

Session 17 - Scala Session - IV

Assignment 1

Task 1:

Write a simple program to show inheritance in scala.

```
package acadgild.Task1

object Inheritance {
  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()
  }
  class Vehicle(speed:Int){
    val mph: Int = speed
    def race() = println("Racing")
  }
  class Car(speed:Int) extends Vehicle(speed){
    override val mph: Int = speed
    override def race() = println("Racing Car")
  }
  class Bike(speed:Int) extends Vehicle(speed){
    override val mph: Int = speed
    override def race() = println("Racing Bike")
  }
}
```

1. Create a Parent class called Vehicle.
2. Create Child classes Car and Bike that inherit the Vehicle class.
3. We can override the mph and race() methods accordingly.
4. Inside the main method, we create a Car object and a Bike object and we then access there property mph and method race.

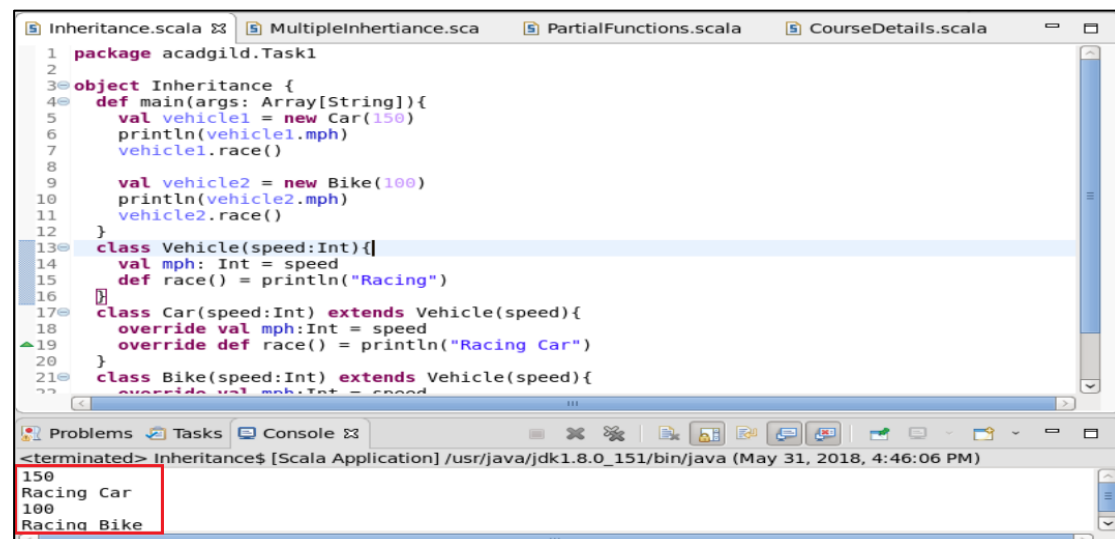
Output :

150

Racing Car

100

Racing Bike



```
1 package acadgild.Task1
2
3 object Inheritance {
4   def main(args: Array[String]){
5     val vehicle1 = new Car(150)
6     println(vehicle1.mph)
7     vehicle1.race()
8
9     val vehicle2 = new Bike(100)
10    println(vehicle2.mph)
11    vehicle2.race()
12  }
13  class Vehicle(speed: Int){
14    val mph: Int = speed
15    def race() = println("Racing")
16  }
17  class Car(speed: Int) extends Vehicle(speed){
18    override val mph: Int = speed
19    override def race() = println("Racing Car")
20  }
21  class Bike(speed: Int) extends Vehicle(speed){
22    override val mph: Int = speed
23  }
24 }
```

Problems Tasks Console

<terminated> Inheritance\$ [Scala Application] /usr/java/jdk1.8.0_151/bin/java (May 31, 2018, 4:46:06 PM)

```
150
Racing Car
100
Racing Bike
```

Task 2:

Write a simple program to show multiple inheritance in scala

Scala doesn't allow for multiple inheritance per se, but allows us to extend multiple traits.

Traits are used to share interfaces and fields between classes. They are similar to Java 8's interfaces. Classes and objects can extend traits but traits cannot be instantiated and therefore have no parameters.

Scala resolves the diamond problem by defining one main super trait, whose code will be used, among all super traits. The main one is set with the 'extends' keyword, while the others are set with 'with'.

```
package acadgild.Task2
```

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

```

trait Mammal {
  var mammalName: String = _
  def action = {
    println(mammalName + " is a mammal")
  }
}
trait WingedAnimal {
  var animalName: String = _
  def action = {
    println(animalName + " is a winged animal")
  }
}
class Bat extends Mammal with WingedAnimal {
  mammalName = "Bat";
  animalName = "Bat";
  override def action = {
    super[Mammal].action
    super[WingedAnimal].action
  }
}
var ab = new Bat
ab.action
}

```

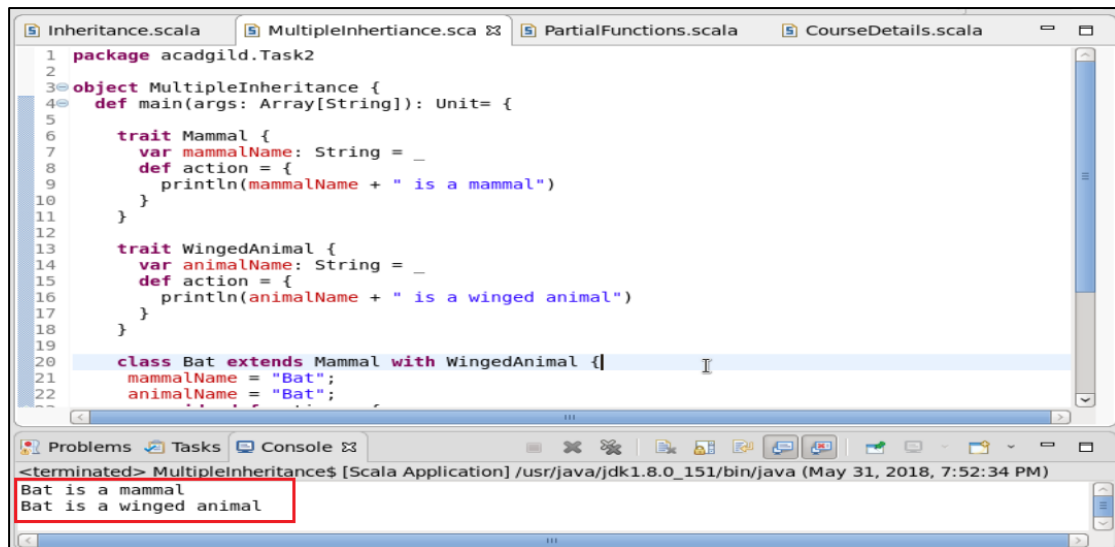
A class Bat is derived from traits Mammal and WingedAnimal. It makes sense because bat is a mammal as well as a winged animal.

Since the class 'Mammal' is extended, it will be the main super trait. And the 'WingedAnimal' will be the alternate trait.

Output :

Bat is a mammal

Bat is a winged animal



```
1 package acadgild.Task2
2
3 object MultipleInheritance {
4   def main(args: Array[String]): Unit = {
5
6     trait Mammal {
7       var mammalName: String = _
8       def action = {
9         println(mammalName + " is a mammal")
10      }
11    }
12
13    trait WingedAnimal {
14      var animalName: String = _
15      def action = {
16        println(animalName + " is a winged animal")
17      }
18    }
19
20    class Bat extends Mammal with WingedAnimal {
21      mammalName = "Bat";
22      animalName = "Bat";
23    }
24  }
25}
```

Problems Tasks Console

<terminated> MultipleInheritance\$ [Scala Application] /usr/java/jdk1.8.0_151/bin/java (May 31, 2018, 7:52:34 PM)

```
Bat is a mammal
Bat is a winged animal
```

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.

```
package acadgild.Task3

object PartialFunctions {
  def main(args: Array[String]){
    val inputNumber1 = 1
    val inputNumber2 = 2
    val result = getSquare(inputNumber1, inputNumber2)
    println("Square of given numbers : "+ result)
  }

  val addNumbers: PartialFunction[(Int, Int), Int] = {
    case (x, y) => x + y + 2
  }

  def getSquare(num1: Int, num2: Int): Int = {
    val res = addNumbers(num1, num2)
    res * res
  }
}
```

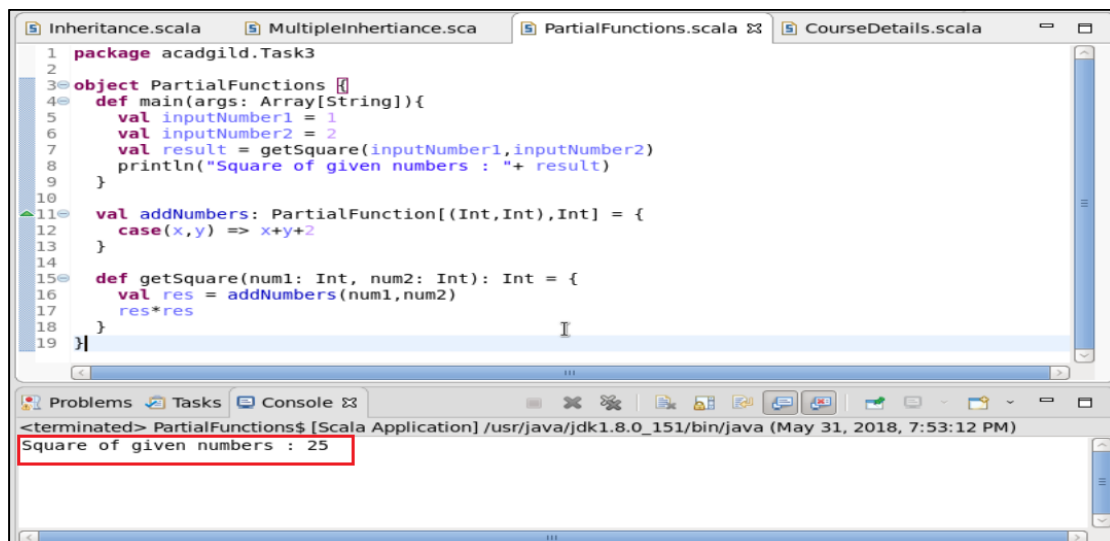
Partial functions in scala :

A partial function is a function that does not provide an answer for every possible input value it can be given. It provides an answer only for a subset of

possible data, and defines the data it can handle. In Scala, a partial function can also be queried to determine if it can handle a particular value.

Output :

Square of given numbers : 25



The screenshot shows an IDE with a Scala file named `PartialFunctions.scala`. The code defines a package `acadgild.Task3` and an object `PartialFunctions`. Inside the object, there is a `main` method that takes an array of strings, sets `inputNumber1` to 1 and `inputNumber2` to 2, and calls `getSquare` with these values. The `getSquare` method uses a partial function `addNumbers` to calculate the square of the sum of the two numbers. The output of the program is displayed in the console: "Square of given numbers : 25".

```
1 package acadgild.Task3
2
3 object PartialFunctions {
4   def main(args: Array[String]){
5     val inputNumber1 = 1
6     val inputNumber2 = 2
7     val result = getSquare(inputNumber1, inputNumber2)
8     println("Square of given numbers : " + result)
9   }
10
11   val addNumbers: PartialFunction[(Int, Int), Int] = {
12     case (x, y) => x + y + 2
13   }
14
15   def getSquare(num1: Int, num2: Int): Int = {
16     val res = addNumbers(num1, num2)
17     res * res
18   }
19 }
```

Problems Tasks Console

<terminated> PartialFunctions\$ [Scala Application] /usr/java/jdk1.8.0_151/bin/java (May 31, 2018, 7:53:12 PM)

Square of given numbers : 25

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.

```
package acadgild.Task4
```

```
object CourseDetails {
  def main(args: Array[String]){
    println(matchTest("Android App Development"))
    println(matchTest("Data Science"))
    println(matchTest("Big Date Hadoop & Spark Developer"))
    println(matchTest("BlockChain Certification"))
    println(matchTest("Java Development"))
  }
}
```

```

}
def matchTest(x:Any): Any = x match{
  case "Android App Development" => "14,999 INR"
  case "Data Science" => "49,999 INR"
  case "Big Date Hadoop & Spark Developer" => "24,999 INR"
  case "BlockChain Certification" => "49,999 INR"
  case _ => "Details not found!"
}
}

```

Output :

14,999 INR

49,999 INR

24,999 INR

49,999 INR

Details not found!

The default case is implied by case `_`, where in the input matches to anything, when none of the above cases match.

The screenshot shows an IDE with a Scala file named `CourseDetails.scala`. The code defines a `matchTest` function using pattern matching. The `main` function calls `matchTest` with four different inputs: "Android App Development", "Data Science", "Big Date Hadoop & Spark Developer", and "BlockChain Certification". The output in the console window is as follows:

```

<terminated> CourseDetails$ [Scala Application] /usr/java/jdk1.8.0_151/bin/java (May 31, 2018, 7:53:50 PM)
14,999 INR
49,999 INR
24,999 INR
49,999 INR
Details not found!

```