CMPU-102 – Data structures and Algorithms (S22)

Hannah Gommerstadt & Rui Meireles

Homework #3 – Polynomial calculator

Due April 29th 11:59 PM, checkpoint April 15th

Introduction

In this homework you will implement a Java-based polynomial calculator. This calculator will be able to add, subtract, multiply and divide polynomials. A polynomial can be thought of as a sum of monomials, which are themselves polynomials with a single term. A monomial is composed of a coefficient and an exponent that are associated with a given variable.

In our program, all monomials will be a function of the variable x. For example, the polynomial $-2x^3-4$ is the sum of the monomial $-2x^3$ and the monomial -4. The monomial $-2x^3$ has a coefficient of -2 and an exponent of 3. The calculator should support polynomials with real coefficients, but you may assume that the exponents are nonnegative integers.

Implementation requirements

The starter code has two files: `Poly.java` and `PolyTest.java`. `Poly.java` has stubs for the following methods:

1. A constructor `Poly(double coef, int exp)` that makes a single term polynomial out of the given coefficient and exponent. For example, `new Poly(2.3, 4)` should create the polynomial $2.3x^4$.

2. A method `public boolean equals(Object o)` that returns true if `o` represents a polynomial that is equivalent to the one on which the method is called. For example, $3.0x^4+0.0x^2+5$ and $3.0x^4+5.0$ should be considered equivalent, but $3.0x^4+5.0$ and $3.0x^4+3.0$ should not. This method is used by JUnit in the tests in the `PolyTest.java` file.

3. A method `toString()` that produces a textual representation of a polynomial. For example, `"2.0x^3 - 4.0x + 5.0"`. The Javadoc in `Poly.java` provides a detailed specification of the format that should be used for the textual representation.

4. A method `Poly add(Poly p)` that adds the polynomial passed in as an argument to the polynomial on which the method is called on, and returns a new polynomial with the resulting sum.

5. A method `Poly subtract(Poly p)` that subtracts the polynomial passed in as an argument from the polynomial on which the method is called on, and returns a new polynomial with the resulting difference.

6. A method `Poly multiply(Poly p)` that multiplies the polynomial passed in as an argument with the polynomial on which the method is called on, and returns a new polynomial with the resulting product.

7. A method `Poly divide(Poly p)` that divides the polynomial on which the method is called on, by the polynomial passed in as an argument, and returns a new polynomial with the resulting quotient. Note that not all polynomials are divisible by all others. If a division yields a non-zero remainder, this method should return `null`. `null` should also be returned if a division by zero is attempted. For a refresher on polynomial long division, we recommend this article: `https://en.wikipedia.org/wiki/Polynomial_long_division`.

The `PolyTest.java` file contains JUnit tests to help your test your calculator. Your program will need to pass all of them. **We also require you to add 5 more test methods at the end of this file**. Your tests must make polynomials, call methods on them, and evaluate their return values. The quality and usefulness of your tests will be evaluated.

In addition to implementing the polynomial operations, your calculator will have a text interface. The text interface will:

1. Be executed when the program is executed through the command line with `java PolyCalc`. I.e., your program's `main` method should be part of a class called `PolyCalc`.

2. Print a welcome message with your name.

3. Ask the user for the first polynomial. Polynomials will be specified using space-separated pairs of coefficient and exponents. For example, `2.5x^2 - 1` should be input as `2.5 2 -1 0`. If the user doesn't provide a valid polynomial, they should be asked again, until they do.

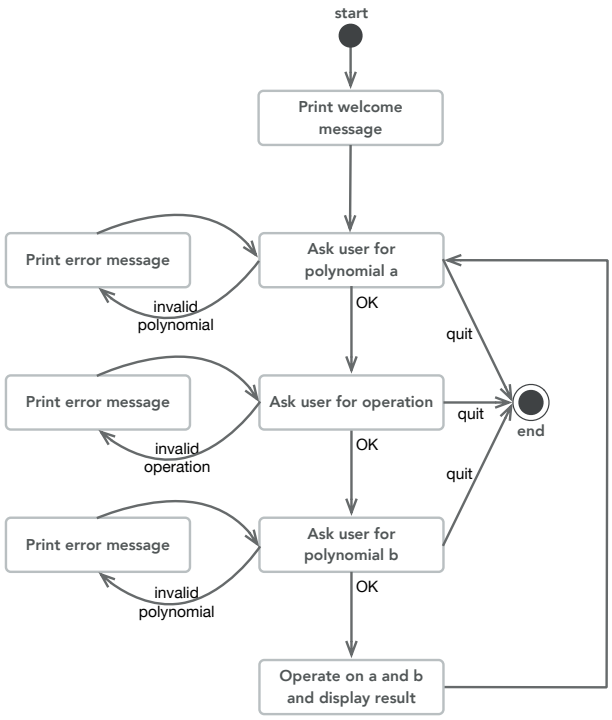
4. Ask the user for the operation. The possible choices are: `+`, `-`, `*` or `/`. If the user doesn't provide a valid operation, they should be asked again, until they do.

5. Ask the user for the second polynomial.

6. Execute the selected operation on the two polynomials and display the result. Note that if the operation is division and the two polynomials are not divisible, an error message should be printed.

7. The user can enter `quit` at any time to exit the program.

The below diagram shows this interaction:



Your calculator must have an object oriented design and be completely documented through Javadoc. You may write additional helper methods and design the calculator as you see fit, as long as the program meets the requirements above.

Strategies for success

- Remember that a polynomial can be represented as a collection of monomials.
- Start by implementing the `Poly` class, leaving the text-based interface for later. We recommend you begin with the polynomial constructor and `equals()` methods.
- Then implement the `toString()` method and print some polynomials to ensure that you can create polynomials correctly.
- Implement the arithmetic operations in the following order (testing each one as you implement it!): add, subtract, multiply, divide.
- You can use the Wolfram Alpha computational knowledge engine (`https://www.wolframalpha.com`) to cross-check your calculator's answers.
- In order to handle the space-separated input when writing the text-based interface, we recommend using two `Scanner` objects. The first `Scanner` will take input from `System.in` using the `nextLine()` method and get a line of input. The second `Scanner` will process that line of input to extract the individual coefficients and exponents using

the `nextDouble()` and `nextInt()` methods. The reason we recommend this strategy, is that it prevents the standard input scanner from unexpectedly blocking waiting for input when the user deviates from the intended input format.

Here is some code to get you started:

```
Scanner keybScanner = new Scanner(System.in); // scanner for System.in input
String line = keybScanner.nextLine(); // gets a line of input, e.g., "2.3 4"
Scanner lineScanner = new Scanner(line); // scanner to scan inside line
double firstCoef = lineScanner.nextDouble(); // gets first coefficient 2.3
```

Checkpoint

By April 15th, you must show a coach or instructor your code (during lab or coaching/office hours). At this point, you should have the polynomial constructor, `equals()`, `add()`, `subtract()`, `multiply()`, and `toString()` methods implemented and passing the respective unit tests. You should also have started working on `divide()`.

Submission

Once you are satisfied with your program, compress (tar, tar.gz or zip formats only, please) and submit your code for evaluation through the course's Moodle page. Your program will be evaluated on correctness, design, and documentation.