LLD - Class Diagram and Schema Design

Agenda

* Assignment
  - Usecase diagram
  - PlantUML
* Class Diagrams
  - Mermaid
* Schema design
  - Case study

Requirements

↳ Tech Designs

↳ DFD

↳ Data flow diagram

→ Approaches

→ Use case diagram

→ ① Class diagram

→ ② Schema

→ APIs

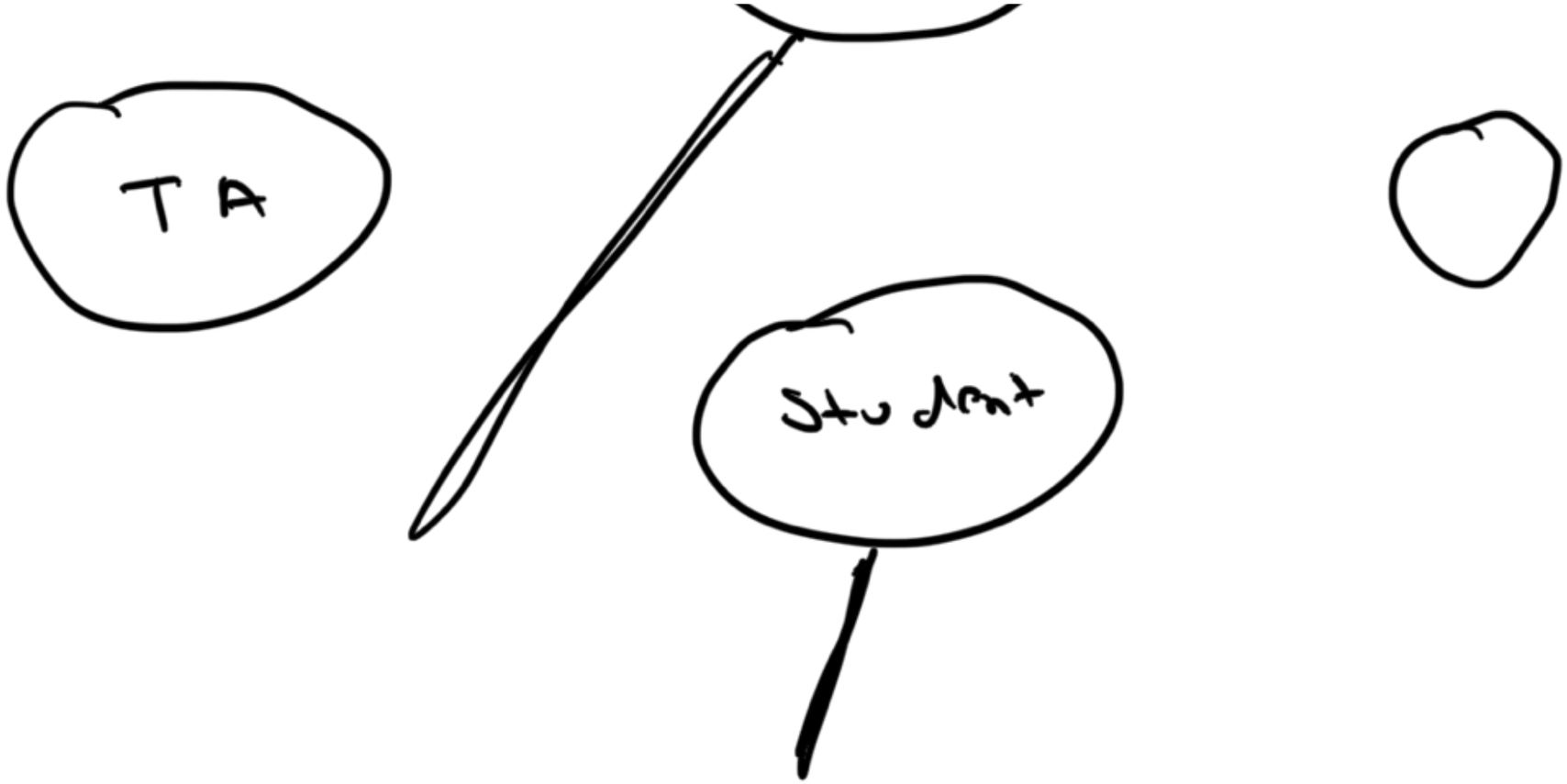
→ contact

→ Examples

→ Postman

DFD — contact

DFD1 Data flow diagrams



Microservices

modules

Actors

* Student

Re Scaler

* Instructor
* TAs
* Refual manager
* Mentor

Use cases

* Student - join class
* Student - apply for a job
* Student - solve assignment
* Student - schedule

Student — Schedule meeting

session

Instructor — conduct a class

Instructor — Create Poll

— Schedule classes

Extends

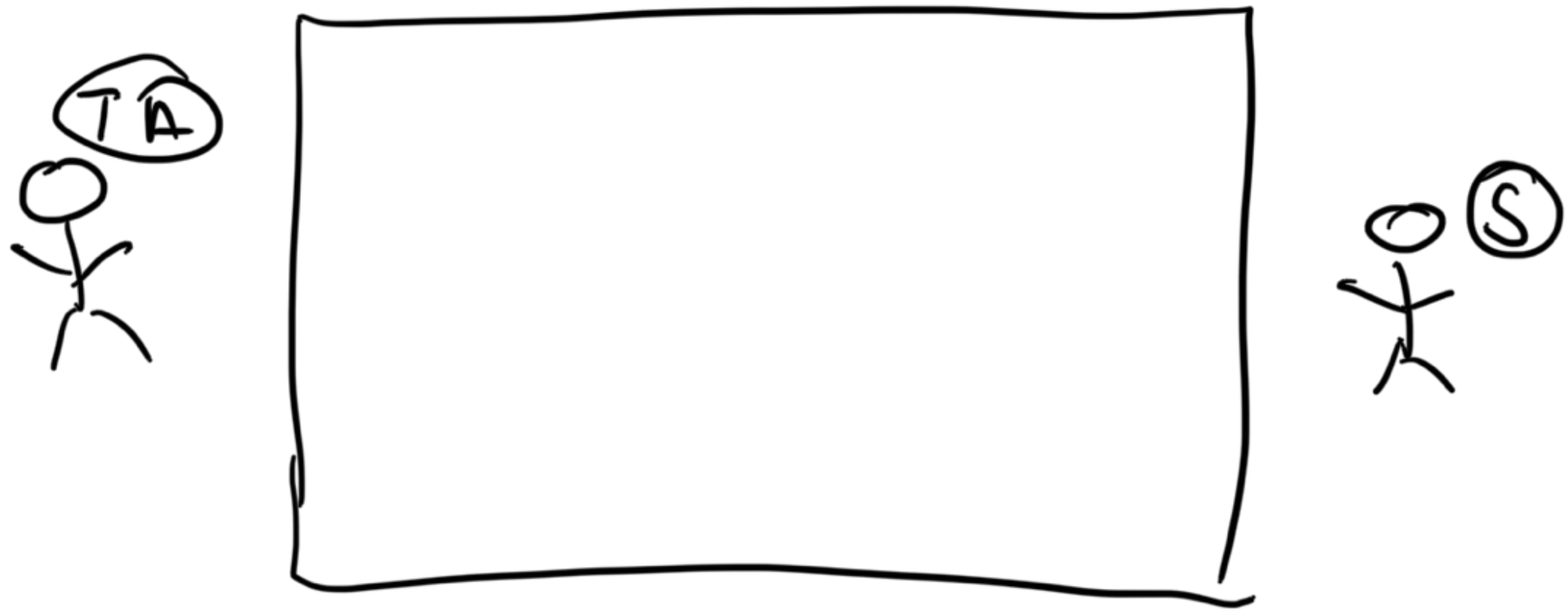
Join class — Join as host

— Join as audience

* Solve assignment
  - mcq
  - Coding assignment
  - Job assignment
* Conduct° class - master class
  - batch
  - AMAs

Includes - Usecase1 has to complete in order to complete usecase 2

- book a mentor session
  - check mentor availability
- Apply for a job
  - Solve their assignment
- Raise a TA request
  - check TA availability



Break

5:43 – 5:47

= 10:17

Class Diagram

– Blue print of LLD

Entities

Object

- Class
- Interface
- Abstract
- Enums

Packages

Person

→ Sherlock
→ Static

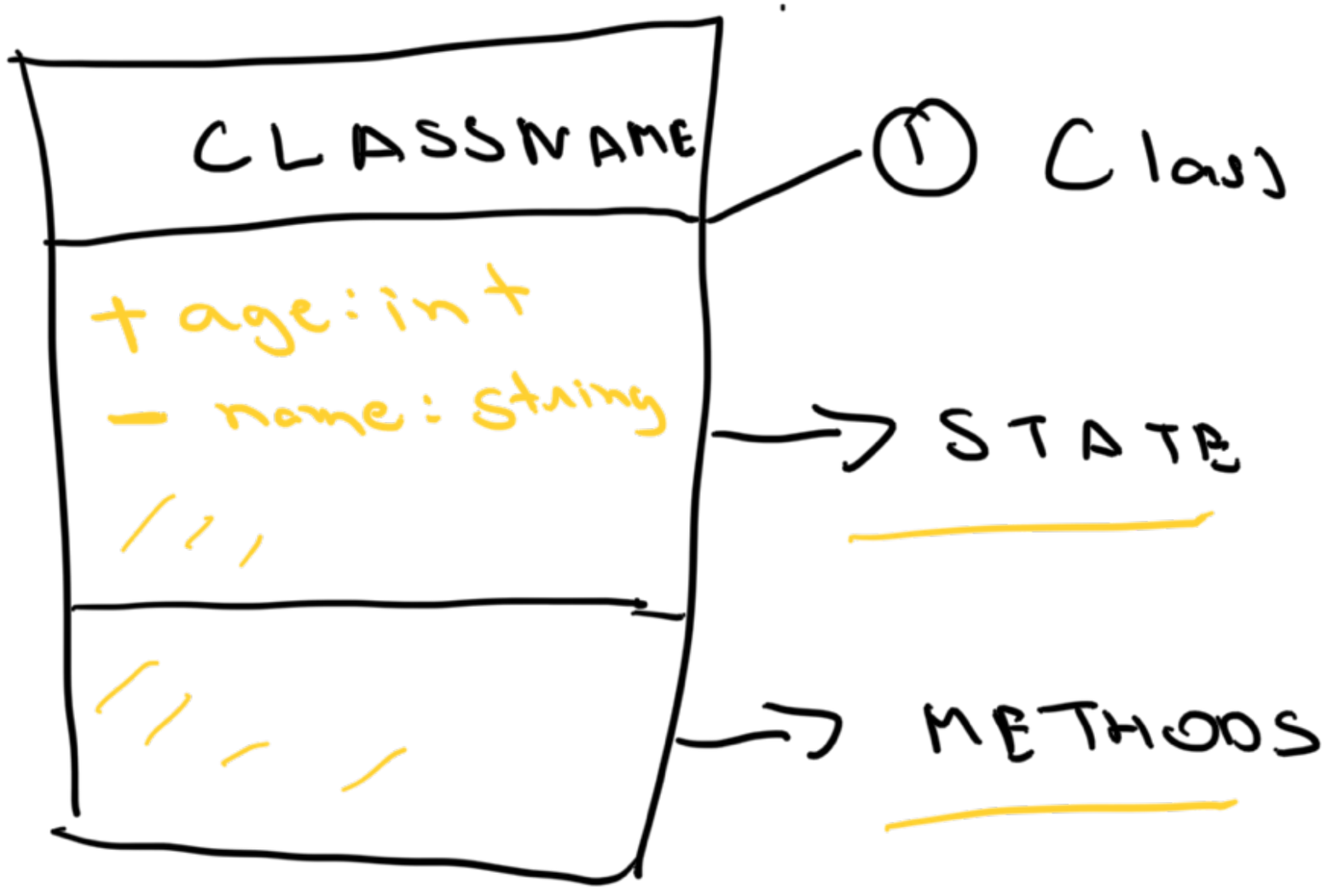
John

Relationships

- implement an interface
- parent class
- child classes
- if a class has another class as attribute

Class - state — attributes

- behaviour - methods



A Attributes

data name visibility

type

private String name

[access-modifier] [name]: [type]

+ name: string

→ + (public)

-> - (private)

-> # (protected)

-> ~ (package) ~

TILDE

protected int age;

age: int

int age

Methods

String a, int b

private String methodName (
parameters)

[access] [name] (parameters): [return]

- methodName (string, int): string

public Int getName(int id, String parameter)

↓

t getName(int, String): int

Static methods - underline attributes

int[ ]

string [ ]

Interface

« name »

<<interface>>

Abstract classes | methods

L italicised <i>

Player

italicise

Mermaid - class diagram

Relationships

- inheritance
  - IS-A
- Association
  - has-a

class A {

Association → Composition

↓

Aggregation

→ lifecycle

Composition

→ child cannot exist w/o parent

→ book → page

A & B → composition

↓

Inheritance

Association

↳ Composition

↳ Aggregation

↳ A {

b b

}

A @ = new A(b);

Player → User

Batch → Student

Aggregation

Relationships

- Inheritance - is - a

Player → Playable

- Association -

او

- Composition

- child can not exist w/o parent

- Aggregation

child can exist w/o parent

Batch —— student

Player {

User user;

}

aggregation

Cardinality

House — Floor

١ : ٣

٣
١

Student — Mentor

M

M : 1

Student → Class

1 M

M : N

husband | wife ← 1:1

husband
id wife fk

1
2

wife
id husband fk

1
2

1:1 - any side

1 : m en n:1

M

Student

id	mentor_id
1	
2	
3	

1

M enter

id	student_id
1	[ ]

Many to many

class → new table

PK - composite pk

Schema design 3 (2)

= Design a pen