COMP 7035
LINEAR DATA STRUCTURES & ALGORITHMS

ASSIGNMENT 1 – Linked Lists, Stacks & Queues

SUBMISSION & DUE DATE

This assignment should be submitted to Canvas before 11:59pm on **Friday 18/10/2024**. The usual late submission penalties apply in accordance with MTU's marks and standards policy.

Please submit a single ZIP file with your student number and name in the filename. Your submission should contain **exactly 5 files**:

- *DoubleLinkedList.java*
- *ArrayStack.java*
- *DynamicStack.java*
- *ArrayQueue.java*
- *DynamicQueue.java*

Please do NOT include any other files in your submission.

Make sure that you document your code and include sufficient comments where appropriate. Explain exactly what you are implementing and provide arguments for the correctness of your solution.

EVALUATION PROCEDURE

You can achieve a total of 25 points for the submission as indicated in the tasks. For each subtask you are given full marks for correct answers in your submission, 70% for minor mistakes, and 35%, 15%, or 0% for major mistakes or omissions depending on severity.

To mitigate against any form of plagiarism, including but not limited to the use of generative AI tools, **you MUST also present and explain your solution and confidently answer questions** about your submission to be graded. This oral assessment will take place during the scheduled labs after the submission deadline (or the week after, should we run out of time during these sessions).
If you do not show up your submission will not be graded unless there are approved IECs!

Your ability to fully explain your own submission will be crucial. You will only receive full marks if you are able to confidently present and explain all your work. Your assignment grade will be capped at 70% or 50% in case of minor issues with your explanations, and at 35%, 15%, or 0% in case of major problems with your explanations during the oral assessment.

The oral assessment will cover everything you uploaded onto Canvas, so be very careful what you include into your submission and make sure that you **only submit what you fully understand** to avoid your final marks being capped; in particular, do NOT include anything, including snippets taken from labs or lectures, unless you are able to explain it in detail or risk the whole rest of the assignment being marked down.

TASK 1 (linked lists, 8 points)

In this task you will implement the following interface

```
public interface List<T> {
    void prepend(T x);
    T getFirst();
    void deleteFirst();
    void append(T x);
    T getLast();
    void deleteLast();
    boolean empty();
}
```

as a double linked list. Download the Java source files *Assignment_LDSA_1.zip* from Canvas and edit the file *DoubleLinkedList.java* to implement the missing code in

```
public class DoubleLinkedList implements List<Object>
```

by replacing the `RuntimeException("Not implemented yet!")` where indicated. Use the provided internal class `private class ListNode` to store the list nodes and add more data fields as required. Make sure that the `prev` and `next` pointers are always correctly maintained.

A. Initialise an empty list in the constructor [1 point]

```
public DoubleLinkedList()
```

B. Write a function to insert a new element at the first position into the list [1 point]. Implement this function in the method

```
public void prepend(Object x)
```

C. Write a function to return the first element from the list [1 point]. Implement this function in the method

```
public Object getFirst()
```

D. Write a function to delete the first element from the list [1 point]. Implement this function in the method

```
public void deleteFirst()
```

E. Write a function to insert a new element at the last position into the list [1 point]. Implement this function in the method

```
public void append(Object x)
```

F. Write a function to return the last element from the list [1 point]. Implement this function in the method

```
public Object getLast()
```

G. Write a function to delete the last element from the list [1 point]. Implement this function in the method

```
public void deleteLast()
```

H. Write a function to determine if the list is empty [1 point]. Implement this function in the method

```
public boolean empty()
```

TASK 2 (stacks, 8.5 points)

In this task you will implement the following interface for the stack abstract data type

```
public interface Stack<T> {
    void push(T x);
    T pop();
    T peek();
    boolean empty();
}
```

A. Your first implementation should store the objects in a static array

```
private Object[] S
```

Continue using the source files from the previous task and edit the file *ArrayStack.java* to implement the missing code in

```
public class ArrayStack implements Stack<Object>
```

by replacing the `RuntimeException("Not implemented yet!")` where indicated.

a. Initialise an empty stack able to store a given maximum number of elements in the constructor [1 point]

```
public ArrayStack(int capacity)
```

b. Write a function to push an element onto the stack by implementing the method [1 point]

```
public void push(Object x)
```

c. Write a function to remove and return an element from the stack by implementing the method [1 point]

```
public Object pop()
```

d. Write a function to inspect the topmost element on the stack by implementing the method [1 point]

```
public Object peek()
```

e. Write a function to determine if the stack is empty by implementing the method [1 point]

```
public boolean empty()
```

B. For the implementation in the previous task the maximum number of elements on the stack must be know in advance. At the expense of slightly more overhead a dynamic implementation using the data structure implemented in TASK 1 of this assignment can be used instead to store the objects

```
List<Object> S = new DoubleLinkedList();
```

Continue using the source files from the previous tasks and edit the file *DynamicStack.java* to implement the missing code in

```
public class DynamicStack implements Stack<Object>
```

by replacing the `RuntimeException("Not implemented yet!")` where indicated.

a. Write a function to push an element onto the stack by implementing the method [1 point]

```
public void push(Object x)
```

b. Write a function to remove and return an element from the stack by implementing the method [1 point]

```
public Object pop()
```

c. Write a function to inspect the topmost element on the stack by implementing the method [1 point]

```
public Object peek()
```

d. Write a function to determine if the stack is empty by implementing the method [0.5 points]

```
public boolean empty()
```

TASK 3 (queues, 8.5 points)

In this task you will implement the following interface for the queue abstract data type

```
public interface Queue<T> {
    void enqueue(T x);
    T dequeue();
    T next();
    boolean empty();
}
```

A. Your first implementation should store the objects in a static array

```
private Object[] Q
```

Continue using the source files from the previous task and edit the file *ArrayQueue.java* to implement the missing code in

```
public class ArrayQueue implements Queue<Object>
```

by replacing the `RuntimeException("Not implemented yet!")` where indicated.

a. Initialise an empty queue able to store a given maximum number of elements in the constructor [1 point]

```
public ArrayQueue(int capacity)
```

b. Write a function to add an element to the queue by implementing the method [1 point]

```
public void enqueue(Object x)
```

c. Write a function to remove and return an element from the queue by implementing the method [1 point]

```
public Object dequeue()
```

d. Write a function to inspect the next element in the queue by implementing the method [1 point]

```
public Object next()
```

e. Write a function to determine if the queue is empty by implementing the method [1 point]

```
public boolean empty()
```

B. For the implementation in the previous task the maximum number of elements in the queue must be know in advance. At the expense of slightly more overhead a dynamic implementation using the data structure implemented in TASK 1 of this assignment can again be used instead to store the objects

```
List<Object> Q = new DoubleLinkedList();
```

Continue using the source files from the previous tasks and edit the file *DynamicQueue.java* to implement the missing code in

```
public class DynamicQueue implements Queue<Object>
```

by replacing the `RuntimeException("Not implemented yet!")` where indicated.

a. Write a function to add an element to the queue by implementing the method [1 point]

```
public void enqueue(Object x)
```

b. Write a function to remove and return an element from the queue by implementing the method [1 point]

```
public Object dequeue()
```

c. Write a function to inspect the next element in the queue by implementing the method [1 point]

```
public Object next()
```

d. Write a function to determine if the queue is empty by implementing the method [0.5 points]

```
public boolean empty()
```