String Class Constructor

String s = new String() → Create an empty string object.

String s = new String (String literal);

↳ Creates an object with string literals on heap

String s = new String (StringBuffer Sb) →

Creates an equivalent String object for StringBuffer.

String s = new String (char[] ch) ⇒ Creates an equivalent string object for char array

String s = new String (byte[] b) ⇒ creates an equivalent object for byte array.

tx:- Ch[] ch = {'s', 'a', 'v', 'a'};

String s = String (ch);

S.O.P (s); java.

Methods of String :-

1. public char charAt (int index)

tq:- String s = "Sachin";

S.O.P (s.charAt(0)); // S

S.O.P (s.charAt(-1)); // String Array out of bonds exception.

S.O.P (s.charAt(10)); // string Array out of bounds exception.

(2) public String concat (String str)

eg:- String s = "Sachin";

S.O.p (s.concat ("tendulkar"));

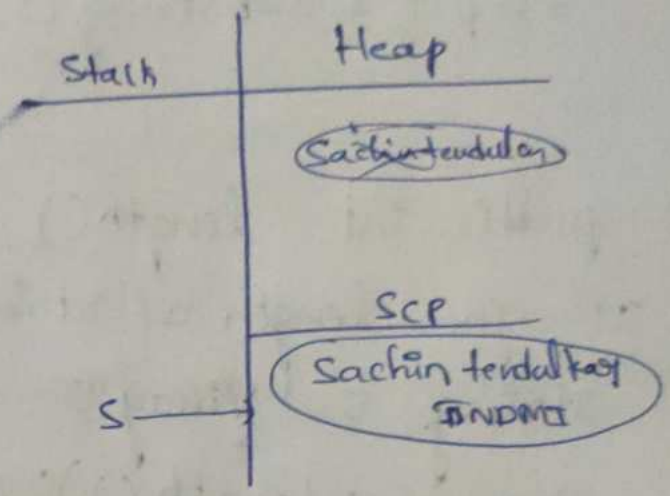
st = "IND";

s = st + "MI";

SO. p(s);

// sachin tendulkar.

// Sachin tendulkarINDMI



(3) public boolean equals (object o)

It is used for Content Comparision, In String equals() method is overriden to check the content of object.

(4) public boolean equalsIgnoreCase (String s)

It is used Content Comparasion without comparision the case.

eg:- String s = "java";

S.O.p (s. equals ("JAVA"));

S.O.p (s. equalsIgnoreCase ("JAVA"));

false

True

(5) public String substring (int begin)

It gives the string from begin index to the end of string

String s = "ineuron";

s.o.p (s.substring(2)); // euron. - search from 2 to end.

⑥ public String substring(int begin, int end)

It gives the string from begin index to end-1 to String

```
String s="ineuron";
S.O.p(s.substring(2,6); // euro
```

Searching from 2 to 5 will happen

⑦ public int length()

It give length of str

```
String s="vinay";
S.O.p(s.length());
```


⑧ public String replace(old char, new char);

```
String s="Vinay";
S.O.p(s.replace('i','a'); // vinay
String s="ababab";
S.O.p(s.replace('a','b')); // bbbbbb
```

⑨ public string to lowerCase();

⑩ public string toUpperCase();

```
String s="VINAY";
S.O.p(s.toUpperCase()); // VINAY
S.O.p(s.toLowerCase()); // vinay
```

till the printing the jvm have control then no gc will collect that object.

(11) public String trim();

To remove the blank spaces present at the beginning and end of string but not the blank spaces present at middle.

eg: String name = "_vinay kumar_";
s.o.p (name.trim()); // vinay kumar.

String name = "_vijay kumar_"
s.o.p (length()) // 13
s.o.p (name.trim()); // vinay kumar - runtime operation.
s.o.p (name.length()); 13.

(12) public int indexOf(char ch)

it returns the index of 1st occurance of specified character if the specified character is not available then it returns -1

string s = "Sachinramesh";
s.o.p (s.indexOf('a')); 1
s.o.p (s.index of ('z')); -1

(13) public int lastIndex of (char ch)

it returns the index of last occurance of specified. if the specified character is not there then it returns -1

string s = "Sachin ramesh";
s.o.p (s.lastindexOf('a')); 7
s.o.p (s.last indexOf('z')); -1;

(14) public String toString();

When ever we print any reference, by default JVM will call toString() on the reference to print variable

```
public Student {
    String name = "sachin";
    int id = 10;
}
Student std = new student();
S.O.p (std); // student@Hexadecimal.
S.O.p (std.toString()); // student@ Hexdecimal.
```

xxx imp

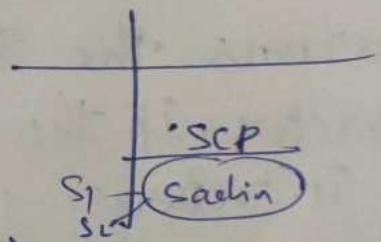
call back method.

toString

(5) String S_1 = "sachin";

String S_2 = S_1.string();

S.O.p (S_1 == S_2); true



(6) String S_1 = new String ("sachin");

String S_2 = S_1.toString();

// S_3 = S_1.toUppercase();

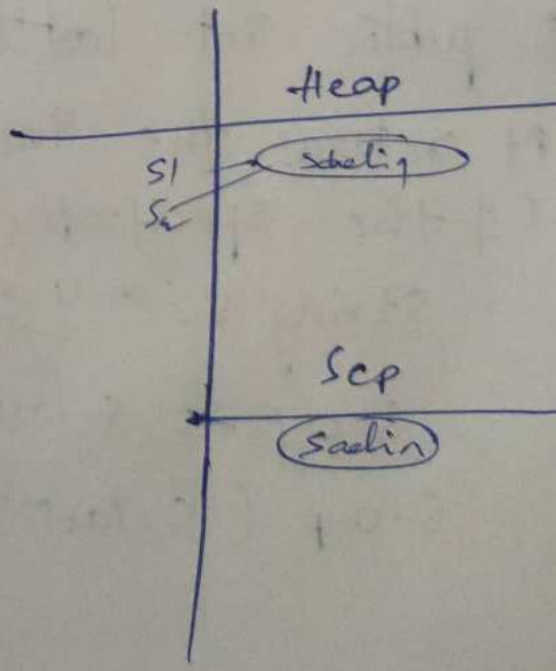
S_4 = S_1.toLowercase();

// S_5 = S_1.toUppercase();

// S_6 = S_1.toLowercase();

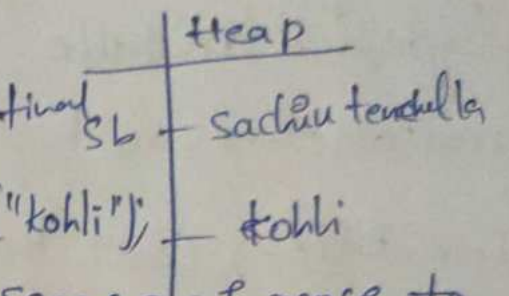
S.O.p(S_1 == S_6) false

S.O.p(S_3 == S_5); True



Mutable → we can change and changes will be reflected in same memory.

```
final StringBuffer sb = new StringBuffer ("sachin");
    sb.append ("tendulkar");
    S.O.p(sb);
final StringBuffer sb = new StringBuffer ("kohli");
```



By using keyword we cannot use same reference to another object. But we can change the content in final reference becoz StringBuffer is Mutable.

final Vs Immutability

→ final is a modifer application for Variable, where as immutability is applicable only for objects.

→ If reference Variable is declared as final it means we cannot perform reassignment for reference Variable. It does not mean we cannot perform any changes in that object.

→ By declaring a reference Variable as final we want get immutability nature

→ final and immutability is different concept.

eg:
```
final StringBuilder sb = new StringBuilder("Sa
                                    ("sachin");
    sb.append ("tendulkar");
    S.O.p (sb)
sb = new StringBuilder ("dhoni"); CE;
```
Cannot assign a value to finchal Variable

Note :- final variable (valid), final objcat (invalid).
immuatable variable / objcat (invalid).

StringBuilder, StringBuffer and all Wrapper classes.
Bydefault mutable.

mutable → can be changed

Immutable → can't be changed.

String Buffer :- 1. if the content will change frequently then it is not recommended to go for string objcat becaz for every new change new objcat will be created

2. To handle this type of requirment we have string Buffer / StringBuilder concept

Constructors :-

```
StringBuffer sb1 = new StringBuffer();
    S.o.p (sb1.length()); //0
    S.o.p (sb1.capacity()); //16
  sb1.append ("abcdefghijklmnop");
    S.o.p(sb1.length()); //16
    S.o.p (sb1.capacity()); //16
  sb1.append ("q");
  S.o.p (sb1.length()); //17
  S.o.p (sb1.capacity()); // (old capacity +1)* 2 = new capacity.
```

StringBuffer Sb = new StringBuffer();
Creates an empty StringBuffer object that than default.
Initial capacity is 16
Once StringBuffer reaches the maximum capacity a new
String will be created.
new capacity = (Current capacity + 1) * 2;

2. StringBuffer Sb = new StringBuffer (initial capacity);
~~it creates~~ creates an empty string with specified initial capacity.

Eg:- 1 - StringBuffer Sb = new StringBuffer (19);
S.O.p (Sb.capacity); = 19.

3. StringBuffer Sb = new StringBuffer (String s)
it creates a StringBuffer object for given String with
the capacity = s.length() + 16;

eg:- StringBuffer Sb = new StringBuffer ("Sachin");
S.O.p (Sb.length()); //6
S.O.p (Sb.capacity())) //22

(string length
6 + 16 = 22
(by default value of capacity in string buffer.

Methods

public StringBuffer charAt();
String Sb = StringBuffer ("Vinay");
Sb.charAt (3); // a
Sb.charAt (100); // out of

2) public StringBuffer setCharAt();

StringBuffer sb = new StringBuffer("Vinay");

sb1. setCharAt(5, 'A');

s.o.p(sb1); // VinayA

3) public StringBuffer append(String s)

a) public StringBuffer append(int i)

5) " " (boolean b)

6) " " (double d)

7) " (float f)

8) " " (int index, Object o)

9) " "

Ex:- StringBuffer sb = new new StringBuffer();

sb.append("PI Value is:");

sb.append. (3.1414);

s.o.p(sb); PI Value is::3.1414.

append method is overloaded method

method name is same but change in argument type.

int set (overloaded method)

public StringBuffer insert(int index, String s)

" " (int index, i)

(int index, long l)

(int index, double d)

(" " float f)

Object o).

To insert the string at specified position we use insert method.

Ex:- StringBuffer sb = new StringBuffer ("sachin");
sb.insert (3,'h');
S.o.p (sb);

method delete

public StringBuffer delete (int begin, int end)
it deletes the character from specified index to end-1.
public StringBuffer .delete (CharAt (int index).
it deletes the character at specific index
Ex:- StringBuffer sb = new StringBuffer ("Vinay");
sb . delete (2,5);
sb . delete CharAt(1);
S.o.p (sb // V.

Reverse (It is used to reverse given String)
StringBuffer sb = new StringBuffer ("Vinay");
sb . reverse ();
S.o.p (sb); // yaniv

length (it is used to consider only specified no of character and remove all remain char)

StringBuffer sb = new StringBuffer ("Vinaykumar");
sb . set length (6);
S.o.p (sb).

Trim to Size : used to deallocate the extra allocated free memory such that capacity and size are equal.

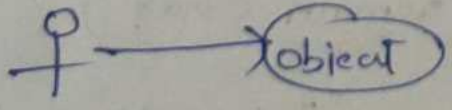
```
StringBuffer sb = new StringBuffer (1000);
  S.O.p (sb . capacity ()); // 1000
  Sb . append ("sachin");
  S.O.p (sb . capacity ()); // 1000
  Sb . trimtoSize ();
  S.O.p (Sb . capacity ()); 6
```

public Void ensure capacity (int Capacity)
It is used to increase capacity dynamically based on our requirements

```
eg: StringBuffer sb = new StringBuffer ();
  S.O.p (sb . capacity ()); //16
  Sb . ensureCapacity (1000);
  S.O.p (sb . capacity ()); //1000
```

StringBuffer

Thread → person who is using object

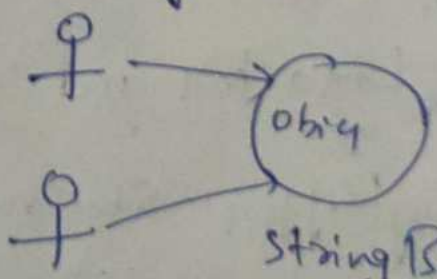


StringBuffer (1.0V)
(Synchronized)

Waiting State

Thread Safe

StringBuilder



StringBuilder (1.5V)
(non synchronized)

Not Thread Safe

StringBuffer :-

Every method present in StringBuffer is Synchronized. So at a time only one thread can allowed to operate on StringBuffer object. it would create performance problem, to overcome this problem we should go for StringBuilder.

StringBuilder (1.5v)

StringBuilder is same as StringBuffer (1.0v) with few difference.

StringBuilder

No methods are Synchronized.

At a time more than one can thread can operate is not thread safe.

Threads are not required to wait so performance is high. introduced in jdk 1.5 version.

String → we opt if the content is fixed and it won't change frequently.

StringBuffer :- we opt if content changes frequently but thread safe is required.

StringBuilder :- we opt if content changes frequently but thread safety is not required.

Method Chaining

Most of the methods in String, StringBuilder, StringBuffer return the same type only hence. after applying method on the result we can call another method which forms method Chaining.

Eg:-
```
StringBuffer sb = new StringBuffer();
sb.append("Sachin").insert(6,"Tendular").reverse()
.append("IND").delete(0,4).reverse();
S.o.p(sb);
```

---

toString() - method.

```
class Student {
    String name = "Sachin";   } - User defined class
    int id = 10;
}
main(){
    Student std = new student();  User defined object
    s.o.p(std); // (Sachin10) (Sachin@Hexadecimal)
    S:o.p ln();
    String name = new string("dhoni");
    System.out.println(name); // dhoni
```

When we trying to print variable JVM will call tostring(); call back method.

→ to String return type

```
public String toString(){
    return "";
}
```

when ever we print any refernce, by default JVM will call to string() on the object only I will say.

tostring(); ——— o ———

why to String method.

→ student to String method will called.

→ if we don't override to string method()

→ it will call object class to string method.

→ Object class to string() will never print value of associated with instances Variables

to print the instances Variables we have to call tostring method and override that method.

return type is String

(or)

else it will not print & Value.

ex: Student object name with Hexadecimical Value.

OOps

③ weeks.

Object Oriented programming

i) class

ii) objects

iii) Methods

① → Encapsulation
② → Inheritance
③ → polymorphism
④ → Abstraction.

① privacy (or) Security ⇒ Data hiding (or) Binding (Encapsulation)
② code Reusability ⇒ (Inheritance)
③ code flexibility (polymorphism)
④ implementation hiding → feature Visibility (Abstraction)

Encapsulation Beautiful..

① Encapsulation

→ Data hiding
→ Data binding
→ providing Security
→ providing Controlled access to datamembers.

(privite, setter, getter, this keyword, shadowing)

```
class Student {
    int age;
    string name;
    String city;
}
```

} Instance Variable / Data member / / feilds / / properties

Private → To provide Security for instance Variables, we can access in same class. But outside class we can't access.

Instaziation → Creation of object.

Setter → if a method is doing activity of recieving data from outside and setting it to its data member

getter → if a method is doing activity of return value. Anyone wants to use a value from outside of privite class we can use Get method name.

data binding : For every Variable, we have method. And Variable, age. are working together.

In a class which has all data members are private is called Bean.

Shodowing problem → whenever there is a name conflict b/w instance Variable. and local Variable. is called shodowing problem.

// To solve this problem we can use(this) key word.

eg:- Class Student {
    private int age;
    private String Name;
    private string City;

```
Void setName (String Name) {
    this.Name = Name;
}
String getName() {
    return name;
}
Void setAge(int a) {
    this.age = age;
}
int getAge() {
    return age;
Void setCity (String city) {
    this.city = city;
}
String getCity() {
    return city;
}
public static void main (String[] args) {
        St. SetName ("Vijay");
        St. getoName ();
        System.out.println (St.getName());
}
```

- right click
- go to Source.
- generate setters and getters by selecting all.

it will generate setters and getters. by Automatically

output: Vijay

7/11/22

Getter : Syntax

get propertyname () it should not have any parameters. Ex: int getAge () {
return age;
}
followed by variable name

Setter : set propertyname () it can have parameters

```
Void SetAge (int age) {
    this.age = age;
```

public: within entire project we can access anywhere by using public key word it increase accessibility.

```
public boolean isMarried () {
    return married;
}
```
- boolean typegetter

**

this → which will hold reference of currently executing object.

Class → create many objects.

this : refers to current object in a method (or) constructor. The most common use of this keyword is to eliminate the confusion b/w class attributes and parameters with same name.

tx:-

```
Class Student {
    int age;                          parameter (or) local variable
    public void setAge (int age)
        this.age = age;
    }
```

this.age → instance variable. age → parameter (or) local variable

Common setter we can have, not common getter. because return type is different

Common setter :

```
public void setData (String name, int age, String city){
    this.name = name;
    this.age = age;
    this.city = city;
}
```

Constructor explained in detail..

Constructor :- Class name and method name is same

```
Class Student1 {
    int age;
    String name;
public  Student1 (String name, int age) {
        this.age = age;
        this.name = name;
    }
}
public static void main (String[] args) {
    Student1 std = new Student1 ("Rohit", 12);
}
```

Constructor will not have return type. Constructor will be called while creating object.

Student1 std = new Student1( )
↳ Constructor call.

→ Constructor need is Initialazation not for object creation.

when object creation is happening JVM will called Constructor if any present in class.

if Constructor method is not there in class by default JVM have Constructor method add.

JVM will automatically include default Constructor with 0 parameters.

Default Constructor

Student1 std1 = new Student1("Vinay");

Student2 std2 = new Student1( ); error

whenever there is a callet to Constructor if progsammer is not Specified any Constructor that time . JVM will include default Constructor.

And if progsammer Specified any one Constructor in progsam in that case JVM will not include

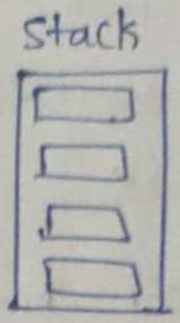
Constructor Overloading : Same Constructor name but different parameter.

Super(); → it will call to parent class

Constructor chaining → Calling one Constructor to another Constructor.

Stack overflow (

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



till stack area got full then it leades to (Stack overflow)

if we call method in same method is called stack overflow

Ex:-

```
void disp(){
    disp();
}
```

```
Class Dog{
private age;
   String name;

public Dog {
  this ("shera",9999);
 }
Dog (string name, int age){
  this.name =name;
  this.cost = cost;
 }
 public int getName(){
    return name;
}
public string getString(){
    return String;
}
}
main(){
Dog d1 = new Dog();
 d1.getName();
 d1.getAge();
dog d2 = new Dog("Vijay",10000)
 d2.get Name();
 d2.getAge();
```

Constructor :- Same name as classname

Constructor parameters will work how method parameters will work

Constructor is called during object creation
(or)
instantited (creation of object).

return statement is invalid.

Constructor will not have return type explicity

Inside constructor first statement super() & this()

super() → call parent constructor
this() → calls constructor of same class

Constructor chaining ⇒ one Constructor calling another constructor.

Both this() & super() in same constructor (X)

Can write this()(or) super() ⇒ apart from first statement. (X) (X) Not Valid.

purpose :- if some statement has to be executed the moment we create object we can write inside Constructor.

this() → inside a one Constructor ⇒ requirement :
→ TO another body of Constructor.

this	this()
Key word	method
refer to current object	it will call same class Constructor.

Constructor overloading :- more than one constructor but different parameters

method	constructor.
Call explicity by Calling name	when object is created Constructor is called
return type explicity	no explicit return type
Inside method return Statement is allowed	no return Statement.

Static keyword :- non-access modifier used for methods and attributes.

Static keyword	
class & Static Variables	non Static Variables (or) instance Variables normal java Block.
Static Block	
Static method	

normal method (or) non-static method.
}

if we create static variables, we can use inside static block and static method.

and we can't use in java block, non-static method.
if we create non-static variables (instance variables we can use in non-static java block) and ~~(and we can use in non-static java block)~~ non-static method. But we can't use in static method and static Block.

instance variables memory allocated inside object
~~first we need to create object. Inside object.~~

first we need to create object. Inside object memory for instance variables are allocated. we can say object dependent also. we can't use in static method or static block because static method, block are object independent.

```
class Demo{
Static int a;
Static int b;
 static {
   a=10;
   b=20;
}
```

```
Static void disp(){
 S.o.p(a);
 S.o.p(b);
}
int x;
int y;
{
x = 40;
y = 50;
}
```

implicit → jvm will
explicit → we have to do externally

```
Demo(){
  S.o.p("Constructor");
}
void add(){
 S.o.p(x);
 S.o.p(y);
}
main(){
 Demo.disp(); — we can call Static method without object.. it is object independent
 Demo.d = new Demo();
    d.add();
```

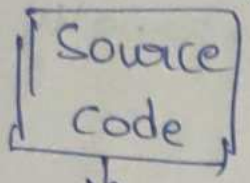
In a class if all elements are three then it will start execute as follows:

1) Static Variable → (1)
2) static Block → (2)
3) Static method → (1)
4) non-static block → (4)
5) non-static Variables → (5)

5) Constructor → (6)

6) non-static method → (7)

⇒ Java program



↓

Javac

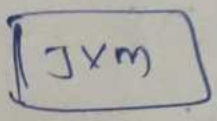
↓

Byte code class file

↓

JVM → It has to execute

JRE



JVM → ① Class loader Subsystem

② Data Areas

③ execution will happen (intrepreter JIT compiler).

Class ⇒ JVM ⇒ Class loader Sub System

↓ ↓ ↓

loading linking Initialazation

From method area byte code file will take to class loader system.

loading → To load the codes class in general

Boot Strap loader → It will load all API Classes. (or) in built classes like scanner class (or) Stringclass

extension & Application loader → It will load all classes which will written by programmer.

linking :-

Verification :- Byte code Verifier It will check No one is trying to hack. All byte codes are Correct it will check.

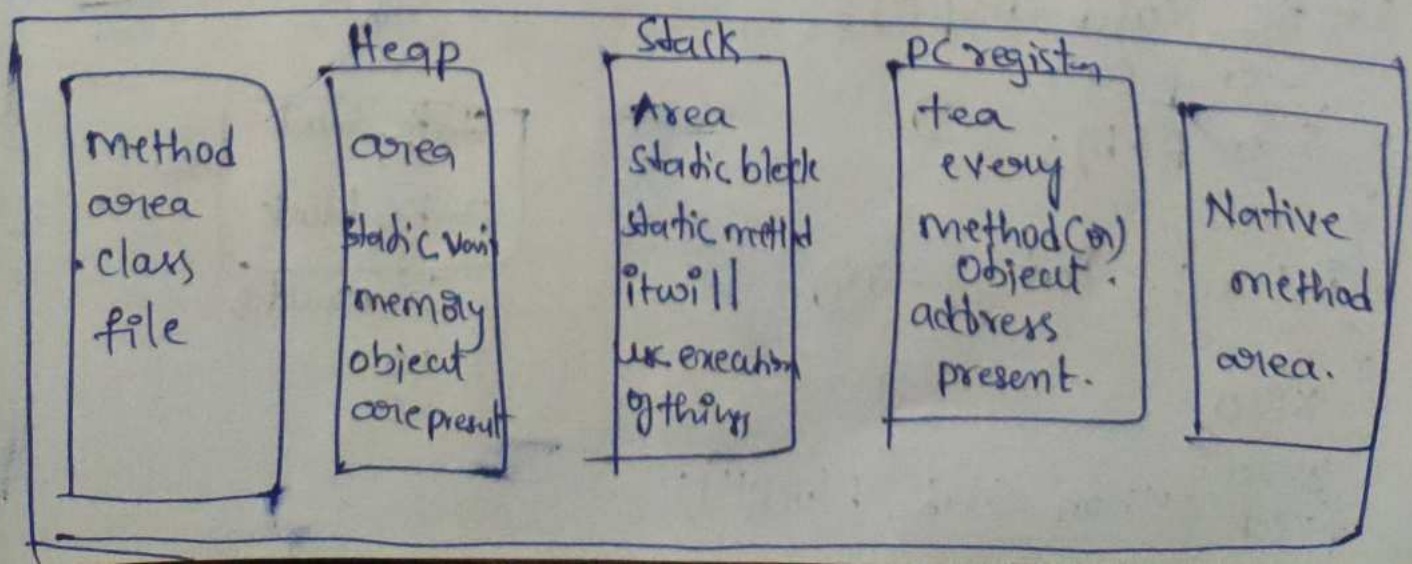
preparation :- It will check are there any VIP (or) static Variables are there. if any static Variables are there in heap memory static Variables allocated. default values also given.

initilization :-

Static Block is executed.

like a=10; b=20;.

JVM Data Areas

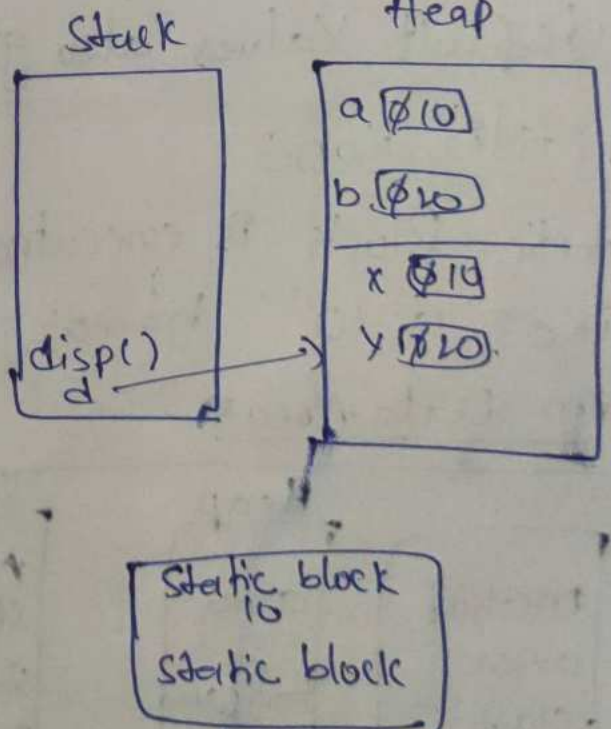


later execution will start. JVM will use interpreter to convert 0's & 1's. JIT will use only at specific time if some method is there which is been called multiple times. if suppose add() is there we are calling so many times. again and again. no need to convert in 0's & 1's JIT will check add() is converted before. or not. if it is converted. it will just use. (or) it will avoid duplicates. Interpreter will execute line by line.

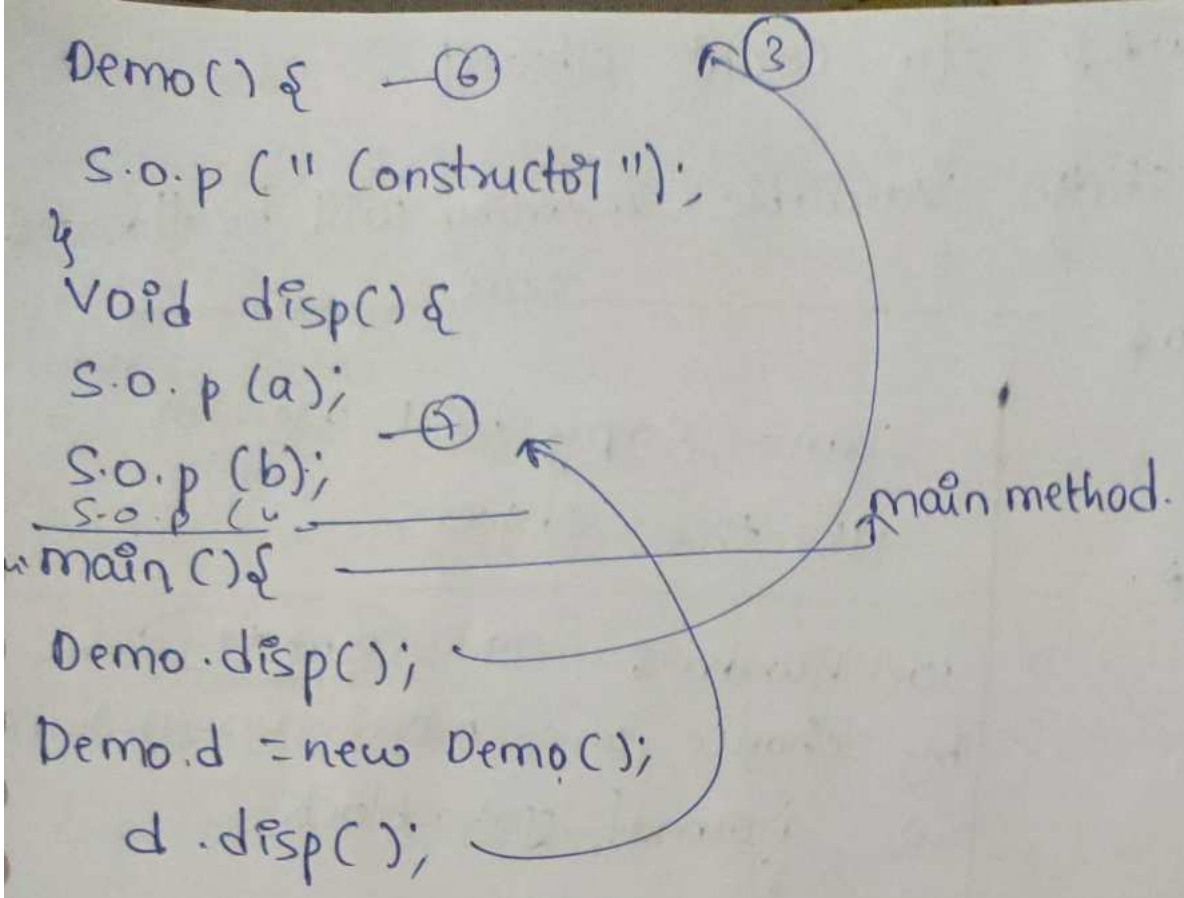
garbage collector → if any object with no reference it will collect.

Ex!

```
class Demo {
 Static int a;
 static int b;          —①

 Static {
 S.o.p ("Static block");
   a=10; b=20;          —②
 }
 Static void disp(){
    S.o.p(a);           ③
    S.o.p(b);
 }
  int x;  int y;        —④
 {
   x=10;
   y=20;
  S.o.p ("non Static block");
```



Static block
10
Static block
Constructor
10
20.



Demo d = new Demo(); while creation object.

3 things happen

1. memory of instance variable inside object
2. Java block executed.
3. Constructor (include constructn

execution :-

1) Static variables ⇒ Heap area ⇒ during Class loading

2) Static blocks ⇒ To initilation Static variable.
⇒ Class loading.

3) Static method ⇒ main method.
//
Static method should call from main method.
without creating object

Why? WAP to count object.

→ for static variables memory will be allocated once.

```
Class Demo{
 int a;
 int b;
 int count;
Demo()
{
 count ++;
}
 main(){
Demo d1 = new Demo();
 "  d2 = new Demo();
    d3 = new Demo();
```

Same copy will use for all the objects.

100 hundred 99 - count ++ only by chance one missing wrong count.

So. normal Java block

```
{
 count++;
}
```

Static Variables ⇒ Static keyword used to Create Variable.

→ memory allocated during class loading.

→ memory allocated Heap area.

→ memory allocated only once in Heap

→ one copy of static variables used by all objects.

→ using class we can call Static Variable.

→ It is also called as class Variable

→ object independent

→ accessed inside static & non static.

→ Initialize static variable (purpose).

Static Block vs main method

↳ first Static Block will executed.

After main method will execute.

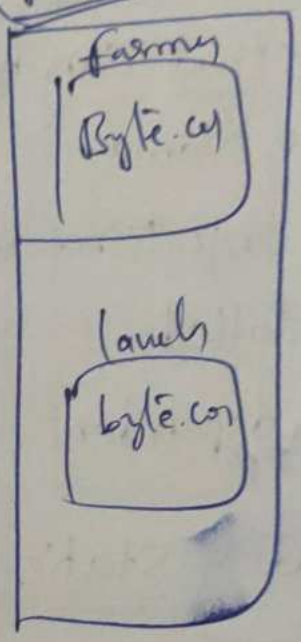
Static method can be using className.	non Static method. object creation is needed.
by using object reference can be called.	by using object reference can be called object Dependent
object Independent	

Diff blw Static Variable vs instance variable.

```
class farmer{
  private float pa;
  private float td;
  private float si;
  private static float ri;
  Static ____ it is same for all farmer
{ (for initialize)  so. in memory once it will
  ri = 2.55;          set
}
Void input (){
Scanner scan=new Scanner (System.in);
  S.o.p("Kindly enter principal amount");
      pa= scan.nextfloat();
```

```
S.O.p("kindly enter time duration");
td = scan.nextFloat; }
Void compute()
{
si = (pi * ri * td)/100;
}
Void disp(){
S.O.p("SI is "+si);
}
main(){
farmer f1 = new farmer();
farmer f2 = new farmer();
f1.inp();
f1.disp();
f1.compute();
f2.input();
f2.disp();
f2.compute;
```

(method) for(every class)



(mature java developer)

---

whenever common copy of data has to shared among all objects of class they are not specific to any object then that data is used to store in static.

Static method also called as (generic method.)
non-static method also called as (specific method).

Inheritance :- Code Reusability.

Relationship :-

i) IS-A → Inheritance → Base class - Subclass
parent - child relationship

ii) Has-A - Dependency injection (Aggregation & Composition)

one class acquiring properties and behaviour.
to another class. ⇒ Inheritance.

⇒ un-related. class.

⇓

we developed relation. by using extends.
keyword.

```
Class Demo1 { // parent / Base / existing.
String name = "Vinay";
 int age = 28;
  Void disp(){
   S.o.p("Demo1" + age+name);
  }
 }
Class Demo2 extends Demo1 {
  // child class / derived Sub
}
main(){
Demo2 d= new Demo2();
     d.disp();
```

Important Rules : (Inheritance / IS-A part)

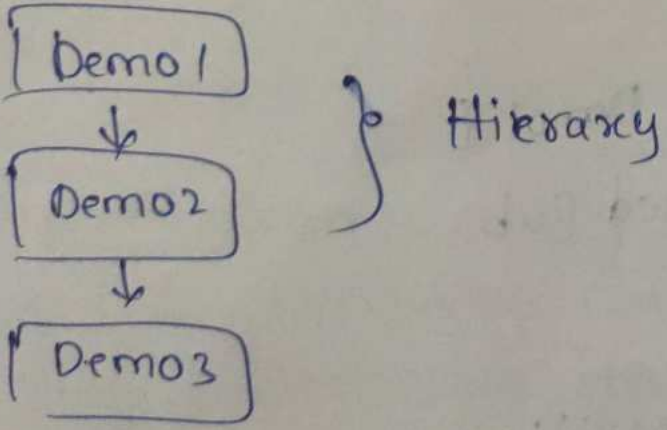
i) Single Inheritance is allowed (one class extends another class).

ex:-

```
Class Demo {
    String name = "Vinay";
    Void disp(){
    S.o.p (name);
}
Class Demo2 extends Demo1 {
}
main(){
Demo2 d= new Demo2();
   d.disp();
```

ii) Object class is parent of all classes.

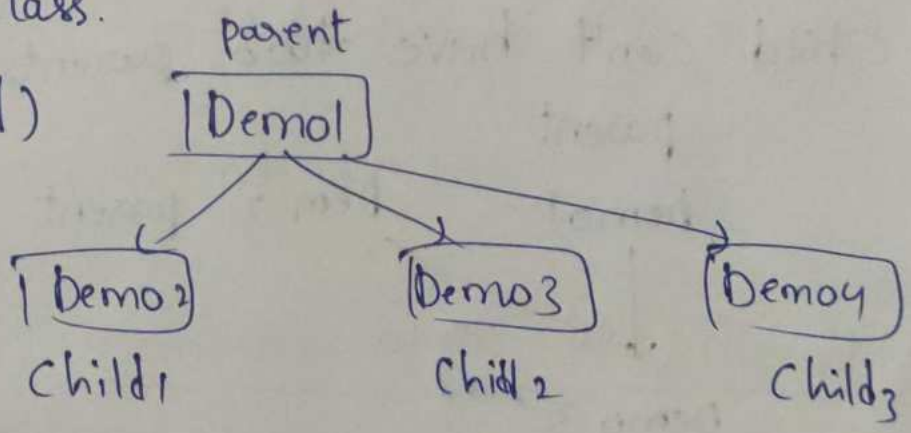
iii) Multilevel inheritance is allowed.



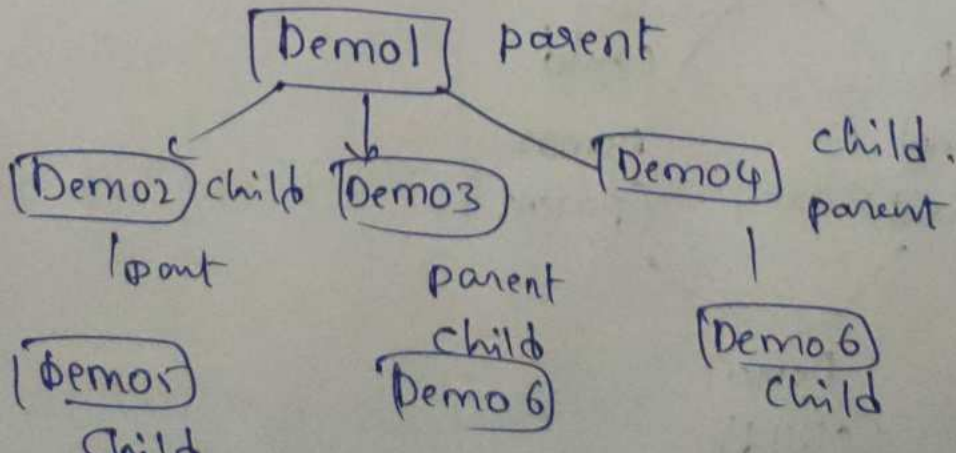
ex: Class Demo1 { // extends Object class.
String name = "Vinay"; properties and
Void disp(){ methods
S.o.p (name);
}

```
Class Demo2 extends Demo1
{

}
class Demo3 extends Demo2
{

}
main(){
  Demo3 d= new Demo();
    d.disp();
```

(iv) One parent class can have any number of child class.
(Hierarchical)



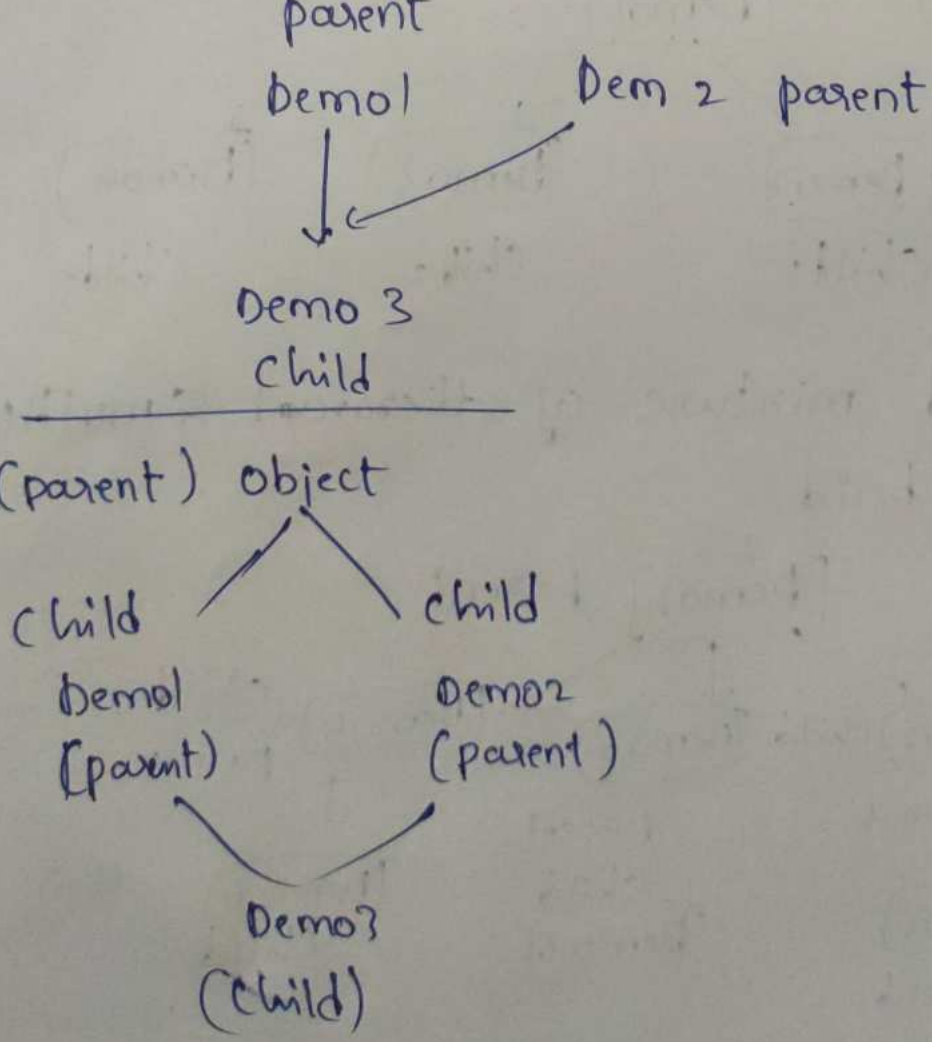
v) Hybrid mixture of hierarcyal & multilevel is called Hybrid.



tx:-

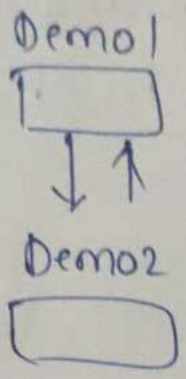
```
Class Demo1 {
  String name = "Vinay";
    Void disp (){
  S.O.p (name);
  }
Class Demo2 extends Demo1 {
  }
class Demo3 extends Demo1 {
}
```

vi) multiple inheritance is not allowed.
(one child can't have two parents (Fathers)



To achive multiple inheritance we can use interfaces.

vii) Cyclic inheritance is not allowed



Sometimes Demo1 will say he is parent. } ⊗ Not allowed
and Demo2 → will say I am parent.

Ex:-

```
Class pen1 extends pen2 {
}
Class pen2 extends pen1 {
}
```

viii) private members of a class doesn't participate in inheritance. (→ why (→ To preserve encapsulation)

```
Class parent1 {
  private String name;
    Void disp()
  {
   S.o.p ("parent");
  }
}
Class child extends parent1 {
    Void disp2()
{   name = "Hyder"; → error because private
}}
```

private member not allowed to avoid directe access (encapasulation following)

* Constructor ⇒ will not parcipate in inheritance. Because constructor will participate while object creation.

But parent class constructor will get called. Because of super() present in child class

ex:

```
Class Demo1{
  String name;
Demo1()
{
 S.O.P("parent");
}
Void disp(){
 S.O.P("name");
}
Class Demo2 extends Demo1{
      Demo2(){
      Super(); ⟶ To call parent class constructor.
    (by default it is presnt)
}
main(){
Demo2 d = new Demo2();
    d.disp();
```

Constructn extends (Super(), this()) in inheritance.

```
class parent {
  int a, b;
parent (){
  a=10;
  b=20;
S.o.p ("parent constructn");
}
parent (int a, int b){
   this.a=a;
   this.b= b;
   S.o.p ("parent para constructn");
  }
 }
class  child extends parent {
    int x, y;
  child()
  {
    super();
    x=100;
    y=200;
  S.o.p ("child constructn);
 }
  child (intx, inty){
    this.x=x;
    this.y=y;
  }
```

constructn in parent class

calling parent class

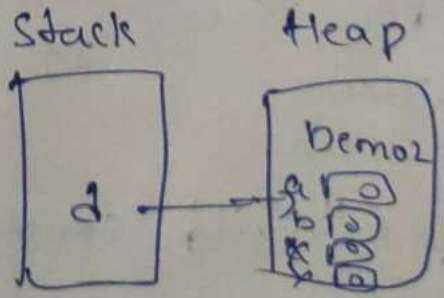
Call to parent constructn. It will not visible in constructn by default.

```
Void disp()
 S.O.p (a);
 S.O.p (b);
 S.O.p (x);
 S.O.p (y);
}
main(){
 Child d= new Child();
    d.disp();
```

---

i) Constructor Zero parameter in inheritance.

```
Class Demo1 {
   int a,b;
    Demo1(){
    a=10;
    b=20;
  S.O.p("parent Constructor");
 }
}.
 Class Demo2 extends Demo1{
   int x,y;
   Demo 2 (){
    Super();
   x=100;
   y=200;
  S.O.p ("child Constructor");
 }
}
```



```
void disp(){
S.O.p(a);
S.O.p(b);
S.op.(x);
S.O.p(y);
main(){

Demo2 d= new Demo2();
    d. disp();
```

Output:

```
parent Constructr.
Child Constructr.
 10
 20
 100
 200.
```

Parameters in Constructr example.

```
Class Demo1{
    int a,b;
Demo1(int a, int b){
    this.a=a;
    this.b = b;
S.O.p ("parent Constructr");
Class Demo2. extends Demo1() {
    int x,y;
Demo2 (int x, int y);
this.x=x; Super(x,y);
this.y=y;
```

```
    S.O.p("child constructor");
}
- void disp() {
    S.O.p(a)
    S.O.p(b)
    S.O.p(x)
    S.O.p(y)

main();
Demo2 d = new Demo2(100, 200);
        d.disp();
```

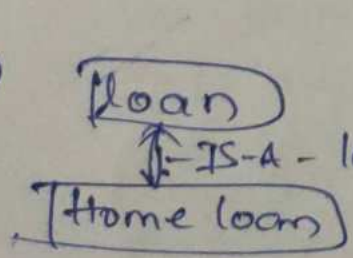
14/011/22

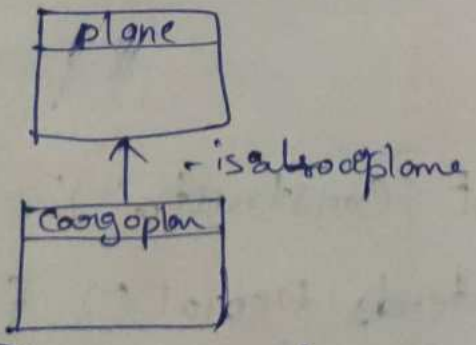
focus ⇒ skills. }

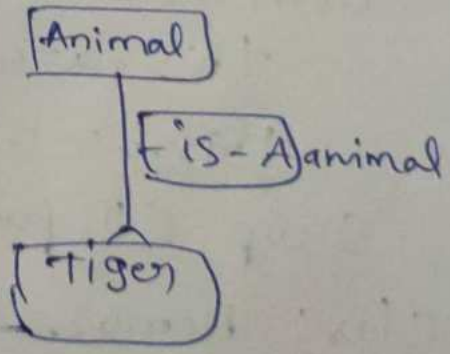
Inheritance: process of one class (parent) acquiring properties and behaviours of another class (child). (hierarcy)) (Extends Keyword)

IS-A relationship - parent-child relationship

→







⇒ loan - personal
→ Education
→ Car
→ Home

} common data we can write in parent class. (Code reusability)

Access-modifiers and Access specifiers.

Access-specifiers - 4 type
1) public
2) protected
3) default
4) private.

applied - class, method, constructor, Variable

if you not write anything (it is a default)

```
void disp() {
}
```

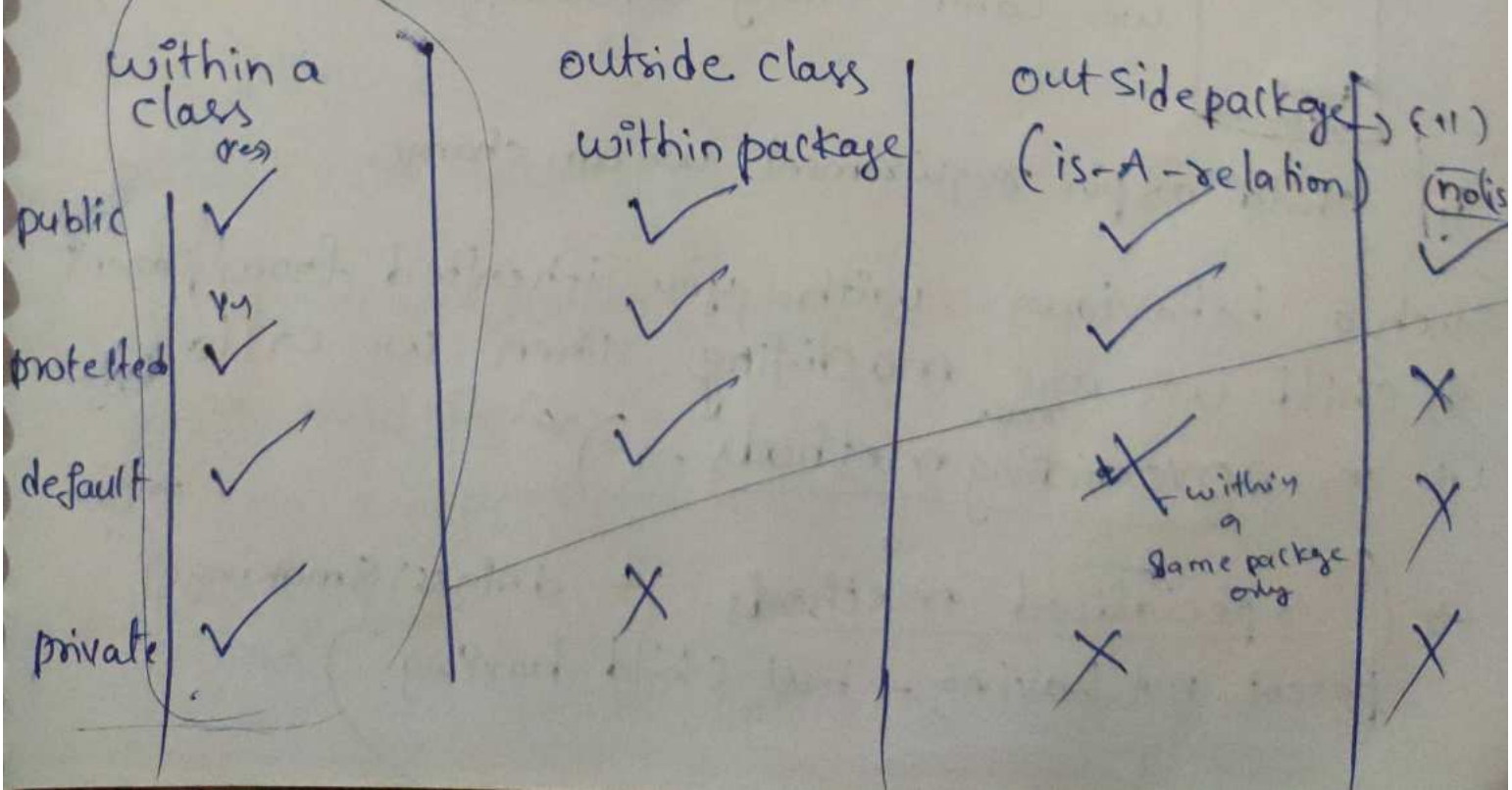
project ⇒ Multiple functionality ~~private~~

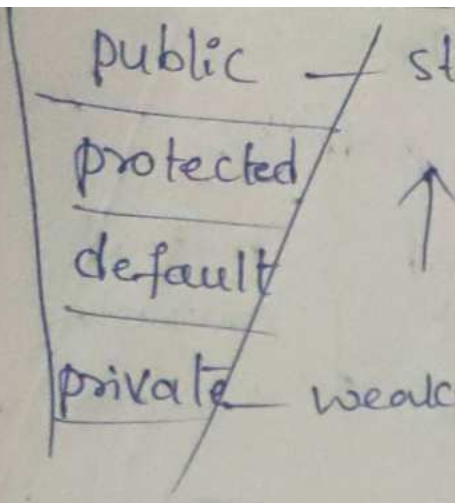
Ex:- ↳ Addition
↳ Subtraction
↳ multiplication
↳ division

when we are doing project we will divide into multiple ~~project~~ parts (or) modules

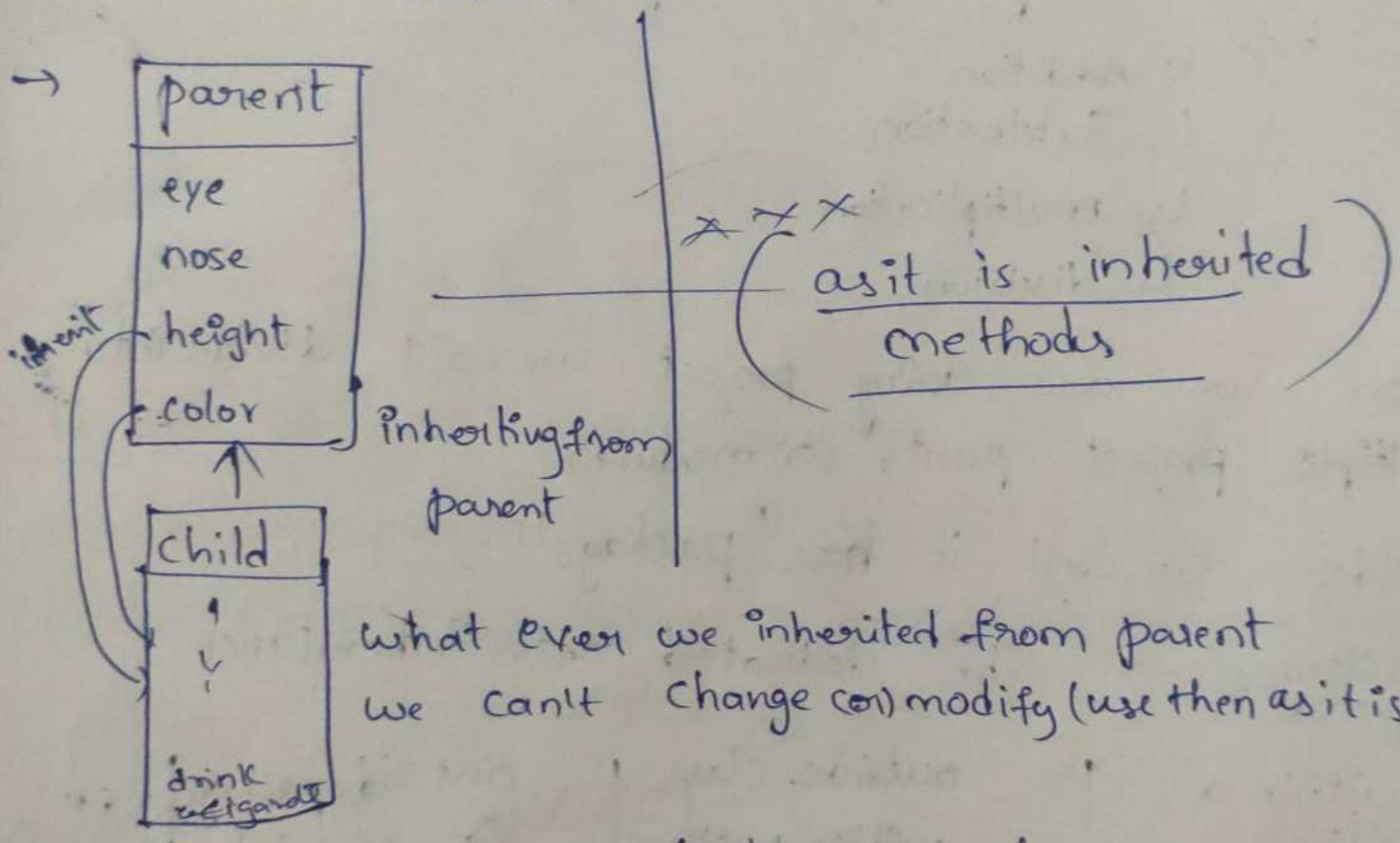
folder we call it has "package"

packages → 3hr dedicated After inheritance.





→ All the methods and constructor "public" is high recommended.



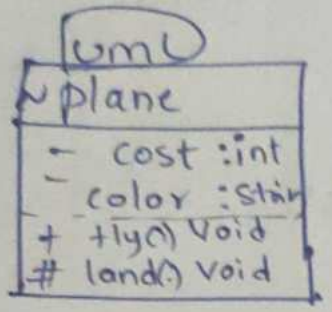
in child as per requirement we can change.

(Such a behaviour with you inherited from parent in child we are modifing then we called it as Overriding methods →)*(>

→ (Specialized methods - drink() Smoking()
parent not having - but child having.) ∞

tx:- Symbols (+, #, -, ~) (Unified modeling language - pictorial rep of what code your excute)

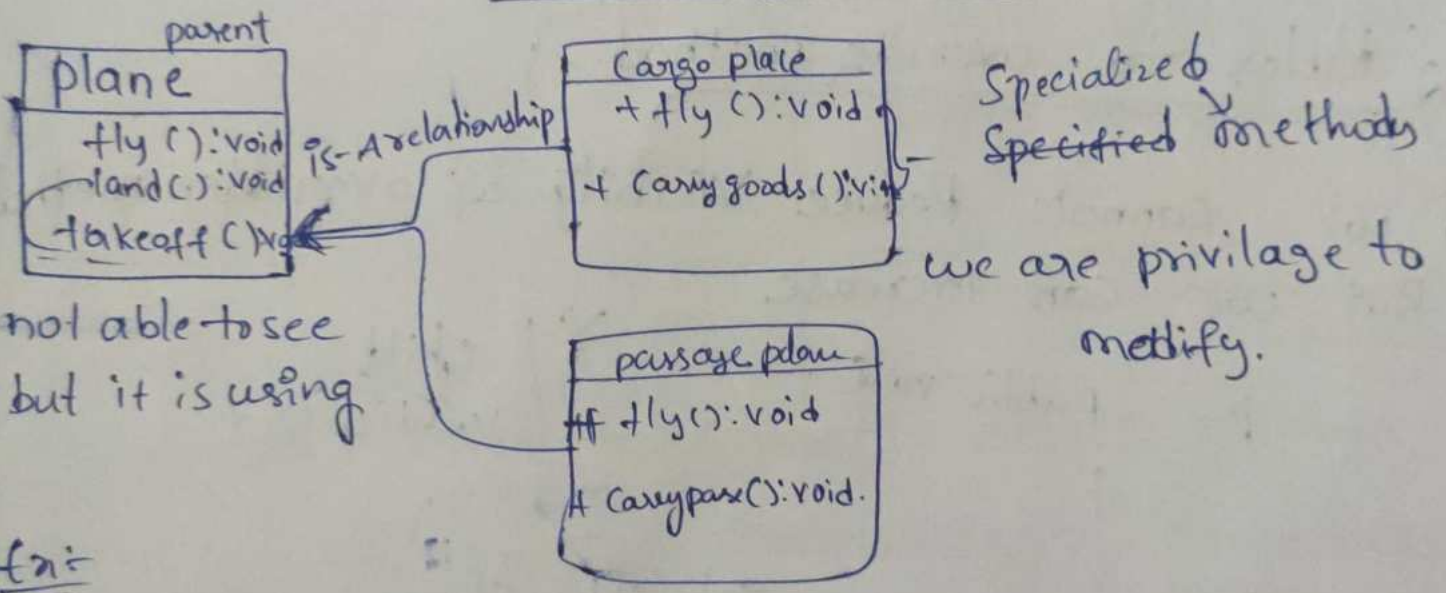
+ → public , # protected , ~ → default , - private



→

class

```
class plane {
    private int cost;
    private string color;
    public void fly() {
    }
    protected land() {
    }
}
```



ex:-

```
class plane {
    public void takeOff()
    {
        System.out.println("plane is taking off");
    }
    public void fly()
    {
        s.o.p("plane is flying");
    }
public void landing()
    {
        s.o.p.out.println("plane is landing");
    }
```

Class Cargo plane extends plane
{ public void fly(){ — overiding
S.O.P.("Cargoplane flies at low hieght");
} }

Class passenger plane extends plane

} right now ~~your files~~ one package 3 class

carry goods.

→ who is parent of All Classes.
→ Object class.

⇒ {parent → child → inherited ⇒ overiding ⇒ Specialized}

imp { Rules to overide method. }

① we cannot Reduce visibity of overridden method
But we can increase

p- public void

}

child
void disp() {

}

In Java so many inbuilt class
we should increase visibility of access specifiers.

② Return type of (child) overridden method must be same ~~tts~~ that of overriding (parent) method.
Return type we cannot Change.
possibility of Change of Rectrn type.

③ Return type of overidden method in child class can be different as that of parent — loans, personal loans
if it is co-variant return type (return type is-A relationship)
(public parent int val)
(return it = new subclass)
return it

④ parameters of overidden method must be same that of parent else it will be consider as Specilied method. Consider (method overloading)

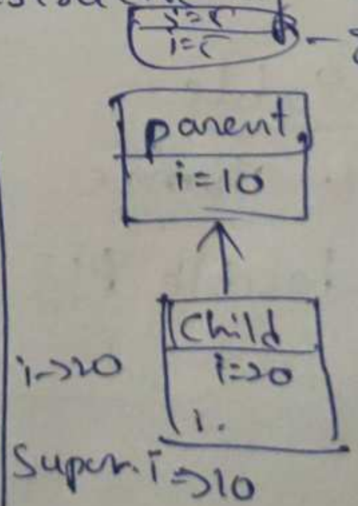
-(method) ✓
Super(); → inside constructor.
→ it will call parent class constructor

super; → keyword.

ex! public class void main(int a, int)
{
}
class — —
public void main(int a).
Specialized method

this; → refer to current object runtime

this(); → calling constructor of same class



— o —

```
Class Demo 1
{
  int age =26;
}
class Demo2 extends Demo1
{ int age=28;      (28) - normal
  Void()
  S.O.P (super.age);
}  = Super.age;   (26)
```

Super.i = 10
super.super.i = 5 ⓧ invalid

→ when ever we want invoke instance variable of parent class. this super keyword we will use.

super;
→ calling instance variable of parent class
(super. variable;)

super();
→ to calling parent class constructn.
by-default constructn having super method.

(*) final keyword. (class - to create class)

final → apply on class (new - object creation)
→ method
→ variable ← Abstract

final class :- will not inherit. not nothing.

final method :- will get inherit, but method will not overidding.

final variable :- acts as constant we can't change.

ex: final double pi = 3.14;

15-11-2022

→ polymorphism ①, Abstraction

↓ ②

abstract

↓

interfaces ④

functional ⑤ interfaces

Lambda ⑥ Expressions

IS-A - inheritance

③ HAS-A - Abstraction

→ polymorphism → poly - means - 'many'

(many forms) morphism - means - forms

(one : many)

Compile time polymorphism - method overloading

→ no. of paramter

→ order. datatype

→ datatype of parameters.

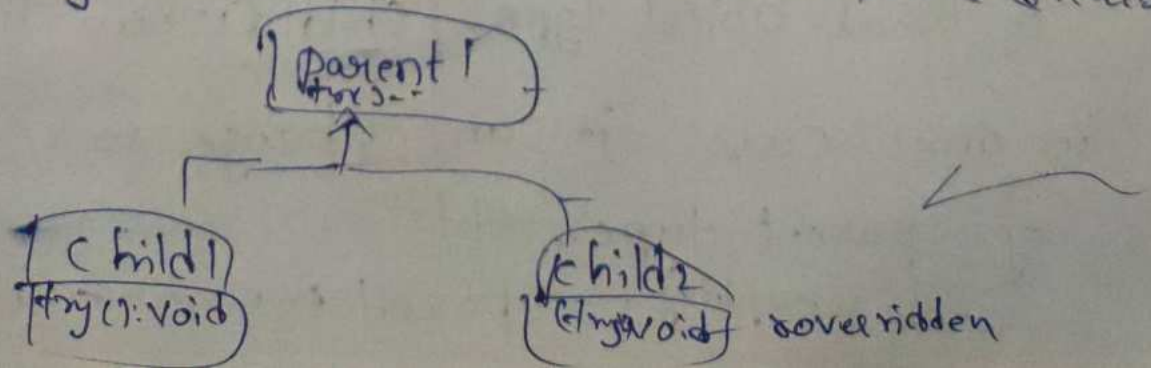
(pm - polymorphism)

→ Run time polymorphism. (code size reduce)

also called → True (pm)

↳ method overriding.

* Advantage code size reduces, and (flexibility)



ex:- (non-polymorphism)

```
class parent1 {
    public void cry() {
        S.o.p("parent crying");
    }
}

class child1 extends parent1 {
    public void cry() {
        S.o.p("child1 cries at low voice");
    }
}

class child2 extends parent1 {
    public void cry() {
        S.o.p("child2 cries at high voice");
    }
}

main() {
    child1 c1 = new child1();
    child2 c2 = new child2();
    c1.cry();
    c2.cry();
}
```

overriding method

same

overridden method

same

parent ref;
ref = c1;
c1.cry();

type of creating like this tightly coupling

//output- child1 cries at low voice
child2 cries at high voice.

whenever we create object the reference type same of that object type. ex:- child1 c1 = new child1()

but in one case it can change to reference type of parent type. ex:-

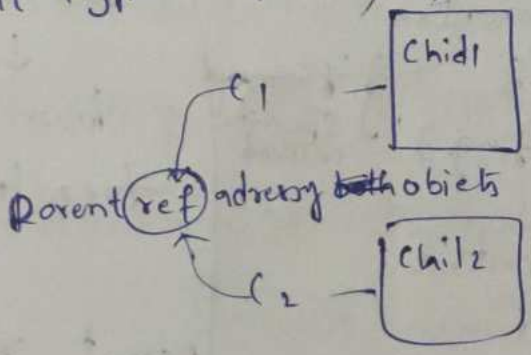
loose coupling: parent1 p1 = new child1();

Parent ref
ref = c1;
(loose coupling) - runtime polymorphism
Car ref.cry(); - same line
ref = c2
ref.cry() - same line
} doing many forms
multiple tasks.

1) Rule

→ to achieve polymorphism is whenever you create object that type of object must be parent type.

(Creating parent type reference) 123



at a time only one address it will hold

new value come old value will go

Ex: program.

```
class plane {
  public void takeOff() {
    S.o.p("plane is taking off");
  }
  public void landing() {
    S.o.p("plane is landing");
  }
  public void fly() {
    S.o.p("plane is flying");
  }
}

class Cargoplane extends plane {
  public void takeoff() {
    S.o.p("Cargo is planing");
  }
}

class passengerplace extends plane {
  public void fly() {
    S.o.p("passenger is planing");
  }
}

class Airport {
  public void permit(plane plane) {
    plane.takeoff();
    plane.landing();
    plane.fly();
  }
}
```

parent reference
parent type.

~~Airport~~

```
main(){
Cargoplane cp = new Cargoplane();
Passengerplane pp = new passengerplane();
Airport A = new Airport();
   A.permit(cp);
   A.permit(pp);
   A.permit(
```

---

ex:

```
Class Parents{
public void cry(){
   System.out.println("parent crying");
}
}
child1 extends parents{
   public void cry(){
   SO.p(child1 crying);
}
}
child2 extends parents{
   public void cry(){
   S.O.p(child2 is crying);
}}
main()
parents p = new child1
   p.cry();

parents p = new child2
   p.cry();
   p.eat(); - this Specialised method
```

So we have have call By changing the behaviour of parent

```
((child1)p).eat();
```

```
((child1)p).eat();
   down casting
```

if

-to achieve polymorphism

```
parent p = new child1()
```

↑

In child type object creating parent type before is called Upcasting.

(o) lose couplity

```
((child)p).eat();
```

↓

down casting

to called Specilized method

parent ref temporilarly converting to child to access Specilized method down casting

```
Class Dog{
   -
   -
}
Class Shenu extends Dog

Dog dog = new Dog();
Shenu s = new Shenu();
```

Abstraction

① Can be achieve by ① using abstract keyword (100% (or) 10% may be not).

② But using interface you can achieve 100% Abstraction.

→ hiding actual implementation showcasing only the feature.

→ Ex: ATM - we don't know implement: but we getting feature

→ Abstract method which does not have body (or) implementation. only Body Signature is there.

→ In a class if one method also abstract method then that we should write class also abstract

Ex: abstract class plane {
    abstract public void take off ();
}

→ abstract key word can be used to method.

abstract method a such method implementation is not there only method signature.

Abstract key word can be used to class and method.

Abstract keyword can't be used to Variables.

tx:- abstract class loan1{

```
abstract public void dispInt();
}
class Homeloan1 extends loan1.{

}
```

→ We can create ref of abstract of class.

loan1 l=new Homeloan(); ✓

→ we can't create object for abstract class.

Loan1 l= new loan(); ✗

→ Abstract class have both abstract & normal method.

Abstract class have → all normal methods.

Abstract class have all abstract method.

The subclass/ child class is extending abstract class then either have to implement abstract method.

(or) declare that class abstract

→ Constructor can't be made abstract.

because already super(); method is there inside ∴ so abstrat rule implementation hiding but super method is there.

→ abstract → incomplete class ;

(we can't create object. → inherit concept

) child class will inherit and create object

Can we make abstract class as final ?
N/o. final class will not participate inheritance
~~heri~~.

abstract method: it will inherit not overidden.

Variable ⇒ abstract ⇒ ⊗

→ abstract cla constructor will call child super();
parent class.

Can we have constructor in abstract class - yes

How

```
abstract class demo1 {
    if it not include constructor
}
class Demo2 Extend Demo1 {
    Defaut (0) parameter
               constructor.
    }          super();
```

Through IS-A relationship
relating and building relation blw class

through HAS-A relationship
building Communication and data navigation
blw class.

Imp ✓ HAS-A Relationship

→ IS-A relationship ex: inheritance
parent-child relationship
we can achieve this by using "extends"
key word.

Ex:- ① class Engine { → CAR HAS-A Engine
}
class Car {
}

② class Address { Dependant object
} → student HAS-A Address
class student { // Target Object
// HAS-
Address address;
}

③ class Account { Employee HAS-A Address.
}
class Employee {
Account account; — Variable declaration
} that Variable Should be
initialized.

→ class Engine {
}
Class Car {
HAS-A relationship
Engine engine; = new Engine(); // instance Variable
}

HAS- A - Relation ship → to establish another class ref should be a part of this another class ref.

→ The engine class object created and injected to Car class

ex:- class Engine { // Dependant Object

}

class Car { // Target Object
//HAS-A-relationship
Engine engine; // instance variable

Dependancy Injection :- The process of injecting dependant object into target object is called as.

" Dependacy Injection :-

→ In how many instance Variables will initialize two ways.

a. Constructor dependancy injection

b. -) Injecting dependant object into target object through a constructa.

b. Setter denpendancy injection

→ Injecting dependant object into target object through a setter.

Relationships in Java

As part of Java application development, we have use entities (class) as per the application requirements..

In Java application development, if we want to provide optimizations over memory utilization, code Reusability, Execution Time, Sharability then we have to define relationships b/w entities.

→ There are three types of relationship b/w entities

1. HAS-A-relationship (extensively used in projects) ✓
2. IS-A relationship ✓
3. USE-A relationship (not popular) ✗

Q) what is the difference b/w HAS-A relationship and IS-A relation ship

Ans: HAS-A relationship will define associations between entities (class) in Java applications, here associations b/w entities will improve Communication b/w entities and data navigation between entities.

IS-A Relationship is able to define Inheritance b/w entity Classes, it will improve "Code Reusability" in Java applications.

association: Relating two class. in HAS-A Style

We can achive HAS-A - Relationship

Associations in JAVA.

There are four types of associations b/w entities

1. One-To-One Association (1:1)
2. One-To-many Association (1:M)
3. Many-To-one Association (M:1)
4. Many-To-many Assiciation (M:M).

(Target object make private variables)

To achieve associations between entities, we have to declare either single reference or array of reference variables of an entity (class) in another entity (class.)

fn:-

```
class Address {
    --
}
class Account {
    ---
}
Class Employee {
    --
    Address[] addr; // it establish One-To-many Asstiation.
    Account account; // one-to one Association
}
```

(reference variable)

employee has account
employee has many adhrs
current, home

⇒ reference Variable of one class inside one mode class - Association 1:1 one to one

relation relating two class
communication b/w two class will build.

→ many adhrs associated with employee
→ that be in Array one to many.

Code reusuability :

```
class object {
    public boolean equals (object obj){
    }
    public String tostring(){
    }
}
class student {
}
System.out.println(new Student());
```

Some one wrote code fa tostring already we reusing to this

– internally tostring method called

Relating two class

HAS A relationship in creating reference Variable in Target class → Two types → 1:M (+) 1:1

array — variable

→ Java project — package.

This is primitive.

```
Student class {
attributes of student
privite String sname;
private Integer sage ;
private Double ssalary;
        Integer sid;
}
```

wrapperclass keeping

{ - Instance Variables.

two ways we can Set a value

Constructn,

Setter

'generate Constructor

generate getter and Setters..

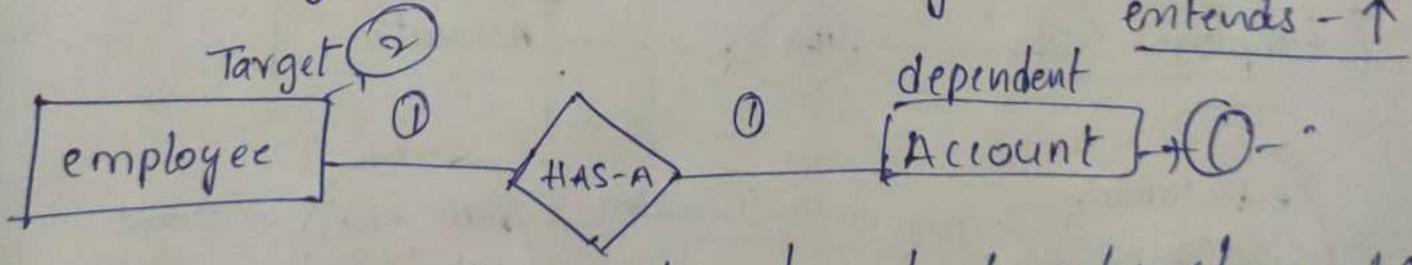
→another class TestApp
printing student object. toString method called bak
So we should overriden toString();

→ anywhere two class used established association (no)
But we used constructd only. even though ~~no~~ to initialize instances variables but those variable even though they are object type not of usr defined. they are predefined.
string, Integer, - prefined - I didn't wrote code

output :- Student [sname=Vijay, sid=10, sage=23],
typeobjelt

Associations :- (1.) one to one AssociationMapping.

tg:- Every employee have only one Account
extends - ↑



First we have create dependent Objelt and class
Accont
then we have inject to target object.
(employee)
if Account is ready So. we can give to employee.

→ In.ineuron.bean-) - Account, employee;
in.ineuron.om.

→ Dependent Object code is ready injecting value by through constructo.

• printing employee and Account details

→ if they are private Variables, How we can access and print..

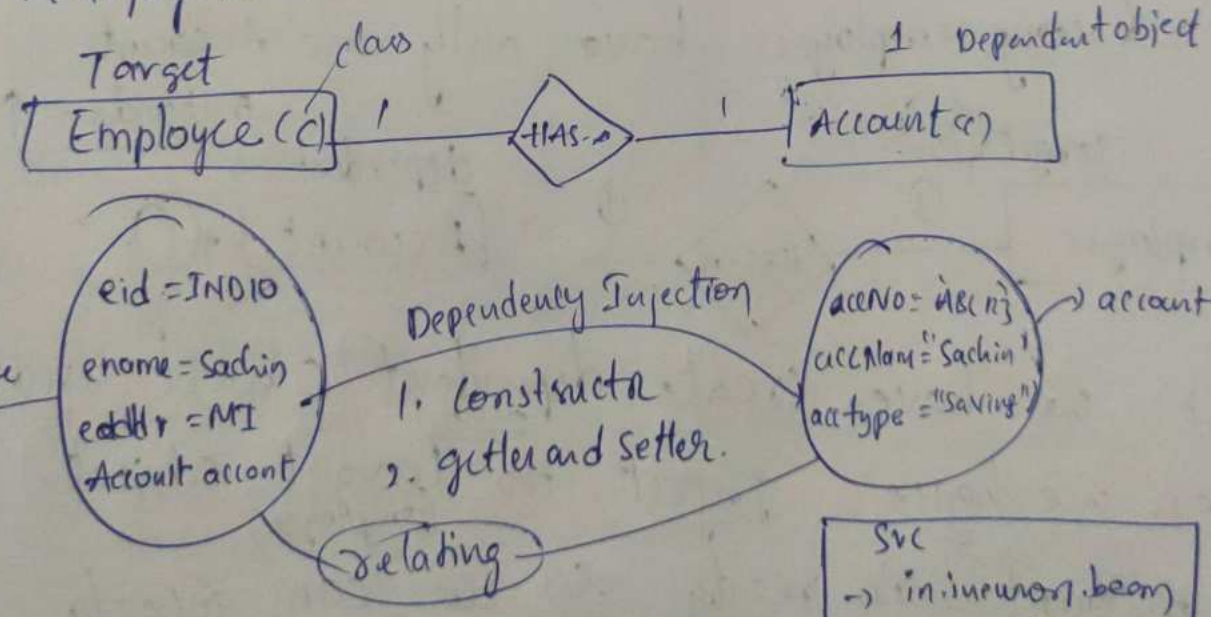
remove private. call ((accounti. acctype;))

-then in main method.

first create Account Object. (inject Values)
then Employee Object we are injecting account through Constructo. of employee..
then call method to print details.

we have import class to main method.

in.vijay.bean.. Account;



we are getting details of account in employee. iscalled data navigation.

Communcation and data navigation

Src
→ in.ineuron.beam
Account
employee
→ in.ineuron.main
→ TestApp.

1. One to One Association.

It is a relation b/w entities, where one instance of an entity should be mapped with exactly one instance of another entity

eg: Every employee should have exactly one Account.

---

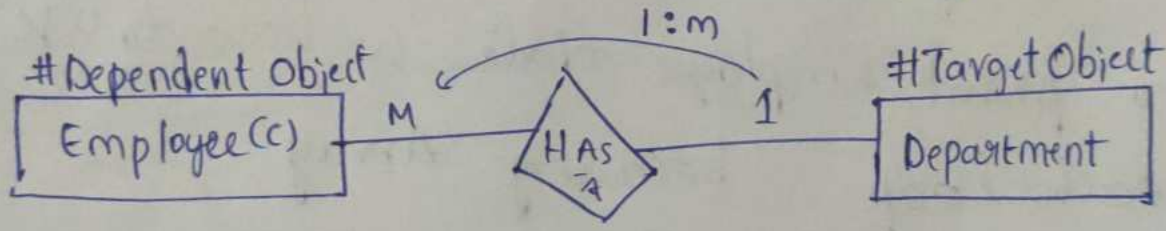
② One to many Association:

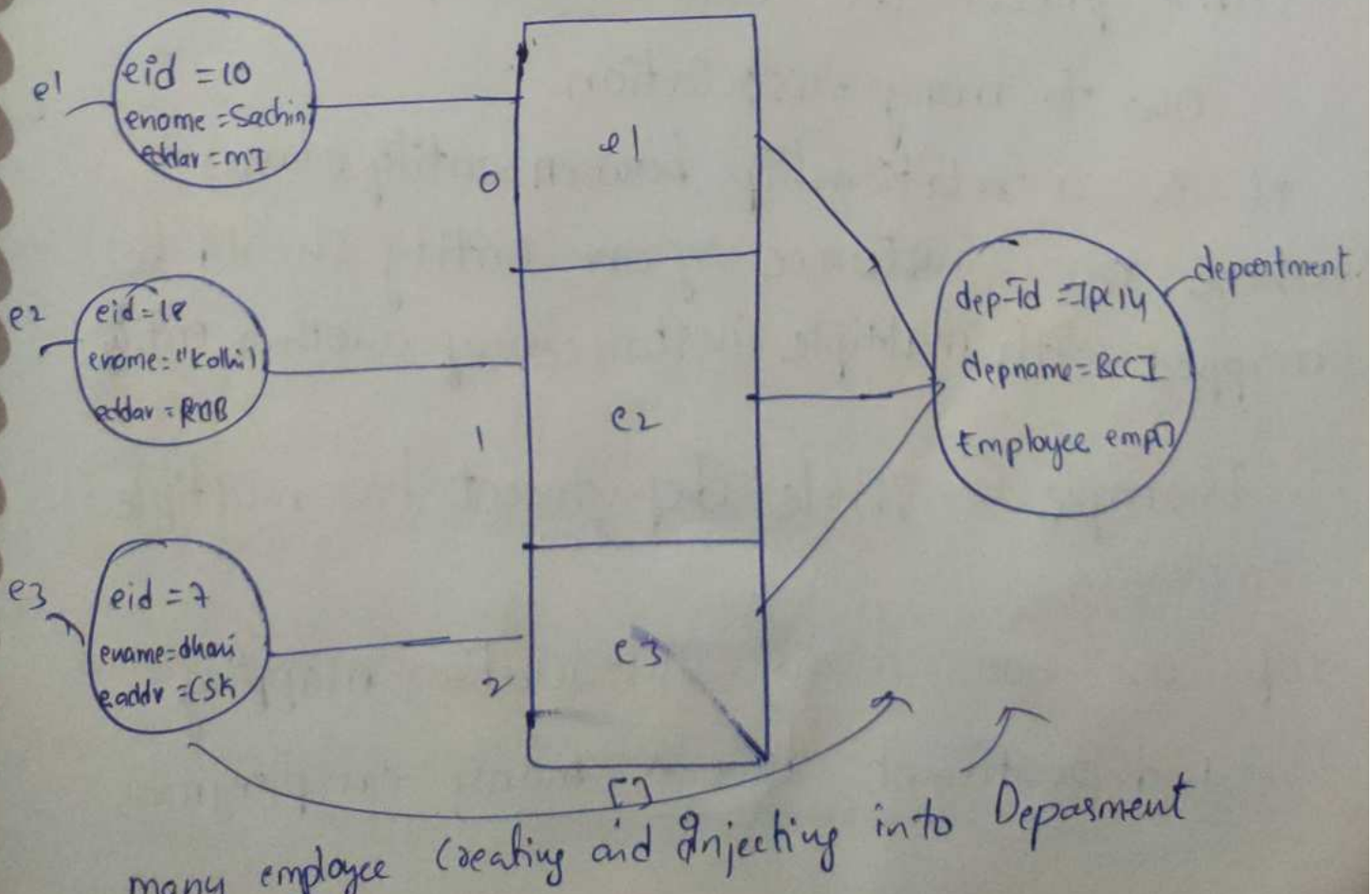
*the Single department has multiple employees.

One to many

many means (data is arrays)

Employee [] emps;





first we have to create employee Class and Object

then, department class

we have to create employee [] emps - reference of employee (arrays) through constructor.

→ then department class. we have to inject reference of employee class.

→ main method. we have first employee object. (employee.ename)

the array with objects.

→ to print employee date we have to use for each loop. because Array same process as one to one process

→ One to many Association.

It is a relationship between entity classes. where one instance of an entity should be mapped with multiple instances of another entity

Example : Single department has multiple employees.

Ref :: 05- one - many - Association mapping.

1 department has ↔ many employees

3. Many-to-one Association. (like (1:M))

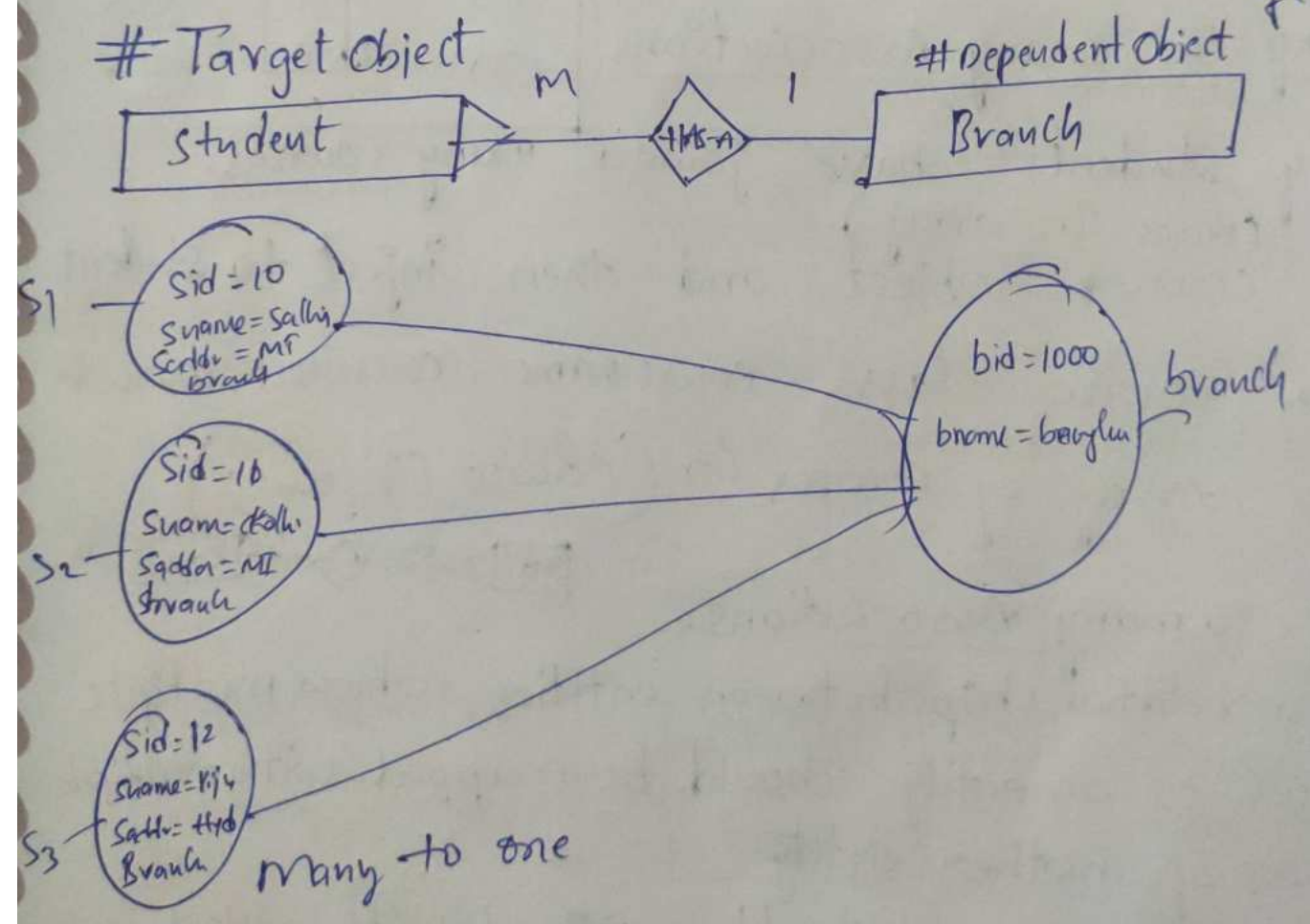
Eg: Multiple Student have joined with a single branch.

dependent object is branch.

Target object is Student

first we have create branch class and then use reference in Student class

Setter inject is bit costly. - means more lines of code



many Students getting into branch.

process ① create branch class first

② create student then reference use in Target class

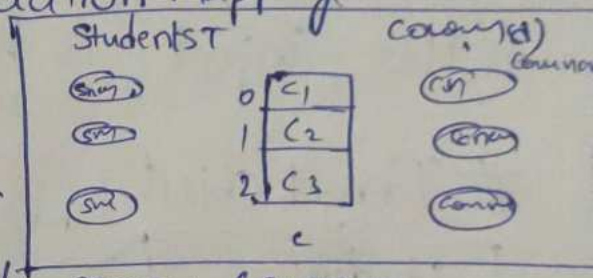
In main method give values by setter and getter. method chaining. ~~branch~~ student. ~~refer~~ IdE

→ Many-To-one Association.

it is a relationship between entities, where multiple instances of an entity should be mapped with exactly one instance of another entity.

ex:- multiple student have joined with a single branch.

eg:- refer :: of-many-one-Association mapping



(4)

→ Many-To-many Associations.

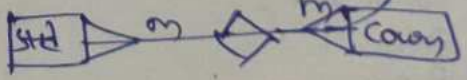
many students have joined many courses.

(courses in array) →

first courses object and then inject to student

create course class. course name, course cost, course id

create Array of courses. (eg (Course [] cour;)



Many-TO many Associations.

it is a relationship between entities, where multiple instances of an entity should be mapped with multiple instances of another entity

ex:- multiple students have joined with multiple courses.

eg:- of-many-many-Association-mapping

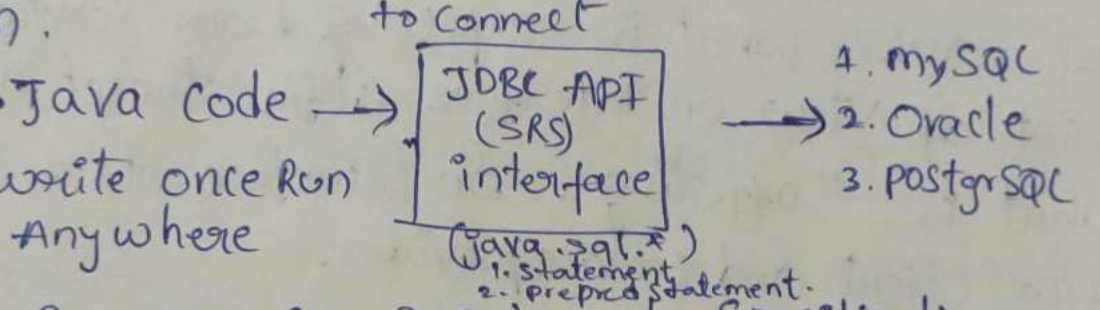
Interface :- It's imp topic to server side

Any SRS (Service Requirment Specification) is called as Interface.

data base means where we can store data storage medium that keep data, in table form record files.

ex:- Sql, mySQL, Oracle, postgreSQL

→ ① There are 3 data base to connect with them.

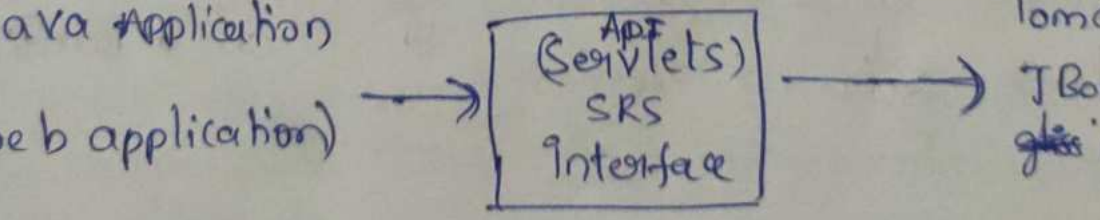


this is service requirment specification

to promote this Sun people gave us API to connect JDBC (Java database Connectivity)

API → Collection of .class files

②



to run web Applications we need Special Software (Servers) will not Jvm

→ .class Jvm (part of JDK Software)

→ .class web Application → Jvm (part of Servers Software), tomcat.

→ Sun people responsible to define JDBC API and database Vendors will provide implementation for that.

Buiding Banking Application

Bank is giving some services to customer
Customer will not contact java code..
So. GUI will come to picture(graphical user Interface) blw.
Connect java code and Customer.
GUI- interface using this interaction is happening blw Application and Customer.

Def 2:- From Client point of view an interface define the set of services what is expecting from Service. provider point of view an interface define the set of services what is "offering". So interface acts a contract blw client and serviceprovider.

eg:: GUI Screen of ATM defines the set of services what the customer is expecting,
Bank people offered the same set of services what the customer is expecting.

Customer ⇒ GUI ⇒ Bank. - contract blw customer and Bank.
eg: ↳ GUI ← offering
expecting
client

Def 3:- Inside interface every method is always abstract whether we are declaring or not hence interface is considered as 100% pure abstract class.

fn:- Interface Account
{
// it is 100% abstract class
// the methods are "abstract and public" by default
Void withDraw();
Void deposit();
Void checkBalance();

Summary :- interface corresponds to service Requirement Specifications (SRS) or Contract b/w client and service provider (or) 100% pure abstract class.

→ fn:- use Isample - interface with pascal convention

ISample - to give class name take sample and add Impl → SampleImpl - class name

Declaration and Implementation of interface (Block).

i) whenever we are implementing an interface compulsory for every method of that interface we should provide implemenation otherwise we have to declare class as abstract in that case child class is responsible to provide implementation for remaining methods.

ii) whenever we are implementing an interface method compulsory it should be declared as public other it would result in compile time error (decreasing visibility).

Ex:-

```
Interface Isample{
    Void m1();  // 100% abstract class.
    Void m2();  // methods by default abstract and public
}
Class SampleImpl implement Isample{
    public Void m1();{
     S.O.p ("m1()");
    }
    public Void m2(){
     S.O.p("m2()");
    }
}
main(){
Isample Sample = new SampleImpl();
     Sample.m1();
     Sample.m2();
}
```

Realtime coding Hiverarcy (Developers approach).

interface (100% abstraction)

↑ implements

abstract class (not 100% abstraction)

↑ extends

concrete class (no abstraction).

Ex:-

```
Interface Isample{
    void m1();
    void m2();
}
abstract class IsampleImpl implemets Isample
    public void m1(){
      S.O.p("m1()");
    }
    void m2(); → By default abstract.
}
class SubIsampleImpl entends IsampleImpl {
    public void m2(){
      S.O.p("m2()");
    }
  }
  main(){
  Isample Sample = new SubIsampleImpl();
        Sample.m1();
        Sample.m2();
```

———— o ————

Inner classes:-

→ The classes that are declared inside another class are called as inner class or nested class.

1) Inner classes have a special type of relationship that it can access all the members of outer class including private.

2) More readable and maintainable.

3) Inner class requires less code.

4) If a class is useful to only one other class then it is logical to embed it in that class and keep the two together.

→ four types.

→ Regular Inner class - class inside side

→ method local Inner class - method inside inner class

→ Anonymous Inner class - object using creating class

→ Static Inner class. - class inside static inner class

① Regular Inner class. (for inner class .class file is outer $1.class).

```
→ class outer {
    class Inner {
        Void Innermethod() {
            S.O.p ("inner method");
        }
    }
    Void outermethod() {
        S.O.p ("outer method");
```

```
    Inner i= Inner(); } → TO call innerclass. ① type
      i. innermethod ();
  }
}
main (){
  outer o=outer();
    o. outermethod ();
      (or)
outer Inner i=new outer(). newinner(); -) to call
inner class.                                ② type.
```

② Method local Inner class:-

```
class outer {
    Void outermethod (){
      class Inner {
        Void Inner method (){
          S.O.p ("inner);
        }
      }
      S.O.p ("outer");
    Inner i= Inner();
      i. innermethod ();
    }
main() {
  outer o= new outer ();
    o. outer method ();
```

3. Annoymous inner class.

```
class parent1 {
    void msg(){
        s.o.p ("hi");
    }
}
main(){
parent p= new parent();
        p.msg();
parent p1 =new parent() {
        void msg(){
        s.o.p("hello");
    }
};
```

④ Static inner class

```
class outer {
    static class inner {
        void inner(){
        s.o.p ("inner");
    }
}
    void outer(){
        s.o.p ("outer");
    }
}
main(){
Inner i= new Inner();
        i.inner();
}
```

//

21/11/22

Difference b/w : extends vs implements.

①

→ Extends :: one class can extends only class at a time

eg :- class One{}

class Two extends One{},

Interface → Implements: one class implements any no of interface at a time.

ex:

```
interface Ione {
    Void m1();
}
interface Itwo {
    Void m2();
}
Class CommonImpl implements Ione, ITwo {
    public Void m1(){}
    public void m2(){}
}
```

② A class can a single class extends and implement simultaneously if it is first we have ~~implementation~~ Extends. and followed by implementations.

eg :-

```
interface One {
    public Void methodOne();
}
class Two {
    public void methodTwo(){}
Class Three extends Two, implements one {
```