OOP
↳ Object Oriented Programming.

Car making company → Hyundai / Honda / BMW

Blueprint
engine → ___
design → ___

alto → object
↓
object

↓
class → function
↙
properties / attributes

car
class → color
model No
doNothing()

Syntax

class Car {

}

Car c1

Object syntax

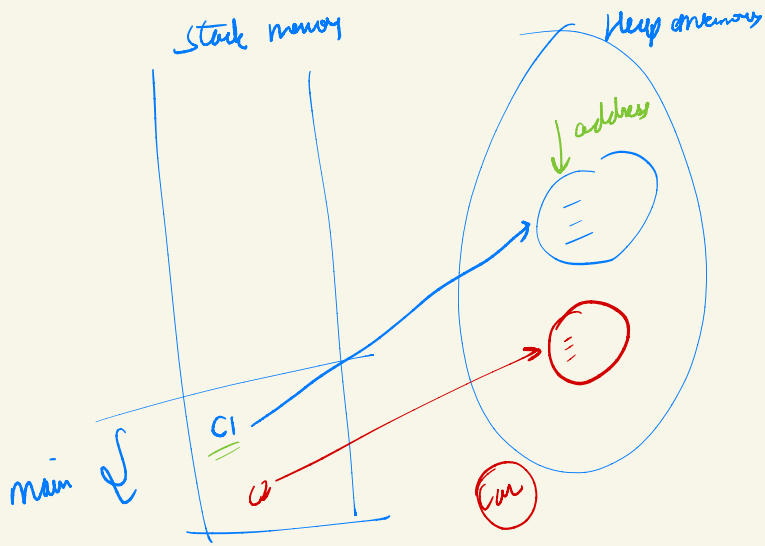
className objectName = new className();

Car c1 = new Car();

c1 ↓ address

function / property access

c1.



Ques) Create a class Student

give attributes → name, roll_no, age

give function → print "anything"

default values

Access Specifiers

1) public → anyone can access (even outside the class)

2) private → no one from outside can access

3) protected → only sub class can access.

4) default → same package

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

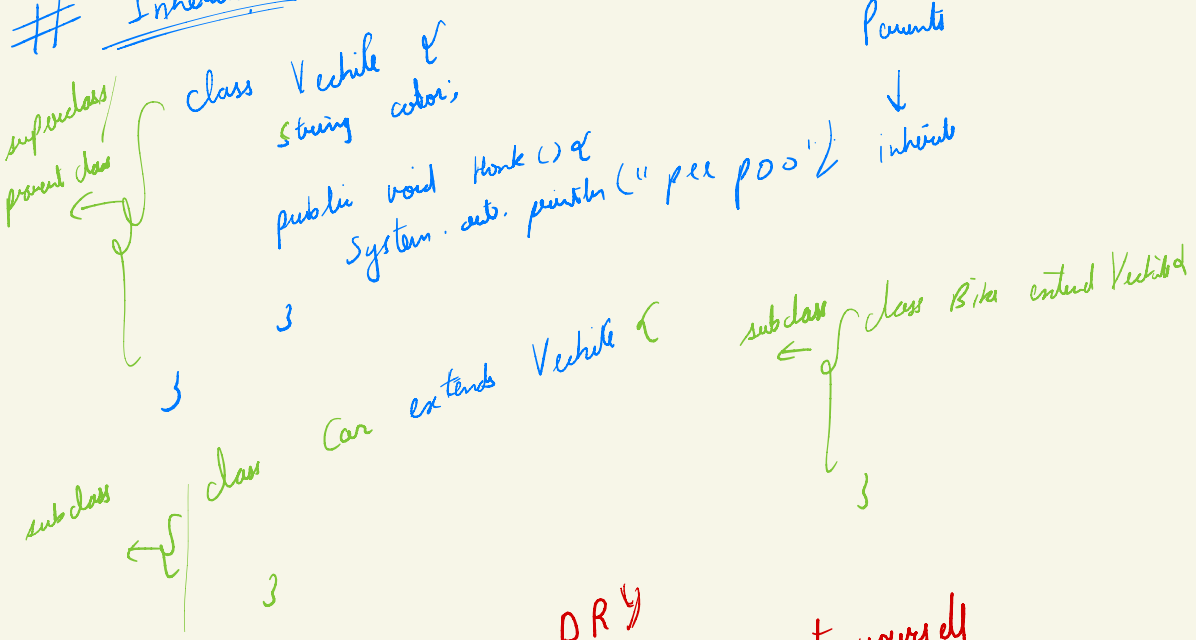
Encapsulation

↳ enclosing something in a capsule.

→ It can be achieved by using "private".

```
class Student {
    private String name;
    int age;
}

class OOP {
    psv main () {
        Student s = new Student ();
        s.name .
    }
}
```

Inheritance

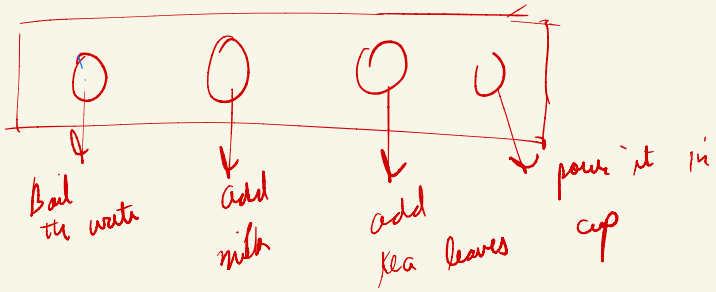


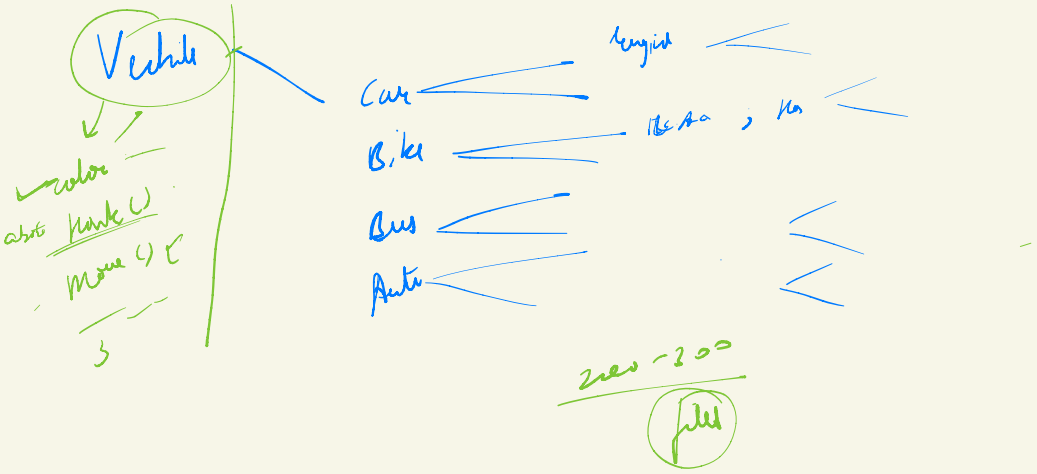
DRY
↳ don't repeat yourself

Abstraction

↳ hiding implementation

Mocha Tea





Interface

↳ can be used to achieve abstraction.

Java

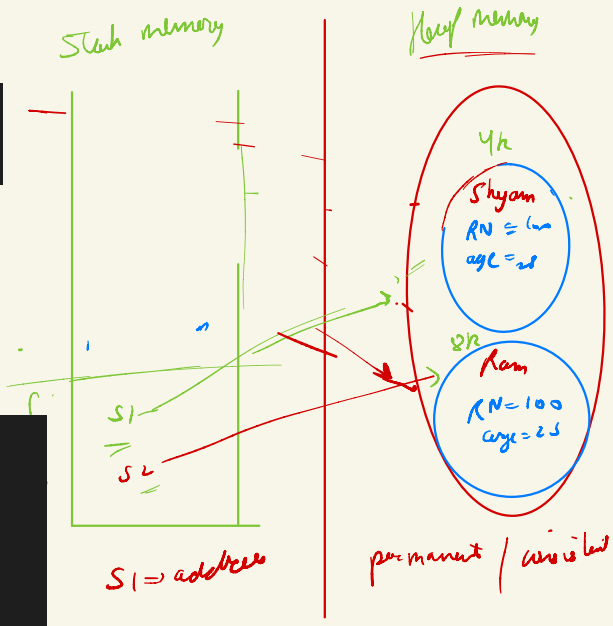
Memory mapping

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

```java
public static void swapNames(Student s1, Student s2){
    String nameOfS1=s1.getName();
    String nameofS2=s2.getName();

    s1.setName(nameofS2);
    s2.setName(nameOfS1);
}
```

```java
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);
```

S1 = 4k
S2 = 8k



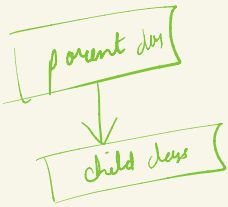
Inheritance

class Parent {
int id;
String color;
}

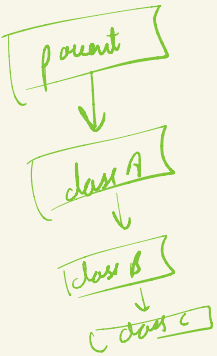
class A extends Parent {
}

this, super

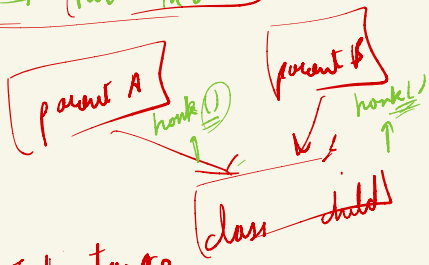
★ Single Inheritance



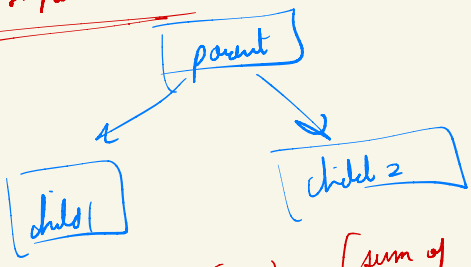
★ Multi level Inheritance



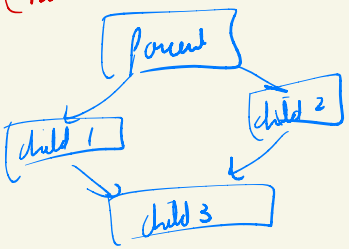
Multiple Inheritance → not in Java → to reduce complexity → to reduce ambiguity();



Hierarchical Inheritance



Hybrid Inheritance ✗ (not in Java) (sum of Hierarchical and multiple ✗)



Ques Create one multi level inheritance.
Create default constructors for each class.
Create the instance for lower most class.

---

static keyword

↳ for those attributes, whose value will stay the same irrespective of the object.

→ Value for that attribute will be the same for all objects.

Ques Create one class "Car". Then create some Car objects and keep a count of how many objects you created.

Constructor and Constructor Chaining → a constructor is called from a different constructor

default constructor
↓
no parameters

parameterized constructor
(parameter)

Types of Constructor Chaining

1) Same class
2) from other class.

⇒ Rules for constructor chaining

→ constructor calling should be just link.

→ order don't matter.

→ there should be at least one constructor without any chain call.

---

final keyword

Polymorphism + Overloading