Assignment 14

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)

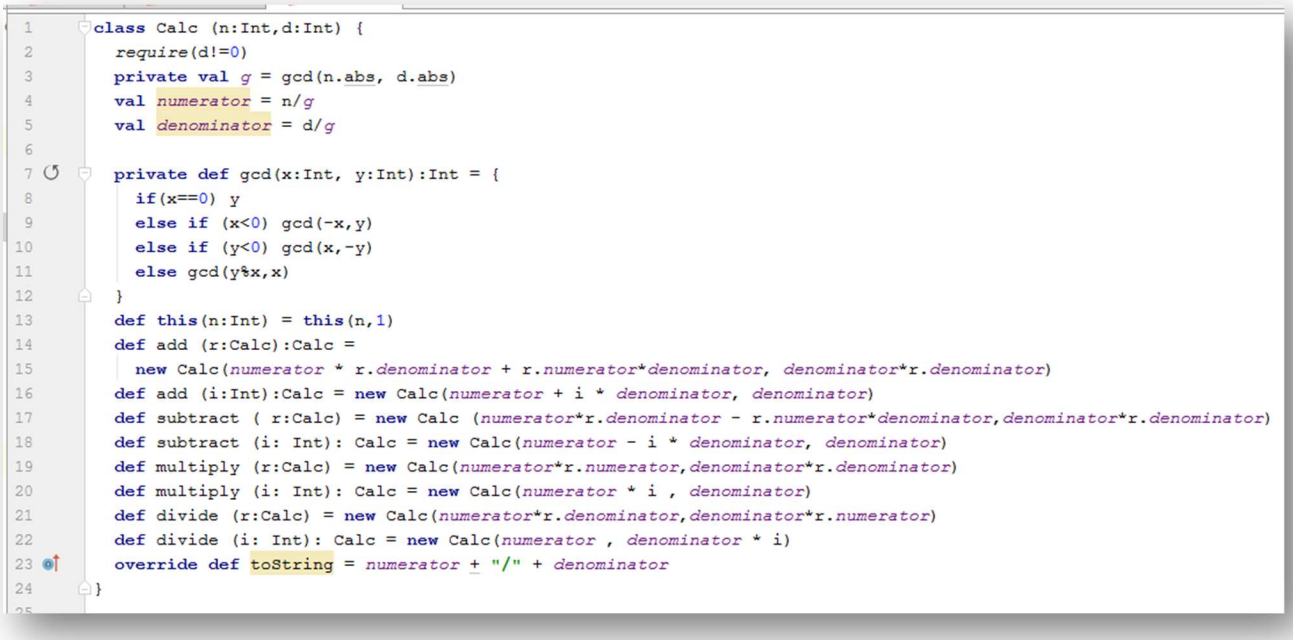
Add option to work with whole numbers which are also rational numbers i.e. (n/1)

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

Scala Program Code:

```
class Calc (n:Int,d:Int) {
  require(d!=0)
  private val g = gcd(n.abs, d.abs)
  val numerator = n/g
  val denominator = 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)
  def add (r:Calc):Calc =
    new Calc(numerator * r.denominator + r.numerator*denominator,
denominator*r.denominator)
  def add (i:Int):Calc = new Calc(numerator + i * denominator, denominator)
  def subtract ( r:Calc) = new Calc (numerator*r.denominator -
r.numerator*denominator,denominator*r.denominator)
  def subtract (i: Int): Calc = new Calc(numerator - i * denominator, denominator)
  def multiply (r:Calc) = new Calc(numerator*r.numerator,denominator*r.denominator)
  def multiply (i: Int): Calc = new Calc(numerator * i , denominator)
  def divide (r:Calc) = new Calc(numerator*r.denominator,denominator*r.numerator)
  def divide (i: Int): Calc = new Calc(numerator , denominator * i)
  override def toString = numerator + "/" + denominator
}
```



```
class Calc (n:Int,d:Int) {
  require(d!=0)
  private val g = gcd(n.abs, d.abs)
  val numerator = n/g
  val denominator = 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)
  def add (r:Calc):Calc =
    new Calc(numerator * r.denominator + r.numerator*denominator, denominator*r.denominator)
  def add (i:Int):Calc = new Calc(numerator + i * denominator, denominator)
  def subtract ( r:Calc) = new Calc (numerator*r.denominator - r.numerator*denominator,denominator*r.denominator)
  def subtract (i: Int): Calc = new Calc(numerator - i * denominator, denominator)
  def multiply (r:Calc) = new Calc(numerator*r.numerator,denominator*r.denominator)
  def multiply (i: Int): Calc = new Calc(numerator * i , denominator)
  def divide (r:Calc) = new Calc(numerator*r.denominator,denominator*r.numerator)
  def divide (i: Int): Calc = new Calc(numerator , denominator * i)
  override def toString = numerator + "/" + denominator
}
```

The statement: `def this(n:Int) = this(n,1)` works as auxiliary constructor

Below is the object class to run the above program code.

This statement enables us to work with whole numbers which are also rational numbers i.e.(n/1)

each function- add, subtract, multiply, divide has been defined in such a manner via method overloading that it allows the user to work with numbers and rational.

```
object CalcObj {

  def main(args: Array[String]): Unit = {
    val a = new Calc(29,10)
    val b = new Calc(17)
    val c = new Calc(18,2)
    val d = new Calc(11)
    val p = a add 5
    println(p)
    val q = b multiply new Calc(11,9)
    println(q)
    val r = c subtract new Calc(16,1)
    println(r)
    val s = d divide 51
    println(s)
  }
}
```

```
object CalcObj {

  def main(args: Array[String]): Unit = {
    val a = new Calc(29,10)
    val b = new Calc(17)
    val c = new Calc(18,2)
    val d = new Calc(11)
    val p = a add 5
    println(p)
    val q = b multiply new Calc(11,9)
    println(q)
    val r = c subtract new Calc(16,1)
    println(r)
    val s = d divide 51
    println(s)
  }
}
```

Output: