Programming in Android

Nick Bopp

nbopp@usc.edu

Types of Classes

- Activity
  - This is the main “Android” class that you will be using. These are actively displayed on the screen and allow for user interaction.
- Service
  - Non-visible code that can be started by an activity or bound to an activity. Usually spawns background threads for transferring data, playing music, or doing any CPU heavy process outside the main UI thread.

Types of Classes 2

- Content Provider
  - This is a means of exposing data available for use by other applications, whereas by default data is typically private to single applications.
- Broadcast Receiver
  - Essentially a listener that waits for specific events and then begins activities, or alerts the user in some way.

Intents – How Things Are Started

- In normal Java and most of Android, an object is instantiated:
  - E.g. Object o = new Object();
- In Android, all new Activities and Services are started using Intent objects, and then a start method()
  - Intent j = new Intent(this, ActivityToStart.class);
  - startActivity(j); OR startActivityForResult(j);
- This makes navigation through an application rather tricky.

The Activity Stack

- When an application is created, one activity is specified with an intent filter to receive the launcher's request to start the application. This essentially makes the activity the "starting" activity for the app.
- Once in this activity, you can either add a new activity to the stack or finish() the activity, which effectively pops the only activity off the stack and returns to the home screen.

Stack Example

- Launcher starts A().
- A() fires an Intent for B().
- B() fires an Intent for A().
- User presses the back button(pops an activity off the stack).
- Usually, this will result in B() being displayed: (A->B->A) pops top A to become (A->B), a successive pop would go to A, and one more would return to the home screen.

How To Change Up the Order (Somewhat)

- “Launch Modes” and “Affinities”: attributes you can set in the manifest that determine whether there can be multiple instances of an Activity and what tasks they belong to.
- However, you can never simply hop around the stack choosing whichever previously called activity you like and returning to it.

When the User Goes Away

- There are several ways to deal with the user leaving the application for a while.
- Usually Android handles this by just getting rid of stack, and retaining only the Activity at the very base.
- However, developer can specify extremes here
  - No matter how long you are gone, try to hold onto the state of the call stack (dependent upon available memory and priority)
  - As soon as you leave the application for any reason, immediately clear the call stack (useful for smaller apps).

The Activity Lifecycle

- http://developer.android.com/images/activity_lifecycle.png

Saving States

- Android activities can be killed quite often, and while when this happens is technically defined, in practice you'll find it can seem quite erratic.
- For this reason, you'll want to make calls to onSaveInstanceState() and onRestoreInstanceState() so that the user cannot tell what's going on in the background.

UI Design

- UI's are built out of Views and ViewGroups.
- These are arranged into a tree with a single root ViewGroup.
- ViewGroups define how their members are arranged.
- E.g. LinearLayout is a child of ViewGroup which arranges all Views within it in a simple linear patter (horizontally or vertically).
- Different attributes can be specified, these can all be found in the developer pages online.

Two Ways to Make UI Layouts

- Programatically inside of Activity's onCreate() code.
- Inside of layout XML files.
- Generally XML is the way to go.
- The Java then identifies these views by ID's specified by the XML file and allows for user input to interact with the system.

Built in Widgets

- Widgets combine Views, custom graphics / drawables, and add handy methods for performing some of the most basic UI tasks.
- TextView – displays text
- EditView – allows user to input text
- Button
- NumberScroll – Useful for entering dates / times
- Lists

Menus

- Users can hit the menu button to pull up a menu.
- That menu then has a series of buttons that you can label, and another method that reacts to whatever button it was that was hit.

Dialogs

- These are the equivalents of pop up windows. They usually tell a user than they've made a mistake or need to redo something.
- The activity is faded into the background and the Dialog box pops up.

Toasts

- A toast is a smaller, non-interactive version of Dialog that doesn't fade out or impede the Activity in any way. These provide nice short little notifications.
- Personally, I use them to let the user know when files download, when their GPS location changes, etc.
- Generally this information should not be crucial to the user, because this does not persist. If the phone is in your pocket, the toasts don't queue up and wait for the screen to be turned on again.

Data Storage

- Several choices:
  - Preferences
  - Files on the SDCard or internal memory
  - SQLite database on the phone
- Preferences is basically a map objects that only works for primitives.
- Limited experience using SQLite on the phone, though I suppose it would come in handy at times.

File I/O – Your Best Friend

- Pictures take up lots of space!
- Running low on memory causes Android to reclaim memory by destroying older / non-visible activities.
- Dealing with all these random kills is a hassle.
- Solution – Use as little memory as possible by storing temporary data in files and reading / writing.

A Few Minor Differences

- A file can't be created directly in the normal java fashion of: File f = new File("filename");
- Instead use openFileOutput() and openFileInput(). These return FileInputStreams or FileOutputStreams that can then in turn work just like every other Java I/O.
- Files can be made to either be private to the application (default), or globally viewable.

Application Manifest

- This is where you state all the stuff that goes in your application.
- Generally this includes: any of the 4 “Android classes”, permissions, and so on.
- Also, Intent Filters are specified here, and in particular, the main Intent Filter must be specified around your “root” activity so that it is possible to open up your app in the first place.

Log Class

- This is just handy.
- It is basically your bread and butter for debugging, when you're running the emulator or the app on your device, you will pull up the DDMS perspective in eclipse. This has tons of useful information (threads running, heap size, file tree) as well as the Log screen, which timestamps every message sent out from the phone.
- Useful for tracking variables, and the state of your app.

Of Particular Interest to Project

- Multithreading is very important for the sorts of things I (and you) will be doing.
- Networking (hand in hand with multithreading) is likewise important, downloading a file in the main thread will cause issues. If the user presses the screen (EVER) and nothing happens for 5 seconds, you get a non-responsive window the user can forcibly close the application.
- GoogleMaps and Location data

Threading and UI Updates

- Threads are made and run in the same way as usual Java, but there are some specific concerns.
- There is no way to update the screen from a separate thread. For this reason you use a Handler object, which you can give a runnable to run on the main thread. Handlers always "remember" what thread they were created in, and can post Runnables from any thread to the thread they were created on.

Networking In Android

- There are several ways to do this.
- I believe standard Socket programming in Java works? Not too sure here, haven't used it.
- A much simpler way is to use the URLConnection class and others in java.net package.
- Also, Apache has its own packages that can be used for networking which provide the same basic function, but in a different structure.

GoogleMaps and Location

- Google has built in a lot of functionality to using their GoogleMaps in Android.
- In particular, the MapView is a view that can be used to display a GoogleMap on screen without too much work required on developers.
- NOTE: When you try this the first time, you will see gray tiles but no actual map. You need to generate a key from your debug.keystore online, just search for Android GoogleMap Registration or something similar and you'll see what I'm talking about.

Location and Sensor Updates

- Location updates are pretty simple, you just create an Activity that requests updates for GPS updates every 10 seconds or so, and then there is a method in that Activity that receives the new location and does whatever you want.
- Generally, getting accelerometer data works in a very similar fashion.