Agenda

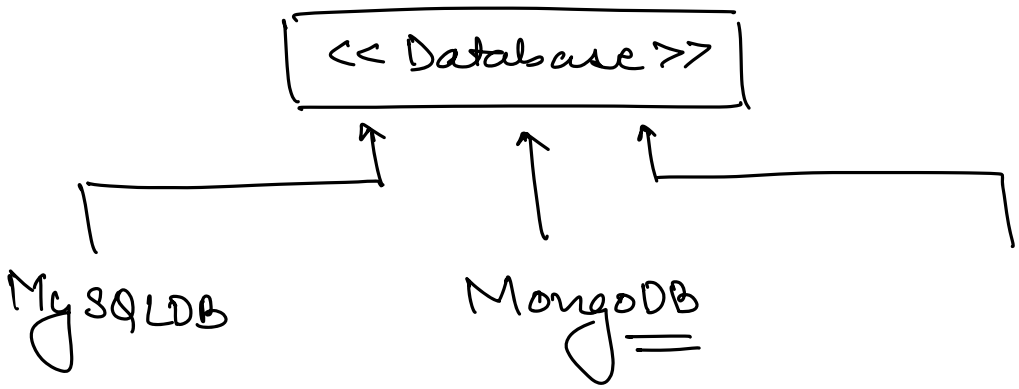
→ factory
→ Abstract factory
→ Practical factory.

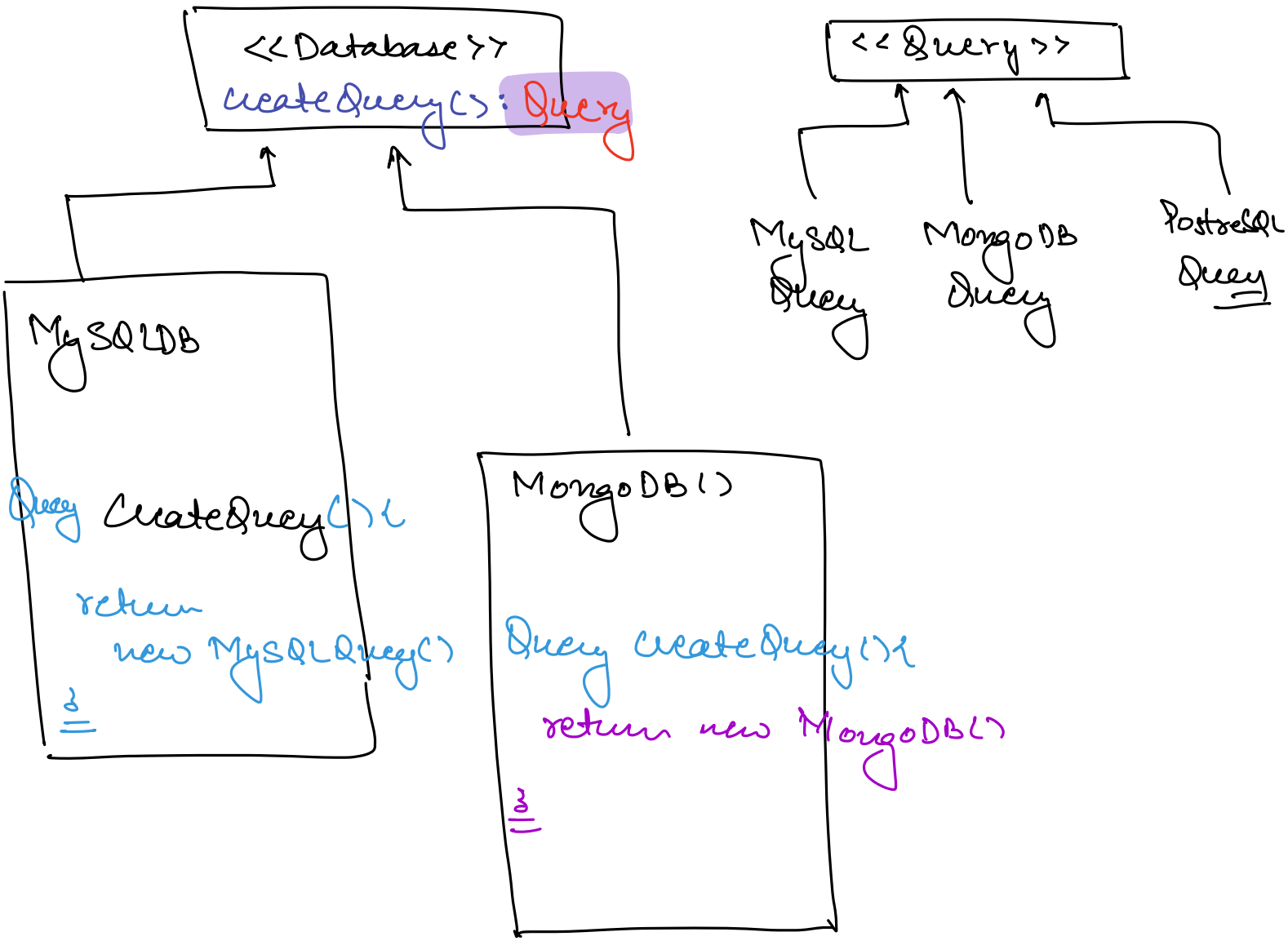
#

```
Class UserService {
    Database db = new MySQL() / new PostreDB();

    createUser(------) {
        Query q = createQuery(----);
        db.execute(q);
    }

    getUser(id) {
        Query q = createQuery(----);
        db.execute(q);
    }
}
```

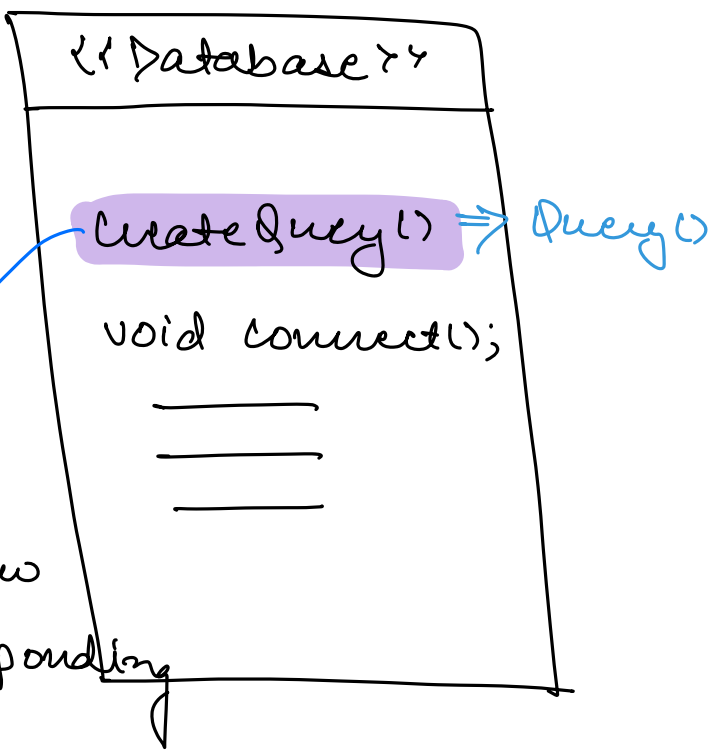
⇒ If DB was a class, Dependency Inversion principle would have violated as No 2 concrete classes should depend on each other directly

⇒ Database should either be an interface / Abstract class. So that our code is loosely coupled with DB.


<< Database >>
MySQLDB
MongoDB


<< Database >>
createQuery(): Query
<< Query >>
MySQL Query
Mongo DB Query
PostreSQL Query
MySQLDB
Query createQuery(){
return new MySQLQuery()
}
MongoDB()
Query createQuery(){
return new MongoDB()
}

```
Class UserService {
    Database db = new MongoDB();   // MySQLDB()

    createUser(- - - - - -) {
        Query q = db.createQuery(- - - );
        db.execute(q);
    }

    getUser(id) {
        Query q = createQuery(- - - );
        db.execute(q);
    }
    ——
    ——
    ——
}
```



Purpose of this method is to return the new object of corresponding query.

↳ Factory Method

#

```
UserService {
    Database db = ——
    Query q;

    fun() {
        if (db.instanceOf(MySQL))
            q = new MySQLQuery();
        }
        else if(db.instanceOf(MongoDB)) {
            q = new MongoDBQuery();
        }
        }
        ——
        ——
        ——
    }
```

SRP

OCP

<<Database>>

Query createQuery();

Void connect();

Void refresh();

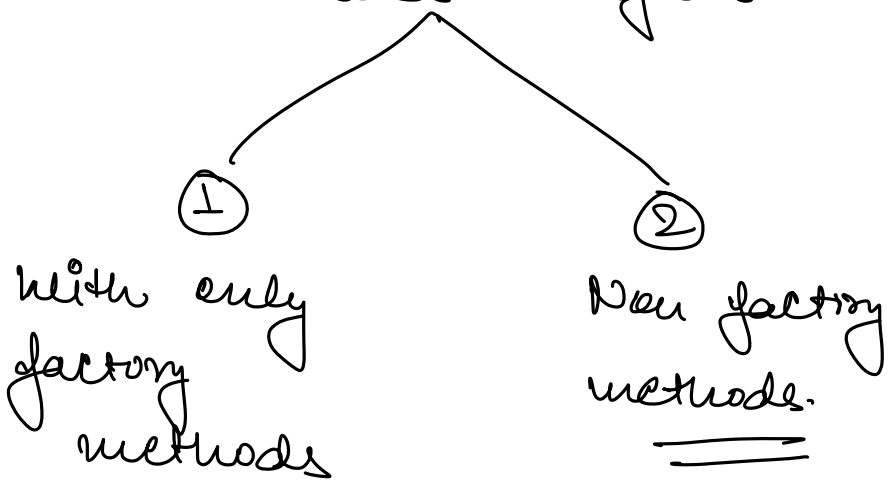
Transaction createTxn();

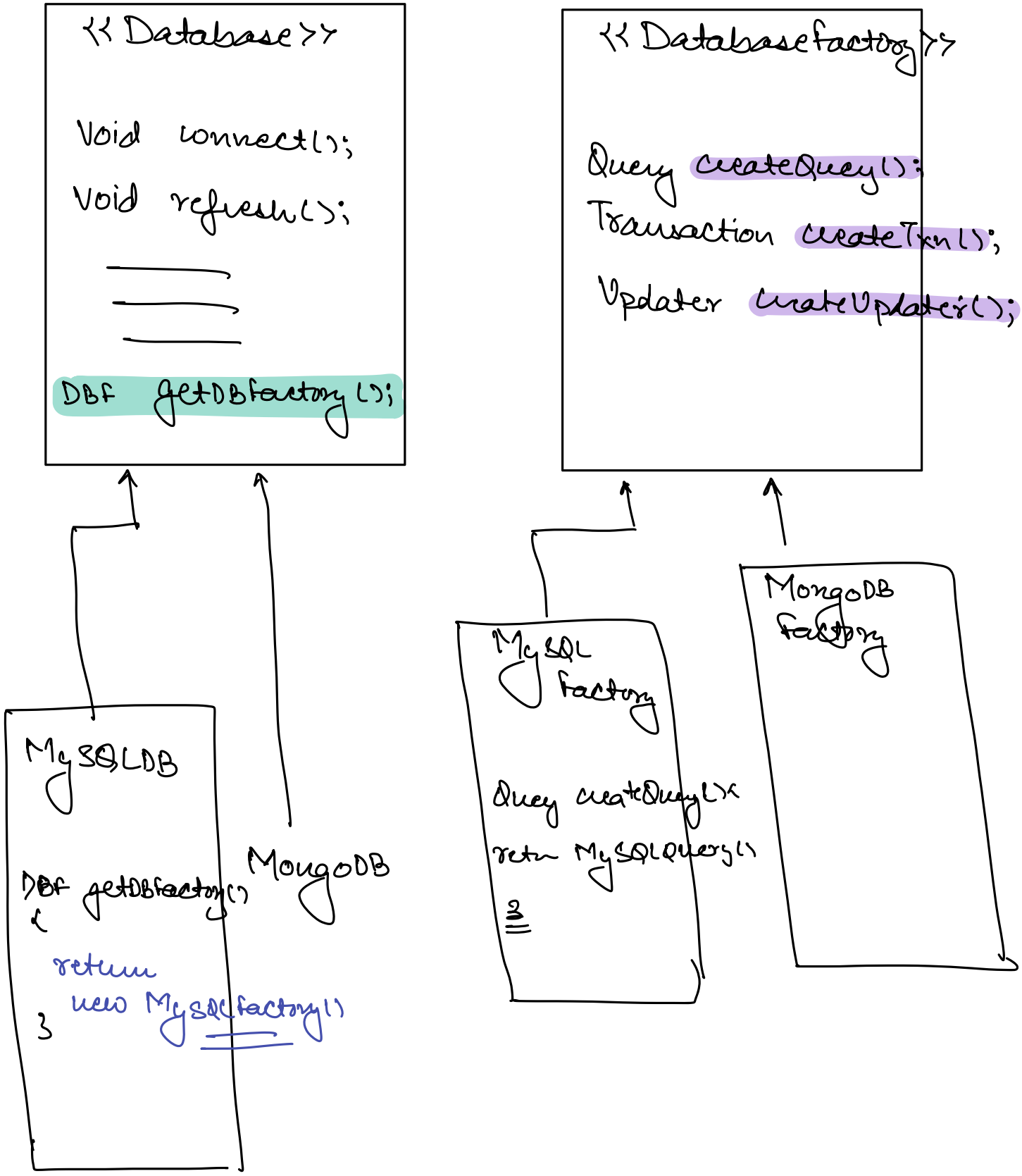
Updater createUpdater();

⇒ Lot of factory method.

⇒ SRP ✗

⇒ Divide the Database interface into 2 parts.



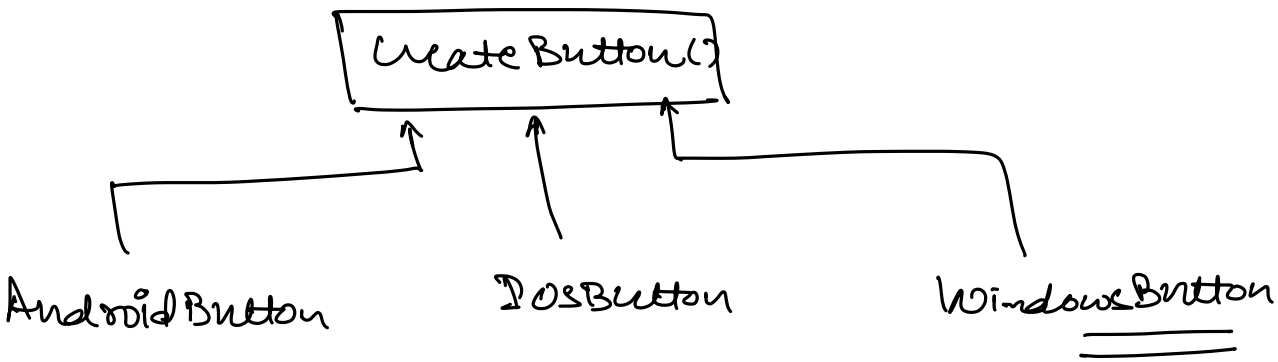

<<Database>>
Void connect();
Void refresh();
DBF getDBfactory();
<<Databasefactory>>
Query createQuery();
Transaction createTxn();
Updater createUpdater();
MySQLDB
DBF getDBfactory()
{
return
new MySQLfactory()
}
MongoDB
MySQL
factory
Query createQuery()
retn MySQLQuery()
}
MongoDB
factory

```
UserService {
    Database db = ____ ;        // MongoDB() / ~~MySQLDB()~~
    DatabaseFactory dbf = db.getDBfactory();

    CreateUser(----) {
        Query q = dbf.createQuery();
        execute()
    }
}
```

3

=> UI Libraries.

Flutter : Cross platform framework.



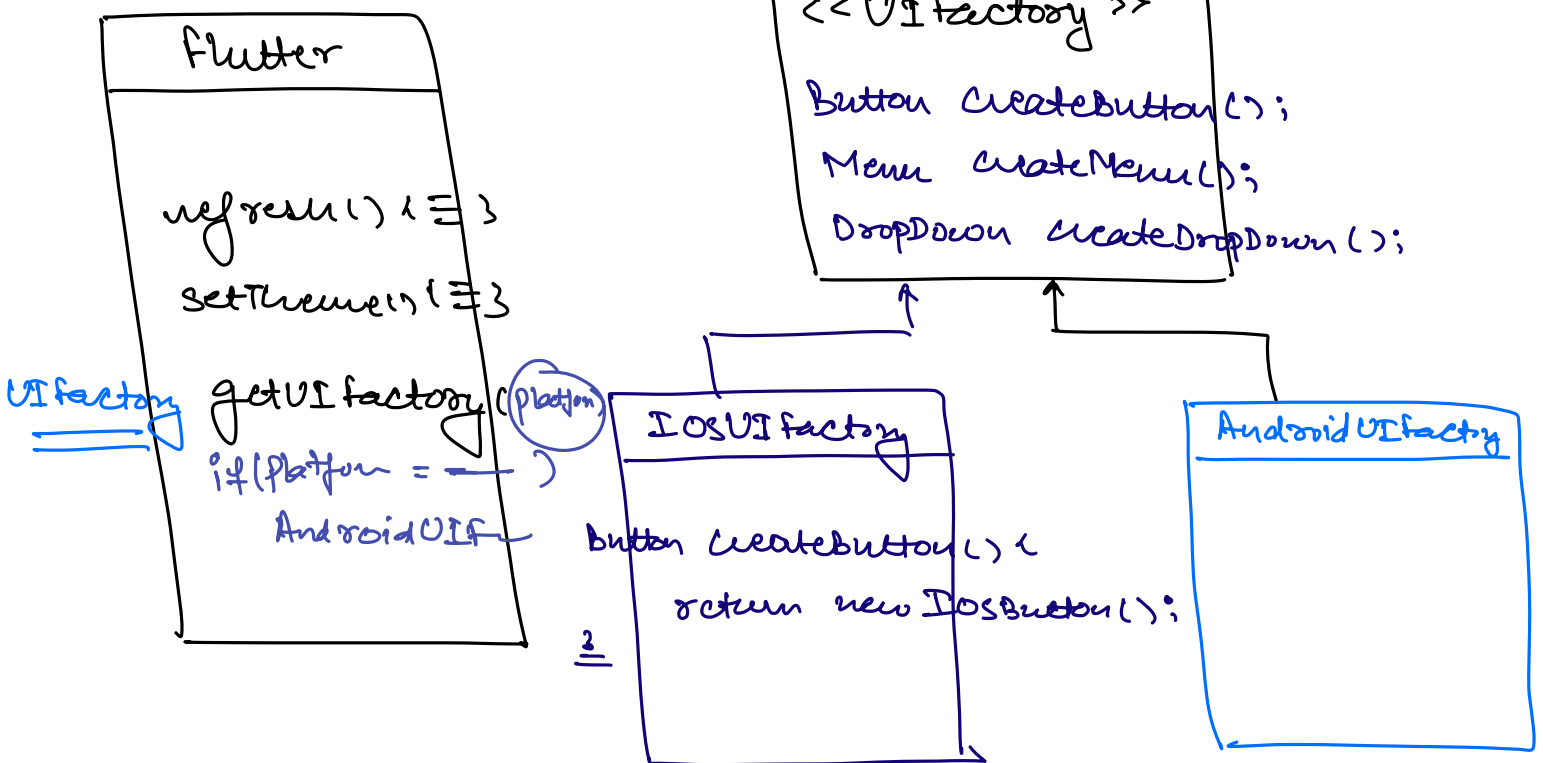
class Flutter {

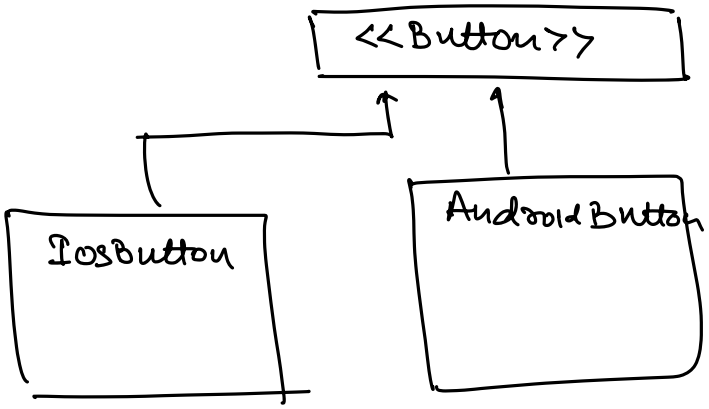
```
CreateButton(Platform){
    if (platform == "Android") {
        return new AndroidButton();
    }
    else if (platform == "IOS") {
        return new IOSButton();
    }
}
```

OCP X

SRP X

```
CreateDropDown(Platform){
    ...
}
}
```





Client {

Flutter flutter = ——————— ;

VIfactory uifactory = flutter.getVIfactory(IOS);

uifactory.createButton()

↓

IOSButton.

Summary

Factory : Method that helps to create the object of corresponding classes.

Abstractfactory

↳ lot of factory methods.

Practical factory

⇒ When we need to create an object of corresponding factory, it will lead to lot of if-else conditions in the primary class, we can move this logic to some other class. ⇒ Practical factory