Data Structures
COMP 2100
Fall 2022

Lab 4: Working with Polynomials

In this lab you will implement and test the `Polynomial` class. A `Polynomial` object represents any polynomial in one variable, x, with integer coefficients. For example, here is a cubic polynomial:

$$f(x) = 4x^3 - x^2 + 3x + 5$$

This polynomial has four terms, each with an integer coefficient and integer exponent as shown in the table below.

Term	Coefficient	Exponent
$4x^3$	4	3
$-x^2$	–1	2
$3x$	3	1
5	5	0

Strategy

Your class should consist of a list of terms. You should interpret this as 'HAS-A', not 'IS-A'. You should use the `LinkedList` class from the JCF (java.util). See pages 3-4 of this document for implementation notes.

The API for `Polynomial` is **here** and the API for `Term` is **here**. I have completely implemented `Term` for you.

I am providing three files for this lab in the accompanying zip file.

1. `Term.java`, the completed, correct implementation of the `Term` class.
2. `TermPolyDriver.java`, a test driver. You can use this to test your `Polynomial` methods.
3. `PolynomialForDriver.java`, another Java file which is needed by the test driver to assure that methods in your solution can be tested in isolation from each other.

Assignment

- Your code must be stored in `Polynomial.java`
- Your code must contain Javadoc-style comments that reproduce as closely as you can the `Polynomial` API at the link above.
- Begin by created a stubbed-in version of Polynomial that will compile, but with default or dummy code for each method. This will allow the `TermPolyDriver` program to run without error.
- I encourage you to Implement methods one at a time, testing each one thoroughly as you go.

Remember that individual work is expected on the lab projects!

Points	Description
3	Comments including Javadoc
1	`Polynomial` no argument constructor
3	`Polynomial` one argument constructor
4	`Polynomial addterm()` method
3	`Polynomial evaluate()` method
5	`Polynomial sum()` method
5	`Polynomial product()` method
3	`Polynomial equals()` method
3	`Polynomial toString()` method

To Submit: Email your `Polynomial.java` source file as an attachment to dstucki@otterbein.edu

Implementation notes

1. Start by implementing `Polynomial` with stub versions of all its methods. This will allow you to quickly run the test driver and fail its tests! The `Polynomial` API has all the information you need. Constructors and methods declared void will do nothing; methods declared to return something will return a default value or null for objects.
2. Use the provided `Term` class in your implementation. The `Term` API is provided so you know how to use its methods.
3. The `Polynomial` class will not extend anything. Instead it will have one instance variable with this declaration:

   ```
   protected LinkedList<Term> poly;
   ```

4. The `LinkedList` class is, like `ArrayList`, part of the Java Collections Framework in the java.util package (you should look up its API online). Don't be concerned with how it is implemented, since you have your client hat on. The relationship between `Polynomial` and LinkedList is "has-a". A polynomial has-a linkedList.
5. You'll find the one-parameter constructor, once implemented, to be very handy to use for testing. The parameter is a `String` containing a list of *space-separated integers*. The integers are logically grouped in pairs. Each pair represents a term: the first number of the pair is the coefficient and the second is the exponent. For example the `String` `"3 2 -5 1 4 0"` represents polynomial $3x^2 - 5x + 4$. The trick to implementing this constructor is pulling out those values one by one and converting them from `String` to `int`! You will break it up into *tokens*, where each number represents a token, or item of interest. Here, the tokens are separated by spaces. The preferred (but not only) technique for doing this is to use the `split()` method from class `String`. You need to specify the delimiter (separator) as a *regular expression*. In this case, the regular expression should be the string `"\\s+"`, which will match one or more space characters. Elements of the resulting `String` array still need to be converted to integers. Remember how to do this? *Hint*: a static method in the `Integer` class.
6. Most of the methods require you to traverse through the terms of the polynomial. There is a very easy way to do this using an `Iterator`. `Iterator` is a java.util class. You can get an `Iterator` from `poly` (that's the instance variable), then use its `hasNext()` and `next()` methods together in a while loop to process the polynomial term by term. You might guess that `boolean` method `hasNext()` will let you know if there are more terms in the polynomial, and Object method `next()` returns the next term. These two collaborate to traverse the list something like this:

   ```
   Iterator<Term> iter = poly.iterator();  // start at position 0 of list
   while (iter.hasNext()) {
      Term term = iter.next();
      // do whatever processing you need to do with this term
   }
   ```

   Alternatively, you could use the foreach idiom in Java to accomplish the same thing like this:

   ```
   for (Term term : poly) {
      // do whatever processing you need to do with this term
   }
   ```

All of the `Polynomial` methods (aside from constructors) will do list traversals. Although everything that iterators do can also be done using `LinkedList` methods `get(int index)` and `size()`, iterators eliminate the need to work with indexes and are generally a cool thing to know.

7. The `toString()` method should produce the polynomial with terms ordered in decreasing order of exponent. For example: "+3x^2-x+2". Sorting these is very easy to do, thanks to a java.util class called `Collections` which provides many handy static methods that can be applied to an object of any of the Java collection classes. All you need in your program is the line:

```
Collections.sort(poly, Collections.reverseOrder());
```

Interesting fact: The sort method determines order by using the Term class compareTo() method!

Hint: You can simplify implementation of `toString()` by calling the `toString()` method from the `Term` class on each of its terms.

Hint: You can simplify implementation of the `equals()` method by using `toString()`.