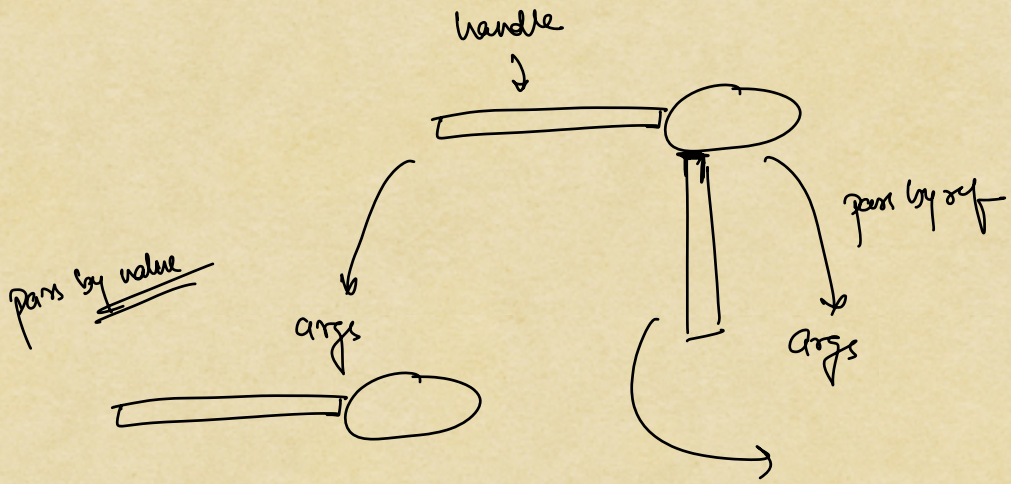
Agenda:

i) Pass By Value / Pass By Reference
ii) Static
iii) Final
iv) Abstract class
v) Associations

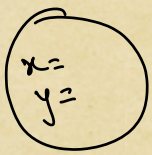
⇒ Pass by Value / Pass by references



Java ⇒ Pass by value

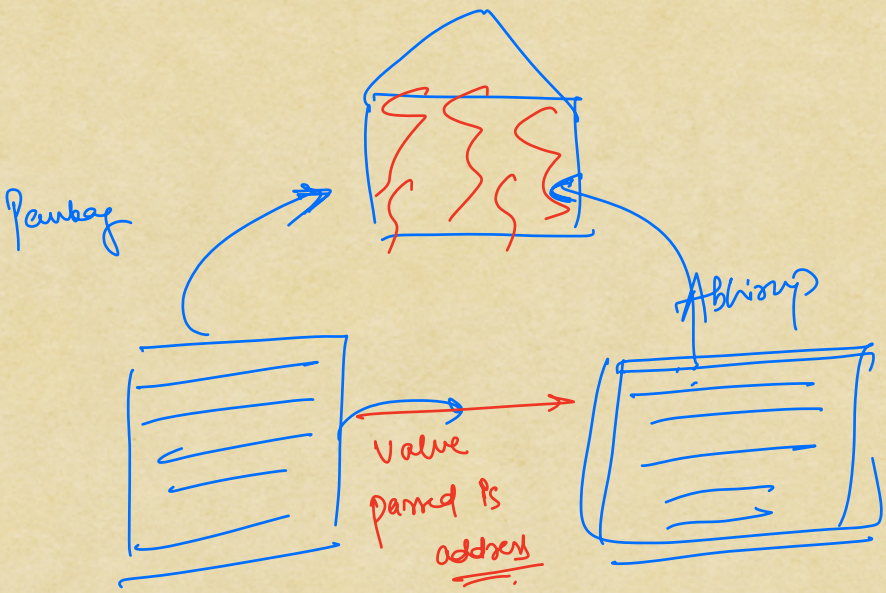
Whatever argument we pass to a method, it passed by creating a new copy of the attribute as value.

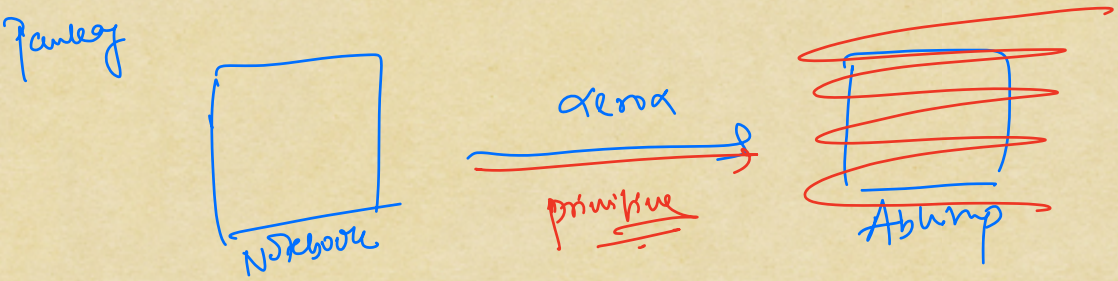
```
psvm() {
    int x = 10;
    int y = 20;
    swap(x, y);        // 10 20
    print(x, y);
}

swap(int x, int y) {   // [x=10, y=20]
    int temp = x
    x = y              // x = 20
    y = temp           // y = 10
}
}

Point {
    x, y
}
psvm() {
    Point p = new Point();
    p.x = 10;
    p.y = 20;
    swap(p)   <-- p holds ref/address to the Object
    print(p.x, p.y);
```



```
}
swap( Point p ) {
    int temp = p.x
    p.x = p.y
    p.y = temp
}
}
```

a copy of address is being passed

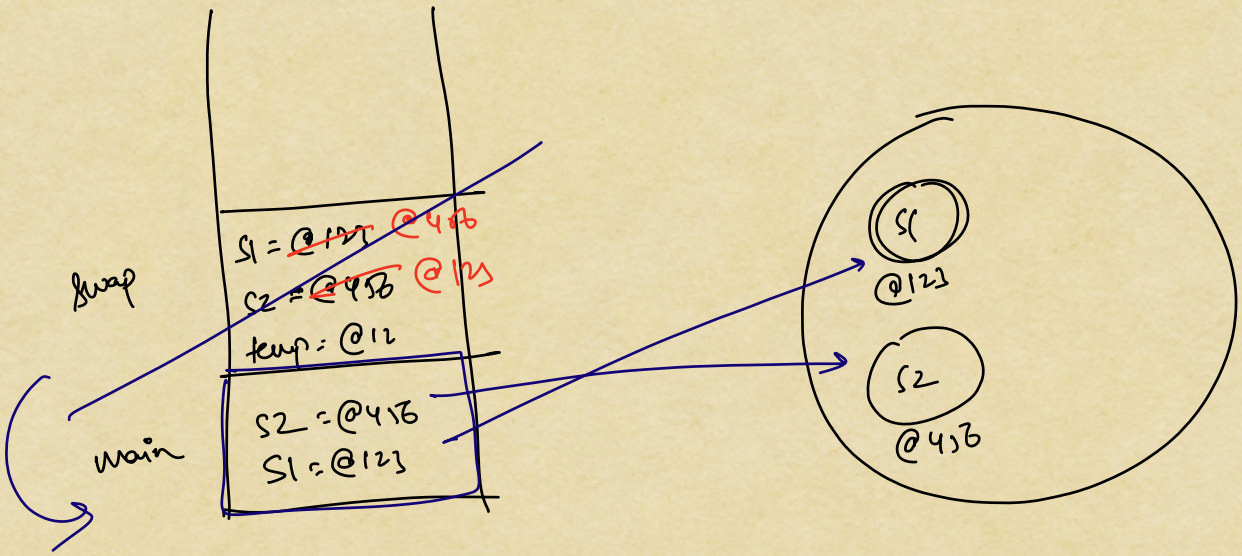




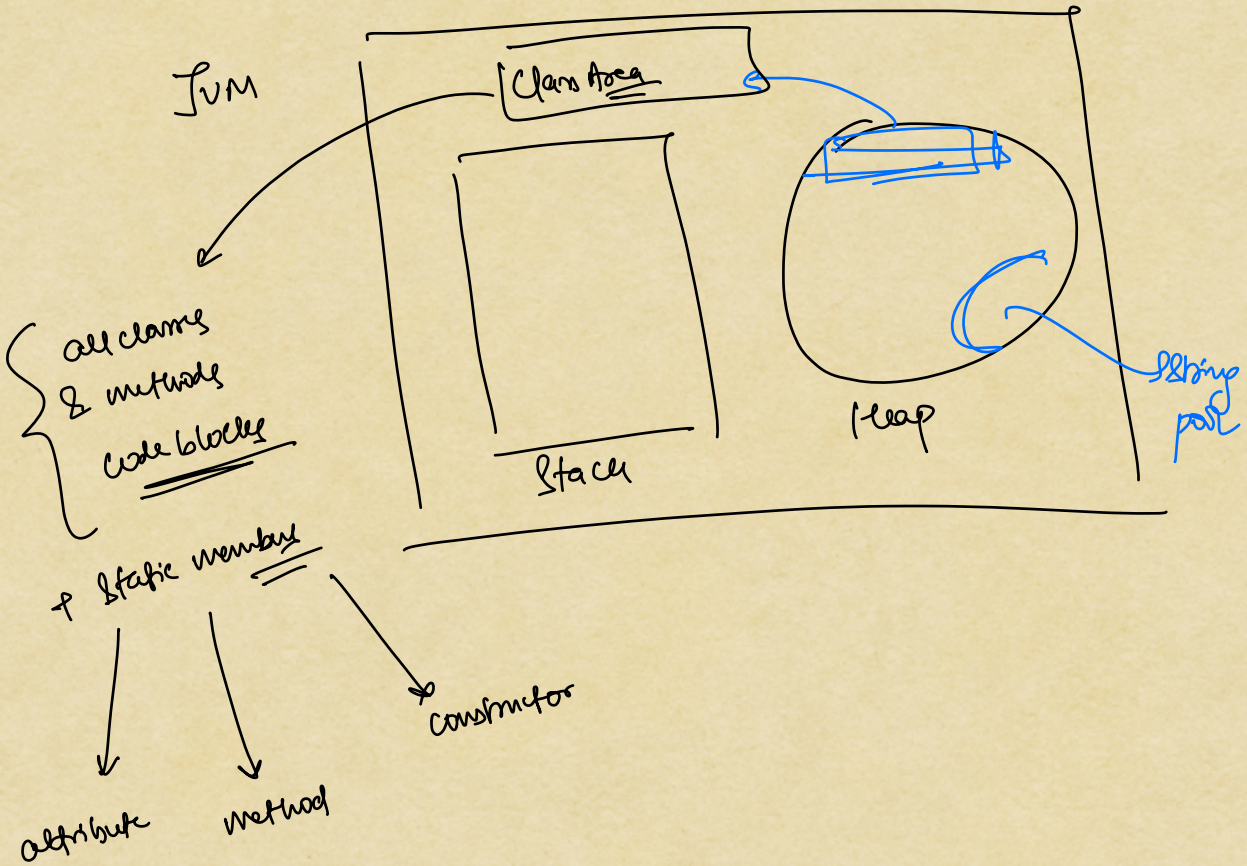
```
psvm() {
    Student s1 = new Student();
    Student s2 = new Student();

    swap(s1, s2);
    s1.display();
}
```

```
swap(Student s1, Student s2) {
    Student temp = s1
    s1 = s2
    s2 = temp
}
```

```
int tempage = s1.age
s1.age = s2.age
s2.age = tempage
```

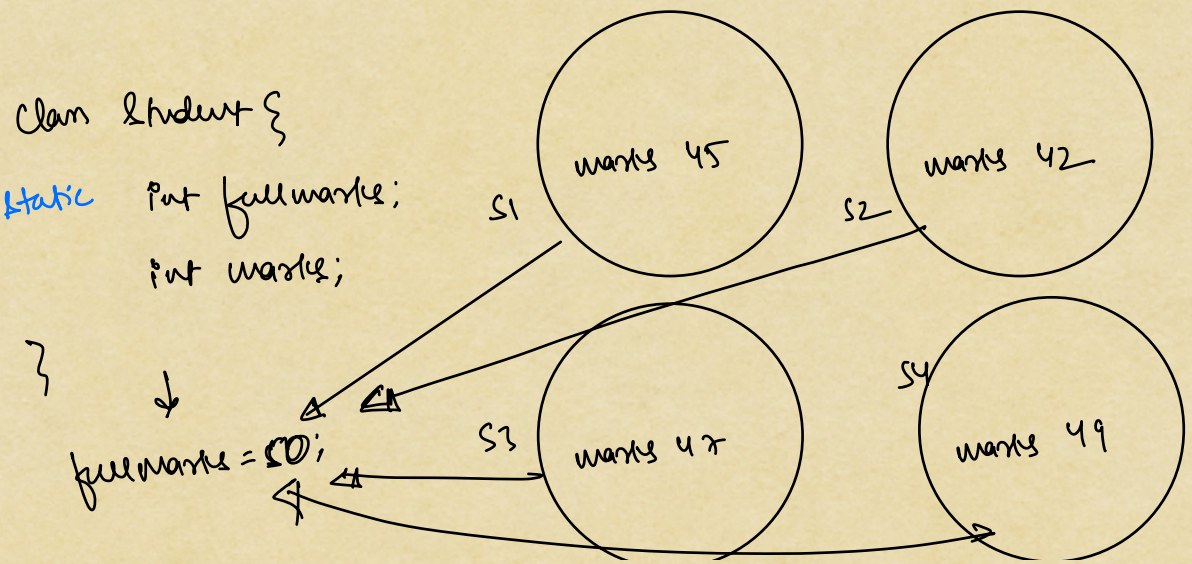


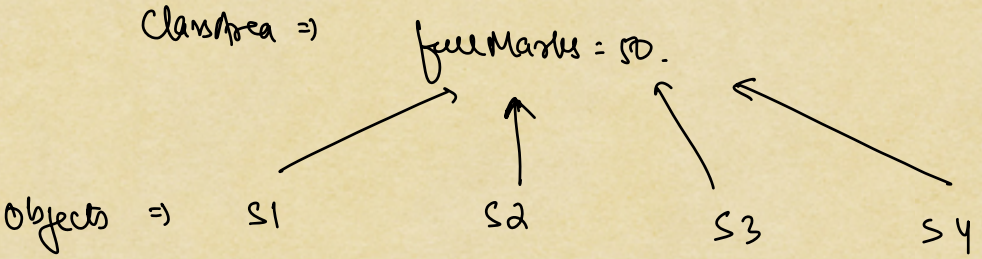
⇒ Static Keyword



° Static attribute :-

attributes of a class level





- Static methods

  - methods which are present at class levels

→ Utility methods

```
Class Utils {
    ...
}
```

→ Call method without creating object

⇒ Static methods

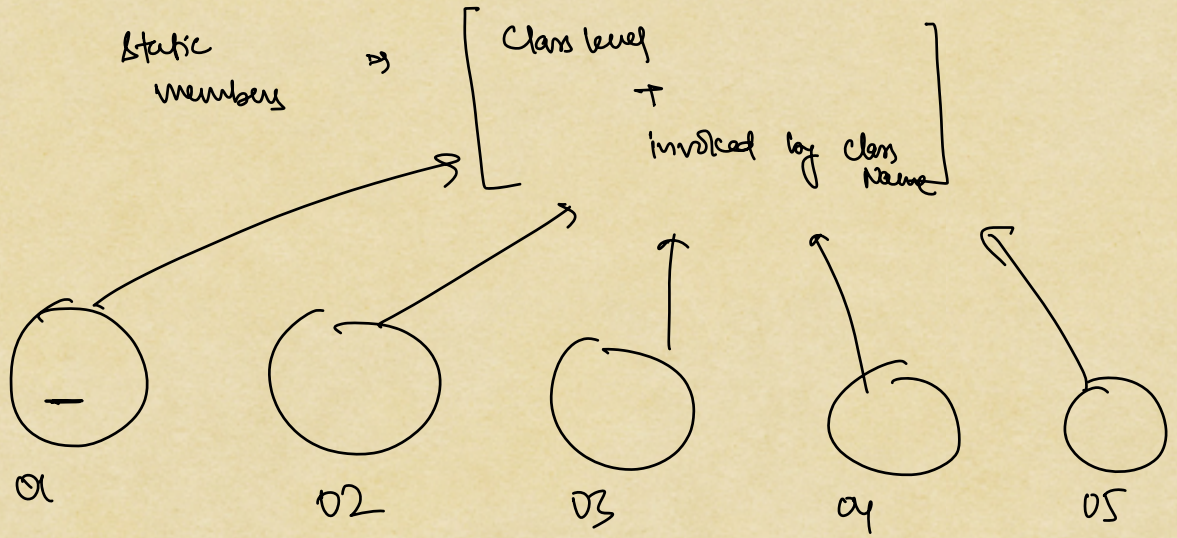
→ Always a static member (attribute/method) is called/invoked using class reference [class name]

Student.staticMethod() Student s = new Student();

```
S.StaticAttribute = 10;
S.staticMethod();
```

works but not a good practice

Internally, Java fetches class name reference



Object level attributes & methods → Inside each object in heap, invoked by objectName

	Static attribute or methods	Object level: Non-static attribute or methods
non-static method →	Yes	Yes
Static method →	Yes	No.

Static method always called using className, but non-static variable need

Object name]

⇒ Overridding of Static methods :

* Yes

* polymorphism ⇒ NO

[parent ref → child]

Student { teacher }

List<User>

printName(List<User> users) {

⇒ final keyword :

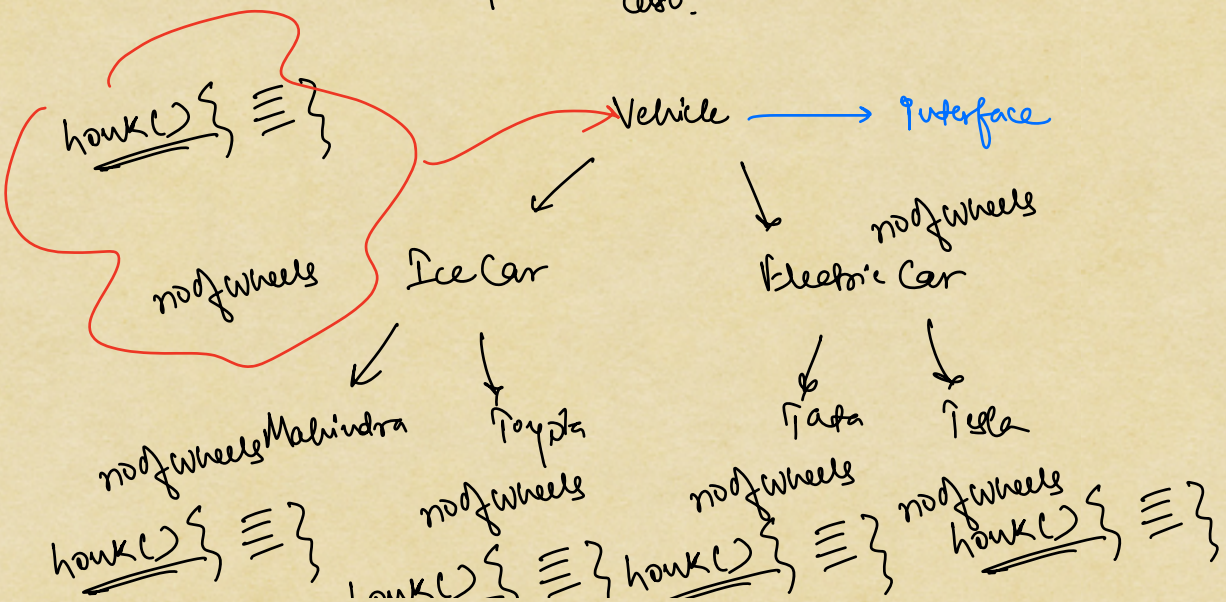
- attribute
  - it will constant once the object is created
- method
  - Can't override
- ClassName
  - can't extend it / can't have any child classes

⇒ Interface

- we want methods to be present
- we want each method to have its own implementation

abstract class

- we want methods to be present
- we want each method to have its own implementation
- we can have common attributes as well
- we can have some common methods also.



Vehicle → ~~interface~~ abstract class

→ Abstract class is a class that can contain abstract methods.

↓

ability

* Should an abstract class have atleast 1 abstract method ⇒ NO

* Can we create objects of an abstract class

⇒ NO

* Can abstract class have a constructor?

⇒ Yes

↳ but to initialise the variables

⇒ abstract class extend ← lose the ability to inherit

⇒ interface implement ← can still inherit another class

can implement multiple interfaces

class A extends B implements I1, I2 {

}

Abstract class X

class Y extends X

~~class Y extends X, 2~~

class Student {

int id,

int marks,

Address address

}

class Address {

}