# Linked List Operations and Recursion

This lab will ask you to revisit an old friend: the linked list. However, instead of implementing solutions iteratively, we choose to live dangerously and implement several functions recursively.

## LinkedList Methods

The provided skeleton of the `LinkedList<Item>` class implements a singly linked list with sentinel nodes: head (`_head`) and tail (`_tail`). Your goal will be to implement the operations described in Table 1.

**Table 1: Recursive Implementation for the `LinkedList` class**

| Return Type | Method | Description |
|---|---|---|
| `boolean` | `contains(Item target)` | Returns true if the list contains `target`. |
| `void` | `clear()` | Removes all elements from the list. |
| `Item` | `middle()` | Returns the element in the middle of the list. For an even length list, the 'left' middle value will be returned (e.g., `middle([1, 2, 3, 4])` returns `2`.<br>You may not count the number of nodes or use the size method in your implementation.<br>Throws a `NoSuchElementException` when the list is empty. |
| `void` | `reverse()` | Reverses all the elements in the list. |

It is suggested that you implement and test these methods one at a time as well as *in the order presented above*. Reversing a linked list is something you should work out on paper before implementing.

To facilitate these operations, it suggested you implement some support methods as described in Table 2.

**Table 2: Support Methods**

| Return Type | Method | Description |
|---|---|---|
| `boolean` | `atEnd(Node n)` | Returns true if `n` is the tail node. |
| `boolean` | `atNextToEnd(Node n)` | Returns true if `n` is the node prior to the tail node. |
| `Node` | `deleteNextNode(Node n)` | Deletes the node from the list that is after node `n`.<br>If a linked list contains `n x y`, the resulting list will be `n y` (deleting `x`) and node `n` is returned.<br>Throws a `RuntimeException` when the given node does not point to a valid list node. |

### JUnit Testing

A small set of unit tests have been provided. Please add more. ***No output*** should be produced by your tests; we are seeking only a 'green' output indication in Eclipse.

## Submitting Source Code

Your code should be well documented, including docstring comments of methods, blocks of code, and header comments in ***each*** file.

Testing code needs fewer comments as they should be self-descriptive; however, it is recommended that each individual test or family of tests be numbered and have a brief comment.

### *Header Comments*

Your program must use the following standard comment at the top of *each source code file*. Copy and paste this comment and modify it accordingly.

```
/**
 * Write a succinct, meaningful description of the class here. You should avoid wordiness
 * and redundancy. If necessary, additional paragraphs should be preceded by <p>,
 * the html tag for a new paragraph.
 *
 * <p>Bugs: (a list of bugs and / or other problems)
 *
 * @author <your name>
 * @date   <date of completion>
 */
```

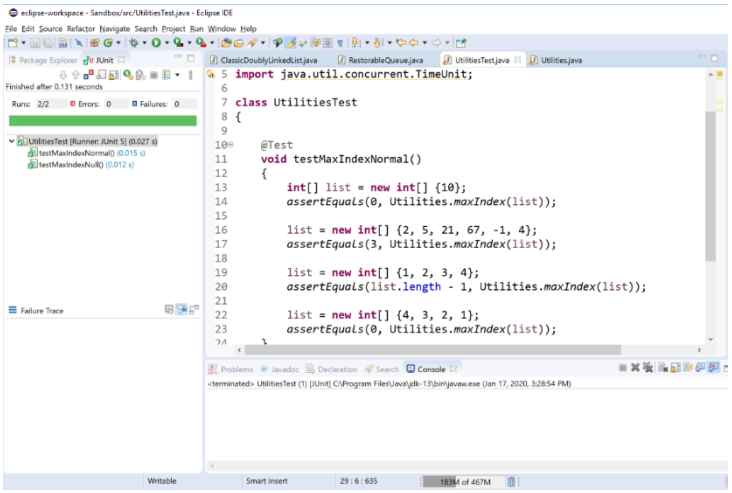
### *Inline Comments*

Comment your code with a *reasonable amount of comments* throughout the program. Each method should have a comment that includes information about input, output, overall operation of the function, as well as any limitations that might raise exceptions; Javadoc comments are ideal. Each *block* of code (3-4 or more lines in sequence) in a function should be commented.

It is ***prohibited*** to use *long* comments to the right of lines of source code; attempt 80 to 100 character-wide text in source code files.

## Submitting; Proof of Program Execution

Execute your code and take a screenshot of the associated output console. Place these screenshots into a word processing document (Word, OpenOffice, GoogleDocs, etc.). If multiple screenshots are necessary, label each clearly. Please make sure to crop and enlarge the screenshots so that the picture and / or text is clear (and doesn't strain my old eyes). For example, ***the screenshot on the next page is not appropriately sized*** although it contains ideal information (output console, code, etc.). Create a PDF of this document and call it `evidence.pdf`.



***Source Code Files***

You are to submit your entire project folder (including any files provided to you).

***Final Submission File***

In a folder named `lab`, place (1) the project code folder and (2) `evidence.pdf`. Zip folder `lab` and label that zip file as `lab.zip`. This zip file is to be submitted via Moodle.