Brandon Bardwell

3/22/20

Computer Science II

Module 5: Assignment 2:

For this assignment I used Solution Manual to help me code the program.

<http://soultionmanual.blogspot.com/2016/12/chapter-13-exercise-19-introduction-to.html>

My Code:

**import** java.util.Scanner;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Scanner input = **new** Scanner(System.***in***);

System.***out***.print("Enter a decimal number: ");

String s = input.next();

System.***out***.println("The fractional number is " + **new** BigRational(s));

}

}

**import** java.math.BigDecimal;

**import** java.math.RoundingMode;

**public** **class** BigRational **extends** Number **implements** Comparable<BigRational> {

// Data fields for numerator and denominator

**private** BigDecimal numerator = BigDecimal.***ZERO***;

**private** BigDecimal denominator = BigDecimal.***ONE***;

/\*\* Construct a rational with default properties \*/

**public** BigRational() {

**this**(BigDecimal.***ZERO***, BigDecimal.***ONE***);

}

/\*\* Construct a rational with specified numerator and denominator \*/

**public** BigRational(BigDecimal numerator, BigDecimal denominator) {

BigDecimal gcd = *gcd*(numerator, denominator);

**this**.numerator = ((denominator.compareTo(BigDecimal.***ZERO***) > 0) ? BigDecimal.***ONE*** : **new** BigDecimal(-1)).multiply(numerator).divide(gcd);

**this**.denominator = denominator.abs().divide(gcd);

}

**public** BigRational(String decimal) {

**int** index = (decimal.contains(".")) ? decimal.indexOf('.') : decimal.indexOf('/');

BigDecimal d;

BigDecimal n;

// if string is in decimal form

**if** (decimal.contains(".")) {

**int** power = decimal.substring(index + 1, decimal.length()).length();

d = **new** BigDecimal(Math.*pow*(10,power));

n = **new** BigDecimal(**new** StringBuilder(decimal).deleteCharAt(index).toString());

} **else** {

// if string contains '/'

n = **new** BigDecimal(decimal.substring(0, index));

d = **new** BigDecimal(decimal.substring(index + 1, decimal.length()));

}

BigDecimal gcd = *gcd*(n, d);

**this**.numerator = ((d.compareTo(BigDecimal.***ZERO***) > 0) ? BigDecimal.***ONE*** : **new** BigDecimal(-1)).multiply(n).divide(gcd);

**this**.denominator = d.abs().divide(gcd);

}

/\*\* Find GCD of two numbers \*/

**private** **static** BigDecimal gcd(BigDecimal n, BigDecimal d) {

BigDecimal n1 = n.abs();

BigDecimal n2 = d.abs();

BigDecimal remainder = n1.remainder(n2);

**while** (remainder.compareTo(BigDecimal.***ZERO***) > 0) {

n1 = n2;

n2 = remainder;

remainder = n1.remainder(n2);

}

**return** n2;

}

/\*\* Return numerator \*/

**public** BigDecimal getNumerator() {

**return** numerator;

}

/\*\* Return denominator \*/

**public** BigDecimal getDenominator() {

**return** denominator;

}

/\*\* Add a rational number to this rational \*/

**public** BigRational add(BigRational secondBigRational) {

BigDecimal n1 = numerator.multiply(secondBigRational.getDenominator());

BigDecimal n2 = denominator.multiply(secondBigRational.getNumerator());

BigDecimal n = n1.add(n2);

BigDecimal d = denominator.multiply(secondBigRational.getDenominator());

**return** **new** BigRational(n, d);

}

/\*\* Subtract a rational number from this rational \*/

**public** BigRational subtract(BigRational secondBigRational) {

BigDecimal n1 = numerator.multiply(secondBigRational.getDenominator());

BigDecimal n2 = denominator.multiply(secondBigRational.getNumerator());

BigDecimal n = n1.subtract(n2);

BigDecimal d = denominator.multiply(secondBigRational.getDenominator());

**return** **new** BigRational(n, d);

}

/\*\* Multiply a rational number to this rational \*/

**public** BigRational multiply(BigRational secondBigRational) {

BigDecimal n = numerator.multiply(secondBigRational.getNumerator());

BigDecimal d = denominator.multiply(secondBigRational.getDenominator());

**return** **new** BigRational(n, d);

}

/\*\* Divide a rational number from this rational \*/

**public** BigRational divide(BigRational secondBigRational) {

BigDecimal n = numerator.multiply(secondBigRational.getDenominator());

BigDecimal d = denominator.multiply(secondBigRational.numerator);

**return** **new** BigRational(n, d);

}

@Override

**public** String toString() {

**if** (denominator.equals(BigDecimal.***ONE***))

**return** numerator + "";

**else**

**return** numerator + "/" + denominator;

}

@Override // Override the equals method in the Object class

**public** **boolean** equals(Object other) {

**if** ((**this**.subtract((BigRational)(other))).getNumerator().equals(BigDecimal.***ZERO***))

**return** **true**;

**else**

**return** **false**;

}

@Override // Implement the abstract intValue method in Number

**public** **int** intValue() {

**return** (**int**)doubleValue();

}

@Override // Implement the abstract floatValue method in Number

**public** **float** floatValue() {

**return** (**float**)doubleValue();

}

@Override // Implement the doubleValue method in Number

**public** **double** doubleValue() {

**return** numerator.divide(denominator).doubleValue();

}

**public** BigDecimal bigDecimalDouble() {

**return** numerator.divide(denominator, 100, RoundingMode.***HALF\_DOWN***);

}

@Override // Implement the abstract longValue method in Number

**public** **long** longValue() {

**return** (**long**)doubleValue();

}

@Override // Implement the compareTo method in Comparable

**public** **int** compareTo(BigRational o) {

**if** (**this**.subtract(o).getNumerator().compareTo(BigDecimal.***ZERO***) > 0)

**return** 1;

**else** **if** (**this**.subtract(o).getNumerator().compareTo(BigDecimal.***ZERO***) < 0)

**return** -1;

**else**

**return** 0;

}

}