### Portfolio Function Title: LinkedList

Version number: 1

### Function Description:

Allows for a user to build singly linked list of objects. Once a linked list is built a user can add or remove an object to in any position in the linked list including after a reference object that is already in the list. Users can also get the length of the list, check if it is empty and check if the list contains a certain object.

### Author:

Nathan Mankovich

### Date Written/Last Modified:

04-27-2015 / 05-01-2015

### Design overview:

The function relies on the Node class that is an inner class of LinkedList. This class creates nodes which are populated with an object (called data) and a next Node (called next). To link two nodes together the first node’s “.next” must be equal to the second node. What this does is point the first node’s “.next” the same as the second node’s pointer to a memory location.

The “head” of the list is the pointer to the first node in the linked list. So, to add an element to the front of the list the head must point to a new node that contains the added object.

Whenever an object is added to the list at a location other than the end, that object’s node must first point to the first node that follows it.

Note -- the semantics in terms of where exactly each node is stored in memory depends on the language used (ex: Java is different than C).

If an object is added to the list the integer variable called length is incremented by on. Conversely, if an object is removed from the list length is decremented by one.

If the list is empty, then the head of the list is null.

### Feature Specifications:

Generates Node objects which have a data field to store an element (in LinkedList.java element of generic type E) in the list and a next field to point at the next node.

Points the head to the first Node.

Links Node objects together using “.next”

Adds elements to the front or the back of the list.

Inserts elements after an input element or at an index location (the first element is at zero).

Removes elements from the front or the back of the list.

Removes a specific input element or at an element at the input index location (the first element is at zero).

Checks if the list is empty.

Return the size of the list.

### Programmer User Interface:

User Interface: Java code and command line only.

### Input and Output Requirements and Restrictions:

* AddToFront:
  + Input: object to be added
  + Output: none
* AddToBack:
  + Input: object to be added
  + Output: none
* AddAfter:
  + Input:
    - the object to be added
    - the reference object that the added object is to be put after
  + Output: none
* InsertAtPostion:
  + Input:
    - object to be added
    - integer position where it is to be added (the first element is zero)
  + Output: none
* RemoveFirst:
  + Input: none
  + Output: the removed object
* RemoveLast:
  + Input: none
  + Output: the removed object
* RemoveCertain:
  + Input: the object to remove
  + Output: the removed object
* GetFirst:
  + Input: none
  + Output: the first object in the list
* GetLast:
  + Input: none
  + Output: the last object in the list
* GetAtPosition:
  + Input: an integer of an object in the list (the first element is zero)
  + Output: the object at location of the input integer
* GetLength:
  + Input: none
  + Output: the length of the list
* CheckIfEmpty:
  + Input: none
  + Output: a Boolean, true if empty
* CheckIfContains:
  + Input: object
  + Output: a Boolean, true if the list contains the input object

### Assumptions and Dependencies:

The linked list assumes that a user will not use any removing or getting methods without checking to see that there are elements in the list.

### Known problems and limitations:

If the any removing or getting methods is used on an empty list, the program will return a NoSuchElementException. This can be fixed in future versions of LinkedList by always checking if the list is empty before running any of these processes.

Singly linked lists are notorious for processing times for adding or accessing elements at the end of the list that increase linearly as the list size increases. This happens because the program must iterate through the entire list. Since checking the size of the list requires iterating through the entire list, it also shares similar big O characteristics with other end-of-list processes.

Also, there is no iterator class in LinkedList all list-traversing code lies in whichever method that needs to use it. Future versions could clean up the code and make a forward iterator that keeps track of the three nearest elements in order to insert, remove or access certain elements in the list.

### Use Cases:

Implementation in a stack class due to easy access to the top of the stack via the head.

For any process where access to the last added element matters the most and the length of the list does not need to be monitored. The methods for this application would generally be AddToFront, GetFirst, and RemoveFirst.

### Testing Methodology:

Be sure to put the tLinkedList.java and the portfolio.jar file in the same folder.

To compile:

javac -cp .;portfolio.jar tLinkedList.java

To run:

java -cp .;portfolio.jar tLinkedList > LogLLTest.txt

All command line outputs will be in the new LogLLTest.txt file in the folder that contains the tLinkedList and the portfolio.jar files.

### Modification history:

Version 1: 05-01-2015

### Design detail and/or Diagrams:

When the program starts to run the head is initialized at null, the length counter is initialized at zero.

Each time an element is added to the list, a new node with data = element and it is linked in its proper place. See design overview for linking specifics. Also, adding an element increments the length counter by one.

Each time an element is removed from the list the links are arranged so they skip the removed element and the data is left unaccusable until the jvm garbage collector picks it up. Also, removing an element decrements the length counter by one.

All methods in the LinkedList are public so that they can be accessible to other programs.