

ASSIGNMENT SCALA - 4

PROBLEM STATEMENT

TASK 1: Write a simple program to show inheritance in scala.

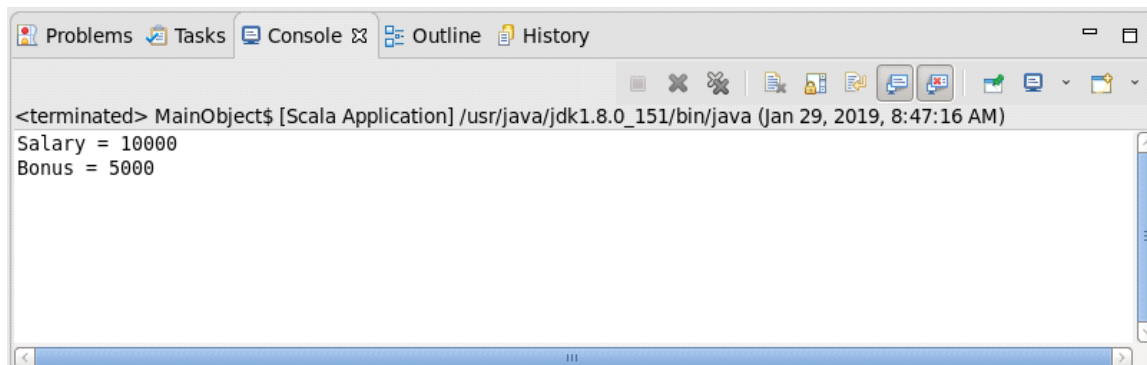
EXPLANATION : HERE A CLASS EMPLOYEE IS MADE IN WHICH SALARY OF A EMPLOYEE IS DECLARED INSIDE THE CLASS EMPLOYEE AND ANOTHER CLASS PROGRAMMER IS IS MADE WHICH IS DERIVED FROM THE CLASS EMPLOYEE USING THE KEYWORD "EXTENDS" IN MEANS THAT THE CLASS PROGRAMMER IS A CLASS WHICH IS INHERITED FROM ITS BASE CLASS EMPLOYEE. THUS WHEN WE PRINT THE INFORMATION IN CLASS PROGRAMMER IT WILL ALSO DISPLAY THE DETAILS OF CLASS EMPLOYEE WHICH IS INHERITED TO IT. THUS THIS IS CALLED AS SINGLE INHERITANCE.

CODE :

```
class Programmer extends Employee{  
}
```

SOLUTION REPORT :

```
class Employee{  
    var salary:Float = 10000  
}  
  
//Derived From Base Class Employee  
class Programmer extends Employee {  
    var bonus:Int = 5000  
    println("Salary = "+salary)  
    println("Bonus = "+bonus)  
}  
  
object MainObject{  
    def main(args:Array[String]){  
        new Programmer()  
    }  
}
```



TASK 2 : Write a simple program to show multiple inheritance in scala

EXPLANATION: IN SCALA MULTIPLE INHERITENCE IS NOT SUPPORTED BUT WE CAN SHOW

THE MULTIPLE INHERITANCE WITH HELP OF TRAITS. AS SCALA DOES NOT ALLOW FOR MULTIPLE INHERITANCE BUT ALLOWS TO EXTEND MULTIPLE TRAITS. TRAITS ARE USED TO SHARE INTERFACES AND FIELDS BETWEEN CLASSES. CLASSES AND OBJECTS CAN EXTEND TRAITS BUT TRAITS CANNOT BE INSTANTIATED AND THEREFORE HAVE NO PARAMETERS. SO HERE CREATED TWO TRAITS A AND B WITH DETAILS HAVING DISTANCE AND SPEED RESPECTIVELY. THEN CREATED A CLASS AB WHICH WILL EXTEND TRAITS A AND B. USING TRAITS AS SUPER HERE CLASS AB HAS IHERITED A WITH B. THUS MULTIPLE INHERITANCE OCCURS HERE.

SOLUTION REPORT:

```
package scala_assignment
```

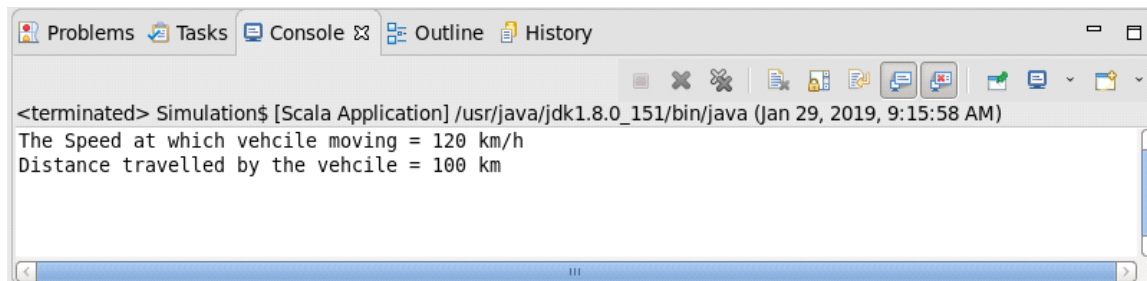
```
trait A
{
    var distance: Int = _
    def action = {
        distance = distance + 5
    }
}
trait B
{
    var speed: Int = _
    def action = {
        speed = speed + 100
    }
}
```

```
//Multiple Inheritance where Class A and Class B are the derived from Class AB
```

```
class AB extends A with B
{
    distance = 95;
    speed = 20;
    override def action = {
        super[A].action
        super[B].action
    }
}
```

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

        var ab = new AB
        ab.action
        println(s"The Speed at which vehcile moving = ${ab.speed} km/h")
        println(s"Distance travelled by the vehcile = ${ab.distance} km")
    }
}
```



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:

```
val add = (a: Int, b: Int, c: Int) => a + b + c
val sum = add(4, _: Int, _: Int)
```

EXPLANATION: SO HERE CREATED PARTIAL FUNCTION FOR ADDITION OPERATION IN WHICH WE DO THE SUM OPERATION OF THREE VARIABLES IN WHICH ONE NUMBER IS MADE CONSTANT AND OTHER TWO NUMBERS ARE PASSED AS ARGUMENTS.

CODE:

```
val square : PartialFunction[Int, Int] = {
    case x if x!= 0 => x*x
}
```

EXPLANATION: A PARTIAL FUNCTION CAN ONLY HAVE TWO ARGUMENTS ONE FOR TESTING AND OTHER TO RETURN THE OUTPUT. HERE VARIABLE X IS USED IN "CASE" TO CHECK WHETHER WHATEVER VALUE IS PASSED TO IT SHOULD NOT BE EQUAL TO 0. IF THE VALUE PASSED TO THE FUNCTION IS NOT 0 THEN THE FUNCTION WILL DO THE SQUARE OF WHATEVER ARGUMENT IS PASSED TO IT AND RETURNS THE SQUARED RESULT. HERE THIS FUNCTION WILL RETURN INTEGER VALUE AS THE FUNCION IS DEFINED AS "INT".

"**val add = (a: Int, b: Int, c: Int) => a + b + c**" --> THIS IS A LAMBDA FUNCTION USED FOR SUM OPERATION

"**val sum = add(4, _: Int, _: Int)**" --> HERE VALUE FOR VARIABLE A IS CONSTANT AND "_" TELLS THE COMPILER TO REPLACE IT WITH WHATEVER ARGUMENTS ARE PASSED TO THE FUNCTION.

```
package scala_assignment
```

```
object functions {
```

```
    def main (arg: Array[String]) { //Partially applied functions
```

```
        val add = (a: Int, b: Int, c: Int) => a + b + c
        val sum = add(4, _: Int, _: Int)
```

```
        val square : PartialFunction[Int, Int] = { //Partial Functions
```

```
            case x if x!= 0 => x*x
```

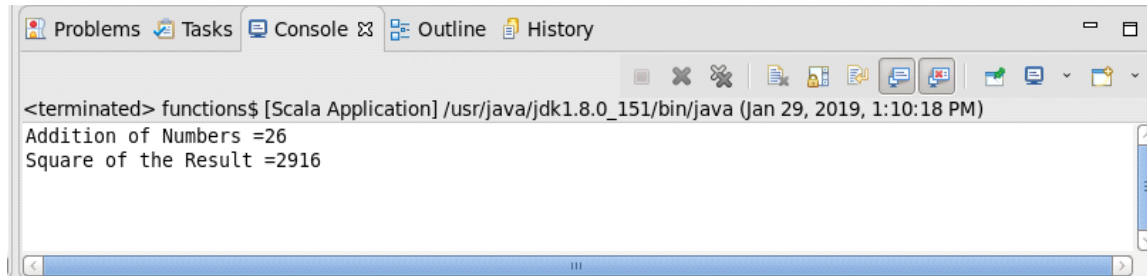
```
        }
```

```
        println("Addition of Numbers =" + sum(10,12))
```

```

    println("Square of the Result =" + square(54))
  }
}

```



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 â€" 24,999 INR

Blockchain Certification â€" 49,999 INR

using match and add a default condition if the user enters any other course.

EXPLANATION: HERE A MATCH WORKS AS A SWITCH STATEMENT WHICH HAS MULTIPLE INPUTS. THESE INPUTS ARE PROVIDED USING KEYWORD "CASE". AS THE HERE THE OUTPUT SHOULD BE IN STRING DATA TYPE THUS CREATED A FUNCTION TO RETURN VALUE WITH STRING DATA TYPE. SO WHEN , THE COMPILER EXECUTES THE MAIN FUNCTION THEN:

```

[ println("\nEnter name of the course(without space):")
  var c = scala.io.StdIn.readLine.toUpperCase()
  println("\nPrice of the Course = "+acadgild(c)) ]

```

AS SOON AS THE COMPILER EXECUTES THE ABOVE STATEMENT. IT WILL READ THE WHATEVER INPUT IS PROVIDED FROM THE USER FOR EXAMPLE THIS CODE PRODUCES THE PRICES FOR THE CORRESPONDING COURSE AVAILABLE. WHEN THE USER ENTERS A COURSE NAME IT WILL READ THE STRING AND THEN WILL SAVE IT THE VARIABLE DECLARED "c" . WHEN COMPILER EXECUTES "acadgild(c)" THE COMPILER WILL DIRECTLY GO TO FUNCTION DEFINED i.e FUNCTION "acadgild()". THE VALUE INSIDE THE "c" WILL BE MATCHED WITH THE AVAILABLE VALUES IN CASES DECLARED IN THE MATCH AND OF AVAILABLE WILL RETURN THE RESULT.

[case _ => "No such course available"] --> THIS IS A DEFAULT CONDITION TO PRINT THE ERROR MESSAGE TO THE USER WHEN THE USER ENTERS A VALUE WHICH IS NOT THE PART OF THE MATCH CASE.

SOLUTION REPORT:

```

package scala_assignment

```

```

object Courses {

    def acadgild (a: String) = a match {

        case "ANDROIDAPP" => "Android App Development --> 14,999 INR"

        case "DATASCIENCE" => "Data Science --> 49,999 INR"

        case "BIGDATA" => "Big Data Hadoop & Spark Developer --> 24,999 INR"

        case "BLOCKCHAIN" => "Blockchain Certification --> 49,999 INR"

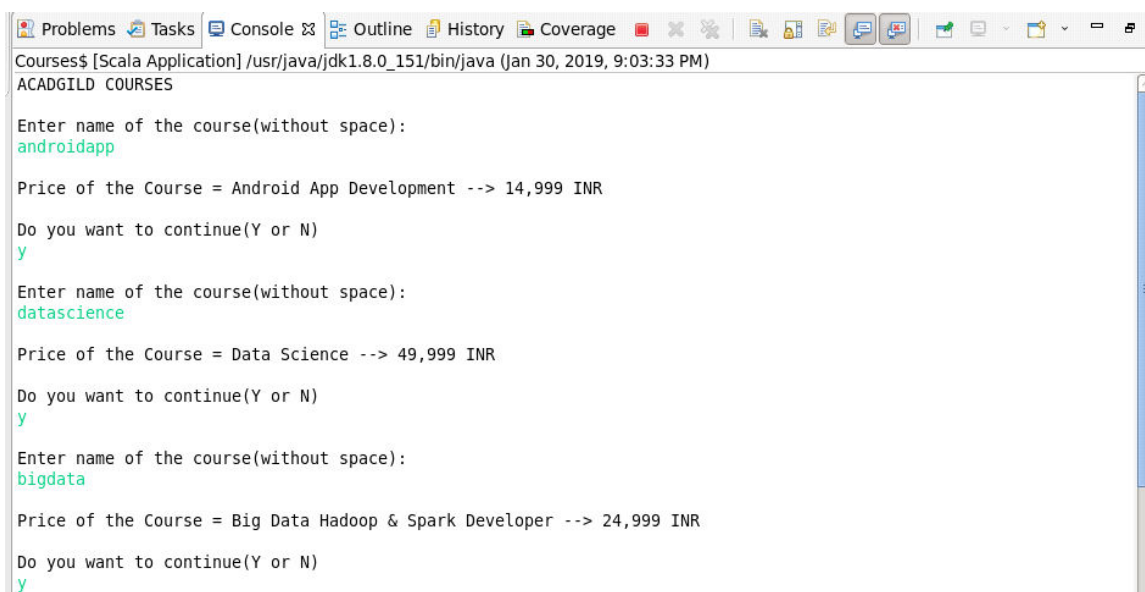
        case _ => "No such course available"
    }

    def main(arg: Array[String]) {
        var continue = ""
        println("ACADGILD COURSES")
        do
        {
            println("\nEnter name of the course(without space):")
            var c = scala.io.StdIn.readLine.toUpperCase()

            println("\nPrice of the Course = "+acadgild(c))

            println("\nDo you want to continue(Y or N)")
            continue = scala.io.StdIn.readLine.toUpperCase()
        }
        while(continue == "Y")
    }
}

```



```

Courses$ [Scala Application] /usr/java/jdk1.8.0_151/bin/java (Jan 30, 2019, 9:03:33 PM)
ACADGILD COURSES

Enter name of the course(without space):
androidapp

Price of the Course = Android App Development --> 14,999 INR

Do you want to continue(Y or N)
y

Enter name of the course(without space):
datascience

Price of the Course = Data Science --> 49,999 INR

Do you want to continue(Y or N)
y

Enter name of the course(without space):
bigdata

Price of the Course = Big Data Hadoop & Spark Developer --> 24,999 INR

Do you want to continue(Y or N)
y

```

Enter name of the course(without space):

blockchain

Price of the Course = Blockchain Certification --> 49,999 INR

Do you want to continue(Y or N)

y

Enter name of the course(without space):

webdevelopment

Price of the Course = No such course available

Do you want to continue(Y or N)

<

|||

>