Design Patterns

1. Creational DP → 2 classes
2. Structural DP
3. Behavioural DP

Support @ Scaler.com

Agenda

Creational DP

1. Singleton DP
2. Builder DP
3. Factory Method DP

③ Factory DP → Abstract Factory DP

④ Prototype DP → Common Usages of Factory

Design Patterns

⇒ Established solutions to common software design problems

⇒ if you encounter a situation like ____

use ____ pattern

↳ follows design principles

3 Types of Software DP

① Creational DP : How to create an object.
where to create an object.
How many objects to create

② Structural DP : How to compose a class
what attributes to have

③ Behavioral DP :

How will 2 or more objects/ classes
interact with each other

↳ to cause a particular

behavior

Creational DP

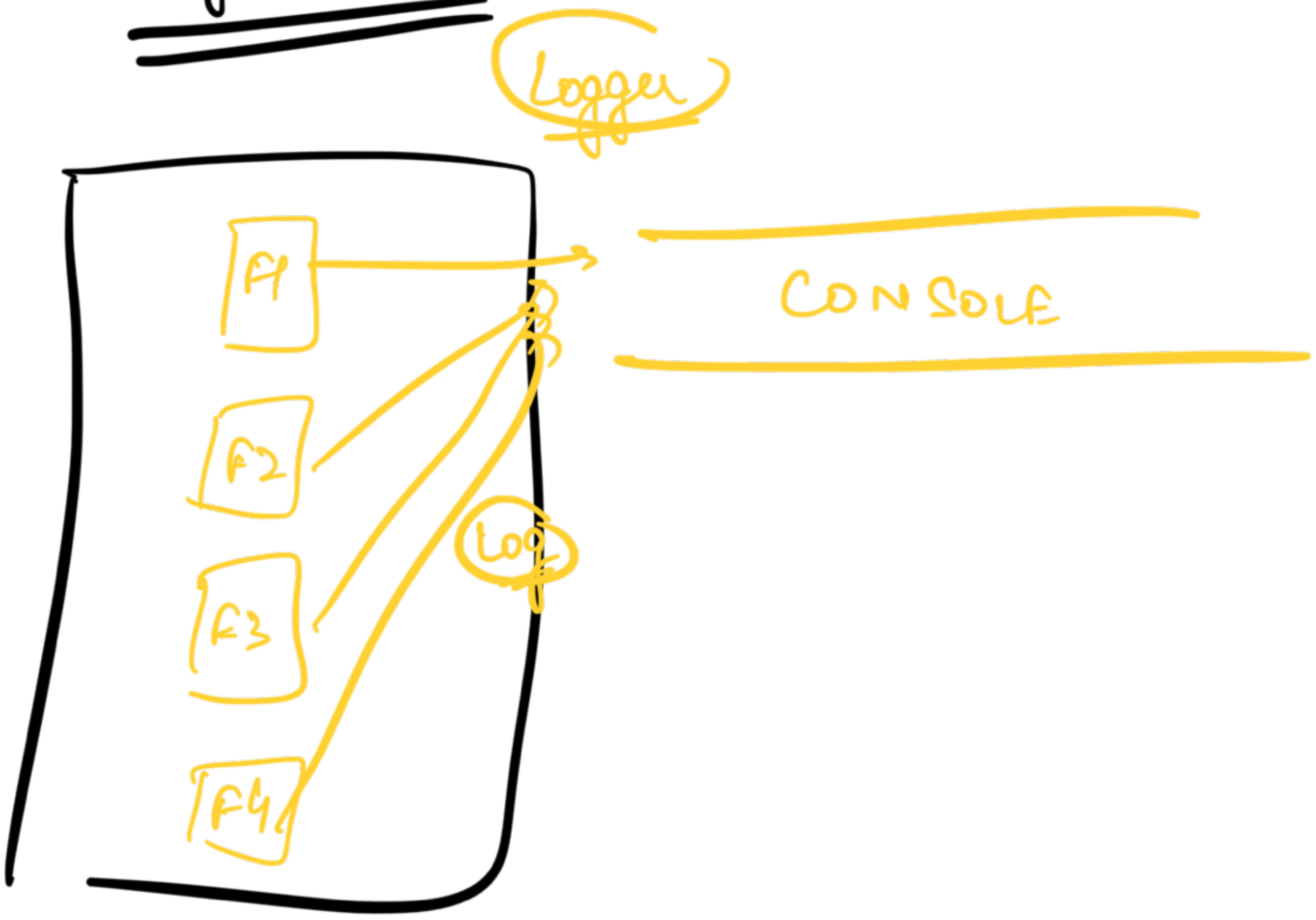
⇒ Sol^n to common software design problems around creation of objects of diff classes

① How many objects to create

② Where to create an object

③ How to create a difficult to create an object

① Singleton DP



f1 {

logger 1();

f2 {

logger 2();

f3 {

logger 3();

f4 {

logger 4();

Inputs

Singleton Class

Shared Resource

console

0

0

Singleton

→ Class whose only one instance can be created

Why to Have Singleton

① A Shared resource → db, logger.

② Class which has NO attributes (No State)

(2) Users

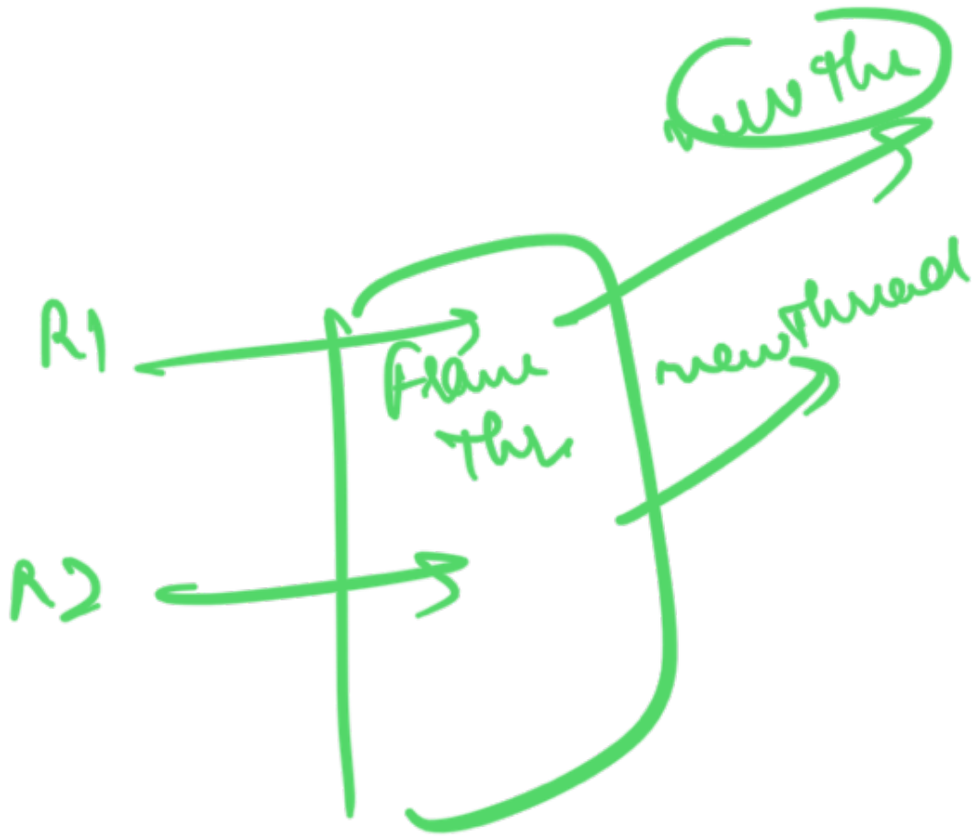
User Service {

registerUser (name, ph, password);

login (user_id, password);

}


Database
Register
Register
Parent




```
registerUser(username, password) {
    User u = new User(name,
                      password)
```

db.save(u);

}

Logger {

log(message) {
~~int a=1;~~
→ console.print(message)

→ int b=2;

}

l1.log(message)

}

l1. log (message)

How to make a class Singleton

→ Only one object of that class should be created

A constructor should always

① Return a new object

OR

② Throw an exception

class MyClass {
private static MyClass instance = new MyClass();

L

3

?

class MyClass {

⇒ private MyClass() {}

⇒ Not even a single obj can be create

}

```
class MyClass {
    private MyClass() {}

    static
public ↓ MyClass getInstance() {
        return | new MyClass();
    }
}
```

b) Creates a new instance if not already exists and returns it

② Return an already existing instance

```
class MyClass {

    private MyClass() {}
    private static MyClass instance = null;

    public static getInstance() {
        if (instance == null) {
            instance = new MyClass();
        }
        return instance;
    }
}

Client {

    MyClass myClass = MyClass.getInstance();
    MyClass myClass2 = MyClass.getInstance();
```

My Class

?

class MyClass {

private static MyClass

private MyClass() {}

public static MyClass getInstance() {

if (instance == null)

instance = new MyClass();

return instance;

}

}

This will slow down perf of app^n

only one thread will be allow to enter

instance = null;

true

}

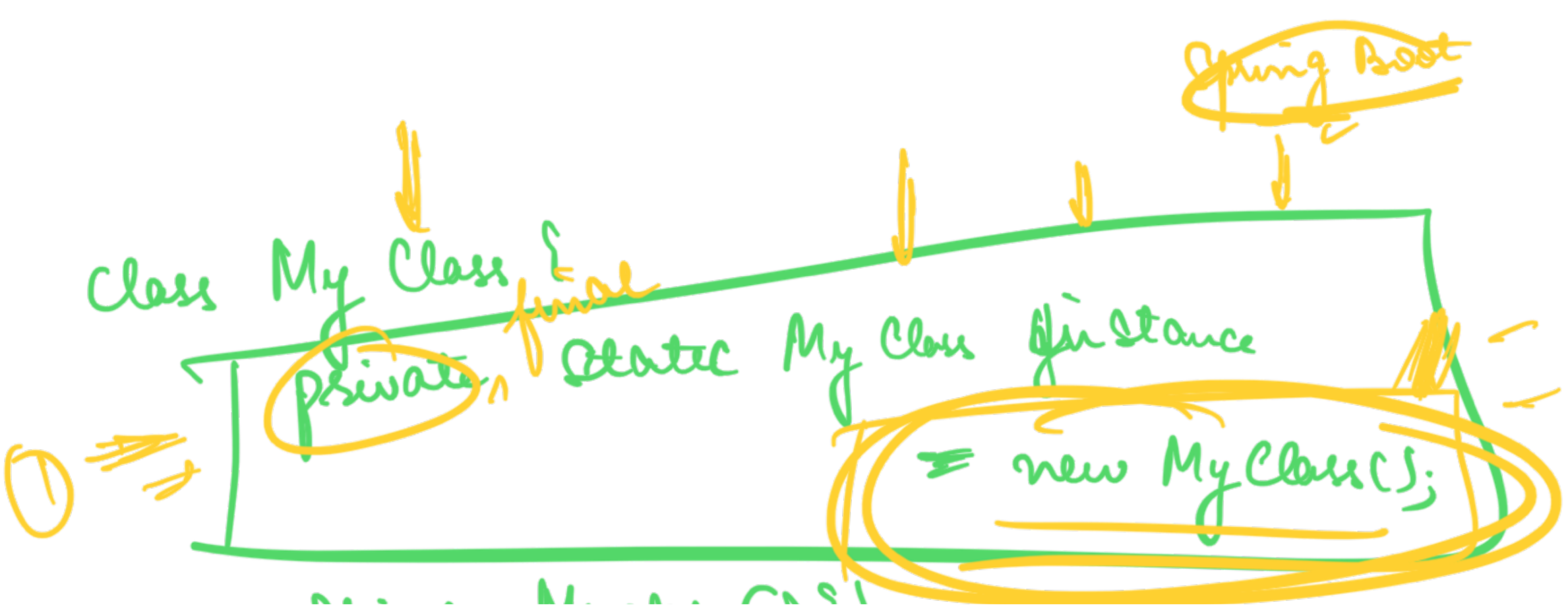
Client {

MyClass obj1 = MyClass.getInstance();

MyClass obj2 = MyClass.getInstance();

MyClass obj3 = obj2;

}



private MyClass() {}

public static getInstance() {

    return instance;

}

2 Phase lock

getInstance() {

if (instance == null) {

Synchronized (instance) {

instance = new MyClass()

}

}

return instance;

}

1

Sync getInstance {

if (instance==null)

instance = new MyClass

get Instance {

Sync (instance) {

if (instance==

return instance;

}

instance = new

My Cl

return instance;

}

05

99%

getInstance() {

if (instance == null) {

Synchronized

Sync (instance) {

if (instance == null)

instance = new MyClass();

Only one thread should go

return instance;

}

Police

Thief

Yes

(is instance is present)

Take them to jail

↳ No (instance not present)

↳ Take them to jail (if instance not present)

↳ Beat them

↳ Take them out

HLD

3P

11:30

Builder DP

→ Class that has Multiple attributes of Multiple Types

you can't modify

→ Class is also immutable

attributes of the class once we create the object

→ Before constructing an instance of a class we need to validate it

Student(, , new

Name

phNo

email

passw

)

Mutate ⇒ change

Student

Name

How will you create

email

phNo

password

age

college

an object of

Readable

StudentC

String name

String email

String phNo;

Double

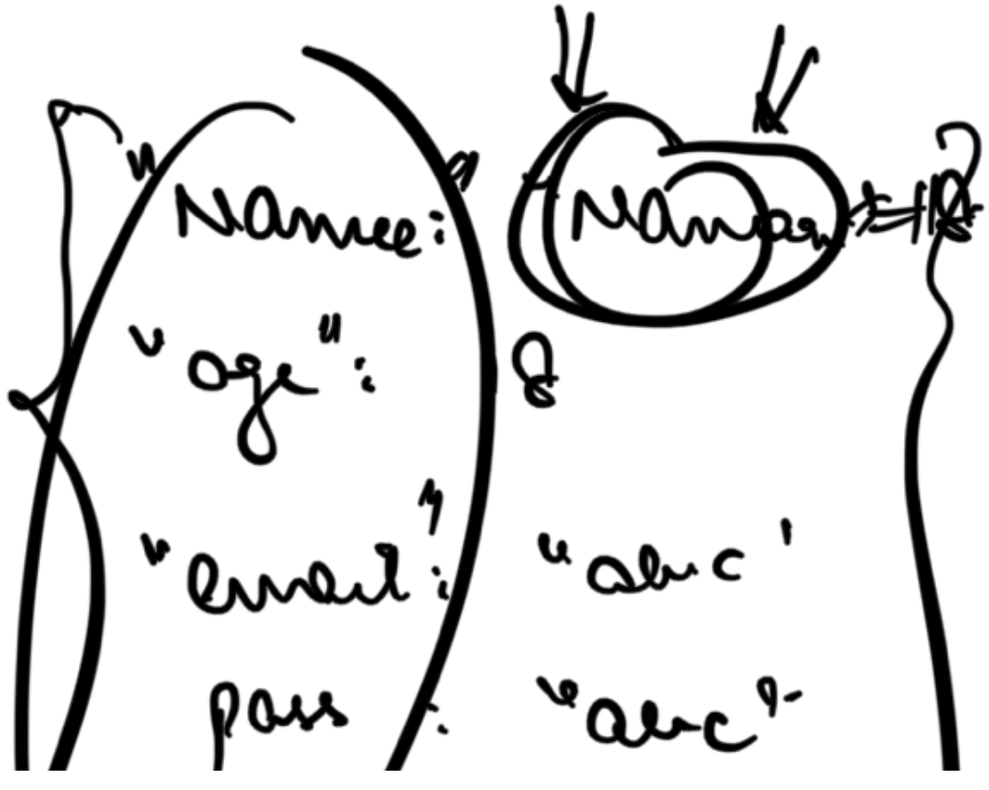
Multiple Parameter
of diff type

Student

Map < String, Object

Map<String, object>

put value of multiple types with diff Names



Parameter Enum {

Name = "Name"

age = "age"

}

{

Class → A collection of
Multiple value of
diff type

}

↓

Student (Map<String, Object>)

Class ⇒ Builder

immutable

Student (Builder builder)

Builder {

String name;

int age;

String email;

String phone; }

Set Name (name) { this. Name = name }

set Age (age.) { this. age = age }

}

Student (Builder) {

this.name = builder. Nam

this . age = builder. ag

. email . = builder. emet

}

```
Builder b = new Builder()
    b.setName(Naman)
    b.setAge(18)
    b.setEmail(abc)

Student st = new Student(b);
```