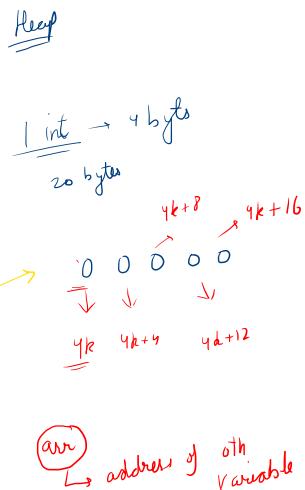
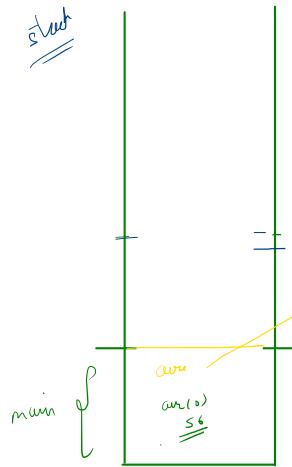


# # Memory Mapping

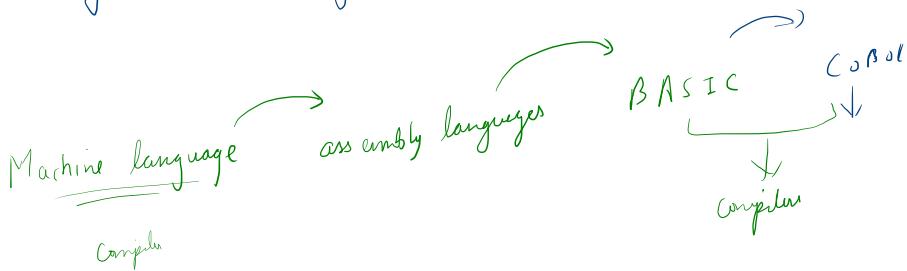


```
public static void swap(int a, int b){  
    int temp = a;  
    a = b;  
    b = temp;  
}  
  
public static void increaseArray(int[] arr){  
    arr[0] = 56;  
}  
  
public static void main(String[] args){  
    1) int[] arr = new int[5];  
    2) increaseArray(arr);  
  
    //int a = 2;  
    //int b = 5;  
  
    //swap(a,b);  
    //System.out.println(a+" "+b);  
}
```

arr (3)  
↓  
4k + 12  
arr (0)  
↓  
base address + 5 \* 4

garbage - collector

# # OOPs → Object Oriented Programming



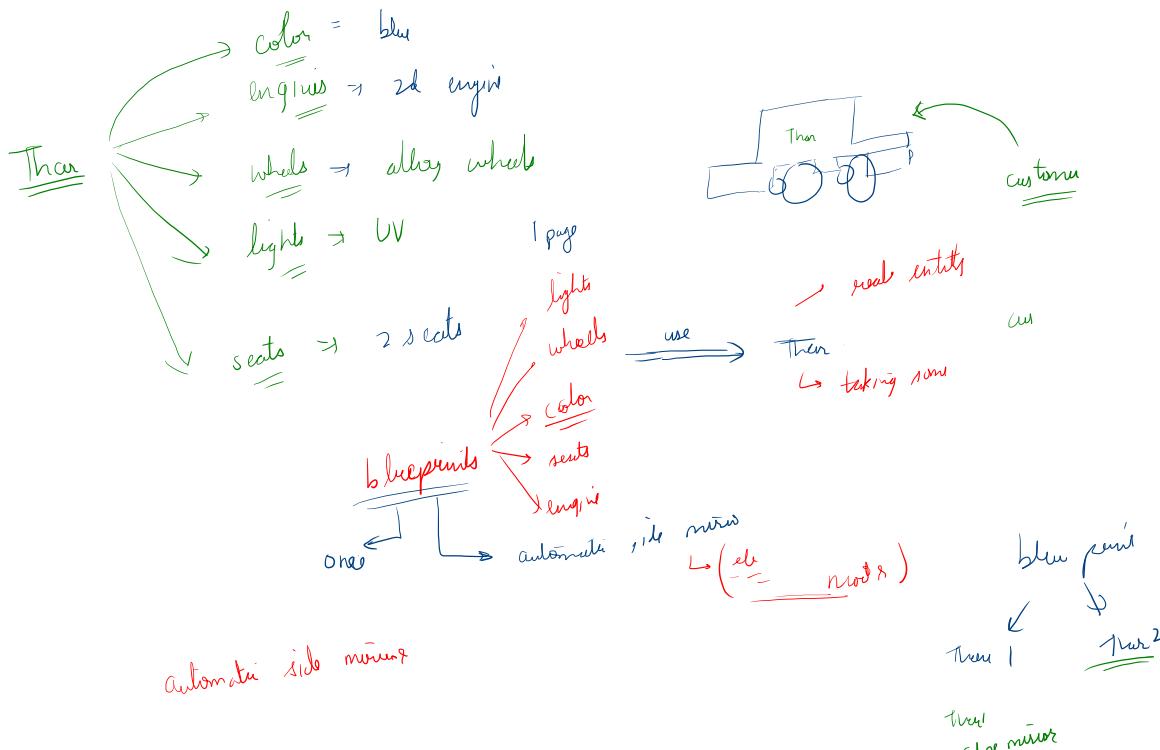
# C → purely functional language

c++ → C with OOP features

Java → Object oriented

#

Car



→ Class → a blueprint

↙ properties / attributes  
methods / functions  
class (or of  
→ String color;  
→ int seats;  
→ String engine)

name - of - class name of object  
= new Name of der

)

Object  
↓  
object

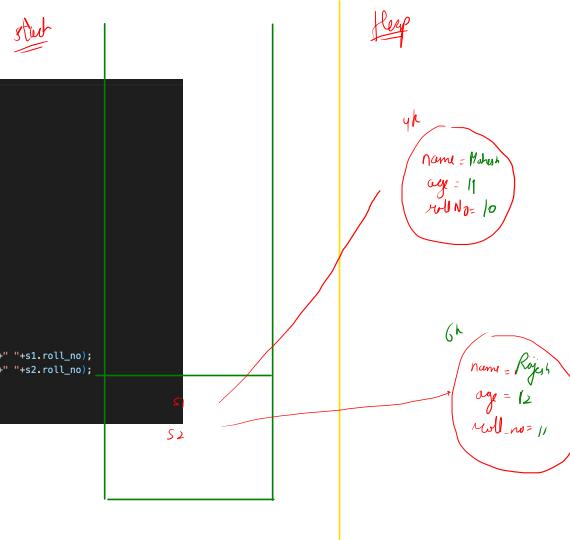
Car c1 = new Car();  
c1. color = "Red";  
c1. seats = 3;  
c1. engine = "V8";  
System.out.println(c1.color)

class student {  
String name;  
int roll\_no;  
int age

Student s1 = new student();  
s1.name = "Mahesh"  
s1.roll\_no = 1;  
s1.age = 11;

Student s2 = new student();  
s2.name = "Akash"  
s2.roll\_no = 2;  
s2.age = 12;

```
class Student {  
    String name;  
    int age;  
    int roll_no;  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Student s1 = new Student();  
        s1.name = "Mahesh";  
        s1.age = 11;  
        s1.roll_no = 10;  
  
        Student s2 = new Student();  
        s2.name = "Rajesh";  
        s2.age = 12;  
        s2.roll_no = 11;  
  
        System.out.println("Properties of s1 -> " + s1.name + " " + s1.age + " " + s1.roll_no);  
        System.out.println("Properties of s2 -> " + s2.name + " " + s2.age + " " + s2.roll_no);  
    }  
}
```



(ArrayList<Integer> al = new ArrayList<>();

ArrayList<String> al = new ArrayList<>();

ArrayList<E> al =

```

class Student {
    String name = "Amit";
    int age;
    int roll_no;

    public void makeNoise(){
        System.out.println("Making noise from "+name);
    }

    public void increaseAge(){
        age++;
    }
}

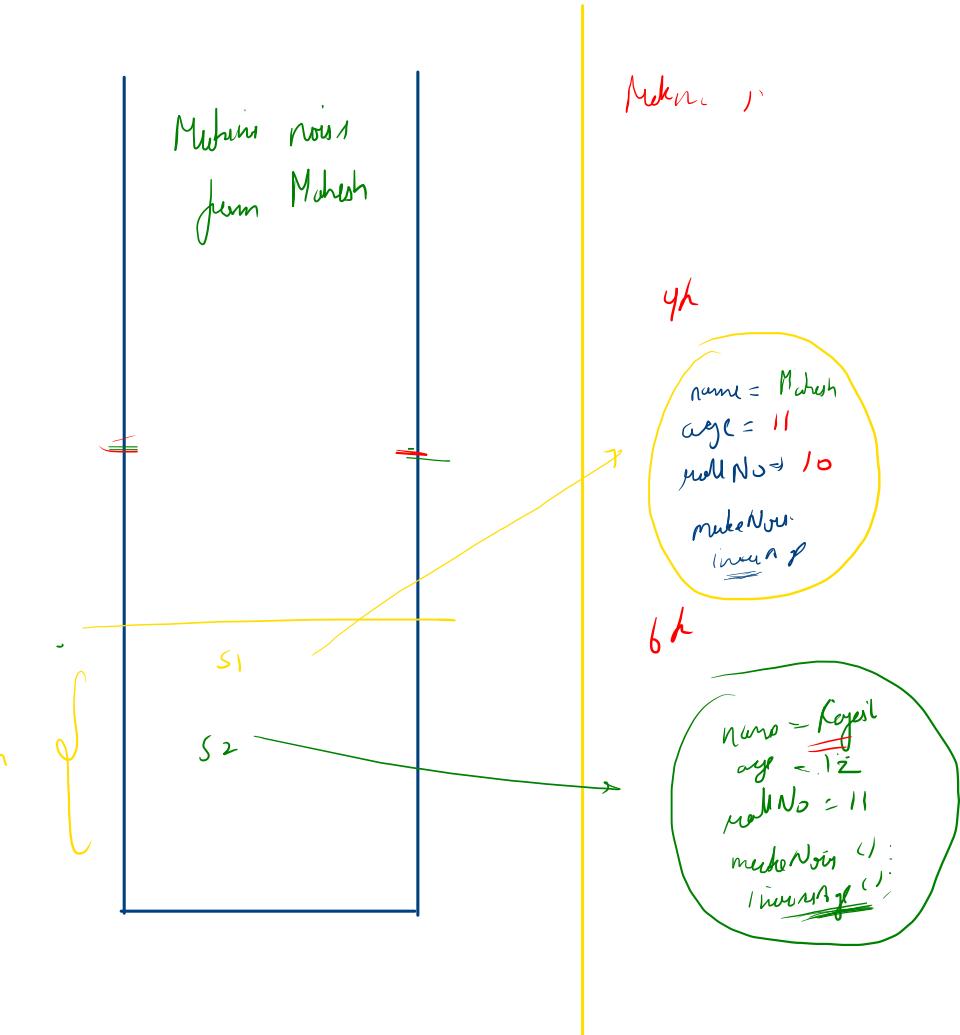
public class Main {
    Run | Debug
    public static void main(String[] args) {
        Student s1 = new Student();
        s1.name = "Mahesh";
        s1.age = 11;
        s1.roll_no = 10;

        Student s2 = new Student();
        s2.name = "Rajesh";
        s2.age = 12;
        s2.roll_no = 11;

        // System.out.println("Properties of s1 -> " + s1.name+" "+s1.age+" "+s1.roll_no);
        // System.out.println("Properties of s2 -> " + s2.name+" "+s2.age+" "+s2.roll_no);

        s1.makeNoise();
        s2.makeNoise(); → Making noise from Rajesh
    }
}

```

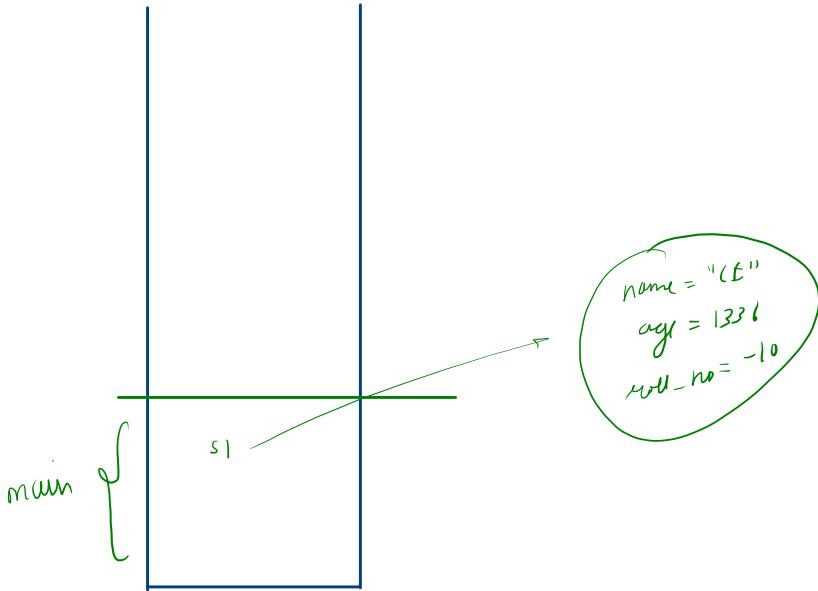


→ DR Y  
↳ don't repeat yourself.

#

## Constructors

function inside your class with same name as 'Class' and no return type.



```
class Student {
    String name = "CE";
    int age = 1336;
    int roll_no = -10;
}

public class ConstructorExample {
    public static void main(String[] args) {
        Student s1 = new Student();
    }
}
```

As soon as we create a constructor, default constructor is deleted.

## # Types of Constructors

default

↙  
no input arguments

```
public Student() {
    // logic
}
```

Parameterized

↙  
arguments / parameters

```
public Student(---)
```

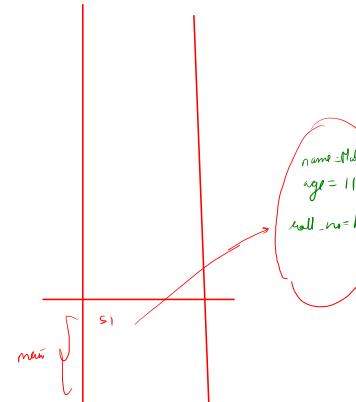
}

```
class Student {
    String name = "CE";
    int age = 1336;
    int roll_no = -10;

    // default constructor
    public Student(){
        System.out.println("My constructor is called");
    }

    // parameterized constructor
    public Student(String newName, int newAge, int newRoll_no){
        name = newName;
        age = newAge;
        roll_no = newRoll_no;
    }

    public class ConstructorExample {
        Run [Debug]
        public static void main(String[] args) {
            Student s1 = new Student();
            Student s2 = new Student("Mahesh", 11, -10);
            System.out.println("Student Object with properties "+s1.name+" "+s1.age+" "+s1.roll_no);
        }
    }
}
```

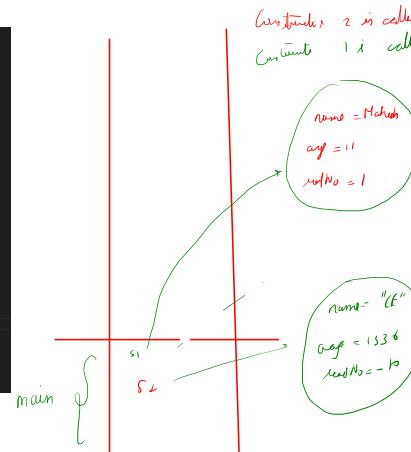


```
class Student {
    String name = "CE";
    int age = 1336;
    int roll_no = -10;

    // default constructor
    public Student(){
        System.out.println("Constructor 1 is called");
    }

    // parameterized constructor
    public Student(String newName, int newAge, int newRoll_no){
        System.out.println("Constructor 2 is called");
        name = newName;
        age = newAge;
        roll_no = newRoll_no;
    }

    public class ConstructorExample {
        Run [Debug]
        public static void main(String[] args) {
            Student s1 = new Student();
            Student s2 = new Student("Mahesh", 11, -10);
            System.out.println("Student Object with properties "+s1.name+" "+s1.age+" "+s1.roll_no);
            System.out.println("Student Object with properties "+s2.name+" "+s2.age+" "+s2.roll_no);
        }
    }
}
```



# this  
✓

it refers to  
attributes of class

Class Student {  
String name;  
int age;

public void fun (int age){  
age++;

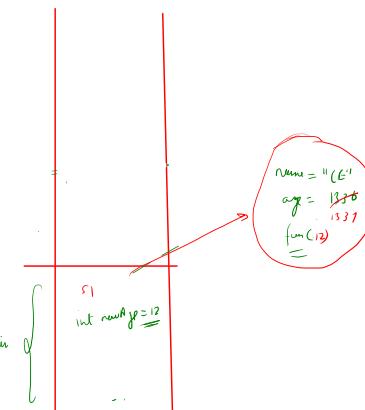
s1 → name = "CE"  
age = 1326

scope of variable

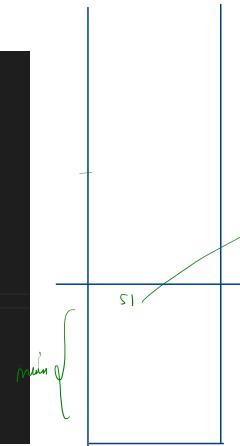
```
class Student {  
    String name = "CE";  
    int age = 1336;  
  
    public void fun(int age){  
        age++;  
        System.out.println(age);  
    }  
}  
  
public class ThisKeywordExample {  
    Run | Debug  
    public static void main(String[] args) {  
        Student s1 = new Student();  
  
        s1.fun(age: 123);  
  
        System.out.println(s1.age);  
    }  
}
```

```
class Student {  
    String name = "CE";  
    int age = 1336;  
  
    public void fun(int newAge){  
        age++;  
    }  
}  
  
public class ThisKeywordExample {  
    Run | Debug  
    public static void main(String[] args) {  
        Student s1 = new Student();  
  
        s1.fun(newAge: 123);  
  
        System.out.println(s1.age);  
    }  
}
```

```
class Student {  
    String name = "CE";  
    int age = 1336;  
  
    public void fun(int newAge){  
        newAge++;  
        age++;  
    }  
}  
  
public class ThisKeywordExample {  
    Run | Debug  
    public static void main(String[] args) {  
        Student s1 = new Student();  
  
        s1.fun(newAge: 12);  
        System.out.println(newAge);  
        System.out.println(s1.age);  
    }  
}
```



```
class Student {  
    String name = "CE";  
    int age = 1336;  
  
    public void fun(int age){  
        this.age++;  
    }  
  
    // public Student(String name, int age){  
    //     this.name = name;  
    //     this.age = age;  
    // }  
}  
  
public class ThisKeywordExample {  
    Run | Debug  
    public static void main(String[] args) {  
        Student s1 = new Student();  
  
        s1.fun(age: 100);  
  
        System.out.println(s1.age);  
        // System.out.println(s1.name+ " "+s1.age);  
    }  
}
```



```
class Student {  
    String name; // ←  
    int age; // ←  
  
    public void fun(int age){  
        this.age++;  
        System.out.println(age);  
        System.out.println(this.age);  
    }  
  
    public Student(String name, int age){  
        this.name = name;  
        this.age = age;  
    }  
}  
  
public class ThisKeywordExample {  
    Run | Debug  
    public static void main(String[] args) {  
        Student s1 = new Student(name: "Accio", age: 23);  
  
        // s1.fun(-100);  
  
        // System.out.println(s1.age);  
        System.out.println(s1.name+ " "+s1.age);  
    }  
}
```

## # Constructor Chaining

constructor 1 is called

- ✓ Constructor 2 is called
- ✓ Constructor 3 is called

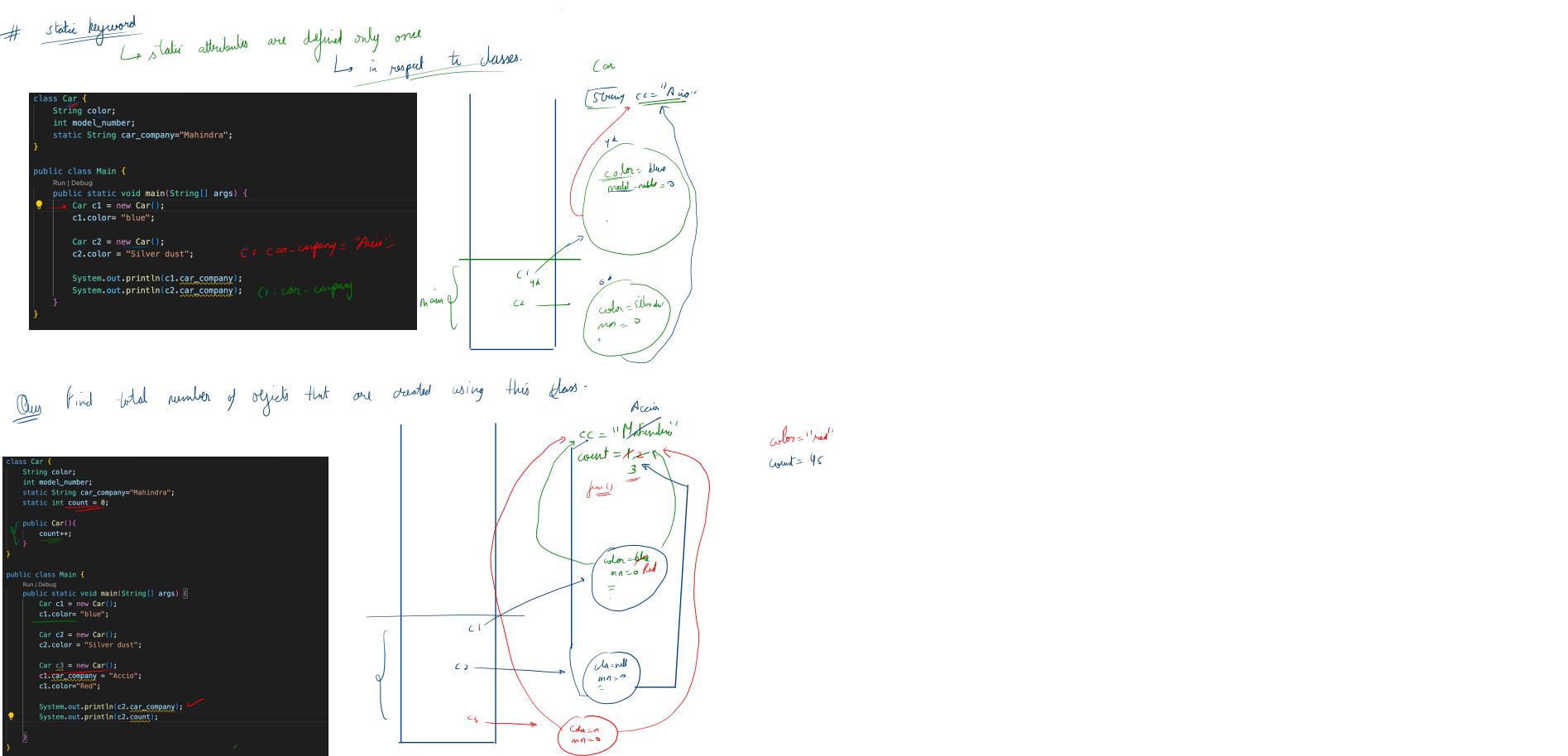
✓ name = Accio ✓  
✓ age = 12  
✓ roll\_no = 234

new (Accio, 12)  
↓  
(Accio, 12, 234)

```
class Student {  
    String name;  
    int age;  
    int roll_no;  
  
    public Student(){  
        System.out.println("Default constructor is called");  
    }  
    public Student(String name){  
        this.name = name;  
        System.out.println("constructor 1 is called");  
    }  
  
    public Student(String name, int age){  
        this.name = name;  
        this.age = age;  
        System.out.println("constructor 2 is called");  
    }  
  
    public Student(String name, int age, int roll_no){  
        this(name,age);  
        this.roll_no = roll_no;  
        System.out.println("constructor 3 is called");  
    }  
  
}  
  
public class ConstructorChaining {  
    public static void main(String[] args) {  
        Student s3 = new Student(name: "Accio", age: 12, roll_no: 234);  
    }  
}
```

## # Rules

- 1) Constructor calling should be first line
- 2) Ordering doesn't matter.
- 3) In the chain, there should be at least one constructor with no constructor call;



- Ques Find total number of objects that are created using this class -
- ```

class Car {
    String color;
    int model_number;
    static String car_company="Mahindra";
    static int count = 0;

    public Car(){
        count++;
    }
}

public class Main {
    Run | Debug
    public static void main(String[] args) {
        Car c1 = new Car();
        c1.color= "blue";

        Car c2 = new Car();
        c2.color = "Silver dust";

        Car c3 = new Car();
        c3.car_company = "Accio";
        c3.color=Red;

        System.out.println(c2.car_company);
        System.out.println(c2.count);
    }
}

```
- 

1) static properties & functions belong to classes  
 2) static functions can only change static variables. (why)  
 ↳ non static functions can change static, non-static variables.  
 3) static functions can only call other static functions  
 4) we can access static properties through class name

```

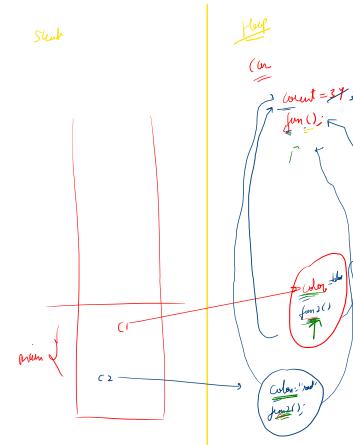
class Car {
    String color;
    static int count=34;

    public static void fun(){
        color = "red";   fun()
        count = 39;      ↳ you cannot call non-static
    }                      fun from static functions.

    public void fun2(){
        color = "red";  ↳
        count = 44;     ↳
    }
}

public class Example {
    Run | Debug
    public static void main(String[] args) {
        Car c1 = new Car();
    }
}

```



## # Pillar of OOPS

1) Inheritance

2) Encapsulation

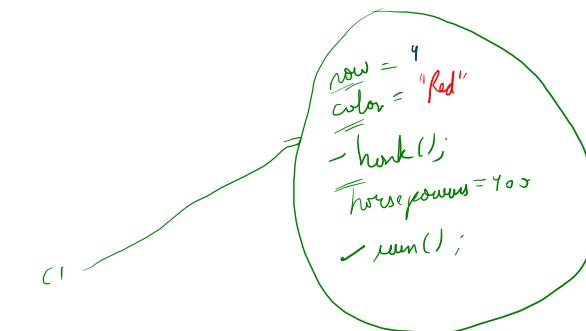
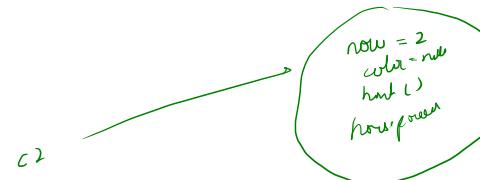
3) Abstraction

4) Polymorphism

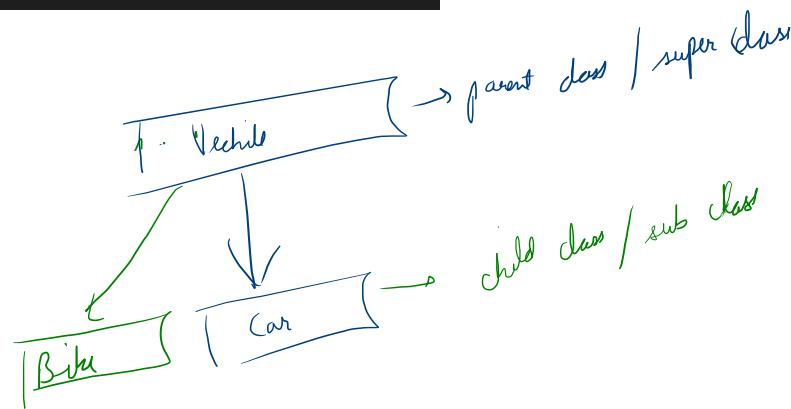
## # Inheritance

DRY  
↓  
Code reusability

```
class Vechile {  
    int now = 2; // number of wheels  
    String color;  
  
    public void honk(){  
        System.out.println("Honking from Vechile class");  
    }  
  
    class Car extends Vechile {  
        int horsePowers;  
  
        public void run(){  
            System.out.println("Running from child class");  
        }  
  
        public class Main {  
            Run | Debug  
            public static void main(String[] args) {  
                Car c1 = new Car();  
                c1.now = 4;  
                c1.color = "Red";  
                c1.horsePowers = 400;  
  
                c1.honk();  
                c1.run();  
            }  
        }  
    }  
}
```

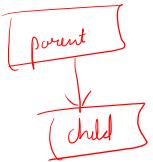


Parent class constructor is called  
first, then child class

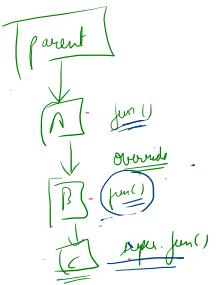


## # Type of Inheritance

### 1) Single Inheritance



### 2) Multilevel Inheritance



## # super keyword

↳ to access parent class attributes.

```

class Vehicle {
    String color;
    int seats;

    public Vehicle(){
        // System.out.println("Default constructor");
    }

    public Vehicle(String color, int seats){
        System.out.println("Parametrized constructor");
        this.color = color;
        this.seats = seats;
    }

    public void VehicleHonk(){
        System.out.println("Honking from the vehicle class");
    }
}
  
```

```

class Car extends Vehicle {
    int engine=47;

    public void VehicleHonk(){
        System.out.println("Honking from the Car class but vehicle");
    }

    public void CarHonk(){
        System.out.println("Honking from the car!!!!!!!");
        this.VehicleHonk();
    }

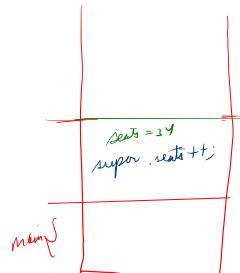
    public void doSomething(int engine){
        this.engine++;
    }

    public void doSomethingAgain(int seats){
        super.seats++;
    }
}
  
```

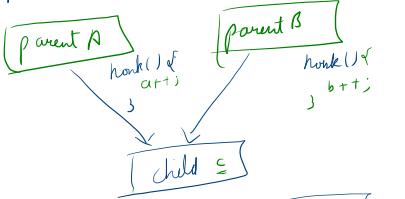
```

public class Example {
    Run | Debug
    public static void main(String[] args) {
        Car c1 = new Car();
        c1.seats = 4;
        c1.doSomethingAgain(seats: 34);
        System.out.println(c1.seats);
    }
}
  
```

"this"

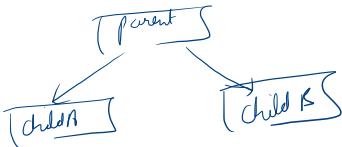


### 3) Multiple Inheritance (not possible in Java)

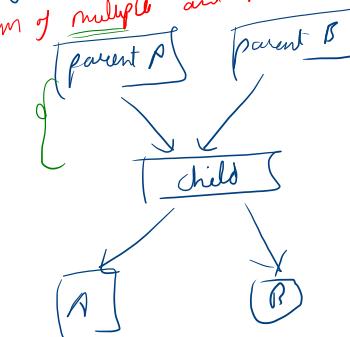


c.honk():  
ambiguity  
reduce complexity

### 4) Hierarchical Inheritance



### 5) Hybrid (not in Java)

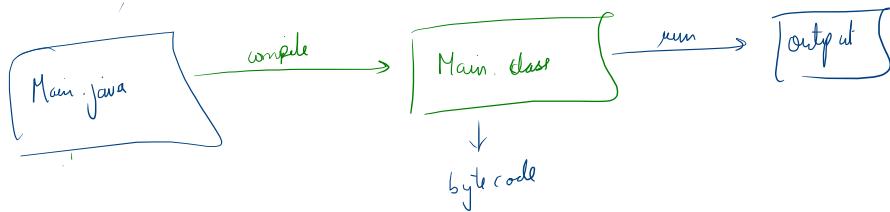
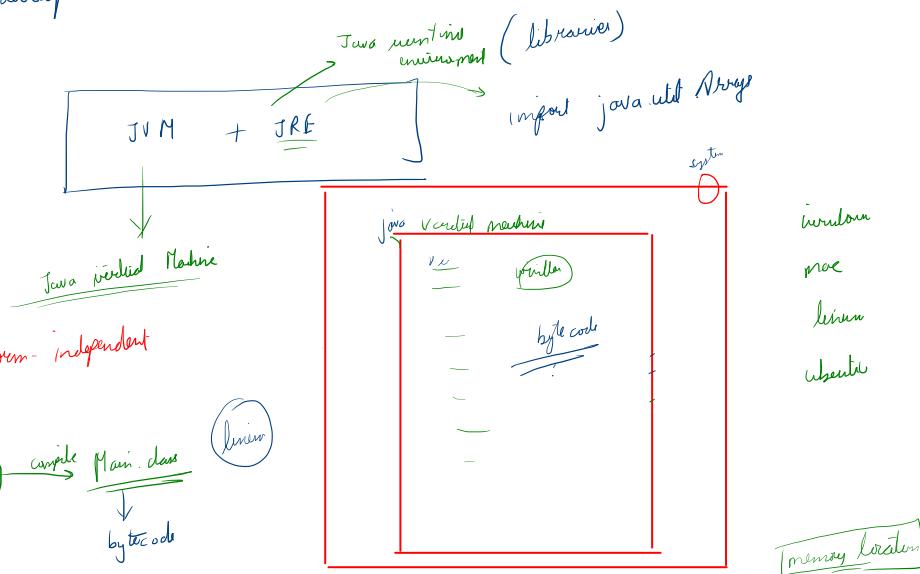


↳ extend car super.Vehicle + this

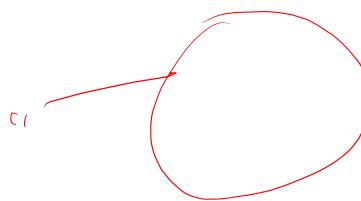
color = null  
seats = 45  
VehicleHonk()  
engine()  
VehicleHonk()  
CarHonk()  
seats  
45

JDK

Java development kit



✓ (or c1 = new Coal();  
✓ c1. weightMark(s);



## # Polymorphism

Poly → many

morph → join

- 1) Method overloading

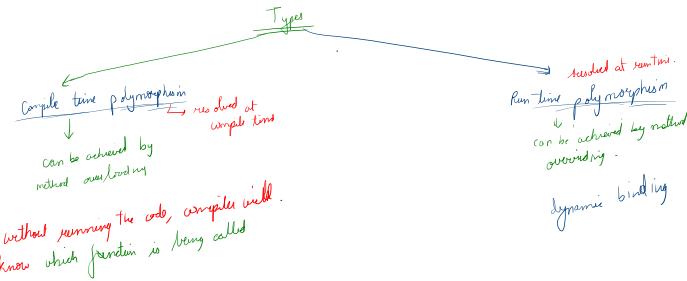
  - change number of parameters.
  - change the type of parameters.

2) Method overriding

we override the parent class function

↓

no change in parameters.



```
class Vehicle {
    String color;
    int seats;

    public void VehicleColor(){
        System.out.println("Working from the vehicle class");
    }

    public int addInts(int a, int b){
        return a+b;
    }
}

class Car extends Vehicle {
    int enginePower;

    // method overriding
    public void VehicleColor(){
        System.out.println("Working from the Car class");
    }

    // method overriding
    public int addInts(int a, int b){
        return a + 1 + b + 1;
    }
}

public class Example {
    public static void main(String[] args) {
        Car c = new Car();
        c.VehicleColor(); // 5% visible at compile time
        // upcasting
    }
}
```

FUD

every car is Vehicle

(Car) -> new (Car), -> a  
Car() -> new (Car)

Vehicle V = new Vehicle

L + R = C(p)

Inheritance  
 Vehicle v = new Car()  
 Vehicle v = new Vehicle()  
 ✓ add(1, 2)  
 ↳ overrides  
 function called  
 ① ↳ vec11  
 ↳ (car)  
 car  
 Vehicle

↳ dynamic binding  
late binding

parent → display 

~~# Packages~~

↳ group of similar files

↓  
built-in, user defined packages

import java.util.\*;

package set;

## # Access Modifiers

1) private

↳ It can be accessed  
only within the class.

3) protected

↳ accessible within package

↳ outside package  
↳ using child class

2) public

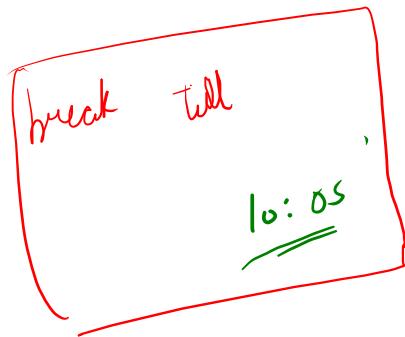
↳ this can be accessed by  
any one  
↳ sub class

4) default

↳ if no modifier, default  
↳ accessible within  
package  
cannot be accessed outside  
the package

# Encapsulation  
↳ hiding / wrapping of data into a single unit.

↳ is achieved by using "private" keyword.



final  
wrapper class  
copy constructor

# Abs<sup>r</sup>action  
↳ hiding of implementation  
↳ showing only function name

↳ can be achieved by abstract class / Interface



=) Ab<sup>r</sup>act classes  
↳ classes having ab<sup>r</sup>act methods with keyword ab<sup>r</sup>act ↳ no implementation

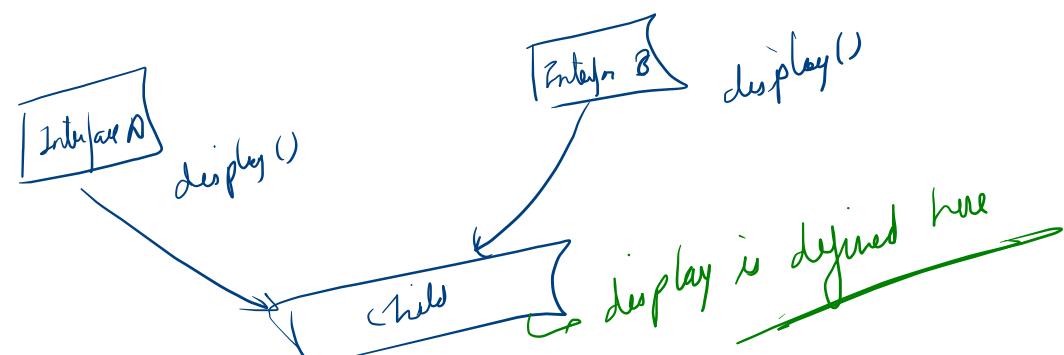
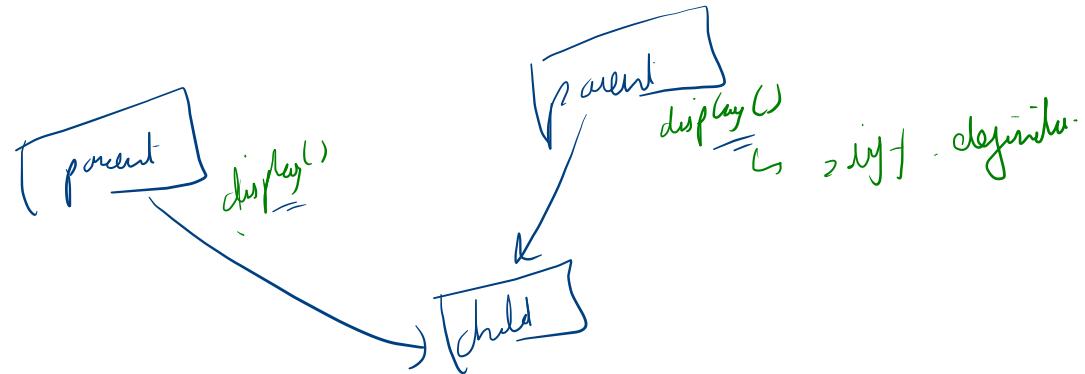
⇒ can have - constructor, static methods

⇒ It won't let us achieve 100% ab<sup>r</sup>action.

Interface → you can have only abstract methods.  
↳ they by default will add public and abstract

100% abstract

Is multiple inheritance possible in Java?  
↳ possible using interface



# final key word → You can not change it.

↳ class  
↳ method  
→ variable

attribute ↗ method  
→ variable

final class → cannot inherit it

final method → cannot override

final Variable → cannot be changed to some other value once defined.