Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

uOttawa

Introduction to Computing II (ITI 1121)
Final Examination: Part 2 of 2

Prof. Wail Mardini

April 20, 2022, duration: 110 minutes (exam) + 10 minutes (submission) = **120 minutes**

Instructions

1. This is an open book examination, and the only authorized resources are:
   - Course lecture notes, laboratories, and assignments.
   - Java Development Kit (JDK) on your local computer.
2. *By submitting this examination, you agree to the following terms:*
   - **You understand the importance of professional integrity in your education and future career in engineering or computer science.**
   - **You hereby certify that you have done and will do all work on this examination entirely by yourself, without outside assistance or the use of unauthorized information sources.**
   - **Furthermore, you will not provide assistance to others.**
3. Anyone who fails to comply with these terms will be charged with academic fraud.

Marking scheme

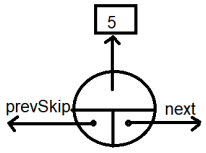
Questions	Maximum
1	**30**
2	**10**
Total	**100**

Question 1 (30 marks)

For this question, you need to write a program that implements a **doubly-like linked structure.** Your code should be able to create such a structure and add objects (of generic type E) to this structure.

Each node in the list is an instance of a (nested) `Node` class (see attached code). Each instance of `Node` points to two other `Node` instances: **the node immediately to its right (next) and the node before it by skipping one (prevSkip)!** *(see examples below for prevSkip illustration)*

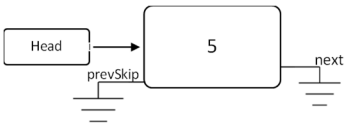
A single node in this structure will look like this:

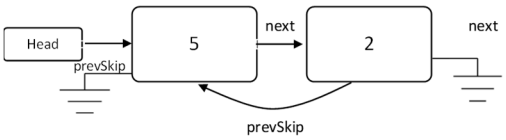


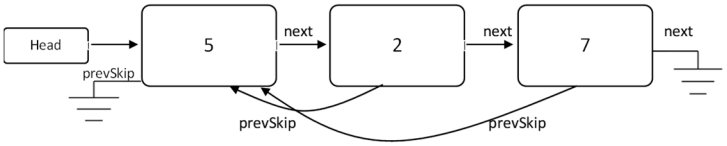
For simplicity, the node will be drawn as:

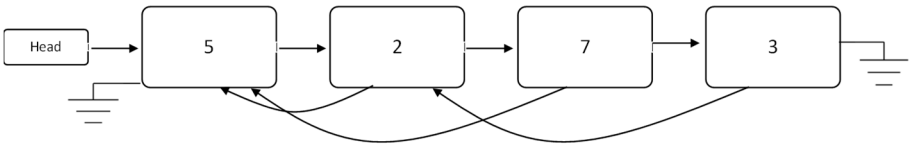
Once the node point to a null, the typical symbol used in lectures will be used 

This class has a main function called **add**. Once the **add function is used repeatedly for the following four integer objects: 5, 2, 7, 3 starting from an empty list,** the result after each add is:





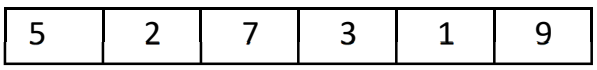




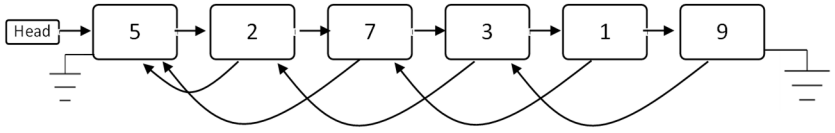
Moreover, your code should be able to transform a given one-dimensional array of objects (of a generic type `E`) into this structure. This can be done at the construction time (using a specific constructor). Example:

`Integer[6]`

As an array:



As a doubly-like linked structure:



Question 1 is decomposed into five sub-questions: Questions 1.1 to 1.5. **The implementation of all these questions will all be done in `LinkedDoublySkip.java` (The only file to be uploaded back).**

Questions:

1.1) public void add(E object) { …. //8 marks
This is a public function that adds one extra node to the structure.

1.2) public LinkedDoublySkip() { …. //2 marks
This is the default constructor.

1.3) public LinkedDoublySkip(E[] array) { …. //5 marks
This constructor receives an array and creates a list structure from the array content. Hint: you can call other methods from the same class (e.g. add(…)).

1.4) public E getElementAt(int index) { …. //6 marks
This function returns the element at position index, assuming the first node in the list is at **position 0**

1.5) private class LinkedDoublyIterator implements Iterator<E> {… //9 marks
Finish the constructor ***LinkedDoublyIterator*** and methods ***hasNext*** and ***next*** and add ***any necessary private variables*** (if needed!) **to iterate the structure as shown in the sample output (once you reach the last node, follow prevSkip to the first node).**

Sample test class:

(See Q1Test.java file)

Sample output:

```
List 1 with toString            : 1, 2, 3, 4
List 2 with toString            : 1, 2, 3, 4, 5
List 1 with getElementAt        : 1, 2, 3, 4
List 2 with getElementAt        : 1, 2, 3, 4, 5
List 1 with iIterator           : 1, 2, 3, 4, 2, 1
List 2 with iIterator           : 1, 2, 3, 4, 5, 3, 1

-- second set of test cases: using the second constructor --
List 1 with toString            : 5, 2, 7, 3, 1, 9
List 2 with toString            : 5, 2, 7, 3, 1, 9, 10
List 1 with getElementAt        : 5, 2, 7, 3, 1, 9
List 2 with getElementAt        : 5, 2, 7, 3, 1, 9, 10
List 1 with iIterator           : 5, 2, 7, 3, 1, 9, 3, 2, 5
List 2 with iIterator           : 5, 2, 7, 3, 1, 9, 10, 1, 7, 5
```

Important Restrictions for Question 1

- You **cannot** define any new variable or method in the classes **`LinkedDoublySkip`** or **`Node`**
- You **may** define new private variables in the inner class **`LinkedDoublyIterator`**.
- **Do not add any exception handling beyond what is provided to you in the shell implementation.**
- You **cannot** import anything at all.
- You **cannot** use any other classes than the ones we provide.
- **The file LinkedDoublySkip.java should contain all your answers to all five questions.**

Files

- **`LinkedDoublySkip.java`**
  - Add your name and student number at the top of the file.
  - You need to update this file and **UPLOAD it back as your solution for this question**.
- **`Iterator.java`**
  - Do not modify and do not upload this file with your solution)
- **`Q1Test.java`**
  - Do not modify and do not upload this file with your solution)

Question 2 (10 marks)

For this question, you will write a static method that can receive an array of Queues and returns one Queue that is the result of the merge operation of these queues. The merge operation is illustrated in the following examples:

Example 1: Array of three queues

Q[0] front -> 1, 2, 3, 5 <- rear
Q[1] front -> 6, 1, 8, 9, 15 <- rear
Q[2] front -> 4, 2, 1 <- rear
Q result front ->1, 6, 4, 2, 1, 2, 3, 8, 1, 5, 9, 15 <- rear

Example 2: Array of three queues

Q[0] front -> 1, 3 <- rear
Q[1] front -> 6, 1, 8, 9, 15 <- rear
Q result front ->1, 6, 3, 1, 8, 9, 15 <- rear

Queue interface and LinkedQueue classes are provided, and you are not supposed to change/add anything inside them. In addition, a test class is provided (Q2Test). Your work should be done in the class **QueueMerger**. You should implement the following method:

public static Queue merge(Queue q[ ]){ ….}

Sample test class:

```
(See Q2Test.java file)
```

Sample output:

```
---- First test: Integer Queue Arrays -----
Original Integer Queue[0]: Front -> [0, 1] <- Rear
Original Integer Queue[1]: Front -> [1, 2, 2] <- Rear
Original Integer Queue[2]: Front -> [2, 3, 3, 3, 4] <- Rear
Merged Integer Queue: Front -> [0, 1, 2, 1, 2, 3, 2, 3, 3, 4] <- Rear

---- Second test: Integer Queue Arrays -----
Original Integer Queue[0]: Front -> [] <- Rear
Original Integer Queue[1]: Front -> [] <- Rear
Original Integer Queue[2]: Front -> [2, 3, 3, 3, 4] <- Rear
Merged Integer Queue: Front -> [2, 3, 3, 3, 4] <- Rear

---- Third test: String Queue Array -----
Original String Queue[0]: Front -> [a, a, b, b, b, d, e, e] <- Rear
Merged String Queue: Front -> [a, a, b, b, b, d, e, e] <- Rear
```

Files

- **QueueMerger.java**
  - Add your name and student number at the top of the file.
  - You need to update this file and **UPLOAD it back as your solution for this question**.
- **LinkedQueue.java**
  - Do not modify and do not upload this file with your solution)
- **Queue.java**
  - Do not modify and do not upload this file with your solution)
- **Q2Test.java**
  - Do not modify and do not upload this file with your solution)