CS1027

LAB 5

Computer Science Fundamentals II

Learning Outcomes

- Implement missing methods in a stack made with an array data structure
- Consider and compare the implications of using different implementations of stacks
- Use stacks in real-world applications

Pre-Lab

- Create a new Java project called Lab5
- Download the files: StackADT.java, ArrayStack.java, EmptyCollectionException.java, TestStack.java, PostfixEvaluator.java, and Postfix.java
- Save these downloaded files into the Lab5 src folder

Exercise 1 – Completing the Array Stack class

1. Open ArrayStack.java and TestStack.java and examine the code in both classes.
2. Note that ArrayStack is not complete; most of its methods are left empty. Without adding any additional instance variables or methods or changing any of the provided code, complete the six empty methods given the following instructions:
   a. `push(T element)` – if the array is full, call `expandCapacity()` (which is provided). Add the given element to the top of the stack.
   b. `pop()` – if the stack is empty, throw an EmptyCollectionException. If not, remove and return the element at the top of the stack.
   c. `peek()` – if the stack is empty, throw an EmptyCollectionException. If not, retrieve the element at the top of the stack and return it without removing it.
   d. `isEmpty()` – return true if there are no elements, and false otherwise.
   e. `size()` – return the number of elements in the stack.
   f. `toString()` – if the stack is empty, return "The stack is empty." Otherwise return a single-line string starting with "Stack: " and then containing all the elements of the stack from the top to the bottom. There must be a comma and space between elements but the last element must end with a period instead. i.e. Stack: 1, 2, 3, 4, 5.
   Hint: how can you determine when the loop has reached the last element?
   g. NOTE: In this implementation, the default value of top is 1 rather than 0. This is different than the example shown in lecture. Do **not** change this default value, but rather think about how to adjust these methods to work with this value of top.

3. Run the TestStack file. This will test that the ArrayStack methods were properly implemented. If any of the test failed, fix the corresponding methods in ArrayStack until all the tests pass. You may add print lines in ArrayStack to help with the debugging. Do not add print lines in TestStack or alter this file in any way.

Exercise 2 – Postfix Expressions

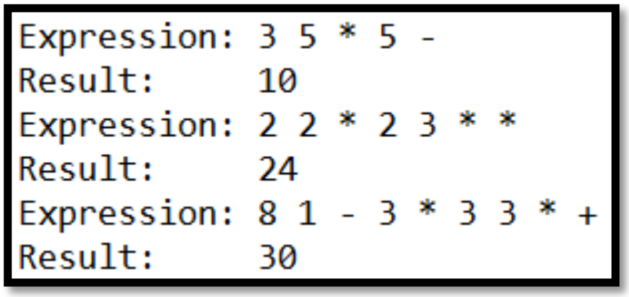
Postfix is a notation for expressions in which the operators are shown after the operands rather than in between them (as we are used to seeing with infix notation). For example, 2 3 + is the postfix equivalent to the infix expression 2 + 3. For another example, 1 4 + 3 7 * + is the postfix equivalent to the infix expression (1 + 4) + (3 * 7). We will use a stack to evaluate a postfix expression by pushing operands (numbers), popping twice when we see an operator, and pushing the result of that single operation onto the stack.

We will also use a stack to keep track of recently submitted expressions so that we can display the recently viewed expressions, similar to the History feature in a browser.

1. Open PostfixEvaluator.java and Postfix.java and examine the code in both classes. These classes are partially provided but some code will need to be filled in to complete them.
2. Complete PostfixEvaluator.java using the following steps and hints.
    a. Add the line in the constructor to properly initialize the stack object.
    b. Complete the code within the while-loop in the `evaluate()` method. Each "token" will be one value from the expression, either an integer or an operator character. This code must check which type it is and then perform the proper actions accordingly. If the token is an integer, simply push it on the stack. If the token is an operator, pop the top two items off the stack, evaluate that single operation on the two numbers, and then push the result on the stack.

String.

    c. Run this file to test that it works properly. The expected output is:



```
Expression: 3 5 * 5 -
Result:     10
Expression: 2 2 * 2 3 * *
Result:     24
Expression: 8 1 - 3 * 3 3 * +
Result:     30
```

3. Complete Postfix.java using the following steps and hints.
    a. Add the two lines in the constructor to properly initialize the expStack and PostfixEvaluator objects.

b. Fill in the two incomplete sections in the `run()` method.
    i. Most of the "e" (evaluate) section is finished, but it requires one additional line to keep track of the expression so that it can be retrieved when looking at the recently viewed expressions. Add it to the stack.
    ii. The "r" (recent) section is empty so you need to add a few lines here. If there are 3 or more expressions stored on the expression stack, call showRecent to look at the last 3 items only. If there are less than 3 expressions on the stack, call showRecent with that amount. For example, if there are 1 or 2 expressions, show those 1 or 2. If there or 4 or 5, we only want to see the last 3 expressions.

c. Now we have to complete the `showRecent()` method that we just referred to in the last step. Get the latest n expressions from the stack, where n is the numToShow parameter value. For each of these expressions, print out the expression followed by the result of the expression (for clean formatting, add a tab at the start of the line and add " = " (space equals space) between the expression and its result. Make sure the stack has these items added back on top in the correct order (not in reverse order!). Use a temporary local variable stack to help within this method.

d. Run this file to test that it works properly. Since it's user-input-driven, test it several times with various expressions and combinations of hitting "e" and "r" to ensure both parts work. Test the "r" portion when having < 3 expressions submitted and again when having 3 or more expressions submitted.

Submission

When you have completed the lab, navigate to the weekly module page on OWL and click the Lab link (where you found this document). Make sure you are in the page for the correct lab. Upload the files listed below and remember to hit Save and Submit. Check that your submission went through and look for an automatic OWL email to verify that it was submitted successfully.

Rules

- Please only submit the files specified below. Do not attach other files even if they were part of the lab.
- Do not ZIP or use any other form of compressed file for your files. Attach them individually.
- Submit the lab on time. Late submissions will receive a penalty.
- Forgetting to hit "Submit" is not a valid excuse for submitting late.
- Submitting the files in an incorrect submission page will receive a penalty.

- You may re-submit code if your previous submission was not complete or correct, however, re-submissions after the regular lab deadline will receive a penalty.

Files to submit

- ArrayStack.java
- PostfixEvaluator.java
- Postfix.java