CSCI 2600 Principles of Software

Homework 3: Java and Coding to Specification

Due: July 10, 2020, 11:59:59 pm

As always, we recommend that you read the entire homework before you begin work. As always, you must commit and push to your git repository, then submit on Submitty.

Submission Instructions

- Follow the directions in the version control handout for cloning your git repo.
- Be sure to commit and push the files to Submitty. Follow the directions in the version control handout for adding and committing files.
- Be sure to add the problem1.pdf, problem2.pdf, problem3.pdf, collaboration.pdf, and reflection.pdf files to the answers folder in your repo in Eclipse using Team/Add to Index.

  You MUST type up your answers. Handwritten solutions will not be accepted or graded, even if they are scanned into a PDF file.
  We recommend using LaTeX. If you have never used LaTeX, take a look at this tutorial.
- **Important**: You must press the **Grade My Repository** button, or your answers will not be graded.

Introduction

This homework focuses on reading and interpreting specifications, and reading and writing Java source code. Additionally, you will be given an introduction to using `checkRep` methods and testing strategies. You will complete the implementation of a polynomial with rational coefficients, and you will answer questions about both the code you are given and the code you are going to write.

To complete this homework, you will need to know:

1. Basic algebra (rational and polynomial arithmetic) and polynomial arithmetic
2. How to read and write basic Java code
   - code structure and layout (class and method definition, field and variable declaration)
   - method calls
   - operators for:
     - object creation: `new`
     - field and method access: `.`
     - assignment: `=`
     - comparison: `==`, `!=`, `<`, `>`, `≤`, `≥`
     - arithmetic: `+`, `-`, `*`, `/`
   - control structure: loops (`while` and `for`) and conditional branches (`if`, `else`)
3. How to read procedural specifications in the Principles of Software convention (requires, modifies, effects, returns and throws)

Problems

You should have files `RatNum.java` and `RatPoly.java` in directory `src/main/java/hw3/`, and files `RatNumTest.java`, `RatPolyTest.java` and `RatPolyDivideTest.java` in directory `src/test/java/hw3/`. Read through the specifications in `RatNum.java` and `RatPoly.java` to help you work through the problems below.

Problem 1: RatNum (5 pts)

For this first problem you don't have to write any code, but you do have to answer written questions. Read the specifications and provided implementation in `RatNum.java`, a class representing rational numbers.

You may want to look at the code in `RatNumTest.java` to see example usages of the RatNum class (albeit in the context of a test driver, rather than application code).

Answer the following questions, writing your answers in the file `answers/problem1.pdf`. **Two or three sentences should be enough to answer each question. For full credit your answers should be short and to the point. Points will be deducted for answers that are excessively long or contain irrelevant information.**

1. Classify each public method of RatNum as a creator, observer, producer or mutator.
2. `add`, `sub`, `mul`, and `div` all require that "arg != null". This is because all of the methods access fields of 'arg' without checking if 'arg' is null first. But the methods also access fields of 'this' without checking for null; why is "this != null" absent from the requires-clause for the methods?
3. Why is RatNum.valueOf(String) a static method? What alternative to static methods would allow one to accomplish the same goal of generating a RatNum from an input String?
4. `add`, `sub`, `mul`, and `div` all end with a statement of the form `return new RatNum ( numerExpr , denomExpr);`. Imagine an implementation of the same function except the last statement is:

   ```
   this.numer = numerExpr;
   this.denom = denomExpr;
   return this;
   ```

   For this question, pretend that the `this.numer` and `this.denom` fields are not declared as `final` so that these assignments compile properly. How would the above changes fail to meet the specifications of the function (Hint: take a look at the `@requires` and `@modifies` statements, or lack thereof) and fail to meet the specifications of the RatNum class?

5. Calls to `checkRep` are supposed to catch violations in the classes' invariants. In general, it is recommended that one call `checkRep` at the beginning and end of every method. In the case of `RatNum`, why is it sufficient to call `checkRep` only at the end of the constructors? (Hint: could a method ever modify a `RatNum` such that it violates its representation invariant? Could a method change a `RatNum` at all? How are changes to instances of `RatNum` prevented?)

Problem 2: RatPoly (34 pts: Code: 30 pts autograded, Answers: 4 pts)

Read over the specifications for the `RatPoly` class, making sure you understand the overview for RatPoly and the specifications for the given methods. Read through the provided skeletal implementation of `RatPoly.java`.

Fill in an implementation for all methods.

You may define new private helper methods as you like. You may not add public methods. The external interface must remain the same. If you define new methods, you must specify them completely. You can consider the specifications of existing methods (where you fill in the body) to be adequate. You should comment any code you write, as needed; please do not over-comment.

We have provided a `checkRep()` method in RatPoly that tests whether or not a RatPoly instance violates the representation invariants. We highly recommend you use `checkRep()` where appropriate in the code you write. Think about the issues discussed in the last question of problem 1 when deciding where `checkRep` should be called.

There is a fairly rigorous test suite in `RatPolyTest.java`. You can run the given test suite with JUnit to evaluate your progress and the correctness of your code. It is probably a good idea to run tests individually rather than running the entire suite at once.

Answer the following questions, writing your answers in the file `answers/problem2.pdf`. **Remember, keep your answers to 2-3 sentences.**

1. We have chosen the array representation of a polynomial: RatNum[] coeffs, where coeffs[i] stores the coefficient of the term of exponent i. An alternative data representation is the list-of-terms representation: List<Term> terms, where each Term object stores the term's RatNum coefficient and integer exponent. The beauty of the ADT methodology is that we can switch from one representation to the other without affecting the clients of our RatPoly. Briefly list the advantages and disadvantages of the array representation versus the list-of-terms representation.
2. Where did you include calls to checkRep in RatPoly (at the beginning of methods, the end of methods, the beginning of constructors, the end of constructors, some combination)? Why?

Problem 3: Polynomial division (10 pts: Pseudocode, invariants and proofs: 5 pts, Java code: 5 pts autograded)

Write a pseudocode algorithm for division. State the loop invariant for the main loop and prove partial correctness. Write your answer in the file `answers/problem3.pdf`

When writing pseudocode use symbols +,-,* and / to express rational number and polynomial arithmetic. You may also use u[i] to retrieve the coefficient at power i of polynomial u, as well as c*x^i to denote the single-term polynomial of degree i and coefficient c. Important: write your pseudocode, invariants and proofs first, then write the Java code. Going backwards will be harder.

Use the following definition of polynomial division:

> Given two polynomials u and v, with v != "0", we can divide u by v to obtain a quotient polynomial q and a remainder polynomial r satisfying the condition u = "q * v + r", where the degree of r is strictly less than the degree of v, the degree of q is no greater than the degree of u, and r and q have no negative exponents.
>
> **For the purposes of this class, the operation "u / v" returns q as defined above**.
>
> The following are examples of division's behavior:
>
> - (x^3-2*x+3) / (3*x^2) = 1/3*x (with r = "-2*x+3").
> - (x^2+2*x+15) / (2*x^3) = 0 (with r = "x^2+2*x+15").
> - (x^3+x-1) / (x+1) = x^2-x+2 (with r = "-3").
>
> Note that this truncating behavior is similar to the behavior of integer division on computers.

For the proof question, you do not need to handle division by zero; however, you will need to do so in the Java program.

Next, complete the public method `public RatPoly div(RatPoly)` in the `RatPoly` class and transfer your pseudocode algorithm into `div`. You may assume that the divisor `p` is non-null. If `p` is zero `div` returns some `q` such that `q.isNaN`. As with the other operations (e.g., `mul`), if `this.isNaN` or `p.isNaN`, `div` returns some `q` such that `q.isNaN` is true.

There is an extensive test suite for division in `RatPolyDivideTest.java`. You can run this test suite to evaluate your progress and the correctness of your code.

Collaboration (0.5 pts)

Please answer the following questions in a file named `collaboration.pdf` in your `answers/` directory.

The standard academic integrity policy applies to this homework.

State whether or not you collaborated with other students. If you did collaborate with other students, state their names and a brief description of how you collaborated.

Reflection (0.5 pts)

Please answer the following questions in a file named `reflection.pdf` in your `answers/` directory. Answer briefly, but in enough detail to help you improve your own practice via introspection and to enable the course staff to improve Principles of Software in the future.

1. In retrospect, what could you have done better to reduce the time you spent solving this homework?
2. What could the Principles of Software staff have done better to improve your learning experience in this homework?
3. What do you know now that you wish you had known before beginning the homework?

What to submit

Push to git the following files. Don't forget to submit in Submitty!

- `answers/problem1.pdf`
- `answers/problem2.pdf`
- `answers/problem3.pdf`
- `src/main/java/hw3/RatPoly.java`
- `answers/collaboration.pdf`
- `answers/reflection.pdf`

All of the unfinished methods in RatPoly throw RuntimeExceptions. When you implement a method, you should be sure to remove the `throw new RuntimeException();` statement. For those of you who use Eclipse, we have also added a `TODO:` comment in each of these methods. The "Tasks" window will give you a list of all `TODO:` comments, which will help you find and complete these methods.

Think before you code! The polynomial arithmetic functions are not difficult, but if you begin implementation without a specific plan, it is easy to get yourself into a terrible mess.

The provided test suites in this homework are the same ones we will use to grade your implementation. In later homework assignments the staff will not provide such a thorough set of test cases to run on your implementations. You will be responsible for writing your test suites. For this homework, you can consider the provided set of tests to be rigorous enough that **you do not need to write your own tests**.

Errata

Errata will be posted on the Submitty Discussion Forum.