CS 4518 Mobile and Ubiquitous Computing
Lecture 6: Android Activity Lifecycle and Intents

Emmanuel Agu
Android Activity LifeCycle
Starting Activities

- Android applications don't start with a call to main(String[])
- Instead 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
- callback methods also invoked to destroy Activity /app
Activity Callbacks

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

http://developer.android.com/training/basics/activity-lifecycle/starting.html
OnCreate() 

- Initializes activity once created 
- Operations typically performed in onCreate() method:
  - Inflate 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
- Example

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

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

- A running app is one that user is currently using or interacting with
  - Visible, in foreground
Activity State Diagram: 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
Activity State Diagram: 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 if appropriate
Activity State Diagram: Resuming Paused 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
An app is **stopped** if it **no longer visible and 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 a new application
  - Activity 1 launches new Activity 2
- 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)
Activity State Diagram: 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
Activity State Diagram: Starting New App

- To start new app, app is launched
- 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.
- Error logging is in `Log` class of `android.util` package.
  ```java
  import android.util.Log;
  ```
- Turn on logging of different message types by calling appropriate method.
- Logged errors/warnings displayed in Android Studio window.

<table>
<thead>
<tr>
<th>Method</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Log.e()</code></td>
<td>Log errors</td>
</tr>
<tr>
<td><code>Log.w()</code></td>
<td>Log warnings</td>
</tr>
<tr>
<td><code>Log.i()</code></td>
<td>Log informational messages</td>
</tr>
<tr>
<td><code>Log.d()</code></td>
<td>Log debug messages</td>
</tr>
<tr>
<td><code>Log.v()</code></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 when they are called
- E.g. 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

  ```
  public static int d(String tag, String msg)
  ```

- E.g.

  ```
  Log.d(TAG, "onCreate(Bundle) called");
  ```

- Then declare string for TAG

  ```
  public class QuizActivity extends Activity {
      private static final String TAG = "QuizActivity";
      ...
  }
  ```
QuizActivity.java

- Putting it all together

```java
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
- Superclass calls called in 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 **creates, starts and resumes** an activity.

- 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

- When device in landscape, uses layout (XML) file in res/layout-land/
- Copy XML layout file (activity_quiz.xml) from res/layout to res/layout-land/ and tailor it
- When 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
- Saves state required to recreate Activity later
  - E.g. Save current positions of game pieces
onSaveInstanceState
onRestoreInstanceState()

- Systems write info about views to Bundle
- other (app-specific) information must be saved by programmer
  - E.g. board state in a board game such as mastermind
- When Activity recreated Bundle sent to **onCreate** and **onRestoreInstanceState()**
- Can use either method to restore state data / instance variables
Saving State on Activity Destruction

1. onSaveInstanceState()
2. onCreate()
3. onRestoreInstanceState()

Can restore state data in either method.
Saving Data Across Device Rotation

- Since rotation causes activity to be destroyed and a new one created, values of variables lost or reset.
- To stop lost or reset 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, if we want to save the value of a variable `mCurrentIndex` during rotation
- First, create a constant 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);
}
```
Quiz

- 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 Activity life cycle method should code be put into?
  - How?
Intents
**Intent**

- **Intent**: a messaging object used by a component to request action from another app or component
- 3 main use cases for Intents
- **Case 1 (Activity A starts Activity B, no result back):**
  - Call `startActivity()`, pass an Intent
  - Intent describes Activity to start, carries any necessary data
Intent: Result Received Back

- **Case 2 (Activity A starts Activity B, gets result back):**
  - Call `startActivityForResult()` , pass an Intent
  - Separate Intent received in Activity A’s `onActivityResult()` callback

- **Case 3 (Activity A starts a Service):**
  - E.g. Activity A starts service to download big file in the background
  - Activity A calls `StartService()` , passes an Intent
  - Intent describes Service to start, carries any necessary data
Implicit Vs Explicit Intents

- **Explicit Intent:** If components sending and receiving Intent are in same app
  - E.g. Activity A starts Activity B in same app

- **Implicit Intent:** If components sending and receiving Intent are in different apps
Intent Example: Starting Activity 2 from Activity 1
Allowing User to Cheat
Ref: Android Nerd Ranch (2nd edition) pg 87

- **Goal**: Allow user to cheat by getting answer to quiz
- Screen 2 pops up to show Answer

![Diagram showing screens and user interactions](image-url)
Add Strings for Activity 1 and Activity 2 to strings.xml

```xml
<string name="question_asia">Lake Baikal is the world's oldest and deepest freshwater lake.</string>
<string name="warning_text">Are you sure you want to do this?</string>
<string name="show_answer_button">Show Answer</string>
<string name="cheat_button">Cheat!</string>
<string name="judgment_toast">Cheating is wrong.</string>
```
Create Blank Activity (for Activity 2) in Android Studio
Specify Name and XML file for Activity 2

Screen 2 Java code in CheatActivity.java
Layout uses activity_cheat.xml
Design Layout for Screen 2

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
>
    <TextView
        android:id="@+id/answerTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/warning_text"
    />
    <TextView
        android:id="@+id/showAnswerButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/show_answer_button"
    />
    <Button
        android:id="@+id/showAnswerButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/show_answer_button"
    />
</LinearLayout>
```
Write XML Layout Code for Screen 2

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    tools:context="com.bignerdranch.android.geoquiz.CheatActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/warning_text"/>

    <TextView
        android:id="@+id/answer_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="Answer"/>

    <Button
        android:id="@+id/show_answer_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/show_answer_button"/>

</LinearLayout>
```

Activity 2
Declare New Activity in AndroidManifest.xml

- Create new activity (CheatActivity) in Android Studio

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.beginnerbranch.android.geoquiz">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">

        <activity
            android:name=".QuizActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
            </intent-filter>
        </activity>

        <activity
            android:name=".CheatActivity"
            android:label="@string/title_activity_cheat" />
    </application>

</manifest>
```
Starting Activity 2 from Activity 1

- Activity 1 starts activity 2
  - through the Android OS
  - by calling `startActivity(Intent)`

- Passes Intent (object for communicating with Android OS)

- Intent specifies which (target) Activity Android ActivityManager should start
Starting Activity 2 from Activity 1

- Intents have many different constructors. We will use form:

```java
public Intent(Context packageContext, Class<? extends Activity> cls)
```

- Actual code looks like this

```java
mCheatButton = (Button) findViewById(R.id.cheat_button);
mCheatButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Start CheatActivity
        Intent i = new Intent(QuizActivity.this, CheatActivity.class);
        startActivity(i);
    }
});
...
Implicit vs Explicit Intents

- Previous example is called an **explicit intent**
  - Activity 1 and activity 2 are in same app
- If Activity 2 were in another app, an **implicit intent** would have to be created instead
- Can also pass data between Activities 1 and 2
  - E.g. Activity 1 can tell new activity correct answer (True/False)
Passing Data Between Activities

- Need to pass answer (True/False from QuizActivity to CheatActivity)
  - Pass answer as **extra** on the Intent passed into **StartActivity**
  - **Extras** are arbitrary data calling activity can include with intent
Passing Answer (True/False) as Intent Extra

- To add extra to Intent, use `putExtra()` command
- Encapsulate Intent creation into a method `newIntent()`

```java
public class CheatActivity extends AppCompatActivity {

    private static final String EXTRA_ANSWER_IS_TRUE = "com.bignerdranch.android.geoquiz.answer_is_true";

    public static Intent newIntent(Context packageContext, boolean answerIsTrue) {
        Intent i = new Intent(packageContext, CheatActivity.class);
        i.putExtra(EXTRA_ANSWER_IS_TRUE, answerIsTrue);
        return i;
    }
}
```

- When user clicks cheat button, build Intent, start new Activity

```java
mCheatButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Start CheatActivity
        Intent i = new Intent(QuizActivity.this, CheatActivity.class);
        boolean answerIsTrue = mQuestionBank[mCurrentIndex].isAnswerTrue();
        Intent i = CheatActivity.newIntent(QuizActivity.this, answerIsTrue);
        startActivity(i);
    }
});
updateQuestion();
```
Passing Answer (True/False) as Intent Extra

- Activity receiving the Intent retrieves it using `getBooleanExtra()`

```java
public class CheatActivity extends AppCompatActivity {
    private static final String EXTRA_ANSWER_IS_TRUE = "com.bignerdranch.android.geoquiz.answer_is_true";

    private boolean mAnswerIsTrue;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_cheat);

        mAnswerIsTrue = getIntent().getBooleanExtra(EXTRA_ANSWER_IS_TRUE, false);
    }

    ...}
```

**Important:** Read Android Nerd Ranch (2nd edition) pg 87
Implicit Intents

- **Implicit Intent**: Does not name component to start.
  - Specifies
    - **Action** (what to do, example visit a web page)
    - **Data** (to perform operation on, e.g. web page url)
  - System decides component to receive intent based on **action, data, category**
- Example Implicit Intent to share data

```java
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");
```

- **ACTION** (No receiving Activity specified)
- **Data type**
References

- Android Nerd Ranch, 1st edition
- 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