Day 10: Design Pattern: Factory

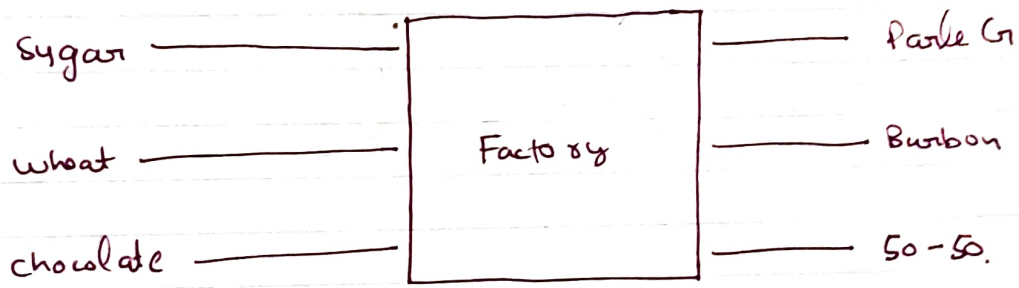
(1) Motivation
- Real world
- Technical

(2) simple Factory
- Implementation
- code.

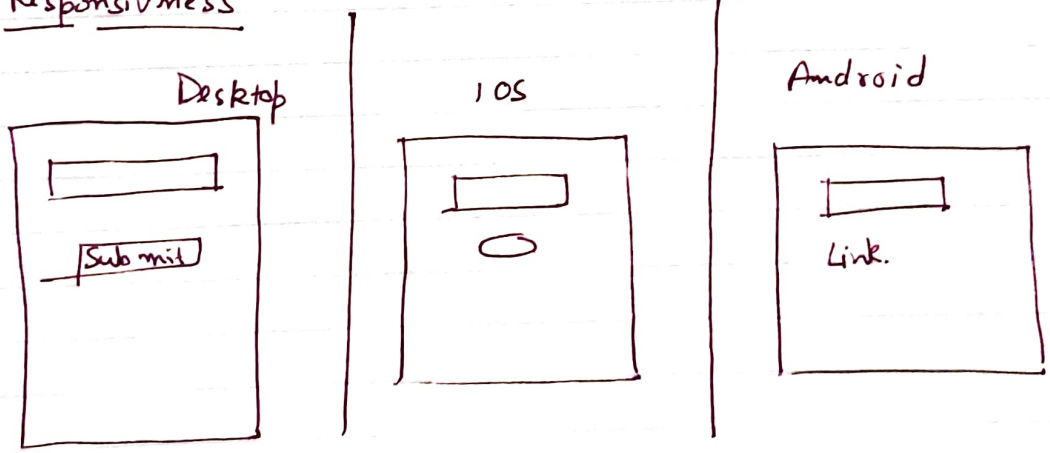
(3) Factory Method.

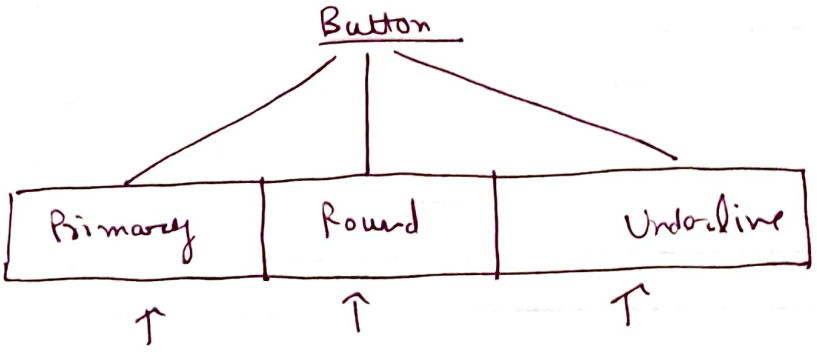
(4) Abstract Factory

Motivation



Responsiveness

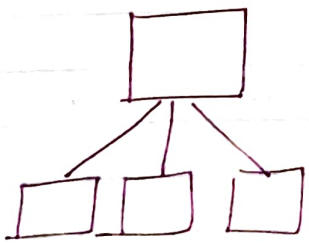




```
render () {
    if (platform == iOS) {
        return ROUND
    }
    else if (platform == ANDORID) {
        LINK
    }
    else {
        PRIMARY BUTTON
    }
}
```

1. SRP violations — }
2. OCP }
3. complex logic around creation
4. Sub class usage ☆

→ Backward incompatible.
→ Not aware about subclasses.

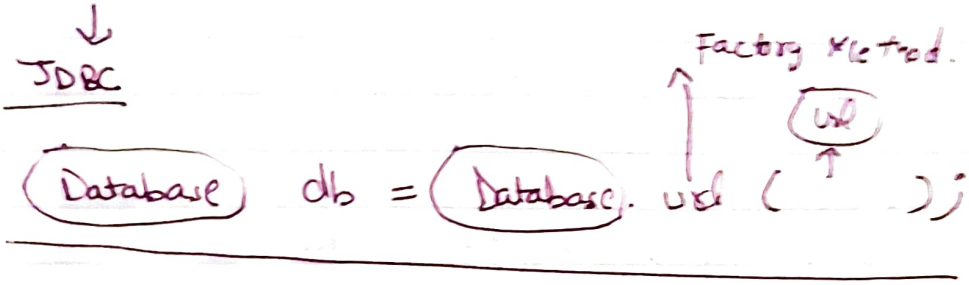


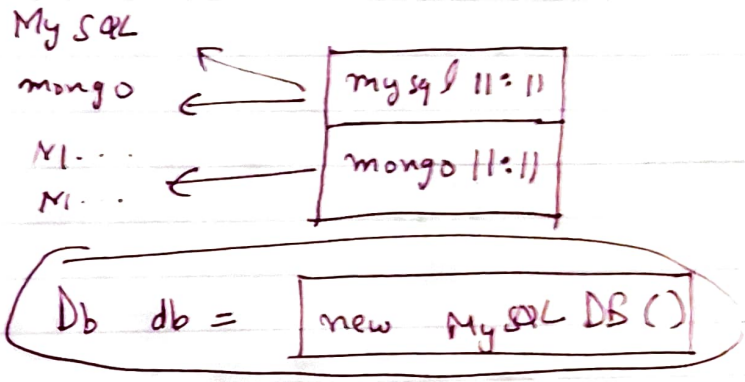
Factory → create objects.
→ w/o specifying sub class.

Database

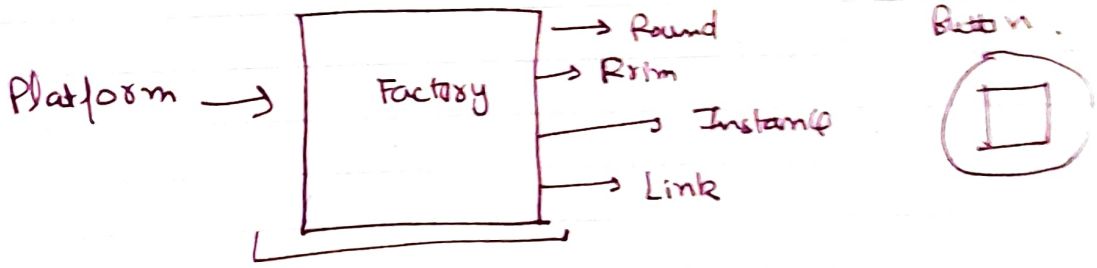
↓

JDBC





Simple factory



If I want to create an instance from multiple subclasses on the basis of a type / conditional,

SIMPLE Factory

```
class ButtonFactory {
    static Button getButton(platform) {
        switch(platform) {
        }
    }
}
```

Button round = ButtonFactory.getBtn(platform);

Simple factory ≠ Design Pattern.

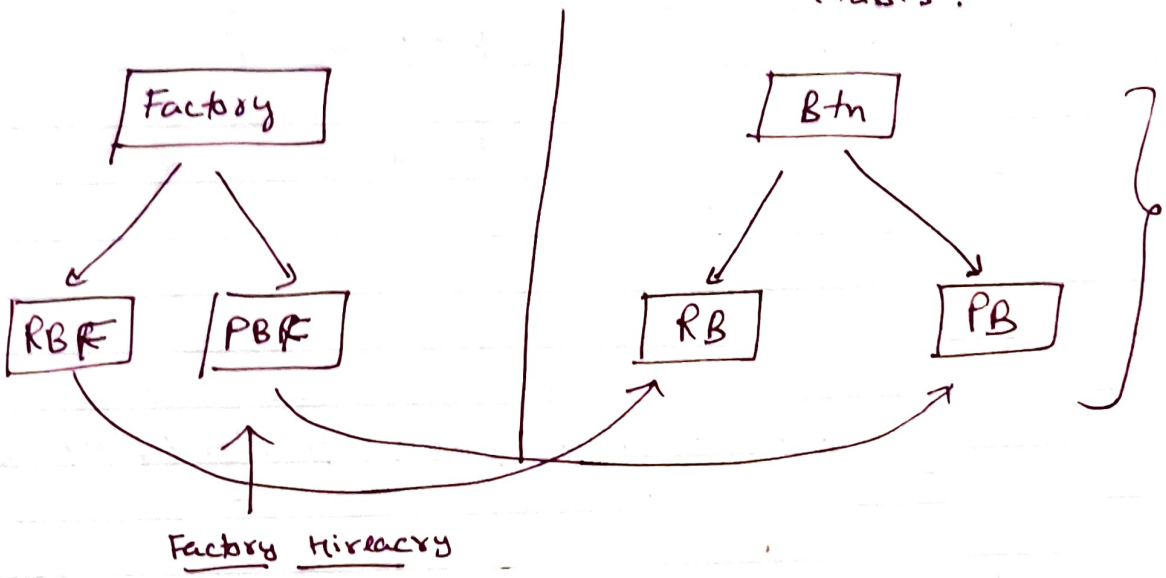
Prototype	Factory
→ Multiple instances without recreating.	→ single instance from multiple types.
→ Still dependent on constructor / subclasing.	→ independent of subclasses.

PROTOTYPE + FACTORY

Factory Method.

one class ⟶ Multiple instances.

1 Factory class ⟶ Multiple product classes.



RBF → Round Button Factory.

① Create factory interface

interface ButtonFactory {

→ Button createBtn (x) ;

}

② Create factory classess.

class RoundButtonFact. imp ButtonFact() {

createButton() {

. . .

}

}