

チュートリアル：ノートパッドエクササイズ1

本エクササイズではシンプルなノートリストを作ります。このノートリストを使ってユーザは新規ノートを追加することができますが、ノートの編集を行うことはできません。本エクササイズでは以下の事項をデモンストレートします。

- *ListActivities* の基本、メニューオプションの作成・操作の基本
- SQLite データベース内のノートへのアクセス/保存方法
- *ArrayAdapter* (*ListView* にバインドするためのもっともシンプルな方法の一つ) を用いてデータを *ListView* にバインドする方法
- リストビューの配置の仕方やアクティビティメニューへのアイテムの追加のやり方、アクティビティがアクティビティメニューアイテムをどのように扱うか、といったスクリーンレイアウトの基本

[[エクササイズ1](#)] [[エクササイズ2?](#)] [[エクササイズ3?](#)] [[追加事項?](#)]

Step 1

Notepadv1 プロジェクトを開きます。

Notepadv1 は開始点として提供されるプロジェクトです - it takes care of some of the boilerplate work that you have already seen if you followed the Hello Android tutorial (すでに[Hello Android\(もしもしアンドロイド\)](#)のチュートリアルで見ているかもしれませんが、共通する説明があります。)

2. browse ボタンを押し、あなたが本エクササイズをコピーしたフォルダを選択し、その中からNotepadv1を選んでOKを押ししてください。
 - a. Package Explorer で右クリックし、New->Project... を選択してください。
 - b. Android/Android Project を選択し、Next > を押ししてください。
 - c. New Android Project のダイアログで、Create project from existing source(既存のソースからプロジェクトを作成)を選択してください。
 - d. browse ボタンを押し、あなたが本エクササイズをコピーしたフォルダを選択し、その中からNotepadv1を選んでOKを押ししてください。
 - e. Project name: 欄に Notepadv1、Location: 欄に選択したパスが表示されることを確認してください。
 - f. Finish を押します。
 - g. 本エクササイズのプロジェクトが Eclipse の Package Explorer に開かれ準備が完了します。
 - h. もし AndroidManifest.xml にエラー表示が出ていたり、Android の zipファイルに関連する問題が表示された場合は、プロジェクト

Step 2

データへのアクセスと更新について

このエクササイズにおいては、単にSQLiteデータベースを直接使ってデータを保存しますが、実際のアプリケーションでは正当なContentProvider も興味があれば、content provider やデータの保存/検索/表示などに関する情報をいろいろと見つけることが出来るでしょう。

DBHelper クラスを見てみてください - このクラスは、我々が作成するノートのデータを保持するとともにその更新も可能にする、SQLiteへのデータアクセスをカプセル化するために提供されています。

典型的にはContentProviderを用いることでこのクラスを実装します。そして実際、SDKには行っている完全なNotepadアプリケーションはそのようなContentProviderを実装しています。しかし、我々がこれからするように、単にSQLiteデータベースを直接利用しなくてもいい理由はありません、このクラスについて注意すべき重要なことは、それがSQLiteデータベースのデータの保存、参照、更新の詳細の面倒をみてくれるということです。全ての行を取得するメソッド、行IDに基づいて行を取得するメソッド、新しい行を作成するメソッド、既存の行を削除するメソッド、既存の行を更新するメソッドがあります。もしあなた自身のアプリケーションでSQLiteデータベースを利用する方法に関する入門を希望なら、このクラスをしてみるか、より良くは、ContentProviderの利用例として SDK の samples/ ディレクトリに入っている完全なNotepadアプリケーションを見てみてください。

Step 3

Layout と activity とについて

ほとんどのActivityはそれに関連付けられたレイアウトを持っています。レイアウトはユーザに対してそのactivityの「顔」となります。しかし、フルスクリーンレイアウトはActivityにとっての唯一の選択肢ではありません。フローティングレイアウト(たとえばダイアログと

Open the notepad_list.xml file in res/layout and take a look at it:

This is a layout definition file with a default starting point in it, we have provided this as a convenience to get you going quickly.

1. All Android layout files must start with the XML header line: `<?xml version="1.0" encoding="utf-8"?>`.
2. Also, the next definition will often (but not always) be a layout definition of some kind, in this case a `LinearLayout`.
3. Note also that the xml namespace of Android should always be defined in the top level component or layout in the XML.

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

Step 4

We need to create the layout to hold our list. Add code inside of the LinearLayout tag so the whole file looks like this: (you may have to hit the Source tab in order to edit the XML file)

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

```
</LinearLayout>
```

1. The ListView and TextView can be thought as two alternative views, only one of which will be displayed at once. ListV
 2. The @ in the id strings of the ListView and TextView means that the XML parser should parse and expand the rest of th
 3. And, the android:list and android:empty are IDs that are already provided for us by the Android platform, empty is us
- More broadly, the android.R class is a set of predefined resources provided for you by the platform, while your proje

Step 5

Resources and the R class

The folders under res/ in the Eclipse project are special. There is a specific structure to the folders and files under this folder.

In particular, resources defined in these folders and files will have corresponding entries in the R class allowing them to be easily accessed and used from your application. Furthermore, they will be bundled and deployed as part of the application.

To make a list view, we also need to define a view for each row in the list:

1. Create a new file under res/layout called notes_row.xml.
2. Add the following contents (note: again the xml header is used, and the first node defines the Android xml namespace)

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

3. This is the view that will be used for each notes title row — it has only one text field in it.
4. In this case we create a new id called text1. The + after the @ in the id string indicates that the id should be auto
5. After saving this file, open the R.java class in the project and look at it, you should see new definitions for notes

Step 6

Next, open the Notepadv1 class in the source. We are going to alter this class to become a list adapter and display our notes, and also allow us to add new notes

Notepadv1 will be a subclass of Activity called a ListActivity, which has extra functionality to accommodate the kinds of things you might want to do with a list, for example: displaying an arbitrary number of list items in rows on the screen, moving through the list items, and allowing them to be selected.

Take a look through the existing code in Notepadv1 class. There are some constant definitions at the top, followed by a private field we will use to create numbered note titles, and some overrides of methods from the superclass.

Step 7

Change the inheritance of Notepadv1 from Activity to ListActivity:

```
public class Notepadv1 extends ListActivity
```

Note: you will have to import ListActivity into the Notepadv1 class using Eclipse, ctrl-shift-O on Windows or Linux, or cmd-shift-O on the Mac (organize imports) will do this for you.

Step 8

There are already three override methods defined: onCreate, onCreateOptionsMenu and onOptionsItemSelected, we need to fill these out:

- * onCreate() is called when the activity is started — it is a little like the "main" method for the activity. We use t
- * onCreateOptionsMenu() is used to populate the menu for the activity. This is shown when the user hits the menu button
- * onOptionsItemSelected() is the other half of the menu equation, it is used to handle events generated from the menu (

Step 9

Fill out the body of the onCreate() method.

Here we will set the title for the activity (shown at the top of the screen), use the notepad_list layout we have created for the activity display contents, set up the DBHelper instance we will use to access notes data, then populate the list with the available note titles:

1. call super() with the icicle parameter passed into our method
2. setContentView to R.layout.notepad_list
3. Create a new private class field called dbHelper of class DBHelper (before the onCreate method)
4. Back in the onCreate method, construct a DBHelper instance — assign to the dbHelper field (note, you must pass this
5. Finally, call a new method -fillData()- gets the data and populates it using the helper, we haven't defined it yet
6. onCreate() should now look like this:

```
@Override
public void onCreate(Bundle icicle)
{
    super.onCreate(icicle);
    setContentView(R.layout.notepad_list);
    dbHelper = new DBHelper(this);
    fillData();
}
```

And remember to add the DBHelper field definition (right under the noteNumber definition):

```
private DBHelper dbHelper;
```

Step 10

More on menus

The notepad application we are constructing only scratches the surface with menus.

You can also add shortcut keys for menu items, create submenus and even add menu items to other applications!

Fill out the body of the onCreateOptionsMenu() method.

We are going to add just one menu item for now, "Add Item", using a string we will create in strings.xml, and defined with a constant we will create at the top of the class to identify the Add Item operation.

1. In strings.xml resource (under res/values), add a new string for menu_insert with text "Add Item"

2. Also, you need a menu position constant at the top of the Notepadv1 class (right under the KEY_BODY definition):

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

3. In the onCreateOptionsMenu() method, add the menu item. Also take care of the result of the super call being returned

```

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

```

Step 11

Fill out the body of the `onOptionsItemSelected()` method:

This is going to handle our new "Add Note" menu item. When this is selected the `onOptionsItemSelected()` method will be called with the `item.getId()` set to `INSERT_ID` (the constant we used to identify the menu item). We can detect this, and take the appropriate actions:

1. The `super.onOptionsItemSelected(item)` method call goes at the end of this method — we want to catch our events first
2. Switch statement on `item.getId()`
3. case `INSERT_ID`:
4. calls new method `createNote()`
5. break at the end of the case
6. return the result of the superclass `onOptionsItemSelected()` method at the end
7. The whole `onOptionsItemSelected()` method should now look like this:

```

@Override
public boolean onOptionsItemSelected(Item item) {
    switch (item.getId()) {
        case INSERT_ID:
            createNote();
            break;
    }

    return super.onOptionsItemSelected(item);
}

```

Step 12

Add a new `createNote()` method:

In this first version of our application, `createNote()` is not going to be very useful. We will simply create a new note with a title assigned to it based on a counter ("Note 1", "Note 2"...) and with an empty body. At present we have no way of editing the contents of a note, so for now we will have to be content making one with some default values:

1. `String noteName = "Note " + noteNumber++;` (Construct the name using "Note" and the counter we have defined in the class)
2. Call `dbHelper.createRow()` using `noteName` as the title and "" for the body
3. Call `fillData()` method again after adding (inefficient but simple)
4. The whole `createNote()` method should look like this:

```

private void createNote() {
    String noteName = "Note " + noteNumber++;
    dbHelper.createRow(noteName, "");
    fillData();
}

```

Step 13

List adapters

Our example uses a very simple array adapter which binds an array or list of items into a `ListView`. More commonly in Android, List Adapters go hand in hand with `ContentProviders`, and this is also a very easy way to use lists.

To bind a `ContentProvider` to a `ListView` you can use a `android.widget.SimpleCursorAdapter` to bind data from a `ContentProvider` into a `ListView`

Define the `fillData()` method. This is fairly long:

This method uses ArrayAdapter, which is the simplest way of putting data into a ListView. ArrayAdapter takes either a List or an array of Strings, and binds them into a text view provided in the layout defined for the list row (this is the text1 field in our notes_row.xml layout). The method simply obtains a list of notes from the database helper, constructs a List of Strings using the title strings from each row, and then creates an ArrayAdapter out of those items and bound to use the notes_row we defined.

```
private void fillData() {
    // We need a list of strings for the list items
    List<String> items = new ArrayList<String>();

    // Get all of the rows from the database and create the item list
    List<Row> rows = dbHelper.fetchAllRows();
    for (Row row : rows) {
        items.add(row.title);
    }

    // Now create an array adapter and set it to display using our row
    ArrayAdapter<String> notes =
        new ArrayAdapter<String>(this, R.layout.notes_row, items);
    setListAdapter(notes);
}
```

1. ArrayAdapter needs a List of Strings (List<String>) containing the items to display
2. The data is read out of the database as rows, and the title field from each row is used to populate the list of strings
3. We specify the notes_row view we created as the receptacle for the data
4. If you get compiler errors about classes not being found, ctrl-shift-0 or (cmd-shift-0 on the mac) to organize imports

Note: that for this exercise we use an ArrayAdapter, this is not a very scalable solution and more typically a SimpleCursorAdapter would be used with a ContentProvider or at least a Cursor returned from a query. See the sidebar on List Adapters for more information.

Step 14

Run it!

1. Right click on the Notepadv1 project
2. From the popup menu, select Run As -> Android Application
3. If you see a dialog come up, select Android Launcher as the way of running the application (you can also use the link)
4. Add new notes by hitting the menu button and selecting Add Item from the menu

Solution and Next Steps

You can see the solution to this class in Notepadv1Solution from the zip file to compare with your own.

Once you are ready, move on to Tutorial Exercise 2 to add the ability to create, edit and delete notes.

Back to the Tutorial main page...