Структури от Данни и Обектно-Ориентирано Програмиране
спец. Компютърни Науки
Упражнение №4
18.03.2010г.

ТЕМА: Изброени типове. Използване на стек. Структура от данни дек

Изброени типове
Типове, чиито възможни стойности са множества от константи.

```
public class Seasons {
        public enum Season { WINTER, SPRING, SUMMER, FALL }

        public static void main(String[] args) {
                for (Season s : Season.values())
                        System.out.println(s);
        }
}
```

```
import java.util.*;
public class WeekDays {
        enum Day { MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
                                        FRIDAY, SATURDAY, SUNDAY }

        public static void main(String[] args) {
                System.out.print("Weekdays: ");
                for (Day d : EnumSet.range(Day.MONDAY, Day.FRIDAY))
                        System.out.print(d + " ");
                System.out.println();
        }
}
```

```
import java.util.Scanner;
public enum Operation {
        PLUS {
                 double eval(double x, double y) { return x + y; }
        },
        MINUS {
                double eval(double x, double y) { return x - y; }
        },
        TIMES {
                double eval(double x, double y) { return x * y; }
        },
        DIVIDED_BY {
                double eval(double x, double y) { return x / y; }
        };
```

```
        abstract double eval(double x, double y);

        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                double x = sc.nextDouble();
                double y = sc.nextDouble();

                for (Operation op : Operation.values())
                        System.out.println(x + " " + op + " " + y + " = " + op.eval(x, y));
        }
}
```

```
enum Signal { GREEN, YELLOW, RED }

public class TrafficLight {
        Signal color = Signal.RED;

        public void change() {
           switch(color) {
             case RED:    color = Signal.GREEN;
                          break;
             case GREEN:  color = Signal.YELLOW;
                          break;
             case YELLOW: color = Signal.RED;
                          break;
           }
        }

        public String toString() {
           return "The traffic light is " + color;
        }

        public static void main(String[] args) {
           TrafficLight t = new TrafficLight();
           for(int i = 0; i < 10; i++) {
             System.out.println(t);
             t.change();
           }
        }
}
```

```
public class EnumDemonstration {
	enum TrashContainer {
		// name and size for three types of containers
		garbage(30), recycle(60), garden(90);

		// variable initialize when the enum constant is created
		private int containerSize;

		// constructor passed the enum constant integer argument
		private TrashContainer(int size) { containerSize = size; }

		// returns value associated with the enum constant
		public int size() { return containerSize; }
	}

	public static void main(String[] args) {
		// scan values in the TrashContainer enum clsss
		for (TrashContainer c : TrashContainer.values()) {
			// display container name, size, and color
					System.out.println(c + ":   \t Size " + c.size() +
						" l. \tColor " + color(c));
		}
	}

	// private enum class with names for the containers
	private enum ContainerColor {brown, yellow, green}

	// switch statement selects container type and returns color
	private static ContainerColor color(TrashContainer container) {
		ContainerColor color = null;
		switch(container) {
			case garbage:
				color = ContainerColor.brown;
				break;
			case recycle:
				color = ContainerColor.yellow;
				break;
			case garden:
				color = ContainerColor.green;
				break;
		}

		return color;
	}
}
```

Задача: Да се въведе символен низ, представляващ аритметичен израз, който съдържа цели числа, скоби и знаците за операции {+, -, *, /} и да се преобразува в обратен полски запис, след което този запис да се използва за пресметане на стойността му.

Пояснение: Постфиксен (обратен полски) запис на аритметичен израз се нарича такъв запис, при който знакът на двуаргументната операция следва след двата операнда. Необходимостта от използване на скоби при този вид запис отпада. Примери:

5*(((3+8)*(4*6))+7) инфиксен запис
5 3 8 + 4 6 * * 7 + * постфиксен (обратен полски) запис

Алгоритъм за преминаване от инфиксен към постфиксен (обратен полски) запис:
Нека ***input*** е символният низ, съдържащ израза в инфиксен запис, ***rpn*** е символен низ, който в края ще съдържа постфиксния запис, ***ptr*** е указател към поредната лексема в ***input***, ***st*** е стек. Въвеждаме следните тегла на четирите аритметични операции: умножение и деление – тегло 1, събиране и изваждане – тегло 2.

1. Ако ***input*** не е свършил, премини към 2. Иначе премини към т.3.
2. Ако ***ptr*** сочи към операнд (число или променлива), то той се добавя в края на ***rpn.*** Ако сочи лява скоба, тя се добавя в стека. Ако сочи операция, тя се добавя в стека, като предварително вади от там поред операциите, които са с тегло по-малко или равно на дадената и в реда на изваждането им ги добавя към ***rpn.*** Ако сочи дясна скоба, то от стека се изваждат всички операции до първата лява скоба и в реда на изваждането им се добавят в края на ***rpn.*** Лявата скоба се изважда от стека, но не се преписва в ***rpn.*** Указателят ***ptr*** се премества една позиция напред и се преминава към т.1.
3. Останалите в стека операции се изваждат и в същия ред се добавят в края на ***rpn.***

Алгоритъм за пресмятане на аритметичен израз, зададен в обратен полски запис:
Нека ***str*** е символният низ, който съдържа аритметичен израз, зададен в обратен полски запис, ***ptr*** е указател, който сочи към поредната лексема в него (в началото най-лявата), а ***stack*** е стек. Под лексема ще разбираме най-дългата последователност от съседни символи, която е число или знак за операция.

- Ако ***ptr*** сочи към число, добавяме това число в стека и придвижваме указателя;
- Ако ***ptr*** сочи към операция ***op***, то вадим от стека числата ***n*** и ***m*** и извършваме операцията ***m op n***, чийто резултат поместваме в стека, а указателя придвижваме напред в низа.

Когато символният низ свърши, в стека се намира числото, което е стойност на аритметичния израз, зададен от ***str.***

```
import java.util.Scanner;
import java.util.Vector;
import java.util.Stack;
import java.util.EmptyStackException;


public class ToInvPolishNotation {

    // Клас за представяне на операциите, които се записват в стека
    static enum Operation {
        BRACE ("(", 10),
        PLUS   ("+", 5),
        MINUS ("-", 5),
        MULT  ("*", 1),
        DIV     ("/", 1);

        private final String symbol;
        private final int priority;

        Operation(String s, int priority) {
            this.symbol = s;
            this.priority = priority;
        }

        int priority() {
            return priority;
        }

        public String toString() {
            return symbol;
        }
    }


    static char lookahead;
    static Scanner sc = new Scanner(System.in);
    static String input = sc.nextLine();
    static int ind = 0;


    public static void main(String[] args) {
        Vector<Object> v = null;
        try {
            // преобразуване в постфиксен запис
            System.out.println((v = convert()).toString());
        }
```

```
        catch(Exception e) {
                System.out.println(e.toString());
        }
        try {
                // пресмятане на аритметичния израз
                System.out.println(compute(v));
        }
        catch(Exception e) {
                System.out.println(e.toString());
        }
}

// Метод, който преобразува инфиксен запис на ар. израз в постфиксен
static Vector<Object> convert() throws Exception {
        Vector<Object> result = new Vector<Object>();  // Вектор за
                                                       //  получаване на   резултата
        Stack<Operation> st = new Stack<Operation>();  // Стек за временно
                                                       //  пазене на операциите
        int val;
        lookahead = getC();
        do {  // цикъл за четене на лексемите от входа
                if (Character.isDigit(lookahead)){ // четена на лексема - число
                        val = lookahead-'0';
                        while (Character.isDigit(lookahead = getC()))
                                val = val*10+lookahead-'0';
                        result.add(new Integer(val));  // добавяне  във вектора
                }
                else {  // четене на лексема - операция
                        Operation t = null;
                        switch(lookahead) {// добавяне на операцията в стека
                            case '(':  st.add(Operation.BRACE);
                                       break;
                            case '+': t = Operation.PLUS; insert(t, st, result);
                                       break;
                            case '-':  t = Operation.MINUS; insert(t, st, result);
                                       break;
                            case '*':  t = Operation.MULT; insert(t, st, result);
                                        break;
                            case '/':  t = Operation.DIV; insert(t, st, result);
                                       break;
                            case ')' : insertBrace(st, result);
                                       break;
                            default: throw new Exception("Error ");
                        }
                        lookahead=getC();
                }
```

```
        } while (lookahead != '$');
        while(!st.isEmpty()) //след край на вх. лексеми, останалите в стека
                result.add(st.pop()); // операции се прехвърлят във вектора
        return result;
}


// Метод за пресмятане на аритметичен израз, зададен в ОПЗ
static double compute (Vector<Object> v) throws Exception {
        double result = 0;
        char op;
        Stack<Number> st = new Stack<Number>(); // Запис в стека
        Object o = null;
        for(int i=0; i < v.size(); i++) {
                o = v.elementAt(i);  // четене на поредната лексема от входа
                if(o instanceof Number) // ако е число - записва се в стека
                        st.push((Number)o);
                else {
                        double operand1 = 0;
                        double operand2 = 0;
                        try {// ако е оперция - от стека се вадят  втория
                                operand2 = ((Number)st.pop()).doubleValue();
                                // и първия му операнд
                                operand1 = ((Number)st.pop()).doubleValue();
                        }
                        catch(EmptyStackException e) {
                                throw new Exception("Wrong Expression");
                        }
                        op = o.toString().charAt(0);
                        switch(op) {   // проверява се коя е операцията
                                        // и се извършва
                                case '+': result = operand1 + operand2; break;
                                case '-': result = operand1 - operand2; break;
                                case '*': result = operand1 * operand2; break;
                                case '/': result = operand1 / operand2; break;
                        }
                        st.push(new Double(result));  // запис в стека
                }
        }
        int numb = 0;  // Когато входният поток свърши
        while(!st.isEmpty()) {  // ако в стека има точно едно число
                                // това е резултатът от пресмятането
                result = ((Double)st.pop()).doubleValue();
                numb++;
        }
```

```
            if(numb != 1)
                throw new Exception("Wrong Expression");
            return result;
        }


        // Метод, който включва поредната прочетена операция в стека, като първо
        // прехвърля всички операции с по-висок приоритет от стека във вектора.
        static void insert(Operation t, Stack<Operation> st, Vector<Object> v) {
            Operation tt;
            while(!st.isEmpty()) {
                tt = st.peek();
                if(tt.priority() <= t.priority())
                    v.add(st.pop());
                else break;
            }
            st.push(t);
        }


/* Методът се вика при наличие на затваряща скоба във входния поток и води до
*  прехвърляне на всички знаци за операции от стека във вектора до срещане на
*/ отваряща скоба. Самата отваряща скоба не се прехвърля във вектора

        static void insertBrace(Stack<Operation> st, Vector<Object> v) {
            Operation tt;
            while(!st.isEmpty()) {
                tt = st.pop();
                if(tt != Operation.BRACE)
                    v.add(tt);
                else
                    break;
            }
        }

// Метод, който чете и връща поредния символ от входа, при край връща '$'
        static char getC() {
            try {
                while(Character.isWhitespace(input.charAt(ind++)));
                return input.charAt(ind-1);
            }
            catch (Exception e) {
                return '$';
            }
        }
}
```

Interface Deque<E>

Интерфейсът включва методи за достъп до двата края на линейната структура. Има методи за добавяне на елемент, премахване на елемент и проверка на елемент в началото и края на структурата. Предвижда се класовете, които го реализират да имат или не ограничение на броя на елементите на структурата. Съответно на това всеки метод съществува в две форми: в едната се предизвиква изключение при некоректно приключване на метода, а при другата се връща специална стойност.

	First Element (Head)		**Last Element (Tail)**	
	Throws exception	*Special value*	*Throws exception*	*Special value*
Insert	`addFirst(e)`	`offerFirst(e)`	`addLast(e)`	`offerLast(e)`
Remove	`removeFirst()`	`pollFirst()`	`removeLast()`	`pollLast()`
Examine	`getFirst()`	`peekFirst()`	`getLast()`	`peekLast()`

Задача: Да се състави клас на Java, който реализира интерфейса **Deque** в два варианта: със и без горна граница на броя на елементите на структурата.