Singleton Design Pattern

⇒ Class for which at max only one Object can be created

Why?

① A class whose all objects have to share a common resource.

Codebase

```
//
```

Database db = new Database ( )

- url
- username
- password

Console

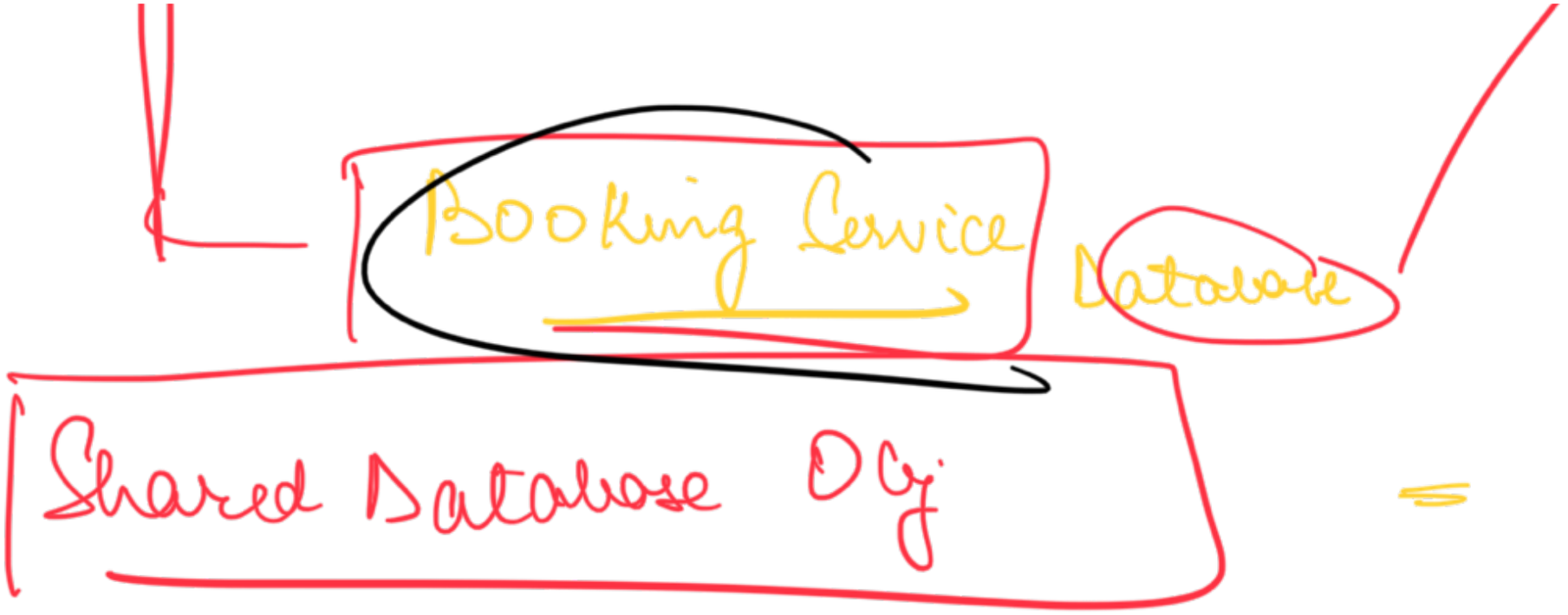
BookMy Show

User Service

Database

MovieService

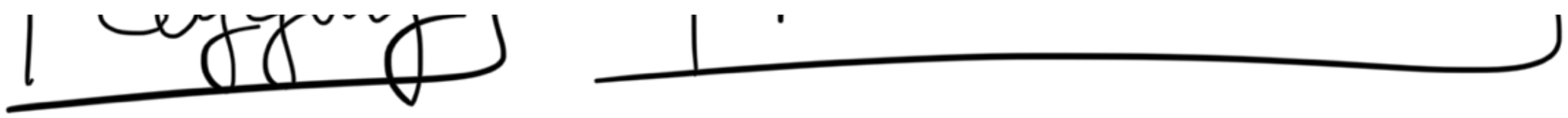
Database



=

Any class in my codebase which needs an object of Database

⇒ they should get the same obj.



⇒ across the codebase

⇒ to tell the current situation

Singleton : Creating only one object of a class and using that across the codebase wherever an object of this class is needed.

Database : Creation of an object is expensive.

↳ Connection Pool ≈ Thread Pool

1 machine ⇒ 10 queries / 10 ms

⇒ 1 query / ms

1 sec = 1000

1 Server → 1000 qps

10K+

45 Servers

Singleton DP → For a particular class ensuring only 1 obj is created for that class EVER

If any class needs to use object of Singleton class rather than creating a

New obj ~~they~~ will use same shared obj-

Why?

① Shared Resource

- Database
- HTTPCon
- ConnectionPool

a.) Creation of an object is expensive (use a lot of time/memory)

b.) It makes no sense to have multiple instance (No attributes)

(2) Class that MUST be common for everyone in my codebase

Config files

Software Project =

How to implement Singletons

⇒ Only 1 obj at max

If class has a public constructor
⇒ NO Singleton

Constructor
→ always create a new obj
→ never return an old obj

Database {

Database() {

throw new Exception();

}

}

Database {

private Database() {

}

}

attributs

Database instance

Database.getInstance()

class Database

private Database instance = null;

private Database() {}

public

public Database getInstance() {

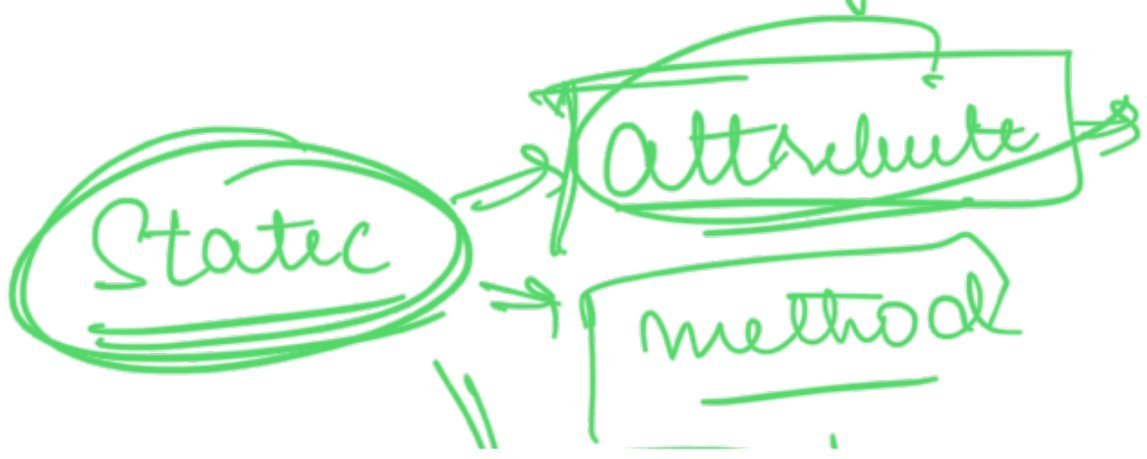
if (instance == null) ⇒ Double

Instance = new Database();

return instance.

To call a method of a class I need to

Create an object of class



Attribute at class level

Value of this attribute will

~~class~~

be same across all obj of the class

1. Class level method
2. you don't need to create an object of the class to call this method
3. Can only access Static attribute.

Static

Attribute

→ Class level attribute

→ attribute is not attached to an instance of that class

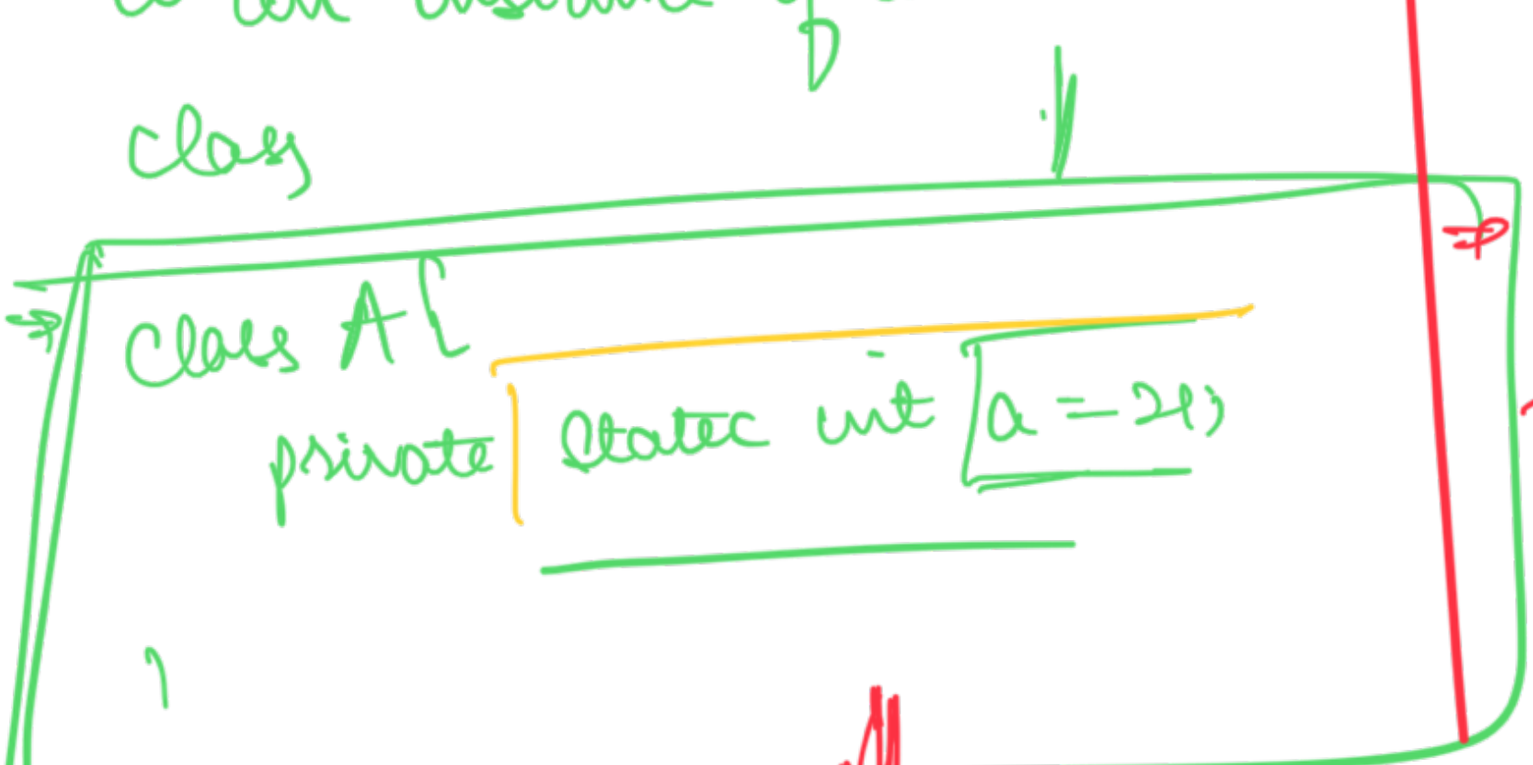
Method

→ Class level Methods

→ No need to create an object of the class to call these methods

Class

Can only access static attributes



obj1

obj2

obj3

obj3

A - a

obj1.a ⇒ 24

obj1.a = 24

Singleton

class Database {

private static Database instance = null;

private Database () {}

public static Database getInstance() {

synchronized

if (instance == null)

instance = new Database();

return instance;

}

Lazy loading → You are creating something only when it is needed

Eager loading → Create in advance

Main {

Database d = Database.getInstance();

• instance ⇐

)

}

If when 2 classes are trying to call getInstance() when no instance of DB has been created

⇒ concurrency issues

→ more than one obj will be created for that class.

Sol^n for Concurrency

① Synchronized Method

② Using Block

Cons ① Performance

private Database ()

public static Database getInstance ()

return instance;

}

}

→ The obj will be created at

start of the program

⇒ Slow down the startup of

app^n

→ Memory cost even if obj

is never needed

⇒ Parameters in the constr

① Normal

② Sync Method

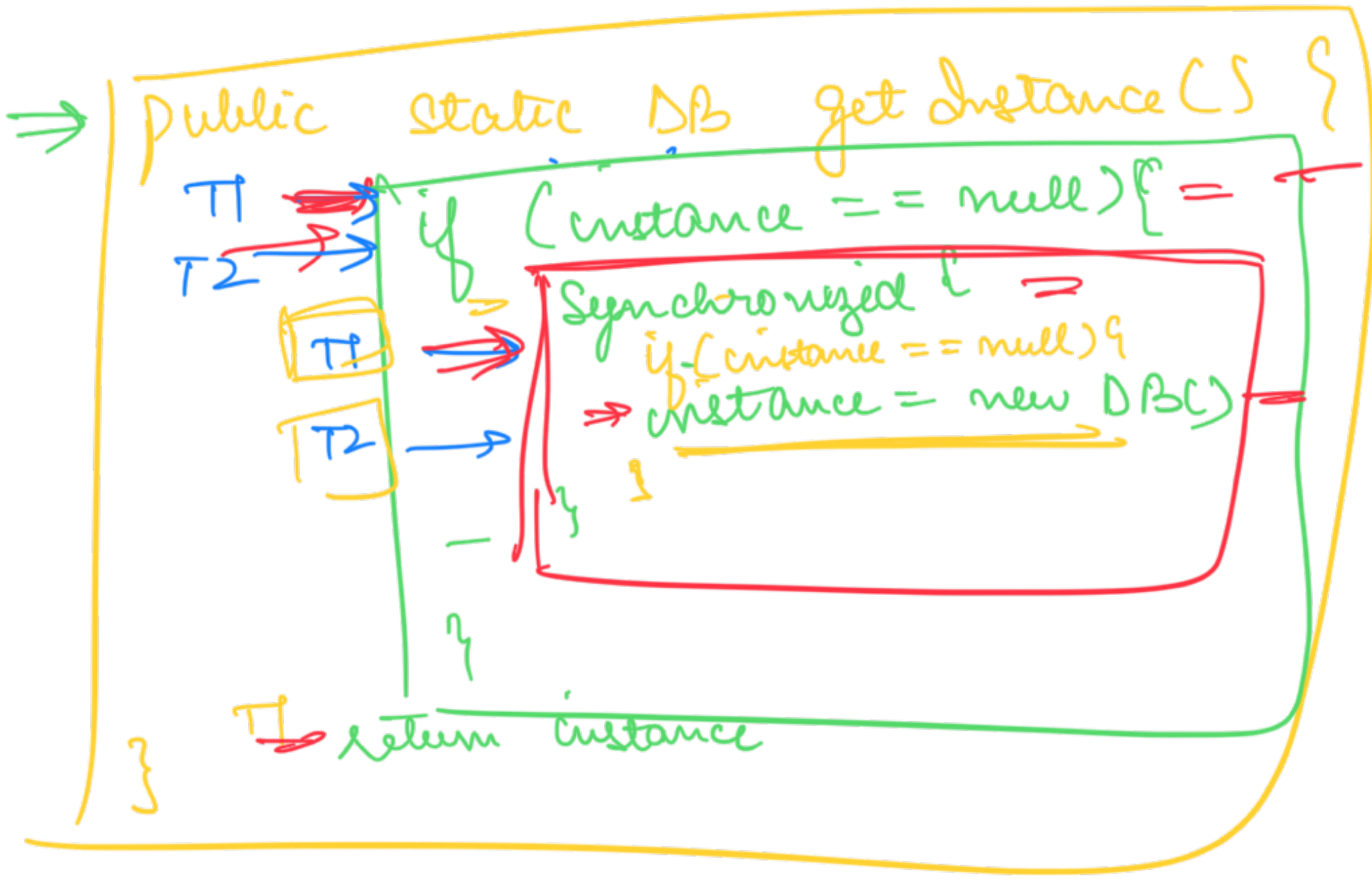
③ Static attribute calling constructor (Eager loading)

④ Double Check locking ←

Database {

Object static DB instance = null;

private DBC() {}



→ Double Check locking

① Check without lock ⇒ Performance

Real World Example

① Go to washroom

② Check is Anyone there ⇒ No one

Phone lock

Check again if No one is in

Washroom

① Go to Washroom
② Check anyone in
③ Go inside

③ Go inside

```
getInstance() {
    if (instance == null) {
        synchronized {
            // if (instance == null) {
                instance = new DB().
            // }
        }
    }
    return instance
}
```

T1 →
T2

false

3

HW

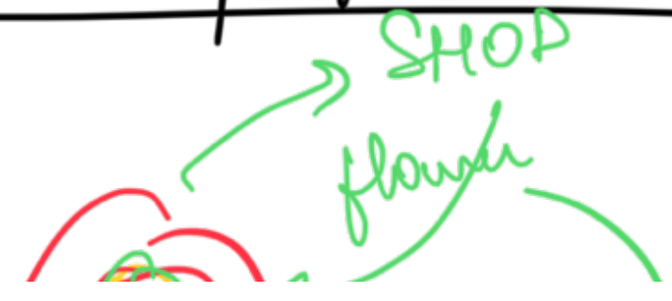
Serialization

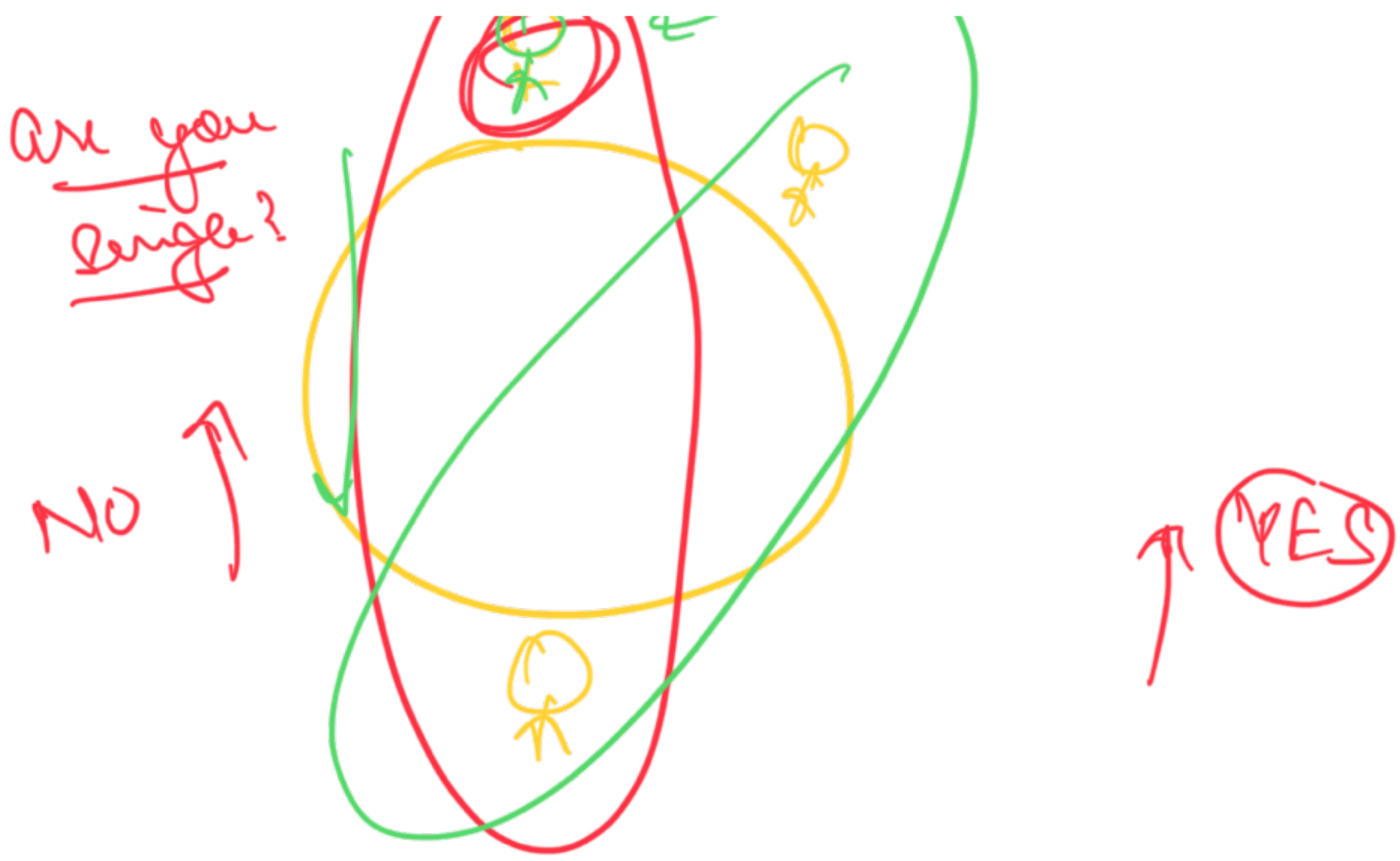
⇒ In this Sol^{n} Singleton breaks when I am Serializing

⇒ Creating Singleton via Enum.

① Go on a date with a person and your friend

② You want to propose to that person





Cons of Singleton

→ When I do unit testing, I often need to

→ Mock some attributes / methods.

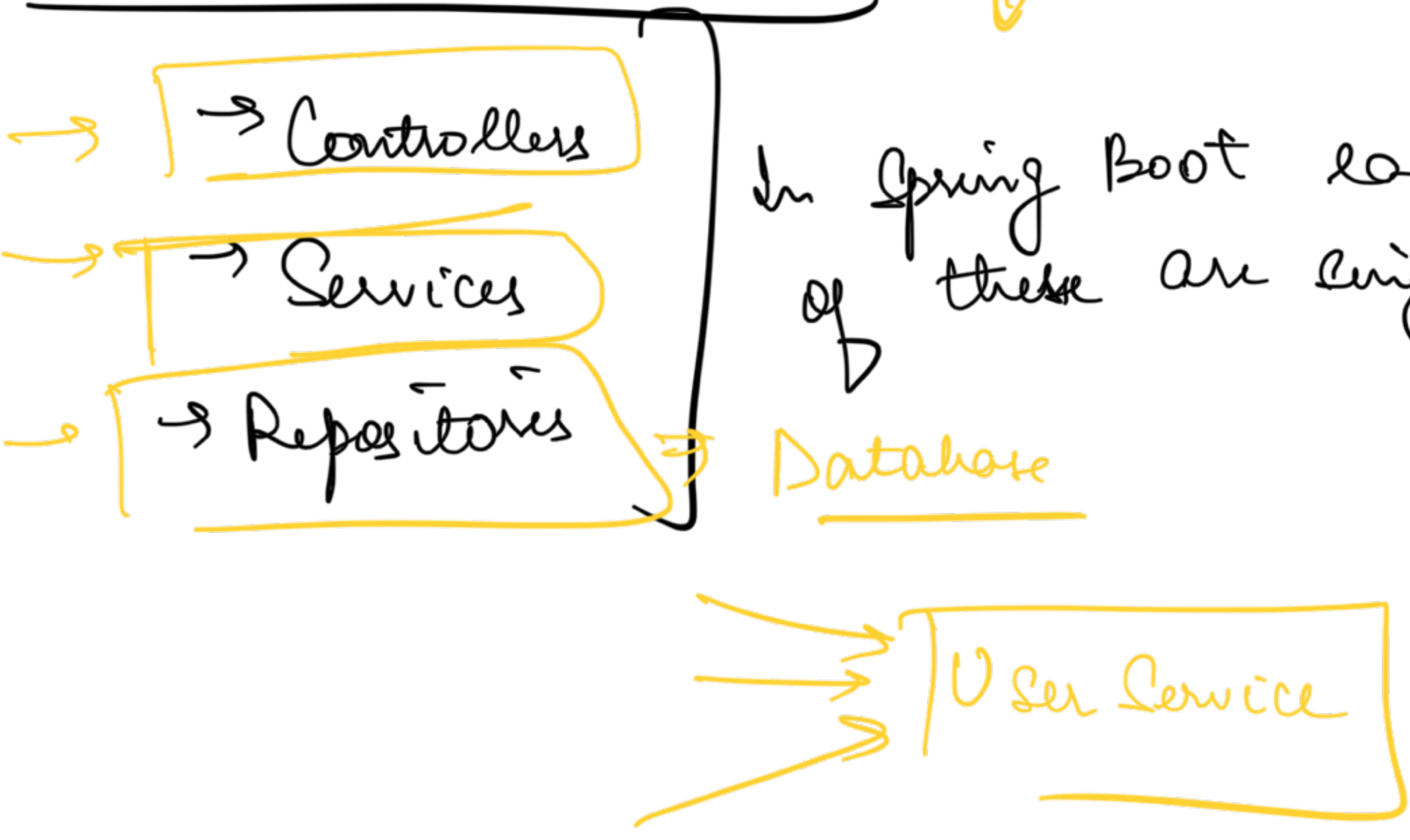
→ With Singletons it is not possible

① Many testing frameworks inherit a class to it

→ Most of the times Singleton objects are immutable.

Where are Singletons Used

① In your web frameworks



In Spring Boot each of these are singleton

Singleton : Class with at max one obj

Ways to create Singleton

① Normal

② ~~Eager Loading~~ Lazy loading

③ Eager loading

④ Lazy loading with Sync

⑤ Lazy loading with Double Check

When to use Singleton

① Shared resource

(1) Shared resource

(2) Expensive creation

(3) Stateless (No attributes)

(4) ~~Same obj~~ Only one obj to be present across my app
(Config file / Mock Database)

~~When to~~ Why not to use

(1) Makes testing difficult

→ Recommend against using Singleton

→ Why

① You can't subclass a Singleton

② You can't change the attrs once created / can't mock the attributes.

Singleton Database

↳ Test if DB gives correct response of URLs

wrong

↳ Test of DB throws XYZ when you wrong obj.

Page

volatile

→ is not a sol^{n} of concurrency

→ is a sol^{n} of visibility

Builder
Factory