5COSC023W - MOBILE APPLICATION DEVELOPMENT

Lecture 1: Introduction to Android Programming and Kotlin

Dr Dimitris C. Dracopoulos

Introduction to the Module

- Syllabus
- Lectures
- Tutorials (Practicals)
- Software
- Assessment
- Schedule
- What is expected from you?
  - Lecture Attendance
  - Tutorial Attendance (actual not just swiping of card!)
  - Completion of ALL Tutorial Exercises within the week (if not possible within the tutorial session then on your own time).
  - Code of Conduct

Code of Conduct

- Do not cheat on assignments (this is *INDIVIDUAL* work and **NOT** the product of collaboration!):
  - Discuss only general approaches not specific details of implementation
  - Do not take written notes on other's work and do not exchange code
- Cheating is reported to university and then it is out of the module lecturers hands (independent committee decision without the participation of the module tutors)
- Possible consequences:
  - A mark of 0 for assignment
  - A mark of 0 for the course
  - A permanent note on student record
  - Suspension/Expulsion from university

Code of Conduct (cont'ed)

- Any code found in the web or textbook and used in your work should be properly referenced in comments within your code.

Academic Integrity

- The University of Westminster is committed to the highest standards of academic integrity and honesty. Students are expected to be familiar with these standards regarding academic honesty and to uphold the policies of the University in this respect. Students are particularly urged to familiarise themselves with the provisions of the Academic Regulations and in this case with Academic Misconduct Regulations (https://www.westminster.ac.uk/sites/default/public-files/general-documents/Section-10-Academic-Misconduct-v2.pdf) and avoid any behaviour which could potentially result in suspicions of cheating, plagiarism, misrepresentation of facts and/or participation in an offence. Academic dishonesty is a serious offence and can result in suspension or expulsion from the University.

Why Kotlin?

- Recommended by Google for Android programming
- Java for Android is NOT obsolete and still supported by Google!
- Some Kotlin features make it easier to use (at least for some programmers)
- Java code can be called by Kotlin and Kotlin code can be called by Java
- Kotlin is a hybrid programming language: it supports both functional programming and object-oriented programming

How to Practice Kotlin outside Android Studio

For the purposes of practice your Kotlin knowledge and not for producing a mobile application, you have the following options:

1. Use a Web browser with the following url and press the button to run the code:

   `https://play.kotlinlang.org/`

2. Install IntelliJ from:

   `https://www.jetbrains.com/idea/`

3. Use the command line utilities: `kotlinc`, `kotlin` by installing the *kotlin* package in your operating system.

Kotlin Basics

Variables

Identifiers can be used by using `var` and `var`:

```
val identifier1: Type = initialisation
var identifier2: Type = initialisation
```

Examples:

```
val hoursPerDay: Int = 24
var hoursWorkedThisMonth: Int = 110
```

- An identifier defined with `val` is constant and cannot be changed:

```
hoursPerDay = 35  // invalid - compiler will give an error
hoursWorkedThisMonth = 222  // OK - valid
```

- The compiler will do *type inference* if a type is omitted:

```
var colour = "red"  // type String is inferred
```

- Note that Kotlin does not require a semicolon at the end of an expression or statement!

Data Types

- Byte
- Short
- Int
- Long
- Float
- Double
- String
- Boolean

Functions

Syntax:

```
fun functionName(arg1: Type1, arg2: Type2, ...): ReturnType {
    // ... body with lines of code
    return result
}
```

Example:

```
fun sum(a: Int, b: Int, c: Int): Int {
    var res:Int = a + b + c

    return res
}
```

Usage:

```
var result = sum(5, 8, 9)
println(result)
```

Functions (cont'ed)

- If a function consists of a single expression the following alternative syntax can be used:

```
fun functionName(arg1: Type1, arg2: Type2, ...): ReturnType =
    result
```

- In this case the return type can be omitted and it will be automatically inferred.

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

- If a function does not return anything, this can be declared with the `Unit` type (equivalent to `void` in Java) or simply omitting the return type:

```
fun abc(a: Int): Unit =
    print(a)
```

Conditionals

`if` conditionals are expressions producing a result, which can then be assigned to variables.

```
fun check(x: Int): String {
    val message =
        if (x < 0)
            "x is negative"
        else if (x >= 0 && x <=5)
            "x is between 0 and 5"
        else
            "x is greater than 5"

    return message
}
fun main() {
    print(check(4))
}
```

The output is:

```
x is between 0 and 5
```

String Templates

A value can be inserted inside a string by:

- Using $ before an identifier.
- Placing an expression inside ${}.

```
fun main() {
    var x = 5

    println("The value of x is $x")
    println("""${if (x >=0) "positive"  else  "negative" }""")
}
```

The output is:

```
The value of x is 5
positive
```

- String literals can be enclosed within double or triple quotes. Triple quotes literals can span across multiple lines.

Loops

Use either of:

- `for`
- `while`
- `repeat`

Loops (cont'ed)

Example:

```
fun main() {
    for (x in "abcde")
        println(x)

    // using ranges
    for (i in 1..10)
        println(i)

    // using ranges - upper bound value excluded
    for (i in 1 until 10)
        println(i)

    var n = 1
    while (n <= 10) {
        print(" " + (n+1))
        n += 1
    }

    println()
    repeat (5) {
        print("once more ")
    }
}
```

Loops (cont'ed)

The output is:

```
a
b
c
d
e
1
2
3
4
5
6
7
8
9
10
 2 3 4 5 6 7 8 9 10 11
once more once more once more once more once more
```

Lists

A list is a collection, a container which holds other objects.

- Use the `listOf` and `mutableListOf` functions to create a read-only and a mutable list respectively:

```
fun main() {
  var a = listOf(5, 10, -1, 99, 133, 0)
  println(a[3])  // displays 99
  //a[2] = 22  // cannot modify a read-only list

  var b = mutableListOf(4, 77, 1, 88, -2)
  b[2] = 5  // OK
  b.add(5)

  println(b)
  print("size of b is " + b.size)
}
```

The output is:

```
99
[4, 77, 5, 88, -2, 5]
size of b is 6
```

The in Keyword

- in can be used in both loops and to check whether "something" is a member of "something else".

```
fun main() {
    var a = listOf(5, 10, -1, 99, 133, 0)
    println(10 in a)  // it is there
    println(11 in a)  // not in there!

    for (c in "penguin")
        print(c + "-")
}
```

The output is:

```
true
false
p-e-n-g-u-i-n-
```

Parameterised Types and Equality

Type parameters describe the type of containers:

```
var l1: List<Int> = listOf(100, 5, 99)
val l2 = listOf("abc", 8, 10)
val l3: List<Int> = listOf("abc", 8, 10)  // Invalid-> compiler error

val l4: List<Int> = listOf(5,6,7)
// comparing objects
println(l1 == l4) // comparing contents -> true
println(l1 === l4) // comparing references (memory addresses) -> false
```

Default Values for Function Arguments

Function arguments can have an optional name and an optional default value.

- The order of arguments can be changed if their names is used.

```
fun colour(red: Int = 0, green: Int = 0, blue:Int = 0) {

}

fun main() {
    // default value for green is used, i.e. 0
    colour(blue = 255, red = 125)
}
```