Factory Design Pattern

Agenda

- Factory Method Design Pattern
- Abstract factory Design Pattern
- Practical factory Design Pattern

Gang of four book

Common day to day life factory

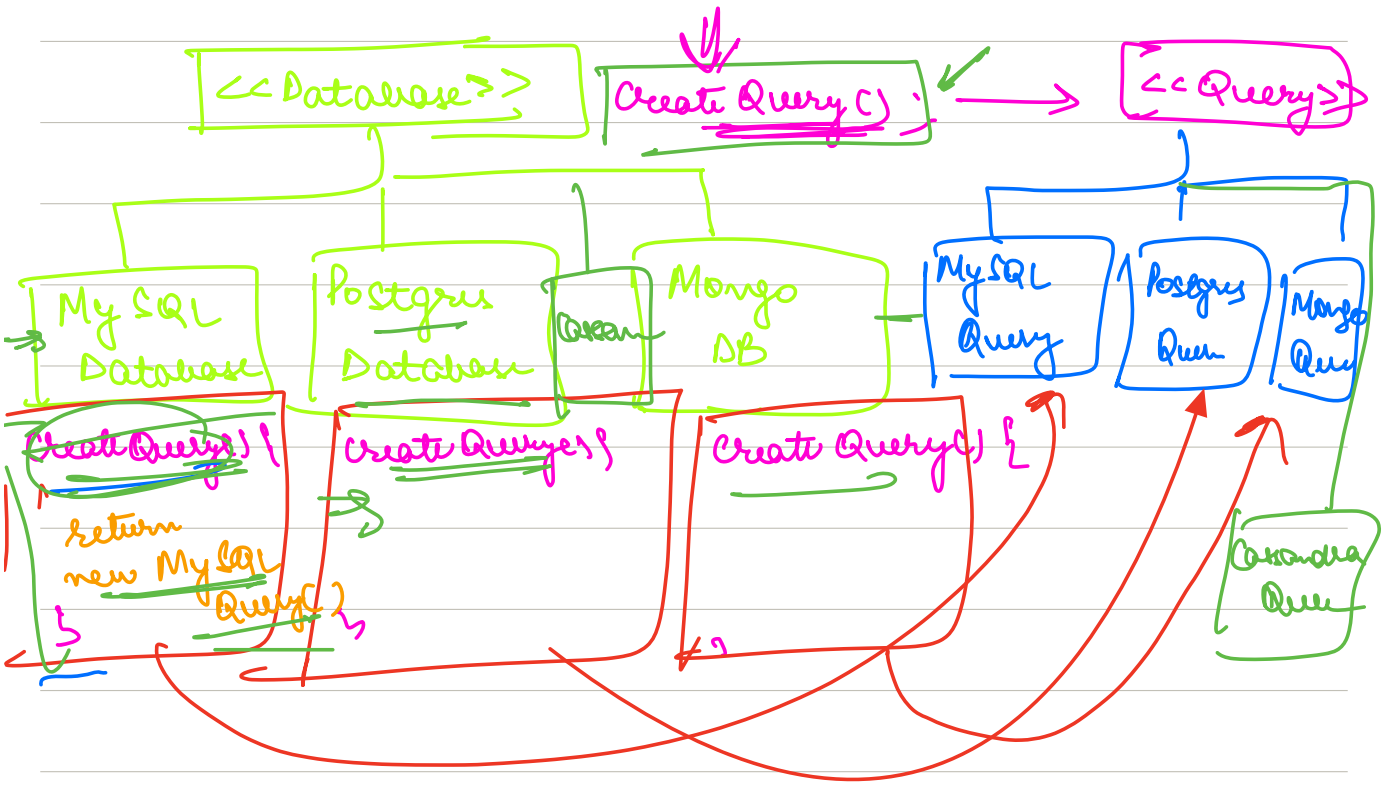
Factory Method Design Pattern

```
class UserService {
    Database db = ____

    void registerUser() {
        Query q = db.createQuery("Select ... from ____ ");
        q.execute();
    }

    void login() {
        // fetch user details from DB
    }
}
```

// store user in DB



⇒ Assume there was no createQuery() method in <<Database>> . But still we need to create a query object for corresponding type of DB. How will I do this

```
Class UserService {
    Database db;

    void registerUser() {
        Query q;
        if (db.type == MySQL)
            q = new MySQLQuery
        else if (db.type == Postgre)
            ;
    }
}
```

→ violates SRP and OCP

Purpose of createQuery()

Based upon type of database create an object of corresponding Query class

Factory ⇒ Create a new object of a corresponding class based on something.

Factory Method

Often we have use cases where from one object we might need to get other associated objects.

A method in the class that does nothing but return an object of associated corresponding class ⇒ Factory Method

→ Prevent you from storing mapping b/w classes in 3rd class

Problem Statement

→ From one class, I need to get corresponding object of another class

Problem Statement

Everyone has a best friend.

Based on who the person is you want to know their best friend

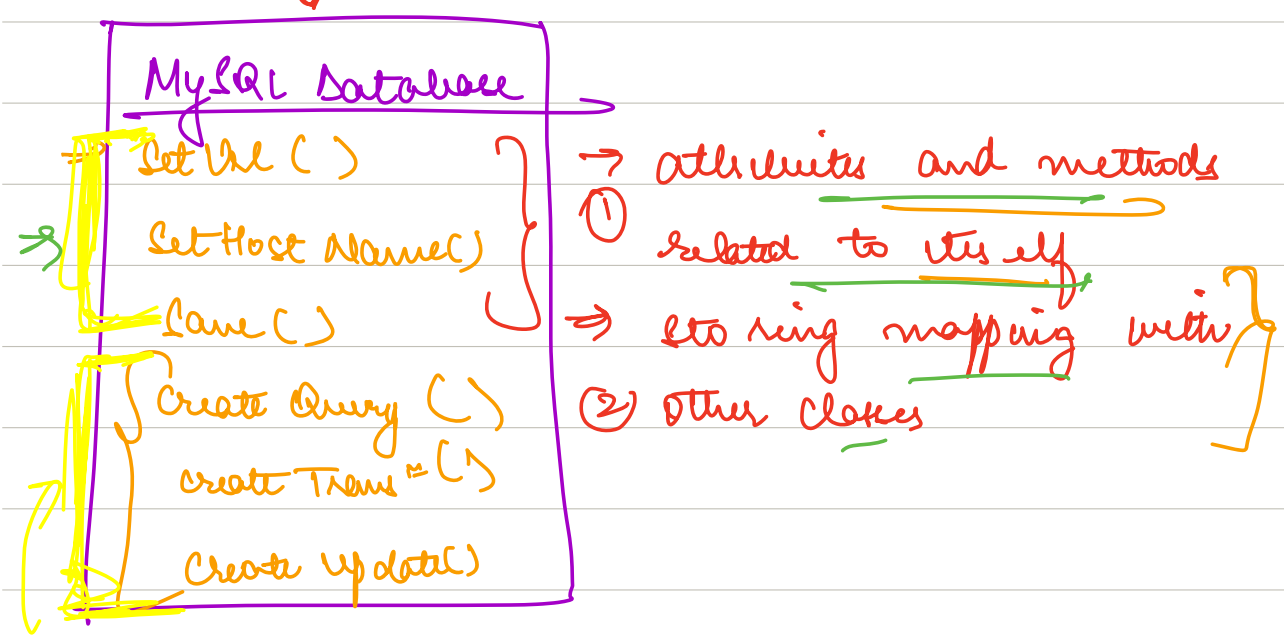
→ Maintain a register

Person	Best Friend

=

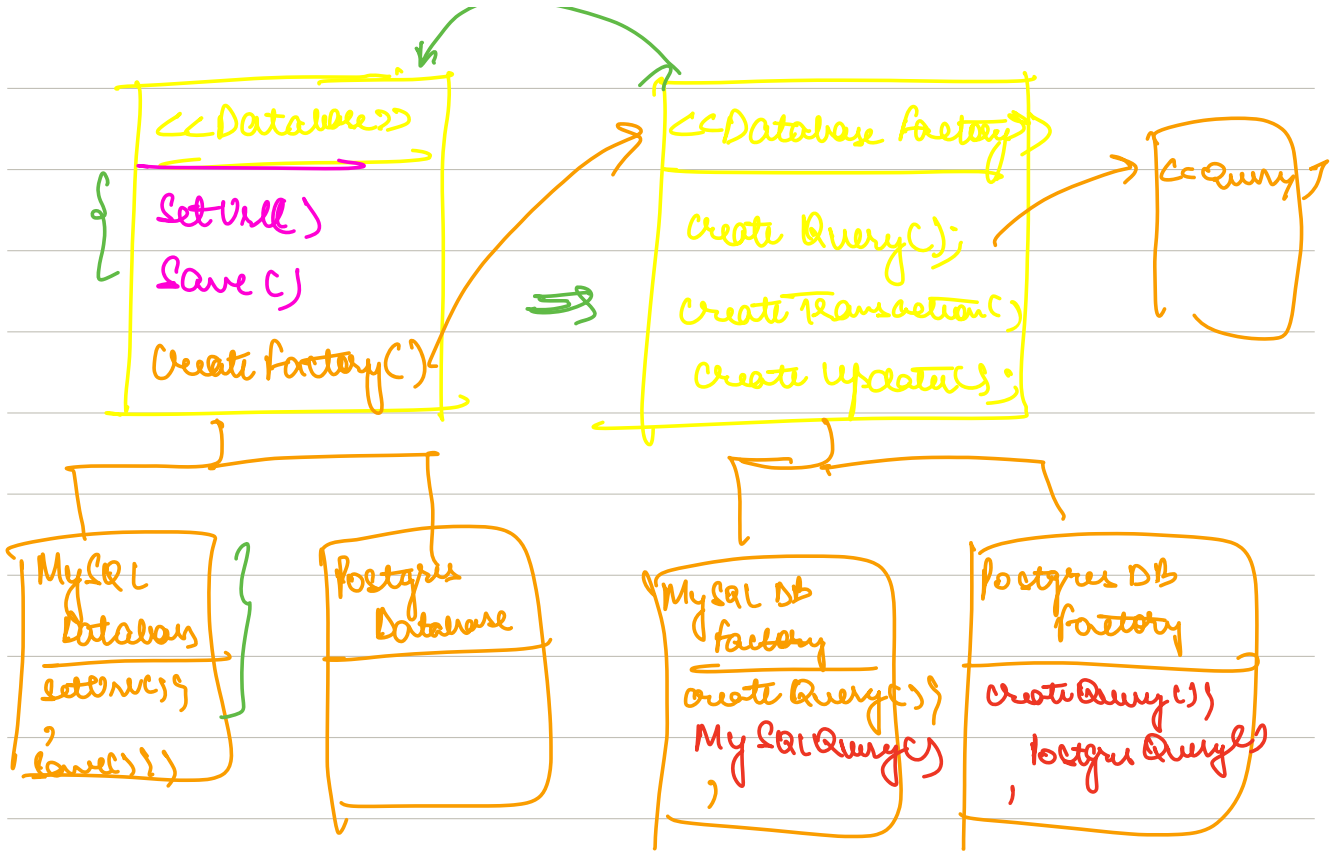
→ Ask the Person

Database → Query ⇒ createQuery()
→ Transaction ⇒ CreateTransaction()
→ Update ⇒ createUpdate()



Abstract Factory

When you have a class that starts having a lot of factory methods, SRP principle starts getting violated ⇒ Put all factory methods in a separate class.



Class UserService {

→ Database db;

registerUser() {

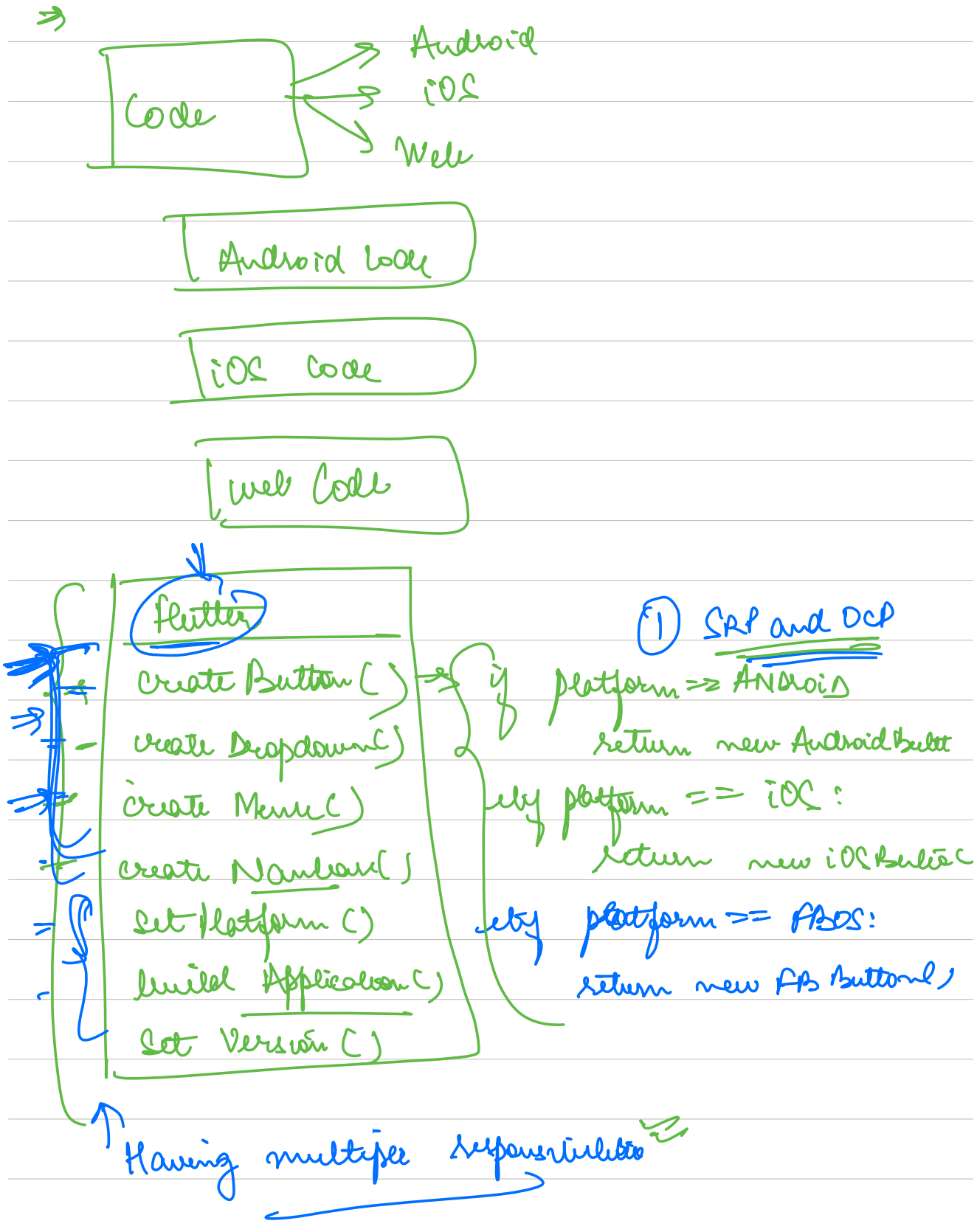
Query q = db.createFactory()
.createQuery()

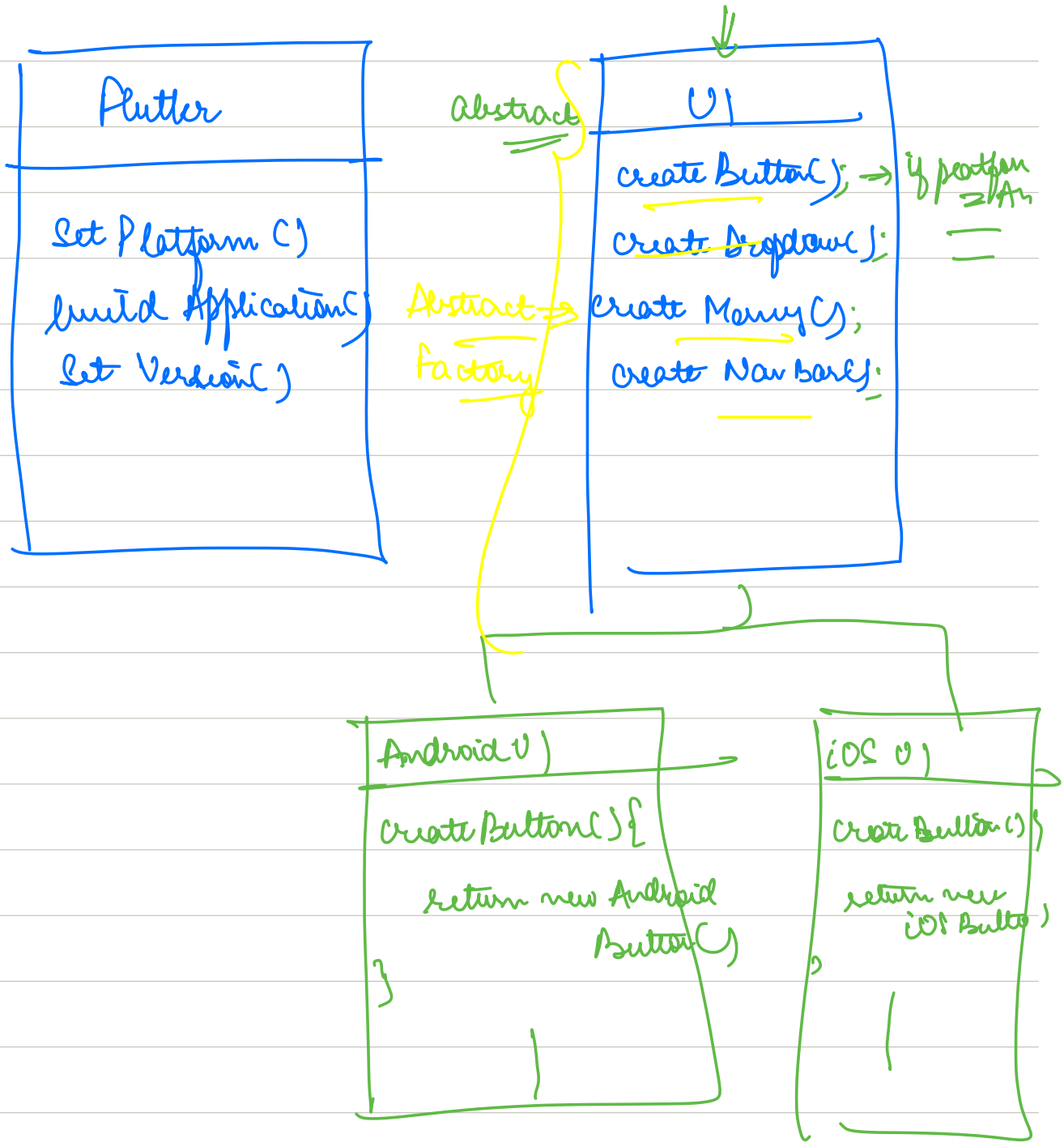
;

.

}

}


Code
Android
iOS
Web
Android Code
iOS Code
Web Code
Flutter
createButton()
createDropdown()
createMenu()
createNavbar()
setPlatform()
buildApplication()
setVersion()
(1) SRP and OCP
if platform => ANDROID
return new AndroidButton
else if platform == iOS:
return new iOSButton
else if platform == FBOS:
return new FBButton()
Having multiple responsibilities


Flutter
Set Platform()
build Application()
Set Version()
abstract
UI
createButton();
createDropdown();
createMenu();
createNavBar();
if platform = An
Abstract
Factory
Android UI
createButton(){
return new Android
Button()
}
iOS UI
createButton()
return new
iOS Butto

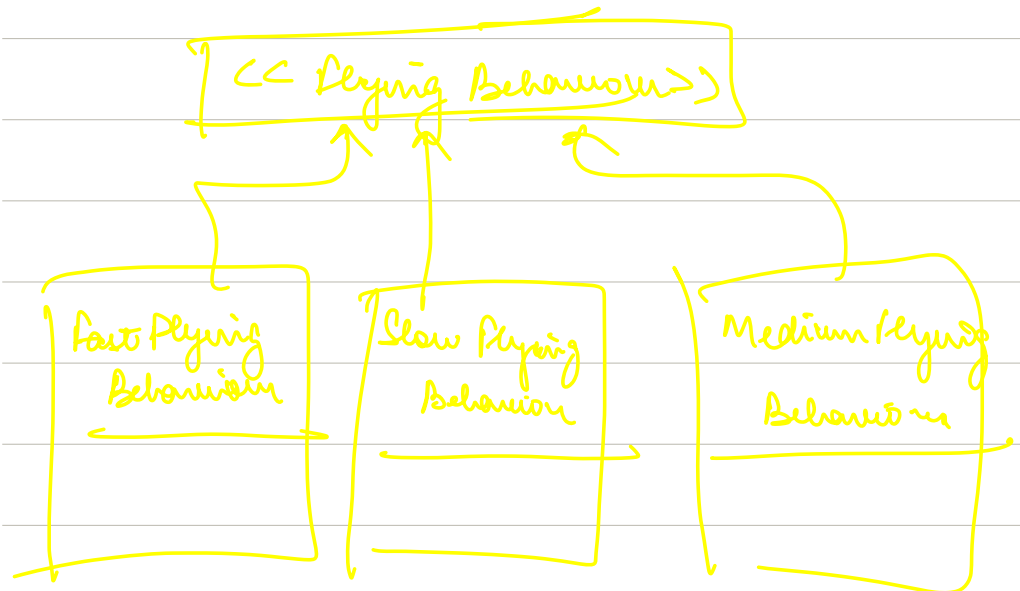
Summary :

Factory Method : From (1) class I need an object of (2) class. The object of (2) class depends on (1) class ie, a method that returns this ⇒ Factory Method

Abstract factory : Moving all factory methods to an independent class

Database ⇒ Query / Trans^{n} / Updates

Practical Factory



Client {

String inp = input();

FlyingBehaviour fb;

OCP

if inp == ____

fb = new FastFlyingBe

} else if inp == ____

fb = new MediumFlyin

if type == ____

fb = ____

⇒ We need to create an object of correct type of flying behaviour based on a given parameter

⇒ DON'T PUT THIS CODE IN CLIENT

⇒ OCP

⇒ Code Duplication

Class Flying Behaviour factory {

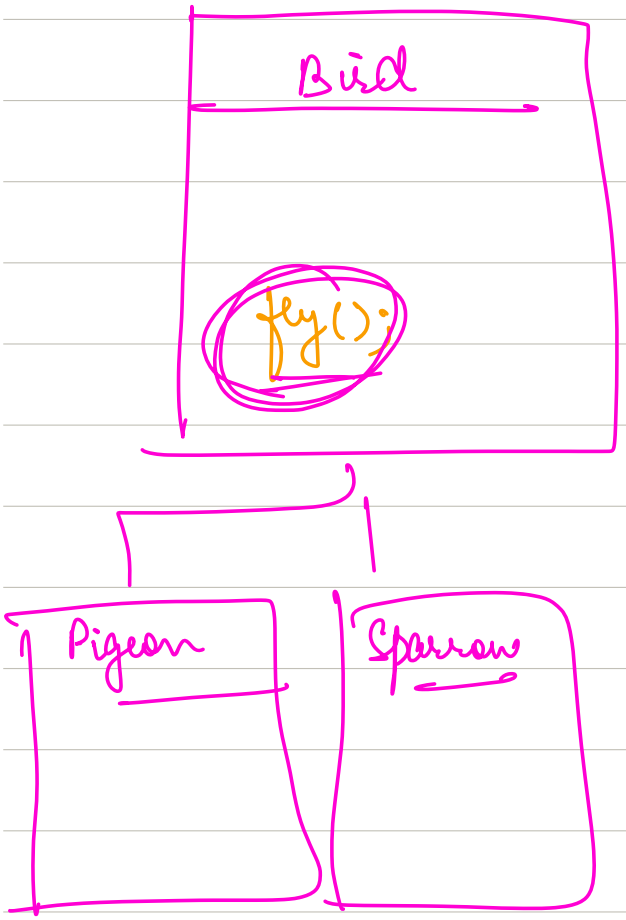
⇒ will have methods that give obj of Flying Behaviour based on some param

Flying Behaviour getFlyingBehaviour fromUserInput(String input)

```
if (input ==) {
} else if (input ==) {
} else {
}
```

⇒ Here if else are part of business logic ⇒ okay

}



PS = enum called BirdType which you will get in a HTTP req.

You have to create that Bird and fly that Bird

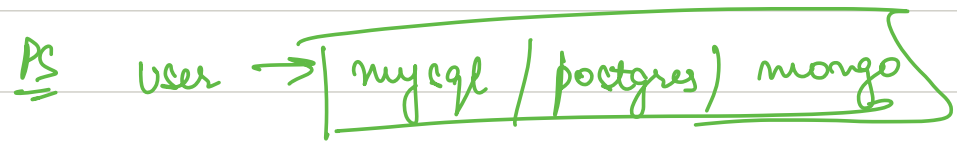
```
class BirdService {
⇒   flyBird(BirdType type) {

    }
}
```

```
class BirdFactory {
    Bird getBirdFromEnum(BirdType type){
        if type == pigeon:
            return new Pigeon();
        |
        |
    }

    Bird getBirdFromString(String bird){
            |
            |
    }
}
```

PRACTICAL FACTORY : Create an object of correct subtype of a class / interface based on a param.




```
class DBFactory {
    DB getDBByName(String name) {
        if (name == mySql)
            return new MySQLDBC()
        else if ( — — —
            — — —
    }
}
```

```
Client {
    psvm() {
        String inp = getInput()
        DB db = DBFactory.getDBFromName
                (inp);
    }
}
```

You have an object of a class.
Based on that obj you need to get
an object of another class.

BAD WAY : if - else in client

GOOD WAY : Create factory Methods