EX.NO : (i)

```
// Java code for stack implementation

import java.io.*;
import java.util.*;

class Test
{
    // Pushing element on the top of the stack
    static void stack_push(Stack<Integer> stack)
    {
        for(int i = 0; i < 5; i++)
        {
            stack.push(i);
        }
    }

    // Popping element from the top of the stack
    static void stack_pop(Stack<Integer> stack)
    {
        System.out.println("Pop Operation:");

        for(int i = 0; i < 5; i++)
        {
            Integer y = (Integer) stack.pop();
            System.out.println(y);
        }
    }

    // Displaying element on the top of the stack
    static void stack_peek(Stack<Integer> stack)
    {
        Integer element = (Integer) stack.peek();
        System.out.println("Element on stack top: " + element);
    }

    // Searching element in the stack
    static void stack_search(Stack<Integer> stack, int element)
    {
        Integer pos = (Integer) stack.search(element);

        if(pos == -1)
            System.out.println("Element not found");
        else
            System.out.println("Element is found at position: " + pos);
    }
    public static void main (String[] args)
    {
        Stack<Integer> stack = new Stack<Integer>();

        stack_push(stack);
        stack_pop(stack);
        stack_push(stack);
        stack_peek(stack);
        stack_search(stack, 2);
        stack_search(stack, 6);
    }
}
```

Output:

```
Pop Operation:

4

3

2

1

0

Element on stack top: 4

Element is found at position: 3

Element not found
```

EX.NO : (ii)

```
/ implementation of queue
 // A class to represent a queue
class Queue {
    int front, rear, size;
    int capacity;
    int array[];

    public Queue(int capacity)
    {
        this.capacity = capacity;
        front = this.size = 0;
        rear = capacity - 1;
        array = new int[this.capacity];
    }
     // Queue is full when size becomes
    // equal to the capacity
    boolean isFull(Queue queue)
    {
        return (queue.size == queue.capacity);
    }
     // Queue is empty when size is 0
    boolean isEmpty(Queue queue)
    {
        return (queue.size == 0);
    }
    // Method to add an item to the queue.
    // It changes rear and size
    void enqueue(int item)
    {
        if (isFull(this))
            return;
        this.rear = (this.rear + 1)
                    % this.capacity;
        this.array[this.rear] = item;
        this.size = this.size + 1;
        System.out.println(item
                           + " enqueued to queue");
    }
     // Method to remove an item from queue.
    // It changes front and size
    int dequeue()
    {
        if (isEmpty(this))
            return Integer.MIN_VALUE;

        int item = this.array[this.front];
        this.front = (this.front + 1)
```

```
                    % this.capacity;
        this.size = this.size - 1;
        return item;
    }
    // Method to get front of queue
    int front()
    {
        if (isEmpty(this))
            return Integer.MIN_VALUE;

        return this.array[this.front];
    }

    // Method to get rear of queue
    int rear()
    {
        if (isEmpty(this))
            return Integer.MIN_VALUE;

        return this.array[this.rear];
    }
}

// Driver class
public class Test {
    public static void main(String[] args)
    {
        Queue queue = new Queue(1000);

        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);
        queue.enqueue(40);

        System.out.println(queue.dequeue()
                           + " dequeued from queue\n");

        System.out.println("Front item is "
                           + queue.front());

        System.out.println("Rear item is "
                           + queue.rear());
    }
}
```

Output

```
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
40 enqueued to queue
10 dequeued from queue
Front item is 20
Rear item is 40
```