Lab 5

Application for the Stack ADT

I. Create a new directory for this lab (call it Lab5). Copy AStack class from Lab4 into Lab5.

II. Design and implement a Java class StringChecker with the following 2 static methods:

- isBalanced: This method has one String type parameter and returns a boolean value – true if in the parameter string all brackets, braces, and parenthesis are balanced, and false otherwise. We say that brackets, braces, and parenthesis are balanced if every such right symbol has its corresponding left counterpart (and vice versa), and they are correctly arranged – for example {{[]}} is legal while {{[ ]]} is not. Note that the parameter string may have symbols other than brackets, braces, and parenthesis but those don’t play any part in the decision making.

The following simple algorithm requires a stack – use your own AStack<T> class created in Lab4. You will need to create and use a stack of Characters for your algorithm.

Note: let’s agree to call brackets, braces, and parenthesis as “symbols of interest”.

Algorithm:
Make an empty stack. Analyze characters of the parameter string one by one.
If a character is not an opening or closing symbol of interest, then move to the next character of the string.
If the character is an opening symbol, then push it onto the stack.
If the character is a closing symbol, then:
- If the stack is empty, that means there is no opening counterpart for that symbol and you can make a conclusion that the string is not balanced
- If the stack is not empty, pop the top. If the popped element is not the opening counterpart of the symbol in consideration, then you can make a conclusion that the string is not balanced; otherwise (this is the case when the closing symbol had its opening counterpart) move to the next character of the parameter string.
If all characters of the input string are scanned, check the stack – if the stack is not empty, then you can make a conclusion that the string is not balanced since there were some opening symbols in the stack whose closing counterparts were never found; otherwise (i.e. the stack is empty) the string IS balanced.

- main method is to test if your isBalanced method is working correctly. Here is the work you need to do in main:
  For as long as the user wants to (make a loop):
  a) Prompt the user to enter a string to be checked
  b) Enter the string and invoke the isBalanced method to find out whether that string is balanced or not.
  Output feedback to the user (the string is balanced or not).

III. Compile your program.

IV. Test your program thoroughly: Make sure the program works as expected. Give long strings with many symbols of interest (not just 2-3 pairs). Here are some test cases – test each case several times for different values:

a) balanced strings with no symbols of interest
b) balanced strings with several pairs of symbols of interest
c) unbalanced strings: there are opening symbols of interest that do not have closing counterparts; however, all paired symbols of interest are correctly arranged.
d) unbalanced strings: there are closing symbols of interest that do not have opening counterparts; however, all paired symbols of interest are correctly arranged.
e) unbalanced strings: all opening symbols have closing counterparts, but they are not correctly arranged.
f) unbalanced strings: there are opening symbols of interest that do not have closing counterparts and some paired symbols of interest are NOT correctly arranged.
g) unbalanced strings: there are closing symbols of interest that do not have opening counterparts and some paired symbols of interest are NOT correctly arranged.
Helpful Information:

For this assignment you need your AStack class from Lab4. Instead of copying the AStack class from the folder Lab4 into the folder Lab5, you can arrange it so that when running the program from the command line of the terminal window, Java SDK tools will consider not only Lab5 directory, but Lab4 directory as well. Let’s assume that your Lab4 directory is in the cpe103 directory which is in its turn in your home directory; the path to your Lab4 then will be ~/cpe103/Lab4 (~ stands for home directory).

You can do one of the following two things:

1) You can set the environment of the whole session of the terminal window which will hold as long as the window is open.

   For UNIX/Linux C shell, on the command line of the terminal window type:
   ```
   setenv CLASSPATH . : ~/cpe103/Lab4
   ```
   **Attention:** there is NO space between ‘.’ and ‘:’, and between ‘:’ and “~/cpe103/Lab4”

   For UNIX/Linux Bash shell, on the command line of the terminal window type:
   ```
   export CLASSPATH=. : ~/cpe103/Lab4
   ```
   **Attention:** there is NO space around any of the symbols ‘=’, ‘;’, ‘:’, and ‘/’

   Now you can simply compile and run Lab5 programs as many times as you want using SDK tools:
   ```
   javac myProgram.java and java myProgram
   ```
   **Note:** the symbol ‘.’ is for the current directory, ‘~’ is for the home directory, and ‘:’ is a separator.

   With this setting, SDK tools will look for classes in two places – the current directory (Lab5) and in the Lab4 directory. This setting is for the whole session of the terminal window, i.e. it will hold for as long as the window is open.

2) When running the SDK tool (e.g. javac or java), you can set the classpath in the command itself:

   ```
   javac –classpath . : ~/cpe103/Lab4 StringChecker.java
   ```
   **Attention:** there is NO space between ‘.’ and ‘:’, and between ‘:’ and “~/cpe103/Lab4”

   This setting will be hold only for this one command.

**Attention:** in DOS, for the options 1) and 2) you need to write respectively:

1) set classpath = . ; ~\cpe103\Lab4

2) javac –classpath . ; ~\cpe103\Lab4 StringChecker.java

**Note:** in DOS ‘;’ is the separator (instead of ‘:’) and in the path you need to use ‘\’ (instead of ‘/’).