CST246 – Homework #3
Brackets

For this project you will write a program that takes a Java source code file (e.g., the program you have just written) and validates that the brackets (parentheses ( ), brackets [ ], and braces { }) are balanced and match up. To do this you will need to use the Java Library's `Stack` class. The file you read in will come from the redirected standard input (`System.in`) (when you run the program, the command line should look something like this:

prompt> `java Brackets < Brackets.java`

You will need to use the `Scanner` class to read in lines from the input file. The source code may contain bracket characters in places other than the actual instruction statements. For example, inside of string literals (inside paired quotes), character literals (inside paired apostrophes), and line based comments (started off with the sequence `//`) (because of its complexity you should not worry about block comments, the text between `/*` and `*/` which may run over multiple lines of source code). These extra locations of bracket characters should be removed from the source lines after the lines have been read into the program (I would suggest that you write the following helper methods):

```
private String removeStringLiterals(String line)
private String removeCharacterLiterals(String line)
private String removeLineComments(String line)
```

Each of these methods will return a `String` containing the contents of the method argument with the appropriate text removed. For the first two methods consider that there might be multiple such literals on a line and *all* should be processed. I would suggest the use of the `StringBuilder` class to allow you to manipulate the `String` objects (consult the online Javadoc). If you should find and opening literal marker (quote or apostrophe) and do not find its closing literal marker throw an `IndexOutOfBoundsException` indicating the problem and the text of the line that contained the problem.

Note: the appearance of quote or apostrophe characters inside of the program, as data, causes problems with the model described above. Therefore, I would suggest the following symbolic constants be defined to assist you.

```
//    Define character symbolic constants for the quote and apostrophe.
//    This is needed since we have trouble with embedded single quotes
//          and apostrophes.
//    This is a bit of a kludge, but it works.
private static final char QUOTE_CHAR = 0x22;
private static final char APOST_CHAR = 0x27;
private static final String QUOTE = "" + QUOTE_CHAR;
private static final String APOST = "" + APOST_CHAR;
```

The characters that you should consider as *bracket characters* should be defined in two symbolic constants:

```
//   Define OPEN and CLOSE bracket characters.
private static final String OPEN  = "([{";
private static final String CLOSE = ")]}";
```

I did not include the chevron (`<` `>`) characters since they are also used as parts of relational expressions and need not appear in pairs. Whereas the compiler can distinguish such usage to properly deal with them, our simple program would be much too complex if we tried to do that.

The primary method of the class should be:

```
public void processInput()
```

This method will create a `Scanner` object instance to read from `System.in`. It will also create a `Stack<Integer>` object instance to store the index values of the OPEN characters that are pending matches with the CLOSE characters (as discussed in class).

When a line has been properly filtered (see above) then all of the remaining characters on the line should be looked at to see if there are any bracket characters. Non-bracket characters can be ignored. OPEN bracket characters are pushed onto the stack. CLOSE bracket characters are compared with the value of the popped top of the stack. If they match, continue processing. If they do not match, report this fact and terminate the program.

Note that after the input file has been exhausted there may or may not be entries left on the stack. If the stack is empty then report that the checked file is balanced. If the stack still contains entries then report that the checked file is NOT balanced and display the contents of the stack.

Note that you should check if the stack is empty before trying to pop an element off of the stack. I suggest the following statement to do this.

```
int top = stack.empty() ? -1 : stack.pop();
```

If you are unfamiliar with the ternary (`? :`) operator, look it up online.

One additional note: while writing this program is used two helpful helper methods (shown below). One to display debug information (which is triggered by the symbolic constant `DEBUG` which either has the value `true` (display the debug information) or `false` (do not display the debug information). The second is to simply report some information. Both of these methods utilize the `System.out.printf()` method and take one or more arguments (which are passed to the `printf()` method.

```
//   Define state of DEBUG flag.
private static final boolean DEBUG = false;

private void debug(String fmt, Object... args)
{    if (DEBUG) System.out.printf(fmt, args);    }

private void report(String fmt, Object... args)
{    System.out.printf(fmt, args); }
```