Assignment 17 Scala IV

Task 1

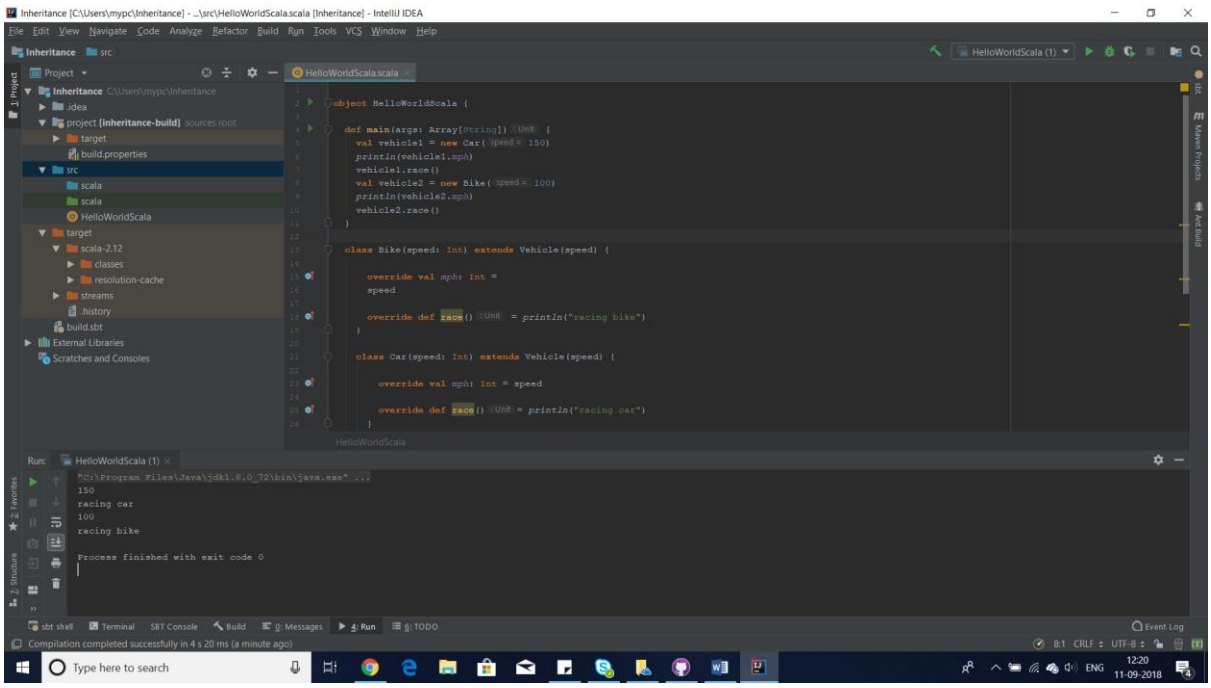
Write a simple program to show inheritance in scala.

Like Java, Scala supports single inheritance, not multiple inheritance.

“extends” keyword should be used when a child class inherits a parent class.

```
//Create a Parent class called Vehicle.
Class vehicle(speed:Int) {
Val mph :Int=speed
def race()= println(“Racing”)
}
//create Child classes Car and Bike that inherit the Vehicle class.
//We can override the mph and race() methods accordingly.
class Bike (speed:Int) extends vehicle{
def main(args:Array[String]){
override val mph :Int=speed
override def race()=println(“racing bike”)
}
class Car (speed:Int) extends vehicle{
def main(args:Array[String]){
override val mph:Int=speed
override def race()=println(“racing car”)
}
Object HelloWorldScala{
def main(args:Array[String]) {
val vehicle1=new Car(150)
```

```
println(vehicle1.mph)

vehicle1.race()

val vehicle2 =new Bike(100)

println(vehicle2.mph)

vehicle2.race()

}

}
```



Task 2

Write a simple program to show multiple inheritance in scala

Scala supports multiple inheritance using Traits.

Traits in Scala are very much similar to Java Interfaces. There is just one exception if we compare traits with interfaces in Java (Java 7 and earlier version) and that is traits can also have methods with implementation.

Rules for defining Traits

- Traits can have methods as well as variables
- Traits may or may not have abstract methods
- Traits doesn't have constructor with parameters
- One class can be extended from multiple traits using extend and with keywords
- One class can also implement multiple traits using extend and with keywords

```
//Trait can have abstract method
trait Flyable{
   def fly()
}

//Trait may not have any abstract method
trait Speakable{
   def makeNoice() = println("Generic Sound.....!!!!!")
}

//Trait can extand one or more traits
trait Quackable extends Speakable{

   //have to override method of parent
   override def makeNoice() = println("Quack Quack.....!!!!!")

   def quack() = makeNoice
}

//one class can extand multiple traits
class Duck extends Quackable with Flyable{
   def swim() = println("Duck is Swimming.....!!!!!")

   //have to override method of parent
   override def fly() = println("Duck is Flying.....!!!!!")
}

object MainClass {
  def main(args:Array[String]){
   var duck = new Duck
   duck.swim
   duck.fly
   duck.makeNoice
   duck.quack
```

```
  }
}
```

```
object MainClass {
  def main(args:Array[String]): Unit ={
    var duck =new Duck()
    duck.swim
    duck.fly
    duck.makeNoise
    duck.quack

  }
  //trait can have abstract method
  trait Flyable{
    def fly()
  }
  //Trait may not have any abstract method
  trait Speakable{
    def makeNoise() = println("Generic Sound.....!!!!!")
  }
  //Trait may not have any abstract method
  trait Quackable extends Speakable{

    //have to override method of parent
    override def makeNoise() = println("Quack Quack.....!!!!!")

    def quack() = makeNoise
  }
  class Duck extends Quackable with Flyable{
    def swim() = println("Duck is Swimming.....!!!!!")
    //override def makeNoise() = println("Duck is making sound.....!!!!!")
    override def quack() = makeNoise


    //have to override method of parent
    override def fly() = println("Duck is Flying.....!!!!!")
  }

}
```

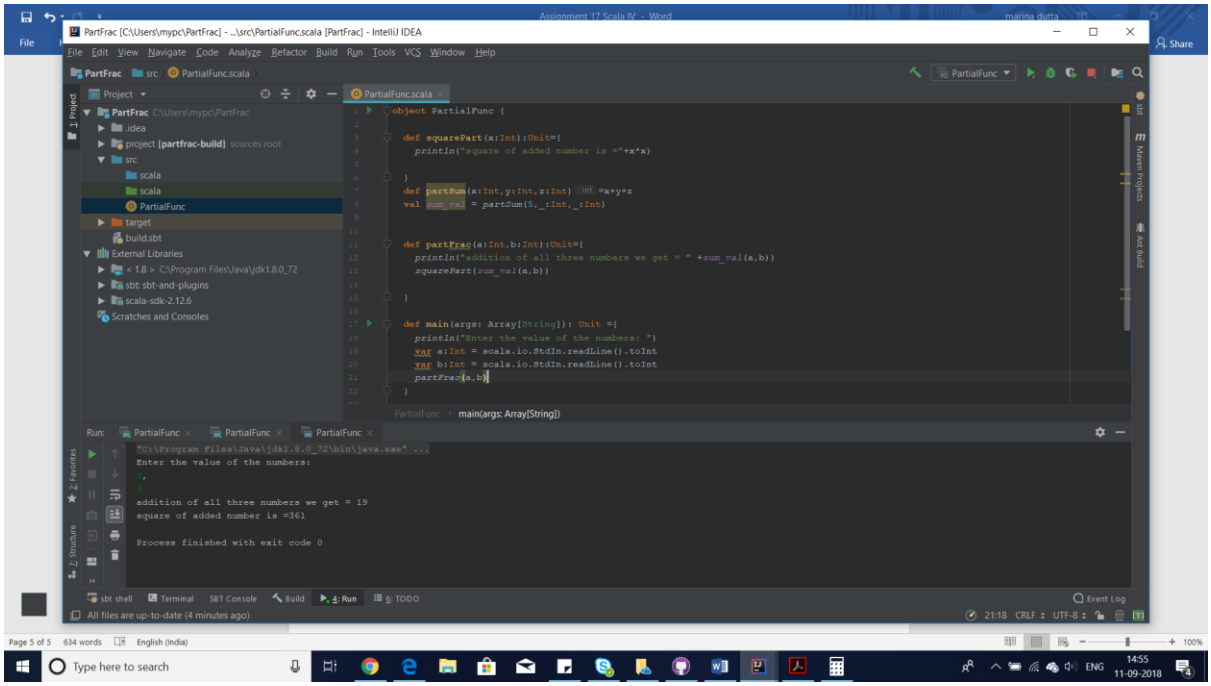
Task 3

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.

Solution:

```
// singleton object to call the functions
object PartialFunc {


 //Sqaure function that takes an integer & returns square of that integer.
def squarePart(x:Int):Unit={
```

```
   println("square of added number is ="+x*x)


 }
//A Partial function to add two integers passed during runtime with one pre-defined integer
& returning the sum as Int
 def partSum(x:Int,y:Int,z:Int)=x+y+z
 val sum_val = partSum(5,_:Int,_:Int)


//A function to call both the functions using values passed from main method. //A function
to call both the functions using values passed from main method.
 def partFrac(a:Int,b:Int):Unit={
//Printing sum of all the the values adding a & b with the constant input.
   println("addition of all three numbers we get = " +sum_val(a,b))
//Calling the squareFun function & passing returned value of partial function as arguments
to it.
   squarePart(sum_val(a,b))


 }
//Entry point for the application.
 def main(args: Array[String]): Unit ={
   println("Enter the value of the numbers: ")
   var a:Int = scala.io.StdIn.readLine().toInt
   var b:Int = scala.io.StdIn.readLine().toInt
   partFrac(a,b)
 }
```



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

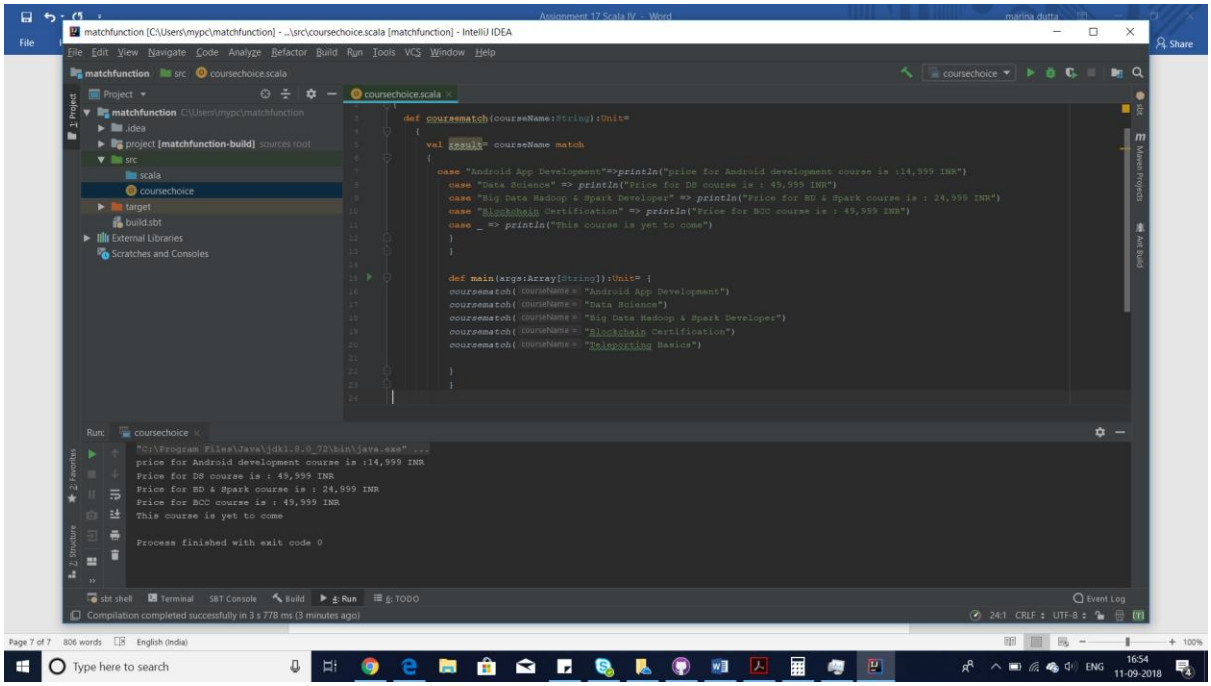
using match and add a default condition if the user enters any other course.

Solution:

Pattern matching is a mechanism for checking a value against a pattern. Here we are using pattern matching to find the prices of the following courses.

```
object coursechoice
{
  def coursematch(courseName:String):Unit=
    {
      val result= courseName match
      {
```

```
      case "Android App Development"=>println("price for Android development course is :14,999 INR")
       case "Data Science" => println("Price for DS course is : 49,999 INR")
       case "Big Data Hadoop & Spark Developer" => println("Price for BD & Spark course is : 24,999 INR")
       case "Blockchain Certification" => println("Price for BCC course is : 49,999 INR")
       case _ => println("This course is yet to come")
       }
       }

       def main(args:Array[String]):Unit= {
       coursematch("Android App Development")
       coursematch("Data Science")
       coursematch("Big Data Hadoop & Spark Developer")
       coursematch("Blockchain Certification")
       coursematch("Teleporting Basics")

       }
       }
```


matchfunction [C:\Users\mypc\matchfunction] - ...\src\coursechoice.scala [matchfunction] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
matchfunction src coursechoice.scala
coursechoice
Project
matchfunction C:\Users\mypc\matchfunction
.idea
project [matchfunction-build] sources root
src
scala
coursechoice
target
build.sbt
External Libraries
Scratches and Consoles
coursechoice.scala
def coursematch(courseName:String):Unit=
{
val result= courseName match
{
case "Android App Development"=>println("price for Android development course is :14,999 INR")
case "Data Science" => println("Price for DS course is : 49,999 INR")
case "Big Data Hadoop & Spark Developer" => println("Price for BD & Spark course is : 24,999 INR")
case "Blockchain Certification" => println("Price for BCC course is : 49,999 INR")
case _ => println("This course is yet to come")
}
}
def main(args:Array[String]):Unit= {
coursematch(courseName = "Android App Development")
coursematch(courseName = "Data Science")
coursematch(courseName = "Big Data Hadoop & Spark Developer")
coursematch(courseName = "Blockchain Certification")
coursematch(courseName = "Teleporting Basics")
}
}
Run: coursechoice
"C:\Program Files\Java\jdk1.8.0_72\bin\java.exe" ...
price for Android development course is :14,999 INR
Price for DS course is : 49,999 INR
Price for BD & Spark course is : 24,999 INR
Price for BCC course is : 49,999 INR
This course is yet to come
Process finished with exit code 0
sbt shell Terminal SBT Console Build Run TODO
Event Log
Compilation completed successfully in 3 s 778 ms (3 minutes ago)
24:1 CRLF UTF-8
Page 7 of 7 806 words English (India)
Type here to search
ENG
16:54
11-09-2018