

# JAVA

---

## History

- James gosling is the father of java, and introduced in the year 1991.
- The software was named as green talk and the team which developed is called green team. Later the software was renamed as Oak. Oak is a symbol of strength and it is a national tree for Germany.
- There was already existing company called Oak technology which had become a legal issues due to this legal issues they changed the software name Oak to java.
- James gosling and his team went for a coffee to an island and the coffee shop named was java hence they kept the software name as java , since they went for a coffee they kept the coffee bug as a logo for the java software.
- Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX

---

## Tokens

Token is the smallest unit of program.

In token we have,

- |               |               |             |
|---------------|---------------|-------------|
| 1. Identifier | 2. Keywords   | 3. Literals |
| 4. Operators  | 5. Separators | 6. Comments |

---

### 1. Identifier

Identifier is a name given for the java program.

---

### 2. Keywords

Keywords are the pre – defined word which has its own meaning.

In java , we have 50 keywords as follows,

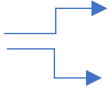
Abstract	continue	for	new	switch
Assert	default	goto	package	synchronized
Boolean	do	implements	private	this
Break	double	import	protected	throw
Byte	else	instanceof	public	throws
Case	enum	int	return	transient
Catch	extends	interface	short	try
Char	final	long	static	void
Class	finally	native	strictfp	volatile
Const	float	new	super	while

---

### **3. Literals**

Literals are the value which is used in java programming languages.

In literals we have,

1. **Number literals:**  Integer literals → (9,8,7,6,5)  
Decimal literals → (0.55, 0.75)

---

2. **Character literals** : enclosed in single quotes ( ' ' ) → ( 'A', '5', '\$', ' , ' ' )

---

3. **String literals** : enclosed in double quotes ( " " ) → ( " Hello", "\$#\*&" )

---

4. **Boolean literals** : TRUE, FALSE

---

### **4. Operators**

Operator is a symbol which is used to perform some operations on the operands.

Ex: 5 + 3 → operands

↓  
Operator

Operator type	Category	Precedence
UNARY	Postfix Prefix	Expr++, Expr--, ++Expr , -- Expr , +Expr , -Expr=!
ARITHMETIC	Additive, multiplicative	*/ % , + , -
SHIFT	Shift	<< , >> , >>>
RELATIONAL	Comparison Equality	< > , <= , >= , instance of == , !=
BITWISE	Bitwise AND , Bitwise exclusive OR, Bitwise inclusive OR	& ^ 
LOGICAL	Logical AND Logical OR	&& 
TERNARY	Ternary	? :
ASSIGNMENT	Assignment	+= , -= , *= , /= , %= , &= , ^= , <<= , >>= , >>>=

## **5. Separators**

Separators is used to separate the given code.

In java we have,

{ } → braces, [ ] → brackets, ( ) → parenthesis, ; → semicolon , ( , ) → comma

## **6. Comments**

Comments are used to provide the additional information for the java program.

In comments we have,

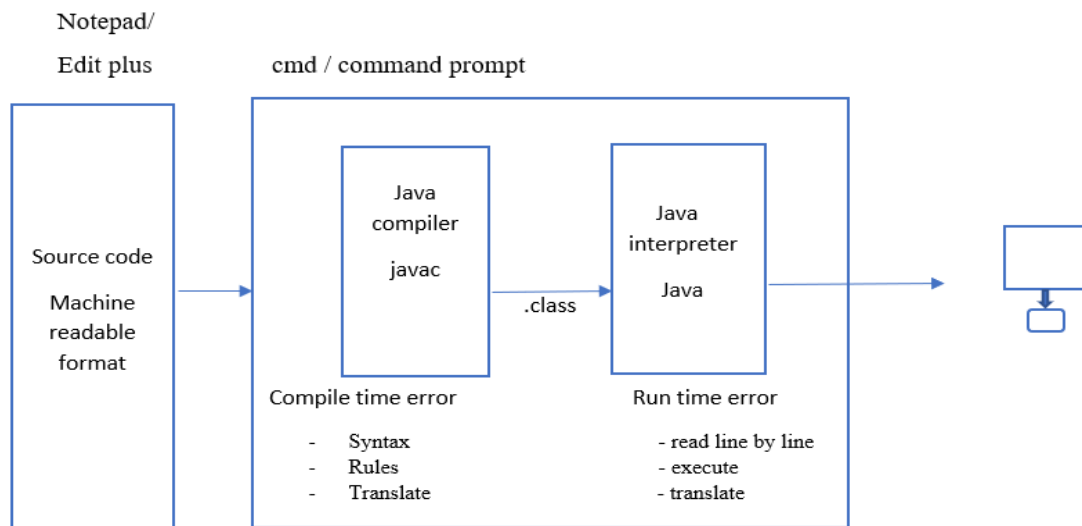
1. Single line comments

```
//.....//
```

2. block line comments

```
/*.....*/
```

## Java architecture



- We will write the program in edit plus / notepad. It is also called as source code which is in human readable format.
- To convert this human readable format into machine readable format, we have to go to command prompt and perform 2 operations
  1. Javac ( java compiler)
  2. Java ( java interpreter)
- Once after writing the program we have to save the file with the extension of .java in below path  
c:/pgmfiles/java/ Jdk 1.8 / bin
- Once after saving we have to give this .java file as an input to the compiler where it will check for 1). Syntax 2). Rules 3). Translate from .java to .class file
- If there is any syntax or rules violation we get compile time error.
- If there is no violation then the .class file will be generated.
- .class is an intermediate code which is in byte code format which cannot understood neither by human nor by machine.
- .class file will be given as input for the interpreter which 1). Reads line by line 2). Execute ( JVM ) 3). Translate .class file to binary language.
- If we find any abnormal statements like 1/0 which is infinite , which is not defined in java hence we get run time error called Arithmetic exception.

**JIT [Just in time ] :**

It is whole responsible to convert . class file to binary format.

---

**JVM [Java virtual machine ] :**

It is whole responsible to execute the java program.

---

**JRE [Java runtime environment ] :**

It is an environment setup provided to run the java program.

---

**JDK [Java development kit ] :**

In this kit which consist of all library files and utilities to develop a java software.

All the java program will execute from left to right and top to bottom.

---

**Java software ( JDK )**

JDK 1.0	JDK 1.4	JDK 1.8	JDK 1.12
JDK 1.1	JDK 1.5 / 5.0	JDK 1.9	JDK 1.13
JDK 1.2	JDK 1.6	JDK 1.10	JDK 1.14 → latest version
JDK 1.3	JDK 1.7	JDK 1.11	

---

**IDE ( INTEGRATED DEVELOPMENT ENVIRONMENT )**

**ECLIPSE → SE ( Standard edition ) → standalone application**

**EE ( Enterprise edition ) → Enterprise application**

**ME ( Micro edition ) → mobile application**

**NETBEANS**

---

**COMMANDS USED IN JAVA**

Cls → clear screen , Mkdir → make directory , cd → change directory

cd.. → change directory from current directory , java → java interpreter,

Javac → java compiler

**JAVA** : Java is a programming language which is used to develop software.

Java is object oriented programming language.

**Java features :**

- It is secured language.
- It is robust.
- It is platform independent.
- High performance.
- Compiled and interpreted.
- Multi – threaded.
- It is object oriented.
- It is polyglot (it means the program which we write using java syntax will be understood by various programming language).

---

In java , we have 4 types

- |          |               |
|----------|---------------|
| 1. Class | 2. Interface  |
| 3. Enum  | 4. Annotation |

---

**1. Class**

Class is a blueprint or template to create an object.

**Ex: : program to print Hello java**

keyword    identifier

class    Sample                      // **class Declaration**

```
{  
    Public static void main ( String [ ] args )            // method Declaration  
    {  
        System.out.println ("Hello java");  
    }  
}
```

**Syntax for compilation**

Javac filename.java

**Ex:** Javac Sample.java

---

**Syntax for Interpretation**

Java filename

**Ex:** Java Sample

---

**Ex:** program to print integer , decimal , character , string and Boolean

```
class Sample
{
Public static void main ( String [ ] args )
{
System.out.println (20);
System.out.println (20.56);
System.out.println ('A');
System.out.println ("Java");
}
}
```

---

**Ex:** program to add two no

```
class Sample {
Public static void main ( String [ ] args ) {
System.out.println (20+20);
System.out.println ("The value is "+20);
System.out.println (20+" is the value");
System.out.println (20+20+"is the value");
```

---

```
System.out.println ("The value is " +20+20);  
System.out.println ("The value is " +(20+20));  
} }
```

---

## **VARIABLES**

Variable is a named memory location which is used to store some values or data and it can change N no of times during execution.

Variables are of two types

---

### **1. Primitive data types**

* byte	* short	* int	* long
* double	* float	* char	* Boolean

---

### **2. Non Primitive / references / class type**

* Arrays	* strings or any class type
----------	-----------------------------

---

### **Variable declaration**

Syntax: datatype variable \_ name;

**Ex:** int a;

---

### **Variable initialization**

Syntax: variable \_ name = values;

**Ex:** a = 10;

---

### **Variable utilization**

System.out.println(a);

o/p : a=10;

---

### **Variable declaration and initialization in a single value**

Syntax: datatype variable \_ name = values;



**Ex:** int a = 10;

---

### **Variable Re- initialization**

**Ex:** int y= 80;

int y= 100;

System.out.println(y);

o/p : y=100;

---

### **Copying the values from one variable to another**

**Ex:** int a = 20;

int b = a;

System.out.println(a);

System.out.println(b);

o/p : a=20;

o/p : b=20;

---

Variables are classified into

#### **1. Local variables**

- Any variables which is declared within the method is called as local variables.
- The scope of the local variables is from the beginning of the method till the end of the method or within the method.
- It cannot be classified into static and non – static.
- It will not have default values.
- Local variables should be initialized before utilization.

**Ex:** *class Sample {*

*Public static void main ( String [ ] args ) {*

*Int x=20;*

*System.out.println (x);*

```
} }
```

---

## **2. Global variables**

- Any variables which is declared outside method and inside the class is called as Global variables.
- The scope of the Global variables is from the beginning of the class till the end of the class.
- It can be classified into static and non – static.
- It will have default values.
- Once the global variables is declared immediately in the next line we cannot initialized or re – initialized , if so we will get compile time error .

**Ex:** *class Sample {*

*Static int a=20;*

*Public static void main ( String [ ] args ) {*

*System.out.println (a);*

*} }*

<b>Primitive data types</b>	<b>Default values</b>	<b>Size ( in bits)</b>	<b>Range</b>
byte	0	8	-128 to 127
short	0	16	-32,768 to 32,767
Int	0	32	-2 <sup>31</sup> to 2 <sup>31</sup> -1
long	0	64	-2 <sup>63</sup> to 2 <sup>63</sup> -1
Double	0.0d	64	1.4e-045 to 3.4e + 0.38
float	0.0f	32	4.9e -324 to 1.8e + 308
Char	'\v0000'(unique)	16	0 to 65,535
Boolean	False	1	True or false
String	Null	-	-

---

### **Difference b/w primitive data type**

Byte, short , int , long → Integer

Float , double , char , Boolean → Decimal

1 byte = 8 bits →  $-2^{n-1}$  to  $2^{n-1} - 1$

---

### **In a class , we have 3 members**

1. variables : it is used to store some data or values.
2. methods : it is used to perform some operations.
3. constructor : it is used to initialize variables.

---

Class class \_ name {

Variable / data members

Method / function members

Constructor

}

---

### **METHODS**

Method is a block of statements which will get executed whenever it is called or invoked.

#### **Syntax:**

<b><u>Access specifier</u></b>	<b><u>modifier</u></b>	<b><u>return type</u></b>	<b><u>method name</u></b>	<b><u>(arguments)</u></b>
Public	Static	void	identifier	[ optional ]
Private	Non – static	int		
Protected		double		
Package / default		char , String		
		Float, Boolean, class type		

---

#### **Syntax For arguments:**

Datatype variable \_ name

**Ex:** ( int a; ) ( int a , double b; )

---

### **Final variable :**

Any variable which is declared with a keyword final is called as final variable.

System.out.println(1/2); → 0 ( if both are integer)

System.out.println(1/2.0); → 0.5 ( if one of them is decimal)

**Ex:** *class Circle {*

```
    static void area() {  
        final double pi=3.142;  
        int r=4;  
        double result=pi*r*r;  
        System.out.println(result);  
    }  
    public static void main(String[] args) {  
        area();  
    } }
```

---

### **Method with parameter**

Whenever we want to give input for the method then we should go for method with parameter.

**Ex:** *class Circle {*

```
    static void area(int r) {  
        final double pi=3.142;  
        double result=pi*r*r;  
        System.out.println(result);  
    }
```

```
public static void main(String[] args) {  
    area(6);  
} }
```

---

### **Method with return type**

Whenever we want the result for further operation then we should go for method with return type.

**Ex:**    *class Circle {  
 static double area ( ) {  
 int r=5;  
 final double pi=3.142;  
 double result=pi\*r\*r;  
 return result;  
 }  
 public static void main(String[] args) {  
 double x=area();  
 System.out.println("area is "+x);  
 } }*

---

### **Conditional statements**

To check for logical conditions, we should go for conditional statements.

---

#### **1. If – conditions**

**Syntax :**    if ( condition )

```
{  
    -----  
    -----
```

```
}
```

**Ex:** *class Demo {*  
    *public static void main(String[] args) {*  
        *If (5>3)*  
    *{*  
        *System.out.println("hi");*  
    *}}*

---

## 2. If – else conditions

**Syntax :** if ( condition )

```
{  
-----  
} else {  
-----  
}
```

**Ex:** *class Demo {*  
    *public static void main(String[] args) {*  
        *If (5>30)*  
    *{*  
        *System.out.println("hi");*  
    *} else {*  
        *System.out.println("hello");*  
    *}}}*

---

## 3. Else - if conditions

**Syntax :** if ( condition )

```
    {  
        -----  
    } else if ( condition ){  
        -----  
    } else {  
        -----  
}  
}
```

**Ex:** *class Demo {  
 public static void main(String[] args) {  
 If (5>30)  
 {  
 System.out.println("hi");  
 } else if ( 5>2) {  
 System.out.println("hello");  
 } else {  
 System.out.println("cool");  
 }  
 }  
}*

---

#### **4. Nested if -Else conditions**

**Syntax :** if ( condition )  
 {  
 -----  
 } if ( condition ){  
 -----  
 } else {

```
-----  
} else {  
-----  
}
```

---

## **Loops**

Whenever the starting and ending range is given then we want to go for FOR loop.

**Syntax:**    For ( initialization ; condition ; increment (++) / decrement (--))  
                  {  
                  }

**Ex 1:**    *For ( int i=1 ; i<=4 ; i++)*

```
{  
    System.out.println("cool");    →   cool , cool , cool , cool  
}
```

**Ex 2:**    *For ( int i=1 ; i<=10 ; i++)*

```
{  
    System.out.println(i);    →   1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10  
}
```

**Ex 3:**    *For ( int i=10 ; i<=1 ; i--)*

```
{  
    System.out.println(i);    →   10 , 9 , 8 , 7 , 6 , 5 , 4 , 3 , 2 , 1  
}
```

**Ex 4: without using semicolon :**

```
For ( int i=1 ; i<=n ; System.out.println(i) )  
i++;
```



**Switch case :**

Switch case is used for pattern matching. The particular case will be executed based on the input.

**Syntax:**

```
Switch ( character)
{
    Case charseq : statement ;
                Break;
    Case charseq : statement ;
                Break;
    Default : statement ;
                Break;
}
```

```
Ex: class Demo {
    public static void main(String[] args) {
        Int input=3;
        Switch ( input )
    {
        Case 1 : System.out.println("FCD");
                Break;
        Case 2 : System.out.println("FC");
                Break;
        Default : System.out.println("Invalid input") ;
                Break;
    }
}}
```

## **Post increment and Pre increment**

### **1. Post increment – ( use it and then increment i++ )**

I = 1;

I = I++ + I++ I++ + I++

$$1 + 2 + 3 + 4 = 10$$

I = 1;

I = I++ + I + I++ + I++ + I + I++ + I++ + I++

$$1 + 2 + 2 + 3 + 4 + 4 + 5 + 6 + 7 = 27$$

I = 1;

I = I++ + I++ + I + I + I + I++ + I++

$$1 + 2 + 3 + 3 + 3 + 3 + 5 + 6 = 24$$

### **2. Pre increment – (increment ++i and then use it )**

I = 1;

I = ++I + I + ++I + ++I + I

$$2 + 2 + 3 + 4 + 4 = 15$$

I = 1;

I = ++I + ++I + ++I + ++I + I + ++I + I + ++I

$$2 + 3 + 4 + 5 + 5 + 6 + 6 + 7 = 38$$

## **In a class , we have 3 members**

1. variables

2. methods

3. constructor

- Variables and methods can be classified into static and non - static.
- Constructor is always non – static.

```
Class class _ name {  
Variable / data members  
Method / function members } static and non – static  
Constructor → non – static  
}
```

---

## **STATIC**

- Any member of the class declared with a keyword static is called as static member of the class.
- Static is always associated with class.
- Static is one copy.
- All the static members will be stored in static pool area.
- Whenever we want to access static members from one class to another class then we should use  
**Class \_ name . variable \_ name ( );    or    Class \_ name . method \_ name ( );**

**Note :** We can develop multiple classes in a single file.

- Whichever class is having main method that class file should be filename.
- For each and every class present in the file will have corresponding . class file .

---

### **Ex:** B / W the classes with a method as static

```
class Circle {  
    static void area() {  
        final double pi=3.142;  
        int r=5;  
        double result=pi*r*r;  
        System.out.println("result is" +result);  
    }  
}
```

```
class Tester {  
    public static void main(String[] args) {  
        Tester.Area();  
    } }  

```

---

**Ex:** B / W the classes with a method as static with parameter

```
class Circle {  
    static void area(int r) {  
        final double pi=3.142;  
        double result=pi*r*r;  
        System.out.println("result is" +result);  
    } }  
  
class Tester {  
    public static void main(String[] args) {  
        Tester.Area(5);  
    } }  

```

---

**Ex:** B / W the classes with a method as static with return type

```
class Circle {  
    static double area( ) {  
        int r=5;  
        final double pi=3.142;  
        double result=pi*r*r;  
        return result;  
    } }  
  
class Tester {  

```

```
public static void main(String[] args) {  
    double x= Circle. area();  
    System.out.println("area is "+x);  
}}
```

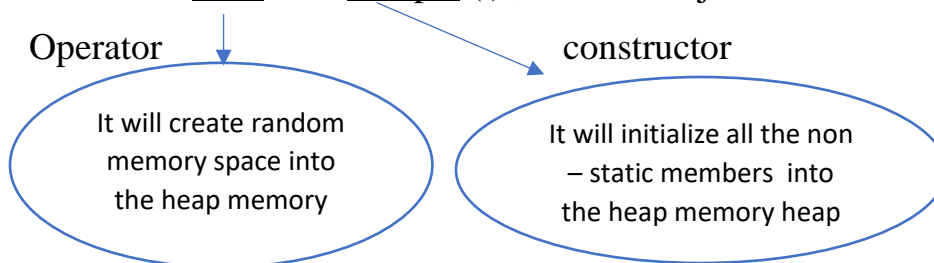
## NON – STATIC

- Any member of the class declared without a keyword static is called as Non - static member of the class.
- Non - Static is always associated with object.
- Non - Static is multiple copy.
- All the Non - static members will be stored in Heap memory.
- Whenever we want to access from Non – static to static then we should use **Object . variable \_ name or object . method \_ name**  
**Reference variable . variable \_ name or**  
**Reference variable . method \_ name**

**Syntax:**

```
New class _name ( ) ;
```

Ex:            New        Sample ( ) ;        → object



# JVM MEMORY

- Whenever class loader loads the class all the static members will get initialized in the static pool area.
- JVM starts executing from main method in the program the first statement is the object creation , equal operator will work from right to left.
- New operator will create a random memory space in the heap memory. Constructor will initialize all the non - static members into the heap memory

while creating the object itself we pass the arguments which will get initialized the constructor and from constructor it will get initialized to the object variable.

- Now in the main method the object address will be stored in the references variable `e1` and through that references variables we point the values.



It has 4 parts

- 1. stack** – It is used for execution which follows last in first out [ LIFO ] .
- 2. Heap memory** – It is used to store non – static members of the class. Whenever we create an object at the instance non – static members will be stored in the heap memory.
- 3. static pool area** - It is used to store static members of the class. Whenever class loader loads the class at that instance static members will get initialized in the static pool area.
- 4. method area** – It is used to store method body or definition . Irrespective of static or non – static all the method bodies will be stored in method area.

---

**Ex: Non – static to static without parameter and return type**

```
class Circle {  
    void area( ) {  
        final double pi=3.142;  
        int r=4;  
        double result=pi*r*r;  
    }  
}
```

```
        System.out.println(result);
    }
    public static void main(String[] args) {
        new Circle().area();
    }
}
```

---

**Ex:** Non – static to static with method as parameter

```
class Circle {
    void area(int r) {
        final double pi=3.142;
        double result=pi*r*r;
        System.out.println(result);
    }
    public static void main(String[] args) {
        new Circle().area(4);
    }
}
```

---

**Ex:** Non – static to static with method as return type

```
class Circle {
    double area( ) {
        final double pi=3.142;
        int r=4;
        double result=pi*r*r;
        return result;
    }
    public static void main(String[] args) {
```

```
Double z = new Circle().area();  
System.out.println(z);  
}}
```

---

**Ex:** B / W the classes with a method as non – static to static

```
class Tester {  
    static void area( ) {  
        final double pi=3.142;  
        int r=5;  
        double result=pi*r*r;  
        System.out.println("result is" + result);  
    }}  
class Circle {  
    public static void main(String[] args) {  
        Tester. area();  
    }}
```

---

## **REFERENCE VARIABLE**

Reference variable is a special type of variable which is used to store object address.

Reference variable can hold either null or object address.

---

### **Reference variable declaration**

**Syntax** : class \_ name reference \_ variable ;

**Ex** : int a ;

Tester t1 ;

---



**Reference variable initialization**

**Syntax** : reference \_ variable = object ;

**Ex** :            t1 = new Tester() ;

---

**Reference variable declaration and initialization**

**Syntax** : class \_ name reference \_ variable = object ; // homogenous type of object creation

**Ex** :    Tester            t1            =            new            Tester ( ) ;  
          ↓                    ↓                    ↓                    ↓  
          Class \_ name    reference variable    operator    constructor    → object

**Ex** : *Class Tester {*

*Void disp ( ) {*

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

*}*

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

*Tester t1 = new Tester ( );*

*t1.disp ( ) ;*

*} }*

---

**Ex**: Non – static to static through reference variable

*class Circle {*

*void area( ) {*

*final double pi=3.142;*

*int r=4;*

*double result=pi\*r\*r;*

*System.out.println(result);*

*}*

```
public static void main(String[] args) {  
    Circle C1 = new Circle();  
        C1.area();  
}}
```

---

**Note :** multiple objects can be stored in multiple references variable and if any changes made to an objects it will not affect other objects.

Ref1 ← address ← object

Ref2 ← address ← object

*Class Demo {*

*Int a=10;*

```
    public static void main(String[] args) {  
        Demo1 d1 = new Demo1( );  
            System.out.println(d1);  
        Demo1 d2 = new Demo1( );  
            System.out.println(d2);  
    }}
```

Output → Demo1 @ 15db9742 ,, Demo2 @ 06d69c

---

**Note :** Whenever we print the reference variable it print address i.e. fully qualified path.

Fully qualified path means package name.class\_name @ hexadecimal no

Ex : non - static . Demo @ 1bcfc76

Ref1 ← address ← object

Ref2 ←

*Class Demo {*

*Int a=10;*

---

```
public static void main(String[] args) {  
Demo1 d1 = new Demo1( );  
Demo1 d2 = d1;  
System.out.println(d1);  
System.out.println(d2);  
}}
```

Output → Demo1 @ 15db9742 ,, Demo2 @ 15db9742

---

### **To Understand Why Static And Non – Static**

```
Class training {  
Int java. mock;  
Static String ins_ name = “ Qspider”;  
Public static void main ( String [ ] args ) {  
Training t1 =new Training ( );  
t1.java. Mock=1;  
System.out.println(t1.java . mock);  
Training t2 =new Training ( );  
t2.java. Mock=2;  
System.out.println(t2.java . mock);  
t2.java. Mock=1;  
}}
```

---

### **COMPOSITION / AGGREGATIONS**

A class having an object of another class is called as composition.  
It is also called as has a relationship.

**Class diagram :**

It is a pictorial representation to represent the member of the class.

*Class Tester {*

*Void add ( ) {*

*System.out.println(“hii”);*

*}}*

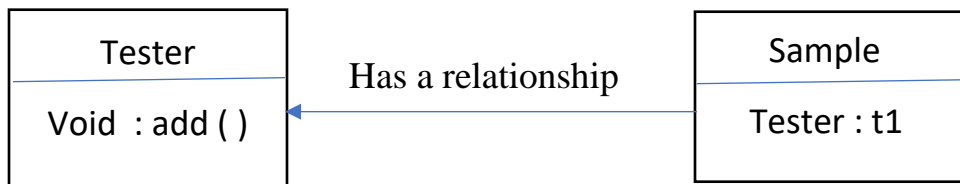
*Class Sample {*

*Public static void main ( String[] args) {*

*Tester t1= new Tester ( );*

*t1. Add ( );*

*}}*



---

**BLOCKS**

There are two blocks

---

**1. Static Initialization block ( SIB )**

- Any block which is declared with a keyword static is called as static initialization block .
- SIB is used to initialize static members.
- SIB will get executed before main method.
- We can have N no of SIB , the order of execution is sequential.

**Syntax :**

Static {

-----

```
}
```

---

```
Ex: Class Demo {  
Static int a;  
Static {  
a=10;  
}  
Public static void main ( String[] args) {  
System.out.println(a);  
}
```

Output : 10

---

## 2. Instance Initialization block ( IIB )

- Any block which is declared without a keyword static is called as Instance initialization block .
- IIB is used to initialize non - static members.
- IIB will get executed whenever an object is created.
- We can have N no of IIB , the order of execution is sequential.

**Syntax :**

```
{  
    -----  
}
```

---

```
Ex: Class Demo {  
{  
System.out.println(" ----IIB ---- ")  
}
```

```
Public static void main ( String[ ] args) {  
new Demo( );  
}}
```

Output :

-----IIB-----

---

### **PASS BY VALUE / CALL BY VALUE**

Calling or invoking a method by passing primitive type of data is called as call by value or pass by value.

```
Ex: Class Sample {  
Static void add ( int a ) {  
System.out.println(a) ;  
}  
Public static void main ( String [ ] args) {  
Int x=10;  
add ( x);  
}}
```

---

### **PASS BY REFERENCE / CALL BY REFERENCE**

Calling or invoking a method by passing references variables is called as call by references or pass by references.

```
Ex: Class Sample {  
Int y=80;  
Static void cool ( Sample s2 ) {  
System.out.println(s2.y) ;  
}  
Public static void main ( String [ ] args) {
```

```
Sample s1 = new Sample ( );  
System.out.println(s1.y) ;  
Cool (s1);  
}}
```

---

```
Ex: Class Amazon {  
Void product ( ) {  
System.out.println("product") ;  
}}  
Class Cust1{  
Static void need product (Amazon a2) {  
a2.product( );  
}}  
Class FedEx {  
Public static void main ( String [ ] args){  
Amazon a1 = new Amazon ( );  
Cust1.needproduct (a1);  
}}
```

---

## **CONSTRUCTOR**

It is a special type of method or special type (member) of the class which is used to initialize data members.

---

### **Rules :**

1. The constructor name should be same as class name.
2. Constructor will not have return type.
3. Constructor will not have return any value.

4. Constructor is always non static.
5. Whenever an object is created , constructor will get invoked.

---

**Syntax :**

```
Class class_name {  
Class_name ( ) {  
----  
Return;  
} }
```

---

**Ex: Class Demo {**  
**Demo ( ) {**  
**----      → constructor**  
**}}**

---

**Ex: Class Sample {**  
**Sample ( ) {**  
**System.out.println("hey I am constructor");**  
**Return;**  
**}**  
**Public static void main ( String [ ] args) {**  
**New Sample ( );**  
**}}**

---

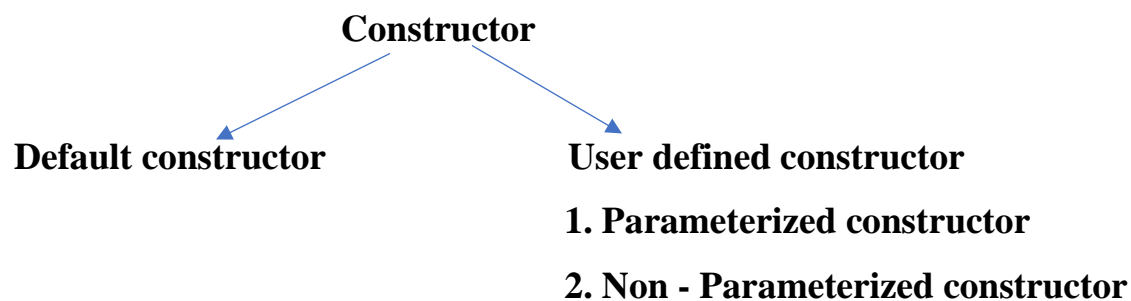
**Ex: write a program to initialize the variables through constructor**

```
Class Sample {  
Int x;
```



```
Sample ( int y ) {  
X = y ;  
}  
  
Public static void main ( String [ ] args) {  
Sample s1 = New Sample ( 10 ) ;  
System.out.println(s1.x) ;  
}}
```

---



---

**Ex: write a program to initialize employee id , employee name , employee salary through constructor**

```
Class Employee {  
Int emp _ id ;  
String emp _ name ;  
Double emp _ Sal ;  
Employee ( int x, String y , double z ) {  
emp _ id = x ;  
emp _ name = y;  
emp _ Sal = z ;  
}  
  
Public static void main ( String [ ] args) {
```

---

```
Employee e1 = New Employee ( 089, "Rakesh",45000 ) ;  
System.out.println(e1. emp _id ) ;  
System.out.println(e1. emp _ name ) ;  
System.out.println(e1. emp _ Sal ) ;  
}}
```

---

**Ex: write a program to initialize Student id , student name , student fees through constructor**

```
Class Student {  
Int std _id ;  
String std _name ;  
Double std _fees ;  
Student ( int x, String y , double z )  
{  
std _id = x ;  
std _name = y;  
std _fees = z ;  
}  
Public static void main ( String [ ] args)  
{  
Student s1 = New Student ( 089, "Rakesh",45000 ) ;  
System.out.println(s1. std _id ) ;  
System.out.println(s1. std _name ) ;  
System.out.println(s1. std _fees ) ;  
}  
}
```

**This keyword**

It is used to point to the current object whenever the local variable and global variable names are same to differentiate between them we use “ this “ keyword.

- This keyword should be used only in the non – static constructor.
- This keyword is the default references variables.

*Class Demo {*

*Int a =10 ;*

*Void add ( ) {*

*Int a= 20;*

*System.out.println (a);*

*System.out.println (this. a);*

*}}*

---

**Ex: write a program to initialize employee id , employee name , employee salary using this keyword**

*Class Employee {*

*Int emp \_ id ;*

*String emp \_ name ;*

*Double emp \_ Sal ;*

*Employee ( int x, String y , double z ) {*

*this. emp \_ id = x ;*

*this. emp \_ name = y;*

*this. emp \_ Sal = z ;*

*}*

*Public static void main ( String [ ] args) {*

*Employee e1 = New Employee ( 089, “Rakesh” ,45000 ) ;*

```
System.out.println(e1. emp _ id ) ;  
System.out.println(e1. emp _ name ) ;  
System.out.println(e1. emp _ Sal ) ;  
}}
```

---

**Ex: write a program to initialize Student id , student name , student fees using this keyword**

*Class Student*

```
{  
Int std _ id ;  
String std _ name ;  
Double std _ fees ;  
Student ( int x, String y , double z )  
{  
this. std _ id = x ;  
this. std _ name = y ;  
this. std _ fees = z ;  
}  
Public static void main ( String [ ] args)  
{  
Student s1 = New Student ( 089, "Rakesh" ,45000 ) ;  
System.out.println(s1. std _ id ) ;  
System.out.println(s1. std _ name ) ;  
System.out.println(s1. std _ fees ) ;  
}  
}
```

## **ARRAYS**

Array is linear data structure which is used to store homogeneous type of data.

### **Drawbacks:**

- In array the size is fixed
- it can store only homogeneous type of data.

Array

```
Int a=10;  Int b=20; Int c=30;
```

```
Int a=b;
```

---

### **Array declaration**

**Syntax:** Datatype [ ] array \_ name ;

Ex: int [ ] arr ;

---

### **Array size Initialization**

**Syntax:** array \_ name = new datatype [ size ] ;

Ex: arr = new int [5] ;

---

### **Store the values into an array**

**Syntax:** array \_ name [ index ] = values ;

Ex: arr [0] = 10 ;

```
arr [1] = 20 ;
```

```
arr [2] = 30 ;
```

```
arr [3] = 40 ;
```

---

### **Array Initialization**

```
System.out.println(arr [0]);
```

```
System.out.println(arr [1]);
```

```
System.out.println(arr [2]);  
System.out.println(arr [3]);  
System.out.println(arr [4]); → Array Index Out of Bound Exception
```

---

### **Array value Re - Initialization**

**Syntax:** array \_ name [ index ]= new values ;

Ex: arr [3] = 90 ;

---

**Note :** If the value is not stored in a particular index and if we try to print , we will get default values.

---

### **To find length of an Array**

**Syntax:** array \_ name . length ;

```
System.out.println(arr.length);
```

```
For(int I =0; I<arr.length ; I++)
```

```
{
```

```
System.out.println(arr[I]);
```

```
}
```

---

### **Array declaration and store the value directly**

**Syntax:** Datatype [ ] array \_ name = { v1 , v2 , v3 , v4 } ;

Ex: int abb [ ] = { 10 , 20 , 30 , 40 } ;

abb →

10	20	30	40	50
0	1	2	3	4

---

### **Copying the value of one Array to another Array**

```
int acc [ ] = abb ;
```

acc →

10	20	30	40	50
0	1	2	3	4

---

```
class Array{  
public static void main ( String [ ] args ) {  
    int[] arr = new int[3] ;  
    arr [ 0 ] = 10 ;  
    arr [ 1 ] = 20 ;  
    arr [ 2 ] = 30 ;  
    for ( int i=0; i<= arr . length ; i++ )  
    {  
        System.out.println(i+"\t"+ arr[i]);  
    } } }
```

---

```
class Array {  
public static void main ( String [ ] args ){  
    int[] arr = {10,20,30,40};  
    System.out.println("index\t values");  
    for ( int i=0; i<= arr . length ; i++ )  
    {  
        System.out.println(i+"\t"+ arr[i]);  
    } } }
```

---

```
Class Sample {  
Public static void main ( String [ ] args ) {  
    Char [ ] arr = new char [ 3 ] ;  
    arr [ 0 ] = ' A ' ;
```

```
arr [ 1 ] = ' B ' ;  
arr [ 2 ] = ' C ' ;  
System.out.println(" index \t values ");  
For ( int i=0; i<= arr . length ; i++ )  
{  
System.out.println(" i+ " \t" + arr [ i ] " ); // \t → tab space and \n → new line  
}}}
```

---

```
Class Sample {  
Public static void main ( String [ ] args ) {  
Boolean [ ] arr = { true , false , true , false } ;  
For ( int i=0; i<= arr . length ; i++ )  
{  
System.out.println(" i+ " \t" + arr [ i ] " );  
}}}
```

---

## **METHOD OVERLOADING**

Developing multiple methods with the same name but variation in argument list is called as method overloading.

### **Variation in argument list means :**

- Variation in the data type.
- Variation in the length of the arguments.
- Variation in the order of occurrence of the arguments.

### **Rules :**

- The method name should be same.
- There should be variations in the argument list.
- There is no restriction on access specifier, modifier and return type.



**Note :**

- We can overload both static and non - static method.
- We can overload main method.

```
Ex: class WhatsApp {  
    Void send ( int no ) {  
        System.out.println ( “ sending no” +no );  
    }  
    Void send ( String msg ) {  
        System.out.println ( “ sending msg” +msg );  
    }  
    Void send ( int no , String msg ) {  
        System.out.println ( “ sending no and msg ”+no+ “ “ +msg );  
    }  
    Void send (String msg , int no ) {  
        System.out.println ( “ sending msg and no ”+msg+ “ “ +no );  
    }  
}  
  
Class Mainclass {  
    Public static void main ( String [ ] args ) {  
        WhatsApp W1 = new WhatsApp ( );  
        W1.send (123);  
        W1.send (“hello”);  
        W1.send (126 ,”hi”);  
        W1.send (“bye”,127);  
    }  
}
```

```
Ex : class Book my show {  
    Static Void book ( int no _ seat ) {  
        System.out.println ( “book by no of seats” );  
    }  
    Static Void book( String movie _ name ) {  
        System.out.println ( “ movie name” );  
    }  
    Void book ( int no _seat , String movie _ name ) {  
        System.out.println ( “book by no of seats and movie name ”);  
    }  
    Void book (String movie , int no _seat ) {  
        System.out.println ( “movie name and book by no of seats”);  
    }  
}}  
Class Mainclass {  
    Public static void main ( String [ ] args ) {  
        Book my show b1 = new Book my show ( );  
        b1.book(5);  
        b1.book (“KGF”);  
        b1.book (3 ,”KGF”);  
        b1.book(“KGF”,2);  
    }  
}}
```

---

## **Class and Object**

Class is a blueprint or a template to create object .

Object is real time entity which has its own state and behaviour.

**State** : state defines the non – static variables , what data it can hold.

**Behaviour** : behaviour defines non – static methods and the way it can behave.

- Whenever we create an object we get both state and behaviour.
- The object address will be stored in reference variable.
- Multiple reference variable can hold multiple object address.

Ref1 ← address ← object

Ref2 ← address ← object

- Any changes made to the object through a reference variable, it will not affect other reference variables.
- Multiple reference variable can point ( hold ) single object address.
- Any changes made through a reference variable, it will affect other reference variables.

Ref1 ← address ← object

Ref2 ←

---

## **INHERITANCE**

Inheriting the property from one class to another class is called as Inheritance.

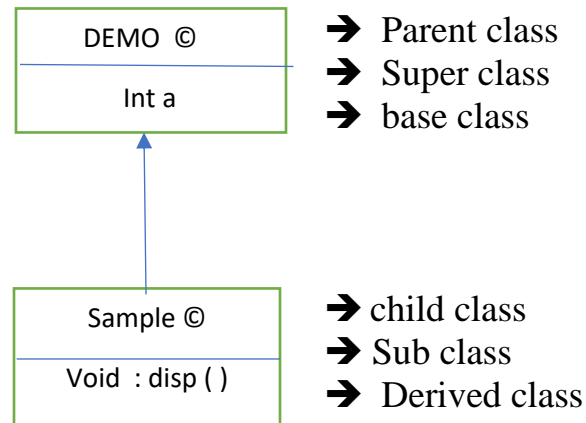
In Inheritance , we have 5 types :

1. single level Inheritance
2. multi - level Inheritance
3. hierarchical Inheritance
4. multiple Inheritance
5. hybrid Inheritance

---

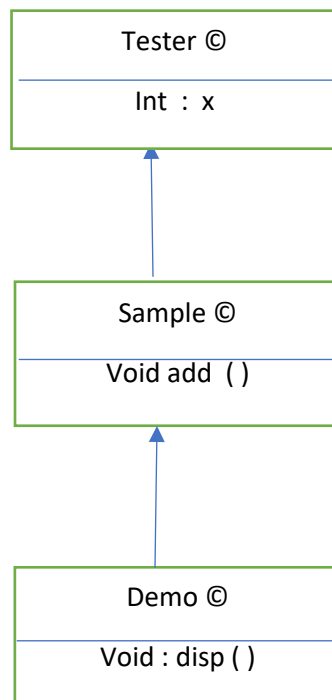
### **1. SINGLE LEVEL INHERITANCE**

A subclass inheriting the properties from only super class is called as **single level inheritance**.



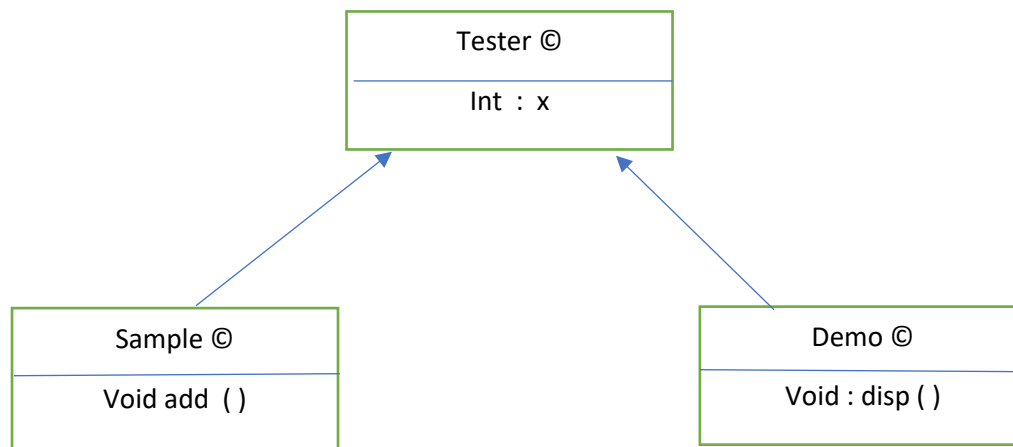
## 2. MULTI - LEVEL INHERITANCE

A subclass inheriting the properties from its super class intern super class inheriting the properties from its super class is called as **multi - level inheritance**.



## 3. HIERARCHICAL INHERITANCE

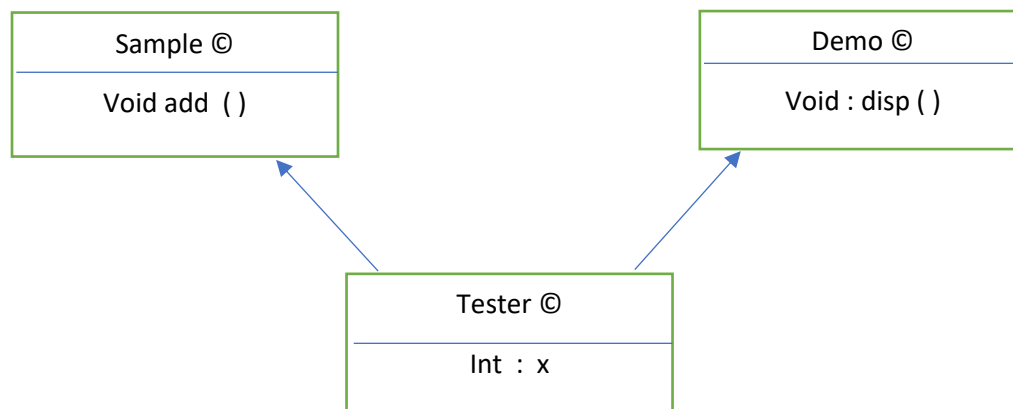
A multiple sub class inheriting the properties from only one super class or common super class is called as **hierarchical Inheritance**.



---

#### 4. MULTIPLE INHERITANCE

A sub class inheriting the properties from multiple super class is called as **Multiple Inheritance** .



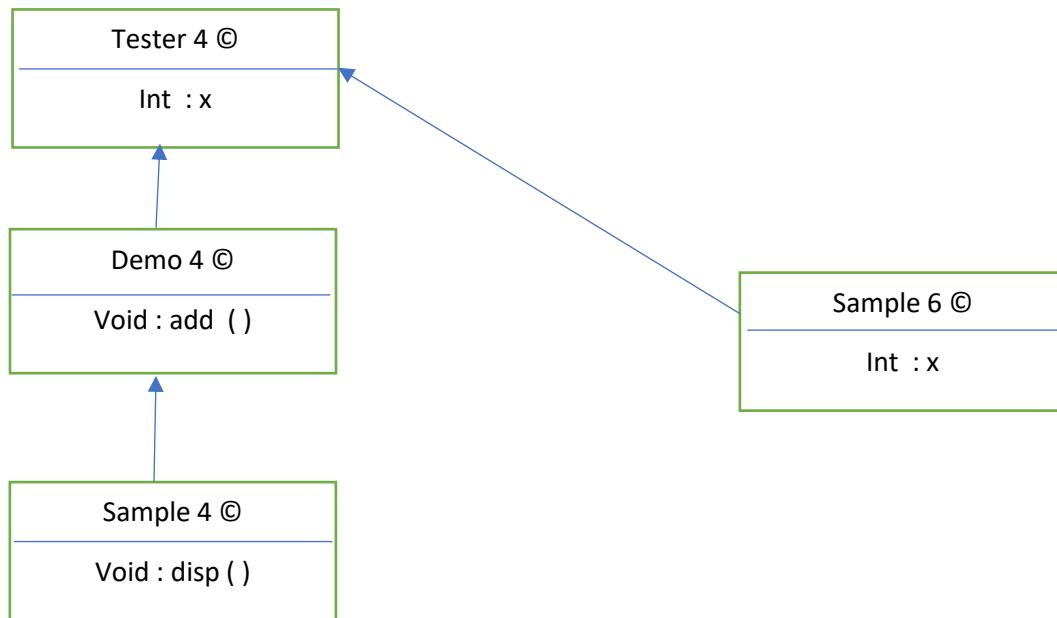
---

#### 5. HYBRID INHERITANCE

It is combination of single level , multi – level and hierarchical inheritance.

**When to go for inheritance ?**

- We go for inheritance for code reusability .



---

***Ex : single level inheritance***

***Class Demo {***

***Int a=10;***

***}***

***Class Sample extends Demo {***

***Void disp ( ) {***

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

***}}***

***Class Mainclass {***

***Public static void main ( String [ ] args ) {***

***Sample s1= new Sample ( );***

***S1.disp( );***

***System.out.println ( s1.a );***

***}}***

---

***Ex : multi level inheritance***

***Class Tester {***

***Int y=10;***

***}***

***Class Sample extends Tester {***

***Void add ( ) {***

***System.out.println ( “ hi “ );***

***}}***

***Class Demo extends Sample {***

***Void disp ( ) {***

***System.out.println ( “ hello “ );***

***}}***

***Class Mainclass {***

***Public static void main ( String [ ] args ) {***

***Demo d1= new Demo ( );***

***d1.disp ( );***

***d1.add ( );***

***System.out.println ( d1.a );***

***}}***

---

***Ex : hierarchical inheritance***

***Class Tester {***

***Int y=10;***

***}***

```
Class Demo extends Tester {  
Void add ( ) {  
System.out.println ( “ hi “ ) ;  
}}  
Class Sample extends Sample {  
Void add ( ) {  
System.out.println ( “ hi “ ) ;  
}}  
Class Mainclass {  
Public static void main ( String [ ] args ) {  
System.out.println ( “Demo object” ) ;  
Demo d3= new Demo ( ) ;  
}}
```

---

## **METHOD OVERRIDING**

Developing a method in the sub class with the same name and signature as in the superclass but with different implementation in the sub class is called as method overriding.

### **Rules :**

1. The method name and signature should be same in the sub class as in the super class.
2. There should be IS A RELATIONSHIP ( inheritance ).
3. The method be non – static.

---

```
Ex: Class WhatsApp _ v1 {  
Void status ( ) {  
System.out.println ( “status with text” ) ;
```



```
}}  
Class WhatsApp _v2 extends WhatsApp _v1 {  
Void status () {  
System.out.println ( "status with text , images ,videos " );  
}}  
Class Mainclass {  
Public static void main ( String [ ] args ) {  
WhatsApp w2 = new WhatsApp _v2 ( ) ;  
W2 . status ( ) ;  
}}
```

---

**Ex:**

```
Class KitKat {  
Void camera () {  
System.out.println ( "Back camera" ) ;  
}}  
Class lollipop extends KitKat {  
Void camera () {  
System.out.println ( "front and back camera" ) ;  
}}  
Class Mainclass {  
Public static void main ( String [ ] args ) {  
lollipop l1 = new lollipop ( ) ;  
l1 . camera ( ) ;
```

```
}}
```

---

## **SUPER KEYWORD**

Super keyword is used in the case of method overriding , along with the sub class implementation , if we need super class implementation thus we should go for **super . method name**

### **Note :**

- We go for inheritance for code reusability .
- We go for method overriding whenever we want to provide new implementation for the old feature .
- We go for method overloading whenever we want to perform common task and operation .

---

### **Ex :**

```
Class phonepe _ v1 {  
Void rewards ( ) {  
System.out.println (“ rewards by money”);  
}}  
  
Class phonepe _ v2 extends phonepe _ v1 {  
Void rewards ( ) {  
System.out.println (“ rewards by coupon”);  
Super . rewards ( ) ;  
}}  
  
Class Mainclass {  
public static void main ( String [ ] args ) {  
Phonepe _ v2 p2 = new phonepe _ v2 ( );  
P2 . rewards ( );
```

```
}}
```

---

### What is System.out.println ?

- System is a class.
- Out is a global static reference variables.
- Println is a non – static method of print stream.

```
Class Demo // print stream
```

```
{
```

```
Void add ( ) { // println
```

```
---
```

```
}}
```

```
Class Tester {
```

```
Static Demo d1 = new Demo ( );
```

```
Static Print stream = new print stream ( );
```

```
}
```

```
Class Mainclass {
```

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

```
Tester.d1.add();
```

```
System.out.println ( );
```

```
}}
```

---

### TYPE CASTING

Converting from one type to another type is called Type casting.

1. Primitive type casting

- Narrowing → explicitly
- Widening → → Implicitly , explicitly

2. Class type casting / Derived type casting

- Up casting → Implicitly , explicitly
- Down casting → explicitly

## 1. Primitive type casting

Converting from one primitive data type to another primitive data type is called Primitive Type casting.

**Widening** → Implicitly , explicitly

Converting from smaller primitive data type to any of its bigger primitive data type is called Widening Type casting.

Ex: Implicitly

```
1. double x = 20 ;  
   s.o.p (x) ; → o/p – 20.0d  
2. double y = 36.6f ;  
   s.o.p (y) ; → o/p – 36.6d  
3. float z = 40 ;  
   s.o.p (z) ; → o/p – 40.0f
```

Explicitly

```
1. double x = (double) 20 ;  
   s.o.p (x) ; → o/p – 20.0d  
2. double y = (double) 36.6f ;  
   s.o.p (y) ; → o/p – 36.6d  
3. float z = (float) 40 ;  
   s.o.p (z) ; → o/p – 40.0f
```

---

**Narrowing** → explicitly

Converting from bigger primitive data type to any of its smaller primitive data type is called Narrowing Type casting.

Explicitly

```
1. int x = (int) 59.9d;  
   s.o.p(x); → o/p – 59  
2. Short y = (short) 36.6l;  
   s.o.p(y); → o/p – 36  
3. byte z = (byte) 99.9f;  
   s.o.p(z); → o/p – 99  
4. Long t = (long) 60.36;  
   s.o.p(y); → o/p – 60
```

---

## 2. Class type casting / Derived type casting / Object type casting

Converting from one class object to another class type is called Derived / class type casting.

**Up casting** → Implicitly , explicitly

Converting from subclass object to superclass type is called as up casting.

- up casting can be done both Implicitly and explicitly.

**Down casting** → explicitly

Converting from superclass object to subclass type is called as Down casting.

- Down casting should be done always explicitly.
- Without performing up casting we cannot perform down casting.
- Direct down casting is not possible.
- To perform upcasting or down casting there should be a Is a relationship.

---

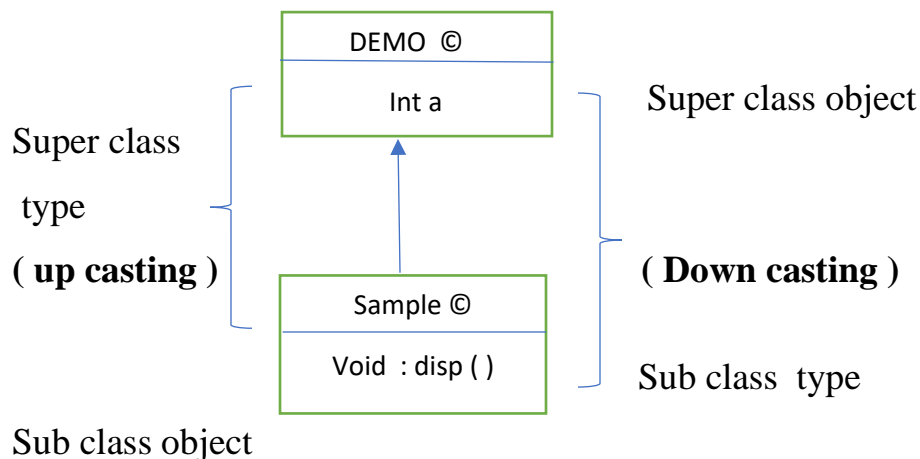
**Note :**

Sample s1 = new Sample ( ) ; // homogenous type of object creation.

Sample s1 = new Demo ( ) ; // heterogenous type of object creation.

In the case of method overriding even though it is upcasted we will get overridden implementation.

---



---

**Ex : Class Demo {**

```
Int a=10;  
}  
Class Sample extends Demo {  
Void disp ( ) {  
System.out.println("hey its disp..");  
}}  
Class Mainclass {  
Public static void main (String [ ] args) {  
Demo D1 = new Sample ( ) ;  
System.out.println(D1.a));  
Sample S1 = ( Sample ) D1 ;  
System.out.println(S1.a));  
S1.disp ( ) ;  
}}
```

---

## **POLYMORPHISM**

An object showing different behaviour at different stages of its lifecycle is called polymorphism.

Poly means “ many “ , morphism means “ forms “

In polymorphism , we have 2 types

1. compile time polymorphism
2. run time polymorphism

---

### **1. COMPILE TIME POLYMORPHISM**

The method declaration getting binded to its definition at the compile time by the compiler based on the arguments passed is called “ compile time polymorphism / static binding”.

Since the method declaration getting binded to its definition at the time itself is called as “early binding”.

Once the method declaration gets binded to its definition it cannot be rebinded, hence it is called “static binding”.

- Method overloading is an example for “compile time polymorphism”.
- At the compile time it's going to check for method declaration and its definition.

Ex: *class WhatsApp {*

*Void send ( int no ) {*

*System.out.println ( “ sending no” +no );*

*}*

*Void send ( String msg ) {*

*System.out.println ( “ sending msg” +msg );*

*}*

*Void send ( int no , String msg ) {*

*System.out.println ( “ sending no and msg ”+no+ “ “ +msg );*

*}*

*Void send (String msg , int no ) {*

*System.out.println ( “ sending msg and no ”+msg+ “ “ +no );*

*}}*

*Class Mainclass {*

*Public static void main ( String [ ] args ) {*

*WhatsApp w1 = new WhatsApp ( );*

*W1.send (123);*

*W1.send (“hello”);*

*W1.send (126 ,”hi”);*

```
W1.send ("bye",127);  
}}
```

---

## 2. RUN TIME POLYMORPHISM

The method declaration gets binded to its definition at the run time by the JVM based on the object created is called as “ run time polymorphism/ Dynamic binding “.

Since the method declaration gets binded to its definition , at the run time , hence it is called as “ late binding “.

Since the method declaration gets binded to its definition , at the rebinded , hence it is called as “ Dynamic binding “.

- Method overriding is an example for runtime polymorphism .

**Ex:**

```
Class Demo {  
Void cool ( ) {  
System.out.println ("hello java");  
}}  
  
Class Tester extends Demo {  
Void cool ( ) {  
System.out.println ("Hello java");  
}}  
  
Class Mainclass {  
Public static void main ( String [ ]args) {  
Demo d1 = new Tester ( );  
d1.cool ( );  
}}    o/p → Hello java
```



**Note :** In the case of method overriding even though it is upcasted we will get overridden implementation.

```
Animal a1=new Cat();
```

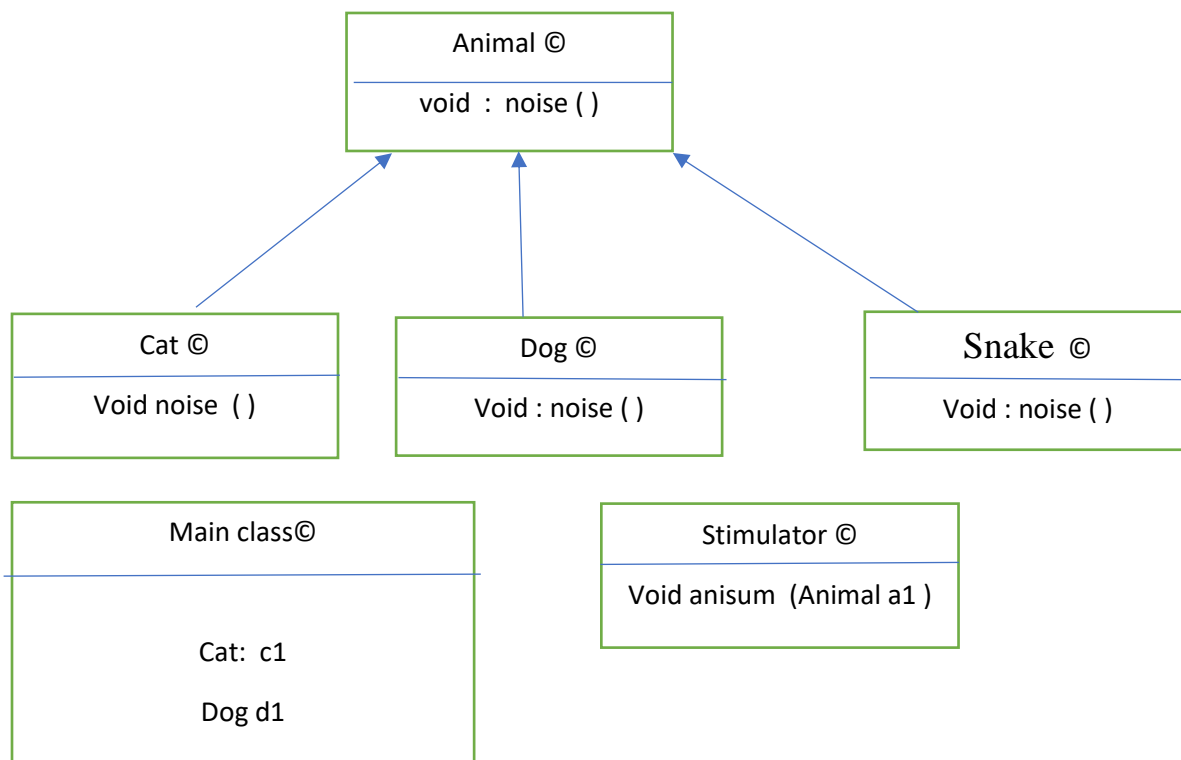
```
A1.noise();
```

```
Animal a1=new Dog();
```

```
A1.noise();
```

```
Animal a1=new Snake();
```

```
A1.noise();
```



**Ex:**

*Package poly;*

*Class Animal {*

*Void noise ( ) {*

```
System.out.println("some noise");
}}
Class Cat extends Animal {
Void noise ( ) {
System.out.println("meow meow...");
}}
Class Dog extends Animal {
Void noise ( ) {
System.out.println("bow bow...");
}}
Class Snake extends Animal {
Void noise ( ) {
System.out.println("buss tuss");
}}
Class Stimulator {
Static void anisum (Animal a1) {
a1.noise ( );
}}
Class Mainclass {
public static void main (String []args) {
Cat c1 = new Cat();
Dog d1 =new Dog();
Snake s1 = new Snake();
Stimulator. anisum (c);
Stimulator. anisum (d1);
```

*Stimulator. anisum (s1);*

*}*

*}*

---

### ***Why do we go for run time polymorphism ?***

To achieve 1. Single point of contact

2. loose coupling – If any internal enhancement is done it will not affect the usage.

3. Generalization – we can access everywhere.

---

## **ABSTRACT CLASS**

---

### **1. CONCRETE METHOD**

Any method which has both declaration and definition is called as “concrete method”.

Ex:

**Void disp () {**

**-----**

**}**

---

### **2. CONCRETE CLASS**

Any class which has only method is called as “concrete class”.

Ex:

**Class Sample {**

**Void disp () {**

**----**

**}**

```
}
```

---

### 3. ABSTRACT METHOD

Any method which is declared with a keyword abstract is called as “Abstract method”.

Ex:

```
Abstract void disp ( ) ;
```

---

### 4. ABSTRACT Class

Any class which is declared with a keyword abstract is called as “Abstract class”.

Ex:

```
Abstract class Demo {
```

```
-----
```

```
}
```

5. If a class is having an abstract method then the class should be declared as abstract but vice versa is not true.

Ex:

```
Abstract class Sample {
```

```
Abstract void disp ( ) ;
```

```
}
```

6. An abstract class can have both concrete method and as well as abstract method.

Ex:

```
Abstract void cool ( )
```

```
Void fool ( ) {
```

```
System.out.println (“hello”);
```

```
}}
```

- We cannot create object for abstract class and interface.
- We can develop only abstract methods in the abstract class.
- The class which provides the implementation for the abstract methods, the subclass is also called as implementation class.
- An abstract method cannot be declared as static , final , private.
- If any one of the abstract method is not overridden in the subclass, then the sub class should be declared as abstract.

***Ex 1: Abstract class Tester {***

***Abstract void disp ( );***

***Abstract void cool ( );***

***}***

***Class Demo extends Tester {***

***Void disp ( ) {***

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

***}***

***Void cool ( ){***

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

***}}***

***Class Mainclass {***

***Public static void main ( String [ ]args ) {***

***Demo d2 = new Demo ( );***

***d2.disp ( );***

***d2.cool ( );***

***}}***

---

***Ex 2:Abstract class Demo {***

```
Abstract void test ( );
Abstract void cool ( );
}
Class Tester extends Demo {
  Void test ( )
  {
    System.out.println("hi");
  }
  // Abstract void cool ( );
}
Class Sample extends Tester {
  Void cool ( ) {
    System.out.println("hello");
  }
  Public static void main ( String [ ]args ) {
    Sample s2 = new Sample ( );
      s2.test ( );
      s2.cool ( );
    }
  }
```

---

## **INTERFACE**

- An Interface is java type which is a pure abstract body.
- In Interface, we have 2 members i.e.
  - 1.variables
  - 2.method
- All the variables are by default static and final.
- All the methods are by default public and abstract.

- Interface does not support constructor.
- Java provides the keyword called implements to inherit the properties from interface to class.
- Interface is by default abstract.
- We cannot create object class and interface.
- Each and every class extends object class , object class is the super most class in the class type.
- Interface does not extends any class , interface itself is a super most type.
- From an interface to interface , to inherit the properties we use the keyword extends.
- The class which provides the implementation for the abstract method , the subclass is also called as implementation class.
- If any one of the abstract method is not overridden in the subclass then the subclass should be declared as subclass.
- Through Interface, we can achieve multiple inheritance and abstraction.
- Through abstract class, we can achieve up to 100% abstraction.
- Through interface we can achieve 100 % abstraction.

**Syntax:**

```
Interface interface _ name {  
Variable / Data members ;  
Method / Function members ;  
}
```

---

**Ex:**

```
( abstract ) interface Sample {  
Int a=10; // static final  
Void disp ( ); // abstract public  
}
```

---

```
Ex 1: abstract interface Tester {  
Public abstract void disp ( );
```

```
Public abstract void test ( );
}
Class Sample implement Tester {
Public void disp ( ) {
System.out.println("hi");
}
Public void test ( ) {
System.out.println("hello");
}}
Class Mainclass {
Public static void main ( String [ ] args) {
Sample s1 = new Sample ( );
s1.disp();
s1.test();
}}
```

---

```
Ex 2:Abstract interface Demo {
Public abstract void test ( );
Public abstract void cool ( );
}
abstract Class Tester implements Demo {
Public Void test ( ) {
System.out.println("hi");
}
// Public Abstract void cool ( );
```



```
}  
Class Sample extends Tester {  
Void cool ( ) {  
System.out.println("hello");  
}  
Public static void main ( String [ ]args ) {  
Sample s2 = new Sample ( );  
    s2.test ( );  
    s2.cool ( );  
} }
```

---

### **How abstract class can be used in real world ?**

*Ex 1 :*

```
Interface Audi {  
Void engine ( ) {  
Class AudiA4 implements Audi {  
Public void engine ( ) {  
System.out.println("5000 cc engine");  
}  
public void wheels ( ) {  
System.out.println("super wheels");  
}  
Public static void main(String [ ] args) {  
AudiA4 a1=new AudiA4 ( );  
a1.engine ( );
```

```
a1.wheel ( );  
}}
```

---

```
Ex 2: Abstract class Audi {  
Abstract void wheel ( );  
Abstract void engine ( );  
Void color ( ) {  
System.out.println("Black color");  
}}  
Class AudiA4 extends Audi {  
Void wheel ( ) {  
System.out.println("super wheels");  
}  
Void engine ( ) {  
System.out.println("5000 cc engine");  
}}  
Public static void main(String [ ] args)  
{  
AudiA4 a1=new AudiA4 ( );  
a1.color ( );  
a1.engine ( );  
a1.wheel ( );  
}  
}
```

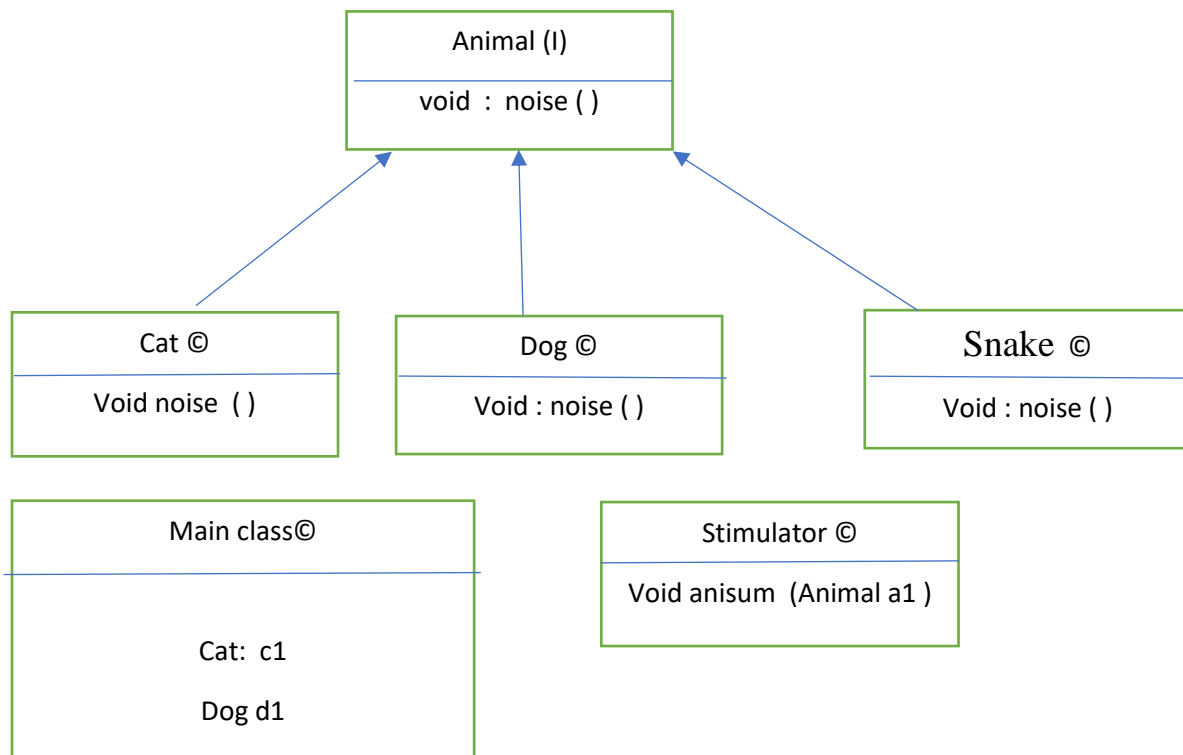
## **ABSTRACTION**

Hiding the complexity of the system and exposing only the required functionality to the end user is called as abstraction.

- To achieve abstraction , declare all the essential properties in the interface and provide the implementation in the sub class.
- Create reference variable of interface type and initialize that reference variable with the implementation class object. This is how we can achieve Abstraction.
- Through Interface we can achieve 100% Abstraction.
- Through abstract class we can achieve up to 100% Abstraction.

### **Note :**

- When we don't know 100% implementation then we should go for interface.
- When we know partial implementation, we go for abstract class.
- The above points are to justify when to go for abstract class and when to for interface.



**Ex:**

```
Interface Animal {  
Void noise ( )  
}  
  
Class Cat implements Animal {  
Void noise ( ) {  
System.out.println("meow meow...");  
}  
}  
  
Class Dog implements Animal {  
Void noise ( ) {  
System.out.println("bow bow...");  
}  
}  
  
Class Snake implements Animal {  
Void noise ( ) {  
System.out.println("buss tuss");  
}  
}  
  
Class Stimulator {  
Static void anisum (Animal a1)  
{  
a1.noise ( );  
}  
}
```

***Class Mainclass***

```
{  
public static void main (String []args)  
{  
Cat c1 = new Cat();  
Dog d1 =new Dog();  
Snake s1 = new Snake();  
Stimulator. anisum (c);  
Stimulator. anisum (d1);  
Stimulator. anisum (s1);  
}  
}
```

---

**PACKAGE**

It is a folder structure which is used to store similar kind of files. The packages should be created always in the reverse order.

Ex : [www.gmail.com](http://www.gmail.com) // commercial

Com → gmail → www

In any java file the 1<sup>st</sup> statement should be package statement , To import the files from 1 package to another package , java produces a keyword called “ import “ where the files will be present virtually.

Import statement should be 2<sup>nd</sup> statement .we can have n no of import statement file in a single java file. It can be any java type, i.e. : class, enum, annotation, interface .

Package movies; → 1

Import movies.kan movies action movies .kgf; → 2

Import movies.kan movies action movies .avn;

## Class Films

```
{    → 3  
}
```

---

In eclipse when we develop multiple classes which ever class is public that class that class is eligible to have main method and it should be file name.

Class C

```
{  
-----  
}
```

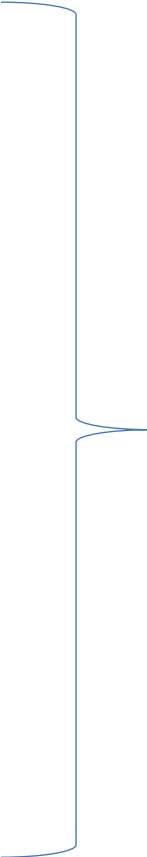
Public class A

```
{  
Public static void main ( String [ ] args )  
{  
-----  
}  
}
```

Class B

```
}  
  
}
```

A.java



---

## **ACCESS SPECIFIER**

Access specifier is used to restrict the access from one class to another class ( or ) from one package to another package.

In java , we have 4 access specifier

1. private

2. default / package level
3. protected
4. public

---

### **1. Private**

Any member of the class declared with a keyword `private` is called as private access specifier.

It can be accessed only within the class .

---

### **2. Default / Package Level**

There is no specific keyword specified for the access specifier. Then it is called as Default access specifier.

Or

If we declare any member of the class without any access specifier ( or ) keyword is called as default access specifier.

Default members can be accessed

1. within the class
2. within the package

---

### **3. Protected**

Any members of the class declared with the keyword `protected` is called as “protected member of the class “.

It can be accessed

1. within the class
2. within the package
3. outside the package with IS A RELATIONSHIP

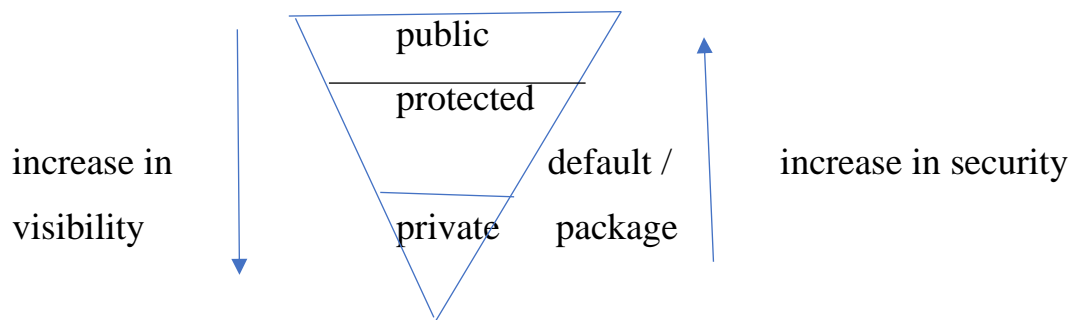
## 4. Public

Any members of the class declared with a keyword `public` is called as “public access specifier”.

It can be accessed

1. within the class
2. within the package
3. outside the package ( or ) any value.

**Note :** All the 3 members of the class can have all access specifier and the class can have only 2 access specifier i.e. : `public` and default .



---

**Note :** 1. We can inherit only non - static members.

2. constructor cannot be inherited.

3. variables and methods can be declared as final.

4. constructor cannot be declared as final.

---

```
package com.family.myfamily;
```

```
public class Father {
```

```
private void atm ()
```

```
{
```

```
    System.out.println("atm");
```

```
}
```

```
void car()
```



```
{  
    System.out.println("car");  
}  
protected void bike()  
{  
    System.out.println("bike");  
}  
public void cycle()  
{  
    System.out.println("cycle");  
}  
}
```

---

```
package com.family.myfamily;
```

```
public class Son {  
    public static void main(String[] args)  
    {  
        Father f1=new Father();  
        f1.car();  
        f1.bike();  
        f1.cycle();  
    }  
}
```

---

```
package com.family.unclefamily;  
import com.family.myfamily. Father;  
public class Aunty extends Father  
{  
  
    public static void main(String[] args) {  
        Aunty a1=new Aunty();  
        a1.bike();  
        a1.cycle();  
    }  
}
```

## **ENCAPSULATION**

It is one of the OOPS principles in java.

Java is by default encapsulated.

- We cannot declare any variable outside the class.
- We cannot have any print statement outside the method.
- Declare the data members as private and restricting the direct access outside the class and provide the indirect access through public services called GETTERS and SETTERS is called “ encapsulation”.
- GET is used to get the value.
- SET is used to set the value.
- It need not be get and set , it is an industrial convention to use get and set.
- Encapsulation is used for protection .

Ex :In ATM - a pin , net banking pwd and provide security and provide access to particular person.

---

### **What is java bean class ?**

Declare the data members as private and restrict the indirect access through public services is called as java bean class

---

**Ex 1 : Class Sample {**

***Private int a =10;***

***Public int get A ( )***

***{***

***return a ;***

***}***

***public void set A ( int a )***

***{***

***this. a =a ;***

***}***

```
}  
  
Class Mainclass {  
Public static void main ( String [ ] args) {  
Sample s1 = new Sample ( ) ;  
Int y = s1.get A ( ) ;  
System.out.println( y ) ;  
S1.set A (123);  
System.out.println( s1.get A ) ;  
System.out.println( s1.get A ) ;  
}}  

```

---

```
Ex 2 : Class ICICI {  
Private int set pin=1234;  
Public int atm pin ( )  
{  
return ;  
}  
Public void set A ( int atm pin )  
{  
this. atm pin = atm pin ;  
}  
}  
  
Class Mainclass {  
Public static void main ( String [ ] args) {  
ICICI card = new ICICI ( ) ;  

```

```
System.out.println( card . get atm pin ( ) ) ;  
Card . set atm pin (1267);  
System.out.println( card . get atm pin ( ) ) ;  
}  
}
```

```
Ex 3 : class Facebook {  
Private int pwd =1234;  
Public int get pwd ( )  
{  
return pwd ;  
}  
Public void set pwd ( int pwd )  
{  
this . pwd = pwd ;  
}  
}  
Public Class Mainclass {  
Public static void main ( String [ ] args) {  
Facebook f1 = new Facebook ( );  
System.out.println( f1. get pwd ( ) ) ;  
F1 . set pwd (1267);  
System.out.println( f1 . get pwd (1 ) ) ;  
}  
}
```

## **OBJECT CLASS**

Object class is the super most class in java .

Each and every class extends object class.

Object class belongs to java. Lang package , which will be imported by default.

In object class we have following methods as follows :

Int hashCode ( )

Boolean equals ( )

Object clone ( )

String toString ( )

Void notify ( )

Void notifyall ( )

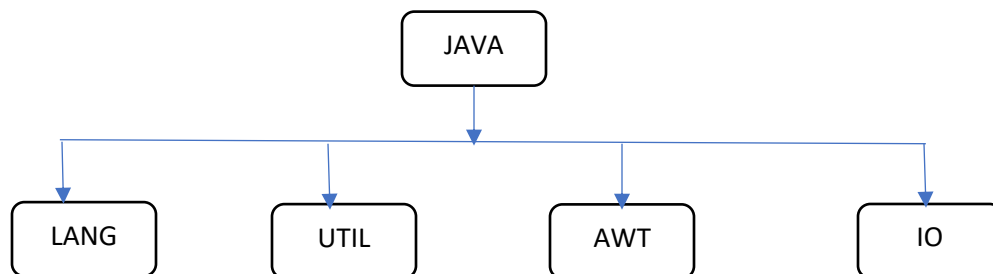
Void wait ( )

Void finalize ( )

---

## **JAVA LIBRARY**

Java library is nothing but inbuilt classes and interfaces , we have following libraries like



---

## **Tostring**

Tostring method is a non – static , non - final method of object class .

- It will be inherited to each and every class.
- Whenever we print the reference variable , toString method will be invoked implicitly , which will return fully qualified path in the form of string.

- Fully qualified path means package name .class name @ hexadecimal number.

**Ex :** *package poly ;*

```
Public class Demo extends object {  
Public static void main ( String [ ] args ) {  
Demo d1 = new Demo ( ) ;  
System.out.println( d1 . toString ( ) ) ;  
} }
```

---

### **HashCode**

HashCode method is a non - static , non - final method of object class.

- Hashcode method will be inherited to each and every class.
- Whenever we invoke the hashcode method , it will return the unique integer no called hash no based on the object address .
- The hash code will be generated using hashing algorithm the signature of hash code method is “ public int hash code “.
- Hash code method should be invoked explicitly.

**Ex :** *Public class Demo extends object {*

```
Public static void main ( String [ ] args ) {  
Demo d1 = new Demo ( ) ;  
System.out.println( d1 . hashCode ( ) ) ;  
Demo d2 = new Demo ( ) ;  
System.out.println( d2 . hashCode ( ) ) ;  
}  
} o / p ➔ 12367134
```

*56732193*

---

*package poly ;*

---

```
Public class Tester extends object {  
Public int hashCode ( )  
{  
return 123 ;  
}  
Public static void main ( String [ ] args ) {  
Tester t1 = new Tester ( ) ;  
System.out.println( t1 . hashCode ( ) ) ;  
} }
```

---

### **Equal Method**

Equal method is a non – static , non – final method of object class.

- Equal method will be inherited to each and every class.
- Equal method is used to compare object address .
- The signature of equal method is ‘public Boolean equals’.
- To compare the address, we use equal method.

**Ex :** *Public class Demo extends object {*  
*Public static void main ( String [ ] args ) {*  
*Demo d1 = new Demo ( ) ;*  
*Demo d2 = new Demo ( ) ;*  
*System.out.println( d1 .equals ( d2 ) ) ;*  
*}}*

---

### **STRING CLASS**

String is a final class which belongs to java. Lang package .

- It should be imported to each and every package.
- It is immutable. [ means it will not change the state of object ] .

```
String s1 = "hi";  
String s2 = "hello";
```

### **Why it is immutable ?**

When multiple reference variable are pointing to a single object and if one of the reference variable de reference with the current object it will not affect other reference variable. This is why it is immutable.

```
String s1 = " cool" ;      String s1 = " cool" ;  
String s1 = " cool" ;      String s1 = " hot " ;
```

---

### **String object can be created in 2 ways**

- 1. string object with new operations.**
- 2. string object without new operations.**

All the string object will be stored in string pool area which is under heap memory.

#### **string pool area is divided into 2**

- 1. constant pool area**
- 2. non - constant pool area**

- All the string object which is created without new operator will be stored in constant pool area.
- All the string object which is created with new operator will be stored in non - constant pool area.

***Package ABC;***

***Public class Tester extends object {***

***Public static void main ( String [ ] args ) {***

***String s1 = new String ( );***

***String s2 = new String ( );***

***System.out.println (s1==s2)***



```
System.out.println ("climax " + s1. Equals ( s2));  
String s3 = ("hi");  
String s4 = ("hi");  
System.out.println (s3==s4)  
}  
}
```

Difference between comparison operator and equal operator

### **Comparison Operator**

1. it is an operator .
2. operator cannot be overridden.
3. it compares address.

### **Equal Operator**

1. It is a non – static method of object.
2. Object can be overridden.
3. Compare object values in string class.

---

## **EXCEPTION HANDLING**

[ for smoother execution ]

It is an event triggered during the execution of the program which interrupt the execution and stops the execution abruptly / suddenly handling this event by using try and catch or throws is called as “Exception handling”.

- It is an abnormal statement which stops the program in order to this we have to go for try , catch or throws.
- All the abnormal statement should be developed in try block and address them in the catch block
- We cannot develop any statement between try and catch block.
- Once the exception occurs in the try block, further statement of the try block will not be executed.
- For one try block there should be one mandatory catch block.
- For one try block we can have more than one catch block but only 1 catch block will get executed.
- Exception class is the super class which can address both checked and unchecked exception.
- All exception classes belongs to java. Lang package.

- Once the exception is addressed, we cannot readdress if we try to do that then we get compile time error.
- Exception class can address all the sub class exception which is either compile time or run time.
- We can develop try , catch , finally or try finally.

An Exception can be address in 2 ways

1. try and catch

2. using throws

Exception are of 2 types

1. compile time exception or checked exception

2. run time exception or unchecked exception

---

### **1. Compile Time Exception ( checked exception )**

Any exception which is caught by the compiler at the compile time , this is called compile time exception or checked exception.

All the compile exceptions can be addressed by using try and catch or throws.

Compile time exceptions should be addressed at the compile time itself.

We can address compile time exception in 2 ways

1. try and catch

2. throws

*Class matrimony {*

*Static void submit ( ) throws matrimonyexception {*

*Int age =18 ;*

*If (age> =21) {*

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

*} else {*

```
Throws new matrimonyexception (“invalid age”);
}}
Public static void main ( String [ ] args ) {
Try {
Submit ( );
}
Catch ( matrimonyexception e ) {
System.out.println( e.getMessage( ));
}}
Class matrimonyexception extends exception {
String msg ;
Matrimonyexception ( String msg ){
This .msg =msg ;
}
Public String getMessage ( ) {
Return msg ;
}}
Class flipflipexception extends Exception {
String msg ;
Flipflipexception (String msg ) {
This.msg =msg;
}
Public string getMessage ( ) {
Return msg ;
}}
```

```
Public class flipkart {  
Static void purchase ( ) throws flipflipexception {  
Int pant =4500;  
If (pant >= 5000) {  
System.out.println(“great 200 rs discount “);  
} else {  
throws new flipflipexception (“no discount “);  
}}  
Public static void main ( String [ ] args ) {  
Try {  
Purchase ( );  
}  
Catch ( flipflipexception e ) {  
System.out.println(e.getMessage( ));  
}}}
```

---

## **2. Run Time Exception ( unchecked )**

Any exception which is not caught at the compile time and found at the run time is called as run time exception or unchecked exception.

Run time exceptions can be addressed by using try and catch or throws.

All the Run time exceptions should be addressed at the Run time itself.

We can address Run time exception in 2 ways

1. try and catch
2. throws

---

Syntax :

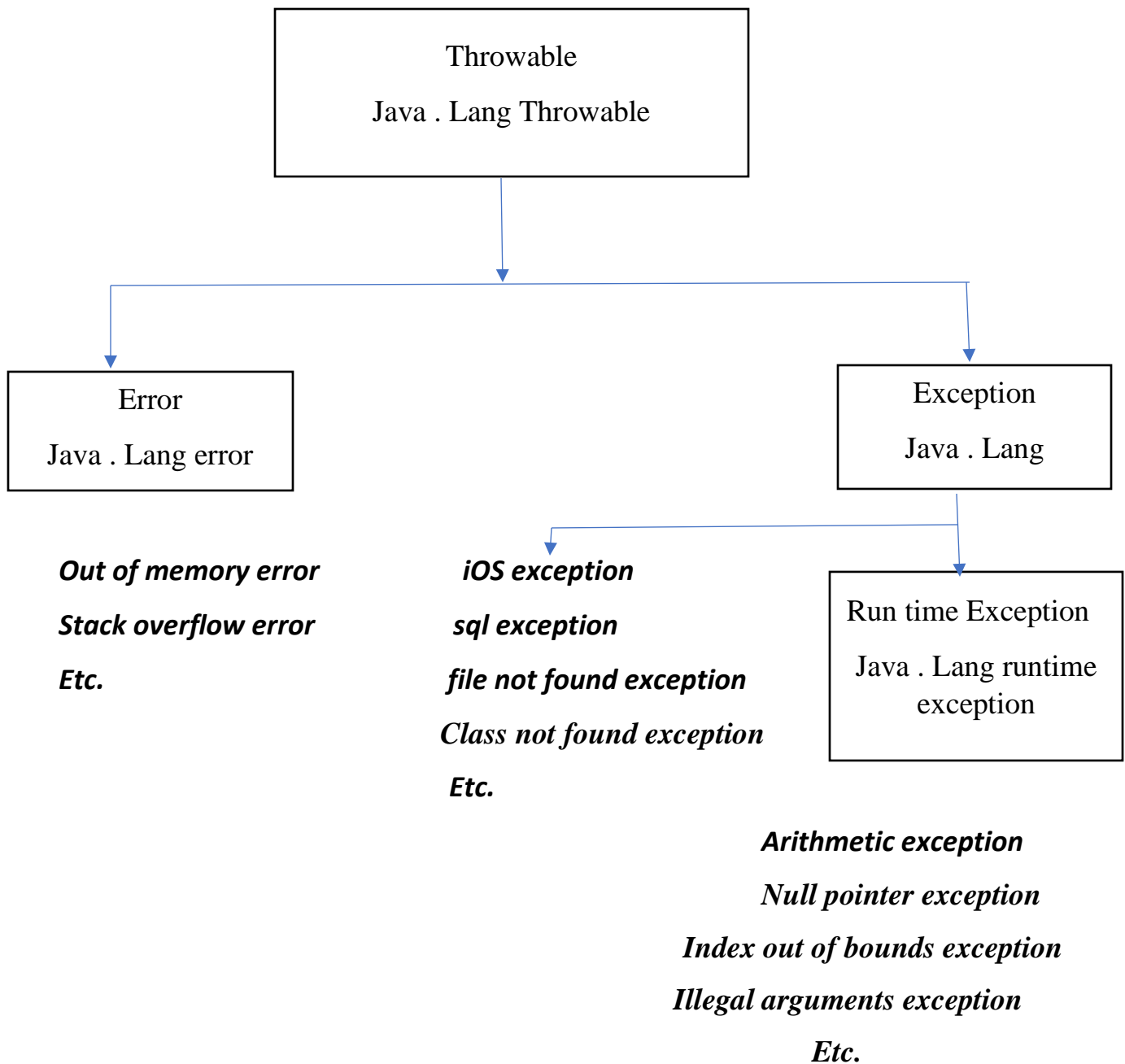
<b>1. try {</b> ---- } <b>Catch ( exception ref variable )</b> { ---- } 	<b>2. try {</b> ---- } <b>Catch ( )</b> { ---- } 
<b>3. try {</b> try { ---- } } <b>Catch ( )</b> { ---- } <b>Catch ( )</b> { ---- } 	<b>4. try {</b> ---- } <b>Finally {</b> ---- } 
<b>5. try {</b> ---- } <b>Catch ( )</b> { ---- } <b>Finally {</b> ---- } 	

### Throw :

It is used to throw the exception of throwable type.

It will be written internally.

Once exception is addressed if we try to readdress then “COMPILE TIME ERROR”.



---

**Ex 1:** `package exception topic {`  
    `Public class Sample {`  
        `Public static void main ( String [ ] args ) {`

```
    Try {  
Int I = 1 \ 10 ; || throw new arithmetic exception ( “ / zero )  
}  
Catch ( Arithmetic exception e ) {  
System.out.println(“ handled “);  
}  
} o / p → handled
```

---

```
Ex 2: Public class Sample {  
    Public static void main ( String [ ] args ) {  
Sample s1 = null;  
Try {  
System.out.println(s2.hashCode());  
}  
Catch ( Null pointer exception e ) // reference variables  
{  
    // [ it can be any letter but in indirectly we use e ]  
System.out.println(“ handled “);  
}} o / p → handled
```

---

**Note :** between try and catch nothing should be written if not compile time error .

---

```
Ex 3: Public class Sample {  
    Public static void main ( String [ ] args ) {  
Int [ ] arr = { 10 , 20 , 30 , 40 } ;  
Try  
{
```

```
System.out.println(" arr [ 5 ]");  
}  
Catch ( Array Index out of Bound exception e )  
{  
System.out.println(" handled ");  
}}
```

---

**Note :** for 1 try block we can develop multiple blocks .

---

*Ex 4: package exception topic {*

```
Public class Sample {  
Public static void main ( String [ ] args ) {  
Try  
{  
Int I = 1 \ 10 ; || throw new arithmetic exception ( "/ zero )  
}  
Catch ( Array Index out of Bound exception e )  
{  
System.out.println(" handled ");  
}  
Catch ( Arithmetic exception e )  
{  
System.out.println(" Addressed ");  
}} o / p → Addressed
```

---

- Compile time error occurs when we write something between try and catch block.



- There may be multiple catch for 1 try block but only one catch block will be executed .
- A reference variable can hold either null or object address

---

**Ex 5: Public class Sample {**

**Public static void main ( String [ ] args ) {**

**Try**

**{**

**Int I = 1 \ 10 ; // Arithmetic exception**

**Int [ ] arr = { 10 , 20 , 30 , 40 } ;**

**System.out.println(“ arr [ 5 ] “); // Index out of Bound**

**}**

**→ won't get executed**

**Catch ( Array Index out of Bound exception e )**

**{**

**System.out.println(“ handled “); // first exception will get executed**

**}**

**Catch ( Arithmetic exception e )**

**{**

**System.out.println(“ Addressed “);**

**}} o/p → Addressed**

---

**Ex 6: Public class Sample {**

**Public static void main ( String [ ] args ) {**

**Try**

**{**

**Int I = 1 \ 10 ;**

```
}  
Catch ( exception e )  
{  
System.out.println(“ handled “);  
}  
Catch ( Arithmetic exception e )  
{  
System.out.println(“ Addressed “);  
}  
} o / p → compile time error because of 2 exceptional cases
```

---

#### *Nested try catch*

- *Throw can throw only 1 exception object at a time.*
- *Throw is used to create the exception and it is developed in method body.*

---

```
Ex 7: Public class Sample {  
    Public static void main ( String [ ] args ) {  
        Try  
        {  
Int I = 1 \ 10 ;  
    }  
    Catch ( Array Index out of Bound exception e )  
    {  
System.out.println(“ handled “);  
    }  
    Catch (exception e )
```

```
{  
System.out.println(" Caught ");  
}  
} o / p → Addressed
```

---

### Finally Block

Finally block is block in exceptional handling which will get executed mandatory, even though the exception is addressed or not addressed.

We can develop finally block in 2 ways

1. try catch finally
2. try finally

➤ *In amazon 1 rupee sale , too many customer access the server and the server might not get crashed even through the application crashes , the database closing application connection statement should get executed , hence we have to develop finally block.*

---

```
Ex 8: Public class Sample {  
    Public static void main ( String [ ] args ) {  
        Try  
        {  
Int I = 1 \ 10 ;  
        }  
        Catch (Arithmetic exception e ){  
System.out.println(" Caught ");  
        }  
        Finally {  
System.out.println(" I am finally block ");
```

```
}  
}
```

---

## Stack Unwinding

If any exception occurs in any 1 of the method and if it is not addressed which will propagate the exception to the called method which in turn reaches main method and if main method propagated to JVM. Hence JVM does not have any handler (throw / try and catch) the stack will get destroyed . hence it is called as Stack unwinding.

---

## Print stack trace method

It is a non – static method of throwable class which will be inherited to each and every class.

It will print the complete back trace of the stack where the error message has occurred.

System.error - all the values and variables will be stored.

Print stack trace will help to find the root cause .

---

```
Class Demo {  
    Static void disp4 ( ) {  
        Int i =1\10;  
    }  
    Static void disp3 ( ) {  
        disp4 ( );  
    }  
    Static void disp2( ) {  
        Disp3 ( );  
    }  
    Static void disp1( ) {
```

```
Disp2 ( );  
}  
Public static void main ( String [ ] args ) {  
Try {  
disp1 ( );  
}  
Catch (Arithmetic exception e ) // throw new Arithmetic exception ( )  
{  
e.printStackTrace ( ); // = new Arithmetic exception ( )  
}  
}  
}
```

---

## **Error**

Error is a class which belongs to java. Lang package error is occurred due to the system configuration.

---

## **Difference b/w error and exception**

### **ERROR**

1. Error is unpredictable
2. Error is occurred due to system configuration.
3. Error can be handled.

---

### **EXCEPTION**

1. Exception is predictable.
2. Exception is occurred due to mistake done by the programmer.

3. exception can be handled.

---

## **Difference b/w throw and throws**

### **THROW**

1. Throw is used to throw the instance of throwable type.
2. Throw should be always developed in method body.
3. Throw is used to create the exception.
4. Throw can throw only 1 object at a time.

---

### **THROWS**

1. Throws is used to propagate the exception from 1 method to the called method.
2. Throws should be developed in method declaration.
3. Throws is used to propagate the exception.
4. Throws can propagate more than 1 exception

---

## **COLLECTIONS**

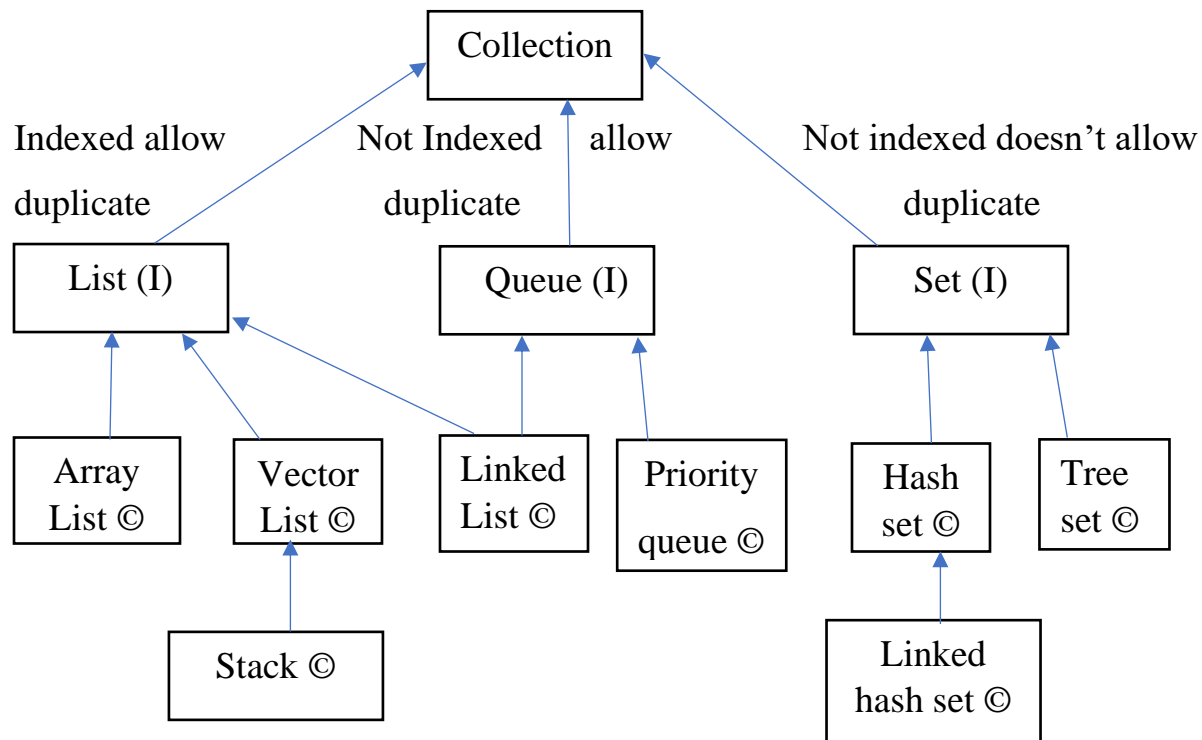
Collections is an unified architecture which consist of interface and class.

- All the collections ,class and interface belongs to java.util.package.
- In order to overcome the drawback of array we will go for collections. i.e.

1. In collection , the size is dynamic which it grows its size by 50% .
2. The default capacity will be 10.

$$\begin{aligned}
 \text{Capacity} &= \text{current capacity} * 3/2 + 1 \\
 &= 10 * 3/2 + 1 \\
 &= 16
 \end{aligned}$$

3. It can store heterogenous type of data



➤ In collection interface , we have 3 sub interfaces

1. List
2. Queue
3. Set

## 1. List

List is an interface which belongs to java.util.package

### **When you will go for list type of collection ?**

Whenever we want to store upon index and allow duplicates then we will go for list type of collection.

### **Features of list**

- Size is dynamic.
- We can store heterogenous type of data.

- It is indexed type of collection.
- It allows duplicates.
- It allows null.
- It will follow order of insertion.
- Since it is indexed type of collection we can do random access.

List interface have 4 sub classes i.e.

1. Array list
2. Vector list
3. Linked list
4. Stack

---

## 1. Array list

Array list is a class which implements list interface.

### Features of Array list

- Size is dynamic.
- We can store heterogenous type of data.
- It is indexed type of collection.
- It allows duplicates.
- It allows null.
- It will follow order of insertion.
- Since it is indexed type of collection we can do random access.
- It will grow its size by 50 %.
- It is not synchronized.
- Since it is not synchronized, the performance is fast.

***Package java.util;***

***Class ArrayList {***

***Void add ( object obj )***

***Void add ( int index , object obj )***

***Void addAll ( collection )***



```
Void addAll ( int index , collection )
Void remove ( )
Void removeAll ( )
Void retainAll ( )
Void get ( int index )
}
```

---

```
Ex : void add ( object obj )
Package collectiontopic ;
Import java.util. ArrayList;
Public Class Sample {
Public static void main ( String [ ] args ) {
ArrayList ll = new ArrayList ( );
ll. Add ( 10 ) ;
ll. Add ( 20.56 ) ;
ll. Add ( 'A' ) ;
ll. Add ( 11 ) ;
}} o/p → 10 , 20.56 , A , 11
```

---

```
Ex : void add ( int index ,object obj )
Package collectiontopic ;
Import java.util. ArrayList;
Public Class Sample {
Public static void main ( String [ ] args ) {
ArrayList ll = new ArrayList ( );
ll. Add ( 10 ) ;
```

```
l1. Add ( 20.56 );  
l1. Add ( 'A' );  
l1. Add ( 11 , " hello " );  
l1. Add ( 11 );  
}} o/p → 10 , 20.56 , A , 11 hello , 11
```

---

```
Ex : Package collectiontopic ;  
Import java.util. *;  
Public Class Sample {  
Public static void main ( String [ ] args ) {  
ArrayList l1 = new ArrayList ( );  
l1. Add ( 10 );  
l1. Add ( 20 );  
l1. Add ( 30 );  
ArrayList l2 = new ArrayList ( );  
l2. Add ( 'A' );  
l2. Add ( 'B' );  
l2. Add ( 'C' );  
System.out.println ( "---- A----");  
System.out.println ( " l1 → " +l1);  
System.out.println ( " l2 → " +l2);  
l1 . addAll ( l2 );  
System.out.println ( "---- B----");  
System.out.println ( " l1 → " +l1);  
System.out.println ( " l2 → " +l2);
```

*}} o/p → -----A-----*

*L1 → [ 10 , 20 , 30 ] l2 → [ A , B , C ]*

*-----B-----*

*L1 → [ 10 , 20 , 30 , A , B , C ] l2 → [ A , B , C ]*

---

*Ex : Package collectiontopic ;*

*Import java.util. \*;*

*Public Class Sample {*

*Public static void main ( String [ ] args ) {*

*ArrayList l1 = new ArrayList ( );*

*l1. Add ( 10 );*

*l1. Add ( 20 );*

*l1. Add ( 30 );*

*ArrayList l2 = new ArrayList ( );*

*l2. Add ( 'A' );*

*l2. Add ( 'B' );*

*l2. Add ( 'C' );*

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

*System.out.println ( " l1 → " +l1);*

*System.out.println ( " l2 → " +l2);*

*l1 . addAll ( l , l2 );*

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

*System.out.println ( " l1 → " +l1);*

*System.out.println ( " l2 → " +l2);*

*}} o/p → -----A-----*

---

$L1 \rightarrow [ 10 , 20 , 30 ] \quad l2 \rightarrow [ A , B , C ]$

-----B-----

$L1 \rightarrow [ 10 , A , B , C , 20 , 30 ] \quad l2 \rightarrow [ A , B , C ]$

---

### **Remove ( ) :**

It will help to remove the elements from the collections object ( it is used for string object)

**Ex : Package collectiontopic ;**

**Import java.util. ArrayList;**

**Public Class Sample {**

**Public static void main ( String [ ] args ) {**

**ArrayList l1 = new ArrayList ( );**

**l1. Add ( “ Bangalore “ ) ;**

**l1. Add ( “ Nelamangala “ ) ;**

**l1. Add ( “ Rajajinagar “ ) ;**

**System.out.println ( “ l1  $\rightarrow$  “ +l1);**

**l1 . remove ( “ Rajajinagar “ );**

**System.out.println ( “ l2  $\rightarrow$  “ +l2);**

**l1 . remove ( 0 );**

**System.out.println ( “ l3  $\rightarrow$  “ +l3);**

**}**

**} o / p  $\rightarrow$  L1  $\rightarrow$  [ Bangalore , Nelamangala , Rajajinagar ]**

**l2  $\rightarrow$  [ Bangalore , Nelamangala ] l3  $\rightarrow$  [ Nelamangala ]**

---

### **RetainAll ( )**

It will retain all the duplicates in l2 w.r.t l2

```
Ex : Package collectiontopic ;
Import java.util . ArrayList;
Public Class Sample {
Public static void main ( String [ ] args ) {
ArrayList l1 = new ArrayList ( );
l1. Add ( 10 ) ;
l1. Add ( 20 ) ;
l1. Add ( 30 ) ;
l1. Add ( 40 ) ;
ArrayList l2 = new ArrayList ( );
l2. Add ( 30 ) ;
l2. Add ( 40 ) ;
l2. Add ( 50 ) ;
l1. Add ( 60 ) ;
System.out.println ( "---- A----");
System.out.println ( " l1 → " +l1);
System.out.println ( " l2 → " +l2);
l1 . retainAll ( l2 );
System.out.println ( "---- B----");
System.out.println ( " l1 → " +l1);
System.out.println ( " l2 → " +l2);
}
} o / p → -----A-----
          L1 → [ 10 , 20 , 30 , 40 ]   l2 → [ 30 , 40 , 50 , 60 ]
          -----B-----
```

$L1 \rightarrow [ 30, 40 ]$     $l2 \rightarrow [ 30, 40, 50, 60 ]$

---

### **RemoveAll ( )**

It removes all the duplicates in l2 w.r.t l1 .

**Ex : Package collectiontopic ;**

**Import java.util . ArrayList;**

**Public Class Sample {**

**Public static void main ( String [ ] args ) {**

**ArrayList l1 = new ArrayList ( );**

**l1. Add ( 10 ) ;**

**l1. Add ( 20 ) ;**

**l1. Add ( 30 ) ;**

**l1. Add ( 40 ) ;**

**ArrayList l2 = new ArrayList ( );**

**l2. Add ( 30 ) ;**

**l2. Add ( 40 ) ;**

**l2. Add ( 50 ) ;**

**l1. Add ( 60 ) ;**

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

**System.out.println ( " l1  $\rightarrow$  " +l1);**

**System.out.println ( " l2  $\rightarrow$  " +l2);**

**l1 . removeAll ( l2 );**

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

**System.out.println ( " l1  $\rightarrow$  " +l1);**

**System.out.println ( " l2  $\rightarrow$  " +l2);**

**}} o/p → -----A-----**

**L1 → [ 10 , 20 , 30 , 40 ]    l2 → [ 30 , 40 , 50 , 60 ]**

**-----B-----**

**L1 → [ 10 , 20 ]    l2 → [ 30 , 40 , 50 , 60 ]**

**Object get ( int index )**

**Class Sample {**

**Public static void main ( String [ ] args ) {**

**ArrayList l1 = new ArrayList ( );**

**l1. Add ( 10 ) ;**

**l1. Add ( 20 ) ;**

**l1. Add ( 30 ) ;**

**l1. Add ( 40 ) ;**

**for ( int I =0 ; i< l1. Size ; i++ )**

**{**

**System.out.println ( l1. Get ( I ) );**

**}**

**}**

<b>Array</b>	<b>Collections</b>	<b>String</b>
Arr .length	L1. Size ( )	S1. Length ( )

**Collections** – it is a class which belongs to java.util.package , it has inbuilt methods like sort , search .

<b>Collections (c)</b>
Sort ( ) Binary search( )

```
Ex : Package collectiontopic ;  
Import java.util . ArrayList;  
Import java.util . collections;  
Public Class Sample {  
Public static void main ( String [ ] args ) {  
ArrayList l1 = new ArrayList ( );  
l1. Add ( 10 ) ;  
l1. Add ( 20 ) ;  
l1. Add ( 30 ) ;  
l1. Add ( 40 ) ;  
System.out.println ( “ l1 → “ +l1);  
Collections. Sort (l1) ;  
System.out.println ( “ l2 → “ +l2);  
}} o/p = l1 → [ 10 , 20 , 30 , 40 ] l2 → [ 30 , 40 , 210 , 410 ]
```

---

In ArrayList class the constructor is overloaded

1. ArrayList constructor without parameter. [ ArrayList ( ) ]

This constructor will be invoked . whenever an object is created which will initialize the default capacity as 10.

2. ArrayList ( int initial capacity ) : it will initialize the capacity as per the user input .

3. ArrayList ( collection c ) : it helps to copy from one collection to another collections object.

---

```
Ex : Package collectiontopic ;  
Import java.util . ArrayList;  
Import java.util . collections;
```



```
Public Class Sample {  
Public static void main ( String [ ] args ) {  
ArrayList l1 = new ArrayList ( );  
l1. Add ( 10 ) ;  
l1. Add ( 210 ) ;  
l1. Add ( 30 ) ;  
l1. Add ( 410 ) ;  
ArrayList l2 = new ArrayList ( l1 );  
System.out.println ( “ l1 → “ +l1);  
System.out.println ( “ l2 → “ +l2);  
}}
```

---

### Arrays :

Arrays is a class which belongs to java.util .package and it has inbuilt methods like sort , stream methods which helps to find maximum values.

*arlist ( ) – it is used to convert array to a list .*

```
class Array {  
static list arlist ( int [ ] acc )  
list l1 = new ArrayList ( );  
l1.add ( acc [0] ) ;  
l1.add ( acc [1] ) ;  
l1.add ( acc [2] ) ;  
return l1;  
}}
```

---

*Class Sample {*

---

```
Public static void main ( String [ ] args ) {  
Int [ ] abb = { 10, 20 , 30 } ;  
Array.arlist ( abb );  
list l2 = arlist ( );  
} }
```

---

```
Ex : Package collectiontopic ;  
Import java.util . ArrayList;  
Import java.util . Arrays;  
Import java.util . collections;  
Import java.util . list;  
Public Class Sample {  
Public static void main ( String [ ] args ) {  
Int [ ] arr = { 100, 200 , 300 } ;  
list l2 = array arlist ( arr );  
} }  
Class Sample {  
Static Sample disp ( ) {  
Sample x = new Sample ( );  
Return x ;  
}  
Public static void main ( String [ ] args ) {  
Sample y = disp ( );  
Int [ ] abb = { 10, 20 , 30 } ;  
Int [ ] acc = abb;
```

```
}}
```

---

## 2. Vector list

vector list is a class which implements list interface.

Default capacity of vector is 10.

### Features of vector list

- Size is dynamic.
- We can store heterogenous type of data.
- It is indexed type of collection.
- It allows duplicates.
- It allows null.
- It will follow order of insertion.
- Since it is indexed type of collection we can do random access.
- It will grow its size by 100 %.
- It is synchronized.
- Since it is synchronized, the performance is slow.

**Ex :** *Package collectiontopic ;*

*Import java.util . Vector;*

*Public Class Sample {*

*Public static void main ( String [ ] args ) {*

*Vector ll = new Vector ( );*

*ll. Add ( 10 ) ;*

*ll. Add ( 20.56 ) ;*

*ll. Add ( null ) ;*

*ll. Add ( 10 ) ;*

*System.out.println ( ll );*

*System.out.println ( “size → “ +ll.size);*

*System.out.println ( “ capacity → “ +ll.capacity);*

---

```
} }    o/p → [ 10 , 20.56 , null ,10 ] , size → 4 , capacity → 10
```

**Ex : Package collectiontopic ;**

**Import java.util . Vector;**

**Public Class Sample {**

**Public static void main ( String [ ] args ) {**

**Vector ll = new Vector ( );**

**ll. Add ( 10 ) ;**

**ll. Add ( 20.56 ) ;**

**ll. Add ( null ) ;**

**ll. Add ( 10 ) ;**

**ll. Add ( 'A' ) ;**

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

**System.out.println ( “size → “ +ll.size);**

**System.out.println ( “ capacity → “ +ll.capacity);**

```
} }    o/p → [ 10 , 20.56 , null , 10 , 'A' ] , size → 5 , capacity → 38
```

---

Arraylist	Vector
1. it increases its size by 50%.	1. it increases its size by 100%.
2. it is not synchronized.	2. it is synchronized.
3. since it is not synchronized the performance is fast.	3. since it is synchronized the performance is slow.

---

### 3. Linked list

**When you will go for linked list type of collection ?**

Whenever we want proper order of insertion , then we should go for linked list.

Linked list is a class which have dual property.

## Features of Linked list

- Size is dynamic.
- We can store heterogenous type of data.
- It is indexed type of collection.
- It allows duplicates.
- It allows null.
- It will follow order of insertion.
- Since it is indexed type of collection we can do random access.
- It inherits the properties from both lists and queue.
- It can use poll ( ) and peek ( ) method or get ( ) method to fetch the data.

**Ex:**

***Package collection topic;***

***Import java.util.LinkedList;***

***Public class Sample***

***{***

***Public static void main(String[ ] args)***

***{***

***LinkedList L1 = new LinkedList ( );***

***L1.add(10);***

***L1.add(20.26);***

***L1.add('A');***

***L1.add("hello");***

***L1.add(10);***

***System.out.println(L1);***

***System.out.println("after get ( ) → "+L1);***

***System.out.println(L1. peek ( ));***

***System.out.println("after peek ( ) → "+L1);***

```
System.out.println(L1.pool ( ));  
System.out.println("after pool ( ) → "+L1);  
}  
}      o/p → { 10 , 20.26 , A , hello , 10 }  
after get ( ) → { 10 , 20.26 , A , hello , 10 }  
10  
after peek ( ) → { 10 , 20.26 , A , hello , 10 }  
10  
after pool ( ) → { 10 , 20.26 , A , hello , 10 }
```

---

## 4. Stack

Stack is class which belongs to java.util.package

Stack extends vector class.

### Features of Stack

- Size is dynamic.
- We can store heterogenous type of data.
- It is indexed type of collection.
- It allows duplicates.
- It allows null.
- It will follow order of insertion.
- Since it is indexed type of collection we can do random access.
- It follows “last in first out”.

### *For – each loop*

If we want to traverse set which is not indexed , then we should go for “for each loop” .

### *Syntax:*

*For ( array\_type variable name : array name ( or ) collection reference \_ variable )*

```
{  
----  
}
```

---

```
Ex: String [ ] arr = { "hi", "hello ", "cool " };  
    For (String a : arr )  
    {  
        System.out.println( a );  
    }
```

---

```
Package collection topic;  
Import java.util. TreeSet;  
Public class Sample  
{  
Public static void main(String[ ] args)  
{  
TreeHashSet L1 = new TreeHashSet ( );  
L1.add(100);  
L1.add(10);  
L1.add(160);  
L1.add(10);  
If (l1.contains(160)  
{  
System.out.println("it contains 160");  
} else {
```

```
System.out.println("it doesn't contain 160");  
}  
For (object o1 : l1 )  
{  
System.out.println(o1);  
}} o / p → it contains 160 [ 10 , 100 ,160 ]
```

---

## **2. Queue**

Queue is an interface which extends collection interface.

In queue interface we have 2 sub classes

1. Linked list
2. priority queue

---

### **1. Priority Queue**

Priority queue is a class which implements queue interface which is of pure queue.

#### **Features of Priority queue**

- Size is dynamic.
- We can store heterogenous type of data.
- It is not indexed.
- It allows duplicates.
- It is auto sorted.
- Since it is not indexed type of collection, we cannot fetch the elements upon index.
- In priority queue to fetch the element we should use poll ( ) and peek ( ).

#### **Poll ( ) method**

Poll ( ) method is non – static member of the queue class , which helps to fetch the top most element of the queue and remove the element from the queue and reduce the size of the queue by 1.



**Peek ( ) method**

Peek ( ) method is non – static method of the priority queue class , which helps to fetch the top most element of the queue and it will not reduce the size of the queue by 1.

*Package collection topic;*

*Import java.util. PriorityQueue;*

*Public class Sample*

*{*

*Public static void main(String[ ] args)*

*{*

*Priorityqueue L1 = new PriorityQueue ( );*

*L1.add(100);*

*L1.add(20);*

*L1.add(16);*

*L1.add(10);*

*L1.add(80);*

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

*System.out.println(L1);*

*System.out.println(L1.poll ( ));*

*System.out.println(L1);*

*System.out.println(L1.poll ( ));*

*System.out.println(L1);*

*System.out.println(L1.poll ( ));*

*System.out.println(L1);*

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

```
System.out.println(L1);
System.out.println(L1.peek ( ));
System.out.println(L1);
System.out.println(L1.peek ( ));
System.out.println(L1);
System.out.println(L1.peek ( ));
System.out.println(L1);
} }
```

O / p --> -----poll -----	-----peek-----
[ 10 , 16 , 20 , 100 , 80 ]	80
10	[ 80 , 100 ]
[ 16 , 80 , 200 , 20 ]	80
16	[ 80 , 100 ]
[ 20 , 80 , 100 ]	80
20	[ 80 , 100 ]
[ 80 , 100 ]	80
	[ 80 , 100 ]

---

### **3. Set**

Set is an interface which extends collection interface.

#### **When do we go for set type of collection ?**

Whenever we want to store the element not upon index and not allowing duplicates, then we should go for set type of collection.

Inside interface we have 3 sub classes

1. Hash set

2. Linked hash set

3. Tree set

---

## 1. Hash set

Hash set is a class which implements set interface.

### Features of Hash set

- Size is dynamic.
- We can store heterogenous type of data.
- It is not indexed type of data.
- It will not allow duplicates.
- It allows null.
- Since it is not indexed type of collection, we cannot do random access.
- It will not follow order of insertion.

*Package collection topic;*

*Import java.util. HashSet ;*

*Public class Sample*

*{*

*Public static void main(String[ ] args)*

*{*

*HashSet L1 = new HashSet ( );*

*L1.add(10);*

*L1.add(20.56);*

*L1.add('A');*

*L1.add(10);*

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

*System.out.println(L1);*

*} } o/p → [ A, 20.56 , 10, hello ]*

## 2. Linked hash set

Linked hash set is a class which extends hash set.

### Features of Linked Hash set

- Size is dynamic.
- We can store heterogenous type of data.
- It is not indexed type of data.
- It will not allow duplicates.
- It allows null.
- Since it is not indexed type of collection, we cannot do random access.
- It will follow order of insertion.

*Package collection topic;*

*Import java.util. LinkedHashSet ;*

*Public class Sample*

*{*

*Public static void main(String[ ] args)*

*{*

*LinkedHashSet L1 = new LinkedHashSet ( );*

*L1.add(10);*

*L1.add(20.56);*

*L1.add('A');*

*L1.add(10);*

*L1.add ("hello");*

*System.out.println(L1);*

*} } o/p → [ 10 , 20.56 , A, hello ]*

---

## 3. Tree set

Tree set is a class which implements set interface.

### **Features of Tree Hash set**

- Size is dynamic.
- We can store heterogenous type of data.
- It is not indexed type of data.
- It will not allow duplicates.
- It allows null.
- Since it is not indexed type of collection, we cannot do random access.
- It is complexity auto sorted.

### ***Public class Sample***

```
{  
Public static void main(String[ ] args)  
{  
TreeHashSet L1 = new TreeHashSet ( );  
L1.add(100);  
L1.add(10);  
L1.add(16);  
L1.add(10);  
System.out.println(L1);  
}}  o/p → [ 10 , 16 ,100 ]
```

---

## **MAP**

Map is an interface , which belongs to java.util package.

Whenever we want to store upon key and value then we should go for MAP.

### **Features of Map**

- Size is dynamic.
- It can store heterogenous type of data.
- It stores the elements upon keys and values.

- It cannot have duplicates keys.
- We can have duplicates Values.

Map has 3 sub classes

1. Hash map
2. Linked Hash map
3. Tree map

---

## 1. Hash map

Hash map is a class which implements map interface.

### Features of Hash Map

- Size is dynamic.
- It can store heterogenous type of data.
- It stores the elements upon keys and values.
- It cannot have duplicates keys.
- We can have duplicates Values.
- It will not follow order of insertion.

*Ex:*

*Package intro;*

*Import java.util.HashMap;*

*Public class Sample*

*{*

*Public static void main(String[] args)*

*{*

*HashMap <String, Integer> L1= new HashMap <String, Integer> ( );*

*L1.put("virat",123);*

*L1.put("Rakesh",321);*

*System.out.println(L1);*

```
}  
} o/p → { virat 123 , Rakesh 321 }
```

---

## 2. Linked Hash map

Hash map is a class which implements map interface.

### Features of Linked Hash Map

- Size is dynamic.
- It can store heterogenous type of data.
- It stores the elements upon keys and values.
- It cannot have duplicates keys.
- We can have duplicates Values.
- It will follow order of insertion.

**Ex:**

**Package intro;**

**Import java.util.LinkedHashMap;**

**Public class Sample**

**{**

**Public static void main(String[] args)**

**{**

**Linked HashMap <String, Integer> L1= new Linked HashMap <String, Integer> ( );**

**L1.put("virat",123);**

**L1.put("Rakesh",321);**

**System.out.println(L1);**

**}**

**} o/p → { virat 123 , Rakesh 321 }**

### 3. Tree map

Tree map is a class which implements map interface.

#### Features of Linked Hash Map

- Size is dynamic.
- It can store heterogeneous type of data.
- It stores the elements upon keys and values.
- It cannot have duplicate keys.
- We can have duplicate values.
- It is completely auto sorted based on key.
- It sorts based on ASCII values.

---

*Ex:*

*Package intro;*

*Import java.util.Treemap;*

*Public class Sample*

*{*

*Public static void main(String[] args)*

*{*

*Tree Map <String, Integer> L1= new Tree Map <String, Integer> ( );*

*L1.put("virat",123);*

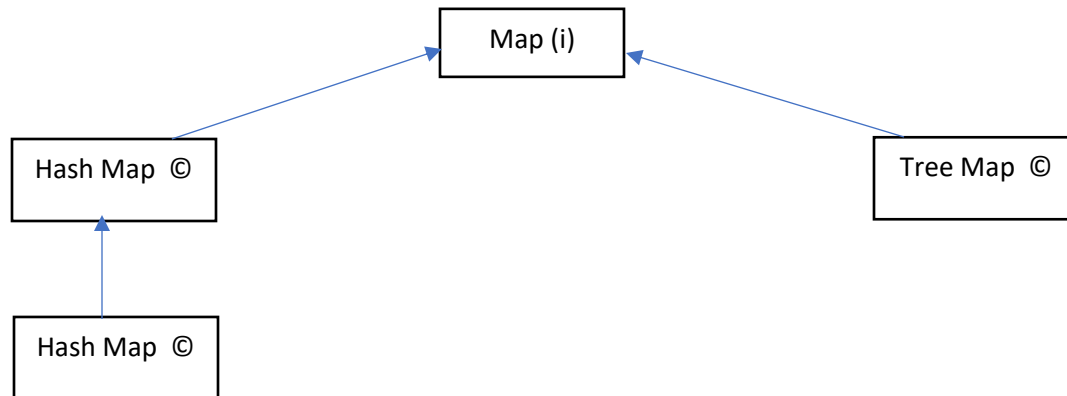
*L1.put("Rakesh",321);*

*System.out.println(L1);*

*}*

*} o/p → { virat 123 , Rakesh 321 }*





---

### **Generic class**

To achieve generic class, we have to use angular braces `< >`, by using generics we can store homogenous objects of a specified class type.

Map is of generic type by default.

#### **Primitive Datatype**

byte  
short  
int  
long  
float  
double  
char  
boolean

#### **Wrapper Datatype**

Byte  
Short  
Integer  
Long  
float  
double  
character  
Boolean

---

### **THREAD**

Thread is an extension instance which owns of its own CPU time and memory.

## Deamon thread

The thread which are running at backend is called as Deamon thread.

---

## Marker interface ?

An empty interface is called as marker interface

Ex: interface A

```
{  
---  
}
```

---

## Functional thread ?

Any interface which has only abstract method is called as functional interface.

Ex: interface A

```
{  
Void disp ( );  
}
```

Thread is a class which belongs to java. Lang package has many inbuild methods like sleep methods.

- Thread pause the execution in milliseconds.

---

## SLEEP METHOD

Sleep method is a static method of thread class which belongs to java.lang package , where it pauses the execution for few milliseconds.

Ex: *Package threadtopic;*

*Public class Sample {*

*Public static void main ( String [ ] args ) {*

```
For( int i=0; i<=10; i++ )  
{  
System.out.println ( I );  
Thread. Sleep (1000);  
}}}
```

---

## **MULTI THREADING**

Processing multiple threads simultaneously is called as multi - threading.

---

## **MULTI TASKING**

It is a Process of executing multiple task simultaneously is called as multi - tasking.

---

## **SYNCHRONIZED**

Processing one by one thread is called as Synchronized .

Processing multiple threads simultaneously is called as “non – Synchronized “.

---

### **Write a program to non – Synchronized multi - threading**

```
Package threadtopic;  
class Sample extends thread {  
Public void run ( )  
{  
For( int i=0; i<=10; i++ )  
{  
System.out.println ( I );  
Try  
{  
Thread. Sleep (1000);
```

```
}  
class Demo implements Runnable  
{  
Public void run ( )  
{  
For( int i=100; i<=110; i++ )  
{  
System.out.println ( I );  
Try  
{  
Thread. Sleep (2000);  
}  
Catch (InterruptedException e)  
{  
e.printStackTrace ( );  
}  
}}}  
Class Mainclass {  
Public static void main ( String [ ] args ) {  
Sample s1 = new Sample ( );  
Thread th1 = new Thread ( s1 );  
th1.start ( );  
Demo D1 = new Demo ( );  
Thread th2 = new Thread ( D1 );  
Th2.start ( );
```

```
}}
```

o/p → 1 100 2 102 3 4 .....

---

### **Thread Class Have 3 Major Properties**

**1. Thread priority :** the default priority will be five .

minimum is one and maximum is ten .

The priority can be send by the users.

**2. thread ID :** Its ID will be generated by system.

**3. thread name :** It is the name given for the thread by the user. The thread name will be by default main.

Main method is default from main thread.

**Current thread** is the static method of thread class which helps to return the instance of currently executing thread .

*Package threadtopic;*

*Public class Mainclass {*

*Public static void main ( String [ ] args ) {*

*Thread th1 = new . current thread );*

*System.out.println ( th1. get name ( ) );*

*System.out.println ( th1. get priority ( ) );*

*System.out.println ( th1. get Id ( ) );*

*th1.set name(“Rakesh thread”);*

*th1. Set priority ( );*

*System.out.println ( th1. get name ( ) );*

*System.out.println ( th1. get priority ( ) );*

*System.out.println ( th1. get Id ( ) );*

**}}**     *o/p → main*

---

### **Wait**

- It is non static method of object class.
- It is used only in the case of synchronize.
- It will wait still it receive notifications.

---

### **Sleep**

- It is a static method thread class.
- It can be used in any code.
- It will pause its execution for few milliseconds.

---

**Notify :** It is the non - static method of object class which notifies the thread which is about access the resource.

---

**Notify all :** It is the non - static method of object class which will notify all the thread which is about access the resource.

---

## **FILE HANDLING**

File is nothing but collection of similar kind of files or data.

In java file is a class which belongs to java.io package which should be imported explicitly.

In a file we can have non – static methods like

- Mkdir ( ) → make directories.
- Create new file ( ) → it helps to create the file.
- Exits ( ) → it check whether the file is present or not , if it is present it will return true else it will return false.
- Delete ( ) → which will help to delete the file in the given path.

**Ex:**

***Package filehandlingtopic;***

***Import java.io.file;***

```
Class Demo {  
Public static void main ( String [ ] args ) {  
File f1 = new file ( “d://java batch”);  
If (f1.mkdir ( ) )  
{  
System.out.println ( “folder created “);  
}  
Else  
{  
System.out.println ( “folder created “);  
}  
If (f1.exists ( ) )  
{  
System.out.println ( “folder exists“);  
}  
Else  
{  
System.out.println ( “folder does not exists“);  
}  
if (f1.delete ( ) )  
{  
System.out.println ( “folder is deleted “);  
}  
Else  
{
```

```
System.out.println ( "folder is not deleted ");  
}}}            o / p → folder created exists    file deleted
```

---

**1. Write a program to create a text files.**

```
Package filehandlingtopic;  
Import java.io.file;  
Import java.io.IOException;  
Public Class Sample {  
Public static void main ( String [ ] args ) throws IO exception  
{  
File f1 = new file ( );  
If (f1.createNewFile ( ) )  
{  
System.out.println ( "file created " );  
}  
Else  
{  
System.out.println ( "file not created " );  
}}}            o / p → file created
```

- File writer and file reader are the inbuilt classes which belongs to java.io package.
- File writer class helps to write the data into the file.
- File reader will helps to read the data from the file.

**2. Write a program to write a data into the files.**

```
Package filehandlingtopic;  
Import java.io.file;
```

---



```
Import java.io.IOException;  
Import java.io.filewriter;  
Public Class Sample {  
Public static void main ( String [ ] args ) throws IO exception  
{  
File f1 = new file (“D://rcb.txt” );  
File writer fw = new file writer (f1);  
Fw. Write(“hello I will get job”);  
System.out.println ( “Data is written “);  
Fw. Flush ( );  
}}          o / p → Data is written
```

---

3. Write a program to read a data from the files.

```
Package filehandler;  
Import java.io.file;  
Import java.io.fileReader;  
Public Class Sample {  
Public static void main ( String [ ] args ) throws IO exception  
{  
File f1 = new file (“D://rcb.txt” );  
FileReader fr = new FileReader (f1);  
Char [ ] arr = new char[(int) f1.length( )];  
Fr.read(arr);  
String s1 = new String(arr);  
System.out.println (s1);
```

**}}                    o / p → hello I will get job**

---

```
Package filehandlingtopic;  
Import java.io.Bufferdwriter;  
Import java.io.file;  
Import java.io.IOException;  
Import java.io.filewriter;  
Public Class Sample {  
Public static void main ( String [ ] args ) throws IO exception  
{  
File f1 = new file ("D://rcb.txt" );  
FileWriter fw = new FileWriter (f1,true);  
BufferdWriter bw = new BufferdWriter (fw);  
Bw.write("hello");  
Bw.line( );  
Bw.write("hello");  
Bw.line( );  
System.out.println ("data is written");  
Bw.flush( );  
}}}
```

---

```
Package filehandlingtopic;  
Import java.io.BufferdReader;  
Import java.io.file;  
Import java.io.IOException;  
Import java.io.filewriter;
```

---

```
Public Class Sample {  
Public static void main ( String [ ] args ) throws IO exception  
{  
File f1 = new file (“D://rcb.txt” );  
FileReader fw = new FileReader (f1);  
BufferedReader br = new BufferedReader (fr);  
String s1 = br.readLine();  
While(s1!=null)  
{  
System.out.println (s1);  
s1 = br.readLine();  
}}}           o / p → hello  hello  hello  hello
```

---

- Data is class which belongs to java.util.package which has inbuilt methods which helps to get the data.

```
Package filehandlingtopic;  
Import java.util.data;  
Public Class Sample {  
Public static void main ( String [ ] args ) throws IO exception  
{  
Data d1 = new Data ( );  
System.out.println (d1.getdate());  
System.out.println (d1.getmonth());  
}}           o/p → 22 02
```

---

- File input stream and file output stream are the input class which belongs to java.io. package . it helps to read and write the media files.

---

#### 4. Write a program to read an image from the folder and write it back with different names.

```
Package qsp ;  
Import java .io. file;  
Import java.io.*;  
Public class Sample  
{  
Public static void main ( String [ ] args ) throws IO exception  
{  
File f1 =new file (“D://k.jpg”);  
File input stream fin = File input stream ( f1 ) ;  
Byte [ ] arr =new byte [(int) f1.length ( )];  
Fin.read (arr);  
File output stream fout = new File output stream (“D:// rak.jpg”);  
Fout.write (arr);  
Fout . flush ( );  
System.out.println (“rcb is back”);  
}  
} remove → f1. Write ( “ “ )
```

#### Write a program to achieve singleton design pattern

Singleton design pattern means restricting the object creation by 1 through - out the life cycle of the application is called as singleton design Pattern.

```
Class Sample {  
Static int count = 0 ;  
Int a =10;  
Static Sample s1;  
Private Sample ( ) {  
Count ++;  
}  
Public static Sample getInstance ( )  
{  
If (count<1)  
{  
S1 = new Sample ( );  
}  
Return s1;  
}  
public static void Instance ( int y )  
{  
S1.a =y;  
}  
Class Mainclass {  
Public static void main ( String [ ] args ) {  
Sample s2 = Sample .getInstance ( );  
System.out.println (s2.a);  
Sample s3 = Sample .getInstance ( );  
System.out.println (s3.a);
```

```
Sample .setinstance (126 );  
System.out.println (s3.a);  
}}
```

---

*An empty interface is called as markee interface.*

*Any interface which has only 1 abstract method is called functional interface.*

---

### **Instance of operator**

It checks does the object belong or instance belong to class hierarchy.

If it belongs then it returns true.

*Package qsp;*

*Interface Animal {*

*Void noise ( );*

*}*

*Class Cat implements Animal {*

*Public Void noise ( ) {*

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

*}}*

*Class Demo {*

*Public static void main ( String [ ] args ) {*

*Animal a1 = null ;*

*System.out.println ("an instance of animal");*

*}*

*}*

## **Constructor overloading**

Developing multiple constructor within the class but variation in argument list is called constructor overloading.

Variation in argument list means :

1. variation in the datatype .
2. variation in the length of the arguments.
3. variation in the order of occurrence of arguments

---

## **This Calling Statement**

It is used to call from one constructor to another constructor within the class.

### **Rules :**

1. This calling statements should be first statement inside the constructor.
2. It is used only in the case of constructor overloading.
3. We can develop 2 this calling statements in a single constructor.

**Ex:**    *class Demo {*  
          *Demo ( int a ) {*  
          *System.out.println( a );*  
          *}*  
          *Demo ( String b ) {*  
          *This ( 10 );*  
          *System.out.println( b );*  
          *}*  
          *Demo ( int x , String y ) {*  
          *This ( “cool” );*  
*System.out.println( x+” “ +y );*  
          *}*

```
Demo ( String y, int x ) {  
    This ( 10 , “hello” );  
    System.out.println( y+” “ +x );  
    } }  
Class Mainclass {  
    Public static void main ( String [ ] args ) {  
        New Demo ( “hi” , 126 );  
    } } o / p → 10 cool 10 hello hi 126
```

---

### **Super Calling Statement**

It is used to call from sub class constructor to the immediate super class constructor .

Rules :

1. super calling statement should be the first statement inside the constructor .
2. it is used only in the case of inheritance.
3. super calling can be written implicitly or explicitly.

```
Ex 1 : class Demo {  
    Demo ( String a ) {  
        System.out.println( “hahaha” );  
    } }  
Class Tester extends Demo {  
    Tester ( double y ) {  
        Super ( “ hi “ );  
        System.out.println( “cool” );  
    } }  
Class Sample extends Tester {
```



```
Sample ( int a ) {  
    Super ( 10.56 ) ;  
    System.out.println( "hii" ) ;  
}}  
  
Class Mainclass {  
    Public static void main ( String [ ] args ) {  
        New Sample ( 10 ) ;  
    }}    o / p → hahaha    cool    cool    hi    10
```

---

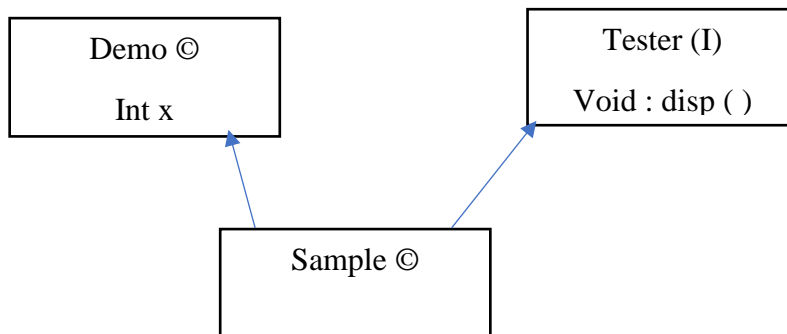
```
Ex 2 : class Demo {  
    Demo ( ) {  
        System.out.println( "cool" ) ;  
    }}  
  
    Class Tester extends Demo {  
        // default constructor will be created  
    }}  
  
    Class Sample extends Tester {  
        Sample ( ) {  
            System.out.println( "hii" ) ;  
        }}  
  
    Class Mainclass {  
        Public static void main ( String [ ] args ) {  
            New Sample ( ) ;  
        }}    o / p →    cool    hi
```

**Why multiple inheritance is not possible in java through classes ?**

- We cannot develop 2 super calling statement in the sub class constructor because super calling statement should be first inherited .
- Due to diamond problem i.e. each and every class extends object class , the sub most class will have ambiguity ( confusion ) from which super classes it should be inherited.

**We can achieve multiple inheritance through interface.**

- Interface does not support constructor . hence we don' t develop 2 super calling statements in the sub class constructor.
- Interface does not extends any class , interface itself is a super most and hence there will be no diamond problem.

***Interface Demo***

```
{  
Int x =10;  
}
```

***Interface Tester {***

```
Void disp ( ) ;  
}
```

***Class Sample implements Demo , Tester {***

```
Public void disp ( )  
{
```

```
System.out.println ("hi");  
}  
Public static void main (String [ ] args )  
{  
Sample s1 = new Sample ( );  
S1 .disp ( );  
System.out.println (s1.x);  
}}
```

---

### **What are java features ?**

1. it is compiled and interpreted.
2. it is platform independent .
3. it is secured .
4. it is robust.
5. it is multi – threaded .
6. it is easy and simple to understand .

---

### **why java is object oriented programming language ?**

Java is object oriented programming language because it supports the following concepts :

1. encapsulation.
2. polymorphism
3. inheritance
4. class and object
5. abstraction.

This is why it is object oriented

**What is the difference between this calling and super calling**

No.	This calling	Super calling
1)	It is used to call form one constructor to another constructor within the class.	It is used to call form one constructor to another constructor between the class.
2)	It is used in the case of constructor overloading	It is used in the case of inheritance.
3)	It should be written explicitly.	It should be written implicitly and explicitly.
4)	It should be first statement inside the constructor.	It should be first statement inside the constructor.

**What is the difference between this keyword and super keyword**

No.	This Keyword	Super Keyword
1)	It is used to point to the current object.	It is used in the method overriding.
2)	When the local variables and global variables are same , to differentiate between them we use this keyword.	Along with the overridden implementation if I need super class implementation the we use super method name.
3)	It can be used only in the non- static context.	It can be used only in method overriding.

### **What is the difference between abstract class and interface**

No.	Abstract class	Interface
1)	Any class which is declared with the keyword abstract is called abstract class.	It is a java type which is by default abstract.
2)	In abstract class , we have 3 members.	In interface , we have 2 members.
3)	Variables in abstract class can be static or non – static .	Variables in interface are by default static and final.
4)	Method can have any access specifier.	The methods are by default public and abstract.
5)	We can develop both concrete method and abstract method.	It is by default abstract.
6)	Abstract class will have constructor.	It do not support constructor.
7)	Abstract class extends object class.	It is the super most class which does not extends object class.
8)	Through abstraction we can achieve constructor chaining.	We cannot achieve constructor chaining.
9)	We have to override all the methods in the sub class which is abstract.	By default all the methods should be overridden.

### **Constructor chaining :**

sub class constructor calling its immediate super class constructor intern super class constructor calling its immediate super class constructor is called as constructor chaining.

**Why the main method is declared as Public static void main (String [ ] args ) ?**

- Public is application level access where JVM should be able to access main method for execution.
- Static is the modifier which helps to load the main method before execution.
- The return type is void where main method does not return any value.
- Main is the method name.
- The parameter are of string [ ] type because main method receives the input in the form of string [ ] array.

Int x = Integer.parseInt ("123") ; → convert from string to int

System.out.println ( x ) ;

---

**Wrapper class**

Wrapper class are the inbuilt classes which belongs to java. Lang package .

- For each and every primitive data type we have the corresponding class and those classes are called as wrapper class.
- We can perform an operation called boxing and unboxing.

**Boxing**

Converting from primitive data type to wrapper class object is called as boxing.

**Unboxing**

Converting from wrapper class object back to its primitive data type is called as Unboxing.

<b>primitive data type</b>	<b>wrapper class</b>
<b>Byte</b>	<b>byte</b>
<b>Short</b>	<b>short</b>
<b>Int</b>	<b>integer</b>
<b>Long</b>	<b>long</b>
<b>Double</b>	<b>double</b>
<b>Char</b>	<b>Char</b>
<b>Boolean</b>	<b>boolean</b>

**Ex: Integer a1 = new Integer (10) ; // Boxing**

**s.o.p (a1);**

**int x = a1 ; // unboxing**

**s.o.p (x);**

**char c1 = new character ('A') ;**

**s.o.p (c1);**

**char ch = c1 ;**

**s.o.p (ch);**

---

### **how the elements are stored in the form of object ?**

whenever we add the elements , the primitive data types will get converted to the corresponding wrapper class object and it will be casted and stored in the form of object.

**Ex: list <integer > l1 = new ArrayList <integer > ( ) ;**

**l1.add(10);**

**l1.add(20);**

**l1.add(30);**

**s.o.p(l1);**

**Ex: l1.add(10);**

**Object obj = new integer (10);**

**s.o.p (obj);**

---

### **Write a program to store user defined objects in an ArrayList class object**

**Package collection topic ;**

**Import java . util. ArrayList ;**

**Class Mobile {**

```
Void call ( ) {  
System.out.println (“ missed call “);  
}}  
  
Class Sample {  
Public static void main (String [] args ) {  
ArrayList <mobile> l1 = new ArrayList <mobile> ( );  
l1.add(new mobile ());  
l1.add(new mobile ());  
l1.add(new mobile ());  
mobile o1 = (mobile ) l1.get(0);  
o1.call ( );  
}}
```

---

```
Class Mobile {  
Int mob_cost ;  
String mob_name ;  
String mob_color ;  
Mobile (Int mob_cost , String mob_name , String mob_color ) {  
This . mob_cost = mob_cost ;  
This . mob_name = mob_name ;  
This . mob_color = mob_color ;  
}  
  
Public static void main (String [] args ) {  
Mobile m1 = new Mobile (25000 , “oneplus 6 “ , “black”);  
System.out.println( m1 . mob_cost);
```



```
System.out.println( m1 . mob_name);  
System.out.println( m1 . mob_color);  
}}
```

---

### **REVERSE Order**

```
Package collection topic ;  
Import java . util. ArrayList ;  
Class Sample {  
Public static void main (String [] args ) {  
ArrayList ll = new ArrayList ( );  
    ll.add(80);  
    ll.add(20.56);  
    ll.add('A');  
    ll.add("hello");  
    ll.add(10);  
for (int i= ll.size ( ) -1 ; i>=0 ; i--)  
{  
System.out.println( ll.get[i]);  
} } }
```

o/p → 10 A 80

---

## **JAVA INTERVIEW QUESTIONS**

### **1. What is Java?**

Java is an object oriented programming language and high-level programming language which is used to develop software.

### **2. List features of Java?**

Some features include Object Oriented, Platform Independent, Robust, Interpreted, Multi-threaded.

### **3. What are the supported platforms by Java Programming Language?**

Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX/Linux like HP-Unix, Sun Solaris, Redhat Linux, Ubuntu, CentOS, etc.

### **4. What is JVM ? Why is Java called the Platform Independent.**

It is whole responsible to execute the java program. JRE is installed in each and every electronics device and .class file can be executed on any electronic device , hence Java is platform independent.

### **5. What is the Difference between JIT , JDK and JRE ?**

It is whole responsible to convert . class file to binary format.

It is whole responsible to execute the java program.

It is an environment setup provided to run the java program.

### **6. List two Java IDE's?**

NetBeans, Eclipse, etc.

### **7. List some Java keywords unlike C,C++ keywords?**

Some Java keywords are import, super, finally, etc.

### **8. What are the Types by Java ?**

1) Class    2) enum    3) Annotation    4) Interface

**9. What are the Data Types supported by Java ?**

byte • short • int • long • float • double • Boolean • char

**10. What is the default value of byte datatype in Java?**

Default value of byte datatype is 0.

**11. What is the default value of float and double datatype in Java?**

Default value of float and double datatype is float its 0.0f ( unique ) and for double it's 0.0d

**12. When a byte datatype is used?**

This data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.

**13. What is Variable?**

Variable is a named memory location which is used to store some values or data and it can change N no of times during execution.

**14. What kind of variables a class can consist of?**

A class consist of Local variable and global variables.

**15. What is a Local Variable?**

Any variables which is declared within the method is called as local variables.

**16. What is a global Variable?**

Any variables which is declared outside method and inside the class is called as Global variables.

**17. What is Method?**

Method is a block of statements which will get executed whenever it is called or invoked.

**18. What is Static ?**

Any member of the class declared with a keyword static is called as static member of the class.

**19. What is Non - Static ?**

Any member of the class declared without a keyword static is called as Non - static member of the class.

**20. Define composition?**

The class having an object of another class is known as composition. It is also called HAS a relationship.

**21. What is static initialization block?**

Any block which is declared with a keyword static is called as static initialization block . SIB is used to initialize static members. SIB will get executed before main method.

**22. What is Non - static initialization block?**

Any block which is declared without a keyword static is called as Instance initialization block. IIB is used to initialize non - static members. IIB will get executed whenever an object is created.

**23. What are pass by value and pass by reference ?**

Calling or invoking a method by passing primitive type of data is called as call by value or pass by value.

Calling or invoking a method by passing references variables is called as call by references or pass by references.

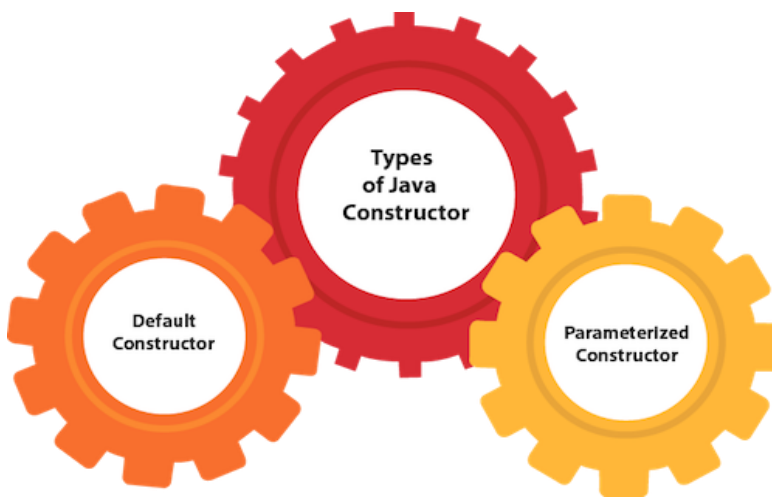
**24. What is a Constructor ?**

It is a special type of method or special type (member) of the class which is used to initialize data members.

**25. How many types of constructors are used in Java?**

There are two types of constructors in Java.

- Default Constructor: default constructor is the one which does not accept any value.
- The default constructor is mainly used to initialize the instance variable with the default values. It can also be used for performing some useful task on object creation. A default constructor is invoked implicitly by the compiler if there is no constructor defined in the class.
- Parameterized Constructor: The parameterized constructor is the one which can initialize the instance variables with the given values. In other words, we can say that the constructors which can accept the arguments are called parameterized constructors.



## **26. What is the purpose of default constructor?**

The java compiler creates a default constructor only if there is no constructor in the class.

## **27. What is This keyword ?**

It is used to point to the current object whenever the local variable and global variable names are same to differentiate between them we use “ this “ keyword.

## **28. What is Method / Function Overriding and Overloading in Java ?**

Developing multiple methods with the same name but variation in argument list is called as method overloading.

Developing a method in the sub class with the same name and signature as in the superclass but with different implementation in the sub class is called as method overriding.

**29. Define class?**

Class is a blueprint or a template to create object .

**30. What do you mean by Object?**

Object is real time entity which has its own state and behaviour.

**31. List the three steps for creating an Object for a class?**

An Object is first declared, then instantiated and then it is initialized.

**32. What is Singleton class?**

Singleton class control object creation, limiting the number to one but allowing the flexibility to create more objects if the situation changes.

**33. What is Inheritance ?**

Inheriting the property from one class to another class is called as Inheritance.

**34. Types of Inheritance ?**

- |                             |                              |
|-----------------------------|------------------------------|
| 1. single level Inheritance | 2. multi - level Inheritance |
| 3. hierarchical Inheritance | 4. multiple Inheritance      |
| 5. hybrid Inheritance       |                              |

**35. Is there any limitation of using Inheritance?**

Yes, since inheritance inherits everything from the super class and interface, it may make the subclass too clustering and sometimes error-prone when dynamic overriding or dynamic overloading in some situation.

**36. When super keyword is used?**

In the case of method overriding , along with the sub class implementation , if we need super class implementation thus we should go for super . method name

**37. Does Java support multiple inheritance ?**

No, Java does not support multiple inheritance. Each class is able to extend only on one class ,but is able to implement more than one interfaces.

**38. What is type casting?**

Converting from one type to another type is called Type casting.

**39. What is Polymorphism?**

An object showing different behaviour at different stages of its lifecycle is called polymorphism.

**40. What is compile time polymorphism ?**

The method declaration getting binded to its definition at the compile time by the compiler based on the arguments passed is called “ compile time polymorphism “.

**41. What is runtime polymorphism or dynamic method dispatch?**

The method declaration gets binded to its definition of its definition at the run time by the JVM based on the object created is called as “ run time polymorphism“.

**42. What is Dynamic Binding and late binding?**

Since the method declaration gets binded to its definition , at the rebinded , hence it is called as “ Dynamic binding “.

Since the method declaration gets binded to its definition , at the run time , hence it is called as “ late binding “.

**43. What is Abstraction?**

Hiding the complexity of the system and exposing only the required functionality to the end user is called as abstraction.

**44. What is Abstract class?**

Any class which is declared with a keyword abstract is called as “ Abstract class“.

**45. When Abstract methods are used?**

Any method which is declared with a keyword abstract is called as “Abstract method”.

**46. What is Encapsulation?**

It is one of the oops principles. Java is by default encapsulated.

**47. What is the primary benefit of Encapsulation?**

The main benefit of encapsulation is the ability to modify our implemented code without breaking the code of others who use our code. With this Encapsulation gives maintainability, flexibility and extensibility to our code.

**48. What is an Interface?**

Interface is by default abstract and it is a pure abstract body.

**49. Give some features of Interface?**

It includes – Interface cannot be instantiated An interface does not contain any constructors. All of the methods in an interface are abstract.

**50. Define Packages in Java?**

It is a folder structure which is used to store similar kind of files. The packages should be created always in the reverse order.

**51. Why Packages are used?**

Packages are used in Java in-order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations, etc., easier.

**52. What do you mean by Access Modifier?**

Access specifier is used to restrict the access from one class to another class ( or ) from one package to another package.

**53. What is Private access modifier?**

Any member of the class declared with a keyword private is called as private access specifier.



**54. What is Default / Package Level access modifier?**

If we declare any member of the class without any access specifier ( or ) keyword is called as default access specifier.

**55. What is protected access modifier?**

Any members of the class declared with the keyword protected is called as “protected member of the class “.

**56. What is public access modifier?**

Any members of the class declared with a keyword public is called as “public access specifier”.

**57. Why is String class considered immutable?**

When multiple reference variable are pointing to a single object and if one of the reference variable de reference with the current object it will not affect other reference variable. This is why it is immutable.

**58. Why is String Buffer called mutable?**

The String class is considered as immutable, so that once it is created a String object cannot be changed. If there is a necessity to make a lot of modifications to Strings of characters then String Buffer should be used.

**59. What is the difference between String Buffer and StringBuilder class?**

Use StringBuilder whenever possible because it is faster than String Buffer. But, if thread safety is necessary then use String Buffer objects.

**60. what is Thread ?**

Thread is an extension instance which owns of its own CPU time and memory.

**61. What do you mean by Multithreaded program?**

Processing multiple threads simultaneously is called as multi - threading.

**62. What is Sleep method ?**

Sleep method is a static method which pause its execution for few milliseconds.

**63. What is multi – tasking ?**

It is a Process of executing multiple task simultaneously is called as multi - tasking.

**64. What is Synchronized ?**

Processing one by one thread is called as Synchronized .

Processing multiple threads simultaneously is called as “non – Synchronized “.

**65. What are the two ways in which Thread can be created?**

Thread can be created by - implementing Runnable interface, extending the Thread class.

**66. What invokes a thread's run method?**

After a thread is started, via its start method of the Thread class, the JVM invokes the thread's run method when the thread is initially executed.

**67. Describe life cycle of thread?**

A thread is a execution in a program. The life cycle of a thread include – Newborn state, Runnable state, Running state, Blocked state, Dead state

**68. What is the difference between processes and threads ?**

A process is an execution of a program, while a Thread is a single execution sequence within a process. A process can contain multiple threads. A Thread is sometimes called a lightweight process.

**69. What is the difference between yielding and sleeping?**

When a task invokes its yield method, it returns to the ready state. When a task invokes its sleep method, it returns to the waiting state.

**70. How does multi-threading take place on a computer with a single CPU?**

The operating system's task scheduler allocates execution time to multiple tasks. By quickly switching between executing tasks, it creates the impression that tasks execute sequentially.

### **71.What is an Exception?**

It is an event triggered during the execution of the program which interrupt the execution and stops the execution abruptly / suddenly handling this event by using try and catch or throws is called as “Exception handling”.

### **72.What do you mean by Checked Exceptions?**

Any exception which is caught by the compiler at the compile time , this is called COMPILE TIME EXCEPTION.

### **73. Explain Runtime Exceptions?**

Any exception which is not caught at the compile time and found at the run time is called as RUN TIME EXCEPTION.

### **74. Which are the two subclasses under Exception class?**

The Exception class has two main subclasses : IO Exception class and Runtime Exception Class.

### **75. When throws keyword is used?**

It is used to throw the exception of throwable type. It will be written internally.

### **76. When throw keyword is used?**

An exception can be thrown, either a newly instantiated one or an exception that you just caught, by using throw keyword.

### **77. How finally used under Exception Handling?**

Finally block is block in exceptional handling which will get executed mandatory even though the exception is addressed or not addressed.

### **78. What is Null Pointer Exception?**

A Null Pointer Exception is thrown when calling the instance method of a null object, accessing or modifying the field of a null object etc.

**79. When Arithmetic Exception is thrown?**

The Arithmetic Exception is thrown when integer is divided by zero or taking the remainder of a number by zero. It is never thrown in floating-point operations.

**80. Which arithmetic operations can result in the throwing of an Arithmetic Exception?**

Integer / and % can result in the throwing of an Arithmetic Exception.

**81. If a variable is declared as private, where may the variable be accessed?**

A private variable may only be accessed within the class in which it is declared.

**82. What class of exceptions are generated by the Java run-time system?**

The Java runtime system generates Runtime Exception and Error exceptions.

**83. Can try statements be nested?**

Yes

**84. Is it necessary that each try block must be followed by a catch block?**

It is not necessary that each try block must be followed by a catch block. It should be followed by either a catch block or a finally block.

**85. What is the Collections API?**

Collections is an unified architecture which consist of interface and class.

**86. Does garbage collection guarantee that a program will not run out of memory?**

Garbage collection does not guarantee that a program will not run out of memory. It is possible for programs to use up memory resources faster than they are garbage collected. It is also possible for programs to create objects that are not subject to garbage collection.

**87. Explain Set Interface?**

It is a collection of element which cannot contain duplicate elements. The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.

**88. Explain Tree Set?**

It is a Set implemented when we want elements in a sorted order.

**89.What is Comparable Interface?**

It is used to sort collections and arrays of objects using the collections. sort and java.util. The objects of the class implementing the Comparable interface can be ordered.

**90. Why Vector class is used?**

The Vector class provides the capability to implement a growable array of objects. Vector proves to be very useful if you don't know the size of the array in advance, or you just need one that can change sizes over the lifetime of a program.

**91. What are the advantages of Array List over arrays?**

Array List can grow dynamically and provides more powerful insertion and search mechanisms than arrays.

**92. Why deletion in Linked List is fast than Array List?**

Deletion in linked list is fast because it involves only updating the next pointer in the node before the deleted node and updating the previous pointer in the node after the deleted node.

**93. How do you decide when to use Array List and Linked List?**

If you need to frequently add and remove elements from the middle of the list and only access the list elements sequentially, then LinkedList should be used. If you need to support random access, without inserting or removing elements from any place other than the end, then Array List should be used.

**94. What is a Values Collection View ?**

It is a collection returned by the values method of the Map Interface, It contains all the objects present as values in the map.

**95. Which package is used for pattern matching with regular expressions?**

java.util.regex package is used for this purpose.

**96. java.util.regex consists of which classes?**

java.util.regex consists of three classes – Pattern class, Matcher class and Pattern Syntax Exception class.

**97. What is finalize method?**

It is possible to define a method that will be called just before an object's final destruction by the garbage collector. This method is called finalize, and it can be used to ensure that an object terminates cleanly.

**98. What is the purpose of File class?**

It is used to create objects that provide access to the files and directories of a local file system.

**99. What is the difference between the Reader/Writer class hierarchy and the Input Stream/ Output Stream class hierarchy?**

The Reader/Writer class hierarchy is character-oriented, and the Input Stream / Output Stream class hierarchy is byte-oriented.

**100. What are Wrapper classes?**

These are classes that allow primitive types to be accessed as objects. Example: Integer, Character, Double, Boolean etc.

**101. What is JAR file?**

JAR files is Java Archive files and it aggregates many files into one. It holds Java classes in a library. JAR files are built on ZIP file format and have .jar file extension.

**102. What is a WAR file?**

This is Web Archive File and used to store XML, java classes, and Java Server pages. which is used to distribute a collection of Java Server Pages, Java Servlets, Java classes, XML files, static Web pages etc.

**103. What is the difference between object oriented programming language and object based programming language?**

Object based programming languages follow all the features of OOPs except Inheritance. JavaScript is an example of object based programming languages.

**104. Can a constructor be made final?**

No, this is not possible.

**105. What is final class?**

Final classes are created so the methods implemented by that class cannot be overridden. It can't be inherited.

**106. How many bits are used to represent Unicode, ASCII, UTF-16, and UTF-8 characters?**

Unicode requires 16 bits and ASCII require 7 bits. Although the ASCII character set uses only 7 bits, it is usually represented as 8 bits. UTF-8 represents characters using 8, 16, and 18 bit patterns. UTF-16 uses 16-bit and larger bit patterns.

**107. Which Java operator is right associative?**

The = operator is right associative.

**108. What is the difference between a break statement and a continue statement?**

A break statement results in the termination of the statement to which it applies switch ,for ,do ,or while. A continue statement is used to end the current loop iteration and return control to the loop statement.

**109. What are Class Loaders?**

A class loader is an object that is responsible for loading classes. The class Class Loader is an abstract class.

**110. What will happen if static modifier is removed from the signature of the main method?**

Program throws "No Such Method Error" error at runtime.

**111. What is the default value of an object reference declared as an instance variable?**

Null, unless it is defined explicitly.

**112. Can constructor be inherited?**

No, constructor cannot be inherited

**113. What is the difference between the >> and >>> operators?**

The >> operator carries the sign bit when shifting right. The >>> zero-fills bits that have been shifted out.

**114. Does Java allow Default Arguments?**

No, Java does not allow Default Arguments.

**115. Where import statement is used in a Java program?**

Import statement is allowed at the beginning of the program file after package statement.

**116. Can an Interface extend another Interface?**

Yes an Interface can inherit another Interface, for that matter an Interface can extend more than one Interface.

**117. Which object oriented Concept is achieved by using overloading and overriding?**

Polymorphism

**118. Can a double value be cast to a byte?**

Yes, a double value can be cast to a byte.

**119. Is there any need to import java. Lang package?**



No, there is no need to import this package. It is by default loaded internally by the JVM.

**120. What environment variables do I need to set on my machine in order to be able to run Java programs?**

CLASSPATH and PATH are the two variables.

**121. What is an enumeration?**

An enumeration is an interface containing methods for accessing the underlying data structure from which the enumeration is obtained. It allows sequential access to all the elements stored in the collection.

**122. What is difference between Path and Class path?**

Path and Class path are operating system level environment variables. Path is defines where the system can find the executables.exe files and class path is used to specify the location of .class files.

**123. What is an Applet ?**

A java applet is program that can be included in a HTML page and be executed in a java enabled client browser. Applets are used for creating dynamic and interactive web applications.

**124. Explain the life cycle of an Applet.**

An applet may undergo the following states:

- Init : An applet is initialized each time is loaded.
- Start : Begin the execution of an applet.
- Stop : Stop the execution of an applet.
- Destroy : Perform a final cleanup, before unloading the applet.

**125. What happens when an applet is loaded ?**

First of all, an instance of the applet's controlling class is created. Then, the applet initializes itself and finally, it starts running.

**126. The immediate superclass of the Applet class?**

Panel is the immediate superclass. A panel provides space in which an application can attach any other component, including other panels.

**127. Define canvas?**

It is a simple drawing surface which are used for painting images or to perform other graphical operations.

**128. Define Network Programming?**

It refers to writing programs that execute across multiple devices computers, in which the devices are all connected to each other using a network.

**129. What is a Socket?**

Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.

**130. Advantages of Java Sockets?**

Sockets are flexible and sufficient. Efficient socket based programming can be easily implemented for general communications. It cause low network traffic.

**131. Disadvantages of Java Sockets?**

Socket based communications allows only to send packets of raw data between applications. Both the client-side and server-side have to provide mechanisms to make the data useful in any way.

**132. Which class is used by server applications to obtain a port and listen for client requests?**

java.net.ServerSocket class is used by server applications to obtain a port and listen for client requests

**133. Which class represents the socket that both the client and server use to communicate with each other?**

java.net.Socket class represents the socket that both the client and server use to communicate with each other.

**134. What is the difference between a Choice and a List ?**

A Choice is displayed in a compact form that must be pulled down, in order for a user to be able to see the list of all available choices. Only one item may be selected from a Choice.

A List may be displayed in such a way that several List items are visible. A List supports the selection of one or more List items.

**135. What is a layout manager ?**

A layout manager is the used to organize the components in a container.

**136. What is the difference between a Scrollbar and a JScroll Pane ?**

A Scrollbar is a Component, but not a Container. A Scroll Pane is a Container. A Scroll Pane handles its own events and performs its own scrolling.

**137. Which Swing methods are thread-safe ?**

There are only three thread-safe methods: repaint, revalidate, and invalidate.

**138. Name three Component subclasses that support painting.**

The Canvas, Frame, Panel, and Applet classes support painting.

**139. What is clipping ?**

Clipping is defined as the process of confining paint operations to a limited area or shape.

**140. What is the difference between a Menu Item and a Check box Menu Item?**

The Check box Menu Item class extends the Menu Item class and supports a menu item that may be either checked or unchecked.

**141. How are the elements of a Border Layout organized ?**

The elements of a Border Layout are organized at the borders (North, South, East, and West) and the center of a container.

**142. How are the elements of a Grid Bag Layout organized ?**

The elements of a Grid Bag Layout are organized according to a grid. The elements are of different sizes and may occupy more than one row or column of the grid. Thus, the rows and columns may have different sizes.

**143. What is the difference between a Window and a Frame ?**

The Frame class extends the Window class and defines a main application window that can have a menu bar.

**144. What is the relationship between clipping and repainting ?**

When a window is repainted by the AWT painting thread, it sets the clipping regions to the area of the window that requires repainting.

**145. What is the relationship between an event-listener interface and an event adapter class ?**

An event-listener interface defines the methods that must be implemented by an event handler for a particular event. An event adapter provides a default implementation of an event-listener interface.

**146. How can a GUI component handle its own events ?**

A GUI component can handle its own events, by implementing the corresponding event-listener interface and adding itself as its own event listener.

**147. What advantage do Java's layout managers provide over traditional windowing systems ?**

Java uses layout managers to lay out components in a consistent manner, across all windowing platforms. Since layout managers aren't tied to absolute sizing and positioning, they are able to accommodate platform-specific differences among windowing systems.

**148. What is the design pattern that Java uses for all Swing components ?**

The design pattern used by Java for all Swing components is the Model View Controller (MVC) pattern.

**149. What is JDBC ?**

JDBC is an abstraction layer that allows users to choose between databases. JDBC enables developers to write database applications in Java, without having to concern themselves with the underlying details of a particular database.

**150. Explain the role of Driver in JDBC.**

The JDBC Driver provides vendor-specific implementations of the abstract classes provided by the JDBC API. Each driver must provide implementations for the following classes of the java. sql package : Connection, Statement, Prepared Statement, Callable Statement, Result Set and Driver.

**151. What is the purpose Class. for Name method ?**

This method is used to method is used to load the driver that will establish a connection to the database.

**152. What is the advantage of Prepared Statement over Statement ?**

Prepared Statements are precompiled and thus, their performance is much better. Also, Prepared Statement objects can be reused with different input values to their queries.

**153. What is the use of Call able Statement ? Name the method, which is used to prepare a Call able Statement.**

A Call able Statement is used to execute stored procedures. Stored procedures are stored and offered by a database. Stored procedures may take input values from the user and may return a result. The usage of stored procedures is highly encouraged, because it offers security and modularity. The method that prepares a Call able Statement is the following: Call able Statement. Prepare Call();

**154. What does Connection pooling mean ?**

The interaction with a database can be costly, regarding the opening and closing of database connections. Especially, when the number of database clients

increases, this cost is very high and a large number of resources is consumed. A pool of database connections is obtained at start up by the application server and is maintained in a pool. A request for a connection is served by a connection residing in the pool. In the end of the connection, the request is returned to the pool and can be used to satisfy future requests.

### **155. What is RMI ?**

The Java Remote Method Invocation (Java RMI) is a Java API that performs the object-oriented equivalent of remote procedure calls(RPC),with support for direct transfer of serialized Java classes and distributed garbage collection. Remote Method Invocation(RMI) can also be seen as the process of activating a method on a remotely running object. RMI offers location transparency because a user feels that a method is executed on a locally running object. Check some RMI Tips [here](#).

### **156. What is the basic principle of RMI architecture ?**

The RMI architecture is based on a very important principle which states that the definition of the behavior and the implementation of that behavior, are separate concepts. RMI allows the code that defines the behavior and the code that implements the behavior to remain separate and to run on separate JVMs.

### **157. What are the layers of RMI Architecture ?**

The RMI architecture consists of the following layers:

- **Stub and Skeleton layer:** This layer lies just beneath the view of the developer. This layer is responsible for intercepting method calls made by the client to the interface and redirect these calls to a remote RMI Service.
- **Remote Reference Layer:** The second layer of the RMI architecture deals with the interpretation of references made from the client to the server's remote objects. This layer interprets and manages references made from clients to the remote service objects. The connection is a one-to-one (unicast) link.
- **Transport layer:** This layer is responsible for connecting the two JVM participating in the service. This layer is based on TCP/IP connections between machines in a network. It provides basic connectivity, as well as some firewall penetration strategies.

**158. What is the role of Remote Interface in RMI ?**

The Remote interface serves to identify interfaces whose methods may be invoked from a non-local virtual machine. Any object that is a remote object must directly or indirectly implement this interface. A class that implements a remote interface should declare the remote interfaces being implemented, define the constructor for each remote object and provide an implementation for each remote method in all remote interfaces.

**159. What is the role of the java. rmi. Naming Class ?**

The java. rmi. Naming class provides methods for storing and obtaining references to remote objects in the remote object registry. Each method of the Naming class takes as one of its arguments a name that is a String in URL format.

**160 What is meant by binding in RMI ?**

Binding is the process of associating or registering a name for a remote object, which can be used at a later time, in order to look up that remote object. A remote object can be associated with a name using the bind or rebind methods of the Naming class.

**161. What is the difference between using bind() and rebind() methods of Naming Class ?**

The bind method bind is responsible for binding the specified name to a remote object, while the rebind method is responsible for rebinding the specified name to a new remote object. In case a binding exists for that name, the binding is replaced.

**162. What are the steps involved to make work a RMI program ?**

The following steps must be involved in order for a RMI program to work properly:

- Compilation of all source files.
- Generation of the stubs using rmic.
- Start the rmi registry.

- Start the RMI Server.
- Run the client program.

### **163. What is the role of stub in RMI ?**

A stub for a remote object acts as a client's local representative or proxy for the remote object. The caller invokes a method on the local stub, which is responsible for executing the method on the remote object. When a stub's method is invoked, it undergoes the following steps:

- It initiates a connection to the remote JVM containing the remote object.
- It marshals the parameters to the remote JVM.
- It waits for the result of the method invocation and execution.
- It unmarshals the return value or an exception if the method has not been successfully executed.
- It returns the value to the caller.

### **164. What is DGC ? And how does it work ?**

DGC stands for Distributed Garbage Collection. Remote Method Invocation (RMI) uses DGC for automatic garbage collection. Since RMI involves remote object references across JVM's, garbage collection can be quite difficult. DGC uses a reference counting algorithm to provide automatic memory management for remote objects.

### **165. What is the purpose of using RMI Security Manager in RMI ?**

RMI Security Manager provides a security manager that can be used by RMI applications, which use their own loaded code. The class loader of RMI will not download any classes from remote locations, if the security manager has not been set.

### **166. Explain Marshalling and demarshalling.**

When an application wants to pass its memory objects across a network to another host or persist it to storage, the in-memory representation must be



converted to a suitable format. This process is called marshalling and the revert operation is called demarshalling.

### **167. Explain Serialization and Deserialization.**

Java provides a mechanism, called object serialization where an object can be represented as a sequence of bytes and includes the object's data, as well as information about the object's type, and the types of data stored in the object. Thus, serialization can be seen as a way of flattening objects, in order to be stored on disk, and later, read back and reconstituted. Deserialization is the reverse process of converting an object from its flattened state to a live object.

### **168. What is a Servlet ?**

The servlet is a Java programming language class used to process client requests and generate dynamic web content. Servlets are mostly used to process or store data submitted by an HTML form, provide dynamic content and manage state information that does not exist in the stateless HTTP protocol.

### **169. Explain the architecture of a Servlet.**

The core abstraction that must be implemented by all servlets is the `javax.servlet.Servlet` interface. Each servlet must implement it either directly or indirectly, either by extending `javax.servlet.GenericServlet` or `javax.servlet.http.HttpServlet`. Finally, each servlet is able to serve multiple requests in parallel using multithreading.

### **170. What is the difference between an Applet and a Servlet ?**

An Applet is a client side java program that runs within a Web browser on the client machine. On the other hand, a servlet is a server side component that runs on the web server. An applet can use the user interface classes, while a servlet does not have a user interface. Instead, a servlet waits for client's HTTP requests and generates a response in every request.

### **171. What is the difference between Generic Servlet and Http Servlet ?**

Generic Servlet is a generalized and protocol-independent servlet that implements the `Servlet` and `Servlet Config` interfaces. Those servlets extending the `Generic Servlet` class shall override the `service` method. Finally, in order to develop

an HTTP servlet for use on the Web that serves requests using the HTTP protocol, your servlet must extend the `HttpServlet` instead. Check Servlet examples [here](#).

### **172. Explain the life cycle of a Servlet.**

On every client's request, the Servlet Engine loads the servlets and invokes its `init` methods, in order for the servlet to be initialized. Then, the Servlet object handles all subsequent requests coming from that client, by invoking the service method for each request separately. Finally, the servlet is removed by calling the server's `destroy` method.

### **173. What is the difference between `doGet()` and `doPost()` ?**

**Do GET:** The GET method appends the name-value pairs on the request's URL. Thus, there is a limit on the number of characters and subsequently on the number of values that can be used in a client's request. Furthermore, the values of the request are made visible and thus, sensitive information must not be passed in that way.

**Do POST :** The POST method overcomes the limit imposed by the GET request, by sending the values of the request inside its body. Also, there is no limitations on the number of values to be sent across. Finally, the sensitive information passed through a POST request is not visible to an external client.

### **174. What is meant by a Web Application ?**

A Web application is a dynamic extension of a Web or application server. There are two types of web applications: presentation oriented and service-oriented. A presentation-oriented Web application generates interactive web pages, which contain various types of markup language and dynamic content in response to requests. On the other hand, a service-oriented web application implements the endpoint of a web service. In general, a Web application can be seen as a collection of servlets installed under a specific subset of the server's URL namespace.

### **175. What is a Server Side Include (SSI) ?**

Server Side Includes (SSI) is a simple interpreted server-side scripting language, used almost exclusively for the Web, and is embedded with a servlet tag. The most frequent use of SSI is to include the contents of one or more files into a

Web page on a Web server. When a Web page is accessed by a browser, the Web server replaces the servlet tag in that Web page with the hypertext generated by the corresponding servlet.

### **176. What is Servlet Chaining ?**

Servlet Chaining is the method where the output of one servlet is sent to a second servlet. The output of the second servlet can be sent to a third servlet, and so on. The last servlet in the chain is responsible for sending the response to the client.

### **177. How do you find out what client machine is making a request to your servlet ?**

The Servlet Request class has functions for finding out the IP address or host name of the client machine. Get Remote Addr() gets the IP address of the client machine and get Remote Host() gets the host name of the client machine.

### **178. What is the structure of the HTTP response ?**

The HTTP response consists of three parts:

- **Status Code:** describes the status of the response. It can be used to check if the request has been successfully completed. In case the request failed, the status code can be used to find out the reason behind the failure. If your servlet does not return a status code, the success status code, Http Servlet Response. SC\_OK, is returned by default.
- **HTTP Headers:** they contain more information about the response. For example, the headers may specify the date/time after which the response is considered state, or the form of encoding used to safely transfer the entity to the user. See how to retrieve headers in Servlet here.
- **Body:** it contains the content of the response. The body may contain HTML code, an image, etc. The body consists of the data bytes transmitted in an HTTP transaction message immediately following the headers.

### **179. What is a cookie ? What is the difference between session and cookie ?**

A cookie is a bit of information that the Web server sends to the browser. The browser stores the cookies for each Web server in a local file. In a future request, the browser, along with the request, sends all stored cookies for that specific Web server. The differences between session and a cookie are the following:

- The session should work, regardless of the settings on the client browser. The client may have chosen to disable cookies. However, the sessions still work, as the client has no ability to disable them in the server side.
- The session and cookies also differ in the amount of information they can store. The HTTP session is capable of storing any Java object, while a cookie can only store String objects.

### **180. Which protocol will be used by browser and servlet to communicate ?**

The browser communicates with a servlet by using the HTTP protocol.

### **181. What is HTTP Tunneling ?**

HTTP Tunneling is a technique by which, communications performed using various network protocols are encapsulated using the HTTP or HTTPS protocols. The HTTP protocol therefore acts as a wrapper for a channel that the network protocol being tunneled uses to communicate. The masking of other protocol requests as HTTP requests is HTTP Tunneling.

### **182. What's the difference between send Redirect and forward methods ?**

The send Redirect method creates a new request, while the forward method just forwards a request to a new target. The previous request scope objects are not available after a redirect, because it results in a new request. On the other hand, the previous request scope objects are available after forwarding. Finally, in general, the send Redirect method is considered to be slower compared to the forward method.

### **182. What is URL Encoding and URL Decoding ?**

The URL encoding procedure is responsible for replacing all the spaces and every other extra special character of a URL, into their corresponding Hex representation. In correspondence, URL decoding is the exact opposite procedure.

**183. What is a JSP Page ?**

A Java Server Page (JSP) is a text document that contains two types of text: static data and JSP elements. Static data can be expressed in any text-based format, such as HTML or XML. JSP is a technology that mixes static content with dynamically generated content. See [JSP example here](#).

**184. How are the JSP requests handled ?**

On the arrival of a JSP request, the browser first requests a page with a .jsp extension. Then, the Web server reads the request and using the JSP compiler, the Web server converts the JSP page into a servlet class. Notice that the JSP file is compiled only on the first request of the page, or if the JSP file has changed. The generated servlet class is invoked, in order to handle the browser's request. Once the execution of the request is over, the servlet sends a response back to the client. See [how to get Request parameters in a JSP](#).

**185. What are the advantages of JSP ?**

The advantages of using the JSP technology are shown below:

- JSP pages are dynamically compiled into servlets and thus, the developers can easily make updates to presentation code.
- JSP pages can be pre-compiled.
- JSP pages can be easily combined to static templates, including HTML or XML fragments, with code that generates dynamic content.
- Developers can offer customized JSP tag libraries that page authors access using an XML-like syntax.
- Developers can make logic changes at the component level, without editing the individual pages that use the application's logic.

**186. What are Directives ? What are the different types of Directives available in JSP ?**

Directives are instructions that are processed by the JSP engine, when the page is compiled to a servlet. Directives are used to set page-level instructions, insert data from external files, and specify custom tag libraries. Directives are

defined between `< %@` and `% >`. The different types of directives are shown below:

- Include directive: it is used to include a file and merges the content of the file with the current page.
- Page directive: it is used to define specific attributes in the JSP page, like error page and buffer.
- Tag lib: it is used to declare a custom tag library which is used in the page.

### **187. What are JSP actions ?**

JSP actions use constructs in XML syntax to control the behavior of the servlet engine. JSP actions are executed when a JSP page is requested. They can be dynamically inserted into a file, re-use JavaBeans components, forward the user to another page, or generate HTML for the Java plugin. Some of the available actions are listed below:

- `jsp : include` - includes a file, when the JSP page is requested.
- `jsp : useBean` - finds or instantiates a JavaBean.
- `jsp : setProperty` - sets the property of a JavaBean.
- `jsp : getProperty` - gets the property of a JavaBean.
- `jsp : forward` - forwards the requester to a new page.
- `jsp : plugin` - generates browser-specific code.

### **188. What are Scriptlets ?**

In Java Server Pages (JSP) technology, a scriptlet is a piece of Java-code embedded in a JSP page. The scriptlet is everything inside the tags. Between these tags, a user can add any valid scriptlet.

### **189. What are Declarations ?**

Declarations are similar to variable declarations in Java. Declarations are used to declare variables for subsequent use in expressions or scriptlets. To add a declaration, you must use the sequences to enclose your declarations.

**190. What are Expressions ?**

A JSP expression is used to insert the value of a scripting language expression, converted into a string, into the data stream returned to the client, by the web server. Expressions are defined between `<% =` and `%>` tags.

**191. What is meant by implicit objects and what are they ?**

JSP implicit objects are those Java objects that the JSP Container makes available to developers in each page. A developer can call them directly, without being explicitly declared. JSP Implicit Objects are also called pre-defined variables. The following objects are considered implicit in a JSP page:

- application
- page
- request
- response

**192. Which package has light weight components?**

javax.Swing package. All components in Swing, except JApplet, JDialog, JFrame and JWindow are lightweight components.

### Java programs

1. *Check if number is odd or even.*
2. *Factorial of a number.*
3. *To find Fibonacci series for 1<sup>st</sup> ten number or within the range 100 using for loop.*
4. *To find Fibonacci series for 1<sup>st</sup> ten number or within the range 100 using while loop.*
5. *To check given number is prime number or not.*
6. *To check given number is prime number or not ( range of input ).*
7. *To find sum of digits of a given number.*
8. *To check whether given number is a Armstrong no or not.*
9. *To check given number is strong or not.*
10. *To check or count how many Binary digit are present in given number.*
11. *To count the number of digits in a given number*
12. *Reverse a given number.*
13. *To check whether the given number is palindrome or not.*
14. *To print the tables .*
15. *To find the power of a number.*
16. *To compute the quotient and remainder.*
17. *To find the simple interest.*
18. *To find the compound interest.*
19. *To reverse a String using for loop.*
20. *To reverse a String using while loop.*
21. *To reverse a String without using loop.*
22. *To check whether a String is palindrome or not.*
23. *To accept a character , determine whether the character is a lowercase or uppercase..*
24. *To find the area and circumference of the circle.*
25. *To convert days into years , months and days.*
26. *To find grade of the student.*



27. *To find the largest of two numbers.*
28. *To find the largest , smallest and second largest of three numbers.*
29. *To check whether that year is leap year or not.*
30. *To find the square and cube of a number .*
31. *To convert the temperature in Fahrenheit into Celsius*
32. *To convert seconds into hours , minutes and seconds.*
33. *To find the area of the triangle for 3 sides.*
34. *Swap two numbers using 3 rd. variable.*
35. *Swap two numbers without using 3 rd. variable.*
36. *To sort an array in ascending order (Bubble sort).*
37. *write a program to generate capca or OTP.*
38. *To generate Random numbers.*
39. *To get the IP Address*
40. *To print the number in pyramid shape.*
41. *To find the GCD of a number.*
42. *To find missing number from the array.*
43. *To find Natural number.*
44. *To find the perfect square.*
45. *To find whether the number is positive or negative .*
46. *Addition of 2 matrices.*
47. *multiplication of 2 matrices.*
48. *write a program for linear search algorithm or count how many times the character is repeated in a given string.*
49. *To find the character position in the String.*
50. *To replace char 'A' with 'O' in the given String.*

**1.     *Check if number is odd or even.***

```
class Oddeven
{
public static void main(String[]args)
{
int n ;
Scanner sc = new Scanner(System.in);
System.out.println("Enter the number : ");
n = sc . nextInt();
if(n%2==0)
{
System.out.println("It is an even number : " +n);
}
else
{
System.out.println("It is an odd number : " +n);
}
}
}
```

**2.     *Factorial of a number.***

```
class Factorial
{
public static void main(String[] args)
{
int n ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the number : ");
n = sc . nextInt ( ) ;
int fact=1;
for(int i=n; i >=1;i--)
{
fact=fact* i ;
}
System.out.println("Factorial of a number : " + fact);
}
}
```

**3.     *To find Fibonacci series for 1<sup>st</sup> ten number or within the range 100 using for loop.***

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

```
{
int a=0,b=1;
System.out.print(a+ " " +b+ " ");
for(int i=1;i<=10;i++)
{
int c=a + b;
System.out.print(c+ " ");
a=b;
b=c;
}
}
}
```

**4.     *To find Fibonacci series for 1<sup>st</sup> ten number or within the range 100 using while loop.***

```
class Fibonacci2
{
public static void main(String[] args)
{
int a=0,b=1;
System.out.print(a+ " " +b+ " ");
int i=1;
while(i<=10)
{
int c= a +b;
System.out.print(c+ " ");
a=b;
b=c;
i++;
}
}
}
```

**5.     *To check given number is prime number or not.***

```
class Primenol
{
public static void main(String[] args)
{
int n ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the number: ");
n = sc . nextInt ( ) ;
boolean flag=true;
for(int i=2;i<n; i++)
```

```
{
if( n % i==0 )
{
flag=false;
break;
}
}
if(flag==true)
{
System.out.println("It is a prime number : " + n);
}
else
{
System.out.println("It is not a prime number : " + n);
}
}
}
```

**6.     *To check given number is prime number or not ( range of input ).***

```
class Primeno2
{
public static void main(String[] args)
{
for(int k=2;k<=100;k++)
{
int n=k;
boolean flag=true;
for(int i=2;i<n ; i++)
{
if(n % i ==0)
{
flag=false;
break;
}
}
if(flag==true)
{
System.out.println("It is a prime number : " +n);
}
}
}
}
```

**7.     *To find sum of digits of a given number.***

```
class Sum
{
public static void main(String[] args)
{
int n ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the number : ");
n = sc . nextInt ( ) ;
int sum=0;
while(n!=0) {
int rem = n % 10 ;
sum = sum + rem ;
n = n / 10;
}
System.out.println("Sum of a number : " +sum);
}
}
```

### **8.     *To check whether given number is a Armstrong no or not***

```
class Armstrongno
{
public static void main(String[] args)
{
int n ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the number : ");
n = sc . nextInt ( ) ;
int copy=n;
int sum=0;
while(n!=0)
{
int rem = n % 10 ;
sum = sum + ( rem * rem * rem ) ;
n = n / 10;
}
if(sum == copy)
{
System.out.println("It is Armstrong number : "+copy);
}
else
{
System.out.println("It is not Armstrong number : "+copy);
}
}
```

```
}
```

**9.     *To check given number is strong or not.***

```
class Strongno
{
public static void main(String[]args)
{
int n ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the number : ");
n = sc . nextInt ( ) ;
int sum=0;
int fact=1;
int copy=n;
while(n!=0) {
int rem=n%10;
for(int i=rem ; i>=1;i--)
{
fact=fact*i;
}
sum=sum + rem;
n=n/10;
}
if(copy==sum) {
System.out.println("It is Strong no : " +copy);
}
else
{
System.out.println("It is not Strong no : " +copy);
}
}
}
```

**10.    *To check or count how many Binary digit are present in given number.***

```
class Binarycount
{
public static void main(String[]args)
{
int n ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the number : ");
n = sc . nextInt ( ) ;
int count=0;
while(n!=0)
```

```
{
int rem=n%10;
if(rem==0 || rem==1)
{
count++;
}
n=n/10;
}
System.out.println(count);
}
}
```

### ***11. To count the number of digits in a given number.***

```
class Digitcount
{
public static void main(String[] args)
{
int n ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the number : ");
n = sc . nextInt ( ) ;
int count=0;
while(n!=0)
{
n=n/10;
count++;
}
System.out.println("count of a number : " +count);
}
}
```

### ***12. Reverse a given number.***

```
class Reverseno
{
public static void main(String[]args)
{
int n ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the number : ");
n = sc . nextInt ( ) ;
int rev=0;
while(n!=0)
{
int rem=n%10;
```

```
rev=rev*10+rem;
n=n/10;
}
System.out.println("reverse of the number is : " + rev);
}
}
```

### ***13. To check whether the given number is palindrome or not.***

```
class Palindromeno
{
public static void main(String[]args)
{
int n ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the number : ");
n = sc . nextInt ( ) ;
int rev=0;
int copy=n;
while(n!=0)
{
int rem=n%10;
rev=rev*10+rem;
n=n/10;
}
if(copy==rev)
{
System.out.println("palindrome number is : " +copy);
}
else
{
System.out.println("Not palindrome number is : " +copy);
}
}
}
```

### ***14. To print the tables .***

```
class Tables
{
public static void main(String []args)
{
int n ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the number : ");
n = sc . nextInt ( ) ;
```



```
for(int i=1;i<=10;i++)
{
System.out.println(n+"*"+i+"="+ (n*i));
}
}
}
```

**15. To find the power of a number.**

```
class Powerno
{
public static void main(String[] args)
{
int n , p ;
Scanner sc = new Scanner(System.in);
System.out.println("Enter the number : ");
n = sc.nextInt();
System.out.println("Enter the power : ");
p = sc.nextInt();
double result = Math.pow(n, p);
System.out.println(n+"^"+p+"="+result);
}
}
```

**16. To compute the quotient and remainder.**

```
class Quetrem
{
public static void main(String[] args)
{
int a , b ;
Scanner sc = new Scanner(System.in);
System.out.println("Enter the value for a and b : ");
a = sc.nextInt();
b = sc.nextInt();
int quot = a / b ;
int rem = a % b ;
System.out.println("Quotient : "+quot);
System.out.println("Remainder : "+rem);
}
}
```

**17. To find the simple interest.**

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

```
{
float p ,t ,r ,Si;
Scanner sc = new Scanner(System.in);
System.out.println("Enter the principal : ");
p = sc . nextFloat();
System.out.println("Enter the Time period  : ");
t = sc . nextFloat();
System.out.println("Enter the Rate of interest : ");
r = sc . nextFloat();
Si = (p*t*r)/100;
System.out.println("Simple interest is : " +Si);
}
}
```

### ***18. To find the compound interest.***

```
class Compoundinterest
{
public static void main(String[] args)
{
int p , t , n ;
double r ;
Scanner sc = new Scanner(System.in);
System.out.println("Enter the principal : ");
p = sc . nextInt();
System.out.println("Enter the Time period  : ");
t = sc . nextInt();
System.out.println("Enter the Rate of interest : ");
r = sc . nextDouble();
System.out.println("Enter the number :");
n= sc.nextInt();
double amount = p * Math.pow(1 + (r / n), n * t);
double compinterest = amount - p;
System.out.println("Compound Interest after " + t + " years :
"+compinterest);
System.out.println("Amount after " + t + " years : "+amount);
}
}
```

### ***19. To reverse a String using for loop.***

```
class Reversestring1
{
public static void main(String[] args)
{
String s1 ;
```

```
Scanner sc = new Scanner(System.in);
System.out.println("Enter the String  :");
s1=sc.nextLine();
String s2 = " ";
for(int i=s1.length()-1;i>=0;i--)
{
s2 = s2 + s1.charAt(i) ;
}
System.out.println("reverse String is : "+s2) ;
}
}
```

**20. To reverse a String using while loop.**

```
class Reversestring2
{
public static void main(String[] args)
{
String s1 ;
Scanner sc = new Scanner(System.in);
System.out.println("Enter the String  :");
s1=sc.nextLine();
String s2 = " ";
int i=s1.length()-1;
while (i>=0)
{
s2 = s2 + s1.charAt(i) ;
i--;
}
System.out.println("reverse String is : "+s2 ) ;
}
}
```

**21. To reverse a String without using loop.**

```
class Reversestring3
{
static String s1="java";
static String s2 = " ";
public static void main(String[] args)
{
int x=s1.length()-1;
disp(x);
System.out.println(s2) ;
}
static void disp(int n)
{
```

```
if( n >= 0 )
{
s2 = s2 + s1.charAt(n) ;
n--;
disp(n);
}
}
}
```

**22. To check whether a String is palindrome or not.**

```
class Palindromestring
{
public static void main(String[] args)
{
String s1 ;
Scanner sc = new Scanner(System.in);
System.out.println("Enter the String :");
s1=sc.nextLine();
String s2 = " ";
for(int i=s1.length()-1;i>=0;i--)
{
s2 = s2 + s1.charAt(i);
}
if(s1.equals(s2))
{
System.out.println("It is a palindrome : "+s2) ;
}
else
{
System.out.println("It is not a palindrome : "+s2) ;
}
}
}
```

**23. To accept a character , determine whether the character is a lowercase or uppercase.**

```
class Character
{
public static void main(String[] args)
{
char ch='R';
if(ch>='A' && ch<='Z')
{
System.out.println("It is a uppercase character : " + ch);
}
```

```
}  
else if(ch>='a' && ch<='z')  
{  
System.out.println("It is a lowercase character : " + ch);  
}  
}  
}
```

#### **24. To find the area and circumference of the circle.**

```
class Circle  
{  
public static void main(String[] args)  
{  
int r ;  
Scanner sc = new Scanner ( System .in ) ;  
System.out.println("Enter the number : ");  
r = sc . nextInt ( ) ;  
final double pi=3.142;  
double area = pi*r*r;  
double circum = 2*pi*r;  
System.out.println("Area of the Circle : " + area);  
System.out.println("Circumference of the Circle : " + circum);  
}  
}
```

#### **25. To convert days into years , months and days.**

```
class Days  
{  
public static void main(String[] args)  
{  
int totaldays ;  
int days , months , years;  
Scanner sc = new Scanner(System .in ) ;  
System.out.println("Enter the totaldays : ");  
totaldays = sc . nextInt();  
years = totaldays/365;  
totaldays = totaldays%365;  
months = totaldays/30;  
days = totaldays%30;  
System.out.println("Years : " + years);  
System.out.println("Months : " + months);  
System.out.println("Days : " + days);  
}  
}
```

**26. To find grade of the student.**

```
class Grade
{
public static void main(String[] args)
{
int marks ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the marks : ");
marks = sc . nextInt ( ) ;
if(marks>=85 && marks<=100)
System.out.println("Distinction : " + marks);
else if (marks>=60)
System.out.println("First class : " + marks);
else if(marks>=50)
System.out.println("Second class : " + marks);
else if(marks>=35)
System.out.println("Pass : " + marks);
else
System.out.println("Fail : " + marks);
}
}
```

**27. To find the largest of two numbers.**

```
class Largest2no
{
public static void main(String[] args)
{
int a , b ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the value of a and b : ");
a = sc.nextInt() ;
b = sc.nextInt() ;
int large=a;
if(b>large)
large=b;
System.out.println("Largest number is : " + large);
}
}
```

**28. To find the largest , smallest and second largest of three numbers.**

```
class Largest3no
{
public static void main(String[] args)
```

```
{
int a , b , c ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the value of a , b and c : ");
a = sc.nextInt();
b = sc.nextInt();
c = sc.nextInt();
int largest=a;
int smallest=a;
if(b>largest)
largest=b;
if(c>largest)
largest=c;
if(b<smallest)
smallest=b;
if(c<smallest)
smallest=c;
int seclargest=(a + b + c) - (largest + smallest);
System.out.println("Largest number is : " + largest);
System.out.println("Second largest number is : " + seclargest);
System.out.println("Smallest number is : " + smallest);
}
}
```

**29. To check whether that year is leap year or not.**

```
class Leapyear
{
public static void main(String[] args)
{
int year ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the year : ");
year = sc . nextInt ( ) ;
if(year%4 == 0 && year!=100 || year%400 == 0)
System.out.println("It is a Leap year : " + year);
else
{
System.out.println("It is not a Leap year : " + year);
}
}
}
```

**30. To find the square and cube of a number .**

```
class sqrcube
{
public static void main(String[] args)
{
int a ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the number :");
a = sc . nextInt ( ) ;
int square = a * a;
int cube = a * a * a;
System.out.println("Square of the number : " +square);
System.out.println("Cube of the number : " +cube);
}
}
```

### ***31. To convert the temperature in Fahrenheit into Celsius***

```
class Temperature
{
public static void main(String[] args)
{
double Fahren ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the Fahrenheit :");
Fahren = sc . nextDouble();
double Celsius ;
Celsius =((5.0 / 9.0) * Fahren - 32.0);
System.out.println("Celsius : " +Celsius);
}
}
```

### ***32. To convert seconds into hours , minutes and seconds.***

```
class Time
{
public static void main(String[] args)
{
int totalseconds ;
int seconds , minutes , hours ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the totalseconds :");
totalseconds = sc . nextInt() ;
seconds = totalseconds;
hours = seconds/3600 ;
seconds = seconds%3600 ;
minutes = seconds/60 ;
}
```



```
seconds = seconds%60 ;
System.out.println("Total seconds : " + totalseconds);
System.out.println("Hours : " + hours);
System.out.println("Minutes : " + minutes);
System.out.println("Seconds : " + seconds);
}
}
```

**33.    *To find the area of the triangle for 3 sides.***

```
class Triangle
{
public static void main(String[] args)
{
int s1 , s2 , s3 ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the value of s1 , s2 and s3 :");
s1 = sc . nextInt ( ) ;
s2 = sc . nextInt ( ) ;
s3 = sc . nextInt ( ) ;
int s=(s1+s2+s3)/2;
int area = (s*(s-s1)*(s-s2)*(s-s3));
System.out.println("Area of a Triangle : " + area);
}
}
```

**34.    *Swap two numbers using 3 rd. variable.***

```
class Swap1
{
public static void main(String[] args)
{
int a ,b ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the value of a and b : ");
a = sc . nextInt ( ) ;
b = sc . nextInt ( ) ;
int temp=a;
a=b;
b=temp;
System.out.println("After swapping the value of a is : "+a);
System.out.println("After swapping the value of b is : "+b);
}
}
```

**35.    *Swap two numbers without using 3 rd. variable.***

```
class Swap2
{
public static void main(String[] args)
{
int a ,b ;
Scanner sc = new Scanner ( System .in ) ;
System.out.println("Enter the value of a and b : ");
a = sc . nextInt () ;
b = sc . nextInt () ;
a = a + b;
b = a - b;
a = a - b;
System.out.println("After swapping the value of a is : "+a);
System.out.println("After swapping the value of b is : "+b);
}
}
```

**36. To sort an array in ascending order (Bubble sort).**

```
class Bubblesort
{
public static void main(String[] args)
{
int [ ] arr = { 8 , 7 , 5 , 9 , 2 , 10 };
int n=arr.length-1;
for( int i=1;i<n; i++)
{
for( int j=1;j<n; j++)
{
if ( arr[ j - 1 ] > arr[ j ] )
{
int temp = arr [ j-1 ] ;
arr [ j-1 ] = arr [ j ] ;
arr [ j ] = temp ;
}
}
}
for( int i=0 ; i<arr.length; i++)
{
System.out.println( arr[ i ]+ " ");
}
}
}
```

**37. write a program to generate capca or OTP.**

```
class OTP
{
public static void main(String[] args)
{
String s1="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
String s2= s1.toLowerCase();
String s3= "123456789";
String s4 =s1+s2+s3;
Random r = new Random ();
char [] pwd = new char[5];
for(int i=0;i<5;i++)
{
pwd[i] = s4.charAt(r. nextInt(s4.length()));
}
for(int i=0;i<5;i++)
{
System.out.println(pwd[i]);
}
}
}
```

### **38. To generate Random numbers.**

```
class GenerateRandom
{
public static void main(String[] args)
{
Random rm = new Random();
System.out.println("Random numbers are : ");
System.out.println("*****");
for(int i=1;i<=5;i++)
{
System.out.println(rm.nextInt(10));
}
}
}
```

### **39. To get the IP Address**

```
class GetMyIPAddress
{
public static void main(String[] args) throws
UnknownHostException
{
InetAddress myIP = InetAddress.getLocalHost();
System.out.println("My IP address is : ");
}
```

```
System.out.println(myIP.getHostAddress());
}
}
```

**40. To print the number in pyramid shape.**

```
class Pyramidshape
{
public static void main(String[] args)
{
int n ;
Scanner sc = new Scanner(System.in);
System.out.println("Enter the number : ");
n=sc.nextInt();
for(int i=1;i<n;i++)
{
for(int j=1;j<=n;j++)
{
System.out.print(" ");
}
for(int k=1;k<=i;k++)
{
System.out.print(" "+k+ " ");
}
for(int m=n-1;m>0;m--)
{
System.out.print(" "+m+ " ");
}
System.out.println();
}
}
}
```

**41. To find the GCD of a number.**

```
class Gcd
{
public static void main(String[] args)
{
int a , b ;
Scanner sc = new Scanner(System.in);
System.out.println("Enter the value of a and b : ");
a = sc.nextInt();
b = sc.nextInt();
while(a != b)
{

```

```
if(a > b)
a = a - b ;
else
b = b - a ;
}
System.out.println("GCD of given numbers is : " +b);
}
}
```

#### ***42. to find missing number from the array.***

```
class Missingnum
{
public static void main(String[] args)
{
int [] arr1 = {7,5,6,1,4,2} ;
System.out.println("Missing number from array arr1 :
"+missingNumber(arr1));
int [] arr2 = {5,3,1,2};
System.out.println("Missing number from array arr2 :
"+missingNumber(arr2));
}
public static int missingNumber (int[]arr)
{
int n = arr.length+1;
int sum = n*(n+1)/2;
int remSum = 0;
for(int i=0;i<arr.length;i++)
{
remSum+= arr[i];
}
int missingNumber = sum -remSum;
return missingNumber;
}
}
```

#### ***43. To find Natural number.***

```
class Naturalno
{
public static void main(String[] args)
{
int n ,sum = 0 ;
Scanner sc = new Scanner(System.in);
System.out.println("Enter the number: ");
n = sc.nextInt();
}
```

```
for(int i=0;i<=n;i++)
{
sum = sum + i ;
}
System.out.println("Sum of natural numbers is : " +sum);
}
}
```

#### ***44. To find the perfect square.***

```
class Perfectsquare
{
static boolean checkPerfectSquare(double x)
{
double sq = Math.sqrt(x);
return ((sq-Math.floor(sq))==0);
}
public static void main(String[] args)
{
double num ;
Scanner sc = new Scanner(System.in);
System.out.println("Enter the number : ");
num = sc.nextDouble();
if(checkPerfectSquare(num))
System.out.println(num+" is a perfect square number");
else
System.out.println(num+" is not a perfect square number");
}
}
```

#### ***45. To find whether the number is positive or negative .***

```
class Posneg
{
public static void main(String[] args)
{
int num ;
Scanner sc = new Scanner(System.in);
System.out.println("Enter the number : ");
num = sc.nextInt();
if(num>0)
{
System.out.println(num+ " is a positive number");
}
if(num<0)
```

```
{
System.out.println(num+ " is a negative number");
}
}
}
```

#### **46.    *Addition of 2 matrices.***

```
class Add2matrix
{
public static void main(String[] args)
{
int rows , cols , c ,d ;
Scanner sc = new Scanner (System.in);
System.out.println("Enter the number for rows and columns : ");
rows = sc.nextInt();
cols = sc.nextInt();
int a [][] = new int [rows][cols] ;
int b [][] = new int [rows][cols] ;
int sum [][] = new int [rows][cols] ;
System.out.println("Enter the elements of 1st matrix : ");
for(c=0;c<rows;c++)
for(d=0;d<cols;d++)
a[c][d] = sc .nextInt();
System.out.println("Enter the elements of 2nd matrix : ");
for(c=0;c<rows;c++)
for(d=0;d<cols;d++)
b[c][d] = sc .nextInt();
for(c=0;c<rows;c++)
for(d=0;d<cols;d++)
sum[c][d] = a[c][d] + b[c][d] ;
System.out.println("Sum of the matrices : ");
for(c=0;c<rows;c++)
{
for(d=0;d<cols;d++)
System.out.println(sum[c][d]+ "\t");
}
}
}
```

#### **47.    *multiplication of 2 matrices.***

```
class Multiply2matrix
{
public static void main(String[] args)
{
```

```
int m , n , p , q , sum = 0 , c , d , k ;
Scanner sc = new Scanner (System.in);
System.out.println("Enter the number for rows and columns of 1st
matrix : ");
m = sc.nextInt();
n = sc.nextInt();
int a [][] = new int [m][n] ;
System.out.println("Enter the numbers of 1st matrix : ");
for(c=0;c<m;c++)
for(d=0;d<n;d++)
a[c][d] = sc .nextInt();
System.out.println("Enter the number for rows and columns of 2nd
matrix : ");
p = sc .nextInt();
q = sc .nextInt();
if (n!=p)
System.out.println("matrices entered order can't be multiplied
with each other");
else
{
int b [][] = new int [p][q];
int multiply[][] = new int[m][q];
System.out.println("Enter the elements of 2nd matrix : ");
for(c=0;c<p;c++)
for(d=0;d<q;d++)
b[c][d] = sc.nextInt();
for(c=0;c<m;c++)
{
for(d=0;d<q;d++)
{
for(k=0;k<p;k++)
{
sum = sum + a[c][k] * b[k][d] ;
}
multiply[c][d] = sum;
sum=0;
}
}
System.out.println("Multiplication of matrices: ");
for(c=0;c<m;c++)
{
for(d=0;d<q;d++)
System.out.println(multiply[c][d]+"\\t");
System.out.println("\\n");
}
```

---



```
}  
}  
}  
}
```

**48.    *write a program for linear search algorithm or count how many times the character is repeated in a given string.***

```
class Linersearch  
{  
public static void main(String[] args)  
{  
String str ;  
Scanner sc = new Scanner(System.in);  
System.out.println("Enter the String : ");  
str = sc.nextLine();  
int count=0;  
char[]arr=str.toCharArray();  
for (int i=0;i<arr.length; i++)  
{  
if(arr[i]=='a')  
{  
count++;  
}  
System.out.println(count);  
}  
}  
}
```

**49.    *To find the character position in the String.***

```
class CharString  
{  
public static void main(String[] args)  
{  
String str ;  
Scanner sc = new Scanner(System.in);  
System.out.println("Enter the String : ");  
str = sc.nextLine();  
for(int i=0;i<str.length();i++)  
{  
char ch = str.charAt(i);  
System.out.println("Character at "+i+" Position : " +ch);  
}  
}
```

```
}
```

**50. To replace char 'A' with 'O' in the given String.**

```
class Replacechar
{
public static void main(String[] args)
{
String s1 ="java";
String s2 = " ";
char [] arr = s1.toCharArray();
for (int i=0;i<arr.length;i++)
{
if(arr[i]=='a')
{
s2 = s2 + 'o' ;
}
else
{
s2 = s2 + arr[i];
}
}
System.out.println(s2);
}
}
```