Lab Report 6 – 22.11.2011 – Sebastian Denzer (533251) – Andy Lindemann (533200)

1.

Sebastian

I started with the idea of two main Nodes, the front and the current Node: each Node should hold its value (for now only Strings; later on Objects), and lastNode and nextNode references to the corresponding Nodes, plus their getters and setters. When implementing all code, I finally figured out that no front Node was needed: we only access the top object of the stack, and never have to deal with the first one, except for checking if there's any other lastNode.

Now for the methods. The push() method shall take a String and create a newNode with it. Now it looks if the current Node is not null, then sets currents' nextNode to the newNode and newNodes' lastNode to current. If there was a current Node or not, it sets current to the newNode. So the whole stack is build up from Node to Node.

Andy and me decided to use a top() and a pop() method, having the ability to ask for the last one again and again, and then deleting it. top() simply returns the currents' Nodes' String.

We first discussed another Exception throwing for the pop() method if the front Node was reached. But seen from the outside, it is not of interest if there is anything deleted. We simply don't know it. As you ask for an Exception in the exercises, I implemented it on the pop() method - but Andy and me both agree that for our problems, we don't need any Exceptions. If the Stack is empty, top() returns null, and if pop() is called one time too often (and Stack is empty), it doesn't do anything... no need for an Exception?!

At this point, I was pretty sure to not use the StackOverflowException... where should it come from, if there is no reason for it to be thrown? (OK, maybe lack of memory.)

I finally added an isEmpty() method, looking if current is null.

The generic version is simply done by "converting" each String from "String" to "E". The class itself (using the generic) must have "<E>" behind its name in the class signature. Of course this has to be passed through from the Node to the Stack itself.

2.

Andy

Implementing the evaluation method for the postfix notation was rather easy (especially compared to the later realization of the infix to postfix conversion).

While Sebastian was working on the stack I knew which methods he would write and how I would be able to use them.

I started by simply putting the input string char by char into a stack starting from the end. All I had to do then was to take those parts of the stack and check wether they are numbers or operators. Numbers went straight onto the number stack, operators where applied to the top two numbers and the result was pushed back on the stack. When there was nothing left on the input stack, the top of the number stack was returned.

When the stack finally worked there was just a minor flaw as I accidentally used == instead of equals() to compare the operator strings. I discovered and fixed this rather quickly. Stupid me.

3.

As I mentioned before, implementing the infix to postfix conversion was a bit more difficult. It was no problem in the prelab, so I understood the basic algorithm.

First I put the whole input string onto a stack, starting from the last character, omitting spaces. I worked my way down the stack. If it was number, it was popped onto the return string.

If it was an operator I had to apply the rules i had applied in my head in the prelab. Check if the stack was empty, if it was push the operator on the operator stack. If it wasn't empty, i had to compare the precedence of the current operator from the input stack with the one from the operator stack. I implemented a method to get the precedence of an operator. As in some cases you have to work your way down in the operator stack, I used a while loop which was run as long as the top of the operator stack was of higher precedence then the current operator from the input and as long as the operator stack wasn't empty.

I am not going to describe the rest of the implementation, as I assume that you know very well how it works and it also can be seen in the source code.

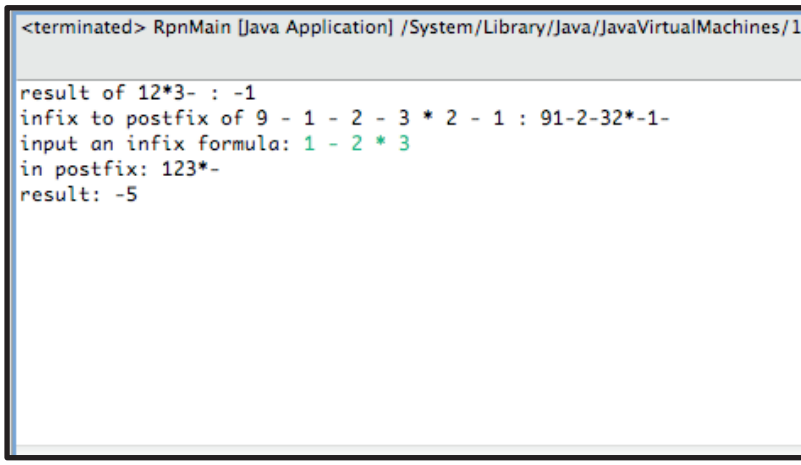
I had to do several tests to get it to return the correct postfix notation. First I forgot to handle the case of two operators with the same precedence, which need to be treated differently if they where "^". Then I had the problem that in some cases some operators of lower precedence disappeared magically. That was simply due to the fact that i didn't push them to the Operator stack once I worked my way down to the bottom of it and the stack was empty. Stupid me again.

In both methods I implemented a basic error handling as they throw an InvalidExpressionException if there are chars in the string which cannot be processed.

4.

Building the user input was relaxing after the brain crunching of the last method. I use a Scanner to read the user input, convert it to postfix and evaluate the resulting string.

File	Lines of Code
InvalidExpressionException.java	6
TestStack.java	47
Stack.java	35
Node.java	24
StackUnderflowException.java	6
PostFix.jaja	172
RpnMain.java	20
TextPostfix.java	42



```java
package rpn;

public class Stack <E> {
    private Node<E> current;

    public Stack () {
        this.current = null;
    }

    public void push (E obj) {
        Node<E> newNode = new Node<E> (obj);
        if (current != null) {
            this.current.setNextNode(newNode);
            newNode.setLastNode(this.current);
        }
        this.current = newNode;
    }

    public E top (){
        if (current != null)  {
            return current.getObject();
        }
        else {
            return null;

        }
    }

    public void pop () throws StackUnderflowException {

        if (this.current != null) {
                this.current = this.current.getLastNode();
        }
        else {
            throw new StackUnderflowException();
        }

    }

    public boolean isEmpty() {
        return (this.current == null);
    }



    public String toString() {
        Node n = current;
        String str = "";
        int count = 0;
        while (n != null) {
            str = str + count +": "+ n.getObject().toString() +"\n";
            n = n.getLastNode();
            count++;
        }
        str = "Stack content: \n"+str;
        return str;
    }
}

```

```
 1 package rpn;
 2
 3 public class Node<E> {
 4
 5     private Node<E> nextNode;
 6     private Node<E> lastNode;
 7     private E object;
 8
 9     public Node(E obj) {
10         this.object = obj;
11
12     }
13
14     public E getObject() {
15         return object;
16     }
17
18     public Node<E> getNextNode() {
19         return nextNode;
20     }
21
22     public void setNextNode(Node<E> nextNode) {
23         this.nextNode = nextNode;
24     }
25
26     public Node<E> getLastNode() {
27         return lastNode;
28     }
29
30     public void setLastNode(Node<E> lastNode) {
31         this.lastNode = lastNode;
32     }
33
34     public Node<E> getFrontNode() {
35
36         if (this.getLastNode() == null) {
37             return this;
38         }
39         else {
40             return getLastNode().getFrontNode();
41         }
42
43     }
44 }
45
```

```java
package rpn;

import java.util.Scanner;

/**
 * @author Andy
 *
 */
public class Postfix {

    /**
     * Evaluates a postfix expression.
     *
     * @param pfx
     *            The expression to be evaluated.
     * @return the result of the expression.
     * @throws InvalidExpressionException
     *             when there are invalid characters in the given string.
     */
    public static int evaluate(String pfx) throws InvalidExpressionException {

        Stack<String> pfxStack = new Stack<String>();
        Stack<Integer> numberStack = new Stack<Integer>();
        for (int i = pfx.length() - 1; i >= 0; i--) {
            String currPfx = String.valueOf(pfx.charAt(i));
            if (!currPfx.equals(" ")) {
                pfxStack.push(currPfx);
            }
            }

        while (!pfxStack.isEmpty()) {
            String currentPfx = "";
            try {
                currentPfx = pfxStack.top();
                pfxStack.pop();
            } catch (Exception e) {
                System.out.println(e.getMessage());
            }
            if ("0123456789".contains(currentPfx)) {
                numberStack.push(new Integer(Integer.parseInt(currentPfx)));
            } else if ("-+*/%^".contains(currentPfx)) {
                try {
                    int rhs = numberStack.top().intValue();
                    numberStack.pop();
                    int lhs = numberStack.top().intValue();
                    numberStack.pop();
                    Integer result = 0;
                    if (currentPfx.equals("-"))
                        result = lhs - rhs;
                    if (currentPfx.equals("+"))
                        result = lhs + rhs;
                    if (currentPfx.equals("*"))
                        result = lhs * rhs;
                    if (currentPfx.equals("/"))
                        result = lhs / rhs;
                    if (currentPfx.equals("%"))
                        result = lhs % rhs;
                    if (currentPfx.equals("^"))
                        result = (int) Math.pow((double) lhs, (double) rhs);
                    numberStack.push(new Integer(result));
                } catch (Exception e) {
                    System.out.println(e.getMessage());
                }
            } else {
                throw new InvalidExpressionException();
            }
        }

        int returnInt = 0;
        try {
            returnInt = numberStack.top().intValue();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return returnInt;
    }

    /**
     * Converts a infix expression to a postfix expression.
     *
     * @param ifx
     *            The infix expression to be converted.
```

```
 83      * @return The generated postfix expression.
 84      * @throws InvalidExpressionException
 85      *             when there are invalid characters in the given string.
 86      */
 87     public static String infixToPostfix(String ifx)
 88             throws InvalidExpressionException {
 89         Stack<String> ifxStack = new Stack<String>();
 90         Stack<String> opStack = new Stack<String>();
 91         String returnString = "";
 92         for (int i = ifx.length() - 1; i >= 0; i--) {
 93             String currIfx = String.valueOf(ifx.charAt(i));
 94             if (!currIfx.equals(" ")) {
 95                 ifxStack.push(currIfx);
 96             }
 97         }
 98         while (!ifxStack.isEmpty()) {
 99             String currIfx = "";
100             try {
101                 currIfx = ifxStack.top();
102                 ifxStack.pop();
103             } catch (Exception e) {
104                 System.out.println(e.getMessage());
105             }
106             if ("0123456789".contains(currIfx)) {
107                 returnString += currIfx;
108             } else if ("-+*/%^".contains(currIfx)) {
109                 if (opStack.isEmpty()) {
110                     opStack.push(currIfx);
111                 } else {
112                     boolean stackIsRelevant = (getPrecedence(opStack.top()) >= getPrecedence(currIfx));
113                     if (!stackIsRelevant)
114                         opStack.push(currIfx);
115                     while (stackIsRelevant && !opStack.isEmpty()) {
116                         if (getPrecedence(opStack.top()) > getPrecedence(currIfx)) {
117                             returnString += opStack.top();
118 
119                             try {
120                                 opStack.pop();
121                             } catch (Exception e) {
122                                 System.out.println(e.getMessage());
123                             }
124                             if (opStack.isEmpty())
125                                 opStack.push(currIfx);
126                         } else if (getPrecedence(opStack.top()) == getPrecedence(currIfx)) {
127                             if (currIfx.equals("^")) {
128                                 opStack.push(currIfx);
129                                 stackIsRelevant = false;
130                             } else {
131                                 returnString += opStack.top();
132                                 try {
133                                     opStack.pop();
134                                 } catch (Exception e) {
135                                     System.out.println(e.getMessage());
136                                 }
137                                 opStack.push(currIfx);
138                                 stackIsRelevant = false;
139 
140                             }
141 
142                         }
143                     }
144                 }
145             } else {
146                 throw new InvalidExpressionException();
147 
148             }
149 
150         }
151         while (!opStack.isEmpty()) {
152             returnString += opStack.top();
153             try {
154                 opStack.pop();
155             } catch (StackUnderflowException e) {
156                 System.out.println(e.getMessage());
157             }
158 
159         }
160 
161         return returnString;
162 
163     }
164 
```

```
165     private static int getPrecedence(String op) {
166         int returnInt = 4;
167         if (op.equals("+") || op.equals("-"))
168             returnInt = 1;
169         if (op.equals("*") || op.equals("/") || op.equals("%"))
170             returnInt = 2;
171         if (op.equals("^"))
172             returnInt = 3;
173         return returnInt;
174 
175     }
176 
177     /**
178      * Runs a user input on the console asking for a infix expression. It is
179      * then converted into postfix and evaluated. The result is printed to the
180      * console.
181      */
182     public static void userInput() {
183         String inputString = "";
184         String pfx = "";
185         int result = 0;
186         Scanner sysin = new Scanner(System.in);
187         System.out.print("input an infix formula: ");
188         inputString = sysin.nextLine();
189         try {
190             pfx = infixToPostfix(inputString);
191         } catch (InvalidExpressionException e) {
192             System.out.println(e.getMessage());
193         }
194         System.out.println("in postfix: " + pfx);
195         try {
196             result = evaluate(pfx);
197         } catch (InvalidExpressionException e) {
198             System.out.println(e.getMessage());
199         }
200         System.out.println("result: " + result);
201 
202     }
203 }
204 
```

```java
package rpn;

public class RpnMain {

    public static void main(String[] args) {
        String pfx1 = "12*3-";
        String pfx2 = "9 - 1 - 2 - 3 * 2 - 1";
        try {
            System.out.println("result of " + pfx1 + " : "
                    + Postfix.evaluate(pfx1));
        } catch (InvalidExpressionException e) {
            System.out.println(e.getMessage());
        }

        try {
            System.out.println("infix to postfix of " + pfx2 + " : "
                    + Postfix.infixToPostfix(pfx2));
        } catch (InvalidExpressionException e) {
            System.out.println(e.getMessage());
        }
        Postfix.userInput();
    }

}
```

```java
package rpn;

public class InvalidExpressionException extends Exception{

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public InvalidExpressionException(){
    }

    public String getMessage(){
        return ("No valid expression.");
    }

}

```

```java
package rpn;

public class StackUnderflowException extends Exception {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public StackUnderflowException() {

    }
    public String getMessage(){
        return ("Stack underflow error.");

    }

}

```

```java
package rpn;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class TestStack {
    Stack<String> st;

    @Before
    public void setUp() {

        st = new Stack<String>();

    }

    @Test
    public void test() {
        st.push("A");
        assertEquals("A", st.top());
        st.push("B");
        assertEquals("B", st.top());
        st.push("C");
        assertEquals("C", st.top());

        try {
            st.pop();
        }
        catch (StackUnderflowException e1) {
            e1.printStackTrace();
            fail("Should have NOT thrown an Exception.");
        }

        assertEquals("B", st.top());

        try {
            st.pop();
        }
        catch (StackUnderflowException e) {
            System.out.println(e.getMessage());
            fail("Should have NOT thrown an Exception.");
        }

        assertEquals("A", st.top());

        try {
            st.pop();
        }
        catch (StackUnderflowException e) {
            System.out.println(e.getMessage());
            fail("Should have NOT thrown an Exception.");
        }

        assertEquals(null, st.top());

        try {
            st.pop();
            fail("Should have thrown an Exception.");
        }
        catch (StackUnderflowException e) {
            System.out.println(e.getMessage());
        }
    }

}
```

```java
package rpn;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class TestPostfix {
    Stack<String> st;

    @Before
    public void setUp() {

    }

    @Test
    public void testString() {
        try {
            assertEquals(5, Postfix.evaluate("12*3+"));
            assertEquals(-78, Postfix.evaluate("12+34^-"));
            assertEquals(-11, Postfix.evaluate("12^34*-"));
            assertEquals(9, Postfix.evaluate("12+3*456-^+"));
            assertEquals(98, Postfix.evaluate("12+34/+5+678+*+"));
            assertEquals(-1, Postfix.evaluate("91-2-32*-1-"));
            assertEquals(-1011, Postfix.evaluate("123*+45^-6+"));
        }
        catch (InvalidExpressionException e) {
            System.out.println(e.getMessage());
        }



        try {
            assertEquals(-1011, Postfix.evaluate("A23*+45^-6+"));
            fail("Should have thrown an Exception.");
        }
        catch (InvalidExpressionException e) {
            System.out.println(e.getMessage());
        }
        try {
            assertEquals(-1011, Postfix.evaluate(" "));
            fail("Should have thrown an Exception.");
        }
        catch (InvalidExpressionException e) {
            System.out.println(e.getMessage());
        }
        try {
            assertEquals(-1011, Postfix.evaluate("90&"));
            fail("Should have thrown an Exception.");
        }
        catch (InvalidExpressionException e) {
            System.out.println(e.getMessage());
        }
        try {
            assertEquals(-1011, Postfix.evaluate("%$ U"));
            fail("Should have thrown an Exception.");
        }
        catch (InvalidExpressionException e) {
            System.out.println(e.getMessage());
        }
    }
}

```