# CSE (DATA SCIENCE)

## B.Tech III Semester L/T/P/C

**0 /0/ 2/ 1**

**INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING & DATA STRUCTURES USING JAVA LAB (CESOP2)**

**Course Objective:**

To implement Object Oriented Data Structures concepts using Java programming in real time applications.

## Course Outcomes:

After completion of course the student will be able to

1. Write the Java programs to implement object-oriented concepts.
2. Implement the Data Structures using Java programming.
3. Understand the usage of Java Collections.

## List of Experiments to be performed during the Course:

**Week 1:**

**Aim:** Try debug step by step with a small program of about 10 to 15 lines which contains at least one if else condition and a for loop.

**Source Code: Testp.java**

import java.util.\*;

public class Testp

{

public static void main(String[] args)

{

System.out.println("Welcome to TKRCET");

System.out.println("--------Prime Number--------");

Scanner sc = new Scanner(System.in);

System.out.println("Enter valid Number");

int n = sc.nextInt();

int c = 0;

for (int i = 1; i <= n; i++)

{

if (n % i == 0)

{

c++;

}

}

if (c == 2)

{

System.out.println(n + " is Prime Number");

}

else

{

System.out.println(n + " is not Prime Number");

} } }

**Output:**

D:\JavaLabPrgms>javac Testp.java

D:\JavaLabPrgms>java Testp

Welcome to TKRCET

--------Prime Number--------

Enter valid Number

45

45 is not Prime Number

**Week 2:**

**Aim:** Write a Java program to create an abstract class named Shape that contains two integers and an empty method named print Area (). Provide three classes named Rectangle, Triangle, and Circle such that each one of the classes extends the class Shape. Each one of the classes. contains only the method print Area () that prints the area of the given shape.

**Source Code : AbstractClass.java**

abstract class shape {

public int x, y;

public abstract void printArea();

}

class Rectangle extends shape {

public void printArea() {

System.out.println("Area of Rectangle is " + x \* y);

}

}

class Triangle extends shape {

public void printArea() {

System.out.println("Area of Triangle is " + (x \* y) / 2);

}

}

class Circle extends shape {

public void printArea() {

System.out.println("Area of Circle is " + (22 \* x \* x) / 7);

}

}

public class AbstractClass {

public static void main(String[] args) {

// TODO code application logic here

Rectangle r = new Rectangle();

r.x = 10;

r.y = 20;

r.printArea();

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

Triangle t = new Triangle();

t.x = 30;

t.y = 35;

t.printArea();

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

Circle c = new Circle();

c.x = 2;

c.printArea();

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

}

}

**Output:**

D:\JavaLabPrgms>javac AbstractClass.java

D:\JavaLabPrgms>java AbstractClass

Area of Rectangle is 200

-------------------------------------

Area of Triangle is 525

-------------------------------------

Area of Circle is 12

-------------------------------------

**Week 3:**

**Aim:** Write a Java program that demonstrate division by zero error exception handling.

**Source Code : MultipleExceptionExample.java**

public class MultipleExceptionExample {

public static void main(String args[]) {

try {

int a[] = new int[5];

a[5] = 30/0;

}

catch(ArithmeticException e){System.out.println("task1 is completed"+e);}

catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}

catch(Exception e){System.out.println("common task completed");}

System.out.println("rest of the code...");

}

}

**Output:**

D:\JavaLabPrgms>javac MultipleExceptionExample .java

D:\JavaLabPrgms>java MultipleExceptionExample

task1 is completedjava.lang.ArithmeticException: / by zero

rest of the code...

**Week 4:**

**Aim:** Write a Java program to demonstrate recursions (ex: Factorial, X power N etc…).

**Source Code : RecursionExampleFact.java**

import java.util.\*;

public class RecursionExampleFact{

static int factorial(int n){

if (n == 1)

return 1;

else

return(n \* factorial(n-1));

}

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

System.out.println("Enter n value: ");

int n=sc.nextInt();

System.out.println("Factorial of "+n+" is: "+factorial(n));

}

}

**Output:**

D:\JavaLabPrgms>javac RecursionExampleFact .java

D:\JavaLabPrgms>java RecursionExampleFact

Enter n value: 6

Factorial of 6 is: 720

**Week 5:**

**Aim:** Write a Java program to demonstrate all Stack operations.

### SourceCode: MyStack.java

/\*\*

\* Stack implementation using array

\* \*/

public class MyStack

{

    int size;

    int arr[];

    int top;

MyStack(int size)

{

        this.size = size;

        this.arr = new int[size];

        this.top = -1;

     }

     public void push(int element)

{

        if (!isFull())

{

            top++;

            arr[top] = element;

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

         }

else

{

            System.out.println("Stack is full !");

         }

     }

public int pop()

{

        if (!isEmpty())

{

            int topElement = top;

            top--;

            System.out.println("Popped element :" + arr[topElement]);

            return arr[topElement];

         }

else

{

            System.out.println("Stack is empty !");

            return -1;

         }

    }

public int peek()

{

        if(!this.isEmpty())

            return arr[top];

        else

        {

            System.out.println("Stack is Empty");

            return -1;

        }

     }

public boolean isEmpty()

{

        return (top == -1);

     }

public boolean isFull()

{

        return (size - 1 == top);

}

public static void main(String[] args)

{

        MyStack myStack = new MyStack(5);

        myStack.pop();

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

        myStack.push(100);

        myStack.push(90);

        myStack.push(10);

        myStack.push(50);

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

        myStack.pop();

        myStack.pop();

        myStack.pop();

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

    }

}

**Output:**

D:\JavaLabPrgms>javac MyStack.java

D:\JavaLabPrgms>java MyStack

Stack is empty !  
=================  
Pushed element:100  
Pushed element:90  
Pushed element:10  
Pushed element:50  
=================  
Popped element :50  
Popped element :10  
Popped element :90  
=================

**Week 6:**

**Aim:** Write a Java program to demonstrate all Queue operations .

**Source Code : MainQueue.java**

//Queue Implementation using an array:

// A class to represent a queue

class Queue

{

    private int[] arr;      // array to store queue elements

    private int front;      // front points to the front element in the queue

    private int rear;       // rear points to the last element in the queue

    private int capacity;   // maximum capacity of the queue

    private int count;      // current size of the queue

    // Constructor to initialize a queue

    Queue(int size)

    {

        arr = new int[size];

        capacity = size;

        front = 0;

        rear = -1;

        count = 0;

    }

    // Utility function to dequeue the front element

    public void dequeue()

    {

        // check for queue underflow

        if (isEmpty())

        {

            System.out.println("Underflow\nProgram Terminated");

            System.exit(1);

        }

        System.out.println("Removing " + arr[front]);

        front = (front + 1) % capacity;

        count--;

    }

    // Utility function to add an item to the queue

    public void enqueue(int item)

    {

        // check for queue overflow

        if (isFull())

        {

            System.out.println("Overflow\nProgram Terminated");

            System.exit(1);

        }

        System.out.println("Inserting " + item);

        rear = (rear + 1) % capacity;

        arr[rear] = item;

        count++;

    }

    // Utility function to return the front element of the queue

    public int peek()

    {

        if (isEmpty())

        {

            System.out.println("Underflow\nProgram Terminated");

            System.exit(1);

        }

        return arr[front];

    }

    // Utility function to return the size of the queue

    public int size() {

        return count;

    }

    // Utility function to check if the queue is empty or not

    public Boolean isEmpty() {

        return (size() == 0);

    }

    // Utility function to check if the queue is full or not

    public Boolean isFull() {

        return (size() == capacity);

    }

}

class MainQueue

{

    public static void main (String[] args)

    {

        // create a queue of capacity 5

        Queue q = new Queue(5);

        q.enqueue(1);

        q.enqueue(2);

        q.enqueue(3);

        System.out.println("The front element is " + q.peek());

        q.dequeue();

        System.out.println("The front element is " + q.peek());

        System.out.println("The queue size is " + q.size());

        q.dequeue();

        q.dequeue();

        if (q.isEmpty()) {

            System.out.println("The queue is empty");

        }

        else {

            System.out.println("The queue is not empty");

        }

    }

}

**Output :**

Inserting 1  
Inserting 2  
Inserting 3  
The front element is 1  
Removing 1  
The front element is 2  
The queue size is 2  
Removing 2  
Removing 3  
The queue is empty

**Week 7:**

**Aim:** Write a Java program to Demonstrate all DeQueue operations.

**Source Code : DequeueOperations.java**

import java.util.\*;

public class DequeueOperations {

public static void main(String[] args) {

//Declare Deque object

Deque<String> deque = new LinkedList<String>();

// add elements to the queue using various methods

deque.add("One"); //add ()

deque.addFirst("Two"); //addFirst ()

deque.addLast("Three"); //addLast ()

deque.push("Four"); //push ()

deque.offer("Five"); //offer ()

deque.offerFirst("Six"); //offerFirst ()

deque.offerLast("Seven"); //offerLast ()

System.out.println("Initial Deque:");

System.out.print(deque + " ");

// Iterate using standard iterator

System.out.println("\n\nDeque contents using Standard Iterator:");

Iterator iterator = deque.iterator();

while (iterator.hasNext())

System.out.print(" " + iterator.next());

// Iterate using Reverse order iterator

Iterator reverse = deque.descendingIterator();

System.out.println("\n\nDeque contents using Reverse Iterator:");

while (reverse.hasNext())

System.out.print(" " + reverse.next());

// Peek () method

System.out.println("\n\nDeque Peek:" + deque.peek());

System.out.println("\nDeque,After peek:" + deque);

// Pop () method

System.out.println("\nDeque Pop:" + deque.pop());

System.out.println("\nDeque,After pop:" + deque);

// contains () method

System.out.println("\nDeque Contains Three: " + deque.contains("Three"));

deque.removeFirst(); //removeFirst ()

deque.removeLast(); //removeLast ()

System.out.println("\nDeque, after removing " + "first and last elements: " + deque);

}

}

**Output :**

Initial Deque:

[Six, Four, Two, One, Three, Five, Seven]

Deque contents using Standard Iterator:

Six Four Two One Three Five Seven

Deque contents using Reverse Iterator:

Seven Five Three One Two Four Six

Deque Peek:Six

Deque,After peek:[Six, Four, Two, One, Three, Five, Seven]

Deque Pop:Six

Deque,After pop:[Four, Two, One, Three, Five, Seven]

Deque Contains Three: true

Deque, after removing first and last elements: [Two, One, Three, Five]

**Week 8:**

**Aim:** Write a Java program to Demonstrate all Singly Linked List operations.

**Source Code : LinkedList .java**

import java.io.\*;

// Java program to implement

// a Singly Linked List

public class LinkedList {

Node head; // head of list

// Linked list Node.

// Node is a static nested class

// so main() can access it

static class Node {

int data;

Node next;

// Constructor

Node(int d)

{

data = d;

next = null;

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*INSERTION\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method to insert a new node

public static LinkedList insert(LinkedList list,int data)

{

// Create a new node with given data

Node new\_node = new Node(data);

new\_node.next = null;

// If the Linked List is empty,

// then make the new node as head

if (list.head == null) {

list.head = new\_node;

}

else {

// Else traverse till the last node

// and insert the new\_node there

Node last = list.head;

while (last.next != null) {

last = last.next;

}

// Insert the new\_node at last node

last.next = new\_node;

}

// Return the list by head

return list;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*TRAVERSAL\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method to print the LinkedList.

public static void printList(LinkedList list)

{

Node currNode = list.head;

System.out.print("\nLinkedList: ");

// Traverse through the LinkedList

while (currNode != null) {

// Print the data at current node

System.out.print(currNode.data + " ");

// Go to next node

currNode = currNode.next;

}

System.out.println("\n");

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*DELETION BY KEY\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method to delete a node in the LinkedList by KEY

public static LinkedList deleteByKey(LinkedList list,int key)

{

// Store head node

Node currNode = list.head, prev = null;

//

// CASE 1:

// If head node itself holds the key to be deleted

if (currNode != null && currNode.data == key) {

list.head = currNode.next; // Changed head

// Display the message

System.out.println(key + " found and deleted");

// Return the updated List

return list;

}

//

// CASE 2:

// If the key is somewhere other than at head

//

// Search for the key to be deleted,

// keep track of the previous node

// as it is needed to change currNode.next

while (currNode != null && currNode.data != key) {

// If currNode does not hold key

// continue to next node

prev = currNode;

currNode = currNode.next;

}

// If the key was present, it should be at currNode

// Therefore the currNode shall not be null

if (currNode != null) {

// Since the key is at currNode

// Unlink currNode from linked list

prev.next = currNode.next;

// Display the message

System.out.println(key + " found and deleted");

}

//

// CASE 3: The key is not present

//

// If key was not present in linked list

// currNode should be null

if (currNode == null) {

// Display the message

System.out.println(key + " not found");

}

// return the List

return list;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*DELETION AT A POSITION\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method to delete a node in the LinkedList by POSITION

public static LinkedList deleteAtPosition(LinkedList list, int index)

{

// Store head node

Node currNode = list.head, prev = null;

//

// CASE 1:

// If index is 0, then head node itself is to be

// deleted

if (index == 0 && currNode != null) {

list.head = currNode.next; // Changed head

// Display the message

System.out.println(

index + " position element deleted");

// Return the updated List

return list;

}

//

// CASE 2:

// If the index is greater than 0 but less than the

// size of LinkedList

//

// The counter

int counter = 0;

// Count for the index to be deleted,

// keep track of the previous node

// as it is needed to change currNode.next

while (currNode != null) {

if (counter == index) {

// Since the currNode is the required

// position Unlink currNode from linked list

prev.next = currNode.next;

// Display the message

System.out.println(

index + " position element deleted");

break;

}

else {

// If current position is not the index

// continue to next node

prev = currNode;

currNode = currNode.next;

counter++;

}

}

// If the position element was found, it should be

// at currNode Therefore the currNode shall not be

// null

//

// CASE 3: The index is greater than the size of the LinkedList

//

// In this case, the currNode should be null

if (currNode == null) {

// Display the message

System.out.println(

index + " position element not found");

}

// return the List

return list;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*MAIN METHOD\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// method to create a Singly linked list with n nodes

public static void main(String[] args)

{

/\* Start with the empty list. \*/

LinkedList list = new LinkedList();

//

// \*\*\*\*\*\*INSERTION\*\*\*\*\*\*

//

// Insert the values

list = insert(list, 1);

list = insert(list, 2);

list = insert(list, 3);

list = insert(list, 4);

list = insert(list, 5);

list = insert(list, 6);

list = insert(list, 7);

list = insert(list, 8);

// Print the LinkedList

printList(list);

//

// \*\*\*\*\*\*DELETION BY KEY\*\*\*\*\*\*

//

// Delete node with value 1

// In this case the key is \*\*\*at head\*\*\*

deleteByKey(list, 1);

// Print the LinkedList

printList(list);

// Delete node with value 4

// In this case the key is present \*\*\*in the middle\*\*\*

deleteByKey(list, 4);

// Print the LinkedList

printList(list);

// Delete node with value 10

// In this case the key is \*\*\*not present\*\*\*

deleteByKey(list, 10);

// Print the LinkedList

printList(list);

// \*\*\*\*\*\*DELETION AT POSITION\*\*\*\*\*\*

// Delete node at position 0

// In this case the key is \*\*\*at head\*\*\*

deleteAtPosition(list, 0);

// Print the LinkedList

printList(list);

// Delete node at position 2

// In this case the key is present \*\*\*in the middle\*\*\*

deleteAtPosition(list, 2);

// Print the LinkedList

printList(list);

// Delete node at position 10

// In this case the key is \*\*\*not present\*\*\*

deleteAtPosition(list, 10);

// Print the LinkedList

printList(list);

}

}

**Output :**

LinkedList: 1 2 3 4 5 6 7 8

1 found and deleted

LinkedList: 2 3 4 5 6 7 8

4 found and deleted

LinkedList: 2 3 5 6 7 8

10 not found

LinkedList: 2 3 5 6 7 8

0 position element deleted

LinkedList: 3 5 6 7 8

2 position element deleted

LinkedList: 3 5 7 8

10 position element not found

LinkedList: 3 5 7 8

**Week 9:**

**Aim:** Write a Java program to Demonstrate all Doubly Linked List operations.

**Source Code: DoublyLinkedList.java**

import java.util.Scanner;

/\* Class Node \*/

class Node

{

protected int data;

protected Node next, prev;

/\* Constructor \*/

public Node()

{

next = null;

prev = null;

data = 0;

}

/\* Constructor \*/

public Node(int d, Node n, Node p)

{

data = d;

next = n;

prev = p;

}

/\* Function to set link to next node \*/

public void setLinkNext(Node n)

{

next = n;

}

/\* Function to set link to previous node \*/

public void setLinkPrev(Node p)

{

prev = p;

}

/\* Funtion to get link to next node \*/

public Node getLinkNext()

{

return next;

}

/\* Function to get link to previous node \*/

public Node getLinkPrev()

{

return prev;

}

/\* Function to set data to node \*/

public void setData(int d)

{

data = d;

}

/\* Function to get data from node \*/

public int getData()

{

return data;

}

}

/\* Class linkedList \*/

class linkedList

{

protected Node start;

protected Node end ;

public int size;

/\* Constructor \*/

public linkedList()

{

start = null;

end = null;

size = 0;

}

/\* Function to check if list is empty \*/

public boolean isEmpty()

{

return start == null;

}

/\* Function to get size of list \*/

public int getSize()

{

return size;

}

/\* Function to insert element at begining \*/

public void insertAtStart(int val)

{

Node nptr = new Node(val, null, null);

if(start == null)

{

start = nptr;

end = start;

}

else

{

start.setLinkPrev(nptr);

nptr.setLinkNext(start);

start = nptr;

}

size++;

}

/\* Function to insert element at end \*/

public void insertAtEnd(int val)

{

Node nptr = new Node(val, null, null);

if(start == null)

{

start = nptr;

end = start;

}

else

{

nptr.setLinkPrev(end);

end.setLinkNext(nptr);

end = nptr;

}

size++;

}

/\* Function to insert element at position \*/

public void insertAtPos(int val , int pos)

{

Node nptr = new Node(val, null, null);

if (pos == 1)

{

insertAtStart(val);

return;

}

Node ptr = start;

for (int i = 2; i <= size; i++)

{

if (i == pos)

{

Node tmp = ptr.getLinkNext();

ptr.setLinkNext(nptr);

nptr.setLinkPrev(ptr);

nptr.setLinkNext(tmp);

tmp.setLinkPrev(nptr);

}

ptr = ptr.getLinkNext();

}

size++ ;

}

/\* Function to delete node at position \*/

public void deleteAtPos(int pos)

{

if (pos == 1)

{

if (size == 1)

{

start = null;

end = null;

size = 0;

return;

}

start = start.getLinkNext();

start.setLinkPrev(null);

size--;

return ;

}

if (pos == size)

{

end = end.getLinkPrev();

end.setLinkNext(null);

size-- ;

}

Node ptr = start.getLinkNext();

for (int i = 2; i <= size; i++)

{

if (i == pos)

{

Node p = ptr.getLinkPrev();

Node n = ptr.getLinkNext();

p.setLinkNext(n);

n.setLinkPrev(p);

size-- ;

return;

}

ptr = ptr.getLinkNext();

}

}

/\* Function to display status of list \*/

public void display()

{

System.out.print("\nDoubly Linked List = ");

if (size == 0)

{

System.out.print("empty\n");

return;

}

if (start.getLinkNext() == null)

{

System.out.println(start.getData() );

return;

}

Node ptr = start;

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

ptr = start.getLinkNext();

while (ptr.getLinkNext() != null)

{

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

ptr = ptr.getLinkNext();

}

System.out.print(ptr.getData()+ "\n");

}

}

/\* Class DoublyLinkedList \*/

public class DoublyLinkedList

{

public static void main(String[] args)

{

Scanner scan = new Scanner(System.in);

/\* Creating object of linkedList \*/

linkedList list = new linkedList();

System.out.println("Doubly Linked List Test\n");

char ch;

/\* Perform list operations \*/

do

{

System.out.println("\nDoubly Linked List Operations\n");

System.out.println("1. insert at begining");

System.out.println("2. insert at end");

System.out.println("3. insert at position");

System.out.println("4. delete at position");

System.out.println("5. Display size and elements");

int choice = scan.nextInt();

switch (choice)

{

case 1 :

System.out.println("Enter integer element to insert");

list.insertAtStart( scan.nextInt() );

break;

case 2 :

System.out.println("Enter integer element to insert");

list.insertAtEnd( scan.nextInt() );

break;

case 3 :

System.out.println("Enter integer element to insert");

int num = scan.nextInt() ;

System.out.println("Enter position");

int pos = scan.nextInt() ;

if (pos < 1 || pos > list.getSize() )

System.out.println("Invalid position\n");

else

list.insertAtPos(num, pos);

break;

case 4 :

System.out.println("Enter position");

int p = scan.nextInt() ;

if (p < 1 || p > list.getSize() )

System.out.println("Invalid position\n");

else

list.deleteAtPos(p);

break;

case 5 :

System.out.println("Size = "+ list.getSize() +" \n");

break;

default :

System.out.println("Wrong Entry \n ");

break;

}

/\* Display List \*/

list.display();

System.out.println("\nDo you want to continue (Type y or n) \n");

ch = scan.next().charAt(0);

} while (ch == 'Y'|| ch == 'y');

}

}

**Output:**

D:\JavaLabPrgms >javac DoublyLinkedList.java

D:\JavaLabPrgms >java DoublyLinkedList

Doubly Linked List Test

Doubly Linked List Operations

1. insert at begining

2. insert at end

3. insert at position

4. delete at position

5. Display size and elements

1

Enter integer element to insert

10

Doubly Linked List = 10

Do you want to continue (Type y or n)

y

Doubly Linked List Operations

1. insert at begining

2. insert at end

3. insert at position

4. delete at position

5. Display size and elements

2

Enter integer element to insert

20

Doubly Linked List = 10 <-> 20

Do you want to continue (Type y or n)

n

**Week 10:**

**Aim:** Write a Java program to Demonstrate all Circularly Linked List operations.

// Java code to perform circular linked list operations

class CircularLinkedList {

static class Node {

int data;

Node next;

};

static Node addToEmpty(Node last, int data) {

if (last != null)

return last;

// allocate memory to the new node

Node newNode = new Node();

// assign data to the new node

newNode.data = data;

// assign last to newNode

last = newNode;

// create link to iteself

newNode.next = last;

return last;

}

// add node to the front

static Node addFront(Node last, int data) {

if (last == null)

return addToEmpty(last, data);

// allocate memory to the new node

Node newNode = new Node();

// add data to the node

newNode.data = data;

// store the address of the current first node in the newNode

newNode.next = last.next;

// make newNode as head

last.next = newNode;

return last;

}

// add node to the end

static Node addEnd(Node last, int data) {

if (last == null)

return addToEmpty(last, data);

// allocate memory to the new node

Node newNode = new Node();

// add data to the node

newNode.data = data;

// store the address of the head node to next of newNode

newNode.next = last.next;

// point the current last node to the newNode

last.next = newNode;

// make newNode as the last node

last = newNode;

return last;

}

static Node addAfter(Node last, int data, int item) {

if (last == null)

return null;

Node newNode, p;

p = last.next;

do {

// if the item is found, place newNode after it

if (p.data == item) {

// allocate memory to the new node

newNode = new Node();

// add data to the node

newNode.data = data;

// make the next of the current node as the next of newNode

newNode.next = p.next;

// put newNode to the next of p

p.next = newNode;

// if p is the last node, make newNode as the last node

if (p == last)

last = newNode;

return last;

}

p = p.next;

} while (p != last.next);

System.out.println(item + "The given node is not present in the list");

return last;

}

// delete a node

static Node deleteNode(Node last, int key) {

// if linked list is empty

if (last == null)

return null;

// if the list contains only a single node

if (last.data == key && last.next == last) {

last = null;

return last;

}

Node temp = last, d = new Node();

// if last is to be deleted

if (last.data == key) {

// find the node before the last node

while (temp.next != last) {

temp = temp.next;

}

// point temp node to the next of last i.e. first node

temp.next = last.next;

last = temp.next;

}

// travel to the node to be deleted

while (temp.next != last && temp.next.data != key) {

temp = temp.next;

}

// if node to be deleted was found

if (temp.next.data == key) {

d = temp.next;

temp.next = d.next;

}

return last;

}

static void traverse(Node last) {

Node p;

if (last == null) {

System.out.println("List is empty.");

return;

}

p = last.next;

do {

System.out.print(p.data + " ");

p = p.next;

}

while (p != last.next);

}

public static void main(String[] args) {

Node last = null;

last = addToEmpty(last, 6);

last = addEnd(last, 8);

last = addFront(last, 2);

last = addAfter(last, 10, 2);

traverse(last);

deleteNode(last, 8);

traverse(last);

}

}

**Week 11:**

**Aim:** Write a Java program to Demonstrate Java Collections.

**Source code: TestSort2.java**

import java.util.\*;

class TestSort2

{

public static void main(String args[])

{

ArrayList<String> al=new ArrayList<String>();

al.add("Viru");

al.add("Saurav");

al.add("Mukesh");

al.add("Tahir");

Iterator i=al.iterator();

while(i.hasNext())

{

System.out.println(i.next());

}

Collections.sort(al,Collections.reverseOrder());

Iterator j=al.iterator();

while(j.hasNext())

{

System.out.println(j.next());

}

}

}

**Output :**

ViruSauravMukesh

TahirViruTahir

Saurav

Mukesh

**Week 12:**

**Aim:** Write a Java program to demonstrate Linear Search, Binary Search and Bubble Sort algorithm.

**Linear Search**

**Source code:Leniear\_Search.java**

import java.util.Scanner;

public class Leniear\_Search

{

public static void main(String[] args)

{

int[] arr={10,23,15,8,4,3,25,30,34,2,19};

int item,flag=0;

Scanner sc=new Scanner(System.in);

System.out.println("Enter Item ?");

item=sc.nextInt();

for(int i=0;i<10;i++)

{

if(arr[i]==item)

{

flag=i+1;

break;

}

else

flag=0;

}

if(flag!=0)

{

System.out.println("Item found at location"+flag);

}

else

System.out.println("Item not found");

}

}

**OUTPUT :**

1. Enter Item ?

12

Item not found

1. Enter Item ?

25

Item found at location7

**Binary Search**

**Source code: BinarySearchExample.java**

class BinarySearchExample{

public static void binarySearch(int arr[], int first, int last, int key){

int mid = (first + last)/2;

while( first <= last ){

if ( arr[mid] < key ){

first = mid + 1;

}else if ( arr[mid] == key ){

System.out.println("Element is found at index: " + mid);

break;

}else{

last = mid - 1;

}

mid = (first + last)/2;

}

if ( first > last ){

System.out.println("Element is not found!");

}

}

public static void main(String args[]){

int arr[] = {10,20,30,40,50};

int key = 30;

int last=arr.length-1;

binarySearch(arr,0,last,key);

}

}

**OUTPUT :**

Element is found at index: 2

**BubbleSort**

**Source code: BubbleSortExample.java**

public class BubbleSortExample {

static void bubbleSort(int[] arr) {

int n = arr.length;

int temp = 0;

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

{

for(int j=1; j < (n-i); j++)

{

if(arr[j-1] > arr[j])

{

//swap elements

temp = arr[j-1];

arr[j-1] = arr[j];

arr[j] = temp;

}

}

}

}

public static void main(String[] args)

{

int arr[] ={3,60,35,2,45,320,5};

System.out.println("Array Before Bubble Sort");

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

{

System.out.print(arr[i] + " ");

}

System.out.println();

bubbleSort(arr);//sorting array elements using bubble sort

System.out.println("Array After Bubble Sort");

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

{

System.out.print(arr[i] + " ");

}

}

}

**OUTPUT :**

Array Before Bubble Sort

3 60 35 2 45 320 5

Array After Bubble Sort

2 3 5 35 45 60 320