OOP-2

Agenda

1. Abstraction ⇐ Principle of OOP
2. Encapsulation ⇐ 1 Pillar of OOP
3. Constructors
   - ↳ Copy Constructor (Deep v/s Shallow Copy)
4. Access Modifiers

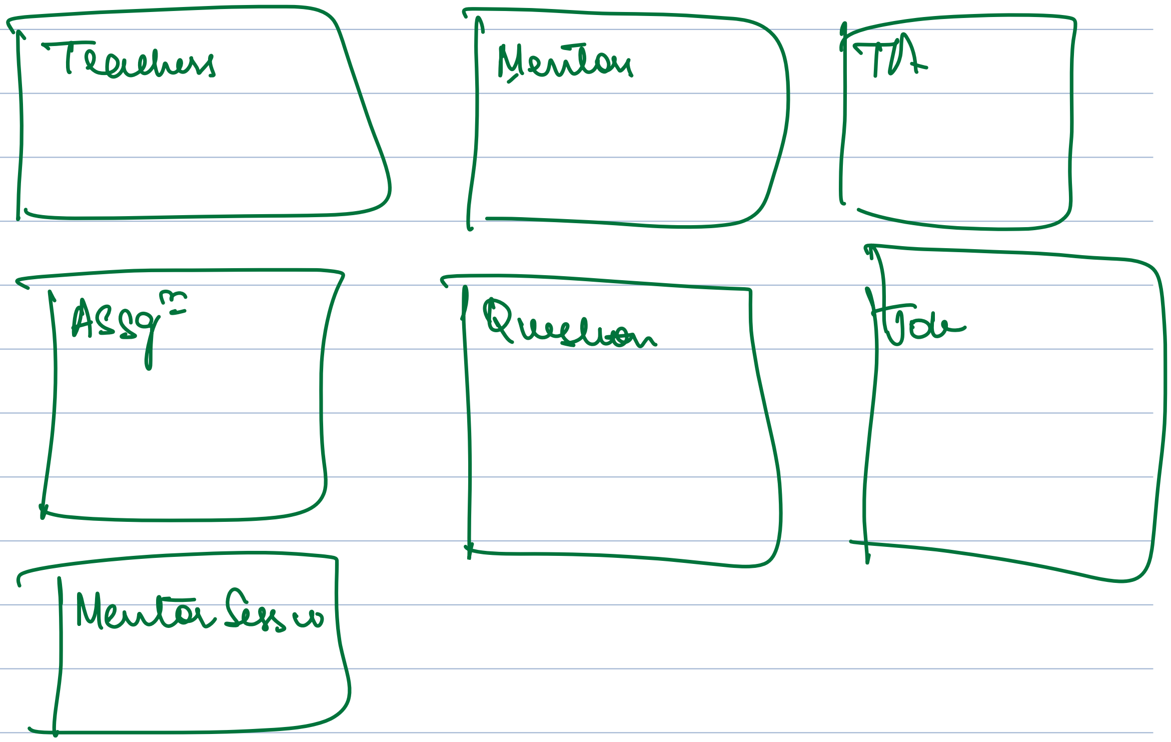
Abstraction

→ The principle on which OOP is based.

↳ making something abstract

↳ idea

→ representing a system as multiple ideas



anything in my s/w system that has attributes (info / data about it) or can cause behaviour

abstraction : representing a complex s/w system in terms of different ideas

Abstraction

① Rep a complex system in terms of ideas:

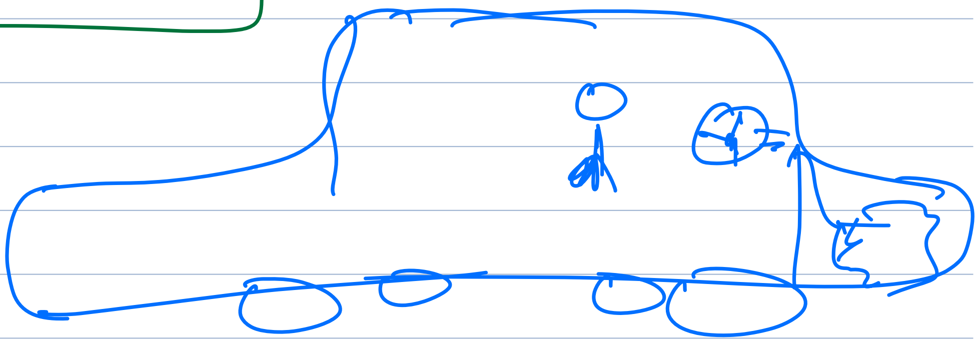
ⓐ has attributes

OR

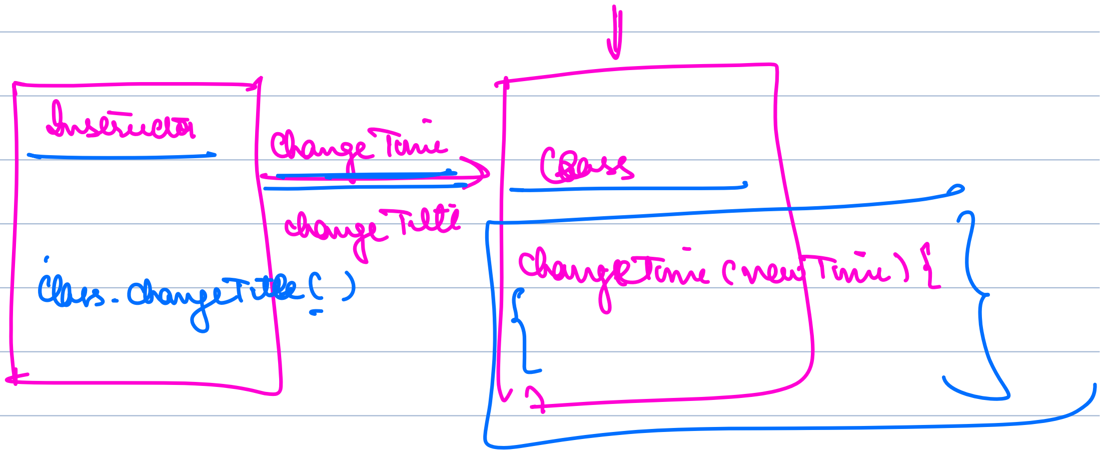
ⓑ can cause behaviour

② others don't need to know all the internal workings of an idea

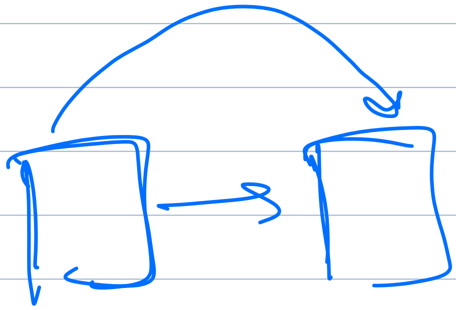
```
ScalerCareerHub {

        List <Job>

}
```



a driver doesn't care about internals of working of a steering wheel.
They only care about how to use a steering wheel!



⇒ a client only cares about what they need.

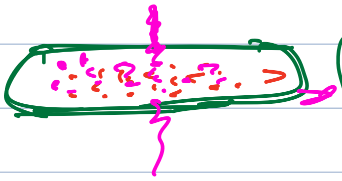


HOW OOP IS BROUGHT INTO PRACTICE

⇒ 3 Pillars

- Encapsulation
- Inheritance
- Polymorphism

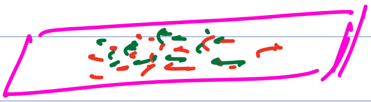
ENCAPSULATION

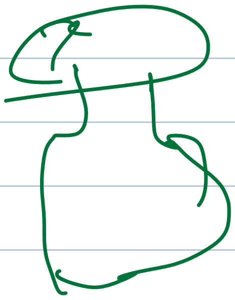


why capsule for medicine:

① Hold the medicines together

② Protect the medicine from harmful environment





Encapsulation in OOP

Procedural ⇒ only attributes

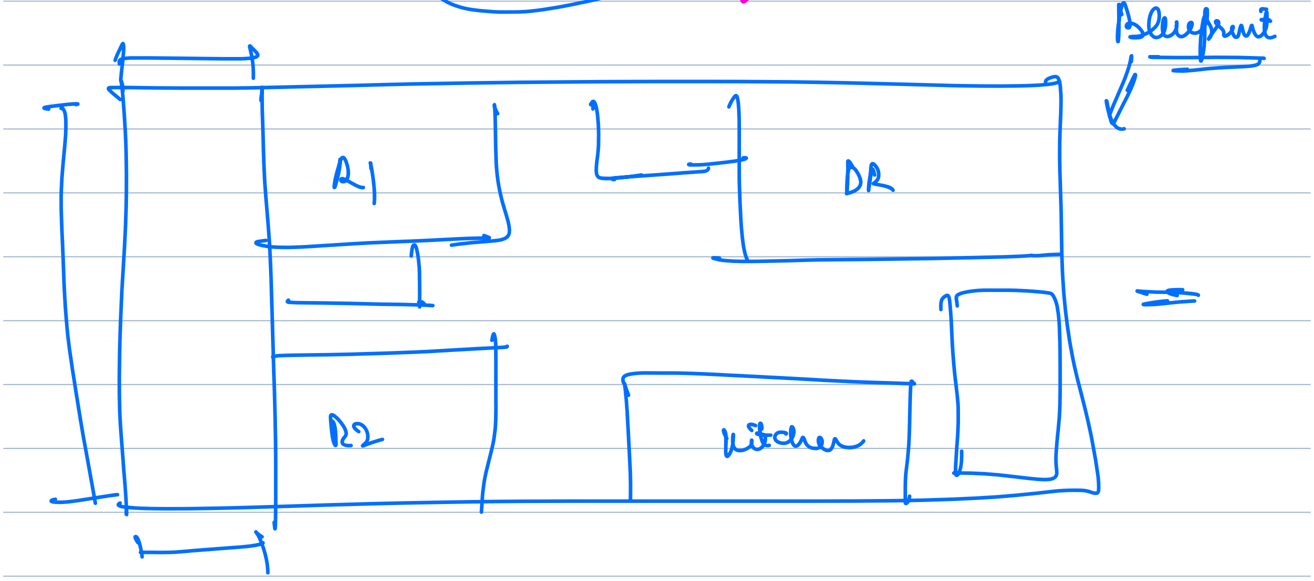
① Stores attributes and behaviours together of entities
(Hold everything wrt an entity together)

② Protects data/behaviours of an entity from illegitimate access

If entity doesn't want to allow, no one should be allowed.

Terms of OOP

① Class : Blueprint of an idea



⇒
- → It is not a real house
- → Just rep of it
- → Doesn't take space.
- → Can create multiple houses using one blueprint

⇓

```
class Student {          ⇒ entity
    - String name;
    - String batch;
    - int psp;
    - String email;

    pauseCourse();
    login();
    joinMentorSession();
}
```

data == attributes → data items about an entity

behaviours == methods

⇓

```
Student-class {
    ____
    ____
    ____
    ____
}
```

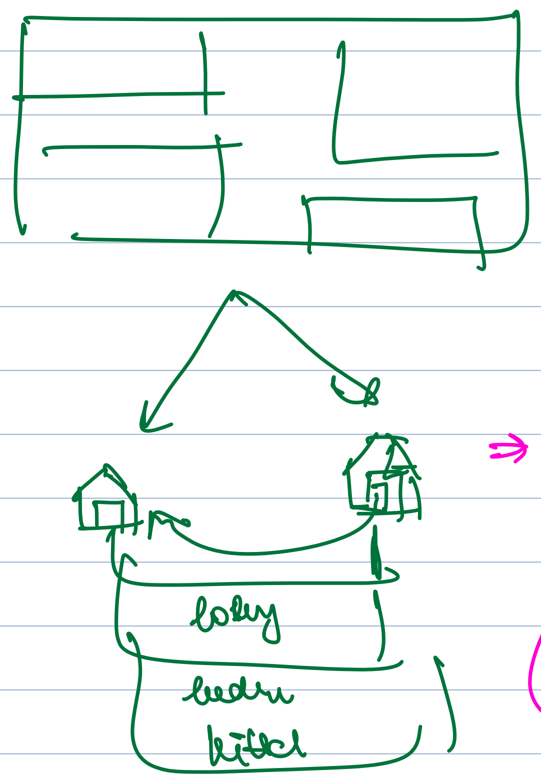
⇒ doesn't take any space in memory ⇒ RAM

⇒ Not a real entity.
Just outline of that

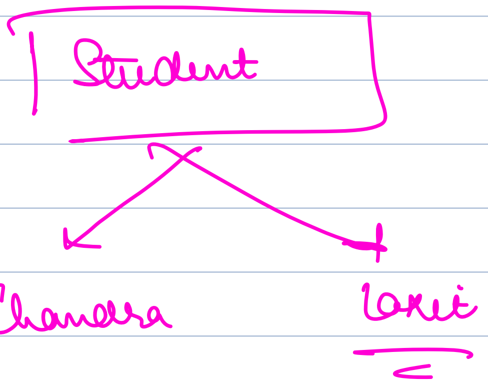
⇒ Multiple instances from one class

OBJECT

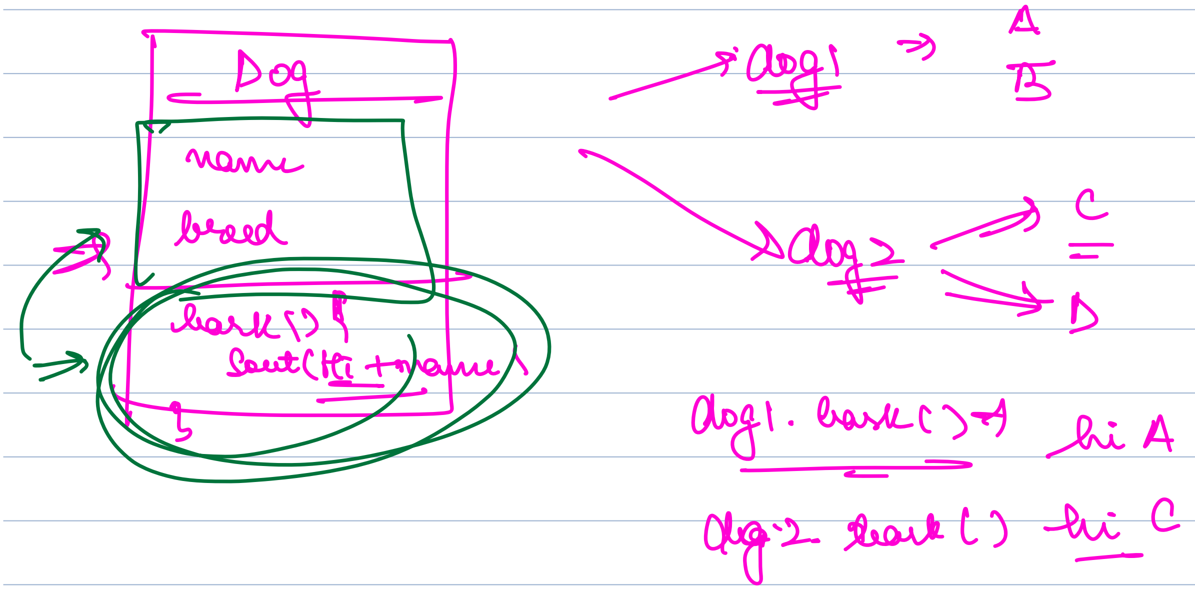
- → Real instance of a class.
- → multiple instances of 1 class.
- → Occupy memory



⇒ each object of a class is completely independent (has its own set of data)



· batch= ABC



① Bind → Class

② Protection →

Access Modifiers

→ access control around who should be able to access what - data / methods.

⇒ with every attr/ method we can attach one of the following access modifiers

① public
② private
③ protected
④ default → when no access modifier

me and my neighbour

most to least strict

	Same Class	Same Folder (Package)	Child Class in Sam Folder	Child Class in other Folder	Anywhere
Private	✓	✗	✗	✗	✗
Default	✓	✓	✓	✗	✗
Protected	✓	✓	✓	✓	✗
Public	✓	✓	✓	✓	✓

```
Class Student {
  Private int age

     doSomething () {
         Cout  age          ✓
     }
}
```

```
Main {
   psvm() {
     Student st = - -
     ~~st. age~~
   }
}
```

```
Class Student {
      int  age;



}
```

package 1

Student. java (default age)

Main . java {

```
psvm() {
    Student st1 = ______
    st1. age = ______
}
```

}

package 2

Main. java {

```
psvm() {
    Student st1 = ______
    st1. age = ✗
}
```

}

CONSTRUCTORS

```
Student st = new Student ( ) ;
```

new → Keyword in java used to create an object

Student ( ) → Constructor

⇒ Special method whose purpose is to create an object of a class

[you can't create an object w/o using new keyword]

Name of cons is same as name of the class

```
private void doSomething(      );
```

Yes:

Even a cons can have access modifier

→ Singleton

No!

→ Return type of a constructor is the class itself

```
public Students {

}
```

DEFAULT CONSTRUCTOR

→ only created if I don't create any constructor myself

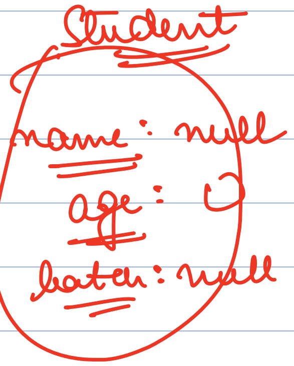
→ it creates an object of the class and instantiates its attributes to default value of their data type

```
Student {

        → Student (name, age)

}
```

~~Student st = new Student()~~

```
Student() {
    String name;
    int age;
    Batch batch;

}
```

Student st = new Student()



CUSTOM CONSTRUCTOR

```
Student {
        String name;
  ①     int age;
        String gender;

Custom
Constructor
        public Student(int age, String name) {
              this.age = age;
              this.name = name;

}       }
```

"self"

⇒ there maybe scenario where params names conflict with class attr names

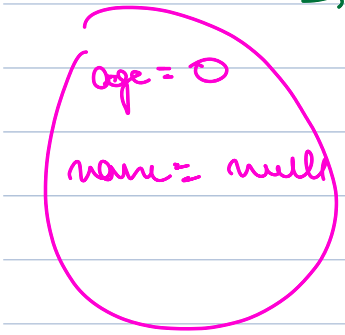
⇒ use "this" keyword

```
public void doSomething (int age) {
          age = 31 }
```

```
}
Main {
    psvm() {
          Student st1 = ~~new Student();~~
          Student st2 = new Student(21, "Name")
}   }
```

How cons gets executed

→ even before first line of code inside a cons is executed, a new obj is created and all attr are initialized to their default values.



```
public Student (int age, String name) {
        Sout( ~~Name~~ Age is + this.~~name~~ age);
}
```