Data Structure Lab8 : Queue 2022-2023

1. Create Queue Interface

Define a `Queue` interface with basic methods:

```
java
Copy code
public interface Queue<E> {
    boolean isEmpty();
    int size();
    E first();
    void enqueue(E e);
    E dequeue();
}
```

---

2. Create Queue Using Array

Implement the `Queue` interface using an array, commonly referred to as a circular array.

---

3. Create Queue Using Linked Lists

Implement the `Queue` interface using a linked list. This approach involves a `Node` class for queue elements.

---

4. Implement Basic Methods of Queue

Implement methods: `isEmpty()`, `size()`, `first()`, `enqueue(E e)`, and `dequeue()` for both array-based and linked list implementations.

Data Structure Lab8 : Queue 2022-2023

Homework

1. Efficient `rotate()` Method for `ArrayQueue`

Rotate the queue by moving the front element to the back without separate `enqueue(dequeue())` calls.

```
java
Copy code
public class ArrayQueue<E> implements Queue<E>, Cloneable {
    private E[] data;
    private int front = 0;
    private int size = 0;

    public ArrayQueue(int capacity) {
        data = (E[]) new Object[capacity];
    }

    @Override
    public boolean isEmpty() {
        return size == 0;
    }

    @Override
    public int size() {
        return size;
    }

    @Override
    public E first() {
        if (isEmpty()) return null;
        return data[front];
    }

    @Override
    public void enqueue(E e) {
        if (size == data.length) throw new IllegalStateException("Queue is full");
        int avail = (front + size) % data.length;
        data[avail] = e;
        size++;
    }

    @Override
    public E dequeue() {
        if (isEmpty()) return null;
        E answer = data[front];
        data[front] = null;
        front = (front + 1) % data.length;
        size--;
        return answer;
    }

    public void rotate() {
        if (!isEmpty()) {
            int avail = (front + size) % data.length;
            data[avail] = data[front];
            data[front] = null;
            front = (front + 1) % data.length;
        }}}
```

Data Structure Lab8 : Queue 2022-2023

2. Implement `clone()` Method for `ArrayQueue`

Add the cloning capability to the `ArrayQueue` class.

```
java
Copy code
@Override
public ArrayQueue<E> clone() {
    try {
        ArrayQueue<E> cloned = (ArrayQueue<E>) super.clone();
        cloned.data = data.clone();
        return cloned;
    } catch (CloneNotSupportedException e) {
        throw new AssertionError(); // Shouldn't happen
    }
}
```

Data Structure Lab8 : Queue 2022-2023

3. Implement `concatenate(LinkedQueue Q2)`

Concatenate two linked queues in O(1)O(1)O(1) time.

```
java
Copy code
public class LinkedQueue<E> implements Queue<E> {
    private static class Node<E> {
        E element;
        Node<E> next;

        Node(E e, Node<E> next) {
            this.element = e;
            this.next = next;
        }
    }

    private Node<E> head = null;
    private Node<E> tail = null;
    private int size = 0;

    @Override
    public boolean isEmpty() {
        return size == 0;
    }

    @Override
    public int size() {
        return size;
    }

    @Override
    public E first() {
        return isEmpty() ? null : head.element;
    }

    @Override
    public void enqueue(E e) {
        Node<E> newNode = new Node<>(e, null);
        if (isEmpty()) {
            head = tail = newNode;
        } else {
            tail.next = newNode;
            tail = newNode;
        }
        size++;
    }

    @Override
    public E dequeue() {
        if (isEmpty()) return null;
        E answer = head.element;
        head = head.next;
        size--;
        if (isEmpty()) tail = null;
        return answer;
    }

    public void concatenate(LinkedQueue<E> Q2) {
        if (Q2.isEmpty()) return;
        if (this.isEmpty()) {
            head = Q2.head;
            tail = Q2.tail;
        } else {
            this.tail.next = Q2.head;
            this.tail = Q2.tail;
        }
        this.size += Q2.size;
        Q2.head = Q2.tail = null;
        Q2.size = 0;
    }
}
```

Data Structure Lab8 : Queue 2022-2023

4. Josephus Problem Using a Queue

Solve the Josephus problem using a circular queue.

```
java
Copy code
public class JosephusProblem {
    public static <E> E solve(Queue<E> queue, int k) {
        while (queue.size() > 1) {
            for (int i = 0; i < k - 1; i++) {
                queue.enqueue(queue.dequeue());
            }
            queue.dequeue(); // Remove the k-th person
        }
        return queue.dequeue(); // Last person remaining
    }
}
```

---

5. Round Robin Scheduling Using a Queue

Simulate Round Robin Scheduling with a queue.

```
java
Copy code
public class RoundRobinScheduling {
    public static void simulate(Queue<String> queue, int quantum) {
        while (!queue.isEmpty()) {
            String process = queue.dequeue();
            System.out.println("Executing: " + process + " for " + quantum +
" units");
            // Simulate process execution (assuming no actual timeslice logic
here)
            if (Math.random() > 0.5) { // Randomly decide if process needs
more time
                queue.enqueue(process);
                System.out.println(process + " re-added to the queue.");
            } else {
                System.out.println(process + " completed.");
            }
        }
    }
}
```