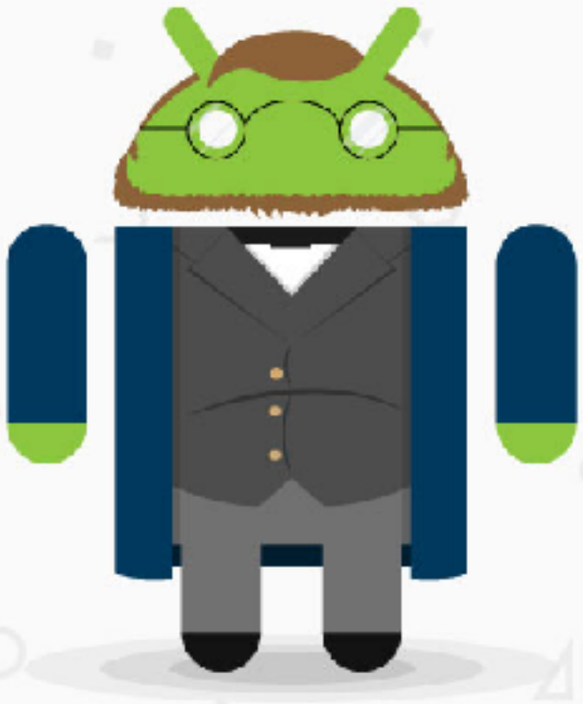
SoftUni

Sofia, 2017

GETTING STARTED

WITH KOTLIN



Dimitar Ivanov

How to install Kotlin

- Command Line
- Compile file with kotlinc
  - kotlinc <kotlin file path>
  - kotlinc <path to kotlin file> -include-runtime -d hello.jar
  - java - jar <path to jar file>

What is Kotlin ?

- JVM Language
  - Java similar / Easier than Scala
- Supports immutability
- Abhors nulls
- OOP
- Functional language
- Less ceremony than Java

Hands dirty with Kotlin

- Hello world app in IntelliJ
- Setup run config
- Create simple class (mutable / unmutable)
- Lets discuss var/val
- Class constructor with name parameters
- Summary
  - Why should we use Kotlin ?

Kotlin without any class definitions

- Java demands class files!
- Show the MainKT class file
- Decompile .class to java readable format with fernflower
    - Java -jar fernflower.jar <class here> decompiled
    - Open the file and discuss it
- Try to run the class file
    - java -cp <class>
    - compile it with kotlinc compiler

Kotlin immutability

- Create simple Question class
    - Show the QuestionKt.class (do not create another file for it)

Kotlin String templates

- **$ - for single variable**
- **${scope} - for the whole object scope**
- **String interpolation**

Kotlin if statements

- == operator works as expected in kotlin rather than java

- Show if as expression stored in var/val

Kotlin Null Variables

- What is the ? operator

- What is the !! operator -> DANGER

When statements in kotlin

- **Kotlin does not have switch concept**
  - **-when is the replacement**
- **when as a expression is not going to work the same way it does for if**
- **Show a demo of when**

Try/catch in Kotlin

- Same as in Java

- try {} catch {} finally {}

Loops in Kotlin

- **Down to**
- **Until**
- **listOf(1,2,3,4,5)**
- **Map TreeMap<String,Int>()**
  - **for(name,age in ages)**

Task

- **Check if numbers in array/list is odd or even**
  - **Use if as expression**

Kotlin functions

Functions Demo

```
fun add(x:Int ,y:Int):Int = x + y
```

```
fun add(x: Int, y: Int): Int {
    return x + y
}
```

Kotlin working with Java

Default Parameters

Named Parameters

Kotlin operators

Interfaces

- They are public by default

- Several demos around interfaces in Kotlin

Interface Example

```
interface Time {
    fun setTime(hours: Int, mins: Int = 0, secs: Int = 0)
    fun setTime(time: KevinTime) = setTime(time.hours)
}
```

Interface Example

```
interface A {
    fun doSomething()
    {

    }
}

interface B {
    fun doSomething()
    {

    }
}

class Foo : A, B {
    override fun doSomething() {
        super<A>.doSomething()
    }
}
```

Classes

- **public by default**
- **abstract classes**
- **Modifiers**
- **Sealed classes (enums on steroids)**
- **Constructors**
- **Data classes**

Classes Example

```
class Student : Person {

    fun getClasses() {}
    //...
}
```

Class is final by default
Methods are final by default

Use open to show function can be overridden

Use open to show class can be derived from

Classes Example

```
abstract class Person {
    abstract fun getName()
    open fun workHard() {}
    fun goOnHoliday() {}
}
```

Class is abstract

getName must be implemented

workHard may be overridden

goOnHoliday cannot
be overridden

Sealed Classes

- **Used to restrict class Hierarchies**

- **‘Enums on steroids’**

Sealed Class Example

```
sealed class Event {
    class SendEvent(id: Int, to: String) : Event()
    class ReceiveEvent(id: Int, from: String) : Event()
}
```

Using sealed classes

```
fun handleEvent(e:Event) =
    when(e) {
        is SendEvent -> print(e.to)
        is ReceiveEvent -> print(e.from)

    }
```

Class constructor

```
open class Person(val name: String)
```

```
val dimitar = Person(“Dimitar”)
```

Class constructor

```
open class Person(name: String) {
  val name: String
  init {
    this.name = name
  }
}

open class Person(_name: String) {
  val name = _name
}
```

Secondary Constructor Usage

```
open class Person(name: String) {
  constructor(name: String, age: Int) : this(name)
}
```

Primary Constructor Usage

```
open class Person(name: String, age = 0) {
}
```

Calling Superclass Constructor

```
class Student(name : String) : Person(name)
```

```
class Student: Person {
  constructor(name: String) : super(name)
}
```

Private Constructor

We will see that later in the class …

Classes

Demo

Data Classes

Provide a convenient way to override equals, hashCode and toString

Typically immutable classes

Kotlin also generates ‘copy’ method

Using Data Classes

```
data class Meeting(val name:String, val location:String)
val aMeeting = Meeting(“A Meeting”, “London”)
val anotherMeeting = aMeeting.copy(location = “New York”)
```

Using Data Classes

data class Meeting(val name:String, val location:String)
val aMeeting = Meeting(“A Meeting”, “London”)
val anotherMeeting = aMeeting.copy(location = “New York”)

show them toString of class and data class!

Nullability

- **Java throws NullPointerException**
- **Supports ‘nullable’ types**
- **Only explicit can be null**

Demo

```
boolean skipClass(Class class) {

if(class.skipped) return true

return false;

}
```

let...

- **Useful when passing nullable variables**

Lateinit

- **I dont want my variables to be null…**

Companion object

- **Kotlin does not have static methods**
- **Can use singletons**
- **Use ‘object’ keyword**
- **Companion only to get statics**

Objects

- **Can’t have constructors**

- **Can have the same behavior as class and other members**

Simple Demo

```
object Projects
{
    var allProjects = arrayListOf<Projects>()
}

Projects.allProjects.add(Project(...))
```

Extending objects

- - companion object
- using companion object

Higher level functions

- **Make factorial demo**

Using with and apply

- Look like language keyword
- Uses lambdas

Extensions methods

Tasks One

- **Create program that create array of all elements from 1…n and using .filter print only the elements that are even**

- **Use higher order of function and print the elements using the strategy pattern**

Task Two

- **Create class student that can enroll for different courses**
- **Use companion object to store the courses**
- **Use filter to find all courses starting with any letter you prefer**

Task three

- **Given certain range of number from 10..50 for example print all prime numbers.**