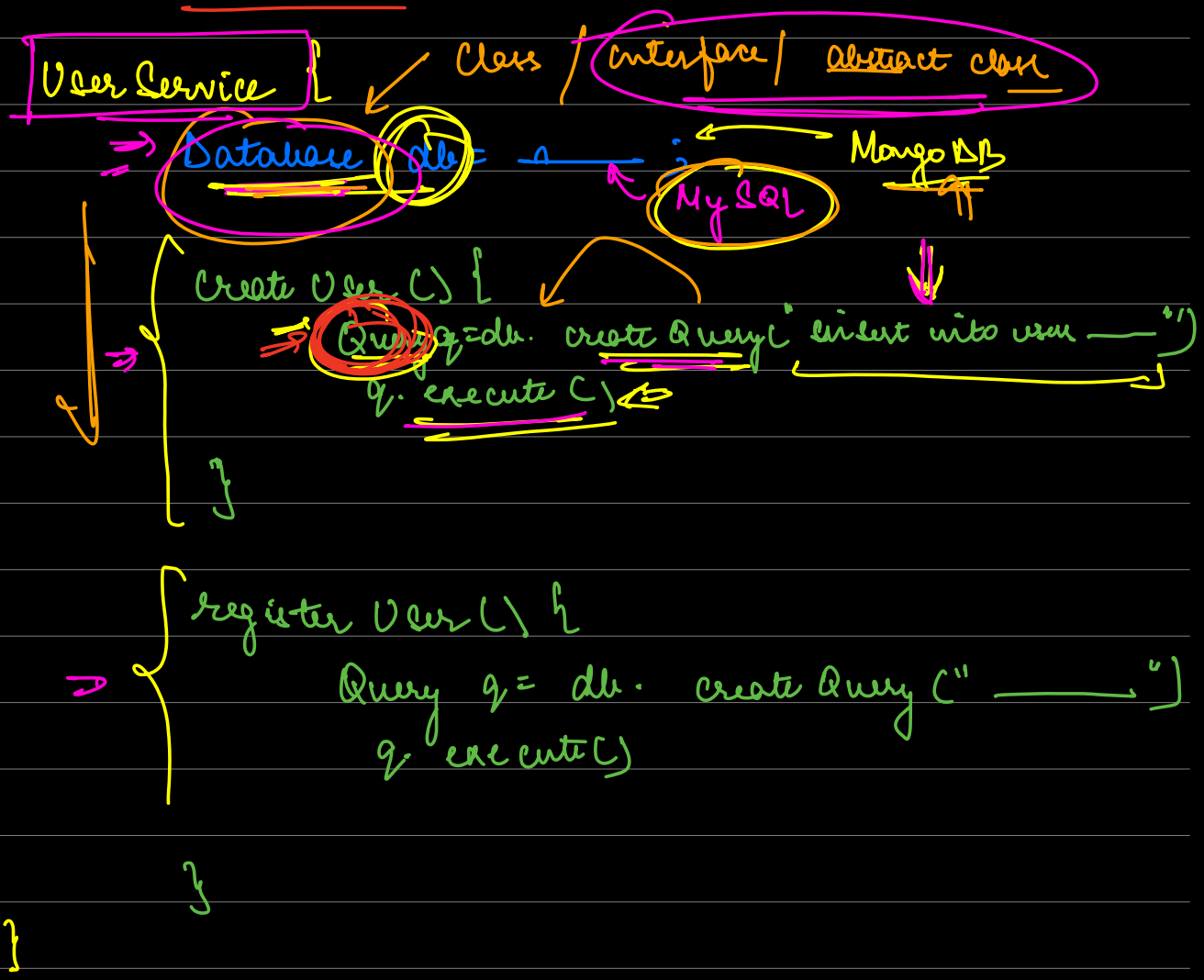
Agenda

Factory Design Pattern
- Factory Method
- Abstract Factory
- Practical Factory

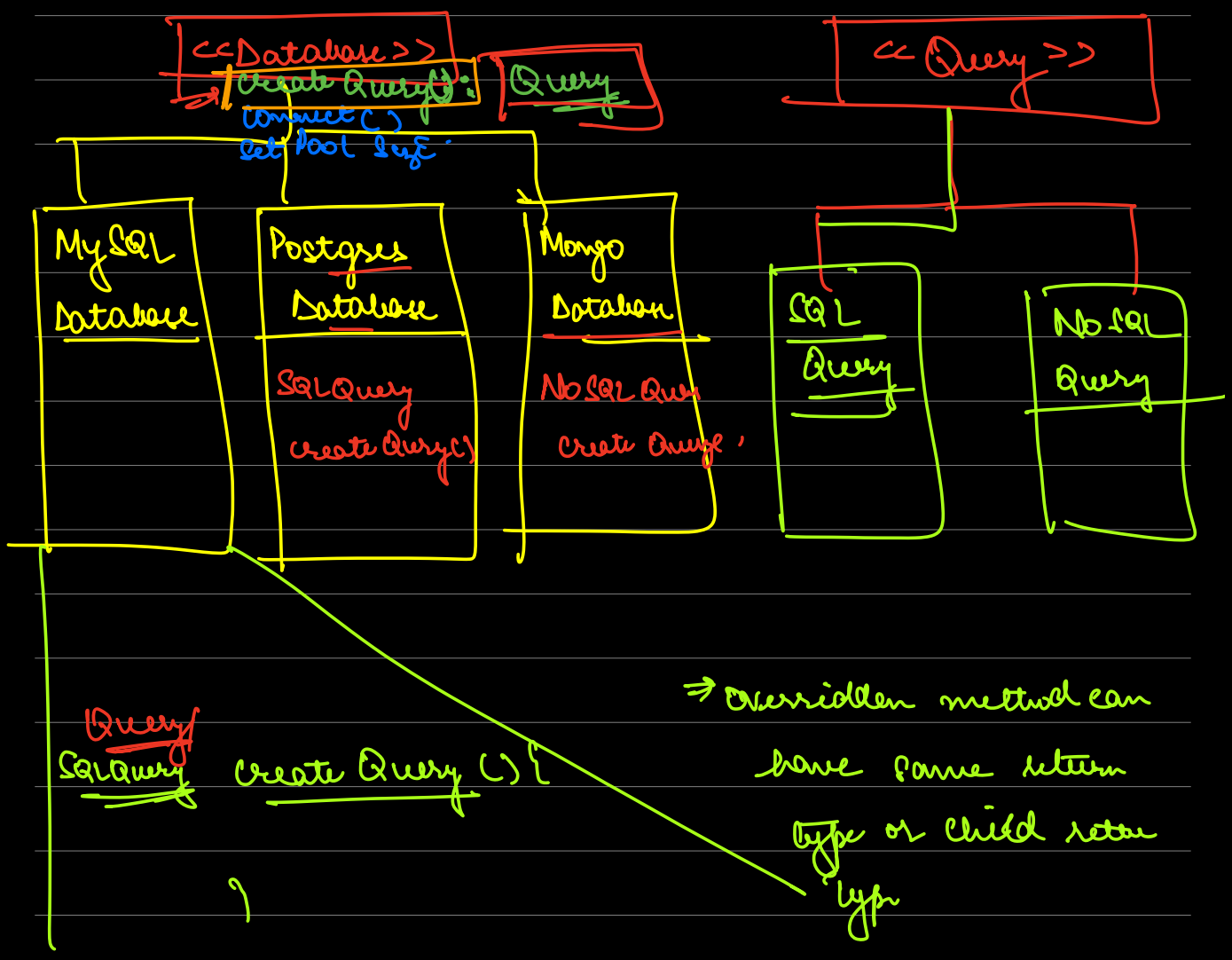
Not defined in GoF book

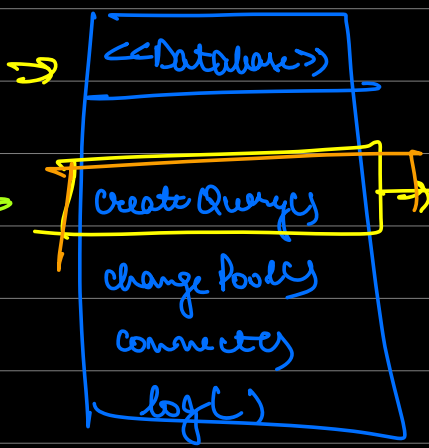
FACTORY METHOD



→ If Database was a class, Dependency Inversion design principle will get violated.

→ We would want DB to be an interface/Abstract class so that in future if we want to change DB, we can change that easily.





returning a query object

Purpose of Create Query() is to do nothing but return a new object of corresponding query. → Factory Method

inside Database create a method to return a query.

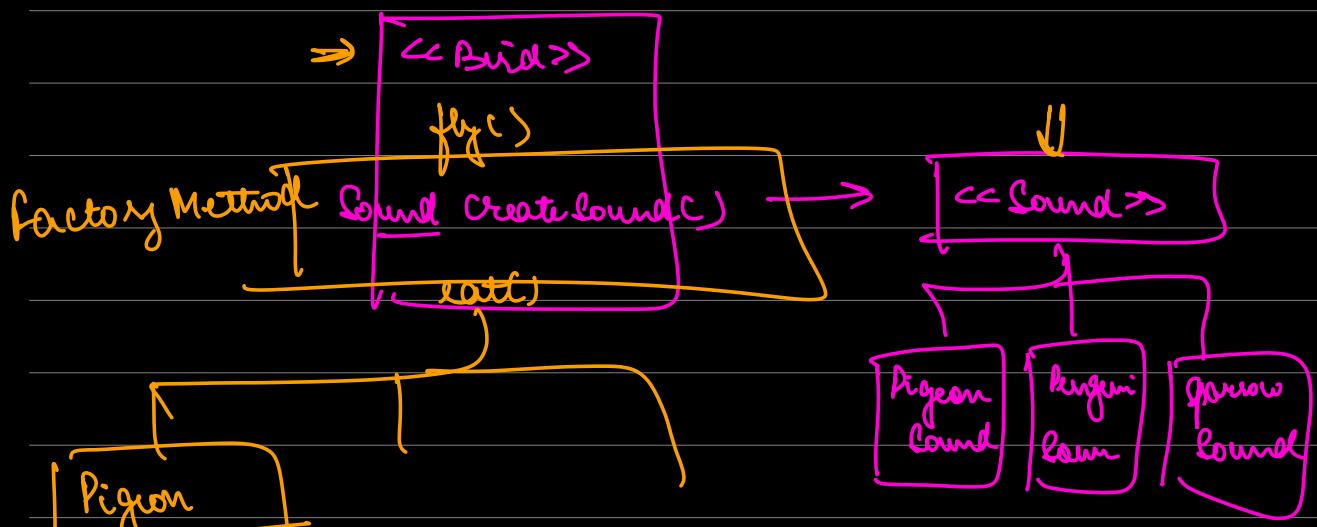
User Service {
Database db = ...
Query query;

User Service () {
if (db instance of MongoDB)
query = new No SQL Query

instead of writing this

OCP

}
}



Pigeon

CreateSound() →
PigeonSound()

OCP

Bird
CreateSound() {
if bird instance of Pigeon
else ——— Penguin
}

<< Database >>

createQuery() => Query

connect()

changeURL()

increasePoolSize()

createTransaction() => Transaction

createUpdater() => Updater

getVersion() =>

← Responsibilities

① attributes and methods needed for a Database

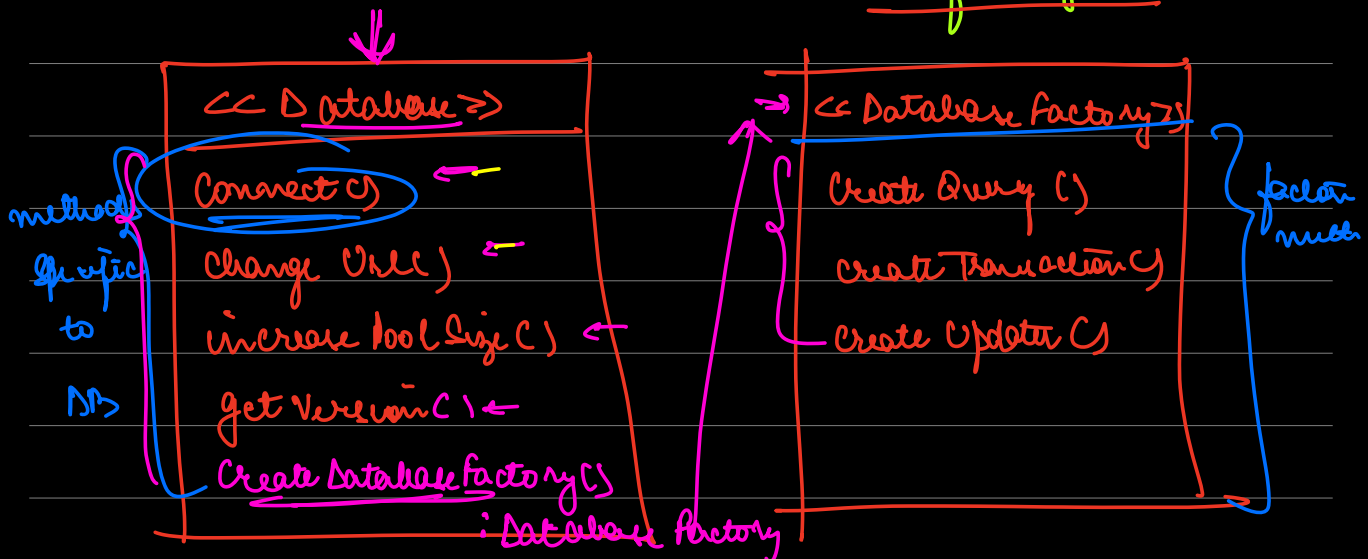
② A lot of factory methods to get corresponding obj of diff types

→ SRP is getting violated

Abstract Factory → Divide the class into 2 classes.

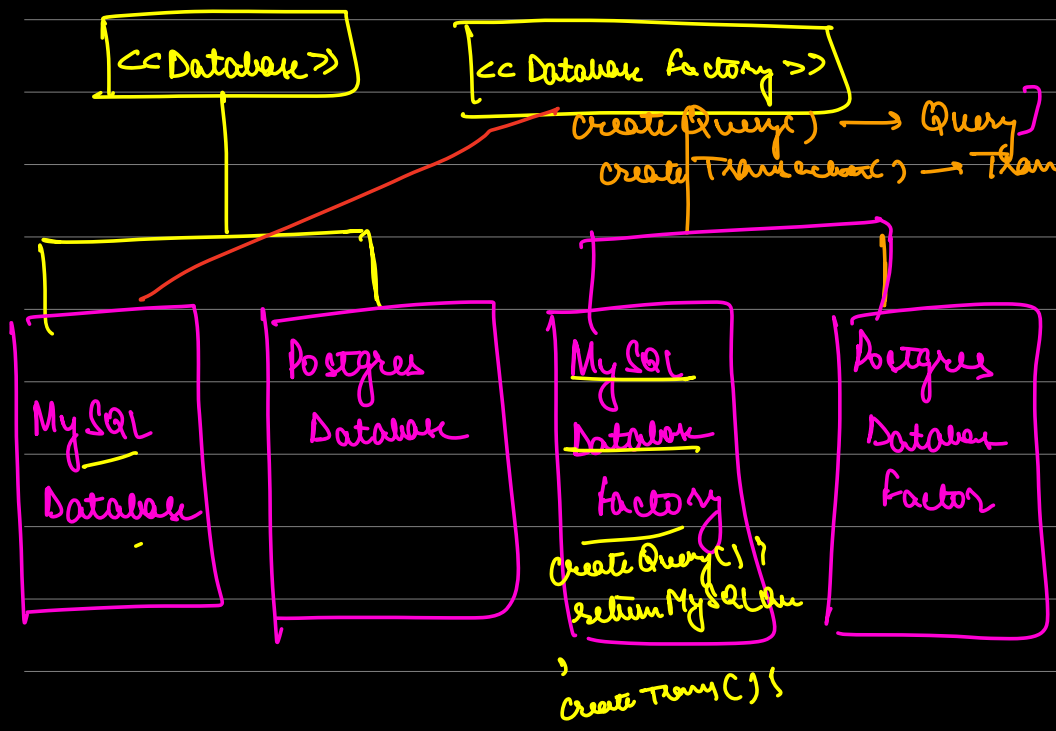
1 to have attributes

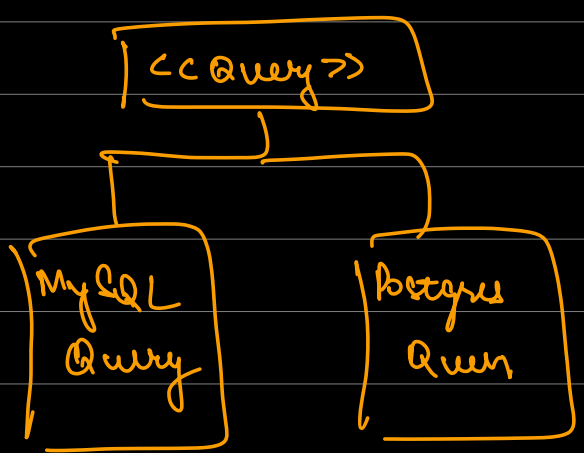
2 to have all factory methods



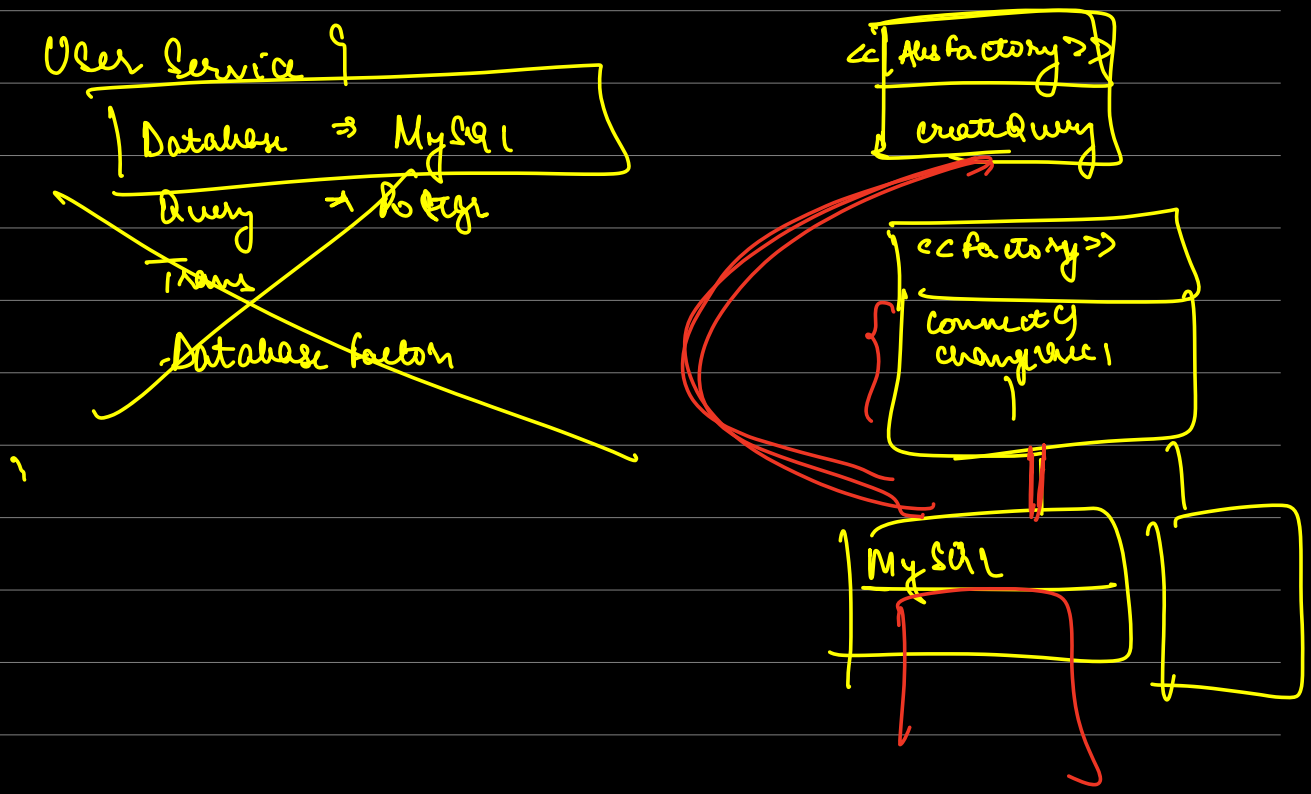

User Service {
Database db;
Database factory dbf;
MySQL DB
MySQL DBF

}



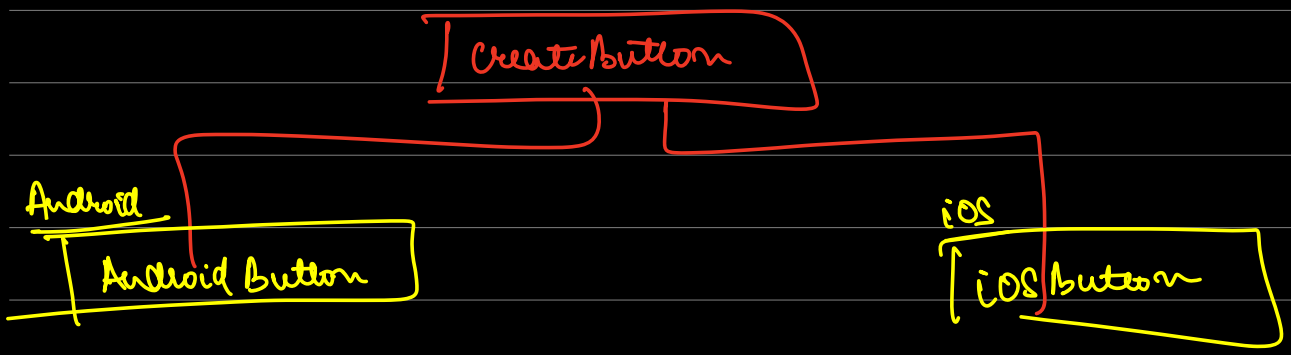


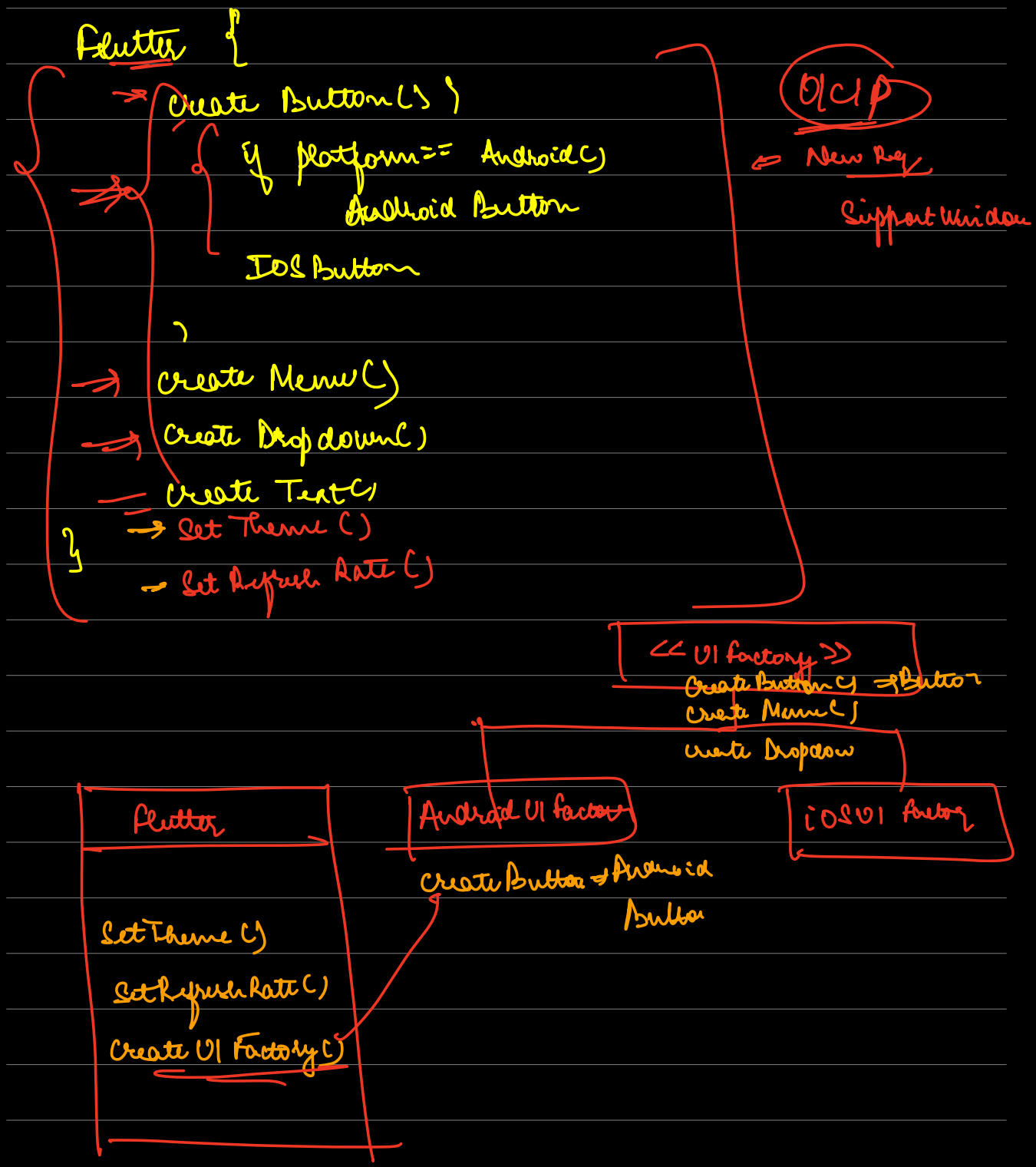

```
UserService {
    Database db;

    createUser() {
        db.createFactory()
        Obj.createQuery()
    }
}
```



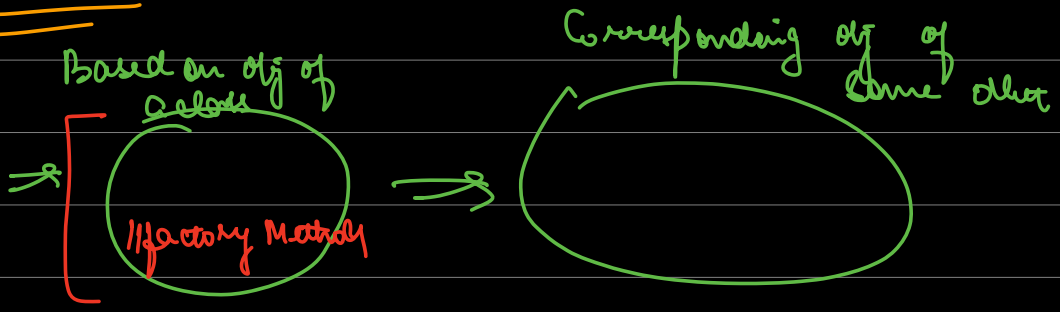
Real Use Case : UI Libraries

- Frameworks to create apps
- Flutter / React Native : Cross Platform Frameworks



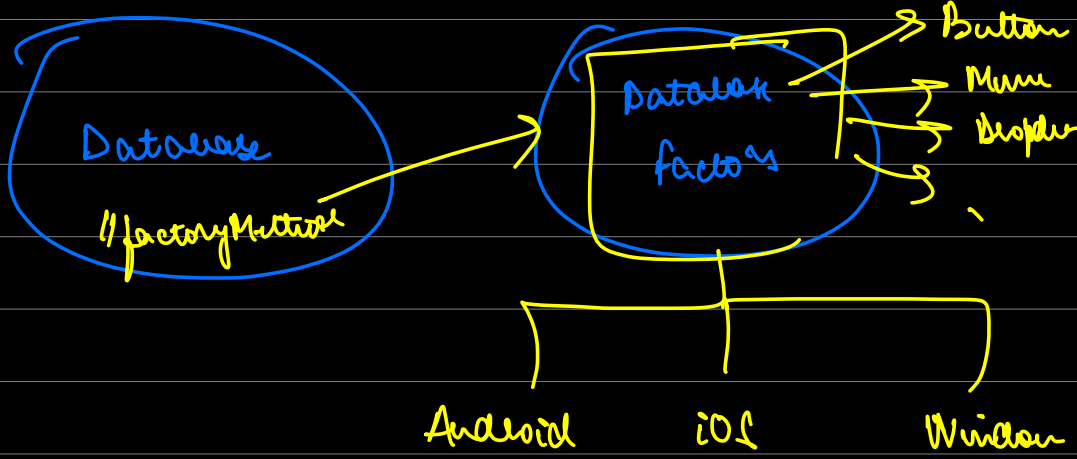

Flutter {
Create Button()
if platform == Android()
Android Button
IOS Button
}
create Menu()
Create Dropdown()
Create Text()
Set Theme()
Set Refresh Rate()
}
OCP
New Req
Support Window
<< UI factory >>
Create Button() → Button
Create Menu()
create Dropdown
Flutter
Android UI factory
iOS UI factory
Set Theme()
Set Refresh Rate()
Create UI Factory()
Create Button → Android Button

SUMMARY



But if lot of factory methods ⇒ SRP is getting violated in the class.

⇒ We break it into 2



10:26 PM Break ⇒ Code for Abstract Factory
⇒ Practical Factory

Factory ⇒ Anything that allows us to create new Things.

DatabaseFactory ⇒ allowing us to create corresponding objects of other classes

FactoryMethod ⇒ Corresponding Class
AbstractFactory ⇒ Corresponding Class.

PRACTICAL FACTORY

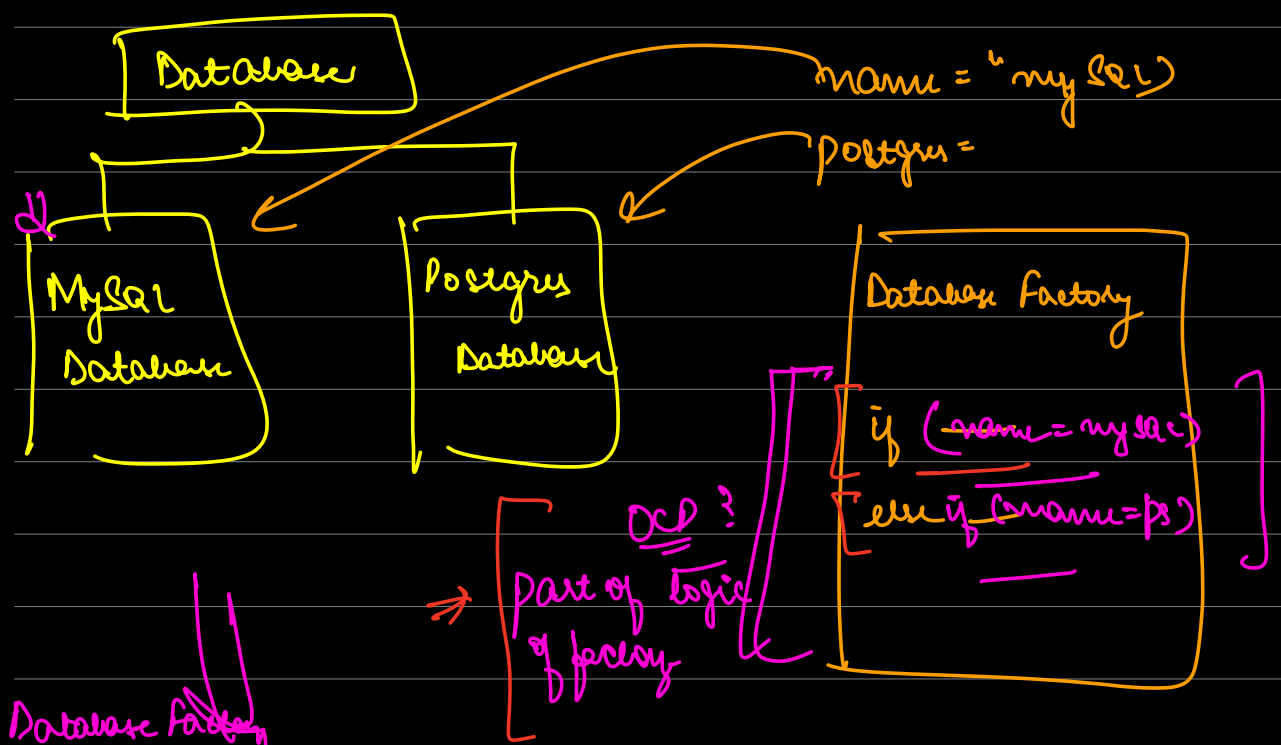
→ Create obj of the same class
→ Class ⇒ (X) Factory
↳ will have methods to create Objects of X based on different criteria

X
DatabaseFactory {

Database createDatabaseByName(name){
If (name == mySQL)
return new MySQL Database()
psql → PSQL Database()
}

Practical factory

Case Studies → Whenever there are multiple variants of a class and you want to create an object of the correct variant based on param



Database Factory

getDBConnection() {

DatabaseType }

MYSQL

Mongo =

Postgres

= Oracle

}

1 → MySQL

Switch (DatabaseType) {

case MySQL:

case PSQL:

}

Bird

BirdFactory {

Bird createBird forType (String type)
if type == crow
return crow

}

}

Client {

Bird b = BirdFactory.createBirdForType(spa

}

Factory Method

a method in a class that returns a new object of related class

Abstract Factory

a collection of factory methods

Practical Factory

whenever multiple variants of class / multiple create obj of correct one based on parameter

Reg

- Multiple furniture combⁿ
  - ↳ Chair
  - ↳ Sofa
  - ↳ Table

↳ store these combⁿ and get corresponding chair, sofa,

Req : Based upon name, get correct concl^n