Agenda

Start @ 9.05 PM

1) Factory Design Pattern

↳ Manufacture something
↳ Manufacture multiple things that are related

eg

```
UserService {
    Database db = ;
    createUser() {
        Query q = createQuery(—);
        db.execute(q);
    }
    updateUser() {

    }
    fetchUser() {

    }
}
```

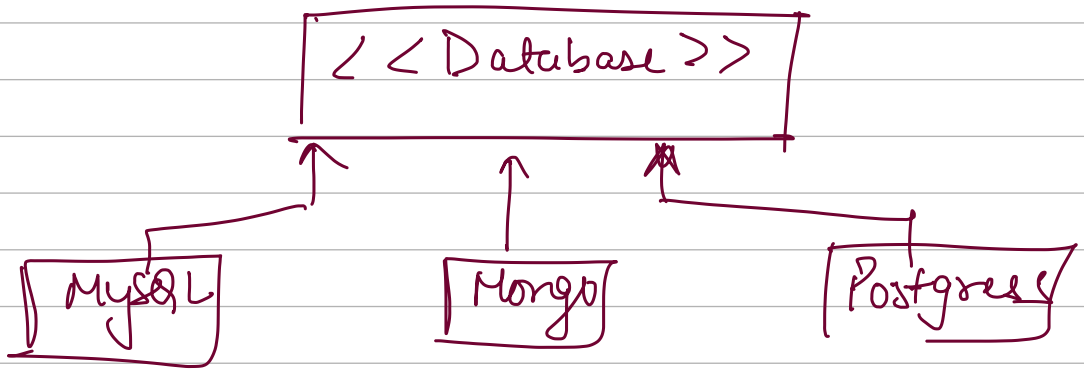
Class | Interface ✓ ✗ | Manages User

DI Violated

MySQL | Postgres | Mongo

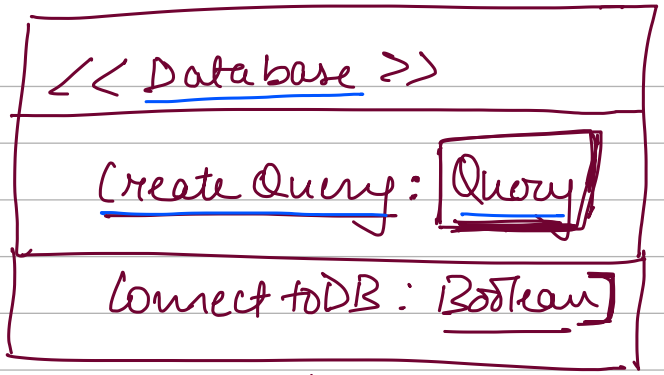
DB

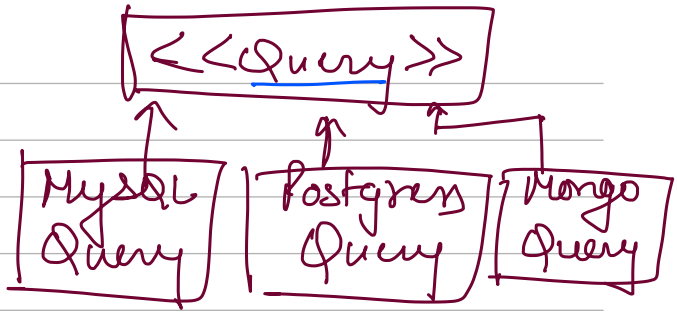
Insert—

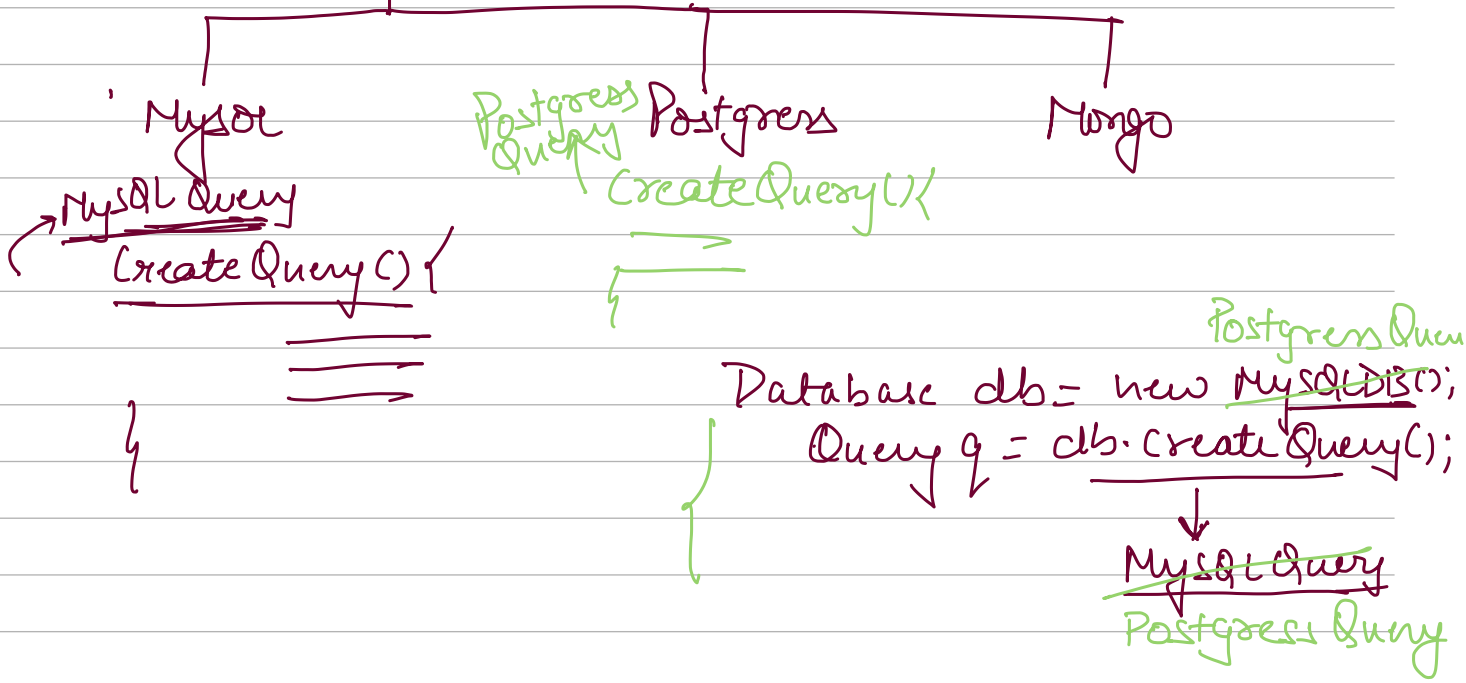


2) If database is a class then DI principle will be voilated

2) We should create DB as an interface and all different type of database (MySQL, Postgres – etc) will implement the interface







⇒ Overriden method can have return type same as parent or it can return any of the child return type.

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

Factory Method

eg Overriden

```
A {
    List<String> foo() {
    }
}

B extends A {
    List foo() {
        ArrayList
        LinkedList
    }
}
```

eg User Service {

```
Database Db = ——;
Query q = —
if(db instance MySQL)
    q = new MySQLQuery();
else if(db instance MongoDB)
    q = new MongodbQuery();
```

}

decision

OCP

+

<< Database >>

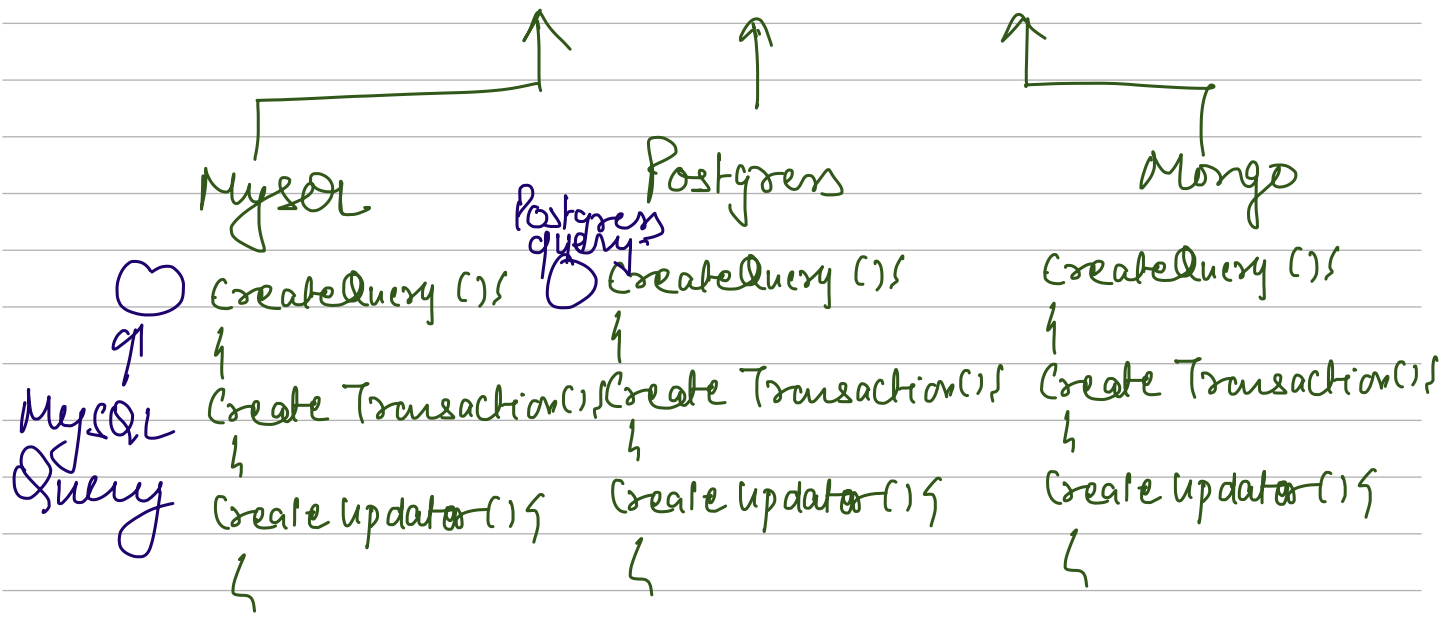
Attrs

Create Query : Query
Create Transaction : Transaction
Create Updater : Updater

} Factory Methods

ConnectToDB : boolean
refresh DB : boolean
Clear Cache : boolean

} Non factory Methods



Responsibilities of database interface

1) Containing Non factory Methods and attributes

2) Also containing all the factory methods to return the object of corresponding class

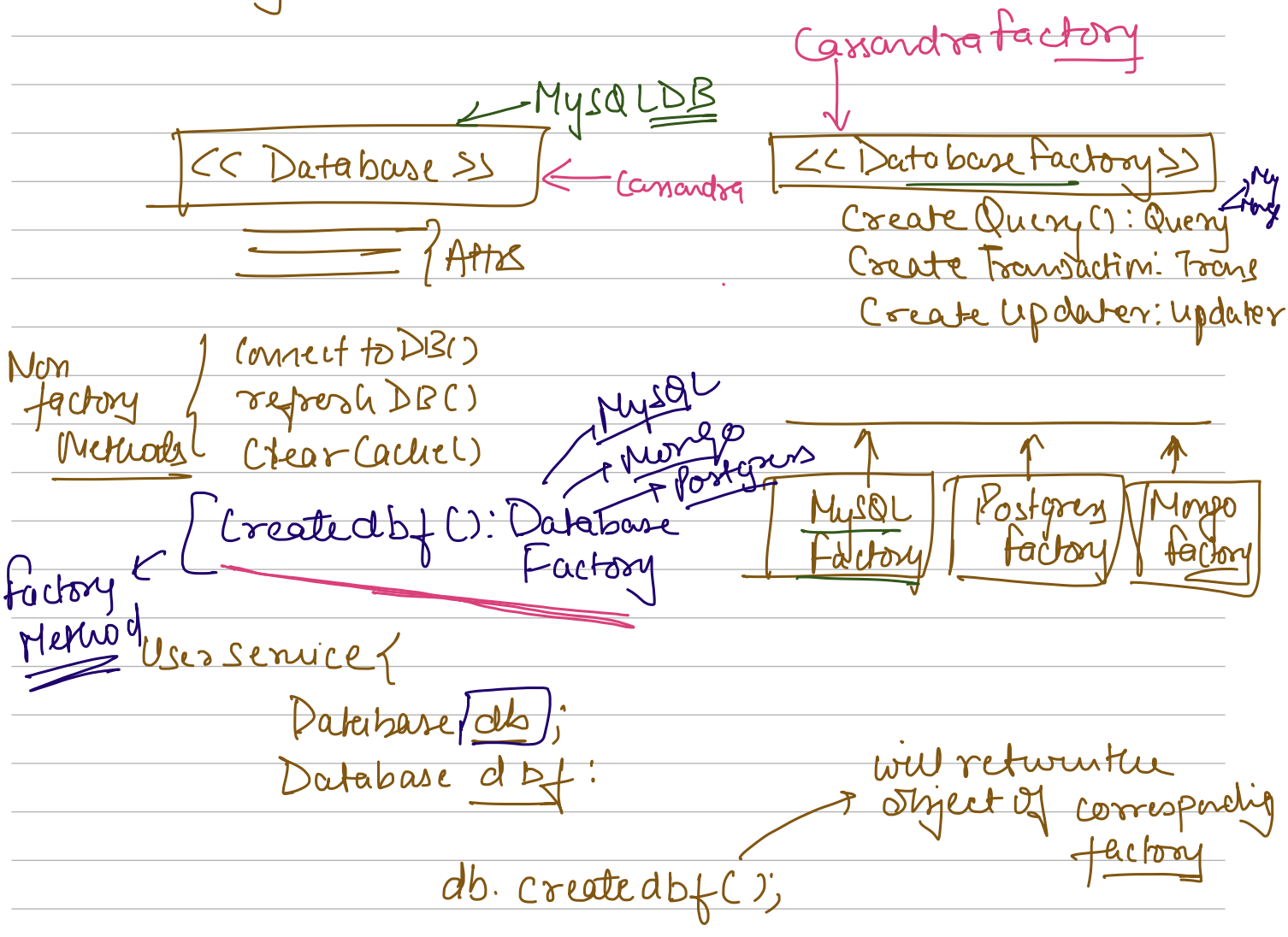
TOO MUCH

⇒ SRP is voilated

Abstract Factory

Split the interface into two parts :

1) Attributes & Non factory methods.
2) Factory Methods

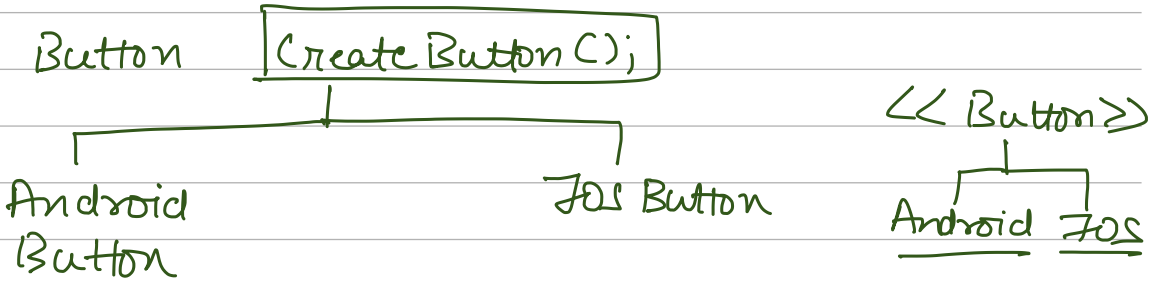


UserService

Postgres

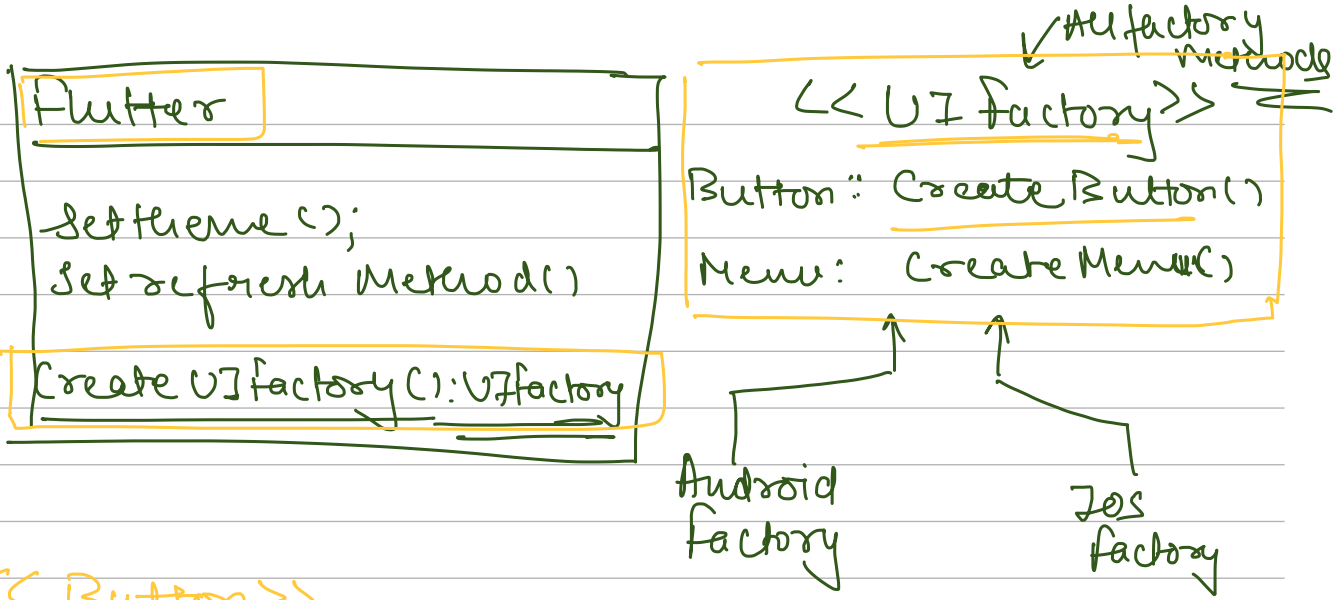
```
Database db = new MySQL();
Database factory dbf = db.createFactory();
Query q = dbf.createQuery();
```

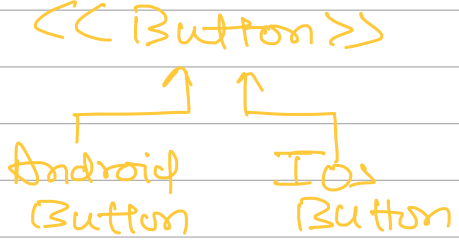
Flutter is frontend Application

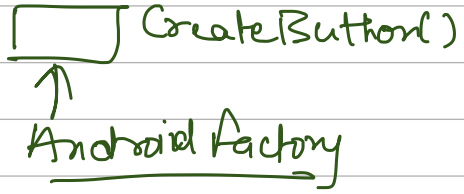



```
Flutter {

    Button CreateButton() {

        if ( platform is IOS)
        {
            return new IOS Button()
        }
        else if ( platform is Android ) {
            return new Android();
    }   }
```

OCP is voilated








```
Flutter f = new Flutter();
UIfactory uf = f.createUIfactory("Ios");
Button b = uf.createButton();
```

Client → Database

Practical factory

Whenever multiple variants of factories are there, it will violate OCP in our main class, so better to move the logic of creation of factory in a separate class