

Lesson – 9 - Lab Homework – Stack and Queue

Problem 1. [User Defined array-based Circular Queue] For this problem, you will implement a queue of `ints`, using an array in the background.

When adding an element to the queue, you add the element to the position pointed to by the variable `rear`. When removing an element from the queue, you remove the element in the position pointed to by the variable `front`.

Implement all of the methods declared below so that your class behaves as a queue using **circular array** to utilize the memory efficiently. The methods to be implemented are: `isEmpty`, `size`, `enqueue`, `dequeue`, `peek`, `resize` and `toString`. A start-up class `ArrayQueueImpl`, listing all of these methods, has been provided for you. You will need to implement the methods in that class. Write a `Main` class with an implemented `main` method to test your own code.

Your code must meet the following requirements:

1. Make sure before inserting you have room and if needed do the resizing.
2. `dequeue` (delete/remove) or `peek` should not cause null pointer exception.
3. You may not use Java library collection classes to create your queue.
4. There must not be any compilation errors or runtime errors in the solution that you submit.

```
public class ArrayQueueImpl {

    private final int DEFAULT_CAPACITY = 10;
    private int size;
    private Integer[] arr;
    private int front;
    private int rear;

    // Add Constructors, Default and Parameterized to initialize instance fields
    public Integer peek() {
        return null;
        //implement
    }
    public void enqueue(int obj){
        //implement
    }

    public Integer dequeue() {
```

```

        return null;
    }

    public boolean isEmpty(){
        //implement
        return false;
    }

    public int size(){
        //implement
        return 0;
    }
    private void resize(){
        //implement
    }
    public String toString(){
        /* Return data in this format, each element separated by comma with in [ ] eg:
[10, 20, 30, 40, 50, 60 ]*/
        return null;
    }
}

```

Problem 2 [User Defined Stack using Linked list]

Take the demo code ArrayStack.java. In that file Stack is implemented using Array. Do the same behaviors by implementing stack using Linked List. [Either using singly linked list / Doubly linked list] - Only user Implementation

Problem 3 [Stack API] - Browser History Navigation

Implement a browser history navigation system using a Stack to allow users to go back and forward in their browsing history.

Class Design:

- Create a class that manages browser history using two stacks and an attribute that will store the current url name. Can have a constructor to start with the url.

Method Requirements:

- `visit(String url)`: This method should handle visiting a new URL. It should update the current URL and manage the history stacks appropriately.
- `back()`: This method should allow the user to navigate back to the previous URL in the history.

- `forward()`: This method should allow the user to navigate forward to the next URL in the forward history.

Use the below Test class code for the better understanding of the requirements.

```
package lesson9.prob3;

public class TestBrowserHistory {
    public static void main(String[] args) {
        BrowserHistory browser = new
        BrowserHistory("home.html");// Start with home.html
        browser.visit("page1.html");
        browser.visit("page2.html");
        browser.back();
        browser.back();
        browser.back();
        browser.forward();
        browser.forward();
        browser.forward();
        browser.visit("page3.html");
    }
}
```

Sample Output

```
Visited: page1.html
Visited: page2.html
Back to: page1.html
Back to: home.html
No history to go back to.
Forward to: page1.html
Forward to: page2.html
No forward history.
Visited: page3.html
```

Problem 4: Queue API - Customer Service Ticketing System

Class Design:

Task 1: Create a `Ticket` class that represents a customer service ticket with attributes such as `id` and `description` with the constructor, necessary getters and setters.

Task 2: Create a `TicketingSystem` class that manages the ticket queue.

- This class need to include the attribute `Queue<Ticket>` and a `nextId` for the ticket. You can auto generate the ticket id or random int id. It's up to you.
- A default constructor to initialize the fields.

Method Requirements:

- `addTicket(String description)`: This method should add a new ticket to the queue with a unique ID and the provided description.
- `processTicket()`: This method should process (remove and return) the ticket at the front of the queue.
- `viewNextTicket()`: This method should allow viewing the ticket at the front of the queue without removing it.
- Implement any additional methods you find necessary to manage the ticket queue effectively.

Task 3: A demonstration of the system in action through a main method by creating a Test class.
