base de datos desde `Notepadv3`. Solo se necesita actualizar el objeto `ListView`.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent intent) {
    super.onActivityResult(requestCode, resultCode, intent);
    fillData();
}
```

Paso 9 Finalmente, eliminaremos también en el método `editNote` de la clase `Notepadv3` las líneas que mueven el cursor a una posición:

```
    Cursor c = mNotesCursor;
    c.moveToPosition(position);
```

Es suficiente con conocer el identificador interno de la fila que se pasa como parámetro en el método `editNote`. Así que podríamos dejar un solo parámetro en el método y eliminar el paso del título y el cuerpo en el objeto `Bundle` eliminando las siguientes líneas:

```
i.putExtra(NotesDbAdapter.KEY_TITLE, c.getString(
                c.getColumnIndex(NotesDbAdapter.KEY_TITLE)));
i.putExtra(NotesDbAdapter.KEY_BODY, c.getString(
                c.getColumnIndex(NotesDbAdapter.KEY_BODY)));
```

También podemos eliminar ahora el atributo `mNotesCursor` de la clase y volver a utilizar una variable local en el método `fillData`:

```
Cursor notesCursor = mDbHelper.fetchAllNotes();
```

Ahora ya puedes ejecutar el proyecto.

3. Entrega de la práctica

A través de una tarea accesible en Moodle, subiréis un documento con formato PDF donde se incluirán las imágenes correspondientes a las capturas de los diagramas de clases y diagramas de secuencia solicitados en la sección 2.2. Cada diagrama irá acompañado de un título descriptivo en dicho documento.

La fecha límite para subir este documento a Moodle será el día anterior a la siguiente sesión de prácticas de cada equipo. Durante la sesión de la práctica 4 presentaréis al profesor el resultado de esta práctica. Recordad que para superar la asignatura hay que presentar todas las prácticas.

En la práctica 4 retomaremos la aplicación de notas en su versión 3 solucionada y actualizaremos los modelos estáticos y dinámicos correspondientes. Por tanto, conviene haber realizado los cambios propuestos en las secciones 2.3 y 2.4.

Referencias

[1] Modelio. Modelio: the open source modeling environment. `http://www.modelio.org/`.