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

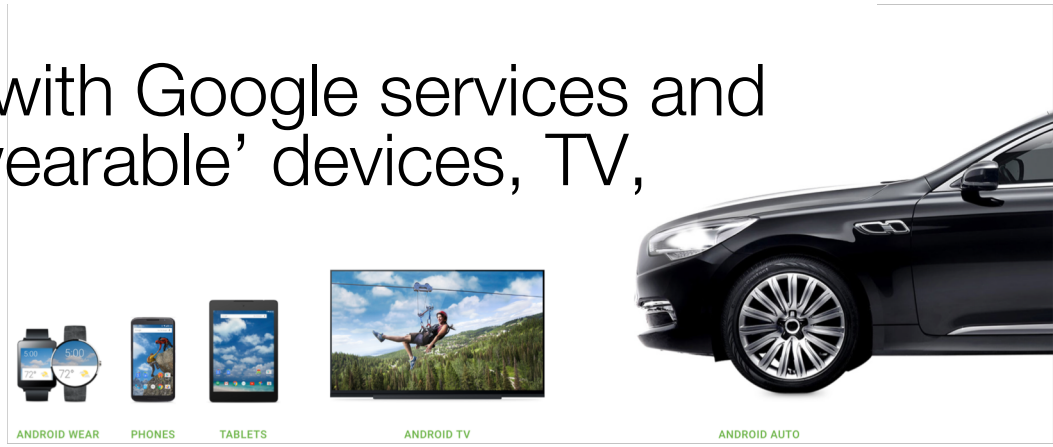
Android Anatomy

Agenda

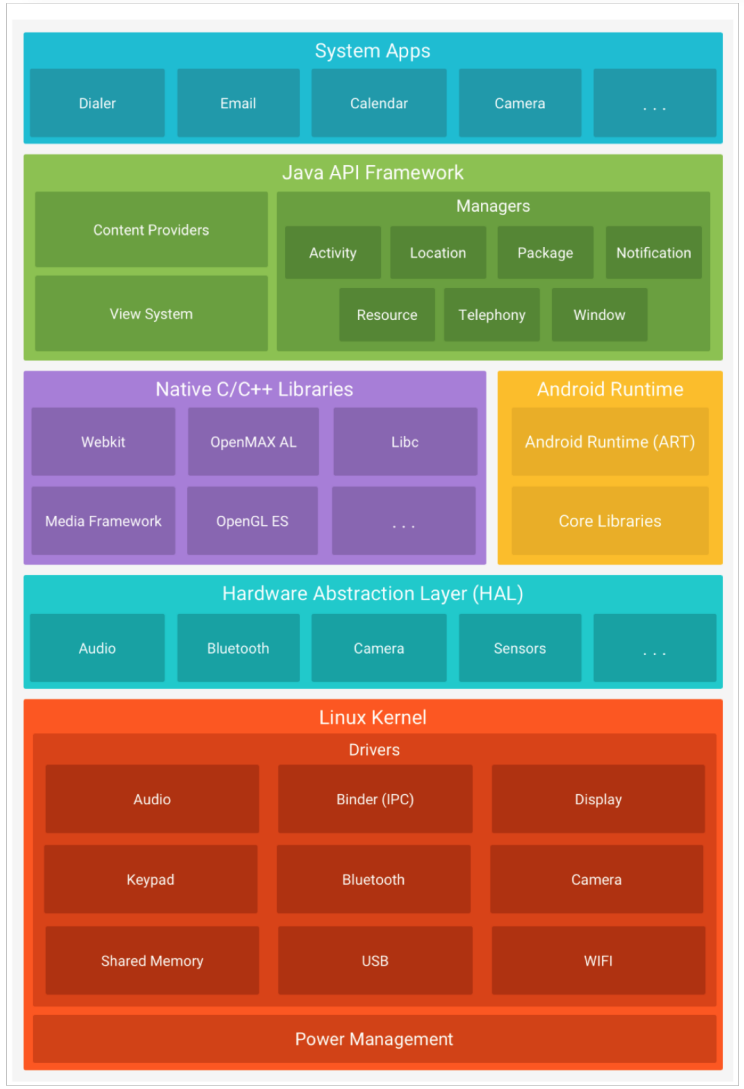
- Quick Recap - What is Android (and it's Layered Framework)
- Important Android Application Components
- The Android Application (Activity/Fragment) Life Cycle
- The Online Developer Resources
- Our "*CoffeeMate*" Case Study – a first look…

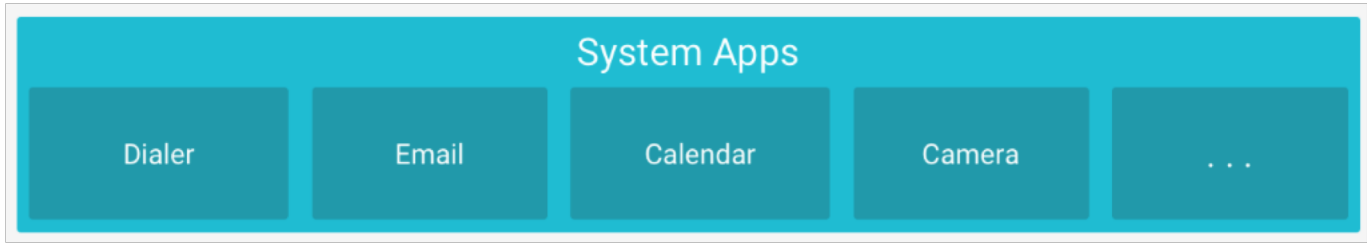
What is Android? (Recap)

- An open source software toolkit created, updated and maintained by Google and the OHA
  - 30+ technology companies
  - Commitment to openness, shared vision, and concrete plans
- Designed for Mobile Devices
  - 2.X series and previous: mobile phones
  - 3.X series: extended to also support tablets
  - 4.X series: unified API framework
  - 5.X / 6.X / 7.X series: more integration with Google services and more tablet-specific features, run on ‘wearable’ devices, TV, vehicles etc..
- Comprehensive Framework

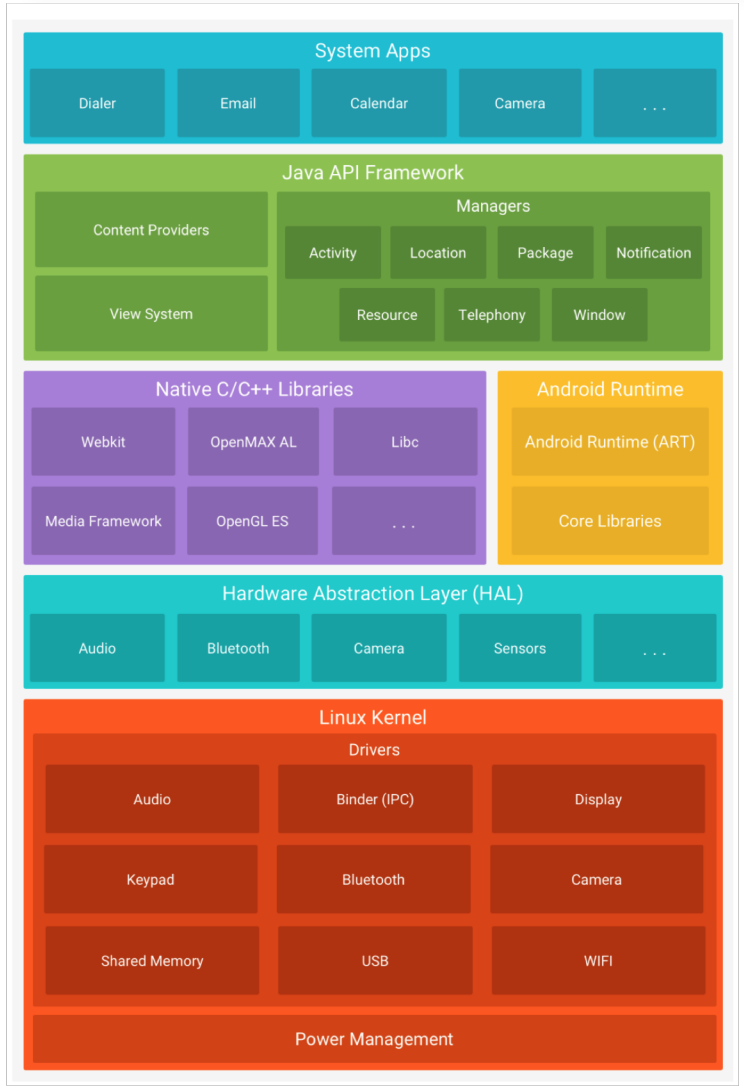


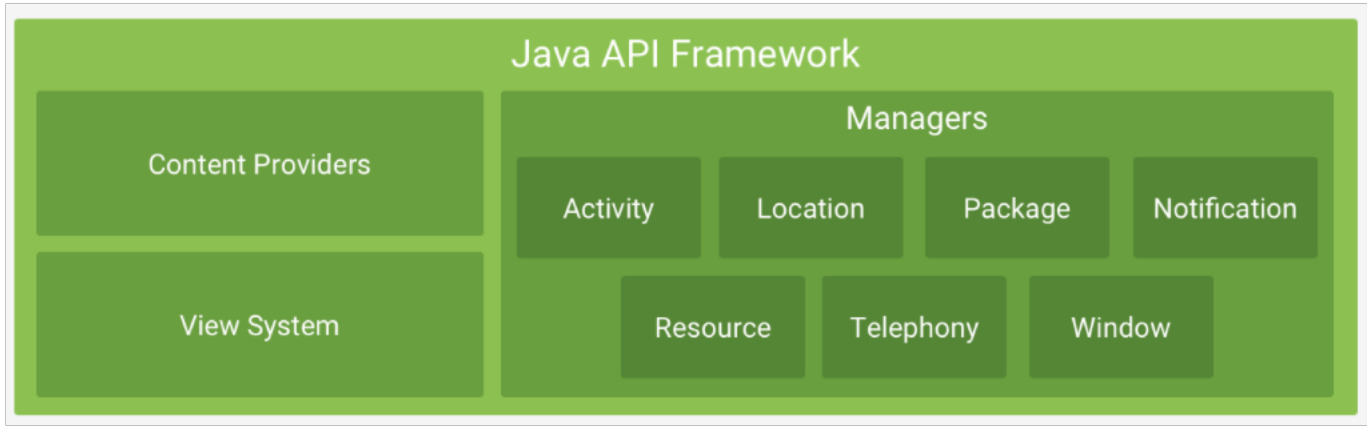
Layered Software Framework (s/w stack)



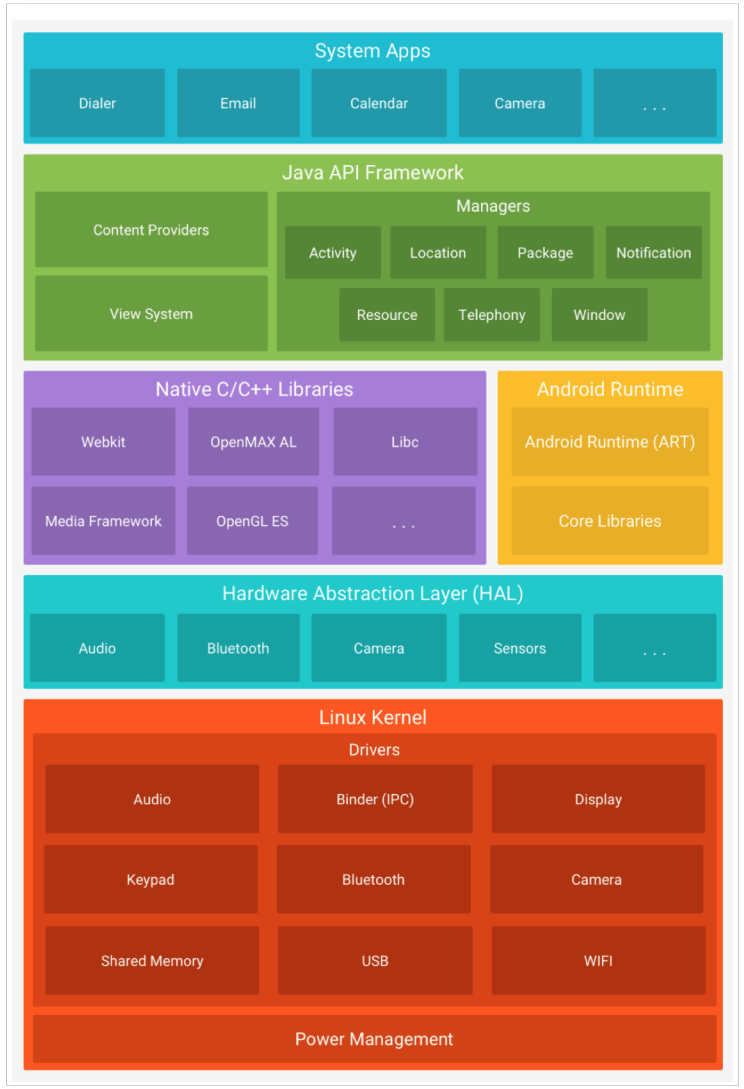


Layered Software Framework (s/w stack)

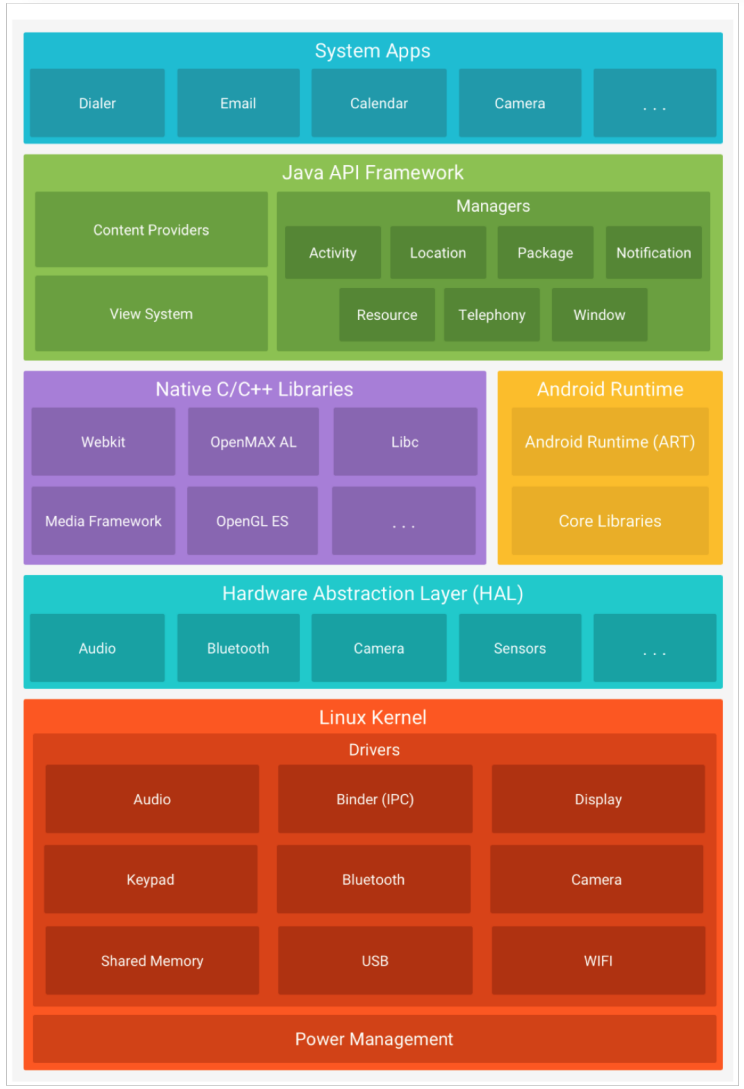


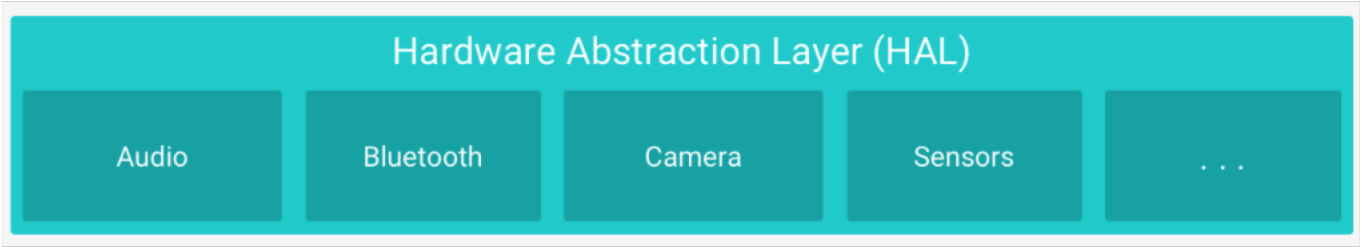


Layered Software Framework (s/w stack)

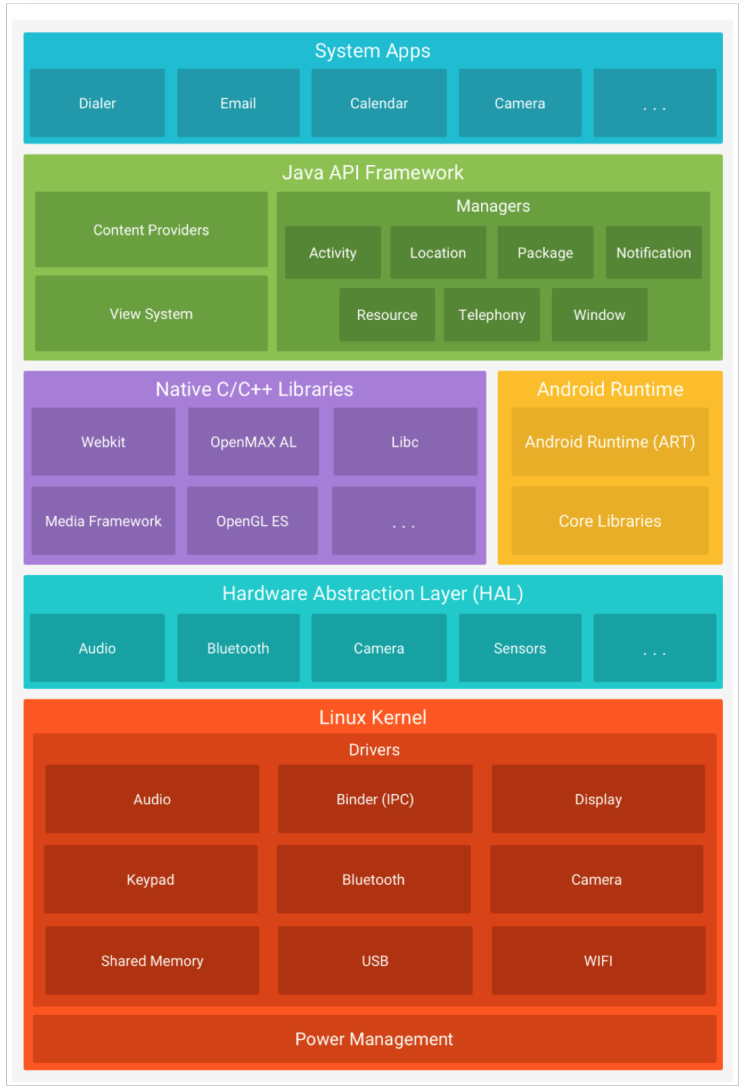


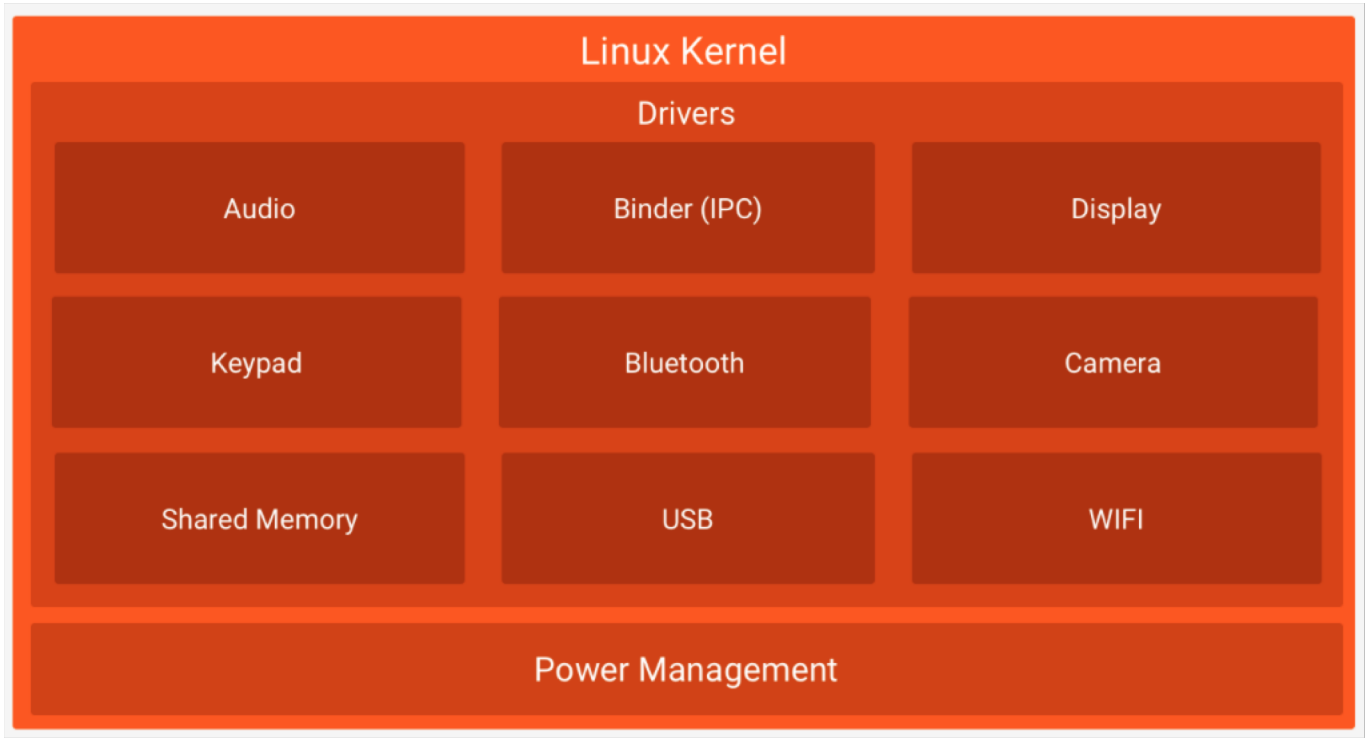
Layered Software Framework (s/w stack)





Layered Software Framework (s/w stack)





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 main 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 *

Big N.B. for all these!

- 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

See the Appendix for a more detailed explanation of these components

- 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*

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.

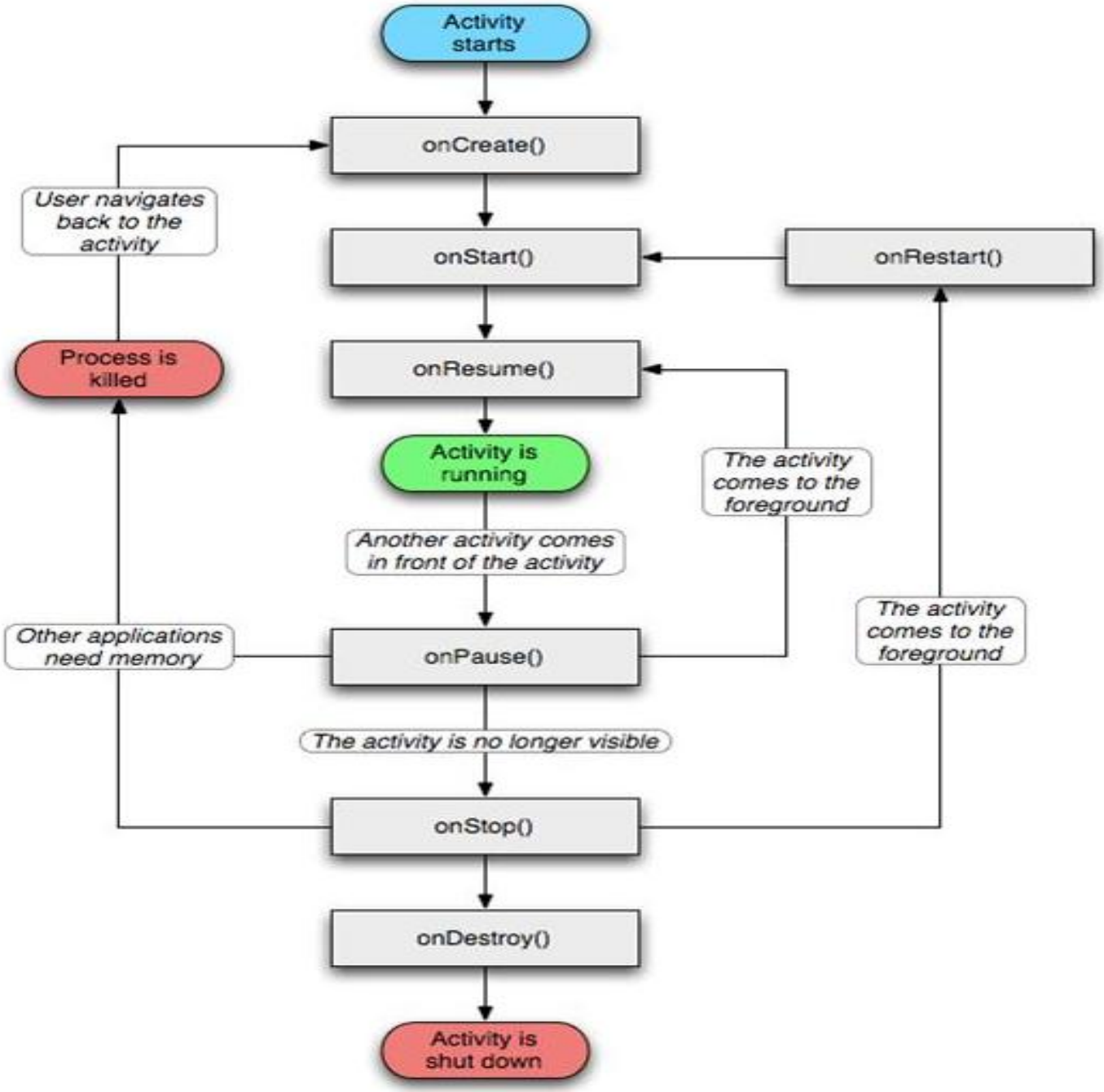
See the Appendix for a more detailed explanation of these ‘states’

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 (the 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

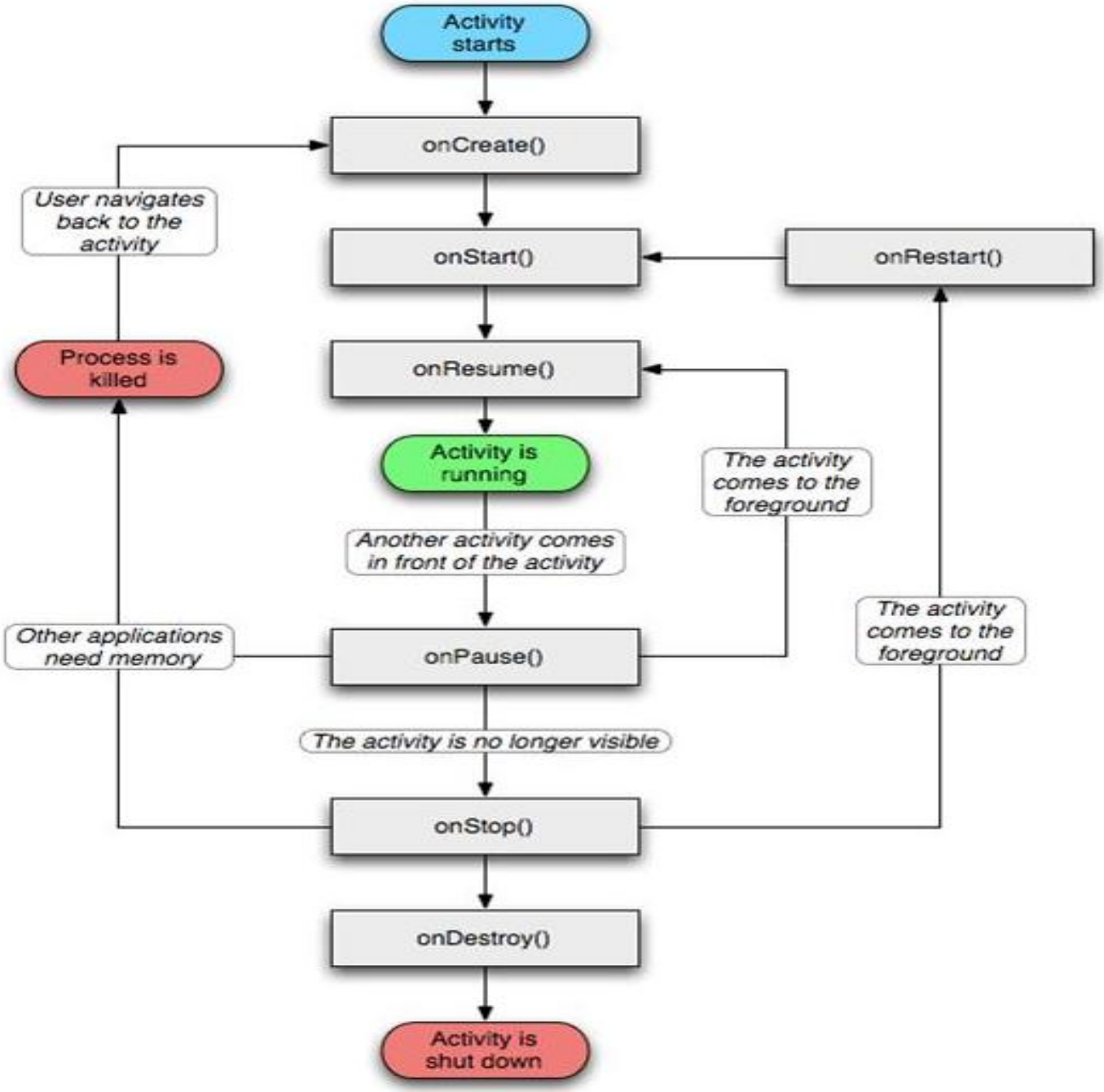
The Activity Life Cycle *

- 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 if necessary, 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()`**



Fragments

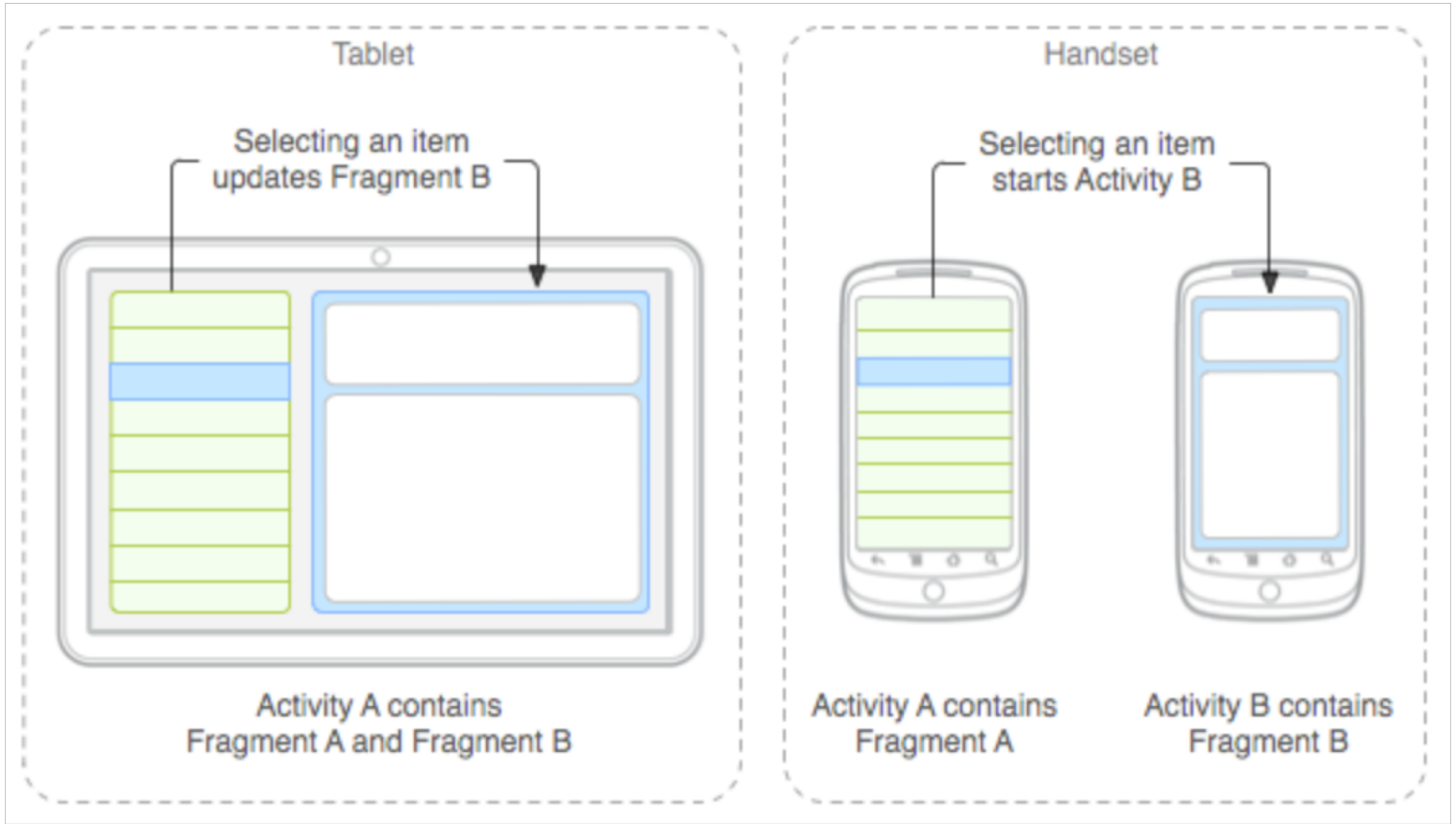
- Fragments represents a behaviour or a portion of a user interface *in an **Activity***.
- Introduced in Android 3.0 (API level 11), primarily supports more dynamic and flexible UI designs on larger screens.
- You can combine multiple fragments in a single activity to build a multi-pane UI and *reuse a fragment in multiple activities*.
- Each Fragment has its own lifecycle, receives its own input events, and you can add or remove it while the activity is running.
- A fragment must always be embedded in an activity and the fragment's lifecycle is directly affected by the host activity's lifecycle.
- When you perform a ***fragment transaction***, you can also add it to a back stack that's managed by the activity.
- The back stack allows the user to reverse a fragment transaction (navigate backwards), by pressing the *Back* button.
- When you add a fragment as a part of your activity layout, it lives in a *ViewGroup* inside the activity's view hierarchy and the fragment defines its own view layout.

Designing Fragments *

- You should design each fragment as a modular and reusable activity component.
- When designing your application to support both tablets and handsets, you can ***reuse your fragments*** in different layout configurations to optimize the user experience based on the available screen space.

- For example, on a handset, it might be necessary for separate fragments to provide a single-pane UI when more than one cannot fit within the same activity. (Next Slide)

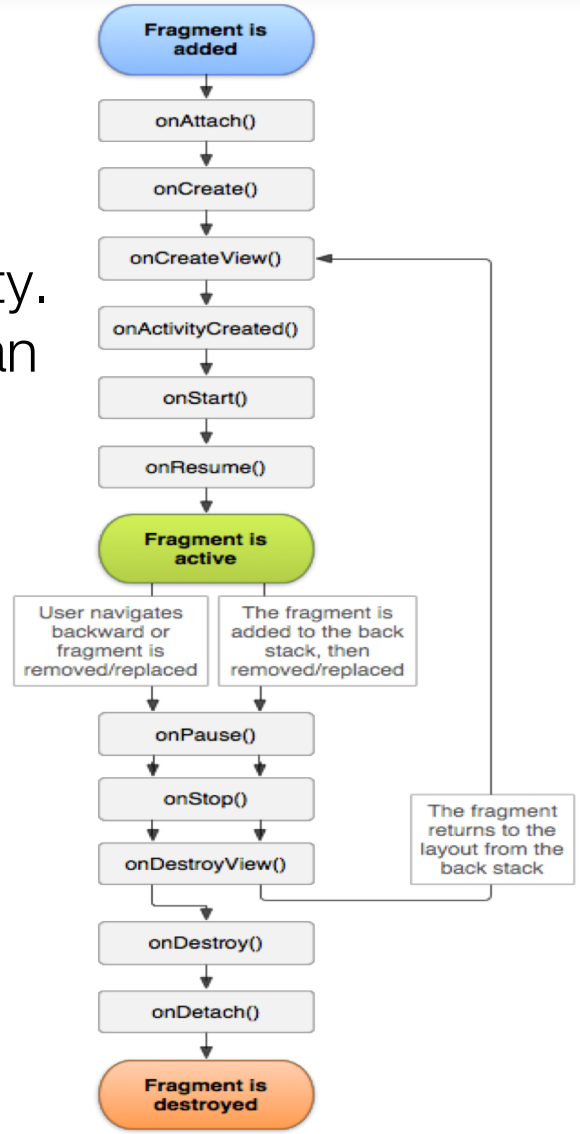
Designing Fragments

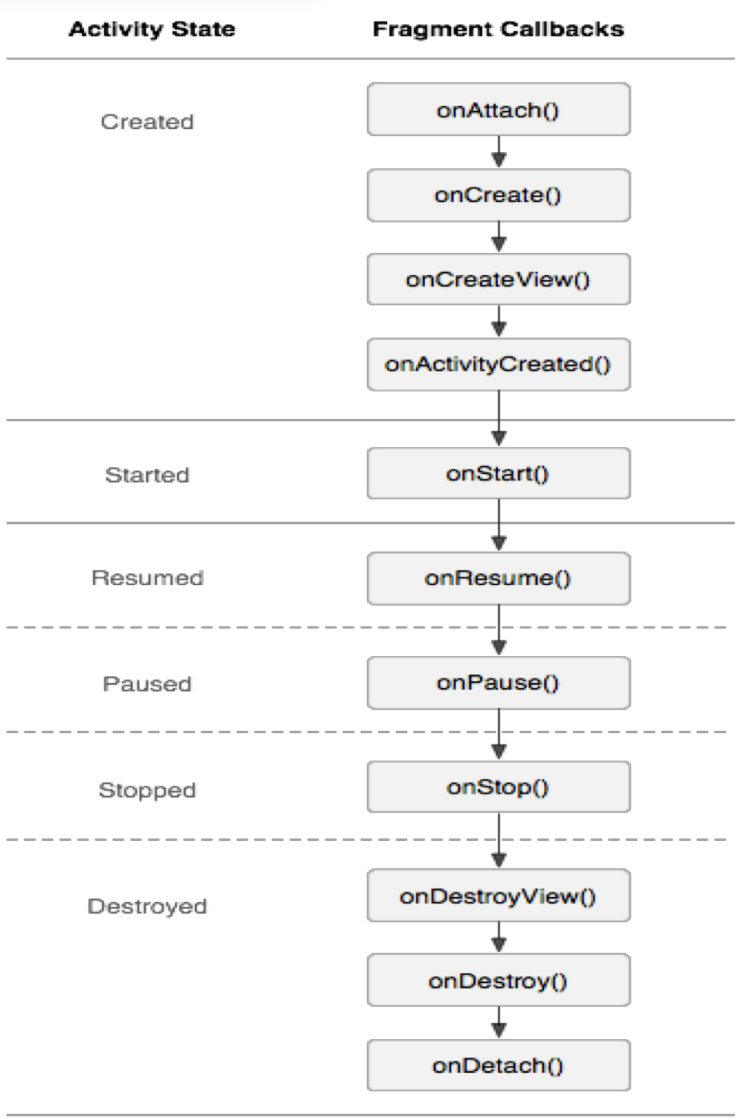


An example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.

The Fragment Life Cycle

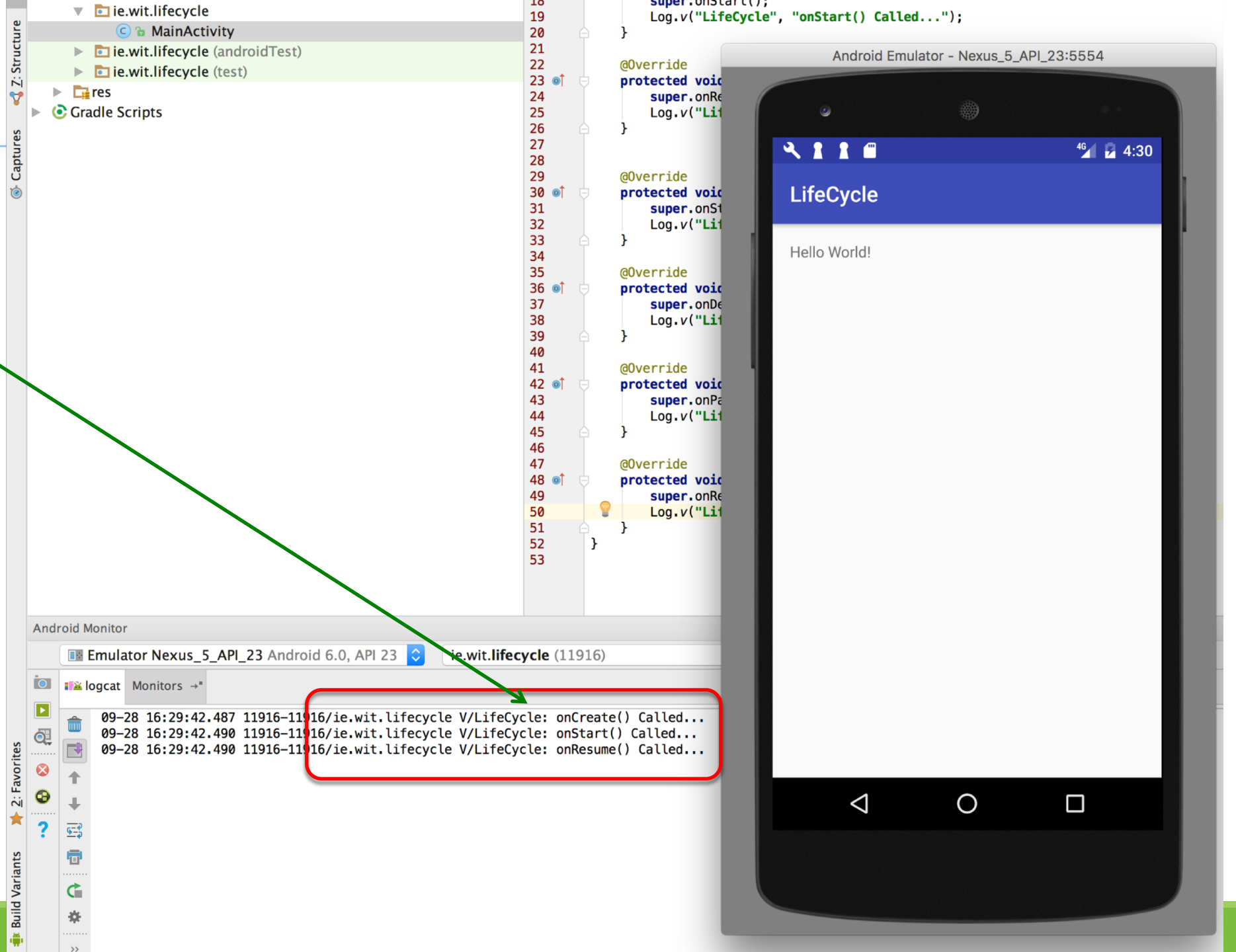
- To create a fragment, you must subclass Fragment (or an existing subclass of it).
- Has code that looks a lot like an Activity. Contains callback methods similar to an activity, such as *onCreate()*, *onStart()*, *onPause()*, and *onStop()*.
- Usually, you should implement at least *onCreate()*, *onCreateView()* and *onPause()*





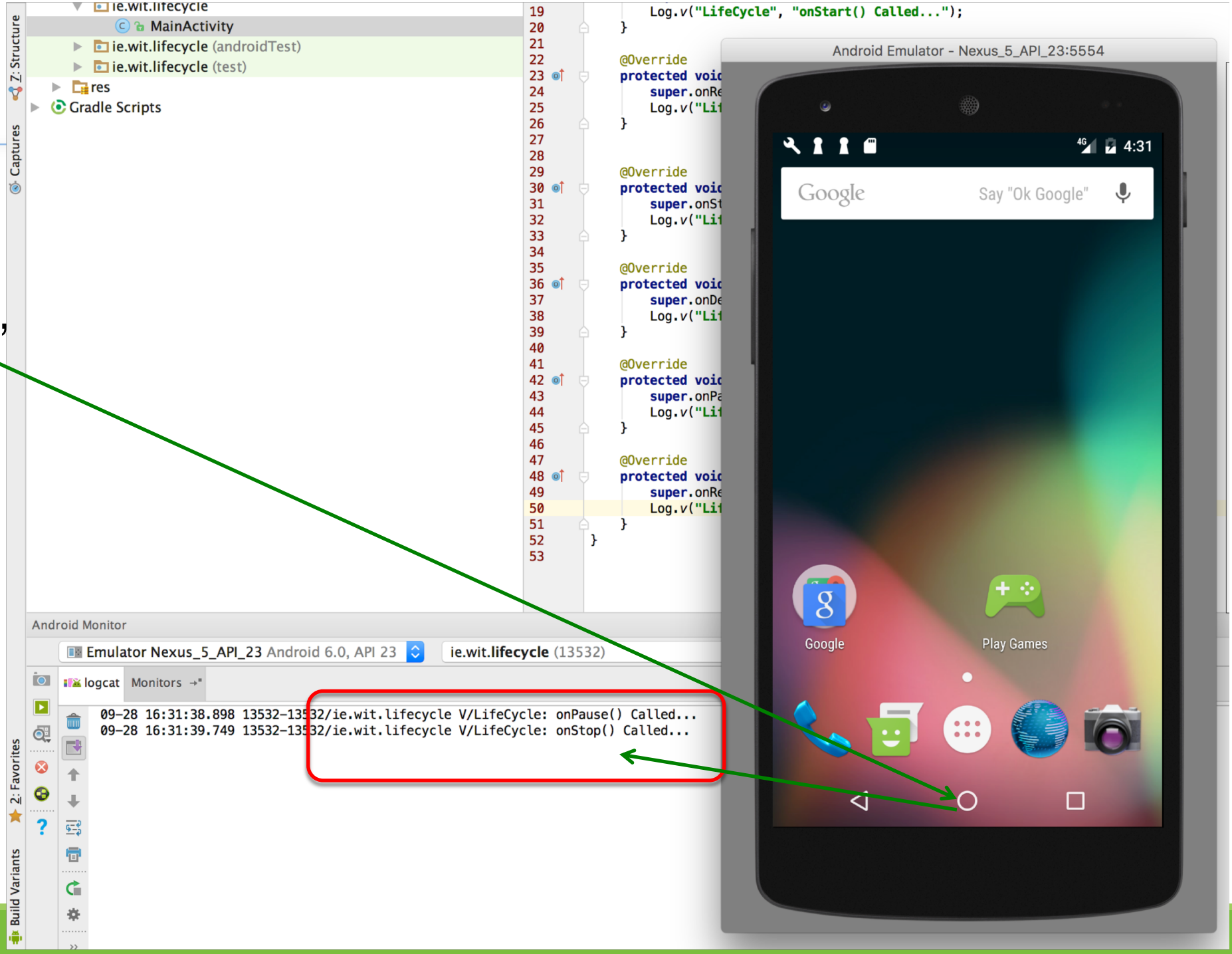
LifeCycle Example (1) *

User Launches App



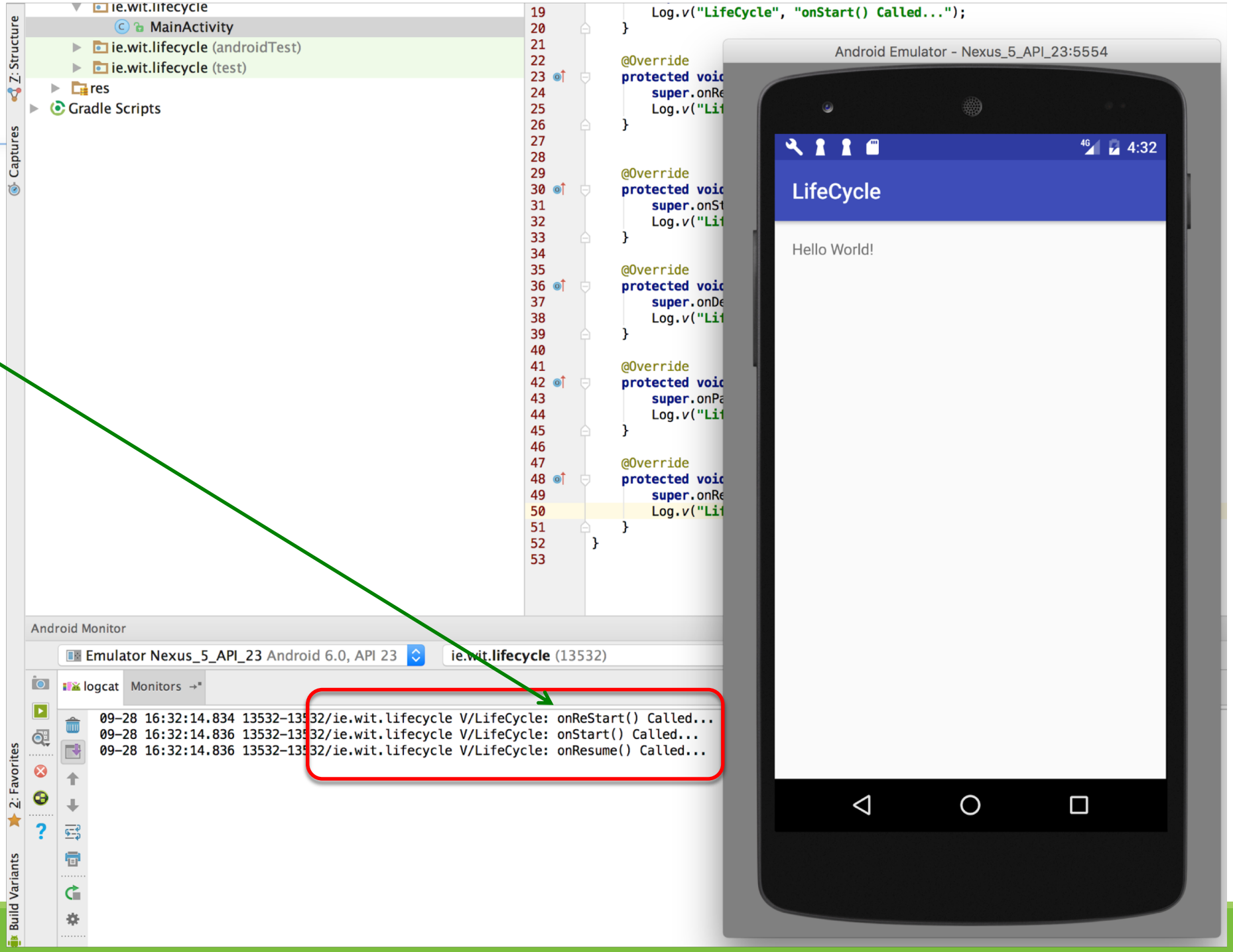
LifeCycle Example (2) *

User Selects ‘Home’



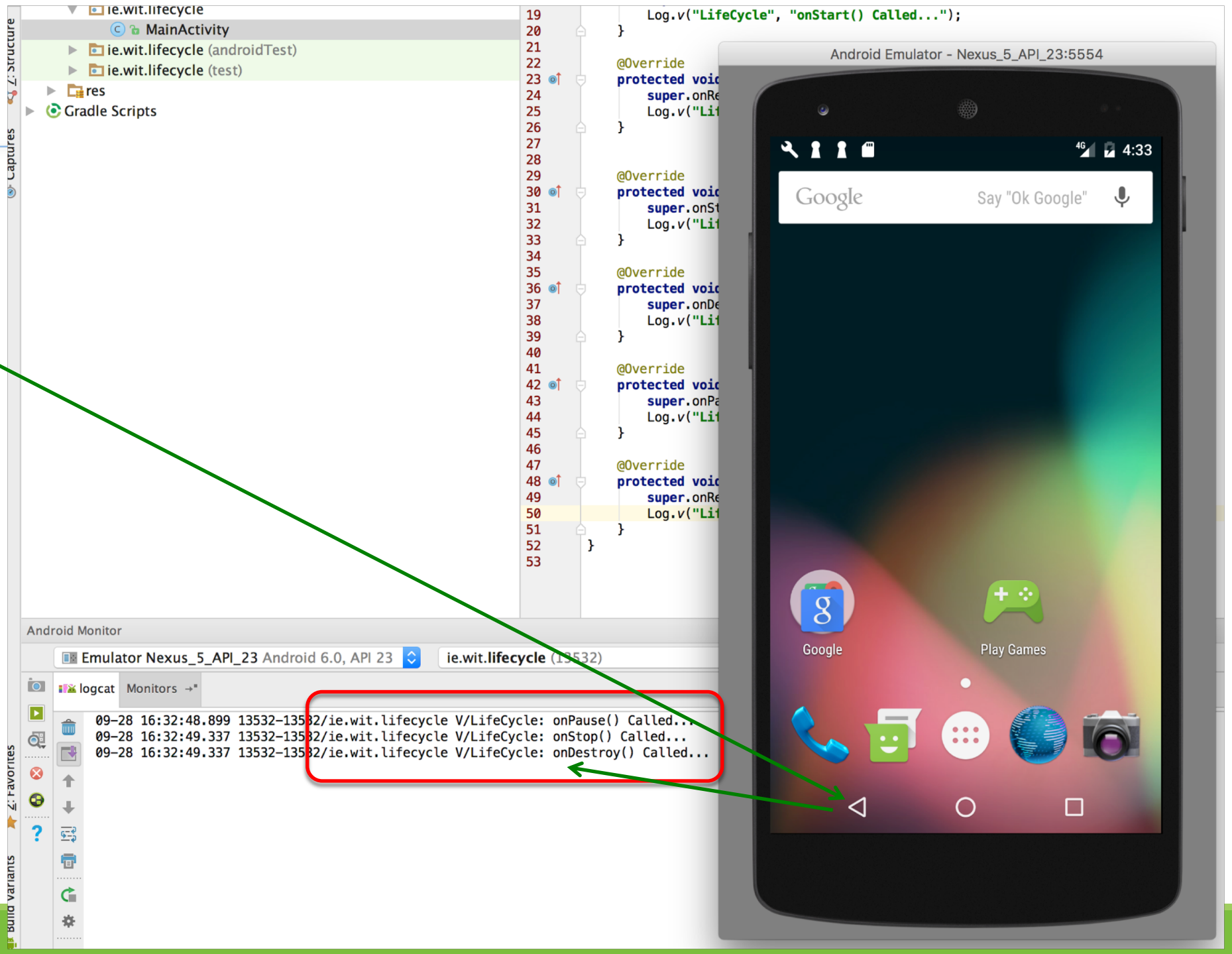
LifeCycle Example (3) *

User restarts App



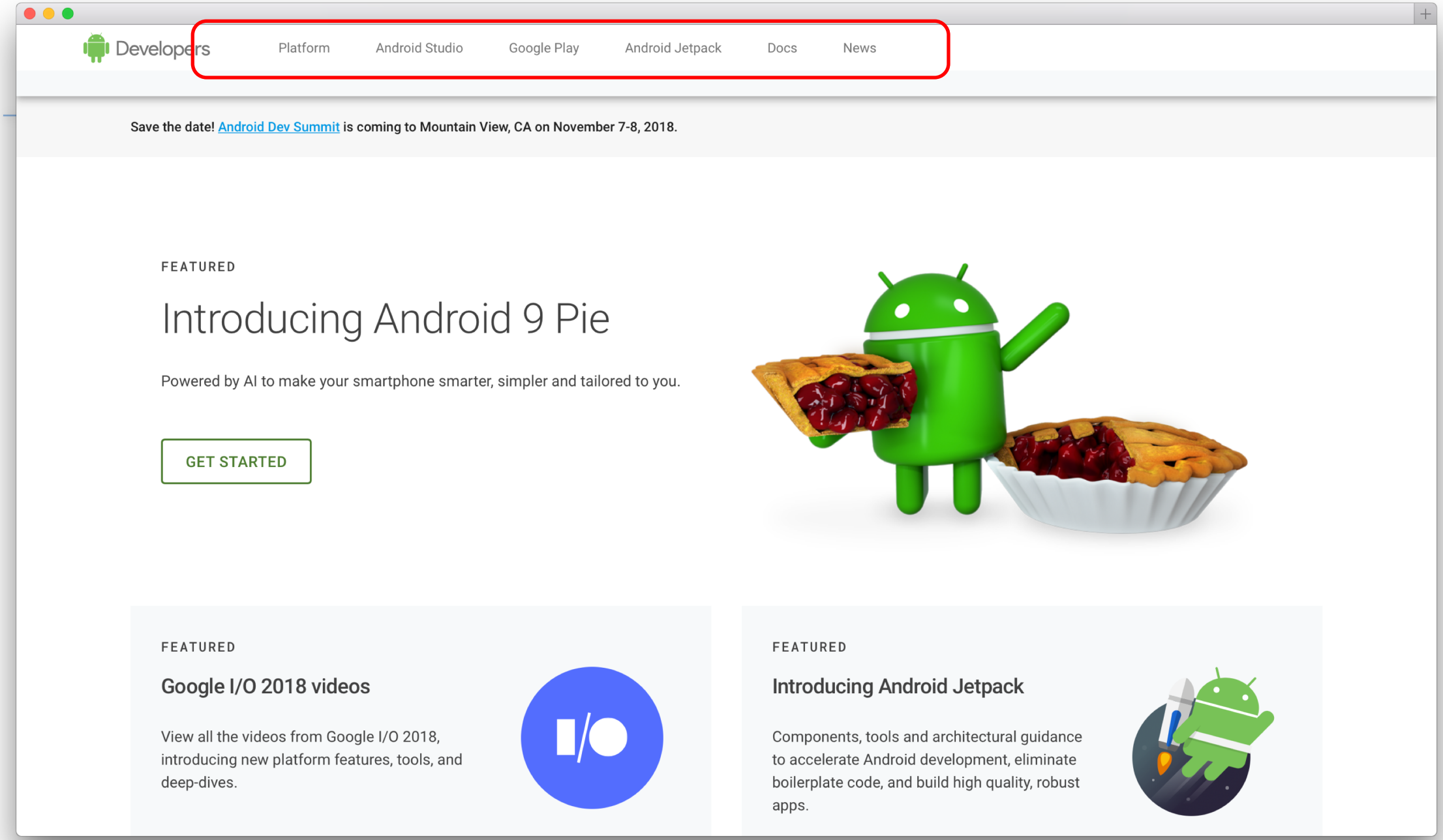
LifeCycle Example (4) *

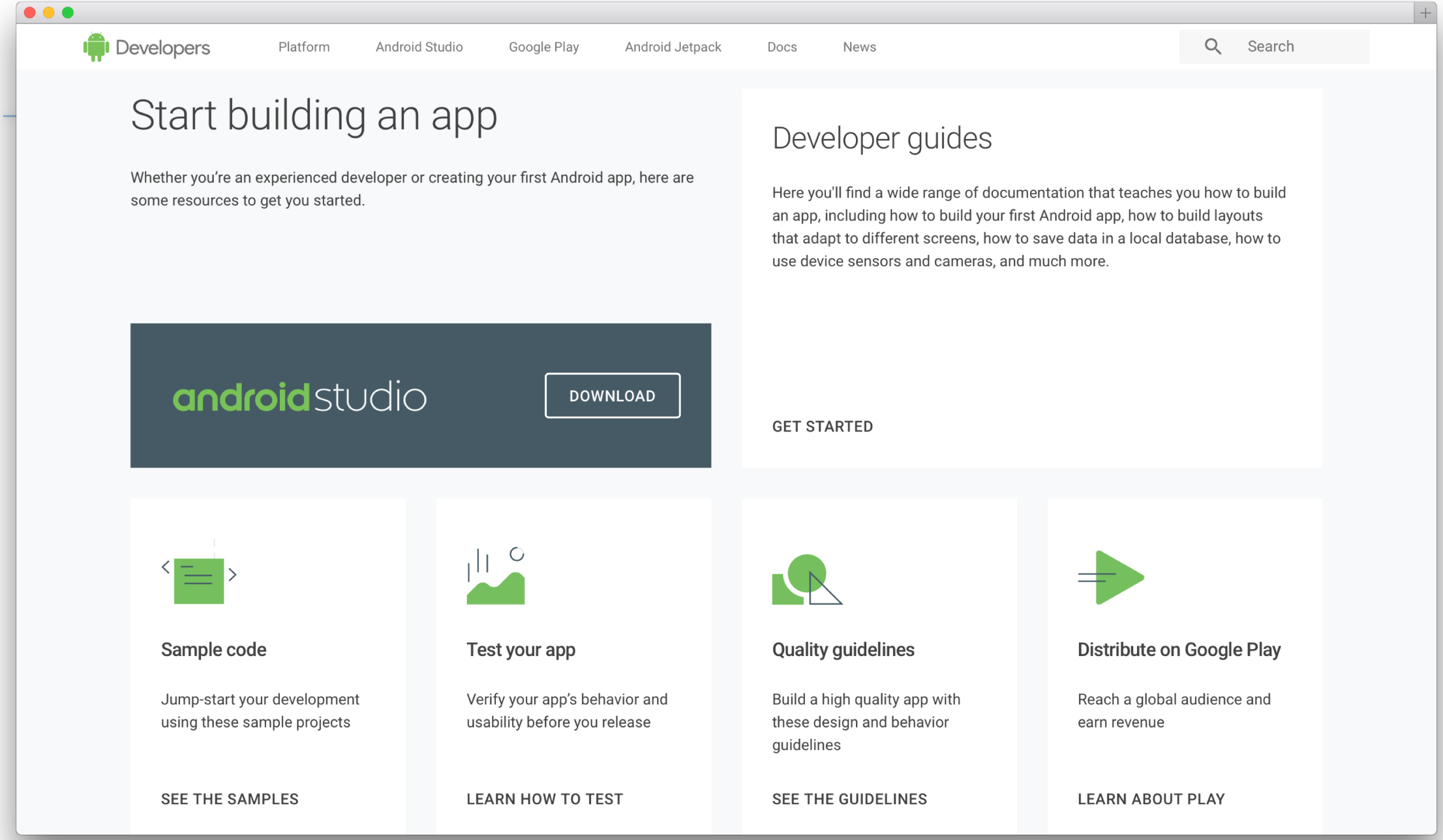
User Selects ‘Back’



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 ONLINE DEVELOPER DOCs & GUIDES (next few slides)


Developers
Platform
Android Studio
Google Play
Android Jetpack
Docs
News
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.

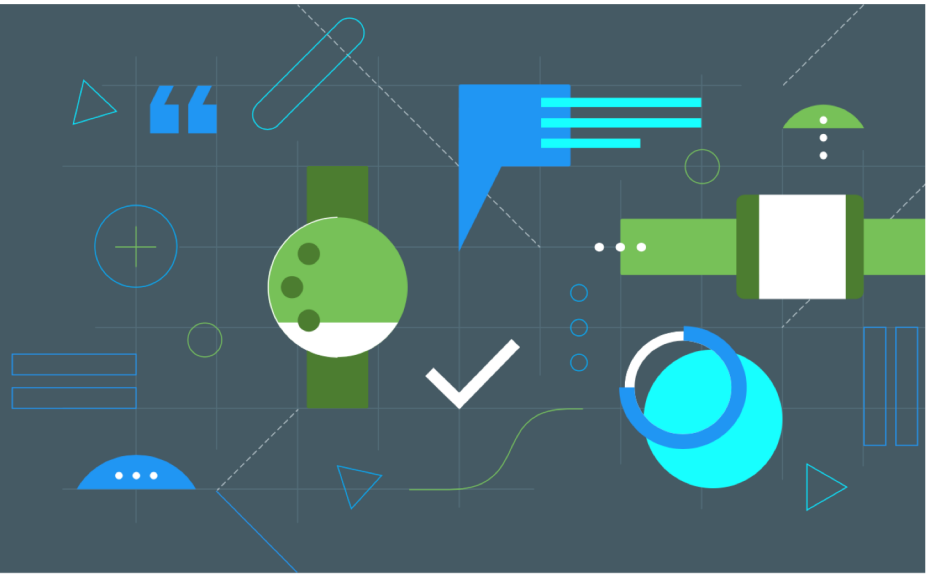

Developers
Platform
Android Studio
Google Play
Android Jetpack
Docs
News
Search
Start building an app
Whether you're an experienced developer or creating your first Android app, here are some resources to get you started.
android studio
DOWNLOAD
Developer guides
Here you'll find a wide range of documentation 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

Developers Platform Android Studio Google Play Android Jetpack Docs News Search

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

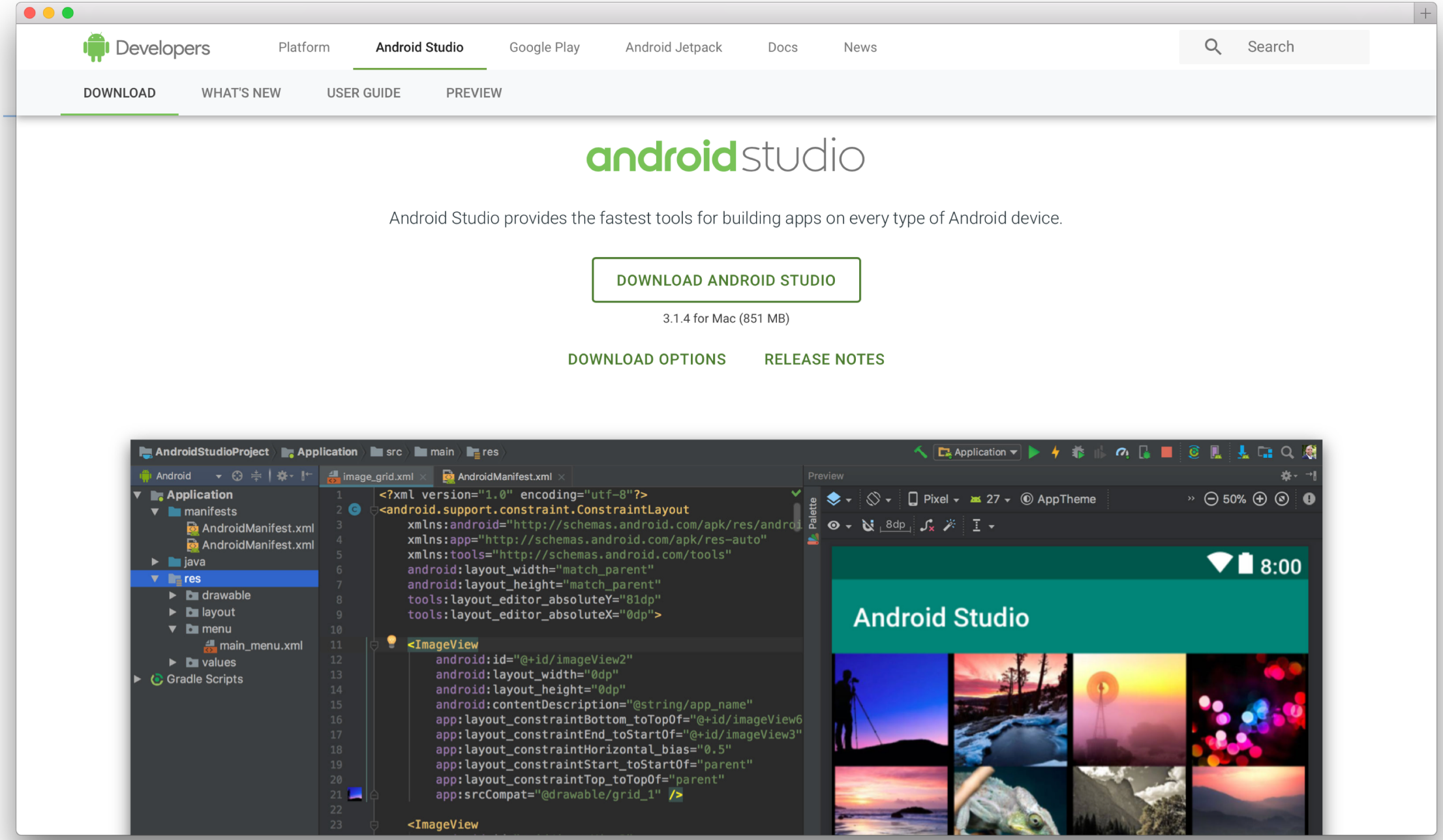
Wear OS

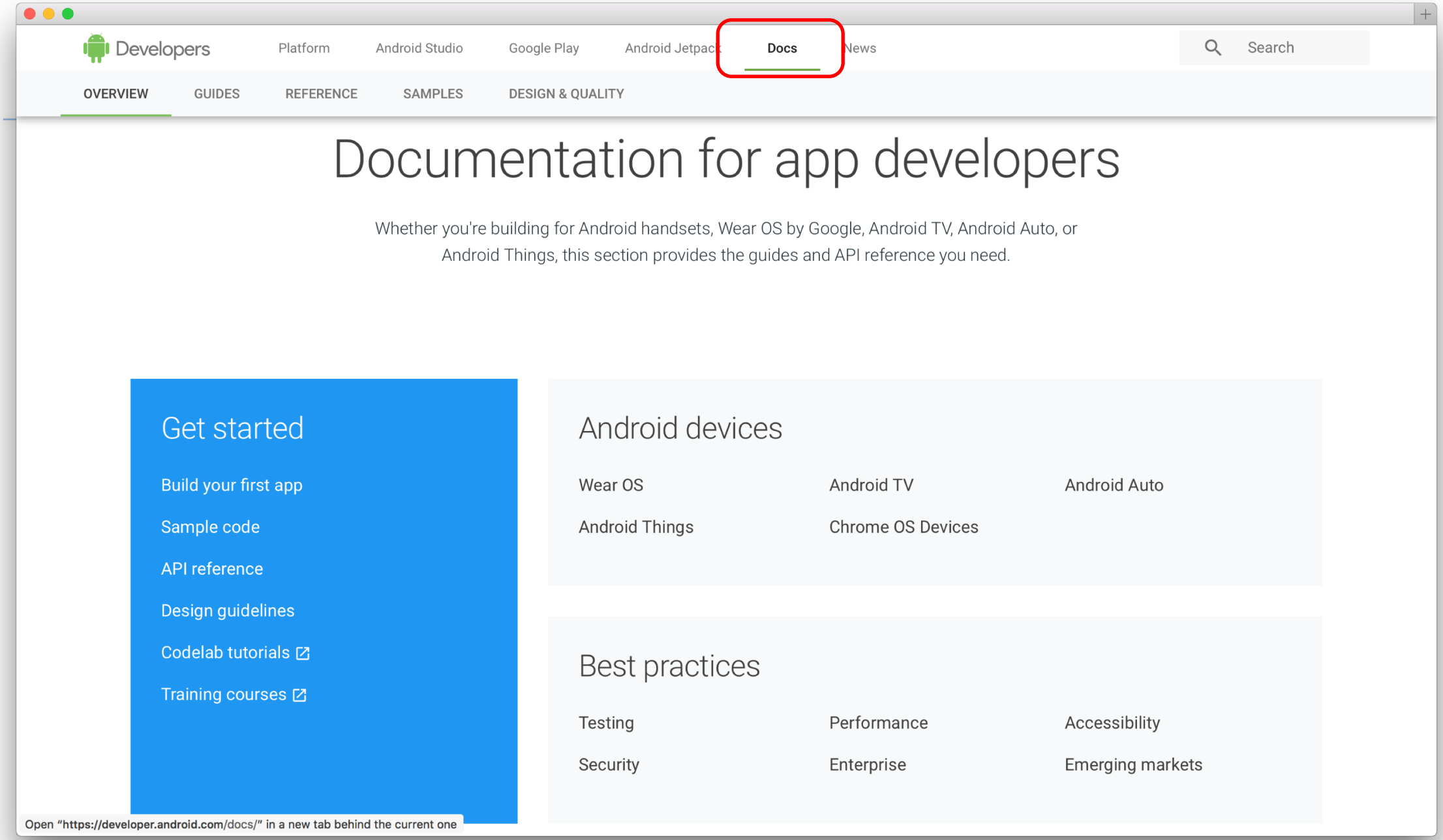
PLATFORM

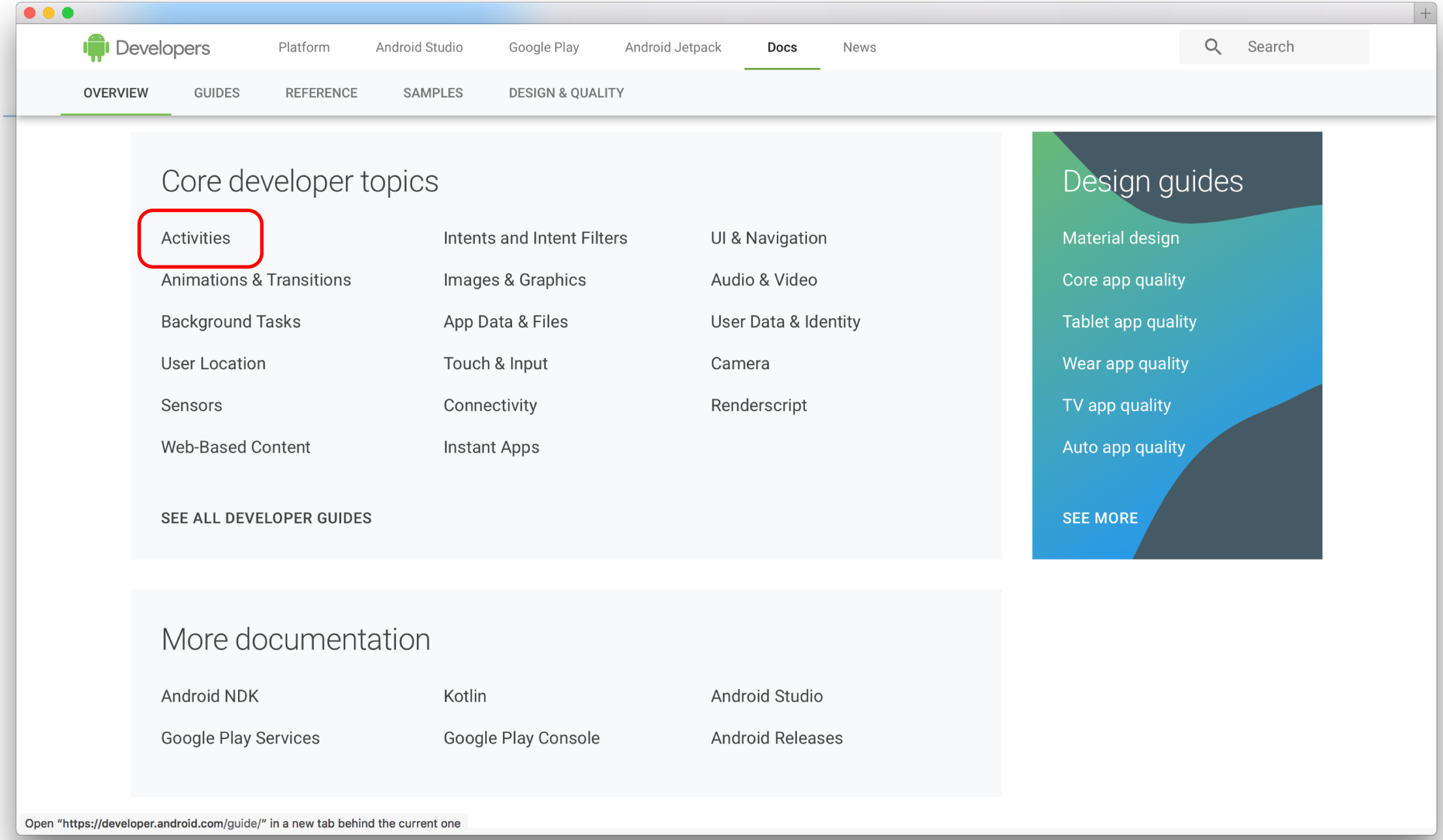
TV

PLATFORM

Auto


Developers
Platform
Android Studio
Google Play
Android Jetpack
Docs
News
Search
DOWNLOAD
WHAT'S NEW
USER GUIDE
PREVIEW
android studio
Android Studio provides the fastest tools for building apps on every type of Android device.
DOWNLOAD ANDROID STUDIO
3.1.4 for Mac (851 MB)
DOWNLOAD OPTIONS
RELEASE NOTES
AndroidStudioProject
Application
src
main
res
Android
image_grid.xml
AndroidManifest.xml
Application
manifests
AndroidManifest.xml
AndroidManifest.xml
java
res
drawable
layout
menu
main_menu.xml
values
Gradle Scripts
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/androi
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:layout_editor_absoluteY="81dp"
tools:layout_editor_absoluteX="0dp">
<ImageView
android:id="@+id/imageView2"
android:layout_width="0dp"
android:layout_height="0dp"
android:contentDescription="@string/app_name"
app:layout_constraintBottom_toTopOf="@+id/imageView6
app:layout_constraintEnd_toStartOf="@+id/imageView3"
app:layout_constraintHorizontal_bias="0.5"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:srcCompat="@drawable/grid_1" />
<ImageView
Preview
Palette
Pixel
27
AppTheme
50%
8dp
8:00
Android Studio


Developers
Platform
Android Studio
Google Play
Android Jetpack
Docs
News
Search
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
Android devices
Wear OS
Android TV
Android Auto
Android Things
Chrome OS Devices
Best practices
Testing
Performance
Accessibility
Security
Enterprise
Emerging markets
Open "https://developer.android.com/docs/" in a new tab behind the current one


Developers
Platform
Android Studio
Google Play
Android Jetpack
Docs
News
Search
OVERVIEW
GUIDES
REFERENCE
SAMPLES
DESIGN & QUALITY
Core developer topics
Activities
Intents and Intent Filters
UI & Navigation
Animations & Transitions
Images & Graphics
Audio & Video
Background Tasks
App Data & Files
User Data & Identity
User Location
Touch & Input
Camera
Sensors
Connectivity
Renderscript
Web-Based Content
Instant Apps
SEE ALL DEVELOPER GUIDES
Design guides
Material design
Core app quality
Tablet app quality
Wear app quality
TV app quality
Auto app quality
SEE MORE
More documentation
Android NDK
Kotlin
Android Studio
Google Play Services
Google Play Console
Android Releases
Open "https://developer.android.com/guide/" in a new tab behind the current one

Developers Platform Android Studio Google Play Android Jetpack Docs News Search

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

Activities

☆ ☆ ☆ ☆ ☆

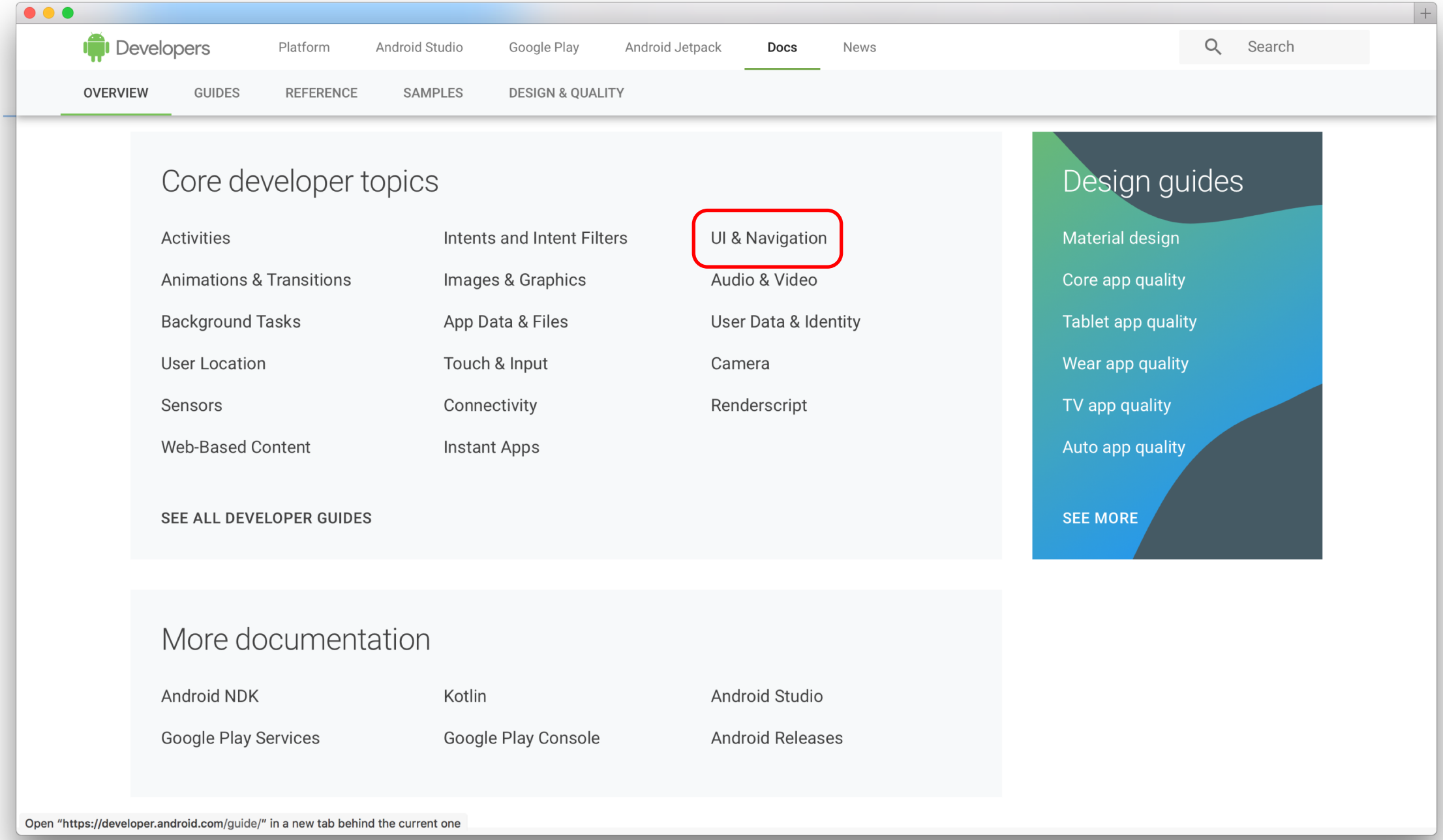
Activities are one of the fundamental building blocks of apps on the Android platform. They serve as the entry point for a user's interaction with an app, and are also central to how a user navigates within an app (as with the Back button) or between apps (as with the Recents button).

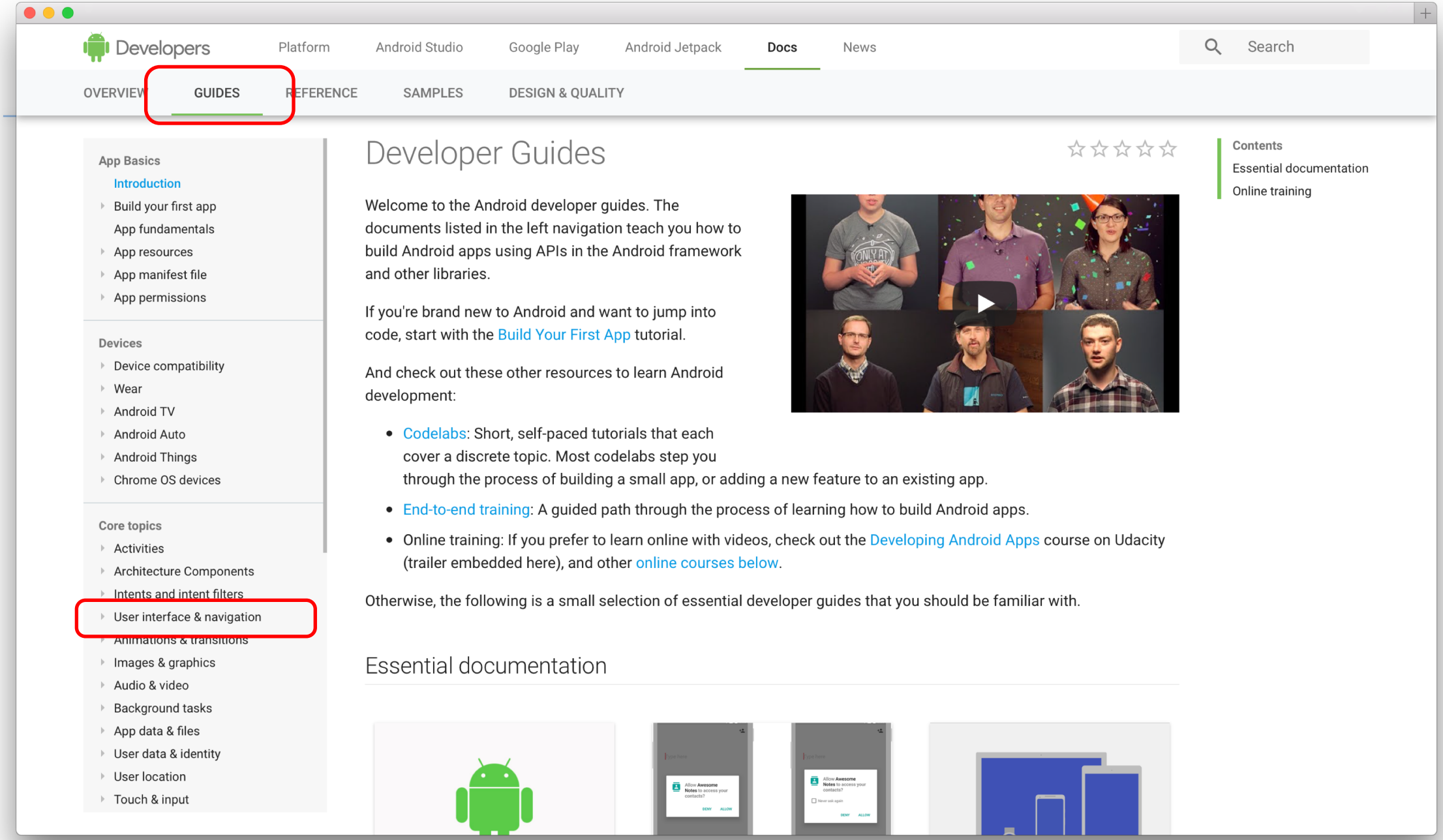
Skillfully managing activities allows you to ensure that, for example:

- Orientation changes take place smoothly without disrupting the user experience.
- User data is not lost during activity transitions.
- The system kills processes when it's appropriate to do so.

This section begins by providing an introduction to the concept of activities. It goes on to describe the activity lifecycle in detail. Next, it discusses state changes and how to accommodate them. After that, this section talks about the relationship between activities and intra- and inter-app navigation. Last, this section explains the relationship between activities and the processes that host them.

Documentation


Developers
Platform
Android Studio
Google Play
Android Jetpack
Docs
News
Search
OVERVIEW
GUIDES
REFERENCE
SAMPLES
DESIGN & QUALITY
Core developer topics
Activities
Intents and Intent Filters
UI & Navigation
Animations & Transitions
Images & Graphics
Audio & Video
Background Tasks
App Data & Files
User Data & Identity
User Location
Touch & Input
Camera
Sensors
Connectivity
Renderscript
Web-Based Content
Instant Apps
SEE ALL DEVELOPER GUIDES
Design guides
Material design
Core app quality
Tablet app quality
Wear app quality
TV app quality
Auto app quality
SEE MORE
More documentation
Android NDK
Kotlin
Android Studio
Google Play Services
Google Play Console
Android Releases
Open "https://developer.android.com/guide/" in a new tab behind the current one


Developers
Platform
Android Studio
Google Play
Android Jetpack
Docs
News
Search
OVERVIEW
GUIDES
REFERENCE
SAMPLES
DESIGN & QUALITY
App Basics
Introduction
Build your first app
App fundamentals
App resources
App manifest file
App permissions
Devices
Device compatibility
Wear
Android TV
Android Auto
Android Things
Chrome OS devices
Core topics
Activities
Architecture Components
Intents and intent filters
User interface & navigation
Animations & transitions
Images & graphics
Audio & video
Background tasks
App data & files
User data & identity
User location
Touch & input
Developer Guides
Welcome to the Android developer guides. The documents listed in the left navigation teach you how to build Android apps using APIs in the Android framework and other libraries.
If you're brand new to Android and want to jump into code, start with the Build Your First App tutorial.
And check out these other resources to learn Android development:
Codelabs: Short, self-paced tutorials that each cover a discrete topic. Most codelabs step you through the process of building a small app, or adding a new feature to an existing app.
End-to-end training: A guided path through the process of learning how to build Android apps.
Online training: If you prefer to learn online with videos, check out the Developing Android Apps course on Udacity (trailer embedded here), and other online courses below.
Otherwise, the following is a small selection of essential developer guides that you should be familiar with.
Contents
Essential documentation
Online training
Essential documentation

Developers | Platform | Android Studio | Google Play | Android Jetpack | Docs | News | Search

Documentation

OVERVIEW | GUIDES | REFERENCE | SAMPLES | DESIGN & QUALITY

Save the date! Android Dev Summit **is coming to Mountain View, CA on November 7-8, 2018.**

- Architecture Components
- Intents and intent filters
- User interface & navigation
  - Overview
  - Layouts
  - Look and feel
  - Notifications
  - Add the app bar
  - Control the system UI visibility
  - Designing effective navigation
  - Implementing effective navigation
  - Slide between fragments using ViewPager
  - Supporting swipe-to-refresh
  - Toasts overview
  - Pop-up messages overview
  - Dialogs
  - Menus
  - Settings
  - Search
  - Copy and paste
  - Drag and drop
  - Creating backward-compatible

User Interface & Navigation

☆☆☆☆☆

Your app's user interface is everything that the user can see and interact with. Android provides a variety of pre-built UI components such as structured layout objects and UI controls that allow you to build the graphical user interface for your app. Android also provides other UI modules for special interfaces such as dialogs, notifications, and menus.

To get started, read Layouts.

Documentation

Layouts

Notifications Overview

Add the app bar

Control the system UI visibility

Contents
- Documentation
- Videos

Developers | Platform | Android Studio | Google Play | Android Jetpack | Docs | News | Search

Documentation

OVERVIEW | GUIDES | REFERENCE | SAMPLES | DESIGN & QUALITY

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

Layouts

A layout defines the structure for a user interface in your app, such as in an activity. All elements in the layout are built using a hierarchy of `View` and `ViewGroup` objects. A `View` usually draws something the user can see and interact with. Whereas a `ViewGroup` is an invisible container that defines the layout structure for `View` and other `ViewGroup` objects, as shown in figure 1.

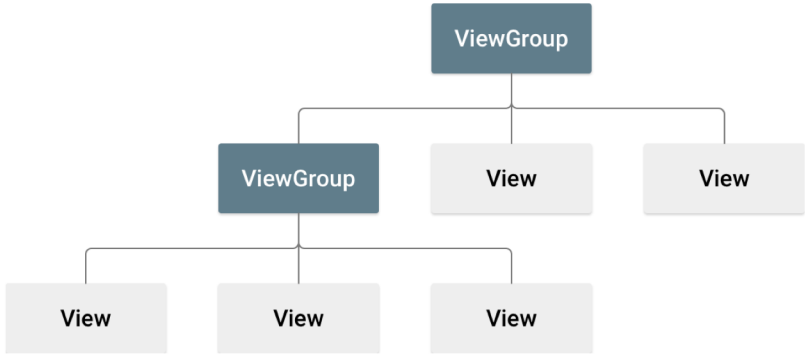


Figure 1. Illustration of a view hierarchy, which defines a UI layout

The `View` objects are usually called "widgets" and can be one of many subclasses, such as `Button` or `TextView`. The `ViewGroup` objects are usually called "layouts" can be one of many types that provide a different layout structure, such as `LinearLayout` or `ConstraintLayout` .

You can declare a layout in two ways:

Developers | Platform | Android Studio | Google Play | Android Jetpack | Docs | News | Search

Documentation

OVERVIEW | GUIDES | REFERENCE | SAMPLES | DESIGN & QUALITY

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

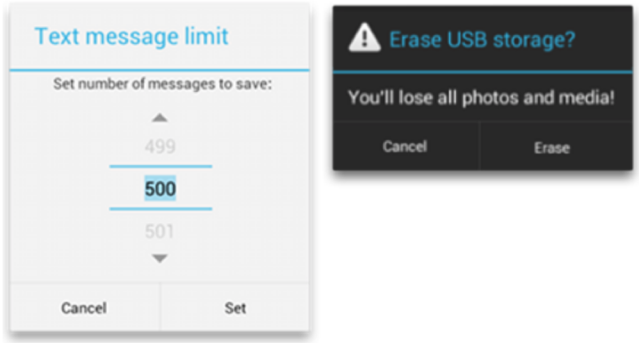
- Supporting swipe-to-refresh
- Toasts overview
- Pop-up messages overview
- Dialogs
- Menus
- Settings
- Search
- Copy and paste
- Drag and drop
- Creating backward-compatible UIs
- Animations & transitions
- Images & graphics
- Audio & video
- Background tasks
- App data & files
- User data & identity
- User location
- Touch & input
- Camera
- Sensors
- Connectivity
- Renderscript

Dialogs

A dialog is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed.

Dialog Design

For information about how to design your dialogs, including recommendations for language, read the Dialogs design guide.



The `Dialog` class is the base class for dialogs, but you should avoid instantiating `Dialog` directly. Instead, use one of the following subclasses:

`AlertDialog`

Contents

Developers | Platform | Android Studio | Google Play | Android Jetpack | Docs | News | Search

Documentation

OVERVIEW | GUIDES | REFERENCE | SAMPLES | DESIGN & QUALITY

Save the date! Android Dev Summit **is coming to Mountain View, CA on November 7-8, 2018.**

Core topics
- Activities
- Architecture Components
- Intents and intent filters
- User interface & navigation
  - Overview
  - Layouts
  - Look and feel
    - Material design
    - Styles and themes
    - Adaptive icons
    - Add a floating action button
    - Create shadows and clip views
    - Text
    - Buttons
    - Checkboxes
    - Radio buttons
    - Toggle buttons
    - Spinners
    - Pickers
    - Tooltips
  - Notifications

Buttons

A button consists of text or an icon (or both text and an icon) that communicates what action occurs when the user touches it.

Depending on whether you want a button with text, an icon, or both, you can create the button in your layout in three ways:

- With text, using the `Button` class:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    ... />
```

- With an icon, using the `ImageButton` class:

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button_icon"
    ... />
```

Contents
- Responding to Click Events
  - Using an OnClickListener
- Styling Your Button
  - Borderless button
  - Custom background
- Additional code samples

Developers | Platform | Android Studio | Google Play | Android Jetpack | Docs | News | Search

OVERVIEW | GUIDES | REFERENCE | SAMPLES | DESIGN & QUALITY

- Add a floating action button
- Create shadows and clip views
- Text
- Buttons
- Checkboxes
- Radio buttons
- Toggle buttons
- Spinners
- Pickers
- Tooltips
- Notifications
- Add the app bar
- Control the system UI visibility
- Designing effective navigation
- Implementing effective navigation
- Slide between fragments using ViewPager
- Supporting swipe-to-refresh
- Toasts overview
- Pop-up messages overview
- Dialogs
- Menus
- Settings
- Search
- Copy and paste
- Drag and drop
- Creating backward-compatible UIs

Responding to Click Events

When the user clicks a button, the `Button` object receives an on-click event.

To define the click event handler for a button, add the `android:onClick` attribute to the `<Button>` element in your XML layout. The value for this attribute must be the name of the method you want to call in response to a click event. The `Activity` hosting the layout must then implement the corresponding method.

For example, here's a layout with a button using `android:onClick`:

```
<?xml version="1.0" encoding="utf-8"?>
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

Within the `Activity` that hosts this layout, the following method handles the click event:

KOTLIN | JAVA

```
/** Called when the user touches the button */
public void sendMessage(View view) {
    // Do something in response to button click
}
```

The method you declare in the `android:onClick` attribute must have a signature exactly as shown above. Specifically, the method must:

- Be public
- Return void
- Define a `View` as its only parameter (this will be the `View` that was clicked)

Contents

Developers | Platform | Android Studio | Google Play | Android Jetpack | Docs | News | Search

Documentation

OVERVIEW | GUIDES | REFERENCE | SAMPLES | DESIGN & QUALITY

Save the date! Android Dev Summit **is coming to Mountain View, CA on November 7-8, 2018.**

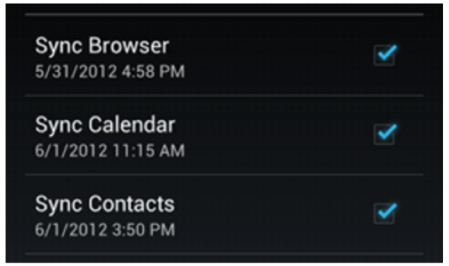
- Overview
- Layouts
- Look and feel
  - Material design
  - Styles and themes
  - Adaptive icons
  - Add a floating action button
  - Create shadows and clip views
  - Text
  - Buttons
  - Checkboxes
  - Radio buttons
  - Toggle buttons
  - Spinners
  - Pickers
  - Tooltips
- Notifications
- Add the app bar
- Control the system UI visibility
- Designing effective navigation
- Implementing effective navigation
- Slide between fragments using ViewPager

Checkboxes

☆ ☆ ☆ ☆ ☆

Contents

Checkboxes allow the user to select one or more options from a set. Typically, you should present each checkbox option in a vertical list.



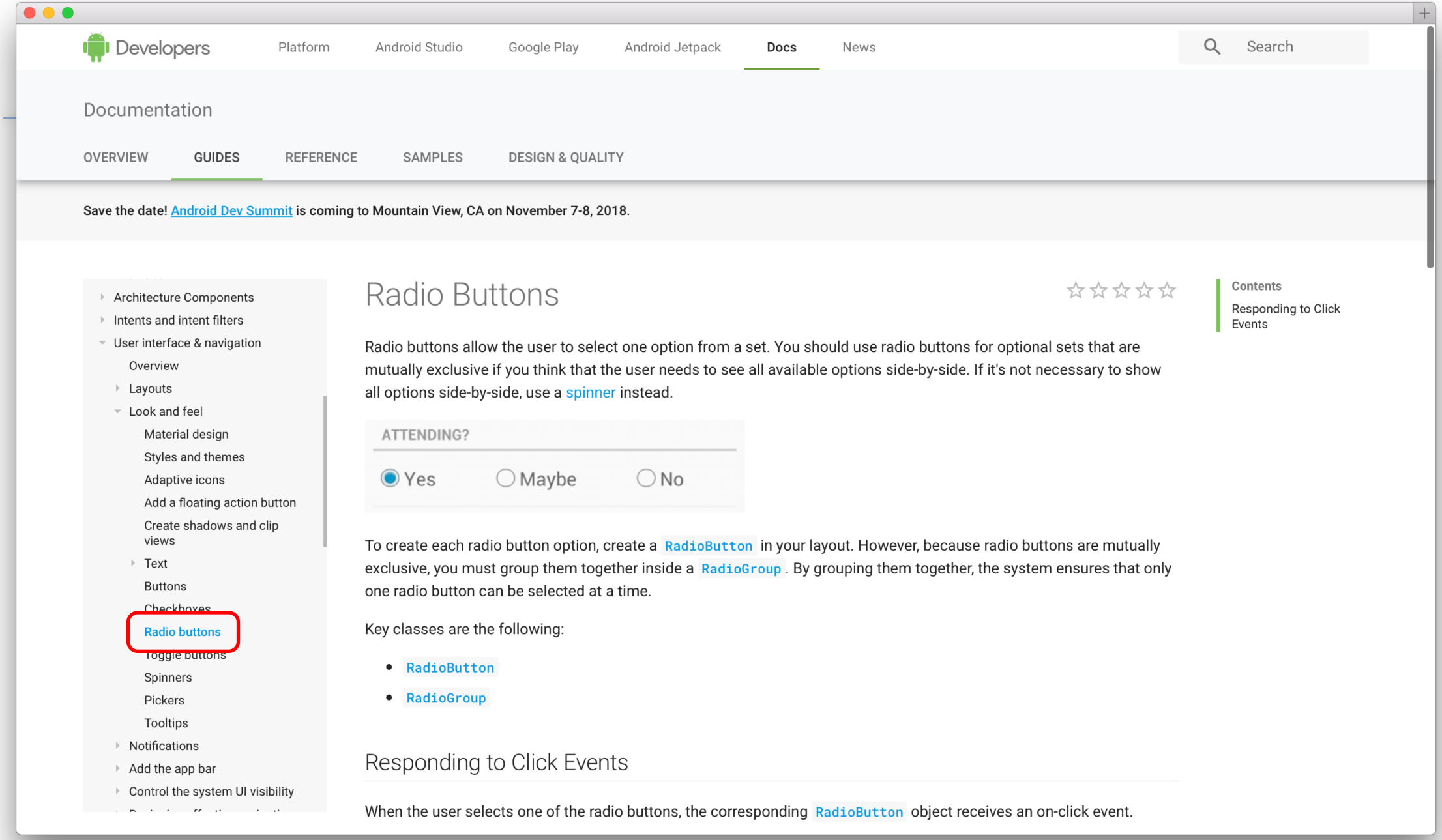
To create each checkbox option, create a `CheckBox` in your layout. Because a set of checkbox options allows the user to select multiple items, each checkbox is managed separately and you must register a click listener for each one.

A key class is the following:

- `CheckBox`

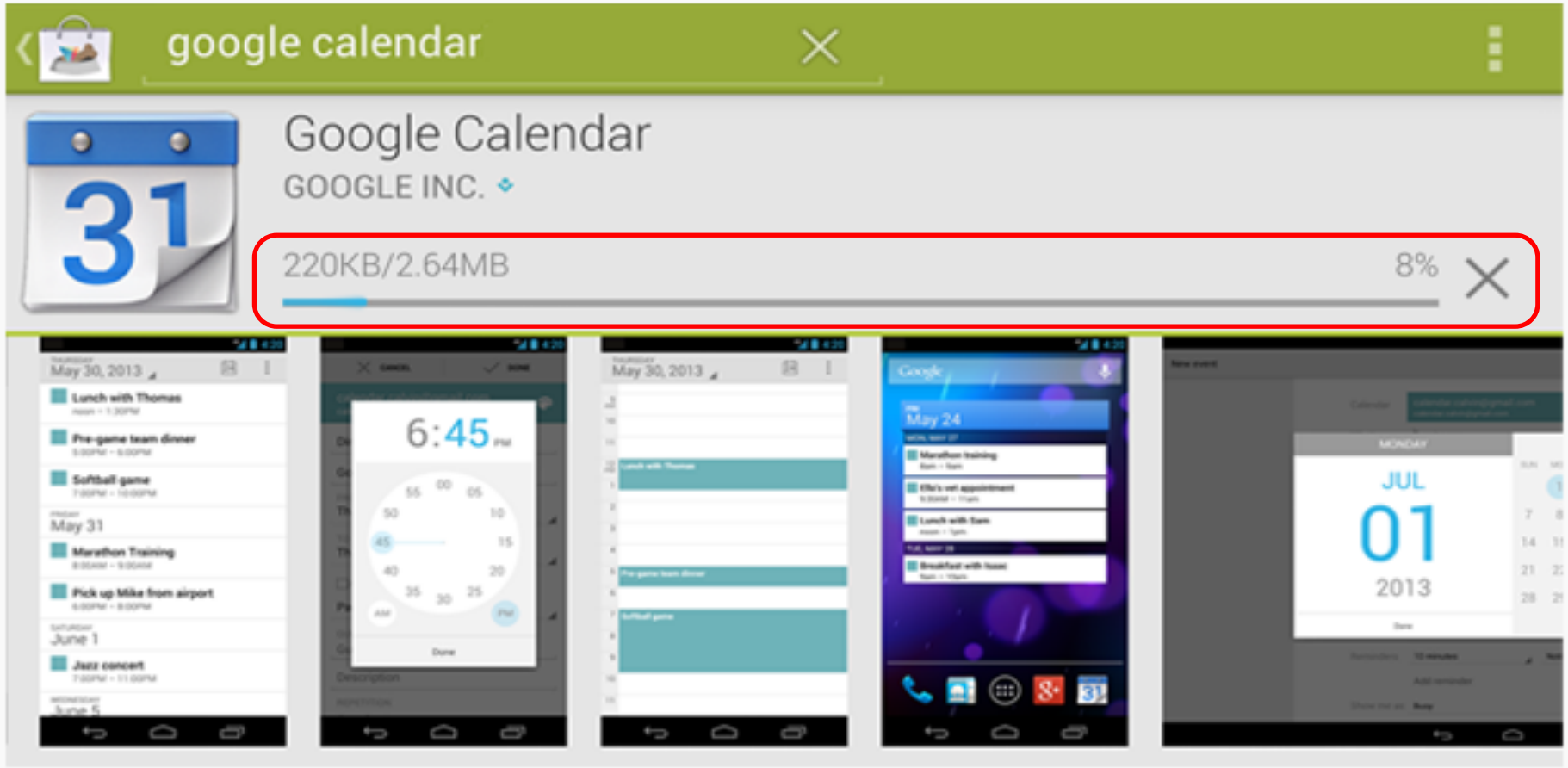
Responding to Click Events

When the user selects a checkbox, the `CheckBox` object receives an on-click event.


Developers
Platform
Android Studio
Google Play
Android Jetpack
Docs
News
Search
Documentation
OVERVIEW
GUIDES
REFERENCE
SAMPLES
DESIGN & QUALITY
Save the date! Android Dev Summit is coming to Mountain View, CA on November 7-8, 2018.
Architecture Components
Intents and intent filters
User interface & navigation
Overview
Layouts
Look and feel
Material design
Styles and themes
Adaptive icons
Add a floating action button
Create shadows and clip views
Text
Buttons
Checkboxes
Radio buttons
Toggle buttons
Spinners
Pickers
Tooltips
Notifications
Add the app bar
Control the system UI visibility
Radio Buttons
Contents
Responding to Click Events
Radio buttons allow the user to select one option from a set. You should use radio buttons for optional sets that are mutually exclusive if you think that the user needs to see all available options side-by-side. If it's not necessary to show all options side-by-side, use a spinner instead.
ATTENDING?
Yes
Maybe
No
To create each radio button option, create a RadioButton in your layout. However, because radio buttons are mutually exclusive, you must group them together inside a RadioGroup. By grouping them together, the system ensures that only one radio button can be selected at a time.
Key classes are the following:
RadioButton
RadioGroup
Responding to Click Events
When the user selects one of the radio buttons, the corresponding RadioButton object receives an on-click event.

Progress bars

Progress bars are for situations where the percentage completed can be determined. They give users a quick sense of how much longer an operation will take.



A progress bar should always fill from 0% to 100% and never move backwards to a lower value. If multiple operations are happening in sequence, use the progress bar to represent the delay as a whole, so that when the bar reaches 100%, it doesn't return back to 0%.

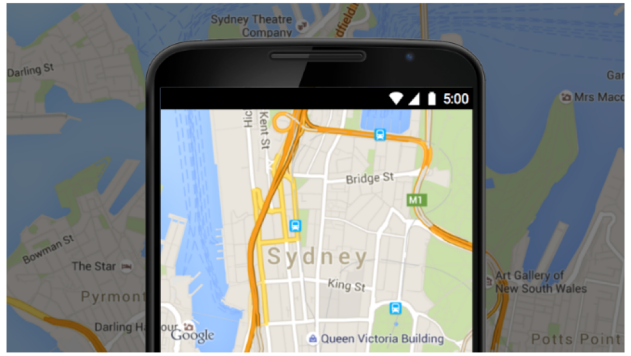
Google Maps Android API

Add Google Maps to your Android app.

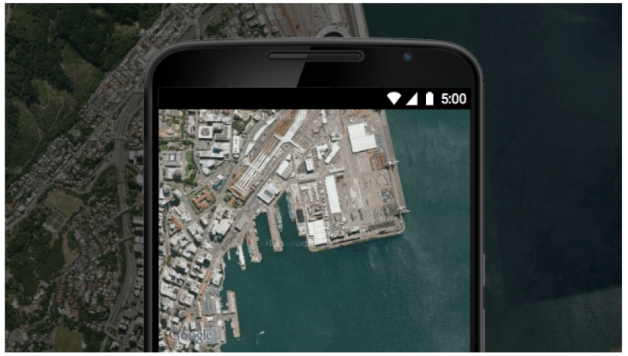
GET A KEY | VIEW PRICING AND PLANS

The best of Google Maps for every Android app

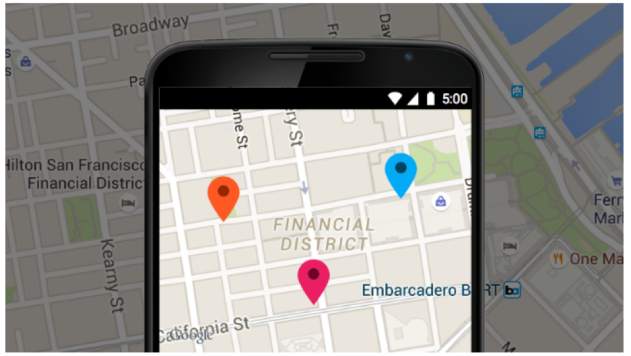
Build a custom map for your Android app using 3D buildings, indoor floor plans and more.



Maps

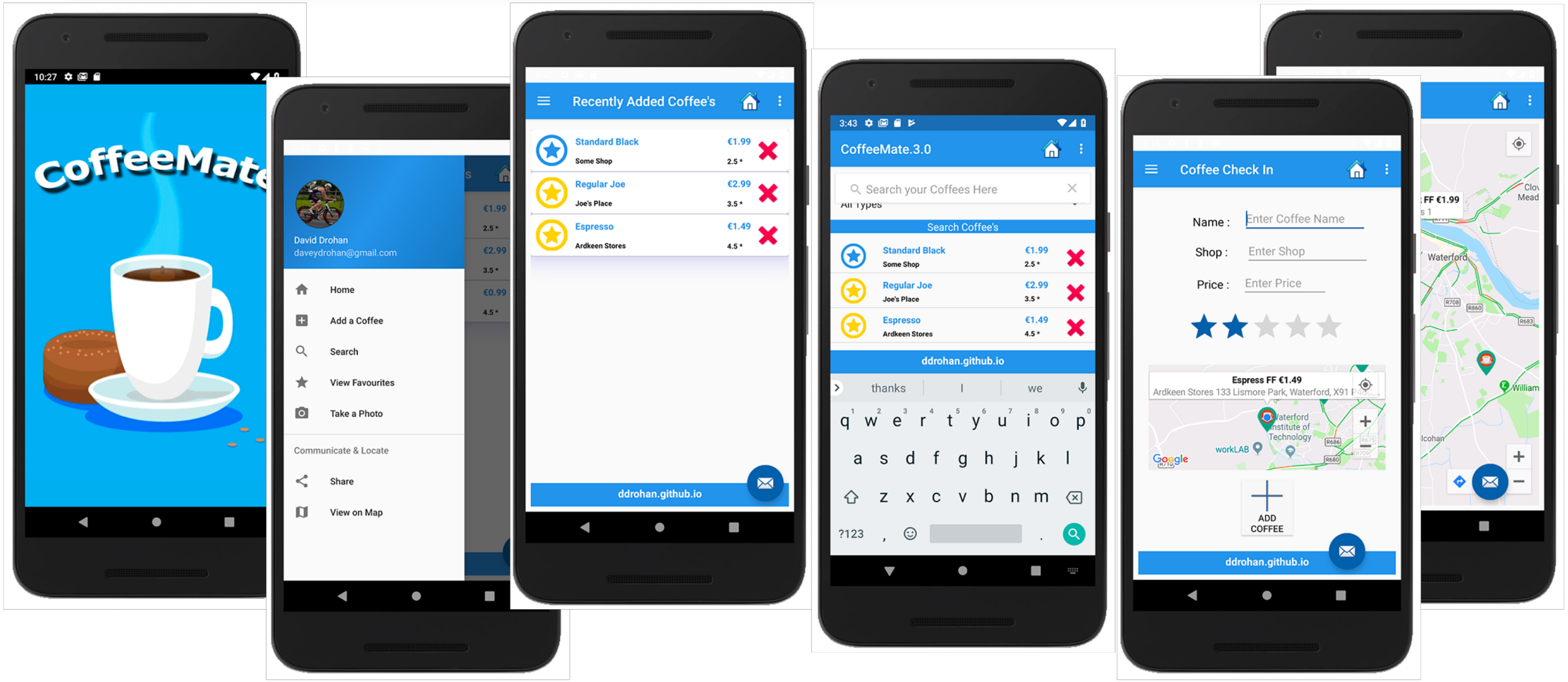


Imagery



Customization

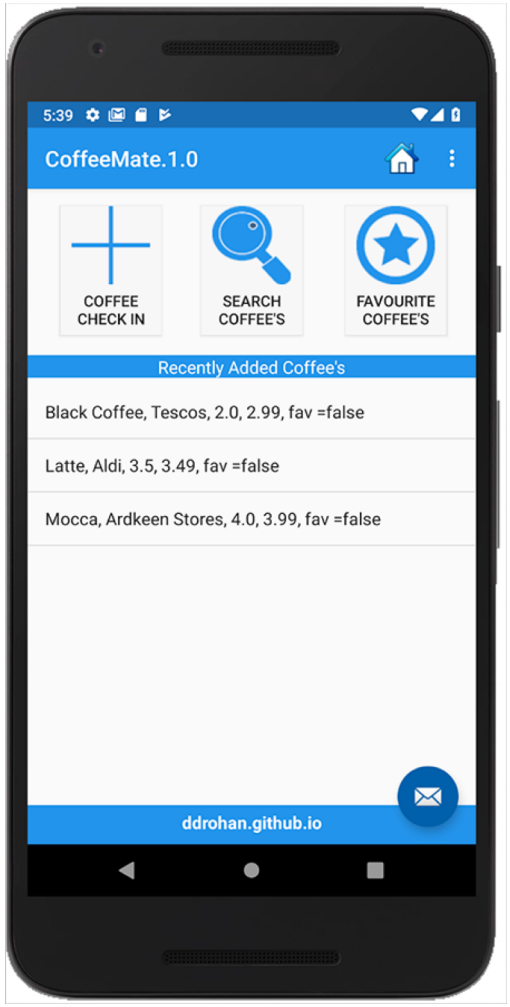
Ultimate Case Study

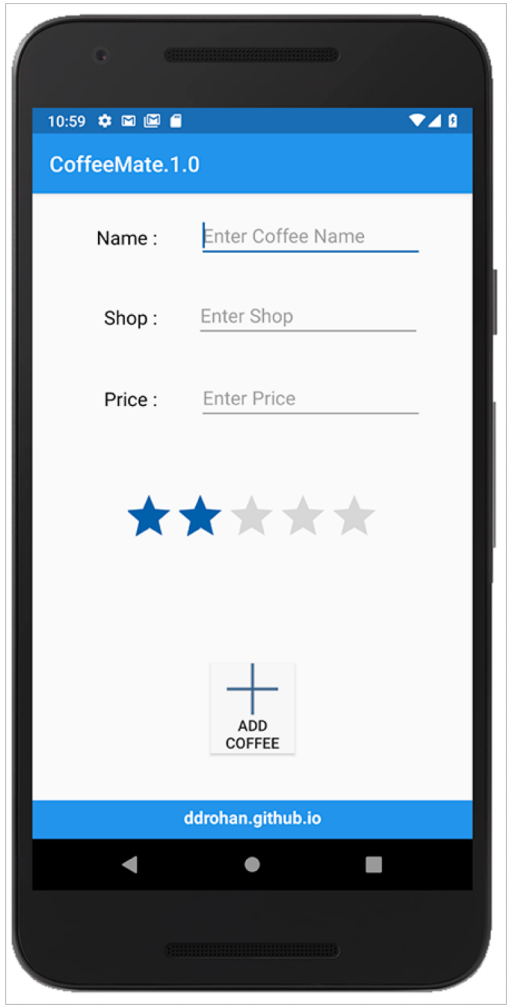


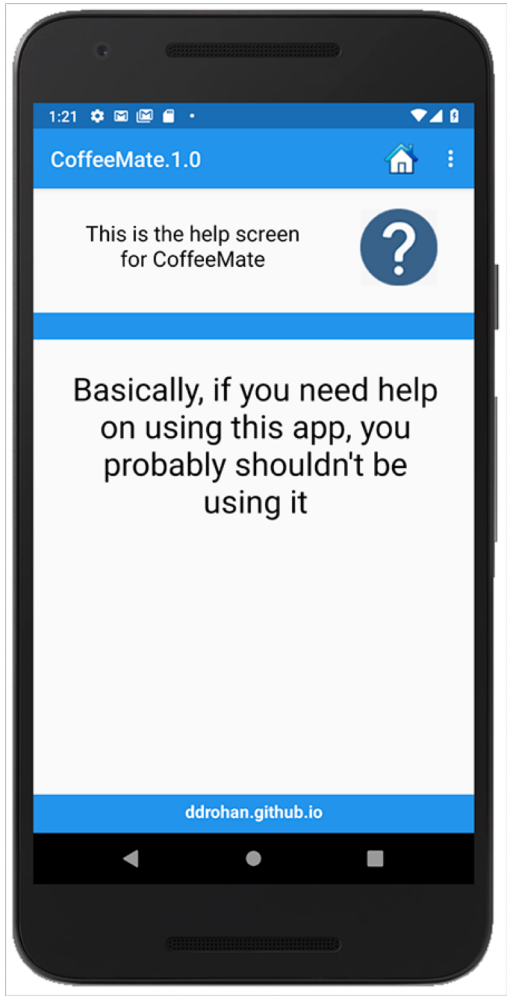
CoffeeMate 1.0

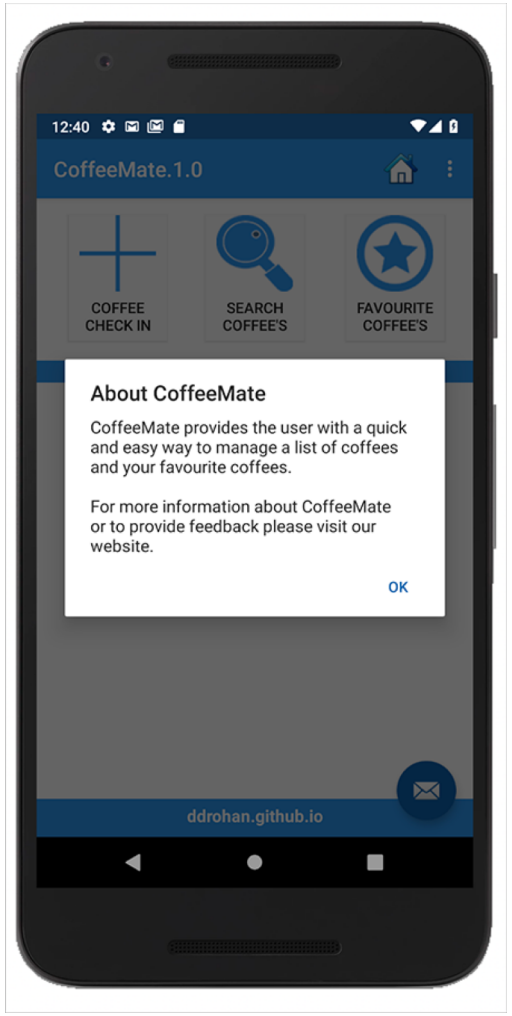
Using Buttons, Multiple Layouts & Menus

CoffeeMate 1.0

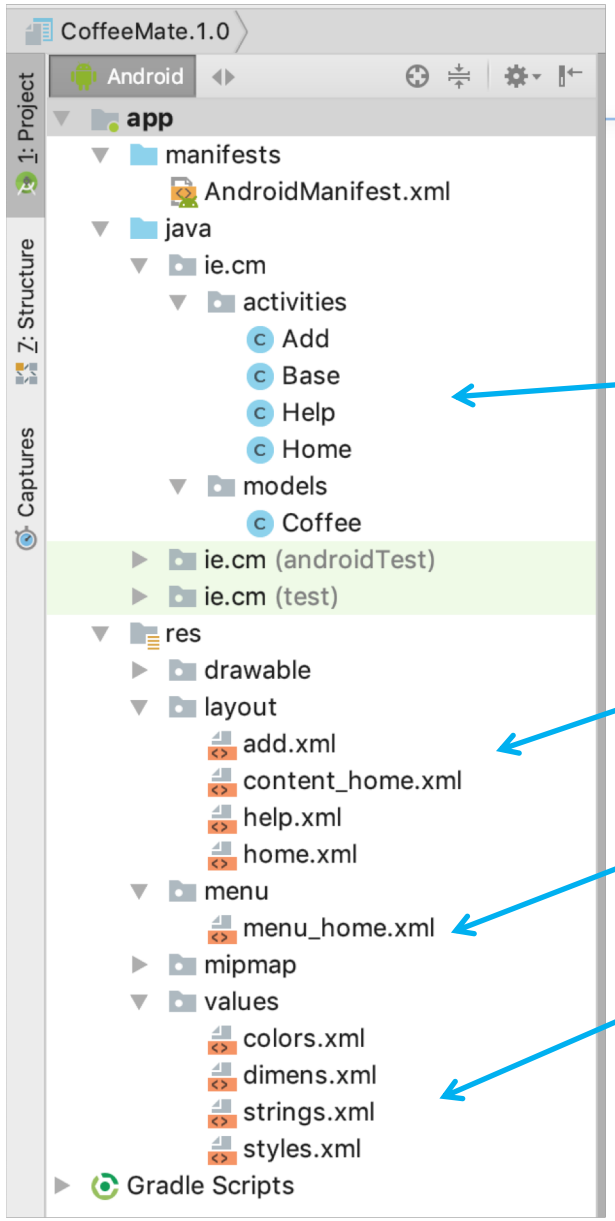






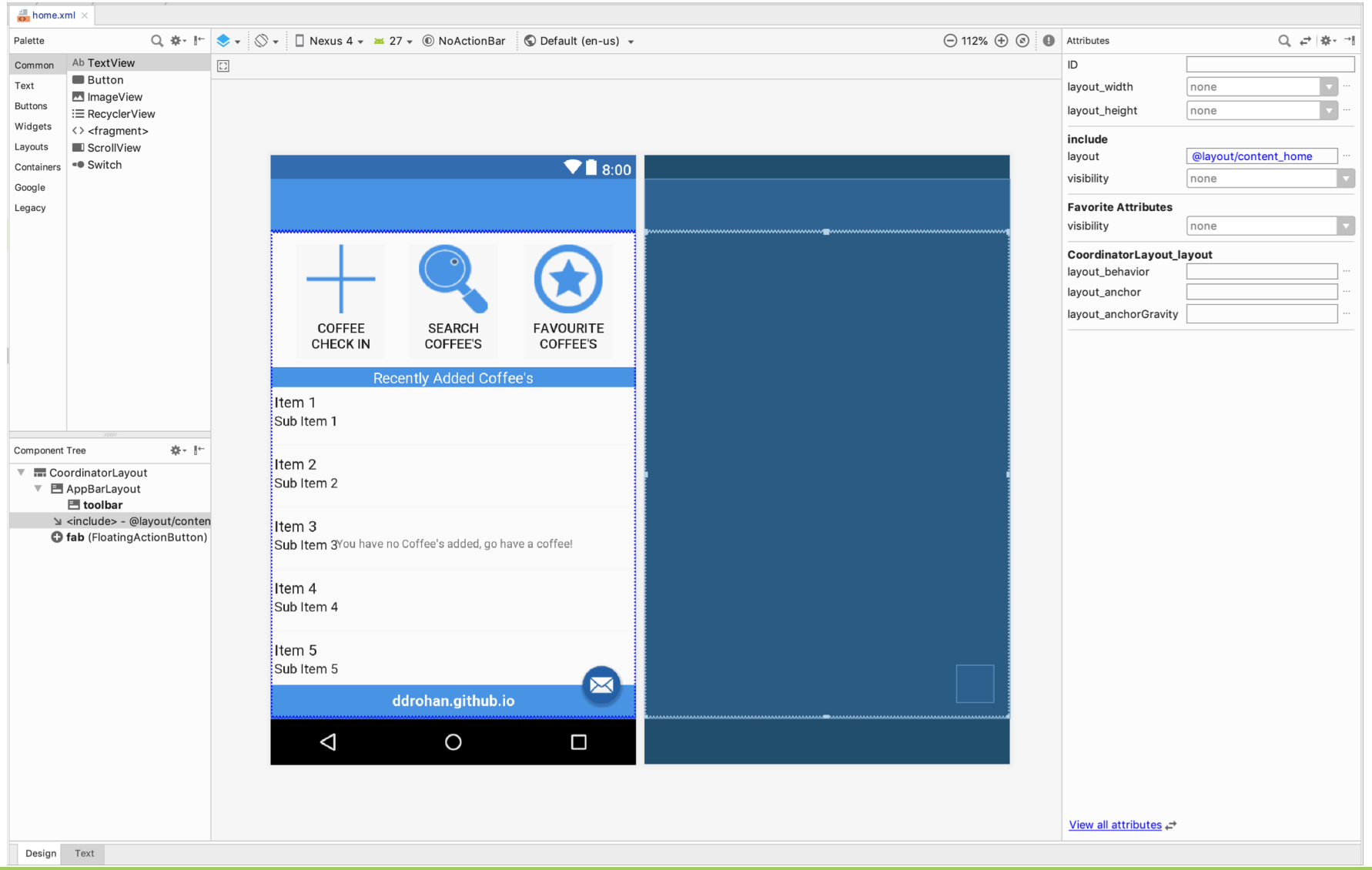


Project Structure – Version 1.0



- 5 java source files
- 4 xml layouts
- 1 xml file for a menu
- 4 separate xml files for color, string, style & dimension resources

Layout – *home*

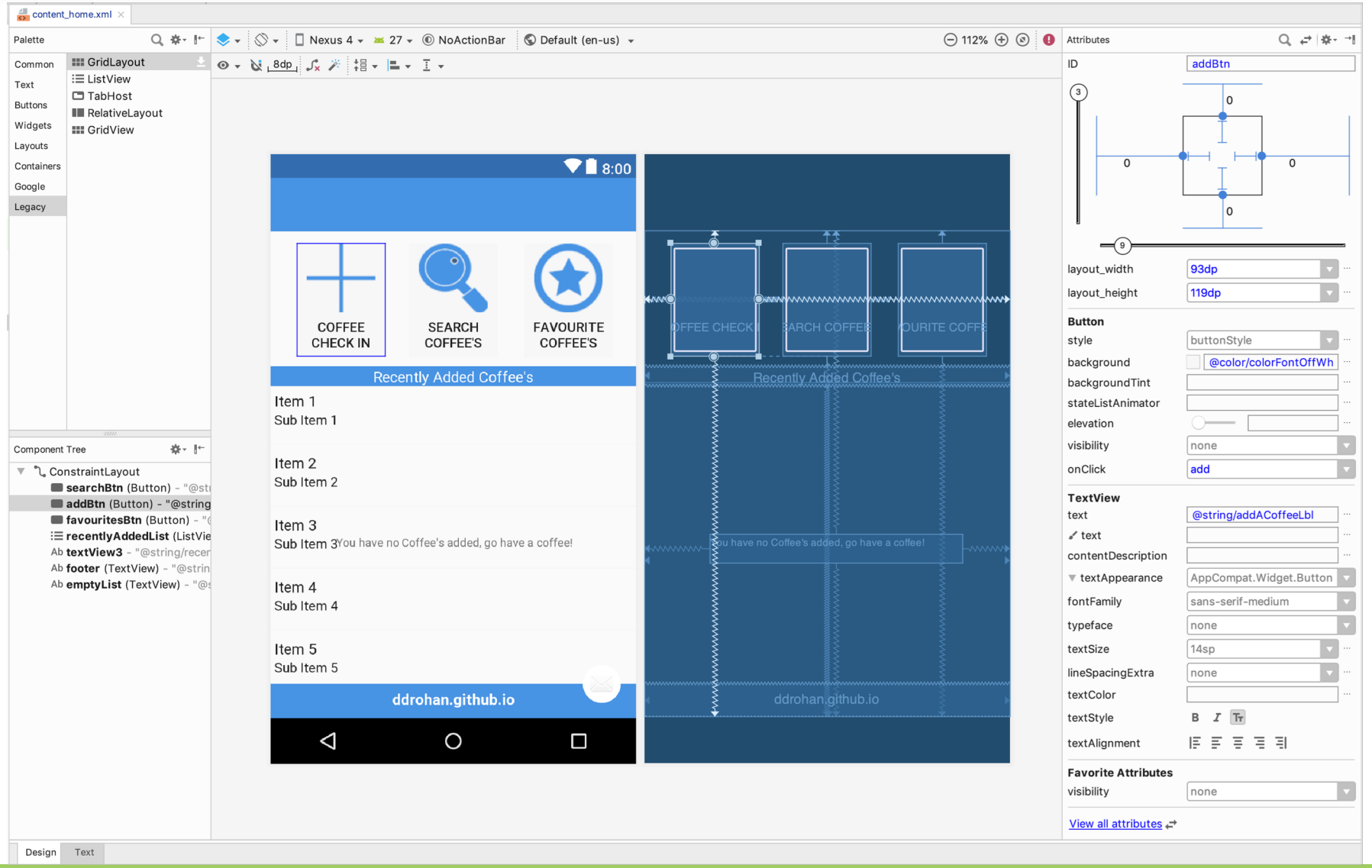

home.xml
Palette
Common
Text
Buttons
Widgets
Layouts
Containers
Google
Legacy
TextView
Button
ImageView
RecyclerView
<fragment>
ScrollView
Switch
Component Tree
CoordinatorLayout
AppBarLayout
toolbar
<include> - @layout/conten
fab (FloatingActionButton)
Nexus 4
27
NoActionBar
Default (en-us)
112%
8:00
COFFEE CHECK IN
SEARCH COFFEE'S
FAVOURITE COFFEE'S
Recently Added Coffee's
Item 1
Sub Item 1
Item 2
Sub Item 2
Item 3
Sub Item 3
You have no Coffee's added, go have a coffee!
Item 4
Sub Item 4
Item 5
Sub Item 5
ddrohan.github.io
Attributes
ID
layout_width
none
layout_height
none
include
layout
@layout/content_home
visibility
none
Favorite Attributes
visibility
none
CoordinatorLayout_layout
layout_behavior
layout_anchor
layout_anchorGravity
View all attributes
Design
Text

XML View – *home **

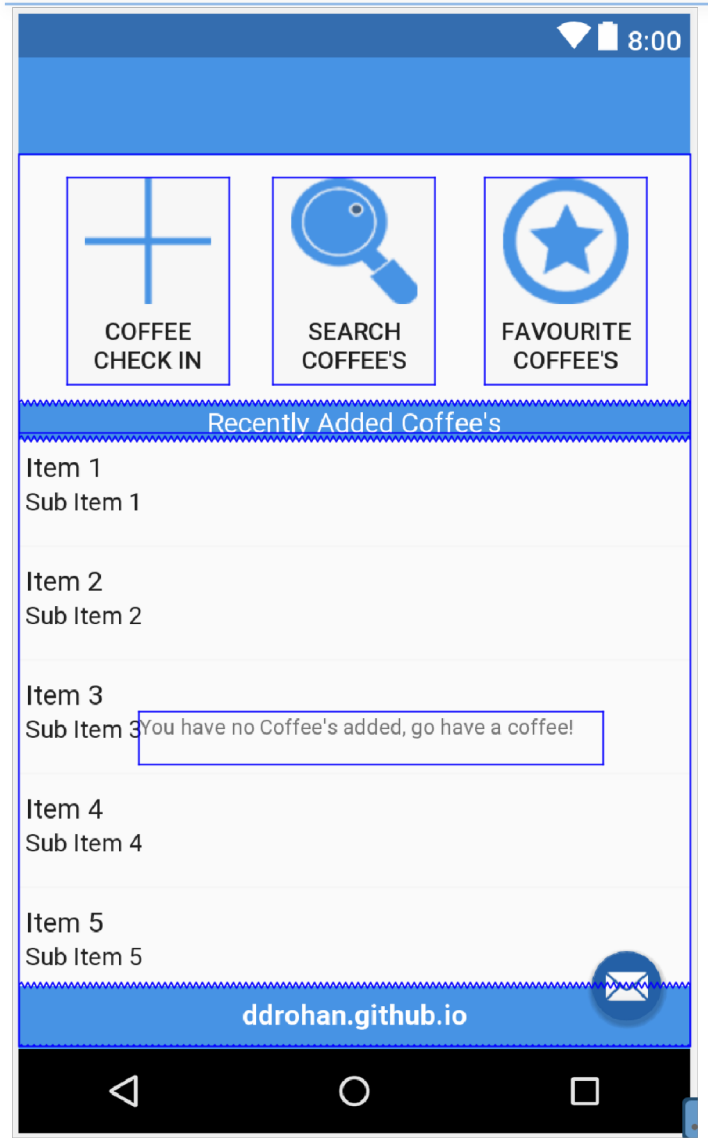
home.xml

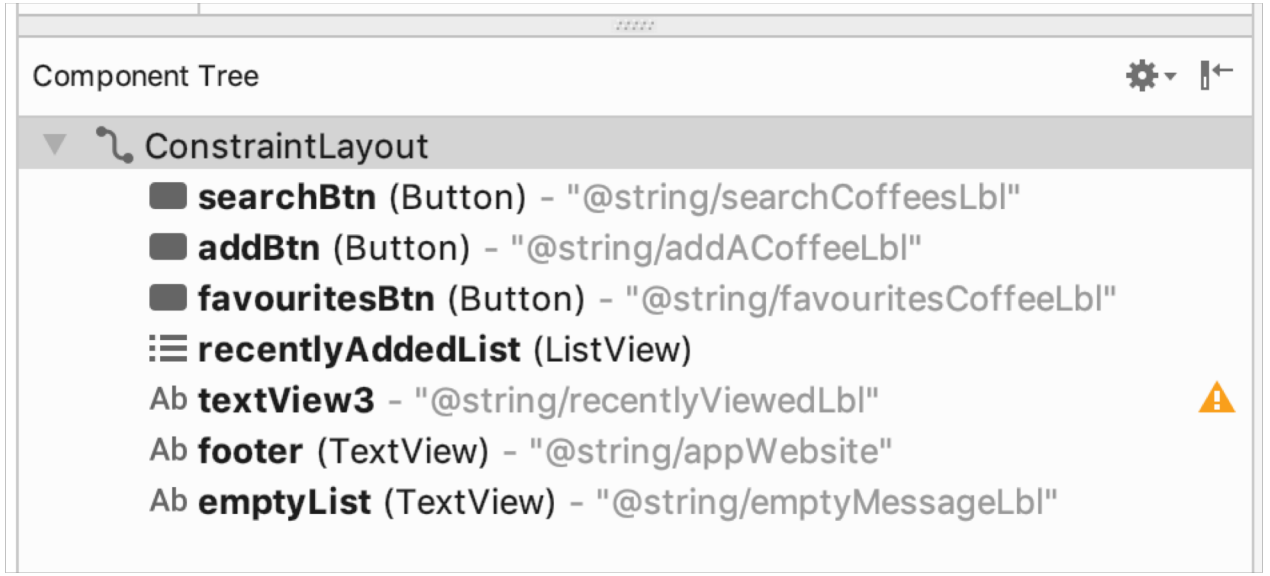
```
1  <?xml version="1.0" encoding="utf-8"?>
2  <android.support.design.widget.CoordinatorLayout
3      xmlns:android="http://schemas.android.com/apk/res/android" xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent" android:layout_height="match_parent" tools:context=".activities.Home">
6
7      <android.support.design.widget.AppBarLayout
8          android:layout_width="match_parent"
9          android:layout_height="wrap_content"
10         android:theme="@style/AppTheme.AppBarOverlay">
11
12         <android.support.v7.widget.Toolbar
13             android:id="@+id/toolbar"
14             android:layout_width="match_parent"
15             android:layout_height="?attr/actionBarSize"
16             android:background="?attr/colorPrimary"
17             app:popupTheme="@style/AppTheme.PopupOverlay" />
18
19     </android.support.design.widget.AppBarLayout>
20
21     <include layout="@layout/content_home" />
22
23     <android.support.design.widget.FloatingActionButton
24         android:id="@+id/fab"
25         android:layout_width="wrap_content"
26         android:layout_height="wrap_content"
27         android:layout_gravity="bottom|end"
28         android:layout_margin="16dp"
29         app:srcCompat="@android:drawable/ic_dialog_email" />
30
31 </android.support.design.widget.CoordinatorLayout>
```

Layout – *content_home*



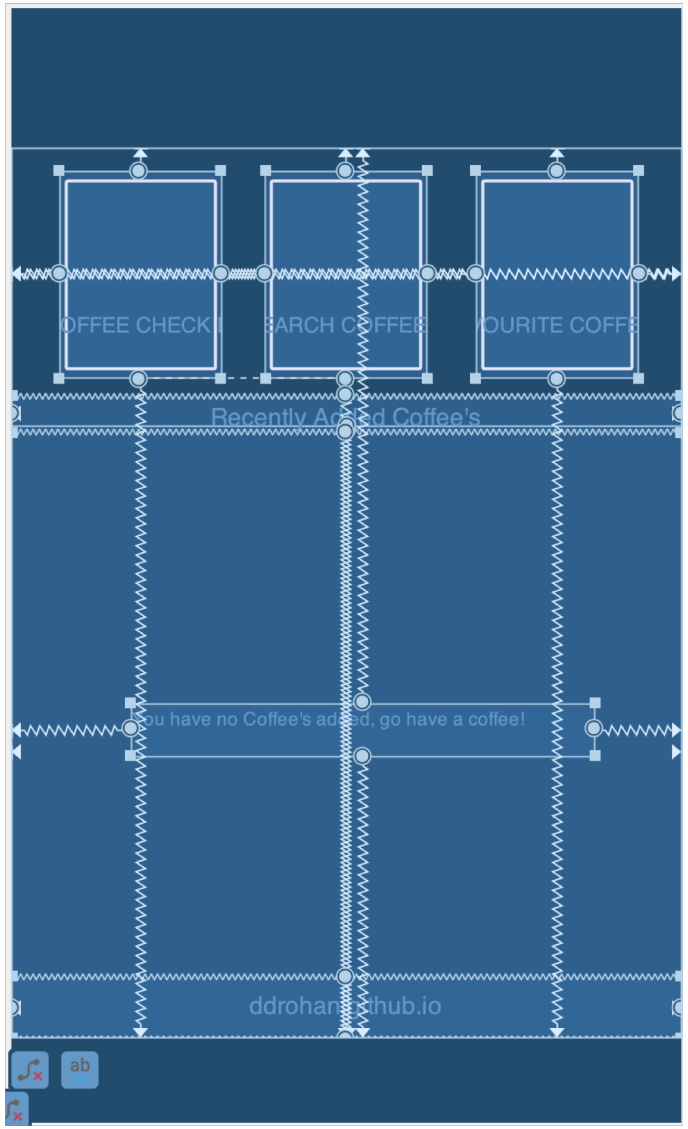
Layout – Outline View *

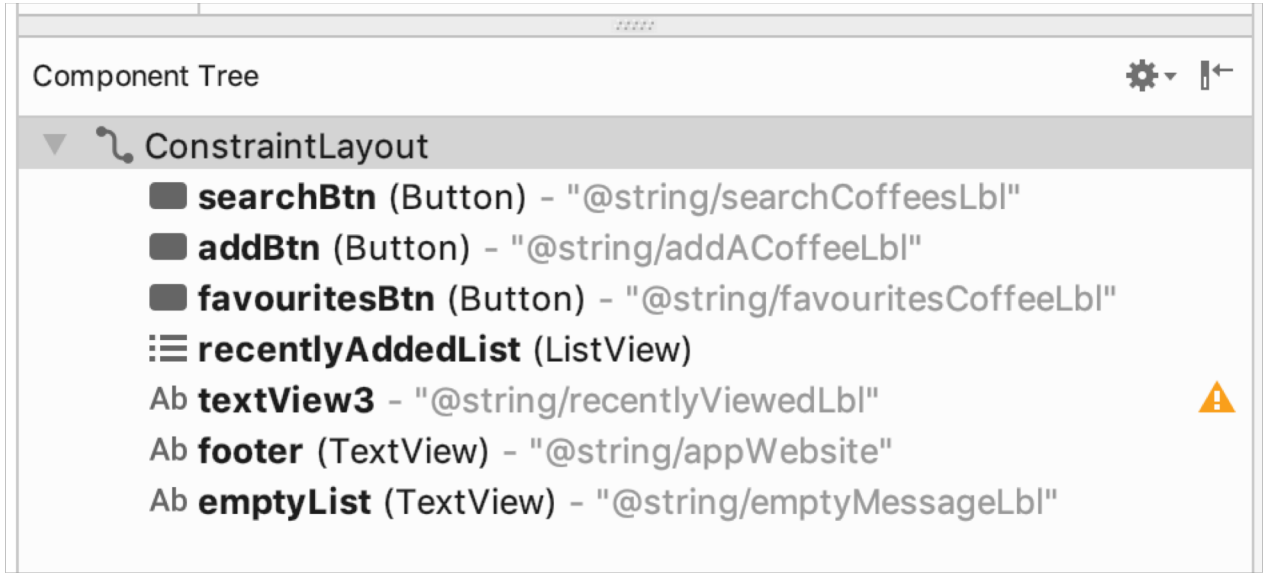




- ❑ Keep track of Outline view
- ❑ Name controls appropriately

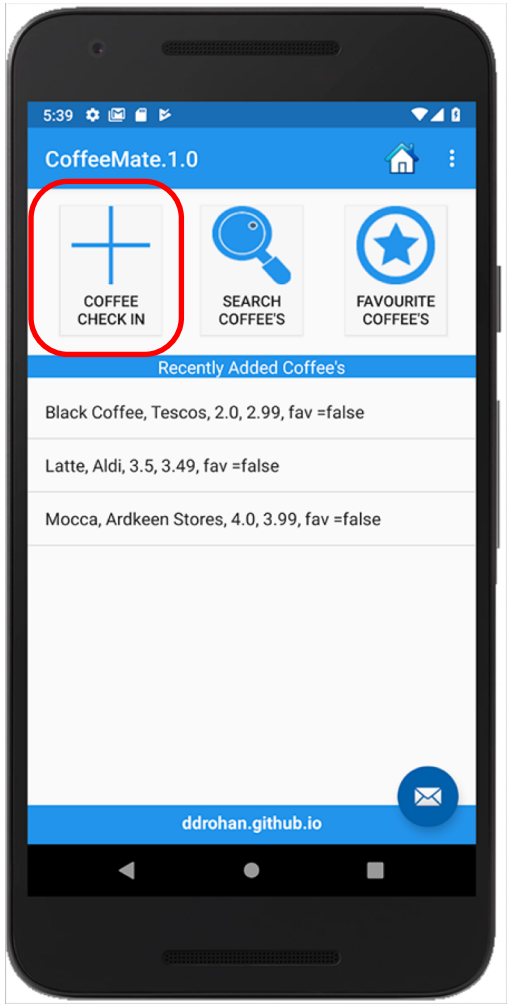
Layout – Outline View *





- Keep track of Outline view
- Name controls appropriately

XML View - content_home *



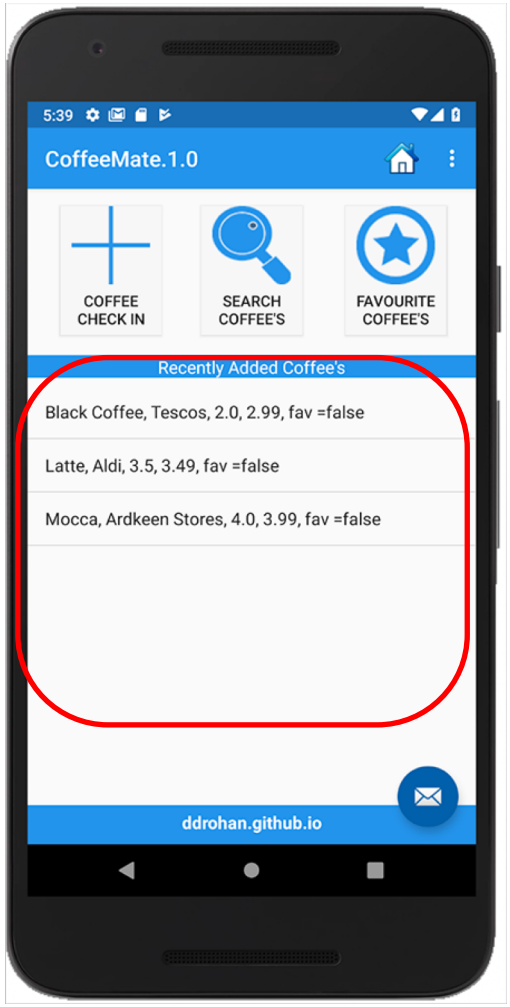
This part defines the one of the buttons shown on the layout summary slide. Each button is given an id so that it can be found in Java via 'findViewById', then assigned an event handler via setOnClickListener (or onClick)

The text (Button label) is taken from strings.xml instead of entered directly here, because the same label will also be used for other widgets later on.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/a
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="android.support.design.widget.
    tools:context=".activities.Home"
    tools:showIn="@layout/home">

    <Button...>

    <Button
        android:id="@+id/addBtn"
        android:layout_width="93dp"
        android:layout_height="119dp"
        android:background="@color/colorFontOffWhite"
        android:drawableTop="@drawable/add_72"
        android:onClick="add"
        android:text="Coffee Check In"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.094"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.033" />

    <Button...>

    <ListView...>

    <TextView...>

    <TextView...>

    <TextView...>

</android.support.constraint.ConstraintLayout>
```

Note the use of an 'onClick' attribute

XML View - content_home



The add and help screens are built and designed in a similar manner

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
*    xmlns:android="http://schemas.android.com/apk/res/android
     xmlns:app="http://schemas.android.com/apk/res-auto"
     xmlns:tools="http://schemas.android.com/tools"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     app:layout_behavior="android.support.design.widget.AppBar
     tools:context=".activities.Home"
     tools:showIn="@layout/home">

    <Button...>

    <Button...>

    <Button...>

    <ListView
        android:id="@+id/recentlyAddedList"
        android:layout_width="411dp"
        android:layout_height="375dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.48"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView3"
        app:layout_constraintVertical_bias="0.135" />

    <TextView...>

    <TextView...>

    <TextView...>

</android.support.constraint.ConstraintLayout>
```

CoffeeMate Event Handler *

```java
public class Home extends Base {

    TextView emptyList;
    ListView coffeeListView;
    ArrayAdapter<Coffee> coffeeAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    public void add(View v) {
        startActivity(new Intent(this, Add.class));
    }

    @Override
    protected void onResume() {
        super.onResume();
        Log.v("coffeemate","Home : " + coffeeList);

        if(!coffeeList.isEmpty())
            coffeeAdapter.notifyDataSetChanged();
    }
}
```

Note the use of a ‘View’ object

content_home

```
<Button
    android:id="@+id/addBtn"
    android:layout_width="93dp"
    android:layout_height="119dp"
    android:background="@color/colorFontOffWhite"
    android:drawableTop="@drawable/add_72"
    android:onClick="add"
    android:text="Coffee Check In"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.094"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.033" />
```

strings.xml *

content_home.xml (and the other layouts) refer to these names with @string/appName, @string/addACoffeeLbl etc.

Each string is used as a resource for one or more of the widgets on out layouts.

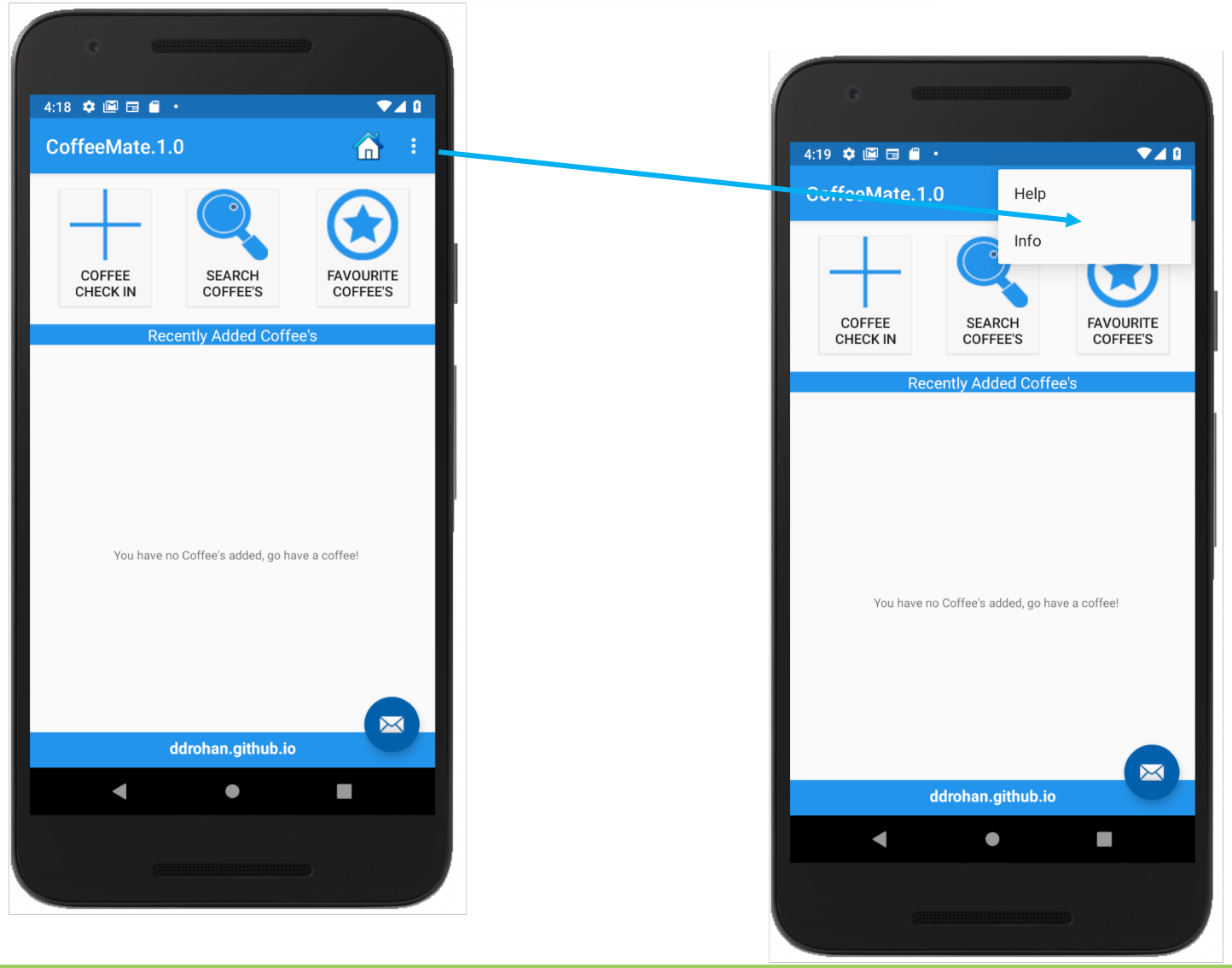
colors.xml & styles.xml are very similar in terms of content

```
<resources>
    <string name="app_name">CoffeeMate.1.0</string>
    <string name="appHelp">This is the help screen for CoffeeMate</str
    <string name="appHelpExtra">Basically\, if you need help on using t
    <string name="appDisplayName">CoffeeMate 1.0</string>
    <string name="appDesc">CoffeeMate provides the user with a quick ar
    <string name="appMoreInfo">For more information about CoffeeMate or
    <string name="appAbout">About CoffeeMate</string>
    <string name="appWebsite">ddrohan.github.io</string>
    <string name="developer">Developed by Davey Drohan</string>

    <string name="searchCoffeesLbl">Search Coffee\'s</string>
    <string name="favouritesCoffeeLbl">Favourite Coffee\'s</string>
    <string name="addACoffeeLbl">Coffee Check In</string>
    <string name="addCoffeeBtnLbl">Add Coffee</string>
    <string name="saveCoffeeBtn">Save Coffee</string>
    <string name="recentlyViewedLbl">Recently Added Coffee\'s</string>
    <string name="coffeeNameLbl">Name :</string>
    <string name="coffeeShopLbl">Shop :</string>
    <string name="coffeePriceLbl">Price :</string>
    <string name="coffeeRatingLbl">Star Rating</string>
    <string name="coffeeDetailsLbl">Full Coffee Details</string>
    <string name="informationLbl">Information</string>
    <string name="emptyMessageLbl">You have no Coffee\'s added, go have
</resources>
```

Menus in *CoffeeMate*

Pressing the “Menu” button on the emulator brings up a menu with the following entries

(we’ll modify this slightly in CoffeeMate 2.0)



Menus

- Menus are a common user interface component in many types of applications.
- To provide a familiar and consistent user experience, you should use the Menu APIs to present user actions and other options in your activities.
- Beginning with Android 3.0 (API level 11), Android-powered devices are no longer required to provide a dedicated *Menu* button.
  - instead provide an **action bar** to present common user actions.

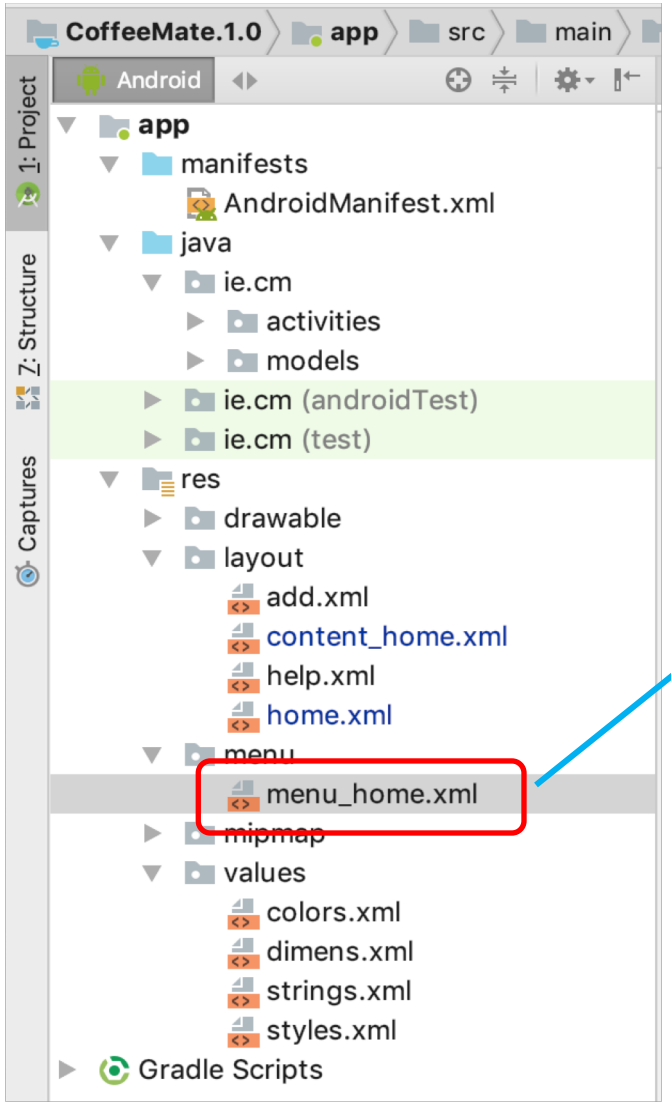
Options Menu & Action Bar

- The options menu is the primary collection of menu items for an activity.
  - It's where you should place actions that have a global impact on the app, such as “Info”, “Help” and “Home” etc.
- If you're developing for Android 2.3 or lower, users can reveal the options menu panel by pressing the *Menu* button.
- On Android 3.0 and higher, items from the options menu are presented by the action bar as a combination of on-screen action items and overflow options.

Enabling/Disabling Menu Items on the fly

- There may be times where you don't want all your menu options available to the user under certain situations
  - e.g – if you've no coffees, why let them see the report?
- You can modify the options menu at runtime by overriding the ***`onPrepareOptionsMenu`*** method
  - called each and every time the user presses the *MENU* button.

Menus in *CoffeeMate* *




```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="ie.cm.activities.Home">

    <item android:id="@+id/menu_help"
        android:title="Help"
        android:orderInCategory="100"
        android:onClick="menuHelp"
        app:showAsAction="never" />

    <item android:id="@+id/menu_info"
        android:title="Info"
        android:orderInCategory="100"
        android:onClick="menuInfo"
        app:showAsAction="never" />

    <item
        android:id="@+id/menu_home"
        android:icon="@drawable/home"
        android:onClick="menuHome"
        android:orderInCategory="100"
        android:title="Home"
        app:showAsAction="ifRoom" />
</menu>
```

Menu Specification

Note the use of an ‘onClick’ attribute

CoffeeMate Menu Event Handler *

```
public class Base extends AppCompatActivity {

    public static ArrayList<Coffee> coffeeList = new ArrayList<~>();

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_home, menu);
        return true;
    }

    public void menuHome(MenuItem m) {
        startActivity(new Intent(this, Home.class));
    }

    public void menuInfo(MenuItem m) {
        new AlertDialog.Builder(this)
                .setTitle(getString(R.string.appAbout))
                .setMessage(getString(R.string.appDesc)
                        + "\n\n"
                        + getString(R.string.appMoreInfo))
                .setPositiveButton("OK", (dialog, which) -> {
                    // we could put some code here too
                })
                .show();
    }

    public void menuHelp(MenuItem m) {
        startActivity(new Intent(this, Help.class));
    }
}
```

Menu Specification

inflate this resource as a 'Menu' (creates the menu)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="ie.cm.activities.Home">

    <item android:id="@+id/menu_help"
        android:title="Help"
        android:orderInCategory="100"
        android:onClick="menuHelp"
        app:showAsAction="never" />

    <item android:id="@+id/menu_info"
        android:title="Info"
        android:orderInCategory="100"
        android:onClick="menuInfo"
        app:showAsAction="never" />

    <item
        android:id="@+id/menu_home"
        android:icon="@drawable/home"
        android:onClick="menuHome"
        android:orderInCategory="100"
        android:title="Home"
        app:showAsAction="ifRoom" />
</menu>
```

Note the use of a 'MenuItem' object

Aside - Why a ‘Base’ Class? *

- *Green* Programming – Reduce, Reuse, Recycle
  - Reduce the amount of code we need to implement the functionality required (Code Redundancy)
  - Reuse common code throughout the app/project where possible/appropriate
  - Recycle existing code for use in other apps/projects

All good for improving Design

Switching to/from Activities - General Approach

- Switch between Activities with Intents when
  - Main screen has buttons and/or menus to navigate to other Activities (your intent)
  - Return to original screen with “back” button (system intent)
- Syntax required to start new Activity
  - Java

```
Intent goToActivity = new Intent(this,OtherActivity.class);
startActivity(goToActivity);
```

  - XML
    - Requires an entry in AndroidManifest.xml (runtime error otherwise!)

CoffeeMate 1.0

Code Highlights

class *Base* (our superclass) *

```
public class Base extends AppCompatActivity {

    public static ArrayList<Coffee> coffeeList = new ArrayList<~>();

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_home, menu);
        return true;
    }

    public void menuHome(MenuItem m) {
        startActivity(new Intent(this, Home.class));
    }

    public void menuInfo(MenuItem m) {
        new AlertDialog.Builder(this)
                .setTitle(getString(R.string.appAbout))
                .setMessage(getString(R.string.appDesc)
                        + "\n\n"
                        + getString(R.string.appMoreInfo))
                .setPositiveButton("OK", (dialog, which) -> {
                        // we could put some code here too
                })
                .show();
    }

    public void menuHelp(MenuItem m) {
        startActivity(new Intent(this, Help.class));
    }
}
```

our list of Coffees (available/shared between all our Activities)

A method to display a Dialog Window in the current Activity (specifically an AlertDialog)

class *Add* (1)

```
public class Add extends Base {

    private String      coffeeName, coffeeShop;
    private double      coffeePrice, ratingValue;
    private EditText name, shop, price;
    private RatingBar ratingBar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.add);

        name = findViewById(R.id.addNameET);
        shop =  findViewById(R.id.addShopET);
        price =  findViewById(R.id.addPriceET);
        ratingBar =  findViewById(R.id.addRatingBar);
    }
```

Binding to our Widgets

class *Add* (2)

```
public void addCoffee(View v) {

    coffeeName = name.getText().toString();
    coffeeShop = shop.getText().toString();
    try {
        coffeePrice = Double.parseDouble(price.getText().toString());
    } catch (NumberFormatException e) {
        coffeePrice = 0.0;
    }
    ratingValue = ratingBar.getRating();

    if ((coffeeName.length() > 0) && (coffeeShop.length() > 0)
            && (price.length() > 0)) {
        Coffee c = new Coffee(coffeeName, coffeeShop, ratingValue,
                coffeePrice, false);

        Log.v("coffeemate","Add : " + coffeeList);
        coffeeList.add(c);
        startActivity(new Intent(this, Home.class));
    } else
        Toast.makeText(
                this,
                "You must Enter Something for "
                        + "\'Name\', \'Shop\' and \'Price\'",
                Toast.LENGTH_SHORT).show();
}
```

Our 'Event Handler' Method

Adding the Coffee to our List
&
Returning to our 'Home' Activity

class *Home*

```
public class Home extends Base {

    TextView emptyList;
    ListView coffeeListView;
    ArrayAdapter<Coffee> coffeeAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.home);
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        emptyList = findViewById(R.id.emptyList);
        coffeeListView = findViewById(R.id.recentlyAddedList);
        coffeeListView.setEmptyView(emptyList);
        coffeeAdapter = new ArrayAdapter<>(this,
                                    android.R.layout.simple_list_item_1,
                                    coffeeList);
        coffeeListView.setAdapter(coffeeAdapter);
       // ...
    }

    public void add(View v) { startActivity(new Intent(this, Add.class)); }

    @Override
    protected void onResume() {
        super.onResume();

        if(!coffeeList.isEmpty())
            coffeeAdapter.notifyDataSetChanged();
    }
}
```

Declaring our Variables for displaying the coffees

Creating the ArrayAdapter and 'setting' it to the ListView

Refreshing the Adapter to check for changes (new coffees)

Questions?

Appendix

Android Components

Content Providers (1)

- A component that stores and retrieves data and make it accessible to all applications.
  - uses a standard interface (URI) to fulfill requests for data from other applications & it's one way to share data across applications.
    - e.g. `android.provider.Contacts.Phones.CONTENT_URI`
  - Android ships with a number of content providers for common data types (audio, video, images, personal contact information, and so on) - SQLite DB
  - Android 4.0 introduces the Calendar Provider.
    - uri – `Calendars.CONTENT_URI;`

Content Providers (2)

- Content providers abstract data storage to other applications, activities, services, etc…
- Roughly SQL based.
- You construct a ContentProvider class that will override methods such as insert(), delete(), and update().
- Then you register your content provider with a URI to handle different types of objects.
  - A Unique Resource Identifier is kind of like a URL
- For example, let's say we want our content provider to allow other applications to access our database of bicycles and also customers.
- We define methods for inserting, deleting, updating, etc… bicycles and customers.
- Then we publish two URIs:
  - BICYCLES_URI
  - CUSTOMERS_URI
- Maybe more URIs for accessing bicycles indexed by serial number?

Broadcast Receivers

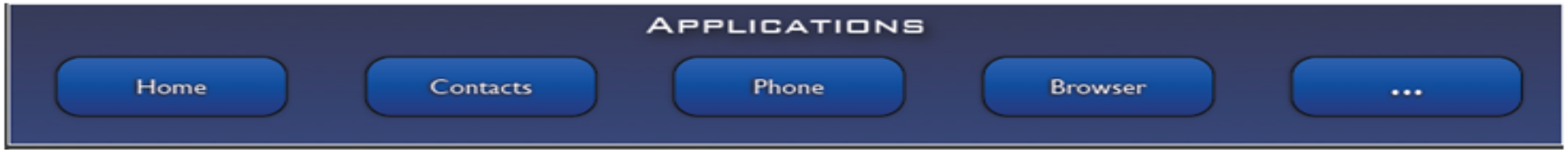
- A component designed to respond to broadcast Intents.
  - Receives system wide messages and implicit intents
  - can be used to react to changed conditions in the system (external notifications or alarms).
  - An application can register as a broadcast receiver for certain events and can be started if such an event occurs. These events can come from Android itself (e.g., battery low) or from any program running on the system.
- An Activity or Service provides other applications with access to its functionality by executing an Intent Receiver, a small piece of code that responds to requests for data or services from other activities.

The Layered Framework

slides paraphrase a blog post by Tim Bray (co-inventor of XML and currently employed by Google to work on Android)
http://www.tbray.org/ongoing/When/201x/2010/11/14/What-Android-Is

The Layered Framework (1)

❑ Applications Layer



- Android provides a set of core applications:
  - ✓ Email Client
  - ✓ SMS Program
  - ✓ Calendar
  - ✓ Maps
  - ✓ Browser
  - ✓ Contacts
  - ✓ YOUR APP
  - ✓ Etc
- All applications are written using the Java language. These applications are executed by the Dalvik Virtual Machine (DVM), similar to a Java Virtual Machine but with different bytecodes

The Layered Framework (2)

Application Framework Layer

- Enabling and simplifying the reuse of components
  - Developers have full access to the same framework APIs used by the core applications.
  - Users are allowed to replace components.
- These services are used by developers to create Android applications that can be run in the emulator or on a device

- See next slide for more…..

The Layered Framework (3)

Application Framework Layer Features

Feature	Role
View System	Used to build an application, including lists, grids, text boxes, buttons, and embedded web browser
Content Provider	Enabling applications to access data from other applications or to share their own data
Resource Manager	Providing access to non-code resources (localized strings, graphics, and layout files)
Notification Manager	Enabling all applications to display custom alerts in the status bar
Activity Manager	Managing the lifecycle of applications and providing a common navigation (back) stack

We'll be covering the above in more detail later on...

The Layered Framework (4)

Libraries Layer

- Including a set of C/C++ libraries used by components of the Android system
- Exposed to developers through the Android application framework

System C library/libc - a BSD (Berkeley Software Distribution) -derived implementation of the standard C system library (libc), tuned for embedded Linux-based devices

Media Framework/Libraries - based on PacketVideo's OpenCORE; the libraries support playback and recording of many popular audio and video formats, as well as static image files, including MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG

Surface Manager - manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications

WebKit/LibWebCore - a modern web browser engine which powers both the Android browser and an embeddable web view

SGL (Scene Graph Library) - the underlying 2D graphics engine

3D libraries - an implementation based on **OpenGL ES** 1.0 APIs; the libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software rasterizer (shapes->pixels)

FreeType - bitmap and vector font rendering

SQLite - a powerful and lightweight relational database engine available to all applications

The Layered Framework (5)

- Core Runtime Libraries
  (changing to ART in Kit Kat)

Next Slide

- Providing most of the functionality available in the core libraries of the Java language
- APIs
  - Data Structures
  - Utilities
  - File Access
  - Network Access
  - Graphics
  - Etc

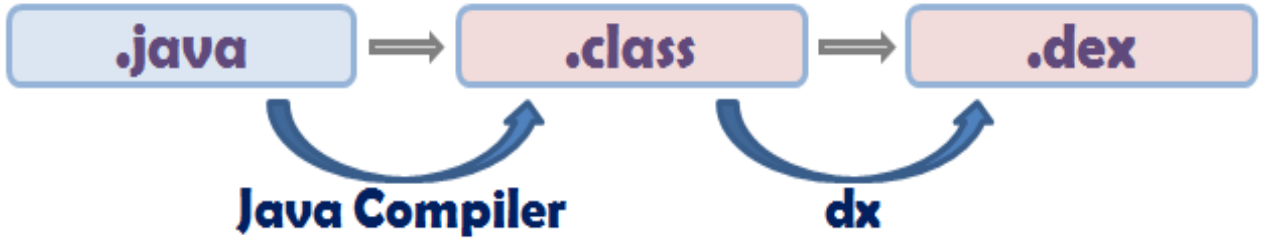
The Layered Framework (6)

- Dalvik Virtual Machine (DVM)
  - Provides an environment on which every Android application runs
    - Each Android application runs in its own process, with its own instance of the Dalvik VM.
    - Dalvik has been written such that a device can run multiple VMs efficiently.
- Android Runtime (ART) 4.4
  (see earlier slides)

The Layered Framework (7)

❑ Dalvik Virtual Machine (Cont'd)

- ✓ Executing the Dalvik Executable (.dex) format
  - ➢ .dex format is optimized for minimal memory footprint.
  - ➢ Compilation



- ✓ Relying on the Linux Kernel for:
  - ➢ Threading
  - ➢ Low-level memory management

ART – Android Runtime

- Handles app execution in a fundamentally different way from Dalvik.
- Current runtime relies on a JIT compiler to interpret original bytecode
  - In a manner of speaking, apps are only partially compiled by developers
  - resulting code must go through an interpreter on a user's device each and every time it is run == Overhead + Inefficient
  - But the mechanism makes it easy for apps to run on a variety of hardware and architectures.
- ART pre-compiles that bytecode into machine language when *apps are first installed*, turning them into truly native apps.
  - This process is called Ahead-Of-Time (AOT) compilation.
- By removing the need to spin up a new VM or run interpreted code, startup times can be cut down immensely and ongoing execution will become faster.

The Layered Framework (8)

Linux Kernel Layer

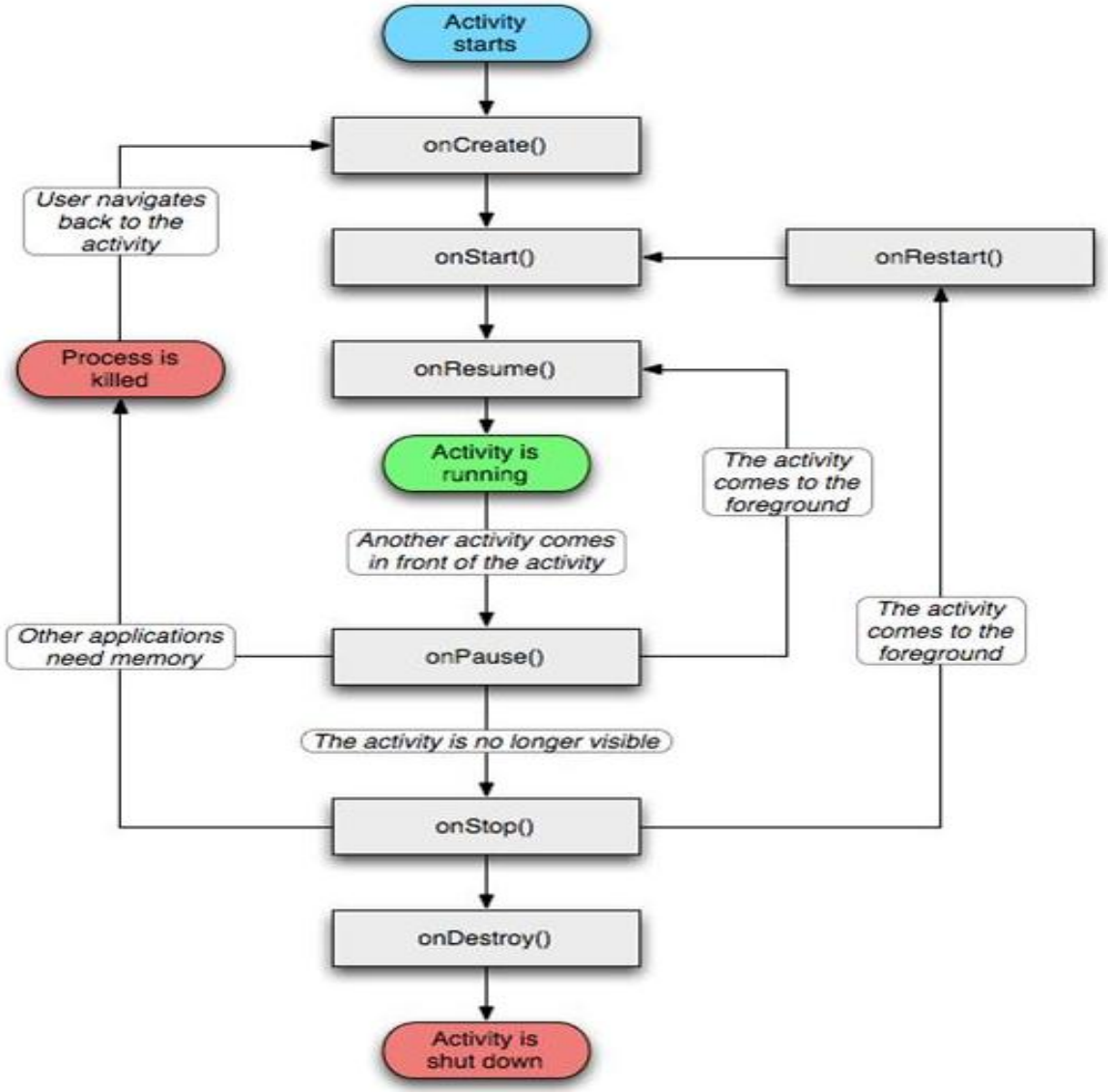
- At the bottom is the Linux kernel that has been augmented with extensions for Android
  - the extensions deal with power-savings, essentially adapting the Linux kernel to run on mobile devices
- Relying on Linux Kernel 2.6 for core system services / 3.8 in Kit Kat
  - Memory and Process Management
  - Network Stack
  - Driver Model
  - Security
- Providing an abstraction layer between the H/W and the rest of the S/W stack

The Application/Activity Lifecycle

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 Activity Life Cycle



- `onStop()` and `onDestroy()` are optional and may never be called
- If you need persistence, the save needs to happen in `onPause()`

The Activity Life Cycle

- An activity monitors and reacts to these events by instantiating methods that override the Activity class methods for each event:

onCreate

- Called when an activity is first created. This is the place you normally create your views, open any persistent data files your activity needs to use, and in general initialize your activity.
- When calling onCreate(), the Android framework is passed a Bundle object that contains any activity state saved from when the activity ran before.

onStart

- Called just before an activity becomes visible on the screen. Once onStart() completes, if your activity can become the foreground activity on the screen, control will transfer to onResume().
- If the activity cannot become the foreground activity for some reason, control transfers to the onStop() method.

The Activity Life Cycle

onResume

- Called right after onStart() if your activity is the foreground activity on the screen. At this point your activity is running and interacting with the user. You are receiving keyboard and touch inputs, and the screen is displaying your user interface.
- onResume() is also called if your activity loses the foreground to another activity, and that activity eventually exits, popping your activity back to the foreground. This is where your activity would start (or resume) doing things that are needed to update the user interface.

The Activity Life Cycle

onPause

- Called when Android is just about to resume a different activity, giving that activity the foreground. At this point your activity will no longer have access to the screen, so you should stop doing things that consume battery and CPU cycles unnecessarily.
  - If you are running an animation, no one is going to be able to see it, so you might as well suspend it until you get the screen back. Your activity needs to take advantage of this method to store any state that you will need in case your activity gains the foreground again—and it is not guaranteed that your activity will resume.
- Once you exit this method, Android may kill your activity at any time without returning control to you.

The Activity Life Cycle

- onStop
  - Called when your activity is no longer visible, either because another activity has taken the foreground or because your activity is being destroyed.
- onDestroy
  - The last chance for your activity to do any processing before it is destroyed. Normally you'd get to this point because the activity is done and the framework called its finish method. But as mentioned earlier, the method might be called because Android has decided it needs the resources your activity is consuming.

Questions?