Lab 2: JavaRPNCalculator

Overview

In this lab, you will implement an RPN (Reverse Polish Notation) calculator in Java using object inheritance and generic collections.

1 RPN Calculators and Post-fix Notation

Reverse Polish notation (RPN) calculators refers to a style of calculator that uses post-fix notation and a stack to maintain computation state.

To implement your calculator, you must first understand post-fix notation.

As a comparison, consider in-fix notation which we are already familiar with.

(9 + 3)

\*

10 / 2

The “fix” refers to the location of the operation with relation to the values it operates over. For example, the plus operator “+” is in-fixed between the numbers being computed, 9 and 3.

This style of notation is very natural for humans because we read the statement “9+3” as “nine plus three,” but in-fix notation is

complicated for computers to parse, and it also requires strict adherence to order of operations, lest the improper result is calculated.

Post-fix notation, in contrast, is harder for humans to read but much easier for computers to parse.

Again, the “fix” refers to the location of the operands, and post-fix notation places the operator following the values being operated over.

So: 1 + 2 is written 1 2 +, and we can rewrite the example above as:

9 3 + 10

\*

2 /

As you might have observed, the advantage to post-fix notation is that the order of operations is syntax directed; that is, there is no need for parenthesis to indicate which operations should be applied first.

Instead, the order of the input and the operations indicate directly which computation should occur first by reading from left to right. In the example above, first 9 3 + is performed, whose results is the first input to 10 \*,

whose result is the first input to 2 /, and so on.

An RPN calculator takes this process one step further by maintaining the states of the calculation in a stack, the last in first out data structure. The idea is that each value that occurs (i.e., a number) is

Pushed onto the stack and operators

Pop from the stack to retrieve their inputs,

Pushing the result of the operation back

onto the stack. As refernce, consider the computation above using an RPN calculator output below.

The “-\*-” indicates an empty stack, and you should consider this output as an example of how your calculator should function.

0: -

\*

-

RPN>10 3 9

2:10.0

1:3.0

0:9.0

RPN>+

1:10.0

0:12.0

RPN>

\*

0:120.0

RPN>2 /

0:60.0

RPN>

The head of the stack, index 0, always stores the result of the last operation or the last value pushed onto the stack.

Traversing this execution trace, you can see that first the values 10, 3 and 9 are pushed onto the 2 stack. The plus operation consumes 9 and 3 and pushes 12 onto the stack. Next the multiplication operation

consumes 10 and 12 and pushes the result 120 onto the stack, and, finally, 2 is pushed onto the stack followed by the division operation which results with 60 on the stack.

+

: plus

-

: minus

\*

: multiplication

/

: division

ˆ

: exponentiation

sq

: square

sqr

: square root,

lg

: natural logarithm, e.g., e lg returns 1

logx

: logarithm base x, e.g., 100 10 logx returns 2

sin

: sin function in radians

cos

: co-sin function in radians

tan

: tangent function in radians

clr

: clear the stack

help

: help function that provides the user info about all operations and constants

e

: the natural logarithm root

pi

: the ratio of the circumference of a circle to its diameter

5