```
package util;

import java.util.*;

public class Polynomial {
	// The order of the polynomial. The size of the coefficient array is this number plus one.
	private int order;
	// Coefficients in ascending order of rank
	private double coefficients[];

	// Returns the order of this polynomial
	public int getOrder() {
		return order;
	}

	// Gets the coefficient for the given order term.
	// If the given integer is less than 0 an IndexOutOfBoundsException is thrown.
	public double getCoefficient(int order)throws IndexOutOfBoundsException{
		if(order < 0)
			throw new IndexOutOfBoundsException();
		if(order > this.order)
			return 0.0;
		return this.coefficients[order];
	}

	//Constructor that takes a list of coefficients in ascending order of rank
	private Polynomial(List<Double> coefficients){
		this.order = coefficients.size()-1;
		this.coefficients = new double[this.order+1];
		for(int i=0; i<=this.order; i++){
			this.coefficients[i] = coefficients.get(i);
		}
	}

	// Default constructor that creates the zero polynomial
	private Polynomial(){
		this.order = 0;
		this.coefficients = new double[1];
		this.coefficients[0] = 0;
	}

	// Creates a polynomial from a list of the coefficients for that polynomial in ascending order
	public static Polynomial create(List<Double> coefficients){
		coefficients = Polynomial.removeLeadingZeroes(coefficients);
		if(coefficients.size() <= 0){
			return new Polynomial();
		}else{
			return new Polynomial(coefficients);
		}
	}

	// Creates a zero polynomial
	public static Polynomial create(){
		return new Polynomial();
	}

	// Recursively removes all of the zeroes from the end of a list
	private static List<Double> removeLeadingZeroes(List<Double> coefficients){
		if(coefficients.size() > 0){
			if(coefficients.get(coefficients.size()-1).equals(new Double(0.0))){
				coefficients.remove(coefficients.size()-1);
				return Polynomial.removeLeadingZeroes(coefficients);
			}
		}
		return coefficients;
	}

	// Returns the sum of this polynomial and the given polynomial
	public Polynomial add(Polynomial b){
		LinkedList<Double> addedCoeffs = new LinkedList<Double>();
		int maxorder = 0;
		if(b.order > this.order)
			maxorder = b.order;
		else
			maxorder = this.order;
		for(int i=0; i<=maxorder; i++){
			double coeff = 0;
			if(i<=this.order)
				coeff += this.coefficients[i];
			if(i<=b.order)
				coeff += b.coefficients[i];
```

```
                        addedCoeffs.add(coeff);
                }
                return Polynomial.create(addedCoeffs);
        }

        // Returns a polynomial that is this polynomial multiplied by the given scalar quantity
        public Polynomial multiplyByScalar(double c){
                if(c == 0.0)
                        return new Polynomial();
                LinkedList<Double> newCoeffs = new LinkedList<Double>();
                for(int i=0; i<=this.order; i++){
                        newCoeffs.add(this.coefficients[i]*c);
                }
                return Polynomial.create(newCoeffs);
        }

        // Returns the derivative of this polynomial
        public Polynomial derivative(){
                if(this.order == 0)
                        return new Polynomial();
                LinkedList<Double> derivCoeffs = new LinkedList<Double>();
                for(int i=1; i<=this.order; i++){
                        derivCoeffs.add(((double)i)*this.coefficients[i]);
                }
                return Polynomial.create(derivCoeffs);
        }

        // Evaluate this Polynomial for the given value of x
        public double eval(double x){
                return this.evalHelper(x, 0);
        }

        // Loops through the polynomial adding up the values and multiplying through by x.
        private double evalHelper(double x, int index){
                if(index <= this.order)
                        return this.coefficients[index] + (x * this.evalHelper(x, index+1));
                else
                        return 0.0;
        }
}
```