Week 6: Circular Linked-Lists

In this lab session, you will work on Circular Linked lists.
If you have not completed the tasks on Linked Lists in Practical–Week-5 (last week's), make sure you do so before next week.

For this Practical, you will find to download a Netbeans project from Blackboard:

- **C_LinkedList_app.zip** for circular linked-lists and

Like last week, each project will provide you with some incomplete source code (three files: two data-classes and a driver class to exercise the good operation of your linked-list) and some unit-tests to check the validity of your code.
Your job is to provide your Linked-List with the expected operations of a Linked-List ADT.

Tasks

Add the methods below in the C_LinkedList.java class (Circular Linked Lists).

boolean isEmpty() – check if the list is currently empty
int getSize() – return the size of the linked list
String toString() – return a description of the content of the linked list, with each item below the previous one. Example: list 2->4->5->7-> null. String = "\n2\n4\n5\n7"
int getFirst() – retrieve the data stored in the first node of the list. Return 999 if the list is empty.
int getLast() – retrieve data in the last node of the linked list. Return 999 if the list is empty.
Void addFirst(int obj) – add a node at the beginning of the list
Void addLast(int obj) – add a node at the end of the list
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 (one-based index). Return true if the node has been deleted, false otherwise.
Boolean addAfterPos(int obj, int index) – add a node with value obj, after position index, if it exists. Return true if the node has been added, false otherwise (one-based index).
Int getAtPos(int index) – retrieve data in node located at position index. Return the data in the node at position index if it exists, 999 otherwise (one-based index).

Each of these operations has been given a method signature and a description. 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 above.
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 **C_LinkedList_App** has also been commented out. As you add completed methods to class **C_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 **C_LinkedList**.

Using the 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 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.

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 (**C_LinkedList_App**) and select *Test*. This runs all unit tests in the project. If they all pass, then you have completed this practical!

Extension Task – ICA tasks

Check in your ICA document the methods that you need to implement in the Circular Linked Lists for you ICA. You might have already implemented some of these methods. Add the remaining methods to the class you have been working on during this lab session. This will help you with your ICA. **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.**