Session 16:
SCALA BASICS 3
Assignment 1

- Prachi Mohite

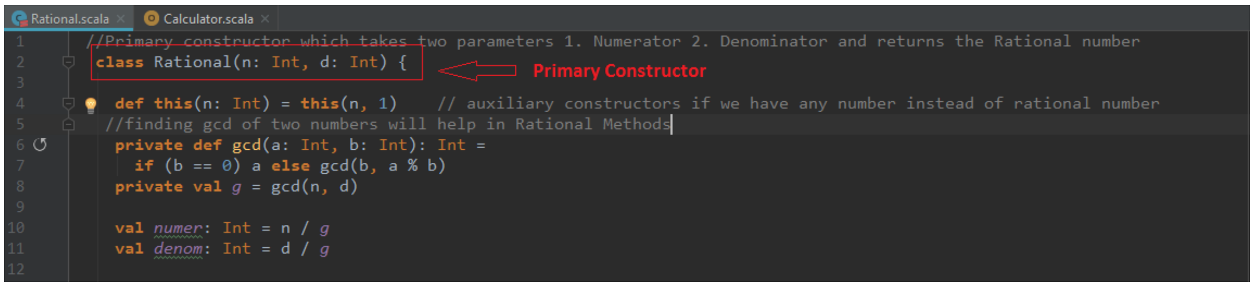
<u>Task 1</u>
Create a calculator to work with rational numbers.
Requirements:
➢ It should provide capability to add, subtract, divide and multiply rational Numbers
➢ Create a method to compute GCD (this will come in handy during operations on rational)
Add option to work with whole numbers which are 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.

Primary Constructor

The primary constructor of a Scala class is a combination of:

- The constructor parameters
- Methods that are called in the body of the class
- Statements and expressions that are executed in the body of the class

In this Example we will be creating the a class named as 'Rational' with Primary Constructor which accepts two values 1. Numerator 2. Denominator. This is part of rational number to be operated on.

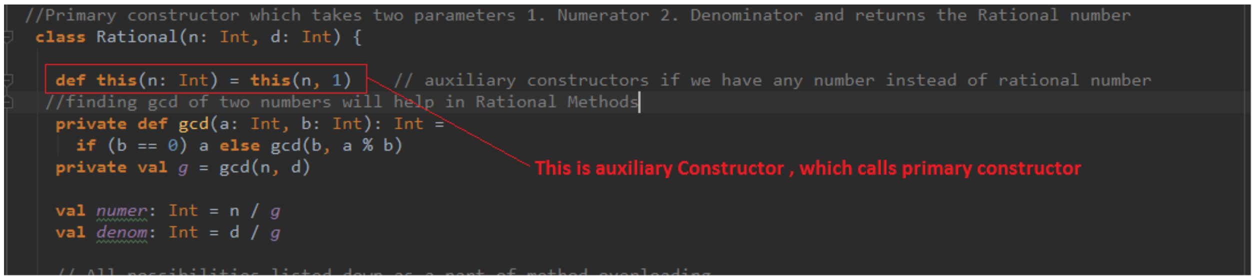


```scala
//Primary constructor which takes two parameters 1. Numerator 2. Denominator and returns the Rational number
class Rational(n: Int, d: Int) {

  def this(n: Int) = this(n, 1)    // auxiliary constructors if we have any number instead of rational number
 //finding gcd of two numbers will help in Rational Methods
  private def gcd(a: Int, b: Int): Int =
    if (b == 0) a else gcd(b, a % b)
  private val g = gcd(n, d)

  val numer: Int = n / g
  val denom: Int = d / g

```

Auxiliary Constructor

- In Scala, We can define Auxiliary Constructors like methods by using "def" and "this" keywords. "this" is the constructor name.
- Auxiliary Constructor is also know as Secondary Constructor. A Scala class can contain zero or one or more Auxiliary Constructors.
- Auxiliary Constructors are used to provide Constructors Overloading.
- Each Auxiliary Constructor should call one of the previous defined constructor. An Auxiliary Constructor can call either Primary Constructor or another Auxiliary constructors by using "this" name.

In this Example we have to create class which should accept Integer number as well Ration Number a well. So to achieve the same we have created the Auxiliary Constructor where it takes only one number (Integer as Parameter) and calls the primary constructor where denum = 1



```
//Primary constructor which takes two parameters 1. Numerator 2. Denominator and returns the Rational number
class Rational(n: Int, d: Int) {

  def this(n: Int) = this(n, 1)   // auxiliary constructors if we have any number instead of rational number
 //finding gcd of two numbers will help in Rational Methods
  private def gcd(a: Int, b: Int): Int =
    if (b == 0) a else gcd(b, a % b)
  private val g = gcd(n, d)

  val numer: Int = n / g
  val denom: Int = d / g
```

This is auxiliary Constructor , which calls primary constructor

Method Overloading

Scala provides method overloading feature which allows us to define methods of same name but having different parameters or data types. It helps to optimize code.

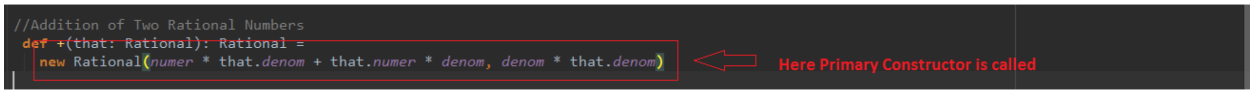
In this example we have write methods for add , Substract, Multiply and divide of

- Two Rational Numbers
- Two Integer Numbers

So we have to write a method with 2 Overloads

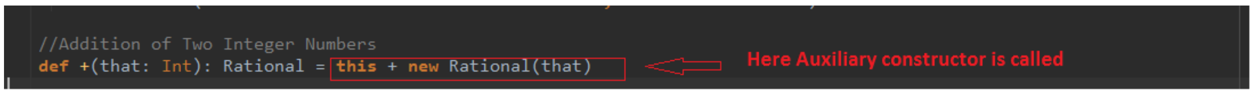
Example of Addition (+)

1. Two Rational Numbers



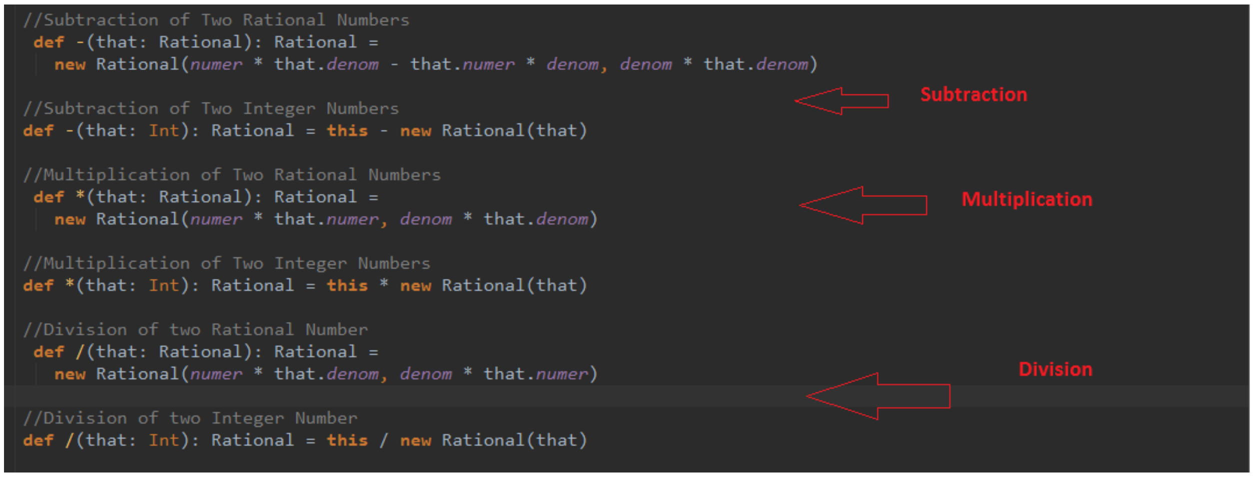
```
//Addition of Two Rational Numbers
 def +(that: Rational): Rational =
  new Rational(numer * that.denom + that.numer * denom, denom * that.denom)
```

- Two Integer Numbers



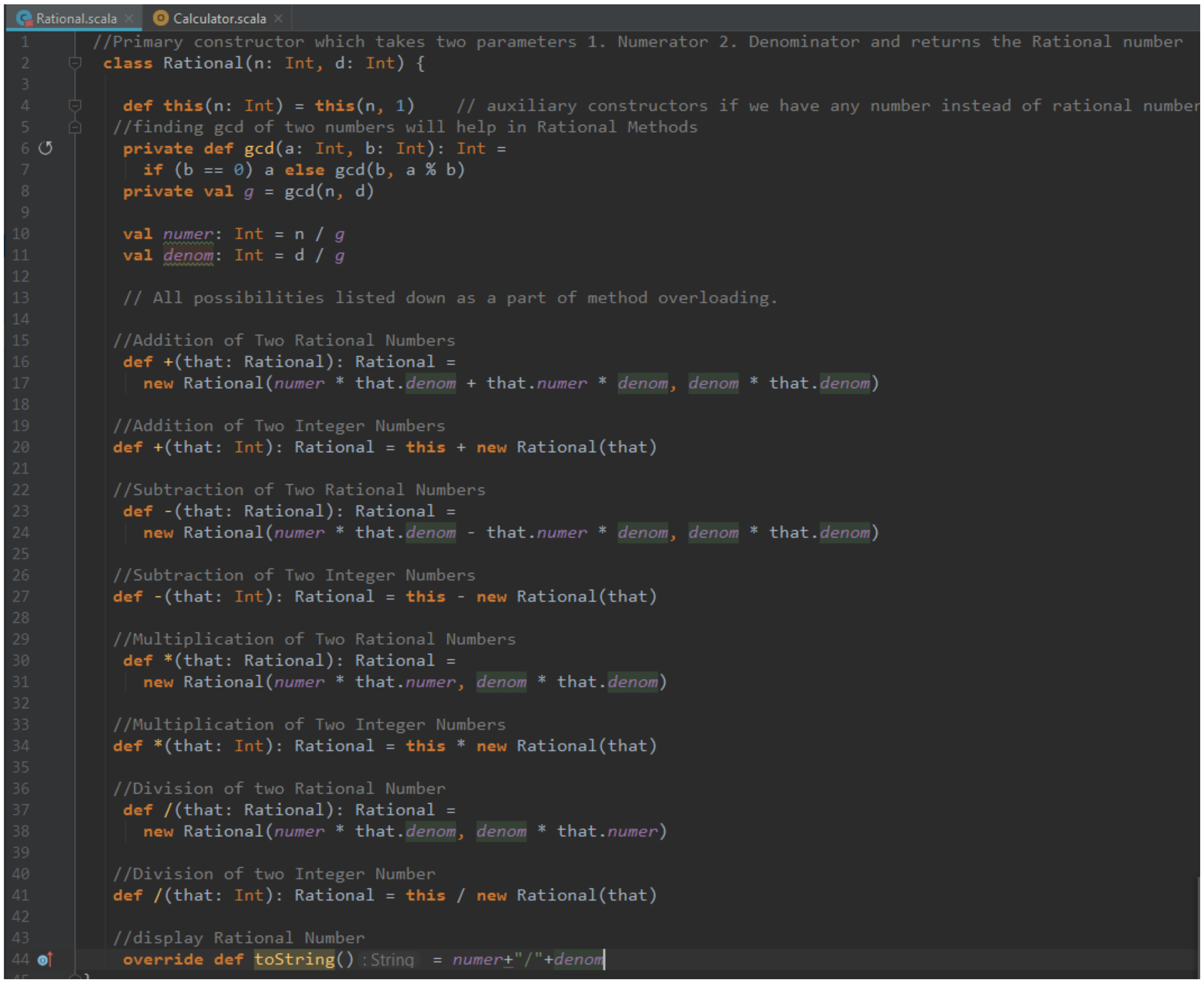
```
  //Addition of Two Integer Numbers
  def +(that: Int): Rational = this + new Rational(that)
```

In same manner methods for subtract(-) , Multiply (*) and Divide(/) are overloaded.



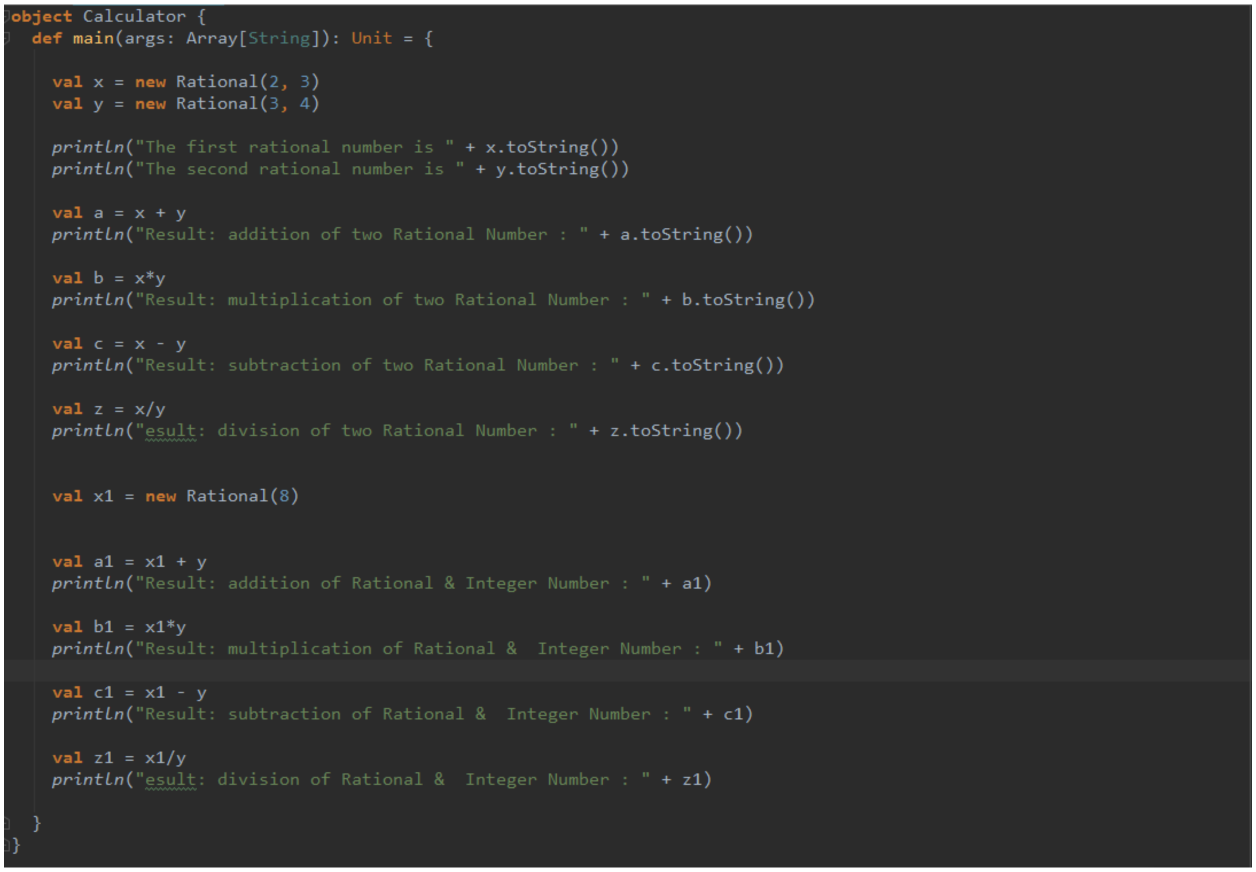
```scala
//Subtraction of Two Rational Numbers
 def -(that: Rational): Rational =
   new Rational(numer * that.denom - that.numer * denom, denom * that.denom)

//Subtraction of Two Integer Numbers
def -(that: Int): Rational = this - new Rational(that)

//Multiplication of Two Rational Numbers
 def *(that: Rational): Rational =
   new Rational(numer * that.numer, denom * that.denom)

//Multiplication of Two Integer Numbers
def *(that: Int): Rational = this * new Rational(that)

//Division of two Rational Number
 def /(that: Rational): Rational =
   new Rational(numer * that.denom, denom * that.numer)

//Division of two Integer Number
def /(that: Int): Rational = this / new Rational(that)
```

Entire Code



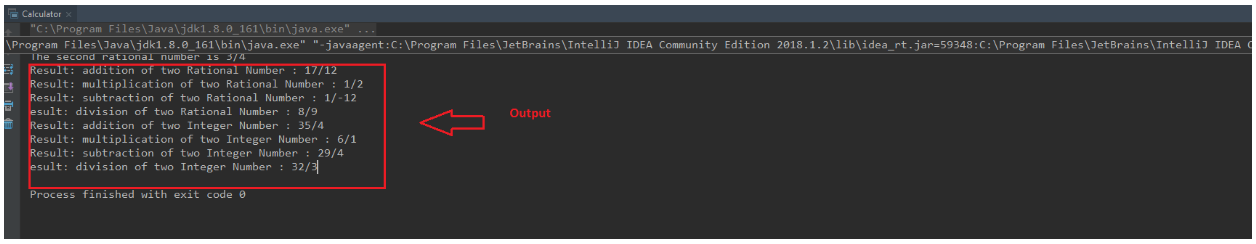
```scala
//Primary constructor which takes two parameters 1. Numerator 2. Denominator and returns the Rational number
 class Rational(n: Int, d: Int) {

   def this(n: Int) = this(n, 1)    // auxiliary constructors if we have any number instead of rational number
  //finding gcd of two numbers will help in Rational Methods
   private def gcd(a: Int, b: Int): Int =
     if (b == 0) a else gcd(b, a % b)
   private val g = gcd(n, d)

   val numer: Int = n / g
   val denom: Int = d / g

   // All possibilities listed down as a part of method overloading.

  //Addition of Two Rational Numbers
   def +(that: Rational): Rational =
     new Rational(numer * that.denom + that.numer * denom, denom * that.denom)

  //Addition of Two Integer Numbers
  def +(that: Int): Rational = this + new Rational(that)

  //Subtraction of Two Rational Numbers
   def -(that: Rational): Rational =
     new Rational(numer * that.denom - that.numer * denom, denom * that.denom)

  //Subtraction of Two Integer Numbers
  def -(that: Int): Rational = this - new Rational(that)

  //Multiplication of Two Rational Numbers
   def *(that: Rational): Rational =
     new Rational(numer * that.numer, denom * that.denom)

  //Multiplication of Two Integer Numbers
  def *(that: Int): Rational = this * new Rational(that)

  //Division of two Rational Number
   def /(that: Rational): Rational =
     new Rational(numer * that.denom, denom * that.numer)

  //Division of two Integer Number
  def /(that: Int): Rational = this / new Rational(that)

  //display Rational Number
   override def toString() : String  = numer+"/"+denom
```

Driver Code to call above Rational / Integer Calculator



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

    val x = new Rational(2, 3)
    val y = new Rational(3, 4)

    println("The first rational number is " + x.toString())
    println("The second rational number is " + y.toString())

    val a = x + y
    println("Result: addition of two Rational Number : " + a.toString())

    val b = x*y
    println("Result: multiplication of two Rational Number : " + b.toString())

    val c = x - y
    println("Result: subtraction of two Rational Number : " + c.toString())

    val z = x/y
    println("esult: division of two Rational Number : " + z.toString())


    val x1 = new Rational(8)


    val a1 = x1 + y
    println("Result: addition of Rational & Integer Number : " + a1)

    val b1 = x1*y
    println("Result: multiplication of Rational &  Integer Number : " + b1)

    val c1 = x1 - y
    println("Result: subtraction of Rational &  Integer Number : " + c1)

    val z1 = x1/y
    println("esult: division of Rational &  Integer Number : " + z1)

  }
}
```

Output of the above Code



```
"C:\Program Files\Java\jdk1.8.0_161\bin\java.exe" ...
\Program Files\Java\jdk1.8.0_161\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2018.1.2\lib\idea_rt.jar=59348:C:\Program Files\JetBrains\IntelliJ IDEA C
The second rational number is 3/4
Result: addition of two Rational Number : 17/12
Result: multiplication of two Rational Number : 1/2
Result: subtraction of two Rational Number : 1/-12
esult: division of two Rational Number : 8/9
Result: addition of two Integer Number : 35/4
Result: multiplication of two Integer Number : 6/1
Result: subtraction of two Integer Number : 29/4
esult: division of two Integer Number : 32/3

Process finished with exit code 0
```