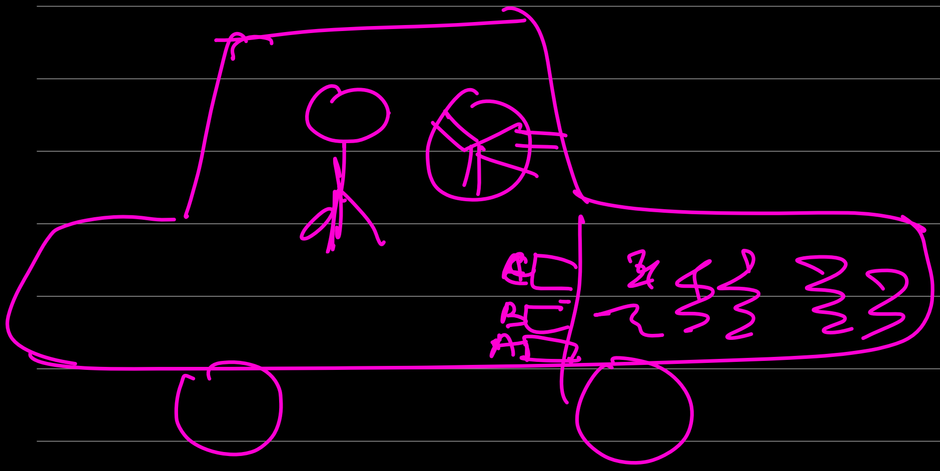
Agenda

① Abstraction ⇒ Principle
② Encapsulation ⇒ Pillar
  → Constructors → Default, Copy
  → Access Modifiers → Public, Private, Protected, Default

ABSTRACTION

→ Hiding ~~details~~

⇒ thinking in "terms of ideas"



⇒ Representing a complex software system in terms of ideas that cause behaviour in the system while hiding away things that are not required by a client.

Ideas in Scaler S/W

Assignment, Dashboard, Student, Instructor, TA, Class, Job, Placement

Placement
- Job
- Student
- Salary

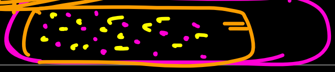
Abstraction

① Rep a complex S/W System in terms of ideas
- ↳ attributes
- ↳ behaviour

② Not showing unnecessary details outside

Pillars

① Encapsulation   Capsule



= ① Protects Medicine from outside env.

→ ② Keep the medicines together

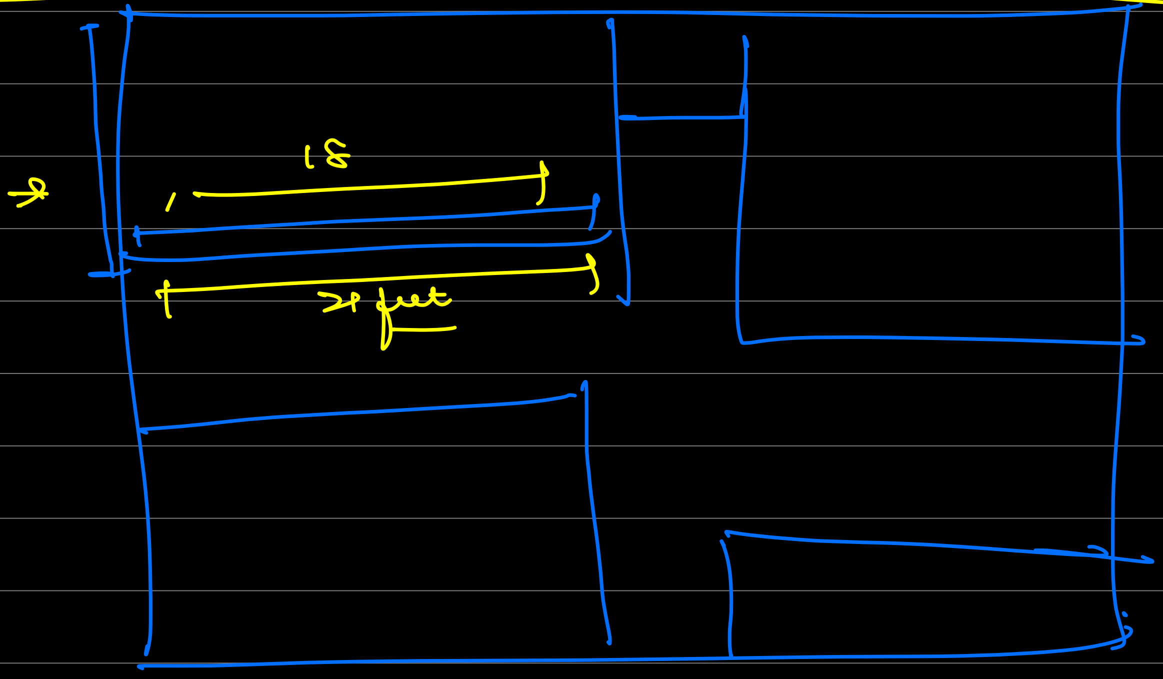
Features of encapsulation

→ Hold attributes and behaviours together

→ Protects attributes and behaviours from illegitimate outside usage.

Fundamental terms of OOP

① Class ⇒ Blueprint of an idea Blueprint



① Blueprint is not a house but it tells how houses will look like

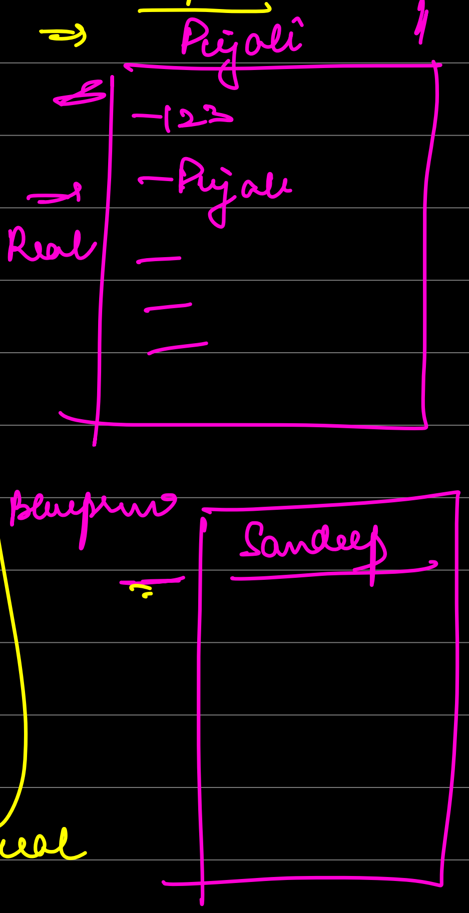
② Not going to take any

Place of plot. Just a design.

③ Use one blueprint to create multiple houses.

```
class Student {
    - id
    - name
    - gender        attributes
    - age
    - batch

                    pause Course()
    behaviour       change Batch()
                    solve Assignments
}
```



① Class tells how every real instance will look like

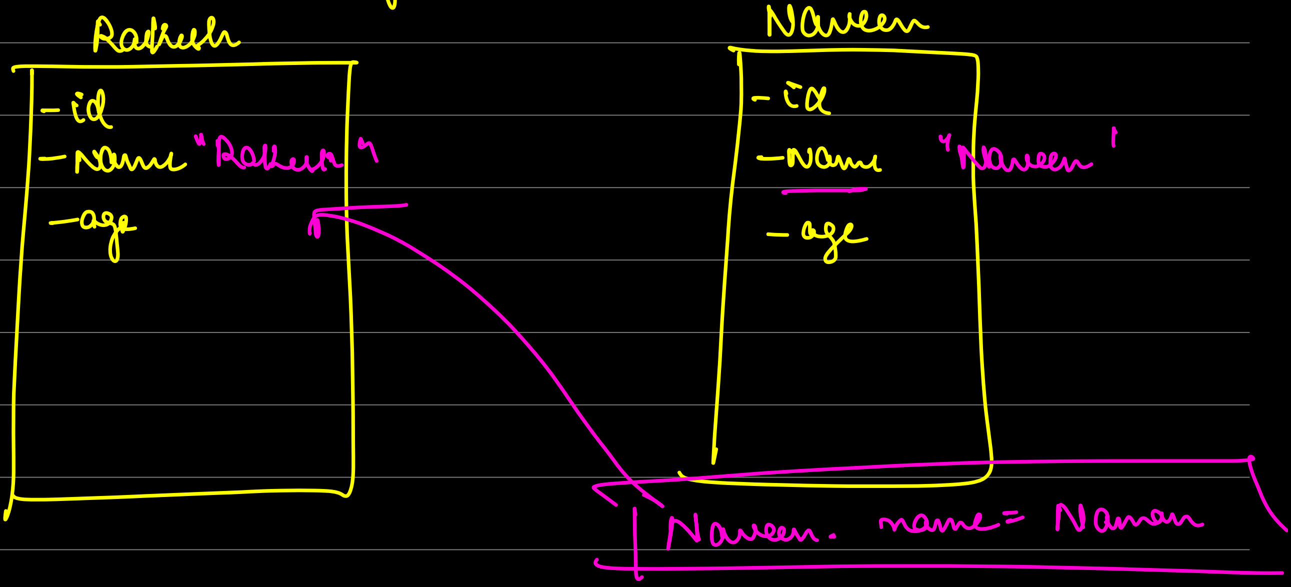
② Class doesn't take any space in memory (RAM)

③ Can be multiple real instances from one class

② Object

→ Real instance of a class

→ occupy space in memory



Access Modifiers ⇒ Modify the access levels of other code to attributes and methods of a class.

1. Public
2. Private
3. Protected
4. Default

→ every attribute and method of a class can have one of these access modifiers

Based upon the access modifier, external code will be allowed or denied to access it.

	Who can access ↓				
	Same Class	Same Package/Folder	Child Class in Same Package	Child Class in Diff package	Anyone
Private (Only Me)	✓	✗	✗	✗	✗
Default (Me and Neighbours)	✓	✓	✓	✗	✗
Protected (Me, Neighbours, Children)	✓	✓	✓	✓	✗
Public (Anyone)	✓	✓	✓	✓	✓

$$\boxed{\text{Attributes + Methods}}$$

$$\Downarrow$$

$$\boxed{\text{Members}}$$

Private = Only I should have access

Default → Me or my neighbourhood

Child Class Non child Class

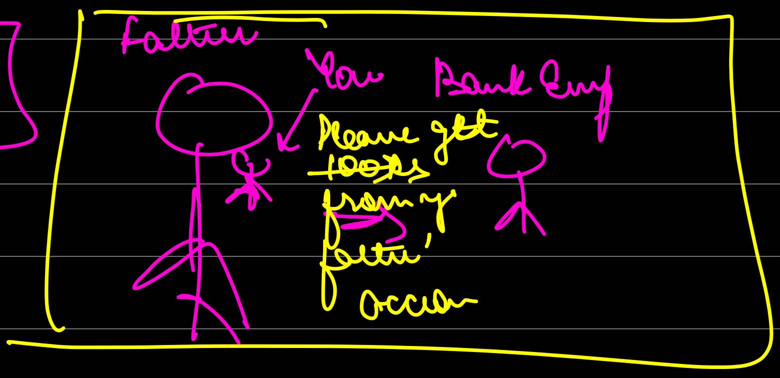
Protected → Me or my neighbourhood or

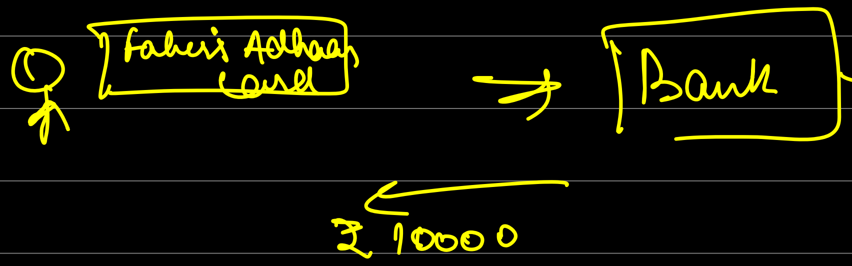
⇒ (Child outside my neighbourhood)

If child accesses from parent ✓

If it accesses from object ✗

Father + Bank





Protected ⇒ Me

My Neighbourhood

Outside Neighbourhood only if my child and child is accessing directly

CONSTRUCTORS

Keyword of Java to Create a new object

```
// int      age = 12
Student     st = new Student();
```

Till a time new keyword is not used anyhow, a new obj can't be created

doSomething( )
↳ Student ( )

→ a constructor is a special method

① Name as name of the class
② It can take params
③ It can have access modifiers
④ No return type
↳ Return Type is the Class Itself

Default Constructor

→ If (and only if) we don't create a constructor ourselves (method with name as Class Name), programming language will automatically create a constructor for us.

⇒ This initializes all attr to their default value

(unless I have given a diff default val)

int	→ 0
double	→ 0.0
boolean	→ false
String	→ null
Object	→ null

→ public

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

Student st = new Student()

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

}
```

Student st = new Student()

↳ 1
↳ null

Custom Constructors

→ Our own constructors
→ We can have multiple constructors in a class (Method overloading)
↳ Each constructor may take diff params

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

    public Student(ageofStudent, nameofStudent) {
        age = ageofStudent;
        name = nameofStudent;
    }
}
```

⇒ ~~Student s = new Student()~~ ?

Student s = new Student(12, "Naman");

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

    public Student (ageOfStudent, nameOfStudent) {
        age = ageOfStudent;
        name = nameOfStudent;
    }
}

~~public Student (String nameOfStudent) }
    name = nameOfStudent.
}
public Student(){}~~

Student c = new Student ("Naman")

Student s = new Student (12, Naman)

Student s = new Student();
```

11/11/22/11/11

Next Class :⇒ Complete custom constructor

⇒ Copy Constructor

⇒ Inheritance

→ Polymorphism