

1. Write a java program to create or to implement a stack using linked list

CODE:

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

class Stack {
    Node top;
    public Stack() {
        top = null;
    }
    public void push(int data) {
        Node newNode = new Node(data);
        if (top == null) {
            top = newNode;
        }
        else
        {
            newNode.next = top;
            top = newNode;
        }
    }
    public int pop() {
        if (top == null) {
            return -1;
        }
    }
}
```

```

int data = top.data;
top = top.next;
return data;
}

    public int peek() {
        return top == null ? -1 : top.data;
    }

public boolean isEmpty() {
    return top == null;
}

public void printStack() {
    Node temp = top;
    while (temp != null) {
        System.out.print(temp.data + " ");
        temp = temp.next;
    }
    System.out.println();
}

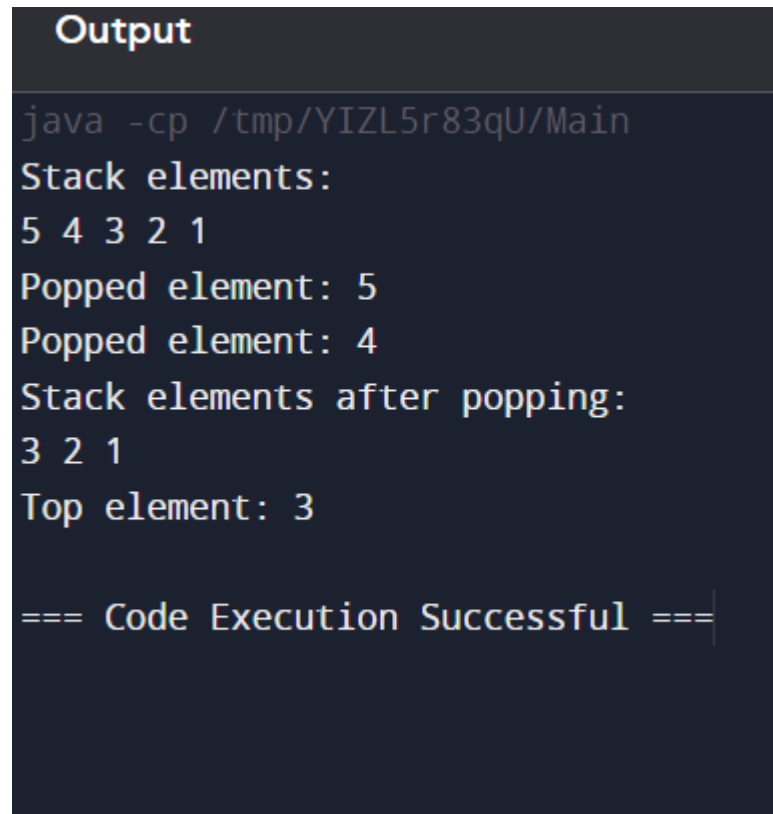
}

public class Main {
    public static void main(String[] args) {
        Stack stack = new Stack();
        stack.push(1);
        stack.push(2);
        stack.push(3);
        stack.push(4);
        stack.push(5);
        System.out.println("Stack elements:");
        stack.printStack();
        System.out.println("Popped element: " + stack.pop());
        System.out.println("Popped element: " + stack.pop());
    }
}

```

```
System.out.println("Stack elements after popping:");
stack.printStack();
System.out.println("Top element: " + stack.peek());
}
}
```

OUTPUT:

A screenshot of a terminal window with a dark background. The title bar of the window says "Output". The terminal shows the execution of a Java program. The output text is as follows:

```
java -cp /tmp/YIZL5r83qU/Main
Stack elements:
5 4 3 2 1
Popped element: 5
Popped element: 4
Stack elements after popping:
3 2 1
Top element: 3

=== Code Execution Successful ===
```

2. Write a java program to create or to implement a queue using linked list

CODE:

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

```
class Queue {  
    Node front;  
    Node rear;  
    public Queue() {  
        this.front = null;  
        this.rear = null;  
    }  
    public void enqueue(int data) {  
        Node newNode = new Node(data);  
        if (rear == null) {  
            front = newNode;  
            rear = newNode;  
        } else {  
            rear.next = newNode;  
            rear = newNode;  
        }  
    }  
    public int dequeue() {  
        if (front == null) {  
            throw new RuntimeException("Queue is empty");  
        }  
        int data = front.data;  
        front = front.next;  
        if (front == null) {  
            rear = null;  
        }  
        return data;  
    }  
    public int peek() {  
        if (front == null) {  
            throw new RuntimeException("Queue is empty");  
        }  
    }  
}
```

```

    }
    return front.data;
}

public boolean isEmpty() {
    return front == null;
}
}

public class Main {
    public static void main(String[] args) {
        Queue queue = new Queue();
        queue.enqueue(5);
        queue.enqueue(4);
        queue.enqueue(3);
        queue.enqueue(2);
        queue.enqueue(1);
        System.out.println("dequeued element: " + queue.dequeue());
        System.out.print("Queue elements after dequeuing: ");
        while (!queue.isEmpty()) {
            System.out.print(queue.dequeue() + " ");
        }
        System.out.println();
        queue.enqueue(1);
        queue.enqueue(2);
        queue.enqueue(3);
        System.out.println("Front element: " + queue.peek());
    }
}

```

OUTPUT:

Output

```
java -cp /tmp/B9izg0qKSX/Main  
dequeued element: 5  
Queue elements after dequeuing: 4 3 2 1  
Front element: 1  
  
=== Code Execution Successful ===
```

3. Hash Map

CODE:

```
import java.util.HashMap;  
import java.util.Map;  
public class HashMapDemo {  
    public static void main(String[] args) {  
        HashMap<Integer, String> map = new HashMap<>();  
        map.put(1, "Apple");  
        map.put(2, "Banana");  
        map.put(3, "Cherry");  
        map.put(4, "Date");  
        System.out.println("Initial HashMap: " + map);  
        System.out.println("Value for key 1: " + map.get(1));  
        System.out.println("Value for key 2: " + map.get(2));  
        System.out.println("HashMap contains key 3: " + map.containsKey(3));  
        System.out.println("HashMap contains key 5: " + map.containsKey(5));  
        System.out.println("HashMap contains value 'Cherry': " + map.containsValue("Cherry"));  
        System.out.println("HashMap contains value 'Grape': " + map.containsValue("Grape"));  
    }  
}
```

```

map.remove(2);

System.out.println("HashMap after removing key 2: " + map);

map.put(3, "Citrus");

System.out.println("HashMap after updating key 3: " + map);

System.out.println("Iterating over HashMap:");

for (Map.Entry<Integer, String> entry : map.entrySet()) {

    System.out.println("Key: " + entry.getKey() + ", Value: " + entry.getValue());

}

System.out.println("Size of HashMap: " + map.size());

map.clear();

System.out.println("HashMap after clearing: " + map);

System.out.println("Size after clearing: " + map.size());

}

}

```

OUTPUT:

Output

Clear

```

java -cp /tmp/uzOgg2dcCp/HashMapDemo
Initial HashMap: {1=Apple, 2=Banana, 3=Cherry, 4=Date}
Value for key 1: Apple
Value for key 2: Banana
HashMap contains key 3: true
HashMap contains key 5: false
HashMap contains value 'Cherry': true
HashMap contains value 'Grape': false
HashMap after removing key 2: {1=Apple, 3=Cherry, 4=Date}
HashMap after updating key 3: {1=Apple, 3=Citrus, 4=Date}
Iterating over HashMap:
Key: 1, Value: Apple
Key: 3, Value: Citrus
Key: 4, Value: Date
Size of HashMap: 3
HashMap after clearing: {}
Size after clearing: 0

=== Code Execution Successful ===

```

4. Compare the students

CODE:

```
import java.util.ArrayList;
import java.util.Collections;
class Student implements Comparable<Student> {
    int rollNo;
    String name;

    Student(int rollNo, String name) {
        this.rollNo = rollNo;
        this.name = name;
    }
    public int compareTo(Student other) {
        if (this.rollNo < other.rollNo) {
            return -1;
        } else if (this.rollNo > other.rollNo) {
            return 1;
        } else {
            return 0;
        }
    }
    public String toString() {
        return "Student{rollNo=" + rollNo + ", name=" + name + "}";
    }
}

public class Main {
    public static void main(String[] args) {
        ArrayList<Student> students = new ArrayList<>();
```



```
students.add(new Student(3, "Alice"));
students.add(new Student(1, "Bob"));
students.add(new Student(2, "Charlie"));

Collections.sort(students);

for (Student student : students) {
    System.out.println(student);
}
}
```

OUTPUT:

Output

```
java -cp /tmp/pp1Qung5VF/Main
Student{rollNo=1, name='Bob'}
Student{rollNo=2, name='Charlie'}
Student{rollNo=3, name='Alice'}

=== Code Execution Successful ===
```