CS 455 Lab 10: Linear Containers

Fall 2021 [Bono]

Goals and Background

In this lab you'll get some practice working with some Java linear contaner classes (besides ArrayList). The first problem uses the LinkedList class, in particular, using an iterator on a LinkedList. The other problems use the Stack class.

Reading and Reference Material

- Horstmann, Sect. 15.2, Using Linked Lists
- Horstmann, Sect. 15.5.1 and 15.6 on Java Stacks
- Horstmann, Section 15.6.1 Balancing parentheses example

Exercise 1 (1 checkoff point)

Write a static method `hasOnePeak` that returns `true` if its `LinkedList` argument has a single peak, and `false` otherwise. This means that the sequence of values consists wholly of an increasing part followed by a decreasing part. You may assume no two consecutive values in the array are the same (i.e., no flat areas). Your solution must use a constant amount of space (e.g., no additional arrays, ArrayLists, or LinkedLists) and run in linear time.

list	**hasOnePeak(list)**
2 4 7	false
2 7 4	true
7 4 2	false
2 4	false
2	false
2 4 7 9 3 2	true
2 4 7 9 7 3 7 1	false
7 5 4	false
2 4 2 3	false
5 4 3	false

We have provided a test driver for your method that tests it on several hard-coded cases, and compares the actual results with the expected results. The test driver source file, `PeakTester.java`, contains the stub for the `hasOnePeak` method. For check-off show your TA your code and a run of the code with the provided test program.

Exercise 2 (1 checkoff point)

Add thorough test cases to the file RemoveEvens.java to test the method `removeEvens` (which is a stub right now). Use the already-written methods `createStack` and `doOneTest`. We provided one test example for you. To get this check-off point, you'll need to save your output from running this version of the program (i.e., that uses the stub version of removeEvens) in a file called `removeTests.out`. Also tell the TA your expected output for each of the tests you provided (i.e., expected for a working version of the removeEvens method). Note: a more detailed explanation of what the method does appears below in Ex 3. For DEN students: you can cut and paste your results of running this version into your README file, and annotate it with a list of expected results for each input.

Exercise 3 (1 checkoff point)

Please read the whole paragraph before starting. Implement a static method to remove all the even values in a stack of integers. After the call, all the odd values will still be in the stack in the same order as they were when you started. You may only use operations push, pop, peek, and empty. You may not use any other kinds of data structures in this method (e.g., no arrays or ArrayLists), but you may use more than one stack.

For the test that we provided in the starter file,

```
[1, 2, 3, 2, 4, 5] <-- top
```

the expected output is

```
[1, 3, 5] <-- top
```

For checkoff, show the TA your code running on the test cases you wrote in Ex 2 and your code to solve the problem.

Exercise 4 (1 checkoff point)

[*Adapted from E15.20 from Big Java: Early Objects, 7ed*] Please read over the whole exercise before beginning. In this exercise you are going to implement a static method `matchingTags` that takes a `Scanner` and returns `true` iff the text from the scanner source is a sequence of properly nested HTML tags, possibly with other text. For the opening tag, such as , there must be a closing tag . A tag such as may have other tags inside. For example the following html tags are properly matched and nested:

```
<p> some text <ul> <li> some more <a> wow!
</a> </li> </ul> <a> </a> </p> some more text
```

The inner tags must be closed before the outer ones. For simplicity, assume the tags are separated by whitespace, as is any other text in the file. You must use a stack to help you match tags.

To make this problem easier, we wrote some helper methods for you: `isOpening`, `isClosing`, and `getTagName`.

Before you start your implementation, do Part 1 to make sure you understand the problem, and expected results.

Part 1: add expected results. The test program `Html.java` contains several test cases for `matchingTags`, but right now the expected results provided are all `false`, whether the string does or does not contains matching tags. Look at the helper method `testOneString` to see what it does with its parameters. Run the starter version of this code.

Change the starter version so all the calls to `testOneString` have the correct *expected* results: i.e., based on whether the string provided has matching tags or not you'll put in a `true` or a `false`. (Some of the tests will still fail, of course, because `matchingTags` is a stub method.)

Part2: implement method. Implement `matchingTags` as described above. When you have a correct implementation (run on correct expected results) none of the test cases should print `FAILED`.

For checkoff, show the TA your expected results, demo your completed `matchingTags` method running using the test program provided, and show them your code to solve the problem.

Checkoff for DEN students

Make sure you put your name and NetID in the source code files and README.

When you click the `Submit` button, it will be looking for and compiling (for source code) the files `README`, `PeakTester.java`, `removeTests.out`, `RemoveEvens.java` and `Html.java`.
