EXPERIMENT NO. 1

Aim: Write some simple programs in Java such as:
i) To find factorial of number.
ii) To display first 50 prime numbers.
iii) To find sum and average of N numbers

Software:

1. Eclipse
2. JDK 16

Theory:

i) To find Factorial of number

Factorial Program in Java: Factorial of n is the *product of all positive descending integers*. Factorial of *n* is denoted by n!. For example:

1. 4! = 4*3*2*1 = 24
2. 5! = 5*4*3*2*1 = 120

Here, 4! is pronounced as "4 factorial", it is also called "4 bang" or "4 shriek".

The factorial is normally used in Combinations and Permutations (mathematics). There are many ways to write the factorial program in java language. There are 2 ways to write the factorial program in java.

- Java for Loop
- Java while and do...while Loop

logic

If a number Is chosen then we need to start multiplying It from 1 to that number

Lets take 5 as an example

1 * 2 *3 *4 * 5 = 120 how to find

1

1* 2=2

2*3=6

6*4=24

24*5=120

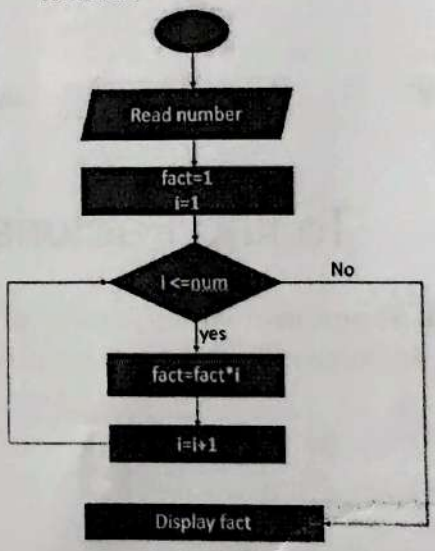
Algorithm

Step 1: start
Step 2: read number
Step 3: set fact=1, i=1
Step 4: check condition i<=number if false go to step 7
Step 5: fact=fact*i;
Step 6: update i=i+1 go to step 4
Step 7: Display fact.
Step 8: Stop

code

```
for(i=1;i<=num;i++)
{
    fact=fact*i;
}
cout<<"factorial of"<< num<< "is"<<fact;
```

Flow chart



Example 1: Find Factorial of a number using for loop

```
public class Factorial {

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

```
        int num = 10;
        long factorial = 1;
        for(int i = 1; i <= num; ++i)
        {
            // factorial = factorial * i;
            factorial *= i;
        }
        System.out.printf("Factorial of %d = %d", num, factorial);
    }
}
```

Output

```
Factorial of 10 = 3628800
```

In this program, we've used for loop to loop through all numbers between 1 and the given number *num* (10), and the product of each number till *num* is stored in a variable *factorial*. We've used *long* instead of *int* to store large results of factorial.

Example2 : Find Factorial of a number by taking from user.

```
Scanner in = new Scanner(System.in);

                System.out.println("Enter the No to calculate its
factorial");
                double no;

                no = in.nextDouble();

                double fact = 1;

                for(int i = 1; i<=no; ++i)
                {
                        fact *=i;
                }

                System.out.println("Factorial =" + fact);
        }
```

Output:

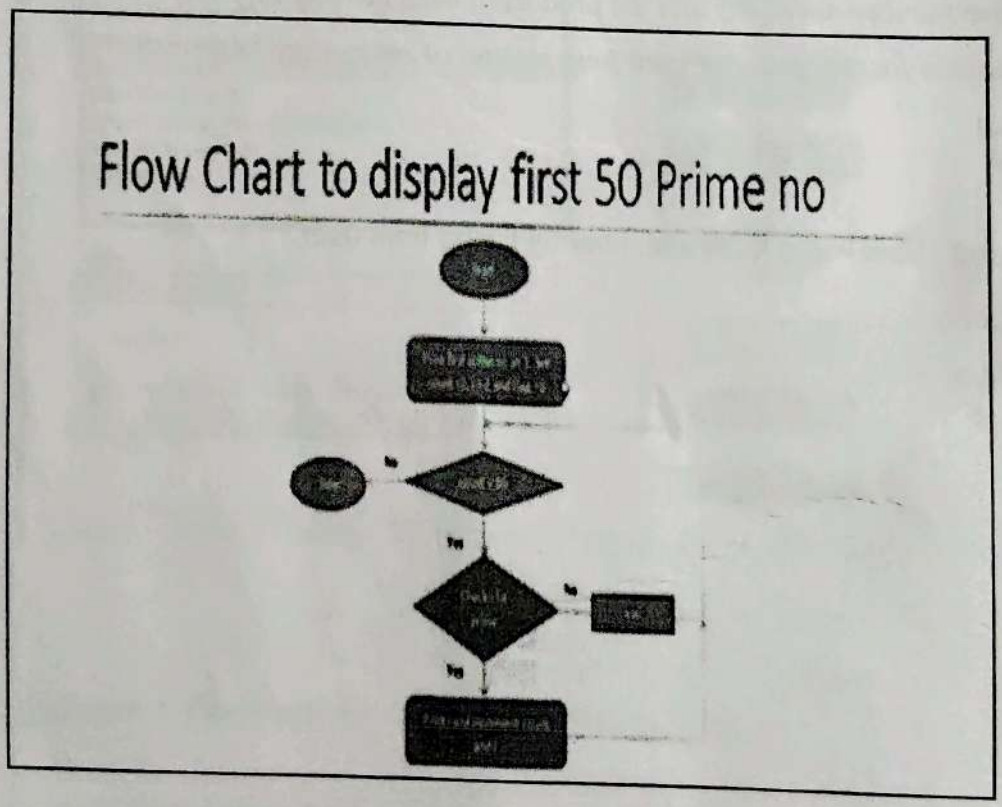
```
Enter the No to calculate its factorial =
5
Factorial =120.0
```

ii) Program to display first 50 prime numbers

Prime numbers are the natural numbers that can be divided by their self or by 1 without any remainder.

For example: 2, 3, 5, 7, 11, 13, 17 etc.

In this program, we need to print the first 50 prime numbers: **2,3,5,7,11,13,17,19,23,29.....**



Program:

```
package primeNo;

public class primeNumber {

	public static void main(String[] args) {
		int i = 1;
		int count = 1, flag =1;

		System.out.println("First Prime No :" + i );

		while(count < 50 )
		{
			++i;

			for( int j = 2; j < i/2; ++j)
			{
				if ( i%j == 0)
				{
					flag = 0;
					break;
				}
			}

			if (flag == 1)
			{
				++count;
				System.out.println(count + "th Prime No = " +
i );

			}
			else
				flag = 1;

		}


	}

}
```

Output:

```
First Prime No = 1
2th Prime No = 2
3th Prime No = 3
4th Prime No = 4
5th Prime No = 5
6th Prime No = 7
```

iii) Program to find sum and average of N numbers

We will write a two java program to Find **Sum and Average** of N numbers using an array. The first program finds the average of specified array elements. The second program takes the value of N and the numbers provided by the user and finds the average of them using an array.
To understand these programs you should have knowledge of following concepts:

- **For loop**
- Java Scanner Class

The average is the outcome from the sum of the numbers divided by the count of the numbers being averaged.

For example: 1,2,3,4,5

Number of all elements = 5

Sum of all elements = 1+2+3+4+5 =15

Average = Sum of all elements / number of all elements = 15/5 =3

Average =3

Program:

```
package primeNo;

import java.util.Scanner;

public class sumAdd {

     public static void main(String[] args) {
     Scanner in = new Scanner(System.in);

          double no,sum =0, avg;
          int n;

          System.out.print("How many numbers you want to enter");
          n = in.nextInt();
```

```
        for( int i = 1; i<= n; ++i)
        {
            System.out.print("Enter the No: ");
            no = in.nextDouble();

            sum += no;
        }

        avg = sum/n;

        System.out.println("Avg of " + n + "number is "+ avg);

    }

    }
```

Output:

```
How many numbers you want to enter2
Enter the No: 10
Enter the No: 10
Avg of 2number is 10.0
```

Conclusion:

We have learnt in the experiment to write a simple program in Java to:-

(1) Find factorial of number
(2) To display first 50 prime numbers
(3) To find sum & average of N numbers

EXPERIMENT NO. 2

Aim: Write a program in Java to implement a Calculator with simple arithmetic operations such as add, subtract, multiply, divide, factorial etc. using switch case and other simple java statements.

Software:

1. Eclipse
2. JDK 16

Theory:

In this Program we are making a simple calculator that performs addition, subtraction, multiplication and division based on the user input. The program takes the value of both the numbers (entered by user) and then user is asked to enter the operation (+, -, * and /), based on the input program performs the selected operation on the entered numbers using switch.

To understand this programming, you should have the knowledge of the following Java programming topics:

- Java switch Statement
- Java Scanner Class

- **Calculator by using do-while:**

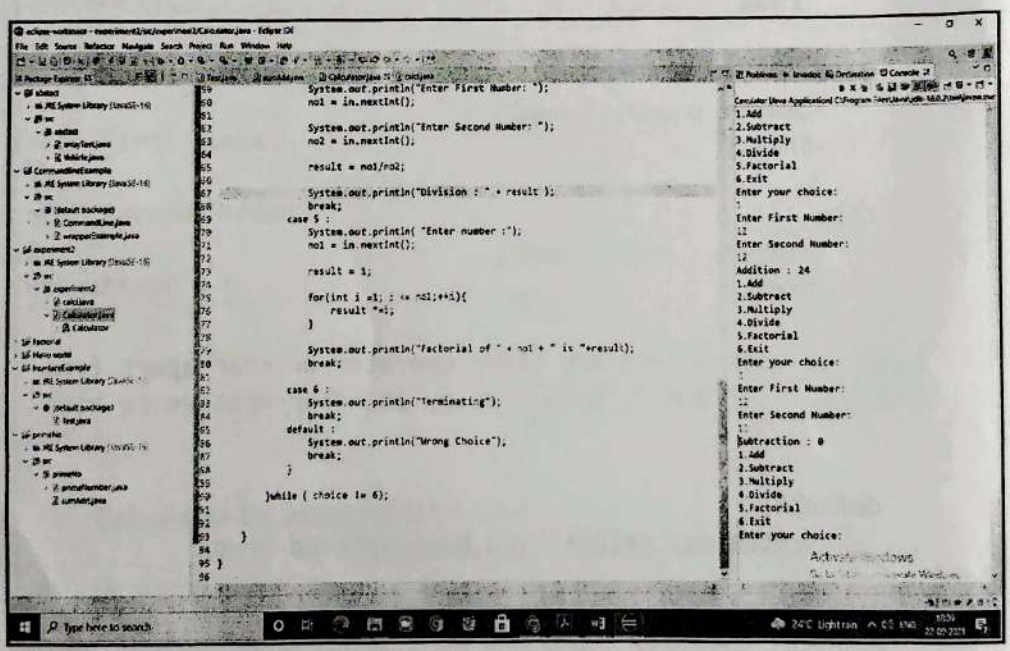
```
package experiment2;
```

```
Enter first number:40
Enter second number:4
Enter an operator (+, -, *, /): /
40.0 / 4.0: 10.0
```

Conclusion:

We have learnt in this experiment to write a program in java to implement a Calculator with simple arithmetic operations such as add, subtract, multiply, divide, factorial, etc. using switch case & other simple java statements

Screenshot's of Program and Result:



EXPERIMENT NO. 3

Aim: Write a program in Java with class Rectangle with the data fields width, length, area and colour. The length, width and area are of double type and colour is of string type. The methods are get_length(), get_width(), get_colour() and find_area(). Create two objects of Rectangle and compare their area and colour. If the area and colour both are the same for the objects then display " Matching Rectangles", otherwise display " Non-matching Rectangle"

Software:

1. Eclipse
2. JDK 16

Theory:

Java Packages & API

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. Packages are divided into two categories:

- Built-in Packages (packages from the Java API)
- User-defined Packages (create your own packages)

Built-in Packages

The Java API is a library of prewritten classes that are free to use, included in the Java Development Environment.

The library contains components for managing input, database programming, and much much more. The complete list can be found at Oracles website: https://docs.oracle.com/javase/8/docs/api/.

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.

To use a class or a package from the library, you need to use the import keyword:

Syntax

```
import package.name.Class;  // Import a single class
import package.name.*;  // Import the whole package
```

Import a Package

To import a whole package, end the sentence with an asterisk sign (*). The following example will import ALL the classes in the java.util package:

Example

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

Example:

Suppose that you want to design the class Rectangle that implements the basic properties of a Rectangle. Now every Rectangle has a width and height, area and colour, which can be a floating-point value. Therefore, when you created an object of the class Rectangle, then you must store the length and width of the Rectangle into that object. Next, the two basic operations that are performed on a Rectangle are to find the area and perimeter of the Rectangle. Thus, the class Rectangle must provide these two operations. This class needs to provide a few other operations to effectively use this class in a program. In skeleton form, the definition of the class Rectangle looks as follows:

```
public class Rectangle
{
   double length;
   double width;

   public double getArea()
   {
      // code to determine the area of the Rectangle
   }

   public double getPerimeter()
   {
      // code to determine the perimeter of the Rectangle
```

```
    }

    // Additional methods as needed ...
}
```

The **public** access modifier permits a member to be accessed by code outside the class.

Variable Declaration and Object Instantiation

Once a class is defined, you can declare reference variables of that class type. For example, the following statement declare rectangle1 to be reference variables of type Rectangle

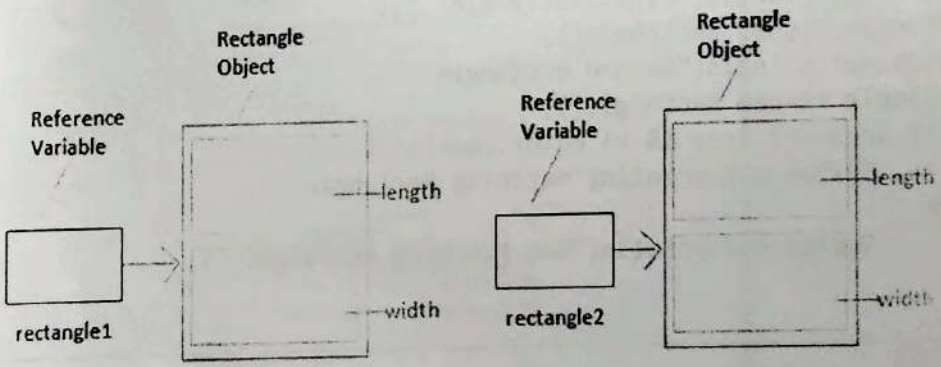
```
Rectangle rectangle1;
```

To store the length and width, we need to create a Rectangle object, which is accomplished by using the operator new :

```
rectangle1 = new Rectangle();
```

You can combine variable declaration and object instantiation in one statement

```
Rectangle rectangle 2 = new Rectangle();
```



Accessing Class Members

Once an object of a class is created, the object can access the members of the class using dot . operator :

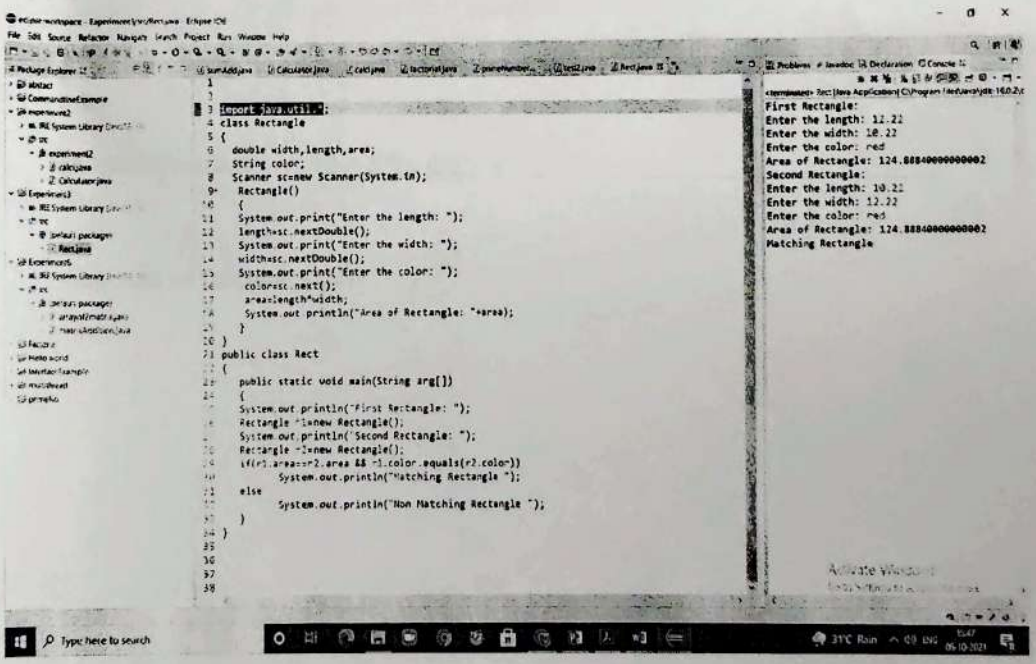
```
rectangle1.set(4.1,6.3)
```

```
Enter the color: blue
Area of Rectangle: 124.88840000000002
Matching Rectangle
```

Conclusion:

We have learnt in this experiment to write a program in java with class Rectangle with the data fields width, length, area and colour. ~~The length, width and area~~

Screenshot's of Program and Result:



EXPERIMENT NO. 4

Aim: Write a program in JAVA to demonstrate the method and constructor overloading

Software:

1. Eclipse
2. JDK 16

Theory:

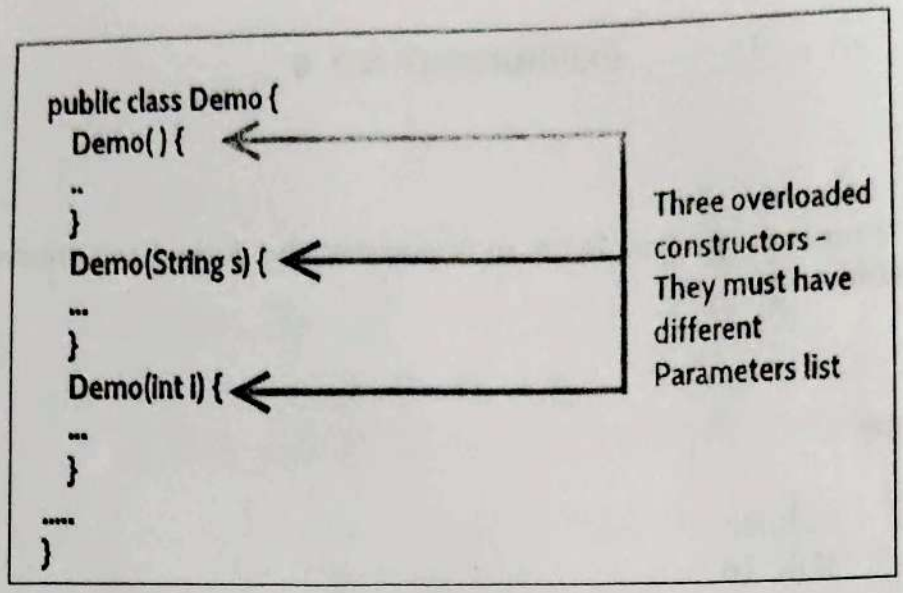
Methods, Constructors can also be overloaded. Overloaded constructor is called based upon the parameters specified when new is executed.

CONSTRUCTOR is a special method that is used to initialize a newly created object and is called just after the memory is allocated for the object. It can be used to initialize the objects to desired values or default values at the time of object creation. It is not mandatory for the coder to write a constructor for a class.
If no user-defined constructor is provided for a class, compiler initializes member variables to its default values.

- numeric data types are set to 0
- char data types are set to null character('\0')
- reference variables are set to null

Rules for creating a Java Constructor

1. It has the **same name** as the class
2. It should not return a value not even ***void***
3. Constructor calling must be the **first** statement of constructor in Java.
4. If we have defined any parameterized constructor, then compiler will not create default constructor. and vice versa if we don't define any constructor, the compiler creates the default constructor(also known as no-arg constructor) by default during compilation



When do we need Constructor Overloading?

Sometimes there is a need of initializing an object in different ways. This can be done using constructor overloading. For example, Thread class has 8 types of constructors. If we do not want to specify anything about a thread then we can simply use default constructor of Thread class, however if we need to specify thread name, then we may call the parameterized constructor of Thread class with a String args like this:

Thread t= new Thread (" MyThread ");

Let us take an example to understand need of constructor overloading. Consider the following implementation of a class Box with only one constructor taking three arguments.

```
// An example class to understand need of constructor overloading.
class Box
{
   double width, height,depth;

   // constructor used when all dimensions specified
   Box(double w, double h, double d)
   {
      width = w;
      height = h;
```

```
    depth = d;
  }

  // compute and return volume
  double volume()
  {
    return width * height * depth;
  }
}
```

As we can see that the Box() constructor requires three parameters. This means that all declarations of Box objects must pass three arguments to the Box() constructor. For example, the following statement is currently invalid:

```
Box ob = new Box();
```

Since Box() requires three arguments, it's an error to call it without them. Suppose we simply wanted a box object without initial dimension, or want to initialize a cube by specifying only one value that would be used for all three dimensions. From the above implementation of Box class these options are not available to us.

These types of problems of different ways of initializing an object can be solved by constructor overloading. Below is the improved version of class Box with constructor overloading.

```
// Java program to illustrate

// Constructor Overloading

class Box

{

  double width, height, depth;



  // constructor used when all dimensions specified
```

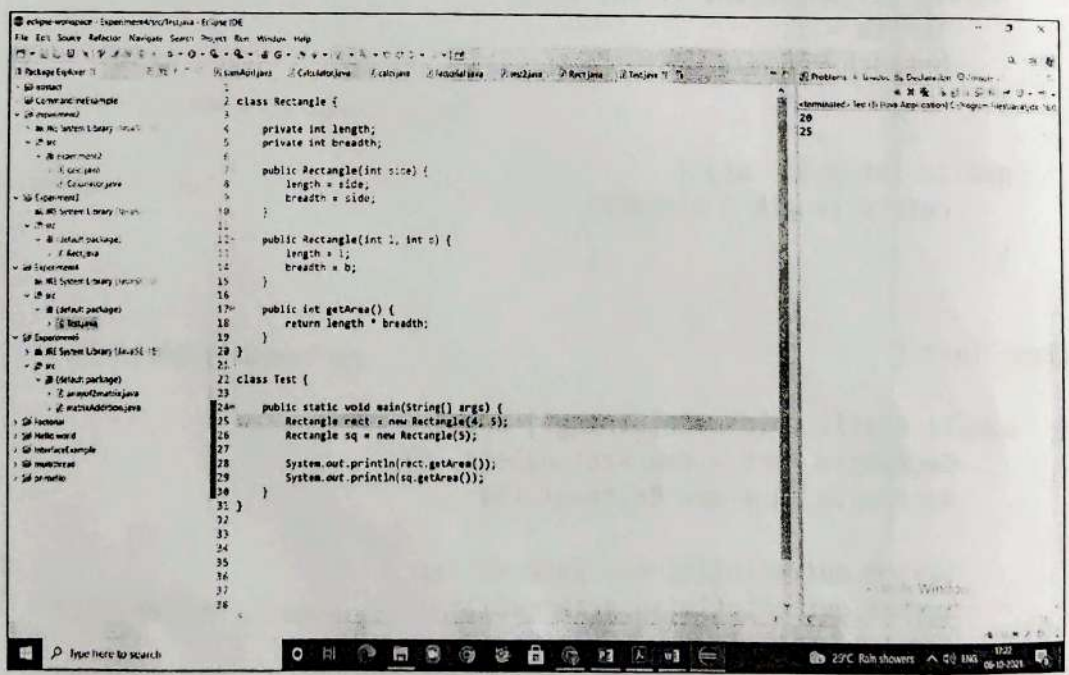
Output:

```
20
25
```

Conclusion:

~~Write~~ We have learnt in this experiment to write a program in JAVA to demonstrate the method and constructor overloading.

Screenshot's of Program and Result:



EXPERIMENT NO. 5

Aim: Write Programs in Java to sort i) List of integers ii) List of names.

Objective: To learn Arrays and Strings in Java

Software:

1. Eclipse
2. JDK 16

Theory:

i) Java Program to sort the names of an array in ascending order

For, sorting names in an Alphabetical order there are multiple ways to sort the array, like using inbuilt Arrays.sort() method or using normal sorting algorithms like the bubble sort, merge sort. Here let's use the bubble sort and inbuilt sort.

Example:
Input : Array[] = {"Sourabh", "Anoop, "Harsh", "Alok", "Tanuj"}
Output: Array[] = {"Alok", "Anoop", "Harsh", "Sourabh", "Tanuj"}

Input : Array[] = {"Bob", "Alice"}
Output: Array[] = {"Alice", "Bob"}

1. Brute-Force Approach

The idea is to compare the strings on the basis of thereunicode and swap them in accordance with the returned int value based on the comparison between the two strings using **compareTo()** method.
In the input, the user has to enter the number of names and the names and on the output, it will sort and display them in alphabetical order. For this, we are going to use the compareTo() method and compare one string with the rest of the strings.

CompareTo() is used to compare two strings lexicographically. Each character of both strings is converted into its **unicode** value. Lexicographical order is nothing but alphabetical order.
This method returns an int data-type which is based on the comparison between the two string. If it returns>0 then the parameter passed to compareTo() method is lexicographically first whereas if returns < 0 then string calling the method is lexicographically correct.

Steps:

- Using **CompareTo()** method compare one string with the rest of the strings
- To swap the elements based on the comparison between the two string.
- Print the Sorted Names in an Alphabetical Order.

Program:

```
// Java Program to Sort Names in an Alphabetical Order
class sorting {
      publicstaticvoid main(String[] args)
      {
            // storing input in variable
            intn = 4;

            // create string array called names
            String names[]
                  = { "Rahul", "Ajay", "Gourav", "Riya" };
            String temp;
            for (inti = 0; i<n; i++) {
                  for (intj = i + 1; j<n; j++) {

                        // to compare one string with other strings
                        if (names[i].compareTo(names[j]) > 0) {
                              // swapping
                              temp = names[i];
                              names[i] = names[j];
                              names[j] = temp;
                        }
                  }
            }

            // print output array
            System.out.println(
                  "The names in alphabetical order are: ");
            for (inti = 0; i<n; i++) {
                  System.out.println(names[i]);
            }
```

```
    }
}
```

Output:

```
The names in alphabetical order are:
Ajay
Gourav
Rahul
Riya
```

2. Inbuilt Sort function

- Using inbuilt Arrays.sort() method to sort the array.
- Print the Sorted Names in an Alphabetical Order.

Below is the implementation of the above approach:

Java

```
// Java Program to Sort Names in an Alphabetical Order

importjava.io.*;

importjava.util.*;

classGFG {

  publicstaticvoidmain(String[] args)

  {

    // storing input in variable

    intn = 4;

    // create string array called names

    String names[]

      = { "Rahul", "Ajay", "Gourav", "Riya"};
```

```
        // print output array

        System.out.println(

          "The names in alphabetical order are: ");

        for(inti = 0; i< n; i++) {

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

        }

    }

}
```

Output

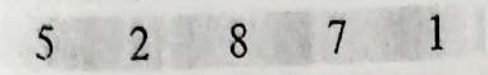
```
The names in alphabetical order are:
Ajay
Gourav
Rahul
Riya
```

ii) Java Program to sort the elements of an array in ascending order

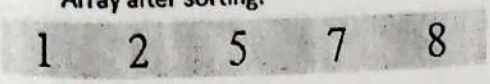
In this program, we need to sort the given array in ascending order such that elements will be arranged from smallest to largest. This can be achieved through two

loops. The outer loop will select an element, and inner loop allows us to compare selected element with rest of the elements.

Original array:



Array after sorting:



Elements will be sorted in such a way that the smallest element will appear on extreme left which in this case is 1. The largest element will appear on extreme right which in this case is 8.

Algorithm

- **STEP 1:** START
- **STEP 2:** INITIALIZE arr[] ={5, 2, 8, 7, 1 }.
- **STEP 3:** SET temp =0
- **STEP 4:** PRINT "Elements of Original Array"
- **STEP 5:** REPEAT STEP 6 UNTIL i<arr.length
  //for(i=0; i<arr.length; i++)
- **STEP 6:** PRINT arr[i]
- **STEP 7:** REPEAT STEP 8 to STEP 9 UNTIL i<arr.length
  //for(i=0; i<arr.length; i++)
- **STEP 8:** REPEAT STEP 9 UNTIL j<arr.length
  //for(j=i+1;j<arr.length;j++)
- **STEP 9:** if(arr[i]>arr[j]) then
  temp = arr[i]
  arr[i]=arr[j]
  arr[j]=temp
- **STEP 10:** PRINT new line
- **STEP 11:** PRINT "Elements of array sorted in ascending order"
- **STEP 12:** REPEAT STEP 13 UNTIL i<arr.length
  //for(i=0;i<arr.length;i++)
- **STEP 13:** PRINT arr[i]

- **STEP 14:** END

Program:

```
// Java Program to Sort integer in an Ascending Order

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

    //Initialize array
    int [] arr = new int [] {5, 2, 8, 7, 1};
    int temp = 0;

    //Displaying elements of original array
    System.out.println("Elements of original array: ");
    for (int i = 0; i < arr.length; i++) {
      System.out.print(arr[i] + " ");
    }

    //Sort the array in ascending order
    for (int i = 0; i < arr.length; i++) {
      for (int j = i+1; j < arr.length; j++) {
        if(arr[i] > arr[j]) {
          temp = arr[i];
          arr[i] = arr[j];
          arr[j] = temp;
        }
      }
    }

    System.out.println();

    //Displaying elements of array after sorting
    System.out.println("Elements of array sorted in ascending order: ");
    for (int i = 0; i < arr.length; i++) {
      System.out.print(arr[i] + " ");
```

EXPERIMENT NO. 6

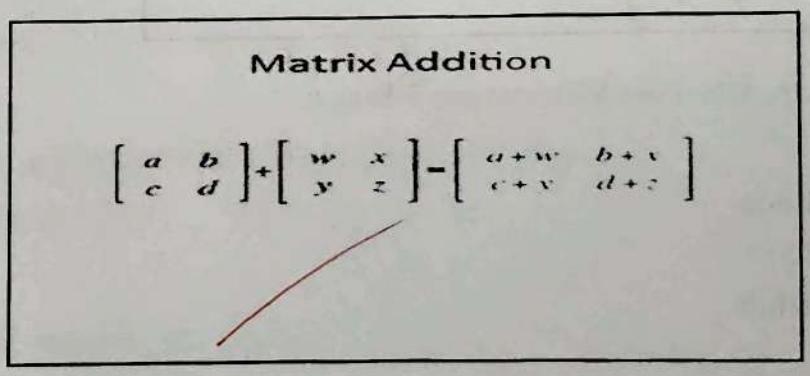
Aim: Write a Program in Java to add two matrices.

Software:

1. Eclipse
2. JDK 16

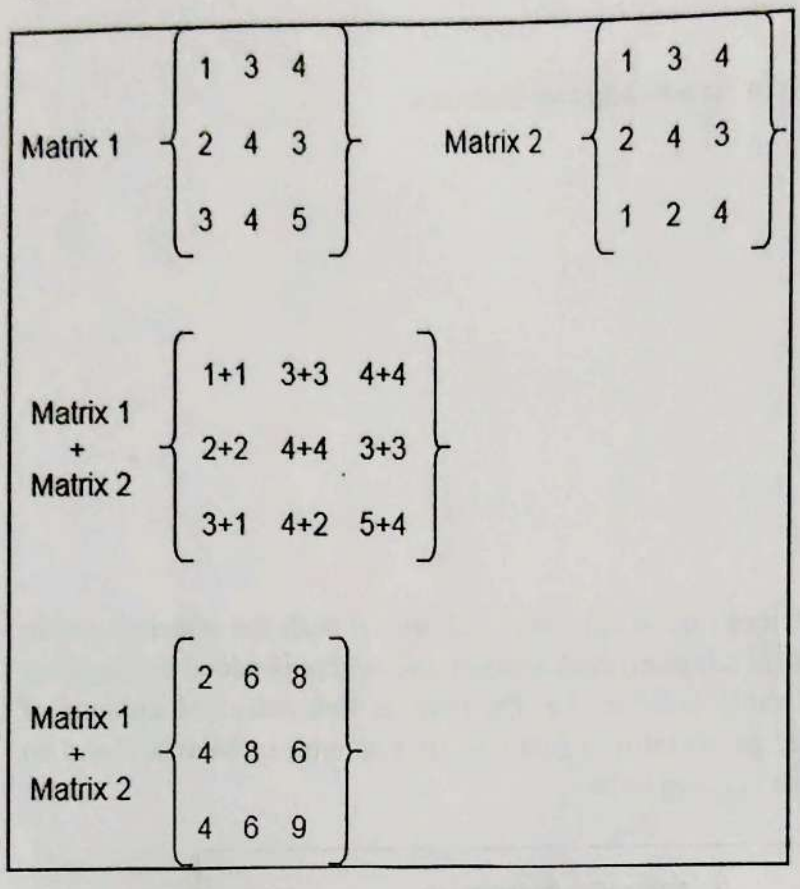
Theory:

Addition of two matrices can be carried if and only if both the matrices are in the same order. In matrix addition, each term of one matrix is added to the other matrix's term, at the same location, i.e. the term at first row first column of Matrix 1 will be added to the term at first row first column of Matrix 2 and so on. A pictorial example is given below.



We can add two matrices in java using binary + operator. A matrix is also known as array of arrays. We can add, subtract and multiply matrices.

Eg.



Java Program to Add Two Matrices are 3 Ways:

- Using For Loop
- Using While
- Using Do-While

Addition of Two Matrices – Using For Loop

1) If both matrices are of the same size then only we can add the matrices.
2) Use the double dimensional array to store the matrix elements.
3) Read row number,column number and initialize the double dimensional arrays mat1[][],mat2[][],res[][] with same row number,column number.

4) Store the first matrix elements into the two-dimensional array mat1[][] using two for loops. i indicates row number, j indicates column index.Similarly matrix 2 elements in to mat2[][].

5) Add the two matrices using for loop

- for i=0 to i<row
- for j=0 to j<col
- mat1[i][j] + mat2[i][j] and store it in to the matrix res at res[i][j] .

Program:

```
importjava.util.Scanner;
classAddMatrix
{
publicstaticvoidmain(Stringargs[])
{
introw,col,i,j;
Scanner in=newScanner(System.in);

System.out.println("Enter the number of rows");
row=in.nextInt();

System.out.println("Enter the number columns");
col=in.nextInt();

intmat1[][]=newint[row][col];
intmat2[][]=newint[row][col];
intres[][]=newint[row][col];

System.out.println("Enter the elements of matrix1");

for(i=0;i<row;i++)
{
```

```
for(j=0;j<col;j++)
mat1[i][j]=in.nextInt();

System.out.println();
}
System.out.println("Enter the elements of matrix2");

for(i=0;i<row;i++)
{

for(j=0;j<col;j++)
mat2[i][j]=in.nextInt();

System.out.println();
}

for(i=0;i<row;i++)
for(j=0;j<col;j++)
res[i][j]=mat1[i][j]+mat2[i][j];

System.out.println("Sum of matrices:-");

for(i=0;i<row;i++)
{
for(j=0;j<col;j++)
System.out.print(res[i][j]+"\t");

System.out.println();
```

EXPERIMENT NO. 7

Aim: Write a program in Java to create a player class. Inherit the classes Cricket_player, Football_player and Hockey_player from player class.

Objectives: To learn the use of inheritance and constructor in java.

Software:

1. Eclipse
2. JDK 16

Theory:

Theory:

Inheritance: Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications. Using inheritance, you can create a general class that defines traits common to a set of related items. This class can then be inherited by other, more specific classes, each adding those things that are unique to it. In the terminology of Java, a class that is inherited is called a superclass. The class that does the inheriting is called a subclass. Therefore, a subclass is a specialized version of a superclass. It inherits all of the instance variables and methods defined by the superclass and adds its own, unique elements.

Important Points

1. Code reuse is the most important benefit of inheritance because subclasses inherits the variables and methods of superclass.
2. Private members of superclass are not directly accessible to subclass. As in this example, Animal variable noOfLegs is not accessible to Cat class but it can be indirectly accessible via getter and setter methods.
3. Superclass members with default access is accessible to subclass ONLY if they are in same package.
4. Superclass constructors are not inherited by subclass.
5. If superclass doesn't have default constructor, then subclass also needs to have an explicit constructor defined. Else it will throw compile time exception. In the

subclass constructor, call to superclass constructor is mandatory in this case and it should be the first statement in the subclass constructor.

6. Java doesn't support multiple inheritance, a subclass can extends only one class. Animal class is implicitly extending Object class and Cat is extending Animal class but due to java inheritance transitive nature, Cat class also extends Object class.
7. We can create an instance of subclass and then assign it to superclass variable, this is called **upcasting**. Below is a simple example of upcasting:

```
Cat c = newCat(); //subclass instance
Animal a = c; //upcasting, it's fine since Cat is also an Animal
```

8. When an instance of Superclass is assigned to a Subclass variable, then it's called **downcasting**. We need to explicitly cast this to Subclass. For example;

```
Cat c = newCat();
Animal a = c;
Cat c1 = (Cat) a; //explicit casting, works fine because "c"is actually of type Cat
```

Note that Compiler won't complain even if we are doing it wrong, because of explicit casting. Below are some of the cases where it will throw ClassCastException at runtime.

```
Dog d = newDog();
Animal a = d;
Cat c1 = (Cat) a; //ClassCastException at runtime

Animal a1 = newAnimal();
Cat c2 = (Cat) a1; //ClassCastException because a1 is actually of type Animal at runtime
```

9. We can override the method of Superclass in the Subclass. However we should always annotate overridden method with @Override annotation. The compiler will know that we are overriding a method and if something changes in the superclass method, we will get a compile-time error rather than getting unwanted results at the runtime.
10. We can call the superclass methods and access superclass variables using **super** keyword. It comes handy when we have the same name variable/method in the subclass but we want to access the superclass variable/method. This is also used when Constructors are defined in the superclass and subclass and we have to explicitly call the superclass constructor.
11. We can't extend Final classes in java.
12. If you are not going to use Superclass in the code i.e your Superclass is just a base to keep reusable code then you can keep them as Abstract class to

avoidunnecessary instantiation by client classes. It will also restrict the instance creation of base class.

Program:

This program shows a class hierarchy of the various players in a team like Cricket player, Football player and Hockey player

```
class Player
{
  String name;
intage;
Player(String n,inta)
{ name=n; age=a; }
void show()
  {
System.out.println("Player name: "+name);
System.out.println("Age: "+age);
  }
}
class cricket_player extends Player
{
  String type;
cricket_player(String n, String t, int a)
  {
super(n,a);
type=t;
  }
public void show()
  {
super.show();
System.out.println("Player type : "+type);
  }
}
class football_player extends Player
  {
     String type;
football_player(String n, String t, int a)
  {
super(n,a);
type=t;
  }
public void show()
   {
super.show();
```

EXPERIMENT NO. 8

Aim: Write a program in Java to create a package in java and import the classes from that package in the main program.

Objectives: To learn the use of packages in java.

Software:

1. Eclipse
2. JDK 16

Theory:

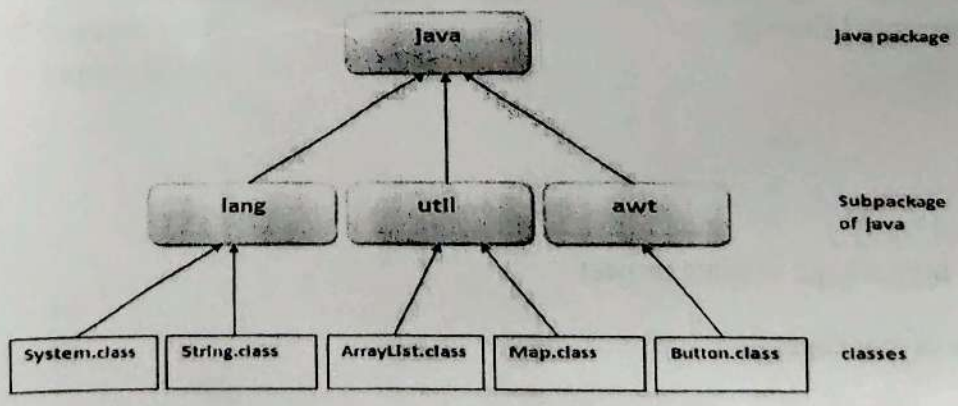
Package: A java package is a group of similar types of classes, interfaces and sub-packages. Package in java can be categorized in two form, built-in package and user-defined package. There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.



There are three ways to access the package from outside the package.

import package.*;

import package.classname;

fully qualified name.

Subpackage in java

Package inside the package is called the subpackage. It should be created to categorize the package further.

Program:

This program shows a main program named PackageDemo. The project creates two packages under current working project. These two packages are named as mypackage and fdp_fjp. The package mypackage contains a public class Balance. The Balance class has a constructor and a show() method. The package fdp_fjp contains a Saving class that extends from Balance class.

```
// Source Code for PackageDemo program

import fdp_fjp.Saving;

public class PackageDemo {
   public static void main(String args[]) {
        Saving current[]= new Saving[3];
        current[0] = new Saving("Mr.RAM",123.23,"Kolkata", 555);
        current[1] = new Saving("Mr.Raj", 157.02, "Chennai", 420);
        current[2] = new Saving("Mrs. Seeta", -12.33, "Mumbai", 999);
        for (int i = 0; i < 3; i++) {
        current[i].show();
        }
}
}


//Create mypackage in same project

package mypackage;

public class Balance {
```

EXPERIMENT NO. 9

Aim: Write a Java program which implements interface.

Objective: To learnimplements interface in Java

Software:

1. Eclipse
2. JDK 16

Theory:

Interface in Java

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

Declaration of interface:

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

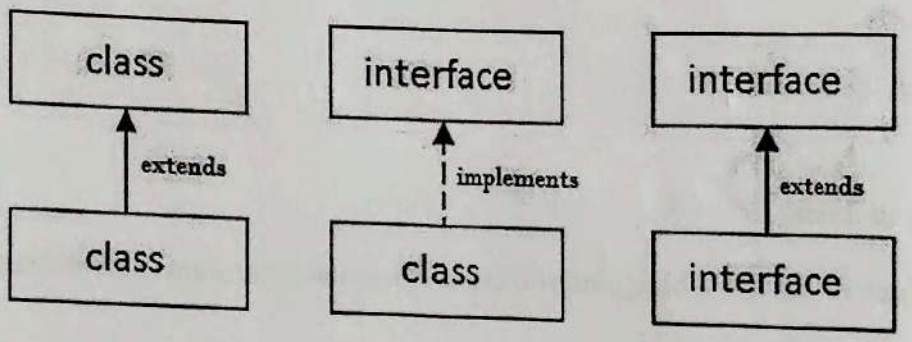
Syntax:

```
interface <interface_name>{

   // declare constant fields
   // declare methods that abstract
   // by default.
}
```

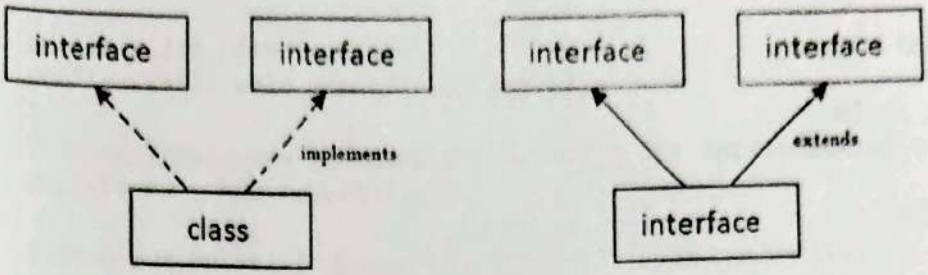
The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Multiple Inheritance in Java

Program:

1. Program for interface:

```
public interface sports {


int sportsWt = 5;
public void putWt();

}
```

2. Program for Class:

```
import java.util.Scanner;

/**Java Program calculating marks of a student using multiple
inheritance
 * implemented through interface
 * Create class result to calculate student result using its test
marks and sports marks
 * Declare class student and class test. Use student class in
test class.
 * Create interface of sports.
 * use sports and test in result class.
 *
 *
 */

/*interface sports
{
int sportsWt = 5;
public void putWt();
```

EXPERIMENT NO. 10

Aim: Write a program to create multiple threads and demonstrate how two threads communicate with each other.

Objective: To learnmultiple threads in Java

Software:

1. Eclipse
2. JDK 16

Theory:

Multi-threaded programs may often come to a situation where multiple threads try to access the same resources and finally produce erroneous and unforeseen results.

So it needs to be made sure by some synchronization method that only one thread can access the resource at a given point of time.

Java provides a way of creating threads and synchronizing their task by using synchronized blocks.

Java Synchronization

Synchronization is a process of handling resource accessibility by multiple thread requests. The main purpose of synchronization is to avoid thread interference. At times when more than one thread try to access a shared resource, we need to ensure that resource will be used by only one thread at a time. The process by which this is achieved is called synchronization. The synchronization keyword in java creates a block of code referred to as critical section.

General Syntax:

```
synchronized(object)

{
```

```
//statement to be synchronized

}
```

Every Java object with a critical section of code gets a lock associated with the object. To enter critical section a thread need to obtain the corresponding object's lock.

Why we need Syncronization?

If we do not use syncronization, and let two or more threads access a shared resource at the same time, it will lead to distorted results.

Consider an example, Suppose we have two different threads **T1** and **T2**, T1 starts execution and save certain values in a file *temporary.txt* which will be used to calculate some result when T1 returns. Meanwhile, T2 starts and before T1 returns, T2 change the values saved by T1 in the file temporary.txt (temporary.txt is the shared resource). Now obviously T1 will return wrong result.

To prevent such problems, synchronization was introduced. With synchronization in above case, once T1 starts using *temporary.txt* file, this file will be **locked**(LOCK mode), and no other thread will be able to access or modify it until T1 returns.

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Thread Synchronization

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

1. Mutual Exclusive
    1. Synchronized method.
    2. Synchronized block.
    3. Static synchronization.

2. Cooperation (Inter-thread communication in java)

Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. It can be achieved by using the following three ways:

1. By Using Synchronized Method
2. By Using Synchronized Block
3. By Using Static Synchronization

Concept of Lock in Java

Synchronization is built around an internal entity known as the lock or monitor. Every object has a lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

Using Synchronized Methods

Using Synchronized methods is a way to accomplish synchronization.

If no Synchronization

Suppose, we are not using synchronization and creating multiple threads that are accessing display method and produce the random output.

Synchronizes Keyword

To synchronize program, we must *synchronize* access to the shared method, making it available to only one thread at a time. This is done by using keyword **synchronized** with shared method.

```
synchronizedvoid display (Stringmsg)
```

1. Java Synchronized Method

If you declare any method as synchronized, it is known as synchronized method.

Synchronized method is used to lock an object for any shared resource.

When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

2. Synchronized Block in Java

Synchronized block can be used to perform synchronization on any specific resource of the method.

Suppose we have 50 lines of code in our method, but we want to synchronize only 5 lines, in such cases, we can use synchronized block.

If we put all the codes of the method in the synchronized block, it will work same as the synchronized method.

Points to Remember

- Synchronized block is used to lock an object for any shared resource.
- Scope of synchronized block is smaller than the method.
- A Java synchronized block doesn't allow more than one JVM, to provide access control to a shared resource.
- The system performance may degrade because of the slower working of synchronized keyword.
- Java synchronized block is more efficient than Java synchronized method.

Syntax

```
synchronized (object reference expression) {
  //code block
}
```

Difference between synchronized keyword and synchronized block

When we use synchronized keyword with a method, it acquires a lock in the object for the whole method. It means that no other thread can use any synchronized method until the current thread, which has invoked it's synchronized method, has finished its execution.

synchronized block acquires a lock in the object only between parentheses after the synchronized keyword. This means that no other thread can acquire a lock on the locked object until the synchronized block exits. But other threads can access the rest of the code of the method.

Which is more preferred - Synchronized method or Synchronized block?

In Java, synchronized keyword causes a performance cost. A synchronized method in Java is very slow and can degrade performance. So we must use synchronization keyword in java when it is necessary else, we should use Java synchronized block that is used for synchronizing critical section only.

Inter-thread Communication in Java

Inter-thread communication or **Co-operation** is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of **Object class**:

- wait()
- notify()
- notifyAll()

1) wait() method

The wait() method causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

Method	Description
public final void wait()throws InterruptedException	It waits until object is notified.
public final void wait(long timeout)throws InterruptedException	It waits for the specified amount of time.

2) notify() method

The notify() method wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation.

Syntax:

1. **public final void** notify()

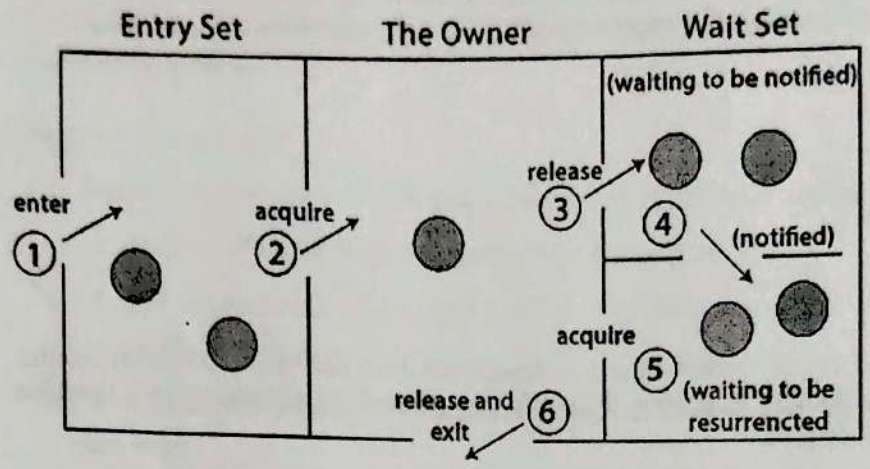
3) notifyAll() method

Wakes up all threads that are waiting on this object's monitor.

Syntax:

```
public final void notifyAll()
```

Understanding the process of inter-thread communication:



The point to point explanation of the above diagram is as follows:

1. Threads enter to acquire lock.
2. Lock is acquired by on thread.
3. Now thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits.
4. If you call notify() or notifyAll() method, thread moves to the notified state (runnable state).
5. Now thread is available to acquire lock.
6. After completion of the task, thread releases the lock and exits the monitor state of the object.

Difference between wait and sleep:

Let's see the important differences between wait and sleep methods.

wait()	sleep()

The wait() method releases the lock.	The sleep() method doesn't release the lock.
It is a method of Object class	It is a method of Thread class
It is the non-static method	It is the static method
It should be notified by notify() or notifyAll() methods	After the specified amount of time, sleep is completed.

Program:

a. Program for Main Thread

```
public class Lab10 {

public static void main(String[] args) {
      Thread obj = Thread.currentThread();
System.out.println("Current thread is " +obj);
System.out.println("Name of current thread is " +obj.getName());
obj.setName("Multi Thread"); // Changing name of main thread.
System.out.println("Name of current thread after changing name is "
+obj);
      Account accObj= new Account(100);
      Thread t1=  new Thread( new DepositeThread(accObj, 30));
t1.setName("Deposite Thread");
System.out.println("Name of current thread is " +t1.getName());
t1.start();
       Thread t2= new Thread(new DepositeThread(accObj, 10));
t2.start();
       Thread t3=new Thread(new WithdrawThread(accObj, 150));
t3.start();
        Thread t4=new Thread(new DepositeThread(accObj, 20));
t4.start();
new Thread(new DepositeThread(accObj, 30)).start();
new Thread( new WithdrawThread(accObj, 30)).start();
    }
}
```

EXPERIMENT NO. 11

Aim: Write a java program which use try and catch for exception handling.

Objective: To learn try and catch for exception handlingin Java

Software:

1. Eclipse
2. JDK 16

Theory:

Java Exceptions

When executing Java code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an **exception** (throw an error).

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Exception Handling in Java :

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained. Try-catch block which is used for exception handling.

Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions.

1. Try block

The try block contains set of statements where an exception can occur. A try block is always followed by a catch block, which handles the exception that occurs in associated try block. A try block must be followed by catch blocks or finally block or both.

Syntax of try block

```
try{
//statements that may cause an exception
}
```

While writing a program, if you think that certain statements in a program can throw a exception, enclosed them in try block and handle that exception

2. Catch block

A catch block is where you handle the exceptions, this block must follow the try block. A single try block can have several catch blocks associated with it. You can catch different exceptions in different catch blocks. When an exception occurs in try block, the corresponding catch block that handles that particular exception executes. For example if an arithmetic exception occurs in try block then the statements enclosed in catch block for arithmetic exception executes.

Syntax of try catch in java

```
try
{
//statements that may cause an exception
}
catch(exception(type) e(object))
{
//error handling code
}
```

3. Java try and catch

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The try and catch keywords come in pairs:

Syntax

```
try
// Block of code to try
}
catch(Exceptione){
// Block of code to handle errors
}
```

4. Multiple catch blocks in Java

The few rules about multiple catch blocks.

1. As I mentioned above, a single try block can have any number of catch blocks.
2. A generic catch block can handle all the exceptions. Whether it is ArrayIndexOutOfBoundsException or ArithmeticException or NullPointerException or any other type of exception, this handles all of them.

```
catch(Exception e){
  //This catch block catches all the exceptions
}
```

3. If no exception occurs in try block then the catch blocks are completely ignored.
4. Corresponding catch blocks execute for that specific type of exception:
catch(ArithmeticException e) is a catch block that can handle ArithmeticException
catch(NullPointerException e) is a catch block that can handle NullPointerException
5. We can also use throw exception.

5. Finally block

Finally block executes whether an exception occurs or not. You should place those statements in finally blocks, that must execute whether exception occurs or not.