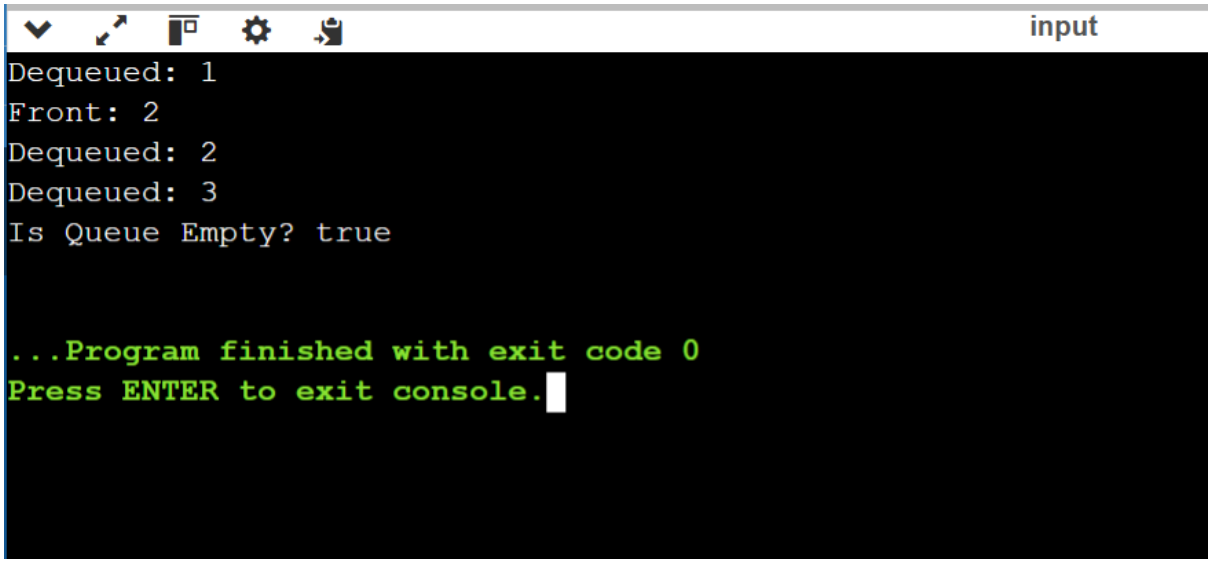
1. Implement queue using stack

```
import java.util.Stack;
class QueueUsingStack {
   Stack<Integer> s1 = new Stack<>();
   Stack<Integer> s2 = new Stack<>();
   public void enqueue(int x) {
      s1.push(x);
   }
   public int dequeue() {
      if (s1.isEmpty() && s2.isEmpty()) {
         System.out.println("Queue is empty");
         return -1;
      }
      if (s2.isEmpty()) {
         while (!s1.isEmpty()) {
            s2.push(s1.pop());
         }
      }
      return s2.pop();
   }
   public int peek() {
      if (s1.isEmpty() && s2.isEmpty()) {
         System.out.println("Queue is empty");
         return -1;
      }
      if (s2.isEmpty()) {
         while (!s1.isEmpty()) {
            s2.push(s1.pop());
         }
      }
      return s2.peek();
   }
   public boolean isEmpty() {
      return s1.isEmpty() && s2.isEmpty();
   }
}
public class Main {
   public static void main(String[] args) {
      QueueUsingStack q = new QueueUsingStack();
      q.enqueue(1);
      q.enqueue(2);
      q.enqueue(3);
```

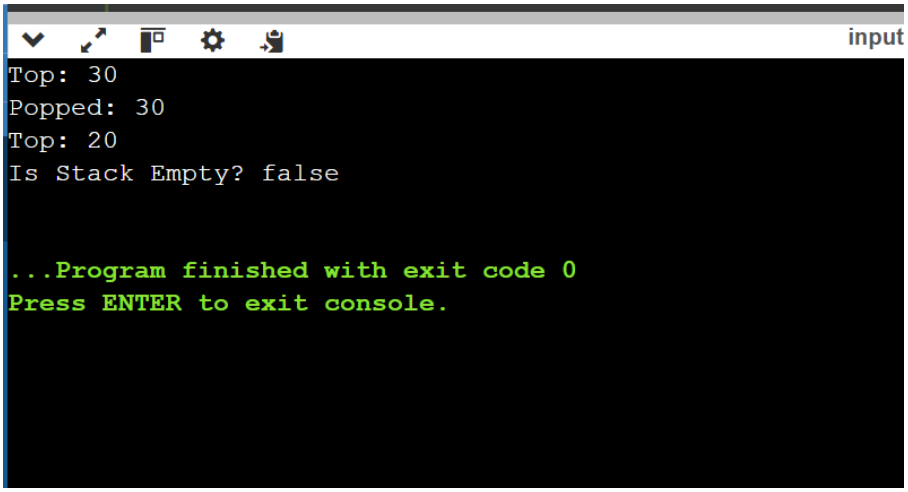
```
        System.out.println("Dequeued: " + q.dequeue()); // 1
        System.out.println("Front: " + q.peek()); // 2
        System.out.println("Dequeued: " + q.dequeue()); // 2
        System.out.println("Dequeued: " + q.dequeue()); // 3
        System.out.println("Is Queue Empty? " + q.isEmpty()); // true
    }
}
```



2. Implement stack using queue

```
import java.util.LinkedList;
import java.util.Queue;
class StackUsingQueue {
    Queue<Integer> q1 = new LinkedList<>();
    Queue<Integer> q2 = new LinkedList<>();
    public void push(int x) {
        q1.add(x);
    }
    public int pop() {
        if (q1.isEmpty()) {
            System.out.println("Stack is empty");
            return -1;
        }
        while (q1.size() > 1) {
            q2.add(q1.poll());
        }
        int poppedElement = q1.poll();
        Queue<Integer> temp = q1;
        q1 = q2;
        q2 = temp;
        return poppedElement;
    }
```

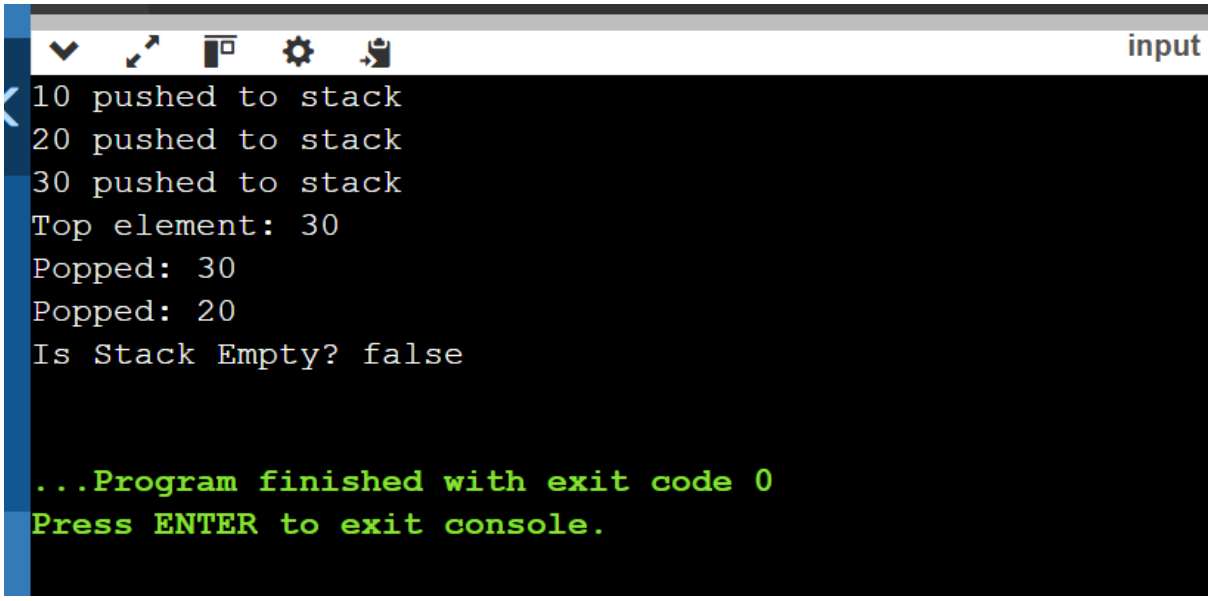
```
    public int top() {
        if (q1.isEmpty()) {
            System.out.println("Stack is empty");
            return -1;
        }
        while (q1.size() > 1) {
            q2.add(q1.poll());
        }
        int topElement = q1.poll();
        q2.add(topElement);
        Queue<Integer> temp = q1;
        q1 = q2;
        q2 = temp;
        return topElement;
    }
    public boolean isEmpty() {
        return q1.isEmpty();
    }
}
public class Main {
    public static void main(String[] args) {
        StackUsingQueue stack = new StackUsingQueue();
        stack.push(10);
        stack.push(20);
        stack.push(30);
        System.out.println("Top: " + stack.top());
        System.out.println("Popped: " + stack.pop());
        System.out.println("Top: " + stack.top());
        System.out.println("Is Stack Empty? " + stack.isEmpty());
    }
}
```



3. Implement stack using array

```
class StackUsingArray {
    private int[] stack;
    private int top;
    private int capacity;
    public StackUsingArray(int size) {
        stack = new int[size];
        capacity = size;
        top = -1;
    }
    public void push(int x) {
        if (top == capacity - 1) {
            System.out.println("Stack Overflow! Cannot push " + x);
            return;
        }
        stack[++top] = x;
        System.out.println(x + " pushed to stack");
    }
    public int pop() {
        if (top == -1) {
            System.out.println("Stack Underflow! Cannot pop");
            return -1;
        }
        return stack[top--];
    }
    public int peek() {
        if (top == -1) {
            System.out.println("Stack is empty");
            return -1;
        }
        return stack[top];
    }
    public boolean isEmpty() {
        return top == -1;
    }
    public boolean isFull() {
        return top == capacity - 1;
    }
}
public class Main {
    public static void main(String[] args) {
        StackUsingArray stack = new StackUsingArray(5);
        stack.push(10);
```

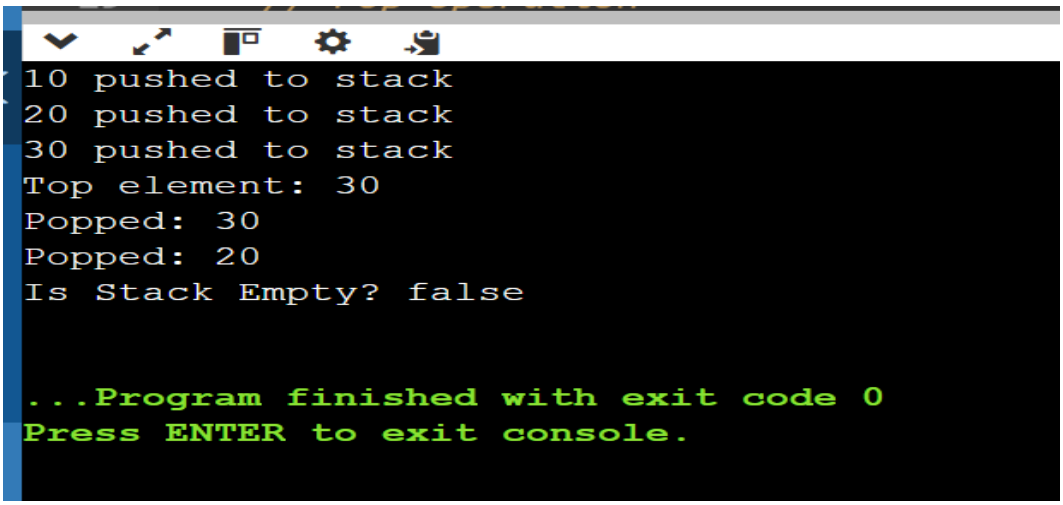
```
        stack.push(20);
        stack.push(30);
        System.out.println("Top element: " + stack.peek()); // 30
        System.out.println("Popped: " + stack.pop()); // 30
        System.out.println("Popped: " + stack.pop()); // 20
        System.out.println("Is Stack Empty? " + stack.isEmpty()); // false
    }
}
```



4. Implement stack using linked list

```
class Node {
    int data;
    Node next;
    Node(int data) {
        this.data = data;
        this.next = null;
    }
}
class StackUsingLinkedList {
    private Node top;
    public StackUsingLinkedList() {
        this.top = null;
    }
    public void push(int x) {
        Node newNode = new Node(x);
        newNode.next = top;
        top = newNode;
        System.out.println(x + " pushed to stack");
    }
    public int pop() {
```

```
        if (top == null) {
            System.out.println("Stack Underflow! Cannot pop");
            return -1;
        }
        int poppedData = top.data;
        top = top.next; // Move top to next node
        return poppedData;
    }
    public int peek() {
        if (top == null) {
            System.out.println("Stack is empty");
            return -1;
        }
        return top.data;
    }
    public boolean isEmpty() {
        return top == null;
    }
}
public class Main {
    public static void main(String[] args) {
        StackUsingLinkedList stack = new StackUsingLinkedList();
        stack.push(10);
        stack.push(20);
        stack.push(30);
        System.out.println("Top element: " + stack.peek());
        System.out.println("Popped: " + stack.pop()); // 30
        System.out.println("Popped: " + stack.pop()); // 20
        System.out.println("Is Stack Empty? " + stack.isEmpty()); // false
    }
}
```



5. Implement bst using linkedlist

```
class Node {
   int data;
   Node left, right;
   public Node(int data) {
      this.data = data;
      this.left = this.right = null;
   }
}
class BST {
   private Node root;
   public BST() {
      root = null;
   }
   public void insert(int key) {
      root = insertRec(root, key);
   }
   private Node insertRec(Node root, int key) {
      if (root == null) {
         return new Node(key);
      }
      if (key < root.data) {
         root.left = insertRec(root.left, key);
      } else if (key > root.data) {
         root.right = insertRec(root.right, key);
      }
      return root;
   }
   public boolean search(int key) {
      return searchRec(root, key);
   }
   private boolean searchRec(Node root, int key) {
      if (root == null) return false;
      if (root.data == key) return true;
      return key < root.data ? searchRec(root.left, key) : searchRec(root.right, key);
   }
   public void inorder() {
      inorderRec(root);
      System.out.println();
   }
   private void inorderRec(Node root) {
      if (root != null) {
         inorderRec(root.left);
```

```
            System.out.print(root.data + " ");
            inorderRec(root.right);
        }
    }
    public void delete(int key) {
        root = deleteRec(root, key);
    }
    private Node deleteRec(Node root, int key) {
        if (root == null) return null;
        if (key < root.data) {
            root.left = deleteRec(root.left, key);
        } else if (key > root.data) {
            root.right = deleteRec(root.right, key);
        } else {
            if (root.left == null) return root.right;
            if (root.right == null) return root.left;
            root.data = minValue(root.right);
            root.right = deleteRec(root.right, root.data);
        }
        return root;
    }
    private int minValue(Node root) {
        int minVal = root.data;
        while (root.left != null) {
            minVal = root.left.data;
            root = root.left;
        }
        return minVal;
    }
}
public class Main {
    public static void main(String[] args) {
        BST tree = new BST();
        tree.insert(50);
        tree.insert(30);
        tree.insert(70);
        tree.insert(20);
        tree.insert(40);
        tree.insert(60);
        tree.insert(80);
        System.out.print("Inorder traversal: ");
        tree.inorder();
        System.out.println("Search 40: " + tree.search(40)); // true
```

```
        System.out.println("Search 90: " + tree.search(90));
        tree.delete(50);
        System.out.print("After deleting 50, Inorder traversal: ");
        tree.inorder();
    }
}
```