18/JAN/2022

JAVA

* **Class :-** → Class is the collection of object.
→ Class is not a real world Entity. it is just a templet of blue print.
→ class does not occupy the memory.

Syntax :- access-modifier class classname
{
- Methods
- Constructors
- fields
- nested class
}

Example :-

```
public class Demo {

    public void abc() {

        System.out.println("This is abc method");
    }
    public void pqr() {

        System.out.println("This is pqr method");
    }
    public static void main(String[] args) {
        Demo d = new demo();
        d.abc(); // This is abc method
        d.pqr(); // This is pqr method
    }
}
```

Syntax :- static method :- access modifier static key word written type methodname.

* **Method :-** A set of code which perform a particular task.

Syntax :- access-modifier return type methodname (list of parameters)
{
Statements
}

Example :-

```
public void pqr() {

    System.out.println("This is pqr method");
}
```

* **Object :-** Object is an instance of class
Object is a real world entity
Object occupies memory

Syntax :- Classname object = new classname();
object.method name();

Example :-

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

    Demo d = new Demo();
    d.abc(); This is abc method
    d.pqr(); This is pqr method
}
```

* Data Types :-

Data type is a classification of the type of data that a variable or object can hold in a computer programming.

Ex:- char, int, float, string etc.

Java support explicit declaration of the data types.

syntax:- data type variable name

ex:- int a; (OR) int a = 100;

Two type of data type

1. Primitive data type.
2. Non primitive data type.

1. Primitive data type.

* Integer data type.

a. byte (8 bit) b. short (16 bits)

byte a = 10; short a = 87;

variable 2^{16} =

$2^8 = 256$

c. int (32 bits) d. long (64bit) (2^{64})

(2^{32})

int a = 10000; long d = 1000000;

* Rational data type.

e. Float (32 bit) f. double (64 bit)

float a = 1.2f; double b = 19.6565;

* Characters

g. char (16 bit) h.

char a = 'z';

* Conditional

h. Boolean (1 bit)

Boolean
a = true;
boolean
b = false;

2. Non Primitive data type

A. Array B. String.

* Normal method → local variable
* main method → instance variable
* static int → static variable.

* Variables :- A variable is a container which holds the value while the java prog. is executed. A variable is assigned with a data type.

* Variable is a name of memory location.
There are three types of variables in java local, instance & static.

① Local Variable :-

A variable declared inside the body of the method is called local variable. you can use this variable only within that method & the other methods in the class aren't even aware that the variable exists

② Instance Variable :-

A variable declared inside the class but outside the body of the method, is called an instance variable

③ Static variable :-

A variable that is declared as static is called a static variable.

* method default असेल तर सगळे क्लासेस access कर सकते
* variable ko private banate hai to usko method usko declare karna padenga.

ex:-
```
public class A
{
static int m = 100; // static variable
void method()
{
int n = 90; // local variable
}
public static void main (String args[])
{
int data = 50; // instance variable
}
} // end of class
```

* Private access modifier.

ki se bhi variable ko ya kisi bhi method ko private bana diya usa access sirf uske class ke liye hi limited rahenga.
usko hum class ke bhar se access nahi kar sakte.

* Default access modifier

The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

अगर हम method ham protected banate hai to us method ka same class me access ho hi jata hai same package me bhi hojata hai & out side package me bhi hajata hai

* Protected access modifier

The access level of a protected modifier is within the package and outside the package through child class.

If you do not make the child class, it cannot be accessed from outside the package

ex:-
```
package Demo Package 2;
import Demo Package.Democlass;
public class Test 2 extends Democlass{
    public static void main(String[] aegs){
    Test2 t2 = new Test2();
    t2.addation ();
    }
}
```

* Public access modifier

Same class में access कर सकते हैं
Same package, & out side the package access कर सकते हैं without inheritance.

```
Package Demo Package 2;
import Demo Package. Demo class;
public class Test2 {
public static void main (String[]args){
    Democlass dc4 = new Democlass();
    dc4. addation ();
}
```

लेकिन inheritance को use करना पड़ेगा, class को sub class करना पड़ेगा, जब हम class me extends कर ते हैं parent class का test2 child class बनजाएगा.

→ The access level of a Public modifier is everywhere. It can be accessed from within the class, outside the class, within the package & out side the package.

① Private Access modifier

ex:-
```
package Access Modifier Demo;
class Private Access Modifier Demo {
Private int a = 7;
private void abc (){
System. out. print ln (a);
}
public static void main(String [] args){
Private Access Modifier Demo pd =
        new Private Access modifier Demo();
pd. abc (); //7
}
}
```

```
package Access Modifier Demo;
```

```
public class Private Access Modifier Demo{

private void pqr(){
System.out.print_ln("I am pqr method");
}

public static void main(String[] aEgs){
Private Access Modifier Demo pd1 =
    new Private Access Modifier Demo();
pd1.abc(); // Compile time EEEOE
}
}
```

② Default

* Default is access modifier, we can allocate it to the variable and method and class.
* It is accessible within same package

package Access Modifier Demo;

ex:-

```
package Access Modifier Demo;
class Private Access modifier Demo{
private int a = 7;
void abc(){
 System.out.print_ln(a);
}
public static void main(String[] args){
Private Access Modifier Demo pd = new
    Private Access modifier Demo();
pd.abc(); //7
}
}
```

```
package Access modifier Demo;
```

```
public class Private Access modifier
                        Demo 1 {
private void pqr(){
system.out.println("I am pqr method");
}
public static void main (String [] args){
Private Access modifier Demo pd1 =
    new Private access modifier Demo();
pd1.abc()
}
}
```

(3) Protected Access modifier.

* We can allocate it to the variable & method not for class.
* It is accessible within same package as well as outside the package by using inheritance package Acm. Demo

ex:-
```
package Access modifier Demo;
class Private Access Modifier Demo {
private int a = 7
protected void abc() {
System out. Print.ln (a);
}
public static void main (String [] args){
Private Access Modifier Demo pd = new
    Private Access Modifier Demo();
pd. abc(); //7
}
}
```

```
package access modifier Demo 1;
import Access modifier Demo. Private
                    access modifier Demo;
```

* public class Protected access modi demo
  extends Private access modifier demo {
public static void main (String [] args){
Protected access modi Demo pro = new
    Protected access modi Demo ();
pro. abc();
}
}

```
package Access Modifier Demo;

public class Private Access Modifier Demo {

private void pqr () {
System.out.Print ln ("I am pqr method");
}

public static void main (String [] args){

Private Access modifier Demo pd1 =

  new Private Access modifier demo ();

pd1. abc ();
}

}
```

④ Public

* Public is access modifier, we can allocate it to the variable, method & class.
* It is accessible within class, within package and out side the package.

Ex :-

```
package Access Modifier Demo;
class Private Access Modifier Demo {

Private Int a = 7;
private void abc () {
 System.out.Print ln (a);
}
public static void main (String [] args){

Private Access Modifier Demo pd =

  new Private Access Modifier Demo ();

pd. abc (); // 7

Private void abc () {
System.out. Print ln (a);
}
Public static void main (String [] args){

Private access modifier Demo pd = new

  Private Access Modifier Demo ();

pd. abc (); // 7

}
}
```

```
package Access modifiee Demo I;
import Access Modifiee Demo. Private Access
                Modifiee Demo;
Public class Public Access Modi Demo {
Public static void main (String [] args) {
Private Access Modifiee Demo pmd = new
    Private Access Modifier Demo ();
pmd. abc ();
System. out. println (pmd. a);
}
}
```

Access Modifier	Within class	Within package	outside pac. by sub class	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

(2) Non-Access Modifiee

* Static :-

Static is the non-access modifiee, use to make class variable and method as static.

If we make variable as a static this variable can use in static method, we can use it in normal method also.

But if we make any variable without static, this can not accessible in static method. If we make any class method as a static so for call this method not need to create an object, direct on class name we can call this method.

* Static मध्ये Non static variable जर घेतला तर तो Run होणार नाही. त्यामध्ये Static variable लागते

* Static ला call करायचे असेल तर Class Name नी call करू शकतो.

ex:- package nonAccessModifiers;

public class static Demo {

Static int a = 10;
int b = 20;

public static void abc() {
System.out.print(a); //10
System.out.print(b); // compile time error
}
public void pqr() {
System.out.print(a); //10
System.out.print(b); // 20
}
public static void main(String[] args) {
Static Demo sd = new Static Demo();
sd.abc(); //10
Static Demo.abc(); //10
sd.pqr(); //10
}
}

जी ची value ही final वापरते तिची value change नाही करू शकत.

* Final :-

Adding the final modifier to a variable declaration makes that variable unchangeable once it's initialized.

Ex:- package nonAccessModifiers;

Public class FinalDemo {

Public static void main(String args[]) {

int a;
int b;
final int c = 0;
a = 2;
b = 4;
// c = 5; compile time error
System.out.println(a); //2
System.out.println(b); //4
System.out.println(c); //0
}
}

method, class ya abstract ko bhi abstract bana sakte hai
Body nahi provide karenge

banana padega.

* Abstract :- (keyword)

Abstract is non-access modifier which is use to create abstract classes and abstract methods.

* We can make method as abstract, when we make any method as an abstract then we need to make this class as a abstract.

* abstract method do not having body, in next concrete class we extend the abstract class & in that we can provide body to this abstract method.

```
abstract class A
{
abstract void start() {
}
}
class B extends A {
void start (){
system.out.println ("Hello Java")
}
}
public static void main (String [] args){
B Obj = new B();
obj. start (); // Hello JAVA
}
```

"+"
Concade

* JAVA OPERATOR *

operators are symbols that perform operations on variables and values. for eg. (+) is an operator used for addition, while * is also an operator used for multiplication.
operators can be classified into 5 types

1. Arithmetic operators
2. Assignment operators
3. Relational operators
4. Logical operator

(1) Arithmetic operators :-

(+) Addition

(-) Subtraction

(*) Multiplication

(/) Division

(%) Modulo operation (Remainder after division)

Example:-

```
Class Main {
    public static void main (String [] args) {
    //declare variables
    int a = 12, b = 5 ;

    //addition operator
    System.out.println ("a+b =" + (a+b));

    //subtraction operator
    System.out.println ("a-b =" + (a-b));

    //multiplication operator
    System.out.println ("a*b =" + (a*b));

    //division operator

    System.out.println ("a/b =" + (a/b));

    //modulo operator
    System.out.println ("a%b =" + (a%b));
    }
}
```

output

```
a+b = 17
a-b = 7
a*b = 60
a/b = 2
a%b = 2
```

(2) Assignment Operators

The operator is used in java to assign values to variables.

eg. `int age;`
`age = 5;`

here, (=) is the assignment operator. It assigns the value on its right to the variable on its left. That is, 5 is assigned to the variable age

Operator	eg.	Equivalent to
=	`a = b;`	`a = b;`
+=	`a += b;`	`a = a+b;`
-=	`a -= b;`	`a = a-b;`
*=	`a *= b;`	`a = a*b;`
/=	`a /= b;`	`a = a/b;`
%=	`a %= b;`	`a = a%b;`

Example:-

```
Class Main {
public static void main (String [] args) {
//Create variables
int a=4;
int var;
// assign value using =
var = a;
System.out.println("var using =:"+var); //14

// assign value using +=
var += a;
System.out.println("var using +=:"+var); //18

// assign value *=
var *= a;
System.out.println("var using *=:"+var); //32
}
}
```

Output

```
var using =: 4
var using +=: 8
var using *=: 32
```

(3) Relational operators

Relational operators are used to check the relationship between two operands.

eg. // check is a less than b
a < b;

here, (<) operator is the relational operator. it checks if a is less than b or not it returns either true or false.

operator	Description	Eg.
==	Is Equal To	3==5 returns false
!=	Not Equal To	3!=5 returns true
>	Greater Than	3>5 returns false
<	Less Than	3<5 returns true
>=	Greater than or Equal to	3>=5 returns false
<=	Less than or equal to	3<=5 returns true

Example:- class Main {
public static void main (String[] args) {

int a=7, b= 11;

System.out.println ("a is"+ a+ "and b is" +b);

System.out.println (a==b); // false
System.out.println (a!=b); // true
System.out.println (a>=b); // false
System.out.println (a<b); // true
System.out.println (a>=b); // false
System.out.println (a<=b); // true

}
}

Java logical operators

logical operators are used to check whether an expression is true or false
They are used in decision making.

Operator	Example	Meaning
&& (logical AND)	expression 1 && expression 2	true only if both exp.1 & exp 2 are true
|| (logical OR)	exp1 || exp2	true if either exp1 or exp2 is true
! (logical NOT)	! exp	true if exp. is false & vice versa

Example:- Class Main {
public static void main (String [] args) {
// && operator
System.out.println ((5>3) && (8>5)); // T
System.out.println ((5>3) && (8<5)); // F

// || operator
System.out.println ((5<3) || (8>5)); // T
System.out.println ((5>3) || (8<5)); // T
System.out.println ((5<3) || (8<5)); // F

// ! operator
System.out.println (!(5==3)); // true
System.out.println (!(5>3)); // false
}
}

Output:-

* (5>3) && (8>5) returns true because both (5>3) and (8>5) are true
* (5>3) && (8<5) returns false because the expression (8<5) is false
* (5>3) || (8>5) returns true because the expression (8>5) is true.
* (5>3) && (8>5) returns true because the expression (5>3) is true.
* (5>3) && (8>5) returns false because both (5>3) and (8<5) are false
* !(5==3) returns true because 5==3 is false
* !(5>3) returns false because 5>3 is true.

* Java If-else statement

The java if statement is used to test the condition. It checks boolean condition: true or false.
There are various types of if statement in java.

* if statement
* if-else-if ladder
* if-else statement
* nested if statement

(1) Java if statement

The java if statement tests the condition. It executes the if block if condition is true

Syntax:-
```
if (condition){
  //code to be executed
}
```

Example:-
```
public class IfExample {
  public static void main (String[] args){
    //defining an 'age' variable
    int age = 20;
    //checking the age
    if (age>18){
      System.out.println("Age is greater than 18");
    }
  }
}
```

O/P:- Age is greater than 18

② Java if-else Statement

The java if else statement also tests the condition it executes the if block if condition is true otherwise else block is executed.

Syntax:-
```
if (condition) {
  // code if condition is true
} else {
  // code if condition is false
}
```

Example:-
```
public class IfElseExample {
    public static void main(String[] args) {
      // defining a variable
    int number = 13;
    // check if the number is divisible by 2 or not
    if (number % 2 == 0) {
      System.out.println("even number");
    } else {
      System.out.println("odd number");
    }
  }
}
```

o/p

odd number

③ Java if-else-if ladder St

The statement executes one condition from multiple statements.

Syntax:-
```
if (condition 1) {
// Code to be executed if Condition 1 is true
} else if (condition 2) {
// Code to be executed if Condition 2 is true
}
else if (condition 3) {
// Code to be executed if Condition 3 is true
}
...
else {
// Code to be executed if all the conditions are false
}
```

Example:- // Java Program to demonstrate the use of If else-if ladder
// It is a program of grading system for fail, D grade, C grade, B, grade, A grade and A+

```
public class IfElseIfExample {
public static void main (String[] args) {
int marks = 65;

if (marks < 50) {
   System.out.println("fail");
}
else if (marks >=50 && marks <60) {
   system.out.println("D grade");
}
else if (marks >=60 && marks <70) {
   System.out.println("C grade");
}
else if (marks >=70 && marks<80) {
   System.out.println("B grade");
}
else if (marks >=80 && marks<90) {
   System.out.println("A grade");
} else if (marks >90 && marks<100) {
   System.out.println("A+ grade");
} else {
   System.out.println("Invalid!");
}
}
}
```

O/P - C grade

(4) Java Nested if Statement

The nested if statement represents the if block within another if block. Here, the inner if block condition executes only when outer if block condition is true.

Syntax:-
```
if (condition) {
  // Code to be executed
   if (condition) {
  // Code to be executed
  }
}
```

Example:- // Java Program to demonstrate the use of Nested If Statement
```
public class JavaNestedIfExample {
public static void main (String[] args) {
  // Creating two variables for age and weight
  int age = 20;
  int weight = 80;
  // applying condition on age and weight
  if (age >= 18) {
       if (weight > 50) {
            System.out.println("you are eligible to
                                donate blood");
  }
  }
}
```

Ex:-> public class Nested If statement {
public static void main (String [ ] args) {
(import karne ka koi utility package se)
Scanner sc = new Scanner (System.in);
System.out.println ("Enter the Age:");
int age = sc.nextInt();
User i/p ko scan karwaye
System.out.println ("Enter the Weight:");
int weight = sc.nextInt();

//int age=20;
//int weight=56;
//apply condition on age and weight
if (age>=18) {
if (weight>50) {
syso ("you are eligible for donate blood");
}
else {
Syso. ("you are not eligible to donate the blood due to less weight");
}
}
else {
syso {" you are not eligible to donate the blood due Age below 18");
}
}
}

i/p user se lena hai
Scanner class use karna padega

Ex:- variable = store karenge

* Java Switch - Case Statement

We all use switches regularly, the electrical switches we use for our lights & fans

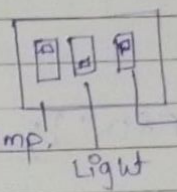


fig. Shows the switch for particular appliance.

Similarly, switch in java is a type of conditional statement that activates only the matching condition out of the given input.

Let us consider the example of a program where the user gives input as a numeric value (only 1 digit in this example), and the output should be the number of words.

The integer variable iswitch, is the input for the switch to work.
The various available options (read cases) are then written as case <value> along with a colon ":"

This will then have the statement to be executed if the case and the input to the switch match.

Example:- Java Switch Example.

```
class switchBoard {
public static void main (String args[]) {
int iswitch = 4;
switch (iswitch) {
case 0:
System.out.println ("ZERO");
beeak;

case 1:
system. out.println ("ONE");
bfeak;

case 2;
System.out.println ("TWO");
becak;

case 3;
system. out. println ("THREE");
break;

case 4;
System. out. println ("FOUR");
break;

default:
system. out. println ("Not in the list");
break;
} }
```

o/p "FOUR"

* Loops in Java *

1. for loop
2. while loop
3. do while loop
4. Enhanced for loop

1. for loop :-

Syntax :-

```
for (data type variable = value (initialization);
        condition ; increment / decrement)
                                        {
        Statement ;
}
```

Example:-

```
package javaPrograms;

public class For Loop Demo {
  public static void main (String [] args) {
    for (int i=0 ; i<=10 ; i++) {
        System.out.println (i);
    }
  }
}
```

(2) While loop

Syntax:-

```
data type variable = value;
while (condition) {
    statement;
    increment / decrement
}
```

Example:-

```
public class While Loop Demo {

public static void main (String[] args) {

    // print 1 - 10 numbers

    int i=10;
    while (i>=0) {
        System.out.println (i);
        i--;
    }

}
}
```

(3) do while loop

Syntax:-

```
initialization
do {
    statement
    increment / decrement;
} while (condition);
```

Example:- package java Program.

```
public class Do While Demo {

public static void main (String[] args) {

    int i=0;                          // initialization
    do {
        System.out.println (i);       // Body me print statement
        i++; // increment / decrement.
    } while (i<=10);                  // condition
}
}
```

④ Enhanced for loop

Syntax:- for (data-type var : Var) {

System.out;

}

Example:- package java Programs;

public class EnhancedForLoopDemo {

public static void main (String [] args) {

String variable + Array variable

String language [] = {"C", "JAVA", "C++"};

for (String lang : language) {

data type variable

System.out.println(lang);

}

}

}

* Constructor *

- class name same as method name
- No return type
- Access modifier :- public, protected, default, private
- It activate automatically when we create object.

-> package java Programs;

public class ConstructorDemo {

public ConstructorDemo () {

System.out.println(" This is constructor demo ");

}

public static void main (String [] args) {

ConstructorDemo cd = new ConstructorDemo();

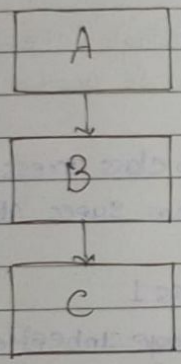
}

}

class 2

```
package Inheritance;
public class Test2 extends Test1 {

    public static void main (string[] args) {

        Test2 t2 = new Test2 ();
        t2.test1Demo ();
    }
}
```

② Multilevel inheritance :-



A ke property B me access kar sakti hai aur B ki property C me access kar sakti hai.
property access karne ke liye "extends" key word use karte hai.

* class C is sub class of B
& B is subclass of A
C is access the properties from B and B is access the properties from A.

Ex:-

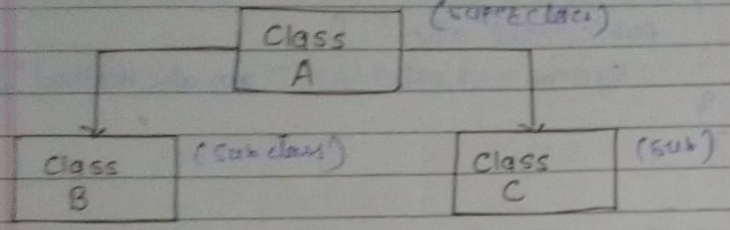
(A)
```
package Inheritance;
public class A {
    public void abc () {
        System.out.println ("I am abc method");
    }
}
```

(B)
```
package Inheritance;
public class B extends A {
    public static void main (string[] args) {
        B obj1
    public void pqr () {
        System.out.println ("I am pqr method");
    }
    public static void main (string[] args) {

        B.obj1 = new B ();
        Obj1.abc ();
    }
}
```

(C)
```
package Inheritance;
public class C extends B {

    Public static void main (string[] args) {
        C obj2 = new C ();
        Obj2.pqr ();
    } Obj2.abc (); - // By this we acces C from A.
}
```

o/p :- I am pqr method.

③ Hierarchical Inheritance :-



Ex:-

The multiple subclass access the properties from one superclass.

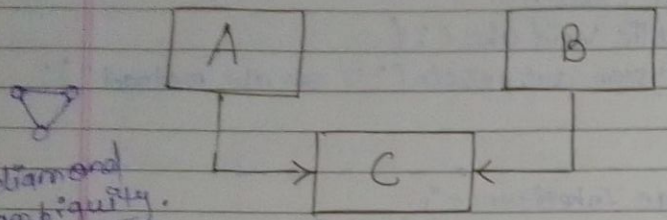
Example- package Inheritance;

(A)
```
public class A {
    public void abc() {
        System.out.println("I am abc method");
    }
}
```

(B)
```
package Inheritance;
public class B extends A {
    public void pqr() {
        System.out.println("I am pqr method");
    }
    public static void main(String[] args) {
        B obj1 = new B();
        obj1.abc();
        obj1.pqr();
    }
}
```

(C)
```
package Inheritance;
public class C extends A {
    public static void main(String[] args) {
        C obj2 = new C();
        obj2.abc();
    }
}
```

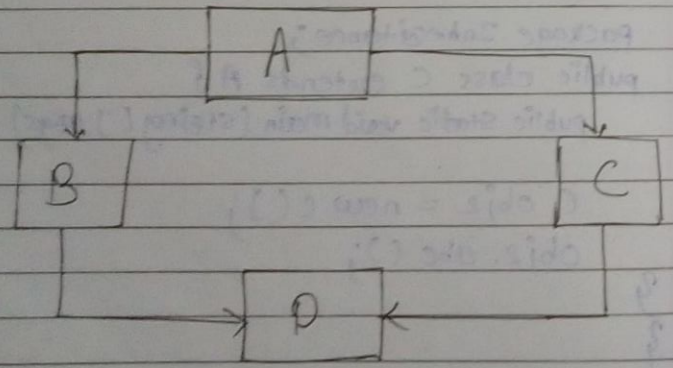
④ Multiple Inheritance :-



diamond ambiguity.

In multiple inheritance when one subclass try to access the properties from multiple super classes so the compiler is confused from which superclass he need to access the properties so here diamond ambiguity problem is occured so the multiple inheritance is not possible in java.

⑤ Hybrid Inheritance



Hybrid inheritance is the combination of hierarchical and multiple inheritance this is not supported in java.

syntax: Interface interface name {
//public abstract void show();
}

* Interface *

* Interface are similar to abstract class but having all the method of abstract name.
* Interface are the blueprint of the class.
* It is use to achieve abstraction.
* It support multiple inheritance

Variable : public static final int a=0;
(if we are declaring variable in interface that variable should be public static final).

Methods : public abstract void show();
(The method declared in interfaces that should be public abstract void methodname, we not providing the body of that method).

Ex :- Interface 1 :
Program:
package Interface demo;
public interface I1 {
    public static final int a=10;
    public abstract void show();
}

Interface 2 :
Program

→ multiple interface create kar sakte hai aur is ya implement kar sakte ho, interface me jo koi bhi method public, static final hona chaiye aur usi value ko change nahi karni hai

multiple interfaces ko ek class me access kar sakte ho hoti hai vo abstract hani chahiye, aur variable change nahi karni hai

```
package Interface Demo;

public interface I2 {

        public abstract void display ();
}
```

Class (in class we implements the interfaces it means need to provide the body of unimplemented methods which methods are declared in interfaces)

Test:-

```
package Interface;
public class Test implements I1, I2 {
  @override
  public void display () {
     System.out.println ("I am display method");
  }
  @override
  public void show () {
     System.out.println ("I am show method");
  }
  public static void main (String [] args) {
     Test t = new Test ();
     t. show ();
     t. display ();
     System.out.println(a);
  }
}
```

o/p
I am show method
I am display method

↳ Multiple interfaces ka implement karte hai ek concrete class me.

→ Jab ham class create karenge uske age ham implements keyword lagange (I1, I2) isme jo bhi method jinka body nahi provide kiya hai uska concrete class me body provide karenge.

normal method hota hai jisbhi method ko call karene hai →
class name obj name = new class name ();
        obj name. method name ();

* Abstract :-
→ A abstract class is a class in which complete as well as incomplete methods are present.

* Abstract class contains at least one complete & one incomplete method.

* We can not create a object of abstract class because to create a object all the incomplete methods should be completed first.

* We cannot create the object of abstract class because of abstract method so all the abstract method inside a abstract class are completed in a sub-class this class is called as concrete class

* Abstraction *

* An abstraction means showing only essential features of application and hiding the details

ie. focusing on main things.

If we are having abstract method then we should make this class as abstract class

In next class we extends the abstract class then we can access the abstract method in this class and can provide body to this method

*

First class : (abstract class / method should be abstract

Ex:- package Abstraction;

```
public abstract class Vehicle {
    int no_of_types;
    abstract void start();
}
```

*

In second class we first extends the abstract class and adding unimplemented method of abstract class or providing body to them.

Ex:- package Abstraction;

```
public class Car extends Vehicle {
    @Override
    void start() {
        System.out.println("start with key");
    }
}
```

*

next class we just extends the abstract class and adding unimplemented method of abstract class or providing body to them and also writing main method and in that call the method of respective classes by creating object of that particular class.

Ex:- package Abstraction;

```
public class Scooter extends Vehicle {
    @Override
    void start() {
        System.out.println("start with kick");
    }
    public static void main(String[] args) {
        Scooter sc = new Scooter();
        sc.start();
    }
}
```

Encapsulation :- It is a mechanism to wrap up variables & methods together as a single unit.
It the process of hiding information details & protecting behaviour of the object

* Polymorphism *

* Poly means "many" & morphism means "form".
* It is the ability to take more than one form.
  ie. one name many forms
  eg:- water - solid, liquid, gas.

Types of Polymorphism

① Compile time polymorphism / static polymorphism
method overloading.

② Run time polymorphism / Dynamic polymorphism
method ~~overloading~~. overridding.

* (imp)

① Method overloading :-

* Multiple methods with same name but different parameter
* Same class
* parameter should be different (no. of parameter / type of parameter)

→ package Polymorphism;

```
public class Test {

    public void show () {
        system.out.print.ln ("1");
    }
```

* ek class me multiple method same name ki hoti hai, likin parameter different hona chahiye.

```
    public void show (int a) {
        System.out.print.ln ("2");
    }
    public void show (char a) {
        system.out.print.ln ("3");
    }
    public void show (int a, char b) {
        system.out.print.ln ("4");
    }
    public static void main (String [] args) {

        Test t = new Test ();
        t.show ();  //1
        t.show (5); //2
        t.show ('p'); //3
        t.show (6, 'b'); //4
    }
```

P.M
C.P
M.O

→ when a object or element perform different behaviour at different stage of life cycle is called as polymorphism.

→ In compile time P.M method decn get binded to its decn (body) at the time of compilator based on argument / parameter is called compile time P.M.

M.O
R.P

→ when method selection get binded to its decn at the time of execution based on object / instance is called as Run time Poly.

② Method overEidding :-

* One class having one method and next class having method which is same as first class, method name are same as well as parameter are also same it is called as method overEidding
* Different class
* parameters should be same (no. of parameter, type of parameter)

Prog1 :- package Polymorphism;

public class overEidding Demo1 {
    public void show() {
        System.out.println("1");
    }
}

Prog2 :- package Polymorphism;

public class overEidding Demo2 extends overEidding Demo1 {
    public void show() {
        System.out.println("2");

multiple class me same name ki method create karte hai / parameter bhi same rakhe

    public static void main(String[] args) {
        overEidding Demo1 demo1 = new overEidding Demo1();
        demo1.show(); //1
        overEidding Demo2 demo2 = new overEidding Demo2();
        demo2.show(); //2
    }
}

→ Abstract class | Interface

Abstract class	Interface
* abstract class doesn't support multiple inheritance	* interface supports multiple inheritance.
* A can have final non static & final static & non static final variable variable	* interface has only static & final variable.
* The abstract keyword is used to declare abstract class	* the interface keyword is used to declare interface
* An abstract class can be extended using keyword extends	* An interface can be implemented using keyword implements.

② Method overEidding :-

* One class having one method and next class having method which is same as first class, method name are same as well as parameters are also same it is called as method overEidding
* Different class
* Parameters should be same (no. of parameter, type of Parameter)

Prog1:- package Polymorphism;

public class overEidding Demo1 {
    public void Show() {
        System.out.print_ln("1");
    }
}

Prog2:- package Polymorphism;

public class overEidding Demo2 extends overEidding Demo1 {
    public void Show() {
        System.out.print_ln("2");

multiple class me same name ki method create karte hai / parameter bhi same rakhe

public static void main(String[] args) {

overEidding Demo1 demo1 = new overEidding Demo1();
demo1.Show(); //1

overEidding Demo2 demo2 = new overEidding Demo2();
demo2.Show(); //2

    }
}

Abstract class	Interface
* abstract class doesn't support multiple inheritance	* interface supports multiple inheritance.
* A can have final non static & final static & non static final variable.	* interface has only static & final variable
* The abstract keyword is used to declare abstract class	* the interface keyword is used to declare interface
* An abstract class can be extended using extends keyword	* An interface can be implemented using keyword implements.

Imp

	Method overloading	Method overriding
(1)	Multiple methods with same name but diff. signature.	(1) When subclass contain method with same name & same signature as super class.
(2)	Same class	(2) different class
(3)	Different argu-ments No. of argument, segment of argument Type of argument	(3) Same argument - No. of argument - Segment of argument - Type of argument
(4)	method overloading is a compile-time polymorphism	(4) method overriding is a run time polymorphism
(5)	method overloading may or may not require inheritance	(5) While method overriding always needs inheritance.
(6)	Return type can or Cannot be same but we must have to change the parameter	(6) Return type must be same or covariant.

Imp

* This keyword :-

When the name of local variable & global variable are same within a class then to use the global variable in a same class we use this keyword.

Note:- This keyword cannot be called under a static method.

* Super keyword :-

It is used to call the global variable of super class in sub class when we have same variable name of the global variable

Note:- super keyword cannot be called within a static method.

* Final :- It is a keyword used in Java used to restrict the value of variables or overriding of method or inheritance

* Final variable :- We cannot change its value final method cannot be overrided A final class cannot be inherited.

* Finally :-
It is a block used with try & catch blocks.
The code under finally block will be executed either exception is arrived or exception is not arrived.

* Finaliger() :-
It is a method used to erase the garbage values or garbage data just before object get garbage.

* Hashmap	Hash set
① Hashmap is the implemetn of map interface.	① Hash set an other hand is the implemetn of set interface.
② Put method of hashmap is used to add element in hashmap.	② on other hand add method of hashset is used to add element in hash set
③ hashmap internally do not implements hashset or any set for its implementation	③ Hash set internally uses hashmap for its Implement.

* Different hash map	hash table.
- It is not synchronized	It is synchronized.
- It is not legacy class	is a legacy class
- It is not thread safe	is a thread safe.
- It can store one null key + any no of null values.	

* Array list	Hash set
① Array list implementation of the list interface	① it is implementation of a set interface.
② Array list internally implits for its implimentation	② it uses hash map array for its implemetn
③ allows duplicate value	③ Not allows duplicate value.
④ Any no. of null value inserted.	④ only null value it its collection.

* String Handling in Java *

→ String is a sequence of characters written in double quote.

→ String may have alphabets, number and special characters.

int a = 23; / char a = 'e';

String abc = "samarth";

String pqr = "123 san@";

Prog:-
```
package stringHandling;
public class DemoString {
    public static void main (String [] args) {
        String mytool = "Selenium";
        String mytools [] = {"UFT","Selenium","RFT"};
        System.out.println (mytool); // Selenium
```
ex:-
```
//  for (String tools : mytools) {
//      System.out.println (tools);
//  }
}
```
o/p - ~~selenium~~
UFT
Selenium
RFT

* Concatination of Strings

① String + String = Concatination

② String + Number = Concatination

③ Number + String = Concatination

① String + String = Concatination.

Ex:-
```
public class DemoString {
    public static void main (String [] args) {
        String str1 = "Selenium";
        String str2 = "Testing";
        System.out.println (str1+str2); // SeleniumTesting
    }
}
```

② String + Number = Concatination

Ex:-
```
System.out.println ("selenium"+1+1);
// Selenium11
```

③ Number + String = Concatination

Ex:-
```
System.out.println (1+1+"Selenium"); // 2 selenium
```

* String Comparison

① Using Comparison Operator

② Using equals() method

③ Using CompareTo() method

① Using Comparison Operator :

Ex:-
```
String ste1 = "SELENIUM";
String ste2 = "Selenium";
String ste3 = "SELENIUM";
String ste4 = "2Selenium";

System.out.println(ste1==ste3); //true
System.out.println(ste1==ste2); //false
System.out.println(ste1==ste4); //false
System.out.println(ste1!=ste4); //true
```

② Using equals() method :

```
String ste1 = "SELENIUM";
String ste2 = "selenium";
String ste3 = "SELENIUM";
String ste4 = "2Selenium";

System.out.println(ste1.equals(ste2)); //false
System.out.println(ste1.equals(ste3)); //true
```

③ Using compareTo() method :

Ex:-
```
String ste1 = "SELENIUM";
String ste2 = "selenium";
String ste3 = "SELENIUM";
String ste4 = "2selenium";

System.out.println(ste1.compareTo(ste2)); //negative
System.out.println(ste2.compareTo(ste1)); //positive
System.out.println(ste1.compareTo(ste3)); //0
```

* Concats() : Join two Strings

Ex:-
```
package StringHandling;

public class StringConcatination {

    public static void main(String[] args) {

        String ste1 = "Selenium";
        String ste2 = "Testing";

System.out.println(ste1.Concat(ste2));
                    // seleniumTesting
    }
}
```