18/11/19

Course Outcomes:-

CO_1:- Describe Java Virtual Machine and Type Casting (K_2)

CO_2:- Demonstrate Concepts like Constructors, Arrays, nested classes and Command line Arguments (K_3)

CO_3:- Implement Concepts of Inheritance and Exception Handling (K_3)

CO_4:- Develop programs on Multithreading and files (K_3)

CO_5:- Demonstrate applet programming and AWT Components (K_3)

CO_6:- Demonstrate Event Handling and Swings (K_3)

2/11/19

ILO:- Features of Java

① Simple:-

It Contains 4 Reasons to make the feature Simple

→ It Supports all the Syntaxes in c and c++
→ It doesn't Supports the pointers
→ It doesn't Supports the Operator Overloading
→ It is Confined to the Small Devices (Limited Resources) and designed for Embedded Devices.

② platform Independent (or) Architectural Neutral (or) System Independent:-

platform - programming Environment (or) Operating System(OS)

The same Java Program yields the same Results on Different platforms like windows, Linux, Mac

③ Robust (Strong):-

It is nothing but Strong. Unlike C programs, Java programs does not crash the system. because Java Supports Exception Handling Mechanism to handle Runtime Errors (Divide by zero)

④ Interpreted:-

Some Languages are Compile based (c and c++), Some Other Languages are Interpreter based (BASIC) but the Java program Should be Compiled as well as Interpreted to make the Java as platform Independent

⑤ portable:-

portable Independence of Java makes it as portable.

It means We can Execute the Same class file on Different platforms without any issue

⑥ Scalable:-

Java can be Implemented On a wide range of Devices that is from Embedded Systems (Small Hand Held devices) to main frame Computers

⑦ High performance:-

Due to Compilation and Interpretation the Execution of a Java program is slower when Compared to C

The High performance will be achieve with the help of JIT (Just in time Compiler) of JVM (Java Virtual Machine). The Combination of Interpreter and JIT Compiler is known as Adaptive Optimizer

⑧ Multi-threaded:-

Java can Execute more than One block at a time (or) parallely. In Multithreading, a block of Statements is treated as thread

* thread - Light weight Process
* process - Executional instance of a program

⑨ Distributed:-

Using TCP/IP & UDP protocols the Java Can perform Distributed Computing On a network of Computers.

TCP - Transmission Control protocol
IP - Internet protocol
UDP - User Datagram protocol

⑩ Dynamic:-

Initially, the content of web is static. The Applet Programming feature of Java makes the content of web as Dynamic.

→ Name of the main class should be same as file name

22/11/19

IIo: Java Program Structure

```
Package  Package name;
impolt statements
class  classname
{
    public  static  void  main(string  arg:[])
    {
        .....
    }
}
```

Comments:-

It provides Readability

// - Single line

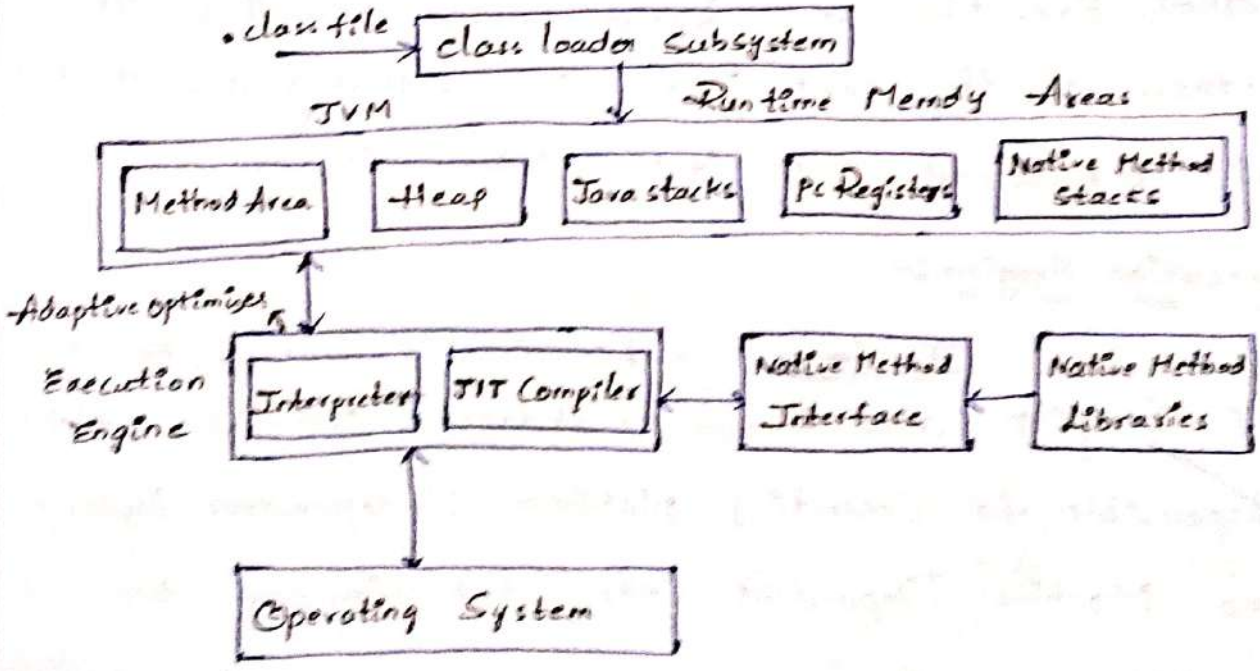
/* ... */ - Multi line

/** .... */ - Documentation (It creates API Document)
Application program Interface

Java Virtual Machine (JVM):-

JVM is the heart of Entire Java Program Execution process.

Components in Java Virtual Machine:-



Native Methods: (C and C++)

Class Loader Subsystem:-

It loads .class file into JVM. If there are any errors in the byte code then it doesn't load the file into JVM

There are 5 Runtime Memory Areas

① Method Area:- The class code, variables and the Method Code is Saved Under Method Area.

② Heap:- All the Objects are Created Under Heap

③ Java Stacks:- The tempolary Results during Method Execution and Return values as well as the address of next instruction / next Statement of after a method call are Saved Under Java Stacks.

④ PC (program Counter) Registers:-

Specifies the address of next instruction to be Executed.

⑤ Native Method Stacks:-

The tempolary Results during the native method Execution and Return values as well as the address of the next instruction after a native method call are saved Under Native Method Stacks.

Execution Engine:-

It is a Combination of Interpreter and JIT Compiler also Known as Adaptive Optimizer. Which is Responsible for Converting platform Independent Byte Code into Machine Dependent Code and fowards the Code to Microprocessor.

Native Method Interface:-

This is an Interface between Java program and Native Methods (e.g. C++ functions)

Native Method Libraries:-

These are C & C++ Header files such as stdio.h, and iostream.h

JIT Compiler:-

print a; - 4 nsec

print b; - 4 nsec

Repeat for 10 times - 4nsec

print c; - 4nsec

For the above pseudocode if we interpret the above code

Let us assume that it will take 4 nsec for each instruction. It takes 4 nsec time for every statement So, it allocates time for print c statement 10 times. The total time taken to execute the code is

$4+4+4+4*10 = 52$ nsec

In case of Repeated Execution of a Code. we Use JIT Compiler. It takes time for Only Once and save it in memory. It takes 4 nsec for every instruction and also 4 nsec to save the Repeated instruction. So the total time taken is

$4+4+4+8 = 20$ nsec

So, By using JIT Compiler we Can Reduce the time taken to Execute the code.

23/11/19

Compilation and Execution of a Java program:-

Sam.java

```
class Sam
{
    public static void main (String args[])
    {
        System.out.println ("Hello World!");
    }
}
```

Compilation:-

javac Sam.java

The above Command Generates .class file that is Sam.class. If there are no errors.

Execution of a Java program:-

java Sam

Hello World!

The Keyword public:-

The main method should be accessed Outside of a class that is by java runtime Environment. So, the access Specifier should be public

Static Keyword:-

We can access the methods of a class by using objects. A static method can be accessed by using classname

The Meaning of java. Sam is nothing but calling the static main method (java Sam. main())

System. Out. println("Hello World!"); :-

print() and psint(n() are the methods Used for Displaying the text / variables on the monitor

These methods belongs to a predefined class PrintStream. The Object for PrintStream class is

out

→ The Object out is declared in System class. So, that we can access print() method & println() methods can be accessed by using the Object out which belongs to System class.

→ The Main method Consists of Command line arguments i.e; String array. Using these Command line arguments, we can pass input to the main method.

Sam.java

```
class Sam
{
    public static void main(String args[])
    {
        int a=2, b=3, sum; → sum = a+b
        System.out.println("Sum is :" + sum);
        System.out.println("Sum is :" + a+b);
    }
}
```

Output:- Sum is : 5
Sum is : 23

```
class Sam
{
    public static void main (String args[])
    {
        System.out.println (" Hello world! ");
        int a=2, b=3, s;
        s=a+b;
        System.out.println (" Sum:"+s); // String and numeric Concatination
        System.out.println ("Sum :"+a+b); // string and numeric & again string and numeric Concatination.
        System.out.println (" Sum:"+(a+b)); // String and Expression. Concatination
        System.out.println (a+b+" Sum :"); // numeric & numeric & again numeric & String Concatination
    }
}
```

Output:-

Hello world!

Sum: 5

Sum : 23

Sum: 5

5 Sum:

→ * - astress (Means meant for all)

→ lang is the Default package

→ Root class of java is java

→ we coant all classes we declare it as import java.lang.*;

→ For Input purpose we declare it as import java.util.Scanner;

Displaying the Default values of primitive Datatypes (or) Basic Data types :-

```
class Sam
{
    Static int a;
    Static float f;
    Static char c;
    Static double d;
    Static long l;
    Static boolean b; → static String S;
    public static void main (String args[])
    {
        System.out.println (" Default value of int:" + a);
        System.out.println(" Default value of float:" + f);
        System.out.println(" Default value of char:" + c);
        System.out.println (" Default value of double:" + d);
        System.out.println (" Default value of long :" + l);
        System.out.println (" Default value of boolean:" + b);
        System.out.println (" Default value of string:" + s);
    }
}
```

Output:-

```
Default value of int: 0
Default value of float: 0.0
Default value of char: _ (Default value is single space)
Default value of double : 0.0
Default value of long: 0
Default value of boolean: false
Default value of String : null
```

Taking Input from the User Using Scanner class.

The Scanner class is available in package java.util

```
impolt java.util.Scanner;
class Sam
{
    public static void main(String args:[])
    {
        int a;
        float b;
        String s;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter integer:");
        a = sc.nextInt();
        System.Out.println("Entered integer:" + a);
        System.out.println("Enter float:");
        b = sc.nextFloat();
        System.out.println("Entered float:" + b);
        System.out.println("Enter string:");
        S = sc.next();
        System.out.println("Entered string:" +s);
    }
}
```

Outputer

```
Enter integer:
1
Entered integer:1
Enter float
2.0
Entered float: 2.0
Enter string:
svec
Entered string: svec
```

25/11/19

There are two default ~~packages~~ classes in Java. package

① java.lang.System

② java.lang.String

Naming Conventions:-

Java follows Strict naming Conventions for identifiers

→ An Identifier is the name of a class, an Interface, a package, a variable, Constant (or) literal and Methods.

→ An Identifier Should not have Spaces and Special characters except Underscore.

→ The Java class name (or) Interface name should start with a Capital letters. If the class name Contains more than one word then Every words first letter Should be a Capital.

→ The Java Variables / Method names Should start with a lower Case. If the variable name / method name Consists of more than one word then from Second word onwards Every word's first letter Should be Capital.

→ The package names always Starts with a lower Case

Ex: Class or Interface names

Ex 1:- Sample — Arithmetic — Addition } Methods
Division
SumOfIntegers

Ex: Sum
integer Number } variables

Ex: add()
addOfTwoInts() } Methods.

Examples of pre-defined classes:-

① System
② Scanner
③ String

System class:-

In Java, All the I/O operations are perfomed by using strings Input streams is connected with Input Device (keyboard). Output Stream is Connected with Output Devices (Monitor)

→ System.in - in is a parametre of System class Used to Represent Input stream i.e., keyboard

→ System.out - is a parametre of System class Used to Represent Output stream / print Stream to display output on the monitor

System.err - is the parameter of the System class Used to represent output stream/print Stream (flow of data) to display errors on the monitor

Scanner class:-

The Scanner class is used to take input

from the User by using Several Methods of it. The Scanner class is available in a package java.util

→ If we want to use a Scanner class then ~~first~~ time we need to write a import statement

```
import java.util.Scanner;
```

→ We need to create an object for Scanner class by passing System.in as a parameter to its Constructor

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

The Scanner class is having a parameterized Constructor which takes System.in as a parameter while creating an Object.

Methods of Scanner class:-

All the methods of a Scanner class are called by using the Object of Scanner class (sc)

① next() :-
Used to ~~Read~~ take String as input

② nextInt() :-
Used to take Int as input

③ nextFloat() :-
Used to take Float as input

④ nextLong() :-
Used to take long as input

⑤ nextDouble() :-
Used to take Double as input

⑥ ~~nextChar()~~ :-
~~Used to take Char as input~~

⑦ nextByte() :-
Used to take Byte as input

(8) nextBoolean():-

Used to take Boolean as input

(9) nextShort():-

Used to take Short as input

Integer Datatypes:- Byte, Short, int & long
(1) (2) (4) (8) bytes

Float " " :- Float, Double
(4) (8) bytes

Character takes 2 Bytes. (In Java)

→ Unique code is 2 Bytes i.e., it is used by char
(0 - ~~65536~~ $2^{16} - 1$)

→ In Ascii it Represents (0 - 255) 256 characters

Boolean - 1 byte

Basic Datatypes in Java:-

Integer types:- byte (1 byte)
short (2 bytes)
int (4 bytes)
long (8 bytes)

Floating point types:- float - 4 bytes
double - 8 bytes

Character Data type:-

char - 2 bytes

Java Uses Unique Code (UNI) but not Ascii. The Range of UNI Code 0 to $2^{16}-1$

Boolean Data type:

boolean - 1 byte

String Data type:

String - The size depends upon no. of characters in a string.

Input Using Command line Arguments:-

If we give input using Command line Arguments that input is only given to main method

Program to print length & Data (Given)

Displaying the Command line Arguments using the length property of array.

```
class Sam/DemoOfCmdLnArgs
{
    public static void main(String args[])
    {
        System.out.println("Count of Command line args:" +
                                                args.length);

        for (int i=0; i<args.length; i++)
            System.out.println("Argument " +(i+1)": "+ args[i]);
    }
}
```

Output:-

```
Count of Command line args: 2
Argument 1 : 10
Argument 2 : 12.3
```

Write a java program to find the factorial of the Given number (Take the input as scanner class)

```
import java.util.Scanner
class Sam
{
    public static void main(String args[])
    {   int i,n,f=1;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the Number:");
        n = sc.nextInt();
```

```
        for (i=1; i<=n; i++)
            f = f*i;
        System.out.println("factorial is:" +f);
    }
}
```

write a program to find prime or not

```
impdt java.util.Scanner
clas Sam
{
    public static void main(String args[])
    {
        int n,i,c=0;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the Number:");
        n = sc.nextInt();
        for(i=0;i<n;i++)
            if(n%i==0)
                c++;
        if(Count==2)
            System.out.println("prime");
        else
            System.out.println("Not a prime");
    }
}
```

write a program to find Amstrong number.

```
impdt java.util.Scanner
clas Sam
{
    public static void main(String args[])
    {
        int i,n,r,x,Sum=0;
        Scanner sc = new Scanner(System.in);
```

```
System.out.println ("Enter the number");
n = sc.nextInt();
x = n;
while (n != 0)
{
    r = n % 10;
    sum = sum + (r*r*r);
    n = n/10;
}
if (x == sum)
    System.out.println ("it is armstrong");
else
    System.out.println ("Not an armstrong");
}
}
```

write a program to find palindrome or not

```
impolt java.util.Scanner
clas Sam
{
    public static void main (String args[])
    {
        int a, n, r, s = 0;
        Scanner sc = new Scanner (System.in);
        System.out.println ("Enter the number");
        n = sc.nextInt();
        a = n;
        while (n != 0)
        {
            r = n % 10;
            s = (s * 10) + r;
            n = n/10;
        }
        if (a == r)
            System.out.println ("It is palindrome");
        else
            System.out.println ("Not a palindrome");
    }
}
```

27/11/19

Wrapper classes:-

For Every Data type in Java, there is a Coresponding pre-defined class which is known as a Wrapper class.

Datatype	Wrapper class	Conversion Method
byte	Byte	parseByte ( )
Short	Short	parseShort ( )
int	Integer	parseInt ( )
long	Long	parseLong ( )
float	Float	parse Float ( )
double	Double	parseDouble ( )
boolean	Boolean	parseBoolean ( )

The Conversion Methods in Wrapper classes are of type static

Java Program to perform Arithmetic addition using Command line Arguments (Input the two integers via Command line)

```
class Sam
{
    public static void main(String args[])
    {
        int a,b;
        a = Integer.parseInt(args[0]);
        b = Integer.parseInt(args[1]);
    } System.out.println("Sum:" + (a+b));
```

In one line:- (without any variables)

```
class Sam
{
    public static void main(String args[])
    {
        System.out.println("sum:"+(Integer.parseInt(args
                [0])+(Integer.parseInt(args[1]));
    }
}
```

Reading a Single character using next() method of Scanner class.

```
import java.util.Scanner;
class Sam
{
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        char c;
        System.out.println("Enter char:");
        c=sc.next().charAt(0);
        System.out.println("character is:"+c);
    }
}
```

creating reference — creating object for reference (Instantiation)

↳ only Rightside can be written without creating an object known as nameless / Anonymous objects

Reading a Single character from Command line

```
class Sam
{
    public static void main(String args[])
    {
        char c;
        c=args[0].charAt(0);
        System.out.println("character is:"+c);
    }
}
```

29/11/19

110:- Arrays in Java

→ Array is an Object which Consists of similar Elements.

→ There are 3 types of Array.

1. Single Dimensional Array
2. Multi Dimensional Array
3. Jagged Arrays

Declaring the Single Dimensional Array:-

datatype arrayname []; Ex:- int a[]

(or)

datatype [] arrayname; Ex:- int [] a;

(or)

datatype []arrayname; Ex:- int []a;

Initializing 1-D Array:-

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

Char s[] = {'A', 'B', 'C', 'D'};

String str[] = {"abc", "pqr", "mno"};

Instantiating an Array:-

→ The process of creating Instances is Called Instatiation

→ Allocating memory for an Object is called Instantiation

→ There are two ways of Instantiate an Array

1. int a[]; // It's a Declaration
   a = new int [5]; // It's a Instantiation

2. int a[] = new int [5]; // declaration, Instantiation

Declaring, Instantiating & Initializing 1D Array :-

```
int a[];          - Declaration
a = new int[4];   - Instantiation

a[0] = 10;
a[1] = 20;
a[2] = 30;        } Initialization
a[3] = 40;
```

A Java program to illustrate One Dimensional Arrays.

```
import java.io.*;
class Sam
{
    public static void main(String args[])throws Exception
    {
        int a[] = new int [4];
        int i;
        InputStreamReader is = new InputStreamReader (System.in);
        BufferedReader br = new BufferedReader (is);
        System.out.println("Enter array elements:");
        for(i=0; i<a.length; i++)
            a[i] = Integer.parseInt (br.readLine());
        System.out.println("Array Elements printing:");
        for(i=0; i<a.length; i++)
            System.out.println (a[i]);
    }
}
```

Accepting Input Using BufferedReader class:-

The BufferedReader has two methods read() and readline() method. The read() method reads a single character and returns a ASCII value The ReadLine() method returns a String.

→ The BufferRd Reader Object is Connected with InputStream Reader Object, inturn the InputStream Reader Object is Connected with System.in

→ First we need to create an Object for InputStreamReader and pass that object to Buffered Reader class Constructor

```
InputStreamReader is = new InputStreamReader (System.in);
BufferedReader br = new BufferedReader (is);
```

→ We can Combine the above two statements in a single line by passing Anonymous (nameless objects) InputStreamReader Object

```
BufferedReader br = new BufferedReader (new InputStreamReader
(System.in));
```

Program to illustrate BufferedReader class:-

```
import java.io.*;
class Sam
{
    public static void main (String args[]) throws Exception
    {
        int b;
```

```
InputStreamReader is = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(is);
System.out.println("Enter b:");
String s = br.readLine();
b = Integer.parseInt(s);
System.out.println("value of b:" + b);
}
}
```

30/11/19

2D Array:-

```
int a[][] = new int [2][3];
```

Declaring 2D array:-

Syntax: datatype arrayname [][];

(or)

datatype [][] arrayname;

(or)

datatype [][] arrayname;

Instantiating an array:-

```
int a[][] = new int [2][2];
int a[2][3];
a = new int [2][3];
```

Initialising a 2-D array:-

```
int a[][] = {{1,2,3},{4,5,6}};
int a[][] = new int [3][2];
a[0][0] = 1;
a[0][1] = 2;
a[1][0] = 3;
a[1][1] = 4;
a[2][0] = 5;
a[2][1] = 6;
```

Program to Search an Element in an array:-

```
import java.util.Scanner;
class Search
{
    public static void main (String args[])
    {
        int n, x, i;
        Scanner sc = new Scanner (System.in);
        System.out.println (" Enter no of elements ");
        n = sc.nextInt();
        System.out.println (" Enter elements into array");
        int a[] = new int [n];
        for(i=0; i<n; i++)
        {
            a[i]= sc.nextInt();
        }
        System.out.println (" Enter Key");
        x = sc.nextInt();
        for (i=0; i<n; i++)
        {
            if (a[i] == x)
            {
                System.out.println (" Element found");
                break;
            }
        }
        if (i>n)
        {
            System.out.println (" Element not found");
        }
    }
}
```

Program to implement Sorting:-

```
import java.util.Scanner;
class Sorting
{
    public static void main(String args[])
    {
        int n,i,j,t;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter no of elements");
        n = sc.nextInt();
        int a[] = new int[n+1];
        System.out.println("Enter elements into array");
        for(i=0;i<n;i++)
        {
            a[i] = sc.nextInt();
        }
        for(i=0;i<n;i++)
        {
            for(j=0;j<=(n-i-1);j++)
            {
                if(a[j]>a[j+1])
                {
                    t=a[j];
                    a[j]=a[j+1];
                    a[j+1]=t;
                }
            }
        }
        System.out.println("The Sorted array is");
        for(i=0;i<n;i++)
        {
            System.out.println(a[i]);
        }
    }
}
```

Program to add two arrays:-

```
import java.util.Scanner;
class Add
{
    public static void main(String args[]);
    {
        int n,i;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter no.of elements");
        n = sc.nextInt();
        int a[] = new int[n];
        int b[] = new int[n];
        System.out.println("Enta elements into array ");
        for(i=0; i<n; i++)
        {
            a[i] = sc.nextInt();
        }
        for(i=0; i<n; i++)
        {
            b[i] = sc.nextInt();
        }
        int c[] = new int[n];
        System.out.println("The addition is");
        for(i=0; i<n; i++)
        {
            c[i] = a[i]+b[i];
        }
        for(i=0; i<n; i++)
        {
            System.out-println(c[i]);
        }
    }
}
```

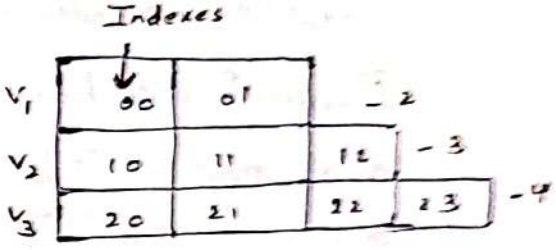
30/11/19

Jagged Arrays:-

Jagged Array is a two-Dimensional Array Each and Every Row may Consists of odd no. of Columns (or) Different no of Columns.

Jagged Array Declaration:-

```
int a[][] = new int [3][];
a[0] = new int [2];
a[1] = new int [3];
a[2] = new int [4];
```



Java program illustrating a 1D Array ~~and its exact~~

```
impolt java.io.*;
class Sam
{
   public static void main (String args[]) throws Exception
   {
      int a[]= new int [4];
      int i;
      InputStreamReader is = new InputStreamReader (System.in);
      BafferedReader br = new BufferedReader (is);
      System.out.println (" Enter array Elements:");
      for (i=0; i<a.length; i++)
         a[i] = Integer.parseInt (br.readLine());
      System.out.println (" Array Elements printing Using for:");
      for (i=0; i<a.length; i++)
         System.out.println (a[i]);
   }
}
```

Illustration of Jagged Arrays:

```
import java.io.*;
class Sam
{
    public static void main (String args[])throws Exception
    {
        int a[][] = new int [2][];
        int i,j;
        InputStreamReader is = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(is);
        a[0] = new int [2];
        a[1] = new int [3];
        System.out.println("Enter array elements:");
        for (i=0; i<2;i++)
            for (j=0; j<a[i].length; j++)
                a[i][j] = Integer.parseInt (br.readLine());
        System.out.println("Array Elements:");
        for (i=0; i<2; i++)
            for (j=0; j<a[i].length; j++)
                System.out.print (a[i][j]+" ");
            System.out.println();
    }
}
```

→ The length of the 2D Array is

```
int a[][] = new int [2][];
```

a.length is 2

→ The length of the 3D Array is

```
int b[][][] = new int [3][2][];
```

b.length is 3

→ In 2D Arrays, we must Specify the first Dimension

```
int b[][] = new int [2][]; // valid Statement
int b[][] = new int [][2]; // Invalid Statement
int b[][] = new int [][]; // Invalid because
                          Error: Array dimension Missing
```

Write a Java program to find the Sum of each and Every Row (vector) of a Jagged Array.

```
impalt java.io.*;
class Sam
{
    public static void main (String args[]) thaows Exception.
    {
        int a[][]= new int [2][];
        int i,j;
        InputStreamReader is = new InputStreamReader (System.in);
        BufferedReader br = new BufferedReader (is);
        a[0]= new int [2];
        a[1] = new int [3];
        System.out.println(" Enter array Elements:");
        for (i=0; i<a.length; i++)
            for (j=0; j< a[i].length; j++)
                a[i][j]= Integer.parseInt (br.readLine());
        int n;
        System.out.println ("Array Elements:");
        for (i=0; i<2; i++)
            n=0;
            for ( j=0; j< a[i].length;j++)
                n=n+a[i][j];
                System.out.print((i+1)+" vector Sum:"+ n);
    }
}
```

Enter array Elements
:1
2
3
4
array Elements:
3 vector Sum:10

Type Conversions and Type Casting:-

Type Conversion (or) Type Casting is nothing but Converting One datatype to another datatype

→ Type Conversion is automatic / Implicit
→ Type Casting is Explicit
→ Converting from lesser sized datatype to larger sized datatype is known as widening.
→ widening is automatic
→ Converting from larger sized datatype to lesser sized datatype is known as Narrowing
→ we have to do this Conversion explicitly by using Type cast.
→ Narrowing doesnot preserve the size whereas widening enhances the size

Conversion from int to long:- (widening)

```
int i = 10;
long l = i;
```

Conversion from long to int:- (Narrowing)

```
long l;
int i = (int) l; // Type casting.
```

Conversion from int to char:- (using br.read() method)

```
char c;
c = br.read();
```

The above statement is invalid because the read() method reads a character and returns Ascii Equivalent which is an integer. So, we need to Typecast it

```
c = (char) br.read();
```

Conversion from String to int :-

```
String s = "123";
int i = Integer.parseInt(s);
```

Converting from String to Boolean :-

① `String s = "true";`
`boolean flag = Boolean.parseBoolean(s);`

② `String s = "ok";`
`boolean flag = Boolean.parseBoolean(s);`
`System.out.println(flag)`

→ It prints false

Converting from ~~Decimal~~ int to ~~Binary~~ string :-

① `int i = 123`
`String s = String.valueOf(i);`

The String class is having a static method valueOf() that converts any datatype to a string

Conversion from float to String :-

② `float f = 123.5f`
`String s = String.valueOf(f);`

→ `float f = 123.4`, which is a invalid statement because defaulty java treats Every floating point value as double. The above statement is trying to assign double value to a floating point variable. To make it as valid `float f = 123.4f;` (or)

`float f = (float) 123.4;`

Converting from Decimal to Binary :-

The Integer wrapper class has another static method that is toBinaryString

```
int i=10;
System.out.println(Integer.toBinaryString(i));
```

o/p- 1010

Conversion from Decimal to octal:-

```
int i=10;
SO.P (Integer.toOctalString(i));
```

o/p- 12

Conversion from Decimal to Hexa

```
int i=10;
System.out.println(Integer.toHexString(i));
```

o/p- a

Conversion from string to Binary:-

Illustration of foreach:-

```
class Sam
{
  public static void main (String args[])
  {
     int a[] = {1,2,3,4,5};
     for(int i : a)
        System.out.println(i);
  }
}
```

Documentation Comments:-

The Documentation Comment Represents as

/** - - - */

The Documentation Comments generates an API Document nothing but we will get some HTML files In the Same folder Sam.html file. In order to get that html file we need to type the Command

javadoc Sam.java

2/12/19

Strings

→ String is an Object of set of characters

→ String is a Collection of Character Sequence

→ We can declare it in 2 ways.

① String literal

② Using new Operator (String Object)

① → String s_1 = "SVEC";

② → String s_2 = new String ("SVEC");

→ Another way Representing String

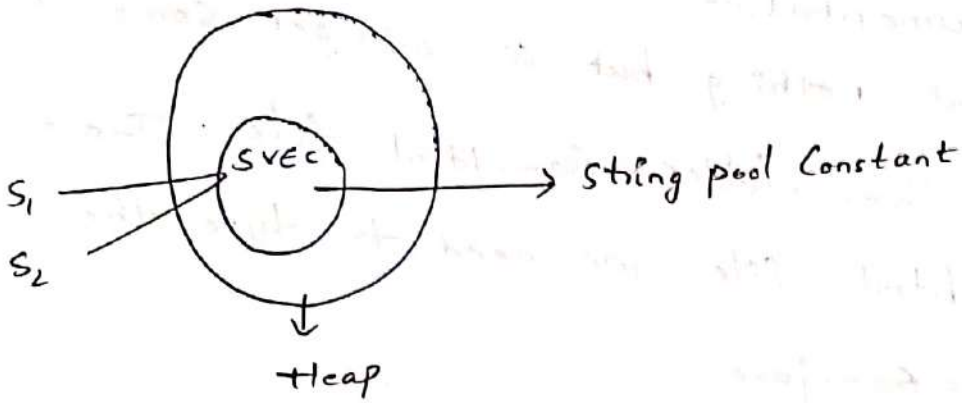
```
char ch[] = {'s', 'v', 'e', 'c'};
String s3 = new String(ch);
```

→ Whenever we Create Strings, the strings are memory for allocated in String pool Constant of Heap by JVM (Java Virtual Machine)

→ If we Create two string literals with the same value

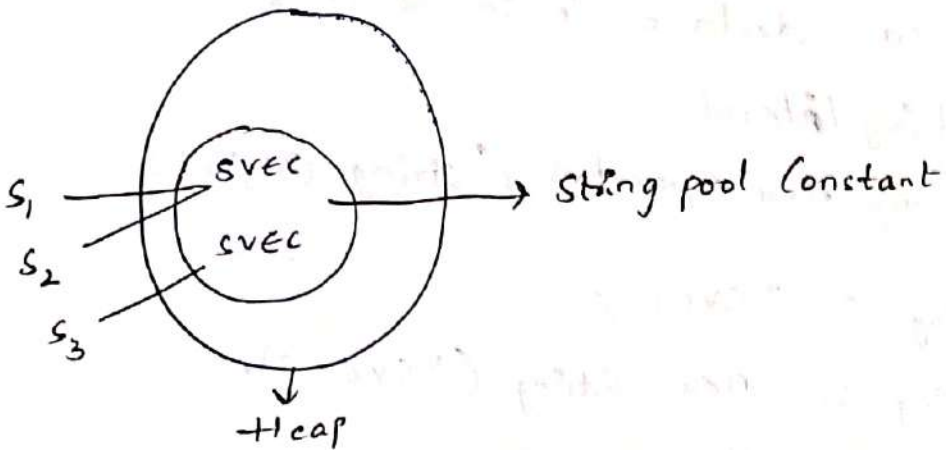
```
String s1 = "svec";
String s2 = "svec";
```

then s1 & s2 Refers the same memory location in String pool Constant



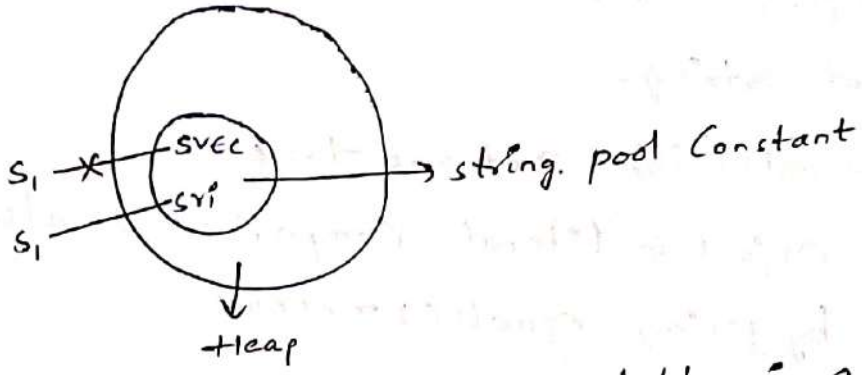
→ If we Create a String using a new Operator with the Same value as that of s1 & s2

```
String s3 = new String("svec");
```



→ Strings are immutable (special feature) means we cannot change / edit the value of a string

```
String s1 = "svec";
s1 = "sri";
```



→ The String class is available in a package java.lang which is a default package

String Comparison:-

```
class Sam
{
    public static void main(String args[]);
    {
        String s0 = "svec";
        String s1 = "svec";
        String s2 = new String("svec");
        String s3 = new String("svec");
        System.out.println("String literals Comparison:");
        System.out.println(s0 == s1);
        System.out.println("String object Comparison:");
        System.out.println(s2 == s3);
        System.out.println("String literal & Object Comparison:");
        System.out.println(s1 == s2);
    }
}
```

O/P:
String literals Comparison
true
String object Comparison
false
String literal & object Comparison
false

→ If we compare two String Objects using a double equal to (==) then it compares the Object id (Hash code) instead of the actual Strings.

→ The equals() method of a String class compares Actual Strings.

→ S_3.equals(S_3) returns true

→ String Object & literal Comparison is also returns true by using equals() method

→ S_2.equals(S_1) returns true

→ The length() method of String returns the no. of characters in a given String.

→ Ex: S_2.length()

String Size:-

→ S_2.length() * 2 gives the size of the string

String Methods:-

S.NO	Method ~~Name~~	Description
1.	int length()	Returns length of a string.
2.	boolean equals(String s)	Compare two strings and returns true if they are equal otherwise returns false
3.	String toUpperCase()	Converts the given string into the UpperCase Ex: S_2.toUpperCase = SVEC
4.	String toLowerCase()	Converts the given String into the lowercase Ex: S_2.toLowerCase = svec

4/12/19

	Method	Description
5.	String Concat(String s)	Concadinate two Strings and returns the Concadinated String.

Ex: String s_1 = "Sri";
String s_2 = "vasavi";
String s_3 = s_1.Concat(s_2)

6.	boolean isEmpty( )	Returns True if the String is Empty Otherwise, false

Ex: String s_1;
boolean b = s_1.isEmpty( )
it returns true

7.	boolean equalsIgnoreCase(String s)	Compares two strings by ignoring the case if they are equal returns true Otherwise, Returns false

Ex: String s_1 = "svec";
String s_2 = "SVEC";
boolean b = s_1.equalsIgnoreCase(s_2)
→ it returns true

8.	Static String valueOf(int i)	Convats a numeral to a string

Ex: int i = 123;
String s = String.valueOf(i);

9.	String trim( )	Removes the Spaces at the beginning and end of a String.

Ex: String s_1 = " svec ";
String s_2 = s_1.trim( )

10. String[] split(String seperators) Splits the given string based on Seperator, Regular exp, delimiter

Ex: String s1 = "Sri vasavi Eng college"

String s2[] = s1.split(" ");

In foreach

for (String i : s2)
System.out.println(i);

11. String Substring(int startindex) Returns a Substring from the specified position to the end

12. String Substring(int startindex, int endindex) Returns a Substring from the specified positions but end index is not inclusive

(11) Ex) String s1 = "Sri vasavi Engineering College";
op (12) Get the Substring from vasavi onwards.

String s2 = s1.Substring(4) = vasavi

String s3 = s1.Substring(4, 10) = vasavi Engineering College.

13. String replace(char old, char new) Replaces old character with a new character (all the occurrences)

Ex: String s1 = "sri vasavi engg. college";

String s2 = s1.replace(' ', '@');

o/p: Sri@vasavi@engg@college

14. String replace(string old, string new) — Replaces all the occurrences of old character sequence with a new character sequence

Ex: String s_1 = "abc and xyz and mno";
String s_2 = s_1.replace("and","&");
o/p abc & xyz & mno

15. boolean contains(String s) — Returns True if the character sequence is in the given string otherwise Returns false.

Ex: String s_1 = "Sri vasavi Engg. College";
String s_2 = s_1.contains(vasavi);

6/12/19

16. int indexOf(char ch) — Returns the position of a particular character in a given string.

Ex: String s_1 = "Sri vasavi";
int p = s_1.indexOf('i');
p = 2

17. int indexOf(char ch, int from) — Returns a position of a character from a given string and search start from Specified position instead of zero.

Ex: ~~int~~ ~~index()~~
String s_1 = "Sri vasavi";
int p = s_1.indexOf('i', 4);

18. Static String join(String delimito, string s_1, string s_2, - - -)

Joins the given set of Strings with a delimited and returns new string.

Ex: String S_1 = "06";
String S_2 = "12";
String S_3 = "2019";
String S_4 = String.join("-", S_1, S_2, S_3);
o/p: "06-12-2019"

Classes and Objects

→ Object is a realtime Entity in general
→ It has runtime Entity in Technical
→ It is instance of class
→ It has State & behaviour
→ It is variable of a class
→ Every Object has 3 characteristics.

① Id
② State &
③ Behaviour

Id: Reference of an Object
State: The value of an Object (value means the value of instance variables in an Object)
Behaviour: The functionality of an Object (Functionality means it uses methods)

→ Object is a physical Entity.
→ Object Behaviour changes the State

Ex: pen, Apple, Rectangle ...

Class: class is a model of a blueprint
→ class is a Collection of Similar Objects
→ class is a logical Entity

→ Every class may consists of

① Fields (property / feature) (or) variables

② Methods

③ Static blocks

④ Constructors

⑤ Nested classes / Interfaces

→ Creating Objects :- There are several ways to create an object out of which

① By using new Operator

② By using newInstance() method

③ By using clone() method

④ By using factory() method

⑤ By using de-Serialisation

→ Initialising the Object State

It is done in 3 ways

① Using reference

② Using Constructor

③ Using method

~~Creating Objects~~ Main inside a class initialising an object using Reference

```
public class Sam
{
    int a;
    int b;
    public static void main(String args[])
    {
        Sam s1 = new Sam();
        System.out.println(s1.a);
        System.out.println(s1.b);
    }
}
```

o/p 0
0

Creating a main in Seperate class and initialising the object using Reference

```
class A
{
    int a;
    int b;
}
class Sam
{
    public static void main(String args[])
    {
        A a1 = new A();
        System.out.println(a1.a);
        System.out.println(a1.b);
    }
}
```

o/p 0
0

7/12/19

→ Only Instance variables having default values.

Creating Multiple Objects in one line and initialising using the Reference

```
class A
{
    int a;
    int b;
}
class Sam
{
    public static void main(String args[])
    {
        A a1 = new A(), a2 = new A();
        a1.a = 10;
        a1.b = 20;
        a2.a = 30;
        a2.b = 40;
        System.out.println(a1.a);
        System.out.println(a1.b);
```

```
        System.out.println(a2.a);
        System.out.println(a2.b);
    }
}
```

o/p: 10, 20, 30, 40

→ Whenever we compile the above program, we can get two dot class files A.class, Sam.class

→ Initialising an Object using method (all the Objects with same values)

```
class A
{
    int a;
    int b;
    void getC()
    {
        a = 10;
        b = 20;
    }
}
class Sam
{
    public static void main (String args[])
    {
        A a1 = new A(), a2 = new A();
        a1.getC();
        a2.getC();
        System.out.println(a1.a);
        System.out.println(a1.b);
        System.out.println(a2.a);
        System.out.println(a2.b);
    }
}
```

o/p: 10, 20, 10, 20

Initialising an Object using method (objects with different values)

```
class A
{
    int a;
    int b;
    void get(int x, int y)
    {
        a=x;
        b=y;
    }
}
class Sam
{
    public static void main(String args[])
    {
        A a1=new A(), a2 = new A();
        a1.get(10,20);
        a2.get(30,40);
        System.out.println(a1.a);
        System.out.println(a1.b);
        System.out.println(a2.a);
        System.out.println(a2.b);
    }
}
```

o/p:
10
20
30
40

Displaying the instance variables without a Reference and using a method

```
class A
{
    int a;
    int b;
    void get(int x, int y)
    {
        a = x;
        b = y;
    }
    void disp()
    {
        S.o.p(a);
        S.o.p(b);
    }
}
class Sam
{
    public static void main(String args[])
    {
        A a1 = new A(), a2 = new A();
        a1.get(10, 20);
        a2.get(30, 40);
        a1.disp();
        a2.disp();
    }
}
```

Initialising the Object Using Constructor

→ Initialising the Object with Constructor

→ A Default Constructor will be created automatically whenever we create an Object

→ There are two types of Constructor in java

① Default Constructor &

② parameterized Constructor.

Default Constructor:- (To initialize all the Objects with the same value)

```
class A
{
    int a;
    int b;
    A()
    {
        a=10;
        b=20;
    }
    void disp()
    {
        S.o.p(a);
        S.o.p(b);
    }
}
class Sam
{
    public static void main(String args[])
    {
        A a1 = new A(), a2 = new A();
        a1.disp();
        a2.disp();
    }
}
```

o/p 10 20 10 20

→ There are no destructors in java

Parameterized Constructor:- Initialize the Objects with different values

```
class A
{
  int a;
  int b;
  A(int x, int y)
  {
    a = x;
    b = y;
  }
  void disp()
  {
     S.o.p(a);
     S.o.p(b);
  }
}
```

```
class Sam
{
  public static void main
                (String args[])
  {
    A a1 = new A(10,20),
      a2 = new A(30,40);
    a1.disp();
    a2.disp();
  }
}
```

Copy Constructor:- Used to copy the instance variables of one object into another objects instance variables

→ Datatype of Object is class name

→ variable of a class is Object

→ Copy Constructor / Constructor overloading

Ex:-

```
class A
{
  int a;
  int b;
  A(int x, int y)
  { a = x;
    b = y;
  }
  A(A x)
  { a = x.a;
    b = x.a;
  }
  void disp()
  { S.o.p(a);
    S.o.p(b);
  }
}
class Sam
{
  public static void main(String args[])
  {
    A a1 = new A(10,20), a2 = new A(a1);
    a1.disp();
    a2.disp();
  }
}
```

Anonymous Objects:- (Nameless objects)

→ Only one Anonymous object can be created at a time

Syntax:-

```
new classname ( )
```

Ex: `new A(30,40);`

```
class A
{
    int a;
    int b;
    A(int x, int y)
    {
        a=x;
        b=y;
    }
    void disp()
    {
        S.o.p(a);
        S.o.p(b);
    }
}
```

```
class Sam
{
    public static void main(
                        String args[])
    {
        new A(30,40).disp();
    }
}
```

o/p: 30 40

Static Keyword:-

→ We used only for the purpose of Memory Management

→ The static keyword can be used with

1. Variables (class variables)
2. Methods (class Methods)
3. Static blocks
4. Static nested classes.

Illustration of Static Variable:-

```
class Student
{
    int rno;
    String name;
```

```
    static String college = "svec";
    Student (int x, int y)
    {
        sno = x;
        name = y;
    }
    void disp()
    {
        S.o.p(sno);
        S.o.p(name);
        S.o.p(college);
    }
}
class Sam
{
    public static void main (String args[])
    {
        Student s1 = new Student (10,"abc"), s2 = new Student(11,"xyz");
        s1.disp();
        s2.disp();
    }
}
```

→ The student Objects s_1 & s_2 is having only two Instance variables i.e., sno & name. whereas College is a class variable because of static keyword.

→ The class variables are created if and only if all the Objects have common property i.e., College name

Updating the class variable with Different Objects

```
class A                    (without static)
{
    int cnt;
    void incr()
    {
```

```
        cnt++;
        S.o.p(cnt);
    }
}
class Sam
{ public static void main(String args[])
  {
    A a1 = new A(), a2 = new A(), a3 = new A();
    a1.incr();
    a2.incr();
    a3.incr();
  }
}
```

o/p :
:
:

Static :-

```
class A
{
  static int cnt;
  void incr()
  {
    cnt++;
    S.o.p(cnt);
  }
}
class Sam
{ public static void main(String args[])
  {
    A a1 = new A(), a2 = new A(), a3 = new A();
    a1.incr();
    a1.incr();
    a3.incr();
  }
}
```

o/p : 1
2
3

9/12/19

Static Method :-

Static Methods can only update static variables. also known as class methods.

→ Static Methods & Static variables can be referred by using class name

Illustration of Static Method -to Update the static variable and calling the static method using class name instead of an Object.

```
class Student
{
    int rno;
    String name;
    static String college = "svec";
    Student(int x, String y)
    {
        rno = x;
        name = y;
    }
    static void change()
    {
        college = "Sri vasavi";
    }
    void disp()
    {
        S.o.p(rno);
        S.o.p(name);
        S.o.p(college);
    }
}
class Sam
{
    public static void main(String args[])
    {
        Student S1 = new Student(10, "abc"), S2 = new Student(11, "xyz");
        Student.change();
        S1.disp();
        S2.disp();
    }
}
```

Static blocks:-

If a block is started with a keyword static then it is called static block.

Syntax:- static {

}

→ Static blocks are always executed prior to the main method

→ we cannot run a java program without a main method. whereas, If our code consists of a static block. then we can run the java program without executing a main method

Illustration of Static block:-

```
class Sam
{
    public static void main (String args[])
    {
        S.o.p ("Main Method");
    }
    static
    {
        S.o.p ("static Block1");
    }
    static
    {
        S.o.p ("static Block 2");
    }
}
```

O/p: static Block 1
static Block 2
Main Method

→ Here whatever it is, first static blocks executed and then main, If there is more than one static block, they execute in order and then it goes to main method.

Illustration of a static block without a main method Execution.

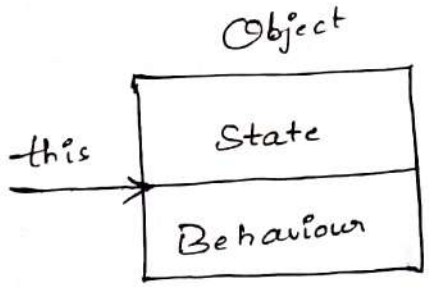
o/p:- static Block

```
class Sam
{
    static
    {
        S.o.p("static Block");
        System.exit(0);
    }
    public static void main(String args[])
    {
        S.o.p("Main Method");
    }
}
```

→ exit(0) is a method of a System class which is used to terminate the program Execution.

this Keywodd:-

this is Used to refer Current class Object



→ this is a Reference Object which will be passed implicitly to the Constructors / methods

Usage of this:-

→ this can be Use to Refer Current class object instance variables

→ this can be Use to Refer/call Current class Object methods

→ this can be Use to Refer Current class Constructor

→ this can be Used as an argument to the Method

→ this can be used as an argument to a parameterized constructor

→ this can be used to Return from a method

Referring current class instance variables using this keyword

without using this & not changing the formal parameter

```
class Std
{
    int rno;
    String name;
    String College;
    Std (int rno, String name, String College)
    {
        rno = rno;
        name = name;
        College = College;
    }
    void disp()
    {
        S.o.p(rno);
        S.o.p(name);
        S.o.p(College);
    }
}
class Sam
{
    public static void main (String args[])
    {
        Std S1 = new Std(10, "abc", "svec");
        S1.disp();
    }
}
```

o/p:- 0
null
null

→ Because the instance variables names, Constructors formal parameters names are same

→ rno = rno ; name = name; college = college

The above three statements assigned to itself but not to the instance variables. So, we will get Only Default values.

→ To avoid that, either we need to Refer Current class Objects instance variables using this / change the formal parameter names in the Constructor

Using this:-

```
class Std
{
  int rno;
  String name;
  String college;
  Std (int rno, String name,
       String college)
  {
    this.rno = rno;
    this.name = name;
    this.college = college;
  }
  void disp()
  {
    S.o.p(rno);  → Also written as S.o.p(this.rno);
    S.o.p(name);
    S.o.p(college);
  }
}
class Sam
{
  public static void main(
       String args[])
  {
    Std s1 = new std(10,"abc","svec");
    s1.disp();
  }
}
```

O/p: rno = 10
name = abc
college = svec

Without using this:-

```
class Std
{
  int rno;
  String name;
  String college;
  std(int r, String n, String c)
  {
    rno = r;
    name = n;
    College = c;
  }
  void disp()
  {
    S.o.p(rno);
    S.o.p(name);
    S.o.p(College);
  }
}
class Sam
{
  public static void main(String args[])
  {
    Std s1 = new std (10, "abc", "svec");
    s1.disp();
  }
}
```

O/p: 10
abc
svec

Rewrite the above program, add another instance variable fee of type float, create three objects for Students of S_1, S_2 and S_3. S_3 should have a fees values of 60,000.0 where as S_1 & S_2 should have a default value (have a fees of 0.0)

```
class Std
{
    int rno;
    String name;
    String college;
    float fee;
    Std(int r, String n, String c)
    {
        rno = r;
        name = n;
        College = c;
    }
    Std(int r, String n, String c, float f)
    {
        rno = r;
        name = n;
        College = c;
        fee = f;
    }
    void disp()
    {
        S.o.p (rno);
        S.o.p (name);
        S.o.p (college);
        S.o.p (fee);
    }
}
class San
{
    public static void main(String args[])
    {
        Std s1 = new Std(10, "abc", "svec"), s2 = new Std(20, "xyz", "svec");
```

```
        S3 = new  Std (30, "mno", "svec", 60000.0f);
        S1.disp();
        S2.disp();
        S3.disp();
    }
}
```

Referring / Calling Current class methods using this.

→ Displaying the Object id's using this.

```
class Std
{
    void disp()
    {
        S.o.p(this);
    }
}
class Sam
{
    public static void main(String args[])
    {
        Std S1 = new std(), S2 = new std(), S3 = new Std();
        S.o.p (S1);
        S1.disp();
        S.o.p (S2);
        S2.disp();
        S.o.p (S3);
        S3.disp();
    }
}
```

O/p:
```
Std@ 1ea2dfc
Std@ 1ea2dfc
Std@ 17182c1
Std@ 17182c1
Std@ 13f5d07
Std@ 13f5d07
```

Referring / Calling the current class's methods. Using this.

```
class Std
{
    void disp()
    {
        S.o.p(this);
    }
    void show()
    {
        S.o.p("disp() from show()");
        this.disp();
    }
}
class Sam
{
    public static void main(String args[])
    {
        Std s1=new std(), s2=new std(), s3=new Std();
        S.o.p(s1);
        s1.show();
        S.o.p(s2);
        s2.show();
        S.o.p(s3);
        s3.show();
    }
}
```

o/p:

```
Std@1ea2dfc
disp() from show()
std@ 1ea2dfc
Std@ 171821
disp() from show()
std@ 171821
std@ 13fedo7
disp() from show()
std@ 13fedo7
```

14/12/19

Calling Default Constructor Using this in the parameterised Constructor.

```
class A
{
    int x;
    A()
    {
        S.o.p("Default Constructor");
    }
    A(int i)
    {
        this();
        x = i;
        S.o.p("parameterised Constructor");
        S.o.p(x);
    }
    public static void main(String args[])
    {
        A A1 = new A(3);
    }
}
```

o/p:- Default Constructor
parameterised Constructor
3

Note:- → Call to this must be first statement in Constructor

Calling parameterised Constructor Using this in the Default Constructor

```
class A
{
    int x;
    A()
    {
        this(3);
        S.o.p("Default Constructor");
    }
    A(int i)
    {
        x = i;
        S.o.p("parameterised Constructor");
        S.o.p(x);
    }
    public static void main(String args[])
    {
        A A1 = new A();
    }
}
```

o/p:- parameterised Constructor
3
Default Constructor

Passing this as a parameter to a method

```
class A
{
    void get()
    {   S.o.p(this);
        print(this);
    }
    void print(A o)
    {
        S.o.p("this is a parameter
                to a method");
        S.o.p(o);
    }
    public static void main(String args[])
    {
        A A1 = new A();
        A1.get();
        S.o.p(A1);
    }
}
```

O/p:

```
A@1ea2d4f2
this is a parameter to a method
A@1ea2d4f2
A@1ea2d4f2
```

Returning this from a method

```
class A
{
    A get()
    {
        return this;
    }
    public static void main(String args[])
    {
        A A1 = new A();
        A A2 = A1.get();
        S.o.p(A1);
        S.o.p(A2);
    }
}
```

O/p:

14/12/19

Array of Objects:-

Syntax:- classname Referencename [ ] = new classname[size]

Ex:- Sam s[ ] = new Sam[10];

→ The above statement creates 10 object References

→ The Memory for each and every Object can be created in the following way.

```
for (int i=0; i<10; i++)
{
    s[i] = new Sam();
}
```

Write a Java program for the class student with Instance variables rno, name and a common property College. The two Instance Methods get and display. Create 5 student Objects, Get the Details & display the details

```
import java.util.Scanner;
class Std
{
    int rno;
    String name;
    static String College = "svec";
    void get()
    {
        Scanner sc = new Scanner (System.in);
        System.out.println (" Enter rno");
        rno = sc.nextInt();
        System.out.println (" Enter name");
        name = sc.next();
    }
    void disp()
    {
        System.out.println (rno);
```

```
        System.out.println(name);
        System.out.println(college);
    }
}
class Sam
{
    public static void main(String args[])
    {
        Std s[] = new Std[5];
        for(int i=0; i<5; i++)
        {
            s[i] = new Std();
        }
        for(i=0; i<5; i++)
        {
            S.o.p("Display Student details");
            s[i].get();
        }
        for(i=0; i<5; i++)
        {
            s[i].disp();
        }
    }
}
```

Java program to display the bank details

```
import java.util.Scanner;
class Bank
{
    int acno;
    float balance;
    Static String bname = "Radhapadu";   // branch name
    void get()
    {
        Scanner sc = new Scanner(System.in);
```

```
        S.o.p (" Enter account number");
        acno = sc.nextInt();
        S.o.p (" Enter bal:");
        balance = sc.nextFloat();
    }
    void disp()
    {
        S.o.p (acno);
        S.o.p (balance);
        S.o.p (branch name);
    }
    void wd (float amt);
    {
        if (bal < amt)
        {
            S.o.p (" Insufficient");
        }
        else
        {
            bal = bal - amt;
        }
    }
    void de (float amt)
    {
        bal = bal + amt;
    }
}
class Sam
{
    public static void main (String args[])
    {
        Bank S[] = new Bank[2];
        for (int i = 0; i < 2; i++)
        {
            S[i] = new Bank();
        }
        for (i = 0; i < 2; i++)
```

```
        {
            s[i].get();
        }
        for(i=0;i<2;i++)
        {
            s[i].disp();
        }
        Scanner sc=new Scanner(System.in);
        S.o.p("Enter acno to deposit:");
        int n=sc.nextInt();
        for(i=0; i<2; i++)
        {
            if(s[i].acno==n)
            {
                S.o.p("Enter amount to deposit:");
                s[i].dep(b); float b = sc.nextFloat();
                s[i].disp();
                break;
            }
        }
        if(i==2)
            S.o.p("Invalid Account");
    }
}
```

Java program to display Bank details in MenuOrder

```
import java.util.Scanner;
class Bank
{
    int ano;
    float balance;
    Static String bank = "Andhra Bank";
    void get()
    {
        Scanner sc=new Scanner(System.in);
        S.o.p("Enter ano");
```

```
        S.o.p (" Enter bal:");
        bal = sc.nextFloat();
    }
    void Balenqry()
    {
        S.o.p (Bal);
    }
    void disp()
    {
        S.o.p(ano);
        S.o.p( bal);
        S.o.p( bank);
    }
    void wd(float amt);
    {
        if (bal < amt)
        {
            S.o.p ("Insufficient Balance");
        }
        else
        {
            bal = bal - amt;
        }
    }
    void dep(float amt)
    {
        bal = bal + amt;
    }
}
class Sam
{
    public static void main (String args [])
    {
```

16/12/19

Nested / Inner classes:- If we declare a class inside another class is known as Nested / Inner class. The Difference between Nested and Inner class is Static

→ Non-static Nested class is known as Inner class

→ The Advantages of Nested classes are

* Grouping Logically Related classes
* Code optimisation

Types of Nested classes:-

→ There are two types of Nested classes

① Nonstatic Nested class (Inner class)
② Static Nested class

Non static Nested classes

It is divided into three types

① Memba Inner class
② Anonymous Inner class
③ Local Inner class

Type	Description
① Memba Inner class	A class created within the class and outside the Method
② Anonymous Inner class	A class created for implementing an interface or ~~abstra~~ extending an abstract class
③ Local Inner class	A class declared inside the method of another class

4) Static Nested class:-

A static class declared within the class

Member Inner class:-

Syntax:

```
class Outer
{
    // Code
    class inner
    {
        // Code
    }
    // Code
}
```

Ex:

```
class Outer1
{
    int a = 10;
    class Inner1
    {
        void msg()
        {
            S.o.p("Hello world");
            S.o.p(a);
        }
    }
    public static void main(String args[])
    {
        Outer1 O1 = new Outer1(); // Object of outer class
        Outer1.Inner i1 = O1.new Inner1();
        i1.msg();
    }
}
```

Note: The Inner class Methods can access outer class members.

Hello world
10

O/p: Hello world
10

Local Inner class:-

Syntax:

```
class Outer
{
    // code
    void method()
    {
        class inner
        {
            // code
        }
    }
}
```

Ex:

```
class Outer
{
    int a = 30;
    void display()
    {
        class Inner
        {
            void msg()
            {
                S.o.p("Hello world");
                So.p(a);
            }
        }
        Inner i = new Inner();
        i.msg();
    }
    public static void main(String args[])
    {
        Outer o1 = new Outer();
        o1.display();
    }
}
```

Hello world
30

O/p: Hello world
30

Static Nested class: It can access only static member of outer class

Ex:

```
class Outer
{
    static int a=10;
    static class Inner
    {
        void msg()
        {
            S.o.p("Hello World");
            S.o.p(a);
        }
    }
    public static void main(String args[])
    {
        Outer.Inner i1 = new Outer.Inner();
        i1.msg();
    }
}
```

Hello World
10

18/12/19

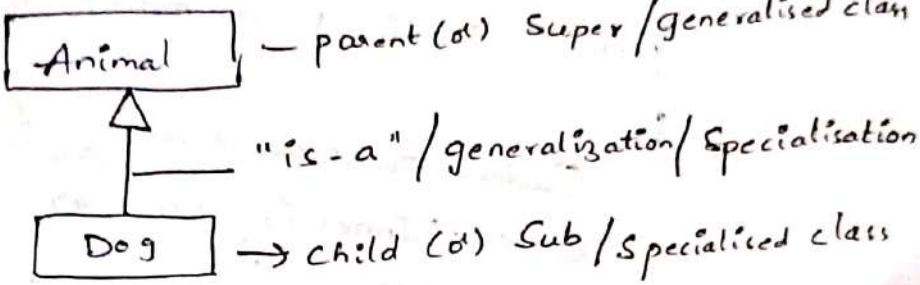
Unit-3

Inheritance & Exception Handling

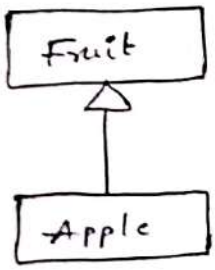
Inheritance:- If an Object acquires the properties of the parent Object then it is called Inheritance.

→ Inheritance is "is-a" kind of Relationship also known as parent-child Relationship which is Represented by a Symbol ⟶

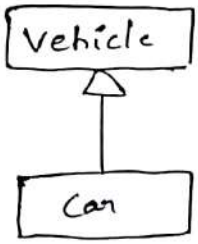
Ex: ①



②



③



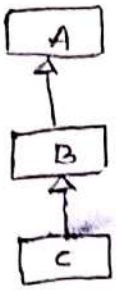
Syntax of Inheritance:-

```
class Subclass (child) extends Superclass (parent)
{
    // Code
}
```

Types of Inheritances:-

1. Single
2. Multilevel
3. Hierarchical

1. Single 2. Multilevel 3. Hierarchical:



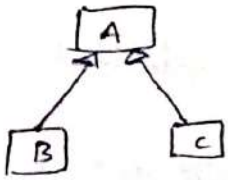


Illustration of Single Inheritance:-

```
class Single Animal
{
    void eat()
    {
        S.o.p("Eating...");
    }
}
class Dog extends Animal
{
    void Bark()
    {
        S.o.p("Barking...");
    }
}
class Single
{
    public static void main(String args[])
    {
        Dog d = new Dog();
        d.eat();
        d.Bark();
    }
}
```

Example for Multilevel Inheritance:-

```
class Animal
{
    void eat()
    {
        S.o.p("Eating...");
    }
}
class Dog extends Animal
{
    void bark()
    {
        S.o.p("Barking....");
    }
}
class Puppy extends Dog
{
    void weap()
    {
        S.o.p("Weaping...");
    }
}
class Multilevel
{
    public static void main(String args[])
    {
        Puppy p = new Puppy();
        p.eat();
        p.bark();
        p.weap();
    }
}
```

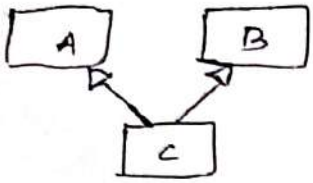
eating
Barking
weaping

Example for Hierarchical Inheritance :-

```
class Animal
{
    void eat()
    {
        S.o.p("Eating...");
    }
}
class Dog extends Animal
{
    void bark()
    {
        S.o.p("Barking...");
    }
}
class Cat extends Animal
{
    void sleep()
    {
        S.o.p("sleeping...");
    }
}
class Hierarchical
{
    public static void main(String args[])
    {
        Dog d = new Dog();
        d.eat();
        d.bark();
        Cat c = new Cat();
        c.eat();
        c.sleep();
    }
}
```

Eating...
Barking...
eating...
sleeping...

→ Java does not Supports multiple Inheritance through classes



→ If class c is Inherited from two classes A, B

Ex:

```
class A
{
    void msg()
    {
        S.o.p ("Eating ...");
    }
}
class B
{
    void msg()
    {
        S.o.p ("Barking ...");
    }
}
class c extends A, B
{
    void weap()
    {
        S.o.p ("weaping ...");
    }
}
class multiple
{
    public static void main (String args[])
    {
        c c1 = new c();
        c1.msg();
        c1.weap();
    }
}
```

Ex:-

→ class A and B both are having the methods msg() with the same prototype. The Compiler gets an ambiguity while calling the method msg()

→ The ambiguity due to multiple Inheritance can be Resolved by using Interfaces

Super Keywold:-

The Super Keywold is a Reference Variable which Refers immediate parent Object

Usage of Super:-

→ Super can be Used to Refer Instance variables of immediate parent class Object.

→ Super can be Used to Refer parent class methods

→ Super can be Used to Refer parent class Constructd

Super to Refer parent class Instance Variables:-

```
class A
{
    int cnt=10;
}
class B extends A
{
    int cnt=20;
    Void disp()
    {
        S.o.p(Super.cnt);
        S.o.p(Cnt);
    }
}
class Single
{
    Public Static Void main(String args[])
    {
        B b1 = new B();
        b1.disp();
    }
}
```

O/P: 10
20

Super to Refer parent class methods :-

```
class A
{
   void disp()
   {
      S.o.p("AAAAAA");
   }
}
class B extends A
{
   void disp()
   {
      super.disp();
      S.o.p("BBBBBB");
   }
}
class Single
{
   public static void main(String args[])
   {
      B b1 = new B();
      b1.disp();
   }
}
```

AA AAAA
BBBBBB

Super to Refer parent class Constructor :-

→ If parent class and child class both are having Constructors, Eventhough we Create an Object for child class, First the parent class Constructor is invoked then child class Constructor

Ex:-

```
class A
{
   A()
   {
      S.o.p("AAA");
   }
}
class B extends A
{
   B()
   {
      S.o.p("BBB");
   }
}
class Single
{
   public static void main(String args[])
   {
      B b1 = new B();
   }
}
```

O/P: AAA
BBB

Adding Super Keyword

```
class A
{
  A()
  {
    S.o.p("AAA");
  }
}
class B extends A
{
  B()
  {
    super();
    S.o.p("BBB");
  }
}
class Single
{
  public static void main(String args[])
  {
    B b1 = new B();
  }
}
```

o/p: AAA
BBB

Without calling child class Constructor

```
class A
{
  A()
  {
    S.o.p("AAA");
  }
}
class B extends A
{
}
class Single
{
  public static void main(String args[])
  {
    B b1 = new B();
  }
}
```

o/p: AAA

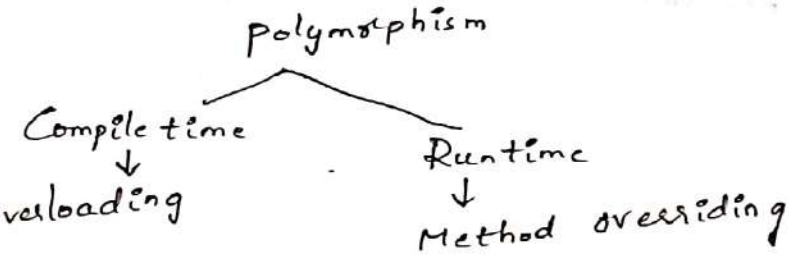
19/12/19

Calling Super class parameterized Constructor from the child class parameterized Constructor

```
class Emp
{
  int id;
  String name;
  Emp(int id, String name)
  {
    this.id = id;
    this.name = name;
  }
}
class Mgr extends Emp
{
  float Sal;
  Mgr(int x, String y, float z)
```

```
    {
        super(x,y);
        sal = z;                    1   abc   1000.0
    }
    void disp()
    {
        S.o.p(id+"  "+name+"  "+sal);
    }
}
class Sam
{
    public static void main(String args[])
    {
        Mgr m = new Mgr(1,"abc", 1000.0f);
        m.disp();
    }
}
```

20/12/19



Method Overloading :- / Code Replacement :-

If two (or) more methods having the Same name but different number of arguments / different type of arguments (prototype of the Method) is known as Method Overloading.

→ The Return type of Methods is not Considered in Method Overloading.

Ex: Illustration of Method Overloading ~~by changing~~ with different no. of arguments.

```
class MethodOverloading.
{
    static void add(int a, int b)
    {
```

```
        S.o.p(a+b);
    }
    static void add(int a, int b, int c)
    {
        S.o.p(a+b+c);
    }
    public static void main(String args[])
    {
        MethodOverloading.add(2,3);
        MethodOverloading.add(2,3,4);
    }
}
```

Illustration Method Overloading by changing type of arguments.

```
class Met
{
    static void add(int a, int b)
    {
        S.o.p(a+b);
    }
    static void add(float a, float b, float c)
    {
        S.o.p(a+b+c);
    }
    public static void main(String args[])
    {
        Met.add(2,3);
        Met.add(2.0f, 3.0f, 4.0f);
    }
}
```

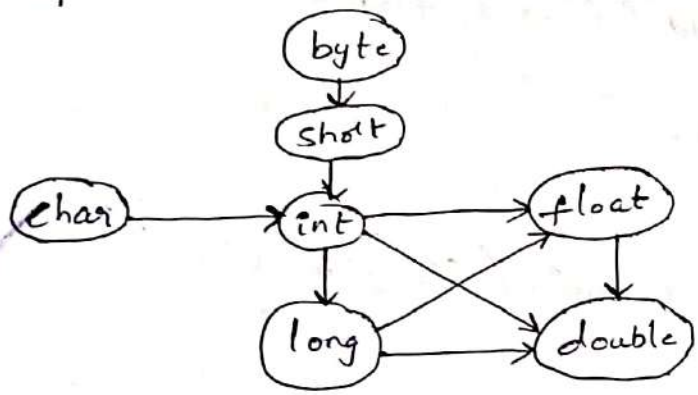
→ Method Overloading does not consider Returntype because

```
class Met
{
    static int add(int a, int b)
```

```
    return (a+b);
}
static float add (int a, int b)
{
    return (a+b);
}
public static void main (String args[])
{
    S.o.p (Met.add (2,3));
    S.o.p (Met.add (4,5));
}
}
```

→ This program raises error because the formal arguments & Methods are same it Results in Ambiguity.

Type Promotions in Method Overloading:



Ex:

```
class Met
{
    static long add(long a, long b)
    {
        return (a+b);
    }
    static double add (double a, double b)
    {
        return (a+b);
    }
    public static void main(String args[])
```

```
    {
        S.o.p(Met.add(2,3));
        S.o.p(Met.add(4.0f, 5.0f));
    }
}
```

Ex:
```
class Met
{
    Static long add(long a, int b)
    {
        return (a+b);
    }
    Static long add(int a, long b)
    {
        return (a+b);
    }
    public static void main(String args[])
    {
        S.o.p(Met.add(2,3));
        S.o.p(Met.add(4,5));
    }
}
```

→ The program Results in error because 2 can be Considered as int & can be promoted as long as well as 3 can be Considered as int & promoted as long.

Overloading the Main Method:-

→ The Main Method can be Overloaded but the method which is having String array can be Executed.

Ex:
```
class Met
{
    public static void main(String a)
    {
        S.o.p("Hello *");
    }
    public static void main(int a)
    {
        S.o.p("Hello ?");
    }
```

```
    public static void main (String args[])
    {
        S.o.p (" Hello! ");
    }
}
```

O/P:- Hello!

Method Overriding:- / Code Refinement:

If The name of the methods and prototype of the methods are same then it is called Method Overriding.

→ Method Overriding is between 2 classes which are having parent-child relationship (Inheritance)

Ex:- Illustration of Method Overriding:

```
class Vehicle
{
    void run()
    {
        S.o.p (" Vehicle is running ");
    }
}
class Car extends Vehicle
{
    void run()
    {
        S.o.p (" Car is running ");
    }
}
class MethodOveriding
{
    public static void main (String args[])
    {
        Car c = new Car();
        c.run();
    }
}
```

O/P:- Car is running

→ The run() method of child class (Car class) Overrides the run method of Super class (Vehicle class)

23/12/19

23/12/19

Unit - III

Abstract classes & Interfaces

→ Abstraction in Java can be achieve through one is

① Abstract classes &

② Interfaces

→ Abstraction is nothing but hiding the internal details.

→ Another definition Abstraction is nothing but what the Object does instead how it does it.

Abstract classes:-

→ The class declaration of Abstract is using the keyword Abstract then it is called Abstract class

→ Abstract class may have either Abstract Methods of Non-Abstract Methods

→ Non-Abstract Method is also known as Concrete Method

→ The Method which is having code (body) is called Non-Abstract Otherwise it is Abstract

→ Abstract class can have Zero (or) more Abstract methods.

Syntax of Abstract class:-

```
abstract class classname
{
    abstract methods
    non abstract methods
}
```

Syntax for abstract method:

```
abstract void methodname();
```

Rules of abstract class:-

→ It should starts with keyword Abstract
→ Abstract class contains Abstract & Non-Abstract methods
→ Abstract classes cannot be instantiated (i.e., we cannot create Object for Abstract class)
→ Abstract class should contains Static methods, Constructors, instance variable etc
→ The Abstract class can be extended by another class
→ And that class should implement the Abstract class methods.

Ex: Abstract class with Abstract Method

```
Abstract class Vehicle
{
    abstract void run();
}
class Car extends Vehicle
{
    void run()
    {
        S.o.p("car is running");
    }
}
class AbstractClassDemo
{
    public static void main(String args[])
    {
        Car c = new Car();
        c.run();
        Vehicle v = new Car();
        v.run();
    }
}
```

O/p: car is Running.

→ The child class Object (the object of car) can be assigned to base class Reference (Vehicle)

Ex: Abstract class with Abstract Method, Non-Abstract Method and Constructor

```
abstract class Vehicle
{
    Vehicle()
    {
        S.o.p("Abstract class Vehicle");
    }
    abstract void run();
    void changegear()
    {
        S.o.p("Gear changed");
    }
}
class Car extends Vehicle
{
    Car()
    {
        S.o.p("car is running");
    }
    Void run()
    {
        S.o.p("Vehicle is running");
    }
}
class Bike extends Vehicle
{
    Bike()
    {
        S.o.p("Bike is running");
    }
```

```
    void run()
    {
        S.o.p (" Bike is running");
    }
}
class Sam
{
    public static void main (String args[])
    {
        Vehicle v;
        v = new Car();
        v.run();
        v.changegear();
        v = new Bike();
        v.run();
        v.changegear();
    }
}
```

Output:
```
Abstract class vehicle
Car is running
Car is running
Gear changed
Abstract class Vehicle
Bike is running
Bike is running
Gear changed
```

Ex:
```
abstract class Bank
{
    Bank()
    {
        S.o.p (" Bank");
    }
    abstract void getInterest();
}
class SBI extends Bank
{
    SBI()
    {
        S.o.p("SBI");
    }
    void getInterest()
    {
        S.o.p(" SBI interest");
```

```
class HDFC extends Bank
{
    HDFC()
    {
        S.o.p("HDFC");
    }
    void getInterest()
    {
        S.o.p("HDFC Interest");
    }
}
class ICICI extends Bank
{
    ICICI()
    {
        S.o.p("ICICI");
    }
    void getInterest()
    {
        S.o.p("ICICI Interest");
    }
}
class Sam
{
    public static void main(String args[])
    {
        Bank b;
        b = new SBIC();
        b.getInterest();
        b = new HDFC();
        b.getInterest();
        b = new ICICI();
        b.getInterest();
    }
}
```

27/12/19

Interfaces:-

→ Interfaces is also a blue print of a class.

→ Abstract classes 0 - 100%.

Interfaces 100%.
- variables – public, static, final
- Methods – public, abstract.

→ final keywords is used to Initialize/declare a Constant

* Interface is a blue print of a class
* An Interface Consists of Only abstract methods & variables.
* Using Interfaces we can achieve 100% abstraction.
* We can achieve Multiple Inheritance
* The keyword Interface is used to create an Interface

Syntax of Interface

```
interface interfacename
{
    // variables
    // methods
}
```

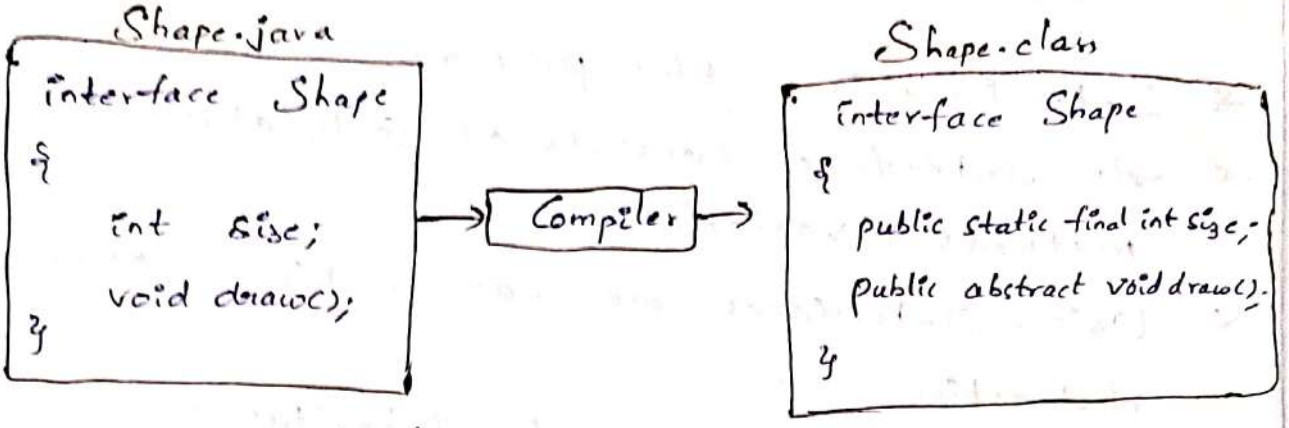
Ex:
```
interface Shape
{
    int size;
    void draw();
}
```

→ Just like Abstract classes we cannot Instantiate interfaces whereas we can create references

→ Every Interface can be Implemented by another class using a keyword "implements"

→ The class which implements the Interface class is called Implementation class

→ The internal Representation of an Interface by the Compiler



→ By Default, the interface variables are public, static, final and the interface Methods are public, abstract

→ The keywold final is used to Create Constants that means we cannot change it

Ex: final int i = 10;

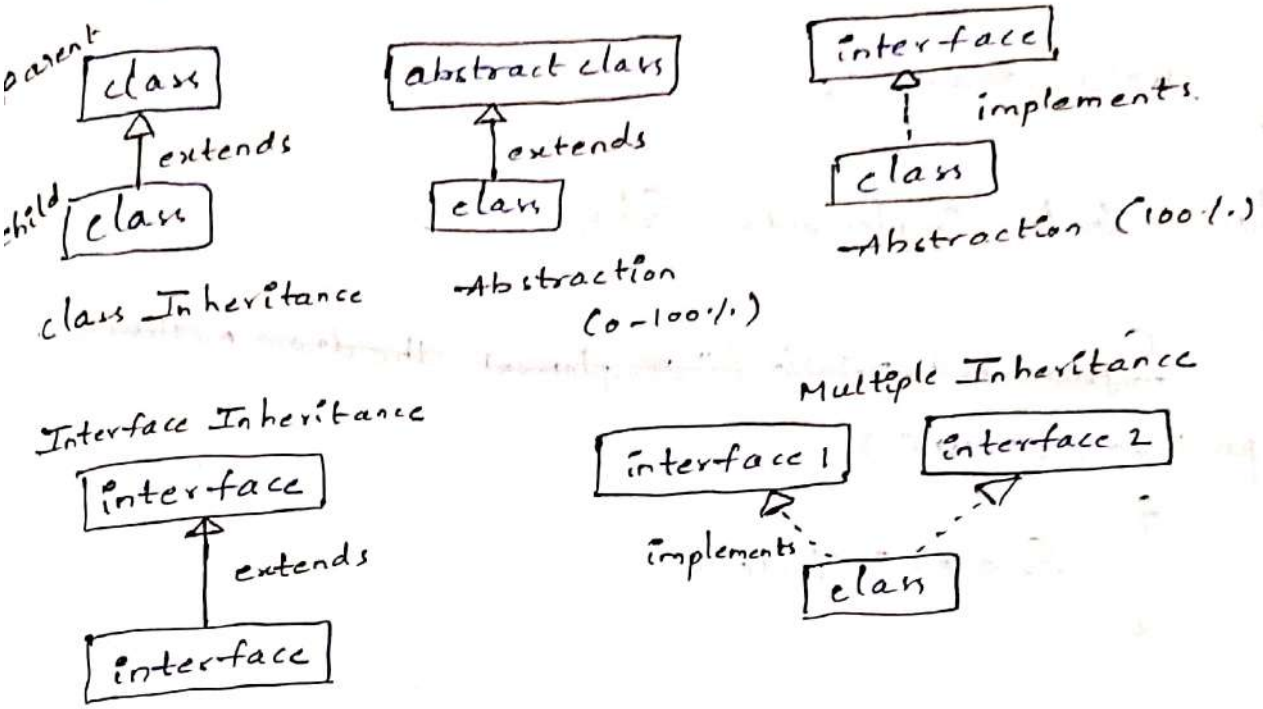
→ final can be Used to avoid Inheritance that means. If we add a keywold final. prid to a class then It cannot be Extended by another class

Ex: final class Shape
{
//
}
X class Circle extends Shape
{
}

It gives an Error

→ class circle cannot extend Shape because Shape is -final class.

→ The Relationship between Interfaces, Abstract classes and classes :-



Comparison between Interfaces and Abstract classes :-

Interfaces	Abstract classes
→ It starts with a keywold Interface	→ It starts with a keywold Abstract.

Ex:- Interface

~~Create an Interface~~

```
interface Shape
{
    void draw();
}
class Circle implements Shape
{
    ~~System.out.println("Implement the draw method");~~
  public void draw()
    {
        S.o.p("Circle");
    }
}
class Sam
{
    public static void main(String args[])
    {
        Shape s = new Circle();
        s.draw();
    }
}
```

Example for Interface Inheritance:-

```
interface Shape1
{
    void draw1();
}
interface Shape2 extends Shape1
{
    void draw2();
}
class Circle implements Shape2
```

```
}
public void draw1()
    {
        S.o.p(" circle Shape1 ");
    }
public void draw2()
    {
        S.o.p(" circle Shape2 ");
    }
}
class Sam
{   public static void main (String args[])
    { Shape2 s = new circle();
      s.draw1();
      s.draw2();
    }
}
```

Example of Multiple Inheritance through Interfaces

```
interface Shape1
{
   void draw();
}
interface Shape2
{
   void draw();
}
class Circle implements Shape1, Shape2
{
    void draw()
    {
        S.o.p (" circle Shape1, shape2 ");
    }
}
class Sam
{
    public static void main (String args[])
    {
        circle c = new circle();
    }   c.draw();
```

28/12/19

```
interface Bank
{
    int i;
    void interest();
}
class SBI implements Bank
{
  public void interest()
    {
        S.o.p("SBI interest");
                    +i
    }
}
class Hdfc implements Bank
{
  public void interest()
    {
        S.o.p("Hdfc interest");
                    +i
    }
}
class Icici implements Bank
{
  public void interest()
    {
        S.o.p("Icici interest");
                    +i
    }
}
class Sam
{
    public static void main(String args[])
    {
        Bank b;
        b=new SBI;
        b.interest();
        b=new Hdfc;
        b.interest();
        b=new Icici;
        b.interest();
    }
}
```

Example for Interface and abstract class

```
interface Bank
{   int i=10;
    public void interest();
    public void print();
}
abstract class sbi implements Bank
{   public void interest()
    {
        S.o.p ("SBI"+i);
    }
}
class Sbh extends sbi
{   public void print()
    {   S.o.p ("SBH"+i);
    }
}
class San
{   public static void main(String args[])
    {   Bank b;
        b= new Sbh();
        b.interest();
        b.print();
    }
}
```

O/p: SBI 10
SBH 10

Packages:-

Package is a directory / folder

Usages of package:

→ Grouping Relevant classes / Interfaces in one unit
→ Easy to maintain
→ To avoid naming conflicts / collisions

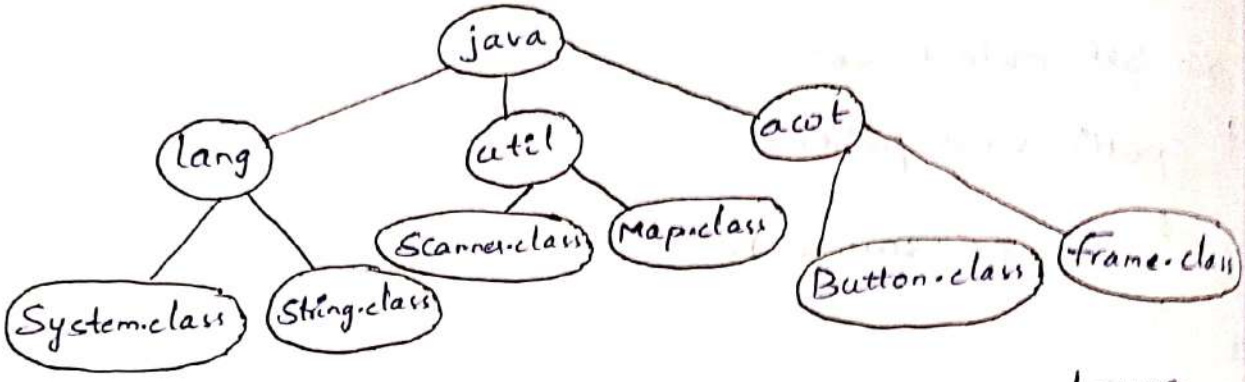
→ There are two types of package
① Built in packages (or) predefined packages
② User defined packages

Built-in-packages:-

These are developed by java developers

→ The root package for all the pre-defined packages is java

→ Package name starts with lower case



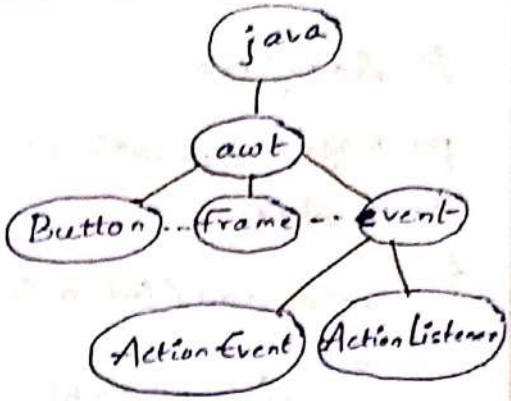
→ Importing packages and using the classes Interfaces of those packages in our program can be done in one of three ways

① import packagename.*;
② import packagename.classname;
③ Fully qualified classname without an import statement

① Ex: import java.util.*;
② Ex: import java.util.Scanner;
③ Ex: java.util.Scanner sc=new java.util.Scanner(System.in);

→ If we import a package in our program then only the classes & Interfaces of that package will only be imported but not the classes & Interfaces of Sub package

Ex: import java.awt.*;



→ The classes and Interfaces of event package cannot be imported into the program.

Ex: import java.awt.*;
import java.awt.event.*;

User-defined Packages:-

We can create our own packages to group the Relevant user-defined classes & Interfaces.

→ A package can be created with a keyword package.

package packagename;

Ex: package Operations;

Java program Structure import packages.

```
package packagename;
import packagename;
class classname
{
    //
}
```

→ Description:
Creating a package Operations and adding the classes into it.

// Add.java

```
package operations;
class Add
{
    void add(int a,int b)
    {
        S.o.p(a+b);
    }
}
```

// Sub.java

```
package Operations;
class Sub
{
    void sub(int a, int b)
    {
        S.o.p(a-b);
    }
}
```

Compilation of the above two programs:-

javac -d . Add.java Sub.java

→ The -d option creates a directoly (package operations) and the '.' (current working directoly)

Importing Package Operations and its classes:-

```
import Operations.Add;
import Operations.Sub;
class Sam
{
    public static void main(String args[])
    {
        Add a1 = new Add();
        a1.Add(2,3);
        Sub s1 = new Sub();
        s1.Sub(2,3);
    }
}
```

o/p 5
-1

With fully Qualified:-

```
class Sam
{   psvm(String args[])
    {   operations.Add a1 = new Operations.Add();
```

```
        a1.add(2,3);
        Operations.Sub s1=new Operations.Sub();
        s1.sub(2,3);
    }
}
```

o/p: 5
-1

Importing all classes in Operations

```
import Operations.*;
class Sam
{   psvm(String arg[])
    {
        Add a1=new Add();
        a1.add(2,3);
        Sub s1=new Sub();
        s1.sub(2,3);
    }
}
```

o/p: 5
-1

Setting Up the class path:-

The class path is User Environment variable which is used to set the path of classes

→ We need to set up a class path if packages are in One directoly and the source code in which the packages have been imported, in the Other directoly.

→ For Example, The source file Sam.java is available Under the directoly c:\Users\svec\Desktop\CSE-Ds. The package Operations is Under D:\CSE-D

(java c -d D:\CSE-D Add.java Sub.java)

→ In order to make use of Add class & Sub class

We need to setup the class path. There are two ways to setup a class path. (1) One way is by using Set - classpath command at the Command prompt

→ (2) by editing the Environment variable classpath (User variable)

→ Set classpath = %.classpath%. D:\CSE-D; (for 1st path)

⇒ Ex:-

```
package movies.telugumovies;
class Kushi
{
    void pspk()
    {
        S.o.p("power star");
    }
}
package movies.telagu;
class Mahesh
{
    void Mb()
    {
        S.o.p("Supa star");
    }
}
package movies.hindi;
class Siddhu
{
    void Sid()
    {
        S.o.p("EK villain");
    }
}
package movies.hindi;
class Varun
{   void vd()
    {   S.o.p("Street Dancer 3D");
    }
}
```

```
impolt  movies.telugu.*;
impolt  movies.hindi.*;
class Sam
{   psvm(String args[])
    {
        Kushi  k = new Kushi();
        k.pspk();
        Mahesh m = new Mahesh();
        m.Mb();
        Siddhu s = new Siddhu();
        s.sid();
        Varun  v = new Varun();
        v.vd();
    }
}
```

30|12|19

Exception Handling:-

The Exception Handling in Java is a powerful Mechanism to handle the runtime errors. So that normal flow of program Execution will not be disrupted.

→ An Exception is an event (object) which disrupts the flow of Execution of a program

→ An Exception is an abnormal Condition.

→ Let us Consider a Scenario.

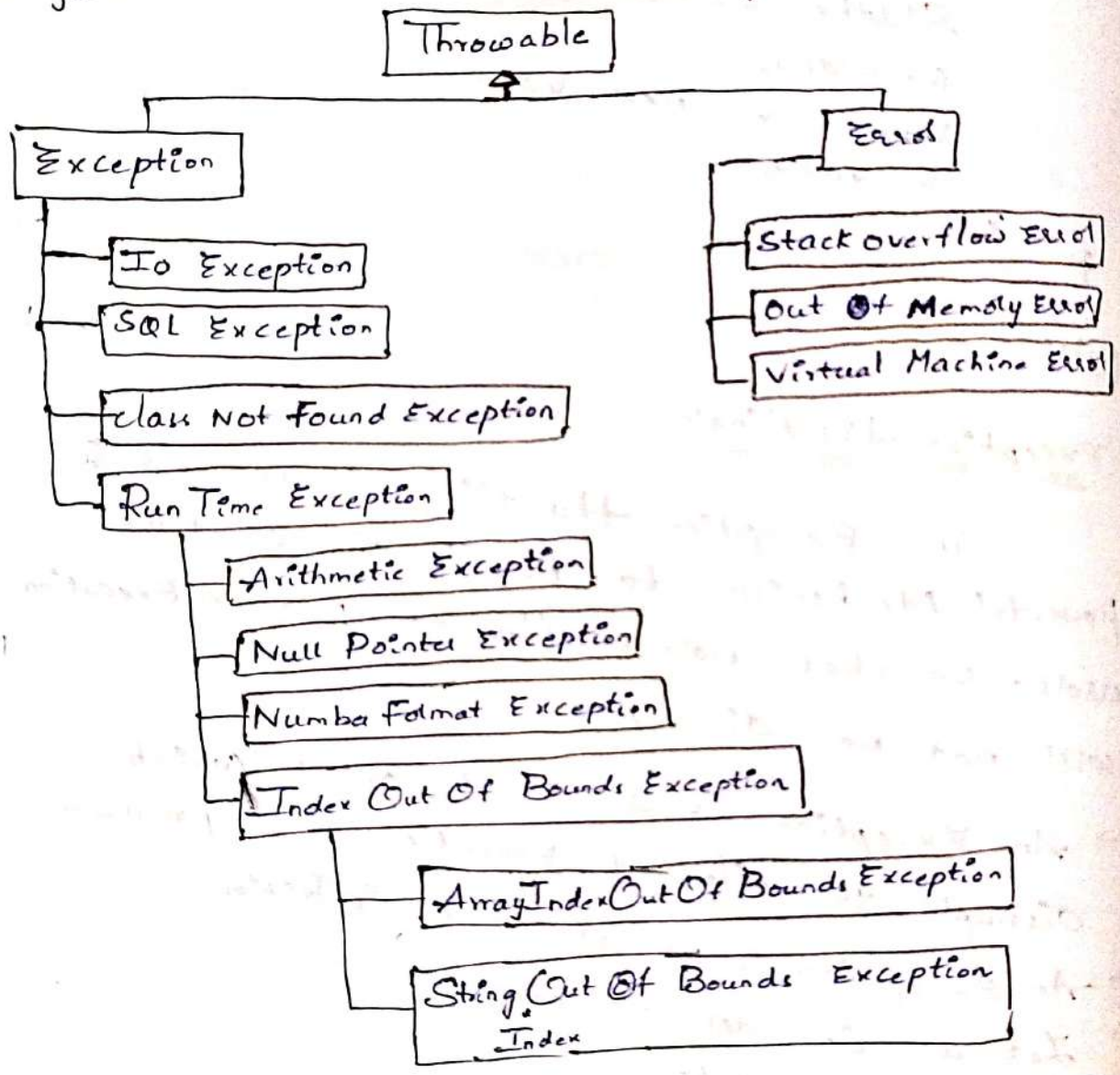
```
Statement 1;
Statement 2;
Statement 3;
Statement 4;
statement 5; → // Exception occurs
Statement 6;
Statement 7;
Statement 8;
Statement 9;
Statement 10;
```

→ At Statement 5; if an Exception raises then the Statements from 6 to 10 will not get executed

→ If we perform Exception Handling then Statement 6 to 10 will get executed, the normal flow of Execution will be maintain.

Hierarchy of Java Exception classes:-

java.lang



→ The Exception class Hierarchy is available in the package java.lang.

Types of Exceptions

There are mainly two types of Exceptions

① Checked Exceptions ② Unchecked Exceptions

Checked Exceptions

The classes which are directly inherited from Throwable class Except RunTimeException are known as checked Exceptions.

→ which are checked during Compile time

Unchecked Exceptions

The classes which are inherited from RunTimeException and checked during Runtime are called Unchecked Exceptions.

Example of Checked Exceptions

IOException, ClassNotFoundException, SQLException

Example of Unchecked Exceptions

ArithmeticException, NullPointerException, NumberFormatException, ArrayIndexOutOfBoundsException, StringIndexOutOfBoundsException

→ Error is irrecoverable, we cannot handle it.

Ex: OutOfMemoryError, StackOverflowError, VirtualMachineError

→ The keywords of Java Exception Handling are

→ They are 5 important keywords.

	Keywords	Description
1.	try	The try keyword is used to specify a block where we should write a set of statements, they may lead to an Exception. The try block must be followed by a catch block or finally block. it means we cannot write try alone

2. catch — The catch block is used to handle Exception. It must be preceeded by a try block. It can be followed by a finally block. It cannot be written alone

3. finally — The finally block is used to Execute important code of the program. It is executed whether an Exception is ~~raised~~ handled or not

3/1/20

4. throw — The throw keyword is used to throw an Exception and it is a Statement.

5. throws — This keyword is used to declare the Exceptions and it should be used after the method signature.

Common Scenarios of Exceptions:-

Scenario-1:- Divide by zero

```
int a = 50/0; // ArithmeticException
```

Scenario-2:- Accessing array Element beyond the size

```
int a[][] = new int [5];
a[50] = 10 // ArrayIndexOutOfBoundsException.
```

Scenario-3:- Finding the length of null string.

```
String s = null;
S.o.p(s.length()); // NullPointerException
```

Scenario-4:- Accepting the input (integer)

```
String s = "abc"
int i = Integer.parseInt(s) // NumberFormatException
```

Example program for Illustrating Exception Handling Using try, catch blocks.

```
class Sam
{
    public static void main (String args[])
    {
        try
        {
            int a = 50/0;
            S.o.p(a);
        }
        catch (ArithmeticException e)
        {
            S.o.p(e);
        }
        S.o.p (" Rest of the code");
    }
}
```

Syntax of try & catch:-

```
try
{
    // statements
}
catch (Exception class ref)
{
    // statements
}
```

o/p: java.lang.ArithmeticException : /o
Rest of the code.

② Write the program for ArrayIndexOutOfBound.

```
class Sam
{
    public static void main(String args[])
    {
        try
        {
            int a[][] = new int [5];
            a[50]=10;
        }
        catch (ArrayIndexOutOfBounds e)
        {
            S.o.p(e);
        }
    }   S.o.p(" Rest of the code");
}
```

O/P: java.lang.ArrayIndexOutOfBounds : 50
Rest of the code.

(3) program for NullPointerException.

```
class Sam
{
    public static void main(String args[])
    {
        try
        {
            String s=null;
            S.o.p(s.length());
        }
        catch(NullPointerException e)
        {
            S.o.p(e);
        }
        S.o.p("Rest of the code");
    }
}
```

O/P: java.lang.NullPointerException : null
Rest of the Code

(4) program for NumberFormatException

```
class Sam
{
    public static void main(String args[])
    {
        try
        {
            String s="abc";
            int i=Integer.parseInt(s);
        }
        catch(NumberFormatException e)
        {
            S.o.p(e);
        }
        S.o.p("Rest of the code");
    }
}
```

O/P: java.lang.NumberFormatException : abc
Rest of the Code

Illustration of finally block:-

```
class Sam
{ public static void main(String args[])
  {
     try
     {
        int a = 50/2;
        S.o.p(a);
     }
     catch (ArithmeticException e)
     {
        S.o.p(e);
     }
     finally
     {
        S.o.p("finally block");
     }
     S.o.p("Rest of the code");
  }
}
```

O/P: 25
finally block
Rest of the code

→ The finally block will get executed always if an exception is raised or not raised

→ If an Exception is raised the catch block and then finally block will get executed

→ If an Exception is not raised then the try block and finally block will get executed

Illustration of Multiple Catches:

→ We can write Multiple catches for a single try block out of which, only one catch block will get executed

→ The order of catch blocks should always be from Specific Exception to Generalised Exception

→ If an Exception does not match with a Specific Exception catch block then the Generalised exception catch block will get executed

```
class Sam
{
    public static void main (String args[])
    {
        try
        {
            int a = 50/0;
            int a[] = new int[5];
            a[50]=10;
            String s = null;
            S.o.p(s.length());
            String s = "abc";
            int i = Integer.parseInt(s);
        }
        catch (ArithmeticException e)
        {
            S.o.p(e);
        }
        catch (ArrayIndexOutOfBounds e)
        {
            S.o.p(e);
        }
        catch (NullPointerException e)
        {
            S.o.p(e);
        }
```

```
        catch (NumberFormatException e)
        {
            S.o.p(e);
        }
        catch (Exception e)
        {
            S.o.p(e);
        }
        S.o.p("Rest of the code");
    }
}
```

11/1/20

Nested-try blocks:-

Example program:

```
class Sam
{
    public static void main (String args[])
    {
        try
        {
            try
            {
                int a = 5010;
                S.o.p(a);
            }
            catch (ArithmeticException e)
            {
                S.o.p(e);
            }
            try
            {
                int a[] = new int[5];
                a[50] = 10;
            }
            catch (ArrayIndexOutOfBound e)
            {
                S.o.p(e);
            }
            try
            {
                String s = null;
                S.o.p(s.length());
            }
```

```
        Catch (NullPointerException e)
        {
            S.o.p(e);
        }
        try
        {
            String s = "abc";
            int i = Integer.parseInt(s);
        }
        Catch (NumberFormatException e)
        {
            S.o.p(e);
        }
    }
    Catch (Exception e)
    {
        S.o.p(e);
    }
    S.o.p("Rest of the code");
  }
}
```

throw Keyword:-

Used to throw an Exception Explicitly.

Syntax: throw Exception reference;

(or) (Anonymous object)

throw new ExceptionClass();

Ex:-

```
class Sam
{
    static void validate(int age)
    {
        if (age < 18)
        {
            throw new ArithmeticException("Not eligible");
        }
        else
        {
            S.o.p("Eligible");
```

```
    }
}
public static void main (String args[])
{
    try
    {
        validate(16);
    }
    catch (ArithmeticException e)
    {
        S.o.p(e);
    }
}
```

o/p: java.lang.ArithmeticException : Not Eligible

throws Keywold:-

Used to declare an Exceptions, the Exception should be declared along with method Signature

Case-1:- ~~The method Signature, where we want to declare Exceptions and~~ the method where we are going to call the ~~previous~~ another method which have Exception declaration should also have a throws Keywold.

Case-2:- The method which have throws Keywold & we are trying to call this method in another method should be enclosed in try-block.

Exn Case-1:-

```
impolt java.io.*;
class Sam
{
    static int a;
    static void input() throws IOException
    {
        InputStreamReader i = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(i);
```

```
        a = Integer.parseInt(br.readLine());
        S.o.p(a);
    }
    public static void main(String args[]) throws IOException
    {
        input();
    }
}
```

Ex: Case-2:-

```
impdt java.io.*;
class Sam
{
    static int a;
    static void input() throws IOException
    {
        InputStreamReader i = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(i);
        a = Integer.parseInt(br.readLine());
        S.o.p(a);
    }
    public static void main(String args[])
    {
        try
        {
            input();
        }
        catch(Exception e)
        {
            S.o.p(e);
        }
    }
}
```

Access modifiers:- Access modifiers are used to specify the accessibility and scope of a field (variables), methods, Constructors and even classes

There are 4 Access Specifiers.

1. private
2. default
3. protected
4. public

S.No	Access Specifier / Scope	Within the class	within the package (another class)	Outside package (childclass)	Outside package (another class)
1.	private	Yes	NO	NO	NO
2.	default	Yes	Yes	NO	NO
3.	protected	Yes	Yes	Yes	No
4.	public	Yes	Yes	Yes	Yes

Program to Illustrate Accessing the private members:-

We Cannot access private members outside the class

```
class PvtDemo
{
    private int a=10;
    private void disp()
    {
        S.o.p(a);
    }
}
class Sam
{
    public static void main(String args[])
    {
        PvtDemo P = new PvtDemo();
        P.disp(); // Compilation Error because disp is a
                  private method of private Demo class
    }
}
```

Private Constructor:-

```
class PvtCon
{
 private int a;
 private PvtCon()
 {
    a=10;
    S.o.p(a);
 }
}
class Sam
{
    public static void main(String args[])
    {
        PvtCon p = new PvtCon();
    }
}
```

→ The Constructor Cannot be invoked while creating an Object because it's a private Constructor

→ We cannot create Objects for a class which is having a private Constructor

default Access Specifier:-

```
class Def
{
    int a;
    Def()
    {
      a=10;
      S.o.p(a);
    }
}
class Sam
{ public static void main(String args[])
    {
        Def d = new Def();
    }
}
```

- protected Access Specifier:-

```
package pack1;
class ProtDemo
{ protected int a=10;
  protected void disp()
  { S.o.p(a);
  }
}
package Pack2;
import Pack1.ProtDemo;
class Sam extends ProtDemo
{
  public static void main(String args[])
  {
    new Sam().disp();
  }
}
```

- Public Access Specifier:-

```
package Pack1;
class PubDemo.
{ public int a=10;
  public void disp()
  { S.o.p(a);
  }
}
package Pack2;
import Pack1.PubDemo.
class Sam
{
  public static void main(String args[])
  {
    new PubDemo().disp();
  }
}
```