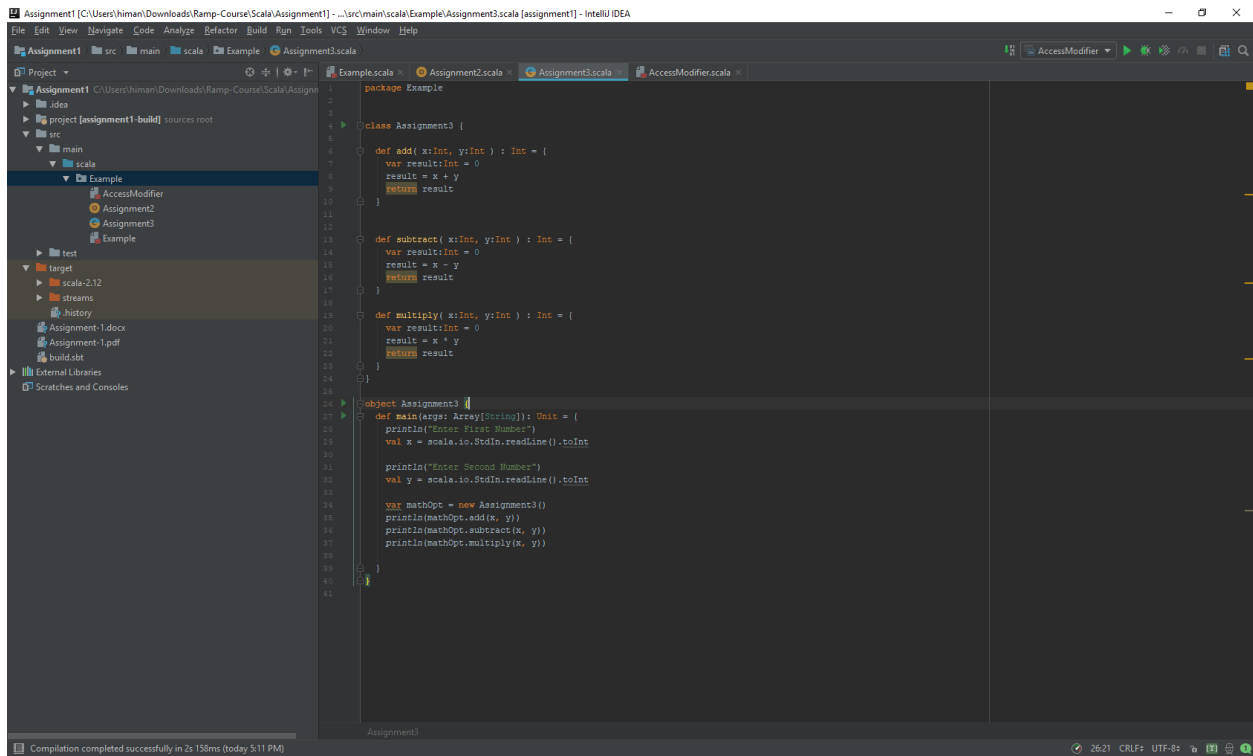


Himanshu Goyal

Assignment 3

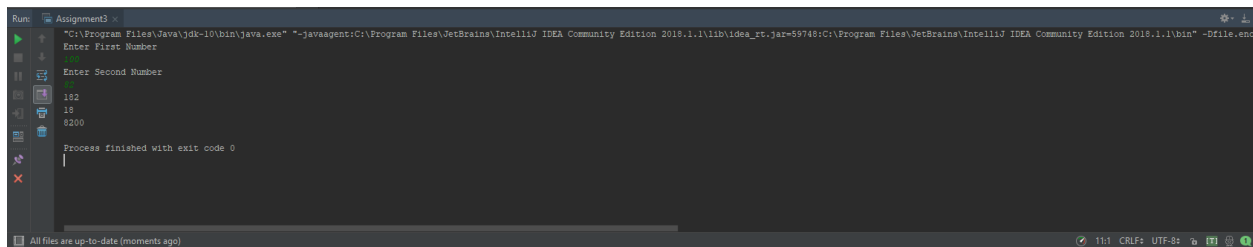
Scala

Q 1) Write a Scala class which has 3 functions add, subtract and multiply and a main method which executes these three functions.



```
1 package Example
2
3 class Assignment3 {
4     def add(x: Int, y: Int): Int = {
5         var result: Int = 0
6         result = x + y
7         return result
8     }
9
10    def subtract(x: Int, y: Int): Int = {
11        var result: Int = 0
12        result = x - y
13        return result
14    }
15
16    def multiply(x: Int, y: Int): Int = {
17        var result: Int = 0
18        result = x * y
19        return result
20    }
21
22    object Assignment3 {
23        def main(args: Array[String]): Unit = {
24            println("Enter First Number")
25            val x = scala.io.StdIn.readLine().toInt
26
27            println("Enter Second Number")
28            val y = scala.io.StdIn.readLine().toInt
29
30            var mathOpt = new Assignment3()
31            println(mathOpt.add(x, y))
32            println(mathOpt.subtract(x, y))
33            println(mathOpt.multiply(x, y))
34        }
35    }
36}
```

output



```
Run: Assignment3
C:\Program Files\Java\jdk-10\bin\java.exe -javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2018.1.1\lib\idea_rt.jar=59748:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2018.1.1\bin -Dfile.encoding=UTF-8
Enter First Number
Enter Second Number
102
10
8200
Process finished with exit code 0
```

Q 2) Create scala classes and objects which makes use of access modifiers.

```

1 class AccessModifier {
2     private def private_print() {Unit {
3         println("Hello...This is Private Method from AccessModifier Class")
4     }}
5
6     protected def protected_print() {Unit {
7         println("Hello...This is Protected Method from AccessModifier Class")
8     }}
9
10    def public_print() {Unit {
11        println("Hello...This is Public Method from AccessModifier Class")
12    }}
13
14    //Inner Class
15    class InnerTestClass {
16        private def inner_print() {Unit {
17            println("Hello...I am inside inner class's method")
18        }}
19    }
20
21    // Child Class
22    class ChildTestClass extends AccessModifier {
23        var inner = new InnerTestClass()
24        inner.inner_print()
25        protected_print() // Child Class can access protected protected_print() from Parent AccessModifier Class
26    }
27
28    object AccessModifier {
29        def main(args: Array[String]): Unit = {
30
31            //Protected Modifier Test
32            new ChildTestClass()
33
34            //Public AccessModifier Test : Public method can access from any where.
35            val modTest = new AccessModifier()
36            modTest.public_print()
37        }
38    }
39 }

```

output

```

Run: AccessModifier x
+ "C:\Program Files\Java\jdk-10\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2018.1\lib\idea_rt.jar-53889rC:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2018.1\bin" -Dfile.encoding=UTF-8
Hello...This is Private Method from AccessModifier Class
Hello...I am inside inner class's method
Hello...This is Protected Method from AccessModifier Class
Hello...This is Public Method from AccessModifier Class
Process finished with exit code 0

```

Q 3) In a few sentences, elaborate your understanding of case classes and normal classes.

Case classes can be seen as plain and immutable data-holding objects that should exclusively depend on their constructor arguments.

This functional concept allows us to

- use a compact initialization syntax (Node(1, Leaf(2), None))
- decompose them using pattern matching
- have equality comparisons implicitly defined

In combination with inheritance, case classes are used to mimic algebraic datatypes.

If an object performs stateful computations on the inside or exhibits other kinds of complex behavior, it should be an ordinary class.

Defining a case class gives you a lot of boilerplate code for free:

- Getters are generated for the constructor parameters. Setters are only generated when the parameters are declared as `var`. They are `Val` by default.
- A nice `toString` method is generated.
- An `equal` and `hashCode` methods are generated.
- A `copy` method is generated to clone an object.
- An `apply` method is generated, removing the need to use the `new` keyword when creating a new instance of the class.