JAVA PROGRAMMING LAB

**LAB 3**

**Submitted by:** Bharat kumar

**Enrollment no:** 12814802719

**Topic covered:** Arrays, Methods, Method Overloading

**LAB 3.1**

**Aim:** Write a program to find out the array index or position where sum of numbers preceding the index is equals to sum of numbers succeeding the index.

**Theory:**

Given, an array of size n. Find an element that divides the array into two sub-arrays with equal sums.  
Examples:

**Input:** 1 4 2 5

**Output:** 2

**Explanation:** If 2 is the partition, subarrays are : {1, 4} and {5}

**Source code:**

public class l3p1 {

public static void main(String[] args) {

int arr[]={1,2,3,4,5,6,7,2,9,10};

int sum=0;

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

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

sum+=arr[i];

}

int num=0;

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

num+=arr[i];

if(num==(sum-arr[i+1])/2){

System.out.print("\nIndex is: "+(i+1));

break;

}

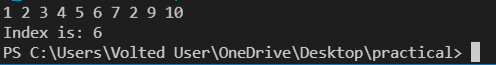
}

//10 9 10 29-9/2

}

}

**Output**



**LAB 3.2**

**Aim:** Write a program that creates and initializes a four-element int array. Calculate and display the average of its values.

**Theory:**

Enter size of array and then enter all the elements of that array. Now using for loop we calculate sum of elements of array and hence we divide it by number of elements in array to get average.

Here is the source code of the Java Program to Calculate Sum & Average of an Array. The Java program is successfully compiled and run on a Windows system. The program output is also shown below.

**Source code:**

import java.util.Scanner;

public class l3p2 {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

var arr=new int[4];

float avg;

System.out.println("Enter the array:");

for(int i=0;i<4;i++)

arr[i]=sc.nextInt();

sc.close();

float sum=0;

for(int i=0;i<4;i++)

sum+=arr[i];

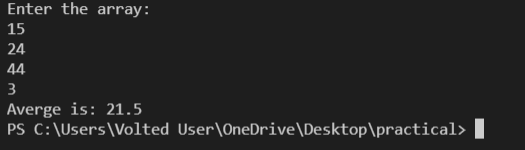
avg=sum/4;

System.out.print("Averge is: " +avg);

}

}

**Output:**



**LAB 3.3**

**Aim:** WAP using Bubble sort for sorting in ascending Order.

**Theory:**

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.  
Example:   
First Pass:   
( 5 1 4 2 8 ) –> ( 1 5 4 2 8 ), Here, algorithm compares the first two elements, and swaps since 5 > 1.   
( 1 5 4 2 8 ) –> ( 1 4 5 2 8 ), Swap since 5 > 4   
( 1 4 5 2 8 ) –> ( 1 4 2 5 8 ), Swap since 5 > 2   
( 1 4 2 5 8 ) –> ( 1 4 2 5 8 ), Now, since these elements are already in order (8 > 5), algorithm does not swap them.  
Second Pass:   
( 1 4 2 5 8 ) –> ( 1 4 2 5 8 )   
( 1 4 2 5 8 ) –> ( 1 2 4 5 8 ), Swap since 4 > 2   
( 1 2 4 5 8 ) –> ( 1 2 4 5 8 )   
( 1 2 4 5 8 ) –> ( 1 2 4 5 8 )   
Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one whole pass without any swap to know it is sorted.  
Third Pass:   
( 1 2 4 5 8 ) –> ( 1 2 4 5 8 )   
( 1 2 4 5 8 ) –> ( 1 2 4 5 8 )   
( 1 2 4 5 8 ) –> ( 1 2 4 5 8 )   
( 1 2 4 5 8 ) –> ( 1 2 4 5 8 )

**Source code:**

import java.util.Scanner;

public class l3p3 {

void bubbleSort(int[] arr,int n){

int temp;

for(int i=0;i<n-1;i++)

for(int j=0;j<n-i-1;j++)

if(arr[j]>arr[j+1]){

temp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

}

System.out.println("Sorted Array:");

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

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

}

public static void main(String[] args) {

l3p3 sort=new l3p3();

int n;

Scanner sc=new Scanner(System.in);

System.out.print("Enter the size of array ");

n=sc.nextInt();

var arr=new int[n];

System.out.println("Enter the array");

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

arr[i]=sc.nextInt();

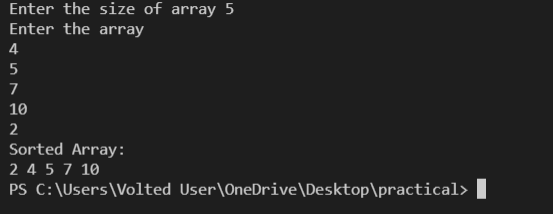
sc.close();

sort.bubbleSort(arr, n);

}

}

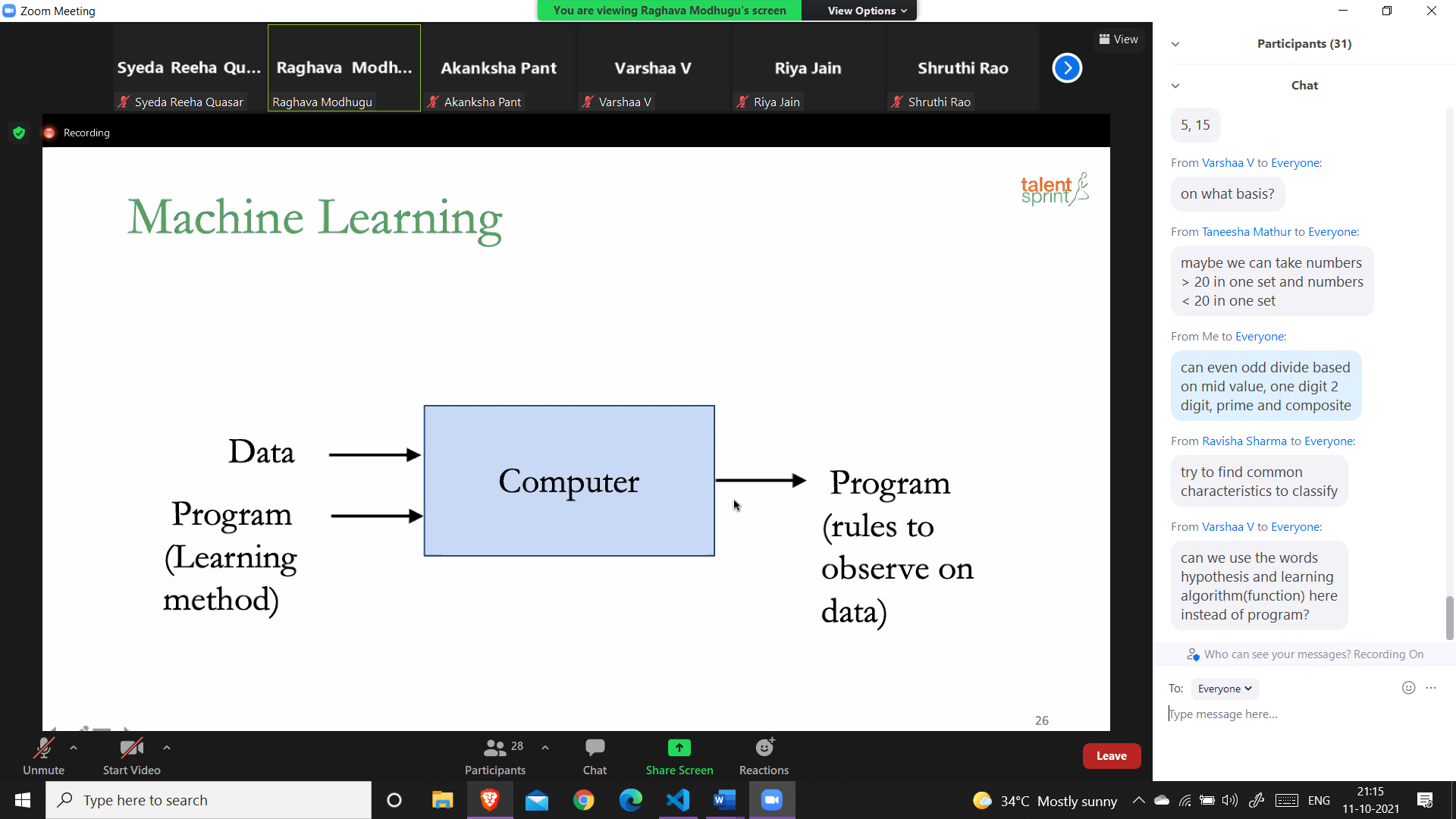
**Output:**



**LAB 3.4**

**Aim:** Create a java program to implement stack and queue concept

**Theory:**

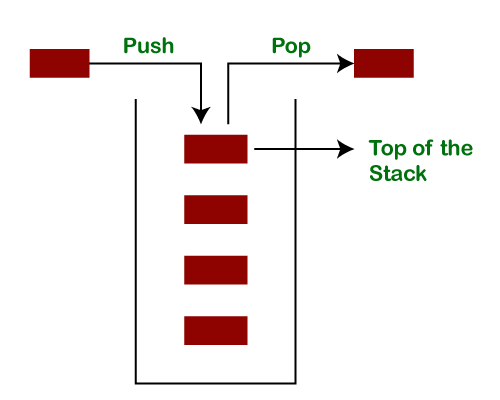


Java Stack

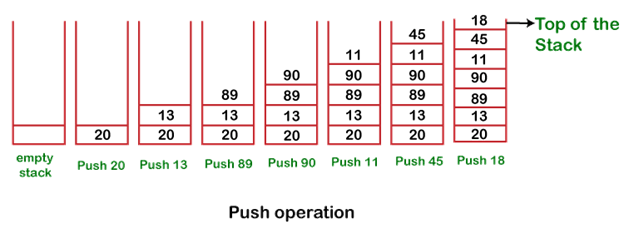
The stack is a linear data structure that is used to store the collection of objects. It is based on Last-In-First-Out (LIFO). [Java collection](https://www.javatpoint.com/collections-in-java) framework provides many interfaces and classes to store the collection of objects. One of them is the Stack class that provides different operations such as push, pop, search, etc.

In this section, we will discuss the Java Stack class, its methods, and implement the stack data structure in a [Java program](https://www.javatpoint.com/java-programs). But before moving to the Java Stack class have a quick view of how the stack works.

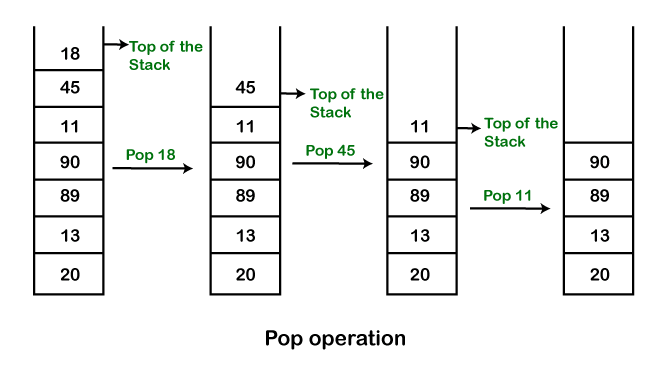
The stack data structure has the two most important operations that are push and pop. The push operation inserts an element into the stack and pop operation removes an element from the top of the stack. Let's see how they work on stack.



Let's push 20, 13, 89, 90, 11, 45, 18, respectively into the stack.



Let's remove (pop) 18, 45, and 11 from the stack.



# Queue Interface In Java

The Queue interface present in the [java.util](https://www.geeksforgeeks.org/java-util-package-java/) package and extends the [Collection interface](https://www.geeksforgeeks.org/collections-in-java-2/) is used to hold the elements about to be processed in FIFO(First In First Out) order. It is an ordered list of objects with its use limited to insert elements at the end of the list and deleting elements from the start of the list, (i.e.), it follows the FIFO or the First-In-First-Out principle.

Being an interface the queue needs a concrete class for the declaration and the most common classes are the [PriorityQueue](https://www.geeksforgeeks.org/priority-queue-class-in-java-2/) and [LinkedList](https://www.geeksforgeeks.org/linked-list-in-java/) in Java.It is to be noted that both the implementations are not thread safe. [PriorityBlockingQueue](https://www.geeksforgeeks.org/priorityblockingqueue-class-in-java/) is one alternative implementation if thread safe implementation is needed.

**Source code:**

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class stackUsingArrayList {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

List<Integer> arrayList=new ArrayList<>();

int val;

System.out.println("Enter your choice: (1-Stack, 2-Queue)");

val=sc.nextInt();

if(val==1){

System.out.println("Push 3 values for Stack");

for(int i=0;i<3;i++){

val=sc.nextInt();

arrayList.add(val);

}

System.out.println("Popping element from stack");

int valuePop=arrayList.get(arrayList.size()-1);

arrayList.remove(arrayList.size()-1);

System.out.println(valuePop+" popped from stack");

}

else{

System.out.println("Enter 3 values for Queue");

for(int i=0;i<3;i++){

val=sc.nextInt();

arrayList.add(val);

}

System.out.println("Removing element from queue");

int valuePop=arrayList.get(0);

arrayList.remove(0);

System.out.println(valuePop+" removed from queue");

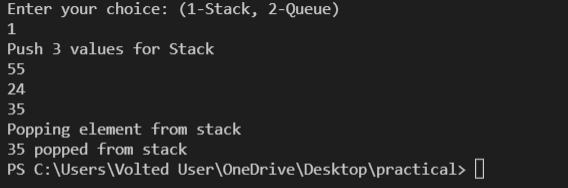
}

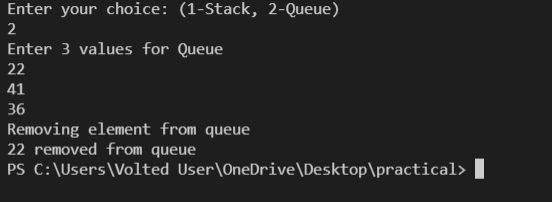
sc.close();

}

}

**Output:**





**LAB 3.5**

**Aim:** Using the concept of method overloading Write method for calculating the area of triangle ,circle and rectangle.

**Theory:**

This is a Java Program to Find Area of Square, Rectangle and Circle using Method Overloading.

We declare three methods of same name but with different number of arguments or with different data types. Now when we call these methods using objects, corresponding methods will be called as per the number of arguments or their datatypes.

Here is the source code of the Java Program to Find Area of Square, Rectangle and Circle using Method Overloading. The Java program is successfully compiled and run on a Windows system. The program output is also shown below.

In this program, we will see how to find the area of a square, rectangle, and circle using Method Overloading.

### Algorithm:

1. Start
2. Declare three different classes for rectangle, square, and circle.
3. Declare two methods of the same name but with a different number of arguments or with different data types.
4. Call these methods using objects.
5. Call the corresponding methods as per the number of arguments or their data types.
6. Display the result.
7. Stop.

**Source code:**

import java.util.Scanner;

public class p5 {

public float area(int a,int b) {

return (float)a\*b; }

public float area(float r) {

return (22\*r\*r/7); }

public float area(float h,float b) {

return h\*b/2; }

public static void main(String[] args) {

p5 ar=new p5();

Scanner sc=new Scanner(System.in);

int ch;

System.out.println("Enter choice of area (1-Triangle , 2-Circle , 3-Rectangle) ");

ch=sc.nextInt();

switch(ch){

case 1:float h,b;

System.out.print("Enter height and base of triangle: ");

h=sc.nextFloat();

b=sc.nextFloat();

System.out.println("Area of Traingle: "+ar.area(h,b));

break;

case 2:float r;

System.out.print("Enter radius of circle: ");

r=sc.nextFloat();

System.out.println("Area of Circle: "+ar.area(r));

break;

case 3:int a,c;

System.out.print("Enter length and breadth of triangle: ");

a=sc.nextInt();

c=sc.nextInt();

System.out.println("Area of Rectangle: "+ar.area(a,c));

break;

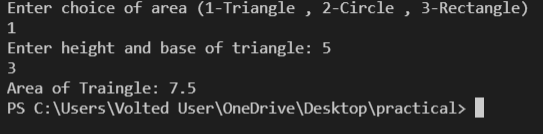
}

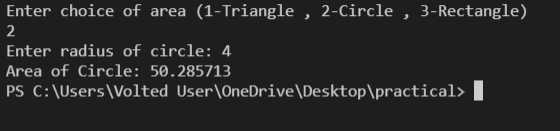
sc.close();

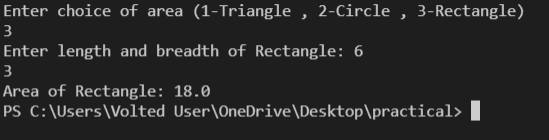
}

}

**Output:**







VIVA VOCE QUESTION

1. What is difference between Path and Class path?

Ans. Path and Class path are operating system level environment variales. Path is used define where the system can find the executables(.exe) files and class path is used to specify the location .class files.

2. What are local variables?

Ans. Local variables are those which are declared within a block of code like methods. Local variables should be initialized before accessing them

3. Can a class declared as private be accessed outside its package?

Ans. Not possible.

4. Can a class be declared as protected?

Ans. A class can't be declared as protected. only methods can be declared as protected.

5. What is the access scope of a protected method?

Ans. A protected method can be accessed by the classes within the same package or by the subclasses of the class in any package.