Stream Programming - 2019
Laboratory 3

Task 1 Write program in Scala with the following requirements. The program should calculate the field and perimeter of the following geometrical figures: circle, square, rectangle, rhombus, regular pentagon, regular hexagon. For this purpose, a hierarchy should be created to support particular types of figures. The root of this hierarchy should be the abstract class tt Figure, implementing an interface containing method declarations for calculating the perimeter and field of a given figure. We assume that the abstract class `Quadrangle` and classes: `Circle`, `Pentagon`, `Hexagon` inherit from the class `Figure`. Then, the classes: `Square`, `Rectangle`, `Rhombus` inherit from the class `Quadrangle`. Create appropriate methods in child classes that will calculate the perimeter and field in a manner specific to a given geometrical figure. In the command line you can specify the following types of geometric figures (c - circle, c-quadrangle, p-pentagon, s-hexagon) and their parameters, where: the circle has one parameter: `radius`, quadrangle has five parameters: `side1, side2, side3, side4, angle`, pentagon and hexagon one parameter: `side`.

Hint 1 In object-oriented programming, inheritance is the mechanism of basing an object or class upon another object or class, retaining similar implementation.

If a class `A` (subclass) inherit from the class `B` (superclass), the class `A` retains all the method and attributes of class `B` and adds its own specific methods. In this way, the classes may form class hierarchies. The simplest form of inheritance looks like this:

```
class Van3 extends Car4
```

The more difficult example is shown on the following listings:

```
class Car (val carType: String, val capacity : Int) {
private val year = 2010
private def getCapacity() : Int ={
  return capacity
}
def printType() ={
  println(carType)
}
protected def printType() ={
  println(carType)
}
}

class Van(override val carType: String) extends Car (carType: String){
def printType1() ={
   printType()
}
def getCapacity1()={
  // getCapacity() does not compile
}
}
```

We present two classes: `Car` and `Van`. The class `Car` has one private attribute `year` and three methods: `getCapacity`, `printType` and `printType` with the access `private`, `public` and `protected`, respectively. The class `Van` inherits from the class `Car`. It means that the class `Van` can use all not private attributes and methods from `Car`. Additionally, the class `Van` implements two new methods: `printType1` and `getCapacity1`. In class `Van` the method `printType` is used inside `printType1`. It comes from the fact that `printType` is defined as protected. Thus, it may be used in the class where is defined and in all inherited classes. On the other hand, the method `getCapacity` cannot be used in `getCapacity1`, because is defined as private and is not accessible in the inherited classes.

In the above listings the constructor inheritance is shown.

```
class Van(override val carType: String) extends Car (carType: String){
}
```

Please note that `Car` has only one constructor. So, the parameters in `Van` constructor must extend the parameters of `Car` ones.