Agenda : Factory Design Patterns

- Factory methods
- Abstract factory
- Practical factory
- this is actually not there in gang of 4 book but it is most commonly used.

① Factory Method

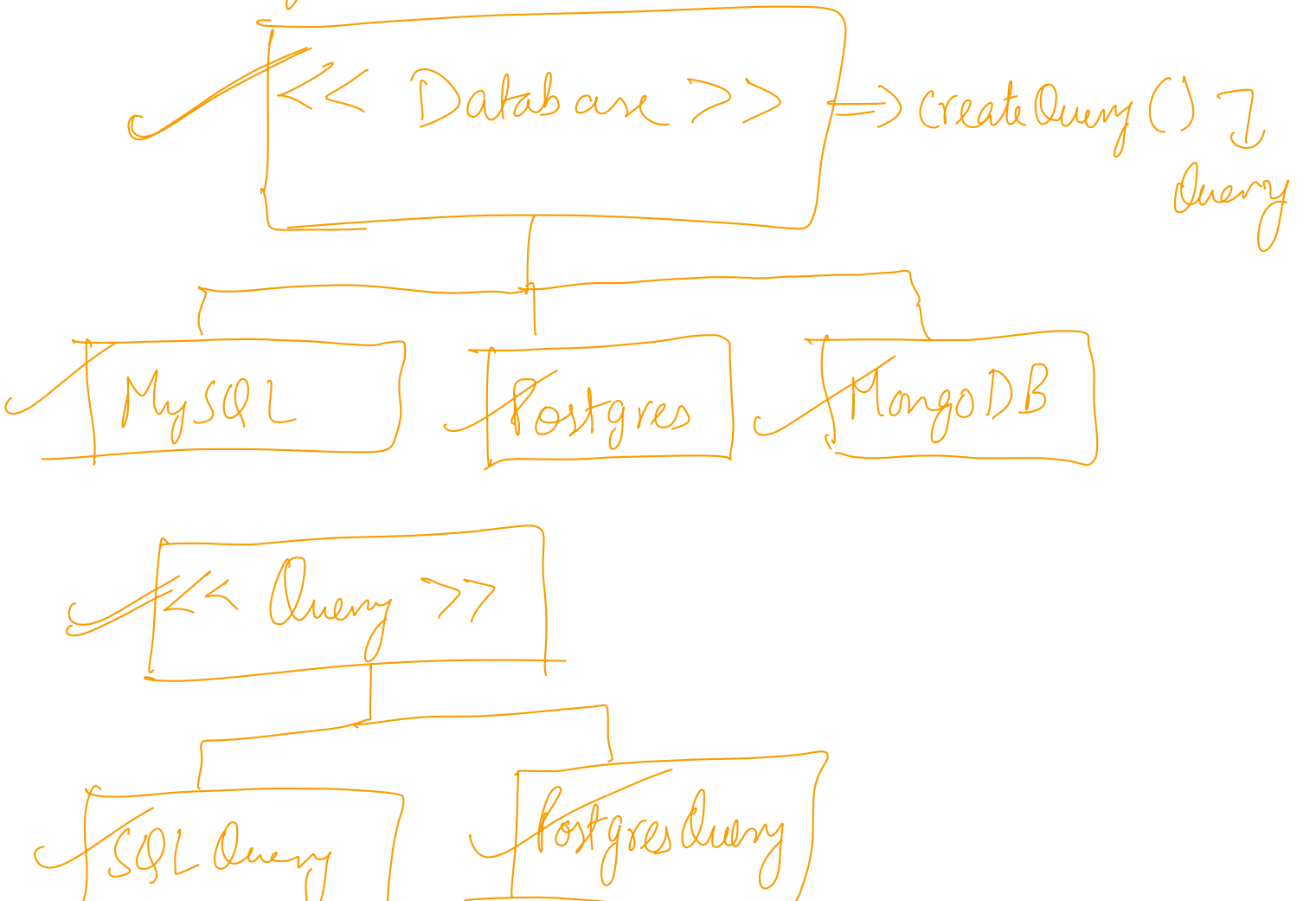
```
UserService {
    ① Database db = ____;
    ② createUser() {
        Query q = db.createQuery();
        q.execute();
    }
    ③ registerUser() {
        Query q = db.registerQuery();
        q.execute();
```

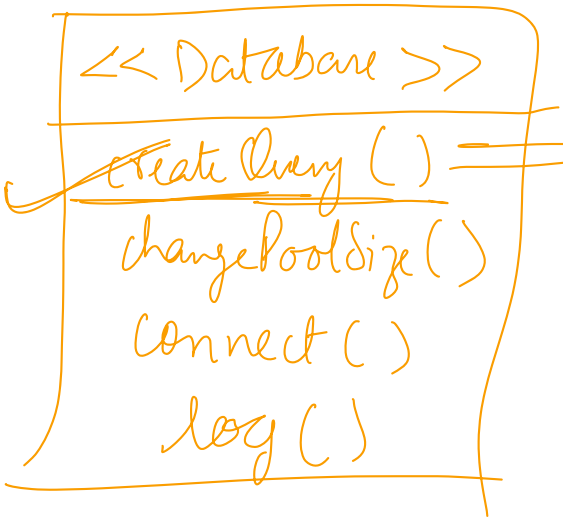
{ Insert into users...

⟹ Database is mostly of type Interface / Abstract class.

⟹ If Database is a concrete class, it would violation Dependency Inversion principle, as userservice is already a concrete class.

⟹ As our database can have multiple types of implementations.





⟹ returning a Query
||

Factory methods ⟹ methods in an interface or a parent class that returns an object of another class.

⟹ Benefit of factory methods

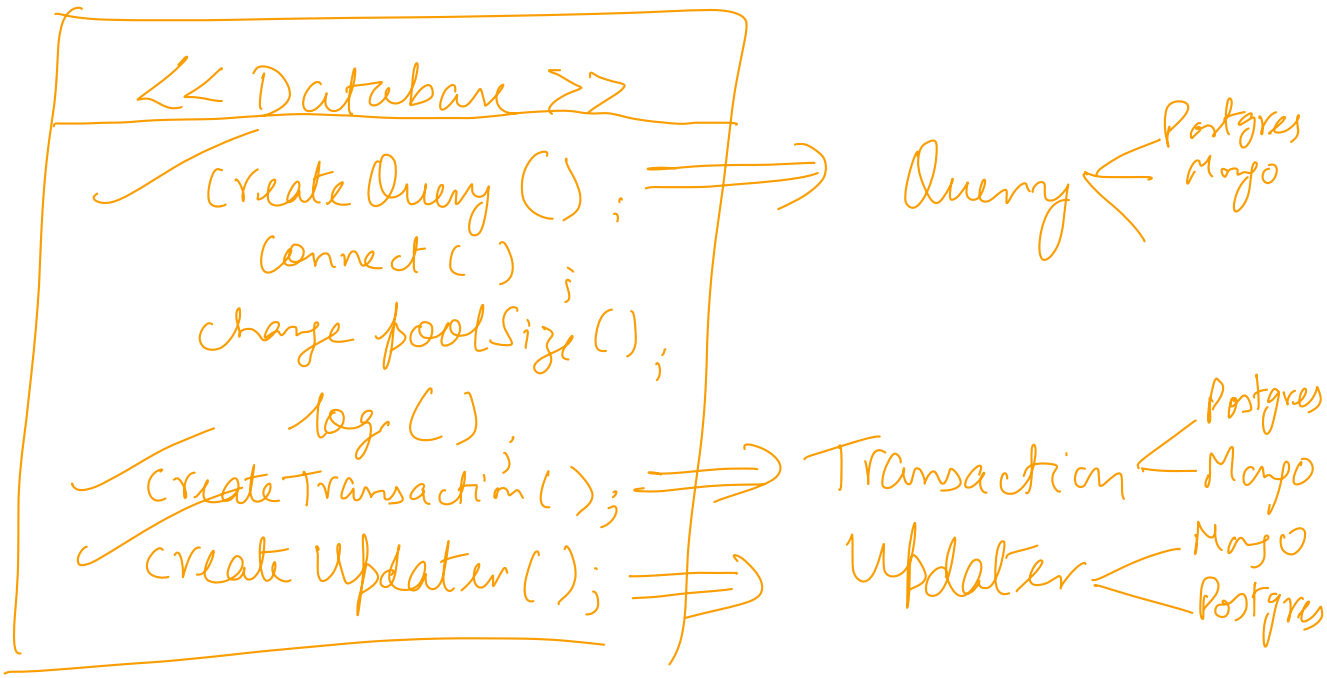
```
UserService {
    Database db;
    Query query;

    UserService() {
        if (db instanceOf MongoDb){
            query = new MongoQuery();
        }
        if (db instance of Postgres){
```

```
        query = new Postgres();
    }
}
```

⟹ will result in a violation of OCP.

⟹



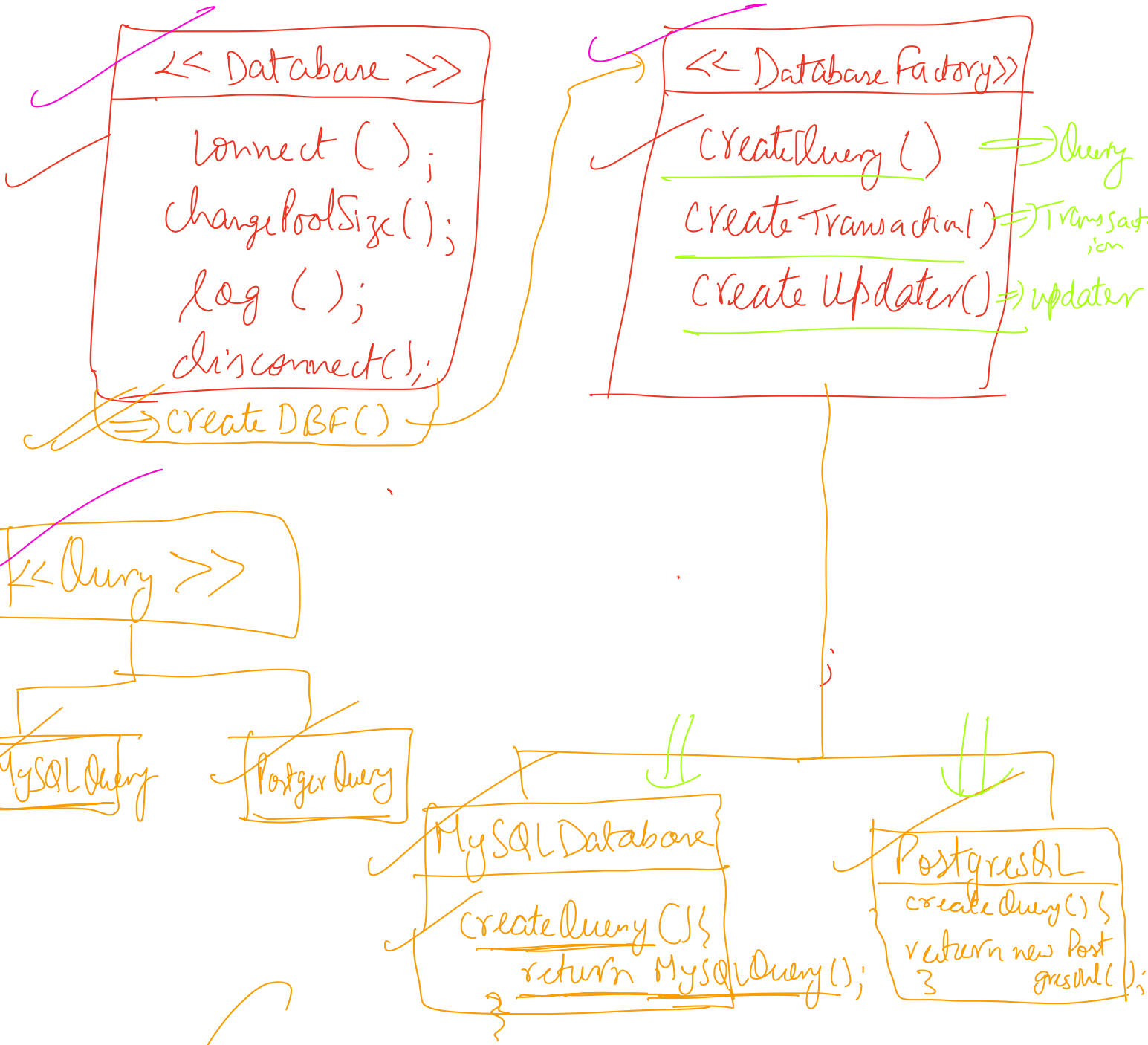
① There are methods that are specific to the database

② There are 3 factory methods as well that will give me object of corresponding database type (Query, Transaction, Updater)

↳ SRP is getting violated here.

↳ this is where Abstract factory comes into the picture.

1. Divide the class/interface into 2 classes/interface
2. One for normal methods
3. One for factory methods.



UserService {

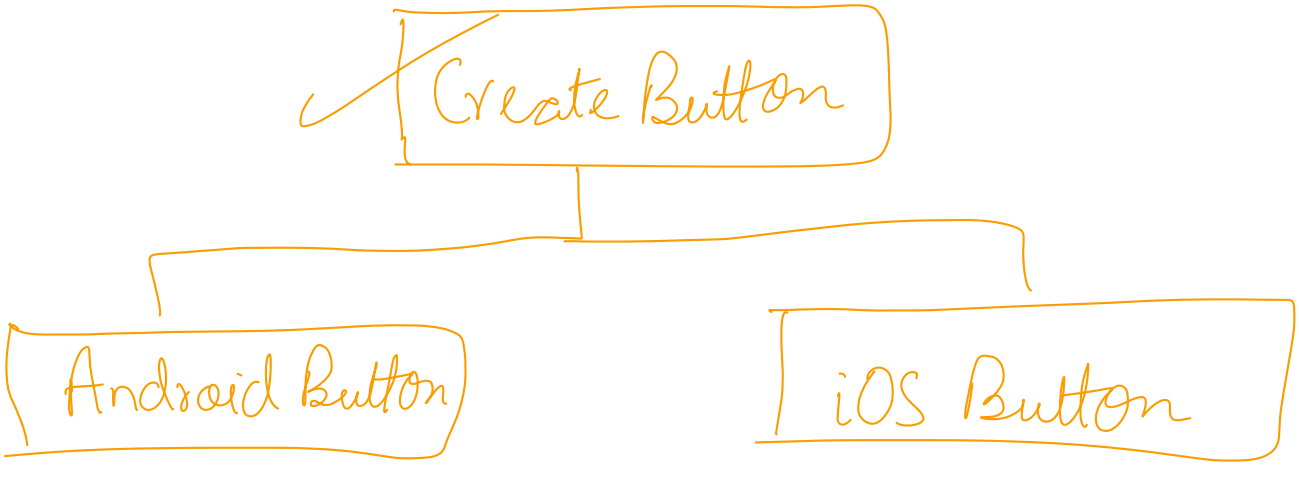
Database db;
DatabaseFactory dbf;

dbf = db.createDBF();
dbf.createQuery();

}

⟹ Real use case: UI libraries

↳ Flutter, React-Native ⟹ cross platform development frameworks.



Flutter {

CreateButton() {

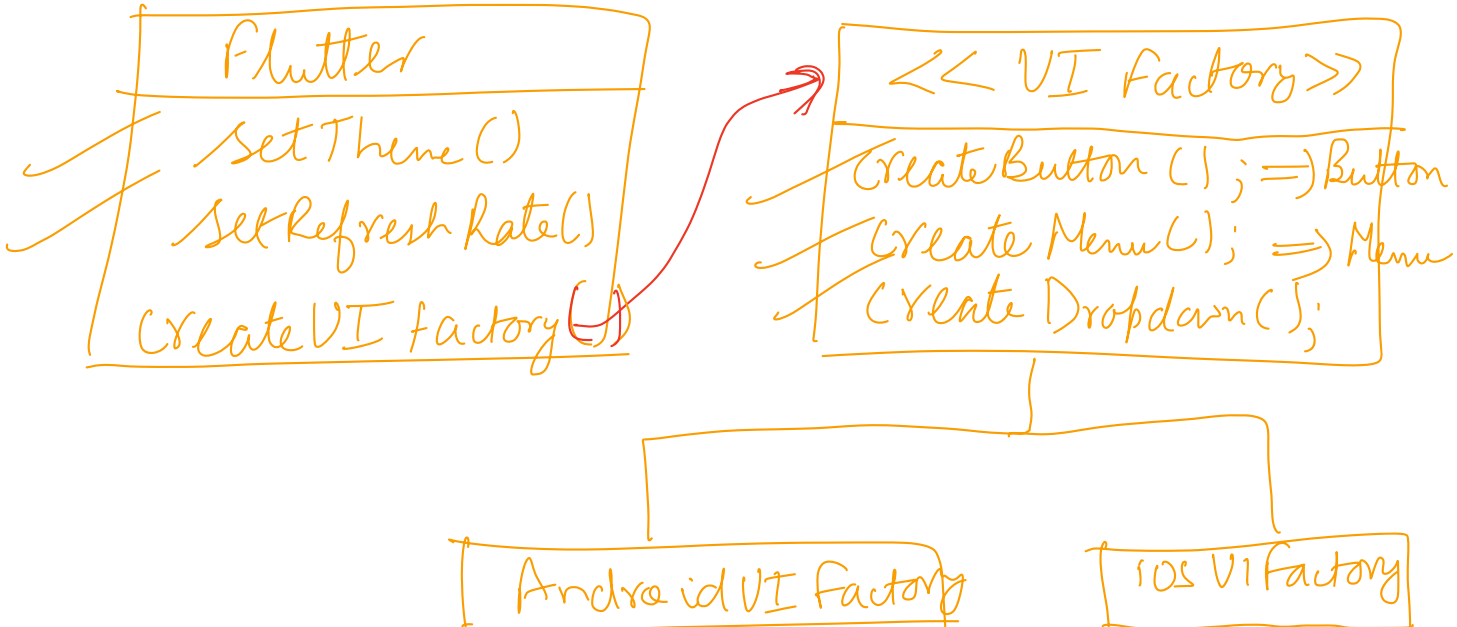
OCP violation

```
if (platform == iOS)
    return iOSButton
if (platform == android)
    return androidButton
}
```

```
CreateMenu() { ... }
CreateDropdown() { ... }
}
```

⇒ Flutter would also have generic methods ⇒ SetTheme()
SetRefreshRate()

⇒ SRP & OCP both are getting violated



createButton() ⇒ Android Button

createButton() ⇒ iOS Button

Summary

Android or iOS

① We have an object of a particular class, and based on that, we need corresponding object of some other class.

⟶ Android Button
iOSButton.

② If a lot of factory methods SRP gets violated, and we go into abstract factory pattern.

⇒

```
UIFactory {

    Button createButton();
    Menu createMenu();
    Dropdown createDropdown();

}
```

① Factory Method ⇒ returns me an

object of corresponding class

② Abstract factory ⇒ returns objects of corresponding class.

③ Practical factory design ⇒ returns objects of same class.

⇒ UI factory factory
returning object of
① Andoi UI factory
② IOS UI factory.

```
DatabaseFactory {

    Database createDatabaseByName(String name){
        if (name == mySQL)
            return new MySQL();
        if (name == Postgres)
            return new Postgres();
    }
}
```

⇒ Practical DBFactory will have methods which return databases

⇒ Whenever there are multiple variants of a class and you want to create an object of correct variant based on parameter ⇒ use Practical factory

⇒ If if-else is not there in Flutter/Database factory, then it will go in client ⇒ BAD Idea

Summary

- Factory Method ⇒ method in a class that returns object of related class
- Abstract factory ⇒ Collection of factory methods.
- Practical factory ⇒ whenever

multiple variants of class, are there
Create object of correct one based
on param passed in a
separate class.