

AD340 Mobile Application development

...

Week 01

Outline

Course Introduction

Why Android Development with Kotlin

Kotlin Basics

Class Setup

6:00 pm - 6:25 pm : session 1

6:25 pm - 6:30 pm : break

6:30 pm - 6:55 pm : session 2

6:55 pm - 7:00 pm : break

7:00 pm - 7:25 pm : session 3

7:25 pm - 7:30 pm : break

7:30 pm - 7:55 pm: session 4

7:55 pm - 8:00 pm: break

8:00 pm - 8:25 pm: session 5

8:25 pm - 8:40 pm : Q&A, extra help

About Me

Name: BC Ko (He/Him)

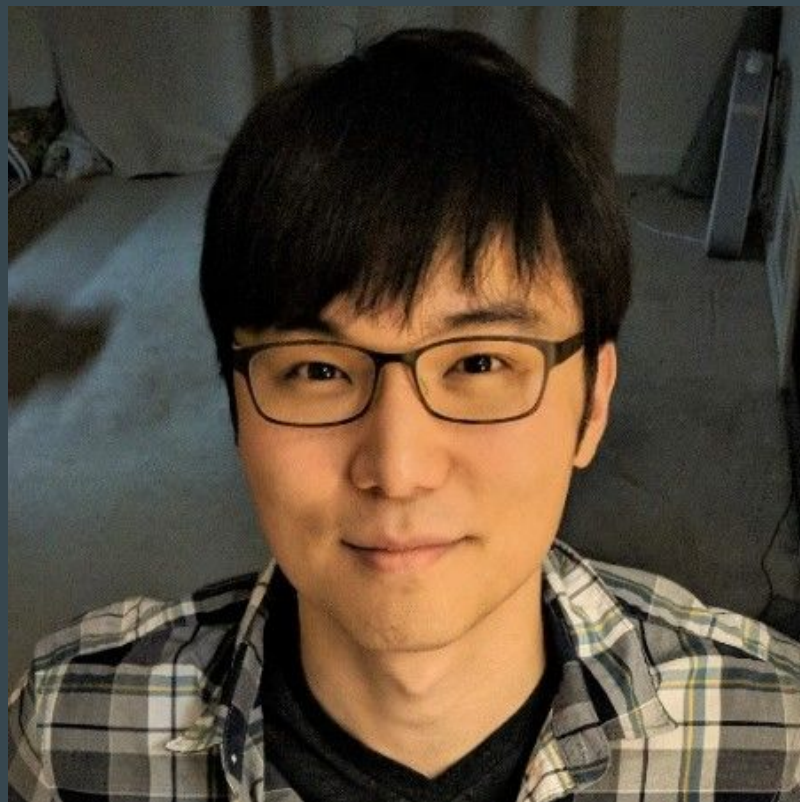
Winter 2023

- AD 320 Web Application Development
- AD 450 Data Science Development

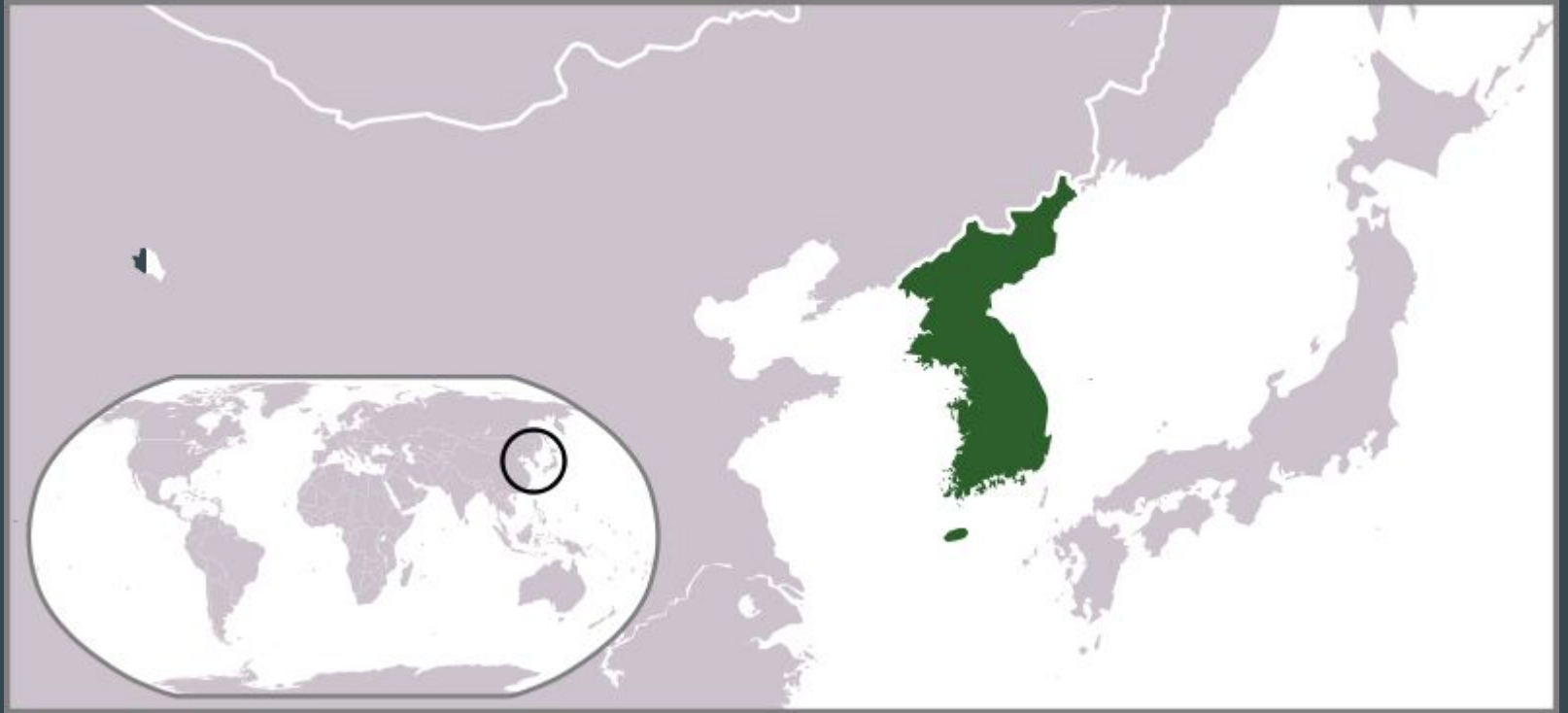
Spring 2023

- AD 340 Mobile Application Development
- AD 410 Web Application Practicum
- AD 470 Data Science Practicum

Reach out to me anytime!



Born in Korea



Not North



But South

The tallest building looks like Sauron tower
from the Lord of the Rings.

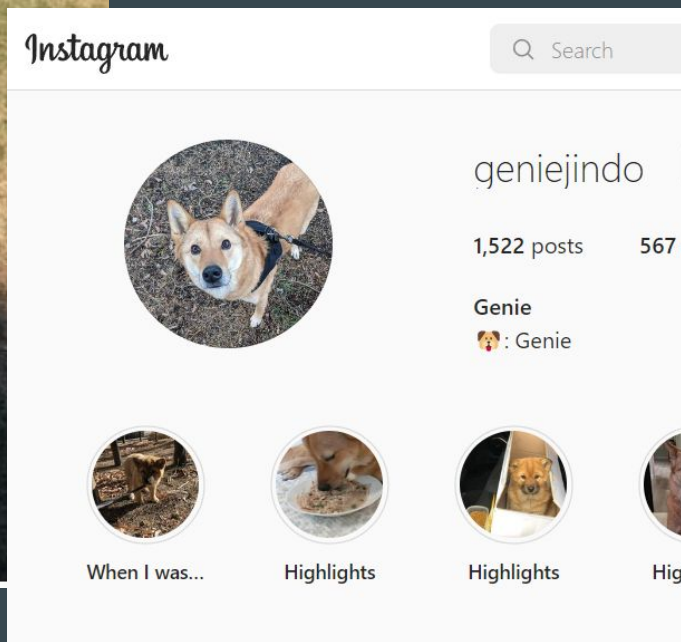
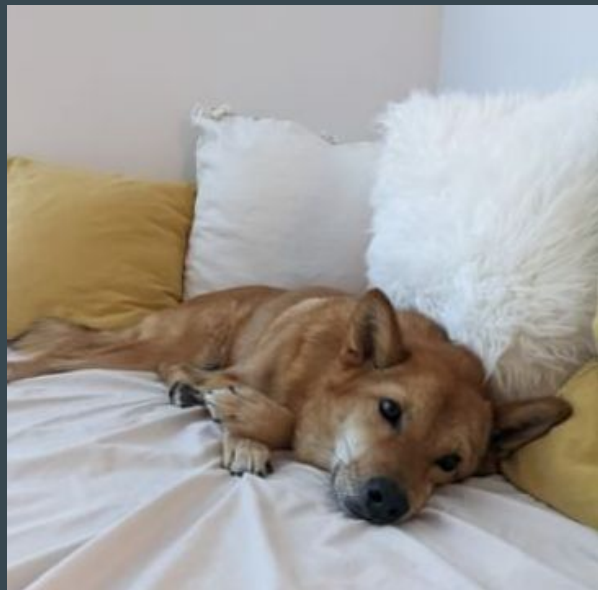


Gangnam style guy lives in South Korea



What do you do when you are not studying?

Genie might show up unexpectedly



Work Experience

Seattle Central & North Seattle

Military

Consumer Finance

Edtech

Social Media Analytics

Introduce yourselves

Name

Pronouns

What you did during the Spring break

Summer vacation plan

Grading

Based on completion of learning activities
due every Sunday night at 11:59 PM Pacific Time.

late assignments may be marked down by

- 10% (less than 7 days late)
- 20% (more than 7 days late).

You are allowed to resubmit

Percentage to GPA Conversion Chart

Same Percentage to GPA Conversion Chart

What is different in Spring (Round up)

Winter Quarter

- 95 → 4.0
- 94.9 → 3.9
- 94.5 → 3.9

Spring Quarter

- 94.9 → 4.0
- 94.5 → 4.0
- 94.4 → 3.9

Percentage-to-GPA Conversion Chart

Name:	Range:
4.0	100 % to 95.0%
3.9	< 95.0 % to 94.0%
3.8	< 94.0 % to 93.0%
3.7	< 93.0 % to 92.0%
3.6	< 92.0 % to 91.0%
3.5	< 91.0 % to 90.0%
3.4	< 90.0 % to 89.0%
3.3	< 89.0 % to 88.0%
3.2	< 88.0 % to 87.0%
3.1	< 87.0 % to 86.0%
3.0	< 86.0 % to 85.0%
2.9	< 85.0 % to 84.0%
2.8	< 84.0 % to 83.0%
2.7	< 83.0 % to 82.0%
2.6	< 82.0 % to 81.0%

Questions?

Syllabus

Course structure

etc

Android Development with Kotlin

What you will learn

How to build a variety of Android apps in Kotlin

Kotlin language essentials

Best practices for app development



Why Mobile Development?

Mobile devices are becoming increasingly commonplace

Mobile apps connect users to information and services that can improve their quality of life

Many industries have yet to be revolutionized through mobile, and offer great opportunities for new businesses and solutions



Android

Open source mobile platform

- You can develop in Windows, Mac, Linux machines

2.5 billion monthly active Android devices*

2+ billion monthly active Google Play users*

* August 2020



Available across different form factors

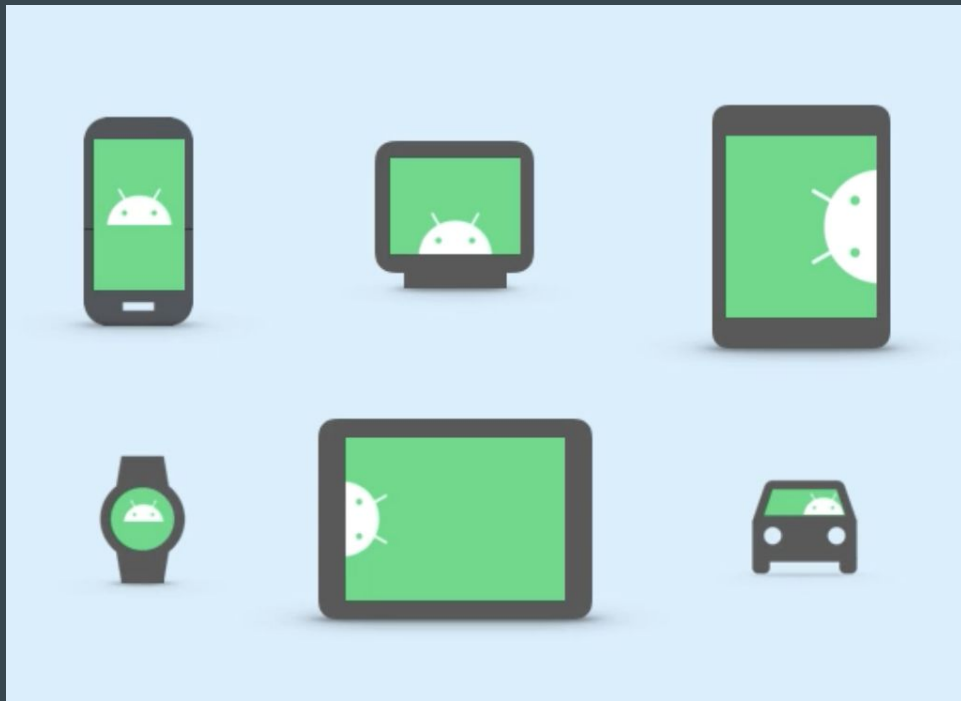
Phones

Tablets

TVs

Watches

Cars



Build Android apps in Kotlin

In 2017, Kotlin was officially announced as another supported language on Android

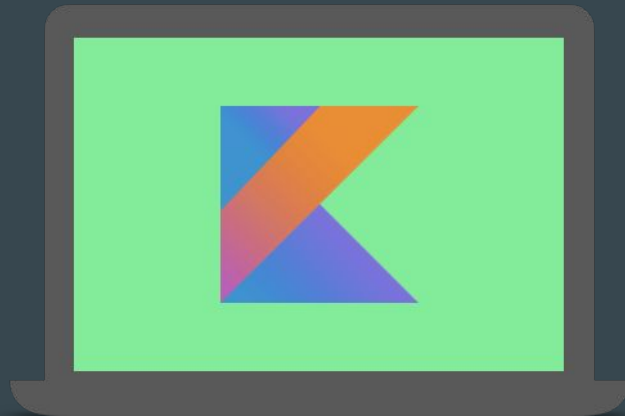
In 2019, Google is making Android development Kotlin-first

- New platform features and libraries will **first be made available in Kotlin**
- New developers are encouraged to build their Android apps in Kotlin



Kotlin

- Open-source statically typed language
 - Supports functional & OOP programming styles
 - Interoperability
 - Safety
- #4 most-loved programming language among developers (2020 Stack Overflow Developer Survey)
- Over 70% of the top 1000 Android apps contain Kotlin code



Benefits of Kotlin

Expressive and concise

- Less boilerplate code → quicker and maintenance is easier
- Type inference → omit data types if the compiler can infer it

Safer code

- Avoid NullPointerExceptions in your code

Interoperable

- Use existing Java classes and libraries with Kotlin.

Structured Concurrency

- Coroutines for asynchronous programming. Simplifies network calls or accessing the database

Kotlin Basics

Install IntelliJ IDEA - Community Edition

<https://www.jetbrains.com/idea/>

Download IntelliJ IDEA

Windows

macOS

Linux

Ultimate

The Leading Java and Kotlin IDE

Download

.exe ▼

Free 30-day trial available

Community Edition

The IDE for pure Java and Kotlin development

Download

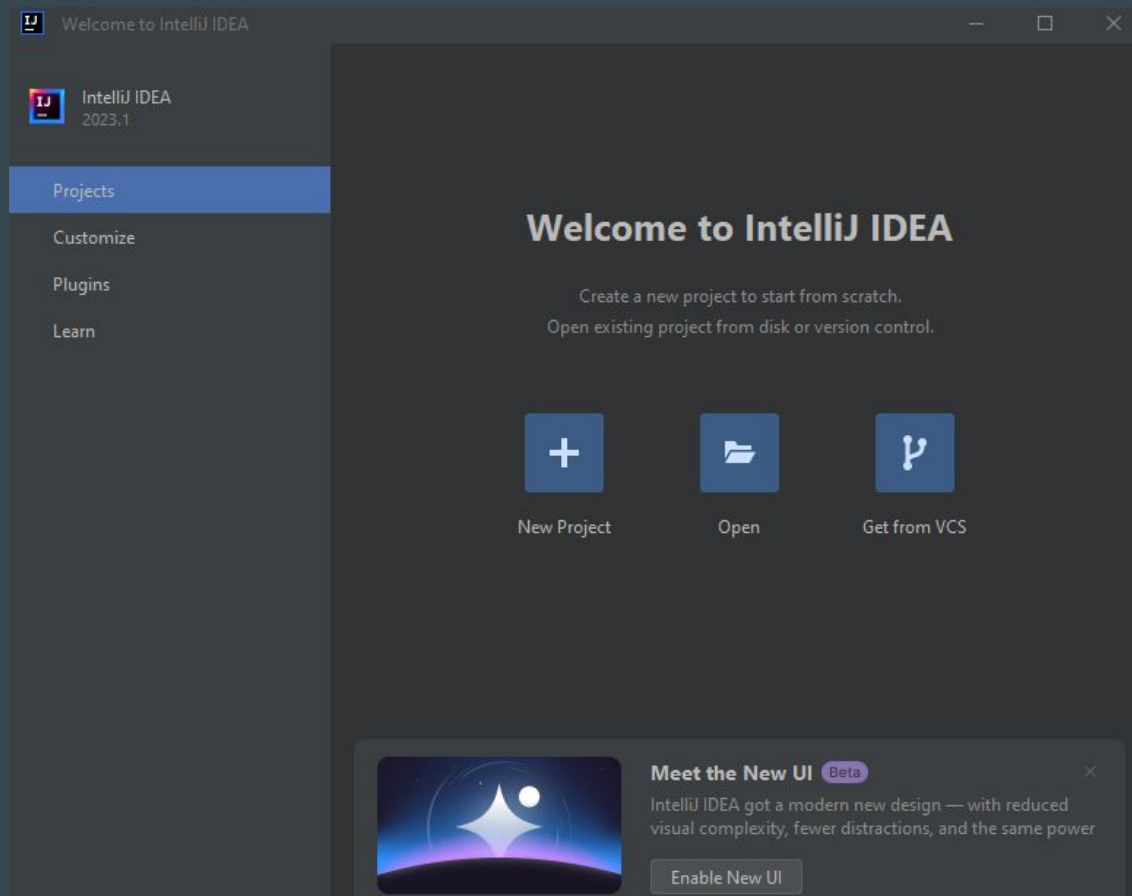
.exe ▼

Free, built on open source

Open IntelliJ IDEA



Create a new project



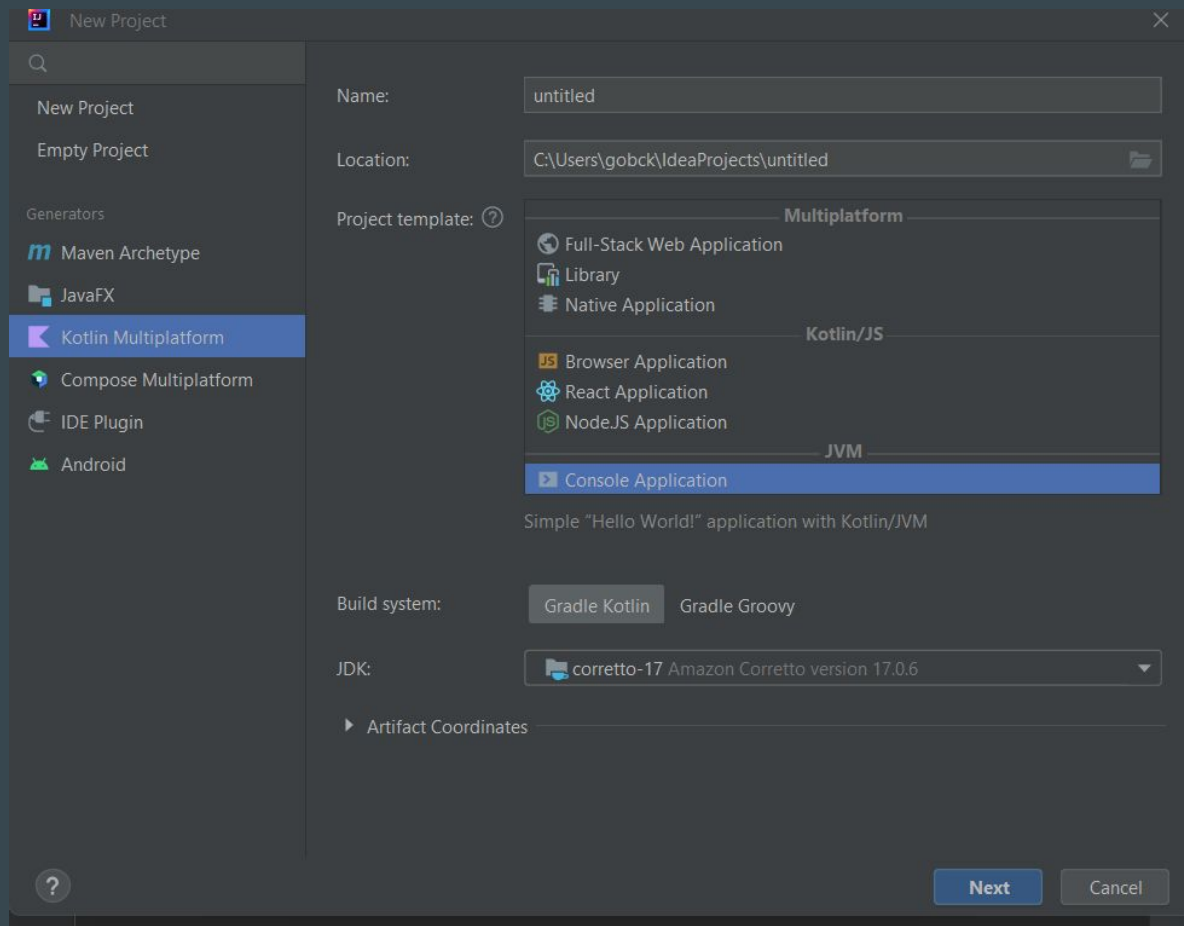
Kotlin Multiplatform

Project template:

- Console Application

JDK:

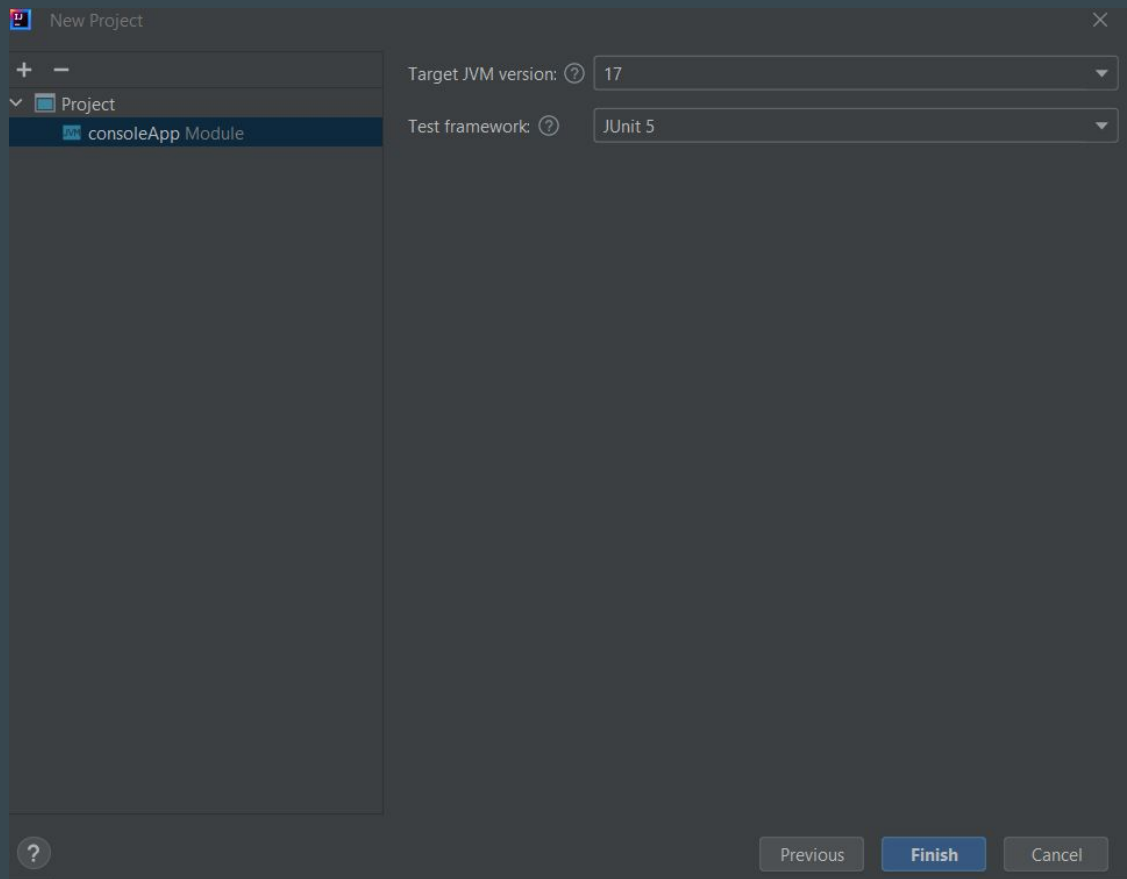
- Corretto 17



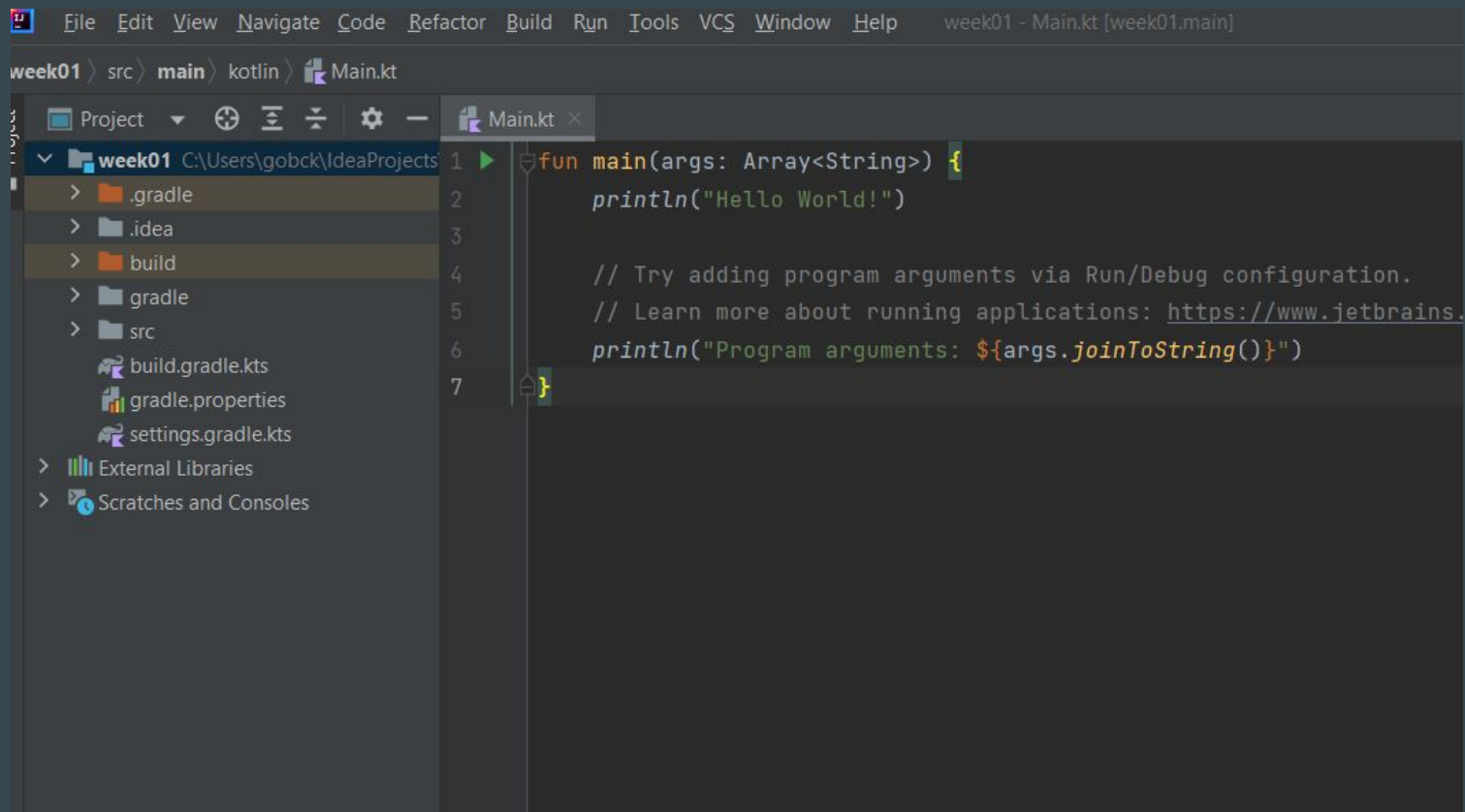
JVM version

JVM version : 17

Test framework: JUnit 5

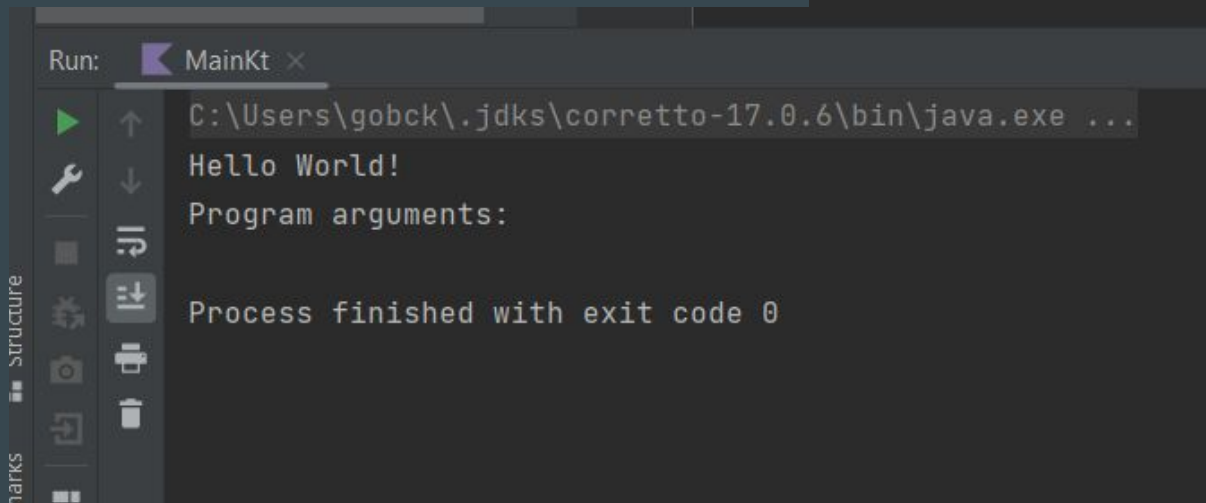
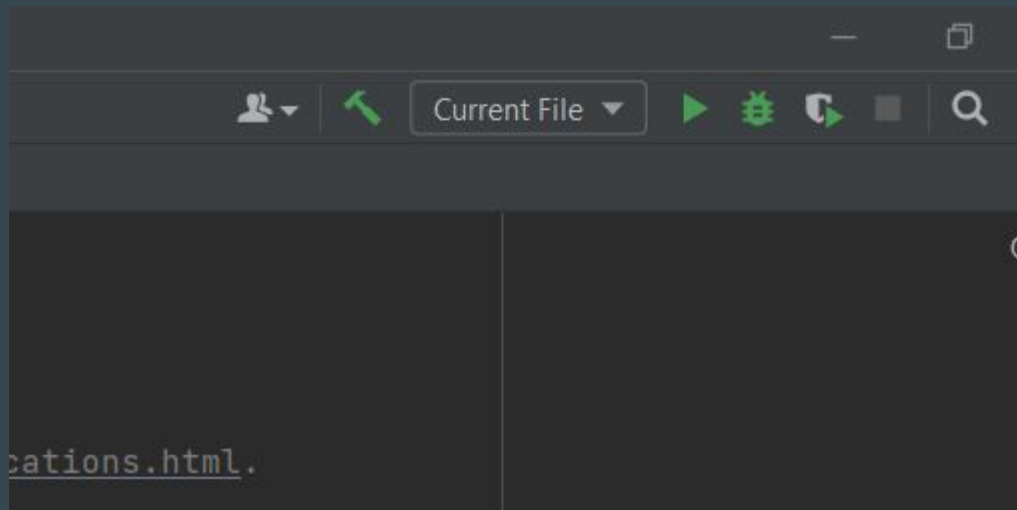


Initial Project



Run Kotlin

Click Run button or Shift + F10 on Windows



Operators

Mathematical operators

- `+` `-` `*` `/` `%`

Increment and decrement operators

- `++` `--`

Comparison operators

- `<` `<=` `>` `>=`

Assignment operators

- `=`

Equality operators

- `==` `!=`

Math operators with integers

Straightforward. Behave just as you would expect based on other programming languages

$$1 + 1 \Rightarrow 2$$

$$53 - 3 \Rightarrow 50$$

$$50 / 10 \Rightarrow 5$$

$$9 \% 3 \Rightarrow 0$$

Math operators with doubles

$1.0 / 2.0 \Rightarrow 0.5$

$2.0 * 3.5 \Rightarrow 7.0$

Numeric operator methods

Kotlin keeps numbers as primitives, but lets you call methods on numbers as if they were objects.

`2.times(3)`

`3.5.plus(4)`

`2.4.div(2)`

Kotlin Data Types

Integer Types

Type	Bits	Notes
Long	64	From -2^{63} to $2^{63}-1$
Int	32	From -2^{31} to $2^{31}-1$
Short	16	From -32768 to 32767
Byte	8	From -128 to 127

Floating-point and other numeric types

Type	Bits	Notes
Double	64	16 - 17 significant digits
Float	32	6 - 7 significant digits
Char	16	16-bit Unicode character
Boolean	8	True or false. Operations include: - lazy disjunction, && - lazy conjunction, ! - negation

Operand types

Results of operations keep the types of the operands

$6 * 50 \Rightarrow \text{Int}$

$6.0 * 50 \Rightarrow \text{Double}$

$1/2 \Rightarrow \text{Int}$

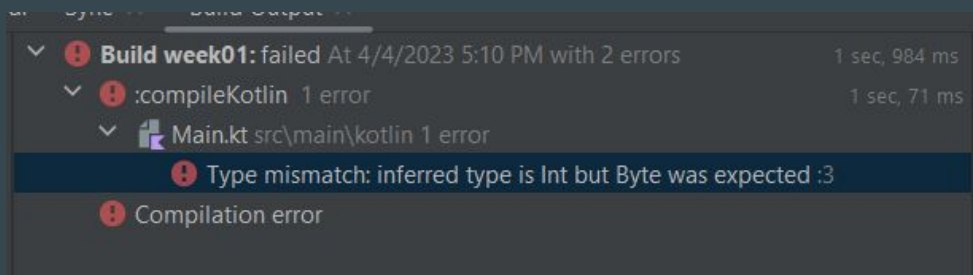
$1.0 * 2.0 \Rightarrow \text{Double}$

Type casting

Kotlin does not implicitly convert between number types

- Ex. can't assign a Short value to a Long variable
- Why? Implicit number conversion is a common source of errors.
- We need to explicitly cast

```
fun main(args: Array<String>) {  
    val i: Int = 6  
    val b: Byte = i  
    println(b)  
}
```



```
Main.kt x  
1 fun main(args: Array<String>) {  
2     val i: Int = 6  
3     val b: Byte = i.toByte()  
4     println(b)  
5 }
```

Underscores for long numbers

```
val oneMillion = 1_000_000
```

```
val idNumber = 999_99_9999L
```

```
val hexBytes = 0xFF_EC_DE_5E
```

```
val bytes = 0b11010010_01101001_10010100_10010010
```

Strings

Strings are any sequence of characters enclosed by double quotes.

```
val s1 = "Hello world!"
```

String literals can contain escape characters

```
val s2 = "Hello world!\n"
```

Or any arbitrary text delimited by a triple quote (""")

```
val text = """
```

```
var bikes = 50
```

```
"""
```


String concatenation

```
fun main(args: Array<String>) {  
  
    val numberOfDogs = 3  
    val numberOfCats = 2  
    val sentence = "I have $numberOfDogs dogs" + " and $numberOfCats cats"  
  
    println(sentence)  
}
```

MainKt x

C:\Users\gobck\.jdk\corretto-17.0.6\bin\java.exe ...

I have 3 dogs and 2 cats

Process finished with exit code 0

String templates

A template expression starts with a dollar sign (\$) and can be a simple value:

```
val i = 10
```

```
println("i = $i")
```

Or an expression inside curly braces:

```
val s = "abc"
```

```
println("$s.length is ${s.length}")
```

Kotlin Variables

Variables

Type inference

- The compiler can often infer the type
- Explicitly declare the type if needed

immutable variables (val)

Mutable variables (var)

Specifying the variable type

Colon Notation

```
var width: Int = 12
```

```
var length: Double = 2.5
```

Conditionals

Control flow

Kotlin features several ways to implement conditional logic:

If/Else statements

When statements

For loops

While loops

if/else statements

```
fun main(args: Array<String>) {  
    val numberOfCups = 30  
    val numberOfPlates = 50  
  
    if (numberOfCups > numberOfPlates) {  
        println("Too many cups!")  
    } else {  
        println("Not enough cups!")  
    }  
}
```


Ranges

Ranges let you specify a subset of a larger group.

```
Main.kt x
1  fun main(args: Array<String>) {
2      val numberOfStudents = 50
3      if (numberOfStudents in 1 ≤ .. ≤ 100) {
4          println(numberOfStudents)
5      }
6  }
```

You can also define a step size between the bounding elements of the range.

```
fun main(args: Array<String>) {
    for (i in 1 ≤ .. ≤ 8 step 2) print(i)
}
```

when statement

Nicer way to write a series of if/else statements

A bit like the "switch" statement in other languages

Conditions in a when statement can also use ranges.

```
fun main(args: Array<String>) {  
    val results: Int = 10  
    when (results) {  
        0 -> println("No results")  
        in 1 ≤ .. ≤ 39 -> println("Got results!")  
        else -> println("That's a lot of results!")  
    }  
}
```

for loops

for loop to iterate through the array and print the elements.

```
fun main(args: Array<String>) {  
    val pets = arrayOf("dog", "cat", "canary")  
    for (element in pets) {  
        print(element + " ")  
    }  
}
```

for loops: elements and indexes

you can loop through the elements and the indexes at the same time.

```
fun main(args: Array<String>) {  
    val pets = arrayOf("dog", "cat", "canary")  
    for ((index, element) in pets.withIndex()) {  
        println("Item at $index is $element\n")  
    }  
}
```

for loops: step sizes and ranges

can specify ranges of numbers or characters,
alphabetically

can define the step size

can also step backward using downTo.

```
fun main(args: Array<String>) {  
    for (i in 1 ≤ .. ≤ 5) print(i)  
  
    for (i in 5 ≥ downTo ≥ 1) print(i)  
  
    for (i in 3 ≤ .. ≤ 6 step 2) print(i)  
  
    for (i in 'd' ≤ .. ≤ 'g') print (i)  
}
```

while and do ... while loops

```
fun main(args: Array<String>) {  
    var bicycles = 0  
    while (bicycles < 50) {  
        bicycles++  
    }  
    println("$bicycles bicycles in the bicycle rack\n")  
}
```

```
fun main(args: Array<String>) {  
    var bicycles = 0  
    do {  
        bicycles--  
    } while (bicycles > 50)  
    println("$bicycles bicycles in the bicycle rack\n")  
}
```

repeat loops

let you repeat a block of code that follows inside its curly braces

The number in parentheses is the number of times it should be repeated.

```
fun main(args: Array<String>) {  
    repeat(2) {  
        print("Hello!")  
    }  
}
```

Lists and arrays

Lists

Lists are ordered collections of elements

List elements can be accessed programmatically through their indices

Elements can occur more than once in a list

Immutable list using listOf()

Declare a list using listOf() and print it out.

```
val instruments = listOf("trumpet", "piano", "violin")  
  
println(instruments)
```

Mutable list using mutableListOf()

Lists can be changed using mutableListOf()

```
val myList = mutableListOf("trumpet", "piano", "violin")  
  
myList.remove("violin")
```

Arrays

Arrays store multiple items

Array elements can be accessed programmatically through their indices

Array elements are mutable

Array size is fixed

Array characteristics

there is no mutable version of an Array.

Once you create an array, the size is fixed

You can't add or remove elements, except by copying to a new array.

Arrays with mixed or single types

An array can contain different types.

```
val mix = arrayOf("hats", 2)
```

An array can also contain just one type (integers in this case).

```
val numbers = intArrayOf(1, 2, 3)
```

Combining arrays

Use the + operator.

```
val numbers = intArrayOf(1,2,3)
```

```
val numbers2 = intArrayOf(4,5,6)
```

```
val combined = numbers2 + numbers
```

```
println(Arrays.toString(combined))
```

Null safety

Null safety

In Kotlin, variables cannot be null by default

You can explicitly assign a variable to null using the safe call operator

Allow null-pointer exceptions using the !! operator

You can test for null using the elvis (?:) operator

Variables cannot be null

In Kotlin, null variables are not allowed by default.

Programming errors involving nulls have been the source of countless bugs.

less risk of code code execution failing due to `NullPointerException`s.

Declare an `Int` and assign null to it.

```
var numberOfBooks: Int = null
```

Safe call operator

The safe call operator (?), after the type indicates that a variable can be null.

Declare an Int? as nullable

```
var numberOfBooks: Int? = null
```

Complex data types

When you have complex data types, such as a list:

- You can allow the elements of the list to be null.
- You can allow for the list to be null, but if it's not null its elements cannot be null.
- You can allow both the list or the elements to be null.

Testing for null

Check whether the `numberOfBooks` variable is not null. Then decrement that variable.

```
var numberOfBooks = 6  
  
if (numberOfBooks != null) {  
    numberOfBooks = numberOfBooks.dec()  
}
```

Now look at the Kotlin way of writing it, using the safe call operator.

```
var numberOfBooks = 6  
  
numberOfBooks = numberOfBooks?.dec()
```

The !! operator

If you're certain a variable won't be null, use !! to force the variable into a non-null type. Then you can call methods/properties on it.

```
val len = s!!.length
```

!! will throw an exception, it should only be used when it would be exceptional to hold a null value.

Elvis operator


Chain null tests with the `?:` operator.

```
numberOfBooks = numberOfBooks?.dec() ?: 0
```

Example

- "if `numberOfBooks` is not null, decrement and use it; otherwise use the value after the `?:`, which is 0."

Does Android Studio Work on Your Computer?

 Welcome to Android Studio



Android Studio
Electric Eel | 2022.1.1 Patch 2

Projects

Customize

Plugins

Learn Android Studio

Welcome to Android Studio

Create a new project to start from scratch.
Open existing project from disk or version control.



New Project



Open



Get from VCS

[More Actions](#) ▾