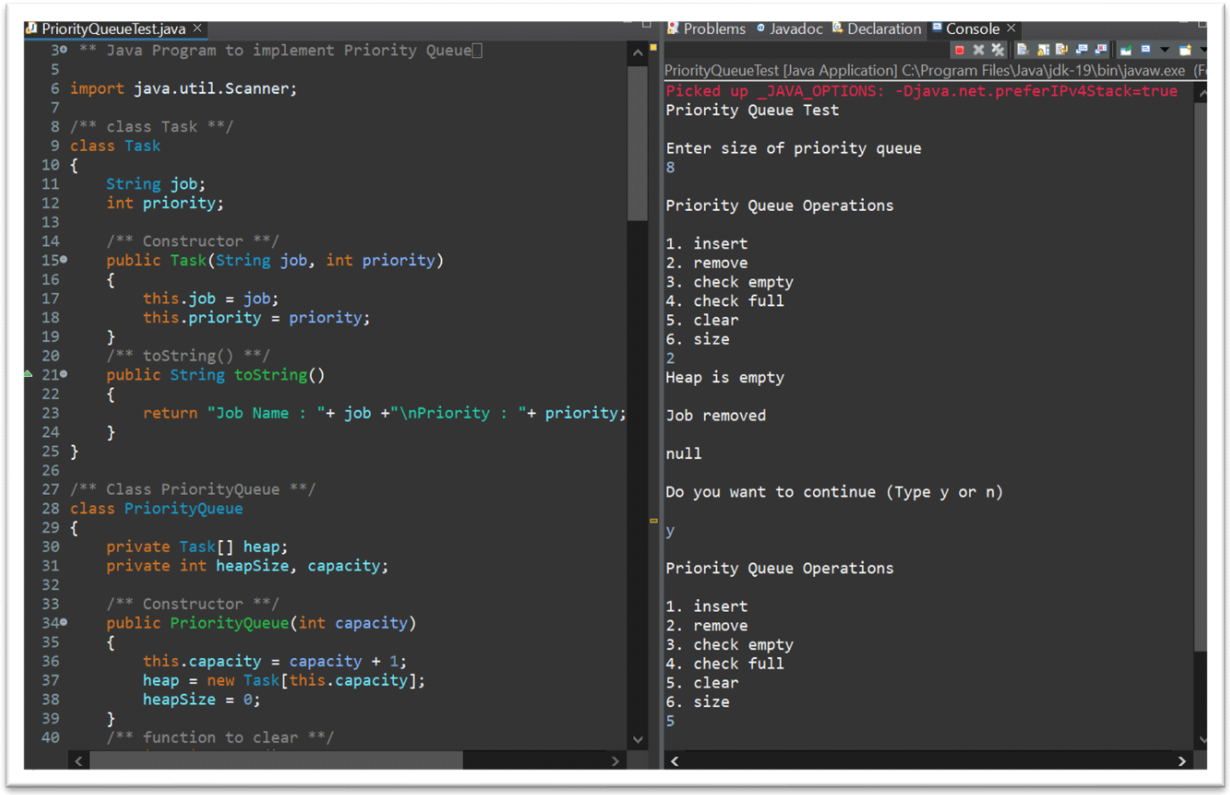
LABSHEET - 3

AIM 1: Understanding the concepts of Stack, Queue , Linked List (10points)

- **Write a Java program to implement a priority queue ADT.**



Code:

```
package q;
/**
 ** Java Program to implement Priority Queue
 **/

import java.util.Scanner;

/** class Task
**/class Task
{
    String job;

     int priority;
```

```
    /** Constructor **/
    public Task(String job, int priority)
    {
        this.job =
        job;
        this.priority
        = priority;
    }
    /**
    toString()
    **/ public
    String
    toString()
    {
        return "Job Name : "+ job +"\nPriority : "+ priority;
    }
}

/** Class
PriorityQueue
**/class
PriorityQueue
{
    private Task[] heap;
    private int heapSize, capacity;

    /** Constructor **/
    public PriorityQueue(int capacity)
    {
        this.capacity =
        capacity + 1; heap =
        new
        Task[this.capacity];
        heapSize = 0;
    }
    /** function
    to clear **/
    public void
    clear()
    {
        heap = new
        Task[capacity];
        heapSize = 0;
    }
```

```
/** function to check if
empty **/public boolean
isEmpty()
{
    return heapSize == 0;
}
/** function to check
if full **/public
boolean isFull()
{
    return heapSize == capacity - 1;
}
/** function to get Size **/

public int size()
{
    return heapSize;
}
/** function to insert task **/
public void insert(String job, int priority)
{
    Task newJob = new Task(job, priority);

    heap[++heapSize
    ] = newJob;int
    pos = heapSize;
    while (pos != 1 && newJob.priority > heap[pos/2].priority)
    {
        heap[pos] =
        heap[pos/2];
        pos /=2;
    }
    heap[pos] = newJob;
}
/** function to
remove task **/
public Task remove()
{
    int
    parent,
    child;
    Task
    item,
    temp;
    if
```

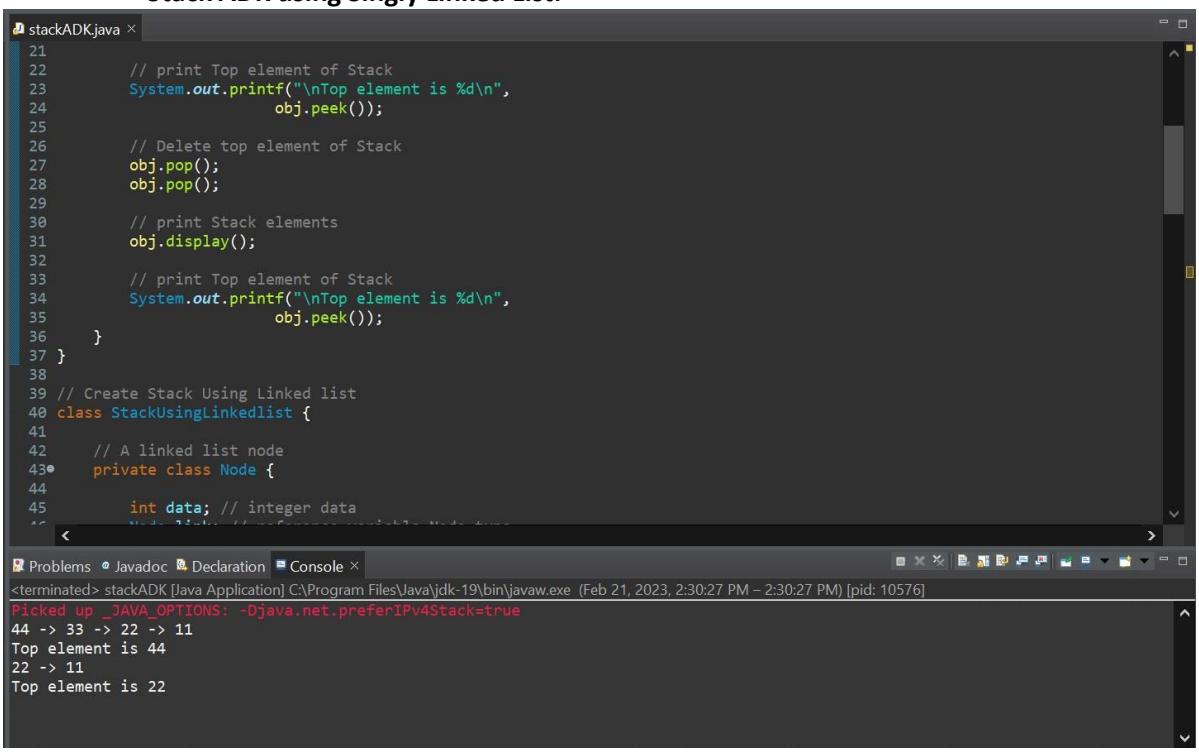
```
        (isEmpt
        y() )
        {
            System.out.println("Heap
            is empty");return null;
        }

        item = heap[1];
        temp = heap[heapSize--];

        parent = 1;
        child = 2;
        while (child <= heapSize)
        {
            if (child < heapSize && heap[child].priority <
heap[child +1].priority)
                child++;
            if (temp.priority >=
                heap[child].priority)break;

            heap[parent] =
            heap[child];
            parent = child;
            child *= 2;

        }
        heap[parent] = temp;

        return item;
    }
}

/** Class
PriorityQueueTest
**/public class
PriorityQueueTest
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Priority Queue Test\n");

        System.out.println("Enter size of priority queue ");
        PriorityQueue pq = new PriorityQueue(scan.nextInt()
        );
```

```
char ch;
/*  Perform Priority Queue
operations */do
{
    System.out.println("\nPriority Queue
    Operations\n");System.out.println("1.
    insert"); System.out.println("2.
    remove"); System.out.println("3. check
    empty"); System.out.println("4. check
    full"); System.out.println("5. clear");
    System.out.println("6. size");

    int choice =
    scan.nextInt();
    switch (choice)
    {
    case 1 :
        System.out.println("Enter job name and
        priority");pq.insert(scan.next(),
        scan.nextInt() );
        b
    r
    e
    a
    k
    ;

    c
    a
    s
    e

    2

    :
        System.out.println("\nJob removed \n\n"+
        pq.remove());break;
    case 3 :
        System.out.println("\nEmpty Status : "+
        pq.isEmpty() );break;
    case 4 :
        System.out.println("\nFull Status : "+ pq.isFull() );
```

```
                break
            ;case 5 :
                System.out.println("\nPriority Queue
                Cleared");pq.clear();
                break
            ;case 6 :
                System.out.println("\nSize = "+ pq.size()
                );break;
            default :
                System.out.println("Wrong Entry \n
                ");break;
            }

            System.out.println("\nDo you want to continue (Type y or n)
            \n");ch = scan.next().charAt(0);
        } while (ch == 'Y'|| ch == 'y');
    }
}
```

- **Write Java programs to implement the following using a singly linked list. (a) Stack ADT (b) QueueADT**

  - **Stack ADK using Singly Linked List.**



Code:

```java
// Java program to Implement a stack
// using singly linked list
// import package
import static java.lang.System.exit;

// Driver code
class stackADK
{
    public static void main(String[] args)
    {
        // create Object of Implementing class
        StackUsingLinkedlist obj
            = new StackUsingLinkedlist();
        // insert Stack
        valueobj.push(11);
        obj.push(22);
        obj.push(33);
        obj.push(44);


         // print Stack
         elements
         obj.display();

         // print Top element of Stack
         System.out.printf("\nTop element is
         %d\n",
                         obj.peek());

         // Delete top
         element of Stack
         obj.pop();
         obj.pop();

         // print Stack
         elements
         obj.display();

         // print Top element of Stack
         System.out.printf("\nTop element is
         %d\n",
                         obj.peek());
    }
}
```

```
// Create Stack Using
Linked listclass
StackUsingLinkedlist {

    // A
    linked
    list node
    private
    class Node
    {

        int data; // integer data
        Node link; // reference variable Node type
    }
    // create global top reference
    variable globalNode top;
    // Constructor
    StackUsingLinkedlist() { this.top = null; }

    // Utility function to add an element x
    in the stackpublic void push(int x) //
    insert at the beginning
    {
        // create new node temp and
        allocate memoryNode temp = new
        Node();

        // check if stack (heap) is full. Then inserting an
        // element would lead to
        stack overflowif (temp ==
        null) {
            System.out.print("\nHeap Overflow");

            return;
        }

        // initialize data into temp
        data fieldtemp.data = x;

        // put top reference into
        temp linktemp.link = top;

        // update top
        referencetop =
        temp;
```

```
}

// Utility function to check if the stack is empty or
// not
public boolean isEmpty() { return top == null; }

// Utility function to return top element
in a stackpublic int peek()
{
    // check for
    empty stackif
    (!isEmpty()) {
        return top.data;
    }
    else {
        System.out.println("Stack
        is empty");return -1;
    }
}

// Utility function to pop top element
from the stackpublic void pop() // remove
at the beginning
{
    // check for stack
    underflowif (top
    == null) {
        System.out.print("\nStack
        Underflow");return;
    }

    // update the top pointer to point to
    the next nodetop = (top).link;
}

public void display()
{

   // check for stack underflow
   if (top == null) {
       System.out.printf("\nStack
       Underflow");exit(1);
   }
   else {
```

```
            Node temp  =  top;
            while (temp != null)
            {

                // print node data
                System.out.print(temp.data);

                // assign temp link to temp
                temp = temp.link;
                if(temp != null)
                    System.out.print(" ->
                    ");
            }
        }
    }
}
```

- **Queue ADK using Singly Linked List.**

```
class Node {
    private static int front, rear, capacity;
    private static int Node[];

    Node(int size) {
        front = rear = 0;
        capacity = size;
        Node = new int[capacity];
    }

    // insert an element into the Node
    static void NodeEnNode(int item)  {
        // check if the Node is full
        if (capacity == rear) {
            System.out.printf("\nNode is full\n");
            return;
        }

        // insert element at the rear
        else {
            Node[rear] = item;
            rear++;
        }
        return;
    }

    //remove an element from the Node
    static void NodeDeNode()  {
        // check if Node is empty
        if (front == rear) {
            System.out.printf("\nNode is empty\n");
            return;
        }

        // shift elements to the right by one place uptil rear
        else {
            for (int i = 0; i < rear - 1; i++) {
                Node[i] = Node[i + 1];
            }


      // set Node[rear] to 0
            if (rear < capacity)
                Node[rear] = 0;

            // decrement rear
            rear--;
        }
        return;
    }

    // print Node elements
    static void NodeDisplay()
    {
        int i;
        if (front == rear) {
            System.out.printf("Node is Empty\n");
            return;
        }

        // traverse front to rear and print elements
        for (i = front; i < rear; i++) {
            System.out.printf(" %d = ", Node[i]);
        }
        return;
    }

    // print front of Node
    static void NodeFront()
    {
        if (front == rear) {
            System.out.printf("Node is Empty\n");
            return;
        }
        System.out.printf("\nFront Element of the Node: %d", Node[front]);
        return;
    }
}

public class Main {
    public static void main(String[] args) {
        // Create a Node of capacity 4
        Node q = new Node(4);

        System.out.println("Initial Node:");
       // print Node elements
        q.NodeDisplay();

        // inserting elements in the Node
        q.NodeEnNode(10);
        q.NodeEnNode(30);
        q.NodeEnNode(50);
        q.NodeEnNode(70);

        // print Node elements
        System.out.println("Node after EnNode Operation:");
        q.NodeDisplay();

        // print front of the Node
        q.NodeFront();

        // insert element in the Node
        q.NodeEnNode(90);

        // print Node elements
```

```
106         q.NodeDisplay();
107
108         q.NodeDeNode();
109         q.NodeDeNode();
110         System.out.printf("\nNode after two deNode operations:");
111
112         // print Node elements
113         q.NodeDisplay();
114
115         // print front of the Node
116         q.NodeFront();
117     }
118 }
```

```
CPU Time: 0.16 sec(s), Memory: 33652 kilobyte(s)          compiled and executed in 0.969 sec(s)

Initial Node:
Node is Empty
Node after EnNode Operation:
 10 =  30 =  50 =  70 =
Front Element of the Node: 10
Node is full
 10 =  30 =  50 =  70 =
Node after two deNode operations: 50 =  70 =
Front Element of the Node: 50
```

Code:

```
class Node {
    private static int front, rear,
    capacity;private static int Node[];

    Node(int size) {
        front = rear =
        0;capacity =
        size;
        Node = new int[capacity];
    }

    // insert an element into the Node
    static void NodeEnNode(int item)  {
        // check if the Node is full
        if (capacity == rear) {
            System.out.printf("\nNode is
            full\n");return;
        }

        // insert element at the rear
        else {
            Node[rear] =
            item;rear++;
        }
        return;
    }

    //remove an element from the Node
    static void NodeDeNode()  {
        // check if Node is empty
        if (front == rear) {
```

```
        System.out.printf("\nNode is
        empty\n");return;
    }

    // shift elements to the right by one place
    uptil rearelse {
        for (int i = 0; i < rear
            - 1; i++) {Node[i] =
            Node[i + 1];
        }


  // set Node[rear] to 0
        if (rear
            <
            capac
            ity)
            Node[
            rear]
            = 0;

        //
        decrem
        ent
        rear
        rear--
        ;
    }
    return;
}

// print Node
elements
static void
NodeDisplay()
{
    int i;
    if (front == rear) {
        System.out.printf("Node is
        Empty\n");return;
    }
```

```
        // traverse front to rear and print
        elementsfor (i = front; i < rear;
        i++) {
            System.out.printf(" %d = ", Node[i]);
        }
        return;
    }

    // print
    front of
    Node static
    void
    NodeFront()
    {
        if (front == rear) {
            System.out.printf("Node is
            Empty\n");return;
        }
        System.out.printf("\nFront Element of the Node: %d", Node[front]);

        return;
    }
}

public class Main {
    public static void main(String[] args) {
        // Create a Node of
        capacity 4Node q =
        new Node(4);

        System.out.println("Initial Node:");
       // print
        Node
        elements
        q.NodeDisp
        lay();

        // inserting elements
        in the Node
        q.NodeEnNode(10);
        q.NodeEnNode(30);
        q.NodeEnNode(50);
        q.NodeEnNode(70);

        // print Node elements
```

```
        System.out.println("Node after EnNode
        Operation:");q.NodeDisplay();

        // print front of
        the Node
        q.NodeFront();

        // insert element in
        the Node
        q.NodeEnNode(90);

        // print
        Node
        elements
        q.NodeDispl
        ay();

        q.No
        deDe
        Node
        ();
        q.No
        deDe
        Node
        ();
        System.out.printf("\nNode after two deNode operations:");

        // print
        Node
        elements
        q.NodeDispl
        ay();

        // print front of
        the Node
        q.NodeFront();
    }
}
```