



The Android Activity Lifecycle

Introduction

- Working with the Android logging system
- Rotation and multiple layouts
- Understanding the Android activity lifecycle

The Android Logging System (2)

- Android supports a shared (system wide) log file
 - You use it to write informational and debugging messages
- It's implemented by the class `android.util.Log`

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

The Android Logging System (2)

`Log` is a logging class that you can utilize in your code to print out messages to the LogCat. Common logging methods include:

- `v(String, String)` (verbose)
- `d(String, String)` (debug)
- `i(String, String)` (information)
- `w(String, String)` (warning)
- `e(String, String)` (error)

For example:

```
Log.i("MyActivity", "MyClass.getView() - get item number " + position);
```

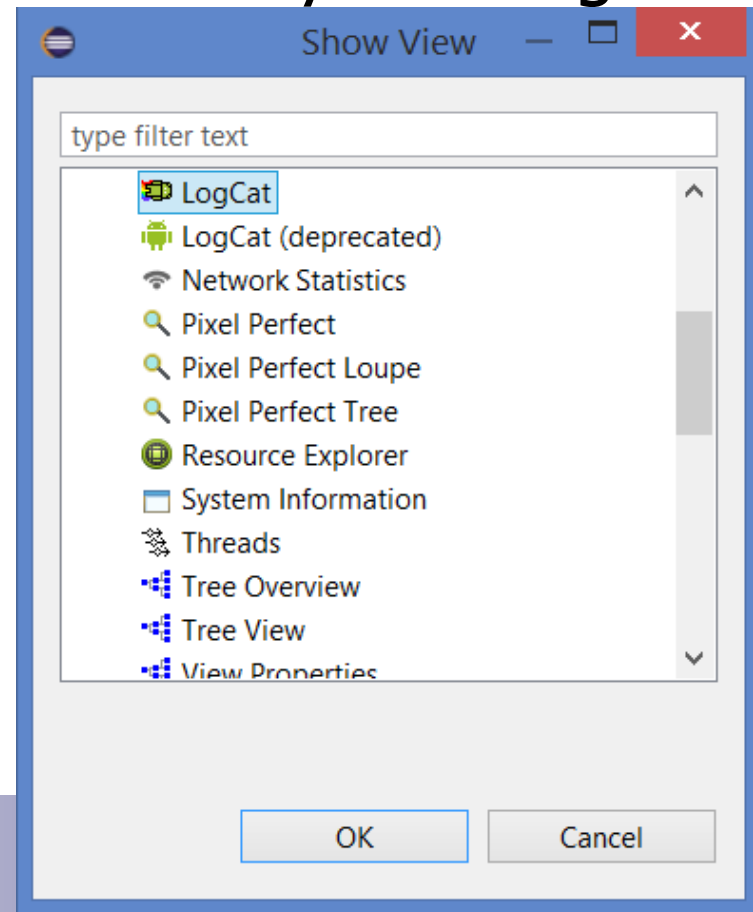
The LogCat will then output something like:

```
I/MyActivity( 1557): MyClass.getView() - get item number 1
```

Viewing std
Debugging \

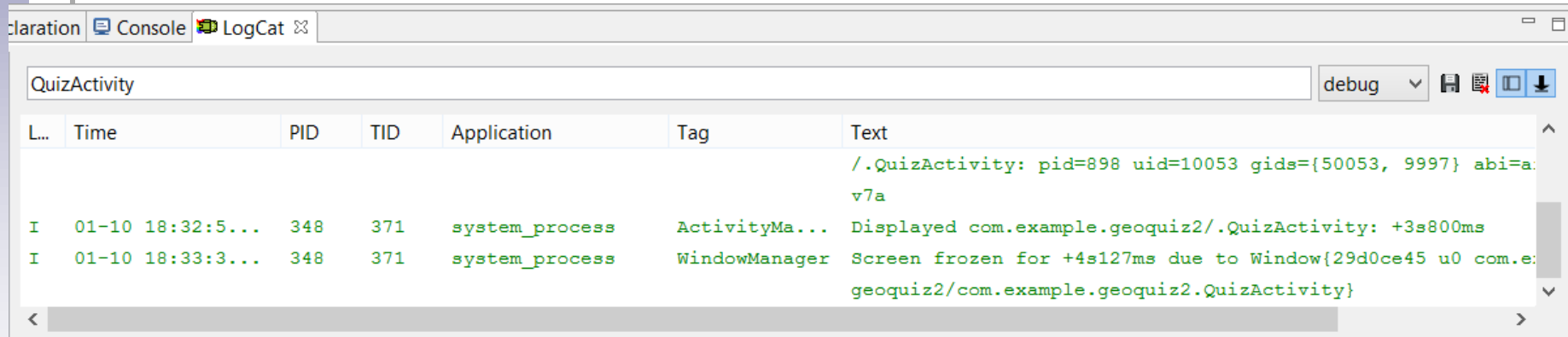
LogCat (Introduction)

- LogCat allows you to view the system log
- In Eclipse, click **Window, Show View, Other**



LogCat (Viewing)

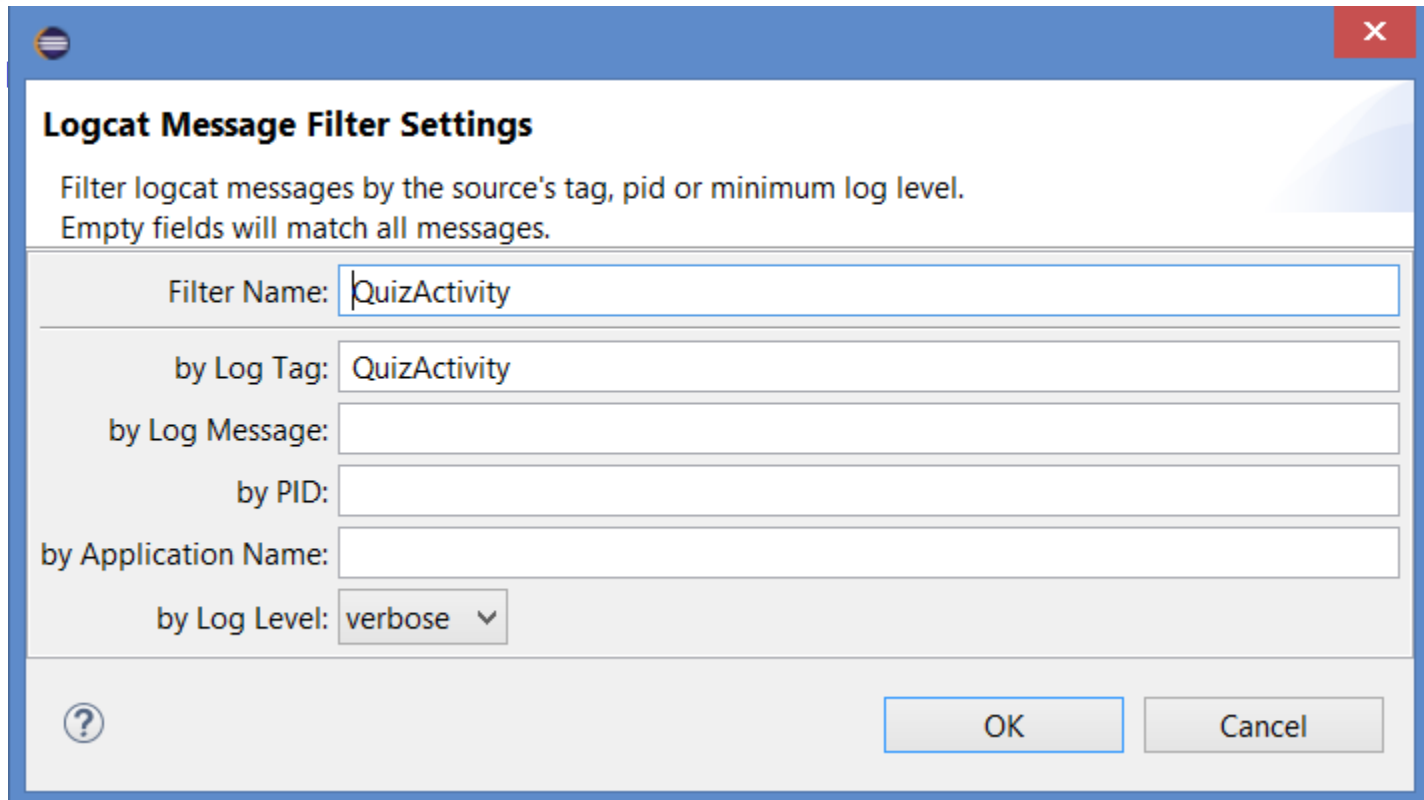
- The LogCat window



QuizActivity debug

L...	Time	PID	TID	Application	Tag	Text
						/.QuizActivity: pid=898 uid=10053 gids={50053, 9997} abi=a... v7a
I	01-10 18:32:5...	348	371	system_process	ActivityMa...	Displayed com.example.geoquiz2/.QuizActivity: +3s800ms
I	01-10 18:33:3...	348	371	system_process	WindowManager	Screen frozen for +4s127ms due to Window{29d0ce45 u0 com.e... geoquiz2/com.example.geoquiz2.QuizActivity}

LogCat (Filter)



The screenshot shows a dialog box titled "Logcat Message Filter Settings". The dialog has a blue header bar with a close button (X) in the top right corner. Below the title, there is a brief instruction: "Filter logcat messages by the source's tag, pid or minimum log level. Empty fields will match all messages." The main area of the dialog contains several input fields and a dropdown menu:

- Filter Name:** A text input field containing "QuizActivity".
- by Log Tag:** A text input field containing "QuizActivity".
- by Log Message:** An empty text input field.
- by PID:** An empty text input field.
- by Application Name:** An empty text input field.
- by Log Level:** A dropdown menu currently set to "verbose".

At the bottom of the dialog, there is a help icon (question mark in a circle) on the left and two buttons, "OK" and "Cancel", on the right.

Saving Data (Bundle)

- We save data for reasons other than rotation
- Note that the data type argument is **Bundle**
 - There are methods to write each of the primary data types
- We typically override `onSaveInstanceState(Bundle outstate)`

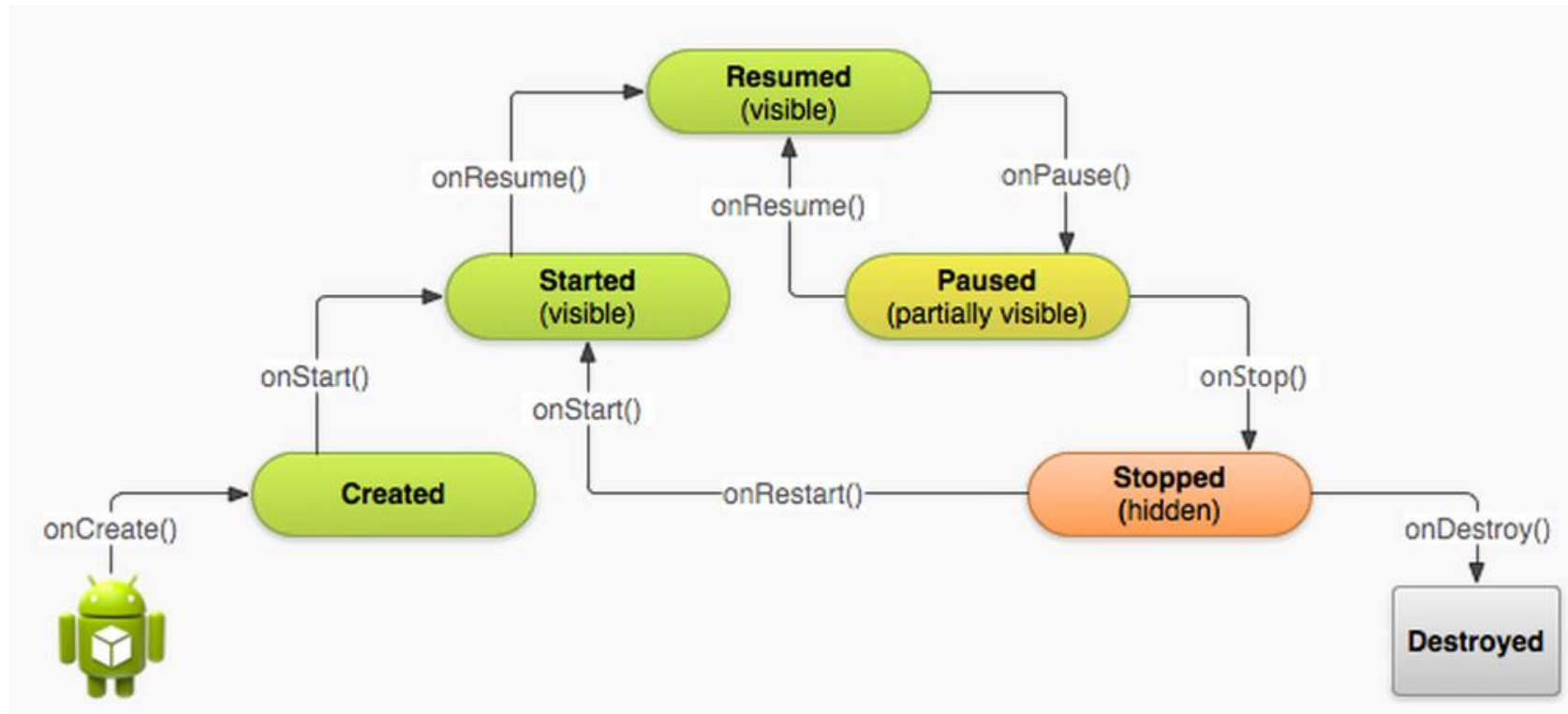
Saving Data (Bundle)

void	<code>putBinder</code> (<code>String</code> key, <code>IBinder</code> value) Inserts an <code>IBinder</code> value into the mapping of this <code>Bundle</code> , replacing any existing value for the given key.
void	<code>putBoolean</code> (<code>String</code> key, <code>boolean</code> value) Inserts a <code>Boolean</code> value into the mapping of this <code>Bundle</code> , replacing any existing value for the given key.
void	<code>putBooleanArray</code> (<code>String</code> key, <code>boolean[]</code> value) Inserts a <code>boolean</code> array value into the mapping of this <code>Bundle</code> , replacing any existing value for the given key.
void	<code>putBundle</code> (<code>String</code> key, <code>Bundle</code> value) Inserts a <code>Bundle</code> value into the mapping of this <code>Bundle</code> , replacing any existing value for the given key.
void	<code>putByte</code> (<code>String</code> key, <code>byte</code> value) Inserts a <code>byte</code> value into the mapping of this <code>Bundle</code> , replacing any existing value for the given key.

The Activity Lifecycle

- And now on to the activity lifecycle
- It is here that we get into the meat of things

Activity Lifecycle (Illustration)



Activity Lifecycle (Introduction)

- As users navigate in and out of your application, activities change states
- As this happens, the Android system calls various **lifecycle methods**
 - You have only seen `onCreate ()` so far
- You can control how your application behaves as the user interacts with your application
- All of this is implemented through **callback methods**

Activity Lifecycle (States)

- Running
 - The activity has focus and is at the top of the activity stack.
- Paused
 - The device goes to sleep
 - The activity is partially hidden
- Stopped
 - The activity is obscured by another activity

Activity States (Types)

- There are two types of activity states
 - Applications can exist for a long period of time in a **static state**
 - Applications can exist in a **transient state** for very short periods of time
 - After `onCreate()`, `onStart()` is called followed by `onResume()`

Starting the Application (1)

- So far you have worked with only a single activity
 - However, many programs will have several activities
- An application has one **launcher activity** that is started when the application is run
 - This activity executes when the user clicks the application icon
- The launcher activity defined in the **AndroidManifest.xml** file

Starting the Application (2)

- Create the main activity and launcher as follows

```
<application
  android:allowBackup="true"
  android:icon="@drawable/ic_launcher"
  android:label="@string/app_name"
  android:theme="@style/AppTheme" >
  <activity
    android:name=".MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
  </activity>
  <activity
    android:name=".SecondActivity"
    android:label="@string/title_activity_second" >
  </activity>
</application>
```

Launcher Activity

Second Activity

Creating an Application Instance

- You typically (must) implement the `onCreate ()` method for each activity
- The code executes once for the entire lifetime of the activity
- In this method you implement basic application startup logic
 - Define the user interface (view to use) here
 - Initialize (instantiate) class-scoped variables here

Understanding onCreate ()

- `onCreate ()` is the first step in an application's lifecycle
 - The `Bundle` object provides the means to save and get state data (more later)
- First, we call the base class method
- Then we typically declare the user interface (view to use) with `setContentView ()`
- After `onCreate ()`, the system calls `onStart ()` and then `onResume ()`

Understanding onCreate ()

- Simple code for onCreate ()
 - Call the base class method
 - **setContentView ()** to the desired activity

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

Understanding `onDestroy()`

- `onDestroy()` is the last callback method called when the application is removed from system memory
- In most cases, you need not implement this method because most cleanup is performed in `onPause()` or `onStop()`
 - Except if you call `finish()` in `onCreate()`
- Use `onDestroy()` to kill long-running resources

Pausing and Resuming an Activity (Introduction)

- When a foreground activity is partially obscured (because of a Toast, for example) the system calls **onPause ()**
 - Note that the user may or may not resume the activity
- If the user returns to your activity **onResume ()** is called
- There is no guarantee that the user will resume the activity though

Pausing an Activity

- When `onPause ()` is called, your application is partially visible
- It is here that you should
 - Stop animations
 - Save changes, as necessary
 - Release system resources such as broadcast receivers (such as a GPS)
 - Release resources that consume battery life
 - Keep the operations here simple

Resuming an Activity

- When the user resumes an activity from a paused state, the system calls `onResume ()`
- The method is called every time an application comes to the foreground including the first time
- So initialize resources released in `onResume ()`

Stopping and Restarting an Activity

- An application might be stopped and restarted because
 - The user switches from app to app in the recent apps window
 - The user performs an action that starts a new activity
 - The user receives a phone call while using your app
- Use `onStop()` for complex tasks, such as a database write

Stopping an Activity

`onStop ()`

- The `onStop ()` callback fires when an application is fully obscured (instead of partially obscured)
 - User's focus is on another activity
- Activity instance still lives in system memory while stopped
 - But the system might destroy the instance if application memory is short
 - So often you need only handle `onPause ()` and `onResume ()`

Restarting an Activity

`onRestart()`

- `onRestart()` is called when an activity comes to the foreground from a stopped state
- It is not called when the application first starts
 - But `onStart()` is called in both cases
- Generally, we only need to handle `onStart()`

Rotation

- When you rotate a device, the current activity is destroyed, and then recreated
 - This resets the application's state
- Rotation also changes the device configuration
- The device configuration describes the current state of the Android device
 - Screen orientation, density, size, keyboard type, docking mode, etc...

Rotation (Implementing)

- Create the layout .xml file with the same name in the folder res/layout-land
 - The filename must be the same so that runtime can figure out which resource to use
 - Note that res/layout-land is magical

