Practical-06 Part I: Data Structure

Due 20 Oct by 22:00 **Points** 100 **Available** after 4 Oct at 0:00

You should use JAVA to complete this practical.

Submission

Note: all class work must be submitted to your SVN repository. For this practice, you shall use:

```
https://version-control.adelaide.edu.au/svn/<YOUR-ID>/2019/s2/fcs/week-09/practical-06
```

Part 1 - Data Structure

Question 1: Set up a new project in your IDE with a Main class. Your project name should be MyDataStructure. All your files for this part should be saved under **practical-06/MyDataStructure/**.

Question 2: Define a **Node** class to represent the elements in a **Stack** or **Queue**. The **Node** class should have a **data** field and a **next** field that would reference another **Node**. In the constructor, **next** should be initialised as NULL.

```
public class Node
Attributes:
private String data; // the data saved in this node
private Node next; // the reference to another node

Methods:
// Print the data saved in the node.
public void printNode();

// You should also implement the constructors, mutators and accessors.
...

Example output for printNode():
Node data: "data saved in this node"
```

Question 3: Define a **MyStack** class. The **Stack** is a 'first in, last out' data structure. You need to implement a stack by using the **Node** class you implemented in Question 2. You need to implement a **push()** method for entering elements, a **pop()** method for retrieving elements from a stack and an **isEmpty()** method returning true if nothing is on the top of the stack.

```
public class MyStack
Attributes:
private Node top; // The reference to the top node in this stack
```

```
Methods:
// You should initialise an empty stack in the constructor.
public MyStack();

// Push a node into stack
public void push(Node node);

// Get and remove the top node from this stack.
// Return Null and print "Stack is empty" when the stack is empty.
public Node pop();

// Get the top node in this stack.
// Return Null and print "Stack is empty" when the stack is empty.
public Node peek();

// Return TRUE when the stack is empty, otherwise, return FALSE.
public boolean isEmpty();
```

Question 4: In your **Main** class, build a **bracketsMatching()** method.

```
public static boolean bracketsMatching(String input);
```

The bracketsMatching method takes a String of lower case letters as input. Your bracketsMatching method should return TRUE when the brackets/parenthesis are matching, and return FALSE if not. For example,

```
Input:  ({})
Output: true

Input:  ({)
Output: false

Input:  (1+2) * {(2+3)*(3+4)}
Output: true

Input:(1+2) * {{2+3)*(3+4}}
Output: false
```

Hint: you may use **MyStack** to help you solve this question.

Question 5: Define a **MyQueue** class. Queue is a 'first in, first out' data structure. You need to implement a queue by the **Node** class you implemented in Question 2. You need to implement a queue by creating **enqueue()** method for entering elements and **dequeue()** method for retrieving elements from a queue. The data type stored in your queue should be String.

```
public class MyQueue
Attributes:
private Node front, rear; // The reference to the first and last node in this queue

Methods:
// You should initialise the an empty queue in the constructor.
public MyQueue();

// Insert one node at the end of the queue.
```

```
public void enqueue(Node node);

// Get and remove the front node from the queue.
// Return Null and print "Queue is empty" when the queue is empty
public Node dequeue();

// Return TRUE when the queue is empty, otherwise, return FALSE.
public boolean isEmpty();

// Print out the data saved in nodes from the first to the last.
// Return and print "Queue is empty" when the queue is empty
// You should call printNode() to print each node in this method.
public void displayQueue();
```

Question 6: In your **Main** class, build a reverseQueue method.

```
public static MyQueue reverseQueue(MyQueue queue);
```

The reverseQueue method takes a Myqueue object as input, and return a Myqueue object which saves a reversed queue. You need to use enqueue, dequeue to solve this problem. For example,

```
Input:  1 4 7 9 2
Output: 2 9 7 4 1

Input:  3 3 4 1 2
Output: 2 1 4 3 3
```

Hint: you may use **MyQueue** to help you solve this question. You can print out the nodes saved in the queue by using displayQueue() method.

Part 2 - Emergency Service System

You are a software developer in an Emergency Service Center. Your team is building a system to help emergency call takers to organise ambulances for patients. Your team has designed and implemented part of the system.

In this system, it will record the **Name**, **Phone Number**, **Triage Level** (a number indicates how serious the patient's condition, from 1 to 5) and **Location** for each patient. When one patient call Emergency Service Center, this system will assign a unique **id** to this patient.

To make sure patients with higher triage level can be sent to the emergency department first, the waiting list will always give patients with higher triage level a higher priority. For patients has the same triage level, the patient who comes earlier should be assigned an ambulance first.

You can download the source code for this system at here: EmergencyServiceSystem.zip

Question 7: Download the code and save the folder under practical-06.

Your task is to help the team to build a waiting list so that when the call takers get an emergency call, they can record the patient's details and save it in the waiting list. They can also pop the patient out when there is an ambulance available.

All your files for this part should be saved under **practical-06/EmergencyServiceSystem/**

Question 8: After doing some research, you found that there are some features to implement in this task:

- Pop out the first patient from the list. (Implement popPatient() in WaitingList class)
- Add a patient into the waiting list, remember you need to take the triage level into consideration. (Implement addToList() in WaitingList class)
- Print out the patient's information (Id, Name, Triage level) in the waiting list. (Implement printList() in WaitingList class and printNode() in Node class)

Question 9: Your team found that sometimes some patients will call the service center to check their position in the waiting list, and some patients may need an ambulance immediately. You are asked to implement these new features:

- In the Main menu, add another option "**3. Assign Ambulance For Patient By Id**". In this option, the system can pop out one patient by his/her id. Implement this option. The example output after selecting this option should be (replace <id> by real id):

```
Success! An ambulance as assigned for patient <id>.
```

- In the Main menu, add another option "**4. Check Position By Id**". In this option, the system can check how many patients are waiting before this patient given the patient's Id. Implement this option. The example output after selecting this option should be (replace <number> and <id> by real number and id):

```
There are <number> patients before patient <id>.
```

Note: In this question, we will not provide a detailed signature specification, you need to design the solution by yourself.

Question 10: Before you deliver this project, the test group found some bugs in this system. Your team asked you to fix it.

- When assigning an ambulance for a patient, the system will crash if the waiting list is empty
  - The system should print out an error message and go back the main menu if someone wants to assign an ambulance when the waiting list is empty. The example output for the error message should be:

```
Error! The waiting list is empty.
```

- The system will crash when the input of triage level or id is not a number.
  - Ask the user to enter the triage level or id again if the input is invalid.
- The system should only accept (1-5) as legal input for triage level.
  - Ask the user to enter the triage level again if the input is invalid.

```
```