CS 1027A - Assignment 2 - Polynomials

● Graded

Student

Mohammed Ali Abdul-nabi

Total Points

20 / 20 pts

Autograder Score

15.0 / 15.0

Passed Tests

[---------- TEST 01 (Newton) ----------] (0.5/0.5)
[---------- TEST 02 (Newton) ----------] (0.5/0.5)
[---------- TEST 03 (Newton) ----------] (0.5/0.5)
[---------- TEST 04 (Newton) ----------] (0.5/0.5)
[---------- TEST 05 (Newton) ----------] (0.5/0.5)
[---------- TEST 06 (Newton) ----------] (0.5/0.5)
[---------- TEST 07 (Newton) ----------] (0.5/0.5)
[---------- TEST 08 (Newton) ----------] (0.5/0.5)
[---------- TEST 09 (Newton) ----------] (0.5/0.5)
[---------- TEST 10 (Newton) ----------] (0.5/0.5)
[---------- TEST 11 (Newton) ----------] (0.5/0.5)
[---------- TEST 12 (Newton) ----------] (0.5/0.5)
[---------- TEST 13 (Newton) ----------] (0.5/0.5)
[---------- TEST 14 (Newton) ----------] (0.5/0.5)
[---------- TEST 15 (Newton) ----------] (0.5/0.5)
[---------- TEST 01 (OLL) ----------] (0.5/0.5)
[---------- TEST 02 (OLL) ----------] (0.5/0.5)
[---------- TEST 03 (OLL) ----------] (0.5/0.5)
[---------- TEST 04 (OLL) ----------] (0.5/0.5)
[---------- TEST 05 (OLL) ----------] (0.5/0.5)

Question 2

Code Logic

1 / 1 pt

✔ **– 0 pts** Correct - Meaningful variable names, private instance variables used

– 0.5 pts Click here to replace this description.

– 1 pt Wrong - No meaningful logic

Question 3

Code Formatting/Readability

2 / 2 pts

✔ **– 0 pts** Correct

– 0.5 pts Click here to replace this description.

– 1 pt Click here to replace this description.

– 1.5 pts Click here to replace this description.

– 2 pts No proper code formatting. Code not readable

Question 4

Comments

2 / 2 pts

✔ **– 0 pts** Correct - Comments are proper and relevant

– 0.5 pts Click here to replace this description.

– 1 pt Click here to replace this description.

– 2 pts Wrong - Comments are NOT proper and relevant or/and no comments included.

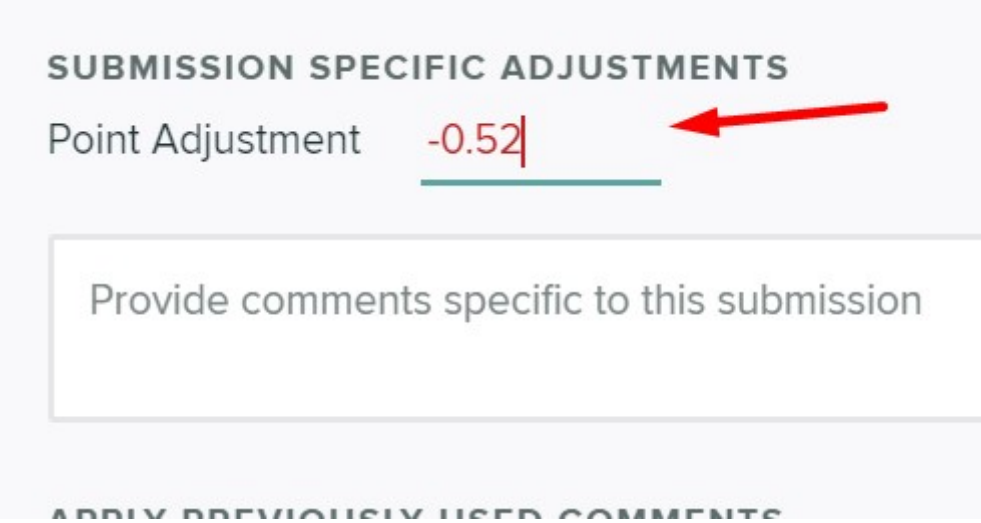
Question 5

Penalties **0** / 0 pts

5.1 *Late Submissions* -2/day **0** / 0 pts

✔ **– 0 pts** Click here if not late.

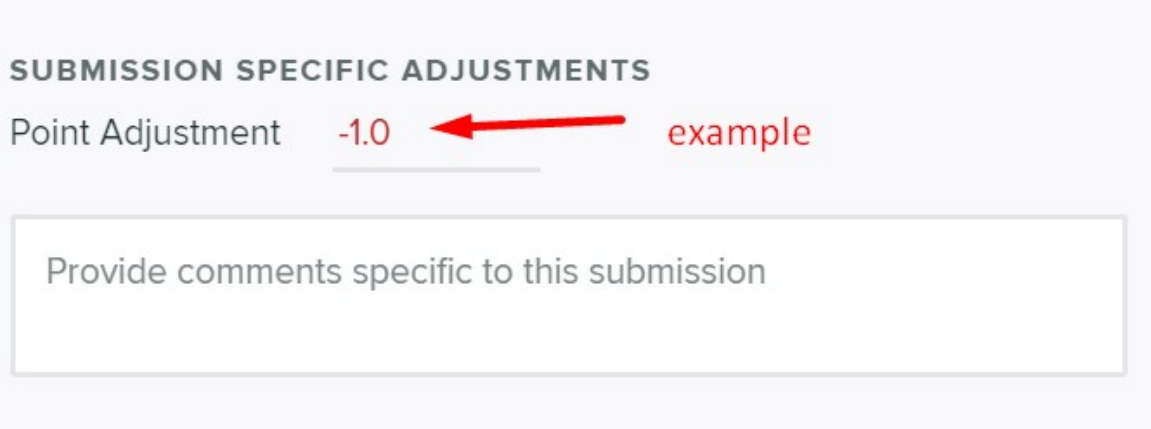
– 0 pts **@TAs: DO NOT ADD YOUR OWN RUBRICS HERE** *Please enter the deduction in the* **Point Adjustment** *field below if late penalty applies.*



5.2 Incorrect submission (doesn't compile, package line, .class file, etc.) -5 **0** / 0 pts

✔ **– 0 pts** Click here if no submission error

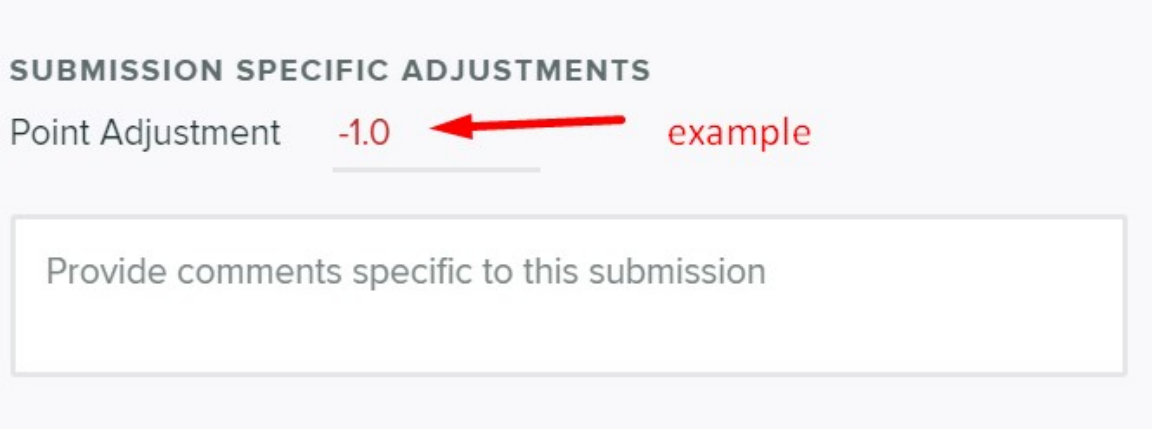
– 5 pts **@TAs: DO NOT ADD YOUR OWN RUBRICS HERE** *Please enter the deduction in the* **Point Adjustment** *field below if submission incorrect* Example:



5.3 **Incorrect instance variables or methods -2** **0** / 0 pts

✔ **– 0 pts** Click here if no submission error

– 2 pts **@TAs: DO NOT ADD YOUR OWN RUBRICS HERE** *Please enter the deduction in the* **Point Adjustment** *field below if submission incorrect* Example:



Autograder Results

[---------- TEST 01 (Newton) ----------] (0.5/0.5)

[---------- TEST 02 (Newton) ----------] (0.5/0.5)

[---------- TEST 03 (Newton) ----------] (0.5/0.5)

[---------- TEST 04 (Newton) ----------] (0.5/0.5)

[---------- TEST 05 (Newton) ----------] (0.5/0.5)

[---------- TEST 06 (Newton) ----------] (0.5/0.5)

[---------- TEST 07 (Newton) ----------] (0.5/0.5)

[---------- TEST 08 (Newton) ----------] (0.5/0.5)

[---------- TEST 09 (Newton) ----------] (0.5/0.5)

[---------- TEST 10 (Newton) ----------] (0.5/0.5)

[---------- TEST 11 (Newton) ----------] (0.5/0.5)

[---------- TEST 12 (Newton) ----------] (0.5/0.5)

[---------- TEST 13 (Newton) ----------] (0.5/0.5)

[---------- TEST 14 (Newton) ----------] (0.5/0.5)

[---------- TEST 15 (Newton) ----------] (0.5/0.5)

[---------- TEST 01 (OLL) ----------] (0.5/0.5)

[---------- TEST 02 (OLL) ----------] (0.5/0.5)

[---------- TEST 03 (OLL) ----------] (0.5/0.5)

[---------- TEST 04 (OLL) ----------] (0.5/0.5)

[---------- TEST 05 (OLL) ----------] (0.5/0.5)

Submitted Files

Monomial.java Download

```java
public class Monomial implements Comparable<Monomial> {
    private int coefficient;
    private int exponent;

    public Monomial(int coefficient, int exponent) { // Initializes the coefficient and exponent of the monomial.
        this.coefficient = coefficient;
        this.exponent = exponent;
    }
    public int getCoefficient() {               // Getter method to retrieve the coefficient.
        return coefficient;
    }

    public int getExponent() {                  // Getter method to retrieve the exponent.
        return exponent;
    }

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

Node.java

Download

```java
public class Node<T> {
    private T data;
    private Node<T> next;

    public Node(T data, Node<T> next) {          // Initializes data and the next node.
        this.data = data;
        this.next = next;
    }

    public T getData() {          //Getter to get data in the node.
        return data;
    }

    public Node<T> getNext() {      //Getter to get next in node.
        return next;
    }

    public void setNext(Node<T> next) { //Setter to set the next node.
        this.next = next;
    }
}
```

OrderedLinkedList.java

Download

```java
public class OrderedLinkedList<T extends Comparable<T>> {
    private Node<T> head;
    private int size;

    public OrderedLinkedList() {       // Initializes an empty linked list
        this.head = null;
        this.size = 0;
    }

    public int getSize() {
        return size;
    }

    public void insert(T element) {     //Inserts the element into the list from largest to smallest.
        Node<T> newNode = new Node<>(element, null);

        if (head == null || element.compareTo(head.getData()) > 0) {
            newNode.setNext(head);  //Insert the start if empty or if element is greater than head.
            head = newNode;
        } else {                    //Go through the list to find the appropriate place to place element.
            Node<T> current = head;
            while (current.getNext() != null && element.compareTo(current.getNext().getData()) < 0) {
                current = current.getNext();
            }
            newNode.setNext(current.getNext());     //Place it when appropriate place is found.
            current.setNext(newNode);
        }
        size++;                             //Increase list size
    }

    public T get(int index) {
        if (index < 0 || index >= size) {             // Checks if the index is out of bounds
            throw new IndexOutOfBoundsException("Index is out of bounds: " + index);
        }

        Node<T> current = head;
        for (int i = 0; i < index; i++) {          //Goes through the list to find correct i
            current = current.getNext();
        }
        return current.getData();                  //Returns the "i-th" element in the list
    }
}
```

Polynomial.java Download

```java
public class Polynomial {
    private OrderedLinkedList<Monomial> terms;

    public Polynomial() {                                   //Creates an empty polynomial.
        terms = new OrderedLinkedList<>();
    }

    public void add(int coefficient, int degree) {              //Takes two inputs to create monomial.
        if (coefficient != 0) {
            Monomial monomial = new Monomial(coefficient, degree);
            terms.insert(monomial);                         //Adds monomial to the empty polynomial.
        }
    }

    public Polynomial derivative() {                        //Finds the derivative of the polynomial.
        Polynomial derivative = new Polynomial();
        for (int i = 0; i < terms.getSize(); i++) {             //Gets terms (degree and coef) for each monomial.
            Monomial crntMono = terms.get(i);
            int crntCoef = crntMono.getCoefficient();
            int crntDegr = crntMono.getExponent();

            if (crntDegr >= 0) {                            //Does the math for finding derivative.
                int newCoef = crntCoef * crntDegr;
                int newDegr = crntDegr - 1;
                derivative.add(newCoef, newDegr);
            }
        }
        return derivative;
    }

    public double eval(double z) {                          //Calculates the value of a polynomial at F(Z).
        double result = 0.0;
        for (int i = 0; i < terms.getSize(); i++) {
            Monomial crntMono = terms.get(i);
            double coefficient = crntMono.getCoefficient();
            int degree = crntMono.getExponent();
            result += coefficient * Math.pow(z, degree);        //Coefficient * input z raised to the
corresponding degree
        }
        return result;
    }

    @Override
    public String toString() {
        if (terms.getSize() == 0) {                 //Returns empty string if polynomial is empty.
            return "";
```

```
46          }
47          StringBuilder sb = new StringBuilder();
48
49          for (int i = 0; i < terms.getSize(); i++) {
50              Monomial currentMonomial = terms.get(i);
51              int coefficient = currentMonomial.getCoefficient();
52              int degree = currentMonomial.getExponent();
53
54              if (i == 0) {                       // Keeps the negative sign if the first coefficient is negative.
55                  if (coefficient < 0) {
56                      sb.append(coefficient);
57                  } else {
58                      sb.append(coefficient);
59                  }
60              } else {                            // For all other monomials
61                  if (coefficient < 0) {
62                      sb.append(" - ").append(-coefficient);      // Add "-" operator between each monomial if coef
    is -ve.
63                  } else {
64                      sb.append(" + ").append(coefficient);       // Add "+" operator between each monomial if coef
    is +ve.
65                  }
66              }
67              sb.append("*x^").append(degree);           //Adds the degree and x^ to the coefficient
68          }
69
70          return sb.toString();
71      }
72
73
74      public double solve(double x0, double e, int T) throws SolutionNotFound {
75          double previous = x0;                       //Initial guess
76
77          if (derivative().eval(previous) != 0) {             // Check if the derivative at the initial estimate is not
    zero.
78              double current;                         //Stores the value of x0+1 after its calculated.
79              current = previous - eval(previous) / derivative().eval(previous);  //Newtons method at x0+1.
80              int iterations = 0;                     //Store current iterations.
81
82              while (iterations < T && Math.abs(current - previous) > e) {    //When Difference is greater than
    tolerance and Max iterations not reached.
83                  previous = current;
84                  if (derivative().eval(previous) != 0) {                 //When the derivative x1+ is not zero.
85                      current = previous - eval(previous) / derivative().eval(previous);  //Find derivative at next x
    value.
86                  } else {
87                      throw new SolutionNotFound("divide by zero error");     //If it is zero throw error.
88                  }
89
```

```
90              iterations++;                       //Increase iterations by 1 each loop.
91          }
92
93          if (iterations >= T) {                  //If max number of iterations is reached we throw error.
94              throw new SolutionNotFound("maximum iteration exceeded");
95          } else {                        //Returns answer for when we have reached max iterations and
    tolerance is within range
96              return current;
97          }
98      } else {                            //Throws error for when derivative is zero.
99          throw new SolutionNotFound("divide by zero error");
100     }
101   }
102
103 }
```

SolutionNotFound.java Download

```
1  public class SolutionNotFound extends Exception {
2      public SolutionNotFound(String message) {
3          super(message);
4      }
5  }
```