**Computer Science 2**   **Lab # 09**



**Dr. Hanh Pham**

**Student Last Name: Martinez Student First Name: Adriel**

**CS2 Section # 01**

**Due:** Problem A by the **end of the lab** and Problems B by the end of **Saturday** of the same week.

**TOPIC: OBJECTS**

**Problem A:**

Please fill-in the field following this symbol images:bighand.jpg with your own words and in **BLUE font color**

## Pre-Lab Visualization

### Rational

Here is a list of responsibilities for the rational class:

1. Know the value of the denominator.
2. Know the value of the numerator.
3. Be able to compute the negation of a rational number.
4. Be able to compute the reciprocal of a rational number.
5. Be able to compare two rational numbers for equality.
6. Be able to compute the sum of two rational numbers.
7. Be able to compute the difference of two rational numbers.
8. Be able to compute the result of multiplying two rational numbers.
9. Be able to compute the result of dividing two rational numbers.
10. Be able to compute a printable representation of the rational number.

1)What data fields will the Rational class need to implement these responsibilities?

The data fields that the class will need to implement for these responsibilities are two private integers, one that holds the value of the numerator and the other for the denominator.

2)Are there any constraints on the values of the data fields?

One constraint on the values of the data fields is that the denominator cannot equal to zero, otherwise the class will throw an exception. Another constraint is that if the denominator is less than 0, meaning it is a negative number, the class will have the negative sign be moved over to the numerator instead.

3)Here is a list of constructors and methods that will be used to implement the responsibilities. Fill in the missing pre-conditions, post-conditions, and test cases.

Rational()

Pre-condition: none.

Post-condition: The rational number 1 has been constructed.

Test cases: none.

Rational(n, d)

Pre-condition: The denominator d is non-zero.

Post-condition: The rational number n/d has been constructed and is in normal form.

Test cases:

n = 2, d = 4; result is 1/2

n = 0, d = 7; result is 0/1

n = 12, d =–30; result is –2/5

n = 4, d = 0; result is Exception

int getNumerator()

Pre-condition: The rational n/d is in a valid state.

Post-condition: The value n is returned.

Test cases:

n/d is 1/2; result is 1

n/d is 0/1; result is 0

n/d is –2/5; result is –2

int getDenominator()

images:bighand.jpg Pre-condition: The rational n/d is in a valid state.

images:bighand.jpg Post-condition: The value d is returned.

images:bighand.jpg Test cases:

n/d is 1/2; result is 2;

n/d is 0/1; result is 1;

n/d is -2/5; result is 5;

4)

Rational negate()

Pre-condition: The rational n/d is in a valid state.

Post-condition: The rational number –n/d has been returned.

images:bighand.jpg Test cases:

n/d is 1/2; result is -1/2;

n/d is 2/3; result is -2/3;

n/d is -2/5; result is 2/5;

5)

Rational reciprocal()

images:bighand.jpg Pre-condition: The rational n/d is in a valid state.

images:bighand.jpg Post-condition: The rational number d/n has been returned.

images:bighand.jpg Test cases:

n/d is 3/4; result is 4/3;

n/d is 6/4; result is 4/6;

n/d is 5/9; result is 9/5;

6)

boolean equals(Object other)

images:bighand.jpg Pre-condition:

images:bighand.jpg Post-condition:

images:bighand.jpg Test cases:

7)

Rational add(Rational other)

Pre-condition: The rational n/d is in a valid state and other is the valid rational x/y.

Post-condition: The rational number (ny+xd)/dy has been returned.

Test cases:

n/d is 1/2, x/y is 1/2; result is 1/1

n/d is 1/2' x/y is 1/6; result is 2/3

n/d is 3/4, x/y is 5/6; result is 19/12

n/d is 1/3, x/y is –2/3; result is –1/3

8)

Rational subtract(Rational other)

images:bighand.jpg Pre-condition: The rational n/d is in a valid state and other is the valid rational x/y.

images:bighand.jpg Post-condition: The rational number (ny-xd)/dy has been returned.

images:bighand.jpg Test cases:

n/d is 1/2, x/y is 1/2; result is 0

n/d is 1/2, x/y is 1/6; result is 1/3

n/d is 3/4, x/y is 5/6; result is -1/12

n/d is 1/3, x/y is –2/3; result is 1

9)

Rational multiply(Rational other)

images:bighand.jpg Pre-condition: The rational n/d is in a valid state and other is the valid rational x/y.

images:bighand.jpg Post-condition: The rational number (nx)/dy has been returned.

images:bighand.jpg Test cases:

n/d is 1/2, x/y is 1/2; result is 1/4

n/d is 1/2, x/y is 1/6; result is 1/12

n/d is 3/4, x/y is 5/6; result is 5/8

n/d is 1/3, x/y is –2/3; result is -2/9

10)

Rational divide(Rational other)

images:bighand.jpg Pre-condition: The rational n/d is in a valid state and other is the valid rational x/y.

images:bighand.jpg Post-condition: The rational number (ny)/dx has been returned.

images:bighand.jpg Test cases:

n/d is 1/2, x/y is 1/2; result is 1

n/d is 1/2, x/y is 1/6; result is 3

n/d is 3/4, x/y is 5/6; result is 9/10

n/d is 1/3, x/y is –2/3; result is -1/2

11)

String toString()

Pre-condition: The rational n/d is in a valid state.

Post-condition: The string “n/d” has been returned.

Test cases:

n/d is 1/2; result is “1/2”

n/d is 0/1; result is “0/1”

n/d is –2/5; result is “–2/5”

### Counter

Our counter will be a class that acts like a simple click counter (used for counting attendance) with a few improvements. The click counter will have a minimum and maximum value. It will start at the minimum value. Each click will add one to the counter, except when the counter hits the maximum value, where it will roll back over to the minimum. The click counter will also support an operation that decreases the value of the counter by one. If this would decrease the value below the minimum, it will roll over to the maximum value.

Think about the preceding description and give a list of responsibilities for the Counter class.

1. Know the value of the counter.
2. Know the if the counter rolled over.
3. Know if two counters are in the same state.
4. Be able to increase the value of the counter.
5. Be able to decrease the value of the counter.
6. Be able to compute a printable representation of the current, maximum, and minimum.

What data fields will the Counter class need to implement these responsibilities?

The data fields this class needed to implement these responsibilities are three private integer values, those being the minimum value, the maximum value, and the current value. Also, an additional Boolean value, rollOver is necessary in order to implement these responsibilities.

Are there any constraints on these data fields?

The constraints that could with these data fields is that the maximum cannot exceed the largest integer possible.

12)

Give a list of constructors and methods that will be used to implement the responsibilities you have listed. Fill in the pre-conditions, post-conditions, and test cases.

Counter ()

Pre-condition: none.

Post-condition: The counter has been constructed (Minimum and current is 0, maximum is the largest integer possible, and rollOver is false).

Test cases: none.

Counter (min, max)

Pre-condition: The minimum value must be less than the maximum value.

Post-condition: The counter has been constructed with given values.

Test cases:

min= 3, max= 6, current= 3, rollOver=false.

min= 2, max= 5, current= 2, rollOver=false.

min= 5, max=3, current= 5, rollOver=false. 🡸 Result is an exception

equals(Object otherObject)

Pre-condition: otherObject is being compared to test for equality (If two counters are both in the same state)

Post-condition:

Test cases:

increase()

Pre-condition: Current is incremented by 1 and then compared to see if it greater than the maximum, if so, then the minimum is assigned to current and rollOver becomes true.

Post-condition:

Test cases:

decrease()

Pre-condition: Current is decremented by 1 and then compared to see if it less than the maximum, if so, then the maximum is assigned to current and rollOver becomes true.

Post-condition:

Test-cases:

value()

Pre-condition: Current is a value that is equal to or greater than zero.

Post-condition: The value current is returned.

Test-cases:

current= 5. result is 5.

current= 1. result is 1.

current= 7. result is 7.

rolledOver()

Pre-condition: none

Post-condition: The value is true or false dependent on whether the count rolled over on the last count

Test-cases: