

## Scala IV

### Task 1

Write a simple program to show inheritance in scala.

**Code:**

```
class inheritance {  
    var bigdata:Float = 1000  
}  
  
class hadoop extends inheritance{  
    var hadoopfee:Float = 24000  
    println ("bigdata=" + bigdata)  
    println("hadoopfee=" + hadoopfee)  
}  
  
object mainclass{  
    def main(args:Array[String]): Unit =  
    {  
        new hadoop()  
    }  
}
```

**Output:**

The screenshot shows an IDE window titled "inheritance [~/IdeaProjects/inheritance] - .../src/main/scala/inheritance.scala [inheritance]". The code defines a class `inheritance` with a variable `bigdata: Float = 1000`. A class `hadoop` extends `inheritance` and adds a variable `hadoopfee: Float = 24000`, along with two `println` statements. An `object mainclass` contains a `main` method that creates a new `hadoop` object. The Run console at the bottom shows the output: `bigdata=1000.0` and `hadoopfee=24000.0`, with a message "Process finished with exit code 0".

```
1 class inheritance {
2     var bigdata: Float = 1000
3 }
4
5
6 class hadoop extends inheritance {
7     var hadoopfee: Float = 24000
8     println("bigdata=" + bigdata)
9     println("hadoopfee=" + hadoopfee)
10 }
11
12 object mainclass {
13     def main(args: Array[String]): Unit = {
14         new hadoop()
15     }
16 }
17
18
```

Run: mainclass x

```
/usr/java/jdk1.8.0_151/bin/java ...
bigdata=1000.0
hadoopfee=24000.0
Process finished with exit code 0
```

## Task 2

Write a simple program to show multiple inheritance in scala

Code:

```
trait A {
    var bigdata: Int = _
    def value = {
        bigdata = bigdata + 1000
    }
}

trait B {
    var hadoopfee: Int = _
    def value = {
        hadoopfee = hadoopfee + 4000
    }
}

class AB extends A with B {
    bigdata = 20000;
    hadoopfee = 30000;
    override def value = {
        super[A].value
        super[B].value
    }
}

object mainclass_MI {
    def main(args: Array[String]): Unit = {

```

```

var ab = new AB
ab.value
println("bigdataMI =" + ab.bigdata)
println("HadoopfeeMI =" + ab.hadoopfee)
}
}

```

Output:

The screenshot shows the IntelliJ IDEA IDE with a Scala project named 'inheritance'. The code in 'multiple.scala' defines two traits, A and B, and a class AB that extends A with B. Trait A has a 'bigdata' variable and a 'value' method that increments it by 1000. Trait B has a 'hadoopfee' variable and a 'value' method that increments it by 4000. Class AB has initial values of 20000 for 'bigdata' and 30000 for 'hadoopfee', and its 'value' method calls the 'value' methods of both A and B. The 'mainclass\_MI' object contains a 'main' method that creates an instance of AB, calls its 'value' method, and prints the results.

```

1 trait A {
2   var bigdata: Int = _
3   def value: Unit = {
4     bigdata = bigdata + 1000
5   }
6 }
7 trait B {
8   var hadoopfee: Int = _
9   def value: Unit = {
10    hadoopfee = hadoopfee + 4000
11  }
12 }
13 class AB extends A with B {
14   bigdata = 20000
15   hadoopfee = 30000
16   override def value: Unit = {
17     super[A].value
18     super[B].value
19   }
20 }
21 object mainclass_MI {
22   def main(args: Array[String]): Unit = {
23     var ab = new AB
24     ab.value
25     println("bigdataMI =" + ab.bigdata)
26     println("HadoopfeeMI =" + ab.hadoopfee)
27   }
28 }

```

The Run window shows the output of the program:

```

Run: mainclass_MI
bigdataMI =21000
HadoopfeeMI =34000
Process finished with exit code 0

```

Compilation completed successfully in 4 s 245 ms (moments ago)

### Task 3

Write a partial function to add three numbers in which one number is constant and two numbers can be passed as inputs and define another method which can take the partial function as input and squares the result.

Code:

```

object mainclass_PC {
  def main(args: Array[String]): Unit = {

    val addnum = new PartialFunction[Int, Int]
    {
      def apply(x: Int) = 42 / x
      def isDefinedAt(x: Int) = x != 0
    }
  }
}

```

```

    def add(x: Int,y: Int)= x + y + 20
  }

  def square (z: Int) = z * z

  def res = square(addnum.add(20,10))

  println (res.intValue())
}
}

```

## Output:

The screenshot shows the IntelliJ IDEA IDE with a Scala file named `partial.scala` open. The code defines a `mainclass_PC` object with a `main` method. Inside `main`, a `PartialFunction` `addnum` is created with an `add` method that returns `x + y + 20`. A `square` method is also defined, and `res` is calculated as `square(addnum.add(20,10))`. The result is printed using `println (res.intValue())`. The Run tab at the bottom shows the execution output: `/usr/java/jdk1.8.0_151/bin/java ... 2500` and `Process finished with exit code 0`. The output `2500` is highlighted with a red box.

## Task 4

Write a program to print the prices of 4 courses of Acadgild:

Android App Development -14,999 INR Data Science - 49,999 INR Big Data Hadoop & Spark Developer – 24,999 INR Blockchain Certification – 49,999 INR using match and add a default condition if the user enters any other course.

## Code:

```

object mainclass_T {
  def main(args: Array[String]): Unit = {

    val android = Tuple2("Android App Development",14999)

```

```

val data = Tuple2("Data Science",49999)
val big = Tuple2("Big Data Hadoop",24999)
val block = Tuple2("Block Chain Certification",49999)

val courselist = List(android,data,big,block)

val price = courselist.foreach { tuple => {
  tuple match {

    case d => println(s"${d._1},${d._2}")
    case _ => None
  }

}

}

}
}

```

## Output:

The screenshot shows an IDE with a Scala file named `matchcase.scala` in the `src/main/scala` directory. The code defines a `main` function that creates a list of course tuples and prints their details using a `match` expression. The output window shows the execution results, which are the course names and their prices, followed by a message indicating the process finished successfully.

```

1  object mainclass_T {
2  def main(args: Array[String]): Unit = {
3
4      val android = Tuple2("Android App Development", 14999)
5      val data = Tuple2("Data Science", 49999)
6      val big = Tuple2("Big Data Hadoop", 24999)
7      val block = Tuple2("Block Chain Certification", 49999)
8
9      val courselist = List(android, data, big, block)
10
11      val price = courselist.foreach { tuple => {
12          tuple match {
13              case d => println(s"${d._1},${d._2}")
14              case _ => None
15          }
16      }
17  }
18  }
19  }
20  }
21  }

```

Run: mainclass\_T x

```

/usr/java/jdk1.8.0_151/bin/java ...
Android App Development, 14999
Data Science, 49999
Big Data Hadoop, 24999
Block Chain Certification, 49999
Process finished with exit code 0

```

Compilation completed successfully with 3 warnings in 5 s 712 ms (moments ago)