Task 1

A Fibonacci series (starting from 1) written in order without any spaces in between, thus producing a sequence of digits.

A: Write a Scala application to find the Nth digit in the sequence. Write the function using standard for loop

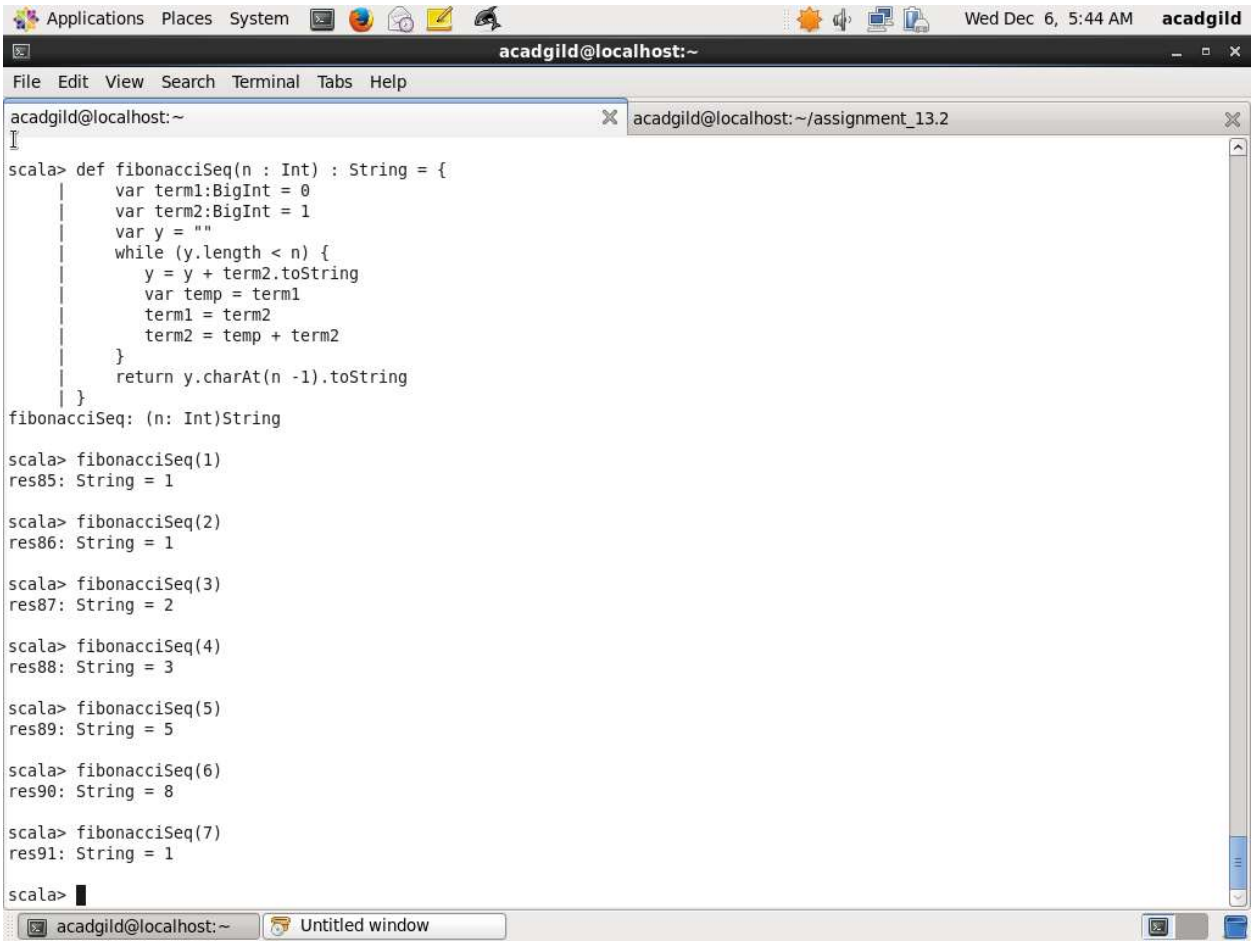
Solution:

Here input is n which is the nth digit. Initialize a string variable y with blank. In a for loop calculate the terms of Fibonacci seq and append to y. When nth digit Is reached, he digit is printed. The code is as below:

Code is as below:

```
def fibonacciSeq(n : Int) : String = {
   var term1:BigInt = 0
   var term2:BigInt = 1
   var y = ""
   while (y.length < n) {
     y = y + term2.toString
     var temp = term1
     term1 = term2
     term2 = temp + term2
   }
   return y.charAt(n -1).toString
}
```

Screenshot is as below. Please note the 7^{th} digit is 1 this is because 7^{th} term is 13



B: Write a Scala application to find the Nth digit in the sequence. Write the function using using recursion

Solution:

Here I use the tail recusion. In the inner method of tail recursion, I use parameters n , original_n, term1 initialized to 0, term2 initialized to 1 and string x which is initialized to blank string. A variable y is used, which is concatenation of x and term2. Each recursive loop, term1 and term2 are modified. The terminating condirion is when nth digit is reached, that time, nth term in x is returned.

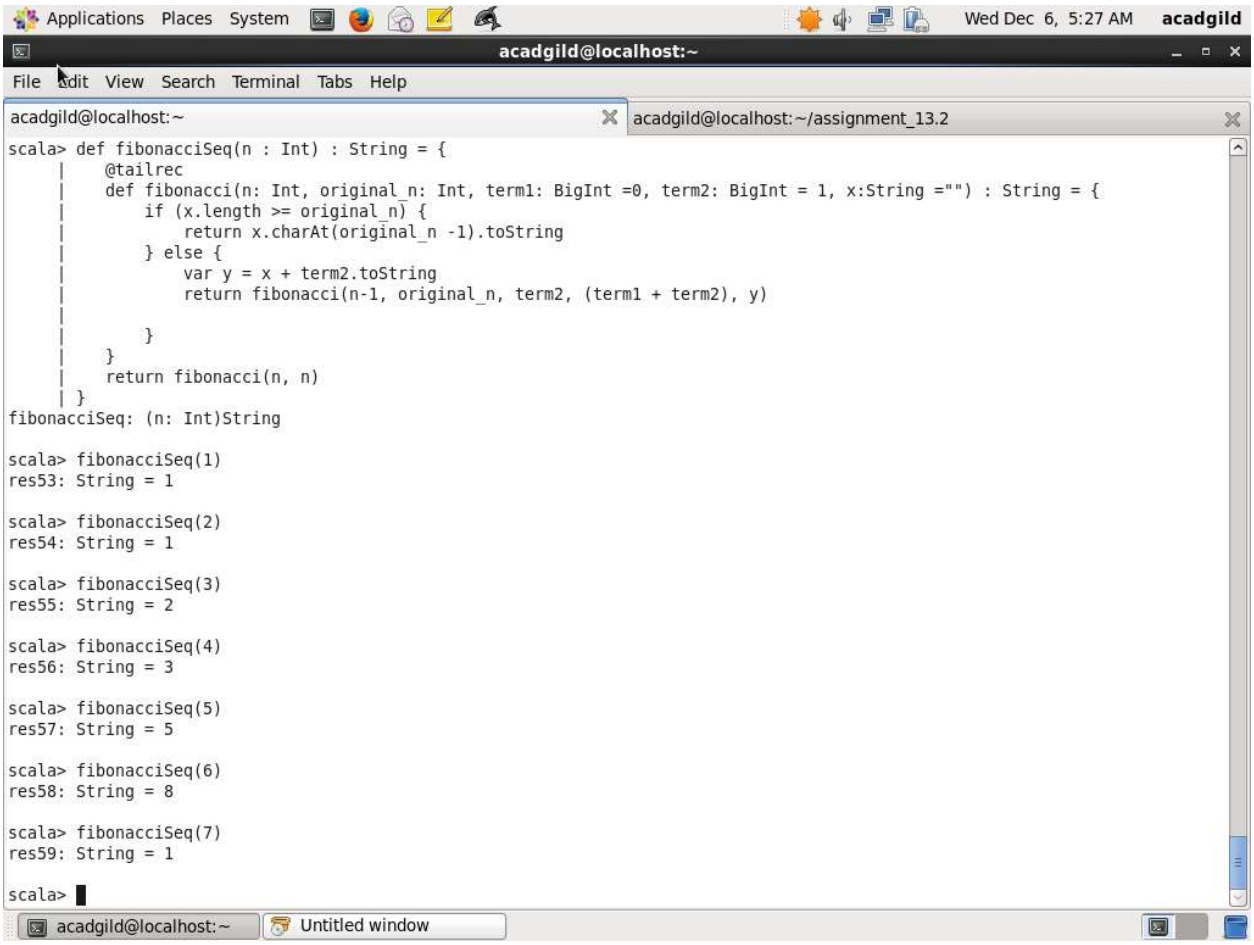
Scala code is as below:

```
import scala.annotation.tailrec
```

```
def fibonacciSeq(n : Int) : String = {
  @tailrec
  def fibonacci(n: Int, original_n: Int, term1: BigInt =0, term2: BigInt = 1, x:String ="") : String = {
    if (x.length >= original_n) {
      return x.charAt(original_n -1).toString
    } else {
      var y = x + term2.toString
      return fibonacci(n-1, original_n, term2, (term1 + term2), y)

    }
  }
  return fibonacci(n, n)
}
```

Screenshot is as below. Please note the 7th digit is 1 this is because 7th term is 13



Task 2:

Create a calculator to work with rational numbers.

Requirements:

○ It should provide capability to add, subtract, divide and multiply rational numbers

○ Create a method to compute GCD (this will come in handy during operations on rational)

Add option to work with whole numbers which are also rational numbers i.e. (n/1)

- achieve the above using auxiliary constructors
- enable method overloading to enable each function to work with numbers and rational.

Solution:

I defined a class Rational and two private attributes numerator and denominator which are of type BigInt. I have chosen BigInt so that same program can be used for very big numbers. I defined one auxiliary constructor so that whole number can be used. Defined the following method:

sum: There are two overloaded methods, first one take Rational as a parameter and returns sum as a rational number. Second one takes BigInt as parameter and returns sum as a Rational number. Sum is computed by:

(Numerator of Object * Denominator of parameter + Denominator of Object * Numerator of parameter) / (Denominator of Object * Denominator of Parameter)

subtract: There are two overloaded methods, first one take Rational as a parameter and returns subtracts from the object and return as a rational number. Second one takes BigInt as parameter and subtracts the number from the object and returns as a Rational number . Subtract is computed by:

(Numerator of Object * Denominator of parameter - Denominator of Object * Numerator of parameter) / (Denominator of Object * Denominator of Parameter)

multiply: There are two overloaded methods, first one take Rational as a parameter and returns multiplies with the object and return as a rational number. Second one takes BigInt as parameter and multiply with the number from the object and returns as a Rational number. Muliply is computed by:

(Numerator of Object * Numerator of Parameter) / (Denominator of Object * Denominator of Parameter)

divide: There are two overloaded methods, first one take Rational as a parameter and divides from the object and return as a rational number. Second one takes BigInt as parameter and divides from the object and returns as a Rational number :

(Numerator of Object * Denominator of Parameter) / (Denominator of Object * Numerator of Parameter)

gcd: There are two overloaded methods, first one take Rational as a parameter and finds gcd with the object and return as a rational number. Second one takes BigInt as parameter and finds gcd with the object and returns as a Rational number. Gcd of Rational Number is calculated by:

GCD of numerator of Object and Parameter/ LCM of denominator of Object and Parameter

LCM is computed using a method compute_lcm which takes two BigInt numbers and computes the LCM, by repeatedly summing till both the numbers are equal

GCD is computed by a method compute_gcd, This method first first finds greater of two numbers and assign to first_number and second_number respectively. Remainder is calculated by using modulus operator (%) between first_number and second_number. This steps is done repeated in while loop till remainder is 0. When remainder becomes 0, first_number is returned as the gcd.

I have also defined a method printObject which will print the numerator and denominator value of Rational object

I have written one singleton Object RationalMain having main method. Here I have taken two sets of Rational Number. In one set I have taken wo Rational Number 15/12 and 6/8 and calculated sum, subtract, multiply, divide, gcd

In another set I have taken two whole Numbers 15 and 6 and computed sum, subtract, multiply, divide, gcd

I have created a file RationalMain.scala and its content is as below:

```
class Rational (x:BigInt, y:BigInt) {
   private val numerator:BigInt = x
   private val denominator:BigInt = y
   def this(a:BigInt) = this(a, 1)
   def sum(b: Rational):Rational = {
      return new Rational(numerator * b.denominator +  denominator * b.numerator, denominator * b.denominator)
   }
   def sum(b: BigInt):Rational = {
      return new Rational(numerator +  b, 1)
   }
   def subtract(b: Rational):Rational = {
      return new Rational(numerator * b.denominator -  denominator * b.numerator, denominator * b.denominator)
   }
```

```
def subtract(b: BigInt):Rational = {
   return new Rational(numerator -  b, 1)
}
def multiply(b: Rational):Rational = {
   return new Rational(numerator * b.numerator, denominator * b.denominator)
}
def multiply(b: BigInt):Rational = {
   return new Rational(numerator *  b, 1)
}
def divide(b: Rational):Rational = {
   return new Rational(numerator * b.denominator, denominator * b.numerator)
}
def devide(b: BigInt):Rational = {
   return new Rational(numerator /  b, 1)
}
def compute_lcm(m: BigInt, n:BigInt):BigInt = {
   var a = m
   var b = n
   while ( a != b) {
      if (a<b )  a = a + m
      else b = b + n
   }
   return a
}

def compute_gcd(a:BigInt, b:BigInt) : BigInt = {
   var first_number:BigInt = 0
   var second_number:BigInt = 0
   if (a>b) {
```

```
        first_number = a
        second_number = b
      } else {
        first_number = b
        second_number = a
      }
      var remainder:BigInt = 1
      while (remainder != 0) {
        remainder = first_number % second_number
        first_number = second_number
        second_number = remainder
      }
      return first_number
    }

    def gcd(b:Rational) : Rational = {
      val x:BigInt = compute_gcd(numerator, b.numerator)
      val y:BigInt = compute_lcm(denominator, b.denominator)
      return new Rational(x, y)
    }

    def gcd(b:BigInt) : Rational = {
      val z:Rational = new Rational(b, 1)
      return gcd(y)
    }

    def printObject = println("numerator =" + numerator + " denominator=" + denominator)
}
```

```
object RationalMain {
  def main(args: Array[String]):Unit = {
     val r1 = new Rational(15,12)
     println("For RationaL Number r1")
     r1.printObject
     val r2 = new Rational(6,8)
     println("For RationaL Number r2")
     r2.printObject
     val sum_r1_r2 = r1.sum(r2)
     println(" For r1 + r2: ")
     sum_r1_r2.printObject
     val subtract_r1_r2 = r1.subtract(r2)
     println(" For r1 - r2: ")
     sum_r1_r2.printObject
     subtract_r1_r2.printObject
     val multiply_r1_r2 = r1.multiply(r2)
     println(" For r1 * r2: ")
     multiply_r1_r2.printObject
     val divide_r1_r2 = r1.divide(r2)
     println( " For r1 / r2: ")
     divide_r1_r2.printObject
     val gcd_r1_r2 = r1.gcd(r2)
     println (" gcd r1 and r2: ")
     gcd_r1_r2.printObject
     val r3 = new Rational(15)
     println("For RationaL Number r3")
     r3.printObject
     val r4 = new Rational(6)
```

```
    println("For RationaL Number r4")
    r4.printObject
    val sum_r3_r4 = r3.sum(r4)
    println(" For r3 + r4: ")
    sum_r3_r4.printObject
    val subtract_r3_r4 = r3.subtract(r4)
    println(" For r3 - r4: ")
    sum_r3_r4.printObject
    subtract_r3_r4.printObject
    val multiply_r3_r4 = r3.multiply(r4)
    println(" For r3 * r4: ")
    multiply_r3_r4.printObject
    val divide_r3_r4 = r3.divide(r4)
    println(" For r3 / r4: ")
    divide_r3_r4.printObject
    val gcd_r3_r4 = r3.gcd(r4)
    println(" gcd r3 and r4: ")
    gcd_r3_r4.printObject
  }
}
```

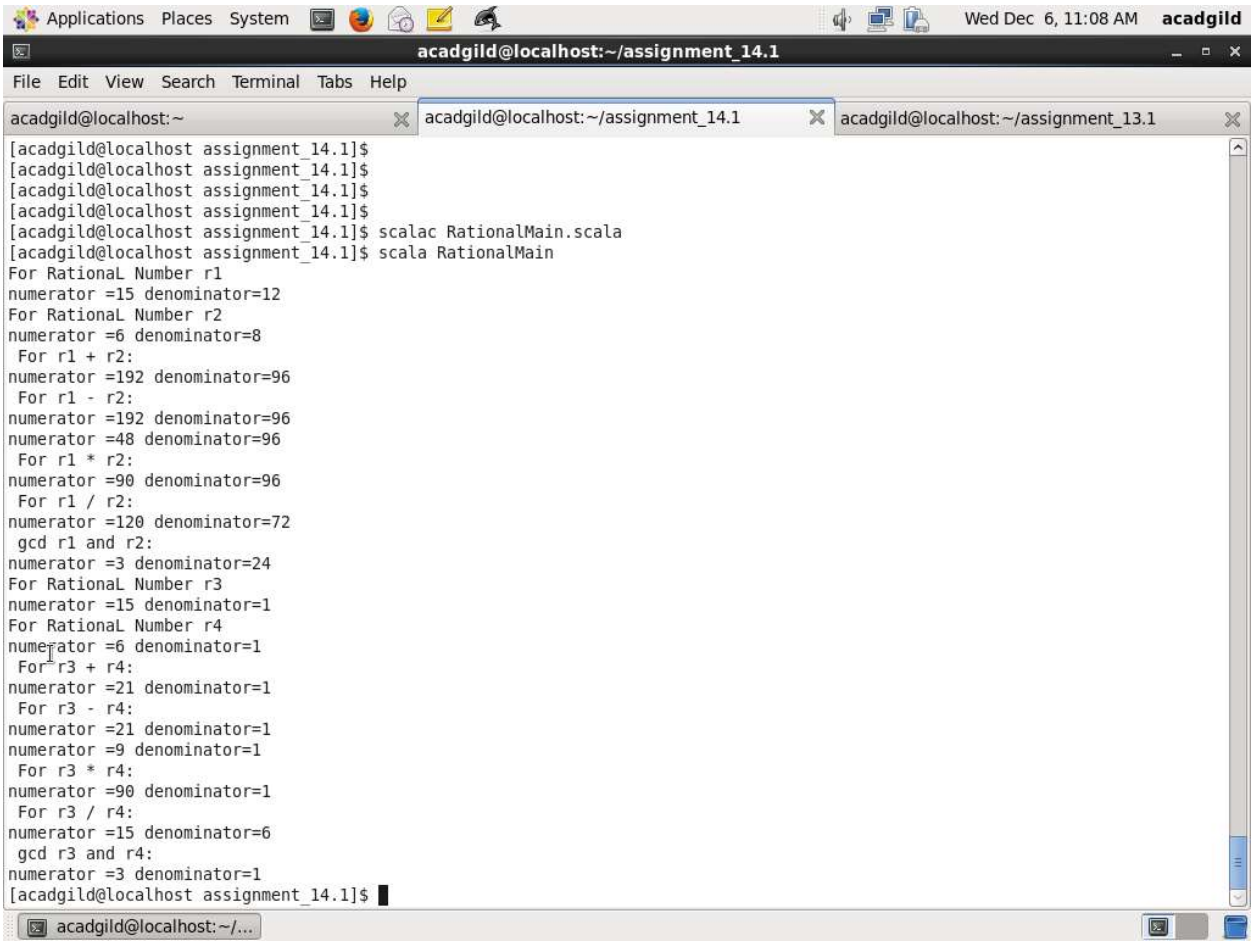
I have compiled the code using

scalac RationalMain.scala

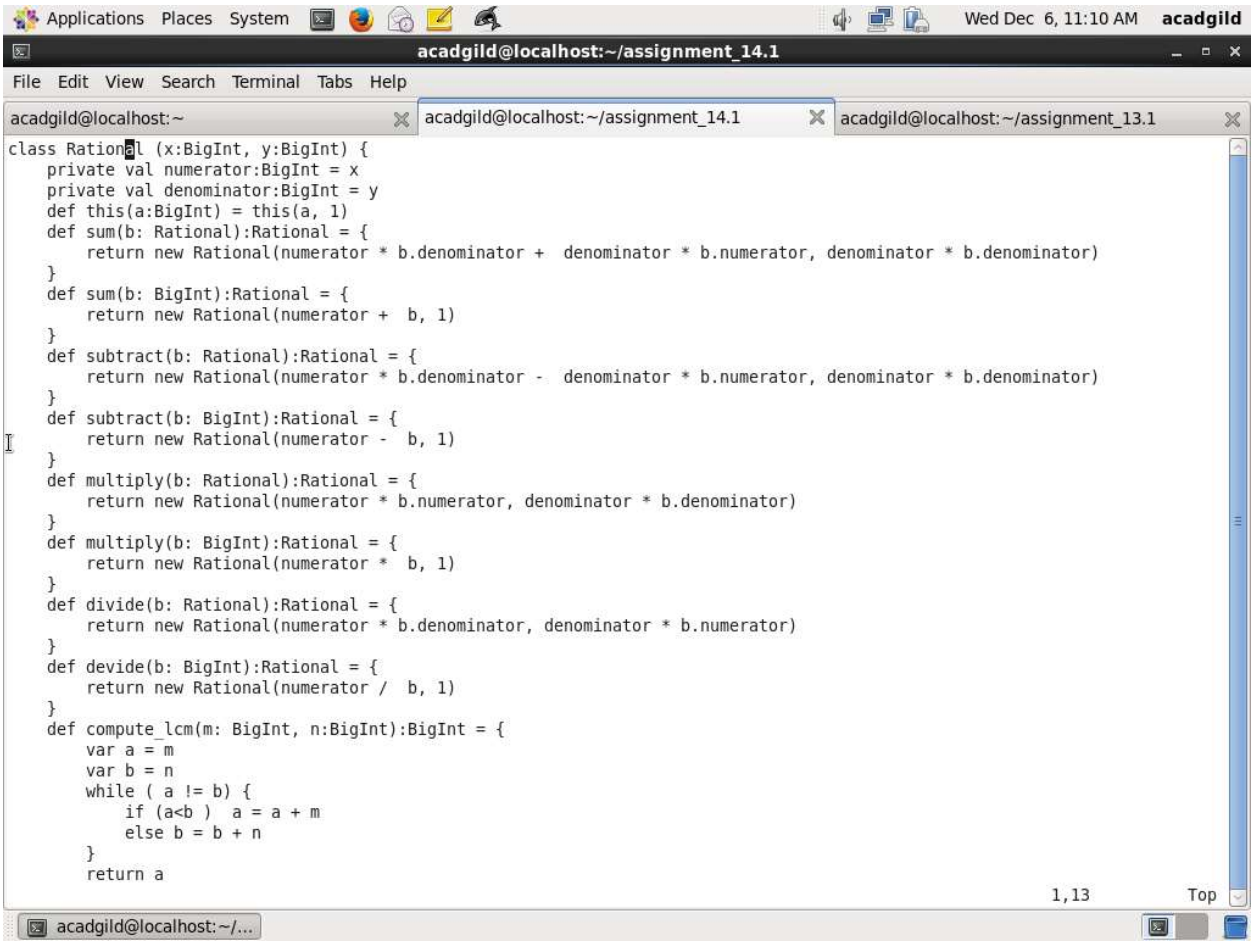
And Run using:

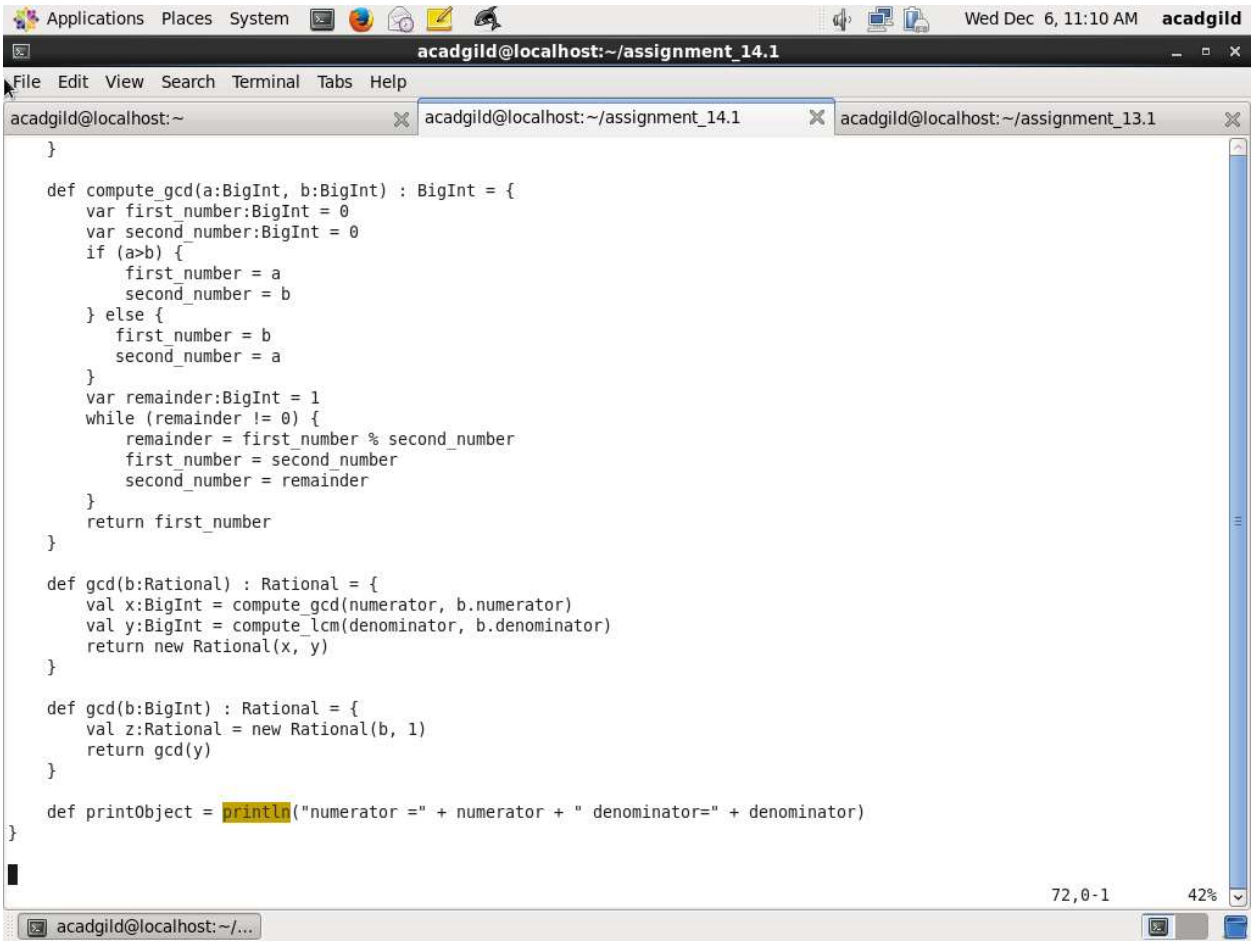
scala RationalMain

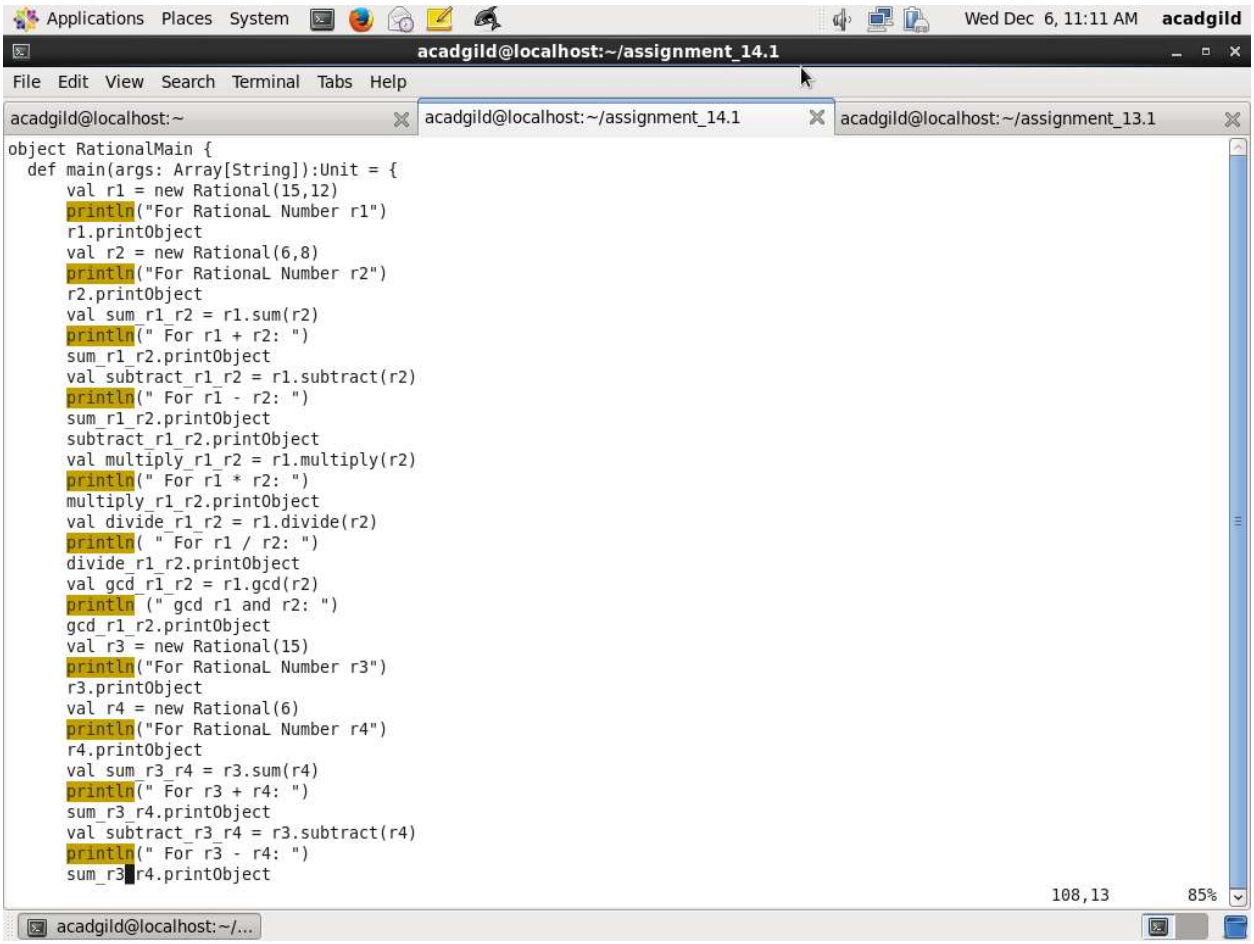
The screenshot of compilation and output is as below:

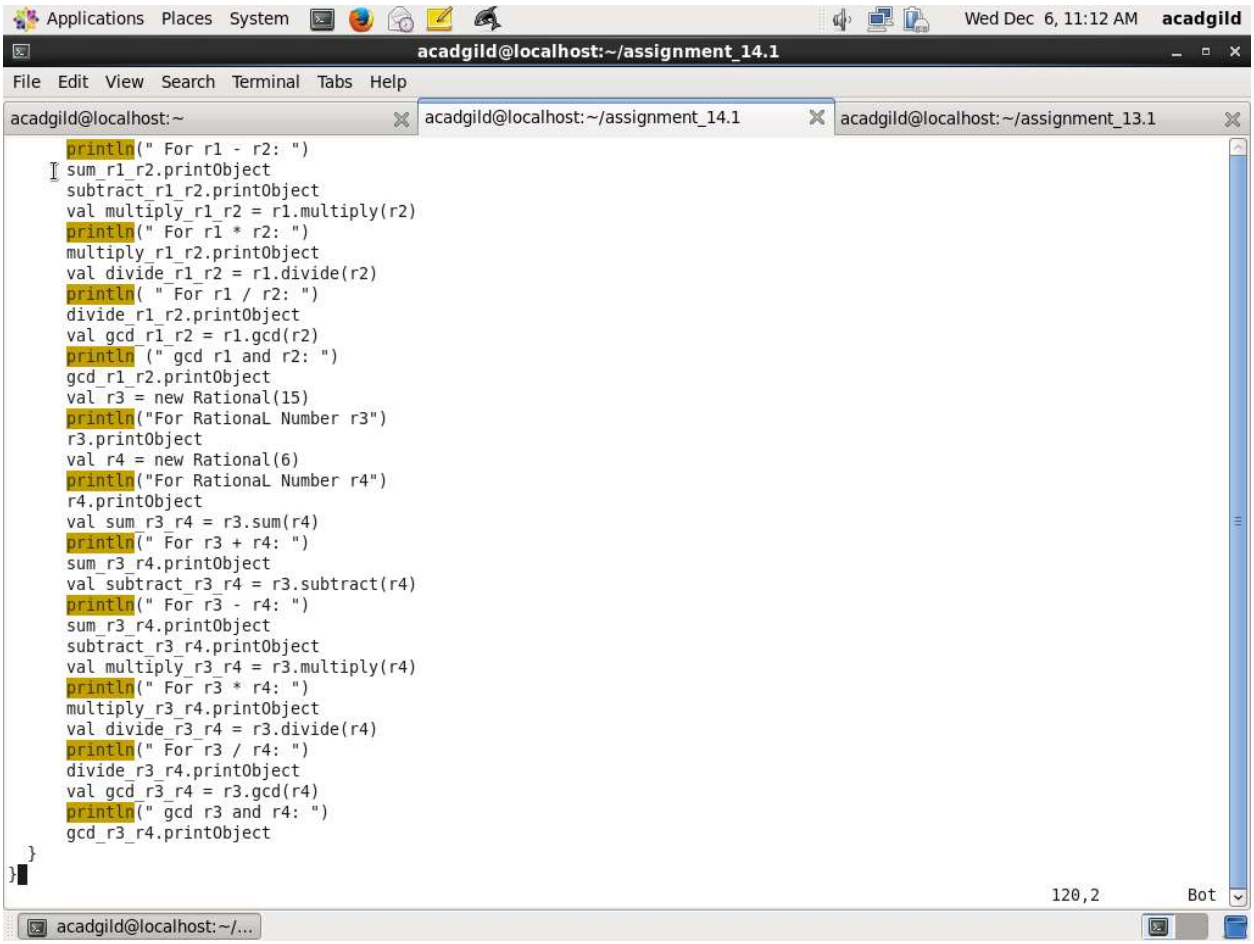


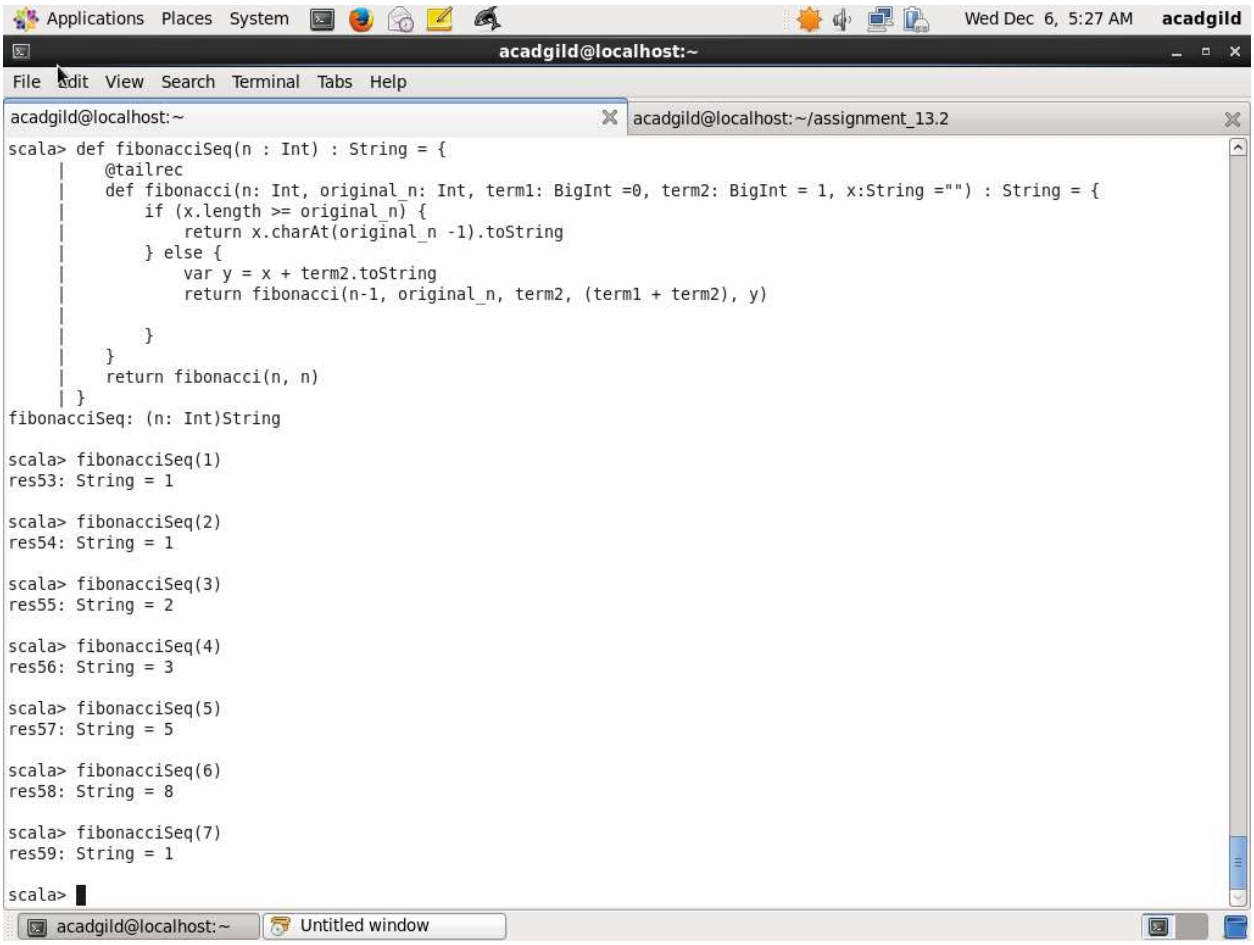
Screenshots of Source Code is as below:


Applications Places System
Wed Dec 6, 11:10 AM acadgild
acadgild@localhost:~/assignment_14.1
File Edit View Search Terminal Tabs Help
acadgild@localhost:~
acadgild@localhost:~/assignment_14.1
acadgild@localhost:~/assignment_13.1
class Rational (x:BigInt, y:BigInt) {
private val numerator:BigInt = x
private val denominator:BigInt = y
def this(a:BigInt) = this(a, 1)
def sum(b: Rational):Rational = {
return new Rational(numerator * b.denominator + denominator * b.numerator, denominator * b.denominator)
}
def sum(b: BigInt):Rational = {
return new Rational(numerator + b, 1)
}
def subtract(b: Rational):Rational = {
return new Rational(numerator * b.denominator - denominator * b.numerator, denominator * b.denominator)
}
def subtract(b: BigInt):Rational = {
return new Rational(numerator - b, 1)
}
def multiply(b: Rational):Rational = {
return new Rational(numerator * b.numerator, denominator * b.denominator)
}
def multiply(b: BigInt):Rational = {
return new Rational(numerator * b, 1)
}
def divide(b: Rational):Rational = {
return new Rational(numerator * b.denominator, denominator * b.numerator)
}
def devide(b: BigInt):Rational = {
return new Rational(numerator / b, 1)
}
def compute_lcm(m: BigInt, n:BigInt):BigInt = {
var a = m
var b = n
while (a != b) {
if (a<b) a = a + m
else b = b + n
}
return a
1,13 Top
acadgild@localhost:~/...


Applications Places System
Wed Dec 6, 11:10 AM acadgild
acadgild@localhost:~/assignment_14.1
File Edit View Search Terminal Tabs Help
acadgild@localhost:~
acadgild@localhost:~/assignment_14.1
acadgild@localhost:~/assignment_13.1
}
def compute_gcd(a:BigInt, b:BigInt) : BigInt = {
var first_number:BigInt = 0
var second_number:BigInt = 0
if (a>b) {
first_number = a
second_number = b
} else {
first_number = b
second_number = a
}
var remainder:BigInt = 1
while (remainder != 0) {
remainder = first_number % second_number
first_number = second_number
second_number = remainder
}
return first_number
}
def gcd(b:Rational) : Rational = {
val x:BigInt = compute_gcd(numerator, b.numerator)
val y:BigInt = compute_lcm(denominator, b.denominator)
return new Rational(x, y)
}
def gcd(b:BigInt) : Rational = {
val z:Rational = new Rational(b, 1)
return gcd(y)
}
def printObject = println("numerator =" + numerator + " denominator=" + denominator)
}
72,0-1 42%
acadgild@localhost:~/...


Applications Places System
Wed Dec 6, 11:11 AM acadgild
acadgild@localhost:~/assignment_14.1
File Edit View Search Terminal Tabs Help
acadgild@localhost:~
acadgild@localhost:~/assignment_14.1
acadgild@localhost:~/assignment_13.1
object RationalMain {
def main(args: Array[String]):Unit = {
val r1 = new Rational(15,12)
println("For RationaL Number r1")
r1.printObject
val r2 = new Rational(6,8)
println("For RationaL Number r2")
r2.printObject
val sum_r1_r2 = r1.sum(r2)
println(" For r1 + r2: ")
sum_r1_r2.printObject
val subtract_r1_r2 = r1.subtract(r2)
println(" For r1 - r2: ")
sum_r1_r2.printObject
subtract_r1_r2.printObject
val multiply_r1_r2 = r1.multiply(r2)
println(" For r1 * r2: ")
multiply_r1_r2.printObject
val divide_r1_r2 = r1.divide(r2)
println(" For r1 / r2: ")
divide_r1_r2.printObject
val gcd_r1_r2 = r1.gcd(r2)
println (" gcd r1 and r2: ")
gcd_r1_r2.printObject
val r3 = new Rational(15)
println("For RationaL Number r3")
r3.printObject
val r4 = new Rational(6)
println("For RationaL Number r4")
r4.printObject
val sum_r3_r4 = r3.sum(r4)
println(" For r3 + r4: ")
sum_r3_r4.printObject
val subtract_r3_r4 = r3.subtract(r4)
println(" For r3 - r4: ")
sum_r3r4.printObject
108,13 85%
acadgild@localhost:~/...


Applications Places System
Wed Dec 6, 11:12 AM acadgild
acadgild@localhost:~/assignment_14.1
File Edit View Search Terminal Tabs Help
acadgild@localhost:~
acadgild@localhost:~/assignment_14.1
acadgild@localhost:~/assignment_13.1
println(" For r1 - r2: ")
sum_r1_r2.printObject
subtract_r1_r2.printObject
val multiply_r1_r2 = r1.multiply(r2)
println(" For r1 * r2: ")
multiply_r1_r2.printObject
val divide_r1_r2 = r1.divide(r2)
println(" For r1 / r2: ")
divide_r1_r2.printObject
val gcd_r1_r2 = r1.gcd(r2)
println (" gcd r1 and r2: ")
gcd_r1_r2.printObject
val r3 = new Rational(15)
println("For RationaL Number r3")
r3.printObject
val r4 = new Rational(6)
println("For RationaL Number r4")
r4.printObject
val sum_r3_r4 = r3.sum(r4)
println(" For r3 + r4: ")
sum_r3_r4.printObject
val subtract_r3_r4 = r3.subtract(r4)
println(" For r3 - r4: ")
sum_r3_r4.printObject
subtract_r3_r4.printObject
val multiply_r3_r4 = r3.multiply(r4)
println(" For r3 * r4: ")
multiply_r3_r4.printObject
val divide_r3_r4 = r3.divide(r4)
println(" For r3 / r4: ")
divide_r3_r4.printObject
val gcd_r3_r4 = r3.gcd(r4)
println(" gcd r3 and r4: ")
gcd_r3_r4.printObject
}
}
120,2 Bot
acadgild@localhost:~/...



Task 3

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

Solution:

Step1: I defined a abtract class Shape which has one method printArea.

```
abstract class Shape {
   def printArea()
}
```

Step2: Class Rectangle extends from Shape having attributes length and breadth and overrides the method printArea. Area is calculated using length*breadth

```
class Rectangle(val param1:Float, val param2:Float) extends Shape {
   val length:Float = param1
   val breadth:Float = param2
   override def printArea() = println("Area of Rectangle =" + (length * breadth))
}
```

Step3: Class Triangle extends from Shape having attributes base and height and overrides the method printArea. area is calculated by 0.5* base*height

```
class Triangle(val param1:Float, val param2:Float) extends Shape {
   val base:Float = param1
   val height:Float = param2
   override def printArea() = println("Area of Triangle =" + (0.5 * base * height))
}
```

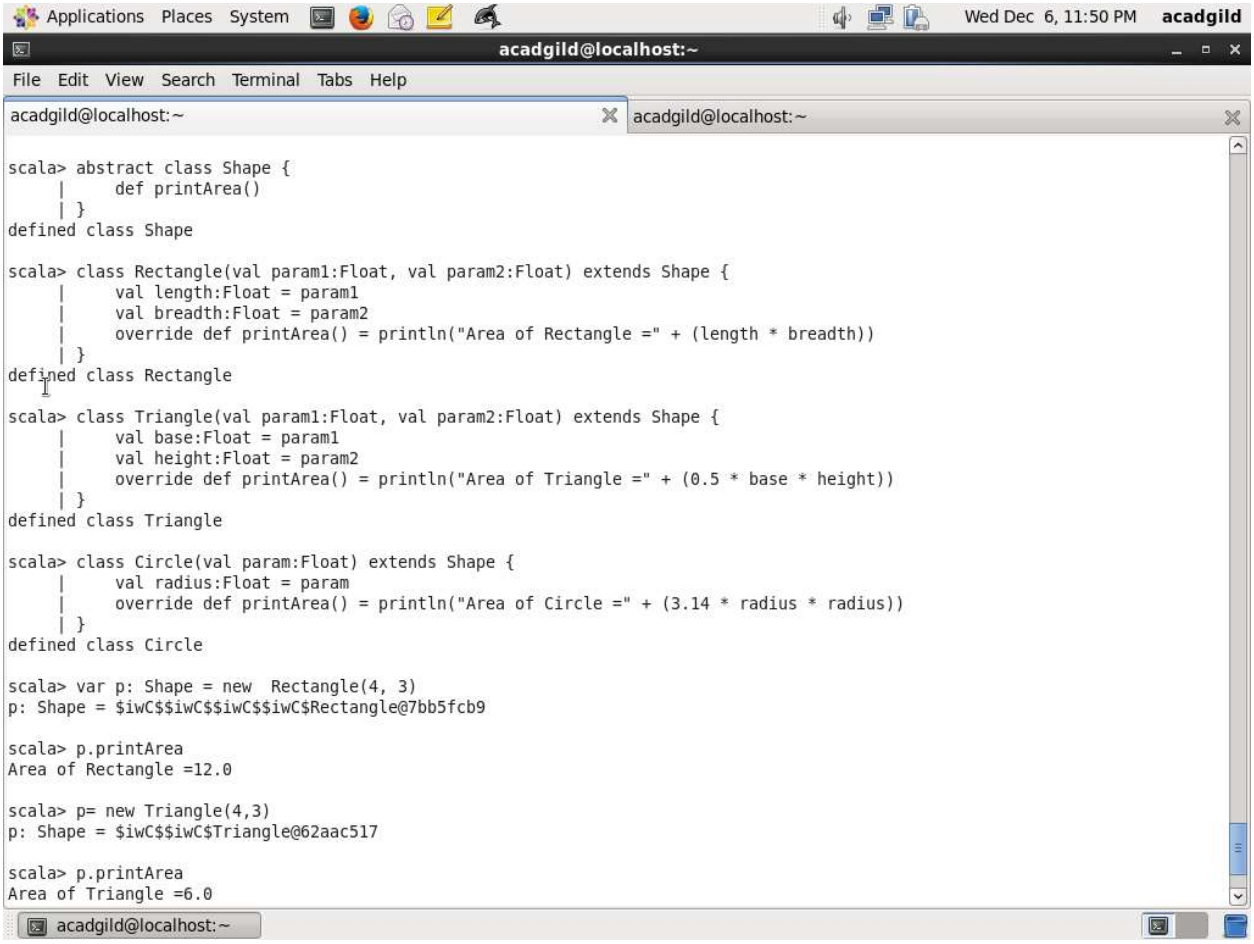
Class Circle extends from Shape having attributes radius and overrides method printArea. area is calculated by 3.14*radius* radius

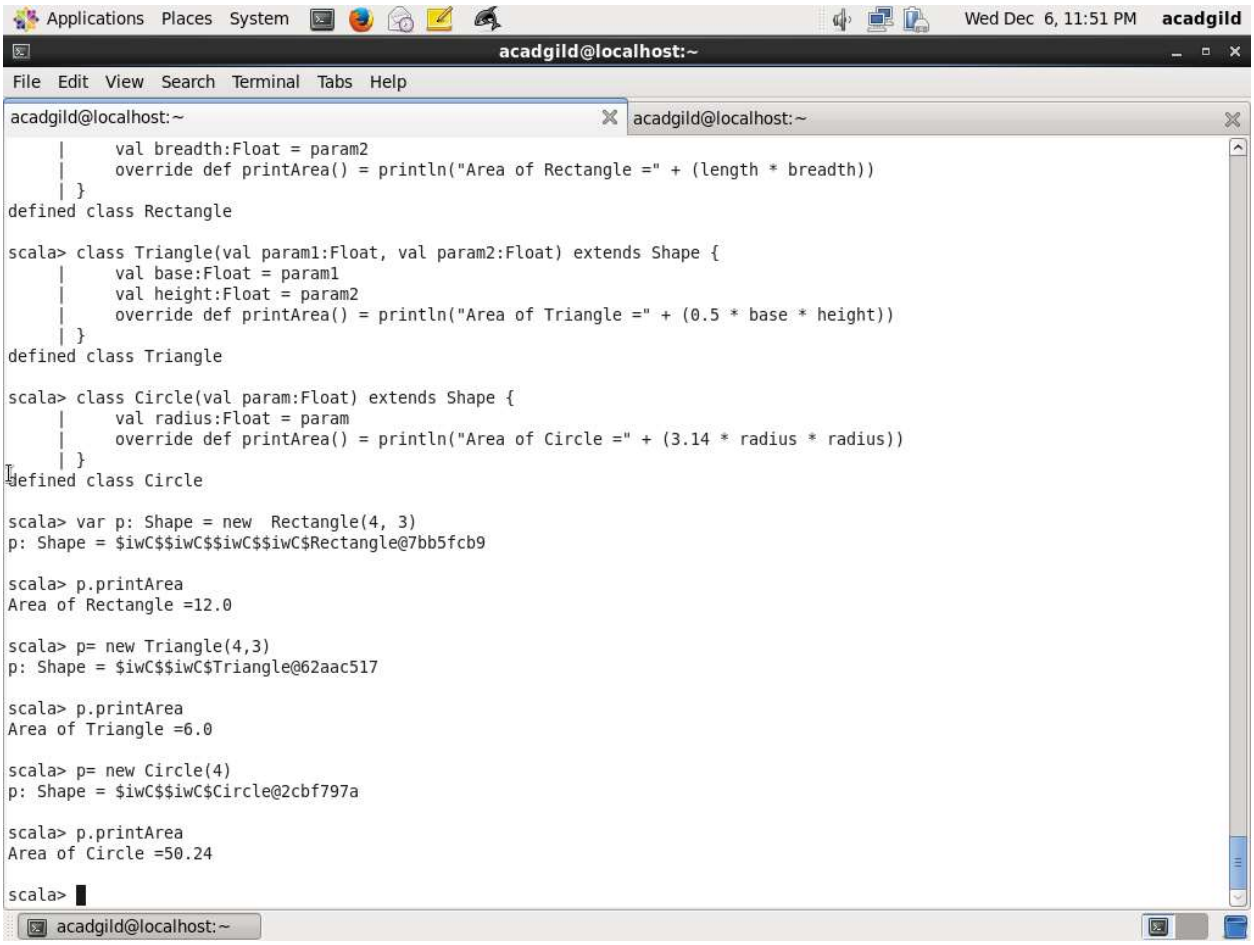
```
class Circle(val param:Float) extends Shape {
   val radius:Float = param
   override def printArea() = println("Area of Circle =" + (3.14 * radius * radius))
}
```

Next, I called created oblect Rectangle, Triangle and Circle and call its printArea method

```
var p: Shape = new  Rectangle(4, 3)
p.printArea
p= new Triangle(4,3)
p.printArea
p= new Circle(4)
p.printArea
```

Screenshot is as below:


Applications Places System
Wed Dec 6, 11:50 PM acadgild
acadgild@localhost:~
File Edit View Search Terminal Tabs Help
acadgild@localhost:~
acadgild@localhost:~
scala> abstract class Shape {
| def printArea()
| }
defined class Shape
scala> class Rectangle(val param1:Float, val param2:Float) extends Shape {
| val length:Float = param1
| val breadth:Float = param2
| override def printArea() = println("Area of Rectangle =" + (length * breadth))
| }
defined class Rectangle
scala> class Triangle(val param1:Float, val param2:Float) extends Shape {
| val base:Float = param1
| val height:Float = param2
| override def printArea() = println("Area of Triangle =" + (0.5 * base * height))
| }
defined class Triangle
scala> class Circle(val param:Float) extends Shape {
| val radius:Float = param
| override def printArea() = println("Area of Circle =" + (3.14 * radius * radius))
| }
defined class Circle
scala> var p: Shape = new Rectangle(4, 3)
p: Shape = $iwC$$iwC$$iwC$Rectangle@7bb5fcb9
scala> p.printArea
Area of Rectangle =12.0
scala> p= new Triangle(4,3)
p: Shape = $iwC$$iwC$Triangle@62aac517
scala> p.printArea
Area of Triangle =6.0
acadgild@localhost:~



2.Write a simple program to show multiple inheritance in scala.

Solution:

Multiple inheritance can be achieved using trait.

Step1: I defined BaseTrait which has a method print.

```
trait BaseTrait {
   def print() { println("Trait: BaseTrait") }
}
```

Step2:

Traits A and B extends from BaseTrait and override method print:

```
trait A extends BaseTrait {
   override def print() { println("Trait: A") }
}
trait B  extends BaseTrait{
   override def print() { println("Trait: B") }
}
```

Step3: Defined a class BaseClass which has method print

```
class BaseClass {
   def print() { println("Class: BaseClass") }
}
```

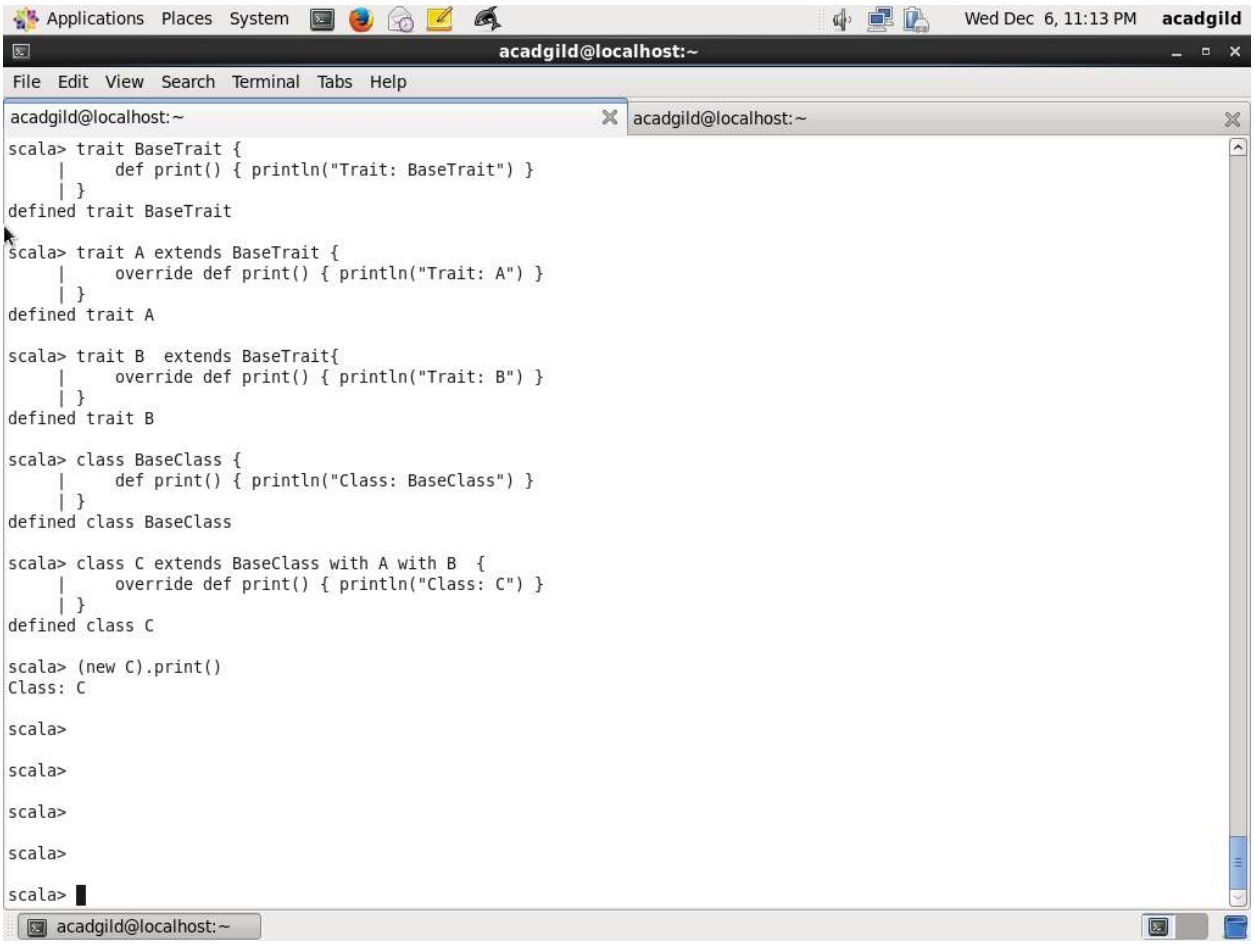
Step4: Defined a class C which extends from class BaseClass and also exends traits A and B and overrides method print:

```
class C extends BaseClass with A with B  {
   override def print() { println("Class: C") }
}
```

Step5; Create a object of C and call print method

```
(new C).print()
```

Screenshot is as below:



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:

Step1: Define method sum which take three integer arguments and sum them

def sum(a:Int, b:Int, c:Int) = a + b + c

Step2: Define partial function modifiedSum which refines sum function by making first argument constant value 5 and takes two arguments

def modifiedSum = sum(5, _:Int, _:Int)

Step2: Define function modifiedSquare which take a method callback as first parameter and two more integer parameters. It calls callback methd two two integer aguments and square the result

```
def modifiedSquare(callback : (Int, Int) => Int, x:Int, y:Int):Int = {
   val z = callback(x,y)
   z * z
}
```

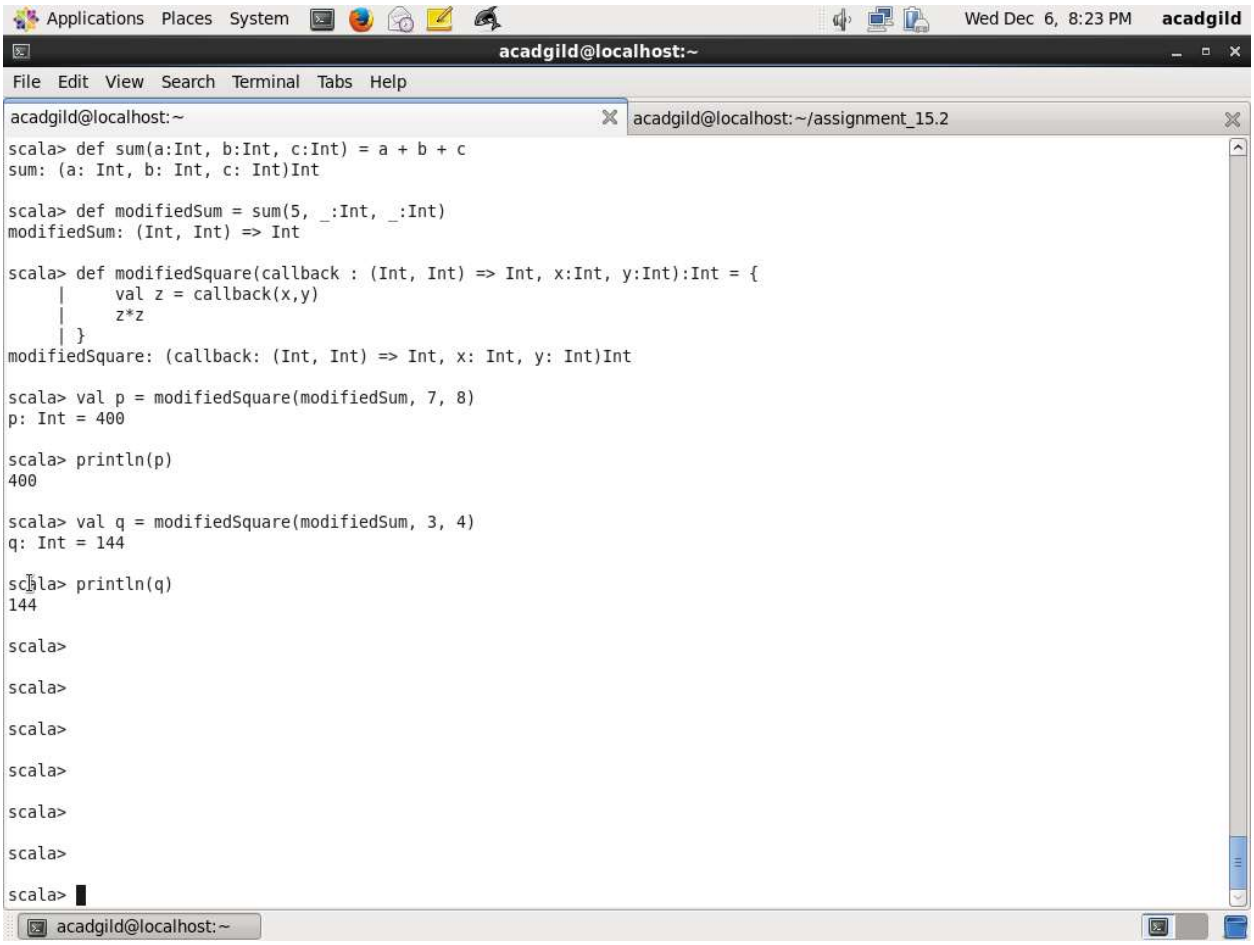
Step3: Call modifiedSqaure with modifiedSum as callback method and arguments 7 and 8

```
val p = modifiedSquare(modifiedSum, 7, 8)
println(p)
```

Step3: Call modifiedSqaure with modifiedSum as callback method and arguments 3 and 4

```
val q = modifiedSquare(modifiedSum, 3, 4)
println(q)
```

Screenshot is as below:



4.Write a program to print the prices of 4 courses of Acadgild: Android-12999,Big Data Development-17999,Big Data Development-17999,Spark-19999 using match and add a default condition if the user enters any other course

Solution:

Note: In the question, there are two same subject Big Data Development gives, if I try I am getting warning. So modified second one as Advanced Big Data Development

Step1: Define function findPrice which will take subject as input and will return price of subject

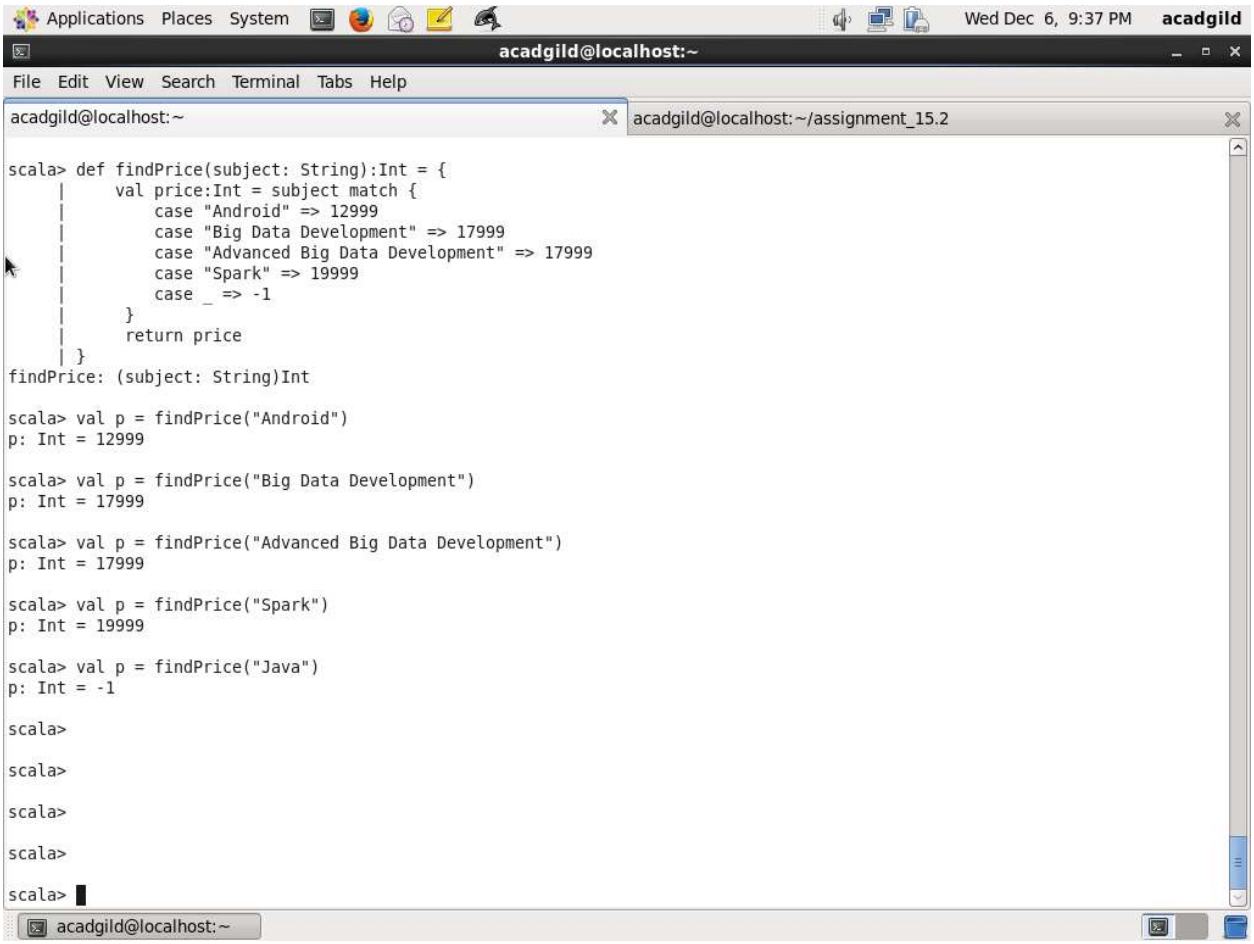
```
def findPrice(subject: String):Int = {
  val price:Int = subject match {
    case "Android" => 12999
```

```
      case "Big Data Development" => 17999
      case "Advanced Big Data Development" => 17999
      case "Spark" => 19999
      case _ => -1
    }
    return price
}
```

Step2: Call the method with various subjects than Android, Big Data Development, Advanced Big Data Development, Spark, Java and return the corresponding price.

```
val p = findPrice("Android")
val p = findPrice("Big Data Development")
val p = findPrice("Advanced Big Data Development")
val p = findPrice("Spark")
val p = findPrice("Java")
```

Screenshot is as below:


Applications Places System
Wed Dec 6, 9:37 PM acadgild
acadgild@localhost:~
File Edit View Search Terminal Tabs Help
acadgild@localhost:~
acadgild@localhost:~/assignment_15.2
scala> def findPrice(subject: String):Int = {
| val price:Int = subject match {
| case "Android" => 12999
| case "Big Data Development" => 17999
| case "Advanced Big Data Development" => 17999
| case "Spark" => 19999
| case _ => -1
| }
| return price
| }
findPrice: (subject: String)Int
scala> val p = findPrice("Android")
p: Int = 12999
scala> val p = findPrice("Big Data Development")
p: Int = 17999
scala> val p = findPrice("Advanced Big Data Development")
p: Int = 17999
scala> val p = findPrice("Spark")
p: Int = 19999
scala> val p = findPrice("Java")
p: Int = -1
scala>
scala>
scala>
scala>
scala>
acadgild@localhost:~