Android: Fundamentals and SDK Tools

Tommy MacWilliam

Harvard University

February 15, 2011

Announcements

- Lecture videos available at: https://www.cs76.net/Lectures
- Section information: https://www.cs76.net/Sections

Today

- Android
- API Overview
- Tools

Section Feedback

- http://tommymacwilliam.com/e76/feedback
  - let me know how I'm doing!
- I don't like long surveys either, so give me feedback via an anonymous (I promise) 140-character tweet!

A Brief History

- July, 2005: Palo Alto startup Android, Inc. acquired by Google
  - omg Google phone omgomgomg
- November, 2007: formation of the Open Handset Alliance
  - business consortium of carriers as well as software, commercialization, semiconductor, and handset companies
  - over 80 members, including Sprint, T-Mobile, Intel, Nvidia, Qualcomm, TI, HTC, LG, Motorola, Samsung
    - also eBay?
- October 2008: Android software open-sourced under Apache license

A Brief History

- April, 2009: Cupcake (1.5)
- September, 2009: Donut (1.6)
  - user: search box, camera improvements, per-app battery usage
  - developer: expanded search framework, text-to-speech, gestures
- October, 2009: Eclair (2.0 / 2.1)
  - user: Exchange support, better keyboard, improved calendar
  - developer: improved graphics architecture, Bluetooth, revamped browser with HTML5 support

A Brief History

- May, 2010: Froyo (2.2)
  - user: portable hotspot, improved performance
  - developer: media framework with local and remote playback, two-way push sync, external storage and data backup
- December, 2010: Gingerbread (2.3)
  - user: word selection, manage running apps
  - developer: NDK (native code) expansion, gyroscope API
- January, 2011: Honeycomb preview (3.0)
  - tablet support

Setup

- required software
  - JDK: for writing Android Java applications and running all development tools
  - Eclipse: a freely available IDE with officially supported plugins for Android dev (http://eclipse.org/)
  - Android SDK: http://developer.android.com/sdk/installing.html
- refer to the Android Setup spec for more detailed instructions
  - https://www.cs76.net/Projects

Creating a New Project

- once Eclipse is set up, File $\rightarrow$ New $\rightarrow$Project $\rightarrow$Android Project
- give the project a unique name (no spaces, start with a capital letter), choose a target, and create a package name
  - traditional Java package naming conventions (e.g. `com.tommymacwilliam.awesome.app`)
- we can leave Min SDK Version blank or give the API Level number (e.g. 10, not 2.3.3)

System Architecture

- application (the apps themselves, including phone, email)
- application layer (views, content providers, resources)
- Dalvik VM and Java subset (core libraries)
- native, C/C++ libraries (WebKit, OpenGL, SQLite)
- Linux kernel (memory/process management, drivers, etc)

Packaging an App

- `javac` used to compile `.java` files to bytecode (`.class` files)
- `dx` used to convert Java bytecode into Dalvik bytecode
  - converts between the two instruction sets
- `aapt` used to create an Android package (`.apk`)
  - resulting archive is ZIP-compatible, like `.jar`
- Eclipse does all this for you!

Resources

- our `.apk` package also contains non-code data, like images
- the `res` folder in our project contains subdirectories for different types of files
  - `res/drawable`: images (PNG, JPG, etc.)
  - `res/layouts`: layouts (we'll see more of these soon!)
  - `res/values/strings.xml`: defined constants to be used by your app
- all resources can be accessed via a special, auto-generated class called `R`

Application Components

- activities: a single “screen” on your app
  - UI elements created within activies with views
- services: code without a visual component that runs in the background
- broadcast receivers: respond to system events like low battery
- content providers: make app data accessible to other apps

Activity Lifecycles

- an activity has one of 3 states: active, paused, or stopped
  - running: currently at the foreground and has focus
  - paused: lost focus but still visible to the user
  - stopped: no longer visible to the user
- activities not in the foreground can be stopped at any time (due to low memory)
  - don't rely on activities remaining in the paused state!

Activity Lifecycles

- a specific callback is fired for each state change
  - `onCreate()`: activity first starts up (used for initialization)
  - `onStart()`: activity is about to be displayed to the user
  - `onRestart()`: activity changes state from stopped to running
  - `onPause()`: activity changes state from running to paused
  - `onResume()`: activity changes state from paused to running
  - `onStop()`: activity changes state from paused to stopped
  - `onDestroy()`: process killed, either from stopped or paused state

Creating an Activity

- an activity has a Java file to define behaviors and an (optional) XML file to define layout
- to create the activity `Hello`, we first add a new `Hello.java` file to our package (right-click package → New → Class)
  - make sure we `import` the necessary Android packages and `extend` the `Activity` class
- now we add a new layout called `hello.xml` in `res/layouts` (right-click layout → New →File)
- finally, we add a new `<activity>` element to `AndroidManifest.xml` so our app knows about our new activity

Intents

- activities, services, and broadcast receivers are triggered via intents
  - an intent is just an object containing an operation to be performed
- `Intent i = Intent(Context packageContext, Class<?> cls);`
- with an intent, we can `startActivity(i)` to show a new activity

Layout

- each GUI element is represented by an XML element
  - attributes specify properties of the element
  - e.g. `android:layout_width` and `android:layout_height` can `fill_parent` or `wrap_content`
- GUI elements must be encapsulated in layouts
  - `FrameLayout`: contains a single element
  - `LinearLayout`: all elements arranged `horizontal`ly or `vertical`ly
  - `TableLayout`: just like HTML tables
  - `RelativeLayout`: elements arrange relative to other elements

Layout

- a handy list of GUI elements is visible on the left sidebar when you click the “Graphical Layout” tab
  - these elements can be dragged and dropped onto the screen
  - to edit XML attributes, right click the element

Event Handling

- just like HTML, we can assign a unique ID to GUI elements
  - this ID is actually numeric, but we give it a string name for ease of use
  - the numeric values of our IDs are stored in `R.id.<id>`
- using the `findViewById` method, we can get the object representing an element of our GUI
  - `findViewById` takes the numerical ID as its argument, not a string!
- we can then call methods like `setOnClickListener` to bind event handlers

Activities

- example time!

Activities and Tasks

- an app can consist of many activities
  - as users navigate among activities, the app maintains an internal stack of activities
- a task is a sequence of activities among potentially different apps
  - a user can proceed from one app to another to accomplish a task (e.g. emailing a photo)
  - using Intent filters (which we'll take a look at later), users can select the app they want to use to complete a step in a task
  - Android also supports multitasking: the user trying to do multiple, unrelated things at once

AVDs

- an Android Virtual Device is an emulator configuration that allows you to simulate a specific device
  - hardware profile: amount of memory on the simulated device, etc.
  - system image: what version of Android to run
  - dedicated storage: installed applications, settings, etc.
  - other options: screen dimensions, etc.
- from Eclipse: Window $\rightarrow$ Android SDK and AVD Manager

AVDs

- creating a new AVD from Eclipse: Window → Android SDK and AVD Manager → New
  - or, run `android` from `sdk/tools`
- specify what AVD to use when debugging from Eclipse: Run → Run Configurations → Android Application

AVDs

- example time!

Emulator

- running an Android project will start the emulator automatically
  - can also start the emulator by running `emulator -avd <avd name>`
- unlike iOS, the simulator startup is extremely SLOW
  - because we're actually simulating the hardware of the phone, which is slow
- if a simulator is already running, Eclipse will just use it
  - lesson: don't quit the simulator, or you'll have to wait for it to load again

ADB

- once we have an emulator up and running, we can connect to it using `adb` (`sdk/platform-tools`)
  - `adb devices` lists the emulators currently running
  - `adb shell` fires up a remote shell to the device
  - `adb pull <from> <to>` copies files from the device to the local filesystem
  - `adb push <from> <to>` copies files from your local filesystem to the device

ADB

- cool things we can do with the `adb shell`
  - `ls`: list the contents of the current directory
  - `pwd`: print the current directory
  - `cd`: change to a directory
  - `sqlite3`: examine a SQLite database associated with our application
  - `logcat`: display system logs (also results from `Log.i("tag", "string")`)
  - `monkey -p <package> -v <events>`: generage pseudo-random inputs to try to crash your application

ADB

- example time!

Telnet

- we can also telnet into the emulator to change configuration settings
  - connect with `telnet localhost <port>`, where `port` starts at 5554
- `power health <percent>`: set the simulated power of the device to `percent`
- `network speed <type>`: set the simulated network speed of the device
  - `<type>` can be `gsm`, `edge`, etc.
- `gsm call <number>`: simulate a phone call from `number`
- `sms send <number> <text>`: simulate a text message containing `text` from `number`
- `geo <longitude> <latitude> <altitude>`: fix the GPS coordinates at a `latitude`, `longitude`, and `altitude`

Telnet

- example time!