Agenda.

→ Intro to OOPS.

→ Pillars | Principles of OOPs.

→ Access Modifiers.

Properties of good code.

1) Extensible : Easy to add new features
2) Maintainable : Current system should keep on running.
3) Readable
4) Modular.

$$\begin{tikzcd} \Rightarrow \text{Phonepe} \arrow[r, leftrightarrow] \arrow[dr] & \text{YB} \ \times \downarrow \\ & \text{ICICI} \end{tikzcd}$$

Intro to OOPs.

$\boxed{C}$

Procedural Programming Paradigm

C++

Java

Python.

Object Oriented Programming Paradigm

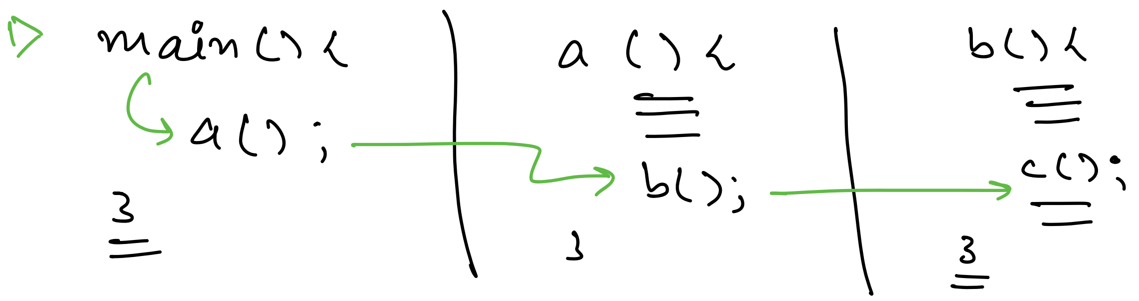
Procedural Paradigm

Procedure. ⇒ fun | method.

→ We organise our code in the form of procedures

→ One procedure may call another procedure internally

→ Execution of program starts from a special procedure which is called as main



```
main() {        | a() {     | b() {
    a();        |   ==      |   ==
}               |   b();    |   c();
                | }         |   ==
                |           | }
```

Problems with procedural languages.

1. Deepak is teaching LLD.
2. Everyone is thinking of a line.
3. Manish is taking notes
4. Abhishek is looking for a job.

⇒ Subject + Verb.

⇒ Entities are performing some actions.

Struct Student

```
PrintStudent (String name, int age, Str batch) {
        print(name)
        ________
        ________
        ________
}
```

Something   Someone

```
PrintStudent ( Struct Student)
        print(name)
        ________
        ________
        ________
}
```

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

⇒ Struct can't have methods.

⇒ Something is happening on someone.

In real life.

Someone should do something

Student. print ( );

Procedural	OOP
Print (Student) ⇒	Student.print().

OOP.

↳ Software Engineering idea which consists of Entities. → Class.

→ Each entity will have some attributes & some behaviours.

⇒ Class Student {

```
    name
    age
    batch
    PSP              Attributes.
    email
    Company
    CTC

    attendClass(){ }
    bookMockInterview(){ }
    SolveAssignment()       behaviours.
    ____
    ____
    ____
    ____
}
```

Class ⇒ Blueprint of an Entity

OOPs → 1 principle ⇒ Rule | fundamental.

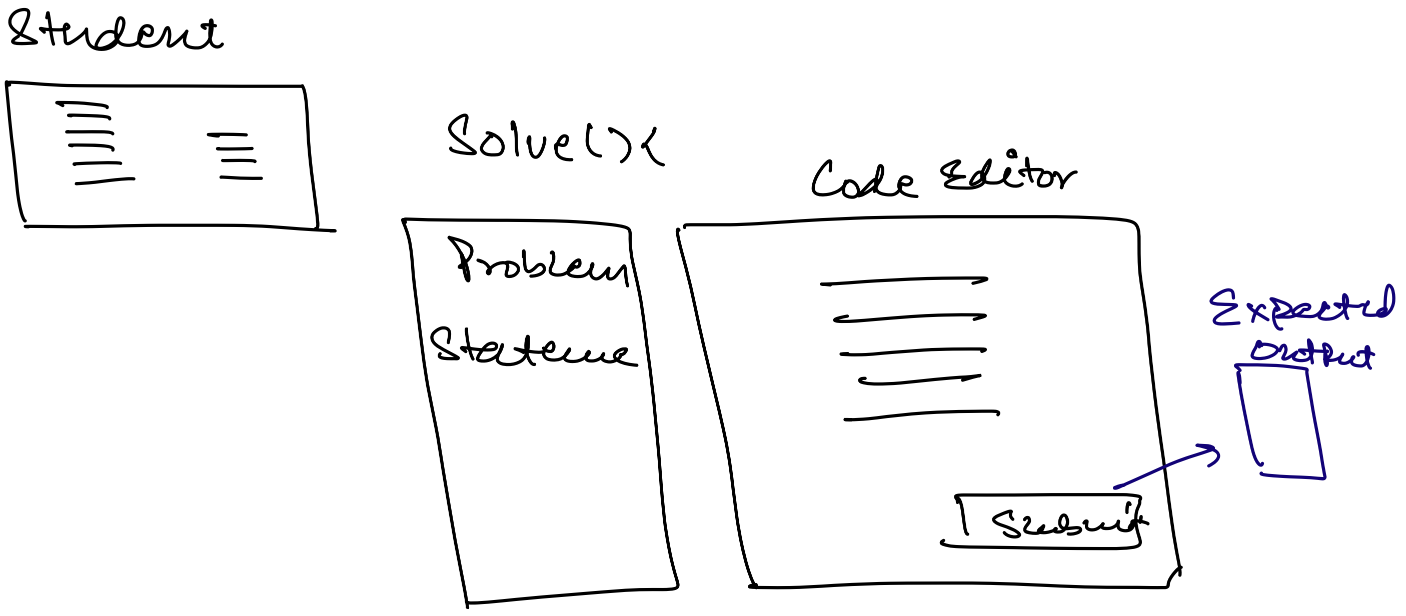
OOPs → 3 pillars.

Principle : Abstraction.

Pillars : ① Inheritance
② Encapsulation
③ Polymorphism.

ABSTRACTION : Concept of making something abstract

⇒ Representing things in terms of ideas



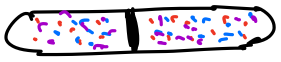
Abstraction.

1. Representing a complex system in the form of ideas | Entities.

$$\begin{tikzcd} & \text{Entities} \arrow[dl, dash] \arrow[dr, dash] & \\ \text{Attributes} & & \text{Behaviours} \end{tikzcd}$$

2. Others need not to know the Internal details of the systems.

ENCAPSULATION.



→ Holds medicines to together

→ Protect medicines from external env.

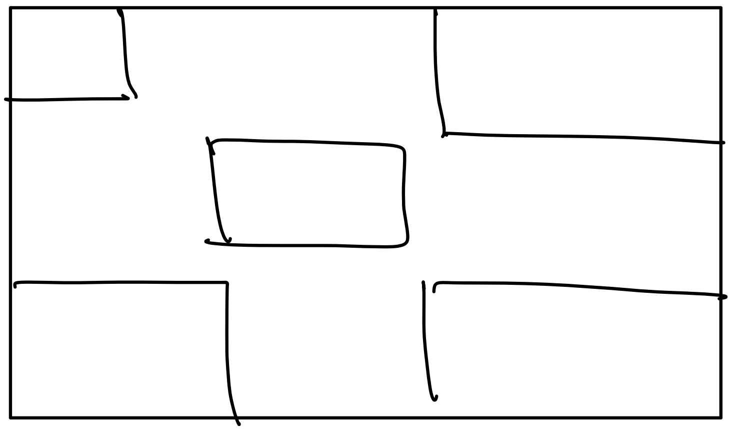
ENCAPSULATION in OOPs.

1. Store attributes & behaviours of an entity together. ⇒ Class.

2. Protect attributes & behaviours from illegitimate access.

Class. ⇒ Blueprint of an Entity

↳ Represents Structure of an entity.



↙ Blueprint

Class.

```
Class Student {
        String name;
        String email;
        int age;
        double psp;
        String batch;
}
```

→ Class takes No space in the memory

→ Not a real entity, it is just a blueprint.

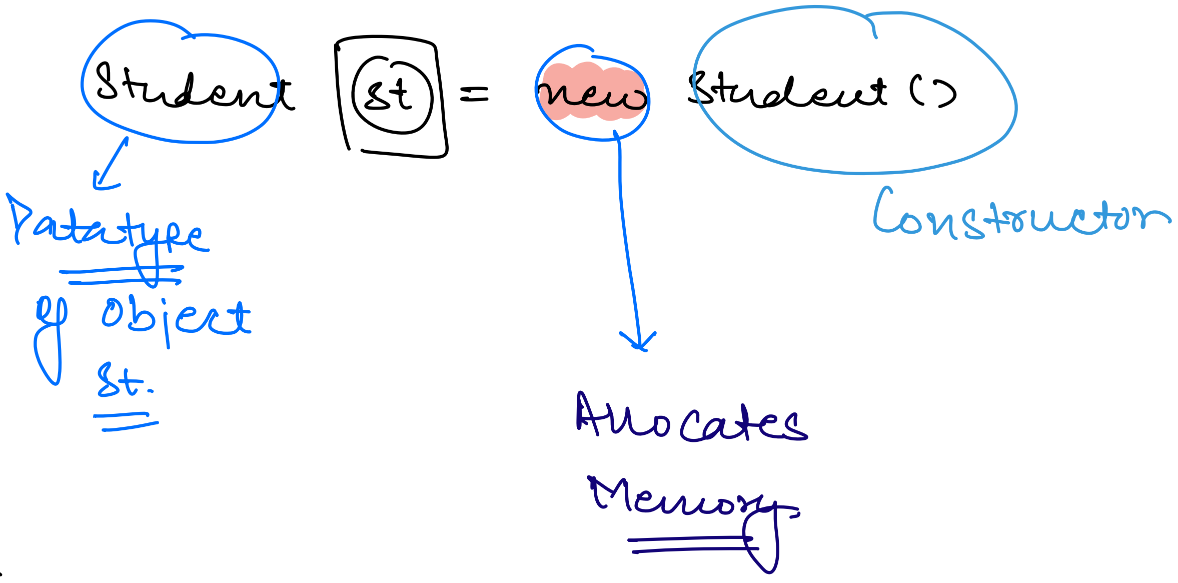
→ Multiple instances can be created of the same class.

Object

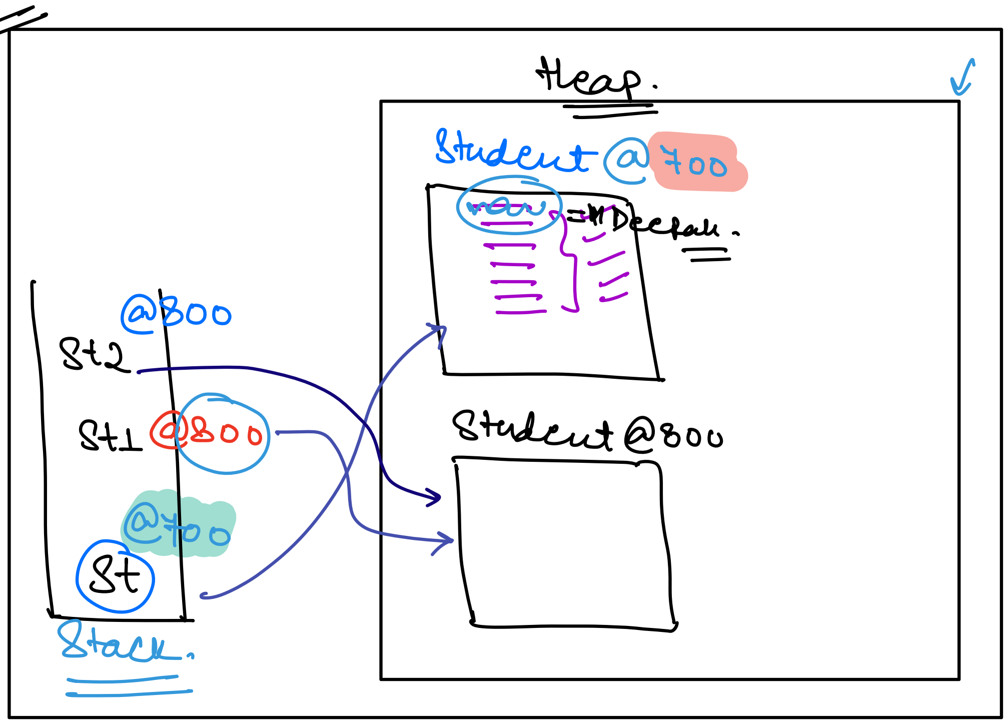
⇒ Real instance of a class.

⇒ Occupies memory

⇒ All the Objects are independent of each other.



Java.



Student St1 = new Student();

print ($\boxed{St}$.$\underline{\underline{name}}$)

@700

St.name = "Deepak"

Student St2 = $\boxed{St1}$; $\Rightarrow$ No new memory location will be allo$\underline{\underline{cated}}$.

$\underline{\underline{@800}}$ $\Downarrow$

because we are not calling new key$\underline{\underline{word}}$.

Access Modifiers.

Public ⇒ Anyone can access.

Protected ⇒ Anyone in the same package + Child classes anywhere

default ⇒ Anyone in the same package

Private ⇒ Only within the class.

↓ Increasing Strictness.

Package ≡ Folder.

Java:

	Class	Package	Child class (Same package)	Child class (Different pack)	World
Public	✓	✓	✓	✓	✓
Protected	✓	✓	✓	✓	✗
default	✓	✓	✓	✗	✗
Private	✓	✗	✗	✗	✗