```java
package myMath;

import java.util.Iterator;

/** This interface represents a simple function of type y=f(x), where both y
and x are real numbers.
**/
public interface function {
	public double f(double x);
}

* The interface represents a continuance function.
public interface cont_function extends function{
	/**
	 * Compute a Riman's integral from x0 to x1 in eps steps.
	 * @param x0 starting pooint
	 * @param x1 end point
	 * @param eps positive step value
	 * @return the approximated area above X-axis below this function
	bounded in the range of [x0,x1]
	 */
	public double area(double x0, double x1, double eps);
}


 * Does the sorting process, using the equation
public class Monom_Comperator implements Comparator<Monom> {
```

```
* This class represents a simple "Monom" of shape a*x^b, where a is a real
number and a is an integer (summed a none negative),
 * see: https://en.wikipedia.org/wiki/Monomial
 * The class implements function and support simple operations as:
construction, value at x, derivative, add and multiply.
 * @author Raam Banin
 * @version 1.0
 */
public class Monom implements function{

	private double _coefficient; //
	private int _power;
	private static Monom init_from_string = null;

	/**
	 * Define string with non-capital letter. define two variables. if the
	string is empty return error message. Define index = x
	 * if x = -1 return a= string, else a is a * x and b= 1 define p_power
	as  power index. if p_power is real so b is all the numbers after the
	power.
	 * @param s - get monom in string.
	 * @return - return a new monom.
	 */
	private Monom init_from_string(String s)

	/**
	 * create a new monom according to string and get it in to the original
	monom.
	 * @param s - monom in the string
	 */
	public Monom(String s)

	/**
	 * constructor with parameterized.
	 * @param a - coefficient.
	 * @param b - power.
	 */
	public Monom(double a, int b)

	/**
	 * copy constructor
	 * @param ot - copy parameterized
	 */
	public Monom(Monom ot)

	/**
	 * The function return the y value if the power is not negative.
	 */
	public double f(double x)

	/** Monom derivative - if power is 0 return 0, else do power * monom
	coefficient and -1 from monom power.
	 * @return new monom after a derived method.
	 */
	public Monom derivative()
```

```
/**
 * addition between two monom. if powers of the monoms is equal sum the
monoms coefficients.
 * @param m - the monom we want to add.
 */
public void add(Monom m)

/**
 * subtraction between two monom - if the monoms coefficients are equal
subtract between the monoms.
 * @param m - the monom we want to subtract.
 */
public void substract(Monom m)
/**
 * multiply between two monoms and sum the powers.
 * @param m - the monom we want to multiply.
 */
public void multiply(Monom m)
/**
* comparison between two monoms - if the monom is not empty and the
coefficients are equal
 and the powers are equals as well so the monoms are equal.
 * @param m - the monom we want to make comparison.
 * @return - if equal or no.
 */
public boolean equals(Monom m)
```

```
/**
 * This interface represents a general Polynom: f(x) = a_1X^b_1 + a_2*X^b_2
... a_n*Xb_n,
 * where: a_1, a_2 ... a_n are real numbers and b_1<b_2..<b_n >=0 are none
negative integers (naturals)
 * For formal definitions see: https://en.wikipedia.org/wiki/Polynomial
 *
 * Such polygon has the following functionality:
 * 1. Init:
 * 1.1 Init(String)
 * 1.2 Init() // zero Polygon
 * 1.3 Polynom copy() // deep copy semantics
 *
 * 2. Math:
 * 2.1 void add(Polygon p) // add p to this Polynon
 * 2.2 void subtract(Polygon p) // subtract p from this Polygon
 * 2.3 void multiply(Polygon p) // multiply this Polygon by p
 *
 * 3. Utils
 * 3.1 isZero()
 * 3.2 Polynom derivative() // returns a new Polygon of the derivative
("NIGZERET").
 * 3.3 double f(x) // return this Polygon value at p(x)
 * 3.4 boolean equals(Polygon p) // returns true iff for any x: this.f(x) ==
p.f(x)
 * 3.5 double root(double x0, double x1, double eps) // assuming
(f(x0)*f(x1)<=0, returns f(x2) such that:
 *
      //     (i) x0<=x2<=x2 & (ii) {f(x2)<eps
 * 3.6 String toString() // returns a String such that it can be used for init
an equal(s) Polygon.
```

```
public interface Polynom_able extends cont_function{
	/**
	 * Add p1 to this Polynom
	 * @param p1
	 */
	public void add(Polynom_able p1);
	/**
	 * Add m1 to this Polynom
	 * @param m1 Monom
	 */
	public void add(Monom m1);
	/**
	 * Subtract p1 from this Polynom
	 * @param p1
	 */
	public void substract(Polynom_able p1);
	/**
	 * Multiply this Polynom by p1
	 * @param p1
	 */
	public void multiply(Polynom_able p1);
	/**
	 * Test if this Polynom is logically equals to p1.
	 * @param p1 – other polynom.
	 * @return true iff this pulynom represents the same function ans p1
	 */
	public boolean equals (Polynom_able p1);
	/**
	 * Test if this is the Zero Polynom
	 * @return Boolean true or false.
	 */
	public boolean isZero();
	/**
	 * Compute a value x' (x0<=x'<=x1) for with |f(x')| < eps
	 * assuming (f(x0)*f(x1)<=0, returns f(x2) such that:
	 * *   (i) x0<=x2<=x2 && (ii) f(x2)<eps
	 * @param x0 starting point
	 * @param x1 end point
	 * @param eps step (positive) value
	 * @return The resulting approximation of the root according to epsilon
	 */
	public double root(double x0, double x1, double eps);
	/**
	 * create a deep copy of this Polynum
	 * @return polynom.
	 */
	public Polynom_able copy();
	/**
	 * Compute a new Polynom which is the derivative of this Polynom
	 * @return derivative polynom.
	 */
	public Polynom_able derivative();
	/**
	 * Compute Riemann's Integral over this Polynom starting from x0 till
	x1 using eps size steps,
```

```
     * see: https://en.wikipedia.org/wiki/Riemann_integral
     * @return the approximated area above the x-axis below this Polynom and
    between the [x0,x1] range.
     */
    public double area(double x0,double x1, double eps);
    /**
     * @return an Iterator (of Monoms) over this Polynom.
     */
    public Iterator<Monom> iteretor();
}
```