CECS 274
Project 2: Polynomial Class

Faith Yap
013257658

Due Date: Monday, September 28, 2015 @ 6:30 pm
Electronic PDF due in DropBox Folder
Hardcopy PDF in Lab Session

Meets Course Goals:

- Design an Abstract Data Type, covert it to Java class and then implement and use the class in a program of moderate complexity.
- Proper Javadoc documentation of Java programs

Instructions

Design and create a class called Polynomial that

1. constructs a polynomial object...
   (a) ...with the array of coefficients input by the user as an explicit parameter (assume the array is in descending order of terms). For example,

   ```
   double[ ] coeff = {2, 4, 5, 1};
   Polynomial h = new Polynomial(coeff);
   ```

   (b) ...without any explicit parameters. For example,
   `Polynomial f = new Polynomial( );`
2. has methods to access...
   (a) the degree of the polynomial.
   (b) the coefficient of a degree i term.
   (c) all the coefficients.
   (d) the string representation of the polynomial expression.

(e) the value of the polynomial evaluated at a given value.

3. has methods to...
   (a) set the coefficients of the polynomial.
   (b) edit the coefficient of a degree i term.
   (c) delete a term.
   (d) insert a term.

Grading Criteria

You will be graded on the following criteria:

1. Does the demo program (coming soon) run correctly? (20 pts.)

2. Does the class declare the appropriate instance variables? (5 pts.)

3. Does the class contain the appropriate constructors? (10 pts.)

4. Does the class define all the specified methods? (40 pts.)

5. Is the class documented using the Javadoc utility, including descriptions for every instance method, explicit parameter, etc.? (10 pts.)

6. Does the class design violate the rule for minimizing dependencies? (e.g. uses System.out., Math, etc.)? (-5 pts. for every class used)

Polynomial Class Code

Insert your code here.

```
/**
    * This program is meant to perform several functions involving
        polynomials,
    * including accessing, altering, and evaluating the polynomial.
    * @author Faith Y.
    * @version 1.0
*/

import java.util.ArrayList;
public class Polynomial
{
    private ArrayList<Double> coeff;
    /**
     * Constructor method: declares the ArrayList to be used. */
    public Polynomial()
    {
        coeff = new ArrayList<Double>();
    }
    /**
     * Loaded constructor method: instantiates values of ArrayList to be
         used
     * @param coefficients ArrayList containing the coefficients to be
         used */
    public Polynomial(ArrayList<Double> coefficients)
    {
        coeff = coefficients;
    }
    /**
     * Instantiates the values of the ArrayList to be used throughout
         the program.
     * @param coefficients ArrayList containing the coefficients to be
         used. */
    public void setAllCoeff(ArrayList<Double> coefficients)
    {
        coeff = new ArrayList<Double>(coefficients);
    }
    /**
     * Creates a String variable that holds the polynomial value.
     * @return the polynomial expression as a String */
    public String getPolyExpr()
    {
        String expression = "";
        if (coeff.isEmpty())
        {
            expression = "Coefficients have not been set.";
            return expression;
```

```
}
for (int i = 0; i < coeff.size(); i++)
{
    if (i == 0)
    {
        if (coeff.get(i) == -1)
        {
            expression += "-x^" + ((coeff.size()-i)-1);
        }
        else if (coeff.get(i) == 1)
        {
            expression += "x^" + ((coeff.size()-i)-1);
        }
        else if (coeff.get(i) > 1 || coeff.get(i) < -1)
        {
            expression += coeff.get(i) + "x^" +
                ((coeff.size()-i)-1);
        }
        else if (coeff.get(i) < -1)
        {
            expression += coeff.get(i) + "x^" +
                ((coeff.size()-i)-1);
        }
        else if (coeff.get(i) == 0)
        {
            expression = "There is no expression.";
        }
    }
    else if (i == coeff.size()-1)
    {
        if (coeff.get(i) == 0)
        {
            continue;
        }
        if (coeff.get(i) > 0)
        {
            expression += " + " + coeff.get(i);
        }
        if (coeff.get(i) < 0)
        {
            expression += " " + coeff.get(i);
        }
    }
    else if (i > 0)
    {
        if (coeff.get(i) == -1)
        {
            expression += " - x^" + ((coeff.size()-i)-1);
        }
        else if (coeff.get(i) == 1)
```

```
            {
                expression += " + x^" + ((coeff.size()-i)-1);
            }
            else if (coeff.get(i) > 1)
            {
                expression += " + " + coeff.get(i) + "x^" +
                    ((coeff.size()-i)-1);
            }
            else if (coeff.get(i) < -1)
            {
                expression += coeff.get(i) + "x^" +
                    ((coeff.size()-i)-1);
            }
            else if (coeff.get(i) == 0)
            {
                continue;
            }
        }
    }
    return expression;
}
/**
 * Adds a new degree and coefficient into the polynomial expression
 * @param degree integer value holding the degree of x it goes to
 * @param coefficient double value holding the coefficient x is to
     be multiplied to */
public void insertDeg(int degree, double coefficient)
{
    // If the degree to be inserted is the highest degree there will
        be...
    if (degree > coeff.size()-1)
    {
        for (int i = coeff.size(); i < degree; i++)
        {
            coeff.add(0, 0.0);
        }
        coeff.add(0, coefficient);
    }
    else if (degree <= coeff.size()-1)
    {
        coeff.set((coeff.size()-degree)-1, coefficient);
    }
}
/**
 * Access the coefficient value when given a certain degree
 * @param degree integer value holding the degree of x it goes to
 * @return double value holding the coefficient x is to be
     multiplied to */
public double getCoeff(int degree)
{
```

```
    return coeff.get((coeff.size()-1)-degree);
}
/**
 * Access the highest degree existing in the polynomial expression
 * @return double holding the highest degree in the expression
 */
public double getPolyDeg()
{
    double degree = coeff.size()-1;
    return degree;
}
/**
 * Access all the coefficient values present in the polynomial
     expression
 * @return arraylist consisting of the coefficient values in the
     expression in form of doubles
 */
public ArrayList<Double> getAllCoeff()
{
    return coeff;
}
/**
 * Replaces the coefficient value of the provided degree
 * @param degree integer value holding the degree of x it goes to
 * @param double value holding the coefficient x is to be multiplied
     to
 */
public void setCoeff(int degree, double coefficient)
{
    coeff.set((coeff.size()-degree)-1, coefficient);
}
/**
 * Sets the coefficient of a given degree to zero, giving the
     illusion that it was removed
 * @param degree integer value holding the degree of x it goes to
 */
public void removeDeg(int degree)
{
    coeff.set(coeff.size()-degree-1, 0.0);
}
/**
 * Evaluates the mathematical answer of the polynomial at a given x
     value
 * @param x number that the polynomial is to be evaluated at
 * @return total of the expression once evaluated at the given x
     value
 */
public double eval(double x)
{
    double evaluated = 0;
```

```
        for (int i = 0; i < coeff.size(); i++)
        {
            double xDegree = 1.0;
            for (int j = 0; j < (coeff.size()-i)-1; j++)
            {
                xDegree *= x;
            }
            evaluated += coeff.get(i) * xDegree;
        }
        return evaluated;
    }
}
```