Assignment Polynomials

Object Orientation

Spring 2021

1 Polynomials

Polynomials are common in all kinds of sciences such as mathematics, physics, and statistics. A polynomial consists of a sum of terms, where every term has the form cx^n. Factors c are nonzero real numbers. Exponents n are natural numbers. For example, $3x^4 - 4x^3 + 2x - 8$ is a polynomial.

2 Learning Goals

In this exercise you design classes that implement and test polynomials. After doing this exercise you should be able to:

- Use iterators to inspect and modify data structures.
- Write unit tests for your code.

3 Problem Sketch

Polynomials are represented by the class *Polynomial*, and terms by the class *Term*. Polynomials can have an arbitrary number of terms, which is why a polynomial is a list of terms. A polynomial should satisfy the following invariants.

- No term should have a factor 0.
- No two terms should have the same exponent.
- Terms should be stored in increasing order of their exponents.

Every method you write can rely on the invariants, and should produce polynomials that satisfy them.

To help you get started, we provide a template project with the classes *Polynomial* and *Term*. The class *Polynomial* has three constructors. One without arguments for creating an empty polynomial, a copy constructor, and a constructor that parses a string representation.

The string representation of a polynomial is an alternating sequence of factors and exponents. For example, the string `"3 0 0.5 1 -2 2 13.37 8"` represents the polynomial $3 + 0.5x - 2x^2 + 13.37x^8$. The parser for this format is given in the template, you do not have to implement it yourself.

The class *Polynomial* has stubs of the functions *equals*, *plus*, *minus*, *times*, and *divide*. These operators perform addition, subtraction, multiplication and division

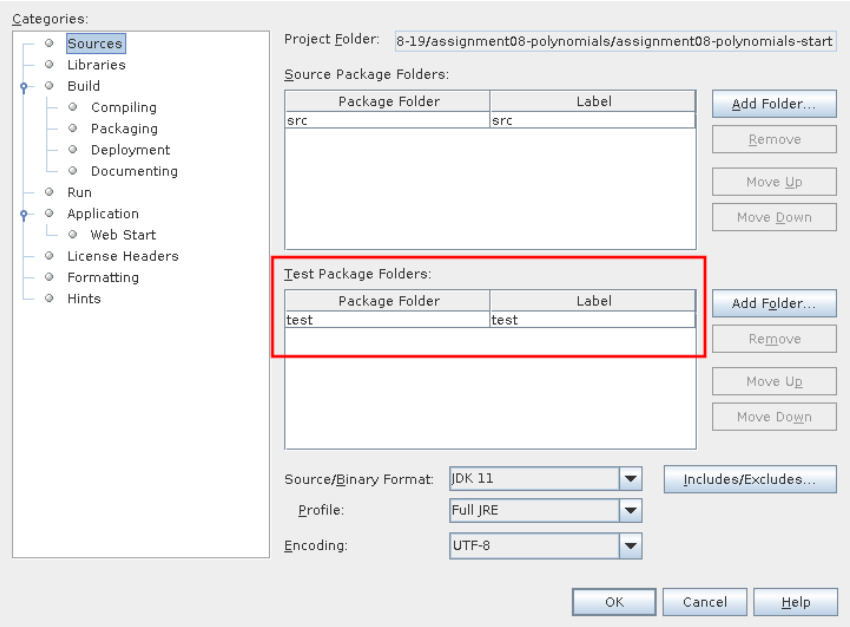
respectively. Your task is to implement these functions, except division. Division is only added for completeness, you are **not** expected to implement it.

The operations are symbolic manipulations. For example, $(2x^2+x^3)+(x-2x^2+x^3) = x+2x^3$.

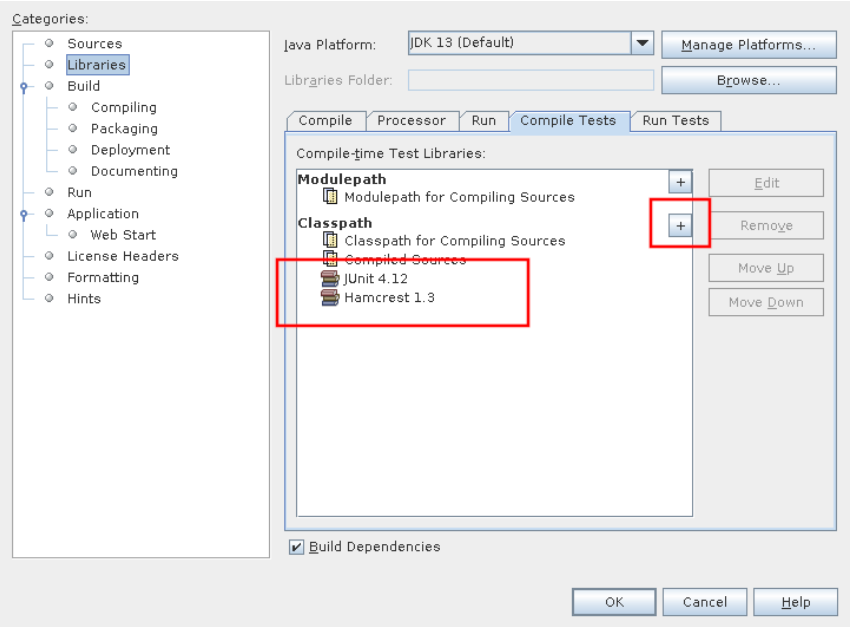
The implementation of the operators should be *destructive*. This means that they modify the polynomial on the left side. For example, `p1.add(p2)` modifies `p1` by adding `p2` to it.

4 Your Tasks

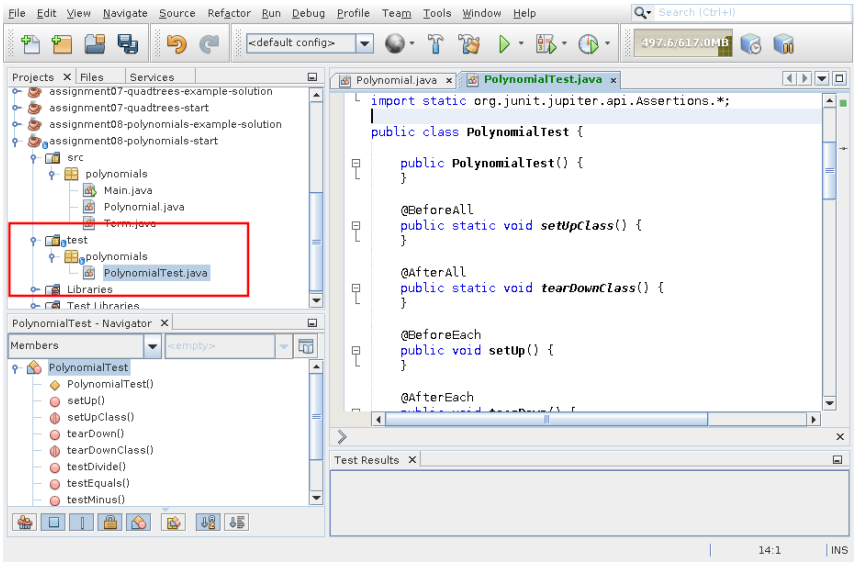
- Import the project template into NetBeans.
- Make sure that the directory *test* is set as a test package folder. To do so, open the Project Properties dialog by right-clicking on the project in NetBeans, then select the menu entry *Properties*. In the Project Properties dialog, make sure that *test* is added to the test package folders.



- Java is in the transition from JUnit 4 to JUnit 5, but JUnit 5 does not work properly in NetBeans. **Before** you generate a test class, add JUnit 4 to the build dependencies. This will cause NetBeans to generate test cases for JUnit 4. To do so, go to the Project Properties dialog, go to Libraries / Compile Tests, and remove the JUnit 5 library if it is there. Then click on the plus symbol at Classpath, and add the libraries Junit 4.12 and Hamcrest 1.3. Close the Project Properties dialog.



- Generate a test class for *Polynomial* by right-clicking on the project, then select the menu entry New / Other... / Unit Tests / Test for Existing Class. In the New Test dialog, select the class *Polynomial* as Class to Test.
  - This creates a package *polynomials* in the *test* folder, with a class *PolynomialTest* inside of it.
  - Your project should now look like in the following screenshot.



- Right-click on Polynomials.java and select Test File to see if the tests are running.
- In this assignment you should test the class *Polynomial*. Make sure that the test cases are in a class called *PolynomialTest*.
- Implement the following operations on Polynomial. Add helper functions when necessary.

  - The *toString* method that generates a text representation of a polynomial.
  - The *plus*, *minus* and *times* methods. Do not use indices to access the list of terms. Work with list iterators.
  - Add a method `evaluate(int x)` that calculates the value of a polynomial for a given argument. For example, the polynomial $p = 3x^4 - 4x^3 + 2x - 8$ will result in `p.evaluate(2)` $= 48 - 32 + 4 - 8 = 12$.
- Write test cases for every method.
  - Test the methods *plus*, *minus*, and *times* with a few example polynomials.
  - Pay attention to edge cases. For example, terms can disappear during addition because their factor becomes zero. Write test cases to check if your implementation takes this into account.
  - Implement a test case that checks subtraction is the same negated addition: `a - b == a + (-1 * b)`. Hint: -1 is `Polynomial("-1 0")`.
  - Test multiplying a polynomial with itself `p1.times(p1)`.
  - Test whether your operators are commutative, associative, and distributive.
    * Commutativity: $a + b = b + a$ and $a \times b = b \times a$
    * Associativity: $(a + b) + c = a + (b + c)$ and $(a \times b) \times c = a \times (b \times c)$
    * Distributivity: $a * (b + c) = (a * b) + (a * c)$

5 Submit Your Project

To submit your project, follow these steps.

1. Use the **project export** feature of NetBeans to create a zip file of your entire project: File → Export Project → To ZIP.
2. **Submit this zip file on Brightspace**. Do not submit individual Java files. Do not submit any other archive format. Only one person in your group has to submit it. Submit your project before the deadline, which can be found on Brightspace.