Laboratory 6: Reverse Polish Notation

(http://people.f4.htw-berlin.de/~weberwu/info2/labs/Exer6.shtml)

1. Implement a class `Stack.java` as discussed in the lecture, using a linked list of objects that you implement yourself! Don't use the Stack or LinkedList that is available by default in Java. Try and type it in yourself, not just copy the handout. How will you test this? Your class should include both an exception on stack underflow as well as stack overflow. Will you really need both exceptions? Why or why not? Override the `toString()` method to provide a useful way of printing a stack. Now make it generic, so it can take values of any type. Coordinate your interface with your partner.

We create a new class Stack in which we implement the methods we need for the stack. The methods we need are push(), pop(), top(), empty() and toString():
The push() method, which creates a new list element and shifts some references:

```
public void push(T o) {
    //Wenn noch nichts im Stack, gebe leerem Stackelement einen Inhalt
    if(top == null) {
        top = new Node<T>();
        top.setValue(o);
    }else {
        //Erstelle neues Stackelement
        Node<T> newNode = new Node<T>();
        //Binde neues Stackelement ein
        newNode.next = top;
        top = newNode;
        //FÃ¼ge Inhalt in Stackelement
        top.setValue(o);
    }
}
```

The pop() method, which saves the element to be removed, removes it and returns the saved element:

```
public Object pop() throws Exception {
    //Wenn Stack leer
    if(top == null) {
        throw new Exception("UnderflowException");
    }
    //Speichere oberstes Stackelement
    Node<T> removedNode = top;
    //Entferne das Element
    top = top.next;
    //Gib den Inhalt des gespeicherten obersten Elements zurÃ¼ck
    return removedNode.getValue();
}
```

The top() method simply returns the value of the top element:

```
public Object top() throws Exception {
    //Wenn Stack leer
    if(top == null) {
        throw new Exception("UnderflowException");
    }
    //Gebe Inhalt des obersten Elements zurÃ¼ck
    return top.getValue();
}
```

The empty() method, which checks if the top variable references to any object:

```
public boolean empty() {
    return (top == null);
}
```

And the toString() method simply prints the value of each element of the stack. For example: “Your stack contains of 3 2 1”:

```
public String toString() {
    Node<T> current = top;
    String text = "Your stack contains of ";
    //So lange noch ein Element vorhanden ist
    while(current != null) {
        //FÃ¼ge den Inhalt des Elements zum String hinzu
        text += current.getValue() + " ";
        //WÃ¤hle das nÃ¤chste Element aus
        current = current.next;
    }
    return text;
}
```

All the methods pretty much explain themselves. Now we go on to task 2 and build the Postfix class to see if everything works.
Since a stack cannot have an overflow exception we only had to throw underflow exceptions, if we try to access the top element of the stack and (top == null) evaluates to true. This needs to be checked in pop() and top().

2. Implement a class `Postfix.java` that has a method `public int evaluate (String pfx){...}` that takes a `String` representing a postfix expression and determines the value represented by that expression. You will need to access the individual characters of the string and store them in a stack. This is necessary for the evaluation, luckily your partner is currently in the process of making one. Build a test class and check the postfix expressions you did in the finger exercises. If there is a difference between the value computed and the value expected, either you were wrong, or the implementation is wrong or both.

We create a new class Postfix with a method called “evaluate”, which takes an postfix String through its parameters, to calculate the result and return it.
At first we need to create a new object stack from class Stack.
Further on we take the whole String and put it in an character array, so every single digit and operator can be accessed on its own.
Now we check each character in the array if it is a digit or an operand. Digits are going to be pushed to the stack and if an operator appears we take two of the digits from the stack and evaluate them and push the result onto the stack. At the end of the for loop we return the result.
The algorithm used is the one from Dr. Weber-Wullfs homepage (http://people.f4.htw-berlin.de/~weberwu/info2/Handouts/Postfix-evaluation.html).

```
        public int evaluate(String pfx) throws Exception{
            Stack<Object> stack = new Stack<Object>();
            int result = 0;
            int rhs = 0;
            int lhs = 0;

            char[] c = pfx.toCharArray();

            //FÃ¼r jedes Element im Array
            for (char token:c) {
                //Wenn der nÃ¤chste Token eine Zahl ist
                if (Character.isDigit(token)) {
                    stack.push(token);
                } else{
                    //Nehme die ersten beiden Elemente vom Stack
                    rhs = Integer.parseInt(stack.pop().toString());
                    lhs = Integer.parseInt(stack.pop().toString());

                    //Identifiziere den Operator und berechne das Ergebnis
                    if(token == '*') {
                        result = lhs*rhs;
                    } else if(token == '/') {
                        result = lhs/rhs;
                    } else if(token == '+') {
                        result = lhs+rhs;
                    } else if(token == '-') {
                        result = lhs-rhs;
                    }
                    stack.push(result);
                }
            }
            return result;
        }
```

Now we create a class Test:

```
public class Test {

    public Test() {
        Postfix pf = new Postfix();
        try {
            System.out.println(pf.evaluate("34+"));
        } catch (Exception e) {}
    }

    public static void main (String[] args){
        new Test();
    }
}
```

We let the program run, it sends "34+" to method evaluate() of class Postfix and returns 7 in the console. The calculation works!

3. Now add another method to the `Postfix.java` class `public String infixToPostfix (String ifx){...}` that converts an infix expression which is presented as a `String` to a `String` representing a postfix expression! Throw an exception if your input is not well-formed.

At first we tried to write the code from the pseudo code on Weber-Wullfs page (http://people.f4.htw-berlin.de/~weberwu/info2/Handouts/Postfix-evaluation.html), but we couldn't get it running form some reason. So we decided to check the internet for another description of an infix to postfix algorithm. And we found some on the page http://geekswithblogs.net/venknar/archive/2010/07/09/algorithm-for-infix-to-postfix.aspx:

Algorithm for Infix to Postfix

One of the applications of stack is in the evaluation of arithmetic expressions. To evaluate any arithmetic expression we convert the infix expression to postfix. Then evaluate the postfix expression using a stack. In this article I would define the standard algorithm for this.

Define a stack
Go through each character in the string
If it is between 0 to 9, append it to output string.
If it is left brace push to stack
*If it is operator *+-/ then*
    If the stack is empty push it to the stack
    If the stack is not empty then start a loop:
        If the top of the stack has higher precedence
        Then pop and append to output string
        Else break
    Push to the stack

If it is right brace then
    While stack not empty and top not equal to left brace
    Pop from stack and append to output string
    Finally pop out the left brace.

If there is any input in the stack pop and append to the output string.

We were able to translate that into Java code and finally it worked.

```
public String infixToPostfix(String ifx) throws Exception {
    Stack<Character> stack = new Stack<Character>();
    char[] characters = ifx.toCharArray();
    String postfix = "";

    for(char t:characters) {
        if(Character.isDigit(t)) {
            postfix += t;
        }else if(t == '(') {
            stack.push(t);
        }else if(t == '+' || t == '-' || t == '*' || t == '/') {
            if(stack.empty()) {
                stack.push(t);
            }else {
                while(!stack.empty() && highPriority(stack.top()) && !highPriority(t)) {
                    postfix += stack.pop();
                }
                stack.push(t);
            }
        }else if(t == ')') {
            while(!stack.empty() && stack.top() != '(') {
                postfix += stack.pop();
            }
            stack.pop();
        }
    }
    while(!stack.empty()) {
        postfix += stack.pop();
    }
    return postfix;
}
```

We made a little helping method to check out, if the operator has highPriority or not, with the method *highPriority()*:

```
private boolean highPriority(char c) {
    return (c == '*' || c == '/');
}
```

4. Now add another method that reads a string from the console, evaluates the result and prints the result to the console.

We create a new method in class Postfix named evaluatePostfixUserInput().
Then we import the scanner from the Java library. The scanner reads the user input and saves everything in a String userInput. Now we pass the String to the evaluate() method, a try/catch observes if everything went well and either the result or an error message gets printed out.

```
public void evaluatePostfixUserInput() {
    Scanner sc = new Scanner(System.in);
    System.out.print("Postfix: ");
    String userInput = sc.nextLine();
    try {
        System.out.println("Result: " + evaluate(userInput));
    }catch(Exception e) {
        System.out.println("Input falsch formatiert.");
    }
}
```

We test the new method and everything works!

All in all the lab took us about 5 hours of work.
We learned how a stack works and how to change math algorithms to Java algorithms.

Source code:

Class Stack:

```
/**
 * Ex06
 * @author jw & ma
 * @since 19.11.2012
 */
public class Stack<T> {
	private Node<T> top;

	public Stack() {
	}

	/**
	 * FÃ¼gt ein neues Element dem Stack hinzu.
	 */
```

```
public void push(T o) {
//Wenn noch nichts im Stack, gebe leerem Stackelement einen Inhalt
if(top == null) {
	top = new Node<T>();
	top.setValue(o);
}else {
	//Erstelle neues Stackelement
	Node<T> newNode = new Node<T>();
	//Binde neues Stackelement ein
	newNode.next = top;
	top = newNode;
	//FÃ¼ge Inhalt in Stackelement
	top.setValue(o);
}
}

/**
 * Gibt Inhalt des obersten Elements im Stack und entfernt es.
 */
public Object pop() throws Exception {
//Wenn Stack leer
if(top == null) {
	throw new Exception("UnderflowException");
}
//Speichere oberstes Stackelement
Node<T> removedNode = top;
//Entferne das Element
top = top.next;
//Gib den Inhalt des gespeicherten obersten Elements zurÃ¼ck
return removedNode.getValue();
}

/**
 * Gibt oberstes Element im Stack zurÃ¼ck ohne es zu entfernen.
 */
public T top() throws Exception {
//Wenn Stack leer
if(top == null) {
	throw new Exception("UnderflowException");
}
//Gebe Inhalt des obersten Elements zurÃ¼ck
return top.getValue();
}

/**
 * Gibt zurÃ¼ck ob der Stack leer ist.
 */
public boolean empty() {
return (top == null);
}

/**
 * Gibt den Inhalt des Stacks zurÃ¼ck.
 */
public String toString() {
Node<T> current = top;
String text = "Your stack contains of ";
```

```
		//So lange noch ein Element vorhanden ist
		while(current != null) {
			//FÃ¼ge den Inhalt des Elements zum String hinzu
			text += current.getValue() + " ";
			//WÃ¤hle das nÃ¤chste Element aus
			current = current.next;
		}
		return text;
		}
}
```

Class Postfix:

```
import java.util.Scanner;

/**
 * Ex06
 *
 * @author jw & ma
 * @since 19.11.2012
 */

public class Postfix {

		//Aufgabe 2
		public int evaluate(String pfx) throws Exception{
		Stack<Object> stack = new Stack<Object>();
		int result = 0;
		int rhs = 0;
		int lhs = 0;

		char[] c = pfx.toCharArray();

		//FÃ¼r jedes Element im Array
		for (char token:c) {
			//Wenn der nÃ¤chste Token eine Zahl ist
			if (Character.isDigit(token)) {
				stack.push(token);
			} else{
				//Nehme die ersten beiden Elemente vom Stack
				rhs = Integer.parseInt(stack.pop().toString());
				lhs = Integer.parseInt(stack.pop().toString());

				//Identifiziere den Operator und berechne das Ergebnis
				if(token == '*') {
					result = lhs*rhs;
				} else if(token == '/') {
					result = lhs/rhs;
				} else if(token == '+') {
					result = lhs+rhs;
				} else if(token == '-') {
					result = lhs-rhs;
				}
```

```
                        stack.push(result);
                }
        }
        return result;
        }

        //Aufgabe 3 (
http://geekswithblogs.net/venknar/archive/2010/07/09/algorithm-for-infix-to-
postfix.aspx )
        public String infixToPostfix(String ifx) throws Exception {
        Stack<Character> stack = new Stack<Character>();
        char[] characters = ifx.toCharArray();
        String postfix = "";

        for(char t:characters) {
                if(Character.isDigit(t)) {
                        postfix += t;
                }else if(t == '(') {
                        stack.push(t);
                }else if(t == '+' || t == '-' || t == '*' || t == '/') {
                        if(stack.empty()) {
                                stack.push(t);
                        }else {
                                while(!stack.empty() && highPriority(stack.top()) &&
!highPriority(t)) {
                                        postfix += stack.pop();
                                }
                                stack.push(t);
                        }
                }else if(t == ')') {
                        while(!stack.empty() && stack.top() != '(') {
                                postfix += stack.pop();
                        }
                        stack.pop();
                }
        }
        while(!stack.empty()) {
                postfix += stack.pop();
        }
        return postfix;
        }

        private boolean highPriority(char c) {
        return (c == '*' || c == '/');
        }

        //Aufgabe 4
        public void evaluatePostfixUserInput() {
        Scanner sc = new Scanner(System.in);
        System.out.print("Postfix: ");
        String userInput = sc.nextLine();
        try {
                System.out.println("Result: " + evaluate(userInput));
        }catch(Exception e) {
                System.out.println("Input falsch formatiert.");
        }
        }
```

```
}
```

Class Test:

```
/**
 * Ex06
 * @author jw & ma
 * @since 19.11.2012
 */

public class Test {

	public Test() throws Exception {
	/*Stack testStack = new Stack();

	System.out.println("> Push 1,2,3");
	testStack.push("1");
	System.out.println(testStack.toString());
	testStack.push("2");
	System.out.println(testStack.toString());
	testStack.push("3");
	System.out.println(testStack.toString());
	*/
	Postfix pf = new Postfix();
	try {
		//System.out.println(pf.evaluate("12-1+"));
		//pf.evaluatePostfixUserInput();
	}catch(Exception e) {}

	System.out.println(pf.evaluate(pf.infixToPostfix("(1+2)+3*4-6")));
	}

	public static void main (String[] args) throws Exception{
	new Test();
	}
}
```

Class Node:

```
public class Node<T> {
	public Node<T> next;
	private T o;

	public T getValue() {
	return o;
	}

	public void setValue(T o) {
	this.o = o;
	}
```

```
}
```