Polymonial – 100 course points

The purpose of this assignment is to broaden your understanding of the Linked List data structure.

Refer to our Programming Assignments FAQ for instructions on how to install VSCode, how to use the command line and how to submit your assignments.

Background

A polynomial may be represented using a linked list as follows: for every term in the polynomial there is one entry in the linked list consisting of the term's coefficient and degree. The entries are ordered according to ASCENDING values of degree, i.e. lowest degree term first, then next lowest degree term and so on, all the way up to the highest degree term. IMPORTANT: Zero-coefficient terms are NOT stored.

For example, the following polynomial (the symbol '^' is used to mean 'raised to the power'):

```
4x^5 - 2x^3 + 2x +3
```

can be represented as the linked list of terms:

```
(3,0) -> (2,1) -> (-2,3) -> (4,5)
```

where each term is a (coefficient,degree) pair.

Notes about representation:

- Terms are stored in ASCENDING *order* of degrees from front to *rear* in a non-circular linked list.
- Zero-coefficient terms are NOT stored.
- An EMPTY (zero) polynomial is represented by a linked list with NO NODES in it, i.e. referenced by NULL.
- Coefficients are real numbers
- Degrees are POSITIVE integers, except if there is a constant term, in which case the degree is zero.
- There will not be more than one term in the same degree.

If you do not represent all your polynomials (the initial inputs as well as those you get out of doing arithmetic on polynomials) as above, you will lose credit even if your results are mathematically correct.

Implementation

At the bottom of the Autolab assignment page, under **Attachments**, you will see a `polynomial_project.zip` file. Download and unzip in your computer. You will see there are:

- 2 directories/folders
  - **src**: contains the folder poly with `Node.java`, `Term.java`, `Polynomial.java`, and `Polytest.java`
  - **bin**: contains the folder poly with the class files after the assignment is compiled
- 4 files
  - `ptest1.txt`, `ptest2.txt`, `ptest1opp.txt`: input files containing polynomials (described in the **Running the Program** section below)
  - `Makefile`: used to automate building and executing the target program

You need to complete the implementation of the `Polynomial` class where indicated in the following methods:

Method	Grading Points
`evaluate`	20
`add`	40
`multiply`	40

You may use `Math` class methods as needed.

There is no formal requirement of efficiency for any of the methods. However, every test case run will be timed out after 3 seconds, which is plenty of time even for inefficient code. If your method is timed out on a test case you will get no credit for that test.

Note: You will get a zero if you use any other data structure (e.g. array/arraylist) *anywhere* in your implementation, for *any* reason, even if it has nothing to do with the actual polynomial operations. You must work with linked lists ONLY all the way through.

- Do not change `Node` and `Term` in any way. You will not be submitting them, and we will be using the original versions to test your `Polynomial` implementation.
- If you wish to change `Polytest`, feel free. You will not be submitting it, and we will not be using it to grade your `Polynomial` submission.

Observe the following rules while working on `Polynomial.java`:

- Only fill in the code in the methods `add`, `multiply`, and `evaluate` where indicated.
- In methods that return a `Polynomial` (`add` and `multiply`), the polynomial that is returned must be represented as described in the "Notes about representation" part of the **Background** section above.
  Your method will not get credit if the returned polynomial does not adhere to this representation, even it is mathematically correct.
  Also see the "Notes about empty (zero) polynomials" at the end of the **Running the program** section below.
- **DO NOT** remove the `package` line at the top of the file.
- **DO NOT** remove any of the `import` statements.
- **DO NOT** add any import statements.
- **DO NOT** change the headers of ANY of the given methods
- **DO NOT** change/remove any of the given class fields
- **DO NOT** add any new class fields – this includes variables and classes.
- **YOU MAY** add new helper methods, but you must declare them **private**.
- **DO NOT** use System.exit()

Before you submit, make sure to check your code against the original `Polynomial.java`, `Term.java`, and `Node.java` files to make sure you have adhered to the rules above.

Compiling and Executing

Once inside the polynomial directory, type:

- **`javac -d bin src/poly/*.java`** to compile
- **`java -cp bin poly.Polytest`** to execute

If you have a unix operating system (Linux or Mac OS) you CAN use the Makefile file provided. Once inside the polynomial directory type:

- **make** to compile
- **make run** to execute
- **make clean** remove all the .class files

Running the Program

There are three sample input files for you to test:

- A file `ptest1.txt` that contains the polynomial

  ```
  4x^5 - 2x^3 + 2x + 3
  ```

- A file `ptest2.txt` that contains the polynomial

  ```
  8x^4 + 4x^3 - 3x + 9
  ```

- A file `ptest1opp.txt` that contains the polynomial

  ```
  -4x^5 + 2x^3 - 2x - 3
  ```

  (the negation of the polynomial in `ptest1`)

In each of these files, each line is a term, with the first value being the coefficient, and the second value being the degree. The terms are listed in **descending** order of degrees and the respective non-zero coefficients. Remember that when you store a polynomial in a linked list, you will store it in **ascending** order of degrees. (This is actually already implemented by the Polynomial constructor when it reads a polynomial from an input file. All you have to do is make sure you stick with this rule when you add and multiply.)

You may assume that we will NOT test with an invalid polynomial file, i.e. every test input file will either have at least one term in the correct format, or will be empty (see **Notes about empty (zero) polynomials** below). So you don't need to check for validity of input.

Here's a sample run of the driver, `Polytest`. Apart from `ptest1.txt`, `ptest2.txt`, and `ptest1opp.txt`, a fourth test polynomial file, `ptestnull.txt` is also used. This is an empty file that stands for a null (zero) polynomial – you will need to create this yourself. See notes after the test run for special instructions regarding zero polynomials.

```
Enter the name of the polynomial file => ptest1.txt

4.0x^5 + -2.0x^3 + 2.0x + 3.0


1. ADD polynomial
2. MULTIPLY polynomial
3. EVALUATE polynomial
4. QUIT
        Enter choice # => 1
Enter the file containing the polynomial to add => ptest2.txt

8.0x^4 + 4.0x^3 + -3.0x + 9.0

Sum: 4.0x^5 + 8.0x^4 + 2.0x^3 + -1.0x + 12.0


1. ADD polynomial
2. MULTIPLY polynomial
3. EVALUATE polynomial
4. QUIT
        Enter choice # => 1
Enter the file containing the polynomial to add  => ptest1opp.txt

-4.0x^5 + 2.0x^3 + -2.0x + -3.0

Sum: 0


1. ADD polynomial
2. MULTIPLY polynomial
3. EVALUATE polynomial
4. QUIT
        Enter choice # => 1
Enter the file containing the polynomial to add  => ptestnull.txt

0

Sum: 4.0x^5 + -2.0x^3 + 2.0x + 3.0


1. ADD polynomial
2. MULTIPLY polynomial
3. EVALUATE polynomial
4. QUIT
        Enter choice # => 2
Enter the file containing the polynomial to multiply  => ptest2

8.0x^4 + 4.0x^3 + -3.0x + 9.0

Product: 32.0x^9 + 16.0x^8 + -16.0x^7 + -20.0x^6 + 52.0x^5 + 38.0x^4 + -6.0x^3 + -6.0x^2 + 9.0x + 27.0


1. ADD polynomial
2. MULTIPLY polynomial
3. EVALUATE polynomial
4. QUIT
        Enter choice # => 3
Enter the evaluation point x  => 2
Value at 2.0: 119.0


1. ADD polynomial
2. MULTIPLY polynomial
3. EVALUATE polynomial
4. QUIT
        Enter choice # => 4
```

The sample tests we have given you are just for starters. You will need to create other tests of your own on which you can run your code. For every test you run, be careful to keep your test input in the same format as the test files provided, otherwise `Polytest` will not work correctly. And make sure your test file is in the same folder as the other files, i.e. under `Polynomial`.

Note on translation from internal to output representation:

The `toString` method in the `Polynomial` class returns a string with the terms in descending order, fit for printing. (It processes the ascending ordered terms of the input linked list in reverse order.) For illustration, see how the `add`method in `Polytest` prints the resulting polynomial:

```
    System.out.println("Sum: " + Polynomial.toString(Polynomial.add(poly1,poly2)) + "\n");
```

Notes about empty (zero) polynomials:

- If you want to test with an empty polynomial input, you should create a file with nothing in it. In Eclipse, you can do this by right clicking on the project name in the package explorer view, then selecting **New**, then selecting **File**. Give a name, and click **Finish**. You new file will show up under the project name folder in the package explorer view, and the file will be opened in the text editor view. But don't type anything in the file.
- Remember that when you add two terms of the same degree, if you get a zero coefficient result term, it should not be added to the result polynomial. As listed in the "Notes about representation" in the **Background** section, zero-coefficient terms are not stored.
- The string representation of a zero polynomial is "0" – see the `toString` method of the `Polynomial` class. So, the `Polytest` driver will print a zero for a zero polyomial input, or a zero polynomial that results from an operation performed on two polynomials.

Before submission

4. *Collaboration policy.* Read our collaboration policy here.
5. *Submitting the assignment.* Submit *Polynomial.java* separately via the web submission system called Autolab. To do this, click the *Assignments* link from the course website; click the *Submit* link for that assignment.

Getting help

If anything is unclear, don't hesitate to drop by office hours or post a question on Piazza. Find instructors office hours by clicking the *Staff* link from the course website. In addition to office hours we have the CAVE (Collaborative Academic Versatile Environment), a community space staffed with lab assistants which are undergraduate students further along the CS major to answer questions.