EAMON SCULLION

INTRODUCTION TO KOTLIN

WHAT ARE WE GOING TO COVER TODAY

- What is Kotlin?
- Who is using Kotlin?
- Why use it?
- Syntax and features (examples)
- How do I get started?

WHAT IS KOTLIN?

<- It's an island

WHAT IS KOTLIN?

- Powerful, elegant programming language for modern, multiplatform applications

- Runs on:
  - **Android**
  - **Server** (JVM, NodeJS)
  - **Browser** (JS)
  - **iOS/native**

WHO'S USING IT?

- Created by JetBrains - originally released in 2011
- Officially supported by Google (I/O 2017)

WHY USE KOTLIN?

- **Concise** - drastically reduce the amount of boilerplate code
- **Safe** - avoid entire classes of errors (e.g null pointer exception)
- **Interoperable** - leverage existing libraries for JVM, Android and the browser
- **Tool-friendly** - choose any existing Java IDE or build directly from command line

SAME AS JAVA

- ✓ Statically typed
- ✓ Use the full JVM ecosystem
- ✓ Development tools (IDEs, build tools etc.)

DIFFERENT FROM JAVA

- Null-safe
- Concise and simple (much less boilerplate)
- Proper functions
- No primitive types
- No checked exceptions
- Syntax

✳ Comparison to Java: https://kotlinlang.org/docs/reference/comparison-to-java.html

SYNTAX

VARIABLE DECLARATION

```
val a: Int = 1
```

VARIABLE DECLARATION

```
val a: Int = 1
```

Final

VARIABLE DECLARATION

```
val a: Int = 1
```

Variable name

VARIABLE DECLARATION

```
val a: Int = 1
```

Variable type (optional)

VARIABLE DECLARATION

```
val a: Int = 1
```

Value

VARIABLE DECLARATION

```
val a: Int = 1
```

End of line semicolons are optional

VARIABLE DECLARATION

```
var x = 5
```

VARIABLE DECLARATION

```
var x = 5
```

Mutable variable - value can be changed

VARIABLE DECLARATION

```
var x = 5
```

Variable type is optional - can be inferred

INSTANTIATING AN OBJECT

```
val rectangle = Rectangle(5.0, 2.0)
```

INSTANTIATING AN OBJECT

```
val rectangle = Rectangle(5.0, 2.0)
```

"new" keyword not required

FUNCTION DEFINITION

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

FUNCTION DEFINITION

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

Function type

FUNCTION DEFINITION

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

Function name

FUNCTION DEFINITION

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

Parameters

FUNCTION DEFINITION

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

Return type

FUNCTION DEFINITION

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

Function body

FUNCTION DEFINITION

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

- Can be shortened down to:

FUNCTION DEFINITION

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

- Can also be shortened down to:

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

FUNCTION DEFINITION

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

- Can also be shortened down to:

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

Return value and type is inferred

STRING TEMPLATES

```
var a = 1
// simple name in template:
val s1 = "a is $a"
```

STRING TEMPLATES

```
var a = 1
// simple name in template:
val s1 = "a is $a"
```

Use $ to insert variables into Strings

FOR LOOPS

- Java

```
for(int i=0; i<arr.length; i++){
        System.out.println(arr[i]);
}
```

FOR LOOPS

- Java

```
for(int i=0; i<arr.length; i++){
        System.out.println(arr[i]);
}
```

- Now for Kotlin…

FOR LOOPS

- Java

```
for(int i=0; i<arr.length; i++){
        System.out.println(arr[i]);
}
```

- Now for Kotlin…

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

Iterate for number of items in list

ENHANCED FOR LOOPS

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

ENHANCED FOR LOOPS

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

Iterates over each item in this list

SWITCH CASE STATEMENTS

- Java

```
switch ( user ) {
    case 18:
        System.out.println("You're 18");
        break;
  case 19:
        System.out.println("You're 19");
        break;
  case 20:
        System.out.println("You're 20");
        break;
  default:
      System.out.println("You're not 18, 19 or 20");
}
```

SWITCH CASE STATEMENTS

- Java

```
switch ( user ) {
    case 18:
        System.out.println("You're 18");
        break;
  case 19:
        System.out.println("You're 19");
        break;
  case 20:
        System.out.println("You're 20");
        break;
  default:
      System.out.println("You're not 18, 19 or 20");
}
```

- Kotlin

switch

cases

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

CLASS DEFINITIONS

- Java

```java
public class Person {
    private String name;
    private int age = 0;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Person person = (Person) o;

        if (name != null ? !name.equals(person.name) : person.name != null) return false;
        if (age != 0 ? age != person.age : person.age != 0) return false;
    }

    @Override
    public int hashCode() {
        int result = name != null ? name.hashCode() : 0;
        result = 31 * result + age;
        return result;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age='" + age + '\'' +
                '}';
    }
}
```

CLASS DEFINITIONS

- Kotlin

CLASS DEFINITIONS

- Drum roll...

CLASS DEFINITIONS

- Kotlin

```
data class Person(val name: String, val age: Int)
```

CLASS DEFINITIONS

- Kotlin

```
data class Person(val name: String, val age: Int)
```

Automatically sets up getters, setters, equals, hashcode, toString and copy methods

CLASS DEFINITIONS

- Kotlin

```
data class Person(val name: String, val age: Int)
```

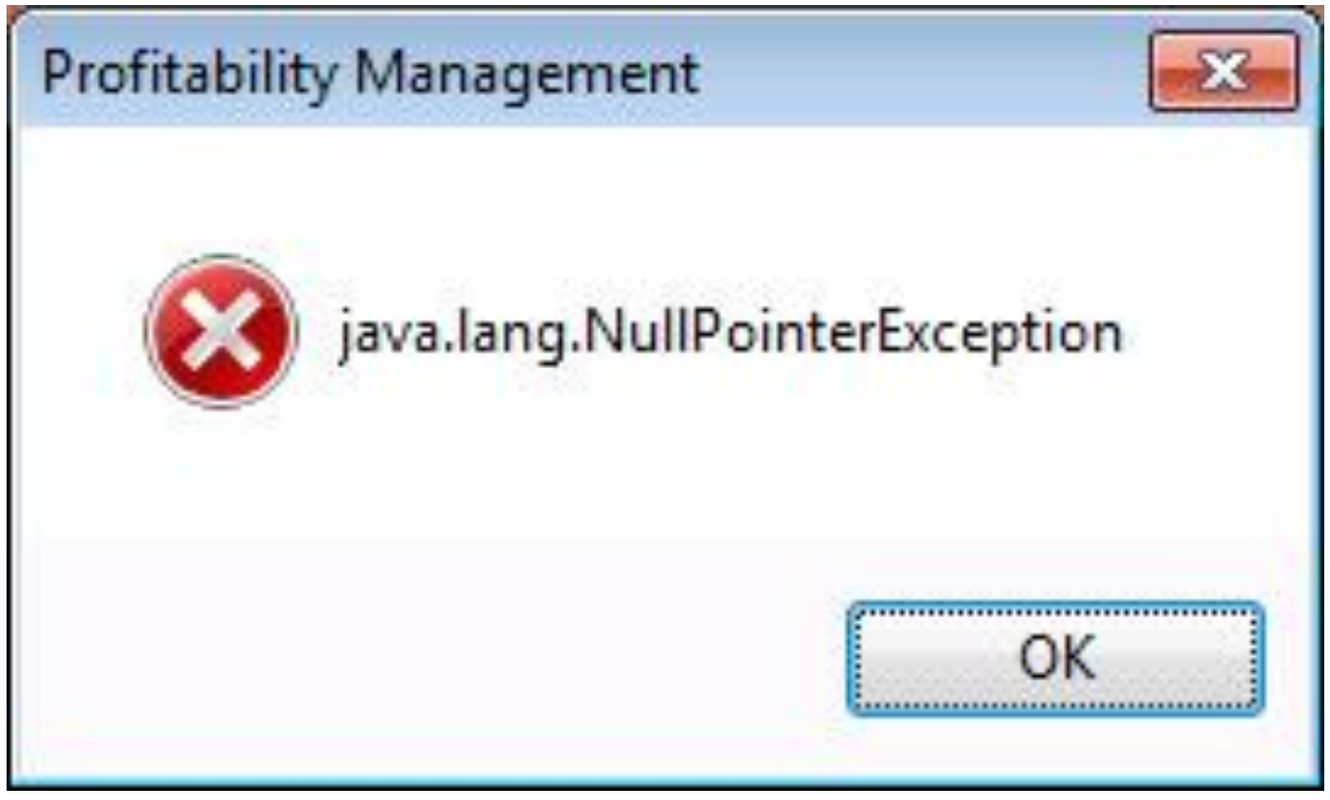
Constructor

NULL SAFETY

- Java

NULL SAFETY

- Java



Tony Hoare (Inventor of the null reference) calls it his "**Billion dollar mistake**" due to all of the "innumerable errors, vulnerabilities, and system crashes"

NULL SAFETY

- Kotlin
  - Numerous ways to handle NPE
  - Will not compile if you haven’t handled all possible NPE’s

```
var a: String = "abc"
a = null // compilation error
```

NULL SAFETY

- Declare a variable as nullable

```
var b: String? = "abc"
b = null // ok
print(b)
```

Declare as nullable

NULL SAFETY

- Safe calls

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

NULL SAFETY

- Safe calls

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

- Access this safely by using `?`:

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

- This returns b.length if b is not null, otherwise it will return null (instead of NPE)

ELVIS OPERATOR

- Allows you to specify what to return if the result is null

```
val l = b?.length ?: -1
```

- This returns length if b is not null, otherwise it will return -1

ELVIS OPERATOR

- Allows you to specify what to return if the result is null

```
val l = b?.length ?: -1
```

- This returns length if b is not null, otherwise it will return -1

```
val name = node.getName() ?: throw IllegalArgumentException("name expected")
```

- Useful for throwing exceptions

WHYS IT CALLED ELVIS OPERATOR?

WHYS IT CALLED ELVIS OPERATOR?

NULL SAFETY

- If you **really** still want null pointer exceptions, you can use the non-null assertion operator (!!).

```
val l = b!!.length
```

- This will throw a null pointer exception if b is null
- ***Use at your own risk!!***

OTHER FEATURES

- **Extension functions** - extend a class with new functionality with affecting the underlying code e.g adding functionality to an external API:

```
fun MutableList<Int>.swap(index1: Int, index2: Int) {
    val tmp = this[index1] // 'this' corresponds to the list
    this[index1] = this[index2]
    this[index2] = tmp
}
```

OTHER FEATURES

- **Extension functions** - extend a class with new functionality with affecting the underlying code e.g adding functionality to an external API:

```
fun MutableList<Int>.swap(index1: Int, index2: Int) {
    val tmp = this[index1] // 'this' corresponds to the list
    this[index1] = this[index2]
    this[index2] = tmp
}
```

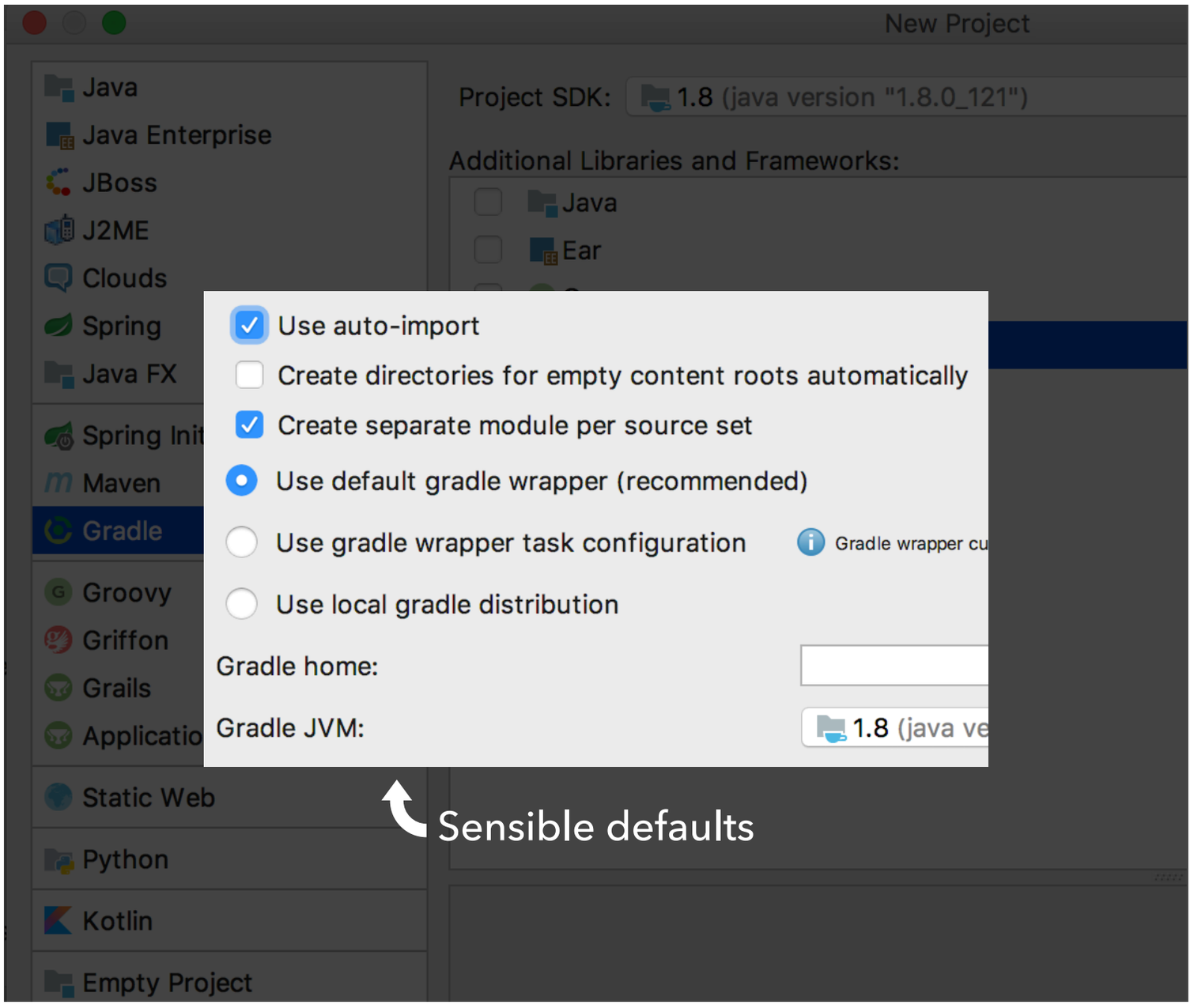
- We can now call this any instance (within the same scope):

```
val l = mutableListOf(1, 2, 3)
l.swap(0, 2) // 'this' inside 'swap()' will hold the value of 'l'
```

HOW TO GET STARTED?

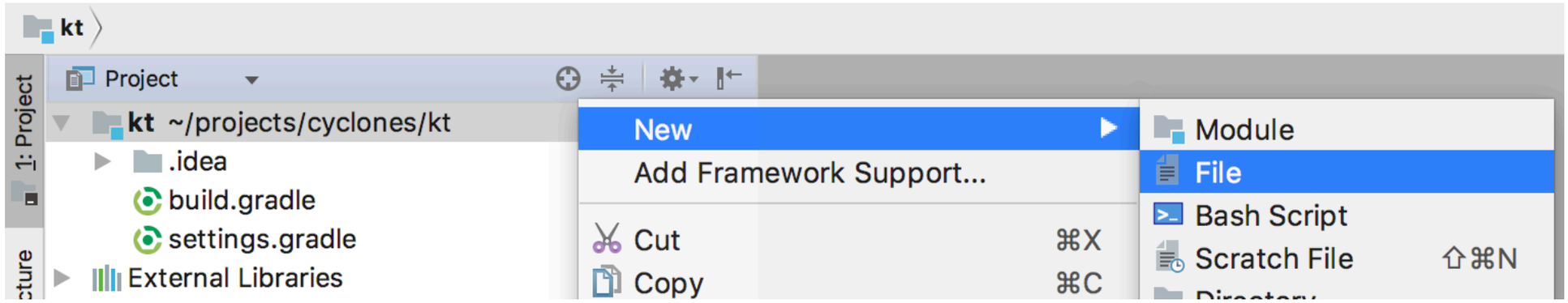
GETTING STARTED

1. Open IntelliJ IDEA
2. File > New > Project
3. Gradle > Kotlin (Java)
4. Next, etc.

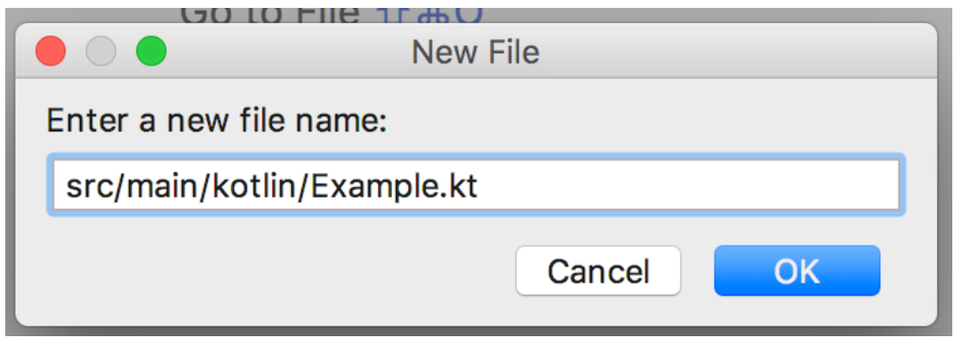


HELLO WORLD

1. Add a Kotlin file



2. Add a Kotlin file



3. Hello world

```
1
2  fun main(args: Array<String>) {
3      println("Hello world")
4  }
```

✳ Basic syntax: https://kotlinlang.org/docs/reference/basic-syntax.html

FURTHER READING

- **Kotlin official**
  - https://kotlinlang.org/
  - https://kotlinlang.org/docs/reference/
- **Kotlin koans**
  - https://kotlinlang.org/docs/tutorials/koans.html
- **Getting started with Android and Kotlin**
  - https://kotlinlang.org/docs/tutorials/kotlin-android.html

USEFUL LIBRARIES

- **ktlint** - linter with built-in formatter
  - https://ktlint.github.io/
- **Gradle Kotlin DSL** - kotlin support for writing gradle files
  - https://github.com/gradle/kotlin-dsl
- **Anko - An**droid **Ko**tlin – simplifies and speeds up Android development using Kotlin
  - https://github.com/Kotlin/anko

QUESTIONS?