**Adv. Java Lab Assignment**

**Data Structures 10 Programs**

**Name:-Milind Bajaj**

**M.Tech(5th Sem)**

**Rollno.:-IT-2k17-35**

**1.Program of Array in Java.**

**Solution:-**

class Myarray{

public static void main(String args[]){

int a[]=new int[5];

a[0]=10;

a[1]=20;

a[2]=70;

a[3]=40;

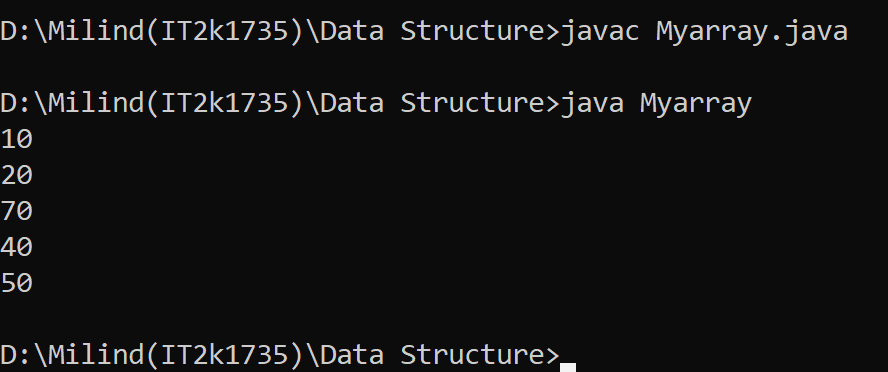
for(int i=0;i<a.length;i++)

System.out.println(a[i]);

}

}

**Output 1:-**

****

**2.Program of Stack Implement in Java.**

**Solution:-**

public class MyStackImpl {

private int stackSize;

private int[] stackArr;

private int top;

public MyStackImpl(int size) {

this.stackSize = size;

this.stackArr = new int[stackSize];

this.top = -1;

}

public void push(int entry) throws Exception {

if(this.isStackFull()){

throw new Exception("Stack is already full. Can not add element.");

}

System.out.println("Adding: "+entry);

this.stackArr[++top] = entry;

}

public int pop() throws Exception {

if(this.isStackEmpty()){

throw new Exception("Stack is empty. Can not remove element.");

}

int entry = this.stackArr[top--];

System.out.println("Removed entry: "+entry);

return entry;

}

public int peek() {

return stackArr[top];

}

public boolean isStackEmpty() {

return (top == -1);

}

public boolean isStackFull() {

return (top == stackSize - 1);

}

public static void main(String[] args) {

MyStackImpl stack = new MyStackImpl(5);

try {

stack.push(4);

stack.push(8);

stack.push(3);

stack.push(89);

stack.pop();

stack.push(34);

stack.push(45);

stack.push(78);

} catch (Exception e) {

System.out.println(e.getMessage());

}

try {

stack.pop();

stack.pop();

stack.pop();

stack.pop();

stack.pop();

stack.pop();

} catch (Exception e) {

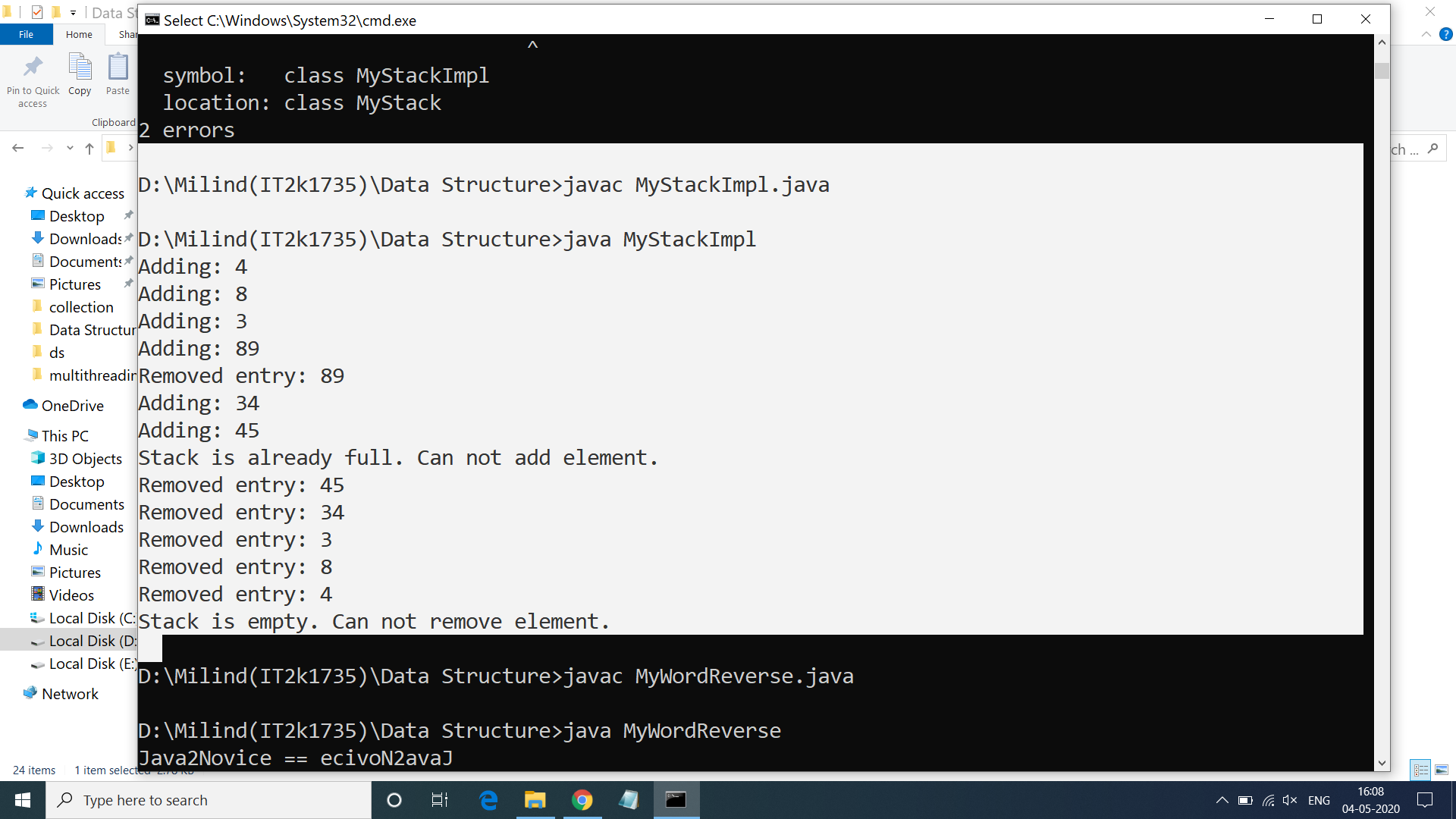
System.out.println(e.getMessage());

}

}

}

**Output 2:-**

****

**3.Program of Queue Implement in Java.**

**Solution:-**

public class QueueImpl {

private int capacity;

int queueArr[];

int front = 0;

int rear = -1;

int currentSize = 0;

public QueueImpl(int queueSize){

this.capacity = queueSize;

queueArr = new int[this.capacity];

}

public void enqueue(int item) {

if (isQueueFull()) {

System.out.println("Overflow ! Unable to add element: "+item);

} else {

rear++;

if(rear == capacity-1){

rear = 0;

}

queueArr[rear] = item;

currentSize++;

System.out.println("Element " + item+ " is pushed to Queue !");

}

}

public void dequeue() {

if (isQueueEmpty()) {

System.out.println("Underflow ! Unable to remove element from Queue");

} else {

front++;

if(front == capacity-1){

System.out.println("Pop operation done ! removed: "+queueArr[front-1]);

front = 0;

} else {

System.out.println("Pop operation done ! removed: "+queueArr[front-1]);

}

currentSize--;

}

}

public boolean isQueueFull(){

boolean status = false;

if (currentSize == capacity){

status = true;

}

return status;

}

public boolean isQueueEmpty(){

boolean status = false;

if (currentSize == 0){

status = true;

}

return status;

}

public static void main(String a[]){

QueueImpl queue = new QueueImpl(4);

queue.enqueue(4);

queue.dequeue();

queue.enqueue(56);

queue.enqueue(2);

queue.enqueue(67);

queue.dequeue();

queue.dequeue();

queue.enqueue(24);

queue.dequeue();

queue.enqueue(98);

queue.enqueue(45);

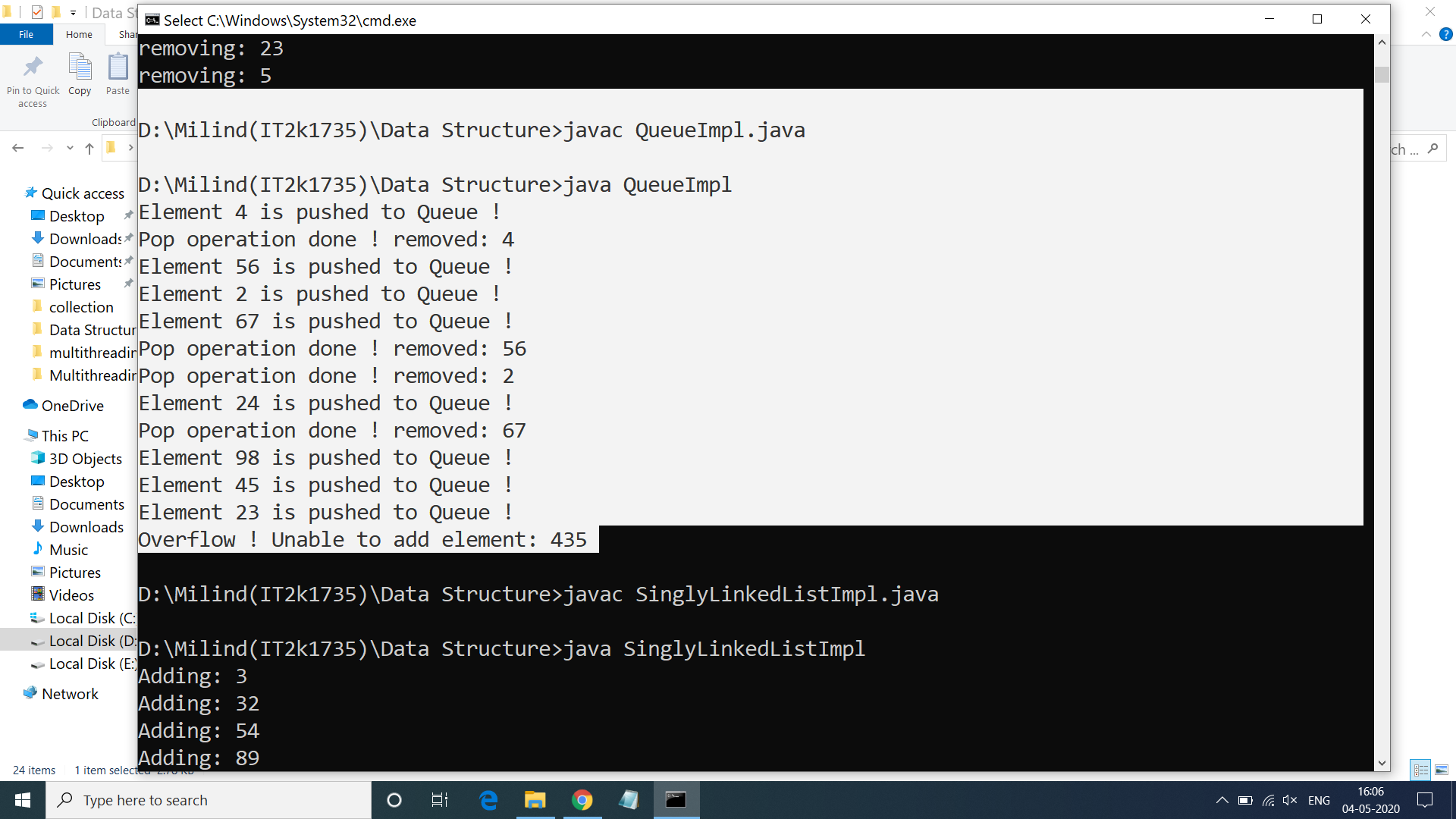
queue.enqueue(23);

queue.enqueue(435);

}

}

**Output 3:-**

****

**4.Program of Single LinkedList In Java.**

**Solution:-**

public class SinglyLinkedListImpl<T> {

private Node<T> head;

private Node<T> tail;

public void add(T element){

Node<T> nd = new Node<T>();

nd.setValue(element);

System.out.println("Adding: "+element);

if(head == null){

head = nd;

tail = nd;

} else {

tail.setNextRef(nd);

tail = nd;

}

}

public void addAfter(T element, T after){

Node<T> tmp = head;

Node<T> refNode = null;

System.out.println("Traversing to all nodes..");

while(true){

if(tmp == null){

break;

}

if(tmp.compareTo(after) == 0){

refNode = tmp;

break;

}

tmp = tmp.getNextRef();

}

if(refNode != null){

Node<T> nd = new Node<T>();

nd.setValue(element);

nd.setNextRef(tmp.getNextRef());

if(tmp == tail){

tail = nd;

}

tmp.setNextRef(nd);

} else {

System.out.println("Unable to find the given element...");

}

}

public void deleteFront(){

if(head == null){

System.out.println("Underflow...");

}

Node<T> tmp = head;

head = tmp.getNextRef();

if(head == null){

tail = null;

}

System.out.println("Deleted: "+tmp.getValue());

public void traverse(){

Node<T> tmp = head;

while(true){

if(tmp == null){

break;

}

System.out.println(tmp.getValue());

tmp = tmp.getNextRef();

}

}

public static void main(String a[]){

SinglyLinkedListImpl<Integer> sl = new SinglyLinkedListImpl<Integer>();

sl.add(3);

sl.add(32);

sl.add(54);

sl.add(89);

sl.addAfter(76, 54);

sl.deleteFront();

sl.deleteAfter(76);

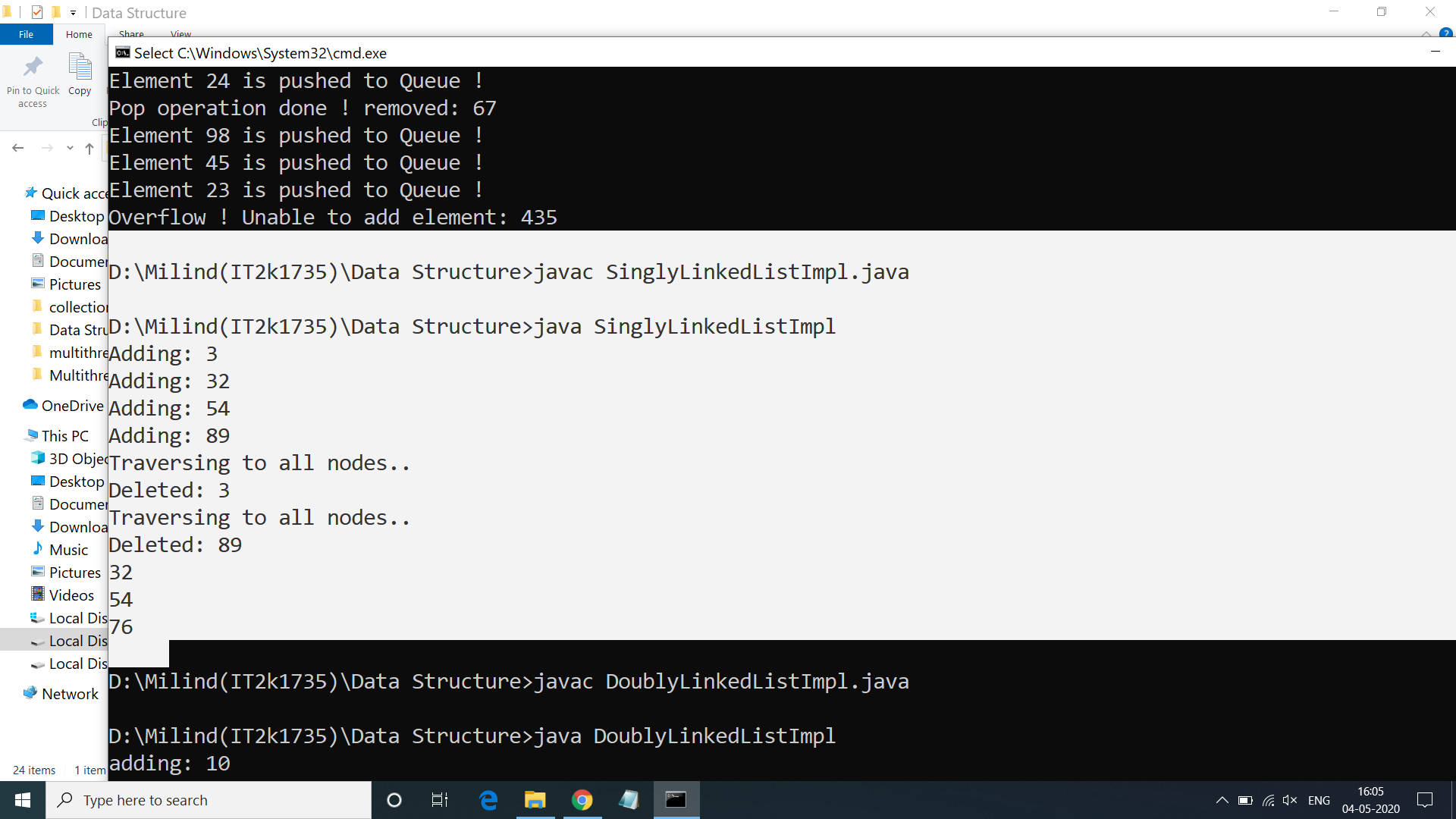
sl.traverse();

}

}

}

**Output 4:-**

****

**5.Program of Matrix in Java**

**Solution:-**

import java.util.Scanner;

public class CreateMatrix {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

System.out.println("Enter The Number Of Matrix Rows");

int matrixRow = scan.nextInt();

System.out.println("Enter The Number Of Matrix Columns");

int matrixCol = scan.nextInt();

int[][] matrix = new int[matrixRow][matrixCol];

enterMatrixData(scan, matrix, matrixRow, matrixCol);

printMatrix(matrix, matrixRow, matrixCol);

}

public static void enterMatrixData(Scanner scan, int[][] matrix, int matrixRow, int matrixCol){

System.out.println("Enter Matrix Data");

for (int i = 0; i < matrixRow; i++)

{

for (int j = 0; j < matrixCol; j++)

{

matrix[i][j] = scan.nextInt();

}

}

}

public static void printMatrix(int[][] matrix, int matrixRow, int matrixCol){

System.out.println("Your Matrix is : ");

for (int i = 0; i < matrixRow; i++)

{

for (int j = 0; j < matrixCol; j++)

{

System.out.print(matrix[i][j]+"\t");

}

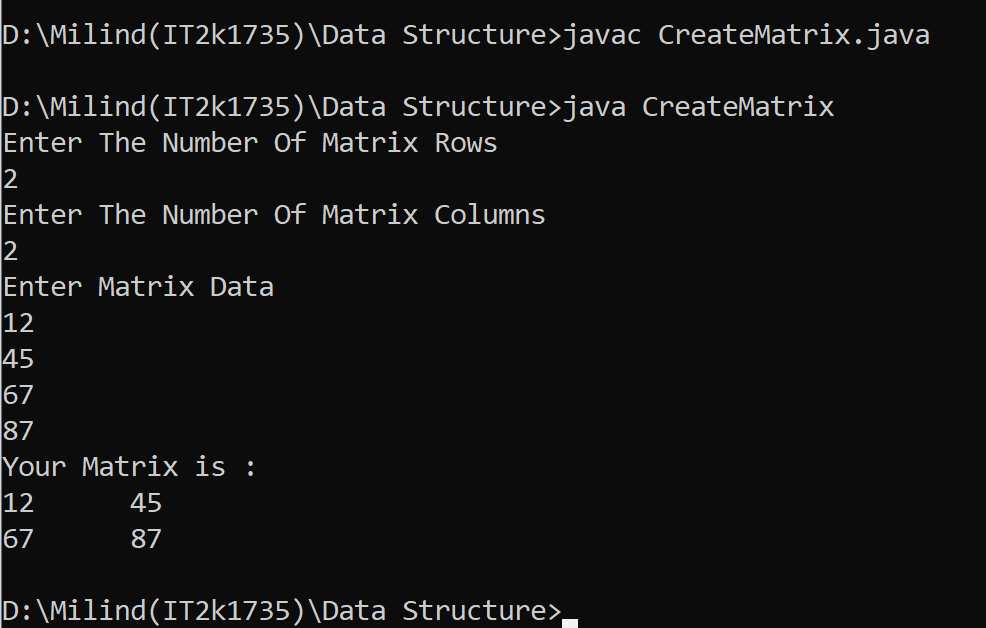
System.out.println();

}

}

}

**Output 5:-**

****

**6.Program of Binary Tree in Java.**

**Solution:-**

public class Tree {

static class Node {

int value;

Node left, right;

Node(int value){

this.value = value;

left = null;

right = null;

}

}

public void insert(Node node, int value) {

if (value < node.value) {

if (node.left != null) {

insert(node.left, value);

} else {

System.out.println(" Inserted " + value + " to left of " + node.value);

node.left = new Node(value); }

} else if (value > node.value) {

if (node.right != null) {

insert(node.right, value);

} else {

System.out.println(" Inserted " + value + " to right of "

+ node.value);

node.right = new Node(value);

}

}

}

public void traverseInOrder(Node node) {

if (node != null) {

traverseInOrder(node.left);

System.out.print(" " + node.value);

traverseInOrder(node.right);

}

}

public static void main(String args[])

{

Tree tree = new Tree();

Node root = new Node(5);

System.out.println("Binary Tree Example");

System.out.println("Building tree with root value " + root.value);

tree.insert(root, 2);

tree.insert(root, 4);

tree.insert(root, 8);

tree.insert(root, 6);

tree.insert(root, 7);

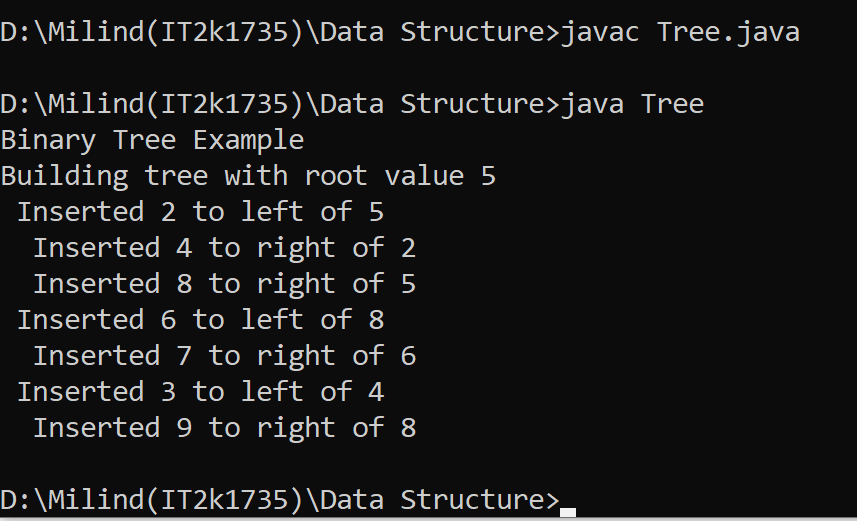
tree.insert(root, 3);

tree.insert(root, 9);

}

}

**Output 6:-**

****

**7.Program of Binary Search Tree in Java.**

**Solution:-**

public class Binary {

private BstNode root;

public boolean isEmpty() {

return (this.root == null);

}

public void insert(Integer data) {

System.out.print("[input: "+data+"]");

if(root == null) {

this.root = new BstNode(data);

System.out.println(" -> inserted: "+data);

return;

}

insertNode(this.root, data);

System.out.print(" -> inserted: "+data);

System.out.println();

}

private BstNode insertNode(BstNode root, Integer data) {

BstNode tmpNode = null;

System.out.print(" ->"+root.getData());

if(root.getData() >= data) {

System.out.print(" [L]");

if(root.getLeft() == null) {

root.setLeft(new BstNode(data));

return root.getLeft();

} else {

tmpNode = root.getLeft();

}

} else {

System.out.print(" [R]");

if(root.getRight() == null) {

root.setRight(new BstNode(data));

return root.getRight();

} else {

tmpNode = root.getRight();

}

}

return insertNode(tmpNode, data);

}

public static void main(String a[]) {

Binary bst = new Binary();

bst.insert(10);

bst.insert(20);

bst.insert(21);

bst.insert(8);

bst.insert(6);

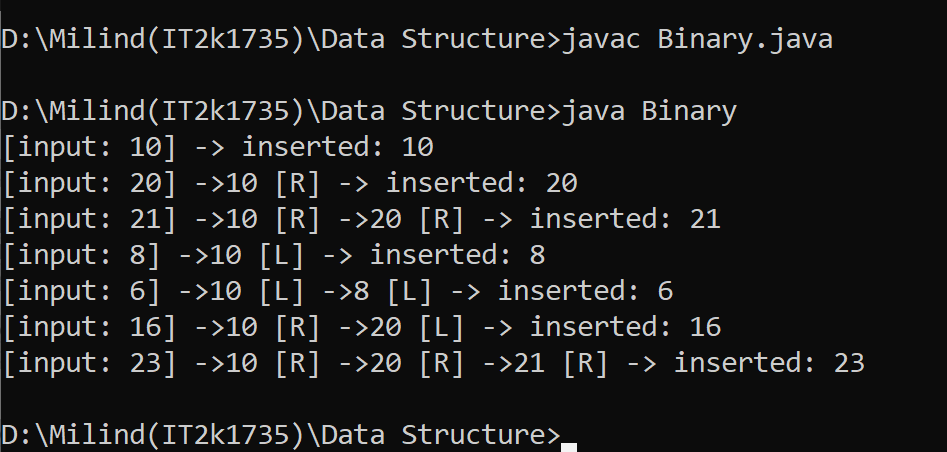
bst.insert(16);

bst.insert(23);

}

}

**Output 7:-**

****

**8.Program of Hash Table in Java.**

**Solution:-**

import java.util.\*;

class Hash {

public static void main(String args[])

{

Hashtable<Integer, String>

hm = new Hashtable<Integer, String>();

hm.put(1, "marvel");

hm.put(84, "hulk");

hm.put(48, "thor");

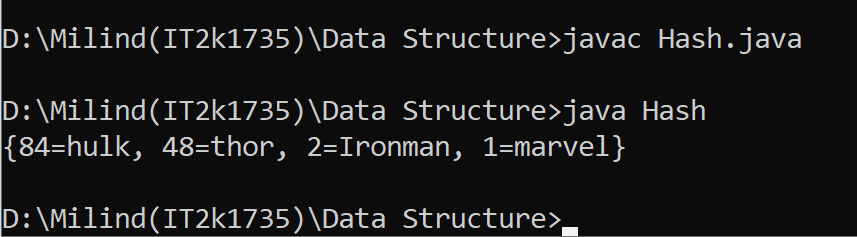
hm.put(2, "Ironman");

System.out.println(hm);

}

}

**Output 8:-**

****

**9.Program of Heap in Java.**

**Solution:-**

public class MinHeap {

private int[] Heap;

private int size;

private int maxsize;

private static final int FRONT = 1;

public MinHeap(int maxsize)

{

this.maxsize = maxsize;

this.size = 0;

Heap = new int[this.maxsize + 1];

Heap[0] = Integer.MIN\_VALUE;

}

private int parent(int pos)

{

return pos / 2;

}

private int leftChild(int pos)

{

return (2 \* pos);

}

private int rightChild(int pos)

{

return (2 \* pos) + 1;

}

private boolean isLeaf(int pos)

{

if (pos >= (size / 2) && pos <= size) {

return true;

}

return false;

}

private void swap(int fpos, int spos)

{

int tmp;

tmp = Heap[fpos];

Heap[fpos] = Heap[spos];

Heap[spos] = tmp;

}

private void minHeapify(int pos)

{

if (!isLeaf(pos)) {

if (Heap[pos] > Heap[leftChild(pos)]

|| Heap[pos] > Heap[rightChild(pos)]) {

if (Heap[leftChild(pos)] < Heap[rightChild(pos)]) {

swap(pos, leftChild(pos));

minHeapify(leftChild(pos));

}

else {

swap(pos, rightChild(pos));

minHeapify(rightChild(pos));

}

}

}

}

public void insert(int element)

{

if (size >= maxsize) {

return;

}

Heap[++size] = element;

int current = size;

while (Heap[current] < Heap[parent(current)]) {

swap(current, parent(current));

current = parent(current);

}

}

public void print()

{

for (int i = 1; i <= size / 2; i++) {

System.out.print(" PARENT : " + Heap[i]

+ " LEFT CHILD : " + Heap[2 \* i]

+ " RIGHT CHILD :" + Heap[2 \* i + 1]);

System.out.println();

}

}

public void minHeap()

{

for (int pos = (size / 2); pos >= 1; pos--) {

minHeapify(pos);

}

}

public int remove()

{

int popped = Heap[FRONT];

Heap[FRONT] = Heap[size--];

minHeapify(FRONT);

return popped;

}

public static void main(String[] arg)

{

System.out.println("The Min Heap is ");

MinHeap minHeap = new MinHeap(15);

minHeap.insert(5);

minHeap.insert(3);

minHeap.insert(17);

minHeap.insert(10);

minHeap.insert(84);

minHeap.insert(19);

minHeap.insert(6);

minHeap.insert(22);

minHeap.insert(9);

minHeap.minHeap();

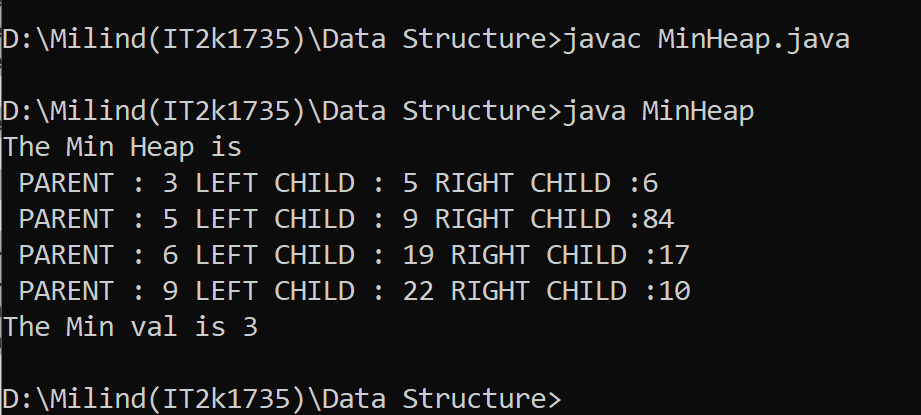
minHeap.print();

System.out.println("The Min val is " + minHeap.remove());

}

}

**Output 9:-**

****

**10.Program of Graph in Java.**

**Solution:-**

import java.util.\*;

class Graph<T> {

private Map<T, List<T> > map = new HashMap<>();

public void addVertex(T s)

{

map.put(s, new LinkedList<T>());

}

public void addEdge(T source,

T destination,

boolean bidirectional)

{

if (!map.containsKey(source))

addVertex(source);

if (!map.containsKey(destination))

addVertex(destination);

map.get(source).add(destination);

if (bidirectional == true) {

map.get(destination).add(source);

}

}

public void getVertexCount()

{

System.out.println("The graph has "

+ map.keySet().size()

+ " vertex");

}

public void getEdgesCount(boolean bidirection)

{

int count = 0;

for (T v : map.keySet()) {

count += map.get(v).size();

}

if (bidirection == true) {

count = count / 2;

}

System.out.println("The graph has “ + count

+ " edges.");

} public void hasVertex(T s)

{

if (map.containsKey(s)) {

System.out.println("The graph contains "

+ s + " as a vertex.");

}

else {

System.out.println("The graph does not contain "

+ s + " as a vertex.");

}

}

public void hasEdge(T s, T d)

{

if (map.get(s).contains(d)) {

System.out.println("The graph has an edge between "

+ s + " and " + d + ".");

}

else {

System.out.println("The graph has no edge between "

+ s + " and " + d + ".");

}

}

@Override

public String toString()

{

StringBuilder builder = new StringBuilder();

for (T v : map.keySet()) {

builder.append(v.toString() + ": ");

for (T w : map.get(v)) {

builder.append(w.toString() + " ");

}

builder.append("\n");

}

return (builder.toString());

}

}

public class Main {

public static void main(String args[])

{ Graph<Integer> g = new Graph<Integer>();

g.addEdge(0, 1, true);

g.addEdge(0, 4, true);

g.addEdge(1, 2, true);

g.addEdge(1, 3, true);

g.addEdge(1, 4, true);

g.addEdge(2, 3, true);

g.addEdge(3, 4, true);

System.out.println("Graph:\n"

+ g.toString());

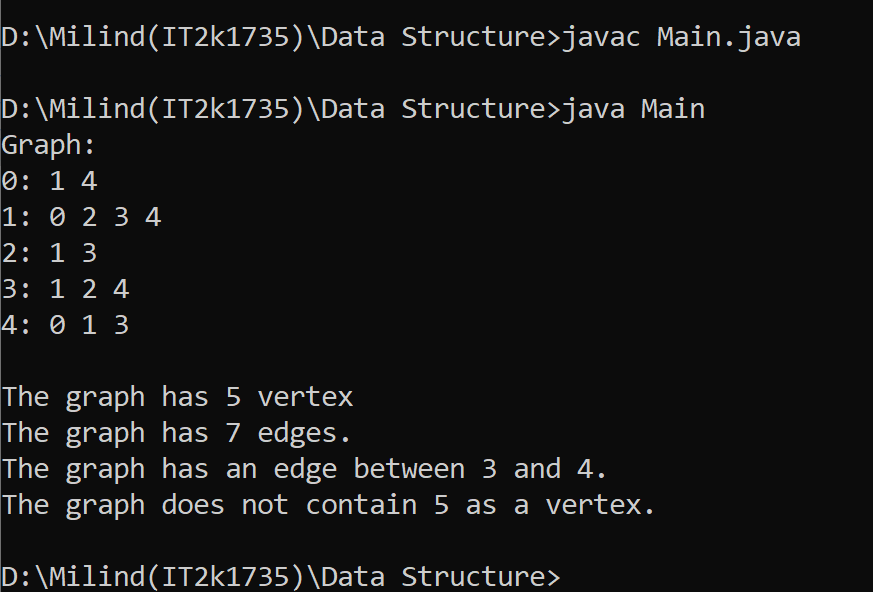
g.getVertexCount();

g.getEdgesCount(true);

g.hasEdge(3, 4);

g.hasVertex(5); } }

**Output 10:-**

****

**END**