C

```
void main()
{
    printf("Bitwik Das");
}
```

C++

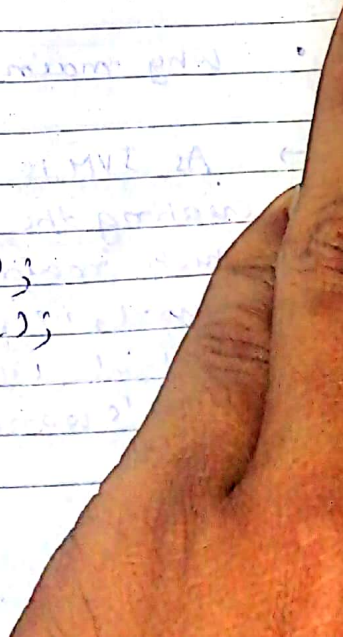
java:-

```
class Test
{
    public static void main (String args[]).
    {
        System.out.println("Bitwik Das");
    }
}
```

println is a method of PrintStream class.

```
class student
{
    int age;
    float mark;
    void initialize()
    {
        age = 5
        mark = 20.5f
    }
}

void show()
{
    System.out.println("Age = " + age);
    System.out.println("Marks = " + mark);
}
}
```



```
class Test
{
    public static void main (String args[])
    {
        Student st = new Student();
        st.initialize();
        st.show();
    }
}
```

- Why the main method is public in java ?

→ During execution of any java program, JVM calls the main method to start execution. As JVM is present outside the class, main method needs to be public to provide access permission to JVM.

If the main method is not public, the source code will be converted to byte code during compilation time without any error but it can't be executed.

- Why main method is static in java ?

→ As JVM is calling the main method before creating the object of the corresponding class to which main method belongs to. So, static keyword is used, i.e., object creation will be done once the control will enter to the main method but JVM's work will start before that. So, main

method in java is declared as static and jvm uses the class name to call the main method directly.

- Why the return type of main method is void in java?

→ To start the execution, JVM calls the main method, but in return it doesn't expect any value from function defⁿ, i.e., defⁿ of the main method. So, the return type is specified as void.

Object Oriented Programming

Four diff features make a language as object oriented programming language.

1. Abstraction
2. Encapsulation
3. Polymorphysm.
4. Inheritance

① Abstraction - It is a process of data hiding where the essential details are hidden to the outside world and the users are given their requirement.

② Encapsulation - The grouping of data members and methods within a single block is called encapsulation.

This also helps in data hiding process.

③. Polymorphism.

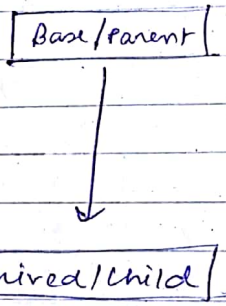
It means many forms, i.e; one object can behave differently at different instant of time known as polymorphism.

Polymorphism can be of 2 types -

- Compiled type polymorphism.
- Runtime polymorphism.

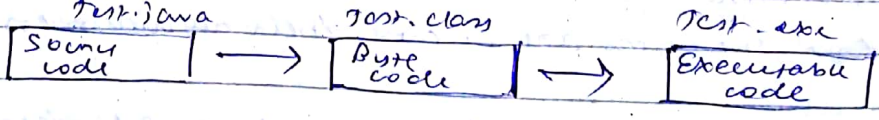
④ Inheritency.

Acquiring the properties from one existing class to a new class is known as inheritance.



⑤. • Features of Java

① . Object oriented.
② . Platform independent.



After successfully compilation, the source code will be converted to its corresponding byte code or known as .class file. The same byte code can be executed in any machine irrespective of operating system or environment. JVM in the destination machine is responsible to read the byte code and execution. So, Java is popularly known as platform independent language as it follows the principle of W.O.R.A (Write once Run Anywhere).

③. Portable :- As java is platform independent, it can be shifted to any machine after compilation.

④. Simple :

Java is very easy to learn and its syntax is simple and easy to understand. The syntax is simple because its syntax is based on C++.

It has removed many confusing features like pointers, preprocessor directive, operator overloading, template, etc. as these are rarely used in programming language.

To deallocate memory, automatic grabage collection feature makes java more simple.

⑤ Robust — Memory management Mechanism.
" — Exception Handling "

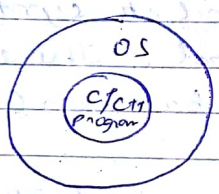
~~Java is considered as strong // robust due to two~~

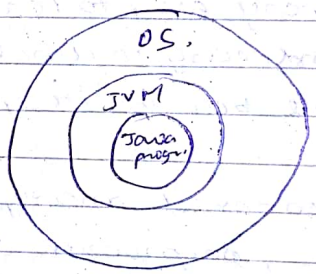
Though memory management mechanism, the language will allocate and deallocate memory automatically as per the requirement of user.

Similarly different exception handling mechanism are followed to check different unhandled errors to provide normal termination to the program.

⑥. Secure :-

Java is best known for its security. It ~~is~~ is secure because no explicit pointer is used and java program run under virtual machine





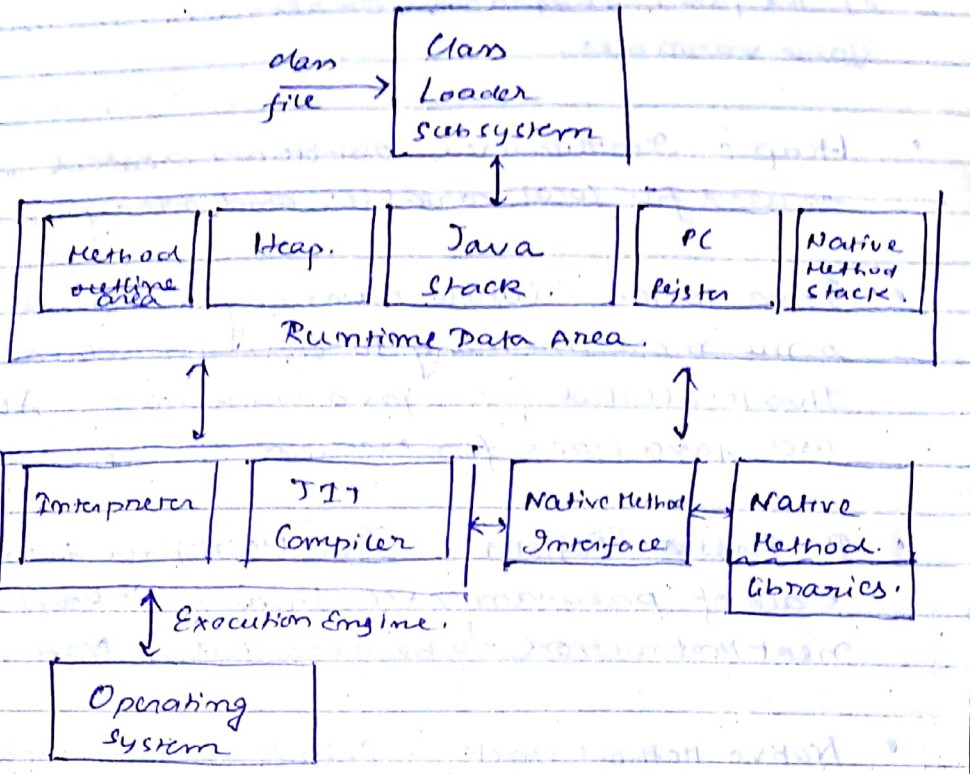
⑦. Multi threaded -

A thread is like a seperate program executing concurrently. Multiple task will run parallely but doesn't occupy diff resources for every thread. It shares a common memory area along with all resources allocated to the single thread. This is important for web application.

JVM is th
It is respon
can conv

• Class loa
It verifies
finds an
rejected
then it a
This me
runtim
runn

JVM (Java Virtual Machine)



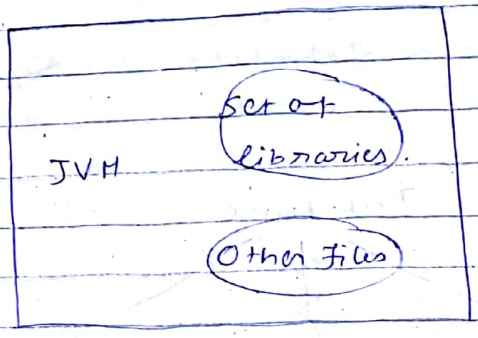
JVM is the heart of entire java program execution process. It is responsible for accepting the .class file or byte code and can convert the instructions into machine language instruction

- Class loader subsystem - it loads the .class file into memory. It verifies all byte code instructions are proper or not. If it finds any instruction as modified, the execution will be rejected immediately. If the byte code instructions are proper, then it allocates necessary memory for execution of the program. This memory is divided into 5 different parts known as runtime data areas, which contains data and results while running the program.

PC - Program Counter

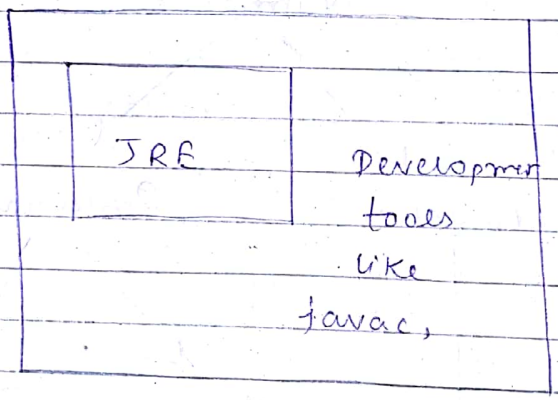
- Method area - This memory block stores the class code, the code of the variable and code of the methods of the java program. It also allocates memory for static variables.
- Heap - In this area, objects are created. This area is used for local variables and array.
- Java stack - While running a method, it needs some more memory to store the data and result. This is allocated from java stack area. Also, JVM uses java stack for execution of different threads.
- PC ~~Rejster~~ Register - This PC register is used to keep track of program execution. It keeps the adress of next instruction to be executed.
- Native Method Stack - Native method include C/C++, which are executed on native method stacks, i.e., the programmes can write java program using C/C++ syntax which is known as native programming. To execute native methods, native method libraries. This header files are located and connected to JVM by a program known as native method interface.
- Interpreter -
- JIT (Just In Time) Compiler -

JRE (Java Runtime Environment).



It is used to provide runtime environment to execute java programs. It is an implementation of JVM including set of libraries and others files that JVM uses at runtime.

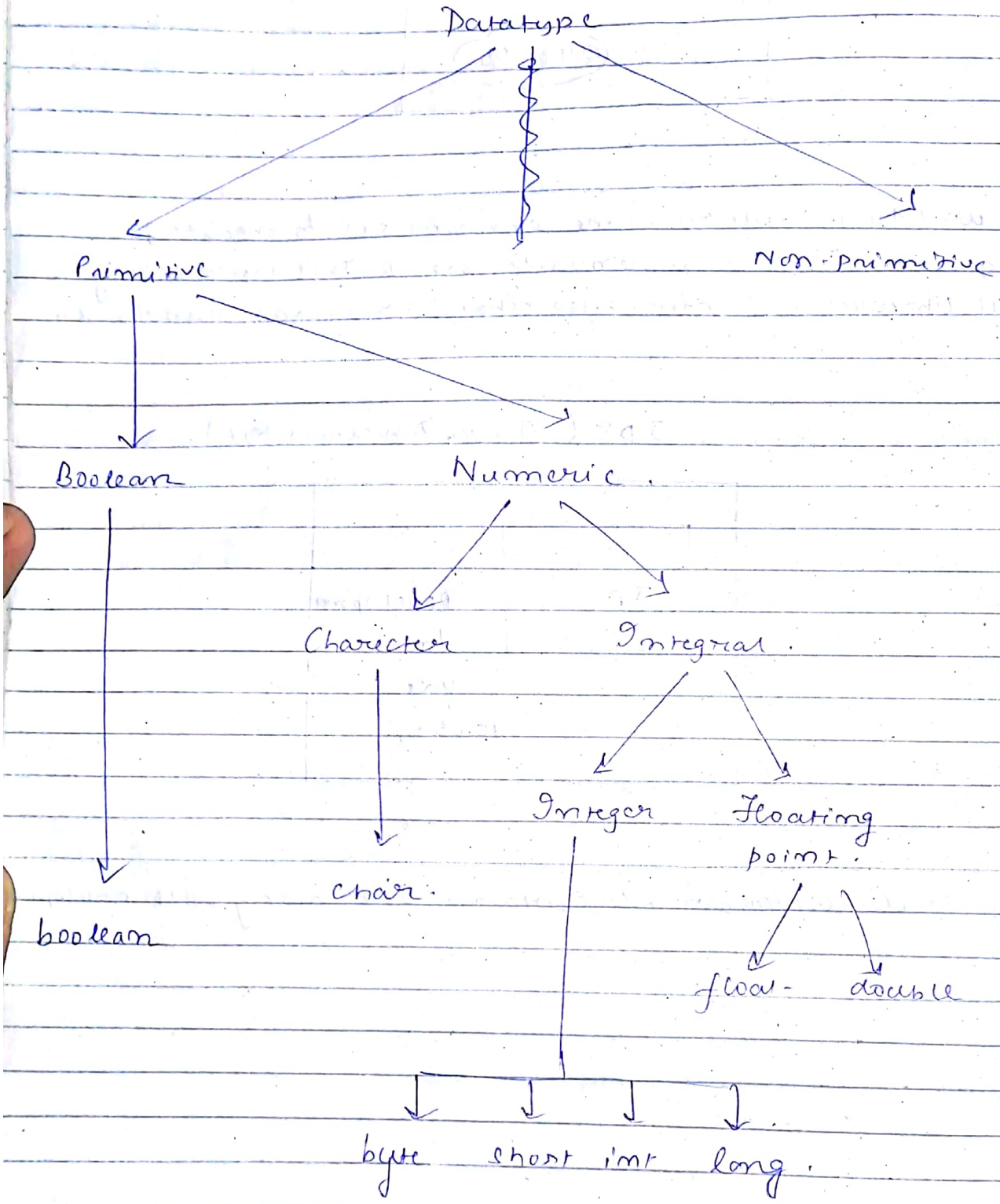
JDK (Java Developer Kit).



The java development kit contains JRE along with developm tools.

Datatype

Datatype ~~~ Default value ~~~ Size.



Datatype	Default value	Size.
boolean	false.	1 bit.
char	'\u0000'	2 Byte
byte	0	1 Byte
short	0	2 Byte.
int	0	4 Byte
long	0L	8 Byte.
float	0.0f.	4 Byte.
double.	0.0	8 Byte.

Q Why the size of character in Java is 2byte?

→ Java support universal code or unicode character, i.e., the character datatype can accomodate for diff. languages, when 65536 diff. types of characters / symbols are present. So, to support all varieties of symbols, in java language, the java soft people have increased the size of character datatype to 2 bytes.

- Float and Double -

float can represent 7 digits accurately after the decimal point. double can represent 15 digits after the decimal point.

If f is not written JVM will allocate 8 bytes by considering it as a double value.

- long - if 'L' is not written to, 1 byte will be allocated to no.1 as 1 byte is sufficient to store.

```
Class test
{
    psvm (String args[]).
    {
        float no = 35.5f.
        int no1 = no; X  int no1=(int)no;✓
        S.o.p(no);
        So p(no1);
    }
}
```

Q. Why there is no sizeof operator in java? → Google it.

Variables - 3 different types of variables available in java. -

1. local.
2. instance
3. static.

Q local variable -

A variable is a container which holds the value while the java program is executed. Every variable is assigned with a datatype. Also, variable is a name of memory location.

① local variable - In java, any variable which is declared inside the body of the method is called the local variable, i.e., this kind of variable can only be used within that method itself and other methods of that class are not aware about the existence of the local variable.

```
Class Test
{
    void func()
    {
        int no = 20;
        System.out.println("Hello" + no);
    }
    public static void main(String args[])
    {
        Test ob = new Test; // Object of class Test.
        ob.func();
        // S.o.p(no); // error as variable not declared.
    }
}
```

Note:
② A local variable can't be defined using static keyword.

② Instance variable -

```
Class Test
{
    int no = 20; // instance variable.
    void func()
    {
        int no
        S.o.pen("Hello" + no);
    }
    psvm(String args[])
    {
        Test ob = new Test;
        ob.func(); S.o.p(no); // error.
    }
}
```

* Sop(ob.no);

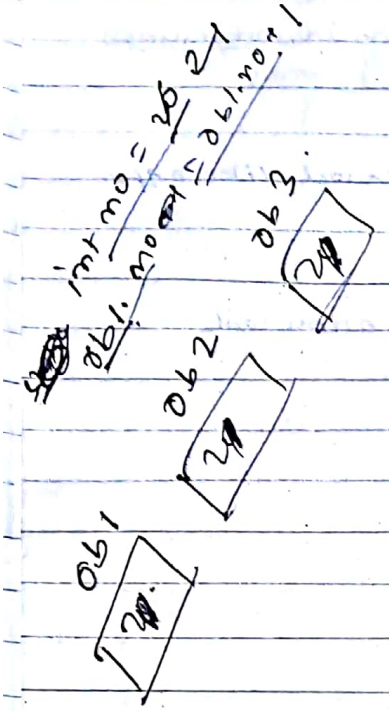
If variable declared ~~the class~~ inside the class but outside the body of the method is called the instance variable.

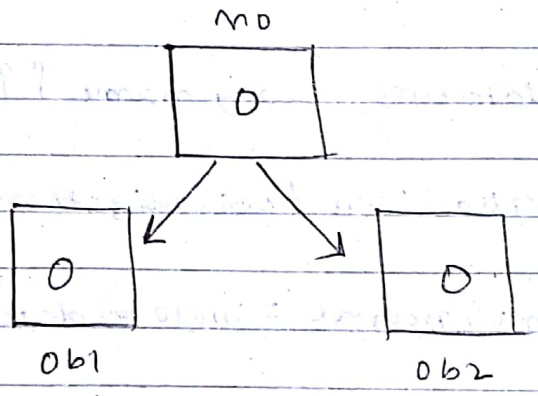
This variable is known as instance variable because its value is instance / object specific and is not shared among diff instances.


```
Class Test
 {
    int no;
    void ~~func~~ show ()
     {
        System.out.println("Hello" +no);
     }
    void input ()
     {
        Scanner sc = new Scanner (System.in);
        S.o.p("Enter value:");
        no = sc.nextInt();
     }
   psvm (String args [])
     {
        Test ob1 = new Test();
        Test ob2 = new Test();
        ob1.input ();
        ob2. ~~show~~ input ();
        ob1. show();
        ob2. show ();
     }
 }
```

① Static variable -

```
Class Test
{
    Static int no;
    void show()
    {
        System.out.println("Hello" + no);
    }
    psvm(String args[])
    {
        Test ob1 = new Test();
        Test ob2 = new Test();
        ob1.show();
        ob2.show();
    }
}
```





A variable which is declared as static is called static variable.

A single copy of the static variable is created and will be shared by all instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

Q. What is the diff between static and instance variable. Explain with example.

Array - is a collection of similar type of objects in java, which gets memory locations in contiguous manner.

- The java array variables can also be declared like other variables.
- The variables in the array present in an ordered manner from index 0. - Why ?

array in java consists of 3 phases. -

① declaration. -

datatype array_name [ ]

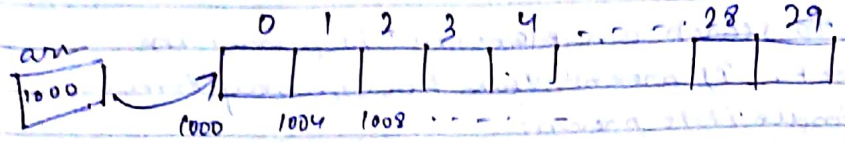
② Memory allocation / ~~initialization~~. instantiation.

array_name = new datatype [size];

③. Initialisation. -

array_name [index] = value.

array name keeps the base adress or starting ~~blocking~~ block's address.




```
Class Test
{
    psvm (String args[])
    {
        int index;
        int arr[];
        arr = new int[30];
        int size;
        Scanner sc = new Scanner (System.in)
        S.o.pen ("Enter size of array");
        size = sc.nextInt();
        for (index = 0; index < size ; index++)
        {
            S.o.pen ("Enter value :");
            arr[index] = sc.nextInt();
        }
        S.o.pen ("values are :");
        for (index = 0; index < size ; index++)
        {
            S.o.pen (arr[index]);
        }
    }
}
```

Q WAP in Java to search an element is present in the array or not. If present then how many times and in which index it is present.

```
class Search
{
public static void main (String args[])
{
int i, arr[], size, no, count=0;
arr = new int[30];
Scanner sc = new Scanner (System.in);
System.out.println("Enter the size!");
size = sc.nextInt();
System.out.println("Enter the no!");
no = sc.nextInt();
System.out.println("Enter values!");
for(i=0; i<size; i++)
    arr[i] = sc.nextInt();
System.out.println("Enter the no!");
no = sc.nextInt(); int index[] = new
for(i=0; i<size; i++)                int[30];
{
    if(no == arr[i])
    {
        index[count] = i;
        count++;
    }
}
if (count == 0)
    System.out.println("No is not
                        found");
else{System.out.println("No is found" +
for(i=0; i<count; i++)                count+
{                                    "no of him");
    System.out.println(index[i]);
}
```

Multi-Dimensional array -

int arr [][] = new int [10][10];

Class and Object:

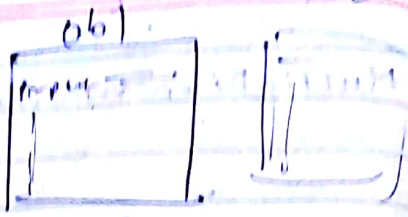
Object :- An object is an instance of a class, i.e., it is an entity that has state and behaviour and also identity. It can be physical or logical.

(i) State - It represents the data or value of an object.
(ii) Behaviour - It represents the behaviours or functionality present inside the class.
(iii) Identity - Every object is implemented through unique id. The unique id is not visible through the external uses but internally it is ~~intern~~ implemented through JVM.

Q Define an employee class with different data members and display the data of two different employees.

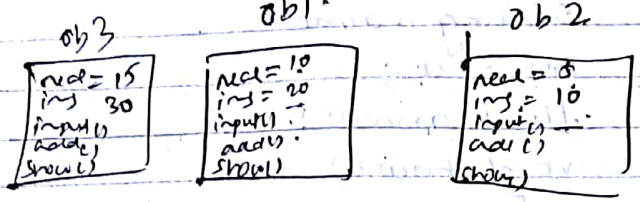
~~class~~

ob2




```
class Employee
{
    int empid;
    String name;
    float salary;
    void input()
    {
        System.out.println("Enter empid : ");
        empid = sc.nextInt();
        System.out.println("Enter name : ");
        name = sc.next();
        System.out.println("Enter salary : ");
        salary = sc.nextFloat();
    }
    void show()
    {
        System.out.println("Empid : " + empid);
        System.out.println("Name : " + name);
        System.out.println("Salary : " + salary);
    }
}
class Test
{
    public static void main(String args[])
    {
        Employee ob1 = new Employee();
        ob1.input();
        ob1.show();
        Employee ob2 = new Employee();
        ob2.input();
        ob2.show();
        ob1.hashCode(); // displays the unique id of
        ob2.hashCode();   the object.
    }
}
```

Q. Define a complex class. Declare two data members real and img. Define one method to take input. Define another method to find the addition of 2 complex nos. Define another method to show the result.

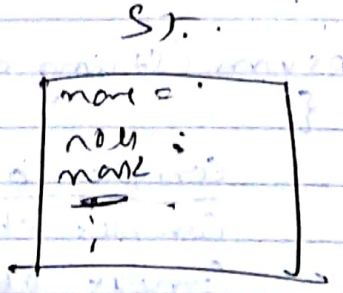



```
class Complex
{
    int real;
    int img;
    void input()
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter real part:");
        real = sc.nextInt();
        System.out.println("Enter imaginary part:");
        img = sc.nextInt();
    }
    void add(Complex ob1, Complex ob2)
    {
        real = ob1.real + ob2.real;
        img = ob1.img + ob2.img;
    }
    void show()
    {
        System.out.println(real + "i" + img);
    }
}
class Test
{
    psvm(String args[])
    {
        Complex ob1 = new Complex();
        Complex ob2 = new Complex();
        Complex ob3 = new Complex();
        ob1.input();
        ob2.input();
        ob3.add(ob1, ob2);
        ob3.show();
    }
}
```

How to initialize the variables of a class :-

```
class Student
 {
    String name;
    int roll;
    float mark;
    void show()
      {
        System.out.println("Name :" + name);
        System.out.println("Roll :" + roll);
        System.out.println("Mark :" + mark);
      }
    void initialize (String name1, int roll1, float mark1)
      {
        name = name1;
        roll = roll1;
        mark = mark1;
      }
 }

class Test // 1st idea.
 {
    public static void main (String args[])
      {
        Student st = new Student();
        st.initialize ("Shree", 31, 25.6f);
        st.show();
      }
 }
```



*

```
class Test   // 2nd idea
{
    public static void main (String args[])
    {
        Student st = new Student();
        String name2;
        int roll2;
        float mark2;
        Scanner sc = new Scanner (System.in);
        System.out.println("Enter name:");
        name2 = sc.next();
        System.out.println("Enter roll:");
        roll2 = sc.nextInt();
        System.out.println("Enter mark:");
        mark2= sc.nextFloat();
        st.initialize(name2, roll2, mark2);
        st.show();
    }
}
```

Q. WAP to define a class Time having data members hour, minute, second. Use initialize method to initialize the data members and show method to display the results. Use add method to find the addition of two times. Display the result in proper format.

```
class Time
{
    int hour;
    int minute;
    int second;
    void initialize (int h, int m, int s)
    {
        hour = h;
        minute = m;
        second = s;
    }
```

```
class Test
{
    public static void main (String args[])
    {
        Time t = new Time ();
        int hour, hr1;
        int min, min1;
        int sec, sec1;
        Scanner sc = new Scanner (System.in);
        System.out.println("Enter hour, min, second:");
        hour = sc.nextInt();
        min = sc.nextInt();
        sec = sc.nextInt();
        t.initialize (hr, min, sec);
        System.out.println("Enter time 2:");
        hr1 = sc.nextInt();
        min1 = sc.nextInt();
        sec1 = sc.nextInt();
        Time t1 = new Time();
        t1.initialize (hr1, min1, sec1);
        Time t2 = new Time();
        t2.add (
```

Q Define a class Number with data member num (int), input method will accept the value and display the result. Define a large method which will find largest among 2 nos.

```
class Number
 {
    int num, large;
    void input()
     { Scanner sc = new Scanner (System.in);
       System.out.println("Enter number:");
       num = sc.nextInt();
     }
    void large (Number n1, Number n2)
     {
        if (n1.num > n2.num)
        numlarge = n1.num;
        else
        numlarge = n2.num;
     }
    void show()
     {
        System.out.println("The largest:" + numlarge);
     }
 }
class Test
 {
    Number n1 = new Number();
    Number n2 = new Number();
    n1.input();
    n2.input();
    Number n3 = new Number();
    n3.large(n1, n2);
    n3.show();
 }
```

Method Overloading :-

It is a mechanism of giving multiple methods with the same name but all methods are different w.r.t number of arguments or type of arguments

```
class Overloading
{
    void add (int no1, int no2)
    {
        int res;
        res = no1 + no2;
        System.out.println ("Sum = " + res);
    }
    void add (int no1, int no2, int no3)
    {
        int res;
        res = no1 + no2 + no3;
        System.out.println ("Sum = " + res);
    }
    void add (float no1, float no2)
    {
        float res;
        res = no1 + no2 + no3;
        System.out.println ("Sum = " + res);
    }
}
class Test
{
    Overloading obj = new Overloading ();
    obj.add (2, 3);
    obj.add (5, 6, 7);
    obj.add (25.6f, 32.5f);
}
```

The linking will be established during the compilation time because due to the method signature it can be differentiated which method body will execute during the method call.

Constructor

```
class ExampleConstructor
{
    int no1, no2;
    ExampleConstructor ()
    {
        no1 = 10;
        no2 = 20;
    }
    void show()
    {
        System.out.println ("Numbers: " + no1 + no2);
    }
}

class Test
{
    psvm (String args[])
    {
        ExampleConstructor ob = new ExampleConstructor ();
        ob.show();
    }
}
```

- Constructor in java is a special type of method that is used to initialize the object.
- It is called automatically during the time of object creation. At it constructs the values, i.e.,

provides data for objects, so it is known as constructor.

- Constructor name must be same as class name.
- Constructor doesn't have any return type, neither void.

grrp

Q. What is the difference between constructor and method in Java?

Types of constructors :-

2 different types of constructors are there present in Java :-

- Default Constructor / Zero argument Constructor.
- Parameterized constructor / constructor with arguments

Default Constructor :-

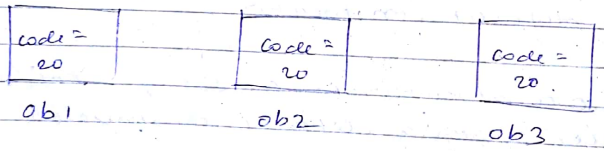
A constructor which has no argument / parameter is known as default constructor

If there is no constructor in a class, JVM will automatically creates a default constructor. It is used to assign same value to all objects.

~~Parameters~~

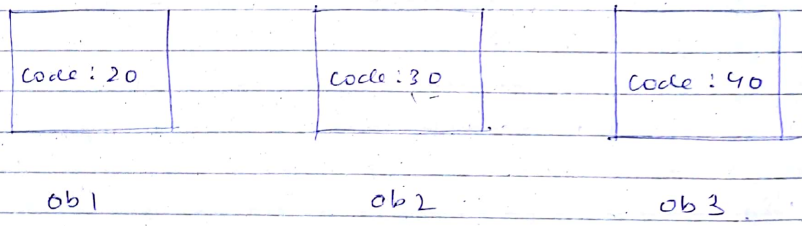
```
class Example
{
    int code;
    Example()
    {
        code = 20;
    }
    void show()
    {
        System.out.println("code: " + code);
    }
}

class Test
{ psvm(String args[])
  { Example ob1 = new Example();
    Example ob2 = new Example();
    Example ob3 = new Example();
    ob1.show();
    ob2.show();
  } ob3.show();
}
```



Parameterized constructor :-

```
class Example
{
    int code;
    Example()
    {
        code = 20;
    }
    Example(int value)
    {
        code = value;
    }
    void show()
    {
        System.out.println("Code : "+code);
    }
}
```



```
class Test
{
    psvm ( String args[] )
    {
        Example ob1 = new Example (10);
        Example ob2 = new Example (30);
        Example ob3 = new Example (40);
        ob1.show();
        ob2.show();
        ob3.show();
    }
}
```

```
class Complex
{
    int real;
    int img;
    void input()
    {
        Scanner
    Complex(int r, int i)
    {
        real = r;
        img = i;
    }
    void add(Complex ob1, Complex ob2)
    {
        real = ob1.read + ob2.real;
        img = ob1.img + ob2.img;
    }
    void show()
    {
        System.out.println(real + " i " + img);
    }
}
class Test
{
    psvm(String args[])
    { int img1, real1, img2, real2;
      Complex ob Scanner sc = new Scanner(System.in)
      S.o.pln("Enter img and real parts : ");
      img1 = sc.nextInt();
      real1 = sc.nextInt();
      Complex ob1 = new Complex(img1, real1);
      S.o.pln("Enter img and real parts : ");
      img2 = sc.nextInt();
      real2 = sc.nextInt();
      Complex ob2 = new Complex(img2, real2);
      Complex ob3 = new Complex();
      ob3.add(ob1, ob2);
    } ob3.show();
}
```

2. class Number
{
int num;
Number(int n)
{
num = n;
}
void large(Number n1, Number n2)
{
if (n1.num > n2.num)
num = n1.num;
else if (n2.num > n1.num)
num = n2.num;
}
void show()
{
S.o.pln("The largest is :" + num);
}
}
class Test
{
int num1, num2;
Scanner sc = new Scanner(System.in);
S.o.pln("Enter a number = ");
num1 = sc.nextInt();
Number n1 = new Number(num1);
S.o.pln("Enter a number = ");
num2 = sc.nextInt();
Number n2 = new Number(num2);
Number n3 = new Number();
n3.large(n1, n2);
n3.show();
}

Constructor	Method.
• Constructor is used to initialize the state of an object, i.e., it initializes the variables of an object.	• Method is used to describe the behaviour of an object, i.e. it is used for specifying the operations of the class.
• It must not have any return type.	• It must have return type.
• Constructor is called automatically when the object is created.	• Method is explicitly called by user.
• The java compiler provides a default constructor if the user does not have any constructor.	• Method is not provided automatically by the compiler in any case.
• Constructor name must be same as the class name.	•
• Syntax :	
• Example.	

Q. Difference between default constructor and parameterized constructor.

Java Copy Constructor :-

```
class Student
{
    String name;
    int roll;
    float mark;
    Student ()
    {
        name = "Shrei";
        roll = 31;
        mark = 82.5;
    }
    Student ( Student obj )
    {
        name = obj.name;
        roll = obj.roll;
        mark = obj.mark;
    }
    void show ()
    {
        S.o.pln ("Name: " + name );
        S.o.pln ("Roll: " + roll );
        S.o.pln ("Mark: " + mark );
    }
}
class Test
{
    psvm ( String args[] )
    {
        Student obj = new Student ();
        Student ob2 = new Student (obj);
        obj.show ();
        ob2.show ();
    }
}
```

~~Constructor overloading~~

~~By a~~ While creating the object, we were assigning the contents of an existing object through a constructor. It is known as copy constructor.

Constructor overloading :-

```
class Student
{
    String name;
    int roll;
    float mark;
    Student()
    {
        name = "three";
        roll = 31;
        mark = 82.5;
    }
    Student (Student obj)
    {
        name = obj.name;
        roll = obj.roll;
        mark = obj.mark;
    }
    Student (String name1, int roll1, float mark1)
    {
        name = name1;
        roll = roll1;
        mark = mark1;
    }
    Student (String name2)
    {
        name = name1;
    }
}
```

```
    void show()
    {
        Sopln("Name: "+name);
        Sopln("Roll: "+roll);
        Sopln("Mark: "+mark);
    }
}
class Test
{
    psvm(String args[])
    {
        Student ob1 = new Student();
        Student ob2 = new Student(ob1);
        Student ob3 = new Student("Ritwik", 12,
                                  98.6f);
        Student ob4 = new Student("abc");
        ob1.show();
        ob2.show();
        ob3.show();
        ob4.show();
    }
}
```

- Multiple constructors with same name but different with respect to number of arguments or type of the arguments is known as Constructor overloading
- The compiler differentiates these constructor by taking into account the no. of parameters in the list and their types

ob.func();
Abc.func();

Java static keyword :-

The static keyword in java is used for memory management. This can be applied at 4 different situations :-

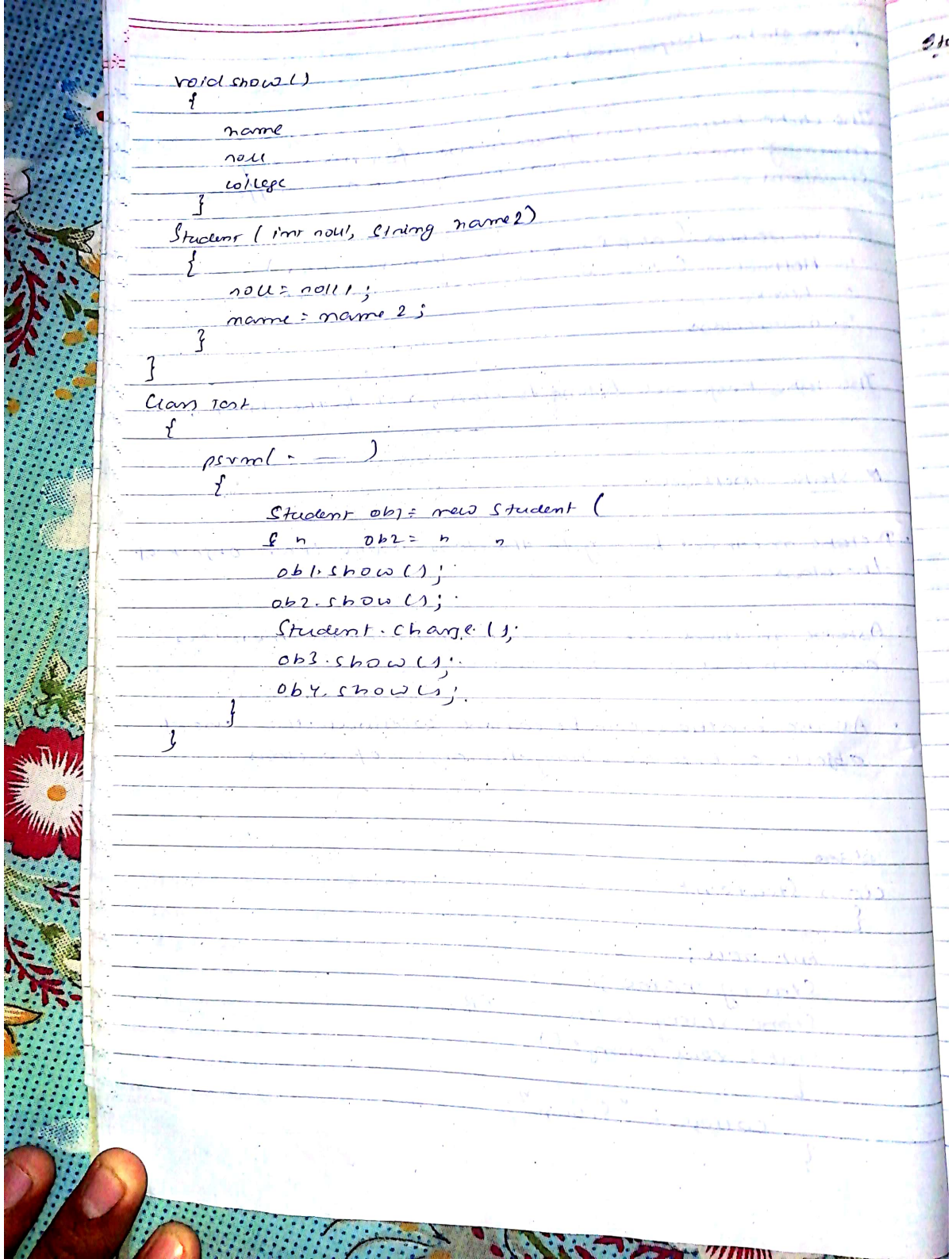
- variables (also known as class variables)
- Method (" " " method)
- block
- nested class

The static keyword belong to class, not to the object.

* Static method :-

- A static method belong to the class rather than object of the class.
- A static method can access only static data members and can call only static methods.
- A static method can be called without the use of object or before creating the object of a class.

```
class Student
 {
    int roll;
    String name;
    static String college = "FBS";
    static void change()
     {
        college = "Silicon";
     }
```



```
void show()
 {
    name
    roll
    college
 }
Student ( int roll, String name2)
  {
    roll = roll1;
    name = name2;
  }
}
Class Test
 {
   psvm( : — )
    {
      Student ob1 = new Student (
      f n    ob2 = n   n
      ob1.show();
      ob2.show();
      Student.change();
      ob3.show();
      ob4.show();
    }
}
```

Static block :-

```
class Test
{
    Test()
    {
        S.o.println("Hello");
    }
}
class Demo
{
    psvm(String args[])
    {
        Test ob = new Test();
    }
}
```

Output : Hello.

```
static
{
    System.out.println("Static Block");
}
```

Output : Static Block
Hello.

It is used to initialize the static data member. It is executed before main method at the time of class loading.

```
class Test
{
    psvm (String args[])
    {   Test ob = new Test();
        System.out.println("Inside main");
    }
    Test()
    {
        System.out.println("Hello");
    }
    static
    {
        System.out.println("Static block");
    }
}
```

Output :- Static block
Hello
Inside main.

```
static
{
    S.o.pen("Static block");
    System.exit(0)
}
```

Output :- Static block.

Q Can you execute a program without main method?

```
class Test
{
    static
    {
        System.out.println("Static block");
        System.exit(0);
    }
}
```

```
class Test
{
    static
    {
        S.o.p("First static");
    }
    Test()
    {
        S.o.p("Default");
    }
    static
    {
        S.o.p("Static block");
    }
    psvm
    {
        S.o.p("Inside main");
        Test ob = new Test();
        S.o.p("Outside main");
    }
    static
    {
        Sop("last static");
    }
}
```

O/P :-

```
First static
Static block
last static
Inside main
Default
Outside main.
```

this keyword :-

```
class Test
{
    int no;
    Test ();
    { }
    Test (int no)
    {
        this.no = no;   // refers to currently created
    }                      object.
    void show()
    {
        Sop(no);
    }
    psvm (String args[])
    {
        Test ob1 = new Test();
        Test ob2 = new Test (20);
        ob1.show();
        ob2.show();
    }
}
```

Output :- 0
20.

This is a keyword which points to the currently created or invoked object.

```
class Student
{
    String name, subject;
    int roll;
    float fee;
    Student(String name, String subject, int roll)
    {
        this.name = name;
        this.subject = subject;
        this.roll = roll;
    }
    Student(String name, String subject, int roll,
                                        float fee)
    {
        this(name, subject, roll);
        this.fee = fee;
    }
    void show()
    {

    }
}
class Demo
{
    psvm(String args[])
    {
        Student obj = new Student("Shree",
                                  "CSE", 31);
        Student ob2 = new Student("Sahil",
                        "EEE", 41, 6000.58f);
        ob1.show();
        ob2.show();
    }
}
```

Package.

Package - It's a group of similar types of classes, interfaces and sub packages.

In JAVA, package is categorized into 2 diff forms -

① Built-in package.
② User defined package.

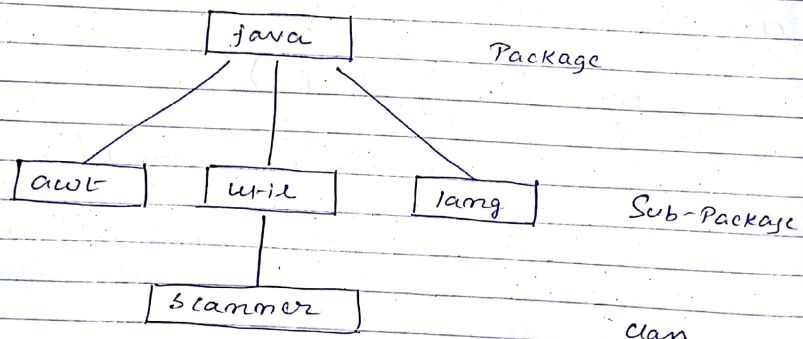
- Advantages of package :-

① It is used to categorized the classes and interfaces so that they can be easily maintained.

② Java package provides access permission.

③ It removes naming collision.

BUILT IN PACKAGE :-



The built-in package consists of predefined classes with their meaning. The content cannot changed but can be used inside the program as per the requirement of the user.

USER DEFINED PACKAGE :-

```
package mypath; // Create a new package with name
                   mypath
class Test
{
    public static void main (String args[]).
    {
        System.out.println("Welcome to package");
    }
}
```

Compile : javac <space> -d <space> . <space> java-file.java

Ex : javac -d . Test.java.

Run : java <space> packagename.classname.

'-d' is a switch that tells the compiler where to put the class file, i.e., it represents destination and '.' represents the current directory.

Q How to access package from another package ?

→ There are 3 different ways to access from outside package

① import <space> packagename.*;
② import <space> packagename.classname;

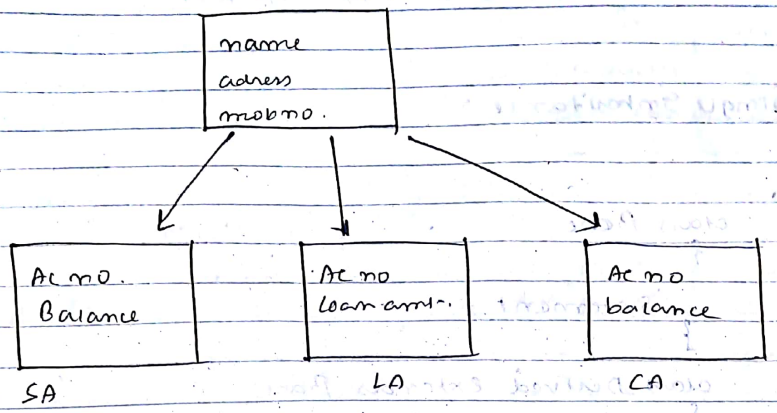
⑨ fully qualified name.

Access Modifier :-

There are four different types of access modifiers present in JAVA which specifies accessibility (scope) of a data member, method, constructor or class.

- Private access modifier - is accessible only within the class.
- Default access modifier - If you do not use any modifier it is treated as default. Default modifier is accessible only within package.
- Protected access modifier - The protected access modifier is accessible within package and outside the package but through inheritance only.
- Protected access modifier can be applied on the data member, from a method or constructor but can't be applied to class.
- Public access modifier - is accessible everywhere, i.e., inside the package as well as outside the package. It has the widest scope among all modifiers.

Inheritance



Inheritance is a mechanism in which one object acquires the properties and behaviour of another existing object. The idea behind inheritance is that the user can create new classes that are built upon existing classes, i.e., when the user inherits from an existing class, the methods and data members can be automatically transferd to the new class. Also the user can add new methods and data members in the new class.

Advantages :-

1. Code Reusability.
2. Method Overriding. (Runtime polymorphism can be achieved).

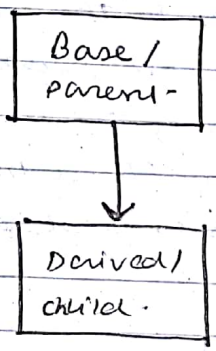
Types of inheritance :-

In JAVA, there are 3 diff types of inheritance mechanism are used -

① Single inheritance.
② Multi-level inheritance.
③. Hierarchial inheritance.

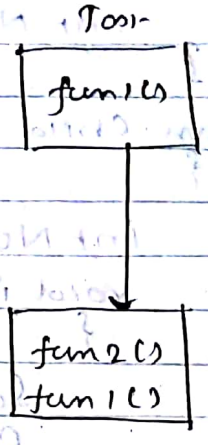
Single Inheritance :-

```
class Base
{
    statement;
}
class Derived extends Base
{
    statement;
}
```



The type of Inheritance where a single child class will acquire the properties of only one parent class, is known as single Inheritance.

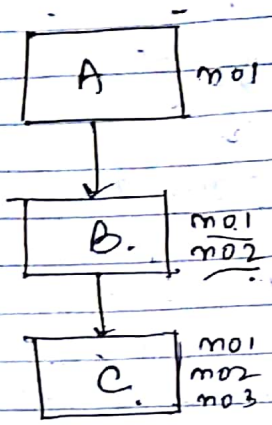
```
class Test
{
    void fun1()
    {
        S.o.p("Base class");
    }
}
class Demo extends Test
{
    void fun2()
    {
        S.o.p("Child class");
    }
}
class Example
{
    psvm(String args[])
    {
        Demo ob = new Demo();
        {
            ob.fun1();
            ob.fun2();
        }
    }
}
```



Q WAP to define a base class having data member Num1 "@" child " " " Num2. and methods input, large and display find the largest among two nos using ~~inner~~ inheritence.

```
class Base
{
    int Num1;
}
class Child extends Base
{
    int Num2, large;
    void input()
    {
        Scanner sc = new Scanner(System.in);
        S.open("Enter two numbers:");
        Num1 = sc.nextInt();
        Num2 = sc.nextInt();
    }
    void large()
    {
        if (Num1 > Num2)
            large = Num1;
        else if (Num < Num2)
            large = Num2;
        else
            S.o.p("Both the nos are same");
    }
    void display()
    { if(Num1 != Num2)
        S.o.pln("largest no. is " + large);
    }
}
class Test
{
    psvm(---)
    {
        Child Ob = new Child();
        Ob.input();
        Ob.large();
    }   Ob.display();
}
```

Multilevel Inheritance:-



The type of inheritance where one can ~~derive~~ inherit from a derived class thereby making this derived class the base class for the new class, i.e., one child class of one parent-class can behave like parent for another child class

```
class Base1
 {
    void fun1()
     {
        S.o.p("Base1");
     }
 }
class Base2
 {
    void fun2()
     {
        Sop("Base2");
     }
 }
class ~~Base3~~ Derived
 {
    void fun3()
     {
        S.o.p("~~Base3~~ Derived");
     }
 }
```

```
class Example
{
    psvm (St. . . )
    {
        Derived ob = new Derived;
        ob.fun2();
        ob.fun3();
        ob.fun1();
    }
}
```

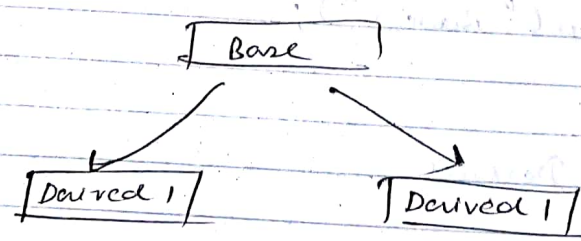
Q. 3 classes - Base 1, Base 2, Derived.

Base 1	Base 2	Derived
Num 1	Num 2	Num 3.
Input1()	Input2()	Input3()

Define smallest method in Derived class to find smallest among 3 nos using conditional operator. Define a display method to print the smallest value.

Hierarchical Inheritance :-

The type of inheritance in which a single base class can contain multiple child classes, is known as hierarchical inheritance.



Method Overriding :-

```
class Base
{
    void show()
    {
        S.o.p("Base class");
    }
}
class Derived
{
    void show()
    {
        S.o.p("Derived class");
    }
}
class Test
{
    perm(String args[])
    {
        Derived ob = new Derived();
        ob.show();
        ob.show();
    }
}
```

Base / Parent - Super
Child / Derived - Sub.

Output :-

Derived class Derived class.

If subclass / child class has the same method as declared in the parent class, it's known as method overriding; i.e., if a child class provides the specific implementation of the method that has been declared by one of its parent class is known as method overriding.

Rules for method overriding :-

- The method in the child class must have same name as the child class.
- The method must have the same parameter as the parent class.
- There must be an IS-A relationship, i.e., inheritance between the classes.

Due to inheritance, when the object of child class is created, all the methods of base class will come to child class. So, if any method is having same name as in the derived class is present in the base class will also get memory inside the child class object. So, while calling the method any no. of times using child class object, the derived class method gets executed and the base class method gets overridden by the derived class method.

Q. Find the difference between over method overloading and overriding.

class Base
{
Base
{
}
}
class Der
{
Deri
{
}
}
class Test
{
psv
{
}

Output :-

Dera

Inheritance and Constructors.

```
class Base
{
    Base()
    {
        S.o.p("Default of Base");
    }
}
class Derived extends Base
{
    Derived()
    {
        S.o.p("Default of Derived");
    }
}
class Test
{
    psvm(String args[])
    {
        Derived ob = new Derived();
    }
}
```

Output :-

Default of Base Default of Derived.

Inheritance and Parameterized Constructor

```
class Base
{
    int no1;
/*  Base()
    {
        S.o.p("Derived Default of Base");
    } */
    Base(int value)
    {
        no1 = value;
    }
}

class Derived extends Base
{
    int no2;
/*  Derived()
    {
        S.o.p("Default of derived");
    } */
    Derived(int value)
    {
        no2 = value;
    }
    void show()
    {
        S.o.p("no1" " " + no2);
    }
}

class Test
{
    psvm(String args[])
    {
        Derived ob = new Derived(20);
        ob.show();
    }
}
```

Output :-

0 20.

When object of Derived class is created with argument, the derived class parameterized constructor will be called, its base class parameterized constructor will not be called automatically like the default constructor of the base class.

final.

final is a keyword in JAVA and is applicable for variable, method and class.

final variable:

```
~~class Test~~
class Test
{
    psvm(String args[])
    {
        final int no = 5;
        no++;
        S.o.p(no);        → error.
        no = 10;
        S.o.p(no);
    }
}
```

Any variable which is declared with final keyword, its value remain constant throughout the program.
Modification ~~through~~ to the content of variable shows error to the user., i.e., the content of the final variable cant be modified.

Final method:-

```
class Base
{
    final void show()
    {
        S.o.p ("Base");
    }
}
class Derived
{
    void show() // Error.
    {
        S.o.p("Derived");
    }
}
```

If any base class method is declared with final keyword, no method with the same name can be present in the child class, i.e., method overriding is not allowed using final method.

Final class :-

```
final class Base
{
    void show()
    {
        S.o.p("Base");
    }
}
```

```
class Derived extends Base // ones
{
    void show2()
    {
        S.o.p ("Derived");
    }
}
```

Any class which is declared with final keyword cannot be inherited further, i.e., final class doesn't have any child class.

Super keyword:

- The super keyword is a reference variable which is used to refer immediate parent class object, i.e., whenever the user creates the instance of child class, an instance of parent class is created automatically which is refered by super.
- Super can be used for refering immediate parent class instance variable.
- Super can be used to call the parent class method.
- Super can be used to call the immediate parent class constructor.

Example-1

```
class Test
{
    psvm (String args[]).
    {
        Derived ob = new Derived ();
        ob.show();
    }
}
class Base
{
    int no = 5;
}
class Derived extends Base
{
    int no = 10;
    void show()
    {
        S.o.p("no : " + no + super.no);
    }
}
```

Output : 10 5

We can use super key word to access the data member of the parent class. Most importantly this is used when parent and child classes have same variable name.

Example-2

```
class Base
 {
    void show()
     {
        S.o.p("Base");
     }
 }
void Derived extends Base
 {
    void show()
     {
        super.show();
        S.o.p("Derived");
     }
 }
class Test
 {
    psvm(String args[])
     {
        Derived ob = new Derived();
        ob.show();
     }
 }
```

Output :- Base Derived.

Example 3 :-

```
class Base
 {
    Base()
     {
        S.o.p("Default of Base");
     }
}
```

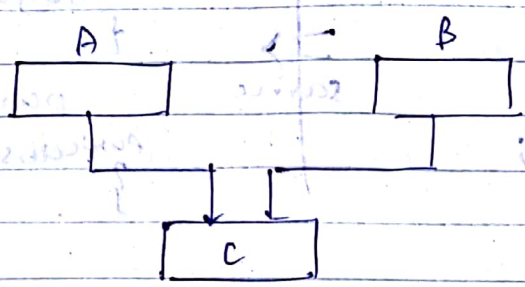
```
class Derived extends Base
{
    Derived()
    {
        super();
        S.o.p("Derived");
    }
}
```

Q. How to call base class parameterized constructor?

```
class Base
{
    int no1;
    Base(int value)
    {
        no1 = value;
    }
}
class Derived extends Base
{
    int no2;
    Derived(int value1, int value2)
    {
        super(value1);
        no2 = value2;
    }
    void show()
    {
        S.o.p(no1+" "+no2);
    }
}
```

```
class Test
{
    psvm (String args[])
    {
        Derived ob = new Derived (5, 10);
        ob.show();
    }
}
```

Multiple Inheritance.



- In OOP, if a single child class inherit the properties of more than one base class, it is known as multiple inheritance.
- Multiple Inheritance is not available in JAVA.

Interface :-

- An interface in JAVA is a blue print of the class, which is required to achieve multiple inheritance in JAVA.
- It is a mechanism to achieve abstraction

There can be only abstract methods and public static final variables in the body of the interface.

```
class A
{
    abstract void show();
}
```

eg:-

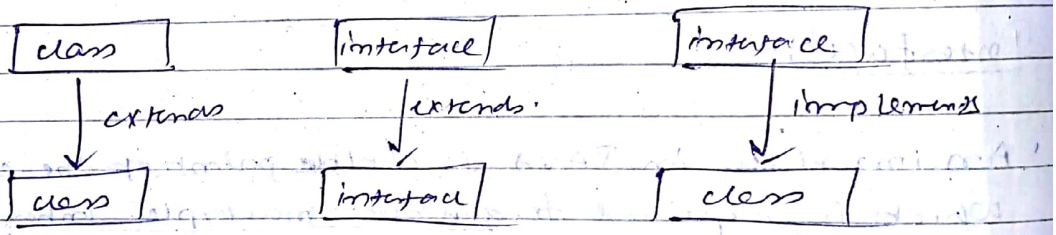
```
interface Test
{
    int no;
    void show();
}
```

same

```
interface Test
{
    public static final int no;
    public abstract void show();
}
```

The JAVA compiler adds public and abstract keywords before the interface methods and also adds public, static and final keywords before the data members.

- Relationship between class and interface :-



Q Can an interface inherit the properties of a class?

, ~~Advantage~~ As class can contain both abstract and non abstract methods as well as data members with public, static and final and also without using this, so as per inheritance if interface inherits the properties of class, then the rule of interface will change, i.e., interface has to allocate all kind of variables as well as all methods. So, to keep the existance of interface ~~from~~, it can't inherit properties of another class.

Eg :-

```
interface Test
{
    public & abstract void display();
    void show();
}
class Demo implements Test
{
    pubic void show()
    {
        S.o.p("Hello");
    }
    pubic void display()
    {
        S.o.p("Hi");
    }
}
```

```
class Example
{
    psvm(String args[])
    {
        Demo ob = new Demo();
        ob.display();
        ob.show();
    }
}
```

Ex:-

```
interface shape
{
    float pi = 3.14f
    void input();
    void area();
    void show();
}
class Circle implements shape
{
    float radius, result;
public void input()
    {
        Scanner sc = new Scanner(System.in);
        S.o.println("Enter radius: ");
        radius = sc.nextFloat();
    }
public void area():
    {
        result = pi * radius * radius;
    }
public void show()
    {
        S.o.println("Area of circle: " + result);
    }
}
```

```
class Triangle implements Shape
{
    double result, a, b, c, s;
public void input()
    {
        Scanner sc = new Scanner(System.in);
        S.o.println("Enter 3 sides of a triangle:");
        a = sc.nextDouble();
        b = sc.nextDouble();
        c = sc.nextDouble();
    }
public void area()
    {
        s = (a+b+c)/2;
        result = Math.sqrt(s*(s-a)*(s-b)*(s-c));
    }
public void show()
    {
        S.o.println("Area of triangle:" + result);
    }
}
class Rectangle implements Shape
{
    float result, l, b;
public void input()
    {
        Scanner sc = new Scanner(System.in);
        S.o.println("Enter length and breadth:");
        l = sc.nextFloat();
        b = sc.nextFloat();
    }
public void area()
    {
        result = l*b;
    }
public void show()
    {
        S.o.println("Area of rectangle:" + result);
    }
}
```

```
Class Test-
{   psvm (String args[])
    {Shape ob;
    ob = new Circle();
    ob.input();
    ob.area();
    ob.show();
    ob = new Triangle();
     ob.input();
    ob.area(); ob.show();
    ob = new Rectangle();
     ob.input();
     ob.area();
     ob.show();
}       }
```

Ex:-

```
Interface First-
  {
      void A;
      void B;
      void C;
      void D;
  }
```

Polymorphism

Polymorphism in JAVA is a concept in which we can perform a single action by different ways.

There are 2 diff types of polymorphisms ~~are~~ present in JAVA -

① Compile time Polymorphism / Static binding / Early binding
② Runtime " / Dynamic " / Late "

Compile time polymorphism can be achieved through method overloading but runtime polymorphism can be achieved through method overriding or dynamic method dispatch.

```
class Base
{
    void show()
    {
        S.o.p("Base");
    }
}

class Derived extends Base.
{
    void show()
    {
        S.o.p("Derived");
    }
}
```

```
class Test
{
    psvm(String args[])
    {
        Base ob;
        ob = new Base();   ← call to base class
        ob.show();
        ob = new Derived();   ← upcasting / dynamic method dispatch
        ob.show();
    }
}
```

call to derived class.

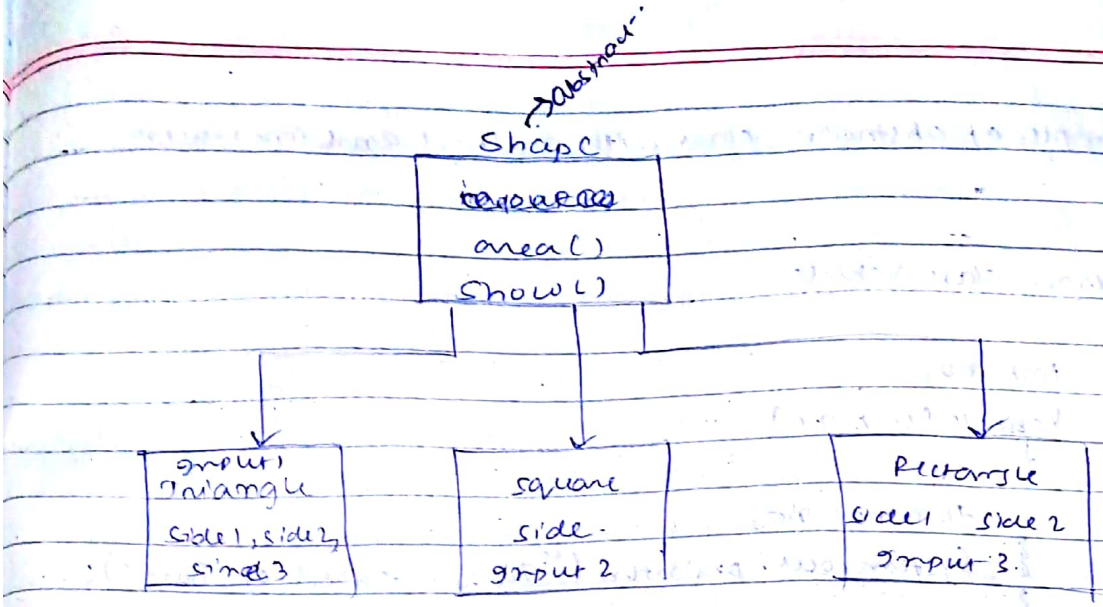
```
class Derived1 extends Derived
{
    void show()
    {
        S.o.p("Derived1");
    }
}

class Test-
{
    psvm (String args[])
    {
        Base ob;
        ob = new Derived();
        ob.show();
        ob = new Base();
        ob.show();
        ob = new Derived1();
        ob.show();
    }
}
```

ABSTRACT CLASS :-

A class that is declared ~~with~~ with abstract keyword is known as abstract class in JAVA. One abstract class can contain abstract and non abstract methods within the class.

This is required to achieve abstraction.




```
class Test
{
    psvm( ---- )
    {
        Shape ob;
        ob = new Triangle ();
        ob.input();
        ob.area();
        ob.show();
        ob = new square ();
          .
          .
          .
        ob = new Rectangle ();
          .
          .
          .
    }
}
```

Example of abstract- class with method and constructor.

```
abstract- class Vehicle
 {
    int no;
    Vehicle(int no)
     {
        this.no = no;
        System.out.println("Parameterized of Base");
     }
    Vehicle()
     {
        S.o.p("Default- of Base");
     }
    abstract- void run();
    void change()
     {
        S.o.p("Changed");
     }
 }
class Honda extends Vehicle
 {
    Honda()
     {
        S.o.p("Default- of- Derived");
     }
    Honda(int no)
     {
        super(no);
        S.o.p("Parameterized of Derived");
     }
    void run()
     {
        S.o.p("Running");
     }
```

```
    void change ()
    {
        super.change ();
        S.o.p ("Changed in derived");
    }
}
class Test
{
    psvm (String args[])
    {
        Vehicle ob;
        ob = new Honda (10);
        ob.num ();
        ob.change ();
    }
}
```

Output :-

Parameterized of Base
Parameterized of Derived Running changed.
Changed in derived

(*) Vehicle ob = new vehicle → error since vehicle is an abstract class.

Q Difference between abstract class and interface.

→

Abstract class	Interface
• It can have abstract and non abstract methods	• It can have only abstract methods
• final, non final, static and non static variables	• only static and final variables.
• doesn't support multiple inheritance.	• supposts multiple inheritance
• can provide the implementation of interface.	• cant provide the implementation of abstract class.
• abstract keyword is required to declare the abstract class.	• interface keyword is regd to declare interface.
• Syntax	• Syntax
• Example	• Example.

Ans

Q How through interface, multiple interface can be achieved?

```
interface First
{
    void input()
}
interface Second
{
    void show()
}
```

```
class Test implements First, Second
{
    int no;
    public void input()
    {
        Scanner sc = new Scanner(System.in);
        no = sc.nextInt();
    }
    public void show()
    {
        System.out.println("value: " + no);
    }
}
class Demo
{
    psvm(String args[])
    {
        Test ob = new Test();
        ob.input();
        ob.show();
    }
}
```

EXCEPTION HANDLING

The Exception handling in java is one of the powerful mechanisms to handle the runtime errors, so that normal flow the application can be maintained.

Q What is exception?

→ Exception is an event that restricts the normal flow of the program. In java, it is an object which is thrown at runtime.

Though exception handling mechanism, the runtime errors, otherwise known as exception can be handled.

• Types of exceptions -

→ 2 types of exceptions are present in java -

* Checked exception.
* Unchecked exception.

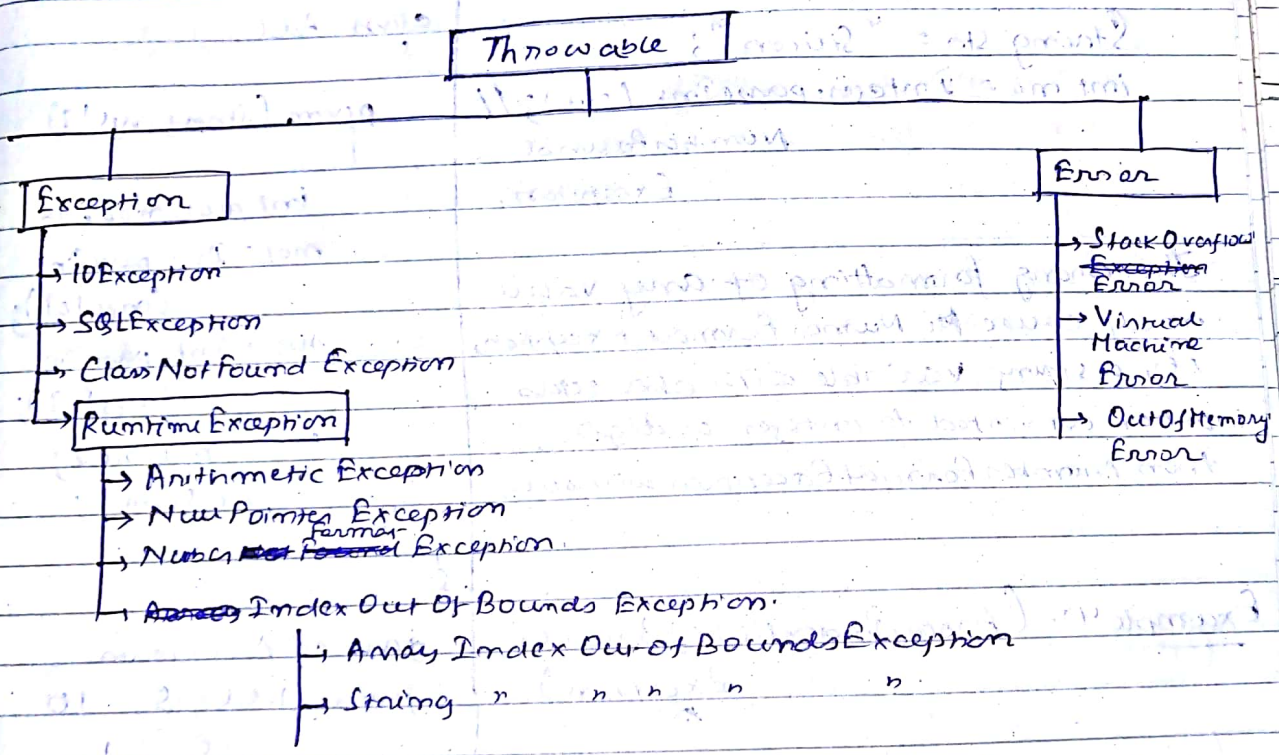
• Checked exception - The type of exception which occurs during ~~input-output~~ I/O operations, insert-delete operations, i.e., the exception which is occured during compilation time due to unavailability of file, class, database, etc is known as checked exception.

Ex - IO Exception
ClassNotFound Exception
SQLException

- Unchecked Exception - The kind of exception which occurs during runtime or execution time are known as unchecked exception.

Example :- Array Out of Bounds Exception

Hierarchy of Exception classes.



Example - 1 (Arithmetic Exception).

```
int no = 50/0;
```

This scenario may occur while dividing any number by zero.

Example 2 - (NullPointerException).

```
String str = null;
S.o.p(str.length());
```

If we have null value in any variable and still trying to perform any operation by the variable will generate NullPointerException.

Example 3 - (NumberFormatException)

```
String sta = "Silicon";
int no = Integer.parseInt(sta); //
```

NumberFormat Exception.

The wrong formatting of any value may cause Number Format Exception, i.e., a string variable with characters when converted to integer or digits, then NumberFormatException will occur.

```
class Add
{
  psvm(String args[])
  {
    int no1, no2, sum;
    no1 = Int.parseInt(args[0]);
    no2 = Int.parseInt(args[1]);
    sum = no1 + no2;
    S.o.p(sum);
  }
}
```

javac Add.java
java Add 5 10

	0	1
args	5	10

Example 4:- (ArrayIndexOutOfBoundsException).

```
int ar[] = new int[5];
ar[5] = 10; //
```

if the user will insert any value in the wrong index or the user will access any wrong index of the array, then ArrayIndexOutOfBoundsException will occur.

• JAVA Exception Handling Keywords :-

→ To handle exceptions in JAVA, 5 diff Keywords are used.

1. try
2. catch
3. finally
4. throw
5. throws.

① Try Block -

The java try block is used ~~that~~ to enclose the code that might throw an exception. It must be used within the method.

② Catch Block - This block is used to handle the exception. It must be used after the try block is complete.

Q. Write a program to enter 2 nos and find the division.

```
Class Div
{
    psvm(String args[])
    {
        Scanner sc = new Scanner(System.in);
        int a, b;
        double div;
        S.o.println("Enter 2 nos:");
        a = sc.nextInt();
        b = sc.nextInt();
        div = a/b;
        S.o.println("Division :-" + div);
    }
}
```

Using exception handling :

```
class Test
{
    psvm(String ...)
    {
        Scanner ...;
        int no1, no2, res;
        no1 : ...
        no2 : ...
        res = no1/no2;
        try
        {
            res = no1/no2;
            S.o.p(res);
        }
        catch (ArithmeticException ob).
        {
            S.o.p(" Division by zero is not allowed ");
            S.o.p(ob);
            S.o.p(ob.getMessage());
        }
    }
}
```

Output :- Division by zero is not possible
java.lang.ArithmeticException : / by zero
/ by zero.

1. Which one of the following code produced by the java compiler after compilation ?

→ Byte code

2. Find the output if there is no error, else point out the error

```
class Test
{
    int n;
}
class Main
{
    public static void main(String args[])
    {
        Test ob;
        System.out.println(ob.n);
    }
}
```

→ Error : Test ob = new Test(); instead of Test ob;

3. Which of the following statement is incorrect ?

→

4.
```
class Main
{
    int fun()
    {
        return 20;
    }
    public static void main(String args[])
    {
        System.out.println(fun());
    }
}
```

→ error as object of Main class is required to call fun() function.

5. What will be the output of the following code if we provide command line argument as!

java Test This is a command line program.

```
class Test
{
    public static void main(String args[])
    {
        System.out.println(args[0]);
    }
}
```

→ This.

6. Which of those acess modifica must be used in main() method for execution?

→ public

7. Which of the following option is correct with respect to "this" keyword in java?

→ refer to parent class object.

8. Which of the following methods is used to recieve the parameter via command-line arguments?

→ main()

9. Fim the output of the following code!

class output
{
public sta
{
Output

10. class Te
{
int
Tes
{
}
pu
{
}

→ error !

11. Which of
and in

→ int

```
class Output
{
    public static void main(String args[])
    {
        int a1[] = new int[10];
        int a2[] = {1,2,3,4,5};
        System.out.println(a1.length + " " + a2.length);
    }
}
```

~~a2 is the array a2 has not been defined~~

Output - 10 5

10.
```
class Test
{
    int x
    Test(int n)
    {
        x = x;
        System.out.println(x);
    }
    public static void main(String args[])
    {
        Test t1 = new Test();
    }
}
```

→ error : no argument passed ~~in the~~ during object creation

11. Which of the following will legally declare, construct and initialize an array?

→ int [] myList = {1, 2, 3};

12. Which of the following will not create an object of class Student ?

→ new Student();

13. Choose the right answer :

On System.out.println(), println() is a method of which class ?

→ PrintStream.

14. What will be the output of the program ?

```
class Test
{
    static int count = 100;
    public void increment()
    {
        count++;
    }
    public static void main(String[] args)
    {
        Test obj = new Test();
        obj.increment();
        Test obj2 = new Test();
        System.out.println(obj2.count);
    }
}
```

Output :- 101

15. What will be the output :-

```
class Test
{
    public static void main (String [] args)
    {
        try
        {
            int an[] = {1};
            an[1] = 22;
            System.out.println ("Inside the block");
        }
        catch (ArrayIndexOutOfBoundsException e)
        {
            System.out.println ("Exception caught" +e);
        }
    }
}
```

Output :- Exception caught java.lang.ArrayIndexOutOfBounds Exception: 1

16. What will be the output of the following program :-

```
class A
{
    A ()
    {
        S.O.println ("A class Const");
    }
}
class B extends A
{
    B ()
    {
        Sopln ("B class Const");
    }
}
```

```
class C extends B
{
    C()
    {
        System.out.println("C class const");
    }
}
class Test
{
    psvm(String args[])
    {
        C ob = new C();
    }
}
```

Output :- A class const
B " "
C " "

17. What exception type does the following program throw?

```
class Test
{
    psvm(String args[])
    {
        int s[] = {1, 2, 3};
        System.out.println(s[3]);
    }
}
```

ArrayIndexOutOfBoundsException.

18. Java supports all types of inheritance except -

→ Multiple inheritance.

19. What will be the output :-

```
Class A
{
    void func()
    {
        Sopen("base");
```

The JVM firstly checks whether the exception is handled or not. If exception is not handled, JVM provides a default exception handler that performs few tasks like -

- It prints out the exception, description, it prints the hierarchy of methods where the exception occured and terminate the program.
- If the exception is handled by the application programmer, normal flow of the application is maintained, i.e., rest of the code is executed.

JAVA MULTI CATCH BLOCK.

The user have to perform different task at different times which may generate diff exceptions for which multiple catch blocks are required.

```
class Test
{
    psvm (String args[])
    {
        try
        {
            int n = args.length;
            int res = 50/n;
            S.o.p(res);
            int arr[] = {20, 30, 40};
            arr[5] = 80;
            S.o.p(arr[1]);
        }
    }
    catch (ArrayIndexOutOfBoundsException ob)
    {
        S.o.p("Accessing value beyond index");
        S.o.p(ob);
    }
    Catch (ArithmeticException ob)
    {
        S.o.p ("Cannot be divided by zero");
        S.o.p(ob);
    }
}
```

- At a time, only one exception will occur and also one catch block will execute.
- If none of the catch block is unable to handle the exception, abnormal termination will occur.

Output :-

Nested try block :-

```
class Test
{
    psvm (String args[])
    {
        try
        {
            int n = args.length;
            int res = 50/n;
            S.o.p (res);
            try
            {
                int arr[] = {20, 30, 40};
                arr[5] = 80;
                S.o.p (arr[1]);
            }
            catch (ClassNotFoundException obj)
            {
                S.o.p (obj);
            }
        }
        catch (ArrayIndexOutOfBoundsException ob)
        {
            S.o.p ("Accessing value beyond index");
            S.o.p (ob);
        }
        catch (ArithmeticException ob)
        {
            S.o.p ("Cannot be divided by zero");
            S.o.p (ob);
        }
    }
}
```

- Any exception which will be generated ~~outside~~ inside the outer try block, the outer catch block are responsible to handle the exception.

- If exception is generated in the inner try block, then the inner catch block is the first to be responsible to handle the exception. If the inner ~~exc~~ catch block can't handle the catch block, then the outer catch blocks will handle the exception.

finally keyword :-

- JAVA finally block will always be executed whether exception is generated or not.

There are 3 different cases where finally block can be used

Case 1:- (Exception not generated).

```
class Test
{
  psvm (St- - )
  {
     try
     {
        int data = 25/2;
        S.o.p(data); // 5.
     }
     catch (NullPointerException ob)
     {
        S.o.p(ob);
     }
     finally
     {
        S.o.p("finally block");
     }
     S.o.p("Rest of code");
  }
}
```

Output :-

```
5
finally block
Rest of code
```

If there will be no exception generated from the try block then catch flow is not going to be exceeded but finally block will execute.

Case 2 :- (Exception generated but not handled)

```
class Test
{
    psvm (String args[])
    {
        try
        {
            int data = 25/0;
            S.o.p (data);
        }
        catch (NullPointerException ob)
        {
            S.o.p (ob);
        }
        finally
        {
            S.o.p ("Finally block");
            S.o.p ("Rest of code");
        }
    }
```

Output:-
Finally block.

If exception will be generated and no corresponding catch flow will handle the exception, then abnormal termination will occur but before that finally block will be executed only.

Case 3:- (Exception generated and handled).

```
class Test
{
    psvm (String args[])
    {
        try {
            int data = 25/0;
            S.o.p (data);
        }
        catch (ArithmeticException ob)
        { S.o.p ("Can't be divided by zero");
          finally
          { S.o.p ("Finally block"); }
          S.o.p ("Rest of code");
        }
    }
```

Output :- Can't be divided by zero
Finally block
Rest of code.

For each try block there can be zero or more than 1 catch block. but only one finally block.

* Generic Catch block :-

```
class Test
{
    psvm (String args[])
    {
        try
        {
            int data = 25/0;
            S.o.p(data);
        }
        catch(NullPointerException ob)
        { S.o.p(ob); }
        Catch(ArrayIndexOutOfBoundsException ob)
        { S.o.p(ob); }
        Catch(ClassNotFoundException ob)
        { S.o.p(ob); }.
        catch(Exception ob)
        {
            S.o.p("Generic catch block");
            S.o.p(ob);
        }
    }
}
```

The generic catch block needs to be at the end of all catch block else syntax error will be generated.

throw:-

```
class Test:
{
    perm(String args[])
    { try {
        int no1, no2, res;
        Scanner sc = new - - - .
        S.o.p("Enter values");
        no1 = sc.nextInt();
        no2 = sc.nextInt();
        if (no2 == 0)
        {
            throw new Arithmetic
                Exception();
        }
```

```
        else
        {
            res = no1 / no2;
            S.o.p(res);
        }
    }
    catch (ArithmeticException ob)
    {
        S.o.p("cant be divided by zero");
    }
}
}
```

When the user wants to throw the exception manually, throw keyword is used to explicitly throws an exception.

* An exception can be handled using try, catch and finally blocks.

* It is possible to handle multiple exceptions using multiple catch blocks.

* A single catch block can also be used to catch multiple exception using the OR operator.

* Even though there are possibilities for several exception in try block, at a time only one exception will be raised.

* A single try block can be followed by several catch blocks.

* We can not write a catch block without a try block but we can write a try block without any catch block.

* It is not possible to insert statements b/w try and catch block.

* It is possible to write a try block within try block known as nested try block.

Throws keyword :-

```
class Test
{
    void fun1() throws ArithmeticException
    {
        throw new ArithmeticException();
    }
    psvm(String args[])
    {
        try
        {
            Test ob = new Test();
            ob.fun3();
        }
        catch(ArithmeticException ob)
        {
            S.o.p("Exception Caught");
        }
    }
}

    void fun2() th
    {
        fun1();
    }
    void fun3() th
    {
        fun2();
    }
```

JAVA throws keyword is used to declare an exception. It gives an information to the programmer, that there may occur an exception, so it is better for the programmer to provide the exception handling code so that the normal flow can be maintained.

or

Q Difference between throw and throws ?

Throw	Throws.
• Java throw keyword is used to explicitly throw an exception	• Throws keyword is used to declare an exception
• Check exception cannot be propagated using throw only	• Check exception can be propagated using throws.
• Throw is followed by an instance	• Throws is followed by a class.
• Throw is used within the method.	• throws is used with the method signature.
• The user cannot throw the multiple exceptions.	• The user can declare multiple exceptions.
• Syntax	• Syntax
• Example	• void sum() throws ArithmeticException

try
{
throw new ArithmeticException ("found");
}

finalize() -

```
class Test{
    {
        public void finalize()
        {
            S.o.p("For memory cleaning");
        }
        psvm(String args[])
        {
            Test ob = new Test();
            ob.gc(); // garbage collection.
        }
    }
```

Q Difference between final, finally and finalize :-

final	finally	finalize
• It is used to apply restrictions on class, method and variable. Where final class cannot be inherited, final method cannot be overridden and final variable value can't be changed.	• finally is used to place imp code, it will be executed whether exception is handled or not.	• it used to perform clean up processing just before object is garbage collected or memory for object is deallocated
• final is a keyword	• finally is a block.	• finalize is a method
• Syntax.		
• Example.		

Package:

Access Specifier	Inside the class	Outside the class but inside the package with inheritance	Outside the package with inheritance	Outside the package with inheritance	Outside the package with no inheritance
private	✓	X	X	X	X
default	✓	✓	✓	X	X
protected	✓	✓	✓	✓	X
public	✓	✓	✓	✓	✓

Example 1:-

```
package pack1;
class A
{
    private void fun1()
    {
        S.o.p("private");
    }
    protected void fun2()
    {
        S.o.p("protected");
    }
    public void fun3()
    {
        S.o.p("public");
    }
    void fun4()
    {
        S.o.p("default");
    }
}
```

```
psvm(String args[])
{
    A ob = new A();
    ob.fun1();
    ob.fun2();
    ob.fun3();
    ob.fun4();
}
```

All kind of methods with any access specifier can be accessed inside the same class through object.

Outside the package with inheritance	Outside the package with no inheritance
X	X
X	X
✓	X
✓	✓

ing args[]).

new A();
();
();
();

thods with
ics can be
the same
ject.

Example 2 :-

```
package pack2;
class A
{
    Same as
    Example 1.
}
class B extends class A
{   psvm(String args[]) {
    B ob = new B();
    ob.fun1(); // Error.
    ob.fun2();
    ob.fun3();
    ob.fun4();
    }
}.
```

Example 3 :-

```
package pack1;
class A
{
    . . . .
}
class B
{
    psvm (. . . . )
    {
        A ob = new A();
        !
        !
    }
}
```

Example 4:-

```
package pack1
class A
{

}

package pack2
import pack1.A;
class B extends A
{
    psvm(---)
    {
        B ob = new B();
        ob.fun1(); // Error

        ob.fun4(); Error.
    }
}
```

Example 5:-

```
package pack1
class A
{

}

package pack2
import pack1.A;
class B
{
    psvm(---)
    {
        A ob = new A();
    }
}
```

Multi-threading

Thread - A thread is a light weight sub process (i.e., a smallest unit of processing. This is also considered as seperate path of execution.

Multithreading is a java feature that allows concurrent execution of 2 or more parts of a program for maximum utilization of C.P.U. Each part of such program is called a thread.

Through multithreading, multiple threads can be executed simultaneously of a single process where they share a common memory and all other resources which are allocated to the process, i.e., no seperate resources are allocated to any thread.

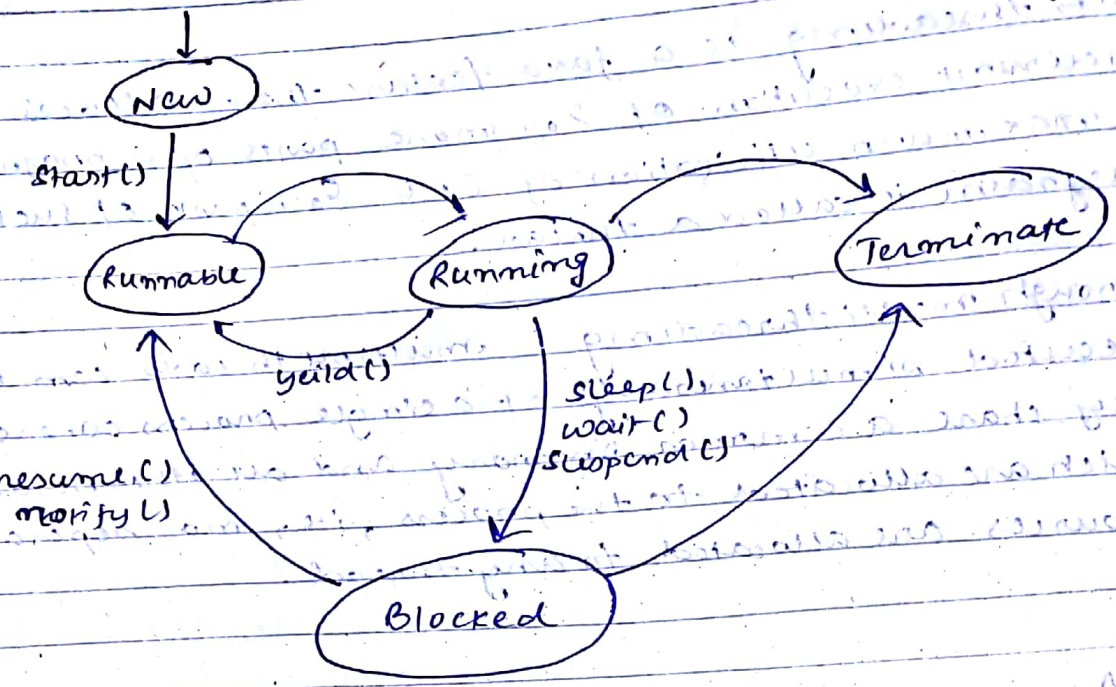
- Advantages of JAVA multithreading :-

→ It doesn't block the user because these threads are independent and the user can perform multiple operations at same time.

→ Time consumption will be less as multiple operations can be done together.

→ Threads are independent, so it doesn't affect other threads if exception occurs in a single thread.

Life cycle of thread -



- **New** - When a new thread is created, it will be entered to the new state. Multiple number of threads can stay in their new state but the thread has not yet started to execute.

- **Runnable** - When the thread is ready to run, it is moved to runnable state. In this state, a thread might actually be running or it might be ready to run at any instant of time.

A thread scheduler has the responsibility to provide time slot for execution of the thread. In multi-threading program, each thread will be allocated fixed amount of time for execution. In runnable state, a queue will be maintained consisting of threads which are ready to execute, i.e., each thread has all its reqd resources with it for execution.

Start method is used to bring the thread from new to runnable.

- Running - The scheduler again selects the thread from runnable queue and allocates processor for execution.

During execution if a thread needs resources, or due to arrival of other threads with more priority may pause their execution from the running state.

After successful completion of thread execution, it moves to terminate state.

- Terminate - A thread terminates because of two reasons; i.e., It exits normally when the code of the thread has entirely executed by the program and is considered as normal termination of threads.

The thread may terminate abnormally due to some unusual events like segmentation fault or unhandled exception.

The thread that dies in terminate state doesn't consume any more CPU cycle.

- Blocked - A thread will move to blocked state from running state due to 3 diff reasons -

(i) due to arrival of higher priority thread in runnable queue, the currently running thread will be suspended using suspend method. After completion

of execution, of higher priority thread in the running state, the blocked thread will move to the end of runnable queue using resume method.

A thread will move to blocked state from running state if it needs some resources and required resources are not available as it is used by other threads. Then, wait method is called for those kind of threads. Notify method is used for those threads when their required resources will be available. The blocked thread will acquire the required resources and will move to the end of runnable queue.

Sometimes, the scheduler forcefully moves a thread from running to blocked using sleep method with given time in millisecond. Once the time will become zero, automatically the blocked thread will move from blocked state to the end of runnable queue.

~~Running Thread Execution~~

* Yield -

One thread present in the running state can also be directly moved to the end of the runnable queue due to same priority thread. The scheduler calls the yield method to allow a thread with equal priority to provide a CPU cycle.

Thread Creation :-

There are two different ways to create a thread in Java :-

① Extending Thread class

② Implementing Runnable interface.

/* Multiple Threads are executing a single task, Way to create multiple threads by inheriting Thread class.

```
class MyThread extends Thread
{
    public void run()
    {
        int index;
        for(index=1; index<=100; index++)
        {
            System.out.println("value = "+index);
        }
    }
    psvm(String args[])
    {
        MyThread ob1 = new MyThread(); // new state
        MyThread ob2 = new MyThread();
        MyThread ob3 = new MyThread();
        ob1.start(); // Move from new to runnable.
        ob3.start();
        ob2.start();
    }
}
```

```
class Multithread extends Thread
{
    public void run()
    {
        try
        {
            // Display the thread that is currently running
            S.o.p ("Thread " + Thread.currentThread().
                    getId() + " is running");
        }
        catch (Exception ob)
        {
            S.o.p("Exception caught");
        }
    }
}
class Test
{
    psvm (String args[])
    {
        int i;
        for(i=0; i<10; i++)
        {
            Multithread ob = new Multithread();
            ob.start();
        }
    }
}
```

Thread scheduling / Thread scheduler in java :-

A thread scheduler in java is the part of the jvm that decides which thread should currently run or will be in the running state. There is no guarantee that which runnable thread will be chosen to run by the thread scheduler.

Only one thread at a time can run in a singe processing system.

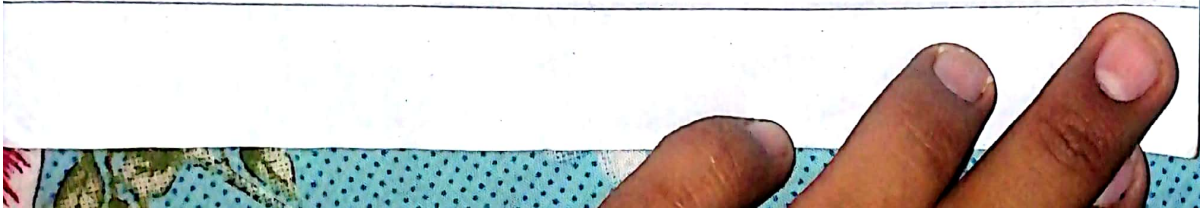
A thread scheduler mainly uses preemptive or timeslicing scheduling for the execution of threads.

Under preemptive scheduling ; the highest priority task executes until it enters the waiting or terminate state or a higher priority task comes for execution.

Under timeslicing, a task executes for a predefined slice of time and then reenters the pool of ready threads, i.e., it will allocate time to the next thread in the runnable state.

The scheduler sometimes determine which task should execute next, based on priority and other factors.

- sleep() :- This method is used to make a thread sleep for a specified amount of time.



```
class SleepExample extends Thread
{
    public void run()
    {
        int index;
        for(index = 1; index<=5; index++)
        {
            try
            {
                Thread.sleep(500);   // ms
            }
            catch (Exception ob)
            {
                S.o.p(ob);
            }
            S.o.p(index + " ");
        }
    }
}
class Test
{
    psvm(String args[])
    {
        SleepExample ob = new SleepExample();
        SleepExample ob1 = new SleepExample();
        SleepExample ob2 = new SleepExample();
        ob.start();
        ob1.start();
        ob2.start();
    }
}
```

Output :-

```
1
1
1
2
2
2
:
:
```

- join() - this method waits for a thread to die, in other words, it causes the currently running thread to stop executing until the thread it joins which completes its task.

```
class JoinExample extends Thread
{
    public void run()
    {
        int index;
        for (index=1; index<=5; index++)
        {
            try
            {
                Thread.sleep(8500);
            }
            catch (Exception ob)
            {
                S.o.p(ob);
            }
            S.o.p("index"+index);
        }
    }
}
class Test
{
    psvm (String args[])
    {
        JoinExample ob1 = new JoinExample();
        ob2?
        ob3?
        ob1.start();
        try
        {
```

```
            obj.join(); // obj.join(1500);
        }
        catch(Exception ob)
        {
            S.o.p(ob);
        }
        ob2.start();
        ob3.start();
    }
}
```

Output :-

1 2 3 4 5 1 1 2 2 3 3 4 4 5 5

1 2 3 1 4

- Thread Priority :-

In multithreading environment, thread scheduler assigns processor to a thread based on priority of thread. based on priority of thread. Whenever we create a thread in java, it always has some priority assigned to it.
Priority can either be a given by JVM while creating the thread or it can be given by the programmer explicitly.
The value of thread priority can range from 1 to 10.
Their are three static variables defined in thread class for priority.

(1) public static int MIN_PRIORITY.

This is minimum priority that a thread can have and whose value is 1.

(2) public static int NORM_PRIORITY.

This is the default priority of a thread if the user does not explicitly define the value, this value is 5.

(3). public static int MAX_PRIORITY.

This is the maximum priority of a thread which has value of 10.

Methods for thread priority :-

(1). public final int getPriority()

This method returns the priority of given thread.

(2). public final void setPriority(int value)

This method changes the value of the thread priority and assigns a new value as priority to the Thread

Eg :-

```
class ThreadDemo extends Thread
{
    public void run()
    {
        S.o.p("Inside run method");
    }
    psvm(S[] - - - )
    {
        ThreadDemo t1 = new ThreadDemo();
        "       t2 = "   "
        "       t3 = "   "
```

```
S.o.p ("Priority of t1 = " + t1.getPriority());
S.o.p ("Priority of t2 = " + t2.getPriority());
S.o.p ("Priority of t3 = " + t3.getPriority());
t1.setPriority(2);
t2.setPriority(8);
t3.setPriority(5);
// t3.setPriority(20); // it will show exception.
S.o.p ("Priority of t1 = " + t1.getPriority());
    "      "    t2 = "  + t2.getPriority();
    "      "    t3 = "  + t3.getPriority();

// Main Thread.
S.o.p (Thread.currentThread().getName());
S.o.p ("Priority of Main Thread " + Thread.currentThread().
                                    getPriority()); // 5
Thread.currentThread().setPriority(10);
S.o.p ("Now Main Thread Priority " + Thread.currentThread().
                                    getPriority()); // 10.
}
}
```

* If two threads have same priority then we cannot expect which thread will execute first. It depends on diff. thread scheduling algorithm, i.e., FCFS (First Come First Serve), RR (Robin Round).

* If we are using thread priority for thread scheduling then the sequence of execution can be given based upon the priority.

* All child Thread will get the same priority which is assigned to parent.

```
class ThreadDemo extends Thread.
{
    public void num()
    {
        S.o.p("invalid num method");
    }
    psvm(St - - - )
    {
        Thread.currentThread().setPriority(10);
        S.o.p("Main Thread Priority" + Thread.currentThread.getPriority());
        ThreadDemo t1 = new ThreadDemo();
        S.o.p("Priority of t1=" + t1.getPriority()); //10
    }
}
```

Q How multiple threads will execute multiple tasks or how to achieve multitasking using multithreading

```
class Simple1 extends Thread
{
    public void run()
    {
        S.o.println("task one");
    }
}
class Simple2 extends Thread
{
    public void run()
    {
        S.o.println("task two");
    }
}
class TestMultitasking3
{
    psvm(St---)
    {
        Simple1 t1 = new Simple1();
        Simple2 t2 = new Simple2();
        t1.start();
        t2.start();
    }
}
```

```
public class JavaSuspendExp extends Thread.
{
    public void run()
    {
        for(int i=1; i<5; i++)
        {
            try
            {
                sleep(500);
                S.o.pen(Thread.currentThread().getName());
            }
            catch(Exception e)
            {
                S.o.pen(e);
            }
            S.o.pen(i);
        }
    }
psvm(String args[])
{
    JavaSuspendexp t1 = new JavaSuspendExp();
                n          t2 =          n          ();
                n          t3 =          n          ();
    t1.start();
    t2.start();
    t2.suspend();
    t2.start();
    t2.resume();
    }
}
```

```
pubvic clam TestGanbage1
{
    public void finalize()
    {
        S.O.pen("object is ganbage collected");
    }
    psvm(String args[])
    {
        TestGanbage1 S1 = new TestGanbage1();
                     S2 =                    ;
        S1 = null;
        S2 = null;
        System.gc();
    }
}
```

Synchronization

Synchronization in JAVA is the capacity to control the access of multiple threads to any shared resource.

It is a better option where we want to allow only one thread to access the shared resource.

Synchronization is mainly used to prevent thread interference and to prevent consistency problem.

* Types of thread synchronization :-

There are two different types of thread synchronization can be implemented i.e.,

(i) Mutual Exclusive.

(ii) Inter-thread Communication.

(1) Mutual Exclusive :-

It helps to keep threads from interfering with one another while sharing data.

Mutual Exclusion can be implemented by using 3 diff methods :-

(i) By using synchronized method.
(ii) " " " block.
(iii) " " static synchronization.

Concept of lock in JAVA :-

Synchronization is build around an internal entity known as the lock or monitor. Every object has a lock associated with it. By convension, a thread that needs consistent access to an object's fields has to acquire the object's lock when the work gets over.

* Problem without synchronization :-

```
class Table
{
  void show(int n)
  {
    for(int i=1;i<=index;i++)
    {
      S.o.p(n*index);
      try
      {
        Thread.sleep(500);
      }
      catch(Exception ob)
      {
        S.o.p(ob); }
    }
  }
}
class MyThread1 extends Thread
{
  Table obj;
  MyThread1(Table obj1)
  {
    obj=obj1; }
  public void run()
  { obj.show(5); }
}
```

```
class MyThread2 extends Thread
{
  Table ob;
  MyThread2(Table ob1);
  { ob = ob1; }
  public void run()
  { ob.show(100); }
}
class Test
{
  psvm(String args[])
  {
    Table ob=new Table();
    MyThread1 ob1=new ...(ob);
    MyThread2 ob2=new ...(ob);
    t1.start();
    t2.start();
  }
}
```

Both class with shared same object as resource.

Using 'synchronized' keyword lock can be applied for any shared resource.

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

Synchronised block in java:-

It can be used to perform synchronisation on any specific resource of the method. For ex, if the method contains 50 lines of code but we need to synchronize only 5 lines, then synchronized block is required instead of synchronised method.

If the user will put all the course of the method in the synchronized block, it will work like synchronised method.

This block is used to block an object for any shared resource whose scope is smaller than synchronized method.

```
class Table
{
    void show (int n)
    {
        S.o.p("start");
        S.o.p("Not belongs to synchronized block");
        Synchronized(this) // synchronized block.
        {
            for(index = 1; index <= 5; index++)
            {
                S.o.p(n * index + "\t");
            }
            try
            {
                Thread.sleep (500);
            }
            catch (Exception ob)
            {
                S.o.p(ob);
            }
        }
        S.o.p("Exit");
    }
}
```

```
class MyThread1 extends Thread
{
    Table obj;
    MyThread1(Table obj1)
    {
        obj = obj1;
    }
    public void run()
    {
        obj.show(5);
    }
}
class MyThread2 extends Thread
{
    Table obj2;
    MyThread2(Table obj3)
    {
        obj2 = obj3;
    }
    public void run()
    {
        obj2.show(100);
    }
}
```

• Static synchronization:

Suppose there are 2 objects of a shared class (Printp-Table) named ob1 and ob2. In case of synchronized method and synchronized blocks, there will not be any interference between t1 and t2 or t2 and t4 because t1 and t2 both refers to a common object that have a single lock. But there can be interference b/w t1 and t3 or t2 and t4 because t1 acquires another lock and t3 acquires another lock. By using static synchronization, The lock will be on the class, not on object, so that no interference will occur b/w t1 and t3 or t2 and t4.

```
class Table
{
    synchronized static void show(int n)
    {
        for (index=1; index<=5; index++)
        {
            S.o.p(n*index, "\t");
        }
        try
        {
            Thread.sleep(500);
        }
        catch (Exception eob)
        {
            S.o.p(eob);
        }
        S.o.p("Exit");
    }
```

```
class MyThread1 extends Thread
{
    public void run()
    {
        Table.show(2);
    }
}
class MyThread2 extends Thread
{
    public void run()
    {
        Table.show(3);
    }
}
class MyThread3 extends Thread
{
    public void run()
    {
        Table.show(4);
    }
}
class MyThread4 extends Thread
{
    public void run()
    {
        Table.show(5);
    }
}
class
{
    psvm(String args[])
    {
        MyThread1 t1 = new MyThread1();
        t2 = new MyThread1();
        t3 = new MyThread1();
        t4 = new MyThread1();
```

```
        t1.start();
        t2.start();
        t3.start();
        t4.start();
    }
}
```

Interthread Communication :-

It is otherwise known as co-operation i.e., ~~allowed~~ allowing synchronised thread to communicate with each other. It is a Mechanism in which a thread is paused running till its critical section and another thread is allowed to enter the same critical section to be executed.

It is implemented by using methods like :-

(1) wait()
(2) notify()
(3) notifyAll().

Procedure :-

→ Threads enter to acquire lock.

→ lock is acquired by one thread at a time.

→ Thread enters to waiting state using wait() from the object. Otherwise it releases the lock and exits.

→ If the user calls notify() or notifyAll(), thread moves to runnable state, i.e., notify state. Now thread is available to acquire lock.

→ After completion, thread releases lock and exist the monitor state.

```
class Customer
{
    int amount = 10,000;
    synchronized void withdraw (int amount)
    {
        S.o.p ("Going to withdraw");
    }
    if (this.amount < amount)
    {
        S.o.p ("less balance, waiting for deposit");
        try
        {
            wait ();
        }
        catch (Exception ob)
        {
            S.o.p ("Exception caught");
        }
    }
    this.amount = this.amount - amount;
    S.o.p ("withdraw Complete");
Syn
    synchronized void deposit (int amount)
    {
        S.o.p ("Going to deposit");
        this.amount += amount;
        S.o.p ("Deposit complete");
        notify ();
    }
} // end of customer class.
```

```
class MyThread1 extends Thread
{
    public void run()
    {
        Customer.withdraw(15000);
    }
}
class MyThread2 extends Thread
{
    public void run()
    {
        Customer.deposit(20000);
    }
}
class InterThread
{
    psvm(String args[])
    {
        MyThread1 t1 = new MyThread1();
        MyThread2 t2 = new MyThread2();
        t1.start();
        t2.start();
    }
}
```

Diff between notify() and notifyAll() :-

On notify(), it wakes up a single thread, i.e., waiting for this object. If multiple no of threads are waiting for same object then one of them will be chosed for execution. This is done arbitarily using scheduling mechanism.

~~On notifyAll()~~

On notifyAll(), it wakes up all threads waiting for the object.

Deadlock
It can o
an objec
and sec
acquire
for each
deadlock

```
class T
{
    P
```

Deadlock - in java is a part of Multi Threading. It can occur in a situation when a thread is waiting for an object lock, that is occupied by another thread and second thread is waiting for an object lock which is acquired by 1st thread. Since both threads are waiting for each other to release the lock, the condition is called deadlock.

```
class TestDeadlock
{
    psvm (String args[])
    {
        String resources = "Silicon";
        String resources = "FBS";
        // t1 tries to lock resource1 and resource2
        Thread t1 = new Thread()
        {
            public void sum ()
            {
                synchronized (resource1)
                {
                    S.o.p ("Thread1 : locked resource1");
                }
                try
                {
                    Thread.sleep (100);
                }
                catch (Exception ob)
                {
                    S.o.p (ob);
                }
                synchronized (resource2)
                {
                    S.o.p ("Thread1 : locked resource2");
                }
            }
        };
```

```
// t2 tries to lock resource1 and resource2
Thread t2 = new Thread()
 {
     public void run()
      {
         synchronized (resource2)
         {
            S.o.p(" Thread2: lock resource 2");
         }
         try
         {
            Thread.sleep(100);
         }
         catch (Exception ob)
          {
              S.o.p(ob);
          }
         synchronized (resource1)
          {
             S.o.p(" Thread2: locked resource1");
          }
      }
 };
 t1.start();
 t2.start();
 }
}
```

- Thread creation by using runnable interface :-

The runnable interface should be implemented by any class whose instances are need to be executed by a thread.

Runnable interface have only one method run().

```
class Test implements Runnable.
{
    public void run()
    {
        S.o.p ("Runnable Example");
    }
    psvm (String args[])
    {
        Test ob = new Test();
        Thread t1 = new Thread (ob);
        t.start();
    }
}
```

If we are not extending the thread class, our class object will not be treated as thread object. So, the user needs to explicitly create Thread class object and also need to pass the object of the class which implement runnable so that the run() method of the class may execute.

Q Why there are 2 different approaches to achieve multithreading ?

- If the user extends the Thread class, own class cannot extend any other class because java doesn't support multiple inheritance. But if the user implements Runnable interface then the class can extend another base class if required.

String

String is a sequence of characters, but in java, string is an object that represents a sequence of characters.

```
char ch[] = {'s', 'i', 'l', 'i', 'c', 'o', 'n'};
String str = new String(ch);
```

OR

```
String str = "silicon";
```

OR

```
String str = new String("silicon");
```

- How to create a string object ?

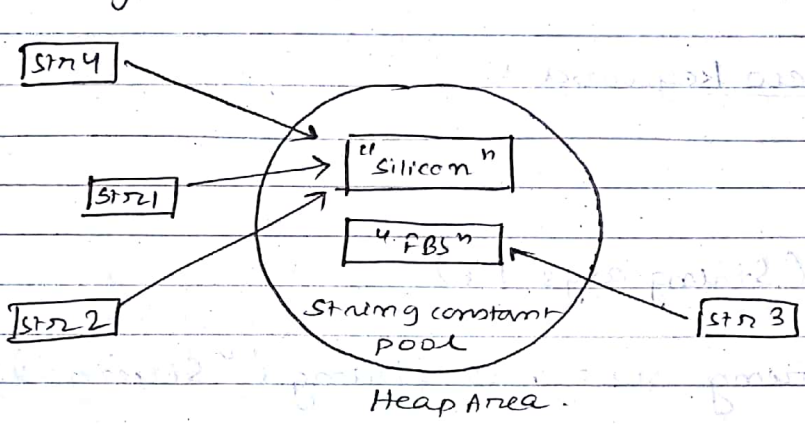
→ There are 2 different mechanisms to create string objects :-

(i) by using string literal.

(ii) by using new keyword.

String literal :-

Java string literal is created by using double quotes.
Eg:- String str = "B Ritwik".




```
/* class Test
{
    psvm (String args[])
    {
        String str1 = "silicon";
        String str2 = "silicon";
        String str3 = "FBS";
        String str4 = "silicon";
    }
}
*/
```

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pool instance is returned but if the string doesn't exist in the pool, a new string instance is created and is placed in the pool.

For eg :- only one object will be created firstly, JVM will not find any string object with value "Silicon" in the string constant pool but latter it will not create any new object but will return the reference to the same instance.

To make Java memory efficient, string literal is used.

- By using new keyword :-

```
class Test
{
    psvm (String args [])
    {
        String str = new String ("Silicon");

    }
}
```

In this case JVM will create a new string object in (non-pool) heap memory and the literal "Silicon" will be placed in the string constant-pool.

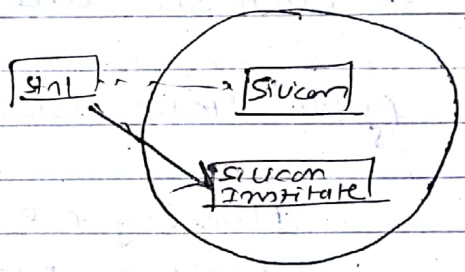
```
class Test
{
    psvm (String args [])
    {
        String str1 = "Java"; // String literal.
        char ch[] = {'H', 'e', 'l', 'l', 'o'};
        String str2 = new String (ch);
```

```
        String str3 = new String("String Example");
        S.o.p(str1);
        S.o.p(str2);
        S.o.p(str3);
    }
}
```

Output :-

JavaHelloStringExample.

In java, string objects are immutable., i.e., unmodifiable, unchangable, i.e., once a string object is created, its data or state cannot be changed but a new string object is created.




```
class Test
{
    psvm(String args[])
    {
        String str1 = "silicon";
        S.o.p(str1); //Silicon
        str1.concat("Institute"); *
        S.o.p(str1); //Silicon
    }
}
```

* str1 = str1.concat("Institute");
S.o.p(str1); // SiliconInstitute.

Two objects are created in the name of silicon and silicon Institute. Ist strl reference variable will still refers to silicon.

Suppose there are 5 reference variables and all refers to one object silicon. If one reference variable changes, the value of the object, then all reference variables will be affected. So, String objects are immutable in java.

WAP to implement producer, consumer problem using inter thread communication.

→
```
public class ProducerConsumerTest
{
    public static void main (String args [])
    {
        Memory c = new Memory();
        Producer p1 = new Producer (c, 1);
        Consumer c1 = new Consumer (c, 1);
        p1.start();
        c1.start();
    }
}
class Memory
{
    private int contents;
    private boolean available = false;
    public synchronized int get()
    {
        while (available == false)
        {
```

```
        try
        {
            wait();
        }
        catch (InterruptedException e)
        {
        }
    }
    available = false;
    notifyAll();
    return contents;
}

public synchronized void put(int value)
{
    while (available == true)
    {
        try
        {
            wait();
        }
        catch (InterruptedException e) { }
    }
    contents = value;
    available = true;
    notifyAll();
}
}

class Consumer extends Thread
{
    private Memory c;
    private int number;
    public Consumer(Memory c, int number)
    {
```

```
        c1 = c;
        this.number = number;
    }

    public void run()
    {
        int value = 0;
        for(int i = 0; i < 10; i++)
        {
            value = c1.get();
            S.o.pln("Consumer #" + this.number
                    + " got : " + value);
        }
    }
}

class Producer extends Thread
{
    private Memory c1;
    private int number;
    public Producer(Memory c, int number)
    {
        c1 = c;
        this.number = number;
    }

    public void run()
    {
        for(int i = 0; i < 10; i++)
        {
            c1.put(i);
            S.o.pln("Producer #" + this.number +
                    "put : " + i);
            try {
                Thread.sleep(1000);
            }
            catch(InterruptedException e) { }
        }
    }
}
```

Methods

(1) St

ex :-

Str

(2).

(2). in

ex

(3). ch

(4).

(5).

Methods of string class :-

(1) String concat (String str)

ex :-

String s3 = s1.concat (s2) ; // String s2 will be concatinated after s1 and will be stored in s3;

(2).

(2). int length ( )

ex :- str.length (); // return the string length.

(3). char charAt(int index)

ex :- str.charAt (5);

(4). int compareTo (String str)

ex :- s1. compareTo (s2);

This method is useful to compare two strings and to know which string is bigger or smaller. If s1 and s2 are equal, then this method returns 0, if s1 > s2, it returns a +ve number, if s1 < s2, it returns a -ve number.

(5). int compareToIgnoreCase (String str).

ex :- s1. compareToIgnoreCase (s2);

⑥ boolean equals (String str).

ex:- s1.equals(s2);

This method returns true if two strings are equal otherwise returns false. It is case sensitive.

⑦ boolean equalsIgnoreCase (String str)

ex:- equalsIgnoreCase(s2);

⑧ boolean startsWith (String str)

⑧ int indexOf (String s)

⑨ boolean endsWith (String s).

ex:- s1 = "Ritwik"

s2 = "ik"

s1.startsWith(s2); // False

s1.endsWith(s2); // True

⑩ int lastIndexOf (String s.)

ex:- s1.lastIndexOf(s2);

```
class Test
{
    psvm (String args[])
    {
        String s1 = "Hello";
        String s2 = "Hello";
        if (s1.equals(s2))
        {
            S.o.p("Equal");
        }
        else
        {
            S.o.p("Not-equal");
        }
    }
}
```

```
class Test
{
    psvm (String args[])
    {
        String s1 = "Hello";
        String s2 = "Good"; String s4 = "Hello I am
        String s3 = s1+s2;                from Silicon";
        S.o.p(s3);
        S.o.p(s1.toUpperCase());
        S.o.p(s2.toLowerCase());
        ss = s4.split(" ");
        for (int index = 0; index < ss.length; index++)
        {
            S.o.p(ss[index]);
        }
    }
}
```

File Handling :-

1. Java FileWriter class -

This class is used to write character oriented data to a file..

Q. WAP to write data into file character wise.

```
import java.io.*;
import java.util.*;
class FileWTest
{
    psvm (String args[])
    {
        // Input a string
        String str = new String ();
        S.o.p("Enter String:");
        Scanner sc = new Scanner(System.in);
        str = sc.nextLine();
        // Attach file to FileWriter class object.
        FileWriter fw = new FileWriter("F:\\Java\\myfile.txt");
                                        path of file.
        // Read characters wise from string and write into
           file.
        for(int index = 0; index < str.length(); index++)
        {
            fw.write(str.charAt(index));
        }
        fw.close();
    }
}
```

Q WAP to write data string by string into the file using FileWriter class.

→
```
import java.io.*;
import java.util.*;
class FileWTest
{
    psvm (String args[])
    {
        String str = new String();
        S.o.p("Enter String:");
        Scanner sc = new Scanner(System.in);
        str = sc.nextLine();
        FileWriter fw = new FileWriter("F:\\Java\
                                        \MyText.txt");
        while(!(str.equals("exit")))
        {
            fw.write(str);
            S.o.p("Enter String");
            str = sc.nextLine();
        }
        fw.close();
    }
}
```

- Java FileReader class :-

WAP to use FileReader class to read data from file character wise.

→ import java.io.*;
import java.util.*;
class FileRTest
{
psvm(String args[])
{
FileReader fr = new FileReader("f:\Java\MyText.txt");
int value;
while((value=fr.read())!=-1)
{
S.o.p((char)value);
}
fr.close();
}
}

Q WAP to use FileReader class to read data from file string wise.

```
import java.io.*;
import java.util.*;
class FileRTest
{
psvm(String args[])
{
FileReader fr=new FileReader("F:\Java\MyText.txt");
BufferedReader br=new BufferedReader(fr);
while((sto=br.readline())!=null)
{
S.o.p(sto);
}
fr.close();
}
}
```

Stream.

- A stream carries data from one place to another. It can be categorized into input stream and output stream.

- Input streams are the streams which recieve or read data while output streams send or write data.

- Input streams read data from keyboard, so we can attatch the keyboard to one input stream, so that a stream can read the data type on the keyboard.

- The System class has 3 different fields :-

  → System.in - it represents the object of InputStream class. The object represents the standard input device, i.e., keyboard by default.

  → System.out - it represents the object of PrintStream class. This object represents the standard output device, i.e, monitor by default.

  → System.err - it represents the object of PrintStream class. The object by default represents the standard output device, i.e., monitor.

Q. What is the difference between System.in and System.out?

→ Both are used to display messages on the monitor, but System.out is used to display normal messages and System.err is used to display any error messages.

Q What is the advantage of stream?

→ Stream is used to move data from one place to another which is used to recieve data from an input device and send data to an output device. Streams are categorized into 2 categories -

(i) byte Stream
(ii) text Stream.

- The classes which are belonging to byte Stream are FileInputStream, FileOutputStream, BufferedInputStream and BufferedOutputStream.

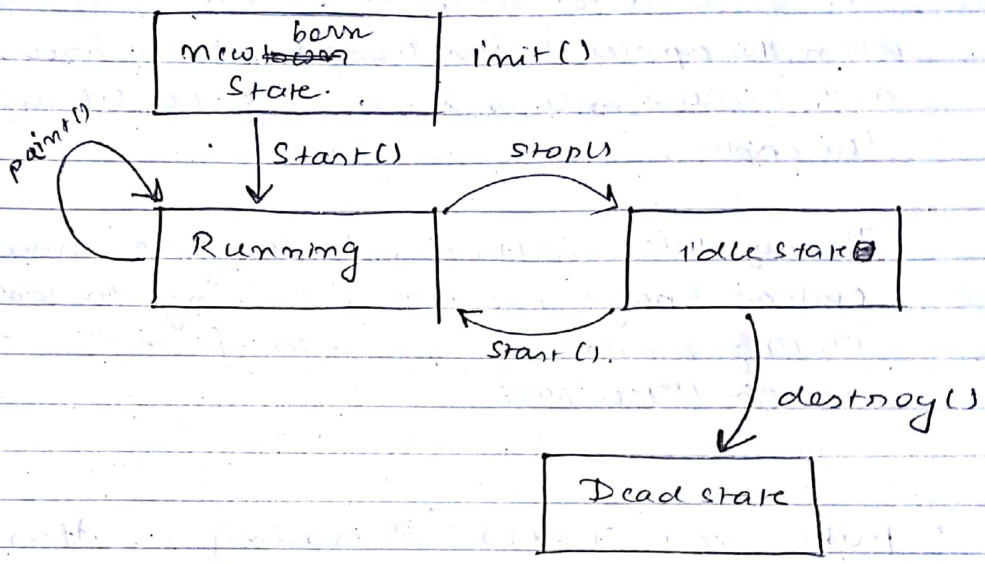
- The classes which are belonging to text Stream are FileReader, FileWriter, BufferedReader and BufferedWriter.

byteStreams are used to handle audio, video and image files and also charecters. But textStreams can always retrieve data in the form of text and charecter only.

Applet.

- Applet is class file that displays graphics applications in the web browser and user can embed or add applet codes in the webpage by using HTML tag.

- Life cycle of Applet :-



Applets donot have any main method or constructor, so one can compile an applet code but cannot run it. To execute / run an applet code HTML / appletviewer is required.

When an applet is executed within the web browser or in an applet window, it goes through the four stages of ——— initialize, started, stop, destroy. This stages correspond to the applet method, i.e., init(), start(), stop() and destroy() respectively. All those methods defined in the ~~app~~ Applet class which is called automatically by the browser or the applet viewer controlling the applet.

All these methods are abstract methods by default, so to perform specific functions, these methods needs to be overridden in the users applet so that the browser can call the user's code correctly.

- public void init() / new born state -

This method is used to perform any initialization that is needed for the applet. It is called automatically when the applet is first loaded into the browser and is called only once during the life cycle of the applet.

Through this method, loading of resources such as image, audio, creating threads and getting string parameters, values from the apler tag in the HTML page.

- public void start() / Running - After the applet is loaded and initialized, the java environment automatically calls the start(). It is also called any time it returns to the running state. This method is used to start the processing for the applet.

start() can be called any no. of times during the life cycle. Through this method action performed like animation and starting of others thread execution can be done.

- public void stop() / Idle state - This method is called automatically by the browser when the user moves off the html page containing the applet. This is generally used to suspend the applet's execution. So that it doesn't consume system resources when the user is not viewing the HTML page ~~th~~.

  Through this method, the background processing or calculation can be done which should not be viewed by the user.

- public void destroy() / dead state - this method is called after the stop() when the user exits the web browser normally. This method is also used to clean up the resources allocated to applet.

  Like init(), destroy() will be called only once during the life cycle of applet.

Q. Write an applet program to display Hello message.

```
import java.awt.*;
import java.applet.*;
class TestApplet extends Applet
{
    // Set the background colour
    public void init()
    {
        SetBackground( Color.yellow);
    }
    // Display the message
    public void paint(Graphics ob)
    {
        ob.drawString("Hello", 50,100);
    }
}

// Demo.html.

<html>
    <applet code = "TestApplet.class"
        height = 200 width = 300 >
    </applet>
</html>
```

Compile :- appletviewer Demo.html.

Q Write an applet program to execute diff. operations of applet.

```
import java.awt.*;
import java.applet.*;
class TestApplet extends Applet
{
    String msg = " ";
    // this method is created when applet
    // will be loaded.
    public void init()
    {
        // set background color
        setBackground(Color.red);
        // set font for text in applet.
        Font ob = new Font("Arial", Font.BOLD, 20);
        setFont(ob);
        msg += "init ";
    }
    // method called after init()
    public void start()
    {
        msg = "start";
    }
public void stop()
    {
        msg = "stop";
    }
    public void destroy()
    {
        msg = "destroy";
    }
    public void paint(Graphics ob1)
    {
        ob1.drawString(msg, 20, 150);
    }
}
```

//Demo.html

```
<html>
    <applet code = "TestApplet.class"
             height = 200  width = 300 >
    </applet>
</html>
```

- WAP to enter string into a file. Read the file. Write all ~~vowels~~ vowels into vowel file and consonant to consonant file.

- Write some nos into a file. Read the content, write all odd into one file and even in other files.

```
class MyThread extends Thread
{
    public void run()
    {
        for (int i = 0; i < 10; i++)
        {
            S.o.pln("Priority" + Thread.currentThread().
                                    getPriority());
            S.o.pln("Thread.currentThread().getName());
            try
            {
                S.o.pln("child Thread" + i);
                Thread.sleep(20);
            }
            catch (Exception ob)
            {
                S.o.p(ob);
            }
        }
    }
}
class parentchild
{
    psvm (String args[])
    {
        MyThread ob = new MyThread();
        "    ob1 = "    "    ();
        "    ob2 = "    "    ();
        ob.start();
        ob2.start();
        ob1.start();
        ob.setName("RBS");
        ob2.setName("silicon");
        ob.setPriority(6);
        ob1.setPriority(5);
        ob.setPriority(Thread.MAX_PRIORITY);
```

```
S.o.pln(ob.getPriority());
for(int i=0; i<10; i++)
{
    try
    {
        S.o.pln("main thread"+i);
        Thread.sleep(1000);
    }
    catch("Exception.obj)
    {
        S.o.p(obj);
    }
}
}
}
```

Q WAP to implement parent child concept using multithreading. Parent thread will print all the odd nos and child thread will print all the even nos b/w a given set of values.

```
import java.applet.*;
import java.awt.*;
public class appletParameter extends Applet
{
    private String strDefault = "Hello! Java Applet.";
    public void paint(Graphics g)
    {
        String strParameter = this.getParameter("Message");
        if (strParameter == null)
            strParameter = strDefault;
            g.drawString(strParameter, 50, 25);
    }
}
```

```
<html>
    <head>
        <title> Passing parameter in Java </title>
    </head>
    <body>
        This is the applet : <p>
        <Applet code = "appletParameter" width = "800"
                                        height = "100" >
        <Param name = "Message" value = "welcome
        in Passing parameter in java applet example."/>
        </applet>
    </body>
</html>.
```