Android Crash Course

CS 236503, Spring 2016

Dr. Oren Mishali
What is a Crash Course?

• “CRASH COURSE : a class in which a lot of information is taught in a short period of time” -- http://www.merriam-webster.com/
About Android

• A mobile operating system currently developed by Google
• Based on the Linux kernel
• Designed primarily for smartphones and tablets
  • Further developed for Android TV, Android Auto, and Android Wear
• The most popular mobile OS
Android’s Current Version

- Cupcake Android 1.5
- Donut Android 1.6
- Eclair Android 2.0/2.1
- Froyo Android 2.2.x
- Gingerbread Android 2.3.x
- Honeycomb Android 3.x
- Ice Cream Sandwich Android 4.0.x
- Jelly Bean Android 4.1.x
- KitKat Android 4.4.x
- Lollipop Android 5.0
- Marshmallow Android 6.0
How Android SHOULD NOT Be Studied

• Do the first “Hello World” example
• Then ask fellows from StackOverflow...
A Better Approach Would Be...

http://developer.android.com/

And other Resources
Have You Completed HW0?

• **Download and Install** Android Studio
• Upgrade the SDK using the SDK Manager
• Complete ‘**Building Your First App**’ Exercise,
  • and deploy on the **Genymotion** emulator
Android Studio

- Android Studio is the official Android development environment
  - It replaces Eclipse
- Based on the IntelliJ IDEA
An Android Project

• Contains all the files and resources that comprise an Android app
• A project is built into a single .apk file that is installed on a device
An Android Activity

• An application component that provides a screen with which users can interact in order to do something
  • E.g., dial the phone, take a photo, send an email, view a map
In More Depth
Create New Project Wizard
Create New Project Wizard (2)
Create New Project Wizard (3)
Let’s Review Some Key Files...

```xml
<manifest>
    <application>
        <activity>
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <activity android:name=".MyActivity" android:label="My First App" android:theme="@style/AppTheme.NoActionBar" />
        </activity>
    </application>
</manifest>
```
Gradle is used to compile and build the app.

A build.gradle file for each module in the project.

A build.gradle file for the entire project.

Usually we'll use the app module's build.gradle.

- **compiledSdkVersion** is the platform version against which you will compile your app, defaults to latest version available.
- **applicationId** is the fully qualified package name for your application.
- **minSdkVersion** is the earliest version of the Android SDK that the app supports.
- **targetSdkVersion** indicates the highest version of Android with which the app is tested.
The res/ folder

- drawable-<density>/
- layout/
- menu/
- mipmap/
- values/
Running Your App On a Real Device

• Enable **USB debugging** and connect your device
• **No Connected Device?** might need to **install a driver**
  • For Samsung devices, installing **KIES** may do the job
• Select the project and click **Run**
Running on the **GenyMotion** Emulator

- Install GenyMotion and add Virtual Device(s)
- Install the [GenyMotion plugin](https://www.genymotion.com) for Android Studio
- Start the Virtual Device and Run your app
Building a Simple User Interface

• The task
  • Main activity with text field and a button
  • Pressing the button displays the text in a second activity
Graphical User Interface of an Activity

• Hierarchy of **View** and **ViewGroup** objects
  • **View** – e.g., buttons, text fields
  • **ViewGroup** – containers defining the layout, e.g., **LinearLayout**

• Android provides a corresponding **XML vocabulary**
  • So UI can be defined in XML

• References
  • **View** Javadoc
  • **ViewGroup** Javadoc
Create a Linear Layout

- **content_my.xml**’s layout is modified from **RelativeLayout** to **LinearLayout**
- **LinearLayout** is a subclass of **ViewGroup**
  - Lays out child views in either a *vertical* or *horizontal* orientation

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:showIn="@layout/activity_my"/>
```
More on Layouts

“A layout defines the visual structure for a user interface”
Declaring a Layout

• A layout can be declared in two ways, via XML, or programmatically
  • By creating and manipulating View and ViewGroup objects at runtime
• A combined approach
  • Declaring a base XML layout, and manipulate programmatically
• Use XML whenever possible, better and more flexible design
XML Layout

• Each XML layout file must contain exactly one root element
  • Additional nested layout and widget elements may be added

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:orientation="vertical">
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView"/>
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button"/>
</LinearLayout>
```
Loading an XML Layout

• An XML layout file is compiled into a **View** resource
  • Which is loaded in `Activity.onCreate()`

```java
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```
The ID Attribute

- Any View object **may** have an integer ID associated with it, to *uniquely identify* the View within the tree

```xml
android:id="@+id/my_button"
```

- The @ symbol identifies the string as an ID resource
- The + symbol tells the parser it is a new resource name
  - Hence should be added to R.java
  - Is not needed when referencing previously-defined IDs
- Defining IDs is important when creating a **RelativeLayout**
The ID Attribute (2)

• A common pattern is to define a widget in the layout file with an id:

```xml
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
```

• And then retrieve its instance in `onCreate()`:

```java
Button myButton = (Button) findViewById(R.id.my_button);
```
Width and Height of a View

• Each view is required to define `layout_width` and `layout_height`
• It is possible to define exact width/height, however the use of these constants is more common
  • `wrap_content` tells the view to size itself to the dimensions required by its content
  • `match_parent` tells the view to become as big as its parent view group will allow
• When an exact width/height is needed
  • Do not use absolute units such as pixels
  • Use relative measurements such as density-independent pixel units (dp)
Common Layouts

• **LinearLayout**
  • Organizes its children into a single horizontal or vertical row
  • Creates a scrollbar if the length of the window exceeds the length of the screen

• **RelativeLayout**
  • Enables to specify the location of child objects relative to each other or to the parent

• **WebView**
  • Displays web pages

Tip: for better performance, keep layout hierarchy as shallow as possible
Layout Weight

- LinearLayout supports layout_weight
- Remaining space is assigned to children in the proportion of their weight
- Default weight is zero

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```
Relative Layout

- A very powerful utility for designing a user interface
- Many nested **LinearLayouts**?
  - Consider **RelativeLayout** instead
- By default, all child views are drawn at the top-left of the layout
  - So the position of each view must be defined using the various layout properties

See **RelativeLayout.LayoutParams** for all attributes available

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp">
    <EditText
        android:id="@+id/name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Spinner
        android:id="@+id/dates"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />
    <Spinner
        android:id="@+id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/times"
        android:layout_alignParentRight="true"
        android:text="@string/done"/>
</RelativeLayout>
```
Back to “Hello World” Example

We have just modified the layout to horizontal LinearLayout...
Add a Text Field

```xml
<EditText android:id="@[id/edit_message"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="@[string/edit_message"
    />
```

Note: no name collisions

• What is the meaning of `@string/edit_message`?
• How to resolve the compilation error?
String Resources

- **Always** specify UI strings as resources
  - Allow to manage all UI text in a single location
  - Straightforward **localization**
Add a Button

• Why no `android:id` for the button?
• Should the resulting layout be improved?

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="android.support.design.widget.ShowIn"
    tools:showIn="@layout/activity_my">

    <EditText android:id="@+id/edit_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="Enter a message" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send" />

</LinearLayout>
```
• Note the use of 0dp in `layout_weight` for improved layout performance
  • Since calculation of `wrap_content` is not needed
Starting Another Activity – **onClickListener** event

- The task: start a new activity when user presses the button

```xml
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
</LinearLayout>
```
Creating a Java Method Handler

- The method’s signature must be
  - public
  - void
  - single View Parameter

```java
/** Called when the user clicks the Send button */
public void sendMessage(View view) {
    // Do something in response to button
}
```
Build an Intent

• Note the use of *this* as first C’tor’s **Context** parameter
  • Second parameter is the **class** of the target Activity
• An **Intent** can carry data types as key-value pairs called **extras**
Create the Second Activity

- **Hierarchical Parent** adds Up Navigation (API 16+)
- The `<meta-data>` element adds support for older versions
Get the Intent & Display the Message

- Every Activity is invoked by an Intent
- Note how the layout is created programmatically
Supporting Different Devices

Different Languages, Different Screen Sizes, Different Android Versions
The Basic Concept

• Leverage alternative resources to support a variety of Android-compatible devices
  • Within a single APK (application package)
Supporting Different Languages (Localization)

• Localization support is based on keeping all UI strings in a single file `res/values/strings.xml`

• To support more languages, create more `values-<ISO_code>` folders
  • At runtime, appropriate strings are used based on device’s language settings
  • This method applies to other folders as well, e.g., the `drawable` folder

```xml
-values/strings.xml
  <resources>
    <string name="title">My Application</string>
    <string name="hello_world">Hello World!</string>
  </resources>

-values-es/strings.xml
  <resources>
    <string name="title">Mi Aplicación</string>
    <string name="hello_world">Hola Mundo!</string>
  </resources>
```
Enhanced Android Studio Support

• Focus on creating the resource, the folder is created for you
How to Use the String Resources?

• From source code, use `R.string.<string_name>`

```java
// Get a string resource from your app's Resources
String hello = getResources().getString(R.string.hello_world);

// Or supply a string resource to a method that requires a string
TextView textView = new TextView(this);
textView.setText(R.string.hello_world);
```

• In XML files, use `@string/<string_name>`

```xml
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
```
Supporting Different Screens

• App may be installed on devices with different screen size and density
  • We may be interested in supplying alternative layouts and bitmaps to improve app’s appearance
  • Same principles as in string localization

• 4 generalized sizes: small, normal, large, xlarge
• 4 generalized densities: low (ldpi), medium (mdpi), high (hdpi), extra high (xhdpi)
Example: Creating Different Layouts

Alternative layout for large screens:

```xml
MyProject/
  res/
    layout/
      main.xml
    layout-large/
      main.xml
```

Alternative layout for landscape orientation:

```xml
MyProject/
  res/
    layout/
      main.xml
    layout-land/
      main.xml
```

- Again, system automatically loads the appropriate files
- By default, XMLs under `layout` are used for portrait orientation
- Suffixes may be combined, e.g., `layout-large-land`
  - Android Studio makes the task easier 😊
Creating Different Bitmaps

• For good graphical quality, provide bitmap resources for the different density buckets

• Use the following size scale
  • xhdpi: 2.0
  • hdpi: 1.5
  • mdpi: 1.0 (baseline)
  • ldpi: 0.75

• For example, if a 100x100 image is used for mdpi, then 200x200 image should be generated for xhdpi

• XML reference: @drawable/awesomeimage
Supporting Different Platform Versions

- We want to take advantage of the latest APIs while continuing to support older versions.
- Good practice – support about 90% of the active devices, while targeting your app to the latest version.

<table>
<thead>
<tr>
<th>Version</th>
<th>Codename</th>
<th>API</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2</td>
<td>Froyo</td>
<td>8</td>
<td>0.1%</td>
</tr>
<tr>
<td>2.3.3-2.3.7</td>
<td>Gingerbread</td>
<td>10</td>
<td>2.6%</td>
</tr>
<tr>
<td>4.0.3-4.0.4</td>
<td>Ice Cream Sandwich</td>
<td>15</td>
<td>2.3%</td>
</tr>
<tr>
<td>4.1.x</td>
<td>Jelly Bean</td>
<td>16</td>
<td>8.1%</td>
</tr>
<tr>
<td>4.2.x</td>
<td></td>
<td>17</td>
<td>11.0%</td>
</tr>
<tr>
<td>4.3</td>
<td></td>
<td>18</td>
<td>3.2%</td>
</tr>
<tr>
<td>4.4</td>
<td>KitKat</td>
<td>19</td>
<td>34.3%</td>
</tr>
<tr>
<td>5.0</td>
<td>Lollipop</td>
<td>21</td>
<td>16.9%</td>
</tr>
<tr>
<td>5.1</td>
<td></td>
<td>22</td>
<td>19.2%</td>
</tr>
<tr>
<td>6.0</td>
<td>Marshmallow</td>
<td>23</td>
<td>2.3%</td>
</tr>
</tbody>
</table>

http://developer.android.com/about/dashboards/index.html

Data collected during a 7-day period ending on March 7, 2016. Any versions with less than 0.1% distribution are not shown.
Android Support Library

• Is a set of code libraries that provide backward-compatible versions of Android framework APIs

• Including the Support Libraries in your Android project is considered a best practice

• Each support library targets a base Android API level and provides a different set of features

• For effective library usage, consider what features you want to support
  • Consult Support Library Features