UML Diagrams

Agenda

- UML Diagrams
  - → Use Case Diagrams
  - → Class Diagrams
- Schema Design
  - → Finding Cardinalities
    - 1:1
    - 1:M
    - M:1
    - M:M
  - → Types of Relations
    - → Inheritance
    - → Association
      - → Composition
      - → Aggregation

COMMUNICATION

Product Manager / Owner :

Non-Tech

Product Requirements (Understand Product Req)

⇒ Timelines

Tester

Architect / Software Eng Manager / Mentor

① Design ⟸

② Best Practices

③ Implementation =

Ways to Communicate ?

① Words (F2F Meeting, Doc) → Verbally
→ Written
→ Email

① Misunderstand / Ambiguity

② Understanding complex things is difficult

② A picture is worth 1000 words

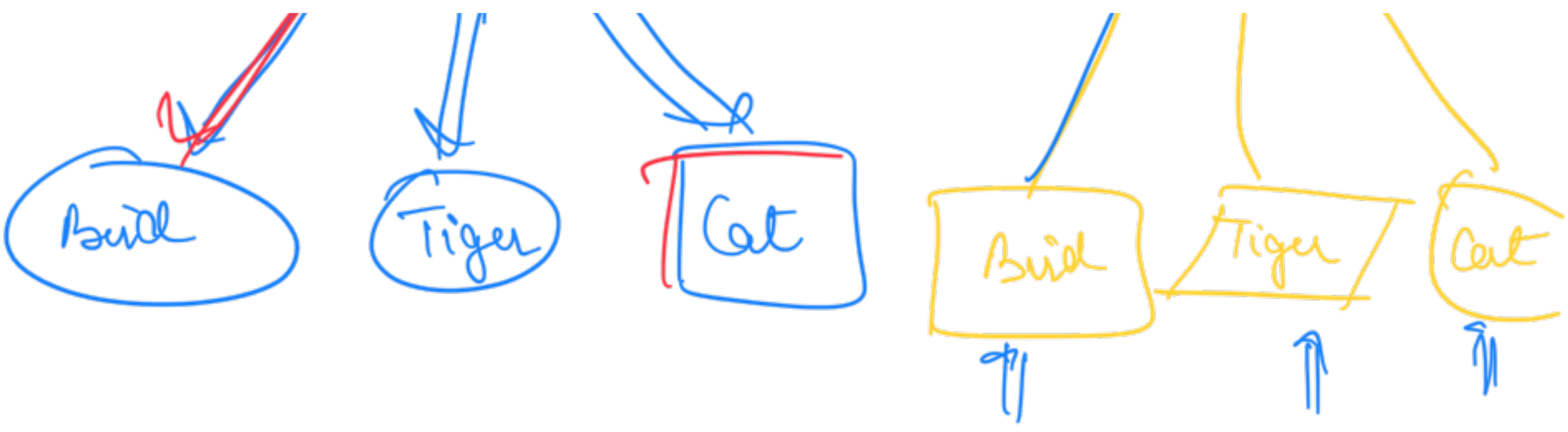
Diagrams <=

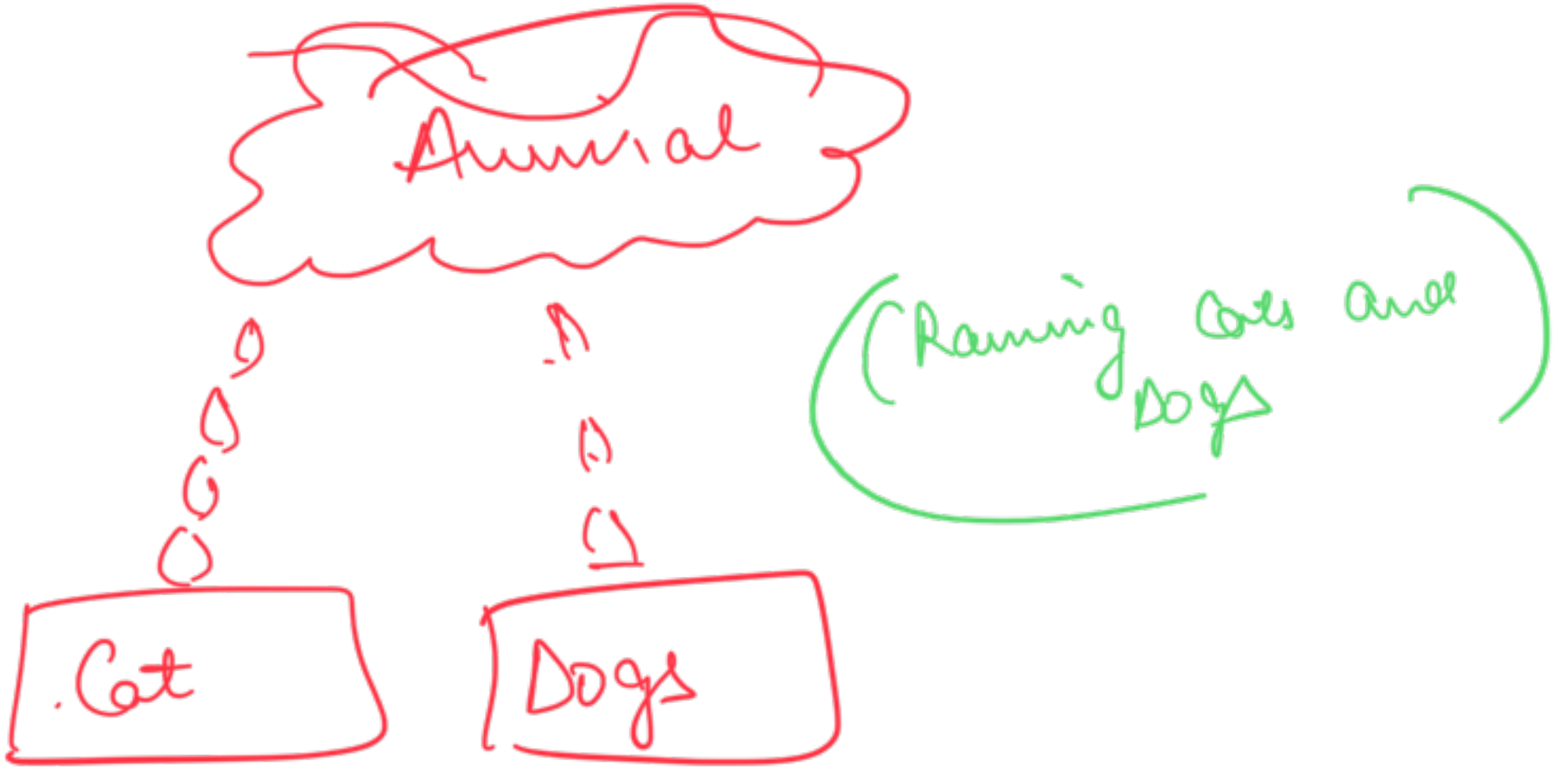
① less Ambiguity

② Easier to understand complex things

↳ Reader can
↳ Ask Clarifications

↳ Give Suggestions


Bird
Tiger
Cat
Bird
Tiger
Cat


Animal
Cat
Dogs
(Running Cats and Dogs

⇒ There is a need of Standardization

(Unified Modeling language)

→ Standardisation on how to represent different software engineering terms as diagrams.

2 Types of UML Diagrams

① Structural UML Diag → how the codebase is structured

② Behavioural UML Diag → how the system behaves

and related to each other

② Behavioural UML Diag

↳ working of a system
↳ diff features supported by a system
↳ how do those features work

Types of these

STRUCTURAL

① Class Diagram

BEHAVIOURAL

H/W

① Activity Diagrams

2. Package Diagram

3. Object Diagram

4. Component Diagram

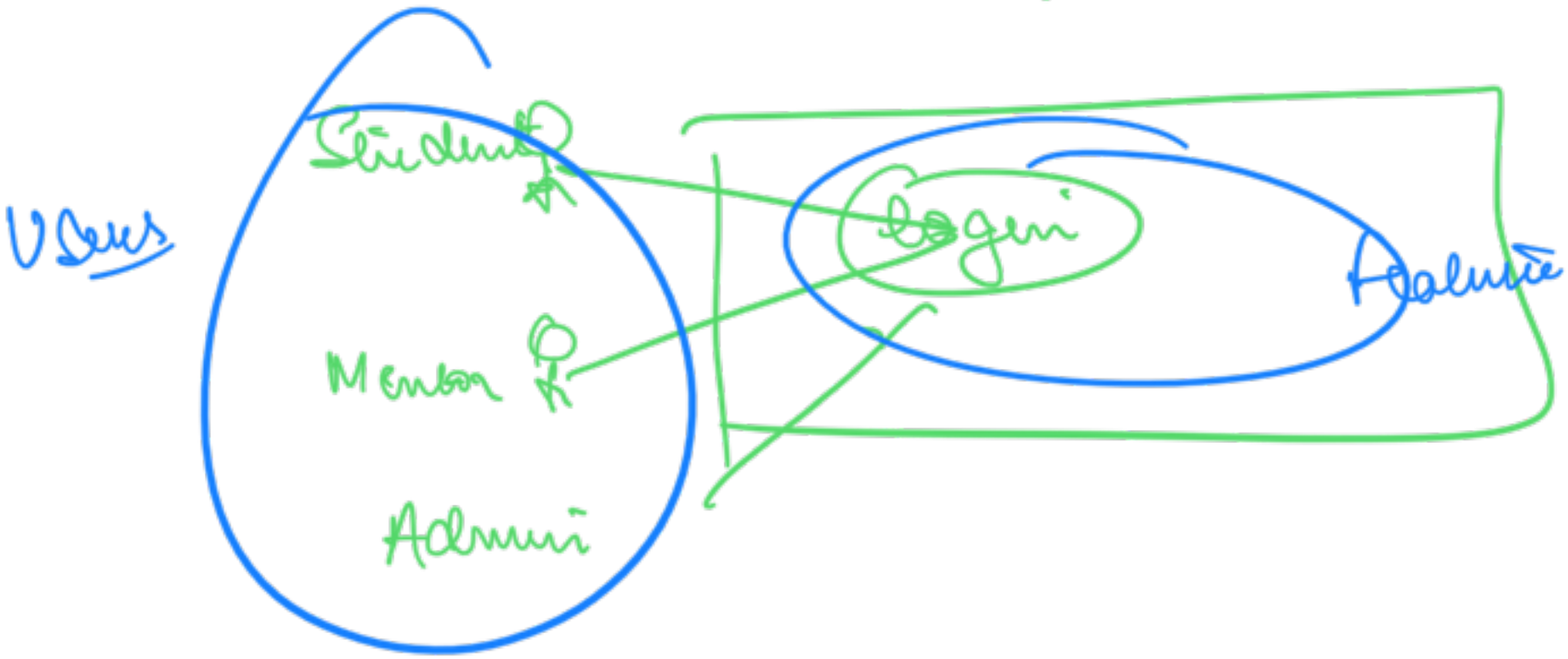
2. Use Case Diagram

3. Sequence Diagram

USE CASE DIAGRAM

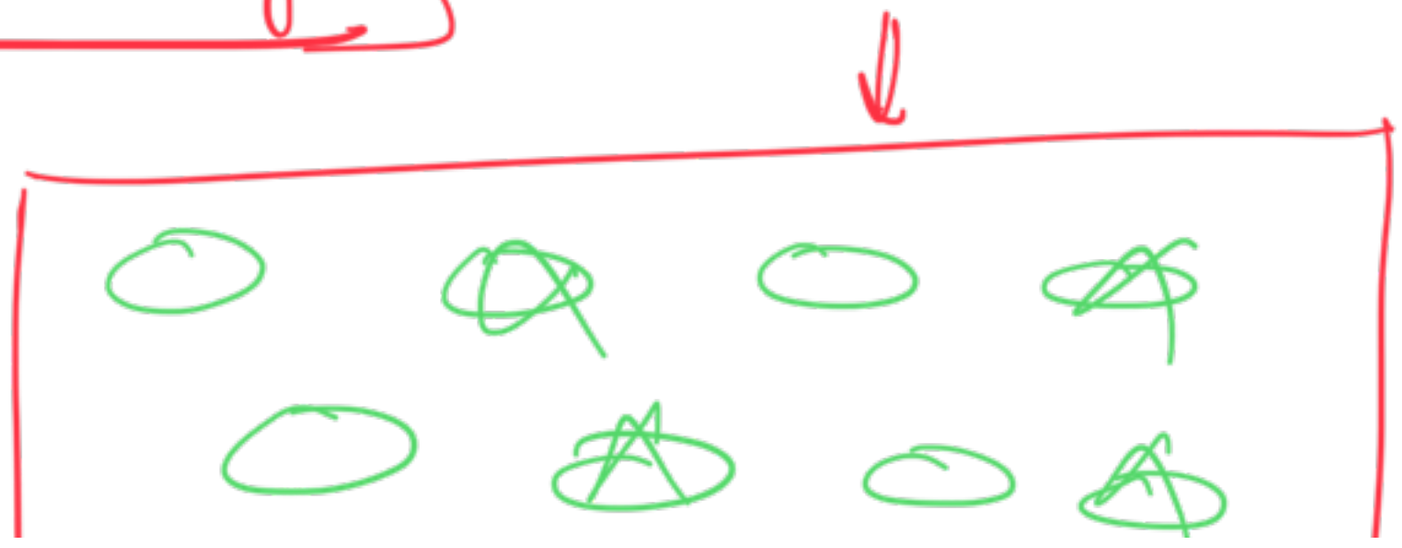
1. Feature / Functionality that a software system supports

2. Who uses these functionalities



5 Key Words

① System Boundary

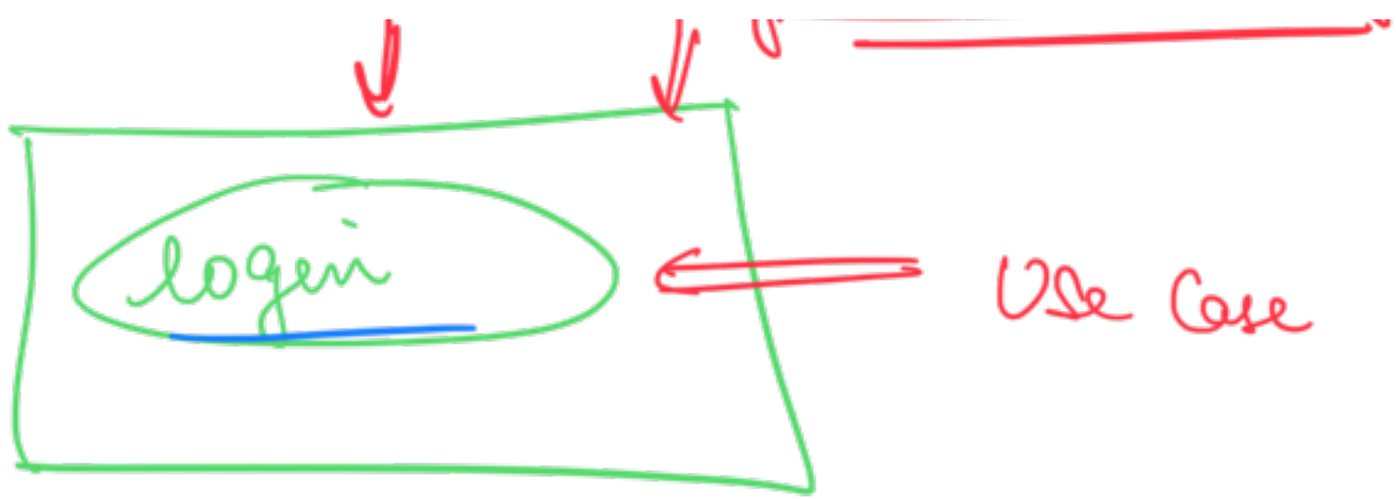


System boundary is going to contain all the use cases that are supported internally by the system

(2) Use Case (Verbs / Actions)

→ Represented as an (oval)

→ A feature that is supported by the software system



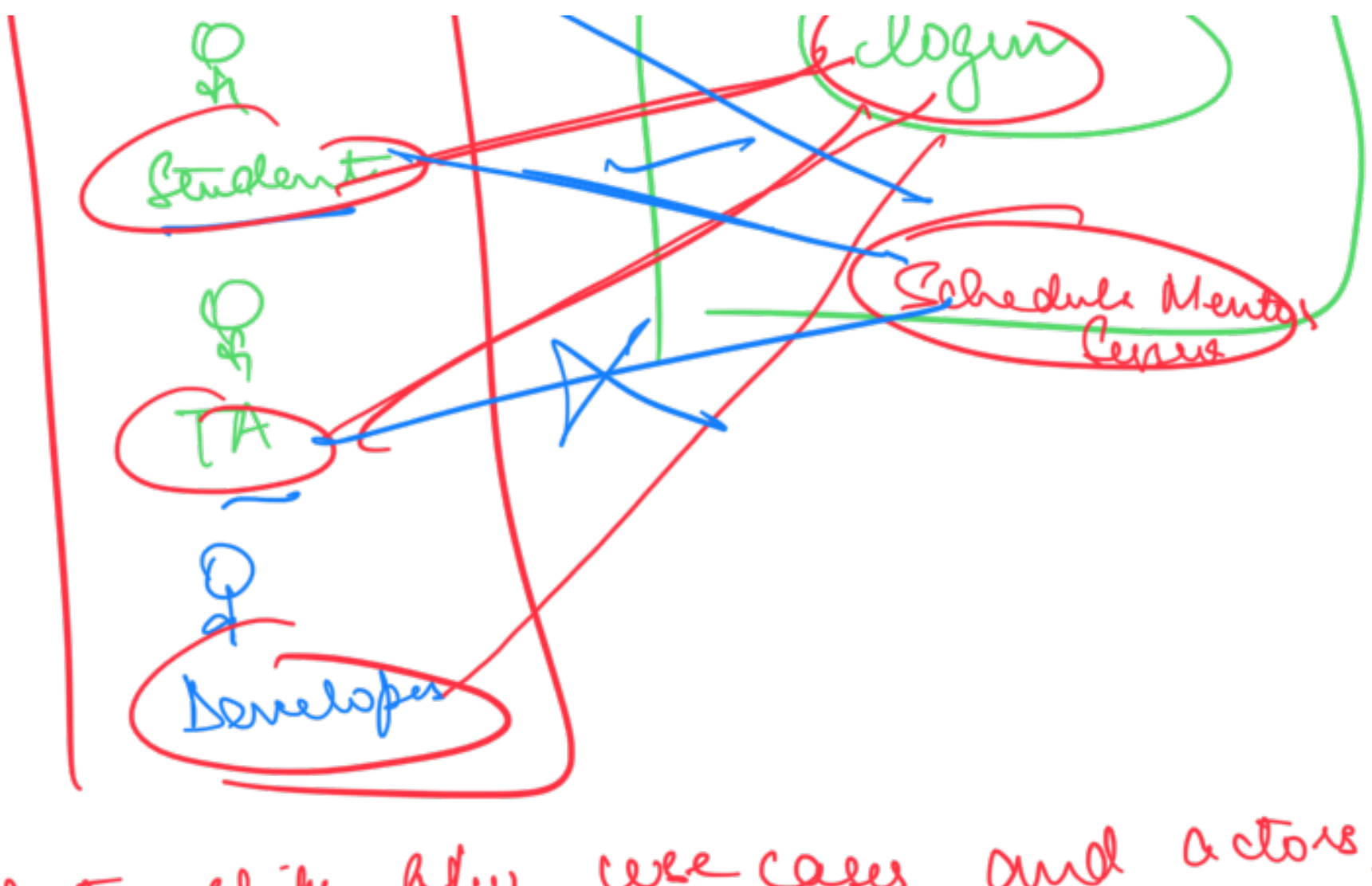
(3) Actor

Nouns

→ Rep by a stick diagram

→ User of my system

→ Type of user of the system

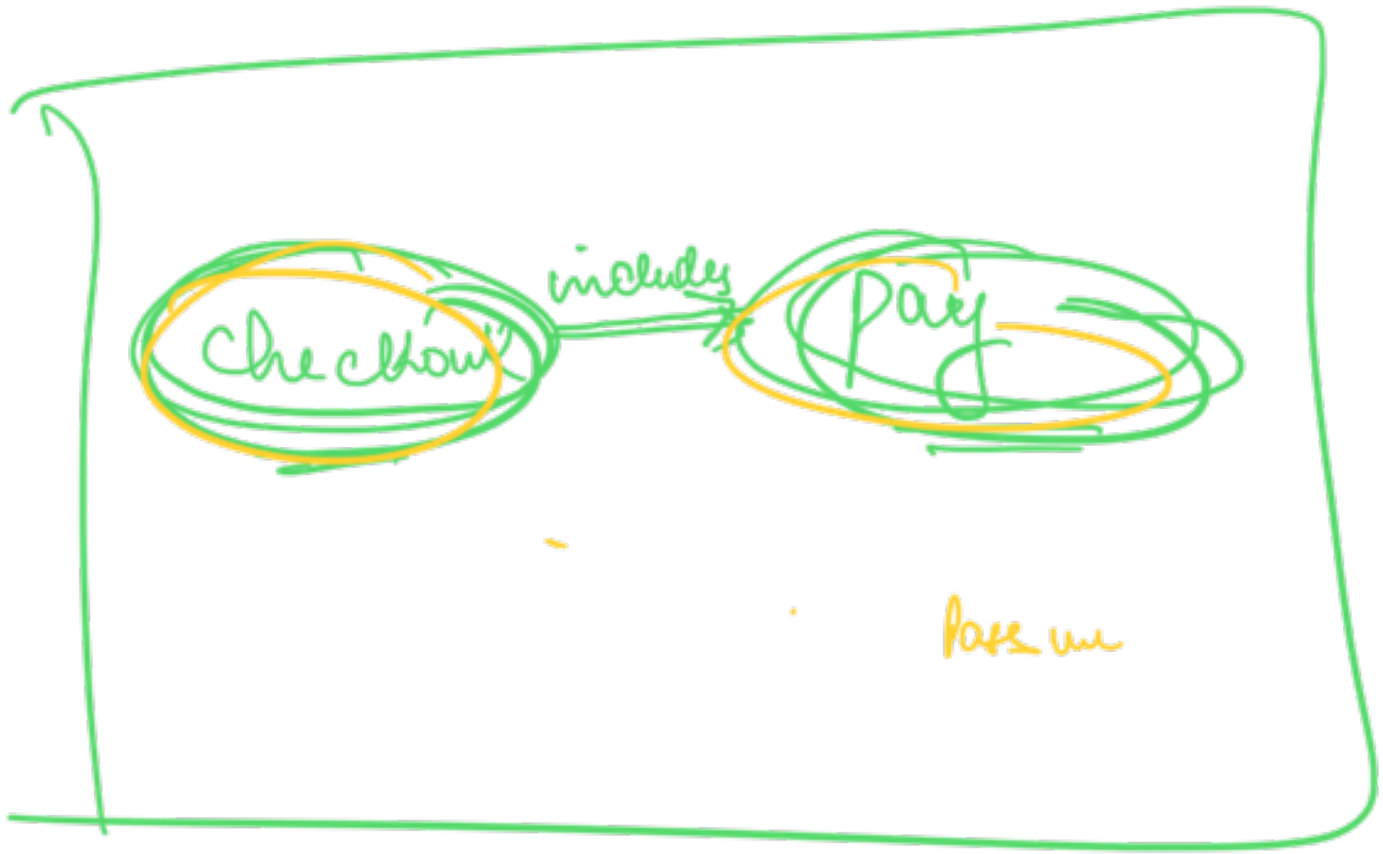


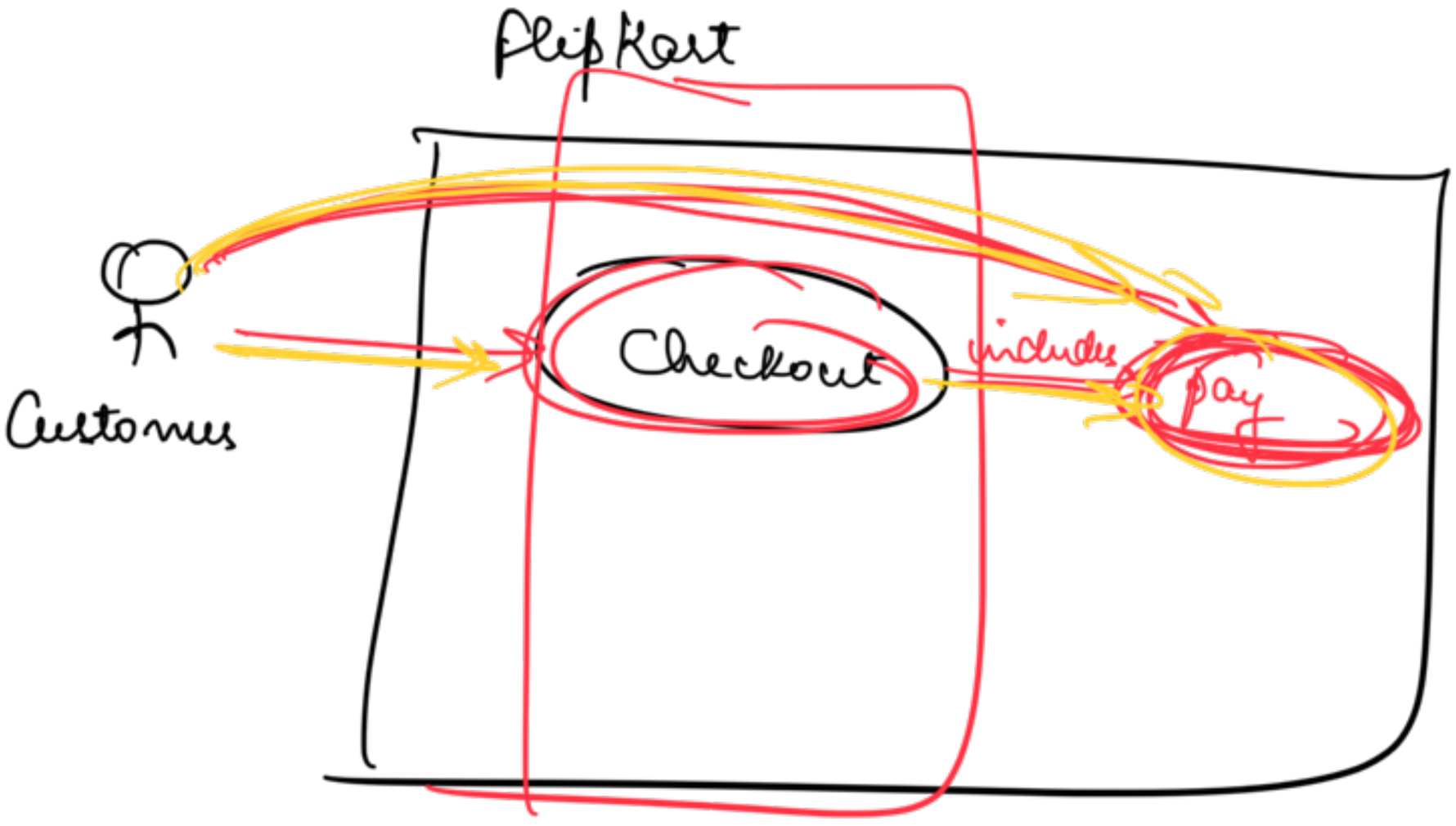
(Relationships b/w use cases and actors)

→ which actors can use which use cases

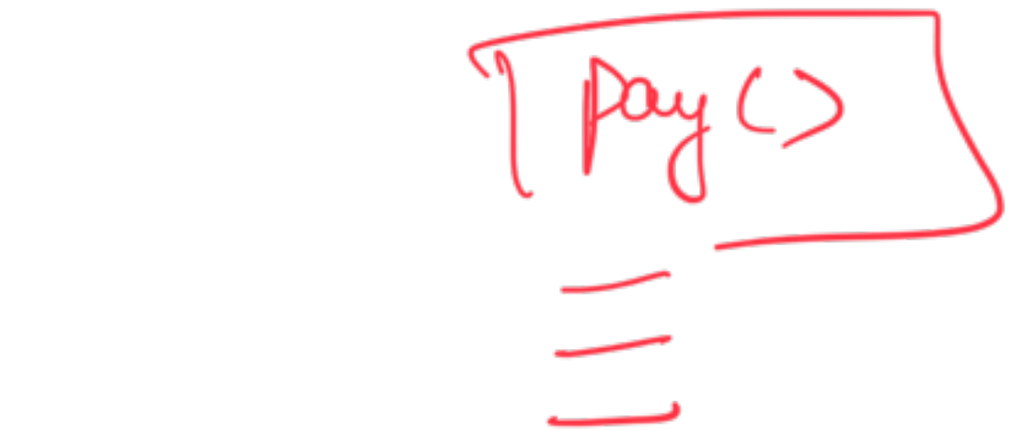
④ Includes

Often a use case for its completion might need other features.






```
void Checkout() {
```



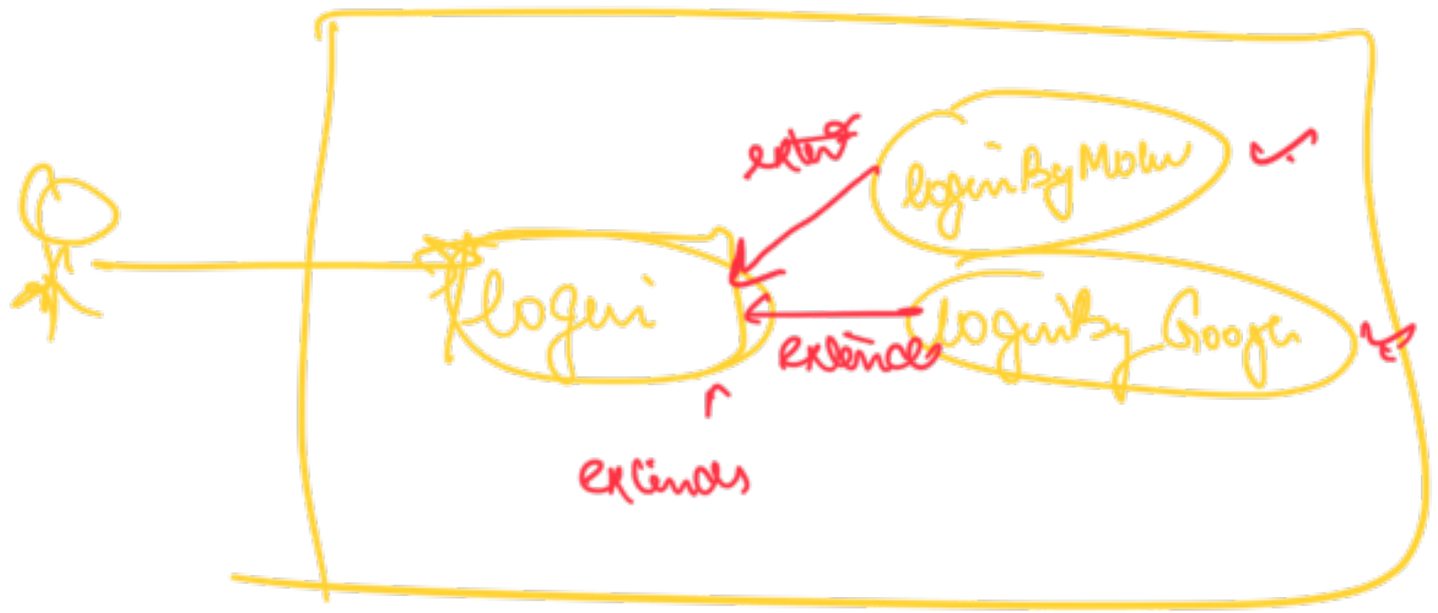
Think every feature as a f^n in the codebase

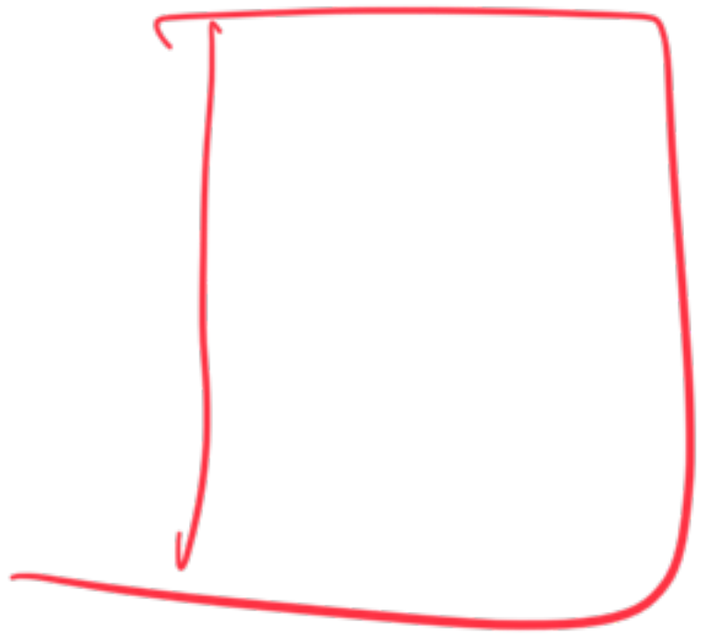
If one f^n calls another f^n
(A) (B)

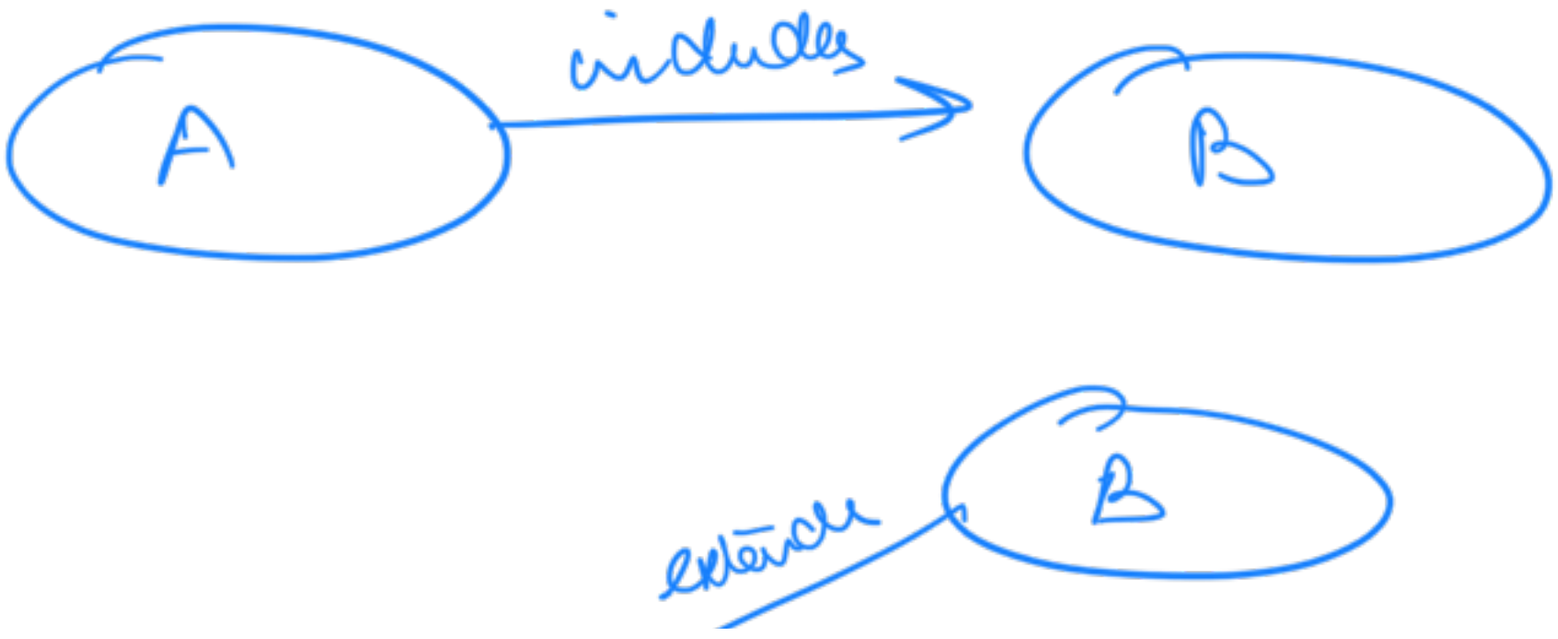
If A includes B ⇒ A can't be finished without B finish

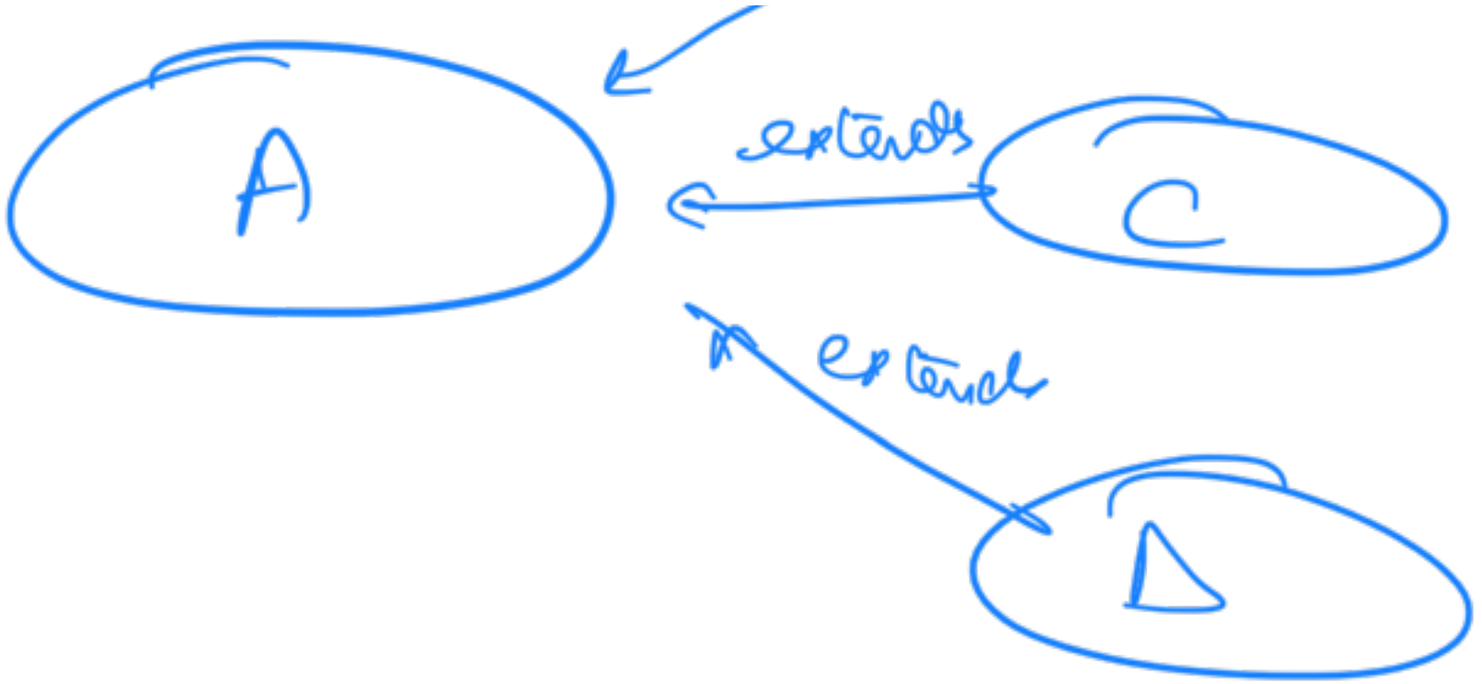
(5) Extends

If a particular feature has multiple special types, we represent each type by using extends






A
includes
B
B
extends



Assignments

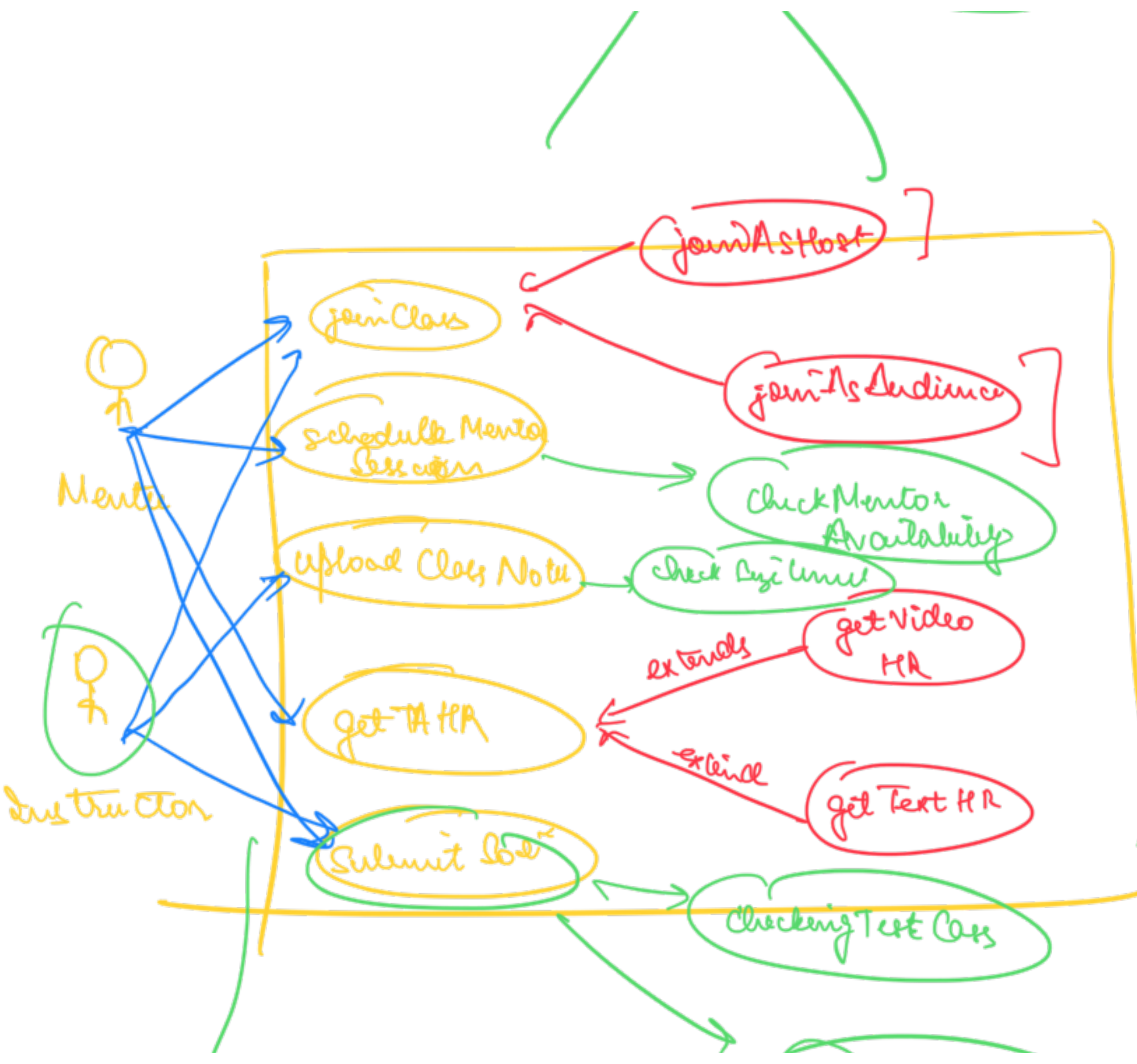
① Draw a use case diagram for Scaler's Software System

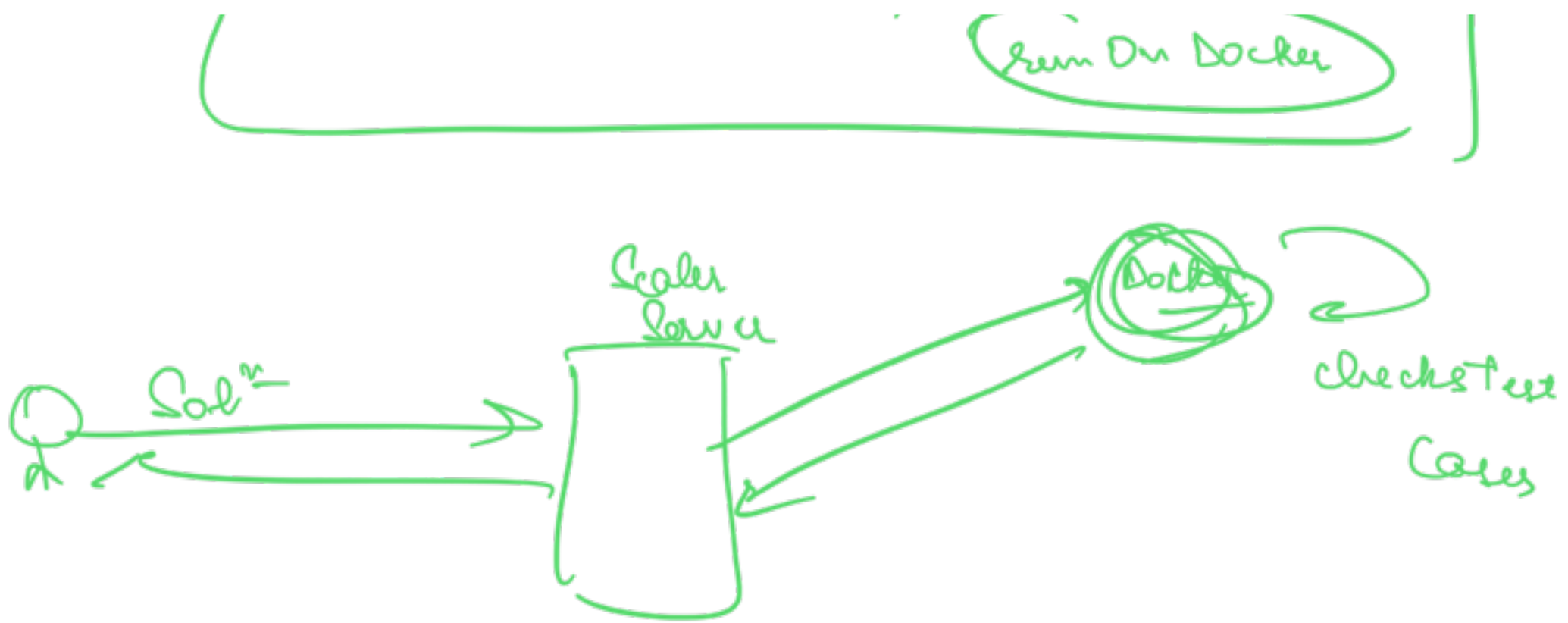
(a) Atleast 5 use cases

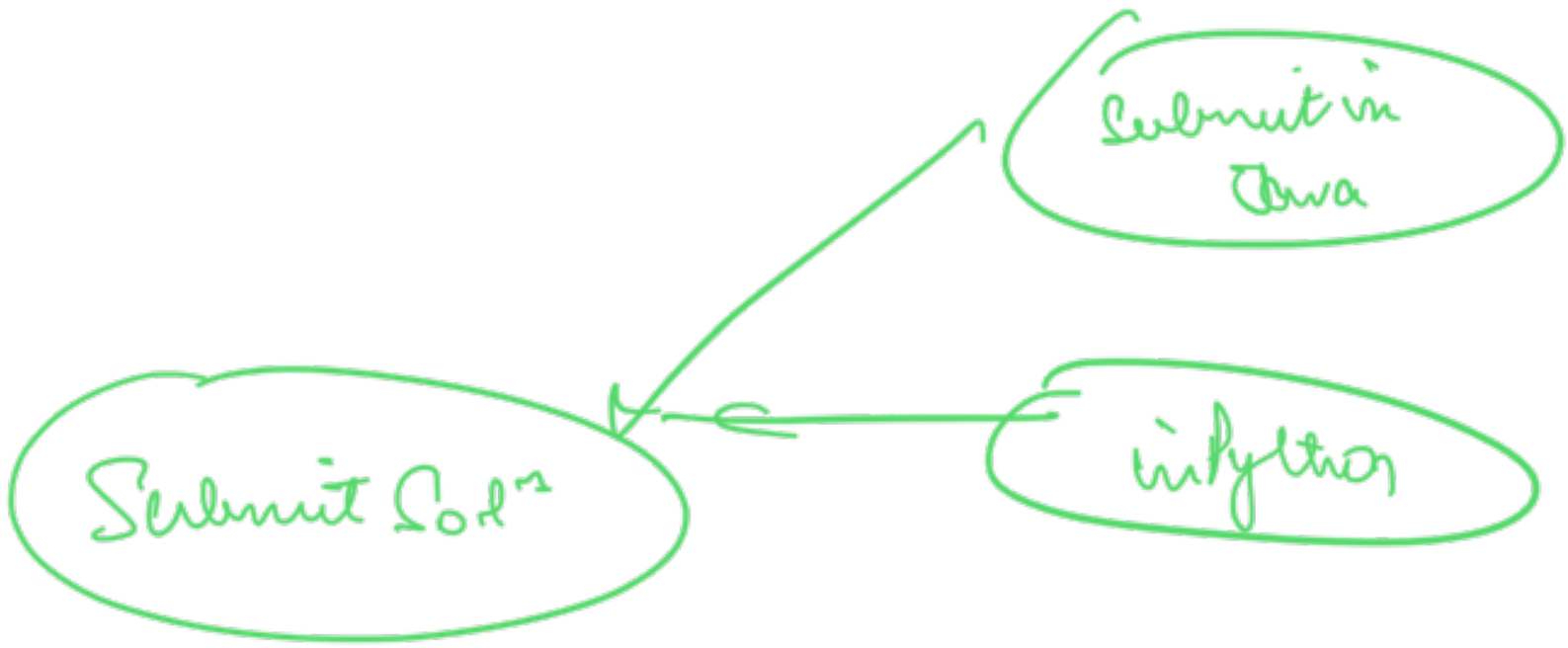
(b) Atleast 2 actors

(c) [Atleast 1/5 use case should have another use case being used (includes)]

(d) Atleast 1/5 use case should have specific types (extends) → Shouldn't be login


joinAsHost
joinClass
joinAsAudience
Mentor
scheduleMentor Session
checkMentor Availability
Upload Class Note
Check Syllabus
getVideo HR
extends
Instruction
getTA HR
extends
getText HR
Submit Test
CheckingTest Cases


Run On Docker
Scala
Server
Soln
Docker
checkstyle
Codes

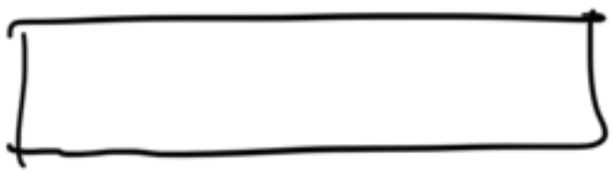

Submit in
Java
Submit Soln
Python

Schedule
mentor
Session
End
Mentor
Session
includes
Get
Feedback
Solve
Any
Problem
Solve

Problem

Solve HW Problem

UCD

① Sys Boundary 

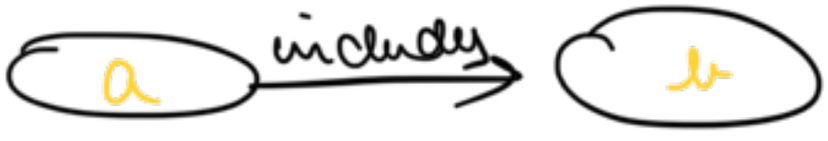
Contains all internal use cases

② Use Case 

Verb → Feature

③ Actor 

Noun → User

④ includes 

a() {

b()

⑤ 

extends

Multiple types of a feature

CLASS DIAGRAMS

① represent diff entities in the system

- → Class
- → Interfaces
- → Abstract Classes
- → Enums

② and rel^{n} b/w these entities

- ⇒ who implements an interface

② who is parent class

③ who is child class

④ which class is an attribute of other

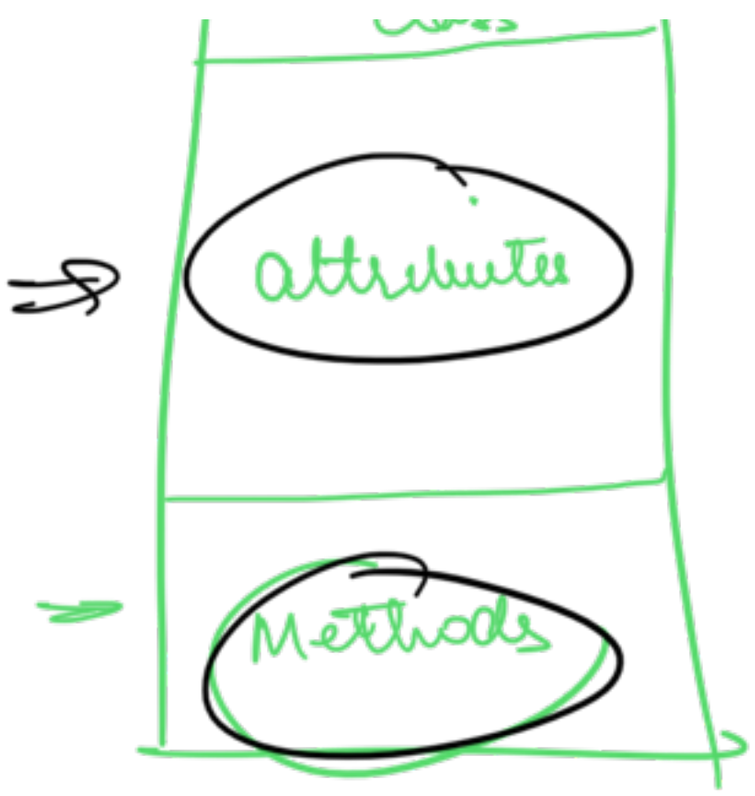
Flying Bullet

Bullet b

Class

Name of Class

Student



attributes

age : int

methods

public void ChangeBatch(String newBatch)

public boolean pauseCourse (Date endDate)

+ pauseCourse (Date,): boolean

Interfaces

<InterfaceName>>

<< Flyable >>

fly ( ) : void

Name (Data Type of Param) : Return Type

<<Some Interface>>

someAttr : int

← underline denotes static

fly() : void

doSomething() : void

Main

main() : void

Abstract Classes

Like a Normal class with following differences

Animal

⇒ Anything abstract is up in Italics

⇓

Class Name OR Class Method

Enums

How to rep rel^n btw diff entities

2 types of rel ship entities

1. Inheritance (IS-A)
2. Association (HAS-A)

Inheritance

1. When we are extending a class
   ⇒ Parent Child Rel^n
2. When we are implementing an interface.

(B) Represented by a

IS-A

IS-A

IS-A

Penguin

Pigeon

Flyable f = new Pigeon()

Association (HAS-A)

assoc^{n} is also of 2 types

1. Composition
2. Aggregation

Composition

Created By

Aggregation ⇒ Collection

If one class has - an attribute of another class ⇒ Association

A → B

Scenario 1

COMPOSITION

Objects of B don't make sense without

Scenario 2

AGGREGATION

Objects of B can exist

Object of A

Objects of B can't exist independently of A

If A dies, B will also die

Strong Association

Class → Assignment

Composition

independently of objects of A

If A dies, B will still remain

Participant

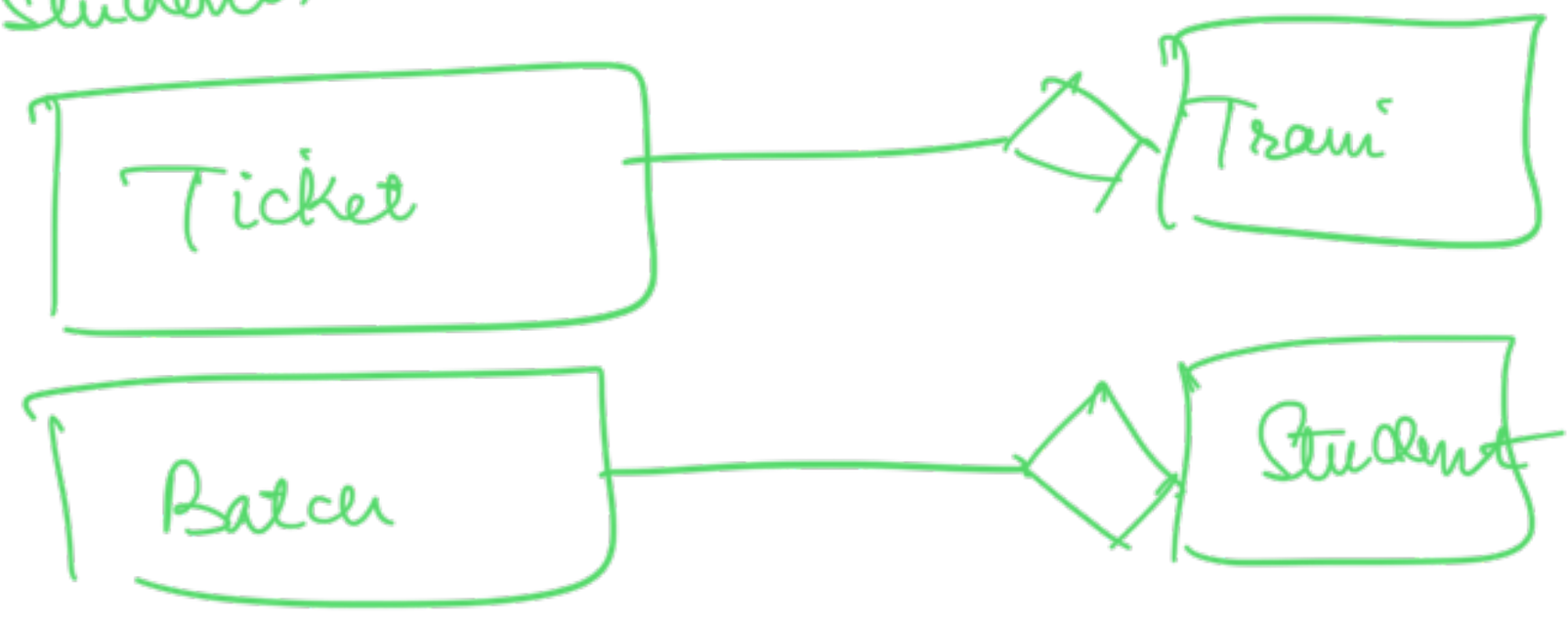
Weak Association

Aggregate
Question

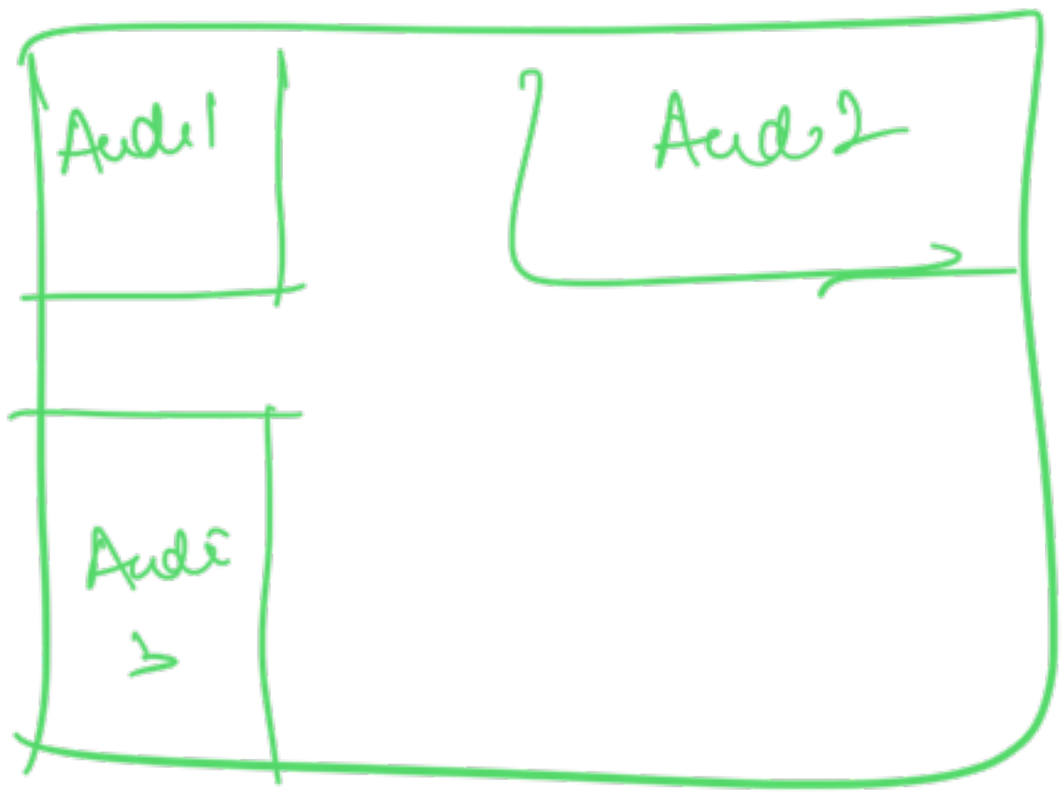

A
B

Batch £

List < Student > Students




Cinema


Audi 1
Audi 2
Audi 3

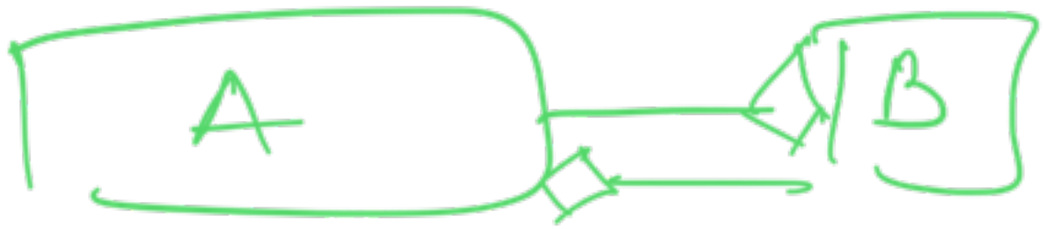

Train

Aggregation

Aggregation

class A {

B {



}

Class B {

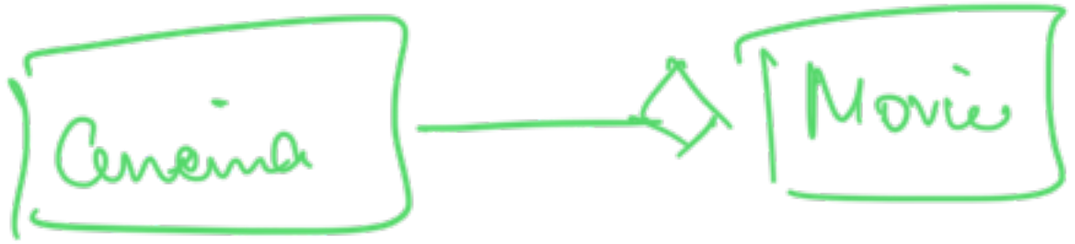
A a;

}

Movie ?

Cast <Cameras>

{



Next class

Cardinality of rel^{n} & Schema Design in next

Machine Coding
Dist Cache

(1, 2)

= (1, 3)

1. Write Back Cache
2. Write Through Cache

10 ques^n in assignment