# Factory Design Pattern

- Factory Method
- Abstract factory
- Practical factory. (Not a Design Pattern as per GOF book).

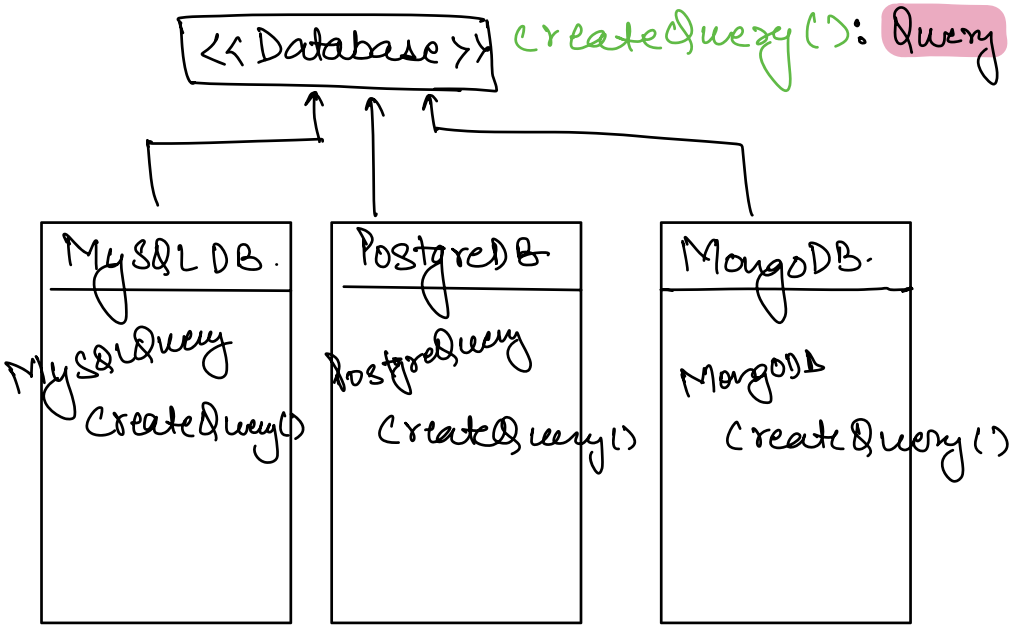
# FACTORY METHOD.

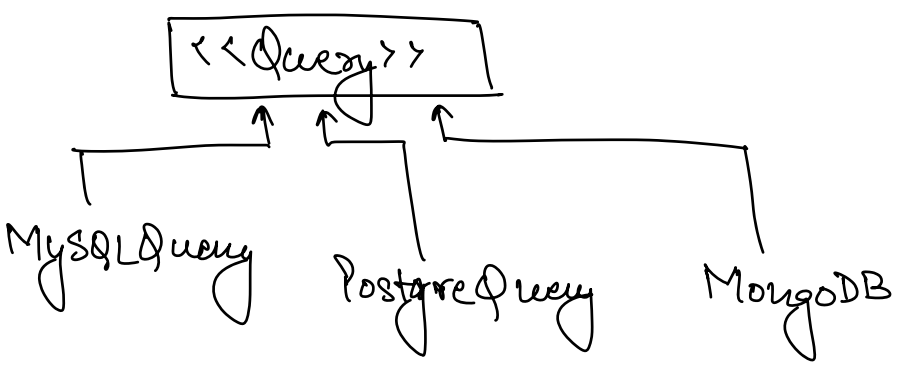
Class | Interface | Abstract Class.

```
UserService {
    Database db = ____________;
                  MySQL | MongoDB

    createUser() {
        Query q = "Insert into users ..... ";
        db.execute(q);
                   createQuery()
    }

    registerUser() {
    }

    fetchUser(name) {
        Query q = "select * from ..... ";
        db.execute(q);
    }
}
```

⇒ If Database was a class then Dependency Inversion principle will get violated.

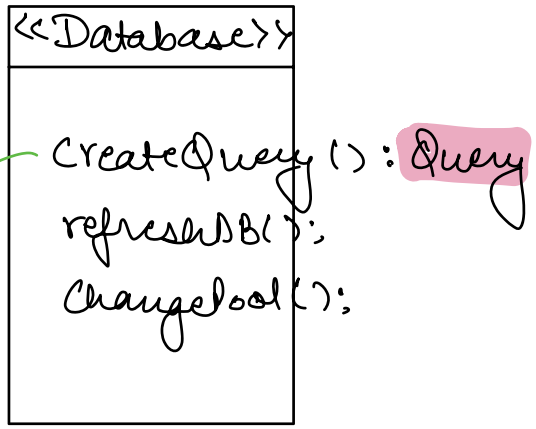
⇒ We would want DB to be an interface / abstract class so that in future if want to change the DB then we'll be able to do that easily.



⇒ createQuery()

Overriden method can return the object of any of the child.



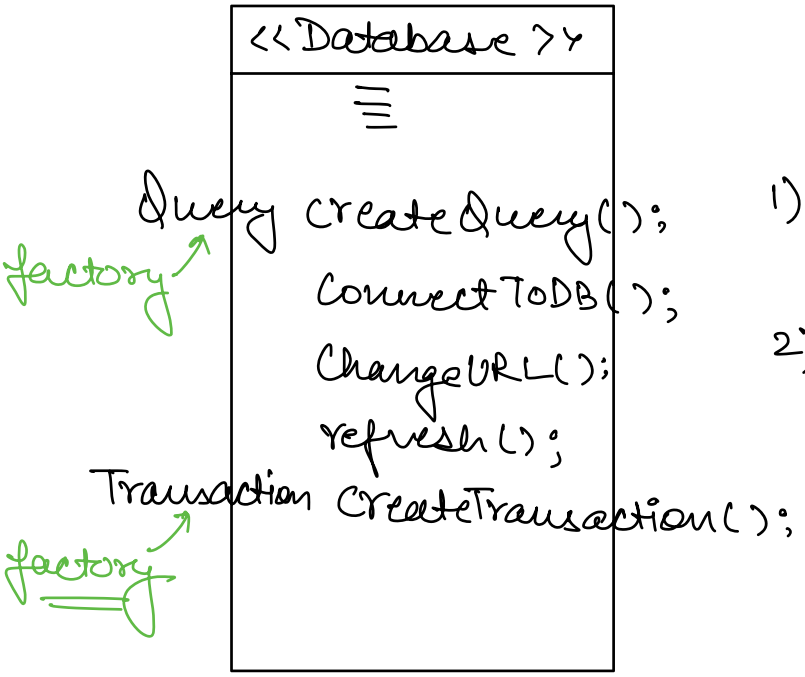

Purpose of createQuery() method is to return the object of corresponding Query.

⇒ Factory Method.

⇒
```
UserService {
    Database db = ————;
    Query q;
    if(db instanceOf MySQLDB){
        q = new MySQLQuery();
    }
    else if(db instanceOf PostgreDB){
        q = new PostgreQuery()
    }
    else{
        ==
    }
```

✗ OCP violation.

⇒




Responsibilities of Database interface.

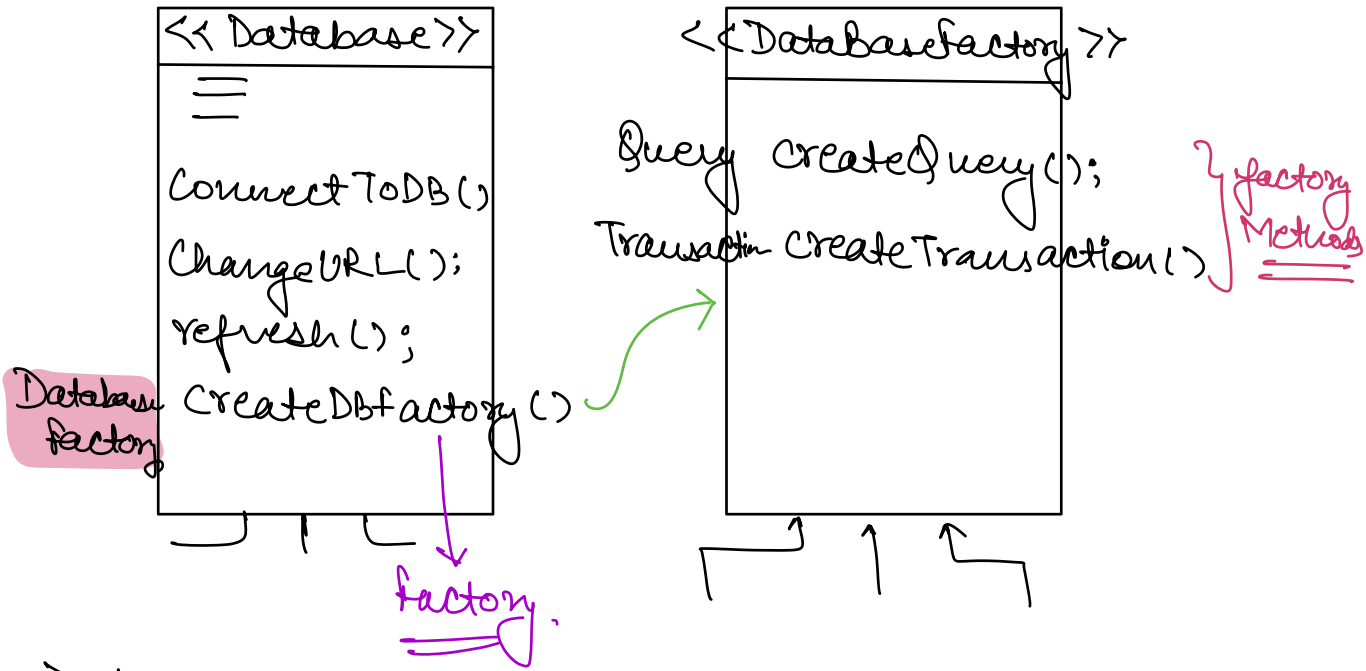
1) Attributes & Methods for Database.
2) A lot of factory methods.

⇒ SRP is getting violated.

⇒ **Abstract factory**: If we have lot of factory methods in an interface.

Divide the interface into 2 parts.

1) Interface with only attributes & Non factory methods.
2) Interface with only factory methods.

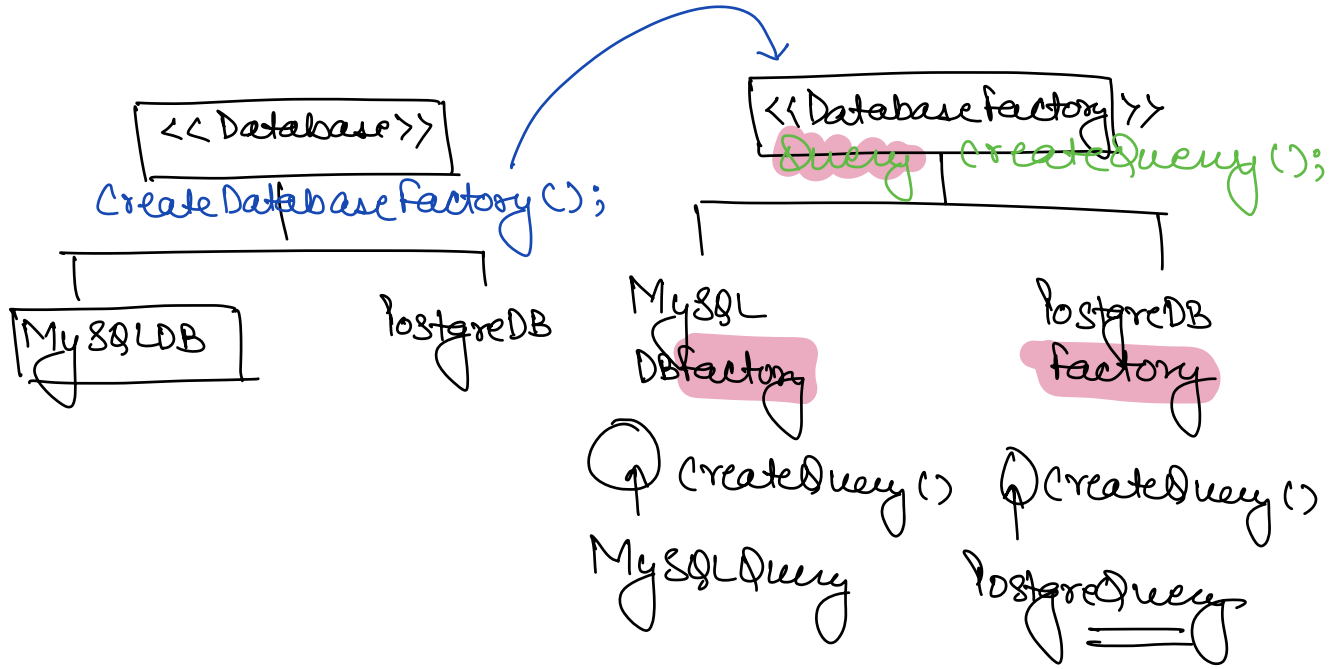



⇒ User {

Database db = —↓— ;

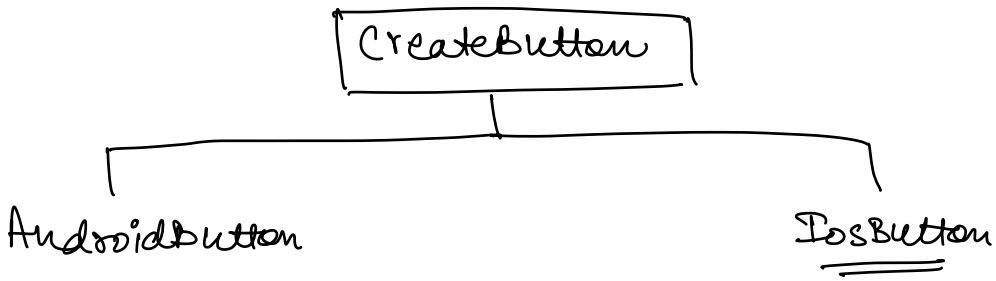
DatabaseFactory dbf ;

⇒ Based on the type of database we should create an object of Database factory




# Real Life Usecase

- UI framework
- Flutter : Cross platform framework.



Flutter {

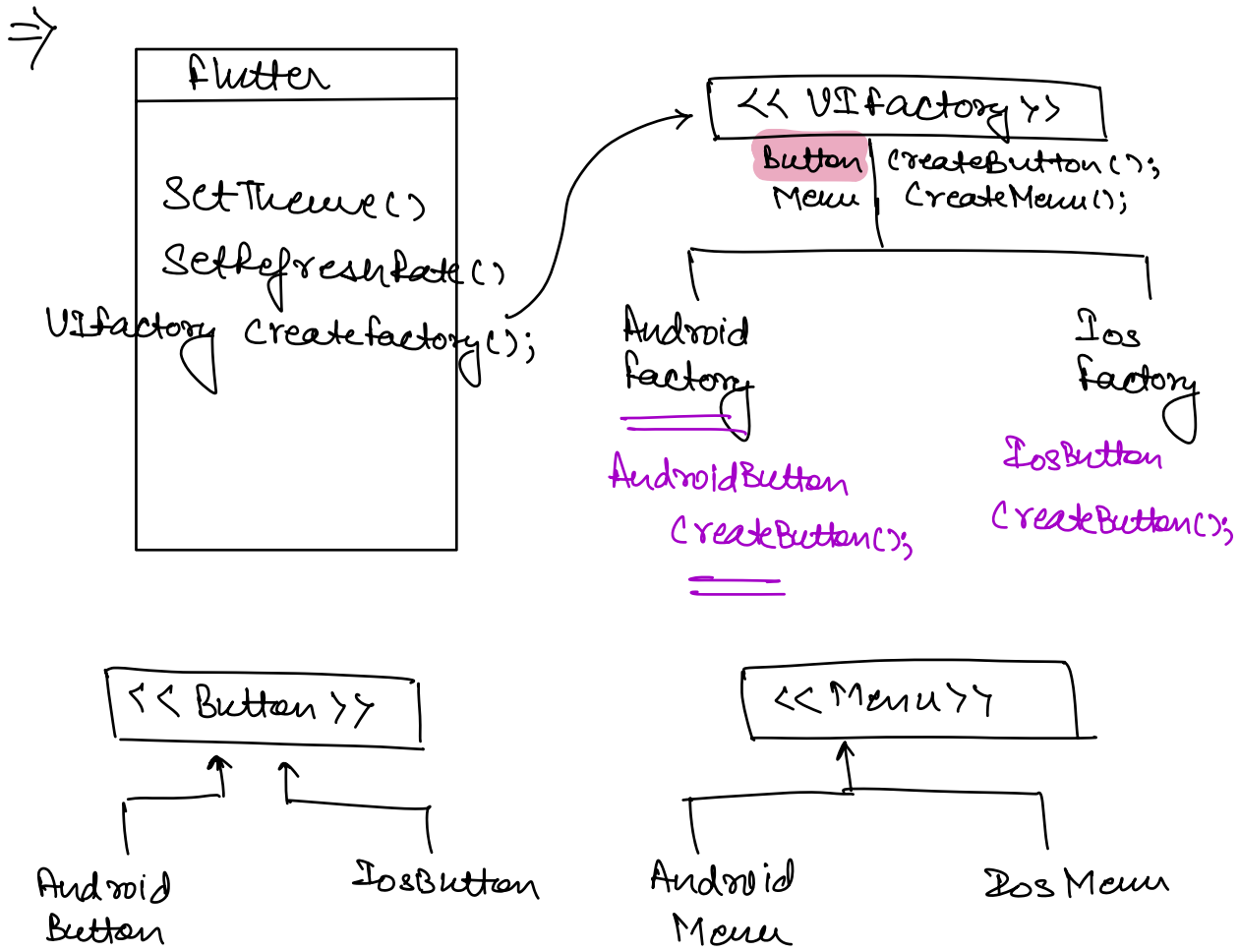
⇒ Returns an object of corresponding class. ⇒ Factory Method.

```
CreateButton(platform) {
    if (platform == "Android") {
        return new AndroidButton();
    }
    else if (platform == "Ios") {
        return new IosButton();
    }
}
CreateDropDown() {
}
CreateMenu() {
}
}
```

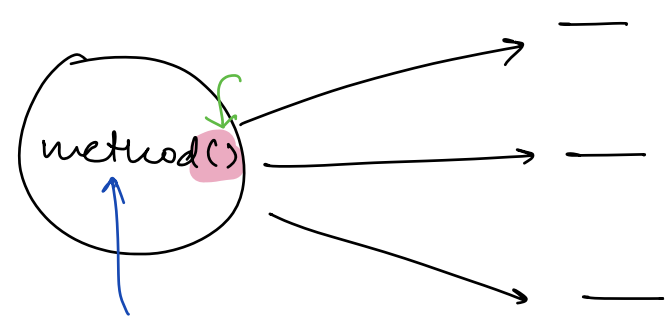
OCP ✗

Flutter will have lot of factory methods.

⇒ AbstractFactory Design Pattern




⇒ factory / Abstractfactory.




Based on some input we want to return the object of corresponding class } factory

Factory : Anything that allows us to create new things.

DatabaseFactory : Allowing us to create the object of corresponding Databases.

Abstract factory : When there are lot of factory methods.

Practical factory : Move the if-else logic to a new & designated place.

———— * ————