⇒ Registry Design Pattern :-

Student

- id
- name
- phone

3 unique

- batchName
- moduleName
- instructorName
- StartYear
- endYear
- batchPsp
- level

7 common

10

↓

Prototype

- id	1	2	3
- name	Divya	Ishan	Paras
- phone	1234	5678	9101
- batchName	Sept23		
- moduleName	DSA		
- instructorName	Mohit		
- StartYear	2023		
- endYear	2024		
- batchPsp	0		
- level	Intermediate		

Instead of keeping a single prototype object that can be cloned to make multiple objects,

we create a registry of multiple prototype objects against some keys.

Prototype

- id → 1
- name → Divya
- phone → 1234
- batchName → Sept23
- moduleName → DSA
- instructorName → Mohit
- startYear → 2023
- endYear → 2024
- batchPsp → 0
- level → Intermediate

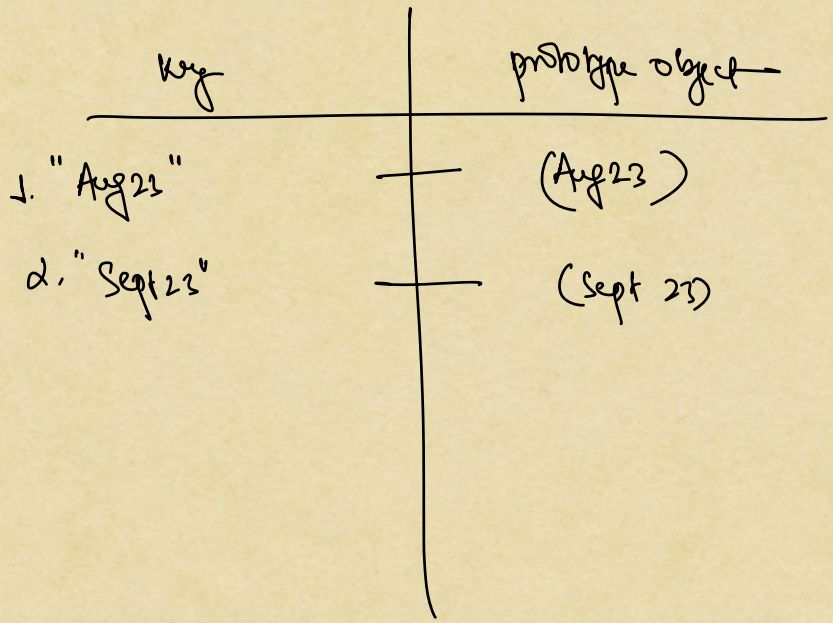
Sept23 batch

Prototype

- id →
- name →
- phone →
- batchName → Aug23
- moduleName → Language fundamentals
- instructorName → Rohit
- startYear → 2023
- endYear → 2023
- batchPsp → 90
- level → Beginner

Aug23 batch

→ we can create multiple prototype objects and store them in a map against some key.

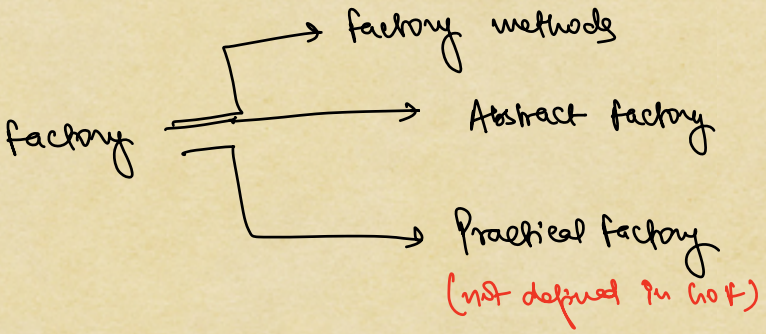


registry.get(batch).clone()

⇒ Factory Design Pattern

↓

Creational



Factory Method :-

```
UserService {

    Database db = new Database("_____");

    saveUser(){

        Query q = db.createQuery("_____");

        q.execute();

    }

}
```

factory method as it is creating Query object

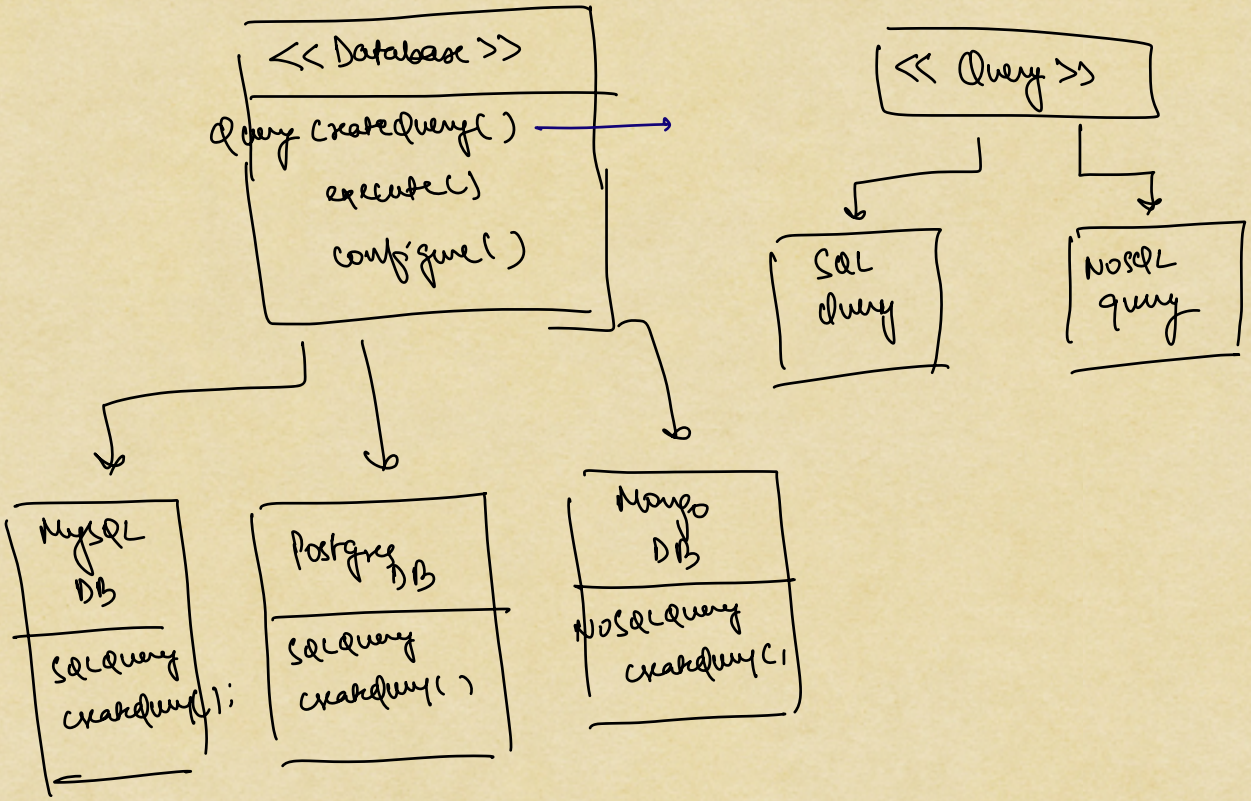
If Database is a concrete class then we are violating DIP as two concrete classes should never depend on each other directly. Instead, they should depend on each other by interface / abstract class

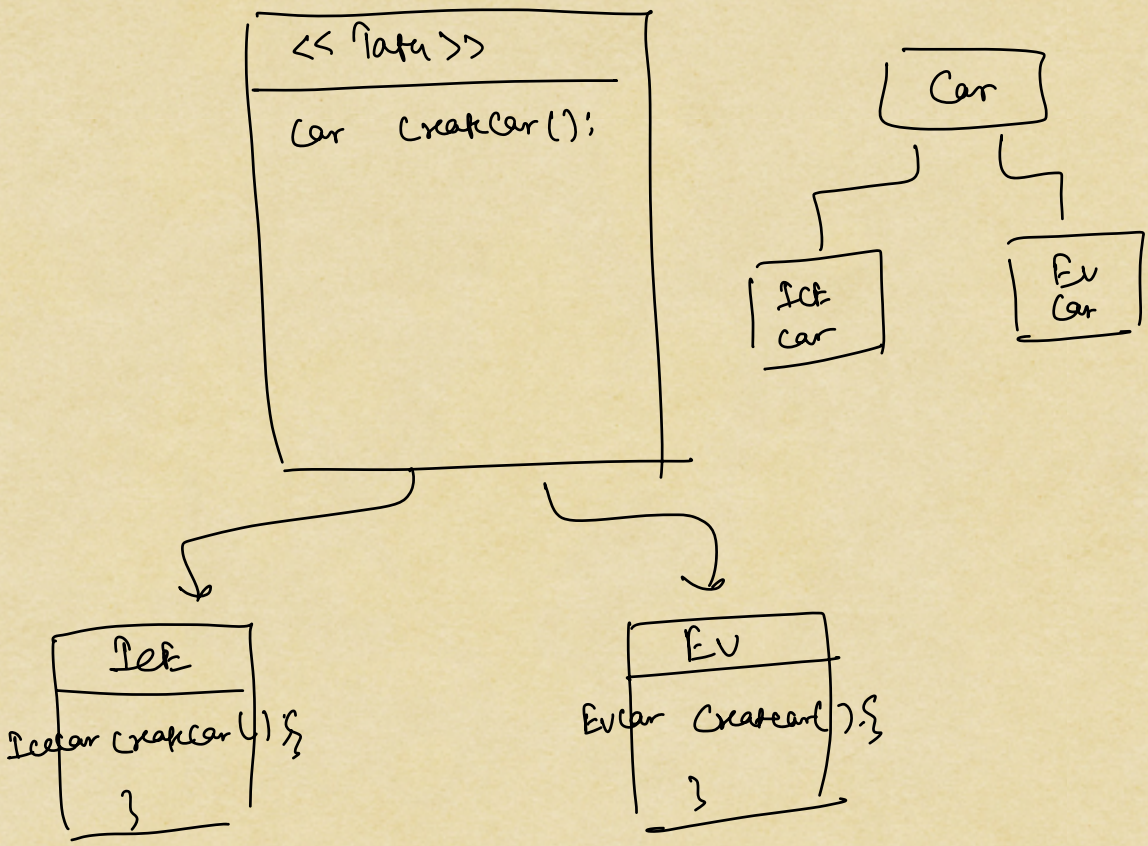
↓

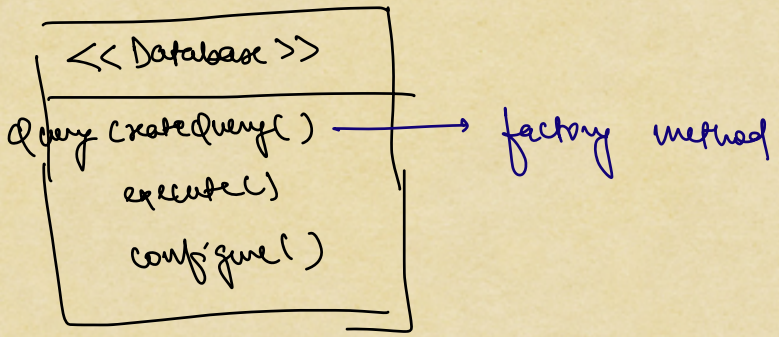
Lets make Database an interface

⇓

<<Database>>

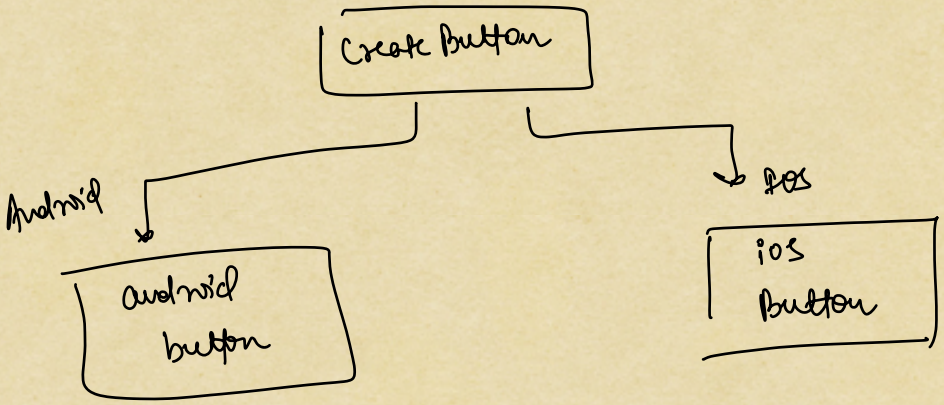

<< Database >>
Query createQuery()
execute()
configure()
<< Query >>
SQL query
NoSQL query
MySQL DB
SQLQuery createQuery();
Postgres DB
SQLQuery createQuery()
Mongo DB
NoSQLQuery createQuery()


<< Data >>
Car createCar();
Car
Ice car
Ev Car
Ice
Icecar createCar(){}
Ev
Evcar createCar(){}



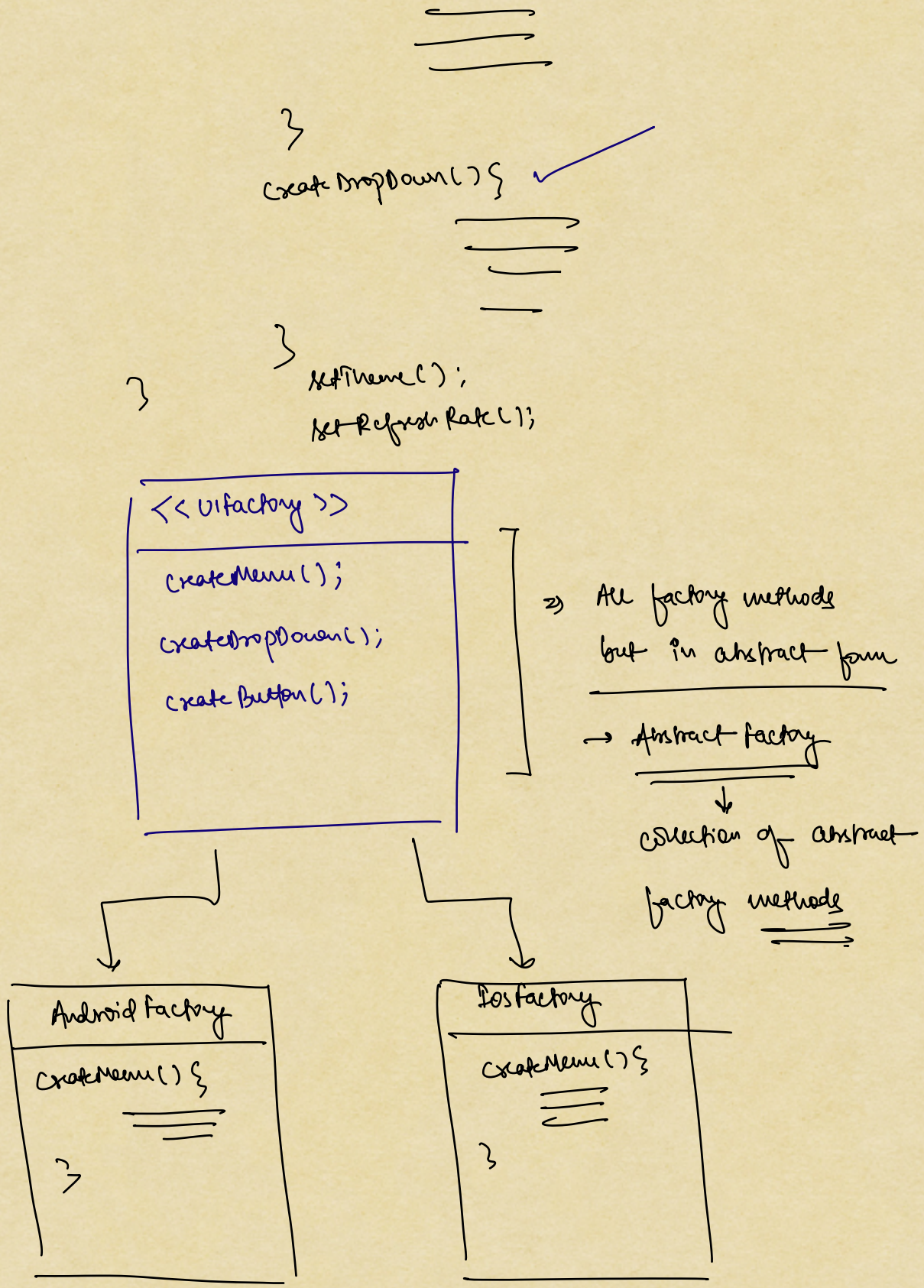
Hybrid app frameworks :

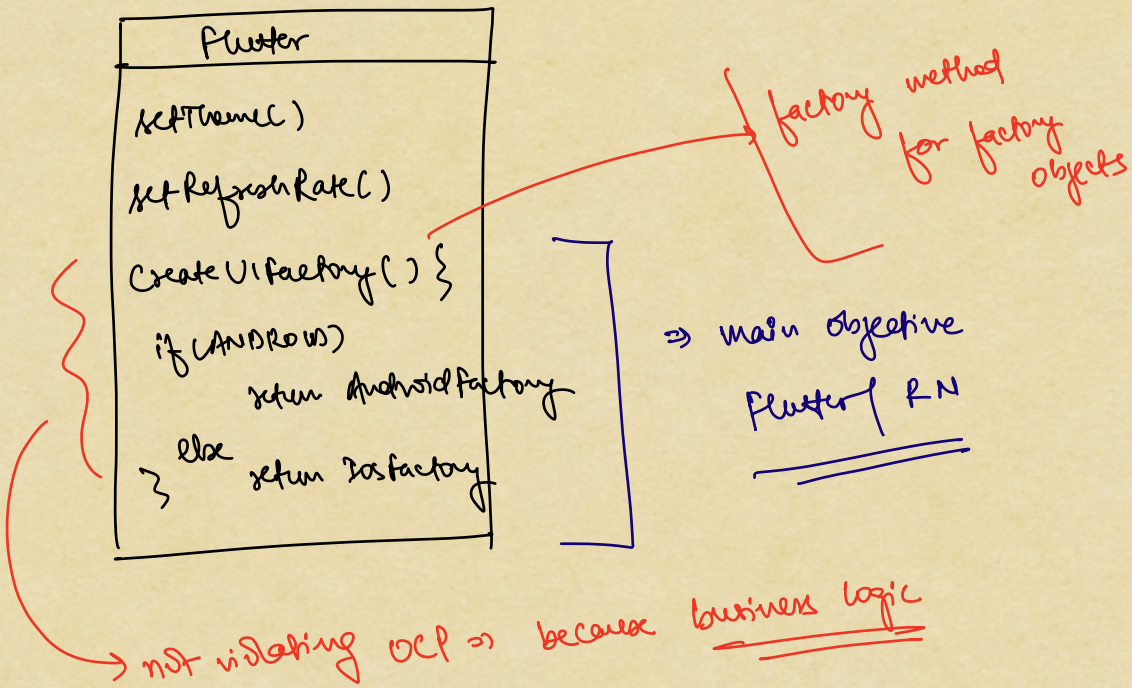
→ frameworks that generate apps in both IOS & Android
eg ⇒ React Native, flutter




```
flutter {
    CreateButton() {
        if( config == ANDROID )
            return new AndroidButton();
        else
            return new iOSButton();
    }
    createMenu() {
```

OCP + SRP violation


createDropDown() {
setTheme();
setRefreshRate();
<< UIFactory >>
createMenu();
createDropDown();
createButton();
=> All factory methods but in abstract form
→ Abstract factory
↓
collection of abstract factory methods
Android factory
createMenu() {
IosFactory
createMenu() {



⇒ Practical factory

* Try to maintain all factory method as abstract and group them together under single interface
* Create a separate class to implement factory
of factory ⇒ OCP violation scope should be minimised