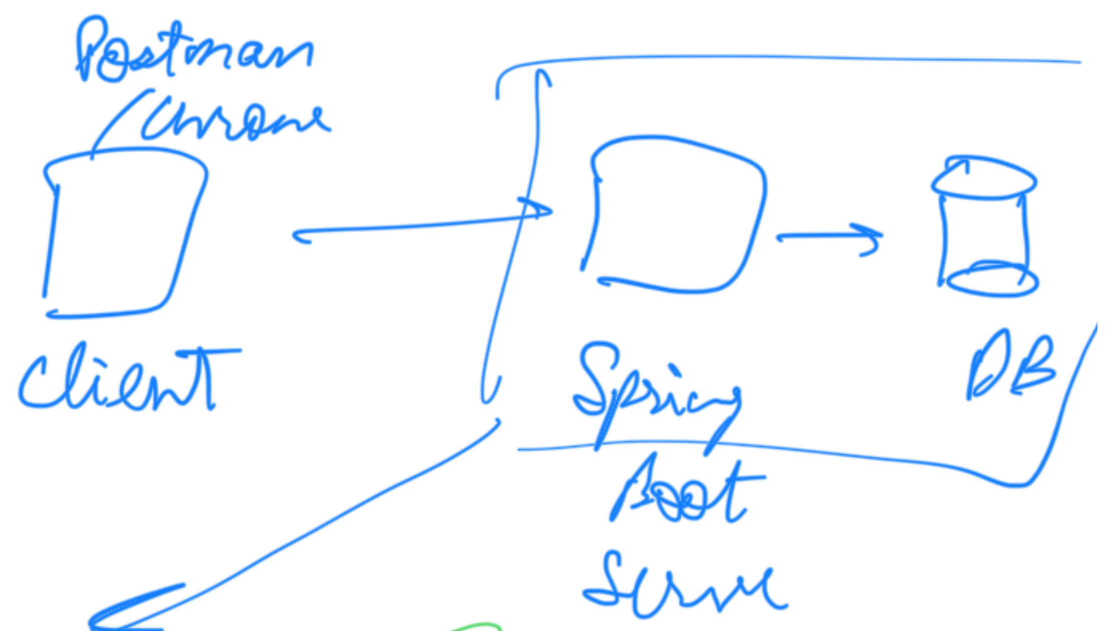
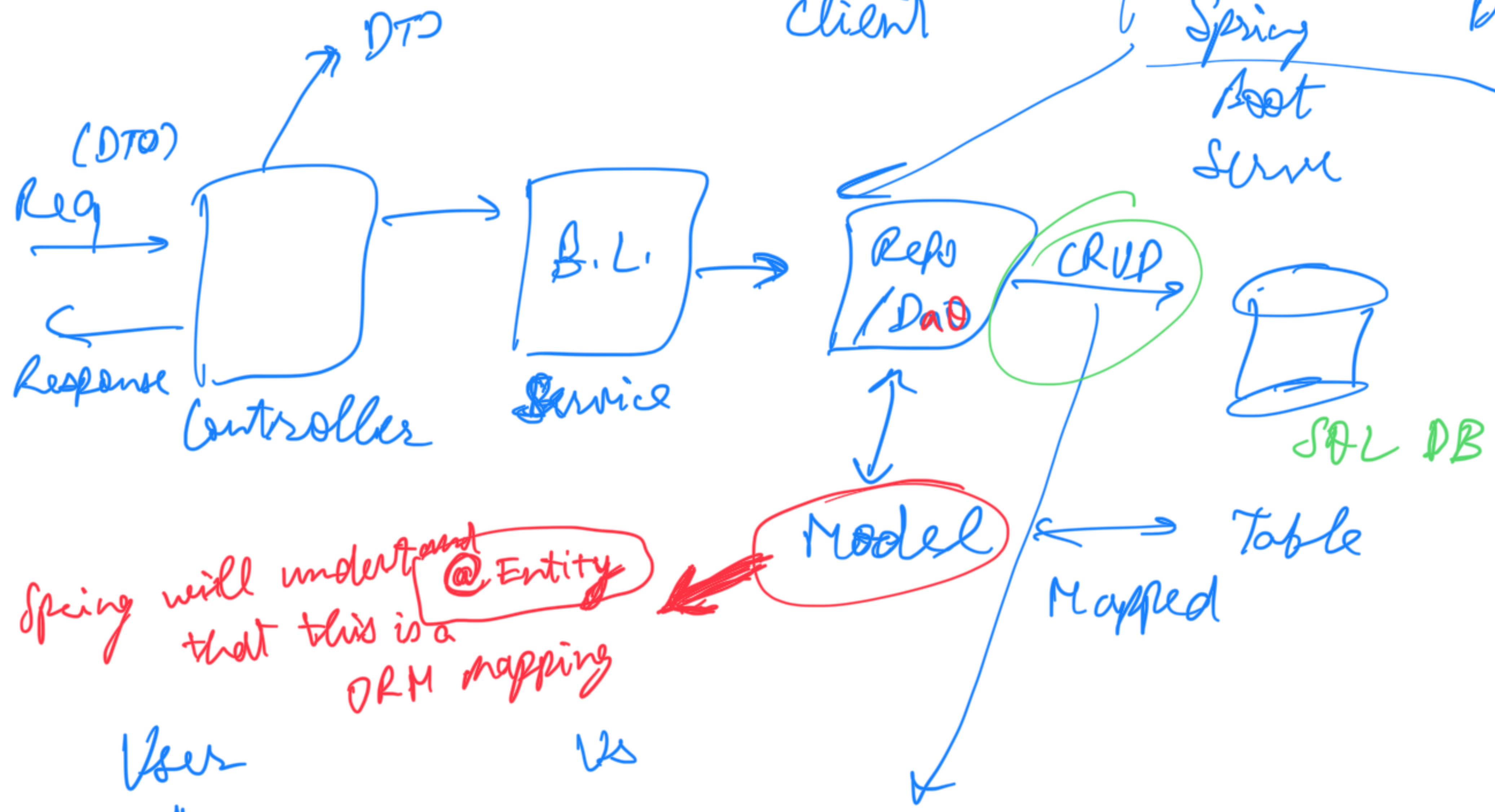
Class → 13

JPA

→ Connected to a DB





@Entity

Spring will understand that this is a ORM mapping

Users
↓
Name
Id
Address

Vs

→ SQL Statements

Select *

inert
Update

JPA
↳ JAVA Persistence API

Hibernate →
→ Implementation of JPA
→ ORM (Object Relational Mapping)
→ framework that provides ~~in~~ imp. of ~~JPA~~ API's (JPA) to interact with DB

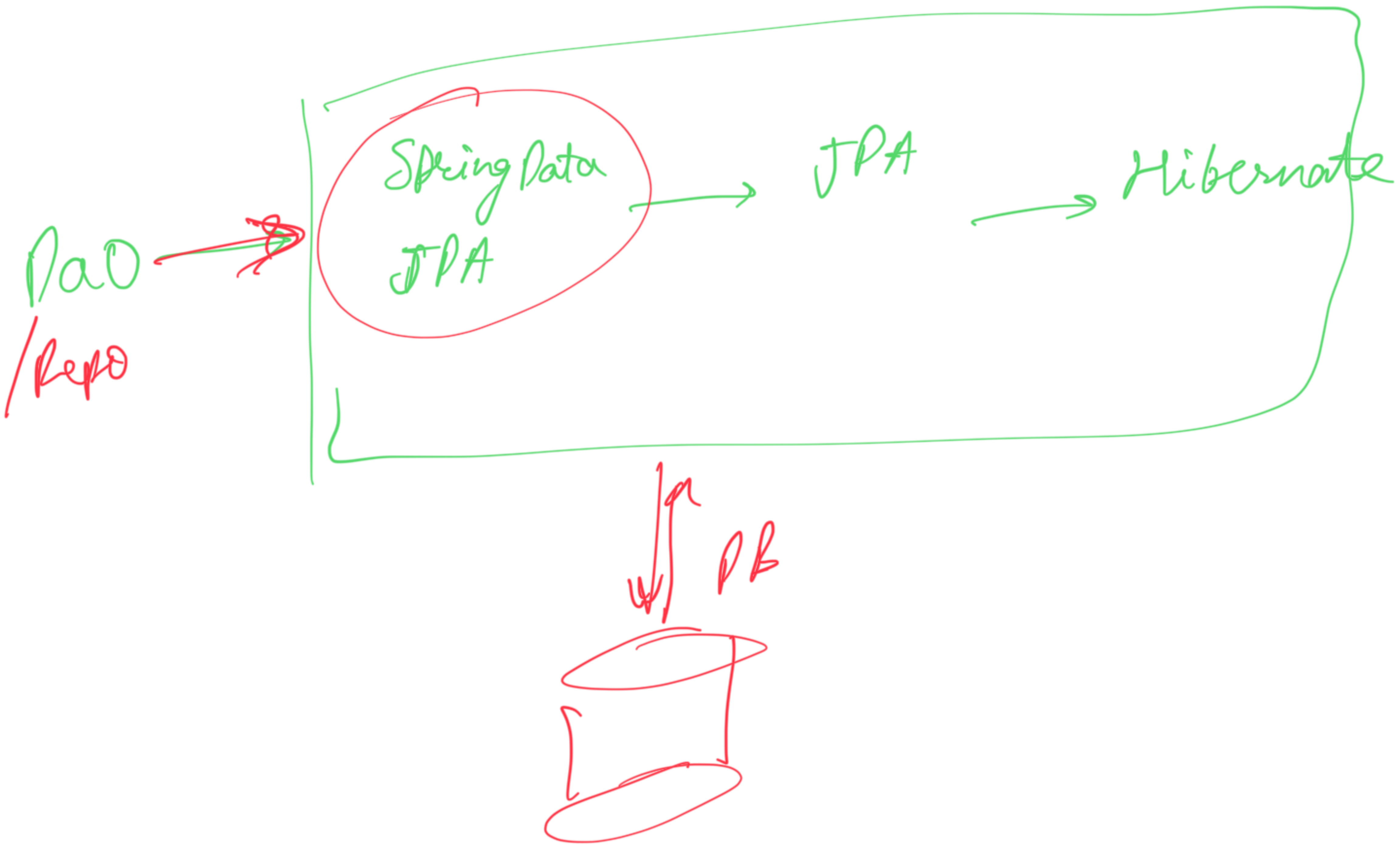
Interface

JPA
↓

Concrete Class

Hibernate (ORM)

OpenJPA



Spring Data JPA →

Springdata

① provides abstraction to repo API's

② Automatically generate methods based on params fields

get

Find all the users whose name = "akshay"

⇓

Key

Value

find (name = "akshay")

get

⇓

findByName

findById

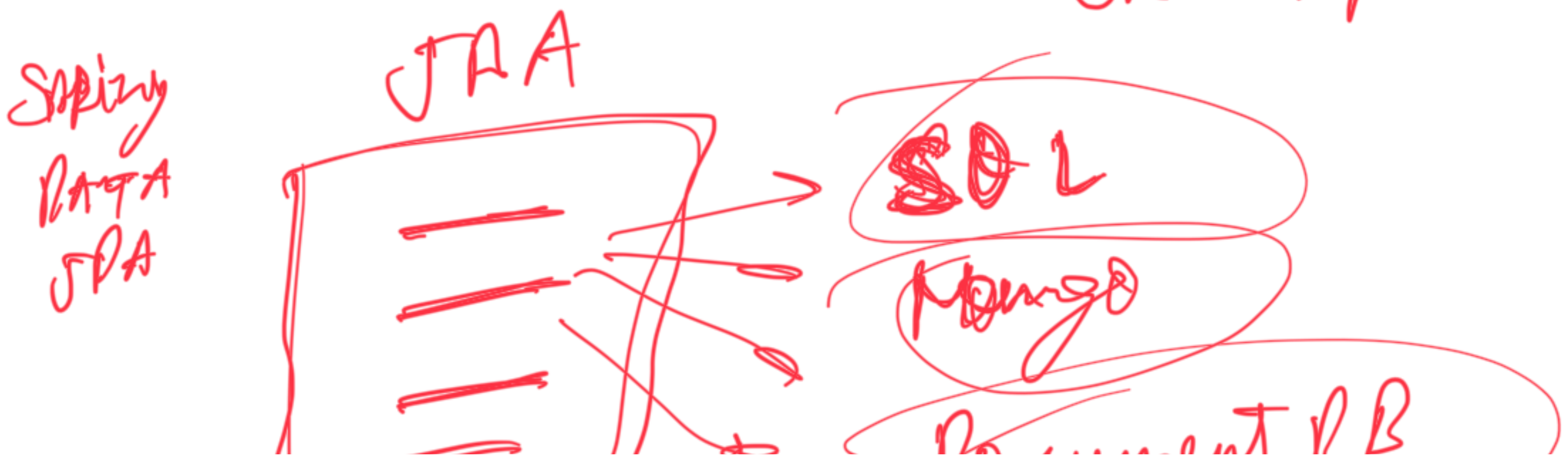
findBy(x)(value)

③ Custom Query @Query

JPA → JPQL (Jakarta Persistence Query Lang)
(SQL)

→ ~~Abstraction~~ Specifies entities & relationships and queries

ORM implementation



Contract
/ interface

dependency → spring-boot-starter-data-jpa
(Spring data JPA)

Why even Use ORM?

1 Mapping Model and DB ~~table~~ Schema
Manually → very difficult

2 Not scalable

ORM →

save()
find()
update()
delete()

JPA layer

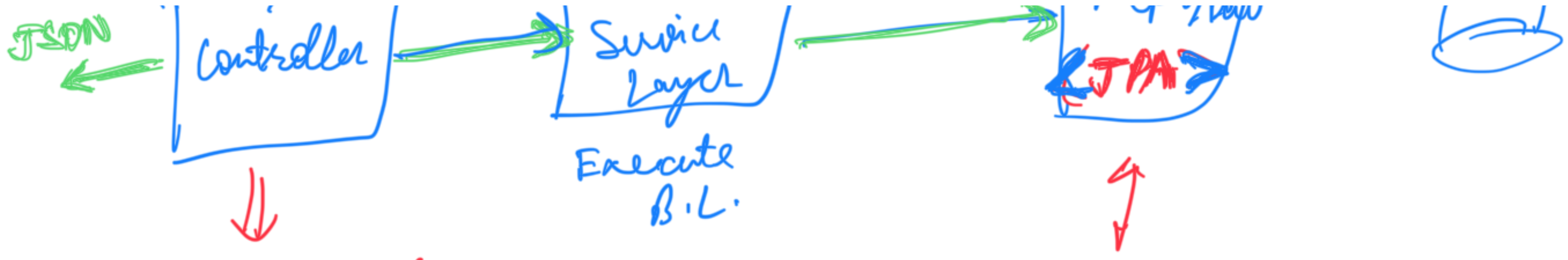
JDBC API | JDBC driver

custom properties / configs

⇓

to connect with DB

application.properties

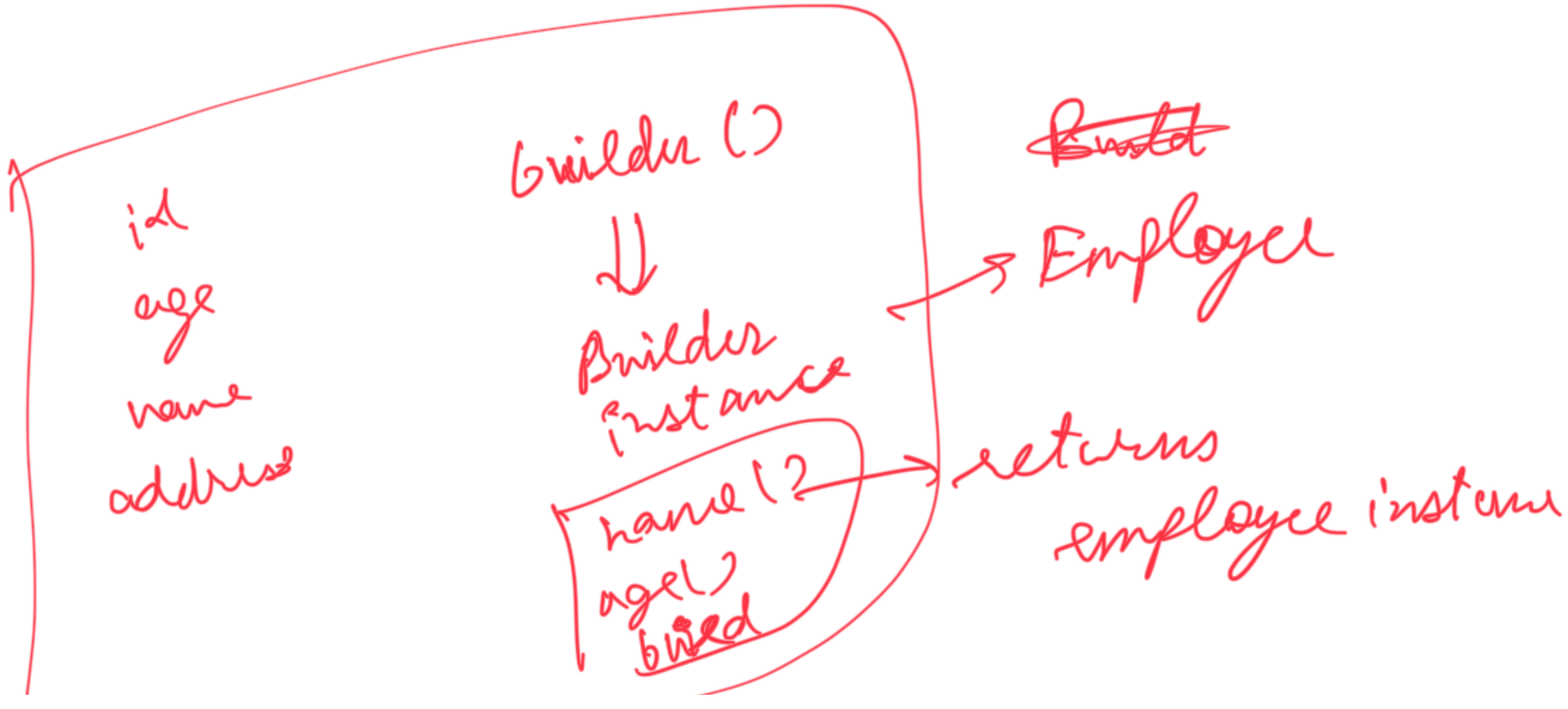


→ Validations
→ Req / Resp
Transformation
DTO ⟷ Model

Response Entity

Model

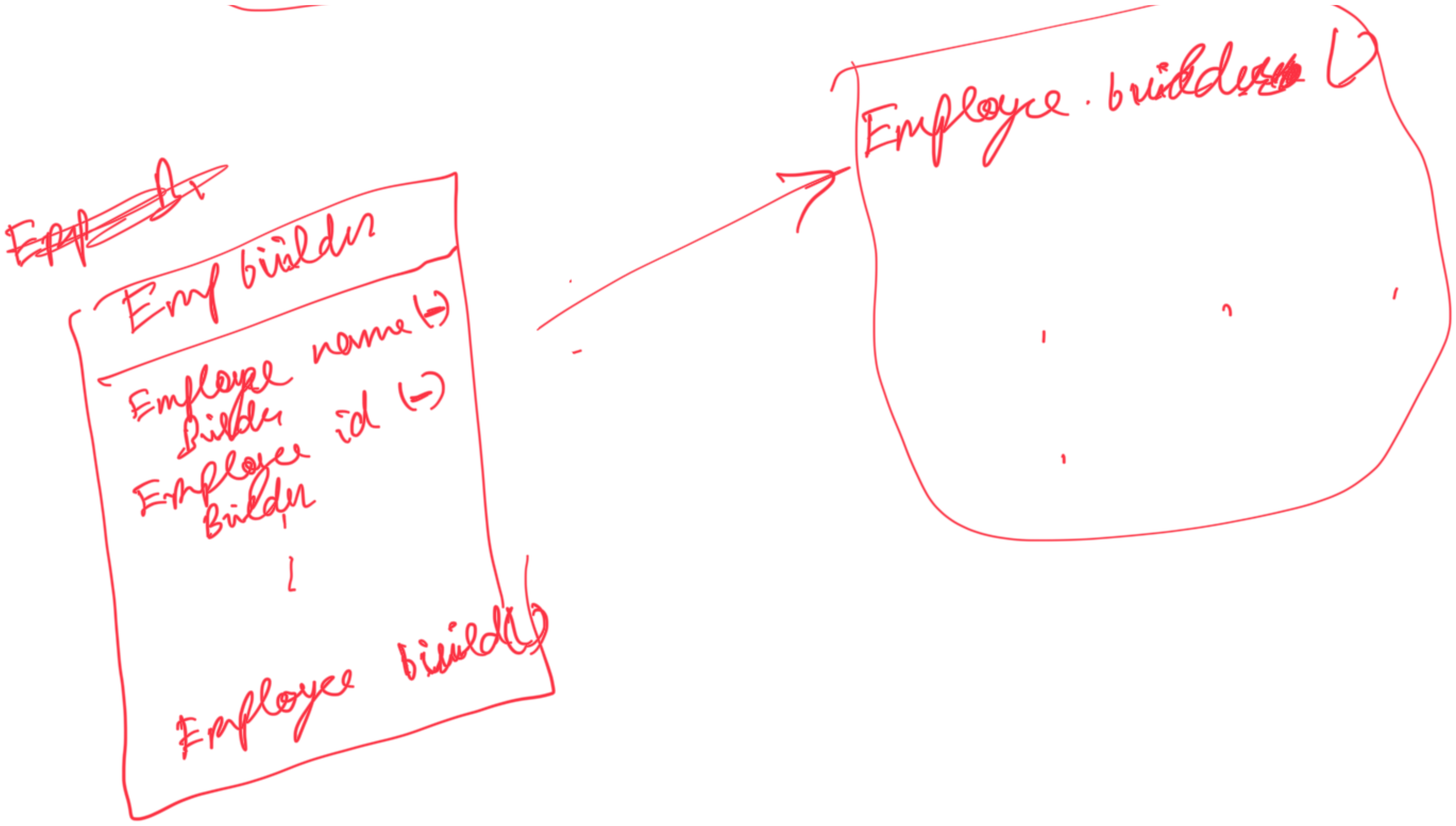
Interacts with DB
(JPA + ORM)



```
Employee.builder().name()
                  addres()
                  age()

Employee name(—){
    this.name=(—)
    return this;

Employee id(—){
    this.id = —
    return this
}

Employee build(){
    return this;
}
```

@ Builder

lombok


~~Emp Bu~~
Emp builder
Employee Builder name (-)
Employee Builder id (-)
Employee build()
Employee.builder()