HW #8

Name: Zhenzhen Su

Date: 4/10/2017

Course: CSC&143

Instructor: Dr. Charles Reid

Problem 1

Problem statement:

Add accessor methods which are manhattanDistance, isVertical, slope and isCollinear to Point class

Code:

```
public class Point {
    int x;
    int y;

    public Point(int initX, int initY){
        x = initX;
        y = initY;
    }

    public int getX(){
        return this.x;
    }

    public int getY(){
        return this.y;
    }



    /*It measures manhattan distance between two point objects
      input: Point object
      output: integer distance
     */
    public int manhattanDistance(Point other){
        int xlength = Math.abs(this.x - other.getX());
        int ylength = Math.abs(this.y - other.getY());
        return xlength + ylength;
    }


    /*Determine if this point aligns vertically with given point
      input: Point object
      output: boolean
     */
    public boolean isVertical (Point other){
        if(this.x == other.getX()){
            return true;
```

```
    }else{
        return false;
    }
}

/*Calculate slope of two points
  input: Point object
  output: double
 */
public double slope (Point other){
    if(other.getX() == this.x){
        throw new IllegalArgumentException();
    }
    return (double)(other.getY()-this.y)/(other.getX()-this.x);
}


/*Determine if this point is collinear with two given points
  input: two Point object
  output: boolean
 */
public boolean isCollinear(Point p1, Point p2){
    double slope1;
    double slope2;
    double slope3;

    //check x axis of this point and p1
    if(this.x != p1.getX()){
        slope1 = this.slope(p1);
        //check x axis of this point and p2
        if(this.x != p2.getX()){
            slope2 = this.slope(p2);

            if(slope1 != slope2){
                return false;
            //check x axis of p1 and p2
            }else if(p1.getX() != p2.getX()){
                slope3 = p2.slope(p1);
                if(slope3 == slope2){
                    return true;
                }else{
                    return false;
                }
            //handle special case of x axis of p1 equals to that of p2
            }else if(p1.getY() == p2.getY()){
                return true;
            }else{
                return false;
            }


        //handle special case when x axis of this point equals to p2's
        }else if(this.y == p2.getY()){
            return true;
        }else{
            return false;
        }
```

```
         //handle special case when x axis of this point equals to p1
        }else if(this.x == p2.getX()){
            return true;
        }else if(this.y == p1.getY()){
            return true;
        }else{
            return false;
        }
    }


}


public class Main {

    public static void main(String[] args) {
       // problem #3, #4, #5, #6, #14, #15, #16
        Point point = new Point(5,7);

        //calculate manhattan distance
        System.out.println("The Manhattan distance of point(1,6) is " +
                point.manhattanDistance(new Point(1,6)));

        //determine if two points are aligned vertically
        System.out.println("Whether point(4,3) and point(5,7) is aligned vertically: "
+
                point.isVertical(new Point(4,3)));

        //calculate slope of two points
        System.out.println("The slope between point(4,2) and point(5,7) is " +
                point.slope(new Point(4,2)));


        //determine if three points are collinear
        Point p1 = new Point(2,5);
        Point p2 = new Point(7,4);
        System.out.println("Whether point(2,5) and point(7,4) is collinear: " +
                point.isCollinear(p1,p2));
    }

}
```

Console output:

The Manhattan distance of point(1,6) is 5

Whether point(4,3) and point(5,7) is aligned vertically: false

The slope between point(4,2) and point(5,7) is 5.0

Whether point(2,5) and point(7,4) is collinear: false

Problem 2

Problem statement:

Create Line class and add accessor methods which are getP1, getP2, toString

Add method getSlope, construct a line based two points' x, y coordinates.

Code:

```
public class Line {
   Point p1;
   Point p2;

   //Constructor that takes in point objects
   public Line(Point p1,Point p2){
      this.p1 = p1;
      this.p2 = p2;
   }

   //Constructor that takes integers of x,y coordinates
   public Line(int x1, int y1, int x2, int y2){
      p1 = new Point(x1,y1);
      p2 = new Point(x2,y2);
   }

   //Get this Line's first endpoint
   public Point getP1(){
      return this.p1;
   }

   //Get this line's second endpoint
   public Point getP2(){
      return this.p2;
   }

   //return a string representation of the line
   public String toString(){
      return "[(" + this.getP1().getX() + ", " + this.getP1().getY() + "), (" +
           this.getP2().getX() + ", " + this.getP2().getY() + ")]";
   }

   //return slope of line
   public double getSlope(){
      return p1.slope(p2);
   }

}
```

```
public class Main {

    public static void main(String[] args) {
        //determine if three points are collinear
        Point p1 = new Point(2,5);
        Point p2 = new Point(7,4);
        System.out.println("Whether point(2,5) and point(7,4) is collinear: " +
                point.isCollinear(p1,p2));
```

```
        //create a line and print it in string
        Line line = new Line(p1,p2);
        System.out.println("The string representation of line that passes " +
                "point(2,5) and point(7,4) is " + line);
        System.out.println("The slope of the above line is " + line.getSlope());

        //create a line using a different constructor
        Line line1 = new Line(2,4,7,4);
        System.out.println("The line that used different constructor is printed as " +
line);

    }
}
```

Console output:

The string representation of line that passes point(2,5) and point(7,4) is [(2, 5), (7, 4)]

The slope of the above line is -0.2

The line that used different constructor is printed as [(2, 5), (7, 4)]

Problem 3

Problem statement:

Utilizes an array to store (double) polynomial coefficients

• Implements an add() method that can take doubles or Polynomial objects, and that returns a new Polynomial object that contains the sum.

• Implements an evaluate() method that can take a double x and returns the value of the polynomial evaluated at x.

• Implements a toString() method that will nicely print polynomials:

Code:

```
package com.company;

/**
 * Created by zsu00 on 4/5/2017.
 */

public class Polynomial {
    //fields
    private double[] coefficients;
    private int[]    power;
    int size;

    //Take inputs of double and integer arrays
    //Precon: inputs are arrays
    public Polynomial(double[] initCoeff, int[] initPower){
```

```
        if( initCoeff == null || initPower == null){
            throw new NullPointerException();
        }
        this.coefficients = initCoeff;
        this.power = initPower;
        this.size = initCoeff.length;
    }

    //Access the array of power of polynomial
    public int[] getPower(){
        return this.power;
    }

    //Access the array of coefficient of polynomial
    public double[] getCoefficients(){
        return this.coefficients;
    }

    //Get the length of coefficient or power
    public int getSize(){
        return size;
    }


/*Implements an add() method that can take doubles or Polynomial objects,
** and that returns a new polynomial object that contains the sum.
**
** input: Polynomial object
** output: Polynomial object
*/
public Polynomial add (Polynomial poly) {
    int powerSize = this.size + poly.getSize();
    int coeffSize = this.size + poly.getSize();
    //Create arrays for power and coefficient for new polynomial
    int[] newPower = new int[powerSize];
    double[] newCoeff = new double[coeffSize];

    int k = 0;
    //Iterates through each element of this polynomial
    // to find match in power with new polynomial
    for(int i = 0; i < this.size; i++) {
            boolean match = false;
            int j = 0;

            while (!match && (j < poly.getSize())) {
                if (this.power[i] == poly.getPower()[j]) {
                    newPower[k] = this.power[i];
                    newCoeff[k] = this.coefficients[i] + poly.getCoefficients()[j];
                    match = true;
                    k++;
                }else {
                    j++;
                }
            }

            if(!match){
                newPower[k] = this.getPower()[i];
                newCoeff[k] = this.getCoefficients()[i];
                k++;
            }
    }

    //Iterates through new polynomial to find power terms
```

```
    // that aren't matched with this polynomial
    for(int i = 0; i < poly.getSize(); i++){
        boolean match = false;
        int j = 0;
        while(!match && j<this.size){
            if(poly.getPower()[i] != this.power[j]) {
                match = false;
                j++;
            }else{
                match = true;
            }
        }
        if(!match){
            newPower[k] = poly.getPower()[i];
            newCoeff[k] = poly.getCoefficients()[i];
            k++;
        }
    }

    Polynomial newPoly = new Polynomial(newCoeff, newPower);
    return newPoly;
}


/*toString method prints string representation of object
input: none
output: String
 */
public String toString (){
    String result = "";
    //Print the first monomial
    if(Math.abs(this.coefficients[0]) > 0.00001){
        //Format decimal coefficient with four  digits
        String coeff = "";
        if(this.coefficients[0] - Math.round(this.coefficients[0]) != 0 ){
            coeff = String.format("%.4f",this.coefficients[0]);
        }else{
            coeff = Double.toString(this.coefficients[0]);
        }

        if(Math.abs(this.coefficients[0]) == 1) {
            if (this.power[0] == 0) {
                result = coeff;
            } else if (this.power[0] == 1) {
                result = "x";
            } else {
                result = "x^" + this.power[0];
            }
        }else{
            if(this.power[0] == 0){
                result = coeff + "";
            }else if(this.power[0] == 1) {
                result = coeff + "x";
            }else {
                result = coeff + "x^" + this.power[0];
            }
        }
    }else{
        result = "0";
    }

    //print the rest monomials
    for(int i = 1; i < this.size; i++){
```

```
            if(Math.abs(this.coefficients[i]) > 0.00001){
                //Format decimal coefficient with four  digits
                String coeff = "";
                if(this.coefficients[i] - Math.round(this.coefficients[i]) != 0 ){
                    coeff = String.format("%.4f",this.coefficients[i]);
                }else{
                    coeff = Double.toString(this.coefficients[i]);
                }

                //negative coefficient
                if(this.coefficients[i] < 0){
                    if(this.coefficients[i] == -1){
                        if(this.power[i] == 0){
                            result = result + " " + coeff;
                        }else if(this.power[i] == 1) {
                            result = result + " " + " - " + "x";
                        }else{
                            result = result + " " + " - " + "x^" + this.power[i];
                        }

                    }else{
                        if(this.power[i] == 0){
                            result = result + " " + coeff;
                        }else if(this.power[i] == 1) {
                            result = result + " " + coeff + "x";
                        }else{
                            result = result + " " + coeff + "x^" + this.power[i];
                        }
                    }

                }else{//positive coefficient case
                    if(this.coefficients[i] == 1){
                        if(this.power[i] == 0){
                            result = result + " + " + coeff;
                        }else if(this.power[i] == 1) {
                            result = result + " " + " + " + "x";
                        }else{
                            result = result + " " + " + " + "x^" + this.power[i];
                        }
                    }else{
                        if(this.power[i] == 0){
                            result = result + " + " + coeff;
                        }else if(this.power[i] == 1) {
                            result = result + " + " + coeff + "x";
                        }else{
                            result = result + " + " + coeff + "x^" + this.power[i];
                        }
                    }
                }
                if(Math.abs(this.coefficients[i]) == 1){
                }
            }
        }
        return result;
    }


    /* It evaluates polynomial at specified position
    input: double position
    output: double result
     */
    public double evaluate(double position){
        double result = 0;
```

```
        for(int i = 0; i< this.power.length; i++){
            result = result + this.coefficients[i]*Math.pow(position,this.power[i]);
        }
        return result;
    }

}
```

Console output:

Call toString() on P: 0.3333x^3 -0.5000x^2

Call toString() on R: 12.0 + 9.0x

The sum of P(x) + R(x) is 0.3333x^3 -0.5000x^2 + 12.0 + 9.0x

P(z) evaluated at z = -2.5 is 1812.832

toString() example 1 is 4.0x^2 + 2.5000x -1.0

toString() example 2 is x -1.0

toString() example 3 is 4.3000x^3 - x + 1.0