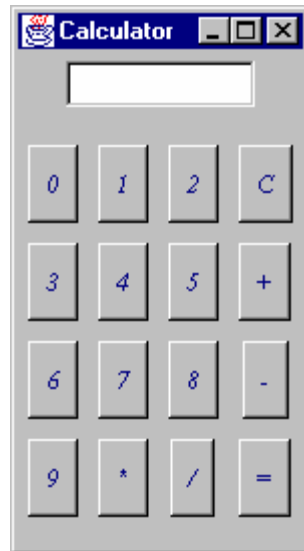


Lab 8. Event Handling

Mini Project

Write a program which implements the following window-based GUI of a simple pocket computer which uses the Polish (postfix) form¹ for expressions built with positive integer operands and four algebraic operations (+, -, *, and /) over integer numbers.



Add the events handling to the GUI in order to simulate the computation of expressions based on four simple binary operations: +, -, *, and / (without parentheses). The solution should use buttons as event sources and the listener should be an object of an internal class called `ActionEventController`.

Example

Let be $(12+2-16)*4$ an expression in infix² (normal) form. Its Polish form is $12\ 2\ +\ 16\ -\ 4\ *$. Here is an example of how the GUI works when this expression is entered:

Entering step	Pushed Buttons History	Stored Result	Displayed Result	Notes
0		0	""	Initial state
1	1	0	"1"	
2	12	0	"12"	
3	12 =	12	"Result = 12"	
4	12 = 2	12	"2"	
5	12 = 2 +	14	"14"	
6	12 = 2 + 1	14	"1"	
7	12 = 2 + 16	14	"16"	
8	12 = 2 + 16 -	-2	"-2"	
9	12 = 2 + 16 - 4	-2	"4"	
10	12 = 2 + 16 - 4 *	-8	"-8"	
11	12 = 2 + 16 - 4 * =	-8	"Result = -8"	
12	12 = 2 + 16 - 4 * = C	0	""	Final state
-----	<i>Variant for step 10:</i>			
11	12 = 2 + 16 - 4 * +	-8	"Missing operand"	Final state

¹ An expression in Polish or postfix form is an expression without parentheses having the operands in the same order with the infix form but with the operators put after their operands.

² Infix is the parenthesized form that we are most accustomed in mathematics books.

Note. The “=” button is used for both displaying results and separating the operands.

Advanced exercises

1. **(General case)** Consider expressions as the following:

$$12 + 8/2*(16-4)$$

The same expression written in Polish form is the following:

$$12\ 8\ 2\ /\ 16\ 4\ -\ *\ +$$

To separate the operands for our GUI the expression becomes:

$$12 = 8 = 2 / 16 = 4 - * +$$

In order to carry out the multiplication, the result of the operation $8 / 2$ should be kept in memory during the operation $16 - 4$. Moreover, for carrying out the last addition the value 12 should be kept in memory for the whole processing. How could we manage such expressions?

The solution is to use a *stack* or a *Last-In-First-Out (LIFO)* list of operands and proceed in the following way:

- a) Each time a new operand is entered, it is pushed in the stack.
 - b) Each time a binary operator is entered, the last two operands existing in the stack (the operands on the stack top) are used to carry out the operation and the operation result is pushed in the stack.
 - c) The computing ends successfully when the last element (probably an operator) is processed and the stack contains only a value, which is the expression result.
2. **(Overloaded operators)** Consider the cases where the operands may be negative integers, too. How could we manage such expressions?