Databases

Dr. Kanda Runapongsa Saikaew
Computer Engineering Department
Khon Kaen University
http://twitter.com/krunapon

SQLite and Content Providers

- SQLite offers a powerful SQL database library that provides a robust persistence layer over which you have total control
- Content Providers offer a generic interface to any data source by decoupling the data storage layer from the application layer
- By default, access to a database is restricted to the application that created it
- Content Providers offer a standard interface your applications can use to share data with and consume data from other applications— including many of the native data stores

Introducing SQLite Databases

- Using SQLite you can create independent relational databases for your applications
- Use them to store and manage complex, structured application data
- Android databases are stored in the /data/data/<package_name>/databases folder on your device (or emulator)
- By default all databases are private, accessible only by the application that created them
- In particular, when you're creating databases for resource-constrained devices (such as mobile phones), it's important to normalize your data to reduce redundancy.

Introducing Content Providers

- Content Providers provide an interface for publishing and consuming data, based around a simple URI addressing model using the content:// schema
- They let you decouple the application layer from the data layer, making your applications data-source agnostic by the underlying data source
- Shared Content Providers can be queried for results, existing records updated or deleted, and new records added
- Many native databases are available as Content Providers, accessible by third-party applications, including the phone's contact manager, media store, and other native databases

What is SQLite?

- *SQLite* is a well regarded relational database management system (RDBMS)
- It is
  - Open-source
  - Standards-compliant
  - Lightweight
  - Single-tier
- It has been implemented as a compact C library that's included as part of the Android software stack
- SQLite has a reputation for being extremely reliable and is the database system of choice for many consumer electronic devices, including several MP3 players, the iPhone, and the iPod Touch.

SQLite vs. Traditional RDBMs

- Lightweight and powerful, SQLite differs from many conventional database engines by loosely typing each column, meaning that column values are not required to conform to a single type
- Instead, each value is typed individually for each row. As a result, type checking isn't necessary when assigning or extracting values from each column within a row
- SQLite is a "zero-configuration" database engine. Programs that use SQLite require no administrative support for setting up the database engine before they are run

DBSample.java (1/2)

```
package edu.kku.android;
import java.util.List;
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;
public class DBSample extends Activity {
   private TextView output;
   private DBHelper dh;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        this.output = (TextView) this.findViewById(R.id.out_text);
        this.dh = new DBHelper(this);
        this.dh.deleteAll();
```

DBSample.java (2/2)

```
        this.dh.insert("Lab 8: Database and Content Provider");
        this.dh.insert("Lab 9: Location-based Services");
        this.dh.insert("Lab 10: Calling API and Working in the
Background");
        List<String> names = this.dh.selectAll();
        StringBuilder sb = new StringBuilder();
        sb.append("Names in database:\n");
        for (String name : names) {
           sb.append(name + "\n");
        }
        Log.d("EXAMPLE", "names size - " + names.size());
        this.output.setText(sb.toString());
    }
}
```

DBHelper (1/3)

```
public class DBHelper {
  private static final String DATABASE_NAME = "example.db";
  private static final int DATABASE_VERSION = 1;
  private static final String TABLE_NAME = "table1";
  private Context context;
  private SQLiteDatabase db;
  private SQLiteStatement insertStmt;
  private static final String INSERT = "insert into "
      + TABLE_NAME + "(name) values (?)";
  public DBHelper(Context context) {
      this.context = context;
      OpenHelper openHelper = new OpenHelper(this.context);
      this.db = openHelper.getWritableDatabase();
      this.insertStmt = this.db.compileStatement(INSERT);
  }
```

DBHelper (2/3)

```
public long insert(String name) {
      this.insertStmt.bindString(1, name);
      return this.insertStmt.executeInsert(); }
public void deleteAll() {
      this.db.delete(TABLE_NAME, null, null); }
public List<String> selectAll() {
      List<String> list = new ArrayList<String>();
      Cursor cursor = this.db.query(TABLE_NAME, new String[] {
"name" },  null, null, null, null, "name desc");
      if (cursor.moveToFirst()) {
        do {
            list.add(cursor.getString(0));
        } while (cursor.moveToNext());
      }
      if (cursor != null && !cursor.isClosed()) {
        cursor.close();
      }
      return list; }
```

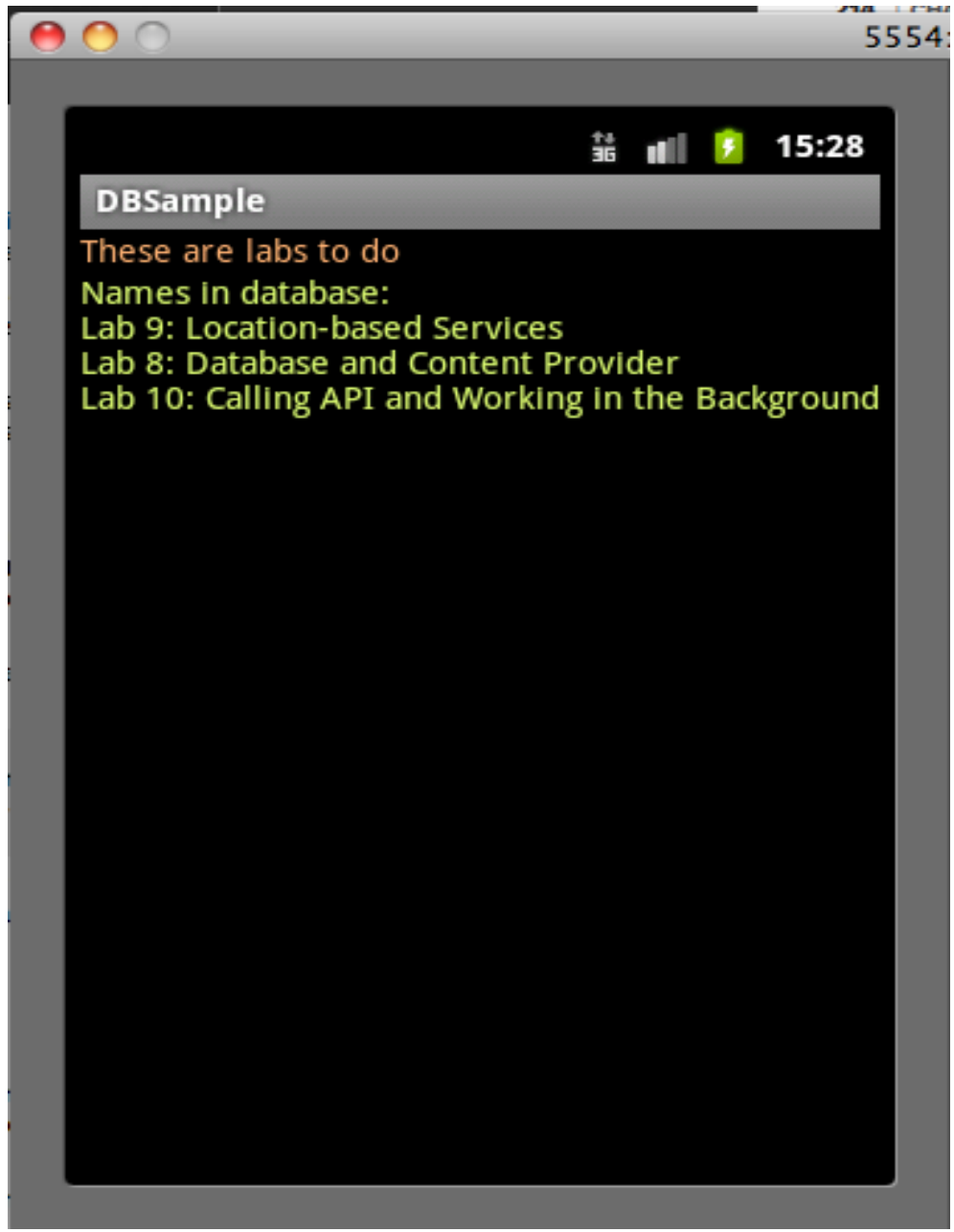
DBHelper (3/3)

```
private static class OpenHelper extends SQLiteOpenHelper {
      OpenHelper(Context context) {
        super(context, DATABASE_NAME, null,
DATABASE_VERSION); }
      @Override
      public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE " + TABLE_NAME +
          "(id INTEGER PRIMARY KEY, name TEXT)"); }
     @Override
      public void onUpgrade(SQLiteDatabase db, int oldVersion,
int newVersion) {
        Log.w("Example", "Upgrading database, this will drop
tables and recreate.");
        db.execSQL("DROP TABLE IF EXISTS " +
TABLE_NAME);
        onCreate(db);
     }}
```

Layout res/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
 xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content">
 <LinearLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
     android:orientation="vertical" android:layout_width="fill_parent"
     android:layout_height="fill_parent">
  <TextView android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/hello"
      android:textColor="#FFB273"/>
  <TextView android:id="@+id/out_text"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:textColor="#D9F970" android:text="" />
 </LinearLayout></ScrollView>
```

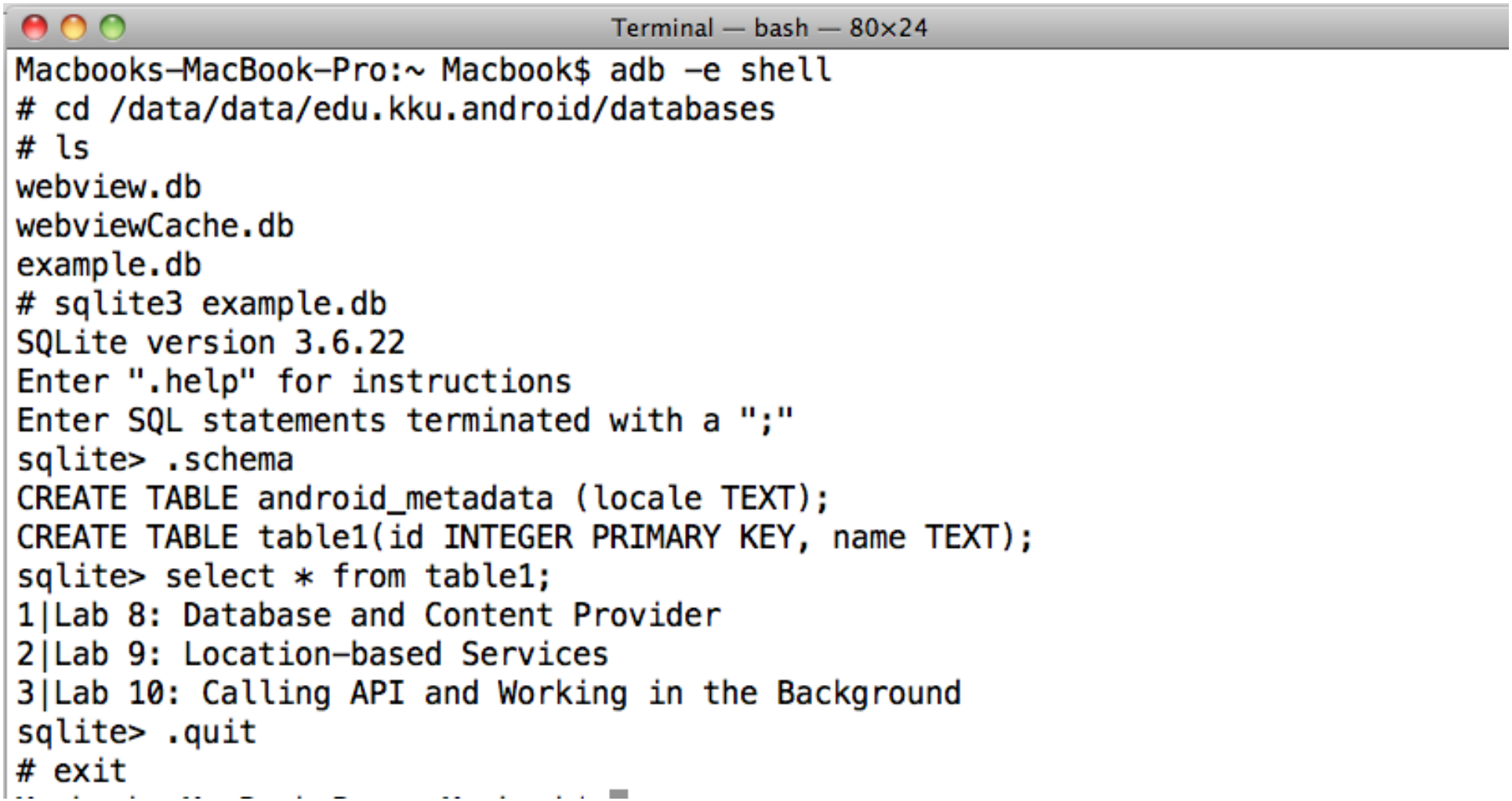
DBSample Result



Accessing Android SQLite Data

- Android data in SQLite is stored in **/data/data/ [PACKAGE_NAME]/databases**
- To access the database, you need to start your application and then open the terminal or the command prompt window and type with command
  - adb -e shell
- Then you access the database with command
  - sqlite3 <database name>
- Once you are in sqlite3 session, you can learn all commands that you can use by typing
  - .help

Android SQLite Database



SQLiteOpenHelper

- SQLiteOpenHelper is an abstract class used to implement the best practice pattern for creating, opening, and upgrading databases
- By implementing an SQLite Open Helper you hide the logic used to decide if a database needs to be created or upgraded before it’s opener
- To use an implementation of the helper class, create a new instance, passing in the context, database name, and current version, and a CursorFactory (if you’re using one).
- Call getReadableDatabase or getWritableDatabase to open and return a readable/writable instance of
- the underlying database

Revisited To-Do List (Saving State)

- Please download and modify the To-Do List sample code in chapter 6 of Professional Android 2 Application Development and then modify these things
  - Modify the display to also show time of task to do
  - Change the color to be what you want
- Saving activity preferences
  - If you want to save Activity information that doesn't need to be shared with other components (e.g., class instance variables), you can call Activity.getPreferences() without specifying a Shared Preferences name
  - Access to the returned Shared Preferences map is restricted to the calling Activity

Saving Activity State

```
protected void onPause() {
    super.onPause();

    // Get the activity preferences object.
    SharedPreferences uiState = getPreferences(0);
    // Get the preferences editor.
    SharedPreferences.Editor editor = uiState.edit();
    // Add the UI state preference values.
    editor.putString(TEXT_ENTRY_KEY, myEditText.getText().
toString());
    editor.putBoolean(ADDING_ITEM_KEY, addingNew);
    // Commit the preferences.
    editor.commit();
  }
```

Restoring Activity State

```
private void restoreUIState() {
    // Get the activity preferences object.
    SharedPreferences settings = getPreferences(0);
    // Read the UI state values, specifying default values.
    String text = settings.getString(TEXT_ENTRY_KEY, "");
    Boolean adding = settings.getBoolean(ADDING_ITEM_KEY,
false);
    // Restore the UI to the previous state.
    if (adding) {
      addNewItem();
      myEditText.setText(text);
    }
  }
```

Saving and Restoring Instance State

- To save Activity instance variables, Android offers a specialized variation of Shared Preferences.
- By overriding an Activity's onSaveInstanceState event handler, you can use its Bundle parameter to save UI instance values.
- Store values using the same get and put methods as shown for
- Shared Preferences, before passing the modified Bundle into the superclass's handler
- Programmers can save and restore instance state by overriding methods onSaveInstanceState and onRestoreInstanceState

Saving Instance State

```
  private static final String SELECTED_INDEX_KEY ="
SELECTED_INDEX_KEY";
@Override
  public void onSaveInstanceState(Bundle outState) {
    outState.putInt(SELECTED_INDEX_KEY, myListView.
getSelectedItemPosition());
    super.onSaveInstanceState(outState);
  }
```

- This handler will be triggered whenever an Activity completes its active lifecycle, but only when it's not
- being explicitly finished (with a call to finish).

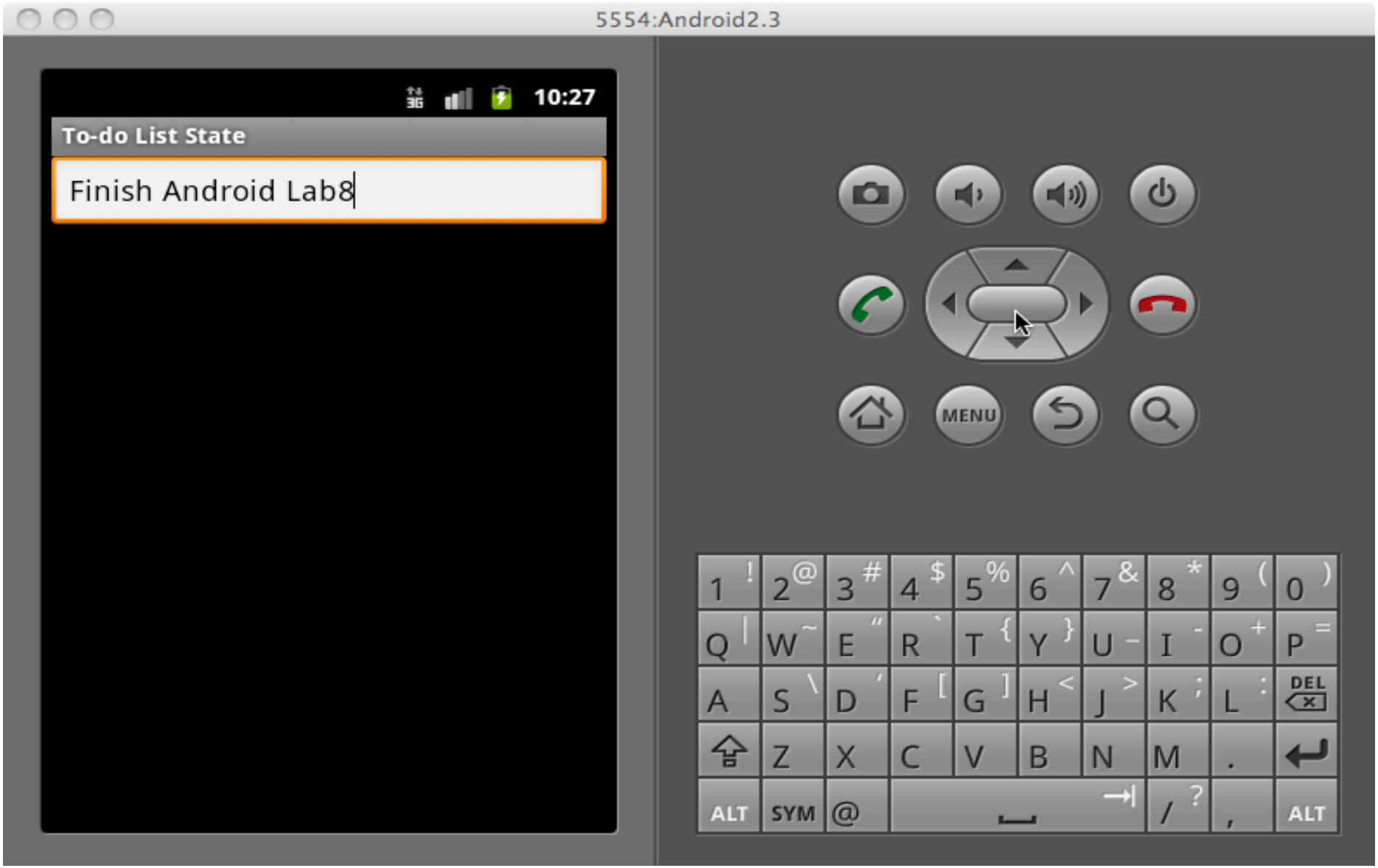
Restoring Instance State

```
@Override
  public void onRestoreInstanceState(Bundle
savedInstanceState) {
    int pos = -1;
    if (savedInstanceState != null)
      if (savedInstanceState.containsKey
(SELECTED_INDEX_KEY))
        pos = savedInstanceState.getInt
(SELECTED_INDEX_KEY, -1);
    myListView.setSelection(pos);
  }
```

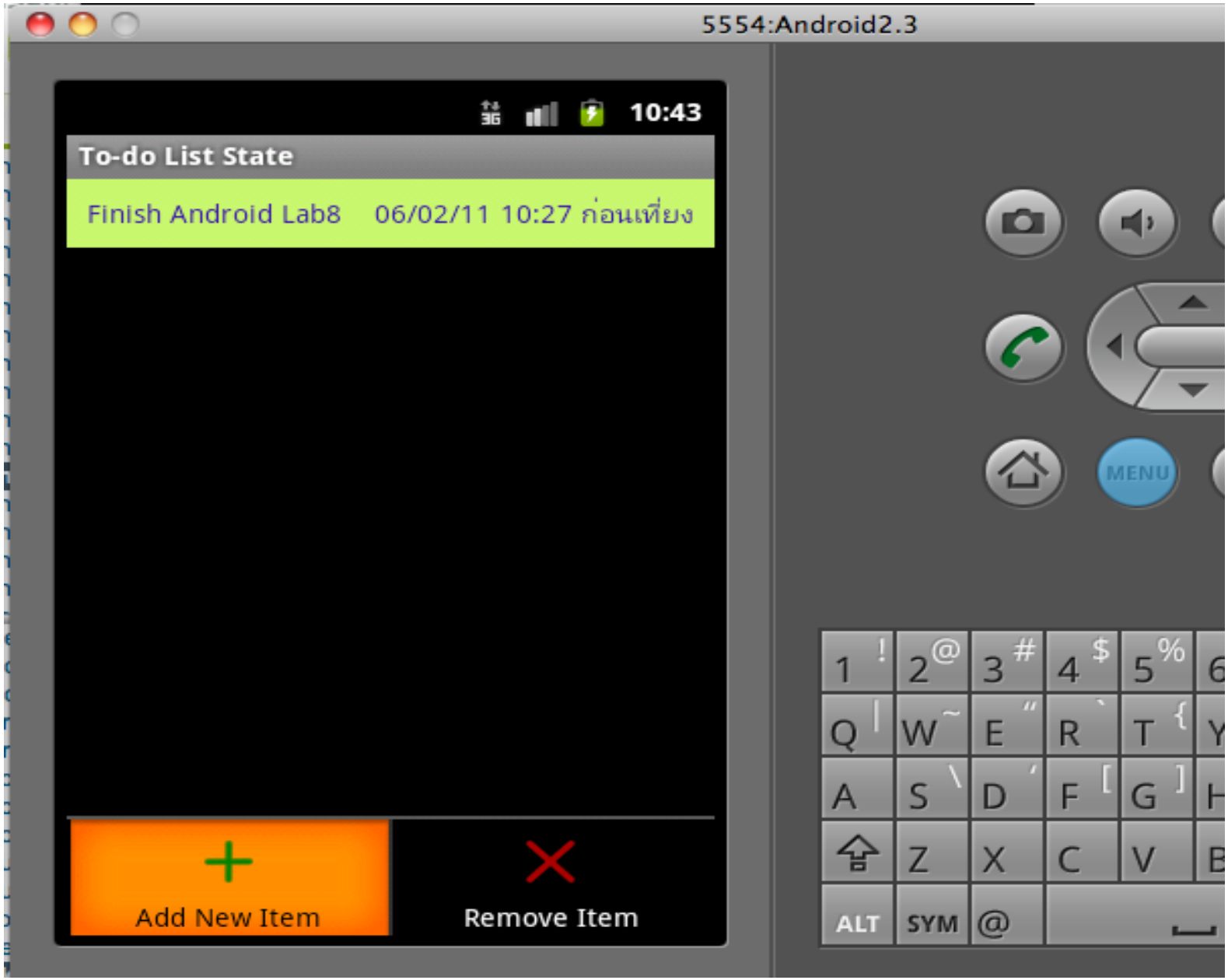
To Do List with State Saving (1/5)

- Click button Menu to display options menu

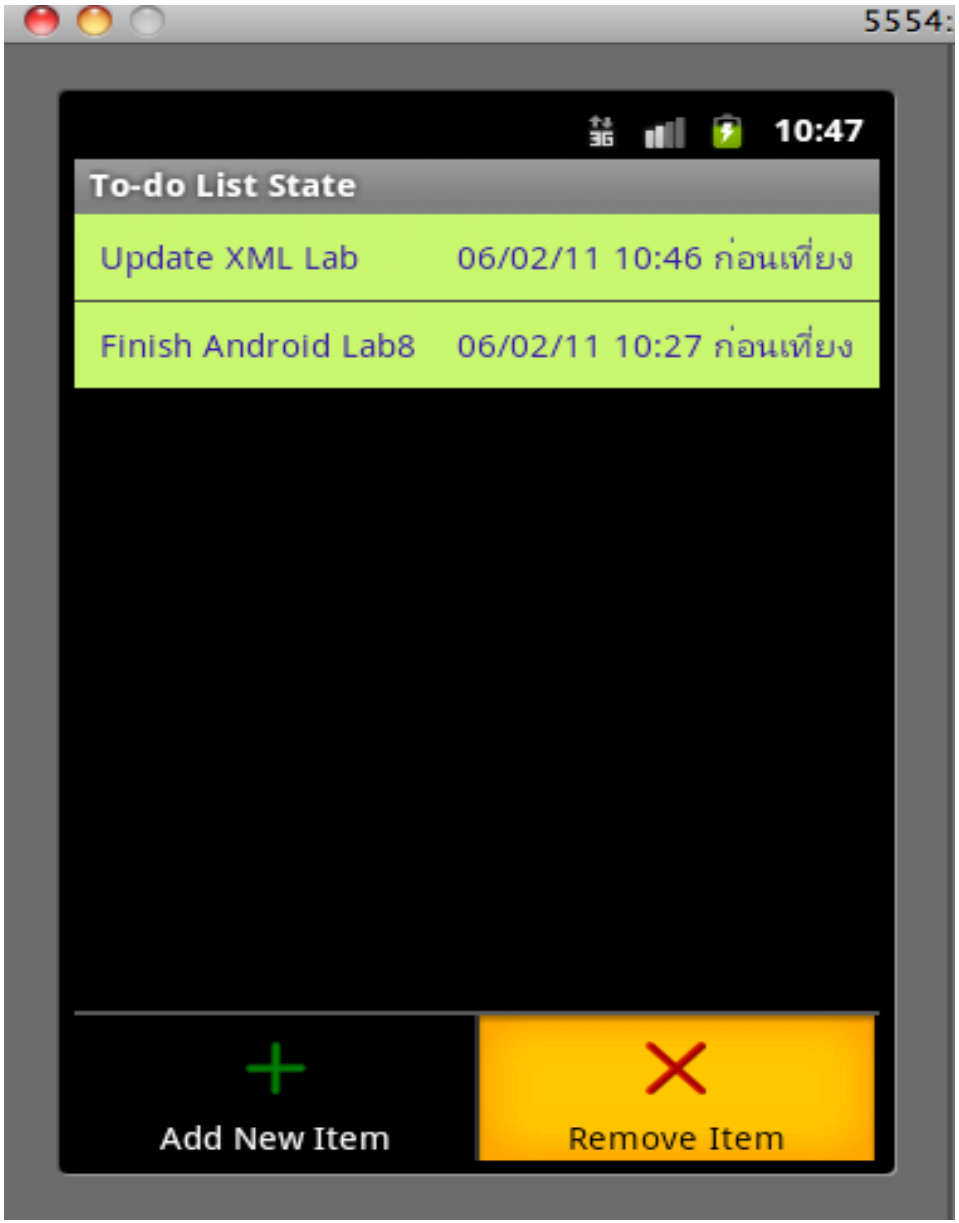
To Do List with State Saving (2/5)



To Do List with State Saving (3/5)

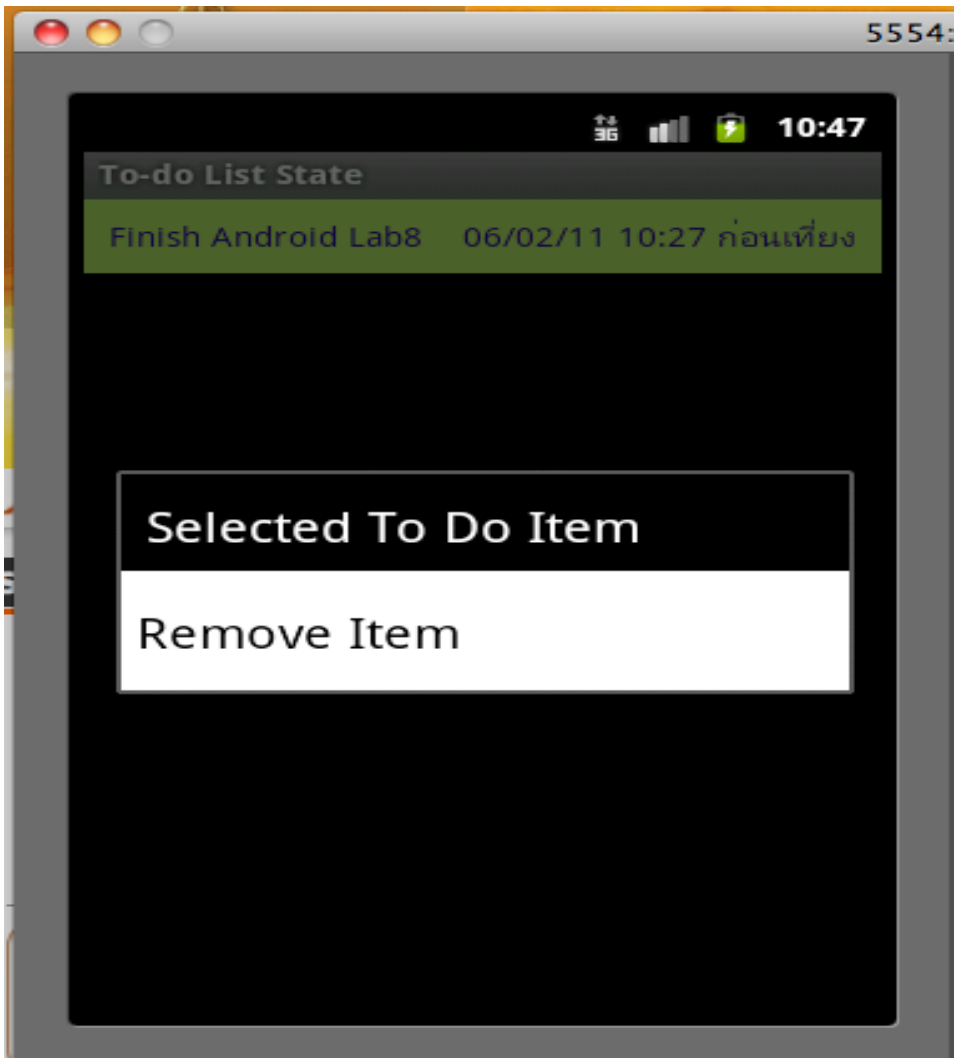


To Do List with State Saving (4/5)



To Do List with State Saving (5/5)

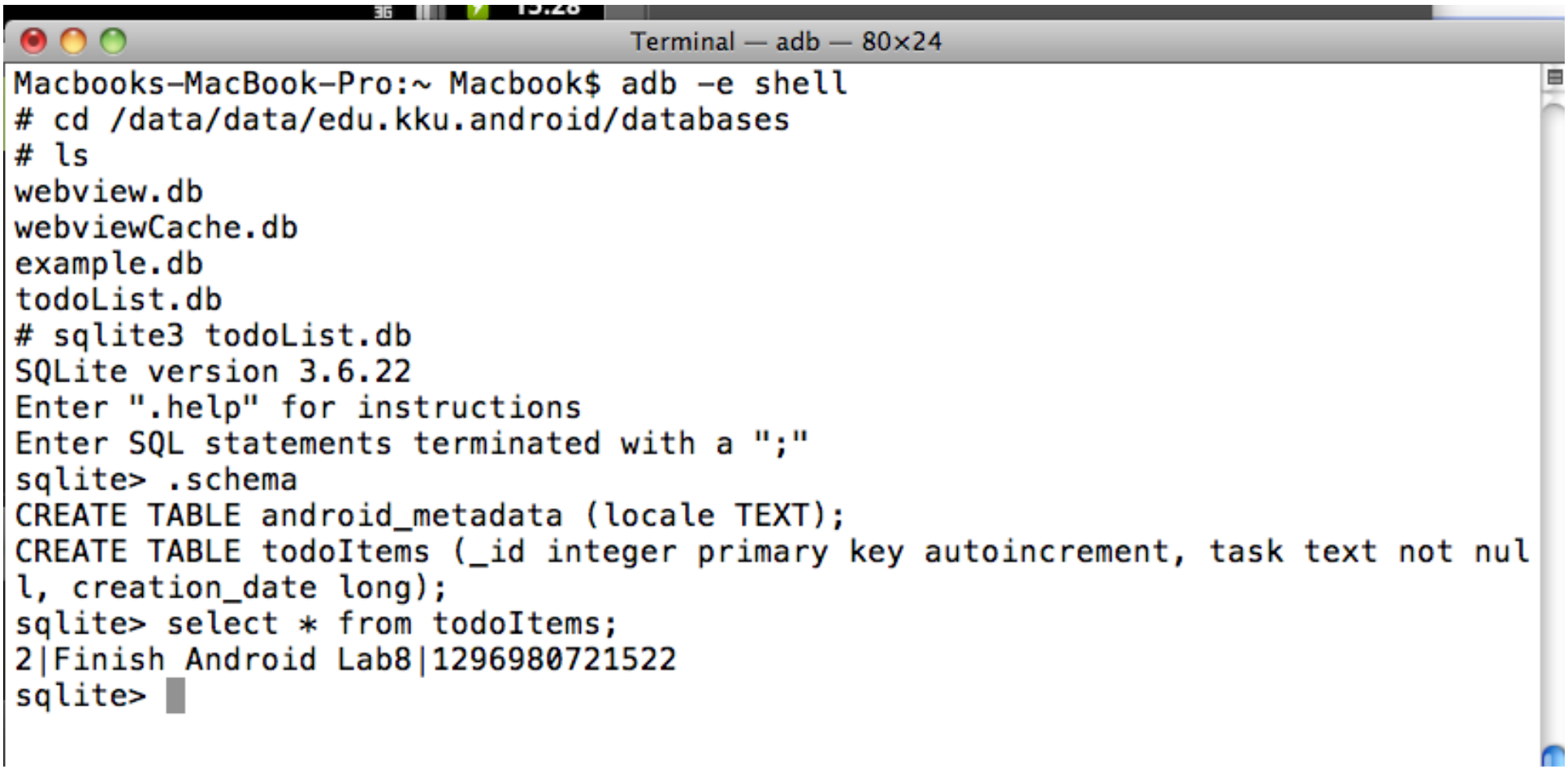
- Press at the view long enough to see the context menu



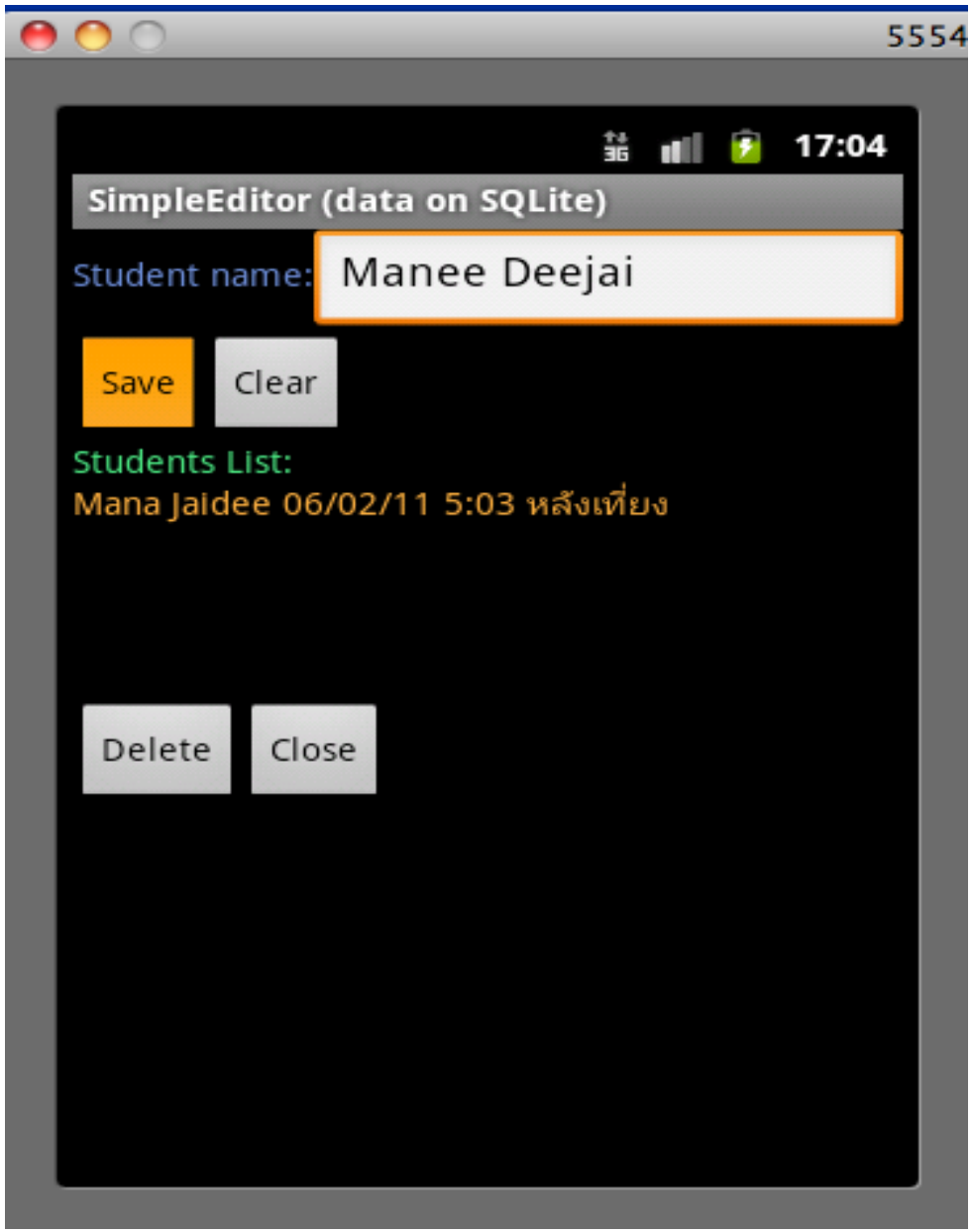
Revisited To-Do List (Using Database)

- In the previous version of To-Do list, after we close the program and the emulator, when we open the program again, all to-do items disappear
- Now we will save all to-do items in a database
- To-Do List sample code in chapter 7 of Professional Android 2 Application
- After you are done with this, Your to-do items will now be saved between sessions

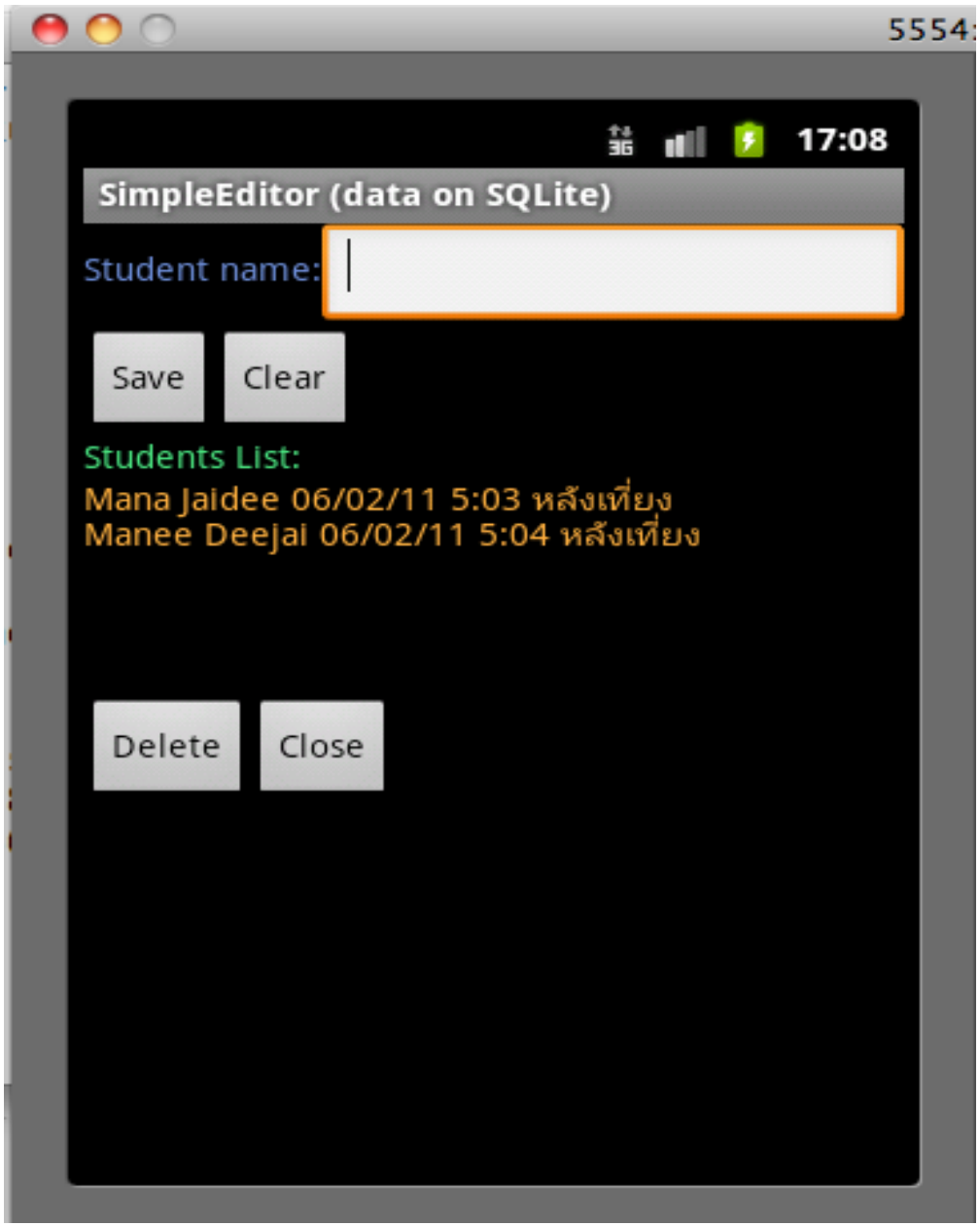
To Do Items in a Database



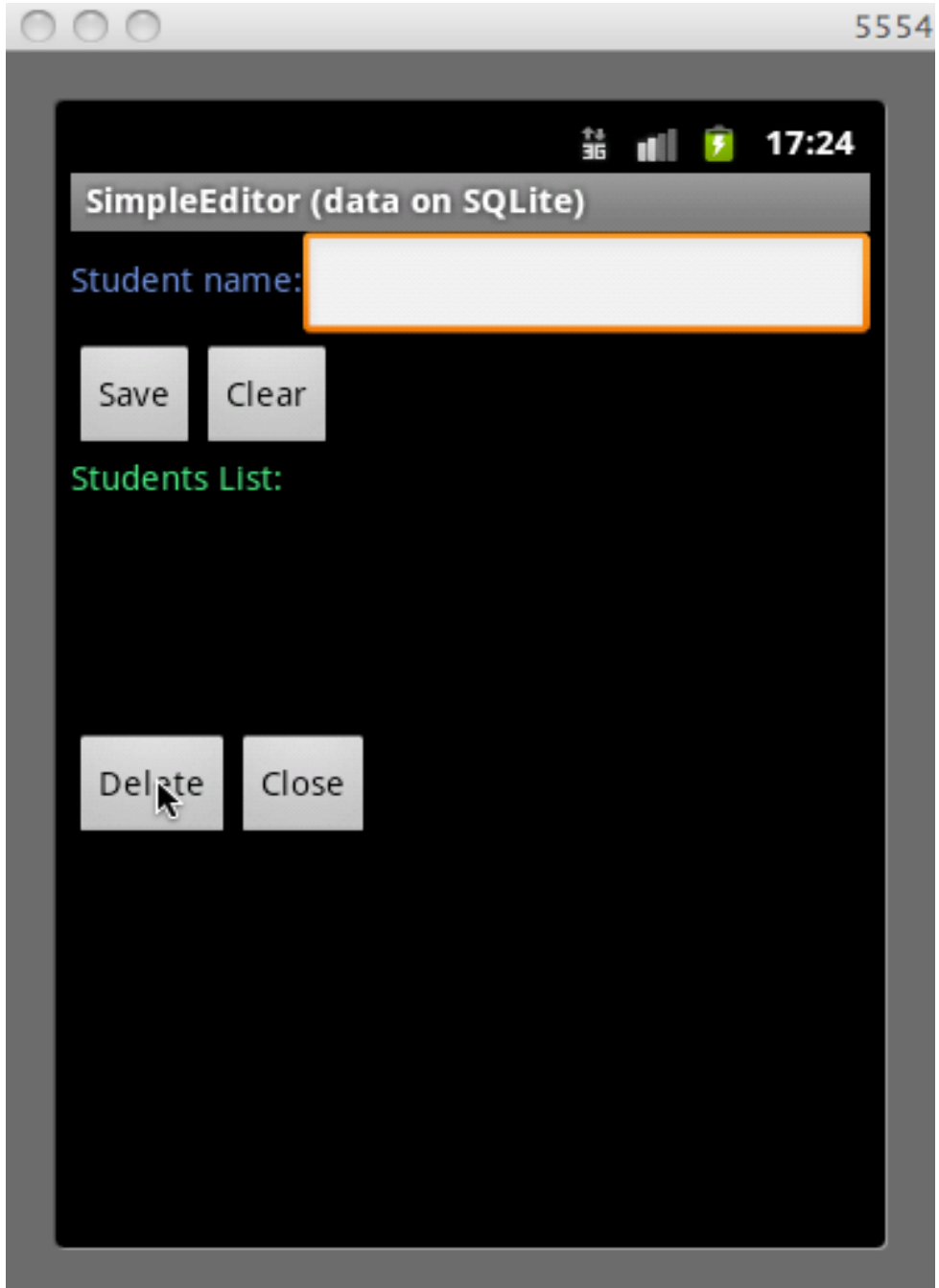
Students DB (1/6)



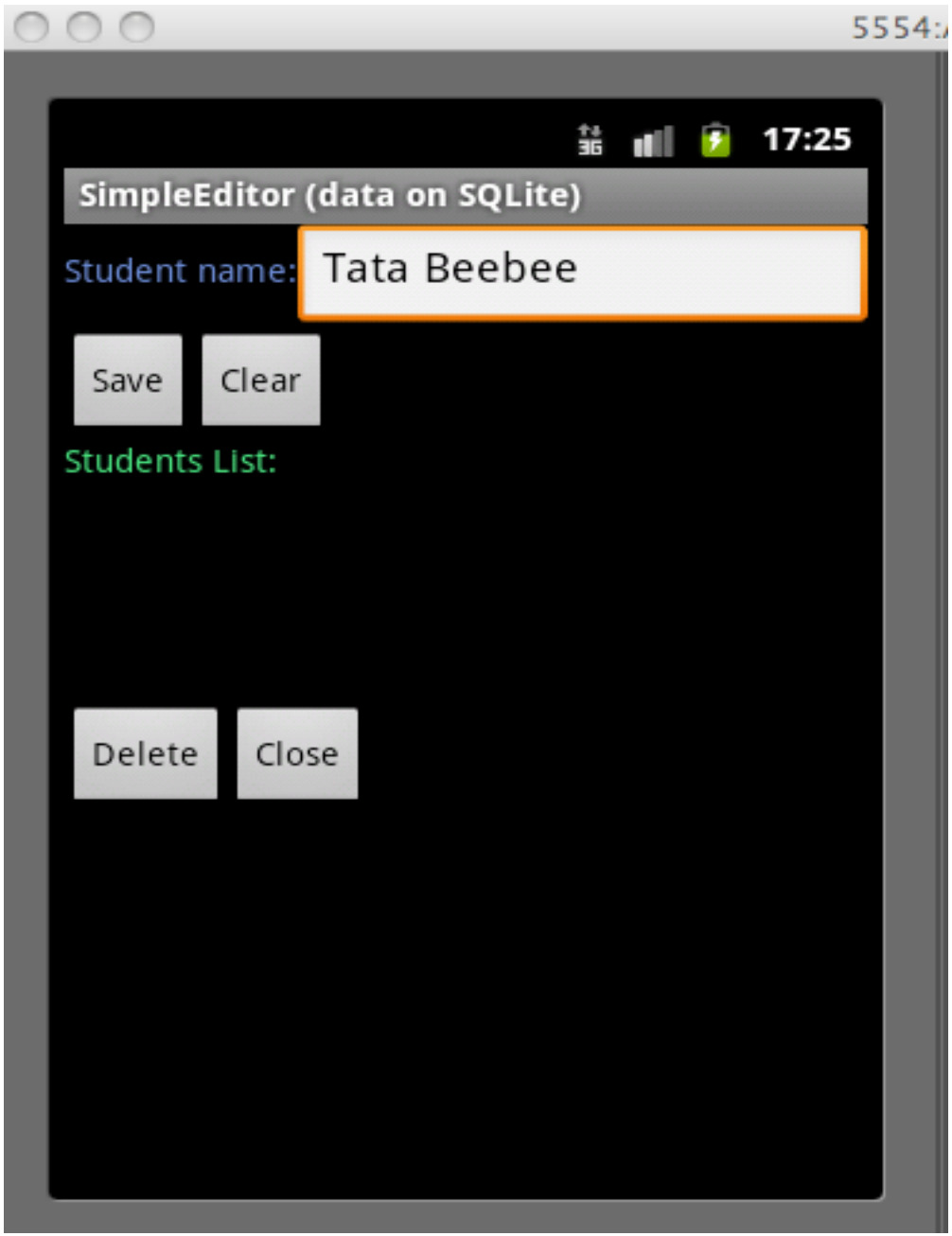
Students DB (2/6)



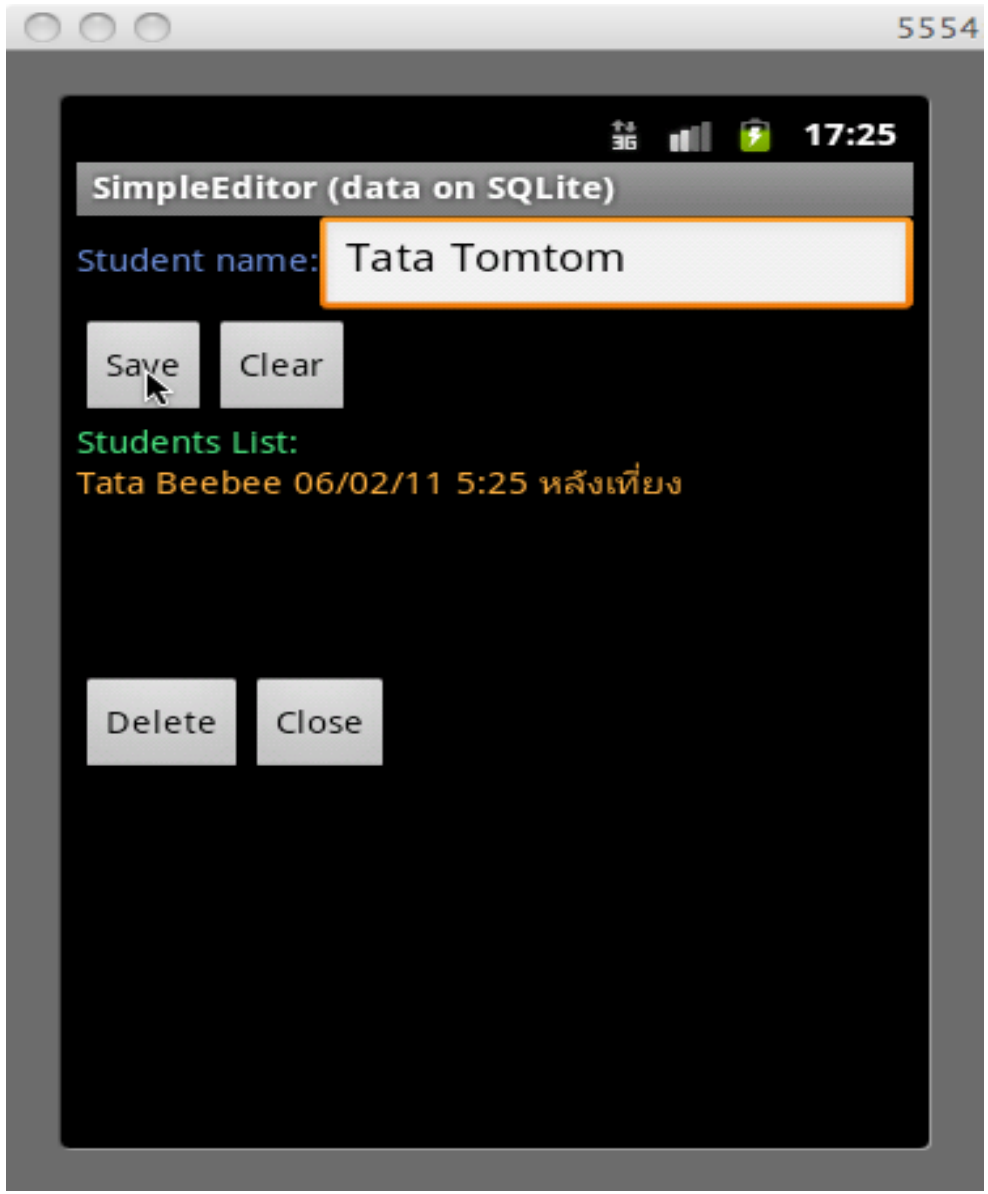
Students DB (3/6)



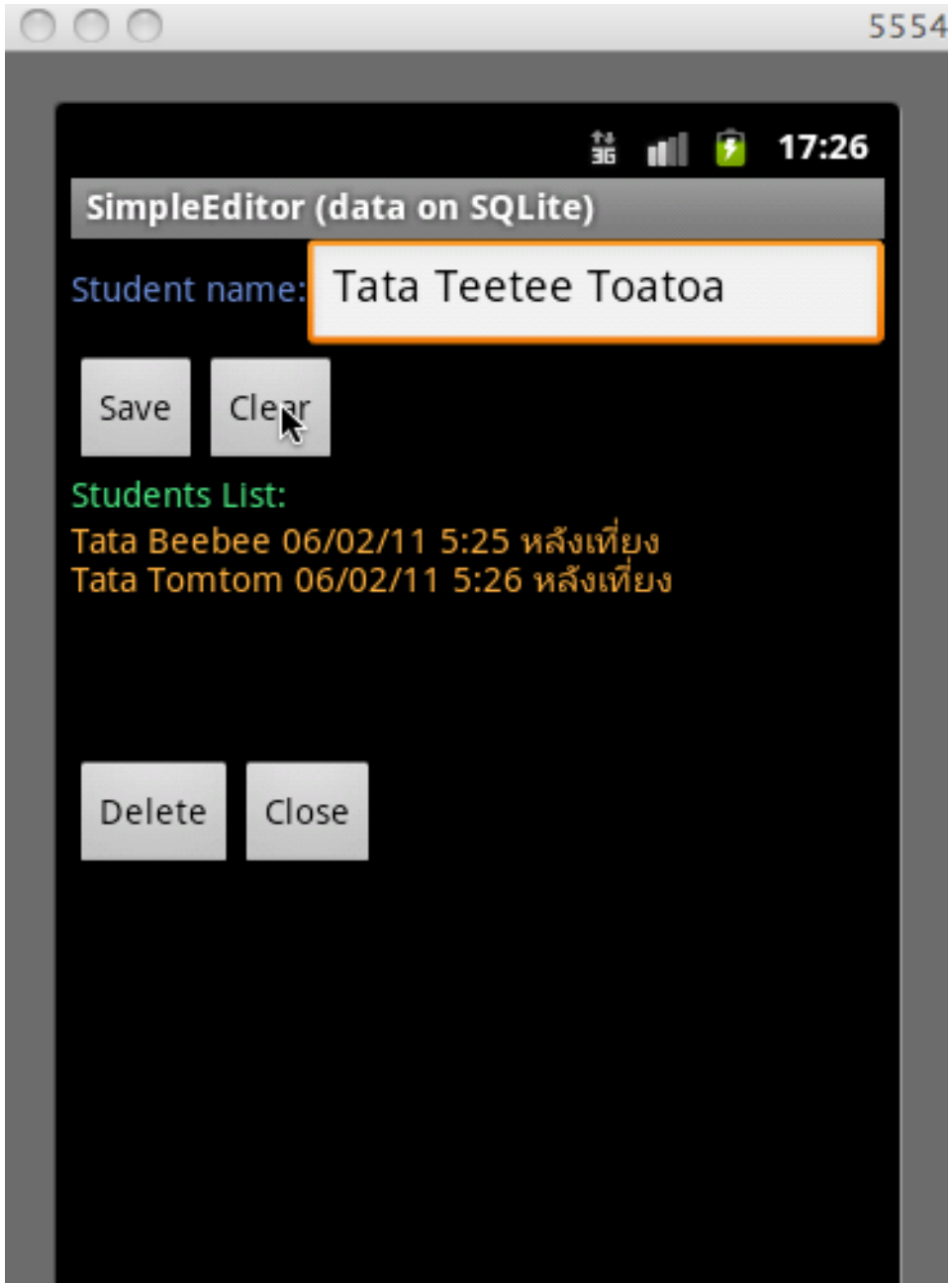
Students DB (4/6)



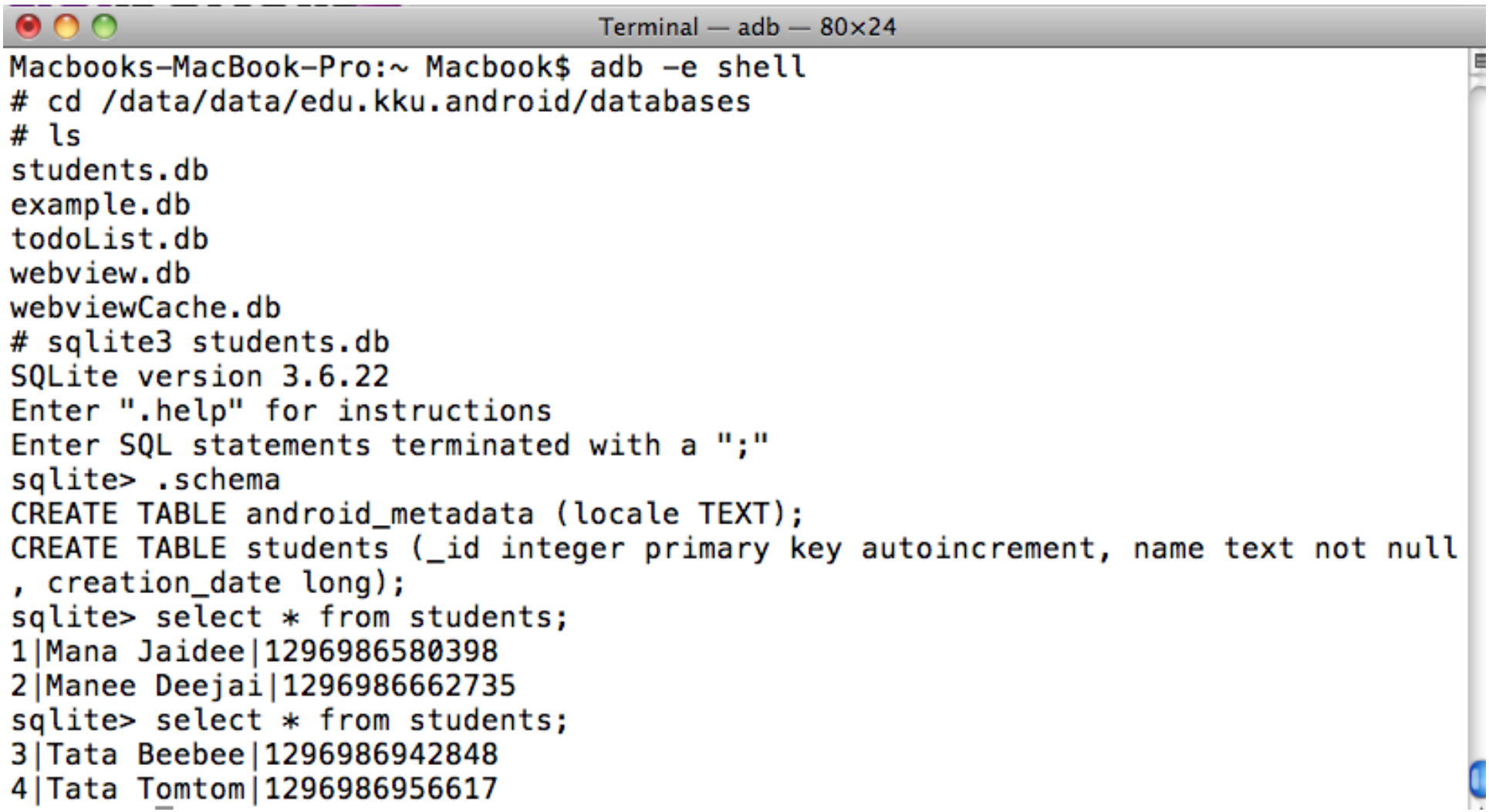
Students DB (5/6)



Students DB (6/6)



Data in SQLite



References

- Charlie Collins, "**Android** SQLite Basics: creating and using a database, and working with **sqlite3"** http://www.screaming-penguin.com/node/7742
- Reto Meier, "Professional Android 2 Application, Development", http://www.wrox.com/WileyCDA/WroxTitle/Professional-Android-2-Application-Development.productCd-0470565527,descCd-DOWNLOAD.html