CS501
Mobile Application Development

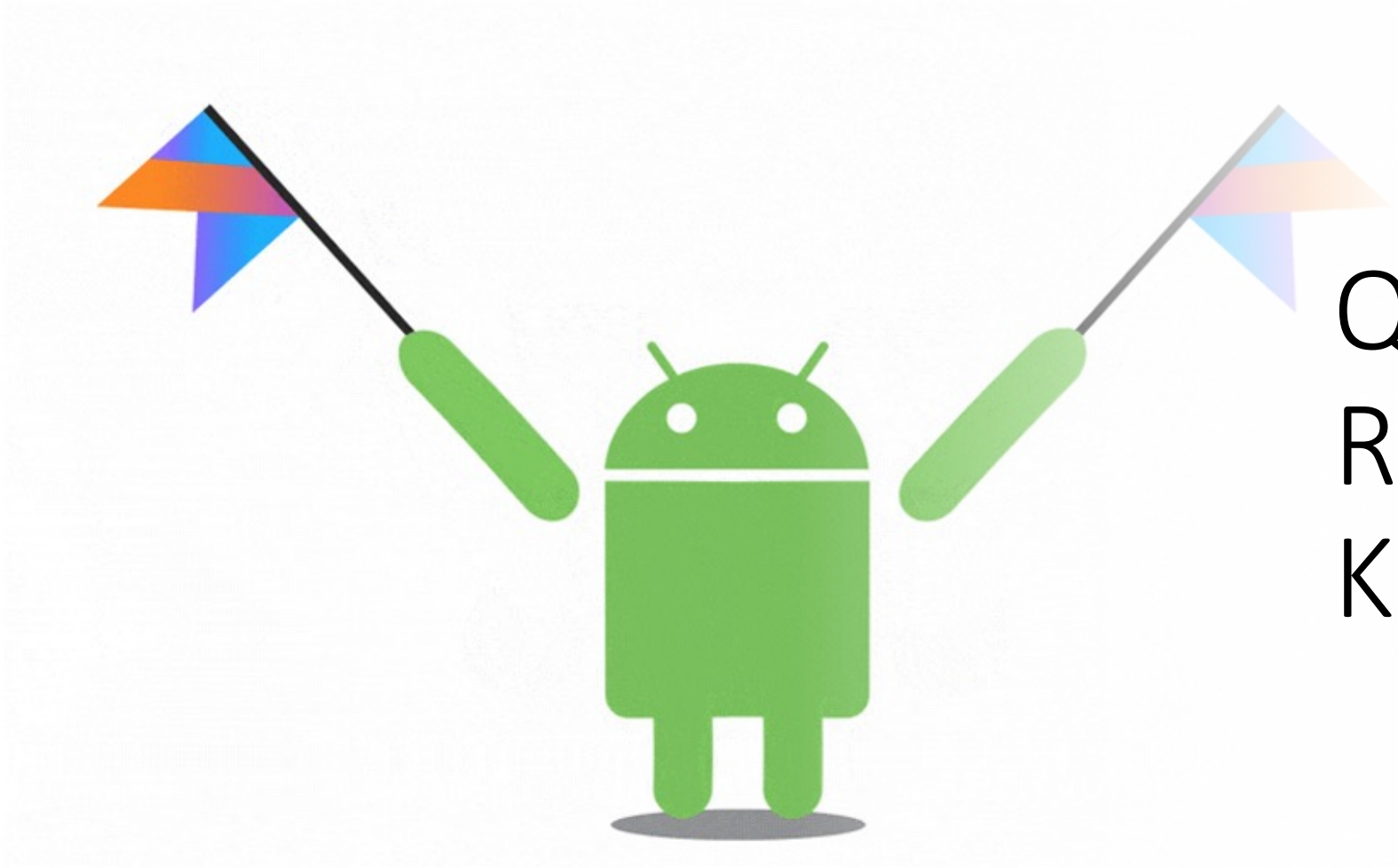
Fall 2023

Lecture 2

Kotlin Review

Layouts

Ron Czik



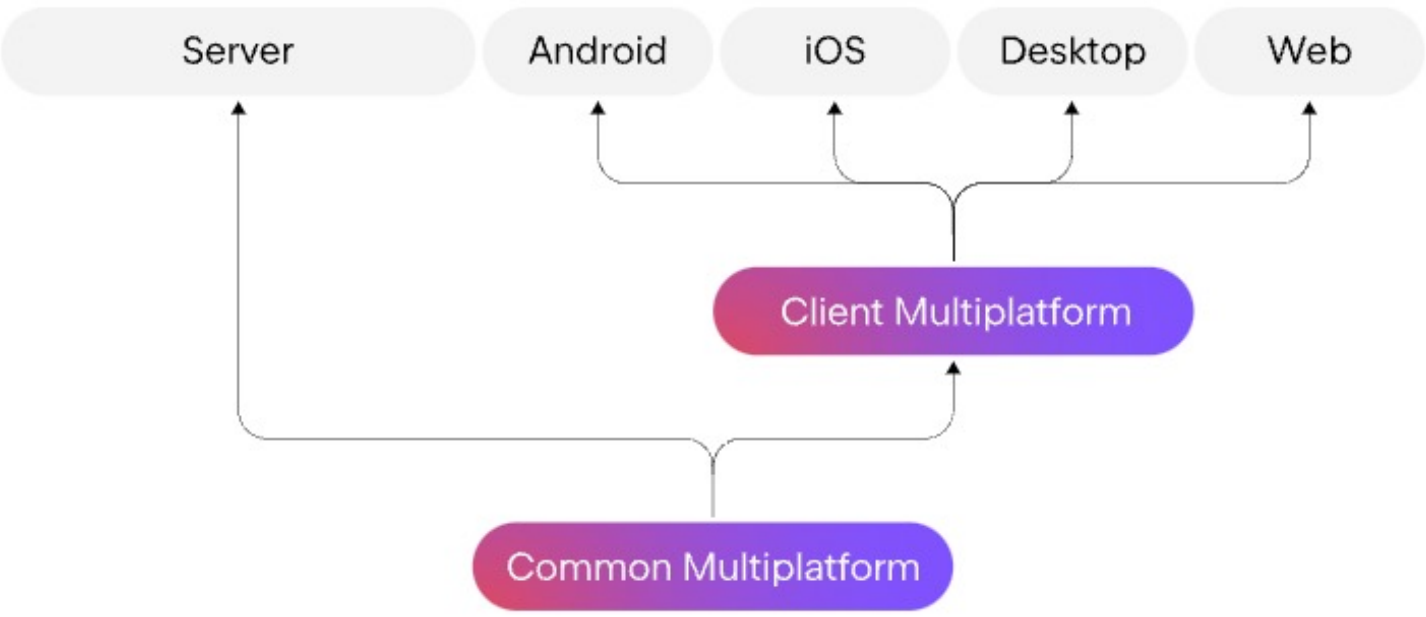
Quick Review of Kotlin

Kotlin is a modern programming language

- Concise and very readable
  - Shortcuts for common actions
  - Reduces repetitive code
  - Rich library of functions
  - Less code – easier to read
- Can use OO and functional programming
  - Supports inheritance and polymorphism as well as functions
- Compiler help
  - Statically typed – can’t do inappropriate actions
  - Most times do not need to specify type – inferred by compiler

Kotlin is flexible

- Android
- Can compile to JVMs
- Can create client-side and server-side JavaScript
- Can compile directly to native machine code for iOS and Linux



https://kotlinlang.org/docs/multiplatform.html#kotlin-multiplatform-use-cases

Why Kotlin for Android?

- Included with Android Studio since 3.0 – 2017
- Preferred language for Android since 2019
- Can build cross-platform (iOS and Android) apps – share business logic
- Support in Android Jetpack (more later)
- Interoperable with Java
- Less code
- Easy to move from Java to Kotlin

Quiz (yes, there are quizzes in this class)

```
val name = ”Misty”
```

Declare a variable called ‘name’, set value to “Misty”

```
val height = 9
```

Declare a variable called ’height’, set value to 9

```
println("Hello")
```

Prints “Hello” to stdout

```
println("My cat is called $name")
```

Prints “My cat is called Misty”

```
println("My cat is $height inches tall")
```

Prints “My cat is 9 inches tall”

```
val a = 6
```

Declare a variable called ‘a’, set value to 6

```
val b = 7
```

Declare a variable called ’b’, set value to 7

```
val c = a + b + 10
```

Declare a variable called ‘c’, set value to 23

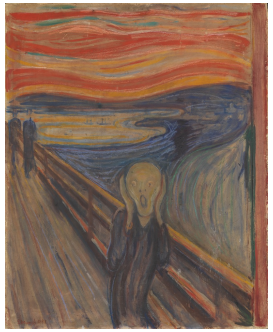
```
val str = c.toString()
```

Declare a variable called ‘str’, set to text “23”

More quiz (not another!)

```
val numList = arrayOf(1, 2, 3)
                                Create an array containing value 1, 2, 3
var x = 0
                                Declare a variable called 'x', set value to 0
while (x < 3) {
                                Continue looping if X is less than 3
      println("Item $x is ${numList[x]}")
                                Print the index and value of array
      x = x + 1
                                Add 1 to x
}
```

One more (when will he stop?)



```
val myCat = Cat(name, height)
                                Declare 'myCat' and create a Cat object
val y = height - 3
                                Declare y, set value to 6
if (y < 5) myCat.mewow(4)
                                If y is less than 5, Cat mewows 4 times
while (y < 8) {
                                Keep looping as long as y is less than 8
    myCat.play()
                                Cat plays
    y = y + 1
                                Add 1 to y
}
```

Entry point

- Entry point of a Kotlin application is the main function

```
fun main()  {
     println("Hello world")
}

fun main(args: Array<String>) {
     println(args.contentToString())
}
```

Accepts a variable number of String arguments

Functions

Function with 2 parameters and returns an Int

```
fun sum(a: Int, b: Int): Int {
    return a + b
}
```

Function body as an expression and an Int return type is inferred

```
fun sum(a: Int, b: Int) = a + b
```

Function returns no meaningful value

```
Fun printSum(a: Int, b: Int) {
    println("Sum of $a and $b is ${a + b}")
}
```

Read-only variables

- Defined using the `val` keyword and be assigned only once

```
val a: Int = 1   // Immediate assignment
val b = 2        // 'Int' type is inferred
val c: Int       // Type required when no initializer is
provided
c = 3            // deferred assignment
```

Reassigned variables

- Use the `var` keyword

```
var x = 5  // 'Int' type is inferred
x += 1
```

Can declare variables at the top level

```
val PI = 3.14
var x = 0

fun incrementX() {
      x += 1
}
```

Creating classes

```
class shape

class Rectangle(var height: Double, var length: Double) {
      var perimeter = (height + length) * 2
}
val rectangle = Rectangle( 5.0, 2.0)
println("The perimeter is ${rectangle.perimter}")

The perimeter is 14.0
```

Output

Inheritance is declared by a colon (:)

- Classes are final by default, to make a class inheritable you need to mark it as `open`

```
open class Shape

class Rectangle(var height: Double, var length: Double): Shape() {
      var perimeter = (height + length) *2
}
```

Comments

```
// This is an end-of-line comment

/* This is a block comment
   on multiple lines */

/* The comment starts here
/* contains a nested comment */
and ends here. */
```

String templates

```
fun main() {
        var a = 1
        simple name in template:
        val s1 = “a is $a”

        a = 2
        // arbitrary express in template:
        val s2 = “${s1.replace(“is”, “was”)}, but now is $a”
        println(s2)
}
```

What do you think is the output of this function?

```
a was 1, but now is 2
```

Conditional expressions

```
fun maxOf(a: Int, b: Int): Int {
        if (a > b {
                return a
        } else {
                return b
        }
}

fun main() {
        println("max of 0 and 42 is ${maxOf(0, 42)}")
}
```

What do you think is the output of this function?

```
Max of 0 and 42 is 42
```

`if` can also be used as an expression – output?

```
fun maxOf(a: Int, b: Int) = if (a > b) a else b

fun main() {
     println("max of 0 and 42 is ${maxOf(0, 42)}")
}
```

```
max of 0 and 42 is 42
```

`for` loop – what's the output?

```
fun main() {
      val items = listOf("apple", "banana", "kiwifruit")
      for (item in items) {
            println(item)
      }
}
```

```
apple
banana
kiwifruit
```

Using indices – what’s the output?

```
Fun main() {
      val items = listOf(“apple”, “banana”, “kiwifruit”)
      for (index in items.indices) {
            println(“item at $index is ${items[index]}”)
      }
}
```

```
item at 0 is apple
item at 1 is banana
item at 2 is kiwifruit
```

`while` loop – what's the output?

```
fun main() {
      val items = listOf("apple", "banana", "kiwifruit")
      var index = 0
      while (index < items.size) {
            println("item at $index is ${items[index]}")
            index++
      }
}
```

```
item at 0 is apple
item at 1 is banana
item at 2 is kiwifruit
```

When expression

```
fun describe(obj: Any): String =
        when (obj) {
                1                -> “One”
                “Hello           -> “Greeting”
                is Long          -> “Long”
                !is String       -> “Not a string”
                else             -> “Unknown”
        }

fun main() {
        println(describe(1))
        println(describe(“Hello))
        println(describe(1000L))
        println(describe(2))
        println(describe(“other”))
}
```

What’s the output?

```
One
Greeting
Long
Not a string
Unknown
```

Check if number is within a range using `in`

```
fun main() {
      val x = 10
      val y = 9
      if (x in 1..y+1) {
            println("fits in range")
      } else {
            println("out of range")
      }
}
```

What's the output?

```
fits in range
```

Out of range – what’s the output?

```
fun main() {
    val list = listOf("a", "b", "c")

    if (-1 !in 0..list.lastIndex) {
        println("-1 is out of range")
    }
    if (list.size !in list.indices) {
        println("list size is out of valid list indices range, too")
    }
```

```
-1 is out of range
list size is out of valid list indices range, too
```

Iterate over a range – what’s the output?

```
fun main() {
    for (x in 1..5) {
        print(x)
    }
}
```

```
12345
```

Over a progression – what’s the output?

```
fun main() {
    for (x in 1..10 step 2) {
        print(x)
    }
    println()
    for (x in 9 downTo 0 step 3) {
        print(x)
    }
}
```

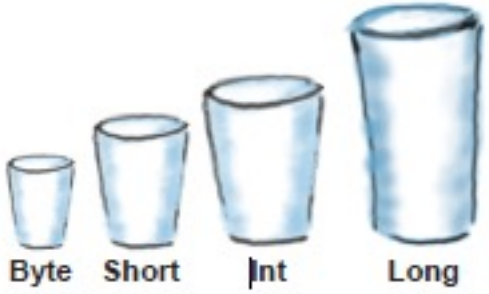
```
13579
9630
```

Types

- Everything is an object
  - Some of special internal representation
- Numbers
- Booleans
- Characters
- Strings
- Arrays

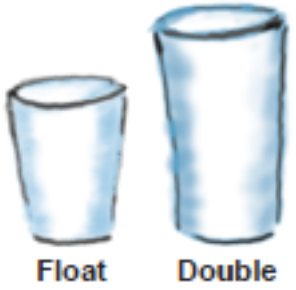
Int

- Byte (8 bits)
- Short (16 bits)
- Int (32 bits)
- Long (64 bits)
- When you initialize a variable with no explicit type specification, the compiler automatically infers the type with the smallest range enough to represent the value.



Floats

- Float (32 bits, 24 significant)
- Double (64 bits, 53 significant)
  - 123.5 or 123.5e10



Conversions

```
toByte(): Byte
toShort(): Short
toInt(): Int
toLong(): Long
toFloat(): Float
toDouble(): Double
```

In many cases, there is no need for explicit conversions because the type is inferred from the context, and arithmetical operations are overloaded for appropriate conversions

What does this do?

```
fun main() {
    val x = 5 / 2
    //println(x == 2.5) // ERROR: Operator '==' cannot be applied to 'Int' and 'Double'
    println(x == 2)
}
```

```
true
```

How about this?

```
fun main() {
    val x = 5L / 2
    println(x == 2L)
}
```

```
true
```

To return a float, explicitly convert one arg

```
fun main() {
    val x = 5 / 2.toDouble()
    println(x == 2.5)
}
```

What's the output?

```
true
```

Booleans

- `true` or `false`
- Operators
  - || – disjunction (logical *OR*)
  - && – conjunction (logical *AND*)
  - ! – negation (logical *NOT*)

What does this do?

```
fun main() {
    val myTrue: Boolean = true
    val myFalse: Boolean = false
    val boolNull: Boolean? = null

    println(myTrue || myFalse)
    println(myTrue && myFalse)
    println(!myTrue)
}
```

```
true
false
false
```

Char

- Go in single quotes: ‘1’

```
fun main() {
    val aChar: Char = 'a'

    println(aChar)
}
```

```
a
```

Strings

- `val str = "abcd 123”`
- Elements of a string are characters that you can access via the indexing operation: `s[i]`
  - You can iterate over these characters with a for loop:

```
fun main() {
val str = "abcd"
    for (c in str) {
        println(c)
    }
}
```

```
a
b
c
d
```

What’s the output?

Kotlin resources

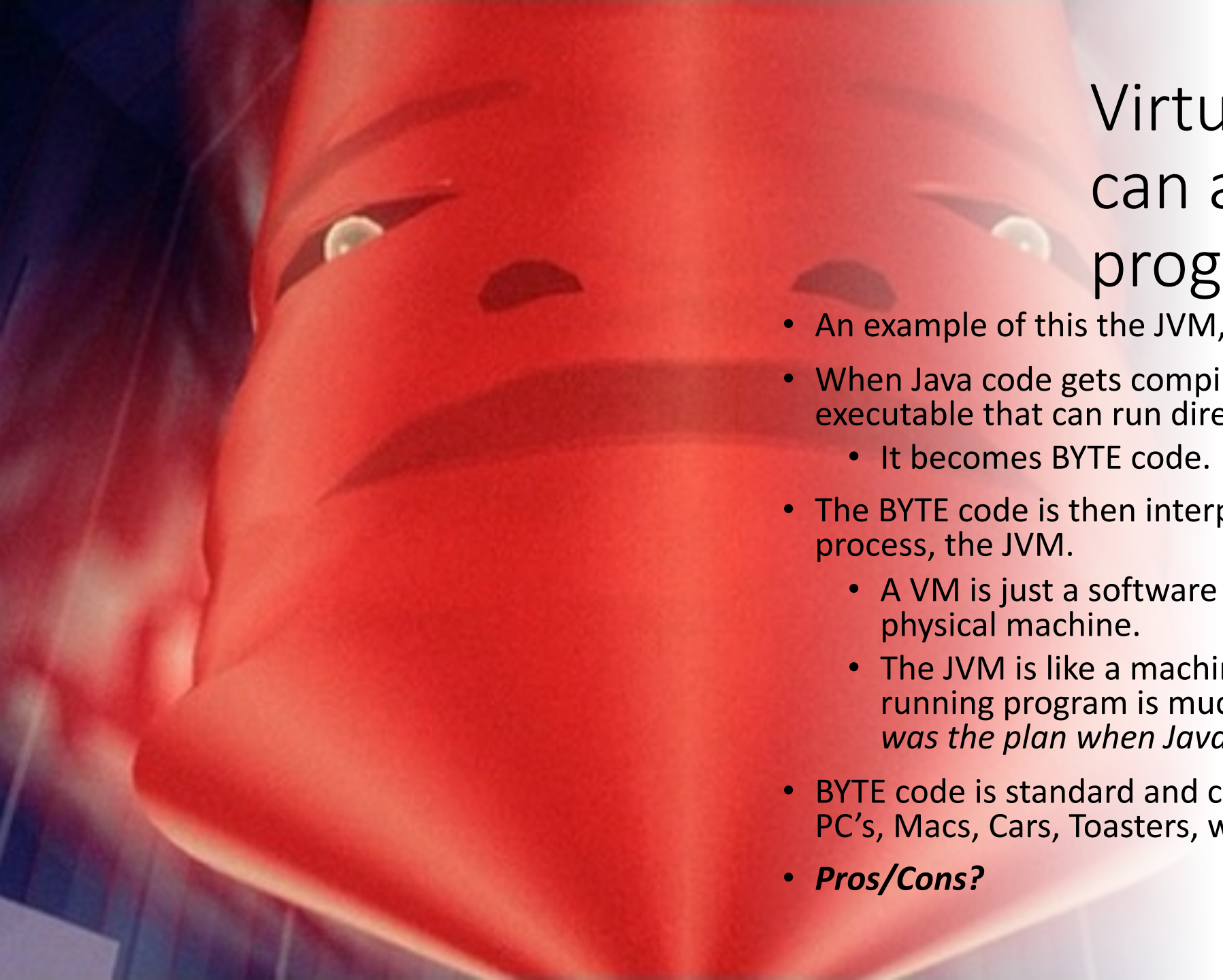
- There's LOTS more. Here are some resources
  - Kotlin: https://kotlinlang.org/
  - Getting started with Kotlin: https://kotlinlang.org/docs/home.html
  - Kotlin language specification: https://kotlinlang.org/spec/introduction.html
  - Android Kotlin: https://developer.android.com/kotlin

Let’s Look Under the Hood

JVM vs. DVM vs. CLR vs. ART

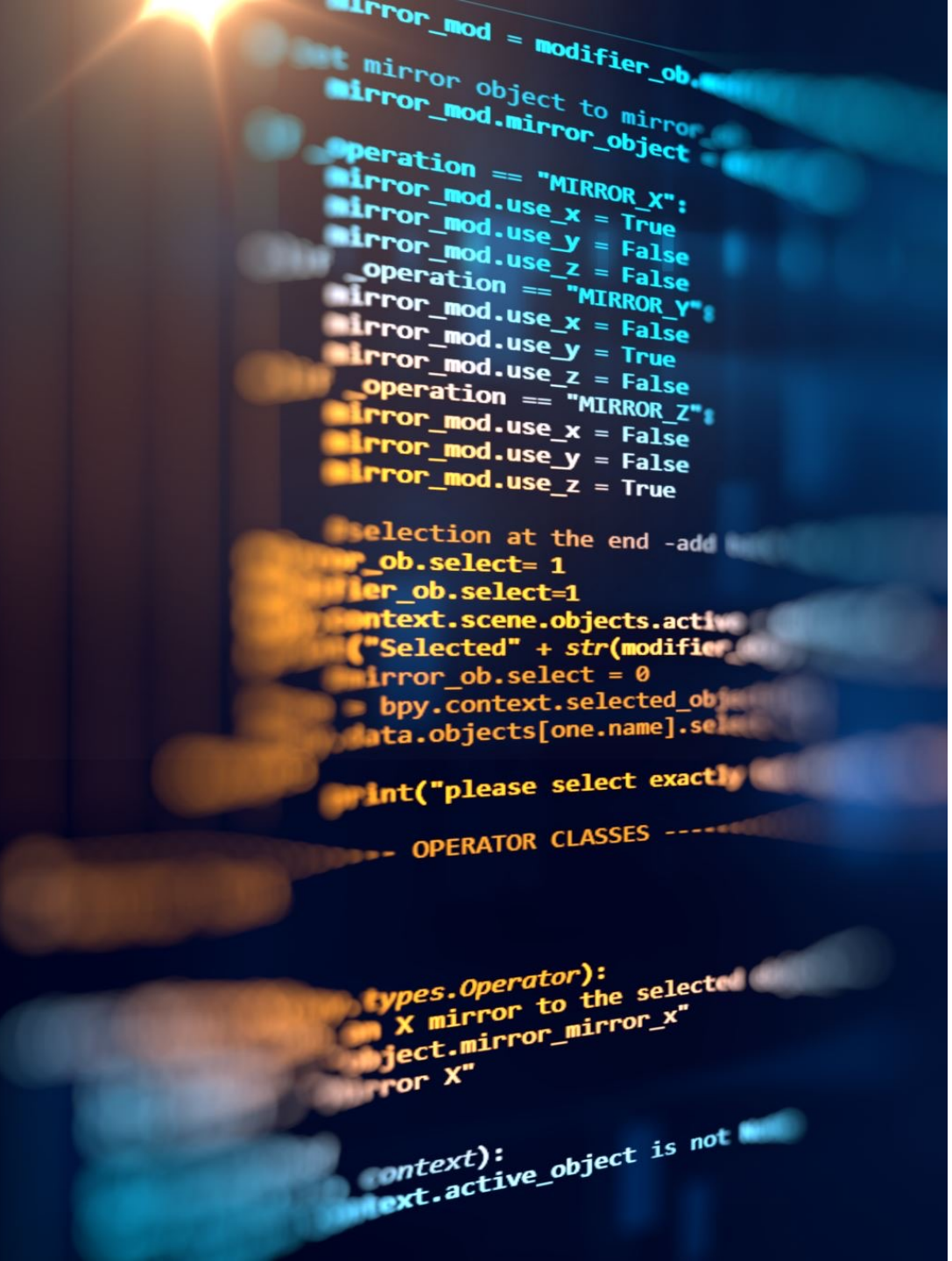
What is a native executable?

- Program the runs directly on a computer
- *What are some IDEs/languages that can compile native executables?*
  - _____
  - _____
  - _____
  - _____
- Native Executables are (typically) unbounded, can go anywhere on the computer and do anything. The OS, offers some minimal protection.
- *What are some Pros/Cons of Native EXE's?*
- Very _____, but can also be _____.



Virtual machines can also run programs

- An example of this the JVM, Java Virtual Machine.
- When Java code gets compiled, it doesn't become an executable that can run directly on your computer.
  - It becomes BYTE code.
- The BYTE code is then interpreted by another running process, the JVM.
  - A VM is just a software application that emulates a physical machine.
  - The JVM is like a machine inside a machine. The running program is much more secure. *At least that was the plan when Java was invented…* ☺
- BYTE code is standard and can run on JVM's anywhere, PC's, Macs, Cars, Toasters, whatever.
- ***Pros/Cons?***



What is managed code?

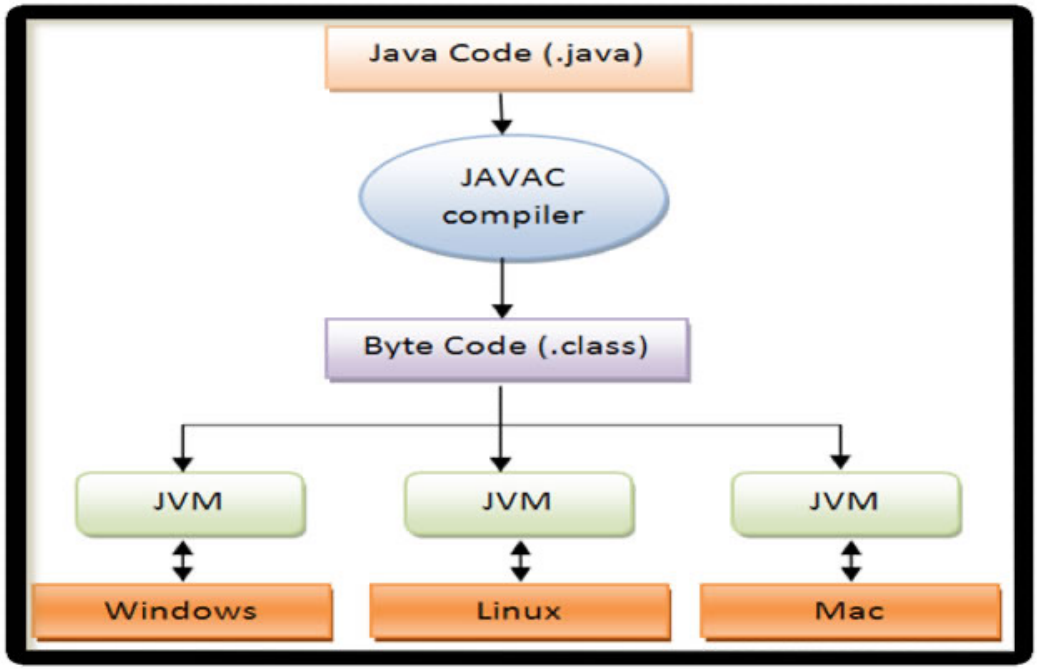
- Managed code is the next evolution of running code inside virtual machines. Came about when MS got sued by SUN (Maker of Java).
- Code is compiled into an executable. It will run directly on the CPU, but only with a Manager running with it.
- The first example of managed Code is the CLR, Common Library Runtime by Microsoft.
- C# gets compiled into a Managed EXE. The EXE will not run without the CLR.
- Not a VM. It runs “next” to the Exe, “Managing”. This includes Memory Management, Garbage Collection and Security. Eg, no Pointers in C#.

So how do Android programs run

- Natively? Inside a VM? In a Managed Environment?
- All of the above are possible.
- But like Microsoft's .NET, Google's Android evolved to run in a Managed Environment, the ART, Android Runtime.
- Replaced Dalvik virtual machine
- Also like .NET, Google got sued by SUN@!!

Mommy, where does managed code come from?

- Java gained popularity in the late 1990's when there were many different types of computers connecting to the Internet. HTML was X-platform, but insufficient.
- Vendors wanted a way to run their Applications on any computer with any architecture/OS. They also only wanted to write the code once. This could not be done natively, eg, with C++
  - Note: Macs were using several different chipsets (not including Intel)
- Customers wanted to be sure these Applications were secure.
- Enter Java and its JVM.



The battle for managed code

- In the late 1990's Java was getting increasing amounts of Market Share.
- Many companies had licensed Java and integrated into their own development environments.
- Microsoft's implementation of Java would support code written on an MS platform but wouldn't necessarily work across platforms.
- Sun sued MS for $35 Million. (THAT'S IT ?!?)
- They eventually settled for $20 Million, Microsoft was permanently prohibited from using "Java compatible" trademarks.
- A few years later Microsoft built their own managed code environment - .NET as well as a new language to go with it, C#

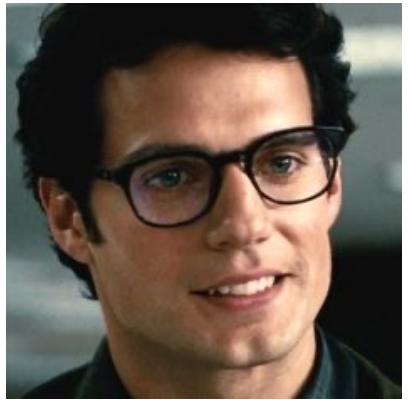
JVM vs. DVM vs. ART vs. NET

- What is the JVM again?
- DVM is very similar to the JVM, but is lightweight and can spawn itself quickly.
  - This is a must! *Why?*
- Every Android App runs in its own DVM.
  - It's possible to get around this, and let Apps share a DVM, but it's not typical.
  - *Well that's how it used to be anyway. Now Android uses the ART.*
- DVM is very similar to JVM.
- Nice Refs:
  - http://stackoverflow.com/questions/13577733/how-an-android-application-is-executed-on-dalvik-virtual-machine

From Source to Executable

- Native EXE's
  - Eg, C++ Source ➔ C++ COMPILER ➔ NATIVE EXE
  - NATIVE EXE'S ARE FAST, BUT INSECURE.
- JVM Apps
  - Eg, Java Source ➔ Java COMPILER ➔ Java Byte Code ➔ JVM
  - Slower (interpreted), more secure, runs in sandbox.
- DVM Apps are based on Java, but there is an extra compilation and a different Virtual Machine.
  - EG, Java Source ➔ Java COMPILER ➔ Java Byte Code ➔ Dalvik COMPILER ➔ DVM
  - Takes a while to compile. Things get compiled twice!
  - Why the hassle? JVM is not Free, DVM is Free. DVM is also optimized for mobile apps?
  - Why use a VM at all?
    - Secure, also "guaranteed" to run across different devices. Just like Java.

From Dalvik to ART (Android Runtime)



- Who is ART? Kind of a Nerd, but with superpowers.
- Compiles code ahead of time (AOT) into runnable executable.
- Very similar to running managed code in .NET framework.
- Takes longer to install, because of the additional compilation, but runs and starts up quicker.
- Launched Experimentally in Android KitKat 4.4
- Now it’s the default standard.

ART

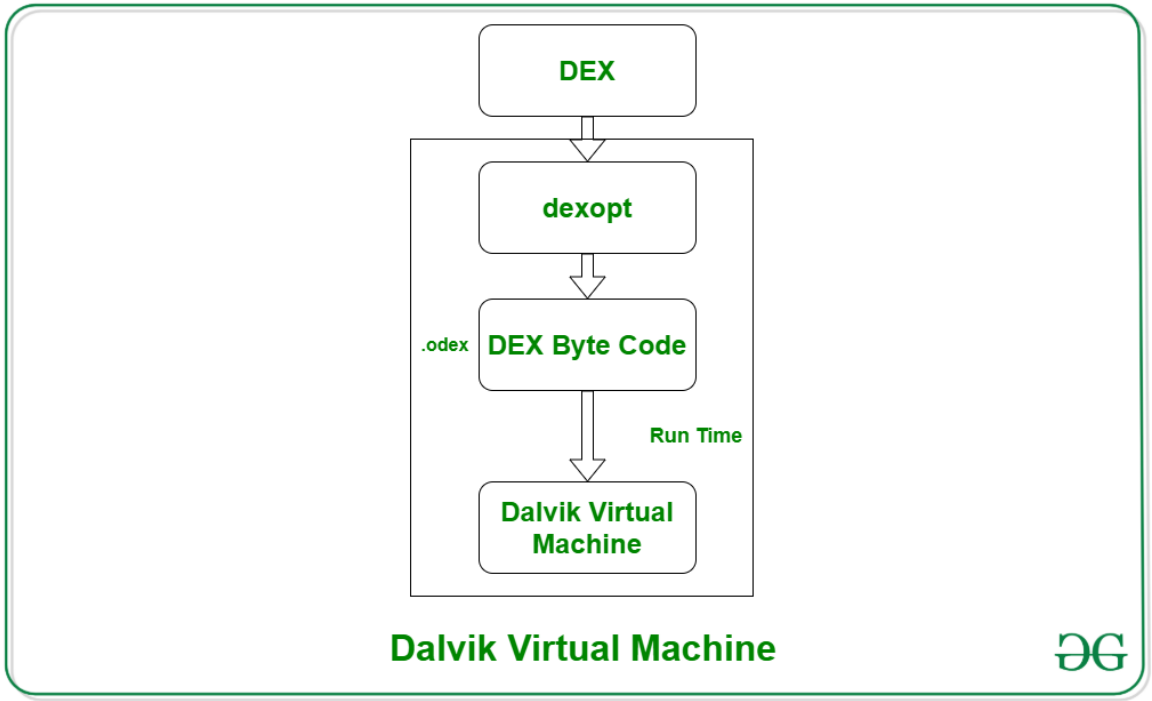
- ART is the managed runtime used by applications and some system services on Android.
- ART and its predecessor Dalvik were originally created specifically for the Android project.
- ART as the runtime executes the Dalvik Executable format and Dex bytecode specification.

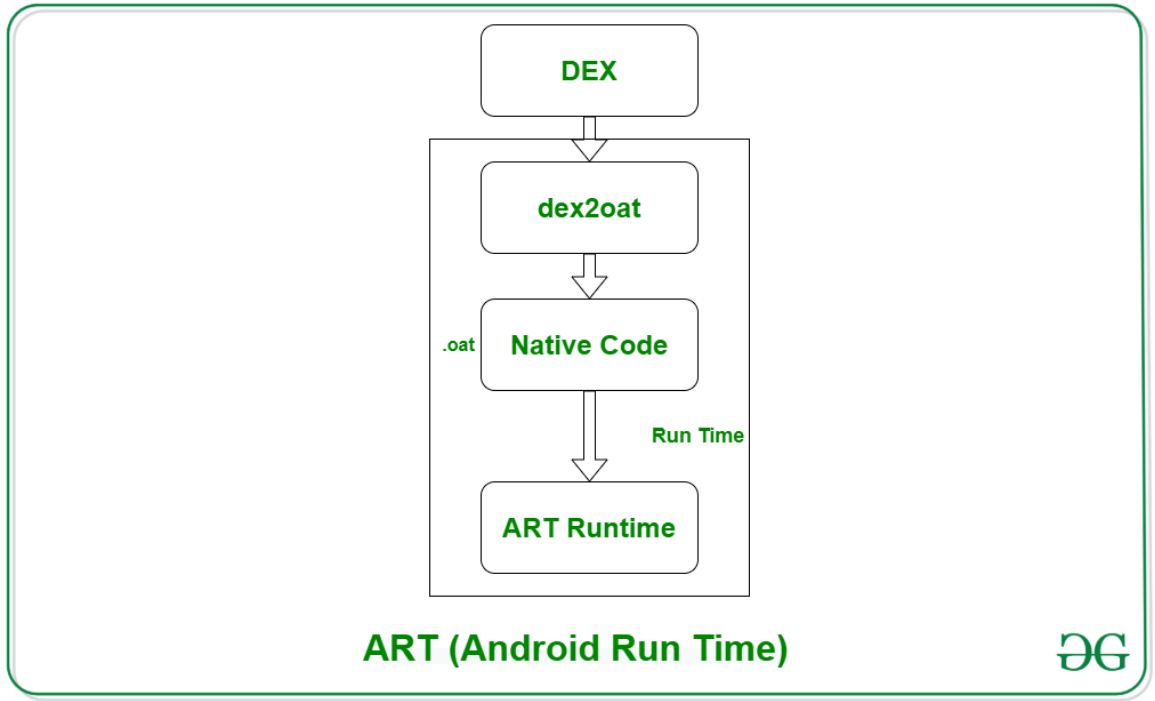
ART Features

- Ahead-of-time (AOT) compilation
  - ART introduces ahead-of-time (AOT) compilation, which can improve app performance. ART also has tighter install-time verification than Dalvik.
- Improved garbage collection
  - Garbage collection (GC) is very resource intensive, which can impair an app's performance, resulting in choppy display, poor UI responsiveness, and other problems.
- Development and debugging improvements
  - Support for sampling profiler
  - Support for more debugging features
  - Improved diagnostic detail in exceptions and crash reports

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

ART vs. Dalvik





https://www.geeksforgeeks.org/difference-between-dalvik-and-art-in-android/

ART vs. Dalvik

DALVIK VIRTUAL MACHINE	ANDROID RUN TIME
Faster Booting time	Rebooting is significantly longer
Cache builds up overtime	The cache is built during the first boot
Occupies less space due to JIT	Consumes a lot of storage space internally due to AOT
Works best for small storage devices	Works best for Large storage devices
Stable and tested virtual machine	Experimental and new – not much app support comparatively
Longer app loading time	Extremely Faster and smoother Faster and app loading time and lower processor usage
Uses JIT compiler(JIT: Just-In-Time) Thereby resulting in lower storage space consumption	Uses AOT compiler(Ahead-Of-Time) thereby compiling apps when installed
Application lagging due to garbage collector pauses and JIT	Reduced application lagging and better user experience
App installation time is comparatively lower as the compilation is performed later	App installation time is longer as compilation is done during installation
DVM converts bytecode every time you launch a specific app.	ART converts it just once at the time of app installation. That makes CPU execution easier. Improved battery life due to faster execution.
It is slower than ART.	It is faster.
It does not provide optimized battery life as it consumes more power.	It provides optimized battery performance as it consumes less power.
While considering Booting, then this device is fast.	It lags in term of booting.

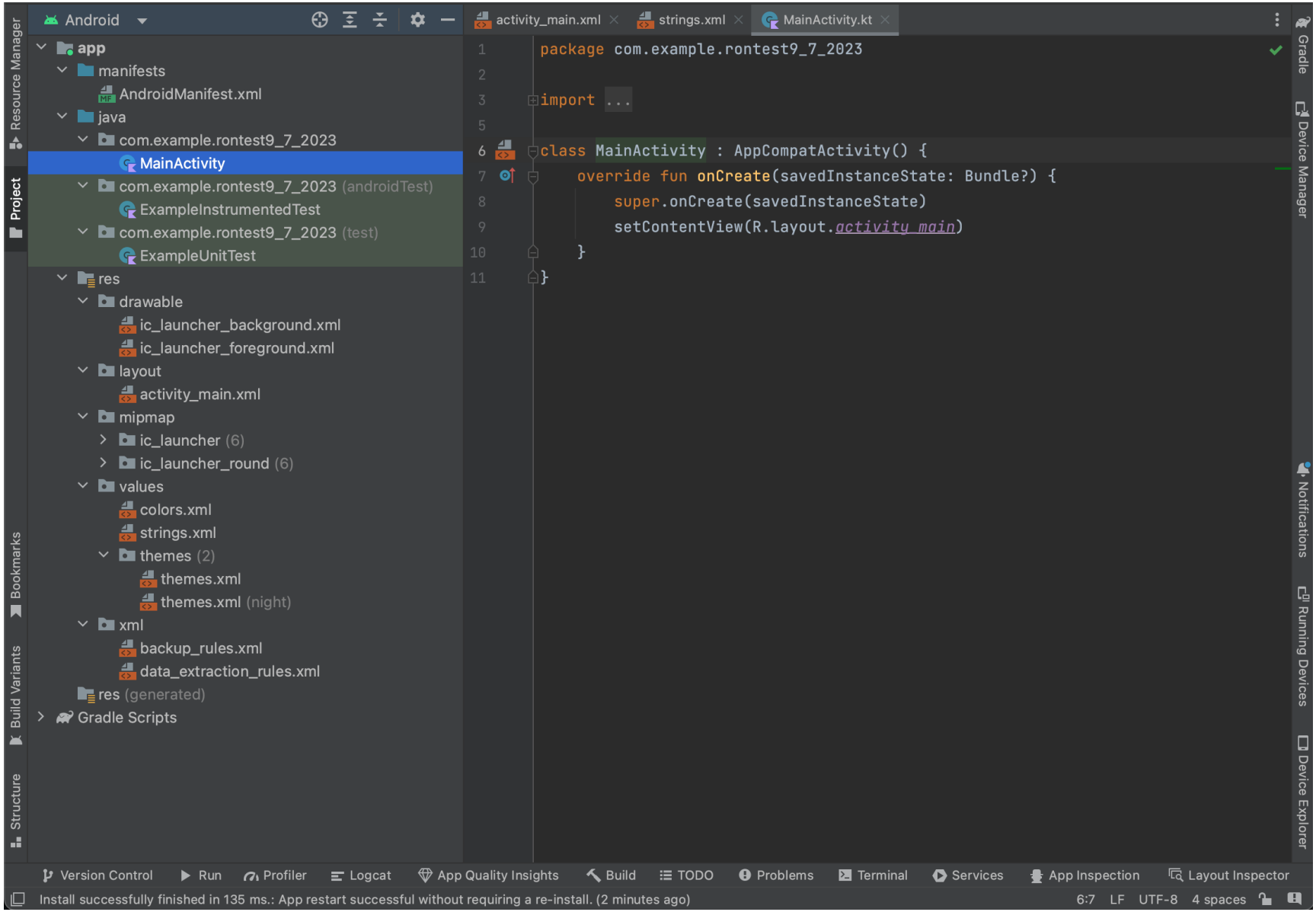
A quick word on Gradle

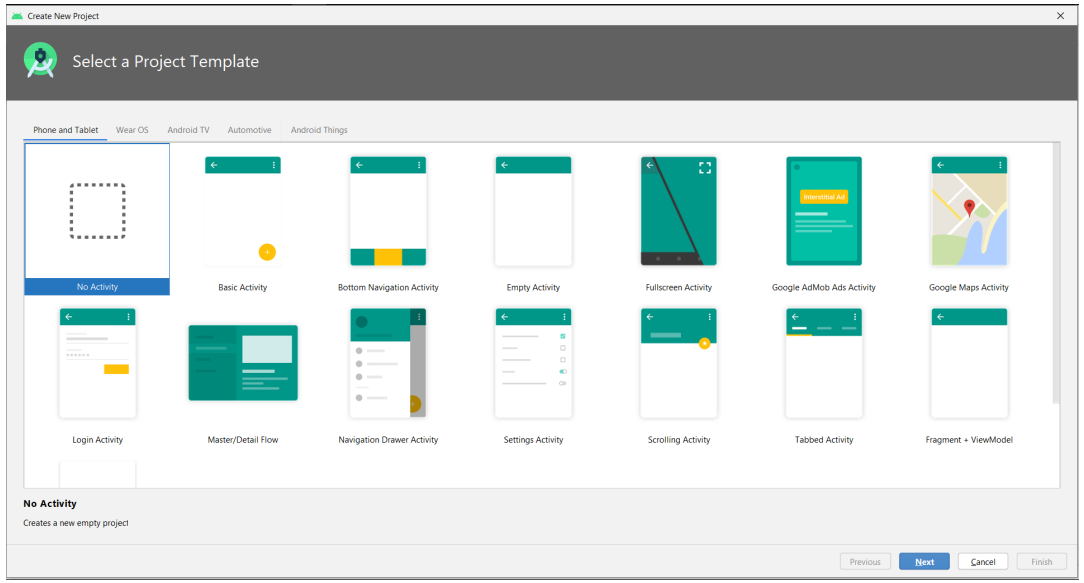
- Gradle is the Build Manager for Android.
- Manages Dependencies
- SDK Versions
- Order of compilation
- Third party Library integration (JARs).
- Stuff you don’t want to have to worry about.
  - *You usually won’t have to go in there,*
    - *Unless your compiling someone else’s code, then you may have to tweak SDK versions (to match the ones you have).*
    - *Or adding a third-party library to your project (dependencies section).*

https://developer.android.com/build#:~:text=Android%20Studio%20uses%20Gradle%2C%20an,all%20versions%20of%20your%20app.

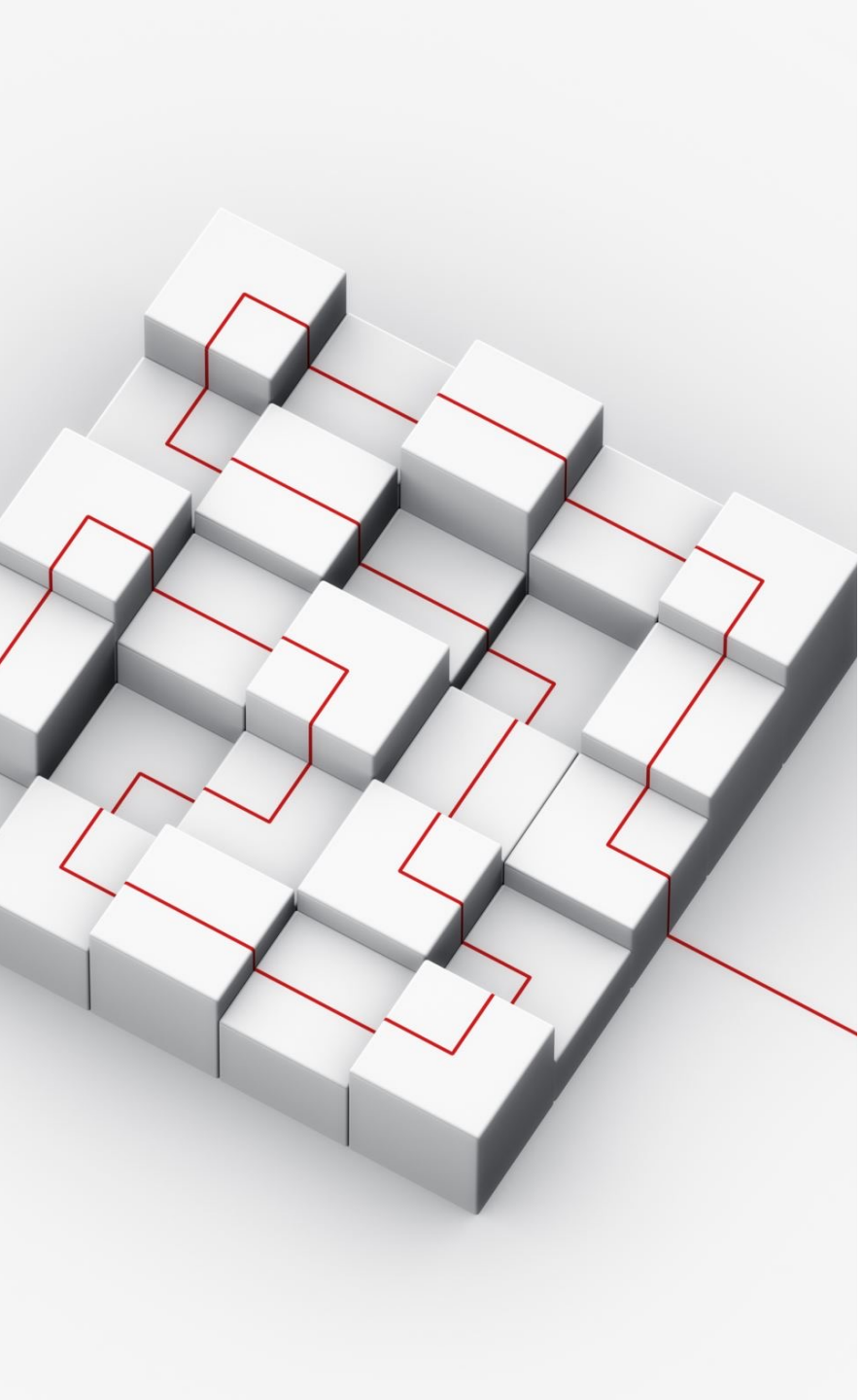
Break

Key files


Android
app
manifests
AndroidManifest.xml
java
com.example.rontest9_7_2023
MainActivity
com.example.rontest9_7_2023 (androidTest)
ExampleInstrumentedTest
com.example.rontest9_7_2023 (test)
ExampleUnitTest
res
drawable
ic_launcher_background.xml
ic_launcher_foreground.xml
layout
activity_main.xml
mipmap
ic_launcher (6)
ic_launcher_round (6)
values
colors.xml
strings.xml
themes (2)
themes.xml
themes.xml (night)
xml
backup_rules.xml
data_extraction_rules.xml
res (generated)
Gradle Scripts
activity_main.xml
strings.xml
MainActivity.kt
package com.example.rontest9_7_2023
import ...
class MainActivity : AppCompatActivity() {
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_main)
}
}
Version Control
Run
Profiler
Logcat
App Quality Insights
Build
TODO
Problems
Terminal
Services
App Inspection
Layout Inspector
Install successfully finished in 135 ms.: App restart successful without requiring a re-install. (2 minutes ago)
6:7
LF
UTF-8
4 spaces

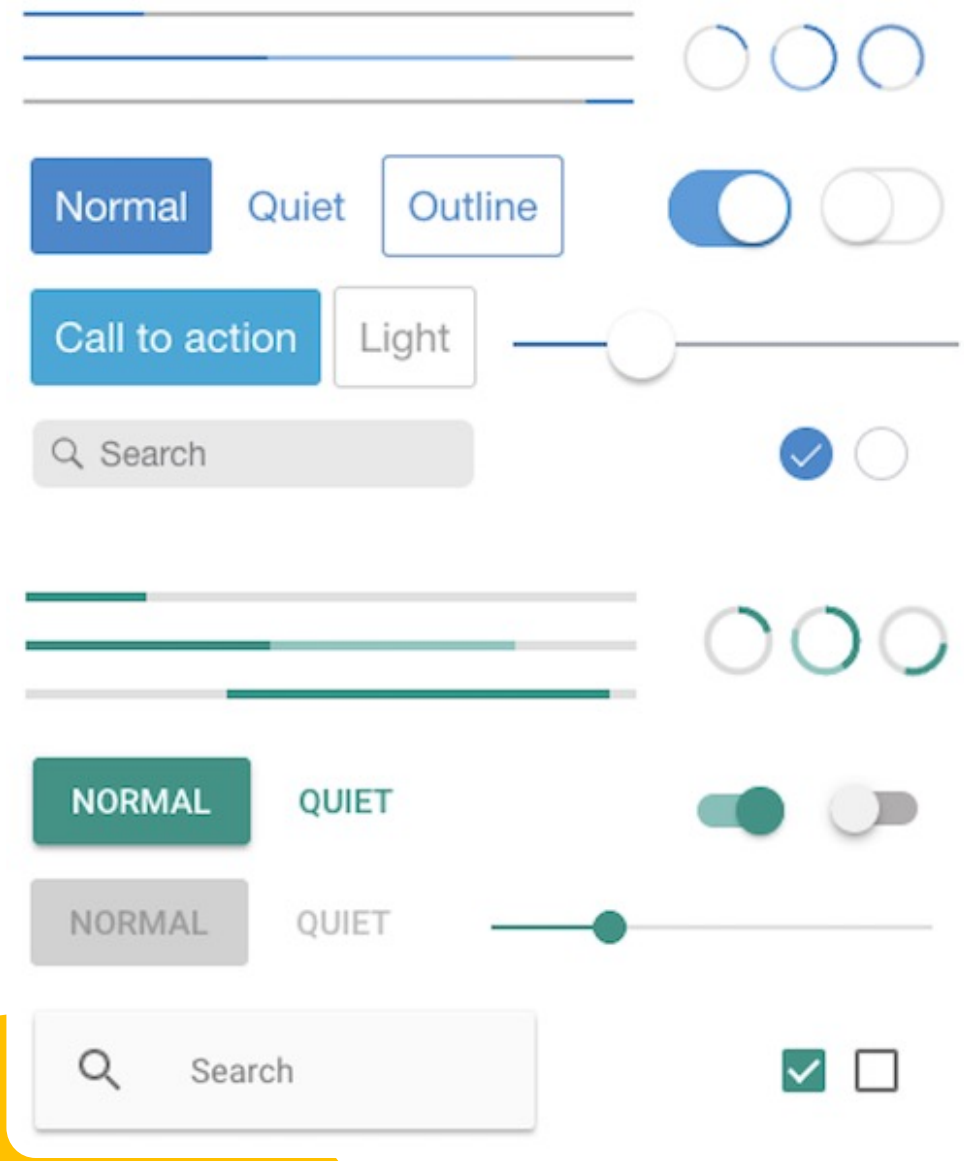


Activities, Views, Layouts



Activities, Views, and ViewGroup

- An activity is an instance of Activity, a class in the Android SDK.
  - An activity is an entry point into your application and is responsible for managing user interaction with a screen of information.
  - Subclass Activity
- View represent the basic building blocks for user interface components
  - Rectangular area on screen
  - Base class for widgets which are used to create interactive UI components (buttons, text fields, etc)
- ViewGroup is the base class for layouts, invisible containers that hold other Views or ViewGroups and define their layout properties
  - ViewGroups don’t display, they orchestrate



GUI Components

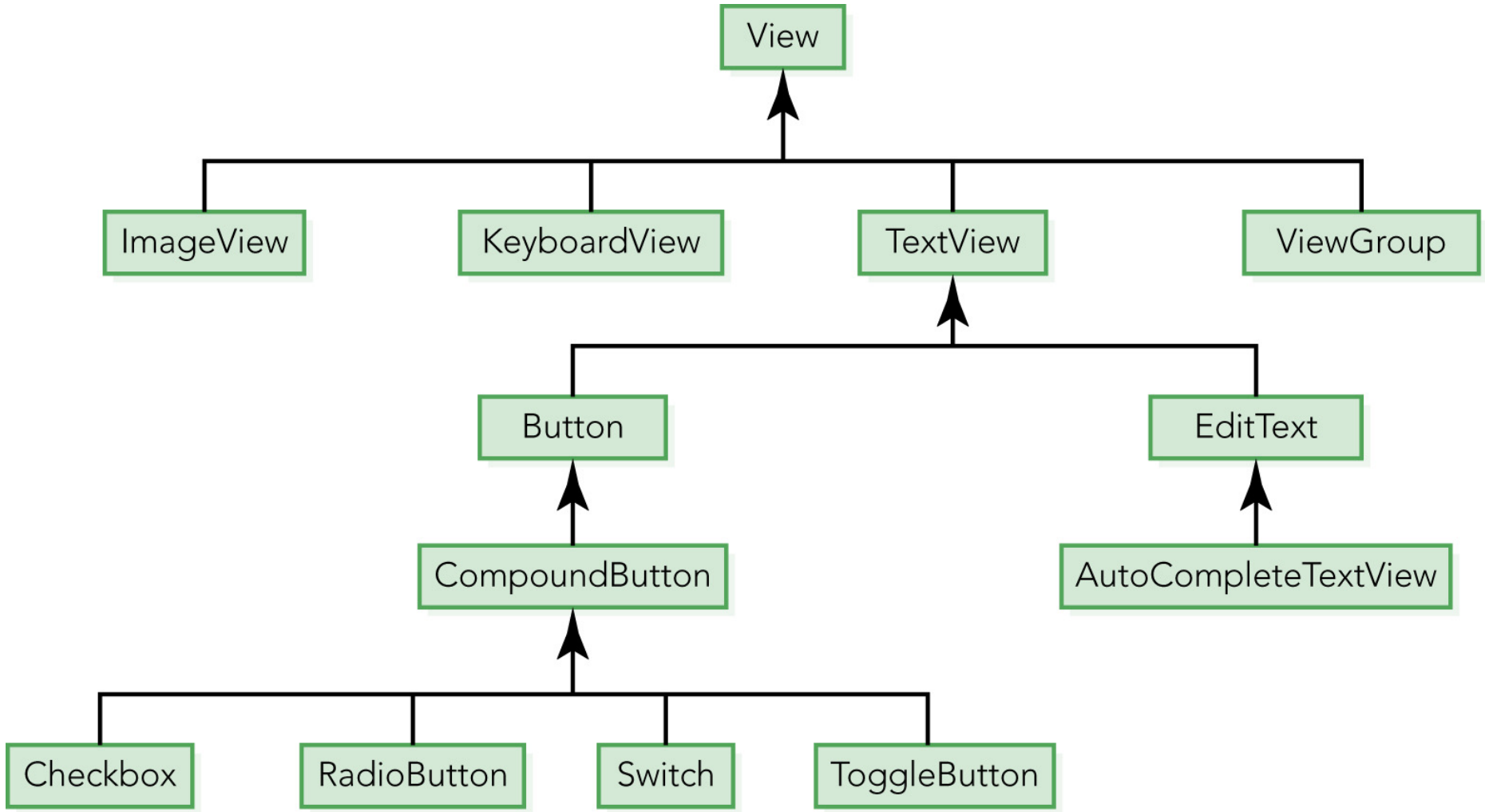
- GUI components can:
  - Display data
  - Capture user input
  - Allow the user to interact with them, which can trigger some action, i.e., a call to a method

Some GUI Components

Component	Class
Panel	View
Keyboard	KeyboardView
Label	TextView
Text field	EditText
Button	Button
Radio button	RadioButton
Checkbox	CheckBox
2-state button	ToggleButton
On-off switch	Switch

GUI Components

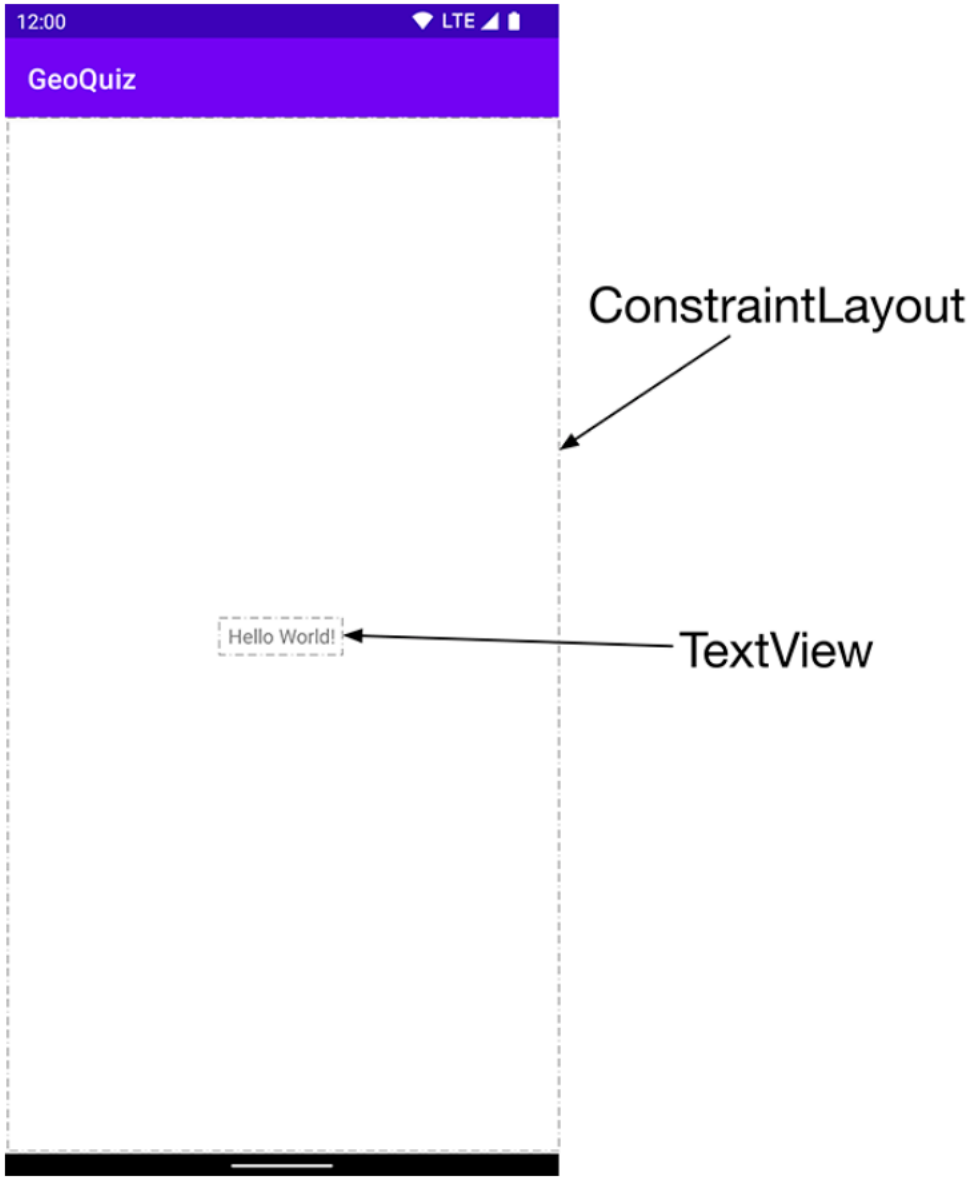
- View is the root GUI component.
- The other GUI components inherit from View, either directly or indirectly.
- View is in the android.view package.
- Most GUI components are in the android.widget package.


View
ImageView
KeyboardView
TextView
ViewGroup
Button
EditText
CompoundButton
AutoCompleteTextView
Checkbox
RadioButton
Switch
ToggleButton

HelloWorld

activity_main.xml

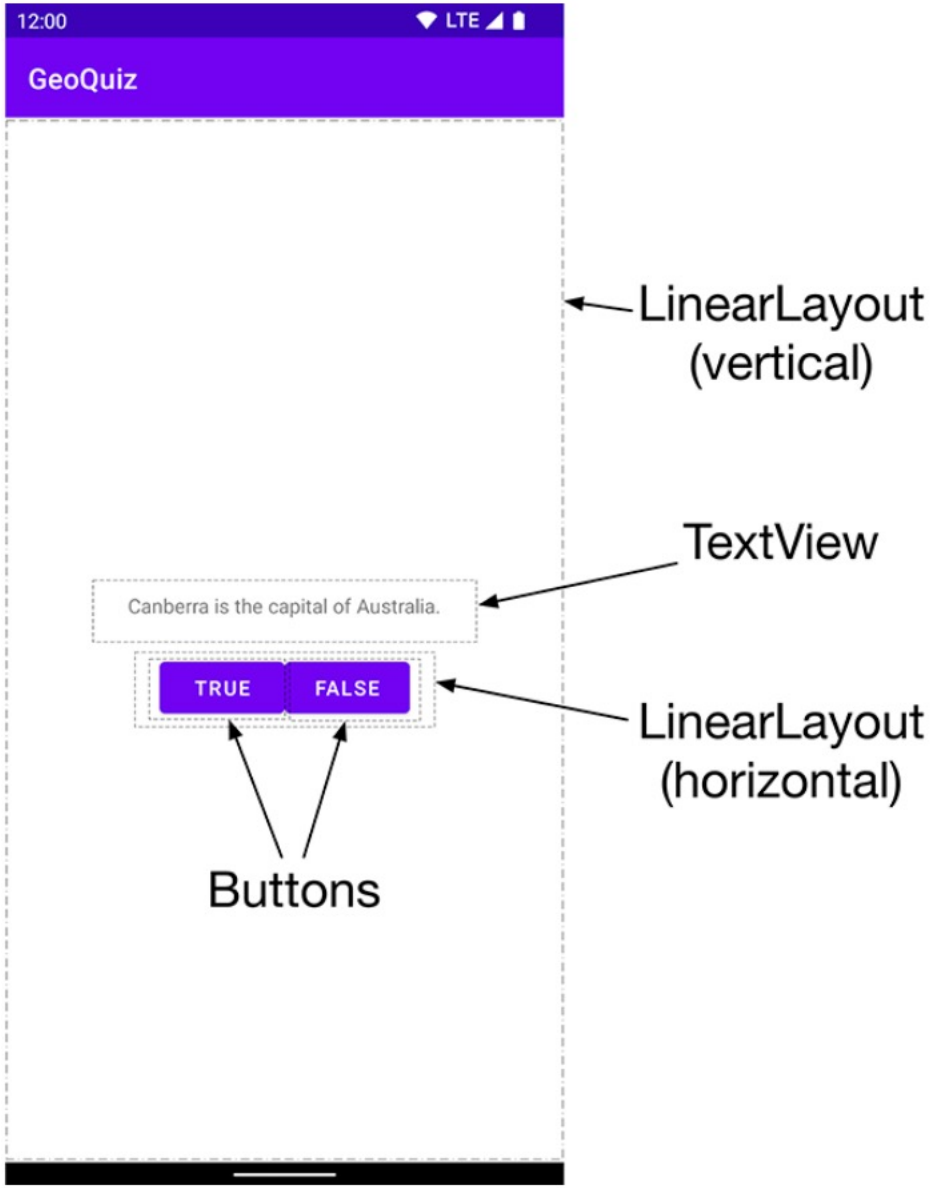
```
<?xml version="1.0" encoding="utf-8"?>
   <androidx.constraintlayout.widget.ConstraintLayout

xmlns:android="http://schemas.android.com/apk/res/android"
            xmlns:tools="http://schemas.android.com/tools"
            xmlns:app="http://schemas.android.com/apk/res-auto"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            tools:context=".MainActivity">

        <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Hello World!"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintLeft_toLeftOf="parent"
                app:layout_constraintRight_toRightOf="parent"
                app:layout_constraintTop_toTopOf="parent"/>

   </androidx.constraintlayout.widget.ConstraintLayout>
```



GeoQuiz

activity_main.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/question_text" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/true_button" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/false_button" />

    </LinearLayout>

</LinearLayout>
```



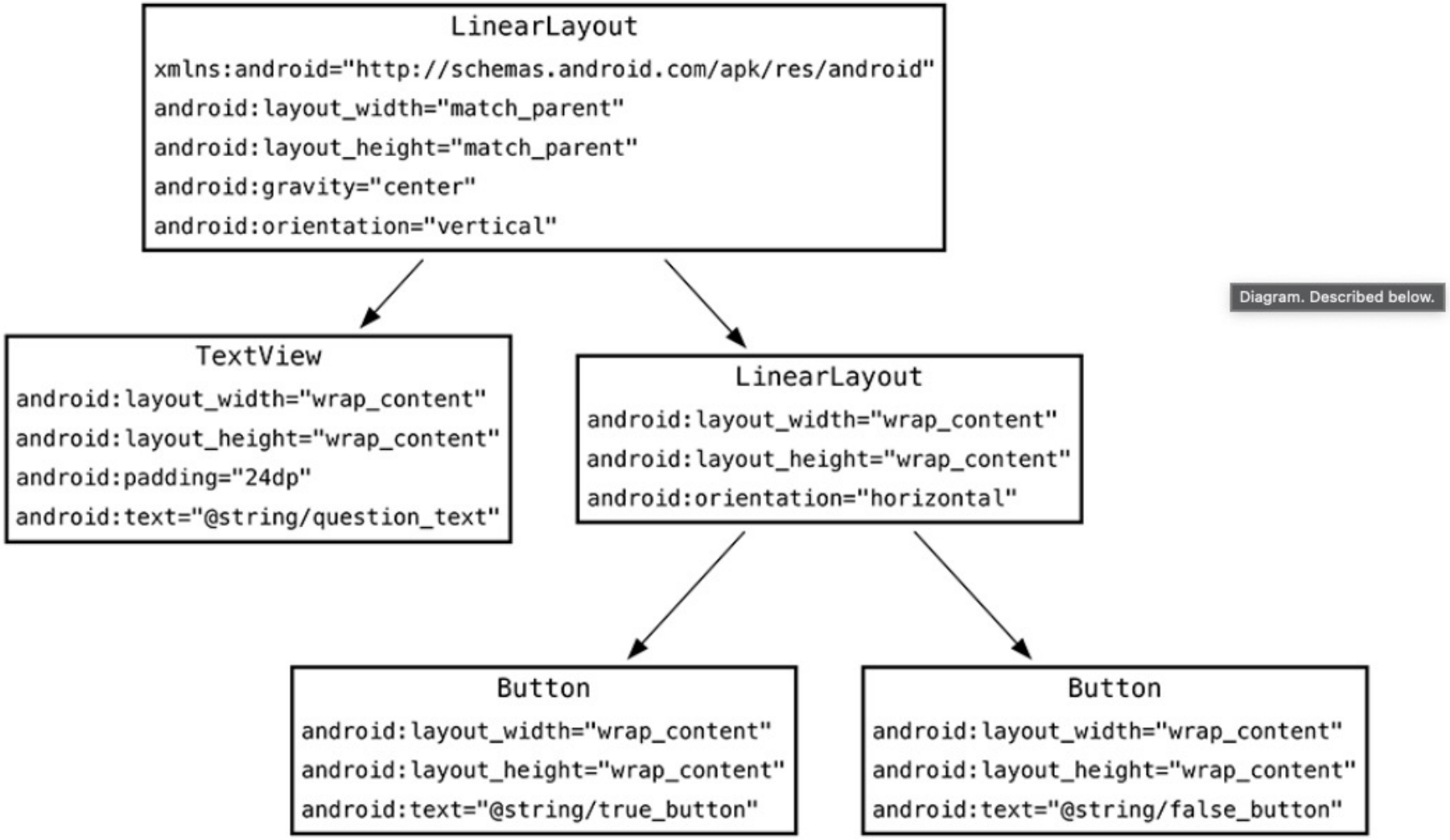


Diagram. Described below.

GeoQuiz

String resources

- In res/values/string.xml

```
<resources>
    <string name="app_name">GeoQuiz</string>
    <string name="question_text">Canberra is the capital of Australia.</string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
</resources>
```

MainActivity.kt

```
package com.bignerdranch.android.geoquiz

    import androidx.appcompat.app.AppCompatActivity
    import android.os.Bundle

    class MainActivity : AppCompatActivity() {

        override fun onCreate(savedInstanceState: Bundle?) {
            super.onCreate(savedInstanceState)
            setContentView(R.layout.activity_main)
        }
    }
```

Called when instance is created

"Inflates" (instantiates the layout) the view according to the resource given

Resources and resource IDs

- Not code – images, audio, XML files, etc
- In the app/res directory
- To access resources, use the resource ID (eg layout.activity_main)

IDs to buttons

```
<LinearLayout ... >

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="24dp"
    android:text="@string/question_text" />

  <LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <Button
      android:id="@+id/true_button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/true_button" />

    <Button
      android:id="@+id/false_button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/false_button" />

  </LinearLayout>

</LinearLayout>
```

ID for true button

ID for false button

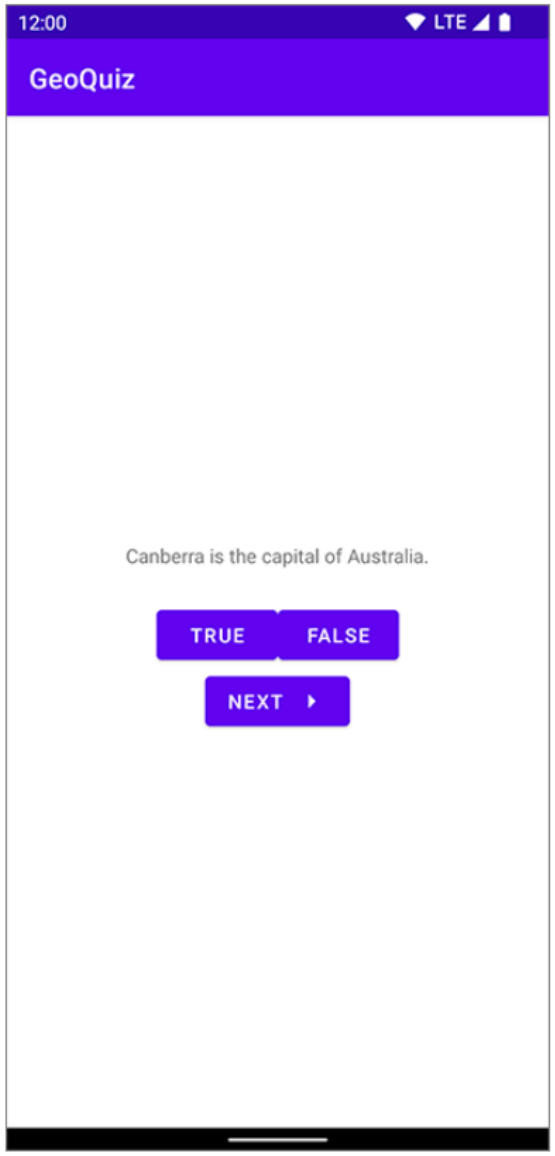
Wire everything up

- Need to add references to the UI element and create them in the code
- In onCreate, lookup and assign the properties
- Set the listeners when the buttons are clicked and popup a toast

```
class MainActivity : AppCompatActivity() {

    private lateinit var trueButton: Button
    private lateinit var falseButton: Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        trueButton = findViewById(R.id.true_button)
        falseButton = findViewById(R.id.false_button)

        trueButton.setOnClickListener { view: View ->
           Toast.makeText(
              this,
              R.string.correct_toast,
              Toast.LENGTH_SHORT
           ).show()
        }

        falseButton.setOnClickListener { view: View ->
           Toast.makeText(
              this,
              R.string.incorrect_toast,
              Toast.LENGTH_SHORT
           ).show()
        }
    }
}
```

Adding a next button

- Say way want to have more than one question to display on our activity, we need to:
  - Create a new class
  - Add a next button
  - Wire that all up



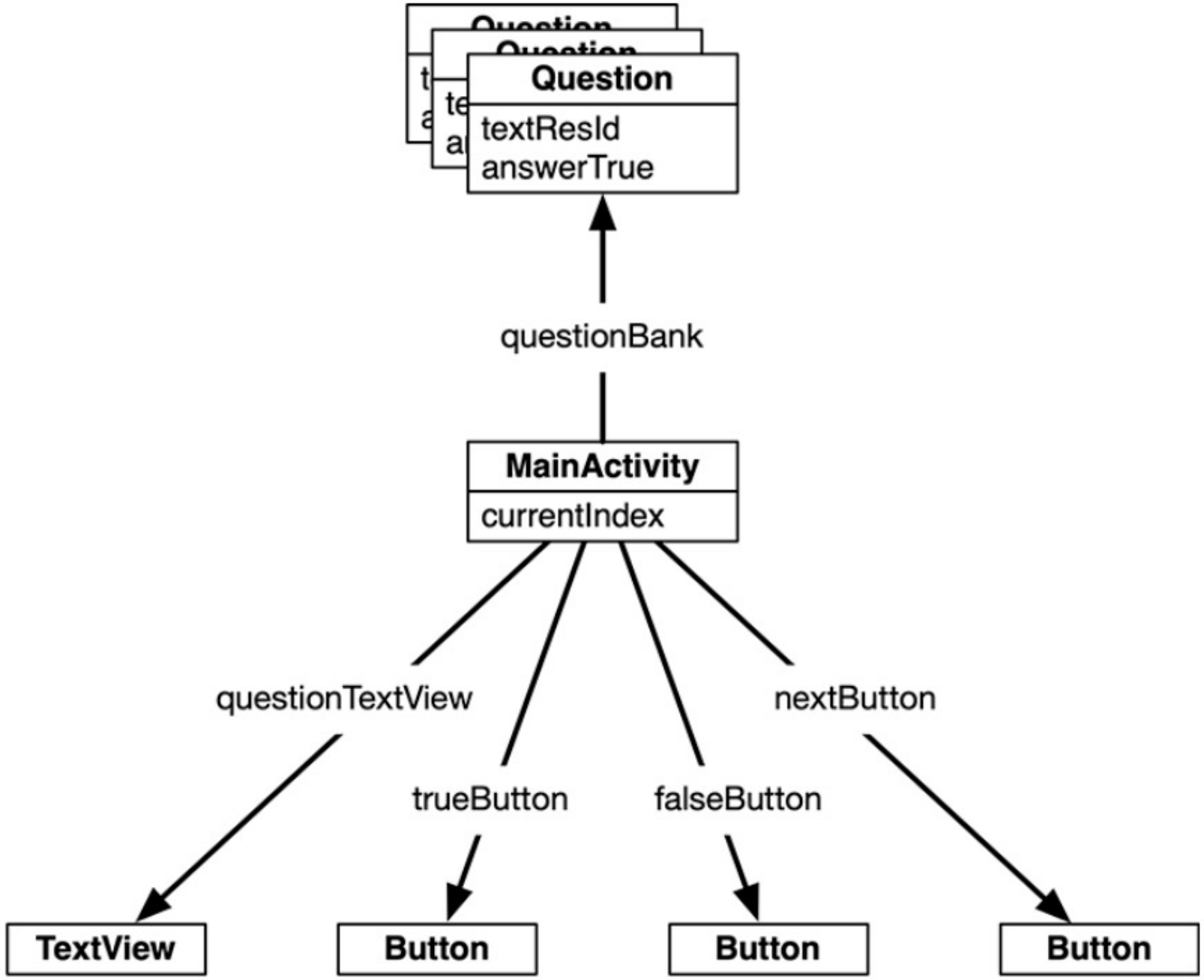
Create a new Kotlin class

- A common way of doing this in software architecture patterns (such as Model-View-Controller, Model-View-Presenter, Model-View-ViewModel, and so on) is the concept of the model.
  - In these architecture patterns, models are classes that contain information that the UI will display.
- Here we're going to create these classes using the data keyword.
  - Doing so clearly indicates that the class is meant to hold model data.
  - Also, the compiler does extra work for data classes that makes your life easier, such as defining useful functions like equals(), hashCode(), and a nicely formatted toString().
- MainActivity will create a list of Question objects.
  - It will then interact with the TextView and the three Buttons to display questions and provide feedback

Question.kt

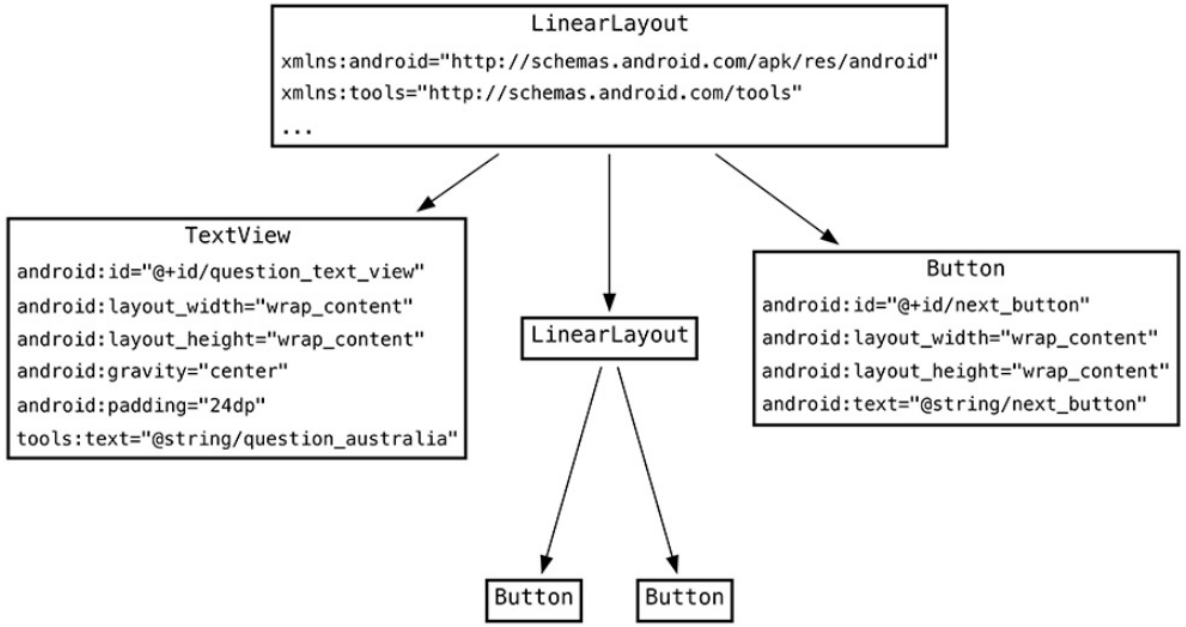
```
import androidx.annotation.StringRes

data class Question(@StringRes val textResId: Int, val answer:
Boolean)
```

textResId is Int and not a String because we will eventually display text to the user, the textResId variable will hold the resource ID (always an Int) of the string resource for a question.


Question
textResId
answerTrue
questionBank
MainActivity
currentIndex
questionTextView
trueButton
falseButton
nextButton
TextView
Button
Button
Button

Updating the layout

- Give the TextView an android:id attribute.
  - This view will need a resource ID so that you can set its text in MainActivity's code.
- Position the TextView's text in the center of the text view by setting gravity to "center".
- Replace the android:text attribute of the TextView with the tools:text attribute and point it to a string resource representing a question using @string/.



res/layout/activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        ... >

    <TextView
        android:id="@+id/question_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:padding="24dp"
        tools:text="@string/question_australia" />

    <LinearLayout ... >
        ...
    </LinearLayout>

    <Button
        android:id="@+id/next_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/next_button" />

</LinearLayout>
```

res/values/string.xml

```
<string name="question_australia">Canberra is the capital of Australia.</string>
<string name="question_oceans">The Pacific Ocean is larger than
  the Atlantic Ocean.</string>
<string name="question_mideast">The Suez Canal connects the Red Sea
  and the Indian Ocean.</string>
<string name="question_africa">The source of the Nile River is in Egypt.</string>
<string name="question_americas">The Amazon River is the longest river
  in the Americas.</string>
<string name="question_asia">Lake Baikal is the world\'s oldest and deepest
  freshwater lake.</string>
```

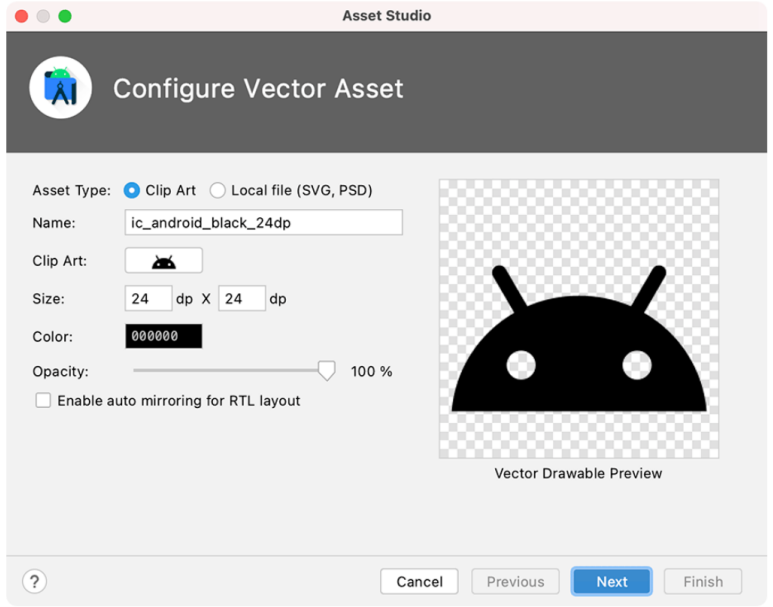
Wiring everything new up

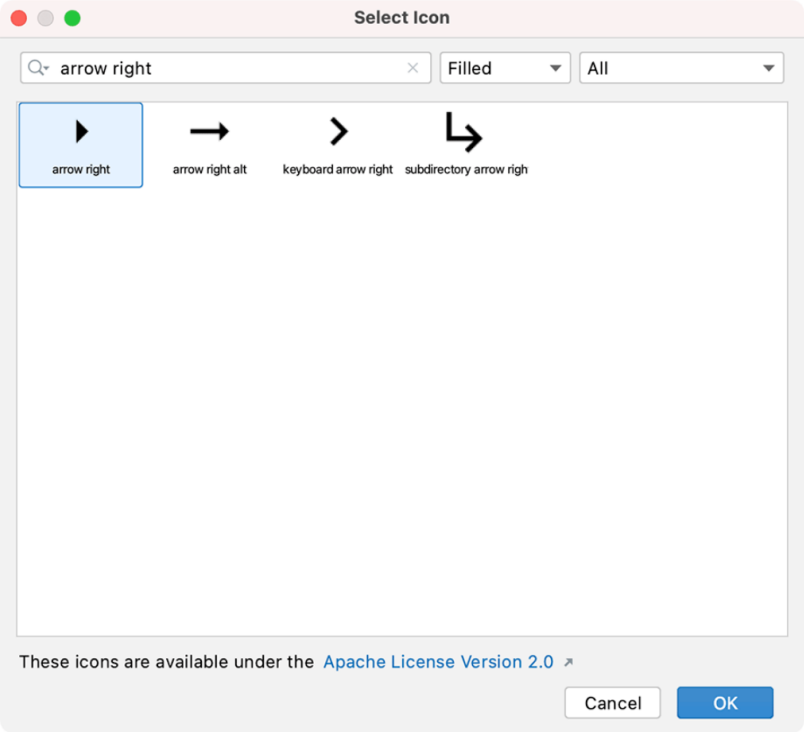
```
class MainActivity : AppCompatActivity() {

    private lateinit var trueButton: Button
    private lateinit var falseButton: Button

    private val questionBank = listOf(
            Question(R.string.question_australia, true),
            Question(R.string.question_oceans, true),
            Question(R.string.question_mideast, false),
            Question(R.string.question_africa, false),
            Question(R.string.question_americas, true),
            Question(R.string.question_asia, true))

    private var currentIndex = 0
    ...
}
```

Call the Question constructor several times and create a list of Question objects.

Adding icons

- Android supports vector graphics – use vector drawables as they scale without compromising visual quality
- Use Vector Asset Studio
  - Can import common file formats (SVG, PSD)
  - Google has icons – Material – free to use (Apache license 2.0)





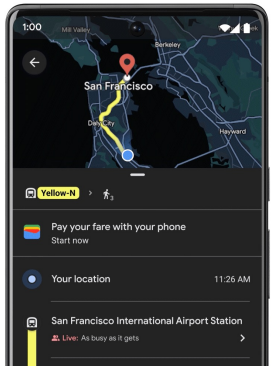
Adding a next button to our GeoQuiz example

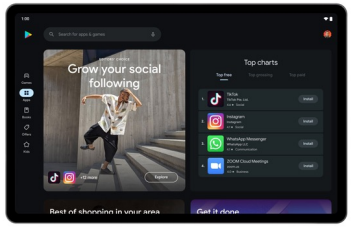
```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:gravity="center"
   android:orientation="vertical">
   ...
   <LinearLayout ... >
      ...
   </LinearLayout>

   <Button
      android:id="@+id/next_button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/next_button"
      app:icon="@drawable/arrow_right"
      app:iconGravity="end" />

</LinearLayout>
```

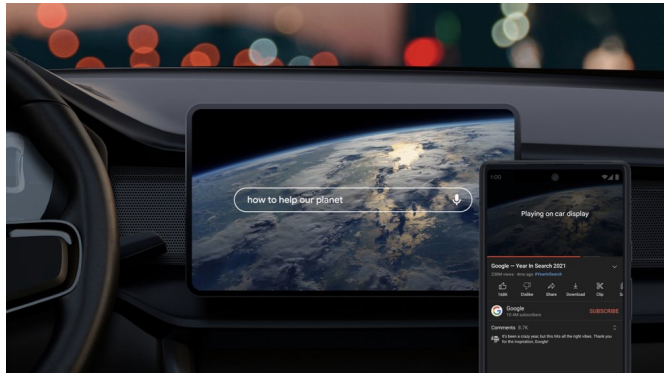
Right arrow reference in activity_main.xml

Screen pixel densities

- Android devices come in many different shapes and sizes
  - Phones 300 – 800 (!) pixels/inch
  - Watches
- Problem: how to display same UI
  - Don’t want giant or tiny buttons
- Answer: density-independent dimension units to get the same size on different screen densities
  - Android translates these units using the device’s density bucket to pixels at runtime
  - Range from low density to high density

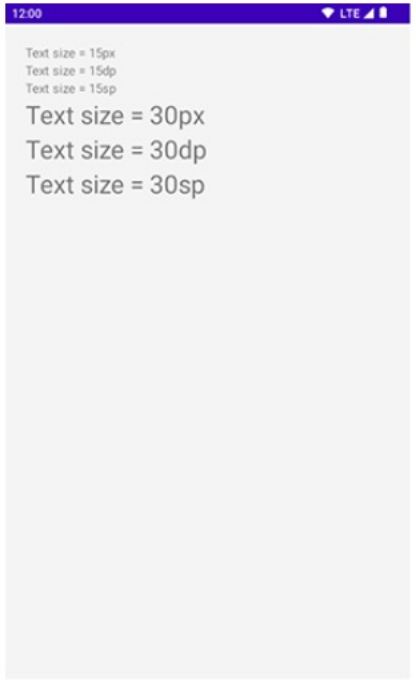






Dimension unites

px	Short for pixel. One pixel corresponds to one onscreen pixel, regardless of the display density. Because pixels do not scale appropriately with device display density, their use is not recommended.
dp	Short for density-independent pixel and usually pronounced “dip.” You typically use this for margins, padding, or anything else for which you would otherwise specify size with a pixel value. One dp is always 1/160 of an inch on a device’s screen. You get the same size regardless of screen density: When your display is a higher density, density-independent pixels will fill a larger number of screen pixels.
sp	Short for scale-independent pixel. Scale-independent pixels are density-independent pixels that also take into account the user’s font size preference. You will almost always use sp to set display text size.
pt, mm, in	These are scaled units, like dp, that allow you to specify interface sizes in points (1/72 of an inch), millimeters, or inches. However, we do not recommend using them: Not all devices are correctly configured for these units to scale correctly.



MDPI

HDPI

HDPI, large text

For next week

- REMINDER: Please give Git access to me and the graders
- Read ch 2-4
- Make sure you signup for Piazza and can access Blackboard
- Assignment 2
- Start thinking about a group you want to create for the project
  - 2-3 people
  - Think about some project proposals