Queue

1. Circular Queue using Array

```java
public class CircularQueuesUsingArray {
    static class Queue {
        static int arr[];
        static int size;
        static int rear;
        static int front;
        Queue (int n) {
            arr = new int[n];
            size = n;
            rear = -1;
            front = -1;
        }
        public static boolean isEmpty() {
            return rear == -1 && front == -1;
        }
        public static boolean isFull() {
            return (rear + 1) % size == front;
        }
        public static void add (int data) {
            if (isFull()) {
                System.out.println("Queue is Full");
                return;
            }
            if (front == -1) {
                front = 0;
            }
            rear = (rear+1) % size;
            arr[rear] = data;
        }
        public static int remove() { // O(1)
            if (isEmpty()) {
                System.out.println("Empty Queue");
                return -1;
            }
            int result = arr[front];
            if (rear == front) {
                rear = front = -1;
            } else {
                front = (front+1) % size;
            }
            return result;
        }
        public static int peek() {
            if (isEmpty()) {
                System.out.println("Queue is Empty");
                return -1;
            }
            return arr[front];
        }
    }
    public static void main(String[] args) {
        Queue q = new Queue(5);
        q.add(1);
```

```
        q.add(2);
        q.add(3);
        while (!q.isEmpty()) {
            System.out.println(q.peek());
            q.remove();
        }
    }
}
```

2. Connect N Ropes with Min Cost

```
import java.util.*;

public class ConnectNropesWithMinCost {
    public static int minCost (int arr[], int n) {
        PriorityQueue <Integer> pq = new PriorityQueue<>();
        for (int i = 0; i < n; i++) {
            pq.add(arr[i]);
        }
        int res = 0;
        while (pq.size() > 1) {
            int first = pq.poll();
            int second = pq.poll();
            res += first + second;
            pq.add(first + second);
        }
        return res;
    }
    public static void main(String[] args) {
        int len[] = {4, 3, 2, 6};
        int size = len.length;
        System.out.println("Total Cost for Connecting ropes is: " + minCost(len, size));
    }
}
```

3. Deque Example

```
import java.util.Deque;
import java.util.LinkedList;

public class DequeExample {
    // DEQUE - Double Ended Queue - allows insertion and deletion of elements from both ends
    public static void main(String[] args) {
        Deque <Integer> deque = new LinkedList<>();
        deque.addFirst(1);
        deque.addFirst(2);
        deque.addLast(3);
        deque.addLast(4);
        System.out.println(deque);
        deque.removeLast();
        System.out.println(deque);
        System.out.println("First Element: " + deque.getFirst());
        System.out.println("Last Element: " + deque.getLast());
    }
}
```

4. First Non-Repeating Letter

```java
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

public class FirstNon_repeatingLetter {
    public static void printNonRepeating(String str) {
        int freq[] = new int[26];
        Queue <Character> q = new LinkedList<>();
        for(int i = 0; i < str.length(); i++) {
            char ch = str.charAt(i);
            q.add(ch);
            freq[ch - 'a']++;
            while (!q.isEmpty() && freq[q.peek() - 'a'] > 1) {
                q.remove();
            }
            if (q.isEmpty()) {
                System.out.println(-1 + " ");
            } else {
                System.out.println(q.peek());
            }
        }
        System.out.println();
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a String: ");
        String str = sc.next();
        System.out.println("Output: ");
        printNonRepeating(str);
    }
}
```

5. Generate Binary Number

```java
import java.util.*;
public class GenerateBinaryNumber {
    public static void generatePrintBinary(int n) {
        Queue <String> q = new LinkedList <String> ();
        q.add("1");
        while (n --> 0) {
            String s1 = q.peek();
            q.remove();
            System.out.println(s1);
            String s2 = s1;
            q.add(s1 + "0");
            q.add(s2 + "1");
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner (System.in);
        System.out.print("Enter value of n: ");
        int n = sc.nextInt();
        generatePrintBinary(n);
    }
}
```

6. Implementation by Array

```java
public class implementingbyArray {
    static class Queue {
        static int arr[];
        static int size;
        static int rear;
        Queue (int n) {
            arr = new int[n];
            size = n;
            rear = -1;
        }
        // Checking Queue is Empty or not
        public static boolean isEmpty() {
            return rear == -1;
        }
        // Enque
        public static void add (int data) {
            if (rear == size-1) {
                System.out.println("Queue is Full");
                return;
            }
            rear = rear + 1;
            arr[rear] = data;
        }
        // Dequeue - O(n)
        public static int remove() {
            if (isEmpty()) {
                System.out.println("Empty Queue");
                return -1;
            }
            int front = arr[0];
            for (int i = 0; i < rear; i++) {
                arr[i] = arr[i+1];
            }
            rear = rear - 1;
            return front;
        }
        // Peek
        public static int peek() {
            if (isEmpty()) {
                System.out.println("Queue is Empty");
                return -1;
            }
            return arr[0];
        }
    }
    // We can make user-input Queue by using switch case. LENGTHY😰😰
    public static void main(String[] args) {
        Queue q = new Queue(5);
        q.add(1);
        q.add(2);
        q.add(3);
        while (!q.isEmpty()) {
            System.out.println(q.peek());
            q.remove();
        }
```

```
    }
}
```

7. Implement Queue using Deque

```
import java.util.Deque;
import java.util.LinkedList;

public class ImplementQueueUsingDeque {
    static class Queue {
        Deque <Integer> deque = new LinkedList<>();
        public void add (int data) {
            deque.addLast(data);
        }
        public int remove() {
            return deque.removeFirst();
        }
        public int peek() {
            return deque.getFirst();
        }
    }
    public static void main(String[] args) {
        Queue q = new Queue();
        q.add(1);
        q.add(2);
        q.add(3);
        System.out.println("Peek: " + q.peek());
        System.out.println(q.remove());
        System.out.println(q.remove());
        System.out.println(q.remove());
    }
}
```

8. Implement Stack using Deque

```
import java.util.Deque;
import java.util.LinkedList;

public class ImplementStackUsingDeque {
    static class Stack {
        Deque <Integer> deque = new LinkedList<>();
        public void push(int data) {
            deque.addLast(data);
        }
        public int pop() {
            return deque.removeLast();
        }
        public int peek() {
            return deque.getLast();
        }
    }
    public static void main(String[] args) {
        Stack s = new Stack();
        s.push(1);
        s.push(2);
        s.push(3);
```

```
        System.out.println("Peek: " + s.peek());
        System.out.println(s.pop());
        System.out.println(s.pop());
        System.out.println(s.pop());
    }
}
```

9. Interleave Two Even Length Halves

```
import java.util.LinkedList;
import java.util.Queue;

public class InterleaveTwoEvenLengthHalves {
    public static void interleave(Queue <Integer> q) {
        Queue <Integer> firstHalf = new LinkedList<>();
        int size = q.size();
        for(int i = 0; i < size/2; i++) {
            firstHalf.add(q.remove());
        }
        while (!firstHalf.isEmpty()) {
            q.add(firstHalf.remove());
            q.add(q.remove());
        }
    }
    public static void main(String[] args) {
        Queue <Integer> q = new LinkedList<>();
        q.add(1);
        q.add(2);
        q.add(3);
        q.add(4);
        q.add(5);
        q.add(6);
        q.add(7);
        q.add(8);
        q.add(9);
        q.add(10);
        interleave(q);
        while (!q.isEmpty()) {
            System.out.print(q.remove() + " ");
        }
    }
}
```

10. Job Sequencing Problem

```
import java.util.*;

public class JobSequencingProblem {
    static class Job {
        char job_id;
        int deadline;
        int profit;

        Job(char job_id, int deadline, int profit) {
            this.deadline = deadline;
            this.job_id = job_id;
```

```
        this.profit = profit;
    }
}

static void printJobSequencing(ArrayList<Job> arr) {
    int n = arr.size();

    // Sorting by deadlines in ascending order
    Collections.sort(arr, (a, b) -> a.deadline - b.deadline);

    // Priority queue to act as a max heap, based on profit (highest profit first)
    PriorityQueue<Job> maxHeap = new PriorityQueue<>((a, b) -> b.profit - a.profit);

    // List to store the result sequence of jobs
    List<Job> result = new ArrayList<>();

    // Iterate over jobs starting from the one with the largest deadline
    for (int i = n - 1; i > -1; i--) {
        int slot_available;

        // Calculate the number of available slots between two jobs
        if (i == 0) {
            slot_available = arr.get(i).deadline;
        } else {
            slot_available = arr.get(i).deadline - arr.get(i - 1).deadline;
        }

        // Add the job to the maxHeap
        maxHeap.add(arr.get(i));

        // While there are available slots and jobs in the heap, assign jobs
        while (slot_available > 0 && maxHeap.size() > 0) {
            Job job = maxHeap.remove();
            slot_available--;
            result.add(job);
        }
    }

    // Sort the result jobs by deadline for the output sequence
    Collections.sort(result, (a, b) -> a.deadline - b.deadline);

    // Print the result job sequence
    for (Job job : result) {
        System.out.print(job.job_id + " ");
    }
    System.out.println();
}

public static void main(String[] args) {
    ArrayList<Job> arr = new ArrayList<Job>();
    arr.add(new Job('a', 2, 100));
    arr.add(new Job('b', 1, 19));
    arr.add(new Job('c', 2, 27));
    arr.add(new Job('d', 1, 25));
    arr.add(new Job('e', 3, 15));

    System.out.println("Following is Maximum Profit Sequence of Jobs: ");
```

```
        printJobSequencing(arr);
    }
}
```

11. Max of all Subarray of Size K

```
import java.util.*;

public class MaxOfAllSubarrOfSizeK {
    public static void printMax(int arr[], int n, int k) {
        // Create a deque to store indices of array elements
        Deque<Integer> Qi = new LinkedList<Integer>();

        // Process the first k elements of the array
        int i;
        for (i = 0; i < k; ++i) {
            // Remove elements smaller than the current one, as they are useless
            while (!Qi.isEmpty() && arr[i] >= arr[Qi.peekLast()]) {
                Qi.removeLast();
            }
            // Add the current element at the rear of the deque
            Qi.addLast(i);
        }

        // Process the rest of the elements
        for (; i < n; ++i) {
            // The element at the front of the deque is the largest of the previous window
            System.out.print(arr[Qi.peek()] + " ");

            // Remove the elements which are out of this window
            while (!Qi.isEmpty() && Qi.peek() <= i - k) {
                Qi.removeFirst();
            }

            // Remove all elements smaller than the current element
            while (!Qi.isEmpty() && arr[i] >= arr[Qi.peekLast()]) {
                Qi.removeLast();
            }

            // Add the current element at the rear of the deque
            Qi.addLast(i);
        }

        // Print the maximum element of the last window
        System.out.print(arr[Qi.peek()]);
    }

    public static void main(String[] args) {
        int arr[] = {1, 2, 3, 1, 4, 5, 2, 3, 6};
        int k = 3;
        printMax(arr, arr.length, k);
    }
}
```

12. Queue JCF

```java
// import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;

public class QueueJCF {
    public static void main(String[] args) {
        Queue <Integer> q = new LinkedList<>();
                        // OR
        // Queue <Integer> al = new ArrayList<>();
        q.add(1);
        q.add(2);
        q.add(3);
        while (!q.isEmpty()) {
            System.out.println(q.peek());
            q.remove();
        }
    }
}
```

13. Queue Reversal

```java
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class QueueReversal {
    public static void reverse(Queue <Integer> q) {
        Stack <Integer> s = new Stack<>();
        while (!q.isEmpty()) {
            s.push(q.remove());
        }
        while (!s.isEmpty()) {
            q.add(s.pop());
        }
    }
    public static void main(String[] args) {
        Queue <Integer> q = new LinkedList<>();
        q.add(1);
        q.add(2);
        q.add(3);
        q.add(4);
        q.add(5);
        q.add(6);
        System.out.println("Original Queue: " + q);
        reverse(q);
        System.out.print("Reversed Queue: ");
        while (!q.isEmpty()) {
            System.out.print(q.remove() + " ");
        }
    }
}
```

14. Queue using Two Stacks

```java
import java.util.Stack;

public class QueueUsingTwoStack {
    static class Queue {
        static Stack <Integer> s1 = new Stack<>();
        static Stack <Integer> s2 = new Stack<>();
        public static boolean isEmpty() {
            return s1.isEmpty();
        }
        public static void add(int data) {
            while (!s1.isEmpty()) {
                s2.push(s1.pop());
            }
            s1.push(data);
            while (!s2.isEmpty()) {
                s1.push(s2.pop());
            }
        }
        public static int remove() {
            if (isEmpty()) {
                System.out.println("Queue is Empty");
                return -1;
            }
            return s1.pop();
        }
        public static int peek() {
            if (isEmpty()) {
                System.out.println("Queue is Empty");
                return -1;
            }
            return s1.peek();
        }
    }
    public static void main(String[] args) {
        Queue q = new Queue();
        q.add(1);
        q.add(2);
        q.add(3);
        while (!q.isEmpty()) {
            System.out.println(q.peek());
            q.remove();
        }
    }
}
```

15. Reversing First K Elements

```java
import java.util.*;

public class ReversingFirstKelements {
    static class Cell {
        int x, y;
        int dis;
        public Cell(int x, int y, int dis) {
            this.x = x;
```

```
            this.y = y;
            this.dis = dis;
        }
    }

    static boolean isInside(int x, int y, int N) {
        return x >= 1 && x <= N && y >= 1 && y <= N;
    }

    static int minStepToReachTarget(int knightPos[], int targetPos[], int N) {
        int dx[] = {-2, -1, 1, 2, -2, -1, 1, 2};
        int dy[] = {-1, -2, -2, -1, 1, 2, 2, 1};

        // Use a queue instead of a Vector
        Queue<Cell> queue = new LinkedList<>();
        queue.add(new Cell(knightPos[0], knightPos[1], 0));

        boolean visit[][] = new boolean[N + 1][N + 1];
        visit[knightPos[0]][knightPos[1]] = true;

        while (!queue.isEmpty()) {
            Cell t = queue.poll(); // Remove the first element

            // If the knight reaches the target
            if (t.x == targetPos[0] && t.y == targetPos[1]) {
                return t.dis;
            }

            // Explore all possible moves
            for (int i = 0; i < 8; i++) {
                int x = t.x + dx[i];
                int y = t.y + dy[i];

                if (isInside(x, y, N) && !visit[x][y]) {
                    visit[x][y] = true;
                    queue.add(new Cell(x, y, t.dis + 1));
                }
            }
        }

        return Integer.MAX_VALUE; // In case no solution is found
    }

    public static void main(String[] args) {
        int N = 30;
        int knightPos[] = {1, 1};
        int targetPos[] = {30, 30};
        System.out.println(minStepToReachTarget(knightPos, targetPos, N));
    }
}
```

16. Stack using two Queues

```
import java.util.LinkedList;
import java.util.Queue;

public class StackUsingTwoQueues {
    static class Stack {
        static Queue <Integer> q1 = new LinkedList<>();
        static Queue <Integer> q2 = new LinkedList<>();
        public static boolean isEmpty() {
            return q1.isEmpty() && q2.isEmpty();
        }
        public static int pop() {
            if (isEmpty()) {
                System.out.println("Empty Stack");
                return -1;
            }
            int top = -1;
            if (!q1.isEmpty()) {
                while (!q1.isEmpty()) {
                    top = q1.remove();
                    if (q1.isEmpty()) {
                        break;
                    }
                    q2.add(top);
                }
            } else {
                while (!q2.isEmpty()) {
                    top = q2.remove();
                    if (q2.isEmpty()) {
                        break;
                    }
                    q1.add(top);
                }
            }
            return top;
        }
        public static void push(int data) {
            if (!q1.isEmpty()) {
                q1.add(data);
            } else {
                q2.add(data);
            }
        }
        public static int peek() {
            if (isEmpty()) {
                System.out.println("Empty Stack");
                return -1;
            }
            int top = -1;
            if (!q1.isEmpty()) {
                while (!q1.isEmpty()) {
                    top = q1.remove();
                    q2.add(top);
                }
            } else {
                while (!q2.isEmpty()) {
```

```
                top = q2.remove();
                q1.add(top);
            }
        }
        return top;
    }
  }
  public static void main(String[] args) {
    Stack s = new Stack();
    s.push(1);
    s.push(2);
    s.push(3);
    while (!s.isEmpty()) {
        System.out.println(s.peek());
        s.pop();
    }
  }
}
```

17. Queue Using Linked List

```
import java.util.LinkedList;

public class usingLL {
  static class Node {
    int data;
    Node next;
    Node (int data) {
        this.data = data;
        this.next = null;
    }
  }
  static class Queue {
    static Node head = null;
    static Node tail = null;
    public static boolean isEmpty() {
        return head == null && tail == null;
    }
    // Adding
    public static void enque(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = tail = newNode;
            return;
        }
        tail.next = newNode;
        tail = newNode;
    }
    // Removing
    public static int remove() {
        if (isEmpty()) {
            System.out.println("Empty Queue");
            return -1;
        }
        int front = head.data;
        if (tail == head) {
```

```
                tail = head = null;
            } else {
                head = head.next;
            }
            return front;
        }
        // Front
        public static int peek() {
            if (isEmpty()) {
                System.out.println("Empty Queue");
                return -1;
            }
            return head.data;
        }
    }
    public static void main(String[] args) {
        LinkedList ll = new LinkedList<>();
        ll.add(1);
        ll.add(2);
        ll.add(3);
        while (!ll.isEmpty()) {
            System.out.println(ll.peek());
            ll.remove();
        }
    }
}
```