DAY 1

ARRAY

Inserting an element in an Array - Tryout

```
class ArrayTest {
public static void insert(char[] ar, int pos, char val){
//Traversing the array from the last position to the position where the element has to be inserted
                for(int i=ar.length-1;i>=pos;i--){
                   //Moving each element one position to its right
                        ar[i]=ar[i-1];
                }
//Inserting the data at the specified position
                ar[pos-1]=val;
        }
}
class Tester{
        public static void main(String args[]){
                char arr[]=new char[6];
                arr[0]='A';
                arr[1]='B';
                arr[2]='C';
                arr[3]='D';
                arr[4]='E';
                //Make changes and try to insert elements at different positions
                ArrayTest.insert(arr, 3, 'J');
                for(int i=0;i<arr.length;i++)
                        System.out.println(arr[i]);
        }
```

```
}
```

Deleting an element in an Array - Tryout

```
class ArrayTest {
        public static void delete(char[] ar, int pos){
           //Traversing the array from the position where the element has to be deleted to the end
                for(int i=pos-1;i<ar.length-1;i++){
                   //Moving each element one position to the left
                        ar[i]=ar[i+1];
                }
//The space that is left at the end is filled with character '0'
                ar[ar.length-1]='0';
        }
}
class Tester{
        public static void main(String args[]){
                char arr[]=new char[6];
                arr[0]='A';
                arr[1]='B';
                arr[2]='J';
                arr[3]='C';
                arr[4]='D';
                arr[5]='E';
//Make changes and try to delete elements from different positions
                ArrayTest.delete(arr, 3);
                for(int i=0;i<arr.length;i++)
                        System.out.println(arr[i]);
        }
}
```

Adding an element to a linked list – Tryout

```
class Node {
	private String data;
	private Node next;
	public Node(String data){
		this.data=data;
	}
	public void setData(String data){
		this.data = data;
	}
	public void setNext(Node node){
		this.next = node;
	}
	public String getData(){
		return this.data;
	}
	public Node getNext(){
		return this.next;
	}
}
class LinkedList {
	private Node head;
	private Node tail;
	public Node getHead(){
		return this.head;
	}
	public Node getTail(){
		return this.tail;
	}
	public void addAtEnd(String data){
//Create a new node
		Node node = new Node(data);
```

```
//Check if the list is empty,
//if yes, make the node as the head and the tail
                if(this.head == null)
                        this.head=this.tail=node;
                else{
   //If the list is not empty, add the element at the end
                this.tail.setNext(node);
   //Make the new node as the tail
                this.tail=node;
                }
        }
}
class Tester{
        public static void main(String args[]){
                LinkedList list = new LinkedList();
                list.addAtEnd("Milan");
                list.addAtEnd("Venice");
                list.addAtEnd("Munich");
                list.addAtEnd("Vienna");
                System.out.println("Adding an element to the linked list");
        }
}
```

Displaying a linked list – Tryout

```
class Node {
        private String data;
        private Node next;

        public Node(String data) {
                this.data = data;
        }
```

```
	public void setData(String data) {
		this.data = data;
	}
	public void setNext(Node node) {
		this.next = node;
	}
	public String getData() {
		return this.data;
	}
	public Node getNext() {
		return this.next;
	}
}
class LinkedList {
	private Node head;
	private Node tail;
	public Node getHead() {
		return this.head;
	}
	public Node getTail() {
		return this.tail;
	}
	public void addAtEnd(String data) {
// Create a new node
		Node node = new Node(data);
// Check if the list is empty,
// if yes, make the node as the head and the tail
		if (this.head == null)
			this.head = this.tail = node;
		else {
// If the list is not empty, add the element at the end
```

```
                this.tail.setNext(node);
// Make the new node as the tail
                this.tail = node;
            }
        }
        public void addAtBeginning(String data) {
// Create a new node
            Node node = new Node(data);
// Check if the list is empty,
// if yes, make the node as the head and the tail
            if (this.head == null)
                this.head = this.tail = node;
            else {
        // If the list is not empty, add the element at the beginning
                node.setNext(this.head);
        // Make the new node as the head
                this.head = node;
            }
        }
        public void display() {
// Initialize temp to the head node
            Node temp = this.head;
// Traverse the list and print data of each node
            while (temp != null) {
                System.out.println(temp.getData());
                temp = temp.getNext();
            }
        }
        public static void main(String args[]) {
            LinkedList list = new LinkedList();
            list.addAtEnd("Milan");
```

```
                list.addAtEnd("Venice");
                list.addAtEnd("Munich");
                list.addAtEnd("Vienna");
                list.display();
        }
}
```

Searching for an element in a linked list – Tryout

```
class Node {
        private String data;
        private Node next;
        public Node(String data) {
                this.data = data;
        }
        public void setData(String data) {
                this.data = data;
        }
        public void setNext(Node node) {
                this.next = node;
        }
        public String getData() {
                return this.data;
        }
        public Node getNext() {
                return this.next;
        }
}

class LinkedList {
        private Node head;
        private Node tail;
```

```
	public Node getHead() {
		return this.head;
	}
	public Node getTail() {
		return this.tail;
	}
	public void addAtEnd(String data) {
// Create a new node
		Node node = new Node(data);
// Check if the list is empty,
// if yes, make the node as the head and the tail
		if (this.head == null)
			this.head = this.tail = node;
		else {
// If the list is not empty, add the element at the end
			this.tail.setNext(node);
// Make the new node as the tail
			this.tail = node;
		}

	}
	public void addAtBeginning(String data) {
// Create a new node
		Node node = new Node(data);
// Check if the list is empty,
// if yes, make the node as the head and the tail
		if (this.head == null)
			this.head = this.tail = node;
		else {
// If the list is not empty, add the element at the beginning
			node.setNext(this.head);
```

```
// Make the new node as the head
                        this.head = node;
                }
        }
        public void display() {
// Initialize temp to the head node
                Node temp = this.head;
// Traverse the list and print data of each node
                while (temp != null) {
                        System.out.println(temp.getData());
                        temp = temp.getNext();
                }
        }
        public Node find(String data) {
                Node temp = this.head;
// Traverse the list and return the node
// if the data of it matches with the searched data
                while (temp != null) {
                        if (temp.getData().equals(data))
                                return temp;
                        temp = temp.getNext();
                }
                return null;
        }
        public static void main(String args[]) {
                LinkedList list = new LinkedList();
                list.addAtEnd("Milan");
                list.addAtEnd("Venice");
                list.addAtEnd("Munich");
                list.addAtEnd("Vienna");
                list.display();
```

```
		if (list.find("Munich") != null)
			System.out.println("Node found");
		else
			System.out.println("Node not found");
	}
}
```

Inserting an element in a linked list – Tryout

```
class Node {
	private String data;
	private Node next;
	public Node(String data) {
		this.data = data;
	}
	public void setData(String data) {
		this.data = data;
	}
	public void setNext(Node node) {
		this.next = node;
	}
	public String getData() {
		return this.data;
	}
	public Node getNext() {
		return this.next;
	}
}

class LinkedList {
	private Node head;
	private Node tail;
```

```
	public Node getHead() {
		return this.head;
	}
	public Node getTail() {
		return this.tail;
	}
	public void addAtEnd(String data) {
// Create a new node
		Node node = new Node(data);
// Check if the list is empty,
// if yes, make the node as the head and the tail
		if (this.head == null)
			this.head = this.tail = node;
		else {
// If the list is not empty, add the element at the end
			this.tail.setNext(node);
// Make the new node as the tail
			this.tail = node;
		}
	}
	public void addAtBeginning(String data) {
// Create a new node
		Node node = new Node(data);
// Check if the list is empty,
// if yes, make the node as the head and the tail
		if (this.head == null)
			this.head = this.tail = node;
		else {
// If the list is not empty, add the element at the beginning
			node.setNext(this.head);
// Make the new node as the head
```

```
                this.head = node;
            }
        }
        public void display() {
// Initialize temp to the head node
            Node temp = this.head;
// Traverse the list and print data of each node
            while (temp != null) {
                System.out.println(temp.getData());
                temp = temp.getNext();
            }
        }
        public Node find(String data) {
            Node temp = this.head;
// Traverse the list and return the node
// if the data of it matches with the searched data
            while (temp != null) {
                if (temp.getData().equals(data))
                    return temp;
                temp = temp.getNext();
            }
            return null;
        }
        public void insert(String data, String dataBefore) {
            Node node = new Node(data);
// Check if the list is empty,
// if yes, make the node as the head and the tail
            if (this.head == null)
                this.head = this.tail = node;
            else {
// Find the node after which the data has to be inserted
```

```
                        Node nodeBefore = this.find(dataBefore);
                        if (nodeBefore != null) {
// Insert the new node after nodeBefore
                                node.setNext(nodeBefore.getNext());
                                nodeBefore.setNext(node);
// If nodeBefore is currently the tail node,
// make the new node as the tail node
                                if (nodeBefore == this.tail)
                                        this.tail = node;
                        } else
                                System.out.println("Node not found");
                }
        }
        public static void main(String args[]) {
                LinkedList list = new LinkedList();
                list.addAtEnd("Milan");
                list.addAtEnd("Venice");
                list.addAtEnd("Munich");
                list.addAtEnd("Vienna");
                list.insert("Prague", "Munich");
                list.display();
        }
}
```

Deleting an element from a linked list – Tryout

```
class Node {
        private String data;
        private Node next;
        public Node(String data) {
                this.data = data;
        }
```

```
	public void setData(String data) {
		this.data = data;
	}
	public void setNext(Node node) {
		this.next = node;
	}
	public String getData() {
		return this.data;
	}
	public Node getNext() {
		return this.next;
	}
}
class LinkedList {
	private Node head;
	private Node tail;
	public Node getHead() {
		return this.head;
	}
	public Node getTail() {
		return this.tail;
	}
	public void addAtEnd(String data) {
// Create a new node
		Node node = new Node(data);
// Check if the list is empty,
// if yes, make the node as the head and the tail
		if (this.head == null)
			this.head = this.tail = node;
		else {
// If the list is not empty, add the element at the end
```

```
                this.tail.setNext(node);
// Make the new node as the tail
                this.tail = node;
            }
        }
        public void addAtBeginning(String data) {
// Create a new node
            Node node = new Node(data);
// Check if the list is empty,
// if yes, make the node as the head and the tail
            if (this.head == null)
                this.head = this.tail = node;
            else {
// If the list is not empty, add the element at the beginning
                node.setNext(this.head);
// Make the new node as the head
                this.head = node;
            }
        }
        public void display() {
// Initialize temp to the head node
            Node temp = this.head;
// Traverse the list and print data of each node
            while (temp != null) {
                System.out.println(temp.getData());
                temp = temp.getNext();
            }
        }
        public Node find(String data) {
            Node temp = this.head;
// Traverse the list and return the node
```

```
// if the data of it matches with the searched data
                while (temp != null) {
                        if (temp.getData().equals(data))
                                return temp;
                        temp = temp.getNext();
                }
                return null;
        }
        public void insert(String data, String dataBefore) {
                Node node = new Node(data);
// Check if the list is empty,
// if yes, make the node as the head and the tail
                if (this.head == null)
                        this.head = this.tail = node;
                else {
// Find the node after which the data has to be inserted
                        Node nodeBefore = this.find(dataBefore);
                        if (nodeBefore != null) {
// Insert the new node after nodeBefore
                                node.setNext(nodeBefore.getNext());
                                nodeBefore.setNext(node);
// If nodeBefore is currently the tail node,
// make the new node as the tail node
                                if (nodeBefore == this.tail)
                                        this.tail = node;
                        } else
                                System.out.println("Node not found");
                }
        }
        public void delete(String data) {
// Check if the list is empty,
```

```
        if (this.head == null)
            System.out.println("List is empty");
        else {
// Find the node to be deleted
            Node node = this.find(data);
// If the node is not found
            if (node == null)
                System.out.println("Node not found");
// If the node to be deleted is the head node
            else if (node == this.head) {
                this.head = this.head.getNext();
                node.setNext(null);
// If the node to be deleted is also the tail node
                if (node == this.tail)
                    tail = null;
            } else {
// Traverse to the node present before the node to be deleted
                Node nodeBefore = null;
                Node temp = this.head;
                while (temp != null) {
                    if (temp.getNext() == node) {
                        nodeBefore = temp;
                        break;
                    }
                    temp = temp.getNext();
                }
// Remove the node
                nodeBefore.setNext(node.getNext());
// If the node to be deleted is the tail node,
// then make the previous node as the tail
                if (node == this.tail)
```

```
                                        this.tail = nodeBefore;
                                    node.setNext(null);
                                }
                        }
                }
        public static void main(String args[]) {
                LinkedList list = new LinkedList();
                list.addAtEnd("Milan");
                list.addAtEnd("Venice");
                list.addAtEnd("Munich");
                list.addAtEnd("Prague");
                list.addAtEnd("Vienna");
                list.display();
                System.out.println("-------------------------");
                list.delete("Venice");
                list.display();
                /*
                 * if(list.find("Munich")!=null) System.out.println("Node found"); else
                 * System.out.println("Node not found");
                 */
        }
}
```

Linked List - Exercise 1

```
class Node {
   private String data;
        private Node next;
        public Node(String data) {
                this.data = data;
        }
        public void setData(String data) {
```

```
            this.data = data;
        }
        public void setNext(Node node) {
            this.next = node;
        }
        public String getData() {
            return this.data;
        }
        public Node getNext() {
            return this.next;
        }
}
class LinkedList {
    private Node head;
        private Node tail;
        public Node getHead() {
            return this.head;
        }
        public Node getTail() {
            return this.tail;
        }
        public void addAtEnd(String data) {
            Node node = new Node(data);
            if (this.head == null) {
                this.head = this.tail = node;
            } else {
                this.tail.setNext(node);
                this.tail = node;
            }
        }
        public void addAtBeginning(String data) {
```

```
        Node node = new Node(data);
        if (this.head == null) {
                this.head = this.tail = node;
        }
        else {
                node.setNext(this.head);
                this.head = node;
        }
}
public void display() {
        Node temp = this.head;
        while (temp != null) {
                System.out.println(temp.getData());
                temp = temp.getNext();
        }
}
public Node find(String data) {
        Node temp = this.head;

        while (temp != null) {
                if (temp.getData().equals(data))
                        return temp;
                temp = temp.getNext();
        }
        return null;
}
public void insert(String data, String dataBefore) {
Node node = new Node(data);
        if (this.head == null)
                this.head = this.tail = node;
        else {
```

```
                Node nodeBefore = this.find(dataBefore);
                if (nodeBefore != null) {
                        node.setNext(nodeBefore.getNext());
                        nodeBefore.setNext(node);
                        if (nodeBefore == this.tail)
                                this.tail = node;
                } else
                        System.out.println("Node not found");
        }
}

public void delete(String data) {
        if (this.head == null)
                System.out.println("List is empty");
        else {
                Node node = this.find(data);
                if (node == null)
                        System.out.println("Node not found");
                if (node == this.head) {
                        this.head = this.head.getNext();
                        node.setNext(null);
                        if (node == this.tail)
                                tail = null;
           }
           else {
                        Node nodeBefore = null;
                        Node temp = this.head;
                        while (temp != null) {
                                if (temp.getNext() == node) {
                                        nodeBefore = temp;
                                        break;
```

```
                    }
                    temp = temp.getNext();
                }
                nodeBefore.setNext(node.getNext());
                if (node == this.tail)
                    this.tail = nodeBefore;
                node.setNext(null);
            }
        }
    }
}
class Tester {
   public static void main(String args[]) {
        LinkedList linkedList = new LinkedList();
        linkedList.addAtEnd("AB");
        linkedList.addAtEnd("BC");
        linkedList.addAtEnd("CD");
        linkedList.addAtEnd("DE");
        linkedList.addAtEnd("EF");
        String elementToBeFound = "CD";
        int position = findPosition(elementToBeFound, linkedList.getHead());
        if (position != 0)
           System.out.println("The position of the element is " + position);
        else
           System.out.println("The element is not found!");
      }

      public static int findPosition(String element, Node head) {
        //Implement your code here and change the return value accordingly
        int position = 1; // Start position from 1
    Node temp = head;
```

```
// Start traversal from the head
        while (temp != null) {
            if (temp.getData().equals(element)) {
                return position; // Return position if element is found
            }
            temp = temp.getNext(); // Move to the next node
            position++; // Increment position counter
        }
                return 0;
             }
}
```

Linked List - Assignment 1

```
class Node {
    private String data;
    private Node next;
    public Node(String data) {
        this.data = data;
    }
    public void setData(String data) {
        this.data = data;
    }
    public void setNext(Node node) {
        this.next = node;
    }
    public String getData() {
        return this.data;
    }
    public Node getNext() {
        return this.next;
```

```
    }
}
class LinkedList {
    private Node head;
    private Node tail;
    public Node getHead() {
        return this.head;
    }
    public Node getTail() {
        return this.tail;
    }
    public void setHead(Node head) {
        this.head = head;
    }
    public void setTail(Node tail) {
        this.tail = tail;
    }
    public void addAtEnd(String data) {
        Node node = new Node(data);
        if (this.head == null) {
            this.head = this.tail = node;
        } else {
            this.tail.setNext(node);
            this.tail = node;
        }
    }
    public void addAtBeginning(String data) {
        Node node = new Node(data);
        if (this.head == null) {
            this.head = this.tail = node;
        } else {
```

```
        node.setNext(this.head);
        this.head = node;
    }
}
public void display() {
    Node temp = this.head;
    while (temp != null) {
        System.out.println(temp.getData());
        temp = temp.getNext();
    }
}
public Node find(String data) {
    Node temp = this.head;
    while (temp != null) {
        if (temp.getData().equals(data))
            return temp;
        temp = temp.getNext();
    }
    return null;
}
public void insert(String data, String dataBefore) {
    Node node = new Node(data);
    if (this.head == null)
        this.head = this.tail = node;
    else {
        Node nodeBefore = this.find(dataBefore);
        if (nodeBefore != null) {
            node.setNext(nodeBefore.getNext());
            nodeBefore.setNext(node);
            if (nodeBefore == this.tail)
                this.tail = node;
```

```
        } else
            System.out.println("Node not found");
    }
}
public void delete(String data) {
    if (this.head == null)
        System.out.println("List is empty");
    else {
        Node node = this.find(data);
        if (node == null)
            System.out.println("Node not found");
        if (node == this.head) {
            this.head = this.head.getNext();
            node.setNext(null);
            if (node == this.tail)
                tail = null;
        } else {
            Node nodeBefore = null;
            Node temp = this.head;
            while (temp != null) {
                if (temp.getNext() == node) {
                    nodeBefore = temp;
                    break;
                }
                temp = temp.getNext();
            }
            nodeBefore.setNext(node.getNext());
            if (node == this.tail)
                this.tail = nodeBefore;
            node.setNext(null);
        }
```

```
        }
    }
}
class Tester {
    public static void main(String args[]) {
        LinkedList linkedList1 = new LinkedList();
        linkedList1.addAtEnd("ABC");
        linkedList1.addAtEnd("DFG");
        linkedList1.addAtEnd("XYZ");
        linkedList1.addAtEnd("EFG");

        LinkedList linkedList2 = new LinkedList();
        linkedList2.addAtEnd("ABC");
        linkedList2.addAtEnd("DFG");
        linkedList2.addAtEnd("XYZ");
        linkedList2.addAtEnd("EFG");
        System.out.println("Initial List");
        linkedList1.display();
        System.out.println("\nList after left shifting by 2 positions");
        shiftListLeft(linkedList1, 2);
        linkedList1.display();
        System.out.println("\nInitial List");
        linkedList2.display();
        System.out.println("\nList after right shifting by 2 positions");
        shiftListRight(linkedList2, 2);
        linkedList2.display();
    }
    public static void shiftListLeft(LinkedList linkedList, int n) {
        if (linkedList.getHead() == null || n <= 0) {
            return; // No shift needed if list is empty or n <= 0
        }
```

```
        Node current = linkedList.getHead();
        int count = 1;

        // Traverse to the nth node
        while (count < n && current != null) {
            current = current.getNext();
            count++;
        }
        if (current == null) {
            return; // If n is greater than or equal to the length of the list, no shift needed
        }
        // current is now the nth node
        linkedList.setHead(current.getNext()); // New head is the node after the nth node
        linkedList.setTail(current); // Set current node as the new tail
        current.setNext(null); // Set next of current node to null to break the chain
    }
    public static void shiftListRight(LinkedList linkedList, int n) {
        if (linkedList.getHead() == null || n <= 0) {
            return;
// No shift needed if list is empty or n <= 0
        }
        Node current = linkedList.getHead();
        Node previous = null;
        int count = 0;

        // Traverse to find the end of the list
        while (current.getNext() != null) {
            previous = current;
            current = current.getNext();
            count++;
        }
```

```
        // current is now the tail node, previous is its previous node
        int length = count + 1; // Length of the linked list
        // Adjust n to be within the length of the list
        n = n % length;
        if (n == 0) {
            return; // No shift needed if n is multiple of the length or zero
        }
        // Move tail to the head and adjust pointers
        previous.setNext(null); // Set next of previous to null to break the chain
        current.setNext(linkedList.getHead()); // Set old tail's next to the old head
        linkedList.setHead(current); // Set current as the new head
        linkedList.setTail(previous); // Set previous as the new tail
    }
}
```

Linked List - Assignment 2

```
class Node {
    private String data;
    private Node next;
    public Node(String data) {
        this.data = data;
    }
    public void setData(String data) {
        this.data = data;
    }
    public void setNext(Node node) {
        this.next = node;
    }
    public String getData() {
        return this.data;
    }
```

```
    public Node getNext() {
        return this.next;
    }
}
class LinkedList {
    private Node head;
    private Node tail;
    public Node getHead() {
        return this.head;
    }
    public Node getTail() {
        return this.tail;
    }
    public void addAtEnd(String data) {
        Node node = new Node(data);
        if (this.head == null) {
            this.head = this.tail = node;
        } else {
            this.tail.setNext(node);
            this.tail = node;
        }
    }
    public void addAtBeginning(String data) {
        Node node = new Node(data);
        if (this.head == null) {
            this.head = this.tail = node;
        } else {
            node.setNext(this.head);
            this.head = node;
        }
    }
```

```
public void display() {
    Node temp = this.head;
    while (temp != null) {
        System.out.print(temp.getData() + " ");
        temp = temp.getNext();
    }
    System.out.println();
}
public Node find(String data) {
    Node temp = this.head;
    while (temp != null) {
        if (temp.getData().equals(data))
            return temp;
        temp = temp.getNext();
    }
    return null;
}
public void insert(String data, String dataBefore) {
    Node node = new Node(data);
    if (this.head == null)
        this.head = this.tail = node;
    else {
        Node nodeBefore = this.find(dataBefore);
        if (nodeBefore != null) {
            node.setNext(nodeBefore.getNext());
            nodeBefore.setNext(node);
            if (nodeBefore == this.tail)
                this.tail = node;
        } else
            System.out.println("Node not found");
    }
```

```
}
public void delete(String data) {
    if (this.head == null) {
        System.out.println("List is empty");
        return;
    }
    Node node = this.find(data);
    if (node == null) {
        System.out.println("Node not found");
        return;
    }
    if (node == this.head) {
        this.head = this.head.getNext();
        node.setNext(null);
        if (node == this.tail)
            tail = null;
    } else {
        Node nodeBefore = null;
        Node temp = this.head;
        while (temp != null) {
            if (temp.getNext() == node) {
                nodeBefore = temp;
                break;
            }
            temp = temp.getNext();
        }
        nodeBefore.setNext(node.getNext());
        if (node == this.tail)
            this.tail = nodeBefore;
        node.setNext(null);
    }
```

```
    }
}
class Tester {
    public static void main(String args[]) {
        LinkedList linkedList = new LinkedList();
        LinkedList reversedLinkedList = new LinkedList();
        linkedList.addAtEnd("Data");
        linkedList.addAtEnd("Structures");
        linkedList.addAtEnd("and");
        linkedList.addAtEnd("Algorithms");
        System.out.println("Initial List:");
        linkedList.display();
        System.out.println();
        reverseList(linkedList.getHead(), reversedLinkedList);
        System.out.println("Reversed List:");
        reversedLinkedList.display();
    }
    public static void reverseList(Node head, LinkedList reversedLinkedList) {
        if (head == null) {
            return; // Base case: end of the list
        }
      // Recursively call for the next node
        reverseList(head.getNext(), reversedLinkedList);
      // Add current node's data to the reversedLinkedList at the end
        reversedLinkedList.addAtEnd(head.getData());
    }
}
```

STACK

Stack – Tryout

```
class Stack {
	private int top;
// represents the index position of the top most element in the stack
	private int maxSize;
// represents the maximum number of elements that can be stored in the stack
	private int[] arr;
	Stack(int maxSize) {
		this.top = -1; // top is -1 when the stack is created
		this.maxSize = maxSize;
		arr = new int[maxSize];
	}
	// Checking if the stack is full or not
	public boolean isFull() {
		if (top >= (maxSize - 1)) {
			return true;
		}
		return false;
	}
	// Adding a new element to the top of the stack
	public boolean push(int data) {
		if (isFull()) {
			return false;
		} else {
			arr[++top] = data;
			return true;
		}
	}
	// Returning the top most element of the stack
	public int peek() {
		if (isEmpty())
			return Integer.MIN_VALUE;
```

```
            else
                return arr[top];
        }
        // Displaying all the elements of the stack
        public void display() {
            if (isEmpty())
                System.out.println("Stack is empty!");
            else {
                System.out.println("Displaying stack elements");
                for (int index = top; index >= 0; index--) {
                    System.out.println(arr[index]); // accessing element at position
index
                }
            }
        }
        // Checking if the stack is empty or not
        public boolean isEmpty() {
            if (top < 0) {
                return true;
            }
            return false;
        }
        // Removing the element from the top of the stack
        public int pop() {
            if (isEmpty())
                return Integer.MIN_VALUE;
            else
                return arr[top--];
        }
}
class Tester {
```

```
public static void main(String args[]) {
        Stack stack = new Stack(5);
        System.out.println("Stack created.\n");
        if (stack.push(1))
                System.out.println("The element is pushed to the stack!\n");
        else
                System.out.println("Stack is full!\n");
        if (stack.push(2))
                System.out.println("The element is pushed to the stack!\n");
        else
                System.out.println("Stack is full!\n");
        if (stack.push(3))
                System.out.println("The element is pushed to the stack!\n");
        else
                System.out.println("Stack is full!\n");
        if (stack.push(4))
                System.out.println("The element is pushed to the stack!\n");
        else
                System.out.println("Stack is full!\n");
        if (stack.push(5))
                System.out.println("The element is pushed to the stack!\n");
        else
                System.out.println("Stack is full!\n");
        stack.display();
        if (stack.push(6))
                System.out.println("The element is pushed to the stack!\n");
        else
                System.out.println("Stack is full!\n");
        System.out.println("The top element is : " + stack.peek());
        int poppedElement = stack.pop();
        if (poppedElement == Integer.MIN_VALUE)
```

```
			System.out.println("Stack is empty\n");
		else
		System.out.println("The element popped out is : " + poppedElement + "\n");


		poppedElement = stack.pop();
		if (poppedElement == Integer.MIN_VALUE)
			System.out.println("Stack is empty\n");
		else
		System.out.println("The element popped out is : " + poppedElement + "\n");
		poppedElement = stack.pop();
		if (poppedElement == Integer.MIN_VALUE)
			System.out.println("Stack is empty\n");
		else
		System.out.println("The element popped out is : " + poppedElement + "\n");
		poppedElement = stack.pop();
		if (poppedElement == Integer.MIN_VALUE)
			System.out.println("Stack is empty\n");
		else
		System.out.println("The element popped out is : " + poppedElement + "\n");
		poppedElement = stack.pop();
		if (poppedElement == Integer.MIN_VALUE)
			System.out.println("Stack is empty\n");
		else
		System.out.println("The element popped out is : " + poppedElement + "\n");
		poppedElement = stack.pop();
		if (poppedElement == Integer.MIN_VALUE)
			System.out.println("Stack is empty\n");
		else
		System.out.println("The element popped out is : " + poppedElement + "\n");
	}
}
```

Stack - Exercise

```
class Stack {
    private int top;
    private int maxSize;
    private int[] arr;
    Stack(int maxSize) {
        this.top = -1;
        this.maxSize = maxSize;
        arr = new int[maxSize];
    }
    public boolean isFull() {
        if (top >= (maxSize - 1)) {
            return true;
        }
        return false;
    }
    public boolean push(int data) {
        if (isFull()) {
            return false;
        }
        else {
            arr[++top] = data;
            return true;
        }
    }
    public int peek() {
        if (isEmpty())
            return Integer.MIN_VALUE;
        else
```

```
            return arr[top];
    }
    public void display() {
        if (isEmpty())
            System.out.println("Stack is empty!");
        else {
            System.out.println("Displaying stack elements");
            for (int index = top; index >= 0; index--) {
                System.out.println(arr[index]); // accessing element at position index
            }
        }
    }
    public boolean isEmpty() {
        if (top < 0) {
            return true;
        }
        return false;
    }
    public int pop() {
        if (isEmpty())
            return Integer.MIN_VALUE;
        else
            return arr[top--];
    }
}
class Tester {
    public static void main(String args[]) {
        Stack stack = new Stack(10);
        stack.push(15);
        stack.push(25);
        stack.push(30);
```

```
        stack.push(40);
        stack.display();
        if (checkTop(stack)) {
            System.out.println("The top most element of the stack is an even number");
        } else {
            System.out.println("The top most element of the stack is an odd number");
        }
    }
    public static boolean checkTop(Stack stack) {
        //Implement your code here and change the return value accordingly
         if (stack.isEmpty()) {
            return false; // Stack is empty, no element to check
        }
        int topElement = stack.peek(); // Get the top element of the stack
        return topElement % 2 == 0; // Check if the top element is even
    }
}
```