

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





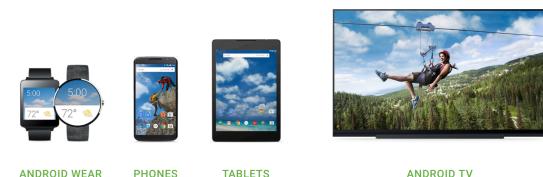
Agenda

- ❑ 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...



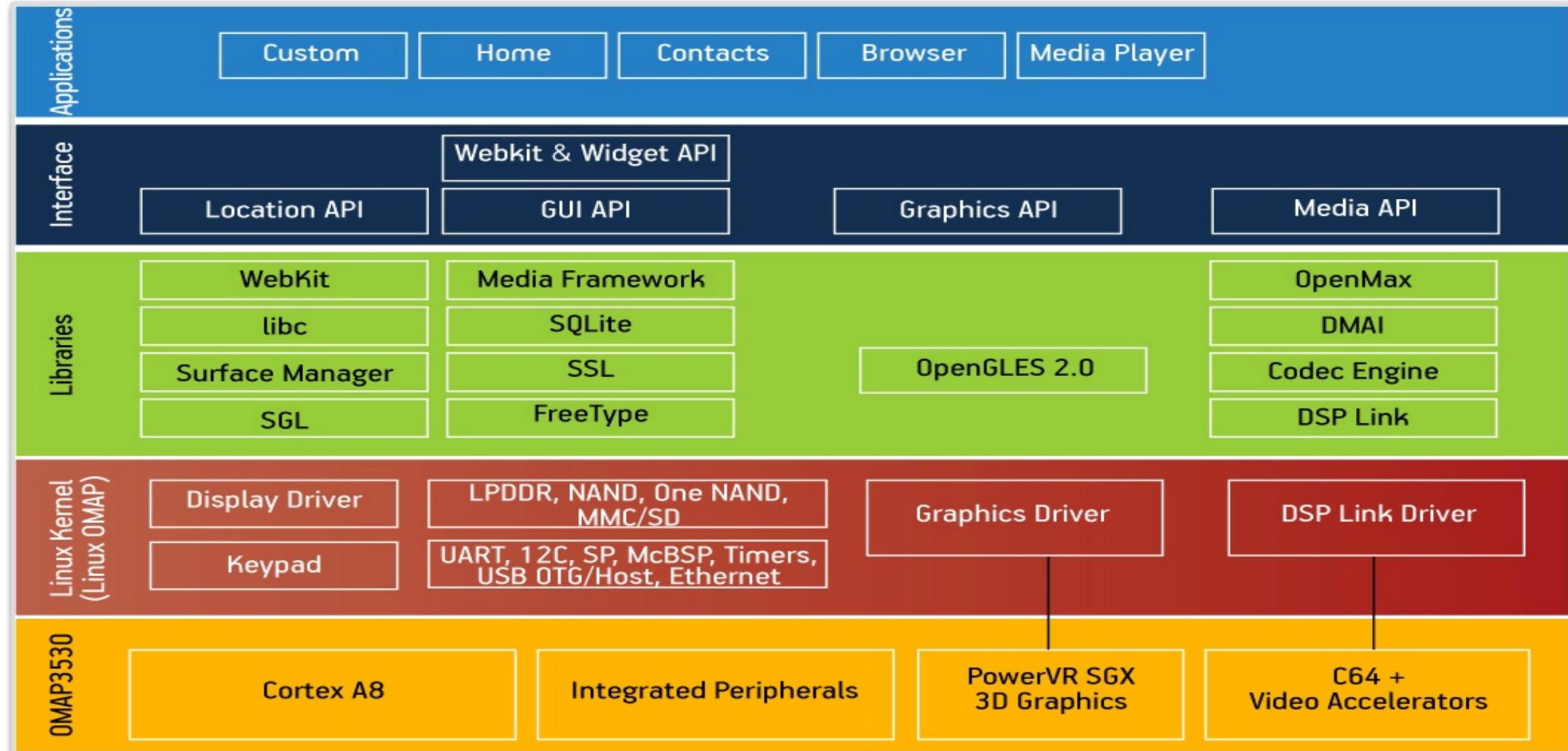
What is Android? (Recap)

- ❑ 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 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)





Key Parts of the Framework

- ❑ The key parts of the Android Framework are
 - The **Activity Manager**: starts, stops, pauses and resumes applications
 - The **Resource Manager**: allows apps to access the resources bundled with them
 - **Content Providers**: objects that encapsulate data that is shared between applications
 - **Location Manager** and **Notification Manager** (events)
 - **Media Manager** for Audio and Video playback



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.



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**

See the Appendix for a more detailed explanation of these components

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





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’

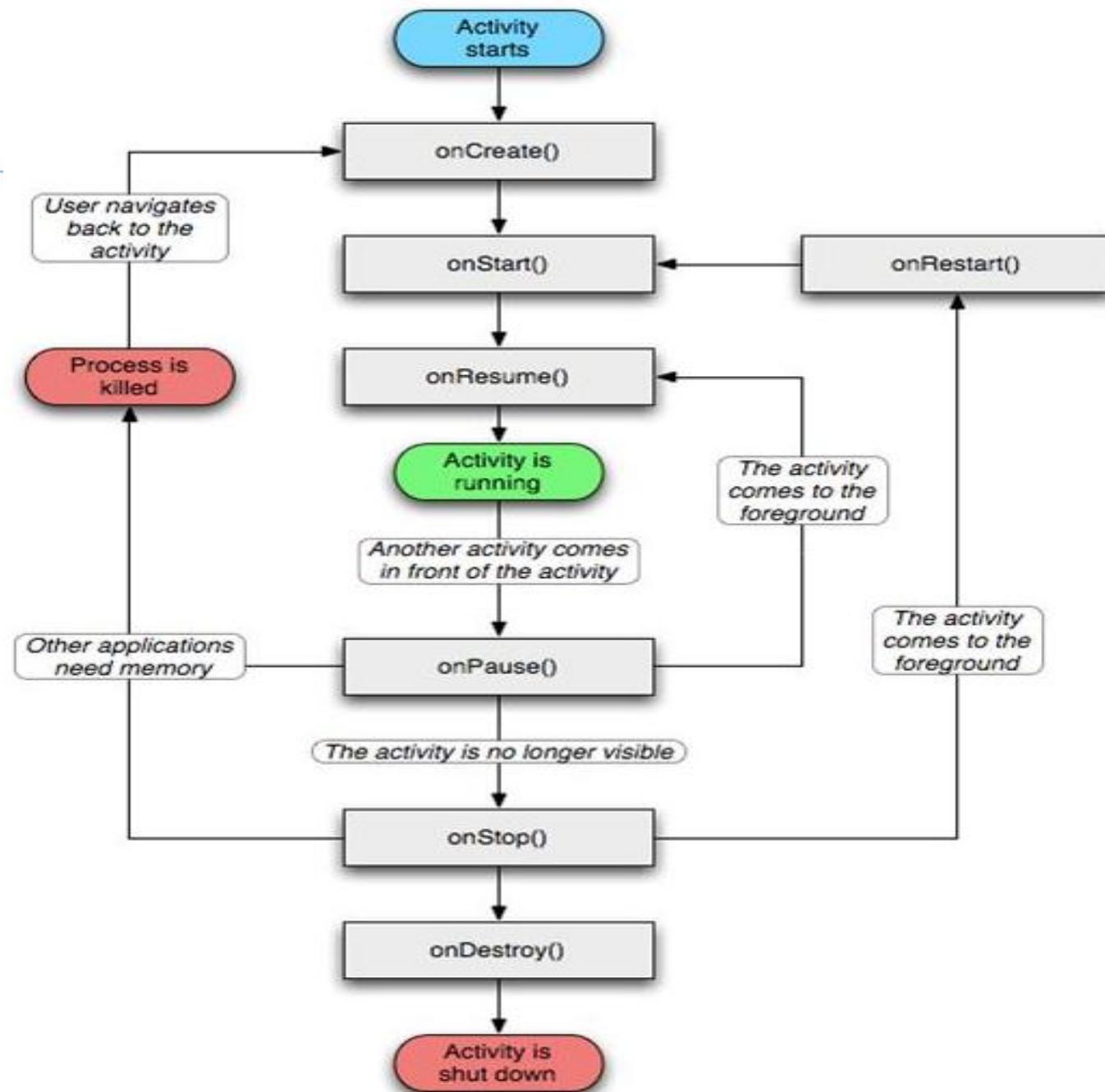


The **(Application) Activity** Life Cycle

- ❑ 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

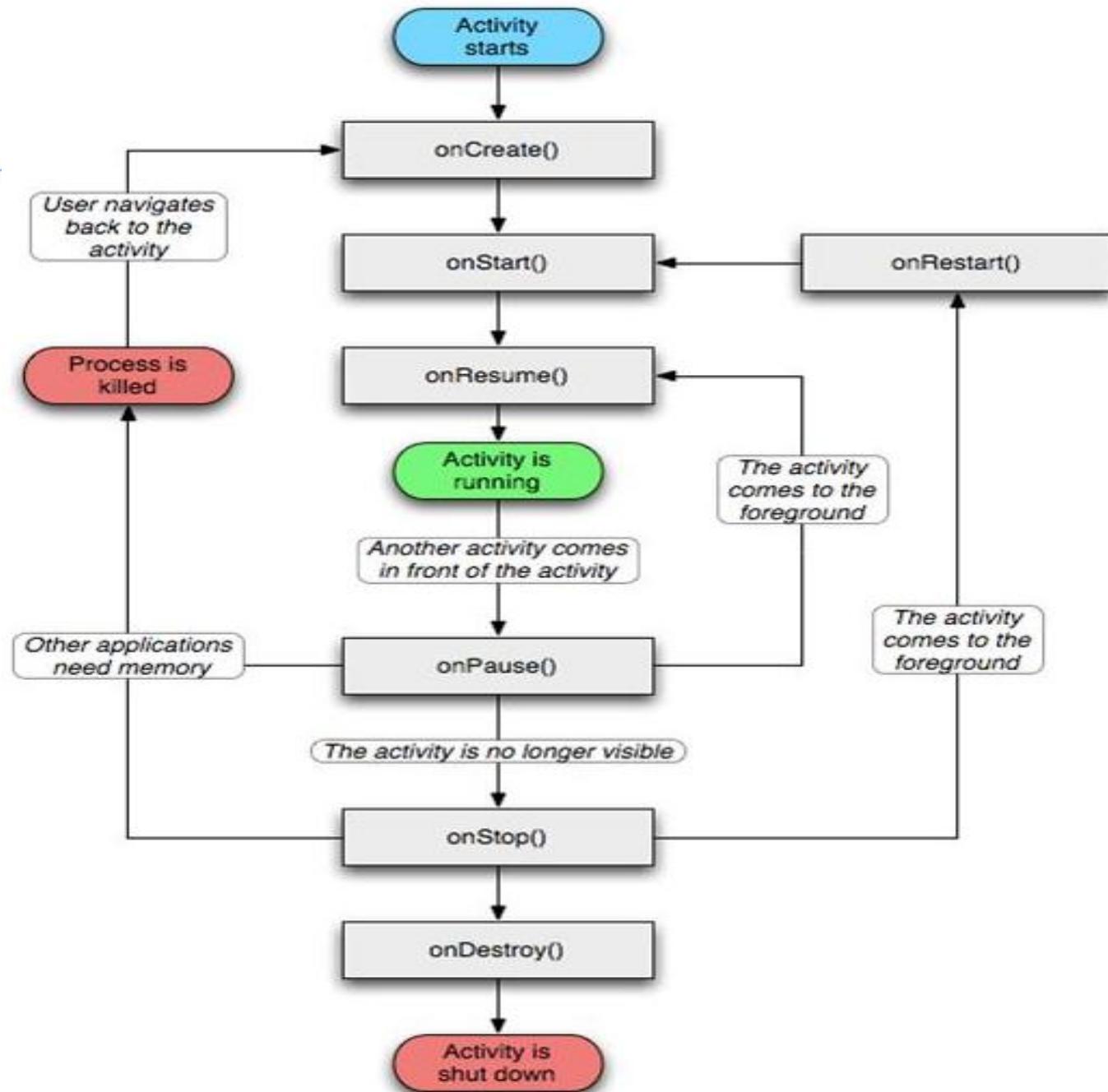
The **Activity** Life Cycle *

- 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.



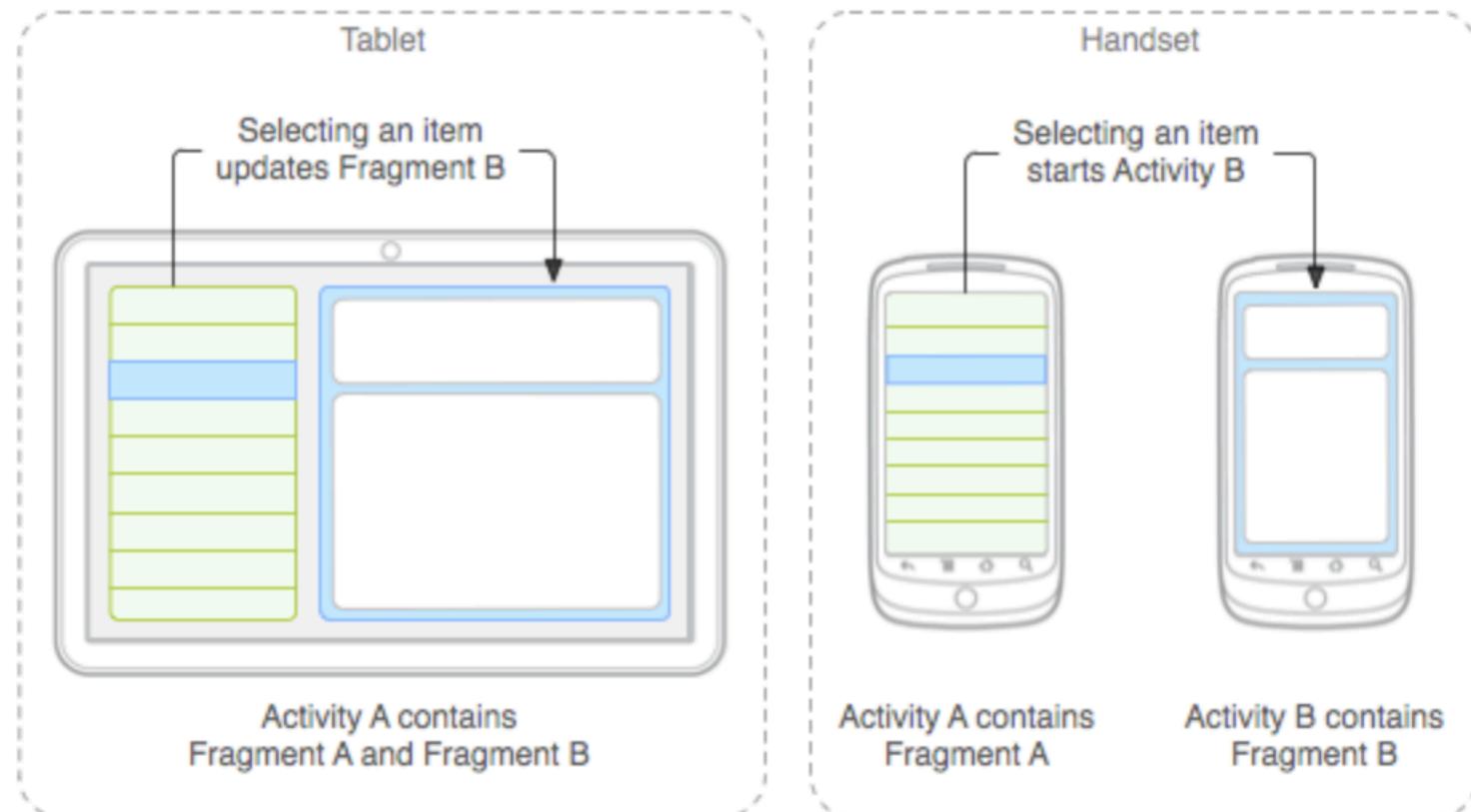
Designing Fragments *

- ❑ 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

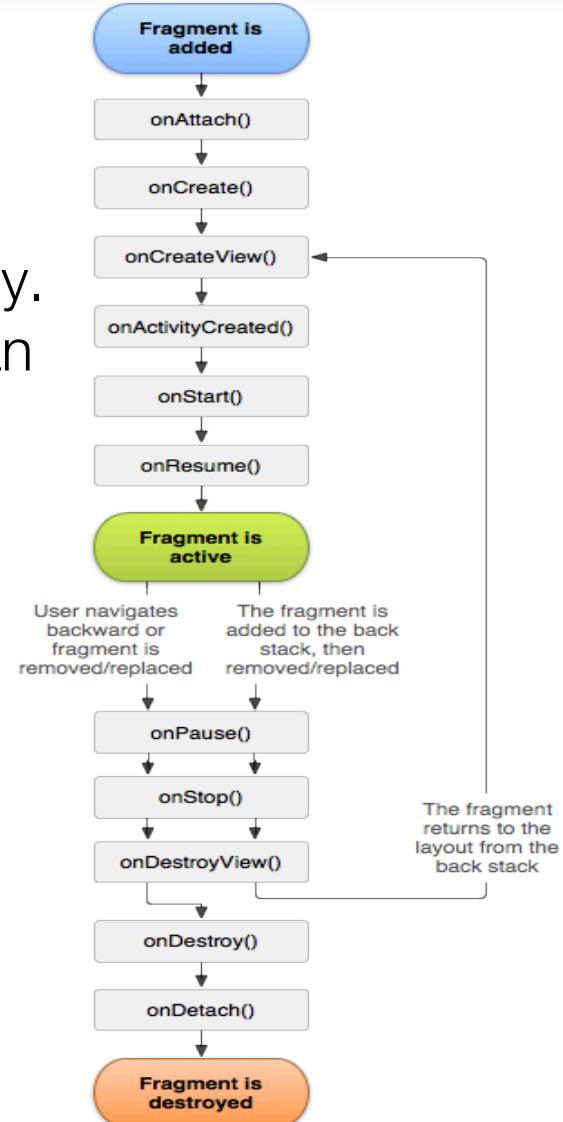


An example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.



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

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

- Project Structure:** Shows the project tree with packages like `ie.wit.lifecycle`, `MainActivity`, and test packages.
- Code Editor:** Displays the `MainActivity.java` file containing lifecycle methods. A yellow lightbulb icon is visible over line 50.
- Android Monitor:** Shows logcat output for the `Emulator Nexus_5_API_23`. The last three entries are highlighted with a red box and connected by a green arrow to the code in the editor:

```
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...
```
- Emulator:** Shows the app running on an Android 5.1 (API 23) emulator with the title "LifeCycle" and the text "Hello World!".

LifeCycle Example (2) *

User Selects 'Home'

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

- Project Structure:** Shows the project tree with packages like `ie.wit.lifecycle`, `MainActivity`, and test packages.
- Code Editor:** Displays the `MainActivity.java` file containing the `onPause()` and `onStop()` lifecycle methods.
- Android Monitor:** Shows logcat output for the emulator, highlighting onPause() and onStop() calls.
- Emulator:** Shows the Android home screen with icons for Google, Play Games, and others.

A red box highlights the logcat output in the Android Monitor, which shows the following messages:

```
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...
```

Green arrows point from the text "User Selects 'Home'" to the Android home screen icon on the emulator and from the onPause() and onStop() log entries to the home screen icon.

```
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
```

```
    Log.v("Lifecycle", "onStart() Called...");
}

@Override
protected void onPause()
{
    super.onPause();
    Log.v("Lifecycle", "onPause() Called...");
}

@Override
protected void onStop()
{
    super.onStop();
    Log.v("Lifecycle", "onStop() Called...");
}

@Override
protected void onDestroy()
{
    super.onDestroy();
    Log.v("Lifecycle", "onDestroy() Called...");
}

@Override
protected void onRestart()
{
    super.onRestart();
    Log.v("Lifecycle", "onRestart() Called...");
}
```

```
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
```

```
    Log.v("Lifecycle", "onStart() Called...");
}

@Override
protected void onPause()
{
    super.onPause();
    Log.v("Lifecycle", "onPause() Called...");
}

@Override
protected void onStop()
{
    super.onStop();
    Log.v("Lifecycle", "onStop() Called...");
}

@Override
protected void onDestroy()
{
    super.onDestroy();
    Log.v("Lifecycle", "onDestroy() Called...");
}

@Override
protected void onRestart()
{
    super.onRestart();
    Log.v("Lifecycle", "onRestart() Called...");
}
```

LifeCycle Example (3) *

User restarts App

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

- Code Editor:** Displays the `MainActivity.java` file. The code contains several lifecycle methods with logging statements. The logcat output in the Android Monitor shows three entries corresponding to these methods.
- Emulator:** Shows a Nexus 5 API 23 emulator displaying the app's UI with the title "Lifecycle" and the message "Hello World!".
- 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...
```

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 onStop()
50     super.onStop();
51     Log.v("Lifecycle", "onStop() Called...");
52
53 }
```

LifeCycle Example (4) *

User Selects 'Back'

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

- Code Editor:** Displays the `MainActivity.java` file. The code implements the `LifecycleObserver` interface and overrides several lifecycle methods (`onStart()`, `onStop()`, `onDestroy()`, `onCreate()`, `onResume()`, `onPause()`, and `onRestart()`). Log statements are included in each method to log the event.
- Android Monitor:** Shows logcat output for the `Emulator Nexus_5_API_23`. The log entries are:

```
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 these three entries.
- Emulator:** An Android emulator running on a Nexus 5 device with API 23. The screen shows the home screen with various app icons. A green arrow points from the 'User Selects 'Back'' text to the back button icon on the emulator's navigation bar.



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

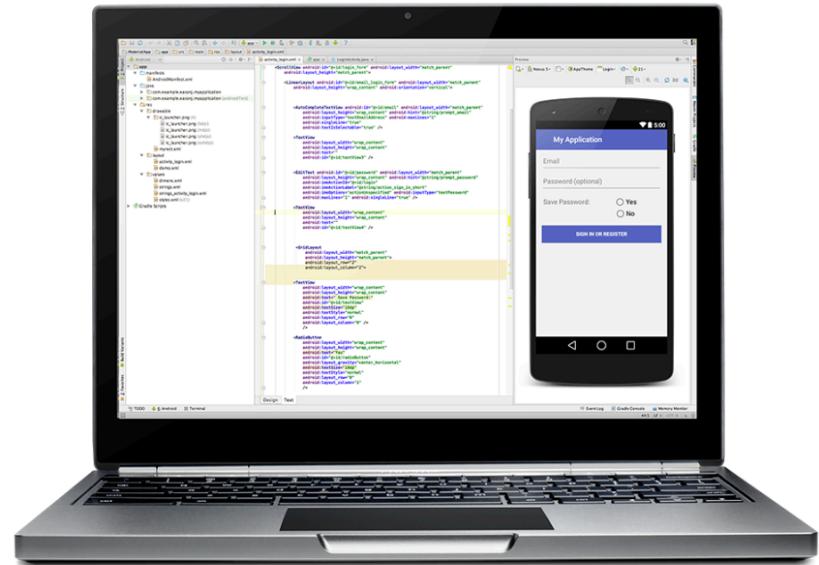
- The way the system architecture is set up is fairly open:
 - App design is somewhat up to you, but you still have to live with the Android execution model.
- Start with the different **screens/layouts (Views)** that the user will see. These are controlled by the different **Activities (Controllers)** 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)



Get Started with Android Studio

Everything you need to build incredible app experiences on phones and tablets, Wear, TV, and Auto.

- › Set up Android Studio
- › Build your first app
- › Learn about Android
- › Sample projects



Latest





Getting Started

- Building Your First App**
- Supporting Different Devices**
- Managing the Activity Lifecycle**
- Building a Dynamic UI with Fragments**
- Saving Data**
- Interacting with Other Apps**
- Working with System Permissions**
- Building Apps with Content Sharing**
- Building Apps with Multimedia**
- Building Apps with Graphics & Animation**
- Building Apps with Connectivity & the Cloud**

Getting Started

Welcome to Training for Android developers. Here you'll find sets of lessons within classes that describe how to accomplish a specific task with code samples you can re-use in your app. Classes are organized into several groups you can see at the top-level of the left navigation.

This first group, *Getting Started*, teaches you the bare essentials for Android app development. If you're a new Android app developer, you should complete each of these classes in order.

If you prefer to learn through interactive video training, check out this trailer for a course about the fundamentals of Android development.

[START THE VIDEO COURSE](#)



Online video courses

If you prefer to learn through interactive video training, check out these free courses.





Introduction

[App Fundamentals](#)

[Device Compatibility](#)

[System Permissions](#)

[App Components](#)

[App Resources](#)

[App Manifest](#)

[User Interface](#)

[Animation and Graphics](#)

[Computation](#)

[Media and Camera](#)

[Location and Sensors](#)

[Connectivity](#)

[Text and Input](#)

Introduction to Android

Android provides a rich application framework that allows you to build innovative apps and games for mobile devices in a Java language environment. The documents listed in the left navigation provide details about how to build apps using Android's various APIs.

If you're new to Android development, it's important that you understand the following fundamental concepts about the Android app framework:

Apps provide multiple entry points

Android apps are built as a combination of distinct components that can be invoked individually. For instance, an individual *activity* provides a single screen for a user interface, and a service independently performs work in the background.

From one component you can start another component using an *intent*. You can even start a component in a different app, such as an activity in a maps app to show an address. This model provides multiple entry points for a single app and allows any app to behave as a user's "default" for an action that other apps may invoke.

Apps adapt to different devices

Android provides an adaptive app framework that allows you to provide unique resources for different device configurations. For example, you can create different XML layout files for different screen sizes and the system determines which layout to apply based on the current device's screen size.

You can query the availability of device features at runtime if any app features require specific hardware such as a camera. If necessary, you can also declare features your app requires so app markets such as Google Play Store do not allow installation on devices that do not support that feature.

[Learn more](#)

To learn how apps work, start with [App Fundamentals](#).

To begin coding right away, read [Building Your First App](#).



Introduction

App Components

Intents and Intent Filters

Activities

Services

Content Providers

App Widgets

Processes and Threads

App Resources

App Manifest

User Interface

Animation and Graphics

Computation

Media and Camera



BLOG ARTICLES

Using DialogFragments

In this post, I'll show how to use DialogFragments with the v4 support library (for backward compatibility on pre-Honeycomb devices) to show a simple edit dialog and return a result to the calling Activity using an interface.

Fragments For All

Today we've released a static library that exposes the

App Components

Android's application framework lets you create rich and innovative apps using a set of reusable components. This section explains how you can build the components that define the building blocks of your app and how to connect them together using intents.

[INTENTS AND INTENT FILTERS >](#)

TRAINING

Managing the Activity Lifecycle

This class explains important lifecycle callback methods that each Activity instance receives and how you can use them so your activity does what the user expects and does not consume system resources when your activity doesn't need them.

Building a Dynamic UI with Fragments

This class shows you how to create a dynamic user

**Introduction****App Components****App Resources****App Manifest****User Interface****Overview****Layouts****Input Controls****Input Events****Menus****Settings****Dialogs****Notifications****Toasts****BLOG ARTICLES****Say Goodbye to the Menu Button**

As Ice Cream Sandwich rolls out to more devices, it's important that you begin to migrate your designs to the action bar in order to promote a consistent Android user experience.

New Layout Widgets: Space and GridLayout

Ice Cream Sandwich (ICS) sports two new widgets

User Interface

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.

[OVERVIEW >](#)**TRAINING****Implementing Effective Navigation**

This class shows you how to plan out the high-level screen hierarchy for your application and then choose appropriate forms of navigation to allow users to effectively and intuitively traverse your content.

Designing for Multiple Screens

Android powers hundreds of device types with several different screen sizes, ranging from small



Introduction

App Components

App Resources

App Manifest

User Interface

Overview

Layouts

Input Controls

Buttons

Text Fields

Checkboxes

Radio Buttons

Toggle Buttons

Spinners

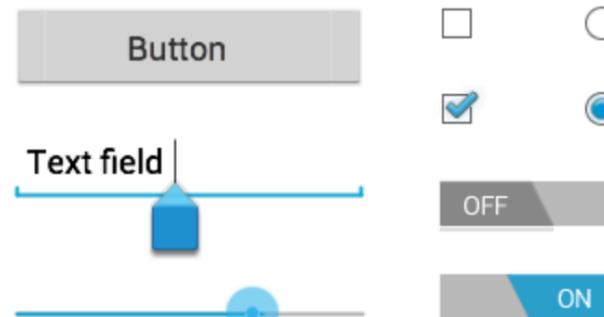
Pickers

Input Controls

Input controls are the interactive components in your app's user interface. Android provides a wide variety of controls you can use in your UI, such as buttons, text fields, seek bars, checkboxes, zoom buttons, toggle buttons, and many more.

Adding an input control to your UI is as simple as adding an XML element to your [XML layout](#). For example, here's a layout with a text field and button:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <EditText android:id="@+id/edit_message"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message" />
    <Button android:id="@+id/button_send"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send"
        android:onClick="sendMessage" />
</LinearLayout>
```





Common Controls

Control Type	Description	Related Classes
Button	A push-button that can be pressed, or clicked, by the user to perform an action.	Button
Text field	An editable text field. You can use the AutoCompleteTextView widget to create a text entry widget that provides auto-complete suggestions	EditText , AutoCompleteTextView
Checkbox	An on/off switch that can be toggled by the user. You should use checkboxes when presenting users with a group of selectable options that are not mutually exclusive.	CheckBox
Radio button	Similar to checkboxes, except that only one option can be selected in the group.	RadioGroup RadioButton
Toggle button	An on/off button with a light indicator.	ToggleButton
Spinner	A drop-down list that allows users to select one value from a set.	Spinner
Pickers	A dialog for users to select a single value for a set by using up/down buttons or via a swipe gesture. Use a DatePicker widget to enter the values for the date (month, day, year) or a TimePicker widget to enter the values for a time (hour, minute, AM/PM), which will be formatted automatically for the user's locale.	DatePicker , TimePicker



Introduction

App Components

App Resources

App Manifest

User Interface

Overview

Layouts

Input Controls

Buttons

Text Fields

Checkboxes

Radio Buttons

Toggle Buttons

Spinners

Pickers

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"  
    ... />
```

In this document

- Responding to Click Events
- Using an OnClickListener
- Styling Your Button
- Borderless button
- Custom background

Key classes

- `Button`
- `ImageButton`



Introduction

App Components

App Resources

App Manifest

User Interface

Overview

Layouts

Input Controls

Buttons

Text Fields

Checkboxes

Radio Buttons

Toggle Buttons

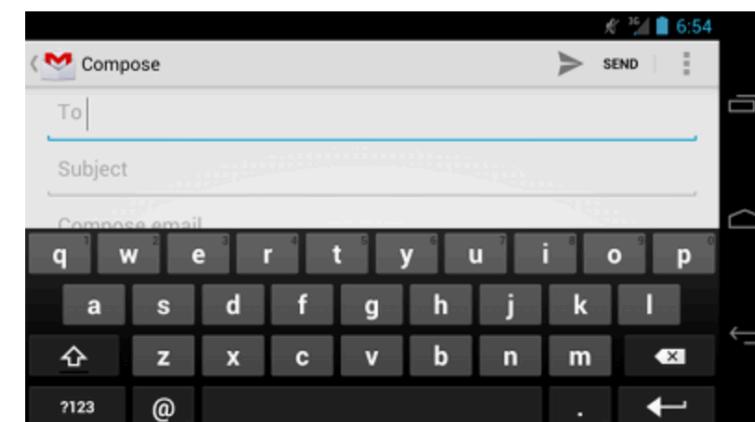
Spinners

Pickers

Text Fields

A text field allows the user to type text into your app. It can be either single line or multi-line. Touching a text field places the cursor and automatically displays the keyboard. In addition to typing, text fields allow for a variety of other activities, such as text selection (cut, copy, paste) and data look-up via auto-completion.

You can add a text field to you layout with the `EditText` object. You should usually do so in your XML layout with a `<EditText>` element.



In this document

- › Specifying the Keyboard Type
 - › Controlling other behaviors
- › Specifying Keyboard Actions
 - › Responding to action button events
- › Setting a custom action button label
- › Adding Other Keyboard Flags
- › Providing Auto-complete Suggestions

Key classes

- › `EditText`
- › `AutoCompleteTextView`

Specifying the Keyboard Type



Introduction

App Components

App Resources

App Manifest

User Interface

Overview

Layouts

Input Controls

Buttons

Text Fields

Checkboxes

Radio Buttons

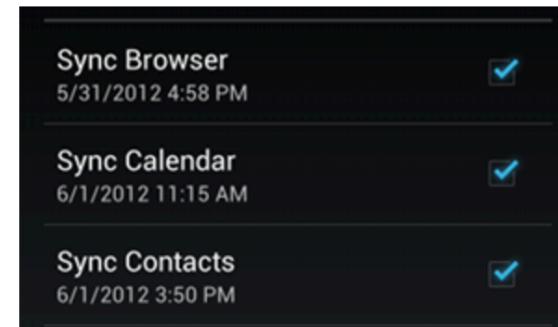
Toggle Buttons

Spinners

Pickers

Checkboxes

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.

Responding to Click Events

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

To define the click event handler for a checkbox, add the `android:onClick` attribute to the `<CheckBox>` 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.

In this document

› [Responding to Click Events](#)

Key classes

› [CheckBox](#)



Introduction

App Components

App Resources

App Manifest

User Interface

Overview

Layouts

Input Controls

Buttons

Text Fields

Checkboxes

Radio Buttons

Toggle Buttons

Spinners

Pickers

Radio Buttons

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.

Responding to Click Events

When the user selects one of the radio buttons, the corresponding [RadioButton](#) object receives an on-click event.

To define the click event handler for a button, add the `android:onClick` attribute to the `<RadioButton>` 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.

In this document

› [Responding to Click Events](#)

Key classes

› [RadioButton](#)

› [RadioGroup](#)



Introduction

App Components

App Resources

App Manifest

User Interface

Overview

Layouts

Input Controls

Buttons

Text Fields

Checkboxes

Radio Buttons

Toggle Buttons

Spinners

Pickers

Input Events

Toggle Buttons

A toggle button allows the user to change a setting between two states.

You can add a basic toggle button to your layout with the `ToggleButton` object. Android 4.0 (API level 14) introduces another kind of toggle button called a switch that provides a slider control, which you can add with a `Switch` object.

If you need to change a button's state yourself, you can use the `CompoundButton.setChecked()` or `CompoundButton.toggle()` methods.



Toggle buttons



Switches (in Android 4.0+)

In this document

> [Responding to Button Presses](#)

Key classes

> [ToggleButton](#)

> [Switch](#)

> [CompoundButton](#)

Responding to Button Presses

To detect when the user activates the button or switch, create an `CompoundButton.OnCheckedChangeListener` object and assign it to the button by calling `setOnCheckedChangeListener()`. For example:

```
ToggleButton toggle = (ToggleButton) findViewById(R.id.togglebutton);
toggle.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
```



Introduction

App Components

App Resources

App Manifest

User Interface

Overview

Layouts

Input Controls

Buttons

Text Fields

Checkboxes

Radio Buttons

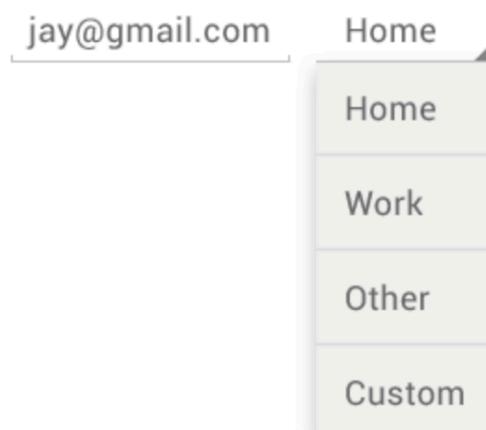
Toggle Buttons

Spinners

Pickers

Spinners

Spinners provide a quick way to select one value from a set. In the default state, a spinner shows its currently selected value. Touching the spinner displays a dropdown menu with all other available values, from which the user can select a new one.



You can add a spinner to your layout with the `Spinner` object. You should usually do so in your XML layout with a `<Spinner>` element. For example:

```
<Spinner  
    android:id="@+id/planets_spinner"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content" />
```

In this document

- › [Populate the Spinner with User Choices](#)
- › [Responding to User Selections](#)

Key classes

- › [Spinner](#)
- › [SpinnerAdapter](#)
- › [AdapterView.OnItemSelectedListener](#)

**Introduction****App Components****App Resources****App Manifest****User Interface****Overview****Layouts****Input Controls**

Buttons

Text Fields

Checkboxes

Radio Buttons

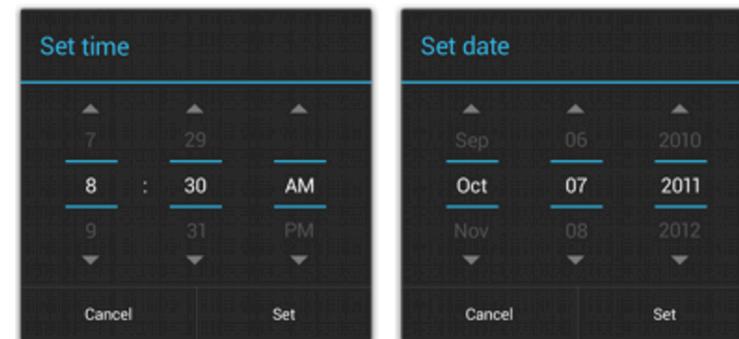
Toggle Buttons

Spinners

Pickers

Pickers

Android provides controls for the user to pick a time or pick a date as ready-to-use dialogs. Each picker provides controls for selecting each part of the time (hour, minute, AM/PM) or date (month, day, year). Using these pickers helps ensure that your users can pick a time or date that is valid, formatted correctly, and adjusted to the user's locale.



We recommend that you use [DialogFragment](#) to host each time or date picker. The [DialogFragment](#) manages the dialog lifecycle for you and allows you to display the pickers in different layout configurations, such as in a basic dialog on handsets or as an embedded part of the layout on large screens.

Although [DialogFragment](#) was first added to the platform in Android 3.0 (API level 11), if your app supports versions of Android older than 3.0—even as low as Android 1.6—you can use the [DialogFragment](#) class that's backward compatible.

In this document

- › [Creating a Time Picker](#)
- › [Extending DialogFragment for a time picker](#)
- › [Showing the time picker](#)
- › [Creating a Date Picker](#)
- › [Extending DialogFragment for a date picker](#)
- › [Showing the date picker](#)

Key classes

- › [DatePickerDialog](#)
- › [TimePickerDialog](#)
- › [DialogFragment](#)

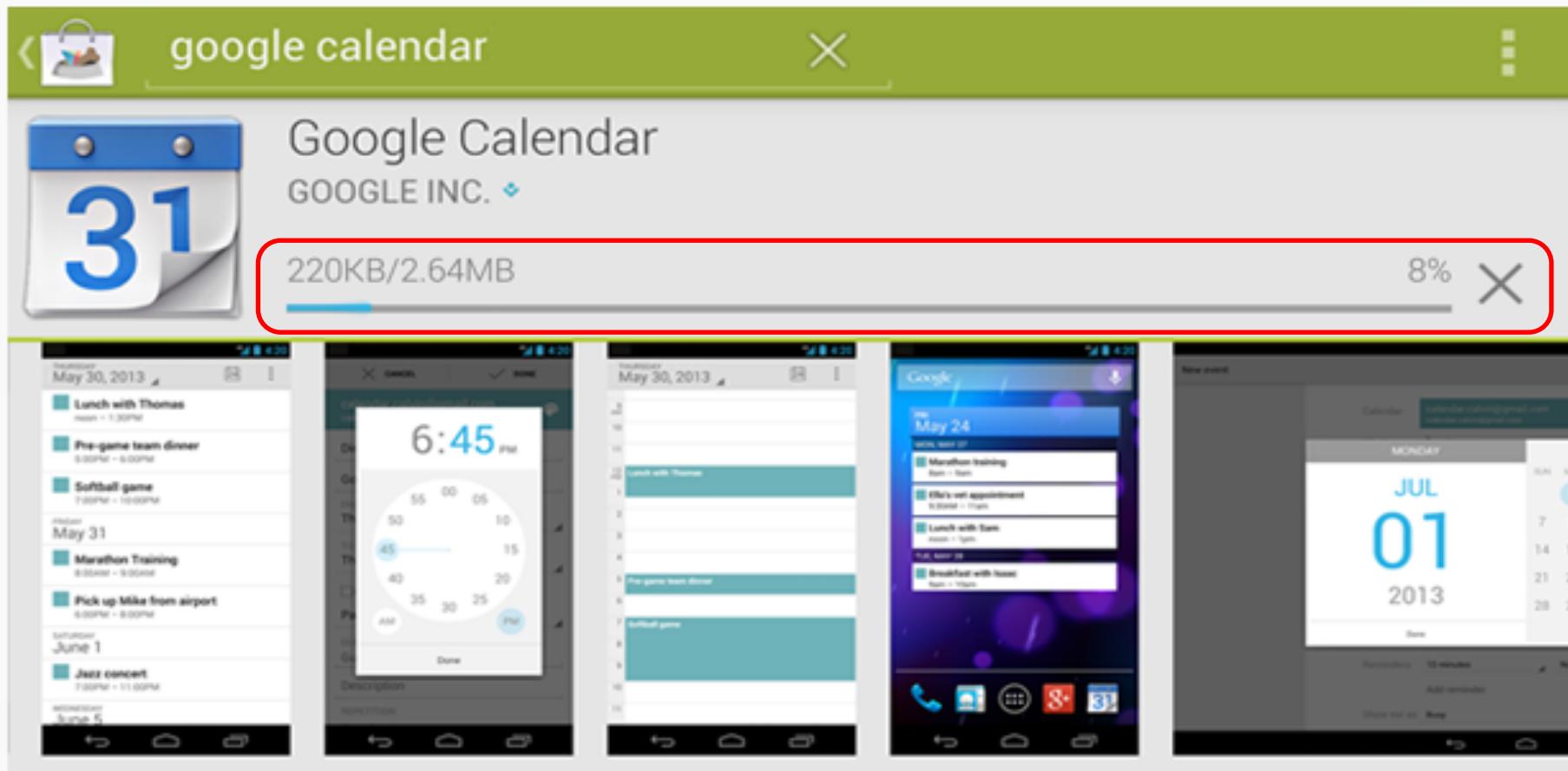
See also

- › [Fragments](#)



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%.



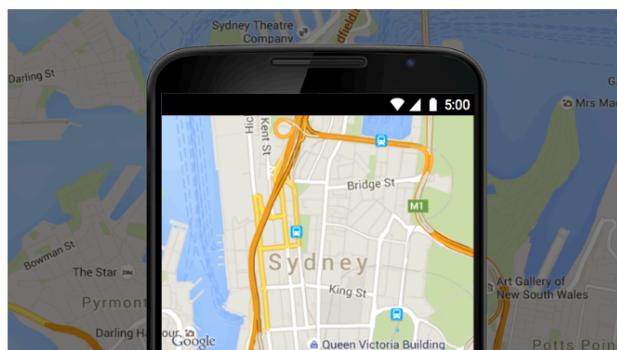
Google Maps Android API

Add Google Maps to your Android app.

[GET A KEY](#)[VIEW PRICING AND PLANS](#)[HOME](#)[GUIDES](#)[REFERENCE](#)[SAMPLES](#)[SUPPORT](#)[SEND FEEDBACK](#)

The best of Google Maps for every Android app

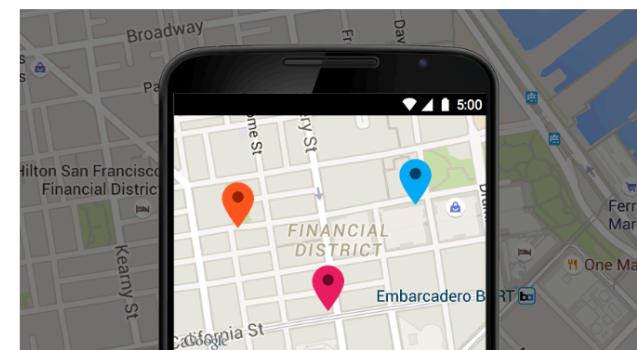
Build a custom map for your Android app using 3D buildings, indoor floor plans and more.



Maps

[Learn more](#)[3D buildings and floors](#)

Imagery

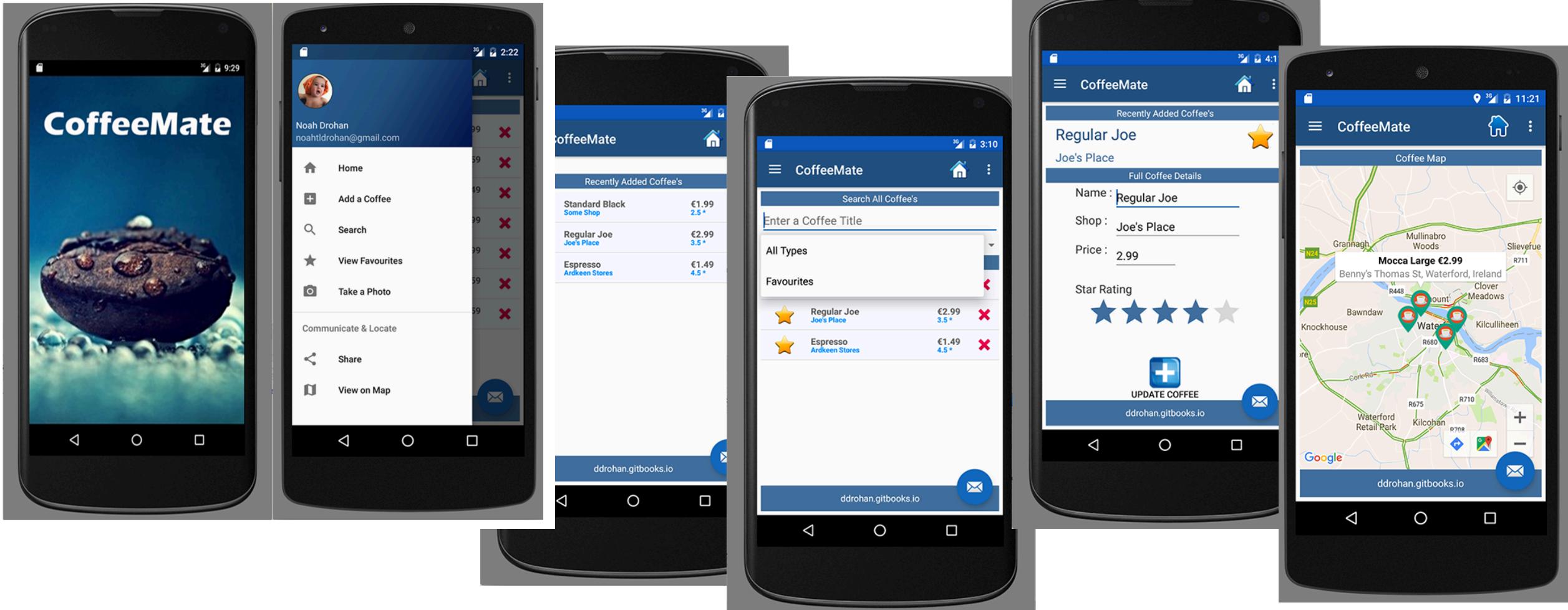
[Aerial, street view, and satellite](#)[Aerial, street view, and satellite](#)[Aerial, street view, and satellite](#)

Customization

[Add controls and infowindows](#)



Ultimate Case Study



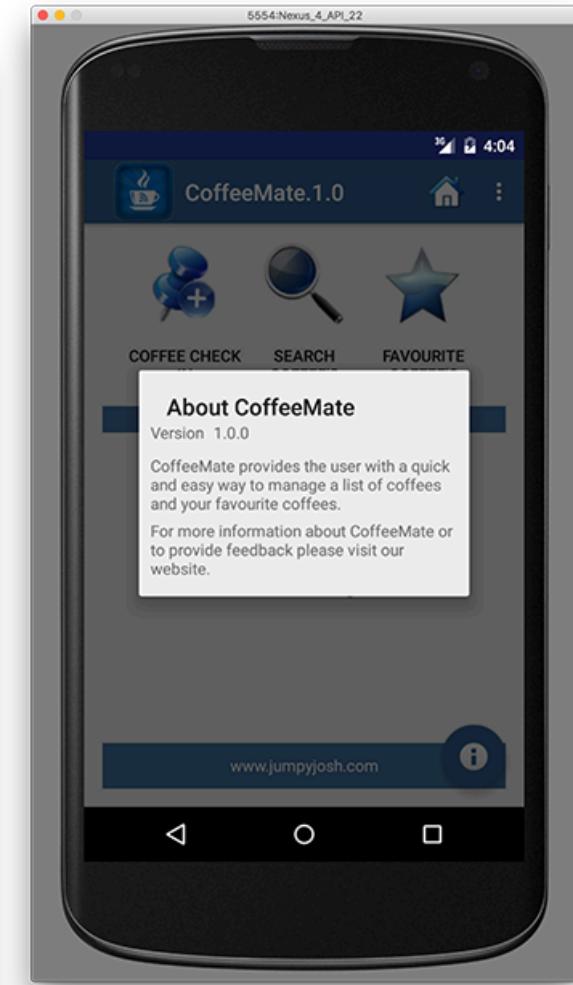
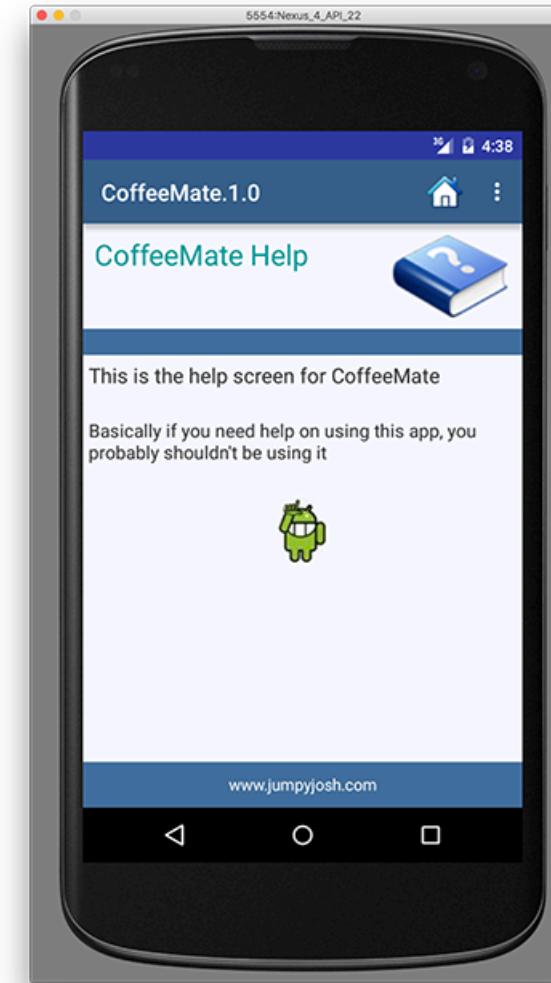
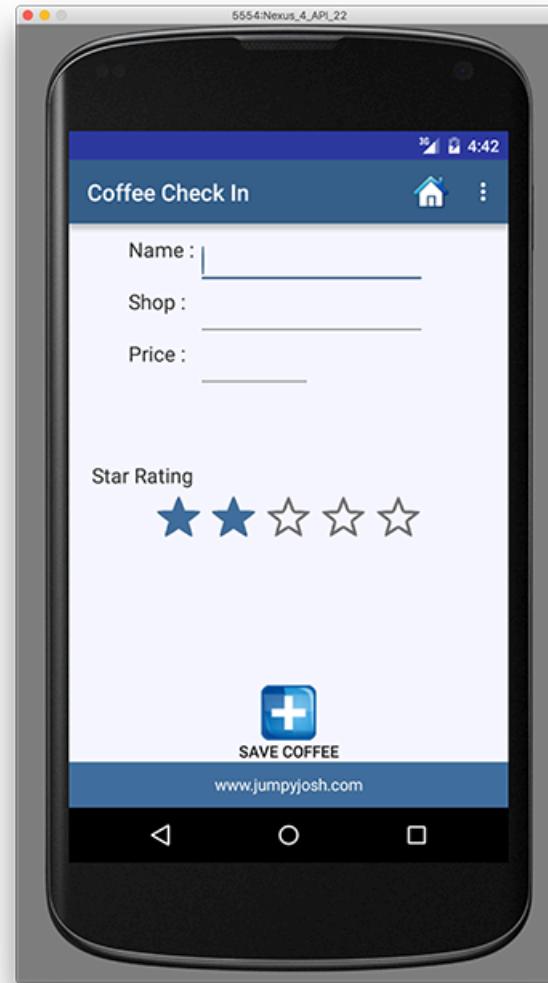
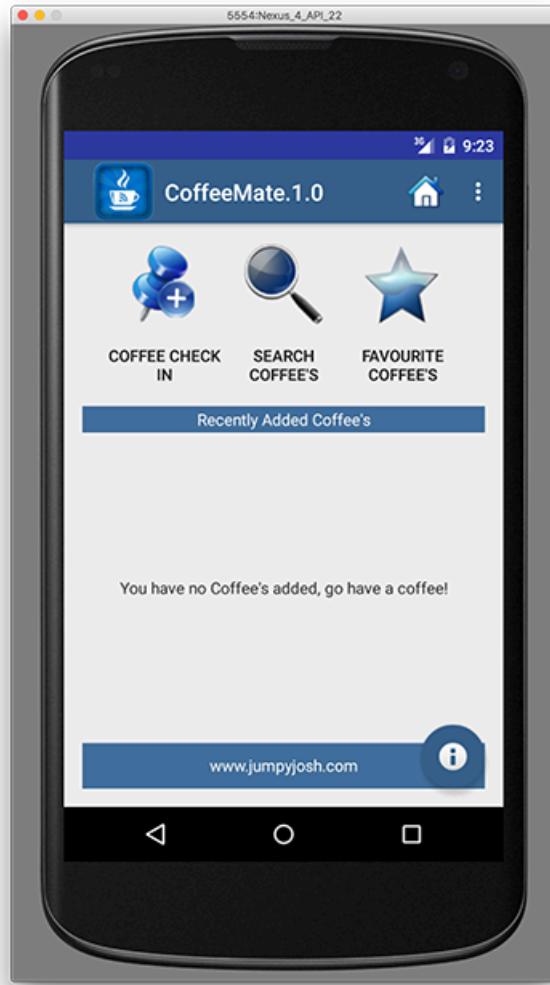


CoffeeMate 1.0

Using Buttons,
Multiple Layouts
&
Menus



CoffeeMate 1.0



The screenshot shows the Android Studio Project Structure view for the 'CoffeeMate.1.0' project. The left sidebar displays the project tree under the 'app' module. A green highlight covers the 'ie.cm (androidTest)' and 'ie.cm (test)' sections. Blue arrows point from the right side of the slide to specific resources in the project tree:

- A single blue arrow points to the 'activities' folder in the 'java' section.
- A horizontal blue arrow points to the 'layout' folder in the 'res' section.
- A horizontal blue arrow points to the 'menu' folder in the 'res' section.
- A horizontal blue arrow points to the 'values' folder in the 'res' section.

Project Structure – Version 1.0

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

Android Anatomy

43



Layout – *home*

The screenshot shows the Android Studio layout editor for the `home.xml` file. The layout consists of a CoordinatorLayout containing an AppBarLayout, an include tag pointing to `@layout/content_home`, and a FloatingActionButton. The content area features three icons: COFFEE CHECK IN, SEARCH COFFEE'S, and FAVOURITE COFFEE'S. Below this is a RecyclerView displaying a list of items: Item 1 (Sub Item 1), Item 2 (Sub Item 2), Item 3 (Sub Item 3 with a note: "You have no Coffee's added, go have a coffee!"), and Item 4 (Sub Item 4). The properties panel on the right shows the `CoordinatorLayout` has `layout:width="match_parent"` and `layout:height="match_parent"`.

Component Tree

- Device Screen
 - CustomView – android.support.design.widget.CoordinatorLayout
 - CustomView – android.support.design.widget.AppBarLayout
 - include – @layout/content_home
 - fab (CustomView) – android.support.design.widget.FloatingActionButton

Properties

<code>layout:width</code>	<code>match_parent</code>
<code>layout:height</code>	<code>match_parent</code>
<code>view:klass</code>	<code>android.support.des</code>
<code>style</code>	
<code>accessibilityLiveRegion</code>	
<code>accessibilityTraversalAfter</code>	
<code>accessibilityTraversalBefore</code>	
<code>alpha</code>	
<code>background</code>	
<code>backgroundTint</code>	
<code>backgroundTintMode</code>	
<code>clickable</code>	



XML View – *home* *

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



Layout – *content_home*

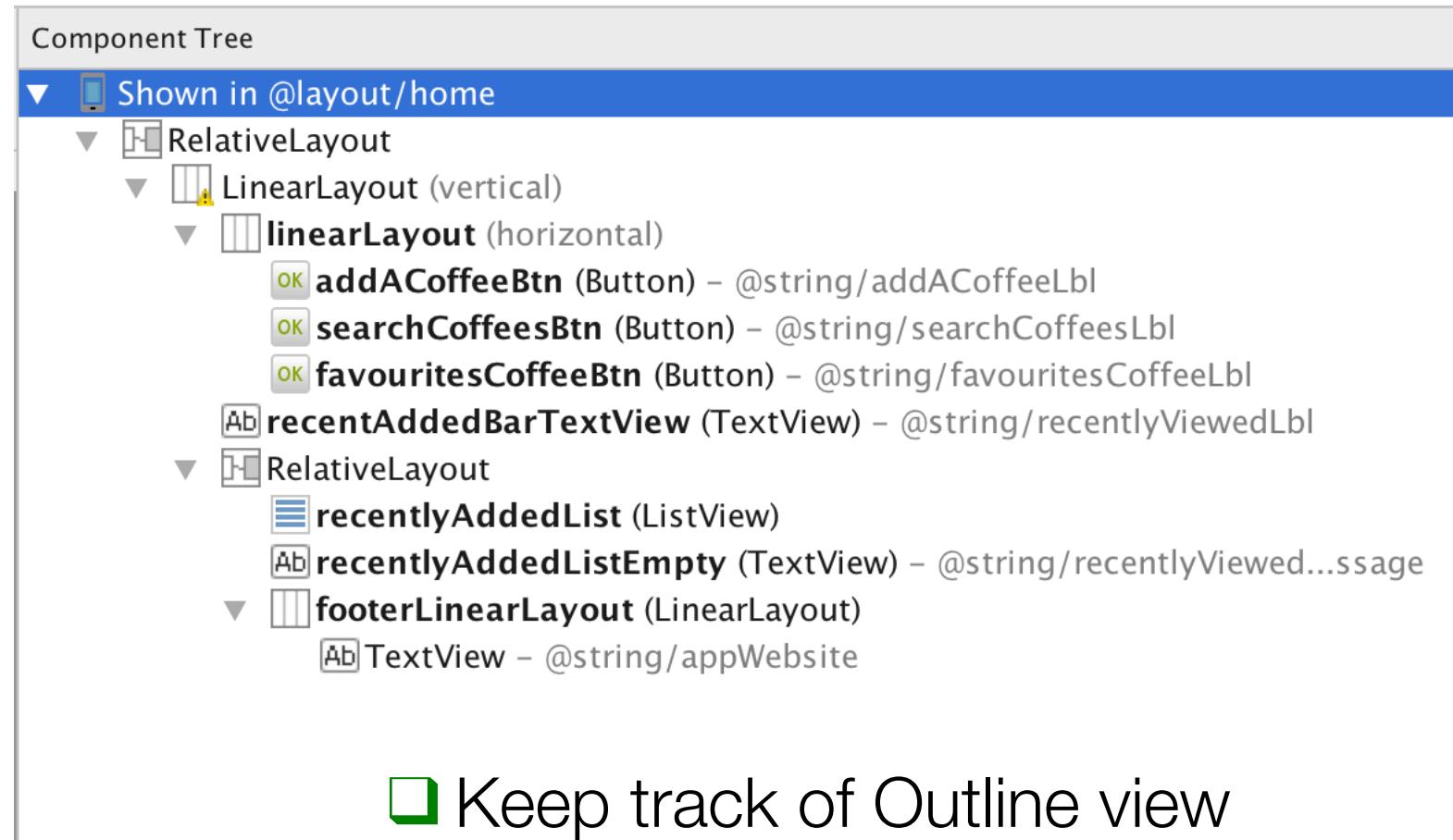
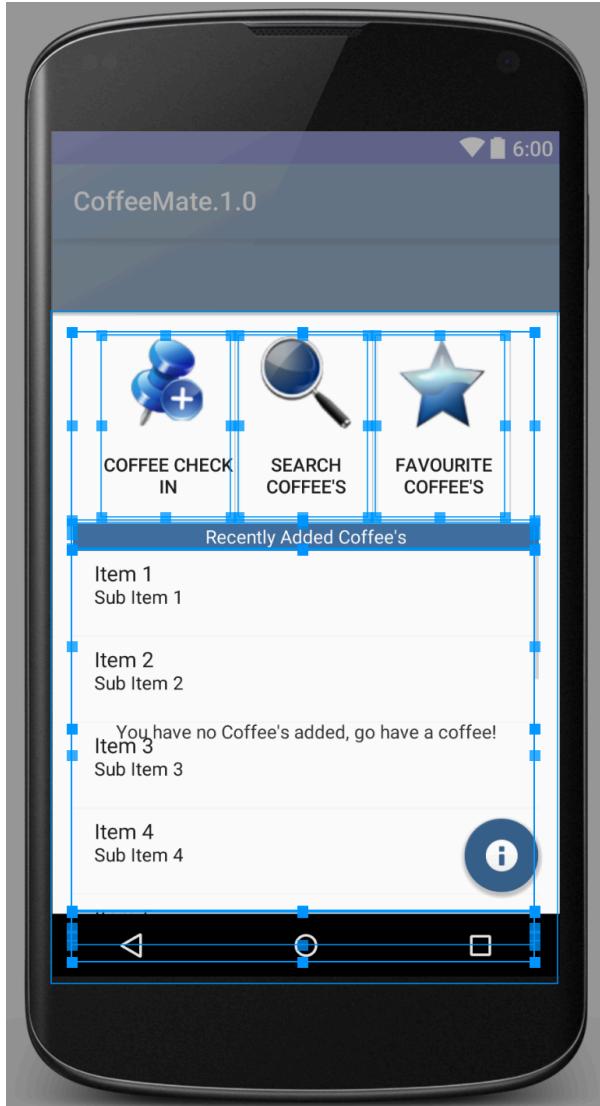
The screenshot shows the Android Studio Layout Editor with the following details:

- Title Bar:** content_home.xml
- Toolbar:** Includes icons for file operations (New, Open, Save, etc.), device selection (Nexus 4), theme (AppTheme), and version (Home, 23).
- Palette:** Shows categories: Layouts (FrameLayout, LinearLayout (Horizontal, Vertical), TableLayout, TableRow, GridLayout, RelativeLayout), Widgets (Plain TextView, Large Text, Medium Text, Small Text, Button, Small Button, RadioButton, CheckBox, Switch, ToggleButton, ImageButton, ImageView, ProgressBar (Large, Normal, Small, Horizontal), SeekBar, RatingBar, Spinner, WebView), and Text Fields (Plain Text, Person Name, Password, Password (Numeric), E-mail, Phone, Postal Address, Multiline Text, Time).
- Component Tree:** Lists the layout structure:
 - Shown in @layout/home
 - RelativeLayout
 - LinearLayout (vertical)
 - linearLayout (horizontal)
 - addACoffeeBtn (Button) – @string/addACoffeeLbl
 - searchCoffeesBtn (Button) – @string/searchCoffeesLbl
 - favouritesCoffeeBtn (Button) – @string/favouritesCoffeeLbl
 - recentAddedBarTextView (TextView) – @string/recentlyViewedLbl
 - RelativeLayout
 - recentlyAddedList (ListView)
 - recentlyAddedListEmpty (TextView) – @string/recentlyViewed...ssage
 - footerLinearLayout (LinearLayout)
 - TextView – @string/appWebsite
- Properties:** A table showing properties and their values:

layout:width	
layout:height	
style	
accessibilityLiveRegion	
accessibilityTraversalAfter	
accessibilityTraversalBefore	
alpha	
background	
backgroundTint	
backgroundTintMode	
clickable	
contentDescription	
contextClickable	
elevation	
focusable	
focusableInTouchMode	
foreground	
foregroundGravity	

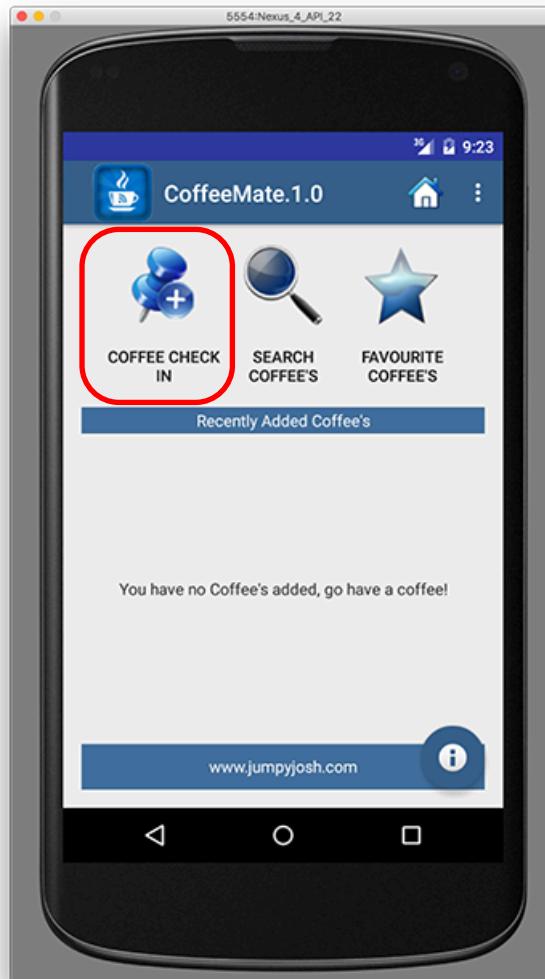


Layout – Outline View *



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

XML View - content_home *



This part defines the 3 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"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto" android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    app:layout_behavior="android.support.design.widget.AppBarLayout$ScrollingViewBehavior" tools:context=".Home">

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <LinearLayout
            android:orientation="horizontal"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:weightSum="1"
            android:id="@+id/linearLayout"
            android:layout_gravity="center_horizontal"
            android:gravity="center">

            <Button
                android:id="@+id/addACoffeeBtn"
                style="@android:style/Holo.Light.ButtonBar"
                android:layout_width="100dp"
                android:layout_height="140dp"
                android:layout_margin="2dp"
                android:drawableTop="@drawable/ic_add_location"
                android:text="Coffee Check In"
                android:layout_gravity="center"
                android:onClick="add" />

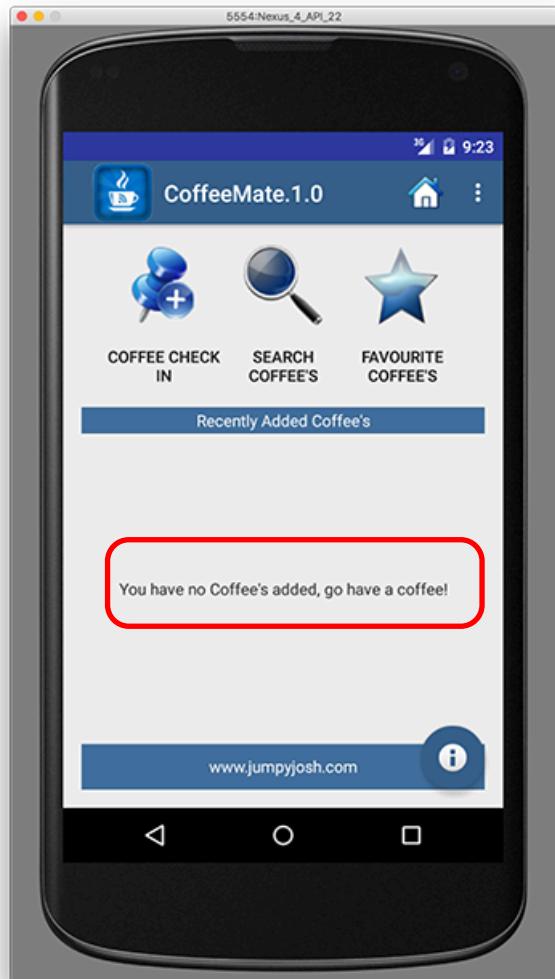
            <Button
                android:id="@+id/searchCoffeesBtn"
                style="@android:style/Holo.Light.ButtonBar"
                android:layout_width="100dp"
                android:layout_height="140dp"
                android:layout_margin="2dp"
                android:drawableTop="@drawable/ic_list"
                android:text="Search Coffee's"
                android:layout_gravity="center" />

            <Button
                android:id="@+id/favouritesCoffeeBtn"
                style="@android:style/Holo.Light.ButtonBar"
                android:layout_width="100dp"
                android:layout_height="140dp"
                android:layout_margin="2dp"
                android:drawableTop="@drawable/ic_favourite_off"
                android:text="Favourite Coffee's"
                android:layout_gravity="center" />

        </LinearLayout>
    </LinearLayout>
</RelativeLayout>
```

Note the use of
an 'onClick'
attribute

XML View - content_home *



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

```
<TextView  
    android:id="@+id/recentAddedBarTextView"  
    style="@style/banner"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="Recently Added Coffee's"  
    android:layout_gravity="center_vertical"  
    android:gravity="center" />  
  
<RelativeLayout  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent" >  
    <ListView  
        android:id="@+id/recentlyAddedList"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:layout_above="@+id/footerLinearLayout"  
        android:layout_alignParentTop="true" />  
  
    <TextView  
        android:id="@+id/recentlyAddedListEmpty"  
        android:layout_width="fill_parent"  
        android:layout_height="fill_parent"  
        android:gravity="center"  
        android:text="You have no Coffee's added, go have a coffee!"  
        android:textColor="@color/appFontColor"  
        android:layout_above="@+id/footerLinearLayout" />  
  
    <LinearLayout  
        android:id="@+id/footerLinearLayout"  
        style="@style/footer"  
        android:background="@color/bannerBGColor"  
        android:layout_width="fill_parent"  
        android:layout_alignParentBottom="true"  
        android:layout_alignParentStart="true">  
  
        <TextView  
            android:layout_width="fill_parent"  
            android:layout_height="wrap_content"  
            android:gravity="center"  
            android:paddingTop="10dip"  
            android:text="ddrohan.gitbooks.io"  
            android:textColor="@color/bgColor" />  
    </LinearLayout>  
</RelativeLayout>  
</LinearLayout>  
</RelativeLayout>
```



CoffeeMate Event Handler *

```
public class Home extends Base {  
  
    TextView recentList;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {...}  
  
    public void add(View v) { goToActivity(this,Add.class,null); } // Line 1  
  
    @Override  
    protected void onResume() {  
        super.onResume();  
  
        if(!coffeeList.isEmpty())  
            recentList.setText(coffeeList.toString());  
        else  
            recentList.setText("You have no Coffee's added, go have a coffee!");  
    }  
}
```

Note the use of a 'View' object

content_home

```
<Button  
    android:id="@+id/addACoffeeBtn"  
    style="@android:style/Holo.Light.ButtonBar"  
    android:layout_width="100dp"  
    android:layout_height="140dp"  
    android:layout_margin="2dp"  
    android:drawableTop="@drawable/ic_add_location"  
    android:text="Coffee Check In"  
    android:layout_gravity="center"  
    android:onClick="add" />
```

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 our layouts.

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

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

    <string name="versionLabel">Current Version:</string>
    <string name="version">1</string>

    <string name="searchCoffeesLbl">Search Coffee\'s</string>
    <string name="favouritesCoffeeLbl">Favourite Coffee\'s</string>
    <string name="addACoffeeLbl">Coffee Check In</string> (highlighted)
    <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="searchCoffeeNameHint">Enter a Coffee Title </string>
    <string name="informationLbl">Information</string>
    <string name="recentlyViewedListEmptyMessage">You have no Coffee\'s add

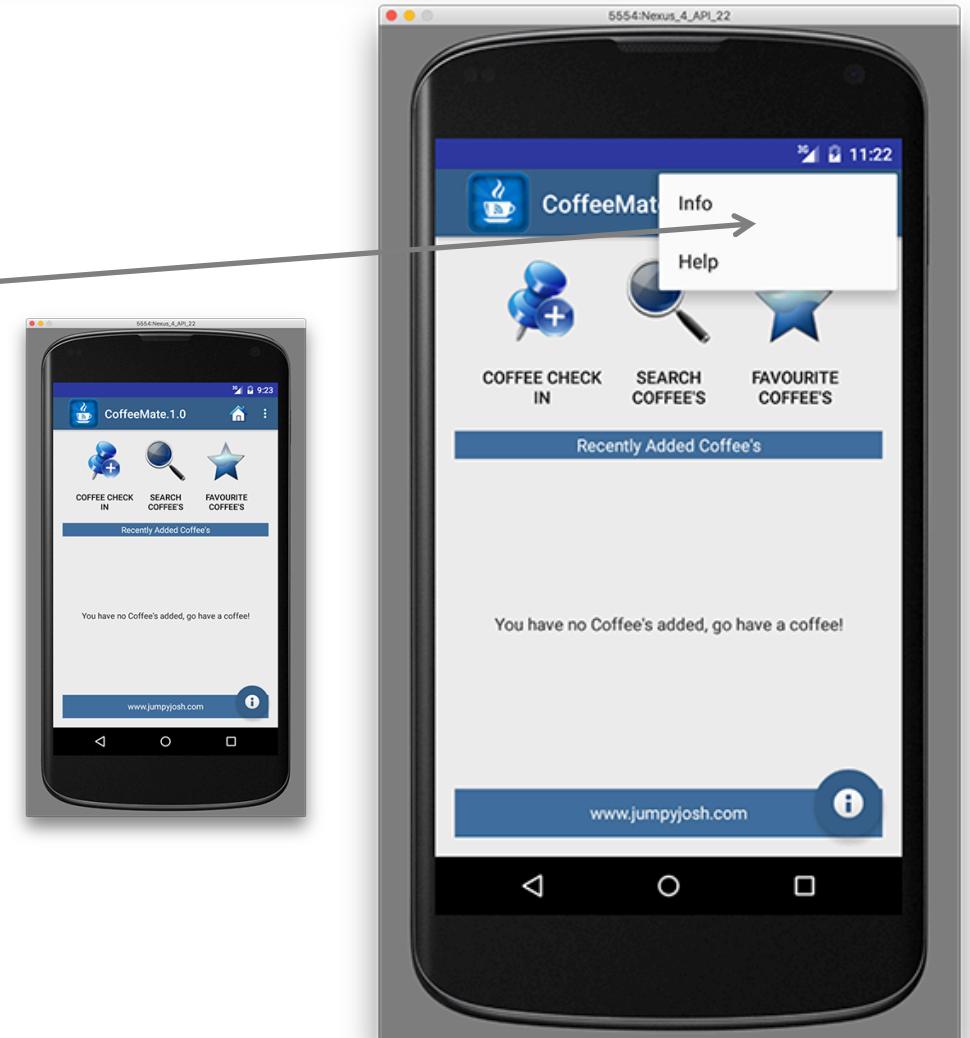
</resources>
```



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 donations, 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** *

The screenshot shows the Android Studio Project Structure. The project is named "CoffeeMate.1.0". The "app" module is selected. Inside "app", there are "manifests", "java", "res", and "menu" folders. The "main_menu.xml" file in the "menu" folder is highlighted with a red border and has a blue bar underneath it, indicating it is the active file. Other files like "AndroidManifest.xml", "ie.cm", "layout", "values", etc., are also visible.

```
<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=".Home">

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

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

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

</menu>
```

Menu Specification

Note the use of
an 'onClick'
attribute



CoffeeMate Menu Event Handler *

```
public class Base extends AppCompatActivity {  
  
    protected static ArrayList<Coffee> coffeeList = new ArrayList<~>();  
  
    protected void goToActivity(Activity current,  
                                Class<? extends Activity> activityClass,  
                                Bundle bundle) {...}  
  
    public void openInfoDialog(Activity current) {...}  
  
    @Override  
    public boolean onCreateOptionsMenu(Menu menu) {  
        // Inflate the menu; this adds items to the action bar if it is pre-  
        getMenuInflater().inflate(R.menu.main_menu, menu);  
        return true;  
    }  
  
    public void menuInfo(MenuItem m) { openInfoDialog(this); }  
  
    public void menuHelp(MenuItem m) { goToActivity(this, Help.class, null); }  
  
    public void menuHome(MenuItem m) { goToActivity(this, Home.class, null); }  
}
```

Menu Specification

```
<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=".Home">  
  
    <item android:id="@+id/menu_info"  
          android:icon="@drawable/about"  
          android:title="Info"  
          android:orderInCategory="100"  
          app:showAsAction="never"  
          android:onClick="menuInfo"/>  
  
    <item android:id="@+id/menu_help"  
          android:icon="@drawable/help"  
          android:title="Help"  
          android:orderInCategory="100"  
          app:showAsAction="never"  
          android:onClick="menuHelp"/>  
  
    <item android:id="@+id/menu_home"  
          android:icon="@drawable/home"  
          android:title="Home"  
          android:orderInCategory="100"  
          app:showAsAction="ifRoom"  
          android:onClick="menuHome"/>  
  
</menu>
```

inflate this resource as a 'Menu' (creates the 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



CoffeeMate - Menu Event Handler Alternative

'Help' Screen launched

check which 'menu item' was selected (by id)

```
@Override  
public boolean onMenuItemSelected(int featureId, MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.help:  
            goToActivity(this, Help.class, null);  
            break;  
        case R.id.info:  
            openInfoDialog(this);  
            break;  
        case R.id.home:  
            goToActivity(this, Home.class, null);  
            break;  
    }  
    return super.onMenuItemSelected(featureId, item);  
}
```





Switching 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 {  
    protected static ArrayList<Coffee> coffeeList = new ArrayList<?>();  
  
    protected void goToActivity(Activity current,  
                                Class<? extends Activity> activityClass,  
                                Bundle bundle) {  
        Intent newActivity = new Intent(current, activityClass);  
  
        if (bundle != null) newActivity.putExtras(bundle);  
  
        current.startActivity(newActivity);  
    }  
  
    public void openInfoDialog(Activity current) {...}  
  
    @Override  
    public boolean onCreateOptionsMenu(Menu menu) {  
        // Inflate the menu; this adds items to the action bar if it is present.  
        getMenuInflater().inflate(R.menu.main_menu, menu);  
        return true;  
    }  
  
    public void menuInfo(MenuItem m) { openInfoDialog(this); }  
  
    public void menuHelp(MenuItem m) { goToActivity(this, Help.class, null); }  
  
    public void menuHome(MenuItem m) { goToActivity(this, Home.class, null); }  
}
```

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

If you have never seen wildcards in generics before, this just means that we can pass in any subclass of Activity (as with Help & Home below).

A method to display a Dialog Window in the current Activity



class Add (1)

```
public class Add extends Base implements  
    OnClickListener { ←
```

```
    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);
```

```
        Button saveButton = (Button) findViewById(R.id.saveCoffeeBtn);  
        name = (EditText) findViewById(R.id.nameEditText);  
        shop = (EditText) findViewById(R.id.shopEditText);  
        price = (EditText) findViewById(R.id.priceEditText);  
        ratingBar = (RatingBar) findViewById(R.id.coffeeRatingBar);  
        saveButton.setOnClickListener(this);
```



Our Listener Interface

Binding to our Widgets

Attaching the Listener to the button



class Add (2)

public void onClick(View v) { ← Our Event Handler Code

```
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);
```

coffeeList.add(c); ← Adding the Coffee to our List

goToActivity(this, Home.class, null); ←

```
} else
    Toast.makeText(
        this,
```

```
        "You must Enter Something for "
        + "'Name'", "'Shop'" and "'Price'",
        Toast.LENGTH_SHORT).show();
```

}

Adding the Coffee to our List

Returning to our 'Home' Activity



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.

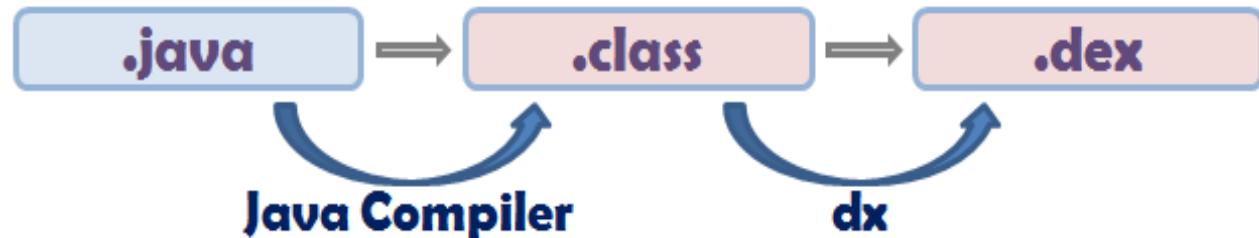
❑ Android Runtime (ART) 4.4 (see slide 12)



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

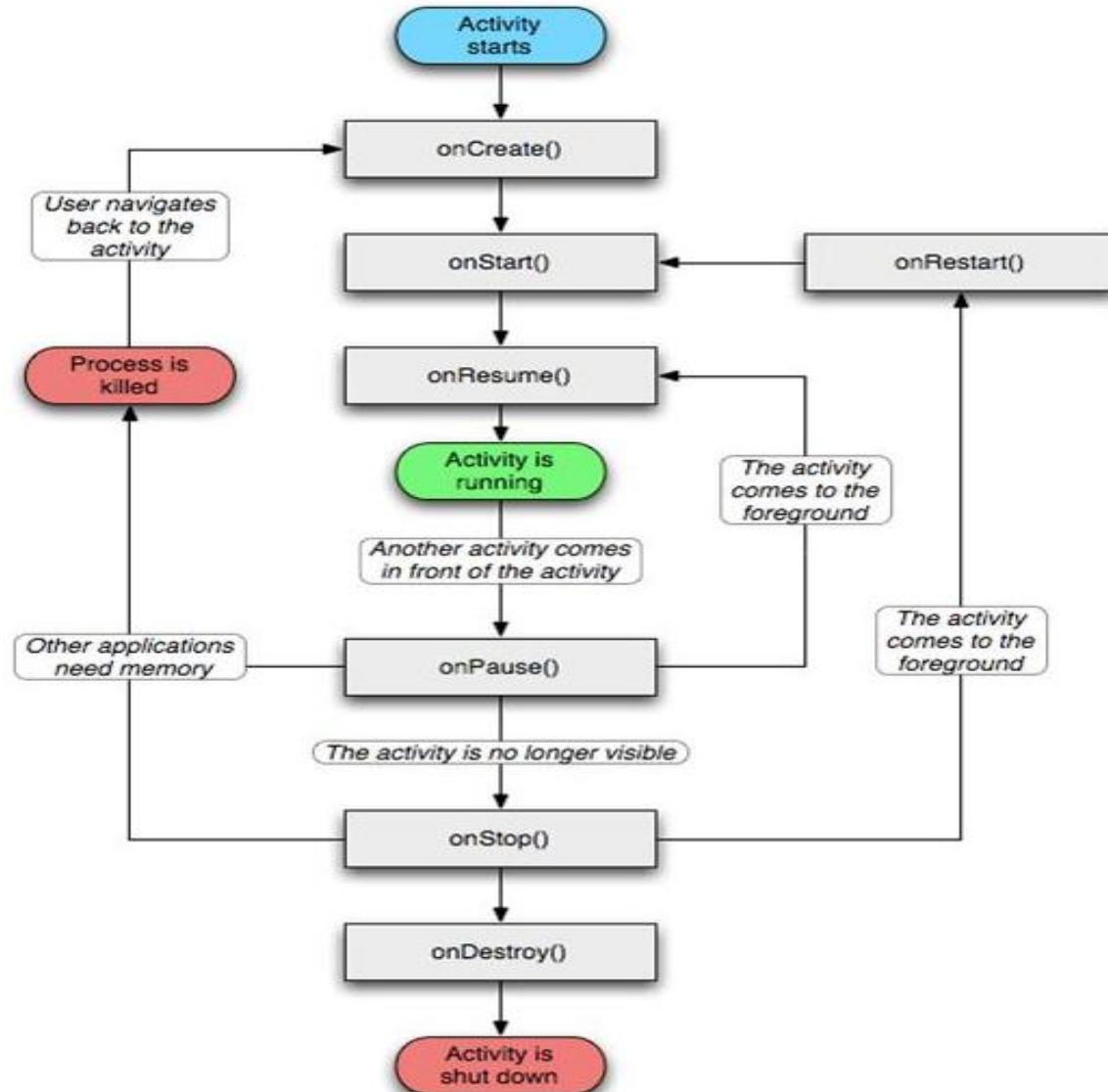


The Application/Activity Life Cycle

- ❑ 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?