CS315

Homework Assignment 3

Subprograms in Kotlin

Full Name: Funda Tan

ID: 21801861

Section: 2

Design Issues

Nested subprogram definitions

Kotlin allows nested subprogram definitions. In my example, I illustrated nested functions which are the function names subprogram and the inner function named subprogramInner.

Example:

```
//Nested subprogram definitions
fun subprogram(){
    fun subprogramInner(){
        println("In inner subprogram")
    }
    val myString = "In outer subprogram"
    subprogramInner()
    println(myString)
}
subprogram()
```

Output:

```
In inner subprogram
In outer subprogram
```

Explanation:

There are two functions in this example and the functions are nested. The outer function is subprogram and the inner function is subprogramInner. I called the inner subprogram from the outer subprogram., then I called the outer subprogram. This can be seen from the outputs.

Scope of local variables

Kotlin is a static scoped language. So, inner subprograms can reach the variables of the outer subprograms in the nested subprograms.

Example:

```
//Scope of local variables
fun subprogram2(){
    val local = 5
    println("Local is $local in subprogram2")
    fun subprogram3(){
        println("Local is $local in subprogram3")
        val local2 = 10
        println("Local2 is $local2 in subprogram3")
    }
    subprogram3()
}
subprogram2()
```

Output:

```
Local is 5 in subprogram2
Local is 5 in subprogram3
Local2 is 10 in subprogram3
```

Explanation:

In the example, we can see that subprogram3 can reach the variable called “local” from its outer subprogram (which is subprogram2). Also, I defined a variable called “local2” and println statement can reach this variable because “local2” is local to subprogram3.

Parameter passing methods

In Kotlin, parameter passing is pass-by-value.

Example:

```
//Parameter passing methods
fun subprogram(a: Int): Int{
    return a * 2
}
println(subprogram(5))
```

Output:

```
10
```

Explanation:

We pass 5 as a parameter and this parameter is passed by value.

Keyword and default parameters

In Kotlin, “fun”, “infix” and “return” keywords can be used in defining subprograms.

Example 1: for “fun” and “return” keywords, and default parameters

```
//fun and return keywords, default parameters
fun subprogram4(string: String = "default argument"): String{
    println(string)
    return string
}

subprogram4()
subprogram4("my argument")
```

Output:

```
default argument
my argument
```

Explanation:

“fun” is used before the signature of the function when defining a function. “return” is used to return a value from the function. Kotlin allows default arguments in subprograms. When we called the subprogram4 function without parameters, function uses its default argument. From the output, we see that when we called the subprogram4 without any argument, it prints out “default argument” because I

assigned its default argument as a string and its value is “default argument”. If we passed “my argument", it prints “my argument” as seen in the output.

Example 2: for “infix” keyword

```
//infix keyword
infix fun Int.keywords(myInt: Int){
    println(this * myInt)
}
3 keywords 2
```

Output:

```
6
```

Explanation:

In the example, I demonstrated the “infix” keyword. Infix keyword enables us to call the function as “3 keywords 2”. The output comes out as 6 as we can see from the output.

Closures

In Kotlin, in nested subprograms, a inner subprogram's closure is the outer subprogram's local variables.

```
//Closures
fun subprogram5(){
    val x = 2
    fun a(){
        println(x)
    }
    a()
}
subprogram5()
```

Output:

```
2
```

Explanation:

In the code, we can see that a() can reach the local variable "x" of the outer function (subprogram5).

Evaluation of Kotlin in terms of readability and writability of subprogram syntax

Kotlin has meaningful keywords like "fun", "return" and "infix" so they are increasing Kotlin's readability.

Kotlin enables to define nested subprograms and this increases the writability. Also, "infix" keyword increases Kotlin's writability since it enables us to write the subprogram in a different way.

Learning strategy

I first checked the documentation posted in the homework description. Then I learnt Kotlin's syntax. I practiced the syntax of the Kotlin language, I then did the experiments. I studied with some online tutorials about Kotlin. My main source when learning was the documentations of Kotlin. I used online Kotlin compiler to write the code.

Online compiler that I used for this assignment:

https://play.kotlinlang.org