Data Structures
Program for Arithmetic with Polynomials

Midterm Individual Exercise 1 (Application of the LinkedList Data Structure)

Objective:
To create an application that uses the LinkedList Data Structure (i.e. LinkedList Data Type in the Java Collections Framework)

Required:
Create an application that may be used as an aid in doing the following:\

1. Evaluating a polynomial involving one literal coefficient
2. Adding polynomials involving one literal cofficient.
3. Subtracting polynomials involving one literal coefficient
4. Multiplying polynomials involving one literal coefficient
5. Dividing polynomials involving one literal coefficient.

Do the application by improving and organizing the following programs in one project using the IntelliJ IDEA IDE.
Name of Project:
<Yourname><ClassCode><CourseNumber>MidtermIndvidualProject1
e.g. DelacruzJuan9300CS211MidtermIndividualProject1

Submit/Upload an archived project through the Submission Bin in the Google Classroom for this class.
e.g. DelacruzJuan9300CS211MidtermIndividualProject1.zip

/*
Name of Student:

Date:
*/

```
/**
 * The following class is a template for a term of an algebraic polynomial that involves only one literal.
 *  3x^2 is an example of a term
 *    where 3 is the numerical coefficient,
 *     x is the literal coefficient and
 *     2 is the degree
 */
public class Term implements Comparable<Term> {
   private double coefficient;   //data member to hold numerical coefficient of a term
   private int degree;  // data member to hold the degree of a term
   private char literal; // data member to hold the literal coefficient of a term

   /**
    * Constructs a term with coefficient set to 0, degree set to 0 and literal set to x.
    */
   public Term() {
      coefficient = 0;
      degree = 0;
```

```
    literal = 'x';
}

/**
 * Constructs a term that sets the coefficient, literal and degree
 * to given the coef, literal and degree
 */
public Term(double coef, char literal, int degree) {
    this.coefficient = coef;
    this.literal = literal;
    this.degree = degree;
}
// sample use
// Term t = new Term(2.0, 'x', 3)

/**
 * Sets the value of the numerical coefficient of this term to the specified coef
 */
public void setCoefficient(double coef) {

    this.coefficient= coef;
}
// sample use
// t.setCoefficient(3)

/**
 * Sets the value of the literal coefficient of this term to the specified character literal
 */
public void setLiteral(char literal){
    this.literal = literal;
}

// t.setLiteral('y')
/**
 * Sets the value of the degree of this term to a specified degree
 */
public void setDegree(int degree) {
    this.degree = degree;
}

/**
 * Returns the numerical coefficient of this term
 */
public double getCoefficient() {

    return this.coefficient;
}

// System.out.println("Coefficient=" + t.getCoefficient())

/**
 * Returns the literal coefficient of this term
 */
```

```
public char getLiteral() {
   return this.literal;
}

/**
 * Returns the degree of this term
 */
public int getDegree() {
   return this.degree;
}

/**
 Returns 0 if the degree of this term is equal to the degree of another term
 else returns an integer that is greater than 0 if the degree of this term is
 greater than the degree of another term else returns an integer that is lesser than
 0 if the degree of this term is lower than the degree of another term.
 */
public int compareTo(Term another){
   if (this.getDegree() == another.getDegree())
      return 0;
   else
   if (this.getDegree() > another.getDegree())
      return -1;
   else
      return 1;

   //sample use
   // return this.toString().compareTo(another.toString());
}

// Term t1 = new Term()
// Term t2 = new Term(1,'x', 1)
// ArrayList<Term> p = new ArrayList();
// ...
// p has been assigned several terms
// Collections.sort(p)

/**
 Returns a string representation of the term
 that follows a format with the example 3x^2

 */
public String toString() {
   if (coefficient == 0 )
      return "";
   else
   if (coefficient==1 && degree == 1)
      return  ""+literal;
   else
   if (coefficient==1 && degree != 1)
      return ""+literal+"^"+degree;
   else
      return (coefficient+literal+"^"+degree);
```

```
  }
}
```

```
public class Quotient {
  Polynomial quotientP;
  Polynomial remainderP;

  public Quotient(){
    quotientP = null;
    remainderP = null;
  }

  public void setQuotientP(Polynomial q){
    quotientP = q;
  }

  public void setRemainderP(Polynomial p){
    remainderP = p;
  }

  public Polynomial getQuotientP() {
    return quotientP;
  }

  public Polynomial getRemainderP() {
    return remainderP;
  }

  public String toString() {
    return (" Quotient: " + quotientP.toString() + "  Remainder: " + remainderP.toString());
  }
}
```

```
/*
Name of Student:

Date:
*/

import java.util.LinkedList;  // access the LinkedList class from package util

/**
 * Template for a polynomial
 */
public class Polynomial {
  private LinkedList<Term> terms;  // data member to reference a
                    // LinkedList representing a polynomial
```

```java
/**
 * Constructs a LinkedList for Terms that is initially empty
 */
public Polynomial() throws Exception{
   terms = new LinkedList<Term>();
}

/**
 *Adds a Term to the polynomial such that the terms are arranged following a decreasing
 * order of degrees.
 *This method inserts a Term in the polynomial at the appropriate location if the degree of the term
 * is not equal to a degree of an existing Term.
 *If the degree of the Term is equal to a degree of an existing Term,  the coefficient of the
 * existing Term is updated by adding the coefficient of the Term being added. If the updated coefficient
 * equals zero, the Term is removed from the  polynomial.
 */
public void addTerm(Term newTerm) throws Exception {
   int ctr;
   boolean found = false;
   Term currTerm = null;

   for (ctr = 0; ctr < terms.size(); ctr++) { // size method of LinkedList is used
      currTerm =  terms.get(ctr);    // get method of LinkedList is used.
      if (currTerm.getDegree() <=  newTerm.getDegree()) {
         found = true;
         break;
      }
   }

   if (!found) {
      terms.add(newTerm);// add method of LinkedList is used.
   } else {
      if (currTerm.getDegree() < newTerm.getDegree()) {
         terms.add(ctr, newTerm); // alternative add method of LinkedList is used
      } else {
         currTerm.setCoefficient(currTerm.getCoefficient() + newTerm.getCoefficient());
         if (currTerm.getCoefficient() == 0) {
            terms.remove(ctr); // remove method of the LinkList class is used
         }
      }
   }
}

/**
 * Returns a string representing this polynomial with the following  sample form 3x^2 - 5X + 3
 */
public String toString() {
   String s = "";
   if (terms == null)
            return " ";

   for (int ctr = 0; ctr < terms.size(); ctr++) {
      Term currTerm = terms.get(ctr);
```

```
      if (currTerm.getCoefficient() > 0) {
        if (ctr != 0) {
          s = s + " +";
        }
      } else {
        s = s + " -";
      }

      if (currTerm.getCoefficient() != 1 || currTerm.getDegree() == 0) {
        s = s+" "+ Math.abs(currTerm.getCoefficient());
      }

      switch (currTerm.getDegree()) {
        case 0 :
          break;
        case 1 :
          s = s + (terms.get(0)).getLiteral();
          break;
        default :
          s = s + (terms.get(0)).getLiteral() +"^" + currTerm.getDegree();

      }
    }
    return s;
}

/**
 * Returns the value of this polynomial if its literal is substituted
 *   by the specified given value.
 */
public double evaluate(double value) throws Exception{
    double sum = 0;
    for (int ctr = 0; ctr < terms.size(); ctr++) {
      Term currTerm =  terms.get(ctr);
      sum += currTerm.getCoefficient() * Math.pow(value, currTerm.getDegree());
    }
    return sum;
}

/**
 * Sets this polynomial to a given LinkedList of Term
 */
public void setTerms(LinkedList<Term> t){
    terms = t;
}

/**
 *Returns this polynomial as a LinkedList of Term
 */
public LinkedList<Term> getTerms() {
    return terms;
}
```

```
  /**
   * public Polynomial add(Polynomial otherPolynomial) throws Exception
   * Adds  otherPolynomial to this polynomial
   1. Declare (let) result as a Polynomial that will eventually represent the sum polynomial
   2. Construct a new LinkedList of Term(i.e. resultTerms) that will eventually hold
       the Terms of the sum Polynomial
   3. Construct a copy of each term of this (first) Polynomial
      and add the constructed Term to the constructed LinkedList (resultTerms)
   4. Let resultTerms be the terms of result(i.e. the sum polynomial)
   5. For each (get each) term of the other polynomial, construct a copy
      of the term and assign such copy to sTerm.
   6. Add sTerm to the sum polynomial (result)
   7.If result polynomial has no term, let result have the term 0x^0
   8. return result.

   */
  public Polynomial add(Polynomial otherPolynomial) throws Exception {
     Polynomial result = new Polynomial();
     LinkedList<Term> resultTerms= new LinkedList<Term>();

     for (int ctr = 0; ctr < this.getTerms().size(); ctr++) {
        Term currentTerm =  this.getTerms().get(ctr);
        resultTerms.add(new Term(currentTerm.getCoefficient(), currentTerm.getLiteral(),
currentTerm.getDegree()));
     }

     // Put missing statement here

     for (int ctr2 = 0; ctr2 < otherPolynomial.getTerms().size(); ctr2++) {
        Term currentTerm = otherPolynomial.getTerms().get(ctr2);
        Term sTerm = new Term(currentTerm.getCoefficient(), currentTerm.getLiteral(), currentTerm.getDegree());
        result.addTerm( sTerm);
     }

     if (result.getTerms().size()==0)
        result.addTerm(new Term(0,'x',0));
     return result;
  }


  /**
   * Subtracts otherPolynomial from this polynomial
   *
   1. Declare (let) result as a Polynomial that will eventually represent the difference polynomial
   2. Construct a new LinkedList of Term(i.e. resultTerms) that will eventually hold
   the Terms of the difference Polynomial
   3. Construct a copy of each term of this (first) Polynomial
   and put the constructed Term to the constructed LinkedList (resultTerms)
   4. Let resultTerms be the terms of result(i.e. the sum polynomial)
   5. For each (get each) term of the other polynomial, construct a copy of the term
      and assign such copy to sTerm.
   6. Multiply the numerical coefficient field of sTerm by -1.
   7. Add sTerm to the difference polynomial (result)
```

```
 8.If result polynomial has no term, let result have the term 0x^0
 9. return result.

 */
public Polynomial subtract(Polynomial otherPolynomial) throws Exception {
   Polynomial result = new Polynomial();
   LinkedList<Term> resultTerms= new LinkedList<Term>();

  // to do

   return result;
}

/**
 * Multiplies this polynomial by otherPolynomial.
 *    The method assumes that the polynomials have the same literals and
 *    it follows the prescription of the Term class.
 */
public Polynomial multiply(Polynomial otherPolynomial) throws Exception
{
   Polynomial result=new Polynomial();

   for (int ctr = 0; ctr < this.getTerms().size(); ctr++) {
      Term currentTerm1 = this.getTerms().get(ctr);

      for (int ctr2 = 0; ctr2 < otherPolynomial.getTerms().size(); ctr2++) {
         Term currentTerm2 =  otherPolynomial.getTerms().get(ctr2);

         double pCoef = currentTerm2.getCoefficient()*currentTerm1.getCoefficient();
         int pDegree = currentTerm2.getDegree()+currentTerm1.getDegree();

         result.addTerm(new Term(pCoef, currentTerm1.getLiteral(), pDegree));
         // Invoke appropriate method to add a term to product polynomial
      } // end of second for ( for ctr2)

   } // end of first for  (for ctr)

   if (result.getTerms().size() == 0)
      result.addTerm(new Term(0,'x',0));

   return result;
}


/**
 * Divides this polynomial by divisor polynomial
 */
public Quotient divide(Polynomial divisor) throws Exception
{
   Quotient result=new Quotient();
   Polynomial quotient= new Polynomial();
   Polynomial remainder = new Polynomial();
```

```
      LinkedList<Term> dividend = new LinkedList<Term>();
      Term qTerm;
      Polynomial subtrahend = new Polynomial();

      for (int ctr = 0; ctr < this.getTerms().size(); ctr++) {
         Term currentTerm = this.getTerms().get(ctr);
         dividend.add(new Term(currentTerm.getCoefficient(), currentTerm.getLiteral(), currentTerm.getDegree()));
      }

      remainder.setTerms(dividend);

      while (((remainder != null)) && ((remainder.getTerms().get(0)).getDegree() >= (
divisor.getTerms().get(0)).getDegree())) {

         Term numTerm =  remainder.getTerms().get(0);
         Term   divTerm =  divisor.getTerms().get(0);

         qTerm = new Term (numTerm.getCoefficient()/divTerm.getCoefficient(), numTerm.getLiteral(),
numTerm.getDegree()-divTerm.getDegree());

         quotient.addTerm(qTerm);

         LinkedList<Term> pQTerm = new LinkedList<Term>();
         pQTerm.add(qTerm);

         Polynomial multiplier = new Polynomial();
         multiplier.setTerms(pQTerm);

         subtrahend = multiplier.multiply(divisor);
         remainder = remainder.subtract(subtrahend);

      }

      if (quotient.getTerms().size()==0)
         quotient.addTerm(new Term(0,'x',0));

      result.setQuotientP(quotient);

      if (remainder.getTerms().size()==0)
         remainder.addTerm(new Term(0,'x',0));

      result.setRemainderP(remainder); // Invoke appropriate method to set remainder member of quotient

      return result;

   }
}
```

```
/*
```

Name of Student:

Date:

```
*/

public class TestPolynomial {
   public static void main(String[] args) {
      try {
         Polynomial p = new Polynomial();
         p.addTerm(new Term(1,'x',2));
         p.addTerm(new Term(4, 'x', 3));
         p.addTerm(new Term(-3, 'x', 1));
         p.addTerm(new Term(1, 'x', 0));

         Polynomial other = new Polynomial();
         other.addTerm(new Term(2,'x',1));
         other.addTerm(new Term(-1,'x',2));

         System.out.println("Sample Polynomial: " + p.toString());
         System.out.println("Value of the sample polynomial at x=2: " + p.evaluate(2));

         System.out.println("\nExample of addition of polynomials");
         Polynomial sP = p.add(other);
         System.out.println("(" + p.toString() + ") + (" + other.toString() + ") = " + sP.toString());

         System.out.println("\nExample of subtraction of polynomials");
         Polynomial dP = p.subtract(other);
         System.out.println("(" + p.toString() + ") - (" + other.toString() + ") = " + dP.toString());

         System.out.println("\nExample of multiplication of polynomials");
         Polynomial pP = p.multiply(other);
         System.out.println("(" + p.toString() + ") * (" + other.toString() + ") = " + pP.toString());

         System.out.println("\nExample of division of polynomials");
         Quotient qP = p.divide(other);
         System.out.println("(" + p.toString() + ") / (" + other.toString() + ") = " + qP.toString());

      } catch (Exception e) {
         e.printStackTrace();
      } // end of catch block

   } // end of main

} // end of TestPolynomial class


---------------------------------------------------------------------------------------------------
/*
Name of Student:

Date:
*/

import java.util.*;
```

```
public class PolynomialArithmetic {
   Scanner keyboard = new Scanner(System.in);
   public void run() throws Exception{
      byte choice = 0;
      while ( choice != 6 ) {
         showMenu();
         choice = readChoice((byte) 1, (byte) 6);
         switch (choice){
            case 1:
               evaluatePolynomial();
               break;
            case 2:
               addPolynomials();
               break;
            case 3:
               System.out.println("to do...");
               break;
            case 4:
               System.out.println("to do...");
               break;
            case 5:
               System.out.println("to do...");
               break;
            case 6:
               System.out.println("Thank you for using this program.");
         } // end of swicth
      }  // end of while
   } // end of run

   private byte readChoice(byte low, byte high) throws Exception{
      byte choice=0;
      System.out.print("Enter your choice<"+ low + "... " + high + ">: ");
      choice = (byte) readInteger(low, high);
      return choice;
   }

   public void showMenu() {
      System.out.println("----------------------MENU-------------------------");
      System.out.println("1. Evaluate a polynomial");
      System.out.println("2. Add two polynomials");
      System.out.println("3. Subtract a polynomial from another polynomial");
      System.out.println("4. Multiply two polynomials");
      System.out.println("5. Divide a polynomial by another polynomial");
      System.out.println("6. Quit");
      System.out.println("---------------------------------------------------");
   }

   public void evaluatePolynomial() throws Exception{
      System.out.println("You want to evaluate a polynomial.");
      Polynomial p = readPolynomial();
      System.out.println("The polynomial entered is " + p.toString());
      System.out.print("What is the value to be assigned to variable of the polynomial? ");
      double value= readDouble();
```

```
    System.out.println("The polynomial evaluates to : "+ p.evaluate(value));
    System.out.println("Press enter to continue.....");
    keyboard.nextLine();
}

private int readInteger(int low, int high){
    boolean validInput = false;
    int value=0;
    while (!validInput){
        try{
            value = Integer.parseInt(keyboard.nextLine());
            if ( value < low){
                System.out.print("The number must not be lower than "+ low + ". ");
            }
            else if ( value > high){
                System.out.print("The number must not be greater than "+ high +". ");
            } else {
                validInput = true;
            }

        } catch (Exception x){
            System.out.println("You have to enter an integer from " + low + " to " + high + ".");
        }
    }
    return value;
}

private double readDouble(){
    boolean validInput = false;
    double value=0;
    while (!validInput){
        try{
            value = Double.parseDouble(keyboard.nextLine());
            validInput = true;
        } catch (Exception x){
            System.out.println("You have to enter a number.");
        }
    }
    return value;
}

public Polynomial readPolynomial() throws Exception{
    Polynomial p = new Polynomial();
    int degree=-1;
    boolean validDegree = false;
    char literalCoefficient = 'x';
    System.out.println("The polynomial should involve one variable/literal only.");
    do {
        System.out.print("What is the literal coefficient of the polynomial in one variable? ");
        literalCoefficient = keyboard.nextLine().charAt(0);
    } while (!Character.isAlphabetic(literalCoefficient));

    do {
```

```
        System.out.print("What is the degree of the polynomial? ");
        degree = readInteger(Integer.MIN_VALUE,Integer.MAX_VALUE);
        validDegree = true;
      } while (!validDegree);

      for (int x=degree; x>=0; x=x-1){
        Term term = readTerm(literalCoefficient, x);
        p.addTerm(term);
      }

      return p;
    }


    public Term readTerm(char literal, int degree)throws Exception{
      double nCoeff=0;

      System.out.print("Enter the numerical coefficient of the term with degree " + degree +":  ");
      nCoeff = readDouble();
      Term term = new Term(nCoeff, literal, degree);

      return term;
    }

    public void addPolynomials() throws Exception {
      System.out.println("You want to add two polynomials.");
      System.out.println("Enter the first polynomial.");
      Polynomial p1 = readPolynomial();

      System.out.println("Enter the second polynomial.");
      System.out.println("Note that the second variable should have the same variable/literal as the first
polynomial.");
      Polynomial p2 = readPolynomial();

      System.out.println("First polynomial : " + p1.toString());
      System.out.println("Second polynomial : " + p2.toString());

      if (p1.getTerms().get(0).getLiteral() == p2.getTerms().get(0).getLiteral()) {
        System.out.println("Sum of the polynomials : " + p1.add(p2));
      } else {
        System.out.println("The two polynomials cannot be added because they have different literals.");
      }

      System.out.println("Press enter to continue.....");
      keyboard.nextLine();
    }

    public static void main(String[] args) {
      PolynomialArithmetic program;
      try {
        program = new PolynomialArithmetic();
        program.run();
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    } // end of main

} // end of PolynomialArithmetic class
```