Agenda.

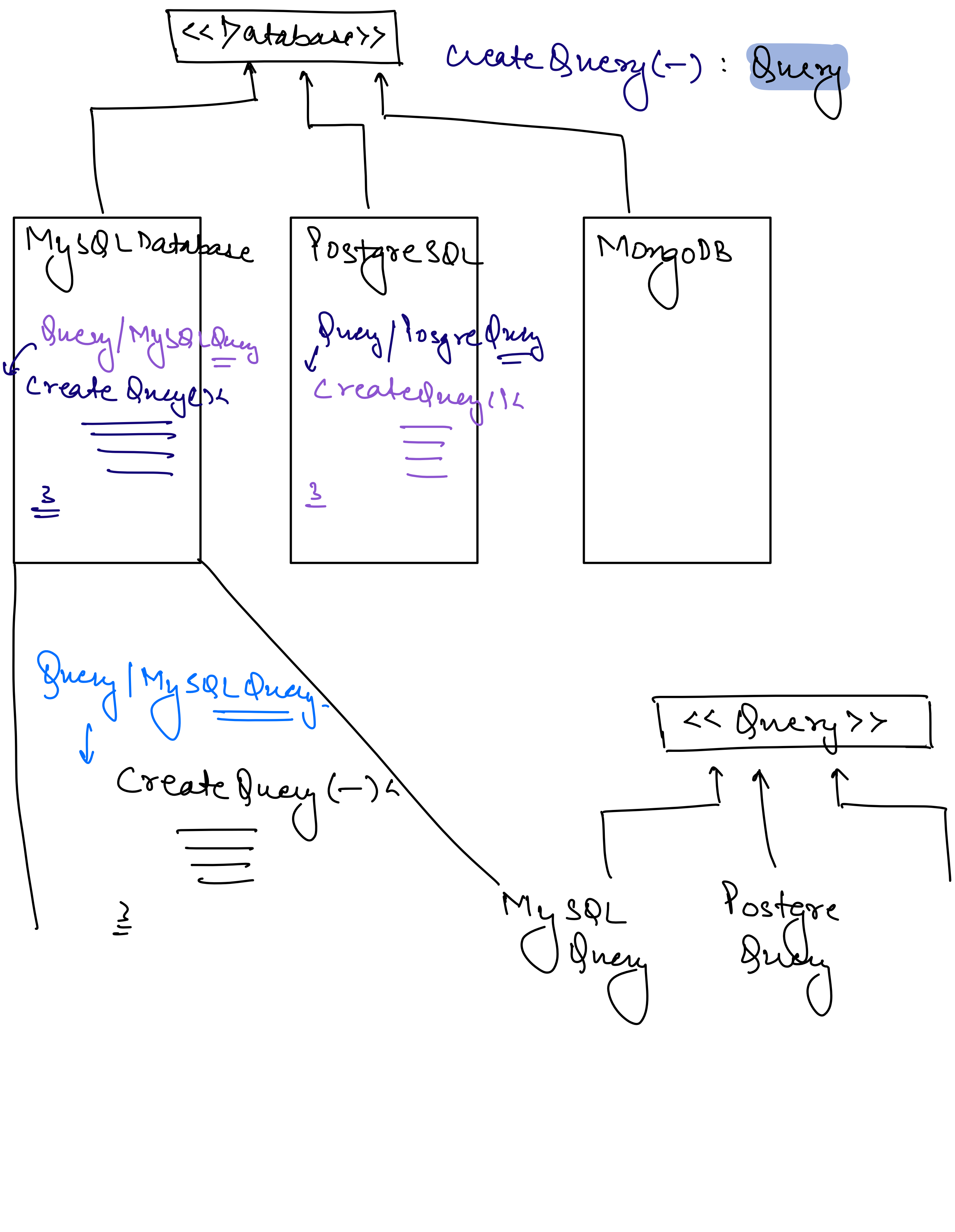
↳ Factory, Abstract Factory & Practical Factory.

#

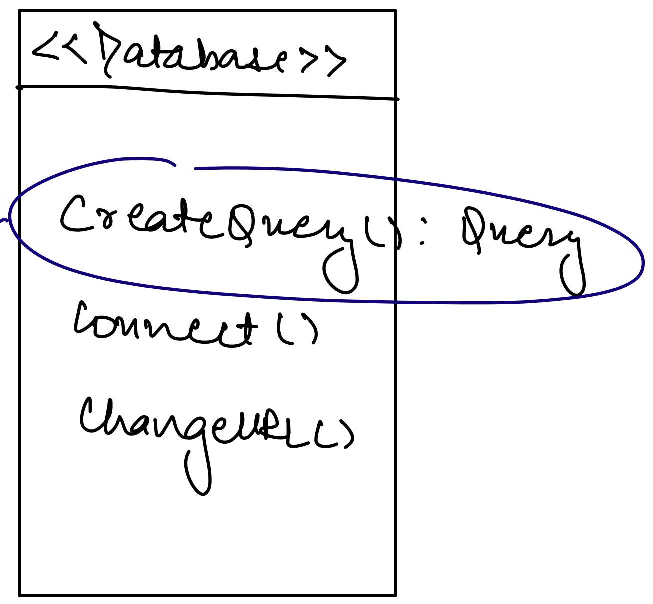
```
Class UserService {
    Database db;  ← MySQL.
                  ← PostgreSQL.
                  ← MongoDB.

    CreateUser ( —— , —— , —— , —— ) {
        Query q = db.createQuery("——");
        q.execute();
    }

    deleteUser( —— ) {
        ——
        ——
        ——
    }
}
```

⇒ If Database is a Normal class then it will violate Dependency Inversion Principle

⇒ Ideally Database should be an interface, so that it would be easier for us to change the underlying object in DB reference.



<<Database>>
createQuery(–) : Query
MySQLDatabase
Query/MySQLQuery
createQuery(){
}
PostgreSQL
Query/PosgreQuery
createQuery(){
}
MongoDB
Query/MySQLQuery
createQuery(–){
}
<<Query>>
MySQL Query
Postgre Query



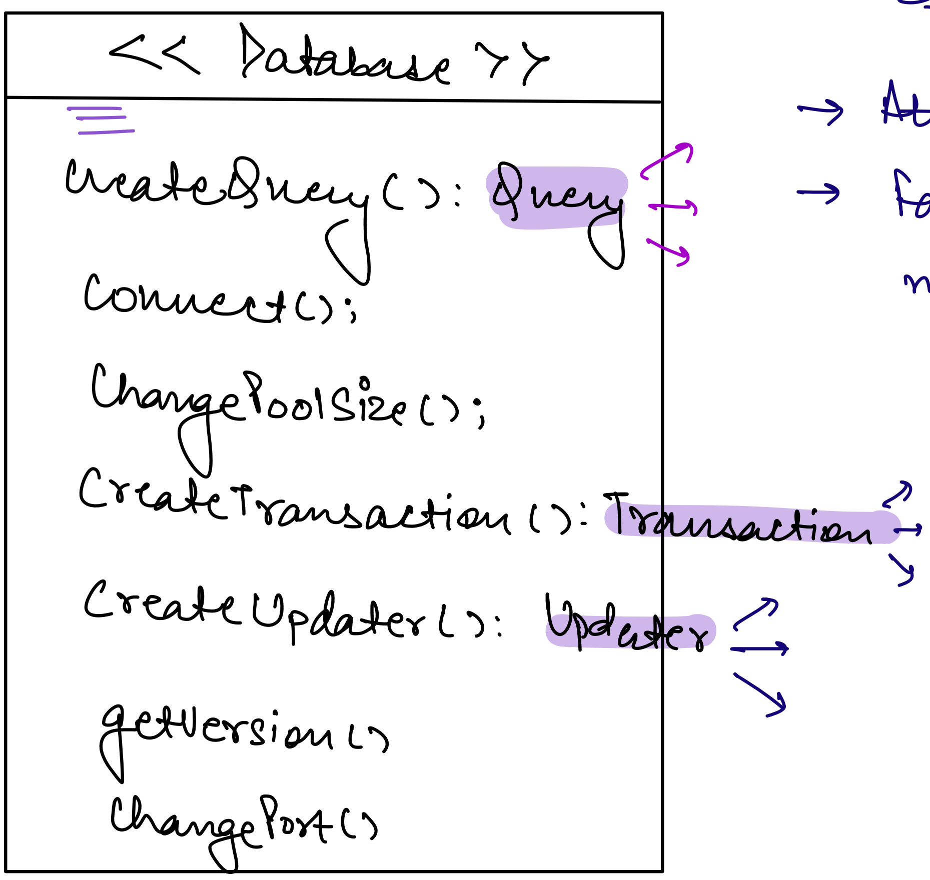
Purpose of Create Query Method.

=> It should return an object of corresponding Query.

=> Factory Method.

```
=> User Service {
       Database db;
       Query q;                              OCP. | SRP.
       createQuery() {
           if (db is MySQL) {
                  q = mySQL query()
           }
           else if (db is Posgre) {
                  q = Postgre Query()
           }
           ______
           ______
           ______
       }
   }
```

⇒

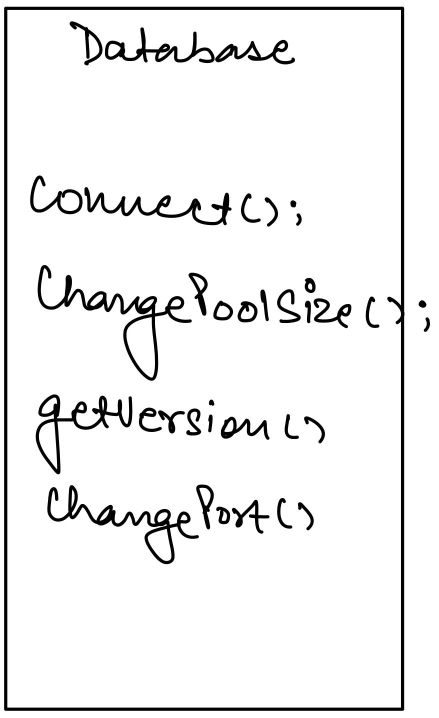


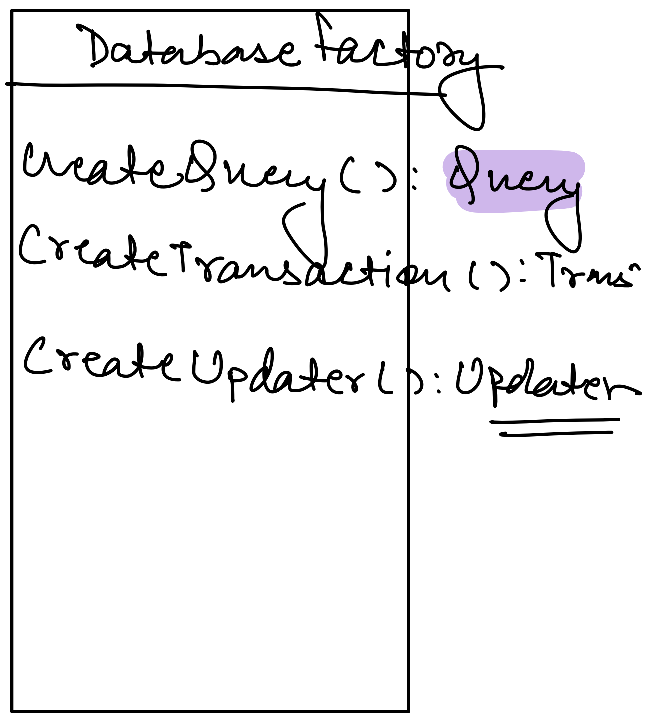
Responsibilities.

→ Attributes

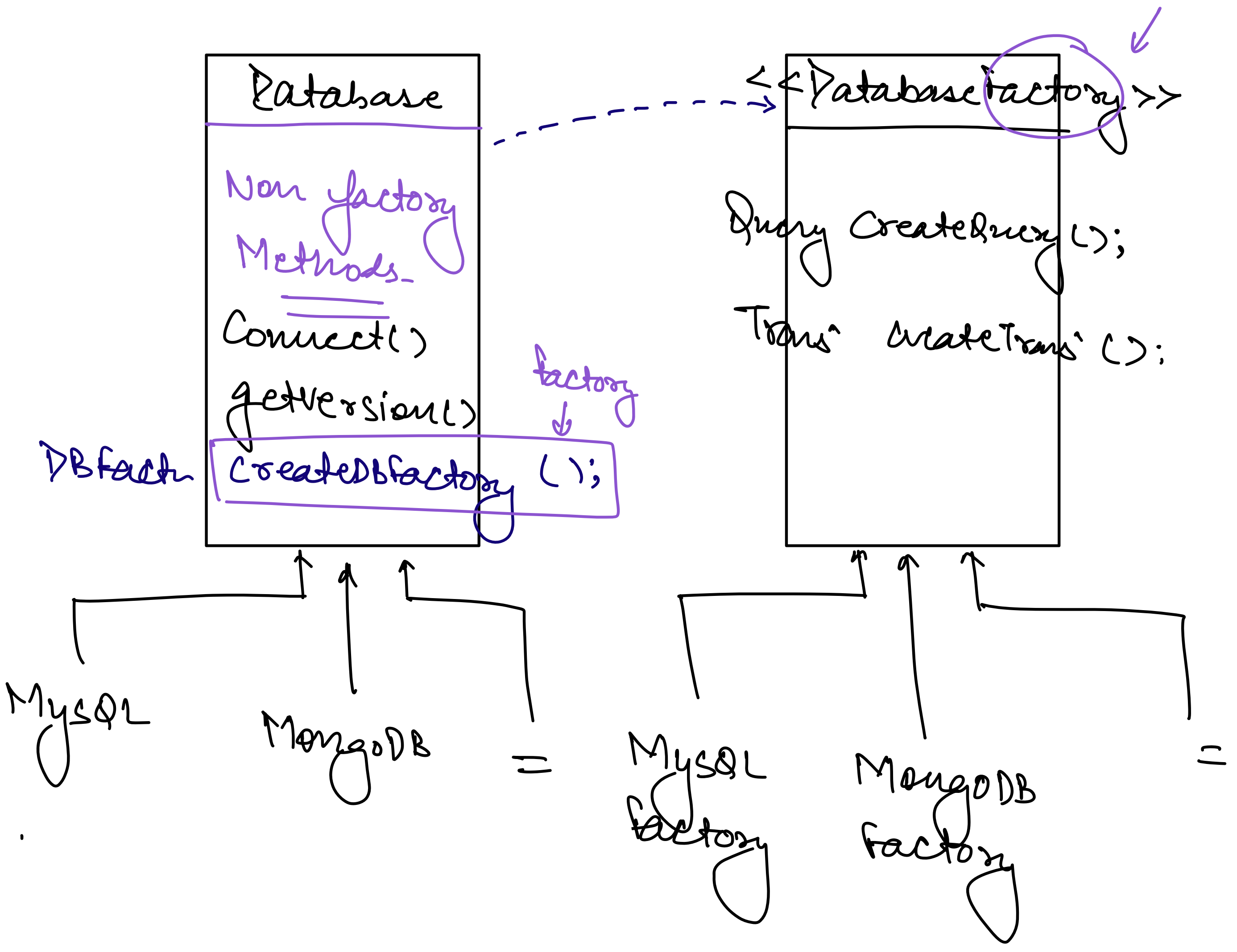
→ factory & Non factory methods.

⇒ SRP is being violated.





⇒ Abstract Factory.

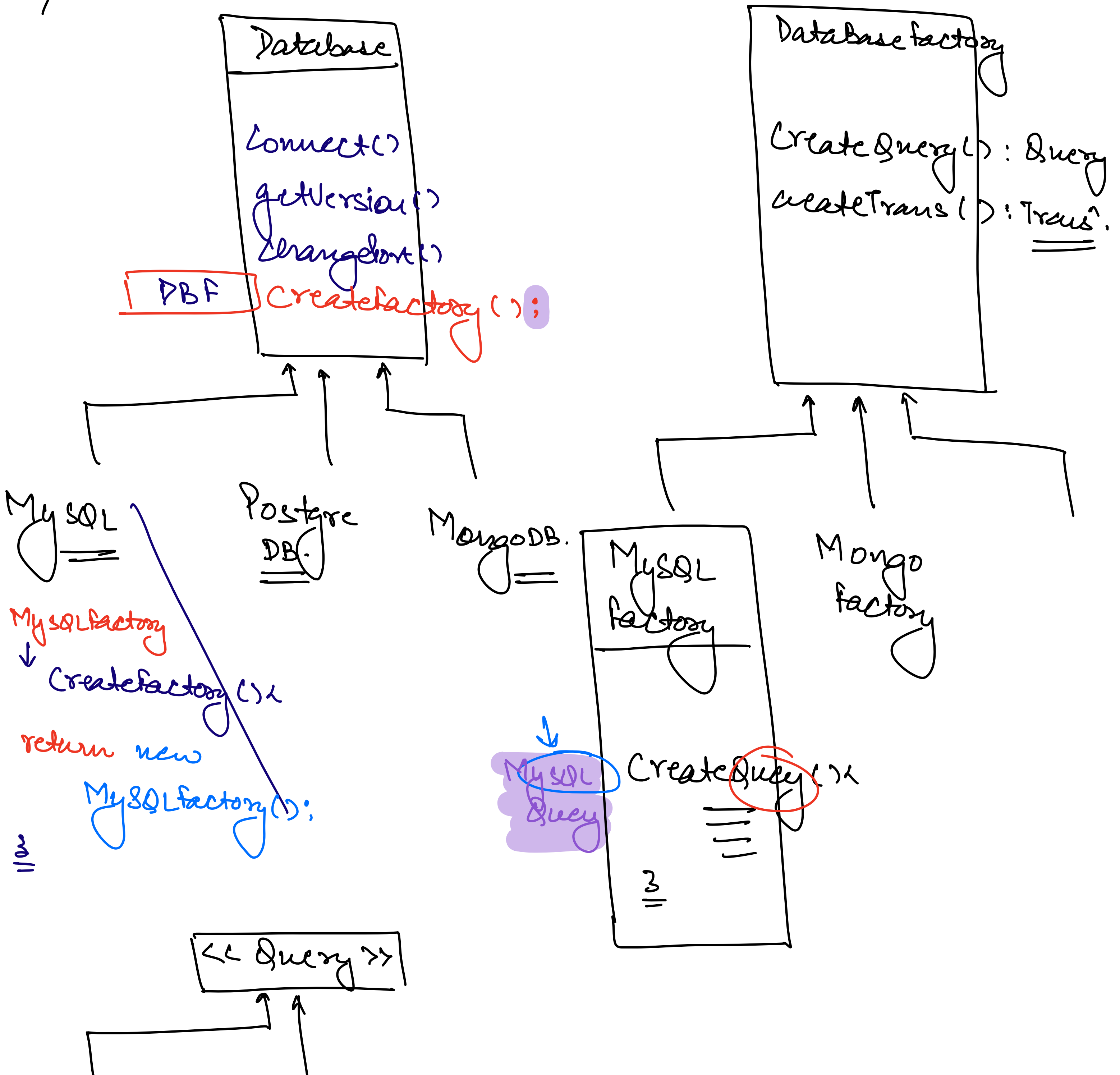


```
UserService {
        Database db; ← ——        MySQL
        Databasefactory dbf;  ← ——
    if (db is MySQL) {
            dbf = MySQLFactory();
    }
    else if (db is MongoDB) {
            dbf = MongoDBfactory();
    }
    ===
}
```

X

dbf = db. createDBFactory ();

⇒

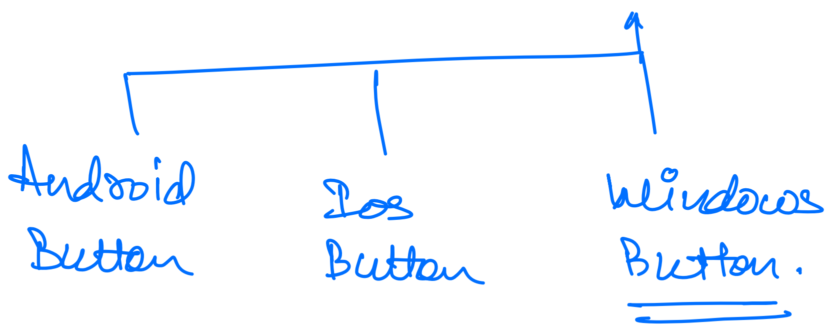


```
UserService {
    Database db = new MySQLDB();
    DatabaseFactory dbf = db.createFactory();

    CreateUser(——) {
        Query q = dbf.createQuery()
                MySQLQuery.

}
```

⇒ UI libraries.

Cross platform frameworks

CreateButton() ⇒ Button

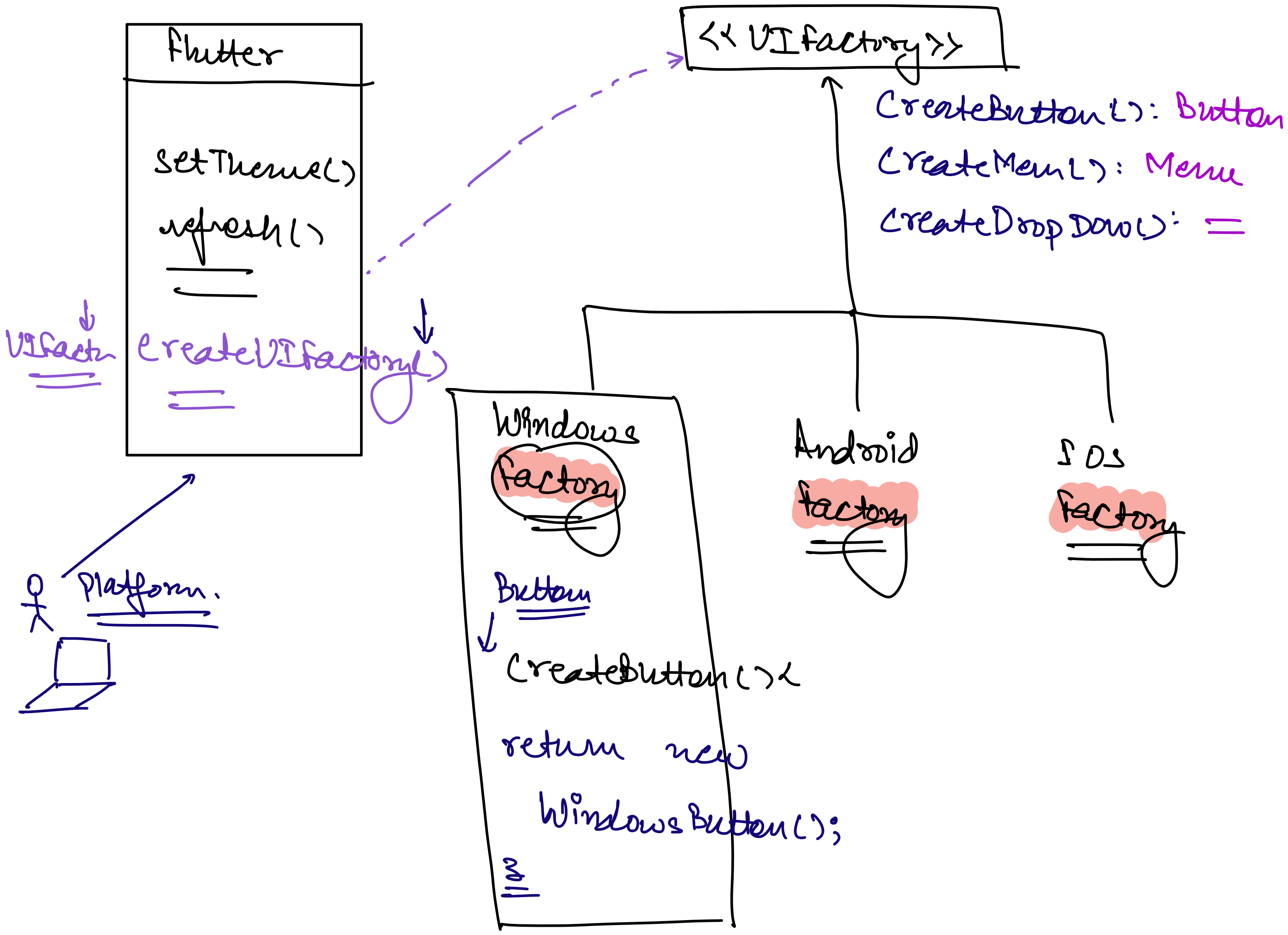


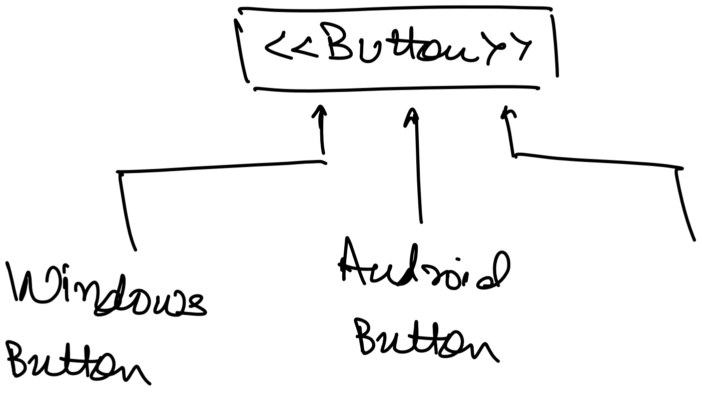
Flutter.

⇒ Flutter { ← SRP / OCP. ×

```
CreateButton() {
    if (platform == "Android") {
        return AndroidButton()
    }
    if (platform == "MacOs") {
        return MacosButton()
    }
    ______
    ______
    ______
    ______
}
CreateMenu() {
        ______
        ______   if - else
        ______
}
CreateDropDown() {
        ______
        ______
        ______
        ______
}
}
```

7


Flutter
setTheme()
refresh()
UIFactory
createUIFactory()
Platform
<< UIFactory >>
CreateButton(): Button
CreateMenu(): Menu
CreateDropDown():
Windows
Factory
Button
CreateButton(){
return new
WindowsButton();
}
Android
factory
iOS
factory


<<Button>>
Windows
Button
Android
Button

——— * ———

Factory : Anything that allows us to create the object of corresponding types.

Abstract Factory : When we have lot of factory methods, we move all the factory methods to a new interface.

Practical Factory

Move the logic of factory object creations to a new class.

——— * ———