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 void rotate() {
    if (isEmpty()) {
    return;
        }
    E firstElement = dequeue();
        enqueue(firstElement);
        }
```

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

```
    public ArrayQueue<E> clone() {
        try {
        ArrayQueue<E> clonedQueue = (ArrayQueue<E>) super.clone(); // إنشاء نسخة من ي
            clonedQueue.elements = Arrays.copyOf(elements, elements.length); // نسخ
            return clonedQueue;
            } catch (CloneNotSupportedException e) {
        throw new AssertionError(e); // لن يحدث هنا طبقًا لواجهة Cloneable
        }     }
```

Data Structure Lab8 : Queue 2022-2023

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 void concatenate(LinkedQueue Q2) {
    if (Q2.isEmpty()) {
    return;
        }

    if (isEmpty()) {
    front = Q2.front;
    } else {
    rear.next = Q2.front;
    }

    rear = Q2.rear;

    size += Q2.size; ي

        Q2.front = null;
    Q2.rear = null;
        Q2.size = 0;
    }
```

4. Use a queue to solve the Josephus Problem.

```
public class Josephus {
  private static class Node {
    int name;
    Node next;
    public Node(int name) {
      this.name = name;
      this.next = null;  }    }
  private int n;
  private int k;
  private Node head;
  public Josephus(int n, int k) {
    this.n = n;
    this.k = k;
    this.head = null; }
 private void createList() {
    for (int i = 0; i < n; i++) {
      Node newNode = new Node(i + 1);
      if (head == null) {
```

```
                head = newNode;
                tail = newNode;
            } else {
                tail.next = newNode;
                tail = newNode;
            }
            tail.next = head;
        }
    }

    private int solve() {
        Node current = head;
        while (current.next != current) {
            for (int i = 0; i < k - 1; i++) {
                current = current.next;
            }
            Node temp = current.next;
            current.next = temp.next;
            temp = null;
        }
        return current.name;
    }

    public static void main(String[] args) {
        int n = 5;
        int k = 2;

        Josephus josephus = new Josephus(n, k);
        josephus.createList();
        int winner = josephus.solve();

        System.out.println("The winner is: " + winner);
    }
}
```

5.

Data Structure Lab8 : Queue 2022-2023

5Use a queue to simulate Round Robin Scheduling.

```java
import java.util.LinkedList;

public class RoundRobin {

    private static class Process {
        int id;
        int burstTime;
        int remainingTime;

        public Process(int id, int burstTime) {
            this.id = id;
            this.burstTime = burstTime;
            this.remainingTime = burstTime;
        }
    }

    public static void main(String[] args) {
        int n = 3;
        int quantum = 2;

        LinkedList<Process> queue = new LinkedList<>();
        for (int i = 0; i < n; i++) {
            queue.add(new Process(i + 1, (int) (Math.random() * 10)));
        }

        while (!queue.isEmpty()) {
            Process process = queue.removeFirst();
            System.out.println("Executing process " + process.id);
            process.remainingTime -= quantum;
            if (process.remainingTime > 0) {
                queue.addLast(process);
            }
        }

        System.out.println("All processes have been executed.");
    }
}
```