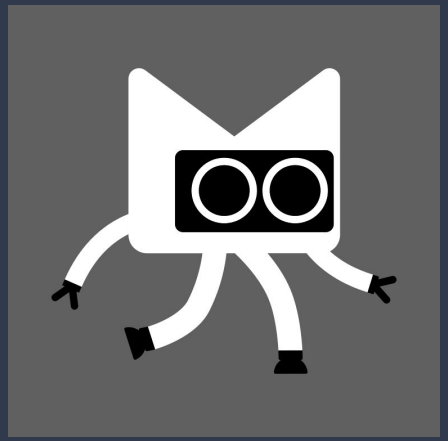
Kotlin Programming Language

Azan Nazar, Jason Dunn,
Jason Scott, Matt Kubisa

Overview



The official Kotlin mascot

- Supports multiple paradigms
  - Often written imperatively, like Java
  - Also supports functions for functional programming
- Compiled before execution
- Growing in popularity: 2nd most popular JVM-focused language

Motivation

- Created by JetBrains in 2010
  - Make IDEs for various languages with different paradigms
- “Future-proofed” Java alternative
  - Introduces new functionality
  - Open source
- Backwards compatible with Java code for easy migration
  - Compatible with Java libraries

Applications

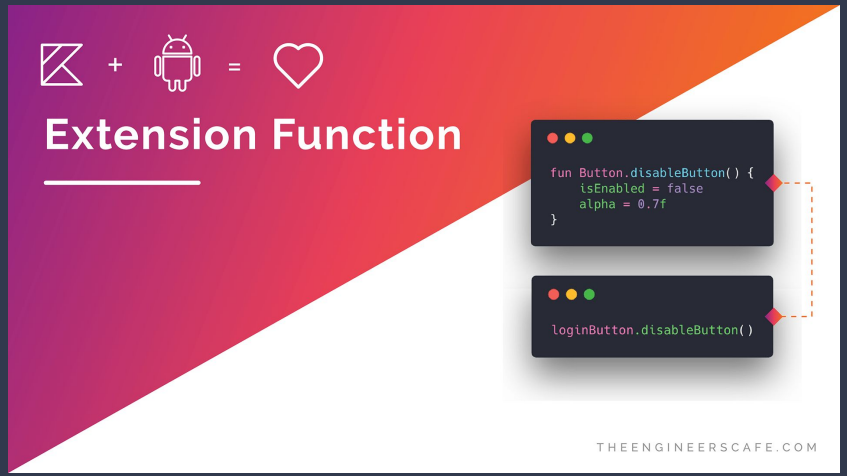
“Now when we want to start a Jetpack Library, we are writing it in Kotlin unless we have a very, very, very good reason not to do that. It’s clear that ***Kotlin is the first-class language****.”*

- Yigit Boyar, Google developer

- Android apps
  - Used by 60% of developers
  - Jetpack Compose: Interface for quickly developing Android UI
- Game Development
  - Minecraft mods
  - KorGE Game Engine

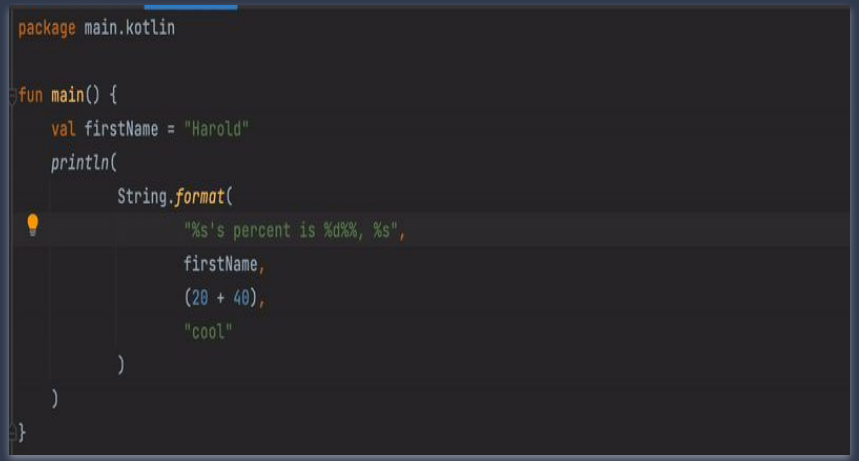
Kotlin Features

Kotlin vs. Java Features (1)



- **Extension functions**: allow existing classes to extend new functionality without modifying the source code.
- **Null safety**: nullable and non-nullable types helps in preventing null pointer exceptions during compile time.
- **Coroutines**: mix of object oriented/functional programming
- **Operator Overloading**: allows custom implementation for predefined operators on types

Kotlin vs. Java Features (2)



```kotlin
package main.kotlin

fun main() {
    val firstName = "Harold"
    println(
        String.format(
            "%s's percent is %d%%, %s",
            firstName,
            (20 + 40),
            "cool"
        )
    )
}
```

```kotlin
when (x) {
    is Int -> print(x + 1)
    is String -> print(x.length + 1)
    is IntArray -> print(x.sum())
}
```

- **String templates**: allows embedded expressions within string literals, making interpolation of string more readable and concise.
- **Smart casts**: allows for automatic casting of types in certain control flows where the compiler inserts (safe) casts automatically when necessary
- **Primary constructor**: declared in class headers, simplifying syntax for classes w/constructor param.
- **Data Classes**: member functions automatically generated by the compiler

Why Choose Kotlin?

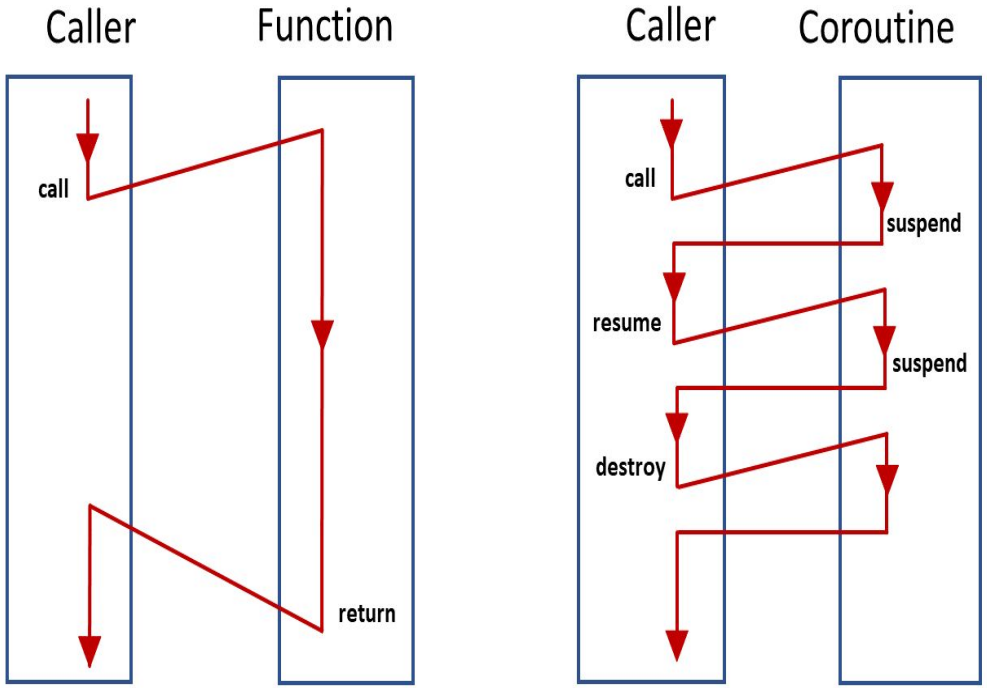
Exclusive Features (not present in Java)

- **Data classes** w/ automatically generates additional member functions to print, compare, copy, etc
- **Separate interfaces** for read-only and mutable collections
- **Proper function types**, as opposed to Java's SAM-conversions
- **Null references** are controlled by the type system.
- **Site variance** without wildcards
- **Arrays** in Kotlin are invariant

Coroutines

Kotlin incorporates coroutines where the execution of a program is suspended and resumed enabling cooperative multitasking

Ex: Similar to multithreading except run during lang runtime and independent of each other



Coroutines occur asynchronously and enable efficient multitasking of executing programs

Operator Overloading

Kotlin also allows altering predefined operators on types to enable efficient and more-legible code

Ex: plus, subtract, multiplication, etc operators

```
data class CustomDate(val year: Int, val month: Int, val day: Int) {
    override fun toString(): String =
        "$year/$month/$day"
}

operator fun CustomDate.plus(days: Int): CustomDate {
    val totalDays = year * 365 + month * 30 + day + days
    val newYear = totalDays / 365
    val newMonth = (totalDays % 365) / 30
    val newDay = (totalDays % 365) % 30
    return CustomDate(newYear, newMonth, newDay)
}

// Usage
val currentDate = CustomDate(2023, 8, 12)
val futureDate = currentDate + 30
println("Current date: $currentDate") // Current date: 2023/8/12
println("Date 30 days from now: $futureDate") // Date 30 days from now: 2023/9/12
```

Custom plus operator for CustomDate allows adding the ‘dates’ opposed to Java concatenation

Concise Syntax

Kotlin attempts to be more-human readable than Java, by using shorter and fewer keywords.

Ex: implicit public scope and variable types

```java
public class MyClass {
   private int myField;

   public MyClass(int myField) {
      this.myField = myField;
   }

   public int getMyField() {
      return myField;
   }

   public void setMyField(int myField) {
      this.myField = myField;
   }
}
```

```kotlin
class MyClass(private var myField: Int) {
   fun getMyField() = myField
   fun setMyField(value: Int) { myField = value }
}
```

Equivalent class definitions written in both Java (top) and Kotlin (bottom).

Protection Against Null Point Exceptions

“I call it my billion-dollar mistake. It was the invention of the null reference in 1965... This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.”

- Tony Hoare speaking at a software conference in 2009

- Kotlin type system is designed to prevent null references
  - Nullable
  - Non-nullable
- Several ways to access properties of nullable types
  - Check for null in conditions
  - Nullable receiver
  - Safe calls
- Operators designed to handle null references
  - Elvis operator
  - The !! operator

Non-nullable type throws an exception error, nullable type (declared by ?) works fine

```
fun main() {

    //non-nullable
    var a: String = "abc" // Regular initialization means non-nullable by default
    a = null // compilation error

    //nullable
    var b: String? = "abc" // can be set to null
    b = null // ok
    print(b)

}
```

Allowed to call a method or access properties on **'a'**. Not allowed on **'b'**, since it can be null

```
...

val l = a.length // good

val l = b.length // error: variable 'b' can be null
```

Using safe calls:

```
val a = "Kotlin"
val b: String? = null
println(b?.length)
println(a?.length) // Unnecessary safe call

...

bob?.department?.head?.name //safe calls useful in chains
```

Nullable and Non-nullable Types

Check if b is null, compiler keeps track of this information, allowing you to use b within the scope of the if statement

```
val l = if (b != null) b.length else -1

...

fun main() {

    val b: String? = "Kotlin"
    if (b != null && b.length > 0) {
        print("String of length ${b.length}")
    } else {
        print("Empty string")
    }

}
```

Does not throw an exception, since toString() is defined on a nullable receiver

```
val person: Person? = null
logger.debug(person.toString()) // Logs "null", does not throw an exception
```

Accessing Properties of Nullable Types

Functional Programming Aspects

```
fun printType(obj: Any) {
    when (obj) {
        is String -> println("String")
        is Int -> println("Int")
        is Double -> println("Double")
        else -> println("Unknown")
    }
}
```

Example from Andrew Couter, published in “Level Up Coding”

- Kotlin offers several functional programming features
  - Lambda Expressions
  - Higher Order functions
  - Extension functions
- Pattern matching using ‘when’ expressions

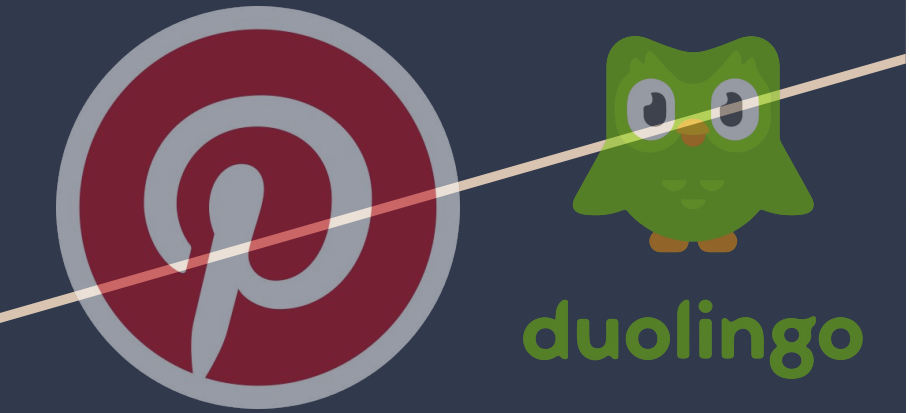
Notable Projects

Notable Kotlin projects

- **Pinterest**
- Announced transition to 100% Kotlin in 2016
- Earlier than most other companies that followed
- Pinterest team noted *slower* compile time on clean builds, but *faster* compile time on incremental builds
- In 2016, Google did not officially support Kotlin as an official Android language

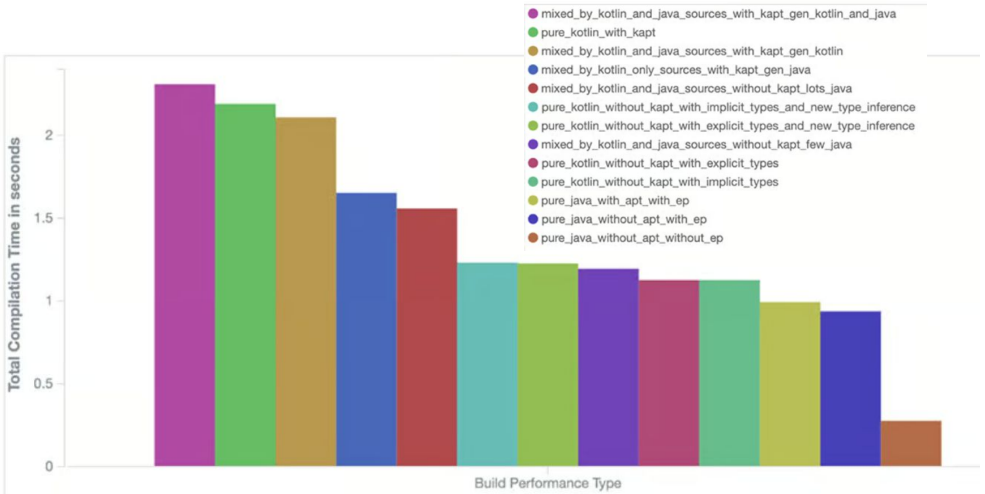
Notable Kotlin projects

Uber



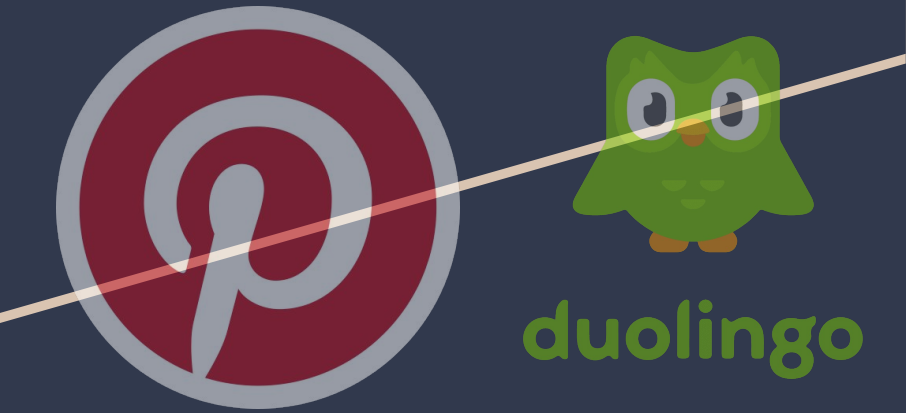
duolingo

- **Uber**
- Had over 20 android applications with 2,000+ modules in mostly Java
- Needed to make sure it was worth it to make this switch
- Conducted research with jetBrains to see if it would be worth it

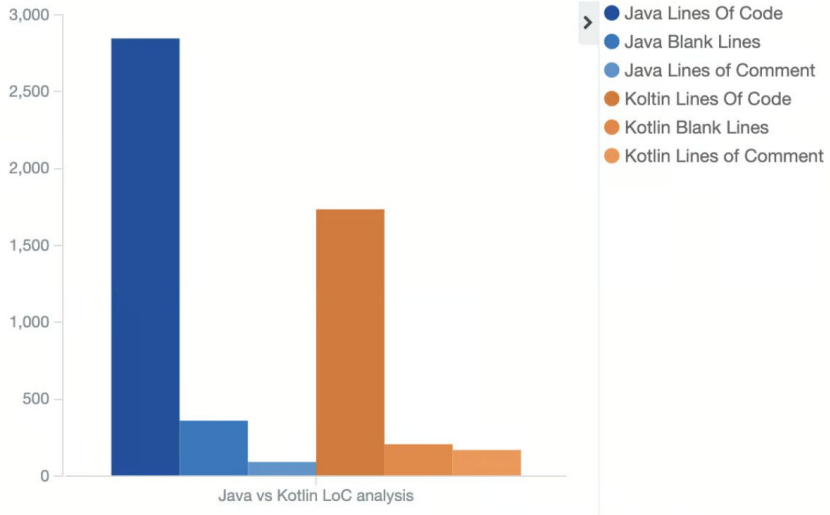


Notable Kotlin projects

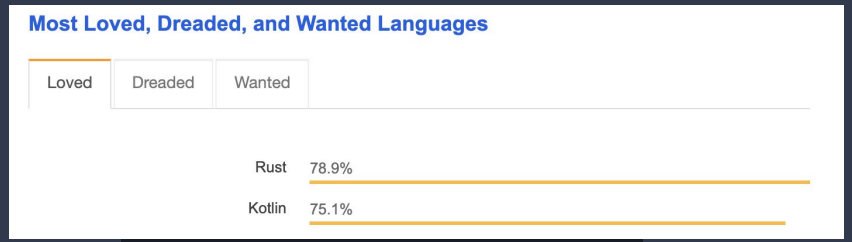
Uber



- **Uber**
- Kotlin resulted in 40 percent fewer lines of source code than Java
- Found that increased safety/manageability +
- fewer LoC made it worth it

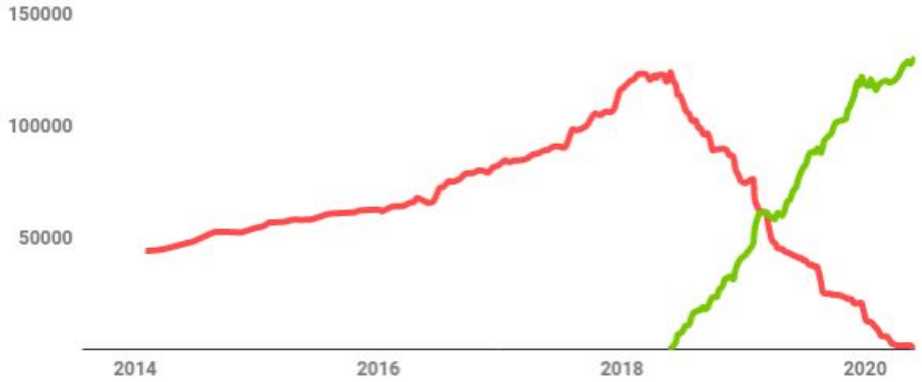


Notable Kotlin projects



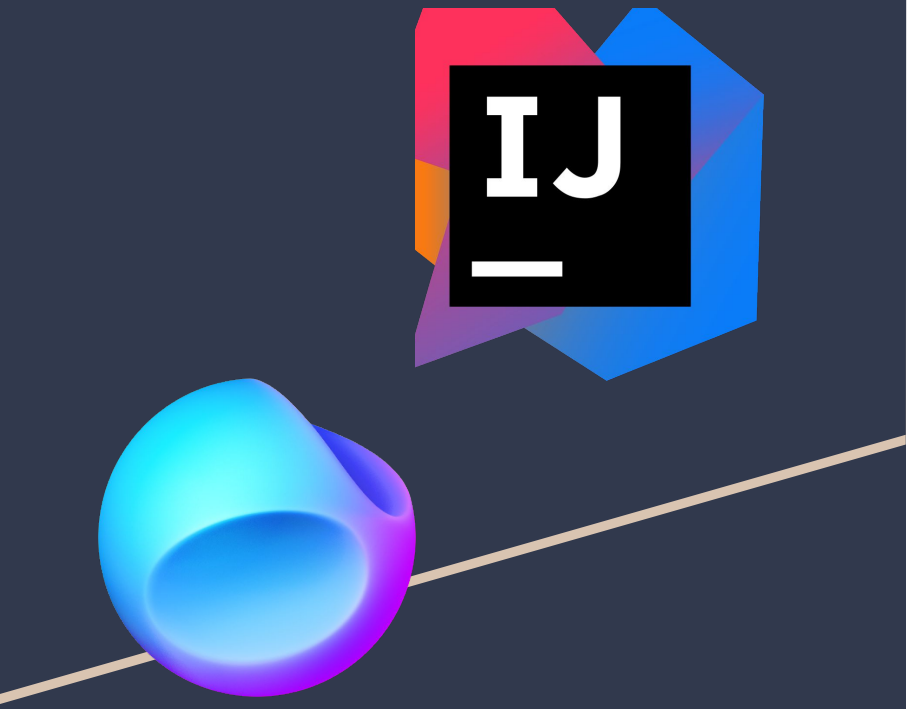
duolingo

- **Duolingo**
- Swapped to Kotlin with developer's workflow in mind
- More productive
- More stable
- More "loved" (2nd most loved in 2018)



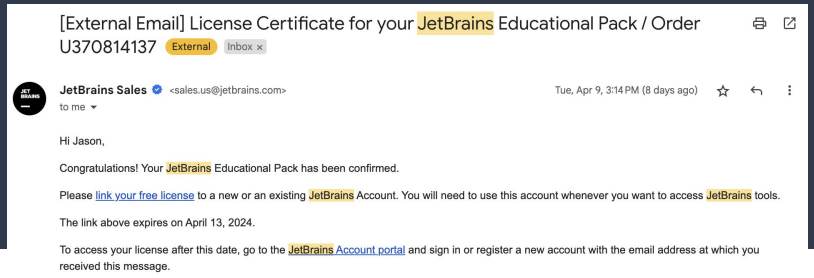
Lines of code of Java (Red) vs. Kotlin (Green) over the years

How to start using Kotlin



- If you want to work with the official Kotlin Plugin you can start by downloading IntelliJ IDEA, Android Studio, or Jetbrains Fleet
- Created and updated for ease of development for Kotlin & Java
- Fleet is a more all around IDE with support for 15+ languages natively

Take advantage of being a Student!


[External Email] License Certificate for your JetBrains Educational Pack / Order U370814137 External Inbox ×

JetBrains Sales <sales.us@jetbrains.com> Tue, Apr 9, 3:14 PM (8 days ago)
to me

Hi Jason,

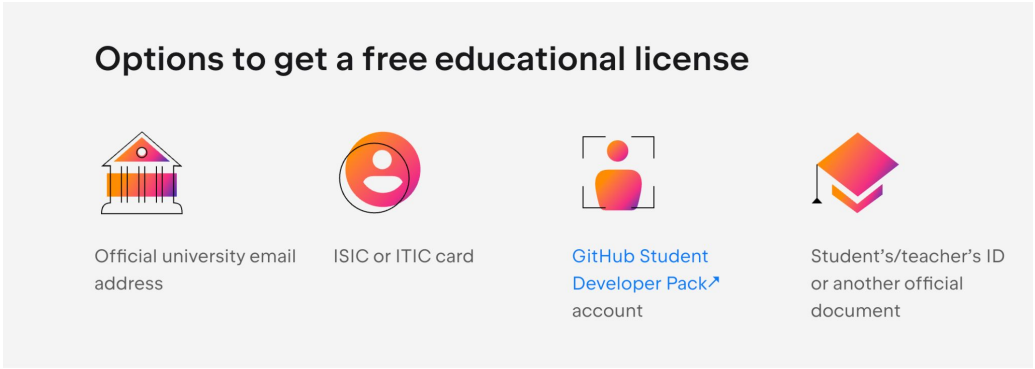
Congratulations! Your JetBrains Educational Pack has been confirmed.

Please link your free license to a new or an existing JetBrains Account. You will need to use this account whenever you want to access JetBrains tools.

The link above expires on April 13, 2024.

To access your license after this date, go to the JetBrains Account portal and sign in or register a new account with the email address at which you received this message.

- To get IntelliJ idea or Fleet for free, all you need to do is sign up with your college email account or send an official document
- Idea alone is 170$, so might as well use your student status to get some free software



- Once you've set up your IDE, consider a framework like Spring or Ktor to create your own Kotlin server-side application
- You've been forced to learn Java here already so you have a good foundation

Android App Live Demo

Android Studio

- Cross-platform IDE for developing Android apps
- Supports Kotlin and Java; most development shifting towards Kotlin
- Can interface with Android device over USB for testing

App Description

- User provides IP Address to ping
- App pings the address and returns response time
- Two-window interface: “enter app” screen and enter IP address screen
- Uploaded over USB to Samsung Galaxy S7 running Android 8.0 Oreo

NavController function at start of code controls which screen is being displayed:

```
@Composable
fun PingtoolApp() {
    val navController = rememberNavController()

    NavHost(navController = navController, startDestination = "main_screen") { this: NavGraphBuilder
        composable( route: "main_screen") { it: NavBackStackEntry
            MainScreen(navController = navController)
        }
        composable( route: "second_screen") { it: NavBackStackEntry
            SecondScreen(navController = navController)
        }
    }
}
```

When the “Ping” button is pressed, a built-in InetAddress command runs the ping operation:

```
Button(
    onClick = {
        // Perform ping operation
        val ipAddress = pingMe.text // copy pingMe into ipAddress variable
        thread {    // start ping operation as a background process in a new thread
            val startTime = System.currentTimeMillis()  // get the current time
            val reachable = InetAddress.getByName(ipAddress).isReachable( timeout: 5000) // Ping with 5 seconds timeout
            val endTime = System.currentTimeMillis()    // get the current time (again)
            if (reachable) {    // if the ping operation returned a value...
                val timeElapsed = endTime - startTime   // calculate time elapsed
                responseTime = "Response Time: $timeElapsed ms" // print time elapsed to screen (temporary)
            } else {
                responseTime = "Ping Failed"    // case: nothing to read in "reachable"
            }
        }
    }
) { this: RowScope
    Text(text = "Ping")
}
```

Code Structure

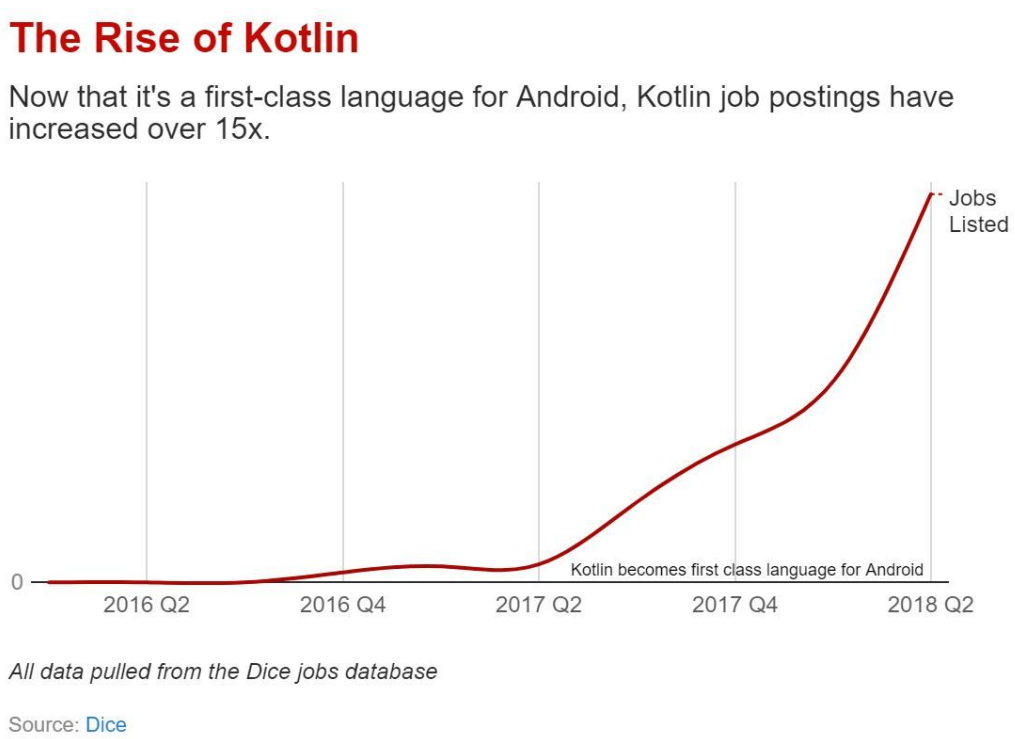
Composable Functions

- Composable functions define each screen and the NavController
- Can be written in XML or Kotlin, but Kotlin provides more flexibility
- All app code shown is in MainActivity.kt Kotlin file

Let’s see it run!

Conclusion / Future of Kotlin

- 14 years in, Kotlin proved itself to be a very solid and relevant language
- (unlike Carbon for C++)
- Accomplished goal to make a safer more maintainable Java
- Array of cool new features over Java
- Null Pointer Safety being among the most important
- However Java will not be completely replaced due to speed and legacy it holds



QUESTIONS?