5/7/19

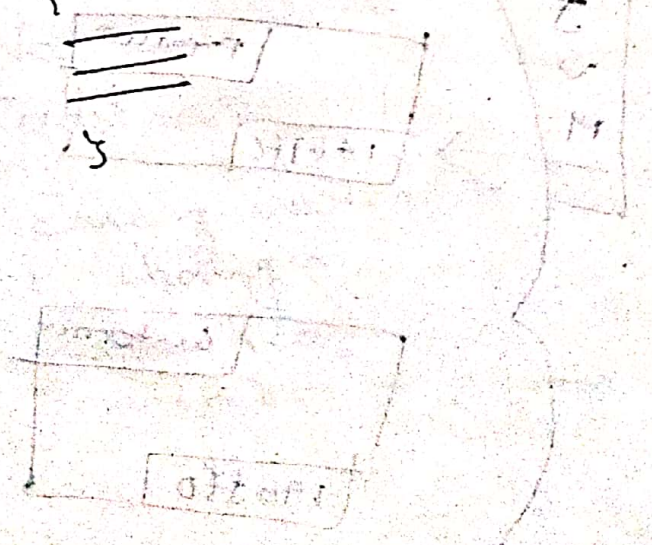
After Succusful, Compilation, the Compiler guerates default Constructor.

the Constructor name must be Same as class name

ex: class Movie

```
{
  Movie()
  {
    ===
    ===
    ===
  }
}
```

```
class product
{
  product()
  {
    ===
    ===
    ===
  }
}
```



Scanned by CamScanner

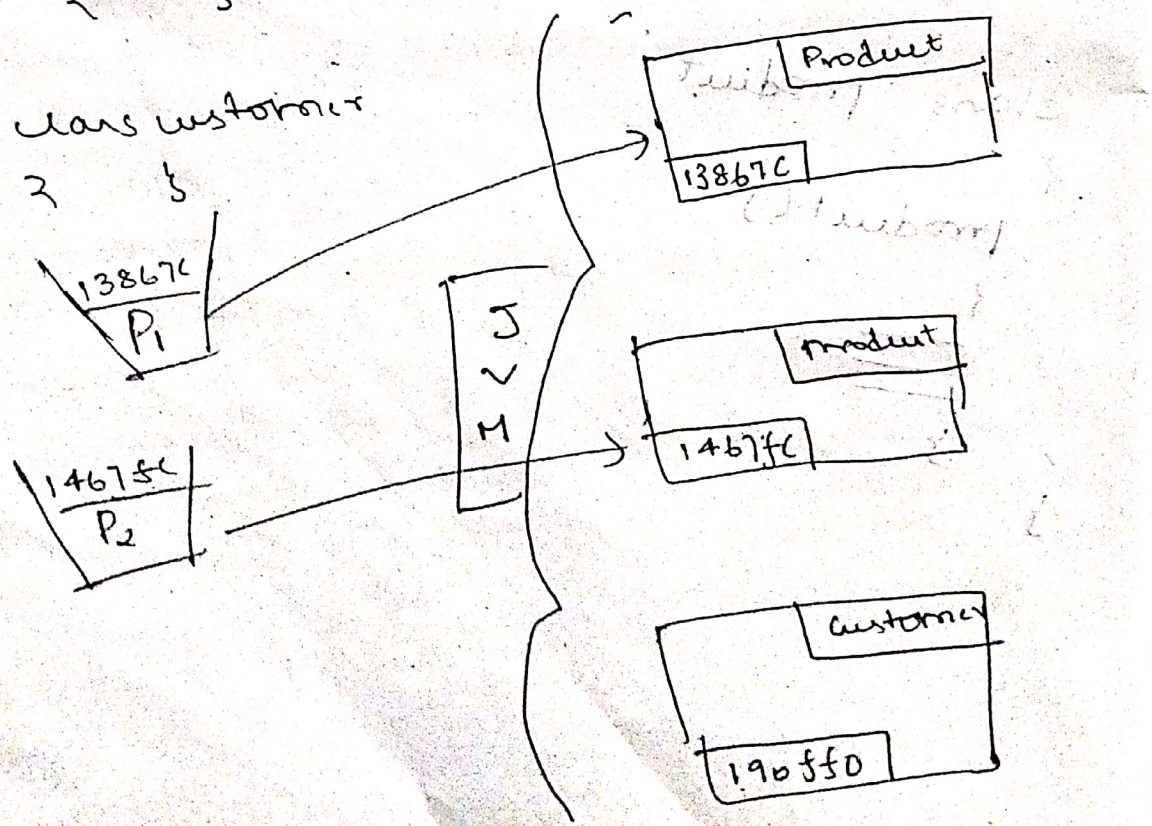
In Java, objects are stored in heap memory

- Memory allocation & memory deallocation, i.e, ~~no~~ Complete memory management is taken care by JVM

JVM creates an object in heap memory when we call the constructor using ~~new~~ new keyword.

Object address is represented in hexadecimal format.

```
class Account
{ }

class product
{ }

class customer
{ }

new product();

new product();

new customer
```



Scanned by CamScanner

Object Reference / Reference Variable

A Reference is the one which stores an object address, Hence it refers object present in heap memory.

ex:

```
product p1 = new product();
product p2 = new product();
Customer c1 = new Customer();
```

State :-

State of an object is nothing but The property / information which describes an object.

The data member is nothing but a data holder which holds / stores The data.

There are 2 type of data members

* Variable
* Constant

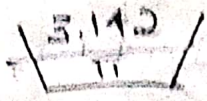
In case of variable The data Varies i.e., changes

Programatical non final data member is called variable

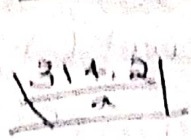
Scanned by CamScanner

Constant is the data member which represents the fixed value, programmatically we declare constant using final keyword.

r = 10



11 = 3.142



a = 11r²

= 3.142 × 10²

a = 314.2

final Name = "SRK"

Since an object is derived from a class, the states and behaviours of an object must be first declared in class, Hence states and behaviours are the constituents of a class.

ex: class pen

{

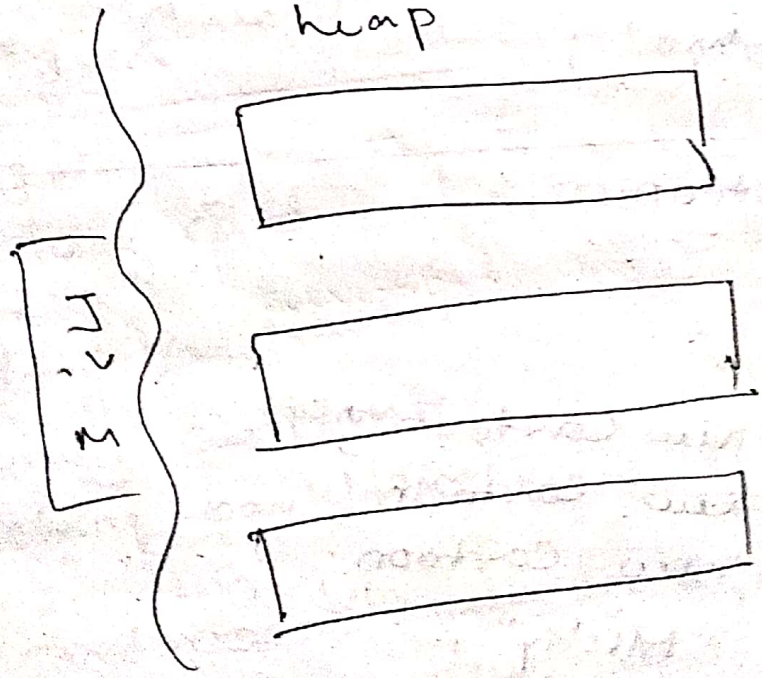
String color = "green";

String type = "marker";

}

Pen P = new pen();

```
class Pen
{
string color;
}
Pen P1 = new Pen();
Pen P2 = new Pen();
Pen P3 = new Pen();
P1.color = "green";
P2.color = "blue";
P3.color = "black";
```



Scanned by CamScanner

```
clars car
{ ~~model~~ ; String model;
  ~~clr~~ ; String clr;
}
car c1 = new car();
car c2 = new car();
c1.model = "beatle";
c1.clr = "yellow";
c2.model = "aviator";
c2.clr = "blue";
```

```
clars Cartoon
{
String name;
}
Cartoon c1 = new Cartoon;
cartoon c2 = new cartoon;
Cartoon c3 = new cartoon;
c1.name = "Micky";
c2.name = "Jerry";
c3.name = "Bheem";
```

Scanned by CamScanner

data-type name = data;

---

final string DOB = "22-NOV";
final string NAME = "Beant";
final string PINCODE = "560076";

---

int monthlySalary = 26000;
string firstName = "Abhishek";
String SmartCityOfIndia = "Mysore";
boolean areYouMarried = false;
long ContactNumber = 9886732160

Prg:-

```
class Card
{
long CardNumber;
String custName;
String bankName;
}
Card C1 = new Card;
Card C2 = new Card;
C1.CardNumber = 4326 0123 4567 8903;
C1.custName = "VijayKumar";
C1 = bankName = "SBI";
C2.CardNumber = 4375 5174 1234 5678;
C2.custName = "RohitSharma";
C2.bankName = "ICIC";
```

Instantiation

Object initialization

Scanned by CamScanner

There 2 catagories of datatype :-

1) primitive
2) Non-primitive/user defined/custom

Primitive datatypes are not the classes, rather they are the keywords

Non primitive data type is nothing but class. In java, it is possible to define our own data type in the form of class. which is called as non primitive datatype

String is a non primitive data type because its a class

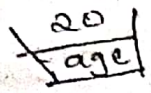
byte
short
int
long
float
double
boolean
char

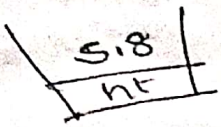
} primitive datatype

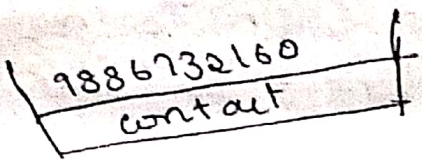
Primitive variable cannot store object address. Only a non primitive variable can store object address.

Scanned by CamScanner

int age = 21;
double ht = 5.8;
long contact = 9886732160L;

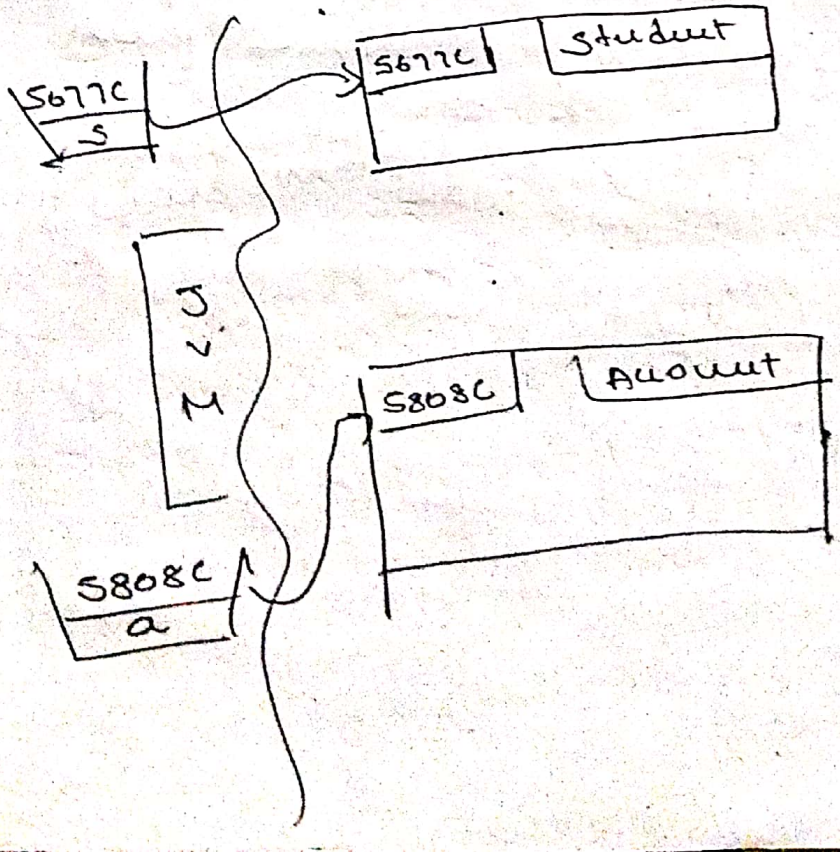






class student
{ }
Student s = new student();

class Account
{ }
Account a = new Account();

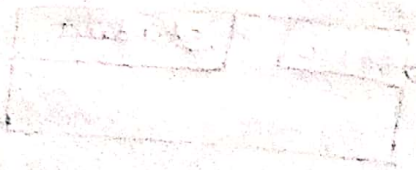


Scanned by CamScanner

Instance Method

Instance Method की Method represents behaviour of an object

```
class Horse
{
    void run()
    {
    }
}

class Tap {
    water open()
    {
    }
}

class water
{
}
```



Scanned by CamScanner

```
class Conductor
{
Ticket Issue()
 {
 }
}
class Ticket
 {
 }
```

```
class shop
{
product sell
 {
 }
}
class product
 {
 }
```

```
class Atm
{
money dispense()
 {
 }
}
class money
{
}
```

Scanned by CamScanner

Method rules

* Syntax:

```
returnType name(params)
{
logic | body | defn | implementation
}
```

* Method must have either the Return type or void
* void method cannot Return any data
* a method can have only one Return Type & only one Return Statement

Return type means the data type of the Returning value

Return type and the Returing data must match

Inside a method, Return Statement must be the last executable Statement

examples

```
int meth()
{
  return 40;
}
```

```
void m2()
{
return 0;
}
```

X

Scanned by CamScanner

```
String meth()
{
String s = "Beast";
return s;
}
```

```
void m1()
{

}
```

```
double m1()
{
return 10;
}
```

```
int ma()
{
return 30.2;
}
```
X

```
void m()
{
return;
}
```

```
String m1()
{
return "50";
}
```

```
boolean m()
{
return true;
return false;
}
```
X

```
Void m()
{
return;
}
```

```
int m()
{
return 87;
S.O.p("hi");
}
```

```
class Ticket
{

}
```

```
class Conductor
{
Ticket issue()
{
Ticket t = new Ticket();
return t;
}
}
```

t is called local reference variable.

Scanned by CamScanner

```
class product
{
}
```

```
class shop
{
product sell()
{
product p = new product();
return p;
}
```

```
class money
{
}
```

```
class ATM {
money dispense()
{
money m = new money();
return m;
}
```

---

* A method gets executed only when we invoke it

* A method can be invoked multiple times.

* main method is not required for compilation rather required for execution

ex:
```
class pen
{
int price = 40;
String clr = "black";
void write()
{
System.out.println("pen writes");
}
```

Scanned by CamScanner

```
Public static void main (String[] args)
 {
System.out.println(" main start!!");
   Pen p = new pen();
   P.write();
   P.write();
   System.out.println("main end.");
   }
}
```

o/p:

main start
pen writes
pen writes
main end

```
class Dancer {
public static void main(String[] args)
 {
Dancer d = new Dancer();
  d.dance();
 }
void dance()
 {
S.o.p("dancer dance");
  String name = " Michel";
   }
}
```

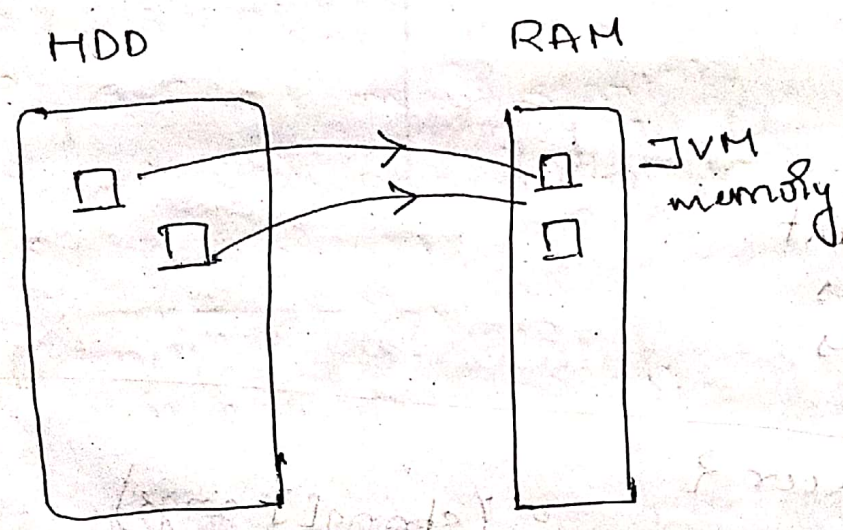
Scanned by CamScanner

Class loading

is the process of loading the .class file (byte code) from Harddisk memory to the JVM memory

A class gets loaded only once.

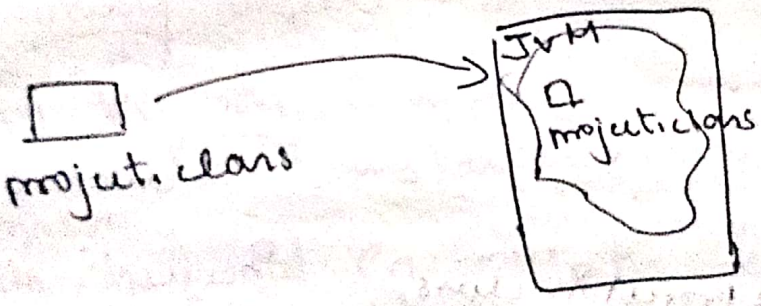
JVM uses class loader to load a .class file



Scanned by CamScanner

```
1  class Projector {
2      void display() {
3          S.o.p("projector display");
4      }
5  public static void main(String[] args) {
6  S.o.P("main start");
7  Projector p = new Projector();
8   p.display();
9   S.o.P("main end");
10  }
11 }
```

c:\.....\Desktop\programs



1. Compilation
2. class loading
3. Execution

5 (main)
6 → print msg
7
8 ⟶ 2 } method
3 } execution
4
⟵
9 print msg
10
⟶ Terminate

Scanned by CamScanner

```
1  class Projector {
2      void display() {
3          S.O.P("projector display");
4      }
5  }
```

```
1  class Test {
2  P.S.V.M (String[] args) {
3  S.O.P("main start");
4  Projector p = new Projector();
5  P.display();
6  P.display();
7  S.O.P("main end");
8  }
9  }
```

---

2 (main)
3 → print
4 → i) load Projector.class
ii) creat an obj of projector

5 display → Projector
2
3 } method execution
4

6 ← display → Projector
2
3 } method execution
4

7 ← print

8 → Termination

Scanned by CamScanner

```
class Test {
  P.S.V.M(String[] args) {
    S.O.P(5+5);
    S.O.P("Narendra" + "Modi");
    S.O.P(5+2+"hello");
    S.O.P(5+2+"hello"+3+1);
```

O/P

10
NarendraModi
7hello
7hello31

---

this Keyword

this is a keyword which refers to current invoking object.

- this keyword is used to access instance variables & instance method.
- this keyword cannot be used within ~~this~~ the static context i.e, this keyword cannot be used within static method or static block.

Scanned by CamScanner

① class Animal {
void makeNoise() {
S.O.P(this);
}
P.S.V.M(String[] args) {
Animal a1 = new Animal();
S.O.P(a1);
a1.makeNoise();
}
}

o/p
Animal@7852e92
Animal@7852e92

---

class Product
{
int price=250;
void printProdDetails()
{
S.O.P("Product price=" + this.price+
"Rs");
}
}

```
Public class person {
  String name = "Beast";
  void eat() {
    this. washHands();
    this. serveFood();
    S.O.P("eat food");
    this. washHands();
    }
  void washHands() {
    S.O.P("wash hands");
    }
  void serveFood() {
    S.O.P("Serve food");
    }
}
```

Variable :

a variable is a data holder which stores the data

There are 2 types of variable

i) Global → (Static) class var
→ (non static) instance var

ii) local variable.

Scanned by CamScanner

Global variable :

A Global variable is a variable which is declared directly within class, Out side method or constructor.

If a global variable is static Then it is called class variable

If a global variable is non static Then it is called instance variable.

```
class Student {
    String name;
    double perc = 87.64;   } instance variable

    static String institute = "Jspiders";

    void study() {
        double noh = 5.5; // local var
    }
}
```

Instance variable :

A non static global variable is called instance variable.

Instance variable is created in memory only when an object gets created

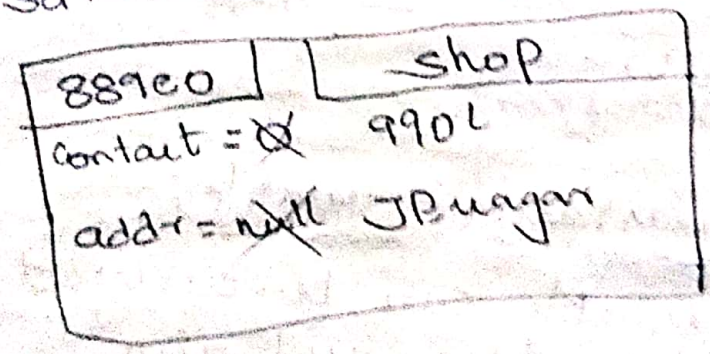
number of copies of each instance variable depends on number of objects

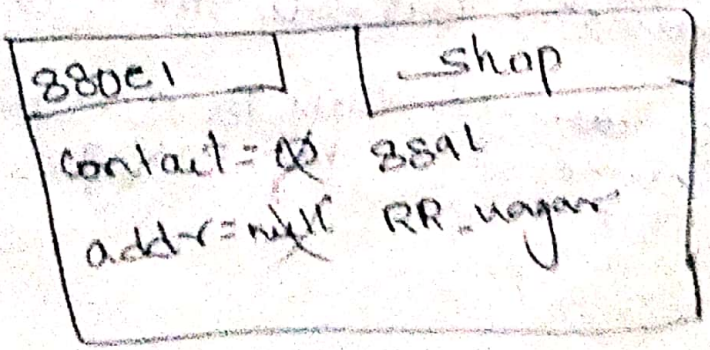
Scanned by CamScanner

Instance variable is stored inside an object as part of heap memory.

If an instance variable is not initialized at the time of declaration then it is initialized to a default value at time of object creation.

```
class shop
{
 long contact;
 string addr;
 }
Shop s1 = new shop();
 s1.contact = 9901;
 s1.addr = "JPnagar";
Shop s2 = new shop();
 s2.contact = 8891;
 s2.addr = "RRnagar";
```

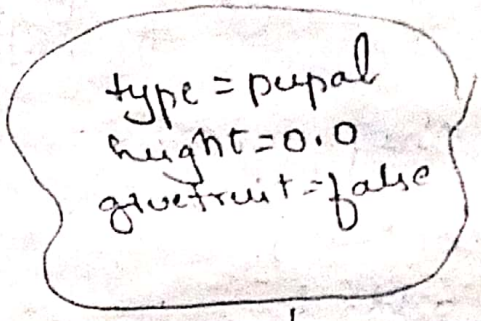




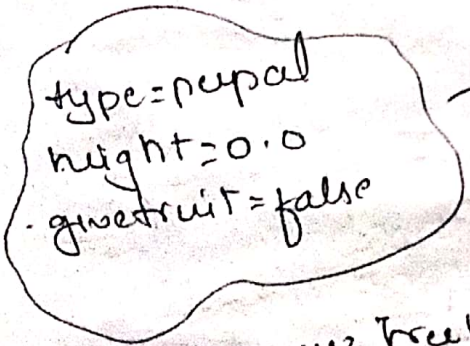
Scanned by CamScanner

```
class Tree2
String type = "peepal";
double height;
boolean giveFruit = false;
```

Tree t1 = new Tree();



instance/object



Tree t2 = new Tree();

Scope of an instance variable :-

- An instance variable can be accessed throughout the class and also can be accessed outside the class
- An Instance variable can be accessed within same class by using keyword this

Instance variable can be accessed outside class using object reference.

```
class Car
{
  string model = "alto";
      void ~~static~~ start().
       {
       s.o.p(this.model);
        }
     void move()
      {
      s.o.p(this.model);
      }
  }

class Driver
{
   void drive()
    {
    Car c = new car();
    s.o.p(c.model);
    }
}
```

Scanned by CamScanner

local variable:

is a variable which is created within a method or constructor or block

local variable must be initialized before the use. i.e., Default initialization is not applicable for local variable.

The Scope of local variable is limited.

local variable cannot be accessed using Object Reference or by using this keyword.

~~Loops~~:

Variable Shadowing:

In java local variable and an instance variable can have same name, And in this case, inside local scope, the local variable dominates over an instance variable, and this concept is called variable Shadowing

```
class Dog {
  double ht = 2.8;
   void jump()
    {
    double ht = 5.2;
     S.O.P(ht); //5.2
     S.O.P (this.ht); //2.8
      }
}
```

Scanned by CamScanner

Different ways of initializing an Instance variable

An Instance variable can be Initialized in many ways.

1) at time of declaration.
2) using an object reference.
3) using instance block.
4) Using Constructor
5) Using Setter method

1)
```
class Team
{
String name = "RCB";
}
```

2)
```
class Team
{
String name;
}
Team t = new Team();
t.nm = "RCB";
```

Instance block:-
is a block which automatically gets executed by JVM while creating an object

We ~~have~~ can have any number of instance block in a class.

The number of time's an instance block executed depends on number of object.

3) class Player
{
{
S.O.P("hello");
}
}
Player P1 = new player();
new player();
player P2 = new Player();

O/P
hello
hello
hello

3) class Player {
String Pnm;
{
This.Pnm = "Dhoni";
}
}
Player P1 = new Player(); S.O.P(P1.Pnm);
S.O.P(new Player().Pnm);
Player P2 = new Player();
S.O.P(P2.Pnm);

Scanned by CamScanner

Constructor:

Constructor is one of the member of a class like method and variable.

Constructor name must be same as class name. Constructor cannot have any return type & not even void.

There are 2 types of constructors.

* Default Constructor
* custom / user defined Constructor.

The whole purpose of Constructor is to initialize instance variables. at the time of object creation

Default Constructor

Developer View

```
class Mobile {
    int price;
    String color;

Public Static void main (String[] args)
{
  Mobile m = new mobile();
  m.price = 15000;
  S.o.P(m.price);
}
}
```

Scanned by CamScanner

Compiler view

```
class Mobile
{
int price;
String clr;
    Mobile()
    {
    }
p s v m(String[] args)
    {
    Mobile m = new Mobile();

    }
}
```

Default constructor is type of constructor which is created by the compiler

Default constructor will always be non parametrized.

Default constructor is created only if there is no custom constructor.

Default constructor is used of created in order to assign default values to the states present in a class.

Scanned by CamScanner

~~Custom Constructor~~

```
Class Box
{
int length = 2;
int breadth = 3;

P. S. V. M (String[] args)
  {
    Box b1 = new Box();
    Box b2 = new Box();
    S.O.P(b1.length);
    S.O.P(b2.length);
    S.O.P(b1.breadth);
    S.O.P(b2.breadth);
    }

}
```

Custom Constructor

Any Constructor which is created by the user or by the developer is called as Custom Constructor

Custom Constructor name must be same as that of the class name

Custom constructor can be parameterized or non parameterized.

In a class there can be either custom constructor or default constructor but not both.

Scanned by CamScanner

Custom Constructor is needed in order to assign dynamic values of user required values to the states that are present in the object.

```
class Box
{
  int length;
  int breadth;
  Box(int x, int y)
  {
  this.length = x;
  this.Breadth = y;
  S.O.P(this.length + " " + this.breadth);
  }
  P.S.V.M(String[] args)
  {
  Box b1 = new Box(2,3);
  Box b2 = new Box(4,5);
  Box b3 = new Box(5,9);
  }
}
```

Scanned by CamScanner

Parameter

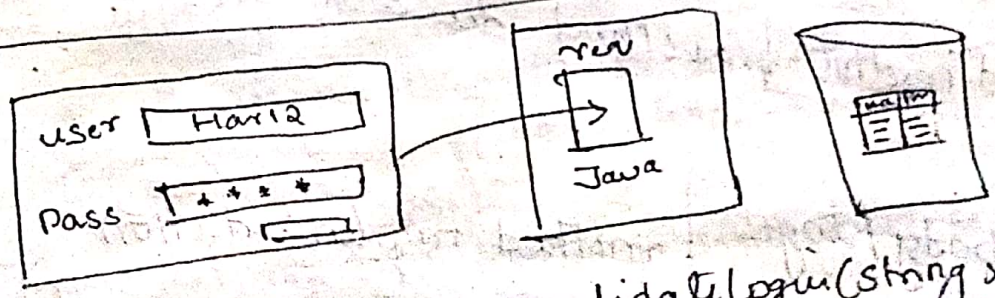
```
Class Employee
{
  int id;
  void work(String talk)
  {
  int noh=9;
  }
Employee emp=new Employee();
  emp.work("Administrator");
```

work (talk)

emp.work("Administrator")

↓

method initialization




```
boolean validateLogin(String x, String y)
{
}
```

Scanned by CamScanner

There are 2 categories of method

1) abstract method
2) Concrete method.

An abstract method is a method which has only method declaration but no method implementation.

Abstract method must be terminated with semicolon.

Abstract method must be declared by using keyword abstract

Abstract method cannot have body.

A Concrete method has both declaration as well as definition

```
Void meth();
```
method declaration

```
{
    body/logic
}
```
method implementation

abstract void doubleMeth();

Void method (Signature)
{
}

Signature:
- No of parameters
- type of parameters
- sequence of parameters.

Method Overloading :

In a class when we have more than one method with the same name but change in Signature is called as method Overloading.

change in Signature means;

* either there has to be change in no of parameters
* (or) there has to be change in the type of parameters
* (or) there has to be change in the sequence of parameters.

In method overloading we dont consider method Return type.

Advantage or purpose of method Overloading is to achieve Compile time polymorphism

method overloading is possible in same class & not possible in case of inheritance.

Scanned by CamScanner

ex: class Website

```
{
    void login(String un, String Pwd)
    {
    |.....
    }
    void login(long mob, int otp)
    {
    |...
    }
}
```

```
Public class Airtel
{
//net banking
Public void makePayment(String un,
  String Pwd)
  {.....}

 // Paytm
Public void makePayment(long mobno)
  {.....}

 // Credit/debit card

Public void makePayment(long cardno,
  int cvv, String name, String expdate)
  {.....}
}
```

Scanned by CamScanner

In a class having more than one method with same name and with same signature is called method duplication.
So compilation error.

Ex: public class Test
{
public int meth()
{
return 20;
}
public String meth()
{
return "hello";
}
}
Test t=new Test();
t.meth(); → Ambiguity / Conflict.

X
method duplication

---

class Quiz
{
void meth(String s, double d)
{...}
void meth(String d, double s)
{...}
}

X
method duplication

Scanned by CamScanner

```
Public class Test
{
    Public int meth()
    {
    return 20;
    }
    public void meth(int i)
    {
    }
}
```

```
Public class Test
{
  private void meth(String ss)
  { ... }
  Public void meth(long mobno)
  { ... }
}
```

```
class Quiz
{
  Void meth(String s)
  { }
  void meth(Product p)
  { }
}
Quiz q = new Quiz();
q.meth(null); X error.
```

Scanned by CamScanner

Constructor Overloading

Having more the one Constructor in a class with change in Signature is called Constructor Overloading.

ex:

```
Public class AccountDetail {
    long acctNumber;
    double balance;
    String name;

Public AccountDetail (long acctNumber,
double balance, String name)
{
this.acctNumber = acctNumber;
this.balance = balance;
this.name = name;
}
Public AccountDetail()
{
}
Public AccountDetail (long actnumb)
{
this.acctNumber = actnumb;
}
Public static void main (String[] args)
{
AccountDetail acl = new AccountDetail();
S.O.P(acl.acctNumber + "  " + acl.name +
"  " + acl.balance);
```

Scanned by CamScanner

```
AccountDetail ac2 = new AccountDetail(
    4578901238L);
    System.out.println(ac2.acctNumber + "  " +
    ac2.name + "  " + ac2.balance.
    }
}
```

Inheritance

Inheritance is the process of acquiring the properties & members (states and behaviours) of one class into other class

Programatic we can achieve inheritance by using keyword **extends**.

Inheritance is also called as is a relationship

Constructor cannot be inherited.

rather only variable and method get inherited.

Uses:

- code reusability
- we can avoid code redundancy & duplication
- we can achieve generalization

Scanned by CamScanner

Inheritance indirectly achieve polymorphism

ex:

```
class card {
    long cardNumber;
    int cvv;
    String name, expDate;
    double balance;

    void swipe() {
    S.O.P ("Swipe the card");
    }

}
class creditCard extends card {
    int creditlimit;

    void payBill() {
    S.O.P ("pay creditcard");
    }
}

class DebitCard extends card {
    int balance;
}
```

The class which has common states and behaviours are called parent class / super / base class

Scanned by CamScanner

The class that inherits the super class is called Sub/child/derived class

Using subclass reference we can access all the inherited variables and methods and also subclass specific variables & methods.

```
Public class Test {
    P.S.V.M. (String args[]) {
    CreditCard CC = new CreditCard();

    cc.balance=75000;
    cc.cardNumber=1234567890L;
    cc.cvv=148;
    cc.expDate="12/22";
    cc.name="Chaitra";
cc.creditLimit=100000;

    cc.swipe();
    cc.payBill;
    }
}
```

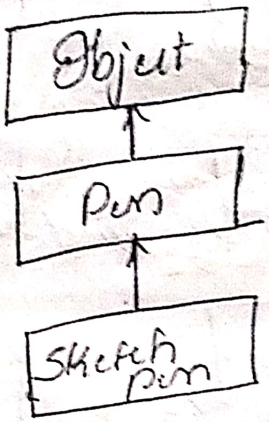
Scanned by CamScanner

Note: Every class in Java automatically extends the Supermost class called Object

Object class has '0' superclass
Object class has 11 concrete method.

```
class Pen
{
}

class SketchPen extends Pen
{
}
```

```
class Pen extends Object
{
}
```



Types of inheritance

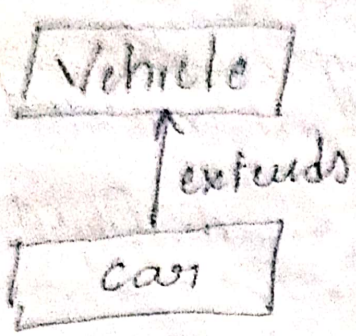
Single level
multi level
Hierarchical level

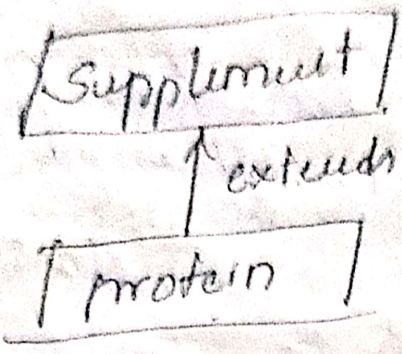
Multiple inheritance
hybrid inheritance
X Not possible using class

can be possible using interface.

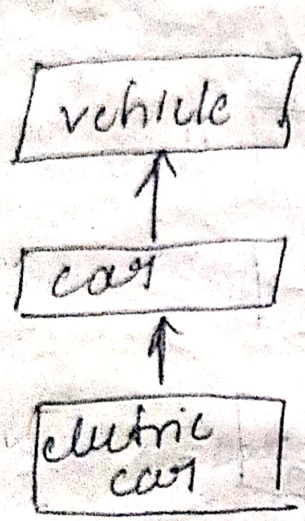
Scanned by CamScanner

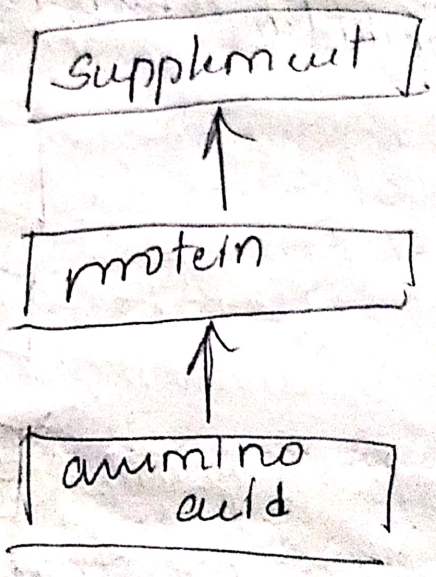
Single level



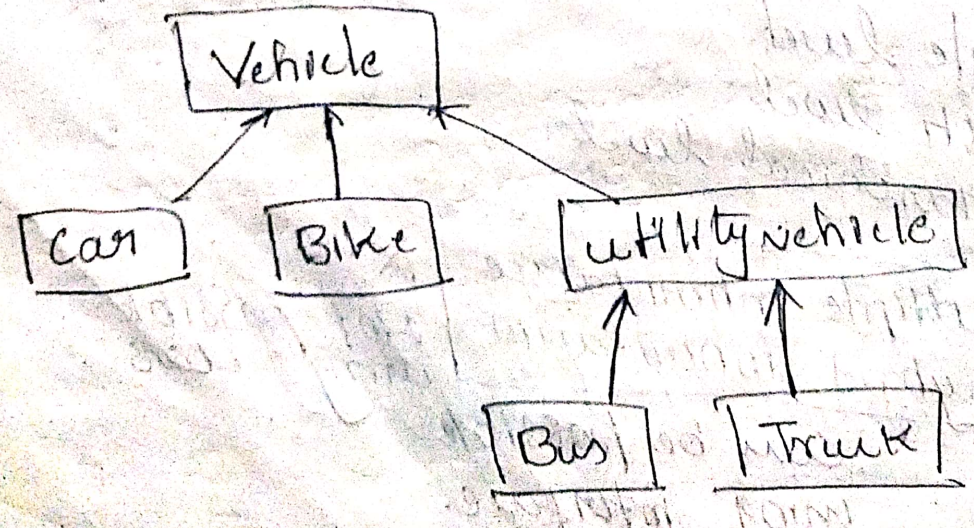


multilevel





Hierarchical inheritance



Scanned by CamScanner

Final class cannot have a Subclass
i.e., final class cannot be extended.

Real time examples of final class

String
Scanner
System
wrapper class
Integer
Double
Long.

final variable & final method can be inherited but private method and private variable cannot be inherited.

```
class Father {
    long money = 10000000L;
    private String girlfriend = "Katrina";
    void doyoga() {
    S.o.p("yoga");
    }
    private void Smoke() {
      S.o.p("Smoke");
      }
}
class Son extends Father {
}
```

Scanned by CamScanner

Method Overriding:-

is the process of providing the Subclass specific method implementation for an inherited method.

when a method from super is inherited to Subclass, in Subclass we can change the method logic by keeping the same method declaration,

or

method Overriding is possible only in case of inheritance

Rules:

The return type of method must be same as declared in Superclass

Method name must be same as declared in Superclass

Signature must be same as declared in superclass.

logic can be different.

Note: we can optionally use an annotation @Override.

Scanned by CamScanner

ex: class P {
void meth()
{
——— Parent. logic
}
}
class C extends P {
@Override
void meth()
{
——— child. logic
}
}
C ref = new C();
ref. meth();

class Engineer {
void work()
{
S.O.P(" Engineer work");
}
}
class civilEngineer extends Engineer {
void work() {
S.O.P(" civilEngineer works");
}
class SoftwareEngineer extends Engineer {
void work() {
S.O.P(" develops SF");
}

Scanned by CamScanner

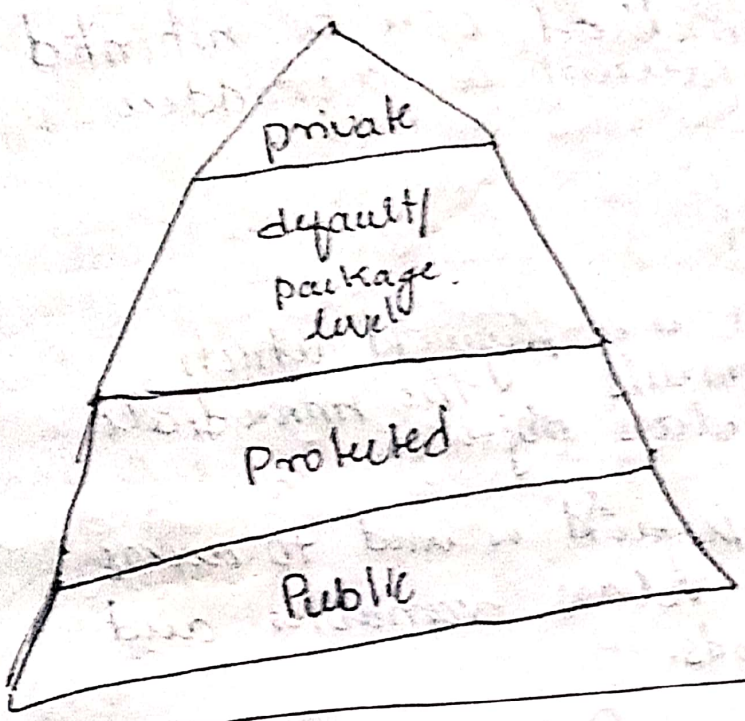
// over-loading in case of inheritance

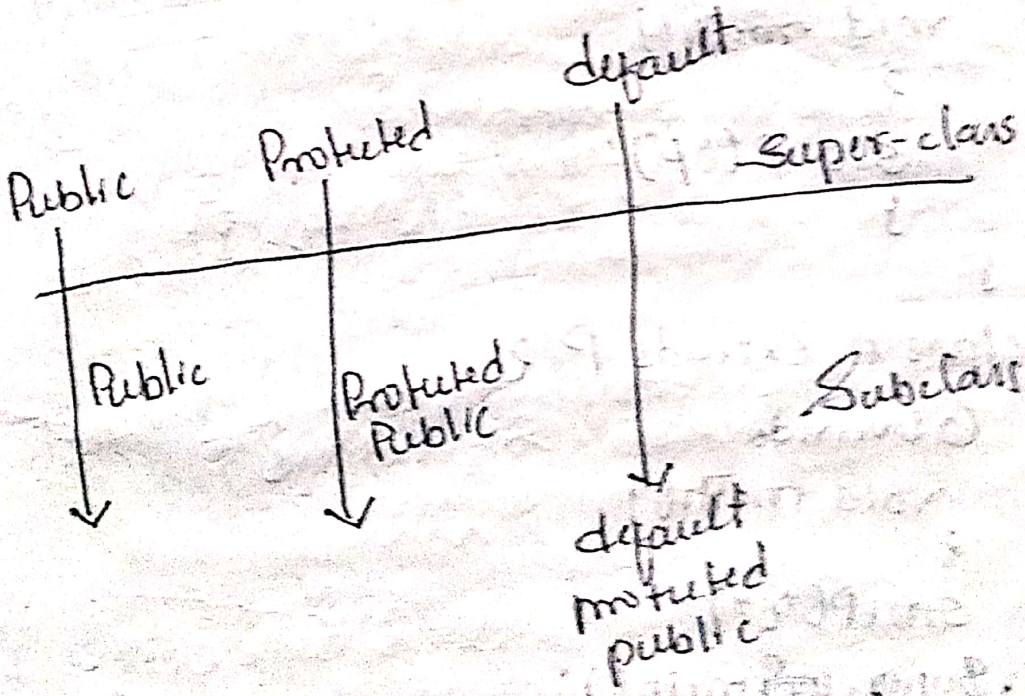
```
class Sup {
    void meth() {
    }
}
class Sub extends Sup {
    void meth(int i) {
    }
}
```

Overriding with different signature

```
class Parent {
    int meth(String s, boolean b) {
        //Something
        return 22;
    }
}
class child extends Parent {
    int meth(String x, boolean q) {
        // Something else
        return 52;
    }
}
```

In case of method overriding, the access modified can be same as declared in superclass or can be of higher visibility.





Private method cannot be overridding
∵ it cannot be inherited

Scanned by CamScanner

Final method can be inherited but cannot be overridden

Super :-

Super is a keyword which represents the immediate superclass object

Super keyword is used to access Super class variables and methods.

```
ex: class P {
      void meth()
      {
      S.O.P("P");
      }
}
class C extends P {
  @Override
   void meth()
   {
   S.O.P("C");
   Super.meth();
   }
}
C ref = new C();
ref.meth();
```

o/p C
P

Scanned by CamScanner

→ what is difference b/w Super & Super()

→ what is difference b/w this & this()

The advantage of purpose of method Overriding is to achieve Runtime polymorphism.

* we can overload main method, we cannot override main method because it is static.

```
class Demo {
P.S.V.M(int[] args) {
 S.O.P("int[]");
}
P.S.V.M(String[] args) {
 S.O.P("String[]");
}
                boolean b
P.S.V.M(~~String[] args~~) {
 S.O.P("boolean");
 }
}
```

```
class Parent {
    static void meth() {
    }
}
class child extends Parent
{
    static void meth() {
    }

}
```

Constructor chaining

Constructor chaining is the process of one constructor calling the other constructor either of the same class or super class constructor.

A Subclass constructor can call the immediate Superclass constructor using super();

A constructor of a class can call the other overloaded constructor of the same class by using this();

A constructor can call only one constructor

Inside a constructor calling the other constructor must be a first executable statement

Scanned by CamScanner

```
class Car extends Vehicle {
    Public Car() {
        This("red");
    }
    Public Car(String clr) {
        This(12500);
    }
    Public Car(int Price) {
        super();
    }
}

Public class Vehicle {
    Public vehicle() {
        S.o.P("Vehicle()");
    }
}
```

Scanned by CamScanner

```
class Vehicle {
    Public Vehicle() {
        S.o.p("vehicle()");
    }
}
class Car extends Vehicle {
    Public Car() {
        S.o.p("Car()");
    }
}
Public class ConstructorTest {
    P.S.V.m(String[] args) {
        Car c1 = new Car();
    }
}
```

Compiler generated Code

```
Public class Vehicle {
    Public Vehicle() {
        super(); -> compiler generated
        S.o.p("vehicle()");
    }
}
class Car extends Vehicle {
    Public Car() {
        super(); -> compiler generated
        S.o.p("Car()");
    }
}
```

Scanned by CamScanner

```
Public class Pen {
    Public Pen() {
        S.o.P("Pen()");
    }
}
class SketchPen extends Pen {
    String clr;
    Public SketchPen(String c) {
    this.clr=c;
    S.o.P("SketchPen()" +c);
    }
}
```

Compiler generated

```
Public class Pen {
    Public Pen() {
        super();
        S.o.P("Pen()");
    }
}
class SketchPen extends Pen {
    String clr;
    Public SketchPen(String c) {
        super();
        this.clr=c;
        S.o.P("SketchPen()" +c);
    }
}
```

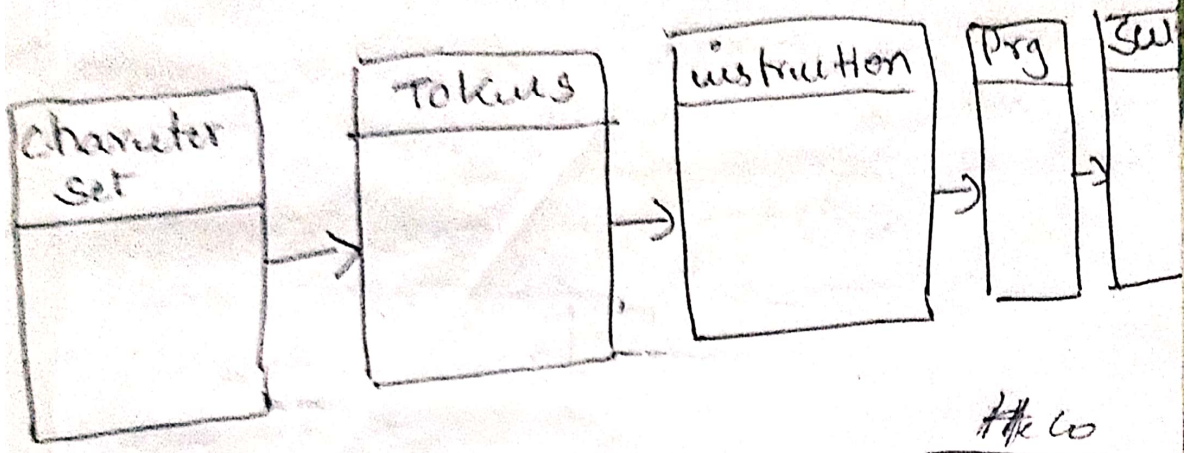
Scanned by CamScanner

```
Public class Vegetable {
    Public Vegetable() {
        S.o.P("Vegetable()");
    }
Public Vegetable(int Price) {
    S.o.P("Vegetable()" + Price);
    }
Public Vegetable(double quantity) {
    S.o.P("Vegetable()" + quantity);
    }
}
class Cucumber extends Vegetable {
    Publ Cucumber() {
        super(2.5);
        S.o.P("Cucumber()");
    }
}
Cucumber C = new Cucumber();
```

o/p:

Vegetable() 2.5
Cucumber()

If the Super class if the Zero Parameter constructor is not available then we get Compilation Error.



High level language:

Java

Middle level

C, C++

Low level

8086