Mobile Application Development

Produced by

David Drohan (ddrohan@wit.ie)
Dr. Siobhan Drohan (sdrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology
http://www.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Introducing Kotlin Syntax - Part 2

Agenda from Part 1

- ❑ Basic Types
- ❑ Local Variables (val & var)
- ❑ Functions
- ❑ Control Flow (if, when, for, while)
- ❑ Strings & String Templates
- ❑ Ranges (and the *in* operator)
- ❑ Type Checks & Casts
- ❑ Null Safety
- ❑ Comments

Agenda for Part 2

- ❑ Writing Classes (properties and fields)
- ❑ Data Classes (just for data)
- ❑ Collections: Arrays and Collections
- ❑ Collections: *in* operator and lambdas
- ❑ Arguments (default and named)

Agenda for Part 2

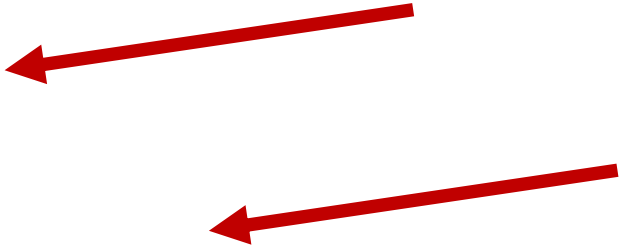
- ❑ Writing Classes (properties and fields)
- ❑ Data Classes (just for data)
- ❑ Collections: Arrays and Collections
- ❑ Collections: *in* operator and lambdas
- ❑ Arguments (default and named)

Writing Classes

Properties & Fields

Writing Classes – properties

- In Kotlin, classes don't have explicit fields; they have properties.



var properties **are** mutable.

val properties **cannot** be changed.

Writing Classes – constructors

- A class in Kotlin can have a primary constructor and one or more secondary constructors.
- The primary constructor is **part of the class header** and it goes after the class name:

```
class Person constructor(firstName: String) {
}
```

```
class Person(firstName: String, lastName: String) {
}
```

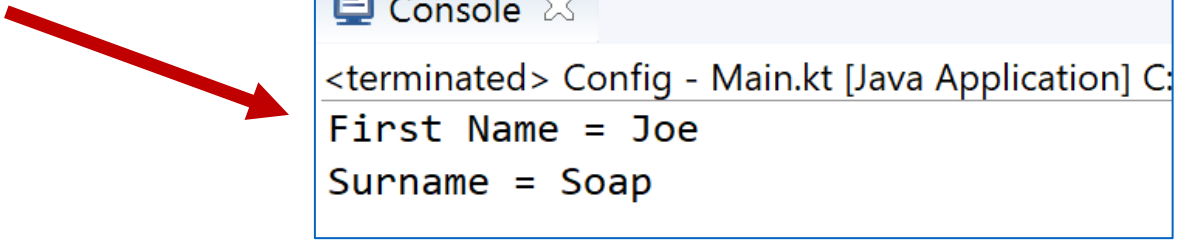
```
class Person(val firstName: String, val lastName: String) {
}
```

Writing Classes – primary constructors

```
class Person(val firstName: String, val lastName: String) {

}
```

```
fun main(args: Array<String>) {
    val person = Person("Joe", "Soap")
    println("First Name = ${person.firstName}")
    println("Surname = ${person.lastName}")
}
```



Writing Classes – primary constructors

- The **primary constructor** cannot contain any code - initialisation code is placed in the init block.
- The use of _ prefixing constructor variables is standard.

```
class Person( _firstName: String = "UNKNOWN FIRSTNAME",
              _lastName: String = "UNKNOWN LASTNAME") {

    val firstName = _firstName
    val lastName  = _lastName

    // initializer block
    init {
        println("First Name = $firstName")
        println("Last Name = $lastName\n")
       }
}
```

Writing Classes – primary constructors

```
class Person( _firstName: String = "UNKNOWN FIRSTNAME",
              _lastName: String = "UNKNOWN LASTNAME") {

    val firstName = _firstName
    val lastName  = _lastName

    // initializer block
    init {
        println("First Name = $firstName")
        println("Last Name = $lastName\n")
         }
}
```

```
fun main(args: Array<String>) {

    println("person1 is instantiated")
    val person1 = Person("Joe", "Soap")

    println("person2 is instantiated")
    val person2 = Person("Jack")

    println("person3 is instantiated")
    val person3 = Person()
}
```

Writing Classes – primary constructors

```
class Person( _firstName: String = "UNKNOWN FIRSTNAME",
              _lastName: String = "UNKNOWN LASTNAME") {

    val firstName = _firstName
    val lastName  = _lastName

    // initializer block
    init {
        println("First Name = $firstName")
        println("Last Name = $lastName\n")
         }
}
```

```
fun main(args: Array<String>) {

    println("person1 is instantiated")
    val person1 = Person("Joe", "Soap")

    println("person2 is instantiated")
    val person2 = Person("Jack")

    println("person3 is instantiated")
    val person3 = Person()
}
```

Console

<terminated> Config - Main.kt [Java Application]

```
person1 is instantiated
First Name = Joe
Last Name = Soap

person2 is instantiated
First Name = Jack
Last Name = UNKNOWN LASTNAME

person3 is instantiated
First Name = UNKNOWN FIRSTNAME
Last Name = UNKNOWN LASTNAME
```

Note: varied parameters allowed in primary constructor as values are defaulted (i.e. optional parameters)

Writing Classes – secondary constructors

- The secondary constructor is prefixed with the keyword constructor. They are not very common in Kotlin.

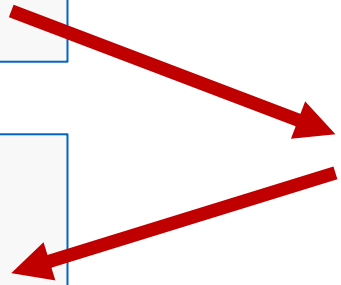
```
class Person {
    constructor(parent: Person) {
                parent.children.add(this)
                }
}
```

- http://kotlinlang.org/docs/reference/classes.html

Writing Classes – getters and setters

- In Kotlin, getters (val and var) and setters (var) are optional and are auto-generated if you do not create them in your program.

```
class Person {
    var name: String = "defaultValue"
}
```



Is equivalent to

```
class Person {
    var name: String = "defaultValue"
    // getter
    get() = field
    // setter
    set(value) {
        field = value
    }
}
```

Writing Classes – getters and setters

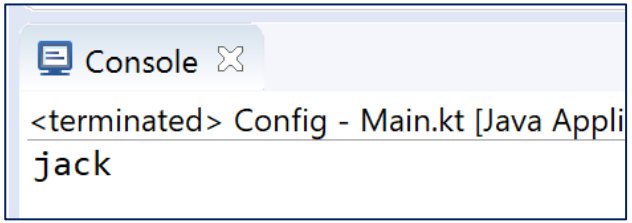
- When you instantiate object of the Person class and initialize the name property, it is passed to the setters parameter value and sets field to value.

```
class Person {
    var name: String = "defaultValue"
    // getter
    get() = field
    // setter
    set(value) {
        field = value
    }
}
```

```
fun main(args: Array<String>) {

    val person = Person()
    person.name = "jack"
    print(person.name)

}
```



Writing Classes – getters and setters

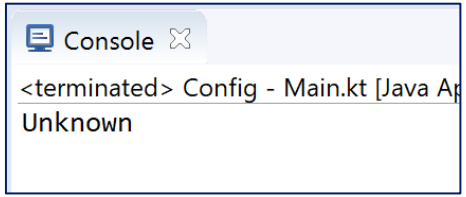
- When you want to add validation to your setter…

```
fun main(args: Array<String>) {

    val person = Person()
    person.name = ""
    print(person.name)

}
```

```
class Person {
    var name: String = "defaultValue"
    get() = field
    set(value) {
        field = if (value.equals(""))
                    "Unknown"
                else
                    value
    }
}
```



Writing Classes – getters and setters

- If we add another property…

```
class Person {
    var name: String = "defaultValue"
        get() = field
        set(value) {
            field = if (value.equals(""))
                        "Unknown"
                    else
                        value
    }

    var id: Int = 10001
        get() = field
        set(value) {
            field = value
        }
}
```

Writing Classes – field & value

- You might have noticed two strange identifiers in all the getter and setter methods - field and value.
- We use **value** as the name of the setter parameter. This is the default convention in Kotlin but you're free to use any other name if you want.
- The **value** parameter contains the value that a property is assigned to. For example, when you write `person.name = "jack"`, the **value** parameter contains the assigned value "jack".
- (Using) **field** helps you refer to the property inside the getter and setter methods. This is required because if you use the property directly inside the getter or setter then you'll run into a recursive call which will generate a *StackOverflowError*.

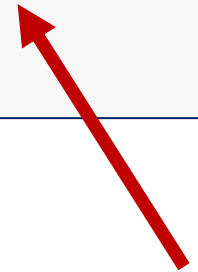
Data Classes

Just for data

Data Classes

- We frequently create classes whose main purpose is to hold data. In such a class some standard functionality and utility functions are often mechanically derivable from the data. In Kotlin, this is called a ***data** class* and is marked as data:
- The compiler automatically generates methods such as **equals()**, **hashCode()**, **toString()**, **copy()** from the primary constructor.

```
data class Person(var firstName: String,
                  var lastName: String) {
}
```



Data Classes - Requirements

- The primary constructor must have **at least** one parameter
- The parameters of the primary constructor must be marked as either **var** or **val**
- The class cannot be open, abstract, inner or sealed
- The class may extend other classes or implement interfaces

```
data class Person(var firstName: String,
                  var lastName: String) {
}
```

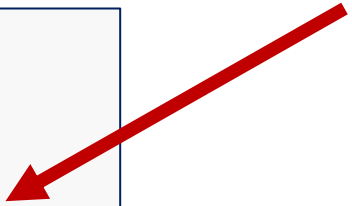
Data Classes – copy and toString Example

```
data class Person(var firstName: String,
                  var lastName: String) {
}
```

```
fun main(args: Array<String>) {
    val person1 = Person("John", "Murphy")

    // using copy function to create an object
    val person2 = person1.copy(firstName="Martin")

    println(person1)
    println(person2.toString())
}
```



Console

```
<terminated> Config - Main.kt [Java Application] C:\Program Files\J
Person(firstName=John, lastName=Murphy)
Person(firstName=Martin, lastName=Murphy)
```

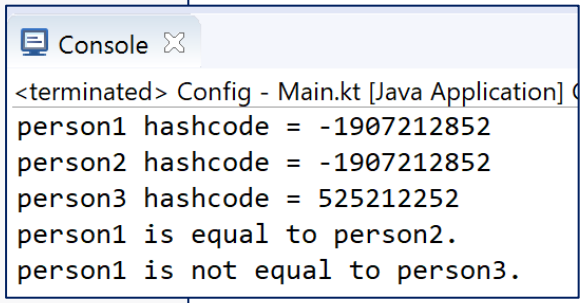
Data Classes – copy, equals and hashCode Example

```
fun main(args: Array<String>) {
    val person1 = Person("John", "Murphy")
    val person2 = person1.copy()
    val person3 = person1.copy(firstName = "Martin")

    println("person1 hashcode = ${person1.hashCode()}")
    println("person2 hashcode = ${person2.hashCode()}")
    println("person3 hashcode = ${person3.hashCode()}")

    if (person1.equals(person2))
        println("person1 is equal to person2.")
    else
        println("person1 is not equal to person2.")

    if (person1.equals(person3))
        println("person1 is equal to person3.")
    else
        println("person1 is not equal to person3.")
}
```



Some additional sources for exploration:

Inheritance	https://www.programiz.com/kotlin-programming/inheritance
Interfaces	https://www.programiz.com/kotlin-programming/interfaces
Collections	https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/index.html
Try examples online	https://try.kotlinlang.org/#/Examples/Hello,%20world!/Simplest%20version/Simplest%20version.kt
Encapsulation & Polymorphism	https://medium.com/@napperley/kotlin-tutorial-12-encapsulation-and-polymorphism-6e5a150f25e1
Spek (testing)	https://objectpartners.com/2016/02/23/an-introduction-to-kotlin/ https://github.com/mike-plummer/KotlinCalendar

References

Sources:

http://kotlinlang.org/docs/reference/basic-syntax.html
http://petersommerhoff.com/dev/kotlin/kotlin-for-java-devs/
https://www.programiz.com/kotlin-programming
https://www.baeldung.com/kotlin-lambda-expressions
https://www.programiz.com/kotlin-programming/lambdas
https://medium.com/@napperley/kotlin-tutorial-5-basic-collections-3f114996692b


Thanks.