Announcement

- No class next week
- I’m out of town for a grant meeting
- Next class in 2 weeks (Sept 27)
Android App Components
Android App Components

- Typical Java program starts from main()

```
class SillyApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

- Android app: No need to write a main
- Just define app components derived from base classes already defined in Android
Android App Components

- 4 main types of Android app components:
  - Activity (already seen this)
  - Service
  - Content provider
  - Broadcast receiver
Recall: Activities

- Activity: main building block of Android UI
- Analogous to a window or dialog box in a desktop application
- Apps
  - have at least 1 activity that deals with UI
  - Entry point of app similar to \texttt{main()} in C
  - typically have multiple activities
- Example: A camera app
  - **Activity 1:** to focus, take photo, start activity 2
  - **Activity 2:** to present photo for viewing, save it
Fragments

- Fragments
  - UI building blocks (pieces), can be arranged in Activities in different ways.
  - Enables app to look different on different devices (e.g. phone vs tablet)

- An activity can contain multiple fragments that are organized differently on different devices (e.g. for phone vs tablet)

- More later
Services

- Activities are short-lived, can be shut down anytime (e.g. when user presses back button)
- Services keep running in background
- Similar to Linux/Unix CRON job
- Example uses of services:
  - Periodically check/update device’s GPS location
  - Check for updates to RSS feed
- Independent of any activity, minimal interaction
- Typically an activity will control a service -- start it, pause it, get data from it
- Services in an App are sub-class of Android’s Services class
Android Platform Services

- Android Services can either be on:
  - On smartphone or Android device (local)
  - Remote, on Google server/cloud

- Android platform local services examples (on smartphone):
  - **LocationManager**: location-based services.
  - **ClipboardManager**: access to device’s clipboard, cut-and-paste content
  - **DownloadManager**: manages HTTP downloads in background
  - **FragmentManager**: manages the fragments of an activity.
  - **AudioManager**: provides access to audio and ringer controls.
Google Services (In Google Cloud)

- Maps
- Location-based services
- Game Services
- Authorization APIs
- Google Plus
- Play Services
- In-app Billing
- Google Cloud Messaging
- Google Analytics
- Google AdMob ads

Typically need Internet connection

Android services on smartphone

Android services in Google cloud
Content Providers

- Android apps can share data (e.g. User’s contacts) as content provider

- Content Provider:
  - Abstracts shareable data, makes it accessible through methods
  - Applications can access that shared data by calling methods for the relevant **content provider**
  - E.g. Can query, insert, update, delete shared data (see below)
Content Providers

- **E.g.** Data stored in Android Contacts app can be accessed by other apps.
- **Example:** We can write an app that:
  - Retrieve’s contacts list from contacts content provider
  - Adds contacts to social networking (e.g. Facebook)
- Apps can also **ADD** to data through content provider. E.g. Add contact
- E.g. Our app can also share its data
- Content provider in an App are sub-class of Android’s **ContentProvider** class
Broadcast Receivers

- Android OS (system), or applications, periodically broadcasts events
- Example broadcasts:
  - Battery getting low
  - Download completed
  - New email arrived
- Any app can create broadcast receiver to listen for broadcasts, respond
- Our app can also initiate broadcasts
- Broadcast receivers typically
  - Doesn’t interact with the UI
  - Creates a status bar notification to alert the user when broadcast event occurs
- Broadcast Receiver in an App are sub-class of Android’s `BroadcastReceiver` class
Quiz

- Pedometer App has the following Android components:
  - **Component A**: Continuously counts user’s steps even when user closes app, does other things on phone (e.g. YouTube, calls)
  - **Component B**: Displays user’s step count
  - **Component C**: Texts user’s friends (from contacts list) every day with their step totals

- What should component A be declared as?
  - Activity, service, content provider, broadcast receiver?

- What of component B?

- Component C?
Android Activity LifeCycle
Starting Activities

- Android Activity callbacks invoked corresponding to app state.

Examples:
- When activity is created, its `onCreate()` method invoked (like constructor)
- When activity is paused, its `onPause()` method invoked
Activity Callbacks

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

Android OS invokes specific callbacks when specific events occur

IMPORTANT: Android OS invokes all callbacks!!

Already saw this (initially called)
Understanding Android Lifecycle


- Many **disruptive** things could happen while app is running
  - Incoming call or text message, user switches to another app, etc

- Well designed app should NOT:
  - Crash if interrupted, or user switches to other app
  - Lose the user's state/progress (e.g. state of chess game app) if they leave your app and return later
  - Crash or lose the user's progress when the screen rotates between landscape and portrait orientation.
    - E.g. Youtube video should continue at correct point after rotation

- To handle these situations, appropriate callback methods must be invoked appropriately to “tidy up” before app gets bumped
OnCreate( )

- Initializes activity once created
- Operations typically performed in onCreate() method:
  - Inflate (create) widgets and place them on screen
    - (e.g. using layout files with setContentView( ) )
  - Getting references to inflated widgets ( using findViewById( ) )
  - Setting widget listeners to handle user interaction
- E.g.

```java
public class QuizActivity extends Activity {
    private Button mTrueButton;
    private Button mFalseButton;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);

        mTrueButton = (Button)findViewByld(R.id.true_button);
        mFalseButton = (Button)findViewByld(R.id.false_button);
    }
}
```

- Note: Android OS calls apps’ onCreate( ) method
Running App

- A running app is one that user is currently using or interacting with
  - Visible, in foreground
Paused App

- An app is **paused** if it is **visible but no longer in foreground**
- E.g. blocked by a pop-up dialog box
- App’s `onPause()` method is called during transition from running to paused state
onPause( ) Method

- Typical actions taken in onPause( ) method
  - Stop animations or CPU intensive tasks
  - Stop listening for GPS, broadcast information
  - Release handles to sensors (e.g. GPS, camera)
  - Stop audio and video
onResume( ): ResumingPaused App

- A **paused** app resumes **running** if it becomes fully visible and in foreground
  - E.g. pop-up dialog box blocking it goes away
- App’s **onResume( )** method is called during transition from **paused** to **running** state
  - Restart videos, animations, GPS checking, etc
**Stopped App**

- An app is **stopped** if it’s **no longer visible + no longer in foreground**
- E.g. user starts using another app
- App’s **onStop()** method is called during transition from paused to stopped state
**onStop() Method**

- An activity is stopped when:
  - User receives phone call
  - User starts another app
- Activity instance and variables of stopped app are retained but no code is being executed by the activity
- If activity is stopped, in `onStop()` method, well behaved apps should
  - save progress to enable seamless restart later
  - Release all resources, save info (persistence)
Resuming Stopped App

- A **stopped** app can go back into **running** state if becomes visible and in foreground
- App’s **onStart()** and **onResume()** methods called to transition from **stopped** to **running** state
Starting New App

- To launch new app, get it to running
- App’s `onCreate()`, `onStart()` and `onResume()` methods are called
- Afterwards new app is running
Logging Errors in Android
Logging Errors in Android

- Android can log and display various types of errors/warnings in Android Studio Window.

- Error logging is in `Log` class of `android.util` package, so need to import `android.util.Log`;

- Turn on logging of different message types by calling appropriate method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log.e()</td>
<td>Log errors</td>
</tr>
<tr>
<td>Log.w()</td>
<td>Log warnings</td>
</tr>
<tr>
<td>Log.i()</td>
<td>Log informational messages</td>
</tr>
<tr>
<td>Log.d()</td>
<td>Log debug messages</td>
</tr>
<tr>
<td>Log.v()</td>
<td>Log verbose messages</td>
</tr>
</tbody>
</table>

Ref: Introduction to Android Programming, Annuzzi, Darcey & Conder
QuizActivity.java

- A good way to understand Android lifecycle methods is to print debug messages in Android Studio when they are called.

```java
onCreate( ){
    ... print message “OnCreate called”...
}

onStart( ){
    ... print message “OnStart called”...
}

... etc
```
QuizActivity.java

- Example: print debug message from onCreate method below

```java
package com.bignerdranch.android.geoquiz;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;

public class QuizActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);
    }
}
```
QuizActivity.java

- Debug (d) messages have the form
  
  ```java
  public static int d(String tag, String msg)
  ```

- E.g.
  
  ```java
  Tag  Message
  QuizActivity: onCreate(Bundle) called
  ```

- Example declaration:
  
  ```java
  Log.d(TAG, "onCreate(Bundle) called");
  ```

- Then declare string for **TAG**
  
  ```java
  public class QuizActivity extends Activity {
      private static final String TAG = "QuizActivity";
      ...
  }
  ```
public class QuizActivity extends Activity {

    ... 

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, "onCreate(Bundle) called");
        setContentView(R.layout.activity_quiz);
        ...
QuizActivity.java

- Can override more lifecycle methods
- Print debug messages from each method

```java
} // End of onCreate(Bundle)

@Override
public void onStart() {
    super.onStart();
    Log.d(TAG, "onStart() called");
}

@Override
public void onPause() {
    super.onPause();
    Log.d(TAG, "onPause() called");
}

@Override
public void onResume() {
    super.onResume();
    Log.d(TAG, "onResume() called");
}

@Override
public void onStop() {
    super.onStop();
    Log.d(TAG, "onStop() called");
}

@Override
public void onDestroy() {
    super.onDestroy();
    Log.d(TAG, "onDestroy() called");
}
```
QuizActivity.java Debug Messages

- Launching GeoQuiz app activities **OnCreate, OnStart and onResume** methods

- Pressing **Back** button destroys the activity (calls **onPause, onStop and onDestroy**)
Rotating Device
Rotating Device: Using Different Layouts

- Rotating device (e.g. portrait to landscape) kills current activity and creates new activity in landscape mode
- Rotation changes **device configuration**
- **Device configuration**: screen orientation/density/size, keyboard type, dock mode, language, etc.
- Apps can specify different resources (e.g. XML layout files, images) to use for different device configurations

- E.g. use different app layouts for portrait vs landscape screen orientation
Rotating Device: Using Different Layouts

- **Portrait**: use XML layout file in `res/layout`
- **Landscape**: use XML layout file in `res/layout-land/`
- Copy XML layout file (activity_quiz.xml) from `res/layout` to `res/layout-land/` and customize it
- If configuration changes, current activity destroyed, `onCreate -> setContentView (R.layout.activity_quiz)` called again
Dead or Destroyed Activity

- `onDestroy()` called to destroy a stopped app
Saving State Data
Activity Destruction

- App may be destroyed
  - On its own by calling finish
  - If user presses **back button**
- Before Activity destroyed, system calls `onSaveInstanceState`
- Can save state required to recreate Activity later
  - E.g. Save current positions of game pieces
onSaveInstanceState: Saving App State

- Systems write info about views to Bundle
- Programmer must save other app-specific information using `onSaveInstanceState()`
  - E.g. board state in a board game such as mastermind
onRestoreInstanceState(): Restoring State Data

- When an Activity recreated saved data sent to **onCreate** and **onRestoreInstanceState()**
- Can use either method to restore app state data
Saving Data Across Device Rotation

- Since rotation causes activity to be destroyed and a new one created, values of variables lost or reset.
- To avoid losing or resetting values, save them using `onSaveInstanceState` before activity is destroyed.
  - E.g., called before portrait layout is destroyed.
- System calls `onSaveInstanceState` before `onPause()`, `onStop()` and `onDestroy()`.
Saving Data Across Device Rotation

- For example, to save the value of a variable `mCurrentIndex` during rotation
- First, create a constant `KEY_INDEX` as a key for storing data in the bundle

```java
private static final String KEY_INDEX = "index";
```

- Then override `onSaveInstanceState` method

```java
@override
public void onSaveInstanceState(Bundle savedInstanceState) {
    super.onSaveInstanceState(savedInstanceState);
    Log.i(TAG, "onSaveInstanceState");
    savedInstanceState.putInt(KEY_INDEX, mCurrentIndex);
}
```
Question

- Whenever I watch YouTube video on my phone, if I receive a phone call and video stops at 2:31, after call, when app resumes, it should restart at 2:31.

- How do you think this is implemented?
  - In which Android methods should code be put into?
  - How?
References

- Busy Coder’s guide to Android version 4.4
- CS 65/165 slides, Dartmouth College, Spring 2014
- CS 371M slides, U of Texas Austin, Spring 2014