Creational Design Patterns

Will Start at 9:10 PM

- Singleton Design Pattern
- Factory Design Pattern

Agenda

1. Intro to Design Patterns
2. Types of Design Patterns
3. Creational Design Patterns
4. Singleton Design Pattern
5. Factory Design Pattern

⇒ Often as SWE we encounter problems that others have also encountered in the past

Design Patterns ⇒ Well established good sol^n to commonly occuring software design problems.

GoF ⇒ Gang of Four

Why Learn
- Jump in daily job
- Jump in interviews
  - Theory
  - Design problem.

How are DP used

1. Shared vocabulary
2. save a lot of time

F/E design pattern
MVC design patterns

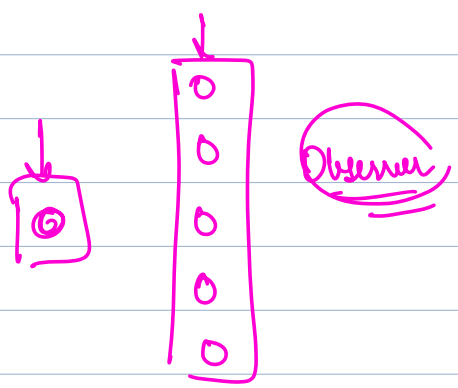
Types of Design Patterns

→ Object Oriented Design Patterns

1. Creational Design Patterns
   → Solve problems around how objects of a class should be created

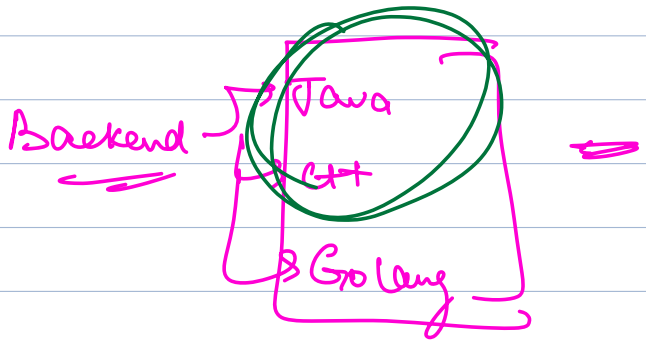
2. Structural Design Patterns
   → Flyweight
   → Adapter
   → Decorator
   → Facade
   → how should a codebase be structured

3. Behavioural design patterns
   → Solve problems around implementing a behaviour



8 design pattern

- Most imp in interviews
- Most common in day to day job.



CREATIONAL DESIGN PATTERNS

- Singleton
- Factory
- Builder
- Prototype

How?

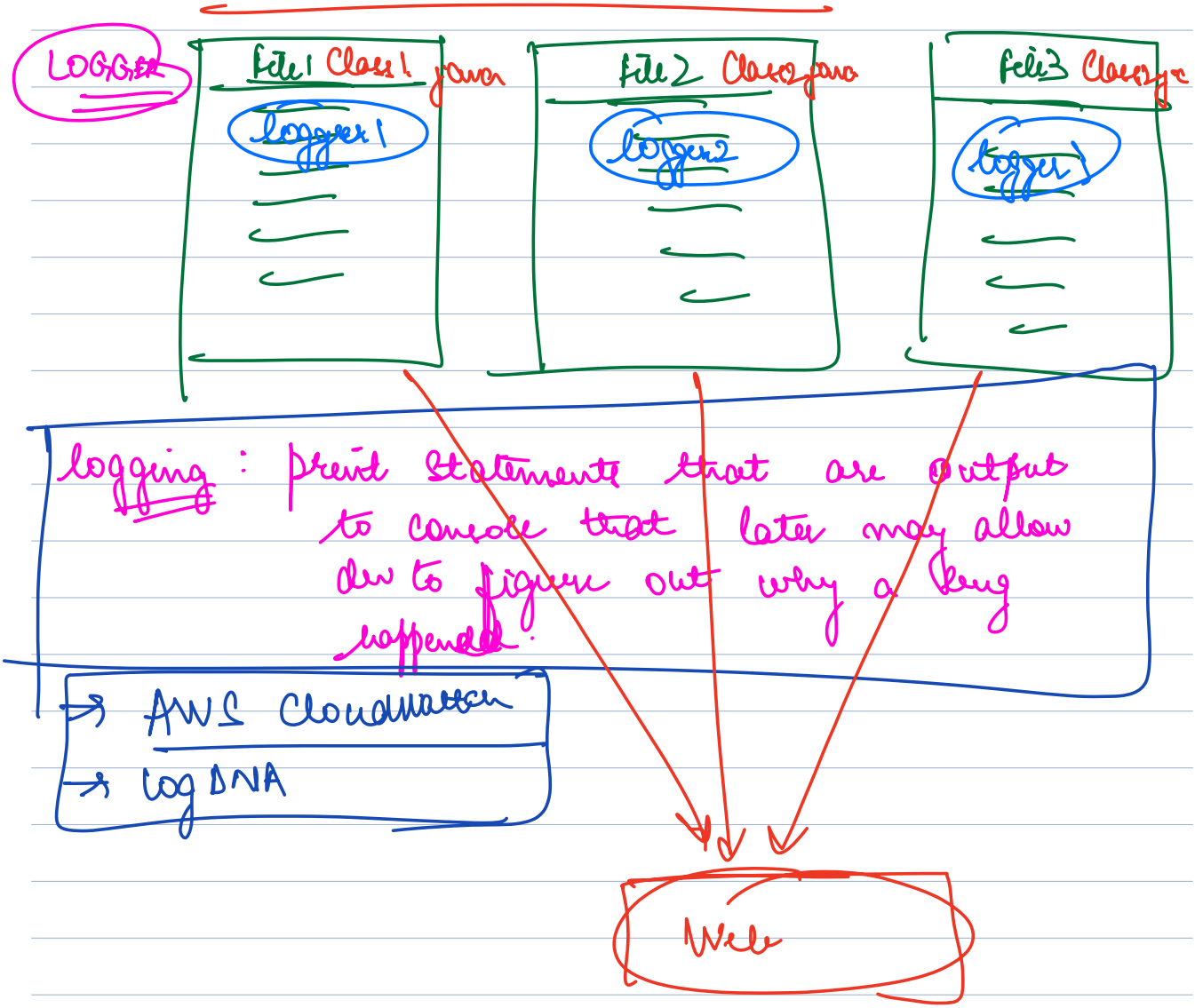
Why?

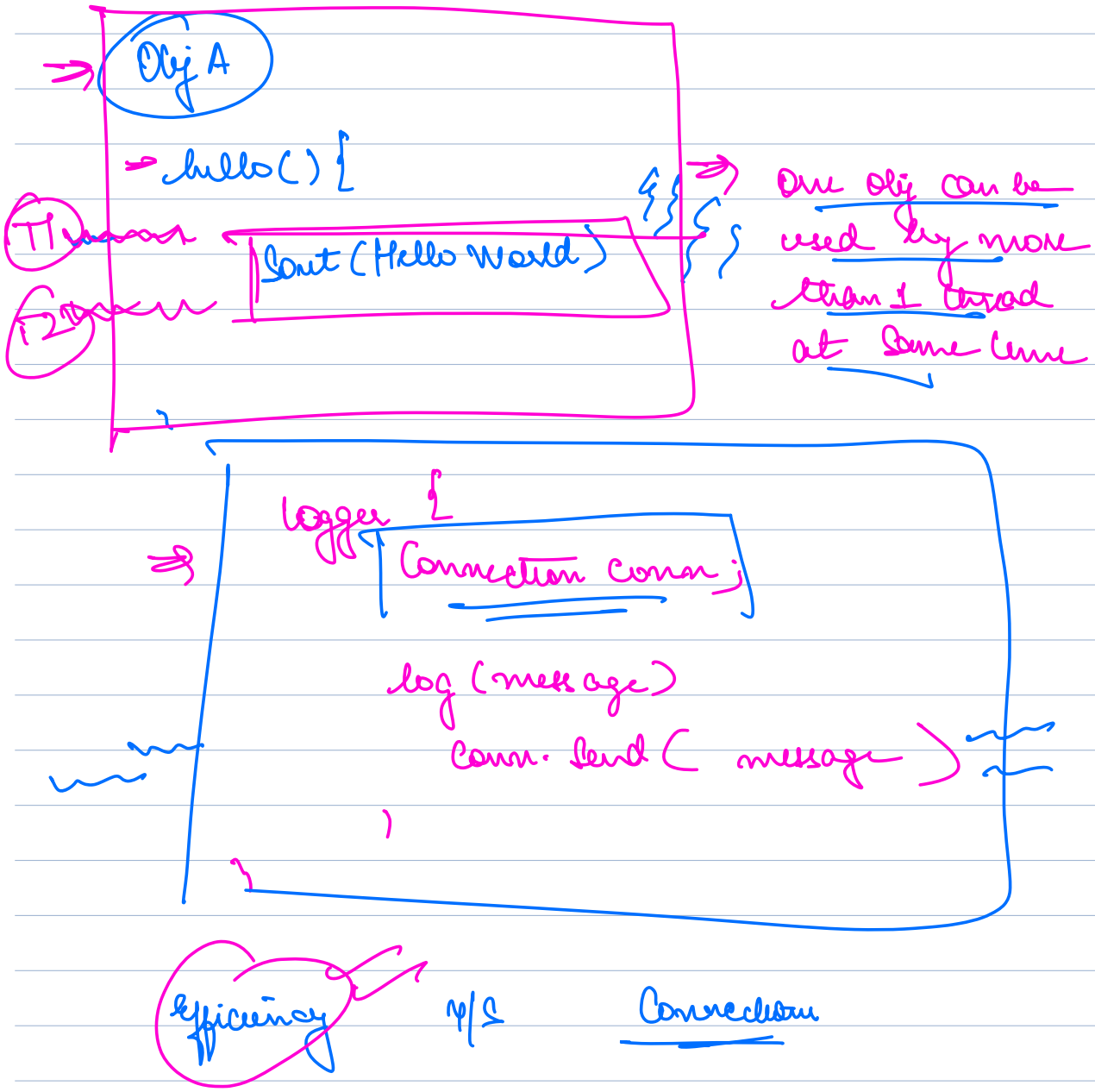
How?

Pros

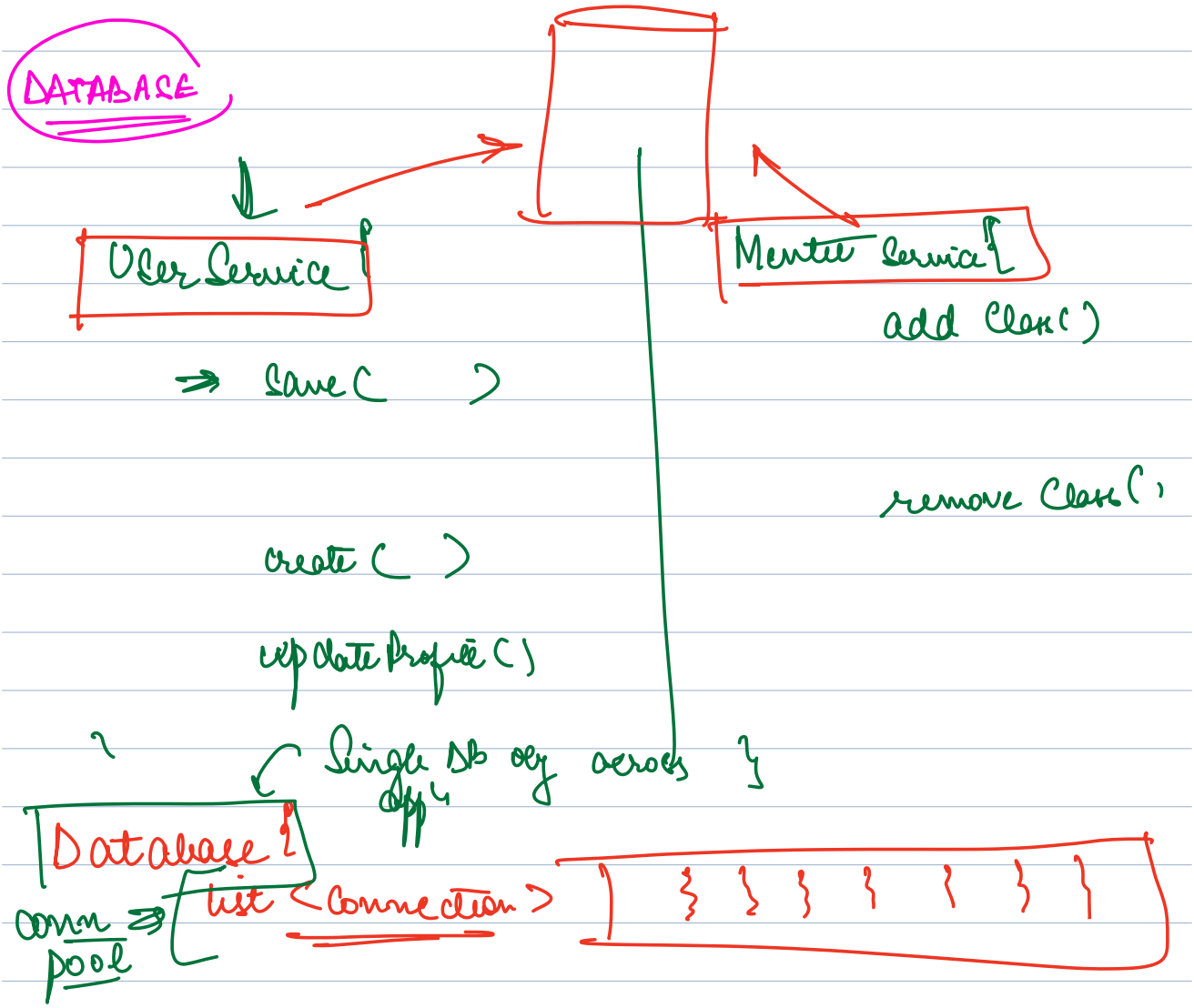
Cons

SINGLETON DESIGN PATTERN

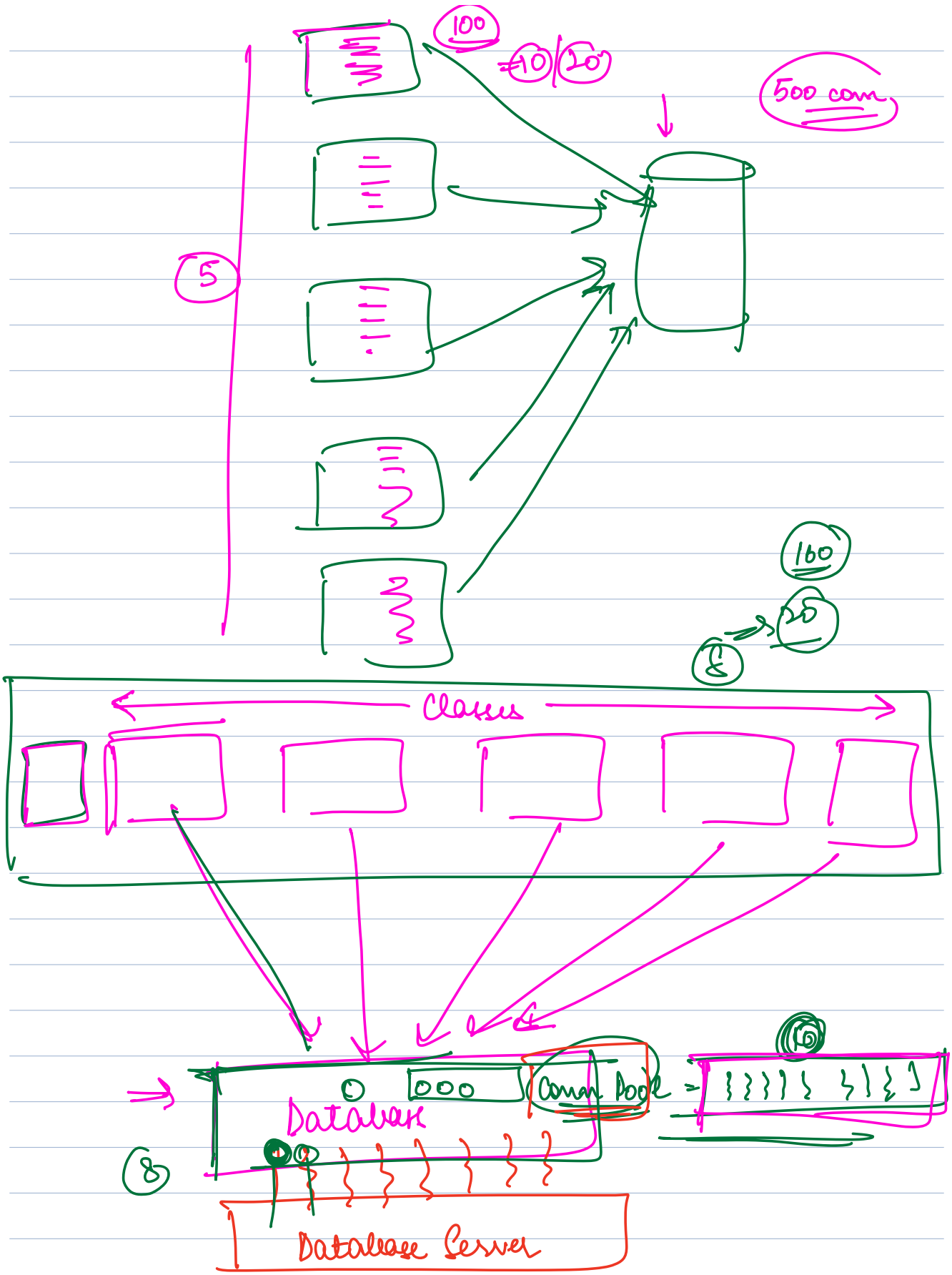


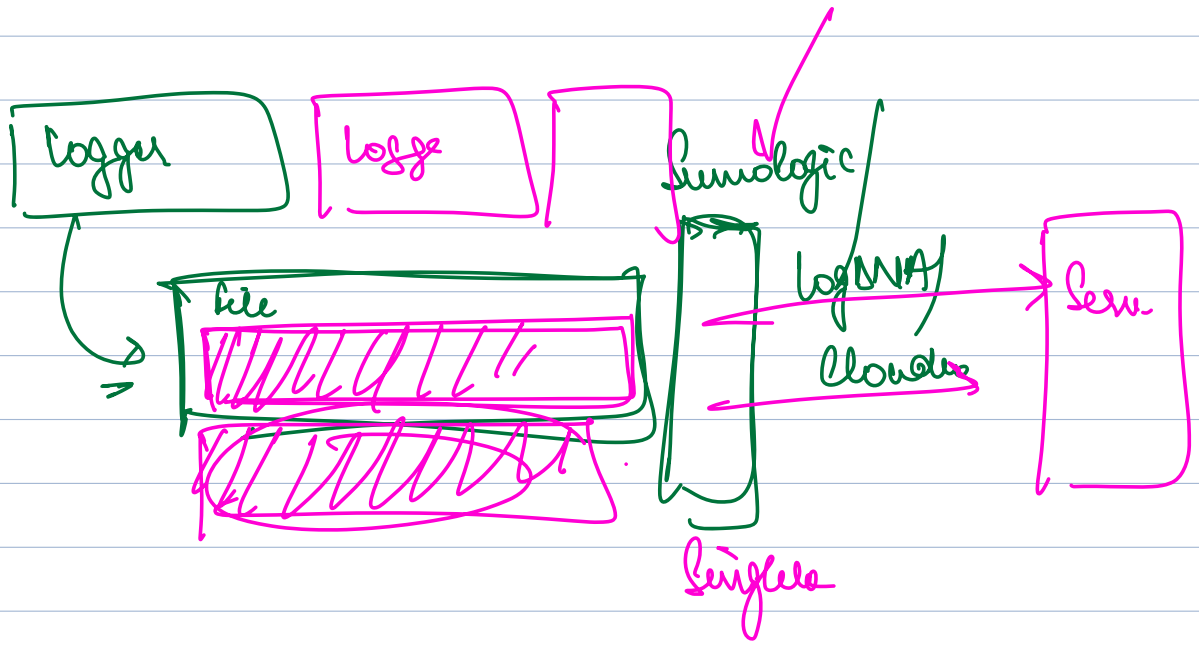


⇒ having single obj of logger across all the classes will be better.

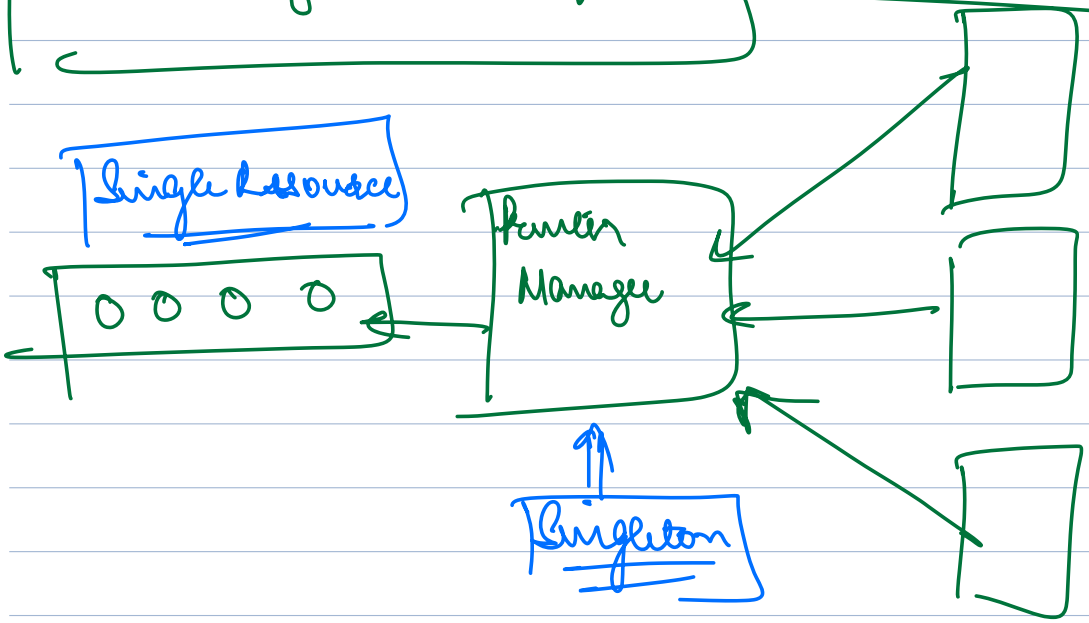


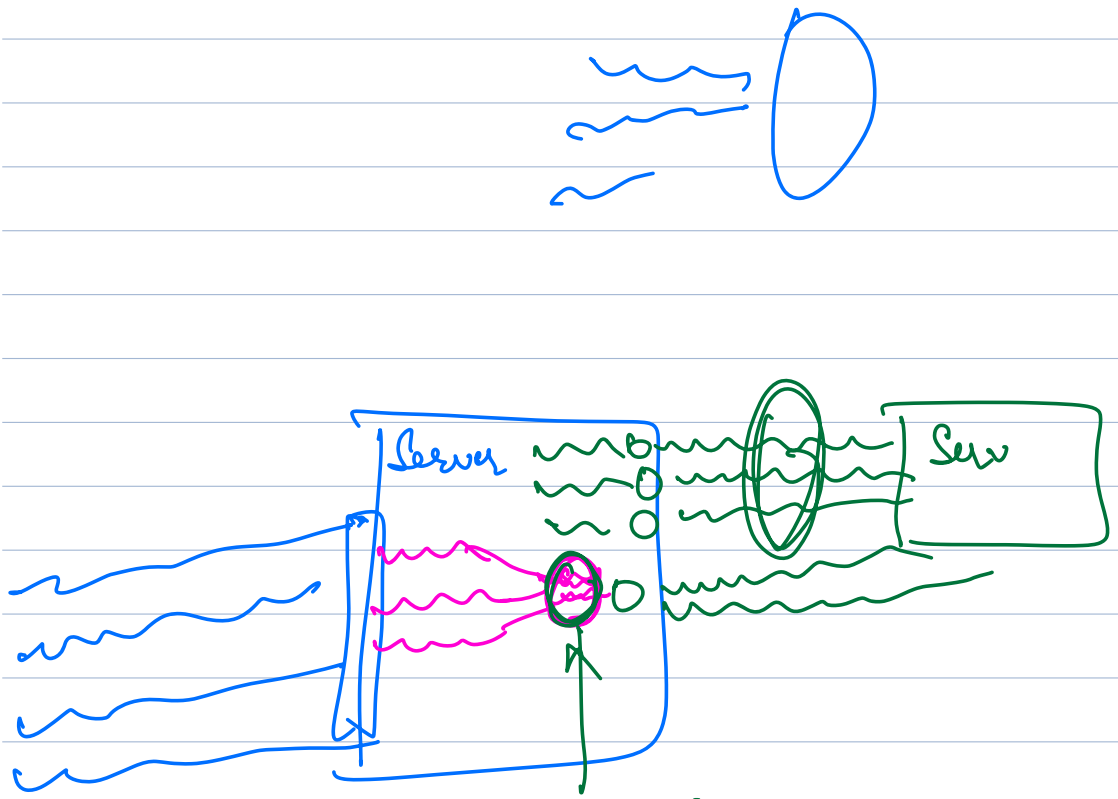
query() {
Check if any conn is available:
if no:
wait
else:
use that conn to execute query
}


100
10/20
500 conn
5
160
25
4
Classes
Database
Conn Pool
8
Database Server



① When we have a shared resources behind the scenes, makes sense to have a singleton in dpfw.



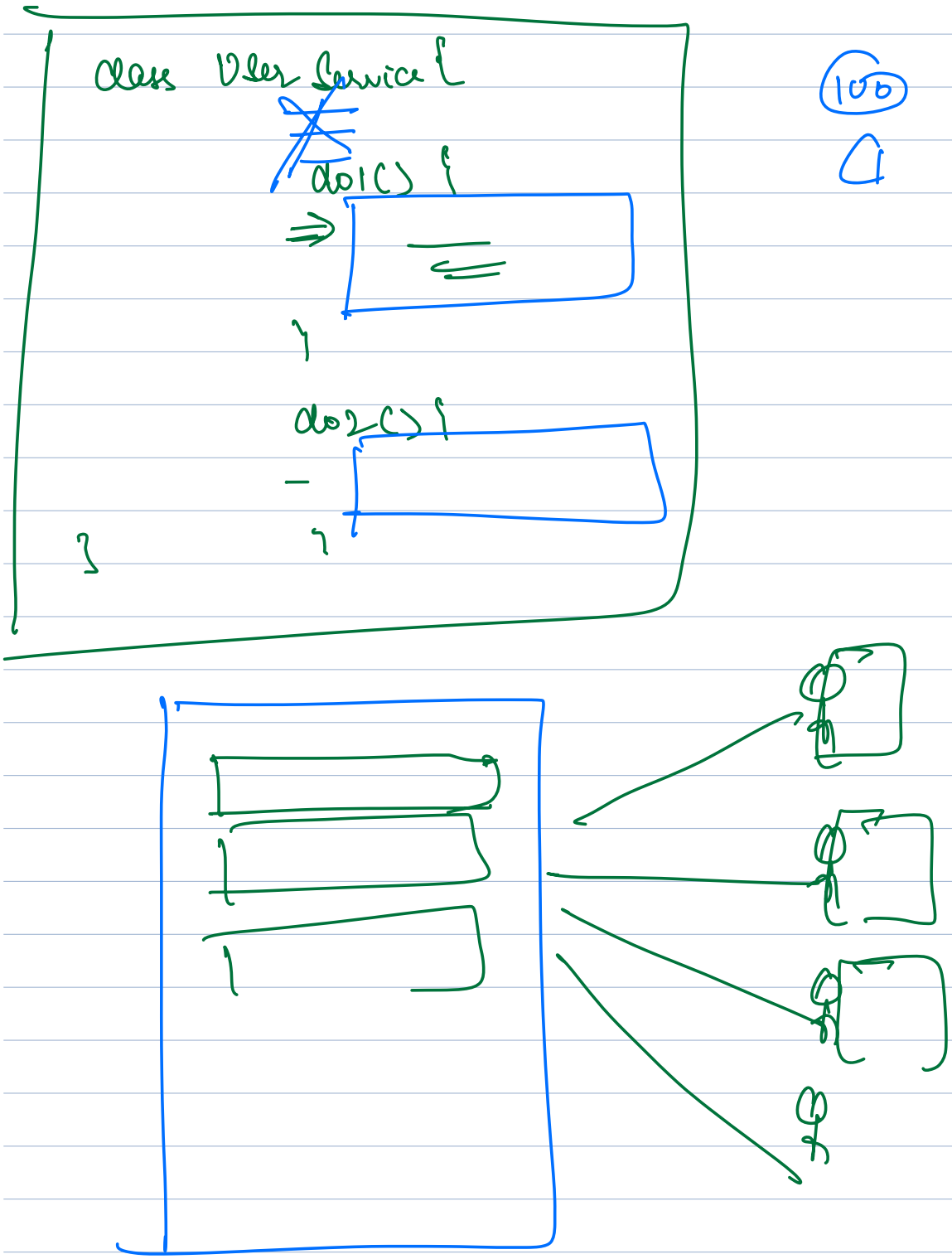


all will execute unless I take a lock

---

② When creating an obj is expensive

③ When an obj has only 1 state / no state (no attrs)


class UserService {
doIC) {
doIC) {

HOW WILL YOU MAKE A SINGLETON

⇒ Till the time constructor of a class is public, can class be singleton

⇒ NO

⇒ Constructor always create a new ob

⇒ Till cons is public, anyone can call cons and create new by

(1) Cons. should be private

```
class Database {
    List<Conn> conns;
    String url;
    String password;

    void executeQuery() {

    }
}
```

Not Singleton

```
class Database {
    List<Conn> conns;
    String url;
    String password;
    private Database(){}
    void executeQuery(){

    }
}
```

⇒ But now I can create 0 objects :-(

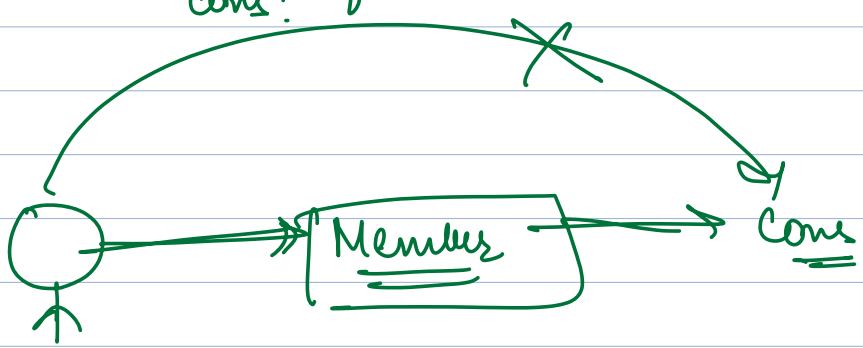
---

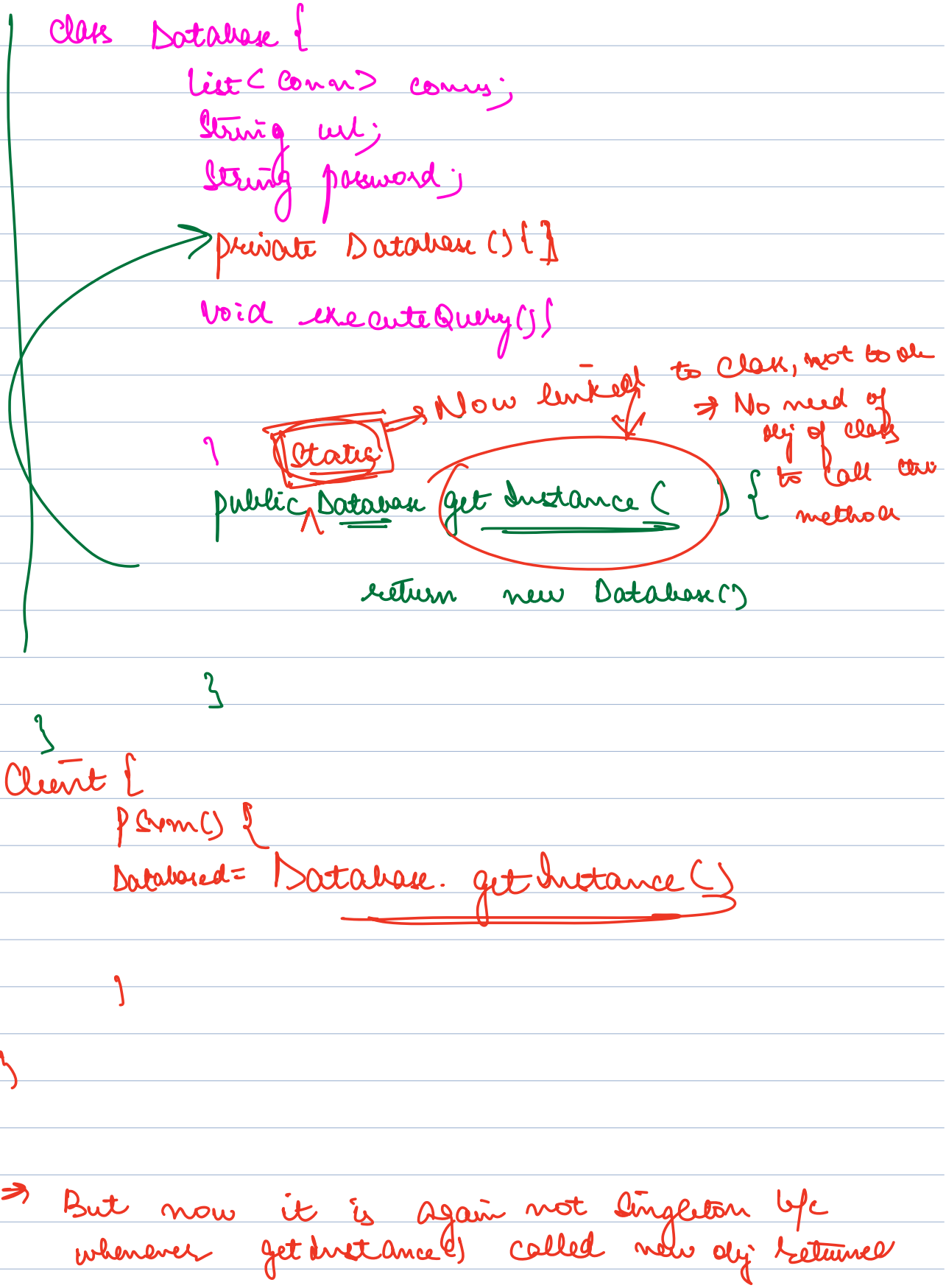
① We can't make constructor public ⇐

② We still need 1 obj of class

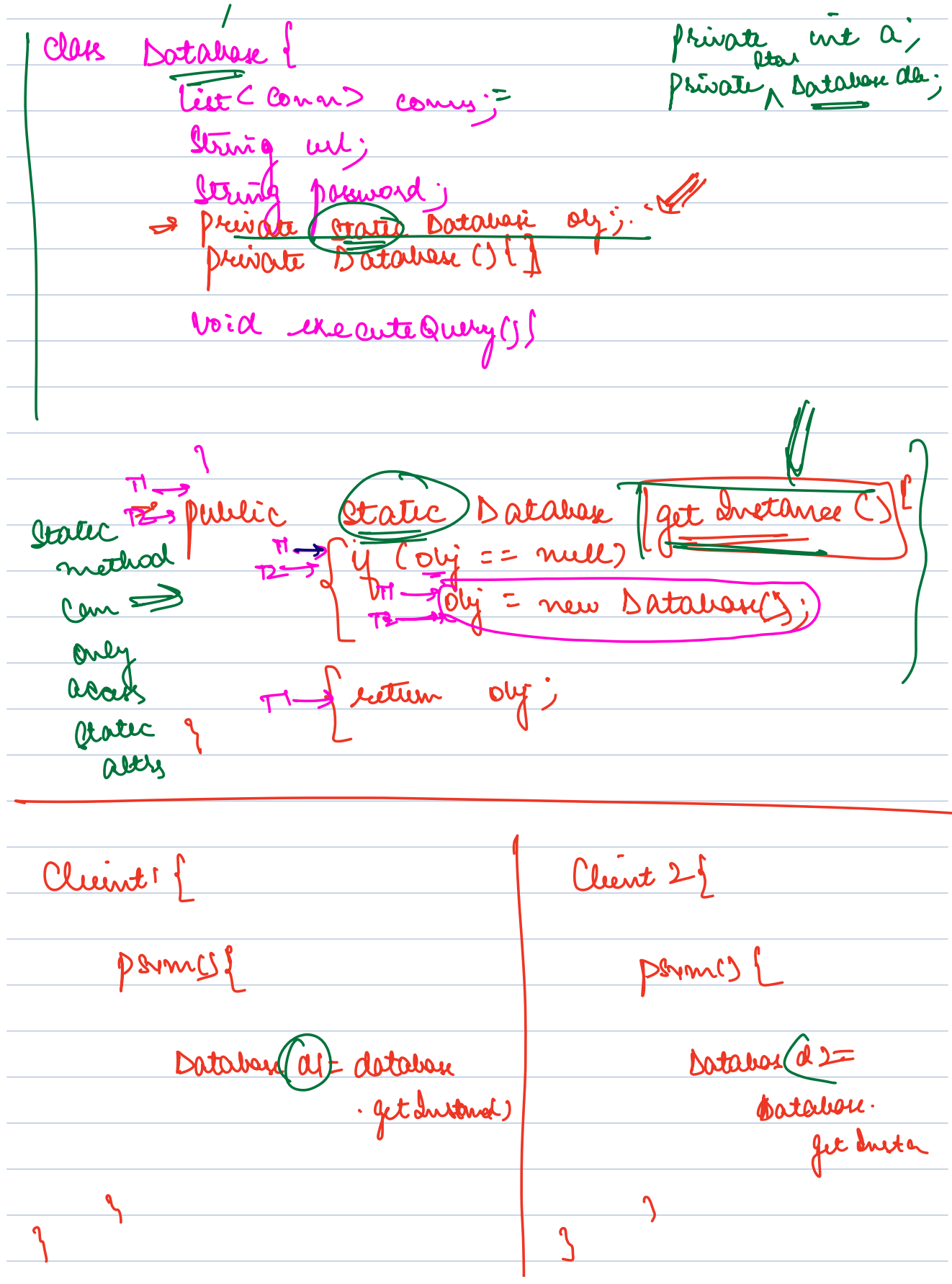
③ Obj of class can't be created w/o calling cons

⇒ ! Members of class can still call that cons!





⇒ But now it is again not Singleton b/c whenever getInstance() called new obj returned


Class Database {
List<Conn> conns;
String url;
String password;
private static Database obj;
private Database(){}
void executeQuery(){}
private int a;
private Database db;
public static Database getInstance()
if (obj == null)
obj = new Database();
return obj;
static method can only acess static attrs
Client1 {
psvm(){
Database d1 = database.getInstance()
Client 2 {
psvm(){
Database d2 = database.getInsta

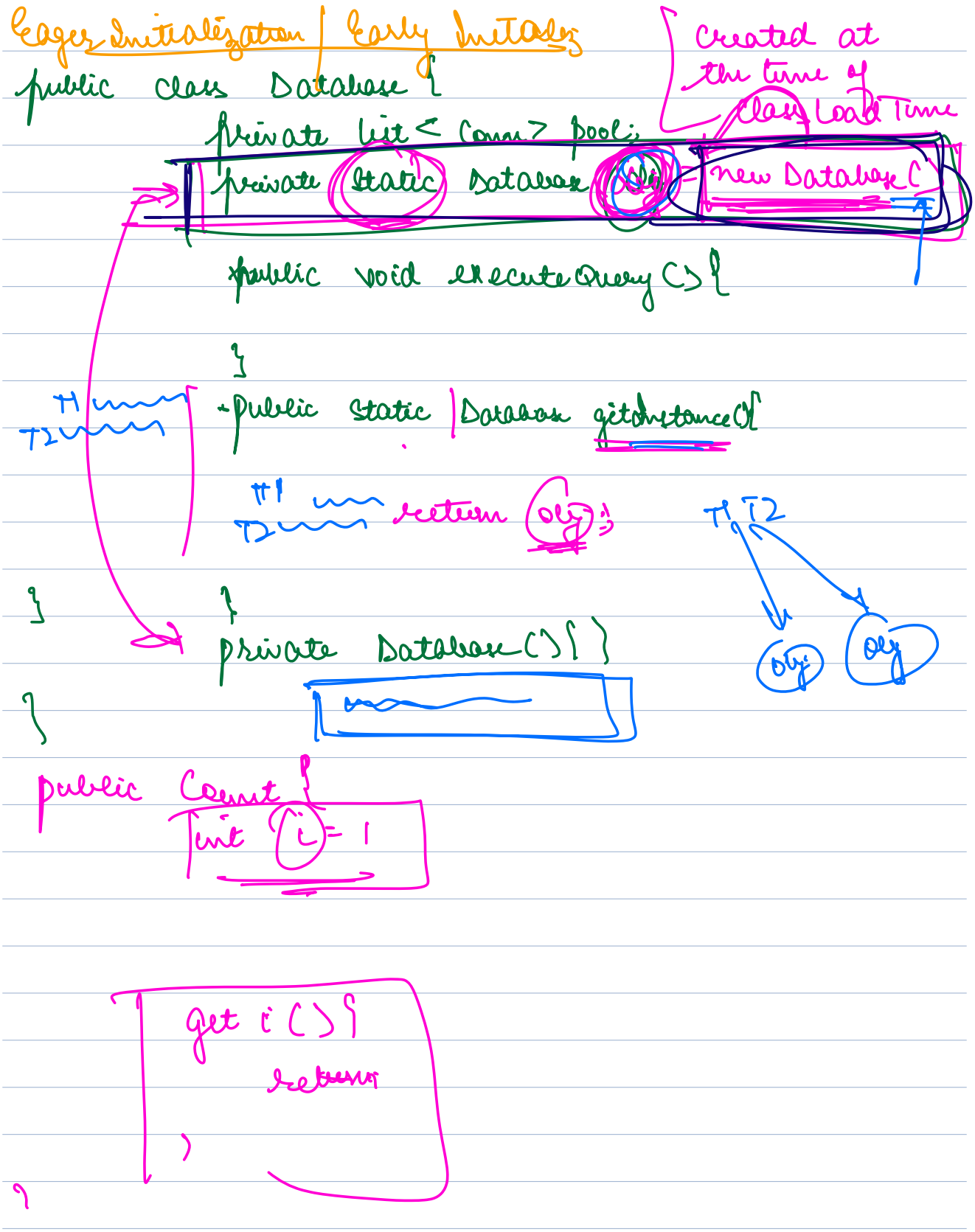
Singleton with conc handled

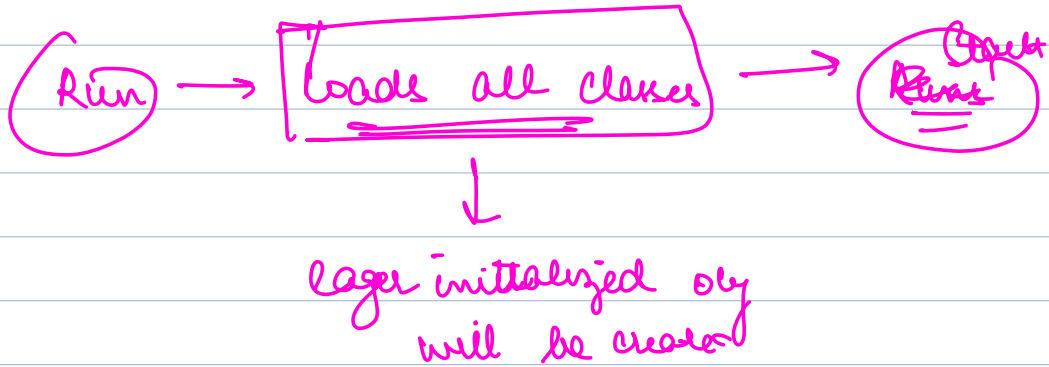
```
public class Database {
    private list<Conn> pool;
    private static Database db;

    public void executeQuery() {
    }

    public static synchronized Database getInstance() {
        if (obj == null)
            obj = new Database();
        return obj;
    }

    private Database() {}
}
```


Eager Initialization / Early Initialz
public class Database {
private List<Conn> pool;
private static Database obj = new Database();
Created at the time of Class Load Time
public void executeQuery() {
}
public static Database getInstance() {
return obj;
T1
T2
private Database() { }
}
public Count {
int i = 1
get i() {
return



Soln to conc

1. Synchronized
2. Lazy initialization

Problems

1. Synchronized

→ take a lock on every call of getInstance
→ Slow performance

→ Conc issues would have happened in first few instances of calling. After that no risk
→ But because of Sync, I have also slowed down future calls

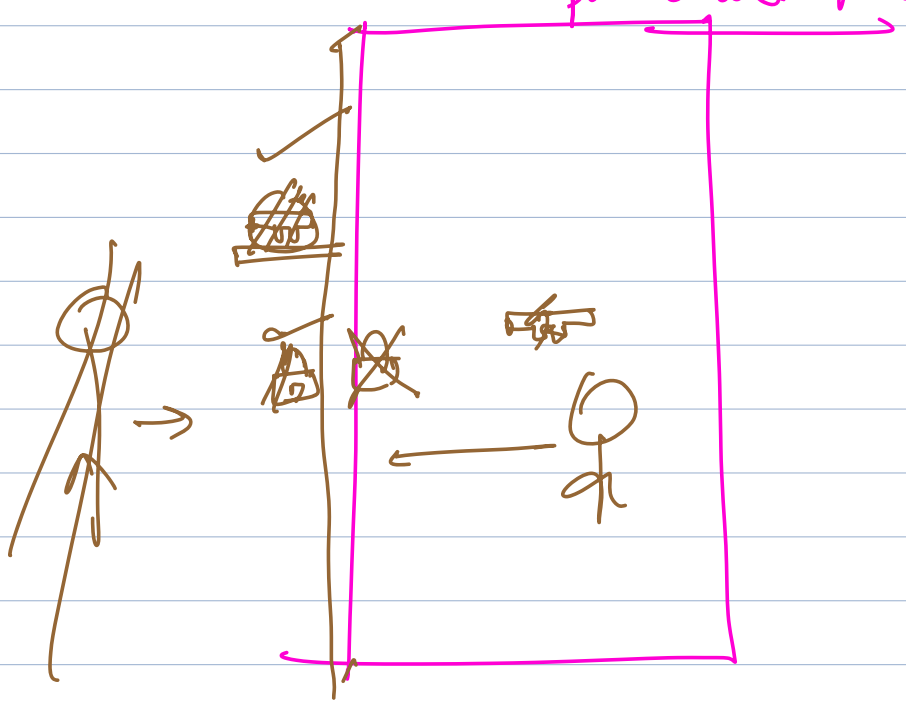
```
getInstance() {
    if (i == null)
        return i;
}
```

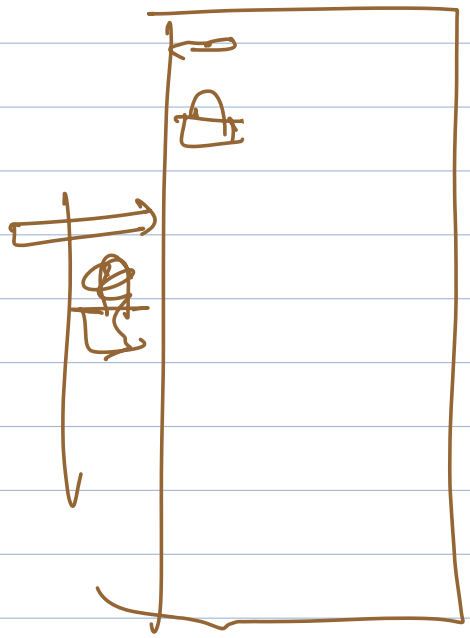
② Eager initialization

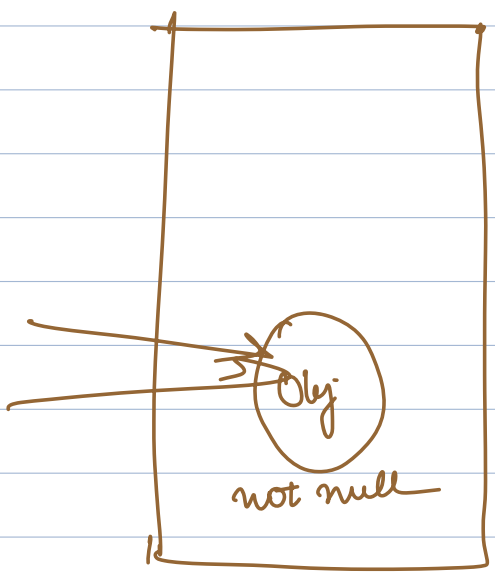
→ Slow down app start time.
→ Not always possible to use
⇒ eg cons takes params that are not there at class load time

DOUBLE CHECK LOCKING

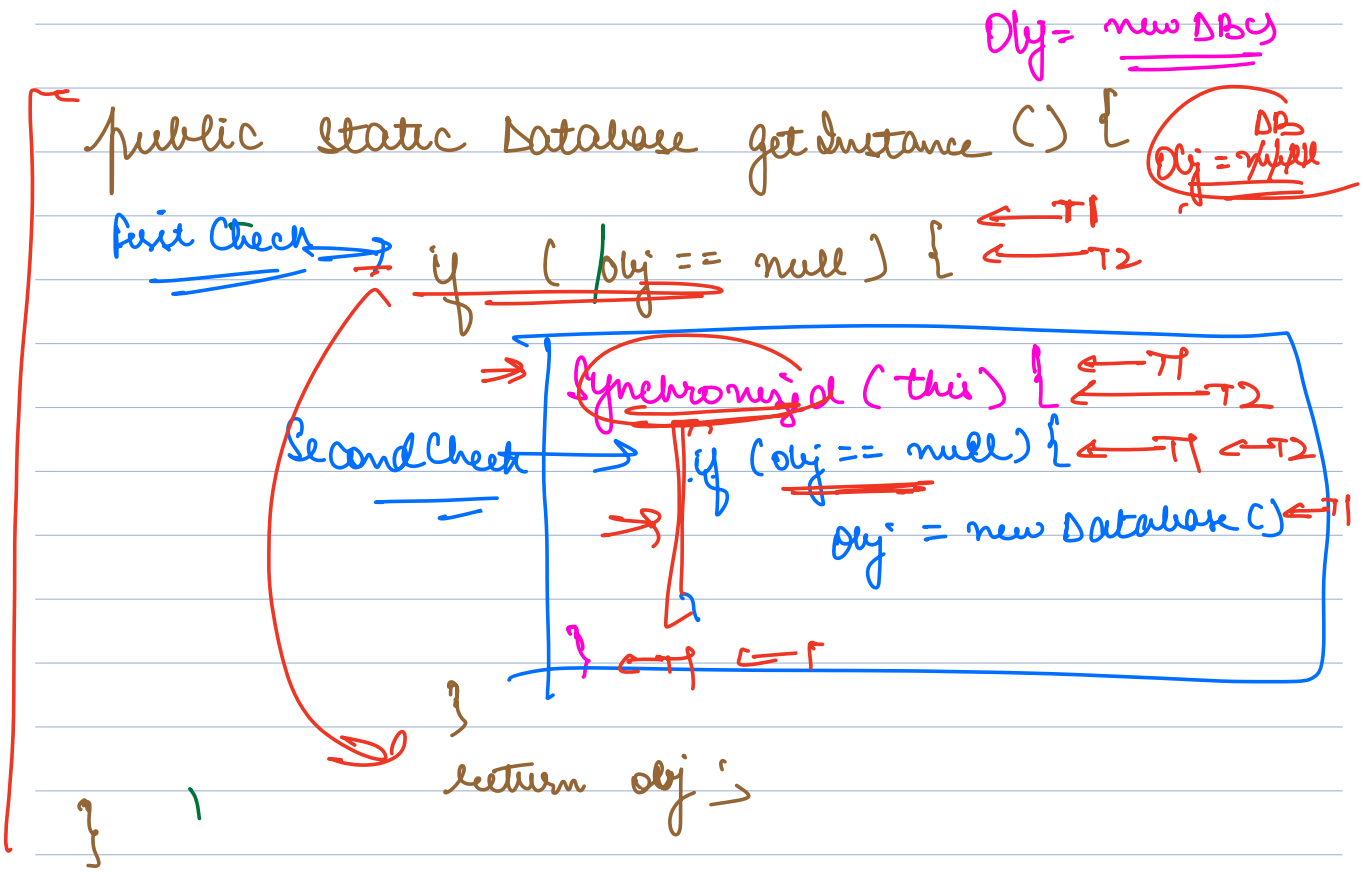
→ everywhere where we need to handle conc while ensuring perf.







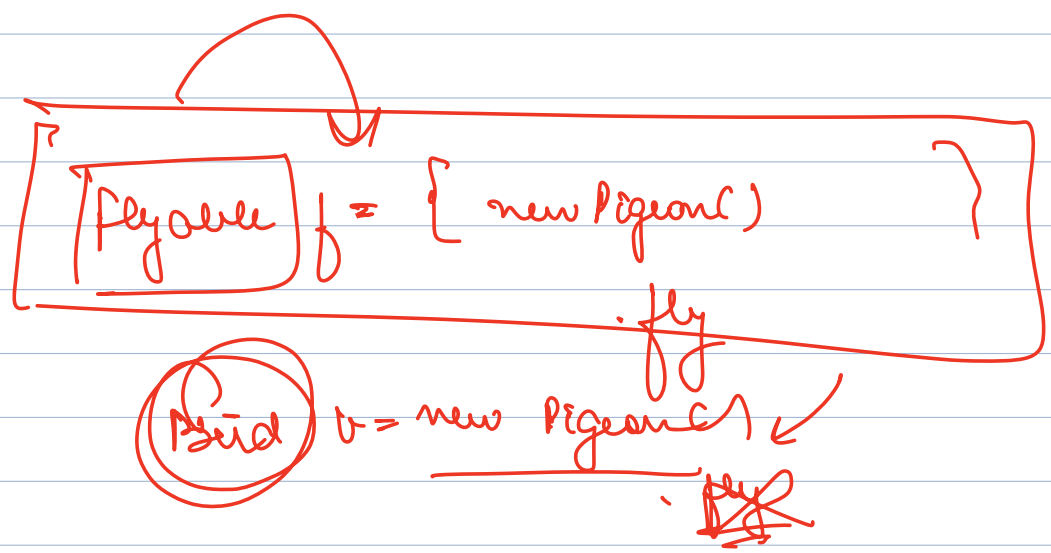
In an ideal scenario, I would not want to take a lock if obj is already initialized.

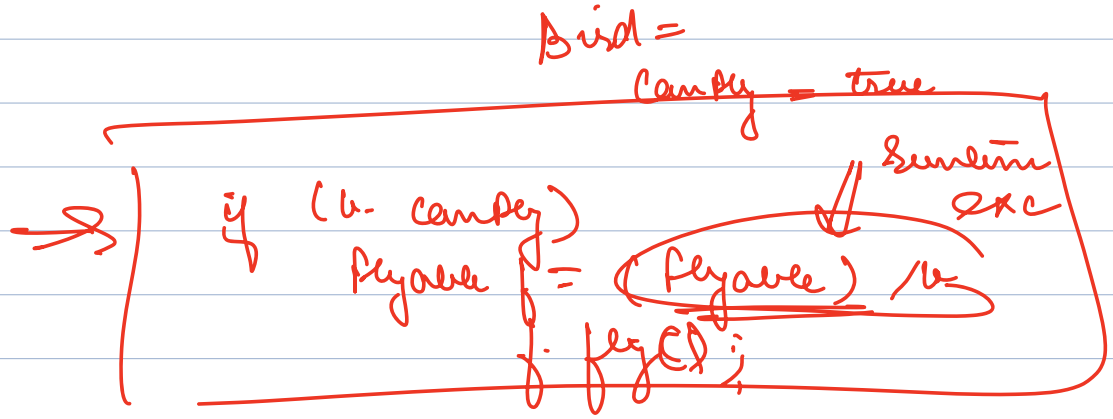


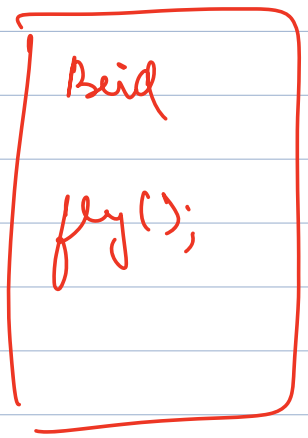
① Did I take lock if obj was not null

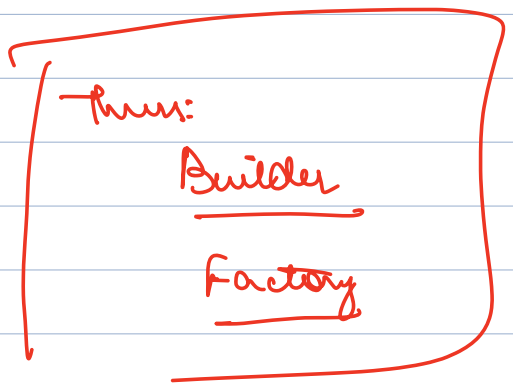
No

② Do I have conc issues




Bird =
canFly = true
if (b.canFly)
flyable = (flyable) b.
fly();
Sending exc


Bird
fly();


Ans:
Builder
Factory