Android Basics

Xin Yang

2016-05-06
Outline of Lectures

• Lecture 1 (45mins) – Android Basics
  – Programming environment
  – Components of an Android app
  – Activity, lifecycle, intent
  – Android anatomy

• Lecture 2 (45mins) – Intro to Android Programming
  – Camera
  – 2D graphics drawing

• Lecture 3 (45mins) – Advanced Topics in Android Programming
  – Interacting with native code via JNI
  – Using opencv library in and Android project
  – Intro to Qualcomm SDK

• Lecture 4 (45mins) – Intro to Google APIs
  – Sensors
  – Animations
  – GPS
  – Google Maps
  – Etc.
Application Example: PhotoTag
Outline - Lecture 1

• Android Programming Basics
  – Walk through app development process via an example
  – Activity, lifecycle and intent

• Android Anatomy
  – Five layers of Android: application, application framework, native libraries, Android runtime and Linux kernel
Developer Site

http://developer.android.com/develop
Development Environment

- Android Studio: IDE based on IntelliJ IDEA
- Android SDK Tools includes:
  - SDK Manager
    - separates the SDK tools, platforms, and other components into packages for easy access and management
    - Android SDK Manager for downloading platforms, Google APIs, etc.,
  - AVD (Android Virtual Devices) Manager
    - provides a graphical user interface in which you can create and manage AVDs, which are required by the Android Emulator.
  - Emulator
  - Dalvik Debug Monitor Server
- A version of the Android platform to compile your app
- A version of the Android system image to run your app in the emulator
  - Supports processor architectures, e.g. ARM EABI, Intel X86 or MIPS
Creating a New Project with Android Studio

Fill in the following:

- **Application name**: app name that appears to users
- **Project name**: name of your project directory in your computer
- **Package name**: package namespace for your app, must be unique across all packages installed in Android system, same naming rules as packages in Java
- **Min SDK**: lowest version of Android that your app supports
- **Target SDK**: highest version of Android with which you have tested with your app
Creating a New Project with Android Studio

- Select: Minimum SDK (default)
- Android APIs can be referenced by:
  - Number: 1 - 21
  - Version: 1.0 - 5.0
- Name: Cupcake - Lollipop
Creating a New Project with Android Studio

- Leave defaults: Activity Parameters
Creating a New Project with Android Studio

- Select: Initial Activity (Blank)
- Activity: A screen with which users can interact in order to do something
Project Directory Structure

src: java code

res: resource files (layout, predefined text, multimedia data used in the app)

AndroidManifest.xml: present essential info of this app to Android system
Graphical User Interface (GUI)

- Android GUI is built using a hierarchy of **View** and **ViewGroup** objects
  - **View**: UI widgets (e.g. button, edit text fields)
  - **ViewGroup**: invisible view containers that define how the child views are laid out

Exemplar View objects

Illustration of how ViewGroup objects form branches in the layout
res/layout/activity_main.xml
Create a Layout in XML

**activity_main.xml file from res/layout/ directory**

```xml
<LinearLayout
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
>
    <TextView
        android:id="@+id/text1"
        android:text="@string/defaultText"
    />
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/photo"
    />
    <Button
        android:id="@+id/button"
        android:text="@string/buttonText"
    />
</LinearLayout>
```
Common ViewGroup Subclasses

- **LinearLayout**: all children are aligned in a single direction, horizontally or vertically
- **RelativeLayout**: Child object relative to each other
- **ListView**: a list of **scrollable** items
- **GridView**: displays items in **two-dimensional, scrollable** grid
Common View Subclasses

- TextView
- ImageView
- Button
- EditText
- Checkbox
- RadioGroup/RadioButton
- ToggleButton
- Spinner
src/com.example.myfirstapp/MainActivity.java
package com.example.myfirstapp;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
Android Manifest*

- Present essential information to Android system
  - Application package name
  - Components of the application
  - Which permissions the application requires
    - ex: camera, write to SDCard
  - Which permissions other applications required to interact with the app’s components
  - Minimum level of Android API
  - Required libraries

Android Manifest*

Unique Package ID

Uses Permissions

Components & their permissions
3. Run

• Run
  – On emulator
    • Create Android Virtual Device (AVD) first
    • AVD is a device configuration for Android emulator to model different devices
  – On devices
    • Connect your device to host machine through USB cable
Activity

• Activity
  – A screen that user sees on the device at one time
  – An app typically has multiple activities and the user flips back and forth among them
  – Each activity is given a window to draw its user interface
Activity Lifecycle

Transition States

Main Activity
Activity Lifecycle

Confirm or edit tagging results
Lifecycle

- 3 States
  - Resumed: Activity running with user focus
  - Paused: Another activity in foreground, but this is still visible
  - Stopped: Activity complete obscured and in the “background”
- In pause or stopped, activity is retained in memory
- But, the system can drop it from memory, if necessary
Callback Functions

- Android system creates new Activity instance by calling its `onCreate()` method
- You must implement `onCreate()` method to perform basic application startup logic

```java
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Set the user interface layout for this Activity
    // The layout file is defined in the project res/layout/main_activity.xml file
    setContentView(R.layout.main_activity);

    // Initialize member TextView so we can manipulate it later
    mTextView = (TextView) findViewById(R.id.text_message);

    // Make sure we're running on Honeycomb or higher to use ActionBar APIs
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        // For the main activity, make sure the app icon in the action bar
        // does not behave as a button
        ActionBar actionBar = getActionBar();
        actionBar.setHomeButtonEnabled(false);
    }
}
```
Callback Functions

- `onResume()` method is called every time when your activity comes into the foreground.

```java
@Override
public void onResume() {
    super.onResume(); // Always call the superclass method first

    // Get the Camera instance as the activity achieves full user focus
    if (mCamera == null) {
        initializeCamera(); // Local method to handle camera init
    }
}
```
Callback Functions

• `onPause()` method is usually used for
  – Stopping animations or other ongoing actions that could consume CPU
  – Committing unsaved changes
  – Release system resources, such as sensors, cameras, etc.

```java
@Override
public void onPause() {
    super.onPause();  // Always call the superclass method first

    // Release the Camera because we don't need it when paused
    // and other activities might need to use it.
    if (mCamera != null) {
        mCamera.release();
        mCamera = null;
    }
}
```
Callback Functions

• When the activity receives a call to `onStop()` method, it is no longer visible and should release almost all unnecessary resources.
• Compared to `onPause()`, `onStop()` performs larger, more CPU intensive shut-down operations, e.g. writing information to a database.

```java
@Override
protected void onStop() {
    super.onStop(); // Always call the superclass method first
    // Save the note's current draft, because the activity is stopping
    // and we want to be sure the current note progress isn't lost.
    ContentValues values = new ContentValues();
    values.put(NotePad.Notes.COLUMN_NAME_NOTE, getCurrentNoteText());
    values.put(NotePad.Notes.COLUMN_NAME_TITLE, getCurrentNoteTitle());
    getContentResolver().update(
        mUri, // The URI for the note to update.
        values, // The map of column names and new values to apply to them.
        null,  // No SELECT criteria are used.
        null   // No WHERE columns are used.
    );
}
```

Eg. saves the contents of a draft note to persistent storage.
Callback Functions

- **onStart()** is called every time your activity becomes visible
  - It is a good place to verify required system features are enabled

```java
@Override
protected void onStart() {
    super.onStart(); // Always call the superclass method first

    // The activity is either being restarted or started for the first time
    // so this is where we should make sure that GPS is enabled
    LocationManager locationManager =
        (LocationManager) getSystemService(Context.LOCATION_SERVICE);
    boolean gpsEnabled = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);

    if (!gpsEnabled) {
        // Create a dialog here that requests the user to enable GPS, and use an intent
        // with the android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS action
        // to take the user to the Settings screen to enable GPS when they click "OK"
    }
}
```
Activity Manager

• Launching an activity is quite expensive
  – Creating new Linux process
  – Allocating resources and memory for UI objects
  – Setting up the whole screen
  – Etc.

• It is wasteful to toss an activity out once user leaves that screen

• Activity manager manages activity lifecycle to avoid waste
Activity Manager: An Example

Activity Manager

Create activity and put it onto screen

Main Activity
Activity Manager: An Example

Activity Manager

Create activity and put it onto screen

Move

Holding place

Multi Activity

Activity
Activity Manager: An Example

Activity Manager

Move

Restart and bring it back onto screen

Main Activity

Holding place

Activity
Activity Manager: An Example

Activity Manager

Exit the app

Destroy

Main Activity

Activity

Holding place

Destroy

Take a photo
Show album
Show faces
Intent

- A messaging object which facilitates communication between activities

Intent

- **Explicit intents**: specify component to start by name. It is used to start component in your own app.
- **Implicit intents**: specify component by declaring general action to perform.

```
intent.setAction(Intent.ACTION_VIEW);
```

![Diagram showing how an implicit intent is delivered to start another activity](image)

**Fig. Illustration of how an implicit intent is delivered to start another activity**
Starting Activities in App

```java
public void sendMessage(View view) {
    Intent intent = new Intent(this, DisplayMessageActivity.class);
    EditText editText = (EditText) findViewById(R.id.edit_message);
    String message = editText.getText().toString();
    intent.putExtra(EXTRA_MESSAGE, message);
    startActivity(intent);
}
```

**public Intent (Context packageContext, Class<? extends Component> cls)**

Create an intent for a specific component. All other fields (action, data, type, class) are null, though they can be modified later with explicit calls. This provides a convenient way to create an intent that is intended to execute a hard-coded class name, rather than relying on the system to find an appropriate class for you; see `setComponent(ComponentName)` for more information on the repercussions of this.

**Parameters**

- **packageContext** A Context of the application package implementing this class.
- **cls** The component class that is to be used for the intent.

**See Also**

- `setClass(Context, Class)`
- `setComponent(ComponentName)`
- `Intent(String, android.net.Uri, Context, Class)`
Starting Activities in App

```java
public static final String EXTRA_MESSAGE = "com.example.myfirstapp.MESSAGE";

public void sendMessage(View view) {
    Intent intent = new Intent(this, DisplayMessageActivity.class);

    EditText editText = (EditText) findViewById(R.id.edit_message);
    String message = editText.getText().toString();
    intent.putExtra(EXTRA_MESSAGE, message);
    startActivity(intent);
}

public Intent putExtra(String name, String value) {
    // Implementation...
}
```

- name: many standardized types, but if defining own, include app's package name as prefix
Starting Activities in App

```java
public void sendMessage(View view) {
    Intent intent = new Intent(this, DisplayMessageActivity.class);
    EditText editText = (EditText) findViewById(R.id.edit_message);
    String message = editText.getText().toString();
    intent.putExtra("EXTRA_MESSAGE", message);
    startActivity(intent);
}
```

- Starts Activity based on Intent Parameters
Getting Intent Extra

```java
@override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Get Intent
    Intent intent = getIntent();
    String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);

    // Create the text view
    TextView textView = new TextView(this);
    textView.setTextSize(40);
    textView.setText(message);

    // Set the text view as the activity layout
    setContentView(textView);
}
```
Android Anatomy*

* http://sites.google.com/site/io/anatomy--physiology-of-an-android
Linux Kernel

• Android is built on the Linux kernel, but **Android is not Linux**
  – No glibc support
  – Does not include full set of standard Linux utilities
  – Android relies on Linux version 2.6 for core system services such as security, memory management, process management, etc.

• Kernel acts as an abstraction layer between hardware and the rest of the software stack
Android Anatomy*
Native Libraries

• Categorization
  – Bionic Libc
    • Custom libc implementation, optimized for embedded use
    • Small size and very fast
Native Libraries

• Categorization
  – Bionic Libc
  – Function Libraries
    • WebKit: web browser engine to render web pages
    • Media Framework: supports standard video, audio, still-frame formats
    • SQLite: light-weight transactional data store
Native Libraries

• Categorization
  – Bionic Libc
  – Function Libraries
  – Native Servers
    • Surface Manager: composes surfaces and hands surfaces to frame buffer devices
    • Audio Manager: manages all audio output devices
Native Libraries

• Categorization
  – Bionic Libc
  – Function Libraries
  – Native Servers
  – Hardware Abstraction Layer
    • Defines interface that Android requires hardware “drivers” to implement
    • Why it is needed?
      – Not all components have standardized kernel driver interfaces
      – Android has specific requirements for hardware drivers
Android Anatomy
Android Runtime

**Libraries**
- Surface Manager
- Media Framework
- SQLite
- OpenGL\|ES
- FreeType
- WebKit
- SGL
- SSL
- Libc

**Android Runtime**
- Core Libraries
- Dalvik Virtual Machine

**Linux Kernel**
- Display Driver
- Camera Driver
- Bluetooth Driver
- Shared Memory Driver
- USB Driver
- Keypad Driver
- WiFi Driver
- Audio Drivers
- Binder (IPC) Driver
- Power Management
Android Runtime

• Core Libraries
  – Provide most of the functionalities available in the core libraries of Java language → powerful, simple and familiar development platform
    • Data Structure
    • Utilities
    • File Access
    • Network Access
    • Graphics
    • ...

Android Runtime

• Dalvik Virtual Machine*
  – Provides Android apps portability and runtime consistency
  – Designed for embedded environment, uses runtime memory very efficiently
  – Convert Java .class/.jar files to .dex (Dalvik executable) at build time

Application Framework
Application Framework

- Contains all classes, cores and services that are used to build Android apps
- Categorization
  - Core platform services
  - Hardware services
Core Platform Services

- Services that are essential to the Android platform, e.g.
  - Manage application lifecycle, manage package, load resources
- Working behind the scenes
  - Applications don’t access/interrupt them directly
- Core platform services
  - Activity Manager
  - Package Manager
  - Window Manager
  - Resource Manager
  - Content Providers
  - View System
Hardware Services

• Telephony Service
• Location Service
• Bluetooth Service
• WiFi Service
• USB Service
• Sensor Service

More information
• At Google I/O
  – “Inside the Android Application Framework”
Hardware Services

• Provide access to lower-level hardware APIs
• Typically accessed through local Manager object

```java
LocationManager lm = (LocationManager)
    Context.getSystemService(Context.LOCATION_SERVICE)
```