



# Android Developer Associate





Android  
Developer  
Associate

# 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

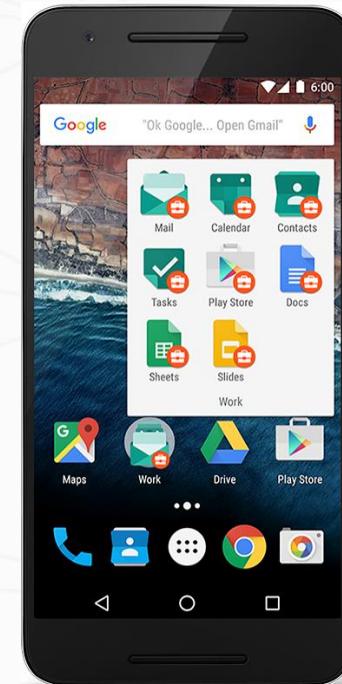
# Android and sensors

Sensors can discover user action and respond

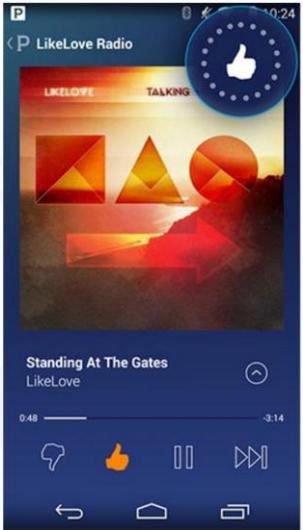
- Device contents rotate as needed
- Walking adjusts position on map
- Tilting steers a virtual car or controls a physical toy
- Moving too fast disables game interactions

# Android home screen

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



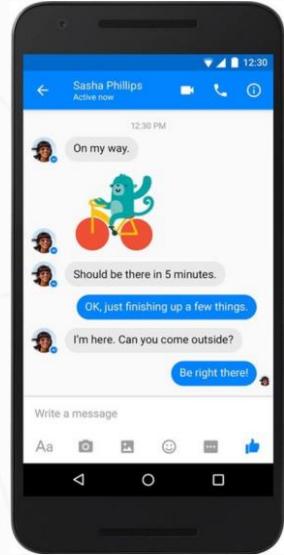
# Android app examples



Pandora



Pokemon GO

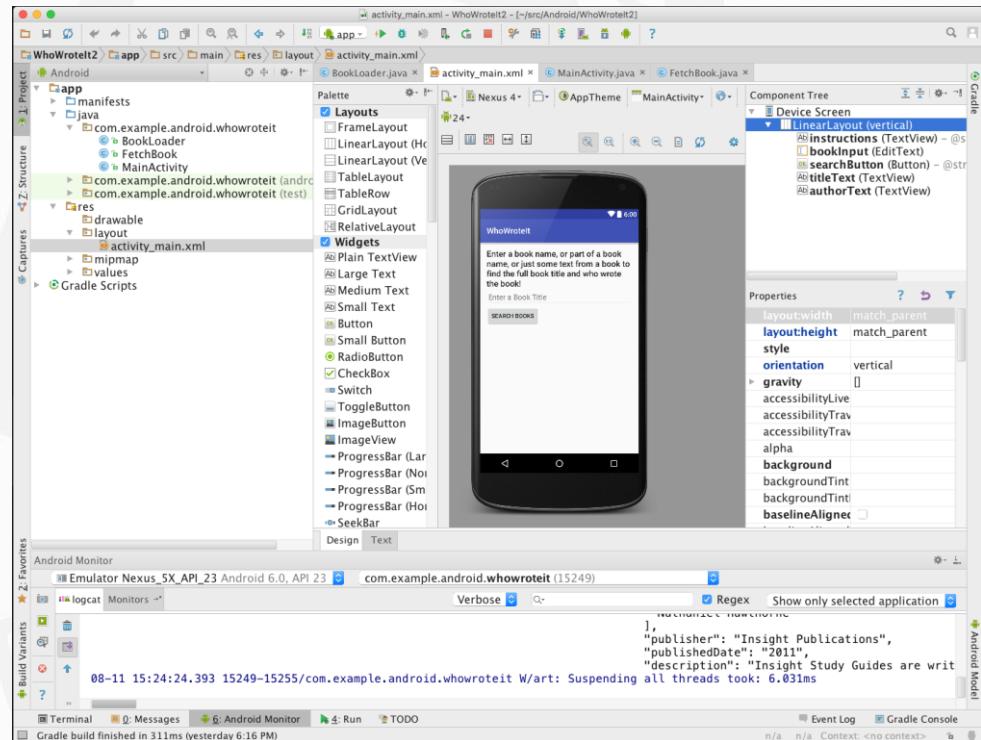


Facebook  
Messenger

# Android Software Developer Kit (SDK)

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

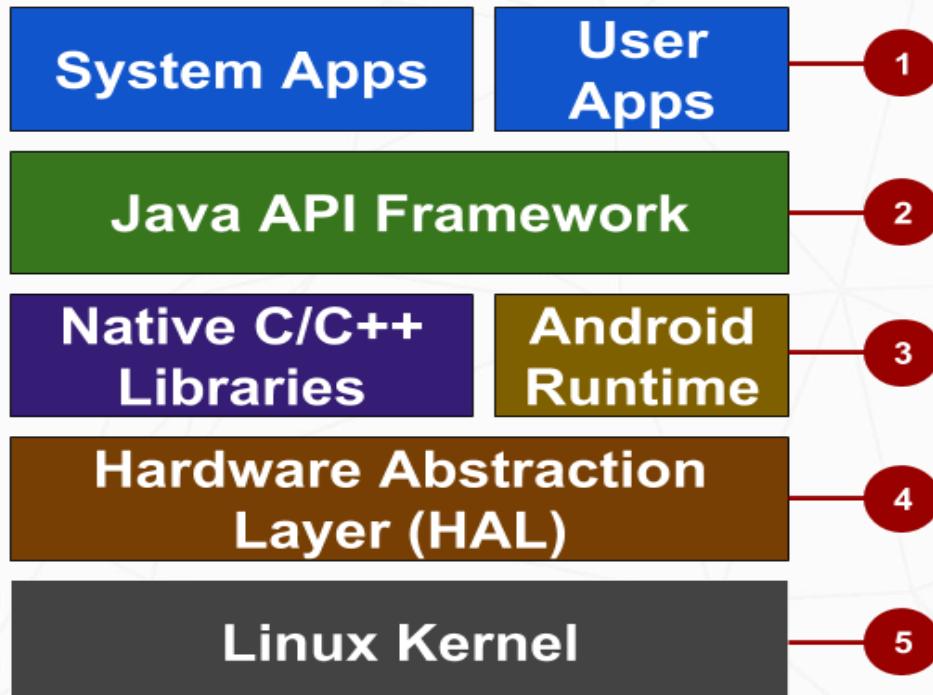


# Architecture

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

# 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
- Content providers to access data from other apps

# 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

# 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
<b>Marshmallow</b>	6.0 - 6.0.1	Oct 2015	23
<b>Nougat</b>	7.0	Sept 2016	24

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

# 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

# Component types

- **Activity** is a single screen with a user interface
- **Service** performs long-running tasks in background
- **Content provider** manages shared set of data
- **Broadcast receiver** responds to system-wide announcements

# Think of Android as a hotel

- Your app is the guest
- The Android System is the hotel manager
- Services are available when you request them (intents)
  - In the foreground (activities) such as registration
  - In the background (services) such as laundry
- Calls you when a package has arrived (broadcast receiver)
- Access the city's tour companies (content provider)

# 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 C Introduction to Android](#)
- Practical: –

**END**



Android  
Developer  
Associate

Lesson 1



# .1 Create Your First Android App

# Contents

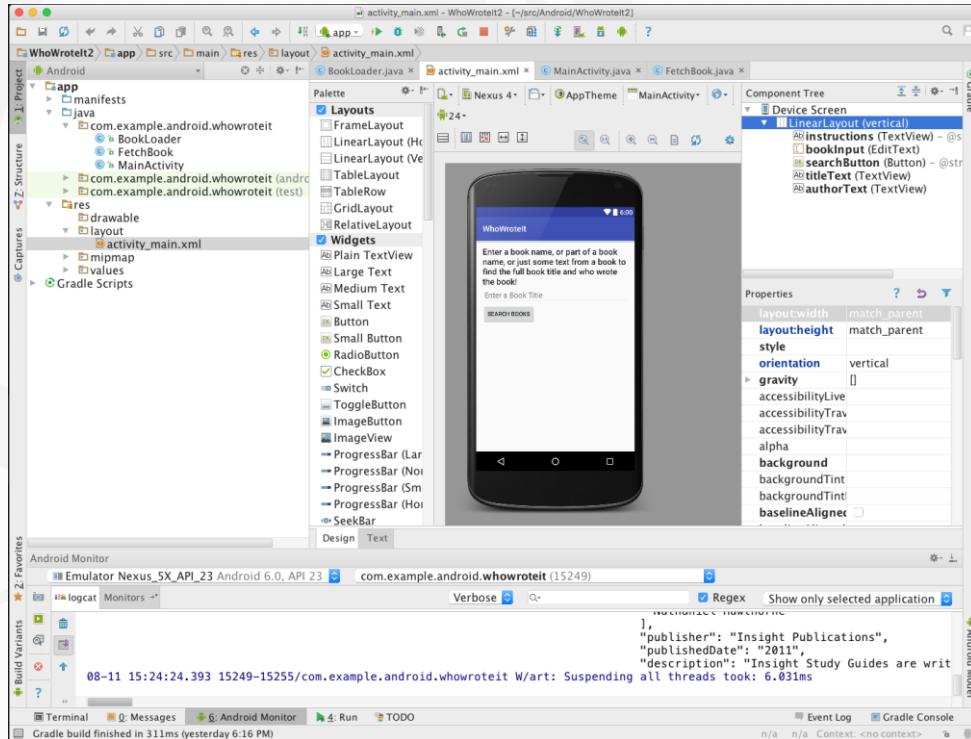
- Android Studio
- Creating "Hello World" app in Android Studio
- Basic app development workflow with Android Studio
- Running apps on virtual and physical devices

# Prerequisites

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

# Android Studio

# What is Android Studio?



- Android IDE
- Project structure
- Templates
- Layout Editor
- Testing tools
- Gradle-based build
- Log Console
- Debugger
- Monitors
- Emulators

# Installation Overview

- Mac, Windows, or Linux
- Requires Java Development Kit (JDK) 1.7 or better from [Oracle Java SE downloads page](#)
- Set JAVA\_HOME to JDK installation location
- Download and install Android Studio from [http://developer.android.com/sdk/index.html](#)
- See [1.1 P Install Android Studio](#) for details

# First Android App

# Start Android Studio



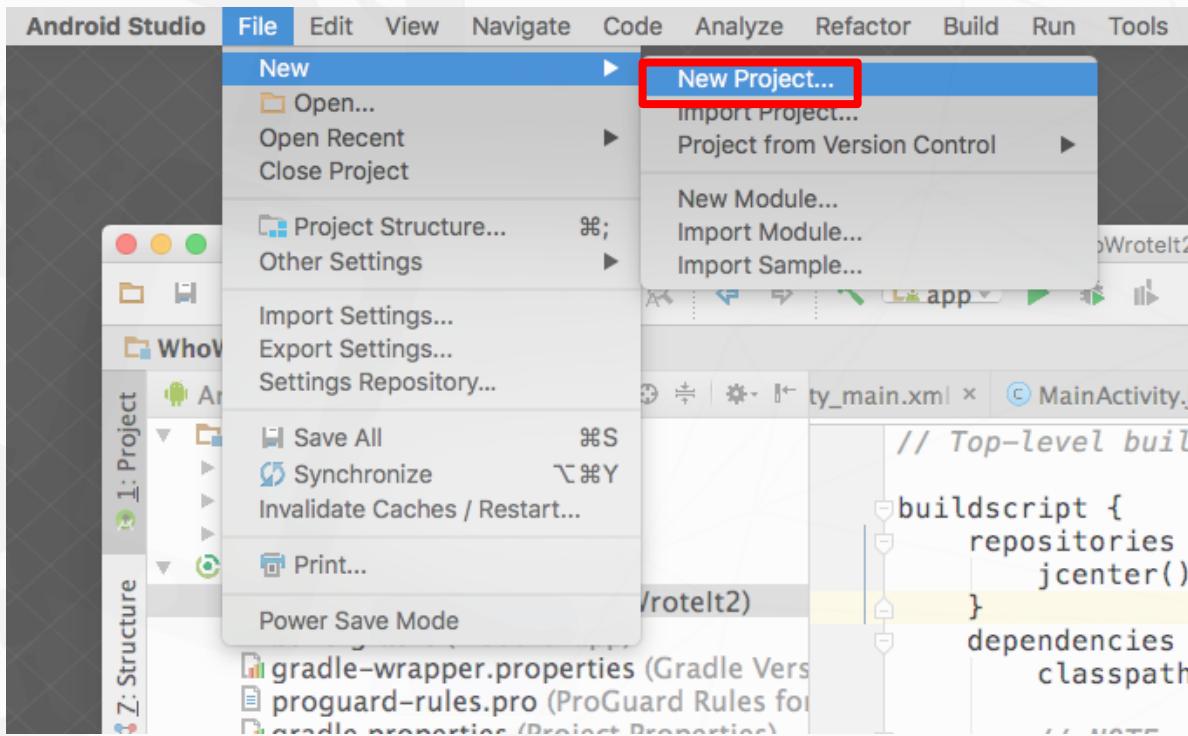
Welcome to Android Studio

The screenshot shows the 'Welcome to Android Studio' screen. At the top center is the Android logo. Below it is the text 'Android Studio'. Underneath that is the version information 'Version 2.2 Beta 2 (AI-145.3200535)'. There are five main options listed: 'Start a new Android Studio project' (highlighted with a red border), 'Open an existing Android Studio project', 'Check out project from Version Control', 'Import project (Eclipse ADT, Gradle, etc.)', and 'Import an Android code sample'. At the bottom right are 'Configure' and 'Get Help' buttons.

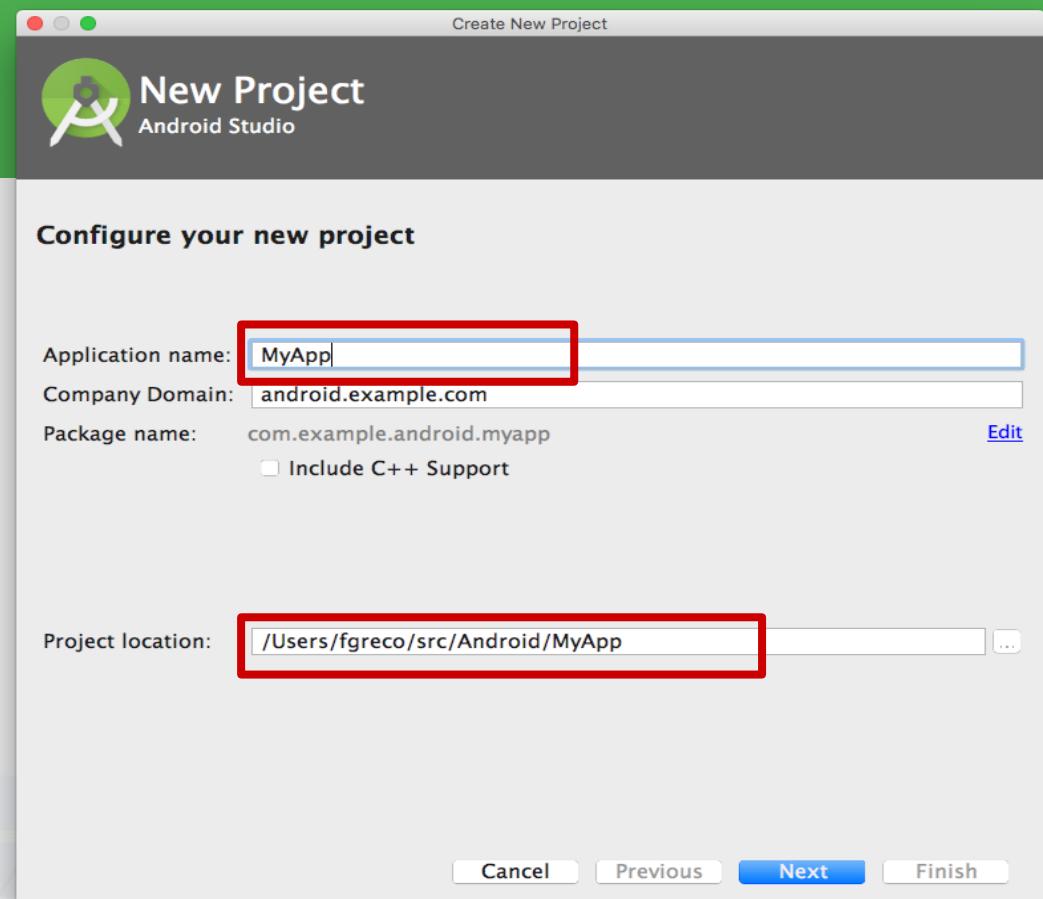
- Start a new Android Studio project
- Open an existing Android Studio project
- Check out project from Version Control ▾
- Import project (Eclipse ADT, Gradle, etc.)
- Import an Android code sample

Configure ▾ Get Help ▾

# Create a project inside Android Studio



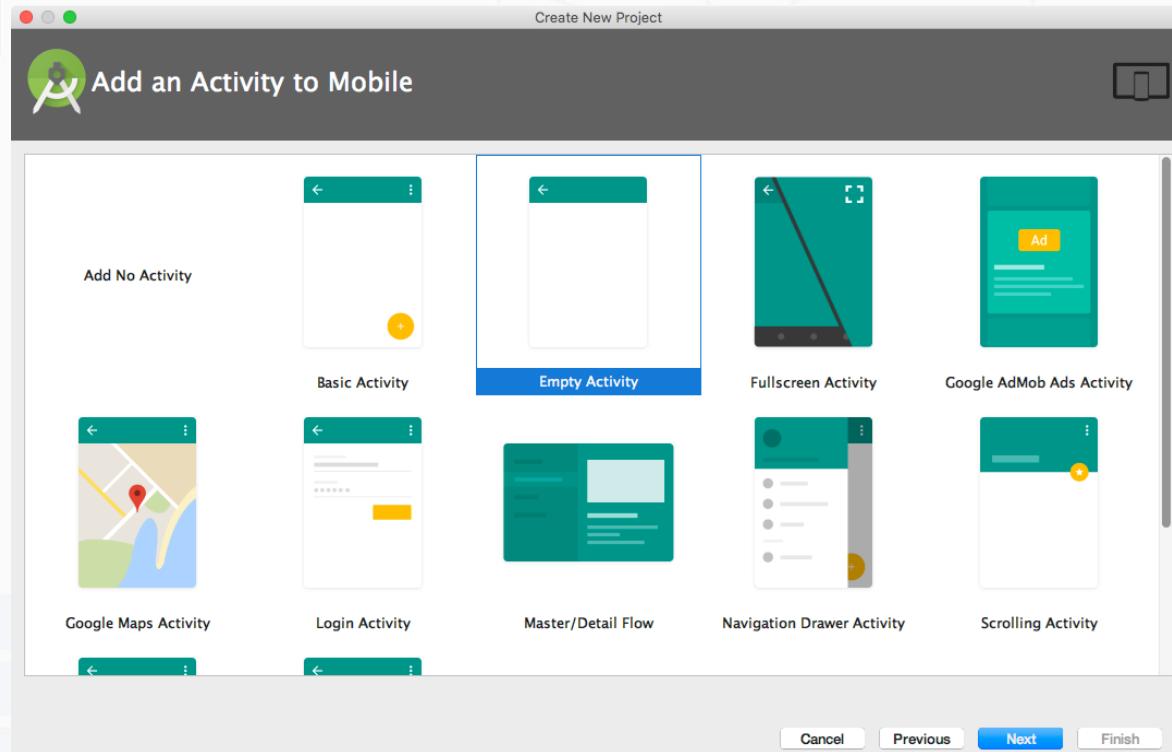
# Name your app



# Pick activity template

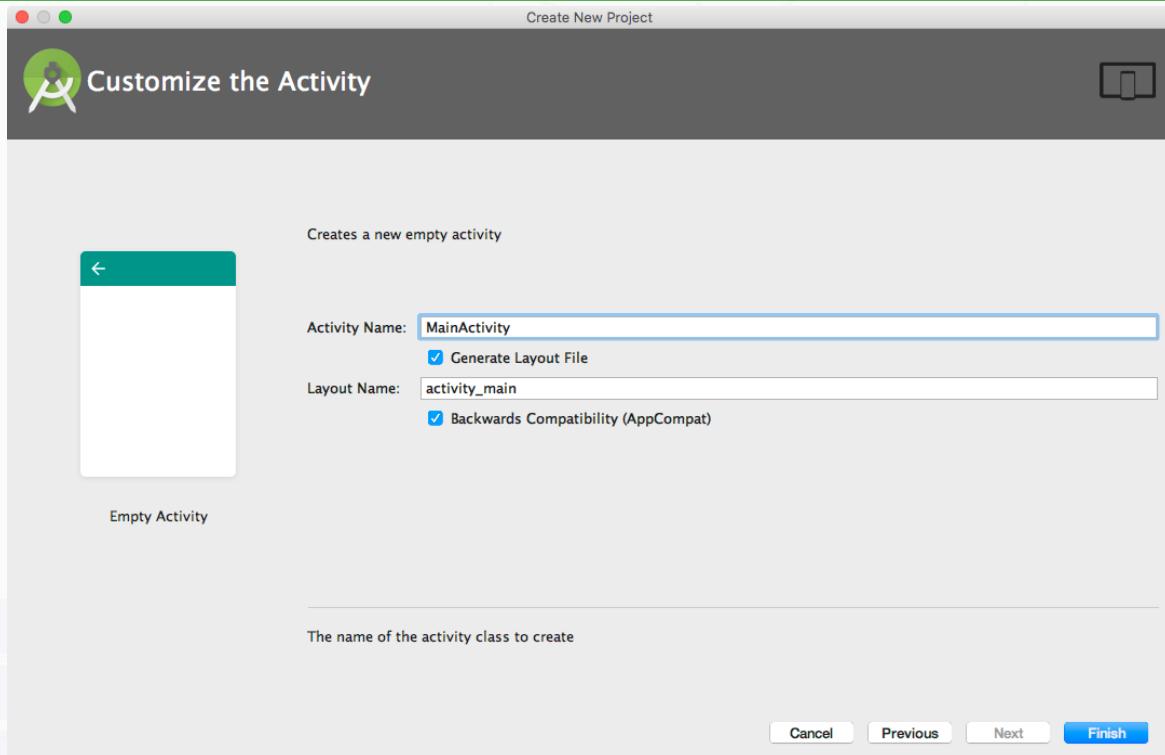
Choose templates for common activities, such as maps or navigation drawers.

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



# Name your activity

- Good practice to name main activity `MainActivity` and `activity_main` layout
- Use AppCompat
- Generating layout file is convenient



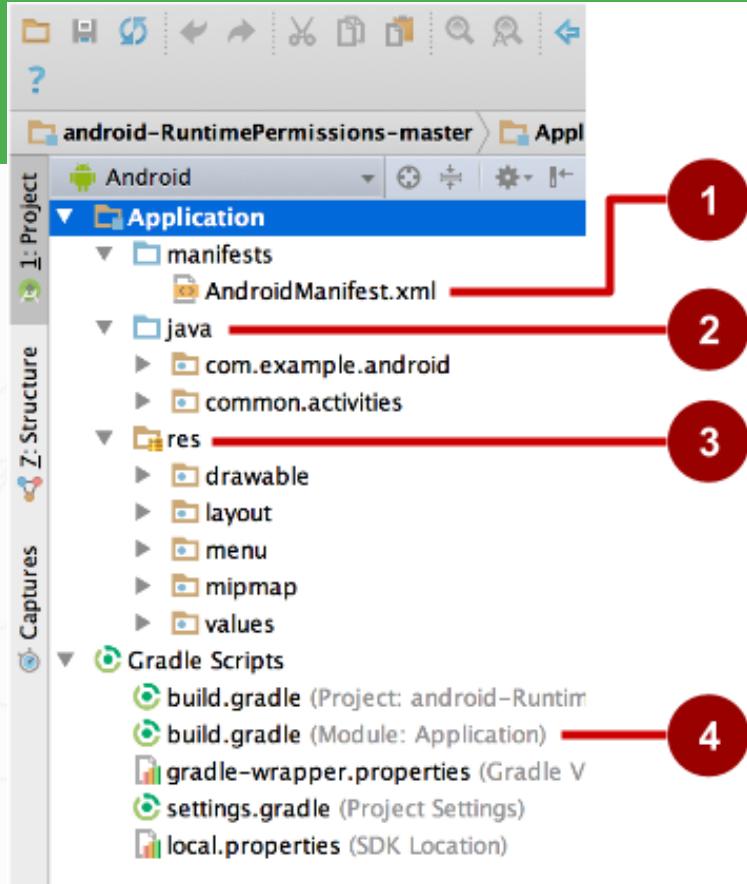
# Android Studio Panes

The screenshot illustrates the Android Studio interface with several open panes:

- Project Files**: Located on the left, this pane shows the project structure with the `activity_main.xml` file selected.
- Layout Editor**: The central pane displays the XML layout for the `activity_main.xml` file. It features a large yellow `TOAST` button with a blue number `0` in the center. The layout includes a `LinearLayout` containing a `TextView` labeled `show_count` and a `Button` labeled `button_toast`. Below the yellow button is a blue `COUNT` button. The `Properties` pane on the right shows the `show_count` `TextView` has an `ID` of `show_count`, `layout_width` of `match_parent`, and `layout_height` of `wrap_content`.
- Android Monitor**: The bottom pane shows logcat output for the emulator. The log includes messages such as `09-26 16:29:17.556 D/`, `09-26 16:29:17.620 D/`, `09-26 16:29:17.627 D/`, and `09-26 16:29:17.642 D/`. The logcat window also includes options for `Monitors` and `logcat`.

# Project folders

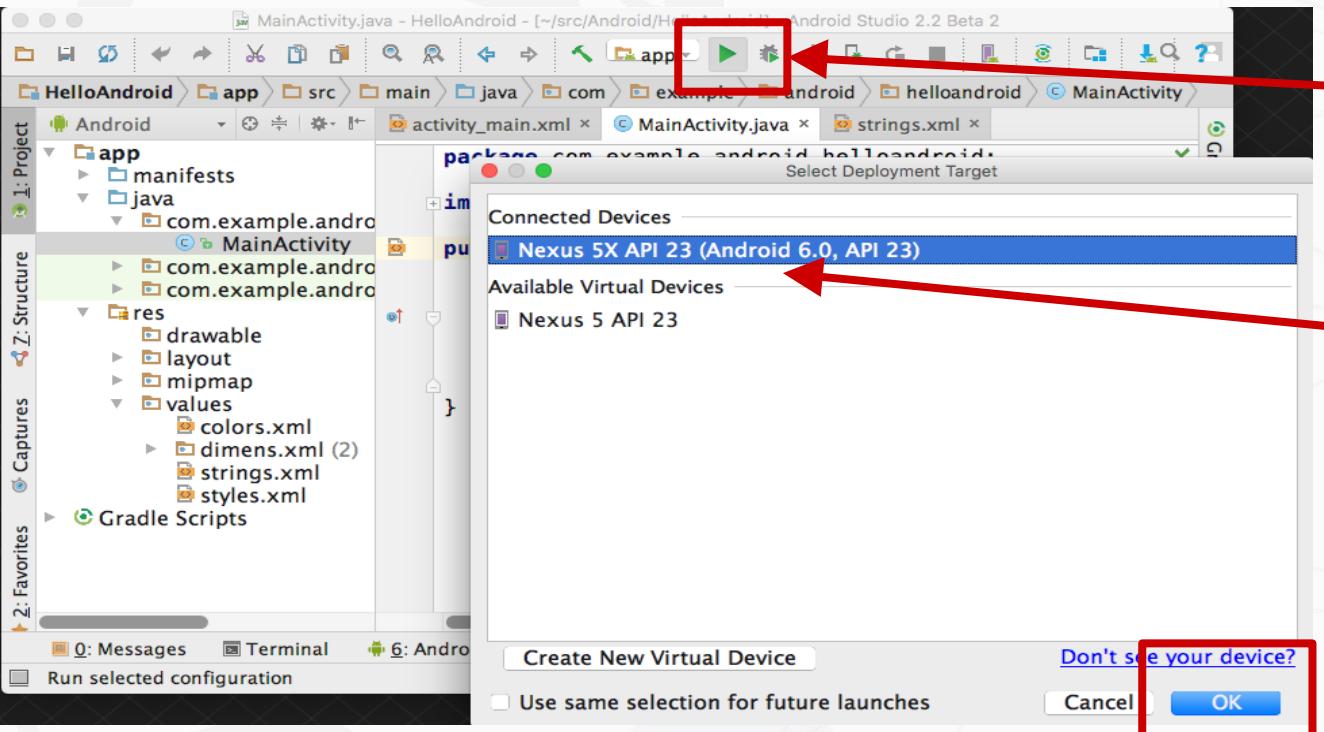
- 1. manifests**—Android Manifest file - description of app read by the Android runtime
- 2. java**—Java source code packages
- 3. res**—Resources (XML) - layout, strings, images, dimensions, colors...
- 4. build.gradle**—Gradle build files



# 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

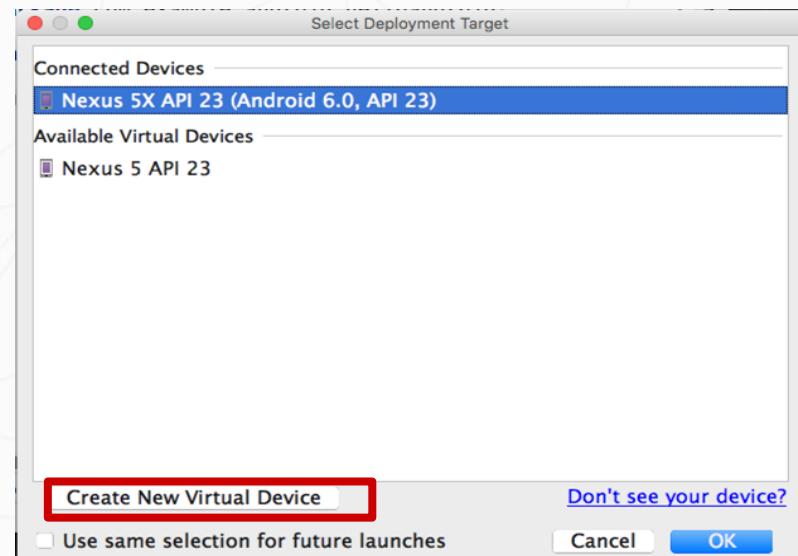
3. OK

# Create a virtual device

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

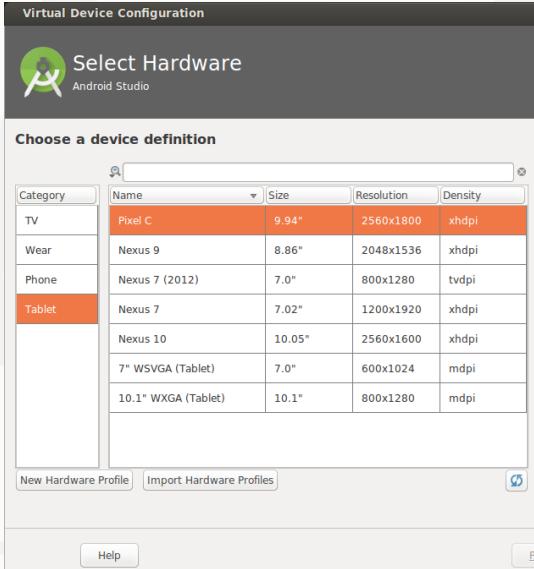
Tools > Android > AVD Manager

or:

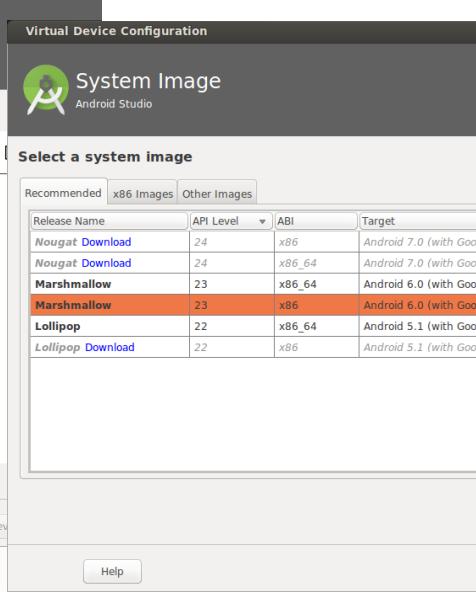


# Configure virtual device

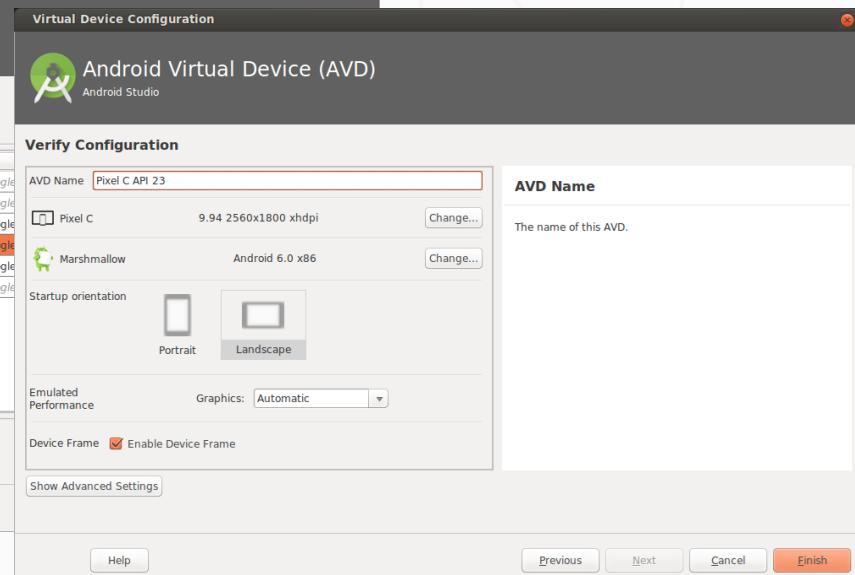
## 1. Choose hardware



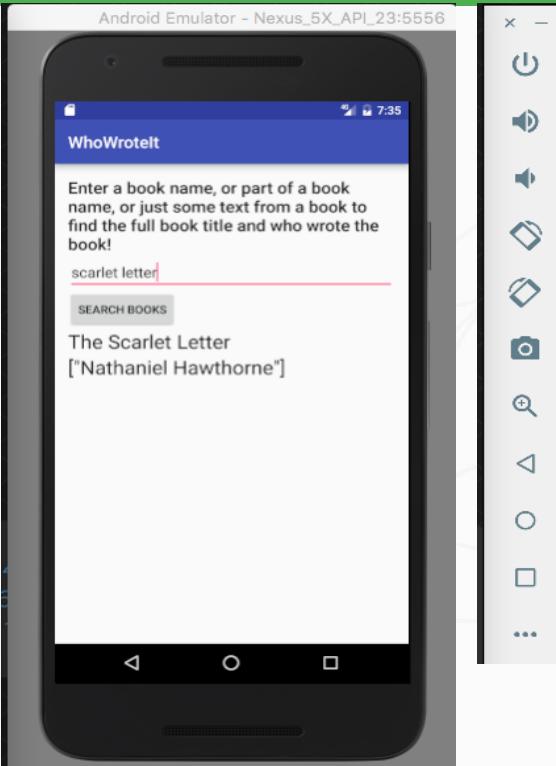
## 2. Select Android Version



## 3. Finalize



# Run on a virtual device



# Run on a physical device

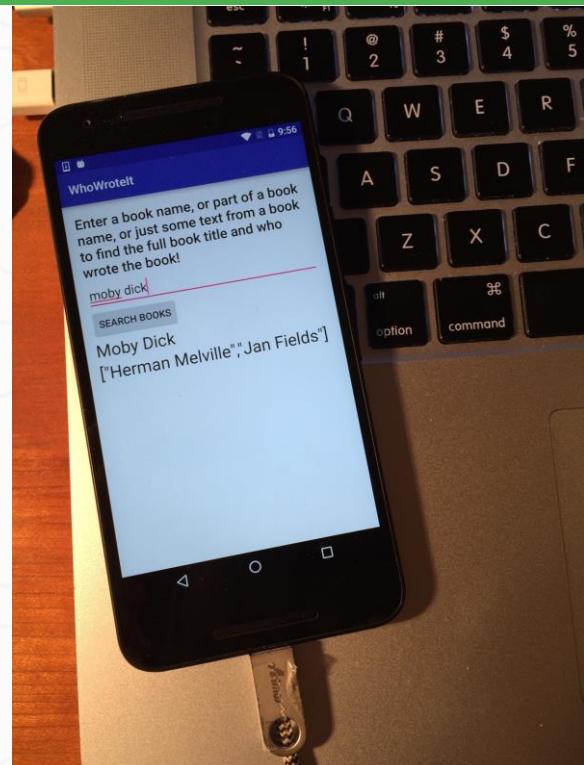
1. Turn on Developer Options:
  - a. **Settings > About phone**
  - b. Tap **Build number** seven times
2. Turn on USB Debugging
  - a. **Settings > Developer Options > USB Debugging**
3. Connect phone to computer with cable

Windows/Linux additional setup:

- [Using Hardware Devices](#)

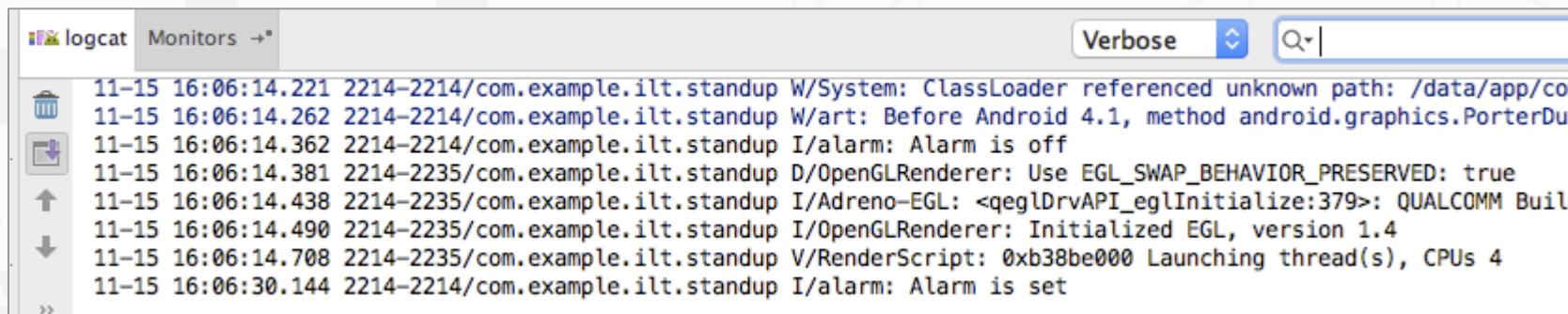
Windows drivers:

- [OEM USB Drivers](#)



# Get feedback as your app runs

- As the app runs, Android Monitor logcat shows information
- You can add logging statements to your app that will show up in logcat.



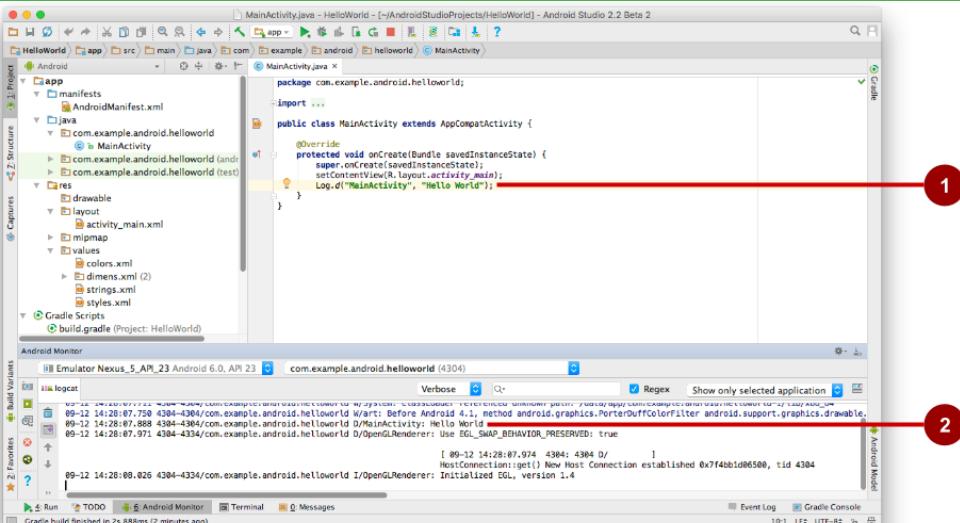
The screenshot shows the Android Monitor application window with the 'logcat' tab selected. The interface includes a toolbar with icons for logcat, monitors, and a search bar. The main area displays a list of log entries. The log entries are as follows:

```
11-15 16:06:14.221 2214-2214/com.example.ilt.standup W/System: ClassLoader referenced unknown path: /data/app/com.example.ilt.standup-1/lib/arm64
11-15 16:06:14.262 2214-2214/com.example.ilt.standup W/art: Before Android 4.1, method android.graphics.PorterDuffColorFilter android.graphics.drawable.Drawable.setColorFilter(int, PorterDuffColorFilter) was defined as void in class android.graphics.drawable.Drawable and @Deprecated annotated. This has been changed to int in 4.1 and will be defined in a class derived from Drawable in 4.4.
11-15 16:06:14.362 2214-2214/com.example.ilt.standup I/Alarm: Alarm is off
11-15 16:06:14.381 2214-2235/com.example.ilt.standup D/OpenGLRenderer: Use EGL_SWAP_BEHAVIOR_PRESERVED: true
11-15 16:06:14.438 2214-2235/com.example.ilt.standup I/Adreno-EGL: <qeglDrvAPI_eglInitialize:379>: QUALCOMM Build ID: 1.1.0.0
11-15 16:06:14.490 2214-2235/com.example.ilt.standup I/OpenGLRenderer: Initialized EGL, version 1.4
11-15 16:06:14.708 2214-2235/com.example.ilt.standup V/RenderScript: 0xb38be000 Launching thread(s), CPUs 4
11-15 16:06:30.144 2214-2214/com.example.ilt.standup I/Alarm: Alarm is set
```

# Logging

```
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...");
```

# Android Monitor > logcat pane



1. Log statements in code.
2. logcat pane shows system and logging messages

- Set filters to see what's important to you
- Search using tags

# Learn more

- [Meet Android Studio](#)
- Official Android documentation at [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 C Create Your First Android App](#)
- Practical: [1.1 P Install Android Studio and Run Hello World](#)

**END**



Android  
Developer  
Associate

Lesson 1



# 1.2 Views, Layouts, and Resources

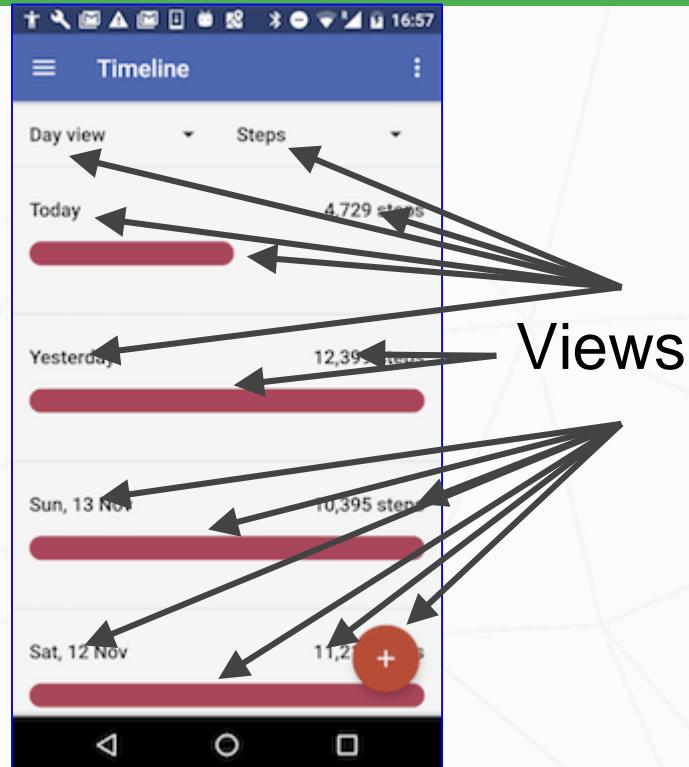
# Contents

- Views, view groups, and view hierarchy
- Layouts in XML and Java code
- Event Handling
- Resources
- Screen 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**.



# What is a view

Views are Android's basic user interface building blocks.

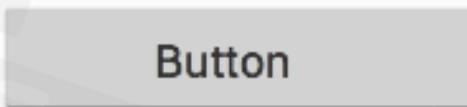
- display text ([TextView](#) class), edit text ([EditText](#) class)
- buttons ([Button](#) class), [menus](#), other controls
- scrollable ([ScrollView](#), [RecyclerView](#))
- show images ([ImageView](#))
- subclass of [View](#) class

# Views have properties

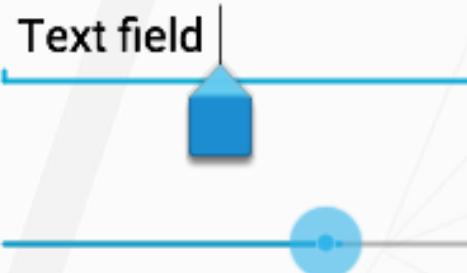
- Have properties (e.g., color, dimensions, positioning)
- May have focus (e.g., selected to receive user input)
- May be interactive (respond to user clicks)
- May be visible or not
- Have relationships to other views

# Examples of views

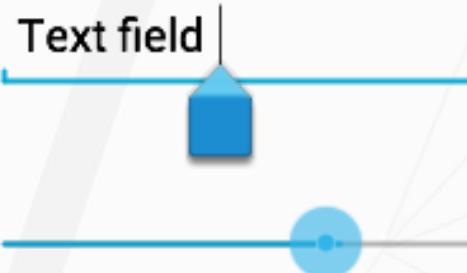
Button



EditText



SeekBar



CheckBox

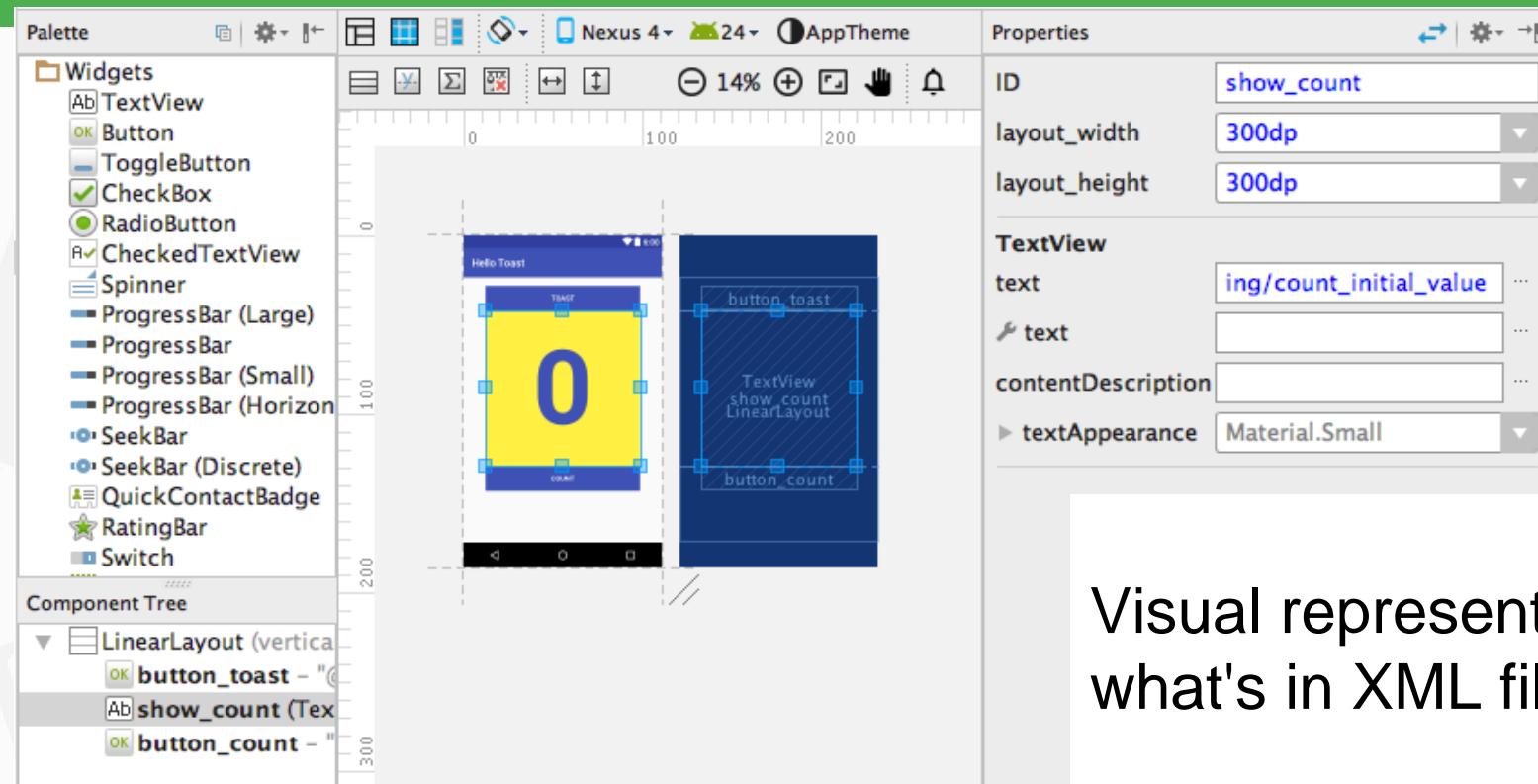
RadioButton

Switch

# Creating and laying out views

- Graphically within Android Studio
- XML Files
- Programmatically

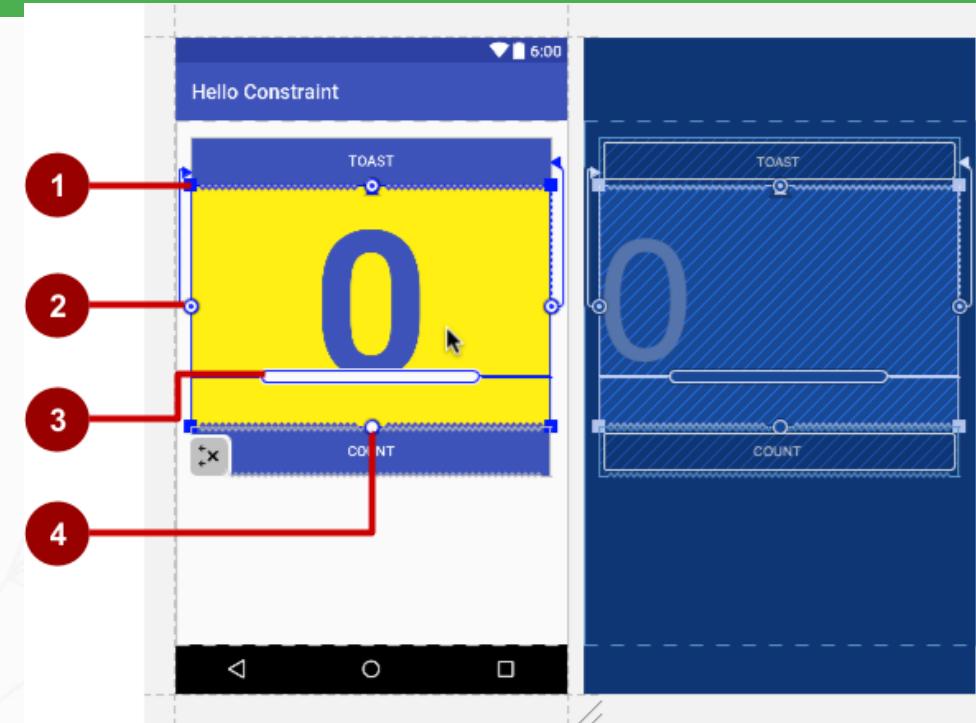
# Views defined in Layout Editor



Visual representation of what's in XML file.

# Using the Layout Editor

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



# Views 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 properties 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:

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

*context*



# 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 & View Hierarchy

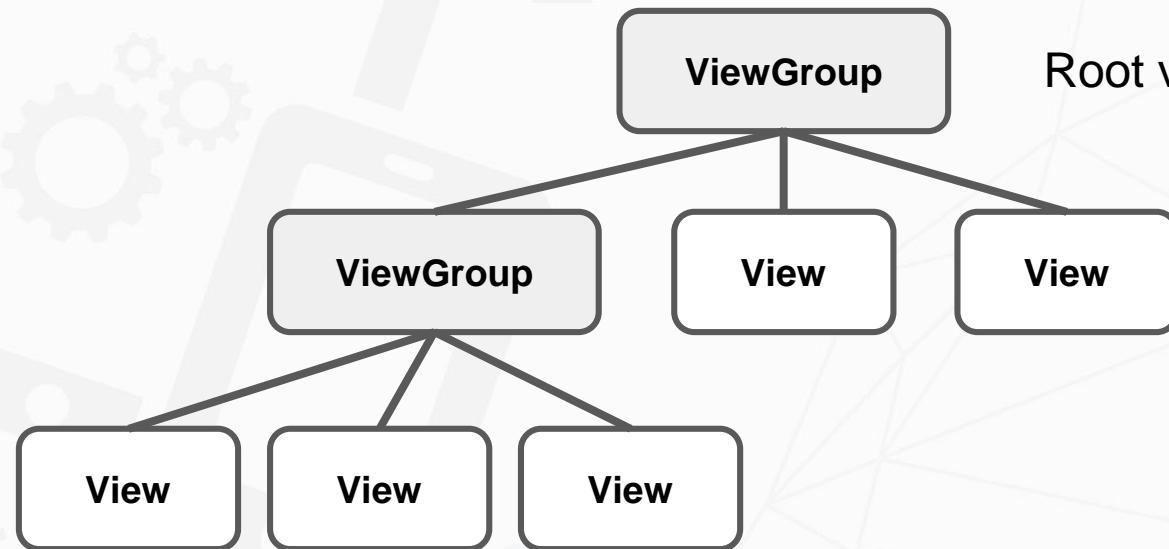
# ViewGroup views

A [ViewGroup](#) (parent) is a type of view that can contain other views (children)

ViewGroup is the base class for layouts and view containers

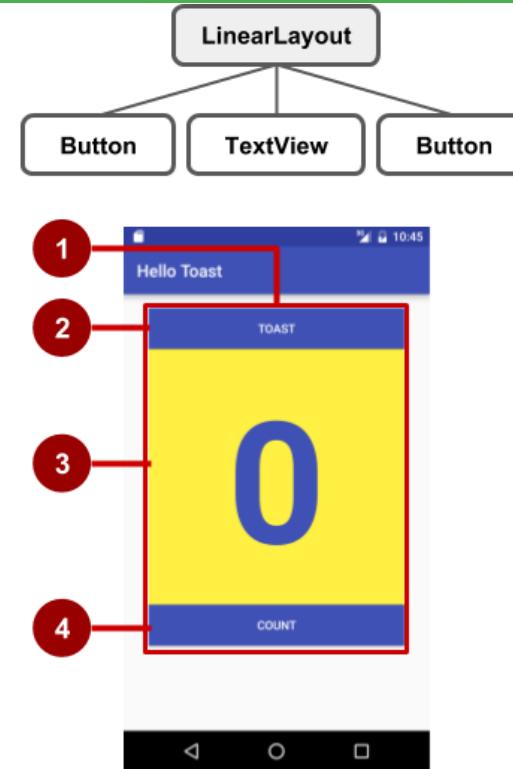
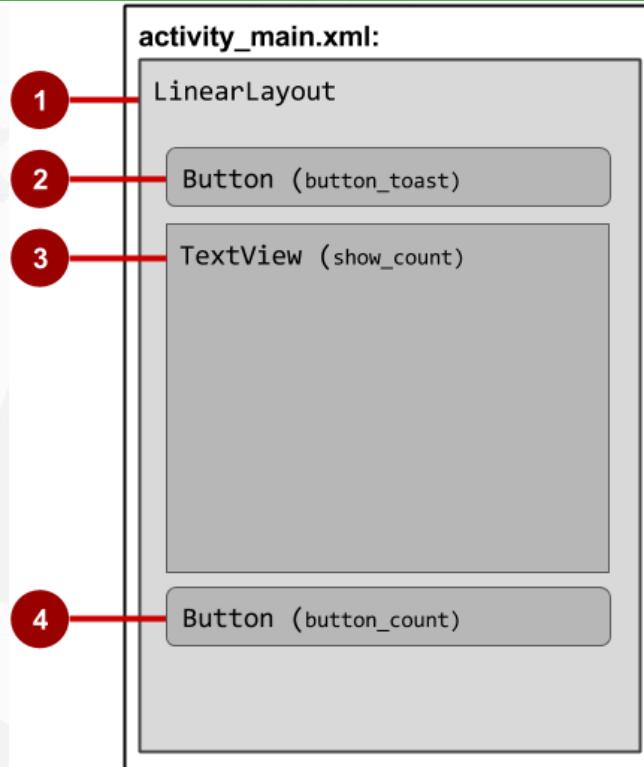
- ScrollView—scrollable view that contains one child view
- LinearLayout—arrange views in horizontal/vertical row
- RecyclerView—scrollable "list" of views or view groups

# Hierarchy of view groups and views

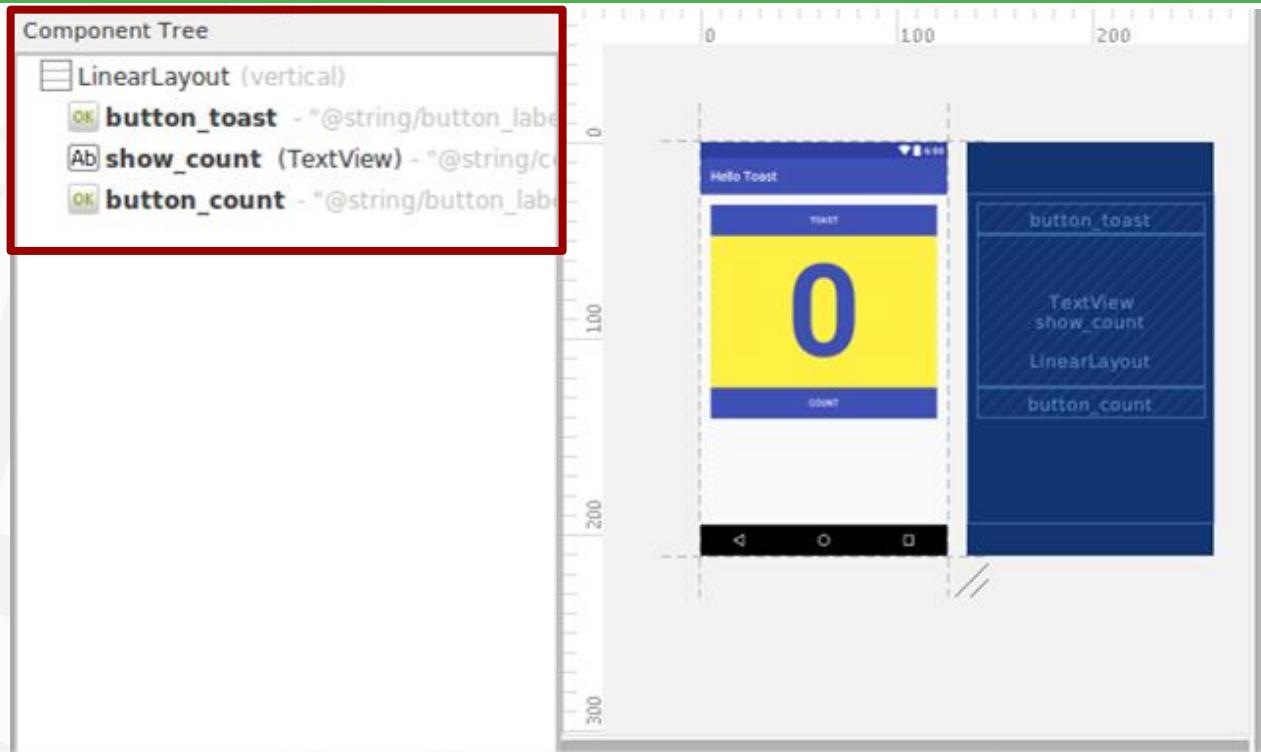


Root view is always a view group

# View hierarchy and screen layout



# View hierarchy in the component tree



# 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

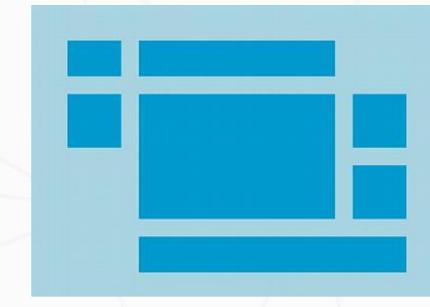
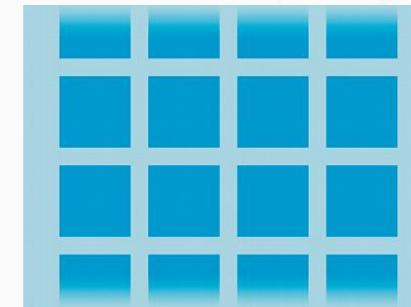
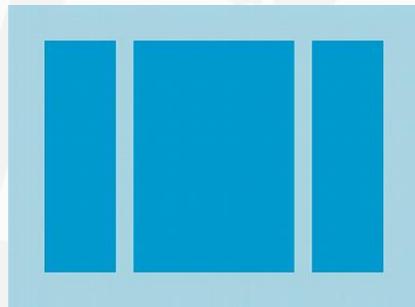
# Layouts

# Layout Views

## Layouts

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

# Common Layout Classes



LinearLayout

RelativeLayout

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
- **GridView** - 2D scrollable grid

# Class Hierarchy vs. Layout Hierarchy

- View class-hierarchy is standard object-oriented class inheritance
  - For example, Button is-a TextView is-a View is-a 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

# Layout created in XML

```
<LinearLayout  
    android:orientation="vertical"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
    <EditText  
        ... />  
    <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);
```

# Setting width and height in Java code

Set the width and height of a view:

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

# 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

# Handling clicks in XML & Java

Attach handler to view in layout:

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

Implement handler in activity:

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

# Setting click handlers in Java

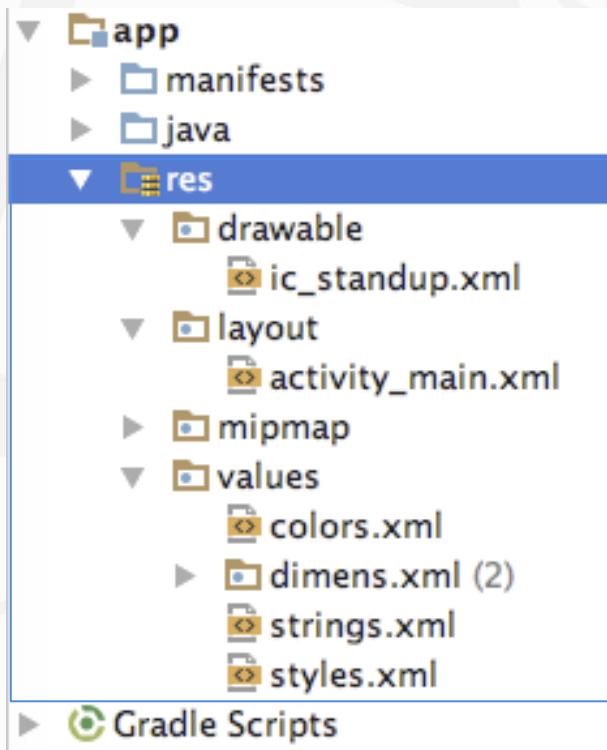
```
final Button button = (Button) findViewById(R.id.button_id);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        String msg = "Hello Toast!";
        Toast toast = Toast.makeText(this, msg, duration);
        toast.show();
    }
});
```

# Resources

# 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

- Device Independent Pixels (dp) - for Views
- Scale Independent Pixels (sp) - for text

Don't use device-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](#)
- [Hierarchy Viewer](#) for visualizing the view hierarchy

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 C Layouts, Views, and Resources](#)
- Practicals:
  - [1.2A P Make Your First Interactive UI](#)
  - [1.2B P Using Layouts](#)

**END**



Android  
Developer  
Associate

Lesson 1



# 1.3 Text and Scrolling Views

# Contents

- TextView
- ScrollView

# TextView

# TextView for text

- [TextView](#) is a view for displaying single and multi-line text
- [EditText](#) is a subclass of TextView with editable text
- Controlled with layout attributes
- Set text statically from a string resource in XML, or 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?

- The user may need to scroll.
  - News stories, articles, ...
- To allow users to scroll a `TextView`, embed it in a **ScrollView**.
- Other Views can be embedded in a **ScrollView**.
  - `LinearLayout`, `TextView`, `Button`, ...

# ScrollView for scrolling content

- [ScrollView](#) is a subclass of [FrameLayout](#)
- Can only hold **one** view (which can be a ViewGroup)
- 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

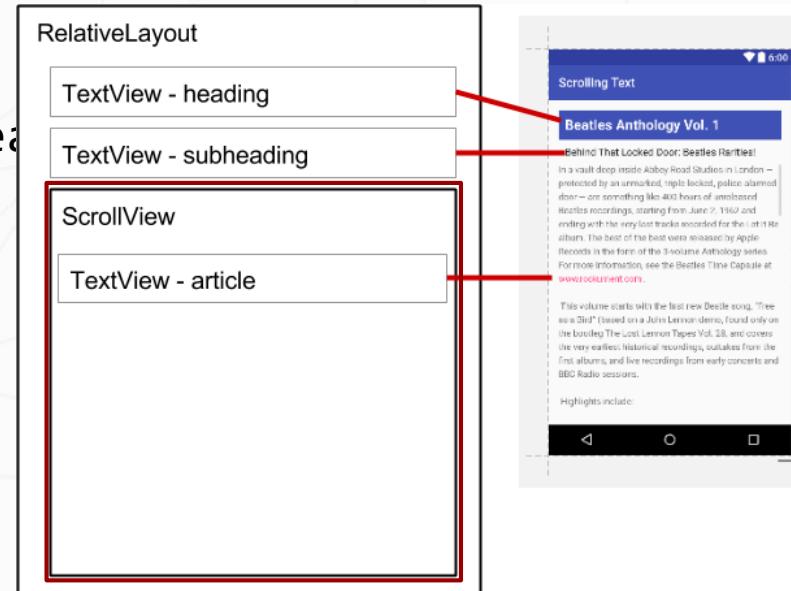
```
<ScrollView
```

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

```
<TextView
```

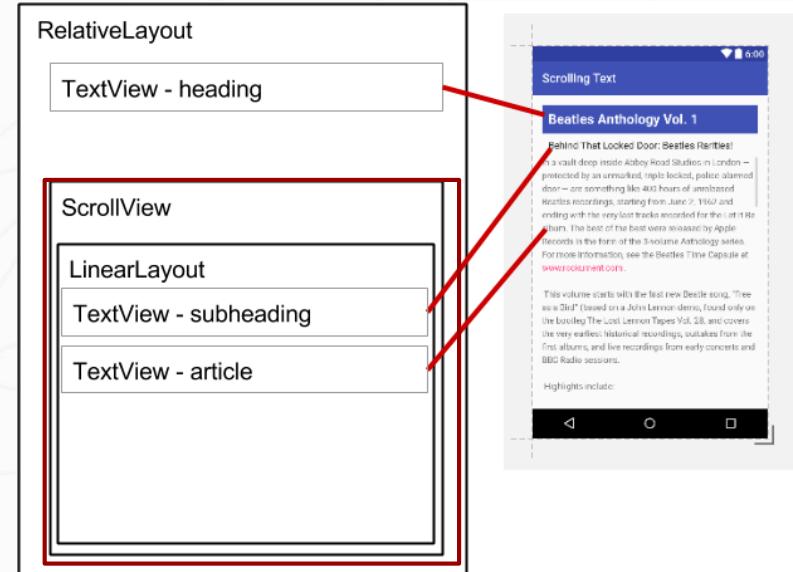
```
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    .../>
```

```
</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 C Text and Scrolling Views](#)
- Practical: [1.3 P Working with TextView Elements](#)

# END





Android  
Developer  
Associate

Lesson 1



# 1.4 Resources that 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 developer documentation website. The top navigation bar includes a 'Developers' icon, 'DESIGN', 'DEVELOP', 'DISTRIBUTE', a search bar, and a 'DEVELOPER CONSOLE' button. The left sidebar has a 'HOME' section with links for 'Android', 'Wear', 'TV', and 'Auto', and sections for 'DESIGN', 'DEVELOP', and 'DISTRIBUTE'. The main content area features a title 'Build Beautiful Apps' and a subtitle 'Resources to get you started with designing and developing for Android.' Below this are three cards: 'Building Apps for Wearables' (TRAINING), 'Material Design for Developers' (TRAINING), and 'Download Android Studio and SDK Tools' (STUDIO). Each card includes a small image, a title, a brief description, and a 'View Details' button.

Build Beautiful Apps

Resources to get you started with designing and developing for Android.

TRAINING

**Building Apps for Wearables**

Learn how to build notifications, send and sync data, and use voice actions.

TRAINING

**Material Design for Developers**

Learn how to apply material design to your apps.

STUDIO

**Download Android Studio and SDK Tools**

Get the official Android IDE and developer tools to build apps for Android.

# Documentation Structure

The image shows a screenshot of the official Android Developers website. At the top, there's a green header bar with the "Android Developers" logo and a search bar. Below the header, the main content area has a light gray background. It features a sidebar on the left with "HOME", "DESIGN", "DEVELOP", and "DISTRIBUTE" buttons. The "DEVELOP" button is currently active, indicated by a blue background. The main content area contains three large sections: "DESIGN - UX approach using ‘Material Design’", "DEVELOP - Software Developer Information, trainings, tutorials, sample code, reference", and "DISTRIBUTE - Delivering apps to users". Red arrows point from the descriptive text to their corresponding buttons in the sidebar.

Search

DESIGN Developers DESIGN DEVELOP DISTRIBUTE

HOME

DESIGN

DEVELOP

DISTRIBUTE

DESIGN - UX approach using “Material Design”

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

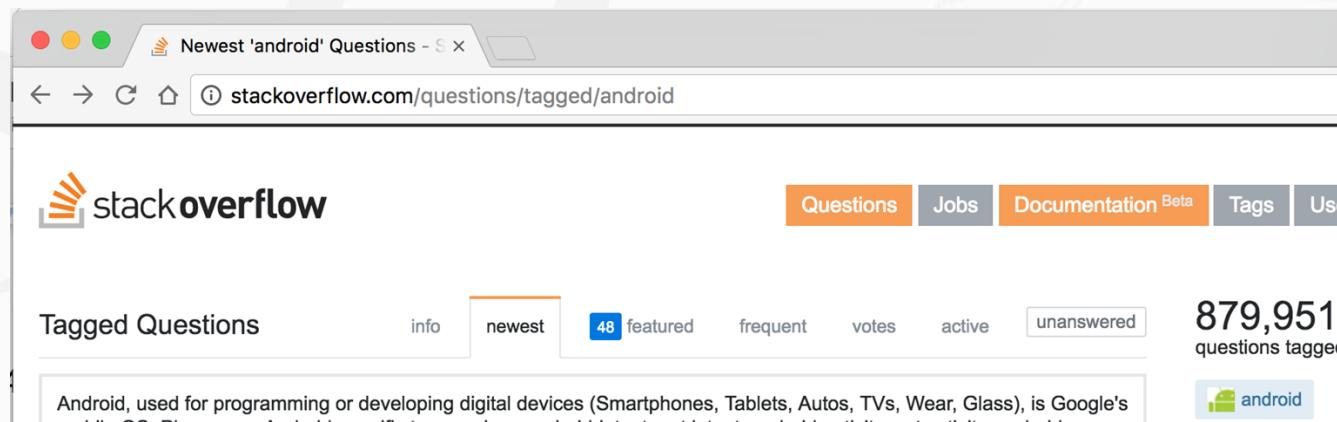
DISTRIBUTE - Delivering apps to users

# Stackoverflow.com

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

Question/Answer format

Android, Java language, other programming topics



# Official Android Blog

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

News, features, high-level



The Keyword

Latest Stories

Product News

Topics



# Android

News about Android



FOLLOW ANDROID

# 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

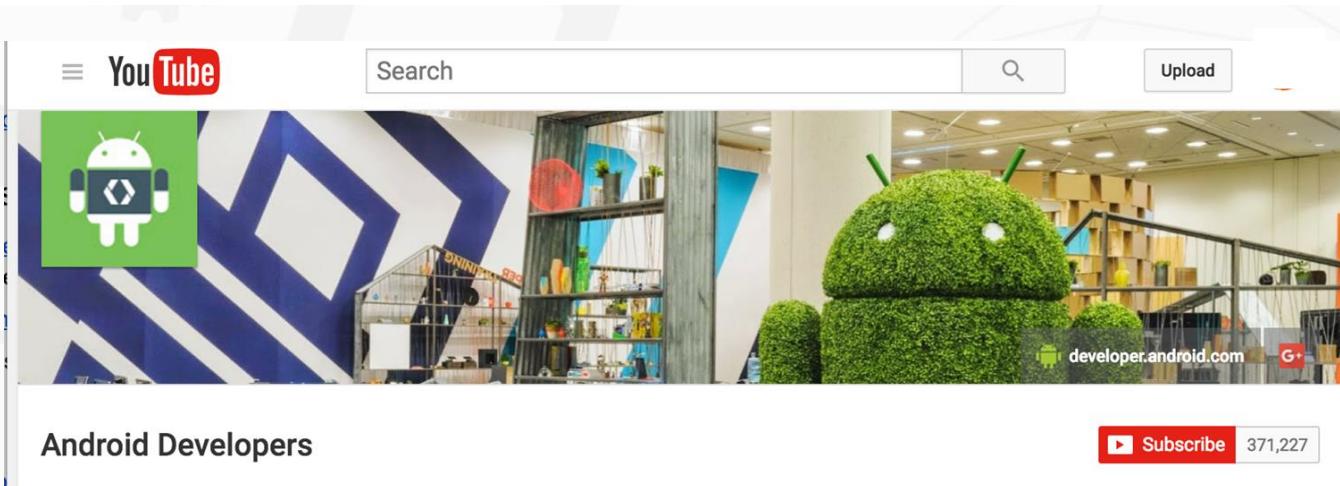


Android Developers Blog

# 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 website. 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!" heading is displayed, followed by a brief description of what codelabs are and their purpose. A "VIEW EVENTS" button is located on the right side of this section. Below the welcome message, there are three codelab cards, each with an Android icon and a duration indicator (38 min, 73 min, 21 min). The cards are titled: "Basic Android Accessibility : making sure everyone can use what you create!", "Agera: reactive Android apps", and "Echo with Android Howie Library". Each card also includes a "START" button and an update date.

Card	Title	Duration	Last Updated
1	Basic Android Accessibility : making sure everyone can use what you create!	38 min	Updated 5/18/2016
2	Agera: reactive Android apps	73 min	Updated 8/2/2016
3	Echo with Android Howie Library	21 min	Updated 12/4/2015

# 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)

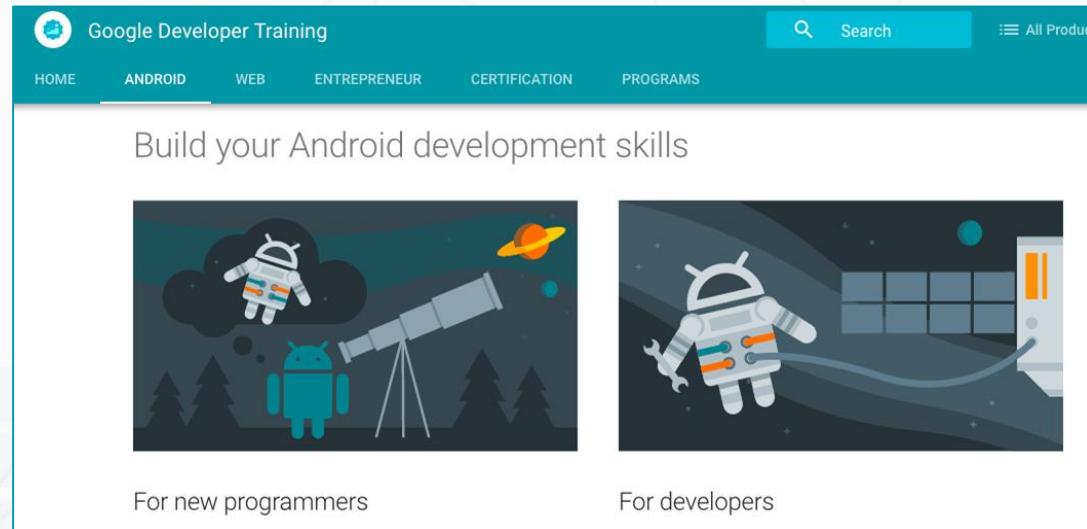
The screenshot shows a mobile-optimized vocabulary tool. On the left, a blue sidebar contains the title "Vocabulary Glossary" and a description: "This glossary of Android and Java vocab words supplements the [Udacity Android for Beginners](#) course. This course is targeted at those who are new to programming but want to start building Android apps." Below the text is a white Android robot icon. On the right, a white main area features a search bar with a magnifying glass icon at the top. A vertical list of vocabulary terms is displayed on the right side of the main area.

- 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)

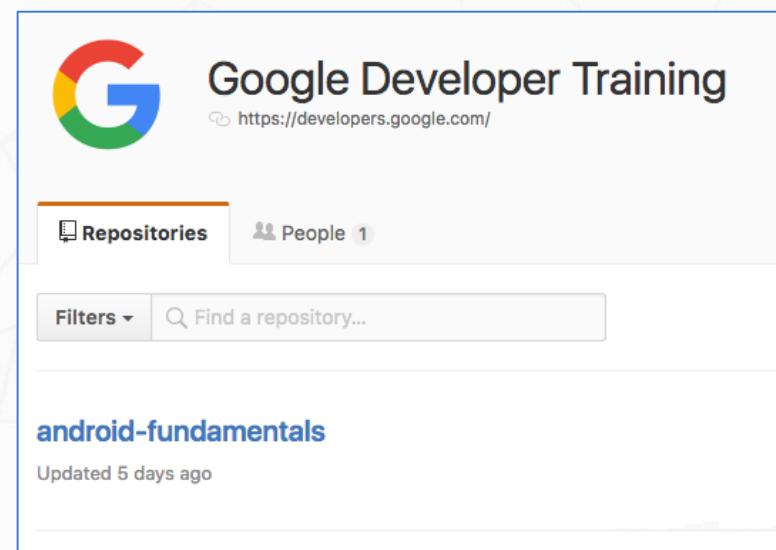
- Course info
- Programs
- Certification details
- Authorized Training Partners



# 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 "google-developer-training / android-fundamentals".

- Step 1:** The repository name "google-developer-training / android-fundamentals" is highlighted with a red box and a red circle containing the number 1.
- Step 2:** A red circle containing the number 2 points to the "Clone or download" button, which is highlighted with a green box.
- Step 3:** A red circle containing the number 3 points to the "Download ZIP" button, which is highlighted with a red box.

The repository details shown include:

- No description or website provided. — Edit
- 4 commits
- 1 branch
- 0 releases
- 3 contributors

The "Clone with HTTPS" section displays the URL: <https://github.com/google-developer-training/android-fundamentals>.

Recent commits listed:

- aleksinthecloud Squashed commit of the following: ...
- AlertSample Initial commit.
- DateTimePickers Initial commit.
- HelloSharedPrefs Initial commit.
- HelloToast Squashed commit of the following:

# 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](#)
- [Image Asset Studio](#)
- [Android Monitor page](#)
- [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 C Resources to Help You Learn](#)
- Practical: [1.4 P Learning About Available Resources](#)

**END**



Android  
Developer  
Associate



# 2.1 Activities

# Contents

- Activities
  - Defining an activity
  - Starting a new activity with an intent
  - Passing data between activities with extras
  - Navigating between activities

# Activities (high-level view)

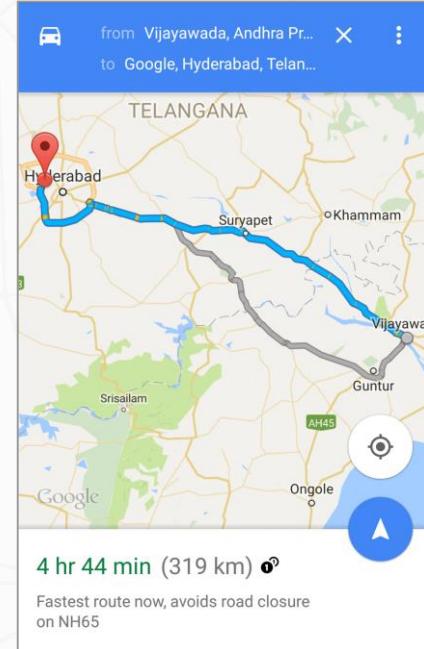
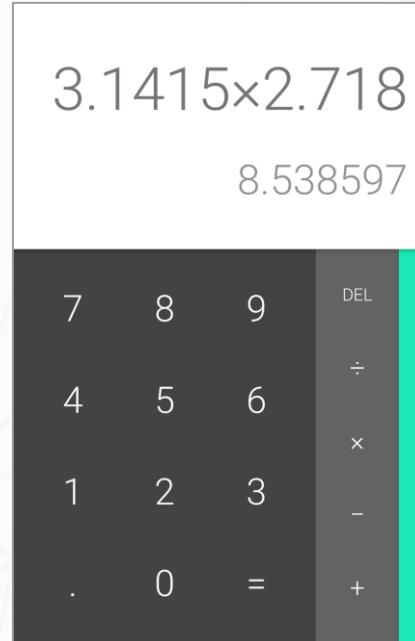
# What is an Activity?

- An **Activity** is an application component
- Represents one window, one hierarchy of views
- Typically fills the screen, but can be embedded in other activity or appear as floating window
- Java class, typically one activity in one file

# What does an Activity do?

- Represents an activity, such as ordering groceries, sending email, or getting directions
- Handles user interactions, such as button clicks, text entry, or login verification
- Can start other activities in the same or other apps
- Has a life cycle—is created, started, runs, is paused, resumed, stopped, and destroyed

# Examples of activities



# Apps and activities

- Activities are loosely tied together to make up an app
- First activity user sees is typically called "main activity"
- Activities can be organized in parent-child relationships in the Android manifest to aid navigation

# Layouts and Activities

- An activity typically has a UI layout
- Layout is usually defined in one or more XML files
- Activity "inflates" layout as part of being created

# Implementing Activities

# Implement new activities

1. Define layout in XML
2. Define Activity Java class
  - extends AppCompatActivity
3. Connect Activity with Layout
  - Set content view in onCreate()
4. Declare Activity in the Android manifest

# 1. Define layout in XML

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Let's Shop for Food!" />
</RelativeLayout>
```

## 2. Define Activity Java class

```
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
    }  
}
```

# 3. Connect activity with layout

```
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
}
```

Resource is layout in this XML file

# 4. Declare activity in Android manifest

```
<activity android:name=".MainActivity">
```

# 4. Declare main activity in manifest

Main Activity needs to include intent to start from launcher icon

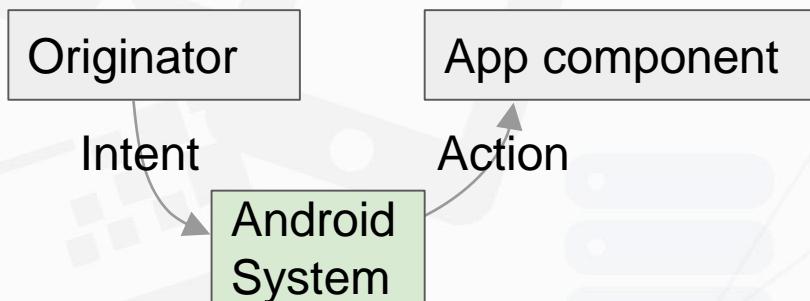
```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

# Intents

# What is an intent?

An intent is a description of an operation to be performed.

An Intent is an object used to request an action from another app component via the Android system.



# What can intents do?

- Start activities
  - A button click starts a new activity for text entry
  - Clicking Share opens an app that allows you to post a photo
- Start services
  - Initiate downloading a file in the background
- Deliver broadcasts
  - The system informs everybody that the phone is now charging

# Explicit and implicit intents

## Explicit Intent

- Starts a specific activity
  - Request tea with milk delivered by Nikita
  - Main activity starts the ViewShoppingCart activity

## Implicit Intent

- Asks system to find an activity that can handle this request
  - Find an open store that sells green tea
  - Clicking Share opens a chooser with a list of apps

# Starting Activities

# Start an Activity with an explicit intent

To start a specific activity, use an explicit intent

1.Create an intent

- `Intent intent = new Intent(this, ActivityName.class);`

2.Use the intent to start the activity

- `startActivity(intent);`

# Start an Activity with implicit intent

To ask Android to find an Activity to handle your request, use an implicit intent

1. Create an intent

- `Intent intent = new Intent(action, uri);`

2. Use the intent to start the activity

- `startActivity(intent);`

# Implicit Intents - Examples

## Show a web page

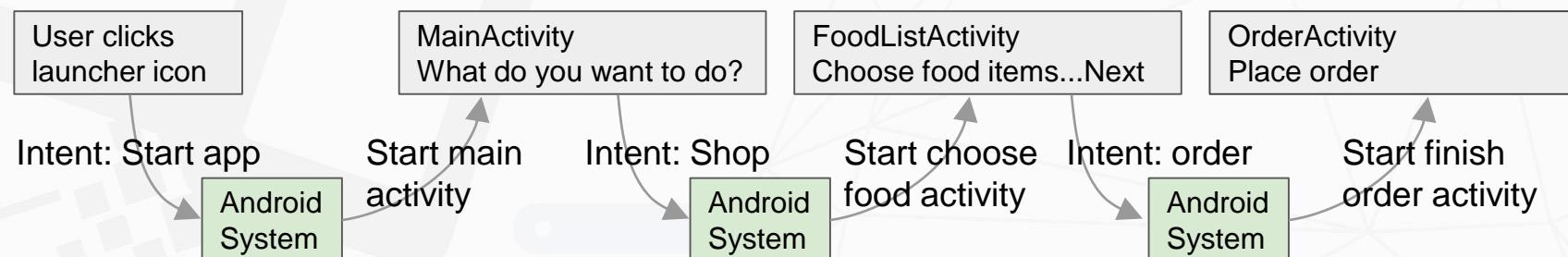
```
Uri uri = Uri.parse("http://www.google.com");  
Intent it = new Intent(Intent.ACTION_VIEW,uri);  
startActivity(it);
```

## Dial a phone number

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

# How Activities Run

- All activities are managed by the Android runtime
- Started by an "intent", a message to the Android runtime to run an activity



# Sending and Receiving Data

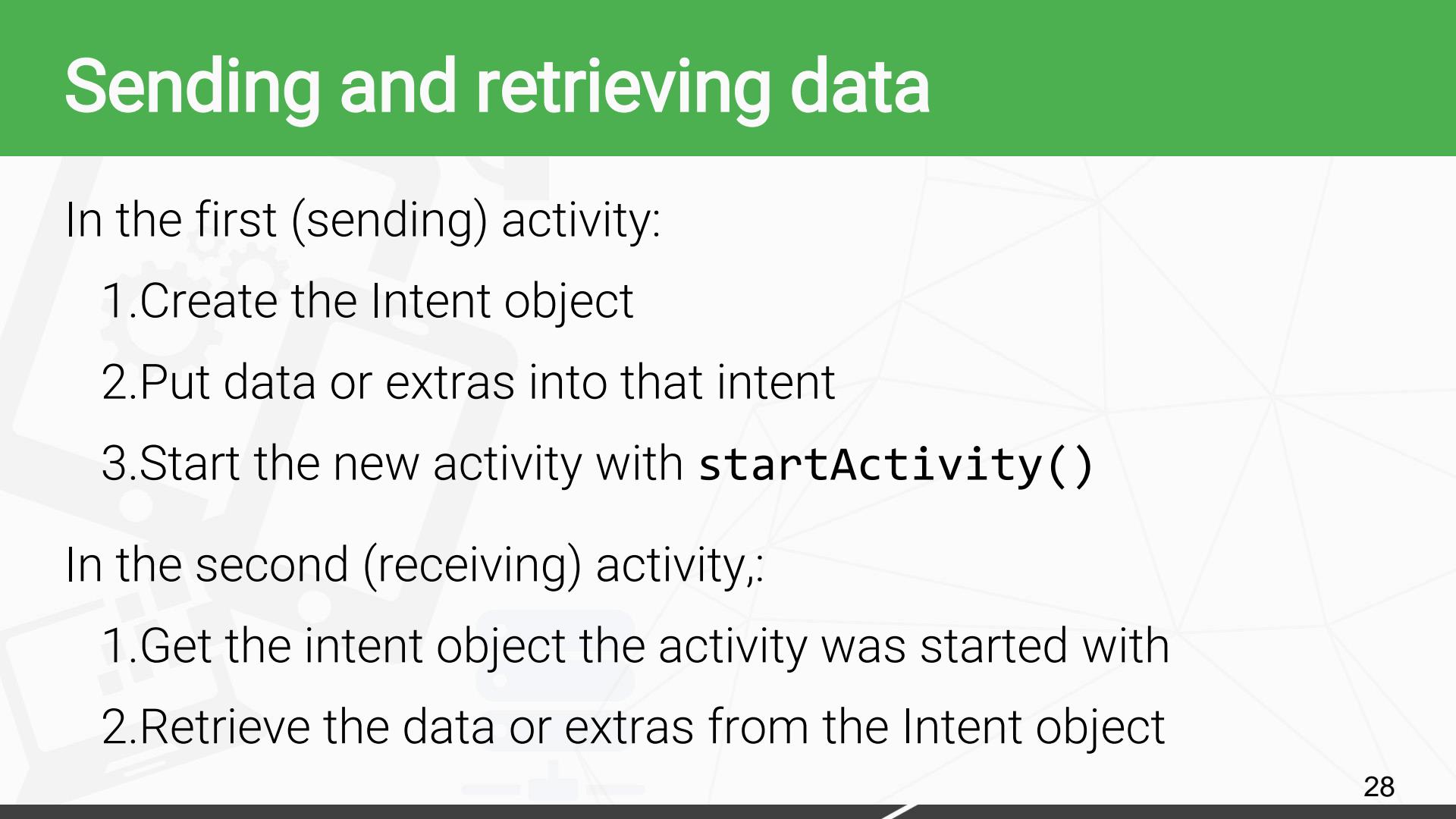
# Two types of sending data with intents

- Data—one piece of information whose data location can be represented by an URI
- Extras—one or more pieces of information as a collection of key-value pairs in a [Bundle](#)

# Sending and retrieving data

In the first (sending) activity:

- 1.Create the Intent object
- 2.Put data or extras into that intent
- 3.Start the new activity with **startActivity()**

In the second (receiving) activity,:  


- 1.Get the intent object the activity was started with
- 2.Retrieve the data or extras from the Intent object

# Putting a URI as intent data

```
// A web page URL  
intent.setData(  
    Uri.parse("http://www.google.com"));  
  
// a Sample file URI  
intent.setData(  
    Uri.fromFile(new File("/sdcard/sample.jpg")));
```

# Put information into intent extras

- `putExtra(String name, int value)`  
⇒ `intent.putExtra("level", 406);`
- `putExtra(String name, String[] value)`  
⇒ `String[] foodList = {"Rice", "Beans", "Fruit"};`  
`intent.putExtra("food", foodList);`
- `putExtras(bundle);`  
⇒ if lots of data, first create a bundle and pass the bundle.
- See [documentation](#) for all

# Sending data to an activity with extras

```
public static final String EXTRA_MESSAGE_KEY =  
    "com.example.android.twoactivities.extra.MESSAGE";  
  
Intent intent = new Intent(this, SecondActivity.class);  
String message = "Hello Activity!";  
intent.putExtra(EXTRA_MESSAGE_KEY, message);  
startActivity(intent);
```

# Get data from intents

- `getData();`  
    ⇒ `Uri locationUri = intent.getData();`
- `int getIntExtra (String name, int defaultValue)`  
    ⇒ `int level = intent.getIntExtra("level", 0);`
- `Bundle bundle = intent.getExtras();`  
    ⇒ Get all the data at once as a bundle.
- See [documentation](#) for all

# Returning data to the starting activity

1. Use `startActivityForResult()` to start the second activity

2. To return data from the second Activity:

- Create a **new** Intent
- Put the response data in the Intent using `putExtra()`
- Set the result to `Activity.RESULT_OK`  
or `RESULT_CANCELED`, if the user cancelled out
- call `finish()` to close the activity

1. Implement `onActivityResult()` in first activity

# startActivityForResult()

`startActivityForResult`(`intent`, `requestCode`);

- Starts activity (`intent`), assigns it identifier (`requestCode`)
- Returns data via intent extras
- When done, pop stack, return to previous activity, and execute `onActivityResult()` callback to process returned data
- Use `requestCode` to identify which activity has "returned"

# 1.startActivityForResult() Example

```
public static final int CHOOSE_FOOD_REQUEST = 1;
```

```
Intent intent = new Intent(this, ChooseFoodItemsActivity.class);  
startActivityForResult(intent, CHOOSE_FOOD_REQUEST);
```

## 2.Return data and finish second activity

```
// Create an intent  
Intent replyIntent = new Intent();  
  
// Put the data to return into the extra  
replyIntent.putExtra(EXTRA_REPLY, reply);  
  
// Set the activity's result to RESULT_OK  
setResult(RESULT_OK, replyIntent);  
  
// Finish the current activity  
finish();
```

### 3. Implement onActivityResult()

```
public void onActivityResult(int requestCode,  
                           int resultCode, Intent data) {  
    super.onActivityResult(requestCode, resultCode, data);  
    if (requestCode == TEXT_REQUEST) { // Identify activity  
        if (resultCode == RESULT_OK) { // Activity succeeded  
            String reply = data.getStringExtra(SecondActivity.EXTRA_REPLY);  
            // ... do something with the data  
        }}}
```

# Navigation

# Activity stack

- When a new activity is started, the previous activity is stopped and pushed on the activity back stack
- Last-in-first-out-stack—when the current activity ends, or the user presses the Back  button, it is popped from the stack and the previous activity resumes

# Activity Stack

After viewing shopping cart, user decides to add more items, then places order.



# Two forms of navigation

-  Temporal or back navigation
  - provided by the device's back button
  - controlled by the Android system's back stack
  
-  Ancestral or up navigation
  - provided by the app's action bar
  - controlled by defining parent-child relationships between activities in the Android manifest



# Back navigation

- Back stack preserves history of recently viewed screens
- Back stack contains all the activities that have been launched by the user in reverse order *for the current task*
- Each task has its own back stack
- Switching between tasks activates that task's back stack
- Launching an activity from the home screen starts a new task
- Navigate between tasks with the overview or recent tasks screen



# Up navigation

- Goes to parent of current activity
- Define an activity's parent in Android manifest
- Set parentActivityName

```
<activity  
    android:name=".ShowDinnerActivity"  
    android:parentActivityName=".MainActivity" >  
</activity>
```

# Learn more

# Learn more

- [Android Application Fundamentals](#)
- [Starting Another Activity](#)
- [Activity \(API Guide\)](#)
- [Activity \(API Reference\)](#)
- [Intents and Intent Filters \(API Guide\)](#)
- [Intent \(API Reference\)](#)
- [Navigation](#)

# What's Next?

- Concept Chapter: [2.1 C Understanding Activities and Intents](#)
- Practical: [2.1 P Create and Start Activities](#)

**END**



Android  
Developer  
Associate

## Lesson 2



# 2.2 Activity Lifecycle and Managing State

# Contents

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

# Activity Lifecycle

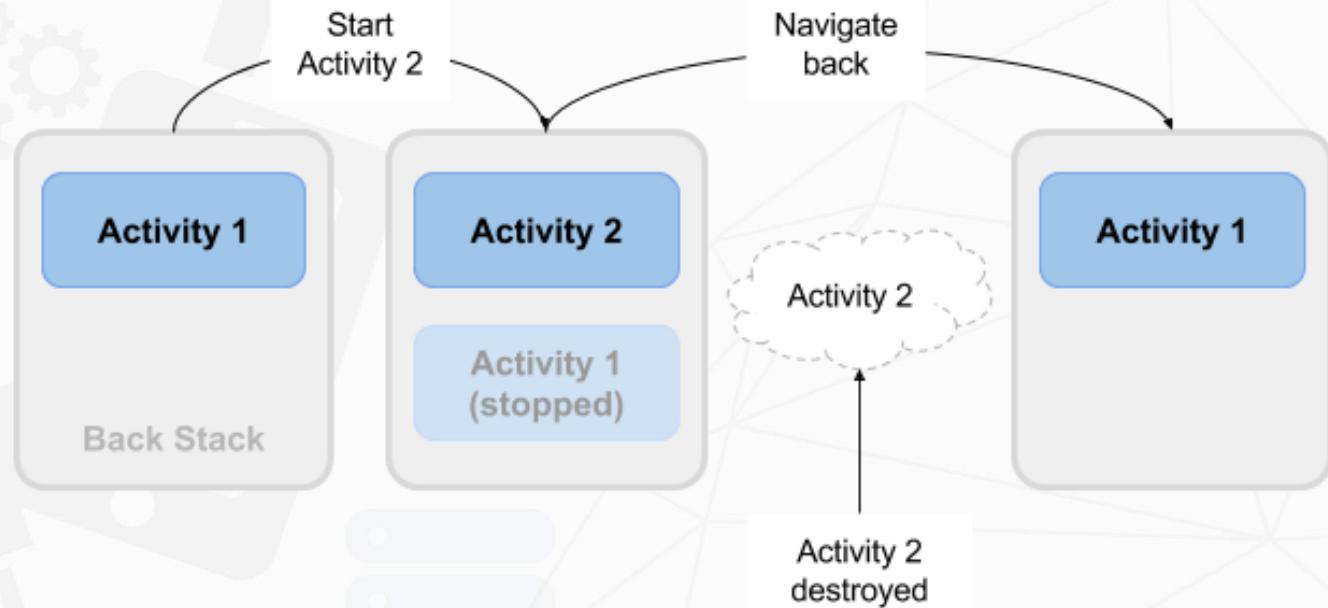
# What is the Activity Lifecycle?

- 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

# 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())

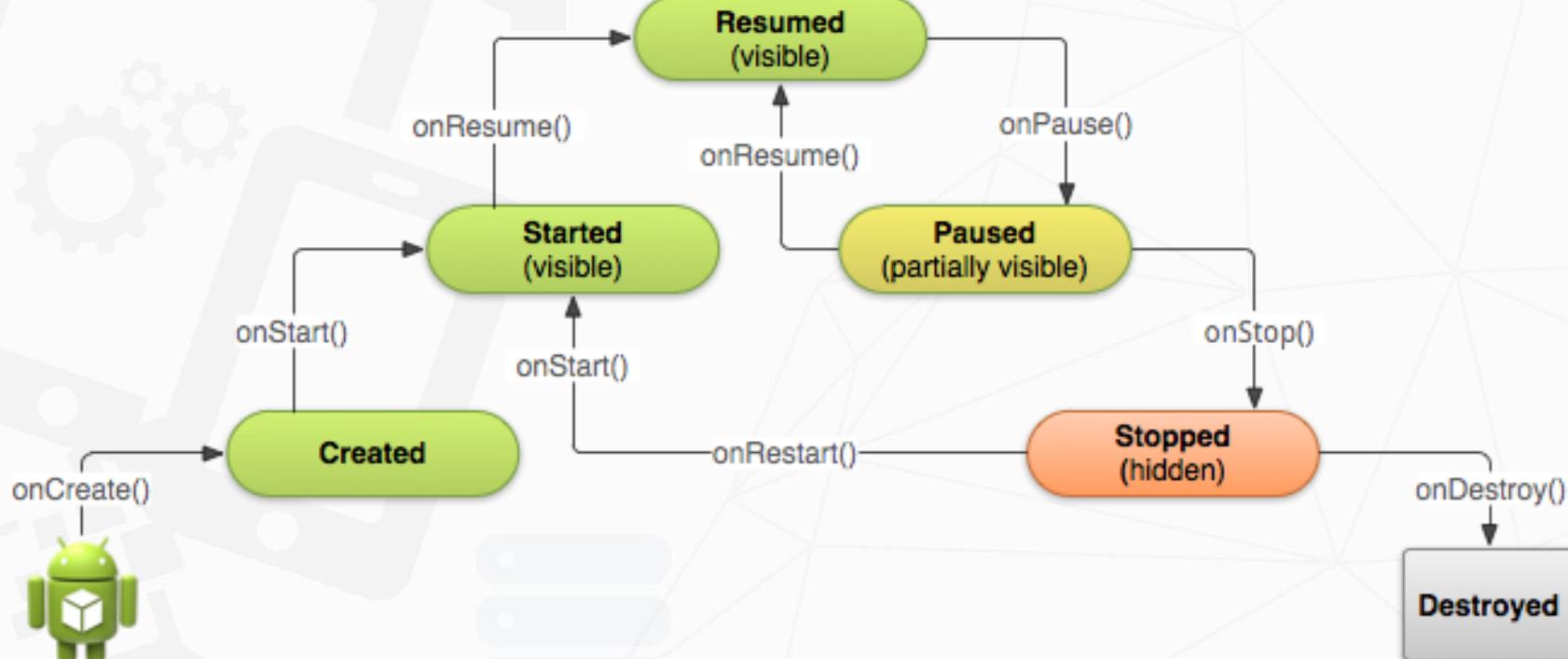
| | | **onResume()**—start to interact with user

| | | **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 this activity is coming back to interact with the user, or onDestroy() if this 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 (Android 7)

# What happens on config change?

On configuration change, Android:

1. shuts down activity  
by calling:

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

2. then starts it over  
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

# Activity instance state

- System only saves:
  - 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 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 = (TextView) 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

- [Activity](#) (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 C Activity Lifecycle and Managing State](#)
- Practical: [2.2 P Activity Lifecycle and Instance State](#)

**END**



Android  
Developer  
Associate

## Lesson 2



## 2.3 Starting Activities with Implicit Intents

# Contents

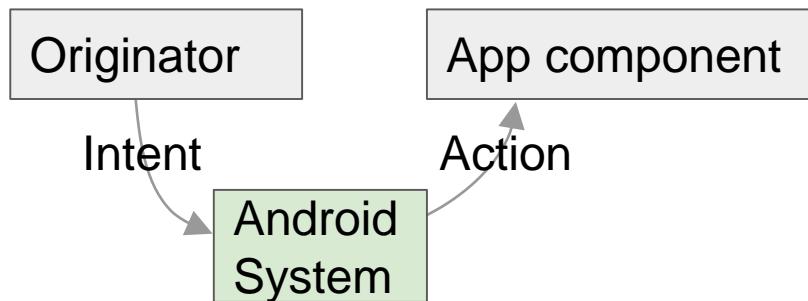
- Intents—recap
- Implicit intents
- Sending implicit intents
- Receiving implicit intents

# Recap: Intents

# What is an intent?

An intent is a description of an operation to be performed.

An Intent is a 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 later lessons

# Explicit and implicit intents

**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 Intents

# Implicit intents

- An implicit intent allows you to 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"
- An implicit intent specifies an action and may provide data with which to perform the action

# Implicit intents

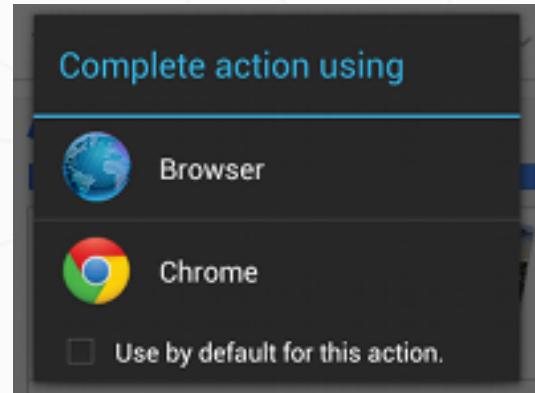
- Implicit intents do not specify the target activity class, just the intended action
- Android runtime matches the implicit intent request with registered intent handlers
- If there are multiple matches, an App Chooser will open to let the user decide

# How do implicit intents work?

- 1.The Android Runtime keeps a list of registered Apps
- 2.Apps have to register via the Android Manifest
- 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 let 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 Implicit Intents

# 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 an activity that allows them to 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 Intents - 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 activities 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 implicit intents

Common actions for implicit intents 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 implicit intents](#)
- [List of all available actions](#)

# Receiving Implicit Intents

# Register your app to receive intents

- Declare one or more intent filters for the activity in the Android manifest
- Filter announces activity's ability to accept implicit intents
- Filter puts conditions on the intents that the activity accepts

# Intent filter in the Android Manifest

```
<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 all intents with [ACTION VIEW](#)
  - `android.intent.action.SEND` – matches all intents 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 intents 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 C Activities and Implicit Intents](#)
- Practical: [2.3 P Start Activities with Implicit Intents](#)

# END





Android  
Developer  
Associate

## Lesson 3



# 3.1 Debugging Apps

# 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

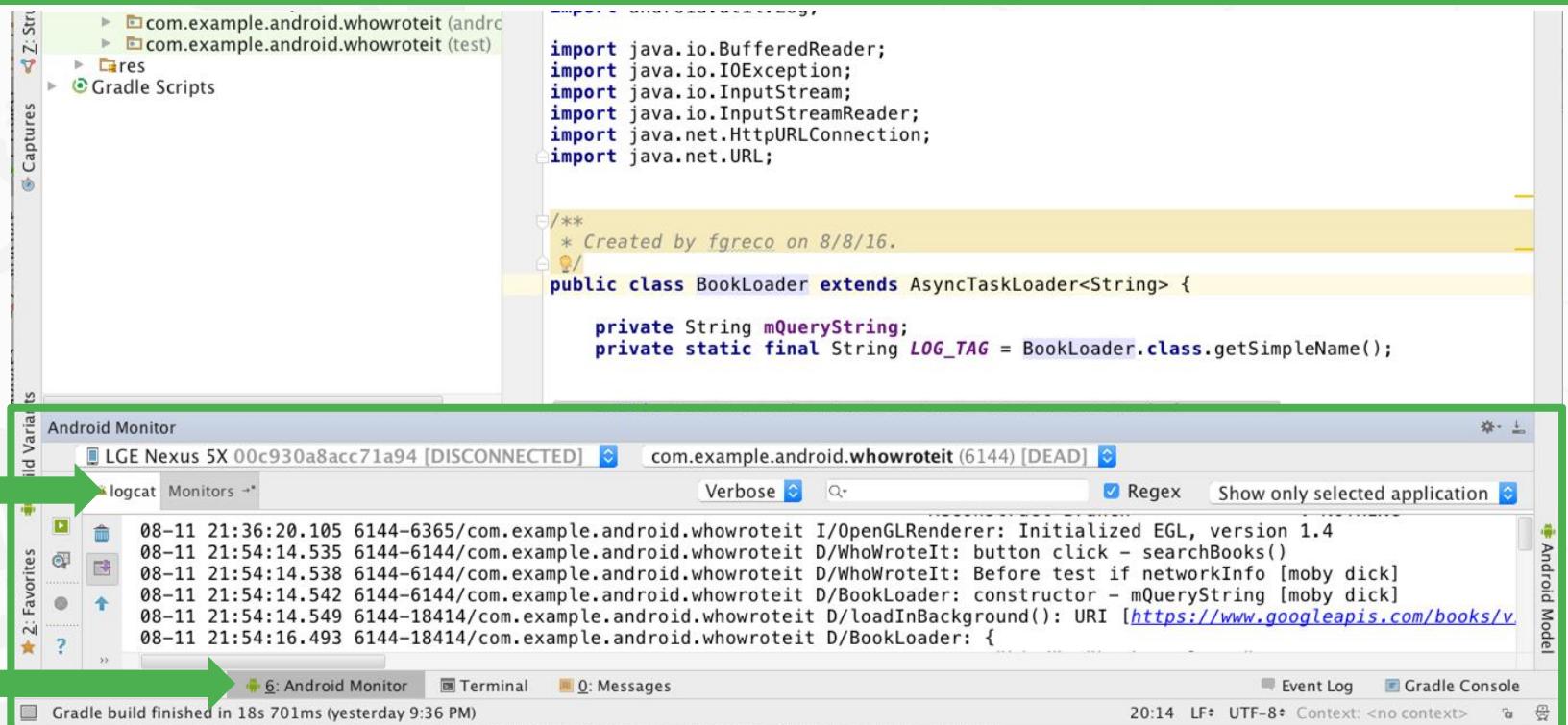
- identify problems
- find where in the source code the problem is created
- so that you can fix it

# 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 Android Monitor, logcat pane  
// Log.<log-level>(TAG, "Message");  
Log.d(TAG, "Hello World");
```

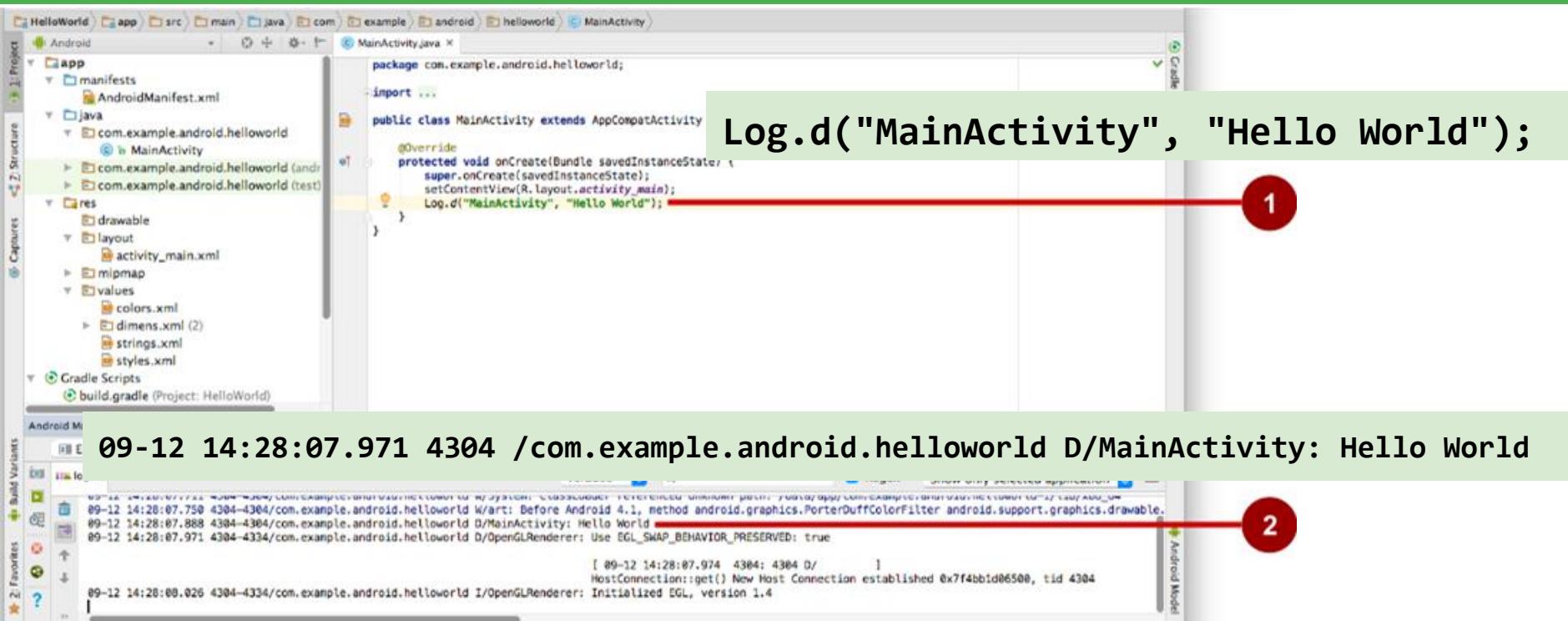
# Open Android Monitor and logcat



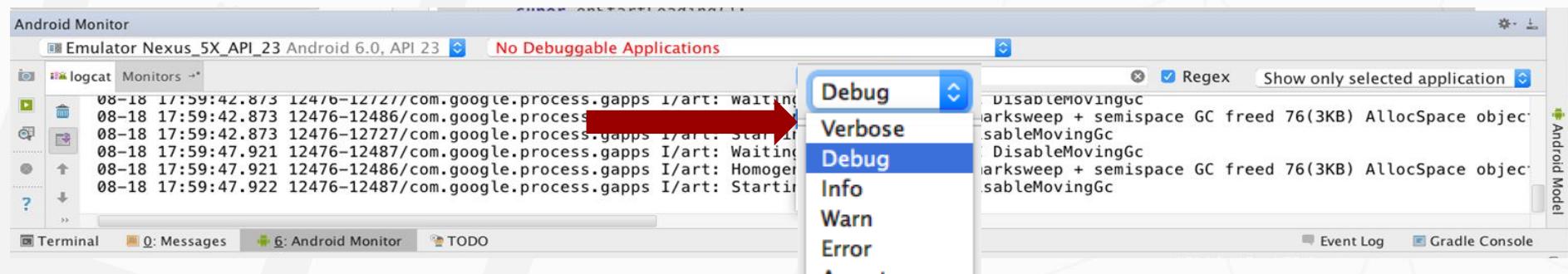
logcat  
pane

Android  
Monitor

# Inspect logging messages



# Choose visible logging level



Displays logs with levels at  
this level or higher

# Log Levels

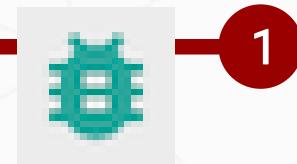
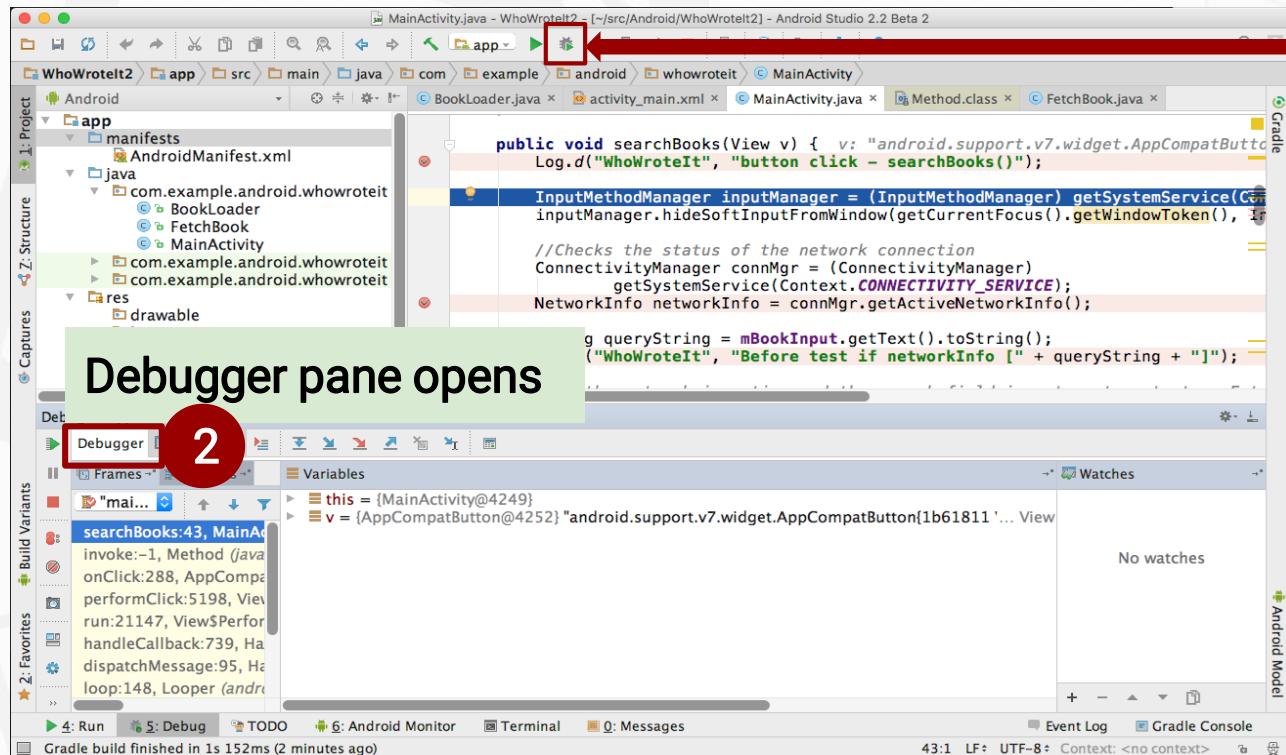
- **Verbose** - All verbose log statements and comprehensive system
- **Debug** - All debug logs, variable values, debugging notes
- **Info** - Status info, such as database connection
- **Warning** - Unexpected behavior, non-fatal issues
- **Error** - Serious error conditions, exceptions, crashes only

# 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

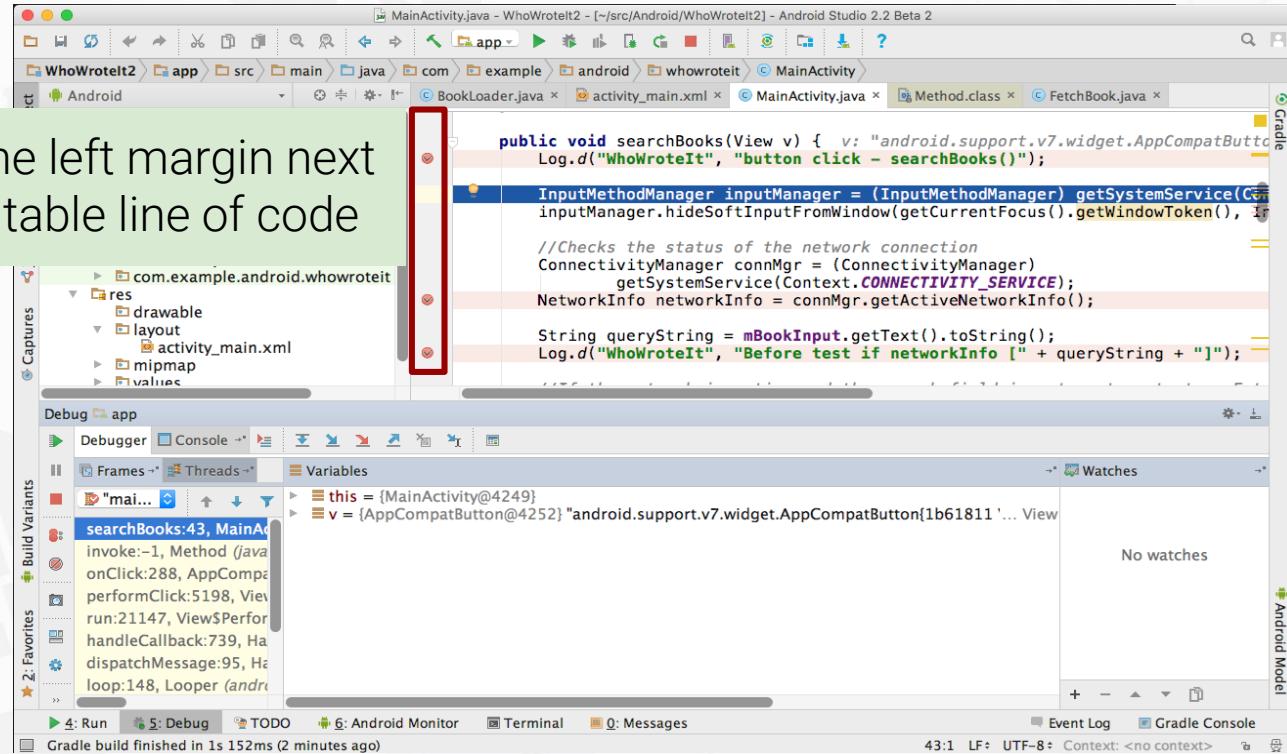
# Run in debug mode



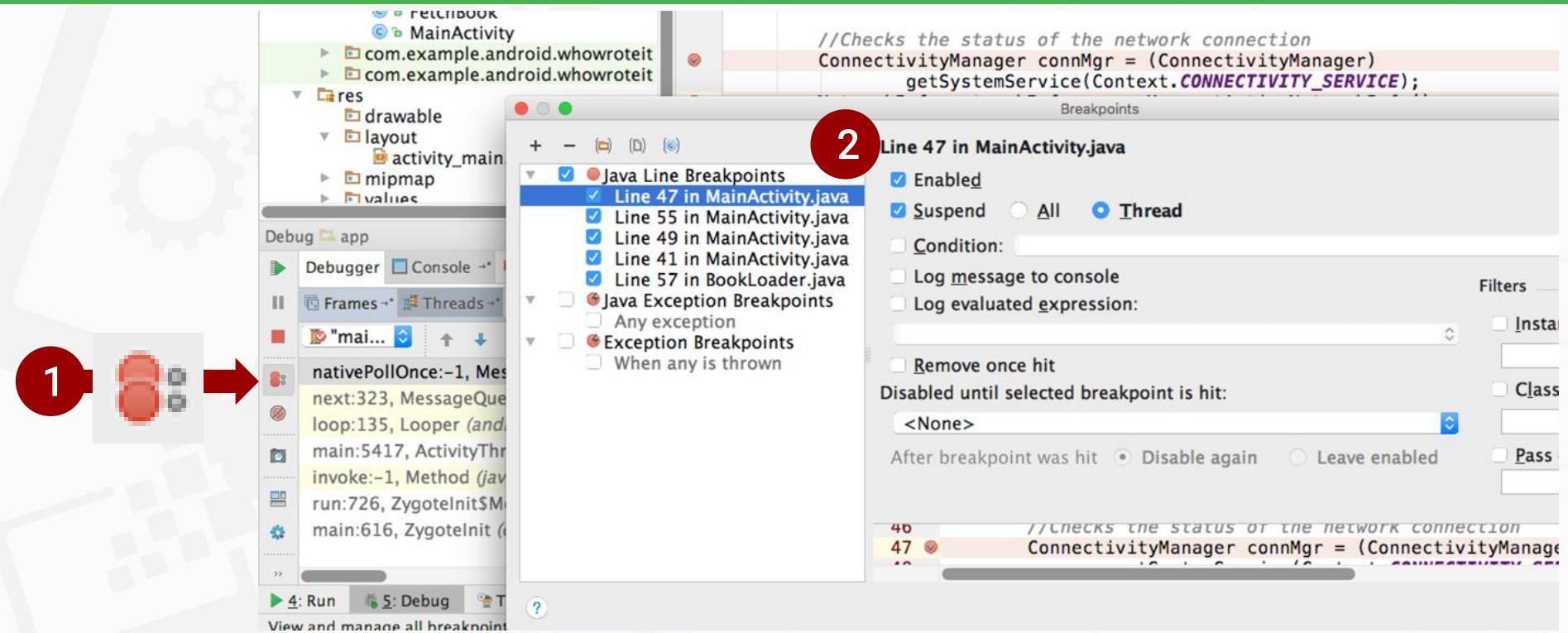
Menu:  
Run > Debug 'your app'

# Set breakpoints

Click in the left margin next to executable line of code

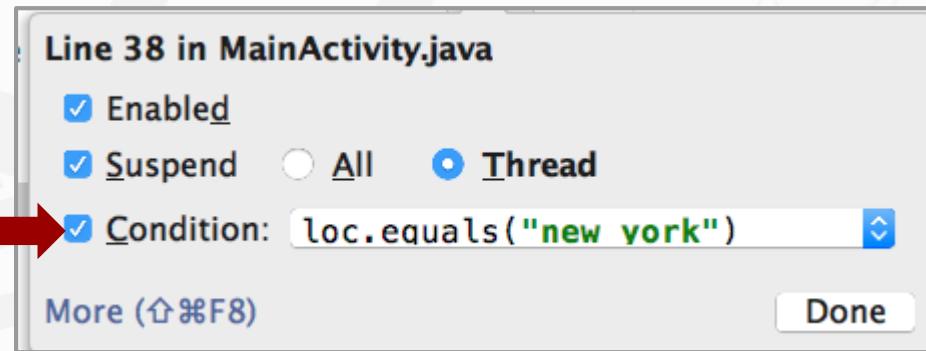


# Edit breakpoint properties

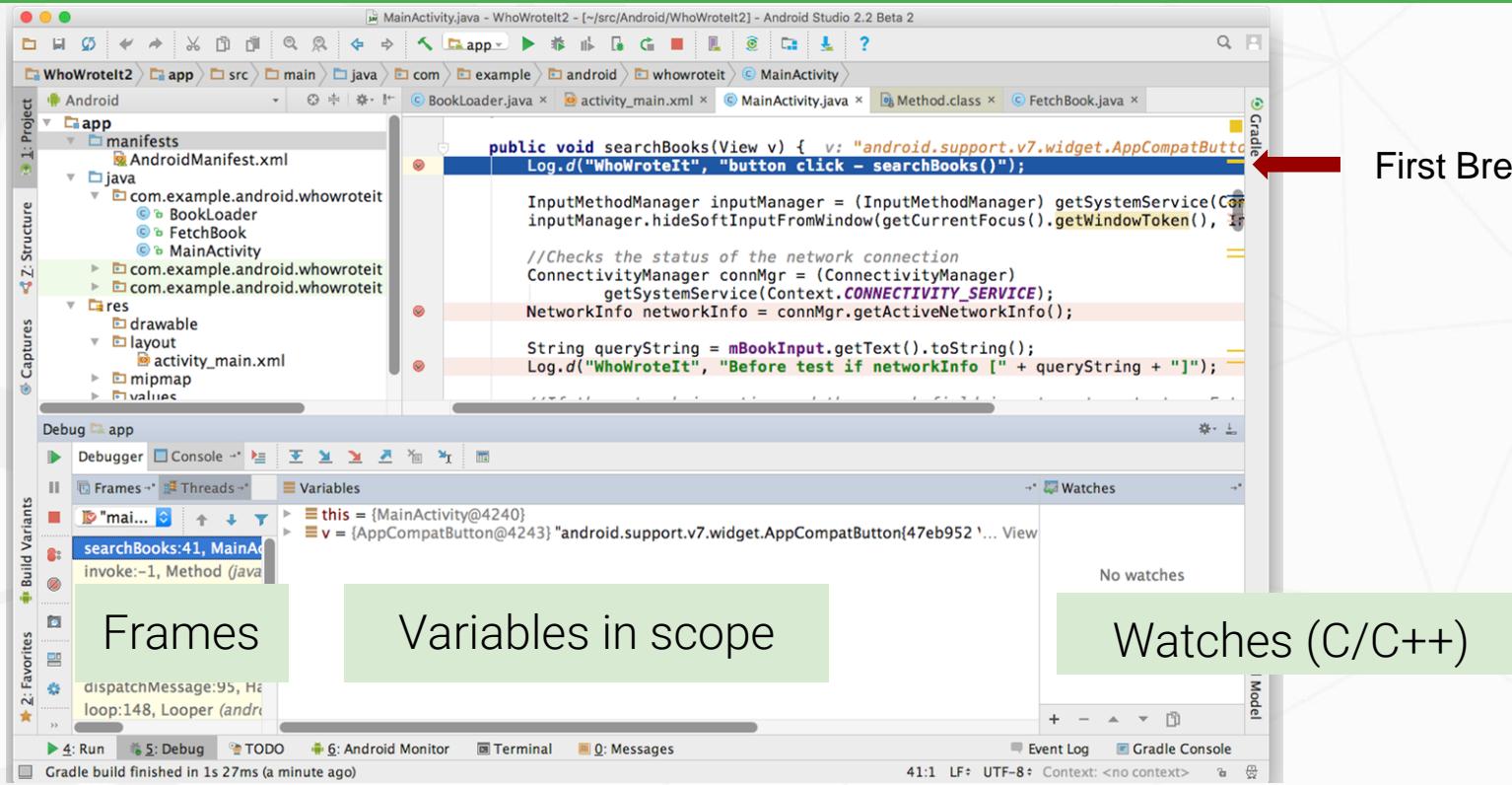


# Make breakpoints conditional

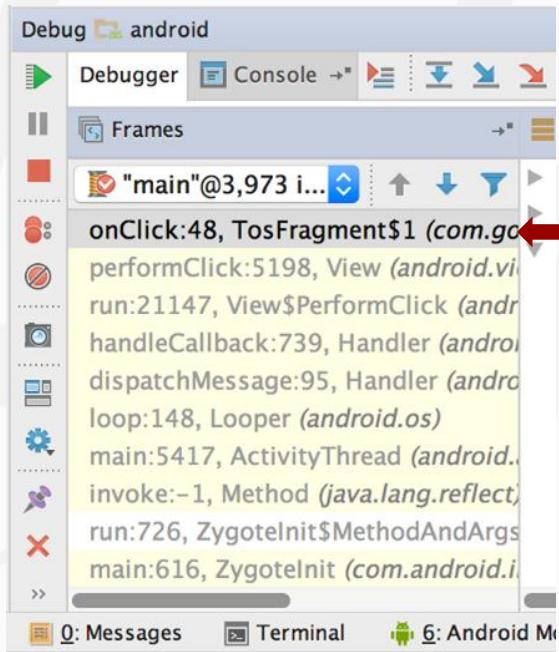
- In properties dialog or right -click existing breakpoint
- Any Java expression that returns a boolean
- Code completion helps you write conditions



# Run until app stops at breakpoint

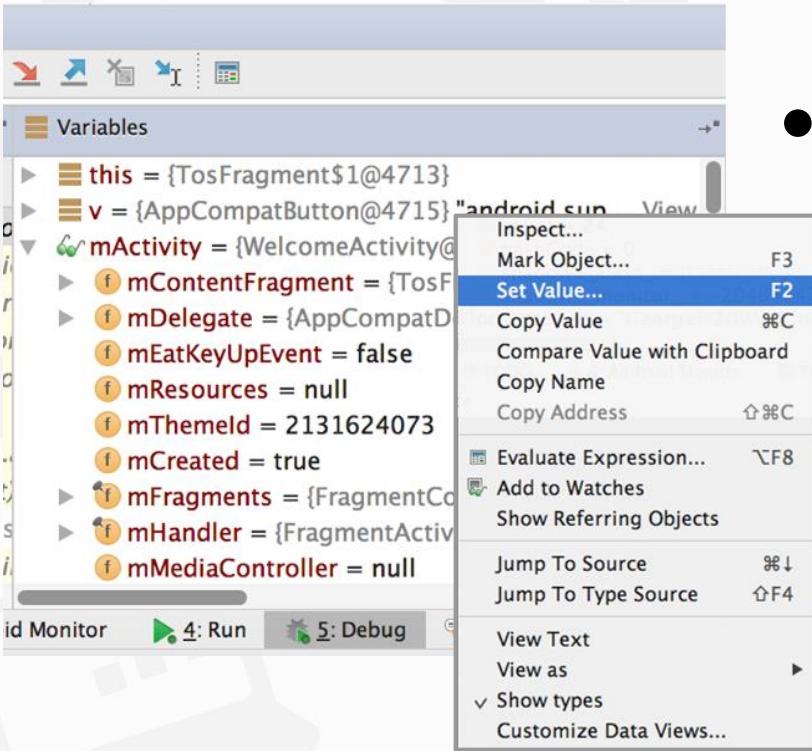


# Inspect frames



Top frame is where execution is halted in your code

# Inspect and edit variables

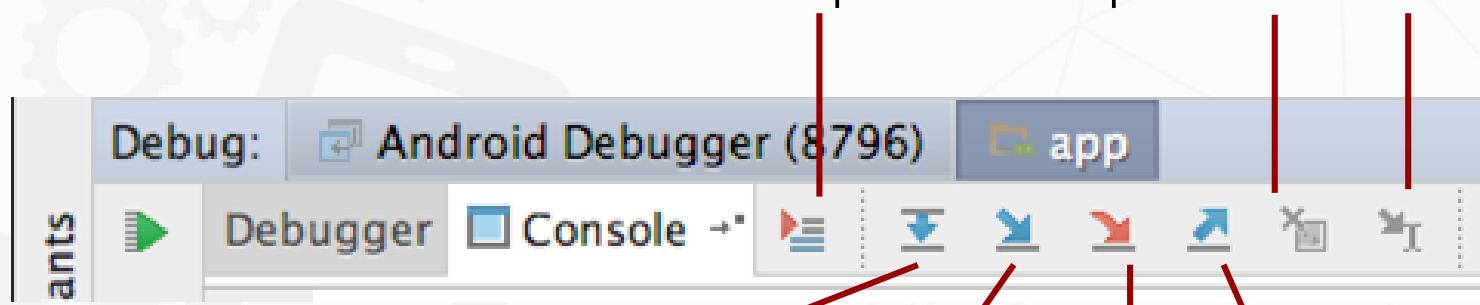


- Right-click on variable for menu

# 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



Show execution point Drop frame Run to cursor

Debug: **Android Debugger (8796)** app

ants



Debugger

Console



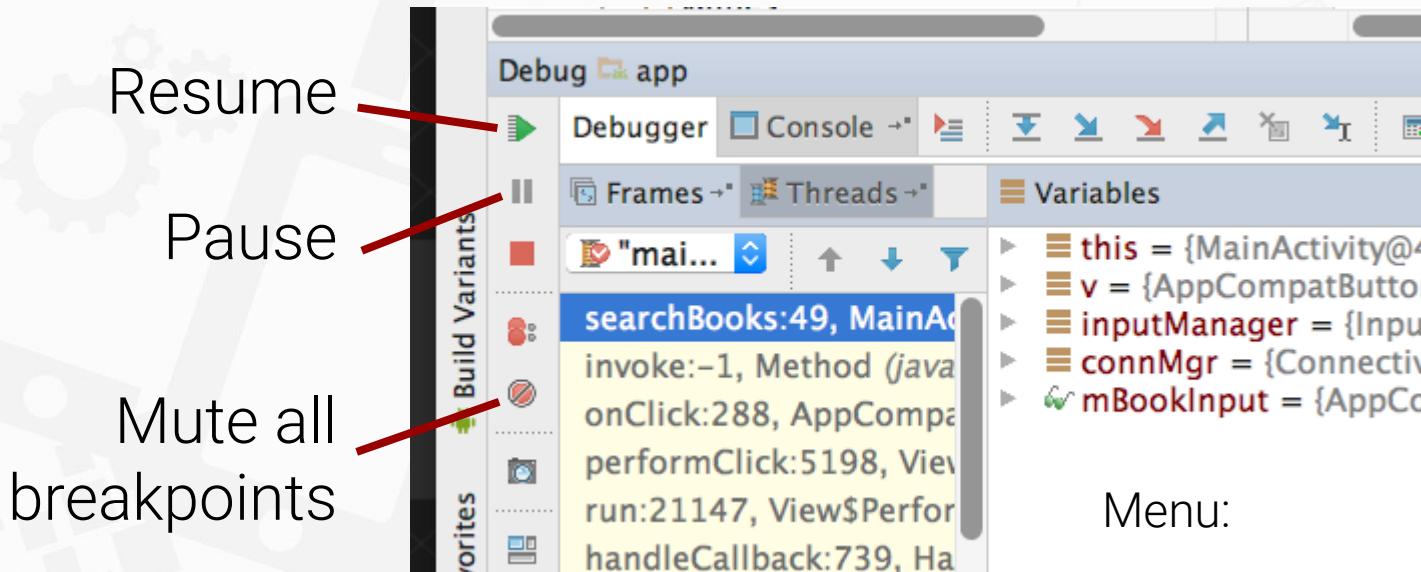
Step over

Step into

Step out

Force step into

# Resume and Pause



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 C The Android Studio Debugger](#)
- Practical: [3.1 P Using the Debugger](#)

**END**



Android  
Developer  
Associate

## Lesson 3

# 3.2 Testing Your App

# Contents

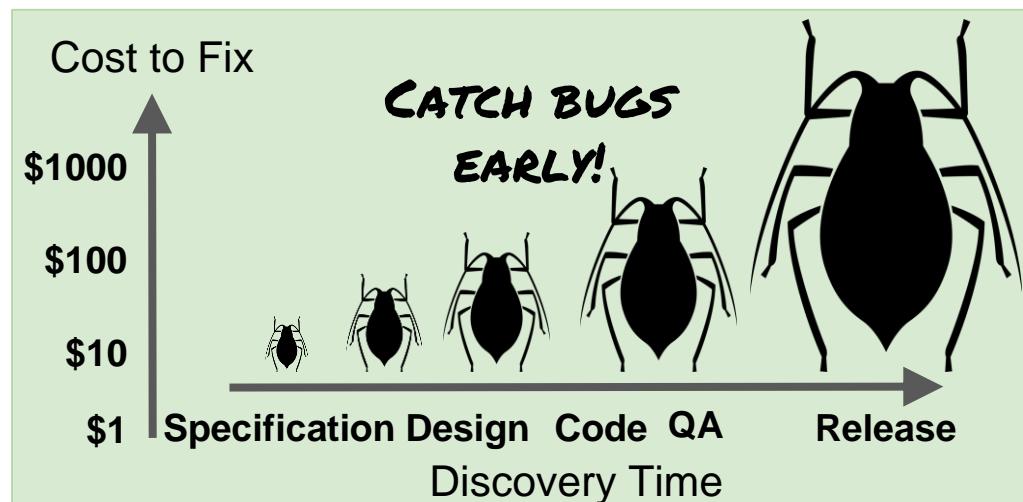
- Why testing is worth your time
- Unit testing

Note: User interface testing (instrumented testing) is covered in a later 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 later)

# 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

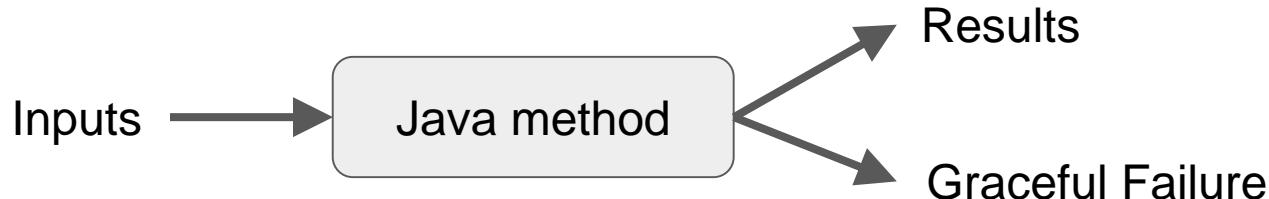
Android Studio creates three source sets for your project

- **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) that do not need access to the Android framework or an Android device or emulator, or those for which 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

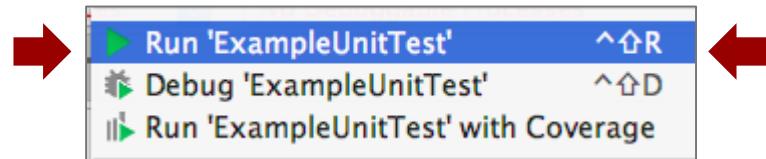
```
/**  
 * Release external resources  
 */  
@After  
public void tearDown() {  
    ....  
}
```

- Frees resources

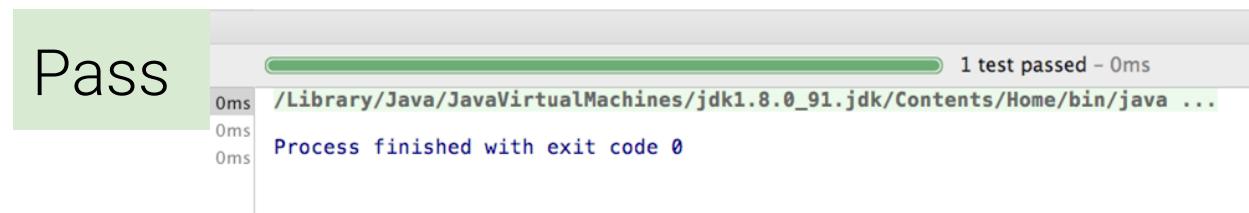
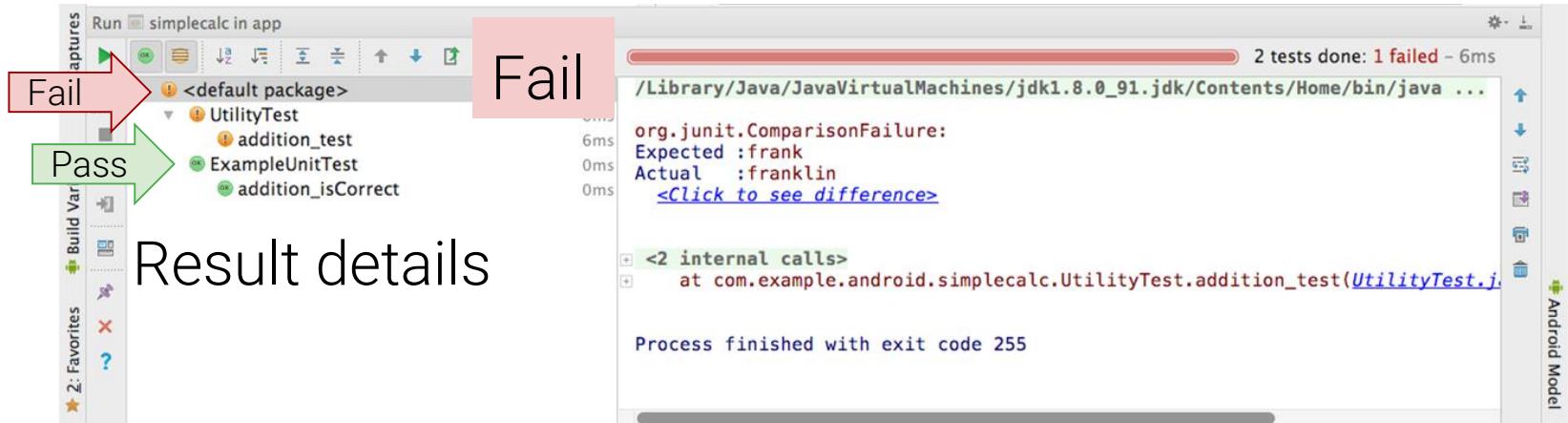
# Running Tests in Android Studio

# 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



# 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 test run interface. At the top, there is a code editor window displaying Java test code. In the middle, a terminal window shows the test results and the error stack trace. A red box highlights the failing assertion in the code and the detailed error message in the terminal.

```
    /**
     * To work on unit tests, switch the Test Artifact in the Build Variants view.
     */
    public class FloatingPointUnitTest {
        @Test
        public void addition_isCorrect() throws Exception {
            assertEquals(.3d, .1d+.2d, 0d); // 3rd arg is 'epsilon'
        }
    }
```

1 test failed - 10ms

```
ms /Library/Java/JavaVirtualMachines/jdk1.8.0_91.jdk/Contents/Home/bin/java ...
ms
java.lang.AssertionError:
Expected :0.3
Actual   :0.3000000000000004
<click to see difference>
```

```
+ <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. The FloatingPointUnitTest package is selected.
- File Explorer:** Shows a folder structure under res: layout and mipmap.
- Code Editor:** Displays the FloatingPointUnitTest.java file with the following code:

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

A green box highlights the float value `.0005d`.
- Run Tab:** Shows the test is running: "Running FloatingPointUnitTest" and "1 test passed - 0ms".
- Logcat Tab:** Shows the log output:

```
FloatingPointUnitTest (com.example.android.si) 0ms  
addition_isCorrect 0ms  
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 C Testing Your App](#)
- Practical: [3.2 P Testing Apps with Unit Tests](#)

**END**



Android  
Developer  
Associate

## Lesson 3



# 3.3 The Android Support Libraries

# 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-v7:24.2.1

# 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 Android Support Libraries

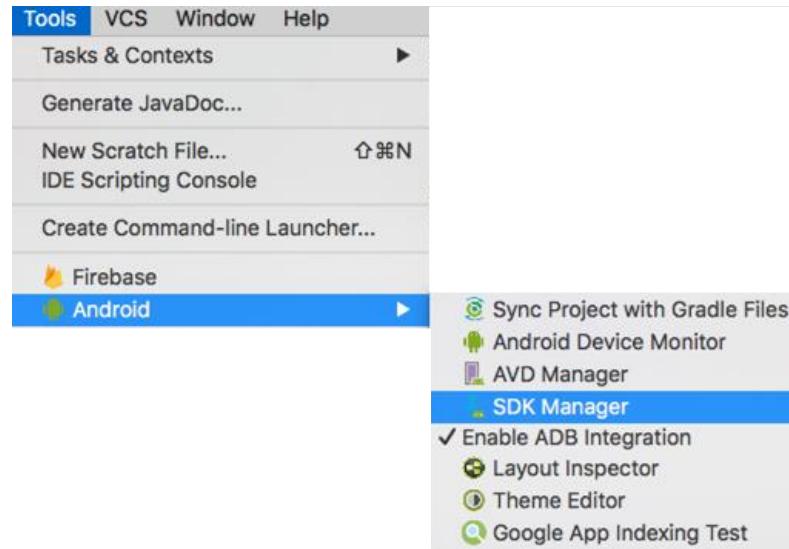
# Support libraries

- 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

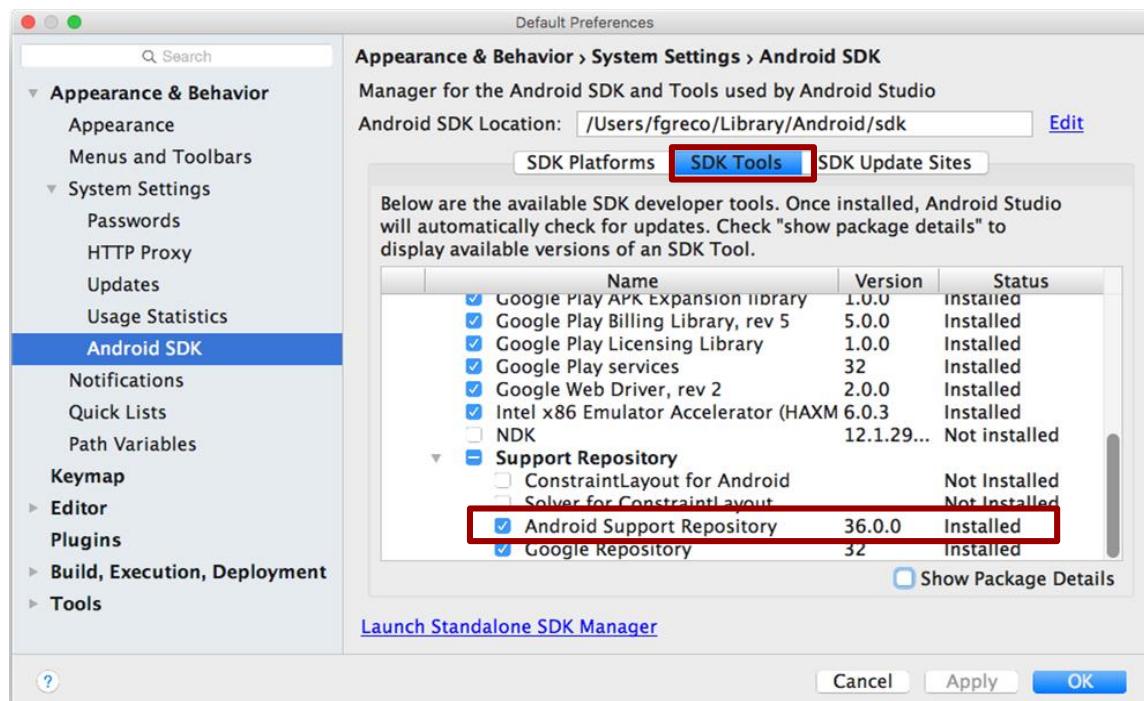


# Find the Android Support Library

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
  - For example, the recommended [v7 appcompat](#) library
3. Copy the build script dependency identifier for the library
  - [com.android.support:appcompat-v7:24.2.1](#)

# Add dependency to build.gradle

- 1.open build.gradle (Module: app)
- 2.In the dependencies section, add a dependency for the support library
  - o compile 'com.android.support:appcompat-v7:24.2.0'
- 3.Update the version number, if prompted
- 4.Sync Now** when prompted

# Updated build.gradle (Module: app)

```
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    testCompile 'junit:junit:4.12'  
    compile 'com.android.support:appcompat-v7:24.2.0'  
}
```

# 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 C The Android Support Library](#)
- Practical: [3.3 P Using the Android Support Libraries](#)

**END**



Android  
Developer  
Associate

## Lesson 4



# 4.1 User Input Controls

# Contents

- User Interaction
- Focus
- Text input and keyboards
- Radio Buttons and Checkboxes
- Making Choices
  - dialogs, spinners and pickers
- Recognizing gestures

# User Interaction

# Users expect to interact with apps

- Clicking, pressing, talking, typing, and listening
- Using user input controls such buttons, menus, keyboards, text boxes, and a microphone
- Navigating between activities

# 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

# Input Controls

# Ways to get input from the user

- Free form

- Text and voice input

- Actions

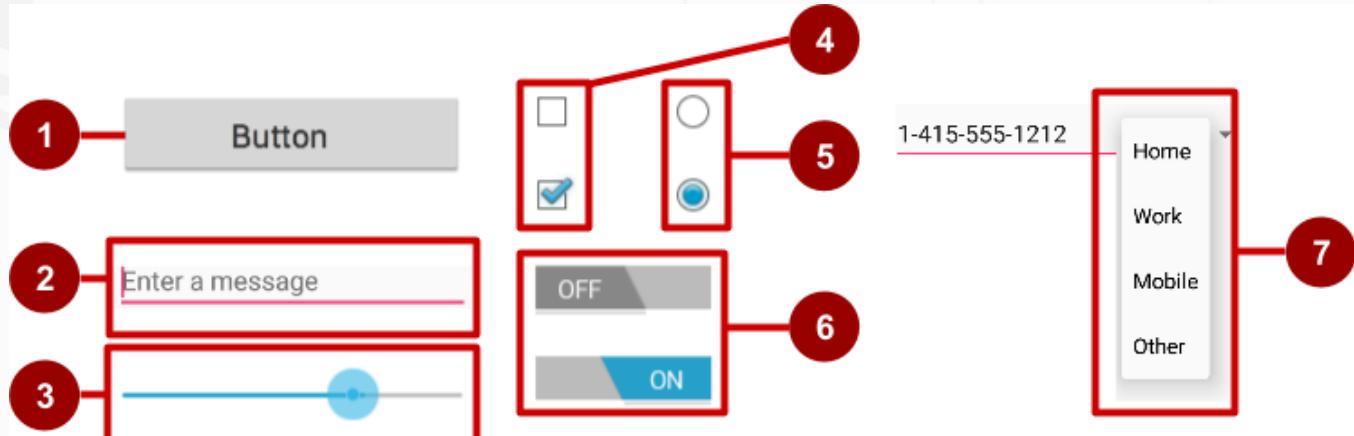
- Buttons
  - Contextual menus
  - Gestures
  - Dialogs

- Constrained choices

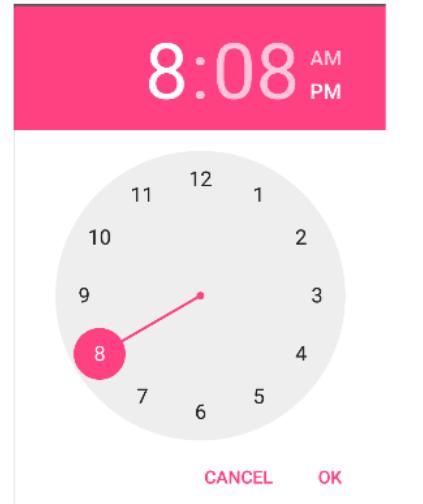
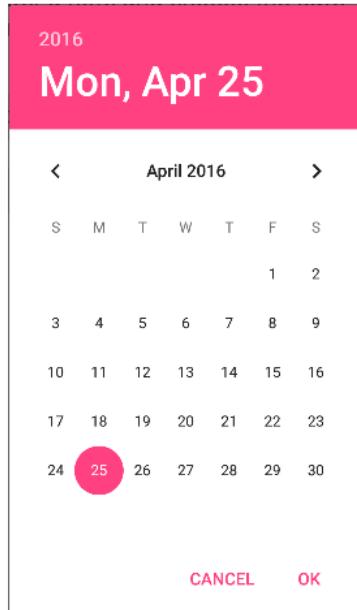
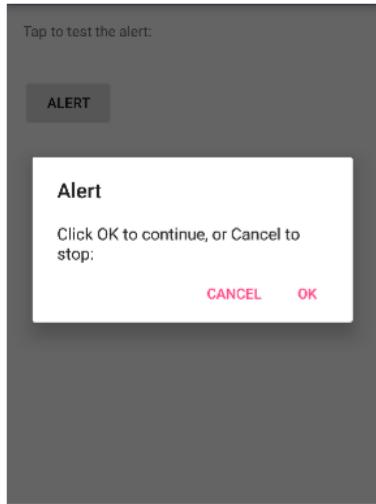
- Pickers
  - Checkboxes
  - Radio buttons
  - Toggle buttons
  - Spinners

# Examples of user input controls

1. Button
2. Text field
3. Seek bar
4. Checkboxes
5. Radio buttons
6. Toggle
7. Spinner



# Alert dialog, date picker, time picker



# 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

# 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

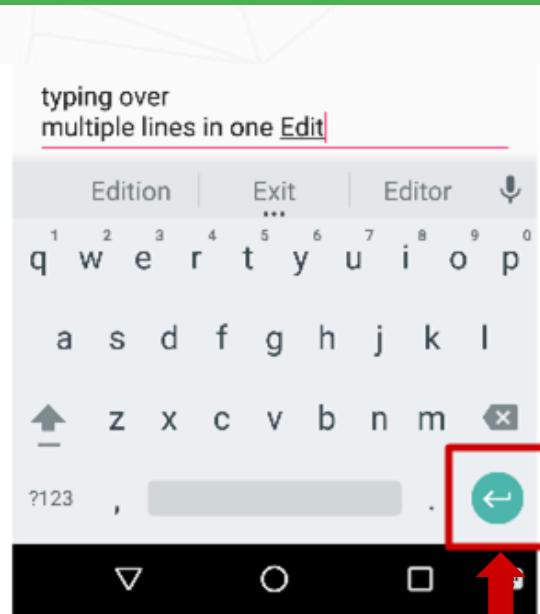
# Find the view with focus

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

# Text Input

# EditText

- [EditText](#) class
- Multiple lines of input
- Characters, numbers, and symbols
- Spelling correction
- Tapping the Return (Enter) key starts a new line
- Customizable



"Action"  
key

# Getting text

- Get the EditText object for the EditText view

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

- Retrieve the CharSequence and convert it to a string

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

# Common input types

- `textShortMessage`—Limit input to 1 line
- `textCapSentences`—Set keyboard to caps at beginning of sentences
- `textAutoCorrect`—Enable autocorrecting
- `textPassword`—Conceal typed characters
- `textEmailAddress`—Show an @ sign on the keyboard
- `phone`—numeric keyboard for phone numbers

`android:inputType="phone"`

`android:inputType="textAutoCorrect | textCapSentences"`

# Buttons

# Button

- View that responds to clicking or pressing
- Usually text or visuals indicate what will happen when it is pressed
- Views: [Button](#) > [ToggleButton](#), [ImageView](#) > [FloatingActionButton](#) (FAB)
- State: normal, focused, disabled, pressed, on/off
- Visuals: raised, flat, clipart, images, text



Alarm



Alarm

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

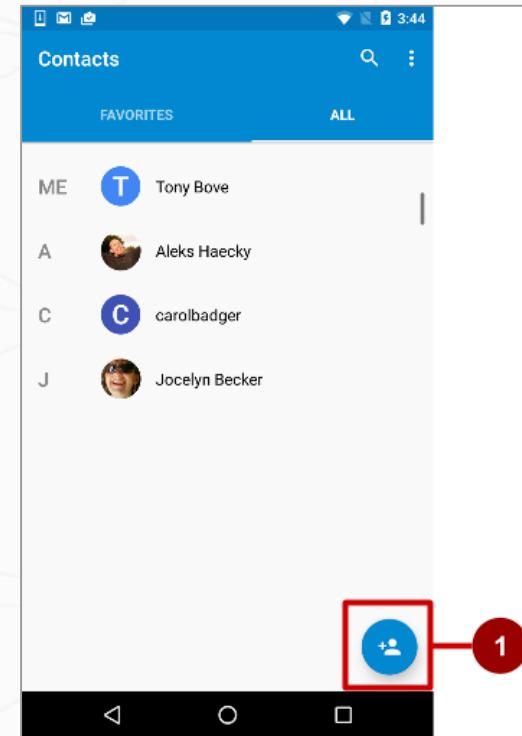
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

# 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

- Add design support library to build.gradle

```
compile 'com.android.support:design:a.b.c'
```

- 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"  
    .../>
```

# 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)



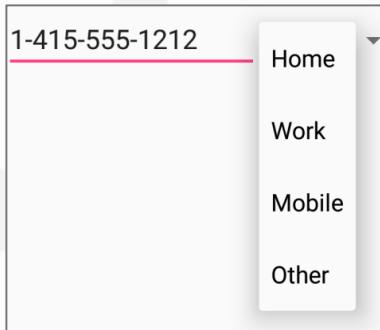
Experiment:

- 1.Choose **New > Vector Asset**

# Making Choices

# So many choices!

- Checkboxes
- Radio buttons
- Toggles
- Spinner



Choose a topping:  
 Chocolate Syrup  
 Sprinkles  
 Crushed Nuts

Choose a delivery method:  
 Same day messenger service  
 Next day ground delivery  
 Pick up

Turn on or off: **ON**

Turn on or off: **OFF**

Turn on or off:

Turn on or off:

# Checkboxes, radio buttons and toggles

# Checkboxes

- 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

Chocolate Syrup

Sprinkles

Crushed Nuts

# Radio buttons

- User can select one of a number of choices
- Put radio buttons in a RadioGroup
- Checking one unchecks another
- Put radio buttons in a vertical list or horizontally if labels are short
- Every radio button can have an 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 2 exclusive states (on/off)
- Use android:onClick+callback—or handle clicks in code

Turn on or off:

ON

Turn on or off:

OFF

## Toggle buttons

Turn on or off:



Turn on or off:

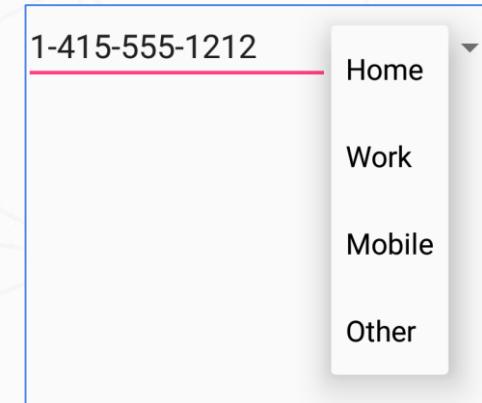


## Switches

# Spinners

# Spinners

- Quick way to select value from a set
- Drop-down list shows all values, user can select only one
- Spinners scroll automatically if necessary
- Use the Spinner class.



# Implementing a spinner

- 1.Create Spinner UI element in the XML layout
- 2.Define spinner choices in an array
- 3.Create Spinner and set [onItemSelectedListener](#)
- 4.Create an adapter with default spinner layouts
- 5.Attach the adapter to the spinner
- 6.Implement `onItemSelectedListener` method

# Create spinner XML

In layout XML file

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

# Define array of spinner choices

In arrays.xml resource file

```
<string-array name="labels_array">
    <item>Home</item>
    <item>Work</item>
    <item>Mobile</item>
    <item>Other</item>
</string-array>
```

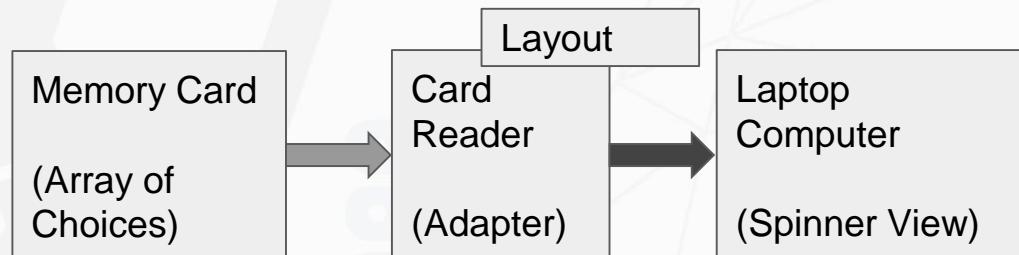
# Create spinner and attach listener

```
public class MainActivity extends AppCompatActivity implements  
AdapterView.OnItemSelectedListener  
  
// In onCreate()  
  
Spinner spinner = (Spinner) findViewById(R.id.label_spinner);  
if (spinner != null) {  
    spinner.setOnItemSelectedListener(this);  
}
```

# What is an adapter?

An adapter is like a bridge, or intermediary, between two incompatible interfaces

For example, a memory card reader acts as an adapter between the memory card and a laptop



# Create adapter

Create ArrayAdapter using string array  
and default spinner layout

```
ArrayAdapter<CharSequence> adapter =  
    ArrayAdapter.createFromResource(  
        this, R.array.labels_array,  
        // Layout for each item  
        android.R.layout.simple_spinner_item);
```

# Attach the adapter to the spinner

- Specify the layout for the drop down menu

```
adapter.setDropDownViewResource(  
    android.R.layout.simple_spinner_dropdown_item);
```

- Attach the adapter to the spinner

```
spinner.setAdapter(adapter);
```

# Implement onItemSelectedListener

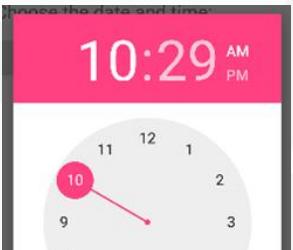
```
public class MainActivity extends AppCompatActivity implements  
AdapterView.OnItemSelectedListener  
  
    public void onItemSelected(AdapterView<?> adapterView,  
        View view, int pos, long id) {  
        String spinner_item =  
            adapterView.getItemAtPosition(pos).toString();  
        // Do something here with the item  
    }  
  
    public void onNothingSelected(AdapterView<?> adapterView) {  
        // Do something  
    }  

```

# Dialogs

# Dialogs

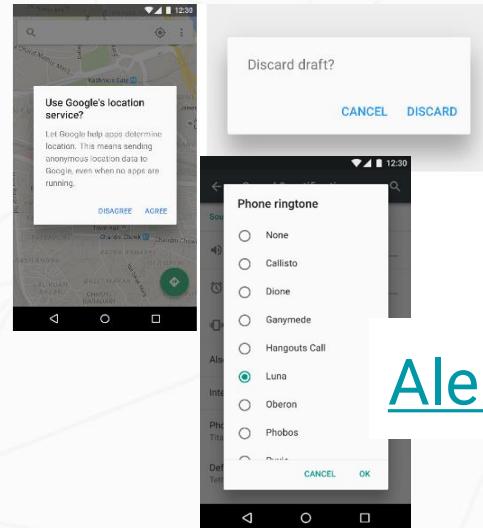
- Dialog appears on top, interrupting the flow of activity
- Require user action to dismiss



TimePickerDialog  
g



DatePickerDialog

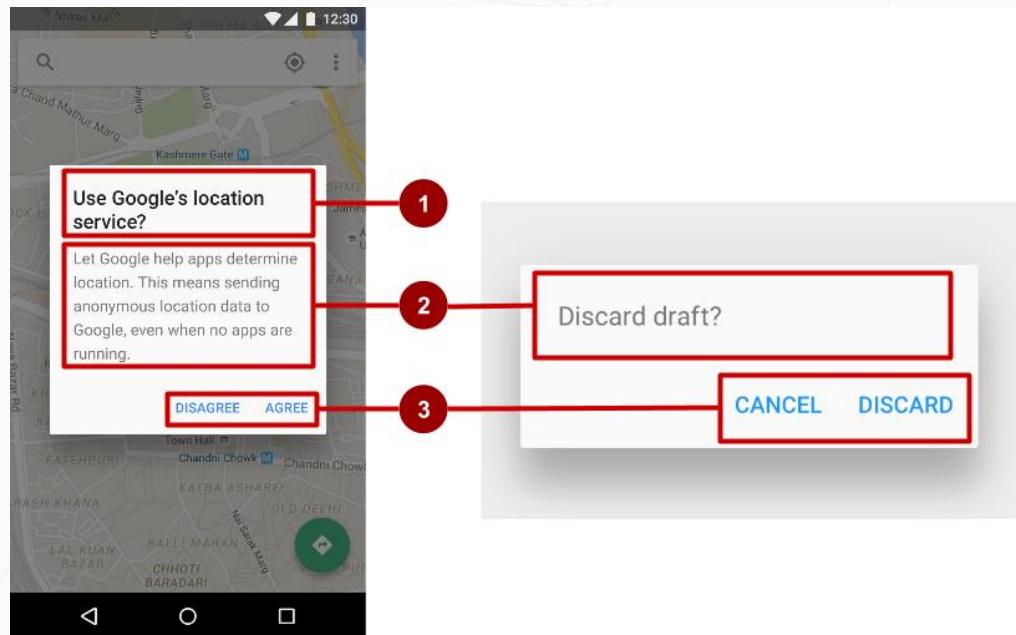


AlertDialog

# AlertDialog

AlertDialog can show:

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



# Build the AlertDialog

Use `AlertDialog.Builder` to build a standard 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);  
    ...
```

# Add the button actions

- alertDialog.setPositiveButton()
- alertDialog.setNeutralButton()
- alertDialog.setNegativeButton()

# AlertDialog code example

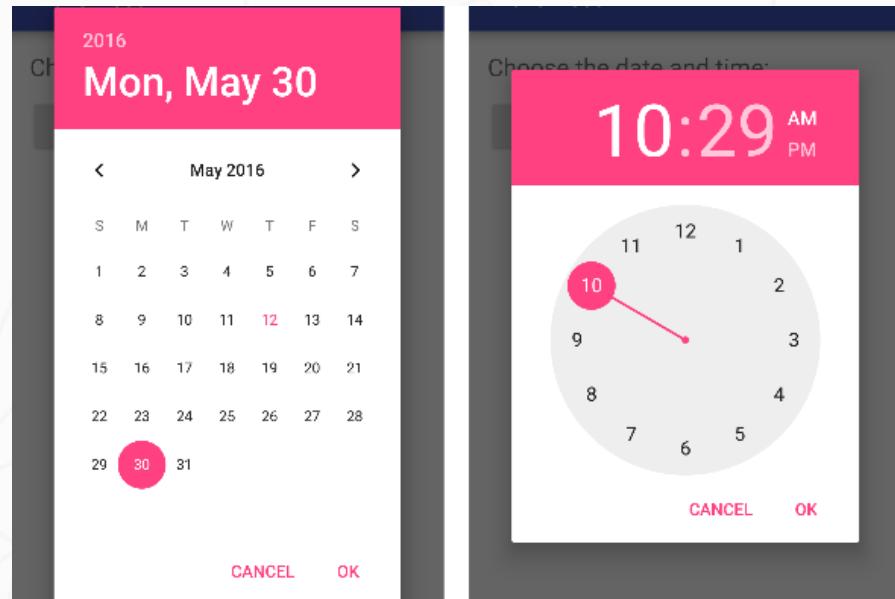
```
AlertDialog.setPositiveButton(  
    "OK", newDialogInterface.OnClickListener() {  
        public void onClick(DialogInterface dialog, int which) {  
            // User clicked OK button.  
        }  
    });
```

Same pattern for setNegativeButton() and setNeutralButton()

# Pickers

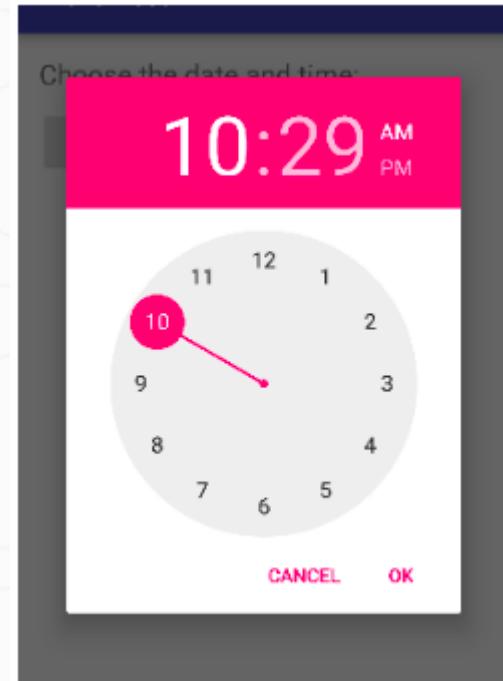
# Pickers

- DatePickerDialog
- TimePickerDialog



# Pickers use fragments

- Use [DialogFragment](#) to show a picker
- DialogFragment is a window that floats on top of activity's 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 multiple activities

# 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 a method to use the 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 a method to use the time

# 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 C User Input Controls](#)
- Practical:  
[4. P Using Keyboards, Input Controls, Alerts, and Pickers](#)

**END**



Android  
Developer  
Associate

## Lesson 4



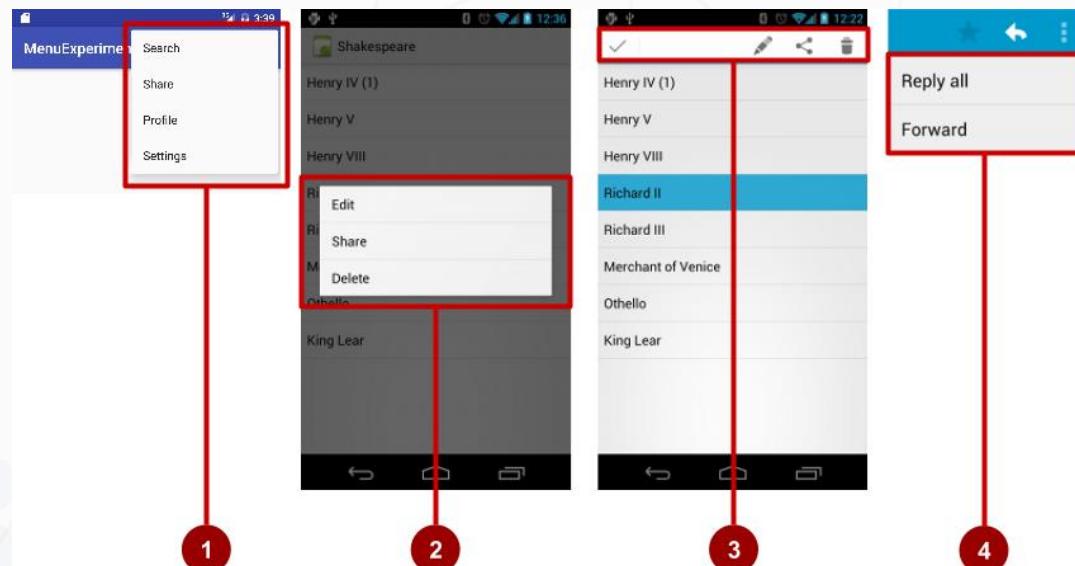
## 4.2 Menus

# Contents

- App Bar with Options Menu
- Contextual menus
- Popup menus

# Types of Menus

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

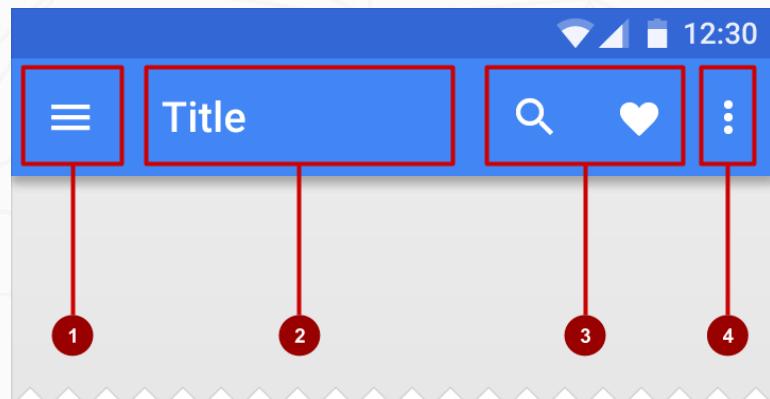


# App Bar with Options Menu

# What is the App Bar?

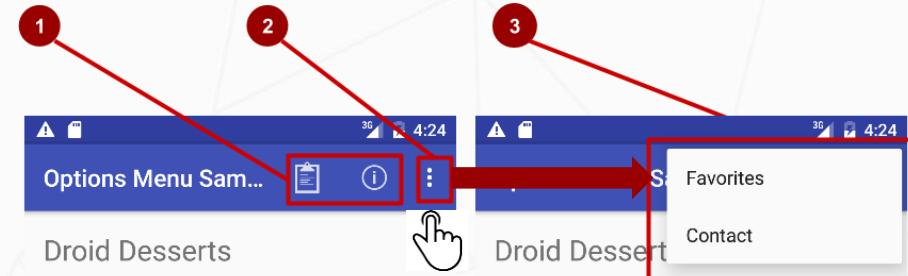
Bar at top of each screen—(usually) the same for all screens

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?

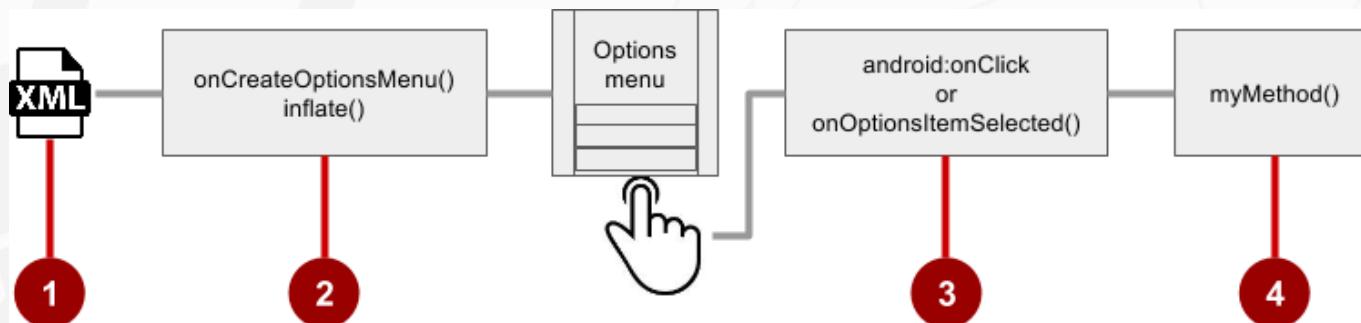
- Action icons in the app bar for important items (1)
- Tap the three dots, the "action overflow button" to see the options menu (2)
- Appears in the right corner of the app bar (3)
- For navigating to other activities and editing app settings



# 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 an entry for each menu item

```
<item android:id="@+id/option_settings"  
      android:title="@string/settings" />  
  
<item android:id="@+id/option_toast"  
      android:title="@string/toast" />
```

# Inflate options menu

- Override onCreateOptionsMenu() in main 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**

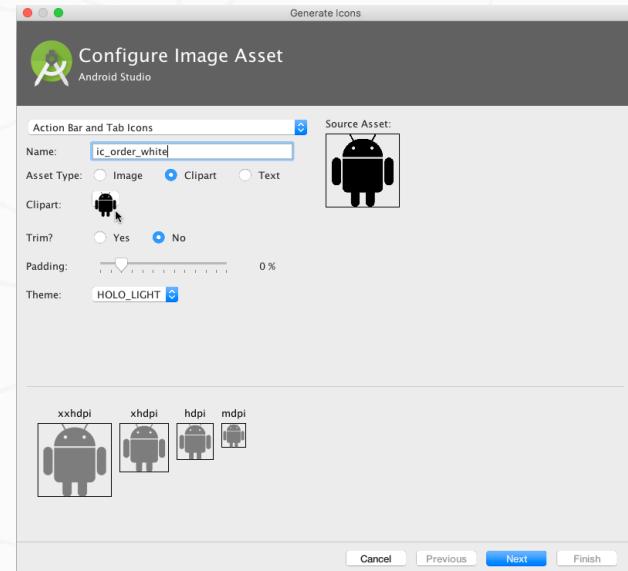
2. Choose **New > Image Asset**

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_order"
      android:icon="@drawable/ic_toast_dark"
      android:title="@string/toast"
      android:titleCondensed="@string/toast_condensed"
      android:orderInCategory="1"
      app:showAsAction="ifRoom" />
```

# Override onOptionsItemSelected()

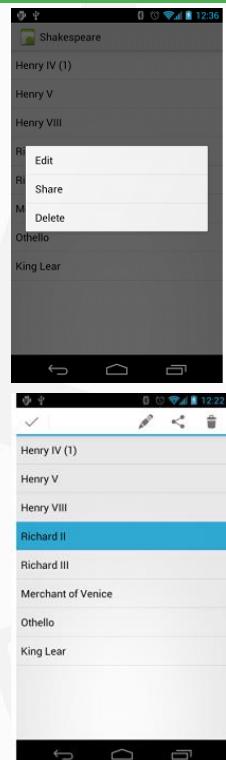
```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.action_order:  
            showOrder();  
            return true;  
        default:  
            return super.onOptionsItemSelected(item);  
    }  
}
```

# Contextual Menus

# What are contextual menus?

- Allow users to perform an action on a selected view or content
- Can be deployed on any View object, but most often used for items in a RecyclerView, GridView, or other view collection

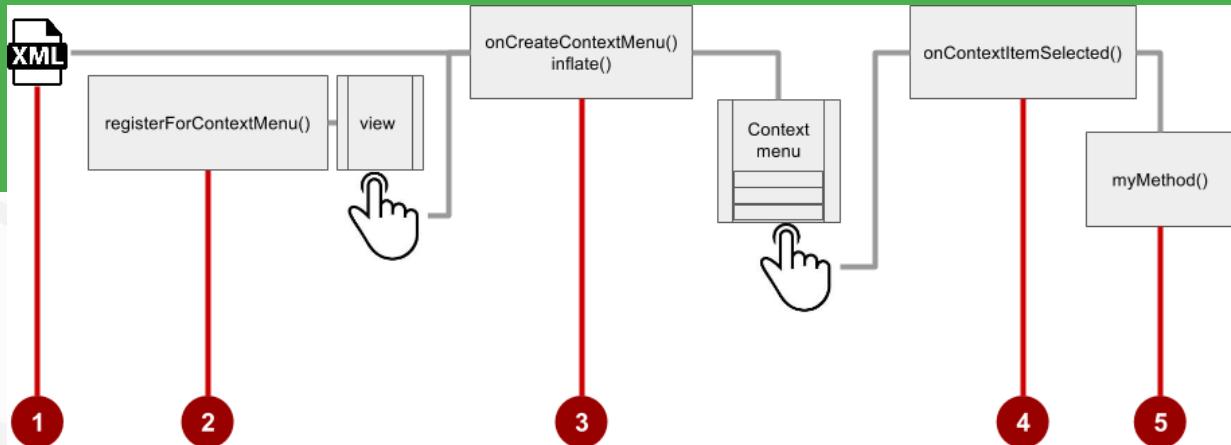
# Types of contextual menus



- Floating context menu—floating list of menu items when long-presses on a view element
  - User can modify the view element or use it in some fashion
  - Users perform a contextual action on one view element at a time
- Contextual action mode—temporary action bar in place of or underneath the app bar
  - Action items affect the selected view element(s)
  - Users 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 to use a context menu using `registerForContextMenu()`
3. Implement `onCreateContextMenu()` in the activity or fragment to inflate the menu
4. Implement `onContextItemSelected()` to handle menu item clicks
5. Create a method to perform an action for each context menu item

# Create menu resource

- Create XML menu resource (`menu_context.xml`)

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

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

# Register a view to a context menu

- in onCreate() of the activity
- registers [View.OnCreateContextMenuListener](#)
- Does not specify which context menu!

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

# Implement onCreateContextMenu()

Specifies which context menu

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

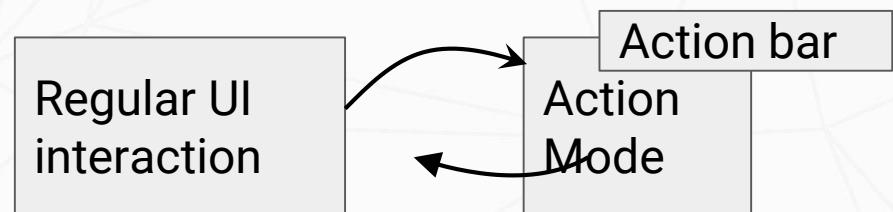
# Implement onContextItemSelected()

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

# Contextual Action Bar

# What is Action Mode?

- ActionMode is a UI mode that lets you replace parts of the 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 the lifecycle methods that you can override
  - `onCreateActionMode(ActionMode, Menu)` once on initial creation
  - `onPrepareActionMode(ActionMode, Menu)` after creation and any time `ActionMode` is invalidated
  - `onActionItemClicked(ActionMode, MenuItem)` any time a contextual action button is clicked
  - `onDestroyActionMode(ActionMode)` when the action mode is closed

# What is a contextual action bar?

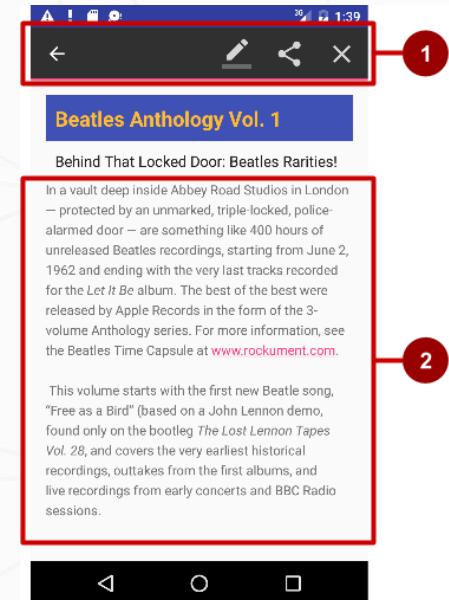
Long-tap on the view shows contextual action bar

## 1. Contextual action bar with actions

- Edit, Share, and Delete
- Done (left arrow icon) on the left side

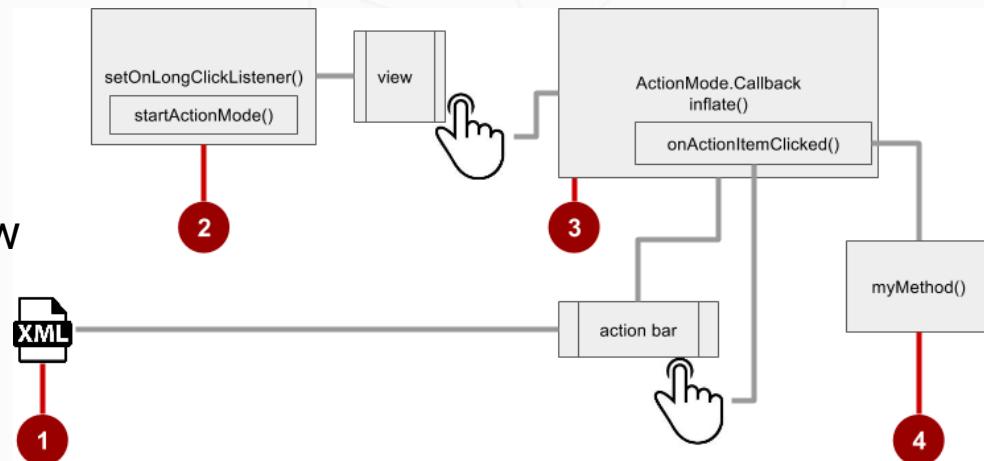
## 2. View on which long press triggers the contextual action bar

- Action bar is available until user taps Done



# Steps for contextual action bar

1. Create XML menu resource file and assign icons for items
2. setOnLongClickListener() on view that triggers the contextual action bar and call startActionMode() to handle click
3. Implement ActionMode.Callback interface to handle ActionMode lifecycle; include action for a menu item click in onActionItemClicked() callback
4. Create a method to perform an 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 the action mode is shown
- Always called after onCreateActionMode, but may be called multiple times if the 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.goodbyetextview:  
            Toast.makeText(getApplicationContext(), "Menu Toast",  
Toast.LENGTH_SHORT).show();  
            mode.finish(); // Action picked, so close the action bar  
            return true;  
        default:  
            return false;  
    }  
}
```

# Implement onDestroyActionMode

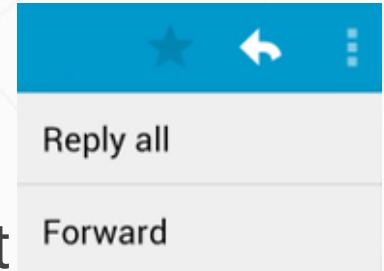
- Called when user exits the action mode

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

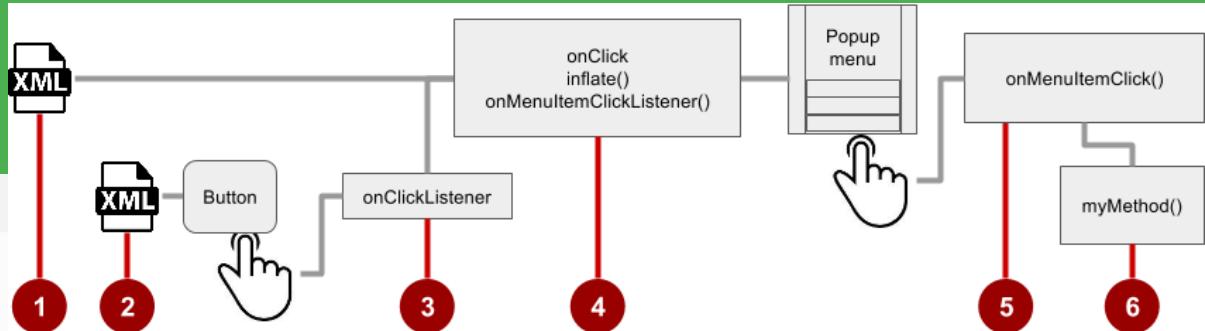
# Popup Menu

# 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
  - The options menu overflow that opens Settings
  - For example, in an email app, Reply All and Forward are related to the email message, but don't affect or act on the message



# Steps



1. Create XML menu resource file and assign appearance and position attributes
2. Add an ImageButton for the popup menu icon in the XML activity layout file
3. Assign onClickListener to the button
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 an 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_toast:  
            Toast.makeText(getApplicationContext(), "Popup Toast",  
                Toast.LENGTH_SHORT).show();  
            return true;  
        default:  
            return false;  
    }  
}
```

# Learn more

- [Adding the App Bar](#)
- [Styles and Themes](#)
- [Menus](#)
- [Menu Resource](#)

# What's Next?

- Concept Chapter: [4.2 C Menus](#)
- Practical: [4.2 P Using an Options Menu](#)

**END**



Android  
Developer  
Associate

## Lesson 4



# 4.3 Screen 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

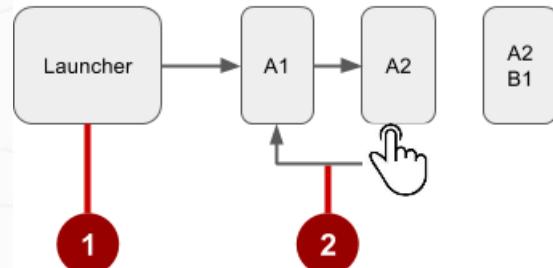
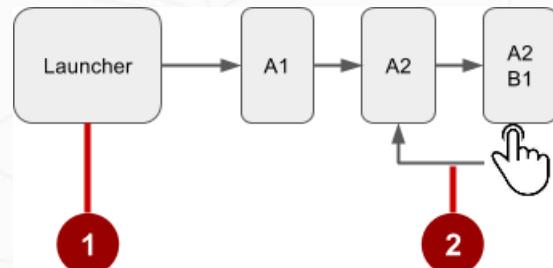
# Two forms of navigation

-  Temporal or back navigation
  - provided by the device's back button
  - controlled by the Android system's back stack
-  Ancestral or up navigation
  - provided by the app's action bar
  - controlled by defining parent-child relationships between activities in the Android manifest

# Back Navigation

# Navigation through history of screens

1. History starts from Launcher
2. User clicks the Back  button to navigate to the 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

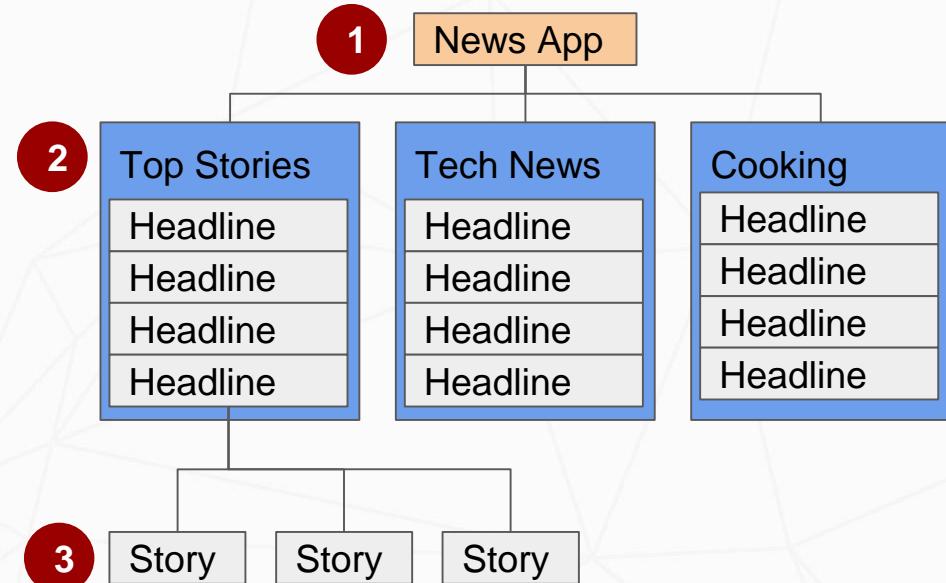
# Hierarchical navigation patterns

- **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 screen
- 2.Children: collection siblings
- 3.Children: section siblings

Use activities or fragments to implement a hierarchy



# Types of hierarchical navigation

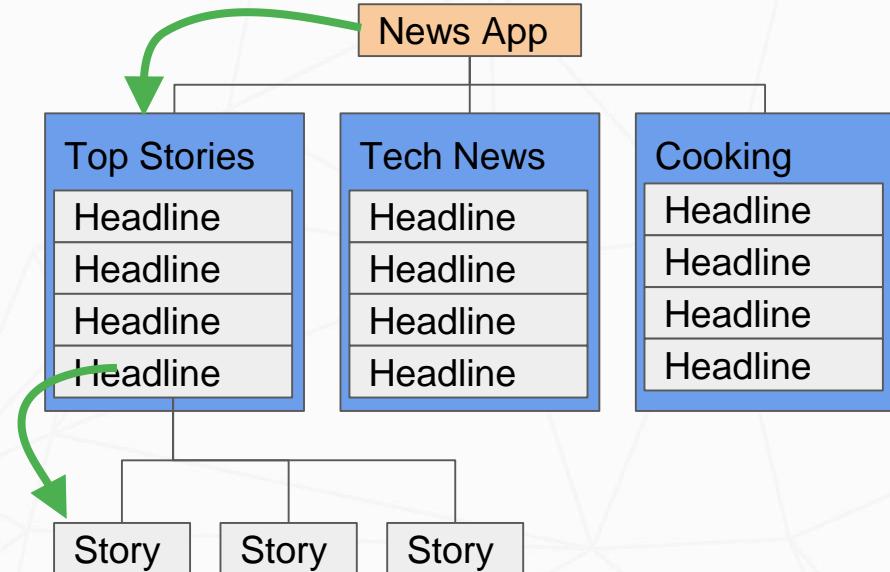
- Descendant navigation
  - Down from a parent screen to one of its children
  - From a list of headlines to a story summary to a story
- Ancestral navigation
  - Up from a child or sibling screen to its parent
  - From a story summary back to the headlines
- Lateral navigation
  - From one sibling to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

# Descendant Navigation

# Descendant navigation

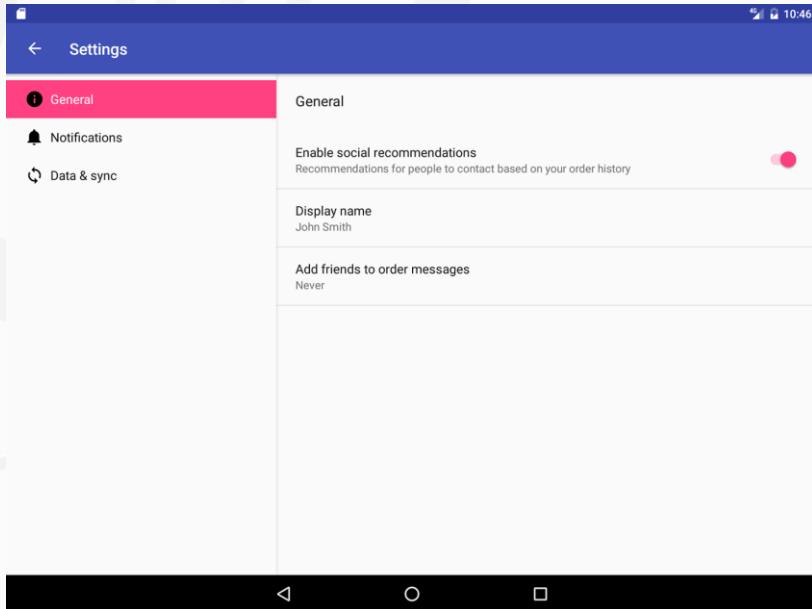
## Descendant navigation

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

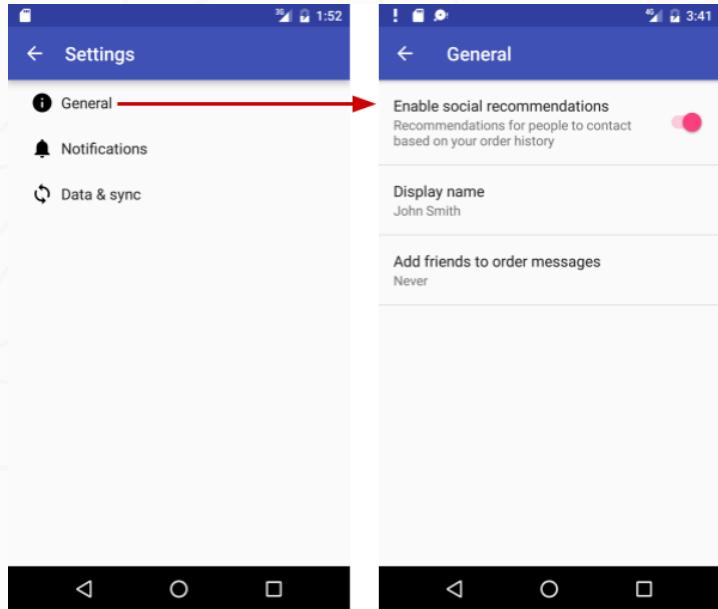


# Master/detail flow

- Side-by side on tablets



- Multiple screens on phone



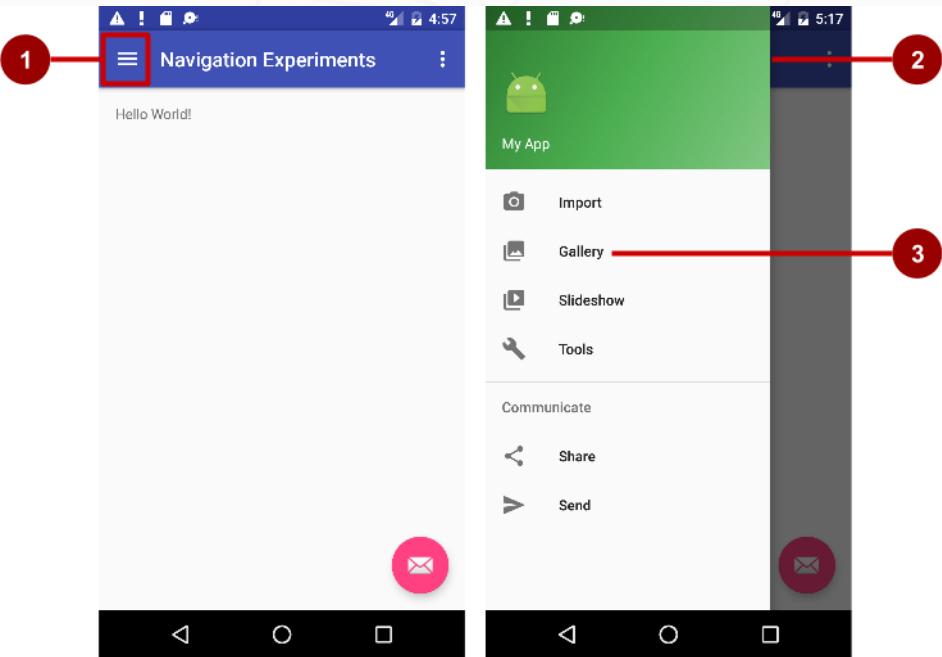
# Controls for descendant navigation

- 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 for Descendant Navigation

# Navigation drawer

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

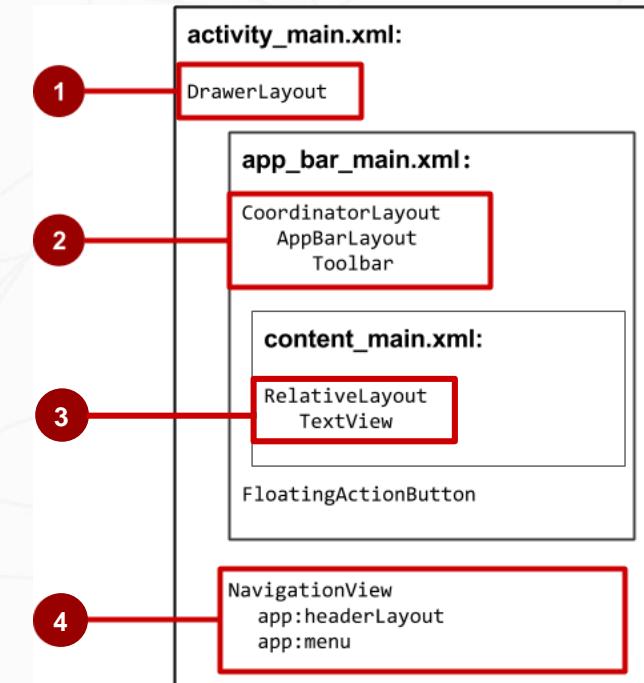


# Steps for navigation drawer

- 1.Create layouts for drawer, drawer header, drawer menu items, app bar, activity screen contents
- 2.Add navigation drawer and item listeners to activity code
- 3.Handle the navigation drawer menu item selections

# 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



# Other descendant navigation patterns

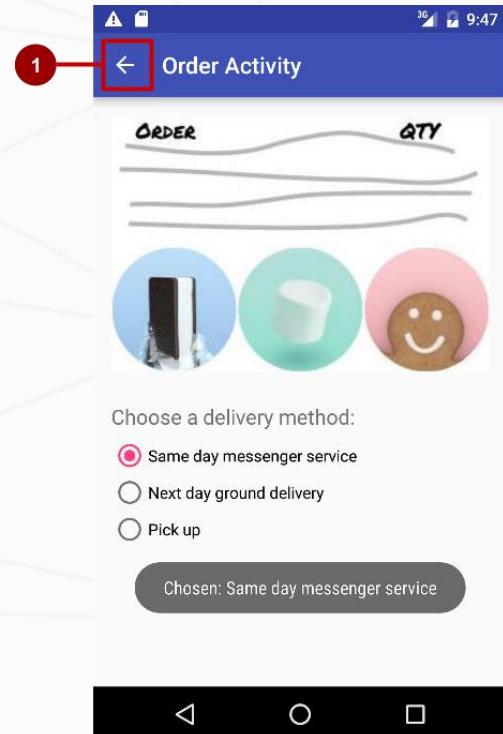
- Vertical list, such as [RecyclerView](#)
- Vertical grid, such as [GridView](#)
- Lateral navigation with a Carousel
- Multi-level menus, such as the Options menu
- Master/detail navigation flow

# Ancestral Navigation

# Ancestral navigation (Up button)



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



# Declare activity's parent in Android manifest

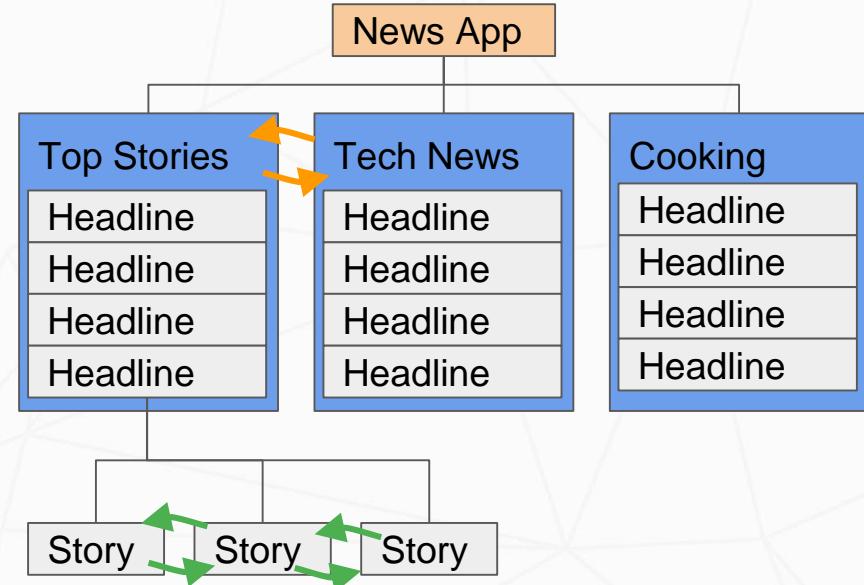
```
<activity android:name=".OrderActivity"
    android:label="@string/title_activity_order"
    android:parentActivityName="com.example.android.
                                optionsmenuorderactivity.MainActivity">
<meta-data
    android:name="android.support.PARENT_ACTIVITY"
    android:value=".MainActivity"/>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

# Lateral Navigation

# Tabs and swipes

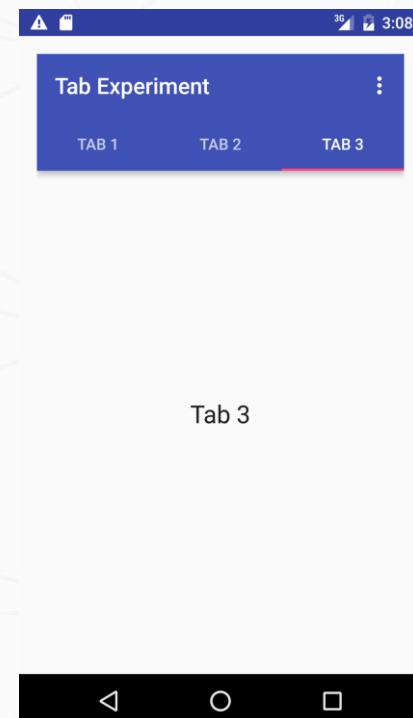
Lateral navigation

- 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. Manage screen views in fragments
6. Set a listener to determine which tab is tapped

See Practical for coding details; summary in following slides

# 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 = (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 = (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: [4.3 C Screen Navigation](#)
- Practical: [4.3 P Using the App Bar and Tabs for Navigation](#)

**END**



Android  
Developer  
Associate

Lesson 4



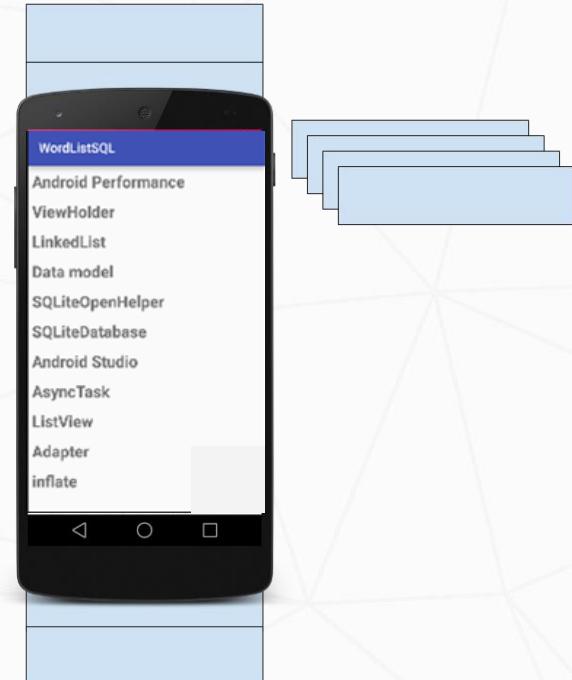
# 4.4 Recycler View

# Contents

- RecyclerView Components
- Implementing a RecyclerView

# What is a Recycler View?

- Scrollable container for large data sets
- Efficient
  - uses and reuses limited number of views
  - Updates changing data fast
- RecyclerView

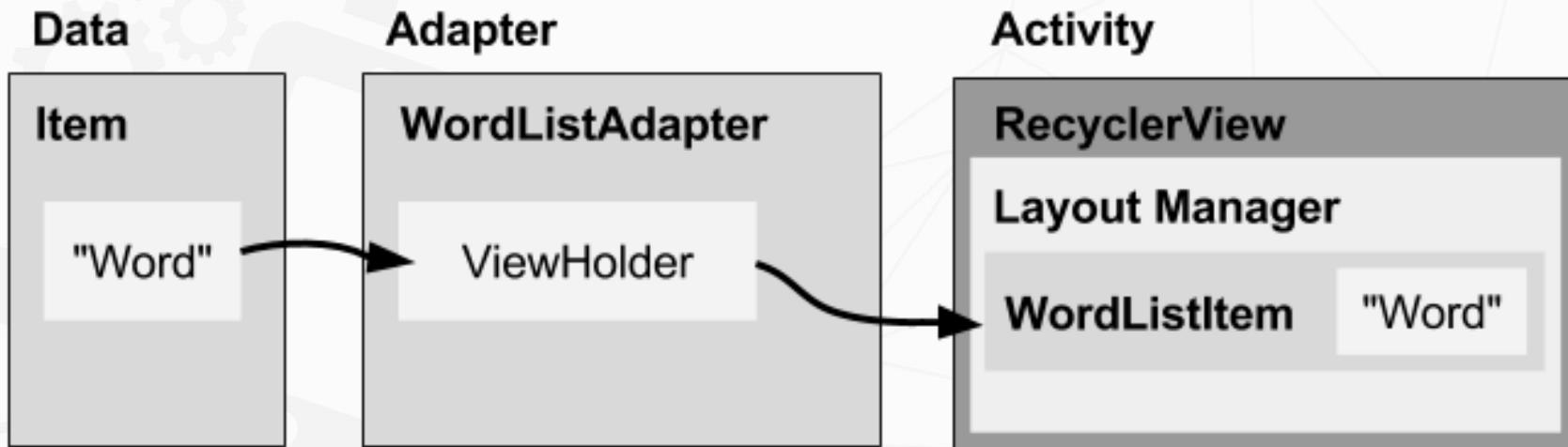


# RecyclerView Components

# All 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](#)
- **View holder** has view information for displaying one item—[RecyclerView.ViewHolder](#)

# How components fit together overview

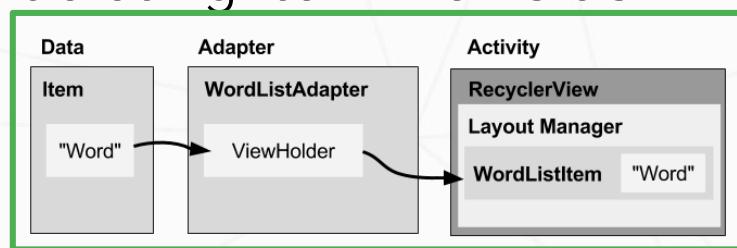


# What is a layout manager?

- All view groups have layout managers
- Positions item views inside a [RecyclerView](#).
- Reuses item views that are no longer visible to the user
- Built-in layout managers include [LinearLayoutManager](#),  
[GridLayoutManager](#), and [StaggeredGridLayoutManager](#)
- For RecyclerView, extend [RecyclerView.LayoutManager](#)

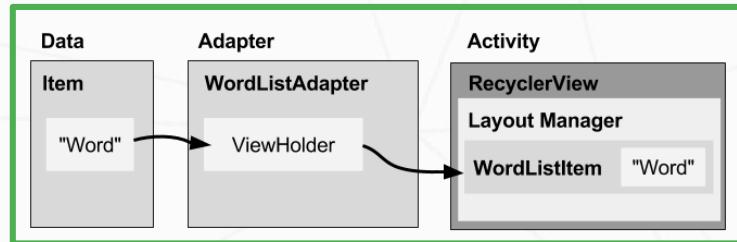
# What is an adapter?

- Helps incompatible interfaces work together, for example, takes data from a database [Cursor](#) and puts them as strings into a view
- Intermediary between data and view
- Manages creating, updating, adding, deleting item views as the underlying data changes
- [RecyclerView.Adapter](#)



# What is a view holder?

- 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 the RecyclerView dependency to app/build.gradle file
- 2.Add RecyclerView to layout
- 3.Create XML layout for item
- 4.Extend RecyclerView.Adapter
- 5.Extend RecyclerView.ViewHolder
- 6.In onCreate of activity, create a RecyclerView with adapter and layout manager

# Add dependency to app/build.gradle

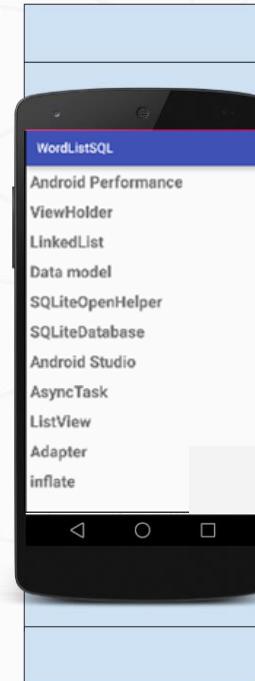
```
dependencies {  
    ...  
    compile 'com.android.support:recyclerview-v7:24.1.1'  
    ...  
}
```

# 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 = (TextView) 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's onCreate()

```
mRecyclerView = (RecyclerView)  
        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: [4.4 C RecyclerView](#)
- Practical: [4.4 P Create a Recycler View](#)

**END**



Android  
Developer  
Associate

Lesson 5



# 5.1 Drawables, Styles, 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

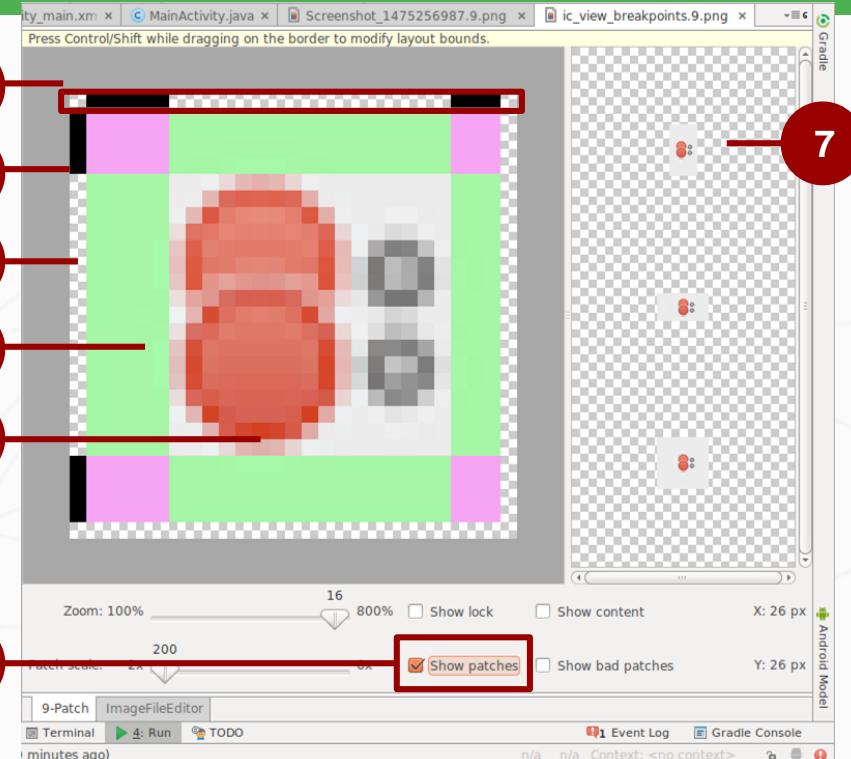
- Nine-patch files (.9.png) are PNG with stretchable regions
- Only stretches bigger, not smaller, so start with small image
- Often used for backgrounds of UI elements
- Example: button background changes size with label length
- Good intro

# 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 = (ImageButton) 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*



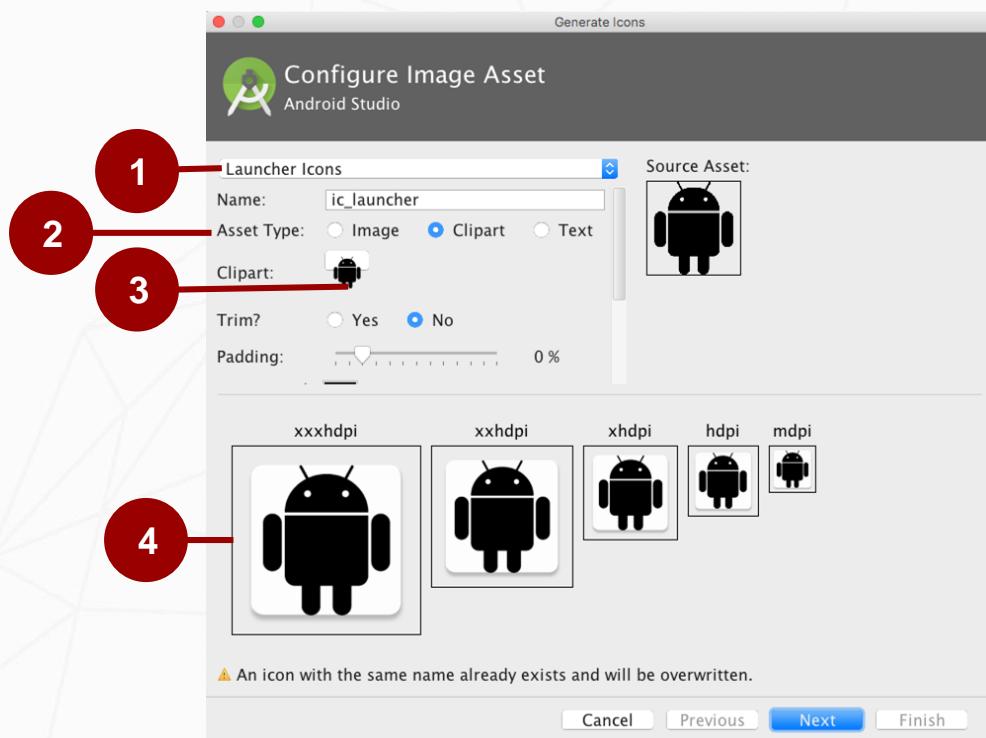
# Image Asset Studio

# 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 **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 choose clipart
4. Inspect assets for multiple screen sizes



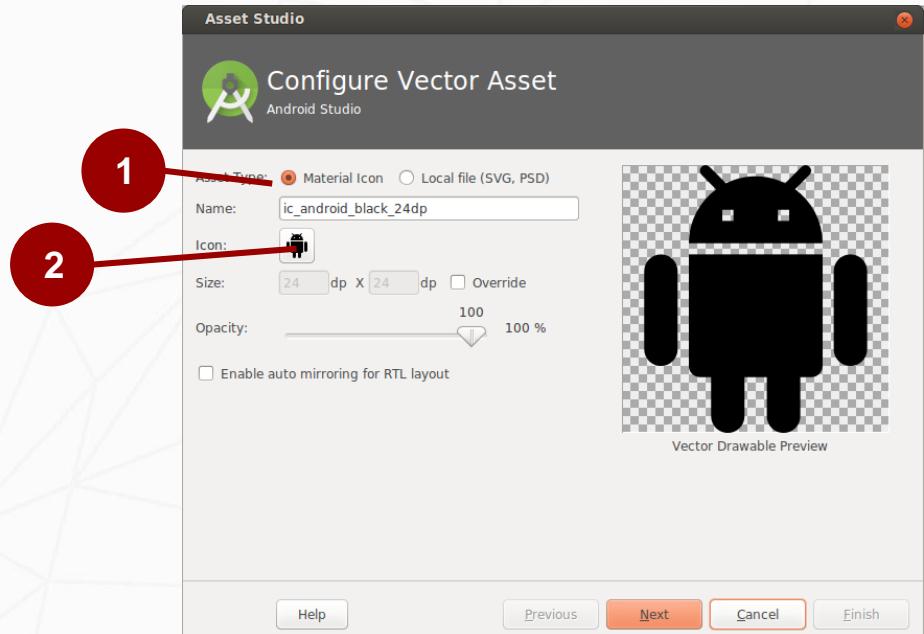
# Vector Asset Studio

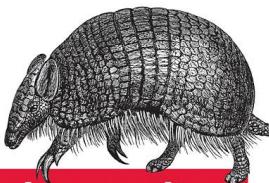
# What is Vector Asset Studio?

- 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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CodeFont">
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

# Inheritance: Parent

Define a parent style...

```
<?xml version="1.0" encoding="utf-8"?>
<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

```
<?xml version="1.0" encoding="utf-8"?>
<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 the Android Manifest

```
<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 in styles and themes

- [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 C Drawables, Styles, and Themes](#)
- Practical: [5.1 P Drawables, Styles, and Themes](#)

**END**



Android  
Developer  
Associate

## Lesson 5



# 5.2 Material Design

# Contents

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

# The Material Metaphor

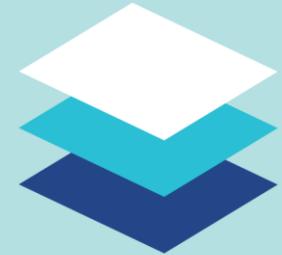
# What is Material Design?

- 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

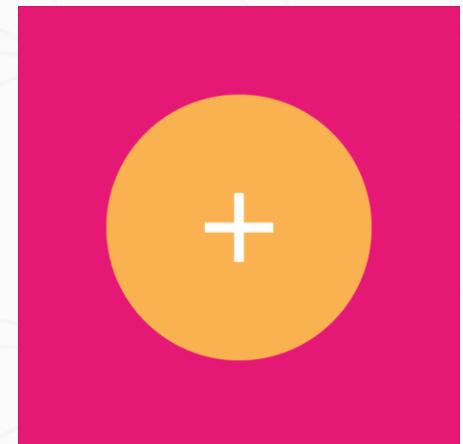
Elements in your Android app should behave similarly to real world materials

- 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

$6.626069 \times 10^{-34}$

*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)

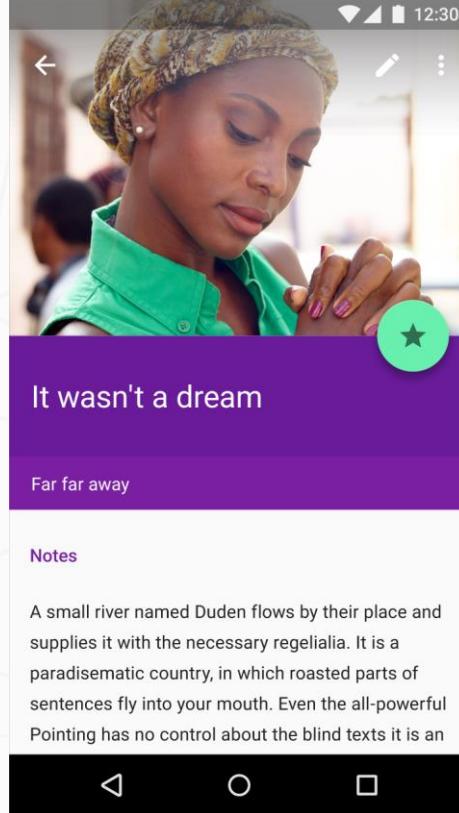
Regular 12sp

MEDIUM (ALL CAPS) 14sp

# Setting text appearance

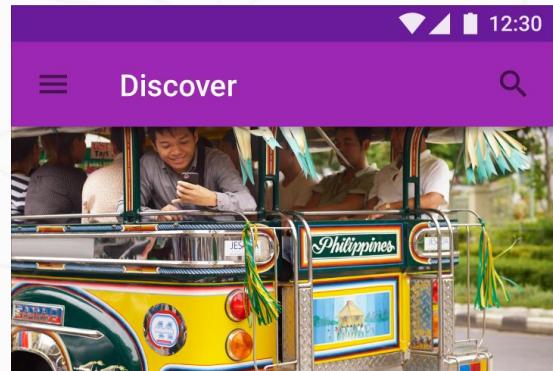
```
android:textAppearance=  
    "@style/TextAppearance.AppCompat.Display3"
```

# 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

Primary – Purple	
500	#9B26AF
700	#7A1EA1
800	#691A99
Accent – Green	
A200	#68EFAD

# Text color and contrast

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

Good choice

Good choice

Bad choice

Bad choice

Bad choice

Good choice

# Motion



# Motion

Motion in Material Design  
describes

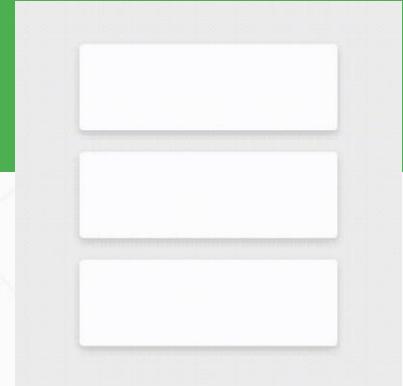
- Spatial relationships
- Functionality
- Intention

Motion is

- Responsive
- Natural
- Aware
- Intentional

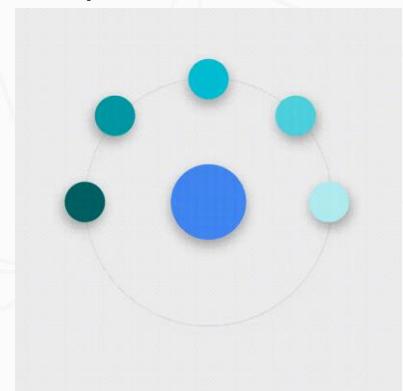
# Motion in your app

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

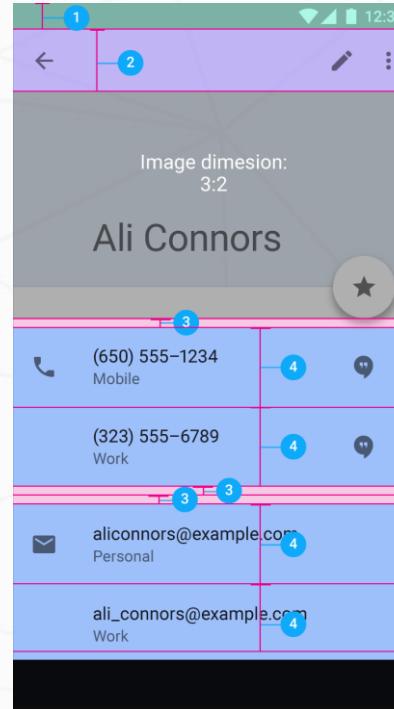


Touch feedback

Responsive interaction



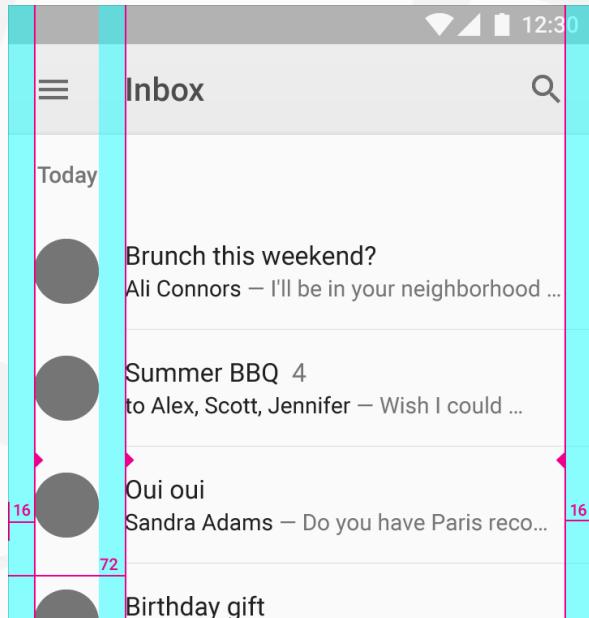
# Layout



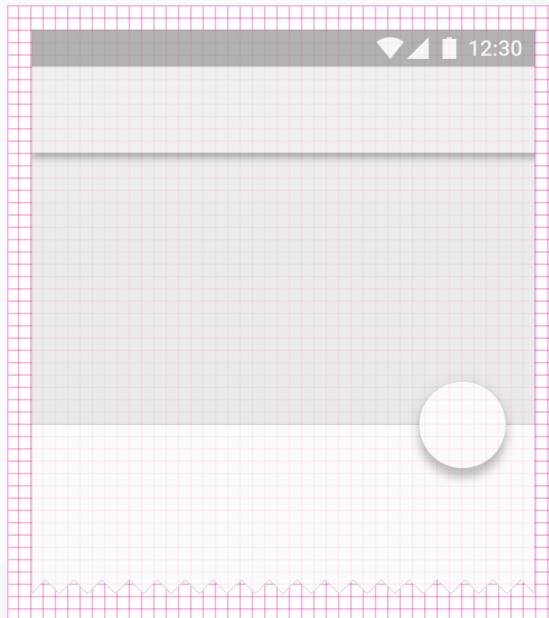
# Layout for Material Design

- 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

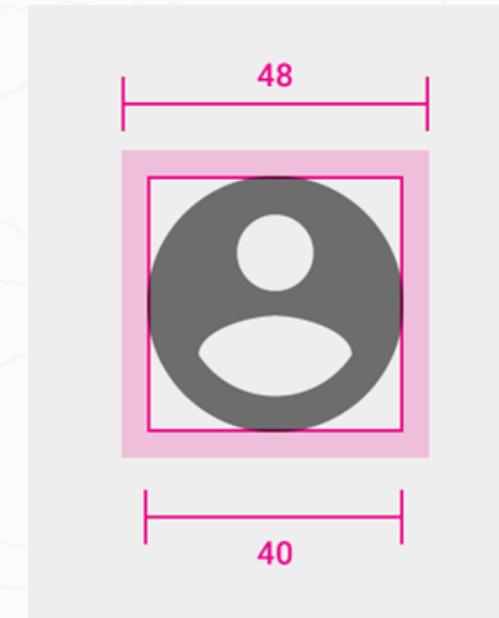
# Layout planning



Spacing

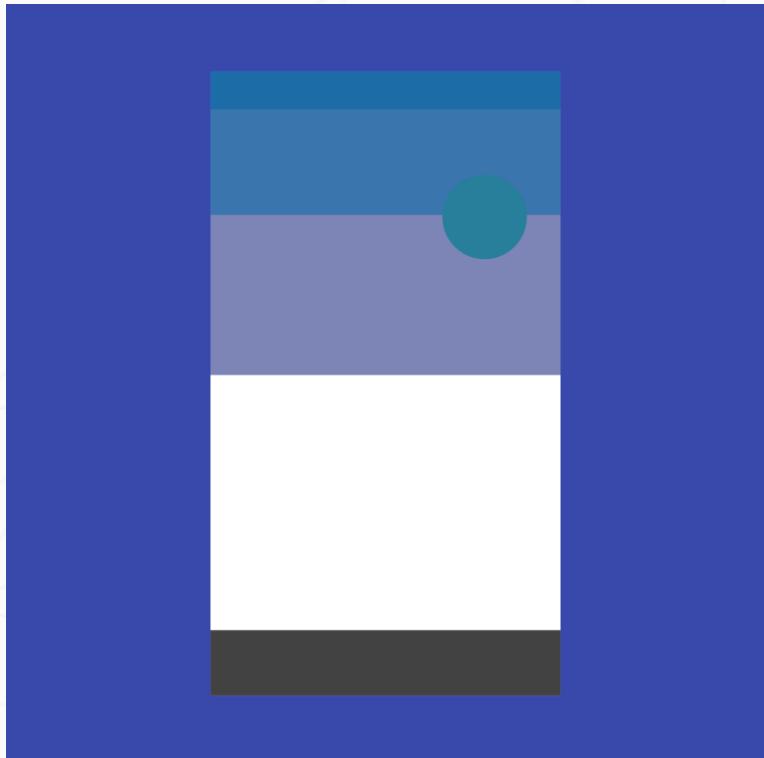


Grid alignment



Sizing

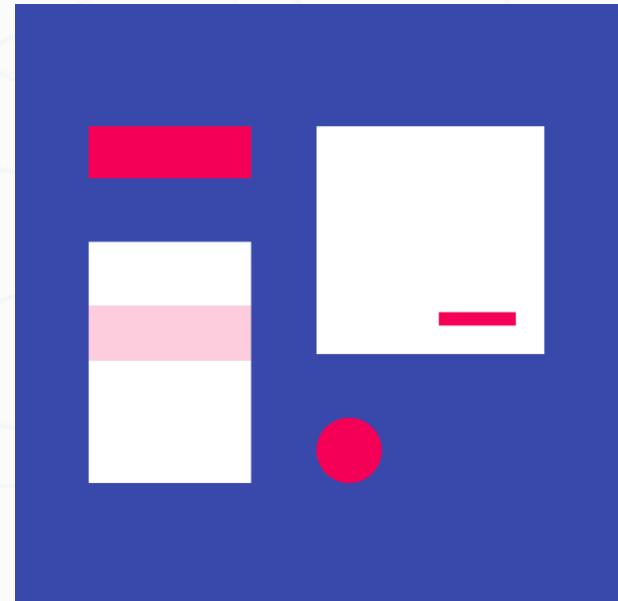
# Components



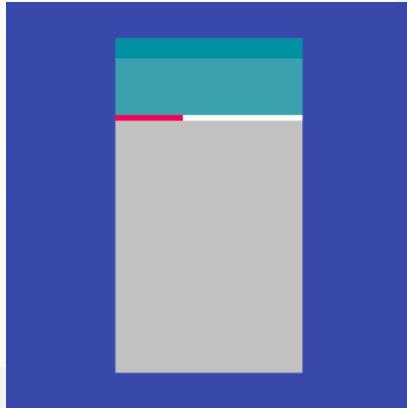
# Components

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

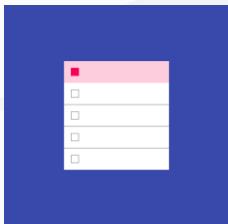
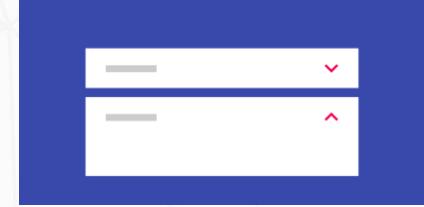
- Bottom Navigation
- Buttons
- Cards
- Chips
- Data Tables
- Dialogs
- Dividers
- Sliders
- Snackbar
- Toasts
- Steppers
- Subheaders
- Text Fields
- Toolbars



# More components

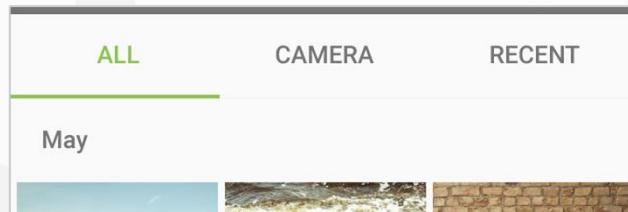
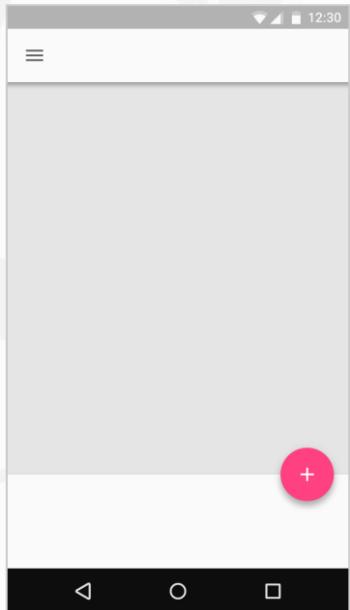


- Expansion Panels
- Grid Lists
- Lists
- Menus
- Pickers
- Progress Bars
- Selection Controls

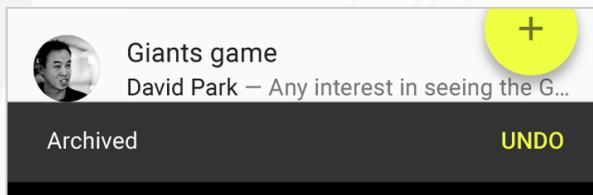


# Consistency helps user intuition

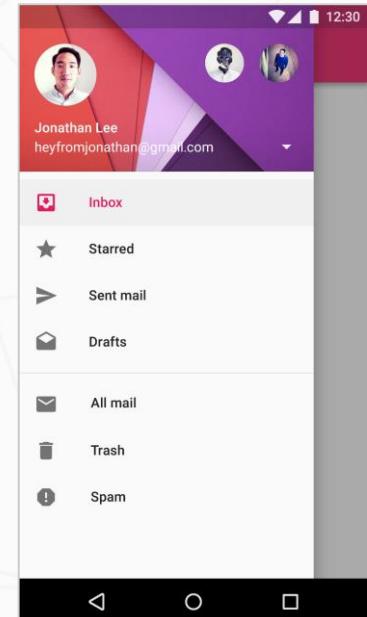
FAB



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 C Material Design](#)
- Practical: [5.2 P Material Design: Lists, Cards, and Colors](#)

**END**



Android  
Developer  
Associate

Lesson 5



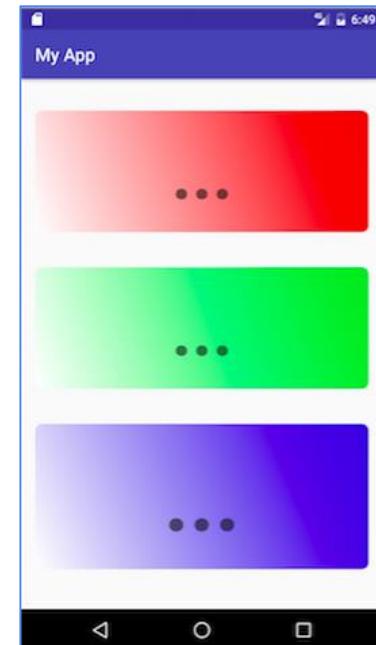
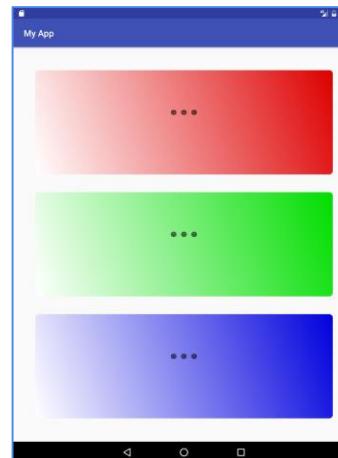
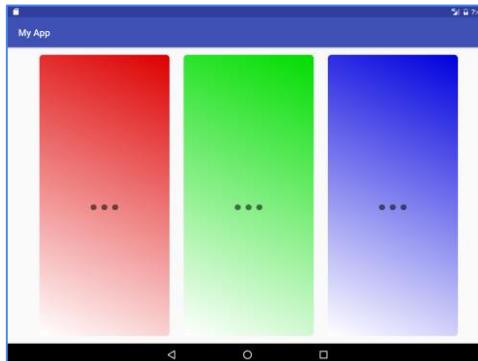
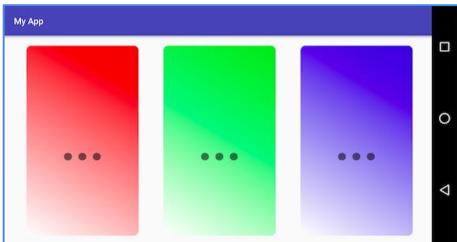
# 5.3 Adaptive Layouts and Resources

# Contents

- Adaptive Layout
- Adaptive Resources
- Alternative Resources
- Default Resources

# What are adaptive layouts?

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



# Adaptive Layouts and Resources

# Adaptive layouts

- 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 folders

# 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

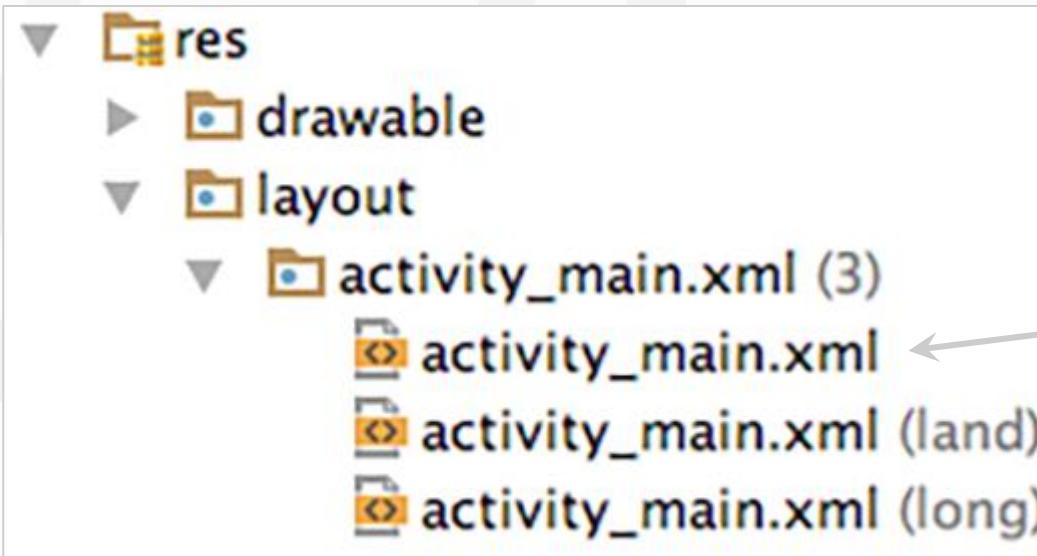
# What are alternative resources?

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>*

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

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

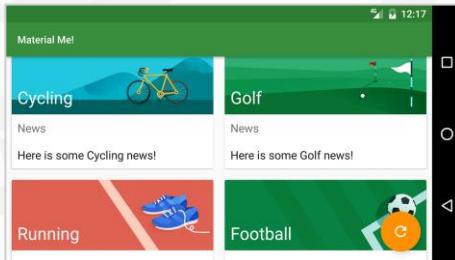
# Screen Orientation

- 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

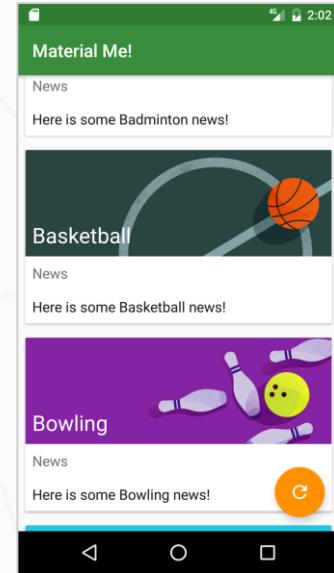
# Simple adaptive layout

## 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 C Providing Resource for Adaptive Layouts](#)
- Practical:  
5.3 P Supporting Landscape, Multiple Screen Sizes, and Location

**END**



Android  
Developer  
Associate

Lesson 6



# 6.1 Testing the User Interface

# Contents

- Testing Methods
- Automated Testing
- Using Espresso
- Testing Environment and Setup
- Creating Espresso Tests
- Espresso Test Examples
- Recording Tests

# Testing Methods

# User interface testing

- Perform all user UI actions with views
  - Tap a UI view, and enter data or make a choice
  - Examine the values of the properties of each view
- Provide input to all UI views
  - 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

# Testing Environment & 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

```
androidTestCompile 'com.android.support:support-annotations:24.1.1'  
androidTestCompile 'com.android.support.test:runner:0.5'  
androidTestCompile 'com.android.support.test:rules:0.5'  
androidTestCompile 'org.hamcrest:hamcrest-library:1.3'  
androidTestCompile  
        'com.android.support.test.espresso:espresso-core:2.2.2'  
androidTestCompile  
        'com.android.support.test.uiautomator:uiautomator-v18:2.1.2'
```

# Add defaultConfig dependencies

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

# Writing 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. 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 ...
```

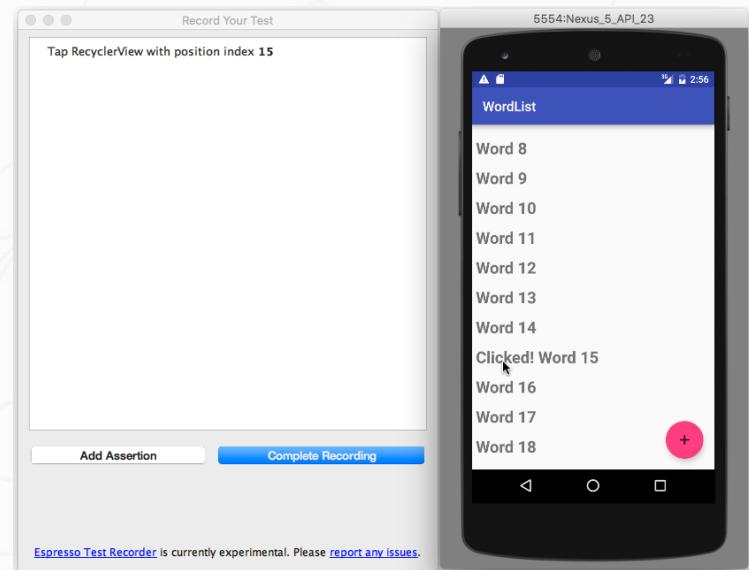
# Recording Tests

# Recording an Espresso test

- Android Studio 2.2
- 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

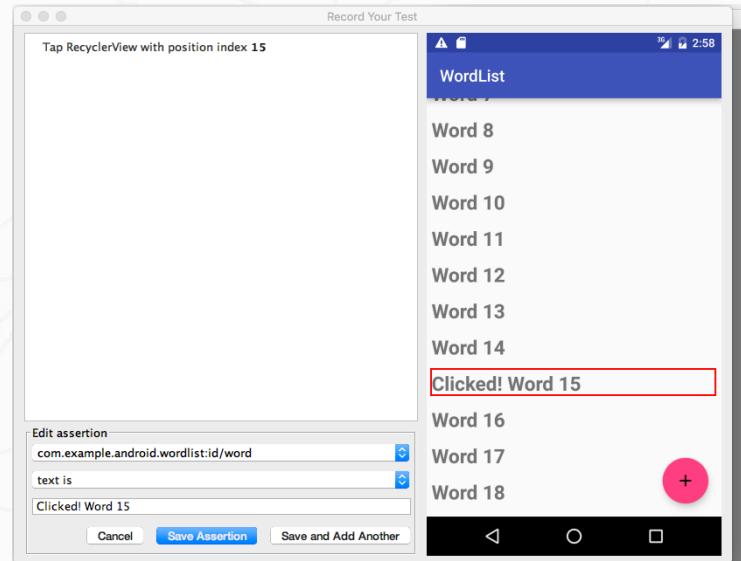
# Start recording an Espresso test

- 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

1. Click **Add Assertion** and select a UI element
2. Choose **text is** and enter the text you expect to see
3. 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 C Testing the User Interface](#)
- Practical: [6.1 P Use Espresso to Test Your UI](#)

**END**



Android  
Developer  
Associate

## Lesson 7



# 7.1 AsyncTask & 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

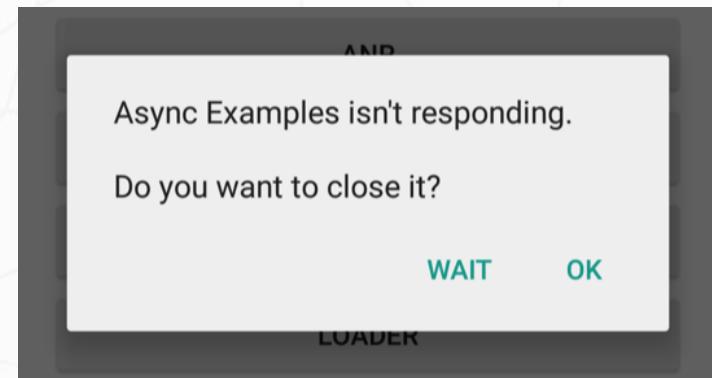
# The Main thread must be fast

- 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



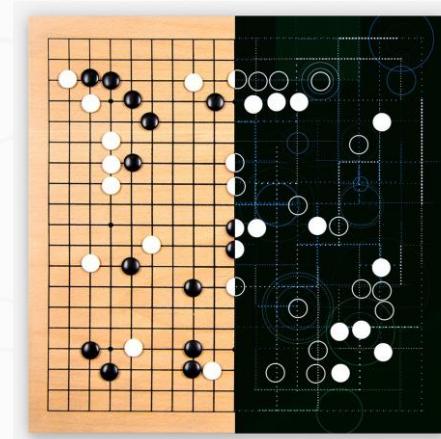
# Users uninstall unresponsive apps

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



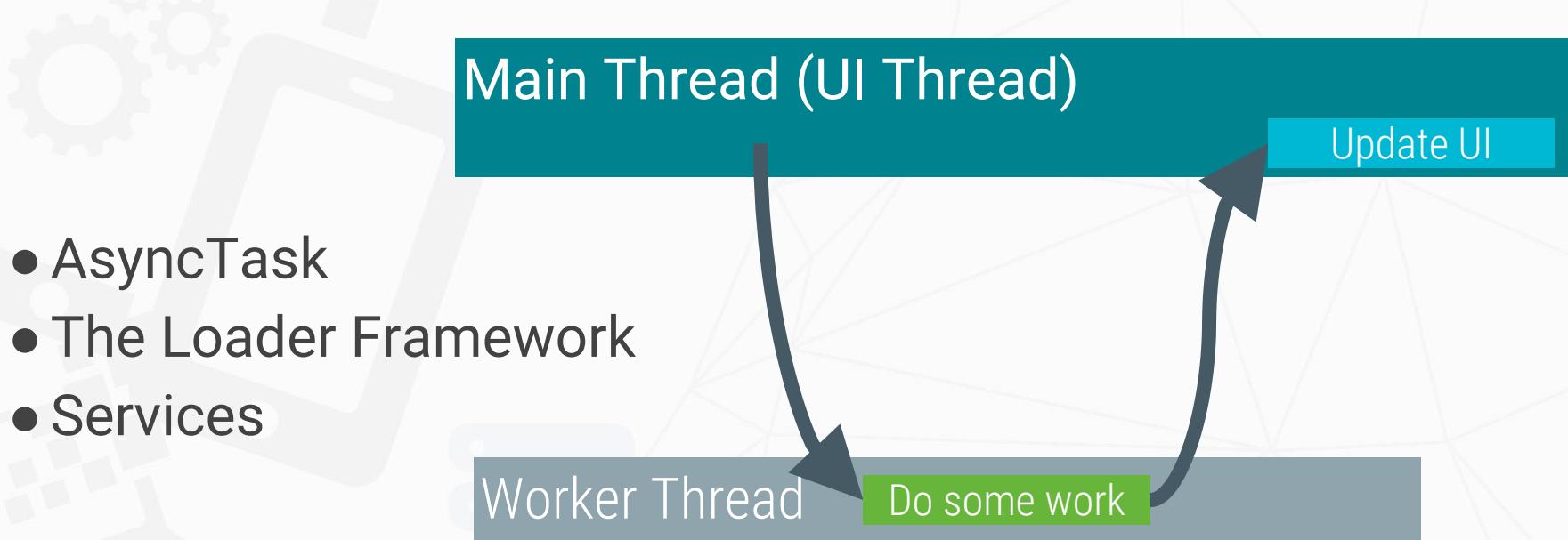
# What is a long running task?

- 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

Main Thread (UI Thread)

onPostExecute()

Worker Thread

doInBackground()

# 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

Main Thread (UI Thread)

onPreExecute()

onProgressUpdate()

onPostExecute()

Worker Thread

publishProgress()

doInBackground()

# 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...

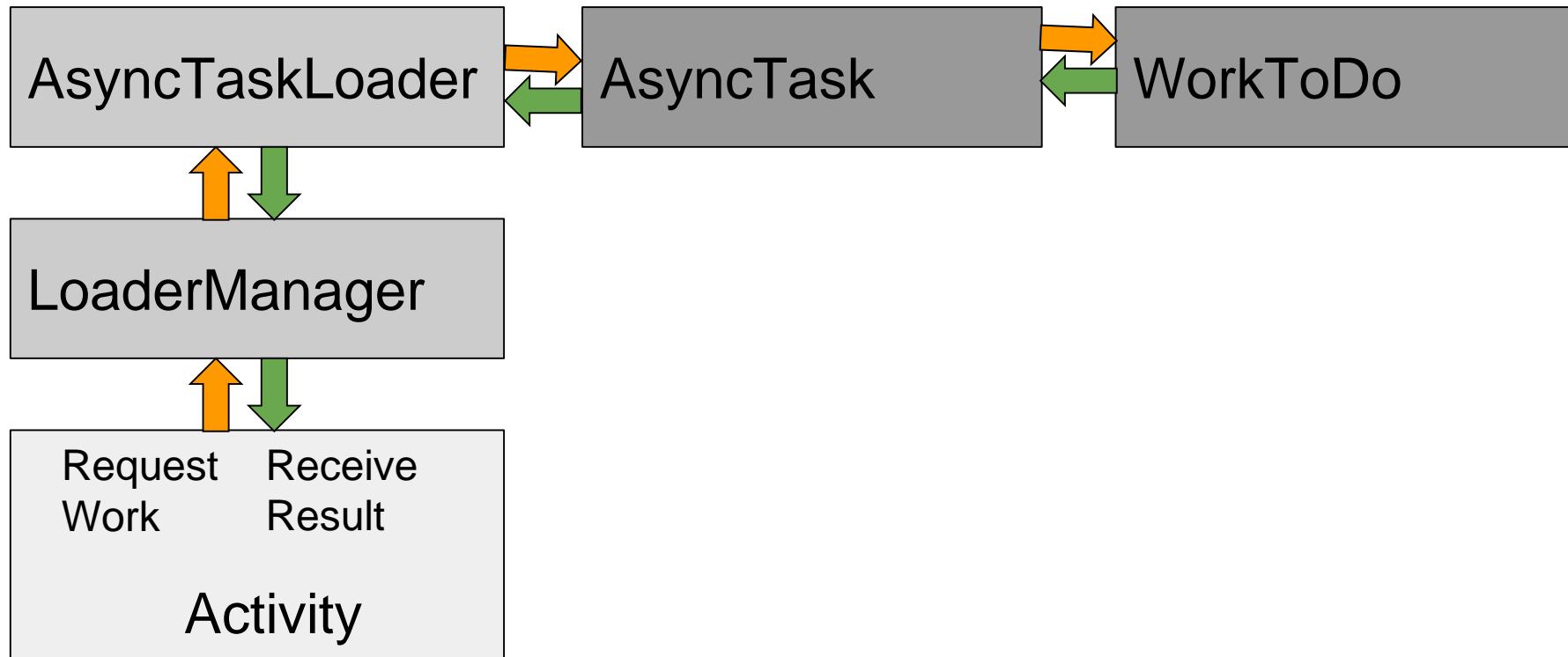
# Get a loader with initLoader()

- 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 AsyncTaskLoade r

# 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 C AsyncTask and AsyncTaskLoader](#)
- Practical: [7.1 P Create an AsyncTask](#)

**END**



Android  
Developer  
Associate

## Lesson 7



## 7.2 Connect to the Internet

# 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

# 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

# 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

# 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 C Connect to the Internet](#)

Practical:

- [7.2 P Connect to the Internet with AsyncTask and AsyncTaskLoader](#)



# **END**



Android  
Developer  
Associate

## Lesson 7



# 7.3 Broadcast Receivers

# Contents

- Broadcast intents
- Broadcast receivers
- Implementing broadcast receivers
- Custom broadcasts
- Security
- Local broadcasts

# Broadcast Intents

# Broadcast vs. Activity

Use implicit intents to send broadcasts or start activities

## Sending broadcasts

- Use `sendBroadcast()`
- Can be received by any application registered for the intent
- Used to notify all apps of an event

## Starting activities

- Use `startActivity()`
- Find a single activity to accomplish a task
- Accomplish a specific action

# Broadcast Receivers

# What is a broadcast receiver?

- Listens for incoming intents sent by sendBroadcast()
  - In the background
- Intents can be sent
  - By the system, when an event occurs that might change the behavior of an app
  - By another application, including your own

# Broadcast receiver always responds

- Responds even when your app is closed
  - Independent from any activity
  - When a broadcast intent is received and delivered to `onReceive()`, it has 5 seconds to execute, and then the receiver is destroyed

# System broadcasts

- Automatically delivered when certain events occur
- After the system completes a boot
  - android.intent.action.BOOT\_COMPLETED
- When the wifi state changes
  - android.net.wifi.WIFI\_STATE\_CHANGED

# Custom broadcasts

- Deliver any custom intent as a broadcast
  - `sendBroadcast()` method—asynchronous
  - `sendOrderedBroadcast()`—synchronously
  - `android.example.com.CUSTOM_ACTION`

# sendBroadcast()

- All receivers of the broadcast are run in an undefined order
- Can be at the same time
- Efficient
- Use to send custom broadcasts

# sendOrderedBroadcast()

- Delivered to one receiver at a time
- Receiver can propagate result to the next receiver or abort the broadcast
- Control order with [android:priority](#) of matching intent filter
- Receivers with same priority run in arbitrary order

# Implementing Broadcast Receivers

# Steps for creating a broadcast receiver

1. Subclass BroadcastReceiver
2. Implement onReceive() method
3. Register to receive broadcast
  - Statically, in AndroidManifest
  - Dynamically, with registerReceiver()

# File > New > Other > BroadcastReceiver

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

# Register in Android Manifest

- <receiver> element inside <application>
- <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

- In onCreate() or onResume()
- Use registerReceiver() and pass in the intent filter
- Must unregister in onDestroy() or onPause()

```
registerReceiver(mReceiver, mIntentFilter)
```

```
unregisterReceiver(mReceiver)
```

# Available intents

- ACTION TIME TICK
- ACTION TIME CHANGED
- ACTION TIMEZONE CHANGED
- ACTION BOOT COMPLETED
- ACTION PACKAGE ADDED
- ACTION PACKAGE CHANGED
- ACTION PACKAGE REMOVED
- ACTION PACKAGE RESTARTED
- ACTION PACKAGE DATA CLEARED
- ACTION PACKAGES SUSPENDED
- ACTION PACKAGES UNSUSPENDED
- ACTION UID REMOVED
- ACTION BATTERY CHANGED
- ACTION POWER CONNECTED
- ACTION POWER DISCONNECTED
- ACTION SHUTDOWN

# Implement onReceive()

```
@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;  
    }  
}
```

# Custom Broadcasts

# Custom broadcasts

- Sender and receiver must agree on unique name for intent (action name)
- Define in activity and broadcast receiver

```
private static final String ACTION_CUSTOM_BROADCAST =  
    "com.example.android.powerreceiver.ACTION_CUSTOM_BROADCAST";
```

# Send custom broadcasts

```
Intent customBroadcastIntent =  
    new Intent(ACTION_CUSTOM_BROADCAST);  
  
LocalBroadcastManager.getInstance(this)  
    .sendBroadcast(customBroadcastIntent);
```

# Destroy!

```
@Override  
protected void onDestroy() {  
    LocalBroadcastManager.getInstance(this)  
        .unregisterReceiver(mReceiver);  
    super.onDestroy();  
}
```

# Security

# Security

- Receivers cross app boundaries
- Make sure namespace for intent is unique and you own it
- Other apps can send broadcasts to your receiver—use permissions to control this
- Other apps can respond to broadcast your app sends
- Access permissions can be enforced by sender or receiver

# Controlling permission sender

- void sendBroadcast (Intent intent,  
String receiverPermission)
- Receivers must request permission with <uses-permission> in AndroidManifest.xml

# Controlling permission receiver

- registerReceiver(BroadcastReceiver,  
IntentFilter, String, android.os.Handler)
- or in <receiver> tag
- Senders must request permission with <uses-  
permission> in AndroidManifest.xml

# Local Broadcast Manager

# Local Broadcast Manager

- For broadcasts only in your app
- No security issues since no cross-app communication

`LocalBroadcastManager.sendBroadcast()`

`LocalBroadcastManager.registerReceiver()`

# Register local broadcast manager

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

# Learn more

- [BroadcastReceiver Reference](#)
- [Intents and Intent Filters Guide](#)
- [LocalBroadcastManager Reference](#)
- [Manipulating Broadcast Receivers On Demand](#)

# What's Next?

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

**END**



Android  
Developer  
Associate

Lesson 7



# 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?

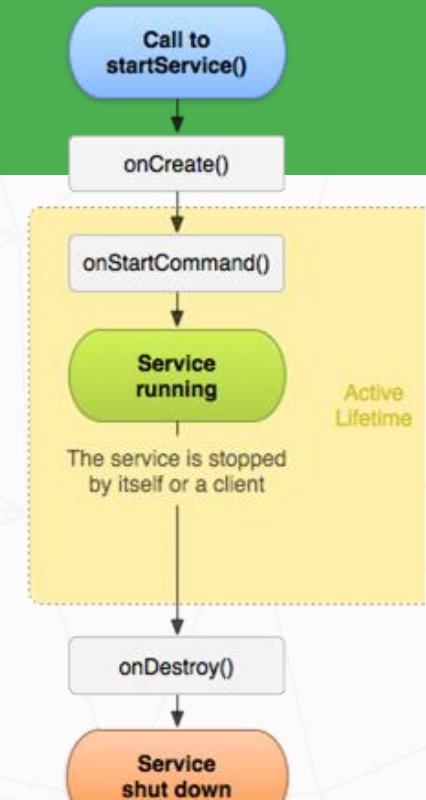
- Network transactions
- Play music
- Perform file I/O
- Interact with a content provider

# 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

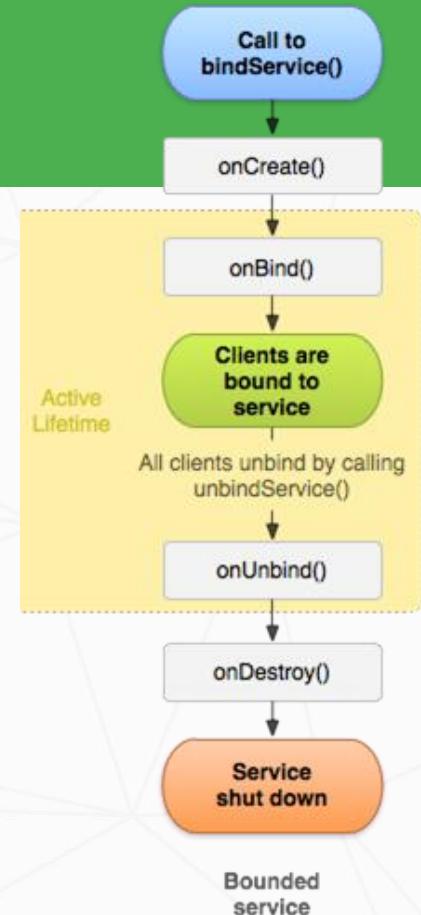
# Forms of services: started

- 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

# 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 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 Guide](#)
- [Running a Background Service](#)

# What's Next?

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



**END**



Android  
Developer  
Associate

Lesson 8



# 8.1 Notifications

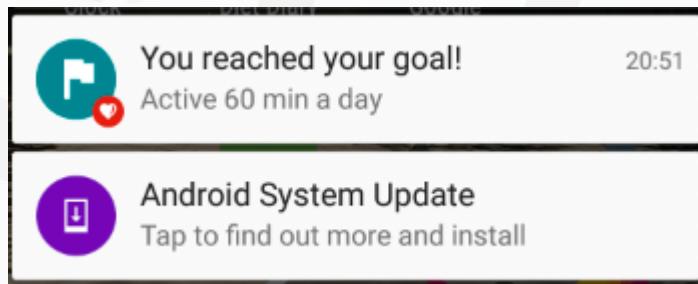
# Contents

- Notifications
- Creating Notifications
- Actions and Pending Intents
- Priority and Defaults
- Common Layouts
- Managing Notifications

# What Are Notifications?

# What is a notification?

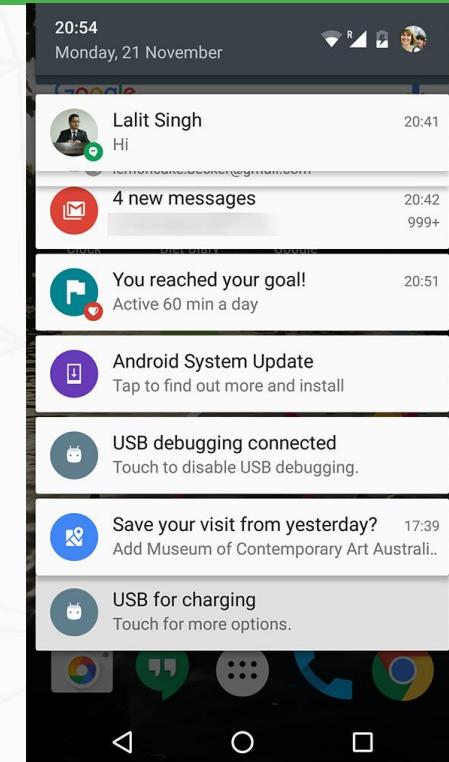
Message displayed to user outside regular app UI



- Small icon
- Title
- Detail text

# How are notifications used?

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



# Creating Notifications

# Two classes

## NotificationCompat.Builder

- Specifies UI and actions
- NotificationCompat.Builder.build() creates the Notification

## NotificationManager / NotificationManagerCompat

- NotificationManager.notify() issues the notification

# Define variables

```
private NotificationCompat.Builder mNotifyBuilder;  
  
private NotificationManager mNotifyManager;  
  
private static final int NOTIFICATION_ID = 0
```

# Instantiate NotificationManager

- In onCreate() of activity

```
mNotifyManager = (NotificationManager)  
    getSystemService(NOTIFICATION_SERVICE);
```

# Build and send notification

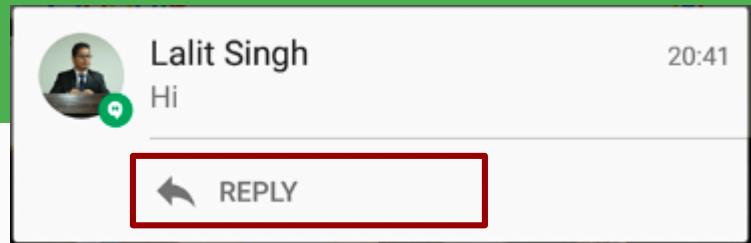
- Define notification and set required attributes

```
mNotifyBuilder = new NotificationCompat.Builder(this)
    .setContentTitle("You've been notified!")
    .setContentText("This is your notification text.")
    .setSmallIcon(R.drawable.ic_android_black_24dp);
```

```
Notification myNotification = mNotifyBuilder.build();
mNotifyManager.notify(NOTIFICATION_ID, myNotification);
```

# Actions and Pending Intents

# User actions



Notifications must be able to perform actions on behalf of your application

- Include specific actions inside the Notification UI
- Launch action when the notification is tapped
- OK for the action to just open an Activity in your app

# 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 application was yourself

# Pending Intents for Notifications

- Content intent is activated when the notification is tapped
  - setContent()
- Actions are choices shown to user
  - setAction()

# Methods to create a PendingIntent

Use method that corresponds to intent

- PendingIntent.getActivity()
- PendingIntent.getBroadcast()
- PendingIntent.getService()

# Methods all take four 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);
```

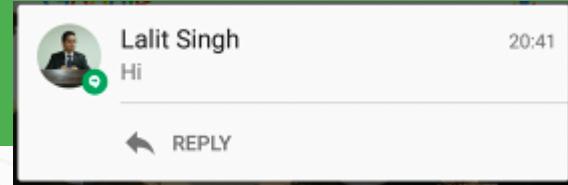
# Step 2: Create PendingIntent

```
PendingIntent notificationPendingIntent =  
    PendingIntent.getActivity(  
        this,  
        NOTIFICATION_ID,  
        notificationIntent,  
        PendingIntent.FLAG_UPDATE_CURRENT);
```

# Step 3: Add to notification builder

```
.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);
```

# Priority and Defaults

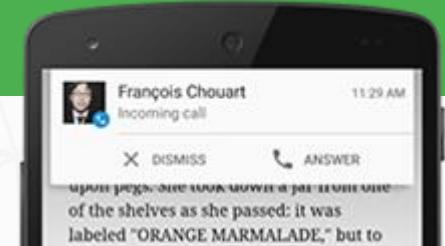
# Notification priority

- Determines how the system displays the notification with respect to other notifications
- Use `NotificationCompat.Builder.setPriority()`
  - pass in priority level

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

# 5 notification priority levels

- `PRIORITY_MIN` (-2) to `PRIORITY_MAX` (2)
- Priority above 0 triggers heads-up notification on top of current UI
- Used for important notifications such as phone calls
- Use lowest priority possible



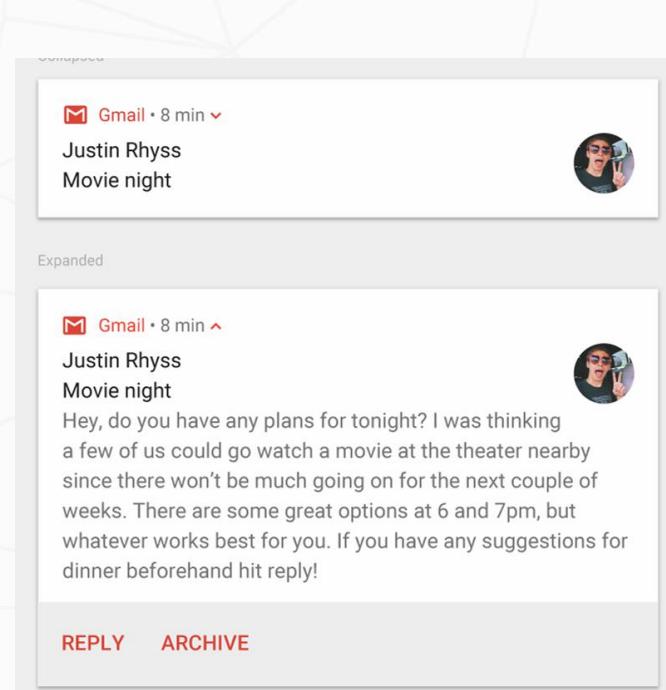
# Notification defaults

- Set sounds, vibration, and LED color pattern
- ```
.setDefaults(NotificationCompat.DEFAULT_ALL)
```

# Common Layouts

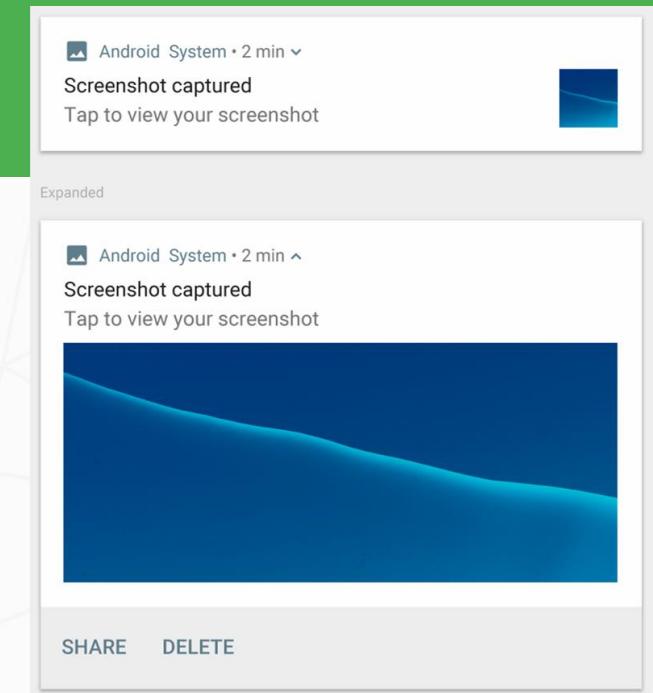
# Big text

- More text than will fit in standard view
- NotificationCompat.BigTextStyle



# Big image

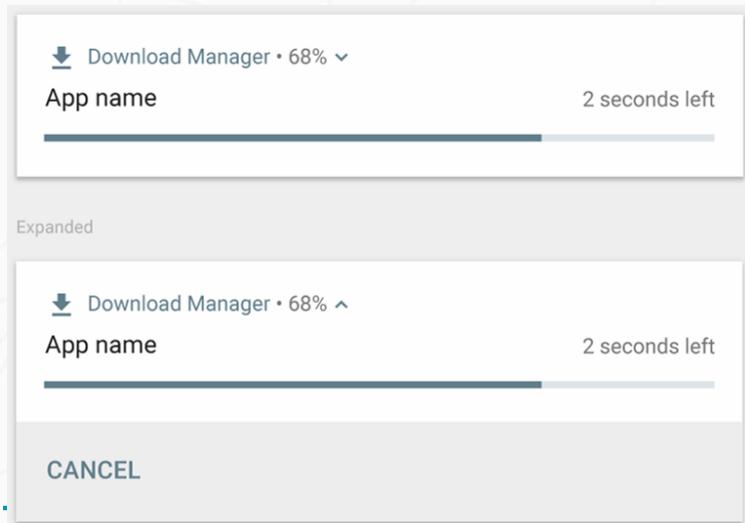
- Include image with notification



- [NotificationCompat.BigPictureStyle](#)

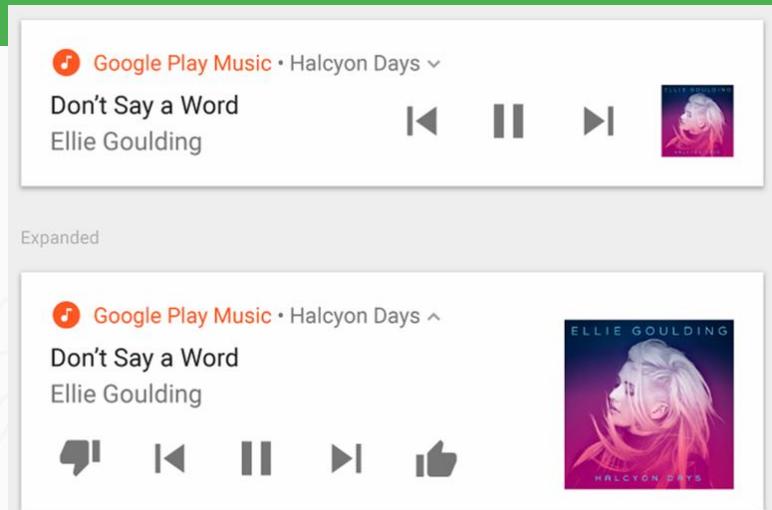
# Progress bar

- Progress bar for ongoing task that can be cancelled
- Not a style
- `.setProgress(100, incr, false,`



# Media

- Actions for controlling media such as music with image for album cover



- NotificationCompat.MediaStyle

# Setting styles

```
mNotifyBuilder  
    .setStyle(new NotificationCompat.BigPictureStyle()  
        .bigPicture(myBitmapImage)  
        .setBigContentTitle("Notification!"));
```

# Managing Notifications

# Updating notifications

- 1.Issue notification with updated parameters using builder
  - 2.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 created

# Canceling notifications

- Users can dismiss notifications
- User launches Content Intent with setAutoCancel() enabled
- 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
- [Notifications](#) design guide

# Learn more

- [Notifications](#)
- [Notification Design Guide](#)
- [NotificationCompat.Builder](#)
- [NotificationCompat.Style](#)

# What's Next?

- Concept Chapter: [8.1 C Notifications](#)
- Practical: [8.1 P Notifications](#)

# The End





Android  
Developer  
Associate

## Lesson 8



# 8.2 Alarm Manager

# 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

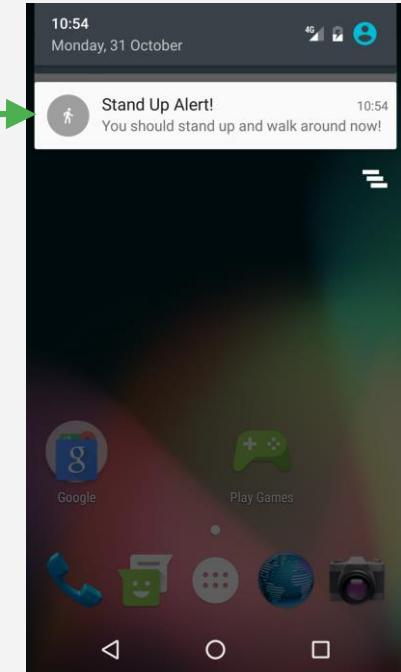
Stand Up Alarm On!

Activity creates  
notification  
manager and  
sets alarm

Alarm triggers and  
sends out intent  
  
App may be  
destroyed so....



BroadcastReceiver  
wakes up and  
delivers 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

|                       |                                           |                                           |
|-----------------------|-------------------------------------------|-------------------------------------------|
|                       | Elapsed Real Time (ERT)—since system boot | Real Time Clock (RTC)—time of day matters |
| 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 Alarm Manager
  - 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 C Scheduling Alarms](#)
- Practical: [8.2 P Alarm Manager](#)

**END**



Android  
Developer  
Associate

## Lesson 8



# 8.3 Transferring Data Efficiently & Job Scheduler

# Contents

- Transferring Data Efficiently
- Job Scheduler
  - JobService
  - JobInfo
  - JobScheduler

# Transferring Data Efficiently

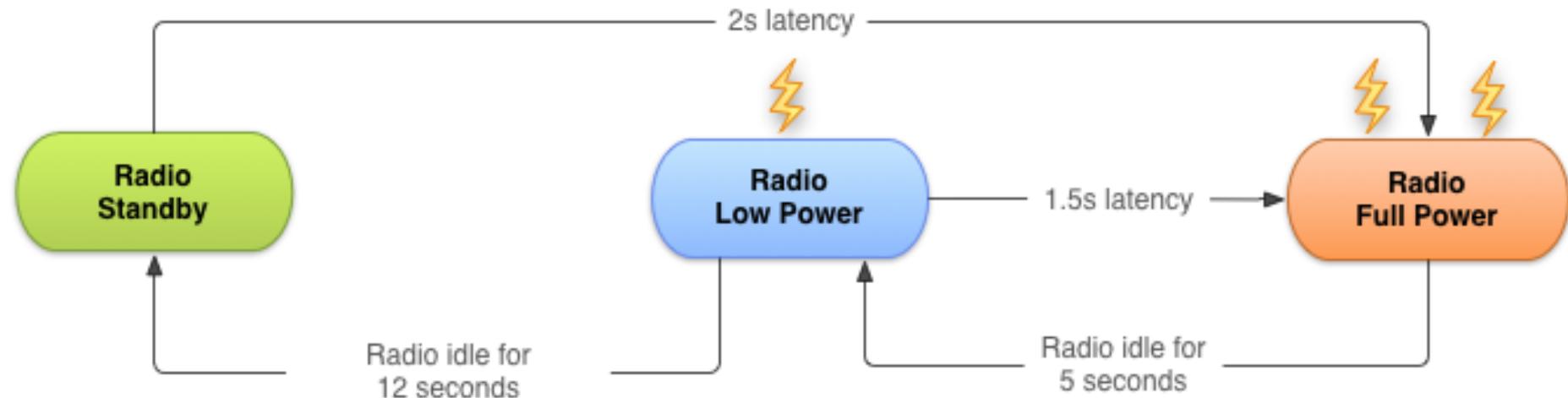
# Transferring data uses resources

- 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

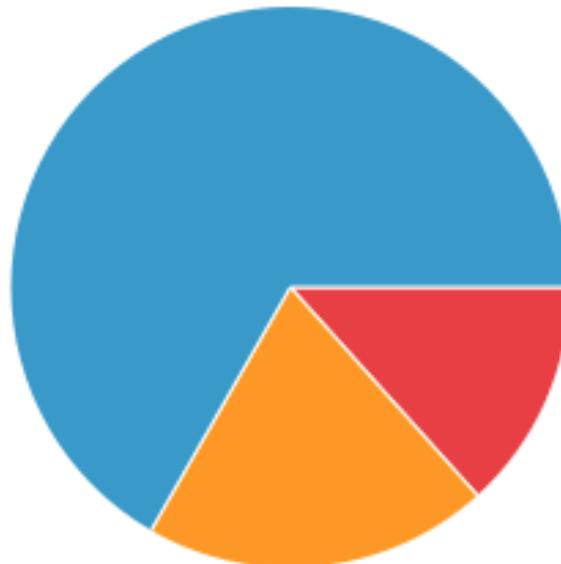
- For a typical 3G device, every data transfer session, the radio draws energy for almost 20 seconds
- Send data for 1s every 18s—radio mostly on full power
- Send data in bundles of 3s—radio mostly idle
- Bundle your data transfers

# Bundled vs. unbundled

Unbundled Transfers



Bundled Transfers



- High Power
- Low Power
- Idle

# 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

- WiFi 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

# 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+ (**no 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

# JobService

- JobService subclass
- 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 off-loaded
- 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 WiFi, 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
- NETWORK\_TYPE\_NOT\_ROAMING
- NETWORK\_TYPE\_UNMETERED

# setBackOffCriteria()

setBackoffCriteria(long initialBackoffMillis, int backoffPolicy)

- How soon to reschedule if task fails
- Arg 1: Initial time to wait after the task fails—default 30s
- Arg 2: How long to wait subsequently
  - BACKOFF\_POLICY\_LINEAR
  - BACKOFF\_POLICY\_EXPONENTIAL
  - Default {30 seconds, Exponential}

# 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

# 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 C Transferring Data Efficiently](#)
- Practical: [8.3 P Job Scheduler](#)

**END**

