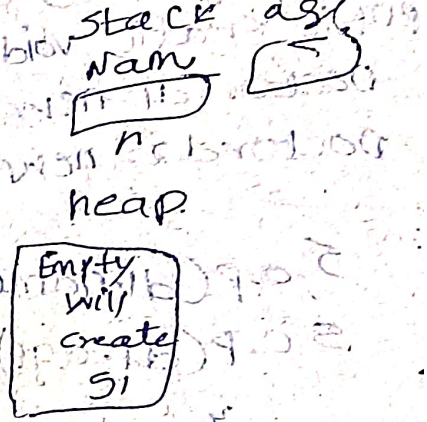
Constructor:- We use constructor we can assign data into objects

→ Syntax:-
Access modifier classname ( )

→ we can initalize objects.
→ constructor name is same as class.
→ main class will be loaded into code segment.
→ we start the execution from main class because it contains main method. the default constructor initializes instance variables with default values like int 0 String null
→ All the class and function will be stored in code segment. Java constructor doesn't have return type.

```
class student{
  public string sname;
  public int age;
  public student(string n, int a){
    sname=n;
    age=a;
  }
}
public class main{
  public static void main(string[] args){
    student s1=new student("ravi",5);
    student s2=new student("samanth",6);
    student s3=new student("devi",4);
  }
}
```



→ when we comes out of constructor every local variable belongs to that will be erased.
→ First from new keyword, and empty objects will create in heap, then after student is there then it will enter into student class, and create all code.
→ Here instance variables are these like sname and age here will be stored in heap with string as null value and age as zero.
→ After it enters to the constructor, here we are using two parameters, then here n is allocated and age is allocated. * local variable stored
→ now sname = ravi age = 5

→ After it comes out of constructor and the local variables destroyed and these constructor object will be given to reference to objects.

→ we can create n-no of constructor in a class with different parameters.

```
class Doctor {
    public string name;
    public int age;
public Doctor(string n, int a) {
    name = n;
    age = a; }
public static void main(string[] args) {
Doctor d1 = new Doctor("Dr. Pitika", 36);
Doctor d2 = new Doctor("Dr. Mandarva", 48);
S.o.p(d1.name);
S.o.p(d1.age);
}
```

During initialization of object, the instance variables name and constructor local variables of constructor should have same name in both

```
class Teacher {
    public string name;
    public int age;
    public string qualification;
Public Teacher(String n, int a, String q) {
    name = this.name;
```

If we do like this we get variable shadowing

this keyword refers to current instance of the

It is used to refer the current object address to avoid variable shadowing we will use this keyword

→ with the help of this, JVM will go to current object

If we don't mention local or instance variable then JVM by default takenas local variables.

→ Create 2-objects for class and initialize the object with the below data & display the data

object 1 → suzuki, black, petrol

Create 3- objects per class Initialize name, pen color, size, ink-type object wise the data of a

objects

class

public

public

public

public

this

this

this

this

objects → boy, blue, electric

class objects {
public string brandname;
public string color;
public string fuel type;
public Car(String brandname, string color, string fueltype
{
this.brandname = brandname;
this.color = color;
this.fueltype = fueltype; }
public static void main(String[] args) {
Car c1 = new Car("suzuki", "black", "petrol");
Car c2 = new Car("benz", "blue", "electric");
s.o.p(bname);
s.o.p(color);
s.o.p(fueltype);
}

Create 3 objects for pen class Initialize pen name, pen color, lead size, ink type in the object where display() the data of all the 3 objects

here if we want to give we can also mention them as this.brname, this.color but here we don't have same local variable so JVM is no get confused so we don't use this key word here

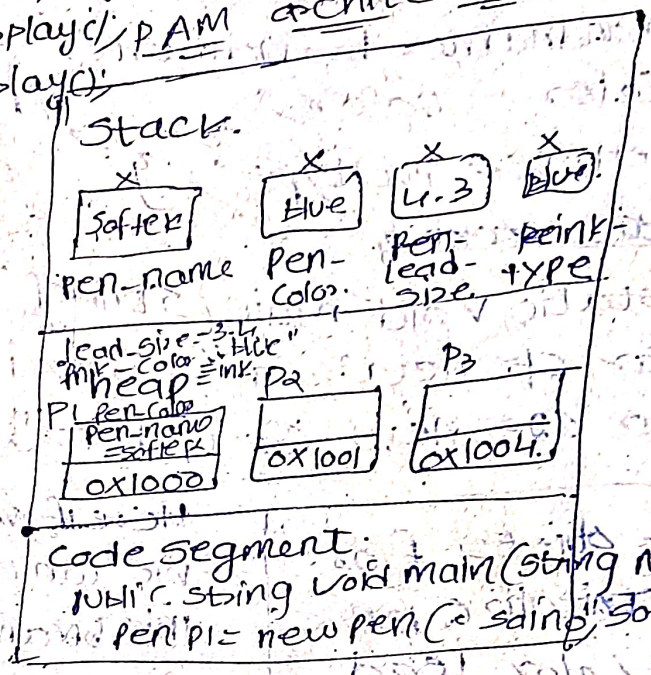
class pen {
public string pen_name;
public string pen_color;
public string lead_size;
public ink_type;
public pen(string pen_name, string pen_color, string lead_size, string ink_type,
this.pen_name = pen_name;
this.pen_color = pen_color;
this.lead_size = lead_size;
this.ink_type = ink_type;

```
public class School {
    public static void main(String[] args) {
        Pen p1 = new Pen("sonio-softek", "blue", 3.5, "Black");
        Pen p2 = new Pen("Mozato", "blue", 4.3, "Blue");
        Pen p3 = new Pen("softek", "pink", 3.2, "Blue");
        p1.display();
        p2.display();
        p3.display();
    }
}
```

```
public void display()
{
    S.O.P(pen_name);
    S.O.P(pen_color);
    S.O.P(pen_lead_size);
    S.O.P(ink_color);
}
```

A constructor can be overloaded but not overridden.

PAM architecture:-



Default Constructor:-

When programmer does not create any constructor in a class then java compiler will create a default Constructor.

Properties of default constructor:-

→ Default constructor access modifier will be always public

→ No parameters will be given for default Constructor.

→ Default constructor contains only one line of code, that is super().

```
public classname()
{
    super();
}
```

```
class A
{
    public void m1()
    {
        S.O.P("Hello");
    }
}
```

cr ⇒ Java compiler ⇒

```
class A
{
    public A()
    {
        super();
    }
    public void main()
    {
        S.O.P("Hello");
    }
}
```

→ programmer can create default constructor.

3 types of constructor:

→ Default Constructor

→ parameterless constructor

→ parameterized constructor ← why like this

A constructor cannot be abstract or static or final.

Encapsulation:- It is used for binding the data & function together as single unit (class) or place.

(OOP)

ding the data from external world and accessing odifying the data using public getters and public tters, methods.

Eg:- Hospital, here for different sections we have diffent maintenance.

Eg:- We By using these we can hide the data, by keeping variables as private

Eg:- Bank

Advantages:-

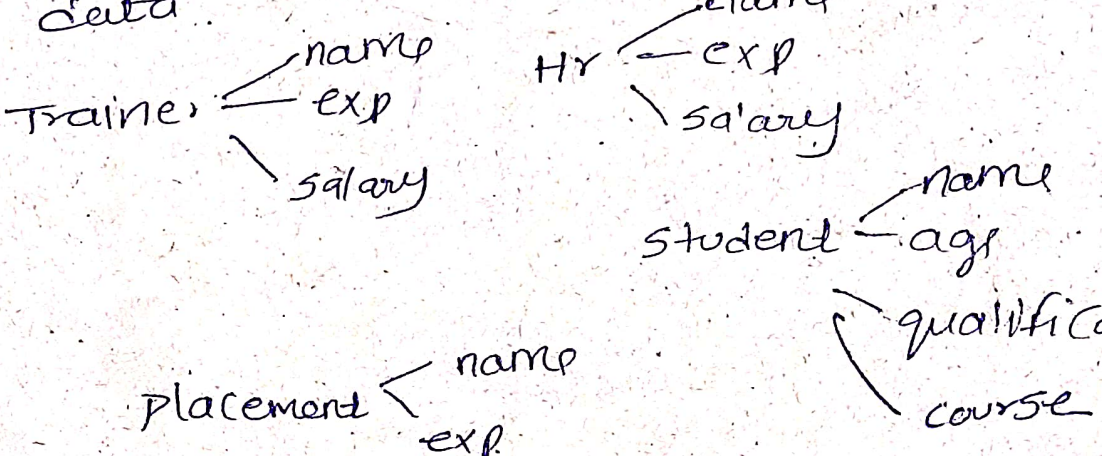
→ Fixing errors are easy.

→ Understandability/readability

→ good Maintainability

Create an application for Training Institute, Follow encapsulation for below requirement.

1. store Trainer related data, Hr related data, students related data and placement related data.



Java private:

```
class Main {
  int i;
  private Main() {
    i=5;
    S.O.P("constructor is called");
  }
```

Parameter less constructor:

```
public static void main (string[] args) {
  main obj = new main();
  S.O.P("value of i" +obj.i);
  }
}
```

It give error.

```
flag=0
n=int(input("Enter number"))
m=n/2;
for i in range (2,m):
    if (n%i==0):
        flag=1
        break;
if flag==1:
    Print("Not Prime")
else:
    Print("prime")
```

⇒ constructor is called
value of i:5

→ Declaring a constructor as private means that the constructor cannot be called from outside of class

→ For default constructor
- char – \u0000
- float – 0.0f
- double – 0.0d
- object – Reference null
- boolean – false
- byte – 0
- short – 0
- int – 0
- long – 0L

Technical quiz
Technical Riddle
Tech snak & ladder
Circuit debugging
Buzz wire
poolef ego
poster
Paper presen
public
public
public

ans=sum+ptr*
ptr*=80ee [0]
sum=sum+ptr*

```
class Trainer {
public String name;
public int experience;
public int salary;
 public Trainer (String name, int experience, int salary)
      {
  this.name = name
  this.experience = experience
  this.salary = salary
      }
```

```
public void display() {
 S.O.P(Trainer.name);
 S.O.P(experience.
       Trainer.experien
S.O.P(Trainer.experience);
      }
```

```
Class Hr {
  public String name;
  public int experience;
  Public int salary;
  public Hr (String name, int experience, int salary) {
    this.name = name
    this.experience = experience
    this.salary = salary
      }
```

```
public void display2() {
  S.O.P(Hr.name);
 S.O.P(Hr.experience)
 S.O.P(Hr.salary);
 }
```

```
Class students {
   public String name;
   Public int age;
   public int qualification;
   public String course;
    }
```

```
public void display() {
  S.O.P(student.name);
  S.O.P(student.age);
 S.O.P(student.qualificatia);
 S.O.P(student.course) }
```

```
public Students (String name, int age, int qualification
                 String course) {
   this.name = name
       this.age = age
       this.qualification = qualification
       this.course = course }
```

```
Class placement {
   public nameString name;
   publi( exp;
          public Placement {
```

```
public void display2()
S.O.P(placement.
S.O.P(placement.
S.O.P(placement.
      }
```

this.name = name
this.exp = exp
}
}

public class Institute {
public static void main(String[] args) {
trainer t = new trainer("pupa", 2, 50,000);
Hr h = new hr("pupa", 1, 25,000);
students s = new students("santhoshi", 25, "B.Tech", "Java");
placement p = new placement("chennam", 2);
}
trainer.display1();
Hr.display2();
student.display3();
placement.display();
}
}

① class person {
public String name;
public int age
public person(String name, int age) {
this.name = name
this.age = age
}
public void greet() {
S.O.P("Hello, How are you?");
}
class public class pupa {
public static void main(String[] args) {
person p = new person("pupa", 25);
S.O.P(name)
S.O.P(age)
greet()
}

```
class Rectangle {
Public int width
Public int height;
  public Rectangle(int width, int height){
   this.width= width
   this.height=height;
  void area{
   Area= length* breadth
   return Area;
public class dimensions {
  public static void main(String[] args) {
   Rectangle R=new Rectangle(2,3);
   S.O.P(width)
   S.O.P(height)
   area()
   }
}
5) class Car {
     String brand,
     String model,
     int year,
  public Car (String brand, String model, int year){
     this.brand=brand
     this.model=model
     this.int=year;}
  void start(){
     S.O.P("Car has started");}
  public class new{
     public static void main(String[] args){
     Car c=new Car("Thar", "Rupa", 2021)
     S.O.P(brand)
     S.O.P(model)
    S.O.P (year)
     start() }}
```

```
    String title;
    String author;
    int year;
}
public Create(String title, String author, int year){
this.title = title;
this.author = author;
this.year = year
}
void description{
  S.O.P("published in 1949")}
public class book{
  public static void main(String[] args){
   Create c = new Create("Rupa", "devi", 2024);
   S.O.P(title)
   S.O.P(author)
   S.O.P(year)
   description();
  }}
```

private String ;

→ To access private variables outside the class (i.e even owner can't able to access), but indirectly we can access with the help of getters and we can modify with the help of setters.

Getter:-

which is used to access private variables

his method has return type which is same as the type of private variable, and no parameters.

e method contains only one line of statement i.e eturn statement.

The getter method always should be public.

method name should be start with the ~~private variable~~ name followed by get followed by private variable name.

Setters:- It is a method which is used to modify the data of the private variable from outside of the class.

→ access modifier is public.
→ return type would be void.
→ method name start with set followed by private variable name.
→ It allows, or type of parameter should be same as private variable type.
→ Inside the method the programmer should assign the parameter value to private variable.

```
class Bank {
    private int balance = 10000;
    public ~~get balance~~ int getBalance()
    {
        return balance;
    }
    public void setBalance(int balance) {
        this.balance = balance;
    }
}
public class School {
    public static void main(String[] args) {
        Bank b = new Bank();
        int account_balance = b.getBalance();
        S.o.P(b.balance)
    }
}
```

Directly we can't access private variable, so it gives errors.

} → getter method, it should be in these format only if we use anything other than these it will work as normal function

Time & distance
Time & work
ratio & propotion
% percentages
boats & streams
Data interpretation
Seating arrangement
Coding & decoding

// S.O.P (account_balance);

b.balance = 50000; → compile time error

b.setBalance(50000); → Directly we can't Modify by using bean class.

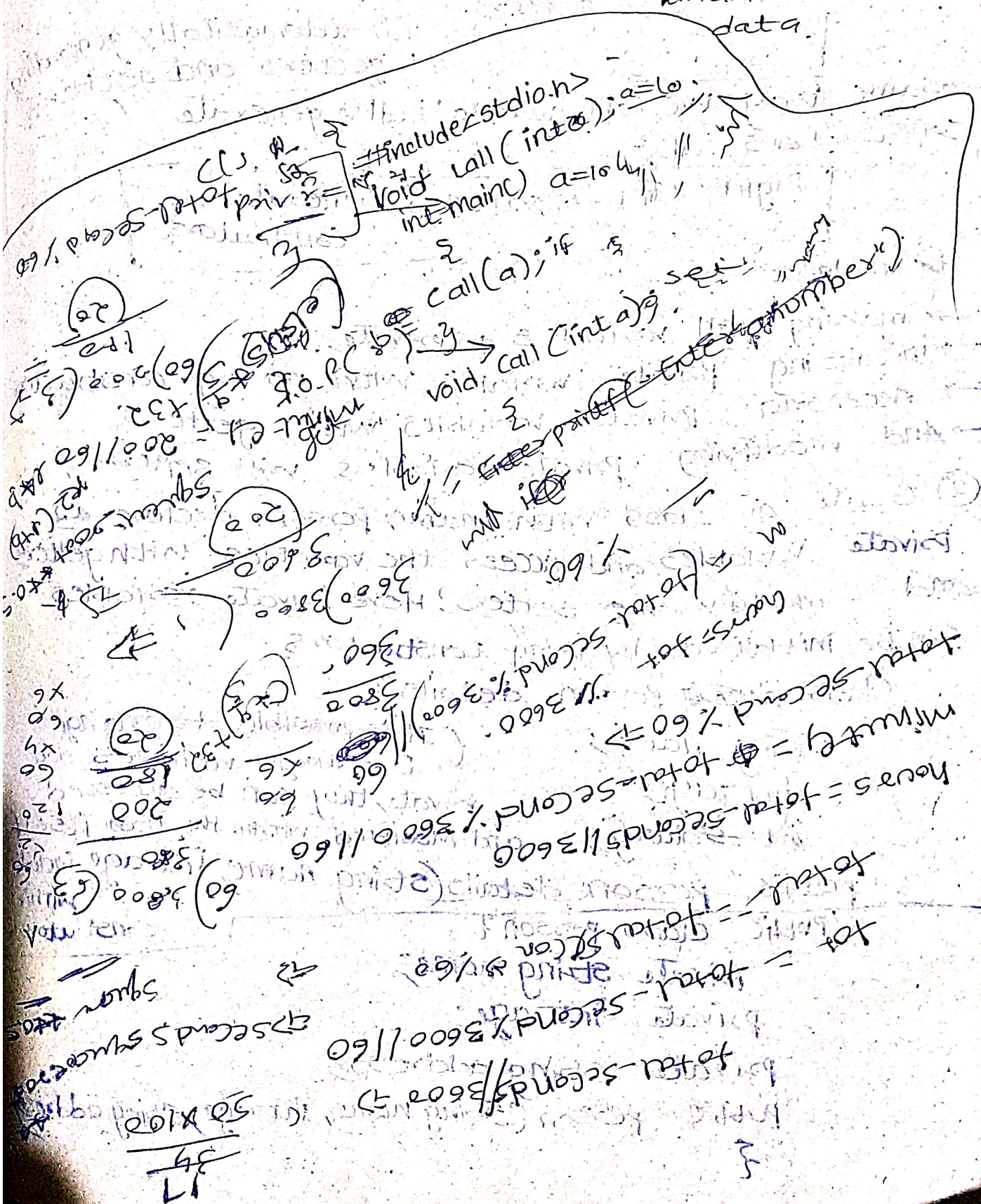
// account_modifying

account_balance =

S.O.P("After modifying:");

S.O.P(account_balance); → By making private variable we can hide data.

→ Encapsulation will hide the data.



→ We can create multiple private variables in a class.

→ We can access multiple private variables from outside the class. By creating methods for each and every variable separately.

→ We can modify multiple private variables from out side the class. By Creating

> Right click → source → Generate getters & setters → for automatically generating getters and setters

→ same way we can automatically generate constructors also.

> Right click → source → Generate constructors

Bean class:-

→ making all variables as private
→ initializing private variable with no arg constructor
→ Accessing private variables with getters.
→ And modifying private variables with setters.

② Create a class with name person declare 3 private variables and access the variables with getters and modify the setters? Here private variables can be initialized by using constructors

```
Class person_details{
    String name;
    int age;
    int salary;
    public person_details(String name, int age, int salary){
    }
```

(It is possible to change even though variables are private, they can be accessed and modified within the class itself including within constructors)

```
Public class person{
    private String name;
    private int age;
    private String address;
    public person(String name, int age, String address){
    }
```

```
        this.name=name;
        this.age=age;
        this.address=address;
    }
public String getName(){
    return name; }
public void setName(String name){
    this.name=name;
}
public int getAge(){
    return age;
}
public void setAge(int age){
    this.age=age;
}
public String getaddress{
    return address;}
public void setAddress(String address){
    this.address=address;}
public static void main(String[] args){
    Person p=new Person("John Doe",30,"123 Main St");
    S.o.println("Name:"+person.getName());
    S.o.println("Age:"+person.getAge());
    S.o.println("Address:"+person.getAddress());
    person.setAge(31);
    S.o.println("updated age:"+person.getAge());
    person.setName("Jane Doe");
    S.o.println("updated name"+person.getName());
}
}
```

We can't access private variable/ outside class by using classname, we can access by object name only.

If we change order of arguments then it will get error.

Abstraction:-

Hiding the implementation details, and providing only required information.

E.g:- Projector, remote, Fridge, Fan

Here we are only using, we don't know any implementation techniques.

Advantages:-

→ code security

→ To achieve abstraction

→ abstract methods & abstract classes

→ Interface.

Inheritance:-

Inheritance:- Acquiring all properties.

```
Class Father{
    public boolean house = True;
    public sting land = 50acres;
    public int car=1;
}
Class son{
    public boolean house=true;
    public string land = 5000acres;
    public int
}
```

→ By using inheritance code duplication will be reduce.

→ And we can reuse the code.

→ Here code duplication occurs, so we use inheritance.

```
Class parentclassname
{
==
}
Class childclassname extends parentclassname{
--
}
```

predefined word, which will inherit all the members from parent class to child class

```
Class Father{
    public boolean house=true;
    public int car=1;
```

public string land = "500 acre";

}

class son extends Father {

public int bike = 1;

}

public class inheritance {

public static void main(String[] args) {

Son s = new son();

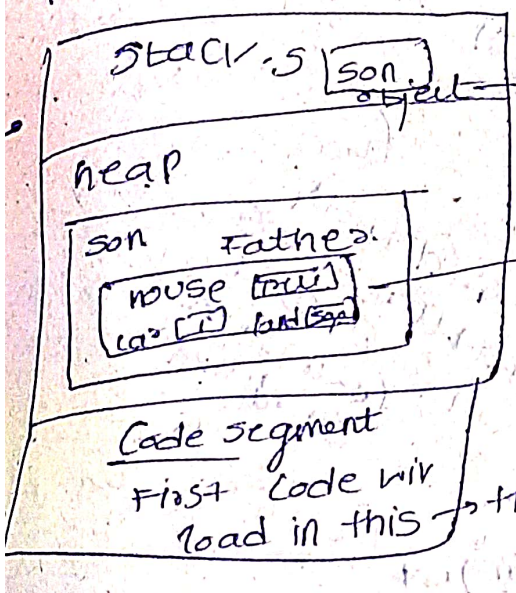
s.o.p(s.bike);

s.o.p(s.car);

here for we will store the address

→ In inheritance object should be created for child, we can access parent class members also.

Ram architecture



→ First parent class will be loaded.

→ then only object created.

→ If parent class and child class having same variable then we will use child class.

→ Create Lion class and create method with no return value and no parameter and print a Lion will roar. (roar())

inside the method.

→ Create cub class which also have same functionality of lion class. roar, then create object of lion class and call the method.

→ How will you inherit the class in java?
By using extends keyword.

→ What is the use of extends keyword?
All the members of parent class to child class.

→ class Car {
}
class Benz extends Car {
}

Ans:- For benz we will create the object.

For which object you will create object

→ which class will be loaded first in ram?
parent class.

→ parent class object is the part of child class object :- True.

→ Accessing child class members using parent object
False.

→ If parent class and contains same method name, which class method will be accessed, if we call using child object variable?
Child class method.

→ Advantages of inheritance
reusability of code.

Types of inheritance:-

→ single level inheritance
→ multi level inheritance
→ Hierachical inheritance
→ multiple inheritance
→ Hybrid inheritance

→ applicable for classes
→ not supported for classes

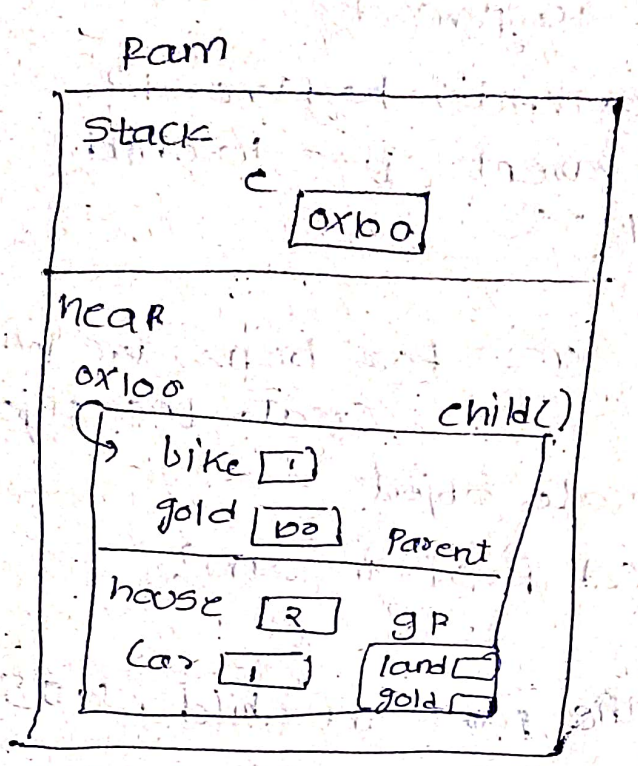
single level inheritance:-

```
class Parent
{
}
class child extends Parent
{
}
```

Multi-level:-


```
class GP
{
}
class P extend GP
{
}
class child extends Parent
{
}
```

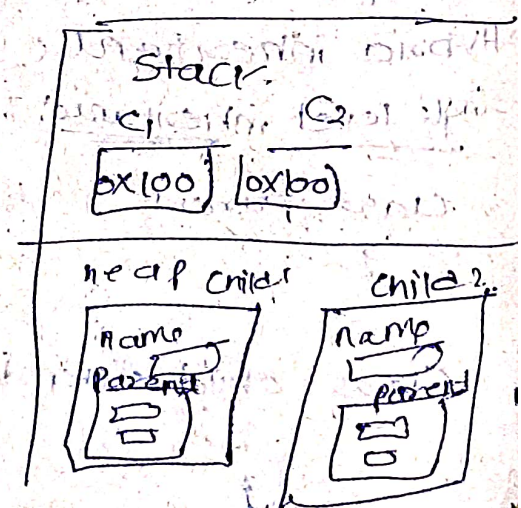
Multilevel:



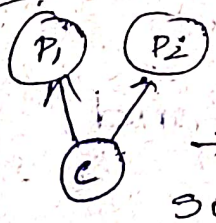

```
class cp {
  public string land = "yes";
  public string gold = "100";
}
class p extends cp {
  public int house = 2;
  public int car = 1;
}
class child extends p {
  public int bike = 1;
  public int gold = 100; }
  main() {
    child c = new child();
    S.O.P(c.bike);
    S.O.P(c.gold);
    S.O.P(c.house);
    S.O.P(c.land);
  O/P:- 1
  100
  2
  yes
```

Hicarchical inheritance:

```
Class parent {
  Public int balance = 20,000;
}
Class child1 extends parent {
  Public string name = "Asha";
}
Class child2 extends parent {
  Public string name = "usha";
}
main() {
  child1 c1 = new child1;
  child2 c2 = new child2;
  S.O.P(c1.name); → Asha
  S.O.P(c1.balance); → 20,000
  S.O.P(c2.name); → usha
  S.O.P(c2.balance); → 20,000
```



Multiple inheritance:-



→ multiple parents classes will be inherited in one child class.

→ In java multiple inheritance are not supported for java. ↑ confusion

→ Class P1 → because of function ambiguity multiple inheritance is not possible.

{ }

→

class P2

{ }

→ once we write, automatically it will throw error.

class C extends P1, P2

{ }

Function ambiguity:-

Lion

```
class Lion {
  P.V. makeSound()
  {
  }
  S.O.P("Roar");
}
```

Lioness

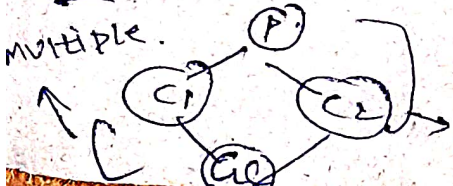
```
class Lioness
{
P.V. makeSound()
{
}
S.O.P("Roar");
}
```

Cub

```
class Cub {
  P.V. display() {
  makeSound();
  }
}
```

Here JVM will get confused that what function we have to call, so that's sun microsystem removed this feature.

Hybrid inheritance:- → Diamond inheritance.



combination of hierachical inheritance and multiple inheritance.

Hierachical

→ Hybrid inheritance is also not supported in java.

→ Create a class with name Calculator and implement add(), sub(), div(), mul(),

Create another class with name ScientificCalculator and implement add(), sub(), div(), mul(), square(), cube().

```
class Calculator{
    p.void add() {
        return 10+20; or
    }

    p.void sub(int a, int b) {
        return a-b;
    }

    public void mul(int a, int b) {
        return a*b; }

    public void div(int a, int b) {
        return a/b; }

    p.void add(int a, int b)
        return a+b;
    }
}

class scientific Cal extends Calculator {
    p.v. square() {
        }

    p.void cube() {
        }
}
```

Object class:- package java.lang

→ Every class in Java directly indirectly, derived from Object class.

→ parent class of all the classes in Java.

In constructor we are not able to create instance.

- can I inherit constructor from parent class to child class?

constructor will not be inherited but it can be inherited because of super() [indicates parents class]

parent or derived or super class

child or sub or derived

↓

mainly super and sub is used.

super() :- used to access parent class members in child class.

super() → it will call parent class constructor from child class constructor.

```
class parent{
    public parent()
    {
    s.o.p("parent con");}}

class child extends parent
{
public child(){
 s.o.p("child constuctor");}}
```

<=> compiler

```
class parent{
public
    parent()
    {
    super();
    s.o.p("parent
     con");
    }}

class child{
public child()
{
super();
s.o.p("child const");
}}
```

Constructor chaining:

```
class parent1{
 public parent1(){
  super()
  s.o.p("parent constructor");}

class child extends parent1{
  public child1()
  {
  super()
  s.o.p("child constructor");}

child c=new child1();
```

here it doesn't have any parent cls, so print constructor will execute

It will check any parent is class there

Initially here there is no super(), but compiler automatically calls JVM

polymorphism:

poly → means → many

morphism → form or behaviour

An entity which has same name with multiple behaviours.

Types of polymorphisms

Compile-time polymorphism or static polymorphism (or) method overloading

Run-time polymorphism (or) dynamic polymorphism (or) method overriding

Method overloading:- [compile time polymorphism]

multiple methods have same name with different parameters

→ compiler will check only the method name and the parameters, so programmers can give any access specifier.

```
class A {
    public void m1() {
        ==
    }
    public int m1(int x) {
        ==
    }
    public string m1(string s) {
        ==
    }
}
```

```
class B
{
    public void m1()
    {
        ==
    }
    public void m2() {
        ==
    }
    public void m3() {
        ==
    } }
```

This is not an example for method overloading because here name of the method is different.

Different parameters:-

→ different type
→ different no. of parameters
→ different order of parameters.

Different type of parameter:-

```
→ class A {
    public void m1(int x) {
        ==
    }
    public int m1(int y) {
        ==
    }
    public boolean m1(string s) {
        ==
    }
}
```

y → this will throw error because above one & this one same data type

while compiling it checks only datatype, if datatype same then it will raise an error.

Methods with different no. of parameters:-

```
class A {
  public void m1(int x)
    {
    =
    }

  public int m1(){
      = y

  public String m1(int x, int y)
    {
    =
  y
```

Different order of parameters:-

```
class A {
  public int add(int x, int y) {
    =
    }
  public float add(float y, int z) {
      = y
  public int sub(int x, int y) {} → This also not taken
    = y
  public int add(int y, float s) {} → This will not allowed
                  }
```

Before overloading:-

Addition of 2 numbers, 3-numbers, 4-numbers

```
class Addition; {
  Public void add(int a, int b)
  {
  s.o.p(a+b);
  }
public void add2(int x, int y, int z)
  {
  s.o.p(x+y+z); }
public void add3(int x, int y,
  int z, int a) {  }}
```

main() {
Addition a = new addition();
a.add1(10,20);
a.add2(10,20,30);

→ remembering all the method names will not be easy for programmer, so whatever you create multiple method with same functionality but different input, you can use method overloading.

Calling Methods:-

```
class A {
  public int fun1(String x) {
    return 10;
  }
  public String fun2(int x) {
    return "Hello";
  }
  public float fun1(int x) {
    return 3.156f;
  }
}
Public class B {
  public static void main(String[] arg) {
    A a1 = new A();
    a1.fun1("hello");
    a1.fun1(10);
  }
}
```

When programmer calls overloaded methods JVM will decide, which method should be executed based on the values passed in the method call.

```
for(int i=0; i<=a.length-i-1; i++) {
  for(int j=0; j<a.length-i-1; j++) {
    if (a[j]>a[j+1])
    {
      int temp= a[j];
      a[j]=a[j+1];
      a[j+1]= atemp; } }
S.O.P(Arrays.toString(a));
```

~~Method overloading~~

Method overriding: Run time polymorphisms.

→ The method name in base class and child class must be same and also no. of arguments and parameters must be same. Return type must be same.

→ private methods cannot be overrided.

→ static methods cannot be overrided.

→ final methods cannot be overrided