CS 145: Preassignment 3

PostOps

1 Description

Your primary goal in this homework is to introduce yourself to arrays. You will do so by writing a calculator to evaluate postfix expressions.

Likely you've never heard of postfix expressions before. That's because many of us use *infix* notation all through our education. With infix notation, operators appear between their operands. In postfix, they appear after. The infix expression `6 * 7` is `6 7 *` in postfix notation.

Postfix is interesting for two reasons. First, it doesn't require parentheses to alter the precedence. For example, infix `(1 + 3) * 2` is postfix `1 3 + 2 *`. Second, and as a direct consequence of the first, it is much easier to teach a machine to evaluate a postfix expression than an infix one.

2 Requirements

To receive full credit for this assignment, you must write a class `PostOps` in package `pre3` that meets the following specification:

- Has a method `contains` that accepts a `String` array parameter for a haystack and a `String` parameter for a needle, in that order. It searches for the needle in the haystack. That is, it checks to see if the needle is identical to at least one of the elements of the array. If found, it returns `true`. Otherwise, it returns `false`. Test this method in isolation before moving on.

- Has a method `evaluate` that accepts a `String` parameter for a postfix expression to evaluate. It returns the answer as a `double`. Support the following operators/functions: `+`, `-`, `*`, `/`, `cos`, `sin`, and `tan`. (Assume the parameters to the trigonometric functions are expressed in radians, just like the methods in the `Math` class.) The algorithm for evaluating a postfix expression is elegant compared to its infix counterpart. We express it here in pseudocode:

```
make a list of tokens comprising expression
make a list of operands, initially empty
for each token in expression
  if token is a number
    append it to operand list
  else (token is an operator/function)
    if token is +
      b = remove operand from end of list
```

```
      a = remove operand from end of list
      append a + b to operand list
    else if ...
the answer is sole remaining operand in list
```

Let's work through an example of evaluating `"3 1 5 - *"` with this algorithm.

1. We first create an array of white-space separated tokens contained in the expression. Writing this yourself is an informative exercise, but the `split` method in the `String` class is happy to do it for you.
2. We then make an array of `double`s. It has to be big enough to hold all the queued-up operands we may run into. We know there will never be more operands than there are tokens in the expression (in this case, 5), so we size our array accordingly. We mark the first spot as the next spot to fill:

operands					
	↑				

3. We see `"3"` as the first token. Is this a number? There are several ways to tell. Assuming we were passed a valid expression, one way is to use our `contains` method and pass it an array of supported operators and functions as the haystack and this token as the needle. Token `"1"` doesn't appear in the list, so we assume it must be a number. We convert it from a `String` into a `double` with the `parseDouble` method found in the `Double` class. We append the number to the operand list and update our end marker:

operands	3.0				
		↑			

4. The next token is `"1"`. This is much like the previous step:

operands	3.0	1.0			
			↑		

5. The next token is `"5"`. Again:

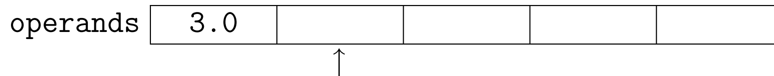
operands	3.0	1.0	5.0		
				↑	

6. The next token is `"-"`. This token is contained in our array of operators. We pop off the number to subtract:

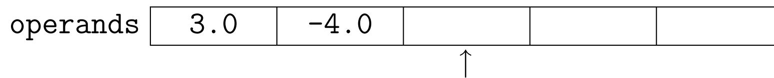
b	5.0				
operands	3.0	1.0			
			↑		

   And we pop off the number to subtract from:

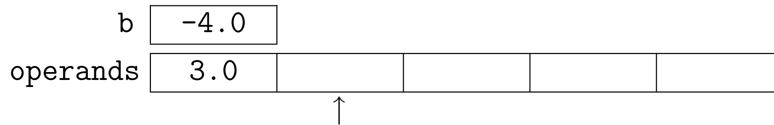
a	1.0



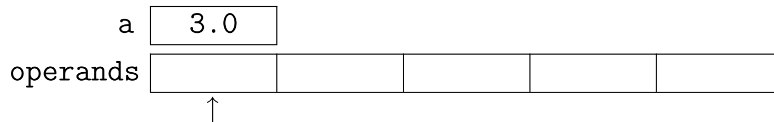
Now we append the difference, `a - b`:



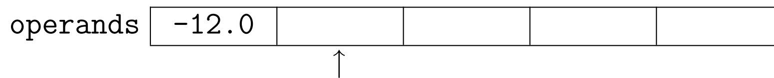
7. The next token is `"*"`. This token is contained in our array of operators. We pop off one operand:

   And we pop off the other:

   Now we append the product, `a * b`:

8. We have no tokens left. Our answer is the sole operand remaining in the list.

Test this method in isolation before moving on.

- Has a method `evaluate` that takes a `String` array of expressions as its only parameter. It returns a `double` array of the exact same size as the expressions array and in which element i is the result of evaluating expression i. Test this method in isolation before moving on.

3 Files

- SpecChecker: twodee.org/teaching/cs145/2012A/homework/speccheck_pre3.jar

4 Suggested Process and Submission

Please see preassignment 1 for the process on how to tackle coding assignments and submit them.