### Portfolio Function Title: Stack

Version number: 1

### Function Description:

Allows for a user to build a stack of objects. Once a stack is built a user can push (add), pop (remove) and peek (get) on the top of the stack. A user can also find the size of the stack (GetSize) and check whether it is empty (CheckIfEmpty).

### Author:

Nathan Mankovich

### Date Written/Last Modified:

04-27-2015 / 05-01-2015

### Design overview:

Stack relies on the instantiation of the LinkedList.

Each method in Stack.java calls the LinkedList program to fill a singly linked list that was instantiated at the beginning of Stack.java.

The Stack is populated in the push method. Objects are removed from the top and returned using the pop method. The peek method gets the object at the top without removing it. The CheckIfEmpty method checks if the stack is empty and returns a true Boolean if it is, otherwise the method returns a false boolean. The GetSize method returns the size of the stack as an integer.

### Feature Specifications:

Push elements onto the top of the stack using AddToFront from LinkedList.

Pop elements off the top of the stack using RemoveFirst from LinkedList.

Peek at the first element on the top of the stack using GetFirst from LinkedList.

Get the size of the stack as an integer using the CheckIfEmpty from LinkedList.

Determine whether there are elements in the stack using the GetLength from LinkedList.

### Programmer User Interface:

User Interface: Java code and command line only.

### Input and Output Requirements and Restrictions:

* push:
  + Input: object
  + Output: none
* pop:
  + Input: none
  + Output: removed object
* peek:
  + Input: none
  + Output: the first object
* CheckIfEmpty:
  + Input: none
  + Output: a Boolean, true if empty
* GetSize:
  + Input: none
  + Output: an integer that is the length of the stack

### Assumptions and Dependencies:

The stack 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:

A stack is limited to access at the front of the list of elements. Yet, all functions in the LinkedList implementation of stack provides fast processing times.

### Use Cases:

Stacks can be used for data sets that need to be reversed in order. All that needs to be done for that is push all the elements onto the stack, then pop them off.

Stacks are best used for data where elements do not need to be accessed after another element has been added to the stack.

### Testing Methodology:

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

To compile:

javac -cp .;portfolio.jar Stack.java

To run:

java -cp .;portfolio.jar tStack > LogStackTest.txt

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

### Modification history:

Version 1: 05-01-2015

### Design detail and/or Diagrams:

When the program starts, an LinkedList is instantiated and all methods in the Stack call methods in the LinkedList class. For more information about the implementation of the stack- specifically the storage of elements in a linked list, see the LinkedListPortfolioDesignDoc in the portfolio folder.