Assignment Day 16

Task 1:

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.

Note: Program files are properly documented for a detailed description of each instruction used within the program.

Ans:

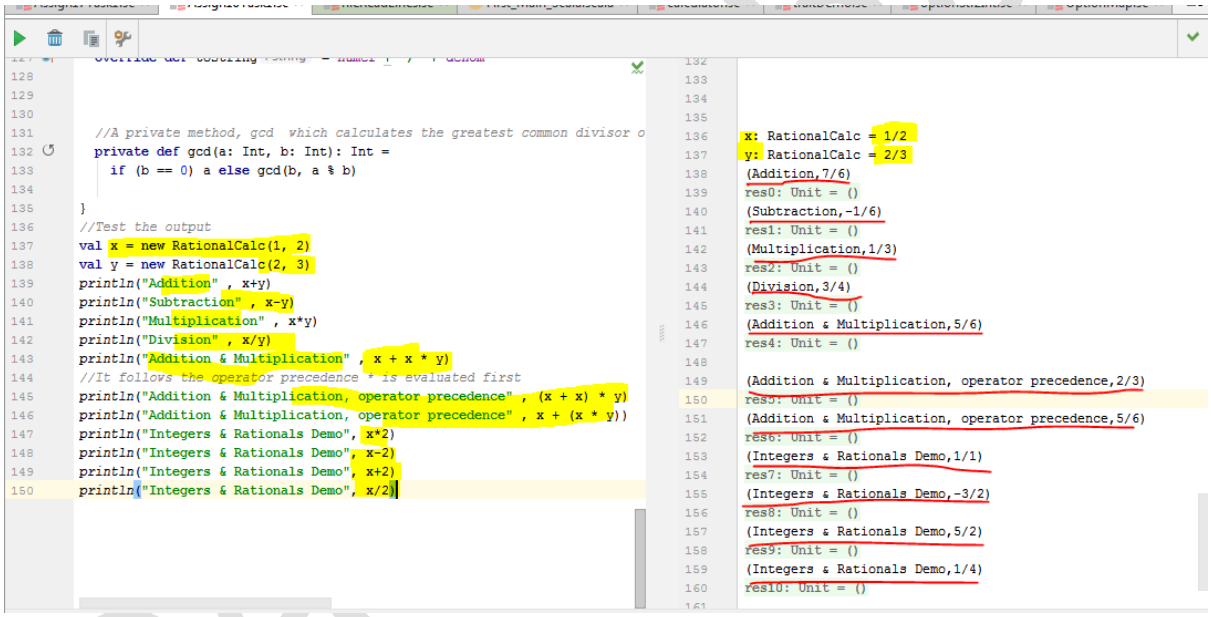
```
//NOTE: Overloaded Methods +-* or / works on (r * 2) does not work on Int args(2 *
//r) as Int class contains no multiplication method that takes
//a Rationalargument because class Rational is not a standard class in the Scala
//library. Will need to create an implicit conversion that automatically
//converts integers to rational numbers when needed. ""implicit def
//intToRational(x: Int) = new Rational(x)"" But this can only be done from
//interpreter:
//& not in this RationalCalc app. Since, this is out of scope of this.


/*
Identifiers 'n' and 'd' in the parentheses after the class name(Rational) are
called class parameters. The
Scala compiler will gather up these two class parameters and create a primary
constructor that takes the
same two parameters.
*/

class RationalCalc(n: Int, d: Int)
{

//A pre-condition of the primary constructor that tells value of "d" must be non-
//zero.
//Otherwise, require will prevent the object from being constructed by throwing
//an IllegalArgumentException.

  require(d != 0)
```

```
//we added a private field "g". It can be accessed inside the body of the class,
//but not outside.
//To normalize a rational number to an equivalent reduced form calling a gcd method
//over rational number.
//To ensure g is always positive, we pass the absolute value of n and d

  private val g = gcd(n.abs, d.abs)


//Since the compiler would not provide the values for these class parameters n &
//d by that.n or that.d
//because that does not refer to the Rationalobject on which add was invoked.To
//access the numerator and denominator
//we added two fields named numer and denom, and initialized them with the values
//of class parameters "n" and "d".
//By dividing n and d by their greatest common divisor, g, every Rational will be
//constructed in its normalized form:

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


//Constructors other than the primary constructor are called auxiliary
//constructors.
//adding an auxiliary constructor "this(n:Int)" to Rational that takes only one
//argument, the numerator, with the denominator predefined to be 1.

  def this(n: Int) = this(n, 1)


// "+" is a legal identifier in Scala.so, we'll define a public "+" method on
//class Rationalthat takes
//another Rational as a parameter. To keep Rational immutable, the "+" method must
//not add the passed
//rational number to itself. Rather, it must create and return a new Rational that
//holds the sum.

  def + (that: RationalCalc): RationalCalc =
    new RationalCalc(
      numer * that.denom + that.numer * denom,
      denom * that.denom
    )


//Method overloading for + ,in order to perform a addition between a rational
//number by an integer
  def + (i: Int): RationalCalc =
    new RationalCalc(numer + i * denom, denom)



// "-" is a legal identifier in Scala.so, we'll define a public "-" method on
//class Rationalthat takes
//another Rational as a parameter. To keep Rational immutable, the "-" method must
//not subtract the passed
//rational number from itself. Rather, it must create and return a new Rational
//that holds the Subtraction.

  def - (that: RationalCalc): RationalCalc =
    new RationalCalc(
      numer * that.denom - that.numer * denom,
      denom * that.denom
    )
```

```
//Method overloading for - ,in order to perform a subtraction between a rational
//number by an integer
  def - (i: Int): RationalCalc =
    new RationalCalc(numer - i * denom, denom)



// "*" is a legal identifier in Scala.so, we'll define a public "*" method on class
//Rationalthat takes
//another Rational as a parameter. To keep Rational immutable, the "*" method must
//not multiply the passed
//rational number to itself. Rather, it must create and return a new Rational that
//holds the Multiplication.

  def * (that: RationalCalc): RationalCalc =
    new RationalCalc(numer * that.numer, denom * that.denom)


//Method overloading for * ,in order to perform a multiplication between a rational
//number by an integer
  def * (i: Int): RationalCalc =
    new RationalCalc(numer * i, denom)



// "/" is a legal identifier in Scala.so, we'll define a public "/" method on class
//Rationalthat takes
//another Rational as a parameter. To keep Rational immutable, the "/" method must
//not Divide the passed
//rational number from itself. Rather, it must create and return a new Rationalthat
//holds the Division.

  def / (that: RationalCalc): RationalCalc =
    new RationalCalc(numer * that.denom, denom * that.numer)


//Method overloading for / ,in order to perform a division between a rational
//number by an integer
  def / (i: Int): RationalCalc =
    new RationalCalc(numer, denom * i)



/*
On Creating an instance of Rational class by default, class Rational inherits the
implementation of defined in
class java.lang.Object, which just prints the class name an @ sign, and a
hexadecimal number.
"override" modifier in front of a method definition "toString" signals that a
previous method definition is
overridden. So,one would get a nice print out of the Rational's numerator and
denominator values.
*/

  override def toString = numer + "/" + denom



//A private method, gcd  which calculates the greatest common divisor of two passed
//Ints.
  private def gcd(a: Int, b: Int): Int =
    if (b == 0) a else gcd(b, a % b)

}
```

```
//Test the output
val x = new RationalCalc(1, 2)
val y = new RationalCalc(2, 3)
println("Addition" , x+y)
println("Subtraction" , x-y)
println("Multiplication" , x*y)
println("Division" , x/y)
println("Addition & Multiplication" , x + x * y)
//It follows the operator precedence * is evaluated first
println("Addition & Multiplication, operator precedence" , (x + x) * y)
println("Addition & Multiplication, operator precedence" , x + (x * y))
println("Integers & Rationals Demo", x*2)
println("Integers & Rationals Demo", x-2)
println("Integers & Rationals Demo", x+2)
println("Integers & Rationals Demo", x/2)
```

ScreenShot:



********** **End** **********