**Student Name: Weight: 20%**

**Student ID:** **Marks:** **/100**

# Assignment: Singly Linked Lists, Serialization and Testing

## Background

A **linked list** is a linear data structure that is formed of a sequence of nodes. Each node contains two parts: Data and Address, which are dynamic and allocate memory as and when required. Insertion and deletion are easy to implement using linked lists, and items in a linked list can be accessed quickly. They are also used to implement other data structures such as Stack and Queue.

**Serialization** is the process of converting an object into a form that can be stored in a file, database or memory, or transferred across a network. Its main purpose is to save the state of the object so that it can be recreated when needed.

**Unit Testing** is a process of testing individual parts of a code to make sure that it works as expected. A unit test is a code implemented by a programmer that tests a small piece of functionality of a larger program.

## Equipment and Materials

For this assignment, you will need:

* Visual Studio IDE
* The supplied interface **LinkedListADT**
* The supplied **LinkedListTest** class
* The supplied **SerializationTests** class unit test cases

## Instructions

1. Create the code for a program that meets the requirements described bellow.
2. Test your code against the expected output provided.
3. Check your program against the detailed marking criteria at the end of this document.
4. Submit the following to Brightspace as a group (*Only one copy is required per group, and any of the group members may submit*):

* GitHub URL for your program code (invite your instructor to be a member of the project repository)
* Screen shots of the test cases run

**Note:** Follow the folder structure outlined in Table 1.

**Peer Assessment (5%)**

Each student must also complete a peer assessment of their group members. Your instructor will provide further submission details.

## Program Details

In this assignment, your goal is to implement an abstract data type (ADT) for a linked list.

* The supplied interface **LinkedListADT**contains the signatures for the methods.
* The SLL class must implement the LinkedListADT interface.
* Each abstract method defined in the LinkedListADT should be completely implemented, and any exceptions are to be appropriately propagated in the SLL class.
* The Node class represents a node in the linked list.
* Ensure that the linked list interface can perform the following functionality.

**Important:** Don’t use already implemented classes, methods or libraries.

* + Prepend an item to the beginning of the linked list.
  + Append an item to the end of the linked list.
  + Remove an item at an index in the linked list.
  + Remove an item from the start of the linked list.
  + Remove an item from the end of the linked list.
  + Insert an item at a specific index in the linked list.
  + Replace an item in the linked list.
  + Get an item at an index in the linked list.
  + Get the index of an item in the linked list.
  + Check if the linked list has an item.
  + Clear all items in the linked list.
  + Get the number of items in the linked list.

Place the supplied interface (LinkedListADT), the implementing class (SLL) and the Node class in the utility folder mentioned in Table 1.

### Unit Testing

The supplied **LinkedListTest** class is a test suite that performs unit testing on an implemented linked list. The following unit test cases have been provided:

* The linked list is empty.
* An item is prepended.
* An item is appended.
* An item is inserted at index.
* An item is replaced.
* An item is deleted from beginning of list.
* An item is deleted from end of list.
* An item is deleted from middle of list.
* An existing item is found and retrieved.

**Important:** Some of the test cases have already been implemented. Do not modify the implemented test cases. Develop and implement any **missing** test cases to achieve the needed code coverage.

### Serialization

* The objects in the linked list SLL must be serialized, and the SLL object must be stored in binary format. You must be able to reconstruct the object from its binary form.
* Use the supplied **SerializationTests** class unit test cases to test for object serialization of the SLL.

**Note:** Do not make any changes to the unit test suite.

* All the unit test cases need to pass with no errors. The objects are serialized to memory in the unit test suite and, therefore, no changes are made on the hard drive.
* Use the supplied User class in the **problemdomain** folder as the data type for the items in the linked list.
* Following the object-oriented principles, your project should contain ONLY the following classes and methods in their respective packages.

Table 1 – Package Folder Structure Outline

|  |  |  |
| --- | --- | --- |
| **Folder** | **Class** | **Methods** |
| utility | LinkedListADT | append, insert, prepend, set, size, get, indexOf, contains, isEmpty, clear, and remove |
|  | SLL | Implementation of the above interface methods |
|  | Node | Getters and setters |
| problemdomain | User | Getters, setters, and equals |

### Expected Output

All the test cases should pass. As well, you should implement at least five additional test cases for the SLL).

## Marking Criteria

### Peer assessment/Attribution list Submission

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Not submitted (0%)** | **Submitted (100%)** | **Marks** |
| **Peer assessment** | * Not submitted | * Completed for the group member | **/5** |

### Group Submission

|  | **Needs Improvement (0–50%)** | **Good (51–75%)** | **Excellent (76–100%)** | **Marks** |
| --- | --- | --- | --- | --- |
| **Working code** | * The project doesn’t run in all scenarios * Many syntax errors | * The project runs in all scenarios but not all tests pass * Some syntax errors | * The project runs in all scenarios and all tests pass * No syntax errors | **/20** |
| * Linked List ADT not added * Incorrect folder structure | * Linked List ADT added * Incorrect folder structure | * Linked List ADT added * Uses proper folders structure. | **/5** |
| The following operations don’t work:   * Add and replace items in linked list * Remove items from linked list * Retrieve items from linked list * Find existing items in linked list * Clear and check if linked list is empty | The following operations partially work:   * Add and replace items in linked list * Remove items from linked list * Retrieve items from linked list * Find existing items in linked list * Clear and check if linked list is empty | The following operations work correctly:   * Add and replace items in linked list * Remove items from linked list * Retrieve items from linked list * Find existing items in linked list * Clear and check if linked list is empty | **/25** |
| SLLTesting (test cases fail):   * Linked List unit testing * Uses implemented linked list ADT as test fixture * Created and implemented new JUnit test cases | SLLTesting (some test cases fail):   * Linked List unit testing * Used implemented linked list ADT as test fixture. * Created and implemented new JUnit test cases | SLLTesting (test cases pass):   * Linked List unit testing * Used implemented linked list ADT as test fixture. * Created and implemented new JUnit test cases | **/15** |
| Serialization unit testing (test cases fail):   * Uses implemented linked list ADT as test fixture * Writes linked list using object stream * Reads linked list using object stream * Applicable data is ignored in serialization process | Serialization unit testing (part of the test cases fail):   * Uses implemented linked list ADT as test fixture * Writes linked list using object stream * Reads linked list using object stream * Applicable data is ignored in serialization process | Serialization unit testing (test cases pass):   * Uses implemented linked list ADT as test fixture * Writes linked list using object stream * Reads linked list using object stream * Applicable data is ignored in serialization process | **/15** |
| **Style** | * Indentation – inconsistent * Readability – poor variable names * Documentation * No comments are included at the top. * No comments indicating major code sections or what they do | * Indentation – some parts are consistent * Readability – some variable names are not ideal * Documentation * Comments at the top are missing or incomplete. * Comments indicating major code sections and what they do are incomplete | * Indentation – consistent * Readability – good variable names * Documentation * Comments at the top are complete and include name, date, program description * Comments indicate major code sections and what they do | **/15** |
| **Total** | | | | **/100** |