

# Where is everything?



This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# Where is everything?

For a list of all the materials for the course,  
see the [course overview page](#) at

[developer.android.com/courses/adf-v2](https://developer.android.com/courses/adf-v2)



# Build your first app

Lesson 1



# 1.0 Introduction to Android



# Contents



- Android is an ecosystem
- Android platform architecture
- Android Versions
- Challenges of Android app development
- App fundamentals

# Android Ecosystem



# What is Android?

- Mobile operating system based on [Linux kernel](#)
- User Interface for touch screens
- Used on [over 80%](#) of all smartphones
- Powers devices such as watches, TVs, and cars
- Over 2 Million Android apps in Google Play store
- Highly customizable for devices / by vendors
- Open source

# Android user interaction

- Touch gestures: swiping, tapping, pinching
- Virtual keyboard for characters, numbers, and emoji
- Support for Bluetooth, USB controllers and peripherals

# Android and sensors

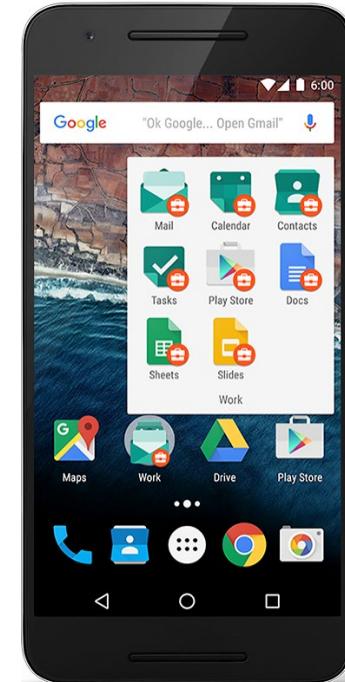
Sensors can discover user action and respond

- Device contents rotate as needed
- Walking adjusts position on map
- Tilting steers a virtual car or controls a physical toy
- Moving too fast disables game interactions



# Android home screen

- Launcher icons for apps
- Self-updating widgets for live content
- Can be multiple pages
- Folders to organize apps
- "OK Google"



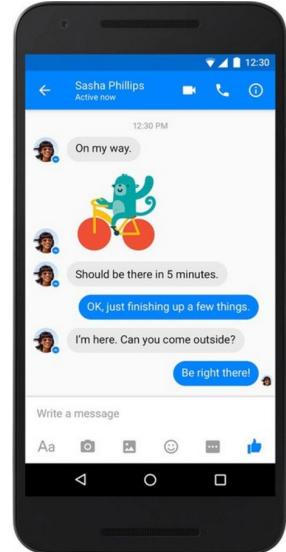
# Android app examples



Pandora



Pokemon GO



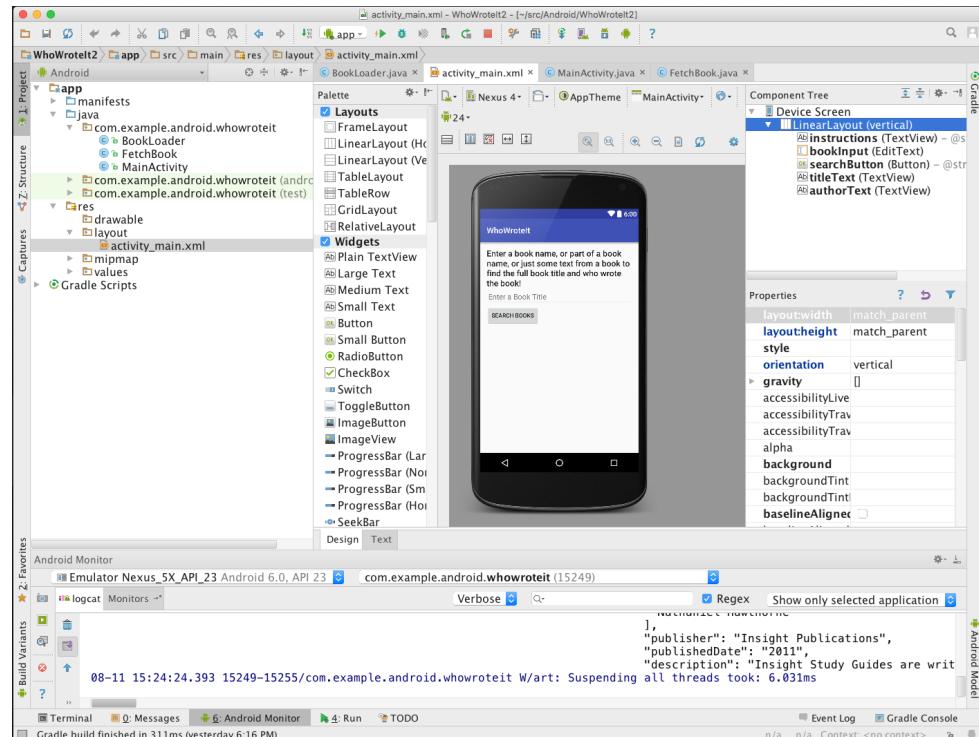
Facebook  
Messenger

# Android Software Developer Kit (SDK)

- Development tools (debugger, monitors, editors)
- Libraries (maps, wearables)
- Virtual devices (emulators)
- Documentation ([developers.android.com](https://developer.android.com))
- Sample code



# Android Studio



- Official Android IDE
- Develop, run, debug, test, and package apps
- Monitors and performance tools
- Virtual devices
- Project views
- Visual layout editor

# Google Play store

Publish apps through [Google Play](#) store:

- Official app store for Android
- Digital distribution service operated by Google

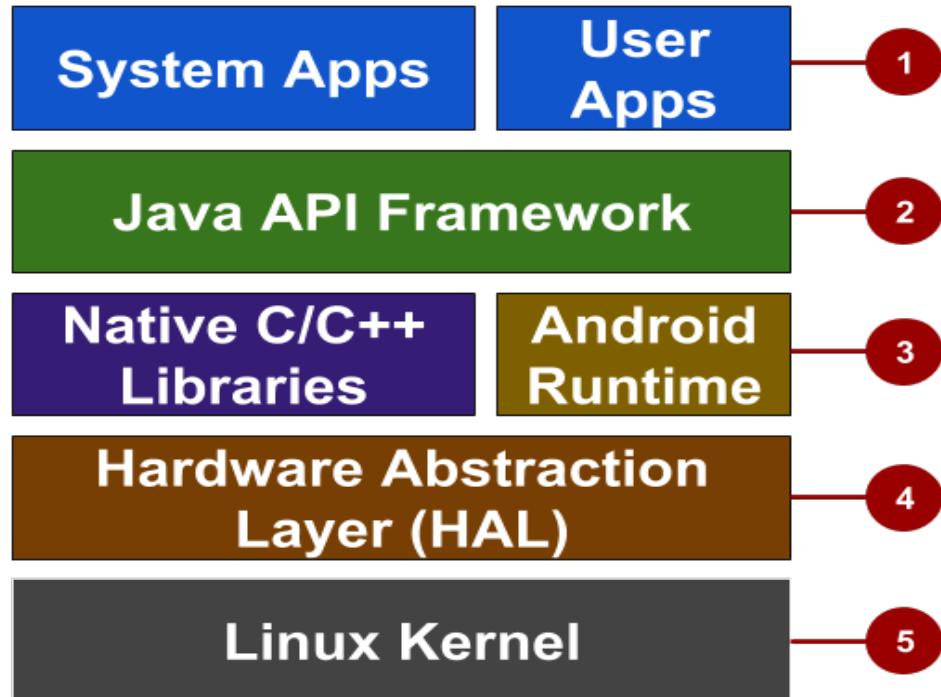


# Android Platform Architecture



# Android stack

1. System and user apps
2. Android OS API in Java framework
3. Expose native APIs; run apps
4. Expose device hardware capabilities
5. Linux Kernel



# System and user apps



- System apps have no special status
- System apps provide key capabilities to app developers

Example:

Your app can use a system app to deliver a SMS message.



# Java API Framework

The entire feature-set of the Android OS is available to you through APIs written in the Java language.

- View class hierarchy to create UI screens
- Notification manager
- Activity manager for life cycles and navigation



# Android runtime

Each app runs in its own process with its own instance of the Android Runtime.

# C/C++ libraries

- Core C/C++ Libraries give access to core native Android system components and services.

# Hardware Abstraction Layer (HAL)

- Standard interfaces that expose device hardware capabilities as libraries

Examples: Camera, bluetooth module

# Linux Kernel

- Threading and low-level memory management
- Security features
- Drivers



# Older Android versions



Codename	Version	Released	API Level
<b>Honeycomb</b>	3.0 - 3.2.6	Feb 2011	11 - 13
<b>Ice Cream Sandwich</b>	4.0 - 4.0.4	Oct 2011	14 - 15
<b>Jelly Bean</b>	4.1 - 4.3.1	July 2012	16 - 18
<b>KitKat</b>	4.4 - 4.4.4	Oct 2013	19 - 20
<b>Lollipop</b>	5.0 - 5.1.1	Nov 2014	21 - 22

[Android History](#) and  
[Platform Versions](#)  
for more and earlier  
versions before 2011

# Newer Android versions



Codename	Version	Released	API Level
<b><i>Marshmallow</i></b>	6.0 - 6.0.1	Oct 2015	23
<b><i>Nougat</i></b>	7.0 - 7.1	Sept 2016	24 - 25
<b><i>Oreo</i></b>	8.0 - 8.1	Sept 2017	26 - 27
<b><i>Pie</i></b>	9.0	Aug 2018	28

# App Development



# What is an Android app?

- One or more interactive screens
- Written using Java Programming Language and XML
- Uses the Android Software Development Kit (SDK)
- Uses Android libraries and Android Application Framework
- Executed by Android Runtime Virtual machine (ART)



# Challenges of Android development

- Multiple screen sizes and resolutions
- Performance: make your apps responsive and smooth
- Security: keep source code and user data safe
- Compatibility: run well on older platform versions
- Marketing: understand the market and your users  
(Hint: It doesn't have to be expensive, but it can be.)

# App building blocks

- Resources: layouts, images, strings, colors as XML and media files
- Components: activities, services, and helper classes as Java code
- Manifest: information about app for the runtime
- Build configuration: APK versions in Gradle config files



# Learn more

- [Android History](#)
- [Introduction to Android](#)
- [Platform Architecture](#)
- [UI Overview](#)
- [Platform Versions](#)
- [Supporting Different Platform Versions](#)
- [Android Studio User's Guide](#)



# What's Next?

- Concept Chapter: [1.0 Introduction to Android](#)
- No Practical



# END



# Build your first app

Lesson 1



# 1.1 Your first Android app



# Contents

- Android Studio
- Creating "Hello World" app in Android Studio
- Basic app development workflow with Android Studio
- Running apps on virtual and physical devices



# Prerequisites

- Java Programming Language
- Object-oriented programming
- XML - properties / attributes
- Using an IDE for development and debugging



# Android Studio

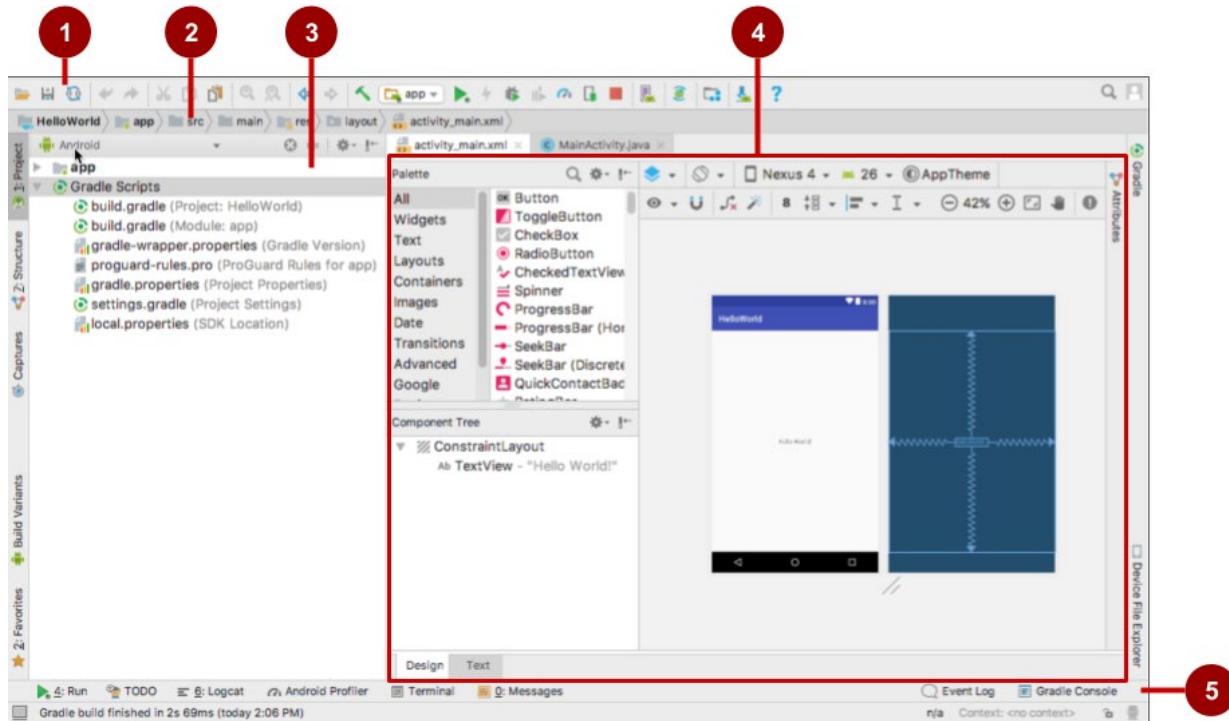


# What is Android Studio?

- Android integrated development environment (IDE)
- Project and Activity templates
- Layout editor
- Testing tools
- Gradle-based build
- Log console and debugger
- Emulators



# Android Studio interface



1. Toolbar
2. Navigation bar
3. Project pane
4. Editor
5. Tabs for other panes

# Installation Overview

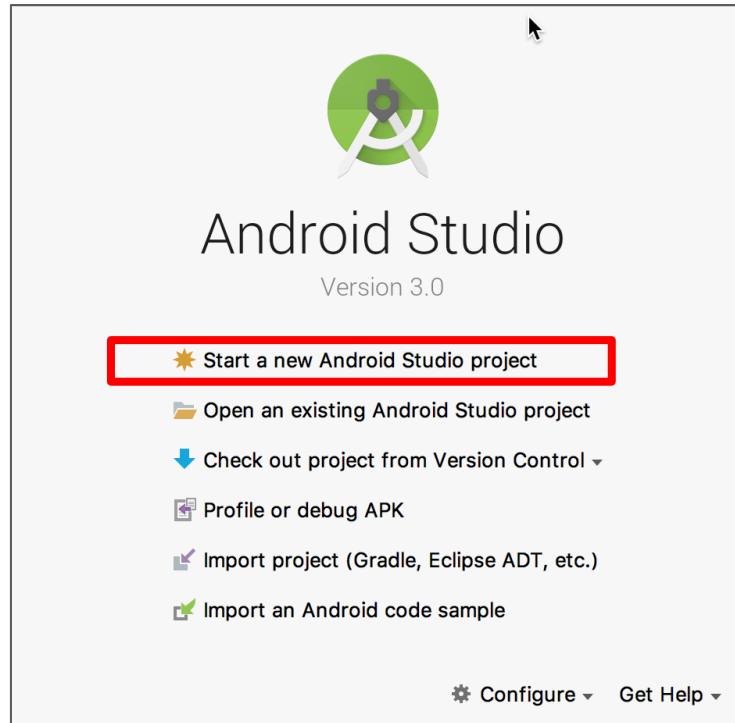
- Mac, Windows, or Linux
- Download and install Android Studio from  
<https://developer.android.com/studio/>
- See [1.1 P: Android Studio and Hello World](#)



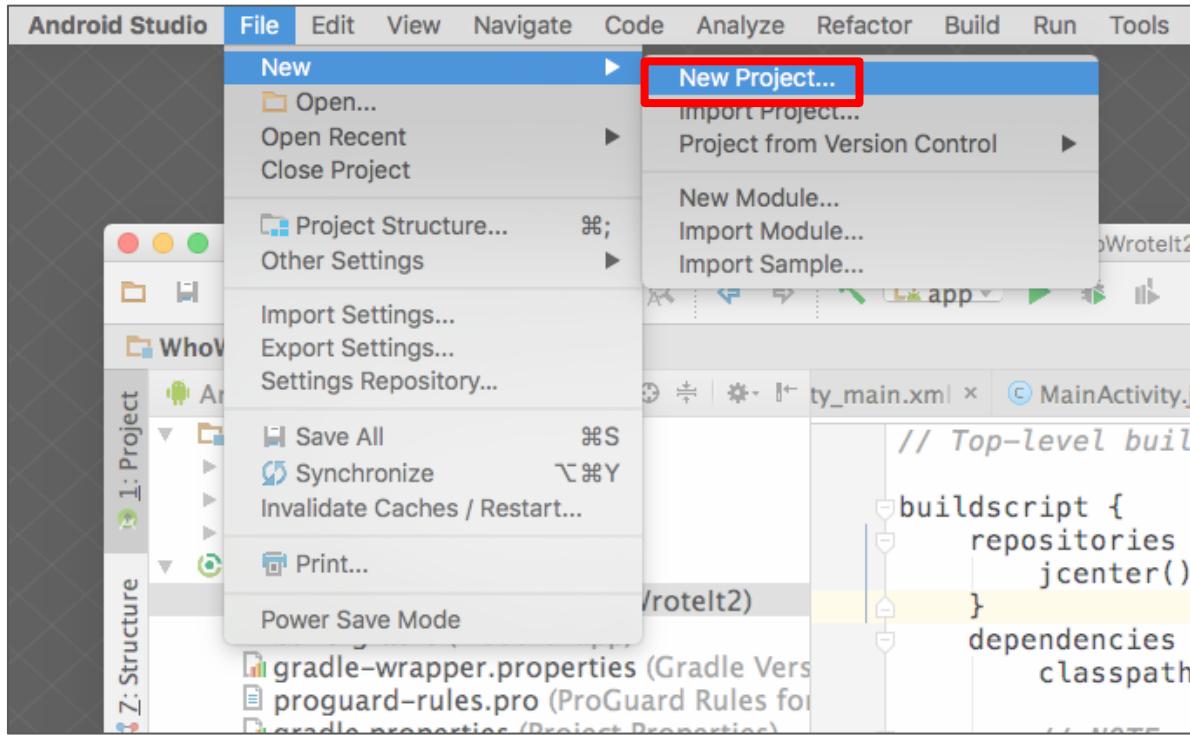
# Creating your first Android app



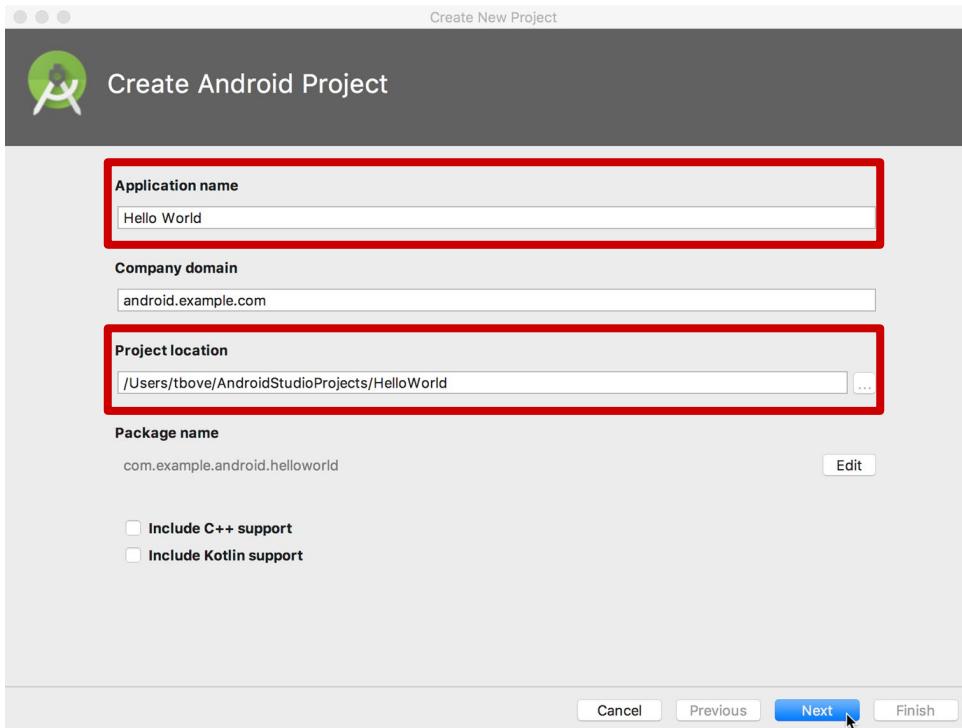
# Start Android Studio



# Create a project inside Android Studio



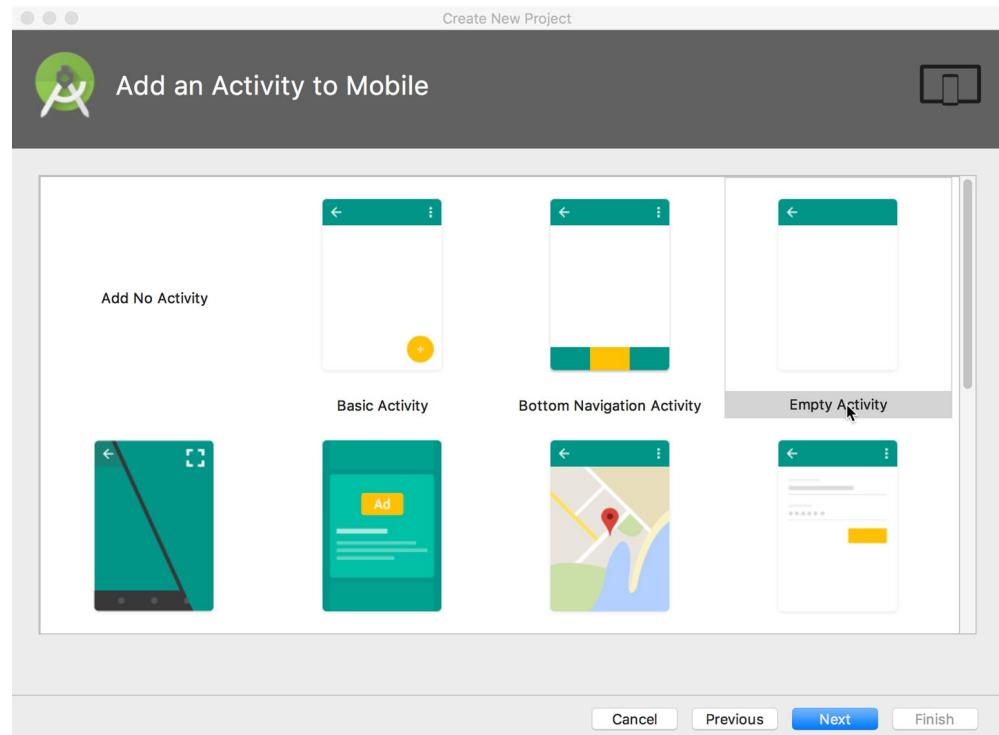
# Name your app



# Pick activity template

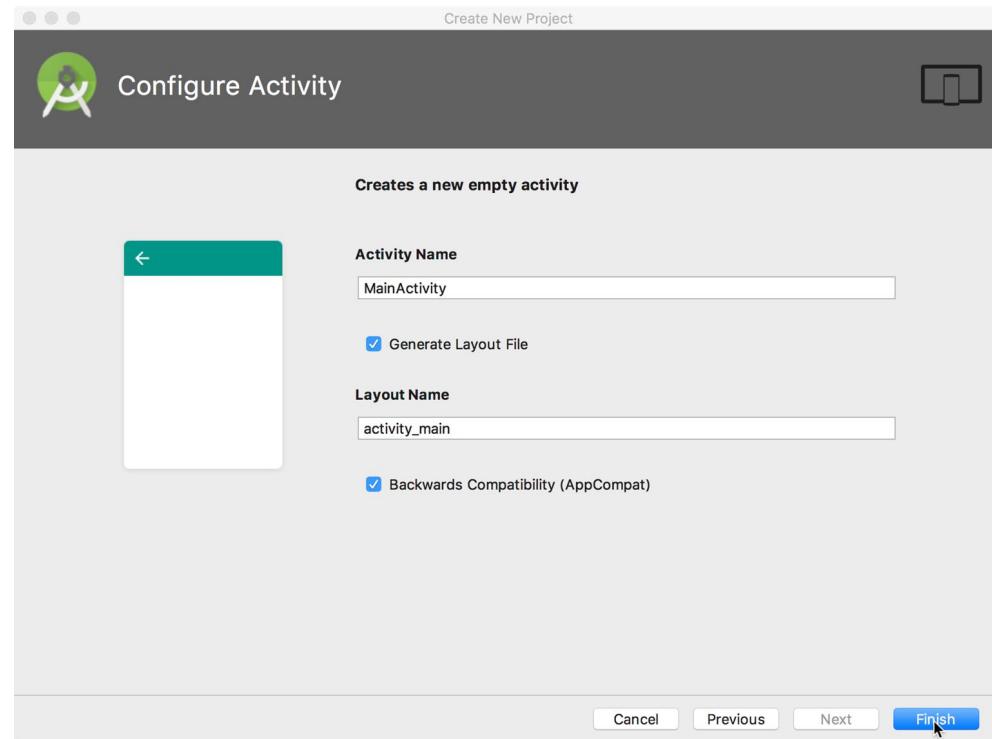
Choose templates for common activities, such as maps or navigation drawers.

Pick Empty Activity or Basic Activity for simple and custom activities.



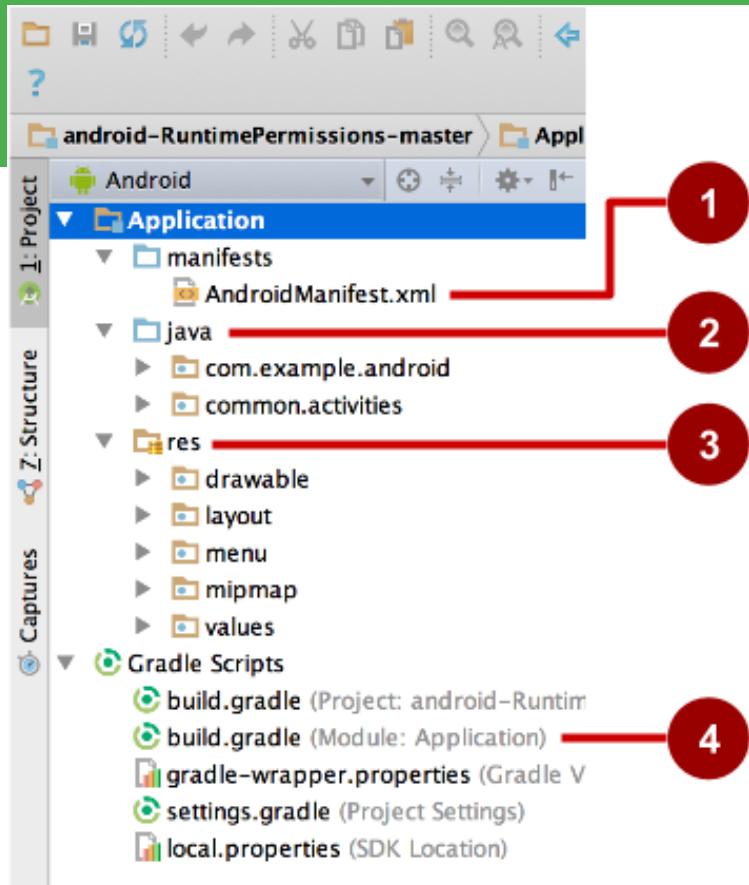
# Name your activity

- Good practice:
  - Name main activity  
`MainActivity`
  - Name layout  
`activity_main`
- Use AppCompat
- Generating layout file is convenient



# Project folders

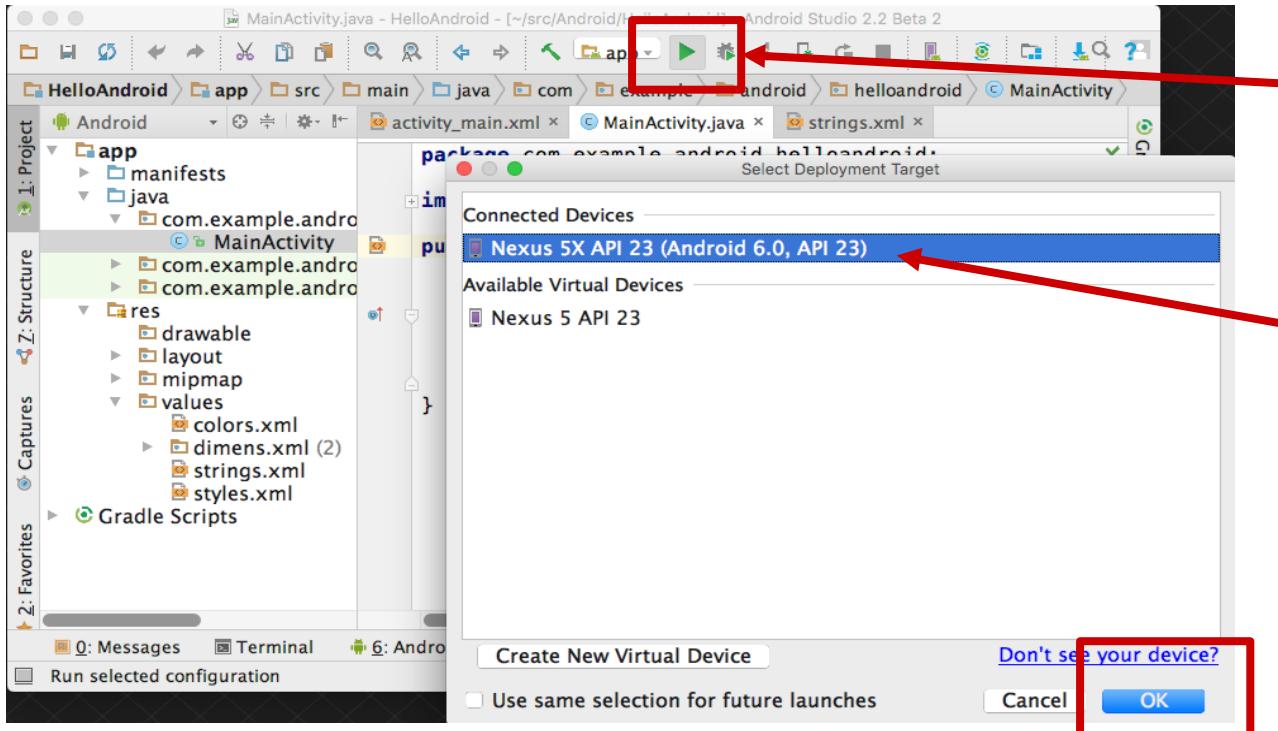
- 1. manifests**—Android Manifest file - description of app read by the Android runtime
- 2. java**—Java source code packages
- 3. res**—Resources (XML) - layout, strings, images, dimensions, colors...
- 4. build.gradle**—Gradle build files



# Gradle build system

- Modern build subsystem in Android Studio
- Three build.gradle:
  - project
  - module
  - settings
- Typically not necessary to know low-level Gradle details
- Learn more about gradle at <https://gradle.org/>

# Run your app



1. Run

2. Select virtual  
or physical  
device

3. OK

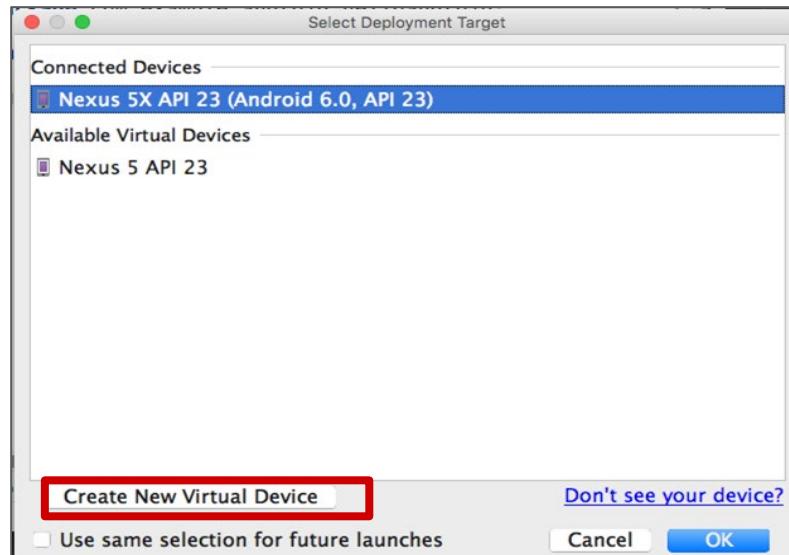
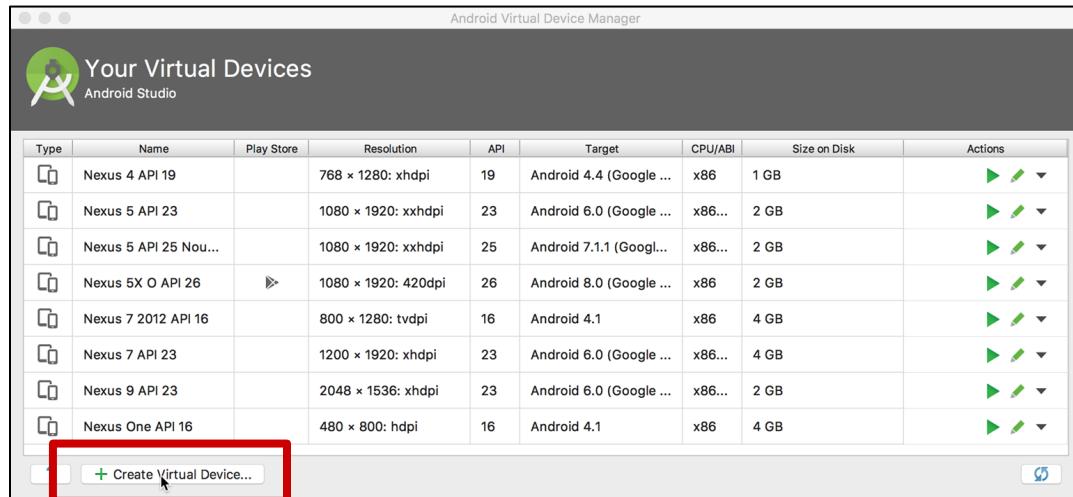


# Create a virtual device

Use emulators to test app on different versions of Android and form factors.

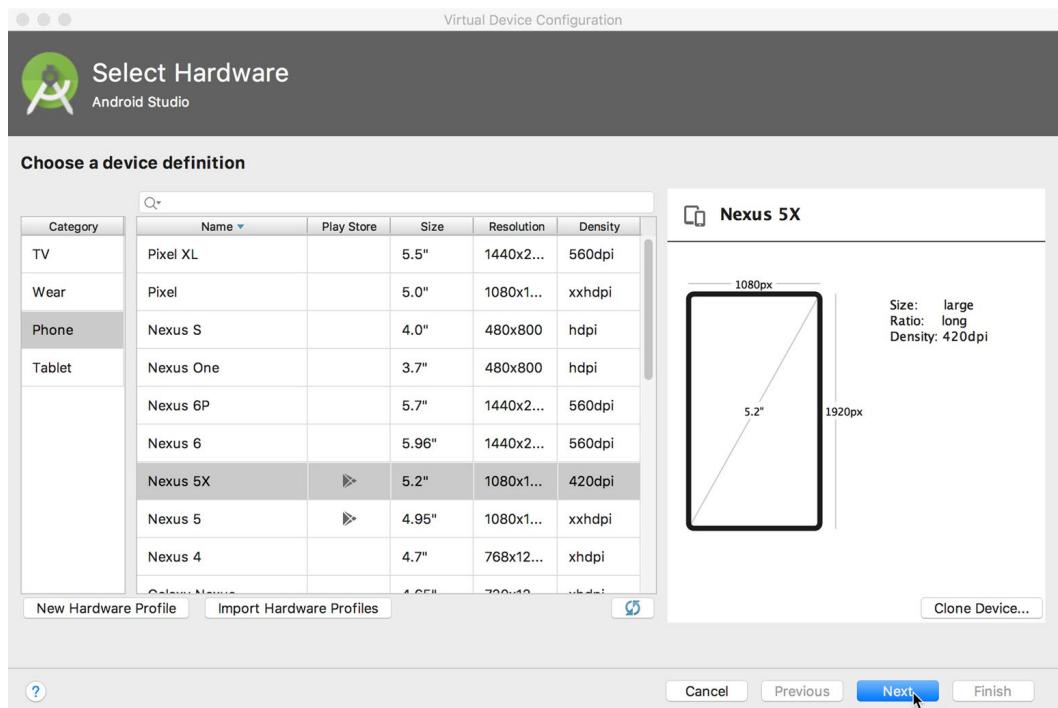
Tools > Android > AVD Manager

or:

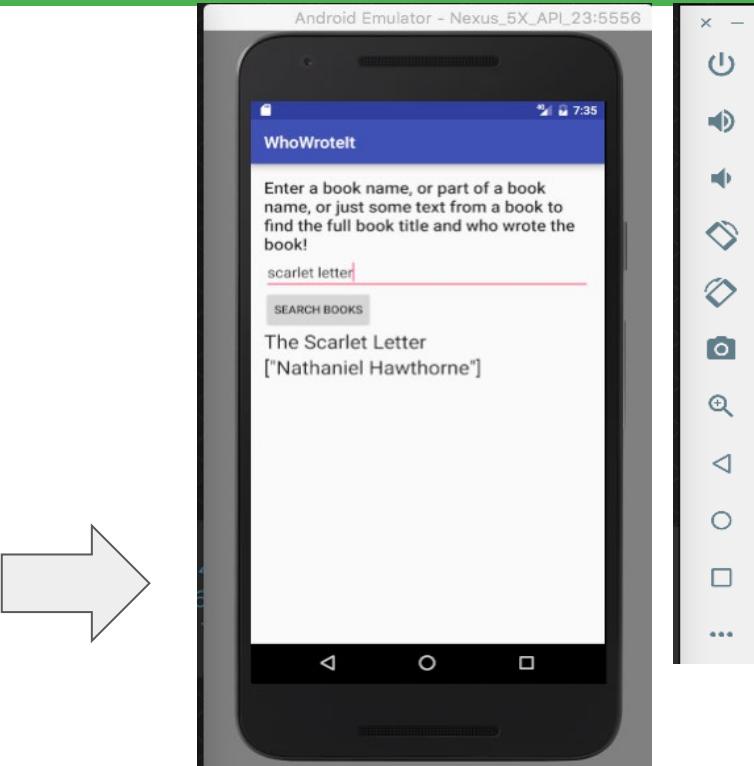


# Configure virtual device

1. Choose hardware
2. Select Android version
3. Finalize



# Run on a virtual device



# Run on a physical device

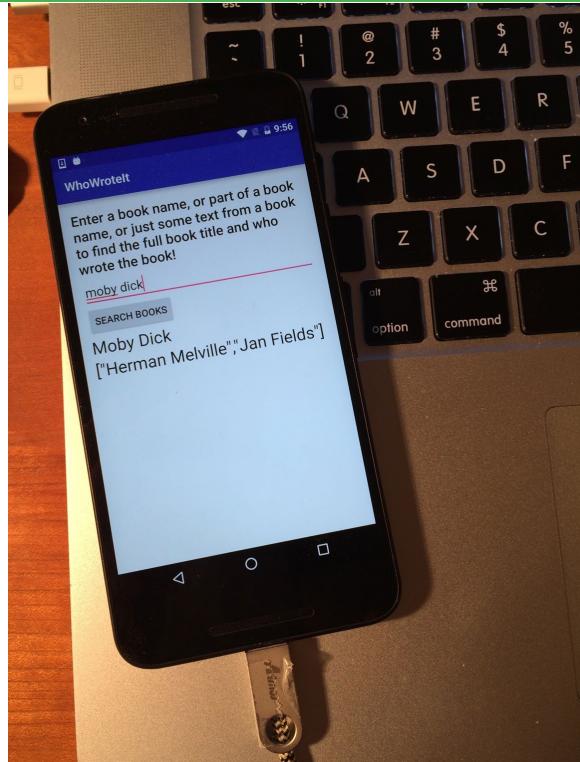
1. Turn on Developer Options:
  - a. **Settings > About phone**
  - b. Tap **Build number** seven times
2. Turn on USB Debugging
  - a. **Settings > Developer Options > USB Debugging**
3. Connect phone to computer with cable

Windows/Linux additional setup:

- [Using Hardware Devices](#)

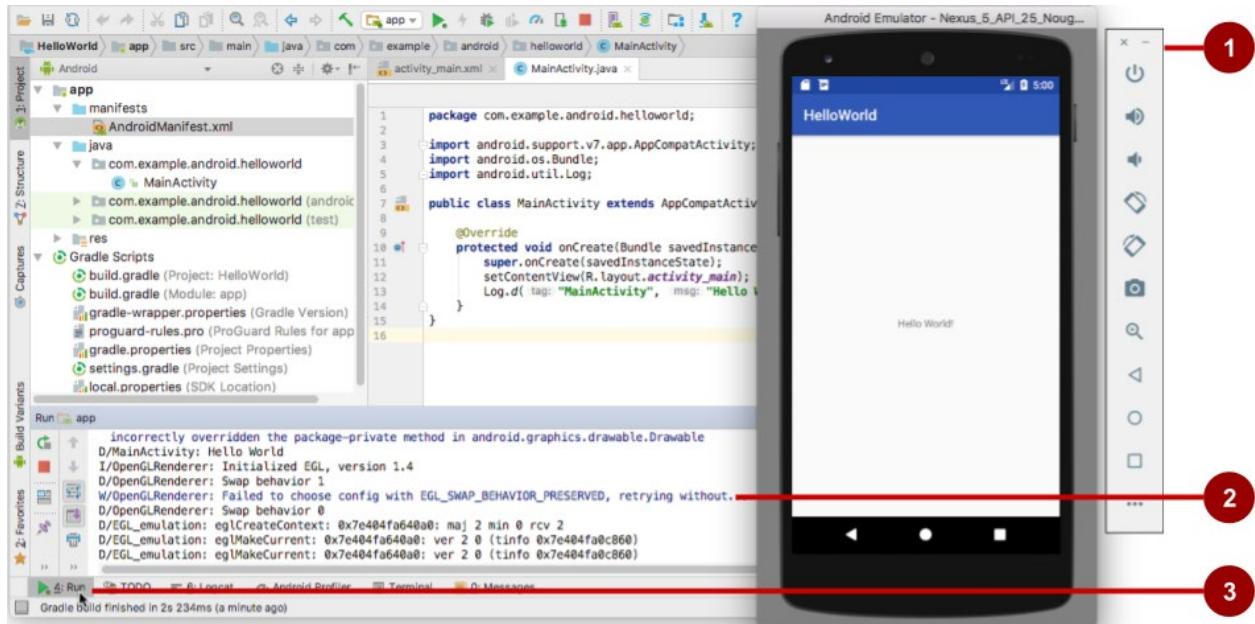
Windows drivers:

- [OEM USB Drivers](#)



# Get feedback as your app runs

1. Emulator running the app
2. Run pane
3. Run tab to open or close the Run pane



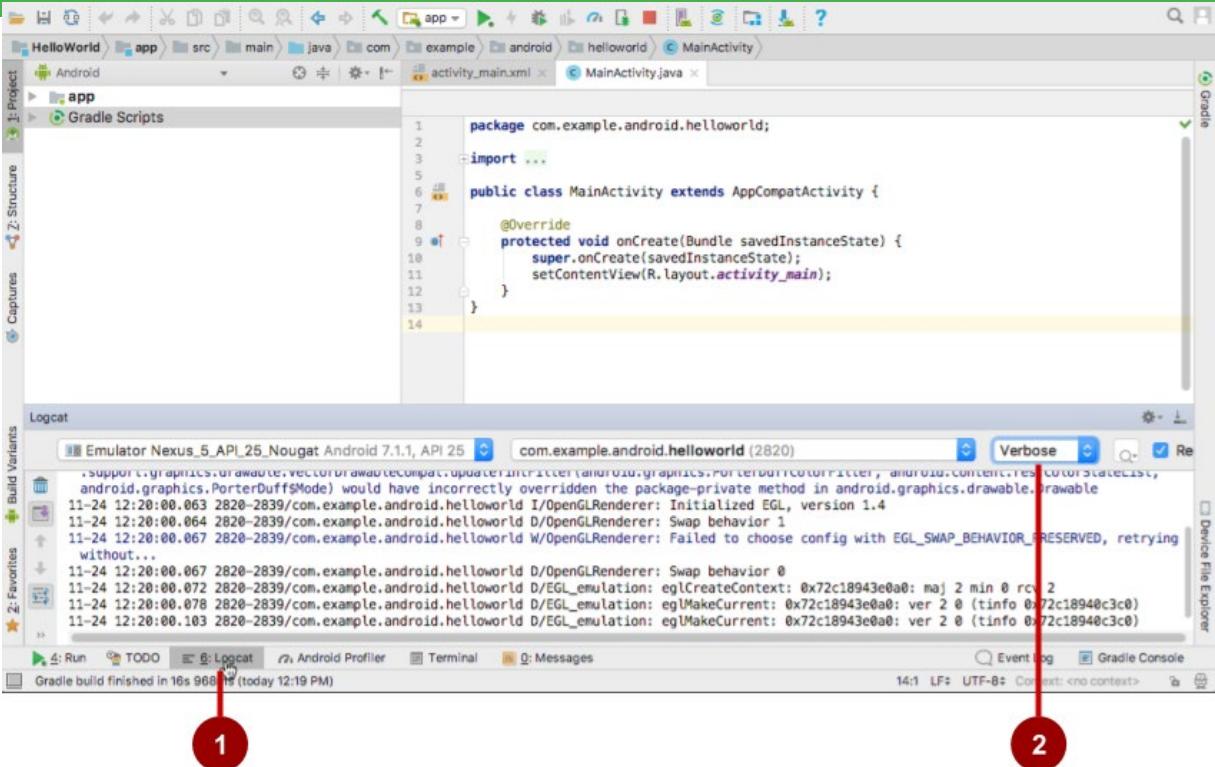
# Adding logging to your app

- As the app runs, the **Logcat** pane shows information
- Add logging statements to your app that will show up in the Logcat pane
- Set filters in **Logcat** pane to see what's important to you
- Search using tags



# The Logcat pane

1. Logcat tab to show Logcat pane
2. Log level menu



# Logging statement

```
import android.util.Log;  
  
// Use class name as tag  
private static final String TAG =  
    MainActivity.class.getSimpleName();  
  
// Show message in Android Monitor, logcat pane  
// Log.<log-level>(TAG, "Message");  
Log.d(TAG, "Creating the URI...");
```

# Learn more

- [Meet Android Studio](#)
- Official Android documentation at [developer.android.com](https://developer.android.com)
- [Create and Manage Virtual Devices](#)
- [Supporting Different Platform Versions](#)
- [Supporting Multiple Screens](#)

# Learn even more

- [Gradle Wikipedia page](#)
- [Google Java Programming Language style guide](#)
- Find answers at [Stackoverflow.com](#)

# What's Next?

- Concept Chapter: [1.1 Your first Android app](#)
- Practical: [1.1 Android Studio and Hello World](#)



# END

# Build your first app

## Lesson 1



Layouts and  
resources for the  
UI

*This work is licensed under a [Creative Commons Attribution 4.0 International License](#).*



# 1.2 Layouts and resources for the UI



# Contents

- Views, view groups, and view hierarchy
- The layout editor and ConstraintLayout
- Event handling
- Resources and measurements

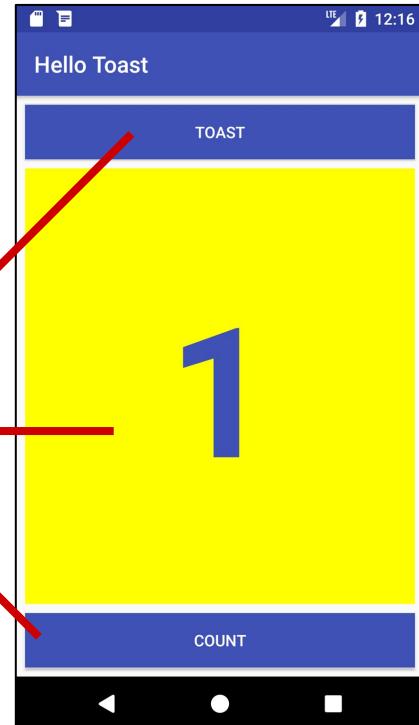


# Views

# Everything you see is a view

If you look at your mobile device,  
every user interface element that you  
see is a **View**.

Views



# What is a view?

View subclasses are basic user interface building blocks

- Display text ([TextView](#) class), edit text ([EditText](#) class)
- Buttons ([Button](#) class), [menus](#), other controls
- Scrollable ([ScrollView](#), [RecyclerView](#))
- Show images ([ImageView](#))
- Group views ([ConstraintLayout](#) and [LinearLayout](#))

# Examples of view subclasses

Button



EditText



Slider



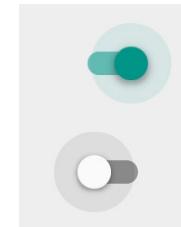
CheckBox



RadioButton



Switch



# View attributes

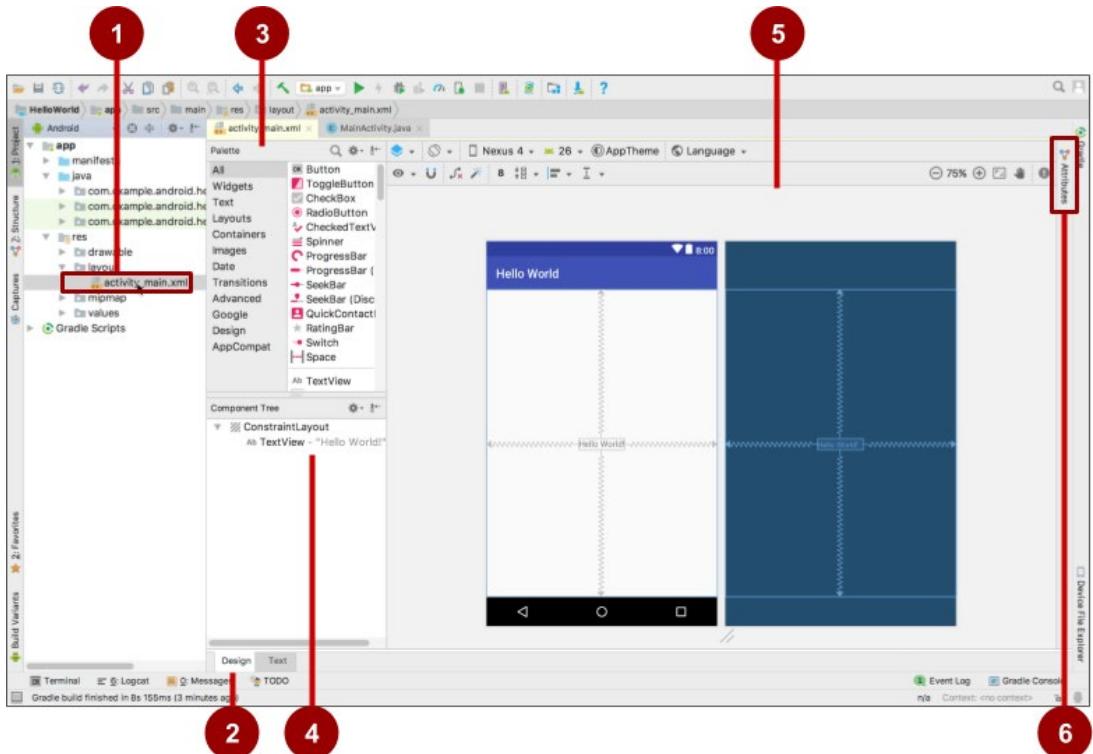
- Color, dimensions, positioning
- May have focus (e.g., selected to receive user input)
- May be interactive (respond to user clicks)
- May be visible or not
- Relationships to other views

# Create views and layouts

- Android Studio layout editor: visual representation of XML
- XML editor
- Java code



# Android Studio layout editor



1. XML layout file
2. Design and Text tabs
3. Palette pane
4. Component Tree
5. Design and blueprint panes
6. Attributes tab

Layouts and  
resources for the  
UI

This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# View defined in XML

```
<TextView
```

```
    android:id="@+id/show_count"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:background="@color/myBackgroundColor"  
    android:text="@string/count_initial_value"  
    android:textColor="@color/colorPrimary"  
    android:textSize="@dimen/count_text_size"  
    android:textStyle="bold"
```

/>

# View attributes in XML

**android:<property\_name>=<property\_value>"**

**Example:** android:layout\_width="match\_parent"

**android:<property\_name>="@<resource\_type>/resource\_id"**

**Example:** android:text="@string/button\_label\_next"

**android:<property\_name>="@+id/view\_id"**

**Example:** android:id="@+id/show\_count"



# Create View in Java code

In an Activity:

*context*



```
TextView myText = new TextView(this);  
myText.setText("Display this text!");
```



# What is the context?

- Context is an interface to global information about an application environment

- Get the context:

```
Context context = getApplicationContext();
```

- An Activity is its own context:

```
TextView myText = new TextView(this);
```



# Custom views

- Over 100 (!) different types of views available from the Android system, all children of the [View](#) class
- If necessary, [create custom views](#) by subclassing existing views or the View class

# ViewGroup and View hierarchy

# ViewGroup contains "child" views

- ConstraintLayout: Positions UI elements using constraint connections to other elements and to the layout edges
- ScrollView: Contains one element and enables scrolling
- RecyclerView: Contains a list of elements and enables scrolling by adding and removing elements dynamically

# ViewGroups for layouts

## Layouts

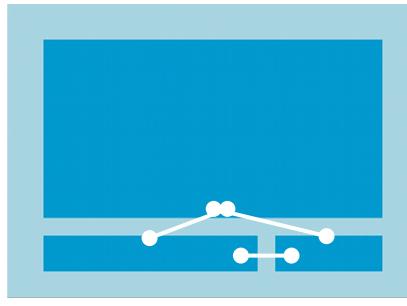
- are specific types of ViewGroups (subclasses of [ViewGroup](#))
- contain child views
- can be in a row, column, grid, table, absolute



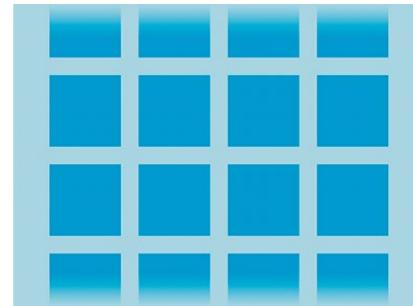
# Common Layout Classes



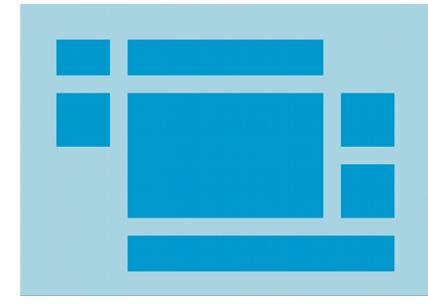
LinearLayout



ConstraintLayout



GridLayout



TableLayout

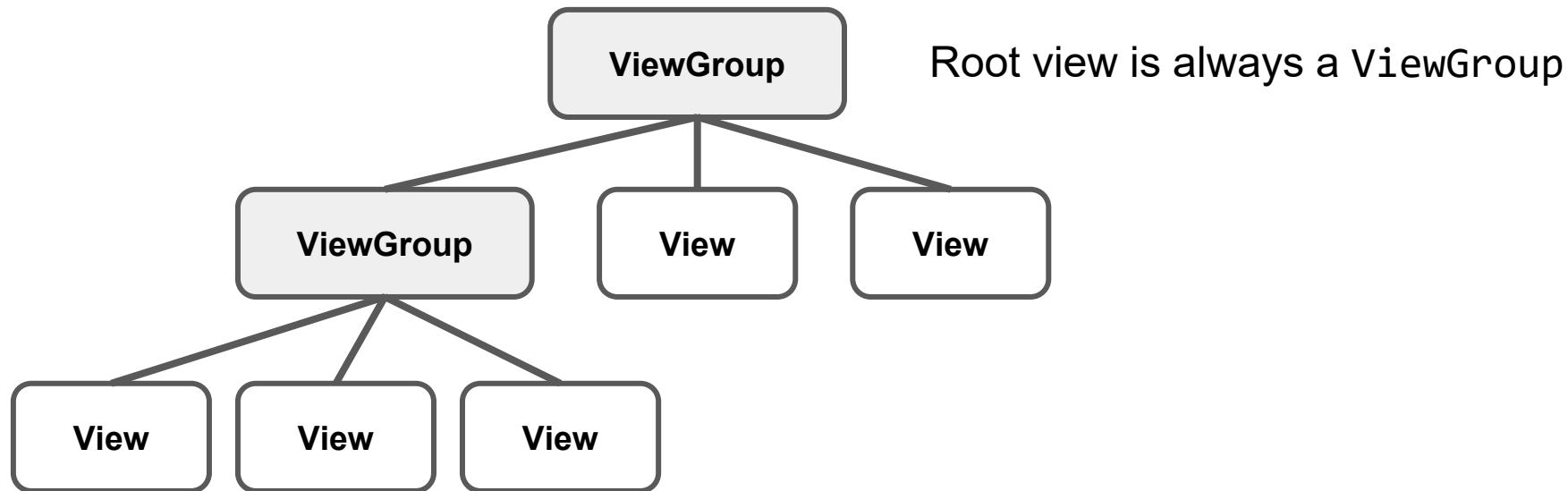
# Common Layout Classes

- ConstraintLayout: Connect views with constraints
- LinearLayout: Horizontal or vertical row
- RelativeLayout: Child views relative to each other
- TableLayout: Rows and columns
- FrameLayout: Shows one child of a stack of children

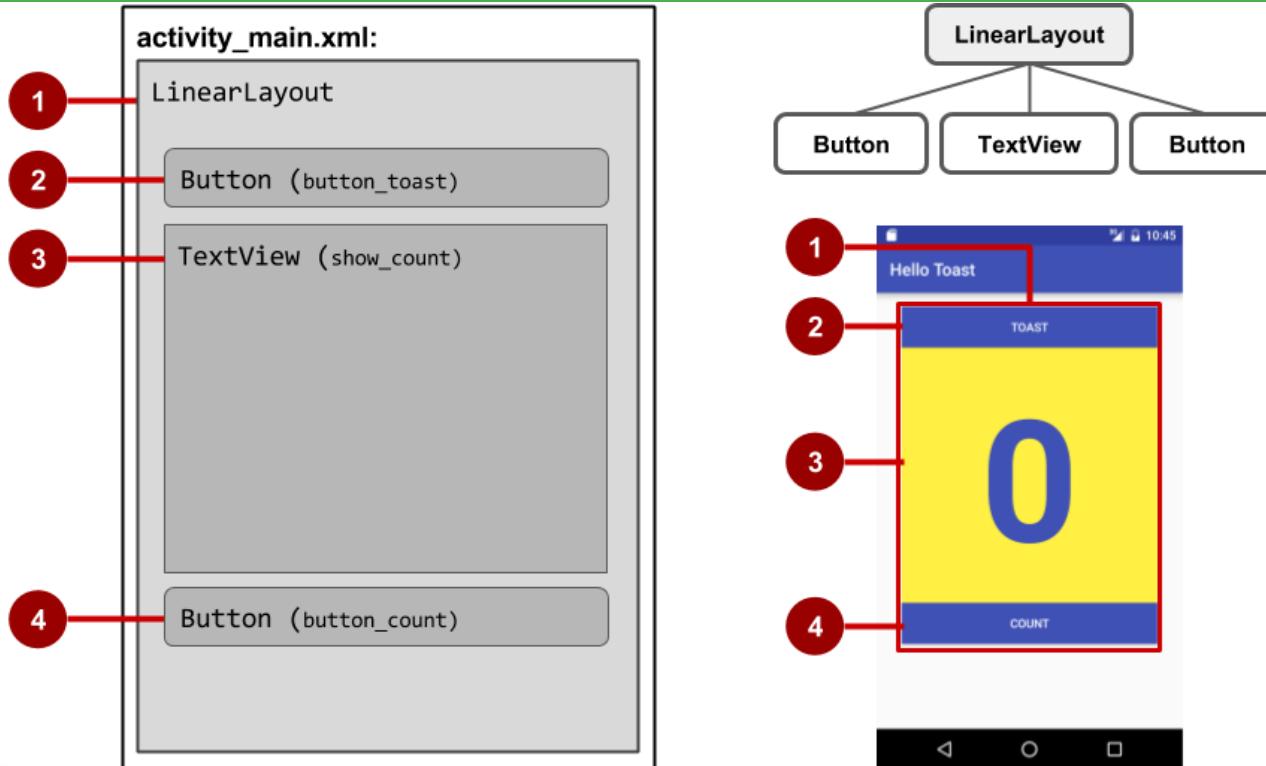
# Class hierarchy vs. layout hierarchy

- View class-hierarchy is standard object-oriented class inheritance
  - For example, Button is-a TextView is-a View is-an Object
  - Superclass-subclass relationship
- Layout hierarchy is how views are visually arranged
  - For example, LinearLayout can contain Buttons arranged in a row
  - Parent-child relationship

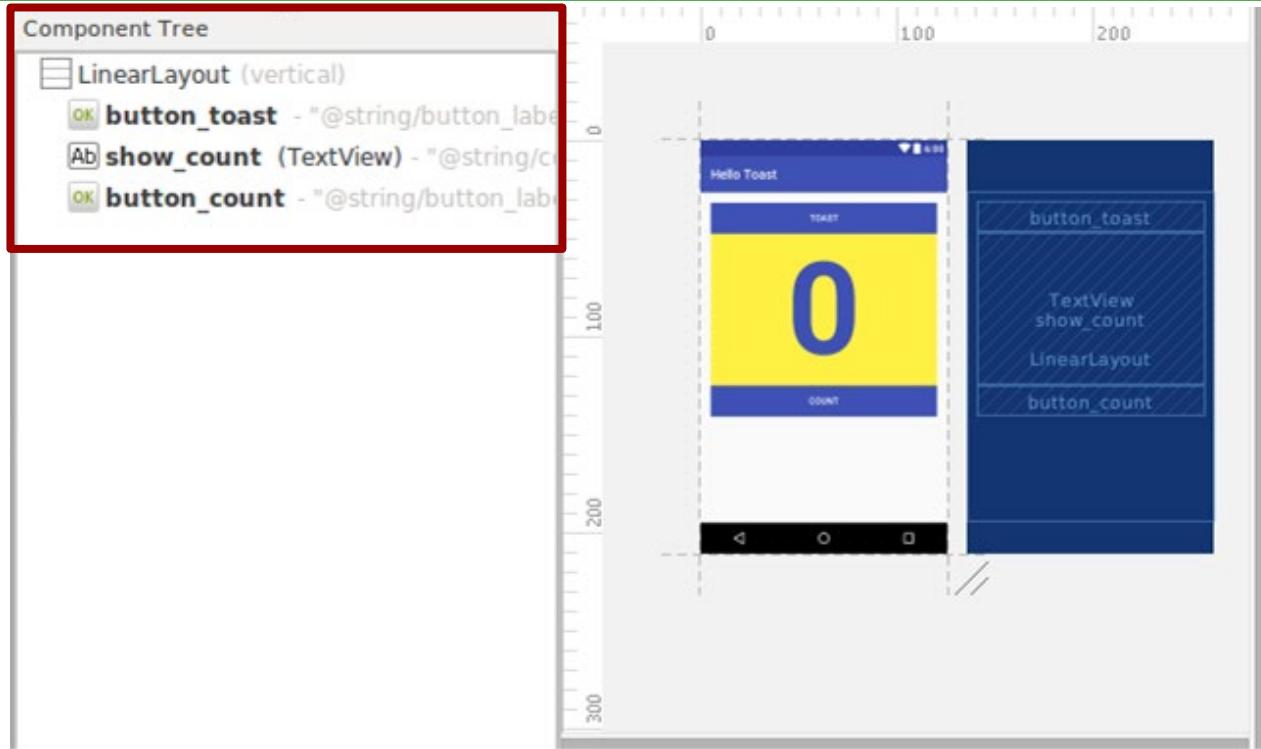
# Hierarchy of viewgroups and views



# View hierarchy and screen layout



# View hierarchy in the layout editor



# Layout created in XML

```
<LinearLayout  
    android:orientation="vertical"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
    <Button  
        ... />  
    <TextView  
        ... />  
    <Button  
        ... />  
</LinearLayout>
```



# Layout created in Java Activity code

```
LinearLayout linearL = new LinearLayout(this);
linearL.setOrientation(LinearLayout.VERTICAL);

TextView myText = new TextView(this);
myText.setText("Display this text!");

linearL.addView(myText);
setContentView(linearL);
```



# Set width and height in Java code

Set the width and height of a view:

```
LinearLayout.LayoutParams layoutParams =  
    new LinearLayout.LayoutParams(  
        LayoutParams.MATCH_PARENT,  
        LayoutParams.MATCH_CONTENT);  
  
myView.setLayoutParams(layoutParams);
```

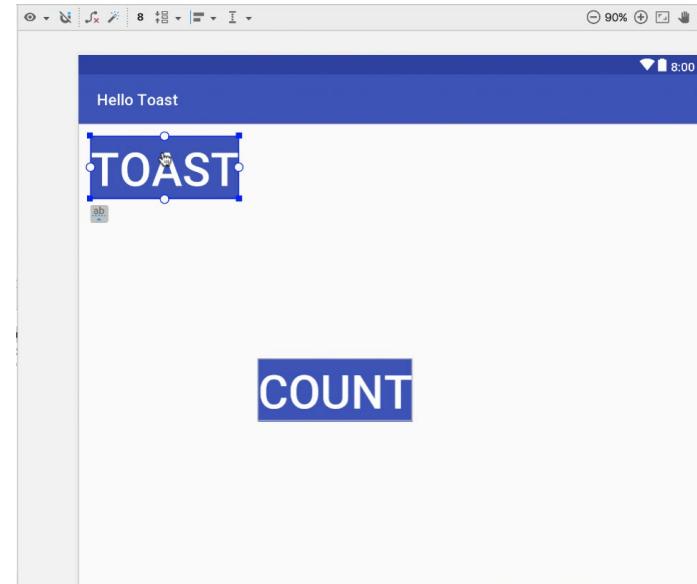
# Best practices for view hierarchies

- Arrangement of view hierarchy affects app performance
- Use smallest number of simplest views possible
- Keep the hierarchy flat—limit nesting of views and view groups

# The layout editor and Constraint Layout

# The layout editor with ConstraintLayout

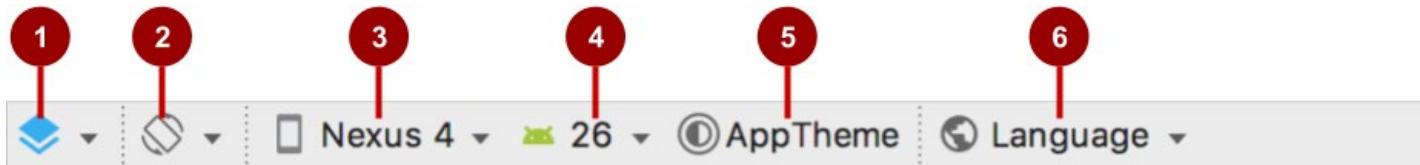
- Connect UI elements to parent layout
- Resize and position elements
- Align elements to others
- Adjust margins and dimensions
- Change attributes



# What is ConstraintLayout?

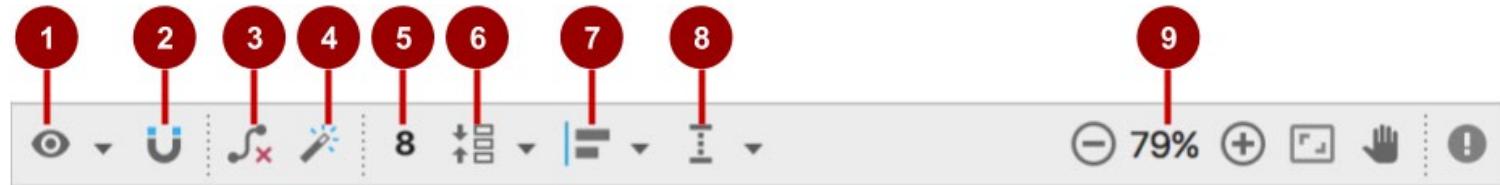
- Default layout for new Android Studio project
- ViewGroup that offers flexibility for layout design
- Provides constraints to determine positions and alignment of UI elements
- Constraint is a connection to another view, parent layout, or invisible guideline

# Layout editor main toolbar



1. Select Design Surface: Design and Blueprint panes
2. Orientation in Editor: Portrait and Landscape
3. Device in Editor: Choose device for preview
4. API Version in Editor: Choose API for preview
5. Theme in Editor: Choose theme for preview
6. Locale in Editor: Choose language/locale for preview

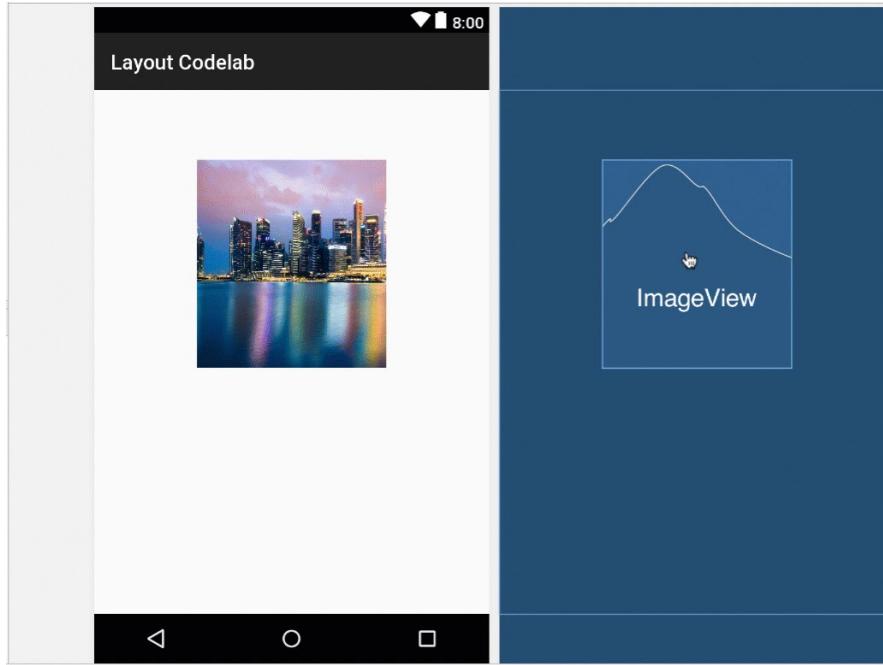
# ConstraintLayout toolbar in layout editor



1. Show: Show Constraints and Show Margins
2. Autoconnect: Enable or disable
3. Clear All Constraints: Clear all constraints in layout
4. Infer Constraints: Create constraints by inference
5. Default Margins: Set default margins
6. Pack: Pack or expand selected elements
7. Align: Align selected elements
8. Guidelines: Add vertical or horizontal guidelines
9. Zoom controls: Zoom in or out

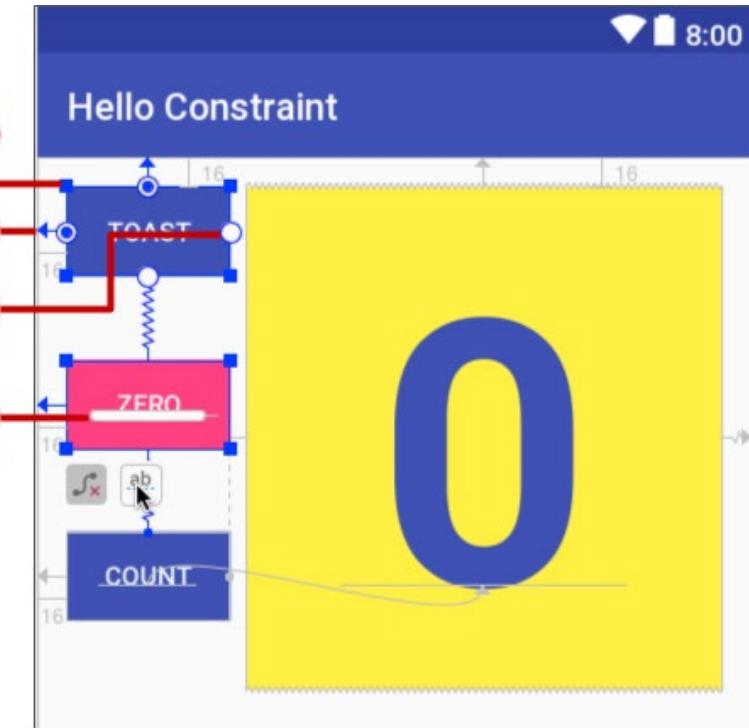
# Autoconnect

- Enable Autoconnect  in toolbar if disabled
- Drag element to any part of a layout
- Autoconnect generates constraints against parent layout



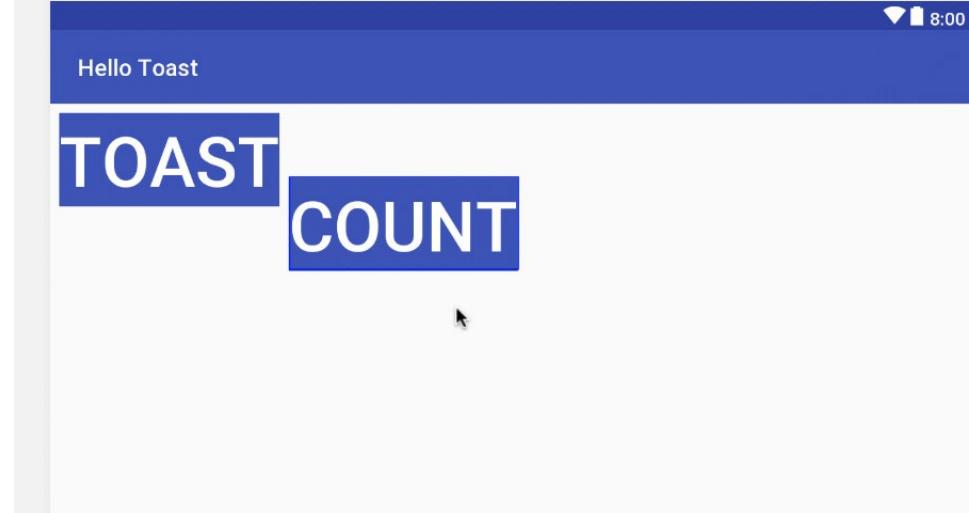
# ConstraintLayout handles

1. Resizing handle
2. Constraint line and handle
3. Constraint handle
4. Baseline handle



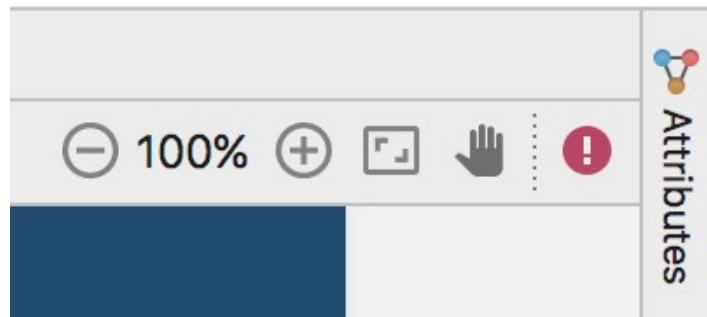
# Align elements by baseline

1. Click the  baseline constraint button
2. Drag from baseline to other element's baseline



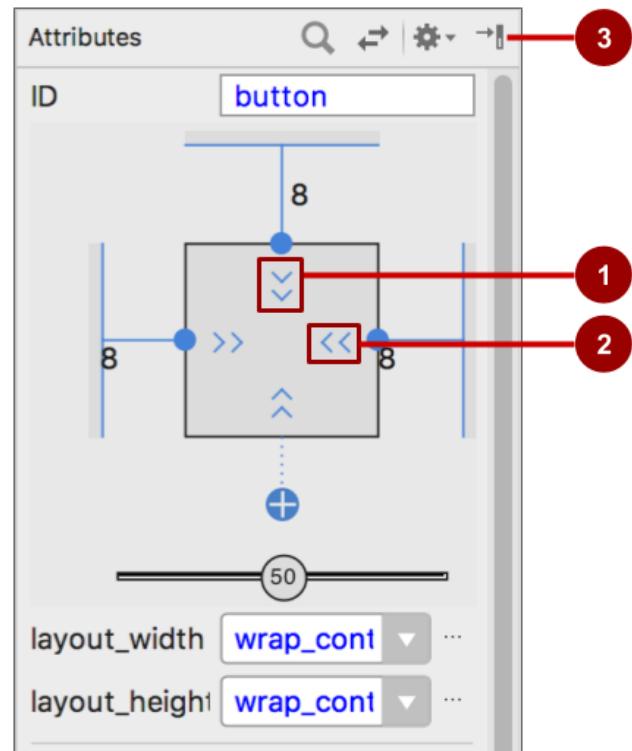
# Attributes pane

- Click the Attributes tab
- Attributes pane includes:
  - Margin controls for positioning
  - Attributes such as `layout_width`



# Attributes pane view inspector

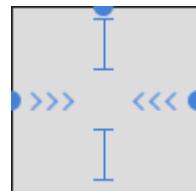
1. Vertical view size control specifies `layout_height`
2. Horizontal view size control specifies `layout_width`
3. Attributes pane close button



# Layout\_width and layout\_height

layout\_width and layout\_height change with size controls

- `match_constraint`: Expands element to fill its parent
- `wrap_content`: Shrinks element to enclose content
- `fixed`: Fixed number of dp (density-independent pixels)

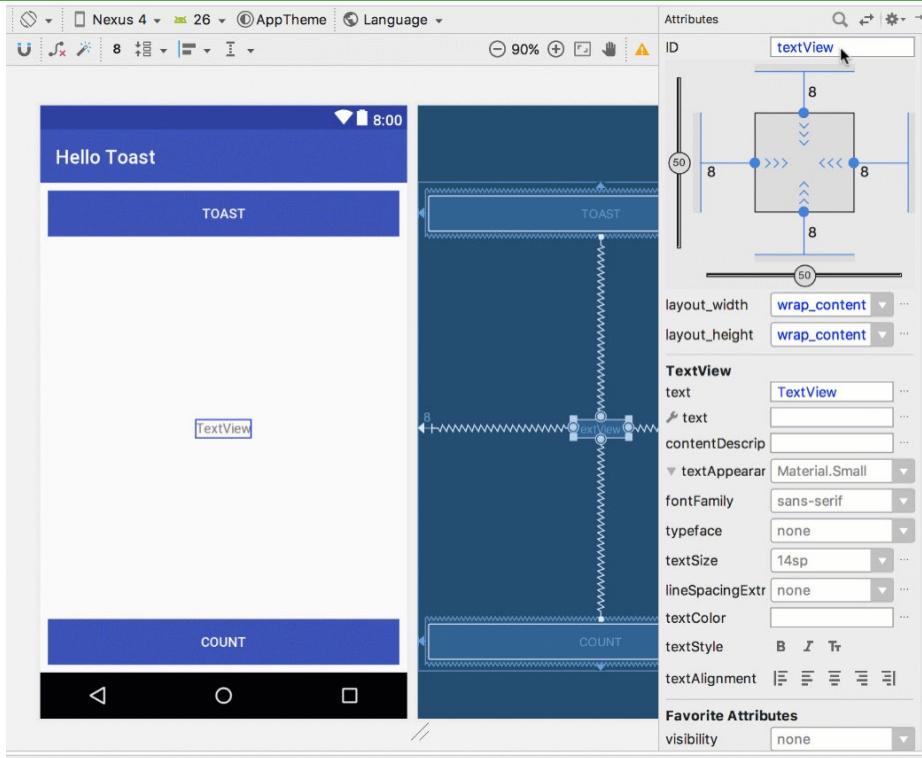


# Set attributes

To view and edit all attributes for element:

1. Click **Attributes** tab
2. Select element in design, blueprint, or Component Tree
3. Change most-used attributes
4. Click  at top or **View more attributes** at bottom to see and change more attributes

# Set attributes example: TextView



# Preview layouts

Preview layout with horizontal/vertical orientation:

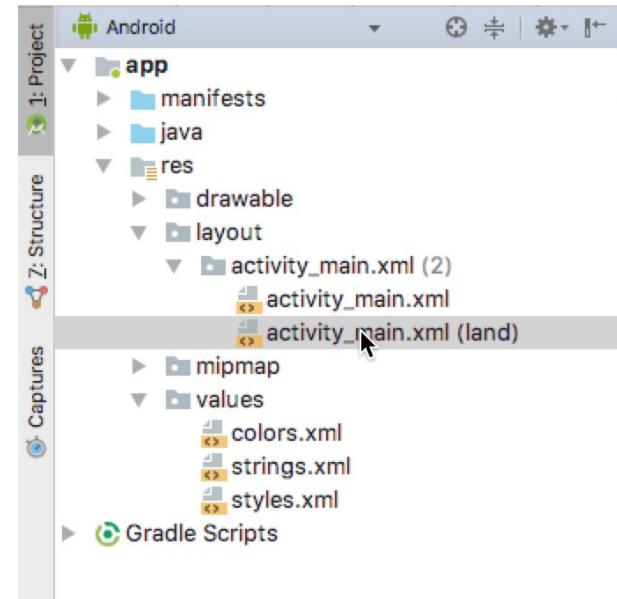
1. Click Orientation in Editor button 
2. Choose **Switch to Landscape** or **Switch to Portrait**

Preview layout with different devices:

1. Click Device in Editor button  Nexus 5 
2. Choose device

# Create layout variant for landscape

1. Click Orientation in Editor button
2. Choose Create Landscape Variation
3. Layout variant created:  
`activity_main.xml (land)`
4. Edit the layout variant as needed



# Create layout variant for tablet

1. Click Orientation in Layout Editor 
2. Choose **Create layout x-large Variation**
3. Layout variant created: `activity_main.xml (xlarge)`
4. Edit the layout variant as needed

# Event Handling

# Events

Something that happens

- In UI: Click, tap, drag
- Device: DetectedActivity such as walking, driving, tilting
- Events are "noticed" by the Android system

# Event Handlers

Methods that do something in response to a click

- A method, called an **event handler**, is triggered by a specific event and does something in response to the event

# Attach in XML and implement in Java

## Attach handler to view in XML layout:

```
android:onClick="showToast"
```

## Implement handler in Java activity:

```
public void showToast(View view) {  
    String msg = "Hello Toast!";  
    Toast toast = Toast.makeText(  
        this, msg, duration);  
    toast.show();  
}
```

# Alternative: Set click handler in Java

```
final Button button = (Button) findViewById(R.id.button_id);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        String msg = "Hello Toast!";
        Toast toast = Toast.makeText(this, msg, duration);
        toast.show();
    }
});
```

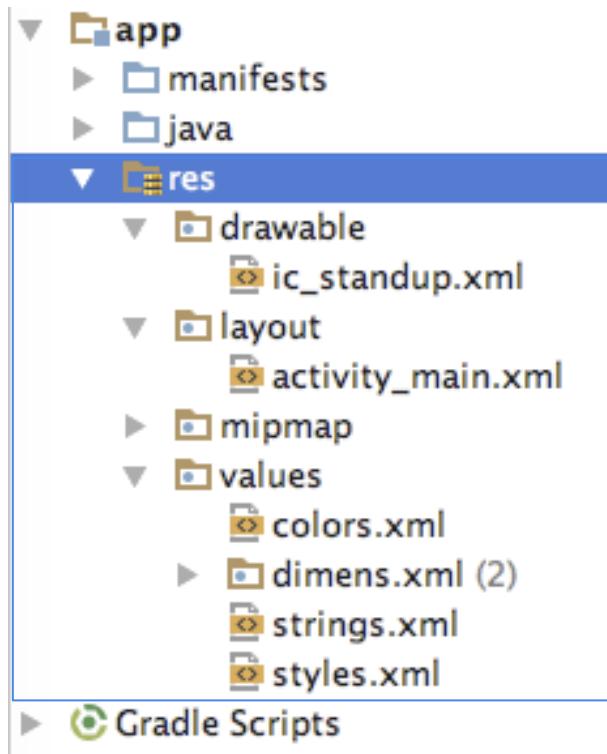
# Resources and measurements

# Resources

- Separate static data from code in your layouts.
- Strings, dimensions, images, menu text, colors, styles
- Useful for localization



# Where are the resources in your project?



resources and resource files  
stored in **res** folder

# Refer to resources in code

- Layout:

```
R.layout.activity_main  
setContentView(R.layout.activity_main);
```

- View:

```
R.id.recyclerview  
rv = (RecyclerView) findViewById(R.id.recyclerview);
```

- String:

In Java: R.string.title

In XML: android:text="@string/title"



# Measurements

- Density-independent Pixels (dp): for Views
- Scale-independent Pixels (sp): for text

Don't use device-dependent or density-dependent units:

- Actual Pixels (px)
- Actual Measurement (in, mm)
- Points - typography 1/72 inch (pt)

# Learn more

# Learn more

## Views:

- [View class documentation](#)
- [device independent pixels](#)
- [Button class documentation](#)
- [TextView class documentation](#)

## Layouts:

- [developer.android.com Layouts](#)
- [Common Layout Objects](#)

# Learn even more

## Resources:

- [Android resources](#)
- [Color class definition](#)
- [R.color resources](#)
- [Supporting Different Densities](#)
- [Color Hex Color Codes](#)

## Other:

- [Android Studio documentation](#)
- [Image Asset Studio](#)
- [UI Overview](#)
- [Vocabulary words and concepts glossary](#)
- [Model-View-Presenter](#)  
(MVP) architecture pattern
- [Architectural patterns](#)

# What's Next?

- Concept Chapter: [1.2 Layouts and resources for the UI](#)
- Practicals:
  - [1.2A : Your first interactive UI](#)
  - [1.2B : The layout editor](#)

# END

Android Developer Fundamentals V2

# Build your first app

Lesson 1



Text and  
Scrolling Views

This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# 1.3 Text and scrolling views



# Contents

- TextView
- ScrollView



# TextView

# TextView for text

- TextView is View subclass for single and multi-line text
- EditText is TextView subclass with editable text
- Controlled with layout attributes
- Set text:
  - Statically from string resource in XML
  - Dynamically from Java code and any source

# Formatting text in string resource

- Use `<b>` and `<i>` HTML tags for bold and italics
- All other HTML tags are ignored
- String resources: one unbroken line = one paragraph
- `\n` starts a new a line or paragraph
- Escape apostrophes and quotes with backslash (`\'', \'`)
- Escape any non-ASCII characters with backslash (`\`)

# Creating TextView in XML

```
<TextView android:id="@+id/textview"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="@string/my_story"/>
```



# Common TextView attributes

android:text—text to display

android:textColor—color of text

android:textAppearance—predefined style or theme

android:textSize—text size in sp

android:textStyle—normal, bold, italic, or bold|italic

android:typeface—normal, sans, serif, or monospace

android:lineSpacingExtra—extra space between lines in sp

# Formatting active web links

```
<string name="article_text">... www.rockument.com ...</string>
```

```
<TextView  
    android:id="@+id/article"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:autoLink="web"  
    android:text="@string/article_text"/>
```

autoLink values:"web", "email", "phone", "map", "all"

Don't use HTML  
for a web link in  
free-form text



# Creating TextView in Java code

```
TextView myTextview = new TextView(this);
myTextview.setWidthLayoutParams.MATCH_PARENT);
myTextview.setHeightLayoutParams.WRAP_CONTENT);
myTextview.setMinLines(3);
myTextview.setText(R.string.my_story);
myTextview.append(userComment);
```

# ScrollView

# What about large amounts of text?

- News stories, articles, etc...
- To scroll a TextView, embed it in a [ScrollView](#)
- Only *one* View element (usually TextView) allowed in a ScrollView
- To scroll multiple elements, use one ViewGroup (such as LinearLayout) within the ScrollView



# ScrollView for scrolling content

- [ScrollView](#) is a subclass of [FrameLayout](#)
- Holds all content in memory
- Not good for long texts, complex layouts
- Do not nest multiple scrolling views
- Use [HorizontalScrollView](#) for horizontal scrolling
- Use a [RecyclerView](#) for lists



# ScrollView layout with one TextView

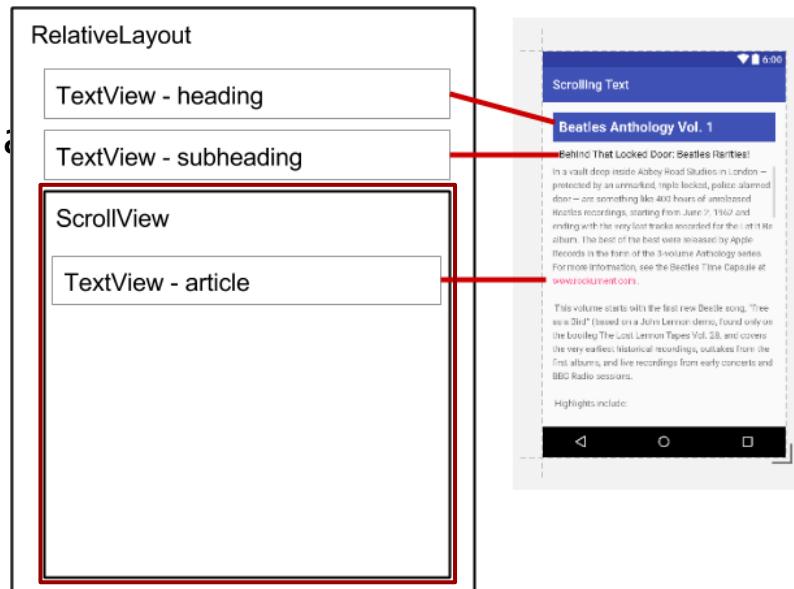
```
<ScrollView
```

```
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_below="@+id/article_subhead
```

```
<TextView
```

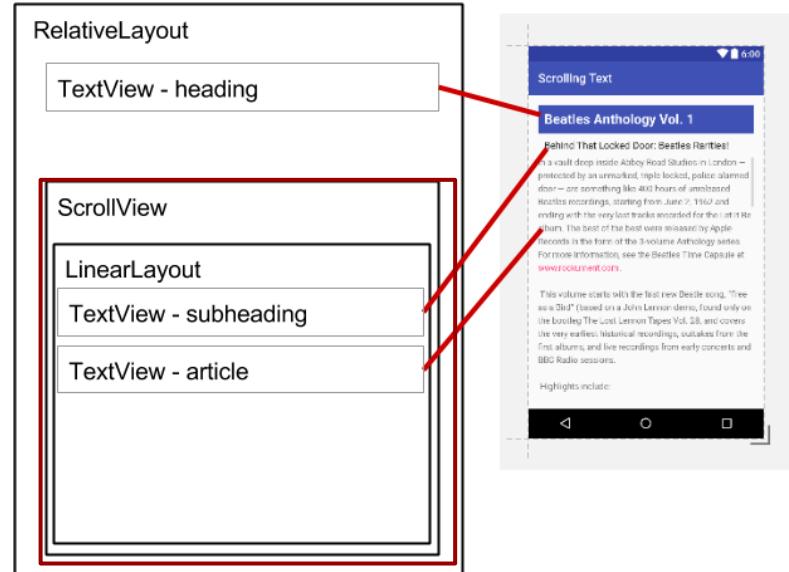
```
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    .../>
```

```
</ScrollView>
```



# ScrollView layout with a view group

```
<ScrollView ...>  
    <LinearLayout  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:orientation="vertical">  
  
        <TextView  
            android:id="@+id/article_subheading"  
            .../>  
  
        <TextView  
            android:id="@+id/article" ... />  
    </LinearLayout>  
</ScrollView>
```



# ScrollView with image and button

```
<ScrollView...>
    <LinearLayout...>
        <ImageView.../>
        <Button.../>
        <TextView.../>
    </LinearLayout>
</ScrollView>
```

One child of ScrollView which can be a layout

Children of the layout

# Learn more

Developer Documentation:

- [TextView](#)
- [ScrollView](#) and [HorizontalScrollView](#)
- [String Resources](#)

Other:

- Android Developers Blog: [Linkify your Text!](#)
- Codepath: [Working with a TextView](#)



# What's Next?

- Concept Chapter: [1.3 Text and scrolling views](#)
- Practical: [1.3 Text and scrolling views](#)



# END

# Build your first app

Lesson 1



Resources to help you learn

*This work is licensed under a Creative Commons Attribution 4.0 International License.*



# 1.4 Resources to help you learn

# Contents

- Documentation
- Tutorials and codelabs
- Blogs and videos
- Udacity courses
- Source code for the practicals



# Official Documentation

[developer.android.com](https://developer.android.com)

The screenshot shows the official Android Developers documentation website. The header features a green bar with the "Developers" logo, navigation links for DESIGN, DEVELOP, and DISTRIBUTE, a search bar, and a "PLAY CONSOLE" button. The main content area has a blue background. On the left, a sidebar lists categories: HOME (Android, Wear, TV, Auto, Chrome OS, Things), DESIGN, DEVELOP, DISTRIBUTE, and STORIES. The central content area is titled "Android Oreo" and includes a sub-headline: "Smarter, faster, and more powerful than ever. The world's favorite cookie is your new favorite Android release." It features a large illustration of the Android robot holding a large Oreo cookie. Below the illustration are two links: "Learn about Android Oreo essentials" and "Get started with Android 8.1". At the bottom of the page are three calls-to-action: "Get Android Studio", "Browse sample code", and "Watch stories".

# Documentation Structure

The screenshot shows the top navigation bar of the Android Developers website. It includes a menu icon, the "Developers" logo, and the words "DESIGN", "DEVELOP", and "DISTRIBUTE". Below this is a search bar with a magnifying glass icon and the word "Search". The main content area has a "HOME" section with three buttons: "DESIGN", "DEVELOP", and "DISTRIBUTE". Red arrows point from the text labels to these buttons. The "DESIGN" button is associated with the text "UX approach using ‘Material Design’". The "DEVELOP" button is associated with the text "Software Developer Information, trainings, tutorials, sample code, reference". The "DISTRIBUTE" button is associated with the text "Delivering apps to users".

Search

DESIGN - UX approach using “Material Design”

DESIGN

DEVELOP

DISTRIBUTE

DEVELOP - Software Developer Information,  
trainings, tutorials, sample code, reference

DISTRIBUTE - Delivering apps to users

# Stackoverflow.com

[stackoverflow.com/questions/tagged/android](https://stackoverflow.com/questions/tagged/android)

Question/Answer format

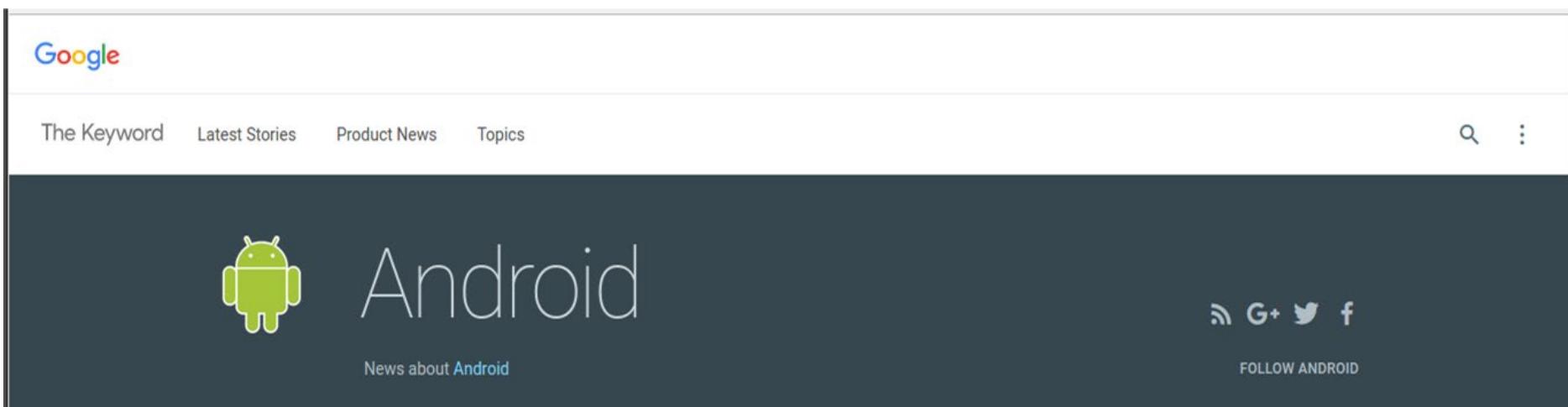
Android, Java language, other programming topics

The screenshot shows a web browser window with the title 'Newest 'android' Questions'. The address bar contains the URL 'stackoverflow.com/questions/tagged/android'. The page itself is the 'Tagged Questions' section for the 'android' tag. It features the Stack Overflow logo at the top left. Navigation tabs include 'Questions', 'Jobs', 'Documentation Beta', 'Tags', and 'Use'. Below these are filters for 'info', 'newest' (which is selected), '48 featured', 'frequent', 'votes', 'active', and 'unanswered'. A large statistic on the right states '879,951 questions tagged android'. At the bottom, there's a snippet of a question about Android development.

# Official Android Blog

[android.googleblog.com](http://android.googleblog.com)

News, features, high-level



The screenshot shows the official Android blog homepage. At the top left is the Google logo. Below it are navigation links: "The Keyword", "Latest Stories", "Product News", and "Topics". On the right side are search and filter icons. The main header features the Android robot icon and the word "Android" in large white letters. Below the header, there's a dark banner with the text "News about Android" and "FOLLOW ANDROID" with social media icons for RSS, Google+, Twitter, and Facebook. The overall design is clean and modern.



# Android Developers Blog

[android-developers.blogspot.com](http://android-developers.blogspot.com)

News, updates, developer stories, and articles on how to make your app successful



Android Developers Blog



Google Developers Training

| Android Developer Fundamentals V2

Resources to help  
you learn

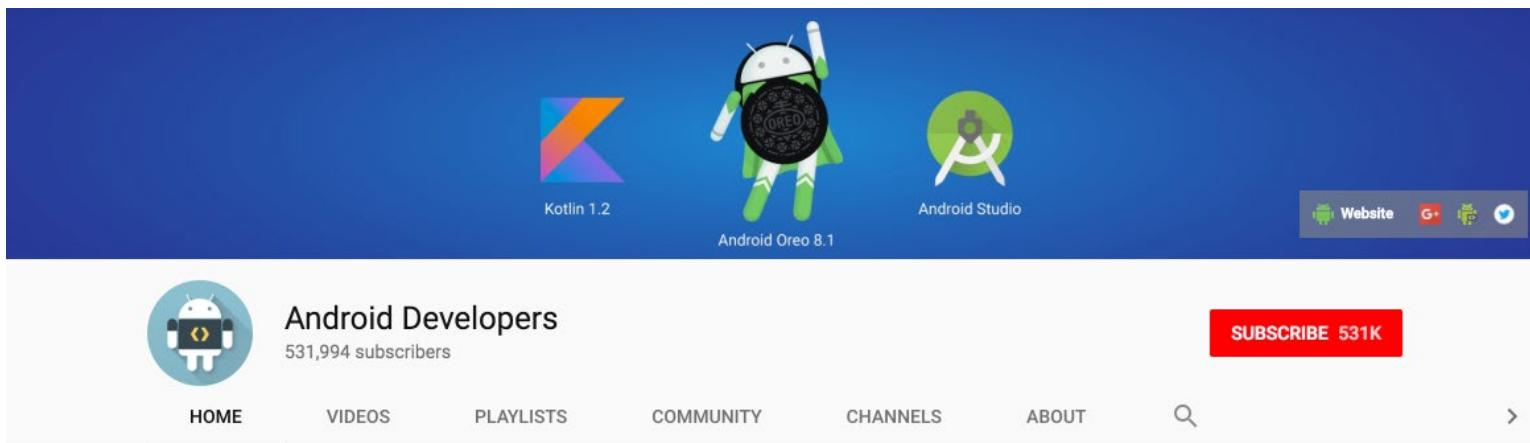
This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# Android Developers YouTube Channel

## Android Developers YouTube channel

News, tools, how to, and playlists around specific themes, such as Android Developer Patterns, Android Developer Stories, and Android Performance Patterns



# Google IO Codelabs

[codelabs.developers.google.com](https://codelabs.developers.google.com)

Short tutorials  
about specific topics

The screenshot shows the Google Developers Codelabs homepage. At the top, there's a navigation bar with the Google Developers logo and a search bar. Below the header, a large "Welcome to Codelabs!" message is displayed. A paragraph of text explains what codelabs are and what topics they cover. On the right side of the header, there's a "VIEW EVENTS" button. The main content area has tabs for "POPULAR", "RECENT", and "DURATION". A dropdown menu shows "Android" is selected. Three codelab cards are listed:

Icon	Name	Duration
	Basic Android Accessibility : making sure everyone can use what you create!	38 min
	Agera: reactive Android apps	73 min
	Echo with Android Howie Library	21 min

Below each card, there's a "START" button and the last update date.

# Online Udacity courses

[www.udacity.com/courses/android](http://www.udacity.com/courses/android)

- Interactive video-based tutorials
- Android courses are built by Google
- Individual courses are free!
- Pay for a Nanodegree
  - build a portfolio of apps
  - get a certificate



**Courses and Nanodegree Programs**

# Udacity Android Course Topics

[www.udacity.com/courses/android](http://www.udacity.com/courses/android)

- Android for Beginners
- Developing Android Apps  
for programmers

- Advanced topics
- Performance
  - Material Design
  - ...



Android Development for Beginners

Beginner

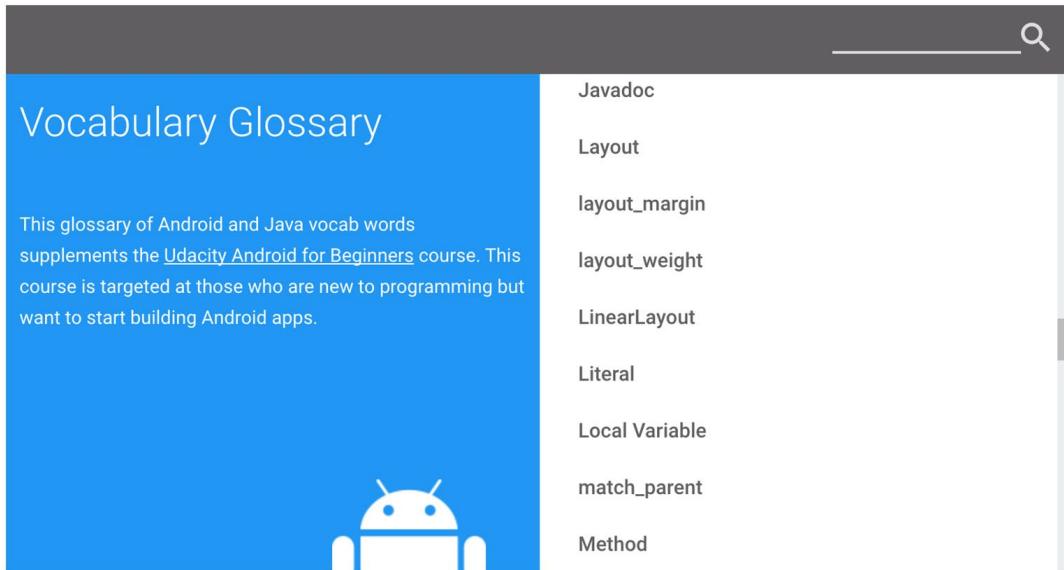
5 PROJECTS

Learn the basics of Android and Java programming, and take the first step on your journey to becoming an Android developer!

BUILT BY Google

# Android Vocabulary tool

[developers.google.com/android/for-all/vocab-words](https://developers.google.com/android/for-all/vocab-words)



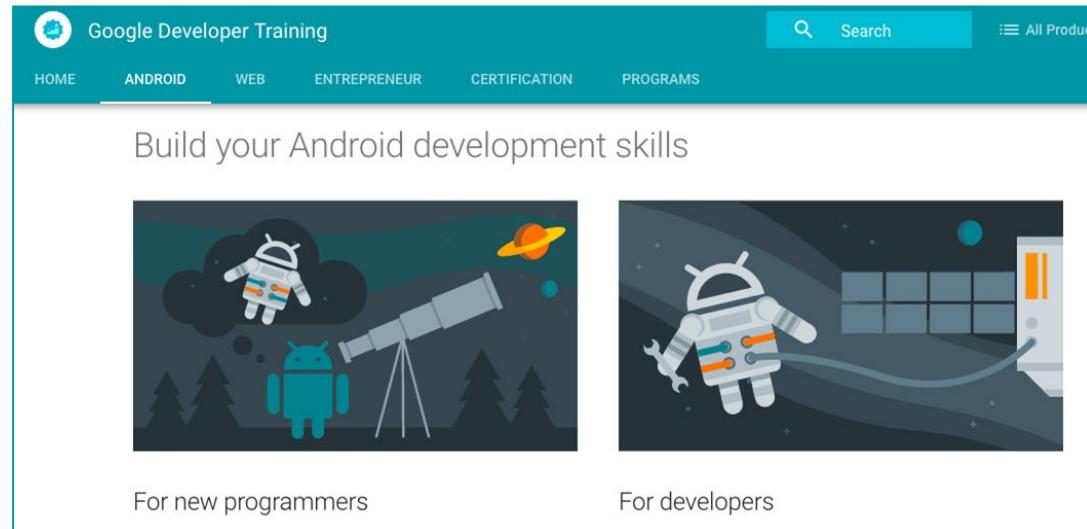
The screenshot shows a mobile-style interface for a vocabulary glossary. At the top is a dark grey header bar with a search icon. Below it is a light blue sidebar containing the title "Vocabulary Glossary" and a paragraph of text about the course. A white Android robot icon is at the bottom of the sidebar. The main content area has a white background with a vertical grey scroll bar on the right. A list of vocabulary terms is displayed in a column:

- Javadoc
- Layout
- layout\_margin
- layout\_weight
- LinearLayout
- Literal
- Local Variable
- match\_parent
- Method

# Google Developer Training website

[developers.google.com/training](https://developers.google.com/training)

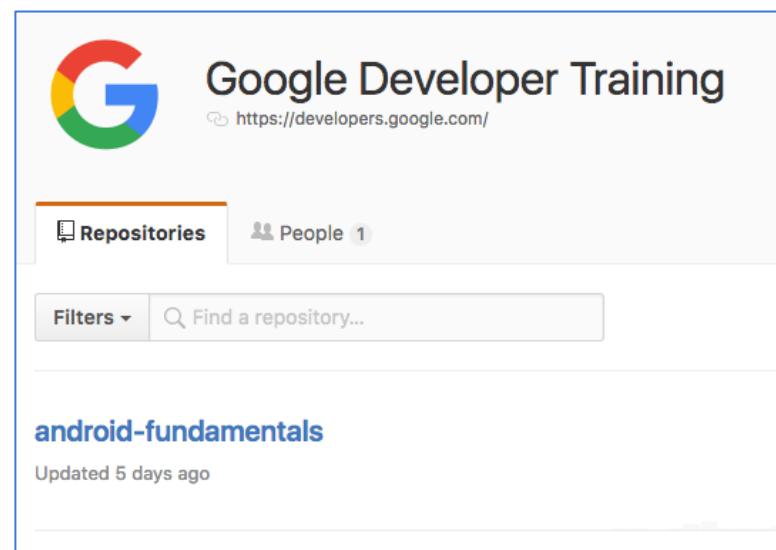
- Course info
- Programs
- Certification details
- Authorized Training Partners



# Source code for exercises on github

<https://github.com/google-developer-training/android-fundamentals>

Starter code and solutions  
for all the practicals and  
many of the challenges



# Download ZIP of repository (or clone)

The screenshot shows a GitHub repository page for the 'google-developer-training / android-fundamentals' repository. The page includes a navigation bar with links for Code, Issues (0), Pull requests (0), Projects (0), Wiki, Pulse, Graphs, and Settings. A red box highlights the repository name in the top left, and a red circle with the number '1' points to it. Below the navigation bar, a message says 'No description or website provided. — Edit'. A summary bar shows 4 commits, 1 branch, 0 releases, and 3 contributors. A red circle with the number '2' points to the 'Clone or download' button, which is highlighted with a green border. The main content area lists several commit squashes by user 'aleksinthecloud'. To the right, a 'Clone with HTTPS' section displays the URL 'https://github.com/google-developer-training/android-fundamentals' and a 'Download ZIP' button, which is also highlighted with a red border and a red circle with the number '3' pointing to it.

1

Unwatch 5 Star 0 Fork 1

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

No description or website provided. — Edit

4 commits 1 branch 0 releases 3 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

aleksinthecloud Squashed commit of the following: ...

AlertSample Initial commit.

DateTimePickers Initial commit.

HelloSharedPrefs Initial commit.

HelloToast Squashed commit of the following:

Clone with HTTPS Use SSH  
Use Git or checkout with SVN using the web URL.  
<https://github.com/google-developer-training/android-fundamentals>

Download ZIP

7 days ago

3

# Get the most from the practicals

- Complete each practical
- Study and learn the corresponding concepts
- Try completing the challenges
  - More detailed app that uses the concepts covered
  - Closer to real-world apps

# Learn more

- [Official Android documentation](#)
- [Get Android Studio](#)
- [Create App Icons with Image Asset Studio](#)
- [Official Android blog](#)
- [Android Developers blog](#)
- [Google I/O Codelabs](#)
- [Stack Overflow](#)
- [Android vocabulary](#)
- [Google Developer Training website](#)

# Learn even more

## Code

- [Source code for exercises on github](#)
- [Android code samples for developers](#)

## Videos

- [Android Developer YouTube channel](#)
- [Udacity online courses](#)



# What's Next?

- Concept Chapter: [1.4 Resources to help you learn](#)
- Practical: [1.4 Available resources](#)

# END

# Activities and Intents

## Lesson 2



Activity lifecycle  
and state

*This work is licensed under a [Creative Commons Attribution 4.0 International License](#).*



# 2.2 Activity lifecycle and state



# Contents

- Activity lifecycle
- Activity lifecycle callbacks
- Activity instance state
- Saving and restoring Activity state

# Activity lifecycle

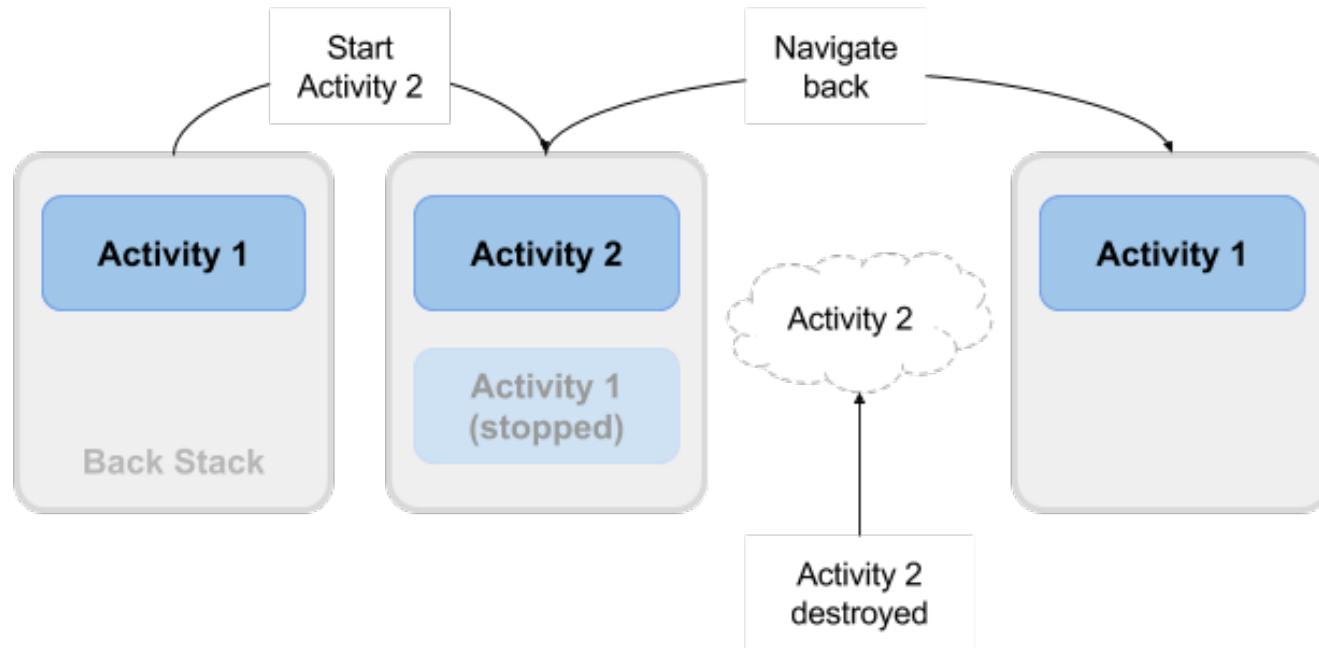
# What is the Activity Lifecycle?

- The set of states an Activity can be in during its lifetime, from when it is created until it is destroyed

More formally:

- A directed graph of all the states an Activity can be in, and the callbacks associated with transitioning from each state to the next one

# What is the Activity Lifecycle?



# Activity states and app visibility

- Created (not visible yet)
- Started (visible)
- Resume (visible)
- Paused(partially invisible)
- Stopped (hidden)
- Destroyed (gone from memory)

State changes are triggered by user action, configuration changes such as device rotation, or system action



# Activity lifecycle callbacks

# Callbacks and when they are called

| **onCreate(Bundle savedInstanceState)**—static initialization

| | **onStart()**—when Activity (screen) is becoming visible

| | **onRestart()**—called if Activity was stopped (calls onStart())

| | | **onResume()**—start to interact with user

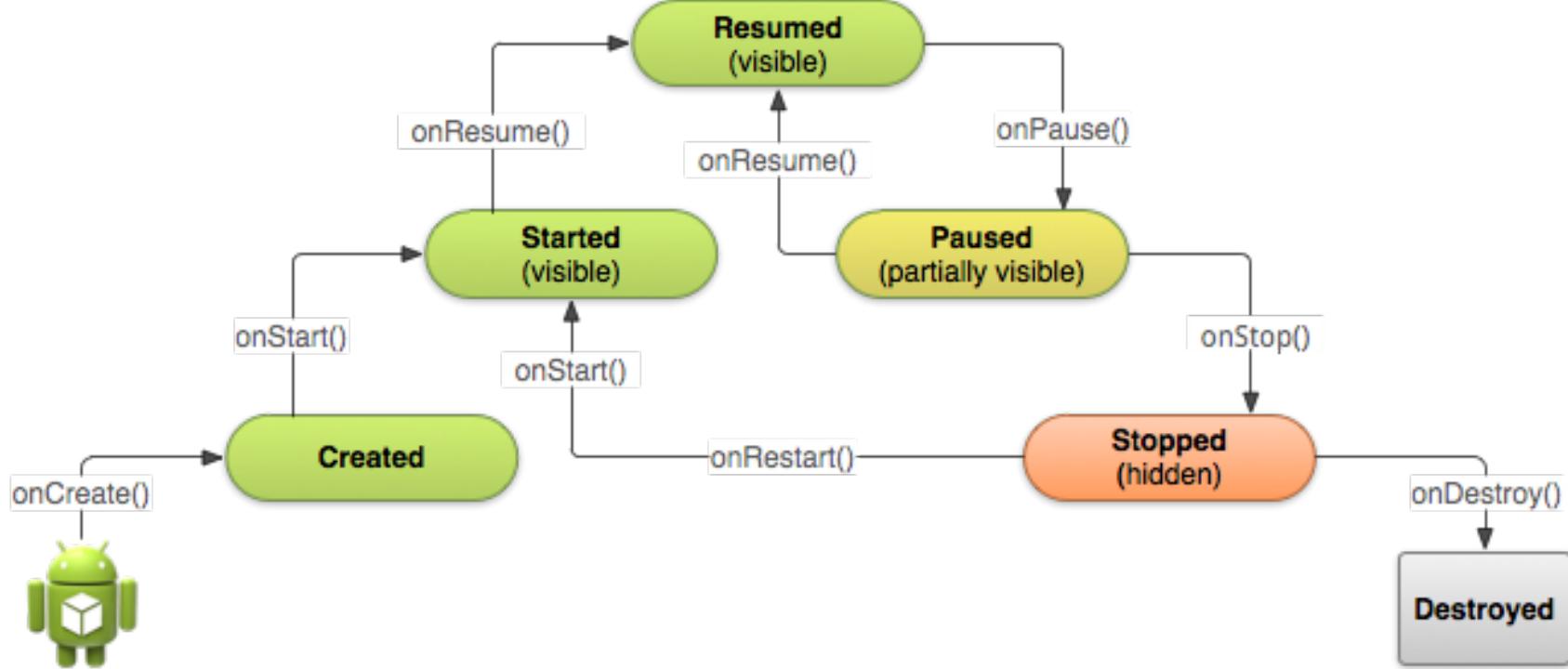
| | | **onPause()**—about to resume PREVIOUS Activity

| | **onStop()**—no longer visible, but still exists and all state info preserved

| **onDestroy()**—final call before Android system destroys Activity



# Activity states and callbacks graph



# Implementing and overriding callbacks

- Only `onCreate()` is required
- Override the other callbacks to change default behavior

# onCreate() -> Created

- Called when the Activity is first created, for example when user taps launcher icon
- Does all static setup: create views, bind data to lists, ...
- Only called once during an activity's lifetime
- Takes a Bundle with Activity's previously frozen state, if there was one
- Created state is always followed by onStart()

# onCreate(Bundle savedInstanceState)

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    // The activity is being created.  
}
```

# onStart() -> Started

- Called when the Activity is becoming visible to user
- Can be called more than once during lifecycle
- Followed by onResume() if the activity comes to the foreground, or onStop() if it becomes hidden



# onStart()

```
@Override  
protected void onStart() {  
    super.onStart();  
    // The activity is about to become visible.  
}
```



# onRestart() -> Started

- Called after Activity has been stopped, immediately before it is started again
- Transient state
- Always followed by onStart()



# onRestart()

```
@Override  
protected void onRestart() {  
    super.onRestart();  
    // The activity is between stopped and started.  
}
```



# onResume() → Resumed/Running

- Called when Activity will start interacting with user
- Activity has moved to top of the Activity stack
- Starts accepting user input
- Running state
- Always followed by onPause()



# onResume()

```
@Override  
protected void onResume() {  
    super.onResume();  
    // The activity has become visible  
    // it is now "resumed"  
}
```



# onPause() → Paused

- Called when system is about to resume a previous Activity
- The Activity is partly visible but user is leaving the Activity
- Typically used to commit unsaved changes to persistent data, stop animations and anything that consumes resources
- Implementations must be fast because the next Activity is not resumed until this method returns
- Followed by either onResume() if the Activity returns back to the front, or onStop() if it becomes invisible to the user

# onPause()

```
@Override  
protected void onPause() {  
    super.onPause();  
    // Another activity is taking focus  
    // this activity is about to be "paused"  
}
```



# onStop() -> Stopped

- Called when the Activity is no longer visible to the user
- New Activity is being started, an existing one is brought in front of this one, or this one is being destroyed
- Operations that were too heavy-weight for onPause()
- Followed by either onRestart() if Activity is coming back to interact with user, or onDestroy() if Activity is going away

# onStop()

```
@Override  
protected void onStop() {  
    super.onStop();  
    // The activity is no longer visible  
    // it is now "stopped"  
}
```



# onDestroy() → Destroyed

- Final call before Activity is destroyed
- User navigates back to previous Activity, or configuration changes
- Activity is finishing or system is destroying it to save space
- Call `isFinishing()` method to check
- System may destroy Activity without calling this, so use `onPause()` or `onStop()` to save data or state

# onDestroy()

```
@Override  
protected void onDestroy() {  
    super.onDestroy();  
    // The activity is about to be destroyed.  
}
```



# Activity instance state

# When does config change?

Configuration changes invalidate the current layout or other resources in your activity when the user:

- Rotates the device
- Chooses different system language, so locale changes
- Enters multi-window mode (from Android 7)



# What happens on config change?

On configuration change, Android:

1. Shuts down Activity  
by calling:

- onPause()
- onStop()
- onDestroy()

2. Starts Activity over again  
by calling:

- onCreate()
- onStart()
- onResume()



# Activity instance state

- State information is created while the Activity is running, such as a counter, user text, animation progression
- State is lost when device is rotated, language changes, back-button is pressed, or the system clears memory



# Saving and restoring Activity state

# What the system saves

- System saves only:
  - State of views with unique ID (`android:id`) such as text entered into `EditText`
  - Intent that started activity and data in its extras
- You are responsible for saving other activity and user progress data

# Saving instance state

Implement `onSaveInstanceState()` in your Activity

- Called by Android runtime when there is a possibility the Activity may be destroyed
- Saves data only for this instance of the Activity during current session



# onSaveInstanceState(Bundle outState)

```
@Override  
public void onSaveInstanceState(Bundle outState) {  
    super.onSaveInstanceState(outState);  
  
    // Add information for saving HelloToast counter  
    // to the outState bundle  
    outState.putString("count",  
        String.valueOf(mShowCount.getText()));  
}
```

# Restoring instance state

Two ways to retrieve the saved Bundle

- in `onCreate(Bundle mySavedState)`  
Preferred, to ensure that your user interface, including any saved state, is back up and running as quickly as possible
- Implement callback (called after `onStart()`)  
`onRestoreInstanceState(Bundle mySavedState)`

# Restoring in onCreate()

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    mShowCount = findViewById(R.id.show_count);  
  
    if (savedInstanceState != null) {  
        String count = savedInstanceState.getString("count");  
        if (mShowCount != null)  
            mShowCount.setText(count);  
    }  
}
```



# onRestoreInstanceState(Bundle state)

```
@Override  
public void onRestoreInstanceState (Bundle mySavedState) {  
    super.onRestoreInstanceState(mySavedState);  
  
    if (mySavedState != null) {  
        String count = mySavedState.getString("count");  
        if (count != null)  
            mShowCount.setText(count);  
    }  
}
```



# Instance state and app restart

When you stop and restart a new app session, the Activity instance states are lost and your activities will revert to their default appearance

If you need to save user data between app sessions, use shared preferences or a database.



# Learn more

- [Activities \(API Guide\)](#)
- [Activity \(API Reference\)](#)
- [Managing the Activity Lifecycle](#)
- [Pausing and Resuming an Activity](#)
- [Stopping and Restarting an Activity](#)
- [Recreating an Activity](#)
- [Handling Runtime Changes](#)
- [Bundle](#)



# What's Next?

- Concept Chapter: [2.2 Activity lifecycle and state](#)
- Practical: [2.2 Activity lifecycle and state](#)



# END

# Activities and Intents

Lesson 2



This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# 2.3 Implicit Intents

# Contents

- Intent—recap
- Implicit Intent overview
- Sending an implicit Intent
- Receiving an implicit Intent

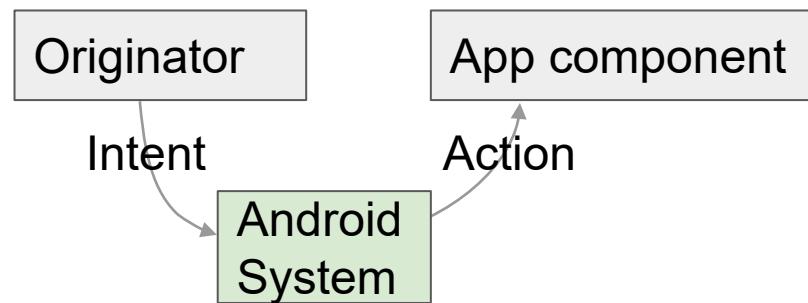


# Recap: Intent

# What is an Intent?

An Intent is:

- Description of an operation to be performed
- Messaging object used to request an action from another app component via the Android system.



# What can an Intent do?

An Intent can be used to:

- start an Activity
- start a Service
- deliver a Broadcast

Services and Broadcasts are covered in other lessons

# Explicit vs. implicit Intent

**Explicit Intent** – Starts an Activity of a specific class

**Implicit Intent** – Asks system to find an Activity class with a registered handler that can handle this request

# Implicit Intent overview

# What you do with an implicit Intent

- Start an Activity in another app by describing an action you intend to perform, such as "share an article", "view a map", or "take a picture"
- Specify an action and optionally provide data with which to perform the action
- Don't specify the target Activity class, just the intended action

# What system does with implicit Intent

- Android runtime matches the implicit intent request with registered intent handlers
- If there are multiple matches, an App Chooser will open to let the user decide

# How does implicit Intent work?

1. The Android Runtime keeps a list of registered Apps
2. Apps have to register via `AndroidManifest.xml`
3. Runtime receives the request and looks for matches
4. Android runtime uses Intent filters for matching
5. If more than one match, shows a list of possible matches and lets the user choose one
6. Android runtime starts the requested activity

# App Chooser

When the Android runtime finds multiple registered activities that can handle an implicit Intent, it displays an [App Chooser](#) to allow the user to select the handler



# Sending an implicit Intent

# Sending an implicit Intent

## 1. Create an Intent for an action

```
Intent intent = new Intent(Intent.ACTION_CALL_BUTTON);
```

User has pressed Call button – start Activity that can make a call (no data is passed in or returned)

## 1. Start the Activity

```
if (intent.resolveActivity(getApplicationContext()) != null) {  
    startActivity(intent);  
}
```

# Avoid exceptions and crashes

Before starting an implicit Activity, use the package manager to check that there is a package with an Activity that matches the given criteria.

```
Intent myIntent = new Intent(Intent.ACTION_CALL_BUTTON);  
if (intent.resolveActivity(getApplicationContext()) != null) {  
    startActivity(intent);  
}
```

# Sending an implicit Intent with data URI

## 1. Create an Intent for action

```
Intent intent = new Intent(Intent.ACTION_DIAL);
```

## 1. Provide data as a URI

```
intent.setData(Uri.parse("tel:8005551234"));
```

## 1. Start the Activity

```
if (intent.resolveActivity(getApplicationContext()) != null) {  
    startActivity(intent);  
}
```

# Providing the data as URI

Create an URI from a string using `Uri.parse(String uri)`

- `Uri.parse("tel:8005551234")`
- `Uri.parse("geo:0,0?q=brooklyn%20bridge%2C%20brooklyn%2C%20ny")`
- `Uri.parse("http://www.android.com");`

[Uri documentation](#)

# Implicit Intent examples

## Show a web page

```
Uri uri = Uri.parse("http://www.google.com");  
Intent it = new Intent(Intent.ACTION_VIEW,uri);  
startActivity(it);
```

## Dial a phone number

```
Uri uri = Uri.parse("tel:8005551234");  
Intent it = new Intent(Intent.ACTION_DIAL, uri);  
startActivity(it);
```



# Sending an implicit Intent with extras

## 1. Create an Intent for an action

```
Intent intent = new Intent(Intent.ACTION_WEB_SEARCH);
```

## 1. Put extras

```
String query = edittext.getText().toString();
intent.putExtra(SearchManager.QUERY, query));
```

## 1. Start the Activity

```
if (intent.resolveActivity(getApplicationContext()) != null) {
    startActivity(intent);
}
```



# Category

Additional information about the kind of component to handle the intent.

- **CATEGORY\_OPENABLE**

Only allow URIs of files that are openable

- **CATEGORY\_BROWSABLE**

Only an Activity that can start a web browser to display data referenced by the URI



# Sending an implicit Intent with type and category

## 1. Create an Intent for an action

```
Intent intent = new Intent(Intent.ACTION_CREATE_DOCUMENT);
```

## 1. Set mime type and category for additional information

```
intent.setType("application/pdf"); // set MIME type  
intent.addCategory(Intent.CATEGORY_OPENABLE);
```

*continued on next slide...*

# Sending an implicit Intent with type and category

## 3. Start the Activity

```
if (intent.resolveActivity(getApplicationContext()) != null) {  
    startActivityForResult(myIntent,ACTIVITY_REQUEST_CREATE_FILE);  
}
```

## 4. Process returned content URI in onActivityResult()

# Common actions for an implicit Intent

Common actions include:

- [ACTION\\_SET\\_ALARM](#)
- [ACTION\\_IMAGE\\_CAPTURE](#)
- [ACTION\\_CREATE\\_DOCUMENT](#)
- [ACTION\\_SENDTO](#)
- and many more

# Apps that handle common actions

Common actions are usually handled by installed apps (both system apps and other apps), such as:

- Alarm Clock, Calendar, Camera, Contacts
- Email, File Storage, Maps, Music/Video
- Notes, Phone, Search, Settings
- Text Messaging and Web Browsing

- [List of common actions for an implicit intent](#)
- [List of all available actions](#)

# Receiving an Implicit Intent

# Register your app to receive an Intent

- Declare one or more Intent filters for the Activity in `AndroidManifest.xml`
- Filter announces ability of Activity to accept an implicit Intent
- Filter puts conditions on the Intent that the Activity accepts

# Intent filter in AndroidManifest.xml

```
<activity android:name="ShareActivity">  
    <intent-filter>  
        <action android:name="android.intent.action.SEND"/>  
        <category android:name="android.intent.category.DEFAULT"/>  
        <data android:mimeType="text/plain"/>  
    </intent-filter>  
</activity>
```

# Intent filters: action and category

- **action** – Match one or more action constants
  - android.intent.action.VIEW – matches any Intent with [ACTION VIEW](#)
  - android.intent.action.SEND – matches any Intent with [ACTION SEND](#)
- **category** – additional information ([list of categories](#))
  - android.intent.category.BROWSABLE – can be started by web browser
  - android.intent.category.LAUNCHER – Show activity as launcher icon

# Intent filters: data

- **data** – Filter on data URIs, MIME type
  - `android:scheme="https"`–require URIs to be https protocol
  - `android:host="developer.android.com"`–only accept an Intent from specified hosts
  - `android:mimeType="text/plain"`–limit the acceptable types of documents

# An Activity can have multiple filters

```
<activity android:name="ShareActivity">  
    <intent-filter>  
        <action android:name="android.intent.action.SEND"/>  
        ...  
    </intent-filter>  
    <intent-filter>  
        <action android:name="android.intent.action.SEND_MULTIPLE"/>  
        ...  
    </intent-filter>  
</activity>
```

An Activity can have several filters

# A filter can have multiple actions & data

```
<intent-filter>  
  
    <action android:name="android.intent.action.SEND"/>  
    <action android:name="android.intent.action.SEND_MULTIPLE"/>  
    <category android:name="android.intent.category.DEFAULT"/>  
    <data android:mimeType="image/*"/>  
    <data android:mimeType="video/*"/>  
  
</intent-filter>
```

# Learn more

# Learn more

- [Intent class documentation](#)
- [Uri documentation](#)
- [List of common apps that respond to implicit intents](#)
- [List of available actions](#)
- [List of categories](#)
- [Intent Filters](#)

# What's Next?

- Concept Chapter: [2.3 Implicit Intents](#)
- Practical: [2.3 Implicit Intents](#)



# END

# Testing, debugging, and using support libraries

## Lesson 3



# 3.1 The Android Studio debugger

# Contents

- All code has bugs
- Android Studio logging
- Android Studio debugger
- Working with breakpoints
- Changing variables
- Stepping through code

# All Code Has Bugs

# Bugs

- Incorrect or unexpected result, wrong values
- Crashes, exceptions, freezes, memory leaks
- Causes
  - Human Design or Implementation Error > Fix your code
  - Software fault, but in libraries > Work around limitation
  - Hardware fault or limitation -> Make it work with what's available

Origin of the term "bug" (it's not what you think)

# Debugging

- Find and fix errors
- Correct unexpected and undesirable behavior
- Unit tests help identify bugs and prevent regression
- User testing helps identify interaction bugs

# Android Studio debugging tools

Android Studio has tools that help you

- identify problems
- find where in the source code the problem is created
- so that you can fix it

# Logging with Android Studio

# Add Log messages to your code

```
import android.util.Log;  
  
// Use class variable with class name as tag  
private static final String TAG =  
    MainActivity.class.getSimpleName();  
  
// Show message in Logcat pane of Android Studio  
// Log.<log-level>(TAG, "Message");  
Log.d(TAG, "Hello World");
```



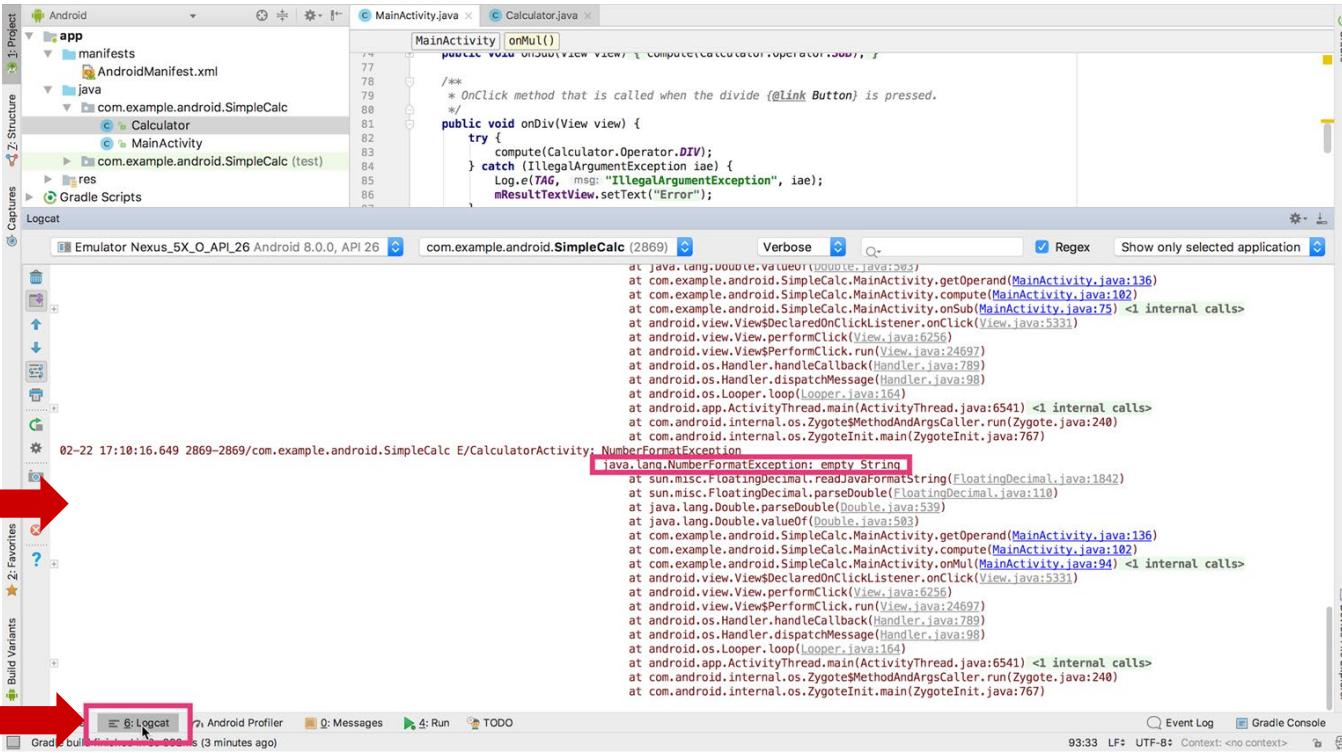
# Open Logcat pane

Logcat

pane

Logcat

tab



# Inspect logging messages

The screenshot shows the Android Studio interface. The top half displays the code editor with `MainActivity.java` open, containing the line `Log.d("MainActivity", "Hello World");`. A red circle labeled '1' highlights this line. The bottom half shows the Logcat window with the log message `09-12 14:28:07.971 4304 /com.example.android.helloworld D/MainActivity: Hello World`. A red circle labeled '2' highlights this message in the log.

```
09-12 14:28:07.971 4304 /com.example.android.helloworld D/MainActivity: Hello World
```

09-12 14:28:07.750 4304 /com.example.android.helloworld D/W/art: Before Android 4.1, method android.graphics.PorterDuffColorFilter android.support.graphics.drawable.Drawable.setPorterDuffColorFilter() was defined without parameters. This has been explicitly disabled in 4.1 to avoid a performance impact. Take a look at the android/support/graphics/Drawable.html source code to see why.

09-12 14:28:07.888 4304 /com.example.android.helloworld D/MainActivity: Hello World

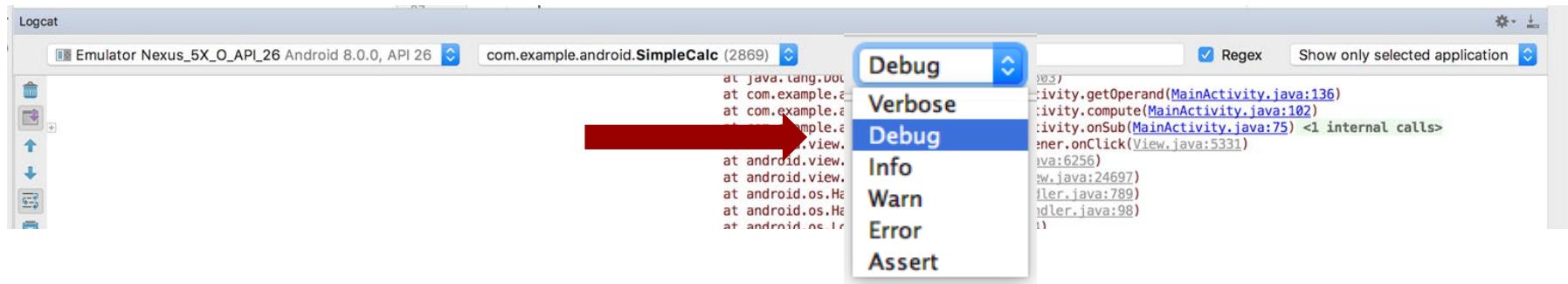
09-12 14:28:07.971 4304 /com.example.android.helloworld D/OpenGLESRenderer: Use EGL\_SWAP\_BEHAVIOR\_PRESERVED: true

[ 09-12 14:28:07.974 4304: 4304 D/ HostConnection::get() New Host Connection established 0x7f4bb1d06500, tid 4304

09-12 14:28:08.026 4304 /com.example.android.helloworld I/OpenGLESRenderer: Initialized EGL, version 1.4



# Choose visible logging level



Displays logs with levels at  
this level or higher

# Log Levels

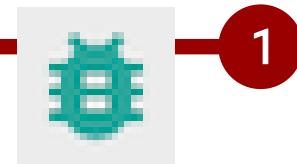
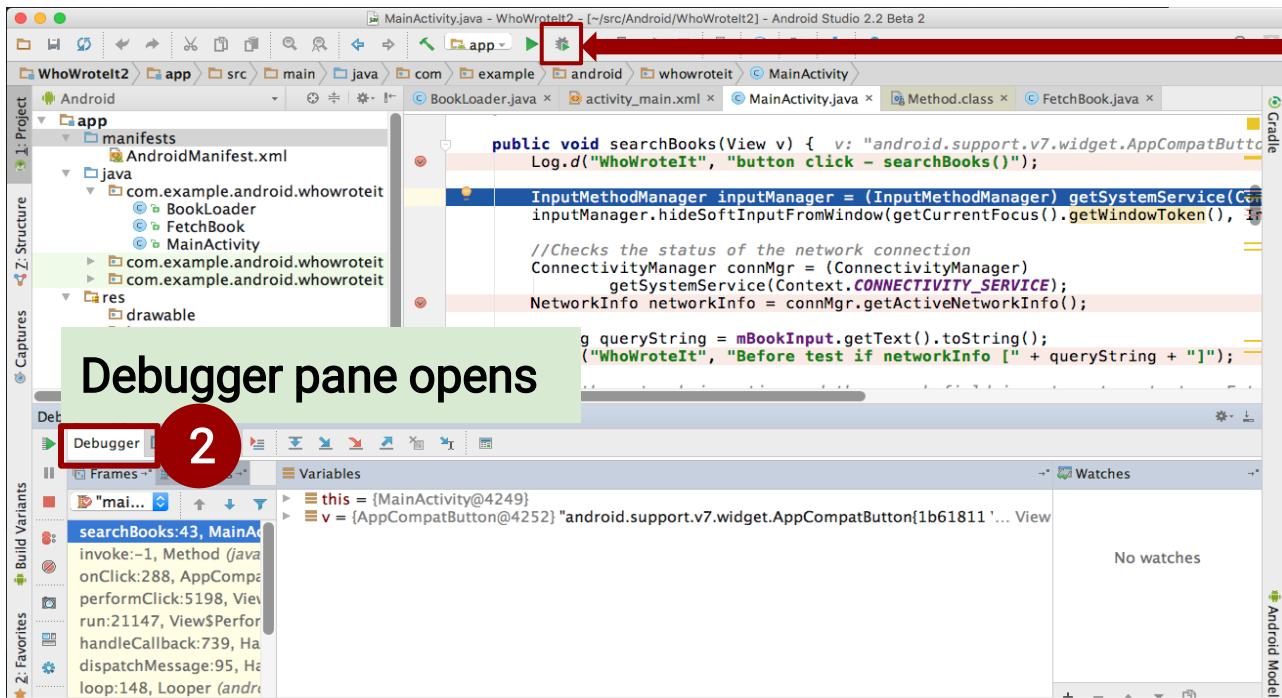
- **Verbose** - All verbose log statements and comprehensive system
- **Debug** - All debug logs, variable values, debugging notes
- **Info** - Status info, such as database connection
- **Warning** - Unexpected behavior, non-fatal issues
- **Error** - Serious error conditions, exceptions, crashes only

# Debugging with Android Studio

# What you can do

- Run in debug mode with attached debugger
- Set and configure breakpoints
- Halt execution at breakpoints
- Inspect execution stack frames and variable values
- Change variable values
- Step through code line by line
- Pause and resume a running program

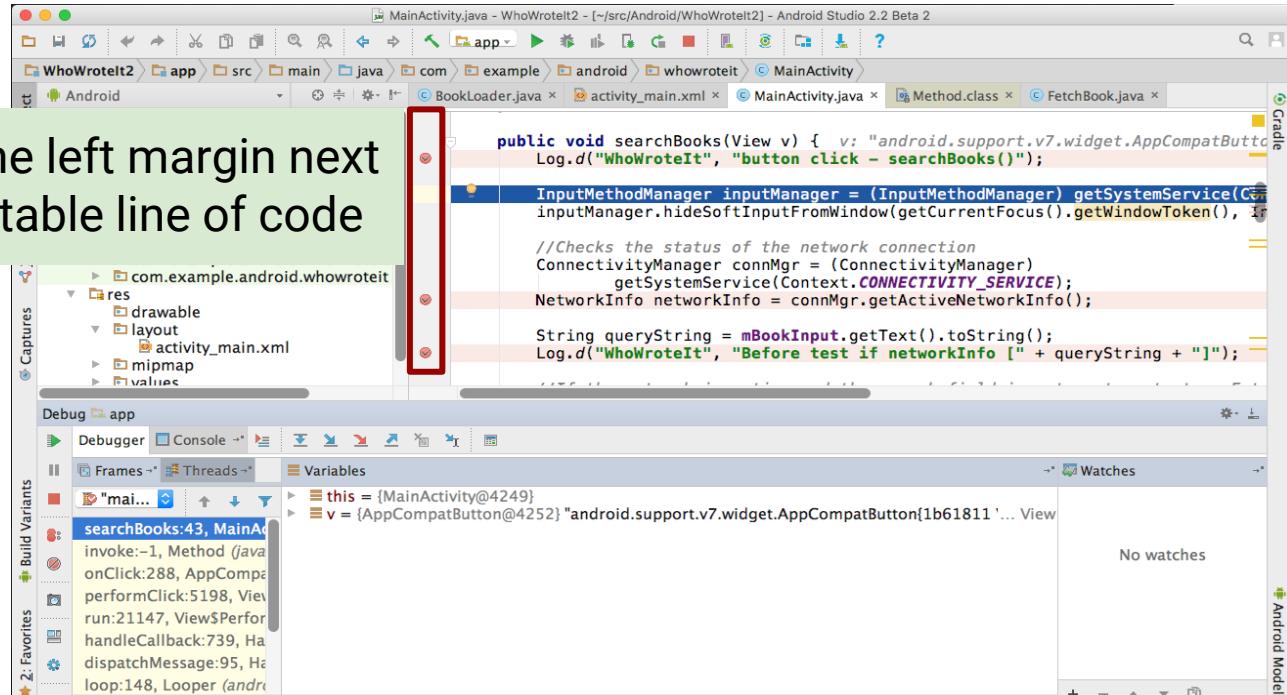
# Run in debug mode



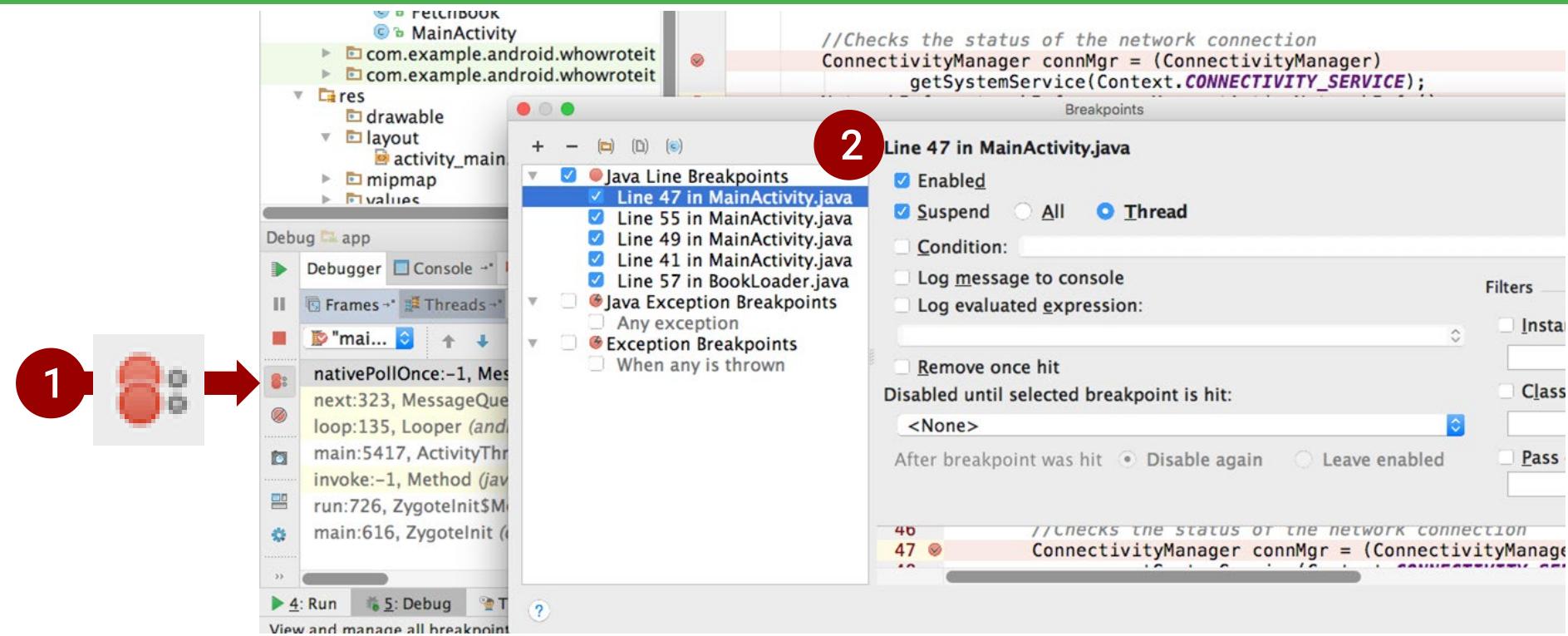
Menu:  
Run > Debug 'your app'

# Set breakpoints

Click in the left margin next to executable line of code

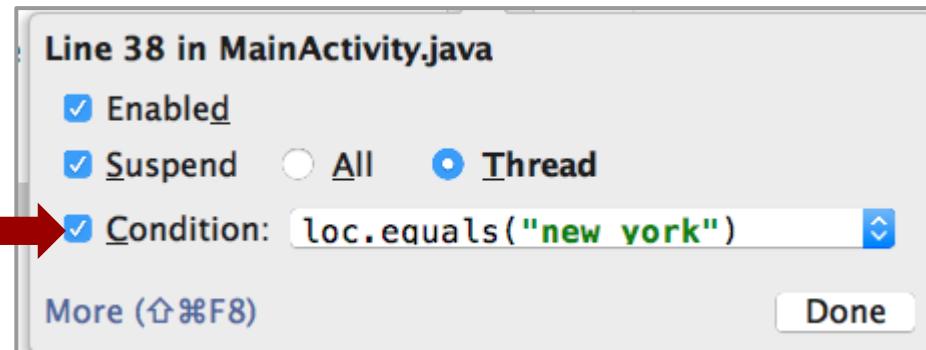


# Edit breakpoint properties



# Make breakpoints conditional

- In properties dialog or right -click existing breakpoint
- Any Java expression that returns a boolean
- Code completion helps you write conditions



# Run until app stops at breakpoint

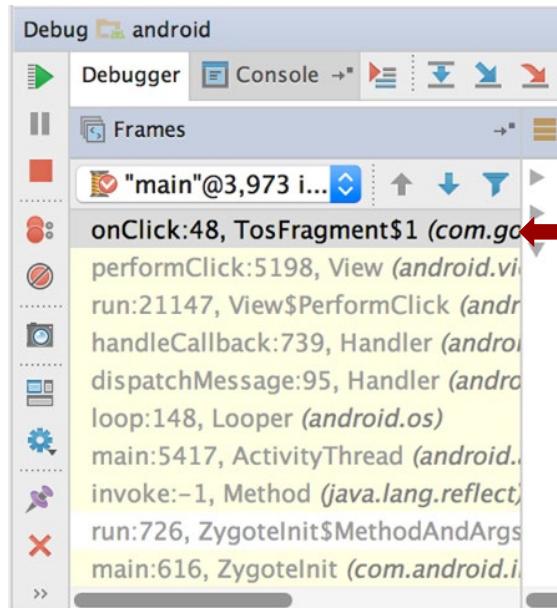
The screenshot shows the Android Studio interface during debugging. The code editor displays a Java file with several breakpoints set. A red arrow points to the first breakpoint in the code:

```
public void searchBooks(View v) { v: "android.support.v7.widget.AppCompatButton"
    Log.d("WhoWroteIt", "button click - searchBooks()");
    InputMethodManager inputManager = (InputMethodManager) getSystemService(Context.INPUT_METHOD_SERVICE);
    inputManager.hideSoftInputFromWindowgetCurrentFocus().getWindowToken(), InputMethodManager.HIDE_NOT_ALWAYS);
    //Checks the status of the network connection
    ConnectivityManager connMgr = (ConnectivityManager)
        getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
    String queryString = mBookInput.getText().toString();
    Log.d("WhoWroteIt", "Before test if networkInfo [" + queryString + "]");
}
```

Below the code editor, three callout boxes provide information about the debugger:

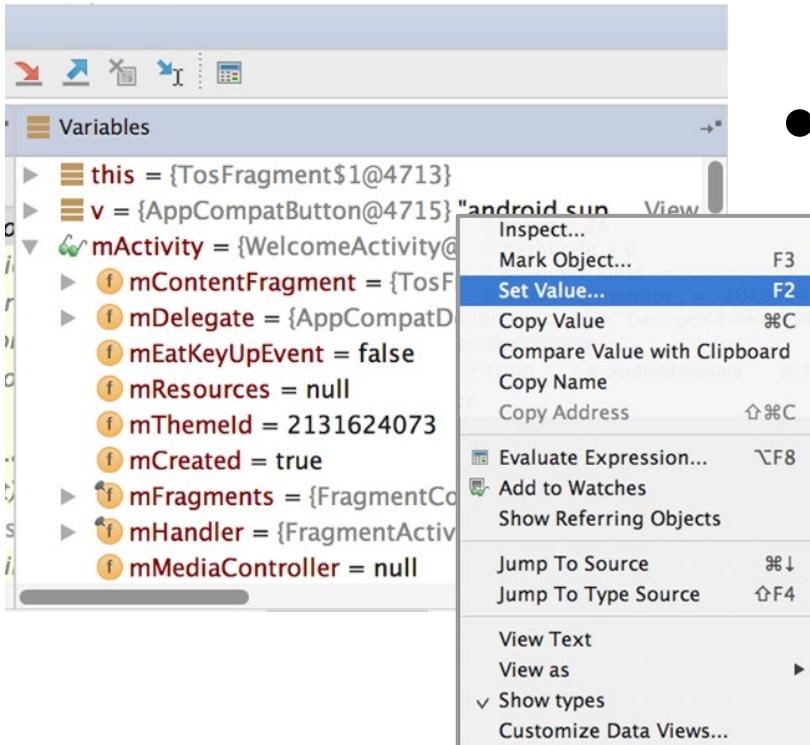
- Frames**: Points to the Frames toolbar in the Debug tab.
- Variables in scope**: Points to the Variables toolbar in the Debug tab.
- Watches (C/C++)**: Points to the Watches toolbar in the Debug tab.

# Inspect frames



Top frame is where execution is halted in your code

# Inspect and edit variables

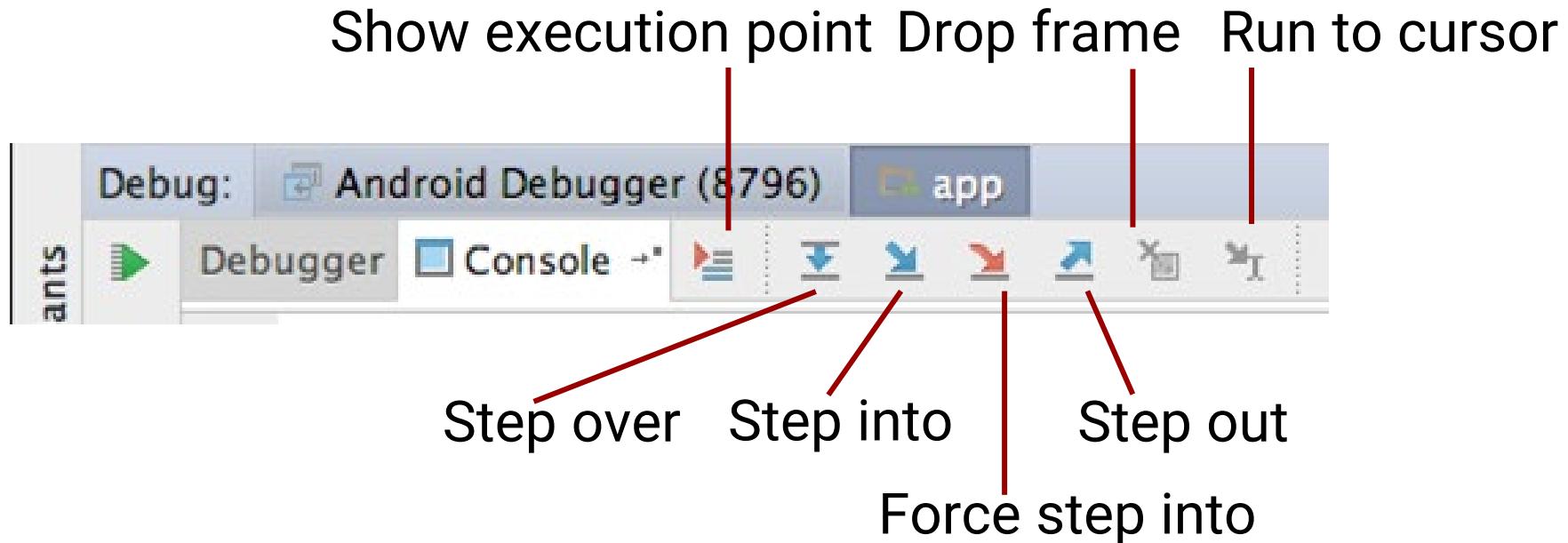


- Right-click on variable for menu

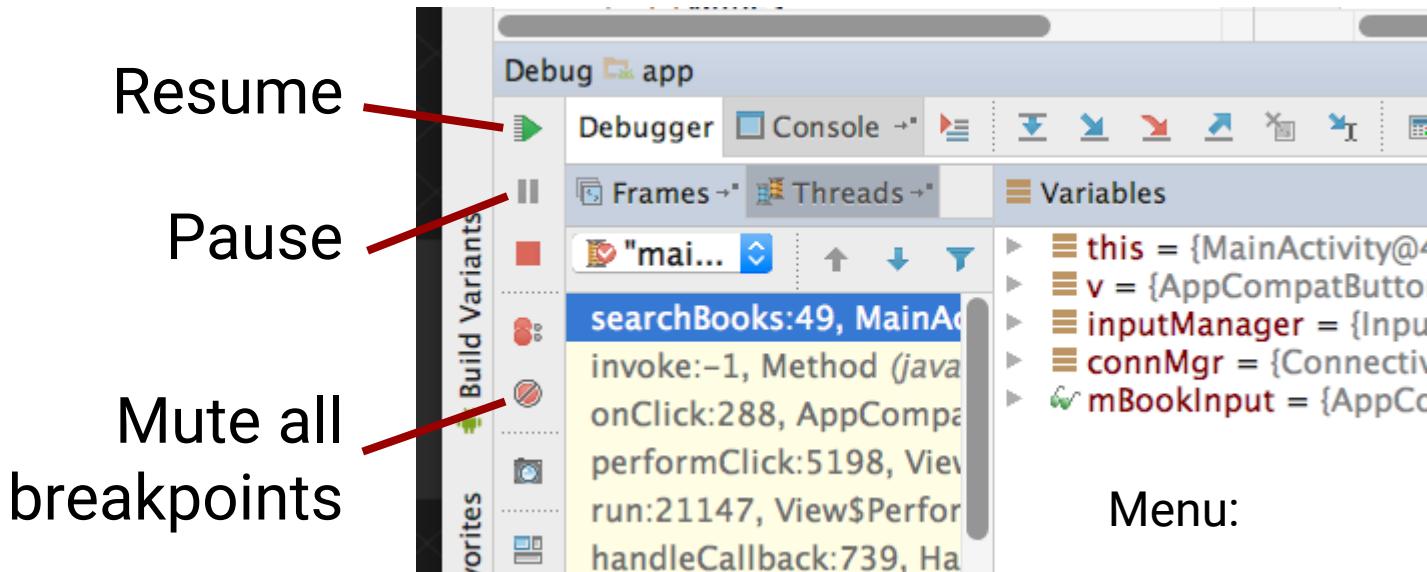
# Basic Stepping Commands

Step Over	F8	Step to the next line in current file
Step Into	F7	Step to the next executed line
Force Step Into	↑F7	Step into a method in a class that you wouldn't normally step into, like a standard JDK class
Step Out	↑F8	Step to first executed line after returning from current method
Run to Cursor	↖F9	Run to the line where the cursor is in the file

# Stepping through code



# Resume and Pause



Menu:

Run->Pause Program...

Run->Resume Program...

# Learn more

- [Debug Your App](#) (Android Studio User Guide)
- [Debugging and Testing in Android Studio](#) (video)



# What's Next?

- Concept Chapter: [3.1 The Android Studio debugger](#)
- Practical: [3.1 The debugger](#)



# END

# Testing, debugging, and using support libraries

## Lesson 3



This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# 3.2 App testing



# Contents

- Why testing is worth your time
- Unit testing

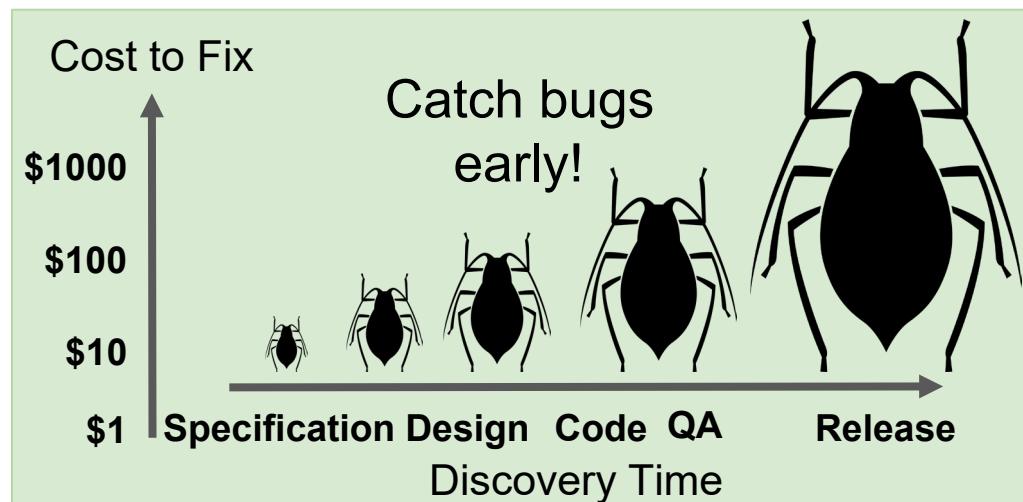
Note: User interface testing (instrumented testing) is covered in another chapter



# Testing rocks

# Why should you test your app?

- Find and fix issues early
- Less costly
- Takes less effort
- Costs to fix bugs increases with time



# Types of testing

- Levels of Testing
  - Component, integration, protocol, system
- Types of Testing
  - Installation, compatibility, regression, acceptance
  - Performance, scalability, usability, security
- User interface and interaction tests
  - Automated UI testing tools
  - Instrumented testing (covered in another chapter)

# Test-Driven Development (TDD)

1. Define a test case for a requirement
2. Write tests that assert all conditions of the test case
3. Write code against the test
4. Iterate on and refactor code until it passes the test
5. Repeat until all requirements have test cases, all tests pass, and all functionality has been implemented

# Tests in your project

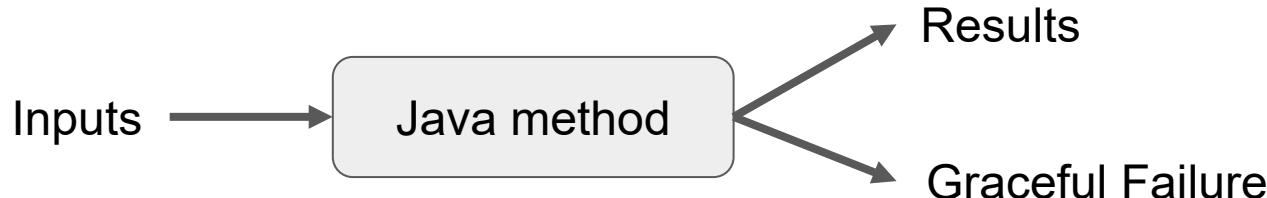
Android Studio creates three source sets for your project

- **main**—code and resources
- **(test)**—local unit tests
- **(androidTest)**—instrumented tests

# Local Unit Tests

# Unit tests

- Smallest testable parts of your program
- Isolate each component and demonstrate the individual parts are correct
- Java Method tests



# Local unit tests in JUnit

- Compiled and run entirely on your local machine with the Java Virtual Machine (JVM)
- Use to test the parts of your app (such as the internal logic):
  - If you don't need access to Android framework or device/emulator
  - If you can create fake (mock) objects that pretend to behave like the framework equivalents
- Unit tests are written with JUnit, a common unit testing framework for Java.

# Local unit tests in your project

- Tests are in the same package as the associated application class
- Only org.junit imported – no Android classes
- Project path for test classes: .../module-name/src/**test/java**/

# Imports for JUnit

```
// Annotations
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;

// Basic JUnit4 test runner
import org.junit.runners.JUnit4;

// assertThat method
import static org.junit.Assert.assertThat;
```

# Testing class

```
/**  
 * JUnit4 unit tests for the calculator logic.  
 * These are local unit tests; no device needed  
 */  
@RunWith(JUnit4.class) // Specify the test runner  
public class CalculatorTest { // Name it what you are testing  
}
```

# ExampleTest

```
/  
**  
* Test for simple addition.  
* Each test is identified by a @Test annotation.  
*/  
@Test  
public void addTwoNumbers() {  
    double resultAdd = mCalculator.add(1d, 1d);  
    assertThat(resultAdd, is(equalTo(2d)));  
}
```

# @Test Annotation

- Tells JUnit this method is a test method (JUnit 4)
- Information to the test runner
- Not necessary anymore to prefix test methods with "test"

# setUp() method

```
/**  
 * Set up the environment for testing  
 */  
  
@Before  
public void setUp() {  
    mCalculator = new Calculator();  
}
```

- Sets up environment for testing
- Initialize variables and objects used in multiple tests



# `tearDown()` method

```
/**  
 * Release external resources  
 */  
@After  
public void tearDown() {  
    ....  
}
```

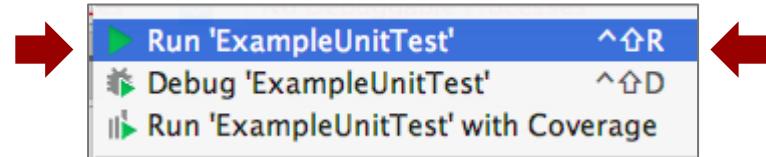
- Frees resources



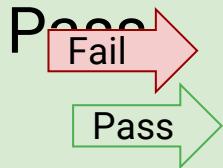
# Running tests in Android Studio

# Starting a test run

- Right-click test class and select  
Run 'app\_name' test
- Right-click test package and select  
Run tests in 'package'



# Passing and failing



Fail

Result details

# Testing floating point results

# Testing floating point

- Be careful with floating point tests
- Recall from basic computer science:
  - Floating point arithmetic is not accurate in binary

# Test fails with floating point numbers

The screenshot shows an Android Studio interface with the following details:

- Code Editor:** Displays a Java test class named `FloatingPointUnitTest`. It contains a single test method `addition_isCorrect` that uses `assertEquals` to compare `.3d`, `.1d+.2d`, and `0d`. The third argument, `0d`, is highlighted with a red box.
- Test Result Bar:** Shows a red bar indicating "1 test failed - 10ms".
- Logcat Output:** Shows the failure message:

```
java.lang.AssertionError:  
Expected :0.3  
Actual   :0.3000000000000004  
<click to see difference>
```

The entire message is highlighted with a red box.
- Call Stack:** Shows the stack trace:

```
<1 internal calls>  
at org.junit.Assert.failNotEquals(Assert.java:834) <2 internal calls>  
at com.example.android.simplecalc.FloatingPointUnitTest.addition_isCorrect()
```

# Fix test with floating point numbers

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

- Project Structure:** Shows three test packages: ExampleUnitTest, FloatingPointUnitTest, and UtilityTest.
- Run Tab:** Set to "FloatingPointUnitTest".
- Code Editor:** Displays the `FloatingPointUnitTest` class with a single test method `addition_isCorrect`. The code is:

```
public class FloatingPointUnitTest {  
    @Test  
    public void addition_isCorrect() throws Exception {  
        assertEquals(.3d, .1d+.2d, .0005d); // 3rd arg is 'epsilon'  
    }  
}
```

A green box highlights the argument `.0005d`.
- Output Window:** Shows the test results:

```
1 test passed - 0ms  
/Library/Java/JavaVirtualMachines/jdk1.8.0_91.jdk/Contents/Home/bin/java ...  
Process finished with exit code 0
```

**Text Overlay:** A green-bordered box contains the text: "They are the same within .0005 in this test"

# Learn more

- [Getting Started with Testing](#)
- [Best Practices for Testing](#)
- [Building Local Unit Tests](#)
- [JUnit 4 Home Page](#)
- [JUnit 4 API Reference](#)
- [Android Testing Codelab](#)
- [Android Tools Protip: Test Size Annotations](#)
- [Android Testing Support - Testing Patterns \(video\)](#)

# What's Next?

- Concept Chapter: [3.2 App testing](#)
- Practical: [3.2 Unit tests](#)



# END

# Testing, debugging, and using support libraries

## Lesson 3



This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# 3.3 The Android Support Library

# Contents

- What are the Android support libraries?
- Features
- Selected libraries
- Setting up and using support libraries



# What are the Android support libraries?

- More than 25 libraries in the Android SDK that provide features not built into the Android framework

# Features

# Support library features

Support libraries provide:

- Backward-compatible versions of components
- Additional layout and UI elements, such as RecyclerView
- Different form factors, such as TV, wearables
- Material design and other new UI components for older Android versions
- and more....



# Backward compatibility

Always use the support library version of a component if one is available

- No need to create different app versions for different versions of Android
- System picks the best version of the component



# Support libraries versions

- Libraries for Android 2.3 (API level 9) and higher
- Recommended you include the [v4 support](#) and [v7 appcompat](#) libraries for the features your app uses

# Libraries

# v4 Support Libraries

- Largest set of APIs
- App components, UI features
- Data handling
- Network connectivity
- Programming utilities



# v4 Support Libraries

- compat—compatibility wrappers
- core-utils—utility classes (eg., AsyncTaskLoader)
- core-ui—variety of UI components
- media-compat—back ports of media framework
- fragment—UI component

# v7 Support Libraries

- Backwards compatibility
- TV-specific components
- UI components and layouts
- Google Cast support
- Color palette
- Preferences



# v7 Support Libraries

- appcompat—compatibility wrappers
- cardview— new UI component (material design)
- gridlayout—rectangular cell (matrix) Layout
- mediarouter—route A/V streams
- palette—extracting color from an image
- recyclerview—efficient scrolling view
- preference—modifying UI settings

# v7 appcompat library

- ActionBar and sharing actions
- You should always include this library and make [AppCompatActivity](#) the parent of your activities

`com.android.support:appcompat-v7:27.1.1`

# Complete list of libraries

- ... and many more
- For latest libraries and versions of libraries
  - [Support Library Features documentation](#)
  - [API Reference](#) (all packages that start with android.support)

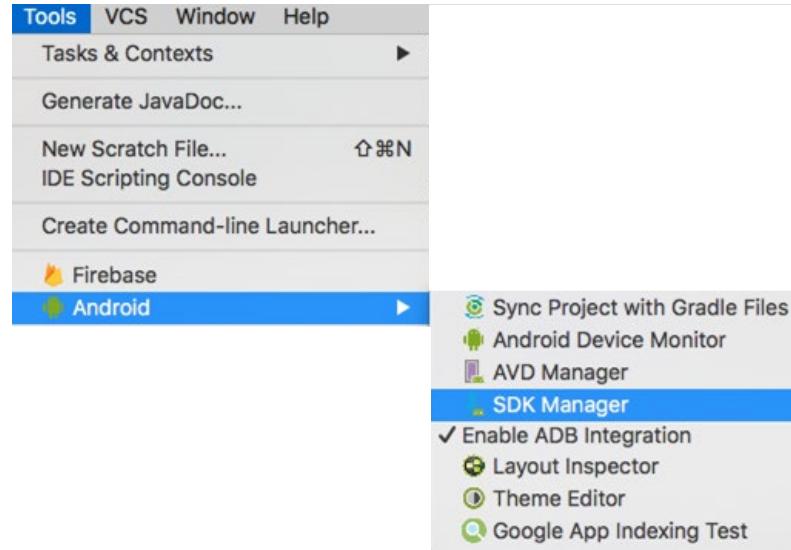
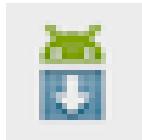
# Using support libraries

# Support libraries

- Part of Android SDK and available through SDK Manager
- In Android Studio, locally stored in Android Support Repository
- Include in build.gradle of module to use with your project

# Start SDK Manager in Android Studio

## 1. Tools > Android > SDK Manager

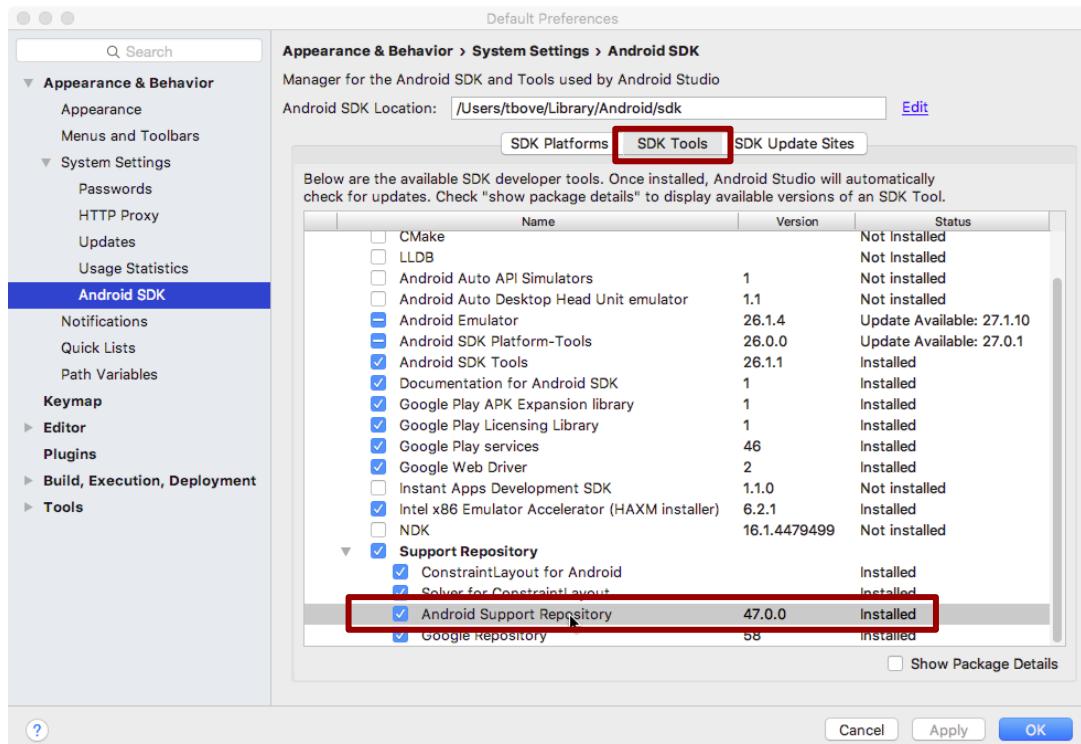


# Find the Android Support Library

1. SDK Tools tab
2. Find Android Support Repository
3. Must be Installed

If Not Installed or Update Available

1. Check checkbox and **Apply**
2. In dialog, confirm components and click **OK** to install
3. When done, click **Finish**
4. Verify that it is **Installed** now



# Find the dependency identifier

1. Open [Support Library Features](#) page
2. Find the feature you want
  - o For example, the recommended [v7 appcompat](#) library
3. Copy the build script dependency identifier for the library
  - o `com.android.support:appcompat-v7:26.1.0`

# Add dependency to build.gradle

1. open build.gradle (Module: app)
2. In the dependencies section, add dependency for support library if not already included from template
  - o `implementation 'com.android.support:appcompat-v7:26.1.0'`
3. Update the version number, if prompted
4. Sync Now when prompted

# Updated build.gradle (Module: app)

```
dependencies {  
    implementation fileTree(dir: 'libs', include: ['*.jar'])  
    implementation 'com.android.support:appcompat-v7:26.1.0'  
    implementation  
        'com.android.support.constraint:constraint-layout:1.0.2'  
    testImplementation 'junit:junit:4.12'  
    androidTestImplementation  
        'com.android.support.test:runner:1.0.1'  
    androidTestImplementation  
        'com.android.support.test.espresso:espresso-core:3.0.1'  
}
```



# Learn more

- [Android Support Library](#) (introduction)
- [Support Library Setup](#)
- [Support Library Features](#)
- [API Reference](#) (all packages that start with android.support)

# What's Next?

- Concept Chapter: [3.3 The Android Support Library](#)
- Practical: [3.3 Support libraries](#)

# END

# User Interaction

## Lesson 4



Buttons and  
clickable images

This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# 4.1 Buttons and clickable images

# Contents

- User interaction
- Buttons
- Clickable images
- Floating action button
- Common gestures

# User interaction



# Users expect to interact with apps

- Tapping or clicking, typing, using gestures, and talking
- Buttons perform actions
- Other UI elements enable data input and navigation

# User interaction design

Important to be obvious, easy, and consistent:

- Think about how users will use your app
- Minimize steps
- Use UI elements that are easy to access, understand, use
- Follow Android best practices
- Meet user's expectations

# Buttons

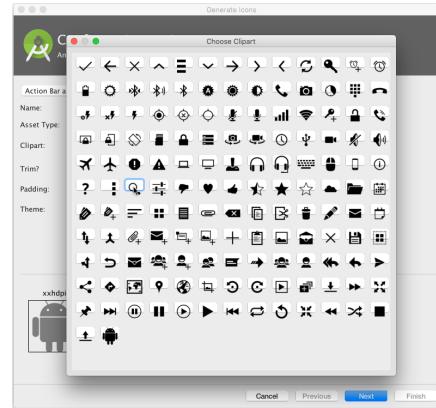
# Button

- View that responds to tapping (clicking) or pressing
- Usually text or visuals indicate what will happen when tapped
- State: normal, focused, disabled, pressed, on/off



# Button image assets

1. Right-click app/res/drawable
2. Choose **New > Image Asset**
3. Choose **Action Bar and Tab Items** from drop down menu
4. Click the **Clipart:** image (the Android logo)



Experiment:  
2. Choose **New > Vector Asset**

# Responding to button taps

- *In your code:* Use OnClickListener event listener.
- *In XML:* use android:onClick attribute in the XML layout:

```
<Button  
    android:id="@+id/button_send"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_send"  
    android:onClick="sendMessage" />
```

android:onClick

# Setting listener with onClick callback

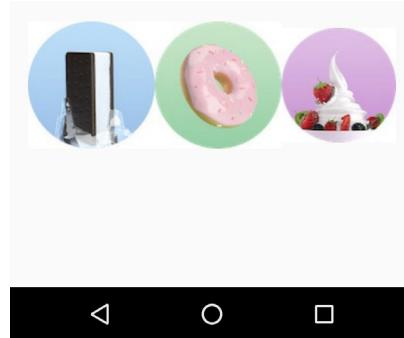
```
Button button = findViewById(R.id.button);

button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

# Clickable images

# ImageView

- ImageView with android:onClick attribute
- Image for ImageView in **app>src>main>res>drawable** folder in project



# Responding to ImageView taps

- *In your code:* Use OnClickListener event listener.
- *In XML:* use android:onClick attribute in the XML layout:

```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/donut_circle"  
    android:onClick="orderDonut"/>
```

android:onClick

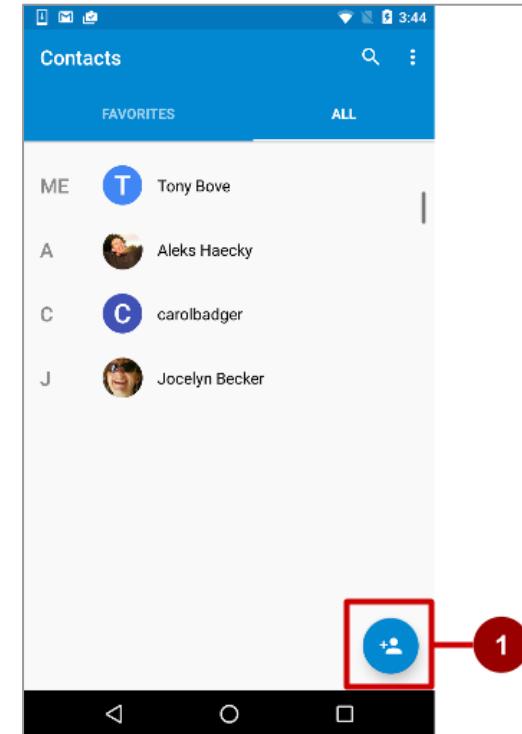
# Floating action button

# Floating Action Buttons (FAB)

- Raised, circular, floats above layout
- Primary or "promoted" action for a screen
- One per screen

For example:

Add Contact button in Contacts app



# Using FABs

- Start with Basic Activity template
- Layout:

```
<android.support.design.widget.FloatingActionButton  
    android:id="@+id/fab"  
    android:layout_gravity="bottom|end"  
    android:layout_margin="@dimen/fab_margin"  
    android:src="@drawable/ic_fab_chat_button_white"  
    .../>
```

# FAB size

- 56 x 56 dp by default
- Set mini size (30 x 40 dp) with app:fabSize attribute:
  - app:fabSize="mini"
- Set to 56 x 56 dp (default):
  - app:fabSize="normal"

# Common Gestures

# Touch Gestures

Touch gestures include:

- long touch
- double-tap
- fling
- drag
- scroll
- pinch

Don't depend on touch gestures for app's basic behavior!

# Detect gestures

Classes and methods are available to help you handle gestures.

- [GestureDetectorCompat](#) class for common gestures
- [MotionEvent](#) class for motion events

# Detecting all types of gestures

1. Gather data about touch events.
2. Interpret the data to see if it meets the criteria for any of the gestures your app supports.

Read more about how to handle gestures in the  
[Android developer documentation](#)

# Learn more

- [Input Controls](#)
- [Drawable Resources](#)
- [Floating Action Button](#)
- [Radio Buttons](#)
- [Specifying the Input Method Type](#)
- [Handling Keyboard Input](#)
- [Text Fields](#)
- [Buttons](#)
- [Spinners](#)
- [Dialogs](#)
- [Fragments](#)
- [Input Events](#)
- [Pickers](#)
- [Using Touch Gestures](#)
- [Gestures design guide](#)

# What's Next?

- Concept Chapter: [4.1 Buttons and clickable images](#)
- Practical: [4.1 Clickable images](#)

# END

# User Interaction

## Lesson 4



This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# 4.2 Input Controls

# Contents

- Overview of input controls
- View focus
- Freeform text and numbers
- Providing choices



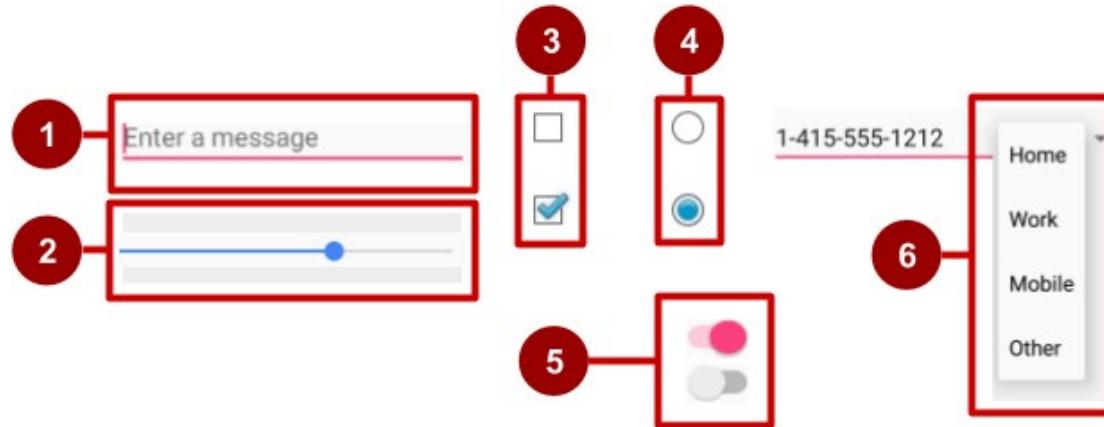
# Overview of input Controls

# Accepting user input

- Freeform text and numbers: `EditText` (using keyboard)
- Providing choices: `CheckBox`, `RadioButton`, `Spinner`
- Switching on/off: `Toggle`, `Switch`
- Choosing value in range of values: `SeekBar`

# Examples of input controls

1. EditText
2. SeekBar
3. CheckBox
4. RadioButton
5. Switch
6. Spinner



# How input controls work

1. Use `EditText` for entering text using keyboard
2. Use `SeekBar` for sliding left or right to a setting
3. Combine `CheckBox` elements for choosing more than one option
4. Combine `RadioButton` elements into [RadioGroup](#) – user makes only one choice
5. Use `Switch` for tapping on or off
6. Use `Spinner` for choosing a single item from a list

# View is base class for input controls

- The [View](#) class is the basic building block for all UI components, including input controls
- View is the base class for classes that provide interactive UI components
- View provides basic interaction through `android:onClick`

# View focus

# Focus

- The View that receives user input has "Focus"
- Only one View can have focus
- Focus makes it unambiguous which View gets the input
- Focus is assigned by
  - User tapping a View
  - App guiding the user from one text input control to the next using the **Return**, **Tab**, or arrow keys
  - Calling `requestFocus()` on any View that is focusable

# Clickable versus focusable

**Clickable**—View can respond to being clicked or tapped

**Focusable**—View can gain focus to accept input

Input controls such as keyboards send input to the view that has focus

# Which View gets focus next?

- Topmost view under the touch
- After user submits input, focus moves to nearest neighbor—priority is left to right, top to bottom
- Focus can change when user interacts with a directional control

# Guiding users

- Visually indicate which view has focus so users knows where their input goes
- Visually indicate which views can have focus helps users navigate through flow
- Predictable and logical—no surprises!

# Guiding focus

- Arrange input controls in a layout from left to right and top to bottom in the order you want focus assigned
- Place input controls inside a view group in your layout
- Specify ordering in XML

```
    android:id="@+id/top"
```

```
    android:focusable="true"
```

```
    android:nextFocusDown="@+id/bottom"
```



# Set focus explicitly

Use methods of the [View](#) class to set focus

- [setFocusable\(\)](#) sets whether a view can have focus
- [requestFocus\(\)](#) gives focus to a specific view
- [setOnFocusChangeListener\(\)](#) sets listener for when view gains or loses focus
- [onFocusChanged\(\)](#) called when focus on a view changes

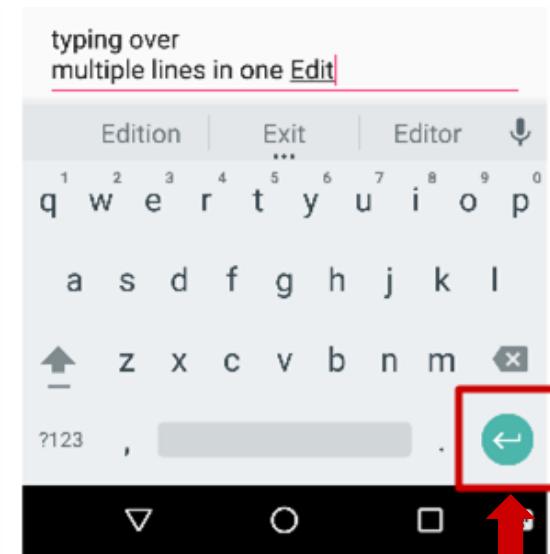
# Find the view with focus

- [Activity.getCurrentFocus\(\)](#)
- [ViewGroup.getFocusedChild\(\)](#)

# Freeform text and numbers

# EditText for multiple lines of text

- EditText default
- Alphanumeric keyboard
- Suggestions appear
- Tapping Return (Enter) key starts new line

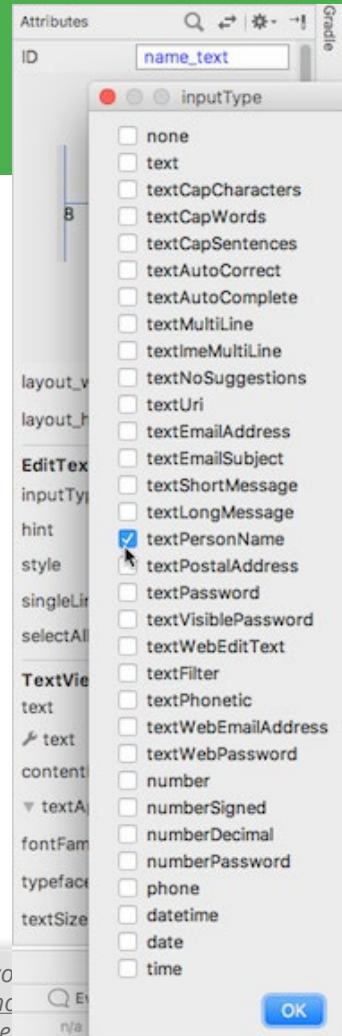


Return key

# Customize with inputType

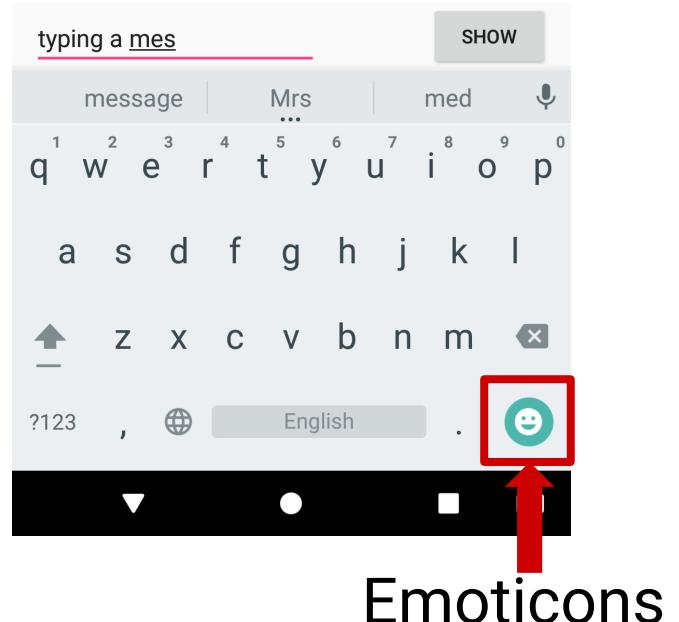
- Set in Attributes pane of layout editor
- XML code for EditText:

```
<EditText  
    android:id="@+id/name_field"  
    android:inputType =  
        "textPersonName"  
    ...
```



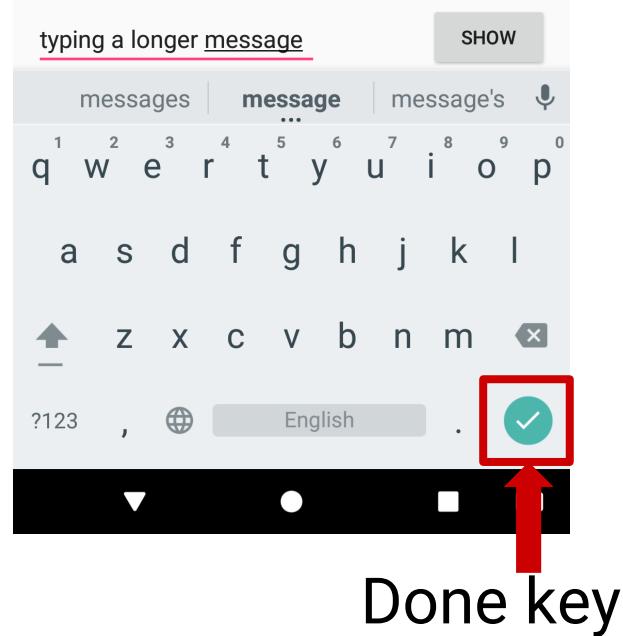
# EditText for message

- `android:inputType  
= "textShortMessage"`
- Single line of text
- Tapping Emoticons key changes keyboard to emoticons



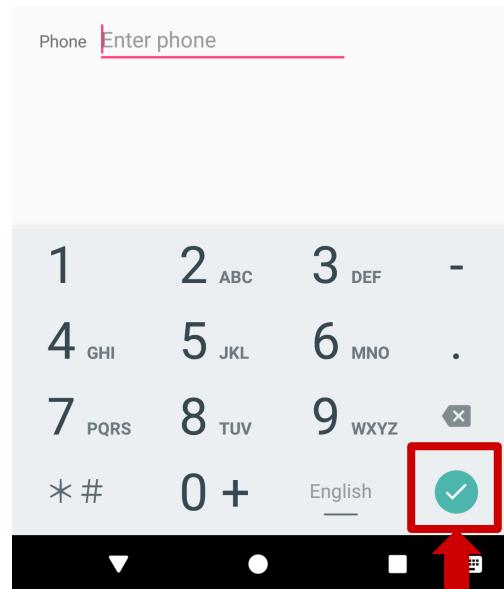
# EditText for single line

- Both work:
  - `android:inputType = "textLongMessage"`
  - `android:inputType = "textPersonName"`
- Single line of text
- Tapping Done key advances focus to next View



# EditText for phone number entry

- `android:inputType = "phone"`
- Numeric keypad (numbers only)
- Tapping Done key advances focus to next View



Done key

# Getting text

- Get the EditText object for the EditText view

```
EditText simpleEditText =  
    findViewById(R.id.edit_simple);
```

- Retrieve the CharSequence and convert it to a string

```
String strValue =  
    simpleEditText.getText().toString();
```



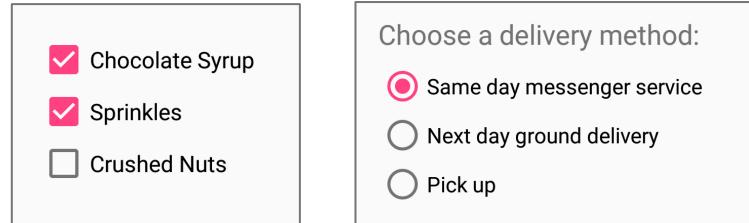
# Common input types

- `textCapCharacters`: Set to all capital letters
- `textCapSentences`: Start each sentence with a capital letter
- `textPassword`: Conceal an entered password
- `number`: Restrict text entry to numbers
- `textEmailAddress`: Show keyboard with @ conveniently located
- `phone`: Show a numeric phone keypad
- `datetime`: Show a numeric keypad with a slash and colon for entering the date and time

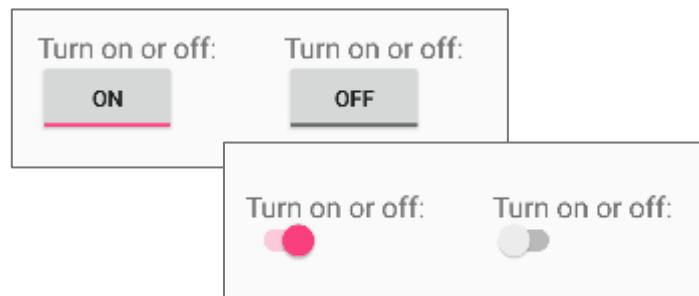
# Providing choices

# UI elements for providing choices

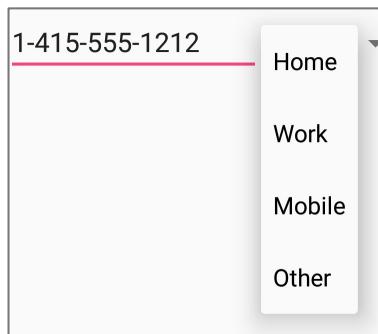
- CheckBox and RadioButton



- ToggleButton and Switch



- Spinner

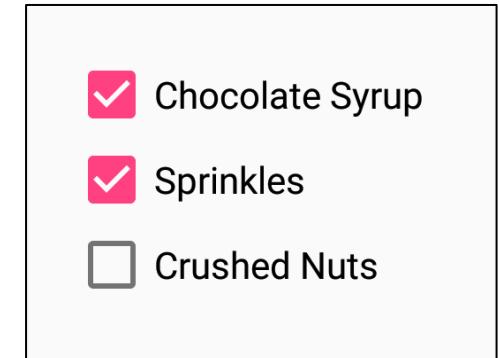


This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# CheckBox

- User can select any number of choices
- Checking one box does not uncheck another
- Users expect checkboxes in a vertical list
- Commonly used with a **Submit** button
- Every CheckBox is a View and can have an **onClick** handler



# RadioButton

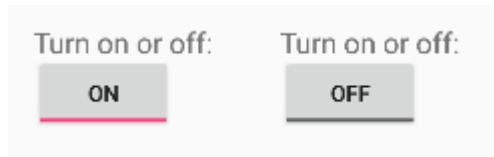
- Put RadioButton elements in a RadioGroup in a vertical list (horizontally if labels are short)
- User can select only one of the choices
- Checking one unchecks all others in group
- Each RadioButton can have onClick handler
- Commonly used with a **Submit** button for the RadioGroup

Choose a delivery method:

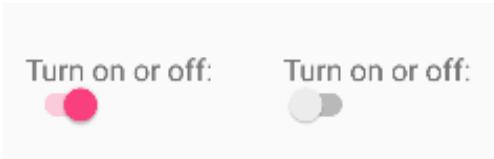
- Same day messenger service
- Next day ground delivery
- Pick up

# Toggle buttons and switches

- User can switch between on and off
- Use android:onClick for click handler



## Toggle buttons



## Switches

# Learn more

- [Input Controls](#)
- [Radio Buttons](#)
- [Specifying the Input Method Type](#)
- [Handling Keyboard Input](#)
- [Text Fields](#)
- [Spinners](#)

# What's Next?

- Concept Chapter: [4.2 Input controls](#)
- Practical: [4.2 Input controls](#)



# END

# User Interaction

## Lesson 4



# 4.3 Menus and pickers



# Contents

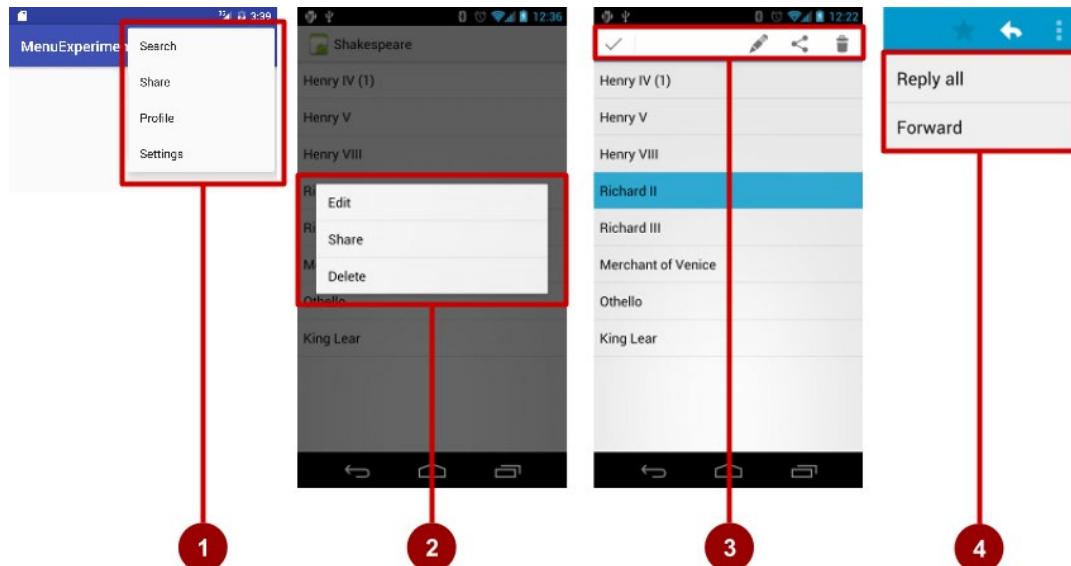
- Overview
- App Bar with Options Menu
- Contextual menus
- Popup menus
- Dialogs
- Pickers



# Overview

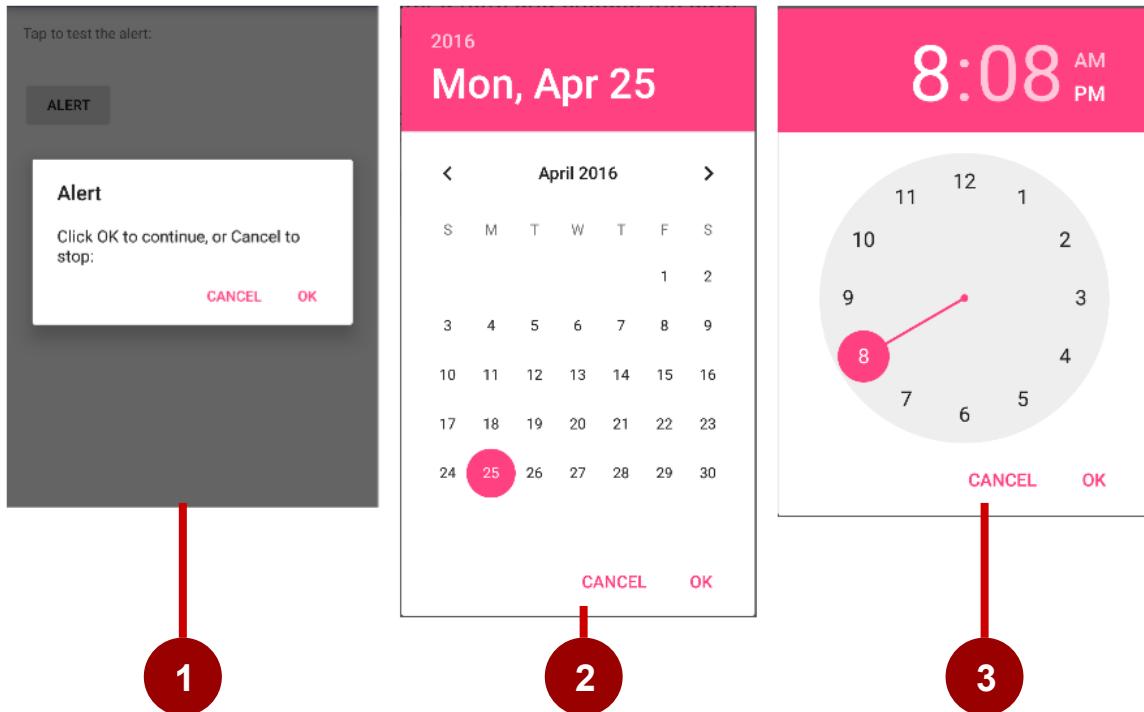
# Types of Menus

1. App bar with options menu
2. Context menu
3. Contextual action bar
4. Popup menu



# Dialogs and pickers

1. Alert dialog
2. Date picker
3. Time picker

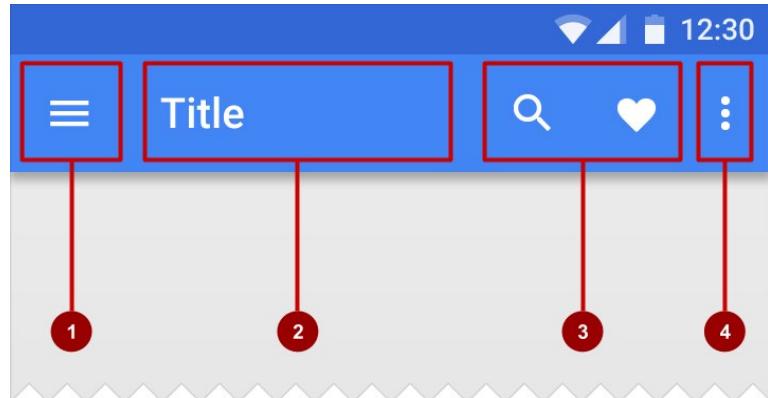


# App Bar with Options Menu

# What is the App Bar?

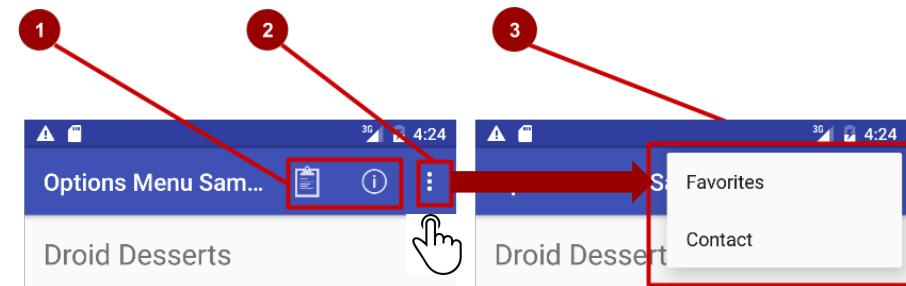
Bar at top of each screen—same for all devices (usually)

1. Nav icon to open navigation drawer
2. Title of current Activity
3. Icons for options menu items
4. Action overflow button for the rest of the options menu



# What is the options menu?

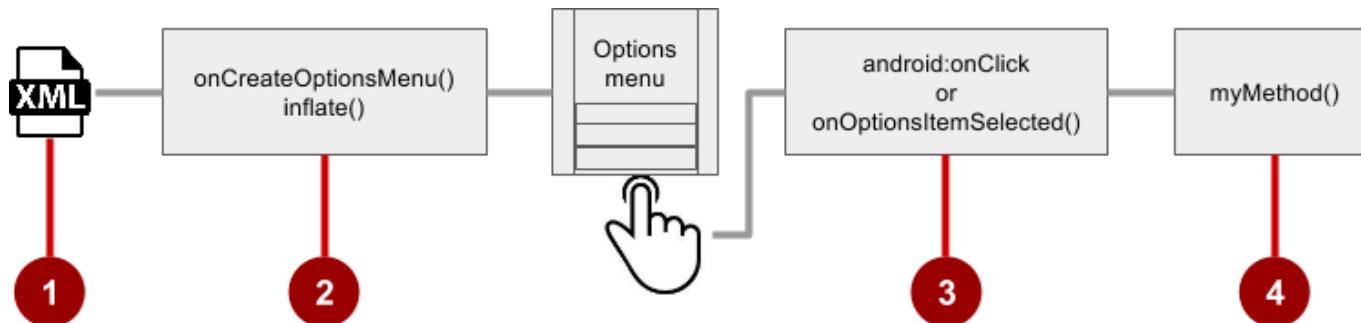
- Action icons in the app bar for important items (1)
- Tap the three dots, the "action overflow button" to see the options menu (2)
- Appears in the right corner of the app bar (3)
- For navigating to other activities and editing app settings



# Adding Options Menu

# Steps to implement options menu

1. XML menu resource (`menu_main.xml`)
2. `onCreateOptionsMenu()` to inflate the menu
3. `onClick` attribute or `onOptionsItemSelected()`
4. Method to handle item click



# Create menu resource

1. Create menu resource directory
2. Create XML menu resource (`menu_main.xml`)
3. Add entry for each menu item (**Settings** and **Favorites**):

```
<item android:id="@+id/option_settings"
      android:title="Settings" />

<item android:id="@+id/option_favorites"
      android:title="Favorites" />
```

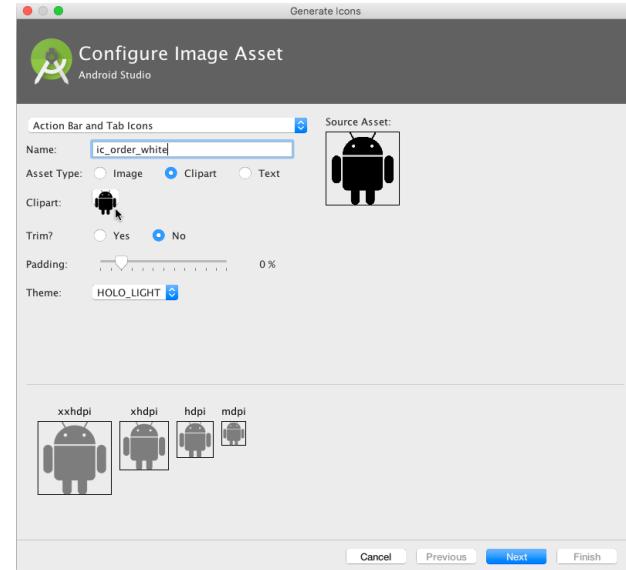
# Inflate options menu

Override onCreateOptionsMenu() in Activity

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

# Add icons for menu items

1. Right-click drawable
2. Choose New > Image Asset
3. Choose Action Bar and Tab Items
4. Edit the icon name
5. Click clipart image, and click icon
6. Click Next, then Finish



# Add menu item attributes

```
<item  
    android:id="@+id/action_favorites"  
    android:icon="@drawable/ic_favorite"  
    android:orderInCategory="30"  
    android:title="@string/action_favorites"  
    app:showAsAction="ifRoom" />
```

# Override onOptionsItemSelected()

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.action_settings:  
            showSettings();  
            return true;  
        case R.id.action_favorites:  
            showFavorites();  
            return true;  
        default:  
            return super.onOptionsItemSelected(item);  
    }  
}
```

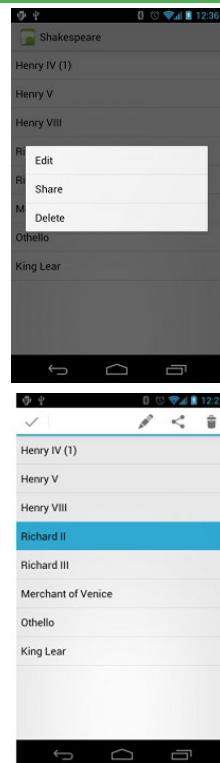


# Contextual Menus

# What are contextual menus?

- Allows users to perform action on selected View
- Can be deployed on any View
- Most often used for items in RecyclerView, GridView, or other View collection

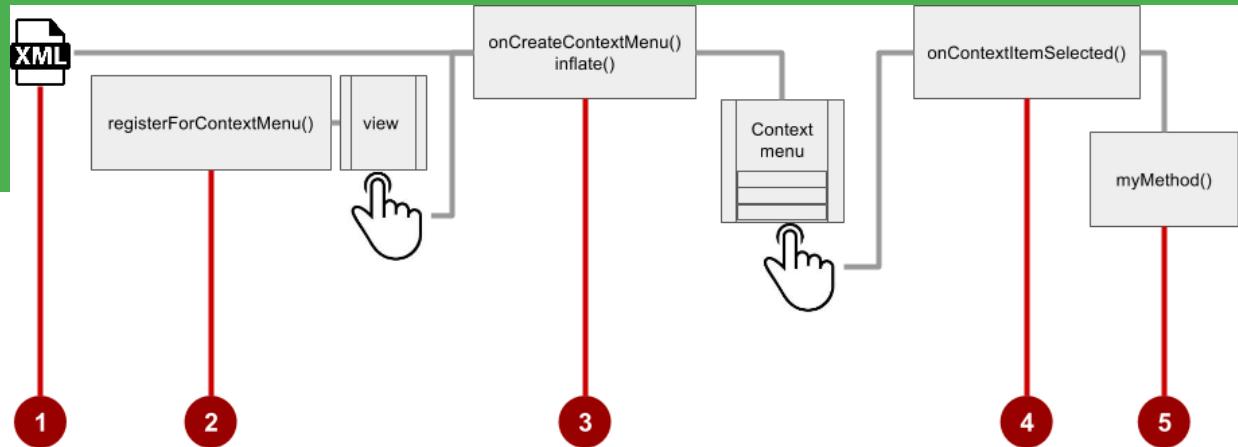
# Types of contextual menus



- Floating context menu—long-press on a View
  - User can modify View or use it in some fashion
  - User performs action on one View at a time
- Contextual action mode—temporary action bar in place of or underneath app bar
  - Action items affect the selected View element(s)
  - User can perform action on multiple View elements at once

# Floating Context Menu

# Steps



1. Create XML menu resource file and assign appearance and position attributes
2. Register View using `registerForContextMenu()`
3. Implement `onCreateContextMenu()` in Activity to inflate menu
4. Implement `onContextItemSelected()` to handle menu item clicks
5. Create method to perform action for each context menu item

# Create menu resource

## 1. Create XML menu resource (`menu_context.xml`)

```
<item  
    android:id="@+id/context_edit"  
    android:title="Edit"  
    android:orderInCategory="10"/>
```

```
<item  
    android:id="@+id/context_share"  
    android:title="Share"  
    android:orderInCategory="20"/>
```

# Register a view to a context menu

In onCreate() of the Activity:

2. Register [View.OnCreateContextMenuListener](#) to View:

```
TextView article_text = findViewById(R.id.article);  
registerForContextMenu(article_text);
```

# Implement onCreateContextMenu()

## 3. Specify which context menu

```
@Override  
public void onCreateContextMenu(ContextMenu menu, View v,  
                               ContextMenu.ContextMenuItemInfo menuInfo) {  
    super.onCreateContextMenu(menu, v, menuInfo);  
    MenuInflater inflater = getMenuInflater();  
    inflater.inflate(R.menu.menu_context, menu);  
}
```

# Implement onContextItemSelected()

```
@Override  
public boolean onContextItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.context_edit:  
            editNote();  
            return true;  
        case R.id.context_share:  
            shareNote();  
            return true;  
        default:  
            return super.onContextItemSelected(item);  
    }  
}
```



# Contextual Action Bar

# What is Action Mode?

- UI mode that lets you replace parts of normal UI interactions temporarily
- For example: Selecting a section of text or long-pressing an item could trigger action mode

# Action mode has a lifecycle

- Start it with `startActionMode()`, for example, in the listener
- `ActionMode.Callback` interface provides lifecycle methods you override:
  - `onCreateActionMode(ActionMode, Menu)` once on initial creation
  - `onPrepareActionMode(ActionMode, Menu)` after creation and any time `ActionMode` is invalidated
  - `onActionItemClicked(ActionMode, MenuItem)` any time contextual action button is clicked
  - `onDestroyActionMode(ActionMode)` when action mode is closed

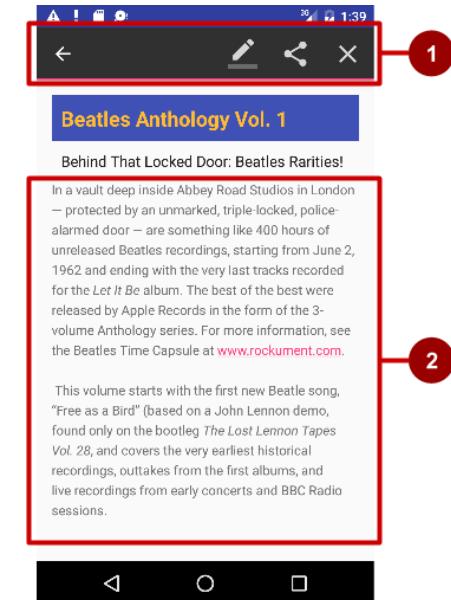
# What is a contextual action bar?

Long-press on View shows contextual action bar

## 1. Contextual action bar with actions

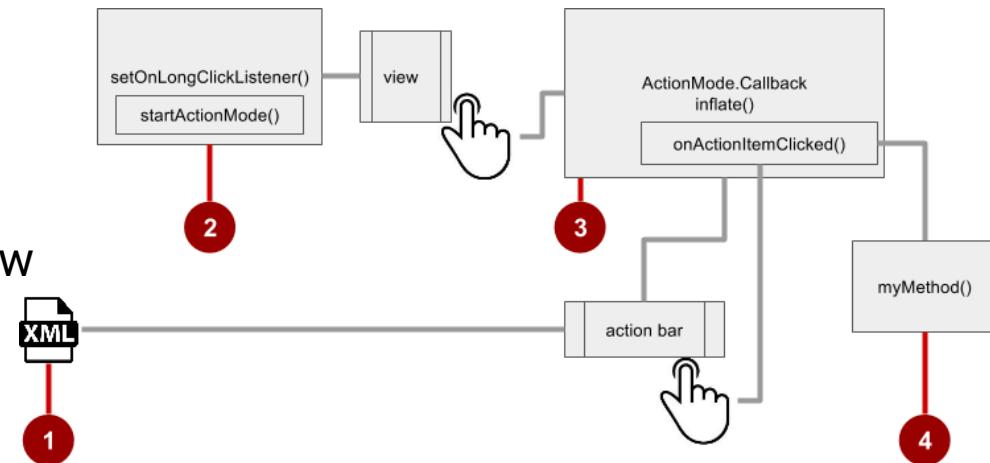
- **Edit, Share, and Delete**
- Done (left arrow icon) on left side
- Action bar is available until user taps Done

## 2. View on which long press triggers contextual action bar



# Steps for contextual action bar

1. Create XML menu resource file and assign icons for items
2. setOnLongClickListener() on View that triggers contextual action bar and call startActionMode() to handle click
3. Implement ActionMode.Callback interface to handle ActionMode lifecycle; include action for menu item click in onActionItemClicked() callback
4. Create method to perform action for each context menu item



# Use setOnLongClickListener

```
private ActionMode mActionMode;
```

In onCreate():

```
View view = findViewById(article);
view.setOnLongClickListener(new View.OnLongClickListener() {
    public boolean onLongClick(View view) {
        if (mActionMode != null) return false;
        mActionMode =
            MainActivity.this.startActionMode(mActionModeCallback);
        view.setSelected(true);
        return true;
    }
});
```

# Implement mActionModeCallback

```
public ActionMode.Callback mActionModeCallback =  
    new ActionMode.Callback() {  
        // Implement action mode callbacks here.  
   };
```



# Implement onCreateActionMode

```
@Override  
public boolean onCreateActionMode(ActionMode mode, Menu menu) {  
    MenuInflater inflater = mode.getMenuInflater();  
    inflater.inflate(R.menu.menu_context, menu);  
    return true;  
}
```

# Implement onPrepareActionMode

- Called each time action mode is shown
- Always called after onCreateActionMode, but may be called multiple times if action mode is invalidated

```
@Override  
public boolean onPrepareActionMode(ActionMode mode, Menu menu) {  
    return false; // Return false if nothing is done.  
}
```

# Implement onOptionsItemSelected

- Called when users selects an action
- Handle clicks in this method

```
@Override  
public boolean onOptionsItemSelected(ActionMode mode, MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.action_share:  
            // Perform action for the Share menu item.  
            mode.finish(); // Action picked, so close the action bar.  
            return true;  
        default:  
            return false;  
    }  
}
```

# Implement onDestroyActionMode

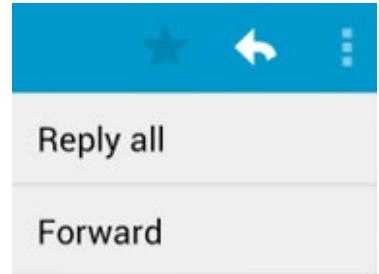
- Called when user exits the action mode

```
@Override  
public void onDestroyActionMode(ActionMode mode) {  
    mActionMode = null;  
}
```

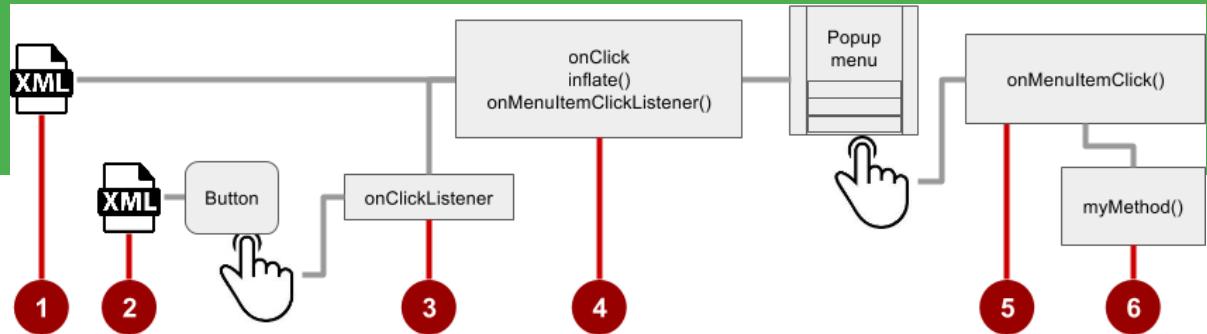
# Popup Menu

# What is a popup menu?

- Vertical list of items anchored to a view
- Typically anchored to a visible icon
- Actions should not directly affect view content
  - Options menu overflow icon that opens options menu
  - In email app, **Reply All** and **Forward** relate to email message but don't affect or act on message



# Steps



1. Create XML menu resource file and assign appearance and position attributes
2. Add ImageButton for the popup menu icon in the XML activity layout file
3. Assign onClickListener to ImageButton
4. Override onClick() to inflate the popup and register it with onMenuItemClickListener()
5. Implement onMenuItemClick()
6. Create a method to perform an action for each popup menu item

# Add ImageButton

⋮

```
<ImageButton  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:id="@+id/button_popup"  
    android:src="@drawable/@drawable/ic_action_popup"/>
```

# Assign onClickListener to button

```
private ImageButton mButton =  
    (ImageButton) findViewById(R.id.button_popup);
```

In onCreate():

```
mButton.setOnClickListener(new View.OnClickListener() {  
    // define onClick  
});
```

# Implement onClick

```
@Override  
public void onClick(View v) {  
    PopupMenu popup = new PopupMenu(MainActivity.this, mButton);  
    popup.getMenuInflater().inflate(  
        R.menu.menu_popup, popup.getMenu());  
    popup.setOnMenuItemClickListener(  
        new PopupMenu.OnMenuItemClickListener() {  
            // implement click listener.  
        });  
    popup.show();  
}
```



# Implement onOptionsItemSelected

```
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.option_forward:  
            // Implement code for Forward button.  
            return true;  
        default:  
            return false;  
    }  
}
```

# Dialogs

# Dialogs

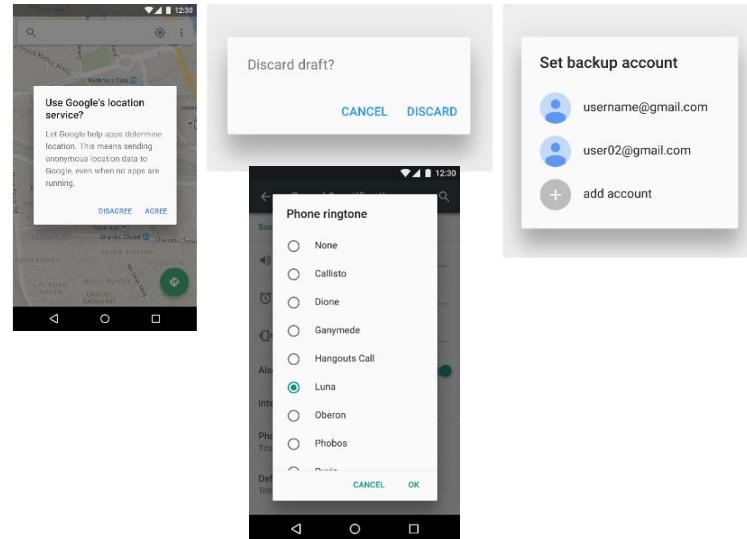
- Dialog appears on top, interrupting flow of Activity
- Requires user action to dismiss



TimePickerDialog



DatePickerDialog



AlertDialog

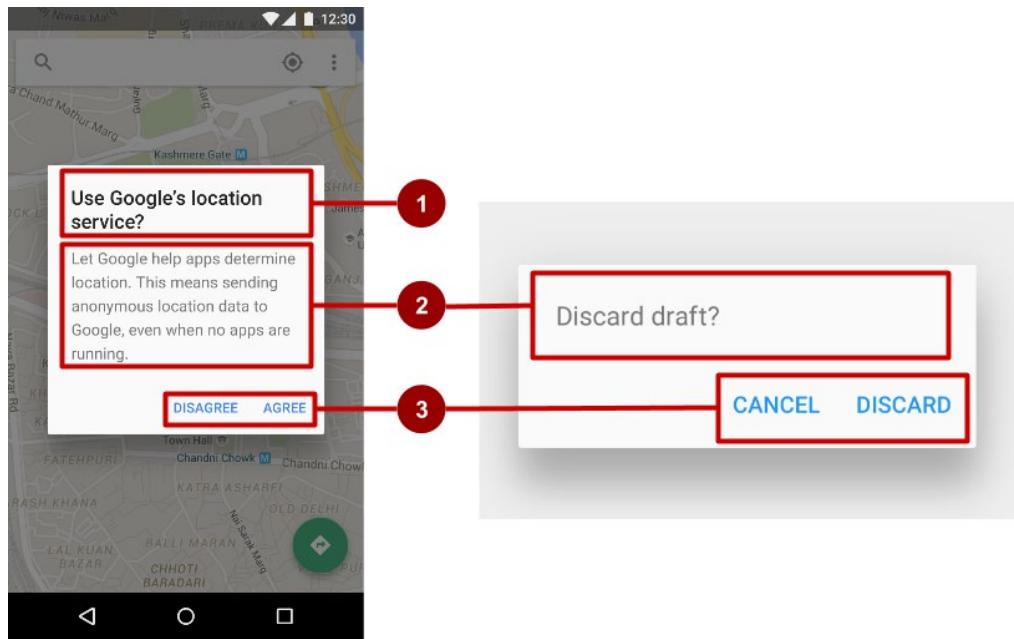
Menus and  
pickers

This work is licensed under a [Creative Commons Attribution 4.0 International License](#).

# AlertDialog

AlertDialog can show:

1. Title (optional)
2. Content area
3. Action buttons



# Build the AlertDialog

Use `AlertDialog.Builder` to build alert dialog and set attributes:

```
public void onClickShowAlert(View view) {  
    AlertDialog.Builder alertDialog = new  
        AlertDialog.Builder(MainActivity.this);  
    alertDialog.setTitle("Connect to Provider");  
    alertDialog.setMessage(R.string.alert_message);  
    // ... Code to set buttons goes here.  
}
```

# Set the button actions

- `AlertDialog.setPositiveButton()`
- `AlertDialog.setNeutralButton()`
- `AlertDialog.setNegativeButton()`

# AlertDialog code example

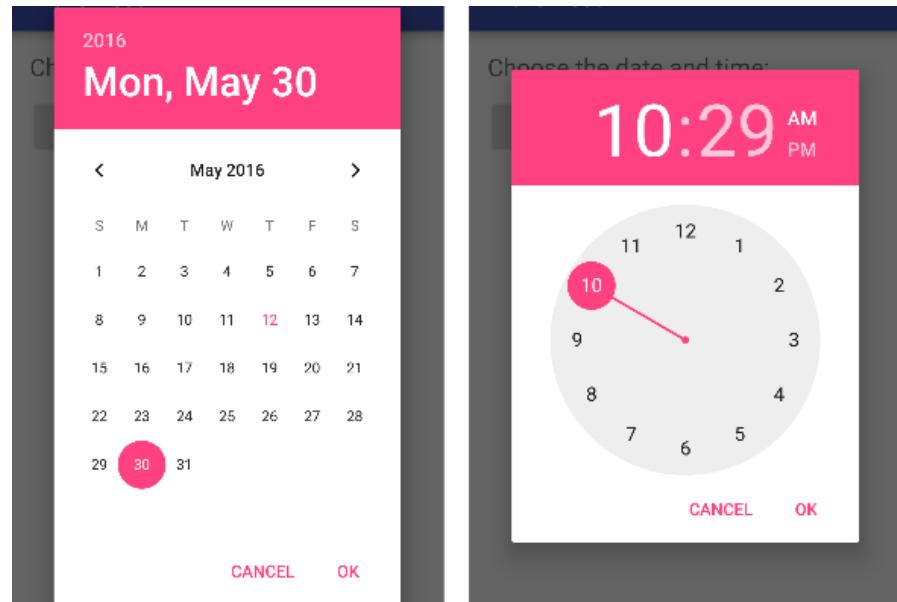
```
AlertDialog.setPositiveButton(  
        "OK", newDialogInterface.OnClickListener() {  
    public void onClick(DialogInterface dialog, int which) {  
        // User clicked OK button.  
    }  
});
```

Same pattern for setNegativeButton() and setNeutralButton()

# Pickers

# Pickers

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



# Pickers use fragments

- Use [DialogFragment](#) to show a picker
- DialogFragment is a window that floats on top of Activity window



# Introduction to fragments

- A Fragment is like a mini-Activity within an Activity
  - Manages its own lifecycle
  - Receives its own input events
- Can be added or removed while parent Activity is running
- Multiple fragments can be combined in a single Activity
- Can be reused in more than one Activity

# Creating a date picker dialog

1. Add a blank Fragment that extends DialogFragment and implements DatePickerDialog.OnDateSetListener
2. In onCreateDialog() initialize the date and return the dialog
3. In onDateSet() handle the date
4. In Activity show the picker and add method to use date

# Creating a time picker dialog

1. Add a blank Fragment that extends DialogFragment and implements TimePickerDialog.OnTimeSetListener
2. In onCreateDialog() initialize the time and return the dialog
3. In onTimeSet() handle the time
4. In Activity, show the picker and add method to use time

# Learn more

- [Adding the App Bar](#)
- [Menus](#)
- [Menu Resource](#)
- [Fragments](#)
- [Dialogs](#)
- [Pickers](#)
- [Drawable Resources](#)

# What's Next?

- Concept Chapter: [4.3 Menus and pickers](#)
- Practical: [4.3 Menus and pickers](#)



# END

# User Interaction

## Lesson 4



This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# 4.4 User navigation

# Contents

- Back navigation
- Hierarchical navigation
  - Up navigation
  - Descendant navigation
- Navigation drawer for descendant navigation
  - Lists and carousels for descendant navigation
  - Ancestral navigation
  - Lateral navigation



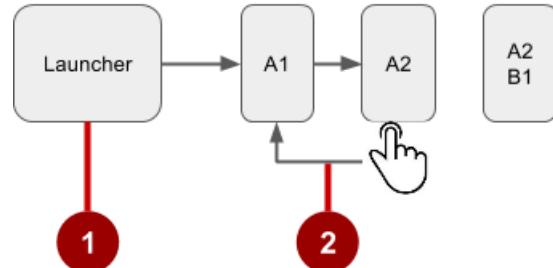
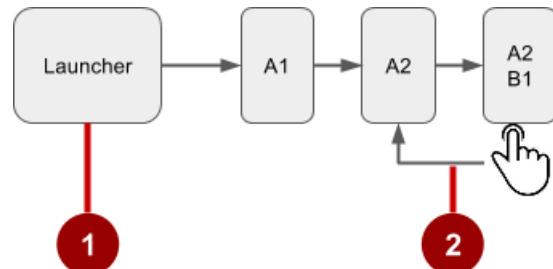
# Two forms of navigation

- ◀ Back (temporal) navigation
  - Provided by the device's Back button
  - Controlled by the Android system back stack
- ⬅ Ancestral (Up) navigation
  - Up button provided in app bar
  - Controlled by defining parent Activity for child Activity in the `AndroidManifest.xml`

# Back Navigation

# Navigation through history of screens

1. History starts from Launcher
2. User clicks the Back  button to navigate to previous screens in reverse order



# Changing Back button behavior

- Android system manages the back stack and Back button
- If in doubt, don't change
- Only override, if necessary to satisfy user expectation

For example: In an embedded browser, trigger browser's default back behavior when user presses device Back button

# Overriding onBackPressed()

```
@Override  
public void onBackPressed() {  
    // Add the Back key handler here.  
    return;  
}
```

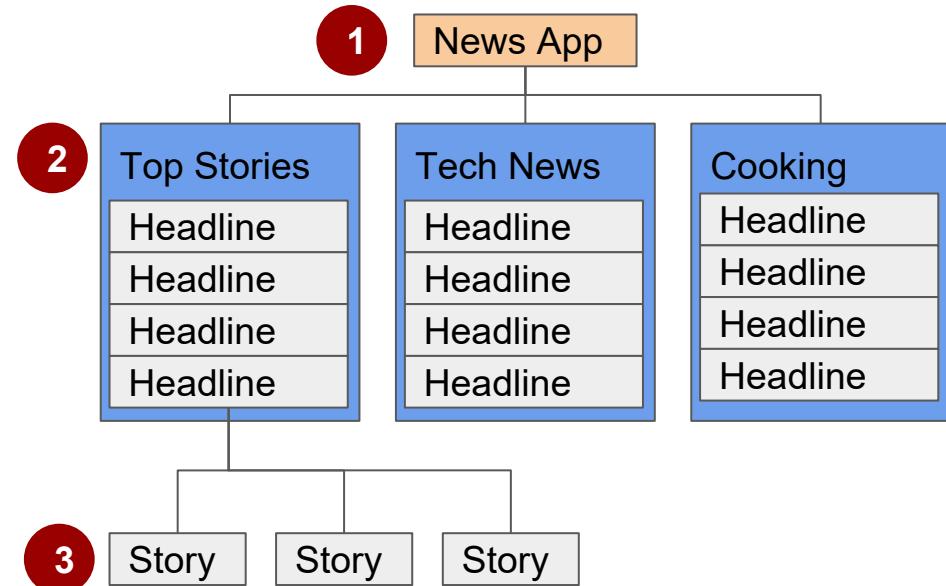
# Hierarchical Navigation

# Hierarchical navigation patterns

- **Parent screen**—Screen that enables navigation down to child screens, such as home screen and main Activity
- **Collection sibling**—Screen enabling navigation to a collection of child screens, such as a list of headlines
- **Section sibling**—Screen with content, such as a story

# Example of a screen hierarchy

1. Parent
  2. Children: collection siblings
  3. Children: section siblings
- Use Activity for parent screen
  - Use Activity or Fragment for children screens



# Types of hierarchical navigation

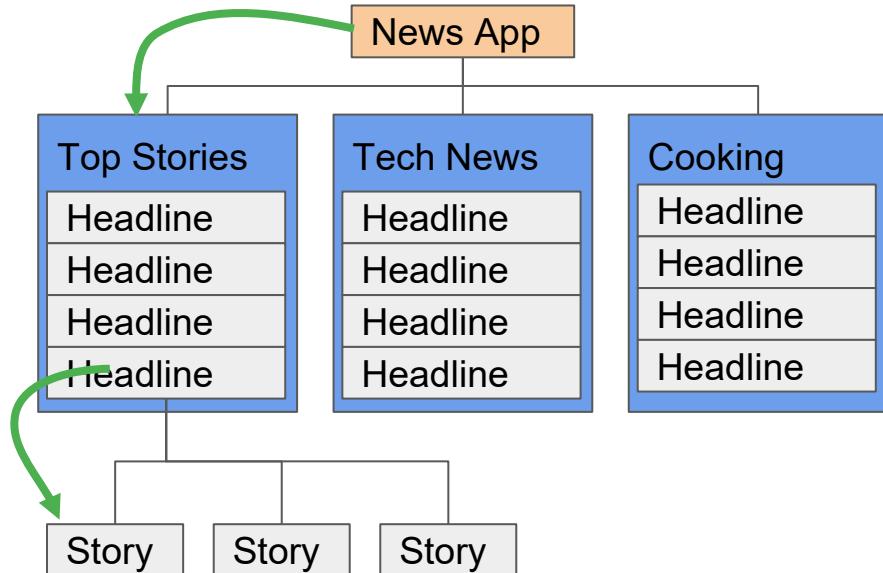
- Descendant navigation
  - Down from a parent screen to one of its children
  - From a list of headlines—to a story summary—to a story
- Ancestral navigation
  - Up from a child or sibling screen to its parent
  - From a story summary back to the headlines
- Lateral navigation
  - From one sibling to another sibling
  - Swiping between tabbed views

# Descendant Navigation

# Descendant navigation

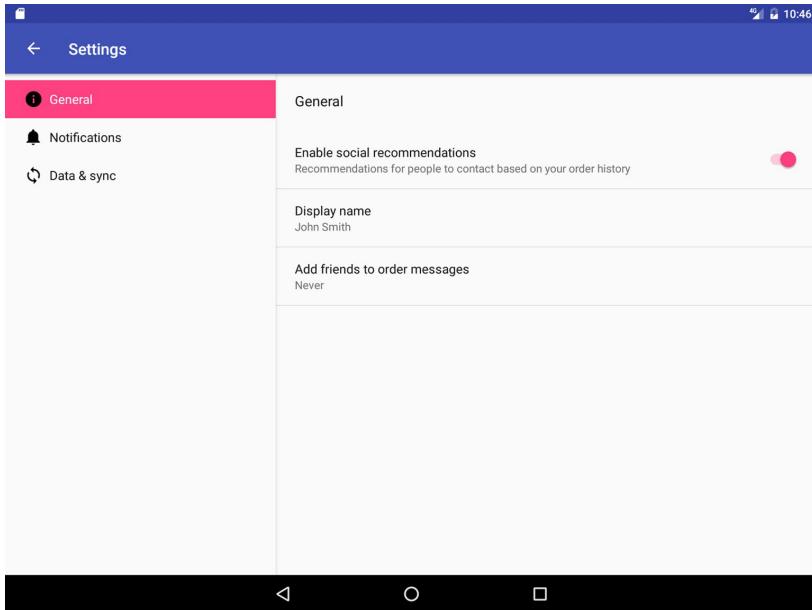
## Descendant navigation

- Down from a parent screen to one of its children
- From the main screen to a list of headlines to a story

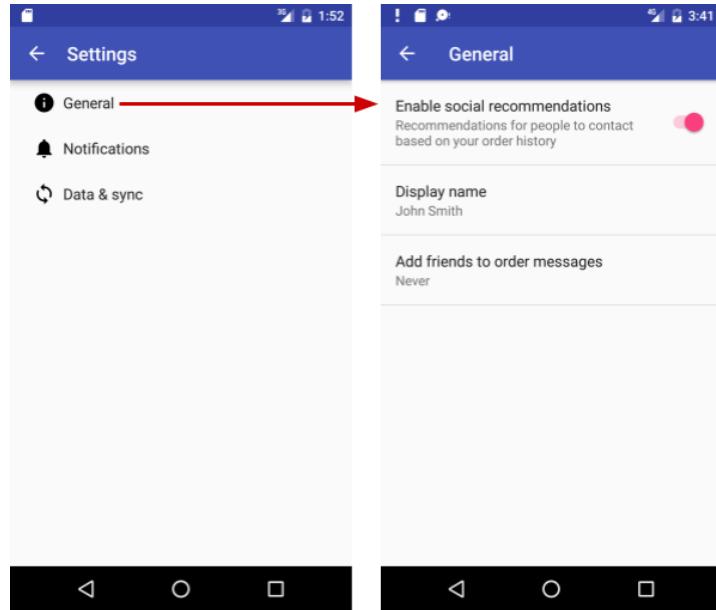


# Master/detail flow

- Side-by side on tablets



- Multiple screens on phone



# Controls for descendant navigation

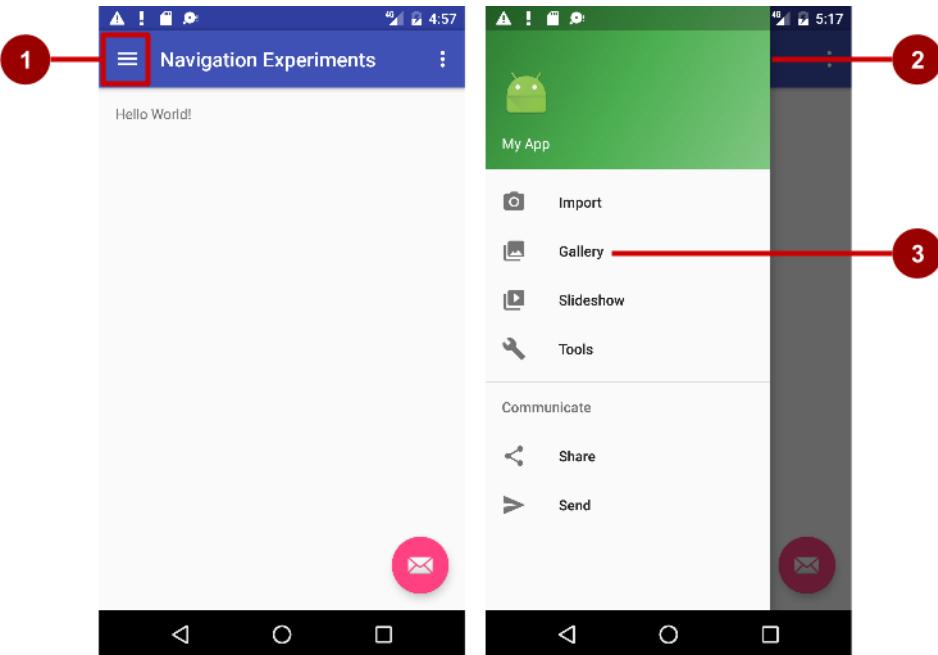
- Navigation drawer
- Buttons, image buttons on main screen
- Other clickable views with text and icons arranged in horizontal or vertical rows, or as a grid
- List items on collection screens

# Navigation Drawer

# Navigation drawer

## Descendant navigation

1. Icon in app bar
2. Header
3. Menu items



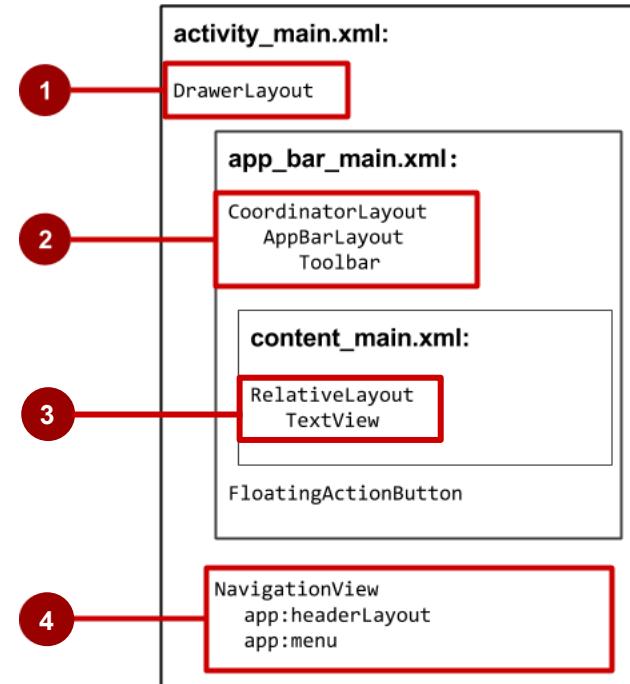
# Layouts for navigation drawer

## Create layouts:

- A navigation drawer as the Activity layout root ViewGroup
- A navigation View for the drawer itself
- An app bar layout that includes room for a navigation icon button
- A content layout for the Activity that displays the navigation drawer
- A layout for the navigation drawer header

# Navigation drawer Activity layout

1. DrawerLayout is root view
2. CoordinatorLayout contains app bar layout with a Toolbar
3. App content screen layout
4. NavigationView with layouts for header and selectable items



# Steps to implement navigation drawer

1. Populate navigation drawer menu with item titles and icons
2. Set up navigation drawer and item listeners in the Activity code
3. Handle the navigation menu item selections

# Other descendant navigation patterns

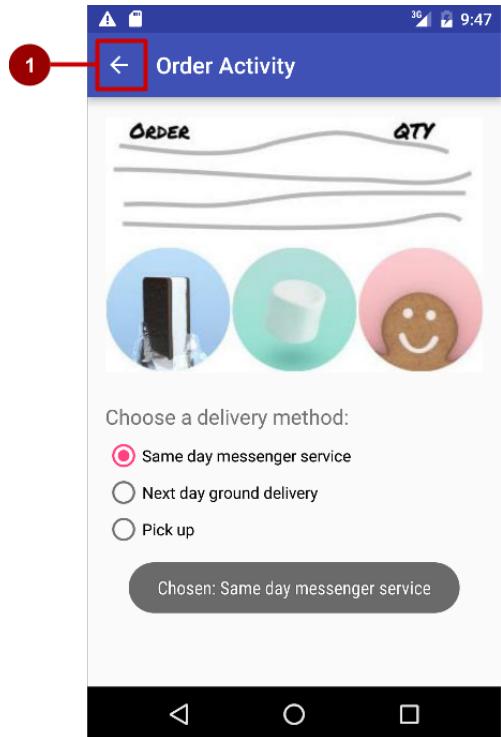
- Vertical list, such as [RecyclerView](#)
- Vertical grid, such as [GridView](#)
- Lateral navigation with a carousel
- Multi-level menus, such as the options menu
- Master/detail navigation flow

# Ancestral Navigation

# Ancestral navigation (Up button)



Enable user to go up from a section or child screen to the parent



# Declare parent of child Activity—AndroidManifest

```
<activity android:name=".OrderActivity"
    android:label="@string/title_activity_order"
    android:parentActivityName="com.example.android.
                                optionsmenuorderactivity.MainActivity">
<meta-data
    android:name="android.support.PARENT_ACTIVITY"
    android:value=".MainActivity"/>
</activity>
```

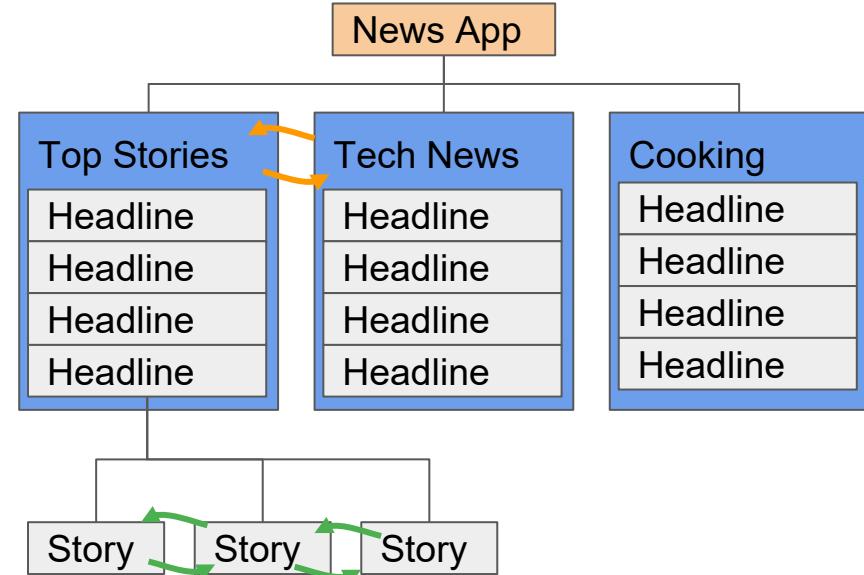


# Lateral Navigation

# Tabs and swipes

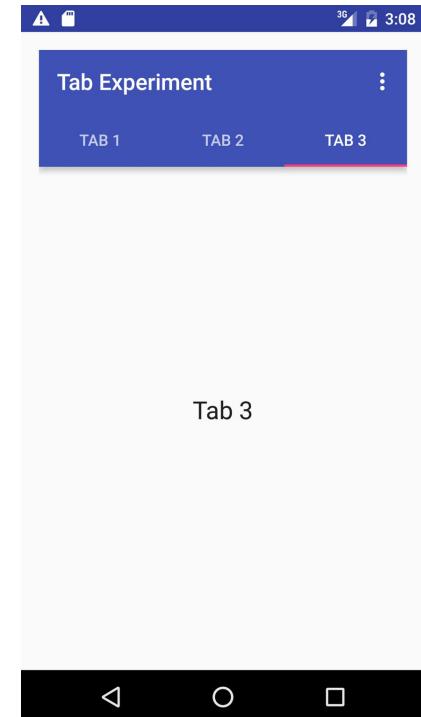
## Lateral navigation

- Between siblings
- From a list of stories to a list in a different tab
- From story to story under the same tab



# Benefits of using tabs and swipes

- A single, initially-selected tab—users have access to content without further navigation
- Navigate between related screens without visiting parent



# Best practices with tabs

- Lay out horizontally
- Run along top of screen
- Persistent across related screens
- Switching should not be treated as history



# Steps for implementing tabs

1. Define the tab layout using [TabLayout](#)
2. Implement a Fragment and its layout for each tab
3. Implement a PagerAdapter from [FragmentPagerAdapter](#) or [FragmentStatePagerAdapter](#)
4. Create an instance of the tab layout
5. Use PagerAdapter to manage screens (each screen is a Fragment)
6. Set a listener to determine which tab is tapped

# Add tab layout below Toolbar

```
<android.support.design.widget.TabLayout  
    android:id="@+id/tab_layout"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_below="@id/toolbar"  
    android:background="?attr/colorPrimary"  
    android:minHeight="?attr/actionBarSize"  
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"/>
```



# Add view pager below TabLayout

```
<android.support.v4.view.ViewPager  
    android:id="@+id/pager"  
    android:layout_width="match_parent"  
    android:layout_height="fill_parent"  
    android:layout_below="@id/tab_layout" />
```

# Create a tab layout in onCreate()

```
TabLayout tabLayout = findViewById(R.id.tab_layout);
tabLayout.addTab(tabLayout.newTab().setText("Tab 1"));
tabLayout.addTab(tabLayout.newTab().setText("Tab 2"));
tabLayout.addTab(tabLayout.newTab().setText("Tab 3"));
tabLayout.setTabGravity(TabLayout.GRAVITY_FILL);
```

# Add the view pager in onCreate()

```
final ViewPager viewPager = findViewById(R.id.pager);
final PagerAdapter adapter = new PagerAdapter (
    getSupportFragmentManager(), tabLayout.getTabCount());
viewPager.setAdapter(adapter);
```

# Add the listener in onCreate()

```
viewPager.addOnPageChangeListener(  
        new TabLayout.TabLayoutOnPageChangeListener(tabLayout));  
tabLayout.addOnTabSelectedListener(  
        new TabLayout.OnTabSelectedListener() {  
            @Override  
            public void onTabSelected(TabLayout.Tab tab) {  
                viewPager.setCurrentItem(tab.getPosition());}  
            @Override  
            public void onTabUnselected(TabLayout.Tab tab) {}  
            @Override  
            public void onTabReselected(TabLayout.Tab tab) {}});
```

# Learn more

- Navigation Design guide  
[d.android.com/design/patterns/navigation.html](https://d.android.com/design/patterns/navigation.html)
- Designing effective navigation  
[d.android.com/training/design-navigation/index.html](https://d.android.com/training/design-navigation/index.html)
- Creating a Navigation Drawer  
[d.android.com/training/implementing-navigation/nav-drawer.html](https://d.android.com/training/implementing-navigation/nav-drawer.html)
- Creating swipe views with tabs  
[d.android.com/training/implementing-navigation/lateral.html](https://d.android.com/training/implementing-navigation/lateral.html)



# What's Next?

- Concept Chapter: [4.4 User navigation](#)
- Practical: [4.4 User navigation](#)

# END

# User Interaction

## Lesson 4



This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# 4.4 RecyclerView

# Contents

- RecyclerView Components
- Implementing a RecyclerView

# What is a RecyclerView?

- RecyclerView is scrollable container for large data sets
- Efficient
  - Uses and reuses limited number of View elements
  - Updates changing data fast



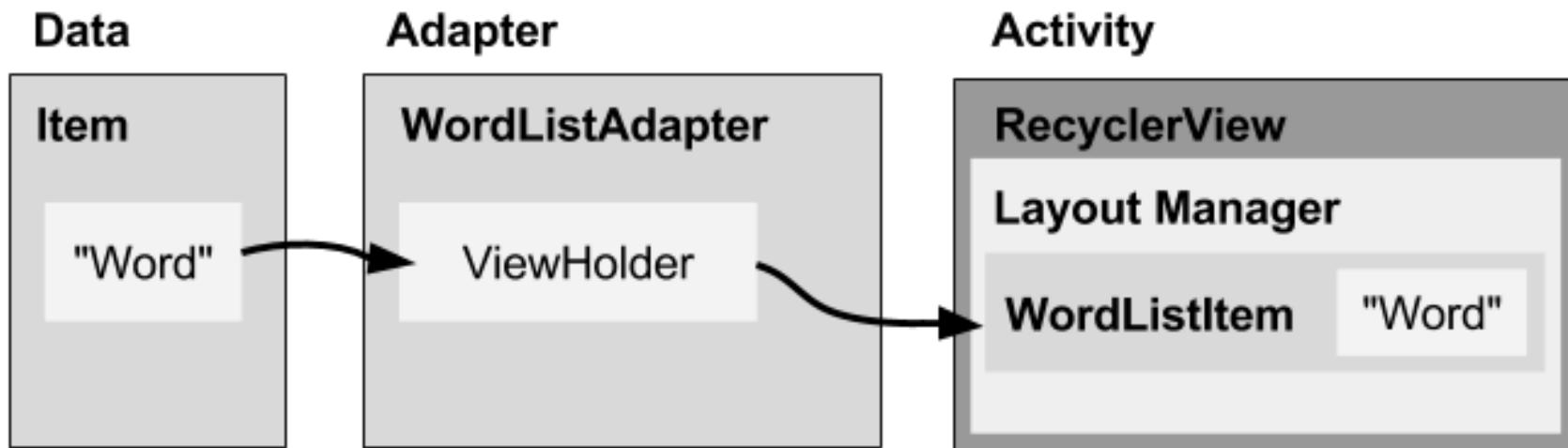
# RecyclerView Components



# Overview

- Data
- RecyclerView scrolling list for list items—[RecyclerView](#)
- Layout for one item of data—XML file
- Layout manager handles the organization of UI components in a View—[Recyclerview.LayoutManager](#)
- Adapter connects data to the RecyclerView—[RecyclerView.Adapter](#)
- ViewHolder has view information for displaying one item—[RecyclerView.ViewHolder](#)

# How components fit together overview

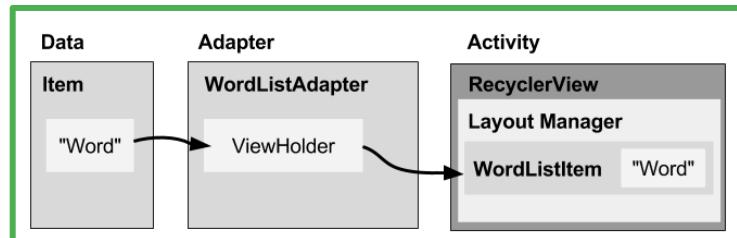


# What is a layout manager?

- Each ViewGroup has a layout manager
- Use to position View items inside a [RecyclerView](#)
- Reuses View items that are no longer visible to the user
- Built-in layout managers
  - [LinearLayoutManager](#)
  - [GridLayoutManager](#)
  - [StaggeredGridLayoutManager](#)
- Extend [RecyclerView.LayoutManager](#)

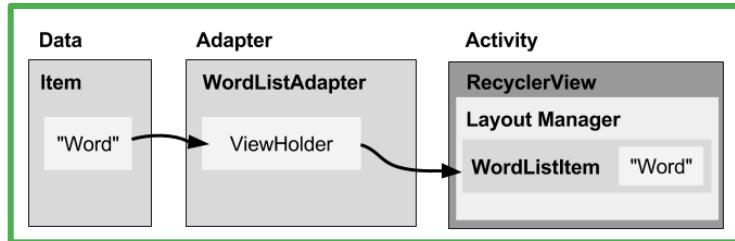
# What is an adapter?

- Helps incompatible interfaces work together
  - Example: Takes data from database Cursor and prepares strings to put into a View
- Intermediary between data and View
- Manages creating, updating, adding, deleting View items as underlying data changes
- RecyclerView.Adapter



# What is a ViewHolder?

- Used by the adapter to prepare one View with data for one list item
- Layout specified in an XML resource file
- Can have clickable elements
- Is placed by the layout manager
- [RecyclerView.ViewHolder](#)



# Implementing RecyclerView



# Steps Summary

1. Add RecyclerView dependency to build.gradle if needed
2. Add RecyclerView to layout
3. Create XML layout for item
4. Extend RecyclerView.Adapter
5. Extend RecyclerView.ViewHolder
6. In Activity onCreate(), create RecyclerView with adapter and layout manager



# Add dependency to app/build.gradle

Add RecyclerView dependency to build.gradle if needed:

```
dependencies {  
    ...  
    compile 'com.android.support:recyclerview-v7:26.1.0'  
    ...  
}
```

# Add RecyclerView to XML Layout

```
<android.support.v7.widget.RecyclerView  
    android:id="@+id/recyclerview"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
</android.support.v7.widget.RecyclerView>
```

# Create layout for 1 list item

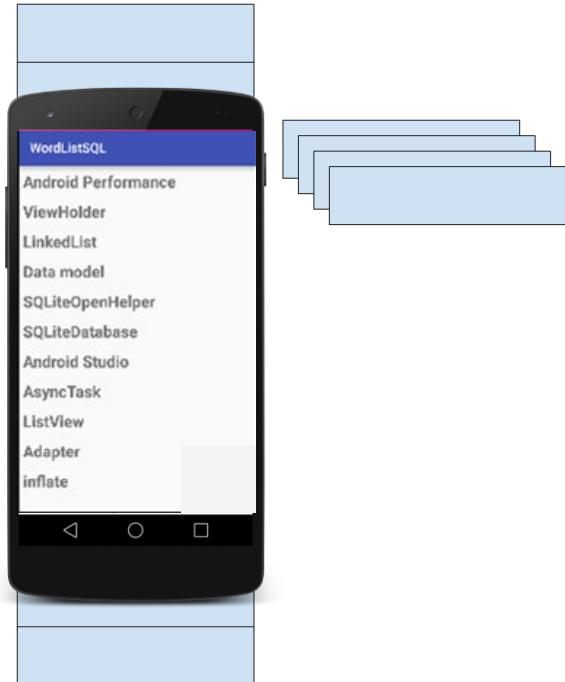
```
<LinearLayout ...>
```

```
    <TextView
```

```
        android:id="@+id/word"
```

```
        style="@style/word_title" />
```

```
</LinearLayout>
```



# Implement the adapter

```
public class WordListAdapter  
    extends RecyclerView.Adapter<WordListAdapter.WordViewHolder> {  
  
    public WordListAdapter(Context context,  
                          LinkedList<String> wordList) {  
        mInflater = LayoutInflater.from(context);  
        this.mWordList = wordList;  
    }  
}
```



# Adapter has 3 required methods

- `onCreateViewHolder()`
- `inBindViewHolder()`
- `getItemCount()`

Let's take a look!



# onCreateViewHolder()

```
@Override  
public WordViewHolder onCreateViewHolder(  
    ViewGroup parent, int viewType) {  
    // Create view from layout  
    View mItemView = mInflater.inflate(  
        R.layout.wordlist_item, parent, false);  
    return new WordViewHolder(mItemView, this);  
}
```

# onBindViewHolder()

```
@Override  
public void onBindViewHolder(  
    WordViewHolder holder, int position) {  
    // Retrieve the data for that position  
    String mCurrent = mWordList.get(position);  
    // Add the data to the view  
    holder.wordItemView.setText(mCurrent);  
}
```

# getItemCount()

```
@Override  
public int getItemCount() {  
    // Return the number of data items to display  
    return mWordList.size();  
}
```

# Create the view holder in adapter class

```
class WordViewHolder extends RecyclerView.ViewHolder { //.. }
```

If you want to handle mouse clicks:

```
class WordViewHolder extends RecyclerView.ViewHolder  
    implements View.OnClickListener { //.. }
```

# View holder constructor

```
public WordViewHolder(View itemView, WordListAdapter adapter) {  
    super(itemView);  
    // Get the layout  
    wordItemView = itemView.findViewById(R.id.word);  
    // Associate with this adapter  
    this.mAdapter = adapter;  
    // Add click listener, if desired  
    itemView.setOnClickListener(this);  
}  
// Implement onClick() if desired
```

# Create the RecyclerView in Activity onCreate()

```
mRecyclerView = findViewById(R.id.recyclerview);  
mAdapter = new WordListAdapter(this, mWordList);  
mRecyclerView.setAdapter(mAdapter);  
mRecyclerView.setLayoutManager(new  
    LinearLayoutManager(this));
```

# Practical: RecyclerView

- This is rather complex with many separate pieces. So, there is a whole practical where you implement a RecyclerView that displays a list of clickable words.
- Shows all the steps, one by one with a complete app

# Learn more

- [RecyclerView](#)
- [RecyclerView class](#)
- [RecyclerView.Adapter class](#)
- [RecyclerView.ViewHolder class](#)
- [RecyclerView.LayoutManager class](#)

# What's Next?

- Concept Chapter: [4.5 RecyclerView](#)
- Practical: [4.5 RecyclerView](#)



# END

Android Developer Fundamentals V2

# Delightful User Experience

Lesson 5



Drawables, styles  
and themes

*This work is licensed under a [Creative Commons Attribution 4.0 International License](#).*



# 5.1 Drawables, styles, and themes



# Contents

- Drawables
- Creating image assets
- Styles
- Themes



# Drawables



# Drawables

- [Drawable](#)—generic Android class used to represent any kind of graphic
- All drawables are stored in the **res/drawable** project folder



# Drawable classes

[Bitmap File](#)

[Nine-Patch File](#)

[Layer List Drawable](#)

[Shape Drawable](#)

[State List Drawable](#)

[Level List Drawable](#)

[Transition Drawable](#)

[Vector Drawable](#)

... and more

Custom Drawables



# Bitmaps

- PNG (.png), JPG (.jpg), or GIF (.gif) format
- Uncompressed BMP (.bmp)
- WebP (4.0 and higher)
- Creates a BitmapDrawable data type
- Placed directly in res/drawables



# Referencing Drawables

- XML: @[package:]drawable/filename

```
<ImageView  
    android:layout_height="wrap_content"  
    android:layout_width="wrap_content"  
    android:src="@drawable/myimage" />
```

- Java code: R.drawable.filename

```
Resources res = getResources();  
Drawable drawable = res.getDrawable(R.drawable.myimage);
```

# Nine-Patch Files

- [Nine-patch](#) files (.9.png) are PNG with stretchable regions
- Only stretches bigger, not smaller, so start with small image
- Often used for backgrounds of UI elements
- Example: button background changes size with label length
- Good [intro](#)



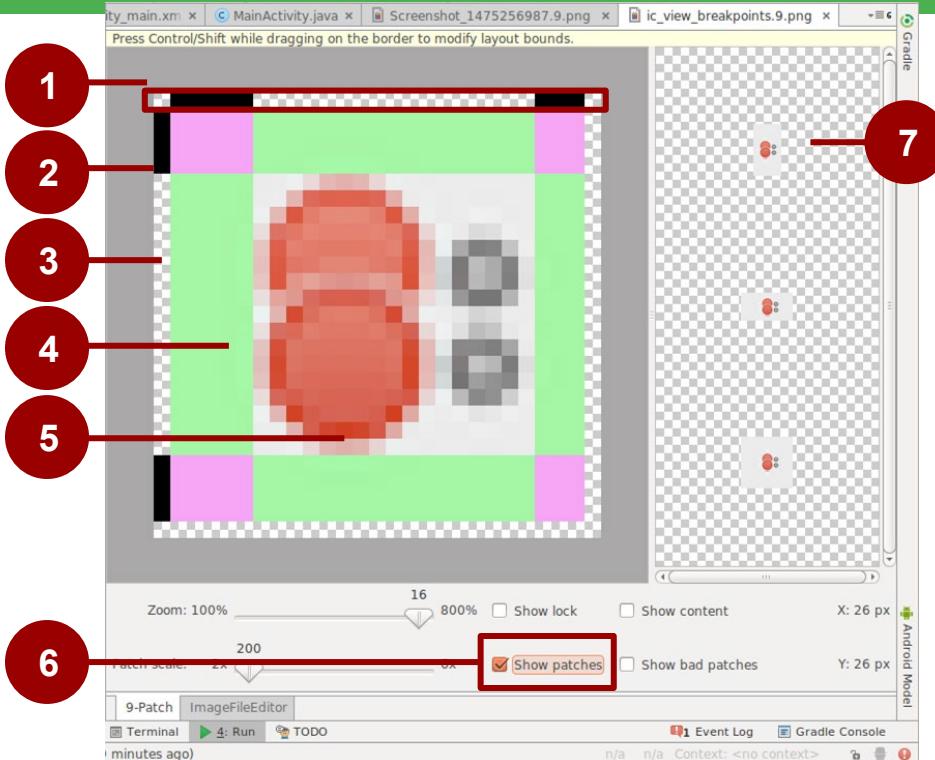
# Creating Nine-Patch Files

1. Put a small PNG file into **res/drawable**
2. Right-click and choose **Create 9-Patch file**
3. **Double-click** 9-Patch file to open editor
4. Specify the stretchable regions (next slide)



# Editing Nine-Patch Files

1. Border to mark stretchable regions for width
2. Stretchable regions marked for height  
Pink == both directions
3. Click to turn pixels black. Shift-click (ctrl-click on Mac) to unmark
4. Stretchable area
5. Not stretchable
6. Check **Show patches**
7. Preview of stretched image



# Layer List

- You can create layered images, just like with drawing tools, such as Gimp
- In Android, each layer is represented by a drawable
- Layers are organized and managed in XML
- List and the items can have properties
- Layers are drawn on top of each other in the order defined in the XML file
- [LayerDrawable](#)



# Creating Layer List

```
<layer-list>
    <item>
        <bitmap android:src="@drawable/android_red"
            android:gravity="center" />
    </item>
    <item android:top="10dp" android:left="10dp">
        <bitmap android:src="@drawable/android_green"
            android:gravity="center" />
    </item>
    <item android:top="20dp" android:left="20dp">
        <bitmap android:src="@drawable/android_blue"
            android:gravity="center" />
    </item>
</layer-list>
```



# Shape Drawables & GradientDrawable

- Define a shape and its properties in XML
  - Rectangle, oval, ring, line
- Styled with attributes such as `<corners>`, `<gradient>`, `<padding>`, `<size>`, `<solid>` and `<stroke>`

See [Shape Drawables](#) for more attributes

- Can be inflated for a [GradientDrawable](#)



# Creating a GradientDrawable

```
<shape ... android:shape="rectangle">  
    <gradient  
        android:startColor="@color/white"  
        android:endColor="@color/blue"  
        android:angle="45"/>  
    <corners android:radius="8dp" />  
</shape>
```

here is a color gradient...

```
Resources res = getResources();  
Drawable shape = res.getDrawable(R.drawable.gradient_box);  
TextView tv = (TextView)findViewByID(R.id.textview);  
tv.setBackground(shape);
```



# Transition Drawables

- Drawable that can cross-fade between two other drawables
- Each graphic represented by <item> inside <selector>
- Represented by [TransitionDrawable](#) in Java code
- Transition forward by calling `startTransition()`
- Transition backward with `reverseTransition()`

# Creating Transition Drawables

```
<transition ...>
    <selector> <item android:drawable="@drawable/on" />
        <item android:drawable="@drawable/off" />
    </selector>
</transition>
```

```
<ImageButton
    android:id="@+id/button"
    android:src="@drawable/transition" />
```

```
ImageButton button = findViewById(R.id.button);
TransitionDrawable drawable =
    (TransitionDrawable) button.getDrawable();
drawable.startTransition(500);
```

# Vector drawables

- Scale smoothly for all screen sizes
- Android API Level 21 and up
- Use Vector Asset Studio to create (slides below)
- [VectorDrawable](#)



# Creating Vector drawables

```
<vector ...  
    android:height="256dp" android:width="256dp"  
    android:viewportWidth="32" android:viewportHeight="32">  
<path android:fillColor="@color/red"  
    android:pathData="M20.5,9.5  
        c-1.955,0,-3.83,1.268,-4.5,3  
        c-0.67,-1.732,-2.547,-3,-4.5,-3 ... />  
</vector>
```



*pathData for heart shape*

# Image Asset Studio

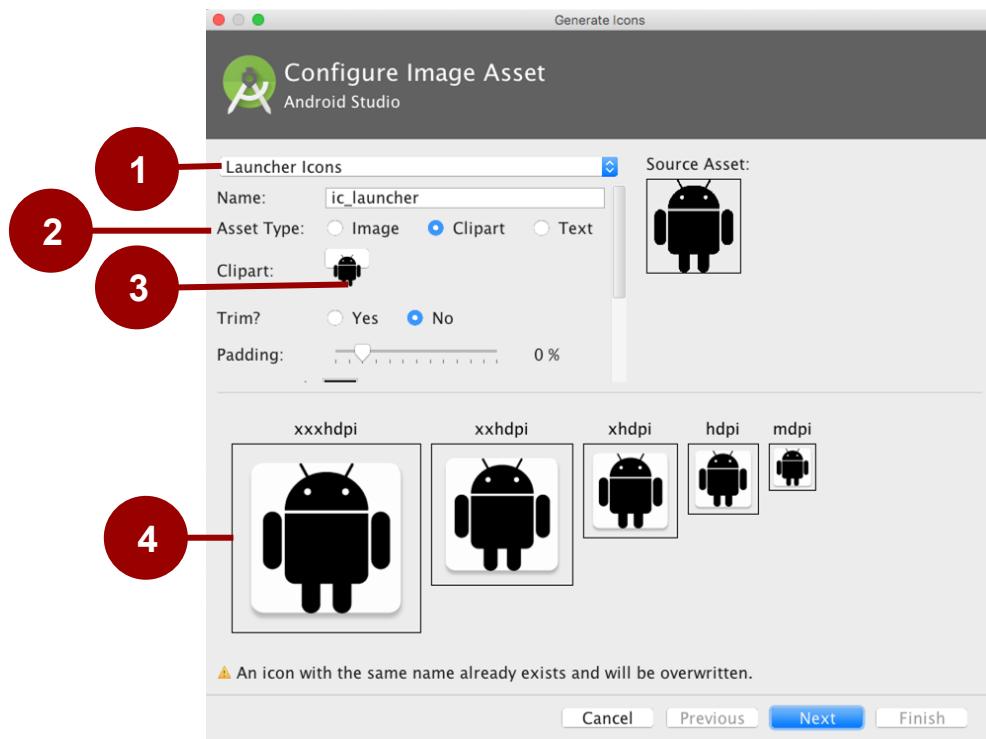
# What is Image Asset Studio?

- Create icons from material icons, images, and text
- Launcher, action bar, tab, notification icons
- Generates a set of icons for generalized screen density
- Stored in /res folder
- To start Image Asset Studio
  1. Right-click the res folder of your project
  2. Choose New > Image Asset



# Using Image Asset Studio

1. Choose icon type and change name
2. Choose Image, Clipart, or Text
3. Click icon to chose clipart
4. Inspect assets for multiple screen sizes



# Vector Asset Studio

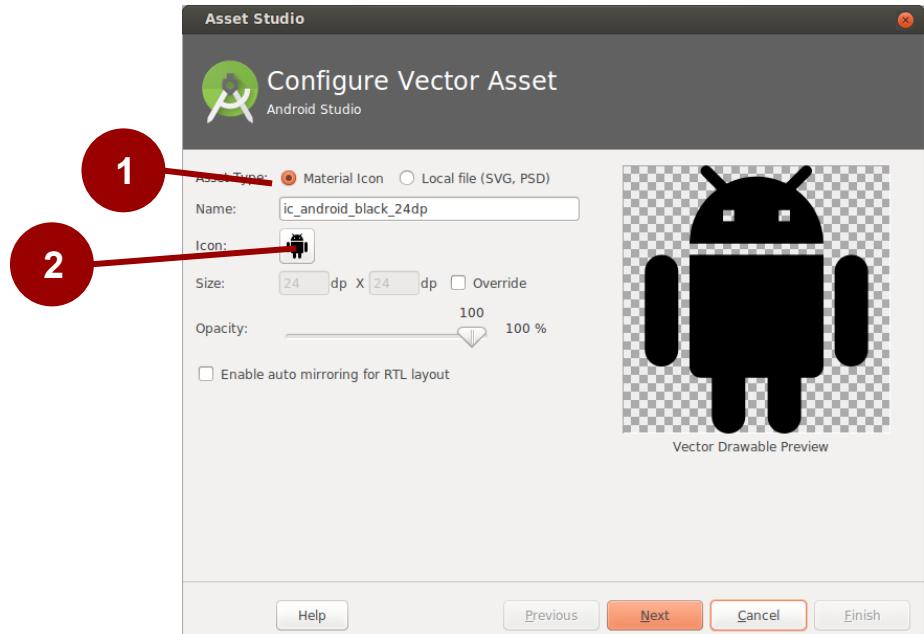
# What is Vector Asset Studio?

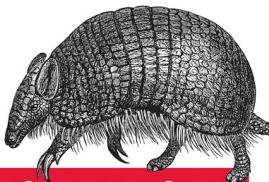
- Create icons from material icons or supply your own vector drawings for API 21 and later
- Launcher, action bar, tab, notification icons
- Generates a scalable vector drawable
- Stored in **res** folder
- To start Image Asset Studio
  1. Right-click **res** folder of your project
  2. Choose **New > Vector Asset**



# Using Image Asset Studio

1. Choose from Material Icon library, or supply your own SVG or PSD vector drawing
2. Opens Material Icon library





Understanding  
Compression

DATA COMPRESSION FOR MODERN DEVELOPERS

Colt McAnlis & Aleks Haecky

# Images, memory, and performance

- Use smallest resolution picture necessary
- Resize, crop, compress
- Vector drawings for simple images
- Use Libraries: [Glide](#) or [Picasso](#)
- Choose appropriate image formats for image type and size
- Use lossy image formats and adjust quality where possible
- Learn about data compression for developers from [Understanding Compression](#)

# Styles



# What is a Style?

- Collection of attributes that define the visual appearance of a View
- Reduce duplication
- Make code more compact
- Manage visual appearance of many components with one style



# Styles reduce clutter

```
<TextView  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:textColor="#00FF00"  
    android:typeface="monospace"  
    android:text="@string/hello" />
```

```
<TextView  
    style="@style/CodeFont"  
    android:text="@string/hello"  
/>
```

# Define styles in styles.xml

**styles.xml** is in **res/values**

```
<resources>
    <style name="CodeFont">
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

# Inheritance: Parent

Define a parent style...

```
<resources>
    <style name="CodeFont">
        <item name="android:layout_width">match_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

# Inheritance: Define child

Define child with Codefont as parent

```
<resources>
    <style name="RedCode" parent="@style/Codefont">
        <item name="android:textColor">#FF0000</item>
    </style>
</resources>
```



# Themes



# Themes

- A Theme is a style applied to an entire activity or even the entire application
- Themes are applied in `AndroidManifest.xml`

```
<application android:theme="@style/AppTheme">
```

# Customize AppTheme of Your Project

```
<!-- Base application theme. -->  
<style name="AppTheme"  
      parent="Theme.AppCompat.Light.DarkActionBar">  
<!-- Try: Theme.AppCompat.Light.NoActionBar -->  
    <!-- Customize your theme here. -->  
    <item name="colorPrimary">@color/colorPrimary</item>  
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>  
    <item name="colorAccent">@color/colorAccent</item>  
</style>
```



# Styles and Themes Resources

Android platform has collection of built

- [Android Styles](#)
- [Android Themes](#)
- [Styles and Themes Guide](#)
- [DayNight Theme Guide](#)



# Learn more

- [Drawable Resource Documentation](#)
- [ShapeDrawable](#)
- [LinearLayout Guide](#)
- [Drawable Resource Guide](#)
- [Supported Media formats](#)
- [9-Patch](#)
- [Understanding Compression](#)



# What's Next?

- Concept Chapter: [5.1 Drawables, styles, and themes](#)
- Practical: [5.1 Drawables, styles, and themes](#)



# END

Android Developer Fundamentals V2

# Delightful User Experience

Lesson 5



Material  
Design

*This work is licensed under a [Creative Commons Attribution 4.0 International License](#).*



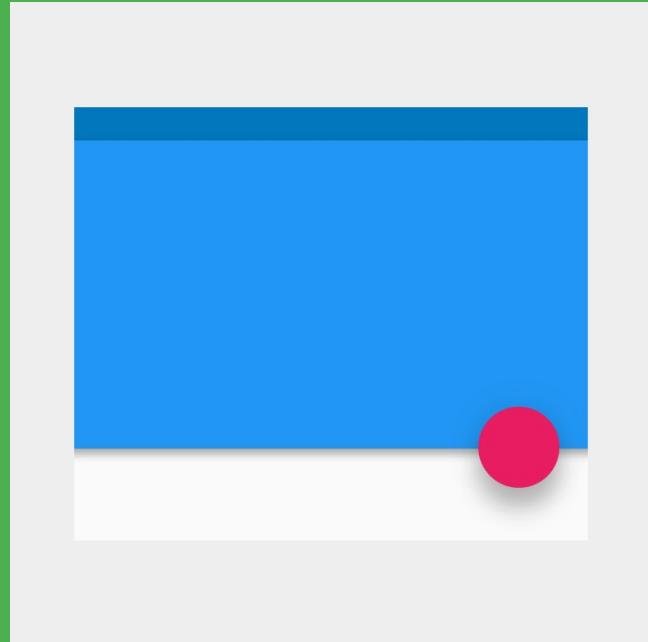
# 5.2 Material Design

# Contents

- The Material Metaphor
- Imagery
- Typography
- Color
- Motion
- Layout
- Components

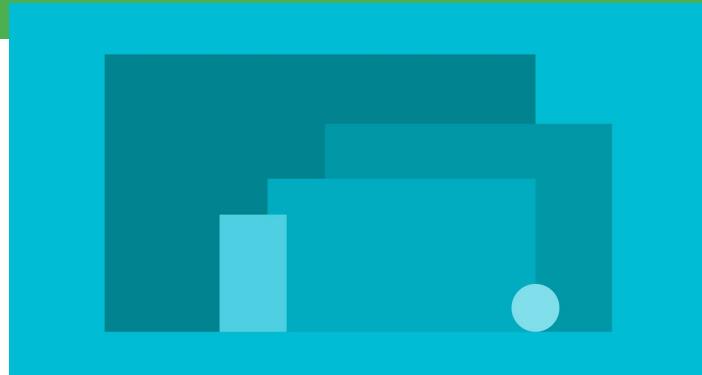


# The Material Metaphor



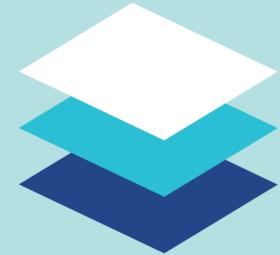
# What is Material Design?

- Design guidelines
- Visual language
- Combine classic principles of good design with innovation and possibilities of technology and science
- [Material Design Spec](#)



# Material metaphor

- Three-dimensional environment containing light, material, and shadows
- Surfaces and edges provide visual cues grounded in reality
- Fundamentals of light, surface, and movement convey how objects move, interact, and exist in space and in relation to each other



# Material design in your app

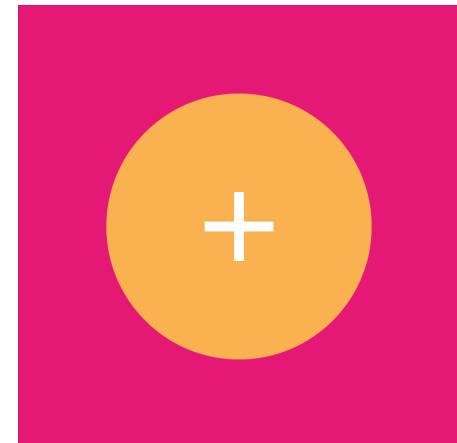
Elements in your Android app should behave similarly to real world materials

- Cast shadows
- Occupy space
- Interact with each other



# Bold, graphic, intentional

- Choose colors deliberately
- Fill screen edge to edge
- Use large-scale typography
- Use white space intentionally
- Emphasize user action
- Make functionality obvious



# Imagery



# Imagery

Images help you communicate and differentiate your app

Should be

- Relevant
- Informative
- Delightful

Best practices

- Use together with text
- Original images
- Provide point of focus
- Build a narrative



# Typography

Quantum Mechanics  
6.626069×10<sup>-34</sup>

*One hundred percent cotton bond*

**Quasiparticles**

It became the non-relativistic limit of quantum field theory

**PAPER CRAFT**

*Probabilistic wave - particle wavefunction orbital path*

**ENTANGLED**

Cardstock 80lb ultra-bright orange

**STATIONERY**

POSITION, MOMENTUM & SPIN

REGULAR

THIN

BOLD ITALIC

BOLD

CONDENSED

LIGHT ITALIC

MEDIUM ITALIC

BLACK

MEDIUM

THIN

CONDENSED LIGHT



# Roboto typeface

Roboto is the standard typeface on Android

Roboto has 6 weights

- Thin
- Light
- Regular
- Medium
- Bold
- Black

Roboto Thin

Roboto Light

Roboto Regular

Roboto Medium

**Roboto Bold**

**Roboto Black**

*Roboto Thin Italic*

*Roboto Light Italic*

*Roboto Italic*

*Roboto Medium Italic*

*Roboto Bold Italic*

*Roboto Black Italic*



# Font styles and scale

- Too many sizes is confusing and looks bad
- Limited set of sizes that work well together

Display 4

Display 3

Display 2

Display 1

Headline

Title

Subheading

Body 2

Body 1

Caption

Button

Light 112sp

Regular 56sp

Regular 45sp

Regular 34sp

Regular 24sp

Medium 20sp

Regular 16sp (Device), Regular 15sp (Desktop)

Medium 14sp (Device), Medium 13sp (Desktop)

Regular 14sp (Device), Regular 13sp (Desktop)

Regular 12sp

MEDIUM (ALL CAPS) 14sp

# Setting text appearance

```
android:textAppearance= "">@style/TextAppearance.AppCompat.Display3"
```

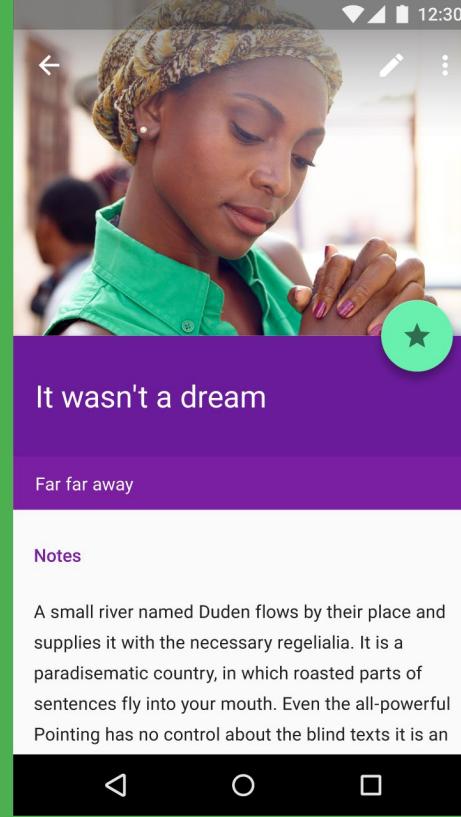
# Fonts as resources

- Bundle fonts as resources in app package (APK)
- Create font folder within res, add font XML file to font
- To access font resource:
  - `@font/myfont`
  - `R.font.myfont`
- Android 8.0 (API level 26) – Android 4.1 (API level 16) and higher, use the Support Library 26
- See [Fonts in XML](#)

# Downloadable fonts

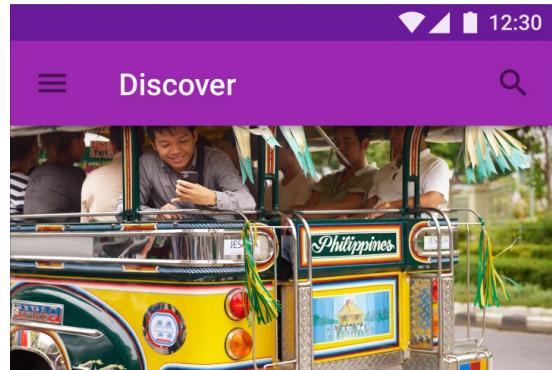
- Download fonts from provider app
  - Reduces APK size
  - Increases the app installation success rate
  - Improves the overall system health, saves cellular data, phone memory, and disk space
- Android 8.0 (API level 26) – API level 14 and higher, use Support Library 26
- See [Downloadable Fonts](#)

# Color



# Color

- Bold hues
- Muted environments
- Deep shadows
- Bright highlights



## Transit in the Philippines

One morning, when Gregor Samsa woke from troubled dreams, he found himself transformed in his bed into a horrible vermin. He lay on his armour-like back, and if he lifted his head a little he could see his brown belly, slightly domed and divided by arches into stiff sections.

# Color palette

Material Design recommends using

- a primary color
- along with some shades
- and an accent color

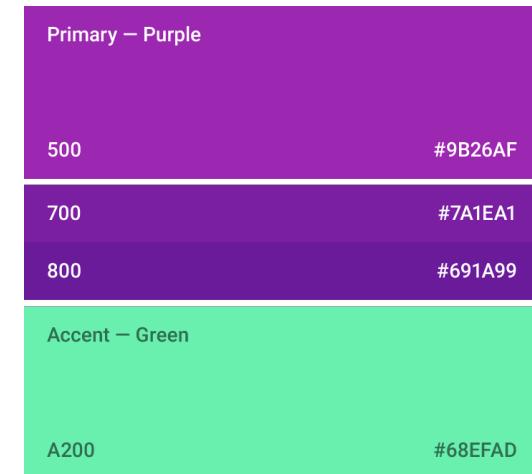


Create a bold user experience for your app

- [Material Design Color Palette](#)

# Color palette for your project

- Android Studio creates a color palette for you
- AppTheme definition in styles.xml
  - colorPrimary—AppBar, branding
  - colorPrimaryDark—status bar, contrast
  - colorAccent—draw user attention, switches, FAB
- Colors defined in colors.xml
- [Color selection tool](#)



# Text color and contrast

- Contrast for visual separation Good choice
- Contrast for readability Good choice
- Contrast for accessibility Bad choice
- Not all people see colors the same Bad choice
- Theme handles text by default
  - Theme.AppCompat.Light—text will be near black Bad choice
  - Theme.AppCompat.Light.DarkActionBar—text near white Good choice

# Motion



# Motion

Motion in Material Design  
describes

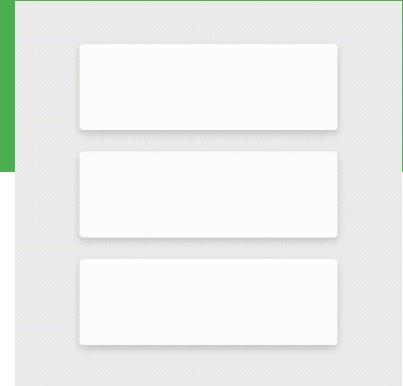
- Spatial relationships
- Functionality
- Intention

Motion is

- Responsive
- Natural
- Aware
- Intentional

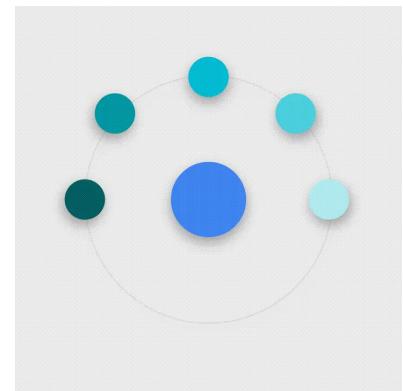
# Motion in your app

- Maintain continuity
- Highlight elements or actions
- Transition naturally between actions or states
- Draw focus
- Organize transitions
- Responsive feedback

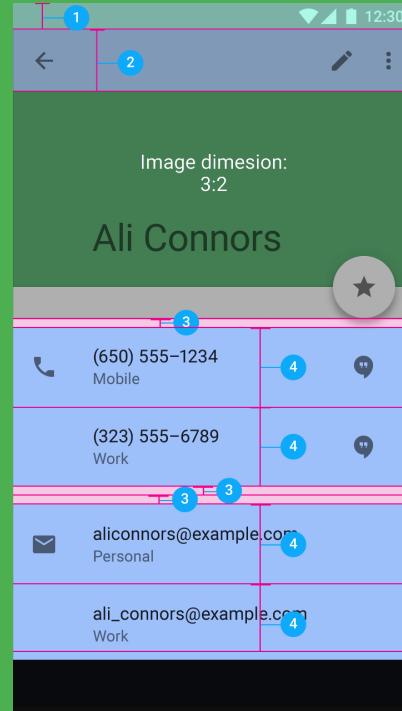


Touch feedback

Responsive interaction



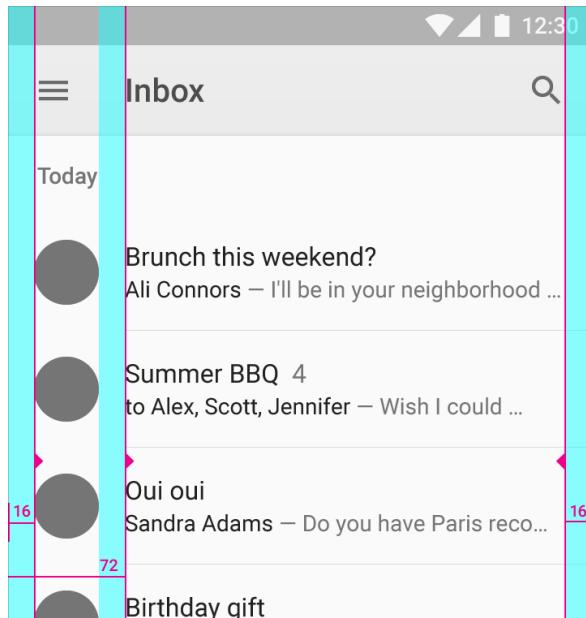
# Layout



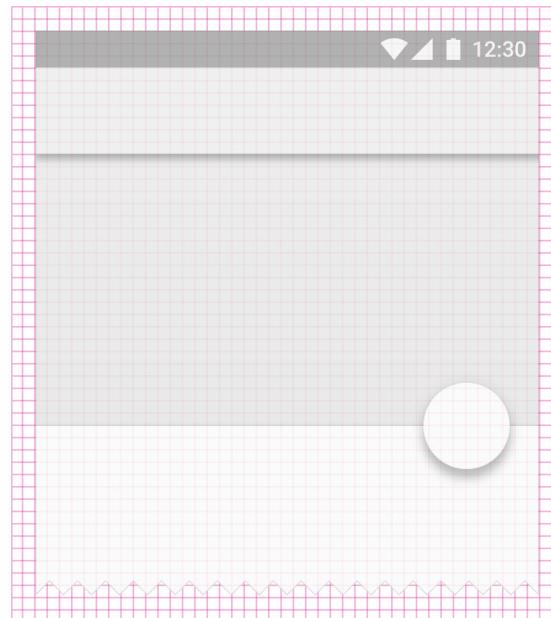
# Layout for Material Design

- Density independent pixels for views—dp
- Scalable pixels for text—sp
- Elements align to a grid with consistent spacing
- Plan your layout
- Use templates for common layout patterns

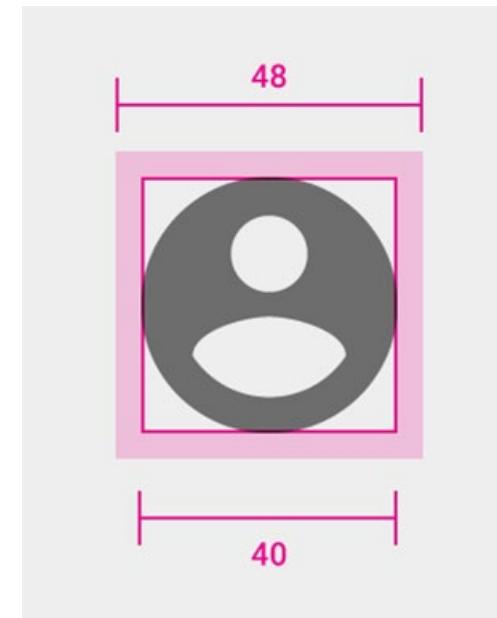
# Layout planning



Spacing

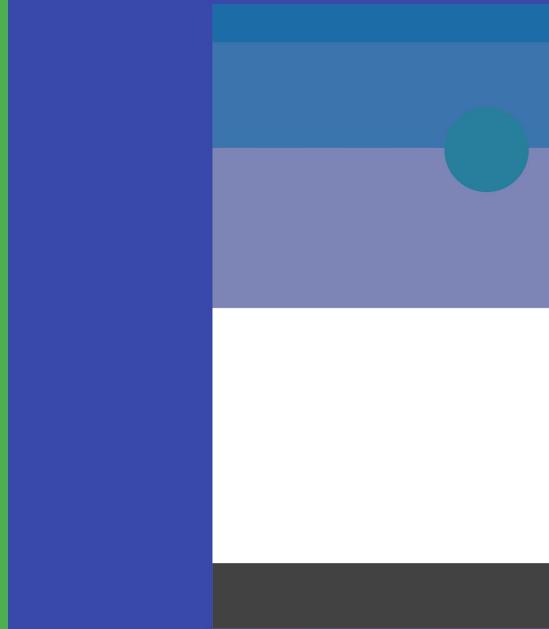


Grid alignment



Sizing

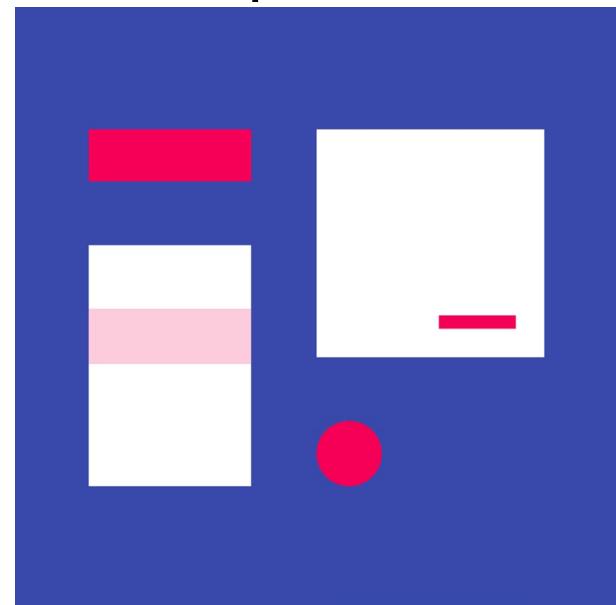
# Components



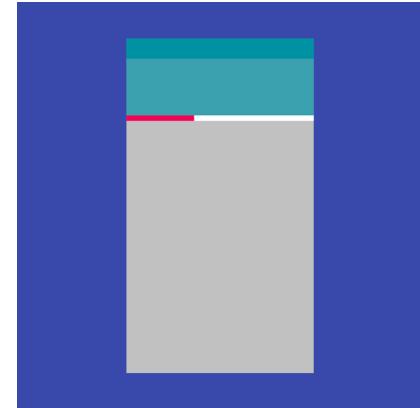
# Components

Material Design has guidelines on the use and implementation of Android components

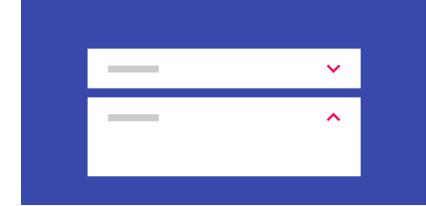
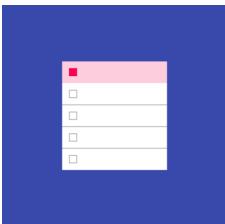
- Bottom Navigation
- Buttons
- Cards
- Chips
- Data Tables
- Dialogs
- Dividers
- Sliders
- Snackbar
- Toasts
- Steppers
- Subheaders
- Text Fields
- Toolbars



# More components

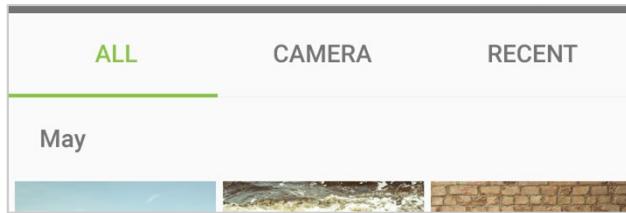
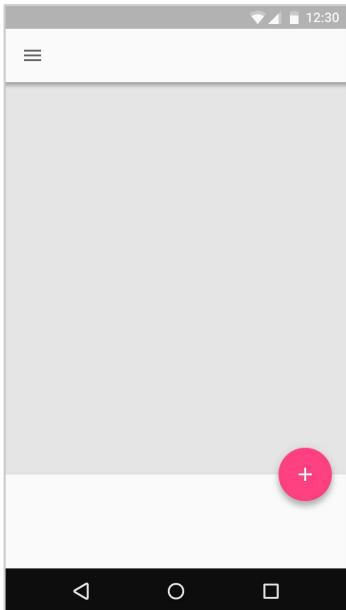


- Expansion Panels
- Grid Lists
- Lists
- Menus
- Pickers
- Progress Bars
- Selection Controls

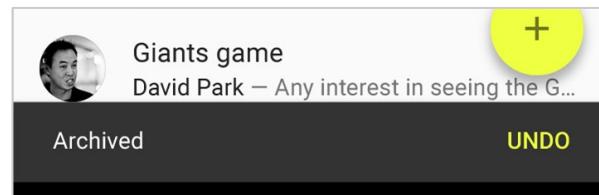


# Consistency helps user intuition

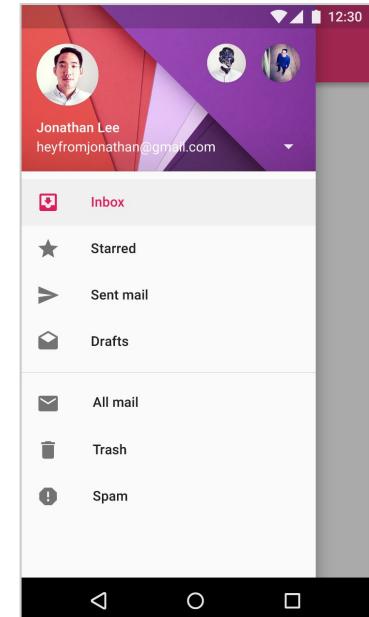
FAB



Tabs



Snackbar



Navigation Drawer

# Learn more

- [Material Design Guidelines](#)
- [Material Design Guide](#)
- [Material Design for Android](#)
- [Material Design for Developers](#)
- [Material Palette Generator](#)
- [Cards and Lists Guide](#)
- [Floating Action Button Reference](#)
- [Defining Custom Animations](#)
- [View Animation](#)



# What's Next?

- Concept Chapter: [5.2 Material Design](#)
- Practical: [5.2 Cards, and colors](#)

# END



Android Developer Fundamentals V2

# Delightful User Experience

Lesson 5



Resources for  
adaptive layouts

*This work is licensed under a [Creative Commons Attribution 4.0 International License](#).*



# 5.3 Adaptive layouts and resources

# Contents

- Adaptive layouts and resources
- Alternative resources
- Default resources



# Adaptive layouts and resources

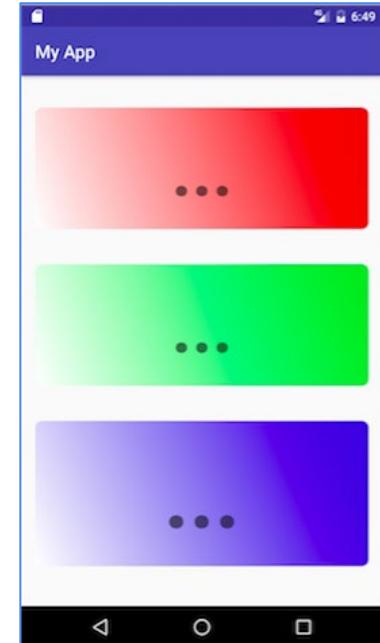
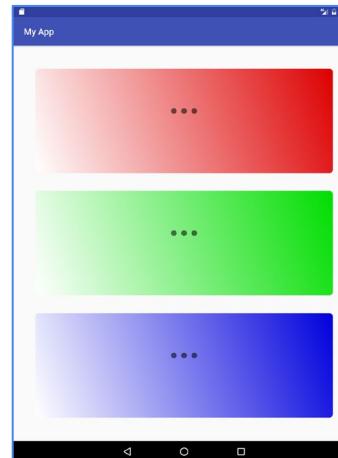
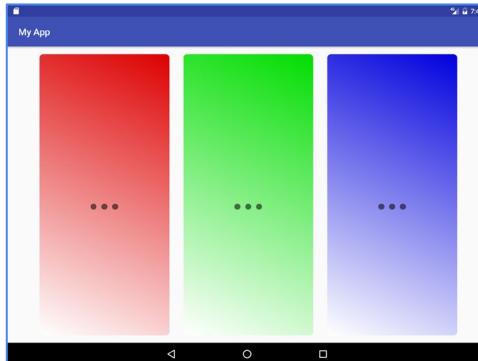
Resources for  
adaptive layouts

*This work is licensed under a [Creative Commons Attribution 4.0 International License](#).*



# What are adaptive layouts?

Layouts that look good on different screen sizes, orientations, and devices



# Adaptive layout

- Layout adapts to configuration
  - Screen size
  - Device orientation
  - Locale
  - Version of Android installed
- Provides alternative resources
  - Localized strings
- Uses flexible layouts
  - GridLayout



# Resource folders of a small app

```
MyProject/  
    src/  
    res/  
        drawable/  
            graphic.png  
        layout/  
            activity_main.xml  
            list_iteminfo.xml  
        mipmap/  
            ic_launcher_icon.png  
    values/  
        strings.xml
```

Put resources in your project's res folder



# Common resource directories

- `drawable/`, `layout/`, `menu/`
- `values/`—XML files of simple values, such as string or color
- `xml/`—arbitrary XML files
- `raw/`—arbitrary files in their raw form
- `mipmap/`—drawables for different launcher icon densities
- [Complete list](#)

# Alternative resources



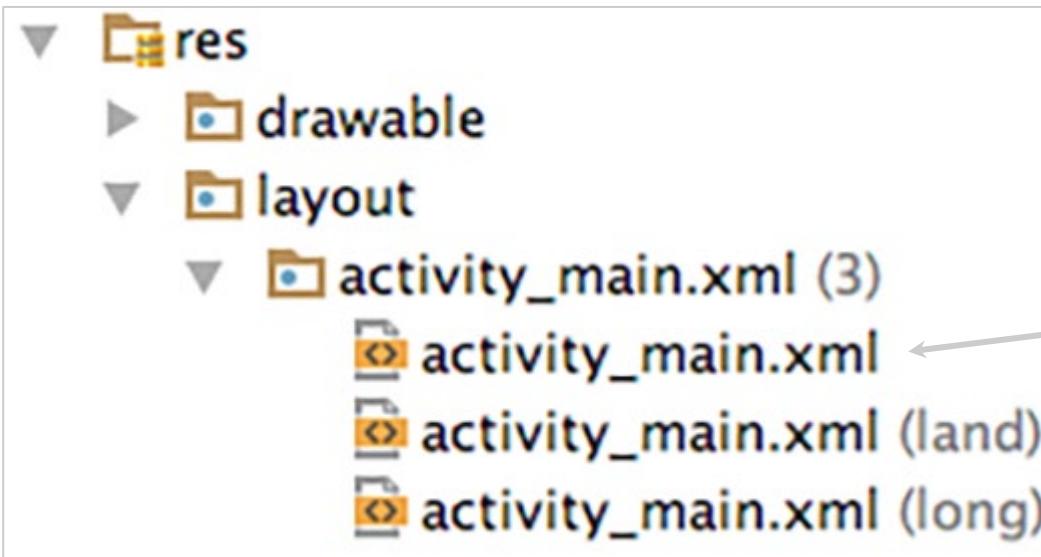
# What are alternative resources?

Different device configurations may require different resources

- Localized strings
- Image resolutions
- Layout dimensions

Android loads appropriate resources automatically

# Create alternative resource folders



Use alternative  
folders for resources  
for different device  
configurations

# Names for alternative resource folders

Resource folder names have the format  
*resources name-config qualifier*

drawable-hdpi	drawables for high-density displays
layout-land	layout for landscape orientation
layout-v7	layout for version of platform
values-fr	all values files for French locale

[List of directories and qualifiers](#) and usage detail



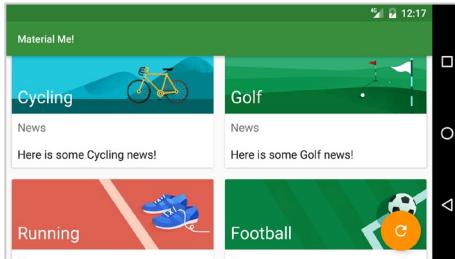
# Screen Orientation

- Use `res/layout` and provide alternatives for landscape where necessary
  - `res/layout-port` for portrait-specific layouts
  - `res/layout-land` for landscape specific layouts
- Avoid hard-coded dimensions to reduce need for specialized layouts

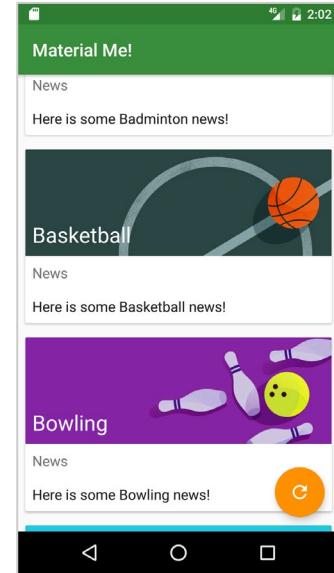
# Simple adaptive layout

## GridLayout

- In values/integer.xml:  
`<integer name="grid_column_count">1</integer>`
- In values/integer.xml-land:  
`<integer name="grid_column_count">2</integer>`



Landscape



Portrait

# Smallest width

- Smallest-width (sw) in folder name specifies minimum device width
  - res/values-sw $n$ dp, where  $n$  is the smallest width
  - Example: res/values-sw600dp/dimens.xml
  - Does not change with orientation
- Android uses resource closest to (without exceeding) the device's smallest width

# Platform Version

- API level supported by device
  - `res/drawables-v14`  
contains drawables for devices that support API level 14 and above
- Some resources are only available for newer versions
  - WebP image format requires API level 14 (Android 4.0)
- [Android API level](#)

# Localization

- Provide strings (and other resources) for specific locales
  - `res/values-es/strings.xml`
- Increases potential audience for your app
- Locale is based on device's settings
- [Localization](#)



# Default resources



# Default Resources

- Always provide default resources
  - directory name without a qualifier
  - res/layout, res/values, res/drawables....
- Android falls back on default resources when no specific resources match configuration
- Localizing with Resources

# Learn more

- [Supporting Multiple Screens](#)
- [Providing Resource](#)
- [Providing Resources Guide](#)
- [Resources Overview](#)
- [Localization Guide](#)



# What's Next?

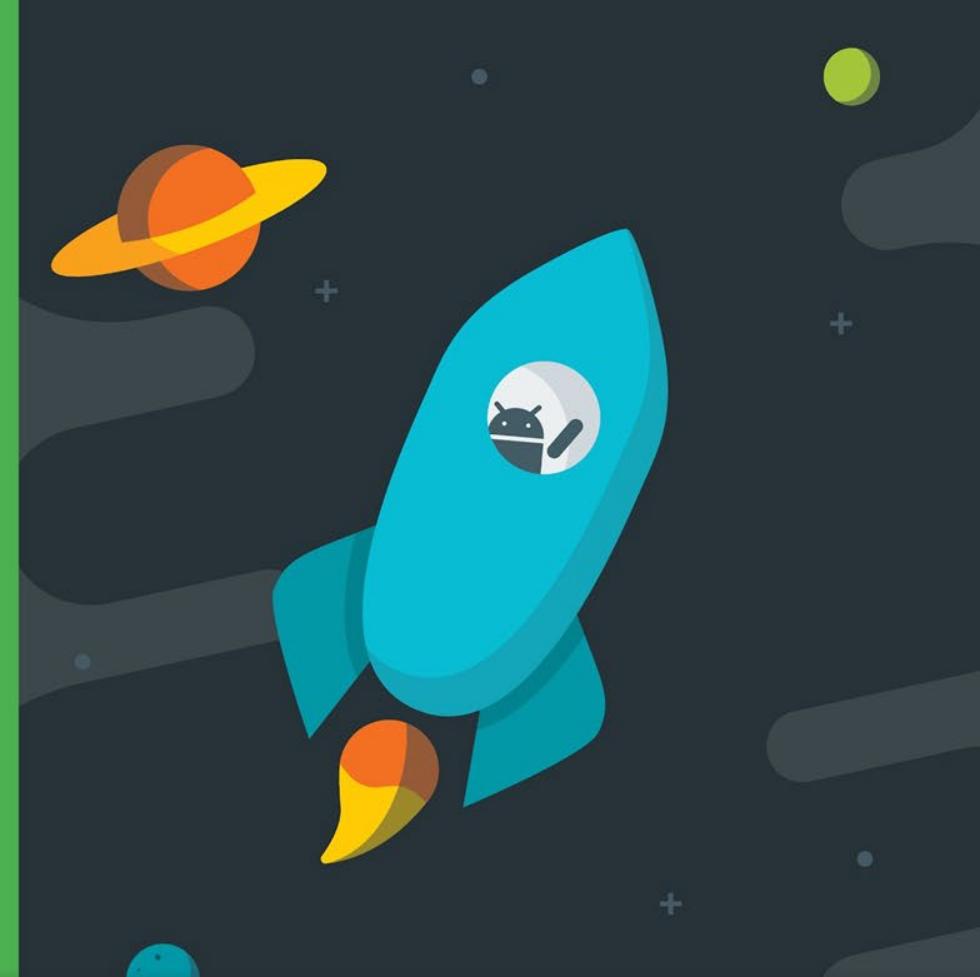
- Concept Chapter: [5.3 Resources for adaptive layouts](#)
- Practical: [5.3 Adaptive layouts](#)



# END

# Testing your UI

## Lesson 6



This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# 6.1 UI testing

# Contents

- UI testing overview
- Test environment and Espresso setup
- Creating Espresso tests
- Espresso test examples
- Recording tests



# UI testing overview



# UI testing

- Perform all user UI actions with View elements
  - Tap a View, and enter data or make a choice
  - Examine the values of the properties of each View
- Provide input to all View elements
  - Try invalid values
- Check returned output
  - Correct or expected values?
  - Correct presentation?



# Problems with testing manually

- Time consuming, tedious, error-prone
- UI may change and need frequent retesting
- Some paths fail over time
- As app gets more complex, possible sequences of actions may grow non-linearly

# Benefits of testing automatically

- Free your time and resources for other work
- Faster than manual testing
- Repeatable
- Run tests for different device states and configurations



# Espresso for single app testing

- Verify that the UI behaves as expected
- Check that the app returns the correct UI output in response to user interactions
- Navigation and controls behave correctly
- App responds correctly to mocked-out dependencies

# UI Automator for multiple apps

- Verify that interactions between different user apps and system apps behave as expected
- Interact with visible elements on a device
- Monitor interactions between app and system
- Simulate user interactions
- Requires instrumentation

# What is instrumentation?

- A set of hooks in the Android system
- Loads test package and app into same process, allowing tests to call methods and examine fields
- Control components independently of app's lifecycle
- Control how Android loads apps

# Benefits of instrumentation

- Tests can monitor all interaction with Android system
- Tests can invoke methods in the app
- Tests can modify and examine fields in the app independent of the app's lifecycle

# Test environment And Espresso setup



# Install Android Support Library

1. In Android Studio choose **Tools > Android > SDK Manager**
2. Click **SDK Tools** and look for **Android Support Repository**
3. If necessary, update or install the library

# Add dependencies to build.gradle

- Android Studio templates include dependencies
- If needed, add the following dependencies:

```
testImplementation 'junit:junit:4.12'  
androidTestImplementation 'com.android.support.test:runner:1.0.1'  
androidTestImplementation  
    'com.android.support.test.espresso:espresso-core:3.0.1'
```

# Add defaultConfig to build.gradle

- Android Studio templates include defaultConfig setting
- If needed, add the following to defaultConfig section:

testInstrumentationRunner

```
"android.support.test.runner.AndroidJUnitRunner"
```

# Prepare your device

1. Turn on USB Debugging
2. Turn off all animations in **Developer Options > Drawing**
  - Window animation scale
  - Transition animation scale
  - Animator duration scale

# Create tests

- Store in *module-name*/src/androidTests/java/
  - In Android Studio: app > java > *module-name* (androidTest)
- Create tests as JUnit classes

# Creating Espresso tests



# Test class definition

**@RunWith(AndroidJUnit4.class)** – Required annotation for tests

**@LargeTest** – Based on resources the test uses and time to run

```
public class ChangeTextBehaviorTest {}
```

**@SmallTest** – Runs in < 60s and uses no external resources

**@MediumTest** – Runs in < 300s, only local network

**@LargeTest** – Runs for a long time and uses many resources



# @Rule specifies the context of testing

**@Rule**

```
public ActivityTestRule<MainActivity> mActivityRule =  
    new ActivityTestRule<>(MainActivity.class);
```

[@ActivityTestRule](#) – Testing support for a single specified activity

[@ServiceTestRule](#) – Testing support for starting, binding, shutting down a service

# @Before and @After set up and tear down

## **@Before**

```
public void initValidString() {  
    mStringToBetyped = "Espresso";  
}
```

**@Before** – Setup, initializations

**@After** – Teardown, freeing resources



# @Test method structure

**@Test**

```
public void changeText_sameActivity() {  
    // 1. Find a View  
    // 2. Perform an action  
    // 3. Verify action was taken, assert result  
}
```



# "Hamcrest" simplifies tests

- “Hamcrest” an anagram of “Matchers”
- Framework for creating custom matchers and assertions
- Match rules defined declaratively
- Enables precise testing
- [The Hamcrest Tutorial](#)



# Hamcrest Matchers

- ViewMatcher – find Views by id, content, focus, hierarchy
- ViewAction – perform an action on a view
- ViewAssertion – assert state and verify the result



# Basic example test

```
@Test  
public void changeText_sameActivity() {  
    // 1. Find view by Id  
    onView(withId(R.id.editTextUserInput))  
        // 2. Perform action-type string and click button  
        .perform(typeText(mStringToBetyped), closeSoftKeyboard());  
    onView(withId(R.id.changeTextBt)).perform(click());  
    // 3. Check that the text was changed  
    onView(withId(R.id.textToBeChanged))  
        .check(matches(withText(mStringToBetyped)));  
}
```



# Finding views with onView

- `withId()` – find a view with the specified Android id
  - `onView(withId(R.id.editTextUserInput))`
- `withText()` – find a view with specific text
- `allOf()` – find a view to that matches multiple conditions
- Example: Find a visible list item with the given text:

```
onView(allOf(withId(R.id.word),  
           withText("Clicked! Word 15"),  
           isDisplayed()))
```



# onView returns ViewInteraction object

- If you need to reuse the View returned by onView
- Make code more readable or explicit
- check() and perform() methods

```
ViewInteraction textView = onView(  
    allOf(withId(R.id.word), withText("Clicked! Word 15"),  
    isDisplayed()));  
textView.check(matches(withText("Clicked! Word 15")));
```

# Perform actions

- Perform an action on the View found by a ViewMatcher
- Can be any action you can perform on the View

```
// 1. Find view by Id  
onView(withId(R.id.editTextUserInput))  
  
// 2. Perform action-type string and click button  
.perform(typeText(mStringToBetyped), closeSoftKeyboard());  
onView(withId(R.id.changeTextBt)).perform(click());
```

# Check result

- Asserts or checks the state of the View

```
// 3. Check that the text was changed  
onView(withId(R.id.textToBeChanged))  
.check(matches(withText(mStringToBetyped)));
```



# When a test fails

## Test

```
onView(withId(R.id.text_message))
    .check(matches(withText("This is a failing test.")));
```

## Result snippet

```
android.support.test.espresso.base.DefaultFailureHandler$Assertion
FailedWithCauseError: 'with text: is "This is a failing test."'
doesn't match the selected view.

Expected: with text: is "This is a failing test."
Got: "AppCompatTextView{id=2131427417, res-name=text_message ...
```



# Recording tests

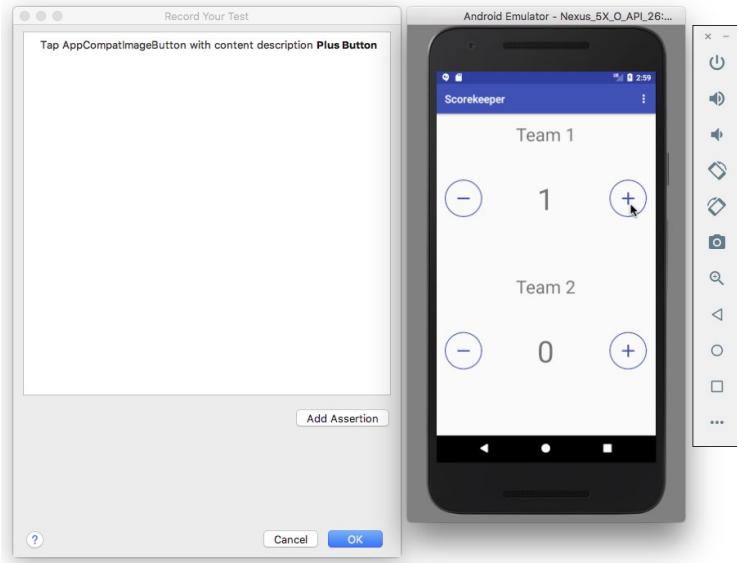


# Recording an Espresso test

- Use app normally, clicking through the UI
- Editable test code generated automatically
- Add assertions to check if a view holds a certain value
- Record multiple interactions in one session, or record multiple sessions

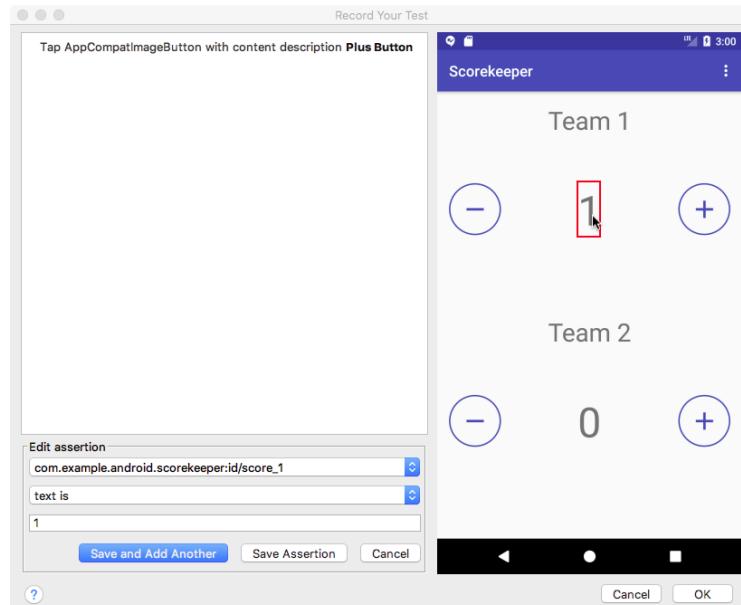
# Start recording an Espresso test

1. Run > Record Espresso Test
2. Click Restart app, select target, and click OK
3. Interact with the app to do what you want to test



# Add assertion to Espresso test recording

4. Click **Add Assertion** and select a UI element
5. Choose **text is** and enter the text you expect to see
6. Click **Save Assertion** and click **Complete Recording**



# Learn more from developer docs

## Android Studio Documentation

- [Test Your App](#)
- [Espresso basics](#)
- [Espresso cheat sheet](#)

## Android Developer Documentation

- [Best Practices for Testing](#)
- [Getting Started with Testing](#)
- [Testing UI for a Single App](#)
- [Building Instrumented Unit Tests](#)
- [Espresso Advanced Samples](#)
- [The Hamcrest Tutorial](#)
- [Hamcrest API and Utility Classes](#)
- [Test Support APIs](#)

# Learn even more

## Android Testing Support Library

- [Espresso documentation](#)
- [Espresso Samples](#)

## Videos

- [Android Testing Support - Android Testing Patterns #1](#) (introduction)
- [Android Testing Support - Android Testing Patterns #2](#) (onView view matching)
- [Android Testing Support - Android Testing Patterns #3](#) (onData & adapter views)



# Learn even more

- Google Testing Blog: [Android UI Automated Testing](#)
- Atomic Object: “[Espresso – Testing RecyclerViews at Specific Positions](#)”
- Stack Overflow: “[How to assert inside a RecyclerView in Espresso?](#)”
- GitHub: [Android Testing Samples](#)
- Google Codelabs: [Android Testing Codelab](#)

# What's Next?

- Concept Chapter: [6.1 UI testing](#)
- Practical: [6.1 Espresso for UI testing](#)



# END

# Background Tasks

Lesson 7



# 7.1 AsyncTask and AsyncTaskLoader

# Contents

- Threads
- AsyncTask
- Loaders
- AsyncTaskLoader



# Threads

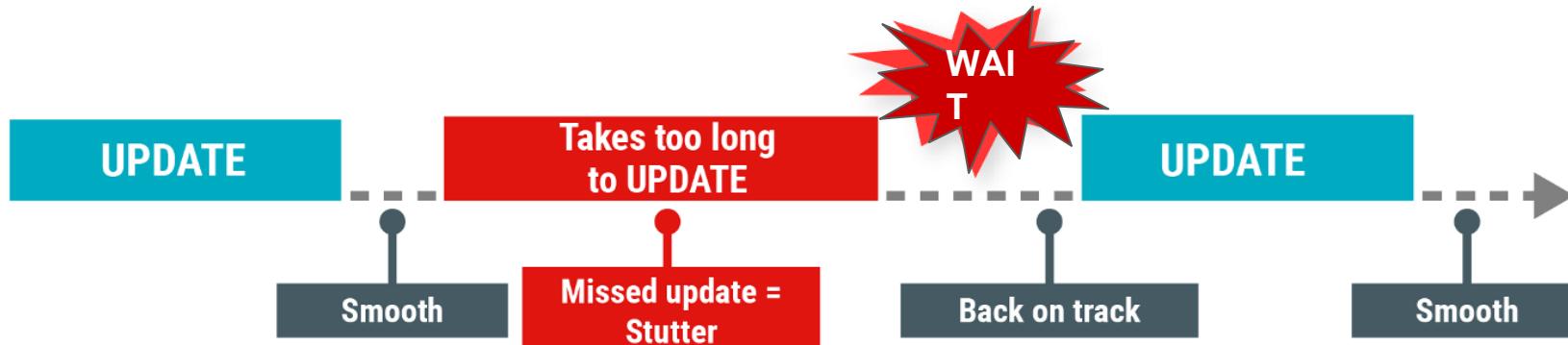


# The main thread

- Independent path of execution in a running program
- Code is executed line by line
- App runs on Java thread called "main" or "UI thread"
- Draws UI on the screen
- Responds to user actions by handling UI events

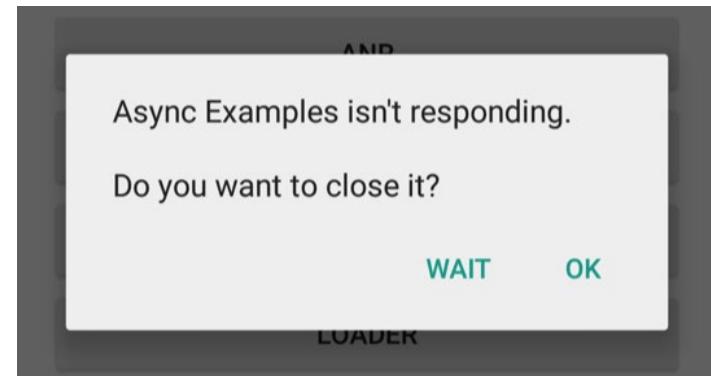
# The Main thread must be fast

- Hardware updates screen every 16 milliseconds
- UI thread has 16 ms to do all its work
- If it takes too long, app stutters or hangs



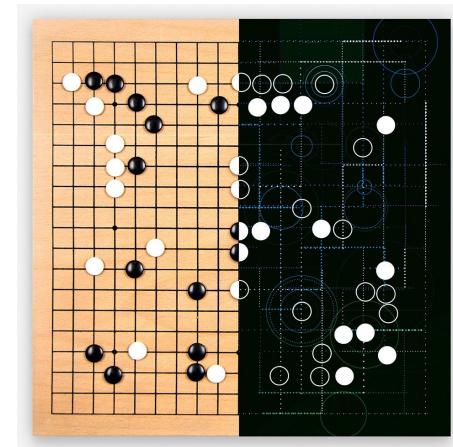
# Users uninstall unresponsive apps

- If the UI waits too long for an operation to finish, it becomes unresponsive
- The framework shows an Application Not Responding (ANR) dialog



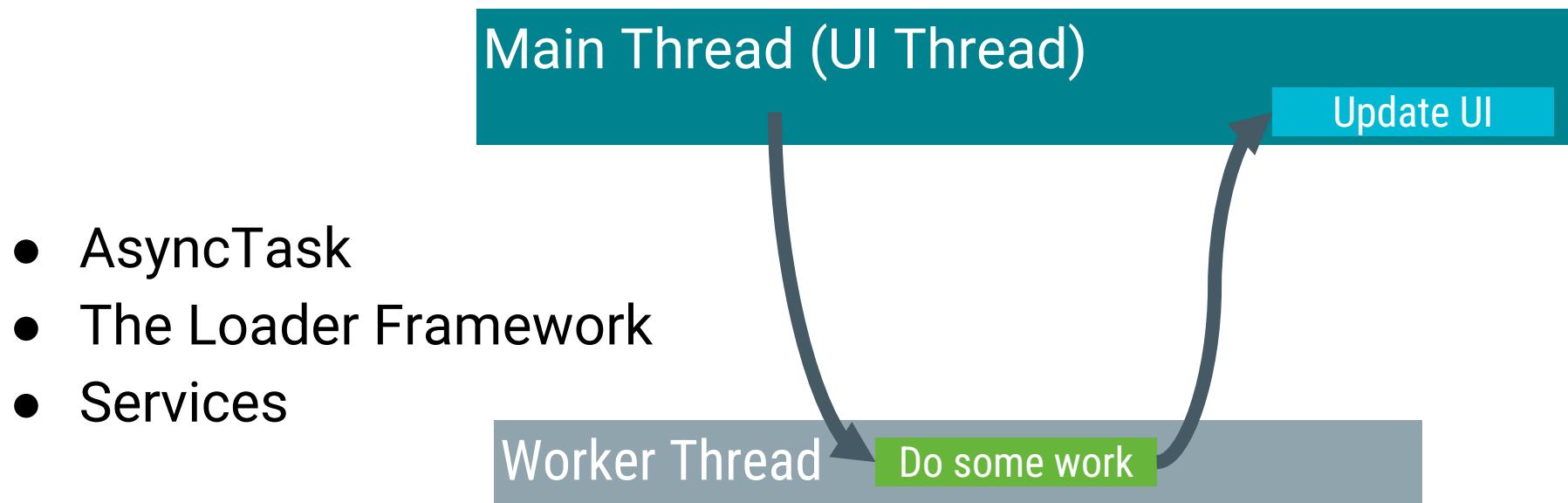
# What is a long running task?

- Network operations
- Long calculations
- Downloading/uploading files
- Processing images
- Loading data



# Background threads

Execute long running tasks on a **background thread**



# Two rules for Android threads

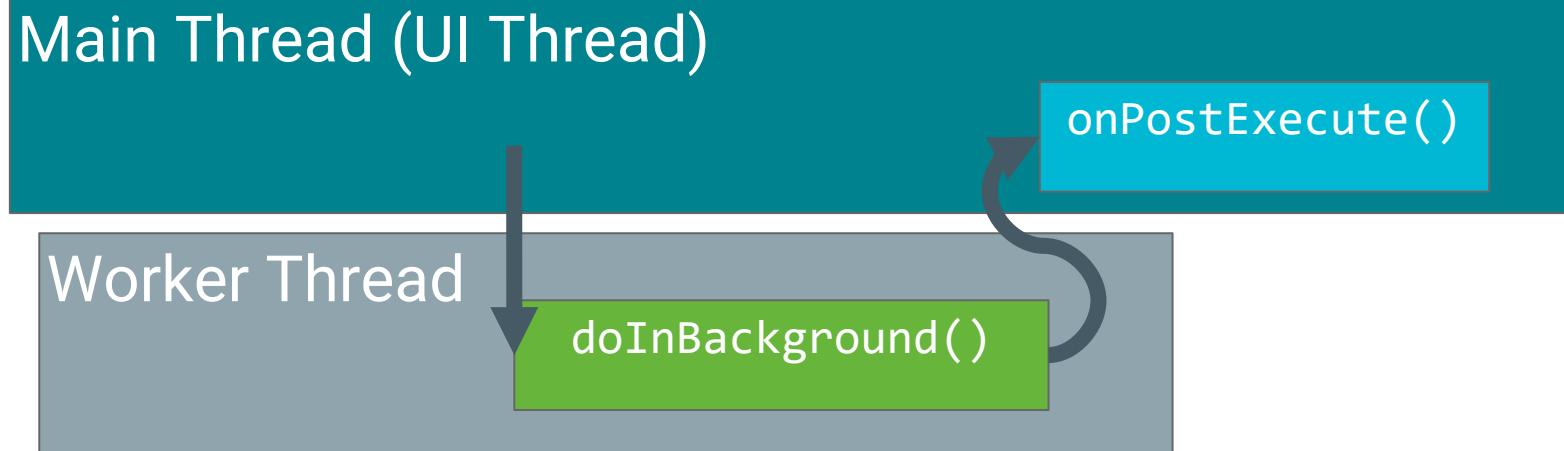
- Do not block the UI thread
  - Complete all work in less than 16 ms for each screen
  - Run slow non-UI work on a non-UI thread
- Do not access the Android UI toolkit from outside the UI thread
  - Do UI work only on the UI thread

# AsyncTask



# What is AsyncTask?

Use [AsyncTask](#) to implement basic background tasks



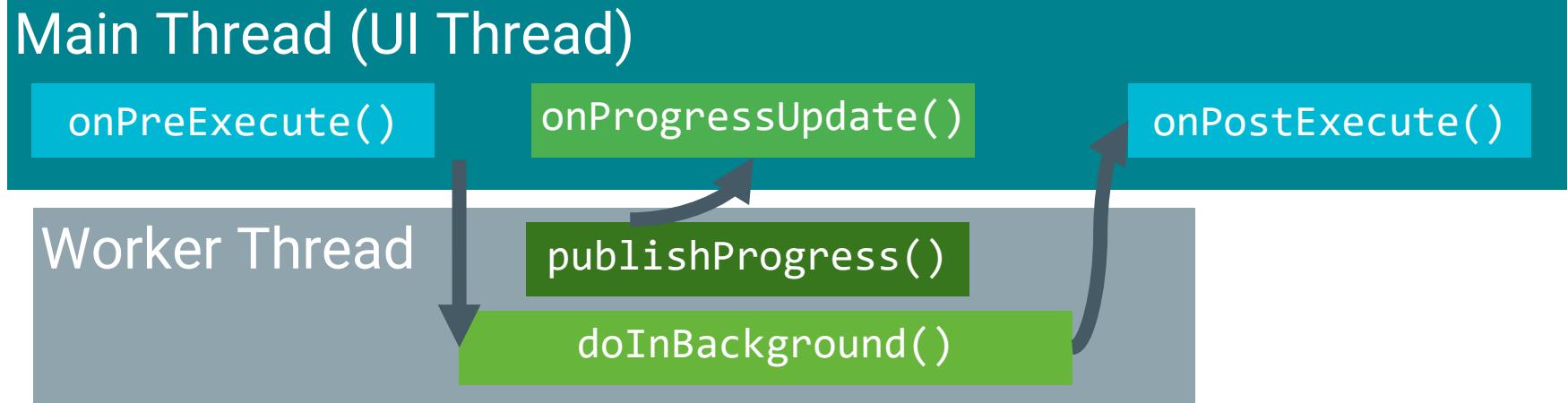
# Override two methods

- `doInBackground()`—runs on a background thread
  - All the work to happen in the background
- `onPostExecute()`—runs on main thread when work done
  - Process results
  - Publish results to the UI

# AsyncTask helper methods

- `onPreExecute()`
  - Runs on the main thread
  - Sets up the task
- `onProgressUpdate()`
  - Runs on the main thread
  - receives calls from `publishProgress()` from background thread

# AsyncTask helper methods



# Creating an AsyncTask

1. Subclass AsyncTask
2. Provide data type sent to doInBackground()
3. Provide data type of progress units for  
onProgressUpdate()
4. Provide data type of result for onPostExecute()

```
private class MyAsyncTask
```

```
    extends AsyncTask<URL, Integer, Bitmap> { ... }
```

# MyAsyncTask class definition

```
private class MyAsyncTask  
    extends AsyncTask<String, Integer, Bitmap> {...}
```

doInBackground()

onProgressUpdate()

onPostExecute()

- String—could be query, URI for filename
- Integer—percentage completed, steps done
- Bitmap—an image to be displayed
- Use Void if no data passed

# onPreExecute()

```
protected void onPreExecute() {  
    // display a progress bar  
    // show a toast  
}
```

# doInBackground()

```
protected Bitmap doInBackground(String... query) {  
    // Get the bitmap  
    return bitmap;  
}
```



# onProgressUpdate()

```
protected void onProgressUpdate(Integer... progress) {  
    setProgressPercent(progress[0]);  
}
```

# onPostExecute()

```
protected void onPostExecute(Bitmap result) {  
    // Do something with the bitmap  
}
```

# Start background work

```
public void loadImage (View view) {  
    String query = mEditText.getText().toString();  
    new MyAsyncTask(query).execute();  
}
```

# Limitations of AsyncTask

- When device configuration changes, Activity is destroyed
- AsyncTask cannot connect to Activity anymore
- New AsyncTask created for every config change
- Old AsyncTasks stay around
- App may run out of memory or crash

# When to use AsyncTask

- Short or interruptible tasks
- Tasks that do not need to report back to UI or user
- Lower priority tasks that can be left unfinished
- Use AsyncTaskLoader otherwise



# Loaders



# What is a Loader?

- Provides asynchronous loading of data
- **Reconnects to Activity after configuration change**
- Can monitor changes in data source and deliver new data
- Callbacks implemented in Activity
- Many types of loaders available
  - [AsyncTaskLoader](#), [CursorLoader](#)



# Why use loaders?

- Execute tasks OFF the UI thread
- LoaderManager handles configuration changes for you
- Efficiently implemented by the framework
- Users don't have to wait for data to load

# What is a LoaderManager?

- Manages loader functions via callbacks
- Can manage multiple loaders
  - loader for database data, for AsyncTask data, for internet data...

# Get a loader with initLoader()

- Creates and starts a loader, or reuses an existing one, including its data
- Use restartLoader() to clear data in existing loader

```
getLoaderManager().initLoader(Id, args, callback);
```

```
getLoaderManager().initLoader(0, null, this);
```

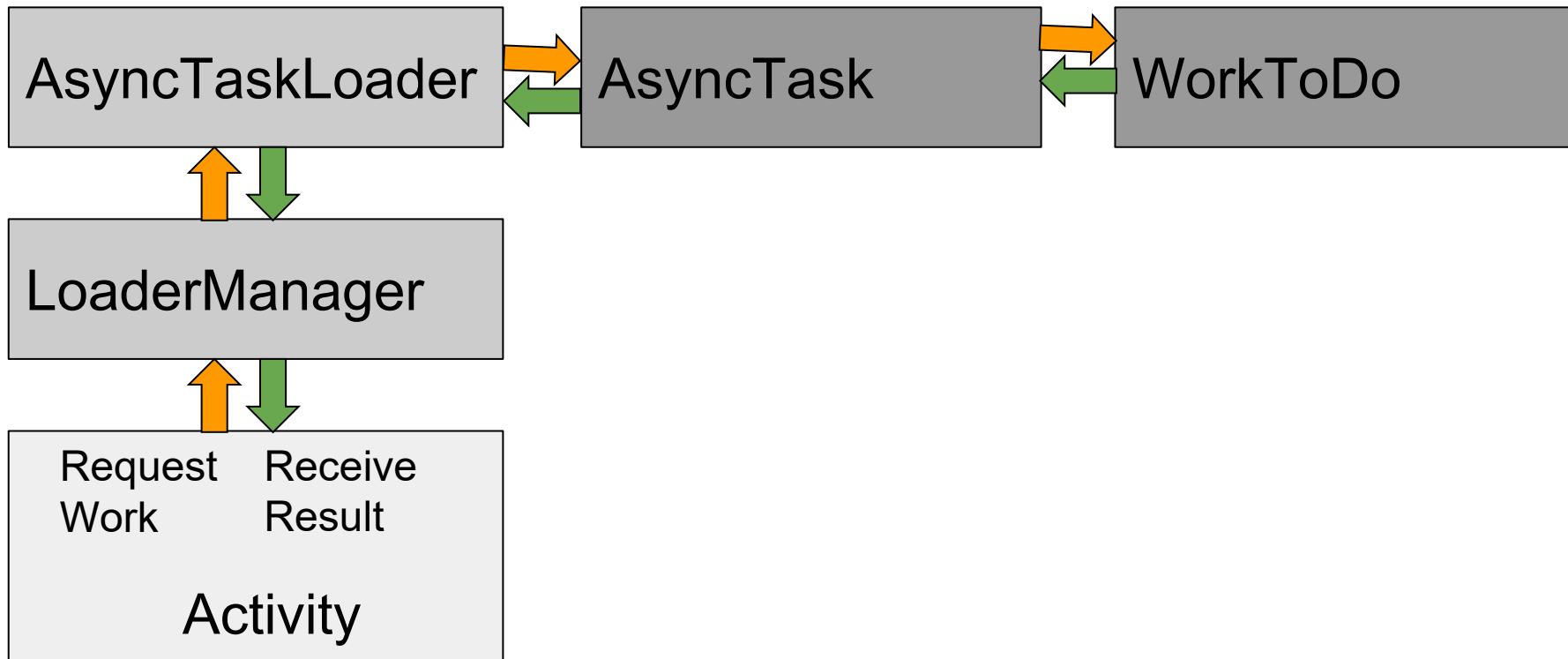
```
getSupportLoaderManager().initLoader(0, null, this);
```



# Implementing AsyncTaskLoader



# AsyncTaskLoader Overview



# AsyncTask —> AsyncTaskLoader

`doInBackground()` → `loadInBackground()`  
`onPostExecute()` → `onLoadFinished()`

# Steps for AsyncTaskLoader subclass

1. Subclass [AsyncTaskLoader](#)
2. Implement constructor
3. `loadInBackground()`
4. `onStartLoading()`

# Subclass AsyncTaskLoader

```
public static class StringListLoader  
    extends AsyncTaskLoader<List<String>> {  
  
    public StringListLoader(Context context, String queryString) {  
        super(context);  
        mQueryString = queryString;  
    }  
}
```



# loadInBackground()

```
public List<String> loadInBackground() {  
    List<String> data = new ArrayList<String>;  
    //TODO: Load the data from the network or from a database  
    return data;  
}
```

# onStartLoading()

When `restartLoader()` or `initLoader()` is called, the `LoaderManager` invokes the `onStartLoading()` callback

- Check for cached data
- Start observing the data source (if needed)
- Call `forceLoad()` to load the data if there are changes or no cached data

```
protected void onStartLoading() { forceLoad(); }
```



# Implement loader callbacks in Activity

- `onCreateLoader()` – Create and return a new Loader for the given ID
- `onLoadFinished()` – Called when a previously created loader has finished its load
- `onLoaderReset()` – Called when a previously created loader is being reset making its data unavailable

# onCreateLoader()

```
@Override  
public Loader<List<String>> onCreateLoader(int id, Bundle args) {  
    return new StringListLoader(this, args.getString("queryString"));  
}
```

# onLoadFinished()

Results of `loadInBackground()` are passed to `onLoadFinished()` where you can display them

```
public void onLoadFinished(Loader<List<String>> loader,  
List<String> data) {  
    mAdapter.setData(data);  
}
```

# onLoaderReset()

- Only called when loader is destroyed
- Leave blank most of the time

```
@Override  
public void onLoaderReset(final LoaderList<String>> loader) { }
```

# Get a loader with initLoader()

- In Activity
- Use support library to be compatible with more devices

```
getSupportLoaderManager().initLoader(0, null, this);
```

# Learn more

- [AsyncTask Reference](#)
- [AsyncTaskLoader Reference](#)
- [LoaderManager Reference](#)
- [Processes and Threads Guide](#)
- [Loaders Guide](#)
- UI Thread Performance: [Exceed the Android Speed Limit](#)

# What's Next?

- Concept Chapter: [7.1 AsyncTask and AsyncTaskLoader](#)
- Practical: [7.1 AsyncTask](#)

# END

# Background Tasks

Lesson 7



# 7.2 Internet connection



# Steps to connect to the Internet

1. Add permissions to Android Manifest
2. Check Network Connection
3. Create Worker Thread
4. Implement background task
  - a. Create URI
  - b. Make HTTP Connection
  - c. Connect and GET Data
5. Process results
  - a. Parse Results



# Permissions



# Permissions in AndroidManifest

## Internet

```
<uses-permission android:name="android.permission.INTERNET"/>
```

## Check Network State

```
<uses-permission  
    android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

# Manage Network Connection

Internet  
connection

*This work is licensed under a [Creative Commons Attribution 4.0 International License](#).*



# Getting Network information

- ConnectivityManager
  - Answers queries about the state of network connectivity
  - Notifies applications when network connectivity changes
- NetworkInfo
  - Describes status of a network interface of a given type
  - Mobile or Wi-Fi

# Check if network is available

```
ConnectivityManager connMgr = (ConnectivityManager)
    getSystemService(Context.CONNECTIVITY_SERVICE);

NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();

if (networkInfo != null && networkInfo.isConnected()) {
    // Create background thread to connect and get data
    new DownloadWebpageTask().execute(stringUrl);
} else {
    textView.setText("No network connection available.");
}
```

# Check for WiFi & Mobile

```
NetworkInfo networkInfo =  
    connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);  
boolean isWifiConn = networkInfo.isConnected();  
  
networkInfo =  
    connMgr.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);  
boolean isMobileConn = networkInfo.isConnected();
```

# Worker Thread



# Use Worker Thread

- AsyncTask—very short task, or no result returned to UI
- AsyncTaskLoader—for longer tasks, returns result to UI
- Background Service—later chapter

# Background work

In the background task (for example in `doInBackground()`)

1. Create URI
2. Make HTTP Connection
3. Download Data



# Create URI



# URI = Uniform Resource Identifier

String that names or locates a particular resource

- file://
- http:// and https://
- content://



# Sample URL for Google Books API

[https://www.googleapis.com/books/v1/volumes?  
q=pride+prejudice&maxResults=5&printType=books](https://www.googleapis.com/books/v1/volumes?q=pride+prejudice&maxResults=5&printType=books)

## Constants for Parameters

```
final String BASE_URL =  
    "https://www.googleapis.com/books/v1/volumes?";  
  
final String QUERY_PARAM = "q";  
  
final String MAX_RESULTS = "maxResults";  
  
final String PRINT_TYPE = "printType";
```



# Build a URI for the request

```
Uri builtURI = Uri.parse(BASE_URL).buildUpon()  
    .appendQueryParameter(QUERY_PARAM, "pride+prejudice")  
    .appendQueryParameter(MAX_RESULTS, "10")  
    .appendQueryParameter(PRINT_TYPE, "books")  
    .build();  
  
URL requestURL = new URL(builtURI.toString());
```



# HTTP Client Connection

Internet  
connection

*This work is licensed under a [Creative Commons Attribution 4.0 International License](#).*



# Make a connection from scratch

- Use [HttpURLConnection](#)
- Must be done on a separate thread
- Requires InputStreams and try/catch blocks



# Create a HttpURLConnection

```
HttpURLConnection conn =  
    (HttpURLConnection) requestURL.openConnection();
```

# Configure connection

```
conn.setReadTimeout(10000 /* milliseconds */);  
conn.setConnectTimeout(15000 /* milliseconds */);  
conn.setRequestMethod("GET");  
conn.setDoInput(true);
```

# Connect and get response

```
conn.connect();  
int response = conn.getResponseCode();  
  
InputStream is = conn.getInputStream();  
String contentAsString = convertIsToString(is, len);  
return contentAsString;
```

# Close connection and stream

```
} finally {  
    conn.disconnect();  
    if (is != null) {  
        is.close();  
    }  
}
```

# Convert Response to String



# Convert input stream into a string

```
public String convertIsToString(InputStream stream, int len)
    throws IOException, UnsupportedEncodingException {

    Reader reader = null;
    reader = new InputStreamReader(stream, "UTF-8");
    char[] buffer = new char[len];
    reader.read(buffer);
    return new String(buffer);
}
```

# BufferedReader is more efficient

```
StringBuilder builder = new StringBuilder();
BufferedReader reader =
    new BufferedReader(new InputStreamReader(inputStream));
String line;
while ((line = reader.readLine()) != null) {
    builder.append(line + "\n");
}
if (builder.length() == 0) {
    return null;
}
resultString = builder.toString();
```



# HTTP Client Connection Libraries

Internet  
connection

*This work is licensed under a [Creative Commons Attribution 4.0 International License](#).*



# Make a connection using libraries

- Use a third party library like [OkHttp](#) or [Volley](#)
- Can be called on the main thread
- Much less code

# Volley

```
RequestQueue queue = Volley.newRequestQueue(this);
String url ="http://www.google.com";

StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
        new Response.Listener<String>() {
    @Override
    public void onResponse(String response) {
        // Do something with response
    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {}
});
queue.add(stringRequest);
```



# OkHttp

```
OkHttpClient client = new OkHttpClient();
Request request = new Request.Builder()
    .url("http://publicobject.com/helloworld.txt").build();
client.newCall(request).enqueue(new Callback() {
    @Override
    public void onResponse(Call call, final Response response)
        throws IOException {
        try {
            String responseData = response.body().string();
            JSONObject json = new JSONObject(responseData);
            final String owner = json.getString("name");
        } catch (JSONException e) {}
    }
});
```



# Parse Results



# Parsing the results

- Implement method to receive and handle results  
(`onPostExecute()`)
- Response is often JSON or XML

Parse results using helper classes

- [JSONObject](#), [JSONArray](#)
- [XMLPullParser](#)—parses XML

# JSON basics

```
{  
  "population":1,252,000,000,  
  "country":"India",  
  "cities":["New Delhi","Mumbai","Kolkata","Chennai"]  
}
```

# JSONObject basics

```
JSONObject json0bject = new JSONObject(response);
String nameOfCountry = (String) json0bject.get("country");
long population = (Long) json0bject.get("population");
JSONArray list0fCities = (JSONArray) json0bject.get("cities");
Iterator<String> iterator = list0fCities.iterator();
while (iterator.hasNext()) {
    // do something
}
```

# Another JSON example

```
{"menu": {  
    "id": "file",  
    "value": "File",  
    "popup": {  
        "menuitem": [  
            {"value": "New", "onclick": "CreateNewDoc()"},  
            {"value": "Open", "onclick": "OpenDoc()"},  
            {"value": "Close", "onclick": "CloseDoc()"}  
        ]  
    }  
}
```



# Another JSON example

Get "onclick" value of the 3rd item in the "menuitem" array

```
JSONObject data = new JSONObject(responseString);
JSONArray menuItemArray =
    data.getJSONArray("menuitem");
JSONObject thirdItem =
    menuItemArray.getJSONObject(2);
String onClick = thirdItem.getString("onclick");
```

# Learn more

- [Connect to the Network Guide](#)
- [Managing Network Usage Guide](#)
- [HttpURLConnection reference](#)
- [ConnectivityManager reference](#)
- [InputStream reference](#)



# What's Next?

- Concept Chapter: [7.2 Internet connection](#)
- Practical: [7.2 AsyncTask and AsyncTaskLoader](#)

# END

# Background Tasks

Lesson 7



This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# 7.3 Broadcasts

# Contents

- Broadcasts
- Send a custom broadcasts
- Broadcast receivers
- Implementing broadcast receivers
- Restricting the broadcasts
- Best practices



# Broadcasts



# Broadcasts

Broadcasts are messages sent by Android system and other Android apps, when an event of interest occurs.

Broadcasts are wrapped in an Intent object. This Intent object's contains the event details such as,

`android.intent.action.HEADSET_PLUG`, sent when a wired headset is plugged or unplugged.

# Types of broadcasts

Types of broadcast:

- System broadcast.
- Custom broadcast.



# System broadcasts

System broadcast are the messages sent by the Android system, when a system event occurs, that might affect your app.

Few examples:

- An Intent with action, [ACTION BOOT COMPLETED](#) is broadcasted when the device boots.
- An Intent with action, [ACTION POWER CONNECTED](#) is broadcasted when the device is connected to the external power.

# Custom broadcasts

Custom broadcasts are broadcasts that your app sends out, similar to the Android system.

For example, when you want to let other app(s) know that some data has been downloaded by your app, and its available for their use.

# Send a custom broadcasts



# Send a custom broadcast

Android provides three ways for sending a broadcast:

- Ordered broadcast.
- Normal broadcast.
- Local broadcast.

# Ordered Broadcast

- Ordered broadcast is delivered to one receiver at a time.
- To send a ordered broadcast, use the [sendOrderedBroadcast \(\)](#) method.
- Receivers can propagate result to the next receiver or even abort the broadcast.
- Control the broadcast order with [android:priority](#) attribute in the manifest file.
- Receivers with same priority run in arbitrary order.

# Normal Broadcast

- Delivered to all the registered receivers at the same time, in an undefined order.
- Most efficient way to send a broadcast.
- Receivers can't propagate the results among themselves, and they can't abort the broadcast.
- The [sendBroadcast \(\)](#) method is used to send a normal broadcast.



# Local Broadcast

- Sends broadcasts to receivers within your app.
- No security issues since no interprocess communication.
- To send a local broadcast:
  - To get an instance of LocalBroadcastManager.
  - Call sendBroadcast () on the instance.

```
LocalBroadcastManager.getInstance(this)  
    .sendBroadcast(customBroadcastIntent);
```

# Broadcast Receivers



# What is a broadcast receiver?

- Broadcast receivers are app components.
- They register for various system broadcast and or custom broadcast.
- They are notified (via an Intent):
  - By the system, when an system event occurs that your app is registered for.
  - By another app, including your own if your app is registered for that custom event.

# Register your broadcast receiver

Broadcast receivers can be registered in two ways:

- Static receivers
  - Registered in your `AndroidManifest.xml`, also called as Manifest-declared receivers.
- Dynamic receivers
  - Registered using app or activities' context in your Java files, also called as Context-registered receivers.

# Receiving a system broadcast

- Starting from Android 8.0 (API level 26), static receivers can't receive most of the system broadcasts.
- Use a dynamic receiver to register for these broadcasts.
- If you register for the system broadcasts in the manifest, the Android system won't deliver them to your app.
- A few broadcasts, are excepted from this restriction. See the complete list of [implicit broadcast exceptions](#).

# Implementing Broadcast Receivers



# To create a broadcast receiver

- Subclass the [BroadcastReceiver](#) class and override its `onReceive()` method.
- Register the broadcast receiver and specify the intent-filters:
  - Statically, in the Manifest.
  - Dynamically, with `registerReceiver()`.

# What are Intent-filters

Intent-filters specify the types of intents a broadcast receiver can receive. They filter the incoming intents based on the Intent values like action.

To add an intent-filter:

- To your `AndroidManifest.xml` file, use `<intent-filter>` tag.
- To your Java file use the `IntentFilter` object.

# Subclass a broadcast receiver

In Android studio, **File > New > Other > BroadcastReceiver**

```
public class CustomReceiver extends BroadcastReceiver {  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        // This method is called when the BroadcastReceiver  
        // is receiving an Intent broadcast.  
        throw new UnsupportedOperationException("Not yet implemented");  
    }  
}
```

# Implement onReceive()

Example implementation of `onReceive()` method which handles power connected and disconnected.

```
@Override  
public void onReceive(Context context, Intent intent) {  
    String intentAction = intent.getAction();  
    switch (intentAction){  
        case Intent.ACTION_POWER_CONNECTED:  
            break;  
        case Intent.ACTION_POWER_DISCONNECTED:  
            break;  
    }  
}
```



# Register statically in Android manifest

- `<receiver>` element inside `<application>` tag.
- `<intent-filter>` registers receiver for specific intents.

```
<receiver  
    android:name=".CustomReceiver"  
    android:enabled="true"  
    android:exported="true">  
    <intent-filter>  
        <action android:name="android.intent.action.BOOT_COMPLETED"/>  
    </intent-filter>  
</receiver>
```



# Register dynamically

- Register your receiver in `onCreate()` or `onResume()`.

```
// Register the receiver using the activity context.  
this.registerReceiver(mReceiver, filter);
```

- Unregister in `onDestroy()` or `onPause()`.

```
// Unregister the receiver  
this.unregisterReceiver(mReceiver);
```

# Register a Local broadcast receiver

Register local receivers dynamically, because static registration in the manifest is not possible for a local broadcasts.

# Register a Local broadcast receiver

To register a receiver for local broadcasts:

- Get an instance of `LocalBroadcastManager`.
- Call `registerReceiver()`.

```
LocalBroadcastManager.getInstance(this).registerReceiver  
(mReceiver,  
    new IntentFilter(CustomReceiver.ACTION_CUSTOM_BROADCAST));
```

# Unregister a Local broadcast receiver

To unregister a local broadcast receiver:

- Get an instance of the LocalBroadcastManager.
- Call [LocalBroadcastManager.unregisterReceiver\(\)](#).

```
LocalBroadcastManager.getInstance(this)  
    .unregisterReceiver(mReceiver);
```

# Restricting broadcasts



# Restricting broadcasts

- Restricting your broadcast is strongly recommended.
- An unrestricted broadcast can pose a security threat.
- For example: If your apps' broadcast is not restricted and includes sensitive information, an app that contains malware could register and receive your data.

# Ways to restrict a broadcast

- If possible, use a [LocalBroadcastManager](#), which keeps the data inside your app, avoiding security leaks.
- Use the [setPackage\(\)](#) method and pass in the package name. Your broadcast is restricted to apps that match the specified package name.
- Access permissions can be enforced by sender or receiver.

# Enforce permissions by sender

To enforce a permission when sending a broadcast:

- Supply a non-null permission argument to `sendBroadcast()`.
- Only receivers that request this permission using the `<uses-permission>` tag in their `AndroidManifest.xml` file can receive the broadcast.

# Enforce permissions by receiver

To enforce a permission when receiving a broadcast:

- If you register your receiver dynamically, supply a non-null permission to `registerReceiver()`.
- If you register your receiver statically, use the `android:permission` attribute inside the `<receiver>` tag in your `AndroidManifest.xml`.

# Best practices



# Best practices

- Make sure namespace for intent is unique and you own it.
- Restrict broadcast receivers.
- Other apps can respond to broadcast your app sends – use permissions to control this.
- Prefer dynamic receivers over static receivers.
- Never perform a long running operation inside your broadcast receiver.



# Learn more

- [BroadcastReceiver Reference](#)
- [Intents and Intent Filters Guide](#)
- [LocalBroadcastManager Reference](#)
- [Broadcasts overview](#)



# What's Next?

- Concept Chapter: [7.3 Broadcasts](#)
- Practical: [7.3 Broadcast Receivers](#)



# END

# Background Tasks

Lesson 7



This work is licensed under a [Creative Commons Attribution 4.0 International License](#).

# 7.4 Services

# Contents

- Services for long tasks.
- IntentService

# Services is an advanced topic

- Services are complex.
- Many ways of configuring a service.
- This lesson has introductory information only.
- Explore and learn for yourself if you want to use services.

# Services for Long Tasks

# What is a service?

A Service is an application component that can perform long-running operations in the background and does not provide a user interface.



# What are services good for?

- Network transactions.
- Play music.
- Perform file I/O.
- Interact with a database.

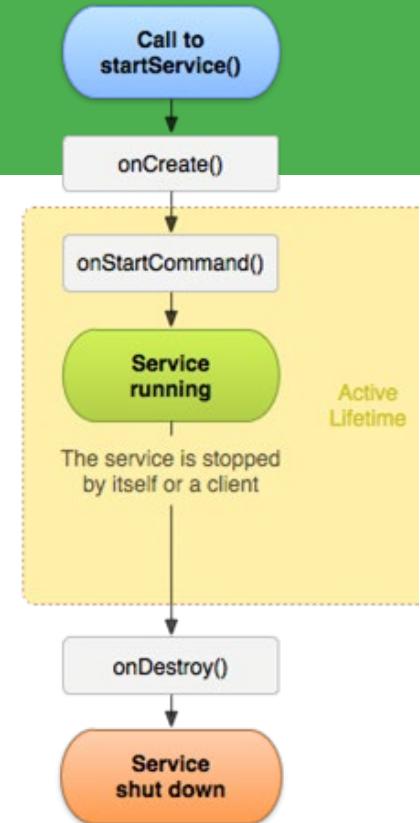


# Characteristics of services

- Started with an Intent.
- Can stay running when user switches applications.
- Lifecycle—which you must manage.
- Other apps can use the service—manage permissions.
- Runs in the main thread of its hosting process.

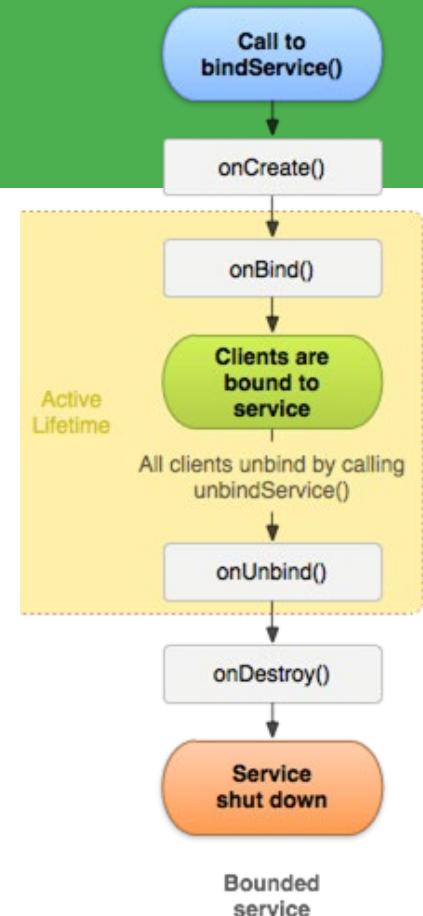
# Forms of services: started

- Started with `startService()`
- Runs indefinitely until it stops itself
- Usually does not update the UI



# Forms of services: bound

- Offers a client-server interface that allows components to interact with the service
- Clients send requests and get results
- Started with `bindService()`
- Ends when all clients unbind



# Services and threads

Although services are separate from the UI, they still run on the main thread by default (except IntentService)

Offload CPU-intensive work to a separate thread within the service

# Updating the app

If the service can't access the UI, how do you update the app to show the results?

Use a broadcast receiver!

# Foreground services

Runs in the background but requires that the user is actively aware it exists—e.g. music player using music service

- Higher priority than background services since user will notice its absence—unlikely to be killed by the system
- Must provide a notification which the user cannot dismiss while the service is running

# Background services limitations

- Starting from API 26, background app is not allowed to create a background service.
- A foreground app, can create and run both foreground and background services.
- When an app goes into the background, the system stops the app's background services.
- The `startService()` method now throws an [IllegalStateException](#) if an app is targeting API 26.
- These limitations don't affect foreground services or bound services.

# Creating a service

- <service android:name=".ExampleService" />
- Manage permissions.
- Subclass IntentService or Service class.
- Implement lifecycle methods.
- Start service from Activity.
- Make sure service is stoppable.



# Stopping a service

- A **started service** must manage its own lifecycle
- If not stopped, will keep running and consuming resources
- The service must stop itself by calling [stopSelf\(\)](#)
- Another component can stop it by calling [stopService\(\)](#)
- **Bound service** is destroyed when all clients unbound
- **IntentService** is destroyed after `onHandleIntent()` returns



# IntentService

# IntentService

- Simple service with simplified lifecycle
- Uses worker threads to fulfill requests
- Stops itself when done
- Ideal for one long task on a single background thread

# IntentService Limitations

- Cannot interact with the UI
- Can only run one request at a time
- Cannot be interrupted

# IntentService restrictions

- IntentService are subjected to the new restrictions on background services.
- For the apps targeting API 26, Android Support Library 26.0.0 introduces a new JobIntentService.
- JobIntentService provides the same functionality as IntentService but uses jobs instead of services.

# IntentService Implementation

```
public class HelloIntentService extends IntentService {  
    public HelloIntentService() { super("HelloIntentService");}  
  
    @Override  
    protected void onHandleIntent(Intent intent) {  
        try {  
            // Do some work  
        } catch (InterruptedException e) {  
            Thread.currentThread().interrupt();  
        }  
    } // When this method returns, IntentService stops the service, as appropriate.  
}
```



# Learn more

- [Services overview](#)
- [Background Execution Limits](#)



# What's Next?

- Concept Chapter: [7.4 Services](#)
- No practical

# END

# Alarms and Schedulers

Lesson 8



# 8.1 Notifications

# Contents

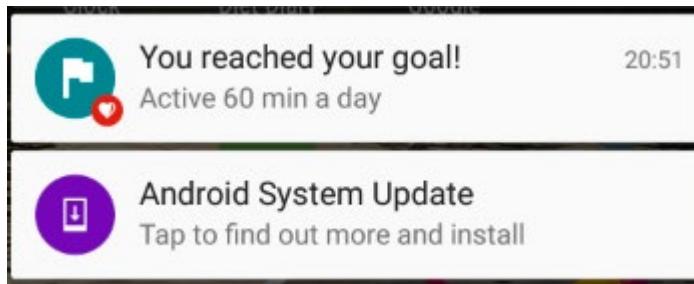
- What are notifications?
- Notification channels
- Creating a notification channel
- Creating notifications
- Tap action and action buttons
- Expanded view notifications
- Delivering notifications
- Managing Notifications



# What Are Notifications?

# What is a notification?

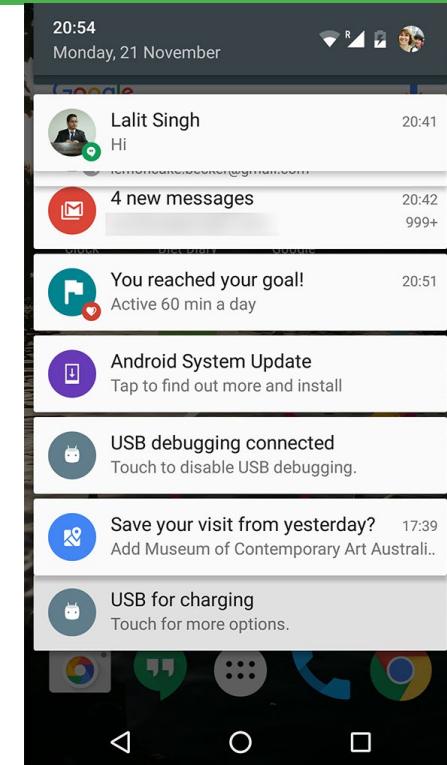
Message displayed to user outside regular app UI



- Small icon
- Title
- Detail text

# How are notifications used?

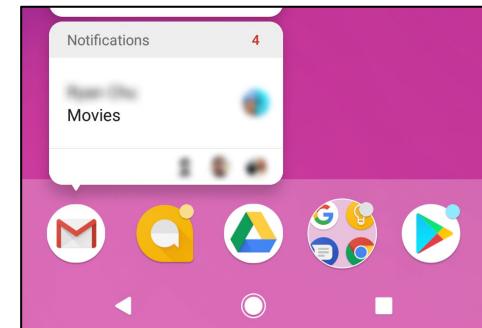
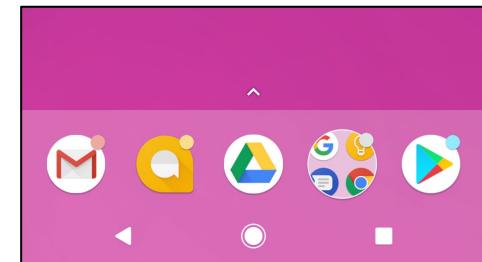
- Android issues a notification that appears as icon on the status bar.
- To see details, user opens the notification drawer.
- User can view notifications any time in the notification drawer.



# App icon badge

Available only on the devices running Android 8.0 (API level 26) and higher.

- New notifications are displayed as a colored "badge" (also known as a "notification dot") on the app icon.
- Users can long-press on an app icon to see the notifications for that app. Similar to the notification drawer.



# Notification Channels

# Notification channels

- Used to create a user-customizable channel for each type of notification to be displayed.
- More than one notification can be grouped in to a channel.
- Set notification behavior like sound, light, vibrate and so on, applied to all the notifications in that channel.

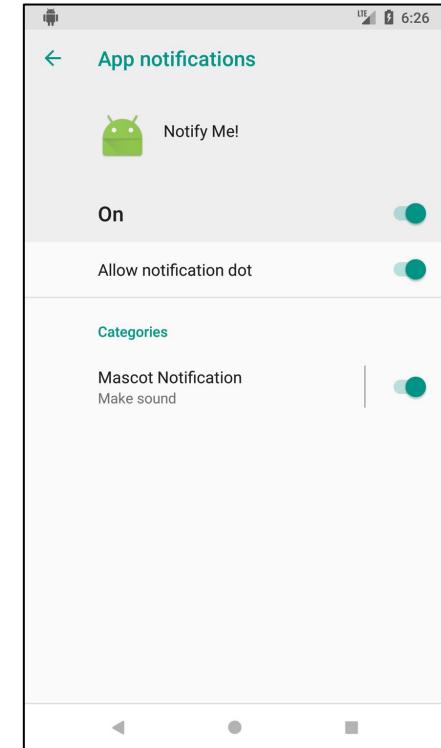


# Notification channels are mandatory

- [Notification channels](#) are introduced in Android 8.0 (API level 26)
- All notifications must be assigned to a channel starting from Android 8.0 (API level 26), else your notifications will not be displayed.
- For the apps targeting lower than Android 8.0 (API level 26), no need to implement notification channels.

# Notification channels in Settings

- Notification channels appear as **Categories** under **App notifications** in the device Settings.



# Creating a Notification channel

# Create a Notification channel

- Notification channel instance is created using [NotificationChannel](#) constructor.
- You must specify:
  - An ID that's unique within your package.
  - User visible name of the channel.
  - The importance level for the channel.

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {  
    NotificationChannel notificationChannel =  
        new NotificationChannel(CHANNEL_ID, "Mascot Notification",  
        NotificationManager.IMPORTANCE_DEFAULT);  
}
```



# Importance level

- Available in Android 8.0 (API level 26) and higher.
- Sets the intrusion level, like the sound and visibility for all notifications posted in the channel.
- Range from IMPORTANCE\_NONE (0) to IMPORTANCE\_HIGH (4).
- To support earlier versions of Android (Lower than API level 26), set the priority.

# Notification priority

- Determines how the system displays the notification with respect to other notifications, in Android version Lower than API level 26.
- Set using the `setPriority()` method for each notification.
- Range from `PRIORITY_MIN` to `PRIORITY_MAX`.

```
setPriority(NotificationCompat.PRIORITY_HIGH)
```

# Importance level and priority constants

User-visible importance level	Importance (Android 8.0 and higher)	Priority (Android 7.1 and lower)
<b>Urgent</b> Makes a sound and appears as a heads-up notification	<a href="#"><u>IMPORTANCE_HIGH</u></a>	<a href="#"><u>PRIORITY_HIGH</u></a> or <a href="#"><u>PRIORITY_MAX</u></a>
<b>High</b> Makes a sound	<a href="#"><u>IMPORTANCE_DEFAULT</u></a>	<a href="#"><u>PRIORITY_DEFAULT</u></a>
<b>Medium</b> No sound	<a href="#"><u>IMPORTANCE_LOW</u></a>	<a href="#"><u>PRIORITY_LOW</u></a>
<b>Low</b> No sound and doesn't appear in the status bar	<a href="#"><u>IMPORTANCE_MIN</u></a>	<a href="#"><u>PRIORITY_MIN</u></a>

# Creating Notifications

# Creating Notification

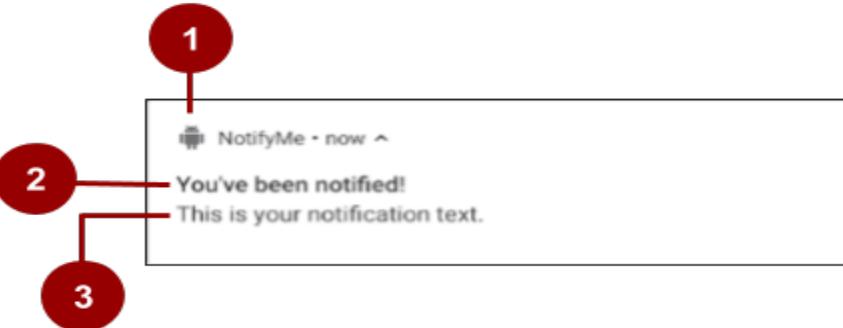
- Notification is created using `NotificationCompat.Builder` class.
- Pass the application context and notification channel ID to the constructor.
- The `NotificationCompat.Builder` constructor takes the notification channel ID, this is only used by Android 8.0 (API level 26) and higher, but this parameter is ignored by the older versions.

```
NotificationCompat.Builder mBuilder = new  
    NotificationCompat.Builder(this, CHANNEL_ID);
```

# Setting notification contents

1. A small icon, set by [setSmallIcon\(\)](#).

This is the only content that's required.



1. A title, set by [setContentTitle\(\)](#).

2. The body text, set by

[setContentText\(\)](#). This is the

notification message.

# Setting notification contents

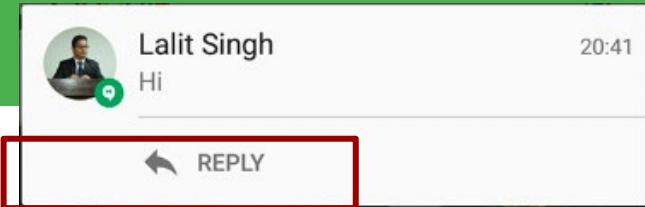
```
NotificationCompat.Builder mBuilder =  
    new NotificationCompat.Builder(this, CHANNEL_ID)  
        .setSmallIcon(R.drawable.android_icon)  
        .setContentTitle("You've been notified!")  
        .setContentText("This is your notification text.");
```

# Tap action and Action buttons

# Add notification tap action

- Every notification must respond when it is tapped, usually launching an Activity in your app.
- Set an content intent using `setContentIntent()` method.
- Pass the Intent wrapped in a `PendingIntent` object.

# Notification action buttons



- Action buttons can perform a variety of actions on behalf of your app, such as starting a background task, placing a phone call and so on.
- Starting from Android 7.0 (API level 24) reply to messages directly from notifications.
- To add an action button, pass a `PendingIntent` to the `addAction()` method.

# Pending intents

- A [PendingIntent](#) is a description of an intent and target action to perform with it.
- Give a PendingIntent to another application to grant it the right to perform the operation you have specified as if the other app was yourself.

# Methods to create a PendingIntent

To instantiate a PendingIntent, use one of the following methods:

- PendingIntent.getActivity()
- PendingIntent.getBroadcast()
- PendingIntent.getService()

# PendingIntent method arguments

1. Application context
2. Request code—constant integer id for the pending intent
3. Intent to be delivered
4. PendingIntent flag determines how the system handles multiple pending intents from same app

# Step 1: Create intent

```
Intent notificationIntent =  
    new Intent(this, MainActivity.class);
```

# Step 2: Create PendingIntent

```
PendingIntent notificationPendingIntent =  
    PendingIntent.getActivity(  
        this,  
        NOTIFICATION_ID,  
        notificationIntent,  
        PendingIntent.FLAG_UPDATE_CURRENT);
```

# Step 3: Add to notification builder

To set tap action to the notification:

```
.setContentIntent(notificationPendingIntent);
```

# Add action buttons



- Use `NotificationCompat.Builder.addAction()`
  - pass in icon, caption, PendingIntent

```
.addAction(R.drawable.ic_color_lens_black_24dp,  
          "R.string.label",  
          notificationPendingIntent);
```

# Expanded view notifications

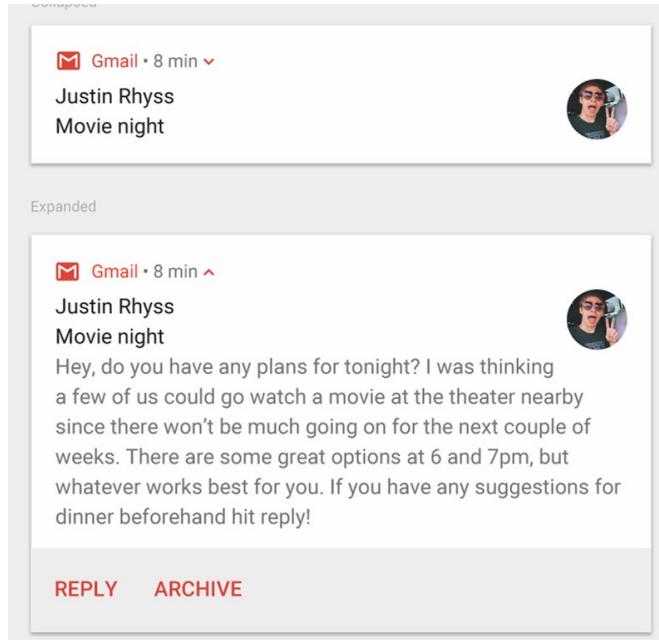
# Expandable notifications

- Notifications in the notification drawer appear in two main layouts, normal view (which is the default) and expanded view.
- Expanded view notifications were introduced in Android 4.1.
- Use them sparingly – they take up more space and attention.

# Big text

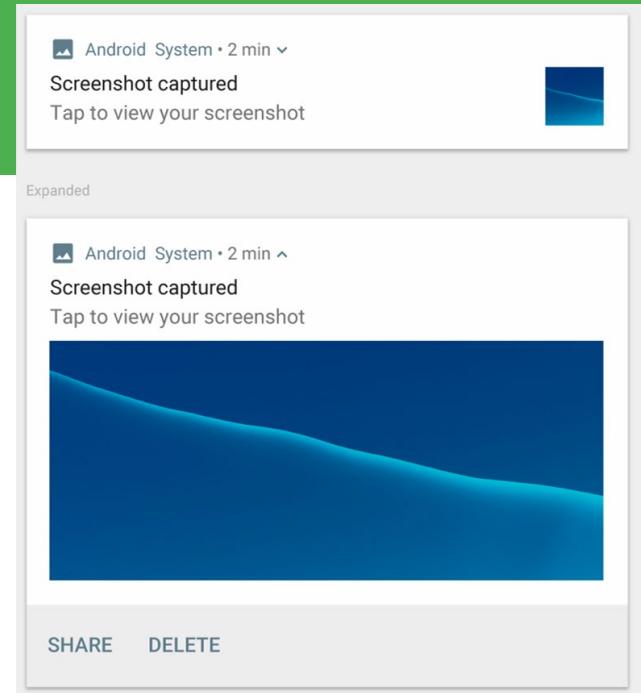
- For large-format notifications that include a lot of text.
- Fits more text than a standard view.
- Use the helper class:

[NotificationCompat.BigTextStyle](#)



# Big image

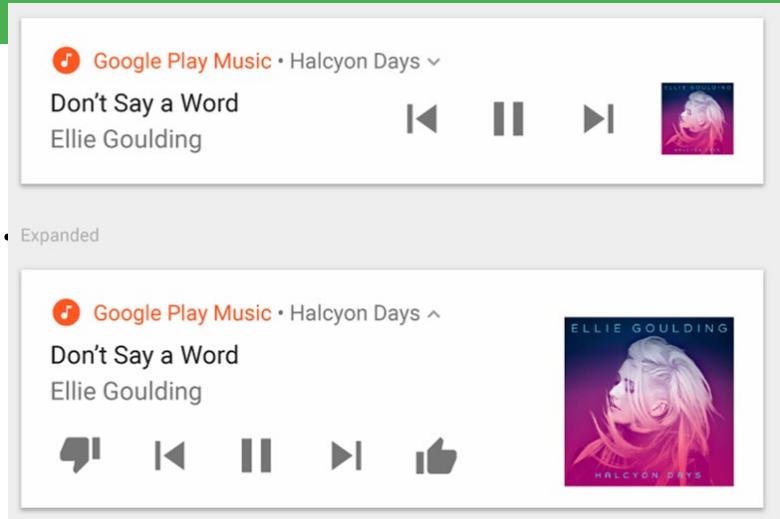
- For large-format notifications that include a large image attachment.
- Use the helper class:



[NotificationCompat.BigPictureStyle](#)

# Media

- For media playback notifications.
- Actions for controlling media such as music
- Image for album cover
- Use the helper class:
- [NotificationCompat.MediaStyle](#)



# Setting styles

To create expandable notification that appear, use one of the helper classes to set the style using the [setStyle\(\)](#) method.

```
mNotifyBuilder  
    .setStyle(new NotificationCompat.BigPictureStyle\(\)  
        .bigPicture(myBitmapImage)  
        .setBigContentTitle("Notification!"));
```

# Delivering Notifications

# Delivering notifications

- Use the [NotificationManager](#) class to deliver notifications.
  - Create an instance of NotificationManager
  - Call `notify()` to deliver the notification.

# Instantiate NotificationManager

Call `getSystemService()`, passing in the [NOTIFICATION\\_SERVICE](#) constant.

```
mNotifyManager = (NotificationManager)  
    getSystemService(NOTIFICATION_SERVICE);
```

# Send notification

- Call `notify()` to deliver the notification, passing in these two values:
  - A notification ID, which is used to update or cancel the notification.
  - The `NotificationCompat` object that you created using the `NotificationCompat.Builder` object.

```
mNotifyManager.notify(NOTIFICATION_ID, myNotification);
```

# Managing Notifications

# Updating notifications

1. Update a notification by changing and or adding some of its content.
2. Issue notification with updated parameters using builder.
3. Call `notify()` passing in the same notification ID.
  - If previous notification is still visible, system updates.
  - If previous notification has been dismissed, new notification is delivered.

# Canceling notifications

Notifications remain visible until:

- User dismisses it by swiping or by using "Clear All".
- Calling `setAutoCancel ()` when creating the notification, removes it from the status bar when the user clicks on it.
- App calls `cancel ()` or `cancelAll ()` on `NotificationManager`.

```
mNotifyManager.cancel(NOTIFICATION_ID);
```

# Design guidelines

If your app sends too many notifications, users will disable notifications or uninstall the app.

- **Relevant:** Whether this information is essential for the user.
- **Timely:** Notifications need to appear when they are useful.
- **Short:** Use as few words as possible.
- Give users the power to choose -- Use appropriate notification channels to categorise your notifications.

# What's Next?

- Concept Chapter: [8.1 Notifications](#)
- Practical: [8.1 Notifications](#)



# The End



# Alarms and Schedulers

Lesson 8



This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# 8.2 Alarms

# Contents

- What are Alarms
- Alarms Best Practices
- Alarm Manager
- Scheduling Alarms
- More Alarm Considerations



# What Are Alarms

# What is an alarm in Android?



- Not an actual alarm clock.
- Schedules something to happen at a set time.
- Fire intents at set times or intervals.
- Goes off once or recurring.
- Can be based on a real-time clock or elapsed time.
- App does not need to run for alarm to be active.

# How alarms work with components

Stand Up!

Stand Up Alarm

ON

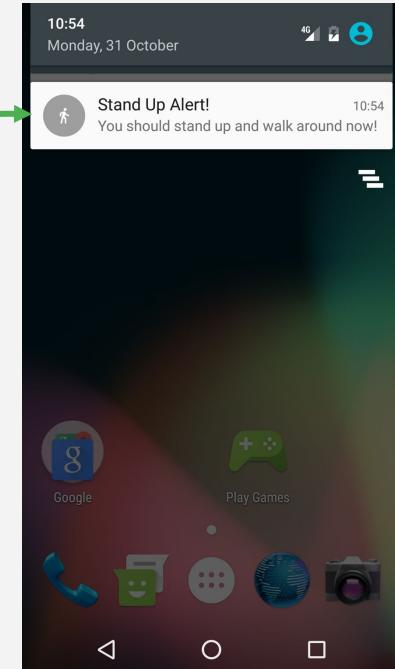
Stand Up Alarm On!

Activity creates  
a notification and  
sets an alarm.



Alarm triggers and  
sends out Intent.  
App may be  
destroyed so....

BroadcastReceiver  
wakes up the app  
and delivers the  
notification.



# Benefits of alarms

- App does not need to run for alarm to be active.
- Device does not have to be awake.
- Does not use resources until it goes off.
- Use with BroadcastReceiver to start services and other operations.



# Measuring time

- Elapsed Real Time—time since system boot.
  - Independent of time zone and locale.
  - Use for intervals and relative time.
  - Use whenever possible.
  - Elapsed time includes time device was asleep.
- Real Time Clock (RTC)—UTC (wall clock) time.
  - When time of day at locale matter.

# Wakeup behavior

- Wakes up device CPU if screen is off.
  - Use only for time critical operations.
  - Can drain battery.
- Does not wake up device.
  - Fires next time device is awake.
  - Is polite.



# Types of alarms

	<b>Elapsed Real Time (ERT)–since system boot</b>	<b>Real Time Clock (RTC)–time of day matters</b>
Do not wake up device	<u>ELAPSED REALTIME</u>	<u>RTC</u>
Wake up	<u>ELAPSED REALTIME WAKEUP</u>	<u>RTC WAKEUP</u>

# Alarms Best Practices

# If everybody syncs at the same time...

Imagine an app with millions of users:

- Server sync operation based on clock time.
- Every instance of app syncs at 11:00 p.m.

 Load on the server could result in high latency or even "denial of service"

# Alarm Best Practices

- Add randomness to network requests on alarms.
- Minimize alarm frequency.
- Use `ELAPSED_REALTIME`, not clock time, if you can.

# Battery

- Minimize waking up the device.
- Use inexact alarms.
  - Android synchronizes multiple inexact repeating alarms and fires them at the same time.
  - Reduces the drain on the battery.
  - Use [setInexactRepeating\(\)](#) instead of [setRepeating\(\)](#).

# When not to use an alarm

- Ticks, timeouts, and while app is running—[Handler](#).
- Server sync—[SyncAdapter](#) with Cloud Messaging Service.
- Inexact time and resource efficiency—[JobScheduler](#).



# AlarmManager

# What is AlarmManager

- AlarmManager provides access to system alarm services.
- Schedules future operation.
- When alarm goes off, registered Intent is broadcast.
- Alarms are retained while device is asleep.
- Firing alarms can wake device.

# Get an AlarmManager

```
AlarmManager alarmManager =  
    (AlarmManager) getSystemService(ALARM_SERVICE);
```

# Scheduling Alarms

# What you need to schedule an alarm

1. Type of alarm.
2. Time to trigger.
3. Interval for repeating alarms.
4. PendingIntent to deliver at the specified time  
(just like notifications).

# Schedule a single alarm

- set()—single, inexact alarm.
- setWindow()—single inexact alarm in window of time.
- setExact()—single exact alarm.

More power saving options AlarmManager API 23+.

# Schedule a repeating alarm

- [setInexactRepeating\(\)](#)
  - repeating, inexact alarm.
- [setRepeating\(\)](#)
  - Prior to API 19, creates a repeating, exact alarm.
  - After API 19, same as setInexactRepeating().

# `setInexactRepeating()`

`setInexactRepeating(`

`int alarmType,`

`long triggerAtMillis,`

`long intervalMillis,`

`PendingIntent operation)`

# Create an inexact alarm

```
alarmManager.setInexactRepeating(  
    AlarmManager.ELAPSED_REALTIME_WAKEUP,  
    SystemClock.elapsedRealtime()  
        + AlarmManager.INTERVAL_FIFTEEN_MINUTES,  
    AlarmManager.INTERVAL_FIFTEEN_MINUTES,  
    notifyPendingIntent);
```

# More Alarm Considerations

# Checking for an existing alarm

```
boolean alarmExists =  
    (PendingIntent.getBroadcast(this,  
        0, notifyIntent,  
        PendingIntent.FLAG_NO_CREATE) != null);
```

# Doze and Standby

- Doze—completely stationary, unplugged, and idle device.
- Standby—unplugged device on idle apps.
- Alarms will not fire.
- API 23+.

# User visible alarms

- `setAlarmClock()`
- System UI may display time/icon.
- Precise.
- Works when device is idle.
- App can retrieve next alarm with `getNextAlarmClock()`.
- API 21+.



# Cancel an alarm

- Call `cancel()` on the `AlarmManager`
  - pass in the `PendingIntent`.

```
alarmManager.cancel(alarmPendingIntent);
```

# Alarms and Reboots

- Alarms are cleared when device is off or rebooted.
- Use a BroadcastReceiver registered for the BOOT\_COMPLETED event and set the alarm in the onReceive() method.

# Learn more

- [Schedule Repeating Alarms Guide](#)
- [AlarmManager reference](#)
- [Choosing an Alarm Blog Post](#)
- [Scheduling Alarms Presentation](#)
- [Optimizing for Doze and Standby](#)



# What's Next?

- Concept Chapter: [8.2 Alarms](#)
- Practical: [8.2 The Alarm Manager](#)

# END

# Alarms and Schedulers

Lesson 8



Efficient data  
transfer

*This work is licensed under a [Creative Commons Attribution 4.0 International License](#).*



# 8.3 Efficient data transfer and JobScheduler

# Contents

- Transferring Data Efficiently
- Job Scheduler
  - JobService
  - JobInfo
  - JobScheduler

# Transferring Data Efficiently

Efficient data  
transfer

*This work is licensed under a [Creative Commons Attribution 4.0 International License](#).*



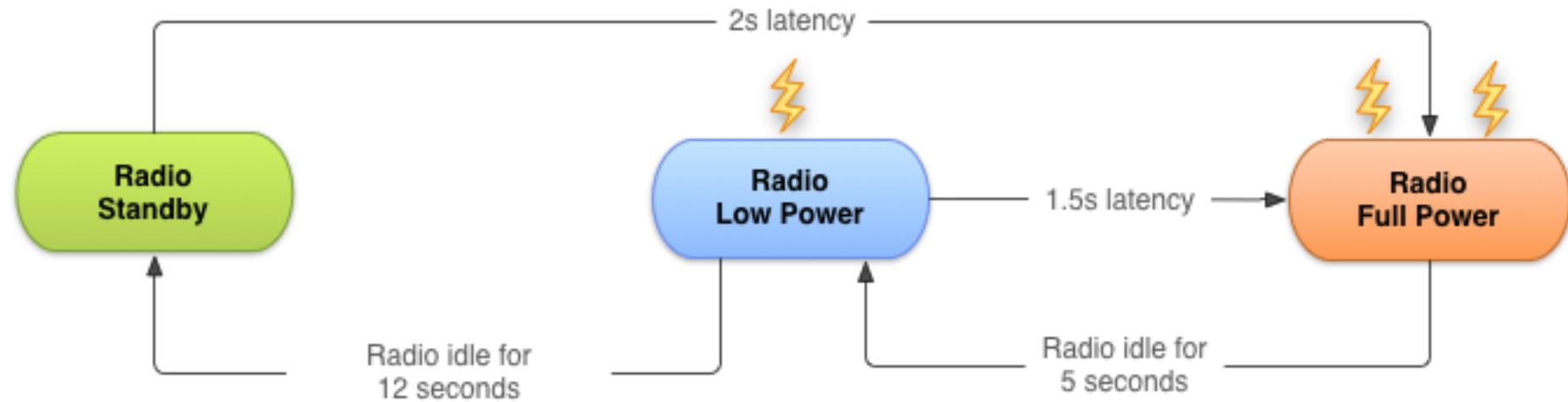
# Transferring data uses resources

- Wireless radio uses battery.
  - Device runs out of battery.
  - Need to let device charge.
- Transferring data uses up data plans.
  - Costing users real money (for free apps...).

# Wireless radio power states

- Full power—Active connection, highest rate data transfer.
- Low power—Intermediate state that uses 50% less power.
- Standby—Minimal energy, no active network connection.

# Wireless radio state transitions for 3G

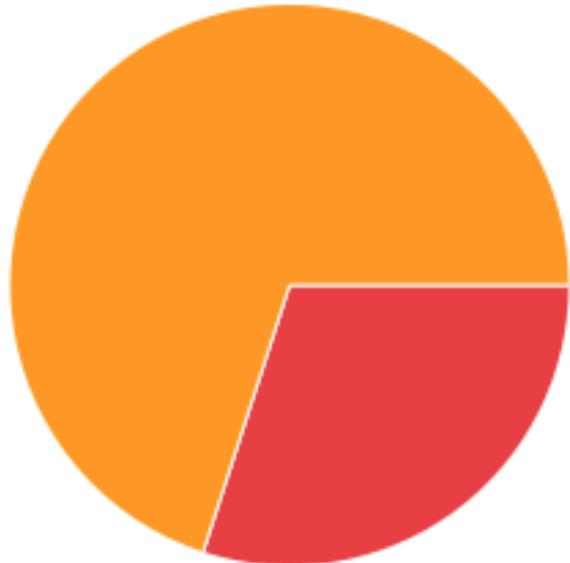


# Bundle network transfers

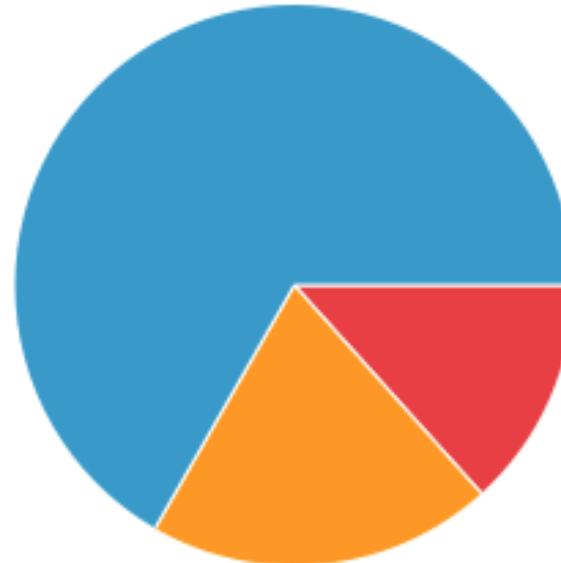
- For a typical 3G device, every data transfer session, the radio draws energy for almost 20 seconds.
- Send data for 1s every 18s—radio mostly on full power.
- Send data in bundles of 3s—radio mostly idle.
- Bundle your data transfers.

# Bundled vs. unbundled

Unbundled Transfers



Bundled Transfers



- High Power
- Low Power
- Idle

Efficient data transfer

*This work is licensed under a [Creative Commons Attribution 4.0 International License](#).*



# Prefetch data

- Download all the data you are likely to need for a given time period in a single burst, over a single connection, at full capacity.
- If you guess right, reduces battery cost and latency.
- If you guess wrong, may use more battery and data bandwidth.

# Monitor connectivity state

- Wi-Fi radio uses less battery and has more bandwidth than wireless radio.
- Use [ConnectivityManager](#) to determine which radio is active and adapt your strategy.

# Monitor battery state

- Wait for specific conditions to initiate battery intensive operation.
- [BatteryManager](#) broadcasts all battery and charging details in a broadcast [Intent](#).
- Use a `BroadcastReceiver` registered for battery status actions.

# Job Scheduler

Efficient data  
transfer

*This work is licensed under a [Creative Commons Attribution 4.0 International License](#).*



# What is Job Scheduler

- Used for intelligent scheduling of background tasks.
- Based on conditions, not a time schedule.
- Much more efficient than AlarmManager.
- Batches tasks together to minimize battery drain.
- API 21+ (not in support library).

# Job Scheduler components

- JobService—Service class where the task is initiated.
- JobInfo—Builder pattern to set the conditions for the task.
- JobScheduler—Schedule and cancel tasks, launch service.

# JobService

Efficient data  
transfer

*This work is licensed under a [Creative Commons Attribution 4.0 International License](#).*



# JobService

- JobService subclass, implement your task here.
- Override
  - [onStartJob\(\)](#)
  - [onStopJob\(\)](#)
- Runs on the main thread.



# onStartJob()

- Implement work to be done here.
- Called by system when conditions are met.
- Runs on main thread.
- **Off-load heavy work to another thread.**



# onStartJob() returns a boolean

**FALSE**—Job finished.

**TRUE**

- Work has been offloaded.
- Must call `jobFinished()` from the worker thread.
- Pass in `JobParams` object from `onStartJob()`.

# onStopJob()

- Called if system has determined execution of job must stop.
- ... because requirements specified no longer met.
- For example, no longer on Wi-Fi, device not idle anymore.
- Before `jobFinished(JobParameters, boolean)`.
- Return TRUE to reschedule.

# Basic JobService code

```
public class MyJobService extends JobService {  
    private UpdateAppsAsyncTask updateTask = new UpdateAppsAsyncTask();  
    @Override  
    public boolean onStartJob(JobParameters params) {  
        updateTask.execute(params);  
        return true; // work has been offloaded  
    }  
    @Override  
    public boolean onStopJob(JobParameters jobParameters) {  
        return true;  
    }  
}
```

# Register your JobService

```
<service  
    android:name=".NotificationJobService"  
    android:permission=  
        "android.permission.BIND_JOB_SERVICE"/>
```

# JobInfo



# JobInfo

- Set conditions of execution.
- [JobInfo.Builder](#) object.

# JobInfo builder object

- Arg 1: Job ID
- Arg 2: Service component
- Arg 3: JobService to launch

```
JobInfo.Builder builder = new JobInfo.Builder(  
    JOB_ID,  
    new ComponentName(getApplicationContext(),  
    NotificationJobService.class.getName()));
```

# Setting conditions

[setRequiredNetworkType\(int networkType\)](#)

[setBackoffCriteria\(long initialBackoffMillis, int backoffPolicy\)](#)

[setMinimumLatency\(long minLatencyMillis\)](#)

[setOverrideDeadline\(long maxExecutionDelayMillis\)](#)

[setPeriodic\(long intervalMillis\)](#)

[setPersisted\(boolean isPersisted\)](#)

[setRequiresCharging\(boolean requiresCharging\)](#)

[setRequiresDeviceIdle\(boolean requiresDeviceIdle\)](#)



# `setRequiredNetworkType()`

`setRequiredNetworkType(int networkType)`

- `NETWORK_TYPE_NONE`—Default, no network required.
- `NETWORK_TYPE_ANY`—Requires network connectivity.
- `NETWORK_TYPE_NOT_ROAMING`—Requires network connectivity that is not roaming.
- `NETWORK_TYPE_UNMETERED`—Requires network connectivity that is unmetered.

# setMinimumLatency()

`setMinimumLatency(long minLatencyMillis)`

- Minimum milliseconds to wait before completing task.

# setOverrideDeadline()

`setOverrideDeadline(long maxExecutionDelayMillis)`

- Maximum milliseconds to wait before running the task, even if other conditions aren't met.

# setPeriodic()

## setPeriodic(long intervalMillis)

- Repeats task after a certain amount of time.
- Pass in repetition interval.
- Mutually exclusive with minimum latency and override deadline conditions.
- Task is not guaranteed to run in the given period.



# setPersisted()

## setPersisted(boolean isPersisted)

- Sets whether the job is persisted across system reboots.
- Pass in True or False.
- Requires RECEIVE\_BOOT\_COMPLETED permission.

# setRequiresCharging()

[setRequiresCharging\(boolean requiresCharging\)](#)

- Whether device must be plugged in.
- Pass in True or False.
- Defaults to False.



# setRequiresDeviceIdle()

[setRequiresDeviceIdle\(boolean requiresDeviceIdle\)](#)

- Whether device must be in idle mode.
- Idle mode is a loose definition by the system, when device is not in use, and has not been for some time.
- Use for resource-heavy jobs.
- Pass in True or False. Defaults to False.

# JobInfo code

```
JobInfo.Builder builder = new JobInfo.Builder(  
    JOB_ID, new ComponentName(getApplicationContext(),  
    NotificationJobService.class.getName()))  
    .setRequiredNetworkType(JobInfo.NETWORK_TYPE_UNMETERED)  
    .setRequiresDeviceIdle(true)  
    .setRequiresCharging(true);  
  
JobInfo myJobInfo = builder.build();
```



# JobScheduler

Efficient data  
transfer

*This work is licensed under a [Creative Commons Attribution 4.0 International License](#).*



# Scheduling the job

1. Obtain a JobScheduler object form the system.
2. Call schedule() on JobScheduler, with JobInfo object.

```
mScheduler =  
    (JobScheduler) getSystemService(JOB_SCHEDULER_SERVICE);  
  
mScheduler.schedule(myJobInfo);
```

# Resources

- [Transferring Data Without Draining the Battery Guide](#)
- [Optimizing Downloads for Efficient Network Access Guide](#)
- [Modifying your Download Patterns Based on the Connectivity Type Guide](#)
- [JobScheduler Reference](#)
- [JobService Reference](#)
- [JobInfo Reference](#)
- [JobInfo.Builder Reference](#)
- [JobParameters Reference](#)
- [Presentation on Scheduling Tasks](#)



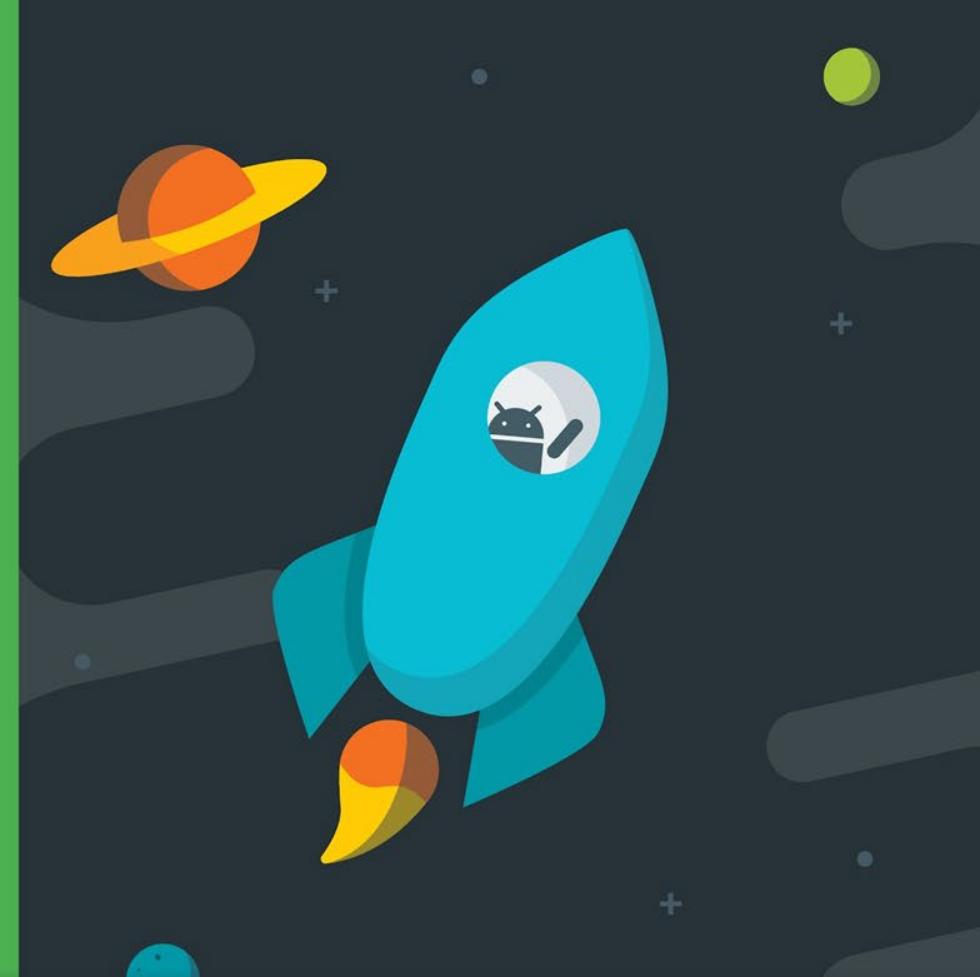
# What's Next?

- Concept Chapter: [8.3 Efficient data transfer](#)
- Practical: [8.3 Job Scheduler](#)

# END

# Preferences and settings

Lesson 9



This work is licensed under a [Creative Commons Attribution 4.0 International License](#).

# 9.0 Data Storage

# Contents

- Android File System
- Internal Storage
- External Storage
- SQLite Database
- Other Storage Options



# Storage Options

# Storing data

- [Shared Preferences](#)—Private primitive data in key-value pairs
- [Internal Storage](#)—Private data on device memory
- [External Storage](#)—Public data on device or external storage
- [SQLite Databases](#)—Structured data in a private database
- [Content Providers](#)—Store privately and make available publicly

# Storing data beyond Android

- [Network Connection](#)—On the web with your own server
- [Cloud Backup](#)—Back up app and user data in the cloud
- [Firebase Realtime Database](#)—Store and sync data with NoSQL cloud database across clients in realtime

# Files

# Android File System

- External storage – Public directories
- Internal storage – Private directories for just your app

Apps can browse the directory structure

Structure and operations similar to Linux and java.io



# Internal storage

- Always available
- Uses device's filesystem
- Only your app can access files, unless explicitly set to be readable or writable
- On app uninstall, system removes all app's files from internal storage

# External storage

- Not always available, can be removed
- Uses device's file system or physically external storage like SD card
- World-readable, so any app can read
- On uninstall, system does not remove files private to app

# When to use internal/external storage

## Internal is best when

- you want to be sure that neither the user nor other apps can access your files

## External is best for files that

- don't require access restrictions and for
- you want to share with other apps
- you allow the user to access with a computer

# Save user's file in shared storage

- Save new files that the user acquires through your app to a public directory where other apps can access them and the user can easily copy them from the device
- Save external files in public directories

# Internal Storage

# Internal Storage

- Uses private directories just for your app
- App always has permission to read/write
- Permanent storage directory—[getFilesDir\(\)](#)
- Temporary storage directory—[getCacheDir\(\)](#)

# Creating a file

```
File file = new File(  
    context.getFilesDir(), filename);
```

Use standard [java.io](#) file operators or streams  
to interact with files

# External Storage

# External Storage

- On device or SD card
- Set permissions in Android Manifest
  - Write permission includes read permission

```
<uses-permission
```

```
    android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

```
<uses-permission
```

```
    android:name="android.permission.READ_EXTERNAL_STORAGE" />
```



# Always check availability of storage

```
public boolean isExternalStorageWritable() {  
    String state = Environment.getExternalStorageState();  
    if (Environment.MEDIA_MOUNTED.equals(state)) {  
        return true;  
    }  
    return false;  
}
```

# Example external public directories

- DIRECTORY\_ALARMS and DIRECTORY\_RINGTONES  
For audio files to use as alarms and ringtones
- DIRECTORY\_DOCUMENTS  
For documents that have been created by the user
- DIRECTORY\_DOWNLOADS  
For files that have been downloaded by the user

[developer.android.com/reference/android/os/Environment.html](https://developer.android.com/reference/android/os/Environment.html)

# Accessing public external directories

1. Get a path [getExternalStoragePublicDirectory\(\)](#)
2. Create file

```
File path = Environment.getExternalStoragePublicDirectory(  
        Environment.DIRECTORY_PICTURES);  
  
File file = new File(path, "DemoPicture.jpg");
```

# How much storage left?

- If there is not enough space, throws [IOException](#)
- If you know the size of the file, check against space
  - [getFreeSpace\(\)](#)
  - [getTotalSpace\(\)](#).
- If you do not know how much space is needed
  - try/catch [IOException](#)

# Delete files no longer needed

- External storage

```
myFile.delete();
```

- Internal storage

```
myContext.deleteFile(fileName);
```



# Do not delete the user's files!

When the user uninstalls your app, your app's private storage directory and all its contents are deleted

***Do not use private storage for content that belongs to the user!***

For example

- Photos captured or edited with your app
- Music the user has purchased with your app

# Shared Preferences & SQLite Database

# SQLite Database

- Ideal for repeating or structured data, such as contacts
- Android provides SQL-like database
- Covered in following chapters and practicals
  - SQLite Primer
  - Introduction to SQLite Databases
  - SQLite Data Storage Practical
  - Searching an SQLite Database Practical



# Shared Preferences

- Read and write small amounts of primitive data as key/value pairs to a file on the device storage
- Covered in later chapter and practical
  - Shared Preferences

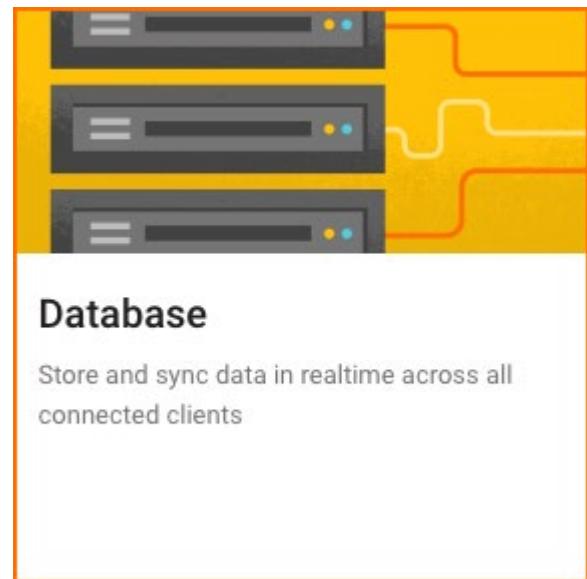
# Other Storage Options

# Use Firebase to store and share data

Store and sync data with the Firebase cloud database

Data is synced across all clients, and remains available when your app goes offline

[firebase.google.com/docs/database/](https://firebase.google.com/docs/database/)



# Firebase Realtime Database

- Connected apps share data
- Hosted in the cloud
- Data is stored as JSON
- Data is synchronized in realtime to every connected client



# Network Connection

- You can use the network (when it's available) to store and retrieve data on your own web-based services
- Use classes in the following packages
  - [java.net.\\*](#)
  - [android.net.\\*](#)

# Backing up data

- Auto Backup for 6.0 (API level 23) and higher
- Automatically back up app data to the cloud
- No code required and free
- Customize and configure auto backup for your app
- See [Configuring Auto Backup for Apps](#)



# Backup API for Android 5.1 (API level 22)

1. Register for Android Backup Service to get a Backup Service Key
2. Configure Manifest to use Backup Service
3. Create a backup agent by extending the `BackupAgentHelper` class
4. Request backups when data has changed

More info and sample code: [Using the Backup AP](#) and [Data Backup](#)

# Learn more about files

- [Saving Files](#)
- [getExternalFilesDir\(\) documentation and code samples](#)
- [getExternalStoragePublicDirectory\(\) documentation and code samples](#)
- [java.io.File class](#)
- [Oracle's Java I/O Tutorial](#)

# Learn more about backups

- [Configuring Auto Backup for Apps](#)
- [Using the Backup API](#)
- [Data Backup](#)

# What's next?

- Concept Chapter: [9.0 Data Storage](#)
- No practical, this was an overview lecture



# END

# Preferences and settings

Lesson 9



# 9.1 Shared Preferences

# Contents

- Shared Preferences
- Listening to changes

# What is Shared Preferences?

- Read and write small amounts of primitive data as key/value pairs to a file on the device storage
- SharedPreferences class provides APIs for reading, writing, and managing this data
- Save data in onPause()  
restore in onCreate()

# Shared Preferences AND Saved Instance State

- Small number of key/value pairs
- Data is private to the application

# Shared Preferences vs. Saved Instance State

- Persist data across user sessions, even if app is killed and restarted, or device is rebooted
- Data that should be remembered across sessions, such as a user's preferred settings or their game score
- Common use is to store user preferences
- Preserves state data across activity instances in same user session
- Data that should not be remembered across sessions, such as the currently selected tab or current state of activity.
- Common use is to recreate state after the device has been rotated

# Creating Shared Preferences

- Need only one Shared Preferences file per app
- Name it with package name of your app—unique and easy to associate with app
- MODE argument for `getSharedPreferences()` is for backwards compatibility—use only `MODE_PRIVATE` to be secure

# getSharedPreferences()

```
private String sharedPrefFile =  
    "com.example.android.hellosharedprefs";  
  
mPreferences =  
    getSharedPreferences(sharedPrefFile,  
                        MODE_PRIVATE);
```

# Saving Shared Preferences

- [SharedPreferences.Editor](#) interface
- Takes care of all file operations
- put methods overwrite if key exists
- apply() saves asynchronously and safely

# SharedPreferences.Editor

```
@Override  
protected void onPause() {  
    super.onPause();  
    SharedPreferences.Editor preferencesEditor =  
        mPreferences.edit();  
    preferencesEditor.putInt("count", mCount);  
    preferencesEditor.putInt("color", mCurrentColor);  
    preferencesEditor.apply();  
}
```



# Restoring Shared Preferences

- Restore in `onCreate()` in Activity
- Get methods take two arguments—the key, and the default value if the key cannot be found
- Use default argument so you do not have to test whether the preference exists in the file

# Getting data in onCreate()

```
mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);  
if (savedInstanceState != null) {  
    mCount = mPreferences.getInt("count", 1);  
    mShowCount.setText(String.format("%s", mCount));  
  
    mCurrentColor = mPreferences.getInt("color", mCurrentColor);  
    mShowCount.setBackgroundColor(mCurrentColor);  
  
    mNewText = mPreferences.getString("text", "");  
} else { ... }
```

# Clearing

- Call clear() on the SharedPreferences.Editor and apply changes
- You can combine calls to put and clear. However, when you apply(), clear() is always done first, regardless of order!

# clear()

```
SharedPreferences.Editor preferencesEditor =  
    mPreferences.edit();  
  
preferencesEditor.clear();  
  
preferencesEditor.apply();
```

# Listening to Changes



# Listening to changes

- Implement interface  
[SharedPreference.OnSharedPreferenceChangeListener](#)
- Register listener with  
[registerOnSharedPreferenceChangeListener\(\)](#)
- Register and unregister listener in [onResume\(\)](#) and  
[onPause\(\)](#)
- Implement on [onSharedPreferenceChanged\(\)](#) callback

# Interface and callback

```
public class SettingsActivity extends AppCompatActivity  
    implements OnSharedPreferenceChangeListener { ...  
  
    public void onSharedPreferenceChanged(  
        SharedPreferences sharedPreferences, String key) {  
        if (key.equals(MY_KEY)) {  
            // Do something  
        }  
    }  
}
```

# Creating and registering listener

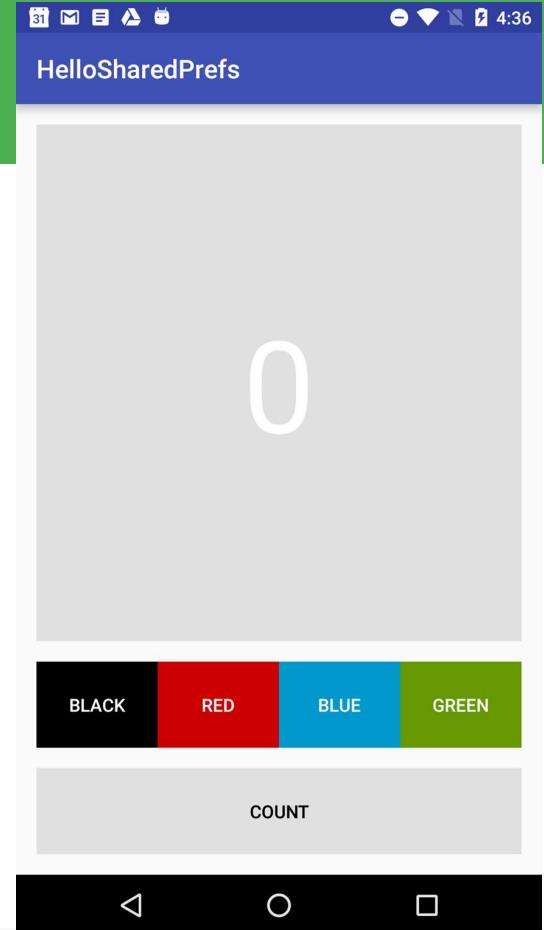
```
SharedPreferences.OnSharedPreferenceChangeListener listener =  
    new SharedPreferences.OnSharedPreferenceChangeListener() {  
        public void onSharedPreferenceChanged(  
            SharedPreferences prefs, String key) {  
            // Implement listener here  
        }  
    };  
prefs.registerOnSharedPreferenceChangeListener(listener);
```

# You need a **STRONG** reference to the listener

- When registering the listener the preference manager does not store a strong reference to the listener
- You must store a strong reference to the listener, or it will be susceptible to garbage collection
- Keep a reference to the listener in the instance data of an object that will exist as long as you need the listener

# Practical: HelloSharedPrefs

- Add Shared Preferences to a starter app
- Add a "Reset" button that clears both the app state and the preferences for the app



# Learn more

- [Saving Data](#)
- [Storage Options](#)
- [Saving Key-Value Sets](#)
- [SharedPreferences](#)
- [SharedPreferences.Editor](#)

## Stackoverflow

- [How to use SharedPreferences in Android to store, fetch and edit values](#)
- [onSavedInstanceState vs. SharedPreferences](#)



# What's Next?

- Concept Chapter: [9.1 Shared Preferences](#)
- Practical: [9.1 Shared Preferences](#)

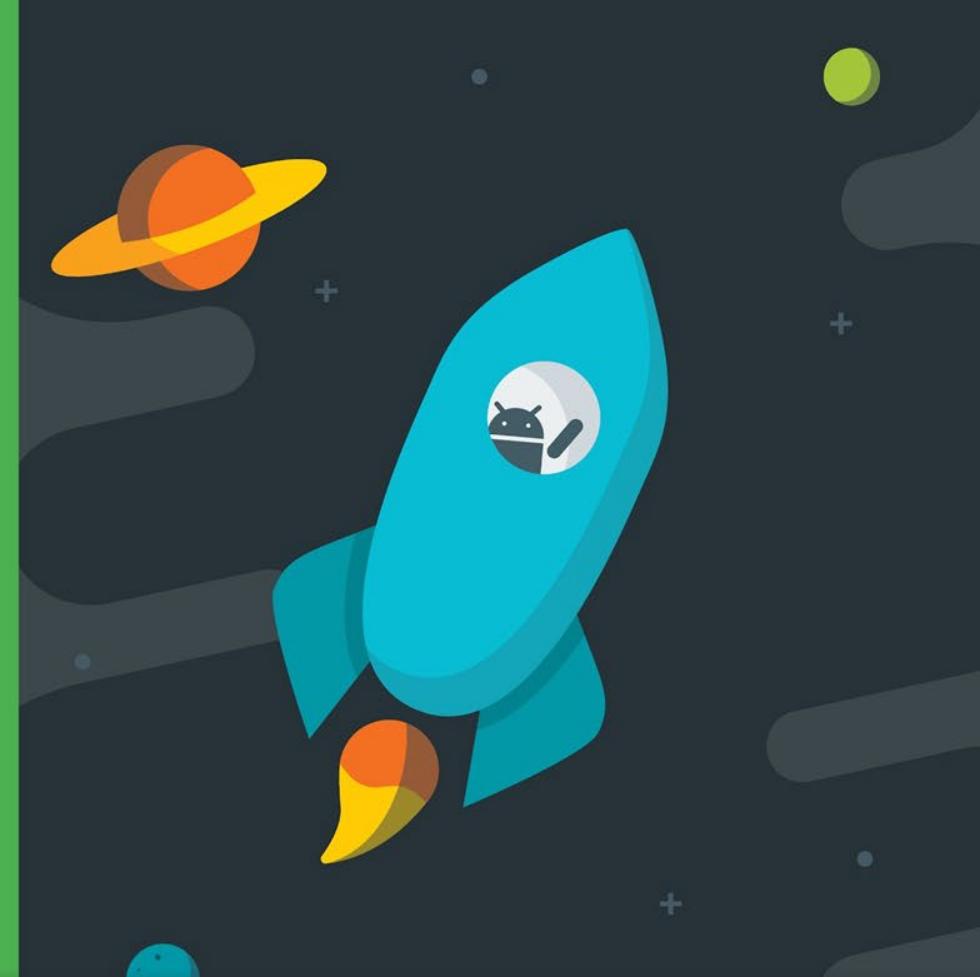


# END



# Preferences and settings

Lesson 9



This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# 9.2 App settings

# Contents

- What are settings?
- Setting screens
- Implement settings
- Default settings
- Save and retrieve settings
- Respond to changes in settings
- Summaries for settings
- Settings Activity template



# Settings



# What are app settings?

- Users can set features and behaviors of app  
Examples:
  - Home location, defaults units of measurement
  - Notification behavior for specific app
- For values that change infrequently and are relevant to most users
- If values change often, use options menu or nav drawer

# Example settings

**Favorite destination**

San Francisco

**CANCEL      OK**

**Sleep through meals?**

You will not be woken for meals



**Preferred snack**

- chocolate
- ice cream
- fruit
- nuts

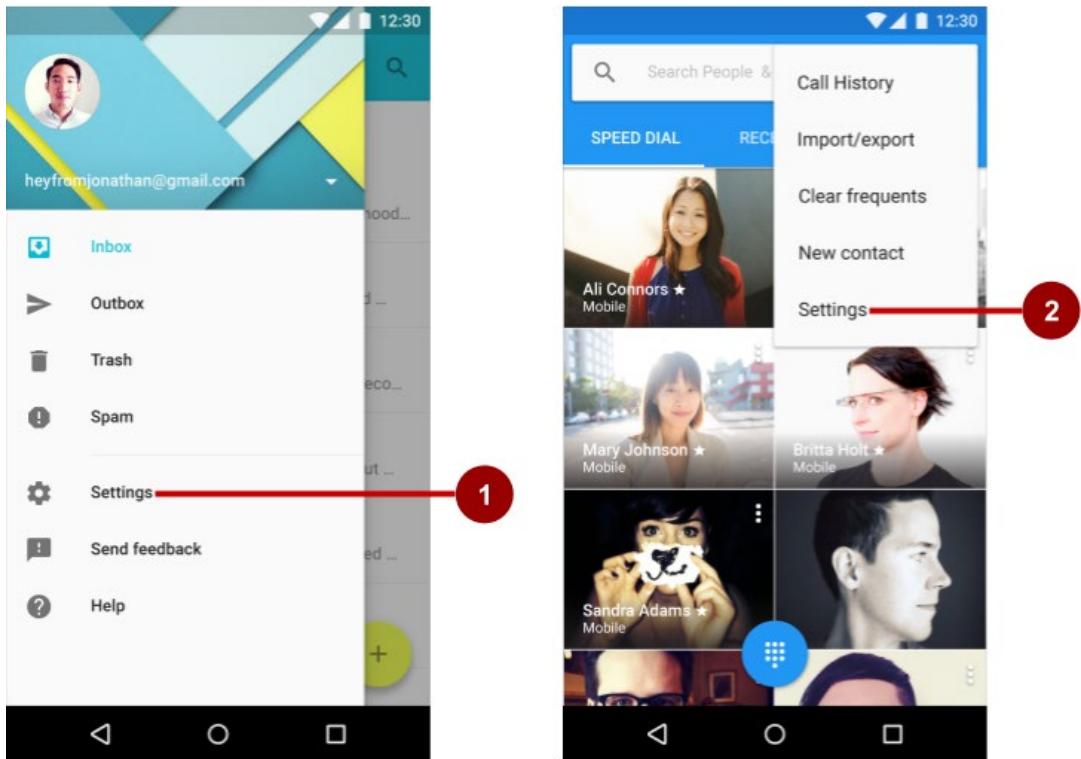
**CANCEL**



# Accessing settings

Users access settings through:

1. Navigation drawer
2. Options menu

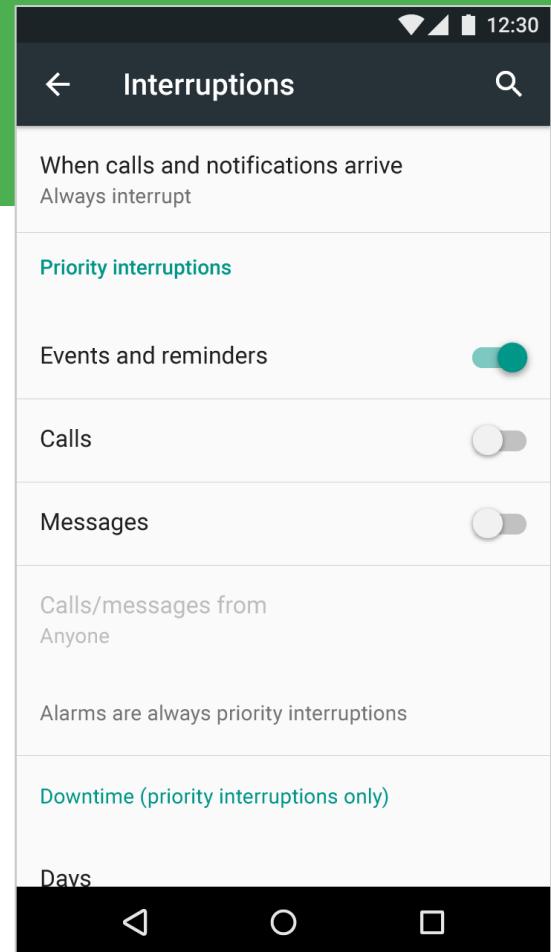


# Setting screens



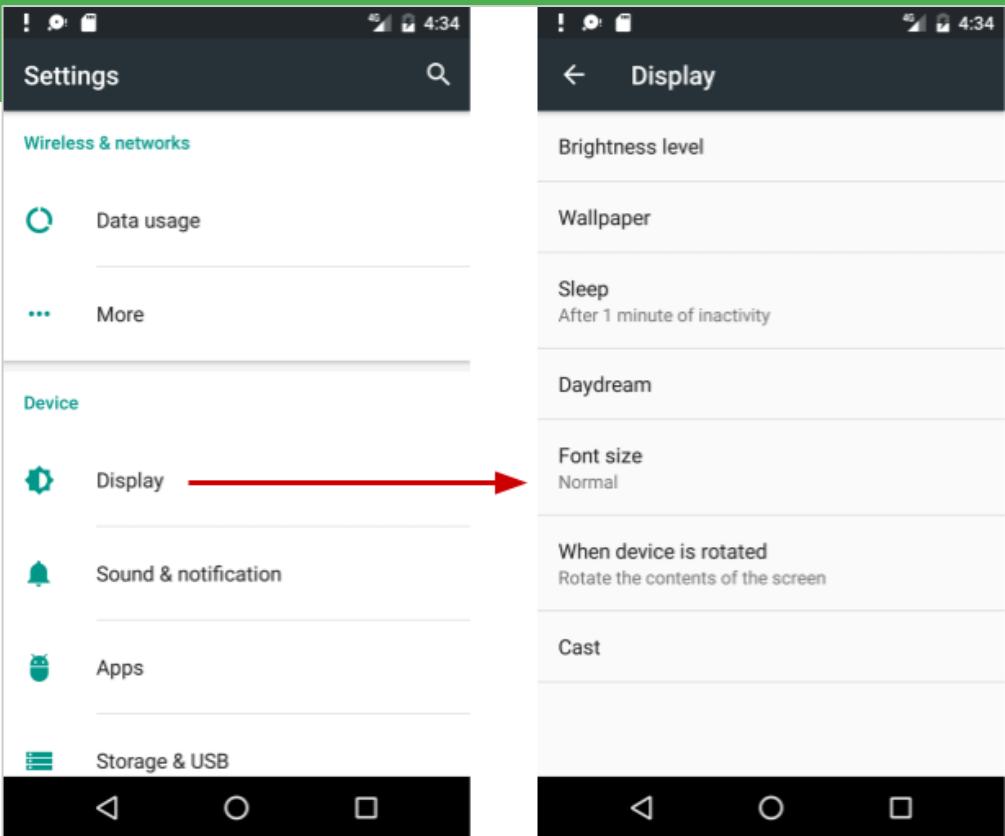
# Organize your settings

- Predictable, manageable number of options
- 7 or less: arrange according to priority with most important at top
- 7-15 settings: group related settings under section dividers



# 16+ Settings

- Group into screens opened from main Settings screen

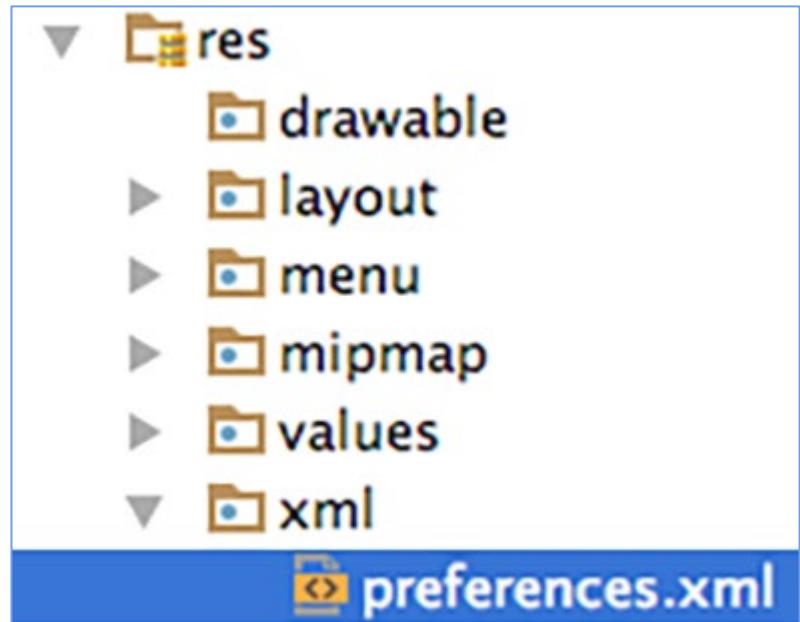


# View versus Preference

- Use Preference objects instead of View objects in your Settings screens
- Design and edit Preference objects in the layout editor just like you do for View objects

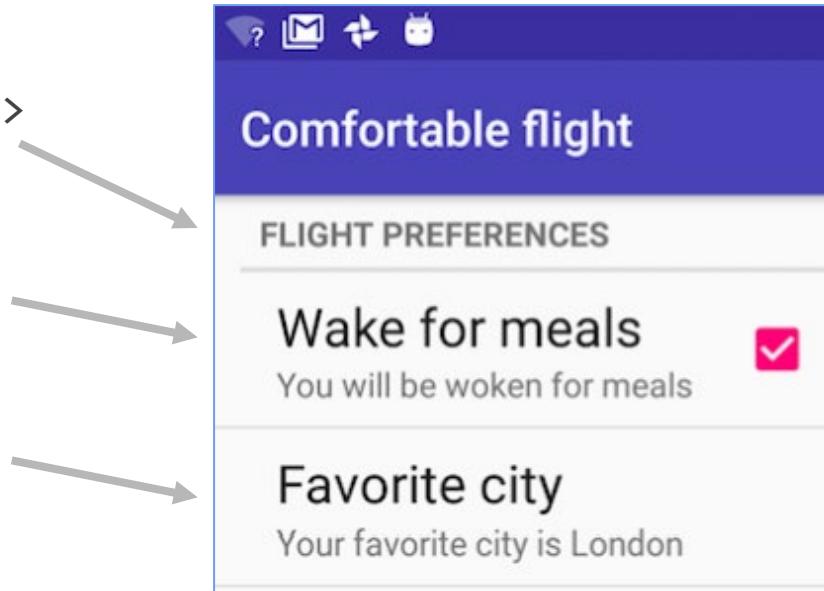
# Define Settings in a Preference Screen

- Define settings in a preferences screen
- It is like a layout
- define in:  
res > xml > preferences.xml



# Preference Screen example

```
<PreferenceScreen>
    <PreferenceCategory
        android:title="Flight Preferences">
        <CheckBoxPreference
            android:title="Wake for meals"
            ... />
        <EditTextPreference
            android:title="Favorite city"
            .../>
    </PreferenceCategory>
</PreferenceScreen>
```



This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# Every Preference must have a key

- Every preference must have a key
- Android uses the key to save the setting value

```
<EditTextPreference
```

```
    android:title="Favorite city"
```

```
    android:key="fav_city"
```

```
... />
```

Favorite city

Your favorite city is London



# SwitchPreference

```
<PreferenceScreen  
    xmlns:android="http://schemas.android.com/apk/res/android">
```

```
    <SwitchPreference
```

```
        android:defaultValue="true"
```

```
        android:title="@string/pref_title_social"
```

```
        android:key="switch"
```

```
        android:summary="@string/pref_sum_social" />
```

```
</PreferenceScreen>
```

Enable social recommendations  
Recommendations for people to contact  
based on your order history



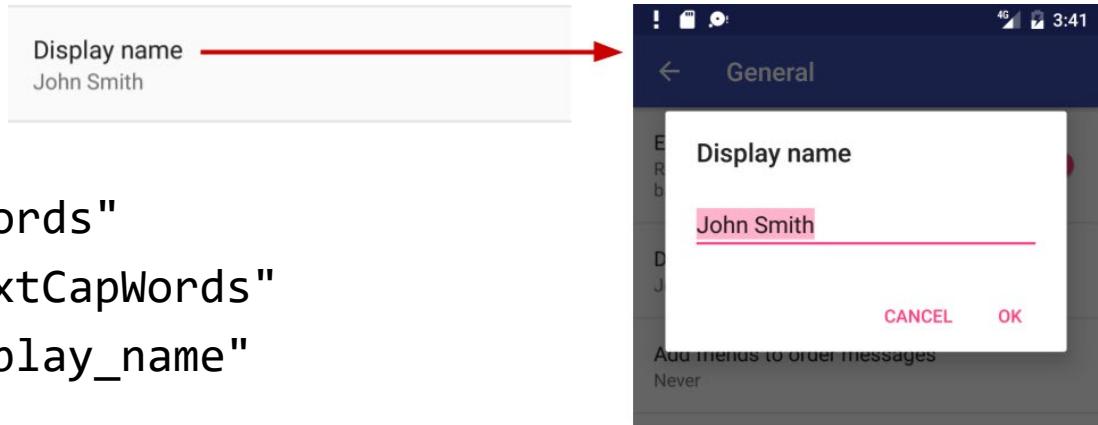
# SwitchPreference attributes

- android:defaultValue—true by default
- android:summary—text underneath setting, for some settings, should change to reflect value
- android:title—title/name
- android:key—key for storing value in SharedPreferences

# EditTextPreference

<EditTextPreference

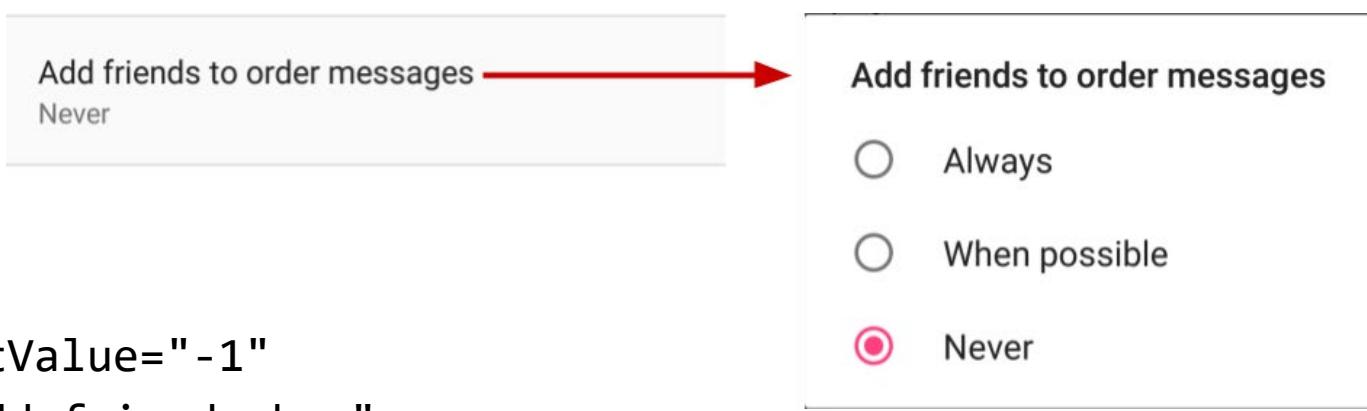
```
    android:capitalize="words"  
    android:inputType="textCapWords"  
    android:key="user_display_name"  
    android:maxLines="1"  
    android:defaultValue="@string/pref_default_display_name"  
    android:title="@string/pref_title_display_name" />
```



# ListPreference

<ListPreference

```
    android:defaultValue="-1"
    android:key="add_friends_key"
    android:entries="@array/pref_example_list_titles"
    android:entryValues="@array/pref_example_list_values"
    android:title="@string/pref_title_add_friends_to_messages" />
```



# ListPreference

- Default value of -1 for no choice
- android:entries—Array of labels for radio buttons
- android:entryValues —Array of values radio button

# Preference class

- [Preference](#) class provides View for each kind of setting
- associates View with [SharedPreferences](#) interface to store/retrieve the preference data
- Uses key in the Preference to store the setting value

# Preference subclasses

- [CheckBoxPreference](#)—list item that shows a checkbox
- [ListPreference](#)—opens a dialog with a list of radio buttons
- [SwitchPreference](#)—two-state toggleable option
- [EditTextPreference](#)—that opens a dialog with an [EditText](#)
- [RingtonePreference](#)—lets user to choose a ringtone

# Classes for grouping

- PreferenceScreen
  - root of a Preference layout hierarchy
  - at the top of each screen of settings
- PreferenceGroup
  - for a group of settings (Preference objects).
- PreferenceCategory
  - title above a group as a section divider



# Implement settings



# Settings UI uses fragments

- Use an Activity with a Fragment to display the Settings screen
- Use specialized Activity and Fragment subclasses that handle the work of saving settings

# Activities and fragments for settings

- Android 3.0 and newer:
  - [AppCompatActivity](#) with [PreferenceFragmentCompat](#)
  - OR use [Activity](#) with [PreferenceFragment](#)
- Android older than 3.0 (API level 10 and lower):
  - build a special settings activity as an extension of the [PreferenceActivity](#) class (use the template!)

Lesson  
focusses  
on this!

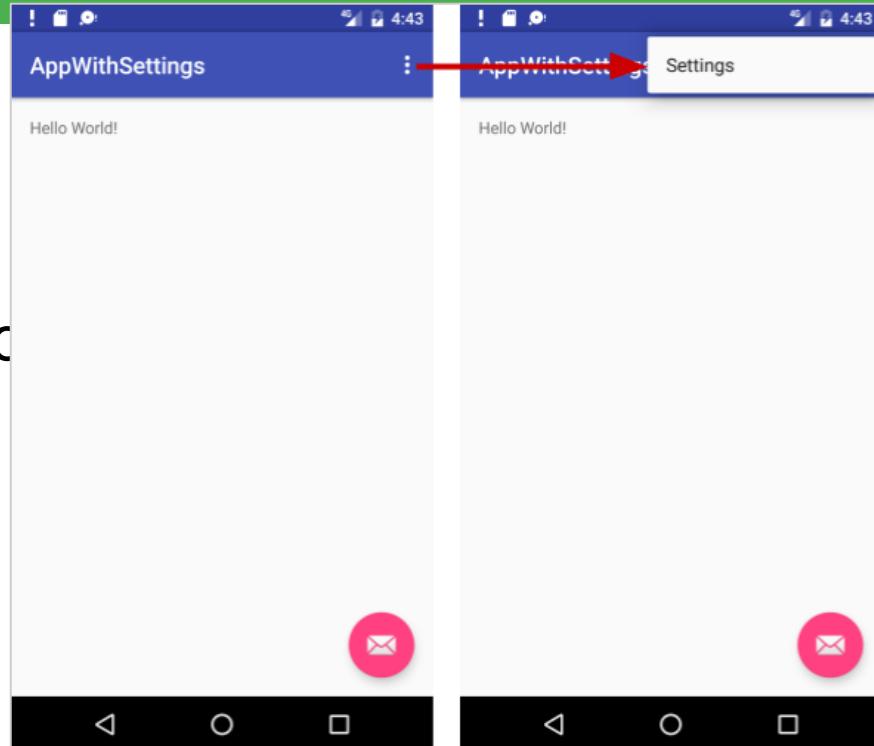
# Steps to implement Settings

For [AppCompatActivity](#) with [PreferenceFragmentCompat](#):

- Create the preferences screen
- Create an Activity for the settings
- Create a Fragment for the settings
- Add the preferenceTheme to the AppTheme
- Add code to invoke Settings UI

# Basic Activity template

- Basic Activity template  
Includes options menu
- Settings menu item provided for  
options menu



# Create a Settings Activity subclass

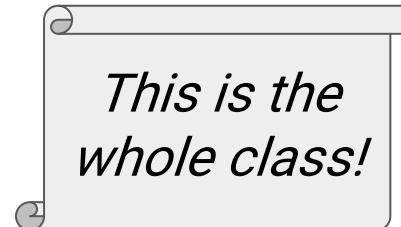
- Extends AppCompatActivity
- in onCreate() display the settings Fragment:

```
getSupportFragmentManager()  
    .beginTransaction()  
    .replace(android.R.id.content,  
            new MySettingsFragment())  
    .commit();
```



# Settings Activity example

```
public class MySettingsActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        getSupportFragmentManager().beginTransaction()  
            .replace(android.R.id.content, new MySettingsFragment())  
            .commit();  
    }  
}
```



This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# Create a Settings Fragment subclass

- Extends PreferenceFragmentCompat
- Implement methods:
  - onCreatePreferences() displays the settings
  - setOnPreferenceChangeListener() handles any changes that need to happen when the user changes a preference (optional)

# PreferenceFragment

```
public class MySettingsFragment  
    extends PreferenceFragmentCompat { ... }
```

- Blank fragments include onCreateView() by default
- Replace onCreateView() with onCreatePreferences()  
because this fragment displays a preferences screen

# Settings Fragment example

```
public class MySettingsFragment extends PreferenceFragmentCompat {  
    @Override  
    public void onCreatePreferences(Bundle savedInstanceState,  
                                  String rootKey) {  
        setPreferencesFromResource(R.xml.preferences, rootKey);  
    }  
}
```

# Add PreferenceTheme to app's theme

If using PreferenceFragmentCompat, set preferenceTheme in styles.xml:

```
<style name="AppTheme" parent="...>
    ...
    <item name="preferenceTheme">
        @style/PreferenceThemeOverlay
    </item>
    ...
</style>
```

# Invoke Settings UI

Send the Intent to start the Settings Activity:

- From Options menu, update `onOptionItemsSelected()`
- From Navigation drawer, update `onItemClick()` on the `OnItemClickListener` given to `setOnItemClickListener`

# Default Settings



# Default settings

- Set default to value most users would choose
  - All contacts
- Use less battery power
  - Bluetooth is off until the user turns it on
- Least risk to security and data loss
  - Archive rather than delete messages
- Interrupt only when important
  - When calls and notifications arrive

# Set default values

- Use android:defaultValue in Preference view in xml:

```
<EditTextPreference  
    android:defaultValue="London"  
    ... />
```

- In onCreate() of MainActivity, save default values.

# Save default values in shared preferences

In onCreate() of MainActivity

```
PreferenceManager.setDefaultValues(  
        this, R.xml.preferences, false);
```

- App [context](#), such as this
- Resource ID of XML resource file with settings
- `false` only calls method the first time the app starts

# Save and retrieve settings



# Saving setting values

- No need to write code to save settings!
- If you use specialized Preference Activity and Fragment, Android automatically saves setting values in shared preferences

# Get settings from shared preferences

- In your code, get settings from default shared preferences
- Use key as specified in preference view in xml

```
SharedPreferences sharedPref =  
    PreferenceManager.getDefaultSharedPreferences(this);  
  
String destinationPref =  
    sharedPref.getString("fav_city", "Jamaica");
```

# Get settings values from shared preferences

- In preference definition in xml:

```
<EditTextPreference  
    android:defaultValue="London"  
    android:key="fav_city" />
```

- In code, get fav\_city setting:

```
String destinationPref =  
    sharedPreferences.getString("fav_city", "Jamaica");
```

default setting value

*is different than*

default value returned by  
pref.getString() if key is  
not found in shared prefs

# Respond to changes in settings



# Listening to changes

- Display related follow-up settings
- Disable or enable related settings
- Change the summary to reflect current choice
- Act on the setting

For example, if the setting changes the screen background, then change the background



# Listen for changes to settings

- Define `setOnPreferenceChangeListener()`
- in `onCreatePreferences()` in the Settings Fragment

# onCreatePreferences() example

```
@Override  
public void onCreatePreferences(Bundle savedInstanceState,  
                             String rootKey) {  
  
    setPreferencesFromResource(R.xml.preferences, rootKey);  
    ListPreference colorPref =  
        (ListPreference) findPreference("color_pref");  
    colorPref.setOnPreferenceChangeListener(  
        // see next slide  
        // ...);  
}
```

# onPreferenceChangeListener() example

Example: change background color when setting changes

```
colorPref.setOnPreferenceChangeListener(  
    new Preference.OnPreferenceChangeListener(){  
        @Override  
        public boolean onPreferenceChange(  
            Preference preference, Object newValue){  
            setMyBackgroundColor(newValue);  
            return true;  
        }  
    });
```

# Summaries for settings



# Summaries for true/false values

Set attributes to define conditional summaries for preferences that have true/false values

Wake for meals

You will be woken for meals



## CheckBoxPreference

defaultValue

false

key

wake\_key

title

Wake for meals

summary

Do you want to be left alone at

dependency

icon

summaryOn

You will be woken for meals

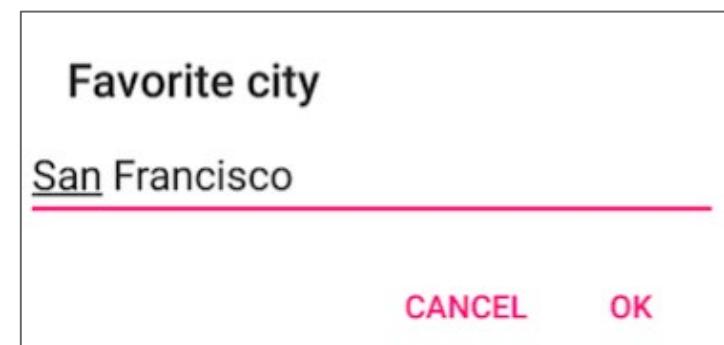
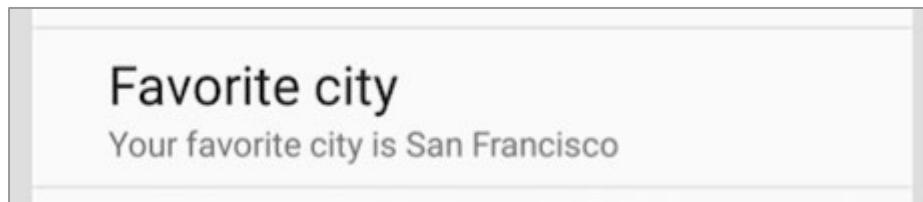
summaryOff

You will not be woken for meals

# Summaries for other settings

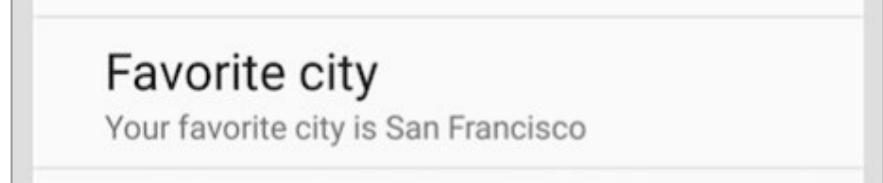
For settings that have values other than true/false, update the summary when the setting value changes

- Set the summary in `onPreferenceChangeListener()`



# Set summary example

```
EditTextPreference cityPref = (EditTextPreference)
                           findPreference("fav_city");
cityPref.setOnPreferenceChangeListener(
    new Preference.OnPreferenceChangeListener(){
        @Override
        public boolean onPreferenceChange(Preference pref, Object value){
            String city = value.toString();
            pref.setSummary("Your favorite city is " + city);
            return true;
        }
    });
}
```



The screenshot shows a preference screen with a single item. The title of the item is "Favorite city". Below the title, the summary text is "Your favorite city is San Francisco".

# Activity Template

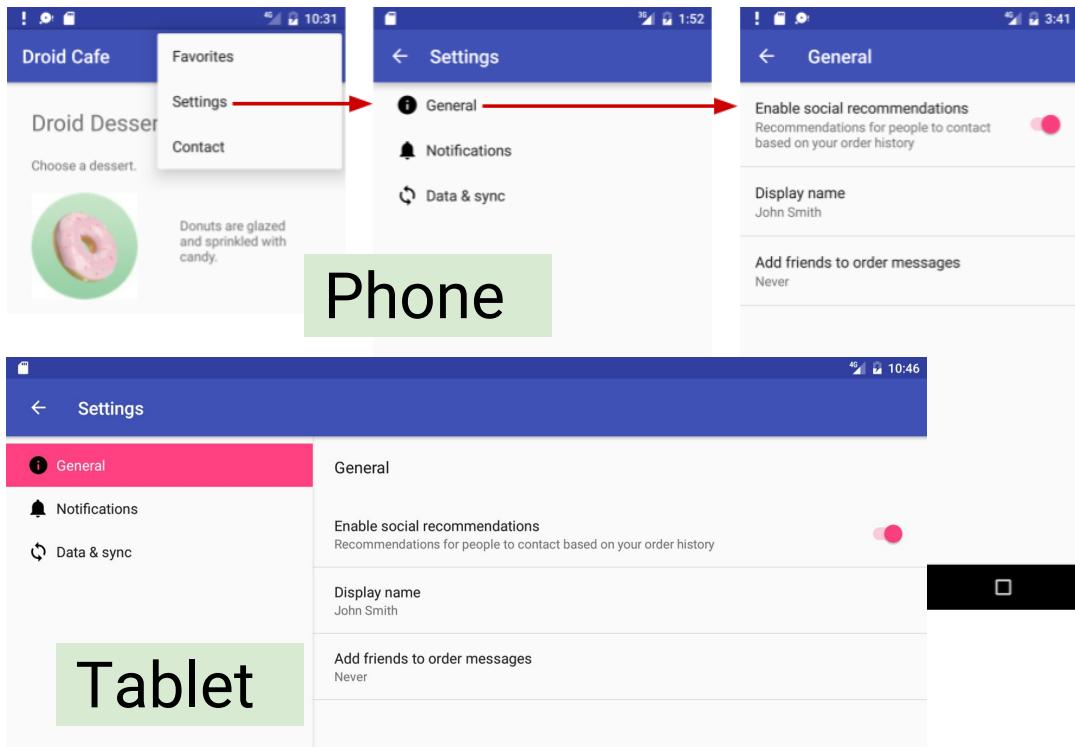


# More complex?

For anything more complex  
?  
use the Settings Activity template!

# Settings Activity template

- Complex Settings
- Backwards compatibility
- Customize pre-populated settings
- Adaptive layout for phones and tablets



This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# Learn more



# Learn more

- [Android Studio User Guide](#)
- [Settings \(coding\)](#)
- [Preference class](#)
- [PreferenceFragment](#)
- [Fragment](#)
- [SharedPreferences](#)
- [Saving Key-Value Sets](#)
- [Settings \(design\)](#)



# What's Next?

- Concept Chapter: [9.2 App settings](#)
- Practical: [9.2 App settings](#)



# END



# Storing Data with Room

Lesson 10



This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# 10.0 SQLite Primer



# Contents

- SQLite Database
- Queries



# This is only a refresher

This course assumes that you are familiar with

- Databases in general
- SQL databases in particular
- SQL query language

This chapter is a refresher and quick reference



# SQLite Database

# SQL Databases

- Store data in tables of rows and columns (spreadsheet...)
- Field = intersection of a row and column
- Fields contain data, references to other fields, or references to other tables
- Rows are identified by unique IDs
- Column names are unique per table



# Tables

## **WORD\_LIST\_TABLE**

<b>_id</b>	<b>word</b>	<b>definition</b>
1	"alpha"	"first letter"
2	"beta"	"second letter"
3	"alpha"	"particle"

# SQLite software library

Implements SQL database engine that is

- self-contained (requires no other components)
- serverless (requires no server backend)
- zero-configuration (does not need to be configured for your application)
- transactional (changes within a single transaction in SQLite either occur completely or not at all)

# What is a transaction?

A transaction is a sequence of operations performed as a single logical unit of work.

A logical unit of work must have four properties

- atomicity
- consistency
- isolation
- durability

*This work is licensed under a [Creative Commons Attribution 4.0 International License](#).*



# All or nothing

All changes within a single transaction in SQLite either occur completely or not at all, even if the act of writing the change out to the disk is interrupted by

- program crash
- operating system crash
- power failure.



# ACID

- **Atomicity**—All or no modifications are performed
- **Consistency**—When transaction has completed, all data is in a consistent state
- **Isolation**—Modifications made by concurrent transactions must be isolated from the modifications made by any other concurrent transactions
- **Durability**—After a transaction has completed, its effects are permanently in place in the system



# Queries

# SQL basic operations

- Insert rows
- Delete rows
- Update values in rows
- Retrieve rows that meet given criteria



# SQL Query

- ```
SELECT word, description
      FROM WORD_LIST_TABLE
     WHERE word="alpha"
```

## Generic

- ```
SELECT columns
      FROM table
     WHERE column="value"
```



# SELECT columns FROM table

- **SELECT columns**
  - Select the columns to return
  - Use \* to return all columns
- **FROM table**—specify the table from which to get results

# WHERE column="value"

- WHERE—keyword for conditions that have to be met
- column="value"—the condition that has to be met
  - common operators: =, LIKE, <, >

# AND, ORDER BY, LIMIT

```
SELECT _id FROM WORD_LIST_TABLE WHERE word="alpha"  
AND definition LIKE "%art%" ORDER BY word DESC LIMIT 1
```

- **AND, OR**—connect multiple conditions with logic operators
- **ORDER BY**—omit for default order, or ASC for ascending, DESC for descending
- **LIMIT**—get a limited number of results

# Sample queries

1	<pre>SELECT * FROM WORD_LIST_TABLE</pre>	Get the whole table
2	<pre>SELECT word, definition FROM WORD_LIST_TABLE WHERE _id &gt; 2</pre>	Returns [["alpha", "particle"]]

# More sample queries

3	<pre>SELECT _id FROM WORD_LIST_TABLE WHERE word="alpha" AND definition LIKE "%art%"</pre>	Return id of word alpha with substring "art" in definition [[ "3" ]]
4	<pre>SELECT * FROM WORD_LIST_TABLE ORDER BY word DESC LIMIT 1</pre>	Sort in reverse and get first item. Sorting is by the first column (_id) [[ "3", "alpha", "particle" ]]

# Last sample query

5

```
SELECT * FROM  
WORD_LIST_TABLE  
LIMIT 2,1
```

Returns 1 item starting at position 2.  
Position counting starts at 1 (not zero!).  
Returns  
[[ "2", "beta", "second letter" ]]



# rawQuery()

```
String query = "SELECT * FROM WORD_LIST_TABLE";  
rawQuery(query, null);
```

```
query = "SELECT word, definition FROM  
WORD_LIST_TABLE WHERE _id > ? ";
```

```
String[] selectionArgs = new String[]{"2"}  
rawQuery(query, selectionArgs);
```

# query()

```
SELECT * FROM  
WORD_LIST_TABLE  
WHERE word="alpha"  
ORDER BY word ASC  
LIMIT 2,1;
```

Returns:

```
[["alpha",  
"particle"]]
```

```
String table = "WORD_LIST_TABLE"  
String[] columns = new String[]{"*"};  
String selection = "word = ?"  
String[] selectionArgs = new String[]{"alpha"};  
String groupBy = null;  
String having = null;  
String orderBy = "word ASC"  
String limit = "2,1"  
  
query(table, columns, selection, selectionArgs,  
groupBy, having, orderBy, limit);
```

# Cursors

Queries always return a Cursor object

[Cursor](#) is an object interface that provides random read-write access to the result set returned by a database query

⇒ Think of it as a pointer to table rows

You will learn more about cursors in the following chapters



# Learn more

- [SQLite website](#)
- [Full description of the Query Language](#)
- [SQLite class](#)
- [Cursor class](#)



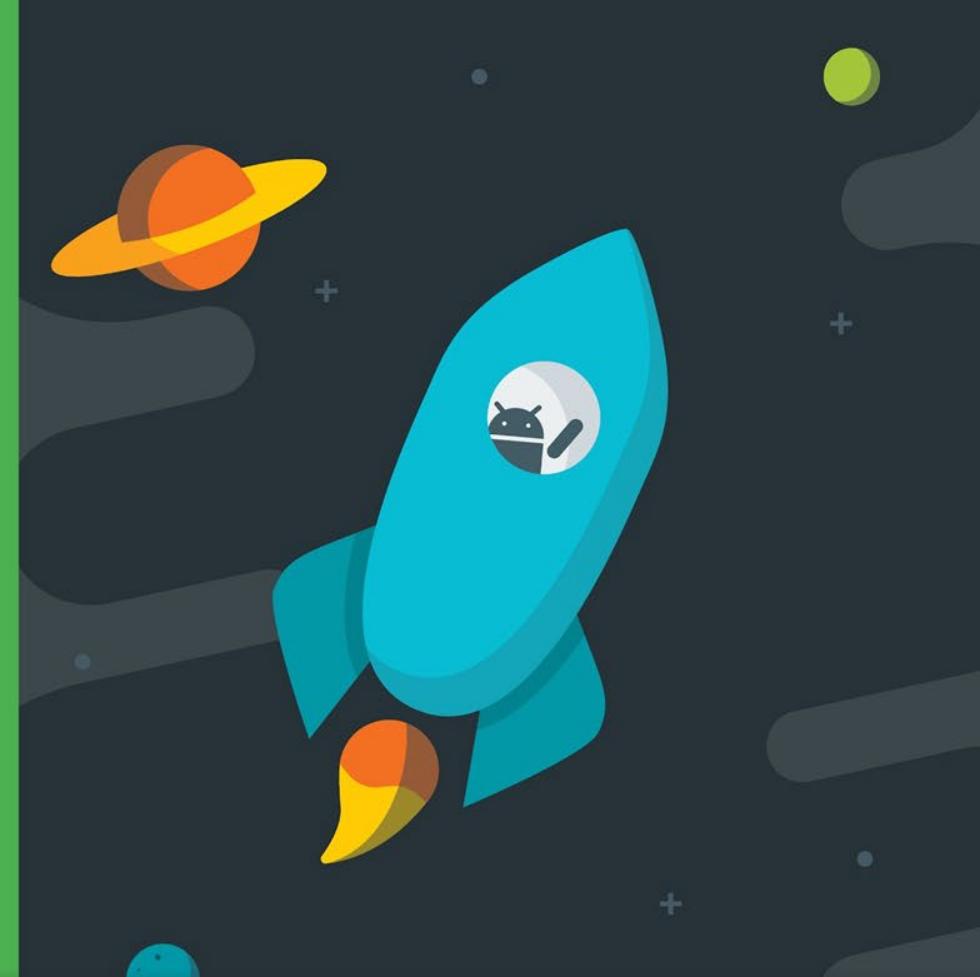
# What's Next?

- Concept Chapter: [10.0 SQLite Primer](#)
- No Practical

# END

# Storing data with Room

Lesson 10



# 10.1 Room, LiveData, and ViewModel



# Contents

- Architecture Components
- Entity
- DAO
- Room database
- ViewModel
- Repository
- LiveData
- Lifecycle



# Architecture Components

# Architecture Components

A set of Android libraries for structuring your app in a way that is robust, testable, and maintainable.



# Architecture Components

- Consist of best architecture practices + libraries
- Encourage recommended app architecture
- A LOT LESS boilerplate code
- Testable because of clear separation
- Fewer dependencies
- Easier to maintain

# Guide to app architecture

The screenshot shows a web browser window for the Android Developers site at <https://developer.android.com/topic/libraries/architecture/guide.html>. The page is titled 'Guide to App Architecture'. The left sidebar lists various libraries, with 'Architecture Components' selected. The main content area discusses common problems faced by app developers and provides links to issue trackers and community resources.

Guide to App Architecture

This guide is for developers who are past the basics of building an app, and now want to know the best practices and recommended architecture for building robust, production-quality apps.

**Note:** This guide assumes that the reader has familiarity with the Android Framework. If you are new to app development, check out the [Getting Started](#) training series, which covers prerequisite topics for this guide.

## Common problems faced by app developers

Unlike their traditional desktop counterparts which, in the majority of cases, have a single entry point from the launcher shortcut and run as a single monolithic process, Android apps have a much more complex structure. A typical Android app is constructed out of multiple [app components](#), including activities, fragments, services, content providers and broadcast receivers.

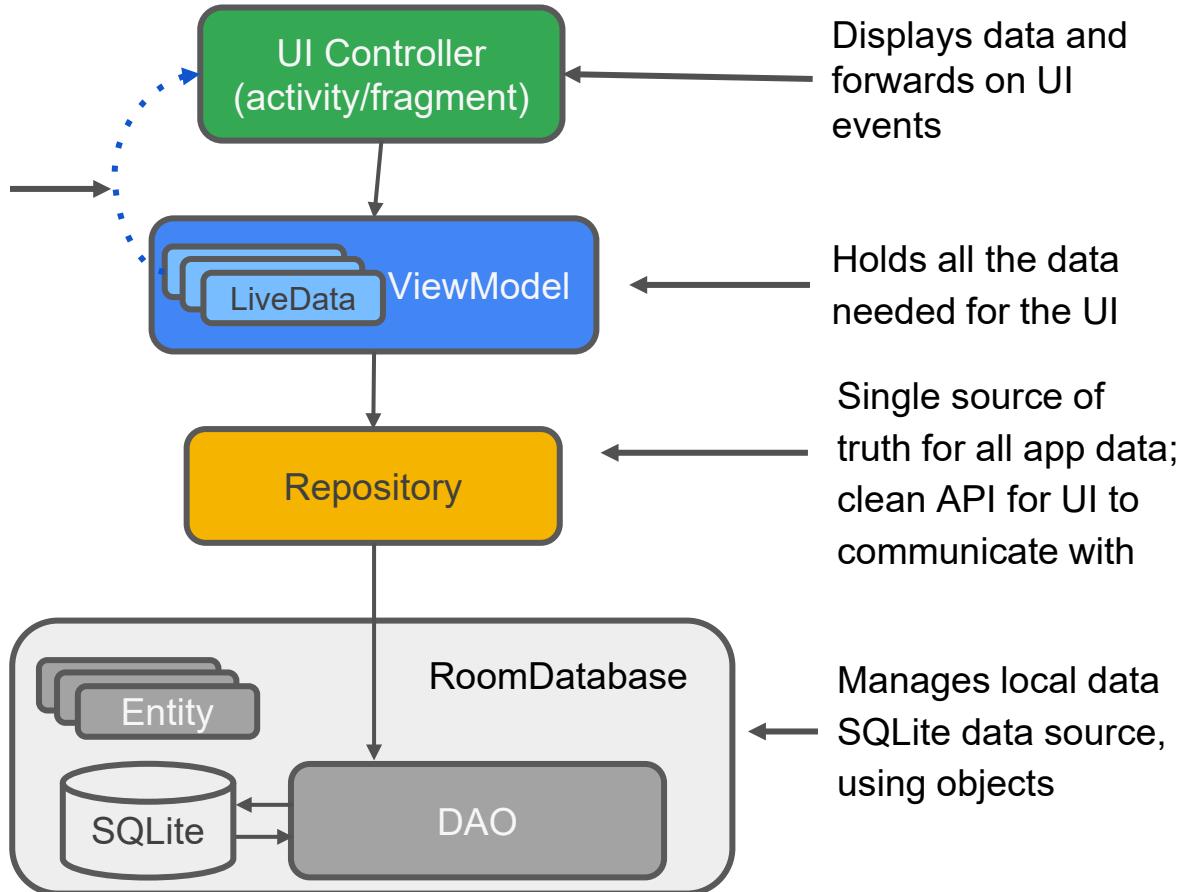
In this document

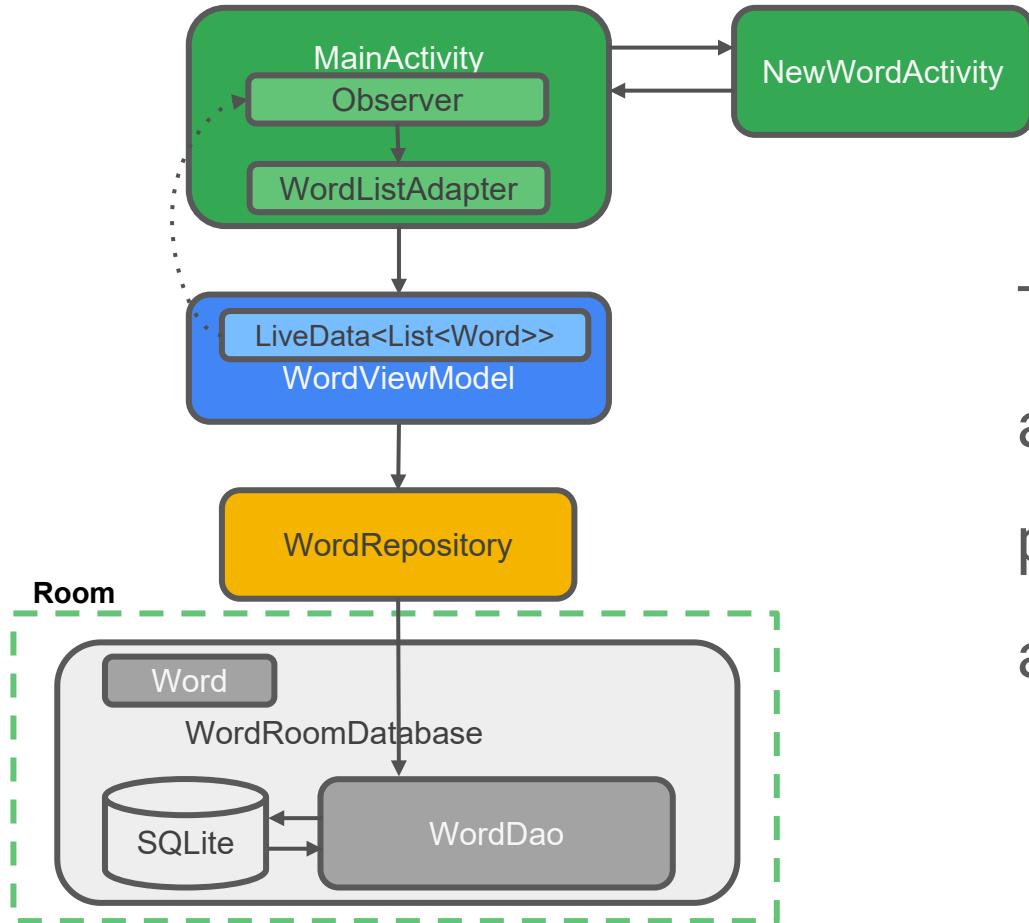
- › Common problems faced by app developers
- › Common architectural principles
- › Recommended app architecture
  - › Building the user interface
  - › Fetching data
  - › Connecting ViewModel and the repository
  - › Caching data

[developer.android.com/arc  
h](https://developer.android.com/topic/libraries/architecture/guide.html)

# Overview

UI is notified of changes using observation





The RoomWordsSample  
app that you build in the  
practical implements this  
architecture

# Room overview

# Room overview

Room is a robust SQL object mapping library

- Generates SQLite Android code
- Provides a simple API for your database

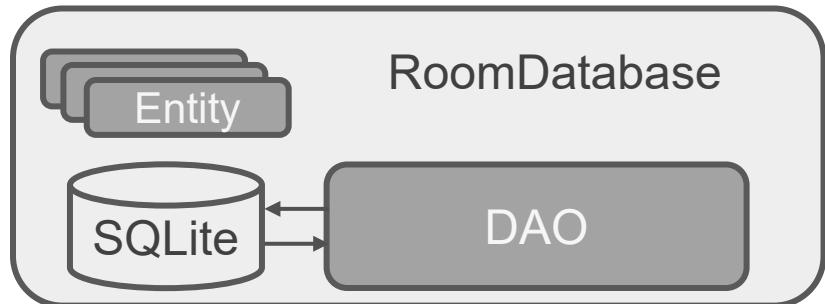


This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# Components of Room

- **Entity:** Defines schema of database table.
- **DAO:** Database Access Object  
Defines read/write operations for database.
- **Database:**  
A database holder.  
Used to create or  
connect to database

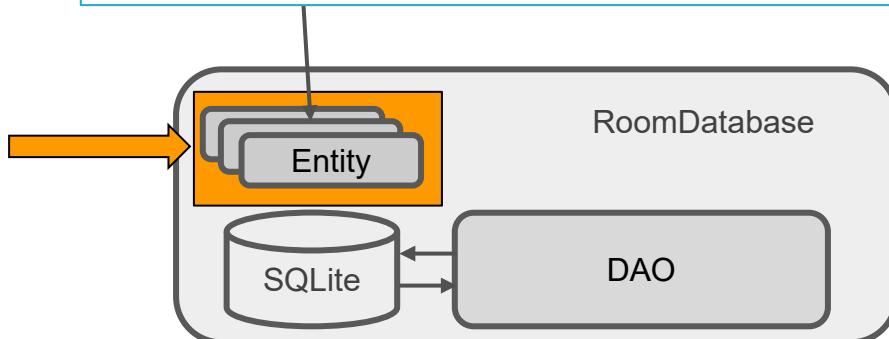


# Entity

# Entity

- Entity instance = row in a database table
- Define entities as POJO classes
- 1 instance = 1 row
- Member variable = column name

```
public class Person {  
    private int uid;  
    private String firstName;  
    private String lastName;  
}
```



# Entity instance = row in a database table

```
public class Person {  
    private int uid;  
    private String firstName;  
    private String lastName;  
}
```

uid	firstName	lastName
12345	Aleks	Becker
12346	Jhansi	Kumar

# Annotate entities

```
@Entity
public class Person {
    @PrimaryKey (autoGenerate=true)
    private int uid;

    @ColumnInfo(name = "first_name")
    private String firstName;

    @ColumnInfo(name = "last_name")
    private String lastName;

    // + getters and setters if variables are private.
}
```



# @Entity annotation

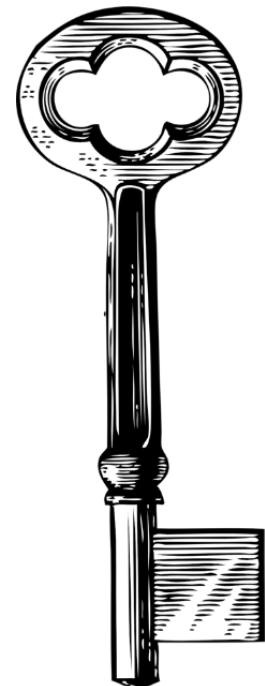
```
@Entity(tableName = "word_table")
```

- Each @Entity instance represents an entity/row in a table
- Specify the name of the table if different from class name

# @PrimaryKey annotation

`@PrimaryKey (autoGenerate=true)`

- **Entity** class must have a field annotated as primary key
- You can auto-generate unique key for each entity
- See Defining data using Room entities



# @NonNull annotation

## @NonNull

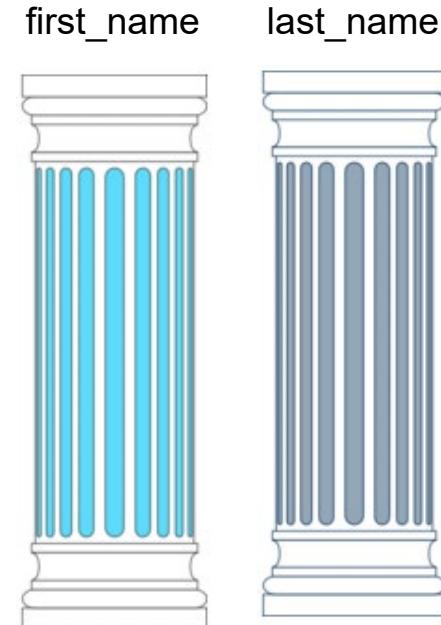
- Denotes that a parameter, field, or method return value can never be null
- Use for mandatory fields
- Primary key must use @NonNull



# @ColumnInfo annotation

```
@ColumnInfo(name = "first_name")  
private String firstName;  
  
@ColumnInfo(name = "last_name")  
private String lastName;
```

- Specify column name if different from member variable name



# Getters, setters

Every field that's stored in the database must

- be public

OR

- have a "getter" method

... so that Room can access it



# Relationships

Use @Relation annotation to define related entities

Queries fetch all the returned object's relations

users table

id	
name	
pet	



pets table

id	
name	
owner	

# Many more annotations

For more annotations, see  
[Room package summary](#)  
[reference](#)

^ android.arch.persistence.room

[Overview](#)

^ [Annotations](#)

[ColumnInfo](#)

[ColumnInfo.Collate](#)

[ColumnInfo.SQLiteTypeAffinity](#)

[Dao](#)

[Database](#)

[Delete](#)

[Embedded](#)

[Entity](#)

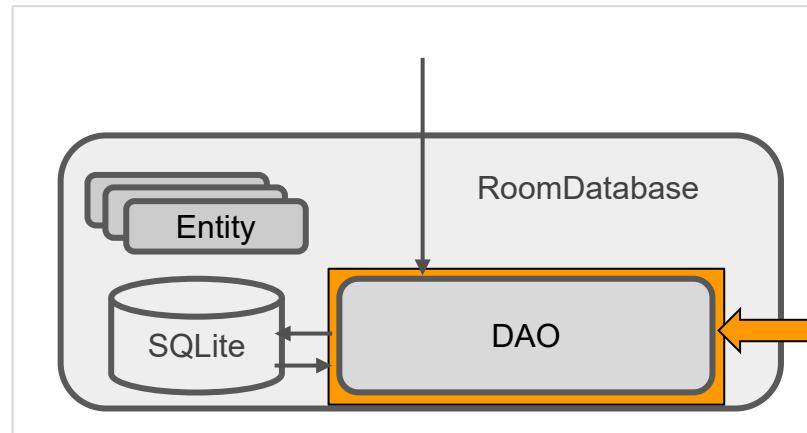
[ForeignKey](#)

[ForeignKey.Action](#)

# Data access object (DAO)

# Data access object

Use *data access objects*, or DAOs,  
to access app data using the  
[Room persistence library](#)



# Data access object

- DAO methods provide abstract access to the app's database
- The data source for these methods are entity objects
- DAO must be interface or abstract class
- Room uses DAO to create a clean API for your code

# Example DAO

```
@Dao
public interface WordDao {
    @Insert
    void insert(Word word);
    @Update
    public void updateWords(Word... words);
}
```

*//... More queries on next slide...*

# Example queries

```
@Query("DELETE FROM word_table")
void deleteAll();
```

```
@Query("SELECT * from word_table ORDER BY word ASC")
List<Word> getAllWords();
```

```
@Query("SELECT * FROM word_table WHERE word LIKE :word ")
public List<Word> findWord(String word);
```

# Room database

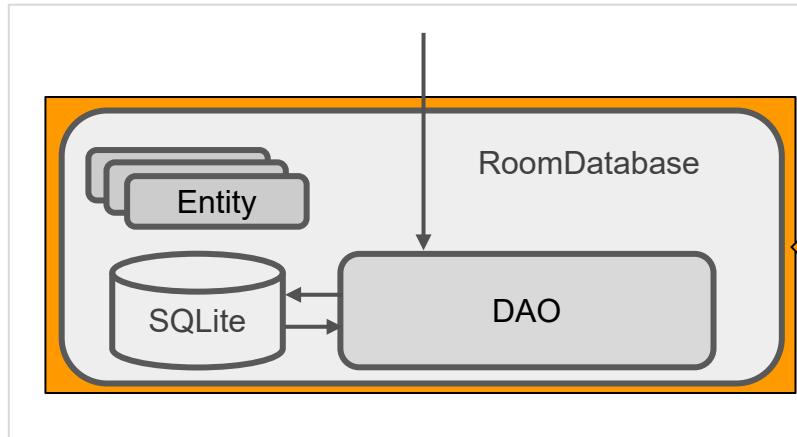
# Room

- Room is a robust SQL object mapping library
- Generates SQLite Android code



# Room

- Room works with DAO and Entities
- Entities define the database schema
- DAO provides methods to access database



# Creating Room database

- Create public abstract class extending RoomDatabase
- Annotate as @Database
- Declare entities for database schema  
and set version number

```
@Database(entities = {Word.class}, version = 1)
```

```
public abstract class WordRoomDatabase extends RoomDatabase
```

# Room class example

```
@Database(entities = {Word.class}, version = 1)
public abstract class WordRoomDatabase
    extends RoomDatabase {
    public abstract WordDao wordDao();
    private static WordRoomDatabase INSTANCE;
    // ... create instance here
}
```

*Entity  
defines  
DB schema*

*DAO for  
database*

*Create  
database  
as  
singleton  
instance*



# Use Database builder

- Use Room's database builder to create the database
- Create DB as singleton instance

```
private static WordRoomDatabase INSTANCE;  
INSTANCE = Room.databaseBuilder(...)  
    .build();
```

# Specify database class and name

- Specify Room database class and database name

```
INSTANCE = Room.databaseBuilder(  
    context,  
    WordRoomDatabase.class, "word_database")  
    // ...  
    .build();
```

# Specify onOpen callback

- Specify onOpen callback

```
INSTANCE = Room.databaseBuilder(  
    context,  
    WordRoomDatabase.class, "word_database")  
    .addCallback(sOnOpenCallback)  
    //...  
    .build();
```



# Specify migration strategy

- Specify migration strategy callback

```
INSTANCE = Room.databaseBuilder(  
    context.getApplicationContext(),  
    WordRoomDatabase.class, "word_database")  
    .addCallback(sOnOpenCallback)  
    .fallbackToDestructiveMigration()  
    .build();
```

# Room database creation example

```
static WordRoomDatabase getDatabase(final Context context) {
    if (INSTANCE == null) {
        synchronized (WordRoomDatabase.class) {
            if (INSTANCE == null) ← {-----}
                INSTANCE = Room.databaseBuilder(
                    context.getApplicationContext(),
                    WordRoomDatabase.class, "word_database")
                    .addCallback(sOnOpenCallback)
                    .fallbackToDestructiveMigration()
                    .build();
            }
    }
    return INSTANCE;
}
```

*Check if  
database exists  
before creating it*



# Initialize DB in onOpen callback

```
private static RoomDatabase.Callback sOnOpenCallback =  
    new RoomDatabase.Callback(){  
        @Override  
        public void onOpen (@NonNull SupportSQLiteDatabase db){  
            super.onOpen(db);  
            initializeData();  
        };  
    };
```

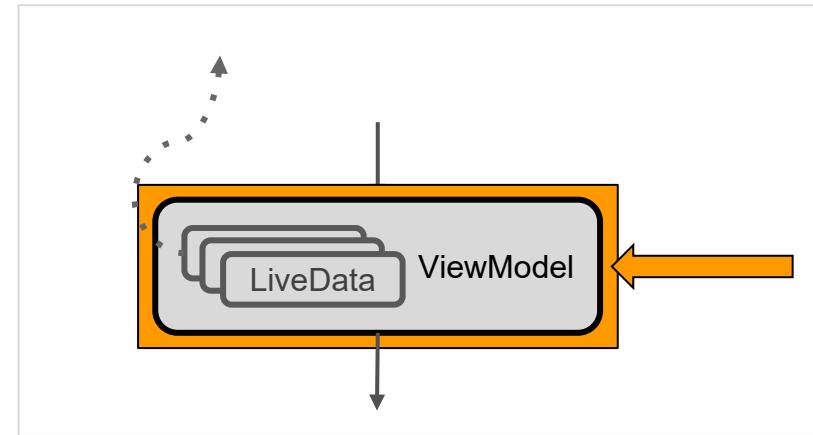
# Room caveats

- Compile-time checks of SQLite statements
- Do not run database operations on the main thread
- LiveData automatically runs query asynchronously on a background thread when needed
- Usually, make your RoomDatabase a singleton

# ViewModel

# ViewModel

- View models are objects that provide data for UI components and survive configuration changes.

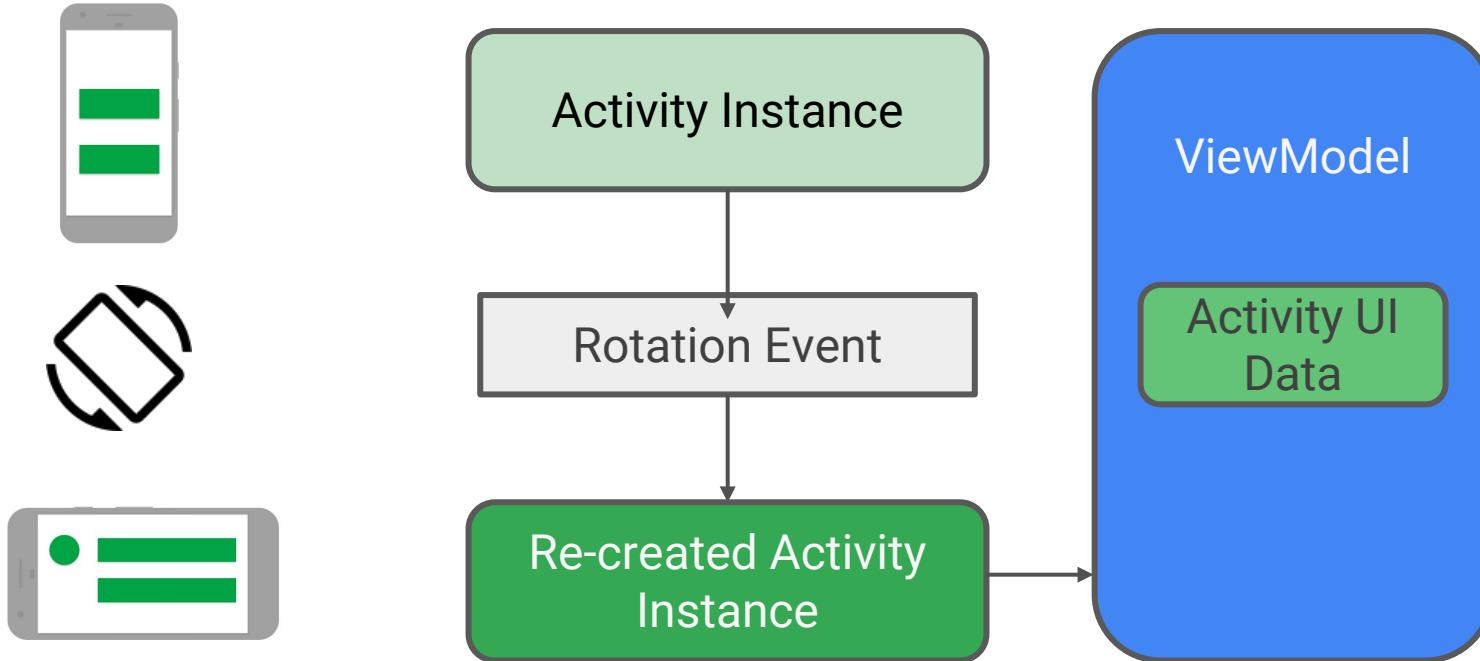


# ViewModel

- Provides data to the UI
- Survives configuration changes
- You can also use a [ViewModel](#) to share data between fragments
- Part of the [lifecycle library](#)



# Survives configuration changes



# ViewModel serves data

- ViewModel serves data to the UI
- Data can come from Room database or other sources
- ViewModel's role is to return the data, it can get help to find or generate the data



This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



# Best practice to use repository

Recommended best practice:

- Use a repository to do the work to get the data
- Keeps ViewModel as clean interface between app and data



*Repository is discussed in next section*

# Restaurant analogy

- Customer requests meal → ● UI requests data from ViewModel
- Server takes order to chefs → ● ViewModel asks Repository for data
- Chefs prepare meal → ● Repository gets data
- Server delivers meal to customer → ● ViewModel returns data to UI

# ViewModel example using repository

```
public class WordViewModel extends AndroidViewModel {  
    private WordRepository mRepository;  
    private LiveData<List<Word>> mAllWords;  
  
    // Initialize the repository and the list of words  
    public WordViewModel (Application application) {  
        super(application);  
        mRepository = new WordRepository(application);  
        mAllWords = mRepository.getAllWords();  
    }  
}
```

# ViewModel example continued

```
LiveData<List<Word>> getAllWords() {  
    return mAllWords;  
}  
  
public void insert(Word word) {  
    mRepository.insert(word);  
}  
  
public void deleteWord(Word word) {  
    mRepository.deleteWord(word);  
}
```

# Do not pass context into ViewModel

- Never pass context into ViewModel instances
- Do not store Activity, Fragment, or View instances or their Context in the ViewModel
- An Activity can be destroyed and created many times during the lifecycle of a ViewModel
- If you need application context, inherit from [AndroidViewModel](#)



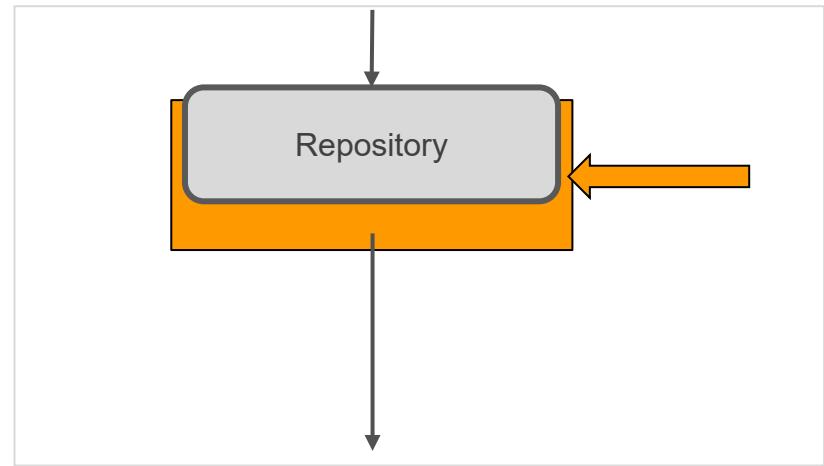
# ViewModel does not survive app closure

- ViewModel survives configuration changes,  
*not* app shutdown
- ViewModel is *not* a replacement for  
onSaveInstanceState()  
(if you are not saving the data with Room)
- See [Saving UI States](#)

# Repository

# Repository

- Best practice, not part of Architecture Components libraries
- Implement repository to provide single, clean API to app data

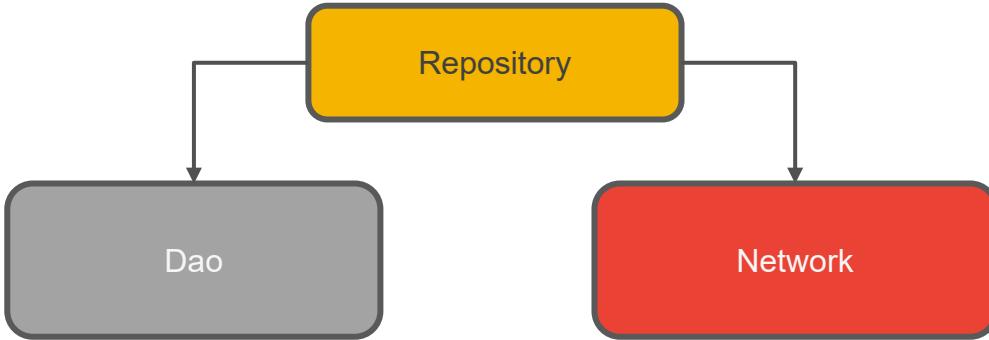


# Repository fetches or generates data

- Use repository to fetch data in the background
- Analogy: chefs prepare meals behind the scenes



# Multiple backends



- Potentially, repository could manage query threads and allow you to use multiple backends
- Example: in Repository, implement logic for deciding whether to fetch data from a network or use results cached in the database

# Repository example

```
public class WordRepository {  
  
    private WordDao mWordDao;  
    private LiveData<List<Word>> mAllWords;  
  
    WordRepository(Application application) {  
        WordRoomDatabase db = WordRoomDatabase.getDatabase(application);  
        mWordDao = db.wordDao();  
        mAllWords = mWordDao.getAllWords();  
    }  
    [... more code...]  
}
```

# Get and insert data

```
LiveData<List<Word>> getAllWords() {  
    return mAllWords;  
}  
  
// Must insert data off the main thread  
public void insert (Word word) {  
    new insertAsyncTask(mWordDao).execute(word);  
}  
  
[... more code...]  
}
```

# Insert off main thread

```
private static class insertAsyncTask extends AsyncTask<Word, Void, Void> {  
    private WordDao mAsyncTaskDao;  
  
    insertAsyncTask(WordDao dao) {  
        mAsyncTaskDao = dao;  
    }  
  
    @Override  
    protected Void doInBackground(final Word... params) {  
        mAsyncTaskDao.insert(params[0]);  
        return null;  
    }  
}
```

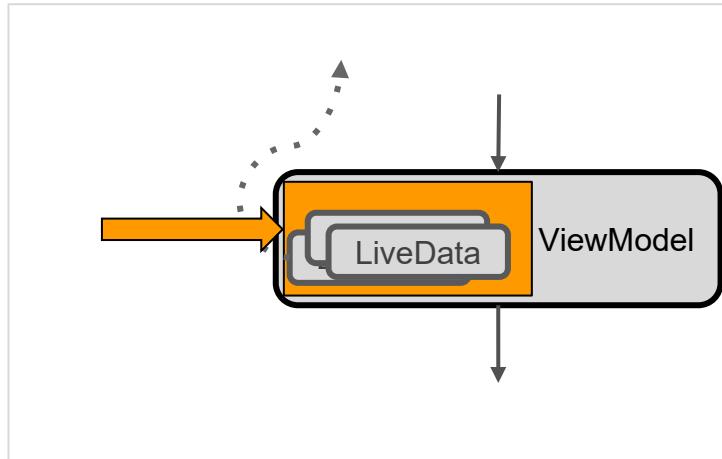


# LiveData

# LiveData

LiveData is a data holder class that is aware of lifecycle events. It keeps a value and allows this value to be observed.

Use LiveData to keep your UI up to date with the latest and greatest data.



# LiveData

- LiveData is observable data
- Notifies observer when data changes
- Is lifecycle aware:  
knows when device rotates  
or app stops



# Use LiveData to keep UI up to date

- Create an observer that observes the LiveData
- LiveData notifies Observer objects when the observed data changes
- Your observer can update the UI every time the data changes



# Creating LiveData

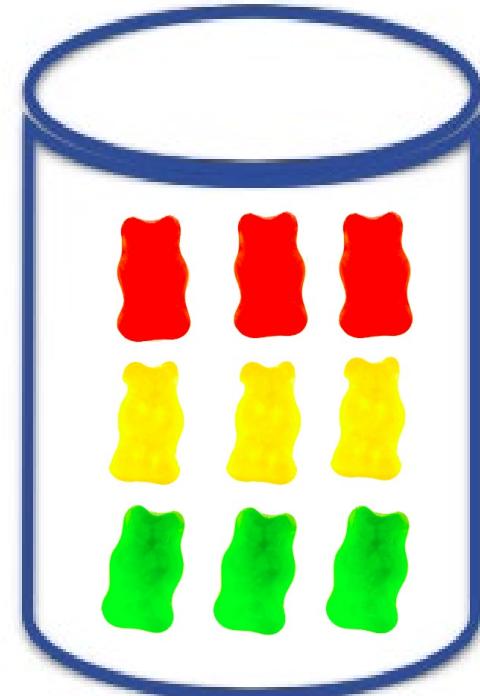
To make data observable, return it as LiveData:

```
@Query("SELECT * from word_table")
LiveData<List<Word>> getAllWords();
```

# Using LiveData with Room



Room generates all  
the code to update the  
LiveData when the  
database is updated



# Passing LiveData through layers

When you pass live data through the layers of your app architecture, from a Room database to your UI, that data must be `LiveData` in all layers:

- DAO
- ViewModel
- Repository

# Passing LiveData through layers

- DAO:

```
@Query("SELECT * from word_table")  
LiveData<List<Word>> getAllWords();
```

- Repository:

```
LiveData<List<Word>> mAllWords =  
    mWordDao.getAllWords();
```

- ViewModel:

```
LiveData<List<Word>> mAllWords =  
    mRepository.getAllWords();
```



# Observing LiveData

- Create the observer in `onCreate()` in the Activity
- Override `onChanged()` in the observer to update the UI when the data changes

When the LiveData changes, the observer is notified and its `onChanged()` is executed

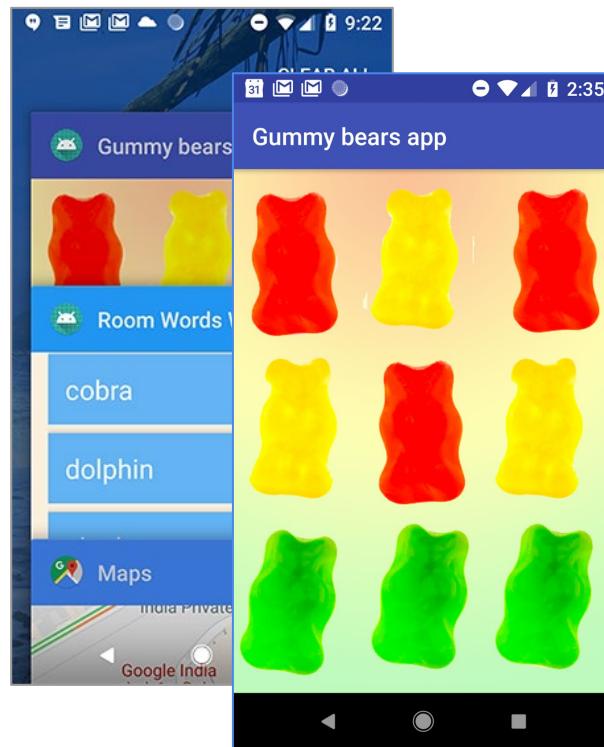
# Observing LiveData: example

```
final Observer<String> nameObserver =  
    new Observer<String>() {  
        @Override  
        public void onChanged(@Nullable final String newName) {  
            // Update the UI, in this case, a TextView.  
            mNameTextView.setText(newName);  
        }  
    };  
  
mModel.getCurrentName().observe(this, nameObserver);
```

# No memory leaks

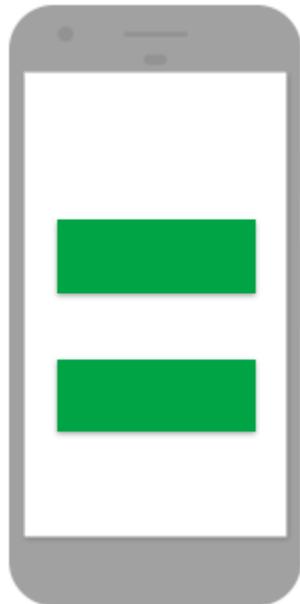
- Observers are bound to [Lifecycle](#) objects which are objects that have an Android Lifecycle
- Observers clean up after themselves when their associated lifecycle is destroyed

# LiveData is always up to date



- If a lifecycle object becomes inactive, it gets the latest data when it becomes active again
- Example: an activity in the background gets the latest data right after it returns to the foreground

# LiveData handles configuration changes



If an activity or fragment is re-created due to a configuration change such as device rotation, the activity or fragment immediately receives the latest available data



# Share resources

- You can extend a LiveData object using the [singleton](#) pattern, for example for services or a database
- The LiveData object connects to the system service once, and then any observer that needs the resource can just watch the LiveData object
- See [Extend LiveData](#)



# Lifecycle

# Lifecycle-aware components

Instead of managing lifecycle-dependent components in the activity's lifecycle methods, `onStart()`, `onStop()`, and so on, you can make any class react to lifecycle events

# Lifecycle-aware components

- Lifecycle-aware components perform actions in response to a change in the lifecycle status of another component
- For example, a listener could start and stop itself in response to an activity starting and stopping

# Use cases

- Switch between coarse and fine-grained location updates depending on app visibility
- Stop and start video buffering
- Stop network connectivity when app is in background
- Pause and resume animated drawables



# Lifecycle library

- Import the [android.arch.lifecycle](#) package
- Provides classes and interfaces that let you build lifecycle-aware components that automatically adjust their behavior based on lifecycle state of activity or fragment
- See [Handling Lifecycles with Lifecycle-Aware Components](#)

# LifecycleObserver interface

- LifecycleObserver has an Android lifecycle.
- It does not have any methods, instead, uses OnLifecycleEvent annotated methods.

# @OnLifecycleEvent

`@OnLifecycleEvent` indicates life cycle methods

```
@OnLifecycleEvent(Lifecycle.Event.ON_START)
public void start() {...}
```

```
@OnLifecycleEvent(Lifecycle.Event.ON_STOP)
public void start() {...}
```

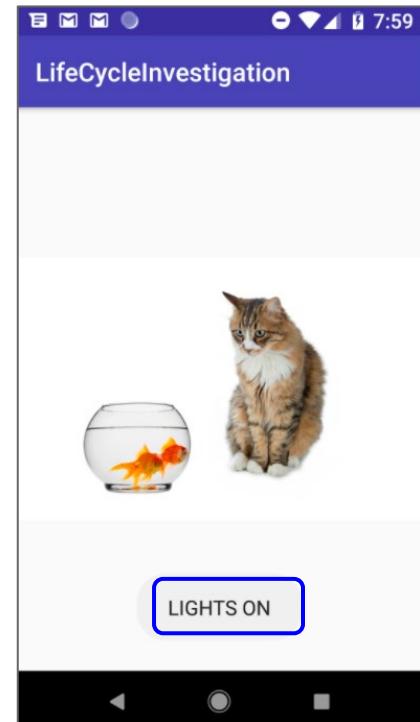
See [Lifecycle.event reference](#) for more lifecycle events

# POJOs can be life cycle aware

You can make *any class* react to lifecycle events



*In these pictures,  
the Toast is  
created by a plain  
old Java class  
when app starts  
or device rotates*



# Adding lifecycle awareness to a POJO

```
public class Aquarium {  
  
    // Constructor takes Application and lifecycle  
    public Aquarium(final Application app,  
                    Lifecycle lifecycle) {  
  
        ...  
    }  
}
```

# Constructor for lifecycle aware POJO

```
public Aquarium(final Application app, Lifecycle lifecycle) {  
    // Add a new observer to the lifecycle.  
    lifecycle.addObserver(new LifecycleObserver() {  
  
        @OnLifecycleEvent(Lifecycle.Event.ON_START)  
  
        public void start() {  
            Toast.makeText(app, "LIGHTS ON", Toast.LENGTH_SHORT).show();  
        }  
    });  
}
```

# Creating an instance

```
public class MainActivity extends AppCompatActivity {  
    private Aquarium myAquarium;  
  
    protected void onCreate(...) {  
        ...  
        // Create aquarium.  
        // Pass context and this activity's lifecycle  
        myAquarium = new Aquarium(this.getApplication(),  
            getLifecycle());  
    }  
}
```



# What's next?

- Concept chapter: [10.1 Room, LiveData, and ViewModel](#)
- Practical: [10.1A : Room, LiveData, and ViewModel](#)
- Practical: [10.1B : Room, LiveData, and ViewModel](#)

# END