```
//implementing a Linked List
import.java.io*;
public class linkedList {
          Node head;
         static class Node {
              int nodeData;
              Node next;
             Note(int x) {
       nodeData = x;
       next = null;
    }
}
public static linkedList insert (linkedList list, int nodeData) {
         Node newNode = new Node (nodeData);
         newNode.next =  null;
         if(list.head == null) {
               list.head == newNode;
}
else
      Node last = list.head;
       while(last.next !=null) {
            last = last.next;
}
          last.next = newNode;
}
return last;
}
public static void printTheList (linkedList list) {
           Node currentNode = list.head;
           System.out.println("This is the linked list: ");
       while (currentNode != null) {
      System.out.println(currentNode.nodeData + " ");
      currentNode = currentNode.next;
   }
}
public static voide main(String[] args) {
       linkedList list = new linkedList;
        list = insert(list, 1);
        list = insert(list, 20393);
        list = insert(list, 353);
        list = insert(list, 1029);
```

```
       list = insert(list 23);
      printTheList(list)
     }
}


//implementing a stack based on linked lists
public class StackUsingLinkedLists {
    private class Node {

         int data;
         Node linked;
}
Node head;
//constructor
StackUsingLinkedLists() {
      this.head = null;
}
public void push(int x) {
    Node temporary = new Node();
       if(temporary ==null) {
       System.out.print("Stack Overflow!");
       return;
}
   temporary.data = x;
   temporary.linked = head;
   head = temporary;
}

//check if the stack is empty
public boolean isEmpty() {
   return head == null;
}
public int peek() {
     if(!isEmpty()) {
      return head.data;
}
else {
    System.out.println(Stack is Empty!");
     return -1;
  }
}
public void pop() {
```

```
      if(head==null) {
       System.out.print(“Stack Underflow!”);
       return;
}
//top pointer points to the next
head = (head).linked;
}

public void show() {
    if (head == null) {
    Systme.out.printf(“Stack Underflow!”);
    exit(1);
}
else {
    Node temporary = head;
    while(temporary!= null) {
        temporary = temporary.linked;
     }
  }
}

public static void main(String[] args) {
     StackUsingLinkedLists y = new StackUsingLinkedList();

    y.push(10);
    y.push(20);
    y.push(30);
    y.push(40);

    y.show();

    y.pop();
    y.pop();

    y.show();
}


//implementing a queue based on linked lists
class QueueNode {
int x;
QueueNode next;
```

```
//constructor
public QueueNode(int x)
{
this.x = x;
this.next = null;
}
}

class queue {
QueueNode head, tail;
public queue()
{
this.head = this.tail = null;
}

void enqueue(int x)
{
QueueNode temporary = new QueueNode(x);
if (this.tail == null) {
this.head = this.tail = temporary;
return;
}
this.tail.next = temporary;
this.tail = temporary;
}

void dequeue()
{
if (this.head == null)
return;
                QueueNode temporary = this.head;
this.head = this.head.next;

if (this.head == null)
this.tail = null;
}
}

public class tryItOut {
public static void main(String[] args) {
queue Queue = new queue();

Queue.enqueue(1);
```

```
Queue.enqueue(2);

Queue.dequeue();
Queue.dequeue();

Queue.enqueue(3);
Queue.enqueue(4);
Queue.enqueue(5);

Queue.dequeue();

System.out.println("Queue Front: " + Queue.head.x);
System.out.println("Queue Rear: " + Queue.tail.x);
}
}


//implementing queue that is created from 2 stacks
public class QueueTwoStacks {

     private Stack <Integer> StackOne = new Stack<>();
     private Stack <Integer> StackTwo = new Stack<>();

        public void enqueue(int x) {
             StackOne.push(x);
}

public void dequeue() {
          if(StackTwo.isEmpty()) {
               if(StackOne.isEmpty()) {
                    System.out.println(“Error. Can’t dequeue value because the queue is
empty!”);
}
else {
       while(!StackOne.isEmpty()) {
          StackTwo.push(StackOne.pop());
     }
  }
}
System.out.println(StackTwo.pop());
}

public static void main(String[] args) {
```

```
    QueueTwoStacks Q = new QueueTwoStacks();
    Q.enqueue(1);
    Q.enqueue(2);
    Q.enqueue(3);

    Q.dequeue();

    Q.enqueue(4);
    Q.enqueue(5);
    Q.enqueue(6);

    Q.dequeue();
    Q.dequeue();
    Q.dequeue();
    Q.dequeue();
   }
}
```