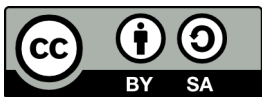
Hands-On Exercises for

Android Development

v. 2012.08

WARNING:
The order of the exercises does not always follow the same order of the explanations in the slides. When carrying out the exercises, carefully follow the exercise requirements. Do **NOT** blindly type the commands or code as found in the slides. Read every exercise **in its entirety** before carrying out the instructions.



These exercises created by: Karim Yaghmour

Originals at: www.opersys.com/training/android-development

Delivered and/or customized by:

Android Class Exercises – Day 1

A quick introduction to Java

1. Install Sun's JDK and JRE (sun-java6-jdk and sun-java6-jre)
2. Create and run a HelloWorld in Java
3. Create a java program that creates two threads that call a same synchronized function that increments a number shared by the two threads and prints out the number's value along with a string identifying the caller.

Introduction to Android Development

1. Install the SDK
2. Install the Eclipse plugin
3. Configure Eclipse for the SDK
4. Use the “android” tool to install platform support for 2.3. and 3.0
5. Create an Android 2.3 emulator image
6. Create a “Hello World” project using Eclipse and run it in the emulator
7. Make your “Hello World” use the toast function
8. Make your “Hello World” use the logging function and use logcat to view your output

Application Fundamentals

1. Implement the onPause() handler for your “Hello World” app and monitor what happens when you switch applications
2. Implement onSaveInstanceState() to store data in a Bundle and check that the data is there in the subsequent onCreate()
3. Implement the onConfigurationChanged() and onRetainNonConfigurationInstance callbacks and monitor their behavior
4. Implement a Service that is started by your “Hello World” ... nothing fancy, just make sure the Service's onStart() is called

User Interface

From here on, we will start building components for a full app to which we will add more functionality along the way. The app is basically a Twitter-like client. The goal in this section is to create the following elements which will become “active” as we tie the components together:

1. Main Activity for:
    1. Displaying list of events (ListView)
    2. Creating a new event (EditText)
    3. Managing list of people “followed” (Other Activity)
2. Menu for:
    1. Entering credentials (Dialog)

    2. Changing background color (Dialog)
3. People Activity:
    1. Adding new person (EditText)
    2. List of people “followed”
4. Context menu for item in PeopleActivity
5. Credentials dialog (custom)
6. Background color dialog (custom)

Android Class Exercises – Day 2

Application resources

1. Have different layout for portrait and landscape mode for the UI elements created earlier.

Intents

1. Implement a Broadcast Receiver that ties in to the Intent of your choice. You can see a full list of Intents you can listen to in the documentation. On firing, the Broadcast Receiver should notify the status bar. Make the selecting of the status bar expanded start the main user interface Activity implemented earlier.

Data storage

1. Use the Shared Preferences to store/retrieve the username/password and color from your user interface.
2. Add a menu option and corresponding Activity or Dialog allowing the user to enter a URL to a file on the web. Implement the necessary functionality to retrieve the file from the net, store it locally and list the download in the event list.
3. Enable the user to "open" the file by selecting the corresponding event in the event list. Try using an Intent to get an app to take care of opening the file and presenting it to the user.
4. Use SQLite to store new events entered in the TextBox by the user. Those text events should also be added to the ListView and the ListView should restore itself on Activity restart. Use SQLiteOpenHelper().

Content providers

1. Take the store/retrieve functionality implemented in #4 of the previous section and wrap it inside a basic Content Provider. Your Activity should now be using a Content Provider to read/write its data. Your Content Provider should continue using SQLite for its storage.
2. Take the file functionality implemented in #2 and #3 of the previous section and make the files stored by the Content Provider instead of locally in the Activity.
3. Make the Broadcast Receiver implemented in #1 of the Intents section exercise fire off a service that adds the description of the Intent captured by the Broadcast Receiver to the Content Provider. You will likely need to start a Service for that. Notice that your App will not see the new data in the Content Provider's DB since its last restart. Fixing this will be part of a later exercise.

Android Class Exercises – Day 3

REST applications

1. Tie the previously implemented Content Provider to the “http://identi.ca” service. Identi.ca is a Twitter-like service. It is based on the StatusNet open source project and provides a REST API. You can use parts of the API without having an account, but you will need to create an account in order to be able to post and to subscribe to friends' feeds. You can try out the REST API commands from your command line using the “curl” command in Ubuntu; some of them can also run from your browser. For example, here's how you:
   - A user's timeline (no authentication required):
     - curl http://identi.ca/api/statuses/friends_timeline/usr.xml
   - Post (authentication required):
     - curl -u usr:pwd http://identi.ca/api/statuses/update.xml -d status='Howdy!'
   - See your friends' timeline (authentication required):
     - curl -u usr:pwd http://identi.ca/api/statuses/friends_timeline.xml

   Use the Sync Adapter example to copy/paste code in as much as possible. By the end of this exercise, you should be able to follow/unfollow other users and post messages to your feed so that everyone can see them.

   You will find more documentation about StatusNet's API here:
   http://status.net/wiki/HOWTO_Use_the_API
   http://status.net/docs/api/
   http://status.net/wiki/Twitter-compatible_API
   In addtion to XML, you can also get information in: json, rss, atom.

Android Class Exercises – Day 4

Remote Interfaces

1. Create a service that provides a remote interface that provides the following through the previously-implemented Content Provider:
   - Sending a new event
   - Getting the last N events
2. Create a new app that has a single Activity that uses the previously-implemented service and has two buttons:
   - One for sending an event with the current time
   - One for getting the last event and showing it in a dialog box

Note here that by using two apps you'll be running in two separate processes, hence realizing the benefits of using a remote interface.

Security and Permissions

1. Modify the service implemented in the previous exercise set to require a set of permissions for accessing its API.
2. Modify the new app implemented in the previous exercise to use the service permissions to access the service's API.

Device Administration

1. Create a device administration app that:
   - Sets the minimum password length to 10 characters and asks the user to enter a password
   - Sets the maximum user inactivity period to 10 seconds
2. Write an app that uses an intent of type ACTION_VIEW on the package manager service to install a “Hello Wold!” app apk. Use the “assets/" directory to store the “Hello World” apk. You will need to copy the asset to local storage to get an absolute path to pass with the Intent.

Android Class Exercises – Day 5

Native Development

1. Get and install the NDK
2. Write an app that uses JNI to get the contents of /proc/cpuinfo and displays them to the user.
3. Write a fully native app that regularly updates a world-readable file and write a simple Activity that opens the file and shows its contents to the user. Note that the app name in the manifest **must** be: “android.app.NativeActivity”.