**Lab Sheet 1**

**Lab Task 1 (Arithmetic Expressions - Stack Algorithms)**

**Regulations and Lab Report Uploads**

* **Preparation and presentation at the beginning of the Labs:** Implementation and test code have to be presented in the beginning of the lab. During Lab time further test cases, debugging and error handling has to done. If your presentation fails, you loose that lab but you are allowed to stay, work and learn. You are allowed to miss one of the seven labs this way.
* **Lab Teams and Responsibilities:** A lab team consists of 2 people. There are 2 problems, one for each member. Decide in the beginning, who will be responsable for which task. This person makes the presentation in the beginning of the lab, comments the source code, works out the test cases, writes the report for that task and uploads it.Regulation for single workers on request.
* **Report and Plagiarism:** Provide a short document where you identify what you did on your own and what help and input you needed and used. All ressouces used have to be cited in the documentation AND in the source-code, in particular what is taken from a collegue or the web.Text and code has to be quoted properly.  *Recommendation:* Right from the start store all used URLs including the access-time in your document working directory.
* **Missing deadlines:** Please obey the DEADLINE for uploads into the EMIL system, usually 6 days after the Lab Date. Repeated uploads are possible, a later upload replaces the earlier one. Each working group is allowed to miss the deadline by three days only once in the term.
* **Private Labtops:** The use of private laptops is encouraged. WLAN access is provided.

Please, upload a **ZIP-file** with the following content:

1. Source code directories with a project file for compilation - Documentation tells with which compiler and IDE and which programs to run,
2. Documentation as a single file.

**Problem 1 (Evaluation of Postfix Expressions)**

  Look at the explainations, code examples and programs presented in the lecture.

* Provide a *class ArithmeticTerm* with a constructor that takes a String with the arithmetic expression as an argument and stores it in a private instance variable. Test this constructor by instantiating an ArithmeticTerm in the *main* program of that class.
* Add a method *String toString()* to that class which returns the arithmetic expression again. Test this by printing an instance of an ArithmeticTerm in the main program of that class.
* Add a method *void reverse()* to that class which uses a stack of Strings (java.util.Stack<String>) to store the tokens of the arithmetic expression (generated with the java.util.StringTokenizer) and writes it in the opposite order to the string instance variable. Test it by extending your main program accordingly.
* Now provide a method *Double evaluate()* to that class which uses a stack of Doubles (java.util.Stack<Double>) to evaluate the postfix expression with the algorithm [postfix2value](http://www.elearning.haw-hamburg.de/mod/resource/view.php?id=704493).Test it by extending your main program accordingly.
* Documentation shall include support of all 5 binary arithmetic operators and provide more than 5 test caes with at least 4 operators in different sequences each.
* Finally, include your partners method *String convert()*of problem 2 into your class, make it run. Write a static main2 method that converts a FPAE and calculates its value.

**Optional:**Your program has to terminate with an appropriate error message if the postfix expression is not valid, i.e. if operands cannot be converted to the required arithmetic data type or if there are not enough operands available to be operated (pop is applied to an empty stack) or if operands are left on the stack at the end of the expression.

**Problem 2 (Generation of Postfix from Infix Expressions)**

The conversion of a Fully Parenthesized Arithmetic Infix Expression (FPAE) to Postfix (Reverse Polish) Notation is simply done by using an operator stack (Stack<String>). Look at the explainations, code examples and programs presented in the lecture.

* Provide a *class ArithmeticTerm* with a constructor that takes a String with the arithmetic expression as an argument and stores it in a private instance variable. Test this constructor by instantiating an ArithmeticTerm in the *main* program of that class.
* Add a method *String toString()* to that class which returns the arithmetic expression again. Test this by printing an instance of an ArithmeticTerm in the main program of that class.
* Add a method *void reverse()* to that class which uses a stack of Strings (java.util.Stack<String>) to store the tokens of the arithmetic expression (generated with the java.util.StringTokenizer) and writes it in the opposite order to the string instance variable. Test it by extending your main program accordingly.
* Now provide a method *String convert()* to that class which uses a stack of Strings (java.util.Stack<String>) to convert the FPAE into a postfix expression with the algorithm [infix2postfix](http://www.elearning.haw-hamburg.de/mod/resource/view.php?id=704494).Test it by extending your main program accordingly, e.g.  with ( ( 4.3 \* 1e-1 ) - .4 ).
* Documentation shall include support of all 5 binary arithmetic operators and provide more than 5 test caes with at least 4 operators in different sequences each.
* Finally, include your partners method *Double evaluate()*of problem 1 into your class, make it run. Write a static main2 method that converts a FPAE and calculates its value.

**Optional:**Your program has to terminate with an appropriate error message if the FPAE is not syntactically valid, i.e. if there are too many or too few right parenthesis (pop is applied to an empty stack or operators are left on the stack at the end of the expression) and if not exactly three tokens are inside a pair of parenthesis, the second of which is one of the 5 binary operators.

**Bonus Task (Calculator with GUI)**

Starting from the [PostfixCalculatorDummy:](http://www.elearning.haw-hamburg.de/mod/resource/view.php?id=704576)

1. Make it work for the binary operators represented in the GUI by implementing the required CalculatePostfix methods. In case of questions publish them in the [user forum](http://www.elearning.haw-hamburg.de/mod/forum/view.php?id=688360), please.
2. Extend the GUI as well as the required methods of the underlying classes to support all operators you would like to use.