Lesson – 9 - Lab Homework – Stack and Queue

Problem 1. **[User Defined array-based Queue]** For this problem, you will implement a queue of `ints`, using an array in the background. To start, initialize an array and two pointers (`front` and `rear`) in your queue class as follows:

```
private int[] arr = new int[10];
private int front = -1;
private int rear = 0;
```

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. The methods to be implemented are: `isEmpty, size, enqueue, dequeue`, and `peek`. 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. The `enqueue` method should never cause an exception to be thrown. In particular, your queue must support unlimited `enqueue(insertion/add)` operations. This means that you will need to incorporate a procedure for resizing the background array periodically.
2. The only situation in which `dequeue(delete/remove)` or `peek` should cause an exception to be thrown is if either of these operations is called when the queue is empty. In that case return -1, also display the message in console as "Queue is Empty" without throwing an 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 int[] arr = new int[10];
        private int front = -1;
        private int rear = 0;


        public int peek() {
                return -1;
                //implement
        }
```

```
	public void enqueue(int obj){
		//implement
	}

	public int dequeue() {
		return -1;
		//implement
	}

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

	public int size(){
		//implement
		return 0;
	}
	private void resize(){
		//implement
	}
}
```

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"); // Current URL Page
page1.html
            browser.visit("page2.html"); // Current URL Page
page2.html
            browser.back(); // Back to: page1.html
            browser.back(); // Back to: home.html
            browser.back(); // Forward to: page1.html
            browser.forward(); // Forward to: page2.html
            browser.forward(); // Forward to: page2.html
            browser.forward();//No forward history.
            browser.visit("page3.html"); // Current URL page
        }
    }
```

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.