Дисциплина: *СД и ООП* **Тема:** *Вектор*
Специалност: *КН 1 курс, 3 и 4 групи* **Седмица**: *23.03 - 29.03.2009 г.*

Задача 1: Даден е интерфейсът:

```
public interface GStack<E> {
        public  boolean isEmpty();
        public void push(E x);
        public E pop();
        public E top();
}
```

Да се реализира родов стек **GVectorStack<E>** с възможност за увеличаване на капацитета при включване, както е при **Vector<E>**:

```
import java.util.Vector;

public class GVectorStack<E> implements GStack<E> {
        //Представяне на елементите

        //Конструктори
        public GVectorStack() {..}
        public GVectorStack(int n) {…}
        public GVectorStack(Object[] data) {…}

        //Реализация на методите на интерфейса GStack
}
```

Предложената реализация използва родов вектор **java.util.Vector<E>** - виж http://java.sun.com/javase/6/docs/api/.

Задача 2: Интерфейсът **BigInteger** представя операции на *голямо цяло десетично число без знак*:

```
public interface BigInteger {
        public int length(); //Връща дължината на текущото число
        public BigInteger add(BigInteger arg); //Събира текущото число с числото arg
        public BigInteger mult(BigInteger arg); //Умножава текущото число по числото arg
        public int compareTo(Object obj); //Сравнява текущото число с числото arg
        public boolean equals(Object obj); //Проверява дали текущото число и числото arg съвпадат
        public Object clone(); //Създава копие на текущото число
        public String toString(); //Преобразува текущото число в низ
}
```

Класът **VectorBigInteger** реализира интерфейса **BigInteger**, като цифрите на голямо цяло десетично число $N = a_n(10^k)^n + a_{n-1}(10^k)^{n-1} + \ldots + a_0$ се съхраняват във вектор $digits = (a_0,\ldots,a_n)$.

```
import java.util.*;

public class VectorBigInteger implements BigInteger {
         //Data
        private Vector<Integer> digits;       //digits = (a0,...,an)
        public static int base = 10;          //base = 10^k, k = 1, 2, ...

        //Constructors
        public  VectorBigInteger() { this("0"); } //N=0
        public  VectorBigInteger(BigInteger arg) { this(arg.toString()); } //N=arg
        public  VectorBigInteger(String arg) {…}  //arg="an...a0" => N=an…a0

        //Private method
        private static int howLong(int n) {…}
```

```
//BigInteger methods implementation
public int length() { return digits.size() * (howLong(base)-1); }

public BigInteger add(BigInteger arg) {
        VectorBigInteger Int1 = this;
        VectorBigInteger Int2 = new VectorBigInteger(arg.toString());
        Vector<Integer> left = Int1.digits;
        Vector<Integer> right = Int2.digits;

        int max = Math.max(left.size(),right.size());
        int min = Math.min(left.size(),right.size());

        VectorBigInteger w = new VectorBigInteger();
        Vector<Integer> result = w.digits = new Vector<Integer>();

        int d, c = 0;
        for(int i = 0; i < min; i++) {
                d = left.get(i) + right.get(i) + c;
                result.add(result.size(),d % base);
                c = d / base;
        }

        Vector<Integer> temp;
        if(max == left.size()) temp = left;
        else temp = right;
        for(int i = min; i < temp.size(); i++) {
                d = temp.get(i) + c;
                result.add(result.size(),d % base);
                c = d / base;
        }
        if(c != 0) result.add(result.size(),c);

        return w;
}

 public BigInteger mult(BigInteger arg) {
        VectorBigInteger Int1 = this;
        VectorBigInteger Int2 = new VectorBigInteger(arg.toString());
        Vector<Integer> left = Int1.digits;
        Vector<Integer> right = Int2.digits;

        VectorBigInteger w = new VectorBigInteger();
        Vector<Integer> result = w.digits = new Vector<Integer>();;

        for(int i = 0; i < left.size() + right.size()-1; i++)
                result.add(0,0);

        int d, c;
        for(int i = 0; i < left.size(); i++) {
                d = left.get(i);
                for(int j = 0; j < right.size(); j++) {
                        c = result.get(i+j) + d * right.get(j);
                        result.set(i+j,c);
                }
        }

        c = 0;
        for(int i = 0; i < result.size(); i++) {
                d = result.get(i) + c;
                result.set(i,d % base);
                c = d / base;
        }

        if(c != 0) result.add(result.size(),c);
```

```
            return w;
        }

        public int compareTo(Object obj){ System.out.println("Not implemented!"); return 1; }

        //Override methods - inherited from class Object
        public boolean equals(Object obj){ System.out.println("Not implemented!"); return true; }
        public Object clone(){ System.out.println("Not implemented!"); return null; }

        public String toString() {…} //N=an…a0 => result="an...a0"

        //Constants
        public static final VectorBigInteger ZERO = new VectorBigInteger("0");
        public static final VectorBigInteger ONE = new VectorBigInteger("1");
        public static final VectorBigInteger TEN = new VectorBigInteger("10");
}
```

Задачи за упражнение:

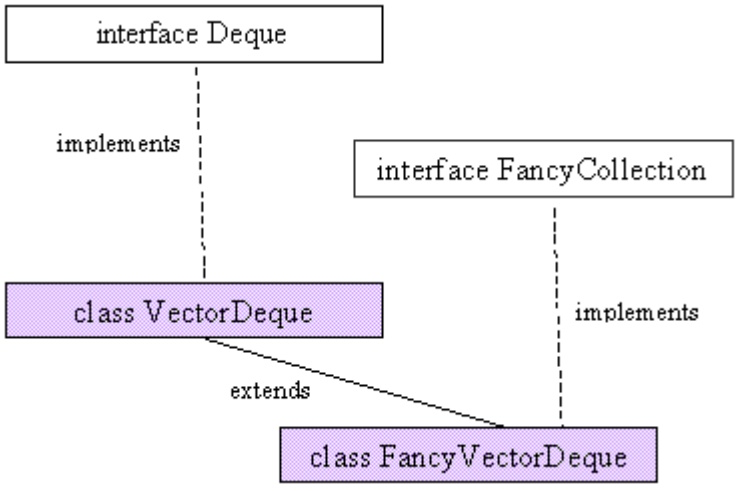
1.Даден е интерфейсът:

```
public interface GDeque<E> {
        public  boolean isEmpty();
        public void addFront(E x);
        public void addRear(E x);
        public E removeFront();
        public E removeRear();
        public E getFirst();
        public E getLast();
}
```

1.1. Да се реализира родов дек с нарастващ капацитет

1.2. Да се реализира родов дек с фиксиран капацитет

2.Дадена е йерархията:



Класът **Vector** е реализация на вектор с елементи от тип **Object**, предоставящ същите операции, както родовия вектор **java.util.Vector<E>**.

2.1. Да се реализира клас **VectorDeque**, съгласно спецификацията:

```
public class VectorDeque implements Deque {
        //Представяне на дек
        protected Vector elements;

        //Конструктори
        public VectorDeque() {…} //Създава празен дек
        public VectorDeque(int n) {…} //Създава празен дек с начален капацитет n
```

```
        public VectorDeque(Object[] arg) {…} //Създава дек от елементите на масива arg, като първият
                                             //елемент на масива е в началото на дека, а последният елемент – накрая
        //Реализация на методите на интерфейса Deque
}
```

2.2. Да се реализира клас **FancyVectorDeque**, съгласно спецификацията:

```
public class FancyStackDeque extends VectorDeque implements FancyCollection {
        //Конструктори
        public FancyVectorDeque() {…} //Създава празен дек
        public FancyVectorDeque(FancyCollection arg) {…} //Създава дек от елементите на  колекцията arg
        public FancyVectorDeque(Object[] arg) {…} //Създава дек от елементите на масива arg

        //Наследяване на методите на интерфейса Deque
        //Реализация на методите на интерфейса FancyCollection
        //Предефиниране на методите, наследени от клас Object
}
```