CIT 5940 - Module 3 Programming Assignment
Stacks & Queues

Contents

Assignment Overview

In this assignment, you will use `Stacks` and `Queues` to check documents for correct nesting. After briefly explaining how this can be used as part of the syntax validation in multiple settings including arithmetic expressions and computer programming languages, you will build a function that applies this knowledge to check strings of nestable characters for valid delimiter nesting. After you have used a Stack to determine if a Queue had valid nesting patterns, you will use a similar approach to implement a Reverse Polish notation (RPN) Calculator.

Learning Objectives

- Become familiar with the methods in the `java.util.Stack` class and `java.util.Queue` interface
- Work with abstract data types including `Queue`, `Stack`, and `Nestable` (unique to this assignment) using only the interface of their implementation.
- Apply what you have learned about how `Stacks` and `Queues` work

Advice

We highly encourage you to utilize the course discussion forum to discuss the project with classmates and TAs. Before asking the question, please search in the forum to see if there are any similar questions asked before. Please check *all* suggestions listed there.

For this and all other assignments in this course, we recommend that you work on your local computer using an integrated development environment (IDE) such as Eclipse, IntelliJ, Visual Studio Code, or whatever IDE you used for any prior Java courses that you are familiar with.
Although you will need to upload your solution to Codio for grading (see the Submission Section below) and could develop your solution on that platform, we recommend using industry standard tools such as Eclipse or IntelliJ so that you become more used to them and can take advantage of their features.

If you have trouble setting up your IDE or cannot get the starter code to compile, please post a note in the discussion forum and a member of the instruction staff will try to help you get started.

We expect that you will create your own JUnit tests to validate your solution before submission.

1 Setup

1. Begin by downloading the `CIT594-stacks-queues.zip` archive from Canvas, then extract the contents. This contains the unimplemented methods for the code that you will write in this assignment.

   You MUST NOT modify the following files:

   (a) `Nestable.java`, which is an **abstract** class representing the concept of a "nestable" entity. You may find `enum NestEffect`, and the `getEffect()` and `matches(Nestable other)` methods useful in your implementation.

   (b) `NestableCharacter.java`, whose class `NestableCharacter` **extends** `Nestable` to represent single characters as nestable entities. The `matches` method, required by `Nestable`, is defined such that `(` matches `)`, `[` matches `]`, and `{` matches `}`, all bi-directionally.

   (c) `NestingReport.java`, whose class `NestingReport` is the compound datatype that you will need to return from your implementation of `NestingChecker.checkNesting`.

   (d) `VanNest.java`, whose class `VanNest` **extends** `Nestable`.

   You can work in the remaining files:

   (a) `RpnCalculator.java`, which contains the unimplemented method for a Reverse Polish notation (RPN) Calculator.

   (b) `NestingChecker.java`, which contains the unimplemented method `checkNesting` that you will write in this assignment. Your submission for NestingChecker MUST be contained entirely in this file.

   (c) `Playground.java`, which you can use to test your `NestingChecker` implementation interactively.

2. Compile this code before making any changes. This will ensure your local environment is configured correctly.

3. Review the current implementation. You are expected to read each of the files to become familiar with its purpose and how to use it. You do not need to understand the implementation details of `NestableCharacter` or `VanNest`. You may treat them as black boxes which use the Nestable interface.

2 Requirements

2.1 Structure and Compiling

1. You MUST use a JDK version of 17 or higher.
2. You MAY use a JDK version higher than 17, but you MUST set the Java language level to 17 for compatibility with Codio.
3. You MUST NOT change any of method signatures for any of the provided methods. This includes the parameter lists, names, and return value types, etc.
4. You MUST leave all classes in the default package.
5. You MUST NOT create additional `.java` files for your solution.
6. If you need additional classes, you MUST define them in `NestingChecker.java`.
7. You SHOULD create JUnit tests to help validate your code. These tests SHOULD NOT be a part of your solution. We do expect you to test your code thoroughly with JUnit tests before submission.
8. You MUST follow our directives in the starter code comments. This means if we ask you to not change a variable's value, you MUST NOT change that variable's value.
9. You MAY create additional helper functions, as long as they meet the other requirements of the assignment (e.g. doesn't crash with invalid inputs, etc.).
10. You MUST fill out the required Academic Integrity signature in the comment block at the top of your submission files.

2.2 Functionality Specifications

2.2.1 General Requirements

1. Your method MUST NOT crash if a user enters an invalid input. Remember you do not have control over what a user inputs as arguments. For example, all methods should handle `null` inputs gracefully.

2.2.2 Algorithm Specification(NestingChecker)

You will be implementing an algorithm which returns a `NestingReport` detailing how the input's structure is, or is not, correctly nested. You MUST use a `Stack` to maintain the **opening** elements which have not yet been closed by a complementary **closing** element. You MUST use a `Queue` to hold the **open** and **close** elements that have not been processed yet.

Consider as an example the following Python expression, which is a queue of nestable elements:

```
[0, (1, 2, {3, (4, 5, 6), (7, 8}), 9), 10, 11, 12 ]
```

This expression contains 3 types of **open** and **close** elements:

Open	**Close**
(	)
[	]
{	}

And follows two rules:

1. a **closing** element must close the *current context*
2. at the end of input, all contexts must be closed

Note that an **open** element starts a context and the complementary **close** element ends that context. Contexts can be *nested* inside each other, but we can't close an outer context until the current context is closed.

Here are some more examples:

```
# valid; stack should be empty
[0, (1, 2, {3, (4, 5, 6), (7, 8)}, 9), 10, 11, 12 ]

# invalid close on }; stack should be '[({('
[0, (1, 2, {3, (4, 5, 6), (7, 8}), 9), 10, 11, 12 ]
                              ^ invalid close

# not terminated; stack should be '['
[0, (1, 2, {3, (4, 5, 6), (7, 8)}, 9), 10, 11, 12
                                                 ^ missing ]

# invalid close on }; stack should be empty
[0, (1, 2, {3, (4, 5, 6), (7, 8)}, 9), 10, 11, 12 ] }
                                                   ^ extra }

# valid; we are only considering nesting and not other aspects of syntax
(+ 1 1)
```

The `NestingReport` generated by the algorithm consists of:

1. the status of the input, (e.g. if it is valid or invalid)
2. the first element, if any, which caused the input not to be valid (i.e. an incorrect close), and
3. the stack at the moment of generating the `NestingReport`

The algorithm MUST end when any of the following is true:

1. the `Queue` and `Stack` are empty (valid nesting),
2. an element in the `Queue` is `null` (null item),
3. the `Queue` has an invalid closing element (invalid close), or
4. the `Queue` is empty, but `Stack` still has elements that were never closed (not terminated)

2.2.3 `NestingReport` Specification

1. If the input queue is `null`, you MUST return a `NestingReport` with
   (a) `status` set to `NULL_INPUT`,
   (b) `badItem` set to `null`, and
   (c) `stackState` set to the empty Stack.
2. When an element e in the Queue is `null`, you MUST return a `NestingReport` with
   (a) `status` set to `NULL_ITEM`,
   (b) `badItem` set to e (null), and
   (c) `stackState` set to the current contents of the `Stack`.
   You MUST NOT include e in the `stackState`.
3. When an element e in the `Queue` is an invalid **closing** element, you MUST return a `NestingReport` with:
   (a) `status` set to `INVALID_CLOSE`,
   (b) `badItem` set to e, and
   (c) `stackState` set to the current contents of the `Stack`
   You MUST NOT include e in the `stackState`.

4. When the `Queue` represents an invalid nesting because the `Queue` is empty but there are still elements remaining in the Stack that were never closed, you MUST return a `NestingReport` with
   (a) `status` set to `NOT_TERMINATED`,
   (b) `badItem` set to `null`, and
   (c) `stackState` set to the current contents of the `Stack`.

5. If the input represents a valid nesting, then return a `NestingReport` with
   (a) `status` set to `VALID`,
   (b) `badItem` set to `null`, and
   (c) `stackState` set to the current contents of the `Stack` (here, the empty `Stack`).

2.2.4 Algorithm Specification(RPN Calculator)

What is a RPN Calculator? See more information here:
`https://en.wikipedia.org/wiki/Reverse_Polish_notation`
Postfix notation entirely avoids nesting by using a stack to manage operands and results. Operands are added to the stack in the order given, and operators pop however many values they require for input, compute the result, and push the result back onto the stack. For example:

```
6 => 6
1 1 + => 2
1 -5 + => -4
2 1 3 + * => 8
2 1 + 3 * => 9
4 13 5 / + => 6
3 2 * 11 - => -5
2 5 * 4 + 3 2 * 1 + / => 2
```

2.2.5 Implement an RPN Calculator

1. In the starter file, implement the evaluateExpression method to return the answer (as an Integer) to the inputted expression (as a List of Strings) as an RPN Calculator would.

2. For simplicity, complete ALL calculations using Integers-only; this includes intermediary calculations. Do not use long, short, byte, float or double to store the strings. This means that $3/2 = 1$, not $3/2 = 1.5$.

3. If the input is null, the method should return null.

4. If an element in the input List is null, the RPN Calculator should return null.

5. If there is a divide-by-zero in any calculation, the RPN Calculator should return null.

6. You may assume that ALL values, including intermediary calculated values, are within the Integer.MIN_VALUE to Integer.MAX_VALUE range: $-2^{31} \leq x < 2^{31}$. This means there will be no overloading issues.

7. You may assume that all the String elements in the input List are either:

   - parsable into Integers
   - one of the four mathematical operators: "+", "−", "∗", or "/"
   - null

8. You may assume that each input List into the RPN Calculator is a valid sequence of Strings. For example, inputs such as ["5", "∗"] will not be tested/graded.

2.3 Writing Test Cases

Upon completion of each functional requirement, always conduct thorough testing of the corresponding functions. Consult the guidelines on composing JUnit tests and creating test cases in Module 1. You MUST develop JUnit tests to cover all corner cases you can think of for each of your implemented function in order to get full credit. This testing component carries a weight of approximately 5% towards the total assignment score.

3 Submission

3.1 Pre-submission Check

Before you submit, please double check that you followed all the instructions, especially the items in the Structure and Compiling section above.

3.2 Codio Submission

1. When you are ready to submit the assignment, go to the Module 3 Programming Assignment Submission item and click the button to go to the Codio platform.

2. Once you are logged into Codio, read the submission instructions in the `README` file. This should be the first thing that appears in the window.

3. Upload your solution and any JUnit test files to the "submit" folder.

4. In the menu bar, select Run JUnit Tests. This will run any unit test files that you have uploaded with your submission. Note that there are no pre-submission checks provided. However you can use this feature as a basic validation check to be sure your code compiles.

5. When you're ready to submit, go to Education and select Mark As Completed. Confirm at the prompt.

6. You will see quite a bit of output, even if all the tests pass. At the bottom of the output, starting at `YOUR AUTOGRADING RESULTS BELOW` you will see the number of successful and failed test cases.
7. If you want to update your code and resubmit, for example if you did not pass all of the tests, un-mark the assignment as complete. You will then be able to edit your code, run your JUnit tests, and resubmit.

4 Grading

4.1 Grading Overview

1. There is no peer review for this assignment. You will not be marked on code style. However, we do expect you to use best practices (comments, indentation, etc), especially if you want to review with a TA.
2. There are no hidden tests. You will be able to see the name of each test, if it passed or failed, and the associated point value.
3. Failed tests will have brief feedback about the specifics of the failure.
4. We will not provide the exact test cases.
5. You have unlimited submissions, until the due date of the assignment as noted in the Syllabus (or your approved extension).
6. The score at the due date is final. Scores will sync to Canvas relatively immediately.

4.2 Rubric Items

This assignment will be marked on 5 rubric items.

- **`CharacterNesting`** : 5 points for correctly handling general `CharacterNesting` checks
- **`NestingChecker`** : 7 points for correctly validating the nesting of unspecified input files
- **HTML Test** : 7 points for correctly validating the nesting of various html files
- **`VanNest`** : 4 points for correctly validating the nesting of various `VanNest` files
- **`RpnCalculatorTest`** : 19 points for 19 tests. Each test will test a different expression to see that the correct answer is returned

Total : 42 points (+ 2 Points for JUnit test cases)

5 Additional Resources

`java.util.Stack` class:
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Stack.html
`java.util.Queue` interface:
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Queue.html
Java tutorial on interfaces and inheritance:
https://docs.oracle.com/javase/tutorial/java/IandI/index.html
RPN Calculators:
`https://en.wikipedia.org/wiki/Reverse_Polish_notation`
Integer Java docs:
`https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Integer.html`

Good Luck!