CSC1016S Assignment 8: Linked Lists, Stacks and Queues

Assignment Instructions

This assignment concerns solving problems with data structures, and reasoning about and devising tests of linked lists.

Question 1 [25 marks]

Queues are used extensively in situations where the input or output has to be delayed because of, for example, slow Internet speeds - these queues are called I/O buffers.

Write a program called `Question1.java` to simulate the operation of an I/O buffer using a linked queue data structure. You may use the built-in Java data structures.

Your program must accept a sequence of lines of text from the keyboard and process them as follows:

- If the first character is O (Oh), extract the next string from the buffer and output it. If there is no data in the queue, output "Buffer empty"
- If the first character is X, exit.
- Otherwise, store the string in the buffer.

Sample Input/Output

```
line1
line2
O
Data: line1
line3
line4
O
Data: line2
O
Data: line3
line5
O
Data: line4
O
Data: line5
O
Buffer empty
X
```

CONTINUED

Question 2 [35 marks]

Write a program to identify mismatched brackets. This happens when there are too many opening brackets ('{', '[', '(' and '<') and not enough closing brackets ('}', ']', ')' and '>'), or vice versa; or when an opening bracket of one type corresponds to a closing bracket of another type, for example if a '(' is closed by a ']'.

Write a program called `Question2.java` that takes a string as input. It should then scan the string, checking that all brackets match. If it encounters an opening bracket that does not match its corresponding closing bracket, it should print a message to that effect and then terminate any further checking as further errors will be ambiguous. If, at the end of the scan, there remain unclosed brackets or excess closing brackets, a message should be printed to that effect.

Hint: this problem is best solved using a stack, by popping brackets from the end off and storing them in another stack until an opening bracket is found. You may write your own stack implementation, or use the built-in java.util.Stack class.

It may be helpful to write other methods such as ones that check whether a bracket is an opening or closing bracket and ones that return the corresponding bracket to a given bracket. However, these are not essential.

Sample Input/Output

```
Enter a string to test:
( < [ { } ( { > ) ] >
error: '>' does not match with '{'.

Enter a string to test:
{ ( [ ] ) { ( ) ( ) } }
The string is correct! There are no mismatched brackets.

Enter a string to test:
{ ( [ ] ) { ( ) ( ) }
error at end: opening bracket '{' remains unclosed.

Enter a string to test:
( [ ] ) < > ] }
error at end: the close bracket ']' does not have a corresponding opening
bracket.
error at end: the close bracket '}' does not have a corresponding opening
bracket.
```

Question 3 [40 marks]

On the Vula assignment page you will find a class called 'SimpleLinkedList'.

The class uses an inner class called 'Node', implements the Iterable interface, and has another inner class called NodeIterator that implements the Iterator interface.

Class SimpleLinkedList

A simple linked list implementation that uses an inner Node class.

Constructors

```
public SimpleLinkedList()
        // Create an empty list.

public SimpleLinkedList(T[] items)
        // Create a list containing the given items.
```

Methods

```
public T get(int index)
        // Obtain the item at the given index.

public void set(int index, T item)
        // Replace the item at the given index with the given item.

public void add(T item)
        // Add the given item to the end of the list.

public void addAll(T[] items)
        // Add the given items to the end of the list.

public void insert(int index, T item)
        // Inserts the given item at the given index.

public void remove(T item)
        // Removes the first occurrence of the given item from the list.

public void removeAt(int index)
        // Removes the item at the given index.

public Iterator<T> iterator()
        // Obtains an iterator for traversing the list.

public int indexOf(T item)
        // Returns the index of the first occurrence of the given item in the list.
        // Returns -1 if item not in the list.

public int size()
        // Obtains the size of the list.

public void trimToSize(int size)
        // Discard all elements beyond index (size-1).
```

Write tests for SimpleLinkedList, putting them in a class called SimpleLinkedListTest (stored in the file `SimpleLinkedListTest.java`). You must include the following 4 tests:

CONTINUED

- check that a linked list created using the constructor (with a list of items passed in as a parameter) contains the items passed in.
- check that indexOf applied to search through a linked list of items returns the correct index for each item in the list.
- check that the insertion of an item in a linked list of items using insert (...) results in the item being inserted and appearing in the correct position.
- check that trimToSize (...) does truncate the list to the size specified.

Your test program should produce output in the same format as used in assignment 2 for TestStudent, TestUberService and TestCollator e.g.

e.g.

```
…
Test 10
Pass
Test 11
Fail
…
```

Each test is numbered. Output for a test consists of two lines: the first identifies the test, '`Test` *n*', and the second gives the result as either '`Pass`', or '`Fail`'.

The automarker will run your tests on a variety of faulty SimpleLinkedList classes as well as an implementation that is correct.

Marking and Submission

Submit `Question1.java`, `Question2.java`, and `SimpleLinkedListTest.java` in a single .ZIP folder to the automatic marker.

The zipped folder should have the following naming convention:

yourstudentnumber.zip

END