BIG DATA HADOOP AND SPARK DEVLOPMENT

ASSIGNMENT 16

Table of Contents:

BIG DATA HADOOPAND SPARK DEVELOPMENT

1. Introduction

In this assignment, the given tasks are performed and Output of the tasks are recorded in the form of Screenshots.

2. Objective

This Assignment consolidates the deeper understanding of the Session – 16 SCALA BASICS 3

3. Problem Statement

- 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.

4. Expected Output

- 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.**

Here we are creating a class Rational

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

   def this(n: Int) = this(n, 1)    // auxiliary constructors
   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.
  def +(that: Rational): Rational =
  new Rational(numer * that.denom + that.numer * denom, denom * that.denom)

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

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

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

  def +(that: Int): Rational = this + new Rational(that)

  def -(that: Int): Rational = this - new Rational(that)

  def *(that: Int): Rational = this * new Rational(that)

  def /(that: Int): Rational = this / new Rational(that)

  override def toString() = numer+"/"+denom

}
```

Here we are creating a main class

```
object ration {

    def main(args: Array[String]): Unit = {
    println("Hello, world!")

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

    val a = x * x
    println("Result a: " +a)

    val b = a * 2
    println("Result b: " +b)

    val z = (x + y) * x
    println("Result z: " +z)

    implicit def intToRational(x: Int) = new Rational(x)
    val r = new Rational(2,3)
    val s = 2 * r
    println("Result s: " +s)

 }                                  //> main: (args: Array[String])Unit
 }
```

Below is the output for **Creating a calculator to work with rational numbers**

```
Hello, world!
Result a: 4/9
Result b: 8/9
Result z: 17/18
Result s: 4/3
```

Below is the screenshot for output of **Creating a calculator to work with rational numbers**