

## Assignment 16– Scala 3

### 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.

Code:

```
class Rational(n: Int, d: Int) {

  require(d != 0)
  private val g = gcd(n.abs, d.abs)

  val numer = n / g
  val denom = d / g
  def this(n: Int) = this(n, 1)
  def + (that: Rational): Rational =
    new Rational(
      numer * that.denom + that.numer * denom,
      denom * that.denom
    )
  def + (i: Int): Rational =
    new Rational(numer + i * denom, denom)
  def - (that: Rational): Rational =
    new Rational(
      numer * that.denom - that.numer * denom,
      denom * that.denom
    )
}
```

## Assignment 16– Scala 3

```
)  
def - (i: Int): Rational =  
new Rational(number - i * denom, denom)  
def * (that: Rational): Rational =  
new Rational(number * that.number, denom * that.denom)  
def * (i: Int): Rational =  
new Rational(number * i, denom)  
def / (that: Rational): Rational =  
new Rational(number * that.denom, denom * that.number)  
def / (i: Int): Rational =  
new Rational(number, denom * i)  
override def toString = number + "/" + denom
```

```
private def gcd(a: Int, b: Int): Int =  
    if (b == 0) a else gcd(b, a % b)  
}
```

```
object RationalMain {  
def Options() = {  
    println("1. Add a rational")  
    println("2. Subtract a rational")  
    println("3. Multiply a rational")  
    println("4. Add a number")  
    println("5. Subtract a number")  
    println("6. Multiply a number")  
    println("7. Exit")  
}
```

```
def Compute(rational: Rational, input: Int): Rational = {
```

```
    input match {  
        case 1 =>  
            val p = scala.io.StdIn.readInt()  
            val q = scala.io.StdIn.readInt()  
            rational + (new Rational(p, q))
```

## Assignment 16– Scala 3

```
case 2 =>
  val p = scala.io.StdIn.readInt()
  val q = scala.io.StdIn.readInt()
  rational.-(new Rational(p, q))
case 3 =>
  val p = scala.io.StdIn.readInt()
  val q = scala.io.StdIn.readInt()
  rational.*(new Rational(p, q))
case 4 =>
  val p = scala.io.StdIn.readInt()
  rational.+(new Rational(p))
case 5 =>
  val p = scala.io.StdIn.readInt()
  rational.-(new Rational(p))
case 6 =>
  val p = scala.io.StdIn.readInt()
  rational.*(new Rational(p))
case _ =>
  rational
}

def main(args: Array[String]): Unit = {
  var rationalNumber: Rational = new Rational(0)

  var input = 1

  do {
    Options()
    input = scala.io.StdIn.readInt()
    rationalNumber = Compute(rationalNumber, input)
    println("Output is : " + rationalNumber.toString)
  } while (input != 7)
}
```

## Assignment 16– Scala 3

### Outcome:

1. Add a rational
2. Subtract a rational
3. Multiply a rational
4. Add a number
5. Subtract a number
6. Multiply a number
7. Exit

The screenshot shows an IDE with the file `Rational1.scala` open. The code defines a `Rational` class with methods for addition, subtraction, multiplication, and division of rational numbers, as well as methods for adding and subtracting integers. The console output shows the execution of a program that uses these methods to perform various operations on rational numbers and integers, matching the tasks listed in the 'Outcome' section.

```
class Rational(n: Int, d: Int) {
  require(d != 0)
  private val g = gcd(n.abs, d.abs)
  val numer = n / g
  val denom = d / g

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

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

  def + (i: Int): Rational =
    new Rational(numer + i * denom, denom)

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

  def - (i: Int): Rational =
    new Rational(numer - i * denom, denom)

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

  def * (i: Int): Rational =
    new Rational(numer * i, denom)
}
```

Console Output:

```
New_configuration [Scala Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (Oct 31, 2017, 8:24:03 PM)
1. Add a rational
2. Subtract a rational
3. Multiply a rational
4. Add a number
5. Subtract a number
6. Multiply a number
7. Exit
```

## Assignment 16– Scala 3

The screenshot shows an IDE with a Scala file named `Rational1.scala`. The code defines a `Rational` class and a `main` function. The `Rational` class has fields `n: Int` and `d: Int`, and methods `q`, `numerator`, `denominator`, `this(n: Int)`, `+(that: Rational)`, `+(i: Int): Rational`, `-(that: Rational)`, `-(i: Int): Rational`, `*(that: Rational)`, `*(i: Int): Rational`, `/(that: Rational)`, `/(i: Int): Rational`, `toString`, and `gcd(a: Int, b: Int)`. The `main` function reads input from the user and performs operations based on the input number (1-6) until the user enters 7 to exit.

```
val q = scala.io.StdIn.readInt()
rational.*(new Rational(p, q))
case 4 =>
  val p = scala.io.StdIn.readInt()
  rational.+(new Rational(p))
case 5 =>
  val p = scala.io.StdIn.readInt()
  rational.-(new Rational(p))
case 6 =>
  val p = scala.io.StdIn.readInt()
  rational.*(new Rational(p))
case _ =>
  rational
}

def main(args: Array[String]): Unit = {
  var rationalNumber: Rational = new Rational(0)

  var input = 1
  do {
    Options()
    input = scala.io.StdIn.readInt()
    rationalNumber = Compute(rationalNumber, input)
    println("Output is : " + rationalNumber.toString)
  } while (input != 7)
}
```

The Outline pane on the right shows the structure of the code:

- `Rational`
  - `n: Int`
  - `d: Int`
  - `q`
  - `numerator`
  - `denom`
  - `this(n: Int)`
  - `+(that: Rational)`
  - `+(i: Int): Rational`
  - `-(that: Rational)`
  - `-(i: Int): Rational`
  - `*(that: Rational)`
  - `*(i: Int): Rational`
  - `/(that: Rational)`
  - `/(i: Int): Rational`
  - `toString`
  - `gcd(a: Int, b: Int)`
- `RationalMain`
  - `Options()`
  - `Compute(rati`
  - `main(args: Ar`
    - `rationalNu`
    - `input`

The Console pane at the bottom shows the output of the program:

```
New_configuration [Scala Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (Oct 31, 2017, 8:24:03 PM)
1. Add a rational
2. Subtract a rational
3. Multiply a rational
4. Add a number
5. Subtract a number
6. Multiply a number
7. Exit
```