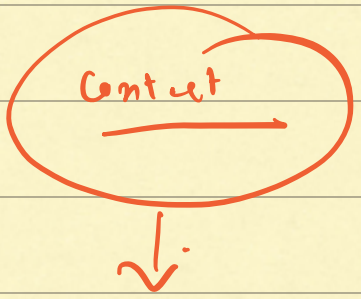


Very Important.

Agenda

1) Use case Diagram
2) Class Diagrams

UML Diagrams

→ Unified Modeling language.

Phonepe.

communication → via email ✓
→ via gmeet (video conference) ✓
→ Slack / messaging apps. ✓

↓ in words

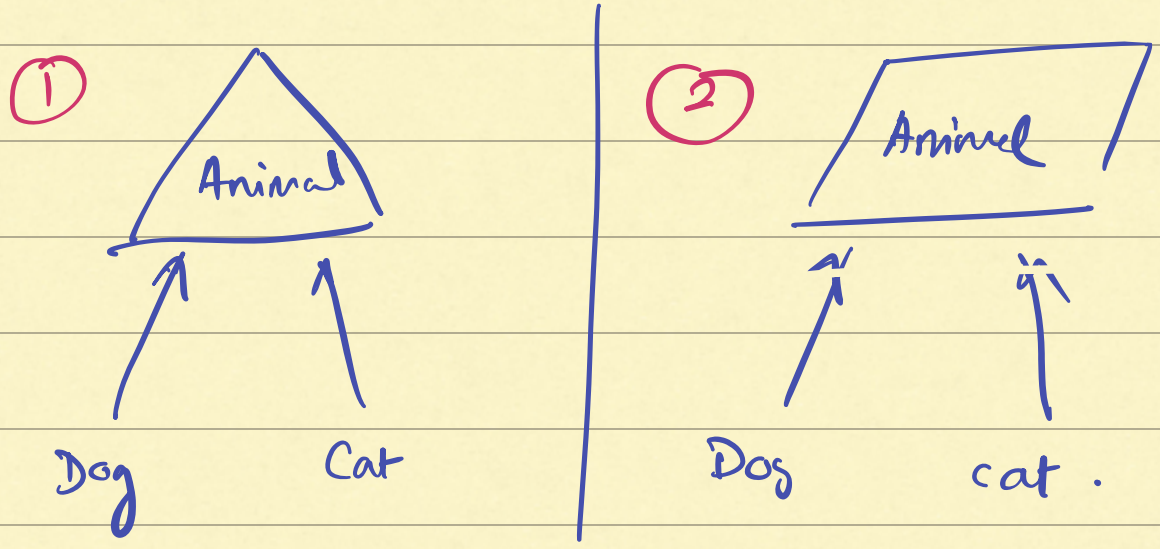
mis communication

→ Solution : Visual Diagrams

Product Manager
Client
} → Explain what you are doing (business)

Fellow Devs / Peers
Senior engineers
} → Get reviews

Technical Architect :- → Get approvals



UML

Structural

- Class Diagram
- Object Diagram
- Package Diagram
- Component Diagram

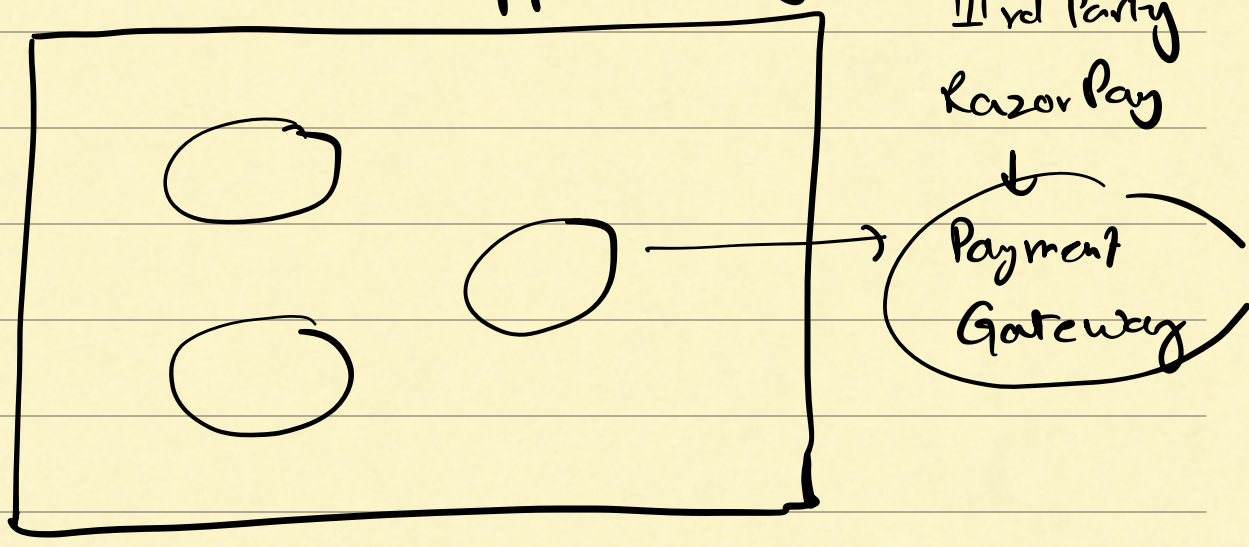
Behavioural

- Use Case Diagram
- Activity Diagram
- Sequence Diagram.

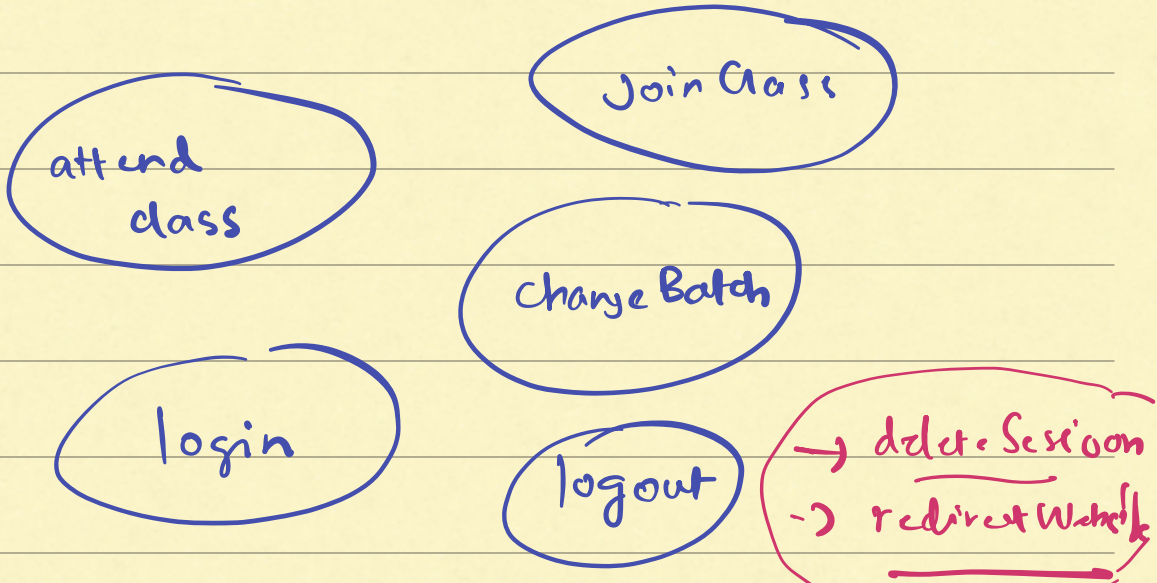
Use Case Diagram :

5 important Keywords fo Use Case Diagram.

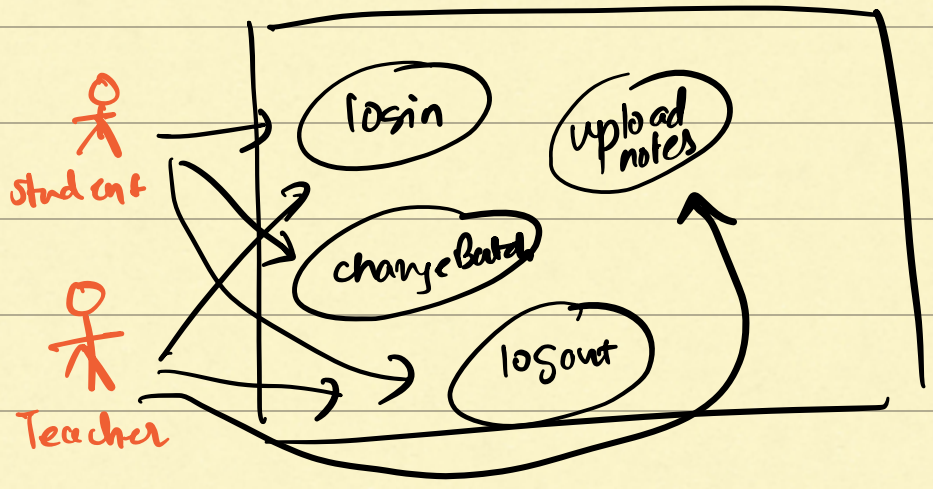
1) System Boundary : It contains functionality supported by you.



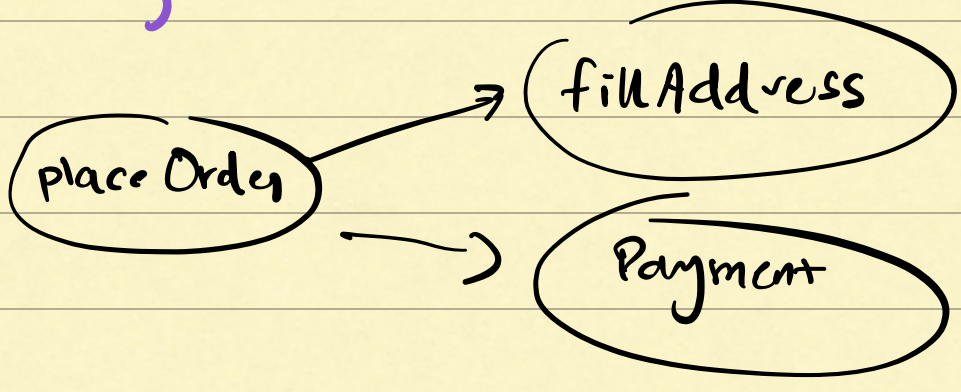
2) Use Case : It is the feature / functionality
It is written in a oval
It must be a verb



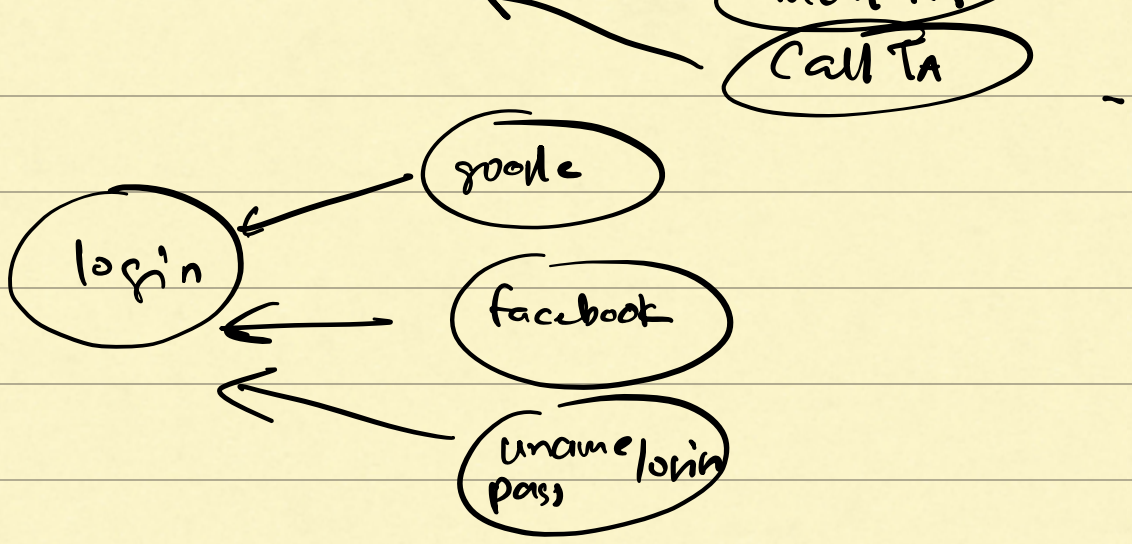
3) Actor : . People who uses the use-case.
Drawn with stick figure



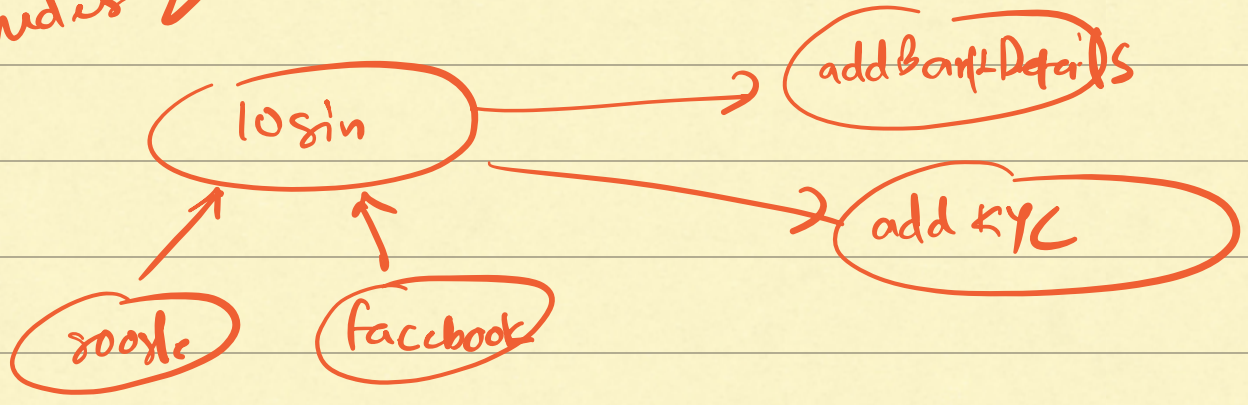
4) Includes : Needed to represent a feature (use-case) which is including other features.



5) Extends : Variant of features.
Like Inheritance.



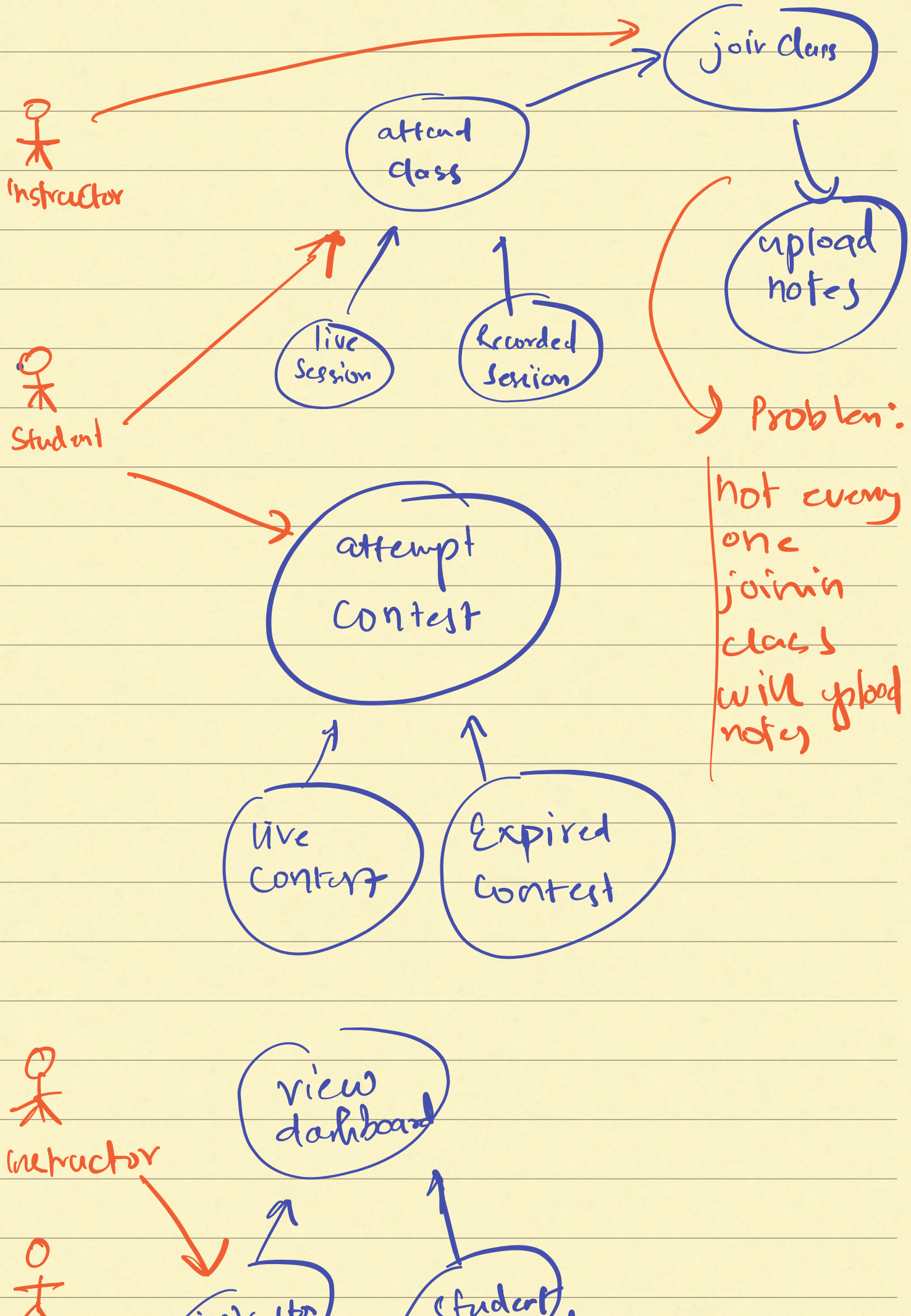
common way of putting includes & extend.



Break till : 8:05

1) 3 use case
2) 2 Actors
3) 1 Include & 1 extend

} Scaler


join class
Instructor
attend class
upload notes
live session
Recorded session
Student
Problem:
not every one joinin class will uplod notes
attempt Contest
live contest
Expired Contest
view dashboard
instructor
student

Class Diagram:

Different Entities:

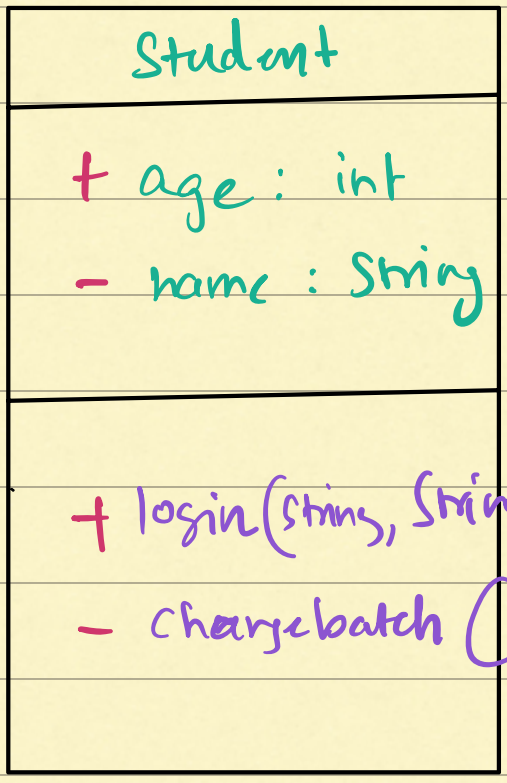
- classes
- interface
- abstract classes
- enums

Different Relationships:

1) IS-A → Inheritance
   → implements

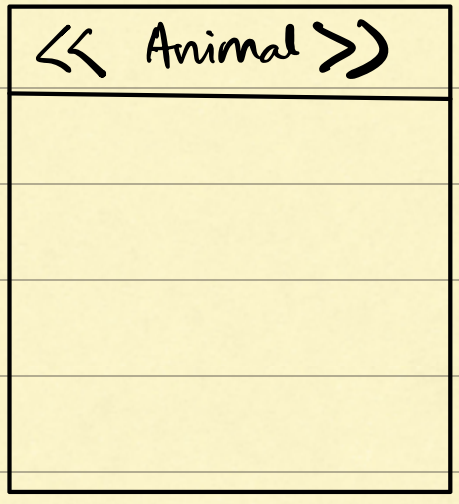
2) HAS-A → Aggregation
   composition

Class

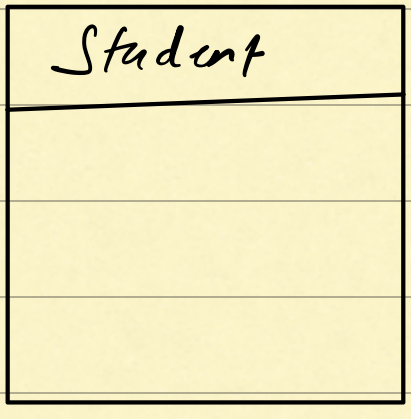


+ public
- private
protected

Interface



Abstract

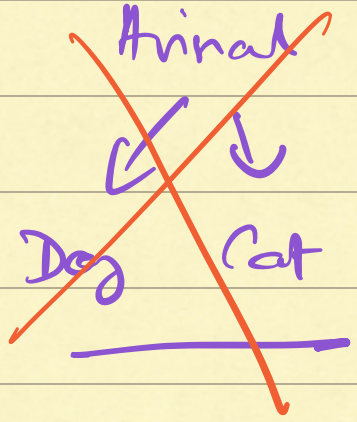


→ Italics

Enums

AnimalType
CAT, BIRD, DOG

→ AnimalEnum.



Relationships

IS-A

Inheritance

Animal

↑ extends

Dog

Interface.

<<Flyable>>

↑ implements

Bird

HAS-A : • Composition
• Aggregation

```
class A {
    classB obj;
}
```

Aggregation

→ It's a collection of

→ weak association

Composition

→ It's made up of

→ strong association

BMS

```
class Booking {
    Show show;
    Movie movie;
}
```

```
class booking {
    Ticket ticket;
}
```

class Order {

Item item;

}

class Order {

PaymentDet pd;

Transaction t;

}