The goal for today’s lab is to write a simple application that uses the stack methods in Java’s generic LinkedList class. The application allows the user to type in a string and checks that the parentheses, brackets, and braces that occur in that string are correctly balanced and properly nested.

Background

A string that may contain parentheses, brackets, and/or braces is balanced if it has the same number of left and right parentheses, the same number of left and right brackets, and the same number of left and right braces. These characters are correctly nested if each right-hand parenthesis, bracket, or brace can be paired with a left-hand parenthesis, bracket, or brace that precedes it and matches its shape, in such a way that the only parentheses, brackets, and braces that occur between the paired characters are similarly paired (within the enclosed string).

So, for example, the string "(a [(b c) d]" is not balanced, because it has two left parentheses and only one right parenthesis, and the string "(a [b c])" is not correctly nested because the parentheses enclose a string containing an unpaired left bracket. But "(a [{b c} (d e)])" is both balanced and correctly nested.

One conventional method for determining that a string is balanced and correctly nested is to draw arcs above the string connecting the paired left and right characters. If there is a way to draw the arcs so that they do not cross and connect characters that match in shape, leaving no parentheses, brackets, or braces unpaired, then the test succeeds.

Exercise 1.

In Eclipse, open a new project for this application. Create a BalanceChecker.java file and write the header and documentation (though not yet the body) for a checkBalance method that determines whether a given string is balanced and correctly nested with respect to parentheses, brackets, and braces.
3. Create a `BalanceCheckerTester.java` file with a `main` method that will invoke the `checkBalance` method on a few sample strings and report its findings to the user.

4. The code that implements `checkBalance` could encounter several kinds of problems. First, when we encounter, say, a right bracket in the string and pop the stack, we might find that the character we popped is a left parenthesis rather than a left bracket. What would this imply about the given string?

5. Second, when we encounter, say, a right brace in the string, and try to pop the stack, we might find that the stack is empty. What would this imply about the given string?

6. Third, we might reach the end of the string and find that the stack still has some characters in it that were pushed early on and have not yet been popped. What would this imply about the given string?

7. Bearing all these possibilities in mind, implement the `checkBalance` method, making sure that the preconditions for all of the methods that you call are satisfied.

8. It might be a good idea for the `checkBalance` method to ensure that the stack is empty before it returns a value. (This is essential if you are storing the stack in a field of the `BalanceChecker` object and reusing it in every call to `checkBalance` — you don’t want cruft left over from one call still to be in the stack when the next call begins.) We can either pop the stack repeatedly until its `isEmpty` method returns `true`, or we can use the `clear` method of the `LinkedList` class (even though that method is not explicitly provided as part of the usual interface for stacks). Which alternative is better?

9. Save and compile `BalanceChecker.java`. Devise some test cases and add them to the `main` method in `BalanceCheckerTester.java`. Include some cases in which the bracketing symbols are balanced and correctly nested, and some in which they are not. For each of the possible kinds of error described in the exercises above, include at least one test case that illustrates an error of that kind. Compile and run `BalanceCheckerTester` and interpret the output to confirm or refute the hypothesis that your implementation handles them all correctly.