Notepad – Exercício 01

Nesse exemplo você vai construir uma lista simples de notas que permite ao usuário adicionar novas notas mas não editá-las. O exercício demonstrará:

- A parte básica sobre ListActivities e criação e gerenciamento de opções de menu;
- Como usar o banco de dados SQLite para guardar as notas;
- Como fazer o bind dos dados de um cursor de banco de dados para uma ListView usando um SimpleCursosAdapter e;
- A parte básica de layout de tela incluindo como inserir uma ListView, como adicionar itens para o menu de atividade e como essas atividades gerenciam as seleções do menu.

Passo 1

Para que você possa rodar o notepad e, consequentemente, seguir esse exercício, você tem que baixar o projeto completo do Notepadv1 da pasta, arquivos, na sala da turma. Após isso, abra o projeto no Android Studio.

Passo 2

Vamos analisar a classe NotesDbAdapter:

```
/*
* Copyright (C) 2008 Google Inc.
*
* Licensed under the Apache License, Version 2.0 (the "License"); you may not
* use this file except in compliance with the License. You may obtain a copy of
* the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS, WITHOUT
* WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
* License for the specific language governing permissions and limitations under
* the License.
*/

package br.com.cd6.notepadv1;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
```

```
import android.util.Log;


/**
* Simple notes database access helper class. Defines the basic CRUD operations
* for the notepad example, and gives the ability to list all notes as well as
* retrieve or modify a specific note.
*
* This has been improved from the first version of this tutorial through the
* addition of better error handling and also using returning a Cursor instead
* of using a collection of inner classes (which is less scalable and not
* recommended).
*/
public class NotesDbAdapter {


  /* Variáveis */
  public static final String KEY_TITLE = "title";
  public static final String KEY_BODY = "body";
  public static final String KEY_ROWID = "_id";


  private static final String TAG = "NotesDbAdapter";
  private DatabaseHelper mDbHelper;
  private SQLiteDatabase mDb;


  /**
```

```
 * Database creation sql statement
 */
private static final String DATABASE_CREATE =
        "create table notes (_id integer primary key autoincrement, "
                + "title text not null, body text not null);";

private static final String DATABASE_NAME = "data.db";
private static final String DATABASE_TABLE = "notes";
private static final int DATABASE_VERSION = 1;

private final Context mCtx;

private static class DatabaseHelper extends SQLiteOpenHelper {

    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {

        db.execSQL(DATABASE_CREATE);
    }
```

```
        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            Log.w(TAG, "Fazendo o Update do banco de dados da versão " + oldVersion
+
                    " para " + newVersion + ", ação essa que destruirá os dados antigos");
            db.execSQL("DROP TABLE IF EXISTS notes");
            onCreate(db);
        }
    }


    /**
     * Construtor - recebe o contexto para permitir que o banco de dados seja
     * criado/aberto
     * @param ctx O contexto no qual trabalhar
     */
    public NotesDbAdapter(Context ctx) {
        this.mCtx = ctx;
    }


    /**
     * Abre o banco de dados das notas. Se não pode ser aberto, cria o banco.
     * se não pode ser criado, emita uma exceção com sinal de falha
     *
     * @return this (Referencia a si mesmo, permitindo que seja canalizado para
     *         uma chamada inicial)
```

```
     * @throws SQLException se o banco de dados não pode ser aberto ou criado
     */
    public NotesDbAdapter open() throws SQLException {
        mDbHelper = new DatabaseHelper(mCtx);
        mDb = mDbHelper.getWritableDatabase();
        return this;
    }

    public void close() {
        mDbHelper.close();
    }


    /**
     * Cria uma nova nota usando o título e corpo provido. A nota criada com sucesso
     * retorna um novo rowId. Caso contrário, retorna -1 como sinal de falha      *
     * @param title O título da nota
     * @param body O corpo da nota
     * @return Retorna rowId ou -1 se houve falha
     */
    public long createNote(String title, String body) {
        ContentValues initialValues = new ContentValues();
        initialValues.put(KEY_TITLE, title);
        initialValues.put(KEY_BODY, body);
```

```
    return mDb.insert(DATABASE_TABLE, null, initialValues);
}

/**
 * Exclui uma nota com um dado rowId
 *
 * @param rowId ID da nota a ser excluída
 * @return Retorna true se excluído e false caso não tenha sido excluído
 */
public boolean deleteNote(long rowId) {

    return mDb.delete(DATABASE_TABLE, KEY_ROWID + "=" + rowId, null) > 0;
}

/**
 * Retorna um Cursor da lista de todas as notas do banco de dados
 *
 * @return Cursor de todas as notas
 */
public Cursor fetchAllNotes() {

    return mDb.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_TITLE,
            KEY_BODY}, null, null, null, null, null);
```

```
}

/**
 * Retorna um cursor posicionado na nota que bate com o rowId fornecido
 *
 * @param rowId ID da nota a ser recuperada
 * @return Cursor posicionado na nota selecionada, se encontrada
 * @throws SQLException Se a nota não pôde ser encontrada/recuperada
 */
public Cursor fetchNote(long rowId) throws SQLException {

    Cursor mCursor =

            mDb.query(true, DATABASE_TABLE, new String[] {KEY_ROWID,
                    KEY_TITLE, KEY_BODY}, KEY_ROWID + "=" + rowId, null,
                null, null, null, null);
    if (mCursor != null) {
        mCursor.moveToFirst();
    }
    return mCursor;

}

/**
```

```
    * Faz o Update da nota usando os detalhes providos. A nota a ser atualizada é
    * especificada usando a rowId e é alterada para usar os valores de título e corpo
    * enviados ao método
    *
    * @param rowId Id da nota a ser atualizada
    * @param title de título a ser atualizado na nota
    * @param body Valor a ser usado como corpo da nota
     * @return Retorna true se a nota foi atualizada com sucesso e false em caso contrário.
    */
   public boolean updateNote(long rowId, String title, String body) {
       ContentValues args = new ContentValues();
       args.put(KEY_TITLE, title);
       args.put(KEY_BODY, body);

       return mDb.update(DATABASE_TABLE, args, KEY_ROWID + "=" + rowId, null) > 0;
   }
}
```

Essa classe é provida para encapsular os acessos de dados ao banco de dados SQLite que vai guardar as notas e nos permitir atualizá-las.

No início da classe existe o comentário da Google e também as definições das constantes que serão usadas na aplicação para pesquisar os dados a partir dos campos apropriados no banco de dados. Existe também uma string de criação de banco de dados que será usado para criar um esquema de banco de dados caso ainda não tenha sido criado.

Nosso banco de dados terá o nome “data.db” e terá uma tabela única chamada “notes” que, internamente, têm três campos: _id, title e body. O _id é nomeado com a convenção de underscore no início dele. O _id usualmente tem de ser especificado quando pesquisando ou fazendo o update do banco de dados. Os outros campos são campos de texto que vão guardar os dados.

O construtor de NotesDbAdapter recebe um Context que o permite se comunicar com aspectos do sistema operacional Android. É muito comum classes necessitarem de manter contato com o sistema Android de alguma maneira. A classe Activity implementa a classe de Context para que usualmente você tenha apenas de passar a palavra this a partir de sua atividade, quando precisar do Contexto.

O método open() chama uma instância de DatabaseHelper que é sua implementação local da classe SQLiteOpenHelper. Ele chama o método getWritableDatabase que manuseia a criação/abertura do banco para nós.

O método close() apenas fecha o banco de dados, liberando recursos relativos à conexão.

createNote() recebe strings para o título e corpo da nova nota e então cria a nota no banco de dados. Assumindo que uma nova nota foi criada com sucesso, o método também retorna o valor “_id” para a nota recém-criada.

deleteNote() recebe um _id relacionado a uma rowId e a exclui do banco de dados.

fetchAllNotes() emite uma pesquisa para retornar um Cursor das notas no banco de dados. A chamada para query() vale a pena ser examinada e entendida. O primeiro argumento passado é o nome da tabela no banco de dados na qual pesquisar (nesse caso a DATABASE_TABLE é “notes”). O próximo argumento é a lista de campos que queremos que seja retornado e nesse caso queremos os campos _id, title e body que então são especificados para uma array de strings. Os argumentos remanescentes são, em ordem: selection, selectionArgs, groupBy, having e order by. Passado null em todos esses últimos argumentos faz que sejam selecionados todos os dados sem necessidades de agrupamento e sem ordenação. Será mantida a ordem na qual as notas foram inseridas no banco de dados.

Nota: Um cursor é retornado em vez de uma coleção de linhas. Isso permite ao Android usar os recursos mais eficientemente – em vez de retornar dados e mais dados direto na memória o cursor vai buscar e retornar os dados à medida que for sendo necessário, o que é muito mais eficiente para tabelas com muitas linhas.

fetchNote() é similar a fetchAllNotes() mas ele apenas busca uma nota com uma rowId que especificamos. Ele usa uma versão diferente do método query() da classe SQLiteDatabase. O primeiro argumento (indicado como true) indica se nós estamos interessados em um resultado distinto. O argumento selection (que é o quarto na listagem) foi especificado para fazer a pesquisa apenas pela row cujo _id foi o passado.

E finalmente o updateNote() que recebe a rowId, e corpo e usa uma instância de ContentValues para fazer o update da nota de uma dada rowId.

Passo 3

Abra o arquivo notepad_list.xml que se encontra em res/layout e veja como ele foi criado.

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"

   android:layout_width="match_parent"

   android:layout_height="match_parent"

   android:orientation="vertical">



   <include layout="@layout/toolbar" />



</LinearLayout>
```

Esse é um arquivo de definição de layout bastante vazio. Aqui estão algumas coisas que você deveria saber a respeito desse arquivo de layout:

Todos os arquivos de layout do Android devem iniciar-se com a linha de cabeçalho como a que vemos acima.

A próxima definição será na maioria das vezes (mas não em todas as vezes) a definição de um tipo de layout. No caso acima, usamos o LinearLayout.

O namespace XML do Android deve sempre ser definido para um componente de alto nível ou layout no XML para que as tags “android:” possam ser usadas em todo o arquivo.

xmlns:android:"https://schemas.android.com/apk/res/android"

Passo 4

Precisamos criar um layout para guardar nossa lista. Então, que tal adicionar códigos dentro do elemento LinearLayout para que nosso arquivo completo se pareça com o que vai abaixo?

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    android:orientation="vertical" >


    <ListView

        android:id="@android:id/list"

        android:layout_width="wrap_content"

        android:layout_height="wrap_content" />

    <TextView

        android:id="@android:id/empty"

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:text="@string/no_notes" />

</LinearLayout>
```

Alguns comentários a respeito do XML acima:

O símbolo @ perto das strings de id da ListView e da TextView significa que o parser XML deverá fazer o parse e expandir o resto da string id e usar um ID de recursos.

A ListView e TextView podem ser pensadas como duas views alternativas. Apenas uma delas será mostrada de cada vez. ListView será usado quando existem notas a serem mostradas enquanto que a TextView (que tem o valor padrão de "Nenhuma nota cadastrada" definido no arquivo de recursos em res/values/string.xml) será exibido se não existir nenhuma nota a ser mostrada.

Os ids list e empty são providos pela plataforma Android então você deverá prefixar o id com android: (e.g. @android: id/list)

A view com o id empty é usado automaticamente quando o ListAdapter não tem dados para a ListView. O ListAdapter sabe procurar por esse nome por padrão. Alternativamente, você poderá mudar a view vazia usando setEmptyView(View) dentro da ListView.

Dentro do recurso strings.xml (dentro de res/values) adicione uma nova string chamada "no_notes" cujo valor esteja setado para "Nenhuma Nota Cadastrada".

```
<string name="no_notes">Nenhuma Nota Cadastrada</string>
```

Passo 5

Para criar uma lista de notas na ListView, também precisamos definir uma View para cada linha.

Crie um novo arquivo dentro de res/layout chamado notes_row.xml.

Adicione o código seguinte dentro do arquivo:

```
<?xml version="1.0" encoding="utf-8"?>

<TextView

  android:id="@+id/text1"

   xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
/>
```

Nesse caso nós criamos um id chamado text1. O + após o @ na id indica que o id deverá ser automaticamente criado como um recurso se ele não existir efetivamente. Então, nós o definimos como text1 e então o usamos.

Salve o arquivo.

Passo 6

Agora vamos abrir a classe Notepadv1. Nos próximos passos vamos alterar essa classe para fazer dela um listAdapter e mostrar nossas notas e também para permitir que possamos adicionar novas notas.

```
package br.com.cd6.notepadv1;

import android.app.Activity;

import android.os.Bundle;

import android.view.Menu;

import android.view.MenuItem;


public class Notepadv1 extends Activity {

  private int mNoteNumber = 1;

  @Override

  public void onCreate(Bundle savedInstanceState) {

     super.onCreate(savedInstanceState);

  }


  @Override
```

```
    public boolean onCreateOptionsMenu(Menu menu) {
        // TODO Auto-generated method stub
        return super.onCreateOptionsMenu(menu);
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // TODO Auto-generated method stub
        return super.onOptionsItemSelected(item);
    }
}
```

A classe Notepadv1 herda métodos de uma subclasse de Activity chamada ListActivity, que tem funcionalidades extras para acomodar os tipos de coisas que você deseja fazer com a lista. Por exemplo: mostrar um número arbitrário de itens de lista em linhas na tela, mover-se através de itens de lista e permitir que eles sejam selecionados.

Veja que no código da classe Notepadv1 existe uma variável privada chamada mNoteNumber que não foi usada e que usaremos para criar títulos de notas numeradas.

Existem também três métodos definidos: onCreate, onCreateOptionsMenu e onOptionsItemSelected.

onCreate() é chamado quando a atividade é iniciada - é mais ou menos como se fosse o método principal da atividade. Nós usamos ele para configurar recursos e o estado da atividade quando ela está rodando.

onCreateOptionsMenu() é usada para popular o menu para a atividade. Ele é mostrado quando o usuário pressiona o botão do menu e tem uma lista que ele pode selecionar (como "Criar Nota").

onOptionsItemSelected() é a outra metade da equação do menu e é usada para manusear eventos gerados a partir do menu (e.g. quando o usuário seleciona o item "Criar Nota").

Passo 7

Mude a herança de Notepadv1 de Activity para ListActivity.

```
public class Notepadv1 extends ListActivity
```

Nota: Você terá de importar ListActivity para a classe de Notepadv1 usando o Eclipse. Para tal, pressione Control+Shift+O.

Passo 8

Vamos preencher o método onCreate().

Nesse método vamos configurar um título para a atividade (mostrado no topo da tela), usar o layout notepad_list que nós criamos em XML, configurar uma instância de NotesDbAdapter que acessa os dados e popula a lista com as notas disponíveis.

Dentro do método onCreate(), chame super.onCreate() com o parâmetro savedInstanceState que é passado para o método.

Chame setContentView() e passe R.layout.notepad_list.

No topo da classe crie um novo campo privado chamado mDbHelper do tipo NotesDbAdapter.

De volta ao método onCreate, construa uma nova instância de NotesDbAdapter e assinale-a para o campo privado mDbHelper (passando this para o construtor de DBHelper).

Finalmente, chame o novo método fillData() que vai buscar os dados e populá-los na ListView usando o helper – que nós ainda não definimos.

O onCreate() deverá se parecer com o código abaixo:

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.notepad_list);
  mDbHelper = new NotesDbAdapter(this);
  mDbHelper.open();
  fillData();
}
```

Não se esqueça de definir o campo privado mDbHelper.

```
private NotesDbAdapter mDbHelper;
```

Passo 9

Vamos agora preencher o método onCreateOptionsMenu().

Nós agora vamos criar um botão de adição de item que poderá ser acessado pressionando o botão de menu no dispositivo. Vamos especificar também que ele ocupa a primeira posição no menu.

Dentro do recurso strings.xml (dentro de res/values) adicione uma nova string chamada “menu_insert” cujo valor esteja setado para “Adicionar Nota”.

```
<string name="menu_insert">Adicionar Nota</string>
```

Crie uma constante de posição de menu no topo da classe:

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

Dentro do método onCreateOptionsMenu() mude a chamada para super para que nós possamos capturar o booleano retornado como resultado. Nós retornaremos esse valor ao final do método.

Por fim, adicione o item de menu com menu.add.

O código completo do método deverá se parecer com esse:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
  boolean result = super.onCreateOptionsMenu(menu);
  getMenuInflater();
  menu.add(0, INSERT_ID, 0, R.string.menu_insert);
  return result;
}
```

O argumento passado para menu.add indica: um identificador de grupo para esse menu (nenhum, no caso), um ID único (definido acima), a ordem do item (0 indica nenhuma preferência) e o recurso da string a ser usada no item).

Passo 10

Por fim, última modificação de método já existente: onOptionsItemSelected().

Esse método será responsável por receber o comando de inserir menu. Quando o método for chamado, ele será chamado com o item.getID setado para o INSERT_ID (que é a constante que usamos para identificar o item de menu). Nós podemos detectar que botão foi pressionado e tomar as ações apropriadas:,

O método super.onOptionsItemSelected(item) vai para o final de seu método - queremos que ele capture os eventos em primeiro lugar.

Escrevemos um switch para item.getItemId().

No caso de ser INSERT_ID, o método createNote() será chamado e retornará true, já que nós manuseamos este evento e não queremos que ele se propague pelo sistema.

Retorna o resultado para o método onOptionsItemSelected() da classe pai (a classe pai, no caso, que é a ListActivity).

O código completo deverá ser como mostrado abaixo:

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

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

Passo 11

Vamos criar o método createNote()?

Sendo essa a primeira versão da nossa aplicação, o createNote() não será muito útil. Nós simplesmente criaremos uma nova nota com um título baseado na contagem de notas já existentes (“Nota 1”, “Nota 2”) e com um corpo vazio. Neste momento, não há nenhuma maneira de se editar os conteúdos de uma nota e por agora nós fazemos com que o conteúdo seja criado com valores padrão.

Construa o nome usando “Nota” e o contador que nós definimos na chasse:

String noteName = “Nota ” + mNoteNumber++

Chame mDbHelper.createNome() usando noteName como título e "" como corpo da nota.

Chame fillData() para popular a lista de notas (ineficiente mas simples) - nós criaremos esse método posteriormente.

O código completo deve ser como mostrado abaixo:

```
private void createNote() {

  String noteName = "Nota " + mNoteNumber++;

  mDbHelper.createNote(noteName, "");

  fillData();

}
```

Passo 12

Vamos, enfim, definir o método fillData().

Esse método usa o SimpleCursorAdapter que toma um Cursor de banco de dados e faz o bind para os campos providos no layout. Esses campos definem os elementos de linha de nossa lista (nesse caso usamos o text1 do layout notes_row.xml), para que se permita facilmente popular a lista com entradas do nosso banco de dados.

Para fazer isso nós devemos prover um mapeamento do campo no Cursor retornado para nosso TextView text1 que deve ser feito definindo suas arrays: a primeira é uma array de strings com a lista de colunas para se mapear a partir dele e, a segunda, um array de inteiros contendo referências para as views onde os dados terão o bind feito. Ou seja, diremos de onde retiramos os dados e onde vamos inserir para ser mostrado na tela.

O código deverá ser:

```
private void fillData() {

  // Capture todas as notas do BD e crie uma lista de itens

  Cursor c = mDbHelper.fetchAllNotes();

  startManagingCursor(c);

  String[] from = new String[] { NotesDbAdapter.KEY_TITLE };

  int[] to = new int[] { R.id.text1 };

   // Agora crie um array adapter e configure-o para ser
mostrado usando nossa linha

  SimpleCursorAdapter notes =

          new SimpleCursorAdapter(this, R.layout.notes_row, c,
from, to);

  setListAdapter(notes);

}
```

O que nós fizemos acima é:

Após obter o Cursor a partir de mDbFetcher.fetchAllNotes(), nós usamos o método da atividade chamado startManagingCursor() que permite ao Android tomar conta do ciclo de vida do Cursor em vez de nós mesmos ter de nos preocupar com ele (nós chegaremos ao ciclo de vida no exercício 3).

Então nós criamos uma array de strings na qual declaramos as colunas que queremos (nesse caso, apenas o título) e um array de inteiros que definirá as Views às quais queremos fazer o bind das colunas.

Por fim, fizemos a instanciação de SimpleCursorAdapter. Como muitas classes do Android, o SimpleCursorAdapter precisa de um Context para que funcione, então nós passamos “this” para o Contexto (isso funciona já que as subclasses de Activity implementam o Context). Nós passamos a View notes_row que criamos como o receptor dos dados, o Cursor que acabamos de criar e então nossos arrays.

No futuro, lembre-se de que o mapeamento entre os recursos “de onde” e “para onde” é feito usando a ordenação específica das duas arrays. Se nós tivéssemos mais colunas que quiséssemos fazer o bind e mais Views a fazer o bind, nós teríamos de especificar em que ordem. O exemplo seria chamar assim: {NotesDbAdapter.KEY_TITLE, NotesDbAdapter.KEY_BODY} e {R.id.text1, R.id.text2} para fazer o bind de dois campos dentro de uma row. É dessa maneira que conseguimos fazer o bind de múltiplos campos em uma única linha (e obter, assim, uma linha customizada de layout).

Se tudo correr bem, o resultado que você terá deverá ser algo como o mostrado no aplicativo.

Até a próxima!