Assignment 2

CS 1027
Computer Science Fundamentals II

Due date: Wednesday, October 25 at 11:55 pm

Learning Outcomes

In this assignment, you will gain experience with:

- Programming according to specifications
- Creating a linked list class
- Creating and using exceptions

Introduction: Polynomials

Polynomials are functions in a single variable x, that is, they take a value for a variable x and produce a result. A **polynomial** is formed by using exponents, multiplication, and addition. The general form of a polynomial is

$$f(x) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x^1 + a_0$$

where n is an integer (called the degree), $a_0, a_1, a_2, \ldots, a_n$ are the coefficients, which are fixed numbers, and x is the variable. For example, $f(x) = 3x^2 + 2x + 2$ is a polynomial of degree 2. In our assignment, our polynomials will have integer coefficients only. We always have that a_n is not equal to zero (since otherwise we don't write that term and use the highest expression whose coefficient is not zero.

We call each of the terms $a_i x^i$ in a polynomial a **monomial**. For instance, in $f(x) = 6x^4 + 2x^2 + 2$, the three monomials are $6x^4$, $2x^2$ and 2. We do not include in our expression for a polynomial any of the monomials where the coefficient is zero.

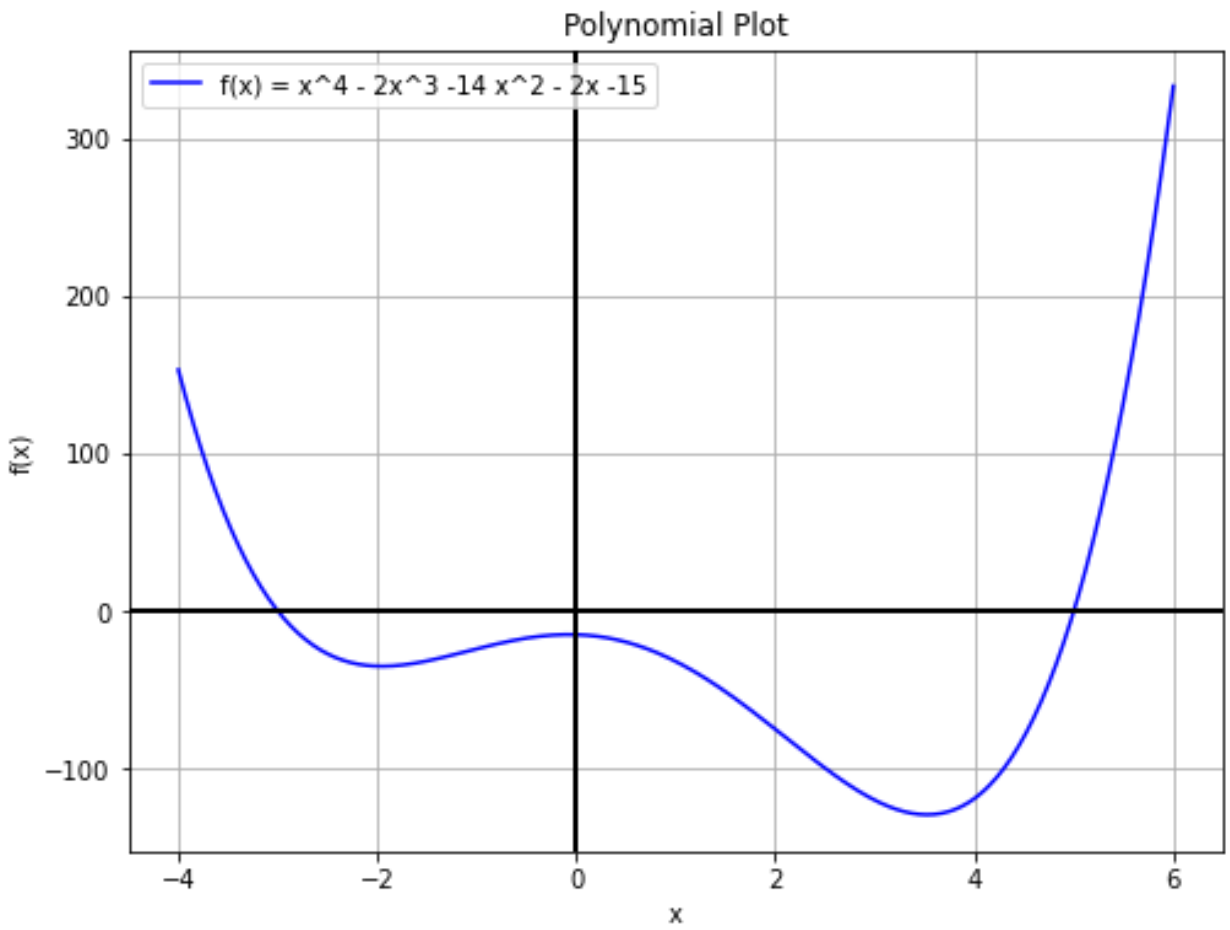
We **evaluate** a polynomial by giving a value to x. For instance, if $f(x) = 3x^2 + 2x + 2$ then $f(3) = 3(3)^2 + 2(3) + 2 = 27 + 6 + 2 = 35$.

We **solve** a polynomial $f(x)$ by finding values of x such that $f(x) = 0$. For polynomials of small degree there are exact formulas for solving polynomials (you may remember the quadratic formula for solving polynomials of degree two). But for general polynomials, no such method exists. A solution for a polynomial $f(x)$ is called a **root** of the polynomial. For instance, for $f(x) = 16x^7 - 384x^6 + 2512x^5 - 1188x^4 - 17549x^3 + 21921x^2 + 4230x^1 - 5400$, the value $x = 1.5$ is a root, which you can verify by evaluating the polynomial at $x = 1.5$. Note that even though our polynomial has integer coefficients, the roots of the polynomial may be floating point numbers.

As another example, here is a plot of the polynomial $f(x) = x^3 - 2x^2 + x^1 + 1$. The plot shows values of x on the bottom and values of $f(x)$ on the left side. The values of the function at any value of x can be seen as the position of the blue plot above the corresponding value of $f(x)$ (so, for example, by looking at the plot we can estimate that the value of $f(x)$ at $x = 2$ is approximately -80; by evaluating the polynomial, we can see that it is exactly $f(x) = -75$. You can see from the plot that the polynomial $f(x)$ has roots at $x = -3$ and $x = 5$, as these are the points where $f(x)$ has the value zero (i.e., crosses the bold line at $f(x) = 0$ on the left hand side).

Assignment 2

CS 1027
Computer Science Fundamentals II



To solve polynomials of higher degree, we will use **Newton's method** in this assignment. Newton's method is an iterative process where the derivative of the polynomial is used to find values that are closer and closer to a root of the polynomial. Newton's method allows us to find one root of the polynomial. While other roots to the polynomial may exist, we will only find one.

To use Newton's method, we need the **derivative** of a polynomial. The derivative of a polynomial

$$f(x) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x^1 + a_0$$

is the polynomial

$$f'(x) = (na_n)x^{n-1} + ((n-1)a_{n-1})x^{n-2} + ((n-2)a_{n-2})x^{n-3} + \cdots + 2a_2 x^1 + a_1$$

For instance, the derivative of $f(x) = 6x^4 + 2x^2 + 2x + 3$ is $f'(x) = 24x^3 + 4x + 2$. Notice that for any polynomial, if there is a final monomial in the polynomial with exponent zero (i.e., a_0) then that term is never included in the derivative.

Newton's method starts with an initial guess of a solution x_0. We can use any guess for x_0, but some guesses will work better than others. (Some values will not work at all, and could cause

Newton's method to crash or go into an infinite loop.) After we choose our initial guess, we find another value that is closer to the solution of the polynomial: if x_i is our current solution, then we let

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

where $f'(x)$ is the derivative of $f(x)$. This process creates a sequence of values $x_0, x_1, \ldots, x_i$. We **stop** the process when we find x_i and x_{i+1} that are very close together.

In this assignment, you will implement a linked list class to store polynomials and use Newton's method to solve these polynomials.

Example of Newton's method

Let x_0 be the polynomial $f(x) = 16x^7 - 384x^6 + 2512x^5 - 1188x^4 - 17549x^3 + 21921x^2 + 4230x^1 - 5400$.

Then the derivative of $f(x)$ is $f'(x) = 112x^6 - 2304x^5 + 12560x^4 - 4752x^3 - 52647x^2 + 43842x^1 + 4230$.

Let's find a root starting with the initial guess of $x_0 = 0$. Then we calculate the next value as $x_1 = x_o - \frac{f(x_0)}{f'(x_0)}$.

In this case, x_0 is 0, $f(0)$= -5400 and $f'(0)$= 4230. So

$$x_1 = 0 - (-5400/4230) = 1.276595$$

(All values are approximations). To calculate the next value of x_2, we use $x_1 = 1.276596$, and then $f(x_1) = 3002.63404$), and $f(x_1) = -9455.21826$. Then

$$x_2 = 1.276596 - (3002.63404/-9455.21826) = 1.59415.$$

Continuing this way, we can calculate further values of x_i.

Iteration	x_{i-1}	x_i	Value of $f(x_i)$
1	0	1.276595	3002.6270002
2	1.276595	1.5941577	-1737.9010444
3	1.5941577	1.5058949	-101.53315372
4	1.5058949	1.5000297	-0.5095444527
5	1.5000297	1.5000000007695344	-1.3186736396E-5
6	1.5000000007695344	1.5000000000000004	-5.4569682106E-12
7	1.5000000000000004	1.5000000000000002	8.18545231595E-12

At this point, the difference between x_i and x_{i-1} is very small, and Newton's method stops. We can also see that $f(x_i)$ is very close to zero. (In this table, numbers like 8.18545231595E-12

represent 8.18545231595 * 10^{-12}, which in this case is very close to zero.) Thus, Newton's method has determined that the value of approximately x_7=1.5 is a root of the polynomial.

Classes to Implement

For this assignment, you must implement five Java classes:

- A Node Class
- An Ordered Linked List Class
- A Monomial Class.
- A Polynomial Class
- A SolutionNotFound Exception Class

In these classes, you may implement more private (helper) methods if you want. However, you may not implement more **public** methods **except** `public static void main(String[] args)` for testing purposes (this is allowed and encouraged).

You may **not** add instance variables other than the ones specified in these instructions nor change the variable types or accessibility (i.e., making a variable `public` when it should be `private`). Penalties will be applied if you implement additional instance variables or change the variable types or modifiers from what is described here.

You may **not** import any Java libraries such as `java.util.Arrays` or `java.util.ArrayList`.

Node Class

You must implement a Node class, which should be generic.

The `Node` class must have the following **private** instance variables:

1. An instance variable `data` of type `T` (the generic type).
2. An instance variable `next` of type `Node<T>`.

The `Node` class must have the following **public** methods:

1. A constructor that takes both data and next parameters.
2. `getNext` and `getData` accessor methods.
3. `setNext` mutator method, which takes a parameter of type `Node<T>`.

Ordered Linked List Class

You should implement a linked list class that allows ordered insertion. This means that your linked list class should take an element to insert and find the proper location for that element in the linked list. This should use a `compareTo` method, as described at the end of this section.

Assignment 2

CS 1027
Computer Science Fundamentals II

You should call your class `OrderedLinkedList`. The class should be generic. However, to ensure that what we are storing in the linked list has a `compareTo` method, we will modify our declaration of the class as follows:

```
public class OrderedLinkedList<T extends Comparable<T>> {
```

By doing this, we are saying "the type T that is stored in this list must be **Comparable** (i.e., it has a `compareTo` method)". You should use this line to declare you `OrderedLinkedList` class. We will see more on `Comparable` later in the course.

Your ordered linked list class must have the following **private** instance variables:

1. A `Node<T>` called `head`, to store the start of the linked list.
2. An integer `size`, which keeps track of the size of the linked list.

Your linked list class must have the following public methods:

1. A constructor that accepts no parameters and creates an empty linked list.
2. An `insert` method that accepts a single parameter of type `T` (the generic type) and has void return type. The method inserts the value into the list in the proper position, as described below.
3. A `get` method that takes a single integer parameter `i` returns the i-th element in the list (of type `T`). The method should be zero-indexed (i.e., `get(0)` should return the first element in the list). If the i-th element in the list does not exist, the method should throw an `IndexOutOfBoundsException` (which is built-in in Java).
4. A `getSize` method that returns the number of elements in the linked list.

Your insert method should add elements to the list, so that they are maintained in order from **largest-to-smallest.** To implement the insert method, you need to calculate the proper position of the element in the Ordered List. This will require you to compare elements that are in the list with the element that is being inserted into the list: as you traverse the list, you should compare elements to the element that is being inserted to determine which is larger and stop when you find the right spot.

Because of the declaration of our `OrderdLinkedList` class with generics, we have ensured that anything stored in the list will have a `compareTo` method. You can use this to determine the order of elements in the list. If two elements x and y are in your list and x.compareTo(y) > 0 then x should come before y in the list.

When inserting into your linked list, you should use the compareTo method to determine the position of the element. If you are inserting an element x into the linked list, use x.compareTo(y) where y are data elements that are already in the list to determine where to insert.

Monomial Class

Create a class called `Monomial` that stores a monomial. A monomial must have a **private** integer `coefficient` instance variable and a **private** integer `degree` instance variable. For instance, if you were storing the monomial $6x^4$, the coefficient would be 6 and the degree would be 4.

Assignment 2

CS 1027
Computer Science Fundamentals II

To be able to use the `OrderedLinkedList` class properly, you need to define the `Monomial` class as having a `compareTo` method. You do so by declaring the class as

```
public class Monomial implements Comparable<Monomial> { … }
```

Implement the following **public** methods for the Monomial class:

1. a constructor for the class that accepts both coefficient and degree instance variables,
2. accessors (but not mutators) for both instance variables.
3. a `compareTo` method that accepts a Monomial as a parameter and returns an integer. It should determine which Monomial has the higher degree. This method should have the following form (which you can paste into your solution):

```
public int compareTo(Monomial m) {
            return this.degree - m.degree;
}
```

Polynomial class

Using your `OrderedLinkedList` class, create a `Polynomial` class. This class will use a linked list to store a polynomial. The class should have one **private** instance variable:

- An `OrderedLinkedList` that stores Monomials. When declaring this instance variable, let the generic type be Monomial.

Your Polynomial class should have the following **public** methods:

- A constructor that accepts no parameters, which creates an empty polynomial (i.e., one with no monomials).
- A method `add`, which takes **two integer** parameters (a coefficient and a degree) and adds a new monomial to the polynomial. The method's return type is **void**. You may assume that a Monomial of the same degree does **not** already exist in the Polynomial and that the degree is greater than or equal to zero.
- A method called `derivative`, which takes no parameters, and returns a **new** Polynomial object. The new polynomial is the derivative of the polynomial whose `derivative` method was called (i.e., the derivative of `this`). As noted above, the derivative of a polynomial

  $$f(x) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x^1 + a_0$$

  is the polynomial

  $$f'(x) = (na_n)x^{n-1} + ((n-1)a_{n-1})x^{n-2} + ((n-2)a_{n-2})x^{n-3} + \cdots + 2a_2 x^1 + a_1$$

- A method called `eval` that evaluates the polynomial. The method takes a **double** parameter `z` and return the value `f(z)` (also a **double**) where `f` is the polynomial stored in the class.
- A `toString` method that gives a string representation of all nonzero monomials in the polynomial. The method should take no parameters and return a **String**. In particular, the representation of the polynomial should be in decreasing order of degree. The operator (+ or -) between monomials should be separated from each monomial by a

single space. There should only be an operator before the initial monomial if it is a minus sign (and is not separated by a space from the coefficient in this case). Each monomial should be written without spaces and should include a power of x even if that power is zero.

For example, the String representation of the polynomial $f(x) = 16x^7 - 384x^6 + 2512x^5 - 1188x^4 - 17549x^3 + 21921x^2 + 4230x^1 - 5400$ is

```
16*x^7 - 384*x^6 + 2512*x^5 - 1188*x^4 - 17549*x^3 + 21921*x^2 + 4230*x^1 - 5400*x^0
```

If the polynomial has no monomials, the toString method should return the empty string. Always display the number 1 if it appears as a coefficient (so the representation of the polynomial $f(x) = -x^2 - x + 1$ should be `-1*x^2 - 1*x^1 + 1*x^0`).

- A `solve` method, which is described in the Newton's Method section below. The method takes thee parameters (two **doubles** and an **integer**) and returns a **double**.

Note that your Polynomial class **must not** be a child of the OrderedLinkedList class.

SolutionNotFound Exception

You must implement a class called `SolutionNotFound` which is a subclass of `Exception`. The class should have a message parameter to its constructor, which should be passed to the constructor of `Exception`. The class does not need any other methods.

Newton's Method

You should implement a public method called `solve` in your Polynomial class to implement Newton's method. The `solve` method takes three parameters:

1. `x0`, a double. This represents the ***initial value*** for the solution search.
2. `e`, a double value representing the ***tolerance*** of the solution. When the absolute value of the difference between two successive approximations of the root is less than e, then Newton's method stops, and a solution is found.
3. `T`, an integer value representing the ***maximum number of iterations*** that Newton's method runs for. If the number of iterations exceeds T, then the method is not successful.

The `solve` method should implement Newton's method as described in the Introduction, or the pseudocode provided at the end of this section. Here are additional details on starting and stopping the method:

1. Start the process at a value of x_0 that is provided as a parameter to the method.
2. Stop this procedure if any of three different conditions occurs:

a. If $|x_{i+1} - x_i| < e$ for a small value e (which is a parameter to the method), then we stop: in this case, we have found an approximate root of the polynomial, which is x_{i+1}. Return this value.
b. If $f'(x_{i+1}) = 0$ at any step, then we stop **before** we get a divide by zero error, and no solution is found. In this case, raise a `SolutionNotFound` exception with the message "divide by zero error" (without quotes).
c. If we don't find a solution after T iterations, we stop and no solution is found. In this case, we raise a `SoutionNotFound` exception with the message "maximum iteration exceeded" (without quotes).

The `solve` method should return a double, representing the root of the polynomial. It should also raise the SoultionNotFound exception in the cases 2b and 2c above. The entire process can be summarized with the following pseudocode.

```
previous = initial value (x0)
if derivative at previous is not zero:
	current = Newton's formula using previous
else:
	raise exception
while (maximum number of iterations is not reached)
	and (absolute difference between previous and current > tolerance):
	previous = current
	if derivative at previous is not zero:
		current = Newton's formula using previous
	else:
		raise exception
	increment number of iterations

if maximum number of iterations was reached:
	raise exception
else:
	return current
```

In this pseudocode, *Newton's formula* refers to the formula given previously:

$$x_{i+1} = x_i - f(x_i)/f'(x_i)$$

Provided files

TestOLL.java is a tester file to **help** check if your Ordered Linked List class is implemented correctly.

TestNewton.java is a tester file to **help** check if your Polynomial class is implemented correctly.

Similar files will be incorporated into Gradescope's auto-grader. Additional tests will be run that will be hidden from you. **Passing all the tests within the provided files does not necessarily mean that your code is correct in all cases.**

Assignment 2

CS 1027
Computer Science Fundamentals II

Marking Notes

Functional Specifications

- Does the program behave according to specifications?
- Does it produce the correct output and pass all tests?
- Are the classes implemented properly?
- Does the code run properly on Gradescope (even if it runs on Eclipse, it is **up to you** to ensure it works on Gradescope to get the test marks)
- Does the program produce compilation or run-time errors on Gradescope?
- Does the program fail to follow the instructions (i.e. changing variable types, etc.)

Non-Functional Specifications

- Are there comments throughout the code (Javadocs or other comments)?
- Are the variables and methods given appropriate, meaningful names?
- Is the code clean and readable with proper indenting and white-space?
- Is the code consistent regarding formatting and naming conventions?
- Submission errors (i.e. missing files, too many files, etc.) will receive a penalty.
- Including a "package" line at the top of a file will receive a penalty.

Remember **you must do** all the work on your own. **Do not copy** or even look at the work of another student. All submitted code will be run through similarity-detection software.

Submission (due Wednesday, October 25 at 11:55 pm)

Assignments must be submitted to Gradescope, not on OWL. If you are new to this platform, see these instructions on submitting on Gradescope.

Rules

- Please only submit the files specified below.
- Do not attach other files even if they were part of the assignment.
- Do not upload the .class files! Penalties will be applied for this.
- Submit the assignment on time. Late submissions will receive a penalty of 10% per day.
- Forgetting to submit is not a valid excuse for submitting late.
- Submissions must be done through Gradescope. **If your code runs on Eclipse but not on Gradescope, you will NOT get the marks! Make sure it works on Gradescope to get these marks.**
- You are expected to perform additional testing (create your own test harness class to do this) to ensure that your code works for a variety of cases. We are providing you with some tests but we may use additional tests that you haven't seen for marking.
- Assignment files are NOT to be emailed to the instructor(s) or TA(s). They will not be marked if sent by email.
- You may re-submit code as many times as you wish, however, re-submissions after the assignment deadline will receive a late penalty.

Assignment 2

CS 1027
Computer Science Fundamentals II

Files to submit:

- Node.java
- OrderedLinkedList.java
- Monomial.java
- Polynomial.java
- SolutionNotFound.java

Grading Criteria

Total Marks: 20

15 marks	Autograder Tests (some tests are hidden from you)
5 marks	Non-functional Specifications (code readability, comments, correct variables and functions, etc.)