Homework M10: Stacks

Due Apr 18 at 11:59pm **Points** 100 **Questions** 23
Available until May 30 at 11:59pm **Time Limit** None

Instructions

Review the **Homework FAQ** page for details about submitting homework. In this homework, you will:

- trace the use of stacks
- use stacks to solve problems
- write code to use stacks as a client
- write code to go inside the stack classes (from the implementation perspective)

Homework Files

Below is the driver/tester program. I strongly recommend using this to test your code before submitting. You can ignore the *test methods* at the end of the file.

Important Note: Use the StackInterface file provided in the zip below. The extra credit method has been added to the interface so it will compile with the tester.

Important Note: There are two driver files to test this week's homework: one for the folder question and one for all the other questions. Two notes for the driver provided to test the folder code:

- The driver for the folder code doesn't have any automated tests. You have to compare the output manually.
- Be sure to run the folder tester code multiple times because the test cases are being randomly generated. If you run it just once and it passes, the code still might not be correct.

HomeworkM10Files.zip (https://ccsf.instructure.com/courses/47904/files/7254658/download?download_frd=1)

This quiz was locked May 30 at 11:59pm.

Attempt History

Attempt	Time	Score

	Attempt	Time	Score
LATEST	Attempt 1	8,860 minutes	83 out of 100

Score for this quiz: **83** out of 100
Submitted Apr 18 at 10:04pm
This attempt took 8,860 minutes.

Coding Questions: LinkedStack and ArrayStack

Write a method called priorityPush that will be added to the LinkedStack class. The goal of the method is to either add a new element to the stack or, if the element already exists in the stack, move it up to the top ("prioritize" it).

The method works as follows:

- when invoked with an element **not** currently in the stack:
  - the element is pushed as normal
  - stack size increases by one
  - false is returned to indicate that the element was *not* prioritized, but just pushed as normal
- when invoked with an element already in the stack:
  - the element is removed from the current position and placed on top of the stack
  - a duplicate element is **not** added; in other words, the stack size stays the same
  - true is returned to indicate that the element was prioritized
  - if more than one element exists in the stack, the **first** occurrence (closest to the top) is prioritized

There are test cases in the driver program.

The method header is:

```
public boolean priorityPush(T element)
```

Question 1

20 / 20 pts

Paste the complete priorityPush method here.

Your Answer:

```
public boolean priorityPush(T element) {
    Node currentNode = null;
    currentNode = topNode;
    boolean priorityPushBoolean = false;

    // if top of stack is the element
    if (topNode != null && topNode.data.equals(element)) {
        return true;
    }

    // if element exists inside stack
    while (currentNode != null) {

        if (currentNode.next != null && currentNode.next.data.equals(elemen
t)) {

            this.push(currentNode.next.data);
            priorityPushBoolean = true;
            currentNode.next = currentNode.next.next;

            return true;
        }

        currentNode = currentNode.next;

    }

    this.push(element);

    return priorityPushBoolean;
}
```

Optional Extra Credit: 10 points

A stack lets you peek at the top entry without removing it. For some applications, you might also want to peek at the entry beneath the top entry without removing it.

Write **two** peekNext methods, one for **LinkedStack** and one for **ArrayStack**.

- If the stack has two entries or more, peekNext returns the second entry from the top without altering the stack.
- If the stack has fewer than two entries, peekNext should either return null or throw an exception.
  - For full credit, make sure your code is consistent with the existing peek method.

Here is the method header for both classes:

```
public T peekNext()
```

Question 2

10 / 0 pts

Paste **both** peekNext methods here. Submit only the two methods (**not** the full classes).

Your Answer:

```
public T peekNext() {
    Node currentNode;
    currentNode = topNode;

    if (topNode != null && currentNode.next != null) {
        return currentNode.next.data;
    } else {
        throw new EmptyStackException();
    }
}



public T peekNext() {
    checkIntegrity();

    if (this.isEmpty()) {
        throw new EmptyStackException();
    } else if ((topIndex + 1) < 2) {
        throw new EmptyStackException();
    } else {
```

```
        return stack[topIndex - 1];
    }
}
```

Coding Questions: Folders

Review the Folder and File classes in the homework files. These are similar to those we used in the recursion module.

Review the printFolderContentsRecursive method in the driver. This method prints a tree-like hierarchy of all of the files and folders in a top level folder.

One use of stacks is to replace recursion. Write a method that accomplishes the same task as the recursive printFolderContentsRecursive method, but using stacks instead.

I've started the method for you, you will complete the code.

Think about how you can use a stack. The approach of recursion is to do a small piece of the problem now and then combine that with the "rest of the solution" later on. You get that "rest of the solution" with your recursive call. For stacks, you can do the same thing. The difference is that instead of using a recursive call, you can use the stack to store the "solve later" part.

Building the folder hierarchy:

- I have helper methods in the driver that build the folder hierarchy.
- When you create your "top folder" (called "Folder0"), you can specify the depth of the hierarchy that you want.
  - I recommend starting your testing with a small number (1 or 2). Then increase the number once your code is working.
- When the hierarchy is built, the methods randomly decide how many

subfolders and files will be included at each level of the hierarchy.
  - This will allow you to make sure your code is robust.
  - **Because randomization is used, be sure to test multiple times for each depth!**
- I've included a picture of an example hierarchy.
- If you have questions about the hierarchy that is being created, please post to the discussion board!

For full credit:

- Use **only** the stack methods push, pop, peek, and isEmpty.
- Do **not** use other data structures.
  - You **can** create local variables of type Stack.
  - You **can** use the returned List<Folder> and List<File> objects.
  - But do not create any **new** lists!

Formatting (5/25 points):

- Match the indentation formatting of the recursive method.
- You can invoke the recursive method with and without formatting to compare outputs to the outputs from your stack.
- My recommendation is to get the **contents** of the print correct first and only once that is working, then work on the indentation.
- Again, use only local variables and no data structures other than a stack!

Unanswered

Question 3

0 / 25 pts

Paste your complete printFolderContentsWithStack method here.

Your Answer:

Short Answer Questions

Convert the following infix expressions into postfix notation.

- Only include the letters and operands in your answer with no spaces.
  - Do **not** include any other symbols or labels in your answer.
  - Do **not** include any spaces in your answer.
- For example: ab*cd+

Question 4

5 / 5 pts

a * b / c * d

Correct!

ab*c/d*

orrect Answers

ab*c/d*

a b * c / d *

Question 5

5 / 5 pts

a / b * c + d

Correct!

ab/c*d+

Correct Answers

a b / c * d +

ab/c*d+

Question 6

5 / 5 pts

a / (b - c) * d

Correct!

abc-/d*

Correct Answers

a b c - / d *

abc-/d*

Evaluate the following postfix expressions using these values:

a = 1.0

b = 2.0

c = 3.0

d = 4.0

e = 5.0

Question 7

5 / 5 pts

a b - c * d +

Correct!

1

orrect Answers

1 (with margin: 0)

Question 8

5 / 5 pts

a b * a c + * d e * +

Correct!

28

orrect Answers

28 (with margin: 0)

Question 9

5 / 5 pts

e c + d / b +

Correct!

4

orrect Answers

4 (with margin: 0)

Multiple Choice Questions

Question 10

0 / 1 pts

The push and pop operations of a stack are both linear O(n).

ou Answered

- (●) True

rrect Answer

- ( ) False

Question 11

1 / 1 pts

In the ArrayStack class, the top of the stack is **always** located in index 0.

- ( ) True

Correct!

- (●) False

Question 12

0 / 1 pts

In the LinkedStack class, for an empty stack, this is true:

```
topNode.data == null
```

ou Answered

- (•) True

rrect Answer

- ( ) False

Question 13

2 / 2 pts

What is the return type of the push method in the Stack class from the Java standard library (java.util.Stack)?

- ( ) int
- ( ) void
- ( ) boolean

Correct!

- (•) none of these is correct

Question 14

2 / 2 pts

What is the contents of the stack after the following code is executed? Answer choices are listed **BOTTOM ... TOP**

```
stack.push(92);
stack.push(56);
System.out.println(stack.peek());
```

- bottom 92 top
- bottom 56 top
- none of these is correct

Correct!

- (selected) bottom 92 56 top
- bottom 56 92 top

Question 15

2 / 2 pts

What is the contents of the stack after the following code is executed? Answer choices are listed **BOTTOM ... TOP**

```
stack.push(74);
stack.push(13);
int n = stack.pop();
stack.push(n);
```

- none of these is correct
- bottom 13 74 top
- bottom 13 top
- bottom 74 top

Correct!

- (selected) bottom 74 13 top

Question 16

2 / 2 pts

What is the contents of the stack after the following code is executed?
Answer choices are listed **BOTTOM ... TOP**

```
stack.push(58);
stack.push(stack.peek());
```

- none of these is correct
- bottom 58 top
- the stack will be empty
- bottom 58 58 58 top

Correct!

- (selected) bottom 58 58 top

Question 17

2 / 2 pts

What is the contents of the stack after the following code is executed?
Answer choices are listed **BOTTOM ... TOP**

```
stack.push(13);
stack.push(22);
stack.push(47);
```

Correct!

- (selected) bottom 13 22 47 top
- bottom 47 22 13 top
- bottom 13 47 22 top
- none of these is correct

Question 18

2 / 2 pts

What is the result of the following code?

```
stack.push(74);
stack.pop();
System.out.println(stack.pop());
```

- [ ] [] is printed
- [ ] null is printed
- [x] an exception or error is thrown
- [ ] none of these is correct
- [ ] false is printed

Correct!

Question 19

2 / 2 pts

How many elements are in the stack after the following code executes? (The stack is initially empty.)

```
stack.push(23);
stack.push(9);
stack.pop();
stack.push(14);
stack.push(3);
stack.push(stack.pop());
stack.push(stack.peek());
stack.push(17);
stack.push(8);
stack.pop();
stack.peek();
```

Correct!

5

orrect Answers

5 (with margin: 0)

For the next set of questions, use the algorithm from the lecture notes and videos (and in Section 5.8 of the textbook) to evaluate whether the parentheses are balanced. Select the situation that will cause the algorithm to end.

Note: parentheses refer to ( ) { } and [ ].

Question 20

2 / 2 pts

{ a * [b - c] / [d + e]

Correct!

- (•) open parenthesis left on the stack
- ( ) closed parenthesis left on the stack
- ( ) none of these is correct
- ( ) empty stack and no more tokens
- ( ) empty stack with no match for a close parenthesis
- ( ) mismatch found when popping from the stack

Question 21

2 / 2 pts

(a { b + c } + [d * e] - f) / g]

- none of these is correct
- **Correct!** empty stack with no match for a close parenthesis
- open parenthesis left on the stack
- empty stack and no more tokens
- mismatch found when popping from the stack
- closed parenthesis left on the stack

Question 22

2 / 2 pts

a [b [c - d] * e + { f / g }) + h

- empty stack with no match for a close parenthesis
- open parenthesis left on the stack
- empty stack and no more tokens
- none of these is correct
- closed parenthesis left on the stack
- **Correct!** mismatch found when popping from the stack

Question 23

2 / 2 pts

a * (b / { c + (d - e) } * f)

Correct!

- (•) empty stack and no more tokens
- ( ) empty stack with no match for a close parenthesis
- ( ) mismatch found when popping from the stack
- ( ) closed parenthesis left on the stack
- ( ) open parenthesis left on the stack
- ( ) none of these is correct

Quiz Score: **83** out of 100