object oriented programming :

classes :- (group of entities (objects)
collection of objects (entities)

objects :- entities in the real world
part of class.

ex: pen → blue } color
object ↘ yellow } property
colours.

Blueprint (1m800) { class
└ { objects

classes :- contains attributes (properties) [colo
+
contains functions (behaviors) [change color

Example :

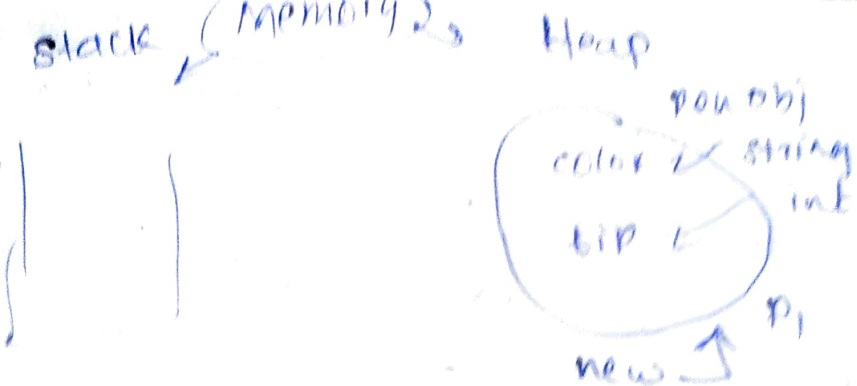
pen

color : String (blue)
tip : 5 (int)
setcolor( )
setTip( ) :

Tip :
classes Names start with Capital letters
function Names start with small letters.

ex :
```
public class oops {
}

class Pen {
```

```
Public class oops {// (main function)
}
class pen {
    String color;
    int tip;
    void setcolor (String newcolor) {
        color = newcolor;
    }
    void setTip (int newTip) {
        tip = newTip;
    }
}
// main function
    public static void main (SAC) {
    }
    pen P1 = new pen(); // constructor (pen)
                        // created a pen object called P1.
    P1.Setcolor ("Blue");
    syso (P1.color);
    P1.setTip (5);
    syso (P1.Tip);
}
```



Example:

Access Modifiers (specifiers):

Access Modifier	within class	within package	outside package by subclass only	outside package
public	Yes	Y	Y	Y
private	yes	N	N	N
Default	yes	Y	N	N
protected	yes	Y	Y	N

Ex:

```
class BankAccount {
    public String username;
    private String password;
default/public void setPass(String pass){
        password = pass;
    }
}
```

```
psvmsA ( ) {
BankAccount MyAcc = new BankAccount();
    myacc.username = "sai";
    myacc.setpassword ("abcd";
        } syso ( )
    }
```

Getters & setters :-

Get :- to return the value (Getters)

set :- to modify the value. (setters)

this :- this keyword is used to refer to the current object.

```
Class pen {
private String color;
private int tip;
    String getcolor () {
        return this.color;
        }
    int getTip ( ) {
        return this.tip;
        }
    void setcolor ( String newcolor ) {
        this.color = newcolor ) }
    void settip ( int newtip ) {
        this.tip = newtip;
        }
```

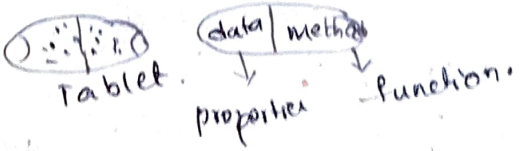
Main function {

```
pen P1 = new pen();
P1.setcolor("Blue");
syso(P1.getcolor());
P1.setTip(5);
syso(P1.gettip());
P1.setcolor("yellow");
syso(P1.getcolor());
P1.settip(6);
syso(P1.gettip());
```

}

Encapsulation :-

Encapsulation is defined as the wrapping up of data & methods under a single unit. it also implements data hiding.

→ (capsule) 

data hiding :- useless | sensitive data basically hides.

By using ~~data~~ Access specifiers data hiding has is possible (or) taken place.

Constructor :-

Constructor is a special method which is invoked automatically at the time of object creation

→ Constructors have the same name as class or structure (C++)

→ Constructors dont have the return type (No void also) (Not even void)

→ constructors are only called once at object creation.

→ Memory allocation happens when constructor is called.

```
Javac filename.java
Java filename
```

Code :-

```
public static vm s+( ) {
    Student s1= new student( ); // ("sai");
    Syso (s1.name);
    }
Class student {
    String name;
    int roll;
Student (String name) { // parameterized constructor.
    this.name = name )
    }
or
student ( ) {
    Syso ( constructor is called); // non parameterized
    }
```

Typed of constructors.

1) Non-parameterized

```
student () {
    sysO ("constructor is called");
}
```

1) parameterized constructor;

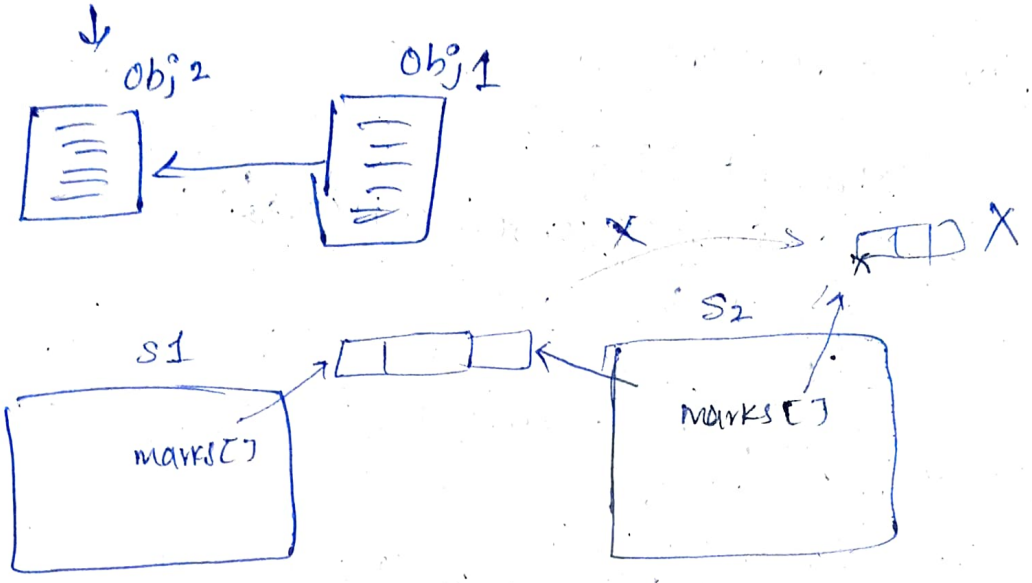
```
student (string name) {
    this.name = name;
}
```

* We can create as many objects until how many constructors we have.

But if there is only one constructor then in the main function only that constructor will work: other than that gives error.

for example :- you have created a parameterized constructor in class and you have created default constructor & parameterized contries in main function. here defaut constructor give error.

Copy Constructor :- By defaut created in C++;



* Imagine that you have Created an array in S_1 and you have copied it into S_2 and after sometime you chave changed the one index value of Array. Now the output will be the copy S_1 values and updated value.

Here. Updation of Array value is done by. firstly S_1 created a array [0|1|2] and copied S_2 in [0|1|2]. Now you have update [2] index. here. the value of S_2(index) is the reference of 2nd index passed to array S_1 2nd index.

Here total array is not copied into anathor array. the reference of S_2 2nd index is passed to S_1 (figure up drawn)

psvm() {
Student s1 = new Student();
Student s2 = new Student(s1);
s1.name = "sai";
sysO(s1.name);
sysO(s2.name);
}

class Student {
String name;
int marks[];
}

Student(Student s1) { // copy constructor.
marks = new int[3];
this.name = s1.name;
}

Shallow & Deep copy :-

Shallow → Changes reflect.
Deep copy → creates new array/object → changes does not reflect.

(Lazy copy)

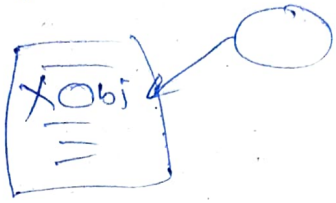
In Deep copy As from the above example. the values are not updated s2.marks[2] = 100; the second index of newly changed value will change. It only copies s1 elements into s2 (only)

Destructors :-

Balance

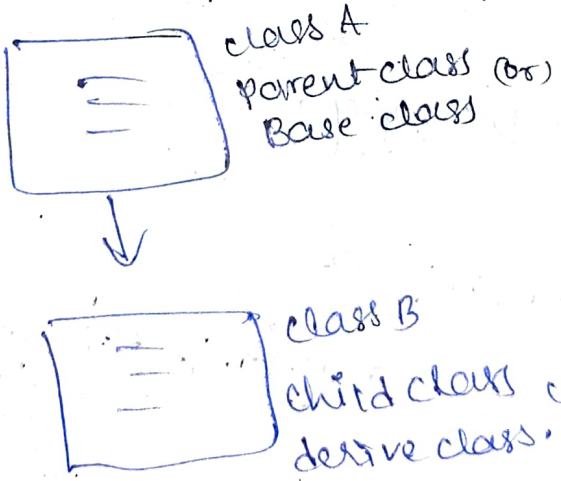
construct — destructor (deletes memory of object)

Garbage collectors / destructors are used to remove or delete the memory allocated by object.



Inheritance :-

inheritance is when properties & methods of base class are passed on to a derived class



Ex:-

Base class

Animal {
eat()
breath()
}

child class

fish {
gills()
swim()
}

Base class
class Animal {
    String color;
    void eat() {
        sysO("eats");
    void breath() {
        sysO("breath");
    }

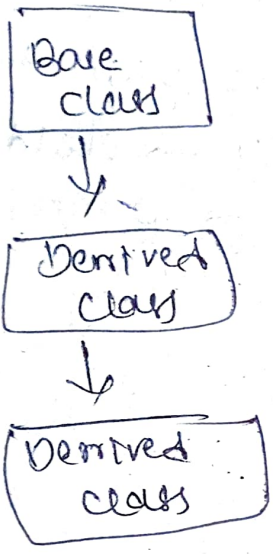
// derrived class. //subclass
class Fish extends Animal
                {
    int fins;
    void swim() {
        sysO("swim");
    }

psvm sA() {
    Fish shark = new Fish();
    shark.eat(); // call
    shark.breath(); // call.
    }.

Single level inheritance

c extends p {
(extends is the keyword used to derrive properties from parent → child class
Base → derived class
)

Multi level inheritance.



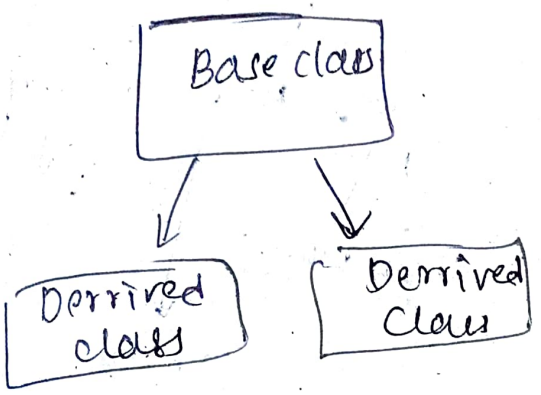
Animal
color
(eat, breath)

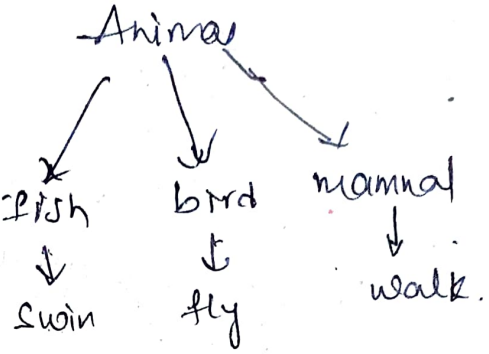
mammal
legs

dog
breed.

```
psvmsA {
 dog d1 = new dog ();
   dog.breed = "hi";
   syso (dog.breed);
   }
class Animal {
      String color;
      void eats() { syso (eats) }
      void breath() { syso (breath) }
      }
class mammal extends Animal {
      int legs;
      }
   class dog extends mammal
         {
         string breed;
         }
```

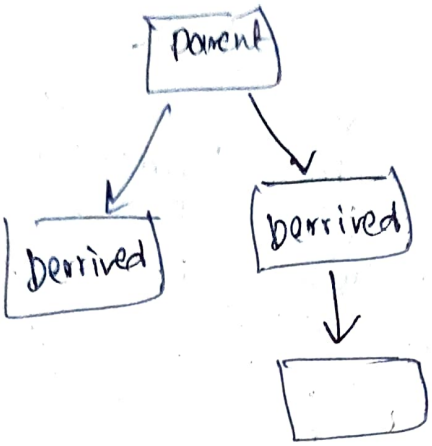
Hierarchial inheritance.

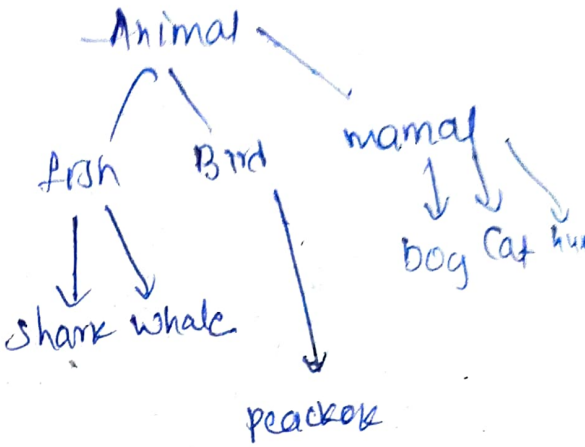





```
class Animal {
    void eats() {
    void breath() {
    }
class mamal extends Animal {
    void walk() {syso
    }
class bird extends animal {
    void fly() { syso
    }
class fish extends animal {
    void swim() {
        syso
    }

psvmsA {
fish f1 = new fish()
    f1.swim();
    f1.eats();
    f1.breath();
```

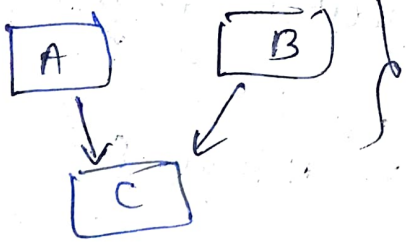
Hybrid Inheritance.





Multiple Inheritance :-

It is not present in Java. used in C++



using Interface we can form multiple

Polymorfism :- when we try to do/achieve same thing in multiple forms.

many forms.

Compile time polymorphism (static)

- method overloading

Run Time polymorphism (dynamic)

- method overriding.

Method overloading :-

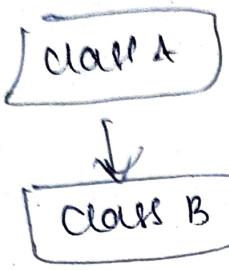
Multiple functions with same name but different parameters:

```
class calculator {

    sum (int a, int b)

    sum (float a, float b)

    sum (int a, int b, int c)
```

```
class calcular {
    int sum (int a, int b) {
        return a+b;
    }
    float sum (float a, float b) {
        return a+b;
    }
    int sum (int a, int b, int c)
    {
        return a+b+c;
    }
    psvm s A {
        calcular calc = new calcular();
        syso (calc.sum (1,2));
        syso (calc.sum (1,2,3));
        syso (calc.sum (1.5, 2.5, 3.2));
    }
```

Method overriding

Parent and child classes both contain same function with different definition.



≡ function

≡ function.

ex:- Animal
void eat (x) → eat everyth

↓

Deer
void eat (x) → eats Gras.

```
class Animal {
    void eat() {
        syso("eat anything")
    }
}
class beer extends Animal {
    void eat() {
        syso("eat grass");
    }
psvm {
    deer d1 = new deer();
    syso("d1.eat()");
```

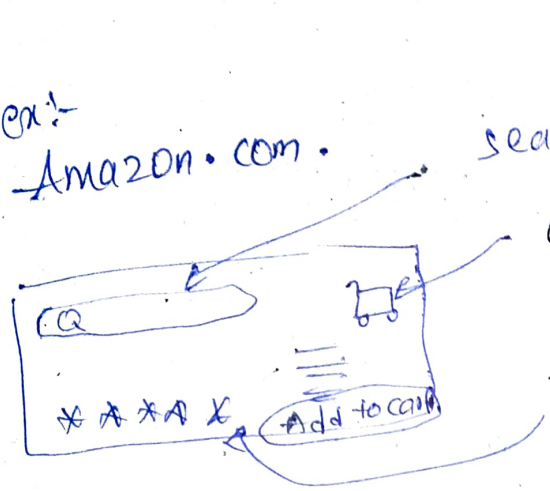
Output :- eat grass

packages in java :-

packages is a similar type of classes, interfaces and sub-packages.

ex:-

Amazon.com.



Google

Maps → one package

Chrome → one package of search

Gpay → one package of payments.

Packages :-

1 → Inbuilt packages → scanner sc = new scanner ( );
import java.util.*;

2 → Userdefined packages

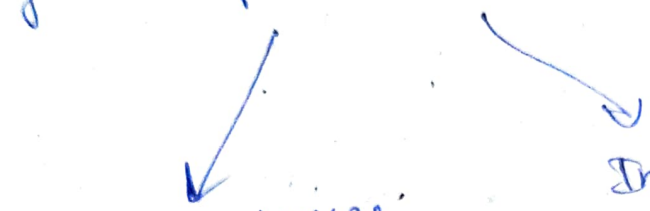
Abstraction :-

OOPs contain 4 pillars

1) Encapsulation
2) inheritance
3) polymorphism
4) Abstraction

Abstraction :-

hiding all the unnecessary details and showing only the important parts to user.



idea ✓
implementation ✗

Abstract class :-

keyword Abstract

ex:- Abstract class Animal {
}

→ cannot create an instance of abstract class object

→ can have abstract | non-abstract method

→ can have constructors.

```
Abstract class Animal {
    void eat() {
        sysO("animal eats");
    }
    abstract void walk();
}

class horse extends animal {
    void walk() {
        sysO("walks");
    }
}

class chicken extends animal {
    void walk() {
        sysO("walks on 2 legs");
    }
}

psvmSA()
{
    horse h = new horse();
    h.eat();
    h.walk();
    chicken c = new chicken();
    c.eat();
    c.walk();
```

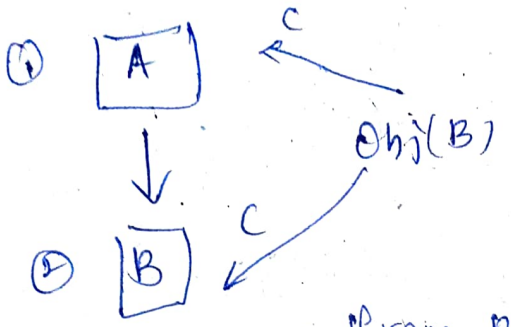
hided implementation

Animal() {
color = "brown";

string color;

void changeColor() {
color = "black"
}

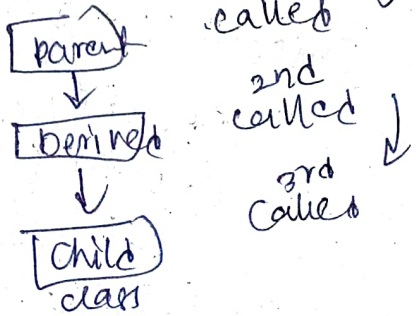
void change color
{
color = "yellow"
}



Constructor is called from parent class to child cla

```
class Animal {
Animal(){ // constructor
    syso("animal constructor called");
    }
}
horse extends animal {

    horse(){
    syso("horse constructor called");
    }

breed extends horse {

    breed(){
        syso("breed constructor called");
        }
```

Constructor heirarcy is



Interface :- Interface is a blueprint of a class.

class is a blueprint of object.

Car [wheels, speed, engine] (Interface)

↓

Maruthi 800 (class) (class)

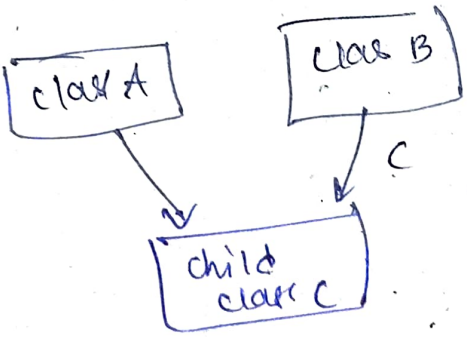
↓ ↓ ↓ ↓

Maruthi 800 ford RR (object)

Interface is used to work in 2 ways.

1) multiple inheritance (5th inheritance)
2) total Abstraction (100% Abstraction)

multiple inheritance :-



Animal

eats → implementation

walks → idea

Interface :-

→ All methods are public, abstract & without implementation

→ used to achieve total abstraction.

→ variables in the interface are final, public and static

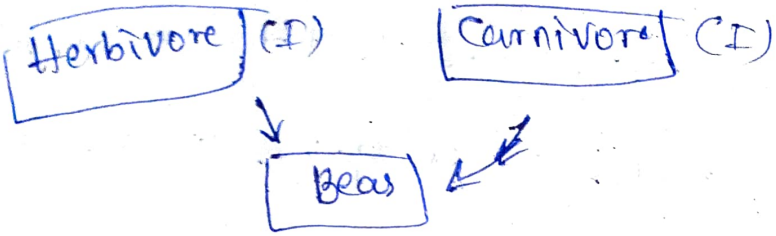
→

Interface	class
Implements	extends

ex:- chess player

code :-

```
interface chessplayer {
    void moves();
}
class queen implements chessplayes {
public void moves() {
        syso("up, down, left, right, diag");
    }
class pawn implements chessplayes {
    public void moves() {
        syso("up");

public s v m() {
    queen q= new queen();
    q.moves();
```



ex:-

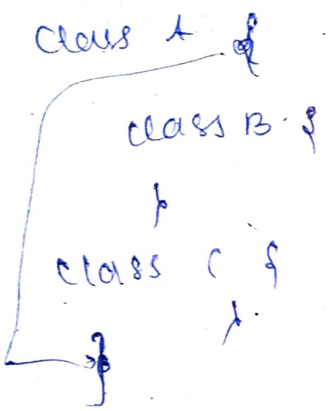
```
interface Herbivore {
}
Interface carnivore {
}
class beer implements Herbivore, Carnivore
{
function
}
Interface Herbivore {
void eatPlant();
}
Interface carnivore {
void eatflesh();
}
class beer implements Herbivore, Carnivore {
public void eatplant() {
sysO ("carnivore eats flesh");
}
public void eatflesh() {
syso("herbivore eats plant");
}
}
PSVM(S[]) {
Beer b1 = new beer();
b1.eatflesh();
b1.eatplant();
```

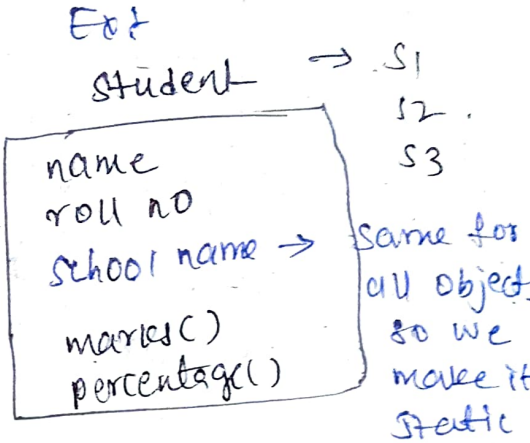
Static Keyword:

Static keyword in Java is used to share the same variable or method of a given class.

- properties
- Functions
- Blocks
- Nested classed

} we can make static



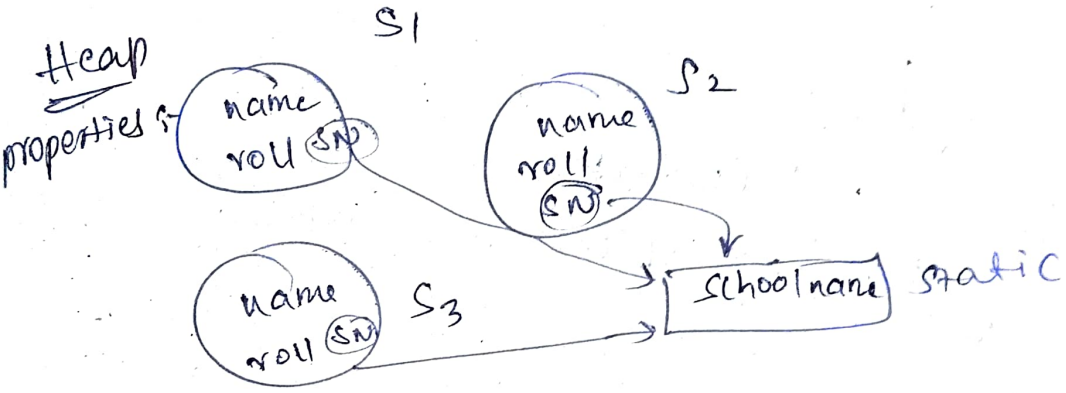


Code

```
class student {
    string name;
    int roll;

  static string schoolname;

    void getName(string name) {
        this.name = name; }

    String getname()
        {
        return this.name;
        }
```

psvms {
student s_1 = new student();
s_1.schoolName = "JMV";
student s_2 = new student();
syso(s_2.schoolname);

student s3 = new student();
s3.schoolname = "ABC";



Functions :-

public static main function()

class student {
static int returnpercent(int m, int p, int chem) {
return (m+p+c)/3;
}

Blocks
Nested classes
} Not important for interview.

Super keyword :

Super keyword is used to refer immediate parent class object

this keyword
↓
current obj
↓
this.value
or
this.function

→ to access parent's properties
→ to access parent's functions
→ to access parent's constructor.

```
class Animal {
    String color;
    Animal() { Constructor
        syso("Animal contuctor is called");
    }
}

class horse extends Animal {
    Horse() {
        super(); // super.color = "brown";
        syso("Horse is called");
    }
}

Public static v m s1 {
    horse h1 = new horse();
    }
    syso(h.color);
}
```

constructor chaining :- in java.

practice questions:

1) find out the correct statement to assign name to object

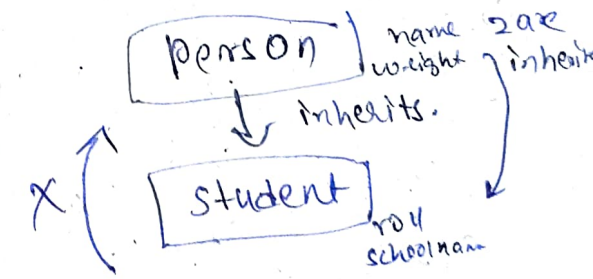
```
class student {
    string name;
    int marks
}
public class Oops
p s v m s A [ ]
    student s = new Student (1);
    // s.name = "aman";
    sysо (s.name);
```

a) s → name = "aman";
b) Student.name = "aman"
(c) s.name = 'aman'

option b = student-name
= "aman"
can be correct when we write static

2) which variable can the class person access in following code

```
class person {
    string name;
    int weight;
}
class student extends person
{
    int roll number;
    string school Name;
}
```



.name & weight are inherited to student but roll, school name will not inherit to person class.

(a) name (b) weight
c) student roll no d) school name

which of the following modifiers are not allowed in front of class.

a) private c) public
b) protected d) default

class A {

public
default
private
protected

}

By default & public are used infront of class.
But if we write private infront of class. then the properties that are defined inside private class cannot be accessed by the objects in main class

→ There is no protected type in java

Java

	privates	default	protected	public
class	No	yes	No	yes
Nested class	yes	yes	yes	yes
constructor	yes	yes	yes	yes
method	yes	yes	yes	yes
field	yes	yes	yes	yes

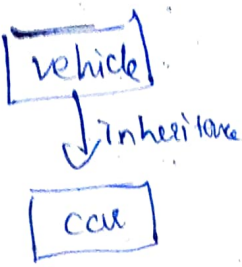
Access specifiers that are used & not used before diffent classes / methods..etc

Ex:- public class {

}

which of the following is a correct statement (both classes in same package)

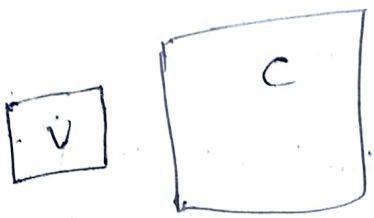
```
class vehicle { }
class car extends vehicle { }
```

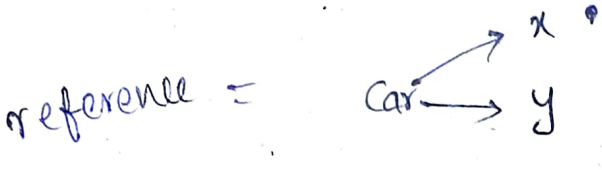


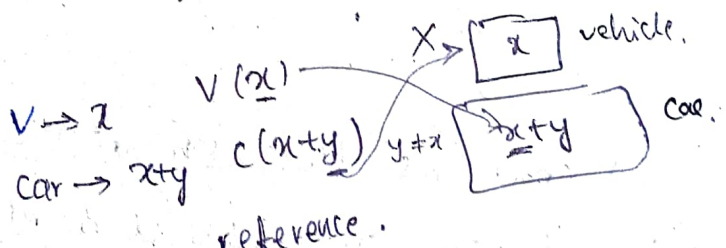
(a) car c = new car ();
(b) vehicle v = new vehicle ();
(c) vehicle v = new car ();
d) car c = new vehicle ();

By default, child objects are assigned to the reference of parent

reference is used to store the address of object







Question 5:

what will be the output of this code (both classes in same package)

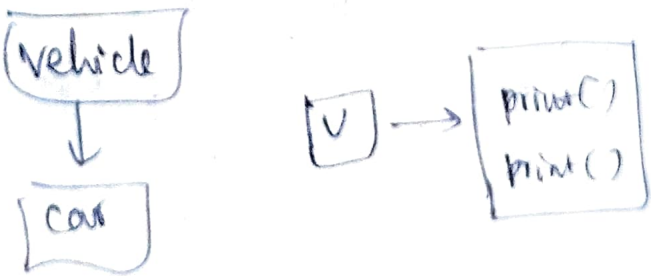
```
public class OOPS {
public static void main(String[] args) {
vehicle obj1 = new car();
  obj1.print();
```

→ function overriding (child class print)

```
vehicle obj2 = new vehicle();
    obj2.print();
```

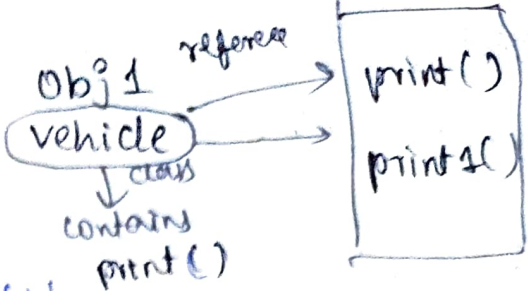
→ (base class (vehicle))

```
}
}

class vehicle {
  void print() {
      sysO("Base class (vehicle)");
  }

class car extends vehicle {
      void print() {
        sysO("child class (car)");
        }
```



child class (car) contains two print()'s but function overrides and prints child class as output

practice Question 6:-

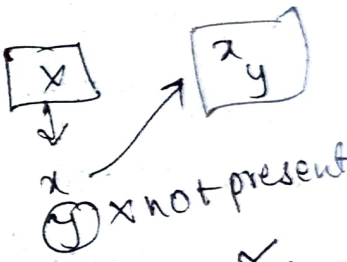



```
public class Oops {
p s v m s A ( ) {
vehicle obj1 = new car();
   obj1.print1();
                  → error
vehicle obj2 = new vehicle();
   obj2.print();
   }                // Base class.
}
```

obj1.print1();

obj1 → only has print() but doesn't have print1()

```
class vehicle {
   void print() {
      sysO("Base class");
   }
class car extends vehicle {
   void print1()
      {
      sysO("derived class");
      }
```



practice Qn-7:-

which of the following is not an OOPs component

major components of OOPs
- Inheritance
- encapsulation
- polymorphism
- abstraction.

1) Inheritance
2) encapsulation
3) polymorphism
4) aggregation/compilation

Question 8

what will be the output of this code.

```
class book {
    int price;
Static int count; // by default count = 0.

public book (int price) {
    this.price = price;
    count++;
    }
}
p static oops {
public static void main () {
sysO (book.count); // 0
book b1 = new book (150);
book b2 = new book (250);
 sysO ( book.count); // 2
```

(if properties are static then we can access from class also
ex:- (book.count) ✓

// we can directly write class name and properties if it is static only
(book.count) // 0

→ 0 ✗ 1

a) error (b) 0, 2

c) 1, 2 d) 2, 2

practice qn 9

which line has error

```
class test {
    Static int marks; → // 0
    void set_marks (int marks) {
        this.marks = marks; → Line 1
    }
```

```
public class oops {
    public static void main (s+) {
        test t = new test ();
        t.set_marks (98);        → Line 2
        sysO (test.marks)        → Line 3.
```

Line 1 → static int marks ; (No error)

Line 2 → t.set marks (98) ; no error)

we can set marks

Line 3 → test.marks; (no error)

Why because in test class marks is static property. so no error.

Practice question 10

What would be the output of following code.

```
class test {
    static int a = 10
    static int b ; // b = 0
    static void changeB () {
        b = a * 3;
    }
}
class oops {
    psvm SA {
        test t = new test ();
        t.changeB ();
        Syso (test.a + test.b);
    }
}
```

a = 10
default b = 0.

t.changeB.
b = 10 x 3 = 30

// 10 + 30 = 40

output = 40