Week 5: Linked-Lists

If required, refresh your knowledge on Linked-Lists by revisiting the lecture slides and your notes from the last lectures on the structure of and operations on a linked-list.

For this Practical, you will find on Blackboard a Netbeans project providing you with some incomplete source code and some unit-tests to check the validity of your code. Download and save the folder in your U: Drive. Then open Netbeans, click on File -> Import Project -> from ZIP and open the LinkedList_App folder.
You will find three source files:
ListNode.java – a class for a list node – have a good look at it. Do not modify this class.
LinkedList.java – a class for a linked-list – have a good look at it. Note that the constructors have been provided for you. <u>You need to add the methods below in this class.</u>
LinkedList_App.java – a driver class to exercise the good operation of your linked-list.

Your job is to provide your Linked-List with the expected operations of a Linked-List ADT.

Each of the operations below has been given a method signature and a description in the **LinkedList.java** class. Your task is to give each method a body with the appropriate code to fulfil the required operation. I would suggest you tackle the methods in the order that they are listed below.
If a method returns a value, it has been commented out to allow you to compile the file without getting lots of errors due to incomplete code.
So whenever you decide to work on a method, remove the comment symbol (//) and attempt to write the necessary code.
Similarly, the code in the driver class **LinkedList_App** has also been commented out. As you add completed methods to the class **LinkedList**, uncomment the bit about the code that tries to exercise that particular method. Then compile and run.
Repeat as you add new methods to class **LinkedList**.

Implement the methods below in the **LinkedList.java** class. Once you have completed a method, read the section below (Unit-Tests) to test your method. You are expected to complete all the methods below during this lab.

int getSize() – return the size of the linked list

void addFirst(int value) – add a node at the beginning of the list
String toString() – return a description of the content of the linked list. If the list is a = {1, 4, 0, 6, 3}, the returned string should be in the format “1\n4\n0\n6\n3\n” - \n means “new line”. Return an empty string “” if the list is empty.
int getFirst() – retrieve the data stored in the first node of the list, if it exists. Return 999 if the list is empty.
int getLast() – retrieve data in the last node of the linked list, if it exists. Return 999 if the list is empty.
void addLast(int obj) – add a node at the end of the list with data value obj
ListNode searchPointer(int n) - return a pointer to node containing n, or null if not found.
int searchIndex(int n) - return a one-based index to node containing n, or -1 if not found
boolean delete(int n) – delete the nth node in the list in a one-based index. Return true if the node is deleted, Return false if the nth node does not exist.
int getAtPos(int index) – retrieve data in node located at position index (in a one-based index), if it exists. Return 999 if the node does not exist.
boolean addAfterPos(int obj, int index) – add a node after position index (in a one-based index), if it exists. Return true is the new node has been added successfully. Return false of the node at position index does not exist.

Testing your methods – Unit-Tests

When you think your code for a particular method is ready, to help you in debugging your code, you have been provided with a suite of unit-tests. These will help you test your individual methods. You will find them, in Netbeans, under the source code in a directory called **Test Packages**.
Each method that you have been asked to write has a corresponding testfile (or unit-test). So for example, the method **getFirst()** has a corresponding testclass called **GetFirstTest**.
If you double click on it, the corresponding class will open, revealing the test code (the unit-tests).
For each method, there is a selection of tests (between 2 and 10).
Each test starts with @**Ignore**, this is to give you control over which test to run at any one time. So at the moment they are disabled. When you are ready to test your method, comment out that line to enable that test, like this
//@Ignore and it will now be possible to run that particular test.
DO NOT ALTER THE CODE INSIDE THE TESTS!

To run a testfile, right click on the testfile of your choice, for example **GetFirstTest**, and select **Test File**. Only the tests that you have enabled will run; the others will be skipped until you enable them.

Initially there will be lots of errors but these will go as you flesh out your methods.

Hopefully, you will find the messages of the various tests useful, as they have been designed to point out where in your code there are errors.

Remember, unit-tests do not test for syntax errors, but check whether or not the functionality of the method has been fulfilled.

Make sure you ask your tutor, if you feel unsure how to proceed.

When you have written all the required methods, right click on the project name (LinkedList_App) and select *Test*. This runs all unit tests in the project. If they all pass, then you have completed this practical!

Advanced work

Add the methods below to your ADT:

1. A method that removes all nodes in the list that contain a certain search value and returns the number of nodes deleted.
2. A method that does the same thing, but instead of returning the number of nodes deleted, returns a pointer to the beginning of the "deleted" list.

If you have completed all the tasks in this lab session, start working on your ICA, programming task. Start from Table 1, which is based on Singly Linked List.

Remember that all the methods in the ICA, use a 0-based index system. Hence, the first node in the list is the node at index 0 not 1.