Programming II
Kotlin functions & Lambdas

CO5225 – Andrew Muncey

Contents

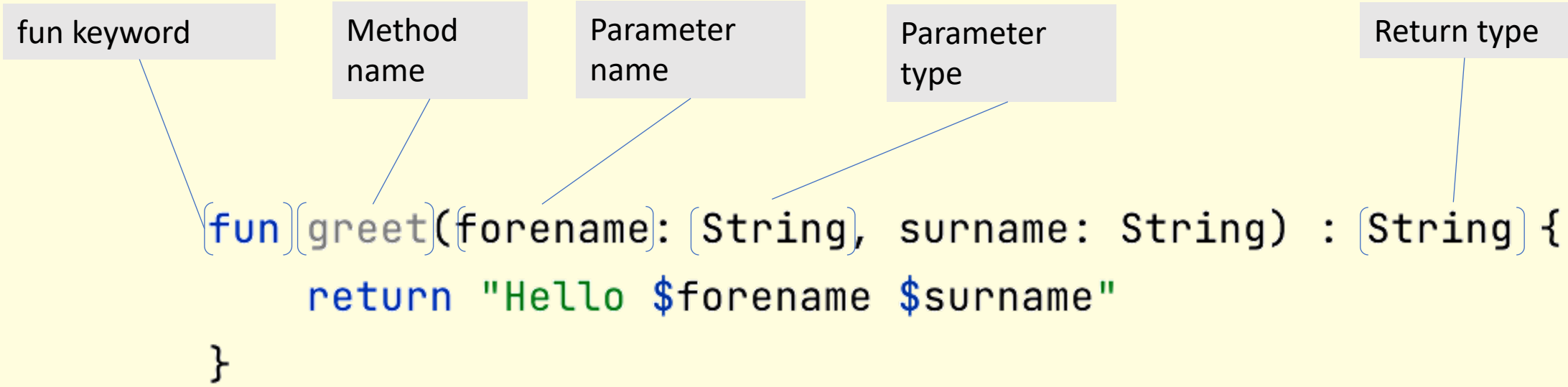
- Methods / Functions
- Default Values & Named Parameter passing
- Functions as variables / Lambda expressions

Methods / Functions

- In Java, we commonly referred to methods
- A method is just a function that belongs to a class
- All methods are functions

- Kotlin allows functions outside of a class
- We will more commonly refer to functions in this module

Function and function types



- The type of this function is (String, String) -> String
- E.g. it takes two parameters, both of which are strings and returns a string
- Called as follows: `greet("John", "Smith")`

Default Values and Named Parameter passing

- Kotlin also allows us to provide default parameters, this is helpful when there are default values we may want to use, and reduces the number of overloads required. e.g.,

```
fun describeMark(mark: Int, pass: Int = 50,  merit :Int = 60,  distinction :Int = 70) : String{
    if (mark >= distinction) {return "Distinction"}
    if (mark >= merit) {return "Merit"}
    if (mark >= pass){return "Pass"}
    return "Fail"
}
```

- Can be called as follows

```
//as usual - pass all parameter in order
describeMark( mark: 62, pass: 40, merit: 60, distinction: 70)

//as usual but omitting trailing default parameters
describeMark( mark: 62, pass: 40)

//with only the non-default parameter(s)
describeMark( mark: 62)

//providing only the required param(s) first, in order
//followed by a subset of named default params in any order
describeMark( mark: 62, merit=65, pass=50)
```

Functions can be stored as variable

- So long as they are *anonymous* (note the missing method name)
- e.g. the following is valid

```
var greet = fun (forename: String, surname: String) : String {
    return "Hello $forename $surname"
}
```

- It would be called as follows

```
greet("John", "Smith")
```

- But we don’t usually write code like this, instead we use…

Assigning an existing function to a variable

- Consider this function

```
fun hello() : String{
    return "Hello"
}
```

- This calls the function and assigns the result

```
val greeting = hello()
```

- This, however, stores the function in a variable

```
val greetingFunction = ::hello
```

- It could then be called

```
greetingFunction()
```

Lambda expressions

- In its simplest form a lambda expression is an anonymous block of code.

```
{
    print("hello")
}
```

- They can be stored in a variable e.g.

```
val printHello = {
    print("hello")
}
```

- And can be called

```
printHello()
```

Lambda expressions

- If stored as a variable a lambda expression can also include parameters

```
val printHello = {name :String ->
    print("Hello $name")
}
```

- And can be called in the normal way

```
printHello("Kotlin")
```

But why bother?

- If we can store blocks of code as regular variables, we can use them in the same way. Consider our lambda from the previous slide

```
val printHello = {
    print("hello")
}
```

- If we wanted to write a method to do something 10 times, but that something could vary, we could set the ‘something’ as a parameter for the method
- First we need to consider the function type
  - It has no parameter and returns nothing (i.e. Unit – Kotlin’s void)
  - So it is `() -> Unit`

Why bother continued

- Function syntax where the parameter is a function is as follows

```
fun do10Times(thingToDo: ()-> Unit){
    for (i in 1 ≤ .. ≤ 10) {
        thingToDo()
    }
}
```

Parameter type

- This is known as a ‘higher order’ function
  - That is a function which takes a function as a parameter and/or returns a function

Calling a higher order function

- If the only required parameter is the function, brackets can be omitted and lambda syntax used

```
do10Times {
    println("Hello")
}
```

- If the function requires other parameters, these are passed in the same way as usual

```
doNTimes( n: 5){
    println("Hello")
}
```

Higher order functions with parameters

- Example:

```
fun filteredGrades(grades: List<Grade>, filter: (grade:Grade) -> Boolean ) : List<Grade>{
    val filteredGrades = mutableListOf<Grade>()
    for (grade in grades) {
        if (filter(grade)){ //apply the 'rule'
            filteredGrades.add(grade)
        }
    }
    return grades
}
```

- Here the ‘filter’ parameter is a function which takes a grade and returns a Boolean*.
- The intent is that when called, the rule will determine whether a grade should be included in a filtered list

*Ordinarily, we would just use the ‘filter’ method on a collection to achieve this aim, but this serves as an example

Lambdas with parameters and return types

- Parameters go inside the braces, -> indicates what is returned (return type inferred)
- Below shows the filtered grade method being called
  - Remember the code in the braces is the second parameter (a function)

```
val filteredGrades = filteredGrades(grades){ grade ->
    grade.mark > 40
}
```

- Kotlin will allow us to omit a single parameter from the lambda expression and refer to the value as ‘it’

```
val filteredGrades = filteredGrades(grades) { it: Grade
    it.mark > 40
}
```