1) Implement a stack using an array in Java. Include the necessary methods such as push, pop, and isEmpty.

```
public class Stack {
    private int maxSize; // Maximum size of the stack
    private int top; // Index of the top element in the stack
    private int[] stackArray; // Array to store the stack elements

    // Constructor to initialize the stack
    public Stack(int size) {
        maxSize = size;
        top = -1; // Initially, the stack is empty
        stackArray = new int[maxSize];
    }

    // Method to push an element onto the stack
    public void push(int value) {
        if (top == maxSize - 1) {
            System.out.println("Stack is full. Cannot push element.");
        } else {
            stackArray[++top] = value;
            System.out.println(value + " pushed to the stack.");
        }
    }

    // Method to pop the top element from the stack
    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack is empty. Cannot pop element.");
            return -1;
        } else {
            int poppedValue = stackArray[top--];
            System.out.println(poppedValue + " popped from the stack.");
            return poppedValue;
        }
    }

    // Method to check if the stack is empty
    public boolean isEmpty() {
        return (top == -1);
    }
```

```
    public static void main(String[] args) {
        Stack stack = new Stack(5);

        stack.push(10);
        stack.push(20);
        stack.push(30);

        stack.pop();
        stack.pop();

        stack.push(40);
        stack.push(50);

        while (!stack.isEmpty()) {
            stack.pop();
        }
    }
}
```

2) Implement a queue using a linked list in Java. Include the necessary methods such as enqueue, dequeue, and isEmpty.

```
class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}

class Queue {
    private Node front; // Points to the front of the queue
    private Node rear; // Points to the rear of the queue

    // Constructor to initialize an empty queue
    public Queue() {
```

```
    front = null;
    rear = null;
}

// Method to check if the queue is empty
public boolean isEmpty() {
    return (front == null);
}

// Method to enqueue (insert) an element into the queue
public void enqueue(int data) {
    Node newNode = new Node(data);

    if (rear == null) {
        front = newNode;
        rear = newNode;
    } else {
        rear.next = newNode;
        rear = newNode;
    }

    System.out.println(data + " enqueued into the queue.");
}

// Method to dequeue (remove) an element from the queue
public int dequeue() {
    if (isEmpty()) {
        System.out.println("Queue is empty. Cannot dequeue element.");
        return -1;
    }

    int dequeuedValue = front.data;
    front = front.next;

    // If front becomes null after dequeuing, set rear to null as well
    if (front == null) {
        rear = null;
    }

    System.out.println(dequeuedValue + " dequeued from the queue.");
```

```
        return dequeuedValue;
    }

    public static void main(String[] args) {
        Queue queue = new Queue();

        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);

        queue.dequeue();
        queue.dequeue();

        queue.enqueue(40);
        queue.enqueue(50);

        while (!queue.isEmpty()) {
            queue.dequeue();
        }
    }
}
```