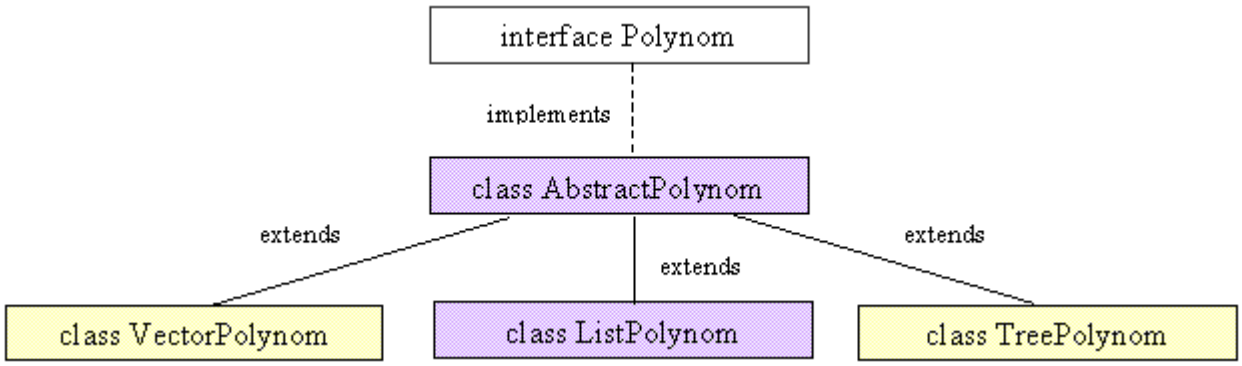
Дисциплина: *СД и ООП* **Тема:** *Полином*
Специалност: *КН 1 курс, 3 и 4 групи* **Дата**: *14.04.2010 г.*

Задача 1: Дадена е йерархията от интерфейси и класове:



Интерфейсът **Polynom** представя операции на *полином на една променлива с коефициенти реални числа*:

```
public interface Polynom {
        public int degree(); //Връща най-високата степен на едночлен в текущия полином.
        public double coeff(int degree); //Връща коефициента пред степен degree в  текущия полином
        public double setCoeff(int degree,double value); //Променя  коефициента пред степен degree в текущия
                                                          //полином на value и връща старата стойност
        public Polynom add(Polynom arg); //Връща полинома-сума на текущия полином и arg
        public Polynom mult(Polynom arg); //Връща полинома-произведение на текущия полином и arg
        public Polynom derive(); //Връща полином, който е първа производна на текущия полином
        public double eval(double value); //Пресмята стойността на текущия полином за стойността value
        public double[] toArray(); //Връща масив с коефициентите на текущия полином и степени -  индексите
                                   //на масива
        public boolean equals(Object obj); //Проверява дали текущия полином и полинома obj съвпадът
        public String toString(); //Връща текущия полином, представен в символен вид
        public java.util.Iterator<Integer> degrees(); //Обхожда само степени на текущия полином с ненулеви
                                                      //коефициенти в намаляващ ред
}
```

Представяне на коефициентите a_i на полином $P(x) = a_n x^n + \ldots + a_0$:

1. Масив double[] coeffs = new double[n + 1], като coeffs[i] = a_i
2. Вектор coeffs, като coeffs = (a_0, a_1, …, a_n)
3. Структура (вектор, списък, дърво) coeffs, като coeffs = {(i, a_i) | $a_i \neq 0$}

Класът **Pair** представя двойка-елемент на структурата coeffs от т.3:

```
public class Pair implements java.lang.Comparable<Pair> {
        //Data
        private double d;
        private int i;

        //Constructor
        public Pair(double d,int i) { this.d = d;  this.i = i; }

        //Access methods
        public double getDouble() { return d; }
        public double setDouble(double value) { double w = d; d = value; return w; }
        public int getInteger() {return i; }
        public int setInteger(int value) { int w = i; i = value; return w; }

        //Comparable method implementation
        public int compareTo(Pair obj) { return this.i - obj.i; }

        //Override methods - inherited from class Object
        public boolean equals(Object obj) {
```

```
                Pair w = (Pair)obj;
                return i == w.i && d == w.d;
        }
}
```

1.Да се реализира клас **AbstractPolynom**, съгласно спецификацията:

```
public abstract class AbstractPolynom implements Polynom {
        //Степен на полином
        protected int degree;

        //Абстрактни методи
        public abstract double coeff(int degree);
        public abstract double setCoeff(int degree,double value);
        protected abstract Polynom create(); //Създава полинома P(x)=0
        public abstract java.util.Iterator<Integer> degrees();

        //Реализация на методи на интерфейса Polynom
        public int degree() {…}
        public Polynom add(Polynom arg) {…}
        public Polynom mult(Polynom arg) {…}
        public double eval(double value) {…}
        public Polynom derive() {…}
        public double[] toArray() {…}

        //Предефиниране на наследени методи от клас Object
        public boolean equals(Object arg) {…}
        public String toString() {…}
}
```

2.Да се реализира клас **VectorPolynom**, съгласно спецификацията:

```
public class VectorPolynom extends AbstractPolynom implements Polynom {
        //Представяне на полином
        private java.util.Vector<Pair> coeffs;          //Ненулеви коефициенти

        //Конструктори
        public VectorPolynom(double coeff,int degree) {…} //P(x)=coeff . x^degree
        public VectorPolynom(double[] coeffs){…} //P(x)=coeffs[0]+…+ coeffs[coeffs.length–1]x**(coeffs.length-1)
        public VectorPolynom(Polynom p) {…} //P(x)=p

        //Собствен метод
        private double remove(int degree) {…} //Премахва едночлен от степен degree от текущия полином, като
                                              //връща коефициента
        //Реализация на абстрактните методи на класа AbstractPolynom
        //Наследяване на методите на интерфейса Polynom, реализирани в класа AbstractPolynom
}
```

3.Да се реализира клас **VectorPolynomDegreesIterator** - итератор за обхождане в намаляващ ред на степените на членовете с ненулеви коефициенти на полином, съгласно спецификацията:

```
public class VectorPolynomDegreesIterator implements java.util.Iterator<Integer> {
        //Данни
        java.util.Vector<Pair> vector;
        //Други променливи

        //Конструктор(и)

        //Реализация на методите на интерфейса Iterator
}
```

4.Да се реализира клас **ListPolynom**, съгласно спецификацията:

```
public class ListPolynom extends AbstractPolynom implements Polynom {
        //Представяне на полином
        private java.util.LinkedList<Pair> coeffs;     //Ненулеви коефициенти

        //Конструктори
        public ListPolynom(double coeff,int degree) {…} //P(x)= coeff . x^degree
        public ListPolynom(double[] coeffs){…} //P(x)=coeffs[0]+…+ coeffs[coeffs.length–1]x**(coeffs.length-1)
        public ListPolynom(Polynom p) {…} //P(x)=p

        //Собствен метод
        private double remove(int degree) {…} //Премахва едночлен от степен degree от текущия полином, като
                                              //връща коефициента
        //Реализация на абстрактните методи на класа AbstractPolynom
        //Наследяване на методите на интерфейса Polynom, реализирани в класа AbstractPolynom
}
```

Реализацията на класа **ListPolynom** се основава на представяне на коефициентите на полином $P(x) = a_n x^n + \ldots + a_0$ чрез списък $coeffs = \{(i,a_i) \mid a_i \neq 0\}$, като се използва родов линеен свързан списък **LinkedList<E> -** виж http://java.sun.com/javase/6/docs/api/.

Класът **ListPolynomDegreesIterator** представя итератор за обхождане в намаляващ ред на степените на членовете с ненулеви коефициенти на полином. Използван е интерфейсът **Iterator<E>** - виж http://java.sun.com/javase/6/docs/api/.

```
public class ListPolynomDegreesIterator implements java.util.Iterator<Integer> {
        //Data
        java.util.Iterator<Pair> it;
        java.util.LinkedList<Pair> list;

        //Constructor
        public ListPolynomDegreesIterator(java.util.LinkedList<Pair> list) {
                this.list = list;
                reset();
        }

        //Methods
        public boolean hasNext() { return it.hasNext(); }
        public Integer next() { return it.next().getInteger(); }
        public void remove() { throw new java.lang.UnsupportedOperationException(); }
        public void reset() { it = this.list.iterator(); }
}
```

***Задача за упражнение*:** Да се реализира клас **SparseMatrix**, съгласно спецификацията:

```
public class SparseMatrix extends Matrix {
        //Представяне на елементите на матрица
        private java.util.LinkedList<Triple> elements;

        //Конструктори
        public SparseMatrix(int rows,int cols) {...}
        public SparseMatrix(double[][] data) {...}
        public SparseMatrix(Matrix arg) {...}

        //Собствен метод
        private double remove(int i,int j) {...}

        //Методи, наследени от клас Matrix
}
```

Реализацията на класа **SparseMatrix** се основава на представяне на елементите a_{ij} на матрица A(nxm), $i \in [1;n]$, $j \in [1;m]$ чрез списък elements, като elements=$\{(i,j,a_{ij}) \mid a_{ij} \neq 0\}$ и се използва родов линеен свързан списък **LinkedList<E>**.

Класът **Triple** представя елемент на списъка:

```
public class Triple extends Pair {
	//Data
	protected int j;

	//Constructor
	public Triple(double d,int i,int j) { super(d,i); this.j = j; }

	//Access methods - inherited from class Pair
	//Access methods
	public int getInteger2() {return j; }
	public int setInteger2(int value) { int w = j; j = value; return w; }

	//Comparable method implementation
	public int compareTo(Triple obj) {
		int temp = super.i - obj.i;
		return temp != 0 ? temp : this.j - obj.j;
	}

	//Override
	public boolean equals(Object obj) {
		Triple w = (Triple)obj;
		return super.equals(new Pair(w.d,w.i)) && this.j == w.j;
	}
}
```