

---

---

---

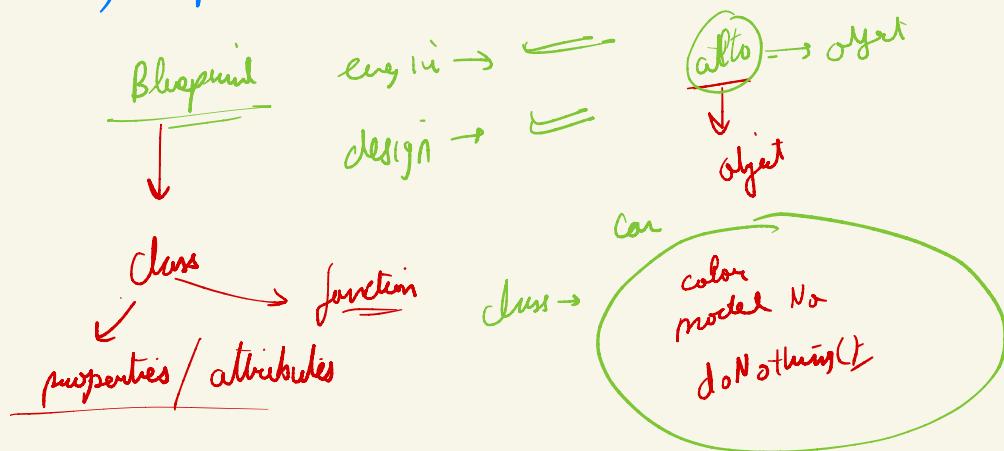
---

---



# # OOP ↳ Object Oriented Programming.

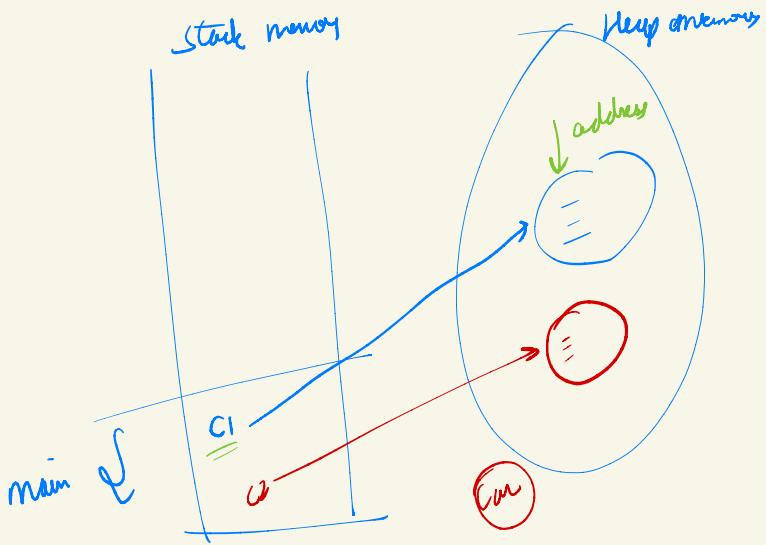
car making company → Hyundai / Honda / BMW



# Syntax      class Car {      Car c1

# Object Syntax

class Name {  
 objectName = new Classname();  
 objectName = new Classname();  
 ↓  
 address  
 C1.  
 # function { property access }



Ques) Create a class Student  
 give attributes → name, roll\_no, age  
 give function → print "anything"  
 # default values

# Access Specifiers  
 anyone can access (even outside the class)

- 1) public → only sub class can access.
- 2) private → no one can access
- 3) protected → same package
- 4) default

# Constructor

↳ function inside your class with name same as that of class and no return type.

Student s1 = new Student();  
↳ constructor call

public Student (int f) {

}

# Pillars of OOPs

⇒ Encapsulation

⇒ Abstraction

⇒ Inheritance

⇒ Polymorphism

# Encapsulation ↳ enclosing something in a capsule.

→ It can be achieved by using "private".

```

class Student {
    private String name;
    int age;
}

```

```

class DOL {
    public main() {
        Student s = new Student();
        s.name = "John";
    }
}

```

# Inheritance

superclass / parent class

class Vehicle {
 String color;
 public void honk() {
 System.out.println("peepo");
 }
}

↓ inheritance

sub class

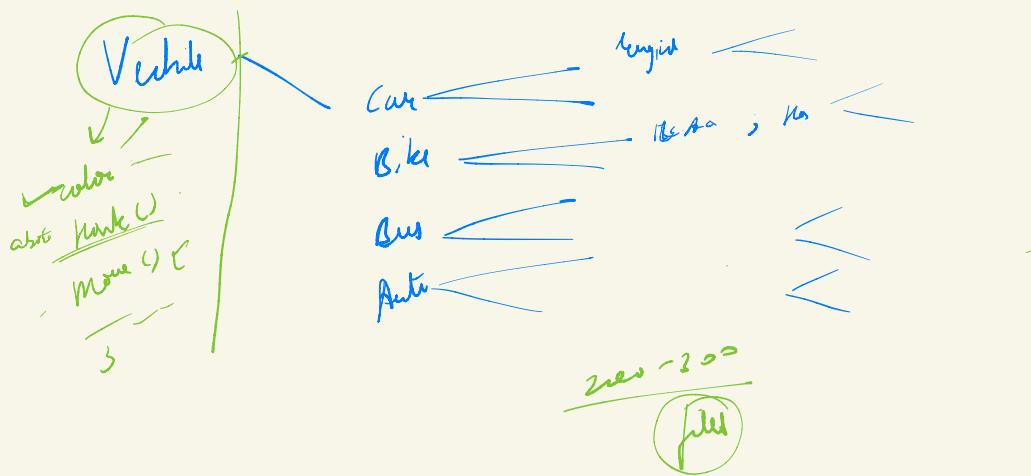
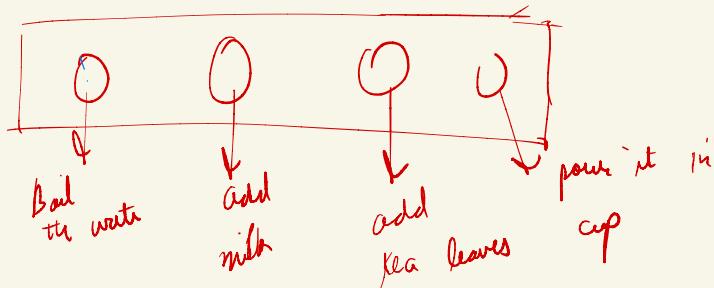
class Car extends Vehicle { }

class Bike extends Vehicle { }

DRY  
↳ don't repeat yourself

# Absstraction  
↳ hiding implementation

Mocha Tea



# Interface  
↳ can be used to achieve abstraction -

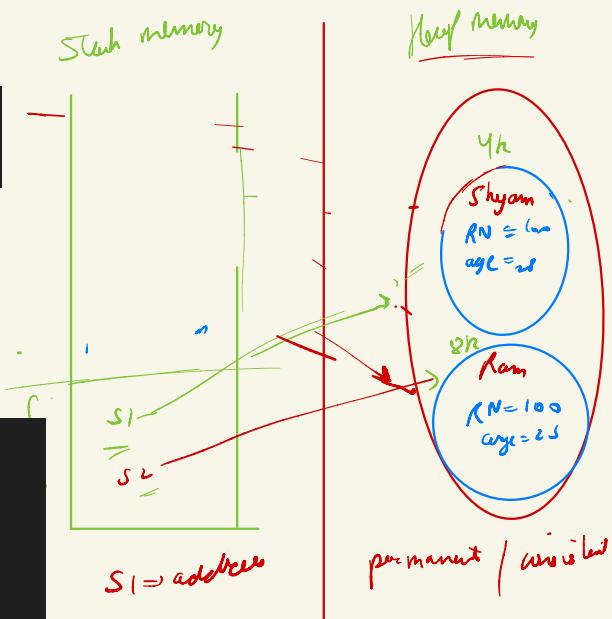
Java

## # Memory mapping

```
class Student {  
    private String name="Samrat";  
    int roll_num=100;  
    int age=25;
```

```
public static void swapNames(Student s1, Student s2){  
    String nameOfS1=s1.getName();  
    String nameOfS2=s2.getName();  
  
    s1.setName(nameOfS2);  
    s2.setName(nameOfS1);  
}
```

```
run| debug  
public static void main(String[] args) {  
    // className objectName = "new" + "ClassName" + "();"  
  
    ✓ Student s1=new Student();  
    ✓ Student s2=new Student();  
  
    ✓ s1.setName(newName: "ram");  
    s2.setName(newName: "shyam");  
  
    ✓ swapNames(s1,s2);  
    ✓ s1.getName(); -  
    ✓ s2.getName();  
    // Student s3=new Student(25);
```



## # Inheritance

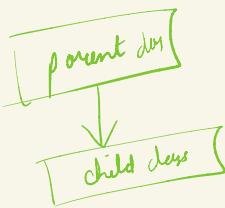
class Parent {  
 int id;  
 String color;

↓  
class A extends Parent {

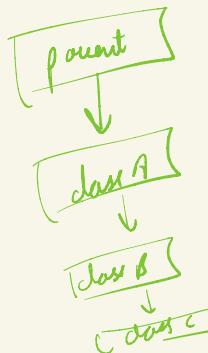
}

this, super

## \* Single Inheritance

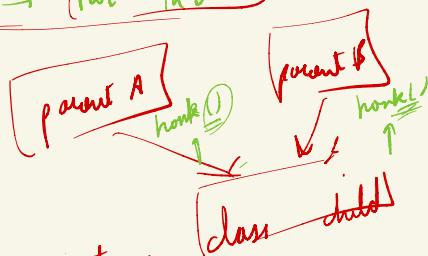


## \* Multi-level Inheritance

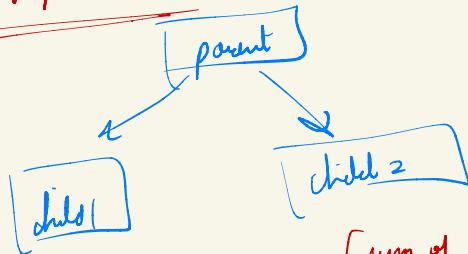


to reduce complexity  
to reduce ambiguity()

## # Multiple Inheritance → [not in Java]

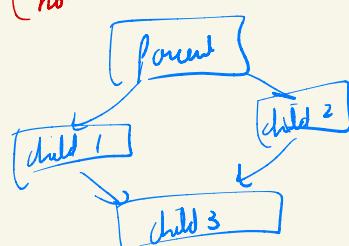


## # Hierarchical Inheritance



[sum of Hierarchical and ~~multiple~~]

## # Hybrid Inheritance



Ques

Create one multi level inheritance.

Create default constructors for each class.

Create the instance for lower most class.