LLD - 1 : Intro to OOP.

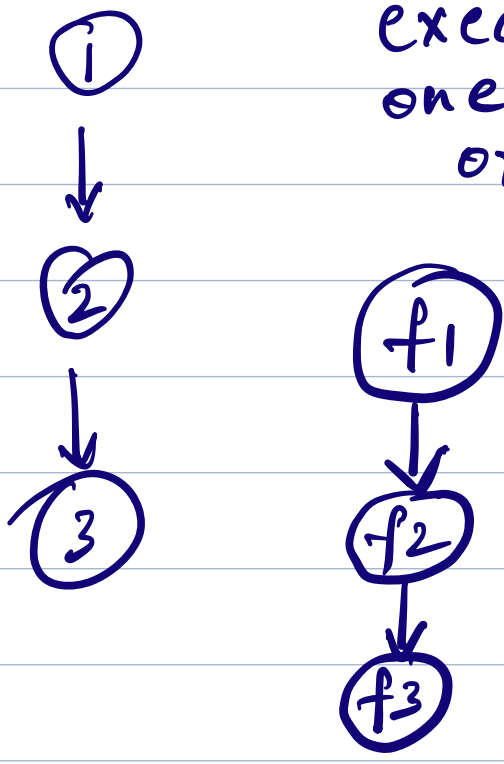
java.

1. 4 pillars of OOPs.
   - ↳ Abstraction
   - ↳ Encapsulation
2. Access modifiers.
   - ↳ public, private, default, protected.
3. Constructor
   - ↳ default
   - ↳ parameterised constructor
   - ↳ copy constructor < shallow copy. / deep copy.

1. Introduction to OOP.

programming paradigms.

- procedural.
  - functions.
    - set of steps
  - C.
    - instructions executed one after other.



main().

I want to print the psp of all the student in June 2024.

① is filter out students belonging to June 2024.

② to print the psp.

fetchStudents (batch b1)

↓ [1, 2, 3, 4, 5 ....]

printpsp (student s).

data.

printpsp ([name, age, psp])

printpsp(Students)

Object oriented.

entity.

↳ student. print ( );

procedural.

↳ C

printPSP(name, batch, age, psp)

↳ structures ξ → fluent.

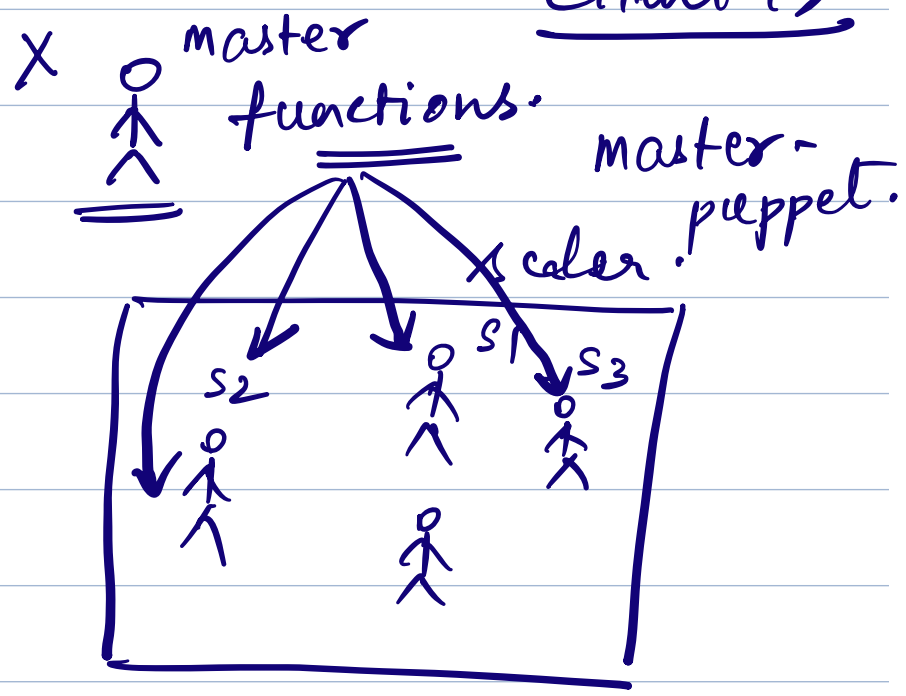
int age;

}

Object oriented.

student. print( );

client().



OOP.

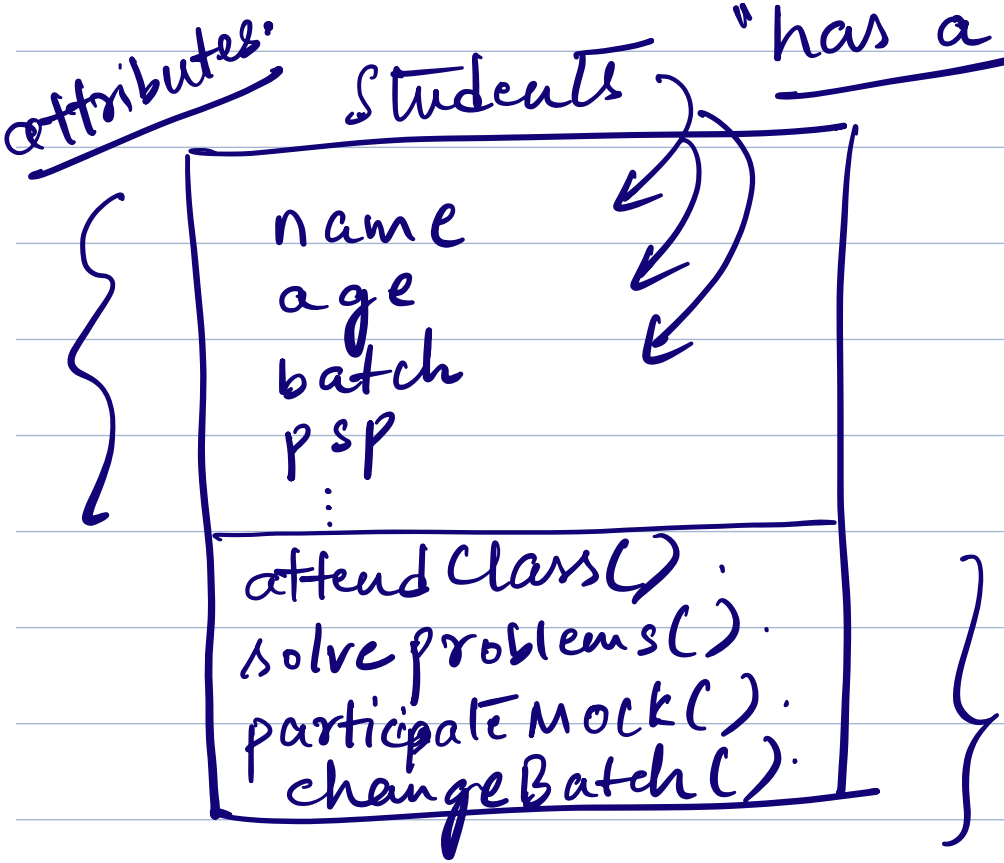
Software system adopted

mapped

OOP → entities → real world objects.

entities → properties / attributes ("has a")

entities → behaviours / methods / functions



4. Principles of OOP.

1. principle Abstraction. → principle. "being a good students"

- → Encapsulation.
- → Inheritance.
- → Polymorphism.

} 3 pillars.

1. Abstraction.

"Abstract" → "concentrating on the idea (entity) rather than the details".

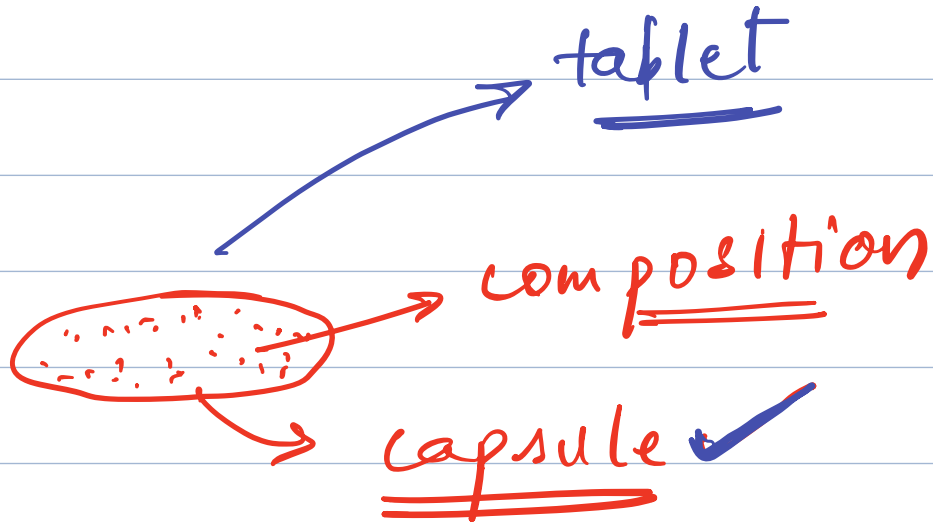
"Apply break to stop a moving car."

① complex system in terms of idea → entity · X attributes ! behaviours.

② users don't have to known the complete details.

② Encapsulation.

capsule.

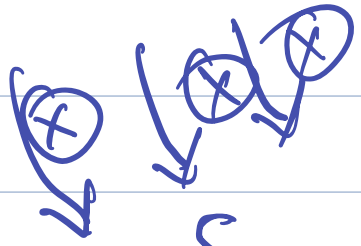


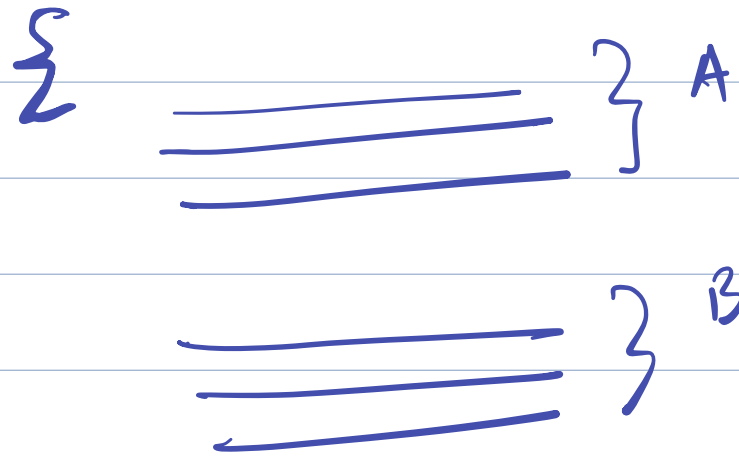
class

↳ ① Binding → binds attributes ⊕ behaviours } entity together.

↳ ② protection → protect attributes and behaviours from the

outside.





}

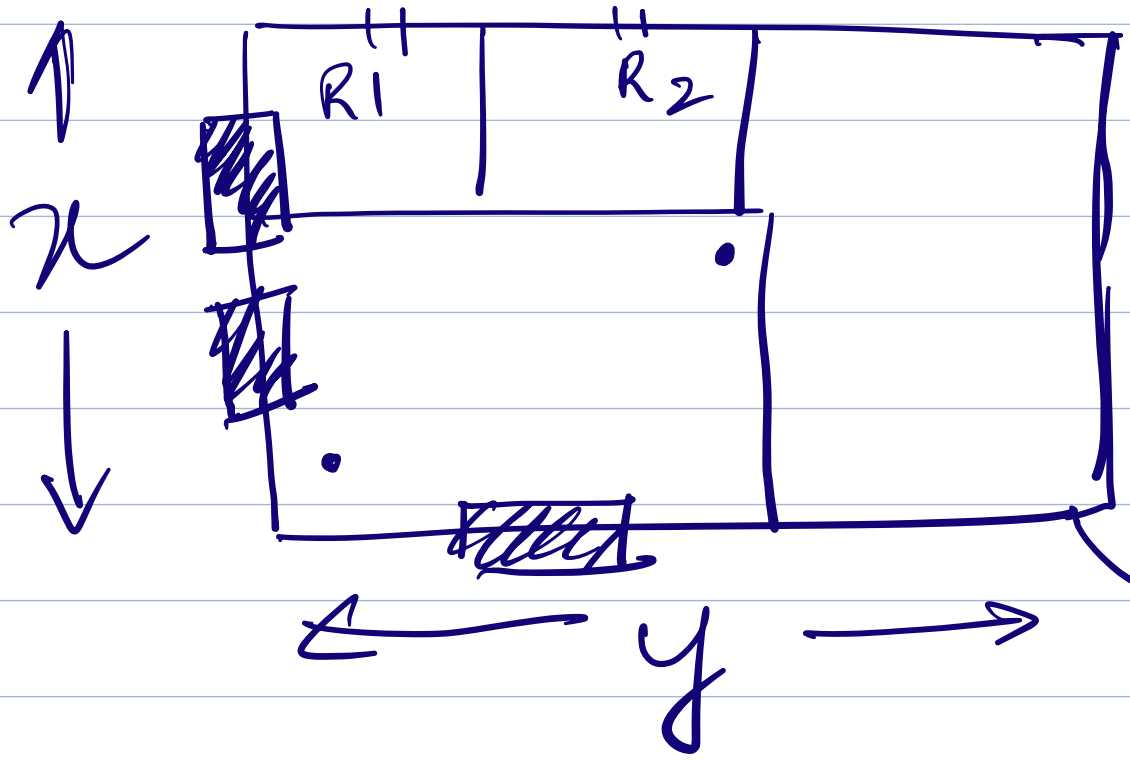
① Binding.

is achieved through classes

```
Student {
    String name;
    int age;
    double psp;

    printPSP() {    // behaviour
    }
}
```

attributes

Class → blueprint / plan of your entity.



↳ 4 people.
↳ like nature open spaces

blueprint.

```
student {
    String name;   } attributes
```

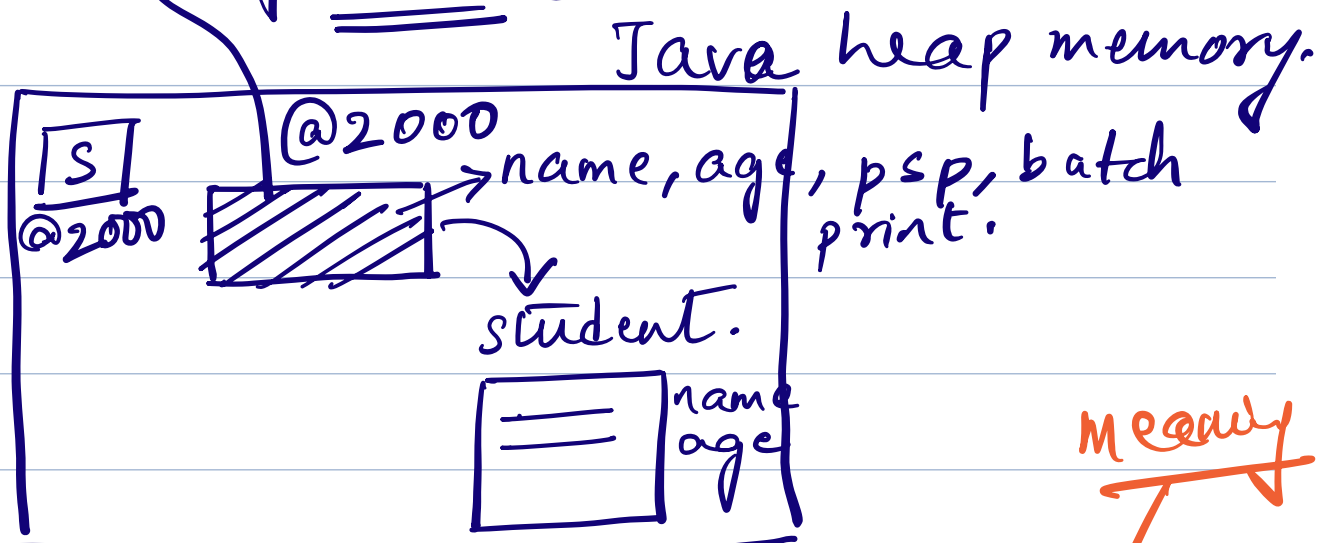
```
    int age;
    double psp;
}

printPSP() {    // behaviour
}
}
```

// create object

```
Student s = new Student();
```

s → @2000

reference (memory address).



custom data-type.

meaning

// creates object.

Student s = new Student();

datatype variable, reference to the object. datatype

// int i = 10;

datatype variable value.

print(s.age)

s.age = 30;

s.printPSP();

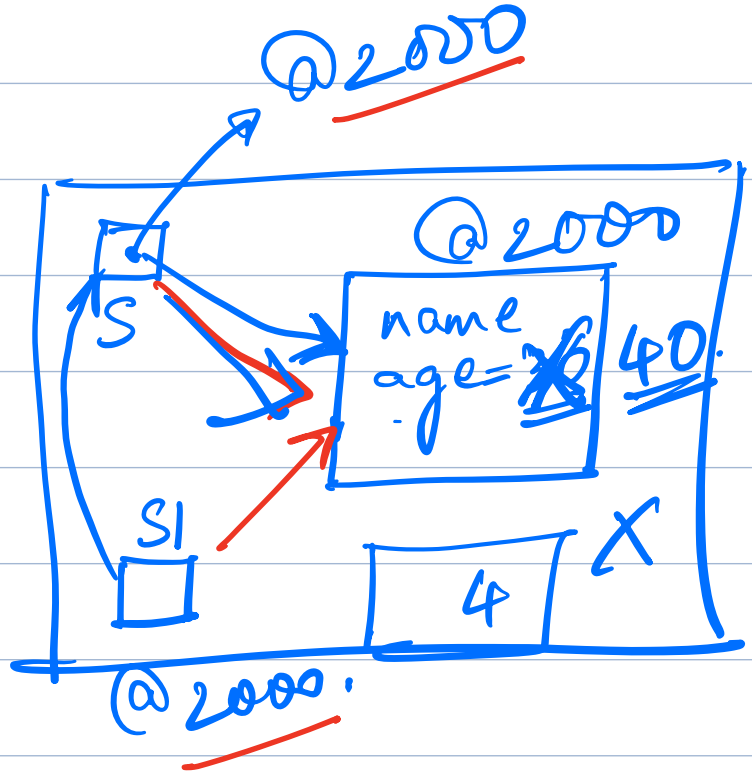
Q. Student s = new Student();

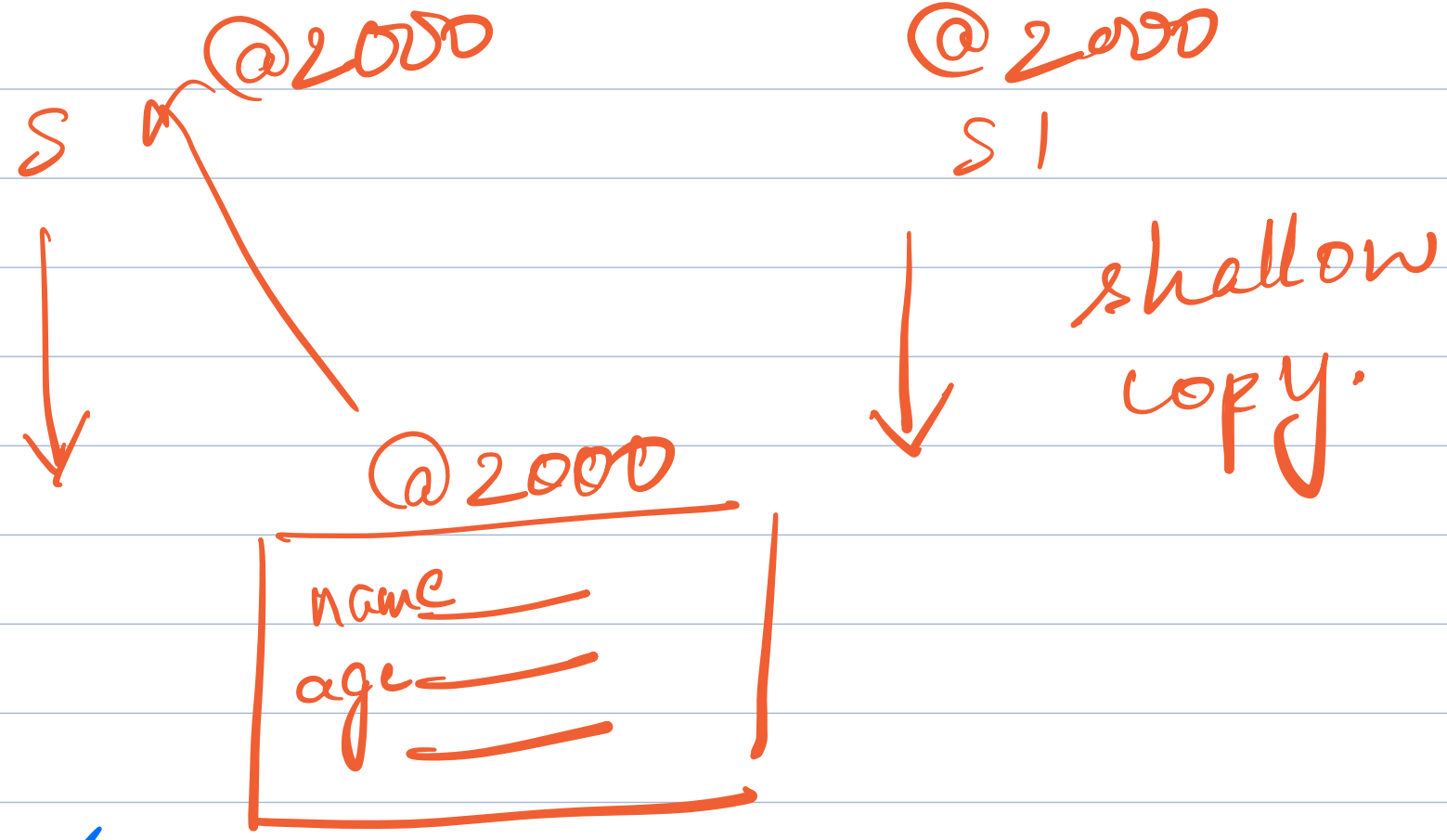
s.age = 30; ✓

shallow copy.

```
Student S1 = S;   // not using new keyword
S1.age = 40;
print(S.age);
```

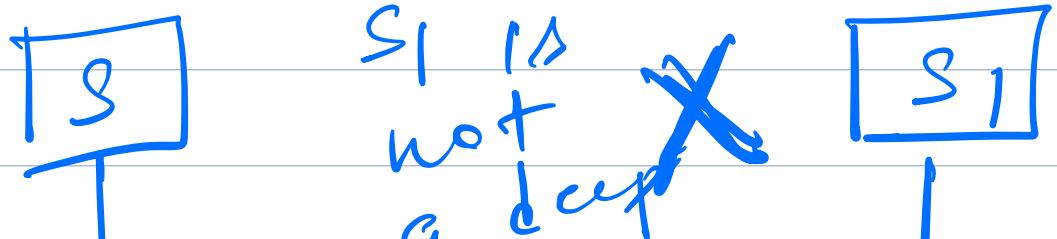
A. 30.
B. 40

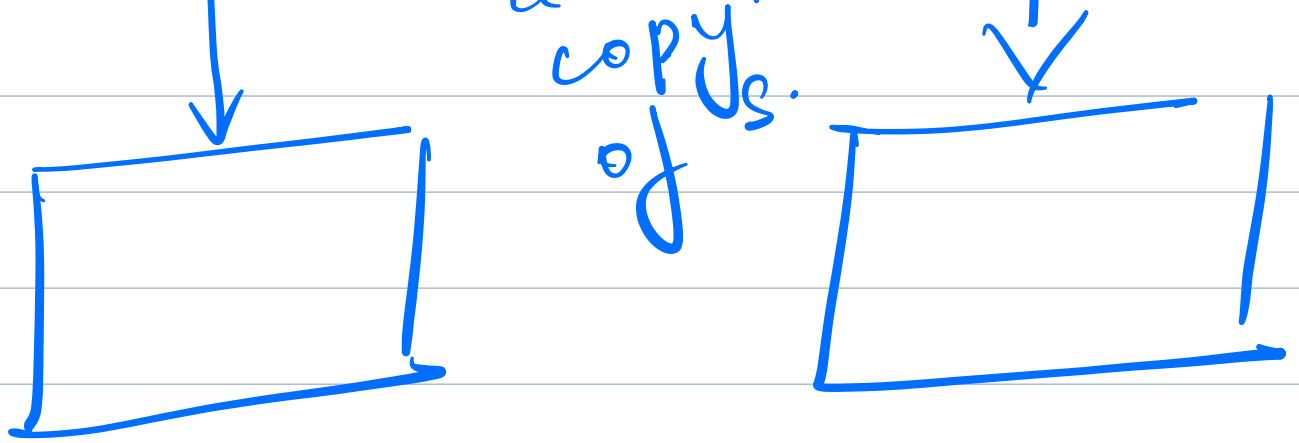


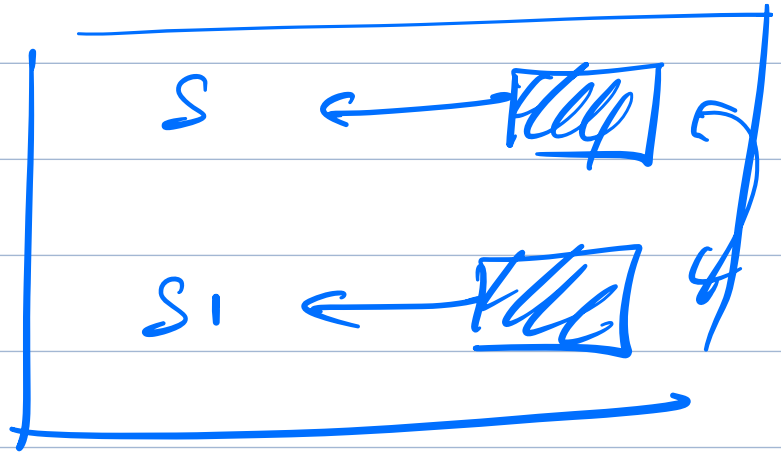


Student s = new Student();

Student s1 = new Student();

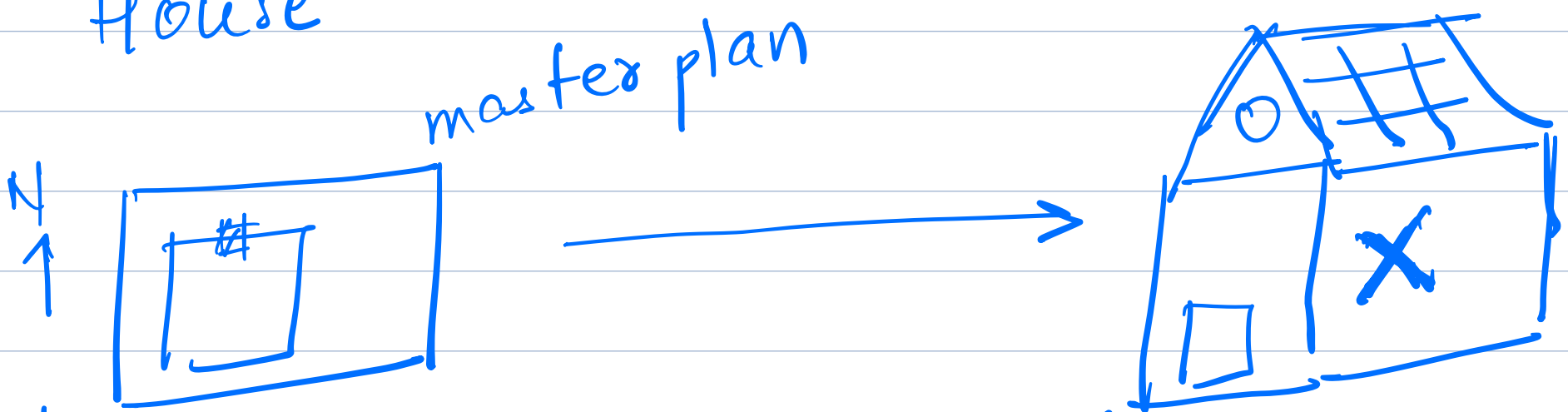






House

master plan



S

reference

address.

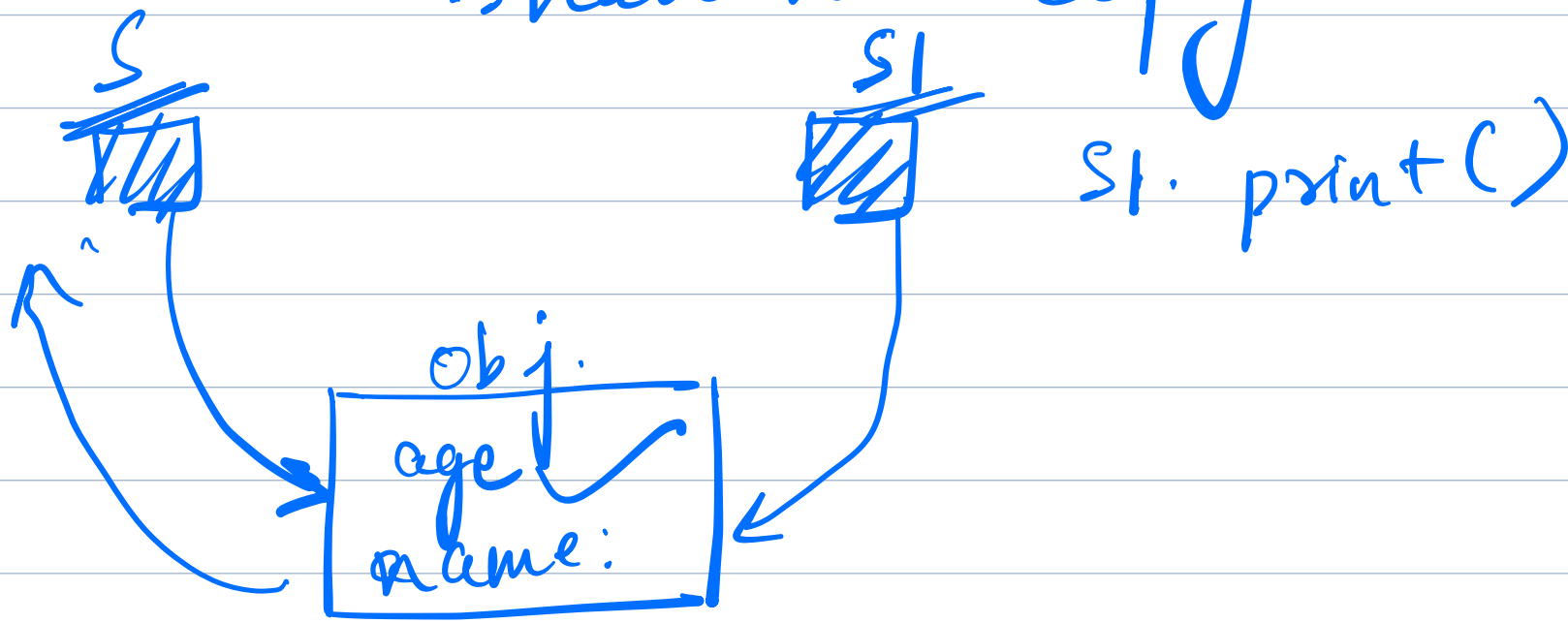
Interior designer

Keys:

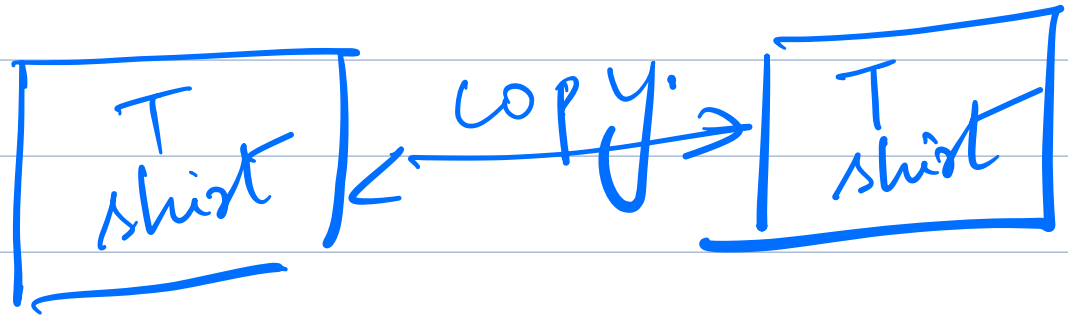
access modifiers

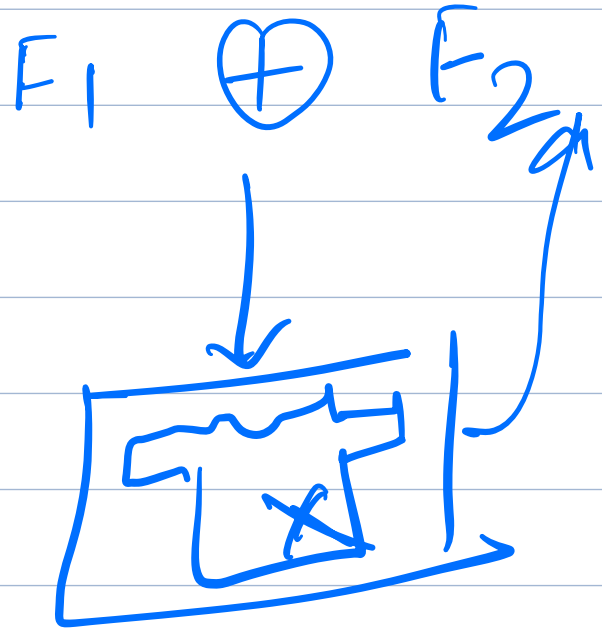
painter

shallow copy



S1. print()





ACCESS MODIFIERS ✓ ⇒ defines protocols for access attr, behaviours from outside the class.

```
Student {
    String name;
```

```
int age;
double psp;

printPSP() { s.o.pln(this.psp)};

}
```

```
Student s = new Student();
s.name = "Nikhil"; X
s.psp = 100.00; ↓

s.psp = 100.00; // should have only proper access.
```

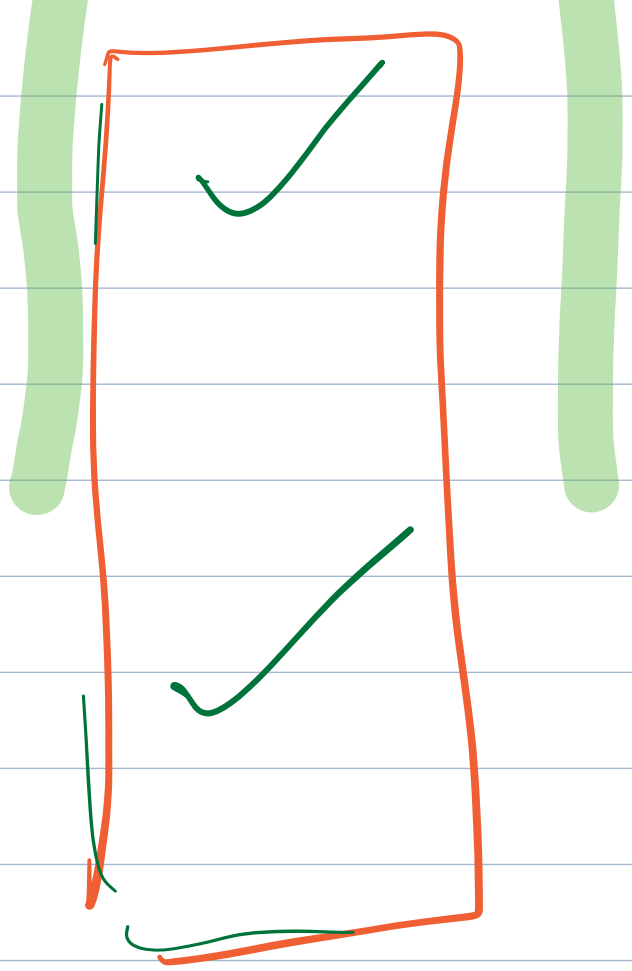
1) public → most open.

2) private → most restrictive (getters & setters).
3) default → it can't be accessed outside the package.
4) protected → accessed outside the package only by a subclass.

	public	private	default	protected
within the class	✓	✓	✓	✓
within same package	✓	X	✓	✓
outside package	✓	X	X	X

accessible by subclass in the same pckg.

accessible by subclass in other packages.

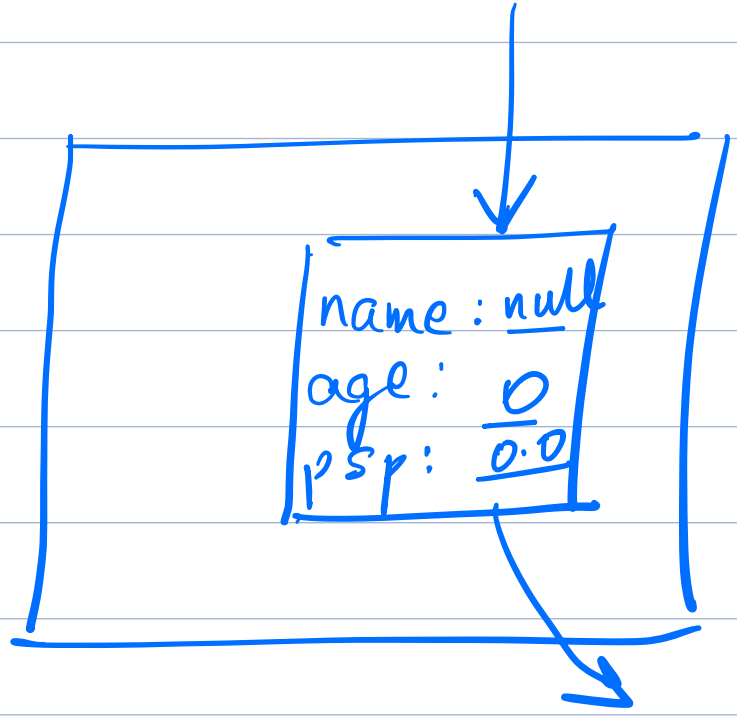


CONSTRUCTORS.

Student s = new Student();

print(s.name); // null.

```
Student {
    String name;
    int age;
    double psp;
}
```



default constructor.

constructor is a special method that help initialize the attributes of a object with default values!

```
student {
    String name;
    int age;
```

```
student ( ) {
    name = null;
    age = 0;
```

} → created by java

```
}
// parameterised constructor.
Student (String newName, int newAge)
{
    name = newName;
    age = newAge;

}
```