**Warm-Up Polynomial Class Part A**

For Part A of the warm-up, you are to create a polynomial class in Java using an array of double coefficients.

The class should consist of *attributes*   
double coeff  
int deg, where deg >= 0.

And *member methods:*constructor(s)  
evaluate(x)   
add- which will add a polynomial q to the given polynomial p = p+q  
subtract – which will subtract a polynomial q from the given polynomial p = p-q  
scale = which will multiply the given polynomial by a constant a p =ap  
multiply – which will multiply the given polynomial by a polynomial q p = p\*q  
  
and *polynomial methods:*sum – which adds 2 polynomials and creates a new polynomial   
 without destroying the original r = p+q  
diff - which subtracts 2 polynomials and creates a new polynomial   
 without destroying the originals r = p-q  
product - which multiplies 2 polynomials and creates a new polynomial   
 without destroying the original r = p\*q

**I AM GRADING YOUR PROJECT ON CORRECTNESS, EFFICIENCY, GOOD COMMENTING AND CODING PRACTICES, AND HOW CLOSELY YOU FOLLOW THE SPECIFICATIONS OF THE PROJECT.**

**import java.util.Arrays;**

**import java.util.Objects;**

**public class Polynomial {**

**private double [] coeff;**

**private int deg;**

**//default constructor**

**public Polynomial() {**

**}**

**//parametrized constructor**

**public Polynomial(double coeff, int deg) {**

**this.setDeg(deg);**

**//sets the**

**this.coeff = new double[deg + 1];**

**this.coeff[deg] = coeff;**

**this.reduce();**

**}**

**/\*\***

**\* Returns the coefficient array**

**\* @return coeff**

**\*/**

**public double[] getCoeff() {**

**return coeff;**

**}**

**public void setCoeff(double[] coeff) {**

**this.coeff = coeff;**

**}**

**/\*\***

**\* Returns the degree of this polynomial.**

**\* @return deg.**

**\*/**

**public int getDeg() {**

**return deg;**

**}**

**/\*\***

**\* Evaluates the provided degree of the polynomial and sets it if valid**

**\* otherwise throws an exception**

**\* @param deg**

**\*/**

**public void setDeg(int deg) {**

**throw new IllegalArgumentException("exponent cannot be negative: " + deg);**

**else this.deg = deg;**

**}**

**/\*\***

**\* Pre-compute the degree of the polynomial, in case of leading zero coefficients**

**\*/**

**private void reduce() {**

**this.deg = -1;**

**for (int i = this.coeff.length - 1; i >= 0; i--) {**

**if (this.coeff[i] != 0) {**

**this.deg = i;**

**break;**

**}**

**}**

**}**

**/\*\***

**\* Returns the resulting polynomial after adding a polynomial q to the given polynomial.**

**\***

**\* @param q the other polynomial**

**\* @return polynomial p**

**\*/**

**public Polynomial add(Polynomial q) {**

**Polynomial p = new Polynomial(0, this.max(this.deg, q.deg));**

**for (int i = 0; i <= this.deg; i++) p.coeff[i] += this.coeff[i];**

**for (int i = 0; i <= q.deg; i++) p.coeff[i] += q.coeff[i];**

**p.reduce();**

**return p;**

**}**

**/\*\***

**\* Returns the result of subtracting the specified polynomial**

**\* from this polynomial.**

**\***

**\* @param q the other polynomial**

**\* @return polynomial p**

**\*/**

**public Polynomial subtract(Polynomial q) {**

**Polynomial p = new Polynomial(0, this.max(this.deg, q.deg));**

**for (int i = 0; i <= this.deg; i++) p.coeff[i] += this.coeff[i];**

**for (int i = 0; i <= q.deg; i++) p.coeff[i] -= q.coeff[i];**

**p.reduce();**

**return p;**

**}**

**/\*\***

**\* Returns the product of this polynomial and the specified polynomial q**

**\***

**\* @param q the other polynomial**

**\* @return the polynomial p**

**\*/**

**public Polynomial multiply(Polynomial q) {**

**Polynomial p = new Polynomial(0, this.max(this.deg, q.deg));**

**for (int i = 0; i < this.deg; i++)**

**for (int j = 0; j < q.deg; j++)**

**p.coeff[i+j] += (this.coeff[i] \* q.coeff[j]);**

**p.reduce();**

**return p;**

**}**

**/\*\***

**\* Returns the resulting polynomial after summing two polynomials p and q**

**\* @param p**

**\* @param q**

**\* @return polynomial r**

**\*/**

**public Polynomial sum(Polynomial p , Polynomial q) {**

**Polynomial r = new Polynomial(0, this.max(p.deg, q.deg));**

**for (int i = 0; i <= p.deg; i++) r.coeff[i] += p.coeff[i];**

**for (int i = 0; i <= q.deg; i++) r.coeff[i] += q.coeff[i];**

**r.reduce();**

**return r;**

**}**

**/\*\***

**\* Returns the result of subtracting the two polynomials**

**\***

**\* @param p**

**\* @param q**

**\* @return polynomial r**

**\*/**

**public Polynomial dff(Polynomial p, Polynomial q) {**

**Polynomial r = new Polynomial(0, this.max(p.deg, q.deg));**

**for (int i = 0; i <= p.deg; i++) r.coeff[i] += p.coeff[i];**

**for (int i = 0; i <= q.deg; i++) r.coeff[i] -= q.coeff[i];**

**r.reduce();**

**return r;**

**}**

**/\*\***

**\* Returns the product of the two polynomials**

**\***

**\* @param p**

**\* @param q**

**\* @return the polynomial r**

**\*/**

**public Polynomial product(Polynomial p, Polynomial q) {**

**Polynomial r= new Polynomial(0, this.max(p.deg, q.deg));**

**for (int i = 0; i <= p.deg; i++)**

**for (int j = 0; j <= q.deg; j++)**

**r.coeff[i+j] += (p.coeff[i] \* q.coeff[j]);**

**r.reduce();**

**return r;**

**}**

**/\*\***

**\* Returns the result of scaling the specified polynomial**

**\***

**\* @param p the polynomial to be scaled**

**\* @param factor the scale factor to be applied to the polynomial**

**\* @return polynomial p**

**\*/**

**public Polynomial scale(Polynomial p, int factor) {**

**for (int i = 0; i <= this.deg; i++) p.coeff[i] += this.coeff[i \* factor];**

**p.reduce();**

**return p;**

**}**

**/\*\***

**\* Returns the result of evaluating this polynomial at the point x.**

**\***

**\* @param x the point at which to evaluate the polynomial**

**\* @return the integer whose value is @ (this(x))**

**\*/**

**public double evaluate(int x) {**

**double p = 0;**

**for (int i = this.deg; i >= 0; i--)**

**p = this.coeff[i] + (x \* p);**

**return p;**

**}**

**/\*\***

**\* Gets the maximum of two values**

**\* @param a**

**\* @param b**

**\* @return max**

**\*/**

**private int max(int a, int b) {**

**int max =a;**

**max = a > b? a: b;**

**return max;**

**}**

**@Override**

**public boolean equals(Object o) {**

**if (this == o) return true;**

**if (o == null || getClass() != o.getClass()) return false;**

**Polynomial that = (Polynomial) o;**

**return deg == that.deg && Arrays.equals(coeff, that.coeff);**

**}**

**@Override**

**public int hashCode() {**

**int result = Objects.hash(deg);**

**result = 31 \* result + Arrays.hashCode(coeff);**

**return result;**

**}**

**/\*\***

**\* Return a string representation of this polynomial.**

**\* @return a string representation of this polynomial in the format**

**\* 2x^3 - x^2 + 7x + 10**

**\*/**

**@Override**

**public String toString() {**

**if (this.deg == -1) return "0";**

**else if (this.deg == 0) return "" + this.coeff[0];**

**else if (this.deg == 1) return this.coeff[1] + "x + " + this.coeff[0];**

**String s = this.coeff[this.deg] + "x^" + this.deg;**

**for (int i = this.deg - 1; i >= 0; i--) {**

**if (this.coeff[i] == 0) continue;**

**else if (this.coeff[i] > 0) s = s + " + " + (this.coeff[i]);**

**else if (this.coeff[i] < 0) s = s + " - " + (-this.coeff[i]);**

**if (i == 1) s = s + "x";**

**else if (i > 1) s = s + "x^" + i;**

**}**

**return s;**

**}**

**}**