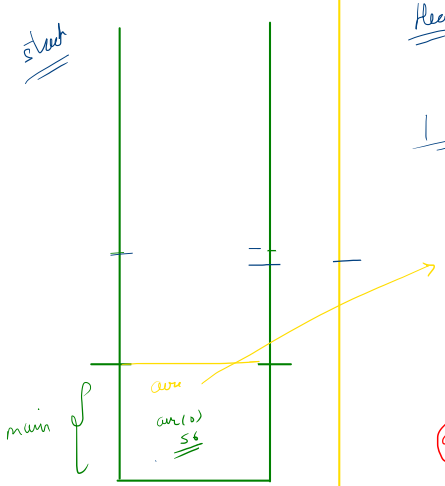
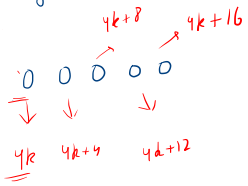
Memory Mapping



Heap

1 int → 4 bytes

20 bytes



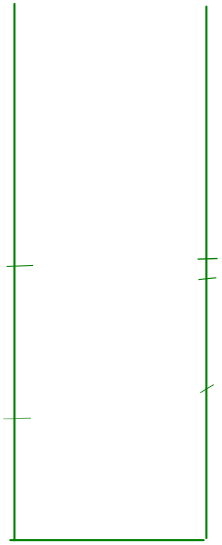
arr → address of 0th variable

arr[idx]

arr[3] = 4k+12

arr[i] → base address + 5*4

```
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){
        int[] arr = new int[5];
        increaseArray(arr);

        //int a = 2;
        //int b = 5;

        //swap(a,b);
        //System.out.println(a+" "+b);
}
```



garbage - collected

OOPs → Object Oriented Programming

Machine language → assembly languages → BASIC → COBOL → Compiler

Compiler

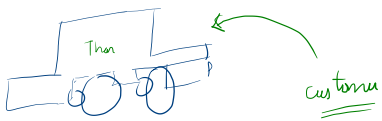
C → purely functional language

C++ → C with OOP features

Java → Object oriented

Car

Thar
- color = blue
- engines → 2 engine
- wheels → alloy wheels
- lights → UV
- seats → 2 seats



1 page

blueprints
- lights
- wheels
- color
- seats
- engine

use → Thar → taking some real entity

one

automatic side mirrors → (ele ... mirrors)

automatic side mirrors

blue print → Thar 1, Thar 2

Thar 1 → Thar close mirror

⇒ Class → a blueprint

properties / attributes
methods / functions

```
class Car {
    → String color;
    → int seats;
    → String engine;
}
```

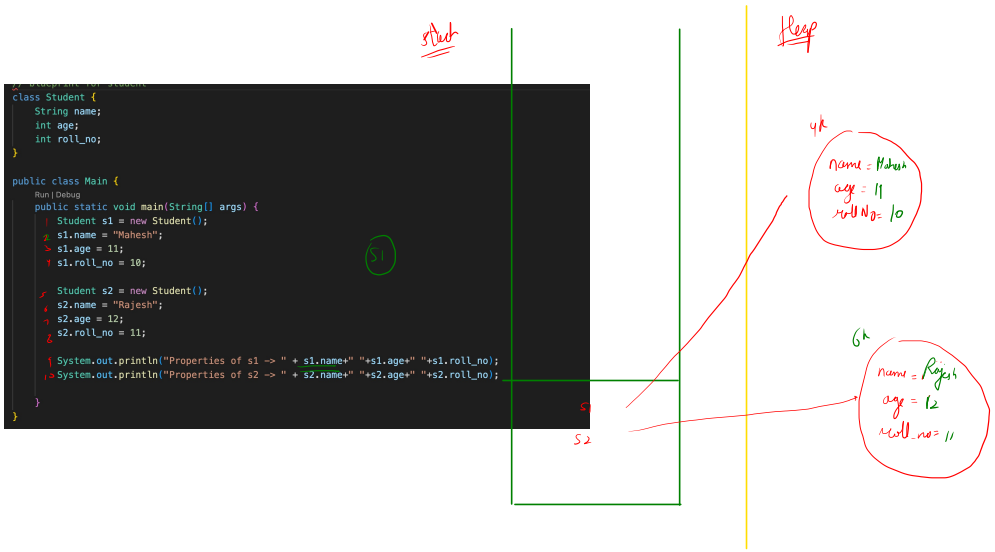
name_of_class name_of_object = new Name_of_class

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

Objects
↓
Object

```
Car c1 = new Car();
c1.color = "Red";
c1.seats = 3;
c1.engine = "v4";
System.out.println(c1.color)
```

```
Student s1 = new Student();
s1.name = Mahesh
s1.rollNo = 1
s1.age = 11;

Student s2 = new Student();
s2.name = "Rakesh"
s1.rollNo = 2;
s2.age = 12;
```



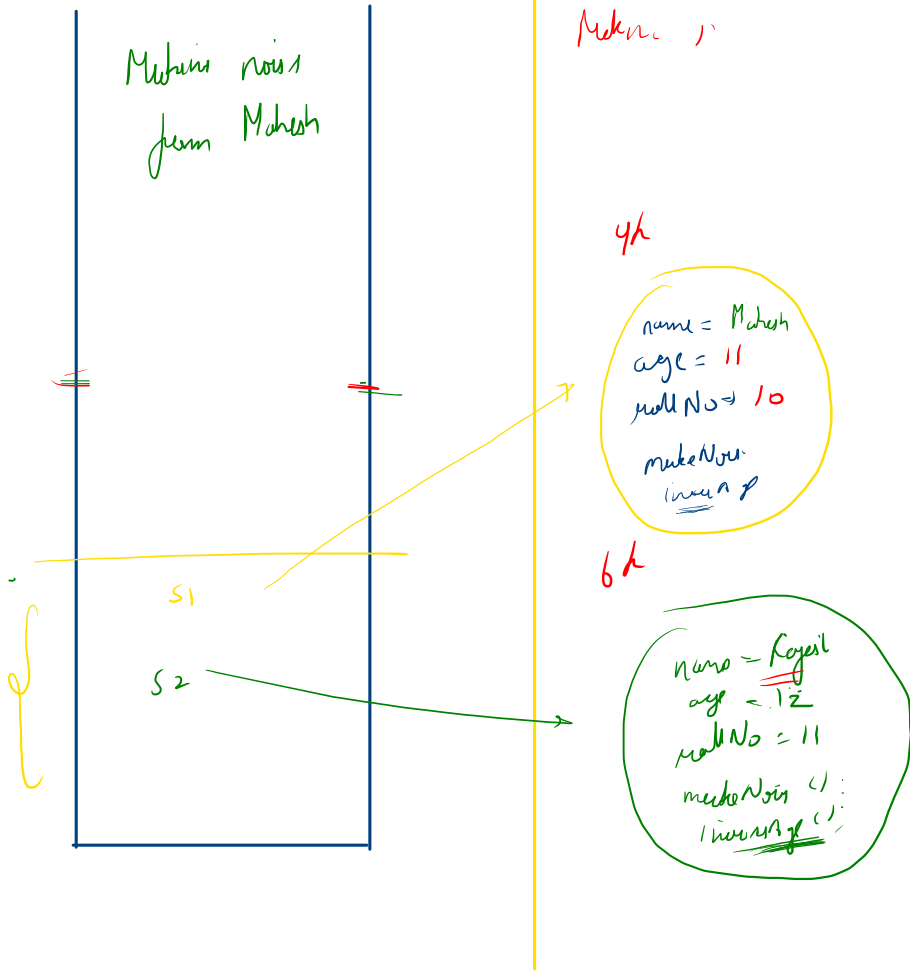
Arraylist <Integer> al = new Arraylist<>();

Arraylist < String > al = new Arraylist <>

Arraylist< E > al =

```java
class Student {
    String name;
    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

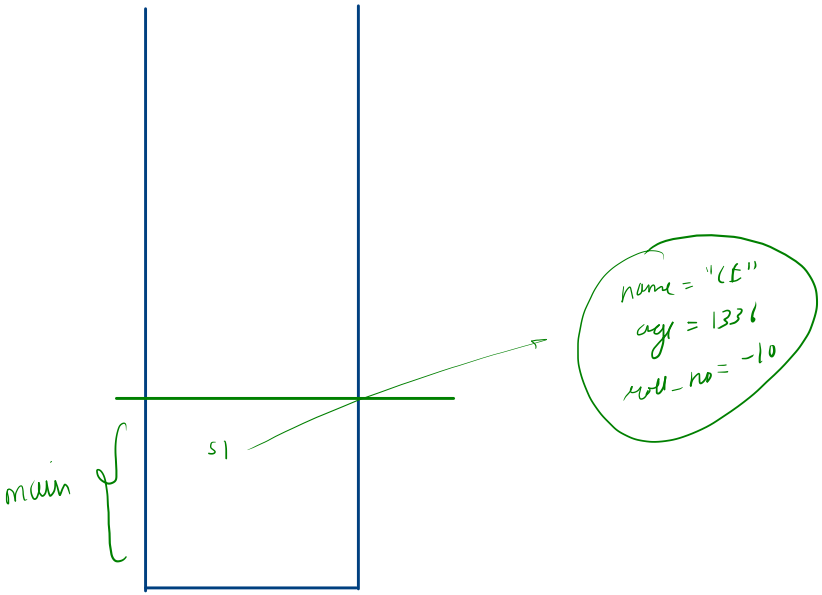


→ DRY

↳ don't repeat yourself.

Constructors

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




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

public class ConstructorExample {
    Run | Debug
    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

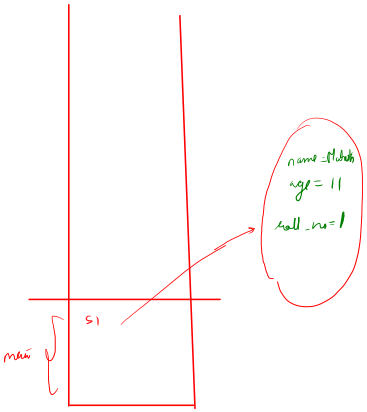
```
public Student() {
    // logic
}
```

Parameterized

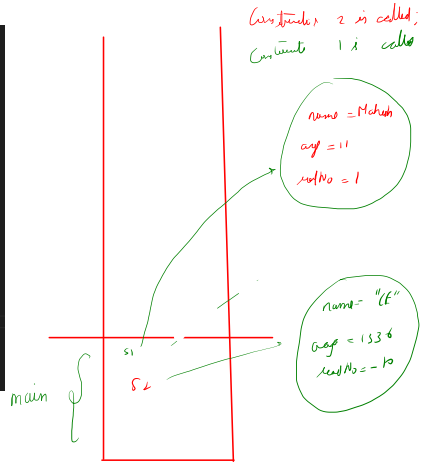
arguments / parameter

```
public Student(_ _ _ _) {

}
```

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

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

    // parametrized 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(ne[redacted]e: "Mahesh",n[redacted]: 11,n[redacted]: 1);
        System.out.println("Student Object with properties "+s1.name+" "+s1.age+" "+s1.roll_no);
    }
}
```




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

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

    // parametrized constructor
    public Student(String newName, int newAge, int newRoll_no){
        System.out.println(x: "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([redacted]e: "Mahesh",[redacted]: 11,[redacted]: 1);
        Student s2 = new Student();

        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++;
}
}

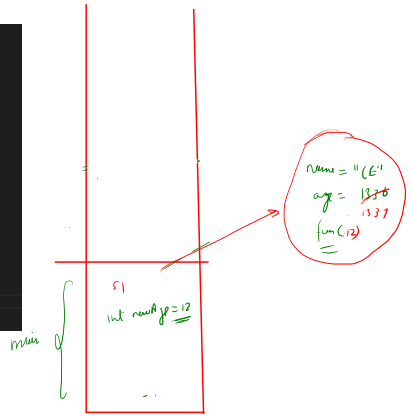
s1 → name = "CE"
age = 1336

Scopes of variable

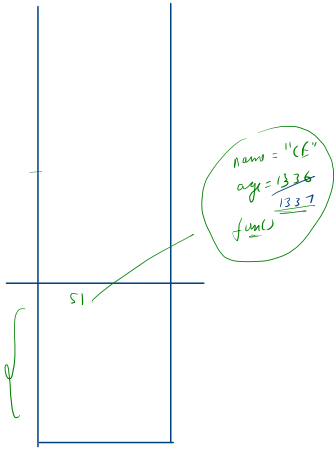
```java
class Student {
    String name = "CE";
    int age = 1336;

    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);
    }
}
```

```java
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);
    }
}
```

```java
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();

        int newAge = 12;
        s1.fun(newAge);

        System.out.println(newAge);
        System.out.println(s1.age);
    }
}
```




```java
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);
    }
}
```



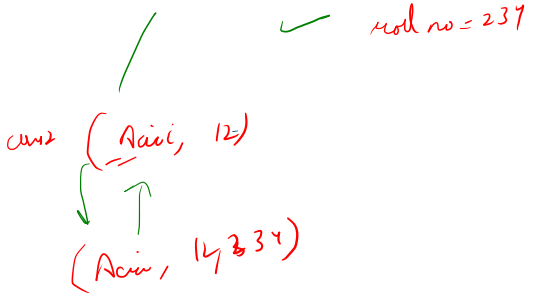

```java
class Student {
    String name = "CE";
    int age = 1336;

    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

```java
class Student {
    String name;
    int age;
    int roll_no;

    public Student(){
        System.out.println(x: "Default constructor is called");
    }
        OOPS.Introduction.Student.Student(String name)
    public Student(String name){
        this.name = name;
        System.out.println(x: "constructor 1 is called");
    }

    public Student(String name, int age){
        this.name = name;
        this.age = age;
        System.out.println(x: "constructor 2 is called");
    }

    public Student(String name, int age, int roll_no){
        this(name,age);
        this.roll_no = roll_no;
        System.out.println(x: "constructor 3 is called");
    }
}

public class ConstructorChaining {
    Run | Debug
    public static void main(String[] args) {
        Student s3 = new Student(name: "Accio",age: 12,roll_no: 234);
    }
}
```

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 atleast one constructor with no constructor call

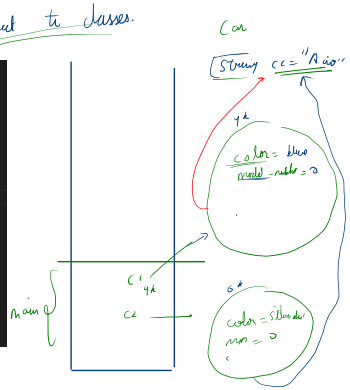
static keyword

↳ static attributes are defined only once
↳ in respect to classes.

```java
class Car {
    String color;
    int model_number;
    static String car_company="Mahindra";
}

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";

        System.out.println(c1.car_company);
        System.out.println(c2.car_company);
    }
}
```

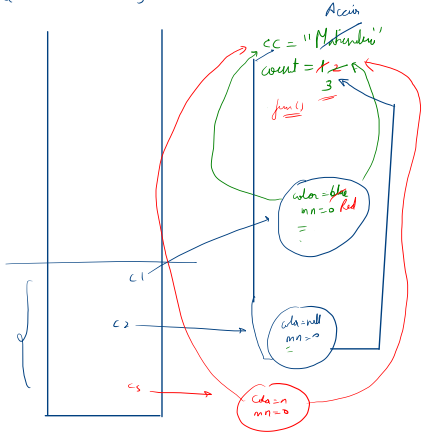
c2.car_company = "Accio"

c1.car_company



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

```java
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();
        c1.car_company = "Accio";
        c1.color="Red";

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



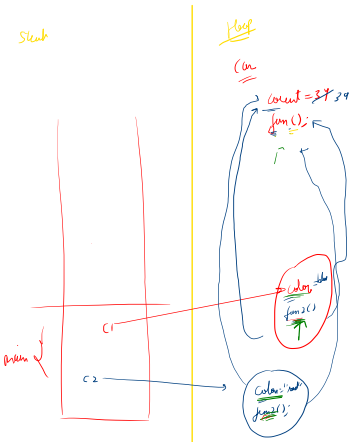
color = "red"
count = 45

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 ClassName;

Main main();
Example main();

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

    public static void fun(){
        color = "red";
        count = 39;
    }

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

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

fun() → you cannot call non-static fun() from static functions.



Pillar of OOPs

1) Inheritance

2) Encapsulation

3) Abstraction

4) Polymorphism

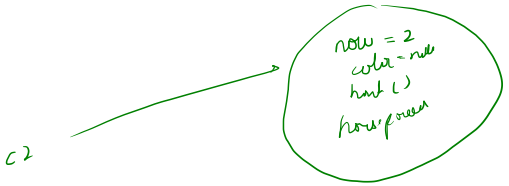
Inheritance

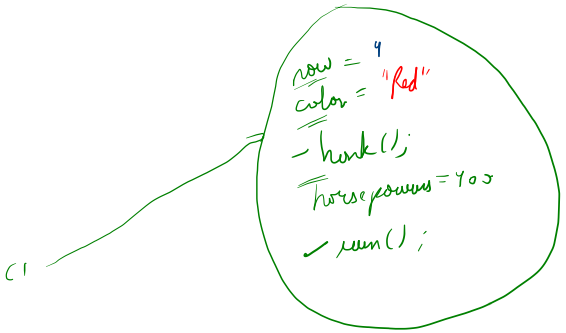
DRY

↓

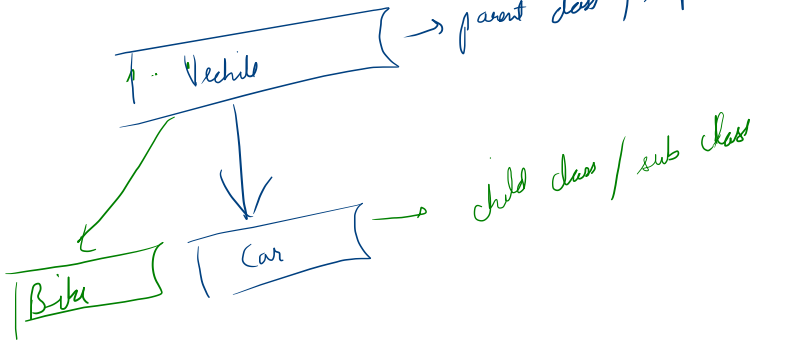
code reusability

```java
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();
    }
}
```



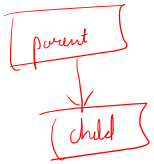


Parent class constructor is called first, then child class

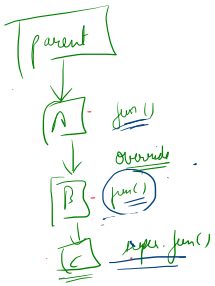


Type of Inheritance

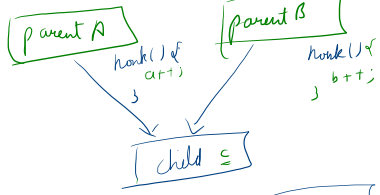
1) Single Inheritance



2) Multilevel Inheritance



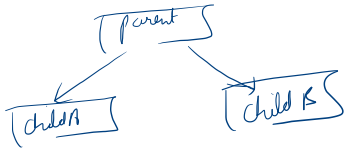
3) Multiple Inheritance (not possible in Java)



ambiguity
reduce complexity

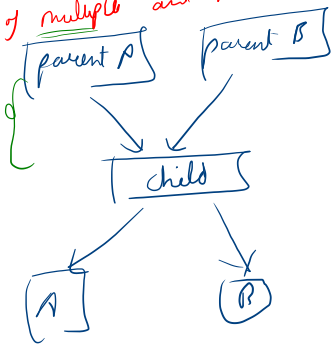
c.hunk();

4) Hierarchical Inheritance



5) Hybrid (not in Java)

sum of multiple and hierarchical



super keyword

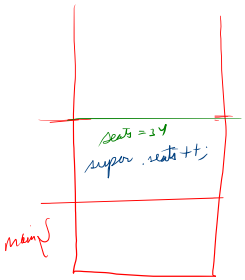
↳ to access parent class attributes.

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

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

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

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

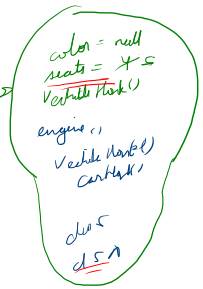
```java
public class Example {
    Run | Debug
    public static void main(String[] args) {
        Car c1 = new Car();
        c1.seats = 4;
        c1.doSomethingAgain(seats: 34);
        System.out.println(c1.seats);
    }
}
```

```java
class Car extends Vehicle {
    int engine=47;

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

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

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

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

"this"



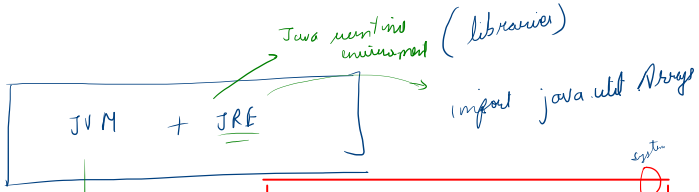
c1



by extend Car super.Vehicle() that

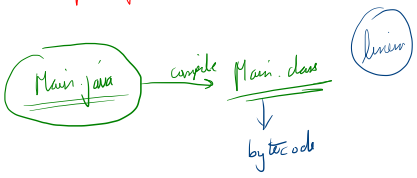
JDK

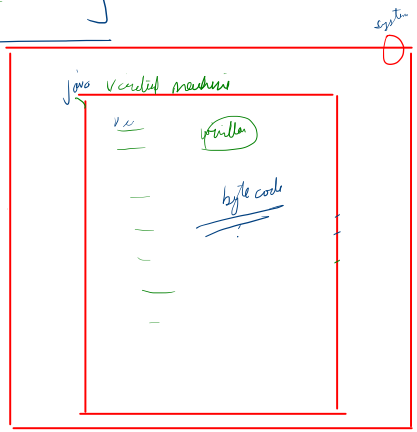
↳ Java development kit



Java Verified Machine

Java is platform-independent

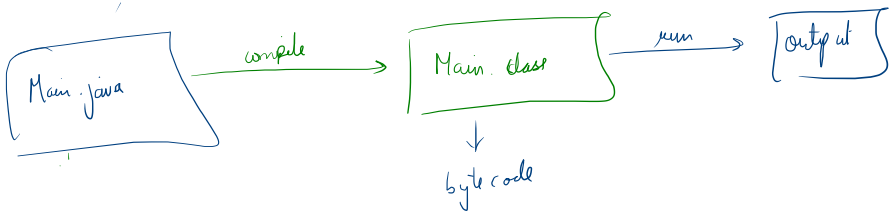




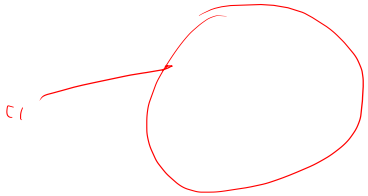
windows
mac
linux
ubuntu

memory location

Java don't have pointers



Car c1 = new Car();
c1.VechicleMark(5);



Polymorphism

Poly + morph

Poly → many

morph → form

1) Method overloading
- → change number of parameters.
- → change data type of parameters.

2) Method overriding
we override the parent class function
↓
no change in parameters.

Types

Compile time polymorphism → resolved at compile time
↓
can be achieved by method overloading

Run time polymorphism → resolved at run time.
↓
can be achieved by method overriding.

without running the code, compiler will know which function is being called

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

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

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

class Car extends Vehicle {
    int engine=47;

    // method overLoading
    public void Vehiclehonk(int a){
        System.out.println("Honking from the Car class");
    }

    @Override // method overRiding
    public int add(int a, int b){
        return a + 1 + b + 1;
    }
}

public class Example {
    Run | Debug
    public static void main(String[] args) {
        Car c1 = new Car();
        c1.Vehiclehonk(5); // this is visible at compile time

        // upcasting
        // Vehicle v = new Vehicle();
        Vehicle v = new Car();
        v.add(2,3); //
```

new Car

→ every car is Vechile

Car c1 = new Car(); ... all vechile attributes
Car c1 = new Car()

Vechile V = new Vechile();
↳ no car properties

// upcasting
Vechile V = new Car();
// Vechile V = new Vechile();

V.add(1, 2)
↳ overridden function called.