

Mobile Application Development

Produced
by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology

<http://www.wit.ie>



Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE



Android Anatomy





Agenda

- ❑ Overview of Android Application Components
- ❑ The Android Application Life Cycle
- ❑ The Online Developer Resources
- ❑ The “*Donation*” Case Study – a first (very brief!) look...



Android App Components

- ❑ App components are the essential **building blocks** of an Android app. Each component is a different point through which the system can enter your app.
- ❑ Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role—each one is a unique building block that helps define your app's overall behavior.
- ❑ There are four different types of app components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.
- ❑ We'll briefly mention a few other components (of sorts) that also make up your App worth knowing about.



Android App Components

1. Activities

- represents a single screen with a user interface
- acts as the '*controller*' for everything the user sees on its associated screen
- implemented as a subclass of [Activity](#)
- e.g. email app (listing your emails)

2. Services

- a component that runs in the background to perform long-running operations or to perform work for remote processes
- does not provide a user interface
- can be started by an activity
- is implemented as a subclass of [Service](#)
- e.g. music player (playing in background)



Android App Components

3. Content Providers

- manages a shared set of app data
- can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your app can access
- through the Content Provider, other apps can query or even modify the data
- e.g. Users Contacts (your app could update contact details)

4. Broadcast Receivers

- a component that responds to system-wide broadcast announcements
- broadcasts can be from both the system and your app
- implemented as a subclass of [BroadcastReceiver](#) and each broadcast is delivered as an [Intent object](#)
- e.g. battery low (system) or new email (app via notification)

How it all Fits Together *

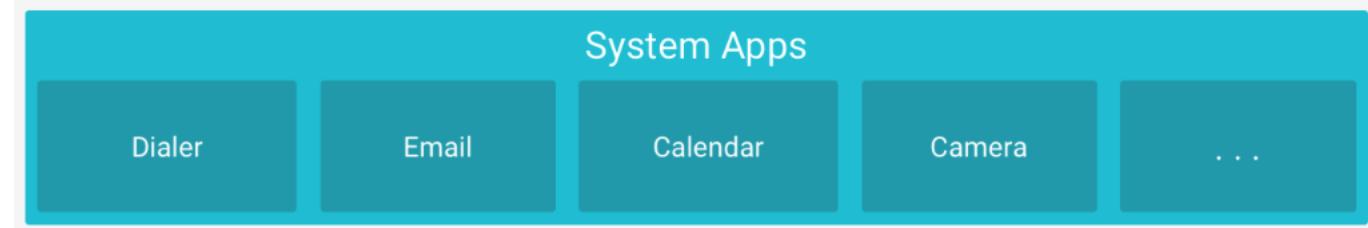
- Based on the **Model View Controller** design pattern.
- Don't think of your program as a linear execution model:
 - Think of your program as existing in logical blocks, each of which performs some actions.
- The blocks communicate back and forth via message passing (**Intents**)
 - Added advantage, physical user interaction (screen clicks) and inter process interaction can have the same programming interface
 - Also the OS can bring different pieces of the app to life depending on memory needs and program use
- For each distinct logical piece of program behavior you'll write a Java class (derived from a base class).
- **Activities/Fragments**: Things the user can **see** on the screen. Basically, the 'controller' for each different screen in your program.
- **Services**: Code that isn't associated with a screen (background stuff, fairly common)
- **Content providers**: Provides an interface to exchange data between programs (usually SQL based)
- You'll also design your layouts (screens), with various types of widgets (**Views**), which is what the user sees via **Activities & Fragments**

**Big N.B.
for all these!**



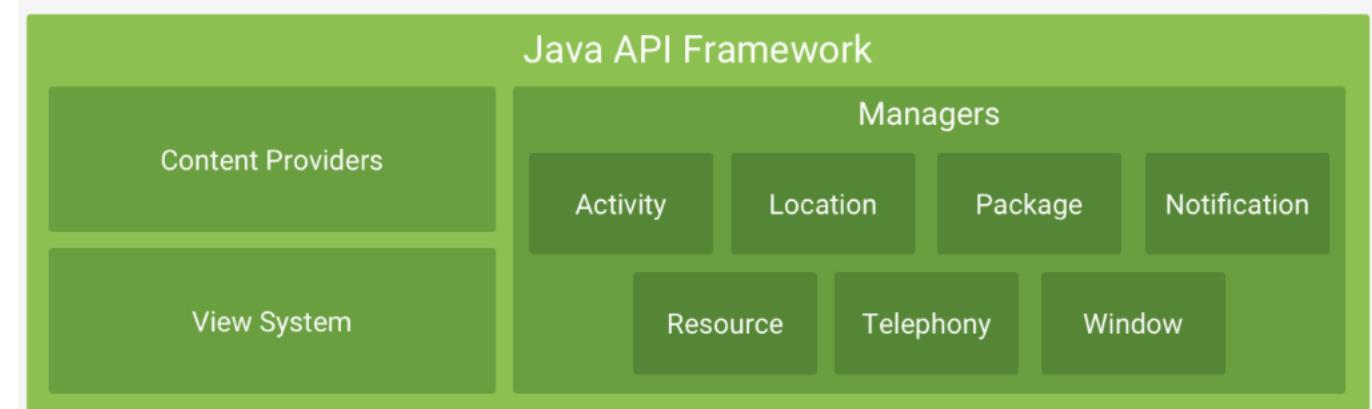


Layered Software Framework (s/w stack)





Layered Software Framework (s/w stack)



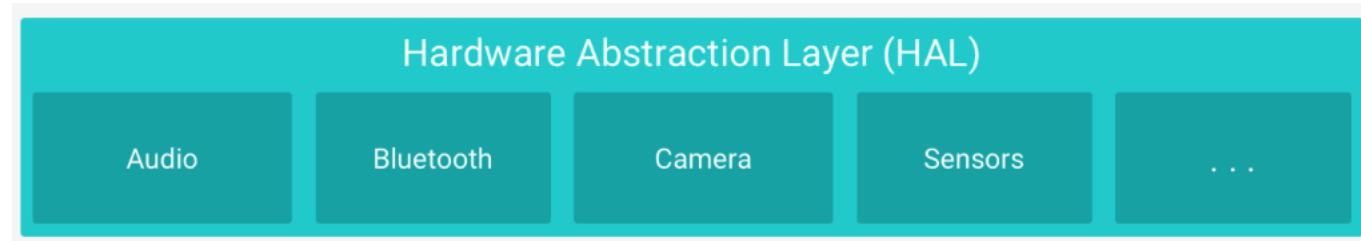


Layered Software Framework (s/w stack)



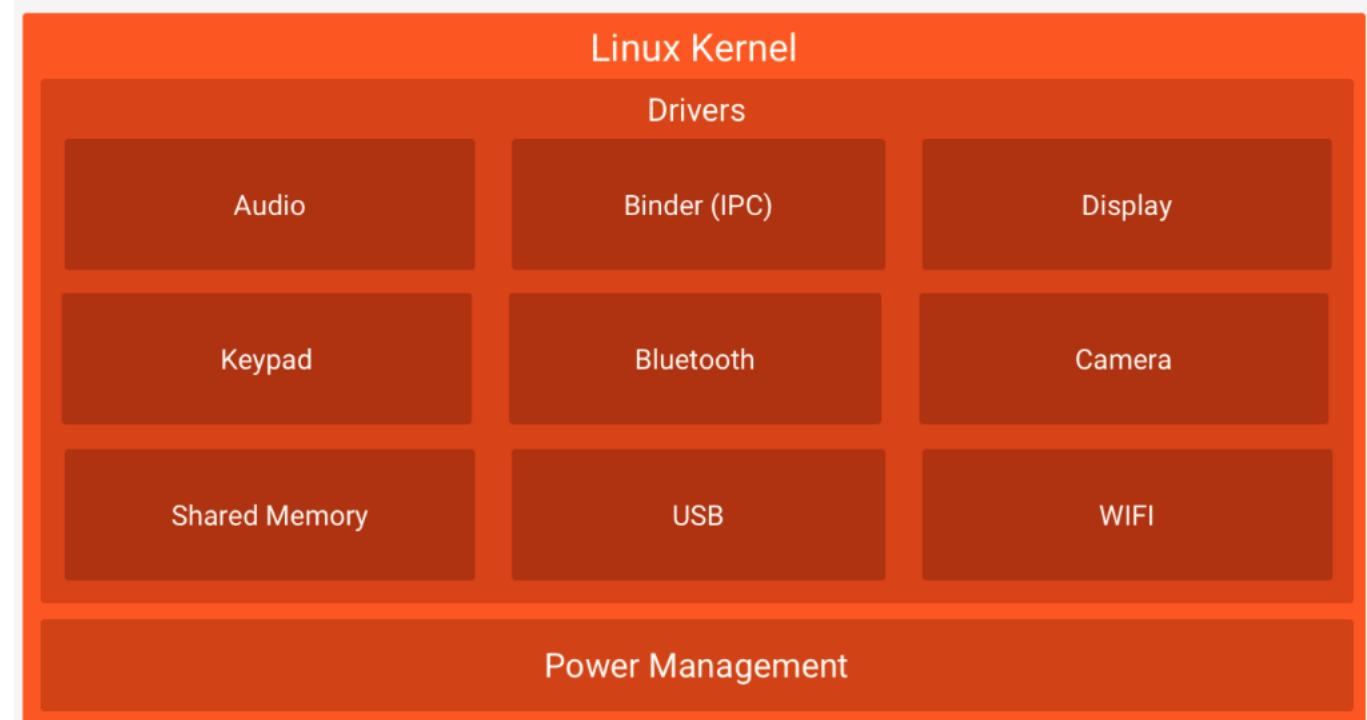


Layered Software Framework (s/w stack)





Layered Software Framework (s/w stack)





The (Application) Activity Life Cycle *

- ❑ Android is designed around the unique requirements of mobile applications.
 - In particular, Android recognizes that resources (memory and battery, for example) are limited on most mobile devices, and provides mechanisms to conserve those resources.
- ❑ The mechanisms are evident in the *Android Activity Lifecycle*, which defines the states or events that an activity goes through from the time it is created until it finishes running.

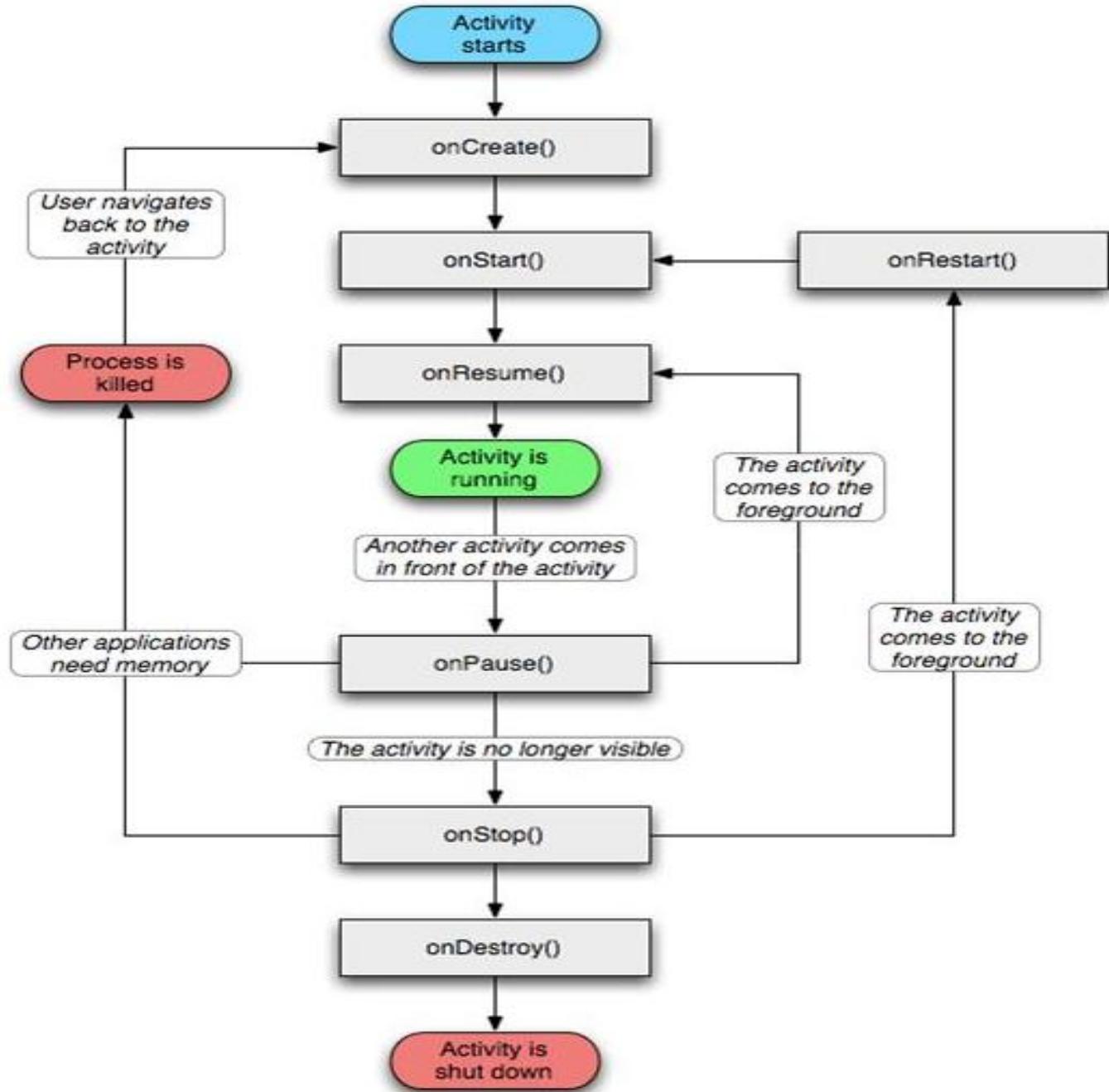


The (Application) Activity Life Cycle

- ❑ An application itself is a set of activities with a Linux process to contain them
 - However, an application DOES NOT EQUAL a process
 - Due to (previously mentioned) low memory conditions, an activity might be suspended at any time and its process be discarded
 - ◆ The activity manager remembers the state of the activity however and can reactivate it at any time
 - ◆ Thus, an activity may span multiple processes over the life time of an application

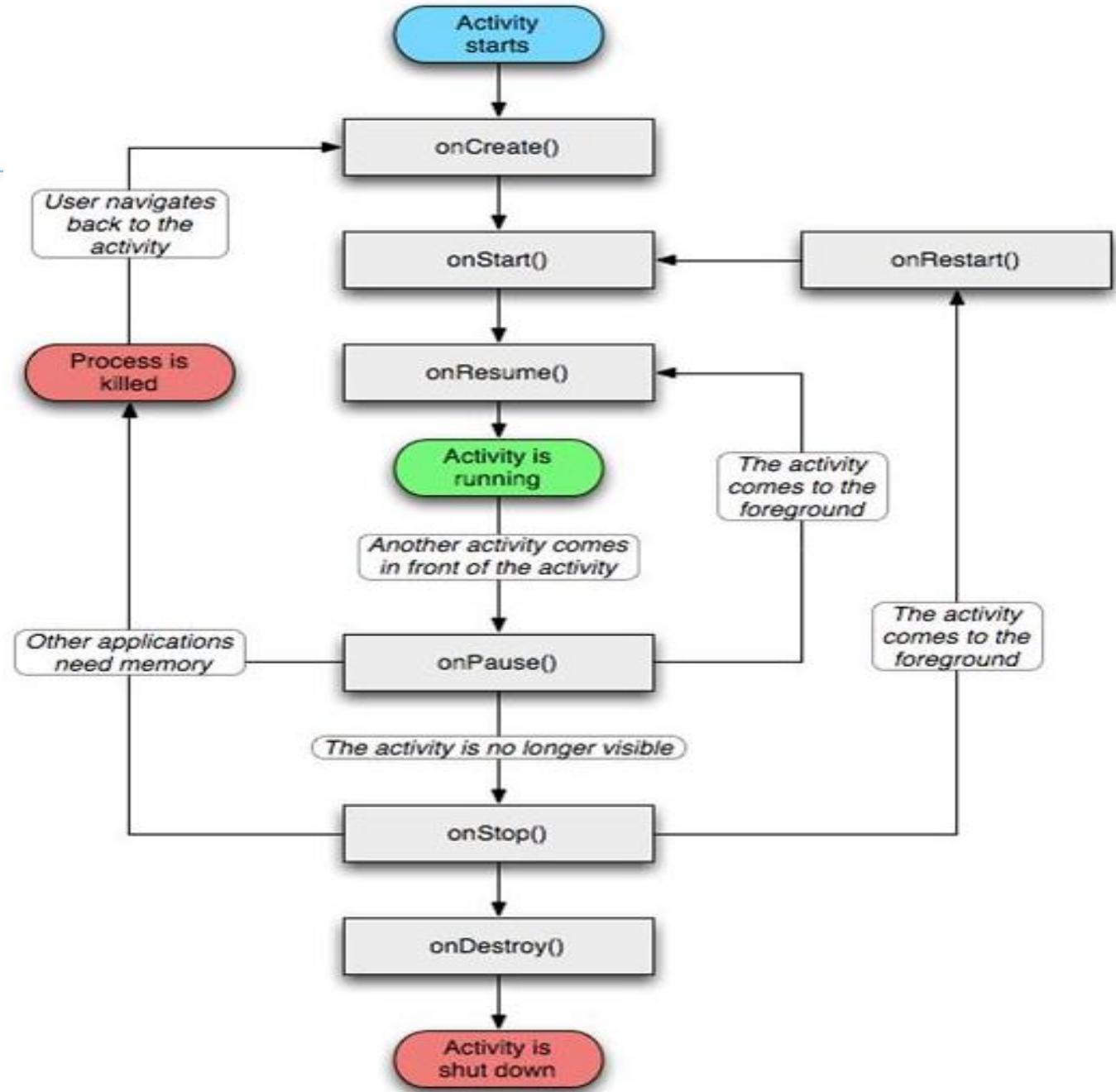
The **Activity** Life Cycle *

- The Activity has a number of predefined functions that you override to handle events from the system.
- If you don't specify what should be done the system will perform the default actions to handle events.
- Why would you want to handle events such as **onPause()**, etc... ?
 - You will probably want to do things like release resources, stop network connections, back up data, etc...



The **Activity** Life Cycle

- ❑ At the *very minimum*, you need (and is supplied) **onCreate()**
 - ❑ **onStop()** and **onDestroy()** are optional and may never be called
 - ❑ If you need persistence, the save needs to happen in **onPause()**



LifeCycle

Example (1) *

User Launches App

The screenshot shows the Android Studio interface with the following components:

- Code Editor:** Displays the `MainActivity.java` file. The code defines a lifecycle method `onResume()` which logs "onResume() Called...".
- Android Monitor:** Shows logcat output for the `Emulator Nexus_5_API_23`. A red box highlights the following log entries:

```
09-28 16:29:42.487 11916-11916/ie.wit.lifecycle V/LifeCycle: onCreate() Called...
09-28 16:29:42.490 11916-11916/ie.wit.lifecycle V/LifeCycle: onStart() Called...
09-28 16:29:42.490 11916-11916/ie.wit.lifecycle V/LifeCycle: onResume() Called...
```

A green arrow points from the text "User Launches App" to the `onResume()` log entry.
- Emulator:** Shows a smartphone screen with the app's UI, displaying "Lifecycle" in the title bar and "Hello World!" in the main view.

LifeCycle

Example (2) *

User Selects 'Home'

The screenshot shows an Android development environment with the following components:

- Project Structure:** Shows the project tree with modules like `ie.wit.lifecycle`, `MainActivity`, and test configurations.
- Code Editor:** Displays the `MainActivity.java` file containing lifecycle methods. A green arrow points from the code editor towards the Android Monitor.
- Android Monitor:** Shows logcat output for the emulator. A red box highlights the following entries:

```
09-28 16:31:38.898 13532-13532/ie.wit.lifecycle V/LifeCycle: onPause() Called...
09-28 16:31:39.749 13532-13532/ie.wit.lifecycle V/LifeCycle: onStop() Called...
```
- Emulator:** An Android 6.0 (API 23) emulator running on a Nexus 5 device. The screen shows the home screen with various app icons. A green arrow points from the red box in the Android Monitor towards the emulator screen.

LifeCycle Example (3) *

User restarts App

The screenshot shows the Android Studio interface with the following components:

- Project Structure:** Shows the project tree with modules like ie.wit.lifecycle, Main Activity, and test suites.
- Code Editor:** Displays the `MainActivity.java` file containing lifecycle methods. Line 50 is highlighted.
- Android Monitor:** Shows logcat output for the Emulator Nexus_5_API_23. A red box highlights the following log entries:

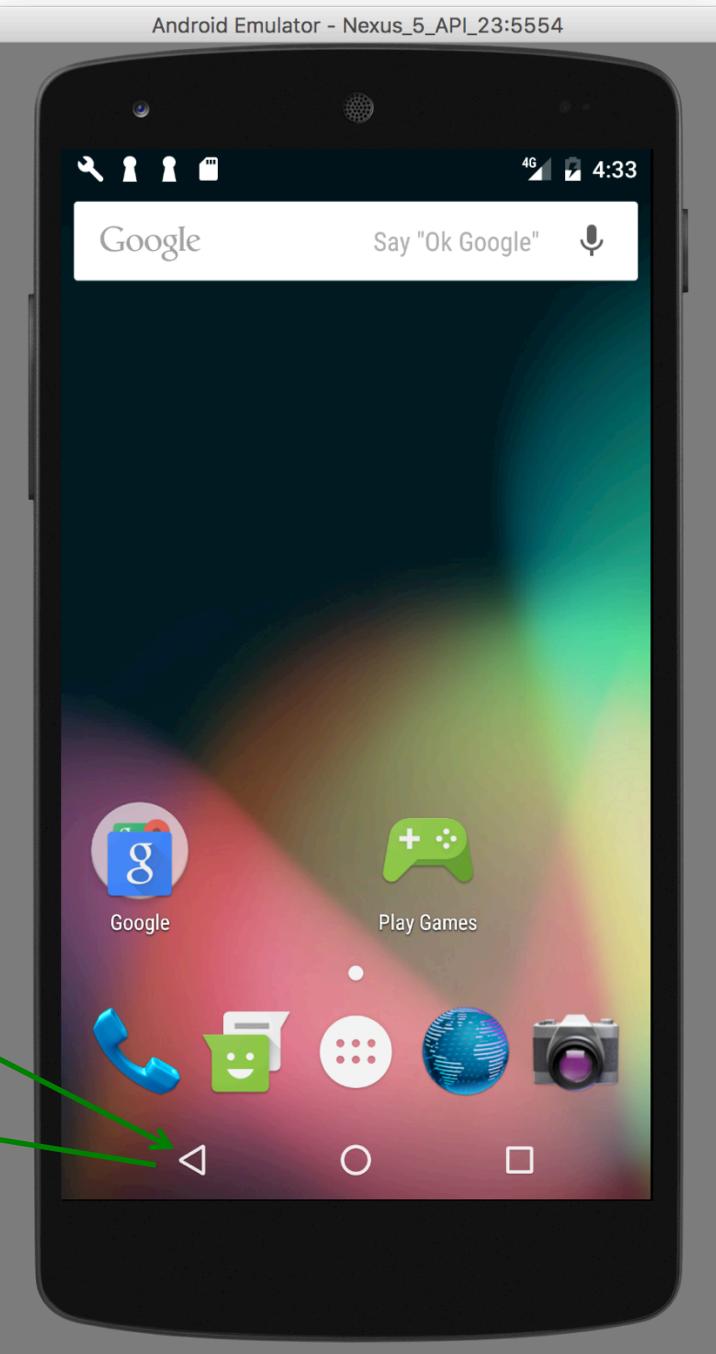
```
09-28 16:32:14.834 13532-13532/ie.wit.lifecycle V/LifeCycle: onRestart() Called...
09-28 16:32:14.836 13532-13532/ie.wit.lifecycle V/LifeCycle: onStart() Called...
09-28 16:32:14.836 13532-13532/ie.wit.lifecycle V/LifeCycle: onResume() Called...
```
- Emulator:** Shows a smartphone screen with the app's name "LifeCycle" and the text "Hello World!".

A green arrow points from the text "User restarts App" to the red box in the Android Monitor.

```
19
20
21
22
23 @Override
24 protected void onRestart()
25     super.onRestart();
26     Log.v("LifeCycle", "onStart() Called...");
27
28
29
30 @Override
31 protected void onStart()
32     super.onStart();
33     Log.v("LifeCycle", "onStart() Called...");
34
35
36 @Override
37 protected void onResume()
38     super.onResume();
39     Log.v("LifeCycle", "onResume() Called...");
40
41
42 @Override
43 protected void onPause()
44     super.onPause();
45     Log.v("LifeCycle", "onPause() Called...");
46
47
48 @Override
49 protected void onDestroy()
50     super.onDestroy();
51     Log.v("LifeCycle", "onDestroy() Called...");
52
53 }
```

LifeCycle Example (4) *

User Selects 'Back'



The screenshot shows an Android emulator running on a Nexus 5 device with API 23. The screen displays the home screen with several icons: Google, Play Games, Phone, Messages, Contacts, and Camera. The status bar at the top shows signal strength, battery level, and the time 4:33.

The code editor window on the left shows the `MainActivity` class from the `ie.wit.lifecycle` package. The code implements the `onPause()`, `onStop()`, and `onDestroy()` methods of the `Activity` lifecycle. The `onDestroy()` method includes a call to `super.onDestroy()` and a log statement.

```
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
```

The Android Monitor window at the bottom shows logcat output for the `Emulator Nexus_5_API_23`. It lists three entries corresponding to the lifecycle events:

- 09-28 16:32:48.899 13532-13532/ie.wit.lifecycle V/LifeCycle: onPause() Called...
- 09-28 16:32:49.337 13532-13532/ie.wit.lifecycle V/LifeCycle: onStop() Called...
- 09-28 16:32:49.337 13532-13532/ie.wit.lifecycle V/LifeCycle: onDestroy() Called...

A red box highlights the last two entries in the logcat output, and a green arrow points from this box to the back button icon on the emulator screen.



So, after all that, how do I Design my App?

- The way the system architecture is set up is fairly open:
 - App design is somewhat up to you, but you still have to live with the Android execution model.
- Start with the different **screens/layouts (Views)** that the user will see. These are controlled by the different **Activities (Controllers)** that will comprise your system.
- Think about the **transitions** between the screens, these will be the **Intents** passed between the Activities.
- Think about what background **services** you might need to incorporate.
 - Exchanging data
 - Listening for connections?
 - Periodically downloading network information from a server?
- Think about what **information** must be stored in long term memory (SQLite) and possibly design a content provider around it.
- Now connect the Activities, services, etc... with Intents...
- Don't forget good OOP ☺ and
- **USE THE DEVELOPER DOCs & GUIDES (next few slides)**

Save the date! [Android Dev Summit](#) is coming to Mountain View, CA on November 7-8, 2018.

FEATURED

Introducing Android 9 Pie

Powered by AI to make your smartphone smarter, simpler and tailored to you.

[GET STARTED](#)



FEATURED

Google I/O 2018 videos

View all the videos from Google I/O 2018, introducing new platform features, tools, and deep-dives.



FEATURED

Introducing Android Jetpack

Components, tools and architectural guidance to accelerate Android development, eliminate boilerplate code, and build high quality, robust apps.



 Developers Platform Android Studio Google Play Android Jetpack Docs News

 Search



Start building an app

Whether you're an experienced developer or creating your first Android app, here are some resources to get you started.



android studio

[DOWNLOAD](#)

Developer guides

Here you'll find a wide range of documentation that teaches you how to build an app, including how to build your first Android app, how to build layouts that adapt to different screens, how to save data in a local database, how to use device sensors and cameras, and much more.

GET STARTED



Sample code

Jump-start your development using these sample projects

[SEE THE SAMPLES](#)



Test your app

Verify your app's behavior and usability before you release

[LEARN HOW TO TEST](#)



Quality guidelines

Build a high quality app with these design and behavior guidelines

[SEE THE GUIDELINES](#)



Distribute on Google Play

Reach a global audience and earn revenue

[LEARN ABOUT PLAY](#)

The screenshot shows the Android Developers website interface. At the top, there's a navigation bar with links for Developers, Platform, Android Studio, Google Play, Android Jetpack, Docs, and News. A search bar is located on the right side of the header. In the top right corner, there's a circular icon featuring a stylized green Android robot head with gear-like details.

DOCS

Material Design

Android apps are designed using the Material Design guidelines. These guidelines provide everything you need to know about how to design your app, from the user experience flow to visual design, motion, fonts, and more.

DESIGN FOR ANDROID

PLATFORM
Wear OS

PLATFORM
TV

PLATFORM
Auto

A large graphic on the right side of the main content area illustrates the Material Design philosophy with various geometric shapes like circles, triangles, and lines in blue, green, and white, set against a dark background with a grid pattern.



Android Developers Platform Android Studio Google Play Android Jetpack Docs News

OVERVIEW GUIDES REFERENCE SAMPLES DESIGN & QUALITY

Documentation for app developers

Whether you're building for Android handsets, Wear OS by Google, Android TV, Android Auto, or Android Things, this section provides the guides and API reference you need.

Get started

- Build your first app
- Sample code
- API reference
- Design guidelines
- Codelab tutorials
- Training courses

Open "<https://developer.android.com/docs/>" in a new tab behind the current one

Android devices

- Wear OS
- Android TV
- Android Auto
- Android Things
- Chrome OS Devices

Best practices

- Testing
- Performance
- Accessibility
- Security
- Enterprise
- Emerging markets

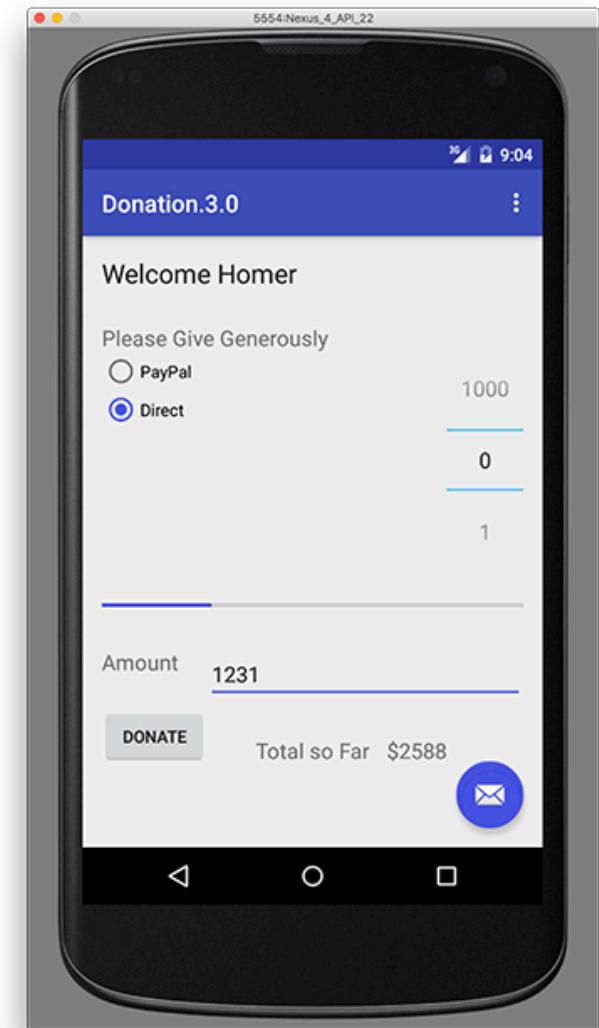


Case Study

❑ **Donation** – an Android App to keep track of donations made to ‘*Homers Presidential Campaign*’.

❑ **App Features**

- Accept donation via number picker or typed amount
- Keep a running total of donations
- Display report on donation amounts and types
- Display running total on progress bar





Ultimate Case Study





Summary

- We looked at the Android Application Components
- Became aware of The Android Application Life Cycle
- Briefly viewed the Online Developer Resources
- Took a quick look at The “*Donation*” Case Study



Questions?



Thanks!

