Lab 3: Linked Lists

F28SG – Introduction to Data Structures and Algorithms (6 marks)

Plagiarism policy

This lab is not group work, and you are assessed individually.

Therefore, the work you submit for this lab **must be entirely your own**. You must not share your code solution with other students, and you must not copy code solutions from others. The university plagiarism policy is clear:

> *"Plagiarism involves the act of taking the ideas, writings or inventions of another person and using these as if they were one's own, whether intentionally or not."*
> ***Definition 2.1.***

The disciplinary action for plagiarism is an award of an F grade (fail) for the course. Serious instances of plagiarism will result in Compulsory Withdrawal from the university.

The project is organized as follows:

- The src directory contains all the source files
  - **LinkedList.java** is the implementation of a linked-list class from the lecture, with the difference that it stores integers and not objects.
  - **Stack.java** is the skeleton implementation of Stacks using linked lists. You should complete this.
- The test directory contains the unit tests for the project.

Q1) Unit tests for iterating linked list (1 point)

In Q2 you will implement two methods to iterate over the list. These are

- `int size()` which should return the number of nodes in the linked list, and 0 for an empty linked list.
- `int total()` which should sum up the values in the linked list, and 0 for an empty linked list.

Empty skeleton implementations of the methods can be found in **LinkedList.java.** Your task in this part is to implement suitable Junit tests for these operations, covering

- empty lists
- list with more than one element

Your task for Q1 is to complete these tests in **LinkedListTest.java**.

When testing `size()` and `total()` on linked lists with multiple elements, complete a method that calls `size()` ***twice in succession*** and use an assertion to check the result returned from `size()` both times. Complete the similar test for calling the `total()` method twice. These successive calls will check that their implementations do not break the `headNode` reference inside **LinkedList.java.**

We're using Test Driven Development (TDD), so at this point your tests will fail.

Q2) Iterating over the List (2 Points)

Your task here is to complete the implementation of the `size()` and `total()` methods described in Q1. You can find empty skeleton implementation for these methods (with dummy return values) in **LinkedList.java**. Implement them and ensure that your tests from Q1 all pass.

State the Big-Oh complexity as a comment above the `size` and `total` methods.

Q3) Implementing and Testing a Stack (2 Points)

The **Stack.java** class contains a skeleton implementation of a Stack using a Linked List. Your task is to complete this by implementing the following methods:

- `push(int o)`
- `pop()`
- `top()`

There is no need to instantiate the `LinkedList` class. Instead, you can build linked nodes starting from the `top` node and the inner `Node` class in **Stack.java.**

State the Big-Oh complexity as a comment above the `push`, `pop` and `top` methods.

See **Hint 1** if you need help.

Predefined tests are provided for you in **StackTest.java.** Use Test Driven Development (TDD) to implement top, pop and push.

Q4) Code Quality (1 point)

Code quality is vitally important for so many reasons. Not least, for readability and maintainability, not just for yourself but for others too since in industry, software engineering is almost always a group exercise. Real world software engineering is mostly about refactoring and testing code, rather than writing new code (more code means higher maintenance costs!).

An additional mark is awarded if your code is deemed to be of high quality:

- **Big-Oh**
  - Has Big-Oh complexity been documented for all methods where requested?
- **Code simplicity**
  - Is the code as short as it can be?
  - Is the Big-Oh complexity as small as it can possibly be?
  - Is the control flow (loops, while statements, if statements, etc.) simple to follow?
- **Documentation**
  - Are you using comments ***inside your implementation methods and test methods*** to provide an algorithmic commentary about what the code is doing.

  Here is a **good** comment:

```
// creates unidirectional connection from the predecessor node to the new node
prevNode.nextNode = newNode;
```

Here is a **less useful** comment:

```
// sets preNode.nextNode to newNode
prevNode.nextNode = newNode;
```

  - Is Javadoc syntax used for documenting the code? ***Hint!*** In Eclipse move your cursor to the text definition of a field, method or a class, then use the following keyboard shortcut: ***Alt + Shift + j*** , and then fill in the generated Javadoc template. If you are using MacOS, the shortcut is: ***⌘ + Alt + j***

- **Conventions**
  - Are sensible Java code conventions used, e.g. for code indentation, declarations, statements, etc? See Sections 4, 5, 6, 7 and 9 of *Java Code Conventions*. ***Hint!*** *Use the keyboard shortcut in Eclipse:* ***Control + Shift + f*** , or on MacOS: ***⌘ + Shift + f***

Reversing the List (optional)

Implement a method that reverses the list. The test method `testReverse()` in `LinkedListTest.java` will test your implementation.

Hint 1: *In the linked list lectures, we've had an object reference called "headNode". In this lab, Stack.java instead has an object reference called "top", reflecting that we're implementing a stack and we have a reference to the top element.*