Data Structure Lab8 : Queue 2022-2023

Topics

1. Create Queue Interface
2. Create Queue Using Array
3. Create Queue Using Linked Lists
4. Implement Basic Methods of Queue
    - isEmpty()
    - size()
    - first()
    - enqueue(E e)
    - dequeue()

Homework

1. Augment the ArrayQueue implementation with a new rotate( ) method having semantics identical to the combination, enqueue(dequeue( )). But, your implementation should be more efficient than making two separate calls (for example, because there is no need to modify the size).

```
public class ArrayQueue<E> {
    private E[] data;
    private int front = 0;
    private int size = 0;
    private static final int CAPACITY = 10;

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

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

Data Structure Lab8 : Queue 2022-2023

```
    public int size() {
        return size;
    }

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

    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++;
    }

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

    // New rotate method
    public void rotate() {
        if (size > 0) {
            int avail = (front + size) % data.length; // Find next available spot
            data[avail] = data[front]; // Move front element to the end
            data[front] = null; // Clear the old front
```

```
            front = (front + 1) % data.length; // Update front pointer
        }
    }
}
```

2. Implement the clone( ) method for the ArrayQueue class.

```
@Override
public ArrayQueue<E> clone() {
    try {
        ArrayQueue<E> cloned = (ArrayQueue<E>) super.clone();
        cloned.data = data.clone(); // Deep copy of the array
        return cloned;
    } catch (CloneNotSupportedException e) {
        throw new AssertionError(); // Should never happen
    }
}
```

3. Implement a method with signature concatenate(LinkedQueue Q2) for the LinkedQueue class that takes all elements of Q2 and appends them to the end of the original queue. The operation should run in O(1) time and should result in Q2 being an empty queue.

```
public class LinkedQueue<E> {
    private static class Node<E> {
        E element;
        Node<E> next;

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

Data Structure Lab8 : Queue 2022-2023

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

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

public int size() {
    return size;
}

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

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; // Nothing to concatenate
    if (this.isEmpty()) {
```

```
            this.head = Q2.head;
            this.tail = Q2.tail;
        } else {
            this.tail.next = Q2.head;
            this.tail = Q2.tail;
        }
        this.size += Q2.size;
        Q2.head = Q2.tail = null; // Empty Q2
        Q2.size = 0;
    }
}
```

4. Use a queue to solve the Josephus Problem.

```
public static int josephusProblem(int n, int k) {
    Queue<Integer> queue = new LinkedList<>();
    for (int i = 1; i <= n; i++) {
        queue.add(i);
    }

    while (queue.size() > 1) {
        for (int i = 0; i < k - 1; i++) {
            queue.add(queue.poll()); // Rotate the queue
        }
        queue.poll(); // Remove the k-th person
    }

    return queue.peek(); // Last person remaining
```

```
}
```

5. Use a queue to simulate Round Robin Scheduling.

```
public static void roundRobinScheduling(String[] tasks, int timeSlice) {
    Queue<String> queue = new LinkedList<>();
    for (String task : tasks) {
        queue.add(task);
    }

    while (!queue.isEmpty()) {
        String currentTask = queue.poll();
        System.out.println("Processing: " + currentTask);
        // Simulate task processing
        // Re-add task to the queue if it needs more time (dummy example)
        if (Math.random() > 0.5) { // Randomly decide if the task is done
            queue.add(currentTask);
        }
    }
}
```