Assignment 1

Instructor: Musard Balliu

September, 2015

Introduction

In this assignment, you will program a mini-calculator using Reverse Polish Notation (RPN) as an internal representation. RPN is a way of writing an expression, which is particularly handy when evaluating the expression. An expression is a sequence of characters. We will be dealing with arithmetic expressions only. Therefore, the characters used in expressions are digits $0, 1 \cdots, 9$, operators $+, -, *, /$ and the left (and right) parentheses. In the usual evaluation form, also called infix form, the operators are placed between their operands and the parentheses are used to clarify in which precedence order the operators will be applied. In infix notation, a precedence relation is specified where $*$ and $/$ have higher priority then $+$ and $-$; in the case of equal precedence, the operators are applied from left to right.

Examples: $2 + 3$, $4 * (2 + 3)$, $((3 + 5 * 1)/8) * 14$

In the reverse polish form (also called postfix), operators are placed after their operands. In this notation, no parentheses are needed:

Examples: $2\,3\,+$, $4\,2\,3\ +\ *$, $3\,5\,1\ *\ +\,8\,/\,14\,*$

Task 1: Conversion from Infix to RPN

We use the subroutine p to specify the priorities of the operators: $p(+) = 0$, $p(-) = 0$, $p(*) = 1$, $p(/) = 1$. This means that addition and subtraction have lower priority compared to multiplication and division.

Given an infix expression `c_1 . . . c_m`, where `c_i` is either an operand (in this case an integer value), an operator or a parentheses, the algorithm below outputs the expression in RPN. A helper string, which we call *tmp*, is used during this transformation.

```
for i=1 to m
    if c_i is an operand:          Transfer c_i to output.
    if c_i is a left parentheses:  Push c_i to tmp.
    if c_i is a right parentheses: Pop elements from tmp and transfer
                                   them to output until a left-parentheses
                                   is met. Pop left-parentheses.
    if c_i is an operator:         Let the top tmp element be t. Pop and
                                   transfer elements from tmp to output
                                   until:
                                       p(t) < p(c_i) or
                                       t is a left-parentheses or
                                       tmp is empty.
                                   Push c_i to  tmp.
Transfer the remaining elements in tmp to output.
```

Example

This example illustrates a run of the algorithm on the expression $((3+5*1)/8)*14$. The symbol ε is used to denote the empty string.

infix	tmp	output
$((3+5*1)/8)*14$	ε	ε
$(3+5*1)/8)*14$	$($	ε
$3+5*1)/8)*14$	$(($	ε
$+5*1)/8)*14$	$(($	3
$5*1)/8)*14$	$+(($	3
$*1)/8)*14$	$+(($	$3\ 5$
$1)/8)*14$	$*+(($	$3\ 5$
$)/8)*14$	$*+(($	$3\ 5\ 1$
$/8)*14$	$($	$3\ 5\ 1*+$
$8)*14$	$/($	$3\ 5\ 1*+$
$)*14$	$/($	$3\ 5\ 1*+8$
$*14$	ε	$3\ 5\ 1*+8\ /$
14	$*$	$3\ 5\ 1*+8\ /$
ε	$*$	$3\ 5\ 1*+8\ /\ 14$
ε	ε	$3\ 5\ 1*+8\ /\ 14*$

Task 2: Evaluating an expression in RPN

Given a postfix expression $v\_1 ... v\_n$, where v_i is either an operand or an operator, the following algorithm evaluates the expression. Again a helper string, *tmp2*, is used during the calculation.

```
i = 1
while i<= n
    if v_i is an operand:   Push v_i to tmp2.
    if v_i is an operator:  Apply v_i to the top two elements of
                            tmp2. Replace these by the result in tmp2.
    i = i + 1
Output result from tmp2.
```

Main Task

You will program a mini-calculator in JAVA, which takes an infix arithmetic expression of the type above and first converts this to RPN and then using the RPN expression evaluates it. Algorithms 1 and 2 will be implemented for this purpose. The input, output, as well as the helper variables will be strings in Java. You can assume that the input is a well-formed expression. The program should present the following interface to the user.

```
+***************************************+
*                                       *
*      Welcome to DIT948 Calculator     *
*                                       *
+***************************************+

Press  "E" to exit or any other button to continue> C

Please enter an arithmetic expression to evaluate> ((3 + 5 * 1) / 8) * 14

The RPN representation of your expression is> 3 5 1 * + 8 / 14 *

The final result is> 14

Press  "E" to exit or any other button to continue> E

Bye Bye
```

Remarks and Notation

1. Variables *tmp, tmp1* contain sequences of characters, namely strings
2. An *element* is a digit (for example 14), a parentheses or an operator
3. The *pop* operation removes the first element of the string each time it is applied. The *top* element of a string is the first element of the string. The *push* operation appends an element in the beginning of the string.
4. You can use the methods $Integer.parseInt(s)$ and $Character.getNumericValue(c)$ to convert a string s and a character c to an integer value, if needed.
5. If needed, you can use other methods of class String as explained in the lecture slides.
6. You are not allowed to use data structures other than Strings to manipulate the expressions.

Grading

- To get a G it is necessary that the program works for 1-digit operands only, including intermediate results and final results.
- To get a VG it necessary that the program works for any-digit operands, including intermediate results and final results.

  Note that these requirements are necessary, but not sufficient, to get the respective grades (see Administrative matters).

Administrative matters

Strive for readable code with appropriate comments! While the ultimate test of a program is that it does what it's supposed to do, **I should be able to read the program and understand it**.

The assignment is to be completed in groups of two students. (Groups of different size should first be agreed with the course instructor.)

It must be uploaded to the GUL system **by 23:59 on September 20**.

Please note: **The submission is to be made via GUL. It's no good sending it to me, either before or after the deadline!**

Only java source files should be uploaded, no class files. Your java files should be put in a zip-archive with the following name:

```
assign1_author1_author2.zip
```

where author1, author2 are the surnames (family names) of the group members. You must also supply a README-file (README.txt) containing the names of the authors and their social security numbers, together with a brief statement about each author's contribution. For example, "author1 has been responsible for user input and author2 for the program logic". A statement such as "All authors have contributed equally to all aspects of the program" is also acceptable.

Note that **each author must submit individually via the GUL** (even though it is the same program!). Only students who submit via the GUL can be graded. All comments etc. will be posted on GUL.

After the deadline has passed, each group **must present solutions to the TAs during the Thursdays supervision session**. Exceptions are to be agreed with the TAs. Group members are expected to know and explain all parts of the code despite their statement of contribution.