UNIVERSITY *of* WASHINGTON

Android 210 - Lecture 2 Activity Lifecycle, Logging & Android UI Part 1

Margaret Maynard-Reid
January 12, 2015

Topics for Today

- Activity Lifecycle
- Logging & Debugging
- Android UI basics - part 1
    - Views & ViewGroups (layouts)
    - How to create UI in Android
    - Commonly used views and layouts
- Git Version Control
- Homework 1 Requirements

Android Stories

- Chinese Mobile App UI Trends
- Latest Android version numbers show no sign of Lollipop, but Froyo is still kicking
- Fying wearable by Intel at CES
- CES takeaway: Android Auto

Review from last week

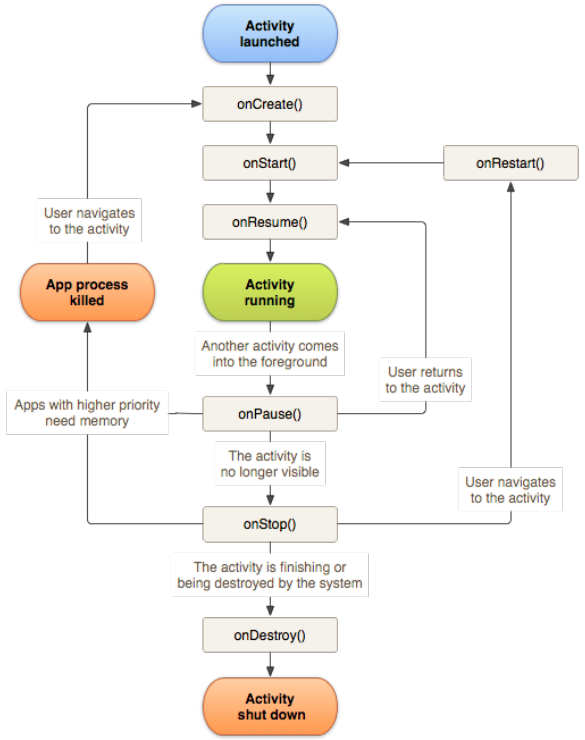
- Roughly how many Android devices are out there?
- What is the latest Android version?
- What are the 4 components of Android?
- What makes up an Android app?
- What is an intent?

Review from last class

- **Activity**: an interactive component that fills a user need, typically a screen
- **Service**: an Application background task, no UI
- **Content Provide**: a cross-process data sharing mechanism
- **Intent Receiver**: an Application callback for intents

Activity Lifecycle

- onStart()
- onResume()
- onCreate()
- onPause()
- onStop()
- onDestroy()


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Logging

- Log.v - verbose
- Log.d - debug
- Log.i - information
- Log.w - warning
- Log.e - error

Logcat

- The output from adb logcat shows the logcat view in DDMS.
- Output can be filtered, ie. change from verbose to error.

Sample Code

- Walk through SampleActivity.java
  - Logging the Activity lifecycle
  - Android Device Monitor, DDMS

Update build.gradle file

- Update build.gradle file under app module
- Debug vs. release build
- Add dependencies (i.e. Android support library v7)

Break

Android UI

UI topics to be covered tonight:

- Views and ViewGroups
- View terminologies:
    - margins vs padding
    - dp vs. sp
    - visibility
- Commonly used views
- Commonly used layouts

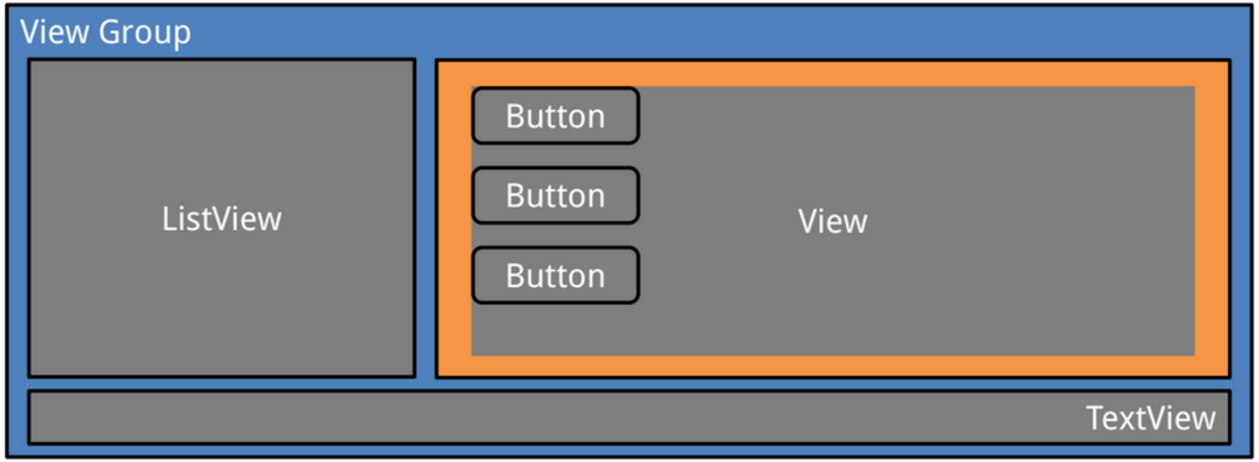
View and ViewGroups

- Views: an UI element such as
  - TextView
  - Imageview
  - Button
- ViewGroups: containers for positioning the views, such as
  - LinearLayout
  - RelativeLayout
  - FrameLayout

Margin vs. Padding

Margin - space outside the view

Padding - space inside the view



Layout Gravity vs. Gravity

Layout Gravity - Sets the placement of the current View in its parent container

Gravity - Sets the placement of the current View's contents

View visibility

- **Visible** - visible in UI
- **Invisible** - invisible in UI but still occupying the space in layout
- **Gone** - not visible and not taking up space

Units of Measurement

- ***dp*** - density independent pixels. ← use this for layouts and views
- ***sp*** - scale-independ pixels. ← use this for font size
- ***px*** - pixels on screen. ← not recommended for android
- More details here:
  http://developer.android.com/guide/topics/resources/more-resources.html#Dimension

How to Create UI in Android?

There are three ways to create the UI:

- By XML
- By Code
- Hybrid

UI By XML

1. Define a UI in a XML Layout File: main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="fill_parent"
 android:layout_height="fill_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>
```

UI By XML

2. Then set the Content View of your Activity

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
}
```

UI by XML - why?

- Resource resolution
- Keeping view and business logic separate
- Designers can easily design with xml

UI By Code

Define content and views purely in code

```
private LinearLayout rootLinearLayout;

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(rootLinearLayout);
    rootLinearLayout.addView(new TextView());
    rootLinearLayout.addView(new Button());
}
```

UI with Hybrid (XML & Code)

- Reference and inflate XML layouts in code:

```
public void onCreate(Bundle savedInstanceState) {
    TextView textView = (TextView) this.findViewById(R.
    id.text);
    textView.setText("Hello, I am a TextView");
}
```

- When would this approach make sense?

What is XML?

- XML is a structured document or data format. The structure is defined by XML elements.
- An element tag is started using angle <> brackets, and ended using a forward-slash with angle brackets</>

XML

- Has a start & end element
- Each element can contain an attribute
- Namespace defines globally unique names for the elements
- XML elements can be nested
- Element can be closed with />
- Each XML file contains 1 root element

Notations to access resources

- Android specific notation used within resource files (not xml)
- @drawable - reference a drawable in res/drawable @String - reference a string in res/values/strings.xml
- @+id - assign an id
- @id - reference an id

TextView

- Used to display some text. Can be used as a label.
- Text value can be set programmatically

Hands-on:

- Add an id to the TextView in the layout as *android:id="@+id/textview1"*
- In the MainActivity.java onCreate function, add the following lines after setContentView

```
TextView textView = (TextView) findViewById(R.id.textview1);
textView.setText("Welcome to Android");
```

Note the text changed from “Hello World” to “Welcome to Android”

- Can change various values such as textSize and textColor...

TextAppearance

- TextAppearanceLarge - 22sp
- TextAppearanceMedium - 18sp
- TextAppearanceSmall - 14sp

ImageView

- Displays an image
- Can load image from resources or content provider
- Can set scaleType

Button

- Derived from TextView
- Can change textSize and color as TextView
- Responds to clicks
- To place an image in button, use drawableTop, drawbleLeft etc.

ImageButton

- A subclass of **ImageView**
- Mixes in the **Button** behavior - responds to clicks
- Cannot set text on ImageButton

Sample Code

- Walk through SampleUI.java for the various Views

Break

Android UI Design Patterns

- The Official Android Developer Design Guide
- Android Niceties
- Android UI Patterns
- Mobile Patterns - for both Android and iOS

Android UI Tools

- Android Assets Studio
- Demo: update launch icon for all drawable folders
- Demo: create a generic icon

Layouts (Containers)

ViewGroup extends from **View** class

Layouts are ViewGroups

- They contain other Views (children)
- They lay them out in a particular way
- Often defined as a resource

Commonly Used Layouts

- **LinearLayout** - Displays children arranged in a single direction (vertically or horizontally)
- **RelativeLayout** - Displays children relative to the parent and other children
- **FrameLayout** - Display children aligned to the top left of the layout

LinearLayout - when to use it?

- Choose LinearLayout when you want to position the children views **vertically** or **horizontally**
- Choose LinearLayout when you want the children views to take up space **proportionally**

LinearLayout: watchout!

Invisible items in a LinearLayout

- The layout orientation is horizontal by default
- Make sure the android:orientation is correctly set as

horizontal or vertical

LinearLayout - layout weight

- Determines the proportion of the children views
- By default it’s assigned as zero
- LinearLayout is the **only** container that allows layout weight

RelativeLayout

- Choose RelativeLayout when you want the flexiblity in positioning the child views

FrameLayout

- Use FrameLayout for a **single child view**
- Default position is **top-left corner**, use gravity attribute to alter view location
- Can stack multiple children **on top of each other**

Layout in Landscape

- Create a new folder called ***layout-land*** under ***/res***
- Create a new file called ***activity_main.xml***, same as the one under ***/res/layout/*** ← note the file name must be the same
- Copy the xml from the **activity_main.xml** under the **/layout** folder to **/layout-land** and make some changes
- Run app, rotate screen and observe the differences

Sample Code

- SampleUI.java demos layouts

Homework 1

- Requirements have been posted on Catalyst Dropbox.
- Homework 1 due by **Monday, Jan 26 6PM**
- We will review solution on Jan 26
- No late homework is accepted!