



SCALA - SESSION III

Assignment

StudentName: Subham Vishal

Course: Big Data Hadoop & Spark Training

Assignment

Create a calculator to work with rational numbers using Scala.

Contents

Introduction	1
Problem Statement	1
Task 1 – Create a Scala Class “ <i>Calc</i> ”	2
Task 2 – Create a Scala Object “ <i>CalObj</i> ”	3
Expected Output	4

Introduction

In this assignment, we are going to write a SCALA code to create a Calculator to work with rational numbers,

Problem Statement

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.



Task 1 - Create a Scala Class “Calc”

Scala Code

```
class Calc (n:Int, d:Int)
{
  require(d!=0)
  private val g = gcd(n.abs,d.abs)
  val num = n/g
  val den = d/g

  private def gcd(x:Int, y:Int) :Int =
  {if(x==0) y else if (x<0) gcd(-x,y) else if (y<0) gcd(x,-y) else gcd(y%x,x)}

  def this(n: Int) = this(n, 1) // auxiliary constructor

  def add (r:Calc): Calc = new Calc(num*r.den + r.num*den , den*r.den)
  def add (i:Int): Calc = new Calc(num + i * den, den) //method overloading for add

  def subtract (r:Calc): Calc = new Calc(num*r.den - r.num*den,den*r.den)
  def subtract (i:Int): Calc = new Calc(num - i * den, den)//method overloading for
subtract

  def multiply (r:Calc): Calc = new Calc(num*r.num,den*r.den)
  def multiply (i:Int): Calc = new Calc(num * i , den)//method overloading for
multiplication

  def divide (r:Calc): Calc = new Calc(num*r.den,den*r.num)
  def divide (i: Int): Calc = new Calc(num , den * i)//method overloading for division

  override def toString: String = num+ "/" + den
}
```

The statement, “**def this(n: Int) = this(n, 1)**” is an auxiliary constructor, we have created an Object “CalcObj” to perform the above functions.

We have Enabled method **overloading** to enable each function (add, sub, multiplication and division) to work with numbers and rational.

We have written the code in such a way that it works with whole numbers as well as with rational numbers (n/1).



IntelliJ console,

```

1  class Calc (n:Int, d:Int)
2  {
3      require(d!=0)
4      private val g = gcd(n.abs,d.abs)
5      val num = n/g
6      val den = d/g
7
8      private def gcd(x:Int, y:Int) :Int =
9      {if(x==0) y else if (x<0) gcd(-x,y) else if (y<0) gcd(x,-y) else gcd(y%x,x)}
10
11     def this(n: Int) = this(n, 1)
12
13     def add (r:Calc): Calc = new Calc(num*r.den + r.num*den , den*r.den)
14     def add (i:Int): Calc = new Calc(num + i * den, den)
15
16     def subtract (r:Calc): Calc = new Calc(num*r.den - r.num*den,den*r.den)
17     def subtract (i:Int): Calc = new Calc(num - i * den, den)
18
19     def multiply (r:Calc): Calc = new Calc(num*r.num,den*r.den)
20     def multiply (i:Int): Calc = new Calc(num * i , den)
21
22     def divide (r:Calc): Calc = new Calc(num*r.den,den*r.num)
23     def divide (i: Int): Calc = new Calc(num , den * i)
24
25     override def toString: String = num+ "/" + den
26
27

```

Task 2 - Create a Scala Object “CalObj”

```

object CalcObj
{
    def main(args: Array[String]): Unit =
    {
        val a = new Calc(22,25)
        val b = new Calc(19)
        val c = new Calc(33,15)
        val d = new Calc(13)

        val p = a add 5
        println(p)

        val q = b multiply new Calc(13,25)
        println(q)

        val r = c subtract new Calc(14,1)
        println(r)

        val s = d divide 51
        println(s)
    }
}

```



Expected Output

1. Example 1,

The screenshot displays an IDE with a project named 'Assignment_14 [assignment_14]' located at 'D:\Abu\Technical\Had'. The project structure includes a 'src' directory with a 'main' subdirectory containing a 'scala' package and a 'CalcObj' object. The 'CalcObj' object is selected in the 'External Libraries' pane.

```
object CalcObj
{
  def main(args: Array[String]): Unit =
  {
    val a = new Calc(22,25)
    val b = new Calc(19)
    val c = new Calc(33,15)
    val d = new Calc(13)
    val p = a add 5
    println(p)

    val q = b multiply new Calc(13,25)
    println(q)

    val r = c subtract new Calc(14,1)
    println(r)

    val s = d divide 51
    println(s)
  }
}
```

The output of the program is shown in the 'Run' pane, indicating that the process finished with exit code 0. The output consists of four lines of text, each representing a calculation result:

```
"C:\Program Files\Java\jdk1.8.0_144\bin\java" ...
147/25
247/25
-59/5
13/51
Process finished with exit code 0
```



2. Example 2,

The screenshot shows an IDE with a project named "Assignment_14 [assignment_14]" located at "D:\Abu\Technical\Had". The project structure includes a "src" directory with a "main" subdirectory containing a "scala" subdirectory. Inside "scala", there is a "Calc" object and a "CalcObj" object. The "CalcObj" object is selected, and its code is displayed in the editor. The code defines a "main" method that creates four "Calc" objects (a, b, c, d) and performs operations on them: a + 2, b * 5, c - 6, and d / 7. The output of the program is shown in the "Run" console, displaying the results of these operations: 6/1, 40/1, 3/1, and 5/7. The process finished with exit code 0.

```
object CalcObj
{
  def main(args: Array[String]): Unit =
  {
    val a = new Calc(4)
    val b = new Calc(8)
    val c = new Calc(9)
    val d = new Calc(5)

    val p = a add 2
    println(p)

    val q = b multiply new Calc(5)
    println(q)

    val r = c subtract new Calc(6)
    println(r)

    val s = d divide 7
    println(s)
  }
}
```

Run CalcObj

```
"C:\Program Files\Java\jdk1.8.0_144\bin\java" ...
6/1
40/1
3/1
5/7
Process finished with exit code 0
```