Agenda → Intro to OOP
Procedural languages
OOP → Abstraction
→ Encapsulation

Class starts at 9:05 PM

Ques → What all prog. paradigms have you heard of
→ functional → Scala
→ procedural → C
→ declarative → SQL
→ Reactive → Java Rx
→ OOP → Java, Python

→ A(function
arguments)

→ functional → functions are first class citizens and you can pass around func^{n}.

→ declarative → prog. languages with

human readable code

→ Reactive ⇒ react to diff. events

⇒ Procedural →

```
A() {
   ——
   ——
   ——
}
```

Procedure ↓ block of instructions / code

⇒ Procedure is the old name for functions

⇒

```
A() {
   ——
   ——
   ——
}

B() {
   - A()
   ——
   ——
}

C() {
   - B()
   - A()
   ——
}
```

→ Can one procedure call other procedure → YES

→ set of procedures ⇒ main() (entry point of the application)

→ printStudent (name, age, batch) {

```
    print (name)
    print (age)
    print (batch)
}
```

Note: properties, fields, attributes, data

Student → age, name, batch, psp

There is something in a procedural programming paradigm/lang → struct

```
struct Student {
    int age
    String name
    String batch
}

printStudent ( struct s ) {
    print (s.name)
    print (s.age)
    print (s.batch)
}
```

```
struct Batch {
    String name
    String instructor
}
```

→ struct → only has data and all of it is public

→ doesnt have any methods or functions.

→ Now lets try to relate it to ther real world

real world → Abhinav is teaching
real world → Shreesh is studying

Abhinav, Shreesh — noun — someone

is teaching, is studying — verb — something

print Student (struct S)

Verb — something

noun — someone

⟹ Procedural prog. lang. is not the natural way of thinking. This is where we get introduced to OOP.

⟹ OOP ⟹ entities form the core of Object oriented prog.

entity ⟹ Student ⟹ age, name, batch

takeATest(), attend a class(), solveProblem()

Data + behaviour ⟹ functions or method that a particular entity can have

Cons of procedural prog.

① It is not natural, something is done upon someone

② lets say we have an application

↳ 10/20 procedures

→ 1000s of procedures
1000s of files

} diff. to understand, diff. to develop, diff. to maintain

(B) struct Student

printDetails() AttendClass() TakeATest()

If diff. func^{n} related to the same struct to are present in diff. places ⟹ Spaghetti code

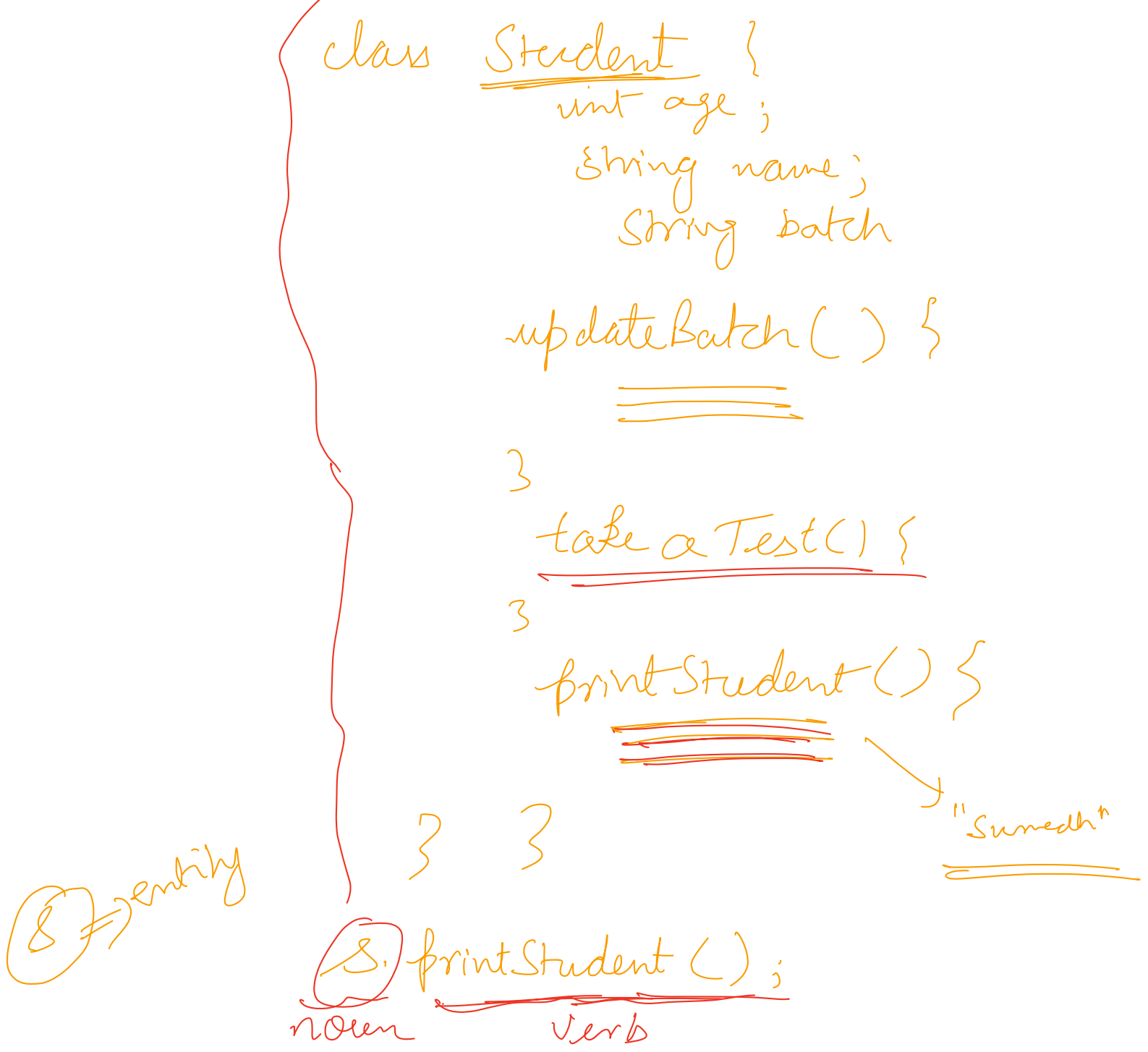
1 year ⟹ new function for } Spaghetti code
2 year ⟹ new function }

⟹ Object Oriented programming →

⟹ Entities form the core of OOP
↳ data
↳ behaviours } (functions)

In OOP → ① we model everything around entities

OOP → ② nearer to the real world



Ques : What are the 4 pillars of OOP?

1. Abstraction ⇒ main principle of OOP
2. Encapsulation
3. Inheritance
4. Polymorphism

} Pillars of OOP

Principle ⟹ core belief or idea

pillar ⟹ help in implementing a principle

→ ~~Principle~~ ⟹ I'll be a good person
~~pillar~~ ⟹ I'll not lie
I'll feed the hungry
I'll be loyal

class starts at 10:18 PM

~~Everything~~ we do in OOP ⟹ to support the core idea of Abstraction

↳ Encapsulation, Polymorphism, Inheritance

(1) Abstraction ⟹ quality of dealing with ideas

In S/W world, representing our concepts in the form of ideas → Abstraction

Scaler S/W system → I have to represent → Pranshu, Lakshmi, Abhishek

↙ general idea

1. Students

Abstraction → not representing individually but abstracting out a general idea

2. Instructors

3. Batch

Ques What do you think is the benefit of representing in form of idea?

↳ others dont need to know the internal details of it

Car ⇒ steer a car
apply brakes
apply handBrake

Student ⇒ printStudent()
takeATest()

⇒ Now when we talk about S/W system we basically represent 2 things
- data or attr's
- behaviours or methods

⇒ how do we implement abstraction
↳ encapsulation

Java : The complete reference

② Encapsulation ⇒ 
- ① holding the medicines together
- ② protect it from outside

⇒ In S/W system ⇒ encapsulation means
- ① holding the related attributes and the behaviour of an entity together
- ② protect it from others/ outside

how do we do — encapsulation — by creating classes

implementing abstraction

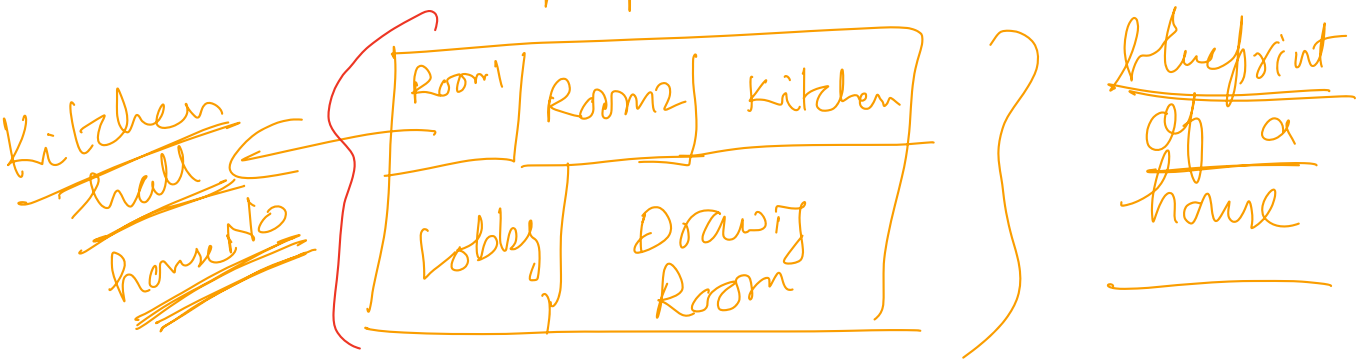
```
class Student {
    int age
    String name
    String batch
    int mobileNO
    void printStudent() { }
    void changeBatch() { }
}
```

Access modifiers ⟹ to protect the attributes & behaviour/ method

Lets explain Terminology

1) Class ⟹ Blueprint of an idea

construction industry ⇒ when a design of a house is made it is done on a blue paper



In S/W class ⇒ represent blueprint of an idea

↳ Student ⇒ name, age, batch

Ques → how much space does a blueprint take → NO → similarly in S/W class doesn't take up any space (minimal)

⇒ (2) Object ⇒ In S/W system if we have, Student as class then

Student s = new Student()

↓

its a real entity

it would take up space in memory

Memory

Student a = new Student()

will have its own attributes
will have its own memory
does not share anything with other objects

Student s = new Student()
s.mobileNO = "8884997776"

If you don't do this then s' mobileNO will be 0

Recap → Procedural lang → deals with procs.
- not natural
- spaghetti code
- diff. to understand

→ we got to OOP

- Everything in form of entities (core of

OOPS

↳ 1 principle of OOPs → Abstraction

3 Pillars → Encapsulation
Polymorphism
Inheritance

Abstraction → deal in terms of ideas

Encapsulation → ① Related attrs and behaviours together
② protect it from outside

↓

→ done by creating classes
→ implements Abstraction

Everything we do in OOP → support the core idea of Abstraction

→ class → Blueprint of an idea
→ objects → actual entities (instance) or an instance of the blueprint

Next class → ~~constructors~~
~~Inheritance~~
~~Polymorphism~~