Class: D10A Seat No.____

Roll No. 01

VIVEKANAND EDUCATION SOCIETY'S INSTITUTE OF TECHNOLOGY

Hashu Advani Memorial Complex, Collector's Colony, R C Marg, Chembur, Mumbai-400074

CERTIFICATE

Certified that Mr./Miss **Aamir Z Ansari** of **D10A** has satisfactorily completed a course of the necessary experiments/assignments in **Java Lab** under my supervision in the Institute of technology in the year 2020 - 2021

Principal Head of Department

Lab In-charge Subject Teacher

Vivekanand Education Society's Institute Of Technology
Department Of Information Technology
2020-2021

Name of the Course: Java Lab
Year/Sem/Class: S.E.(INFT) Sem III (D10A)
Faculty In charge : Mrs. Dimple Bohra
Email: dimple.bohra@ves.ac.in

Sr. No.	Lab Experiments
1	Implement JAVA programs based on the concept of classes, objects and control structures.
2	Implement Java programs to illustrate the concept of classes, objects, constructor, method overloading and array of objects.
3	Implement the concept of recursion
4	Implementation of Inheritance
5	Program based on usage of Final variable
6	Program to illustrate the concept of Interface
7	Lab assignment based on packages
8	Java program to create user defined exception.
9	Java program to demonstrate the concept of multithreading and synchronization
10	Java program to implement AWT components
11	Java program to illustrate the usage of built in layout manager
12	Java program to implement swing components
	2 Programming Assignments

JAVA Lab

Lab Experiment number 1

Roll no. **01** Batch **A**

Aim:

Implement JAVA programs based on the concept of classes, objects and control structures

Theory:

Classes and Objects are basic concepts of Object Oriented Programming which revolve around the real life entities

=> Class :-

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

1. Modifiers : A class can be public or has default access
2. Class name: The name should begin with an initial letter (capitalized by convention).
3. Superclass: The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
4. Interfaces(if any): A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
5. Body: The class body surrounded by braces, { }.

=>Objects:-

It is a basic unit of Object Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

1. State : It is represented by attributes of an object. It also reflects the properties of an object.
2. Behavior : It is represented by methods of an object. It also reflects the response of an object with other objects.
3. Identity : It gives a unique name to an object and enables one object to interact with other objects.

=>Control statements:

Simple if statement

The if statement determines whether a code should be executed based on the specified condition.

```
if (condition) {
Statement 1; //executed if condition is true
}
Statement 2; //executed irrespective of the condition
```

Output:

If statement!

Hello World!

If..else statement

In this statement, if the condition specified is true, the if block is executed. Otherwise, the else block is executed.

Example:

```
public class Main
{
public static void main(String args[])
{
int a = 15;
if (a > 20)
System.out.println("a is greater than 10");
else
System.out.println("a is less than 10");
System.out.println("Hello World!");
}
}
}
```

Output:

a is less than 10

Hello World!

***

Ladder of If..else

```
if (score >= 90)
    grade = ‘A’;
else if (score >= 80)
    grade = ‘B’;
else if (score >= 70)
```

```
    grade = 'C';
 else if (score >= 60)
    grade = 'D';
 else
    grade = 'F';
```

***

Nested if statement

- An if present inside an if block is known as a nested if block. It is similar to an if..else statement, except they are defined inside another if..else statement.

```
if (condition1) {
        Statement 1; //executed if first condition is true
if (condition2) {
        Statement 2; //executed if second condition is true
}
else {
        Statement 3; //executed if second condition is false }}
```

Switch Statements

1. The switch-expression must yield a value of char, byte, short, or int type and must always be enclosed in parentheses.
2. The keyword break is optional, but it should be used at the end of each case in order to terminate the remainder of the switch statement.
3. If the break statement is not present, the next case statement will be executed.
4. The default case, which is optional, can be used to perform actions when none of the specified cases is true.
5. The order of the cases (including the default case) does not matter. However, it is a good programming style to follow the logical sequence of the cases and place the default case at the end.

```
switch (year) {
        case 7: annualInterestRate = 7.25;
                break;
        case 15: annualInterestRate = 8.50;
                break;
```

```
        case 30: annualInterestRate = 9.0;
                 break;
        default: System.out.println("Wrong number of years, enter 7, 15, or 30");
}
```

***

=>Iteration Statement

Statements that execute a block of code repeatedly until a specified condition is met are known as Iteration or looping statements.

Java provides the user with three types of looping statements:

1. while statement
2. do-while statement
3. For statement

while Loop

Known as the most common loop, the while loop evaluates a certain condition. If the condition is true, the code is executed. This process is continued until the specified condition turns out to be false.

```
int i = 0;
while (i < 100) {
        System.out.println("Welcome to Java!");
        i++;
}
```

```
int i = 5;
while (i <= 15) {
        System.out.print(i);
        i = i+2;
}
```

Output: 5 7 9 11 13 15

do-while loop

The do-while loop is similar to the while loop, the only difference being that the condition in the do-while loop is evaluated after the execution of the loop body. This guarantees that the loop is executed at least once.

```
do { // Loop body;}
 while (continue-condition);
```

Eg.

```
public class Main {
        public static void main(String args[]) {
                int i = 1;
                do {
                        System.out.println(i);
                        i = i+1;
                } while (i <= 5);
        }
}
```

Output : 12345

For statement

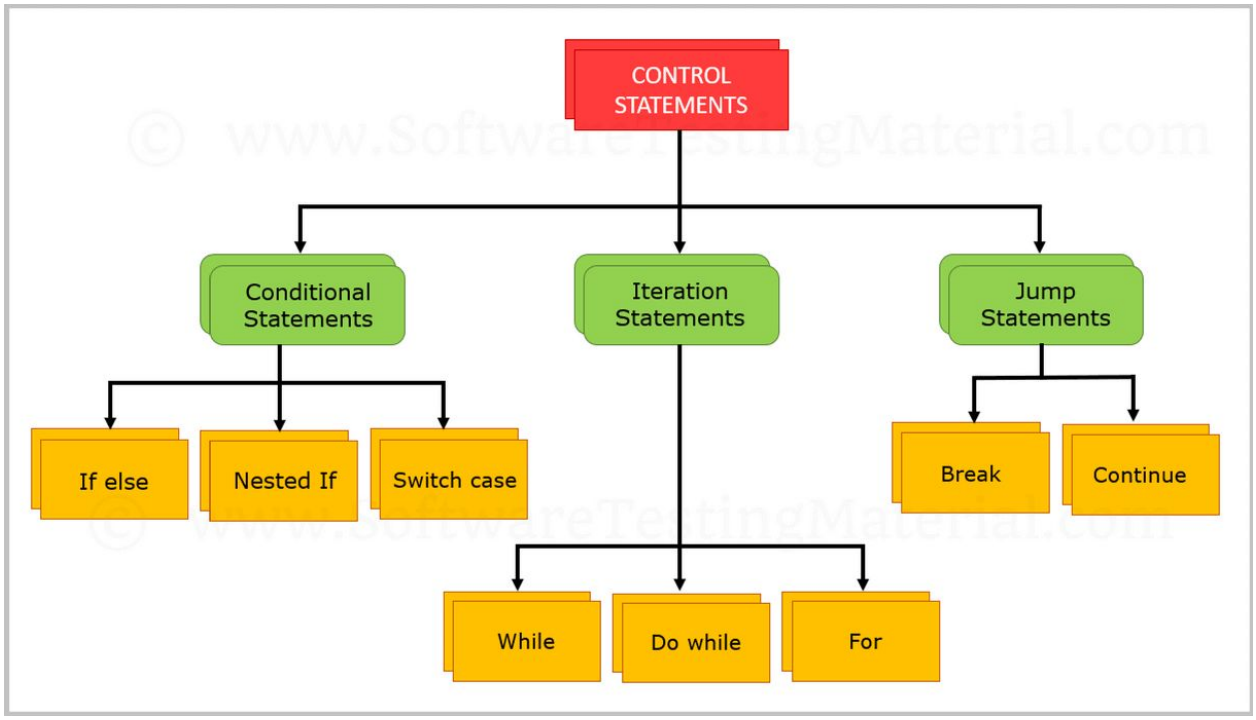
The for loop in java is used to iterate and evaluate a code multiple times. When the number of iterations is known by the user, it is recommended to use the for

loop.
Syntax:

```
for (initialization ; condition ; increment/decrement) {
        Statements;
}
```

Example

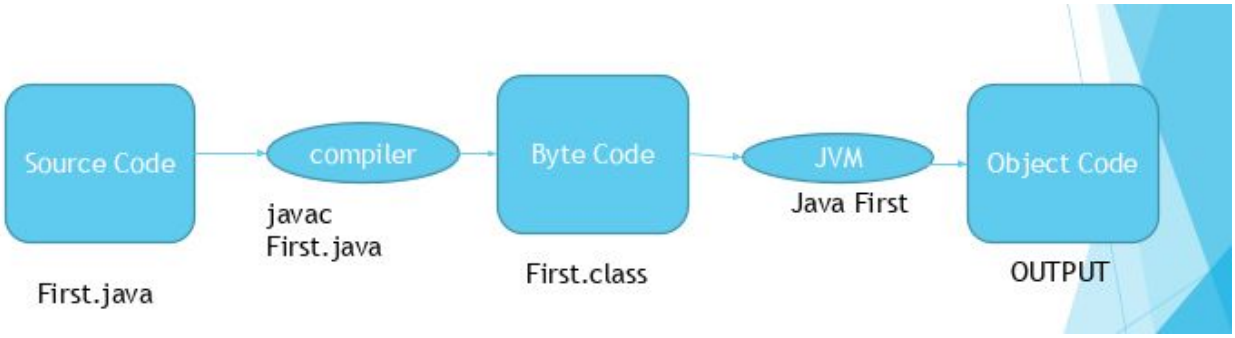
```
int i;
for (i = 0; i<100; i++) {
        System.out.println("Welcome to Java");
}
```



***

=>Compilation and Execution of Programs

First we create a word file with any text editor, type the java program and save the file as all files, with extension of .java.
Using command prompt, locate the file.
Use command “javac <file-name.java> to compile
After successful compilation, class file is created
Use command “java <file-name>” to run the java program



***

Programs:

Five Bikers Compete in a race such that they drive at a constant speed which may or may not be the same as the other. To qualify the race, the speed of a racer must be more than the average speed of all 5 racers. Write a Java program to take as input the speed of each racer and print back the speed of qualifying racers. (make use of array)

//code

```
import java.util.*;
class Race {
	public static void main(String args[]) {
		Scanner sc = new Scanner(System.in);
		float speed[], qualified[];
		speed = new float[10];
		qualified = new float[10];
		float avg=0;
		int k=0;
		//input
		System.out.println("Enter speed of bikers :  ");
		for(int i=0 ; i<5 ; i++) {
			speed[i] = sc.nextFloat();
		}
		//find average;
		for (int i=0 ; i<5 ; i++) {
			avg += speed[i];
		}
		avg = avg/5;
		//check qualified
		for (int i=0 ; i<5 ; i++) {
			if (speed[i] > avg) {
				qualified[k++] = speed[i];
			}
		}
		//printing qulified speed
		System.out.println();
```

```
                System.out.println("Qualified bikers with speed are :  ");
                for(int i=0 ; i<qualified.length && qualified[i]!=0 ; i++) {
                        System.out.print(" "+qualified[i]+" ");
                }
        }
}
```

//output

```
E:\Aamir\Sem-3\OOPM\Lab Assignment 1>java Race
Enter speed of bikers:
18.2  20  12.3  17  10.2

Qualified bikers with speed are :
 18.2  20.0  17.0
E:\Aamir\Sem-3\OOPM\Lab Assignment 1>
```

****

```
E:\Aamir\Sem-3\OOPM\Lab Assignment 1>java Race
Enter speed of bikers:
4  12  14.2  10  8

Qualified bikers with speed are :
 12.0  14.2  10.0
E:\Aamir\Sem-3\OOPM\Lab Assignment 1>
```

****

Implement a java program to calculate gross salary & net salary taking the following data. Input: empno, empname, basic process: DA=70% of basic, HRA=30% of basic, CCA=Rs240/-, PF=10% of basic, PT= Rs100/-

//code

```
import java.util.*;

class Salary {
```

```
        public static void main(String args[]) {
                Scanner sc = new Scanner(System.in);
                float basic, da, hra, gross, cca=240, pf, pt=100, deduction, net;
                System.out.println(“Enter employee number :  “);
                int empno = sc.nextInt();
                sc.nextLine();
                System.out.println(“Enter employee name :  “);
                String empname = sc.nextLine();
                System.out.println(“Enter employee’s basic income :  “);
                basic = sc.nextFloat();
                da = 70*basic/100;
                hra = 30*basic/100;
                pf = 10*basic/100;
                gross = basic+da+hra+cca;
                deduction = pf+pt;
                net = gross - deduction;
                System.out.println("Employee with  Employee number: "+empno+"** Name:
"+empname+" ** With basic salary of "+basic+" has");
                System.out.println("Gross Salary  :  "+gross);
                System.out.println("Net Salary  :  "+net);
        }
}



//output
```

```
E:\Aamir\Sem-3\OOPM\Lab Assignment 1>javac Salary.java

E:\Aamir\Sem-3\OOPM\Lab Assignment 1>java Salary
Enter employee number :
1
Enter employee name :
Aamir
Enter employee's basic income :
55000
Employee with  Employee number: 1** Name: Aamir ** With basic salary of 55000.0 has
Gross Salary  :  110240.0
Net Salary  :  104640.0

E:\Aamir\Sem-3\OOPM\Lab Assignment 1>java Salary
Enter employee number :
62
Enter employee name :
Krishna
Enter employee's basic income :
70000
Employee with  Employee number: 62** Name: Krishna ** With basic salary of 70000.0 has
Gross Salary  :  140240.0
Net Salary  :  133140.0
```

Thank you

JAVA Lab

Lab experiment number 2

Name: Aamir Ansari **Roll no.** 01 **Batch** A

Aim:

Implement Java programs to illustrate the concept of classes, objects, constructor, method overloading and array of objects

Theory:

Method overloading and constructor overloading

1. Method overloading:

   Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different. It is similar to constructor overloading in Java, that allows a class to have more than one constructor having different argument lists.
   There are three types of Method overloading:
   a. By number of parameters
   b. By data type of parameters
   c. By sequence of data type of parameters

2. Constructor overloading

   Like methods, constructors can also be overloaded. Constructor overloading is a concept of having more than one constructor with different parameters list, in such a way so that each constructor performs a different task.
   Its types are similar to that of Method overloading.

Constructor Chaining

Constructor chaining is the process of calling one constructor from another constructor with respect to the current object.

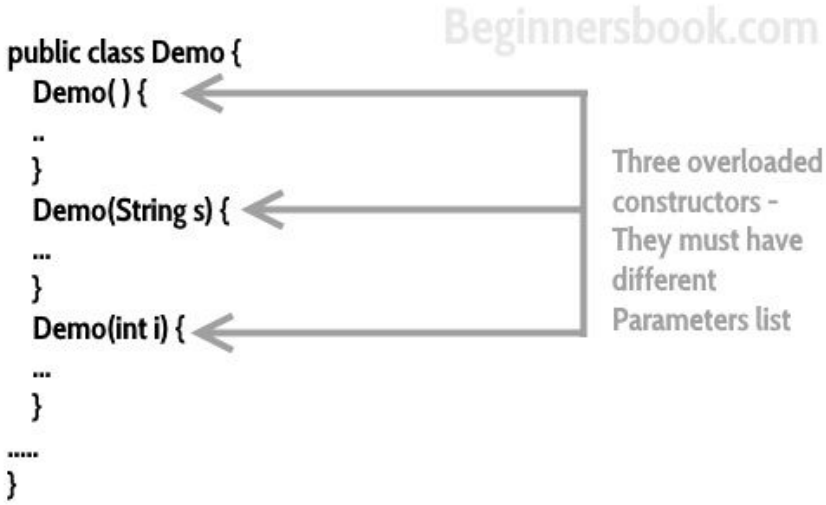
Constructor chaining can be done in two ways:

1. **Within same class:** It can be done using this() keyword for constructors in same class

2. **From base class:** by using super() keyword to call constructor from the base class.

The real purpose of Constructor Chaining is that you can pass parameters through a bunch of different constructors, but only have the initialization done in a single place.
This allows you to maintain your initializations from a single location, while providing multiple constructors to the user.
If we don't chain, and two different constructors require a specific parameter, you will have to initialize that parameter twice, and when the initialization changes, you'll have to change it in every constructor, instead of just the one.
As a rule, constructors with fewer arguments should call those with more

For example:



Usage of “this” and “super” keywords

This and super are reserved keywords in java i.e, we can’t use them as an identifier.

1. Usage of “this”

   ‘this’ is a reference variable that refers to the current object and can be used in various ways:

   A. Using this() to invoke current class constructor
   B. Using ‘this’ keyword to return the current class instance
   C. Using ‘this’ keyword as method parameter
   D. Using ‘this’ keyword to invoke current class method

2. Usage of “super”

   The super keyword in java is a reference variable that is used to refer to parent class objects.
   The keyword “super” came into the picture with the concept of Inheritance. It is majorly used in the following contexts:

   A. Use of super with variables
   B. Use of super with methods
   C. Use of super with constructors

Array of Objects

A Java array of objects, as defined by its name, stores an array of objects. Unlike a traditional array that stores values like string, integer, Boolean, etc an array of objects stores OBJECTS. The array elements store the location of the reference variables of the object.

Syntax:

Class obj[]= new Class[array_length]

To create and array of object, do the following:

1. The following statement creates an Array of Objects.

   Class_name [] objArray;

2. Alternatively, you can also declare an Array of Objects as shown below:

   Class_name objArray[];

   Both the above declarations imply that objArray is an array of objects.

3. So, if you have a class 'Employee' then you can create an array of Employee objects as given below:

   Employee[] empObjects;
   OR
   Employee empObjects[];

   The declarations of the array of objects above will need to be instantiated using 'new' before being used in the program.

4. You can declare and instantiate the array of objects as shown below:

   Employee[] empObjects = new Employee[2];

5. Once the array of objects is instantiated, you have to initialize it with values. As the array of objects is different from an array of primitive types, you cannot initialize the array in the way you do with primitive types.

Program:

Write a program that would print the information (name, year of joining, salary, address) of three employees by creating a class named 'Employee'. (Hint: illustrate the concept of class and object)

//code

```
class Employee {
	String empName;
	int empYear;
	String empAddress;

	Employee(String name, int year, String address) {
		empName = name;
		empYear = year;
		empAddress = address;
	}
	void display() {
		System.out.println(empName+"	"+empYear+"	"+empAddress);
	}

	public static void main(String args[]) {
		Employee emp1 = new Employee("Bond ", 1994, "64C-WallsStreet, Kalwa");
		Employee emp2 = new Employee("James", 2000, "221B-BakerStreet, Kalyan");
		Employee emp3 = new Employee("Bond ", 1999, "30-WellingtonSquare,
Ghatkopar");
		System.out.println("Name	Year of joining	Address");
		System.out.println();
		emp1.display();
		emp2.display();
		emp3.display();
	}
}
```

//output

```
E:\Aamir\Sem-3\OOPM\Lab Assignment 2>javac Employee.java

E:\Aamir\Sem-3\OOPM\Lab Assignment 2>java Employee
Name    Year of joining     Address

Carl          1994          64C-WallsStreat, Kalwa
Mike          2000          68D-WallsStreat, Kalwa
John          1999          26B-WallsStreat, Kalwa


E:\Aamir\Sem-3\OOPM\Lab Assignment 2>javac Employee.java

E:\Aamir\Sem-3\OOPM\Lab Assignment 2>java Employee
Name    Year of joining     Address

Bond          1994           64C-WallsStreet, Kalwa
James         2000           221B-BakerStreet, Kalyan
Bond          1999           30-WellingtonSquare, Ghatkopar
```

Write a program to print the area of a rectangle by creating a class named 'Area' having two methods. First method named as 'setDim' takes the length and breadth of the rectangle as parameters and the second method named as 'getArea' returns the area of the rectangle. Length and breadth of rectangle are entered through the command line.

```
//code
class Area {
	double len, bre;

	void setDim(double l, double b) {
		this.len = l;
		this.bre = b;
		System.out.println("Area is :  "+getArea());
	}
	double getArea() {
		return (len * bre);
	}

	public static void main(String args[]) {
		double length = Double.parseDouble(args[0]);
		double breadth = Double.parseDouble(args[1]);
		Area obj = new Area();
		obj.setDim(length, breadth);
	}
}
```

//output

```
E:\Aamir\Sem-3\OOPM\Lab Assignment 2>javac Area.java

E:\Aamir\Sem-3\OOPM\Lab Assignment 2>java Area 14.2 20
Area is :  284.0

E:\Aamir\Sem-3\OOPM\Lab Assignment 2>java Area 5 7
Area is :  35.0
```

Create a class 'Student' with three data members which are name, age and address. The constructor of the class assigns default values name as "unknown", age as '0' and address as "not available". It has two methods with the same name 'setInfo'. First method has two parameters for name and age and assigns the same whereas the second method takes three parameters which are assigned to name, age and address respectively. Print the name, age and address of 10 students.

```
//code
import java.util.*;
import java.lang.*;
import java.io.*;

class Student {

	String name, addr;
	int age=0;
	Student() {
		this.name = "unknown";
		this.age = 0;
		this.addr = "not available";
	}
	void setInfo(String name, int age, String addr) {
		this.name = name;
		this.age = age;
		this.addr = addr;
	}
	void setInfo(String name, int age) {
		this.name = name;
		this.age = age;
	}
	void display() {
		System.out.println("_____");
		System.out.println("Name: "+this.name);
		System.out.println("Age: "+this.age);
		System.out.println("Address: "+this.addr);
	}
}

class ObjectArray {
	public static void main(String args[]) {
		int choice, age1, age2;
```

```
		String name1, name2, address1, address2;
		Scanner sc = new Scanner(System.in);
		//input
		Student obj[] = new Student[10];
		for(int i=0 ; i<10 ; i++) {
			obj[i] = new Student();
			System.out.println("*1* Name Age");
			System.out.println("*2* Name Age Address");
			System.out.print("Enter your choice :  ");
			choice = sc.nextInt();
			sc.nextLine();
			switch(choice) {
				case 1:
					System.out.print("Enter Name of Student :  ");
					name1 = sc.nextLine();
					System.out.print("Enter Age of Student :  ");
					age1 = sc.nextInt();
					//sc.nextLine();
					obj[i].setInfo(name1, age1);
					break;
				case 2:
					System.out.print("Enter Name of Student :  ");
					name2 = sc.nextLine();
					System.out.print("Enter age of Student :  ");
					age2 = sc.nextInt();
					sc.nextLine();
					System.out.print("Enter Address of Student :  ");
					address2 = sc.nextLine();
					obj[i].setInfo(name2, age2, address2);
					break;
				default :
					System.out.println("Invalid choice");
			}
		}
		//display
		for (int i=0 ; i<10 ; i++) {
			obj[i].display();
		}
	}
}
```

//output

```
E:\Aamir\Sem-3\OOPM\Lab Assignment 2>javac ObjectArray.java

E:\Aamir\Sem-3\OOPM\Lab Assignment 2>java ObjectArray
*1* Name Age
*2* Name Age Address
Enter your choice :  2
Enter Name of Student :  Aamir Ansari
Enter age of Student :  19
Enter Address of Student :  Kalwa
*1* Name Age
*2* Name Age Address
Enter your choice :  2
Enter Name of Student :  Krishna
Enter age of Student :  19
Enter Address of Student :  Thane
*1* Name Age
*2* Name Age Address
Enter your choice :  2
Enter Name of Student :  Kanaiya
Enter age of Student :  18
Enter Address of Student :  Rajkot
*1* Name Age
*2* Name Age Address
Enter your choice :  1
Enter Name of Student :  Ninad
Enter Age of Student :  18
*1* Name Age
*2* Name Age Address
Enter your choice :  1
Enter Name of Student :  Sreekesh
Enter Age of Student :  59
*1* Name Age
*2* Name Age Address
Enter your choice :  2
Enter Name of Student :  Isha
Enter age of Student :  18
Enter Address of Student :  Kalyan
```

```
*1* Name Age
*2* Name Age Address
Enter your choice :  1
Enter Name of Student :  Jisha
Enter Age of Student :  19
*1* Name Age
*2* Name Age Address
Enter your choice :  1
Enter Name of Student :  Suchindra
Enter Age of Student :  48
*1* Name Age
*2* Name Age Address
Enter your choice :  2
Enter Name of Student :  James Bond
Enter age of Student :  47
Enter Address of Student :  Wellington square
*1* Name Age
*2* Name Age Address
Enter your choice :  2
Enter Name of Student :  Sherlock
Enter age of Student :  99
Enter Address of Student :  221B Baker-Street
```

Name: Aamir Ansari
Age: 19
Address: Kalwa

Name: Krishna
Age: 19
Address: Thane

Name: Kanaiya
Age: 18
Address: Rajkot

Name: Ninad
Age: 18
Address: not available

Name: Sreekesh
Age: 59
Address: not available

Name: Isha
Age: 18
Address: Kalyan

Name: Jisha
Age: 19
Address: not available

Name: Suchindra
Age: 48
Address: not available

Name: James Bond
Age: 47
Address: Wellington square

Name: Sherlock
Age: 99
Address: 221B Baker-Street

JAVA Lab
Lab Experiment number 03

Name: Aamir Ansari **Batch:** A **Roll no.** 01

Aim: Write a menu driven program to implement recursive Functions for following tasks:

a) To find GCD and LCM
b) To print n Fibonacci numbers
c) To find reverse of number
d) To solve 1 +2+3+4+.....+(n- l)+n

Theory:

Recursion:

Recursion is the technique of making a function call itself. This technique provides a way to break complicated problems down into simple problems which are easier to solve.

1. Recursion is a process in which the problem is specified in terms of itself.
2. The function should be called itself to implement recursion.
3. The function which calls itself is known as a recursive function.
4. A condition must be specified to stop recursion; otherwise it will lead to an infinite process.
5. In case of recursion, all partial solutions are combined to obtain the final solution.

Iteration and Recursion, comparison:

Time Complexity: Finding the Time complexity of Recursion is more difficult than that of Iteration.

> **Recursion**: Time complexity of recursion can be found by finding the value of the nth recursive call in terms of the previous calls. Thus, finding the destination case in terms of the base case, and solving in terms of the base case gives us an idea of the time complexity of recursive equations.
>
> **Iteration**: Time complexity of iteration can be found by finding the number of cycles being repeated inside the loop.

Usage: Usage of either of these techniques is a trade-off between time complexity and size of code. If time complexity is the point of focus, and the number of recursive calls would be large, it is better to use iteration. However, if time complexity is not an issue and shortness of code is, recursion would be the way to go.

> **Recursion**: Recursion involves calling the same function again, and hence, has a very small length of code. However, as we saw in the analysis, the time complexity of recursion can get to be exponential when there are a considerable number of recursive calls. Hence, usage of recursion is advantageous in shorter code, but higher time complexity.
>
> **Iteration**: Iteration is repetition of a block of code. This involves a larger size of code, but the time complexity is generally lesser than it is for recursion.

Overhead: Recursion has a large amount of Overhead as compared to Iteration.

Recursion: Recursion has the overhead of repeated function calls, that is due to repetitive calling of the same function, the time complexity of the code increases manifold.

Iteration: Iteration does not involve any such overhead.

Infinite Repetition: Infinite Repetition in recursion can lead to CPU crash but in iteration, it will stop when memory is exhausted.

Recursion: In Recursion, Infinite recursive calls may occur due to some mistake in specifying the base condition, which on never becoming false, keeps calling the function, which may lead to system CPU crash.

Iteration: Infinite iteration due to mistake in iterator assignment or increment, or in the terminating condition, will lead to infinite loops, which may or may not lead to system errors, but will surely stop program execution any further.

Example of Recursion:

The simple function count demonstrates the use of recursion

```
function Count (integer N)
        if (N <= 0) return "Must be a Positive Integer";
        if (N > 9) return "Counting Completed";
        else return Count (N+1);
end function
```

The function Count() above uses recursion to count from any number between 1 and 9, to the number 10. For example, Count(1) would return 2,3,4,5,6,7,8,9,10. Count(7) would return 8,9,10. The result could be used as a roundabout way to subtract the number from 10.

Advantages and disadvantages of Recursion:

Advantages

1. The main benefit of a recursive approach to algorithm design is that it allows programmers to take advantage of the repetitive structure present in many problems.
2. Complex case analysis and nested loops can be avoided.
3. Recursion can lead to more readable and efficient algorithm descriptions.
4. Recursion is also a useful way for defining objects that have a repeated similar structural form.

Disadvantages

1. Slowing down execution time and storing on the run-time stack more things than required in a non recursive approach are major limitations of recursion.
2. If recursion is too deep, then there is a danger of running out of space on the stack and ultimately the program crashes.
3. Even if some recursive function repeats the computations for some parameters, the run time can be prohibitively long even for very simple cases.

Program:

```
//code

import java.util.*;

class Recursion {
```

```
static int sum = 0, rem;

// method to calculate GCD
static int gcd(int n1, int n2) {

    if (n2 != 0) {
        return gcd(n2, n1%n2);
    } else {
        return n1;
    }

}

// Fibonacci
static int fibo(int n) {

    if (n==0 || n==1) {
        return n;
    } else {
        return (fibo(n-1) + fibo(n-2));
    }

}

// Reverse
static int reverse(int n) {

    if (n != 0) {
        rem = n % 10;
        sum = sum*10 + rem;
        reverse(n/10);
    } else {
        return sum;
    }
    return sum;

}

// Ramanujan series
```

```
static int seriesSum (int n) {

    if (n != 0) {
        sum += n;
        return seriesSum(n-1);
    }
    return sum;

}

public static void main(String args[]) {

    Scanner sc = new Scanner(System.in);
    int choice;

    while (true) {
        // choices
        System.out.println("**********************");
        System.out.println("*1 GCD and LCM");
        System.out.println("*2 Fibonacci");
        System.out.println("*3 Reverse");
        System.out.println("*4 Ramanujan series");
        System.out.println("*5 EXIT");
        System.out.print("Enter Your choice :  ");
        choice = sc.nextInt();

        switch (choice) {

            case 1:     // GCD and LCM
                System.out.print("Enter First number  :  ");
                int num1 = sc.nextInt();
                System.out.print("Enter Second number :  ");
                int num2 = sc.nextInt();

                // get GCD
                int gcd = gcd(num1, num2);

                // get LCM
                int lcm = (num1 * num2) / gcd;
```

```
    // printing
    System.out.println("GCD of "+num1+" and "+num2+" is :  "+gcd);
    System.out.println("LCM of "+num1+" and "+num2+" is :  "+lcm);

    sum=0;
    break;

case 2:     // Fibonacci
    // input
    int num;
    System.out.print("Enter number of elements required in Fibonacci series :  ");
    num = sc.nextInt();

    // print series
    for (int i=0 ; i<num ; i++) {
       System.out.print(fibo(i)+"  ");
    }
    System.out.println();

    sum=0;
    break;

case 3:     // Reverse
    // input
    int reverse;
    System.out.print("Enter a number to Reverse :  ");
    num = sc.nextInt();
    reverse = reverse(num);

    System.out.println("Reverse of "+num+" is :  "+reverse);

    sum=0;
    break;

case 4:     //Ramanujan series
    // input
    int numRam=0;
    System.out.print("Enter value of n in Ramanujan summation series :  ");
    numRam = sc.nextInt();
```

```
                // display
                System.out.println("Sum of "+numRam+" terms in Ramanujan series is :
"+seriesSum(numRam));
                sum=0;
                break;

            case 5:     // Exit
                System.out.println("*** E X I T I N G ***");
                System.exit(1);

            default:
                System.out.println("INVALID INPUT");

        }
    }
  }
}
```

Output

```
E:\Aamir\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 3\Code>javac Recursion.java

E:\Aamir\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 3\Code>java Recursion
********************
*1 GCD and LCM
*2 Fibonacci
*3 Reverse
*4 Ramanujan series
*5 EXIT
Enter Your choice :  1
Enter First number  :  15
Enter Second number :  20
GCD of 15 and 20 is :  5
LCM of 15 and 20 is :  60
********************
*1 GCD and LCM
*2 Fibonacci
*3 Reverse
*4 Ramanujan series
*5 EXIT
Enter Your choice :  1
Enter First number  :  3
Enter Second number :  4
GCD of 3 and 4 is :  1
LCM of 3 and 4 is :  12
********************
*1 GCD and LCM
*2 Fibonacci
*3 Reverse
*4 Ramanujan series
*5 EXIT
Enter Your choice :  2
Enter number of elements required in Fibonacci series :  5
0  1  1  2  3
********************
*1 GCD and LCM
*2 Fibonacci
*3 Reverse
*4 Ramanujan series
*5 EXIT
Enter Your choice :  2
Enter number of elements required in Fibonacci series :  10
0  1  1  2  3  5  8  13  21  34
********************
```

```
*1 GCD and LCM
*2 Fibonacci
*3 Reverse
*4 Ramanujan series
*5 EXIT
Enter Your choice :  3
Enter a number to Reverse :  12345
Reverse of 12345 is :  54321
********************
*1 GCD and LCM
*2 Fibonacci
*3 Reverse
*4 Ramanujan series
*5 EXIT
Enter Your choice :  3
Enter a number to Reverse :  52384
Reverse of 52384 is :  48325
********************
*1 GCD and LCM
*2 Fibonacci
*3 Reverse
*4 Ramanujan series
*5 EXIT
Enter Your choice :  4
Enter value of n in Ramanujan summation series :  5
Sum of 5 terms in Ramanujan series is :  15
********************
*1 GCD and LCM
*2 Fibonacci
*3 Reverse
*4 Ramanujan series
*5 EXIT
Enter Your choice :  4
Enter value of n in Ramanujan summation series :  8
Sum of 8 terms in Ramanujan series is :  36
********************
*1 GCD and LCM
*2 Fibonacci
*3 Reverse
*4 Ramanujan series
*5 EXIT
Enter Your choice :  5
*** E X I T I N G ***
```

JAVA Lab
Lab Experiment number 04

Name: Aamir Ansari **Batch:** A **Roll no.** 01

Aim: Write a JAVA program to implement the concept of inheritance

Theory:

1. Inheritance

Inheritance is an important pillar of OOP (Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.
The keyword used for inheritance is extends.

example:

```
class derived-class extends base-class  {
        //methods and fields
}
```

2. Types of Inheritance

2.1.Single Inheritance

In single inheritance, subclasses inherit the features of one superclass.

2.2.Multilevel Inheritance

In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class

2.3.Hierarchical Inheritance

In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one sub class

3. Reason for multiple inheritance being unused in JAVA

Multiple inheritance is not used in JAVA, the reason is to prevent ambiguity
Consider a case where class B extends class A and Class C and both class A and C have the same method display().
Now JAVA compiler cannot decide, which display method it should inherit. To prevent such situation, multiple inheritances is not allowed in JAVA.
The two parent classes may define different ways of doing the same thing, and the subclass can't choose which one to pick.

4. Usage of 'super'

The super keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.
super keyword can be used to call constructor of parent class, by using
super(//parameters) in the sub class

super keyword can also call variables or methods from parent class in the following way
super.parentMethod();
super.parentVariable;

5. Method Overriding

In any object-oriented programming language, Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature, and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.
Methods are overridden by using the same name, but with different parameter or by using it in different classes.
When a method is called in sub class, it automatically overrides the method of parent class.

Program:

```
//code
import java.util.*;

class Student {
    // instance variable => general information of all students
    String name;
    int age;
    String programme;

    // constructor
    Student(String name, int age, String programme) {
        this.name = name;
        this.age = age;
        this.programme = programme;
    }

    // method to display
    void display_student_info() {
        System.out.println("Name of Student : "+name);
        System.out.println("Age of Student : "+age);
        System.out.println("Programme of Student : "+programme);
    }
}

class ReasearchStudent extends Student {
    // instance variable => information of Reasearch Student
    String specialization;
    double workingExperience;

    // constructor
    ReasearchStudent(String name, int age, String programme, String specialization, double
workingExperience) {
        super(name, age, programme);    // calls super constructor
        this.specialization = specialization;
```

```
        this.workingExperience = workingExperience;
    }

    // getters and setters of ReasearchStudent
    String getSpecialization() {
        return this.specialization;
    } void setSpecialization(String specialization) {
        this.specialization = specialization;
    }

    double getWorkingExperience() {
        return this.workingExperience;
    } void setWorkingExperience(double workingExperience) {
        this.workingExperience = workingExperience;
    }

    // method to display
    void display_student_info() {
        System.out.println("*   Name  of   Student   :       "+name);
        System.out.println("*   Age   of   Student   :       "+age);
        System.out.println("*   Programme of Student :       "+programme);
        System.out.println("*   specialization  of student :  "+specialization);
        System.out.println("*   Work experience of student :  "+workingExperience);
    }
}

class GradStudent extends Student {
    // instance variable => information of Grad Student
    double percentage;
    String stream;

    // constructor
    GradStudent(String name, int age, String programme, double percentage, String stream) {
        super(name, age, programme);    // calls super constructor
        this.percentage = percentage;
```

```
        this.stream = stream;
    }

    // getters and setters of Grad Student
    double getPercentage() {
        return this.percentage;
    } void setPercentage(double percentage) {
        this.percentage = percentage;
    }

    String getStream() {
        return this.stream;
    } void setStream(String stream) {
        this.stream = stream;
    }

    // method to display
    void display_student_info() {
        System.out.println("*  Name of Student  : "+name);
        System.out.println("*  Age of Student  : "+age);
        System.out.println("*  Programme of Student :  "+programme);
        System.out.println("*  Percentage  of student :  "+percentage);
        System.out.println("*  Stream of student  : "+stream);
    }
}

class Inheritance {

    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);
        int choice;
        int age;
        String name, programme, specialization, stream;
        double workingExperience, percentage;
```

```
        while (true) {
            System.out.println("*1 Reasearch Student");
            System.out.println("*2 Graduate  Student");
            System.out.println("*3 EXIT");
            System.out.print("Enter the Type of student  :  ");
            choice = sc.nextInt();
            sc.nextLine();

            switch (choice) {
                case 1:    // Reasearch Student
                    System.out.print("Enter Name  of   Student   :  ");
                    name = sc.nextLine();

                    System.out.print("Enter Age   of   Student   :  ");
                    age = sc.nextInt();sc.nextLine();

                    System.out.print("Enter Programme of Student :  ");
                    programme = sc.nextLine();

                    System.out.print("Enter specialization  of student :  ");
                    specialization = sc.nextLine();

                    System.out.print("Enter Work experience of student :  ");
                    workingExperience = sc.nextDouble();

                    ReasearchStudent rStudent = new ReasearchStudent(name, age, programme,
specialization, workingExperience);
                    System.out.println();
                    System.out.println("**********************************************");
                    rStudent.display_student_info();
                    System.out.println("**********************************************");
                    break;

                case 2:     // Grag Student
```

```
            System.out.print("Enter Name  of   Student   :  ");
            name = sc.nextLine();

            System.out.print("Enter Age   of   Student   :  ");
            age = sc.nextInt();sc.nextLine();

            System.out.print("Enter Programme of Student  :  ");
            programme = sc.nextLine();

            System.out.print("Enter Percentage of student :  ");
            percentage = sc.nextDouble();sc.nextLine();

            System.out.print("Enter Stream  of  student   :  ");
            stream = sc.nextLine();

            GradStudent gStudent = new GradStudent(name, age, programme, percentage, stream);
            System.out.println();
            System.out.println("************************************************");
            gStudent.display_student_info();
            System.out.println("************************************************");
            System.out.println();
            break;

        case 3:
            System.out.println();
            System.out.println("*** E X I T I N G ***");
            System.exit(1);

        default:
            System.out.println("INVALID INPUT");

      }
    }
  }
}
```

Output

```
E:\Aamir\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 4>javac Inheritance.java

E:\Aamir\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 4>java Inheritance
*1 Reasearch Student
*2 Graduate  Student
*3 EXIT
Enter the Type of student  :  1
Enter Name  of   Student   :  Aamir Ansari
Enter Age   of   Student   :  18
Enter Programme of Student :  Full time
Enter specialization  of student :  Psychology
Enter Work experience of student :  12.5

*********************************************
*   Name  of   Student   :        Aamir Ansari
*   Age   of   Student   :        18
*   Programme of Student :        Full time
*   specialization  of student :  Psychology
*   Work experience of student :  12.5
*********************************************


*1 Reasearch Student
*2 Graduate  Student
*3 EXIT
Enter the Type of student  :  2
Enter Name  of   Student    :  Krishna Subramanian
Enter Age   of   Student    :  18
Enter Programme of Student  :  Part Time
Enter Percentage of student :  95.21
Enter Stream  of  student   :  Science

***********************************************
*   Name  of   Student   :  Krishna Subramanian
*   Age   of   Student   :  18
*   Programme of Student :  Part Time
*   Percentage  of student :  95.21
*   Stream  of  student    :  Science
***********************************************

*1 Reasearch Student
*2 Graduate  Student
*3 EXIT
Enter the Type of student  :  3

*** E X I T I N G ***
```

OOPM Lab
Lab Assingment number 5

Name: Aamir Ansari **Batch:** A **Roll no.** 01

Aim: Write a JAVA program to implement the concept of Final variable

Problem statement: Display PRN numbers of 10 direct diploma students who are taking admission in VESIT for the current academic year. (Hint: use array of objects and final variable)

Theory:

1. Final Keyword:

   In the Java programming language, the final keyword is used in several contexts to define an entity that can only be assigned once.
   Final keyword can be used along with variables, methods and classes.
   A) final variable
   B) final method
   C) final class

2. Define final variable and blank final variable with example

   A) Final variables are nothing but constants.
   B) We cannot change the value of a final variable once it is initialized.
   C) It is good practice to represent final variables in all uppercase, using underscore to separate words.
   D) If final variable is not initialized then it is called as Blank Final Variable.
   E) Final variables can be static as well.

   Example:

```
//a final variable
final int LIMIT = 5;

// a blank final variable
final int LIMIT;

// a final static variable PI
static final double PI = 3.14;

// a blank final static variable
static final double PI;
```

3. Why and when to use final and blank final variables

A) We cannot change the value of a final variable once assigned.
B) Hence final variables must be used only for the values that we want to remain constant throughout the execution of program.
C) Lets say we have an employee class which is having a field called EmpID.
D) Since EmpID should not be changed once the employee is appointed, we can declare it as a final variable in a class, but we cannot initialize EmpID in advance for all the employees.
E) In such case we can declare EmpID variable as blank final and we initialize this value during object creation.

4. Explain the final method and final class with example

A) Final Method

1. When a method is declared with final keyword, it is called a final method.
2. A final method cannot be overridden.
3. Which means even though a sub class can call the final method of parent class without any issues but it cannot override it.
4. We must declare methods with final keyword for which we required to follow the same implementation throughout all the derived classes.
   Example:

```
class Bike {
final void run(){
          System.out.println("running");
     }
}

class Honda extends Bike {
     void run() {
          System.out.println("running safely with 100kmph");
     }
      public static void main(String args[]) {
           Honda obj = new Honda();
           obj.run();
      }
}
// gives compilation error as final method can not be overridden
```

B) Final Class:

1. When a class is declared with final keyword, it is called a final class.

2. A final class cannot be extended(inherited).
3. There are two uses of a final class:
    1. One is definitely to prevent inheritance, as final classes cannot be extended.
    2. The other use of final with classes is to create an immutable class.
4. For example, all Wrapper Classes like Integer, Float etc. and String class are final classes and are immutable too.

Example:

```
final class Bike{}

class Honda extends Bike {
    void run() {
        System.out.println("running safely with 100kmph");
    }
    public static void main(String args[]) {
        Honda obj = new Honda1();
        obj.run();
    }
}

 // gives compilation error as final class can not be extended
```

```
// code
import java.util.*;
```

```
class DiplomaStudent {
    // declaration of blank final variable
    final long PRN;

    DiplomaStudent(long val) {
        PRN = val;
    }

    void display(int number) {
        System.out.println("* PRN of Student "+number+" is :  "+PRN+"  *");
    }
}

class Final {
    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);
        long studentPRN=0;
        DiplomaStudent students[] = new DiplomaStudent[10];

        // input
        for (int i=0 ; i<10 ; i++) {
            System.out.print("Enter PRN number of Student number "+(i+1)+" :  ");
            studentPRN = sc.nextLong();
            students[i] = new DiplomaStudent(studentPRN);
        }

        // display
        System.out.println();
        System.out.println("************************************");
        for (int i=0 ; i<10 ; i++) {
            students[i].display(i+1);
        }
        System.out.println("************************************");
    }
}
```

// output

```
E:\Aamir\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 5>javac Final.java

E:\Aamir\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 5>java Final
Enter PRN number of Student number 1 :  2019001
Enter PRN number of Student number 2 :  2019002
Enter PRN number of Student number 3 :  2019003
Enter PRN number of Student number 4 :  2019004
Enter PRN number of Student number 5 :  2019005
Enter PRN number of Student number 6 :  2019006
Enter PRN number of Student number 7 :  2019007
Enter PRN number of Student number 8 :  2019008
Enter PRN number of Student number 9 :  2019009
Enter PRN number of Student number 10 :  2019010
**********************************
* PRN of Student 1 is :  2019001  *
* PRN of Student 2 is :  2019002  *
* PRN of Student 3 is :  2019003  *
* PRN of Student 4 is :  2019004  *
* PRN of Student 5 is :  2019005  *
* PRN of Student 6 is :  2019006  *
* PRN of Student 7 is :  2019007  *
* PRN of Student 8 is :  2019008  *
* PRN of Student 9 is :  2019009  *
* PRN of Student 10 is :  2019010  *
**********************************



E:\Aamir\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 5>javac Final.java

E:\Aamir\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 5>java Final
Enter PRN number of Student number 1 :  2077001
Enter PRN number of Student number 2 :  2077002
Enter PRN number of Student number 3 :  2077003
Enter PRN number of Student number 4 :  2077004
Enter PRN number of Student number 5 :  2077005
Enter PRN number of Student number 6 :  2077006
Enter PRN number of Student number 7 :  2077007
Enter PRN number of Student number 8 :  2077008
Enter PRN number of Student number 9 :  2077009
Enter PRN number of Student number 10 :  2077010

**********************************
* PRN of Student 1 is :  2077001  *
* PRN of Student 2 is :  2077002  *
* PRN of Student 3 is :  2077003  *
* PRN of Student 4 is :  2077004  *
* PRN of Student 5 is :  2077005  *
* PRN of Student 6 is :  2077006  *
* PRN of Student 7 is :  2077007  *
* PRN of Student 8 is :  2077008  *
* PRN of Student 9 is :  2077009  *
* PRN of Student 10 is :  2077010  *
**********************************
```

OOPM Lab
Lab Assingment number 06

Name: Aamir Ansari **Batch:** A **Roll no.** 01

Aim: Write a JAVA program to implement the concept of Interface

Problem statement:

Create an interface vehicle and classes like Bicycle, Bike and Car having common functionalities like and put all the common functionalities in the interface. Classes like Bicycle, Bike and Car implement all these functionalities in their own class in their own way. Write another class to use these functionalities from at least one class.

Theory:

1) Interface

   An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

   Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.

   Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.

2) Why and When to use

   In JAVA multiple inheritance is not primitively allowed, in order to perform multiple inheritance, Interface is used.

   Interface is used to achieve absolute abstraction.

   Interface is used to achieve loose coupling.

3) Interface with Inheritance example

```
// example
public class MammalInt implements Animal {
public void eat() {
        System.out.println("Mammal eats");
}

public void travel() {
        System.out.println("Mammal travels");
}

public int noOfLegs() {
        return 0;
}

public static void main(String args[]) {
        MammalInt m = new MammalInt();
        m.eat();
```

```
                m.travel();
        }
}

// output
Mammal eats
Mammal travels
```

Code (A)

```
import java.util.*;

interface Vehicle {
   void setDescriptions();
   void displayDescription();
}

class Bicycle implements Vehicle {
   String companyName = "";
   String color = "";
   int registrationNumber = 0;
   int numberOfWheels = 0;

   // gets and sets information of Bicycle
   public void setDescriptions() {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter name of the company :  ");
      this.companyName = sc.nextLine();

      System.out.print("Enter colour of the Bicycle :  ");
      this.color = sc.nextLine();

      System.out.print("Enter registration number of the Bicycle :  ");
      this.registrationNumber = sc.nextInt();

      System.out.print("Enter number of wheels of the Bicycle :  ");
      this.numberOfWheels = sc.nextInt();
   }
   // display information of Bicycle
   public void displayDescription() {
      System.out.println();
      System.out.println("*** Description of BICYCLE ***");
      System.out.println("COMPANY      :  " + this.companyName);
      System.out.println("COLOUR       :  " + this.color);
      System.out.println("Reg Number   :  " + this.registrationNumber);
      System.out.println("No of Wheels :  " + this.numberOfWheels);
      System.out.println("*****************************");
      System.out.println();
   }
}
```

```
class Bike implements Vehicle {
    String companyName = "";
    String color = "";
    int registrationNumber = 0;
    String namePlate = "";
    int numberOfWheels = 0;

    // gets and sets information of Bike
    public void setDescriptions() {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter name of the company :  ");
        this.companyName = sc.nextLine();

        System.out.print("Enter colour of the Bike :  ");
        this.color = sc.nextLine();

        System.out.print("Enter registration number of the Bike :  ");
        this.registrationNumber = sc.nextInt();
        sc.nextLine();

        System.out.print("Enter name plate of the Bike :  ");
        this.namePlate = sc.nextLine();

        System.out.print("Enter number of wheels of the Bike :  ");
        this.numberOfWheels = sc.nextInt();
    }
    // display information of Bike
    public void displayDescription() {
        System.out.println();
        System.out.println("*** Description of Bike ***");
        System.out.println("Company      :  " + this.companyName);
        System.out.println("Colour       :  " + this.color);
        System.out.println("Reg Number   :  " + this.registrationNumber);
        System.out.println("Name plate   :  " + this.namePlate);
        System.out.println("No of Wheels :  " + this.numberOfWheels);
        System.out.println("****************************");
        System.out.println();
    }
}

class Car implements Vehicle {
    String companyName = "";
    String color = "";
    int registrationNumber = 0;
    String namePlate = "";
    int numberOfWheels = 0;

    // gets and sets information of Car
    public void setDescriptions() {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter name of the company :  ");
        this.companyName = sc.nextLine();
```

```
        System.out.print("Enter colour of the Car :  ");
        this.color = sc.nextLine();

        System.out.print("Enter registration number of the Car :  ");
        this.registrationNumber = sc.nextInt();
        sc.nextLine();

        System.out.print("Enter name plate of the Car :  ");
        this.namePlate = sc.nextLine();

        System.out.print("Enter number of wheels of the Car :  ");
        this.numberOfWheels = sc.nextInt();
    }
    // display information of Car
    public void displayDescription() {
        System.out.println();
        System.out.println("*** Description of CAR ***");
        System.out.println("Company      :  " + this.companyName);
        System.out.println("Colour       :  " + this.color);
        System.out.println("Reg Number   :  " + this.registrationNumber);
        System.out.println("Name plate   :  " + this.namePlate);
        System.out.println("No of Wheels :  " + this.numberOfWheels);
        System.out.println("**************************");
        System.out.println();
    }
}

class  InterfaceProgram {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int choice;
        while (true) {
            System.out.println("(1) Bicycle ");
            System.out.println("(2) BIKE");
            System.out.println("(3) CAR");
            System.out.println("(4) EXIT");
            System.out.print("Enter your choice :  ");
            choice = sc.nextInt();

            switch (choice) {
                case 1:
                    Bicycle objOfBicycle = new Bicycle();
                    objOfBicycle.setDescriptions();
                    objOfBicycle.displayDescription();
                    break;

                case 2:
                    Bike objOfBike = new Bike();
                    objOfBike.setDescriptions();
                    objOfBike.displayDescription();
                    break;
```

```
            case 3:
                Car objOfCar = new Car();
                objOfCar.setDescriptions();
                objOfCar.displayDescription();
                break;

            case 4:
                System.out.println("*** E X I T I N G ***");
                System.exit(0);

            default:
                System.out.println("*** INVALID INPUT ***");
        }
    }
  }
}
// output
```

```
E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 6\Code>javac InterfaceProgram.java

E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 6\Code>java InterfaceProgram
(1) Bicycle
(2) BIKE
(3) CAR
(4) EXIT
Enter your choice :  1
Enter name of the company :  Yamaha
Enter colour of the Bicycle :  Neon Blue
Enter registration number of the Bicycle :  8008
Enter number of wheels of the Bicycle :  2

*** Description of BICYCLE ***
COMPANY      :  Yamaha
COLOUR       :  Neon Blue
Reg Number   :  8008
No of Wheels :  2
*****************************

(1) Bicycle
(2) BIKE
(3) CAR
(4) EXIT
Enter your choice :  2
Enter name of the company :  Pagani
Enter colour of the Bike :  Crimson Red
Enter registration number of the Bike :  007
Enter name plate of the Bike :  BABA YAGA 7
Enter number of wheels of the Bike :  4

*** Description of Bike ***
Company      :  Pagani
Colour       :  Crimson Red
Reg Number   :  7
Name plate   :  BABA YAGA 7
No of Wheels :  4
*************************
```

```
(1) Bicycle
(2) BIKE
(3) CAR
(4) EXIT
Enter your choice :  3
Enter name of the company :  Mazda
Enter colour of the Car :  Silver
Enter registration number of the Car :  666
Enter name plate of the Car :  MH04 3393
Enter number of wheels of the Car :  4

*** Description of CAR ***
Company      :  Mazda
Colour       :  Silver
Reg Number   :  666
Name plate   :  MH04 3393
No of Wheels :  4
**************************

(1) Bicycle
(2) BIKE
(3) CAR
(4) EXIT
Enter your choice :  1
Enter name of the company :  Checking input
Enter colour of the Bicycle :  for second time
Enter registration number of the Bicycle :  989
Enter number of wheels of the Bicycle :  2

*** Description of BICYCLE ***
COMPANY      :  Checking input
COLOUR       :  for second time
Reg Number   :  989
No of Wheels :  2
****************************

(1) Bicycle
(2) BIKE
(3) CAR
(4) EXIT
Enter your choice :  4
*** E X I T I N G ***

E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 6\Code>
```

Code (B)

```
import java.util.*;
import java.lang.*;

interface Dad {
   void getXorY();
}

interface Mom {
   void getX();
}

class Child implements Dad, Mom {
   String XorY = "";
   String X = "";

   // get a gene from Dad
   public void getXorY() {
      if (Math.random() < 0.5) {
         this.XorY = "Y";
      } else {
         this.XorY = "X";
      }
   }
   // get a gene from Mom
   public void getX() {
      this.X = "X";
   }

   // Reveal the gender of child
   public void genderReveal() {
      String gender = this.X + this.XorY;
      if (gender.equals("XX")) {
         System.out.println("!!!!!!!!!!!!!!!!!!");
         System.out.println("!! It's a GIRL !!");
         System.out.println("!!!!!!!!!!!!!!!!!!");
      } else if (gender.equals("XY"){
         System.out.println("!!!!!!!!!!!!!!!!!!");
         System.out.println("!! It's a BOY !!");
         System.out.println("!!!!!!!!!!!!!!!!!!");
      }
   }

   public static void main(String args[]) {
      Child baby = new Child();
      baby.getXorY();
      baby.getX();
      baby.genderReveal();
   }
}
```

// output

```
E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 6\Code\B>javac Child.java

E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 6\Code\B>java Child
!!!!!!!!!!!!!!!!
!! It's a BOY !!
!!!!!!!!!!!!!!!!

E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 6\Code\B>java Child
!!!!!!!!!!!!!!!!
!! It's a BOY !!
!!!!!!!!!!!!!!!!

E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 6\Code\B>java Child
!!!!!!!!!!!!!!!!
!! It's a BOY !!
!!!!!!!!!!!!!!!!

E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 6\Code\B>java Child
!!!!!!!!!!!!!!!!
!! It's a BOY !!
!!!!!!!!!!!!!!!!

E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 6\Code\B>java Child
!!!!!!!!!!!!!!!!!
!! It's a GIRL !!
!!!!!!!!!!!!!!!!!

E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 6\Code\B>java Child
!!!!!!!!!!!!!!!!
!! It's a BOY !!
!!!!!!!!!!!!!!!!

E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 6\Code\B>java Child
!!!!!!!!!!!!!!!!!
!! It's a GIRL !!
!!!!!!!!!!!!!!!!!
```

OOPM Lab
Lab Assingment number 07

Name: Aamir Ansari **Batch:** A **Roll no.** 01

Aim: Write a JAVA program to implement the concept of Package

Theory:

Package:

A package in Java is used to group related classes. Think of it as a folder in a file directory. We use packages to avoid name conflicts, and to write a better maintainable code.

Uses of Package:

1. Packages provide code reusability, because a package contains group of classes
2. It helps in resolving naming collision when multiple packages have classes with the same name.
3. Packages also provides the hiding of class facility, Thus, other programs can not use the classes from hidden package.
4. Access limitation can be applied with the help of packages.
5. Nesting of packages, that is, one package can be defined in another package in a hierarchy fashion.

Types of Packages:

1. Built-in Packages (packages from the Java API)
   i. The Java API is a library of pre written classes, that are free to use, included in the Java Development Environment.
   ii. The library is divided into packages and classes. Meaning you can either import a single class (along with its methods and attributes), or a whole package that contain all the classes that belong to the specified package.
2. User-defined Packages (create your own packages)
   i. Packages created by user to increase reusability and simplicity of code

Program

credit package

```
// code
package credit;
public class AmountInWords {
   // data member
   int amount = 0;
   int temp = 0;
   // method to get amount from the user
   public void getAmount(int amount) {
      this.amount = amount;
   }

   // method to display the amount
   public void displayAmount() {
      int num;
      String word = "";
      // units
      String[] units = new String[] {"zero", "one", "two", "three", "four", "five", "six", "seven",
"eight", "nine", "ten", "eleven", "twelve", "thirteen", "fourteen", "fifteen", "sixteen", "seventeen",
"eighteen", "nineteen"};
      // tens
      String[] tens = new String[] {"zero","ten", "twenty", "thirty", "forty", "fifty", "sixty",
"seventy", "eighty", "ninety"};

      while (amount != 0) {
         if (amount == 10000) {    // lakhs
            word = "one lakh";
            break;
         }
         if (amount < 100000 && amount >= 1000) {    // thousand
            if (amount < 100000 && amount >=10000) {
               word += tens[amount/10000];
               word += " ";
               amount = amount % 10000;
            }
            if (amount < 10000 && amount >= 1000) {
               word += units[amount/1000];
            word += " thousand ";
            amount = amount % 1000;
            }
         }
         if (amount<1000 && amount >=100) {    // hundreds
```

```
            word += units[amount/100];
            word += " hundred ";
            amount = amount % 100;
         }
         if (amount == 0) {    // check for zero
            break;
         }
         if (amount < 100) {    // tens
            if (amount < 20) {
               word += units[amount%100];
               word += " ";
               amount = amount / 100;
            } else {
               word += tens[amount/10];
               word += " ";
               amount = amount%10;
            }
         }
      }
      System.out.println(word);
   }
}
```

Driver Class

```
// code
import java.util.*;
import credit.*;

class UsePackage {
   public static void main(String args[]) {
      Scanner sc = new Scanner(System.in);
      AmountInWords obj = new AmountInWords();
      System.out.print("Enter the amount :  ");
      int amount = sc.nextInt();

      obj.getAmount(amount);
      System.out.print("Amount i  n word :  ");
      obj.displayAmount();
   }
}
```

// output

```
E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 7>javac UsePackage.java

E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 7>java UsePackage
Enter the amount :  99999
Amount in word :  ninety nine thousand nine hundred ninety nine

E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 7>java UsePackage
Enter the amount :  54781
Amount in word :  fifty four thousand seven hundred eighty one

E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 7>
```

OOPM Lab
Lab Assingment number 08

Name: Aamir Ansari **Batch:** A **Roll no.** 01

Aim: Write a JAVA program to create user defined exception.

Problem statement:

Write a java program where the user will enter login id and password as input. Program will check whether the entered password is valid or invalid based on following password rules:

1. Passwords should contain at least one digit(0-9).
2. Password length should be between 8 to 15 characters.
3. Password should contain at least one special character (@, #, %, &, !, $, etc….)

If a user enters a valid password satisfying above criteria then show “Login Successful Message”. Otherwise create InvalidPasswordException stating "Please enter valid password of length 8 to 15 characters containing at least one digit and one special symbol"

Theory:

1. Exception handling in java

   An exception (or exceptional event) is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

   1. An exception can occur for many different reasons. Following are some scenarios where an exception occurs.
   2. A user has entered an invalid data.
   3. A file that needs to be opened cannot be found.
   4. A network connection has been lost in the middle of communications or the JVM has run out of memory.

2. Create and throw user defined or custom exceptions

   To display the message override the toString() method or, call the superclass parameterized constructor by passing the message in String format.

```
AgeDoesnotMatchException(String msg){
        super(msg);
}
```

   Or,

```
public String toString(){
        return "CustomException[Age is not between 17 and 24]";
}
```

Then, in other classes wherever you need this exception to be raised, create an object of the created custom exception class and, throw the exception using the throw keyword.

```
MyException ex = new MyException ();
If(condition…….){
  throw ex;
}
```

3. Advantages of exception handling

1. Separating Error-Handling Code from "Regular" Code
2. Propagating Errors Up the Call Stack
3. Grouping and Differentiating Error Types
4. Meaningful Error Reporting

// **code**

```
import java.util.*;
import java.lang.*;


class InvalidPasswordException extends Exception {
   // data member
   int conditionViolated = 0;
   // constructor
   public InvalidPasswordException(int condition) {
      super("Invalic Password: ");
      conditionViolated = condition;
   }

   public String printMessage() {
      switch (conditionViolated) {
         // length
         case 1:
```

```
            return ("Password length should be between 8 to 15 characters");
        // spaces
        case 2:
            return ("Password should not contain SPACES");
        // digit
        case 3:
            return ("Password should contain at least one DIGIT");
        // special characters
        case 4:
            return ("Password should contain at least one SPECIAL CHARACTER");
        // capital alphabet
        case 5:
            return ("Password must contain at least one CAPTIAL alphabet");
        // small alphabet
        case 6:
            return ("Password must contain at least one SMALL alphabet");
      }
      return("");
   }
}

public class PasswordChecker {
   public static void isValid(String password) throws InvalidPasswordException  {
      // password length between 8-15
      if (!(password.length()>=8 && password.length()<=15)) {
         throw new InvalidPasswordException(1);
      }

      // check for presence of spaces
      if (password.contains(" ")) {
         throw new InvalidPasswordException(2);
      }

      // check for presence of digit
      if (true) {
```

```
    boolean flag = false;
    for (int i=0 ; i<=10 ; i++) {
        String number = Integer.toString(i);
        if (password.contains(number)) {
            flag = true;
        }
    }
    if (flag == false) {
        throw new InvalidPasswordException(3);
    }
}

// checks for special characters
if (!(password.contains("@") || password.contains("#")
    || password.contains("!") || password.contains("~")
    || password.contains("$") || password.contains("%")
    || password.contains("^") || password.contains("&")
    || password.contains("*") || password.contains("(")
    || password.contains(")") || password.contains("-")
    || password.contains("+") || password.contains("/")
    || password.contains(":") || password.contains(".")
    || password.contains(",") || password.contains("<")
    || password.contains(">") || password.contains("?")
    || password.contains("|"))) {
    throw new InvalidPasswordException(4);
}

// check for capital letters
if (true) {
    boolean flag = false;
    for (int i=65 ; i<=90 ; i++) {
        char alphabet = (char)i;
        String str = Character.toString(alphabet);
        if (password.contains(str)) {
            flag = true;
```

```
                break;
            }
        }
        if (flag == false) {
            throw new InvalidPasswordException(5);
        }
    }

    // check for small letter
    if (true) {
        boolean flag = false;
        for (int i=97 ; i<=122 ; i++) {
            char alphabet = (char)i;
            String str = Character.toString(alphabet);
            if (password.contains(str)) {
                flag = true;
                break;
            }
        }
        if (flag == false) {
            throw new InvalidPasswordException(6);
        }
    }
}

public static void main(String args[]) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter Password :  ");
    String password = sc.nextLine();
    try {
        System.out.println("Is password "+password+" valid?");
        isValid(password);
        System.out.println("Password is Valid!!");
    }
    catch (InvalidPasswordException e) {
```

```
            System.out.println(e.getMessage());
            System.out.println(e.printMessage());
        }
    }
}
```

// output

```
E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 08\Code>javac PasswordChecker.java

E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 08\Code>java PasswordChecker
Enter Password :  !@Wifi34
Is password !@Wifi34 valid?
Password is Valid!!

E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 08\Code>java PasswordChecker
Enter Password :  hello,there
Is password hello,there valid?
Invalic Password:
Password should contain at least one DIGIT

E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 08\Code>java PasswordChecker
Enter Password :  Wifi34as
Is password Wifi34as valid?
Invalic Password:
Password should contain at least one SPECIAL CHARACTER
```

OOPM Lab
Lab Assingment number 09

Name: Aamir Ansari **Batch:** A **Roll no.** 01

Aim: Write a JAVA program to demonstrate the concept of multithreading and synchronization.

Problem statement:

Create two threads such that one thread will print first 10 even natural numbers (0,2,4..) and another will print first 10 odd natural numbers(1,3,5..) in an ordered fashion.

To implement this you can use any of the following:

i. Synchronized method
ii. Synchronized block
iii. Static synchronization

Theory:

1. What is thread? How is it different from process?
   - **a.** A thread is a series of executed statements
   - **b.** A thread is a nested sequence of method calls
   - **c.** Its shares memory, files and per-process state
   - **d.** Process means any program is in execution. Process control block controls the operation of any process. Process control block contains information about processes for example Process priority, process id, process state, CPU, register, etc. A process can creates other processes which are known as Child Processes. Process takes more time to terminate and it is isolated means it does not share memory with any other process, while

     Thread is the segment of a process means a process can have multiple threads and these multiple threads are contained within a process. A thread have 3 states: running, ready, and blocked.

     Thread takes less time to terminate as compared to process and like process threads do not isolate.

2. What are the ways to implement multithreading? Give syntax.

   There are two ways to create a thread.

   - **a.** It can be created by extending the Thread class and overriding its run() method:

```
public class MyClass extends Thread {
    public void run() {
        System.out.println("This code is running in a thread");
    }
}
```

   - **b.** Another way to create a thread is to implement the Runnable interface:

```
public class MyClass implements Runnable {
```

```
    public void run() {
        System.out.println("This code is running in a thread");
    }
}
```

3. Define synchronization.
   a. Synchronization in java is the capability to control the access of multiple threads to any shared resource. Java Synchronization is better option where we want to allow only one thread to access the shared resource.
   b. The synchronization is mainly used to
      1. To prevent thread interference.
      2. To prevent consistency problem.

4. How to achieve synchronization of threads?

   There are two types of thread synchronization mutual exclusive and inter-thread communication.

   a. Mutual Exclusive
      1. Synchronized method.
      2. Synchronized block.
      3. static synchronization.
   b. Cooperation (Inter-thread communication in java)

   A) Synchronised method

   To make a method synchronized, add the synchronized keyword to its declaration

   B) Synchronised block

   You do not have to synchronize a whole method. Sometimes it is preferable to synchronize only part of a method. Java synchronized blocks inside methods makes this possible.

   C) Static Synchronisation

   Synchronized static methods are synchronized on the class object of the class the synchronized static method belongs to. Since only one class object exists in the Java VM per class, only one thread can execute inside a static synchronized method in the same class

5. Advantages of Multithreading.
   a. Improved throughput. Many concurrent compute operations and I/O requests within a single process.
   b. Simultaneous and fully symmetric use of multiple processors for computation and I/O
   c. Superior application responsiveness. If a request can be launched on its own thread, applications do not freeze or show the "hourglass". An entire application will not block, or otherwise wait, pending the completion of another request.
   d. Improved server responsiveness. Large or complex requests or slow clients don't block other requests for service. The overall throughput of the server is much greater.

e. Minimized system resource usage. Threads impose minimal impact on system resources. Threads require less overhead to create, maintain, and manage than a traditional process.

Program:

```
//code
import java.util.*;
import java.lang.*;

class Numbers {
   synchronized void printNumbers(int n) {
      for (int i=1 ; i<=10 ; i++) {
         System.out.println(n++);
         n++;
         try {
            Thread.sleep(400);
         }
         catch (Exception e) {
            System.out.println(e);
         }
      }
   }
}

class MyThread1 extends Thread {
   Numbers t;
   public MyThread1(Numbers t) {
      this.t = t;
   }
   public void run() {
      t.printNumbers(0);
   }
}

class MyThread2 extends Thread {
   Numbers t;
   public MyThread2(Numbers t) {
      this.t = t;
   }
   public void run() {
      t.printNumbers(1);
   }
}

public class ImplementationOfThreads {
   public static void main(String args[]) {
      Numbers obj = new Numbers();
      MyThread1 t1 = new MyThread1(obj);
      MyThread2 t2 = new MyThread2(obj);
      t1.start();
      t2.start();
   }
```

}

// output

```
E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 09\Code>javac ImplementationOfThreads.java

E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 09\Code>java ImplementationOfThreads
0
2
4
6
8
10
12
14
16
18
1
3
5
7
9
11
13
15
17
19

E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 09\Code>java ImplementationOfThreads
0
2
4
6
8
10
12
14
16
18
1
3
5
7
9
11
13
15
17
19

E:\Sem-3\LabWork - Assignments\OOPM\Lab Assignment 09\Code>
```

JAVA Lab
Lab Experiment No 10

Name: Aamir Ansari **Batch:** A **Roll no:** 01

Aim: To write a Java program to implement AWT components.

Problem statement:
Write a program to create a window/frame with four text fields for the name, street, city and pincode with suitable labels. Also the window/frame contains a button MyInfo. When the user types the name, his street, city and pincode ,then clicks the button, the typed details must appear in Arial Font with Size 32, Italics.

Theory:

What is AWT?
AWT stands for Abstract Window Toolkit. It is a platform-dependent API to develop GUI (Graphical User Interface) or window-based applications in Java. It is heavy-weight in use because it is generated by the system's host operating system. It contains a large number of classes and methods, which are used for creating and managing GUI.

What is the class hierarchy of AWT?
All the elements like buttons, text fields, scrollbars etc are known as components. In AWT we have classes for each component .To have everything placed on a screen to a particular position, we have to add them to a container. A container is like a screen wherein we are placing components like buttons, text fields, checkbox etc. In short a container contains and controls the layout of components. A container itself is a component thus we can add a container inside container.

What is a Layout manager? What are the different types of Layout managers?
The Layout managers enable us to control the way in which visual components are arranged in the GUI forms by determining the size and position of components within the containers.
Types of Layout Manager:
Flow Layout: It arranges the components in a container like the words on a page. It fills the top line from left to right and top to bottom. The components are arranged in the order as they are added i.e. first components appears at top left, if the container is not wide enough to display all the components, it is wrapped around the line. Vertical and horizontal gap between components can be controlled. The components can be left, centre or right aligned.
Border Layout: It arranges all the components along the edges or the middle of the container i.e. top, bottom, right and left edges of the area. The components added to the top or bottom gets its preferred height, but its width will be the width of the container and also the components added to the left or right gets its preferred width, but its height will be the remaining height of the container. The components added to the centre gets neither its preferred height or width. It covers the remaining area of the container.
Grid Layout: It arranges all the components in a grid of equally sized cells, adding them from the left to right and top to bottom. Only one component can be placed in a cell and each region of the grid will have the same size. When the container is resized, all cells are automatically resized. The order of placing the components in a cell is determined as they were added.

Advantages and disadvantages of using Layout managers.
Advantage: The layout manager automatically positions all the components within the container.
Java provide us with various layout manager to position the controls.. When the size of the applet or the application window changes the size, shape and arrangement of the components also changes in response i.e. the layout managers adapt to the dimensions of appletviewer or the application window.
The layout manager is associated with every Container object.
Disadvantage: If you use a Layout Manager, you can no longer change the size and location of a Component through the setSize and setLocation methods.

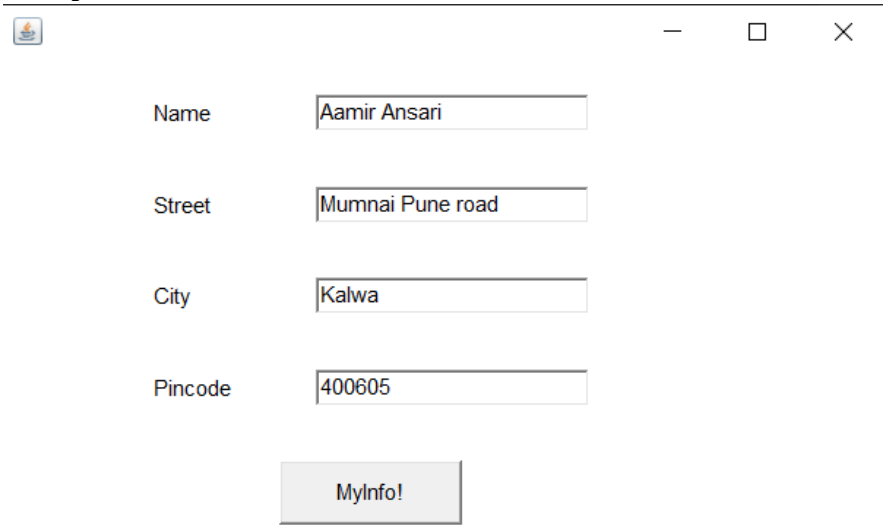
Program:

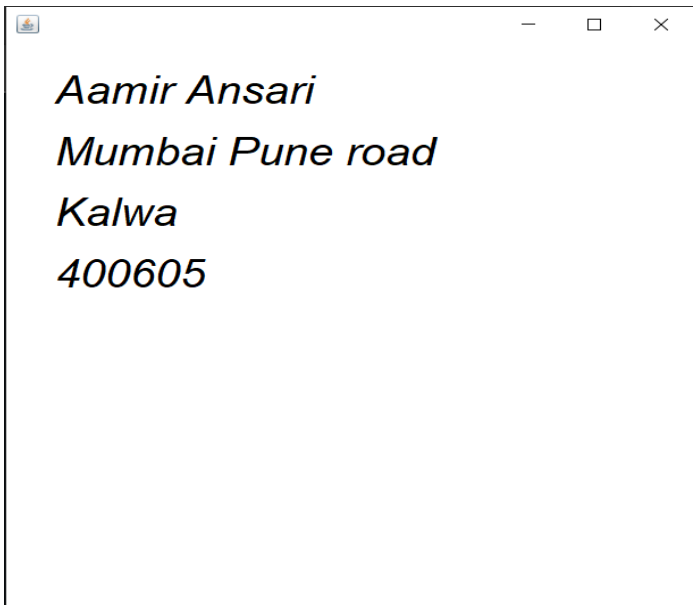
```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class DisplayInfo extends Frame implements ActionListener {
	private static final long serialVersionUID = 1L;

	TextField nameText = new TextField();
	TextField streetText = new TextField();
	TextField cityText = new TextField();
	TextField pincodeText = new TextField();

	DisplayInfo() {

		Label nameLabel = new Label("Name");
		Label streetLabel = new Label("Street");
		Label cityLabel = new Label("City");
		Label pincodeLabel = new Label("Pincode");

		nameLabel.setBounds(90, 50, 50, 20);
		streetLabel.setBounds(90, 100, 50, 20);
		cityLabel.setBounds(90, 150, 50, 20);
		pincodeLabel.setBounds(90, 200, 50, 20);

		nameText.setBounds(180, 50, 150, 20);
		streetText.setBounds(180, 100, 150, 20);
		cityText.setBounds(180, 150, 150, 20);
		pincodeText.setBounds(180, 200, 150, 20);

		Button button = new Button("MyInfo!");
		button.setBounds(160, 250, 100, 35);
		button.addActionListener(this);

		add(nameLabel);
		add(nameText);
		add(streetLabel);
		add(streetText);
		add(cityLabel);
		add(cityText);
		add(pincodeText);
		add(pincodeLabel);

		add(button);
```

```
            setSize(500, 500);
            setLayout(null);
            setVisible(true);

        }

        public static void main(String[] args) {

            new DisplayInfo();

        }

        @Override
        public void actionPerformed(ActionEvent e) {

            String name = nameText.getText();
            String street = streetText.getText();
            String city = cityText.getText();
            String pincode = pincodeText.getText();

            // redirects to Info
            new Info(name, street, city, pincode);
            dispose();
        }
}

public class Info extends Frame {
        private static final long serialVersionUID = 1L;

        Info (String name, String street, String city, String pincode) {

            Font myFont = new Font("Arial",Font.ITALIC,32);

            Label nameLabel = new Label(name);
            nameLabel.setFont(myFont);
            nameLabel.setBounds(40, 50, 400, 40);

            Label streetLabel = new Label(street);
            streetLabel.setFont(myFont);
            streetLabel.setBounds(40, 100, 400, 40);

            Label cityLabel = new Label(city);
            cityLabel.setFont(myFont);
            cityLabel.setBounds(40, 150, 400, 40);

            Label pincodeLabel = new Label(pincode);
            pincodeLabel.setFont(myFont);
            pincodeLabel.setBounds(40, 200, 400, 40);

            add(nameLabel);
            add(streetLabel);
            add(cityLabel);
```

```
            add(pincodeLabel);

            setSize(500, 500);
            setLayout(null);
            setVisible(true);

        }

}
```

// output





JAVA Lab
Lab Experiment No 11

Name: Aamir Ansari **Batch:** A **Roll no:** 01

Aim: To write a Java program to illustrate the usage of built in layout manager.

Problem statement:

Write a Java program to create a simple calculator using java AWT elements. Use a grid layout to arrange buttons for the digits and basic operation +, -, /, *. Add a text field to display the results.

Theory:

Explain grid layout manager in detail.

Grid Layout class represents a layout manager with a specified number of rows and columns in a rectangular grid. The Grid Layout container is divided into an equal-sized of rectangles, and one of the components is placed in each rectangle. Every rectangle cell has the same size therefore, they contain a component, which fills the entire cell. When the user changes or adjusts the size of the container, the size of each rectangles changes accordingly.

Constructors of the class:

GridLayout(): It Creates a grid layout with a default of one column per component, in a single row.

GridLayout(int rw, int cl): It creates a grid layout with the specified number of rows and columns.

GridLayout(int rw, int cl, int hgap, int vgap): It creates a grid layout with the specified number of rows and columns with horizontal and vertical gap.

Commonly Used Methods:

addLayoutComponent(String str, Component cmp): Adds the specified component with the specified name to the layout.

setColumns(int cl): Sets the number of columns in this layout to the specified value.

layoutContainer(Container pr): Lays out the specified container using this layout.

toString(): Returns the string representation of this grid layout's values.

How to add buttons and handle the event of clicking the buttons i.e. explain listeners.

To add button in the java we follow the syntax

Button name_button = new Button("Label_on_button");

The event listener is the feature of java that handles the several events for the several objects. Classes for helping in implementing event listeners are present in the java.awt.event.*; package. All the operations are controlled by the events generated by these buttons.

To handle the events generated by these buttons you add action listeners

e.g. button_name.addActionListener(this);.

When the action event occurs, that object's actionPerformed method is invoked.

```
actionPerformed(ActionEvent e)
{
        //code
}
```

Program:

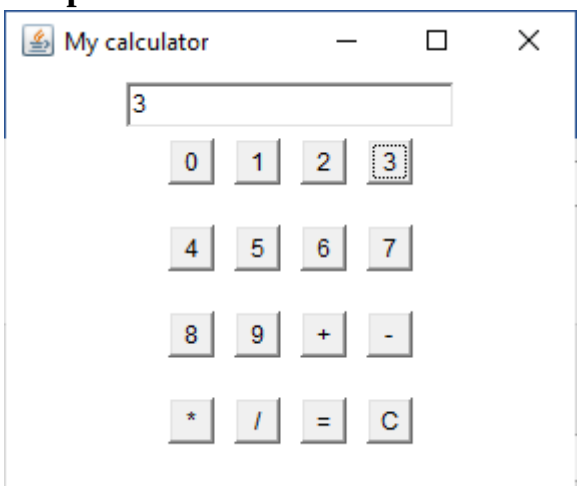
```
import java.awt.*;
import java.awt.event.*;
public class calculator implements ActionListener
{
   int c,n;
   String s1,s2,s3,s4,s5;
   Frame f;
   Button b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12,b13,b14,b15,b16,b17;
   Panel p;
   TextField tf;
   GridLayout g;
   calculator()
   {
      f = new Frame("My calculator");
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.setLayout(new FlowLayout());
      p = new Panel();

      //Assigning buttons
      b1 = new Button("0");
      b1.addActionListener(this);
      b2 = new Button("1");
      b2.addActionListener(this);
      b3 = new Button("2");
      b3.addActionListener(this);
      b4 = new Button("3");
      b4.addActionListener(this);
      b5 = new Button("4");
      b5.addActionListener(this);
      b6 = new Button("5");
      b6.addActionListener(this);
      b7 = new Button("6");
      b7.addActionListener(this);
      b8 = new Button("7");
      b8.addActionListener(this);
      b9 = new Button("8");
      b9.addActionListener(this);
      b10 = new Button("9");
      b10.addActionListener(this);
      b11 = new Button("+");
      b11.addActionListener(this);
      b12 = new Button("-");
      b12.addActionListener(this);
      b13 = new Button("*");
      b13.addActionListener(this);
      b14 = new Button("/");
      b14.addActionListener(this);
      b15 = new Button("=");
      b15.addActionListener(this);
      b16 = new Button("C");
      b16.addActionListener(this);
```

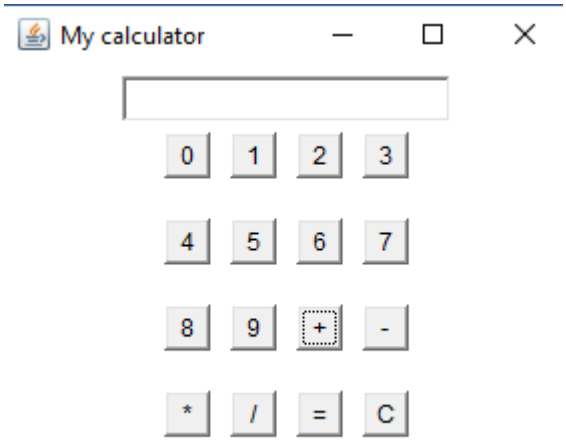
```
    //Text field to display
    tf = new TextField(20);
    f.add(tf);
    //Setting the layout
    g = new GridLayout(4,4,10,20);
    p.setLayout(g);
    //Adding buttons to it
    p.add(b1);p.add(b2);p.add(b3);p.add(b4);p.add(b5);p.add(b6);p.add(b7);p.add(b8);p.add(b9);
    p.add(b10);p.add(b11);p.add(b12);p.add(b13);p.add(b14);p.add(b15);p.add(b16);
    f.add(p); f.setSize(300,300);  f.setVisible(true);
}
public void actionPerformed(ActionEvent e)
{
    //Performing calculations
    if(e.getSource()==b1)
    {
        s3 = tf.getText();
        s4 = "0";
        s5 = s3+s4;
        tf.setText(s5);
    }
    if(e.getSource()==b2)
    {
        s3 = tf.getText();
        s4 = "1";
        s5 = s3+s4;
        tf.setText(s5);
    }
    if(e.getSource()==b3)
    {
        s3 = tf.getText();
        s4 = "2";
        s5 = s3+s4;
        tf.setText(s5);
    }
    if(e.getSource()==b4)
    {
        s3 = tf.getText();
        s4 = "3";
        s5 = s3+s4;
        tf.setText(s5);
    }
    if(e.getSource()==b5)
    {
        s3 = tf.getText();
        s4 = "4";
        s5 = s3+s4;
        tf.setText(s5);
    }
    if(e.getSource()==b6)
    {
        s3 = tf.getText();
```

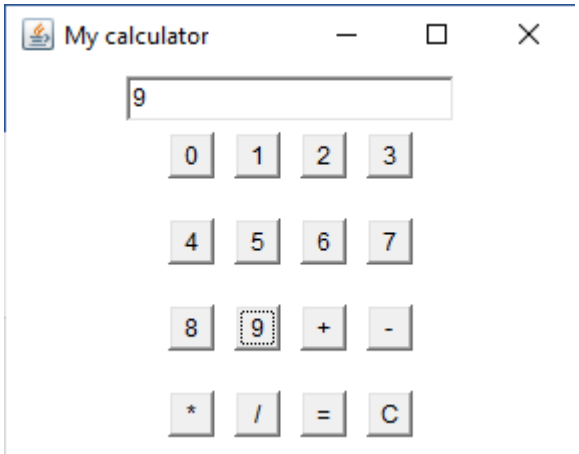
```
    s4 = "5";
    s5 = s3+s4;
    tf.setText(s5);
}
if(e.getSource()==b7)
{
    s3 = tf.getText();
    s4 = "6";
    s5 = s3+s4;
    tf.setText(s5);
}
if(e.getSource()==b8)
{
    s3 = tf.getText();
    s4 = "7";
    s5 = s3+s4;
    tf.setText(s5);
}
if(e.getSource()==b9)
{
    s3 = tf.getText();
    s4 = "8";
    s5 = s3+s4;
    tf.setText(s5);
}
if(e.getSource()==b10)
{
    s3 = tf.getText();
    s4 = "9";
    s5 = s3+s4;
    tf.setText(s5);
}
if(e.getSource()==b11)
{
    s1 = tf.getText();
    tf.setText("");
    c=1;
}
 if(e.getSource()==b12)
{
    s1 = tf.getText();
    tf.setText("");
    c=2;
}
if(e.getSource()==b13)
{
    s1 = tf.getText();
    tf.setText("");
    c=3;
}
if(e.getSource()==b14)
{
    s1 = tf.getText();
     tf.setText("");
```

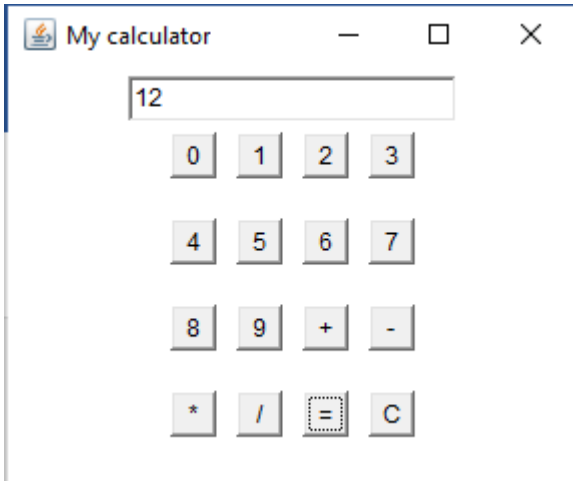
```
            c=4;
        }
        if(e.getSource()==b15)
        {
            s2 = tf.getText();
            if(c==1)
            {
                n = Integer.parseInt(s1)+Integer.parseInt(s2);
                tf.setText(String.valueOf(n));
            }
            else
            if(c==2)
            {
                n = Integer.parseInt(s1)-Integer.parseInt(s2);
                tf.setText(String.valueOf(n));
            }
            else
            if(c==3)
            {
                n = Integer.parseInt(s1)*Integer.parseInt(s2);
                tf.setText(String.valueOf(n));
            }
            if(c==4)
            {
                try
                {
                    int p=Integer.parseInt(s2);
                    if(p!=0)
                    {
                        n = Integer.parseInt(s1)/Integer.parseInt(s2);
                        tf.setText(String.valueOf(n));
                     }
                     else
                       tf.setText("infinite");
                }
                catch(Exception i){}
             }
             if(c==5)
             {
                 n = Integer.parseInt(s1)%Integer.parseInt(s2);
                 tf.setText(String.valueOf(n));
             }
        }
        if(e.getSource()==b16)
        {
            tf.setText("");
        }
    }
    public static void main(String[] abc)
    {
        calculator v = new calculator();
    }
}
```

Output:


My calculator
3
0 1 2 3
4 5 6 7
8 9 + -
* / = C


My calculator
0 1 2 3
4 5 6 7
8 9 + -
* / = C


My calculator
9
0 1 2 3
4 5 6 7
8 9 + -
* / = C


My calculator
12
0 1 2 3
4 5 6 7
8 9 + -
* / = C

JAVA Lab
Lab Experiment No 12

Name: Aamir Ansari **Batch:** A **Roll no:** 01

Aim: Java program to illustrate the usage of one of the swing components.
Problem statement: Write a Java program to take user response using JCheckBox component from swing.

Theory:
Event handling:
Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through a keyboard, selecting an item from a list, scrolling the page are the activities that cause an event to happen.

Types of Event
The events can be broadly classified into two categories:

Foreground Events - Those events which require the direct interaction of the user.They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through the keyboard, selecting an item from a list, scrolling the page etc.

Background Events - Those events that require the interaction of the end user are known as background events. Operating system interrupts, hardware or software failure, timer expires, an operation completion are the examples of background events.

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has the code which is known as an event handler that is executed when an event occurs.

Event handling has the following key participants namely:

Source - The source is an object on which event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provides classes for source objects.

Listener - It is also known as event handler. Listeners are responsible for generating responses to an event. From java implementation point of view the listener is also an object. Listener waits until it receives an event. Once the event is received, the listener processes the event and then returns.
The benefit of this approach is that the user interface logic is completely separated from the logic that generates the event. The user interface element is able to delegate the processing of an event to the separate piece of code. In this model, Listener needs to be registered with the source object so that the listener can receive the event notification. This is an efficient way of handling the event because the event notifications are sent only to those listener that want to receive them.

Steps involved in event handling

1. The User clicks the button and the event is generated.

2. Now the object of the concerned event class is created automatically and information about the source and the event get populated within the same object.
3. Event object is forwarded to the method of registered listener class.
4. The method is now executed and returns.

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener

ContainerEvent	ContainerListener
FocusEvent	FocusListener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

Button

```
public void addActionListener(ActionListener a){}
```

MenuItem

```
public void addActionListener(ActionListener a){}
```

TextField

```
public void addActionListener(ActionListener a){}
public void addTextListener(TextListener a){}
```

TextArea

```
public void addTextListener(TextListener a){}
```

Checkbox

```
public void addItemListener(ItemListener a){}
```

Choice

```
public void addItemListener(ItemListener a){}
```

List

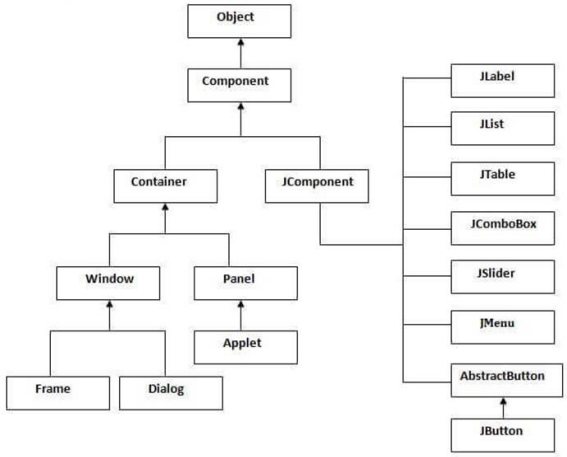
```
public void addActionListener(ActionListener a){}
public void addItemListener(ItemListener a){}
```

Java Swing

1. Java Swing is a part of Java Foundation Classes (JFC) that is used to create window-based applications.

2. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
3. Unlike AWT, Java Swing provides platform-independent and lightweight components.
4. The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Swing class Hierarchy



Difference between AWT and Swing

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent.**	Java swing components are **platform-independent.**
2)	AWT components are **heavyweight.**	Swing components are **lightweight.**
3)	AWT **doesn't support pluggable look and feel.**	Swing **supports pluggable look and feel.**
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC.**

Features of Swing

Light Weight − Swing components are independent of native Operating System's API as Swing API controls are rendered mostly using pure JAVA code instead of underlying operating system calls.

Rich Controls − Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, colorpicker, and table controls.

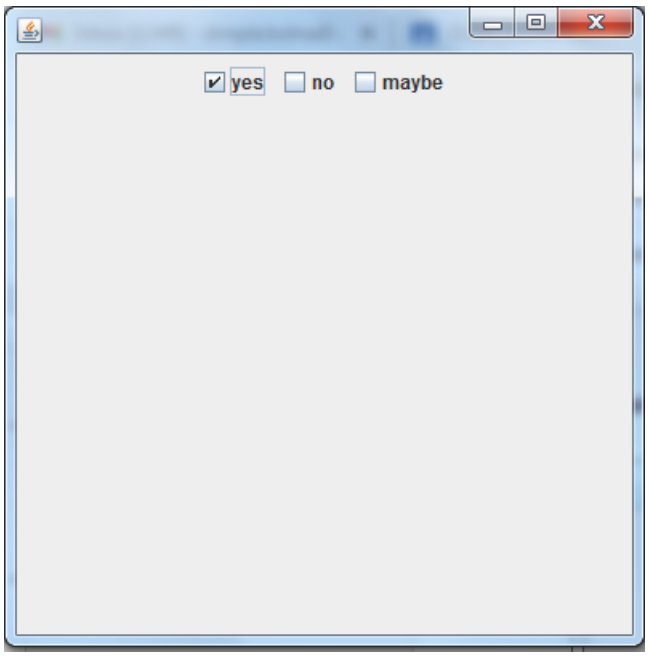
Highly Customizable − Swing controls can be customized in a very easy way as visual appearance is independent of internal representation.

Pluggable look-and-feel − SWING based GUI Application look and feel can be changed at run-time, based on available values.

Program:

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class Test_swing extends JFrame
{
  public Test_swing()
  {
    JCheckBox jcb = new JCheckBox("yes");   //creating JCheckBox.
    add(jcb);                               //adding JCheckBox to frame.
    jcb = new JCheckBox("no");              //creating JCheckBox.
    add(jcb);                               //adding JCheckBox to frame.
    jcb = new JCheckBox("maybe");           //creating JCheckBox.
    add(jcb);                               //adding JCheckBox to frame.
    setLayout(new FlowLayout());
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(400, 400);
    setVisible(true);
  }
  public static void main(String[] args)
  {
    new Test_swing();
  }
}
```

Output:

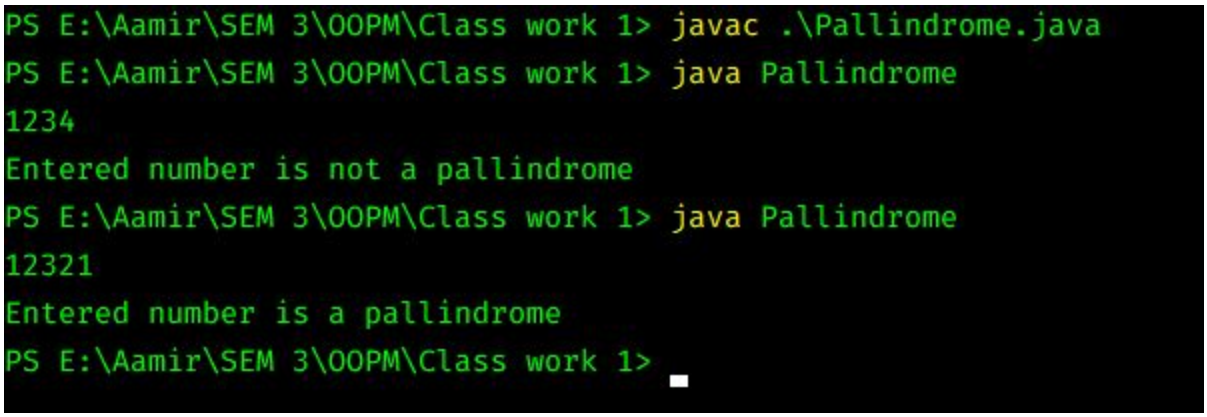

yes
no
maybe

WAP to check whether the entered number is palindrome or not (using if..else)

//CODE

```
import java.util.*;

class Palindrome {
	public static void main(String args[]) {
		int num, rev=0, temp;
		Scanner sc = new Scanner(System.in);
		num = sc.nextInt();
		temp = num;
		while (temp != 0) {
			rev = rev * 10;
			rev += temp % 10;
			temp = temp/10;
		}

		if (rev == num) {
			System.out.println("Entered number is a palindrome");
		} else {
			System.out.println("Entered number is not a palindrome");
		}
	}
}
```

//OUTPUT



Input the length of the three sides of a valid triangle in s1,s2,s3. Verify whether the triangle is Equilateral or Isosceles or Scalene. (Using ladder of if..else)

//CODE

```
import java.util.*;

class Triangle {
	public static void main(String args[]) {
		int s1, s2, s3;
		Scanner sc = new Scanner(System.in);
		s1 = sc.nextInt();
		s2 = sc.nextInt();
		s3 = sc.nextInt();

		if (s1 == s2 && s2 == s3) {
			System.out.println("The Triangle is EQUILATERAL");
		} else if ((s1==s2) || (s2==s3) || (s3==s1)) {
			System.out.println("The Triangle is ISOSCELES");
		} else {
			System.out.println("The Triangle is SCALENE");
		}
	}
}
```

//OUTPUT

```
PS E:\Aamir\SEM 3\OOPM\Class work 1> javac .\Triangle.java
PS E:\Aamir\SEM 3\OOPM\Class work 1> java Triangle
10 10 10
The Triangle is EQUILATERAL
PS E:\Aamir\SEM 3\OOPM\Class work 1> java Triangle
9 5 5
The Triangle is ISOSCELES
PS E:\Aamir\SEM 3\OOPM\Class work 1> java Triangle
6 2 4
The Triangle is SCALENE
PS E:\Aamir\SEM 3\OOPM\Class work 1>
```

A student would get a rank of different status depending upon the number of subjects in which the student scored distinction marks.

//CODE

```
import java.util.*;

class Grades {
	public static void main (String args[]) {
		int num;
		Scanner sc = new Scanner(System.in);
		num = sc.nextInt();
		switch(num) {
			case 0:
				System.out.println("GOOD");
				break;
			case 1:
				System.out.println("VIVID");
				break;
			case 2:
				System.out.println("BRIGHT");
				break;
			case 3:
				System.out.println("BRILLIANT");
				break;
			case 4:
				System.out.println("OUTSTANDING");
				break;
			case 5:
				System.out.println("EXCELLENT");
				break;
			default:
				System.out.println("NOT VALID");
		}
	}
```

}

//OUTPUT

```
PS E:\Aamir\SEM 3\OOPM\Class work 1> javac .\Grade.java
PS E:\Aamir\SEM 3\OOPM\Class work 1> java Grades
0
GOOD
PS E:\Aamir\SEM 3\OOPM\Class work 1> java Grades
1
VIVID
PS E:\Aamir\SEM 3\OOPM\Class work 1> java Grades
2
BRIGHT
PS E:\Aamir\SEM 3\OOPM\Class work 1> java Grades
3
BRILLIANT
PS E:\Aamir\SEM 3\OOPM\Class work 1> java Grades
4
OUTSTANDING
PS E:\Aamir\SEM 3\OOPM\Class work 1> java Grades
5
EXCELLENT
PS E:\Aamir\SEM 3\OOPM\Class work 1>
```

Name: Aamir Ansari **Roll no.** 01

Program based on the concept of Jagged array.

```
//code

import java.util.*;

class Friends {

   public static void main(String args[]) {

      Scanner sc = new Scanner(System.in);
      String hobbies[][] = new String[10][];

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

         System.out.print("Enter number of hobbies of Friend "+(i+1)+" :  ");
         int numHobbies = sc.nextInt();
         sc.nextLine();
         hobbies[i] = new String[numHobbies];

         for (int j=0 ; j<numHobbies ; j++) {

            System.out.print("Enter hobby number "+(j+1)+" :  ");
            hobbies[i][j] = sc.nextLine();
            //sc.nextLine();

         }

      }

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

         System.out.println();
         System.out.print("Hobbies of Friend "+(i+1)+" :  ");
```

```
        for (int j=0 ; j<hobbies[i].length ; j++) {

            System.out.print(hobbies[i][j] + ",  ");

        }

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

    }

  }

}
```

//output

```
E:\Aamir\Sem-3\OOPM\Jagged Array>javac Friends.java

E:\Aamir\Sem-3\OOPM\Jagged Array>java Friends
Enter number of hobbies of Friend 1 :  2
Enter hobby number 1 :  Football
Enter hobby number 2 :  Music
Enter number of hobbies of Friend 2 :  3
Enter hobby number 1 :  Eating
Enter hobby number 2 :  Also eating
Enter hobby number 3 :  Eating yet again
Enter number of hobbies of Friend 3 :  1
Enter hobby number 1 :  Moon walking
Enter number of hobbies of Friend 4 :  2
Enter hobby number 1 :  Games
Enter hobby number 2 :  Movies
Enter number of hobbies of Friend 5 :  1
Enter hobby number 1 :  Existing
Enter number of hobbies of Friend 6 :  2
Enter hobby number 1 :  Thinking
Enter hobby number 2 :  Being useless
Enter number of hobbies of Friend 7 :  2
Enter hobby number 1 :  Sitting
Enter hobby number 2 :  Sitting alone
Enter number of hobbies of Friend 8 :  1
Enter hobby number 1 :  Running
Enter number of hobbies of Friend 9 :  2
Enter hobby number 1 :  Star gazing
Enter hobby number 2 :  doing summer sault
Enter number of hobbies of Friend 10 :  1
Enter hobby number 1 :  Thinking that life means something
```

```
Hobbies of Friend 1 :  Football,  Music,
*****************************************

Hobbies of Friend 2 :  Eating,  Also eating,  Eating yet again,
*****************************************

Hobbies of Friend 3 :  Moon walking,
*****************************************

Hobbies of Friend 4 :  Games,  Movies,
*****************************************

Hobbies of Friend 5 :  Existing,
*****************************************

Hobbies of Friend 6 :  Thinking,  Being useless,
*****************************************

Hobbies of Friend 7 :  Sitting ,  Sitting alone,
*****************************************

Hobbies of Friend 8 :  Running,
*****************************************

Hobbies of Friend 9 :  Star gazing,  doing summer sault,
*****************************************

Hobbies of Friend 10 :  Thinking that life means something,
*****************************************

E:\Aamir\Sem-3\OOPM\Jagged Array>
```