PA02: Stacks, User Input/Output

Due: By the start of class on Tuesday April 7. At 10:25AM or 1:15PM.

Project partners. For this project you will work in self-assigned pairs, preferably different members from the previous projects or HWs in course. You can pair up with someone from Section 1 or 2.

Working in pairs. Here are some guidelines:
Arrange a suitable way to collaborate online, WebEx, HangOut, etc. You should always try to code *together*, i.e., both people at the same workstation at the same time. Alternate who does the typing so that each person gets to code for half the time. *Do not let the same person be the typist for all the code.* Set up a schedule right away, of meeting times to work on the project.

Comments and style. As usual, Javadoc-style comments are required throughout your code for this project and all future projects. As an example, see pages 50-51 in the textbook and the supplied ArrayStack and SinglyLinkedStack class files. For a quick javadoc style guide see: http://en.wikipedia.org/wiki/Javadoc

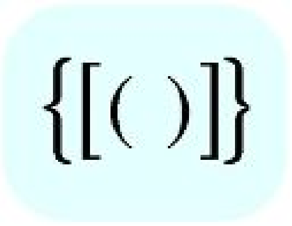
(10 pts) of your grade will be allotted to coding style, documentation/comments, and correctly following submission instructions.

Submission.

1. **Moodle**: Put all your project files in a single folder named username_pa2, for example, for me it would be wtarimo_pa2. Include all the .java files for all your project classes. I suggest that you create this folder when you start working on the project. Compress the folder into a zipped file (**.zip** - not rar or other formats) and then upload it to the PA 2 link on Moodle. See the syllabus file on Moodle for late project policies.
2. Since you're working in pairs, you can choose to have one team member make a submission on Moodle. Make sure to include each of your names in all the submission files.

Part A (90 pts). Matching Brackets.

For this project, you're going to complete a program that performs the bracket matching task we discussed in class several times. You will also learn how to create an interactive program that works from the terminal or command line. Your program will prompt the user for an input string, and then output whether the string has properly-matching brackets or not.



Roughly speaking, brackets are properly matched in a string of text if:

1. every opening bracket has **its own corresponding** closing bracket,
2. every closing bracket corresponds to an opening bracket, and
3. no opening bracket reaches its corresponding closing bracket until all the opening brackets coming after it have already been closed.

The formal definition of whether a string's brackets are matched can be derived from the pseudocode for the algorithm to solve the problem, which we studied in class and can also be found on page 235-236 of the textbook. For examples on correctly matched and incorrectly matched strings of brackets, see page 235 of the textbook. (These might be good test inputs for your program once you think it's working.)

As we discussed in class, a simple way to solve this problem is to use a stack. The class files for the **generic** array-based (ArrayStack) and singly-linked-list-based (SinglyLinkedStack) stacks are provided for your use or reference, you may choose whether to use the array-based or linked-list-based stack. The generic SNode and SinglyLinkedList classes are also provided, as they are needed for the SinglyLinkedStack class. You can test that your stack operations are working properly from a main method (used just for testing) within your stack class.

To solve this Bracket Matching problem, you will also implement the following classes:

1. The `BracketMatcher` class.
   - Fields:
     - hard-coded array of characters (of type `char`) holding the opening brackets (these should include: '(', '[', '{', '<')
     - hard-coded ***parallel*** array of characters (of type `char`) holding the **corresponding** closing brackets. (You might need to review about the Java type `char`, say, here: http://www.ibiblio.org/java/course/week2/24.html)

   - Methods:
     - `boolean isOpeningBracket(char c)`
       returns `true` iff the character `c` is an opening bracket
     - `boolean isClosingBracket(char c)`
       returns `true` iff the character `c` is a closing bracket
     - `boolean corresponds(char open, char close)`
       returns `true` iff the character `open` is a bracket that corresponds to the closing bracket character `close`
       note that we set up **parallel arrays** of brackets to help us with this task
     - `boolean checkBrackets(String s)`
       returns `true` iff the string `s` has brackets that are all matched up properly
       calls the other three methods to achieve this
   - You'll probably also need to familiarize yourself a little more with Java's String class so you know how to do a little string manipulation (like indexing into a String to access the characters that comprise it). Here (http://download.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) is the official Java documentation for the String class. Seems to me like the `charAt(i)` method should be helpful.

2. The `BracketMatchApp` class. (Only contains a main method, no fields or other methods.)
   - the `main` method: /* drives the user input/output of the program */. The structure of this method can be something like:

- Create an instance of the BracketMatcher class
- prompt user for input string using the Scanner class described below
- Use the instance of the BracketMatcher class to check if the string has properly-matched brackets
- Print an appropriate output, based on whether the string has its brackets matched properly

The simplest way to do user input from the console (terminal or command line/prompt) is probably to use Java's built-in Scanner class. Here are some helpful links on how to use scanner objects: http://www.cs.williams.edu/~jeannie/cs136/scanner.pdf (note on this document: apparently the `nextString()` method doesn't work, so you have to use `next()`) and http://www.java-made-easy.com/java-scanner.html.

Part B (up to 5 pts): This is an optional bonus credit component.
Have your `checkBrackets(String s)` method not only output whether or not the input string has matching brackets, but if the brackets don't match and the scenario allows, have it print the character number where it discovered the error (starting at 0 from the left) and the incorrect character it found there. For example, for the input string: `a{b(c]d}e` the program should also print (in addition to returning true/false): `Error: ] at position 5`