Assignment 1: Calculator Component

ETH Zurich

Hand-out:	18 February 2013
Due:	To receive feedback from the teaching assistant, please hand in your Java implementation before Feb. 25, 2013 and C# implementation before Mar. 4, 2013.

1 Description

The aim of this exercise is to write a Reverse Polish notation (RPN) based calculator. In the RPN, also called the postfix notation, the operands precede the operator. The notation has no need for parentheses. For example the expression 3 * (4 + 7) would be written as

```
3
4
7
+
*
```

Make sure to properly use the object-oriented facilities such as encapsulation, inheritance, and design patterns to implement the calculator and try to implement it as cleanly as possible. Goal of this exercise is to focus on the technical aspects of software development without being distracted by the complexity of the domain.

2 Task 1: Coding

In this assignment you should design your calculator as a re-usable component that must implement a certain interface and behavior. To ensure this, we created a JUnit and a NUnit test case. You can find them both on the course homepage. Following a *test-first* approach, we do not specify the detailed interface here; instead, you are supposed to extract the exact signatures for the corresponding methods from the test case.

You should try to make your solution pass all tests with an *unmodified* test file.

The calculator must provide at least the following basic functionality:

push operand: Push an operand of type double onto the stack.

add +: Adds top two elements of the stack.

subtract -: Substracts top-most entry from second top-most entry of the stack.

multiply *: Multiplies top two elements of the stack.

divide /: Divides second top-most entry by top-most entry of the stack.

evaluate: Evaluates expressions on stack.

pop result: Pop the topmost element and return a double as result.

peek result: Like pop result, but does not remove the element from the stack.

Exceptions: Throw exceptions to indicate illegal states during evaluation.

Your solution can consist of one or several classes.

You do not need to provide any input or output facilities, because the calculator is supposed to be used by client classes having their own I/O mechanism.

Note that the calculator should be implemented with lazy evaluation. This means that operations are not evaluated as soon as they are entered, but rather stored as elements on the stack, too. Only when the user enters the evaluation command, the expressions on the stack are evaluated and then the result is pushed back to the stack again.

3 Hints

- Info about RPN calculators at `http://en.wikipedia.org/wiki/Reverse_Polish_notation`.
- Info about the JUnit and NUnit test frameworks at `http://junit.org/` and `http://www.nunit.org/`.
- One possibility to implement lazy evaluation is to use the *visitor pattern*: A visitor could work on the stack (by subsequently popping and visiting the topmost elements) in order to compute and push back the results step-by-step.

4 Task 2: Design

Describe why you chose the design you have chosen. Why do you think that your design makes for a good reusable component? What did you do to make it easy for potential developers who need to use your calculator?

5 Submission

Zip your code into a single file and send it to your assistant via email. Make your email subject informative by including the course id, the assignment number, the implementation language, and your name. For example, for Tom Cruise to hand in his Java implementation for assignment 1, the email subject could be

[JCD]-1-Java-Tom Cruise

where "JCD" stands for "**J**ava and **C**# in **D**epth".

6 Contacts

Christian Estler	christian.estler@inf.ethz.ch
Alexey Kolesnichenko	alexey.kolesnichenko@inf.ethz.ch
Max (Yu) Pei	yu.pei@inf.ethz.ch
Julian Tschannen	julian.tschannen@inf.ethz.ch