LLD - Strategy Design pattern and UML diagrams

1. Strategy
   - Design a bird

2. UML diagrams
   - Types
   - - Use case Diagram
   - - + Class diagram
   - Assignment

Command
- input from command line

Strategy

→ penguin
- LSP

Behaviour + Hierarchy

→ Class explosion

→ maintable

Map

→ Can → algorithms

→ bike →

→ foot →

---

class Map {

public navigate (from , to) {

. . . .

}

}

---

① New function ✗

② Enum → BIKE , ROAD ✗

Map - lean interface

↳ navigate By Car

- OCP X
- SRP X
  - ↳ navigateByRoad
- Duplication
  - ↳ navigateByDragon
- Maintainability
  - Merge conflicts

Map

↳ navigate

↳ From

↳ To

↳ Type

```
navigate(from, to, type) {
    switch(type) {
        case BIKE:
            return ____
        case CAR:
            return ____
    }
}
```

- OCP x
- SRP x
- maintoible x

---

map interface <navigate>

Car Map Bike Map Dragon Map

① OCP – open for extension ✓

② SRP ✓

Problems

- Code duplication
- Quite a few classes

- inheritance is static

(BibeMap) (new CanMap())
X

new BibeMap()

Decoration — Bibe + Can + Dragon

Strategy — Only one thing
— Configurable

Strategy

→ Composition — dynamic

A {

b b1

}

bmd

Flying behaviour

```
map {
    navigate() {

    }
```

① Strategy interface — algorithm
- Flying behaviour
- Navigation Strategy {
* Traveling salesman
* TSP
* VRP
} navigate(From, to)

② Concrete Strategies
⇒ CarNavStrategy

→ Bike Nav Strategy

```
class CarStrat imp. NavStrategy {

public void navigate() {

...

} }
```

③ Add preference to Strategy

```
@AllArgsCtor
class Map {
```

```
private NavStrategy strategy;

public void navigate(from, to) {
    strategy.navigate(from, to);
}
```

```
new Map();
map.setStrategy(new BikeS());
```

```
new Map(new BikeS())
```

```
(new CarS())
```

① Default strategy

- setStrategy()

② Factory → Strategy

① Strategy — NavStrategy
navigate

② ConcreteStrategy — CarNavStrategy
— BikeNav

- Setu

(3) Map 2

private MoveStrategy;

setStrategy();

}

6:12 – 6:15

10:45

Strategy Assignment

→ Factory + Strategy

→

input → Bike

Bike → Bike Strategy

→

→ navigate (from , to , type)

↓ Factory - Strategy

factory . createStrategy (type)

factory of factories

simple factory

→ getStrategyFactory (" Bike ")

→ . createStrategy ()

factory method

→ Command

Build up for design

→ code

PRD

→ Requirements

MAPS

→ Clarity - questions

Maintainability

Problems

60% - 40%

↳ Documentation

↳ Knowledge sharing

↳ Feedback

Document

→ Words | plain text

→ Diagrams

)

— UML diagram

↓

Unified modeling language

UML ⟶ Behavioural

functionality

interaction

↓

Structural

— class, packages

— relationships

Structural

- Class Diagram
  - classes
    - attributes
    - methods
    - relationships

- Object
- package

- Component diagram

---

① Use case

↳ functionalities

ATM

→ Insert Card

→ withdraw cash

→ check balance

(2) Sequence

— Sequence of events

(1) Insert card

(2) Enter Pin

(3) withdraw

(3) Activity → data flow

(4) State

↳

ATM

In use

Use case diagram

1. System boundary
2. Use cases — functionalities
3. Actors
   - type of users
4. Relationships

Use cases (ovals)

Rescaler → students

→ mentors

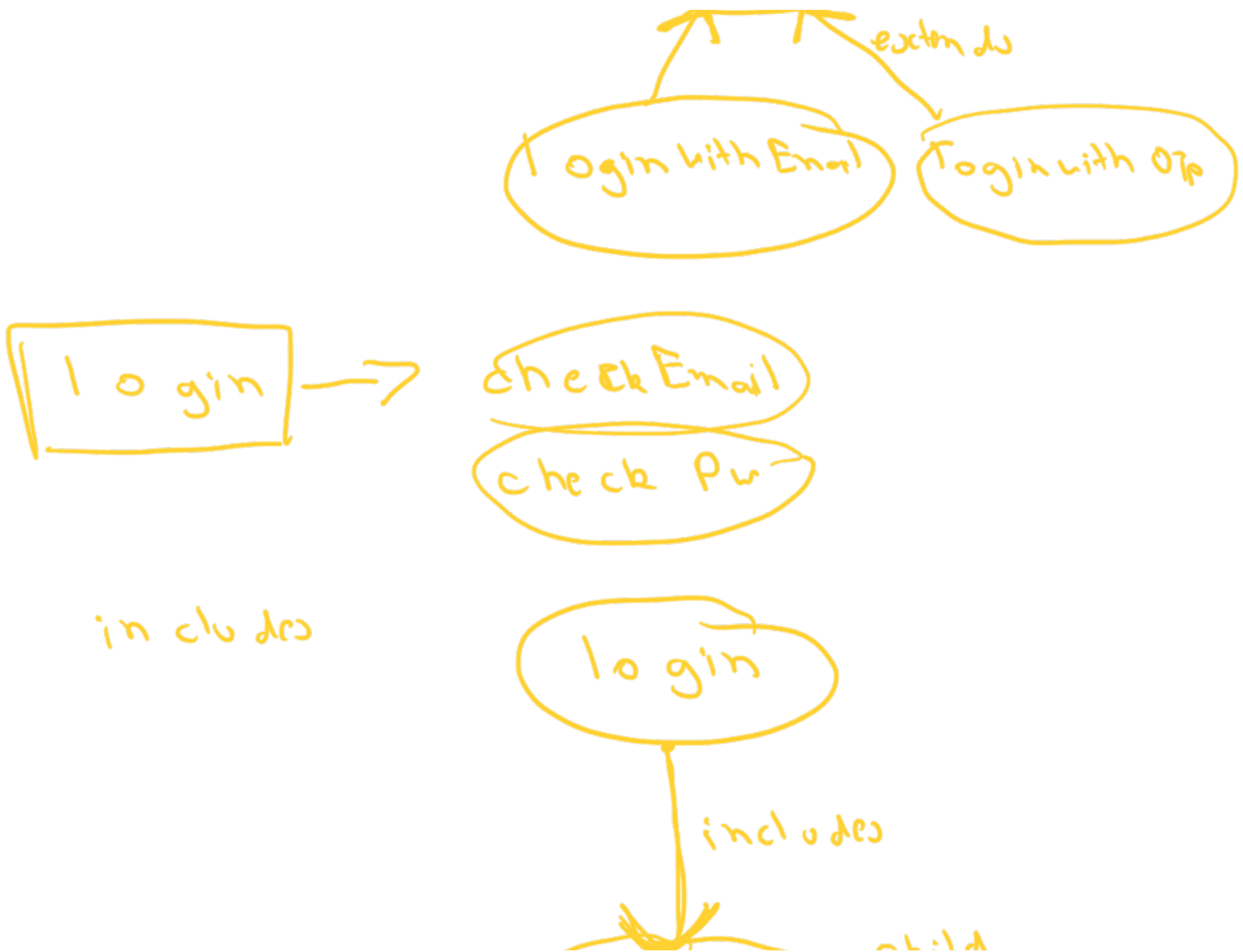
→ ins.

→ TAs

2 extensions

→ includes —

→ extends — login Parent


extends
login with Email
login with Otp
login
check Email
login
includes

check In

(1) Class boundary

→ Scala Academy | Netflix

→ 5 use cases (At least)

→ 2 actors

→ 1 use case with a dependency
(includes)

→ 1 usecase that has multiple
+ types (extends)

⇒ Use cases
→ Actors
→ Includes
→ Extends

PlantUml

Factory

→ Creational

→ create objects

→ sub classes to be used

Decoration

→ Structural

→ how different classes are associated.

→ adding functionality to the same object

A
- a
- b
- c

→ 1 functionality

= 1 class

b=

Database

new Enc(Datab)

new Compr(en)

new Encoding( )

{ Encryption Database

Compressed Database

Database

Decode

→ Dec

Enc

Comp

Decoration

- structural
- composition in main

→ Adds functionality

Strategy

- behavioural
- composition in main class

→ changes functionality

Database

+ with Encryption

+ with Encoding

Database

= Storing strategy

class A {

class B

→ set → field injection

→ Constructor. → injecti.

}

new A(new B())

Dependency injection A → B

↷ A

(b)

Is sus

→ good-first-issue