



Introduction to Android

Contents



- Android ecosystem
- Challenges of Android app development
- Android Versions
 - and their impact of the app development
- App fundamentals / Android Building Blocks

These slides are partially based on the material that Google provides for the course
Android Developer Fundamentals

<https://developer.android.com/courses/fundamentals-training/overview-v2>



Android Ecosystem

What is Android?

A Mobile operating system (OS) based on a ***modified version*** of the ***Linux kernel*** and other open source software

- note that it **is not** Linux!
- **No native windowing system** (e.g., X in Unix/Linux or DWM in Windows)
- Does not include the **full set of Linux utilities**
- Because Android **doesn't include a graphical X server** or all the standard GNU libraries, ***it can't simply run Linux applications***
- You have to run applications written specifically for Android
- ***You can't run Android apps*** on typical Linux distributions

What is Android?

A Mobile operating system (OS) based on a ***modified version*** of the ***Linux kernel*** and other open source software

- note that it **is not Linux!**
- **No native windowing system** (e.g., X in Unix/Linux or DWM in Windows)
- Does not include the **full set of Linux utilities**
- Because Android **doesn't include a graphical X server** or all the standard GNU libraries, ***it can't simply run Linux applications***
- You have to run applications written specifically for Android
- ***You can't run Android apps*** on typical Linux distributions

Android UI

- User Interface for touch screens
- Used on the majority of all smartphones
- Powers devices such as watches, TVs, and cars



[Android Wear ›](#)

[Phones and Tablets ›](#)

[Android TV ›](#)

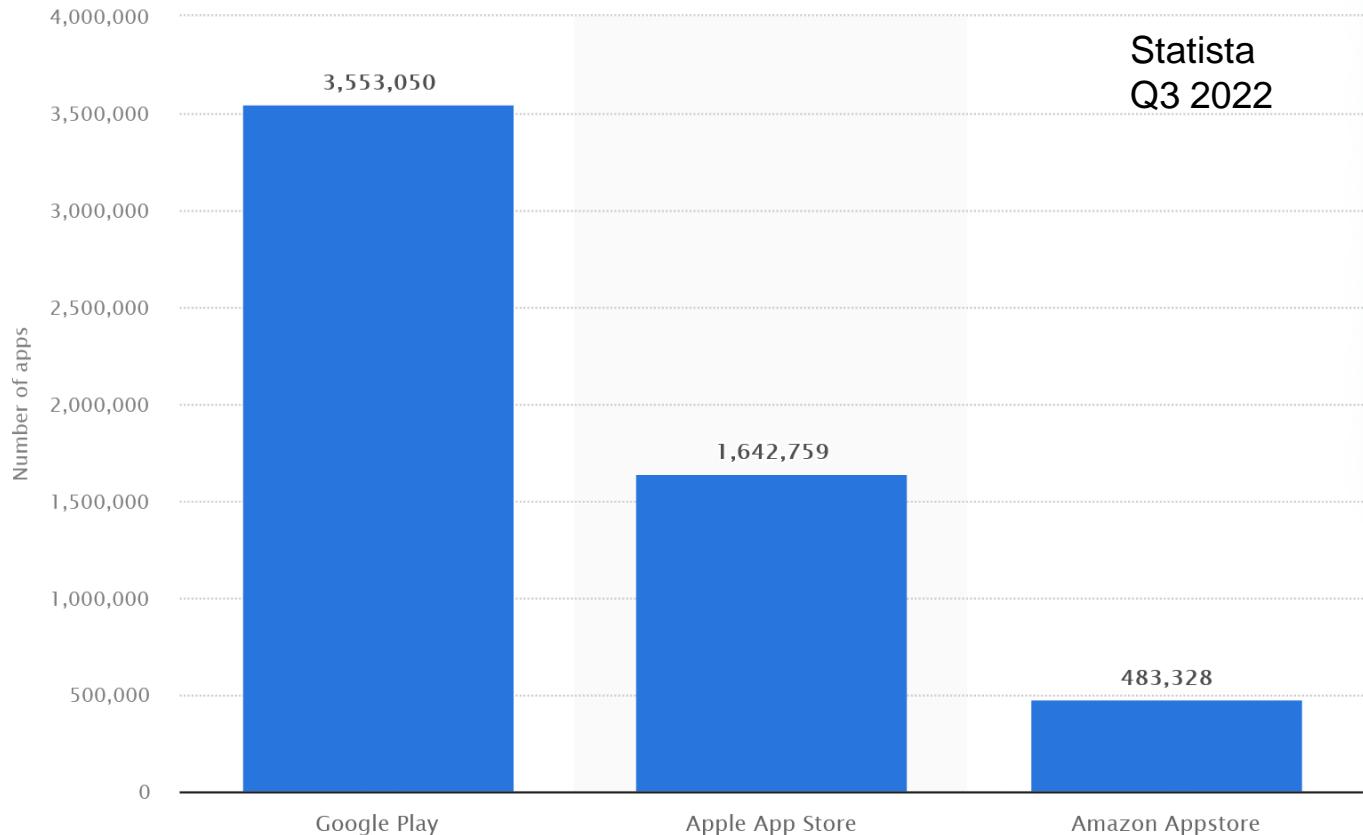
[Android Auto ›](#)

Android Apps

Over 3.5 Million
Android
apps in



Google Play



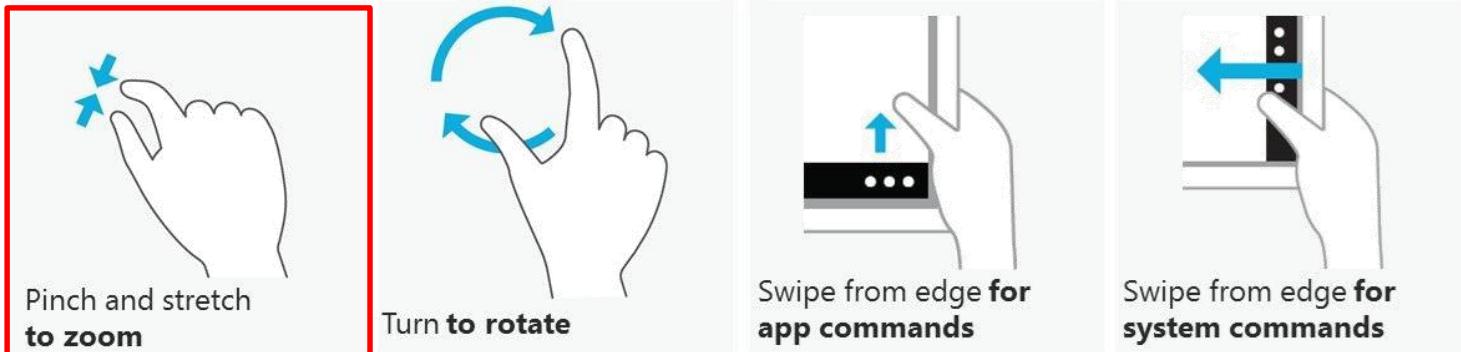
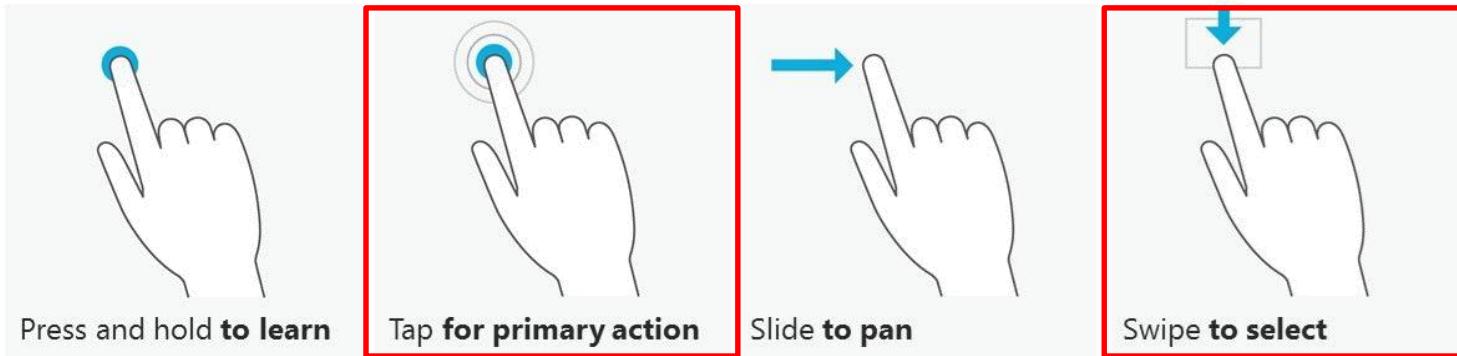
Android Customization

- Highly customizable for devices / by vendors
- Open source



User interaction & Gestures

- Touch gestures: swiping, tapping, pinching



Virtual Keyboard

- Virtual keyboard for characters, numbers, and emoji



Virtual Keyboard

- Virtual keyboard for characters, numbers, and emoji

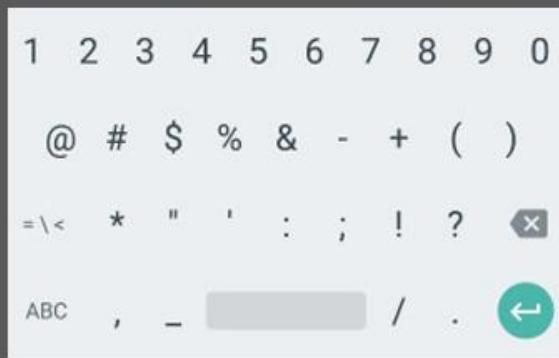


Virtual Keyboard

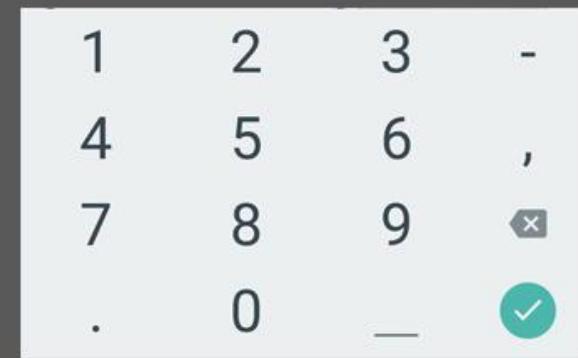
- Virtual keyboard for characters, numbers, and emoji



Default
Alphabet



Default
Numeric



Keypad
Numeric Only

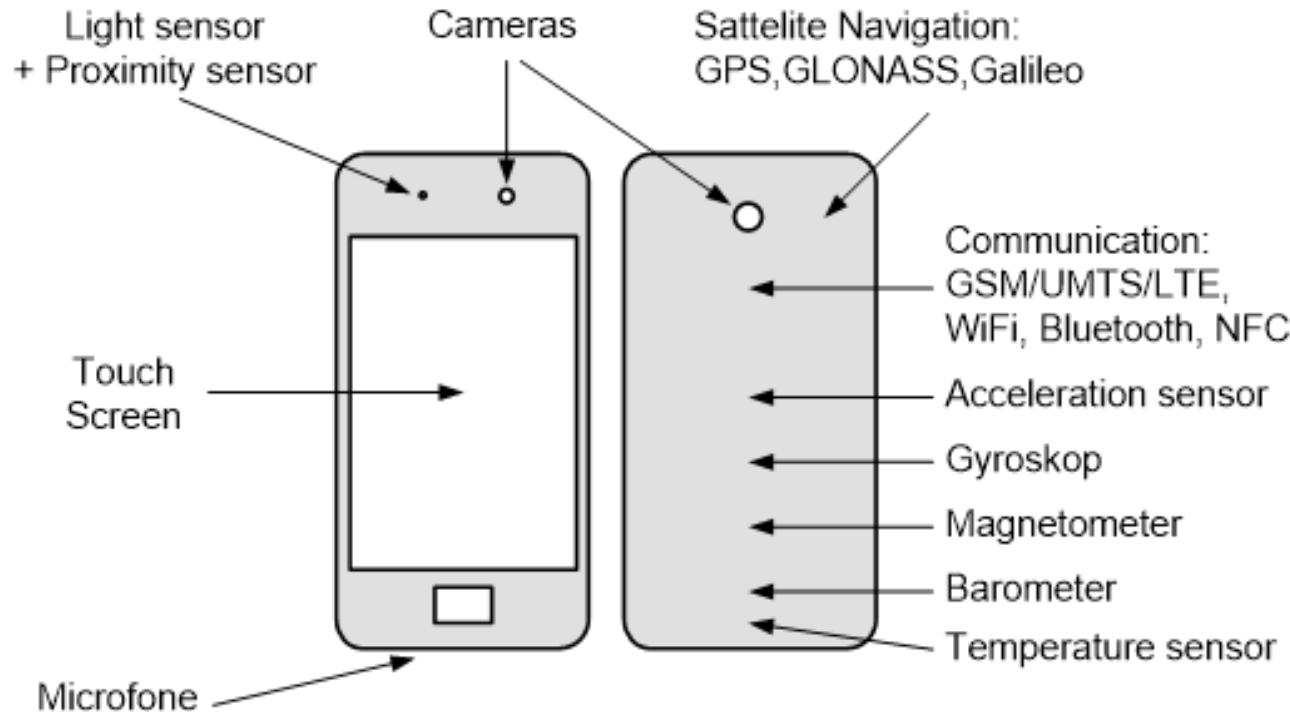
Devices

- Support for Bluetooth, USB controllers and peripherals



Android and sensors

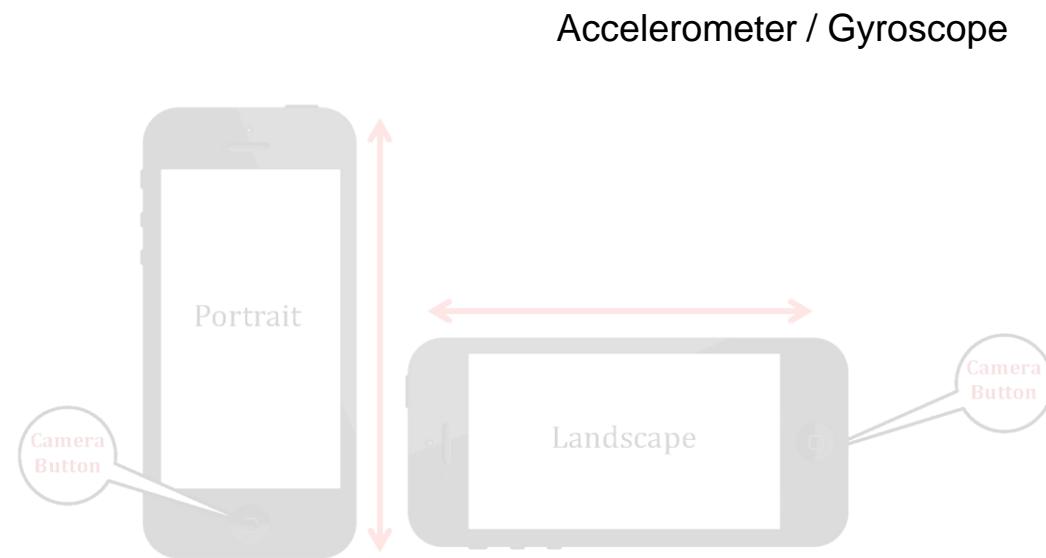
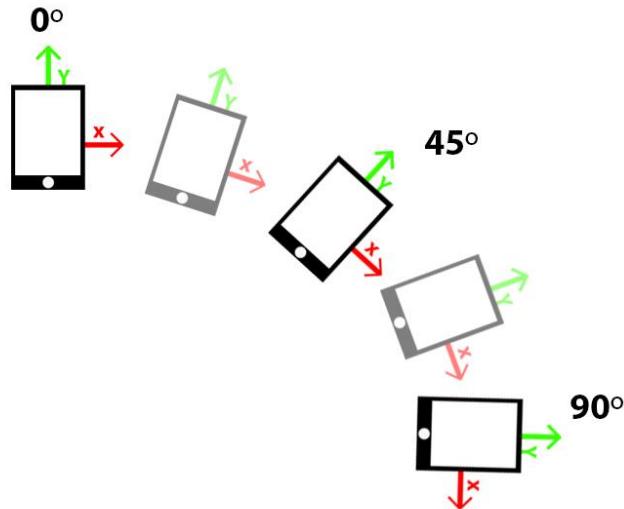
Sensors can discover user action and respond



Android and sensors

Sensors can discover user action and respond

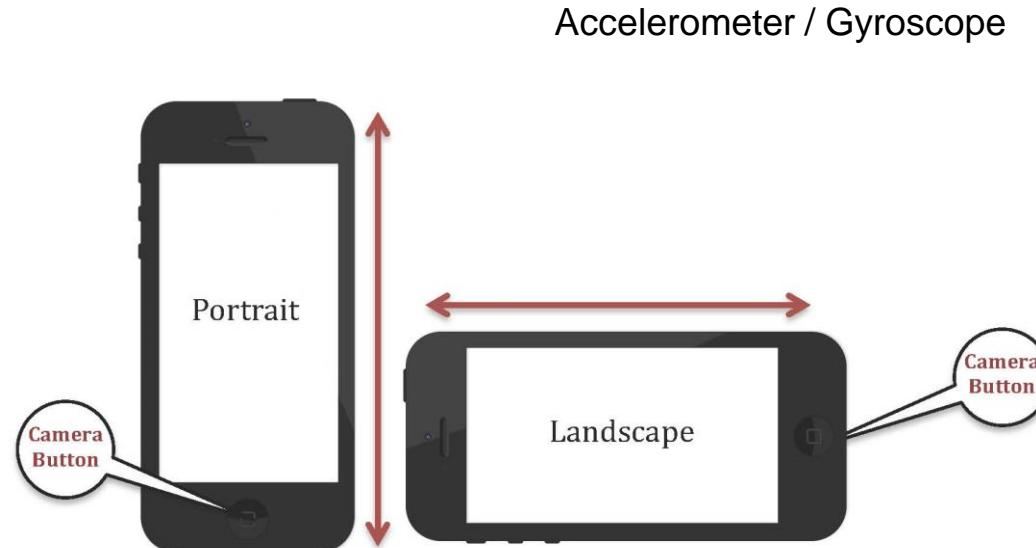
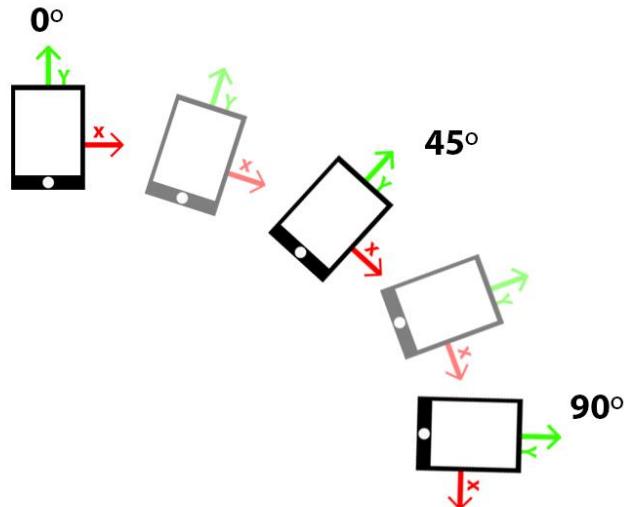
- Device contents rotate as needed



Android and sensors

Sensors can discover user action and respond

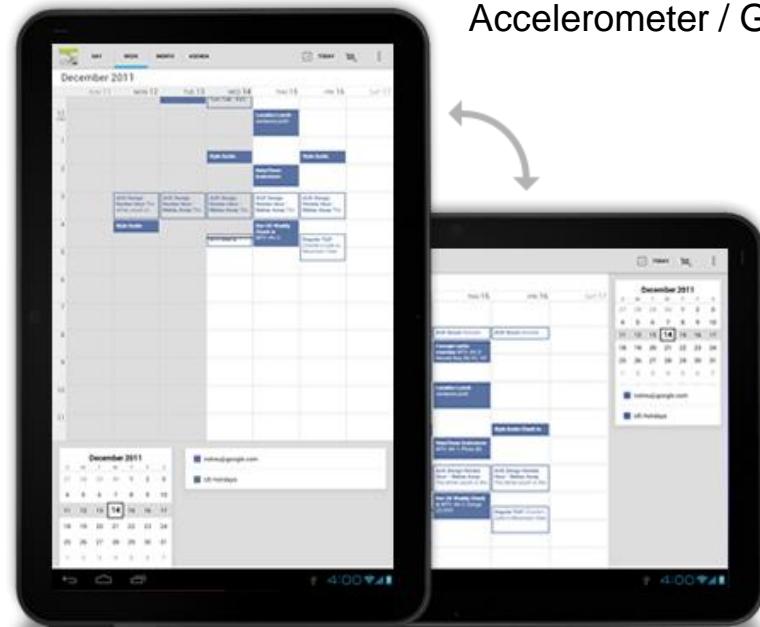
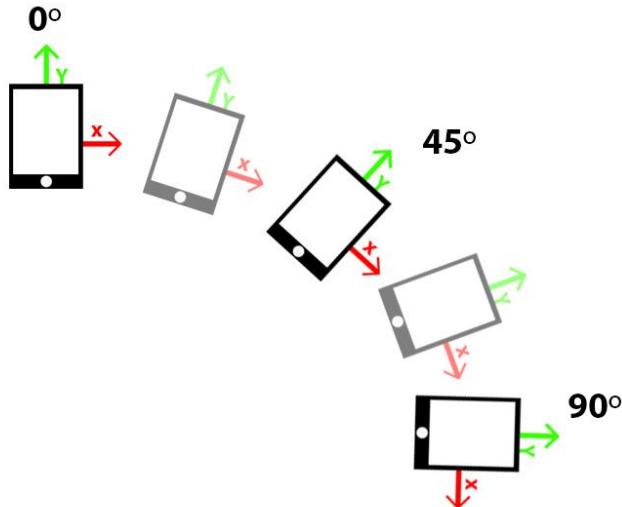
- Device contents rotate as needed



Android and sensors

Sensors can discover user action and respond

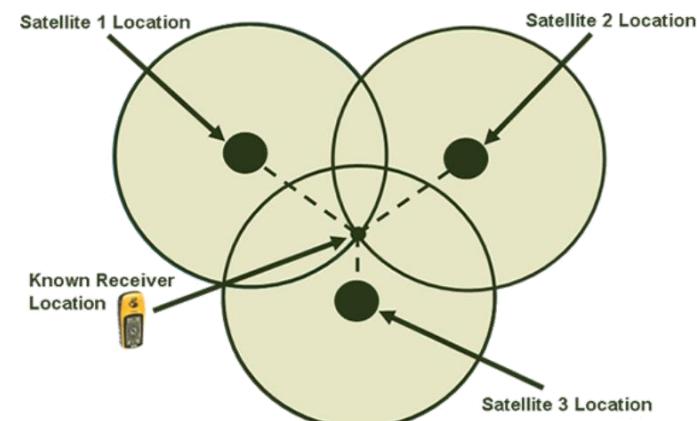
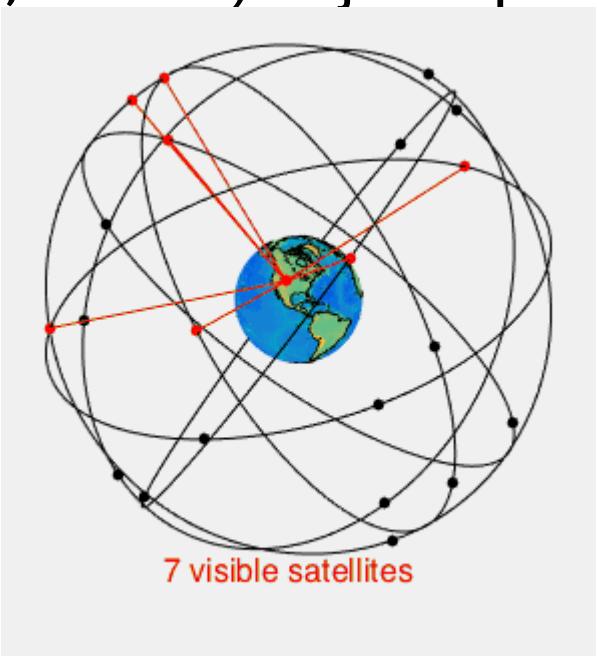
- Device contents rotate as needed



Android and sensors

Sensors can discover user action and respond

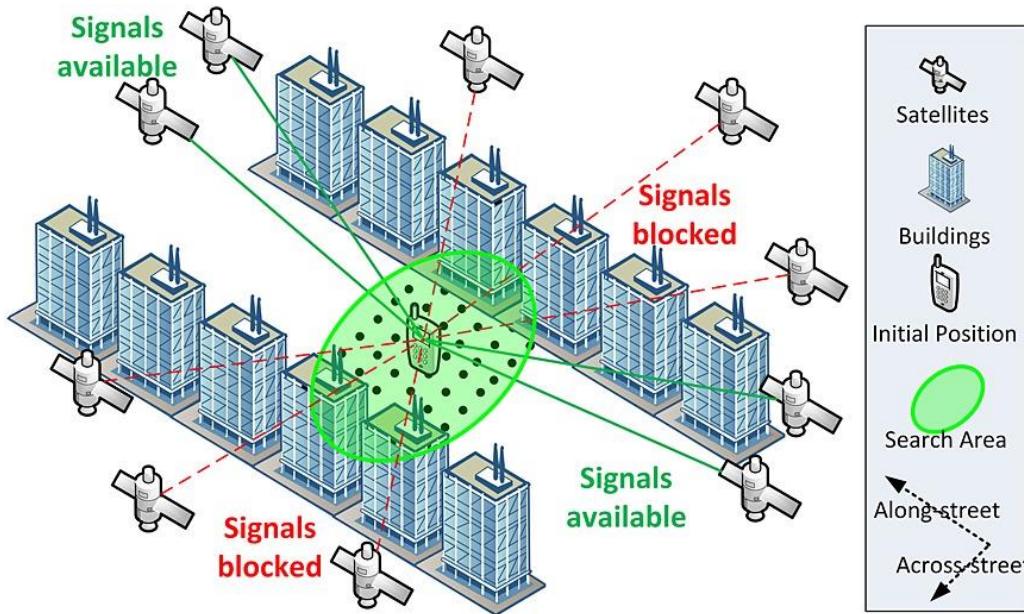
- GPS (GLONASS, Galileo) adjusts position on map



Android and sensors

Sensors can discover user action and respond

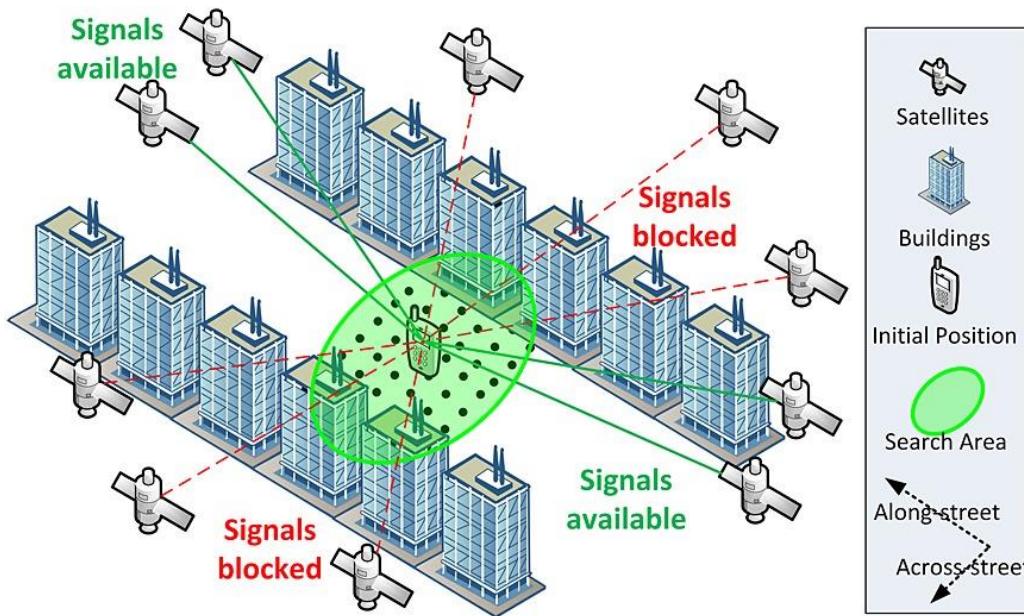
- GPS (GLONASS, Galileo) adjusts position on map



Android and sensors

Sensors can discover user action and respond

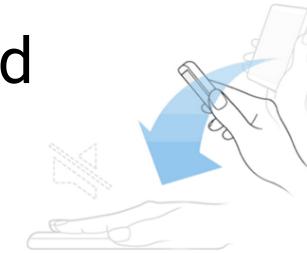
- GPS (GLONASS, Galileo) adjusts position on map



Android and sensors

Sensors can discover user action and respond

- Tilting steers a virtual car or controls a physical toy

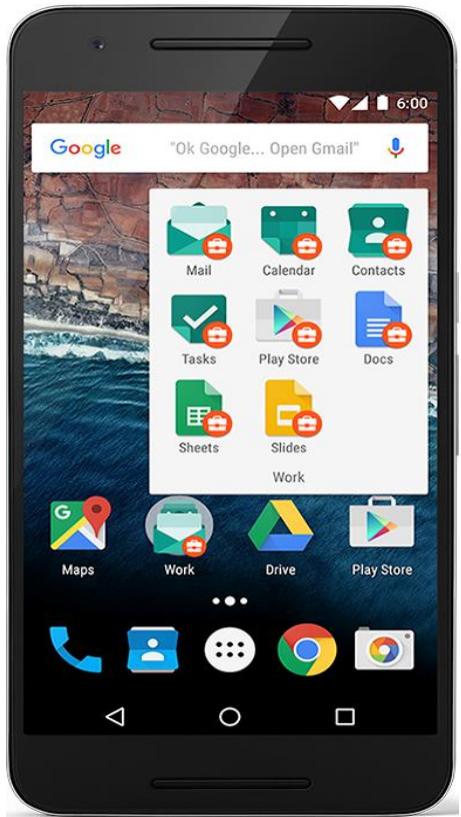


Mute the incoming call by covering or flipping the phone upside down.

- Moving too fast disables game interactions



Android home screen



- Launcher icons for apps
- Self-updating widgets for live content

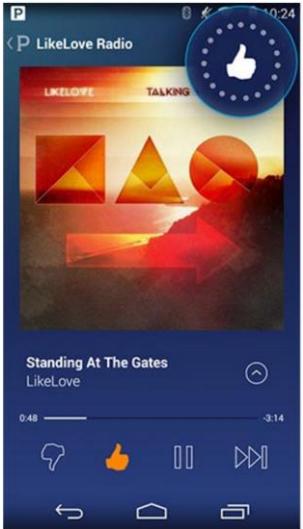


Android home screen

- Can be multiple pages
- Folders to organize apps



Android app examples



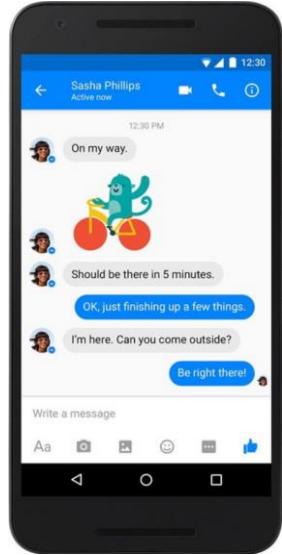
Pandora

Streaming Music,
Radio & Podcasts



Pokemon GO

Games



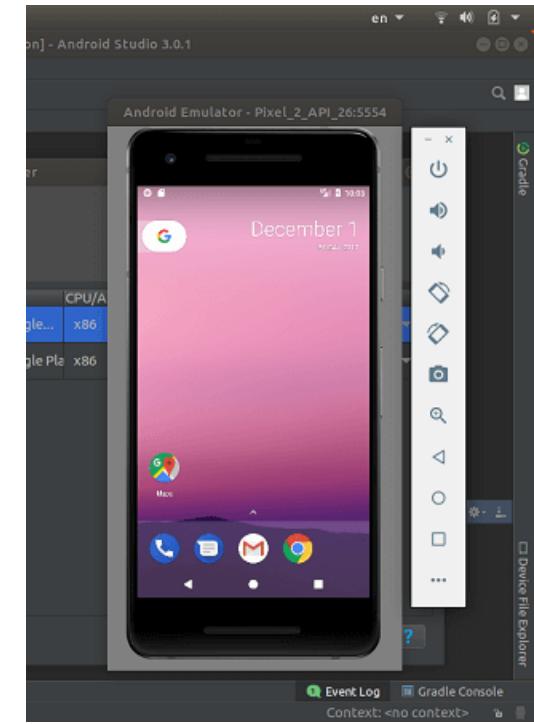
Facebook
Messenger

Android Software Developer Kit (SDK)

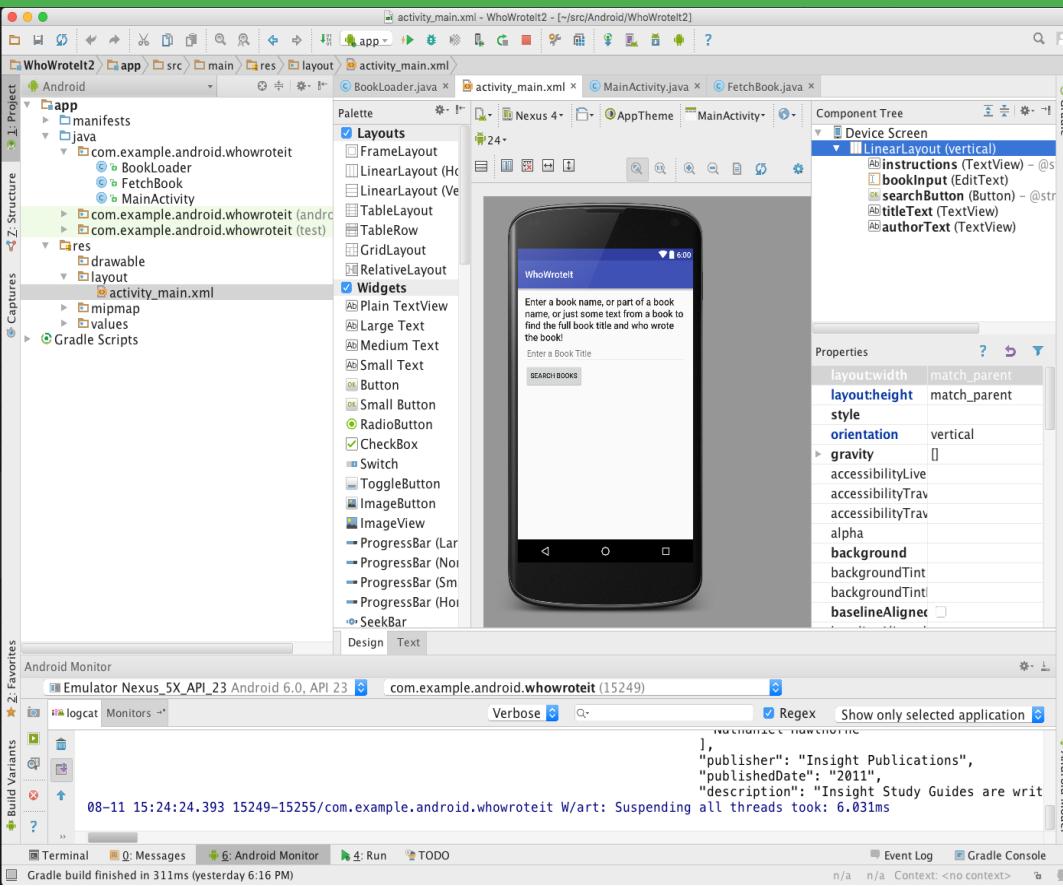
- Development tools (debugger, monitors, editors)
- Libraries (maps, wearables)
- Virtual devices (emulators)
- Documentation (developers.android.com)
- Sample code

Android Software Developer Kit (SDK)

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



Android Studio



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

Google Play store

Publish apps through Google Play store:

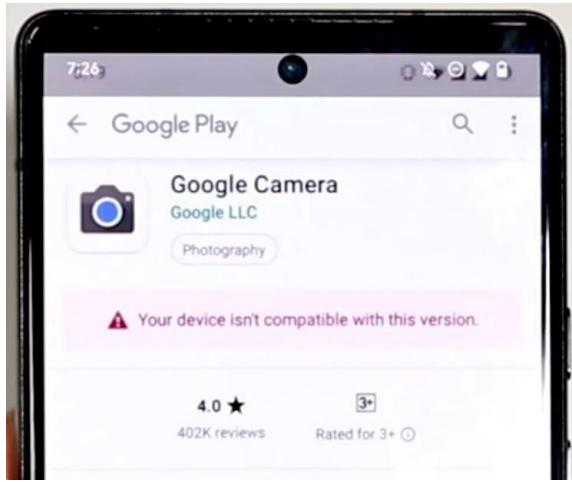
- Official app store for Android
- Digital distribution service operated by Google



Google Play store

Publish apps through Google Play store:

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



App Development

What is an Android app?

- One or more interactive screens
- Written using Java or Kotlin (logic) and XML (UI)
- 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

- Android runs on billions of handheld devices around the world and supports
 - various form factors including wearable devices and televisions
 - various themes



challenge, as a developer, is to **design UI elements that work on all devices**



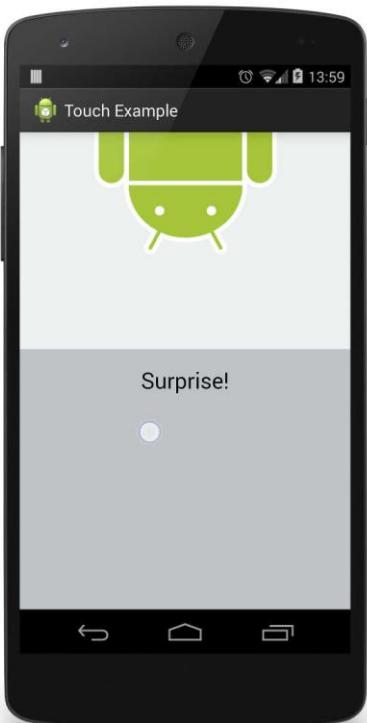
Screen Property - Density



But **this** on a device with an higher pixel density



4k (3840×2160 pixels) Full HD (1920×1080 pixels)
Density e.g., 700ppi Density e.g., 350ppi



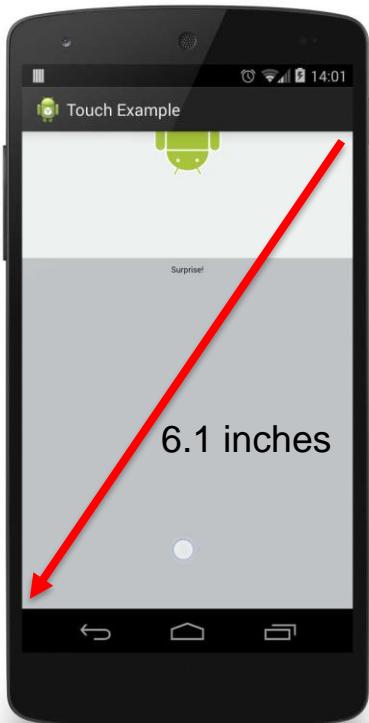
If you use **fixed measures** in your app (e.g., in pixel), you can obtain **this result** on the device you are working



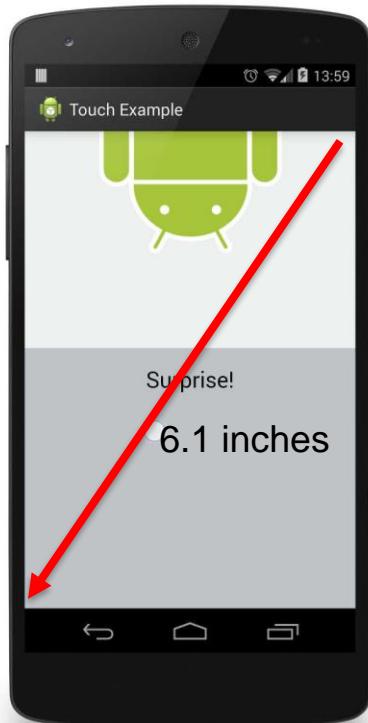
Screen Property - Density



But **this** on a device with an higher pixel density



4k (3840×2160 pixels)
Density e.g., 700ppi



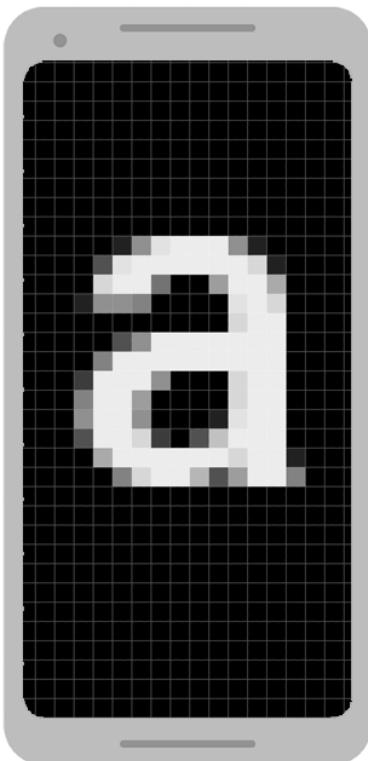
Full HD (1920×1080 pixels)
Density e.g., 350ppi

If you use **fixed measures** in your app (e.g., in pixel), you can obtain **this result** on the device you are working

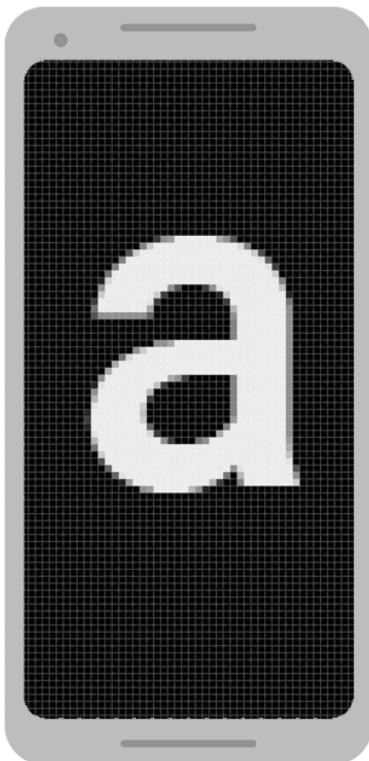


Screen Property - Density

HD (1280x720 pixels)



4k (3840x2160 pixels)



Example of two screens of the **same size** may have a **different number of pixels**

To preserve the visible size of your UI on screens with different densities, you **must design your UI using *density-independent pixels (dp)*** as your unit of measurement.

dp is a **virtual pixel** unit that's **roughly equal to one pixel on a medium-density screen** (160dpi; the "baseline" density)

Android translates this value to the appropriate number of real pixels for each other density

Density-independent pixels



Screen Property - Qualifiers

Qualifiers (standard names) for different pixel densities



Screen Property - Qualifiers

Qualifiers (standard names) for screen

The screenshot shows the Android Studio interface with the 'Configure Image Asset' dialog open. The dialog is titled 'Configure Image Asset' and shows 'Launcher Icons' settings. The 'Name' field is set to 'ic_launcher', 'Asset Type' is 'Image', and the 'Source Asset' is a green Android icon. Below this, five icons are displayed for different density levels: xxhdpi, xxhdpi, xhdpi, hdpi, and mdpi. A warning message at the bottom states: '⚠ An icon with the same name already exists and will be overwritten.' At the bottom of the dialog are 'Previous', 'Next', 'Cancel', and 'Finish' buttons.

1inch

1inch

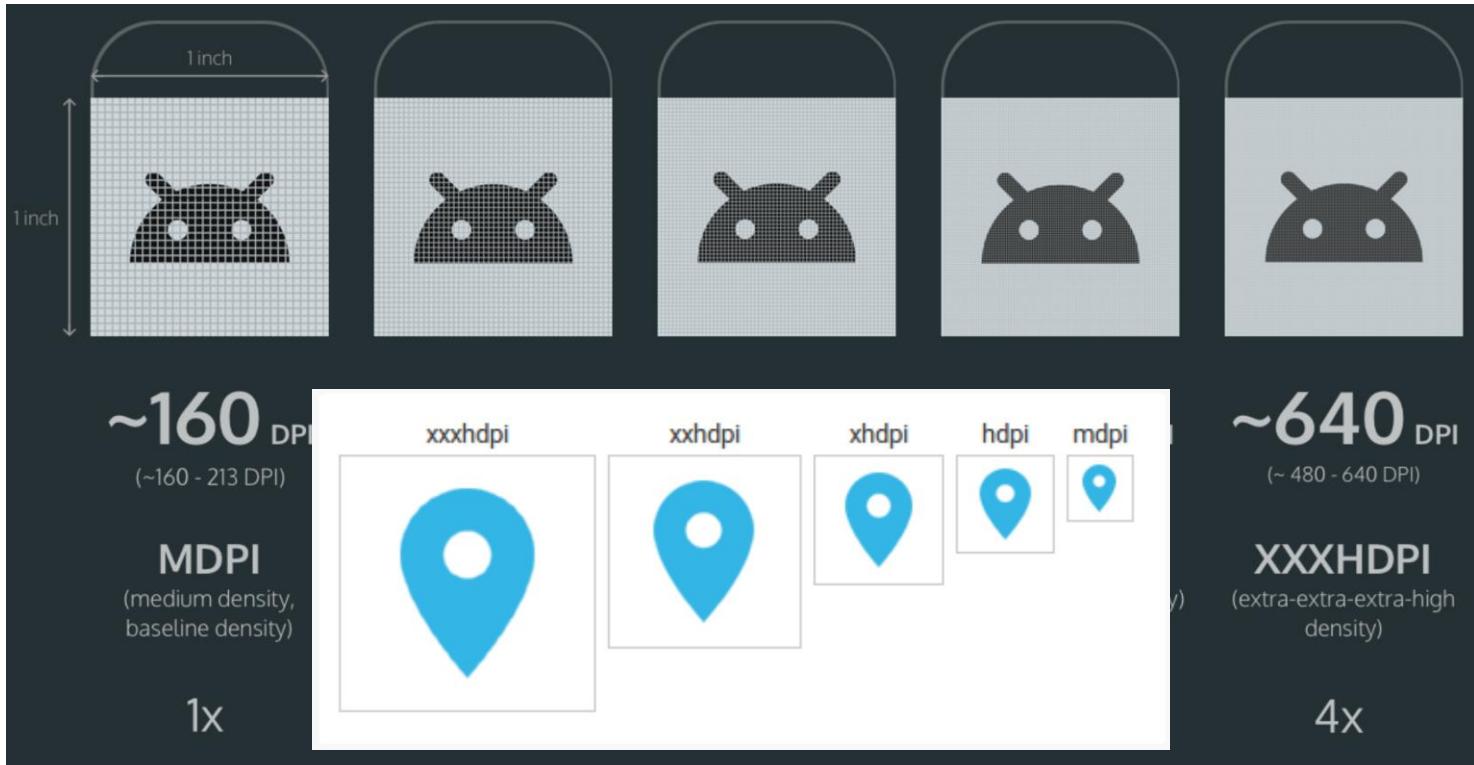
~160 DPI
(~160 - 213 DPI)

MDPI
(medium density,
baseline density)

1x

Screen Property - Qualifiers

Qualifiers (standard names) for different pixel densities

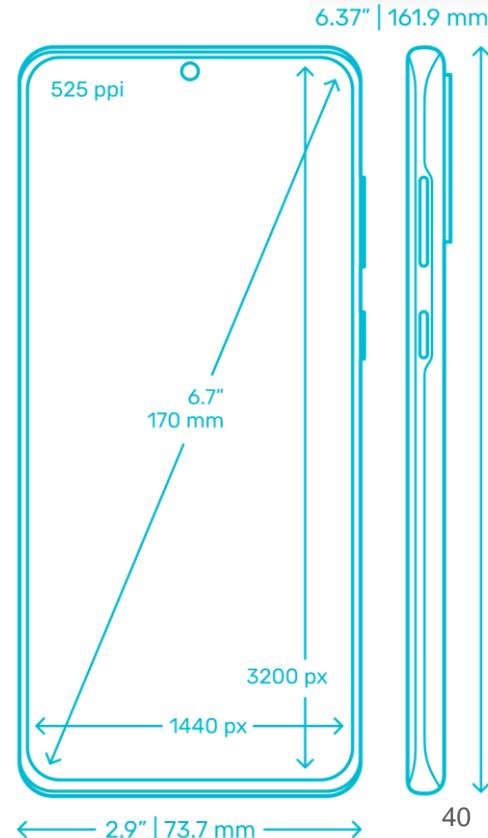


Screen Property - Qualifiers

Samsung Galaxy S20+

Qualifiers (standard names) for different pixel densities

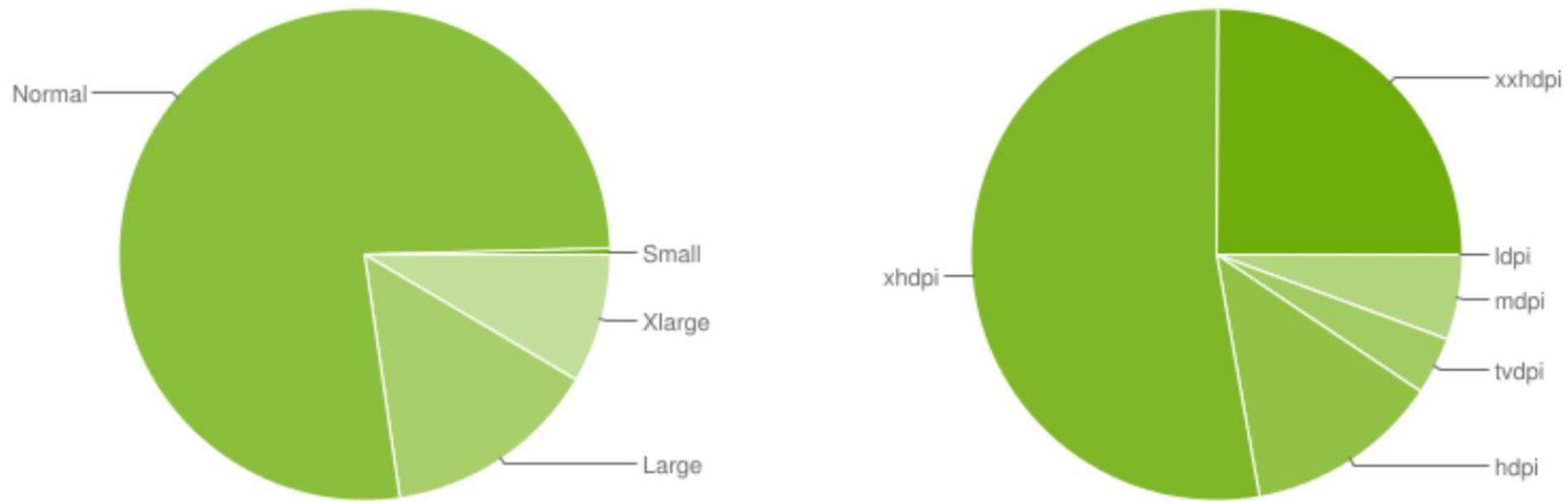
Density qualifier	Description
ldpi	Resources for low-density (<i>ldpi</i>) screens (~120dpi).
mdpi	Resources for medium-density (<i>mdpi</i>) screens (~160dpi). (This is the baseline density.)
hdpi	Resources for high-density (<i>hdpi</i>) screens (~240dpi).
xhdpi	Resources for extra-high-density (<i>xhdpi</i>) screens (~320dpi).
xxhdpi	Resources for extra-extra-high-density (<i>xxhdpi</i>) screens (~480dpi).
xxxhdpi	Resources for extra-extra-extra-high-density (<i>xxxhdpi</i>) uses (~640dpi).
nodpi	Resources for all densities. These are density-independent resources. The system does not scale resources tagged with this qualifier, regardless of the current screen's density.
tvdpi	Resources for screens somewhere between mdpi and hdpi; approximately 213dpi. This is not considered a "primary" density group. It is mostly intended for televisions and most apps shouldn't need it—providing mdpi and hdpi resources is sufficient for most apps and the system will scale them as appropriate. If you find it necessary to provide tvdpi resources, you should size them at a factor of 1.33*mdpi. For example, a 100px x 100px image for mdpi screens should be 133px x 133px for tvdpi.



Screen Property – Market Share 2023

	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Total
Small					0.4%		0.4%
Normal		0.1%	0.3%	7.9%	45.4%	23.2%	76.9%
Large		1.2%	3.4%	1.1%	6.8%	1.7%	14.2%
Xlarge		4.3%	0.1%	3.8%	0.3%		8.5%
Total	0.0%	5.6%	3.8%	12.8%	52.9%	24.9%	

Screen Property – Market Share 2023



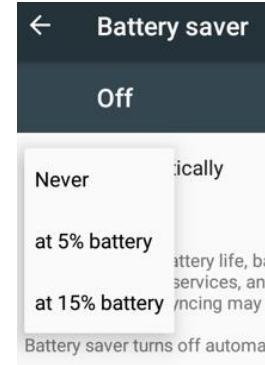
<https://developer.android.com/training/multiscreen/screensizes.html>

<https://developer.android.com/training/multiscreen/screendensities>

<https://developer.android.com/about/dashboards/index.html>

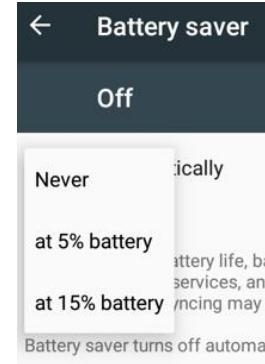
Challenges of Android development

- **Performance:** make your **apps responsive and smooth**. Main aspects:
 - how **fast** it runs
 - how **easily** it connects to the **network**
 - how well it **manages battery** and **memory usage**
- Performances are affected by various factors
 - **battery level** (low battery => low performance)
 - **multimedia content** (HD content reduce overall performances)
 - **internet access** (concurrent streaming reduce network performances)
- Some app **features** cause **performance problems** for users
 - e.g., to save the user's **battery power**, enable **background services** only when they are necessary



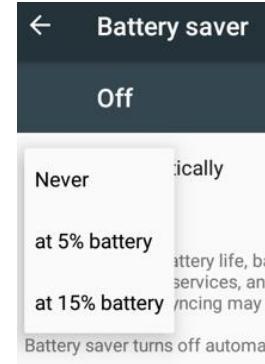
Challenges of Android development

- **Performance:** make your **apps responsive and smooth**. Main aspects:
 - how **fast** it runs
 - how **easily** it connects to the **network**
 - how well it **manages battery** and **memory usage**
- **Performances are affected** by various **factors**
 - **battery level** (low battery => low performance)
 - **multimedia content** (HD content reduce overall performances)
 - **internet access** (concurrent streaming reduce network performances)
- Some app **features** cause **performance problems** for users
 - e.g., to save the user's **battery power**, enable **background services** only when they are necessary



Challenges of Android development

- **Performance:** make your **apps responsive and smooth**. Main aspects:
 - how **fast** it runs
 - how **easily** it connects to the network
 - how well it manages **battery** and **memory usage**
- **Performances are affected** by various **factors**
 - **battery level** (low battery => low performance)
 - **multimedia content** (HD content reduce overall performances)
 - **internet access** (concurrent streaming reduce network performances)
- Some **app features** cause **performance problems** for users
 - e.g., to save the user's **battery** power, enable **background services** only when they are necessary



Challenges of Android development

Security: keep source code and user data safe

take precautions to make the code, and the user' data as secure as possible:

- the compiler provided with Android Studio (R8) could be configured to
 - **detect and remove** unused classes, fields, methods, and attributes
 - **remove** unused resources from the packaged app
 - **obfuscate the code**: shortens the name of classes and members
 - **optimize the code**: e.g., removing else {} branch for a given if/else statement is never taken
- **protect critical user information** such as logins and passwords
- **secure your communication channel** to protect data in transit across the internet
- **secure data at rest** on the device

Challenges of Android development

Security: keep source code and user data safe

take precautions to make the code, and the user' data as secure as possible:

- the compiler provided with Android Studio (R8) could be configured to
 - **detect and remove** unused classes, fields, methods, and attributes
 - **remove** unused resources from the packaged app
 - **obfuscate the code**: shortens the name of classes and members
 - **optimize the code**: e.g., removing else {} branch for a given if/else statement is never taken
- **protect critical user information** such as logins and passwords
- **secure your communication channel** to protect data in transit across the internet
- **secure data** at rest on the device

Challenges of Android development

Security: keep source code and user data safe

take precautions to make the code, and the user' data as secure as possible:

- ~~the compiler provided with Android Studio (R8) could be configured to~~

App security best practices

By making your app more secure, you help preserve user trust and device integrity.

This page presents several best practices that have a significant, positive impact on your app's security.

<https://developer.android.com/topic/security/best-practices>

- **protect critical user information** such as logins and passwords
- **secure your communication channel** to protect data in transit across the internet
- **secure data** at rest on the device

Challenges of Android development

Compatibility: **run well** on ***older platform*** versions

NOT focus only on the **most recent** Android version

NOT all users may **have upgraded** (or may be able to upgrade) their devices

Marketing: understand the market and your users

hint: It doesn't have to be expensive, but it can be

Challenges of Android development

Compatibility: **run well** on ***older platform*** versions

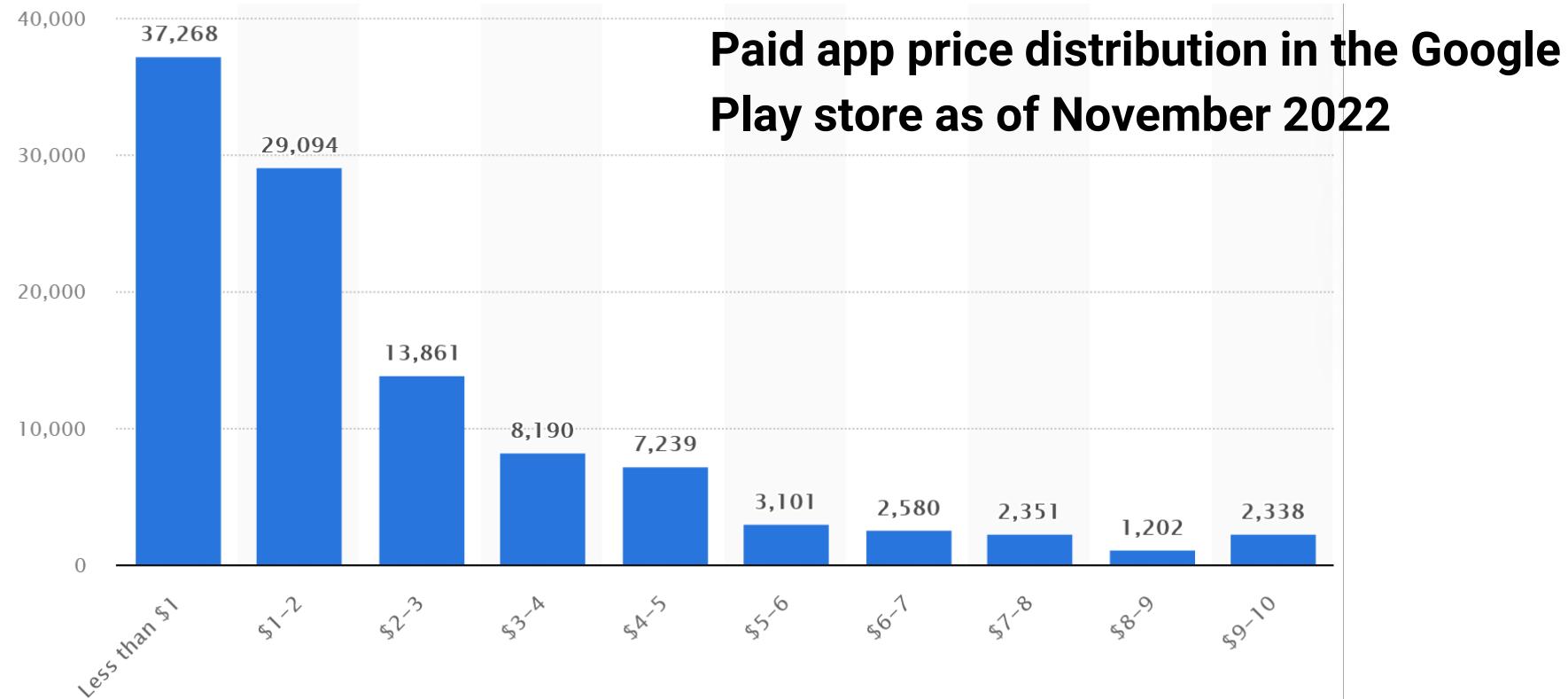
NOT focus only on the **most recent** Android version

NOT all users may **have upgraded** (or may be able to upgrade) their devices

Marketing: understand the market and your users

hint: It doesn't have to be expensive, but it can be

Challenges of Android development



Android Versions

Google provides several major **incremental upgrades** to the Android operating system in the last 15 years

using *confectionery-themed* names up to Android 9



Name	Internal codename ^[11]	Version number(s)	API level	Initial stable release date	Latest security patch date ^[16]	Latest Google Play Services version ^[17] (release date)
Android 1.0	—	1.0	1	September 23, 2008		
Android 1.1	Petit Four	1.1	2	February 9, 2009		
Android Cupcake	Cupcake	1.5	3	April 27, 2009		
Android Donut	Donut	1.6	4	September 15, 2009	—	
Android Eclair	Eclair	2.0	5	October 27, 2009		
		2.0.1	6	December 3, 2009		
		2.1	7	January 11, 2010 ^[18]		
Android Froyo	Froyo	2.2 – 2.2.3	8	May 20, 2010	3.2.25 (October 2014)	
Android Gingerbread	Gingerbread	2.3 – 2.3.2	9	December 6, 2010	—	
		2.3.3 – 2.3.7	10	February 9, 2011		
Android Honeycomb	Honeycomb	3.0	11	February 22, 2011	10.0.84 (November 2016)	
		3.1	12	May 10, 2011		
		3.2 – 3.2.6	13	July 15, 2011		
Android Ice Cream Sandwich	Ice Cream Sandwich	4.0 – 4.0.2	14	October 18, 2011		
		4.0.3 – 4.0.4	15	December 16, 2011	14.8.49 (February 2019)	
Android Jelly Bean	Jelly Bean	4.1 – 4.1.2	16	July 9, 2012		
		4.2 – 4.2.2	17	November 13, 2012		
		4.3 – 4.3.1	18	July 24, 2013	21.33.56 (September 2021)	
Android KitKat	Key Lime Pie	4.4 – 4.4.4	19	October 31, 2013	October 2017	
		4.4W – 4.4W.2	20	June 25, 2014	?	

Api Level

Android Studio **allows to choose the minum API level to support**

balance between

- going **low enough to support as many devices as possible**
- not going **too low that you lose features** your app needs to run

Android Versions

Android Lollipop	Lemon Meringue Pie	5.0 – 5.0.2	21	November 4, 2014 ^[20]	November 2017	<p>The timeline diagram illustrates the progression of Android versions. It starts with Lollipop (5.0) at the top, followed by Marshmallow (6.0), Nougat (7.0), Oreo (8.0), and Pie (9.0). Below these are Android 10 (Quince Tart), 11 (Red Velvet Cake), 12 (Snow Cone), 12L (Snow Cone v2), 13 (Tiramisu), 14 (Upside Down Cake), and ends with Android 15 (Vanilla Ice Cream). The timeline shows a general downward trend from left to right, indicating the chronological order of releases. The versions are color-coded according to the legend: Old version (red), Older version, still maintained (yellow), Latest version (green), and Latest preview version (orange).</p>
		5.1 – 5.1.1	22	March 2, 2015 ^[21]	March 2018	
Android Marshmallow	Macadamia Nut Cookie	6.0 – 6.0.1	23	October 2, 2015 ^[22]	August 2018	
Android Nougat	New York Cheesecake	7.0	24	August 22, 2016	August 2019	
		7.1 – 7.1.2	25	October 4, 2016	October 2019	
Android Oreo	Oatmeal Cookie	8.0	26	August 21, 2017	January 2021	
		8.1	27	December 5, 2017	October 2021	
Android Pie	Pistachio Ice Cream ^[23]	9	28	August 6, 2018	January 2022	
Android 10	Quince Tart ^[24]	10	29	September 3, 2019	February 2023	
Android 11	Red Velvet Cake ^[24]	11	30	September 8, 2020	<p>The timeline diagram illustrates the progression of Android versions from 11 to 15. It shows 11 (Red Velvet Cake), 12 (Snow Cone), 12L (Snow Cone v2), 13 (Tiramisu), 14 (Upside Down Cake), and ends with 15 (Vanilla Ice Cream). The timeline shows a general downward trend from left to right, indicating the chronological order of releases. The versions are color-coded according to the legend: Old version (red), Older version, still maintained (yellow), Latest version (green), and Latest preview version (orange).</p>	
Android 12	Snow Cone	12	31	October 4, 2021		
Android 12L	Snow Cone v2	12.1 ^[a]	32	March 7, 2022		
Android 13	Tiramisu	13	33	August 15, 2022		
Android 14	Upside Down Cake ^[27]	14	34	October 4, 2023		
Android 15	Vanilla Ice Cream ^[28]	15 DP1 ^[29]	V DP1 ^[29]	Q3 2024	February 2024 ^[29]	24.02.15 (January 2024) ^[29]

Legend: Old version Older version, still maintained Latest version Latest preview version

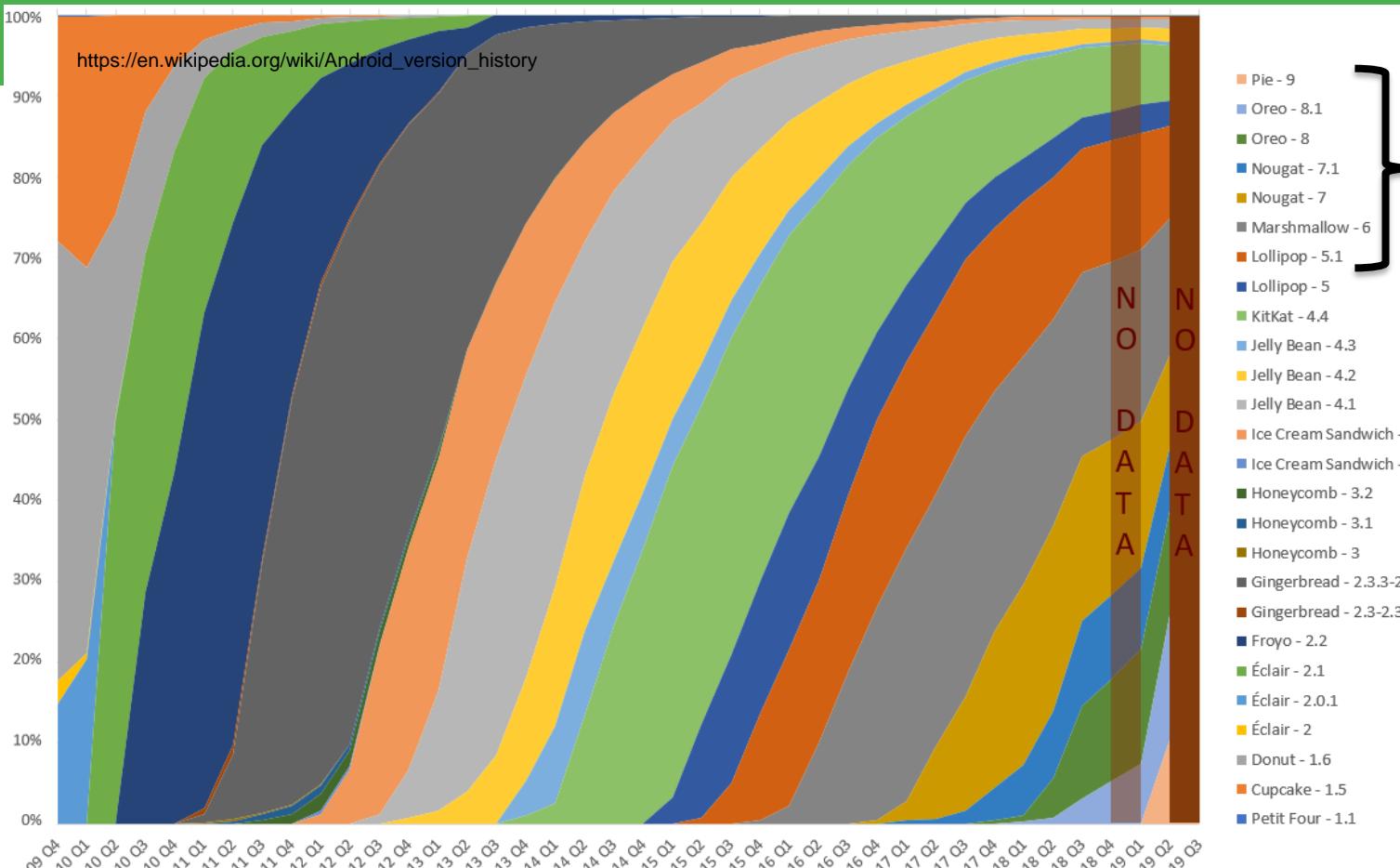
Api Level

Android Studio **allows to choose the minimum API level to support**

balance between

- going **low enough to support as many devices as possible**
- not going **too low that you lose features** your app needs to run

Distribution of Android Versions



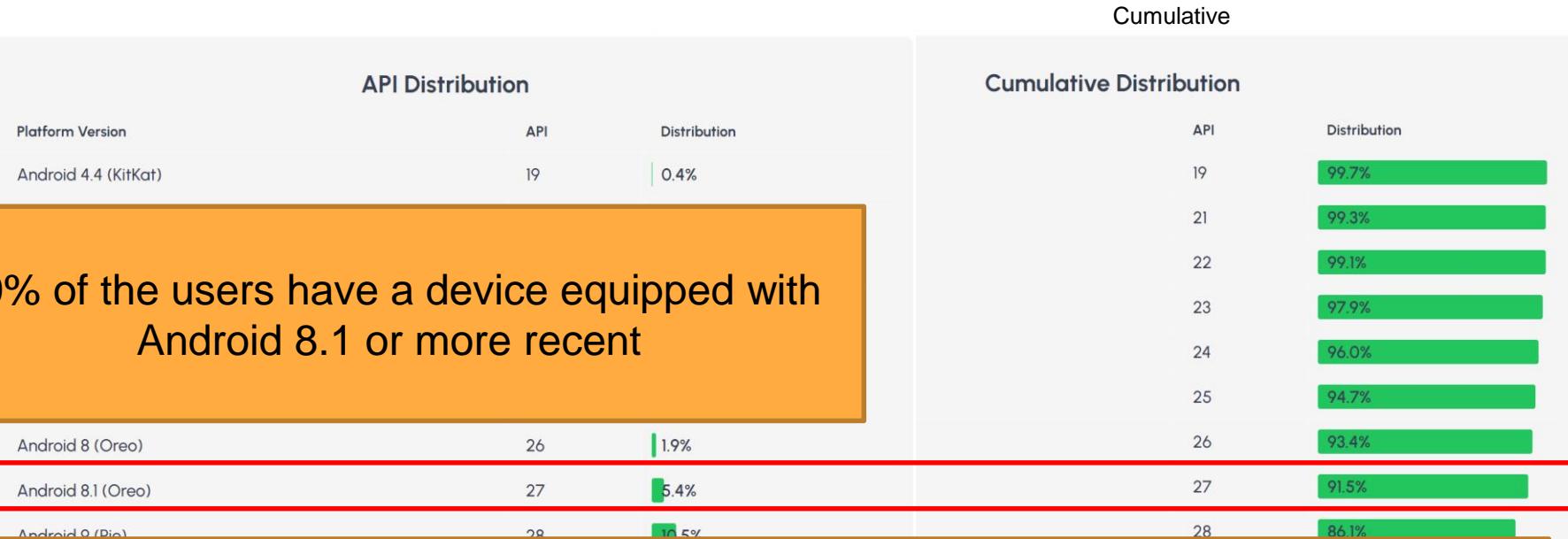
Cover about the
90% of the
deviced used
In 2019

Global Android
version
distribution
according to
Google Play
Store usage,
charted between
Q4 2009 and Q1
2019

Android Distribution - 2024

Platform Version	API	Distribution	Cumulative	
			API	Distribution
Android 4.4 (KitKat)	19	0.4%	19	99.7%
Android 5 (Lollipop)	21	0.2%	21	99.3%
Android 5.1 (Lollipop)	22	1.2%	22	99.1%
Android 6 (Marshmallow)	23	1.9%	23	97.9%
Android 7 (Nougat)	24	1.3%	24	96.0%
Android 7.1 (Nougat)	25	1.3%	25	94.7%
Android 8 (Oreo)	26	1.9%	26	93.4%
Android 8.1 (Oreo)	27	5.4%	27	91.5%
Android 9 (Pie)	28	10.5%	28	86.1%
Android 10 (Q)	29	16.1%	29	75.6%
Android 11 (R)	30	21.6%	30	59.5%
Android 12 (S)	31	15.8%	31	37.9%
Android 13 (T)	33	22.1%	33	22.1%

Android Distribution - 2024



but remember that 10% corresponds to **hundreds of millions of excluded users**

App building blocks

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



Creating the first Android app

Contents

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

These slides are partially based on the material that Google provides for the course
Android Developer Fundamentals

Installation Overview

- Mac, Windows, or Linux
- Download and install Android Studio from
<https://developer.android.com/studio/>
- <https://developer.android.com/studio/install.html>
- Codelab here:
<https://codelabs.developers.google.com/codelabs/android-training-hello-world/>

Installation Overview

- Mac, Windows, or Linux
- Download and install Android Studio from
<https://developer.android.com/studio/>
- <https://developer.android.com/studio/install.html>

- Codelab here:
Try to complete it!

<https://codelabs.developers.google.com/codelabs/android-training-hello-world/>

Installation Overview

- Mac, Windows, or Linux
- Download and install Android Studio

<https://developer.android.com/studio>

<https://developer.android.com/codelabs>

- Codelab here:

<https://codelabs.developers.google.com/codelabs/training-hello-world/>

Setup Android Studio

Install and set up Android Studio, so that you can create your first project and run it on a device or emulator.

6 activities • 1 quiz

0% completed

Save your progress
[Sign in](#)

1 Introduction to Android Studio

Video Optional

2 Download and install Android Studio

Codelab

3 Create your first Android app

Codelab

4 Run your first app on the Android Emulator

Codelab

5 How to connect your Android device

Codelab

6 What's next?

Video Optional

Android Studio download archives



<https://developer.android.com/studio/archive>

This page provides an archive of Android Studio releases. However, we recommend that you download the latest stable version or the latest preview version.

For Android Emulator downloads, see the [Emulator download archives](#).

▼ [Android Studio Jellyfish | 2023.3.1 Canary 11](#) February 22, 2024

▼ [Android Studio Jellyfish | 2023.3.1 Canary 10](#) February 16, 2024

^ [Android Studio Iguana | 2023.2.1 RC 2](#) February 13, 2024

Installers

ChromeOS: [android-studio-2023.2.1.22-cros.deb](#) (944.1 MB)
Mac (Apple Silicon): [android-studio-2023.2.1.22-mac_arm.dmg](#) (1.2 GB)
Mac (Intel): [android-studio-2023.2.1.22-mac.dmg](#) (1.2 GB)
Windows (64-bit): [android-studio-2023.2.1.22-windows.exe](#) (1.1 GB)

SHA-256 checksums

c08425a59f881b82683acc1a656ad314f7e017b12596f04249882daae559635f android-studio-2023.2.1.22-cros.deb
a09efd27cf56755e2e05a2a5cb0719ee8d7331818ff74f1cc9932e3272c8865d android-studio-2023.2.1.22-mac_arm.dmg
9f001458dde35fbad89192c27082d461ebaa8ab0906444611cc24d8fb541244c android-studio-2023.2.1.22-mac.dmg
cfdfa4711055c04d97dd46d7c84aafc3bc87e5e65428765ea6f004f7db6799f android-studio-2023.2.1.22-windows.exe

Zip files

Linux: [android-studio-2023.2.1.22-linux.tar.gz](#) (1.2 GB)
Mac (Apple Silicon): [android-studio-2023.2.1.22-mac_arm.zip](#) (1.2 GB)
Mac (Intel): [android-studio-2023.2.1.22-mac.zip](#) (1.2 GB)
Windows (64-bit): [android-studio-2023.2.1.22-windows-exe.zip](#) (1.2 GB)

SHA-256 checksums

b32e027e0f9de03b88502acf3f169ea69ec9f96d64b2e8f438da4c373691ffa android-studio-2023.2.1.22-linux.tar.gz
44eb017a5f5f41d1c048e09f49b8c0040df29d039d84c914f4386ce55761b270 android-studio-2023.2.1.22-mac_arm.zip
ca326ff801e4ad996da66c9950af5447ab05511a2d65a13904d73806a4f56231 android-studio-2023.2.1.22-mac.zip
b4060d946c256f23695787e034edc9603615735b0ebc15a39fd011fa5e25a224 android-studio-2023.2.1.22-windows-exe.zip

I installed this version.

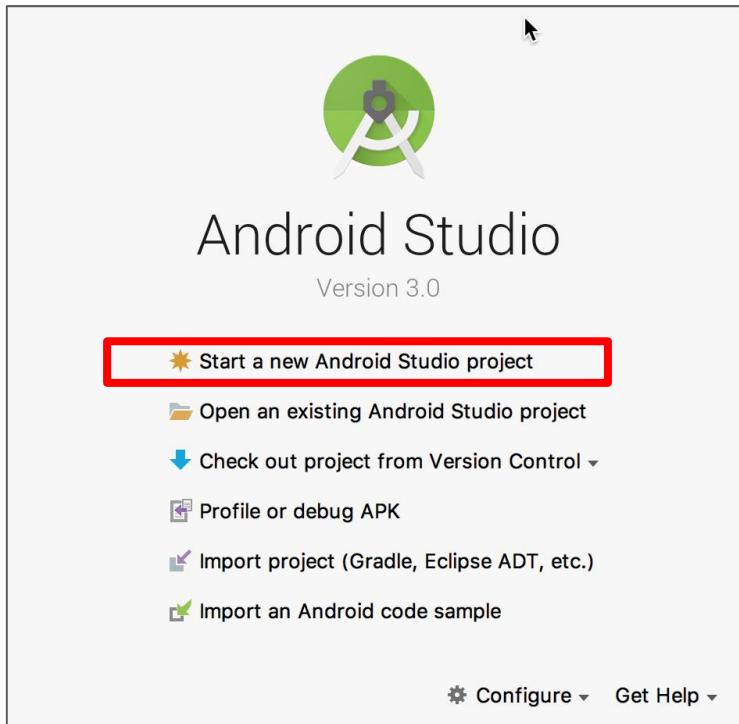
Simply decompress the zip folder and run:
android-studio-2023.2.1.22-windows\bin\studio.bat (windows)



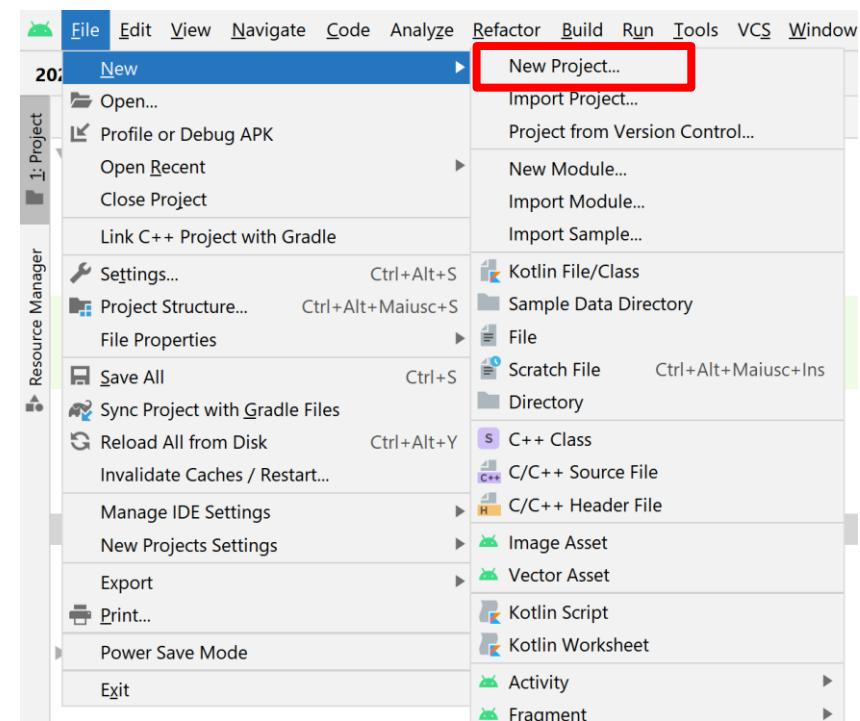
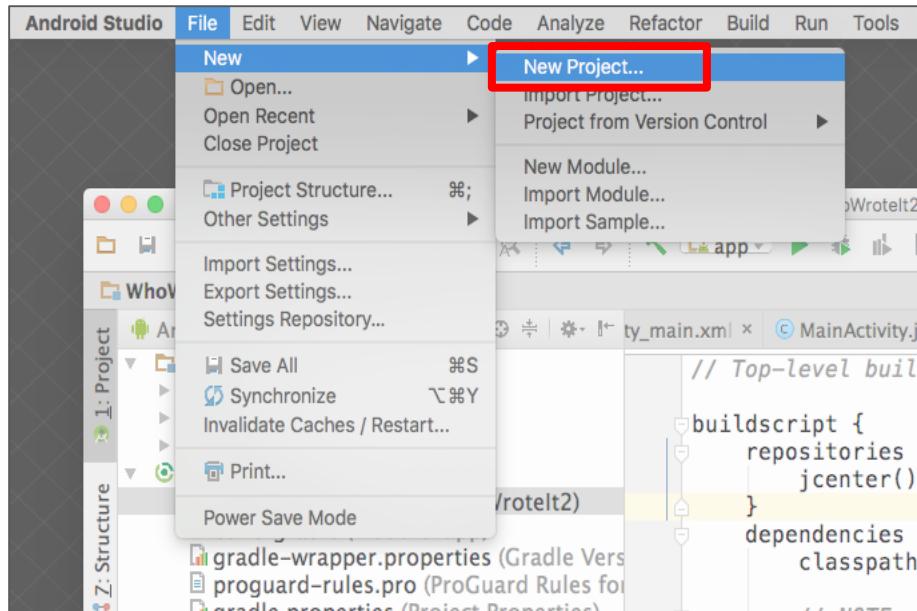
What is Android Studio?

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

Start Android Studio



Create a project inside Android Studio

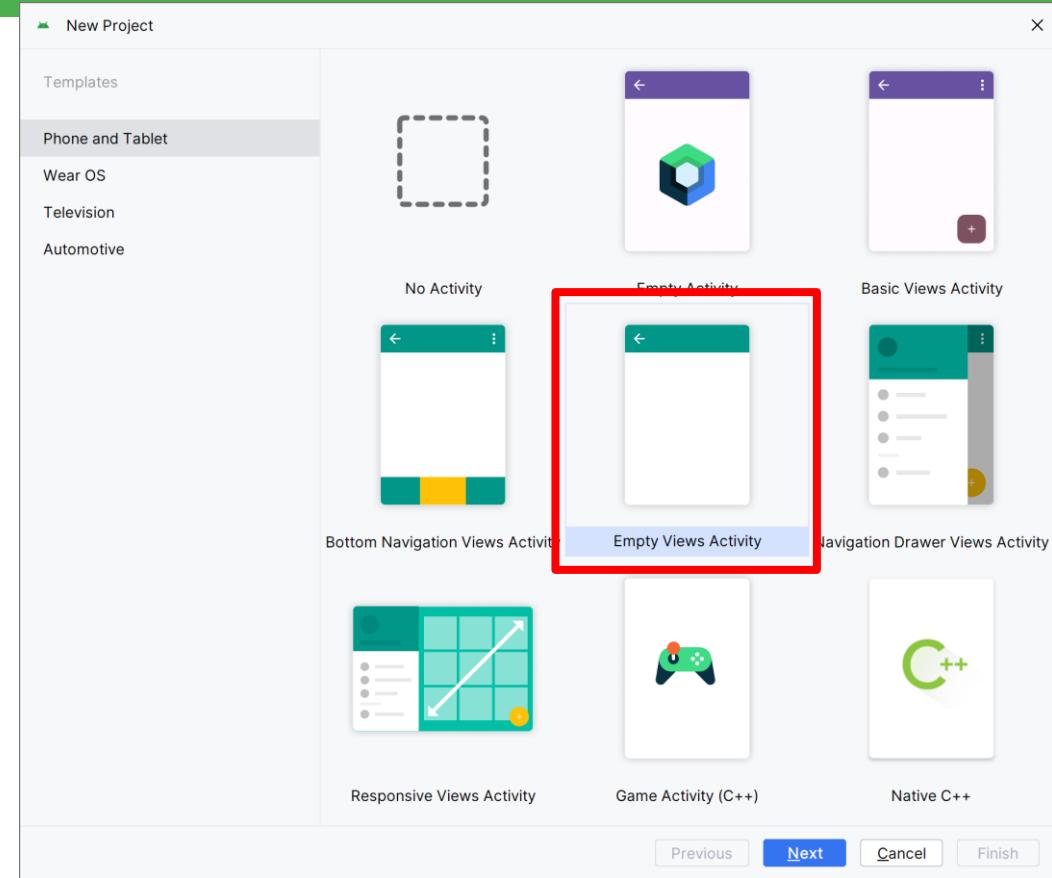


Pick activity template

android-studio-2023.2.1.22

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

Pick **Empty Views Activity** for simple and custom activities

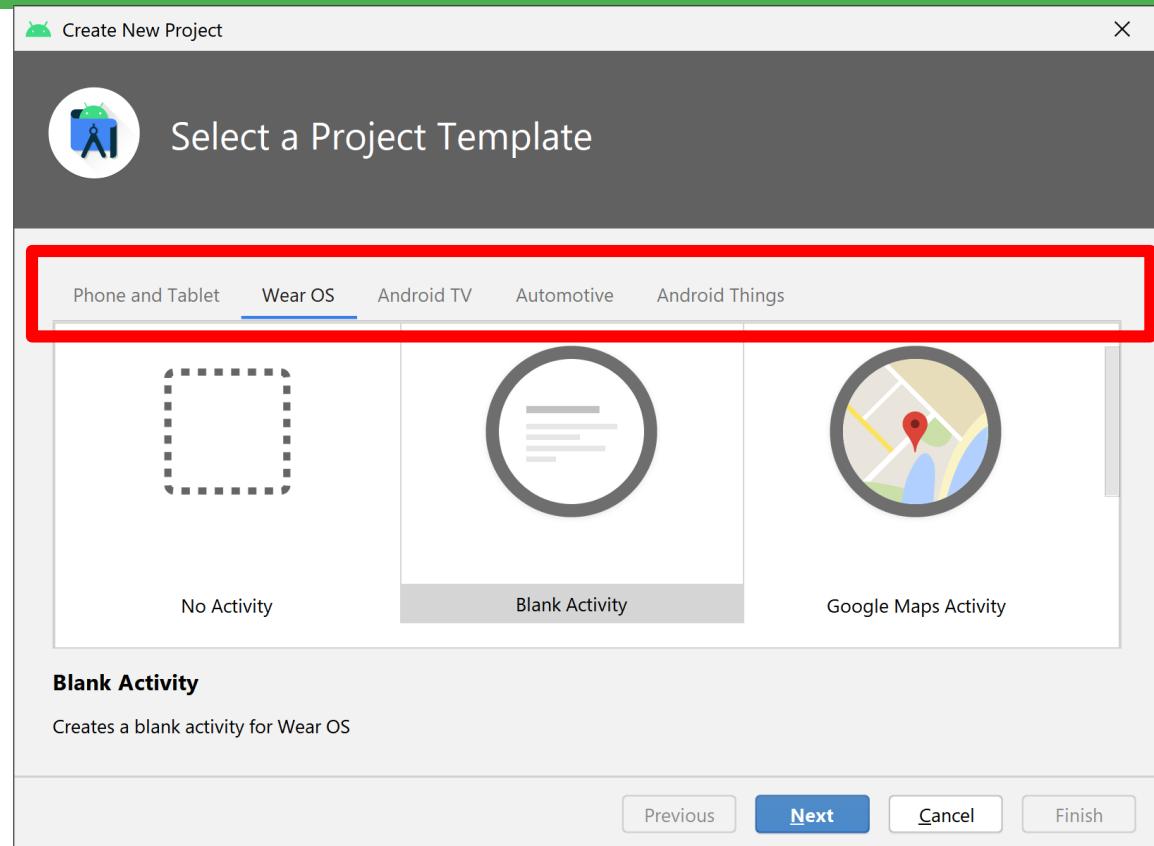


<https://stackoverflow.com/questions/76018702/i-cant-create-a-new-project-with-java-language-anymore-on-android-studio-flamin>

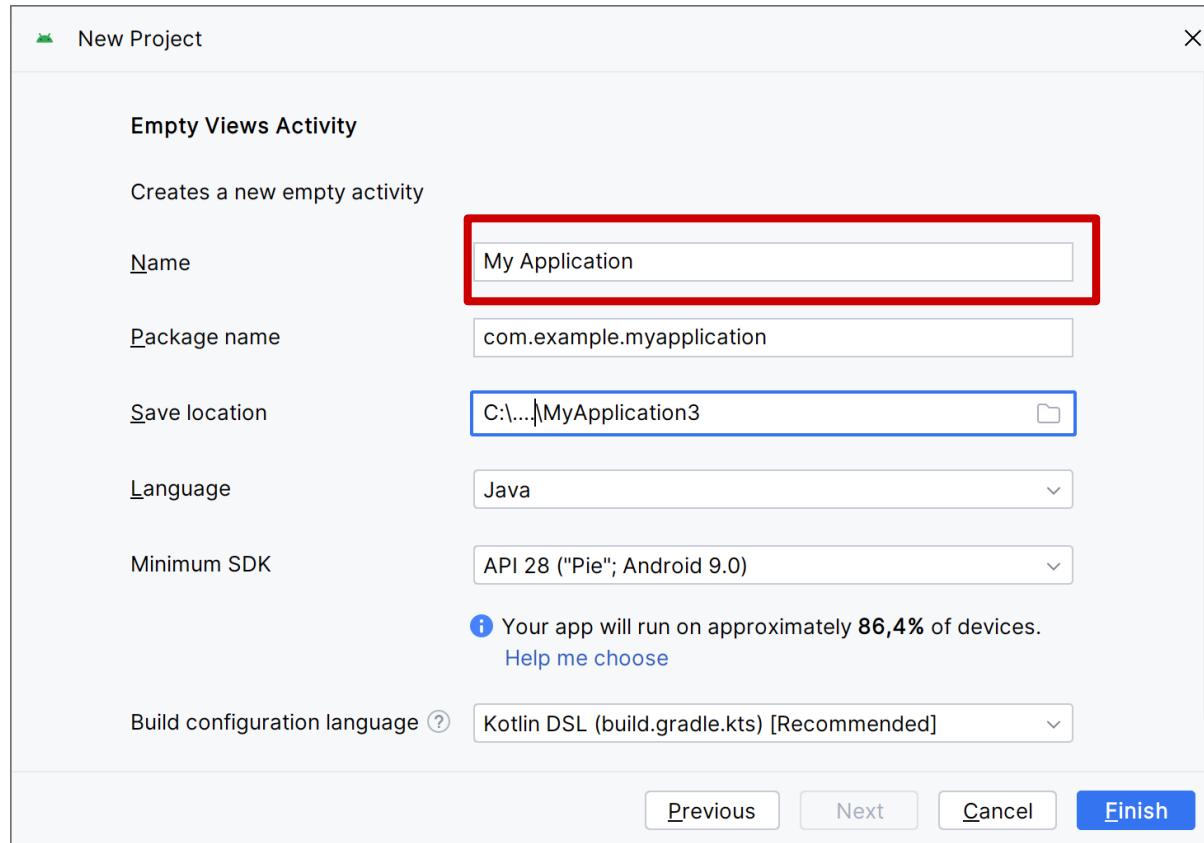
Pick activity template

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

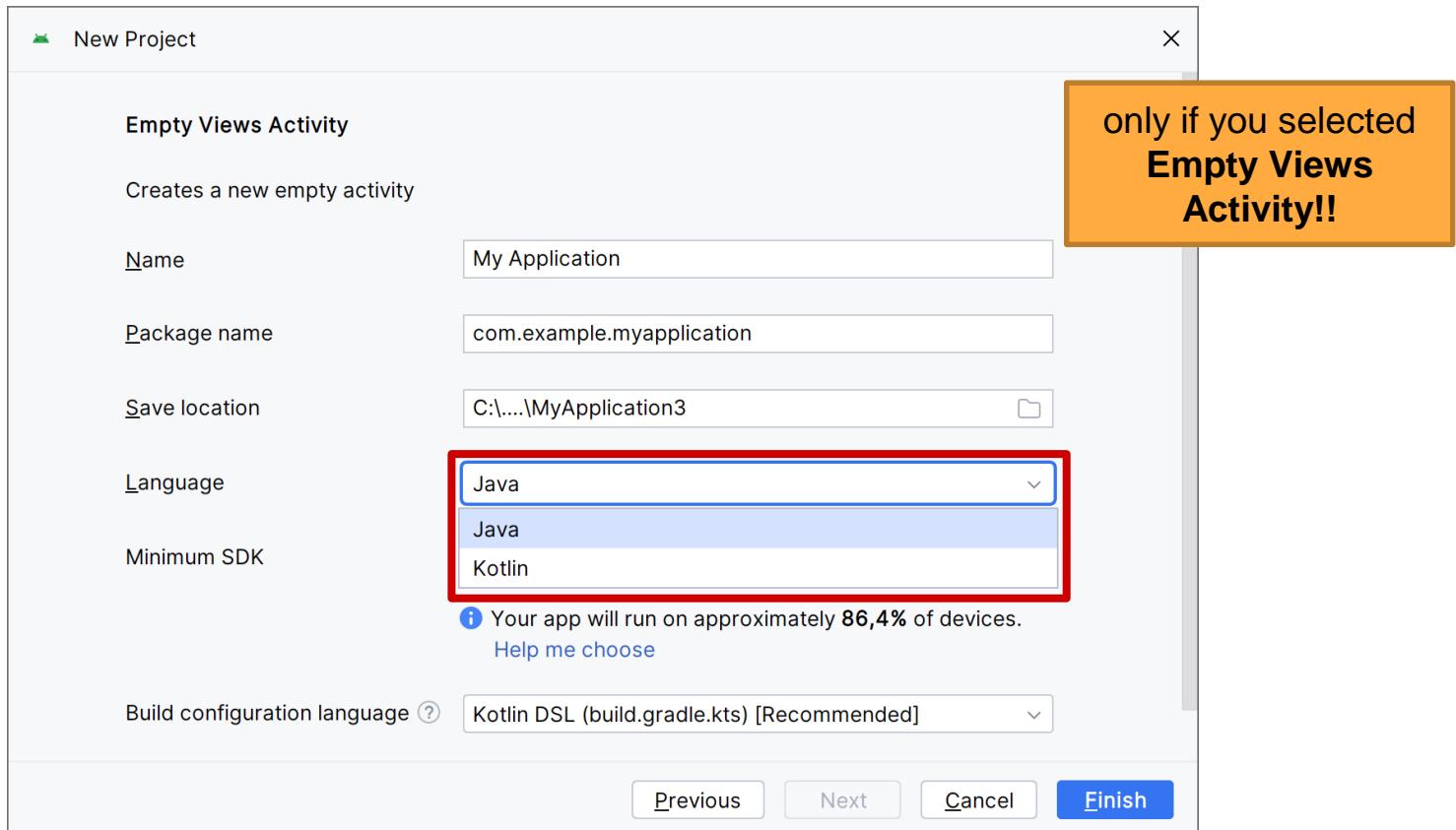
Pick **Empty Views Activity** for simple and custom activities



Name your app



Language to code your app



Min API level for your app

Android Platform/API Version Distribution

ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.4 KitKat	19	99,6%
5 Lollipop	21	99,4%
5.1 Lollipop	22	98,2%
6 Marshmallow	23	96,3%
7 Nougat	24	95,0%
7.1 Nougat	25	93,7%
8 Oreo	26	91,8%
8.1 Oreo	27	86,4%
9 Pie	28	75,9%
10 Q	29	59,8%
11 R	30	38,2%
12 S	31	22,4%
13 T	33	

Last updated: October 1, 2023

<https://developer.android.com/about/versions/pie/android-9.0>

OK Cancel

Pie

System	Security and privacy
Indoor positioning with Wi-Fi RTT	Android Protected Confirmation
Multi-camera support	Biometric authentication dialogs
Display cutout support	Hardware security module
	Secure key import
	Client-side encryption backups
User Interface	Accessibility
Improved notifications	Navigation semantics
Improved text support	Convenience actions
ImageDecoder and new animation classes	Magnifier
Media	
HDR VP9 video	
HEIF image compression	
Improved media APIs	

ct

Name: ApplicationName

Package name: com.example.applicationname

Save location: D:\....\AndroidStudioProjects\MyApplication4

Language: Java

Minimum API level: API 14: Android 4.0 (IceCreamSandwich)

Your app will run on devices running API 14 or higher.

Help me choose

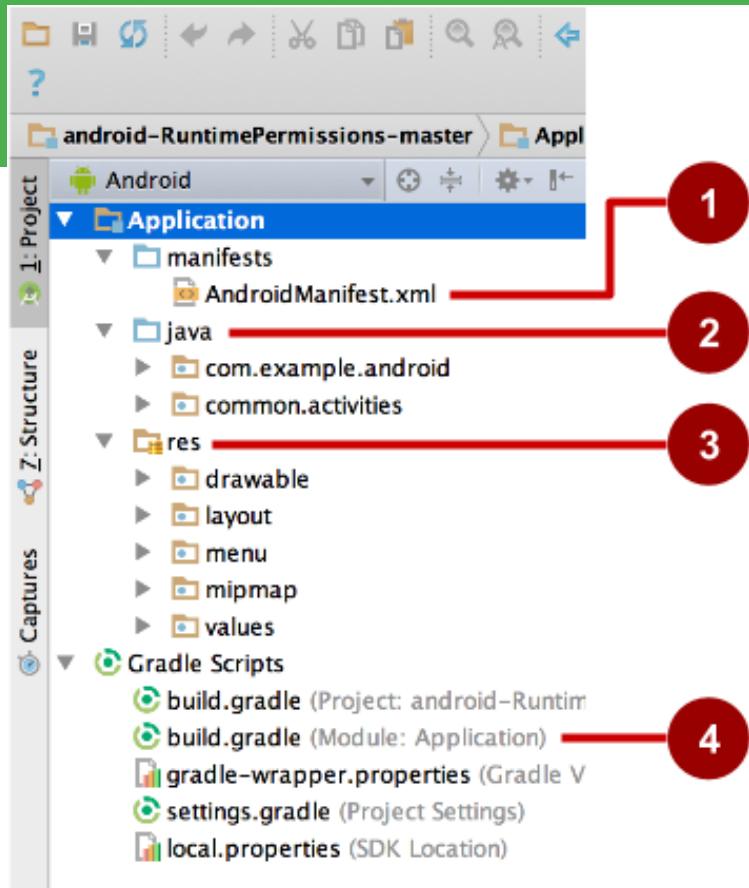
This project will run on devices running API 14 or higher.

Use android.*: a

OK Cancel Previous Next Finish

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



Android / Gradle build system

Android build system

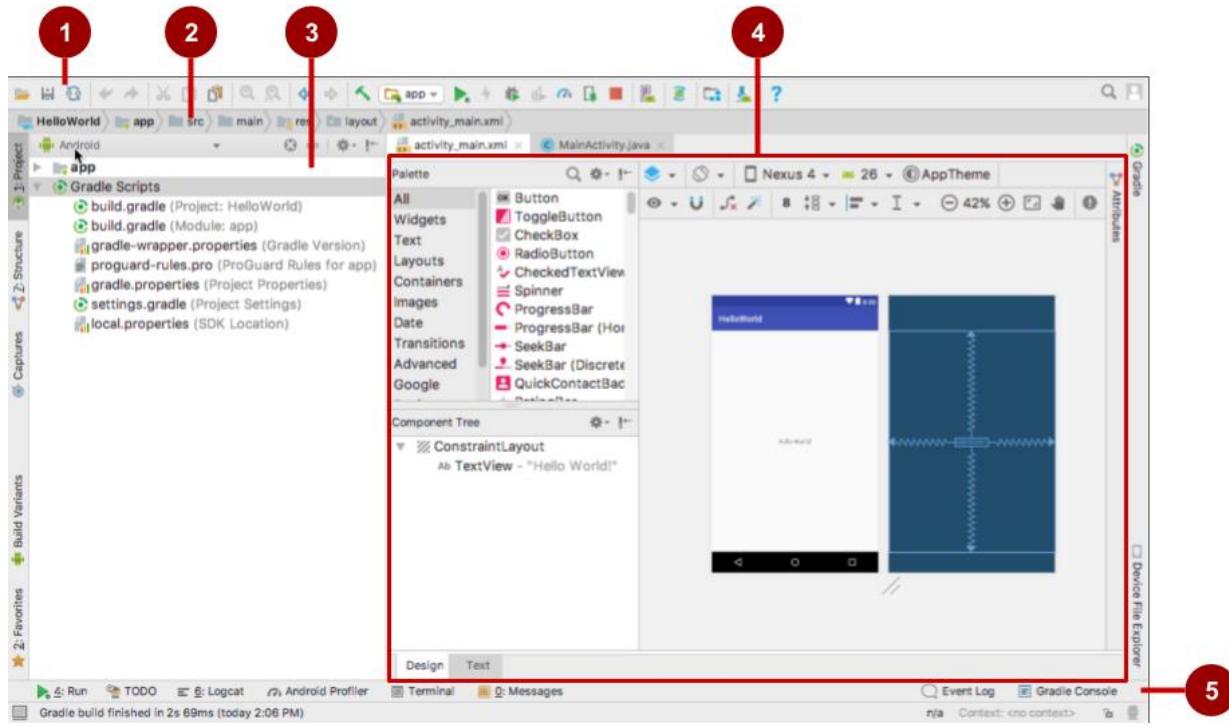
- **compiles app resources** and source code
- **packages them into APKs** or Android App Bundles
 - ready to test, deploy, sign, and distribute

Android Studio uses Gradle, an advanced build toolkit, to **automate and manage the build process**

Gradle and the Android Gradle plugin run independent of Android Studio. Android apps can be built from

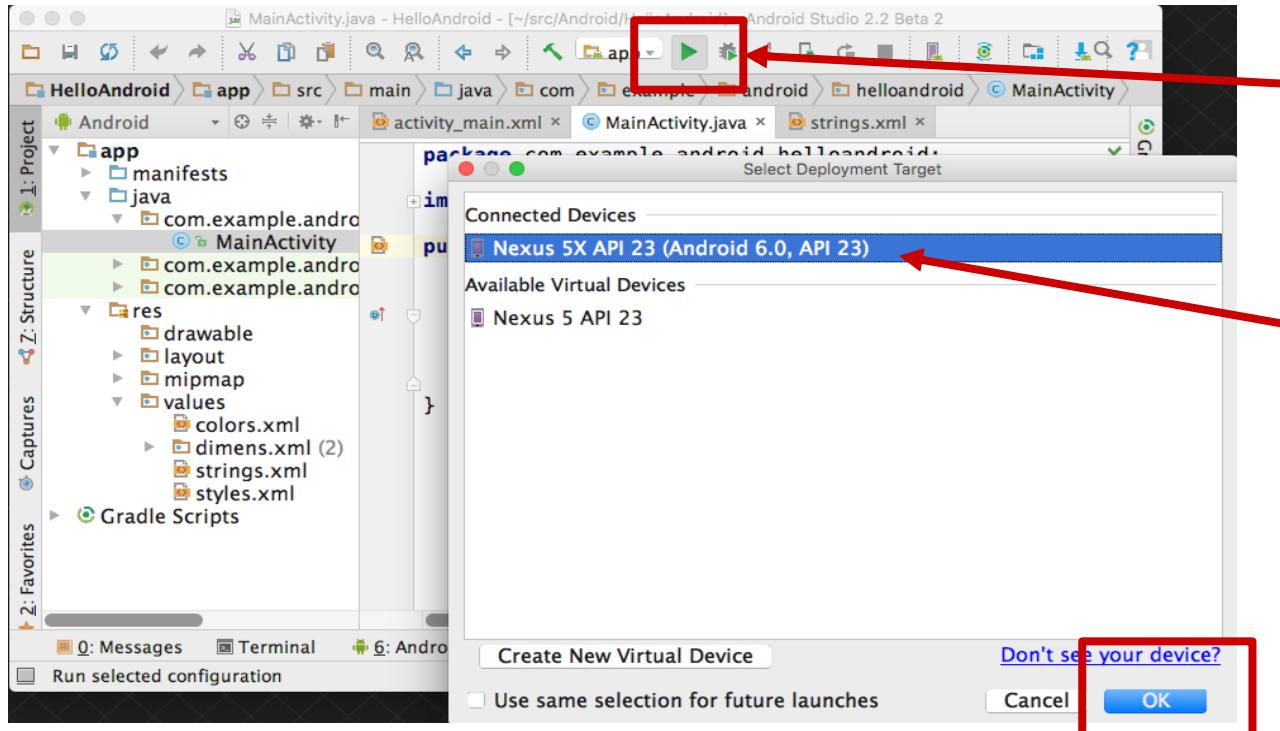
- within Android Studio
- the command line on your machine
- on machines where Android Studio is not installed
 - such as continuous integration servers

Android Studio interface



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

Run your app



1. Run

2. Select virtual
or physical
device

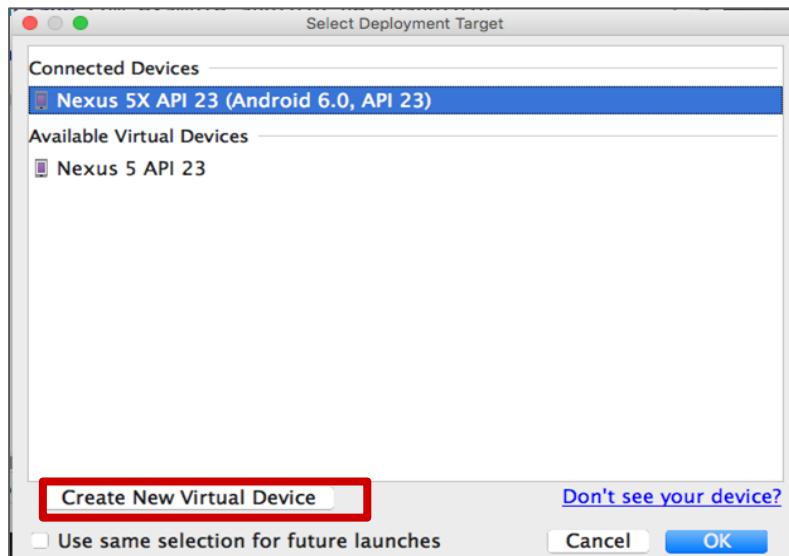
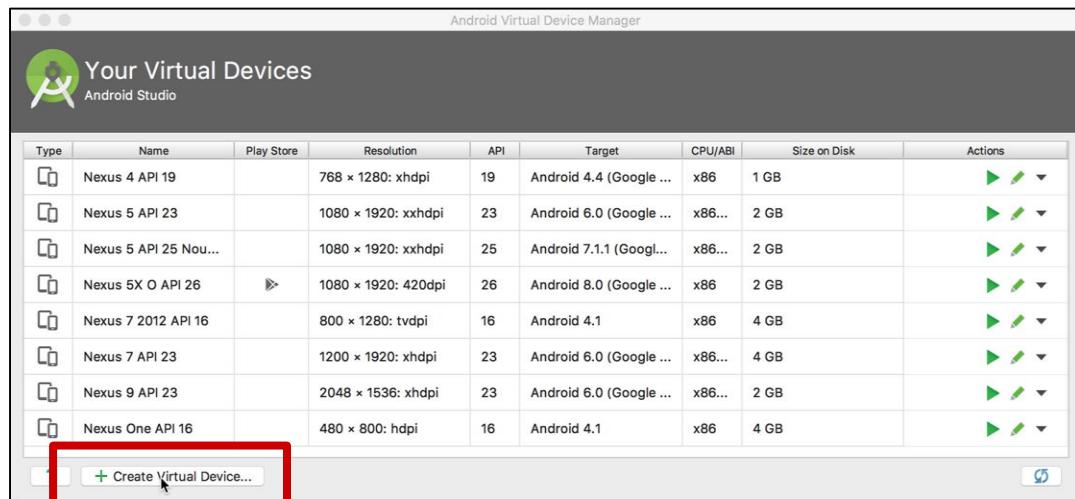
3. OK

Create a virtual device

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

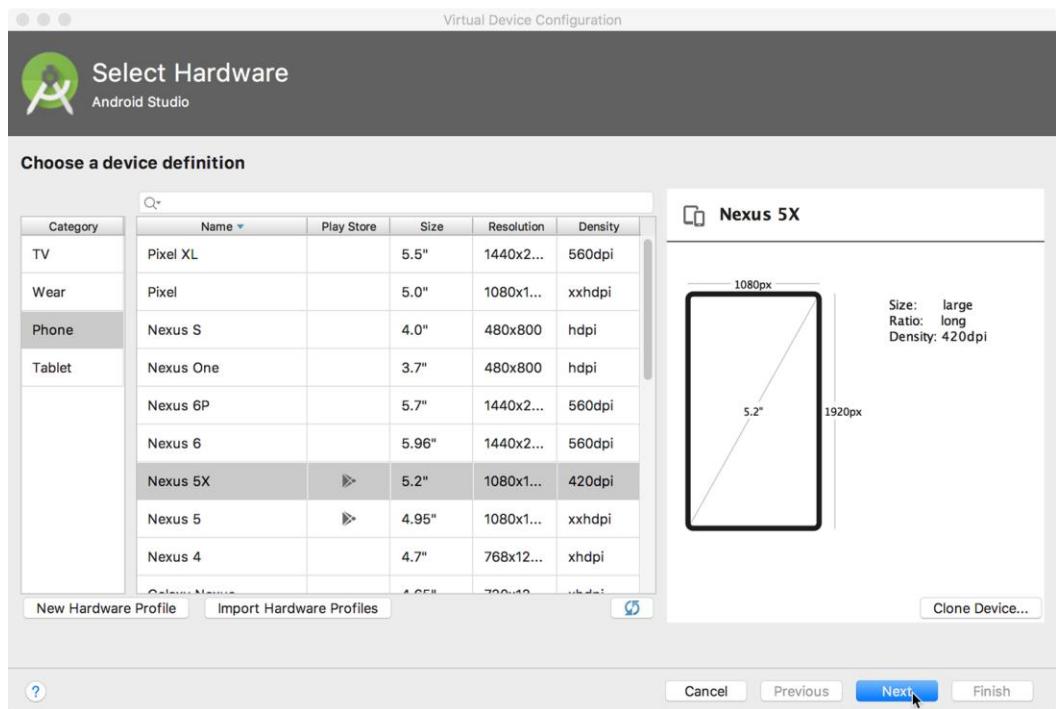
Tools > Android > AVD Manager

or:



Configure virtual device

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



Configure virtual device

Virtual Device Configuration

Select Hardware

Choose a device definition

Name	Play S...	Size	Resolu...	Density
Pixel 4a		5,8"	1080x...	440dpi
Pixel 4 XL		6,3"	1440x...	560dpi
Pixel 4	▷	5,7"	1080x...	440dpi
Pixel 3a XL		6,0"	1080x...	400dpi
Pixel 3a	▷	5,6"	1080x...	440dpi
Pixel 3 XL		6,3"	1440x...	560dpi
Pixel 3	▷	5,46"	1080x...	440dpi

New Hardware Profile Import Hardware Profiles

Clone Device...

Device Manager

Pixel_3a_API_34_extension_level_7_x86_64
Android 14.0 ("UpsideDownCake") | x86_64

API Type

34 Virtual

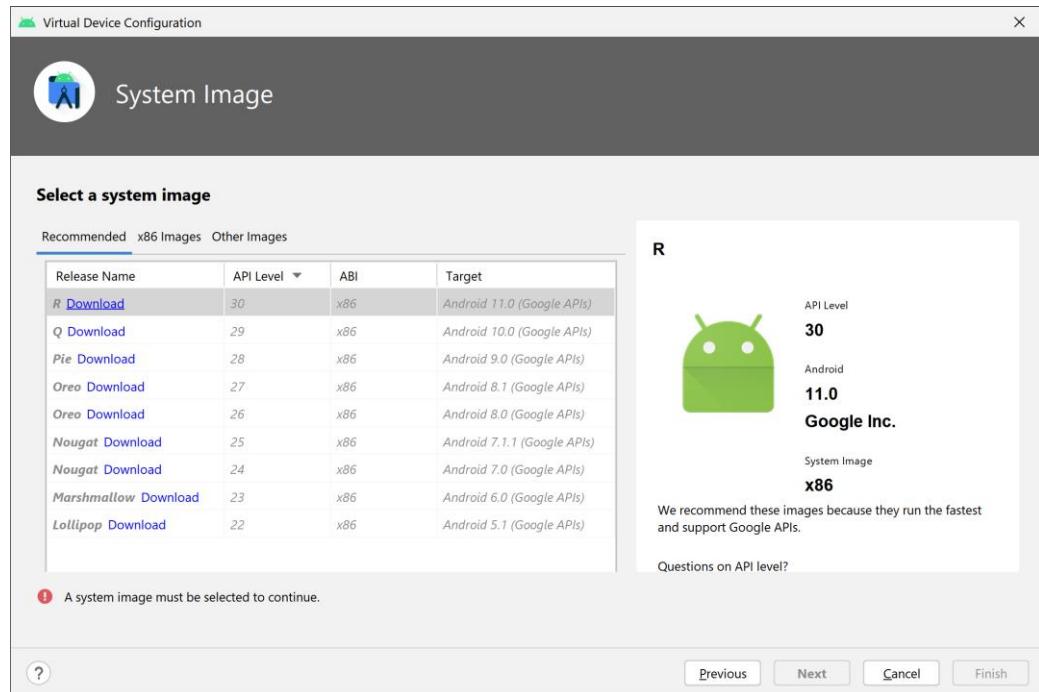
Pixel 4

1080px
5,7"
2280px
Size: large
Ratio: long
Density: 440dpi

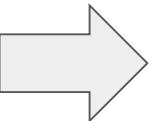
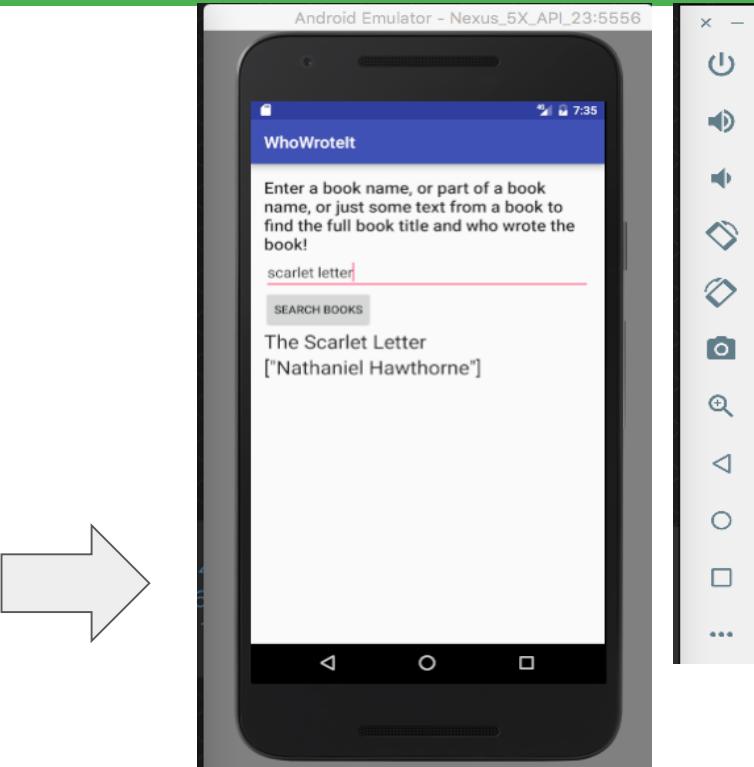
android-studio-2023.2.1.22

Configure virtual device

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



Run on a virtual device



Run on a physical device

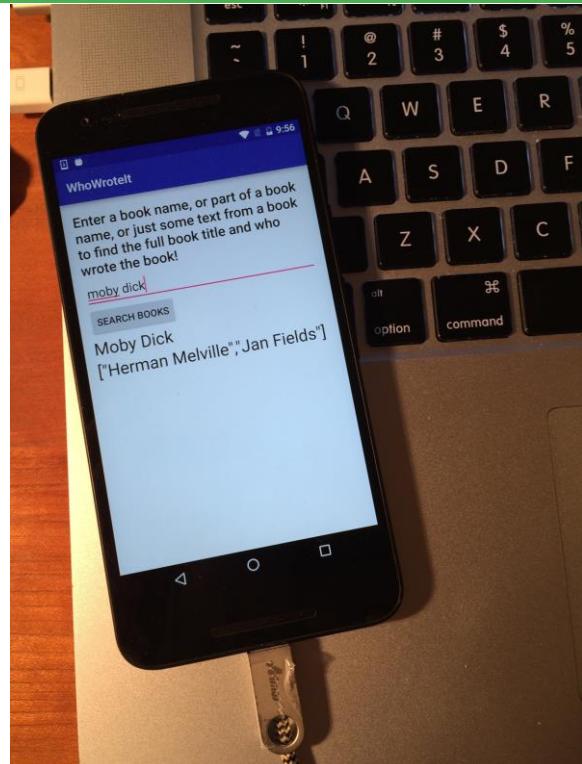
1. Turn on Developer Options:
 - a. **Settings > About phone**
 - b. Tap **Build number** seven times
2. Turn on USB Debugging
 - a. **Settings > Developer Options > USB Debugging**
3. Connect phone to computer with cable

Windows/Linux additional setup:

- <https://developer.android.com/studio/run/device>

Windows drivers:

- <https://developer.android.com/studio/run/oem-usb>



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 >**
3. Connect phone to computer with cable

Windows/Linux additional setup:

- <https://developer.android.com/studio/run/win-udev-setup>

Windows drivers:

- <https://developer.android.com/studio/run/win-hardware-driver-setup>

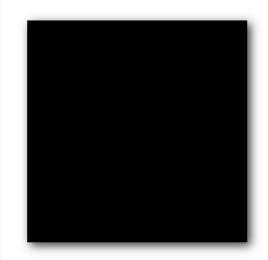
Pair devices over Wi-Fi

Pair new devices over Wi-Fi

Pair devices to enable wireless debugging. Pair camera-enabled devices using a QR code. Other devices can be paired using a pairing code. [Learn more](#)

Pair using QR code Pair using pairing code

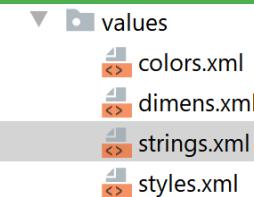
To pair an Android 11+ device
scan the QR code from your device



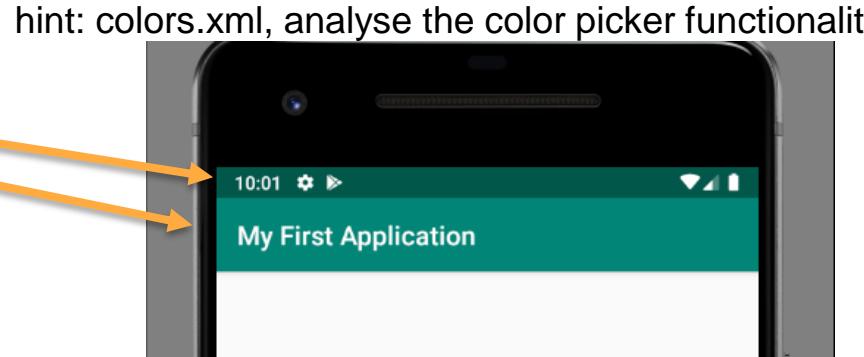
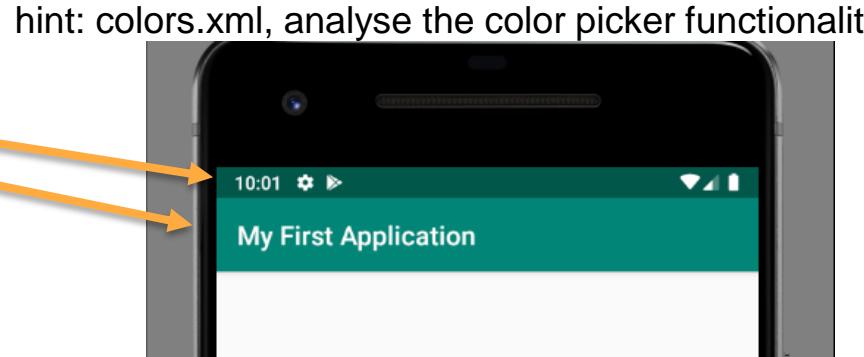
QR scanner available at:
Developer options > Wireless debugging > Pair using QR code

Close

Exercise One



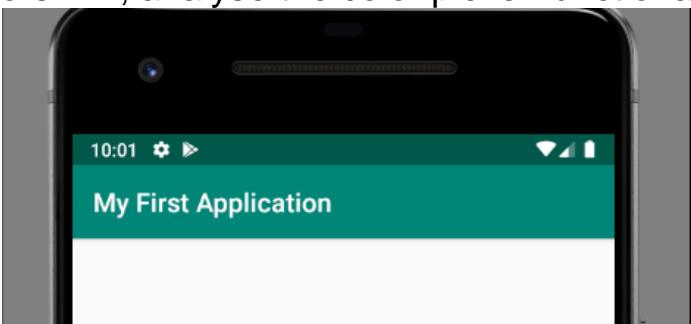
- Develop the Hello World => <https://codelabs.developers.google.com/codelabs/android-training-hello-world/>
 - Run the Hello World app (i.e. an empty app) on the Emulator
 - Run the Hello World app on your real device (if you have one)
 - Change the Name of the App
 - e.g., from 'My Application' to 'My First Application' (see xml files in "res" folder)
 - Change the color of these two components
- Check that everything works



Exercise One

- Develop the Hello World =>
- Run the Hello World app (i.e. on the Emulator)
- Run the Hello World app on your phone
- Change the Name of the Application
 - e.g., from 'My Application' to 'My First Application'
- Change the color of these two components

Check that everything works



values

- colors.xml
- dimens.xml
- strings.xml
- styles.xml

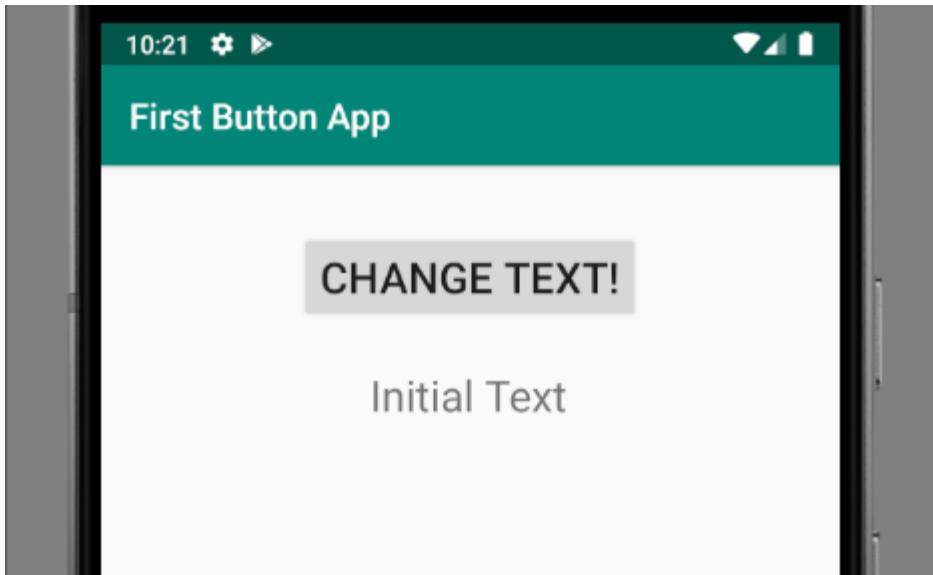
Missing in the latest version of the default app template
(using Material Design 3)

You can skip this part or try to understand how to add the bars to the app:
<https://m3.material.io/components/top-app-bar/overview>

Hint: colors.xml, analyse the color picker functionality

Exercise Two

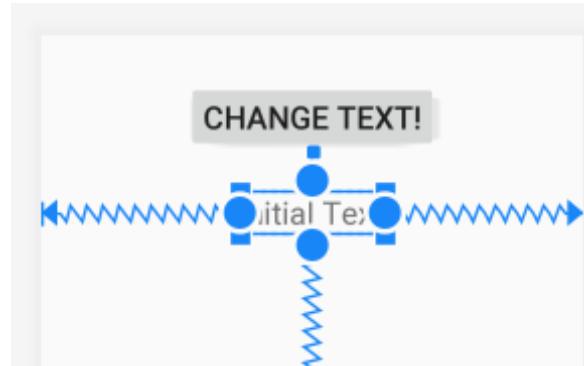
- We want to interact with the app using a button.... Create a new app
- Then create a simple activity like this:



- A title
- A button
- A textView

Hint: open the layout file
e.g. activity_main.xml
as shown in the next slide

Use the default Constraint Layout



Palette



Pixel



30



MyApplication



Default (en-us)



Common

Ab TextView

Button

ImageView

RecyclerView

<fragment>

ScrollView

Switch

Text

Buttons

Widgets

Layouts

Containers

Helpers

Google

Legacy



Exercise Two

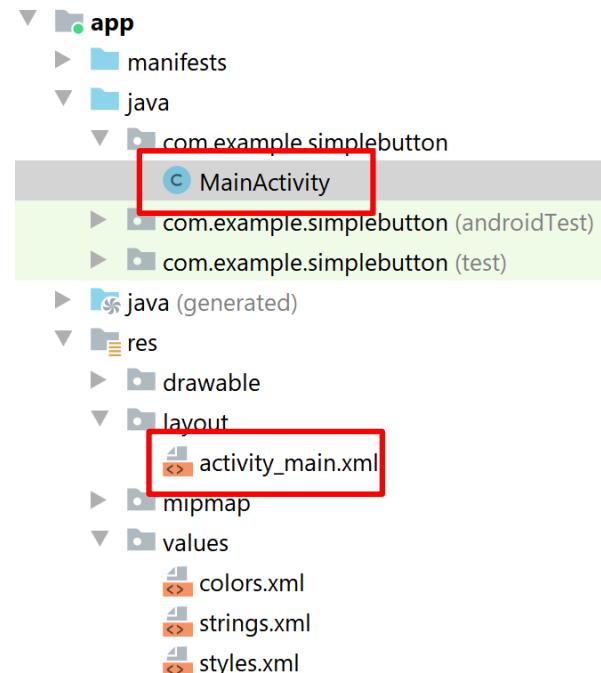
Required Behavior:

- when the **User clicks** on the **button** labelled «*Change Text!*» the **textView** must change its text from «*Initial Text*» to «*Updated Text!!*»

We have to:

- Write a method that implements such behavior (in java/MainActivity.java)
- Associate such method to the button

Project Structure

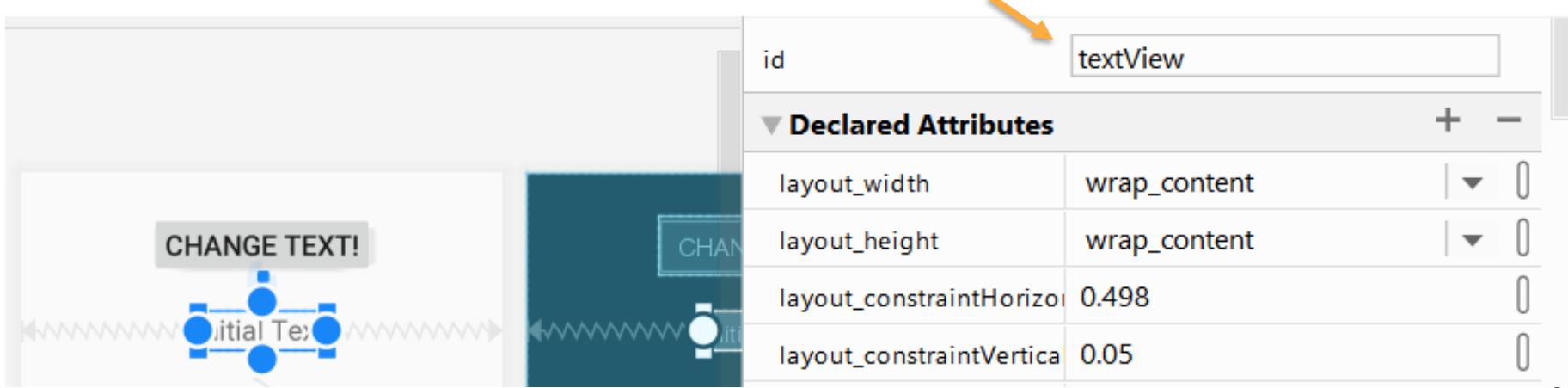


Exercise Two

In detail:

Hint: open the layout file
e.g. activity_main.xml

- Insert the widgets in the main activity (Button and TextView)
- Provide a meaningful ID to our textView



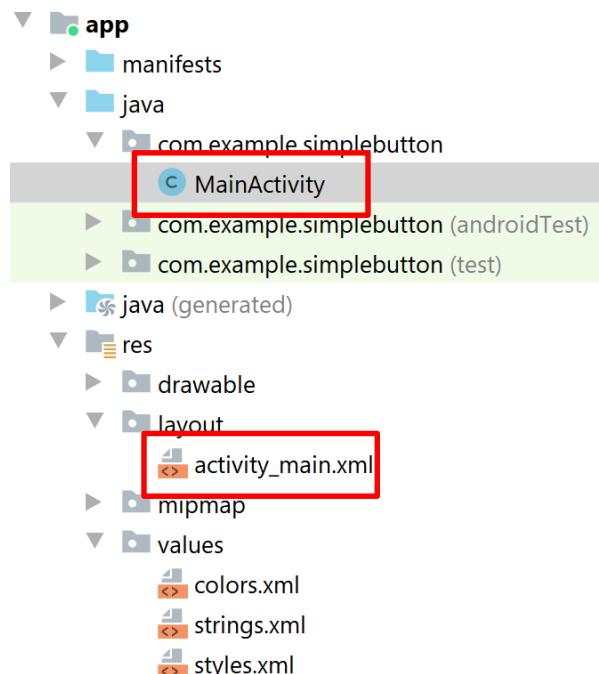
Exercise Two

In detail:

Write the code implementing the behavior in
'MainActivity.java'



Project Structure



Idea:

- when the user clicks on the button
- find the textView element (using the ID)
- assign the new text to it

Exercise Two

In detail:

Write the code implementing the behavior in ‘MainActivity.java’

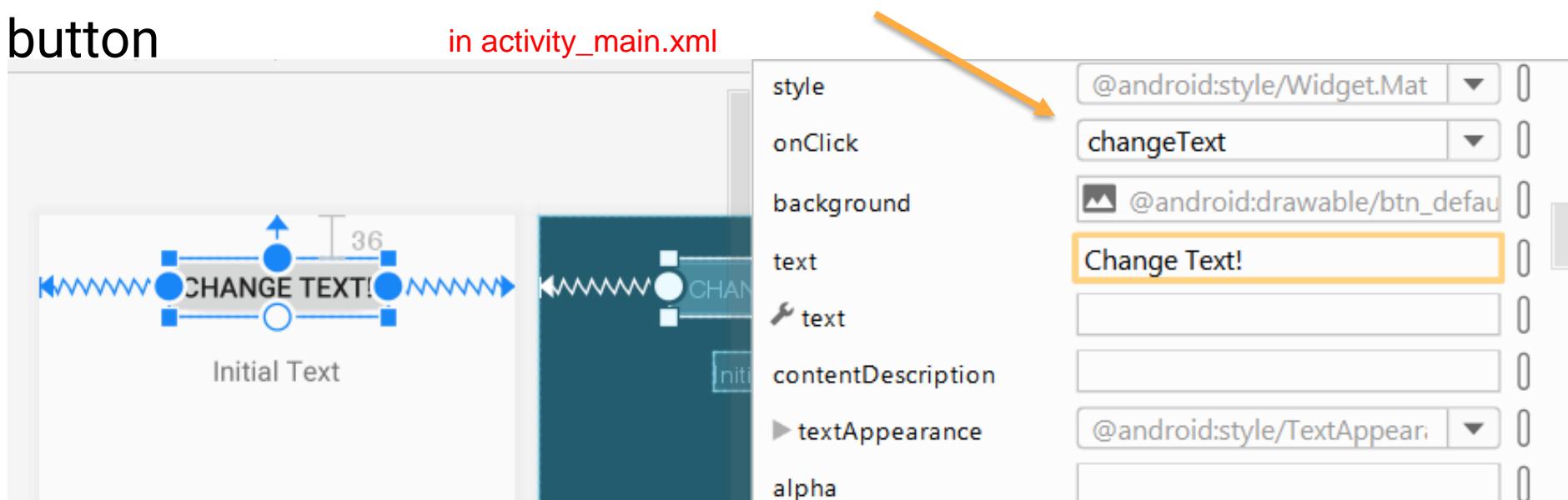
```
public void changeText (View v)
{
    TextView t = (TextView) findViewById(R.id.textView);
    t.setText("Updated Text!!");

    // in this example the View parameter is the Button
    // instance that has fired the method
    ((Button)v).setText("Change Text! (Pressed)");
}
```

Exercise Two

In detail:

Call **changeText** method in response to **onClick** events on the button

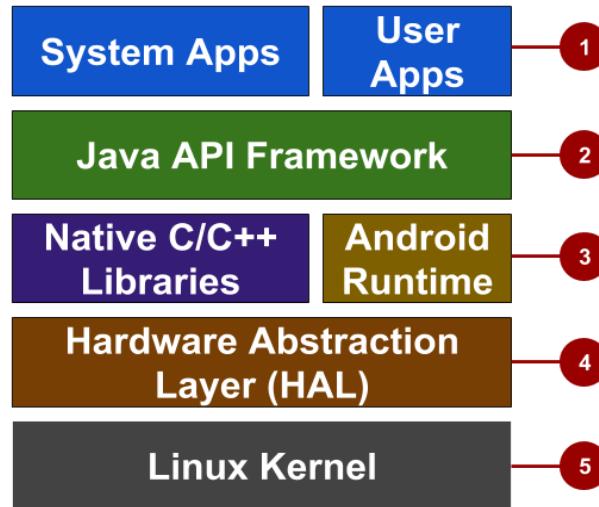


Android Platform Architecture

Android stack

<https://developer.android.com/guide/platform>

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

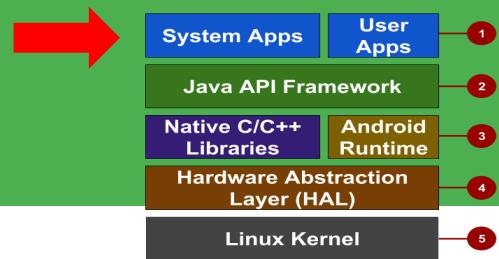


Layers

These slides are partially based on the material that Google provides for the course
Android Developer Fundamentals

<https://developer.android.com/courses/fundamentals-training/overview-v2>

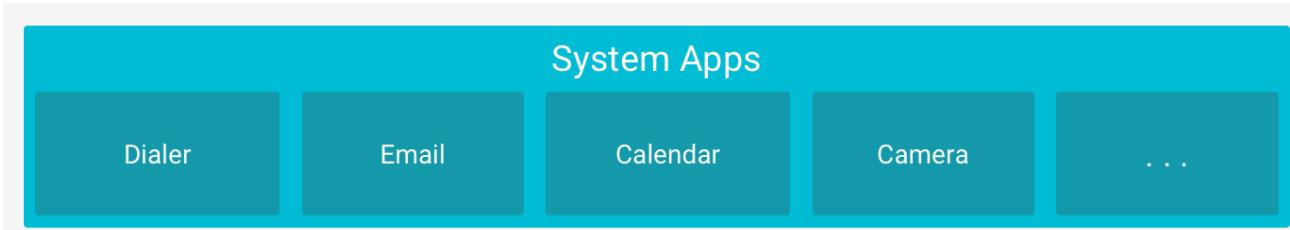
System and user apps



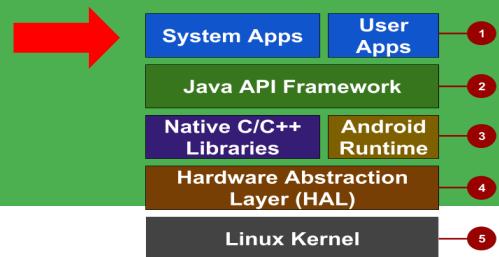
- System apps have ***no special status***
- System apps ***provide key capabilities*** to app developers
 - email, SMS messaging, calendars, internet browsing, contacts

Example:

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



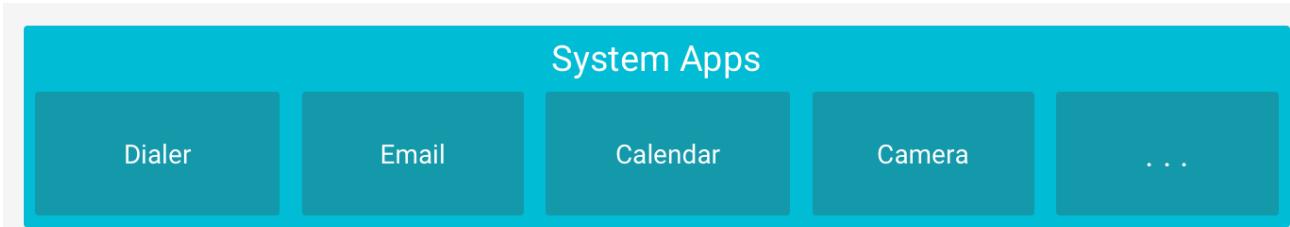
System and user apps



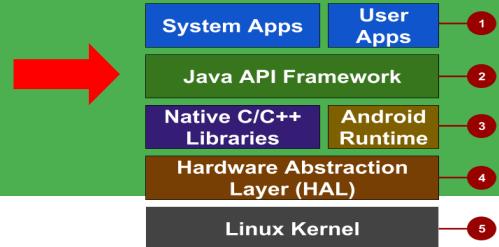
- System apps have ***no special status***
- System apps ***provide key capabilities*** to app developers
 - email, SMS messaging, calendars, internet browsing, contacts

Example:

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

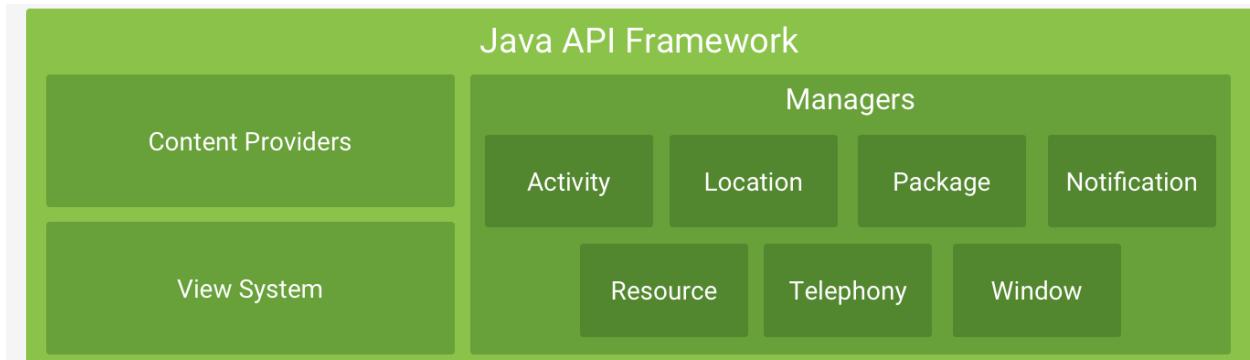


Java API Framework



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

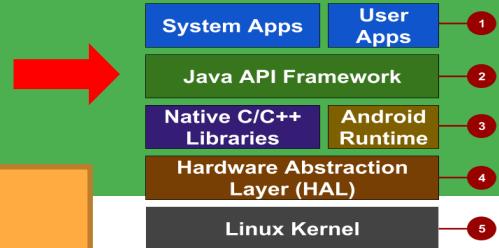
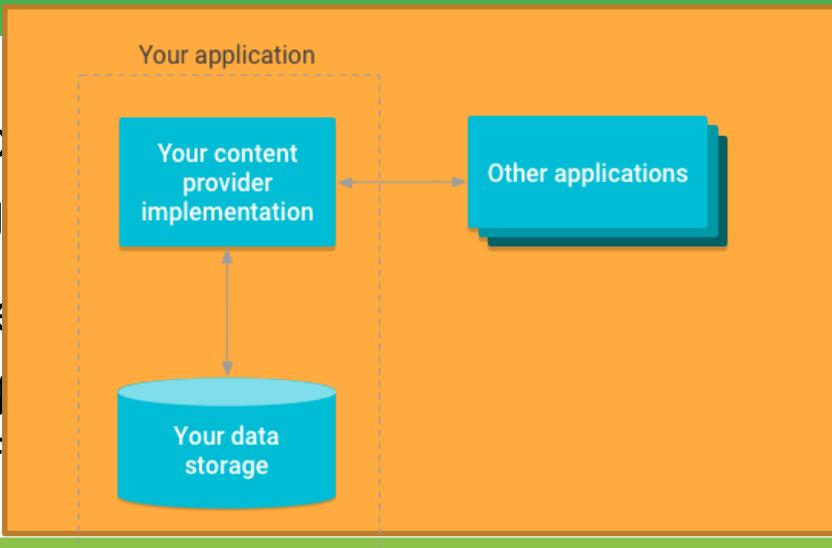
- **View system:** to create UI screens
- **Notification manager:** to display custom alerts in the status bar
- **Activity manager:** for app life cycles and navigation
-



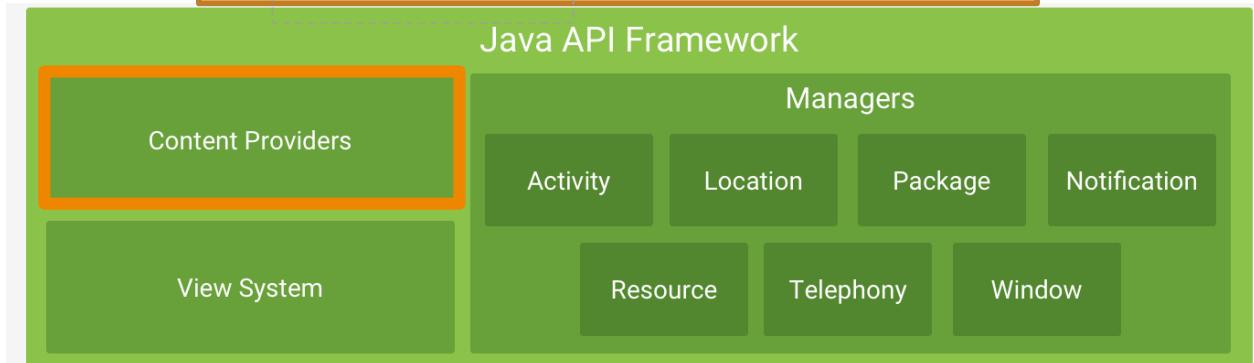
Java API Framework

The entire **feature-set** of the OS is written in the Java language.

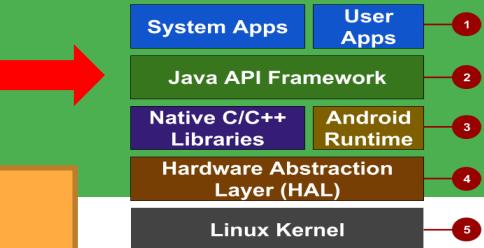
- **View system:** to create UI
- **Notification manager:** to show notifications in the status bar
- **Activity manager:** for managing activities
-



through APIs

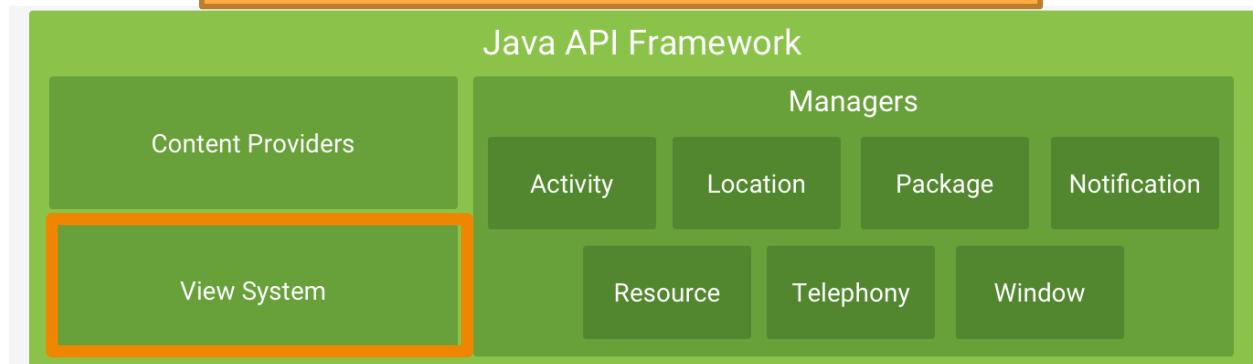
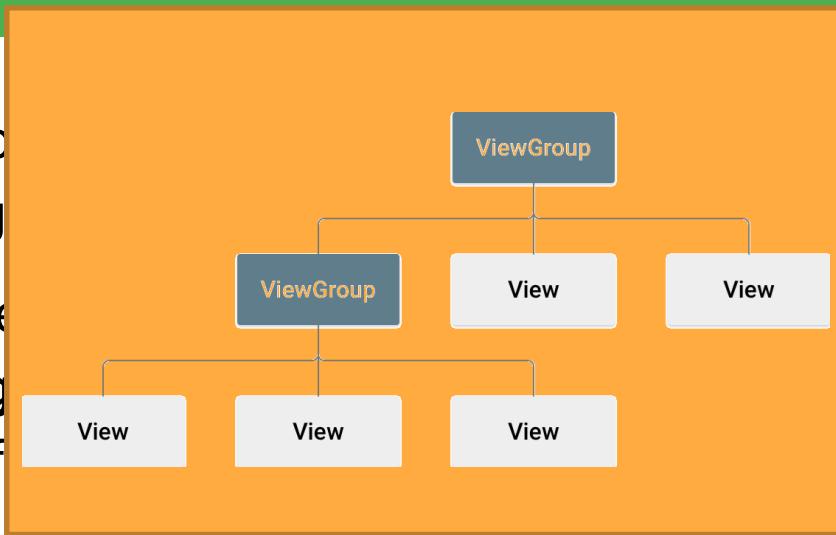


Java API Framework

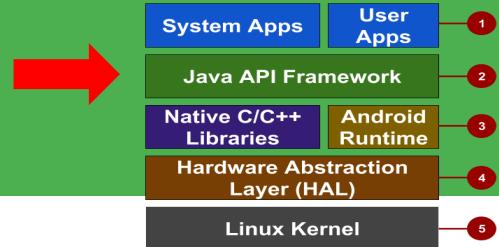


The entire **feature-set** of the OS is written in the Java language

- **View system:** to create UI components
- **Notification manager:** to show notifications in the status bar
- **Activity manager:** to manage activities
-

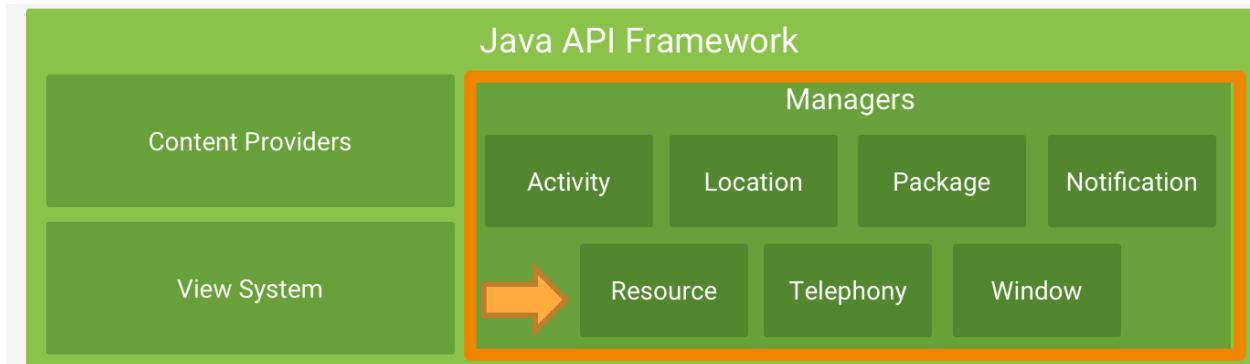


Java API Framework



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

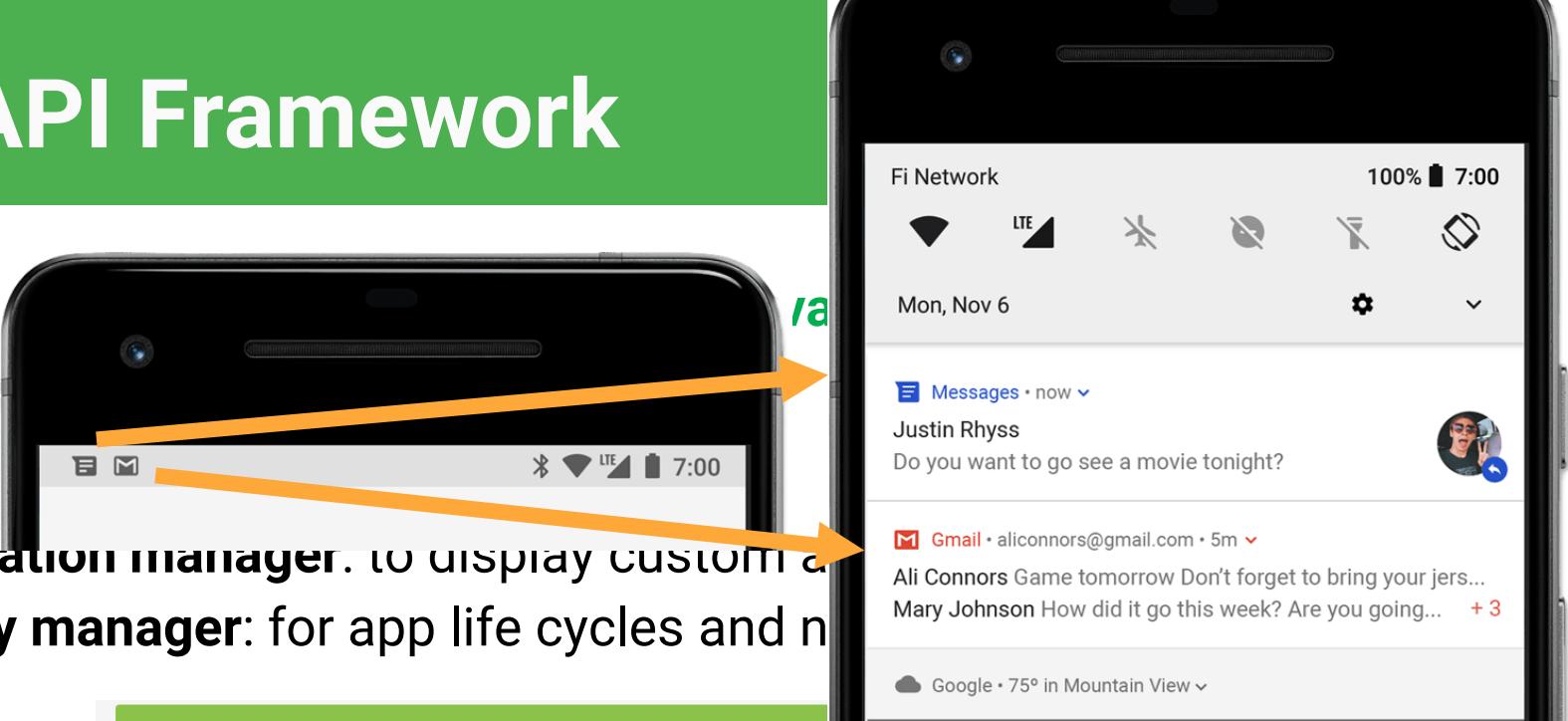
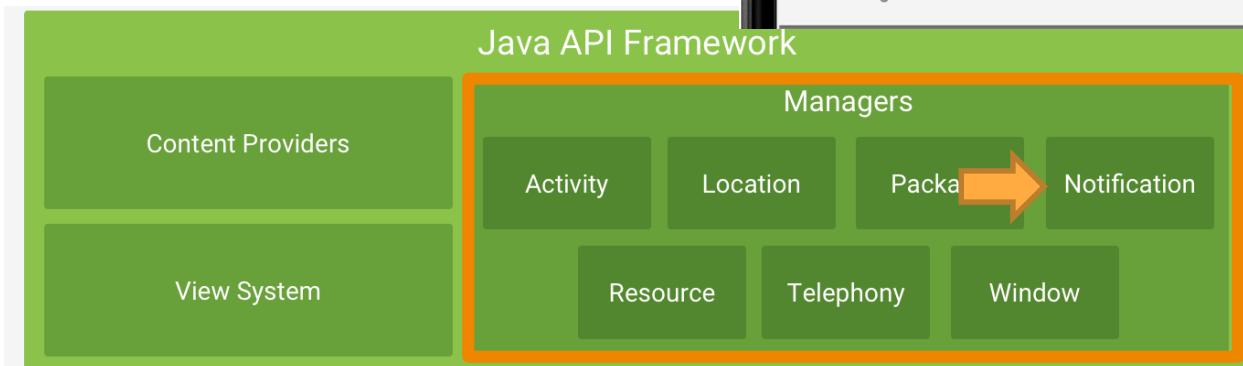
- **View system:** to create UI screens
- **Notification manager:** to display custom alerts in the status bar
- **Activity manager:** for app life cycles and navigation
-



Java API Framework

The entire framework is written in the Java language.

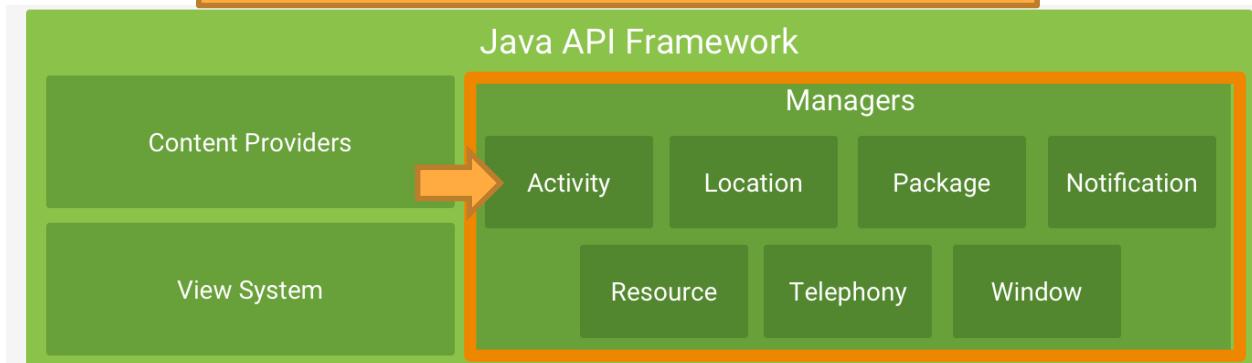
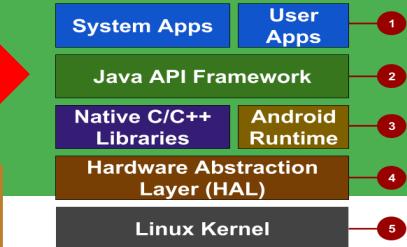
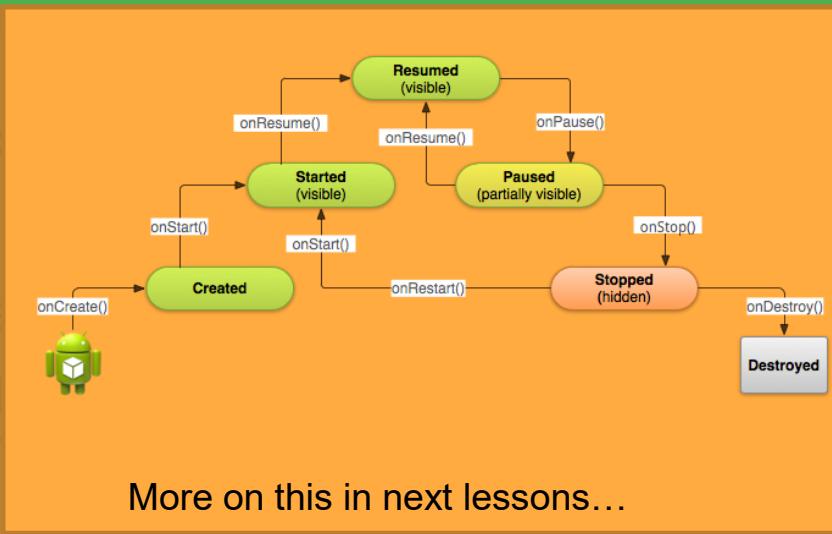
- **View system**: for displaying UI
- **Notification manager**: to display custom notifications
- **Activity manager**: for app life cycles and navigation
-



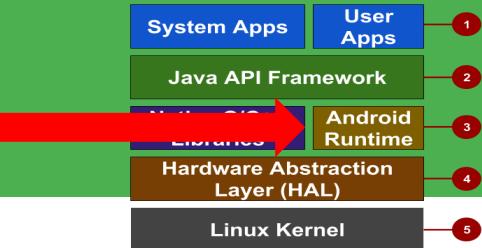
Java API Framework

The entire **feature-set** of the OS
written in the Java language

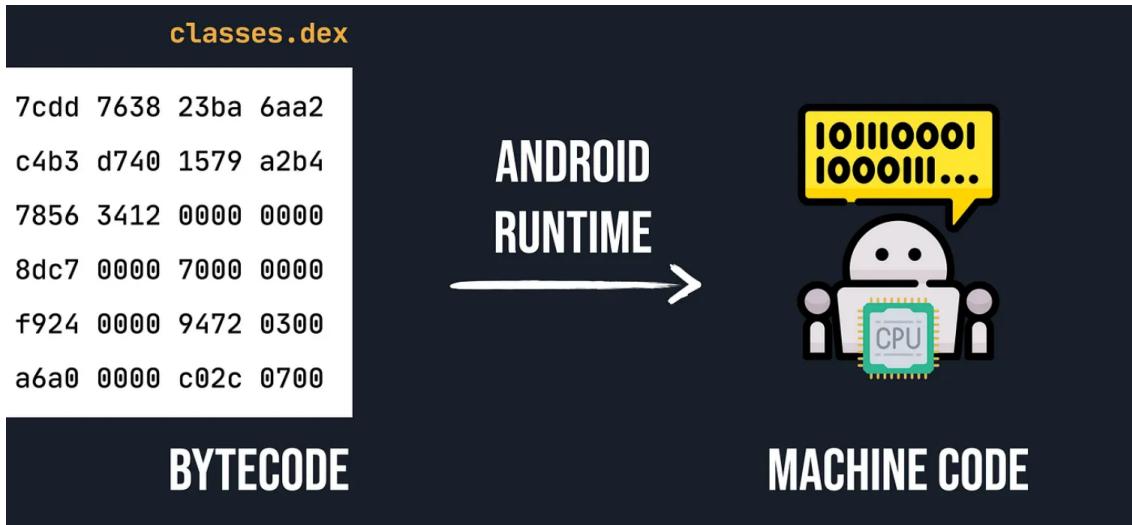
- **View system:** to create UI
- **Notification manager:** to show in status bar
- **Activity manager:** for activities
-



Android runtime

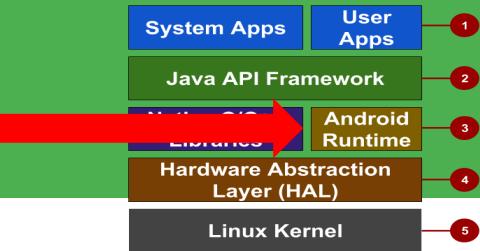


When we build our app and generate APK, part of that APK are **.dex files**.



When a user runs our app the bytecode written in **.dex files** is translated by **Android Runtime** into the machine code

Android runtime



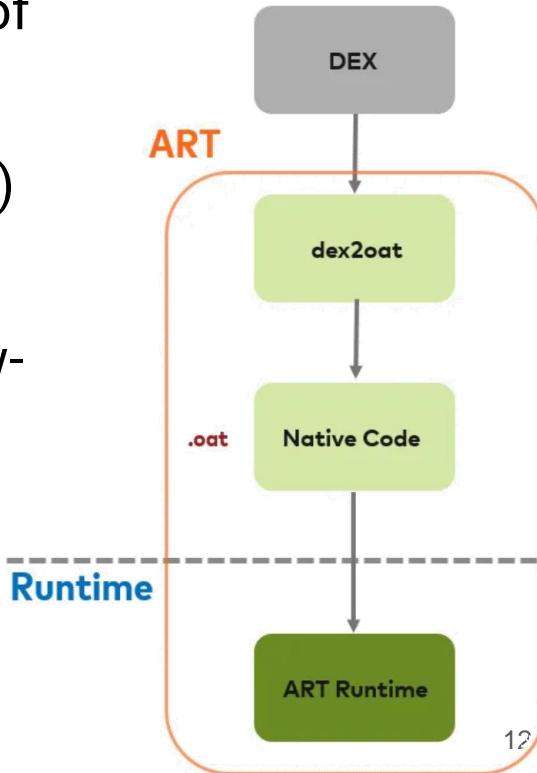
Each app **runs** in *its own process* with its own instance of the Android Runtime (ART)

- For devices running Android version 5.0 (API level 21) or higher
- ART is written to run multiple virtual machines on low-memory devices

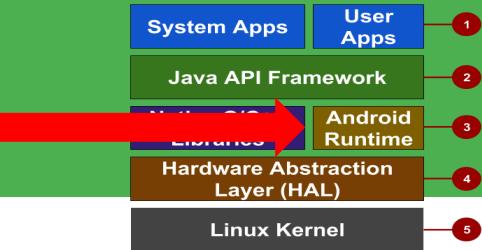
Just an overview, more precise info here:

<https://source.android.com/docs/core/runtime>

<https://source.android.com/docs/core/runtime/configure>



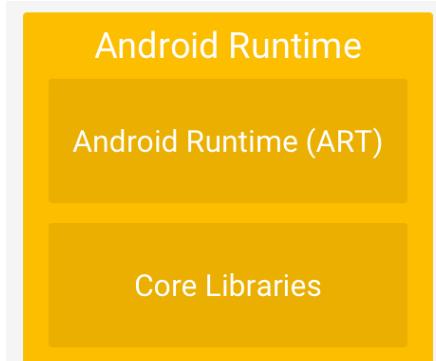
Android runtime



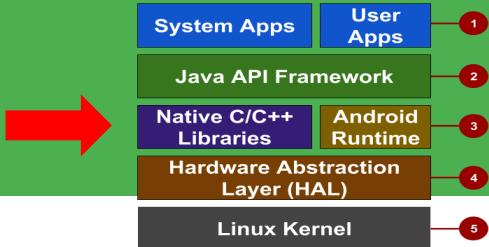
Each app **runs** in *its own process* with its own instance of the Android Runtime (ART)

- For devices running Android version 5.0 (API level 21) or higher
- ART is written to run multiple virtual machines on low-memory devices

Android also includes a set of **core runtime libraries** that provide most of the functionality of the Java programming language

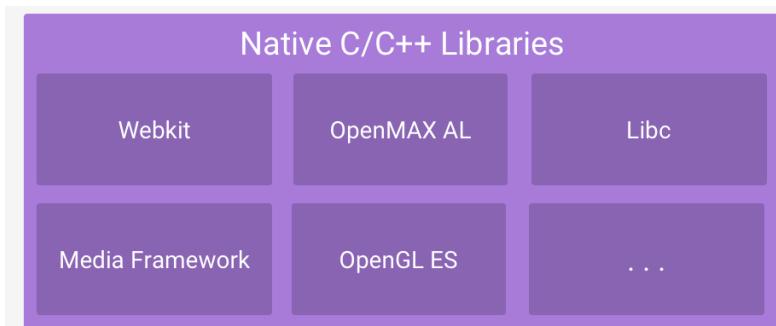


C/C++ libraries

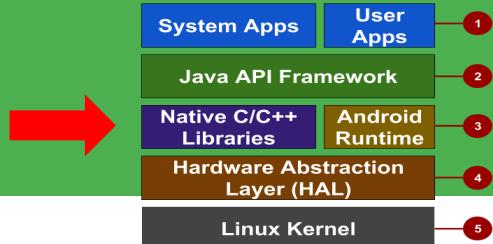


Core C/C++ Libraries give **access** to **core native Android system** components and services

For example, it is possible to access OpenGL ES through the Android framework's Java OpenGL API to *add support for drawing and manipulating 2D and 3D graphics* in your app

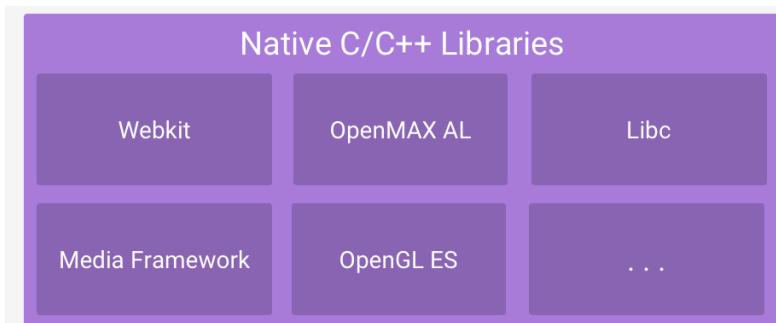


C/C++ libraries

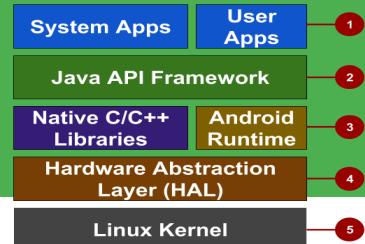


Core C/C++ Libraries give **access** to **core native Android system** components and services

For example, **it is possible to access OpenGL ES** through the Android framework's Java OpenGL API **to add support for drawing and manipulating 2D and 3D graphics** in your app



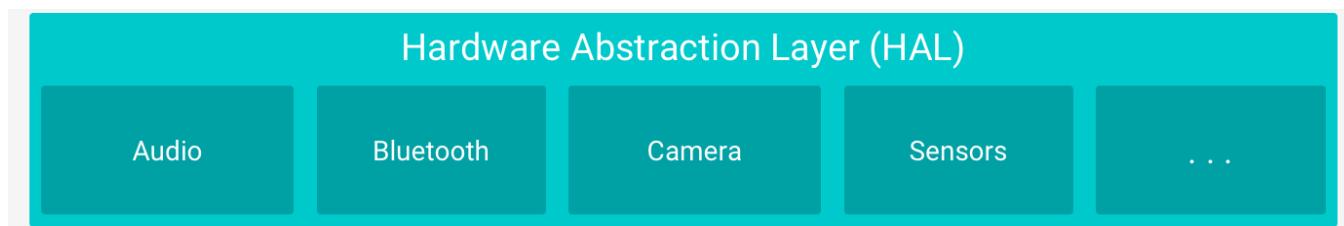
Hardware Abstraction Layer



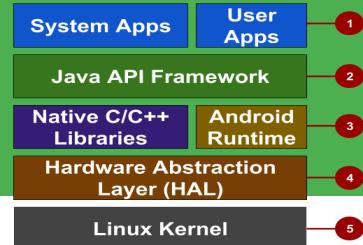
- Standard interfaces that **expose** device **hardware capabilities** as libraries

Examples: Camera, Bluetooth module, sensors

- when a framework API makes a call to access device hardware, the Android system loads the library module for that hardware component

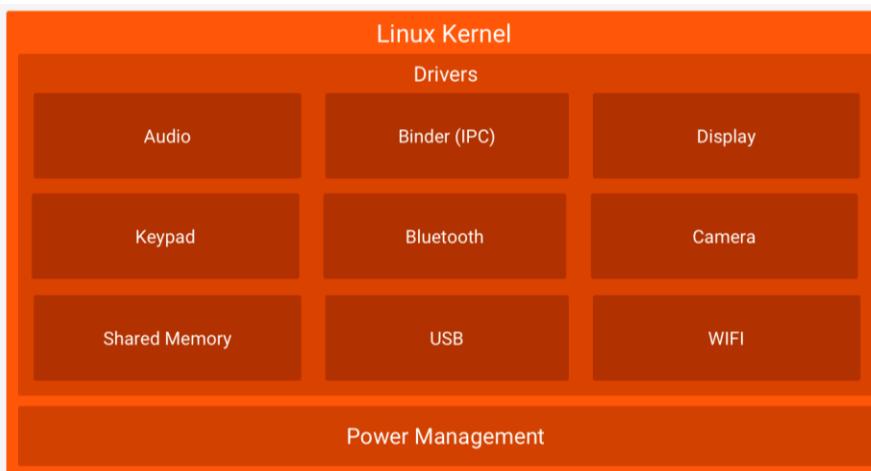


Linux Kernel



The **foundation** of the **Android platform** is the **Linux kernel**. Features include:

- **Threading and low-level memory management**
 - The Android Runtime (**ART**) relies on the Linux kernel for underlying functionalities such as threading and low-level memory management
- **Security**
 - Using a Linux kernel allows Android to take advantage of **key security features**
- **Drivers**
 - Using a Linux kernel allows device manufacturers to **develop hardware drivers for a well-known kernel**

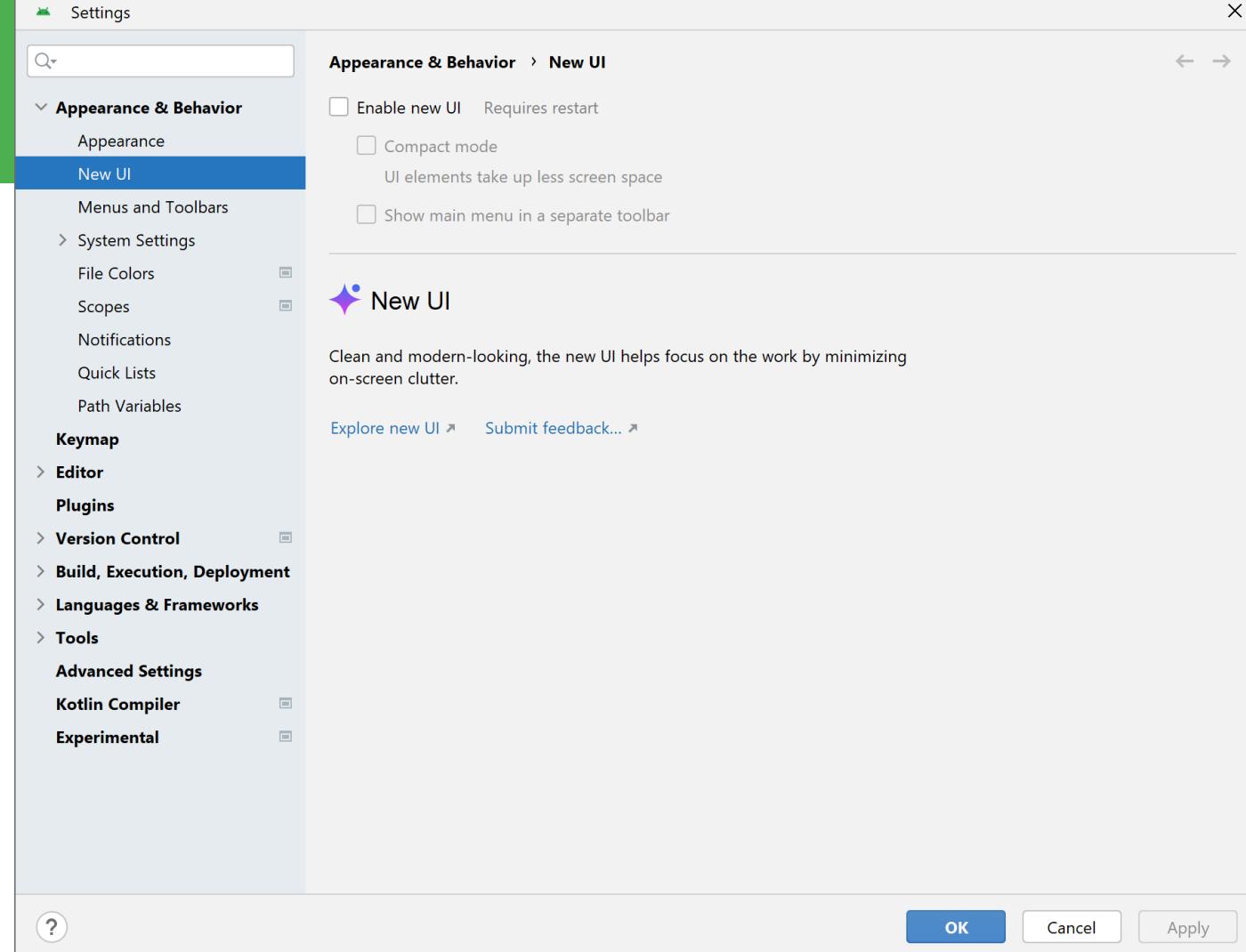


Logging in Android

To see the Android UI
as in the slide go to

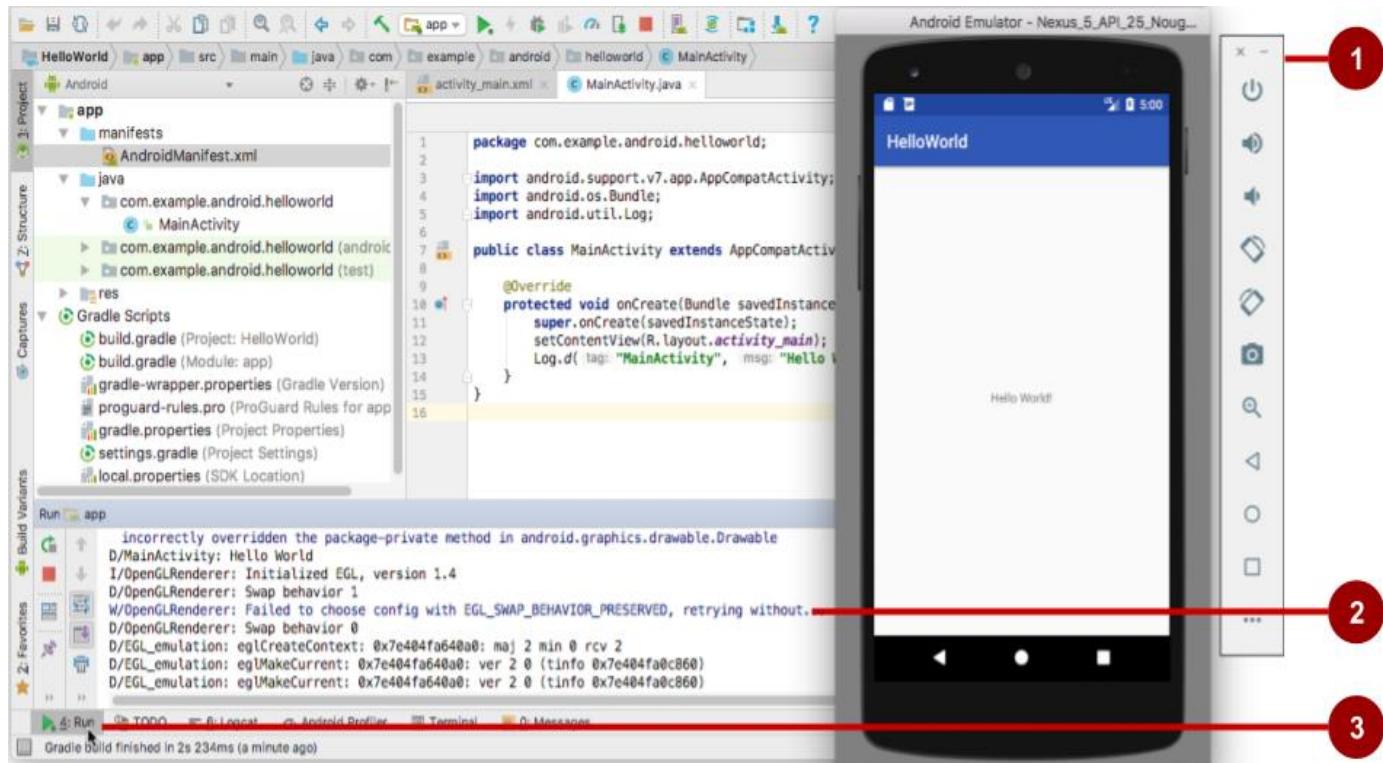
File -> Settings

And disable the “New
UI”



Get feedback as your app runs

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



LogCat pane

Run pane reports some messages but ***cannot be configured***....

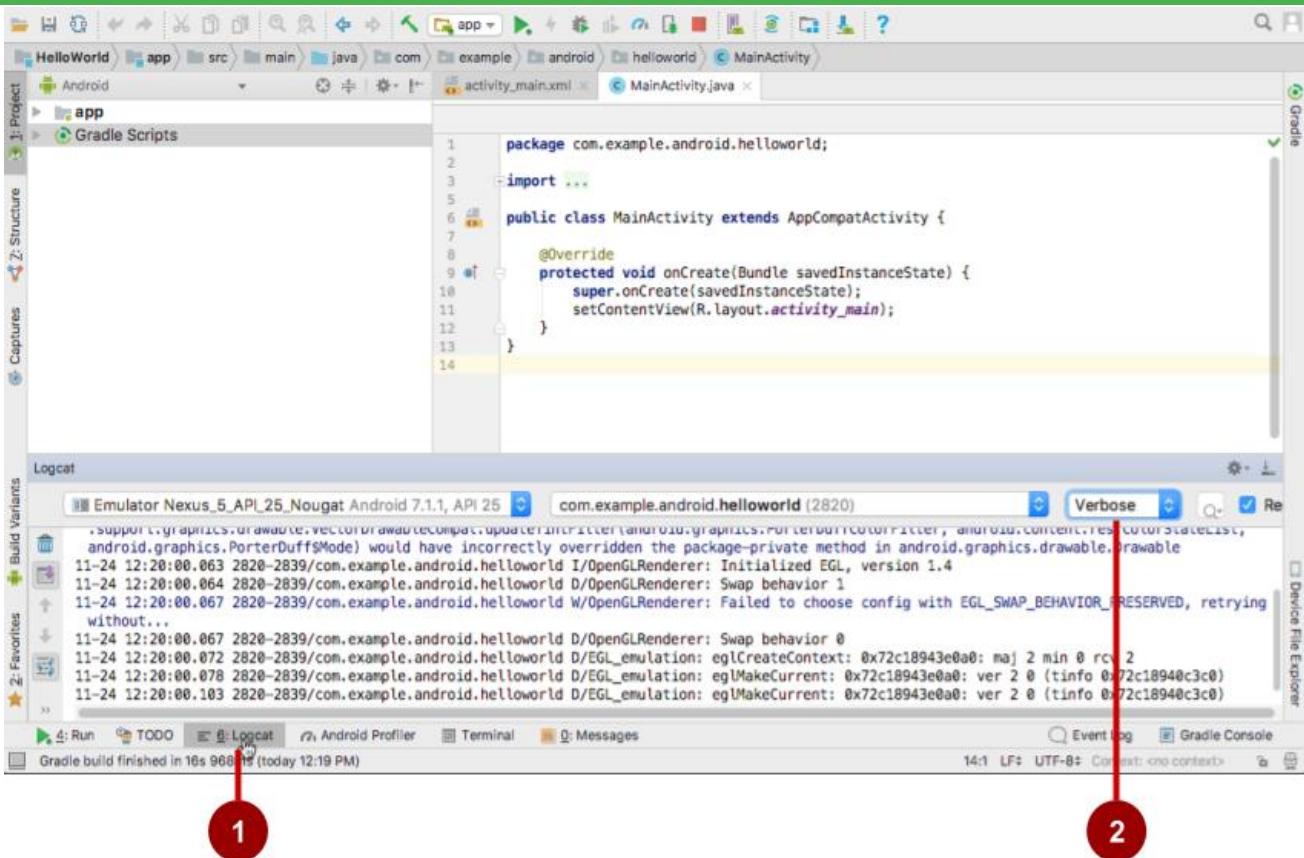
Instead, with the ***configurable Logcat pane*** it is possible to create custom view of:

- **system messages**, such as when a garbage collection occurs
- **messages** added to the app with the ***Log class***

It **displays messages** in ***real time*** and keeps a history so you can view older messages

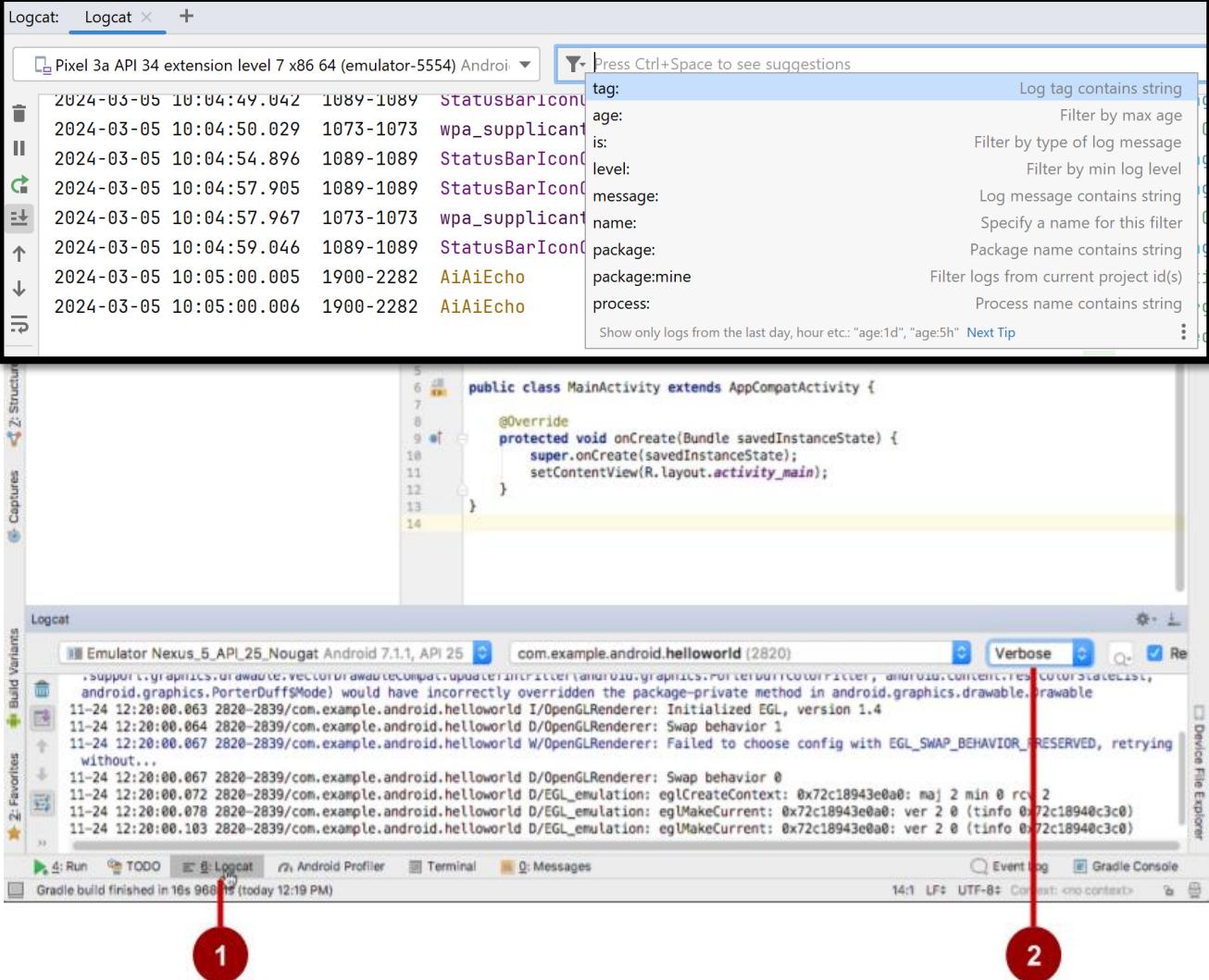
Logcat pane

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



Logcat pane

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



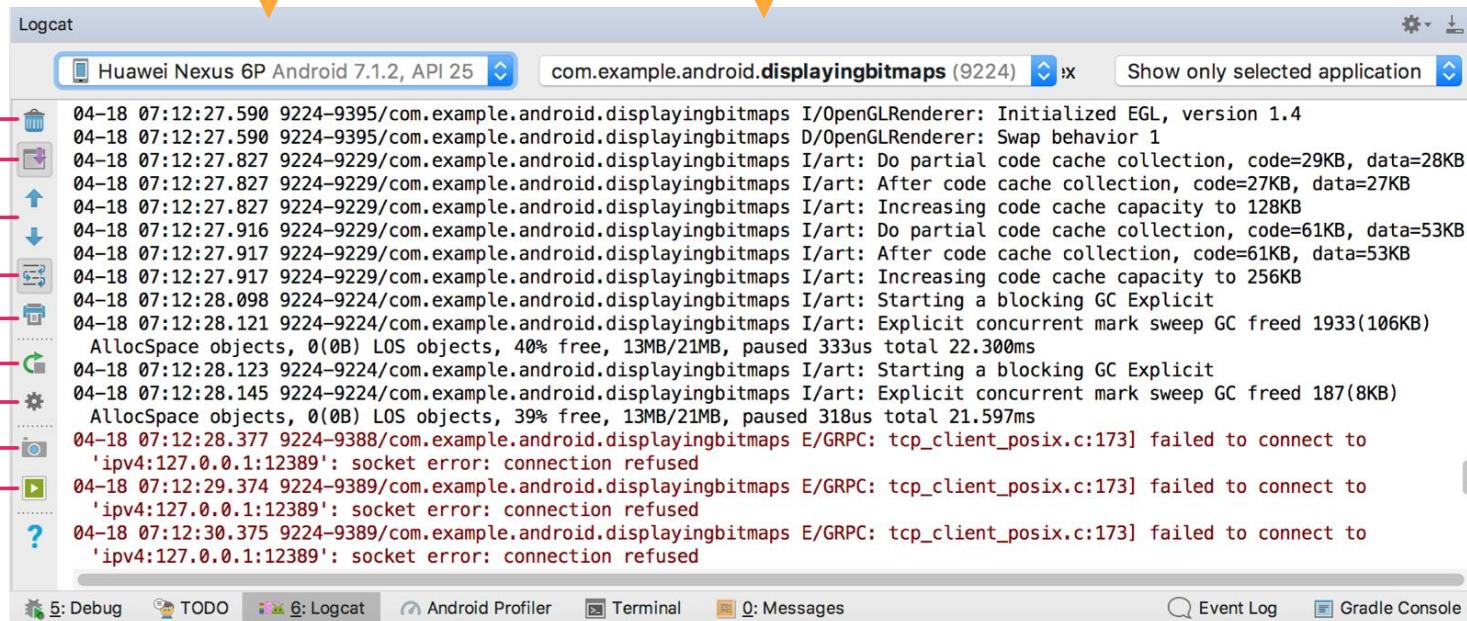
Logcat pane

Details of the LogCat pane

device/emulator running

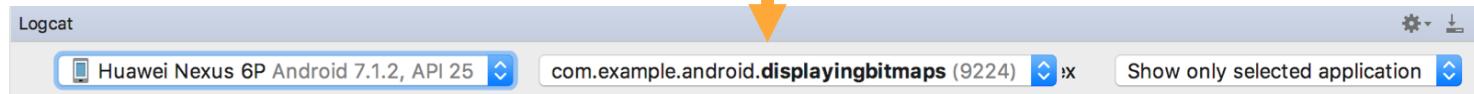


application running



Logcat pane

shows the log messages for the selected app, as selected from the dropdown lists at the top of the window



- 1 **Clear logcat** : Click to clear the visible log.
- 2 **Scroll to the end** : Click to jump to the bottom of the log and see the latest log messages. If you then click a line in the log, the view pauses scrolling at that point.
- 3 **Up the stack trace** and **Down the stack trace** : Click to navigate up and down the stack traces in the log, selecting the subsequent filenames (and viewing the corresponding line numbers in the editor) that appear in the printed exceptions. This is the same behavior as when you click on a filename in the log.
- 4 **Use soft wraps** : Click to enable line wrapping and prevent horizontal scrolling (though any unbreakable strings will still require horizontal scrolling).
- 5 **Print** : Click to print the logcat messages. After selecting your print preferences in the dialog that appears, you can also choose to save to a PDF.

Logcat pane

- 6 **Restart** : Click to clear the log and restart logcat. Unlike the **Clear logcat** button, this recovers and displays previous log messages, so is most useful if Logcat becomes unresponsive and you don't want to lose your log messages.
- 7 **Logcat header** : Click to open the **Configure Logcat Header** dialog, where you can customize the appearance of each logcat message, such as whether to show the date and time.
- 8 **Screen capture** : Click to [capture a screenshot](#). 
- 9 **Screen record** : Click to [record a video](#) of the device (for a maximum of 3 minutes).

Interesting additional functionalities

Write log messages

The **Log class** allows to **create log messages** that appear in logcat.

Use the following log methods, listed in order **from the highest to lowest priority** (or, least to most verbose):

- `Log.e(String, String)` (error)
- `Log.w(String, String)` (warning)
- `Log.i(String, String)` (information)
- `Log.d(String, String)` (debug)
- `Log.v(String, String)` (verbose)



Write log messages

The **Log class** allows to **create log messages** that appear in logcat.

Turn off logging and debugging

Make sure you deactivate logging and disable the debugging option before you build your application for release.

You can deactivate logging by removing calls to Log methods in your source files.

You can disable debugging by removing the android:debuggable attribute from the <application> tag in your manifest file, or by setting the android:debuggable attribute to false in your manifest file. Also, remove any log files or static test files that were created in your project.

<https://developer.android.com/studio/publish/preparing#turn-off-logging-and-debugging>

Write log messages

```
private const val TAG = "MyActivity"  
Log.<log-level>(TAG, "Message");
```

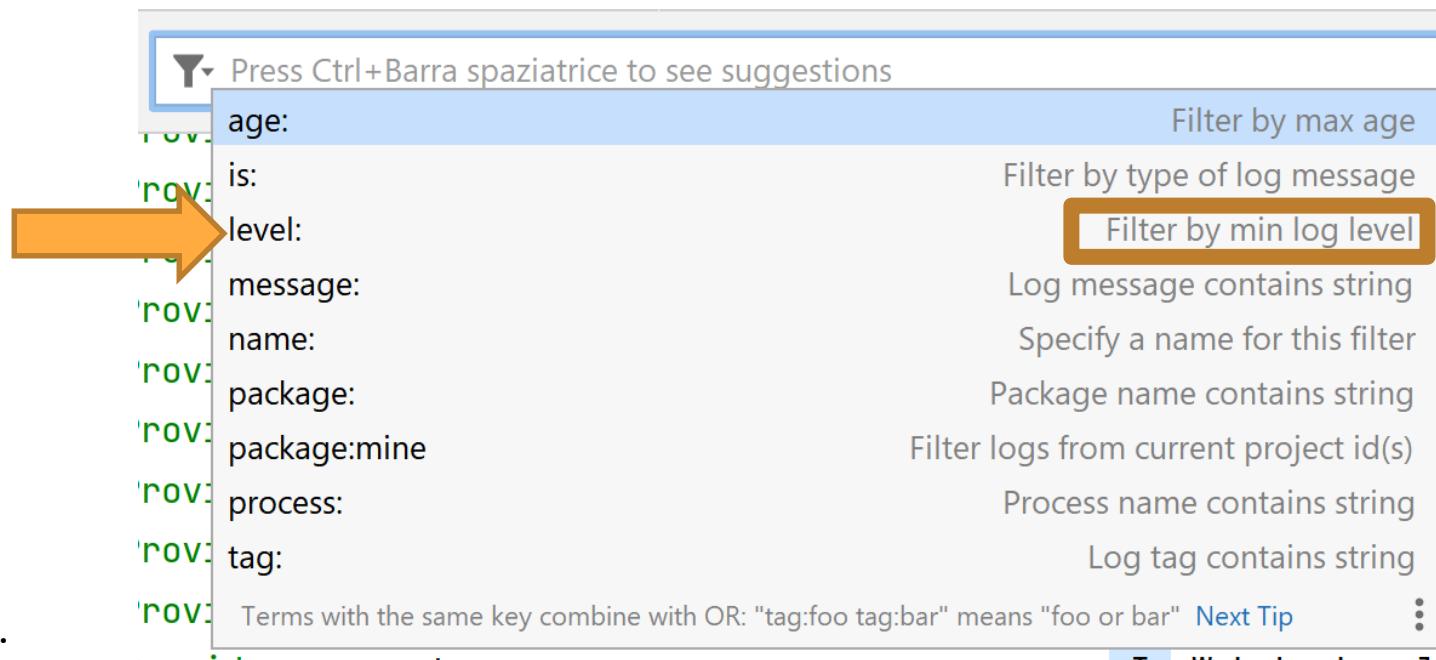
TAG: the first parameter should be a unique tag

a ***short string*** indicating the ***system component*** from which the ***message originates*** (for example, MyActivity). The tag can be any string that you find helpful, such as the name of the current class.

"Message" the second parameter is the message

Set the log level

It is possible to control how many messages appear in logcat by setting the log level. In the Log level menu, select one of the following values:



Set the log level

It is possible to control how many messages appear in logcat by setting the log level. In the Log level menu, select one of the following values:

- **Verbose:** Show all log messages (the default).
- **Debug:** Show debug log messages that are useful during development only, as well as the message levels lower in this list.
- **Info:** Show expected log messages for regular usage, as well as the message levels lower in this list.
- **Warn:** Show possible issues that are not yet errors, as well as the message levels lower in this list.
- **Error:** Show issues that have caused errors, as well as the message level lower in this list.
- **Assert:** Show issues that the developer expects should never happen.

Set the log level

It is possible to control how many messages appear in logcat by setting the log level. In the Log level menu, select one of the following values:

- **Verbose:** Show all log messages (the default).
- **Debug:** Show debug log messages that are useful during development only, as well as the message levels lower in this list.
- **Info:** Show expected log messages for regular usage, as well as the message levels lower in this list.
- **Warn:** Show possible issues that are not yet errors, as well as the message levels lower in this list.
- **Error:** Show issues that have caused errors, as well as the message level lower in this list.
- **Assert:** Show issues that the developer expects should never happen.

??

Assert Level

There is an even higher priority than error:

Log.**e()** will simply log an error to the log
with priority ERROR

What a Terrible Failure: Log.**wtf()**

Report a condition that should never happen.

Log an error with priority level ASSERT, and may (depending on the system configuration) send an error report and terminate the program immediately



Adding logging to your app

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

Logging statement

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

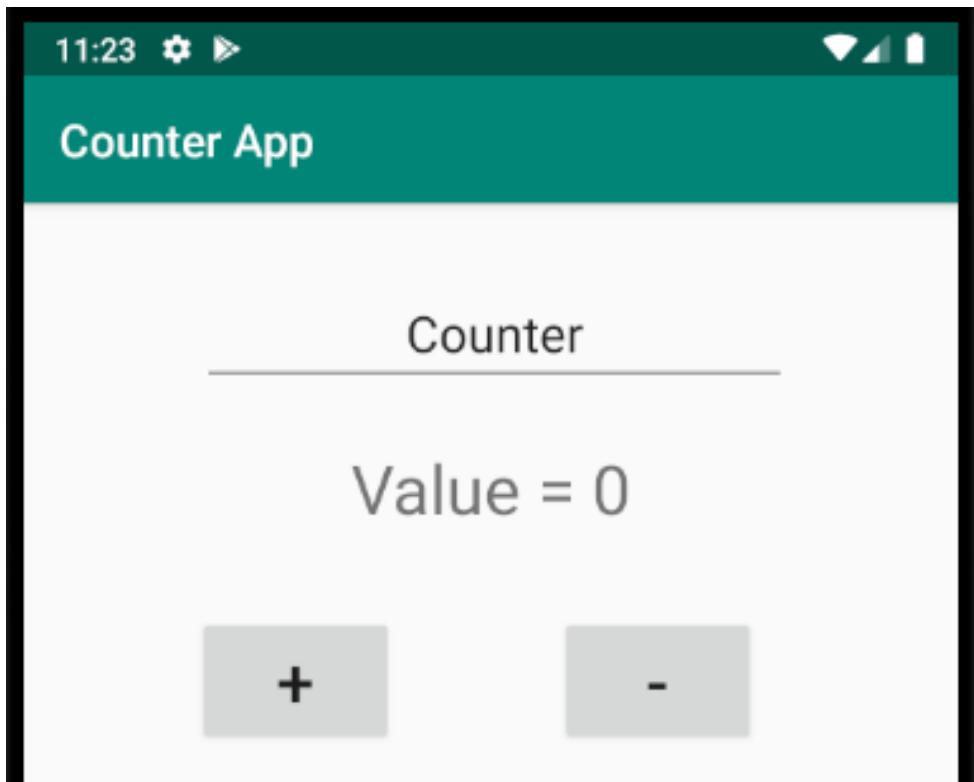
Exercise Counter App

Develop a Counter App

- **+** increases of 1 the value of the counter
- **-** decreases of 1 the value of the counter

initial value = 0

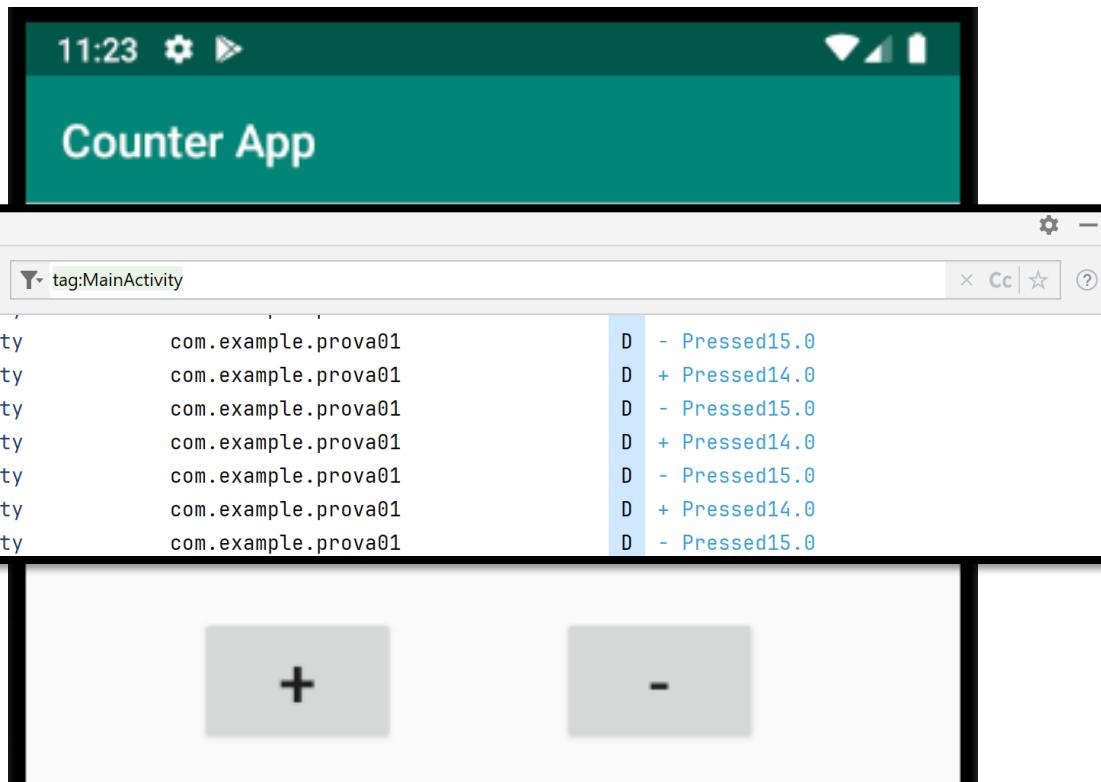
Log the taps on the two buttons



Exercise Counter App

Develop a Counter App

- **+** increases of 1 the value
- **-** decreases the value



initial value = 0



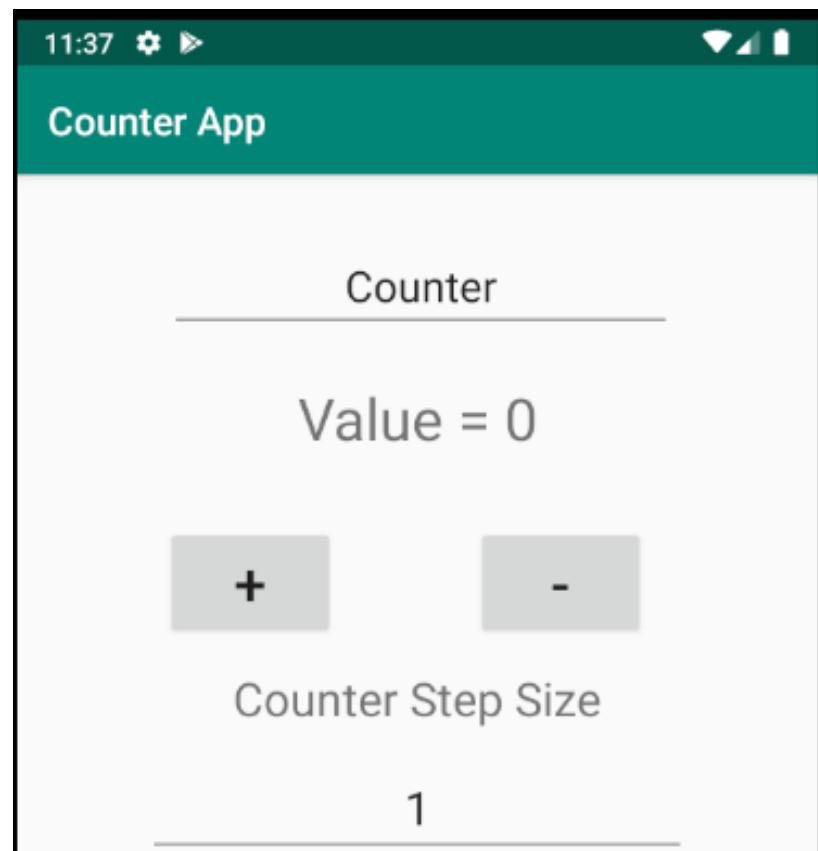
Log the taps on the two buttons

Exercise Counter App (Continue)

Now the user can insert the step for the counter:

e.g., 1 or 2 or 7.56

Constraint the value to numbers only....



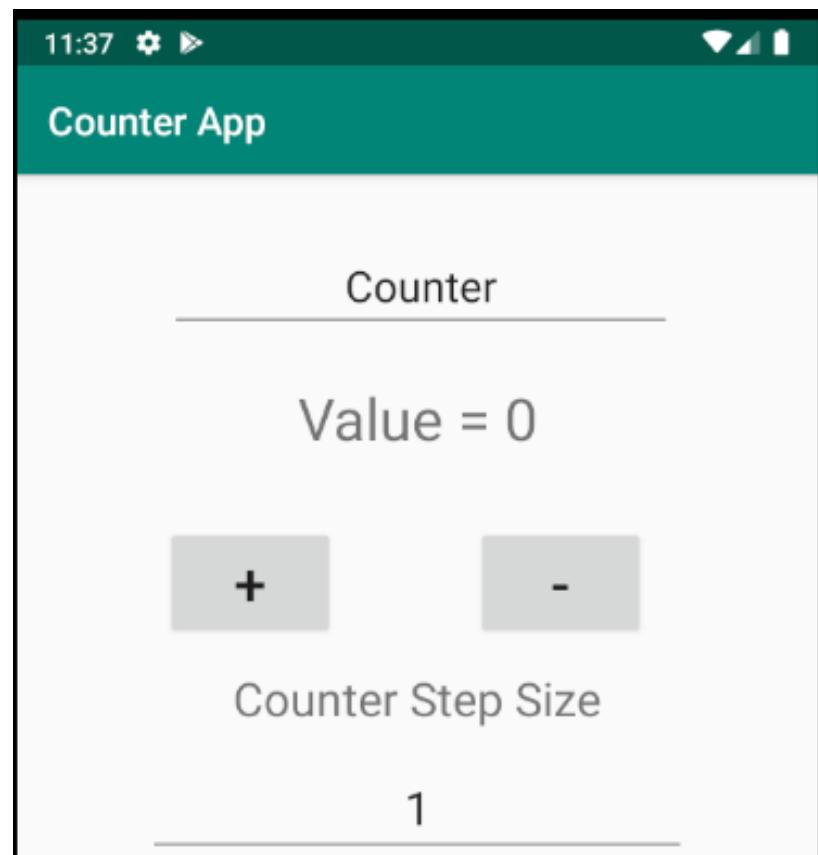
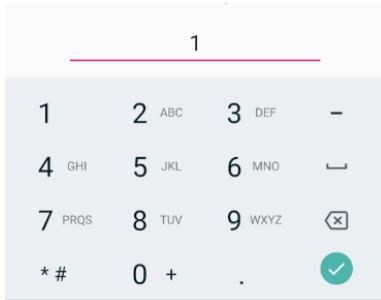
Exercise Counter App (Continue)

Now the user can insert the step for the counter:

e.g., 1 or 2 or 7.56

Constraint the value to numbers only....

Hint: set the
inputType
property





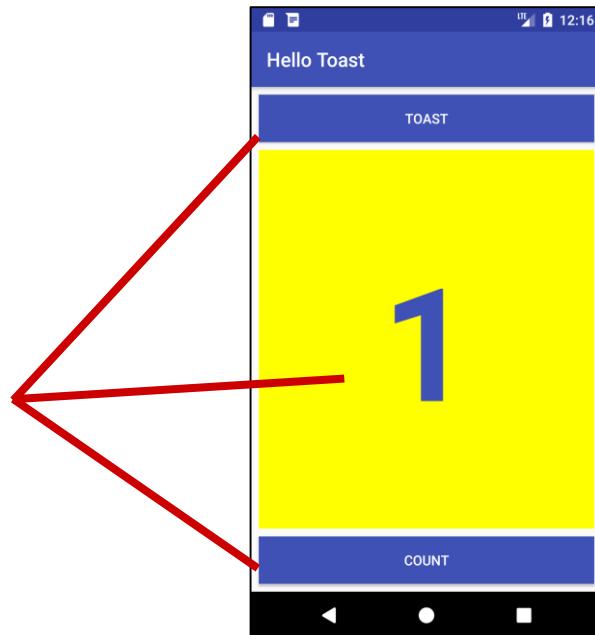
This work is licensed under a [Creative Commons Attribution 4.0 International License](#).

Views, view groups, and view hierarchy

Everything you see is a view

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

UI consists of a hierarchy of objects
called **views**



These slides are partially based on the material that Google provides for the course
Android Developer Fundamentals



What is a view?

View subclasses are basic user interface building blocks

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



What is a view?

View subclasses are basic user interface building blocks

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



View attributes

- Location (i.e., positioning)
 - expressed as a pair of left and top coordinates
- Dimensions
 - two dimensions, expressed as a width and a height
- The unit for location and dimensions is the density-independent pixel (dp)



View attributes

There are a lot of attributes depending on the View type:

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



View attributes

There are a lot of attributes depending on the View type:

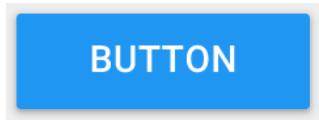
- Color
- May have focus (e.g., selected to receive user input)
- May be interactive (respond to user clicks)
 - Label vs Button
- May be visible or not
- Relationships to other views

*View subclass
specific attributes*

Examples of view subclasses

The Android system provides hundreds of predefined View subclasses. Commonly used View subclasses include:

Button



EditText



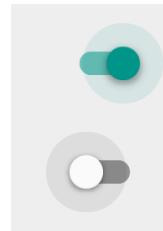
Slider



CheckBox



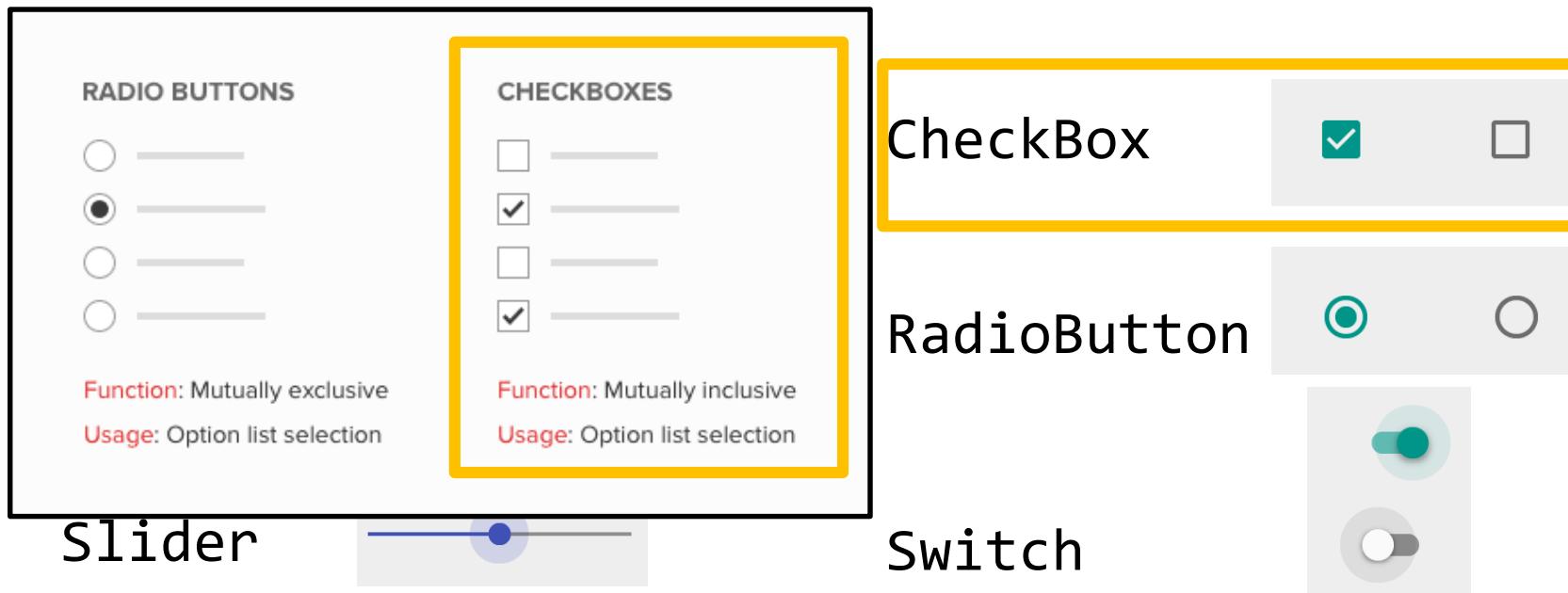
RadioButton



Switch

Examples of view subclasses

The Android system provides hundreds of predefined View subclasses. Commonly used View subclasses include:



RADIO BUTTONS

Function: Mutually exclusive
Usage: Option list selection

CHECKBOXES

Function: Mutually inclusive
Usage: Option list selection

Slider

RadioButton

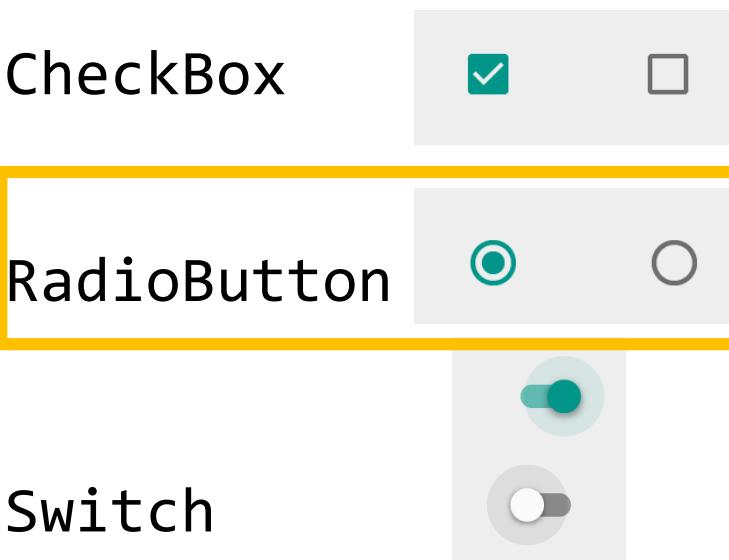
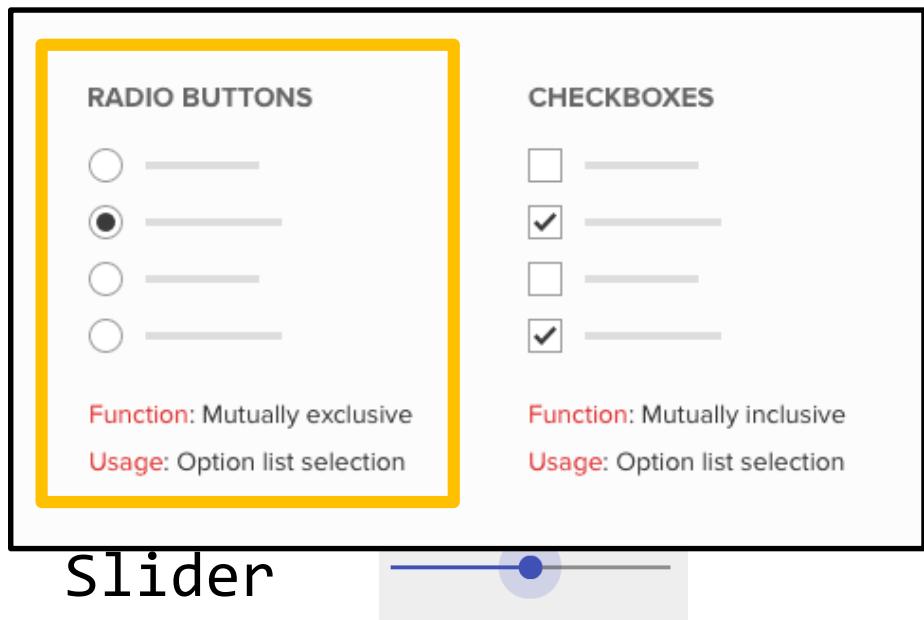
CheckBox

Switch

Detailed description: The diagram illustrates various Android View subclasses. It includes a section for 'RADIO BUTTONS' showing four radio buttons with one selected, and a 'Function' and 'Usage' note. A section for 'CHECKBOXES' shows five checkboxes with two checked, also with a 'Function' and 'Usage' note. Below these are three separate components: a 'Slider' with a blue track and a blue dot; a 'RadioButton' with two options, one selected; and a 'CheckBox' with two options, one checked. At the bottom right is a 'Switch' component with two states, one of which is active.

Examples of view subclasses

The Android system provides hundreds of predefined View subclasses. Commonly used View subclasses include:

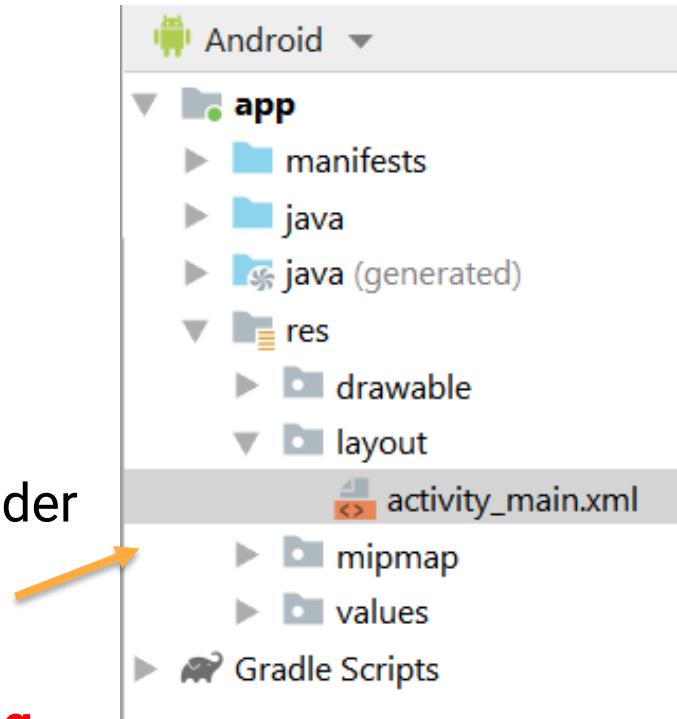


Create views and layouts

View elements can be **created in XML layout resource files**. How?

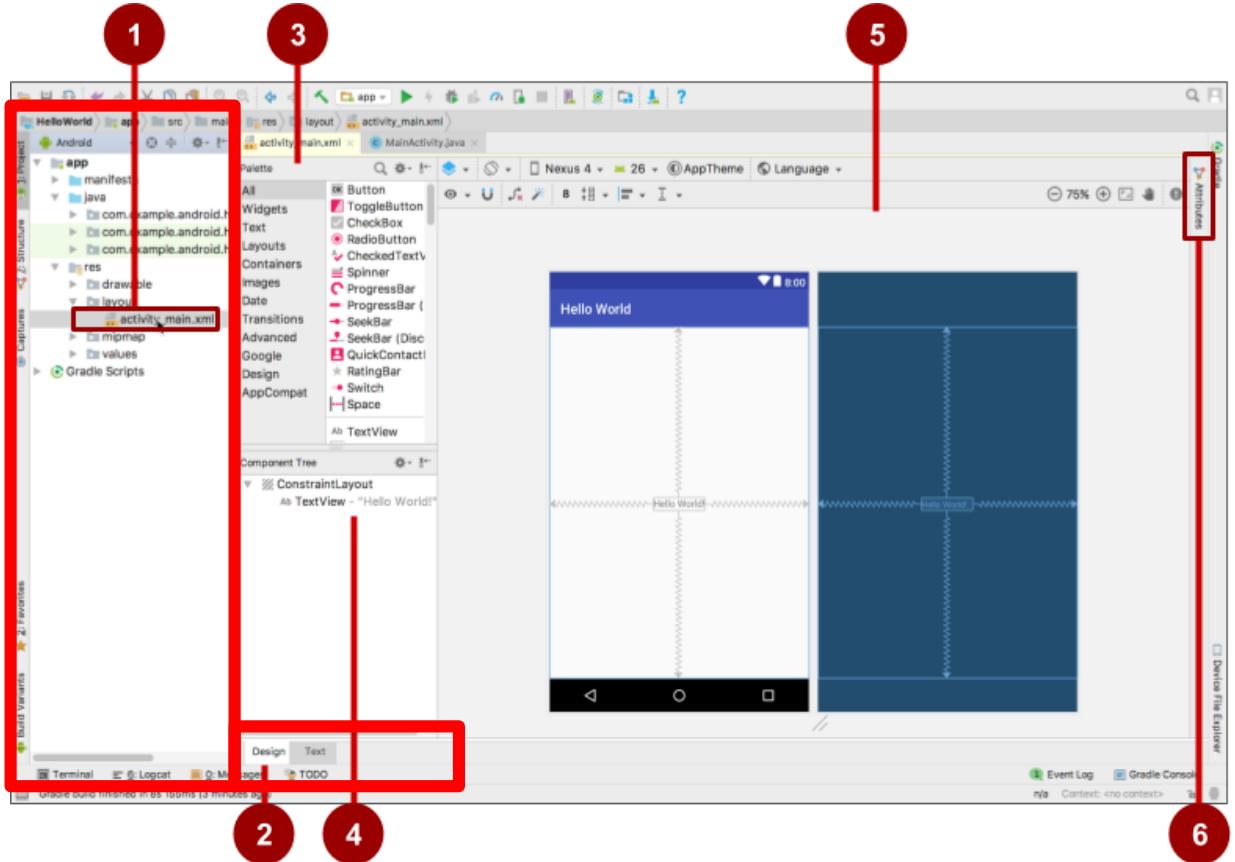
- Android Studio layout editor:
 - visual representation of XML
- XML editor

Layout resources are listed within the layout folder in the res folder in the Project > Android pane



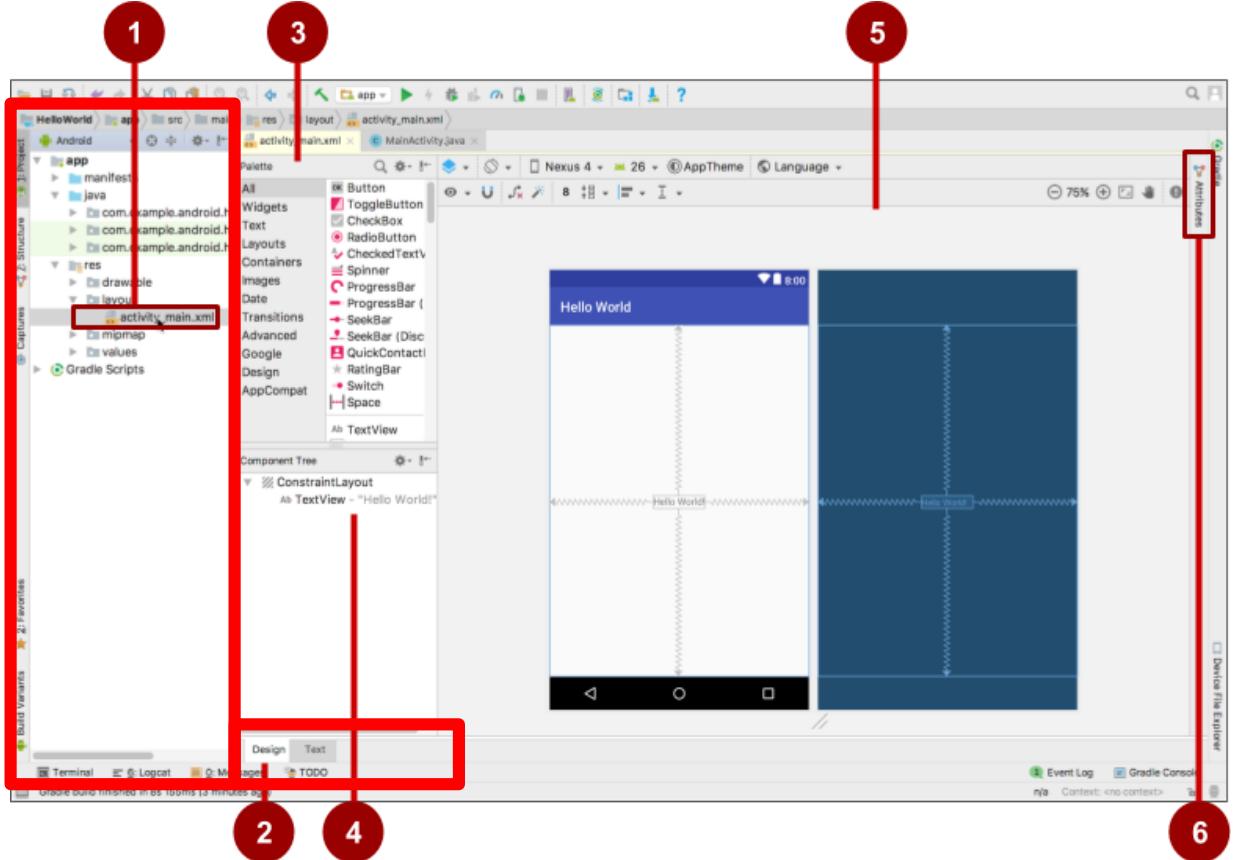
View can be also **created programmately, using Java code**

Android Studio layout editor



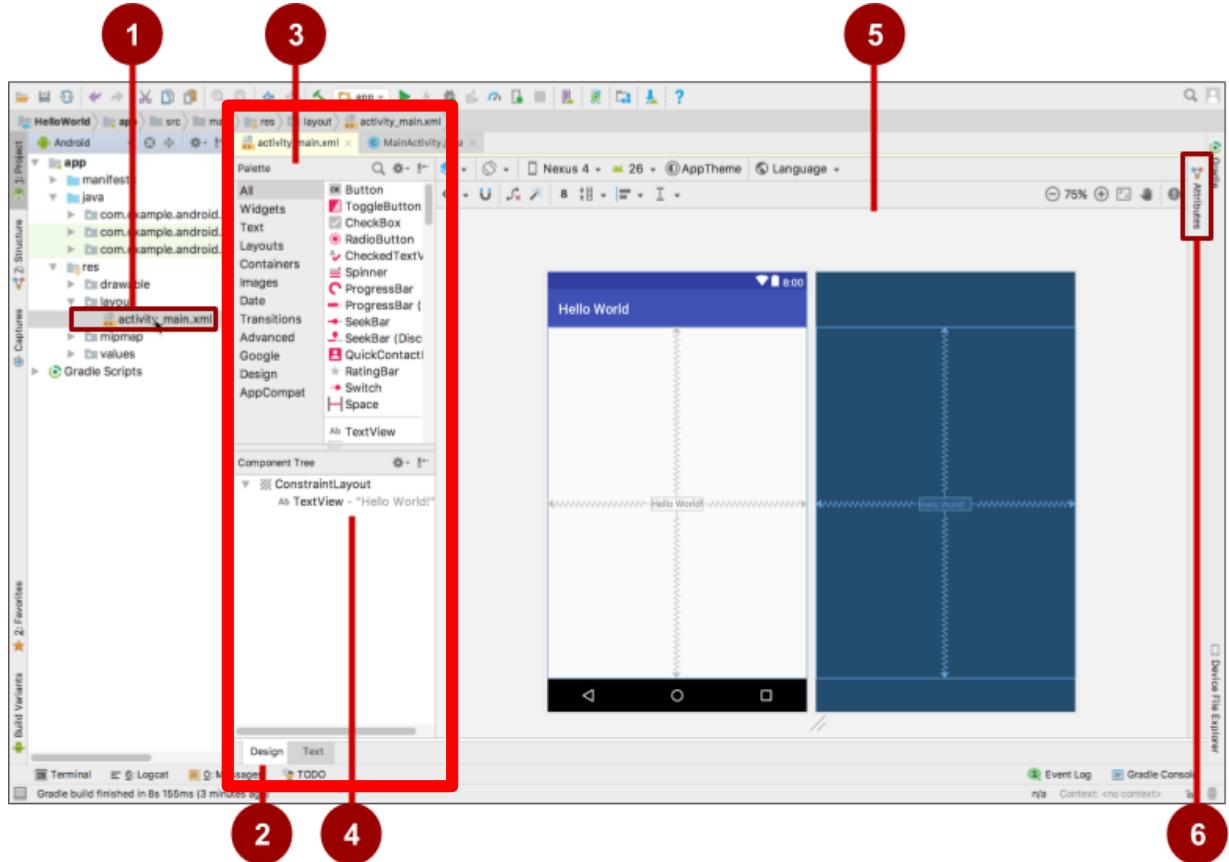
- 1. XML layout file**
- 2. Design and Text tabs**
- 3. Palette pane**
- 4. Component Tree**
- 5. Design and blueprint panes**
- 6. Attributes tab**

Android Studio layout editor



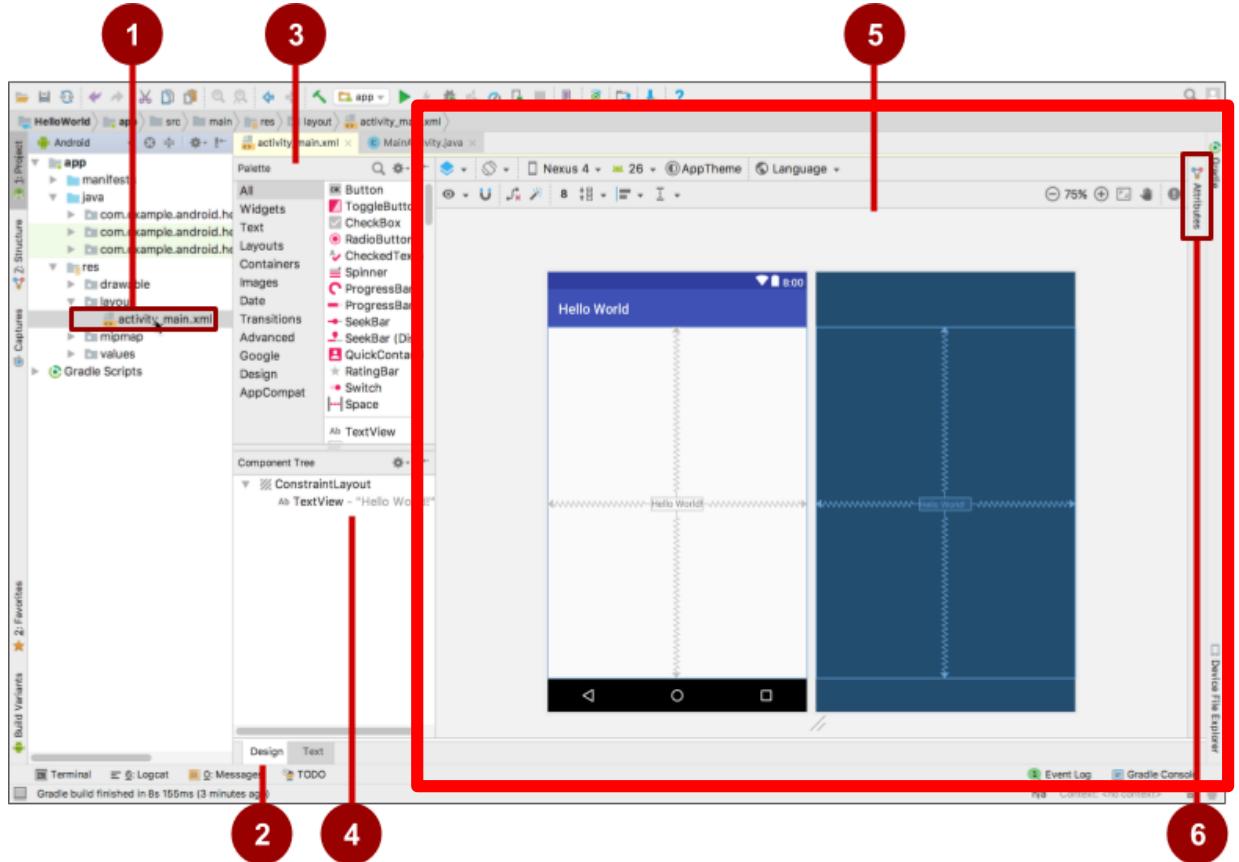
1. XML layout file
2. Design and Text tabs
3. Palette pane
4. Component Tree
5. Design and blueprint panes
6. Attributes tab

Android Studio layout editor



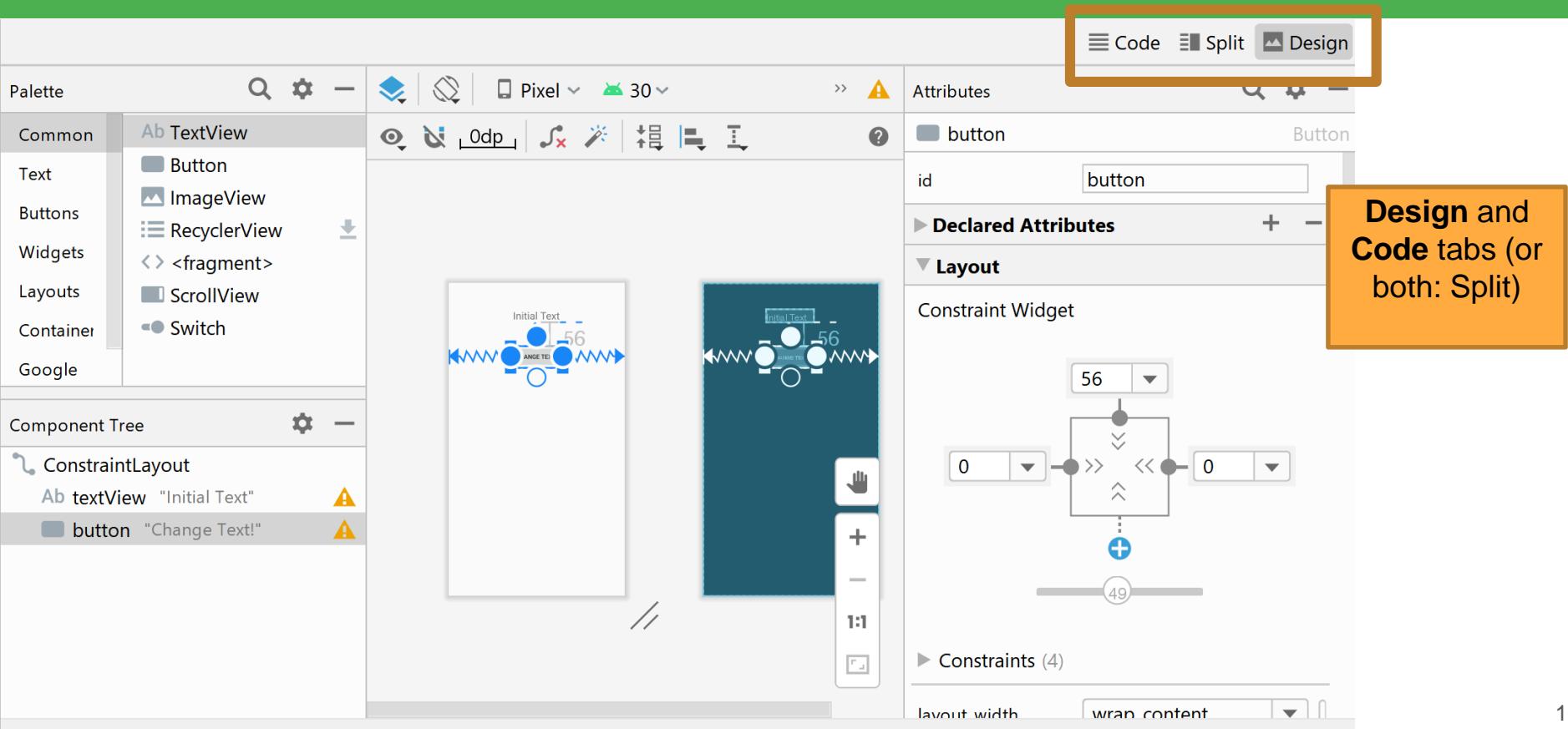
1. XML layout file
2. Design and Text tabs
3. Palette pane
4. Component Tree
5. Design and blueprint panes
6. Attributes tab

Android Studio layout editor



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

Android Studio layout editor



The screenshot shows the Android Studio Layout Editor interface. At the top right, there are three tabs: "Code", "Split", and "Design". The "Design" tab is highlighted with a yellow box around it. On the left, the "Palette" sidebar is open, showing categories like Common, Text, Buttons, Widgets, Layouts, Container, and Google, with "Common" currently selected. Below the palette is the "Component Tree" panel, which lists a ConstraintLayout containing a textView labeled "Initial Text" and a button labeled "Change Text!". The main workspace displays two views of a layout: a white view on the left and a dark teal view on the right. Both views show a blue constraint node with the text "Initial Text" and a dimension of "56". The layout editor's toolbar is visible at the top, and the bottom right corner features a large orange callout box with the text "Design and Code tabs (or both: Split)".

Design and Code tabs (or both: Split)



View defined in XML

```
<TextView
```

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

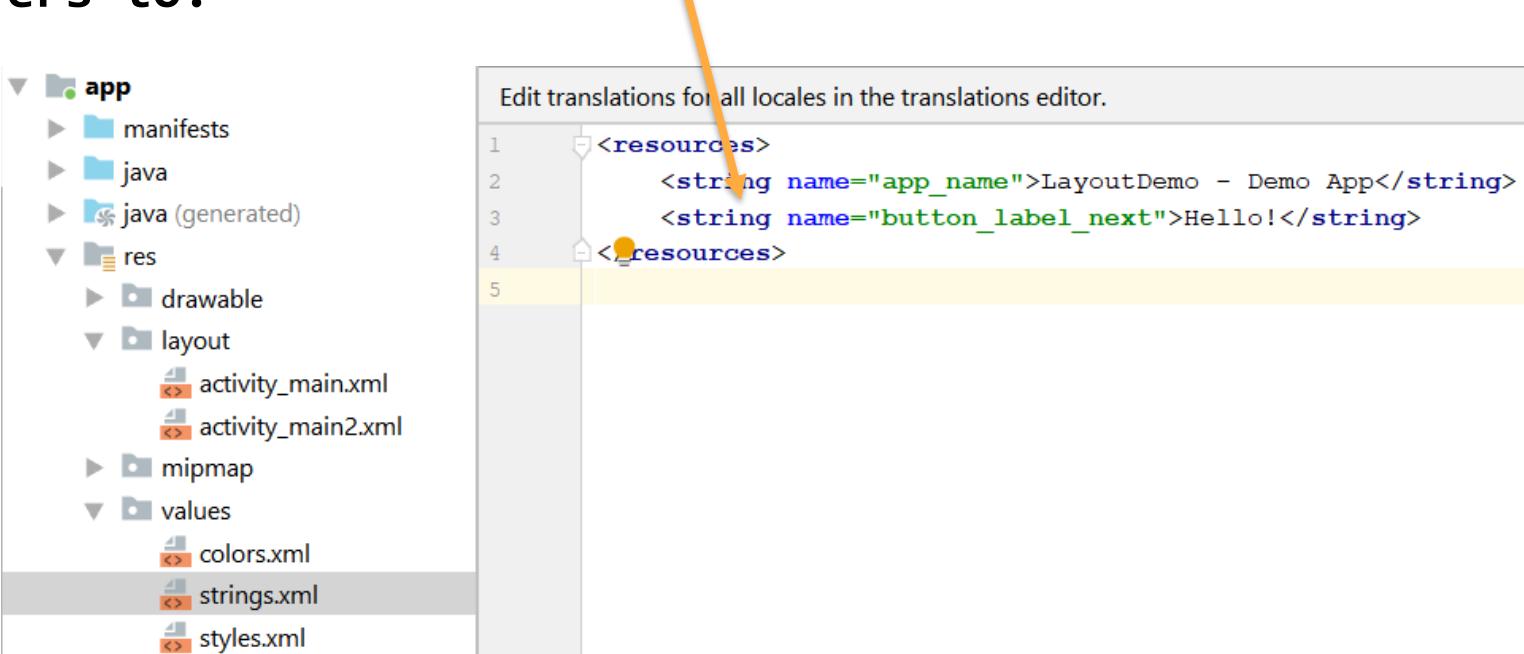
/>

View attributes in XML

android:<property_name>="@<resource_type>/resource_id"

Example: android:text="@string/button_label_next"

refers to:



The screenshot shows the Android Studio interface. On the left, the Project Structure sidebar displays the app module with its sub-directories: manifests, java, java (generated), res, and values. The values directory contains files for colors.xml, strings.xml, and styles.xml. The strings.xml file is currently selected, indicated by a grey background. On the right, the main editor window shows the contents of strings.xml:

```
1 <resources>
2   <string name="app_name">LayoutDemo - Demo App</string>
3   <string name="button_label_next">Hello!</string>
4 <resources>
5
```

An orange arrow points from the word "button_label_next" in the third line of the XML code to the "strings.xml" file in the Project Structure sidebar.



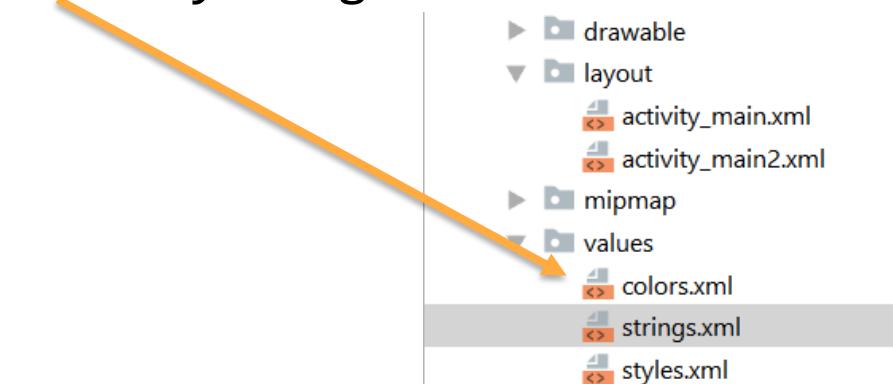
View attributes in XML

android:<property_name>=<property_value>

Example: android:layout_width="match_parent"
 android:textStyle="bold"

android:<property_name>="@<resource_type>/resource_id

Example: android:background="@color/myBackgroundColor"





Custom views

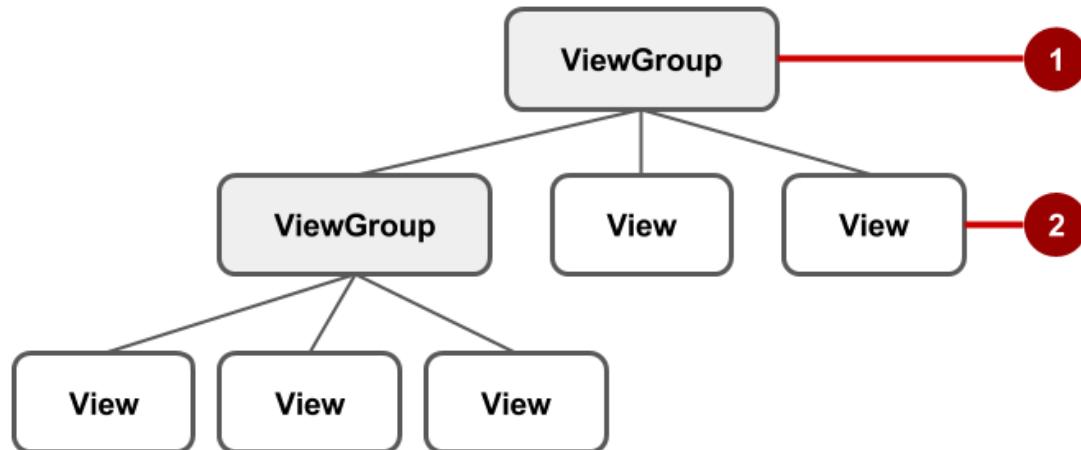
- Over 100 (!) different types of views available from the Android system, all children of the [View](#) class
- If necessary, [create custom views](#) by subclassing existing views or the View class

<https://developer.android.com/develop/ui/views/layout/custom-views/create-view#java>

View Group and View hierarchy

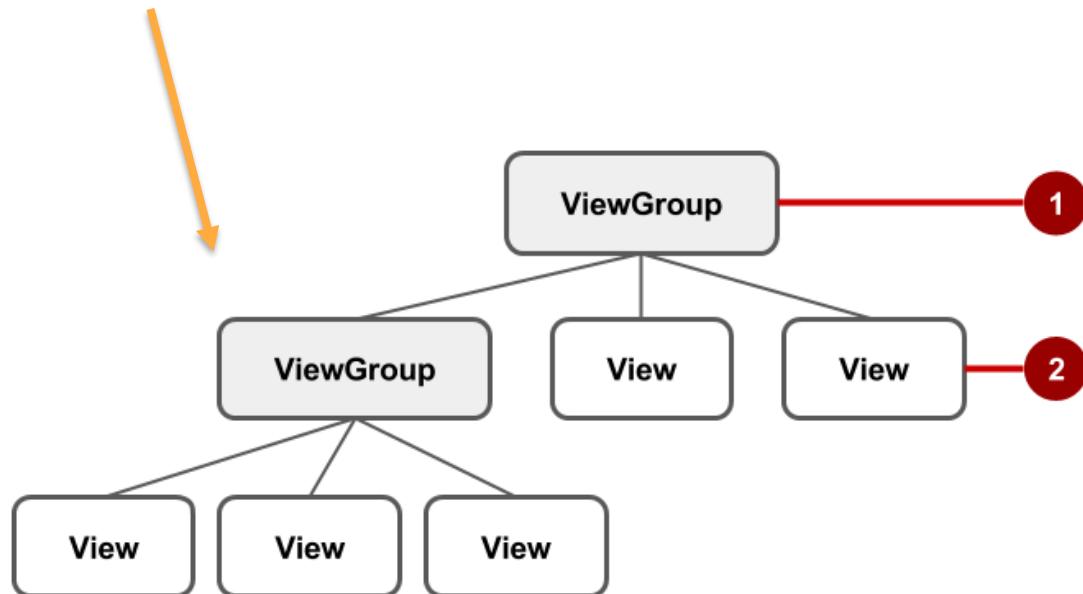
ViewGroup contains "child" views

- **View elements** for a screen **are organized** in a **hierarchy**
- At the root of this hierarchy there is a **ViewGroup (1)**
 - it contains the layout of the entire screen



ViewGroup contains "child" views

- ViewGroup can contain child View elements (**2**)
- or other ViewGroup(s)

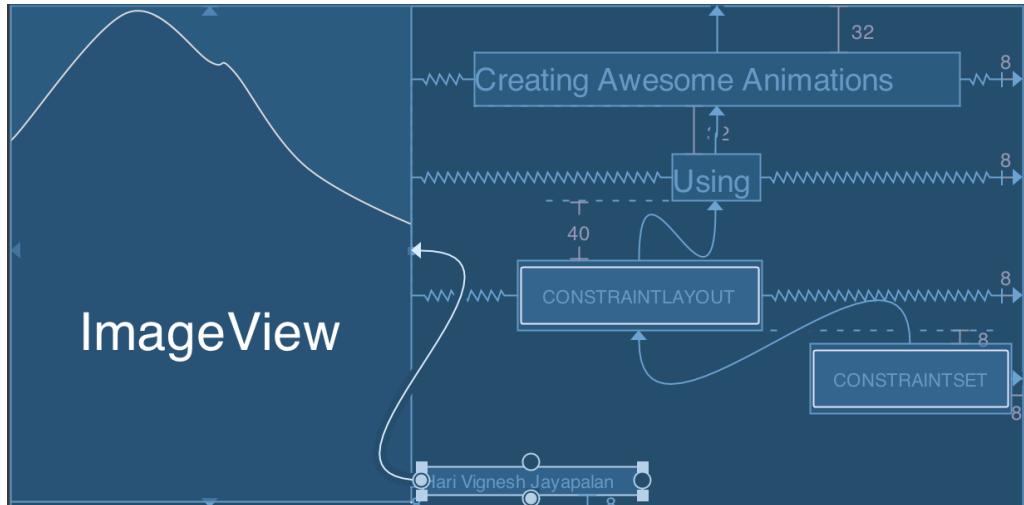


ViewGroup contains "child" views

The following are commonly used ViewGroup(s):

- ConstraintLayout: Positions UI elements using **constraint connections** to other elements and to the layout edges

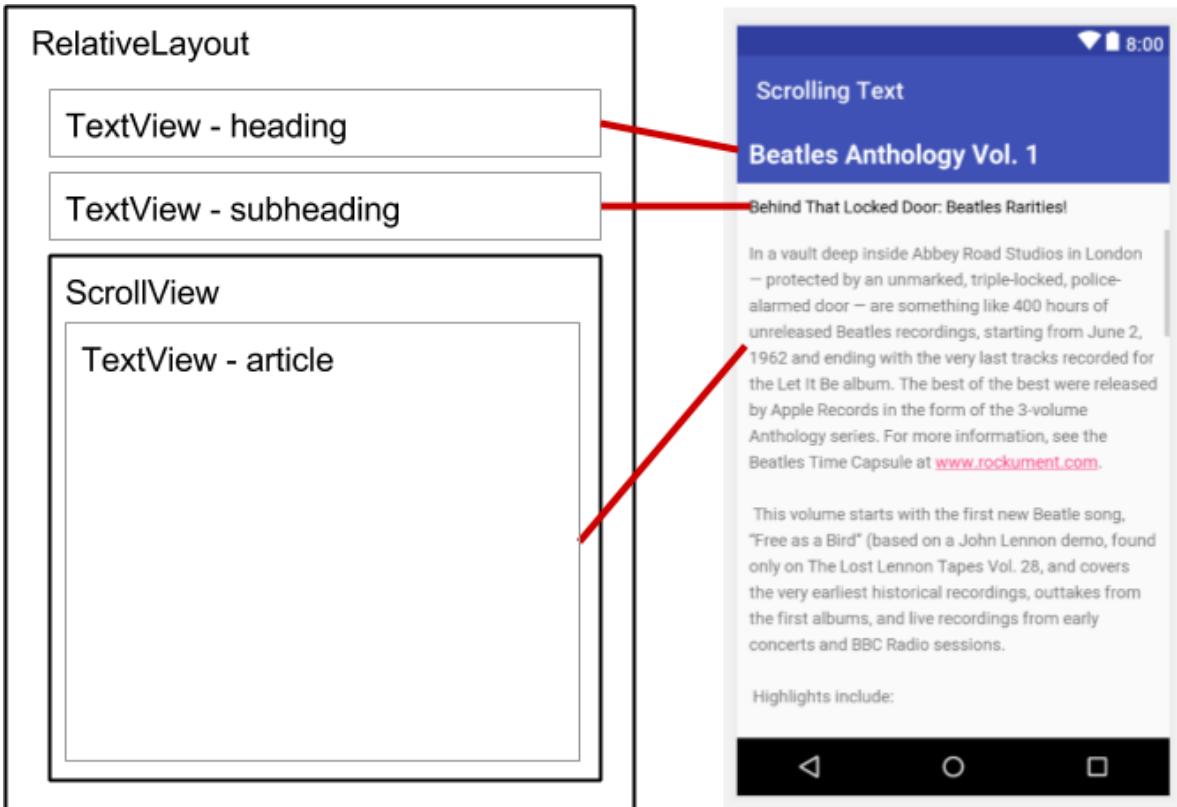
Default in Android Studio



ViewGroup contains "child" views

The following are commonly used ViewGroup(s):

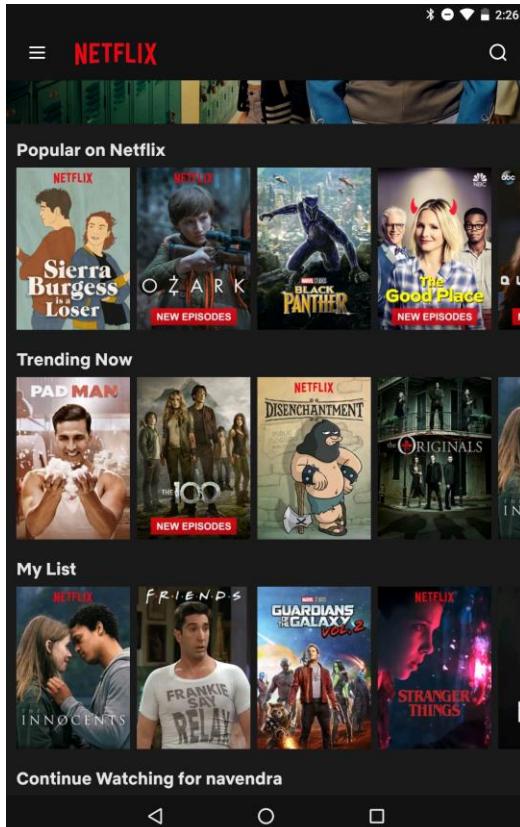
- [ScrollView](#):
Contains **one element** and **enables scrolling**



ViewGroup contains "child" views

The following are commonly used ViewGroup groups:

- RecyclerView: Contains a **list of elements** and **enables scrolling**
 - *Useful when the app needs to display a scrolling list of elements based on large data sets (or data that frequently changes)*



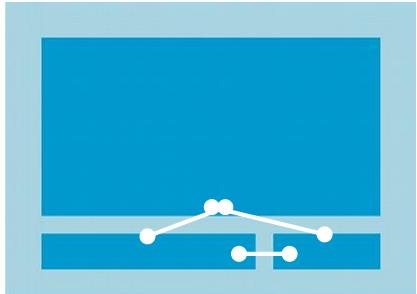


ViewGroups for layouts

Some **ViewGroup** groups are designated as *layouts*

- organize *child View elements* in a specific way
- are typically used as the root ViewGroup
- are specific types of ViewGroups (subclasses of [ViewGroup](#))
- can be in a row, column, grid, table, absolute

Common Layout Classes

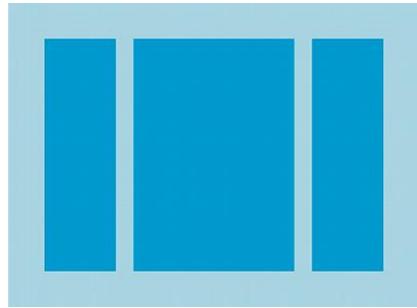


A group of child View elements **using constraints, edges, and guidelines to control** how the **elements are positioned relative to other elements in the layout**

ConstraintLayout

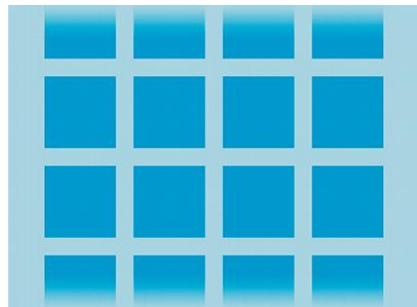
ConstraintLayout **was designed also to make it easy to click and drag View elements in the layout editor**

Common Layout Classes



LinearLayout

A group of **child View elements positioned and aligned horizontally or vertically**



GridLayout

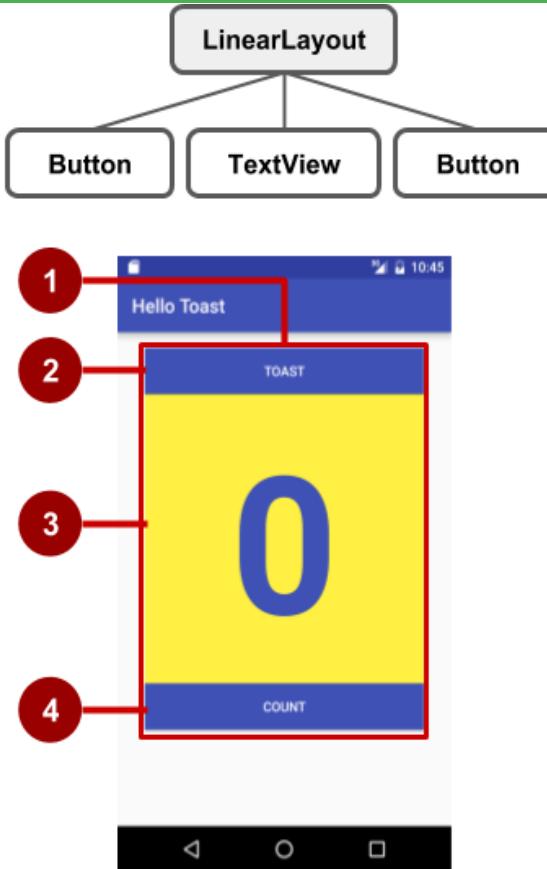
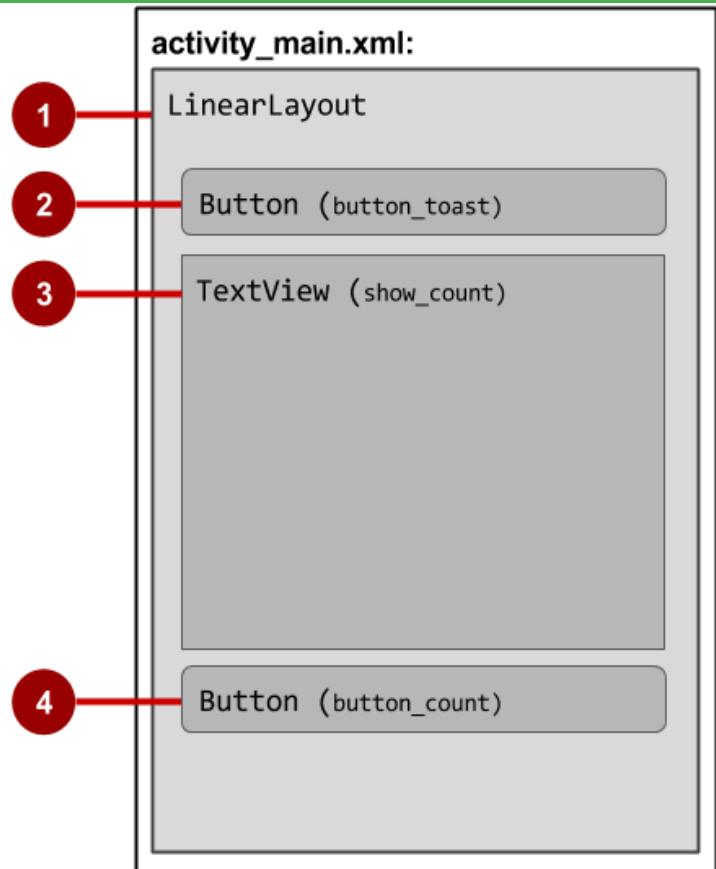
A group that places its **child View elements in a rectangular grid** that can be scrolled

Class hierarchy vs. layout hierarchy

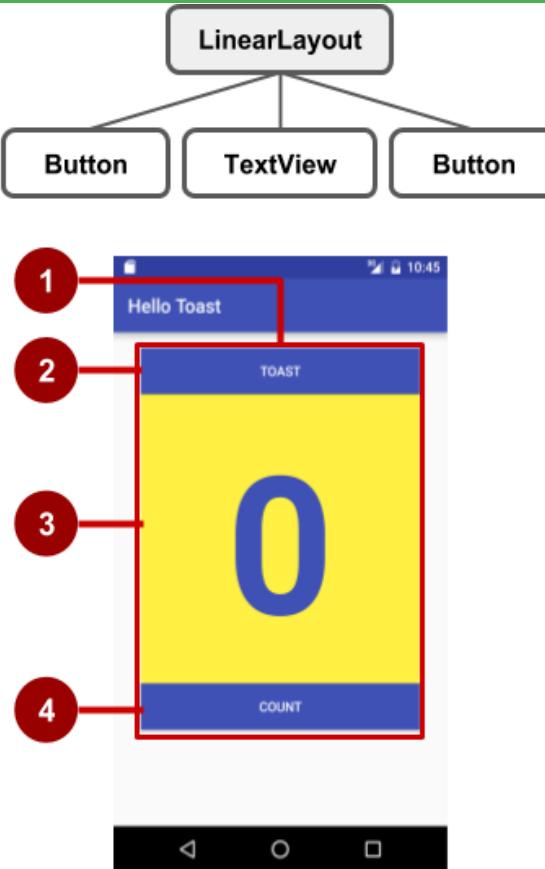
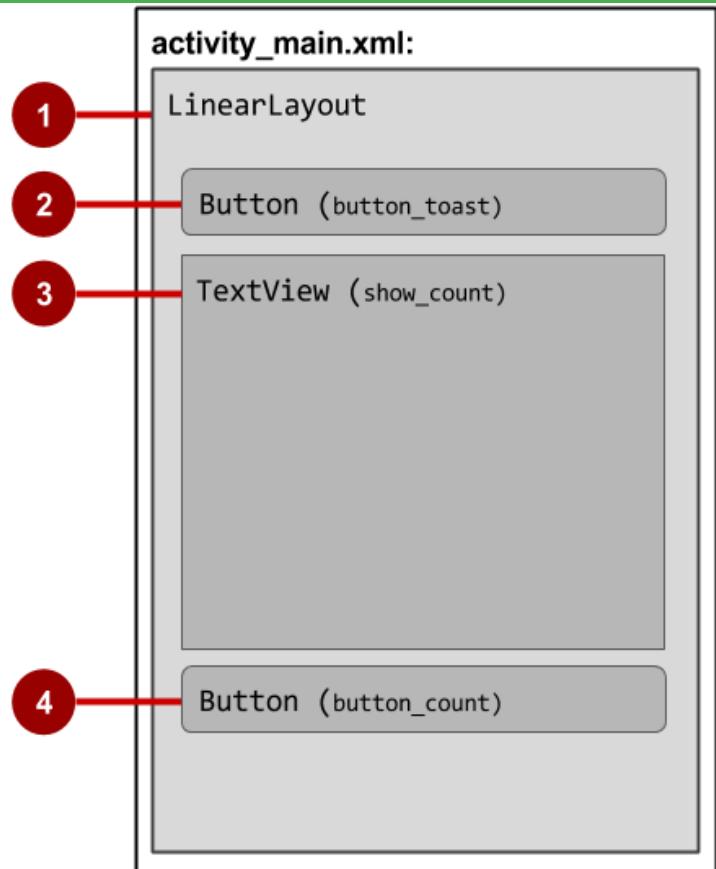
- View class-hierarchy is standard object-oriented class inheritance
 - For example, Button is-a TextView is-a View is-an Object
 - Superclass-subclass relationship
- Layout hierarchy is how views are visually arranged
 - For example, LinearLayout can contain Buttons arranged in a row
 - Parent-child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

```
java.lang.Object
↳ android.view.View
↳ android.widget.TextView
↳ android.widget.Button
```

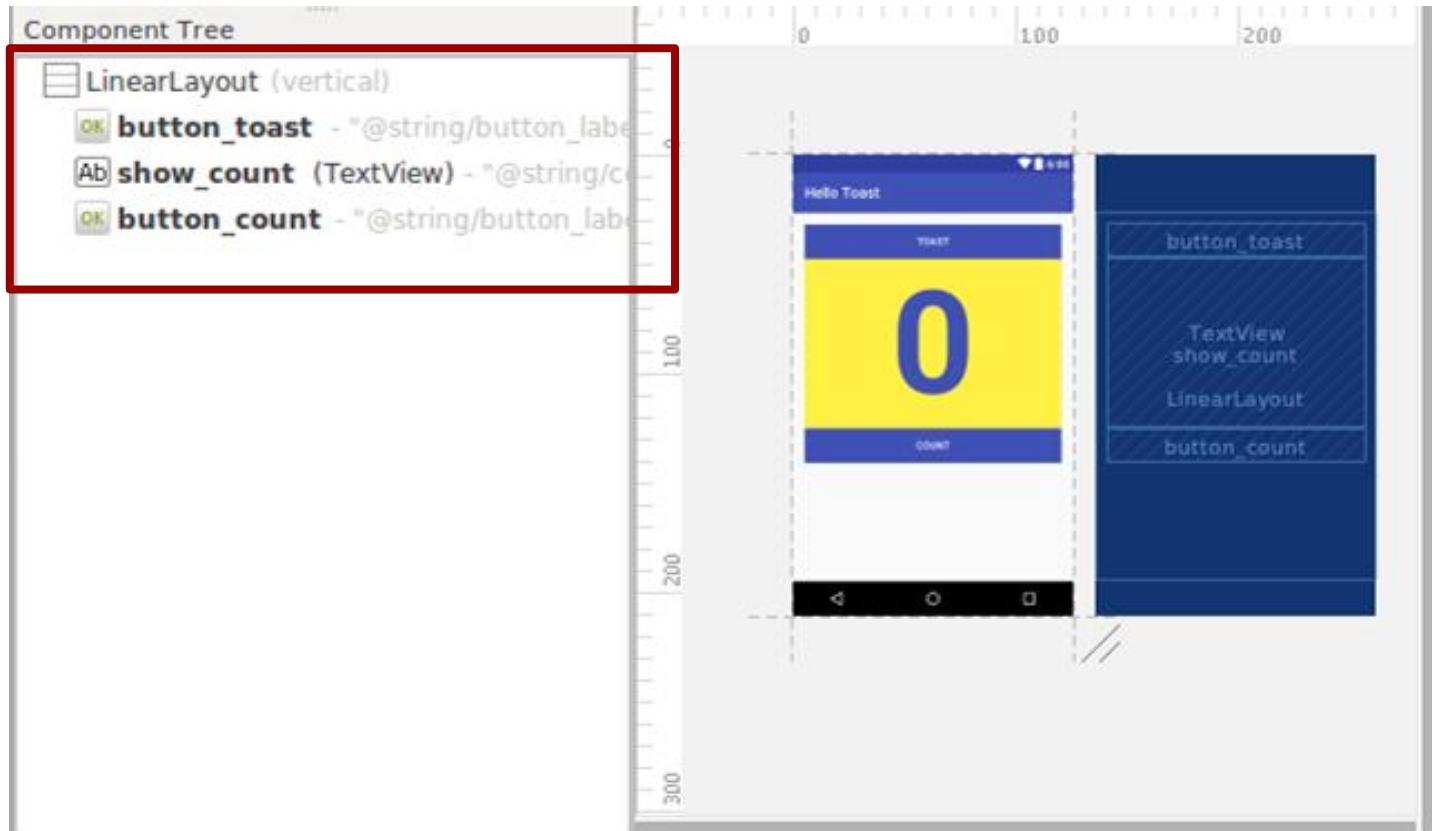
View hierarchy and screen layout



View hierarchy and screen layout



View hierarchy in the layout editor





Layout created in XML

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



Best practices for view hierarchies

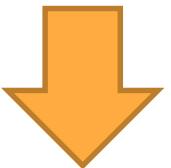
Arrangement of view hierarchy affects app performance

- Use **smallest number of simplest views** possible
- Keep the **hierarchy flat—limit nesting** of views and view groups

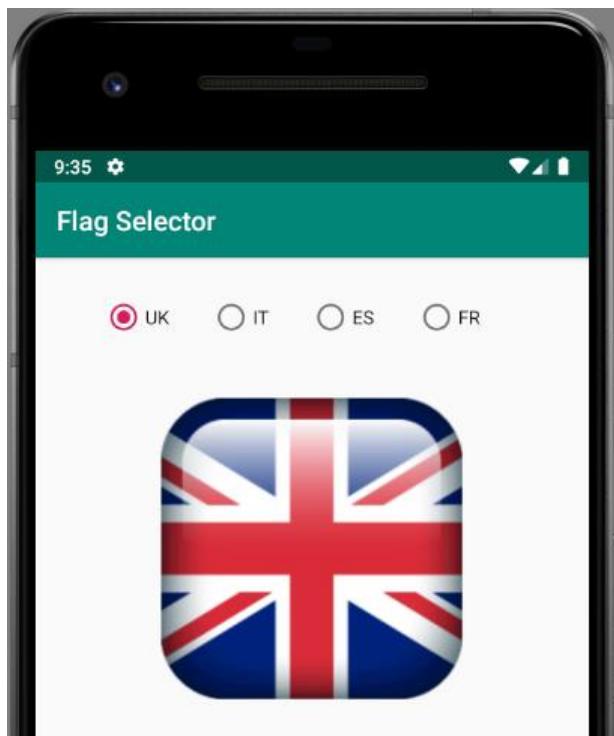
Exercise: Country info app

Behavior Specification (part 1)

each time the user taps on one of the four choices (UK, IT, ES, FR)



the corresponding flag of the country must be displayed



Exercise: Country info app

UI Components

Radio Buttons: four choices



Radio Group: used to create a multiple-exclusion scope for a set of radio buttons.

Checking one radio button that belongs to a radio group **unchecks any previously checked radio** button within the same group.

Note: The RadioGroup is a **subclass of LinearLayout** that has a vertical orientation by default (and we have an horizontal orientation of the Radio Buttons in the app...)



Exercise: Country info app

UI Components

ImageView

Displays image resources, for example
Bitmap or Drawable resources

```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/my_image"  
    android:contentDescription="@string/my_image_description"  
/>
```



Exercise: Country info app

Implementation Hints

Create a method:

```
public void clickFlag(View view) {...}
```

that recognizes which option button has been selected and changes the image displayed in the image view

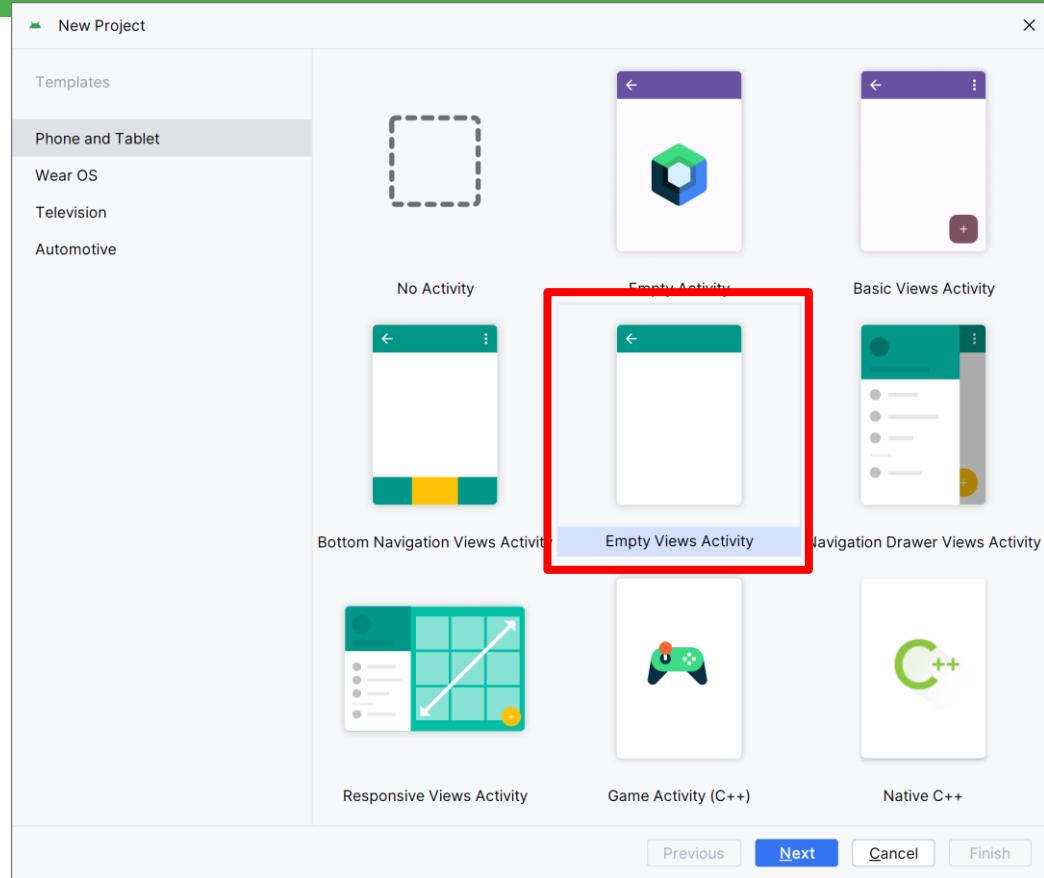
Put the images (e.g., 4 .PNG) in the folder «drawable»:

and access to them using R.drawable.<filename>



Pick activity template

Pick **Empty Views Activity** for simple and custom activities



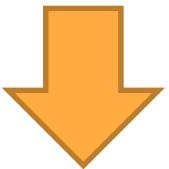
<https://stackoverflow.com/questions/76018702/i-cant-create-a-new-project-with-java-language-anymore-on-android-studio-flamin>

android-studio-2023.2.1.22

Exercise: Country info app

Behavior Specification (part 2)

each time the user taps on one of the four choices (UK, IT, ES, FR)



Below the flag, show also a long description of the country (e.g., a text from wikipedia) in a scrollable TextView

Hint: use ScrollView+TextView





Upload the Assignment on Aulaweb

1. Export the two projects as a Zip file
 - a. (“Counter App” + “Country info app”)
2. Put them in a Zip file named: *Surname.zip*
3. Upload it on Aulaweb



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

Layouts and Event handling

Goals & Contents

Goals

- Clarify some of the notions seen in practice in the previous assignments
- Then, implement a “realistic app”

Contents

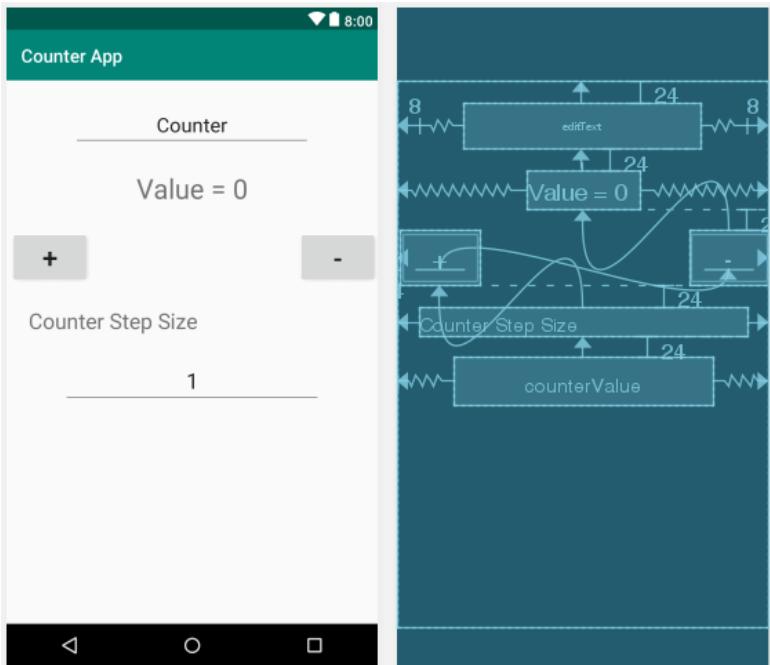
- The layout editor and ConstraintLayout
- Event handling

These slides are partially based on the material that Google provides for the course
Android Developer Fundamentals

<https://developer.android.com/courses/fundamentals-training/overview-v2>

Apps Development Summary

Design the UI



Implement the Behaviour

```
package com.example.helloeword01;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

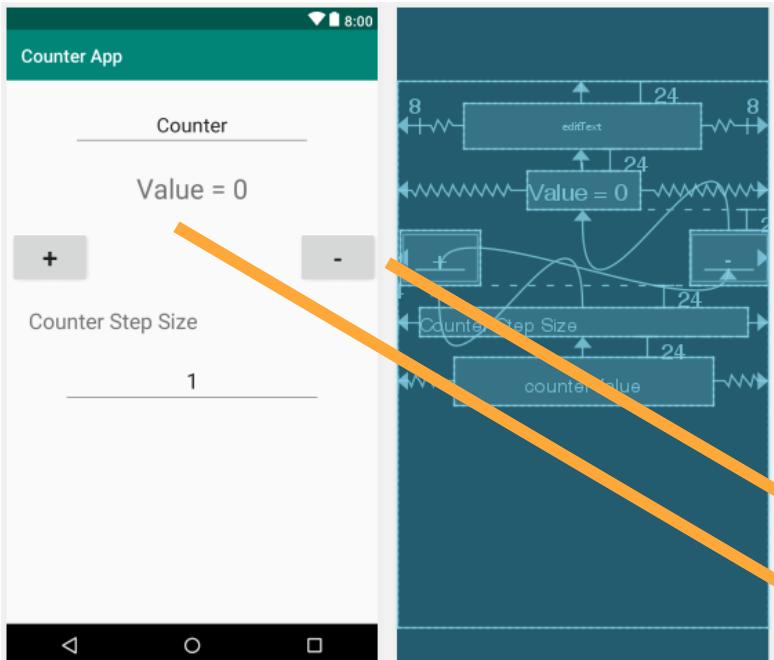
public class MainActivity extends AppCompatActivity {

    private static final String TAG = MainActivity.class.getSimpleName();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Apps Development Summary

Design the UI



Implement the Behaviour

```
package com.example.helloeword01;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    private static final String TAG = MainActivity.class.getSimpleName();

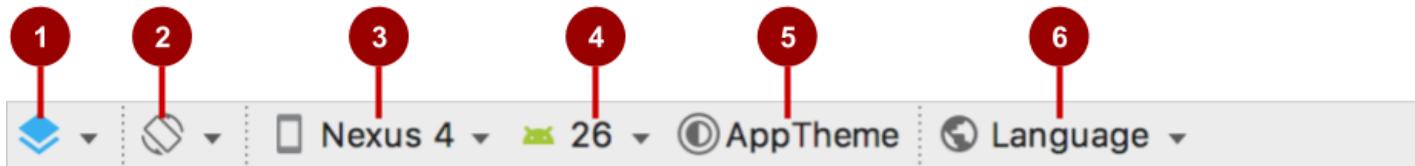
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

Associate event handlers to UI Widgets
e.g., when the – button is pressed do
or **update** the info the Widgets show
e.g., the counter value

The layout editor and Constraint Layout

- We have **already used these tools in practice**
- In the following we will see a more **complete overview** of the **available features**

Layout editor main toolbar



The figure above shows the **top toolbar** of the **layout editor**:

- 1. Select Design Surface:** Select **Design** to display a color preview of the UI elements in your layout, or **Blueprint** to show only outlines of the elements. To see *both* panes side by side, select **Design + Blueprint**.
- 2. Orientation in Editor:** Select **Portrait** or **Landscape** to show the preview in a vertical or horizontal orientation. The orientation setting lets you preview the layout orientations without running the app on an emulator or device.
- 3. Device in Editor:** Select the device type (phone/tablet, Android TV, or Android Wear).

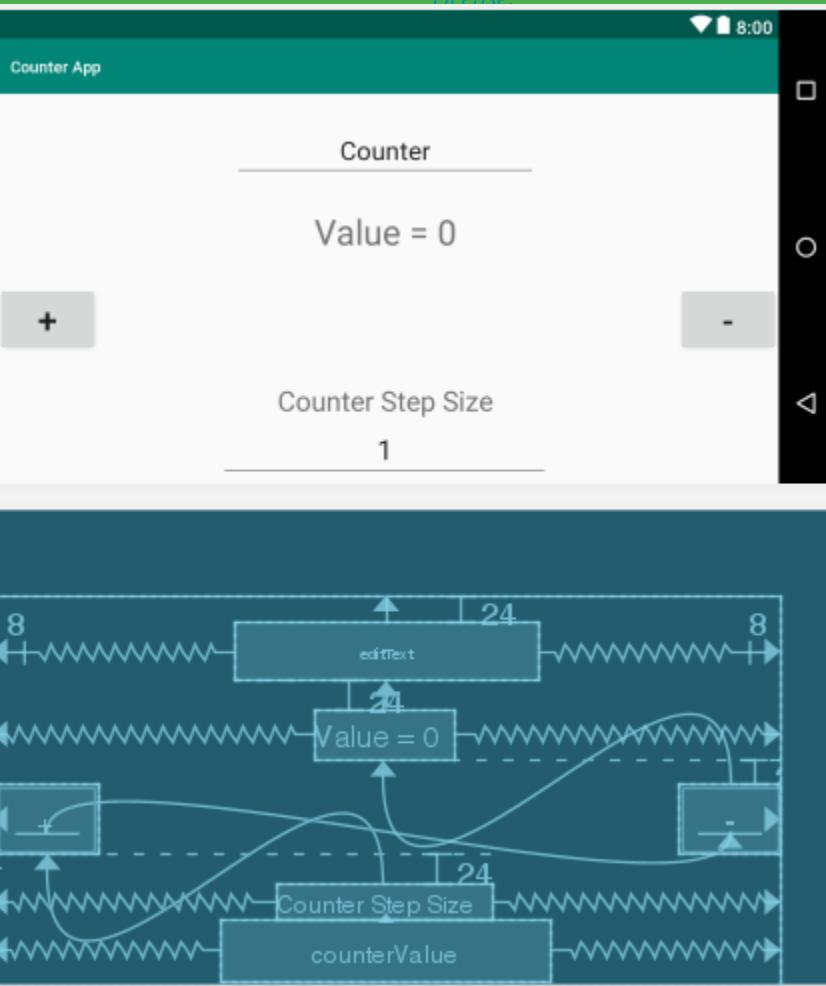
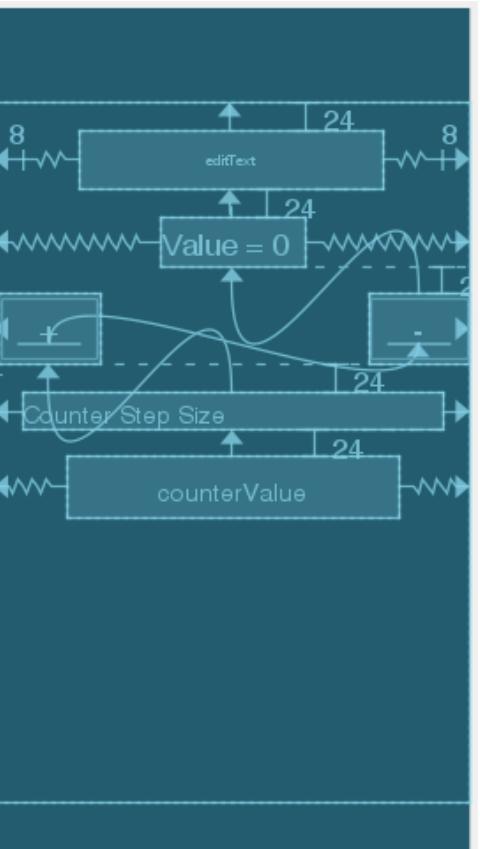


Preview layouts

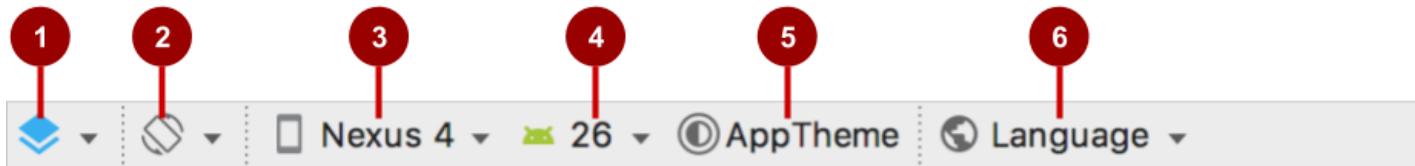
It is possible to ***preview an app's layout*** with a horizontal orientation, **without having to run the app** on an emulator or device.

1. Click Orientation in Editor button A small gray rectangular button containing a white icon of a smartphone with a circular rotation arrow around it, with a downward-pointing arrow to its right.
2. Choose **Switch to Landscape** or **Switch to Portrait**

Preview layouts



Layout editor main toolbar



The figure above shows the **top toolbar** of the **layout editor**:

1. **Select Design Surface:** Select **Design** to display a color preview of the UI elements in your layout, or **Blueprint** to show only outlines of the elements. To see *both* panes side by side, select **Design + Blueprint**.
2. **Orientation in Editor:** Select **Portrait** or **Landscape** to show the preview in a vertical or horizontal orientation. The orientation setting lets you preview the layout orientations without running the app on an emulator or device.
3. **Device in Editor:** Select the device type (phone/tablet, Android TV, or Android Wear).



Preview layouts

Preview layout with **different devices**:

1. Click Device in Editor button 
2. Choose device

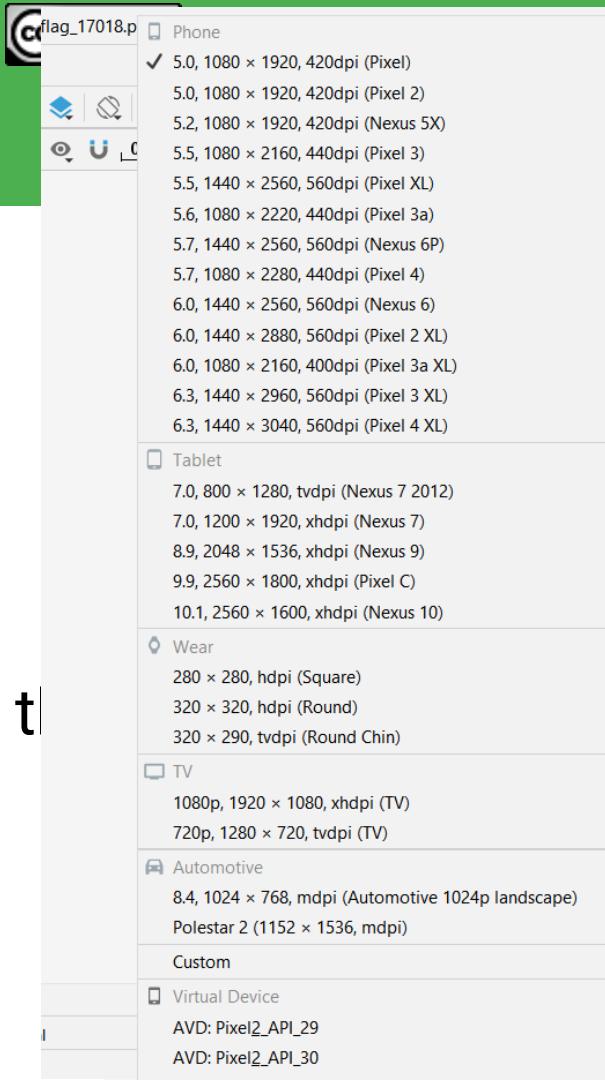
For example, select **Nexus 4**, **Nexus 5**, and then **Pixel** to see differences in the previews.

Preview layouts

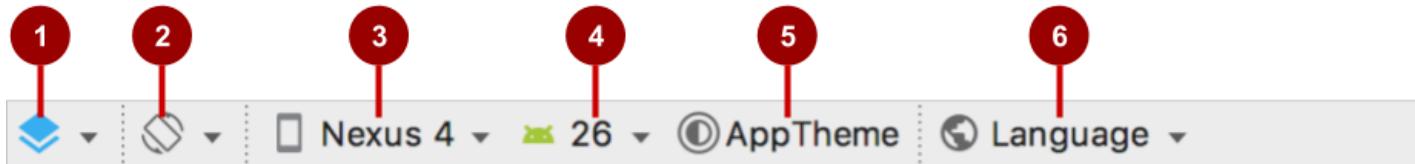
Preview layout with different devices:

1. Click Device in Editor button 
2. Choose device

For example, select **Nexus 4**, **Nexus 5**, and the **Nexus 6** to see the differences in the previews.



Layout editor main toolbar

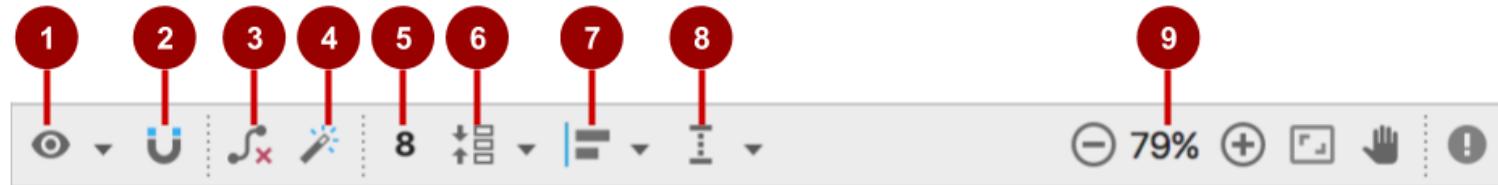


The figure above shows the top toolbar of the layout editor:

4. **API Version in Editor:** Select the version of Android to use to show the preview
5. **Theme in Editor:** Select a theme (such as **AppTheme**) to apply to the preview
6. **Locale in Editor:** Select the language and locale for the preview. This list displays only the languages available in the string resources (see the documentation on localization for details on how to add languages).

You can also select **Preview as Right To Left** to see the layout as if an RTL language had been chosen

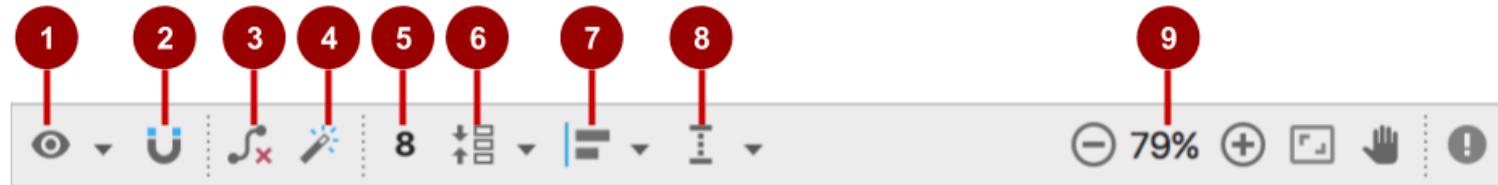
ConstraintLayout toolbar in layout editor



The figure above shows the ConstraintLayout editing toolbar:

- 1. Show:** Select **Show Constraints** and **Show Margins** to show them in the preview, or to stop showing them.
- 2. Autoconnect:** Enable or disable Autoconnect. With Autoconnect enabled, you can drag any element (such as a Button) to any part of a layout to generate constraints against the parent layout.
- 3. Pack: Clear All Constraints:** Clear all constraints in the entire layout.

ConstraintLayout toolbar in layout editor



The figure above shows the ConstraintLayout editing toolbar:

4.Infer Constraints: Create constraints by inference.

5.Default Margins: Set the default margins.

6.Pack: Pack or expand the selected elements.

7.Align: Align the selected elements.

8.Guidelines: Add vertical or horizontal guidelines.

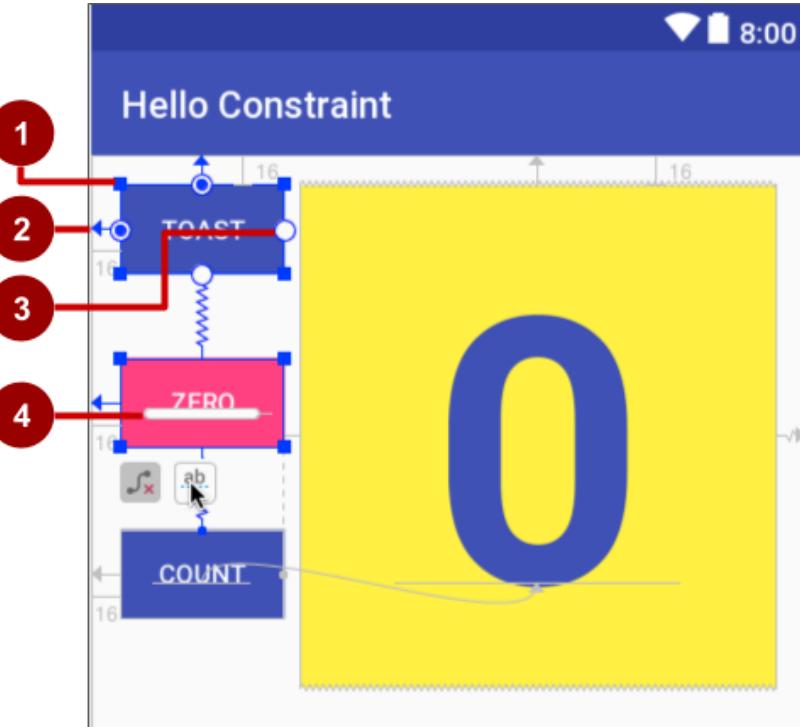
9.Zoom controls: Zoom in or out.

ConstraintLayout handles

A **constraint** is a connection or alignment to

- another UI element
- the parent layout
- an invisible guideline

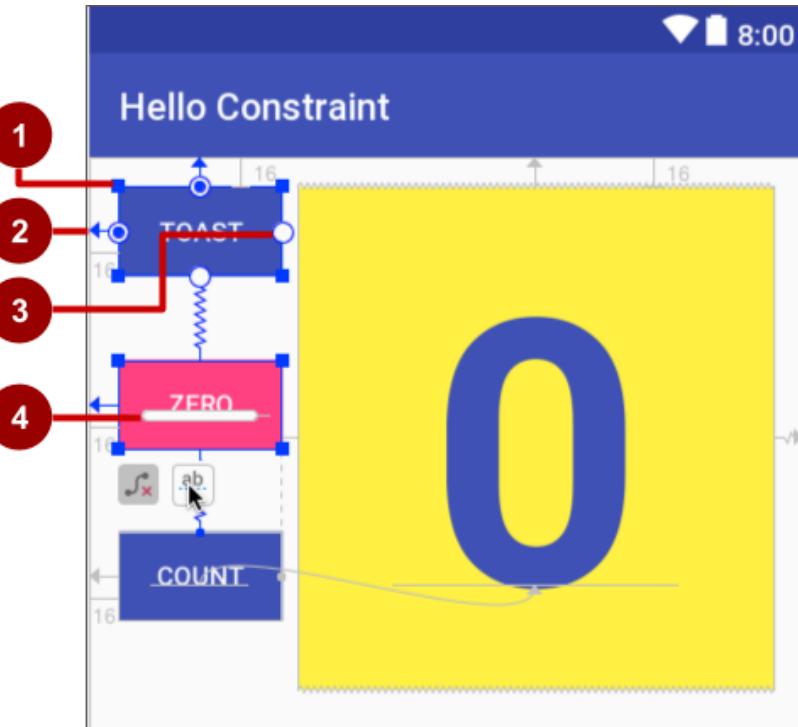
Each **constraint** appears as a line extending from a **circular handle**



ConstraintLayout handles

The figure shows the constraint and resizing handles on View elements in a layout:

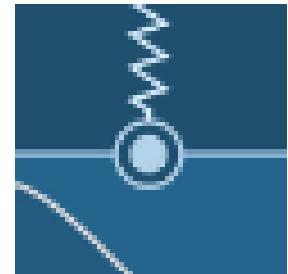
- 1. Resizing square handle.**
- 2. Constraint line and handle.** In the figure, the constraint aligns the left side of the **Toast** Button to the left side of the layout.
- 3. Constraint handle without a constraint line.**
- 4. Baseline handle.** The baseline handle aligns the text baseline of an element to the text baseline of another element.



Constraint handle

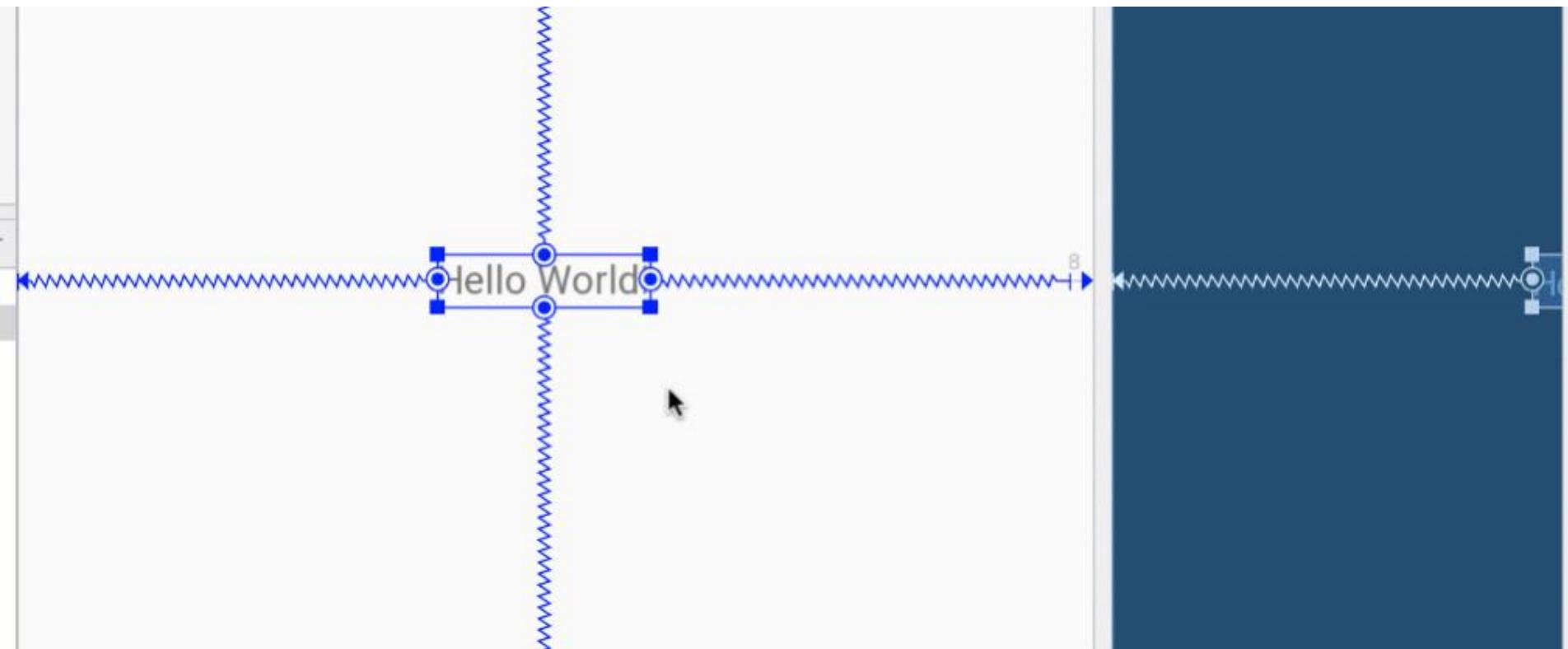
To **create a constraint**:

- **click a constraint handle**
 - shown as a circle on each side of an element
- **drag the circle** to another constraint handle or to a parent boundary



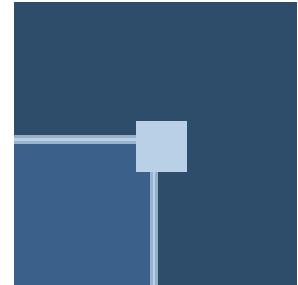
A zigzag line represents the constraint

Constraint handle



Resizing handle

- Drag the **square** resizing handles **to resize the element**
- **Warning!** doing so **hard-codes** the **width** and **height** dimensions
- better to avoid for most elements because **hard-coded dimensions** **don't adapt** to **different screen densities**



Align elements by baseline

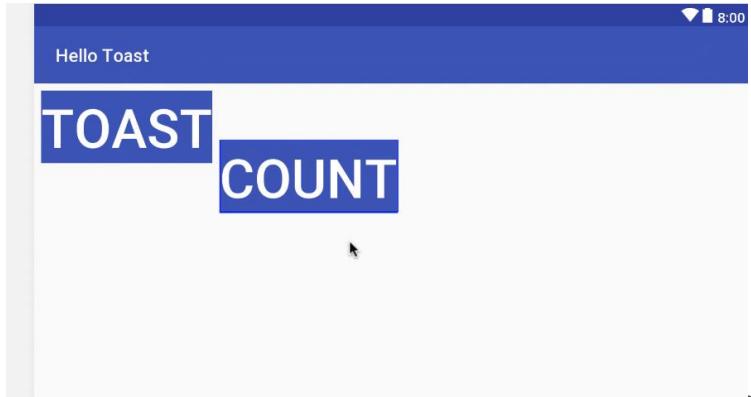
Align **one UI element that contains text**

such as a TextView or Button

with **another UI element that contains text**

A baseline constraint lets you constrain the elements so that **the text baselines match**.

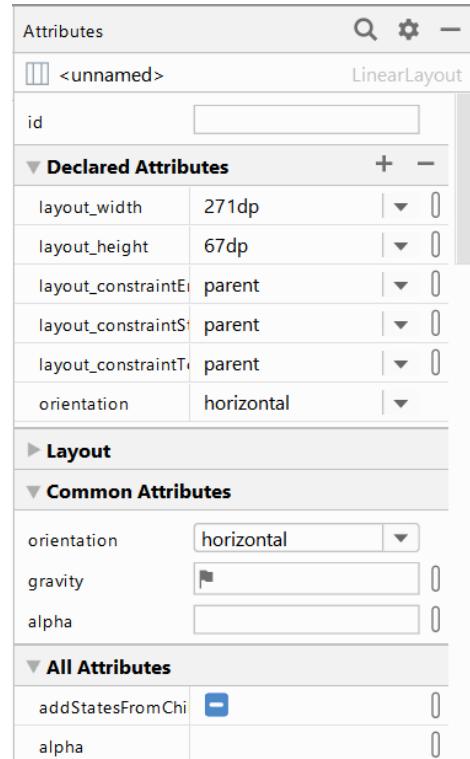
1. Click the  baseline constraint button
2. Drag from baseline to other element's baseline



Attributes pane

The **Attributes** pane *offers access to all of the XML attributes* you can assign to a UI element.

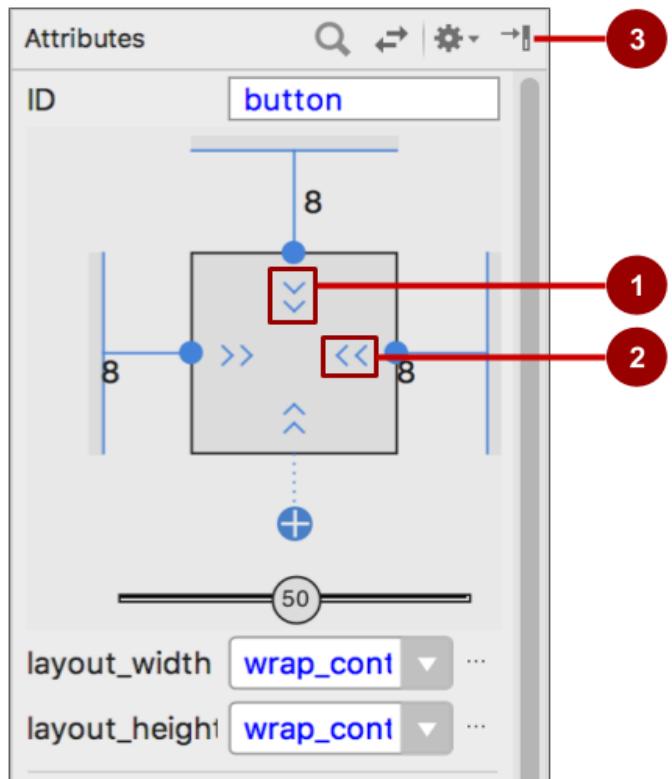
- Click the Attributes tab
- Attributes pane includes:
 - Margin controls for positioning
 - Attributes such as `layout_width`



Attributes pane view inspector

The **Attributes** pane includes a square sizing panel called the *view inspector*.

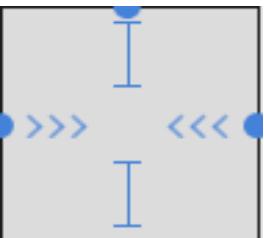
1. Vertical view size control specifies `layout_height`
2. Horizontal view size control specifies `layout_width`
3. Attributes pane close button



Layout_width and layout_height

layout_width and layout_height change with size controls

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



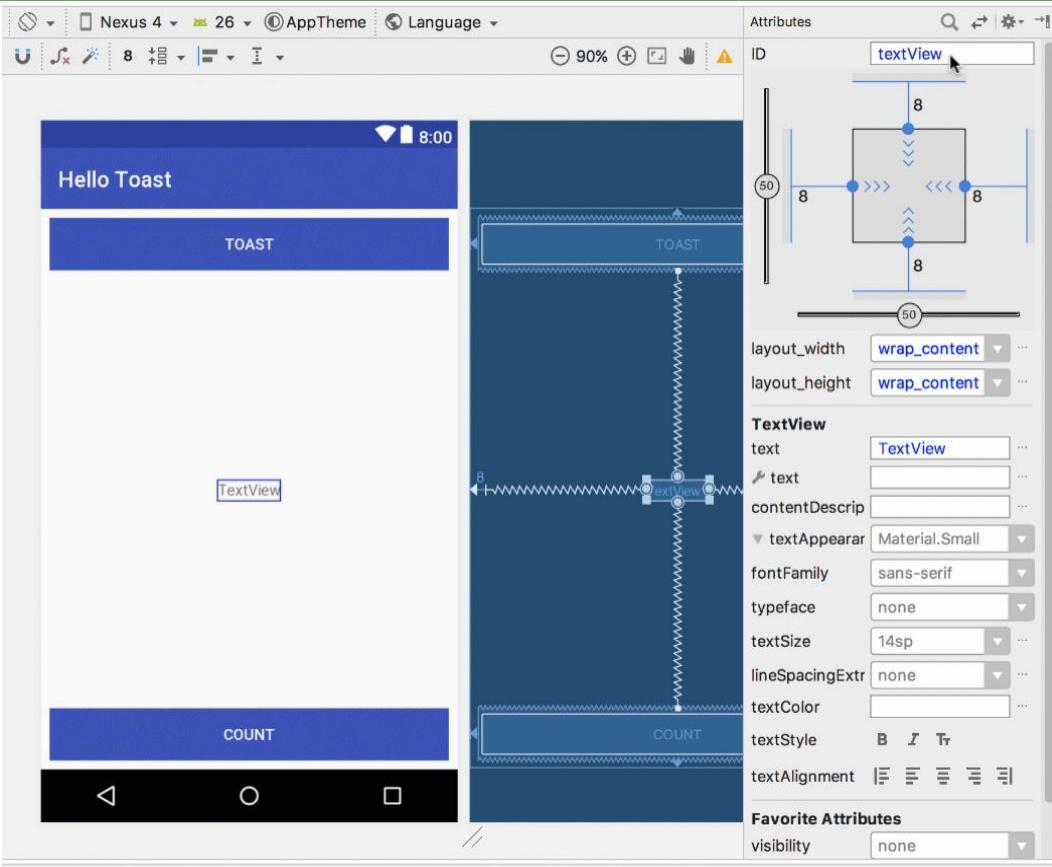


Set attributes

To view and edit all attributes for element:

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

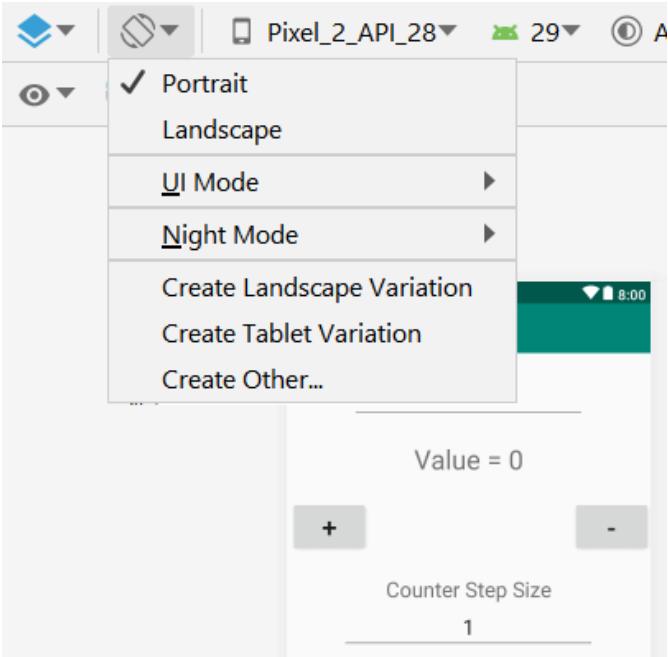
Set attributes example: TextView



Create layout variant for landscape

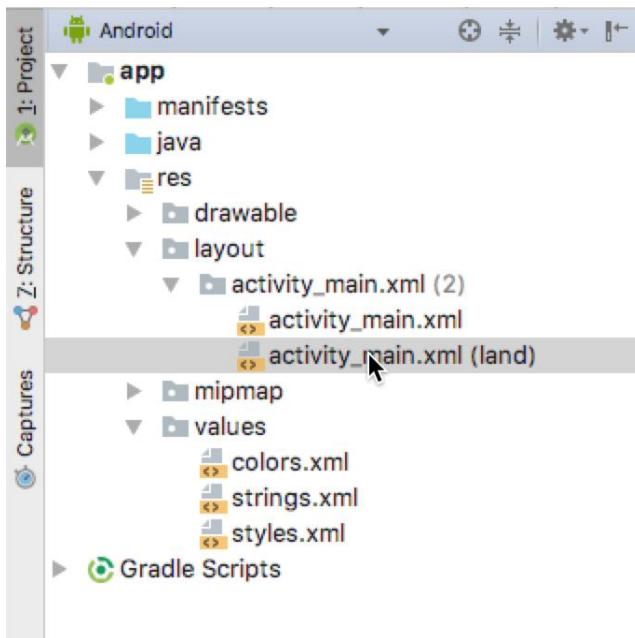
To **create a variant** of the layout ***strictly for the horizontal orientation***, leaving the vertical orientation layout alone:

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



Create layout variant for landscape

- You can **change this layout**, which is specifically **for horizontal orientation**, without changing the original **portrait (vertical) orientation**.
- Android Studio automatically creates the variant for you, called `activity_main.xml (land)`



Create layout variant for tablet

To create a layout variant for tablet-sized screens, click the **Orientation in Editor** button and select **Create Tablet Variation**

1. Click Orientation in Layout Editor 
2. Choose **Create Tablet Variation**
3. Layout variant created: `activity_main.xml (600dp)`
4. Edit the layout variant as needed

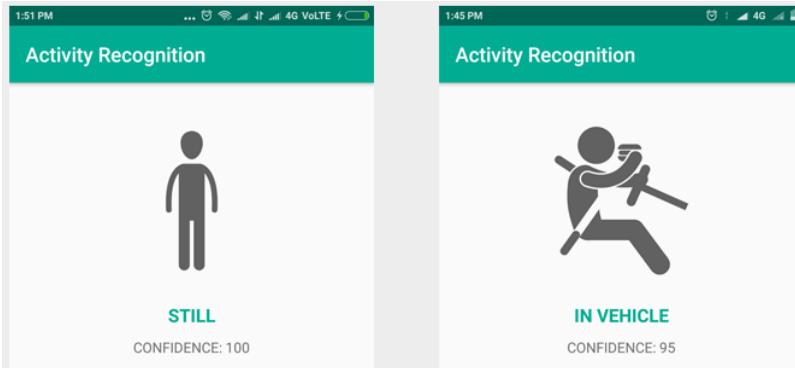
Event Handling

Events

Something that happens

- In UI: Click, tap, drag
- Device: DetectedActivity such as walking, driving, tilting

The detected activity of the device with an associated confidence





Event Handlers

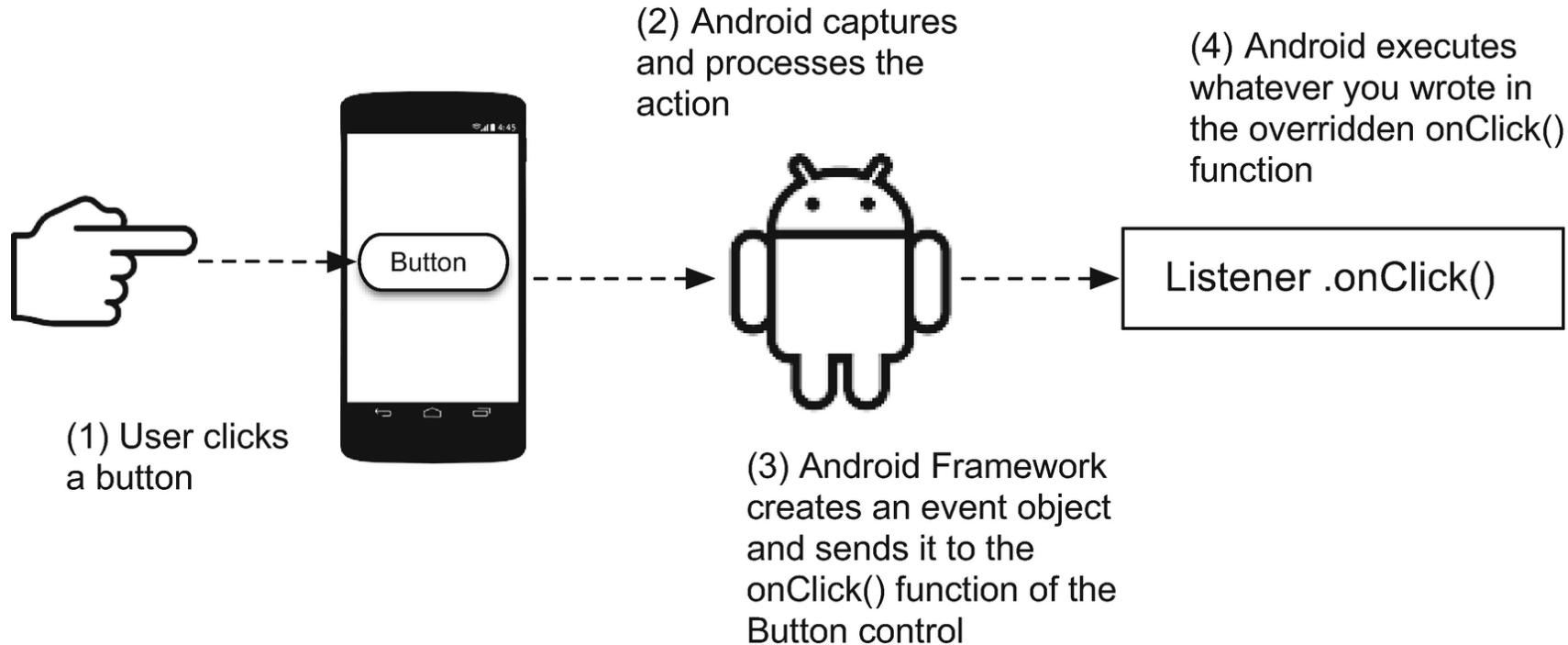
Methods that do something in response to an event (e.g., click or tap)

- an **event handler** is a method **triggered** by a specific event and that **does something in response** to such event
- an **event listener** is an interface in the View class that contains a single callback method. This method will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI

Hints for our first exercises:

- In order to work with the android:onClick attribute the method **must be public** and **return void**
- To know **which View called the method** it requires a **view parameter**

Event Handlers





Attach in XML and implement in Java

Attach handler to view in XML layout:

android:onClick="showToast"

Implement handler in Java activity:

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



Alternative: Set click handler in Java

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

Additional details here:

<https://developer.android.com/guide/topics/ui/ui-events>

https://www.tutorialspoint.com/android/android_event_handling.htm



Updating a View

To update a View the code must **first instantiate an object from the View**. The code can then update the object, which updates the screen.

To refer to the View in the code, **use the `findViewById()` method of the View class**, which looks for a View based on the resource id.

For example, the following statement sets `mShowCount` to be the `TextView` in the layout with the resource id `show_count`:

```
mShowCount = (TextView) findViewById(R.id.show_count);
```



Updating a View

From this point on, the code can use `mShowCount` to represent the `TextView`, so that when you update `mShowCount`, the `TextView` is updated.

For example, when the following Button with the `android:onClick` attribute is tapped, `onClick` calls the `countUp()` method:

```
android:onClick="countUp"
```



Updating a View

You can implement `countUp()` to increment the count, convert the count to a string, and set the string as the text for the `mShowCount` object:

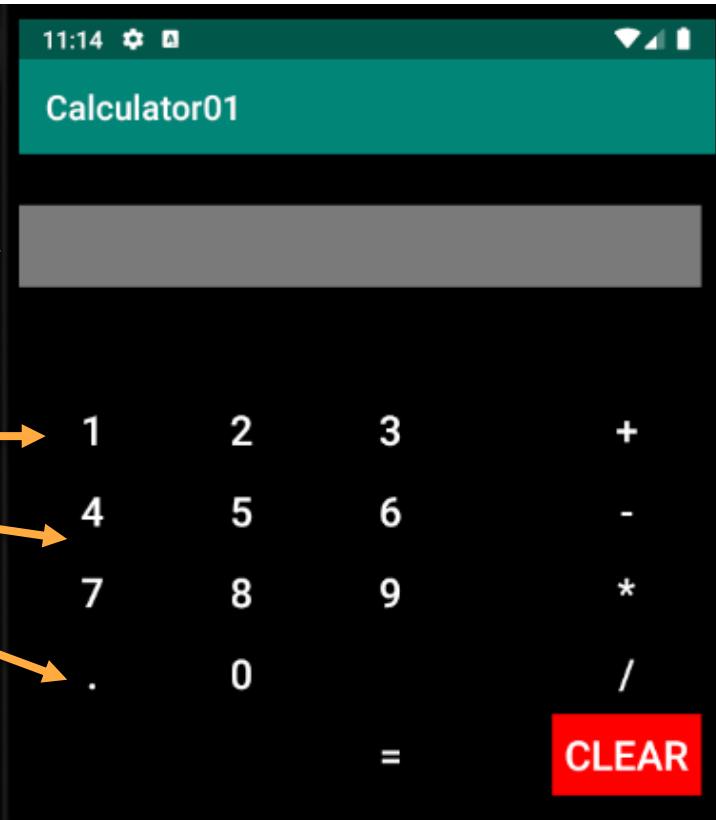
```
public void countUp(View view) {  
    mCount++;  
    if (mShowCount != null)  
        mShowCount.setText(Integer.toString(mCount));  
}
```

Since you had already associated `mShowCount` with the `TextView` for displaying the count, the `mShowCount.setText()` method updates the `TextView` on the screen.

Calculator

The first realistic App!!

- 1 TextView
- Several Buttons





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

Calculator

More in details

EditText: To display numbers and the result

Button: Numbers 0-9

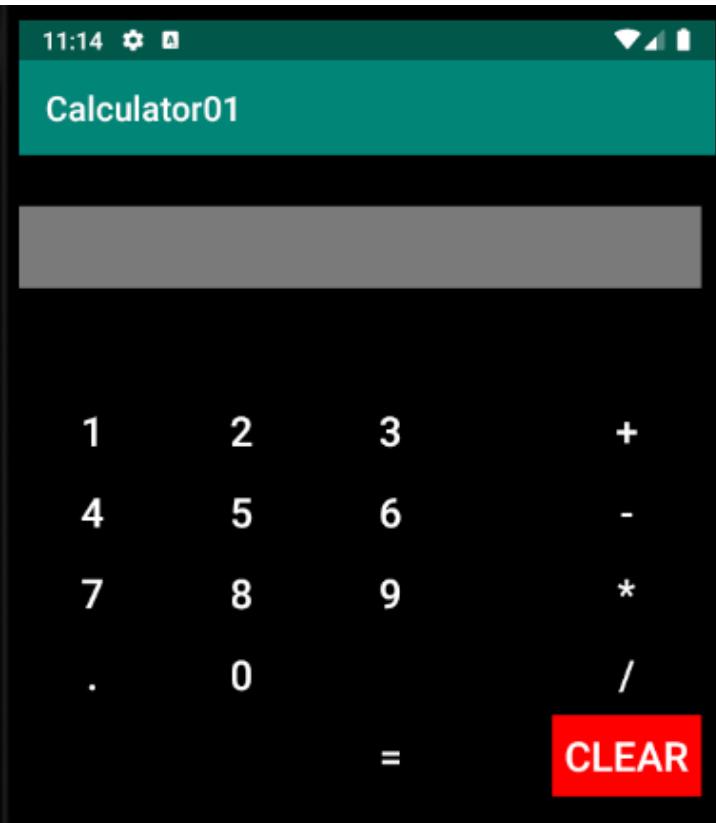
Button: Operation buttons

(+, -, *, /)

Button: Decimal button

Button: Clear display button

Button: Compute result button (=)





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

Calculator

Steps

- Create the Activity (GUI)
- Think to a possible solution
- Implement the Behavior

Simplifications:

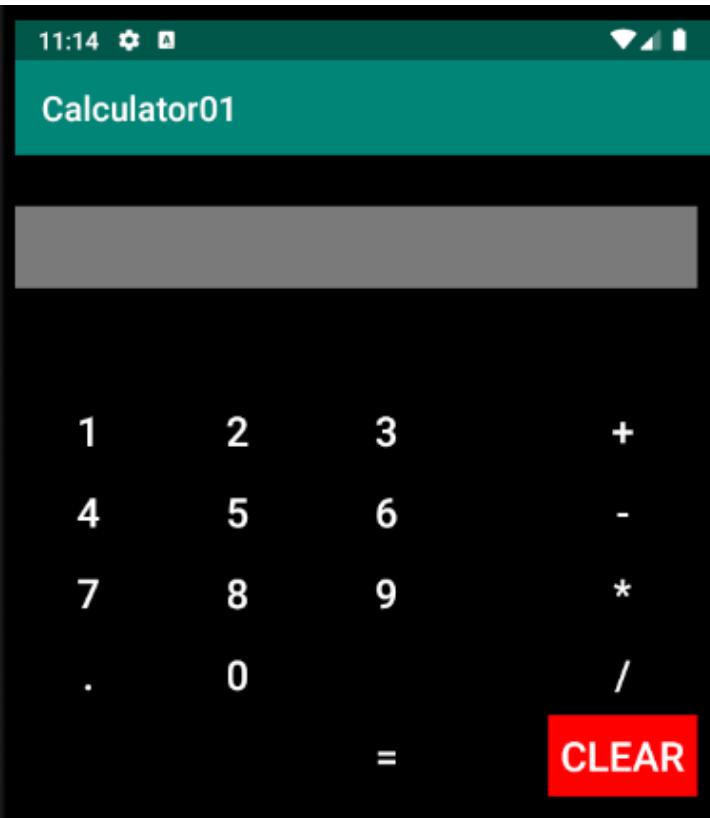
the calculator takes the first number (that must be positive), the operator, the second number and then computes the result

$$N1 \text{ op } N2 = \text{Res}$$

In case more numbers are inserted, only the last two are considered

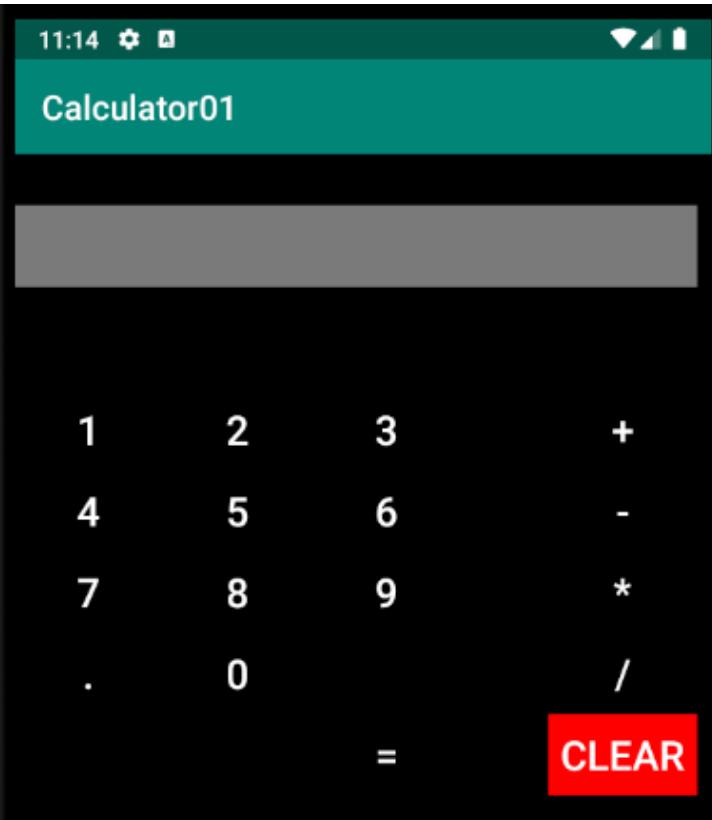
Not required to manage something like

$$X + Y - Z / K = ???$$



Calculator

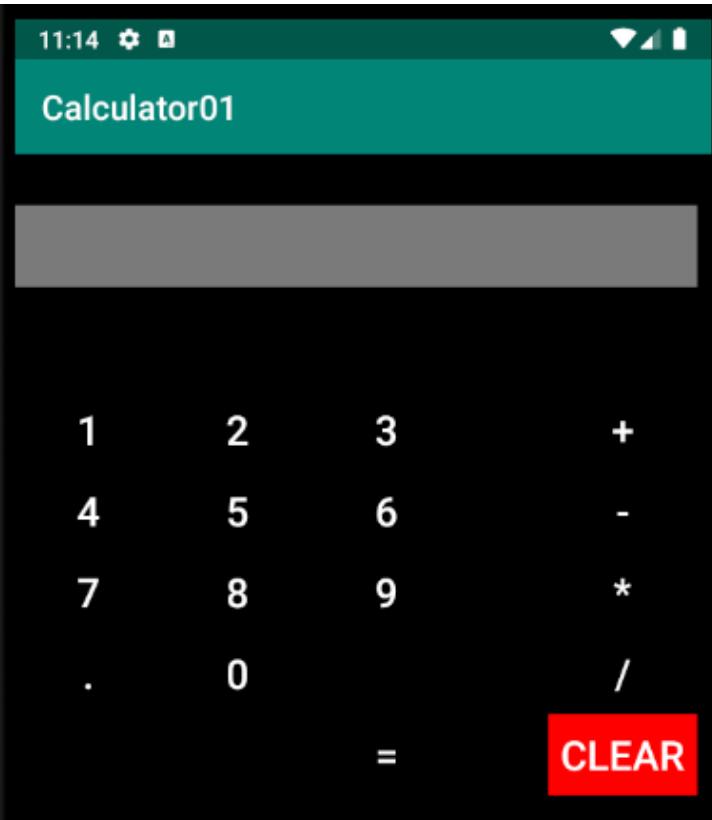
- Number Buttons add a digit to the TextView
- Decimal Point button should be inserted only once in a number
- Operation Buttons should
 - save the first operand in a register
 - save the required operation in another register





Calculator

- Compute Result Button should
 - read the first operand from the register (N1)
 - read the operand selected before (op)
 - read the current value in the Text View (N2)
 - compute $N1 \text{ op } N2 = \text{Res}$
 - update the Text View with Res

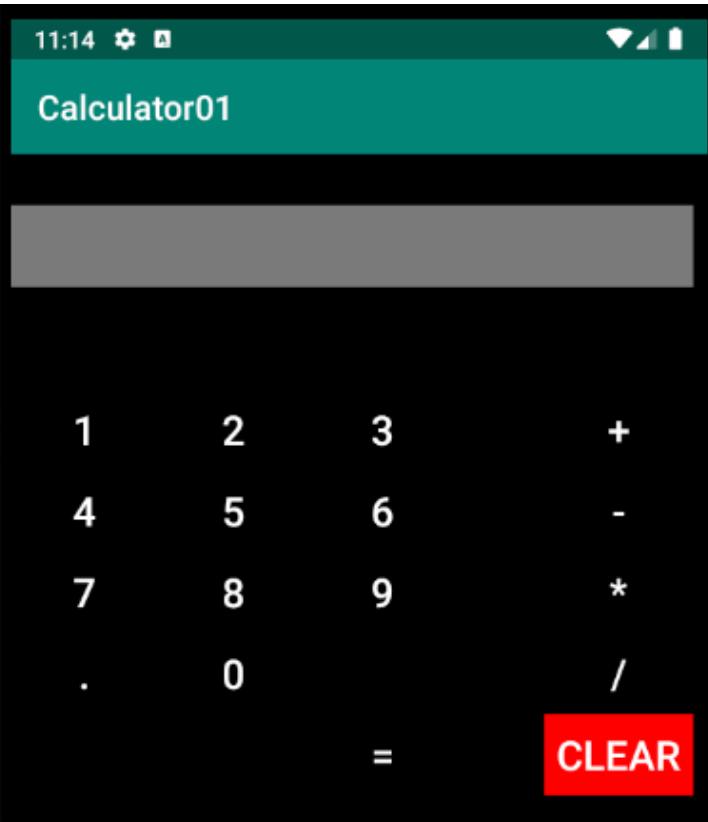




This work is licensed under a [Creative Commons Attribution 4.0 International License](#).

Calculator

- Try to make the app suitable for both Portrait and Lanscape visualizations
 - NOT change visualization during computations... We will see how to manage this in next lessons
- Try to use a colour scheme different from the default one: e.g., similar to the one show in the image
- Manage possible errors like:
 - inserting multiple decimal points
 - leave empty the first number (Null op N2 = ??)
 - avoid to insert non significant zero (e.g., 0012+014)
 - in general clicking randomly on buttons should not raise any error....





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

Activities and Intents

Contents

- Activities
- Defining an Activity
- Starting a new Activity with an Intent
- Passing data between activities with extras
- Navigating between activities

These slides are partially based on the material that Google provides for the course
Android Developer Fundamentals

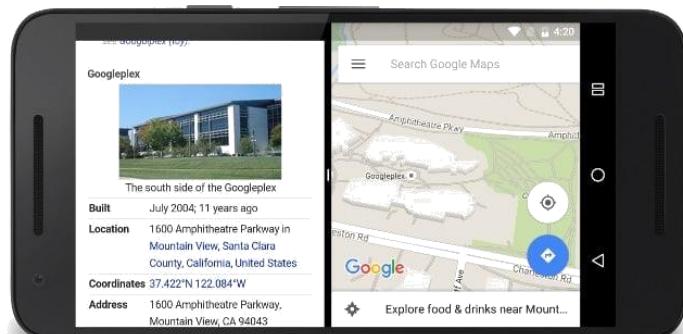
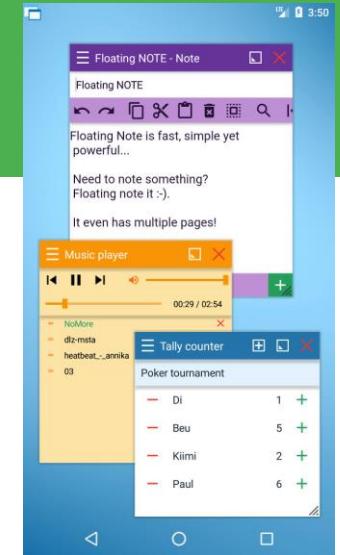
Activities (high-level view)

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
- A Java class (typically one Activity in one file)

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
- A Java class (typically one Activity in one file)



What is an Activity?

- An *activity* **represents a single screen** in the app with an interface the user can interact with
 - e.g. **an email app** might have **3 activities**
 - one to shows a list of received emails
 - one to compose an email
 - one to read individual messages

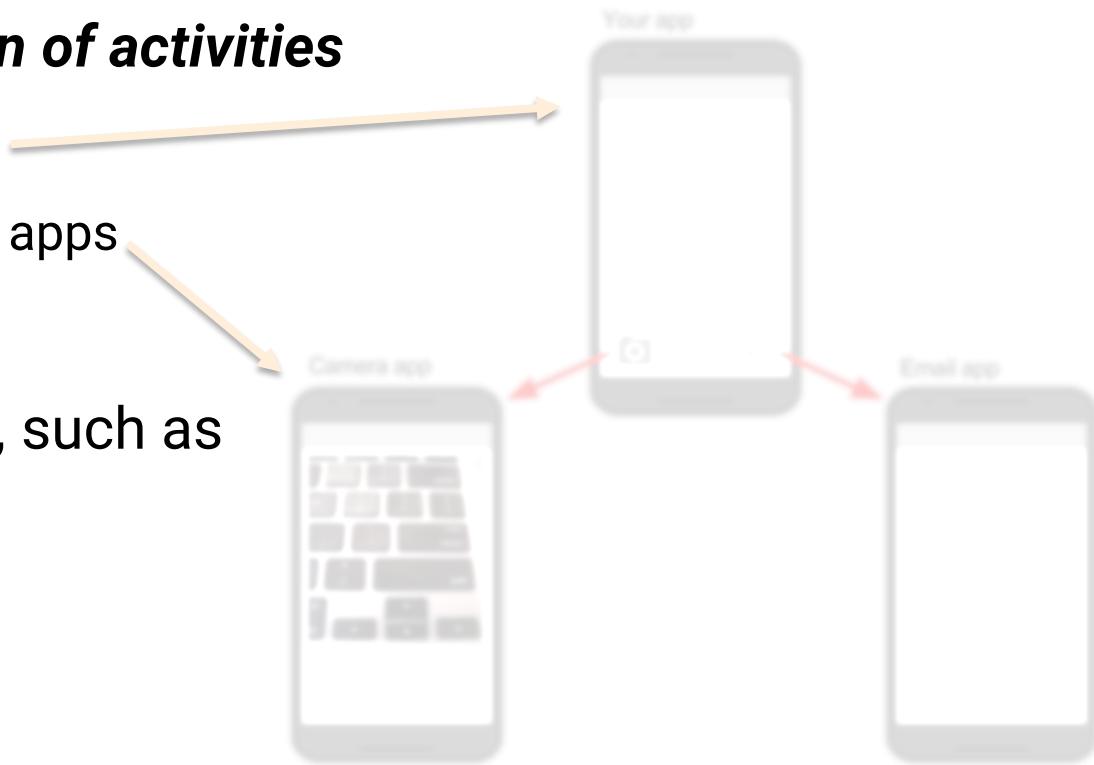
What does an Activity do?

- Apps are often a ***collection of activities***

- that you create yourself
- that you reuse from other apps

- Handles user interactions, such as

- button clicks
- text entry
- login verification

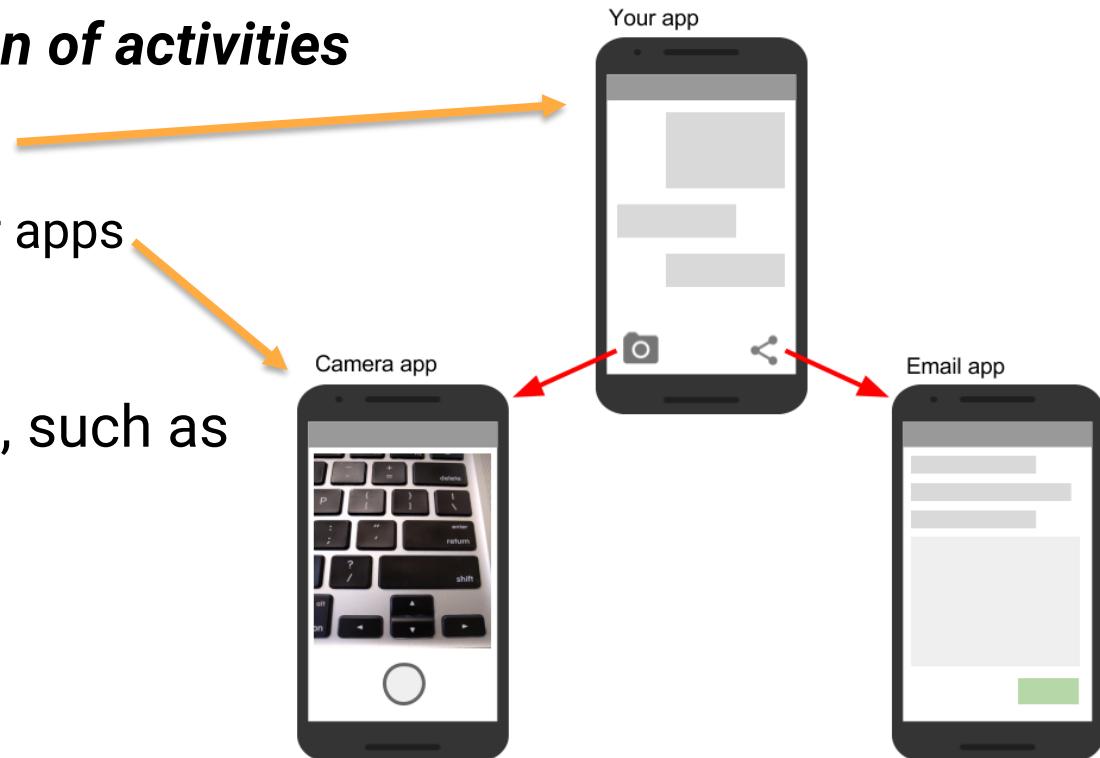


What does an Activity do?

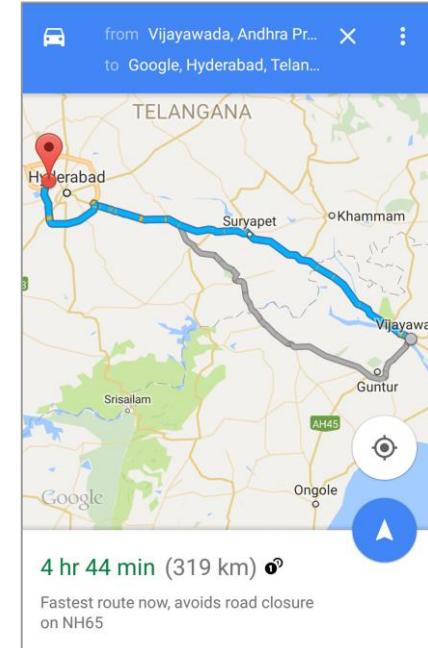
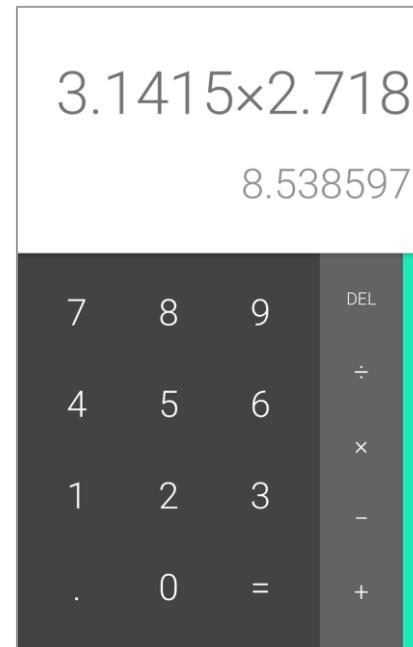
- Apps are often a ***collection of activities***

- that you create yourself
- that you reuse from other apps

- Handles user interactions, such as
 - button clicks
 - text entry
 - login verification

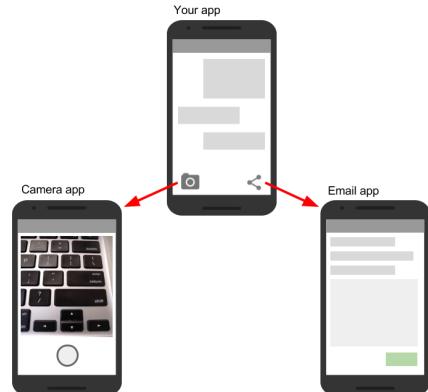


Examples of activities



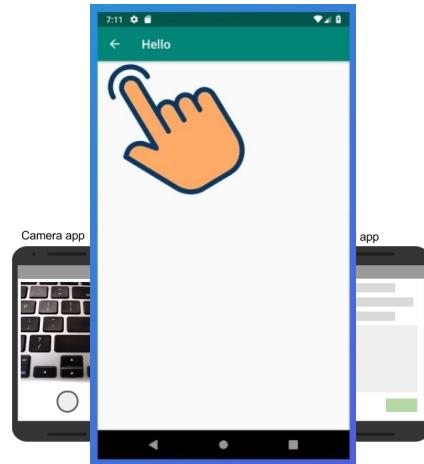
Apps and activities

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



Apps and activities

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

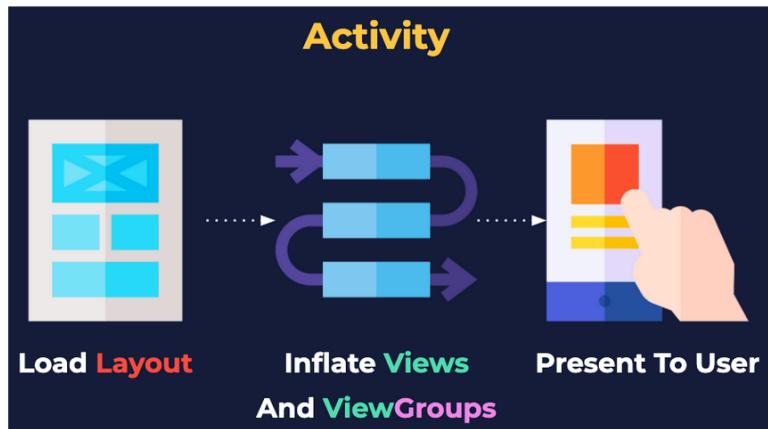


What does an Activity do?

- Has a life cycle: an activity is
 - Created
 - Started
 - Runs
 - Paused
 - Resumed
 - Stopped
 - Destroyed

Layouts and Activities

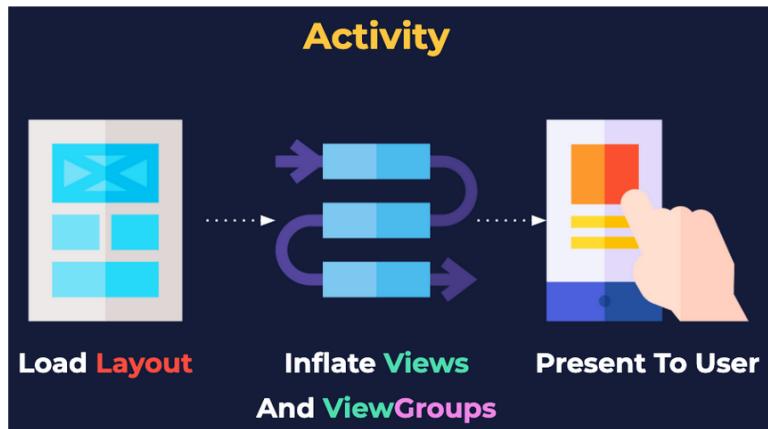
- An Activity typically has a UI layout
- Layout is usually defined in one or more XML files
- Activity "inflates" layout as part of being created



```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);
```

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



```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);
```

Implementing Activities

Implement new activities

When *creating a new project*

or add a new Activity to an app by choosing File > New > Activity

the wizard automatically **performs** the following steps:

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

When *creating a new project* the **MainActivity** is, by default, a **subclass of the AppCompatActivity class**



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

This **allows to use up-to-date Android app** features such as the app bar and Material Design while still **enabling the app to be compatible** with devices running **older versions of Android**.

2. Define Activity Java class

The **first task** in the implementation of an Activity subclass **is to implement the standard Activity lifecycle callback methods** (such as `onCreate()`) to handle the state changes for your Activity.

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

These state changes include things such as when the Activity is **created**, **stopped**, **resumed**, or **destroyed**.

2. Define Activity Java class

The **one required callback** that an app must implement is the **onCreate()** method.

The **system calls this method when it creates the Activity**, and all the essential components of your Activity should be initialized here.

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

3. Connect activity with layout

The `onCreate()` method calls **`setContentView()`** with the path to a layout file.

The **system creates all the initial views** from the specified layout and adds them to your Activity. This is often referred to as *inflating the 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

Each **Activity** in an app **must be declared** in the **AndroidManifest.xml** file with the `<activity>` element, inside the `<application>` section.

The only required attribute is `android:name`, which specifies the class name for the Activity (such as `MainActivity`)



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

4. Declare activity in Android manifest

Each **Activity** in an app **must be declared** in the **AndroidManifest.xml** file with the `<activity>` element, inside the `<application>` section.

The only required attribute is `android:name`, which specifies the class name for the Activity (such as `MainActivity`)



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

4. Declare main activity in manifest

MainActivity needs to include **intent-filter** to start from launcher

The `<action>` element specifies that this is the "main" entry point to the app

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

only the MainActivity should include the "main" action

4. Declare main activity in manifest

MainActivity needs to include **intent-filter** to start from launcher

The `<action>` element specifies that this is the "main" entry point to the app

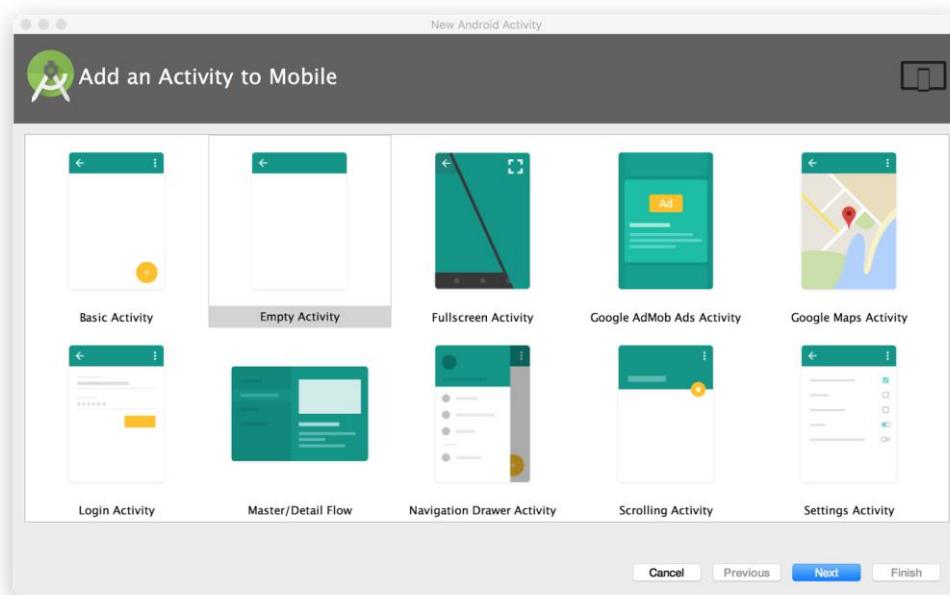
```
<activity android:name=".MainActivity" >  
  
    <intent-filter>  
        <action android:name="android.intent.action.MAIN" />  
        <category android:name="android.intent.category.LAUNCHER" />  
    </intent-filter>  
  
</activity>
```

only the MainActivity should include the "main" action

Add Activities to App

Each Activity of an app and its associated layout file is supplied by an Activity template in Android Studio such as Empty Activity or Basic Activity.

You can add a new Activity to your project by choosing
File > New > Activity

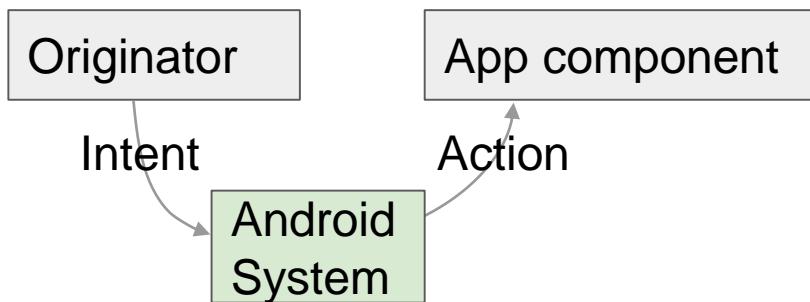


Intents

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.



Starting the main activity

1. When the app is **first started** from the **device home screen**
2. the **Android runtime sends an Intent to the app** to start the app's main activity
the one defined with the MAIN action and the LAUNCHER category in the AndroidManifest.xml file

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

What can intents do?

- Start an Activity
 - A button click starts a new Activity for text entry
 - pass data between one activity and another
 - Clicking Share opens an app that allows you to post a photo
- Start a Service
 - Initiate downloading a file in the background
- Deliver Broadcast
 - The system informs everybody that the phone is now charging

Explicit and implicit intents

Explicit Intent

- Starts a specific Activity
 - e.g. Request tea with milk delivered by a specific Cafe
 - Main activity starts the ViewShoppingCart Activity

Implicit Intent

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

Explicit and implicit intents

Explicit Intent

- Starts a specific Activity
 - e.g. Request tea with milk delivered by a specific Cafe
 - Main activity starts the ViewShoppingCart Activity

Implicit Intent

- Asks system to find an Activity that can handle this request
 - e.g. 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);`

Example:

```
Intent messageIntent = new Intent(this, ShowMessageActivity.class);
startActivity(messageIntent);
```

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);`

Example:

```
Intent messageIntent = new Intent(this, ShowMessageActivity.class);
startActivity(messageIntent);
```

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

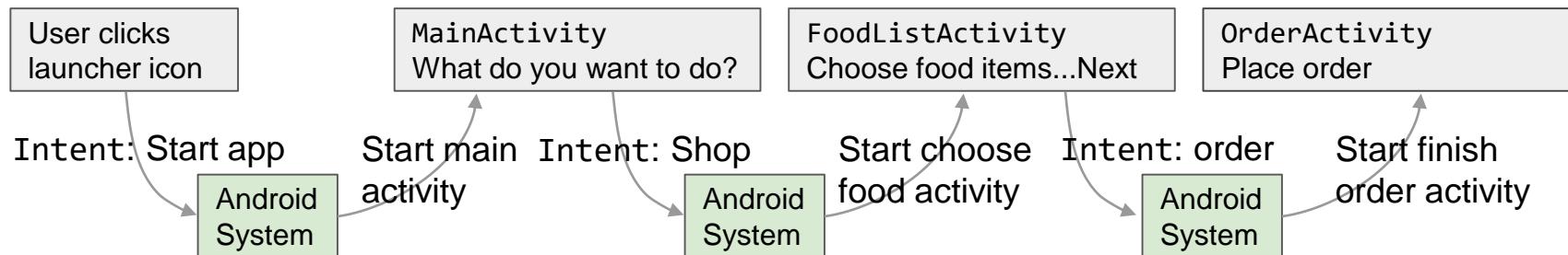
```
Uri uri = Uri.parse("http://www.google.com");  
Intent it = new Intent(Intent.ACTION_VIEW, uri);  
startActivity(it);
```

Dial a phone number

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

How Activities Run

- All Activity instances are managed by the Android runtime
- Started by an "Intent", a message to the Android runtime to run an activity



Sending and Receiving Data

Sending data with intents

In addition to **open a new activity**, **with an intent** it is possible **to pass data** from one Activity to another.

In particular, it is possible to use **Intent data** or **Intent extras**

- **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**

There are several key differences between data and extras that determine which you should use

Intent data

The **Intent data** can hold **only one piece** of information: a **URI** representing the location of the data you want to operate on.

That URI could be:

- a web page URL (`http://`),
- a telephone number (`tel://`),
- a geographic location (`geo://`) or
- any other custom URI you define.

A Uniform Resource Identifier (URI) is a compact string of characters for identifying an abstract or physical resource

<https://www.ietf.org/rfc/rfc2396.txt>

When use Intent data

Use the Intent data field when:

- you only have **one piece of information** that you need to send to the started Activity
- that **information** is a data that can be **represented by a URI**

Extras Data

Intent extras are for **any other arbitrary data** you want to pass to the started Activity.

Intent extras **are stored in a Bundle object** as **key** and **value pairs**.

A **Bundle** is a **map**, optimized for Android, in which

- a **key** is a **string**
- a **value** can be **any primitive** or **object type**
 - objects must implement the Parcelable interface

Extras Data

To put data into the Intent extras you can

- **use** any of the Intent class **putExtra()** methods
- **create your own Bundle**
 - + put the **Bundle** into the Intent with **putExtras()**.

When use Extras Data

Use the **Intent extras**:

- If you want **to pass more than one piece of information** to the started Activity.
- If any of the information you want to pass **is not expressible by a URI**.

NOTE! **Intent data and extras are not exclusive**;

- you **can use data for a URI and**
- **extras for any additional information** the started Activity needs to process the data in that URI.

Passing data from one Activity to another

For **sending data** to an Activity:

- 1. Create the Intent object**
- 2. Put data or extras into that Intent**
- 3. Start the new Activity**

For receiving data from an Activity:

- 1. Get the Intent object, the Activity was started with**
- 2. Retrieve the data or extras from the Intent object**

Sending Data - Create the Intent object

Step 1: **Create the Intent object**

```
Intent messageIntent = new Intent(this,  
ShowMessageActivity.class);
```

Passing data from one Activity to another

For **sending data** to an Activity:

1. Create the Intent object
2. Put **data** or **extras** into that Intent
3. Start the new Activity

For **receiving data** from an Activity:

1. Get the Intent object, the Activity was started with
2. Retrieve the data or extras from the Intent object

Sending Data - Put data into that Intent

Step 2: **Put data into that Intent**

Use the `setData()` method with a URI object to add it to the Intent.

Some examples of using `setData()` with URIs:

```
// 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")));
```

Sending Data - Start the new Activity

Keep in mind that

- the data field **can only contain a single URI**
- If you **call setData() multiple times only the last value is used**
- You should use **Intent extras to include additional information (including additional URIs)**

Step 3: After you've added the data, you can start the new Activity with the Intent:

```
startActivity(messageIntent);
```

Passing data from one Activity to another

For **sending data** to an Activity:

1. Create the Intent object
2. Put data or **extras** into that Intent
3. Start the new Activity

For **receiving data** from an Activity:

1. Get the Intent object, the Activity was started with
2. Retrieve the data or extras from the Intent object

Sending Extras (with multiple putExtra)

Step 1: Create the Intent object (*as seen for data*)

Step 2: Put extras into that Intent

Use a `putExtra()` method with a **key** to put data into the Intent extras. The Intent class defines **many putExtra() methods for different kinds of data**, for example:

- `putExtra(String name, int value)`
⇒ `intent.putExtra("level", 406);`
- `putExtra(String name, String[] value)`
⇒ `String[] foodList = {"Rice", "Beans", "Fruit"};`
`intent.putExtra("food", foodList);`

Step 3: `startActivity(messageIntent);`

How can I pass an **object of a custom type** from one Activity to another using the `putExtra()`?

<https://stackoverflow.com/questions/2139134/how-to-send-an-object-from-one-android-activity-to-another-using-intents>

Example

```
public static final String EXTRA_MESSAGE_KEY =  
    "com.example.android.twoactivities.extra.MESSAGE";
```

Conventionally you define Intent extra keys as static variables with names that begin with EXTRA_. To guarantee that the key is unique, the string value for the key itself should be prefixed with your app's fully qualified class name.

```
Intent intent = new Intent(this, SecondActivity.class);  
String message = "Hello Activity!";  
intent.putExtra(EXTRA_MESSAGE_KEY, message);  
startActivity(intent);
```

Sending Extras (with a Bundle)

Step 2 (alternative): **if lots of data**

- first create a bundle and
- pass the bundle

Bundle defines many "put" methods for different kinds of primitive data as well as objects that implement Android's Parcelable interface or Java's Serializable

Sending Extras (with a Bundle)

```
Bundle extras = new Bundle();
extras.putString(EXTRA_MESSAGE, "this is my message");
extras.putInt(EXTRA_POSITION_X, 100);
extras.putInt(EXTRA_POSITION_Y, 500);
```

After you've populated the Bundle, add it to the Intent with the `putExtras()` method (note the "s" in Extras):

```
messageIntent.putExtras(extras);

startActivity(intent);
```

Passing data from one Activity to another

For **sending data** to an Activity:

1. Create the Intent object
2. Put data or extras into that Intent
3. Start the new Activity

For **receiving data** from an Activity:

1. Get the Intent object, the Activity was started with
2. Retrieve the data or extras from the Intent object

Get data from intents

When you **start an Activity with an Intent**, **the started Activity has access to the Intent** and the data it contains.

To retrieve the Intent the Activity (or other component) was started with, use the `getIntent()` method:

```
Intent intent = getIntent();
```

Get data and extras from intents

Data: Use `getData()` to get the URI from that Intent:

- `getData();`
⇒ `Uri locationUri = intent.getData();`

Extras: Use one of the `getExtra()` methods to extract extra data out of the Intent object:

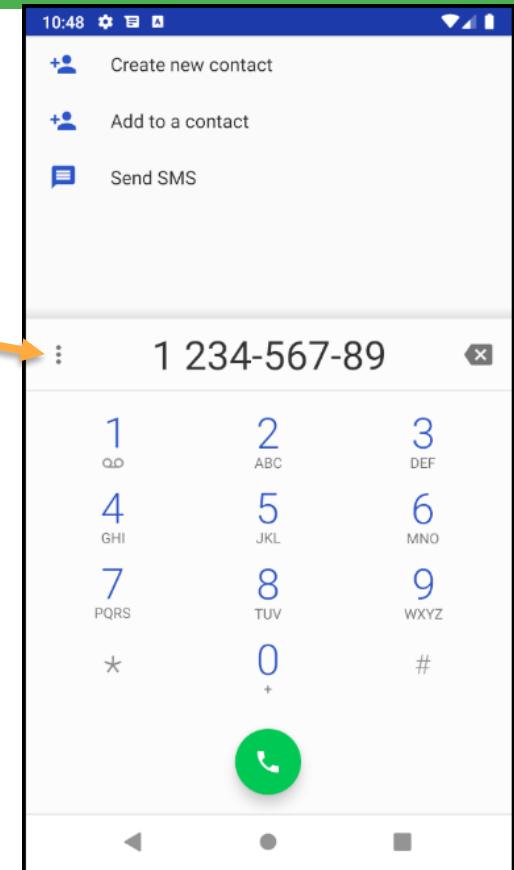
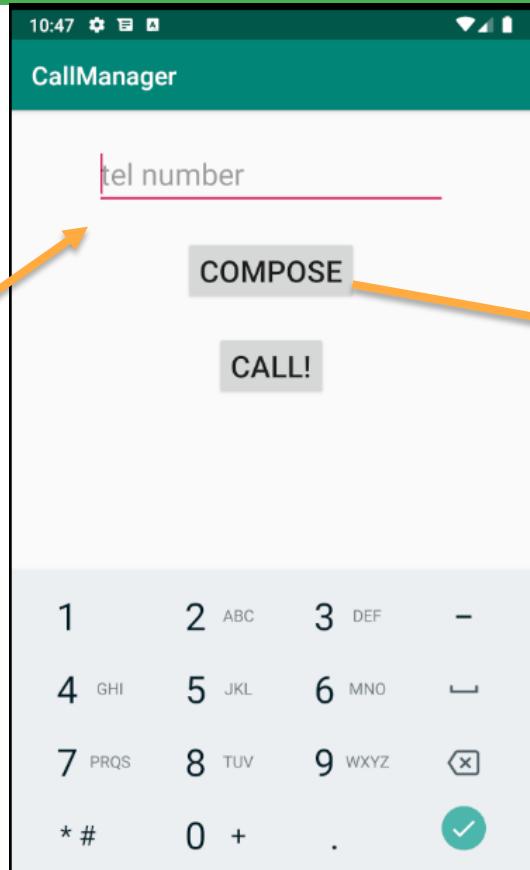
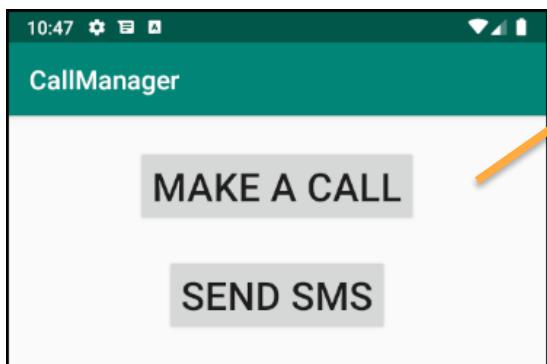
- `int getIntExtra (String name, int defaultValue)`
⇒ `int level = intent.getIntExtra("level", 0);`
- `Bundle bundle = intent.getExtras();`
⇒ Get all the data at once as a bundle.

Calls and SMS Manager Exercise

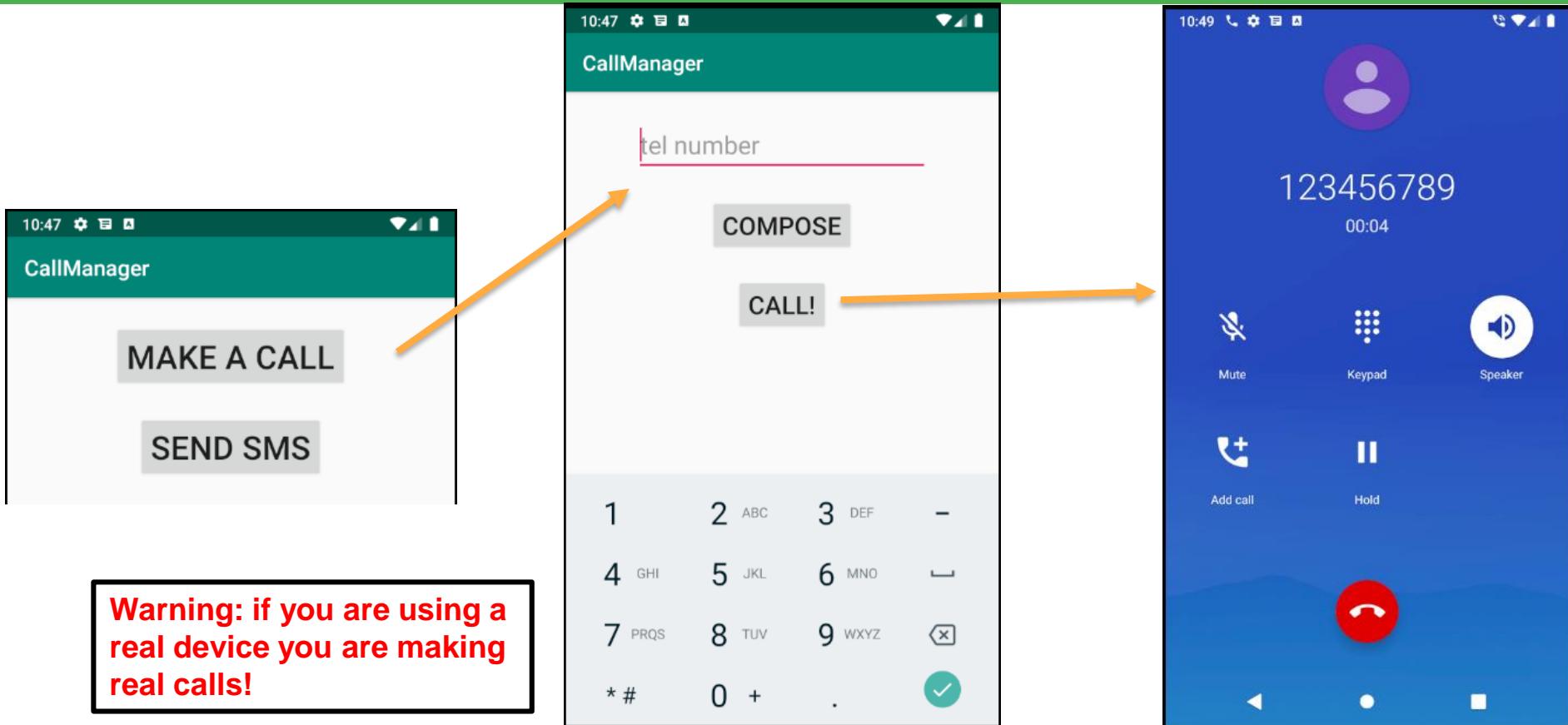
The goal of this exercise is to ***create an application*** that allows to:

- 1) Make a direct call** to a specific telephone number
- 2) Compose** a telephone number
- 3) Send an SMS** to a telephone number
- 4) Make a direct call** to a specific international telephone number
- 5) Compose** an international telephone number

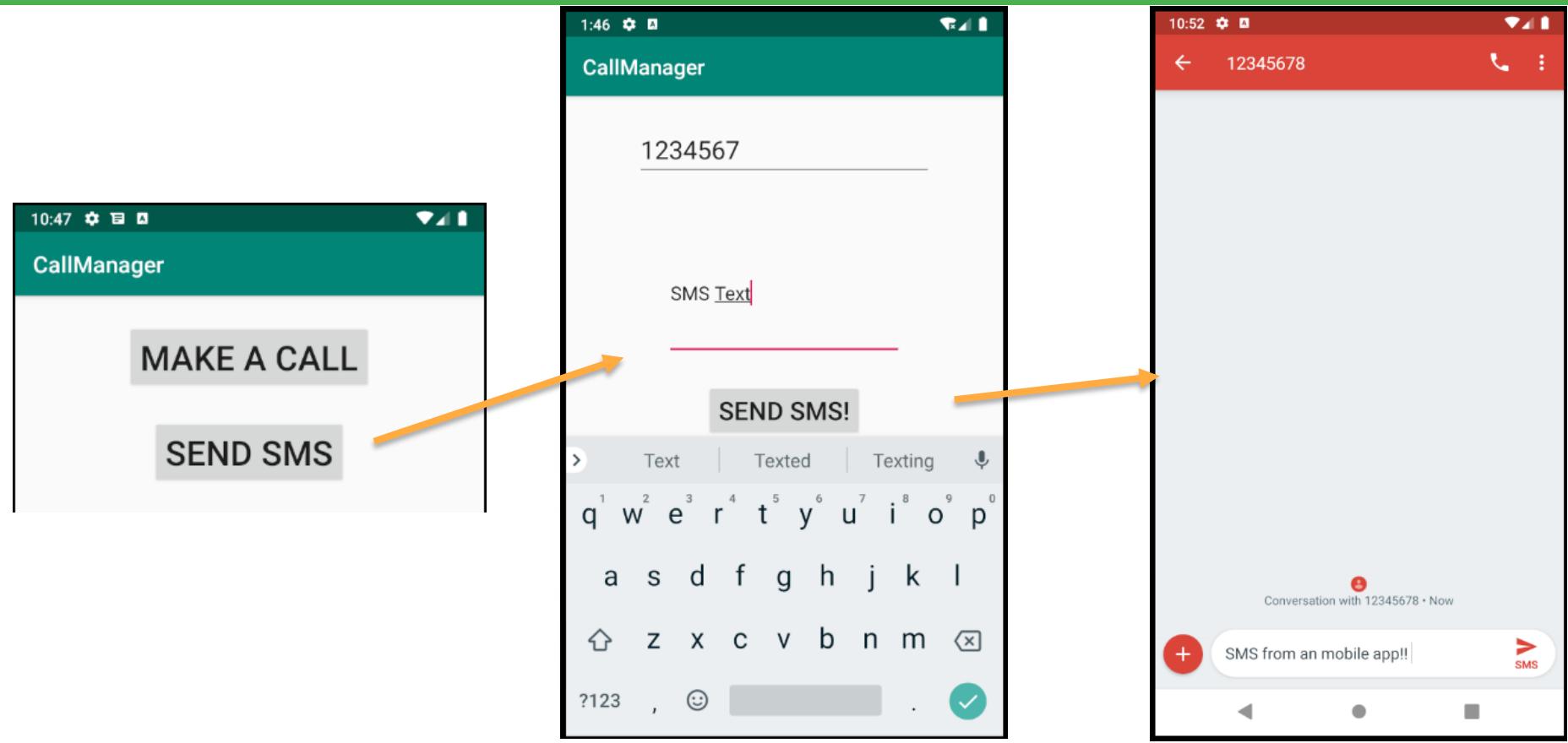
Calls and SMS Manager Exercise



Calls and SMS Manager Exercise



Calls and SMS Manager Exercise



Calls and SMS Manager Exercise

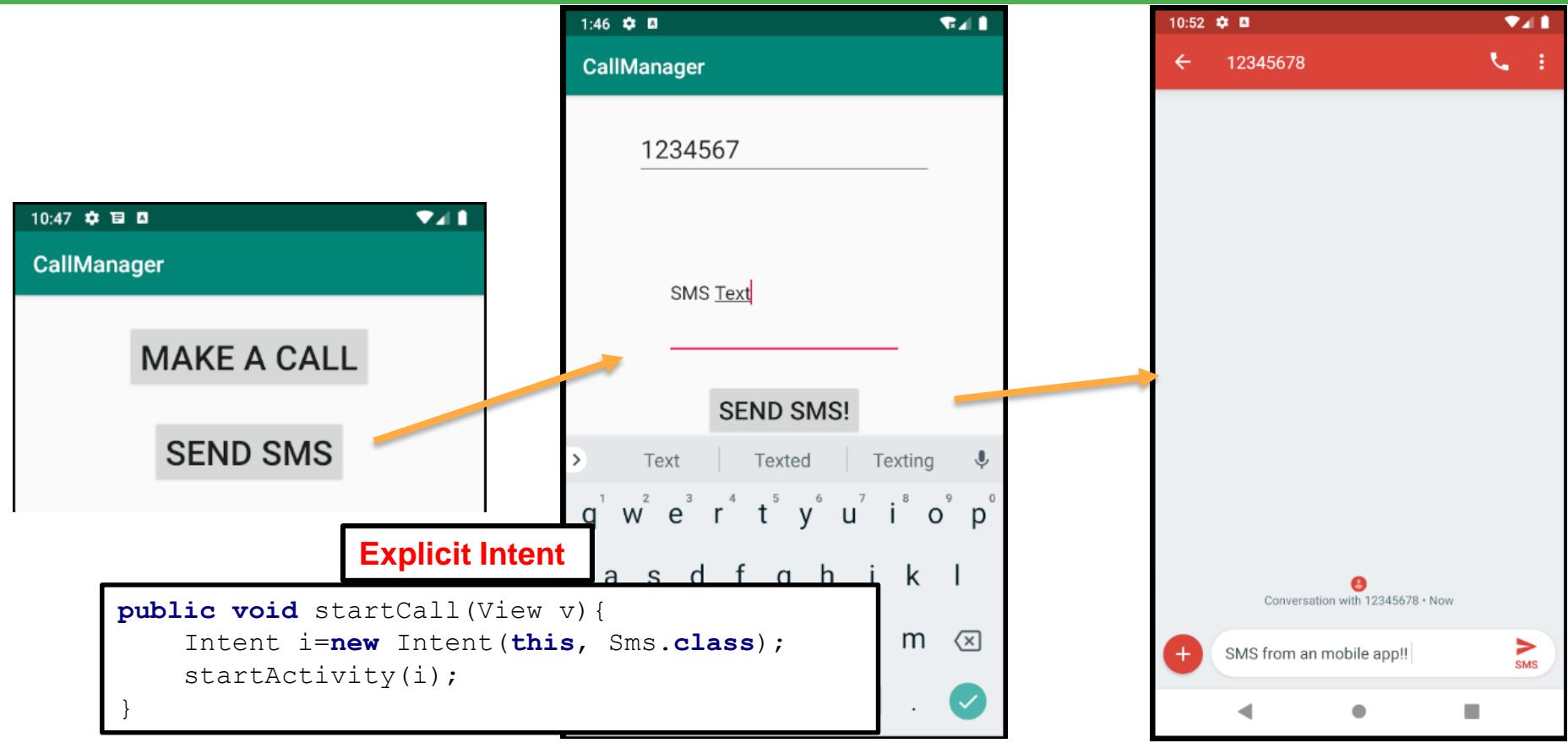
The image displays three screenshots of an Android application named "CallManager".

- Screenshot 1:** Shows a main screen with two buttons: "MAKE A CALL" and "SEND SMS". Below these buttons is a black box containing the text "Explicit Intent".
- Screenshot 2:** Shows a "COMPOSE" screen with a text input field labeled "tel number" and a "CALL!" button.
- Screenshot 3:** Shows a dial pad screen with a list of contacts on the left: "Create new contact", "Add to a contact", and "Send SMS". The phone number "1 234-567-89" is displayed on the dial pad.

An orange arrow points from the "Explicit Intent" box in Screenshot 1 to the "tel number" input field in Screenshot 2, indicating the intent to make a call.

```
public void startCall(View v) {
    Intent i=new Intent(this, Call.class);
    startActivity(i);
}
```

Calls and SMS Manager Exercise



Calls and SMS Manager Exercise

The diagram illustrates the flow of an implicit intent for making a call. It consists of three screens:

- Screen 1:** Shows a "CallManager" title bar and two buttons: "MAKE A CALL" and "SEND SMS".
- Screen 2:** Shows a "CallManager" title bar with an "EditText" labeled "tel number" and two buttons: "COMPOSE" and "CALL!". This screen is overlaid with a red box containing the text "Implicit Intent" and a code snippet.
- Screen 3:** Shows a "Dialer" interface with a list of options: "Create new contact", "Add to a contact", and "Send SMS". Below the list is a numeric keypad and a recipient number "1 234-567-89".

An orange arrow points from the "MAKE A CALL" button in Screen 1 to the "COMPOSE" button in Screen 2. Another orange arrow points from the "CALL!" button in Screen 2 to the "tel number" field in Screen 3.

```
public void compose (View v)
{
    EditText edn = (EditText) findViewById(R.id.editTextNumber);
    Intent intentImplicit=new Intent(Intent.ACTION_DIAL);
    String uri="tel:"+edn.getText().toString();
    intentImplicit.setData(Uri.parse(uri));
    startActivity(intentImplicit);
}
```

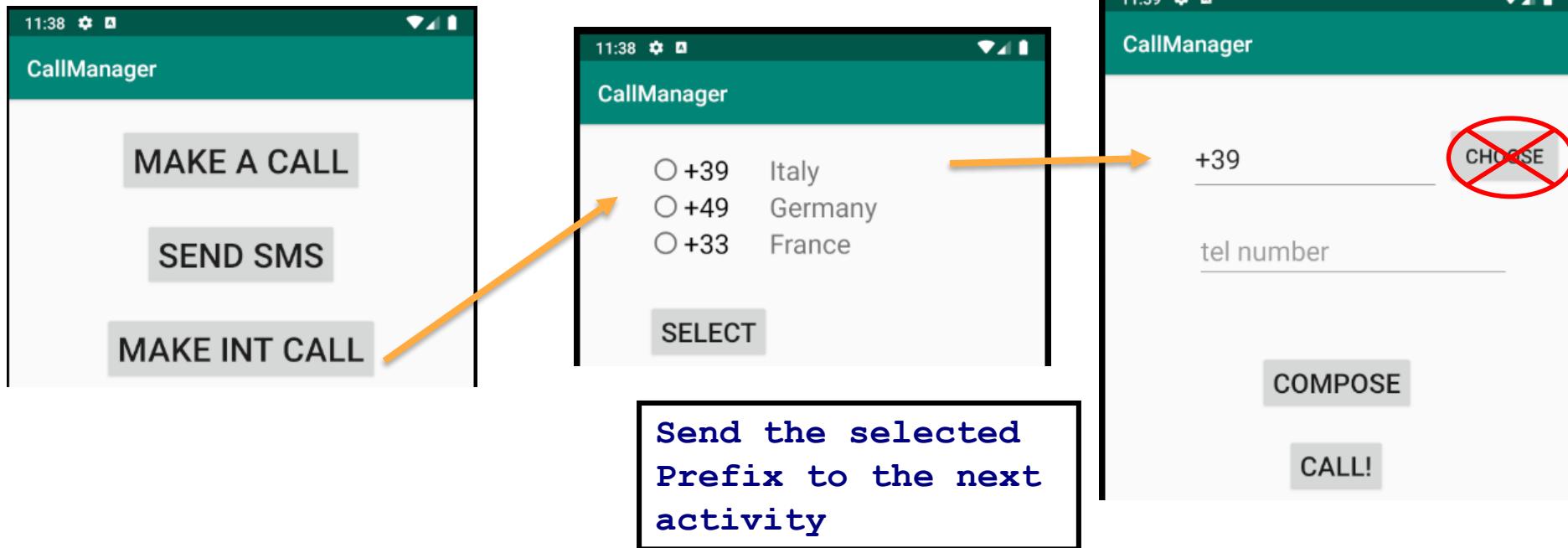
Calls and SMS Manager Exercise

```
public void makeCall (View v)
{
    EditText edn = (EditText) findViewById(R.id.editTextNumber);
    Intent intentImplicit=new Intent(Intent.ACTION_CALL);
    String uri="tel:"+edn.getText().toString();
    intentImplicit.setData(Uri.parse(uri));
    try {
        startActivity(intentImplicit);
    } catch (SecurityException e) {
        ActivityCompat.requestPermissions(
            Call.this,
            new String[] {Manifest.permission.CALL_PHONE},
            1);
    }
    return;
}
```

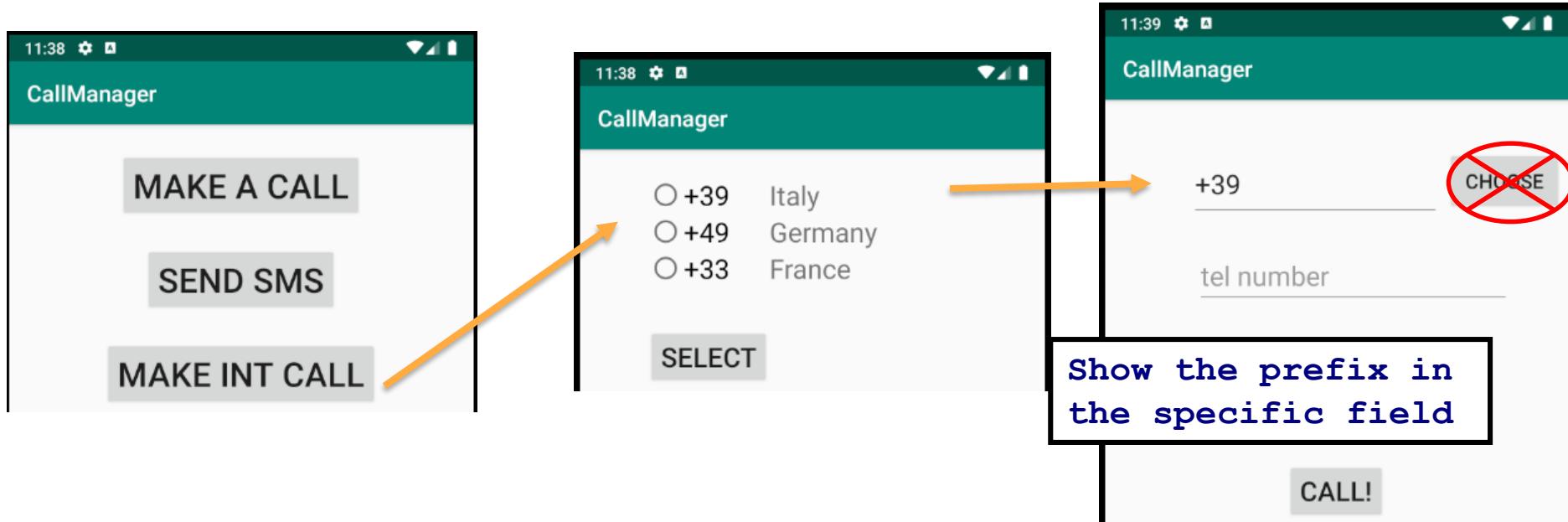
To make a call you need to declare a permission in the manifest....

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

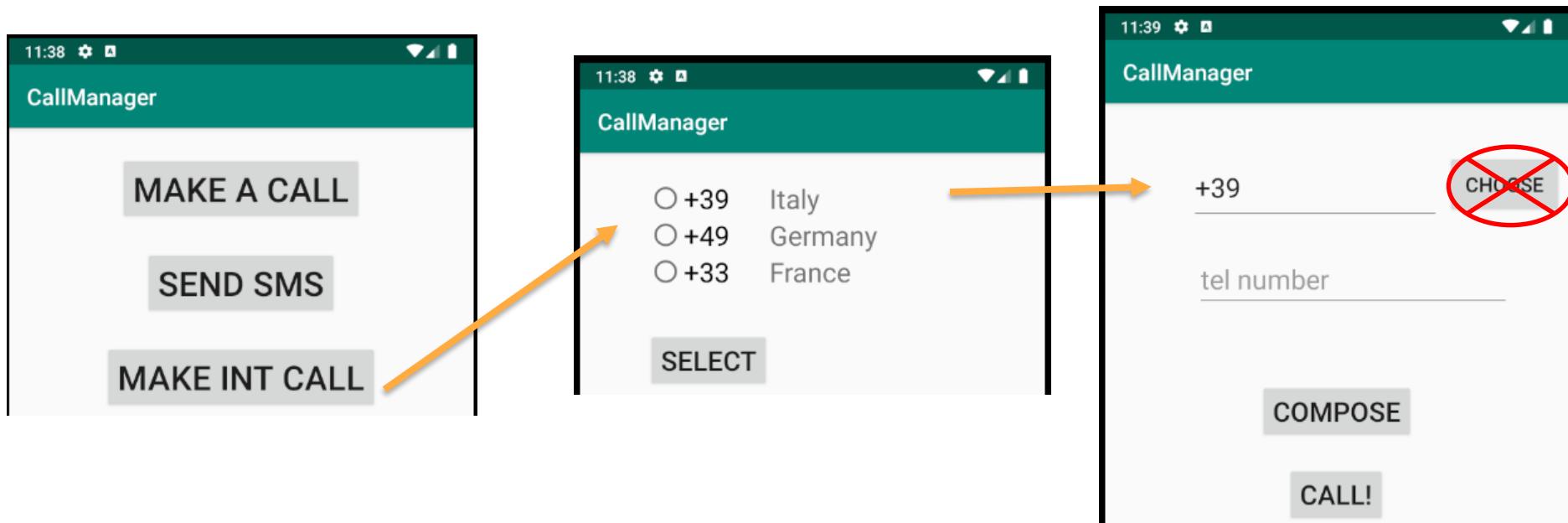
Calls and SMS Manager Exercise (V.2)



Calls and SMS Manager Exercise (V.2)



Calls and SMS Manager Exercise (V.2)



The prefix is NOT an URI =>
use Extras

Calls and SMS Manager Exercise (V.2)

```
public void selectPrefix(View arg0) {  
    Intent intent = new Intent(this, CallInternational.class);  
    String prefix;  
    ...  
    [ to do ]  
    ...  
    intent.putExtra("prefix", prefix);  
    startActivity(intent);  
}  
  
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    ...  
    Intent intent = getIntent();  
    String prefix = intent.getStringExtra("prefix");  
    TextView prefixText = findViewById(R.id.editTextPrefix);  
    prefixText.setText(prefix);  
}
```

In the
prefix
activity

In the activity
that allows to
compose
international
numbers



Activity lifecycle and state

Contents

- Activity lifecycle
- Activity lifecycle callbacks

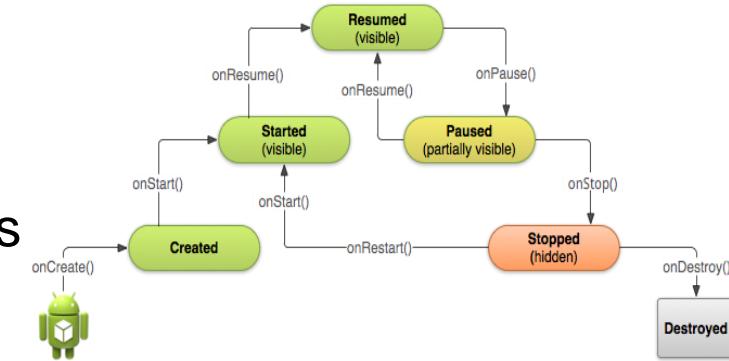
These slides are partially based on the material that Google provides for the course
Android Developer Fundamentals

What is the Activity Lifecycle?

The **Activity Lifecycle** is the **set of states** an **Activity can assume** in **during its lifetime**

- from the *time it is created*
- to when *it is destroyed*

and the system reclaims its resources

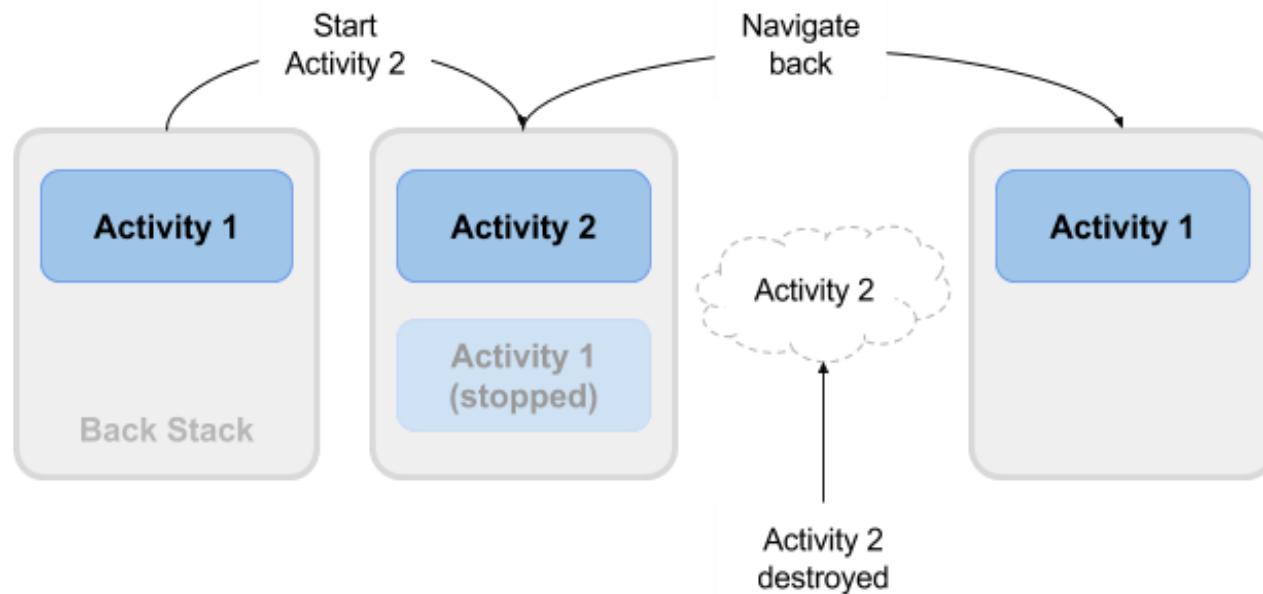


More **formally**, a *directed graph* where

- **Nodes**: all the states an Activity assumes
- **Edges**: the callbacks associated with transitioning from each state to the next one

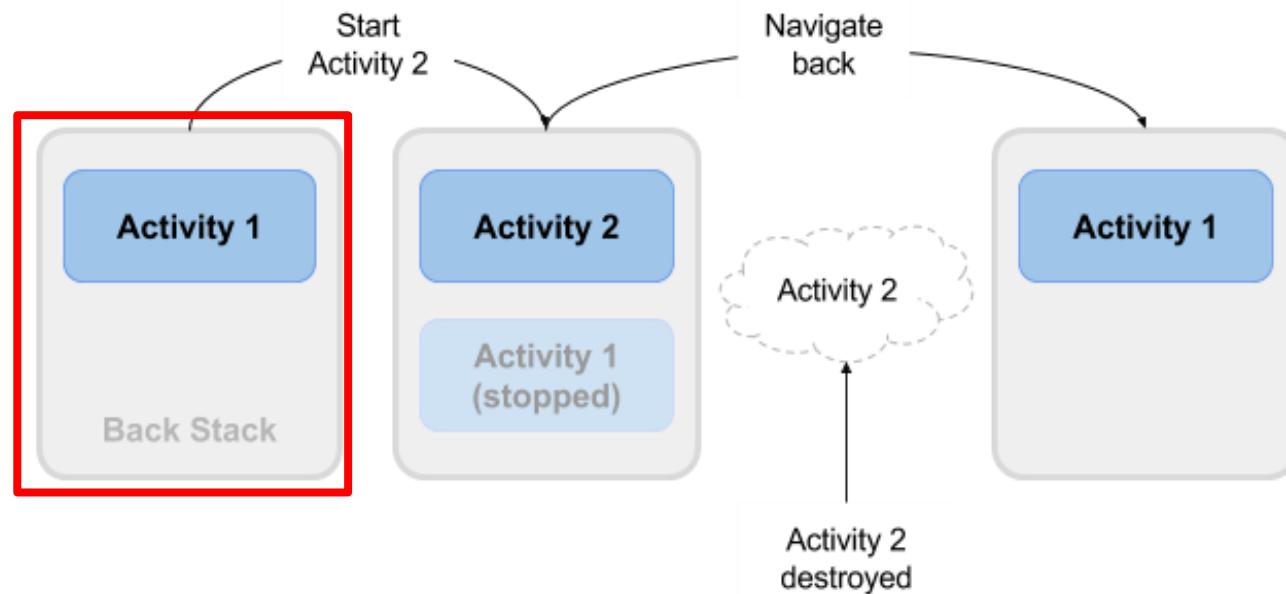
What is the Activity Lifecycle?

As the **user interacts** with the app or other apps on the device,
activities move into different states.



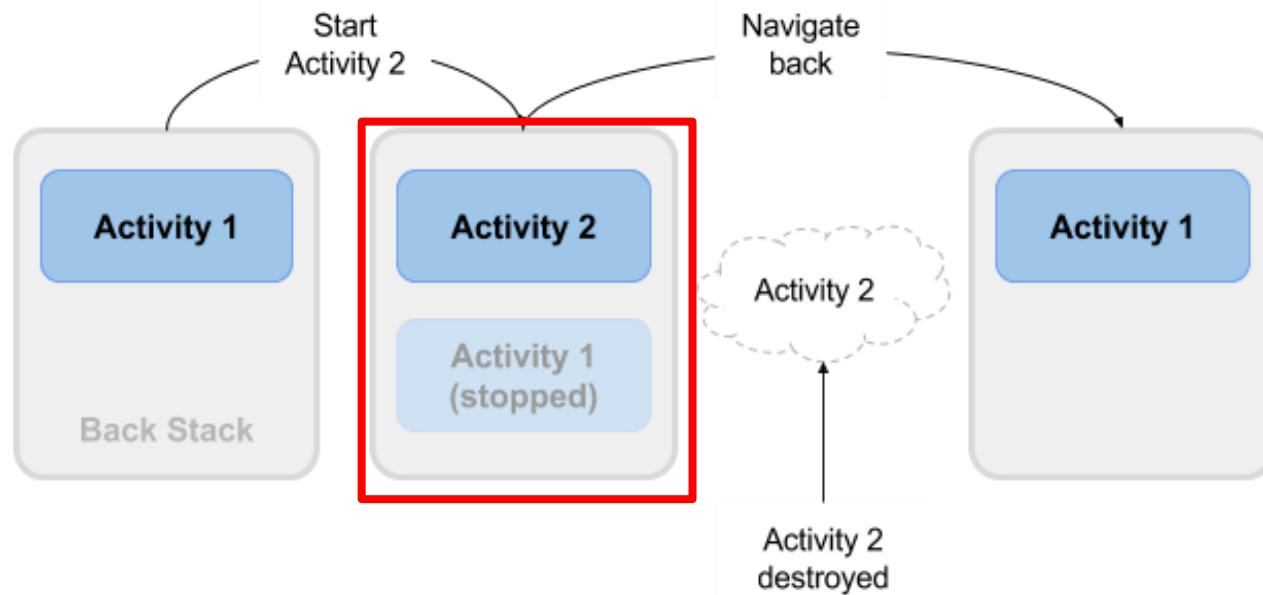
What is the Activity Lifecycle?

When the app **starts**, the app's **main activity** is **started**, **comes to the foreground**, and **receives the user focus**.



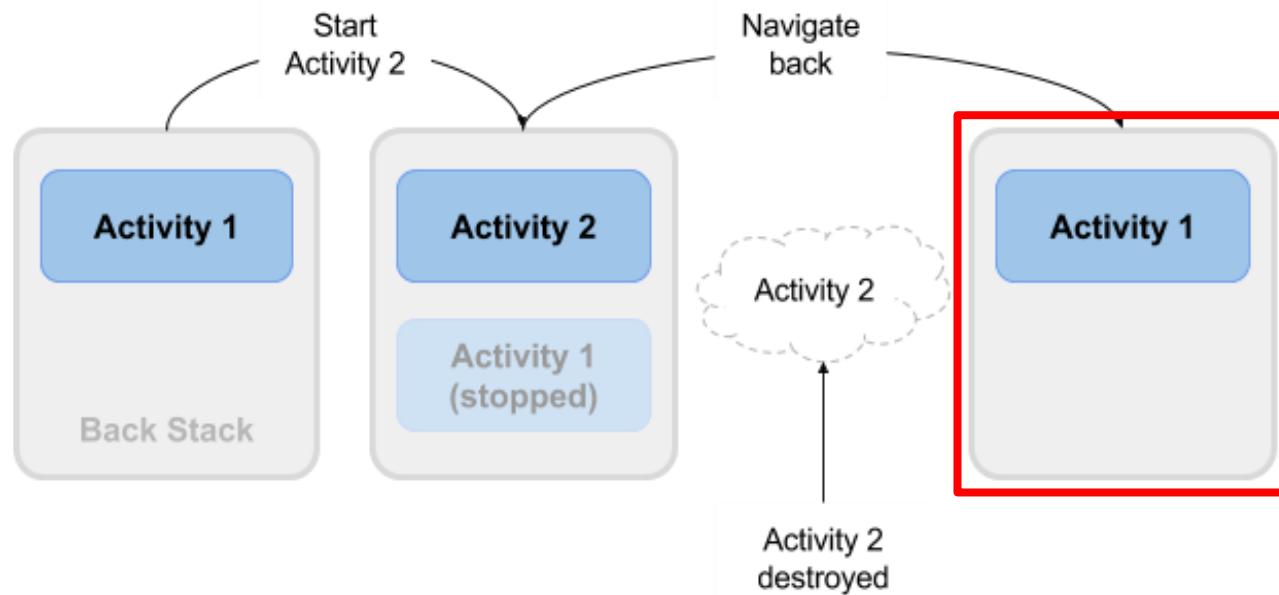
What is the Activity Lifecycle?

When a second activity **starts**, the new activity is **created** and **started**, and the main activity is **stopped**.



What is the Activity Lifecycle?

When the user **finishes to interact** with the **Activity 2** and **navigate back**, **Activity 1 resumes**. Activity 2 stops and is no longer needed.



Activity lifecycle callbacks

Callbacks and when they are called

When an **Activity** make a **transitions into** and **out of the different lifecycle states** as it runs, the **Android system calls several lifecycle callback** methods at each stage.

`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

Callbacks and when they are called

When an **Activity** make a **transitions into** and **out of the different lifecycle states** as it runs, the **Android system calls several lifecycle callback** methods at each stage.

`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 app visibility

The **activity can be visible or not** depending on the current state:

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

State changes are triggered by:

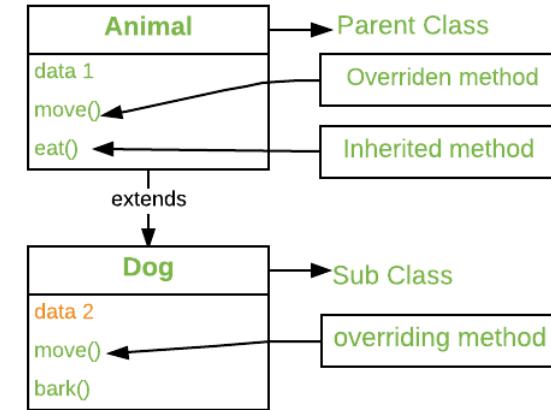
- user actions (e.g., back button)
- configuration changes (e.g. device rotation)
- system actions

Activity states and lifecycle callback methods

All of the **callback methods** are **hooks** that **can be overridden** in each Activity class **to define how that Activity behaves when state transitions happens:**
e.g., when the user **leaves** and **re-enters** the Activity.

Keep in mind that the **lifecycle states** (and callbacks) are per **Activity**, not per app, and you *may implement different behaviors* at different points in the lifecycle of each Activity:

e.g. VideoApp => streaming activity VS settings activity

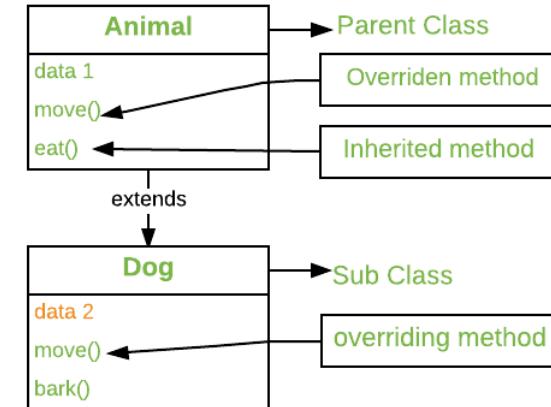


Activity states and lifecycle callback methods

All of the **callback methods** are **hooks** that **can be overridden** in each Activity class **to define how that Activity behaves when state transitions happens:**
e.g., when the user **leaves** and **re-enters** the Activity.

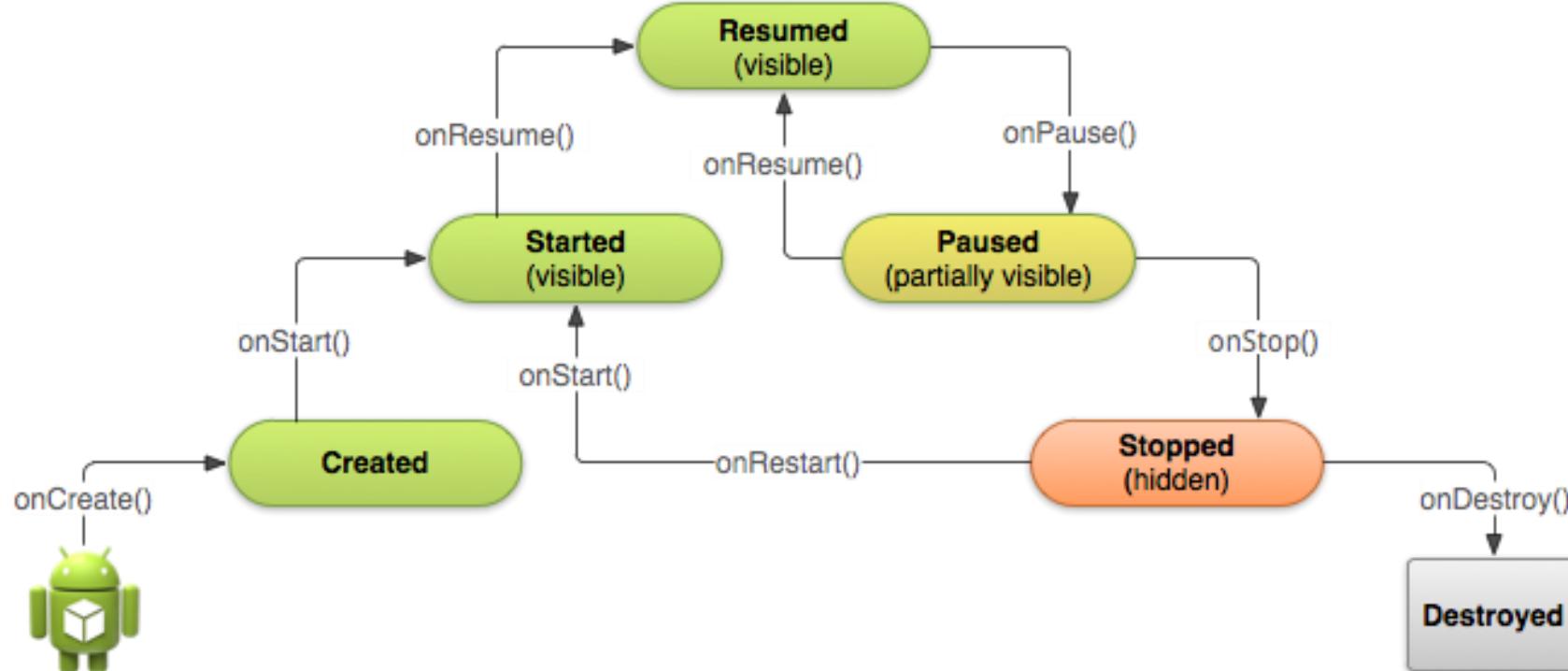
Keep in mind that the **lifecycle states** (and callbacks) **are per Activity, not per app**, and you **may implement different behaviors** at different points in the lifecycle of each Activity:

e.g. VideoApp => streaming activity VS settings activity



Activity states and callbacks graph

This figure **shows each of the Activity states and the callback methods** that occur as the Activity transits between different states



Implementing and overriding callbacks

In general **it is not needed to implement all the lifecycle callback methods.**

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

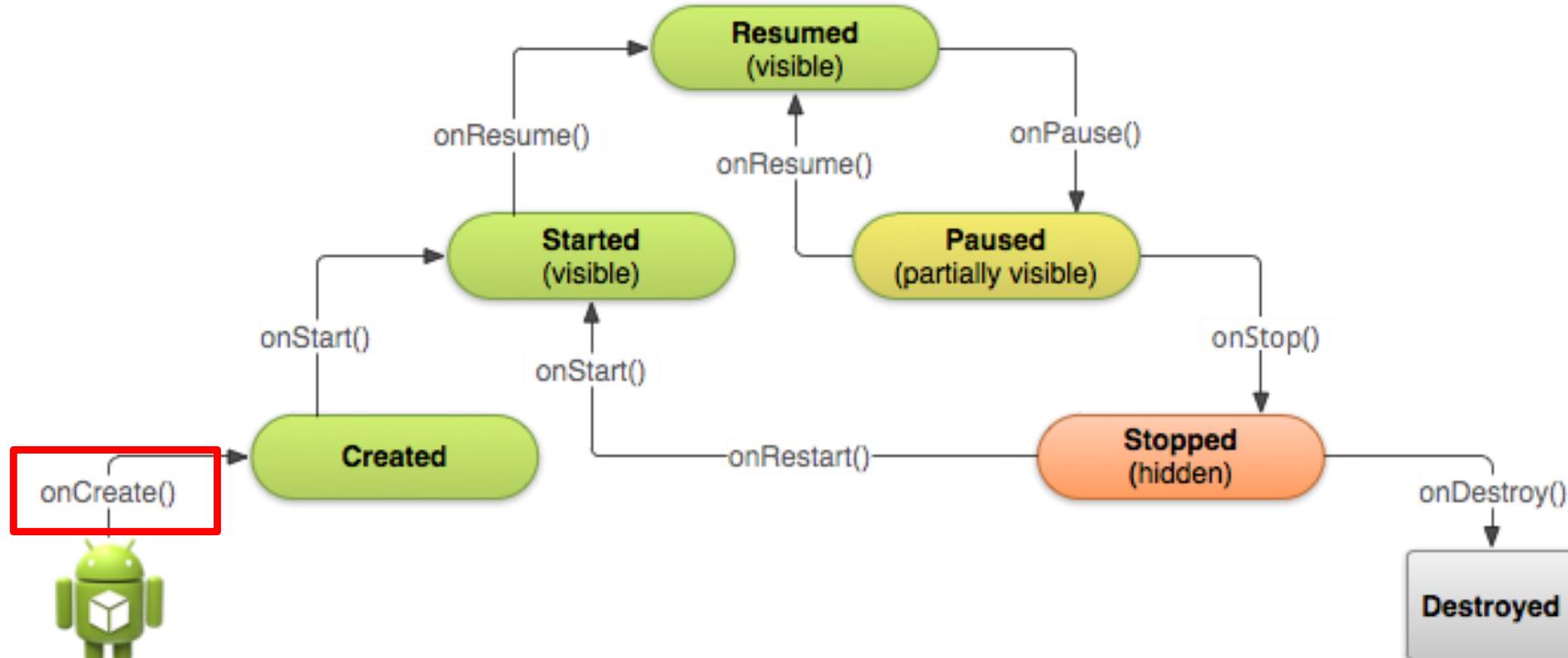
Implementing and overriding callbacks

- However, it's important to understand each one and implement those that ensure your app behaves the way users expect
- Managing the lifecycle of an Activity by implementing callback methods is crucial to developing a strong and flexible app
 - Examples: avoiding crashes, consuming system resources not required in specific states, save user progress in the app usage, saving the app state when the screen rotates, etc...

Implementing and overriding callbacks

- However, it's important to understand each one and implement those that ensure your app behaves the way users expect
- Managing the lifecycle of an Activity by implementing callback methods is crucial to developing a strong and flexible app
 - Examples: avoiding crashes, consuming system resources not required in specific states, save user progress in the app usage, saving the app state when the screen rotates, etc...

Activity states and callbacks graph



onCreate() → Created

- Called when **the Activity is first created**
 - for example when user taps launcher icon
- Similar to the main() method in other programs
- Does **all static setup**, perform basic app startup logic such as
 - setting up the user interface
 - assigning class-scope variables
 - setting up background tasks

onCreate() → Created

- Called when **the Activity is first created**
 - for example when user taps launcher icon
- Similar to the main() method in other programs
- Does **all static setup**, perform basic app startup logic such as
 - setting up the user interface
 - assigning class-scope variables
 - setting up background tasks

onCreate() → Created

- Called when **the Activity is first created**
 - for example when user taps launcher icon
- Similar to the main() method in other programs
- Does **all static setup**, perform basic app startup logic such as
 - **setting up the user interface**
 - **assigning class-scope variables**
 - **setting up background tasks**

onCreate() → Created

- 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()
 - Created is a transient state; the **Activity remains in the created state only as long as it takes to run onCreate()**, and then the Activity moves to the started state.

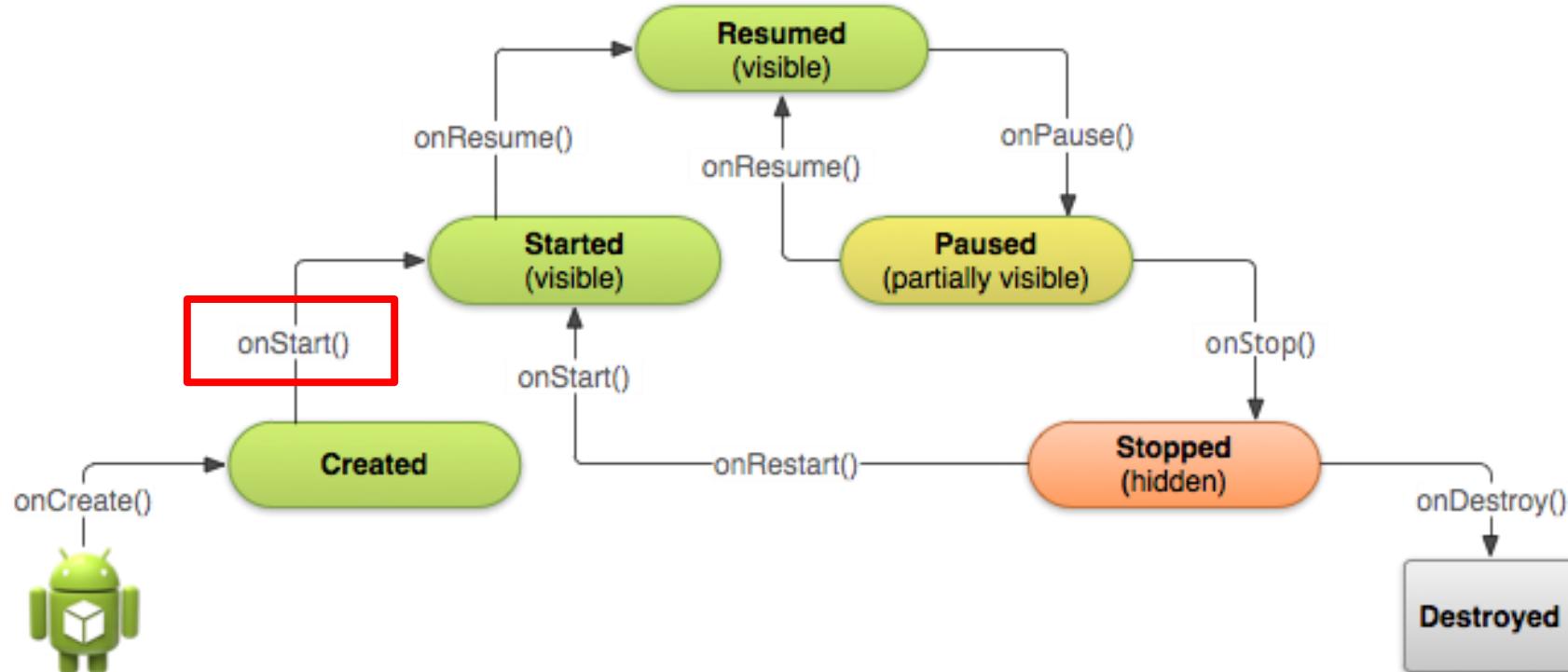
onCreate(Bundle savedInstanceState)

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    // The activity is being created.  
  
    // set the user interface layout for this activity  
    // the layout file is defined in the project  
        res/layout/main_activity.xml file  
    setContentView(R.layout.main_activity);  
}
```

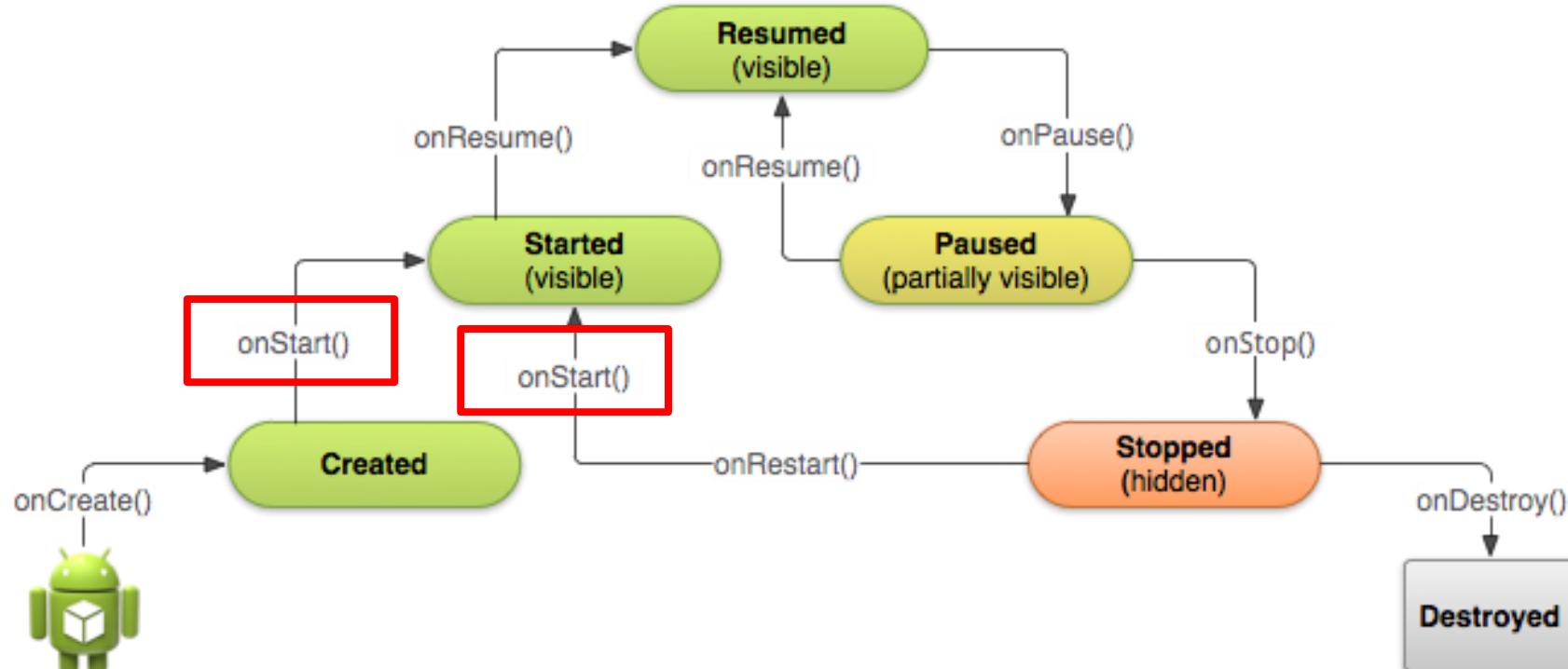
onCreate(Bundle savedInstanceState)

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    // The activity is being created.  
  
    // set the user interface layout for this activity  
    // the layout file is defined in the project  
        res/layout/main_activity.xml file  
    setContentView(R.layout.main_activity);  
}
```

Activity states and callbacks graph



Activity states and callbacks graph



onStart() → Started

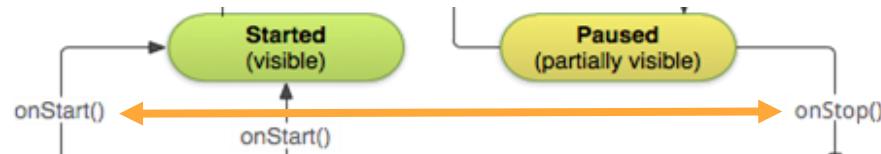
- Called when the **Activity is becoming visible to user**
- Can be **called more than once during lifecycle**
 - While onCreate() is called only once when the Activity is created, the onStart() method may be called many times during the lifecycle of the Activity **as the user navigates around the app**

onStart() → Started

- **Started**, like created, **is a transient state**
 - After starting, the **Activity moves into the resumed** (running) state
- Typically you implement **onStart()** in an Activity **as a counterpart** to the **onStop()** method.
- For example,
 - if you **release hardware resources** (such as GPS or sensors) **when the Activity is stopped**
 - you can **re-register those resources** in the **onStart()** method

onStart() → Started

- **Started**, like created, **is a transient state**
 - After starting, the **Activity moves into the resumed** (running) state
- Typically you implement **onStart()** in an Activity **as a counterpart** to the **onStop()** method.

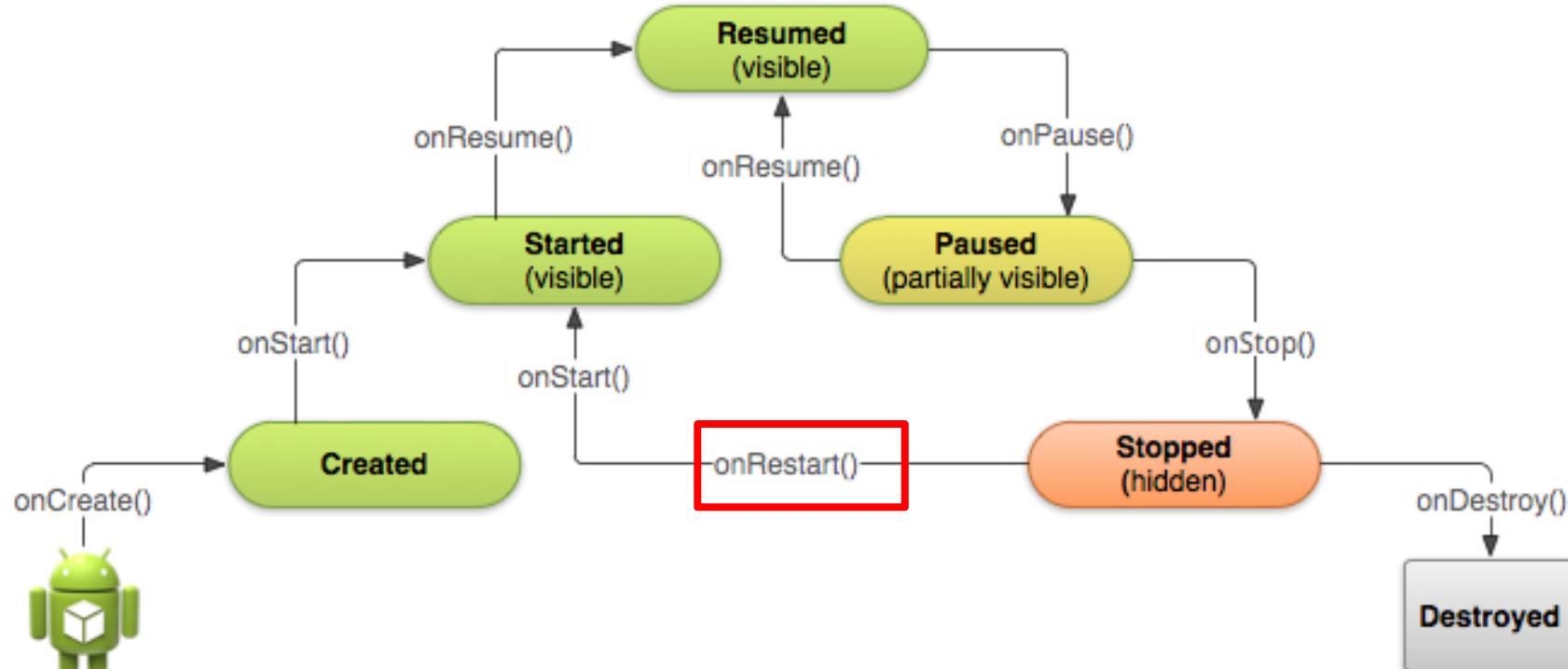


- For example,
 - if you **release hardware resources** (such as GPS or sensors) **when the Activity is stopped**
 - you can **re-register those resources** in the **onStart()** method

onStart()

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

Activity states and callbacks graph



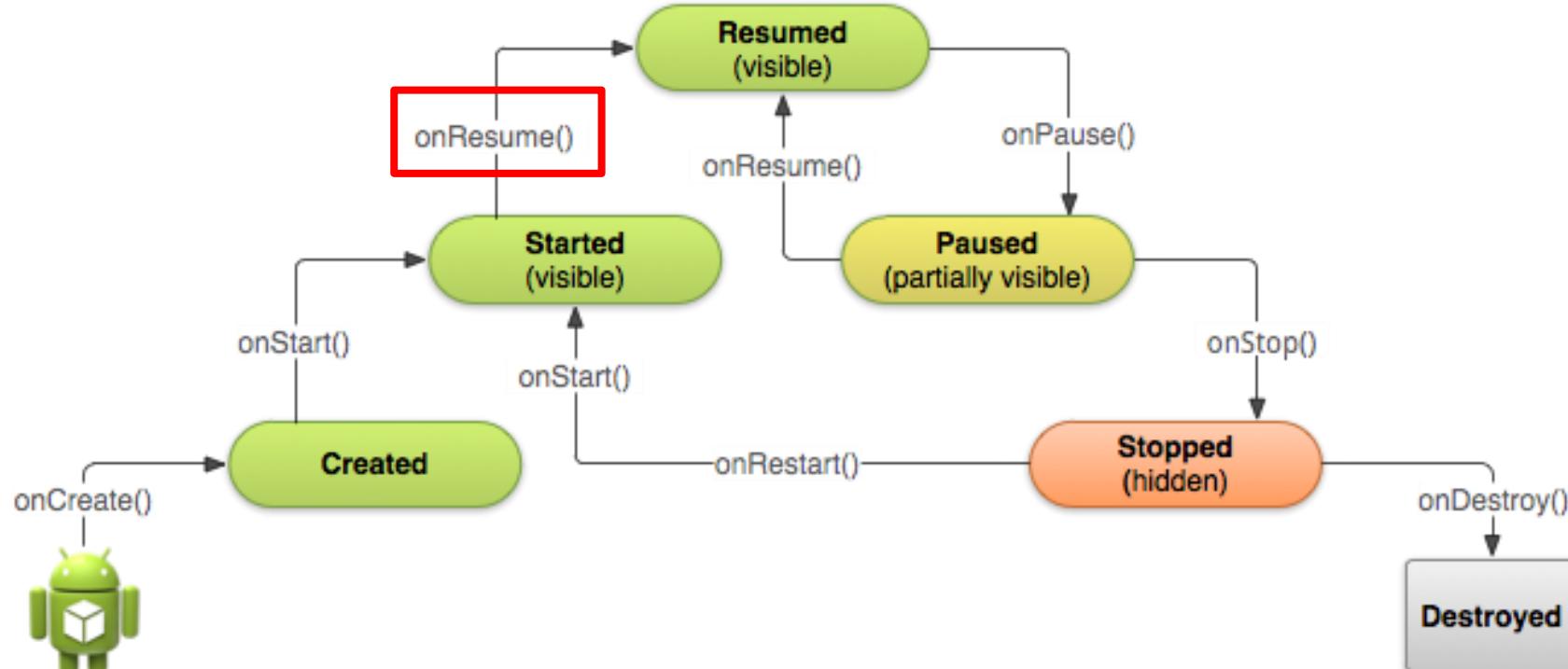
onRestart() → Started

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

onRestart()

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

Activity states and callbacks graph

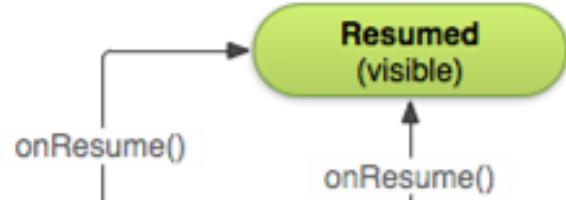


onResume() → Resumed/Running

An Activity is in the **resumed state** when it is **initialized**, **visible on screen**, and **ready to use**.

The **resumed state** is often called the *running state*, because it is in this state that the **user is actually interacting with the app**.

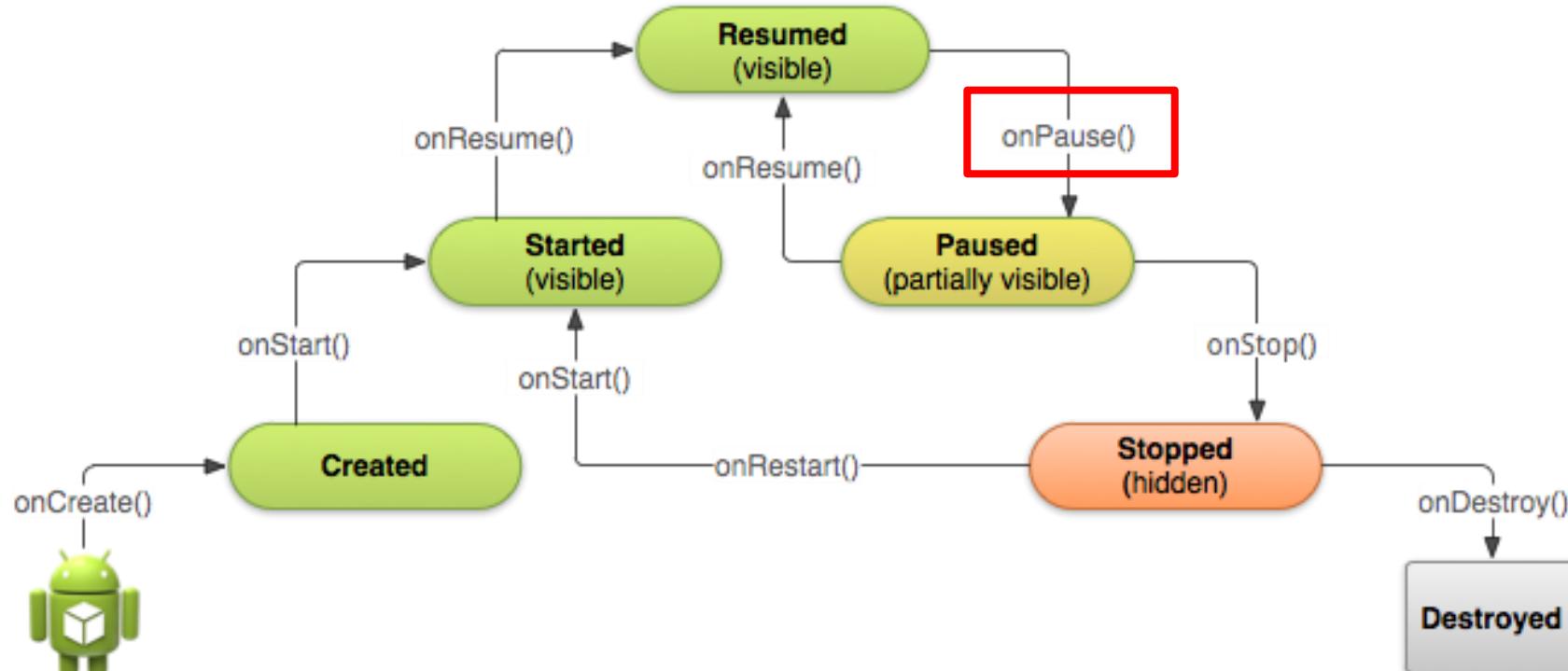
- Called **when Activity will start interacting with user**
- Activity **has moved to top of the Activity stack** (i.e., it is visible)
- Starts accepting user input
- Running state
- Always followed by `onPause()`



onResume()

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

Activity states and callbacks graph



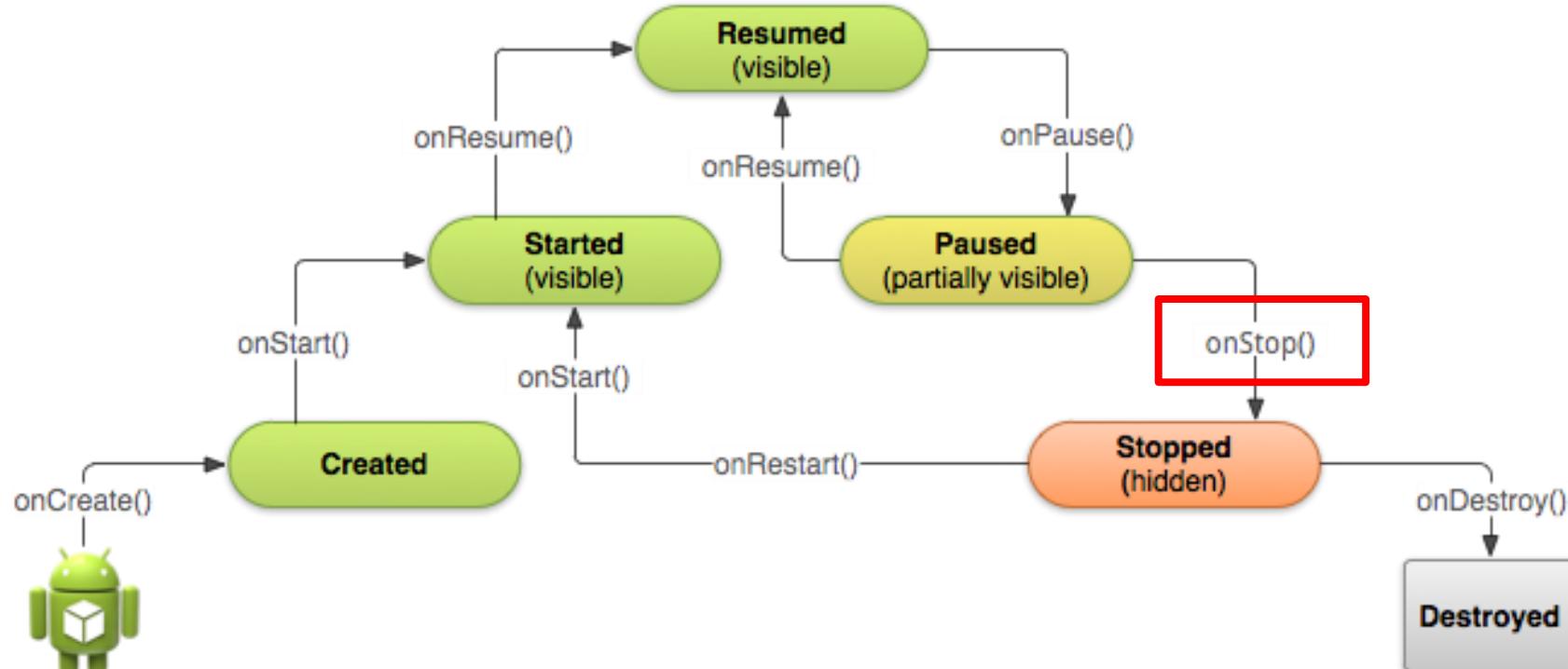
onPause() → Paused

- Called when **system is about 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"  
}
```

Activity states and callbacks graph



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
 - this one is being destroyed
- Operations that were too heavy-weight for onPause()
- Followed by either onRestart() if Activity is coming back to interact with user, or onDestroy() if Activity is going away

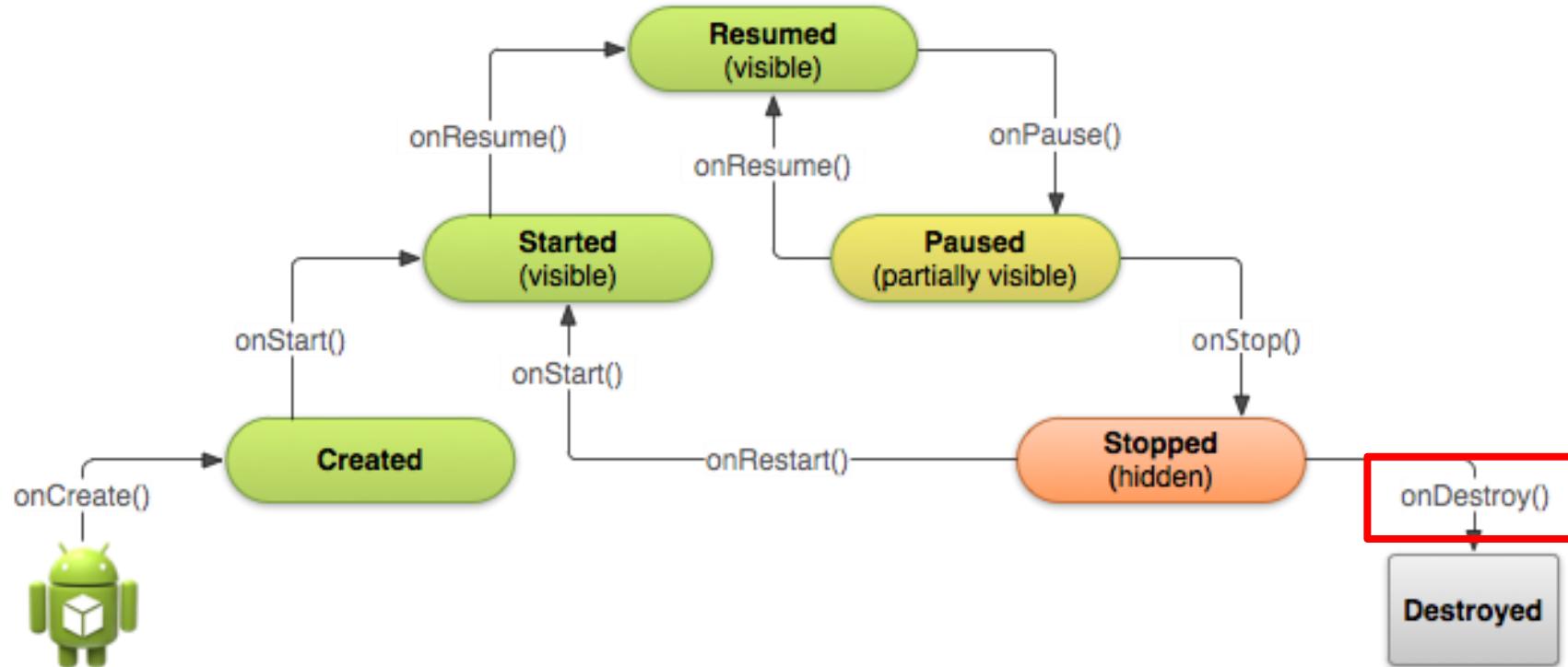
onStop() -> Stopped

- 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
 - this one is being destroyed
- **Operations that were too heavy-weight for onPause()**
- Followed by either onRestart() if Activity is coming back to interact with user, or onDestroy() if Activity is going away

onStop()

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

Activity states and callbacks graph



onDestroy() → Destroyed

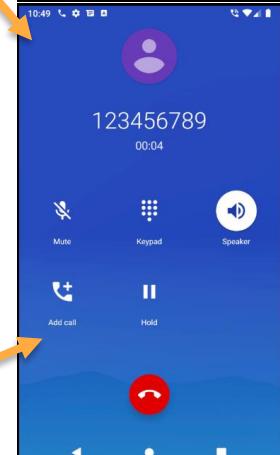
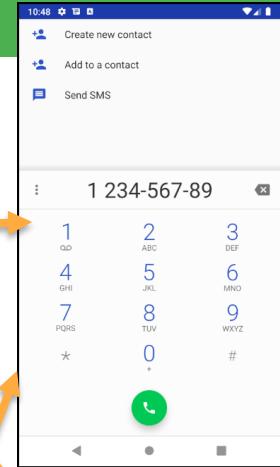
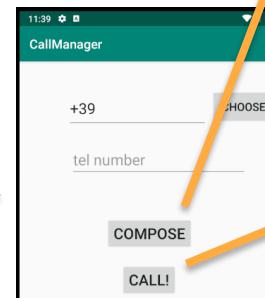
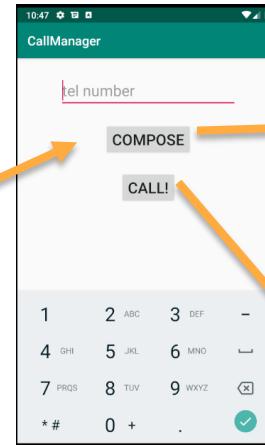
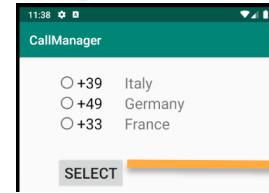
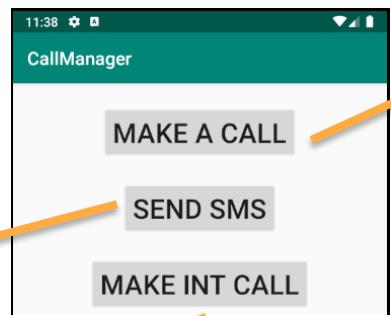
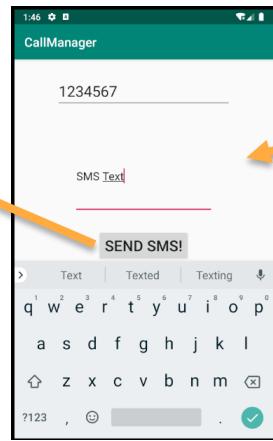
- Final **call before Activity is destroyed**
- **User navigates back to previous Activity**, or configuration changes
- Activity is finishing or system is destroying it to save space
- **System may destroy Activity without calling** this, so use onPause() or onStop() to save data or state

onDestroy()

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

Exercise: log States of Activities in Call Manager

App navigational structure



Exercise: log States of Activities in Call Manager

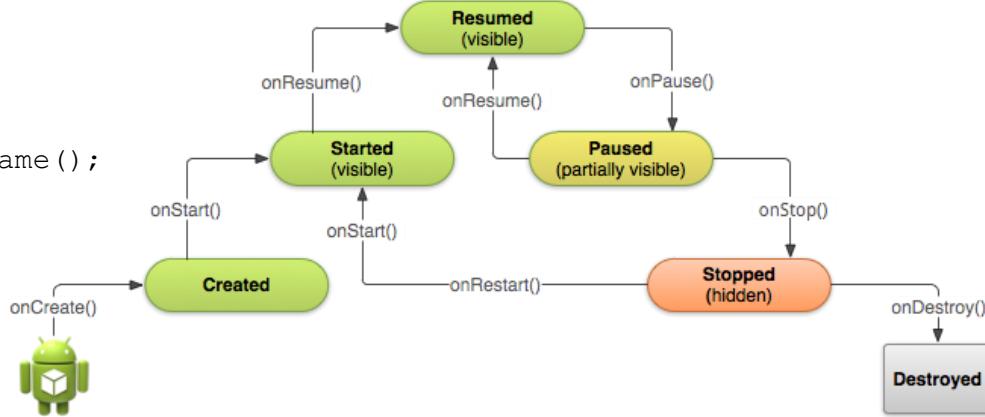
Log all the states of all the activities composing the app.

You can use a code like this:

```
// Class name for Log tag.  
private static final String LOG_TAG =  
    {ClassName}.class.getSimpleName();  
  
@Override  
protected void onRestart() {  
    super.onRestart();  
    Log.d(LOG_TAG, "onRestart");  
}
```

Analyze what happens when:

- You start the app
- You move from an activity to another
- You return on a previous activity
- You open other apps in your smartphone and then return to the app
- You move from a portrait to a landscape layout (add landscape variant to one of the activities)
- You perform a complex computation in onPause (i.e., something requiring 2-3 sec)





Activity instance state

When does config change?

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

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

When does config change?

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

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

What happens on config change?

On configuration change, Android:

1. Shuts down Activity

by calling:

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

2. Starts Activity over again

by calling:

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

Activity instance state

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

What the system saves

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

What the system saves

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

Saving instance state

Implement `onSaveInstanceState()` in your Activity

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

onSaveInstanceState(Bundle outState)

```
@Override  
protected void onSaveInstanceState(Bundle outState) {  
    super.onSaveInstanceState(outState);  
  
    // save info  
    outState.putString("count",  
                      String.valueOf(mShowCount.getText()));  
}
```

onSaveInstanceState(Bundle outState)

```
@Override  
protected void onSaveInstanceState(Bundle outState) {  
    super.onSaveInstanceState(outState);  
  
    // save info  
    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);  
  
    if (savedInstanceState != null) {  
        username = savedInstanceState.getString("user");  
    }  
}
```

onRestoreInstanceState(Bundle state)

```
@Override  
protected void onRestoreInstanceState (Bundle mySavedState) {  
    super.onRestoreInstanceState(mySavedState);  
  
    if (mySavedState != null) {  
        username = mySavedState.getString("user");  
    }  
}
```

Instance state and app restart

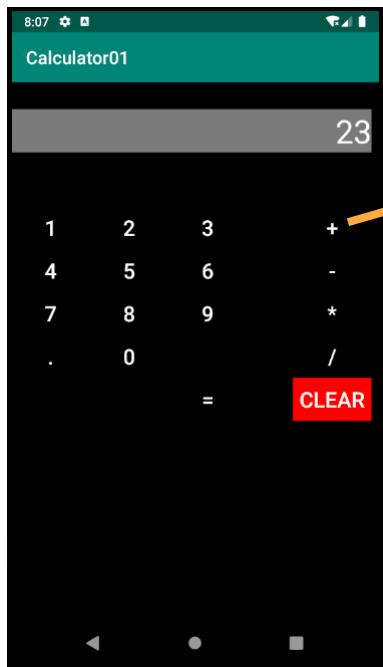
When you stop and restart a new app session, the Activity instance states are lost and your activities will revert to their default appearance

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

(we will see them in future lessons)

Exercise: Refine the Calculator App

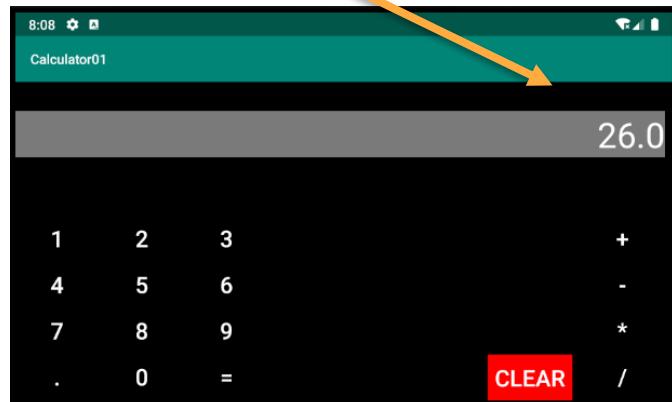
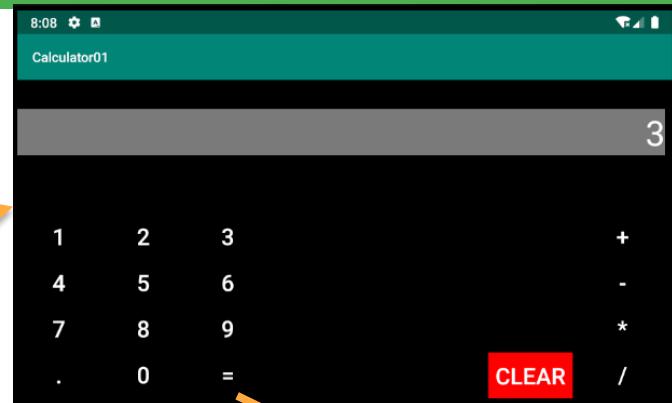
If you support both portrait and landscape layouts (if not add this feature), you have to improve the app to manage these layout changes during computations.



23 + (layout change) 3=26

+
and rotate screen

You have to save and restore the internal values of the app during the layout change, otherwise you lost the data inserted...





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

Activities and Intents (part II)

Contents

In the previous lessons we have seen how to

- Start a **new Activity** with an Intent (explicit / implicit)
- Pass **info between activities** with data or extras

In this lesson we are going to see how to

- Get **data back from** an Activity

These slides are partially based on the material that Google provides for the course

Android Developer Fundamentals

Returning data to the starting activity

When starting an Activity with an Intent:

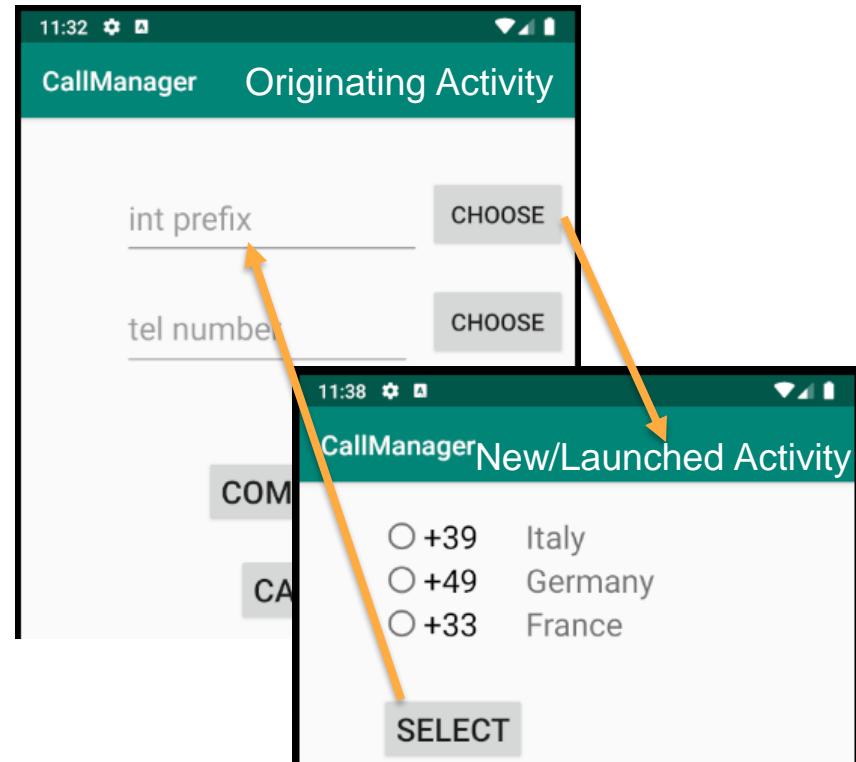
- the **originating Activity is paused**
- the **new Activity remains on the screen until**
 - the **user clicks the Back button**, or
 - the **finish() method is called**
 - in a click handler
 - or in other function that ends the user's involvement with this Activity

Returning data to the starting activity

Sometimes when starting an Activity with an Intent, **you would like to also get data back** from that Intent

E.g.: start an activity that **lets the user pick a prefix**

the **original Activity needs to receive information about the prefix** the user chose back from the launched/new Activity



Returning data to the starting activity

To launch a new Activity and get a result back, do the following steps:

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
3. Implement `onActivityResult()` in first Activity

1. startActivityForResult()

The **startActivityForResult()** method, like `startActivity()`, takes

- an **Intent argument** that contains
 - **information about the Activity to be launched**
 - any **data to send to that Activity (optional)**
- a **request code**

[startActivityForResult](#)(intent, requestCode);

Request code: an integer that identifies the request and can be used to differentiate between results when you process the return data.

For example, if you launch

- one Activity to take a photo and
- another to pick a photo from a gallery

you need **different request codes** to identify which request the returned data belongs to

1. startActivityForResult()

The **startActivityForResult()** method, like `startActivity()`, takes

- an **Intent argument** that contains
 - **information** about *the Activity to be launched*
 - any **data** to send to that Activity
- a **request code**

`startActivityForResult(intent, requestCode);`

Request code: an **integer** that **identifies the request** and can be used to **differentiate between results** when you process the return data.

For example, if you launch

- one Activity to take a photo and
- another to pick a photo from a gallery

you need **different request codes** to identify which request the returned data belongs to

1. startActivityForResult() Example

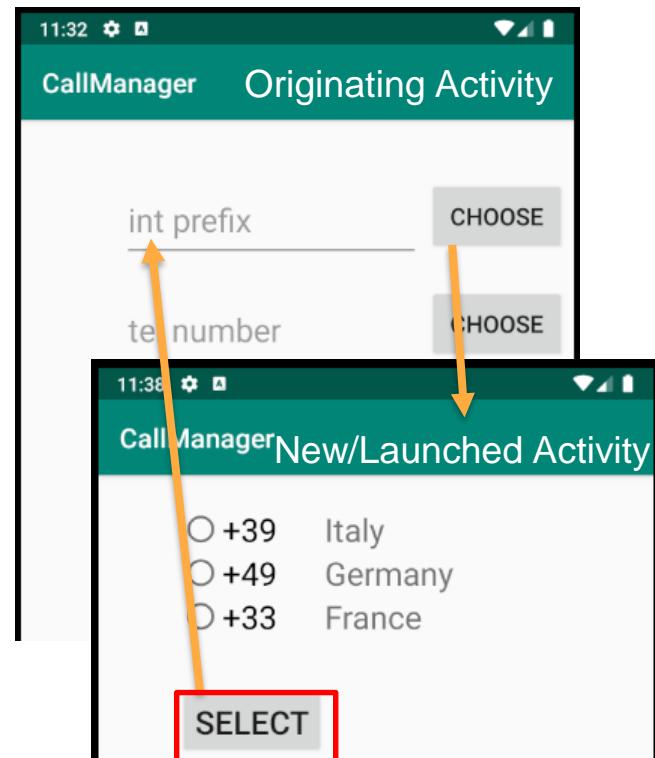
In the **activity that starts** the new one:

```
// define the identifier  
public static final int CHOOSE_FOOD_REQUEST = 1;  
  
// create the intent  
Intent intent = new Intent(this, ChooseFoodItemsActivity.class);  
  
// start the activity  
startActivityForResult(intent, CHOOSE_FOOD_REQUEST);
```

2. Return data and finish second activity

In the **new activity to return** info:

```
// 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();
```



2. Return data and finish second activity

In the **new activity to return** info:

```
// 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();
```

No target Activity class name

Android system directs the response back to the originating Activity automatically.



2. Return data and finish second activity

In the **new activity to return** info:

```
// 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();
```

```
public final static String  
EXTRA_REPLY = "com.example.  
myapp.RETURN_MESSAGE";
```

2. Return data and finish second activity

In the **new activity to return** info:

```
// 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();
```



- RESULT_OK: The request was successful
- RESULT_CANCELED: The user canceled the operation
- RESULT_FIRST_USER: For defining your own result codes

3. Implement onActivityResult()

In the **original activity** to receive the data:

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

3. Implement onActivityResult()

In the **original activity** to receive the data:

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

3. Implement onActivityResult()

In the **original activity** to receive the data:

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

3. Implement onActivityResult()

In t

```
public void onActivityResult(int requestCode, int resultCode, Intent data) {
```

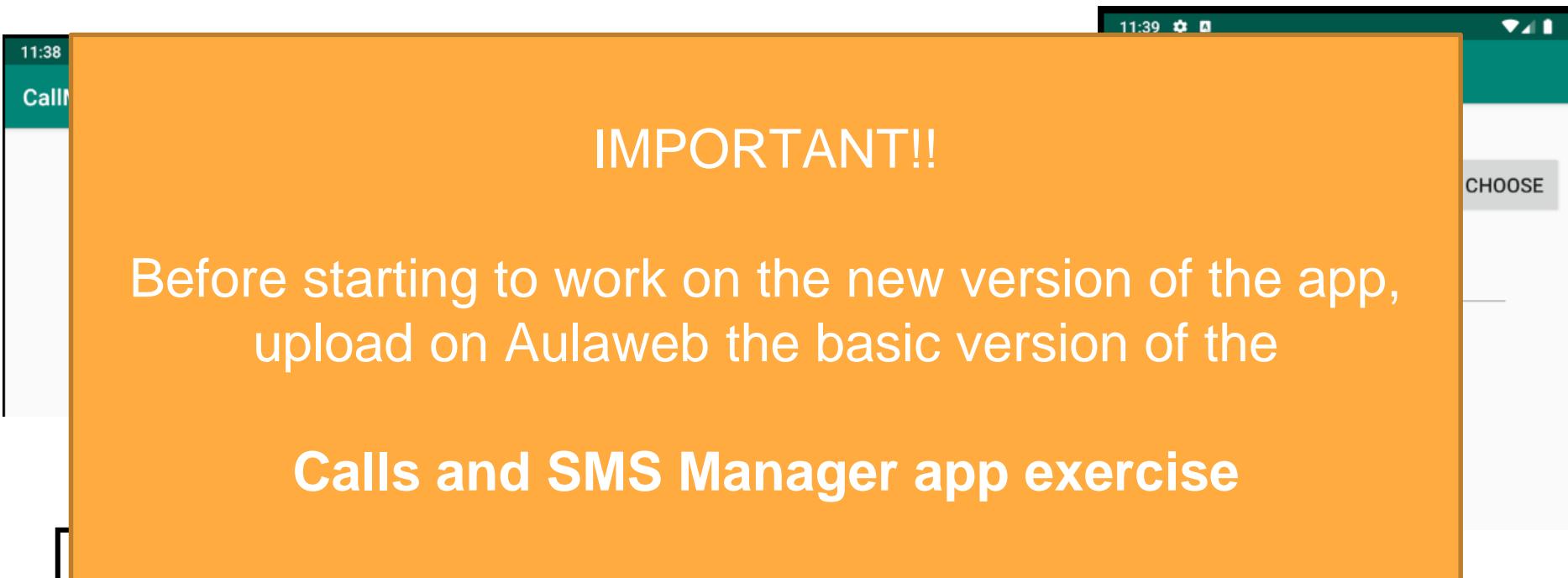
Recently, startActivityForResult() has been deprecated.

You can use it for the exercise but if are interested in the more recent alternative see here:

<https://stackoverflow.com/questions/62671106/onactivityresult-method-is-deprecated-what-is-the-alternative>

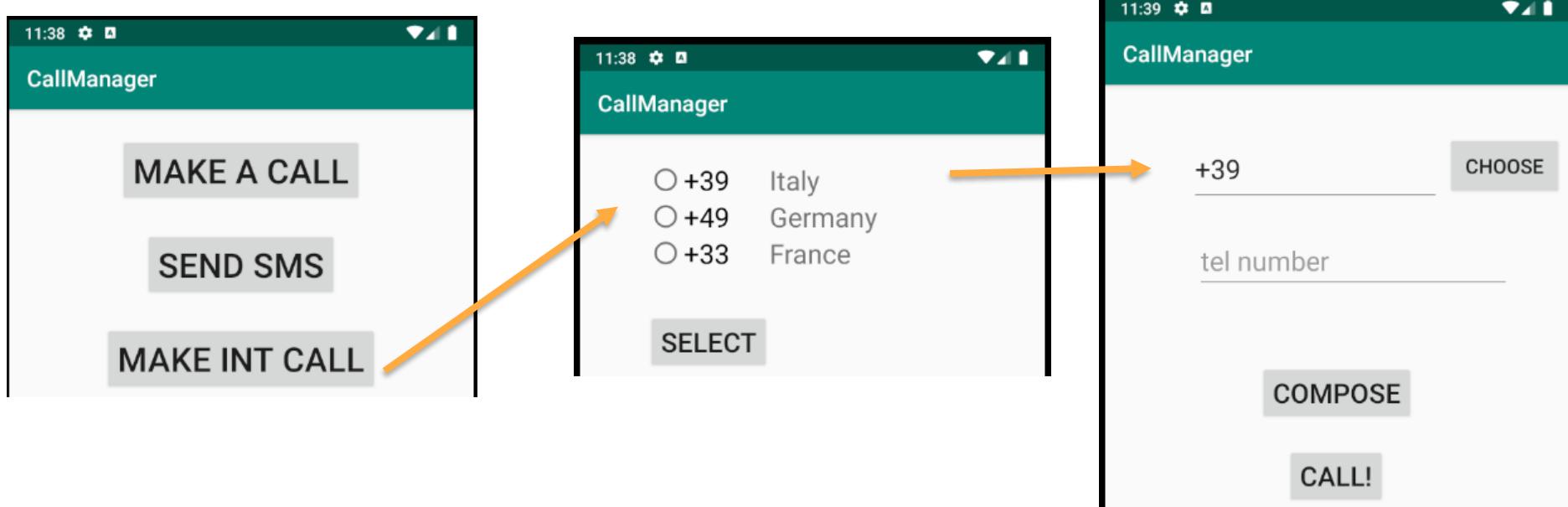
```
}}
```

Calls and SMS Manager Exercise



- First **select the prefix**
- Then **visualize the prefix in the composer**

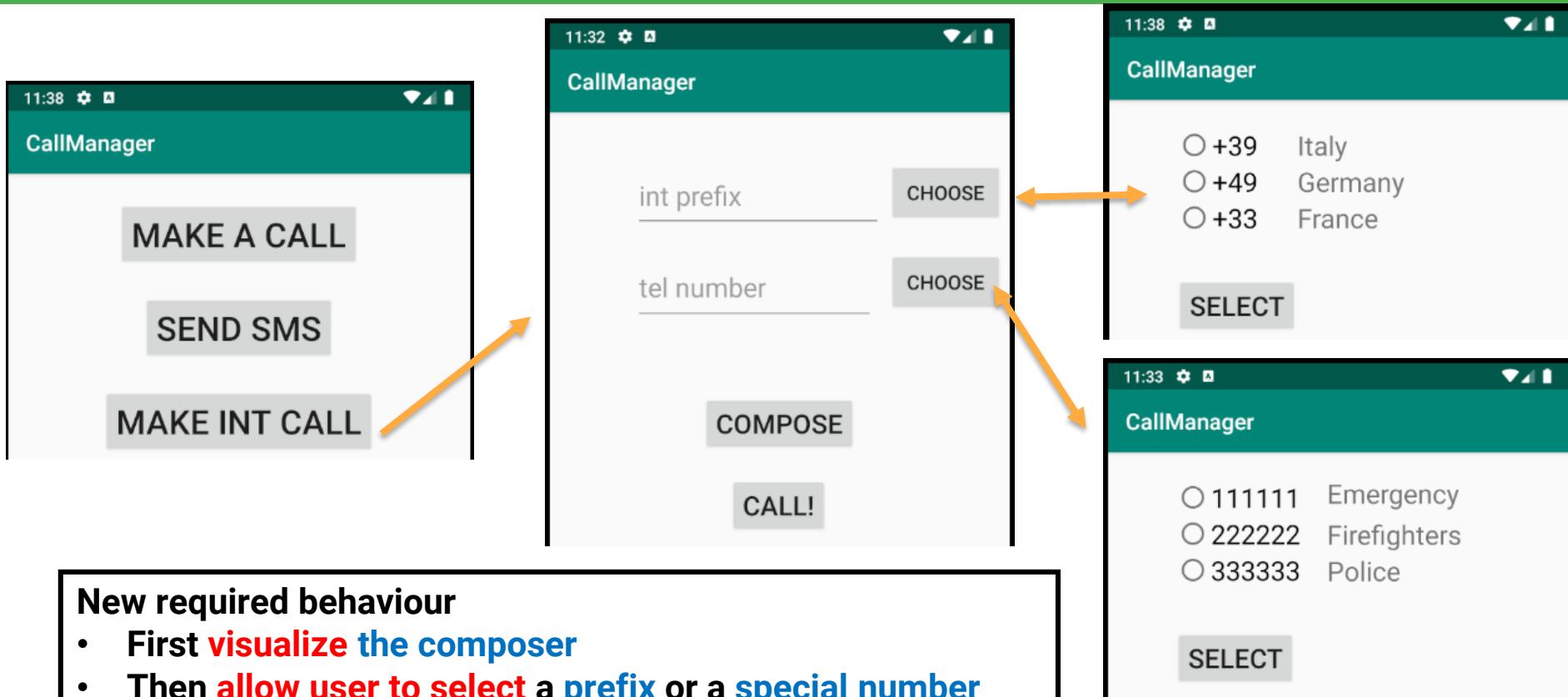
Calls and SMS Manager Exercise



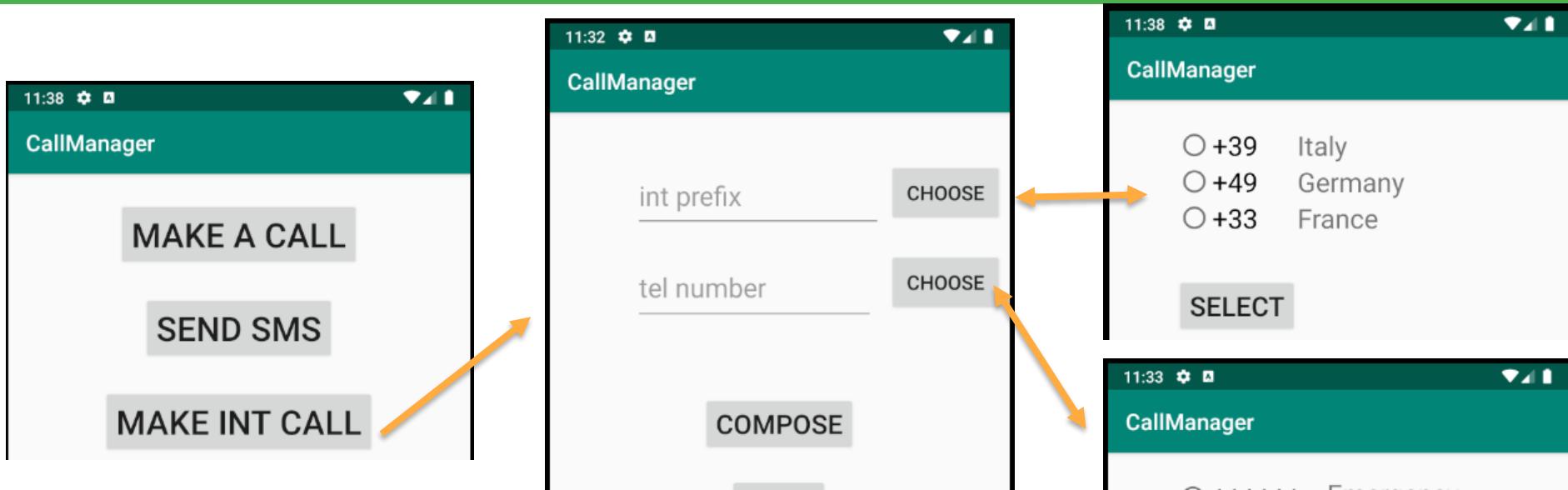
Original version of the app

- First **select the prefix**
- Then **visualize the prefix in the composer**

Calls and SMS Manager Exercise



Calls and SMS Manager Exercise



In the composer activity, use

Two: `startActivityForResult`

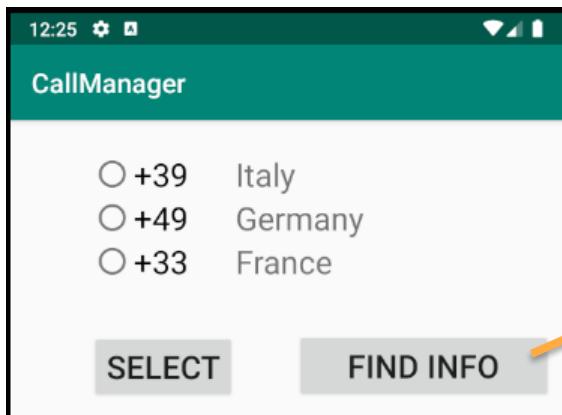
to open the two activities (prefix and numbers)

One: `onActivityResult`

to collect data from the two activities (use **request codes...**)

Calls and SMS Manager Exercise

Add a 'Find Info' button allowing to search for information about international prefixes on wikipedia:



The screenshot shows a web browser displaying the Wikipedia page for "List of international call prefixes". The page title is "List of international call prefixes". The content explains that an international call prefix or dial out code is a trunk prefix used to select an international telephone circuit for placing an international call. It is now called an IDD prefix (international direct dialing) – a country will typically have an NDD prefix as well (national direct dialing). The international dialing prefix must be dialed before the country calling code +39 and the destination telephone number. It is synonymous with international access code or exit code. The international call prefix is part of the telephone numbering plan of a country for calls to another country. The page also mentions the International Telecommunication Union (ITU).

https://en.wikipedia.org/wiki/List_of_international_call_prefixes

Calls and SMS Manager Exercise

Complete assignment:

- **Modify *Calls and SMS Manager* to implement the described behavior for**
 - international calls
 - Manage also the case in which the **user does not select a prefix or a number**
 - special numbers
 - find info on wikipedia

Navigation & Implicit Intents (Receiving an implicit Intent)

Navigation

These slides are partially based on the material that Google provides for the course
Android Developer Fundamentals

<https://developer.android.com/courses/fundamentals-training/overview-v2>

Navigation

The **majority of the apps** will include more than one **Activity**

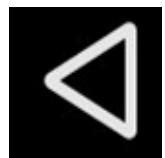
Since, few things **frustrate users** more than **basic navigation** that behaves in **inconsistent** and **unexpected** ways

Consistent navigation is an **essential component** of the **overall user experience**



Two forms of navigation

Android system supports two different forms of navigation
strategies for your app



Back (or temporal) navigation



Up (or ancestral) navigation

Back Navigation

allows to **return to the previous Activity** by **tapping** the device **back button**

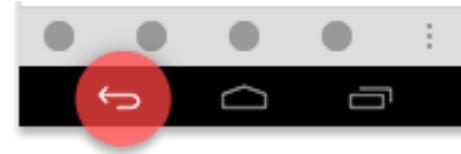
- **controlled by the Android system's back stack**
- **preserves history of recently viewed screens**



Back Navigation

allows to **return to the previous Activity** by **tapping** the device **back button**

- **controlled by the Android system's back stack**
- **preserves history of recently viewed screens**



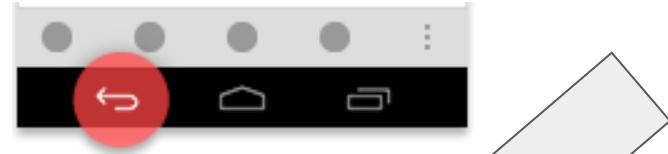
After viewing shopping cart, user decides to add more items, then places order.



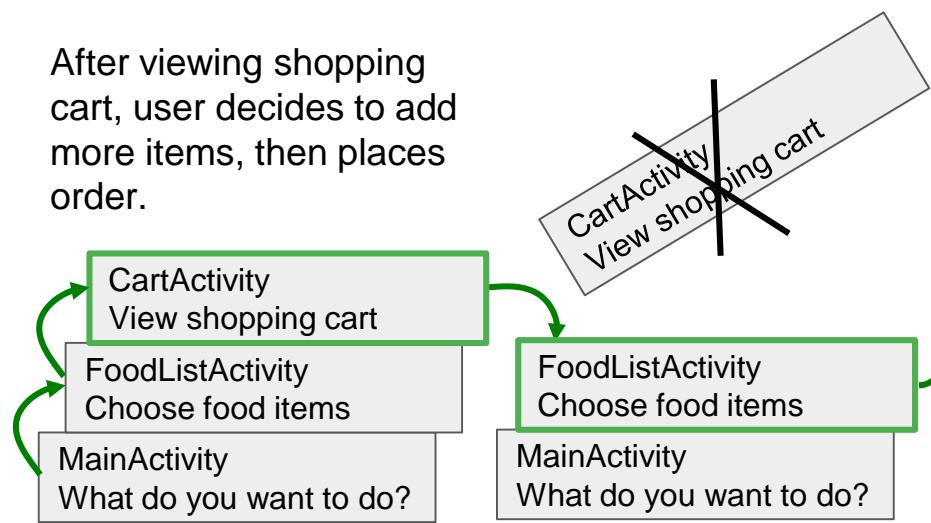
Back Navigation

allows to **return to the previous Activity** by **tapping** the device **back button**

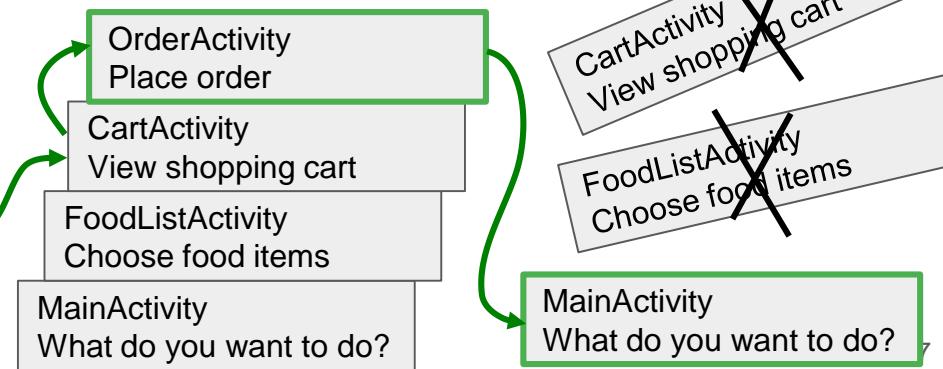
- **controlled by the Android system's back stack**
- **preserves history of recently viewed screens**



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



Activity Stack



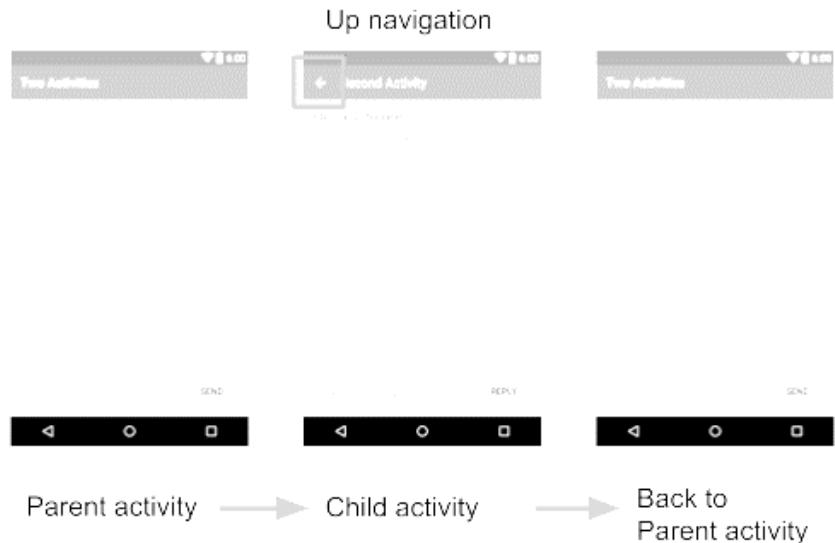
Up Navigation

is used to **navigate within an app** based on the **explicit hierarchical relationships** between screens
(sometimes referred to as **ancestral or logical navigation**)

- Navigate to parent of current Activity
- The Activity parent defined in Android manifest using `parentActivityName`

Example for `ChildActivity`

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



Up Navigation

is used to **navigate within an app** based on the **explicit hierarchical relationships** between screens
(sometimes referred to as **ancestral or logical navigation**)

- Navigate to parent of current Activity
- The Activity parent defined in Android manifest using `parentActivityName`

Example for `ChildActivity`

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



Up Navigation

is used to **navigate within an app** based on the **explicit hierarchical relationships** between screens
(sometimes referred to as **ancestral or logical navigation**)

- Navigate to parent of current Activity
- The Activity parent defined in Android manifest using `parentActivityName`

Example for ChildActivity

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



Up Navigation - Manifest

```
<application ... >
    ...
    <!-- The main/home activity (it has no parent activity) -->
    <activity
        android:name="com.example.myfirstapp.MainActivity" ...>
        ...
    </activity>
    <!-- A child of the main activity -->
    <activity
        android:name="com.example.myfirstapp.MyChildActivity"
        android:label="@string/title_activity_child"
        android:parentActivityName="com.example.myfirstapp.MainActivity" >
        <!-- Parent activity meta-data to support 4.0 and lower -->
        <meta-data
            android:name="android.support.PARENT_ACTIVITY"
            android:value="com.example.myfirstapp.MainActivity" />
    </activity>
</application>
```

Up Navigation - Manifest

```
<application ... >
```

If the Up navigation button is not visible try to add:

```
android:theme="@style/Theme.AppCompat.Light.DarkActionBar"
```

to the activitie(s), in AndroidManifest.xml file

For instance:

```
<activity  
    android:name=". ChildActivity"  
    android:exported="false"  
    android:theme="@style/Theme.AppCompat.Light.DarkActionBar"  
    android:parentActivityName=".MainActivity" />
```

```
</application>
```

Up Navigation - Manifest

```
<application ... >
```

An then add the following commands (in bold) in the onCreate method of the ChildActivity

```
public class ChildActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        ...  
        ActionBar actionBar = getSupportActionBar();  
        actionBar.setDisplayHomeAsUpEnabled(true);  
    }  
}
```

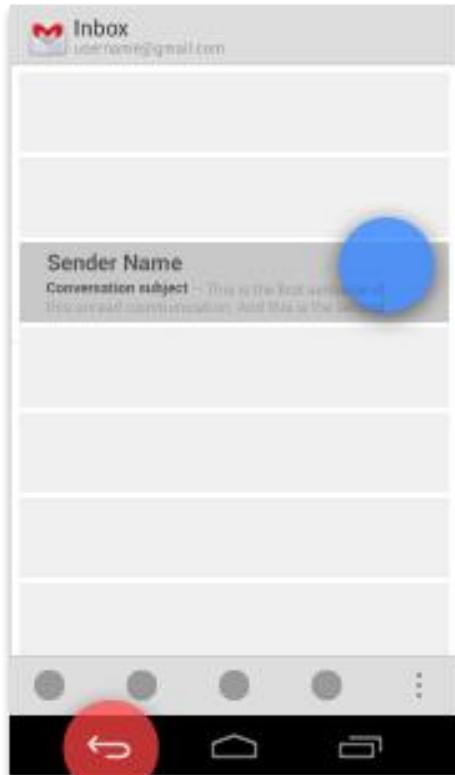
<https://stackoverflow.com/questions/60813166/android-studio-nullpointerexception-with-getsupportactionbar>

```
</application>
```

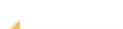
Back vs. Up Navigation



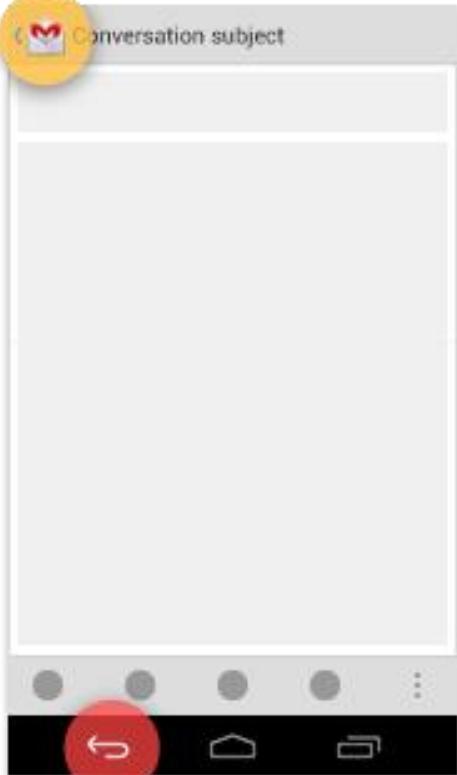
Home



Conversation list



UP



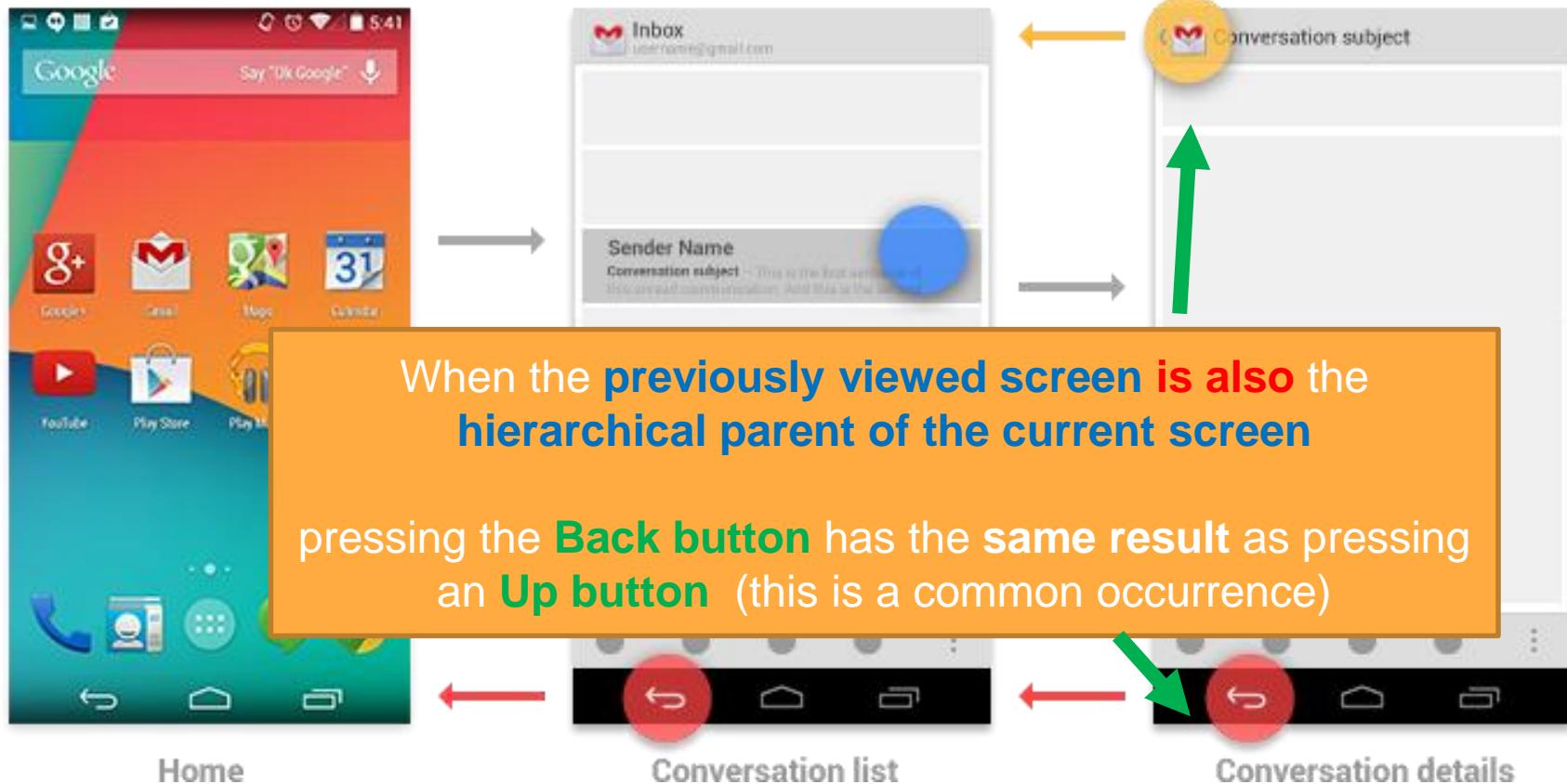
Conversation details



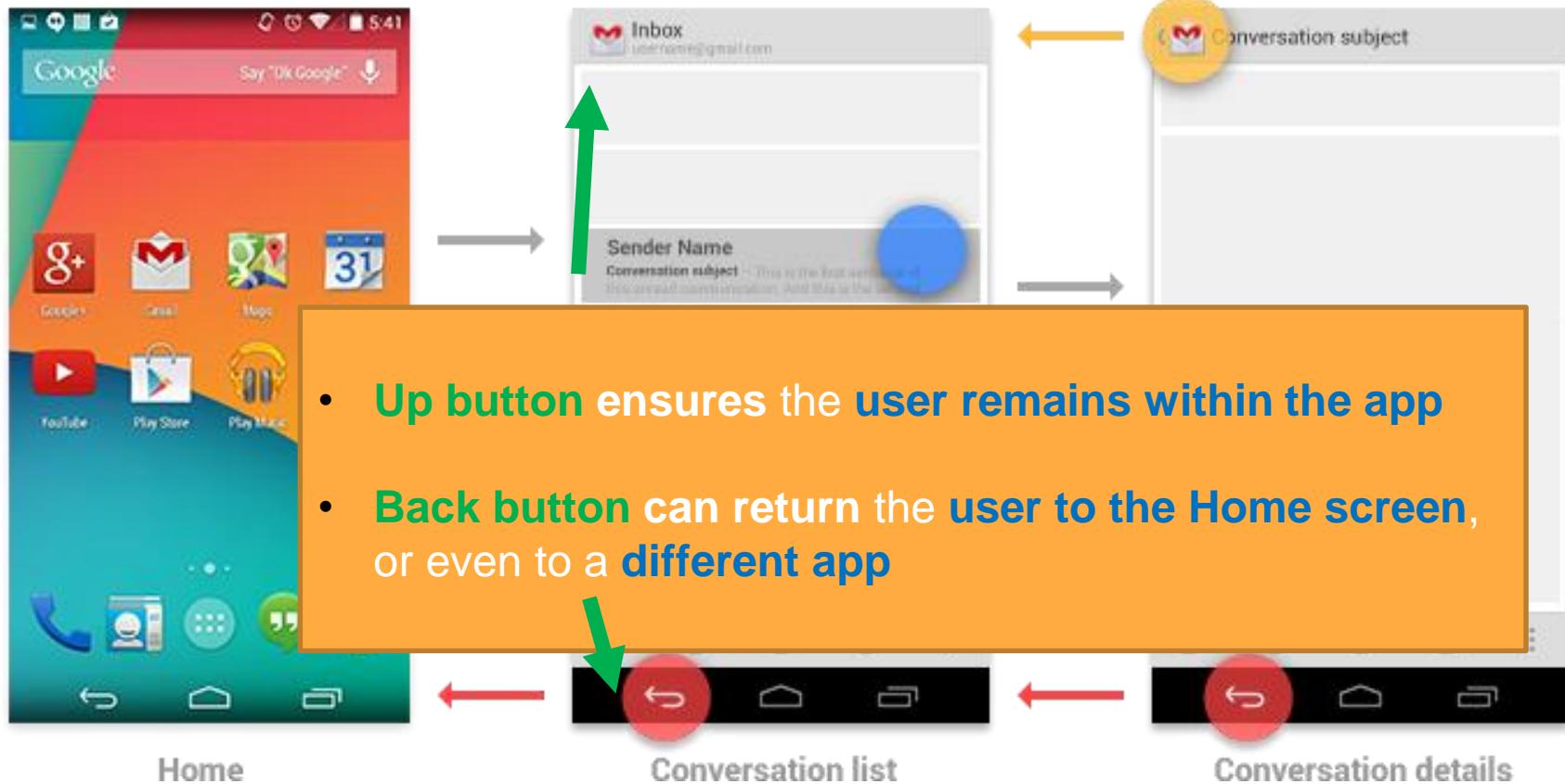
BACK



Back vs. Up Navigation



Back vs. Up Navigation



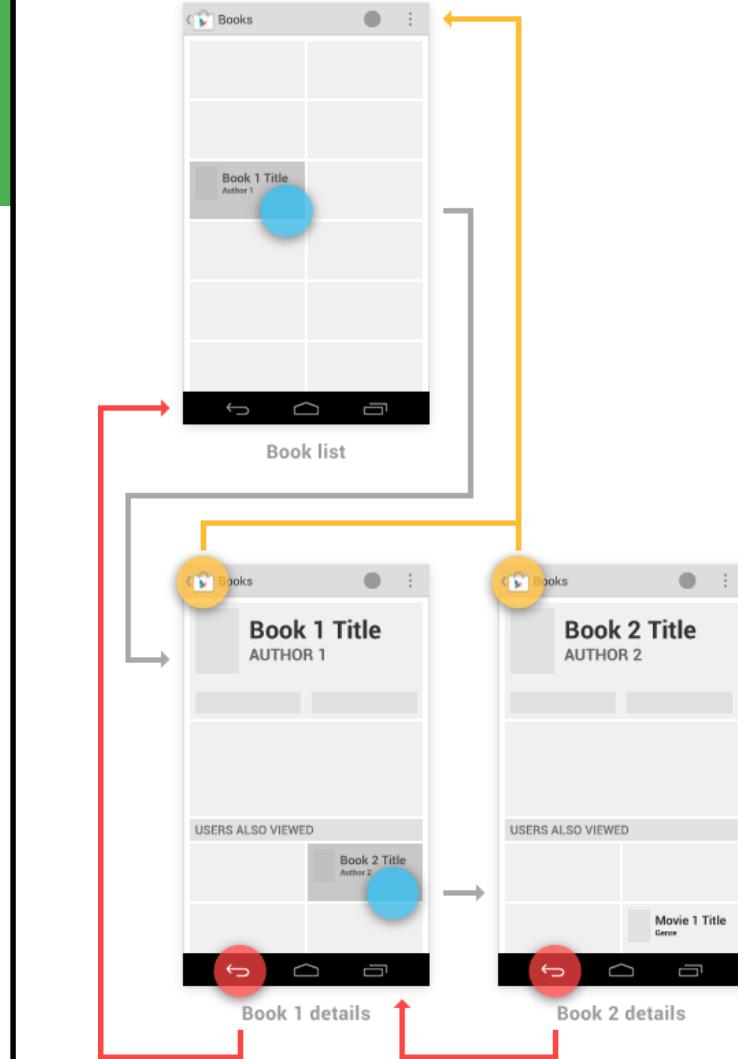
Back vs. Up Navigation

Second example:

1. Select a Book 1
2. View the details of Book 1
3. Select a Book 2
4. View the details of Book 2

Back a **Up** different behaviors

But reach the **same final activity**

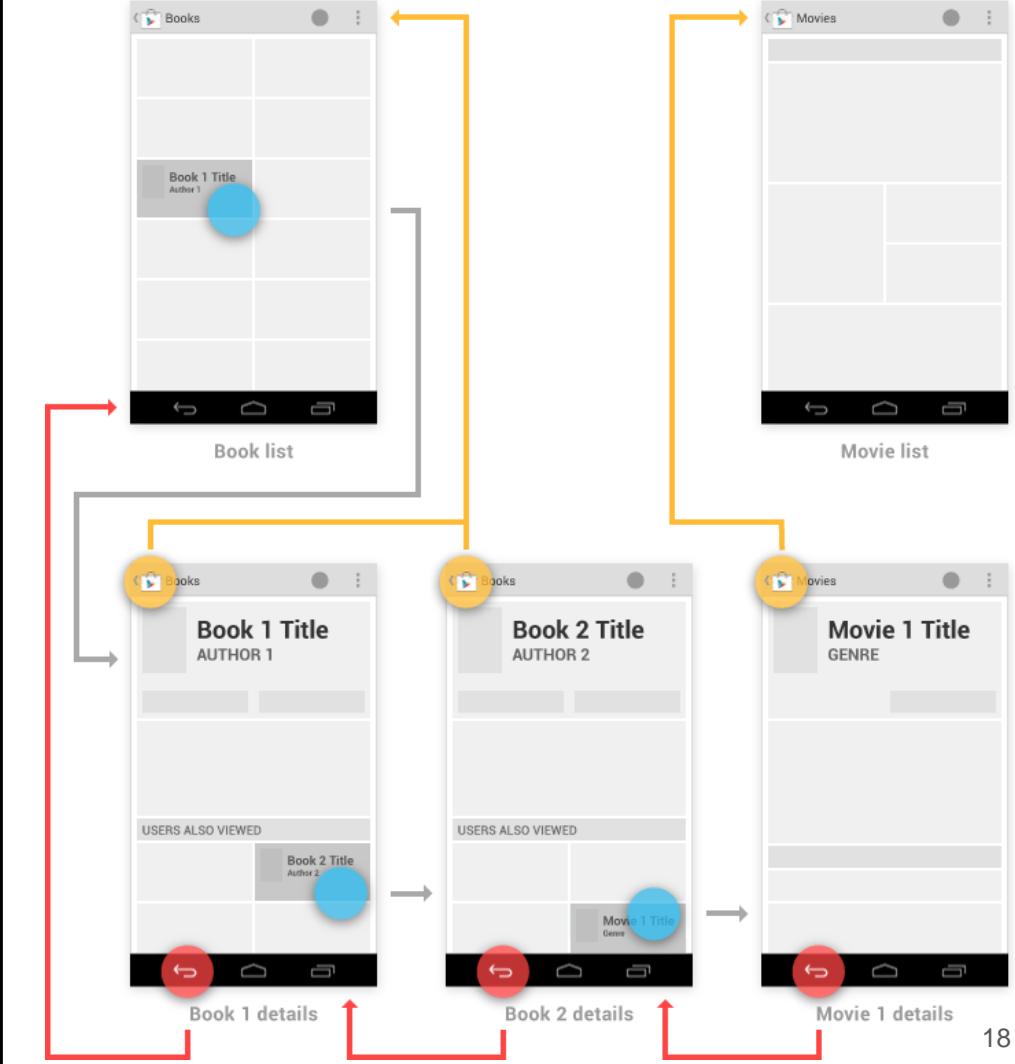


Back vs. Up Navigation

Advanced example:

1. Select a Book 1
2. View the details of Book 1
3. Select a Book 2
4. View the details of Book 2
5. Select a Movie

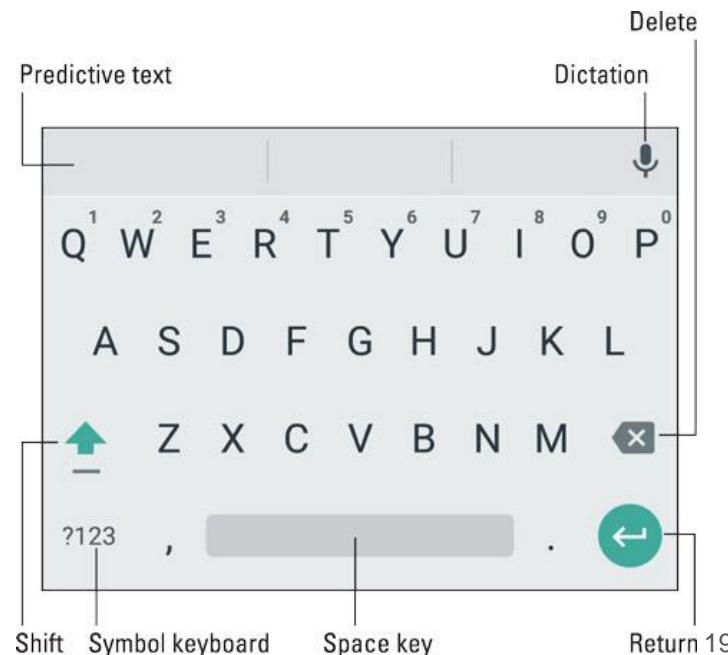
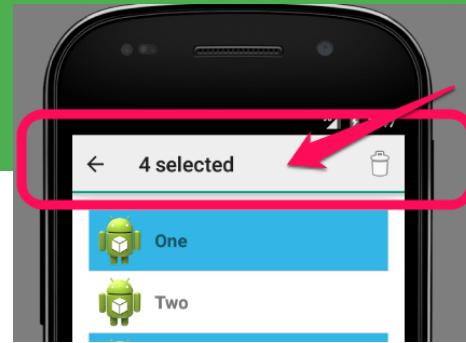
Back a **Up** different behaviors
and reach **different** final activities



Back Button

Back button also *supports* some different behaviors (not directly tied to screen-to-screen navigation):

- *Dismisses floating windows* (dialogs, popups)
- *Dismisses contextual action bars*, and *removes the highlight* from the selected items
- *Hides the onscreen keyboard* (IME)



Implicit Intents

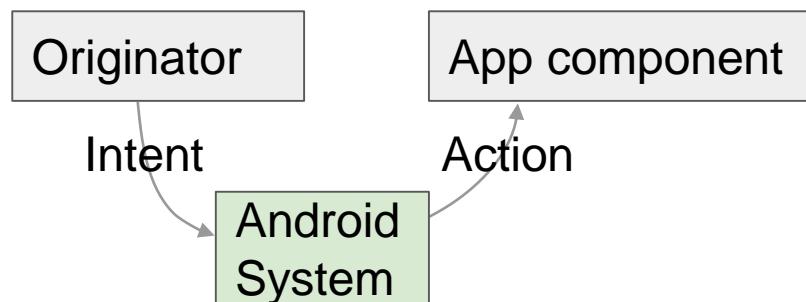
Additional Details and Receiving an implicit Intent

These slides are partially based on the material that Google provides for the course
Android Developer Fundamentals

What is an Intent?

An Intent is:

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



Explicit vs. implicit Intent

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

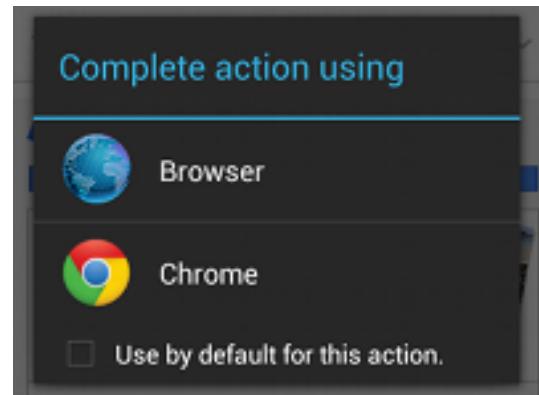
Implicit Intent – Asks **system** to find an **Activity class** with a
registered handler that can handle this request

Implicit Intent

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

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

Show an app chooser

Notice that when you start an activity by passing your `Intent` to `startActivity()` and there is more than one app that responds to the intent, the user can select which app to use by default (by selecting a checkbox at the bottom of the dialog; see figure 1). This is nice when performing an action for which the user generally wants to use the same app every time, such as when opening a web page (users likely use just one web browser) or taking a photo (users likely prefer one camera).

However, if the action to be performed could be handled by multiple apps and the user might prefer a different app each time—such as a "share" action, for which users might have several apps through which they might share an item—you should explicitly show a chooser dialog as shown in figure 2. The chooser dialog forces the user to select which app to use for the action every time (the user cannot select a default app for the action).

To show the chooser, create an `Intent` using `createChooser()` and pass it to `startActivity()`. For example:

See the code example here:

<https://developer.android.com/training/basics/intents/sending>

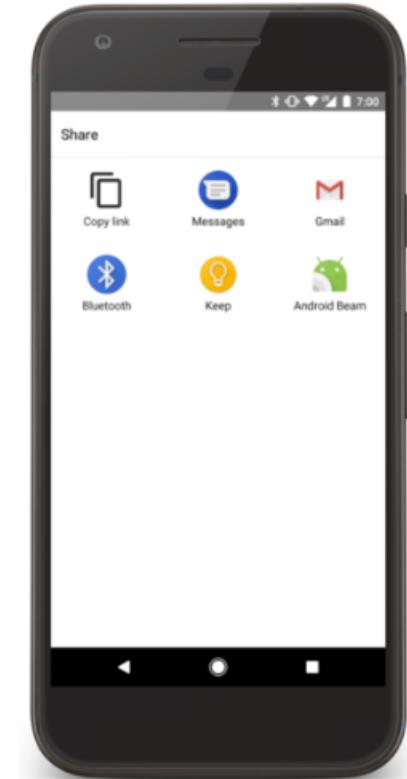


Figure 2. A chooser dialog.

How does implicit Intent work?

- The Android Runtime keeps a **list of registered Apps**
- Apps are register via `AndroidManifest.xml`
- Runtime **receives the request** and **looks for matches**
uses Intent filters for matching from `AndroidManifest.xml`
- If **more than one match**
shows a list of possible matches and lets the user choose one
- Android runtime **starts** the requested (selected) activity

Implicit Intent examples (Invocation)

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);
```

Receiving an Implicit Intent

Register your app to receive an Intent

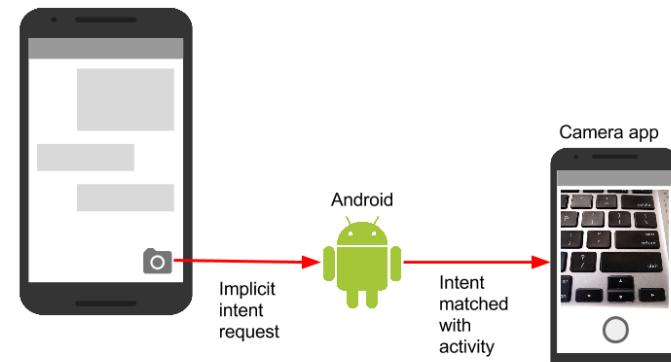
If an **Activity** in your app **have to respond** to an **implicit Intent** (from your own app or other apps)



declare one or more **Intent filters** in the **AndroidManifest.xml** file

Each **Intent filter** specifies **the type of Intent** it accepts based on the **action**, **data**, and **category** for the Intent

The system will deliver an **implicit Intent** to your app component **only if** that Intent can pass through one of your Intent filters



Intent action, data, and category

- **Action** is the generic *action the receiving Activity should perform*. The available Intent actions are defined as constants in the Intent class and begin with the word ACTION_.
- **Category** (optional) provides additional *information about the category of component that should handle the Intent*. Intent categories are also defined as constants in the Intent class and begin with the word CATEGORY_.
- **Data type** indicates the *MIME type of data the Activity should operate on*. Usually, the data type is inferred from the URI in the Intent data field, but you can also explicitly define the data type with the setType() method.

Intent action, data, and category

- **Action** is the generic *action the receiving Activity should perform*. The available Intent actions are defined as constants in the Intent class and begin with the word ACTION_.
- **Category** (optional) provides additional *information about the category of component that should handle the Intent*. Intent categories are also defined as constants in the Intent class and begin with the word CATEGORY_.
- **Data type** indicates the *MIME type of data the Activity should operate on*. Usually, the data type is inferred from the URI in the Intent data field, but you can also explicitly define the data type with the setType() method.

Intent action, data, and category

- **Action** is the generic *action the receiving Activity should perform*. The available Intent actions are defined as constants in the Intent class and begin with the word ACTION_.
- **Category** (optional) provides additional *information about the category of component that should handle the Intent*. Intent categories are also defined as constants in the Intent class and begin with the word CATEGORY_.
- **Data type** indicates the *MIME type of data the Activity should operate on*. Usually, the data type is inferred from the URI in the Intent data field, but you can also explicitly define the data type with the setType() method.

Intent action, data, and category

Intent **actions**, **categories**, and **data** types are used both by

- the ***Intent object you create*** in the sending Activity
- as well as, in the ***Intent filters you define*** in the AndroidManifest.xml file for the receiving Activity

The Android system uses this information to match an ***implicit Intent request with an Activity or other component that can handle that Intent***

Intent filter in AndroidManifest.xml

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

Intent filters: action and category

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

Intent filters: data

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

An Activity can have multiple filters

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

An Activity can have several filters

A filter can have multiple actions & data

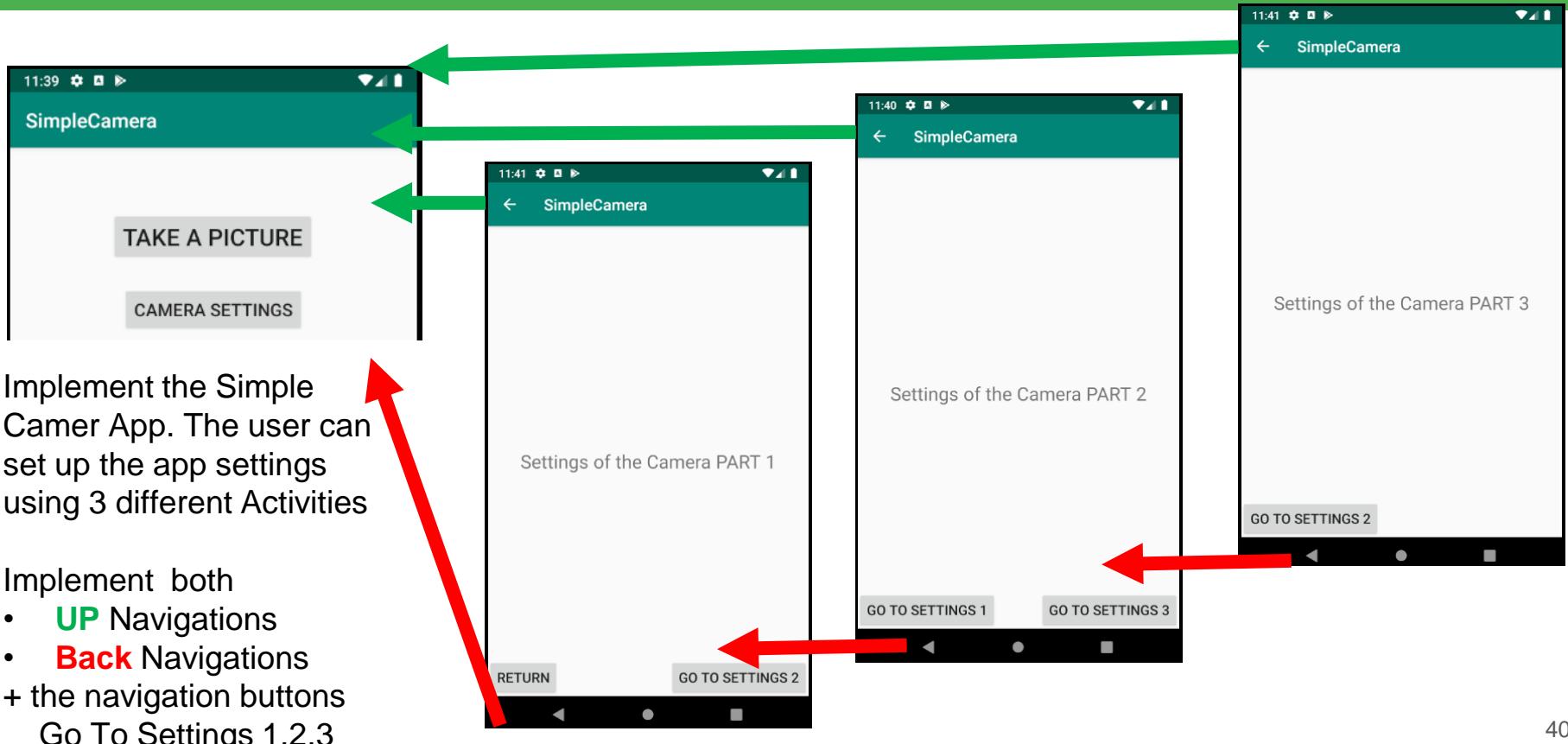
```
<intent-filter>  
    <action android:name="android.intent.action.SEND"/>  
    <action android:name="android.intent.action.SEND_MULTIPLE"/>  
    <category android:name="android.intent.category.DEFAULT"/>  
    <data android:mimeType="image/*"/>  
    <data android:mimeType="video/*"/>  
</intent-filter>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

Receiving an implicit Intent

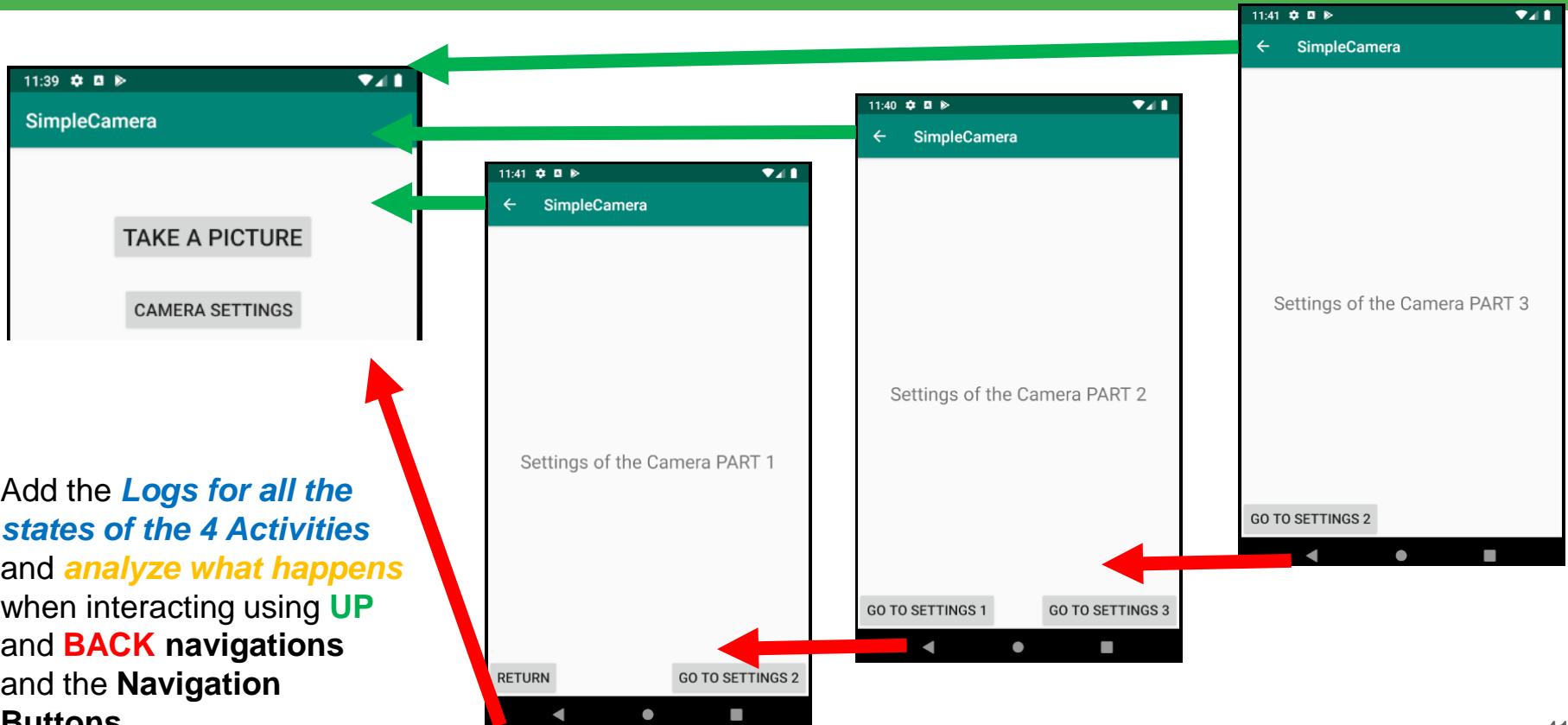
Once the **Activity is successfully launched** with an **implicit Intent**, from that Activity it is possible to **handle the Intent and its data** in the same way you did for an explicit Intent, by:

1. Getting the Intent object with `getIntent()`
2. Getting Intent data or extras out of that Intent
3. Performing the task the Intent requested
4. Returning data to the calling Activity with another Intent, if needed.

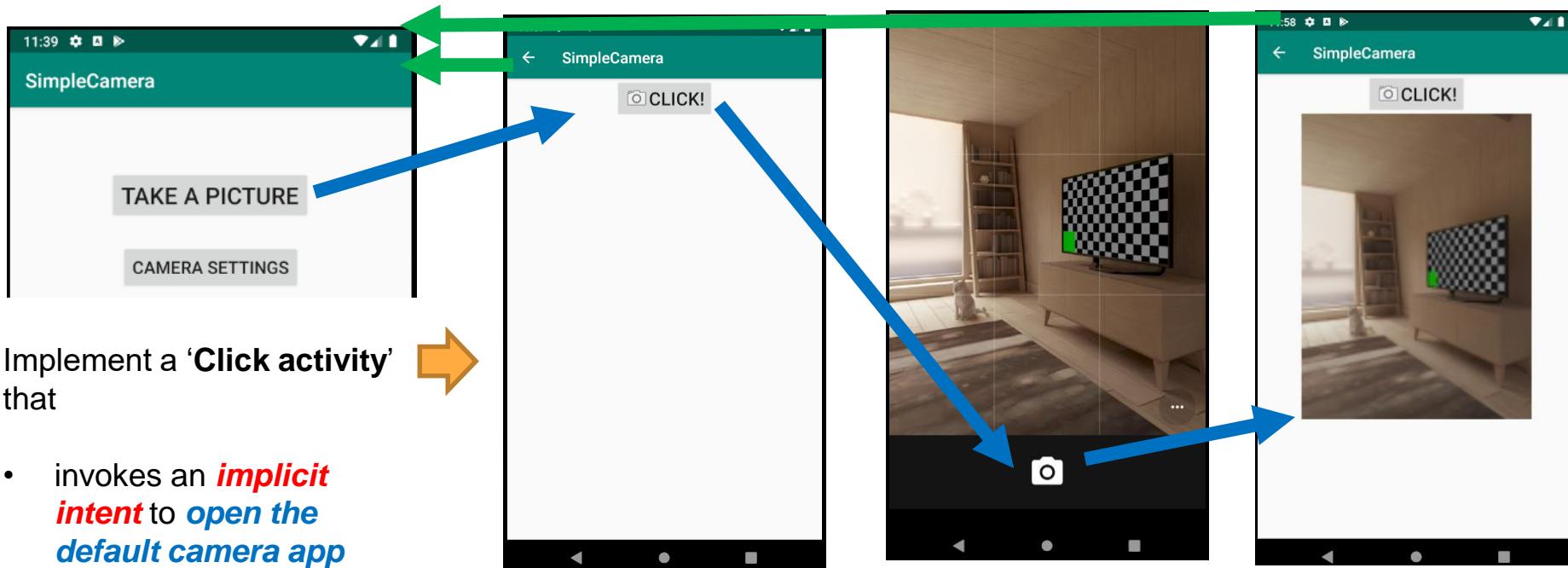
Simple Camera App (1)



Simple Camera App (2)



Simple Camera App (3)



Implement a 'Click activity' that

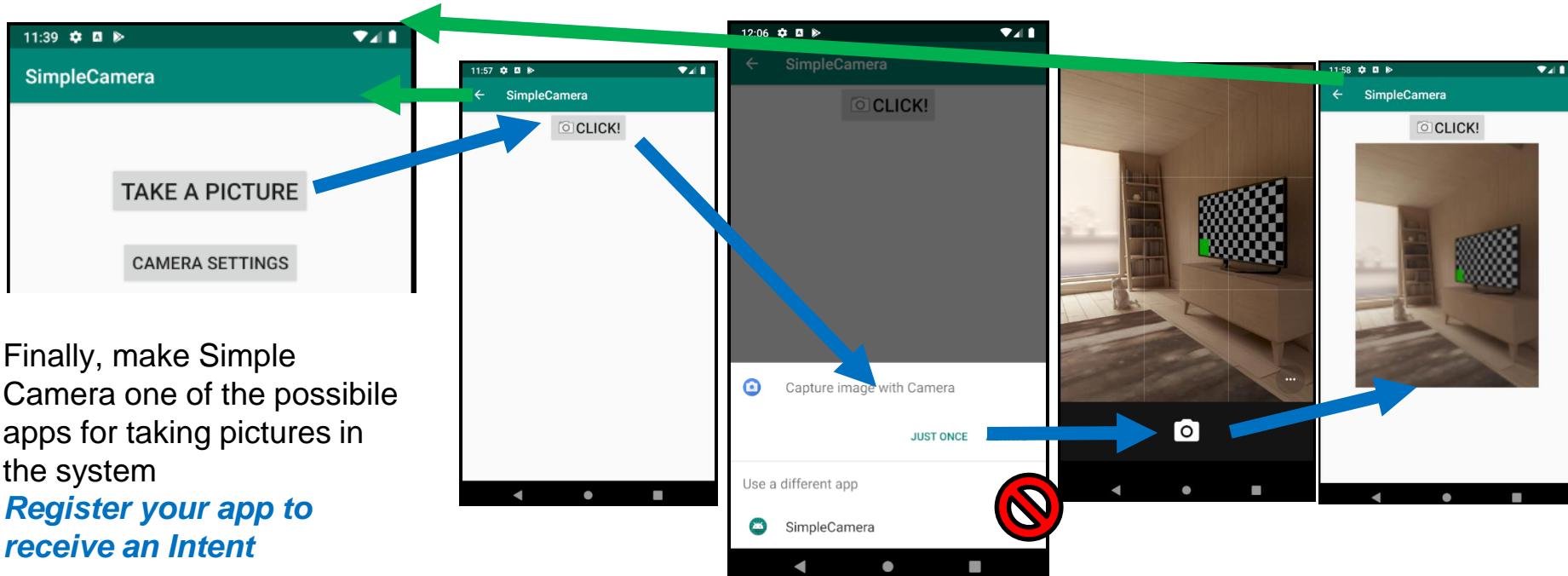
- invokes an *implicit intent* to open the *default camera app*
- and then *visualizes a thumbnail of the picture*

Add also the **UP navigations** as shown in the figures

<https://developer.android.com/media/camera/camera-deprecated/photobasics>

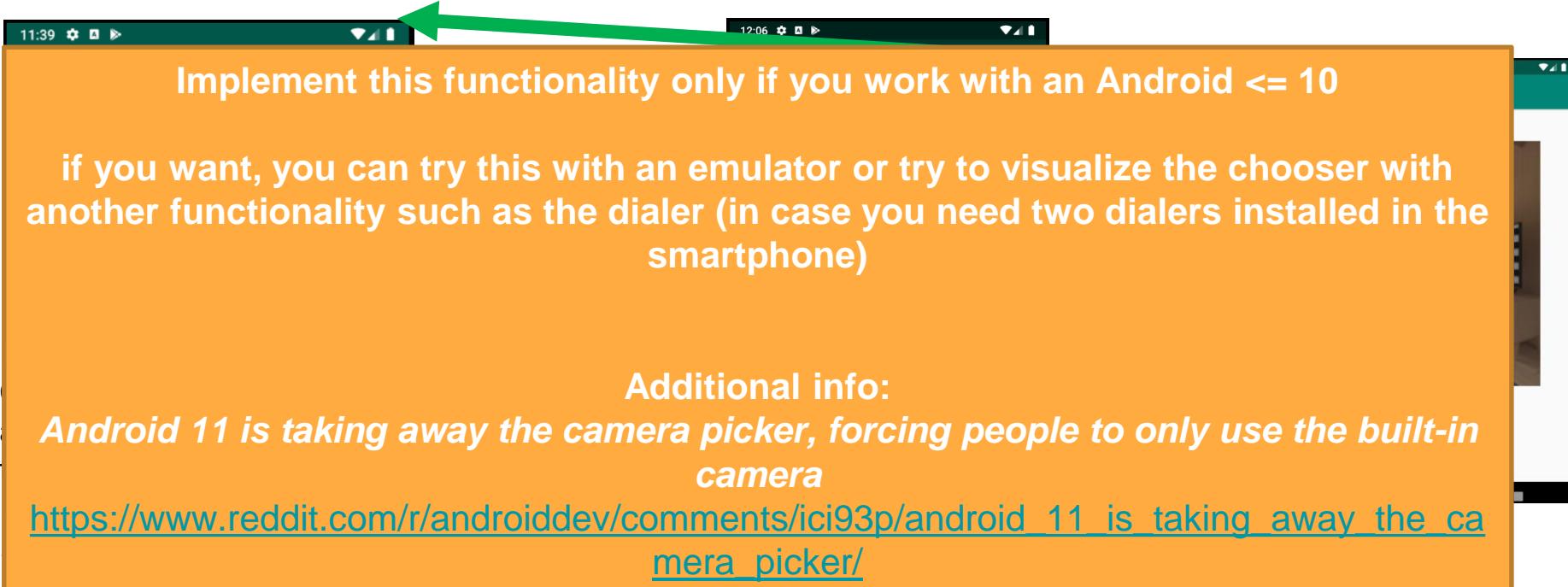
<https://developer.android.com/media/camera/choose-camera-library>

Simple Camera App (4)



Modify the `AndroidManifest.xml`.... **NOTE:** this will allow to select also Simple Camera app and does not make any sense since the intent starts from the same app... Simple Camera is not able to take 'directly' a picture but relies on the default app... We try this only for making an exercise... Thus in the 'Chooser' select always the smartphone or emulator default camera app...

Simple Camera App (4)



A screenshot of an Android emulator interface. At the top, there's a green header bar with the title 'Simple Camera App (4)'. Below it is a white action bar with icons for back, home, and recent apps. The main content area is orange and contains text. A green arrow points from the text 'Implement this functionality only if you work with an Android <= 10' towards the top left of the screenshot.

Implement this functionality only if you work with an Android <= 10

if you want, you can try this with an emulator or try to visualize the chooser with another functionality such as the dialer (in case you need two dialers installed in the smartphone)

Additional info:
Android 11 is taking away the camera picker, forcing people to only use the built-in camera

https://www.reddit.com/r/androiddev/comments/ici93p/android_11_is_taking_away_the_camera_picker/

Modify the `AndroidManifest.xml`.... **NOTE:** this will allow to select also Simple Camera app and does not make any sense since the intent starts from the same app... Simple Camera is not able to take 'directly' a picture but relies on the default app... We try this only for making an exercise... Thus in the 'Chooser' select always the smartphone or emulator default camera app...



User Interaction, Buttons and Clickable Images

Users interacting with apps

In an Android app, **user interaction** typically involves:

tapping (clicking), typing, using gestures, but also talking

The Android framework **provides** corresponding user interface (**UI**) elements such as:

buttons, clickable images, menus, keyboards, text entry fields, and a microphone

Users expectations during interaction

Android users expect *UI elements to act in certain ways*



so it's important that your app is consistent with other
Android apps

To *satisfy the users*, it is important to create a UI that
provides predictable choices / behaviors

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** (see **UI components docs**)
- Meet user's expectations

Material design is a comprehensive guide for visual, motion, and interaction design across platforms and devices

<https://developer.android.com/guide/topics/ui/look-and-feel>

<https://material.io/>

Buttons

Button

Are Views that **respond to tapping** (clicking) or **pressing**

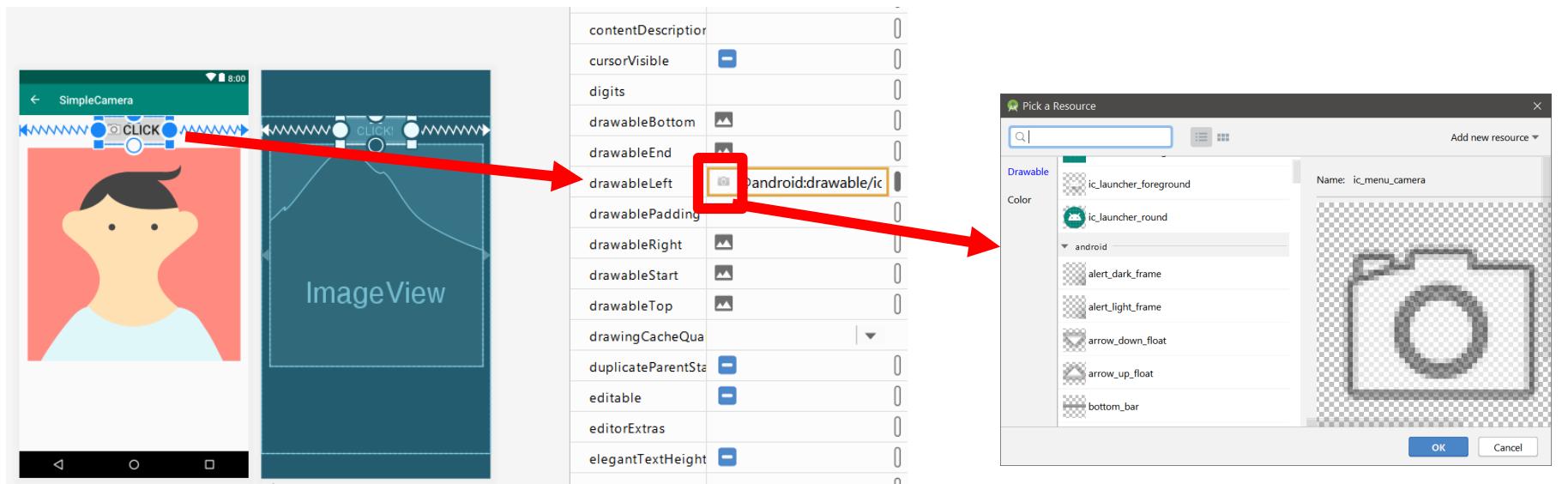
- Usually **text** or **visuals** (e.g., an icon) **indicate what will happen when tapped/pressed**
- **Buttons** can have the **following design**:
 - **Text only**, as shown on the left side of the figure below
 - **Icon only**, as shown in the center of the figure below
 - **Both text and an icon**, as shown on the right side of the figure below



Button image

To **add an icon** to a **button**

- **search for the attribute of interest** in the attributes list (e.g., drawableLeft)
- **click ‘pick a resource’** and **select the icon** (e.g., the camera icon)



Button image

The following **attributes** **manage how an icon is visualized** in a button.

android:drawableBottom

The drawable to be drawn below the text.

android:drawableEnd

The drawable to be drawn to the end of the text.

android:drawableLeft

The drawable to be drawn to the left of the text.

android:drawablePadding

The padding between the drawables and the text.

android:drawableRight

The drawable to be drawn to the right of the text.

android:drawableStart

The drawable to be drawn to the start of the text.

android:drawableTint

Tint to apply to the compound (left, top, etc.) drawables.

android:drawableTintMode

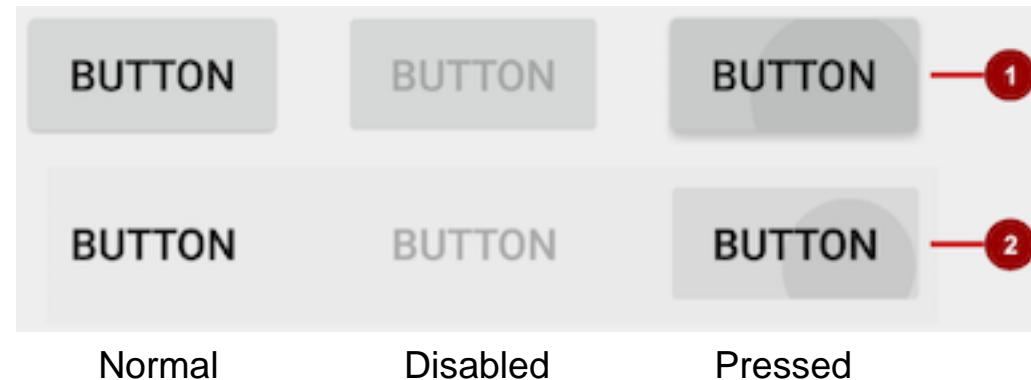
Blending mode used to apply the compound (left, top, etc.) drawables tint.

android:drawableTop

The drawable to be drawn above the text.

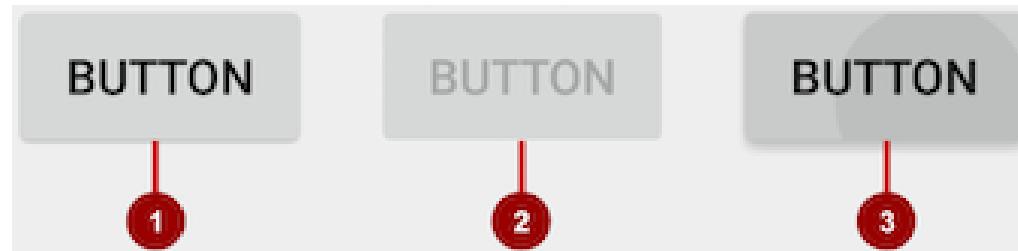
Raised and Flat Buttons

- Android offers **several types of Button** elements, including
 - raised buttons (1)
 - flat buttons (2)
- Each button has three states:
 - Normal
 - Disabled
 - Pressed



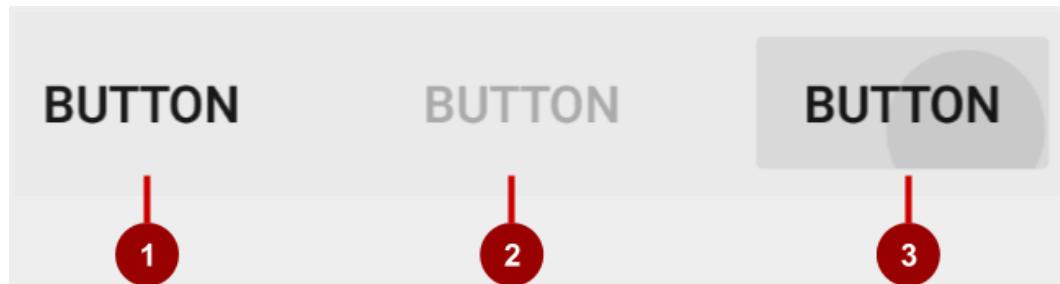
Raised and Flat Buttons

A **raised button** is an **outlined rectangle or rounded rectangle** that **appears lifted from the screen**—the shading around it indicates that it is possible to tap or click it. The raised button can show text, an icon, or both (default style)



Raised and Flat Buttons

A **flat button**, also known as a text button or borderless button, is a text-only button that **looks flat** and **doesn't have a shadow**. The major benefit of flat buttons is simplicity: a flat button doesn't distract the user from the main content as much as a raised button does



Flat buttons

Flat buttons are useful for dialogs that require user interaction.

In this case, the button uses ***the same font and style as the surrounding text*** to keep the ***look and feel consistent*** across all the elements in the dialog.

To **create a flat button** add the ***following attribute*** to your button:

```
style="?android:attr/borderlessButtonStyle"
```

Use Google's location service?

Let Google help apps determine location. This means sending anonymous location data to Google, even when no apps are running.

DISAGREE AGREE

Responding to button taps

In XML: Android Studio provides a shortcut for setting up an OnClickListener for the clickable object in your Activity code, and for assigning a callback method: use the android:onClick attribute within the clickable object's element in the XML layout.

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" />
```



and then define the method (e.g., sendMessage) in the code

Setting listener with onClick callback

In your code: use OnClickListener event listener

```
Button buttonSend = findViewById(R.id.button_send);

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

`View.OnClickListener` is an interface, you don't call it, but creates a new instance of it (`new View.OnClickListener()` is a call to the constructor)

24

The instance you create is of `anonymous class` that implements `View.OnClickListener`, in the brackets right under `new View.OnClickListener()`

✓

Any class that implements `View.OnClickListener` must implement the methods declared in it (e.g. `onClick`)

`setOnClickListener` just saves the reference to the `View.OnClickListener` instance you supplied, and when someone clicks the button, the `onClick` method of the listener you set is getting called.

share improve this answer

edited Aug 4 '11 at 19:02

answered Aug 4 '11 at 18:57



MByD

121k • 23 • 240 • 254

14 small addition: don't be confused by the `.` in `View.OnClickListener`. It hasn't do anything with a method call, the `OnClickListener` interface is just defined inside the class `View` - therefore you have to access the Interface via the class: `View.OnClickListener`. (You could also `import android.view.View.OnClickListener` and access the interface via `OnClickListener` but I don't know if that is against some android coding conventions) – pmnt Aug 4 '11 at 19:18

<https://stackoverflow.com/questions/6946971/view-onclicklistener-method-or-class/6947033#6947033>

<https://developer.android.com/reference/android/view/View.OnClickListener.html>

How and Where to define Listeners

all the Listeners must be initialized as soon as possible, before user gets to interact with the Activity.

Case 1:

Defining the listeners in the `onCreate()` method

Simple approach

Clutters `onCreate` when having a lot of Listeners

The diagram features three orange arrows. One arrow points from the text 'Defining the listeners in the onCreate() method' to the line of code 'awesomeButton.setOnClickListener(new View.OnClickListener() {'. Another arrow points from the text 'Simple approach' to the line 'private void awesomeButtonClicked() {'. A third arrow points from the text 'Clutters onCreate when having a lot of Listeners' to the line 'super.onCreate(savedInstanceState);'.

```
1 public class AwesomeButtonActivity extends AppCompatActivity {
2
3     private Button awesomeButton;
4
5     @Override
6     protected void onCreate(@Nullable Bundle savedInstanceState) {
7         super.onCreate(savedInstanceState);
8         awesomeButton = findViewById(R.id.awesomeBtn);
9
10        awesomeButton.setOnClickListener(new View.OnClickListener() {
11            @Override
12            public void onClick(View v) {
13                awesomeButtonClicked();
14            }
15        });
16    }
17
18    private void awesomeButtonClicked() {
19        awesomeButton.setText("AWESOME!");
20    }
21 }
```

How and Where to define Listeners

Case 2:

Similar to Case 1 except we assign the implementation to a field in the class.

Easy to refactor from Case 1

Promote reusability of the code

Several other ways available, see for instance:

<https://medium.com/@CodyEngel/4-ways-to-implement-onclicklistener-on-android-9b956cbd2928>



```
1 public class AwesomeButtonActivity extends AppCompatActivity {  
2  
3     private Button awesomeButton;  
4  
5     private View.OnClickListener awesomeOnClickListener = new View.OnClickListener() {  
6         @Override  
7         public void onClick(View v) {  
8             awesomeButtonClicked();  
9         }  
10    };  
11  
12    @Override  
13    protected void onCreate(@Nullable Bundle savedInstanceState) {  
14        super.onCreate(savedInstanceState);  
15        awesomeButton = findViewById(R.id.awesomeBtn);  
16        awesomeButton.setOnClickListener(awesomeOnClickListener);  
17    }  
18  
19  
20    private void awesomeButtonClicked() {  
21        awesomeButton.setText("AWESOME!");  
22    }  
23}
```

Responding to button LONG taps

Similarly the buttons can respond to other events. For instance to long click

```
Button button = findViewById(R.id.button);
button.setOnLongClickListener(new View.OnLongClickListener() {
    @Override
    public boolean onLongClick(View v) {
        // Do something in response to button click

        //return true if the callback consumed the long click, false otherwise.
        //return boolean true at the end of OnLongClickListener to indicate
        //you don't want further processing
        return true;
        //return false;
    }
}); https://stackoverflow.com/questions/5428077/android-why-does-long-click-also-trigger-a-normal-click
```

Responding to ImageView taps

Similarly it is possible to associate events to imageViews:

In XML: use android:onClick attribute in the XML layout:

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

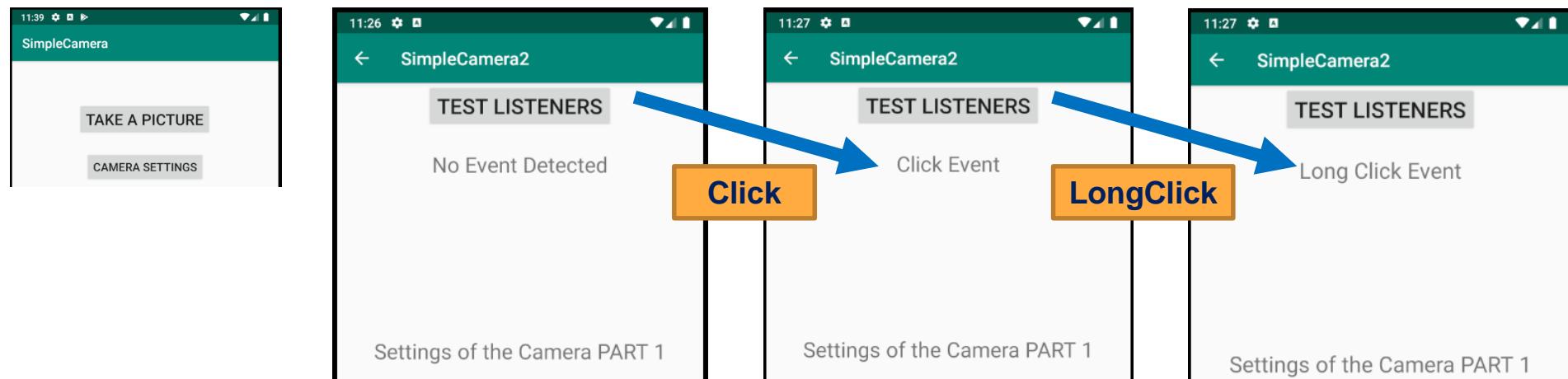
android:onClick



In your code: defining listeners for clicks, longClicks etc, in the code as seen for Buttons

Simple Camera App (Warmup...)

Add a **Button** in the first ‘Camera Setting (Part 1)’ activity and a **Label** describing the *kind of event fired* on such button



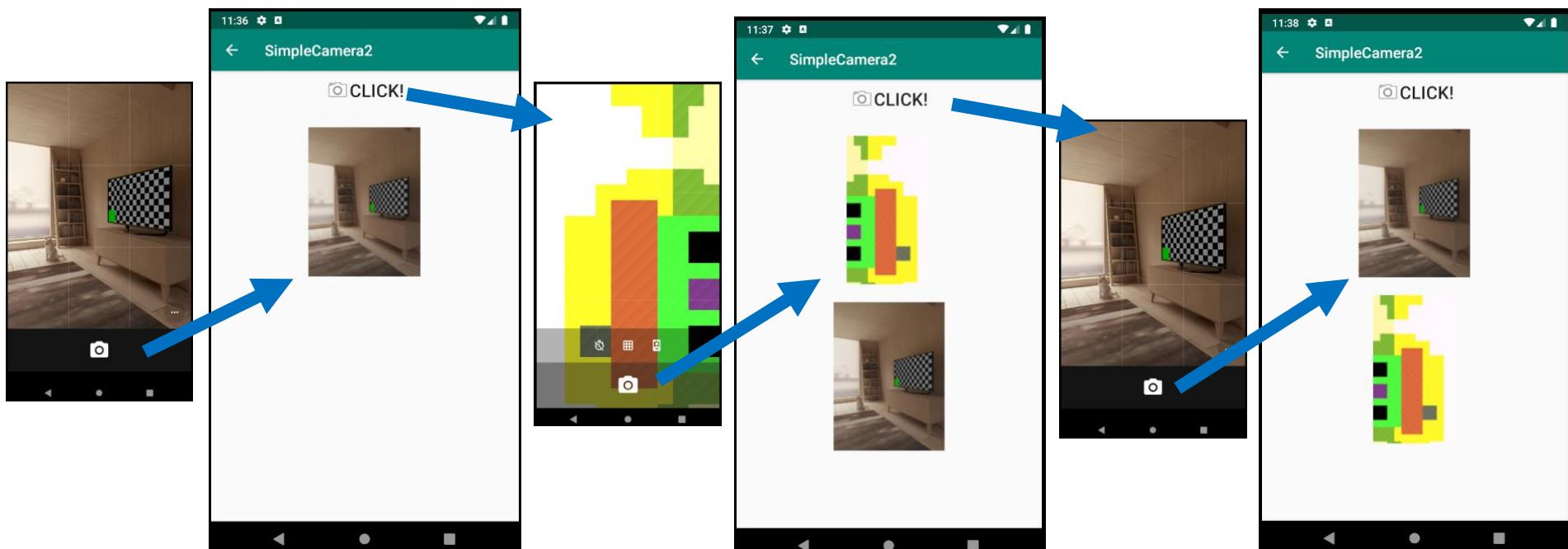
Initially, *no Events are detected*, then *both Click and LongClick can be detected*

Analyze what happens when **changing the return value of onLongClick(...)**
Which is the correct choice? True or False; and why?

Simple Camera App (Multiple Images)

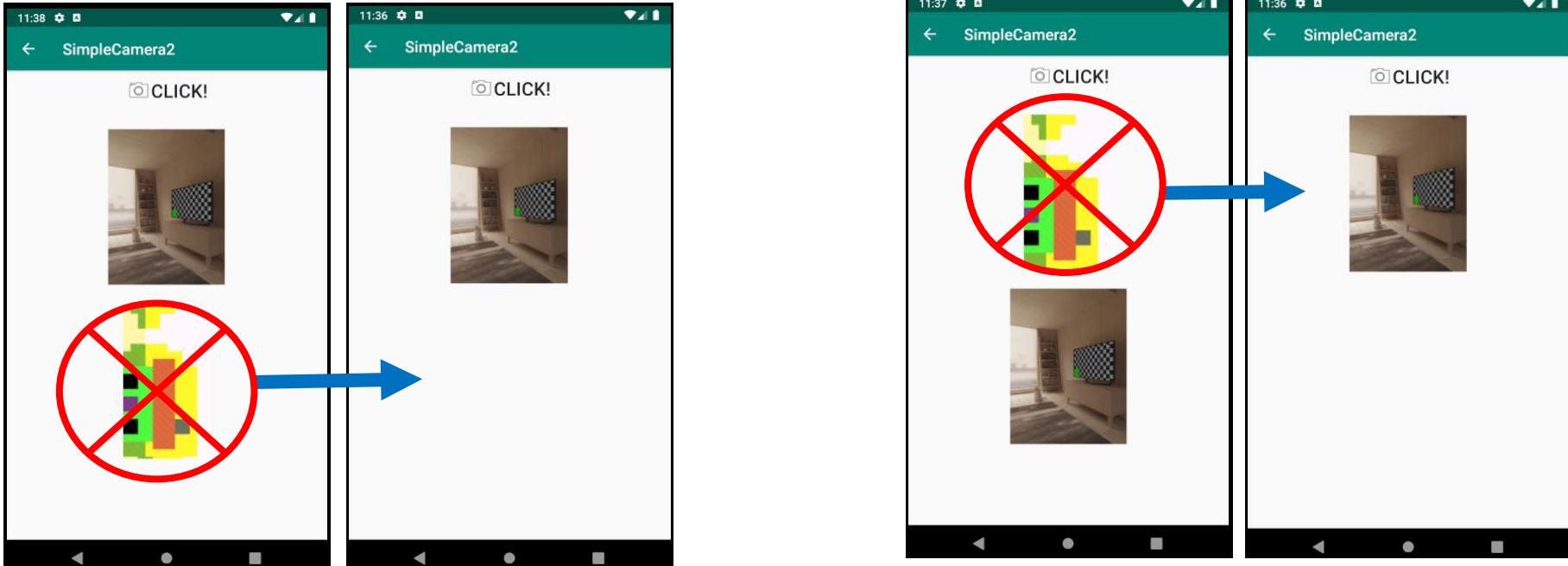
The **app** has to **visualize** the *two last taken images*. The **first image** shown (i.e., the one at the top) **must be the latest taken**

Make **FLAT** the **CLICK** button



Simple Camera App (Multiple Images)

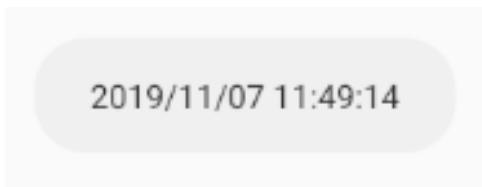
Allow the user **to delete** an **image** by '**longClicking**' on it



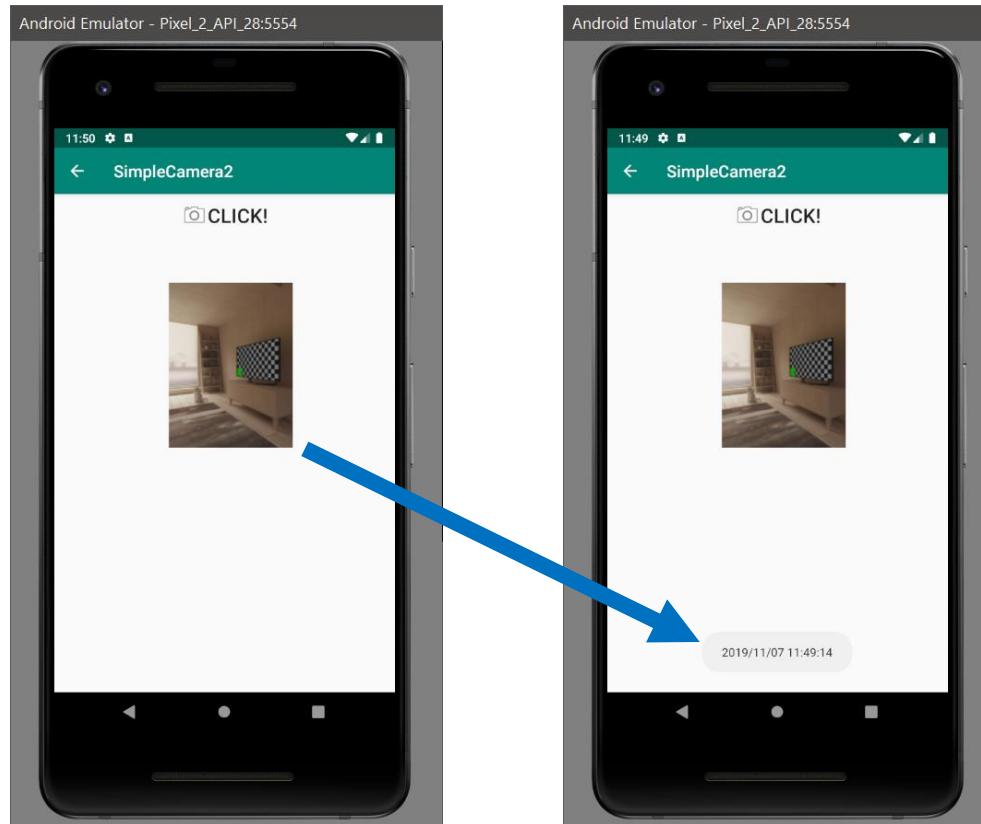
Note! When *deleting the first image*, the *second one should move in the first position*

Simple Camera App (Multiple Images)

Finally, when **clicking on an image** a '**Toast**' must appear showing the ***data and time*** of when the picture was taken



```
Context context = getApplicationContext();  
CharSequence text = "Click Event!";  
int duration = Toast.LENGTH_SHORT;  
  
Toast toast =  
    Toast.makeText(context, text, duration);  
toast.show();
```





Data Storage

Contents

- How to **store data** in Android
- ***Internal Storage***

These slides are partially based on the material that Google provides for the course
Android Developer Fundamentals

Storing data

Android **provides several options** for **saving persistent app data**

Possible solutions include:

- Internal Storage Private data (App-specific) files on device memory
- External Storage App-specific files or Public data on device or external storage
- Shared Preferences Private primitive data in key-value pairs
- SQLite Databases Structured data in a private database

The **best solution depends** on the **app specific needs**

- **data** should be **private** to the app OR **accessible** to other apps/user?
- **how much space** the data requires? **Complex structure** needed?

Storing data beyond Android

Possible Solutions include (continue):

- Network Connection **On the web** with **your own server**
- Cloud Backup Back up app and user **data in the cloud**
- Firebase Realtime Database Store and sync data with
NoSQL cloud database across clients in realtime
 - Consider it for the final project if needed...

Files

Internal and External Storage

Android File System

Android uses a **file system** that is **similar to disk-based file systems** on other platforms such as *Linux*

All **Android devices** have **two file storage areas**:

- *Internal* storage -- Private directories for just your app
- *External* storage -- Public directories

Apps can browse the directory structure

Internal storage

- Always **available to the app**
- Uses **device's file system**
- **Only your app can access files**
- **On app uninstall, system removes all app's files** from internal storage

External storage

- **Uses device's file system or physically external storage** like SD card
- **Not always available**, can be removed (SD card)
- **World-readable**, so **any app can read**
- On **uninstall, system does not remove files**
 - Exception: [getExternalFilesDir\(\)](#).

When to use internal/external storage

Internal is best when

- you **want to be sure** that **neither the user nor other apps can access your files**

External is best for files that

- **don't require access restrictions**
- **you want to share with other apps**
- **you allow the user to access with a computer**

Internal Storage

Internal Storage

Your app **always has permission** to **read** and **write** files in **its internal storage** directory

- **Permanent** storage directory – [getFilesDir\(\)](#)
- **Temporary** storage directory – [getCacheDir\(\)](#)
 - Recommended for **small, temporary files** totaling **less than 1MB**
 - Note that the **system may delete temporary files** if it runs **low on memory**

Internal Storage

Your app **always has permission** to **read** and **write** files in **its internal storage** directory

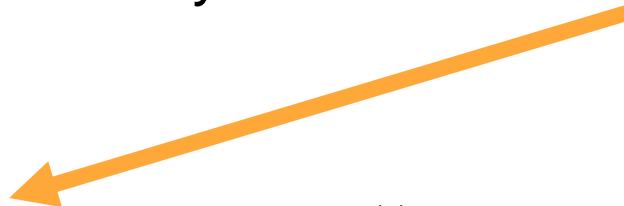
on Android 10 (API level 29) and higher, these locations are encrypted. These characteristics make these locations a good place to store sensitive data that only your app itself can access.

- Note that the **system may delete temporary files** if it runs ***low on memory***

Creating a file

To **create a new file** in **one of these *directories***, **use the `File()` constructor**, passing the File provided by one of the methods (`getFilesDir()`, `getCacheDir()`) that specifies your internal ***storage directory***. For example:

```
File file = new File(context.getFilesDir(), filename);
```



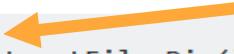
Then use standard [java.io](#) file operators or streams to interact with files

Write Text in an Internal File

First, get a file object

You'll need the storage path. For the internal storage, use:

```
File path = context.getFilesDir();
```



Then create your file object:

```
File file = new File(path, "my-file-name.txt");
```

Write a string to the file

```
FileOutputStream stream = new FileOutputStream(file);
try {
    stream.write("text-to-write".getBytes());
} finally {
    stream.close();
}
```

<https://stackoverflow.com/questions/14376807/how-to-read-write-string-from-a-file-in-android>

Read Text from an Internal File

Similarly to read a file (path defined as before):

```
File file = new File(path, "my-file-name.txt");
```

Read the file to a string

```
int length = (int) file.length();

byte[] bytes = new byte[length];

FileInputStream in = new FileInputStream(file);
try {
    in.read(bytes);
} finally {
    in.close();
}

String contents = new String(bytes);
```

Simple Camera App

Warmup: Save a text in the internal storage of the Simple Camera App

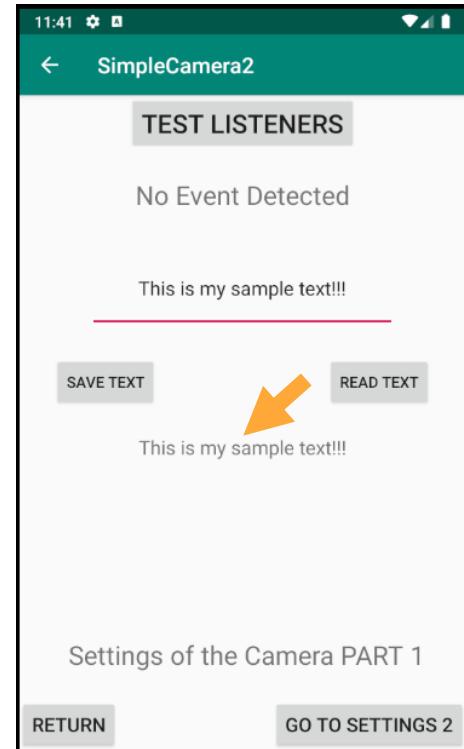
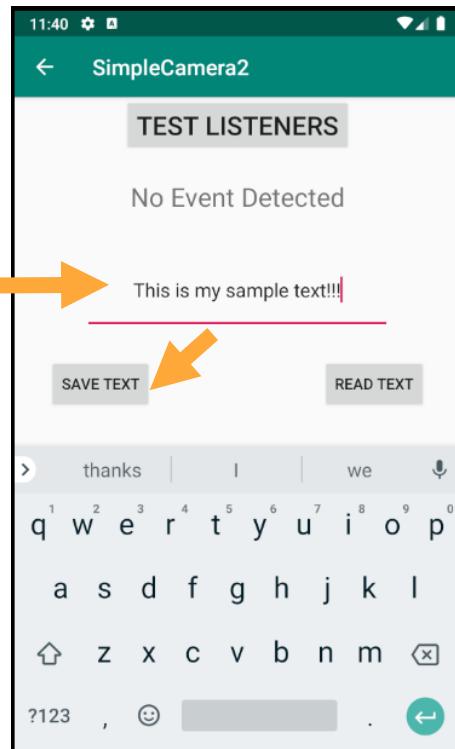
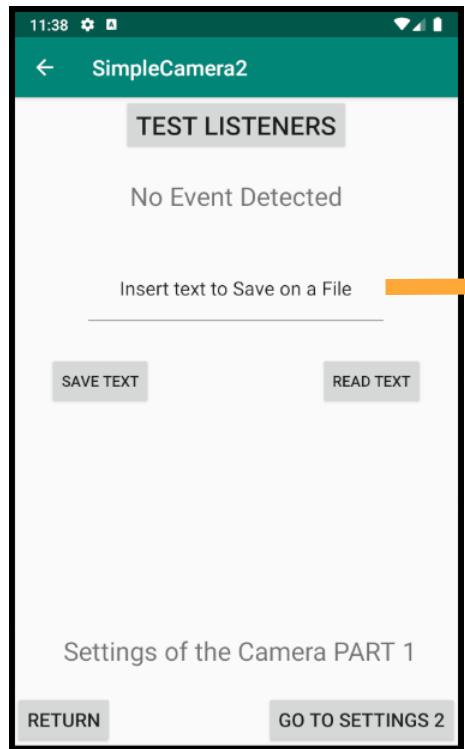
Add the following **UI elements** to the **Settings 1 Activity**

- A **TextView** for **inserting Text**
- A **button** for **saving the Text** in the internal storage (overwrite content)
- A **button** for **reading the text** from the internal storage
- A **TextView** for **showing the read text**



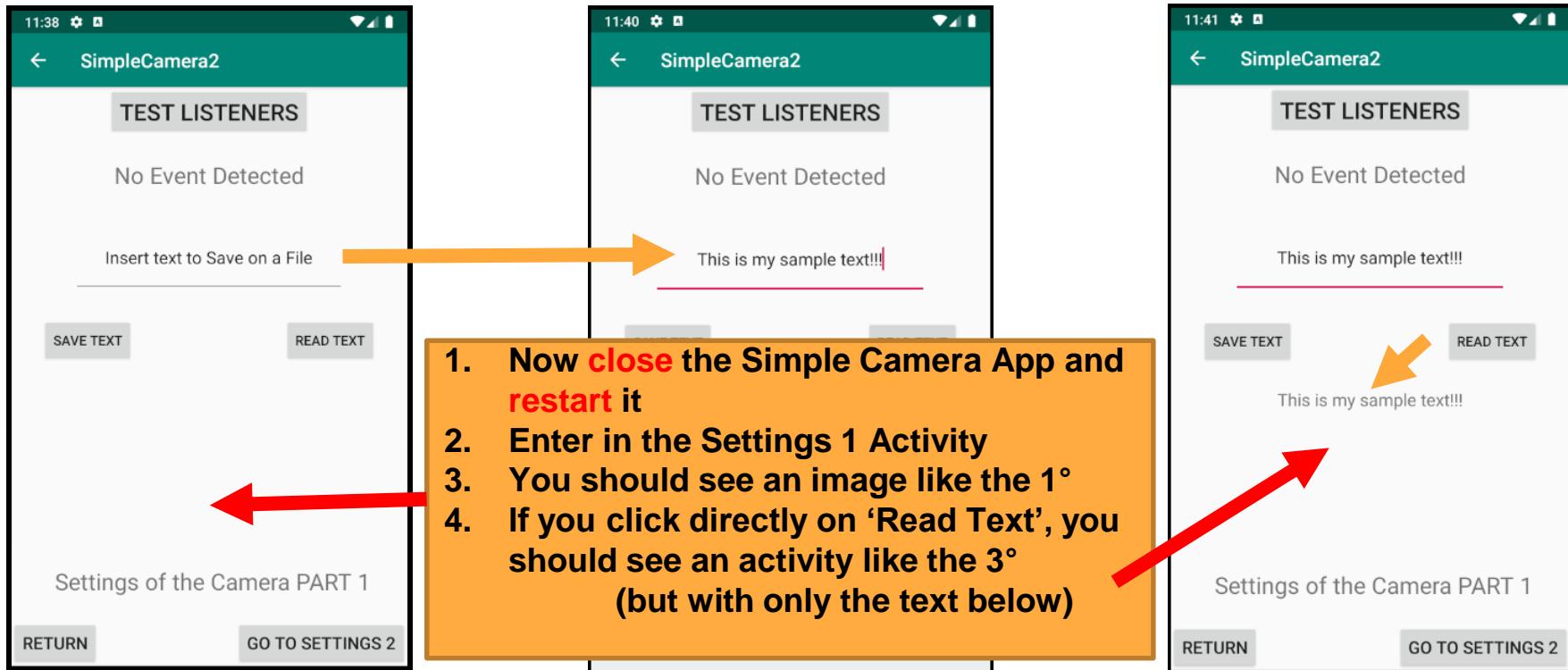
Simple Camera App

Example of usage



Simple Camera App

Example of usage



Simple Camera App

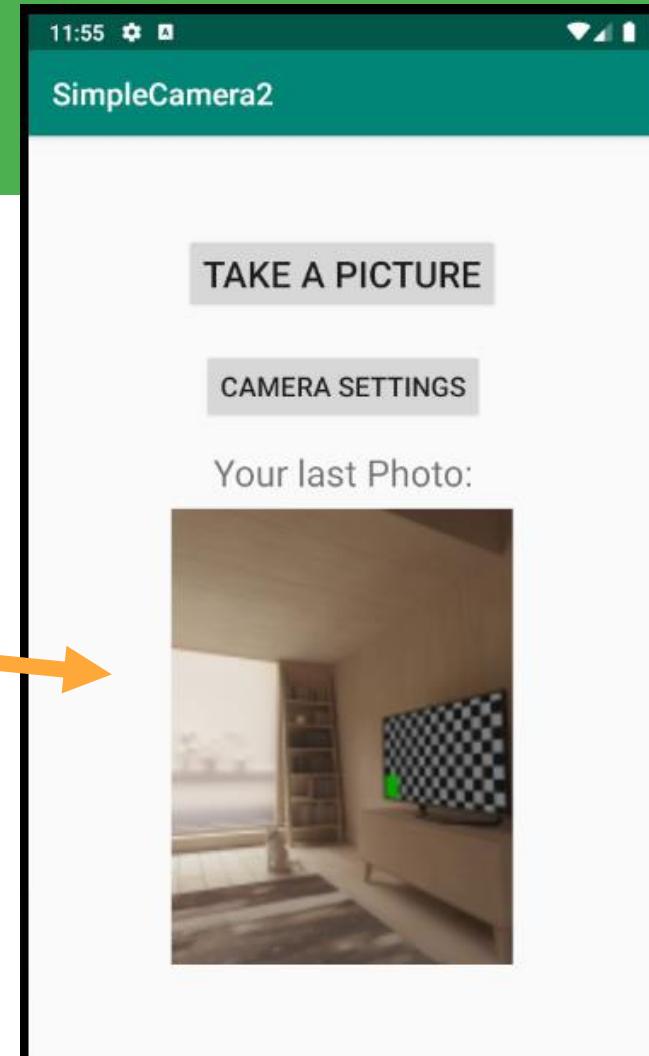
Save a preview of the last picture in the internal storage of the Simple Camera App

The preview (*if available*) should be visualized in the **main activity** of the app

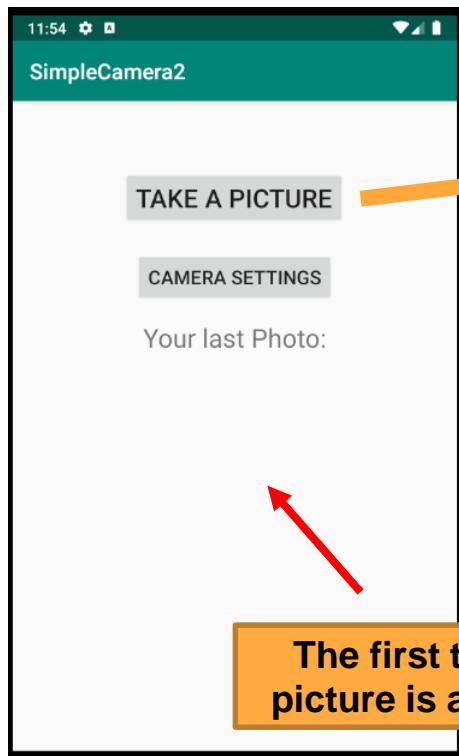
Hint:

Saving and Reading Bitmaps/Images from Internal memory in Android

<https://stackoverflow.com/questions/17674634/saving-and-reading-bitmaps-images-from-internal-memory-in-android>



Simple Camera App



The first time no picture is available

Take a picture and save it!
(or save automatically in 'CLICK!')

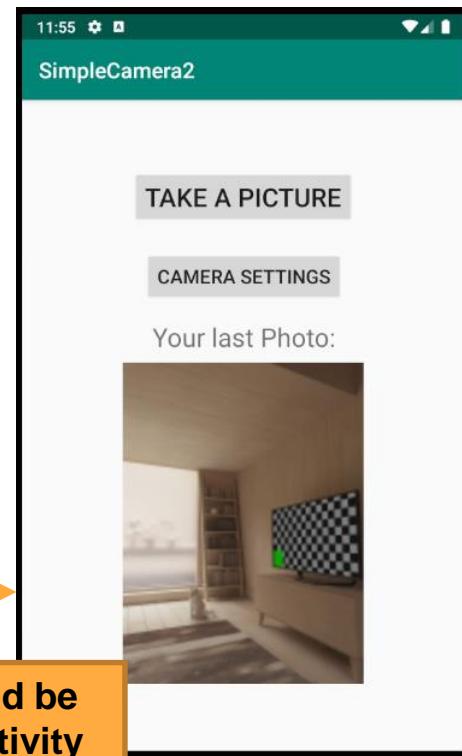
SimpleCamera2

CLICK!

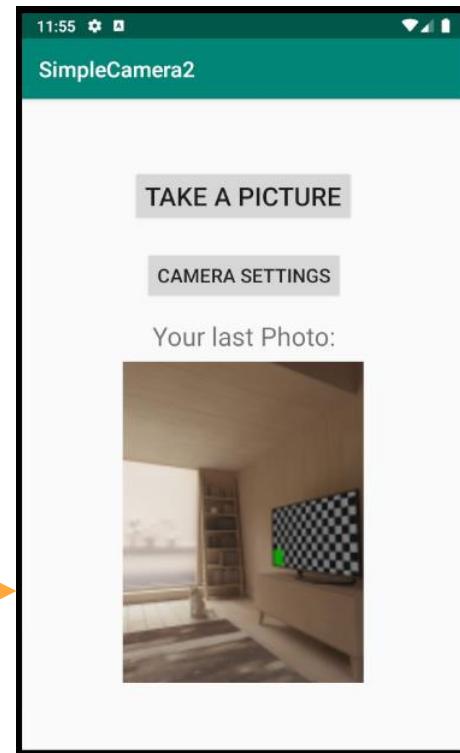
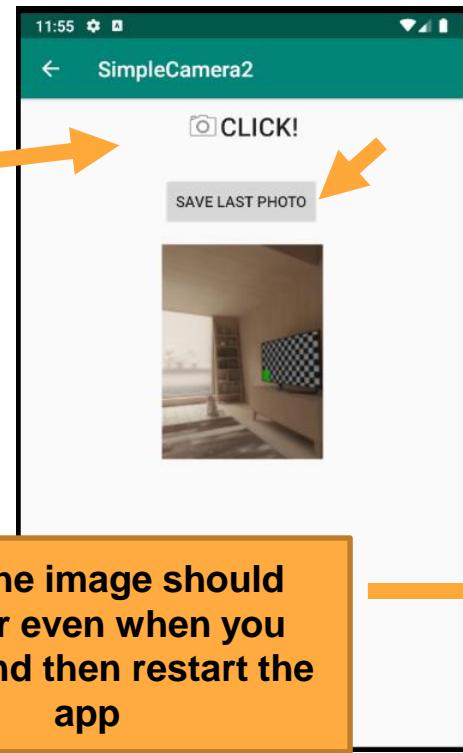
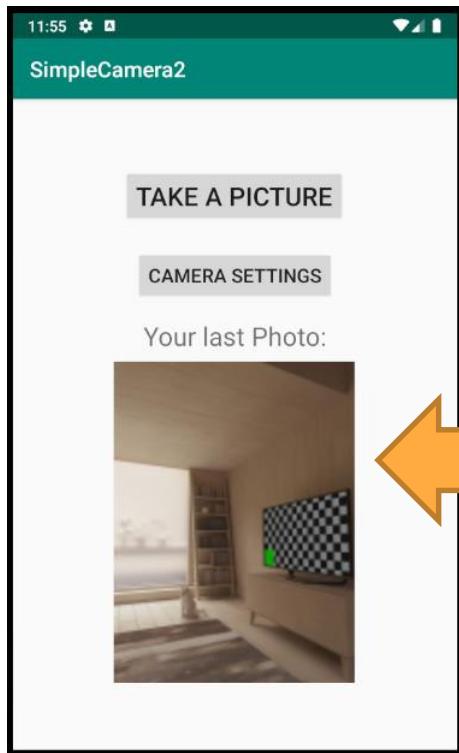
SAVE LAST PHOTO



Now the image should be visible in the Main activity



Simple Camera App

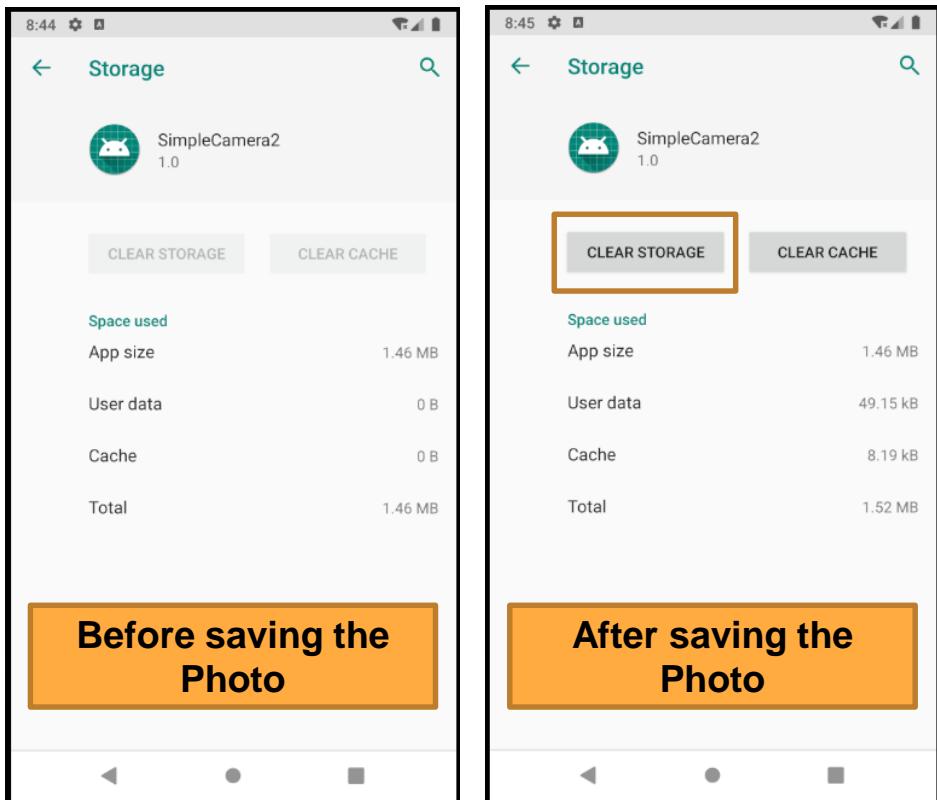


Now the image should appear even when you close and then restart the app

Simple Camera App

In the device/emulator
Settings -> App you can
find info about the size of Data
and Cache for your app

'Clear Storage' button removes
the saved image file (and thus the
preview of the last saved Photo),
as well as, the text file saved in
the first part of the exercise





Shared Preferences

Contents

- Shared Preferences

These slides are partially based on the material that Google provides for the course
Android Developer Fundamentals

What are Shared Preferences?

- Read and write **small amounts of primitive data**
 - as **key/value pairs** to a **file** on the **device storage**
- The **preference file** is **accessible to all the components of your app**, but **it is not accessible to other apps**
- **SharedPreference** provides APIs for **reading, writing, and managing** this data
- Save data in onPause()
restore in onCreate()

Shared Preferences AND Saved Instance State

For both:

- Data represented by a ***small number of key/value pairs***
- ***Data is private*** to the application

Shared Preferences vs. Saved Instance State

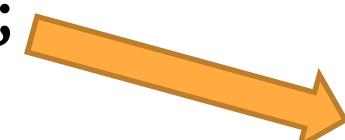
- Persist data **across user sessions**
 - even if app is killed and restarted, or device is rebooted
- Data that should be remembered across sessions, such as a **user's preferred settings** or their **game score**
- **Common use** is to store user preferences
- Preserves state data across activity instances **in same user session**
- Data that should not be remembered across sessions, such as the **currently selected tab** or **current state of activity**
- **Common use** is to recreate state after the device has been rotated

Creating Shared Preferences

- Need **only one Shared Preferences file per app**
- **Name it with package name of your app**
 - unique and easy to associate with app

getSharedPreferences()

```
private String sharedPrefFile =  
    "com.example.android.hellosharedprefs";  
  
mPreferences =  
    getSharedPreferences(sharedPrefFile,  
                        MODE_PRIVATE);
```



Creating Shared Preferences

- MODE argument for `getSharedPreferences()` is for **backwards compatibility**: use only `MODE_PRIVATE` to be secure!
 - The mode argument is required, because older versions of Android had other modes that allowed you to create a world-readable or world-writable shared preferences file. These modes were deprecated in API 17, and are now strongly discouraged for security reasons

Saving Shared Preferences

- SharedPreferences.Editor interface
 - It takes care of all file operations
 - Put methods overwrite if key exists
 - `apply()` saves asynchronously and safely

Saving Shared Preferences (details 1)

Save preferences in the **onPause()** state of the activity lifecycle using the SharedPreferences.Editor interface

- **Get a SharedPreferences.Editor**
 - The editor takes care of all the file operations for you
- **Add key/value pairs** to the editor **using the "put" method** appropriate for the data type
 - for example,.putInt() or putString()
 - These methods **will overwrite previously existing** values of an existing key

Saving Shared Preferences (details 2)

- Call **apply()** to write out your changes.
 - The apply() method **saves the preferences asynchronously, off of the UI thread**
- You don't need to worry about Android component lifecycles and their interaction with apply() writing to disk
 - The **framework makes sure in-flight disk writes from apply() complete before switching states.**

SharedPreferences.Editor

```
public class MainActivity extends AppCompatActivity {  
    private SharedPreferences mPreferences;  
  
    ...  
  
    @Override  
    protected void onPause() {  
        super.onPause();  
        SharedPreferences.Editor preferencesEditor = mPreferences.edit();  
        preferencesEditor.putInt("count", mCount);  
        preferencesEditor.putInt("color", mCurrentColor);  
        preferencesEditor.apply();  
    }  
}
```

Restoring Shared Preferences

- Restore in **onCreate()** in Activity
- Get methods take two arguments
 - the **key**
 - the **default value** (if the key cannot be found)
- Use **default argument** so you **do not have to test whether the preference exists in the file**

Getting data in onCreate()

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    String sharedPrefFile = "com.example.simplesavingsapp";  
    mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);  
    mCount = mPreferences.getInt("count", 1);  
    mCurrentColor = mPreferences.getInt("color", 0);  
    ... // use that values e.g., for initializing UI widgets  
}
```

Clearing

- Call clear() on the SharedPreferences.Editor and apply changes
- You can combine calls to put and clear. However, when you apply(), clear() is always done first, regardless of order!

clear()

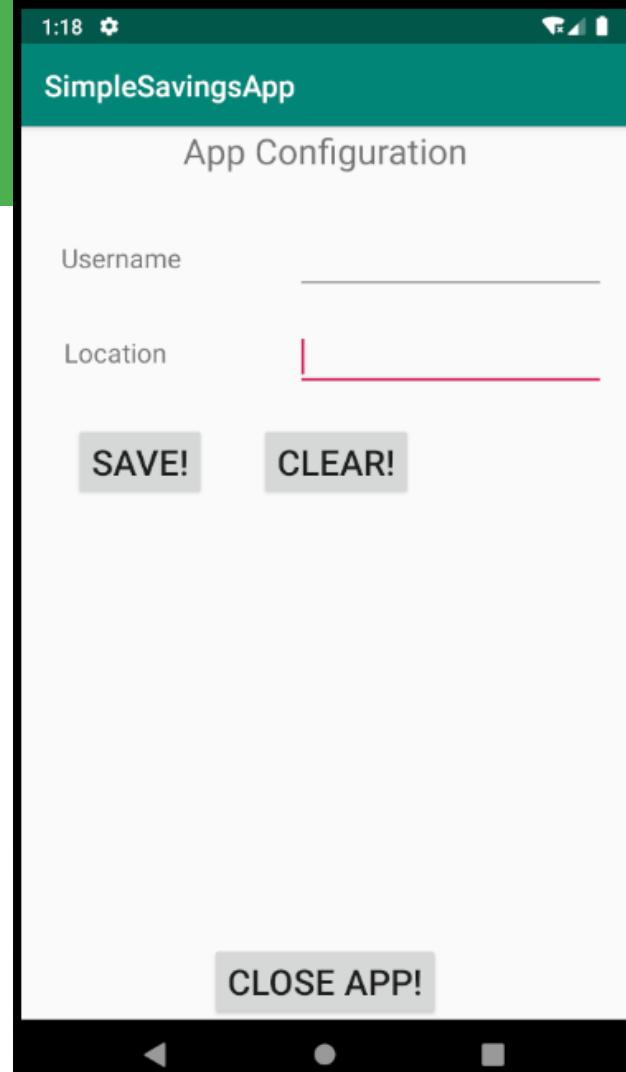
```
SharedPreferences.Editor preferencesEditor =  
    mPreferences.edit();  
  
preferencesEditor.clear();  
  
preferencesEditor.apply();
```

SharedPref App

In this app you will **enable the user** to **define preferences** that should be saved and reused across different executions of the app

The user can:

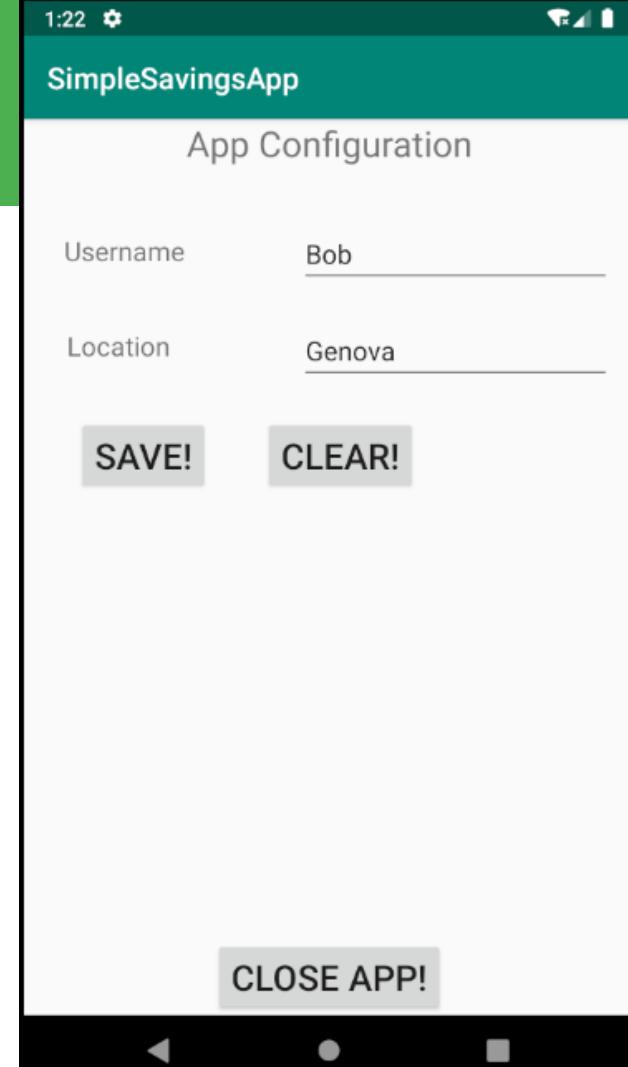
- Save Username and Location
- Delete them
- Close the app
 - `finishAndRemoveTask();`



SharedPref App

When the app starts:

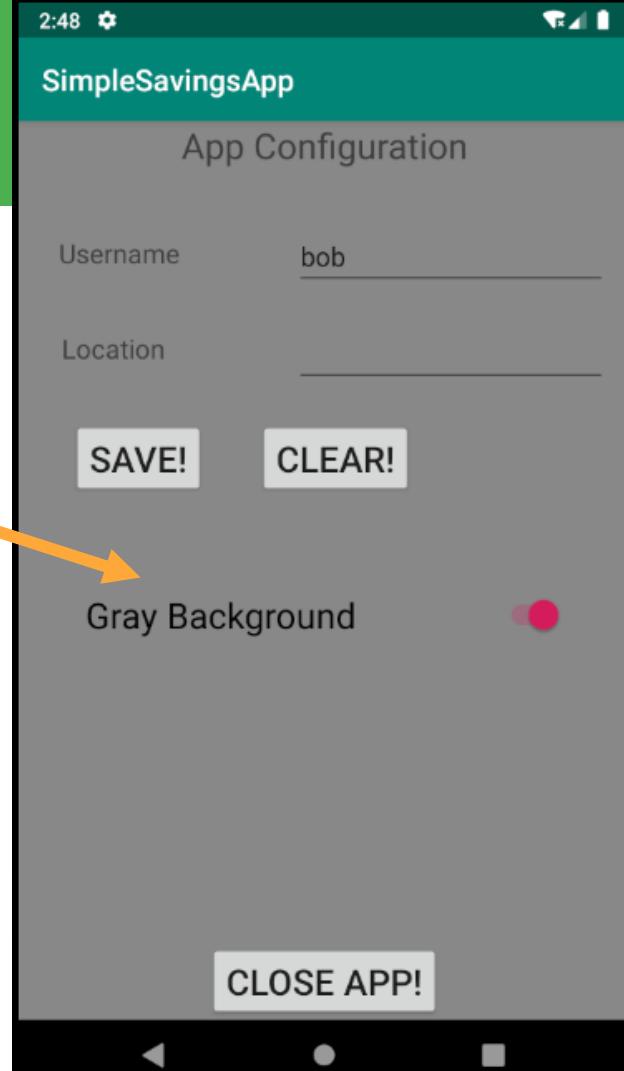
- first time, preferences must be empty
- Otherwise, if preferences has been defined they must be used to populate the fields



SharedPref App

Additional Behaviors:

- Add a choice for selecting two possible background colors (e.g., white and gray)
- Apply the selected color to the app (this color should be used also if the app is re-started)
 - Use a 'Switch' UI widget. Possible useful methods:
 - `setChecked(...)`
 - to set up the switch, depending on the initial value of the background on app startup (maintain consistency between color saved and switch state)
 - `isChecked()`
 - `getWindow().getDecorView().setBackgroundColor(Color.GRAY);`





Data Storage (II)

Android Storage Racap

- Files
 - App-specific storage
 - Internal
 - External
 - Shared storage
 - Media & Doc
- Preferences
- Databases

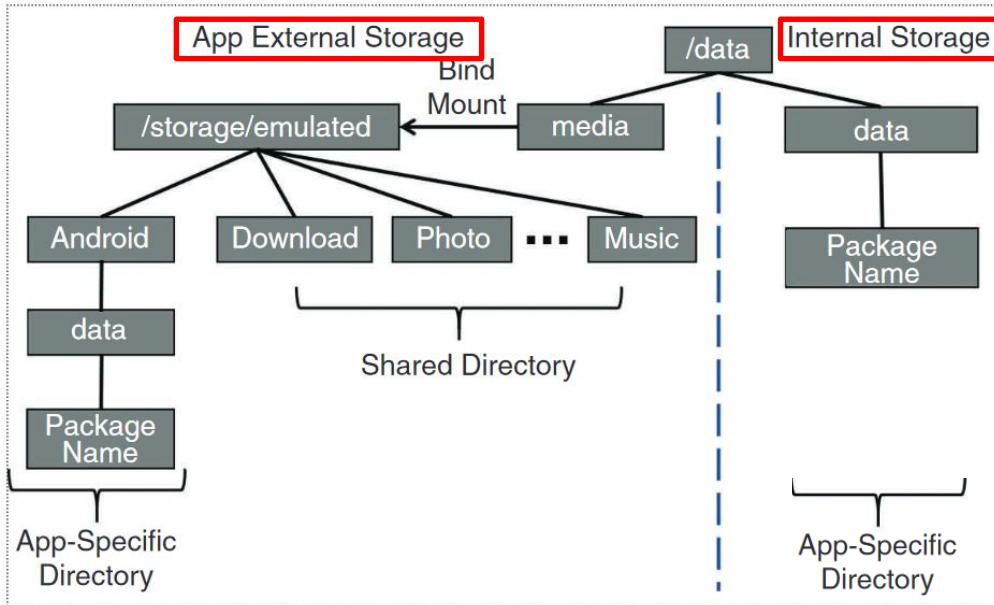


Image adapted from:
IEEE Security & Privacy
Demystifying Android's Scoped Storage Defense. Sept.-Oct. 2021, pp. 16-25, vol. 19
DOI Bookmark: 10.1109/MSEC.2021.3090564

Android Storage Racap

- Files
 - App-specific storage
 - Internal
 - External
 - Shared storage
 - Media & Doc
- Preferences
- Databases

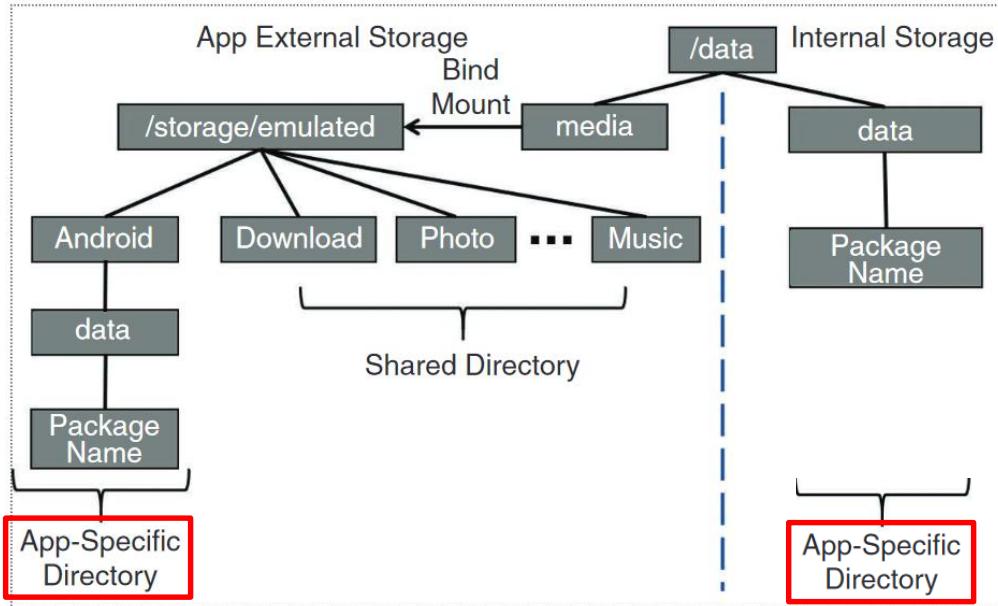


Image adapted from:
IEEE Security & Privacy
Demystifying Android's Scoped Storage Defense. Sept.-Oct. 2021, pp. 16-25, vol. 19
DOI Bookmark: 10.1109/MSEC.2021.3090564

Android Storage Racap

- Files
 - App-specific storage
 - Internal
 - External
 - Shared storage
 - Media & Doc
- Preferences
- Databases

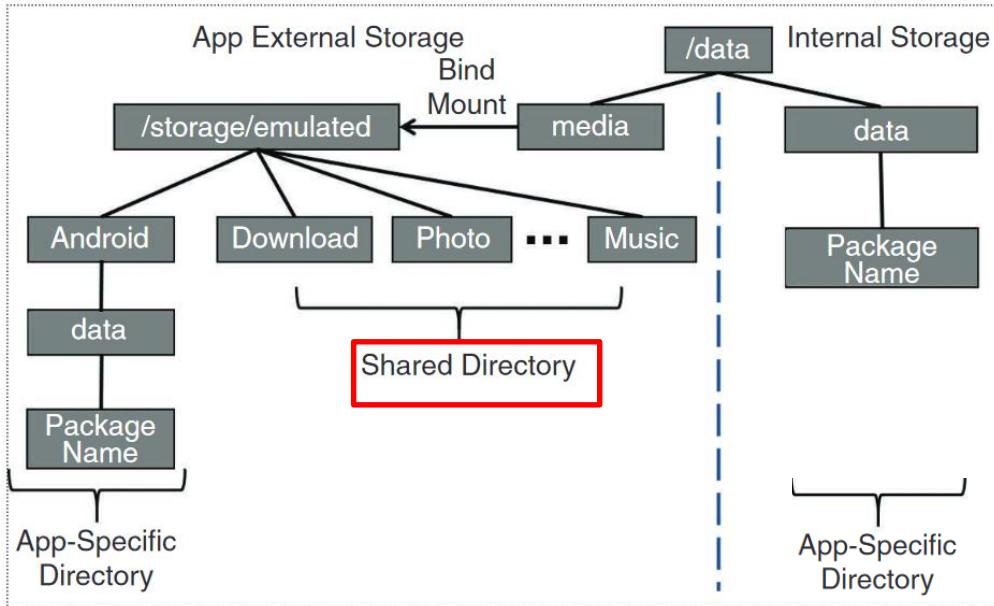
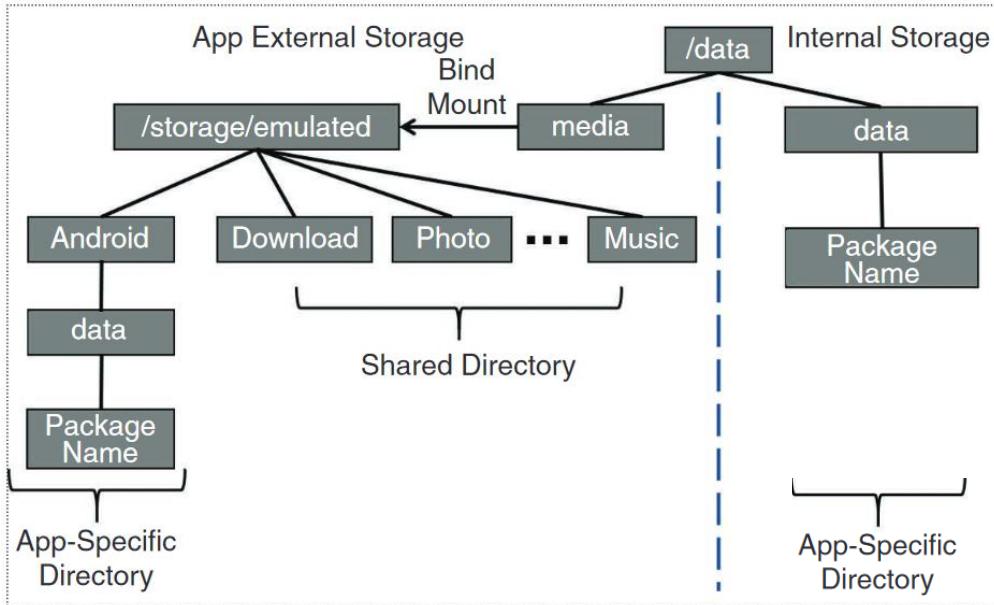


Image adapted from:
IEEE Security & Privacy
Demystifying Android's Scoped Storage Defense. Sept.-Oct. 2021, pp. 16-25, vol. 19
DOI Bookmark: 10.1109/MSEC.2021.3090564

Android Storage Racap

- Files
 - App-specific storage
 - Internal
 - External
 - Shared storage
 - Media & Doc



- Preferences
- Databases

Image adapted from:
IEEE Security & Privacy
Demystifying Android's Scoped Storage Defense. Sept.-Oct. 2021, pp. 16-25, vol. 19
DOI Bookmark: 10.1109/MSEC.2021.3090564

Android Storage Racap

- Files
 - App-specific storage
 - Internal
 - External
 - Shared storage
 - Media & Doc
- Preferences
- Databases

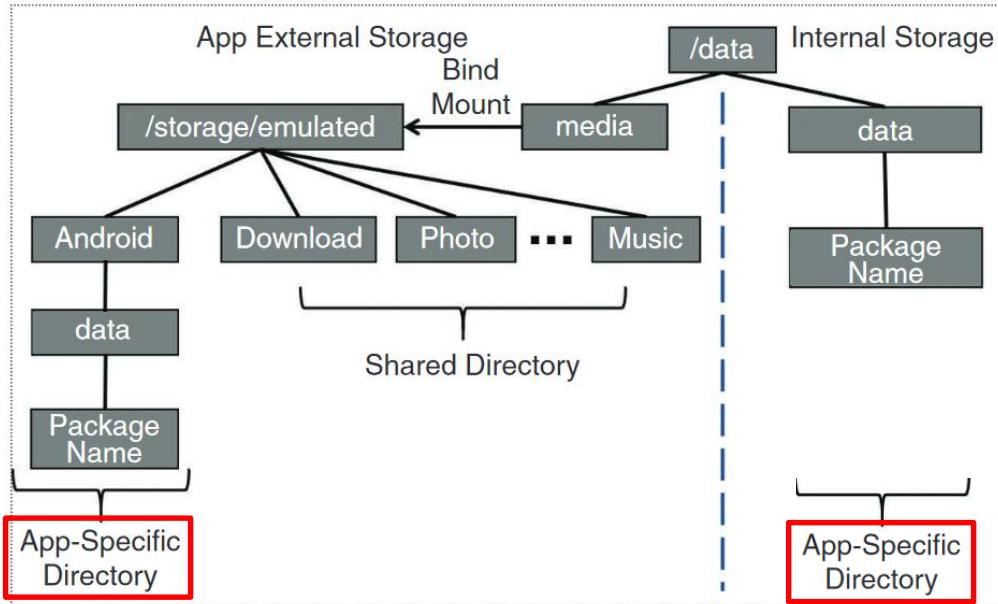


Image adapted from:
IEEE Security & Privacy
Demystifying Android's Scoped Storage Defense. Sept.-Oct. 2021, pp. 16-25, vol. 19
DOI Bookmark: 10.1109/MSEC.2021.3090564

App-specific files Storage

Both Internal and External storage include a dedicated location for

- storing persistent files
- storing cache data

Files stored in these directories are meant for use only by your app
(App-specific)

- Otherwise use Shared storage (Photo, Video, Docs etc)

App-specific files Storage

When storing sensitive data (data that shouldn't be accessible from any other app), use

- internal storage
- Preferences
- database

Internal storage => data being hidden from users

When the user uninstalls your app, the files saved in app-specific storage are removed

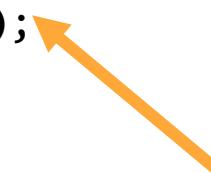
External Storage for App-specific files

- If internal storage doesn't provide enough space to store app-specific files
=> use external storage
- The system provides directories within external storage where an app can organize files that provide value to the user only within your app
- The files in these directories aren't guaranteed to be accessible, such as when a removable SD card is taken out of the device. If your app's functionality depends on these files => internal storage

Type of content	Access method	Permissions needed	Can other apps access?	Files removed on app uninstall?
App-specific files	Files meant for your app's use only	From internal storage, <code>getFilesDir()</code> or <code>getCacheDir()</code> From external storage, <code>getExternalFilesDir()</code> or <code>getExternalCacheDir()</code>	Never needed for internal storage Not needed for external storage when your app is used on devices that run Android 4.4 (API level 19) or higher	No Yes
Media	Shareable media files (images, audio files, videos)	MediaStore API	READ_EXTERNAL_STORAGE when accessing other apps' files on Android 11 (API level 30) or higher READ_EXTERNAL_STORAGE or WRITE_EXTERNAL_STORAGE when accessing other apps' files on Android 10 (API level 29) Permissions are required for all files on Android 9 (API level 28) or lower	Yes, though the other app needs the READ_EXTERNAL_STORAGE permission No
Documents and other files	Other types of shareable content, including downloaded files	Storage Access Framework	None	Yes, through the system file picker No
App preferences	Key-value pairs	Jetpack Preferences library	None	No Yes

Always check availability of storage

- Because the **external storage may be unavailable**
 - such as when the user has **mounted the storage to a PC** or has **removed the SD card** that provides the external storage
- you should **always verify that the volume is available** before accessing it

```
/* Checks if external storage is available for read and write */  
  
public boolean isExternalStorageWritable() {  
    String state = Environment.getExternalStorageState();  
    if (Environment.MEDIA_MOUNTED.equals(state)) {  
        return true;  
    }  
    return false;  
}  
  
  
if the returned state is equal to  
MEDIA_MOUNTED, then you can read  
and write your files
```

Always check availability of storage (2)

```
/* Checks if external storage is available to at least read */

public boolean isExternalStorageReadable() {

    String state = Environment.getExternalStorageState();

    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {

        return true;

    }

    return false;
}
```

Always check availability of storage (2)

```
/* Checks if external storage is available to at least read */

public boolean isExternalStorageReadable() {

    String state = Environment.getExternalStorageState();

    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {

        return true;
    }

    return false;
}
```

Returns the current state of the primary external storage media...

otherwise use (e.g., for SD card etc):

String getExternalStorageState (File path)

Accessing External storage directories

1. Get a path using **getExternalFilesDir()**
2. Create file

Example

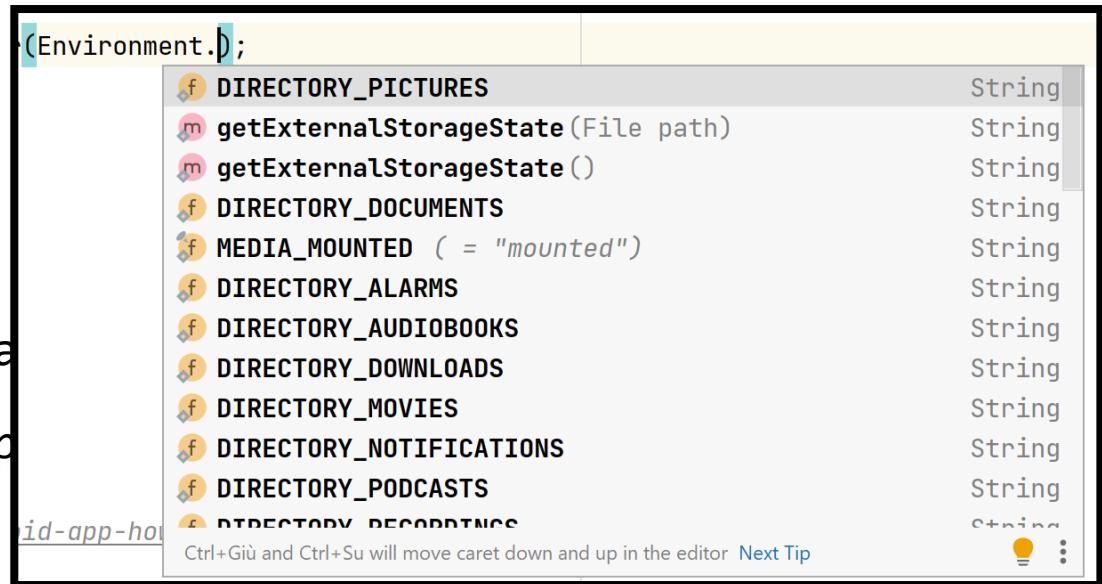
```
File path = getExternalFilesDir(Environment.DIRECTORY_PICTURES);  
File file = new File(path, "DemoPicture.jpg");
```

Accessing External storage directories

1. Get a path using **getExternalFilesDir()**
2. Create file

Example

```
File path = getExternalFilesDir(null);  
File file = new File(path, "myfile.txt");
```



See: <https://developer.android.com/reference/android/os/Environment#fields>

Select a physical storage location

A device that allocates a partition of its internal memory as external storage can also provide an SD card slot.

This means that the device has multiple physical volumes that could contain external storage, so you need to select which one to use for your app-specific storage

```
File[] externalStorageVolumes =  
    ContextCompat.getExternalFilesDirs(getApplicationContext(), Environment.DIRECTORY_PICTURES);  
  
// probably a partition of the device internal memory as external storage  
File pathPrimaryExternalStorage = externalStorageVolumes[0];  
// probably this is the SD card  
File pathSecondaryExternalStorage = externalStorageVolumes[1];
```

the first element in the returned array (i.e., [0]) is considered the primary external storage volume

How much storage left?

- If there is not enough space, throws IOException
- If you know the size of the file, check against space
 - getFreeSpace()
 - getTotalSpace().
- If you do not know how much space is needed
 - try/catch IOException

How much storage left?

- If there is not enough space, throws [IOException](#)
- If you know the size of the file, check against space
 - [getFreeSpace\(\)](#)
 - [getTotalSpace\(\)](#).
- If you do not know how much space is needed
 - try/catch [IOException](#)

Delete files no longer needed

- External storage

```
myFile.delete();
```

- Internal storage

```
myContext.deleteFile(fileName);
```

Do not delete the user's files!

When the user uninstalls your app, your app's private storage directory and all its contents are deleted

Do not use private storage for content that belongs to the user!

For example

- Photos captured or edited with your app
- Music the user has purchased with your app

Simple Camera App

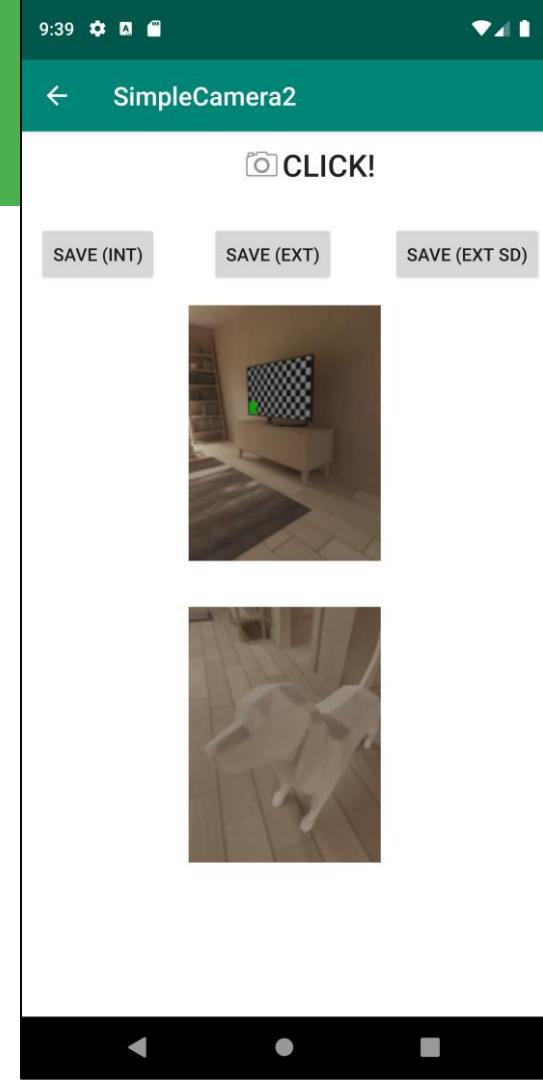
Add the possibility to save the **last picture** taken in the **External App-specific directories**:

in particular in both the:

- **External storage**
- **External SD removable storage**

Add the following *UI elements* to the *Photo Activity*

- A button for saving the Image in the external storage
- A button for saving the Image in the external SD removable storage



Simple Camera App

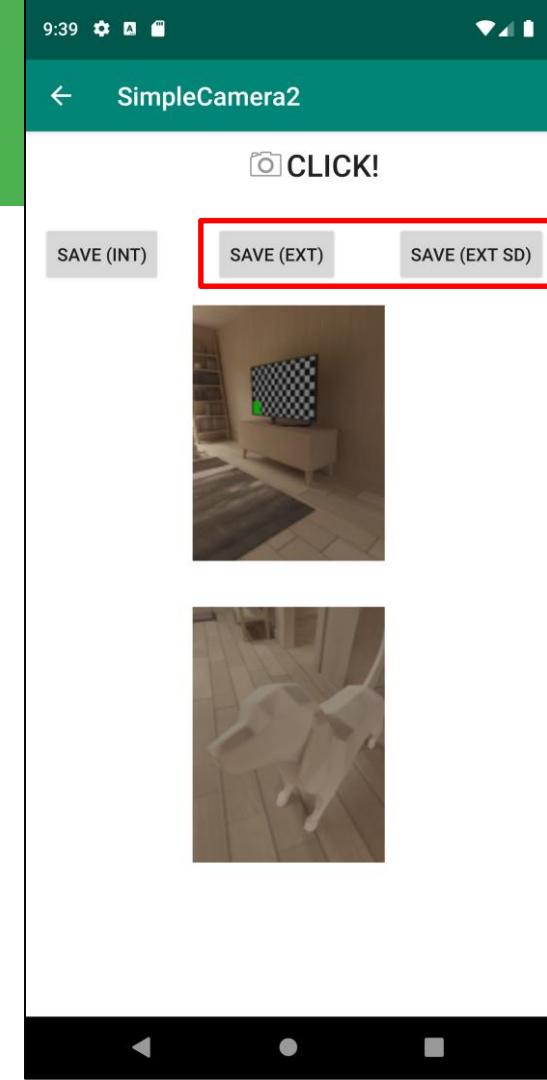
Add the possibility to save the **last picture** taken in the **External App-specific directories**:

in particular in both the:

- **External storage**
- **External SD removable storage**

Add the following ***UI elements*** to the ***Photo Activity***

- A **button for saving the Image** in the external storage
- A **button for saving the Image** in the external SD removable storage

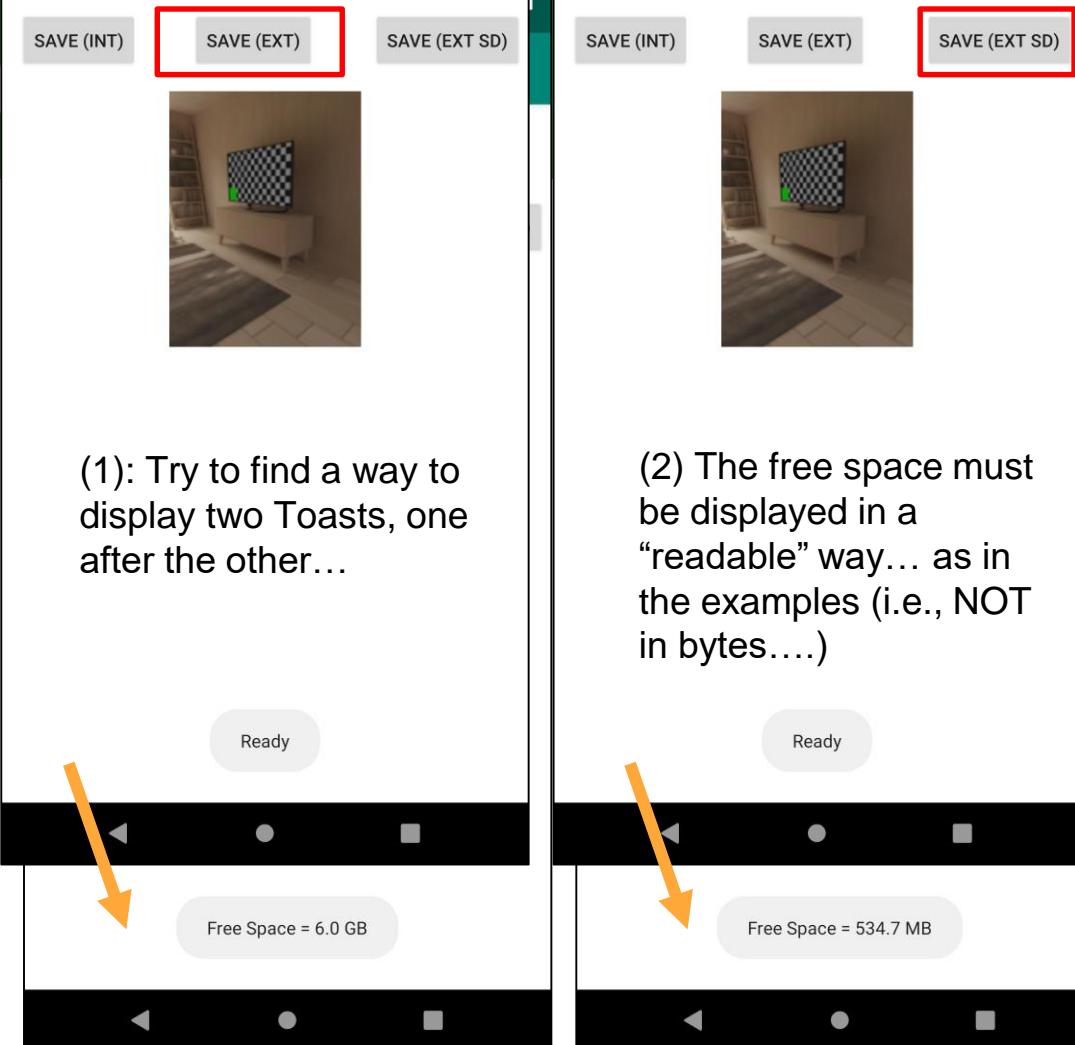


Simple Camera A

Then each time the user clicks on the two new save buttons:

- a Toast appears to notify if the corresponding **external storage is available or not**
External or SD card
- After 2-3 sec another Toast appears that shows the **free space available** on the selected **external/SD storage**

Clearly, proceed with saving the image only if the storage is “Ready”



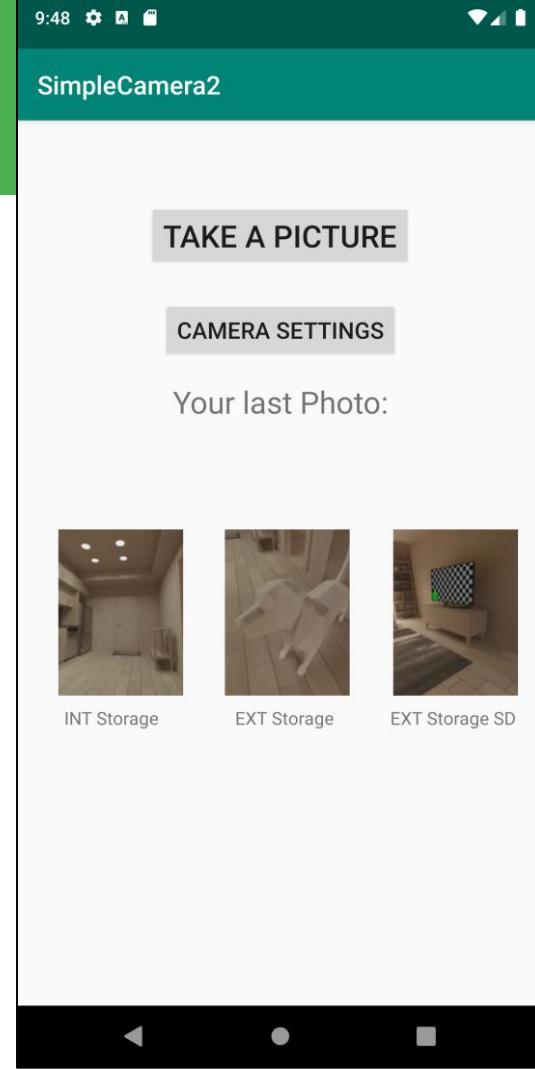
Simple Camera App

Finally, modify the main activity to show the images saved on the three possible locations for the App-specific files:

- *Internal storage*
- *External storage*
- *External SD removable storage*

Navigate the device file system to find the two images saved on the

- *External storage*
- *External SD removable storage*





AsyncTask

Contents

- Threads
 - Main/UI Thread
 - Worker Thread
- AsyncTask
 - Recently deprecated
 - but a simple way for experiment with MultiThreading....



This class was deprecated in API level 30.

Use the standard `java.util.concurrent` or [Kotlin concurrency utilities](#) instead.

These slides are partially based on the material that Google provides for the course
Android Developer Fundamentals

<https://developer.android.com/courses/fundamentals-training/overview>

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



Threads

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



The Main thread

When an **Android app starts**, it creates the **main thread**, which is often called the ***UI thread***

The **UI thread** **needs to give its attention** to **drawing the UI** and keeping the **app responsive** to user input

- App runs on Java thread called "main" or "UI thread"
- UI thread draws UI on the screen

Users uninstall unresponsive apps

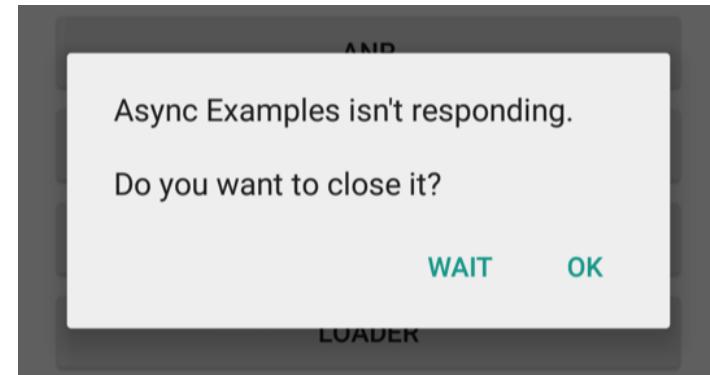
- If the **UI waits too long** for an operation to finish



- it **becomes unresponsive**



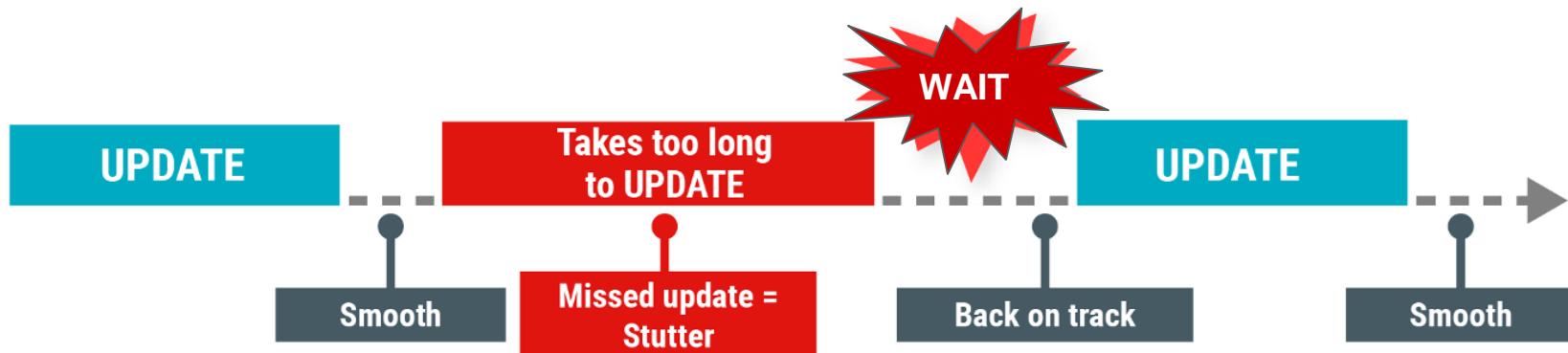
- User not happy!



- The framework shows an Application Not Responding (ANR) dialog

The Main thread must be fast

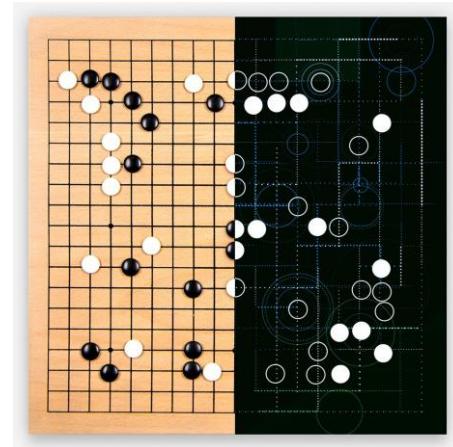
- Hardware **updates screen** every **16 milliseconds** (60 fps)
- UI thread has **16 ms** to do all its work
- If it takes **too long**, app seems **to hang** or **to be blocked**



What is a long running task?

Examples of possible **long running task**:

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

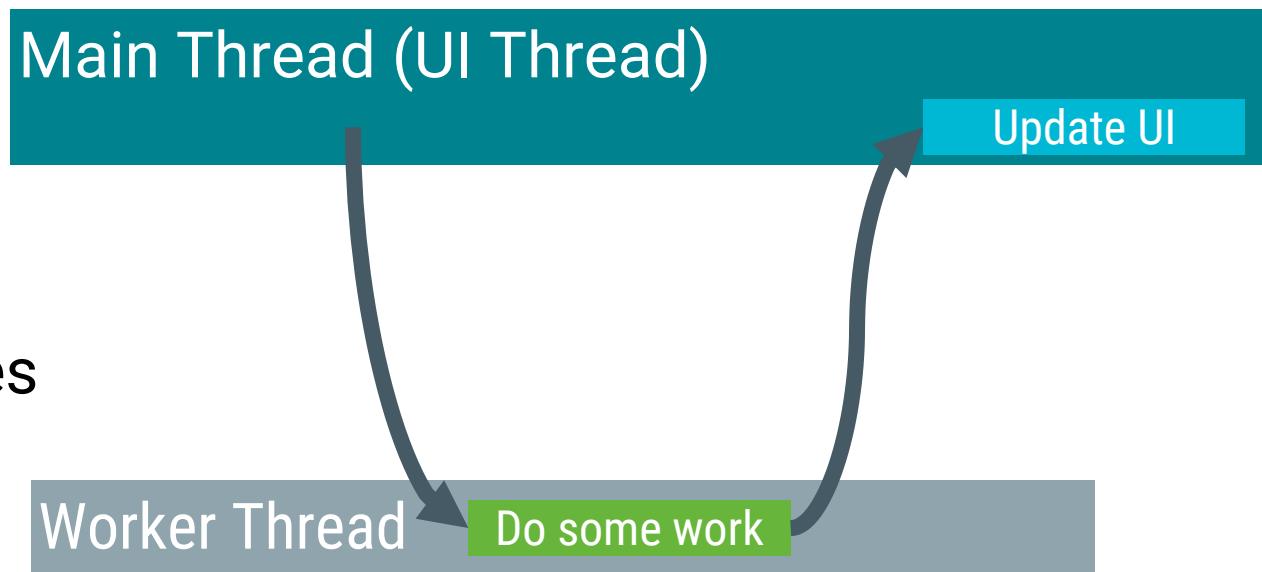


Background threads

Solution: Execute long running tasks on a background thread

How?

- **AsyncTask**
- Kotlin coroutines
- RxJava
- Executors



<https://blog.logrocket.com/moving-away-kotlin-async-task-alternative-solutions/>

Two rules for Android threads

- **Do not block** the UI thread
 - Complete each task in less than 16 ms for each screen
 - Run slow non-UI tasks 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

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



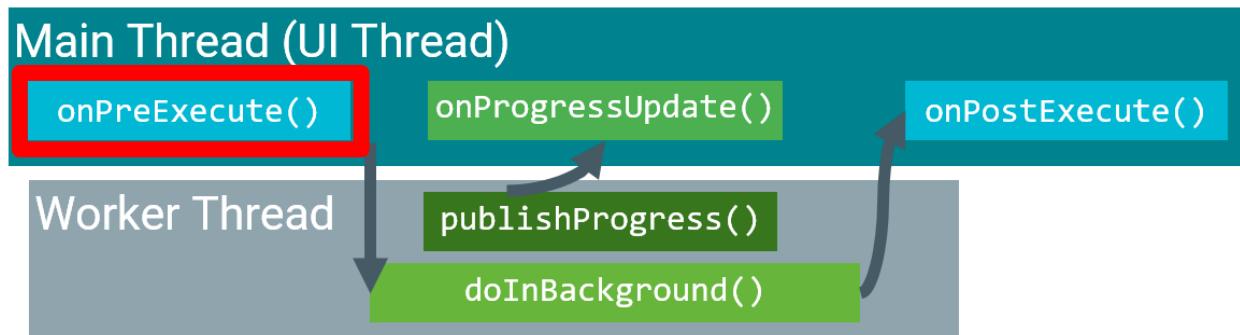
What is AsyncTask?

- **AsyncTask** allows to
 - **perform background operations** on a **worker thread**
 - **publish results** on the **UI thread**
- ***without needing*** to **directly manipulate threads or handlers**
- A **worker thread** is **any thread which is not the main or UI thread**

AsyncTask Execution Steps

When AsyncTask is executed, it goes through several steps:

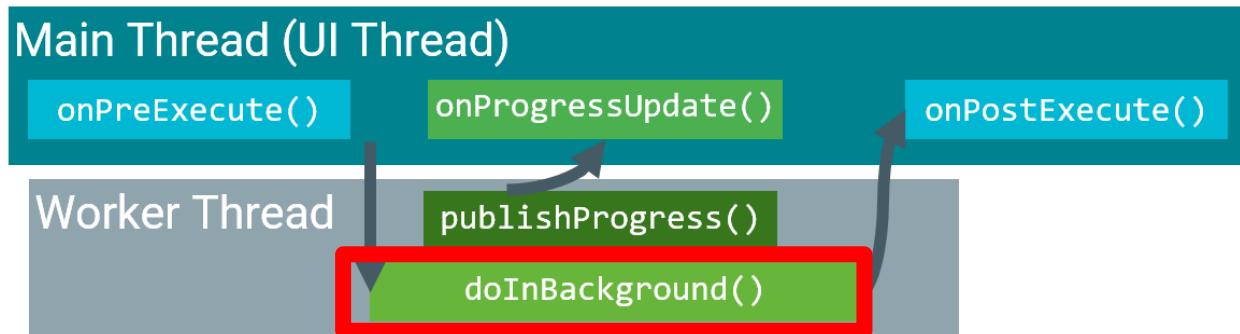
- **onPreExecute()** – is **invoked on the UI thread** before the **task is executed**
 - normally **used to set up the task**
 - for instance by showing a progress bar in the UI



AsyncTask Execution Steps

When AsyncTask is executed, it goes through several steps:

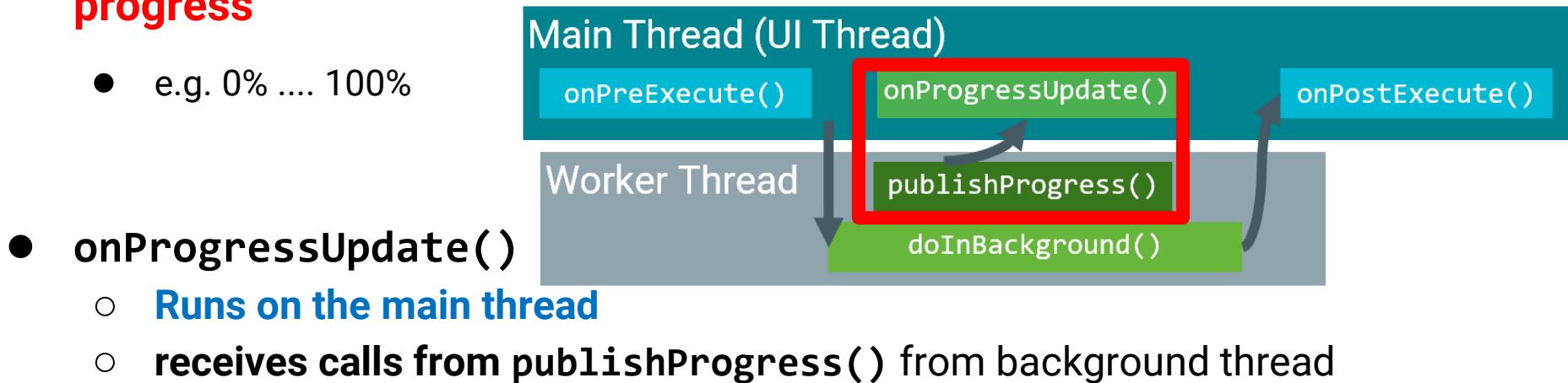
- `doInBackground()`— is **invoked on the background thread** immediately after `onPreExecute()` finishes
 - **performs a background computation**, returns a result, and passes the result to `onPostExecute()`



AsyncTask Execution Steps

When AsyncTask is executed, it goes through several steps:

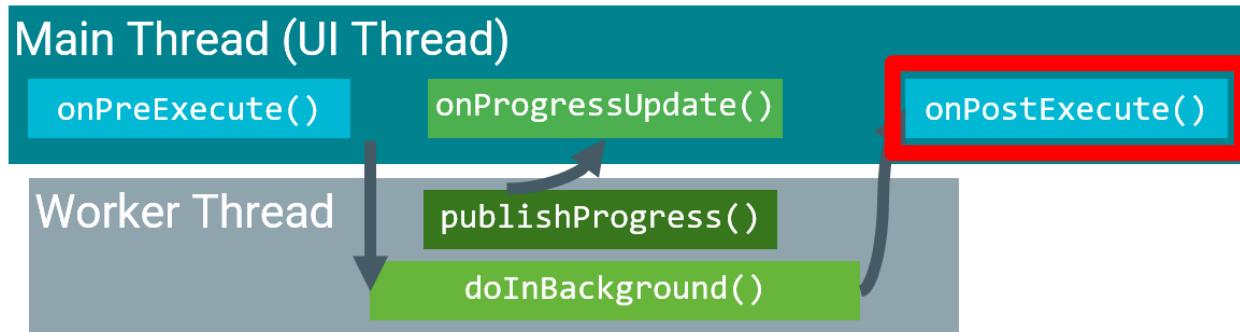
- The `doInBackground()` method can also call `publishProgress(Progress...)` to **publish one or more units of progress**
 - e.g. 0% 100%



AsyncTask Execution Steps

When AsyncTask is executed, it goes through several steps:

- **onPostExecute()**—**runs on the UI thread** after the background computation has finished
 - The result of the background computation is passed to this method as a parameter



Creating an AsyncTask

Subclass AsyncTask:

```
private class MyAsyncTask  
    extends AsyncTask<type1, type2, type3> {...}
```

1. “Params” - Provide data type (*type1*) sent to doInBackground()
2. “Progress” - Provide data type (*type2*) of progress units for onProgressUpdate()
3. “Result” - Provide data type (*type3*) of result for onPostExecute()

MyAsyncTask class definition example

```
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
 - e.g., AsyncTask<void, void, Bitmap>

AsyncTask (template) example

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        ...  
    }  
  
    public void startTask (View view) {  
        // Start the AsyncTask.  
        // The AsyncTask has a callback that will update the text view.  
        new DownloadFilesTask().execute(url1, url2, url3);  
    }  
  
    private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {  
  
        protected Long doInBackground(URL... urls) {  
            int count = urls.length;  
            long totalSize = 0;  
            for (int i = 0; i < count; i++) {  
                totalSize += Downloader.downloadFile(urls[i]);  
                publishProgress((int) ((i / (float) count) * 100));  
                // Escape early if cancel() is called  
                if (isCancelled()) break;  
            }  
            return totalSize;  
        }  
  
        protected void onProgressUpdate(Integer... progress) {  
            setProgressPercent(progress[0]);  
        }  
  
        protected void onPostExecute(Long result) {  
            showDialog("Downloaded " + result + " bytes");  
        }  
    }  
}
```

- The "Three Dots" in java is called the **Variable Arguments** or **varargs**
- It allows the method to accept zero or multiple arguments
- Varargs are very helpful if you don't know how many arguments you will have to pass in the method

AsyncTask (template) example

```
public class MainActivity extends AppCompatActivity {

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

    public void startTask (View view) {
        // Start the AsyncTask.
        // The AsyncTask has a callback that will update the text view.
        new DownloadFilesTask().execute(url1, url2, url3);
    }

    private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {

        protected Long doInBackground(URL... urls) {
            int count = urls.length;
            long totalSize = 0;
            for (int i = 0; i < count; i++) {
                totalSize += Downloader.downloadFile(urls[i]);
                publishProgress((int) ((i / (float) count) * 100));
                // Escape early if cancel() is called
                if (isCancelled()) break;
            }
            return totalSize;
        }

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

        protected void onPostExecute(Long result) {
            showDialog("Downloaded " + result + " bytes");
        }
    }
}
```



AsyncTask (template) example

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        ...  
    }  
  
    public void startTask (View view) {  
        // Start the AsyncTask.  
        // The AsyncTask has a callback that will update the text view.  
        new DownloadFilesTask().execute(url1, url2, url3);  
    }  
  
    private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {  
  
        protected Long doInBackground(URL... urls) {  
            int count = urls.length;  
            long totalSize = 0;  
            for (int i = 0; i < count; i++) {  
                totalSize += Downloader.downloadFile(urls[i]);  
                publishProgress((int) ((i / (float) count) * 100));  
                // Escape early if cancel() is called  
                if (isCancelled()) break;  
            }  
            return totalSize;  
        }  
  
        protected void onProgressUpdate(Integer... progress) {  
            setProgressPercent(progress[0]);  
        }  
  
        protected void onPostExecute(Long result) {  
            showDialog("Downloaded " + result + " bytes");  
        }  
    }  
}
```

```
public class MainActivity extends AppCompatActivity {

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

    public void startTask (View view) {
        // Start the AsyncTask.
        // The AsyncTask has a callback that will update the text view.
        new DownloadFilesTask().execute(url1, url2, url3);
    }

    private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {

        protected Long doInBackground(URL... urls) {
            int count = urls.length;
            long totalSize = 0;
            for (int i = 0; i < count; i++) {
                totalSize += Downloader.downloadFile(urls[i]);
                publishProgress((int) ((i / (float) count) * 100));
                // Escape early if cancel() is called
                if (isCancelled()) break;
            }
            return totalSize;
        }

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

        protected void onPostExecute(Long result) {
            showDialog("Downloaded " + result + " bytes");
        }
    }
}
```

AsyncTask (template) example

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

```
public void startTask (View view) {  
    // Start the AsyncTask.  
    // The AsyncTask has a callback that will update the text view.  
    new DownloadFilesTask().execute(url1, url2, url3);  
}  
  
}
```

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {  
  
    protected Long doInBackground(URL... urls) {  
        int count = urls.length;  
        long totalSize = 0;  
        for (int i = 0; i < count; i++) {  
            totalSize += Downloader.downloadFile(urls[i]);  
            publishProgress((int) ((i / (float) count) * 100));  
            // Escape early if cancel() is called  
            if (isCancelled()) break;  
        }  
        return totalSize;  
    }  
  
    protected void onProgressUpdate(Integer... progress) {  
        setProgressPercent(progress[0]);  
    }  
  
    protected void onPostExecute(Long result) {  
        showDialog("Downloaded " + result + " bytes");  
    }  
}
```

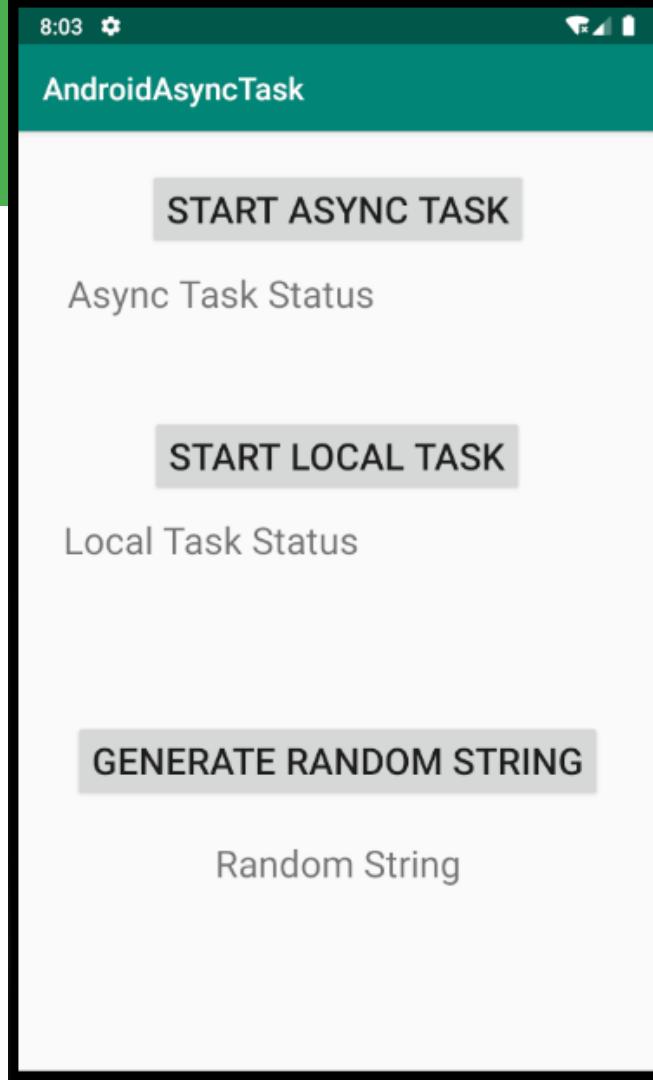
AsyncTask (template) example

AsyncTask Exercise

GOAL: **comparing the app responsiveness** when using or not **AsyncTask** for executing a long computation (e.g., 4 sec long)

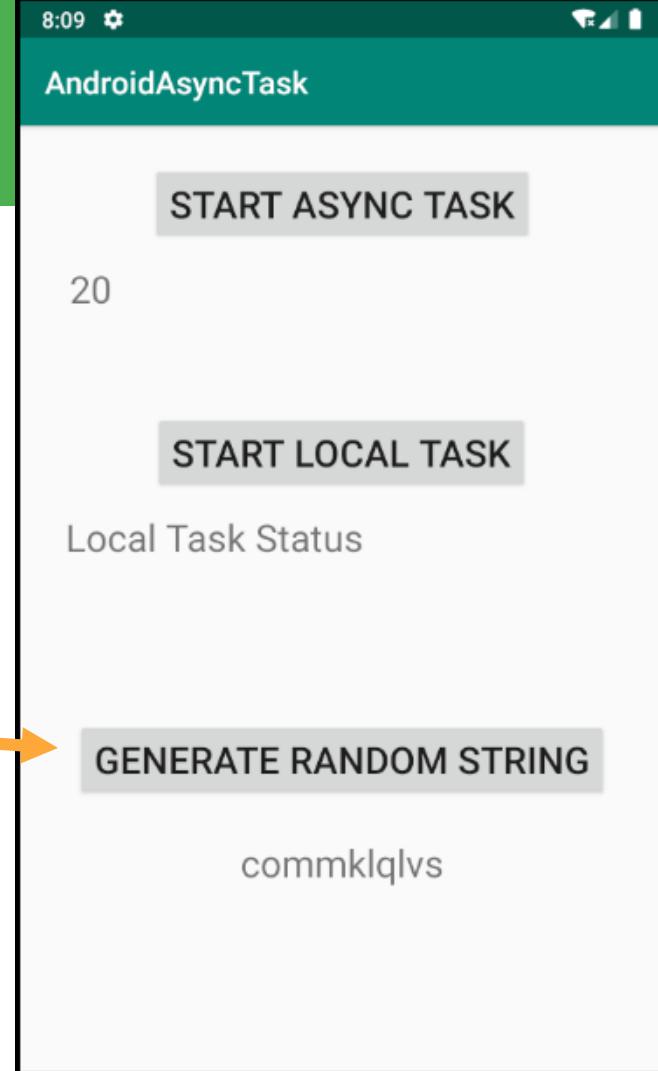
Create:

- A button **starting an AsyncTask**, 4 sec long
- A button **starting a local task**, 4 sec long
 - i.e., executed by the UI thread
- A button **generating random strings**



AsyncTask Exercise

- Visualize the percentage of execution of the Async Task
- When Async Task is executing, try to generate random strings
- Try to do the same with the Local Task. What happens? UI is blocked...



AsyncTask Exercise

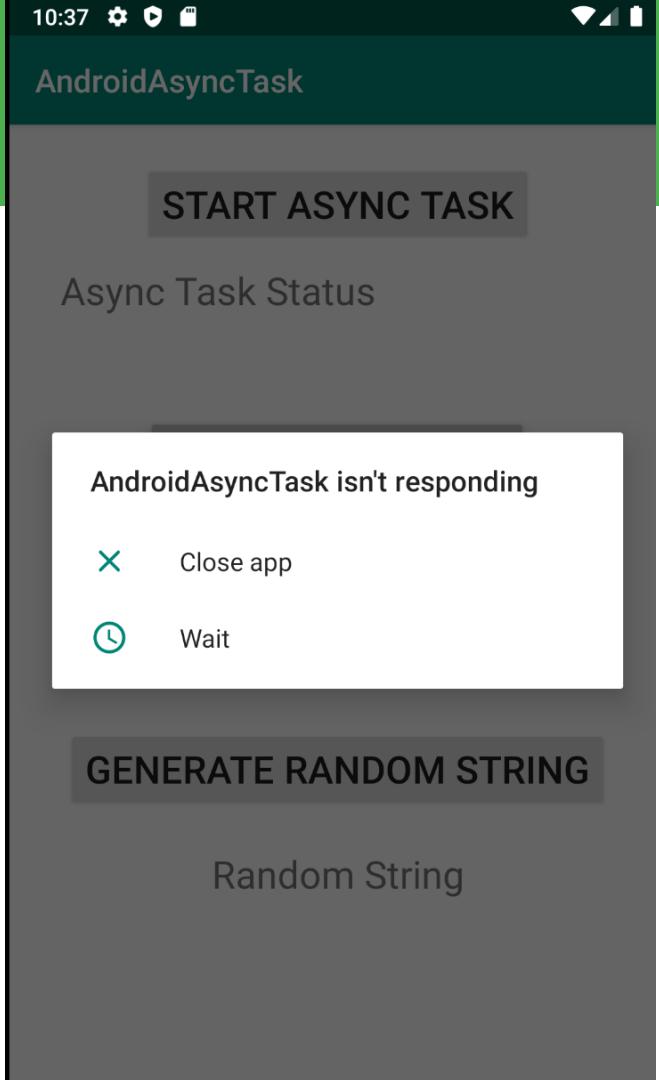
Hints

- Simulate a 4 sec long task using Thread.sleep(millisec)
- Update the UI every 30ms
 - Thus, use a sequence of short Thread.sleep(30)
 - At the end of each Thread.sleep update the UI using publishProgress(percentage)



AsyncTask Exercise

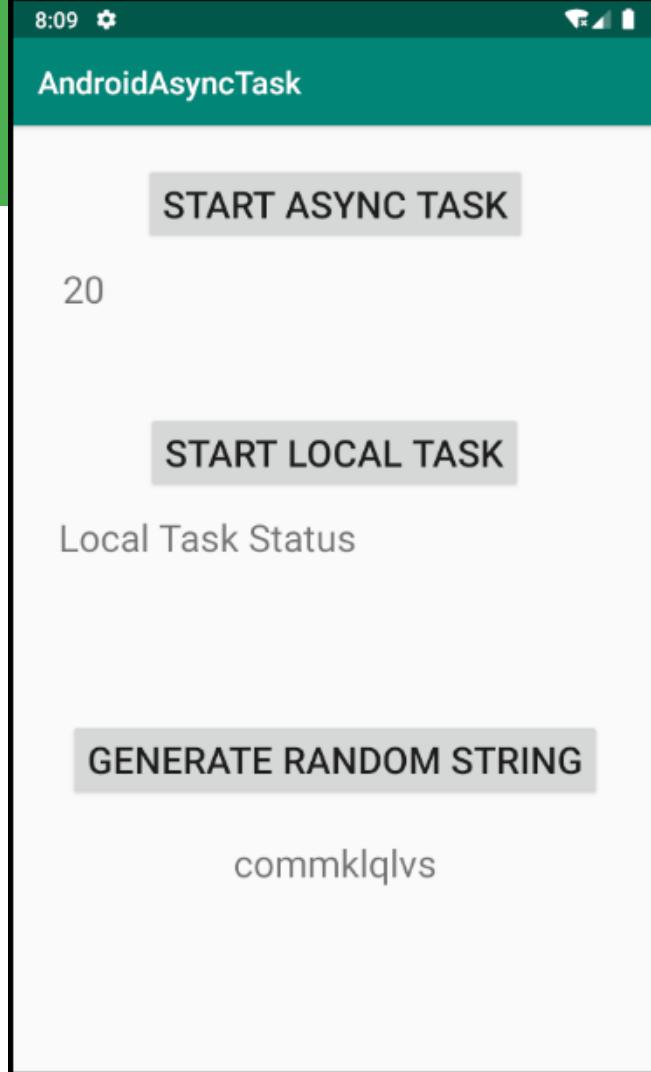
Try to extend the time required for completing the task so that the system displays the [Application Not Responding](#) (ANR) dialog, allowing the user to close the app directly.



AsyncTask Exercise

Considerations

- This is only a simple exercise but the same applies in real cases, for instance when:
 - you have to download/upload a set of files (at the end of each file upload/download you can update the UI and show the progress of the operation)



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
- <https://developer.android.com/reference/android/os/AsyncTask>
- <https://blog.logrocket.com/moving-away-kotlin-asynctask-alternative-solutions/>



This class was deprecated in API level 30.

Use the standard `java.util.concurrent` or [Kotlin concurrency utilities](#) instead.



Concurrent AsyncTasks, AppBar Menu and Android Profiler (in practice)

Contents

- Executing **Multiple AsyncTask Simultaneously**
- Using the *Option Menu* of the App Bar
- Understanding how to analyze *App performances*
 - e.g., using the CPU Profiler

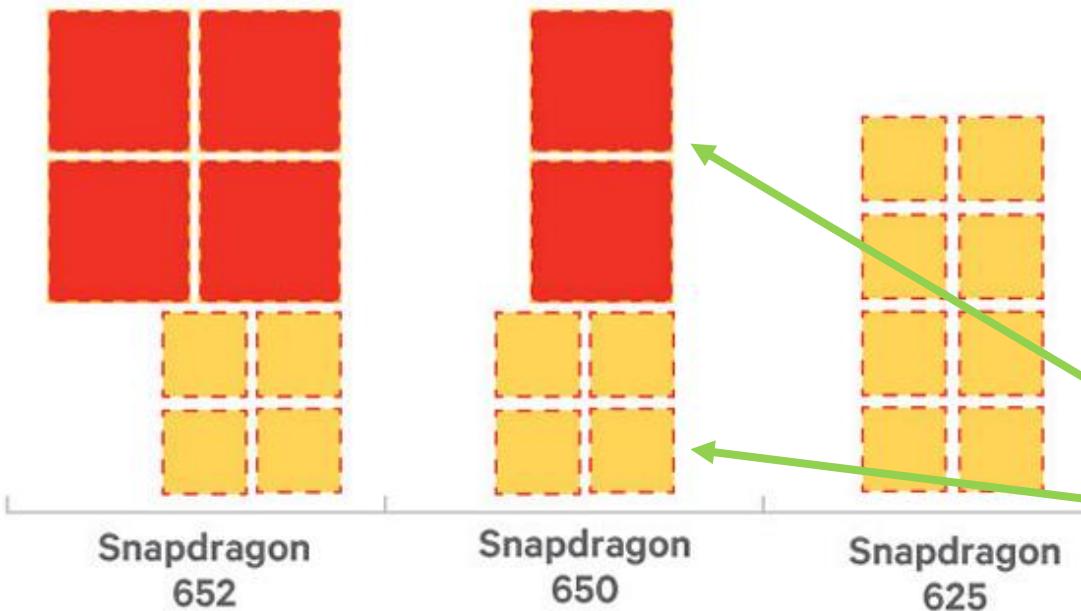
These slides are partially based on the material that Google provides for the course
Android Developer Fundamentals

<https://developer.android.com/courses/fundamentals-training/overview-v2>

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



Mobile Multicore CPU



Today's mobile processors can come with four, six, and eight CPU cores.

To complicate matters further, the architecture and capabilities of those cores might not be the same.

Many CPUs will mix and match high-performance (RED) and low-power cores (YELLOW) in varying combinations.

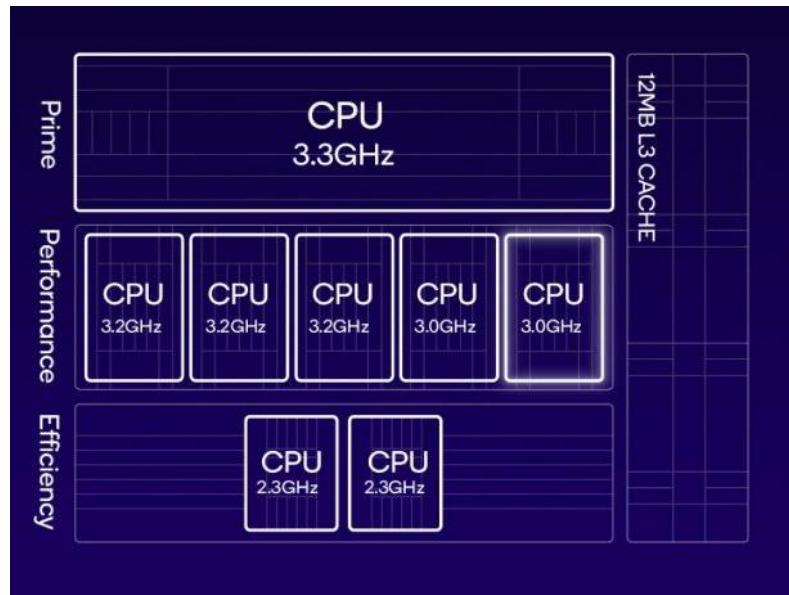
Mobile Multicore CPU

Qualcomm Snapdragon 8 Gen 3

A CPU with 8 cores (3 types):

1 x Prime core at up to 3.3 GHz,
5 x Performance cores at up to 3.2 GHz
2x Efficiency cores at up to 2.3 GHz

<https://www.fonearena.com/blog/408835/qualcomm-snapdragon-8-gen-3-features.html>



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



Contents

- Executing **Multiple AsyncTask Simultaneously**
- Using the ***Option Menu*** of the App Bar
- Understanding how to **analyze App performances**
 - e.g., using the CPU Profiler

These slides are partially based on the material that Google provides for the course
Android Developer Fundamentals

<https://developer.android.com/courses/fundamentals-training/overview-v2>

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



Coins SuperDownloader

We want to **create an app** able to
download the images of the **nine most important coins** for a collector!

Link to high quality coins sample images can be found here:

[https://en.wikipedia.org/wiki/Dollar_coin_\(United_States\)](https://en.wikipedia.org/wiki/Dollar_coin_(United_States))

Simple example but reusable for instance for a Security Cam monitor App



Coins SuperDownloader

We want
downloader
important

Link to
found

<https://en...>



Simple example but reusable for instance for a Security Cam monitor App



Coins SuperDownloader

We want to **create an app** able to
download the images of the **nine most important coins** for a collector!

Link to high quality coins sample images can be found here:

[https://en.wikipedia.org/wiki/Dollar_coin_\(United_States\)](https://en.wikipedia.org/wiki/Dollar_coin_(United_States))

Simple example but reusable for instance for a Security Cam monitor App



Coins SuperDownloader

How can we **download** the coins' images?

HINT: create an **AsyncTask** for downloading a single image and then call it for each image

i.e., for each URL

See for instance:

<https://stackoverflow.com/a/41572119>

Declare the appropriate permissions...

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



```
private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {  
    ImageView bmImage;  
  
    public DownloadImageTask(ImageView bmImage) {  
        this.bmImage = bmImage;  
    }  
    protected Bitmap doInBackground(String... urls) {  
        String urldisplay = urls[0];  
        Bitmap mIcon = null;  
        try {  
            InputStream in = new java.net.URL(urldisplay).openStream();  
            mIcon = BitmapFactory.decodeStream(in);  
        } catch (Exception e) {  
            Log.e("Error", e.getMessage());  
            e.printStackTrace();  
        }  
        return mIcon;  
    }  
  
    protected void onPostExecute(Bitmap result) {  
        bmImage.setImageBitmap(result);  
    }  
}
```

Declare the appropriate permissions...

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



```
private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {  
    ImageView bmImage;  
  
    public DownloadImageTask(ImageView bmImage) {  
        this.bmImage = bmImage;  
    }  
    protected Bitmap doInBackground(String... urls) {  
        String urldisplay = urls[0];  
        Bitmap mIcon = null;  
        try {  
            InputStream in = new java.net.URL(urldisplay).openStream();  
            mIcon = BitmapFactory.decodeStream(in);  
        } catch (Exception e) {  
            Log.e("Error", e.getMessage());  
            e.printStackTrace();  
        }  
        return mIcon;  
    }  
  
    protected void onPostExecute(Bitmap result) {  
        bmImage.setImageBitmap(result);  
    }  
}
```



Declare the appropriate permissions...

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

```
private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {  
    ImageView bmImage;  
  
    public DownloadImageTask(ImageView bmImage) {  
        this.bmImage = bmImage;  
    }  
    protected Bitmap doInBackground(String... urls) {  
        String urldisplay = urls[0];  
        Bitmap mIcon = null;  
        try {  
            InputStream in = new java.net.URL(urldisplay).openStream();  
            mIcon = BitmapFactory.decodeStream(in);  
        } catch (Exception e) {  
            Log.e("Error", e.getMessage());  
            e.printStackTrace();  
        }  
        return mIcon;  
    }  
    protected void onPostExecute(Bitmap result) {  
        bmImage.setImageBitmap(result);  
    }  
}
```

Declare the appropriate permission

```
<uses-permission android:n
```



Main Thread (UI Thread)

onPreExecute()

onProgressUpdate()

onPostExecute()

Worker Thread

publishProgress()

doInBackground()

```
private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {  
    ImageView bmImage;  
  
    public DownloadImageTask(ImageView bmImage) {  
        this.bmImage = bmImage;  
    }  
    protected Bitmap doInBackground(String... urls) {  
        String urldisplay = urls[0];  
        Bitmap mIcon = null;  
        try {  
            InputStream in = new java.net.URL(urldisplay).openStream();  
            mIcon = BitmapFactory.decodeStream(in);  
        } catch (Exception e) {  
            Log.e("Error", e.getMessage());  
            e.printStackTrace();  
        }  
        return mIcon;  
    }  
  
    protected void onPostExecute(Bitmap result) {  
        bmImage.setImageBitmap(result);  
    }  
}
```

12:23

Main Thread (UI Thread)

onPreExecute()

onProgressUpdate()

onPostExecute()

Worker Thread

publishProgress()

doInBackground()



Declare the appropriate permissions...

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



Coins SuperDownloader

What happens if an **image is particularly slow** to download?

- it blocks also the download of all the subsequent images...

WHY? With the standard way of running AsyncTask, they are executed one by one...

Probably better to parallelize the process...



Executing Multiple AsyncTask

When multiple Async Task have to be executed:

- **execute(...)** => serial (one by one)
- **executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR,...)**
=> parallel (execute simultaneously)

```
// The AsyncTask has a callback that will update the text view.  
SimpleAsyncTask st = new SimpleAsyncTask();
```

```
// only one AsincTask in execution  
//st.execute(params...);
```

```
// multiple AsincTask(s) in execution  
st.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR,params...);
```

<https://stackoverflow.com/questions/29937556/asynctask-execute-or-executeonexecutor>

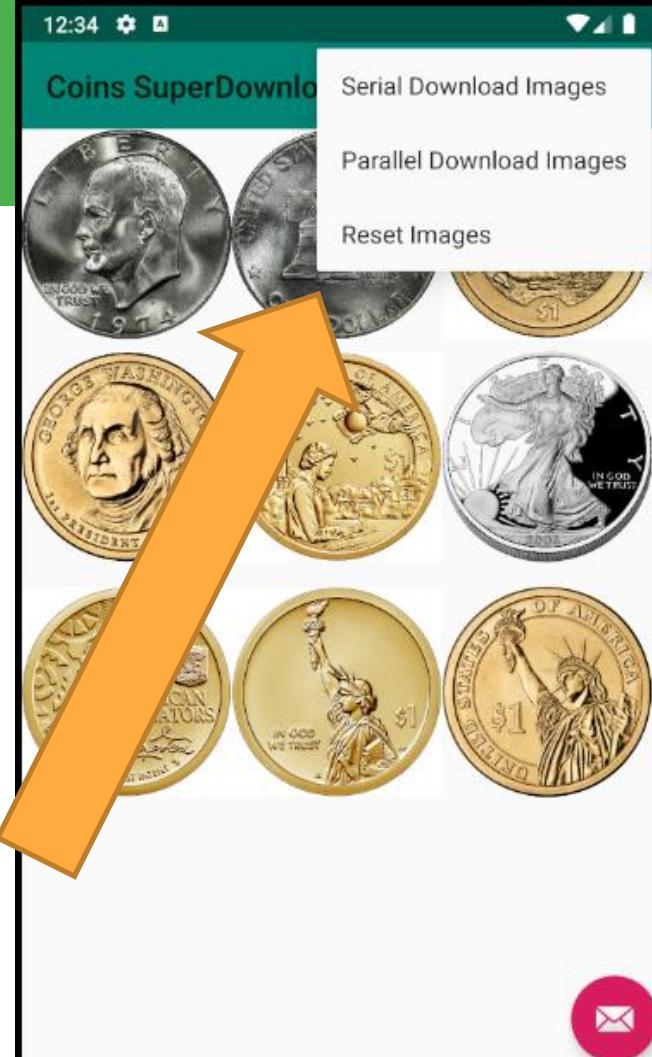
Coins SuperDownloader

We want to **take advantage** of:

- the ***multiple cores*** available in our smartphones
- the ***fast internet connection*** supporting multiple downloads at the same time
 - Warning: this app will download images for several MBs. Thus use an appropriate connection (e.g. WiFi)

One the GUI:

- Allow the user to **select how to download** the images of the coins! One by one or in parallel...
- Allow the user to **reset all the images**



Choose your project

Phone and Tablet

Wear OS

TV

Android Auto

Android Things

Use a Basic Activity instead of an Empty Activity....

Add No Activity



Basic Activity



Basic Activity

Creates a new basic activity with an app bar.

Previous

Next

Cancel

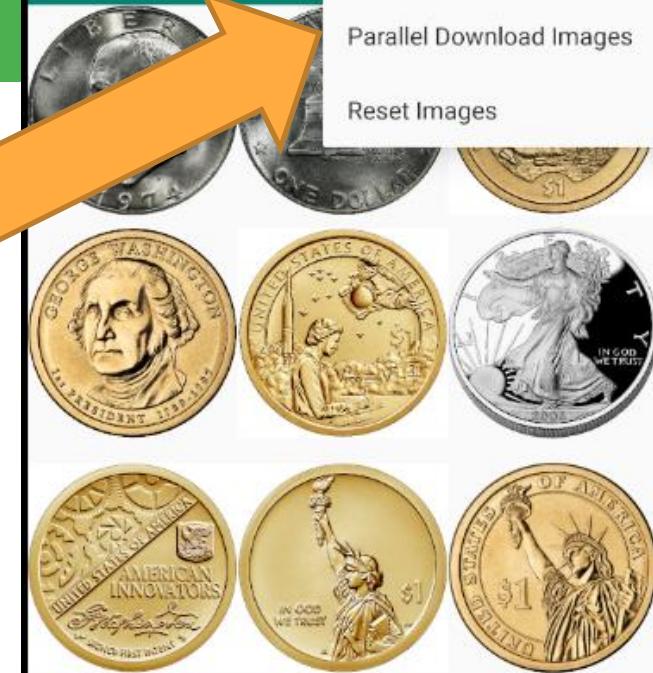
Finish

Coins SuperDownload

Serial Download Images

Parallel Download Images

Reset Images



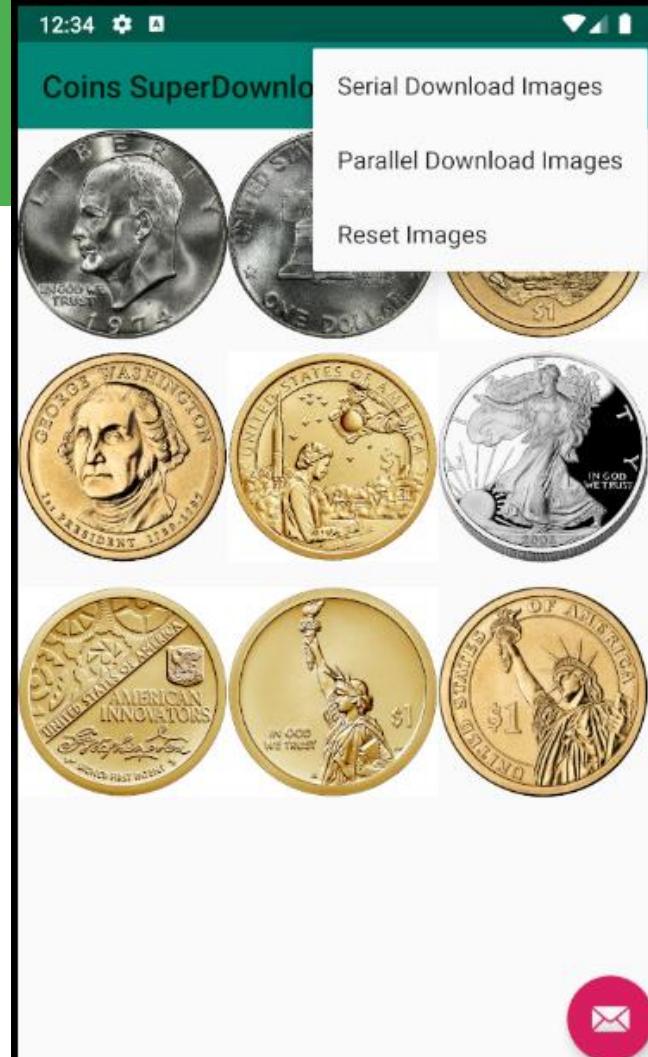
If not available in the latest version of
Android studio try to find how to add it
(or in the worst case use simple buttons)

Coins SuperDownloader

Evaluation: Is the **parallel version** of the image downloader **faster** than the **serial one**?

Maybe... The bottleneck (for perceiving an actual boost in performance) is probably the network connection...

But we want to experiment the advantage of our multicores processors...



Coins SuperDownloader

Thus: insert a **post processing step** for the images

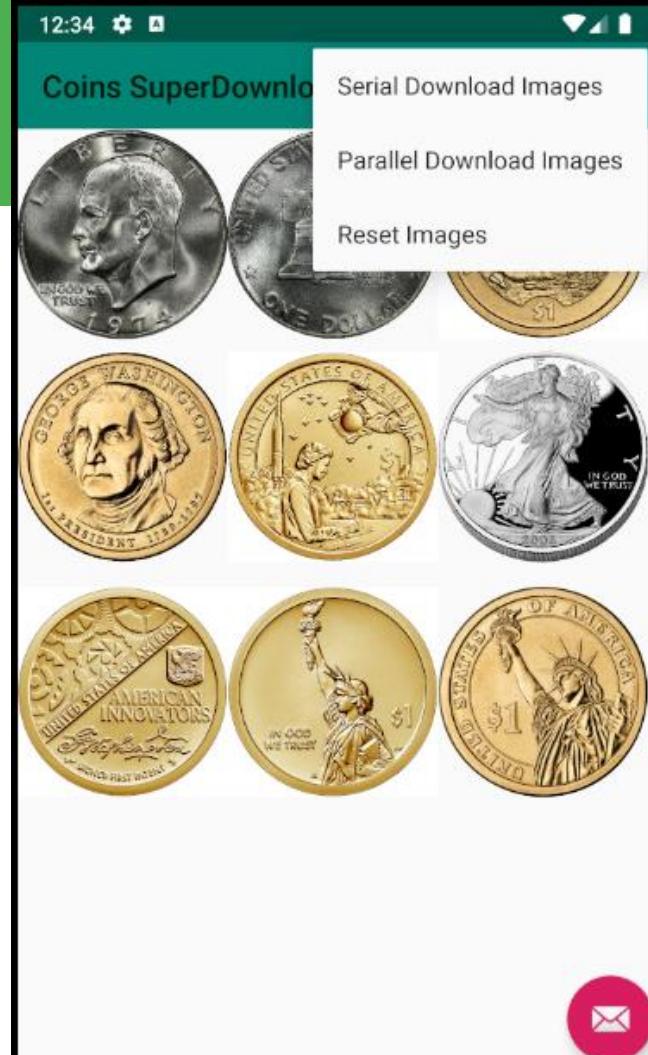
After downloading them, in the same task, **apply a transformation to each image**, multiple times

- e.g., rotate it of 90° for 40 times...

<https://stackoverflow.com/a/14645289>

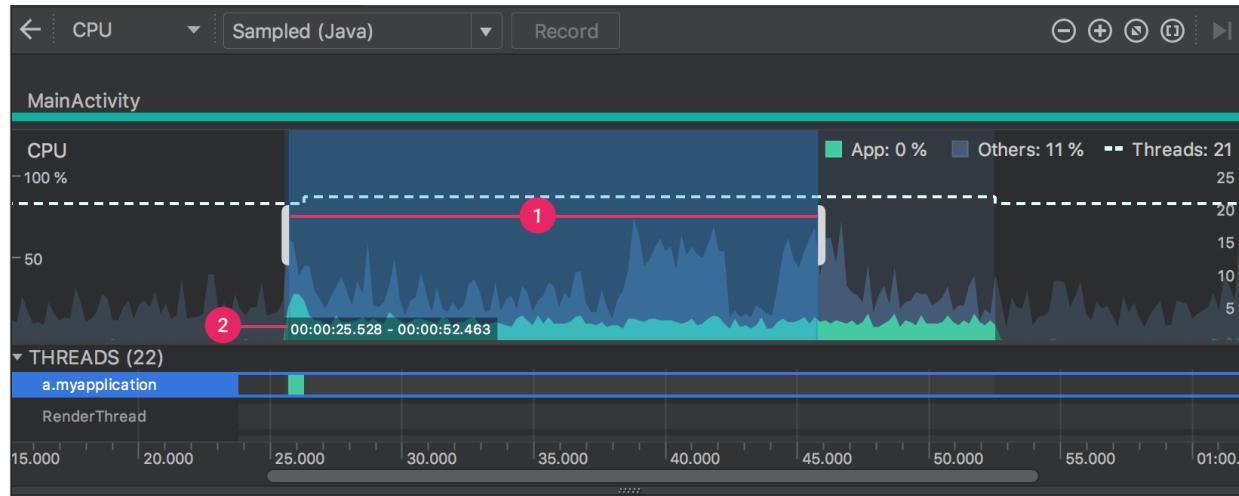
In this way, we can evaluate the difference in performance between serial and parallel tasks when heavy computations are required...

This is only an example, on a real app we can run a real complex algorithm on each image such as a **face recognition algorithm** (in the security cam app mentioned before)...



Coins SuperDownloader

Now the **difference in performances** among the **two approaches** are clearly visible. But **how can we precisely analyses** them?



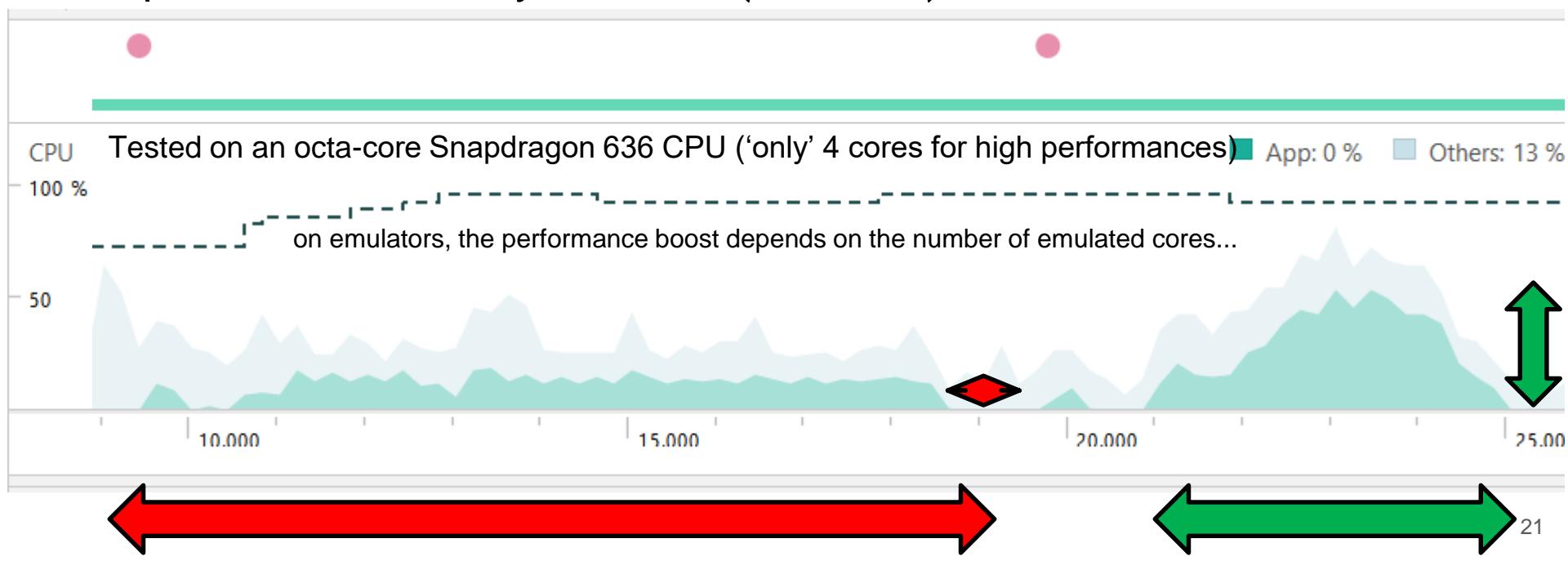
Inspect CPU activity with CPU Profiler

<https://developer.android.com/studio/profile/cpu-profiler>

Coins SuperDownloader

Serial vs **Parallel** execution CPU time (X-axis) vs CPU usage (Y-axis)

The parallel version is by far faster (less time) and uses more the CPU



Assignment

1. Develop the Coin SuperDownloader app as described in this lesson
 - a. Both serial and parallel execution + reset button
2. Manage the fact that a remote image file could be not available and visualize an error image in that case
 - a. for example <https://it.wikipedia.org/wiki/File:Error.svg>
3. Analyze the execution of the serial and parallel version
 - a. On the emulator and on a real device (if you have one)
 - b. Are the obtained results reasonable in relation to the HW of the device?
 - C. Write a one/two pages report with the screenshot of the profiler and an explanation of the results



Receiving and Using Values from Sensors (and plotting them in charts...)

Contents

- How to obtain **the list of sensors** available in the device
- How to **read the values** provided by a sensor
- How to **plot charts of the provided values**

These slides are partially based on the material that Google provides:

Sensors Overview

https://developer.android.com/guide/topics/sensors/sensors_overview

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



Sensors in Android

- Most Android-powered devices have ***built-in sensors*** that measure:
 - motion, orientation, and various environmental conditions
- These **sensors** are **capable** of ***providing raw data*** with **high precision** and accuracy
 - Such data can be exploited for a variety of applications....



Accelerometer
Sensor



Light
Sensor



Orientation
Sensor



Proximity
Sensor



Temperature
Sensor



Gyroscope
Sensor



Sound
Sensor

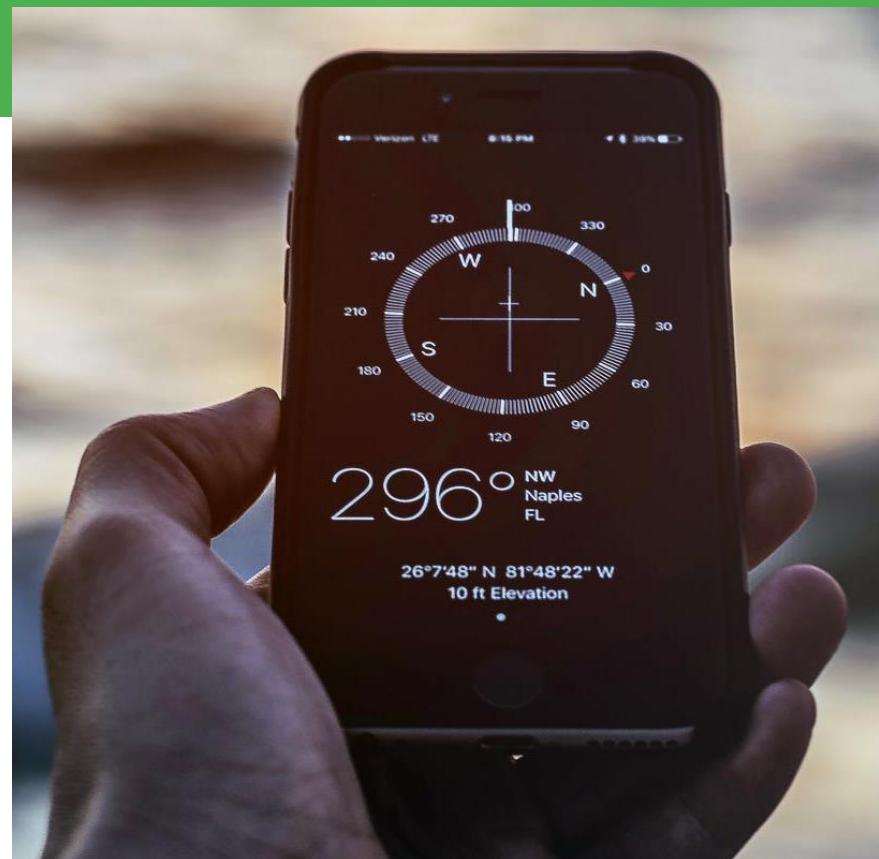


Magnetic
Sensor



Pressure
Sensor

Sensors in Android



Pressure-Pair-Based Floor Localization System Using Barometric Sensors on Smartphones
<https://www.mdpi.com/1424-8220/19/16/3622>

Sensors in Android

The Android platform supports **three broad categories** of **sensors**:

- **Motion sensors:** measure *acceleration* forces and *rotational* forces along three axes
 - this category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors
- **Environmental sensors:** measure various *environmental parameters*, such as ambient air temperature and pressure, illumination, and humidity
 - this category includes barometers, photometers, and thermometers
- **Position sensors:** measure the **physical position** of a device
 - this category includes orientation sensors

Android Sensor Framework

Android sensor framework **allows to access sensors** available on the device and **acquire raw sensor data**.

For example, you can use the sensor framework to do the following:

- Determine **which sensors are available** on a device
- Determine **individual sensor's capabilities**, such as its maximum range, manufacturer, power requirements, and resolution
- **Acquire raw sensor data** and define the minimum rate at which you acquire sensor data
- **Register and unregister sensor event** listeners that **monitor sensor changes**

Examples of Sensors (1)

Sensor	Type	Description	Common Uses
TYPE_ACCELEROMETER	Hardware	Measures the acceleration force in m/s ² that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.).
TYPE_AMBIENT_TEMPERATURE	Hardware	Measures the ambient room temperature in degrees Celsius (°C). See note below.	Monitoring air temperatures.
TYPE_GRAVITY	Software or Hardware	Measures the force of gravity in m/s ² that is applied to a device on all three physical axes (x, y, z).	Motion detection (shake, tilt, etc.).
TYPE_GYROSCOPE	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.).
TYPE_LIGHT	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
TYPE_LINEAR_ACCELERATION	Software or Hardware	Measures the acceleration force in m/s ² that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Monitoring acceleration along a single axis.
TYPE_MAGNETIC_FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in µT.	Creating a compass.

Examples of Sensors (2)

Sensor	Type	Description	Common Uses
<code>TYPE_ORIENTATION</code>	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the <code>getRotationMatrix()</code> method.	Determining device position.
<code>TYPE_PRESSURE</code>	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
<code>TYPE_PROXIMITY</code>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
<code>TYPE_RELATIVE_HUMIDITY</code>	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.
<code>TYPE_ROTATION_VECTOR</code>	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
<code>TYPE_TEMPERATURE</code>	Hardware	Measures the temperature of the device in degrees Celsius (°C). This sensor implementation varies across devices and this sensor was replaced with the <code>TYPE_AMBIENT_TEMPERATURE</code> sensor in API Level 14	Monitoring temperatures.

Steps for using sensors

A typical application based on sensor-related APIs requires to perform two basic tasks:

- **Identifying sensors and sensor capabilities:** (at runtime), useful if the application has features that rely on specific sensor types or capabilities
 - For example, you may want *to identify all the sensors that are present* on a device and *disable any application features that rely on sensors that are not present*
- **Monitor sensor events:** to acquire raw sensor data
 - A sensor event occurs **every time a sensor detects a change** in the parameters it is measuring
 - A sensor event provides you with four pieces of information:
 - the **name of the sensor** that triggered the event
 - the **timestamp** for the event
 - the **accuracy** of the event
 - the **raw sensor data** that triggered the event

Steps for using sensors

A typical application based on sensor-related APIs requires to perform two basic tasks:

- **Identifying sensors and sensor capabilities:** (at runtime), useful if the application has features that rely on specific sensor types or capabilities
 - For example, you may want *to identify all the sensors that are present* on a device and *disable any application features that rely on sensors that are not present*
- **Monitor sensor events:** to acquire raw sensor data
 - A **sensor event** occurs **every time a sensor detects a change** in the parameters it is measuring
 - A sensor event provides you with four pieces of information:
 - the **name of the sensor** that triggered the event
 - the **timestamp** for the event
 - the **accuracy** of the event
 - the **raw sensor data** that triggered the event

Identifying the Available Sensors

To identify the sensors that are on a device you first need to get a reference to the sensor service. To do this, you create an instance of the `SensorManager` class by calling the `getSystemService()` method and passing in the `SENSOR_SERVICE` argument. For example:

KOTLIN JAVA

```
private SensorManager sensorManager;  
...  
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```



Next, you can get a listing of every sensor on a device by calling the `getSensorList()` method and using the `TYPE_ALL` constant. For example:

KOTLIN JAVA

```
List<Sensor> deviceSensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
```



Identifying the Available Sensors

To identify the sensors that are on a device you first need to get a reference to the sensor service. To do this, you create an instance of the `SensorManager` class by calling the `getSystemService()` method and passing in the `SENSOR_SERVICE` argument. For example:

KOTLIN JAVA

```
private SensorManager sensorManager;  
...  
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```



Next, you can get a listing of every sensor on a device by calling the `getSensorList()` method and using the `TYPE_ALL` constant. For example:

KOTLIN JAVA

```
List<Sensor> deviceSensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
```



Finding a specific sensor

You can also determine whether a specific type of sensor exists on a device by using the `getDefaultSensor()` method and passing in the type constant for a specific sensor. If a device has more than one sensor of a given type, one of the sensors must be designated as the default sensor. If a default sensor does not exist for a given type of sensor, the method call returns null, which means the device does not have that type of sensor. For example, the following code checks whether there's a magnetometer on a device:

KOTLIN

JAVA

```
private SensorManager sensorManager;  
...  
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);  
if (sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null){  
    // Success! There's a magnetometer.  
} else {  
    // Failure! No magnetometer.  
}
```



Capabilities and attributes of sensors

getResolution

Added in API level 3

```
public float getResolution ()
```



Returns

float

resolution of the sensor in the sensor's unit.

getMaximumRange

Added in API level 3

```
public float getMaximumRange ()
```



Returns

float

maximum range of the sensor in the sensor's unit.

Monitoring Sensor Events

To monitor raw sensor data you need to **implement two callback methods** that are exposed through the `SensorEventListener` interface: `onSensorChanged()` and `onAccuracyChanged()`. The Android system calls these methods whenever the *following events* occur:

- **A sensor reports a new value.** In this case the system invokes the `onSensorChanged()` method, providing you with a `SensorEvent` object
 - A `SensorEvent` object contains information about the new sensor data, including: the accuracy of the data, the sensor that generated the data, the timestamp at which the data was generated, and the new data that the sensor recorded.
- **A sensor's accuracy changes.** In this case the system invokes the `onAccuracyChanged()` method, providing you with a reference to the `Sensor` object that changed and the new accuracy of the sensor

Monitoring Sensor Events

To monitor raw sensor data you need to **implement two callback methods** that are exposed through the SensorEventListener interface: `onSensorChanged()` and `onAccuracyChanged()`. The Android system calls these methods whenever the ***following events*** occur:

- **A sensor reports a new value.** In this case the system invokes the `onSensorChanged()` method, providing you with a `SensorEvent` object
 - A `SensorEvent` object contains information about the new sensor data, including: the accuracy of the data, the sensor that generated the data, the timestamp at which the data was generated, and the new data that the sensor recorded.
- **A sensor's accuracy changes.** In this case the system invokes the `onAccuracyChanged()` method, providing you with a reference to the `Sensor` object that changed and the new accuracy of the sensor

Monitoring Sensor Events

To monitor raw sensor data you need to **implement two callback methods** that are exposed through the SensorEventListener interface: `onSensorChanged()` and `onAccuracyChanged()`. The Android system calls these methods whenever the ***following events*** occur:

- **A sensor reports a new value.** In this case the system invokes the `onSensorChanged()` method, providing you with a `SensorEvent` object
 - A `SensorEvent` object contains information about the new sensor data, including: the accuracy of the data, the sensor that generated the data, the timestamp at which the data was generated, and the new data that the sensor recorded.
- **A sensor's accuracy changes.** In this case the system invokes the `onAccuracyChanged()` method, providing you with a reference to the `Sensor` object that changed and the new accuracy of the sensor

onSensorChanged() example

The following code shows **how to use the `onSensorChanged()` method** to **monitor data from the light sensor**.

```
public class SensorActivity extends Activity implements SensorEventListener {
    private SensorManager sensorManager;
    private Sensor mLight;

    @Override
    public final void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        mLight = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
    }

    @Override
    public final void onAccuracyChanged(Sensor sensor, int accuracy) {
        // Do something here if sensor accuracy changes.
    }
}
```



onSensorChanged() example

```
@Override  
public final void onSensorChanged(SensorEvent event) {  
    // The light sensor returns a single value.  
    // Many sensors return 3 values, one for each axis.  
    float lux = event.values[0];  
    // Do something with this sensor value.  
}  
  
@Override  
protected void onResume() {  
    super.onResume();  
    sensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL);  
}  
  
@Override  
protected void onPause() {  
    super.onPause();  
    sensorManager.unregisterListener(this);  
}
```

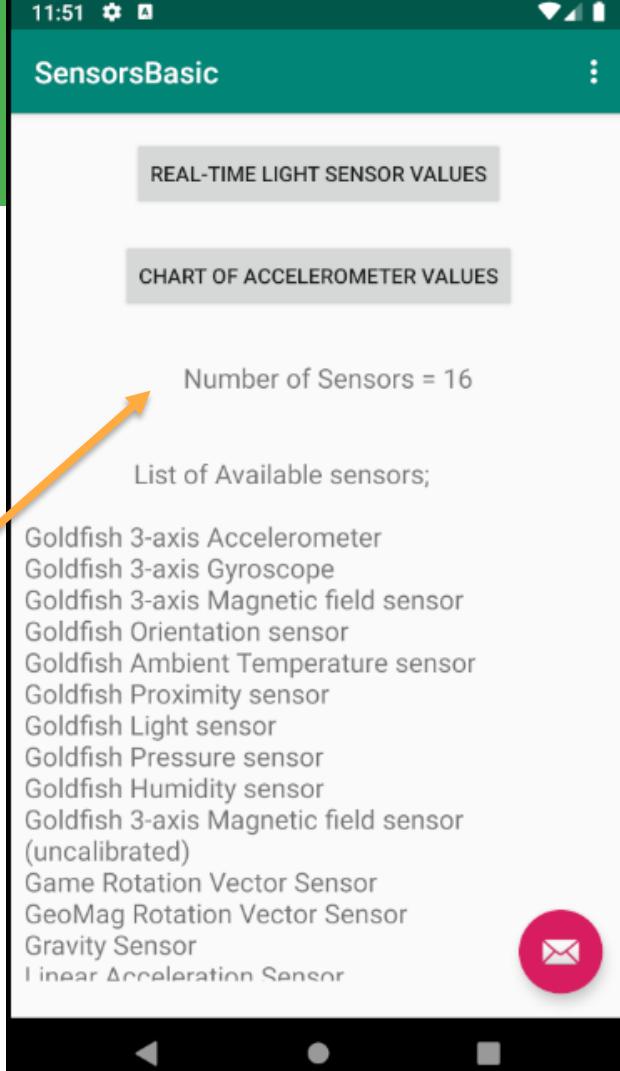
Sensors App

Build a simple app to learn how to interact with sensors and plot the received data.

The functionalities of the app are the following 4:

1. Determine the number of sensors available in the used device
2. List the name of each sensor

Main Activity



Sensors App

Light Sensor Activity

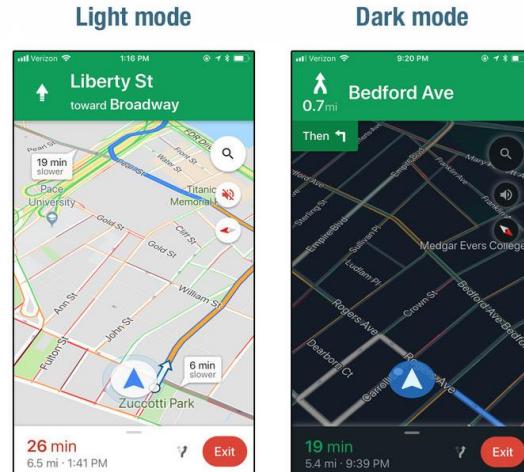
3. Show the value recorded by the light sensor updated in real time
 - try this on your smartphone

Light Sensor Values Update in Real Time:

0.0

Ambient Light Sensor usage example:

If you have used Google Map and driven at night or under a dark tunnel, the app will adjust the display of the map accordingly and show you a "night" version. This simple change makes it much easier to see depending on the current ambient light.



Adapting user interface configuration...



Sensors App

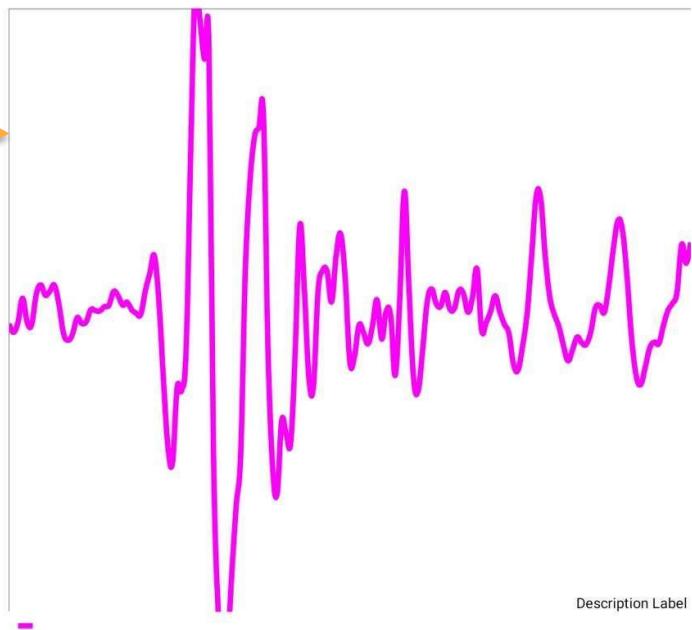
4. Show a chart (update in real time) of the acceleration recorded by the sensor

- **Basic:** Visualize the accelerations on the X axis (`event.values[0]`)
- **Suggested:** Visualize the total acceleration described by a vector consisting of the x, y, and z components that you have from the accelerometer. The magnitude of the acceleration is the square root of the sum of the squares of each component

$$|\vec{a}| = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

SensorsBasic

Accelerometer Chart



Description Label

Sensors App

For implementing the chart use (for instance) the MPAndroidChart library:

<https://github.com/PhilJay/MPAndroidChart>

An example of usage can be found here:

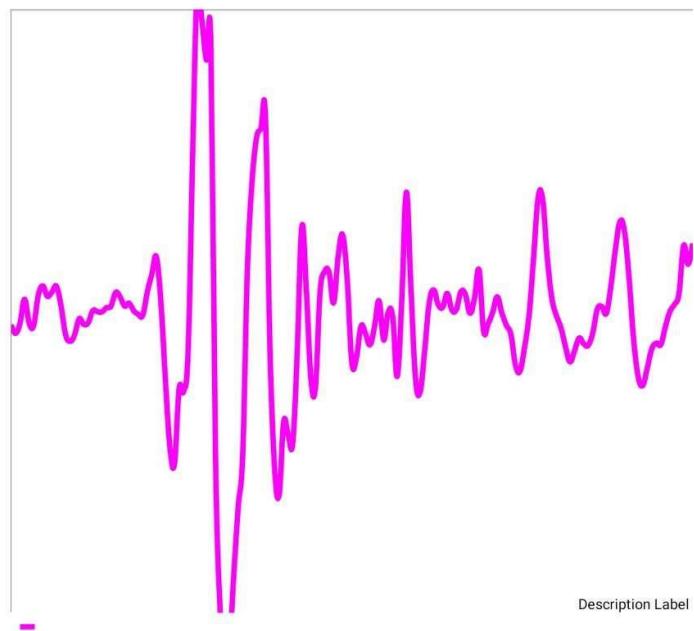
<https://github.com/laxmimerit/Real-Time-Accelerometer-Data-Plot>

12:53 ⌂ ...

76%

SensorsBasic

Accelerometer Chart



Sensors App

How can I solve this error? (ERROR: Gradle DSL method not found: 'implementation()')

<https://stackoverflow.com/questions/57753727/how-can-i-solve-this-error-error-gradle-dsl-method-not-found-implementation>

Accelerometers (and other sensors) usage example:

The use of smartphone for gait analysis

<https://ieeexplore.ieee.org/document/7942756>

On gait recognition with smartphone accelerometer

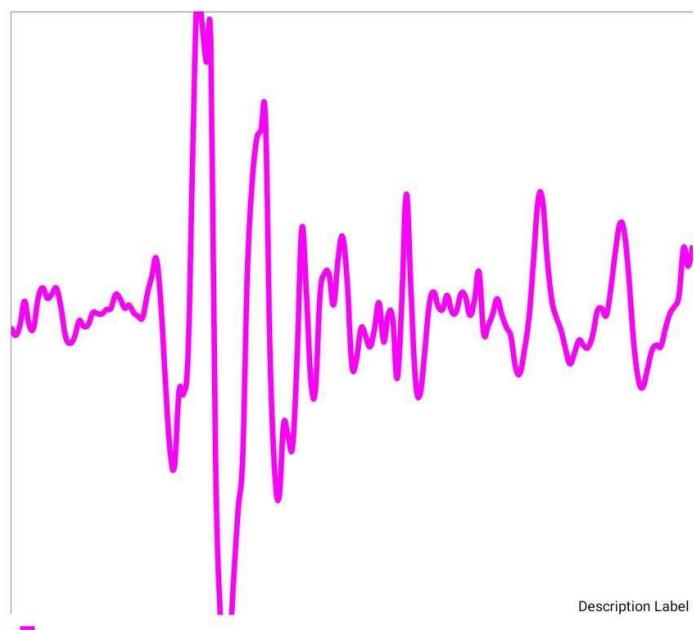
<https://ieeexplore.ieee.org/document/7181946>

12:53 ⌂ ...

76%

SensorsBasic

Accelerometer Chart



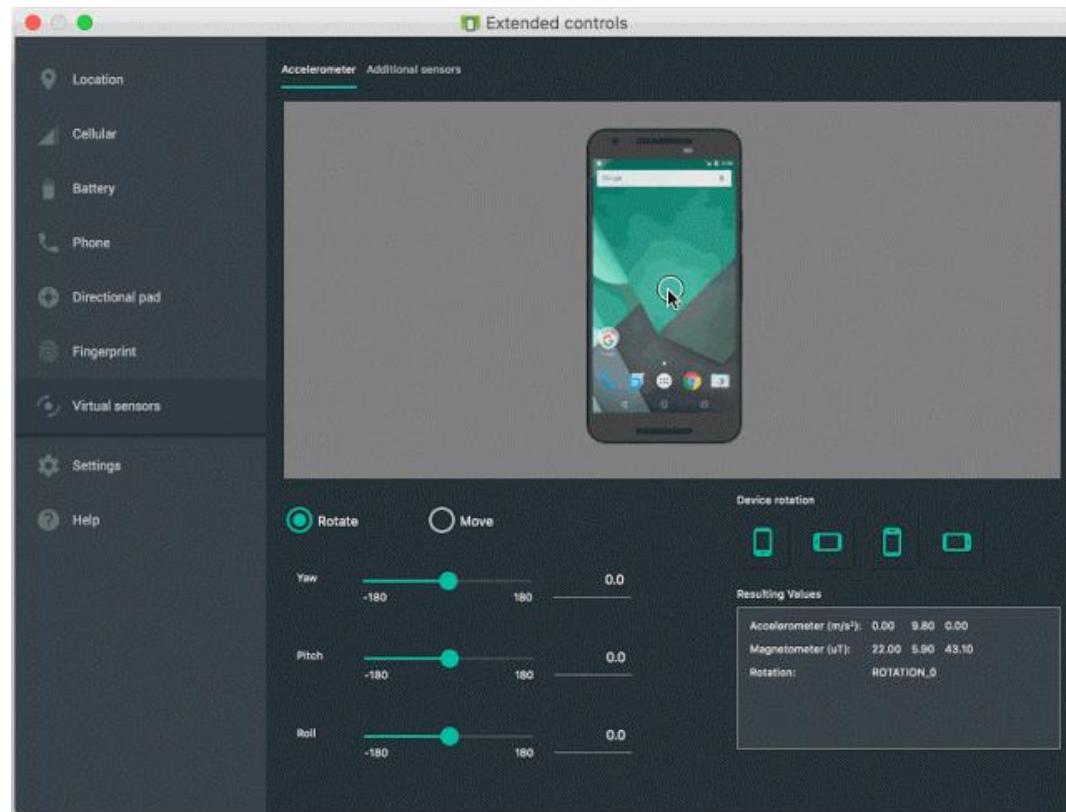
Description Label

Test with the Android Emulator

The Android Emulator includes a set of virtual sensor controls that allow you to test sensors such as accelerometer, ambient temperature, magnetometer, proximity, light, and more.

See:

https://developer.android.com/guide/topics/sensors/sensors_overview





Databases for Mobile Apps

Contents

- Why using a DB
- ***Local*** and ***Remote*** Databases
- **SQLite** and **Room**
- Using an Android DB in practice

These slides are partially based on the material that Google provides for the course
Android Developer Fundamentals

<https://developer.android.com/courses/fundamentals-training/overview-v2>

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



Databases for Android

In an Android app it is possible to use **two different kinds of DBs**

Local: if you want your **data to be device specific**, i.e. stay local, then you should go with a local DB. A local DB might be a preferable choice over remote DB when you have to **simply save data locally** and **avoid the server requests**

e.g. personal Addressbook app



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



Databases for Android

In an Android app it is possible to use **two different kinds of DBs**

Remote: if the **app needs to synchronize data across all its users or involve live data being fed to a user's request** then you need to use a **remote database**

e.g. weather app, almost all complex apps...

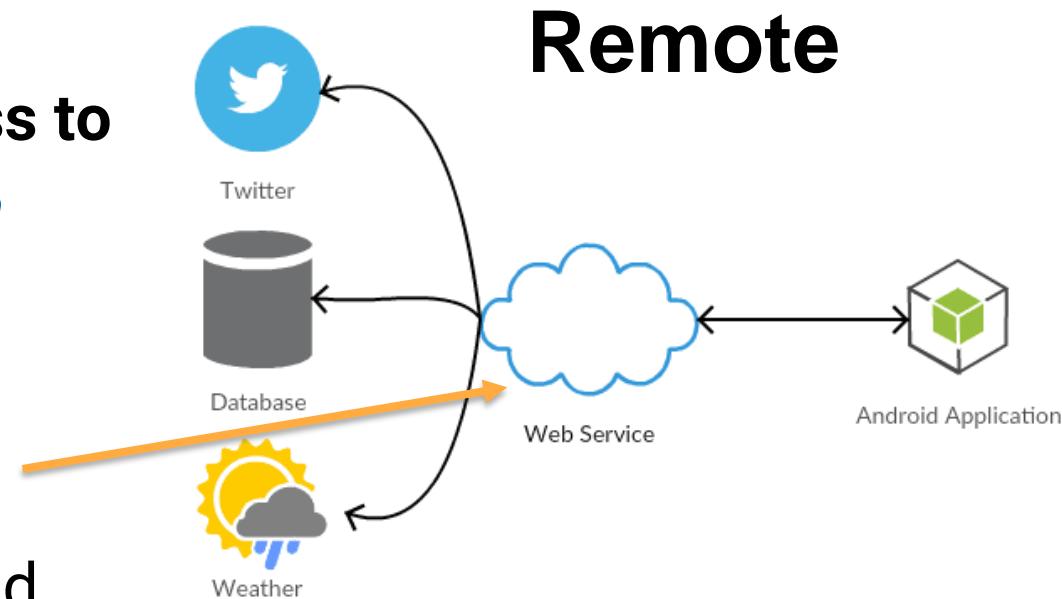


This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



Databases for Android (Remote)

- an **Android application** **should not directly access to** a *database deployed on a remote server*
- **best practices** require to **implement a web service** between the database and the **Android application**

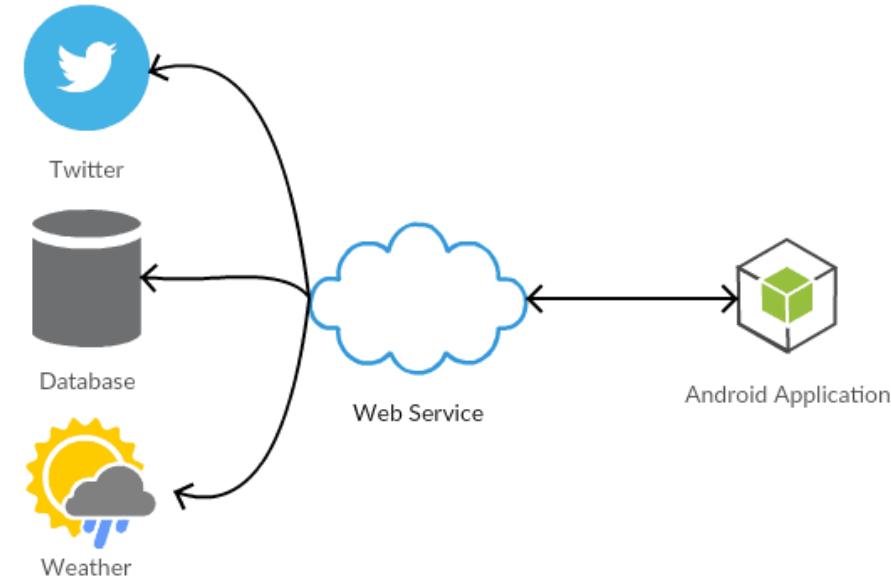


This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



Databases for Android (Remote)

- **Having a web service layer**
 - reduces the **complexity** of the Android application
 - reduces the **dependency** on database specific operations
- The **problem of accessing a client/server database** like MySQL from an Android application **can be defined** as the **problem of consuming a web service** hosted somewhere

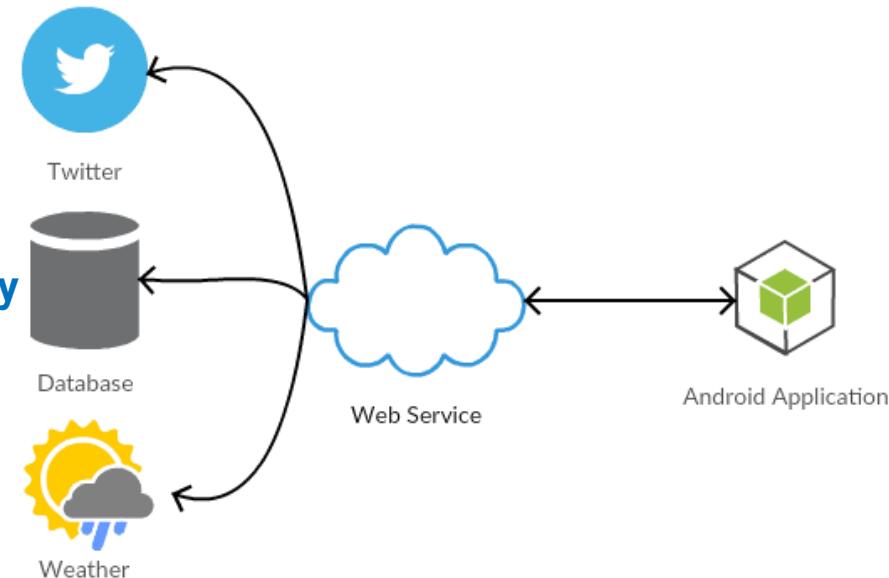


This work is licensed under a [Creative Commons Attribution 4.0 International License](#).



Databases for Android (Remote)

- **Don't need to care about what is behind the web service**
 - it may be a MySQL database, MongoDB database, a Social Media network or even a Weather Network API
- **What you need** is the **API endpoints exposed by the web service**



Examples:

How to connect Android with PHP, MySQL

https://www.tutorialspoint.com/android/android_php_mysql.htm

Access MySQL from Android through RESTful Web Service

<https://phpspot.com/php/php-mysql-rest-api-for-android/>

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



Databases for Android (Local)

On the other hand, if **all the information should be stored on a mobile device**, there are only some different options including **SQLite** and **Realm**

- **SQLite** is a *opensource SQL database* that **stores data** to a **text file on a device**. Android comes in with **built in SQLite database implementation**. SQLite supports all the relational database features

<https://developer.android.com/training/data-storage/sqlite>

Local

<https://www.sqlite.org/whentouse.html>

https://www.tutorialspoint.com/android/android_sqlite_database.htm

- **Realm** is a *NoSQL* mobile database that runs directly inside phones, tablets or wearables

<https://github.com/realm/realm-java>

<https://www.mongodb.com/blog/post/realm-now-part-atlas-platform>

<https://www.mongodb.com/docs/realm/sdk/java/>

<https://www.mongodb.com/docs/realm/sdk/kotlin/>

SQLite

- The inbuilt **SQLite core library is within** the Android OS
- It will **handle CRUD** (Create, Read, Update and Delete) ***operations*** required for a database
- Java classes and interfaces for SQLite are provided by the android.database
- SQLite maintains an effective database management system

SQLite

But this **conventional method** has its **own disadvantages**

- You have to **write long repetitive code**, which will be **time-consuming** as well as **prone to mistakes**
- It is very **difficult to manage SQL queries** for a complex relational database

```
private static final String SQL_CREATE_ENTRIES =
    "CREATE TABLE " + FeedEntry.TABLE_NAME + " (" +
    FeedEntry._ID + " INTEGER PRIMARY KEY," +
    FeedEntry.COLUMN_NAME_TITLE + " TEXT," +
    FeedEntry.COLUMN_NAME_SUBTITLE + " TEXT)";

private static final String SQL_DELETE_ENTRIES =
    "DROP TABLE IF EXISTS " + FeedEntry.TABLE_NAME;
```



Save data using SQLite

<https://developer.android.com/training/data-storage/sqlite>

 **Caution:** Although these APIs are powerful, they are fairly low-level and require a great deal of time and effort to use:

- There is no compile-time verification of raw SQL queries. As your data graph changes, you need to update the affected SQL queries manually. This process can be time consuming and error prone.
- You need to use lots of boilerplate code to convert between SQL queries and data objects.

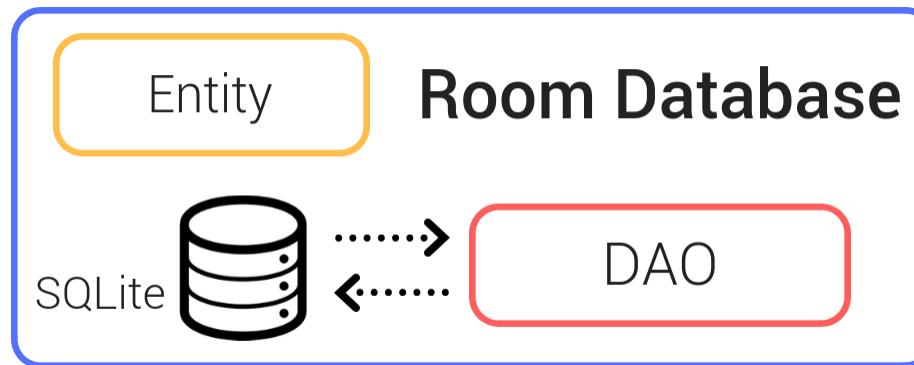
For these reasons, we **highly recommended** using the [Room Persistence Library](#) as an abstraction layer for accessing information in your app's SQLite databases.

Room Persistence Library

To **overcome this**, Google has introduced **Room Persistence Library**

This acts as an **abstraction layer for the existing SQLite APIs**

All the required packages, parameters, methods, and variables are imported into an Android project by using simple annotations



under a [Creative
Commons
4.0 International](#)



What is the Room Persistence Library?

Room is an SQLite object mapping library. It was built to assist developers implementing local SQLite database transactions. It helps to avoid the boilerplate code that was previously associated with interacting with the SQLite database.

Essentially the room persistence library **allows developers to easily convert SQLite table data into java objects**. Room provides compile time checks of SQLite statements

Save data in a local database using Room

<https://developer.android.com/training/data-storage/room>

SQLite and the Room Persistence Library

<https://codingwithmitch.com/blog/sqlite-and-the-room-persistence-library/>

What to choose Realm or SQLite with Room?

<https://itnext.io/what-to-choose-realm-or-sqlite-with-room-e55c34b1675c>

Using a Local DB Exercise

Try to follow the following tutorial for better understanding how to save data in a local database for your app:

Making a Notes App Using Room Database

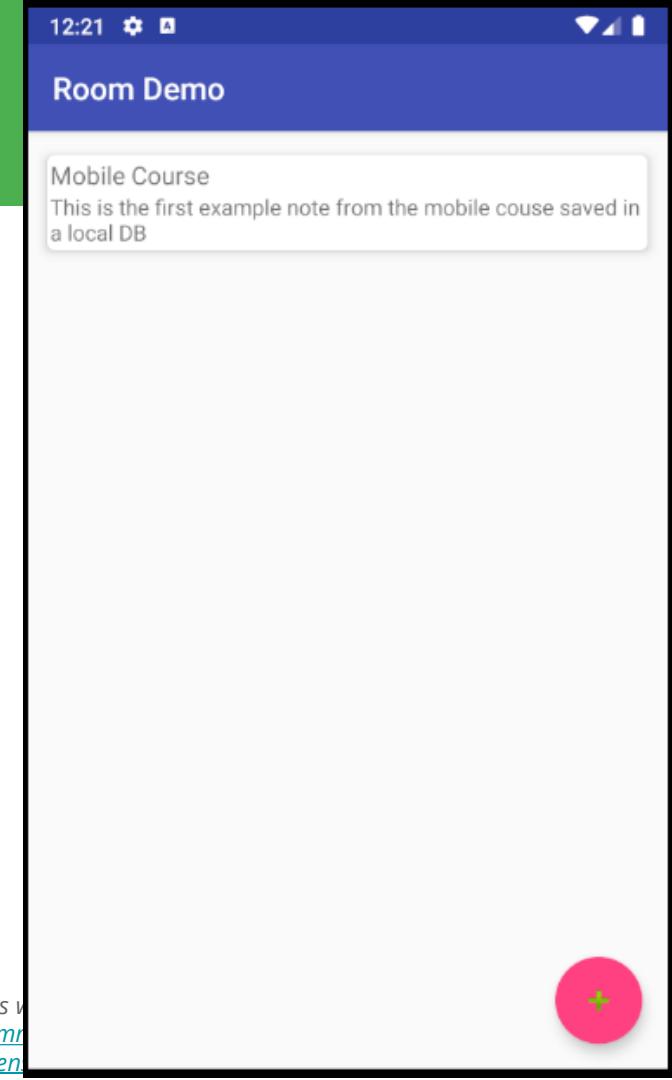
<https://www.pluralsight.com/guides/making-a-notes-app-using-room-database>

The code of the app is already available if you prefer to inspect the working code:

<https://github.com/Pavneet-Sing/RoomDemo>

Read also the ***Official Room Android guide***:

<https://developer.android.com/training/data-storage/room/>



Using a DB Exercise

The demo application demonstrates how to use a local database and get you started with Room. Conceptually, this code can further be extended or changed to build alarm apps, scheduling apps, SMS-driven applications, and more

To implement your project app you may need to:

- a remote and/or a local DB

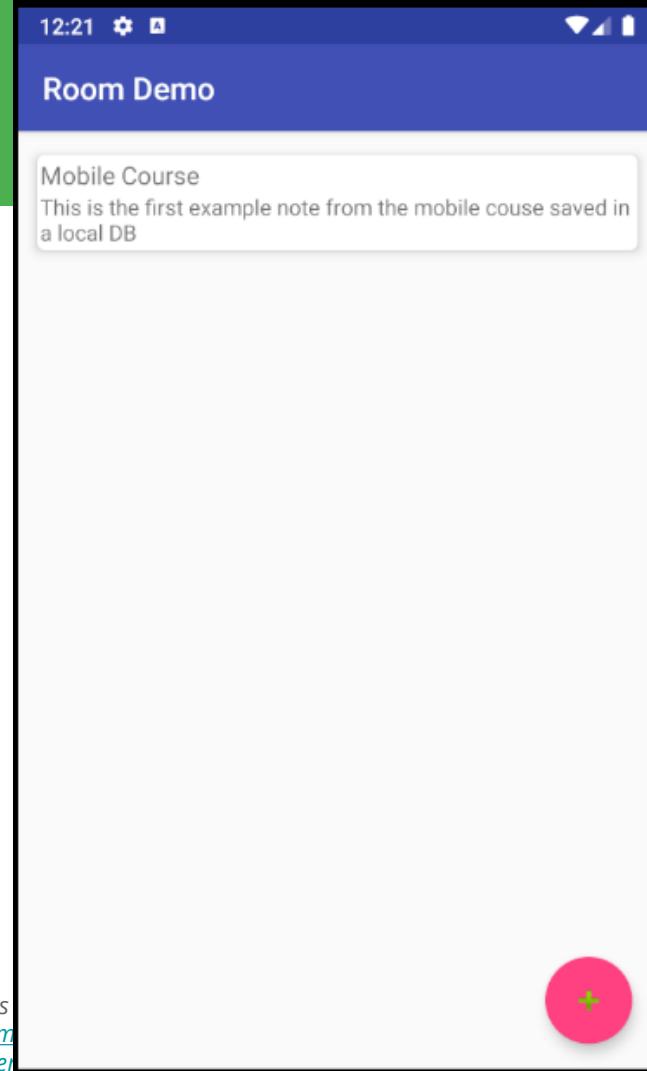
if you need a **remote database** for your final project you can try to use the Google Firebase Realtime Database

<https://youtu.be/U5aeM5dvUpA>



<https://firebase.google.com/docs/database/android/start>

<https://firebase.google.com/docs/database>





Introduction to Kotlin



Contents

- Introduction to Kotlin Language
- Migrating an existing Java app to Kotlin
- Kotlin to Java Converter

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



Kotlin

- **Kotlin** is a **programming language** **widely used** by Android developers
- Kotlin is **designed** to **fully interoperate with Java**
 - its **runs on a Java Virtual Machine** (JVM)
 - by **compiling Kotlin code into Java byte-code**
- **Kotlin syntax** allows to be **more concise**

Kotlin

- Kotlin is **officially supported by Google** for **mobile development on Android**
- At Google I/O 2017, Google announced that Android will support Kotlin as a first-class programming language from now on
 - Kotlin is included as an alternative to Java

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



Kotlin

There are a **some things** that make Kotlin interesting for Android development:

- **Compatibility**: it's compatible with JDK 6, so **older devices aren't left behind**
- **Performance**: it's on **par with Java**
- **Interoperability**: it's **100% interoperable with Java** including annotations
- **Footprint**: the **runtime** library for **Kotlin is tiny**
- **Compilation Time**: there's a little overhead on **clean builds** but it's way faster with **incremental builds**
- **Learning Curve**: it's **easy to learn**, especially for people used to modern languages. The Java to Kotlin converter in IntelliJ and Android Studio makes it even easier

Note! You can also use a **mix of Kotlin and Java in a project**, so take your time learning Kotlin and add it in when you feel comfortable.

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



Kotlin

There are a **some things** that make Kotlin interesting for Android development:

- *Compatibility*: it's compatible with JDK 6, so **older devices aren't left behind**
- *Performance*: it's on **par with Java**
- *Interoperability*: it's **100% interoperable with Java** including annotations
- *Footprint*: the **runtime** library for **Kotlin is tiny**
- *Compilation Time*: there's **a little overhead on clean builds** but it's **way faster with incremental builds**
- *Learning Curve*: it's **easy to learn**, especially for people used to modern languages. The Java to Kotlin converter in IntelliJ and Android Studio makes it even easier

Note! You can also use a **mix of Kotlin and Java in a project**, so take your time learning Kotlin and add it in when you feel comfortable.

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



Kotlin vs Java

Learn the Kotlin programming language

<https://developer.android.com/kotlin/learn>

<https://developer.android.com/kotlin>



For a brief overview of the differences:

Kotlin vs Java – All that you need to know

<https://ayusch.com/kotlin-vs-java/>

Differences Between Java vs Kotlin

<https://www.educba.com/java-vs-kotlin/>

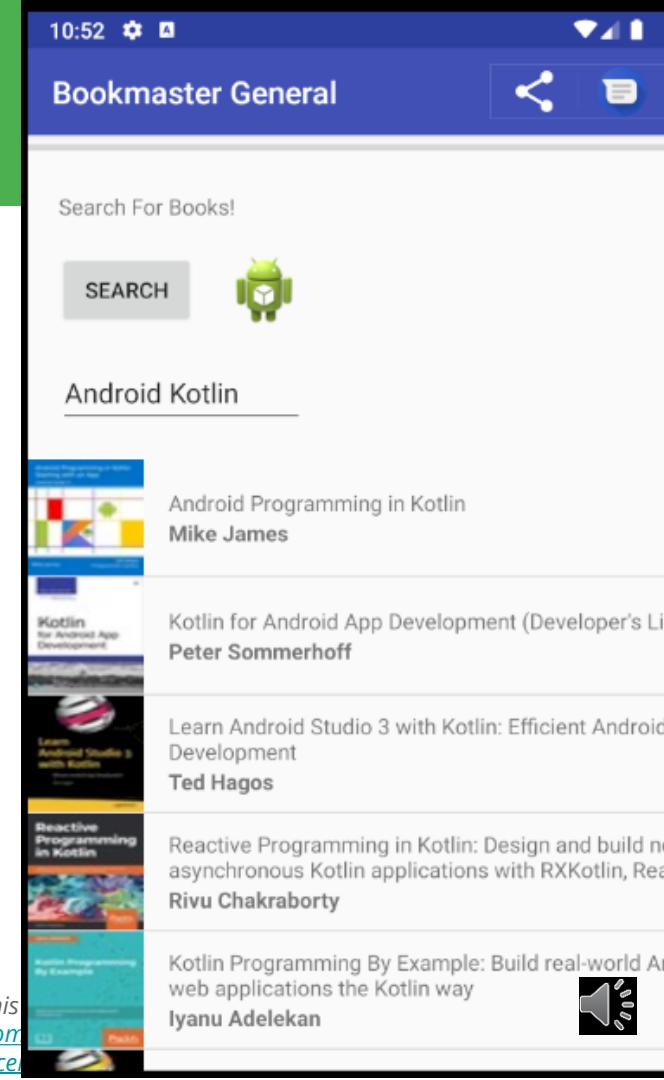
This work is licensed under a [Creative Commons Attribution 4.0 International License](#).

Exercise: from Java to Kotlin

Idea: starting from an **existing Java** app, try to **migrate it** to **Kotlin**

The existing app allows users to

- **search for books**
- **see book covers**
- **share books with friends**

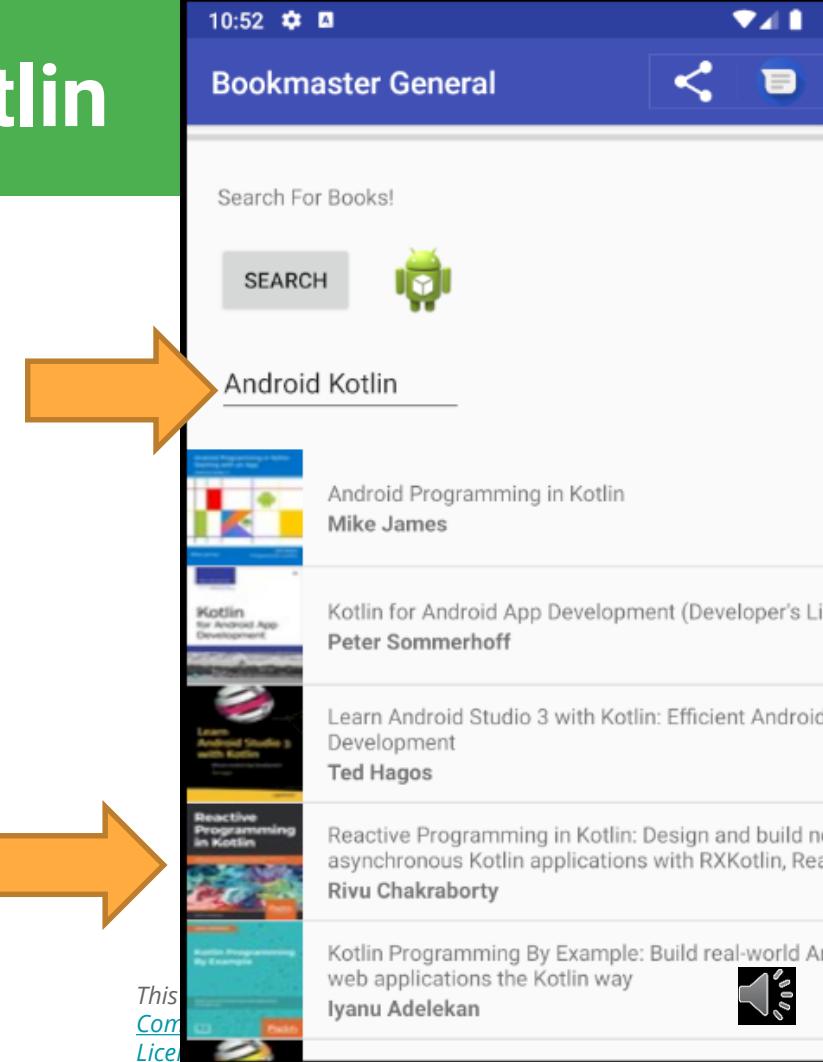


Exercise: from Java to Kotlin

Idea: starting from an **existing Java** app, try to **migrate it** to **Kotlin**

The existing app allows users to

- **search for books**
- **see book covers**
- **share books with friends**

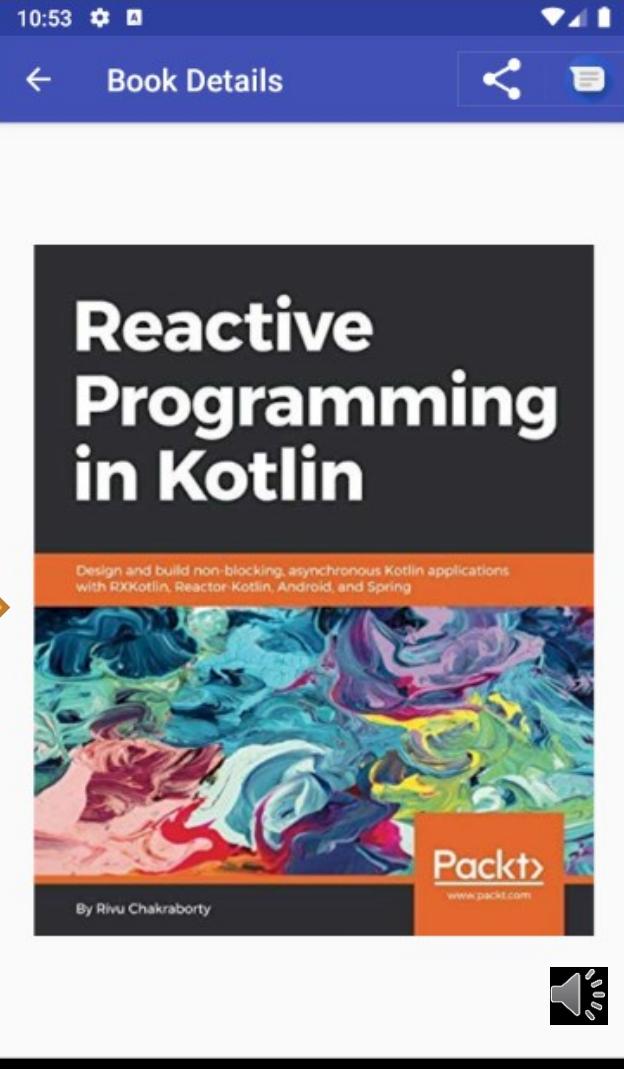
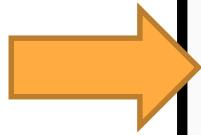


Exercise: from Java to Kotlin

Idea: starting from an **existing Java** app, try to **migrate it** to **Kotlin**

The existing app allows users to

- **search for books**
- **see book covers**
- **share books with friends**

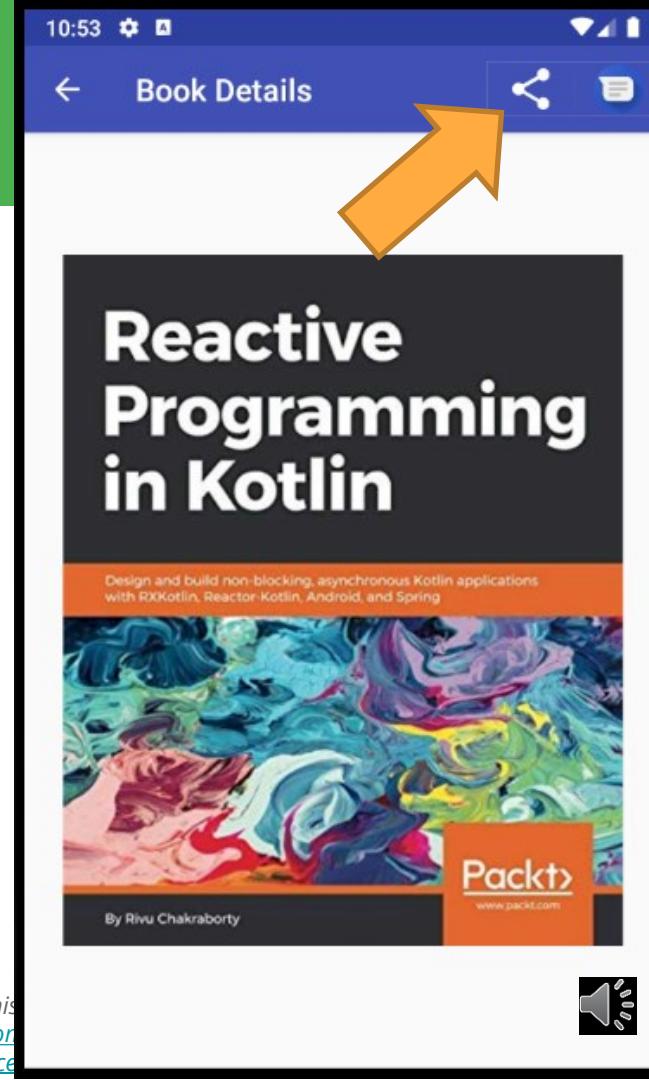


Exercise: from Java to Kotlin

Idea: starting from an **existing Java** app, try to **migrate it** to **Kotlin**

The existing app allows users to

- **search for books**
- **see book covers**
- **share books with friends**



Exercise: from Java to Kotlin

The **project** provided in the tutorial (see next two slides) contains **three source code files** written in **Java**:

- **MainActivity.java**: *shows* the screen for **searching** and **displaying** a **list of books**
- **DetailActivity.java**: *displays* the **book cover** for the ID passed to it
- **JSONAdapter.java**: a custom BaseAdapter that **transforms** a **JSON object** into a **list view item**

Exercise: from Java to Kotlin

The app relies on an **external remote JSON data source** for obtaining books info:

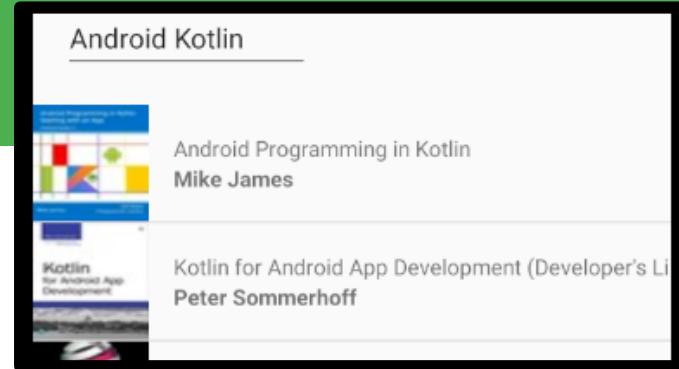
<http://openlibrary.org/search.json?q=>

See for instance the results shown on the website for the search 'Android Kotlin'

<https://openlibrary.org/search?q=Android+Kotlin&mode=everything>

and the corresponding JSON file

<http://openlibrary.org/search.json?q=android+kotlin>



Web Site



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

Exercise: from Java to Kotlin

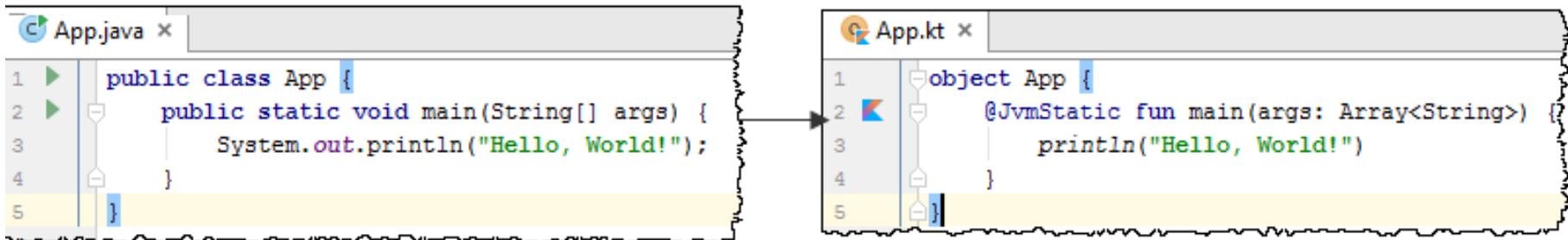
ToDo: follow the tutorial available here (based on this app):

<https://www.raywenderlich.com/1144981-kotlin-for-android-an-introduction>

- **Analyze the structure of the app** to understand how it works (includes possible interesting functionalities for the final app)
- **Create the Kotlin variant** of the `DetailActivity` file
- **Compare the implementations** of `DetailActivity.java` and `DetailActivity.kt` to understand the differences between the two languages

The Java to Kotlin Converter

How can **Java code** be converted automatically in **Kotlin code**?



```
App.java
1 public class App {
2     public static void main(String[] args) {
3         System.out.println("Hello, World!");
4     }
5 }
```

```
App.kt
1 object App {
2     @JvmStatic fun main(args: Array<String>) {
3         println("Hello, World!")
4     }
5 }
```

You don't need to install any plugin to convert Java code to Kotlin code. Now, Google is officially supporting Kotlin language.

Android Studio Menu -> **Code** -> Convert Java File to Kotlin File

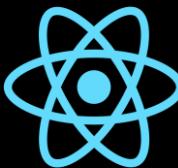
<https://stackoverflow.com/a/39602657>

To do: Try to convert to Kotlin one of the apps developed during the course using this tool



Introduction to React Native





React Native

React Native is an open-source mobile application framework created by Facebook

It is used to develop applications for Android, iOS, and Web by enabling developers to use React along with native platform capabilities

<https://github.com/facebook/react-native>

<https://facebook.github.io/react-native/docs/getting-started.html>

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



Building a React Native app

Try to follow the **Stopwatch tutorial in React Native**

<https://codersera.com/blog/first-react-native-app-stopwatch/>

```
# All source is available here, you can either download  
# or follow the tutorial to understand  
# each and every component individually
```

<https://github.com/codersera-repo/reactnative-stopwatch>

Problem with *npm install -g expo-cli*

Try to uninstall and reinstall the last version of Node.js

<https://nodejs.org/it/>

Expo fails to start the project: error Invalid regular expression

<https://github.com/expo/expo-cli/issues/1074#issuecomment-546167583>

About Expo: <https://expo.io/learn>

This work is licensed under the [Creative Commons Attribution-NonCommercial-ShareAlike License](#).

