Creational Design Patterns

→ Factory

→ Builder

→ Prototype

Builder Pattern

① Class with many attributes

② Class is immutable

Not every attribute might be having values

Private attributes

No setters

③ Validate attributes before creating class

Sol^{n} 1 Some Class (A a, B b, C c, D d) String name String email

Sol^{n} 2 Some Class (Map<String, Object> map)

Sol^{n} Some Class (Builder builder) {

Some attributes as 'Some Class'

① Copy values

this.a = builder.a;

this.b = builder.b;

)

Builder

. get A() ⟹ Builder return this.

. getB()

. getC()

Google

(1) We have to construct the object of Some Class

SomeClass a = new SomeClass();

↓ provide a method called getBuilder()

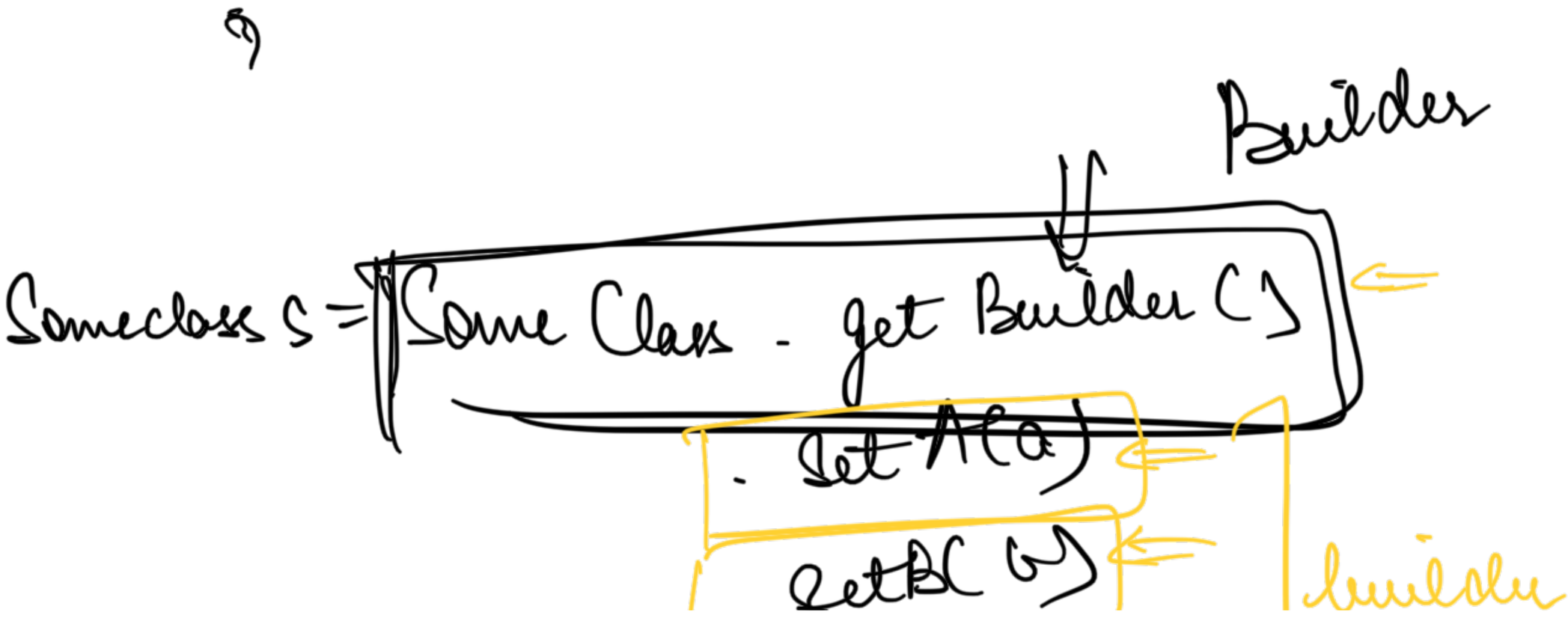
SomeClass.getBuilder()
- setA(a)
- setB(b)
- setC(c);

Class SomeClass

public static Builder getBuilder() {

return new Builder()

}



Some Class (Builder)

⇒ Builder {

return new SomeClass();

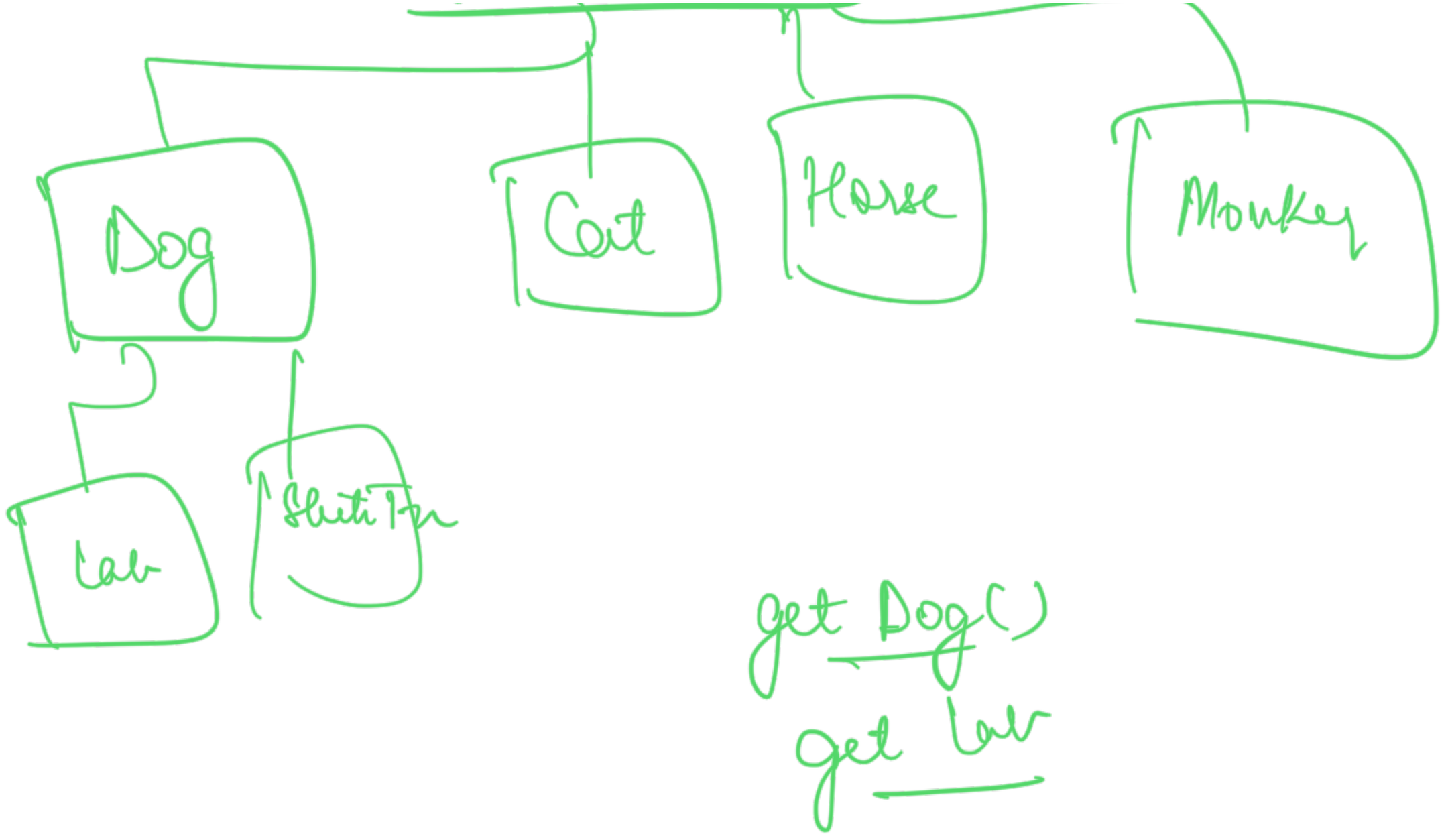
}

}

Factory Design Pattern

Often when we use polymorphism

Animal =



getDog()

getLab

Factory of a particular class is a class whose methods return objects of that class or any of its subclass

Animal factory {

Animal getAnimalOfType(String type) {

if (type == "dog")
return new Dog();

if type == "cat"
return new Cat

}

}

Animal a = Animal factory.getAnimal Of Type ("Cat");

⇓

ICICI → ICICI API

YES → YESBank API

class BankAPIFactory {

→ (BankAPI) getAPI For Provider (Enum Provide) {

interface

if provider == "ec":
    return new ECBankAPI()

if provider == "icici":
    return new ICICIAPIAd

}

}

① There are more than one implementation
Subclasses of a particular interface (class.

↳ Strategy) Adapter

App(Theme)

new frontend app

new UI (Theme. BLACK)

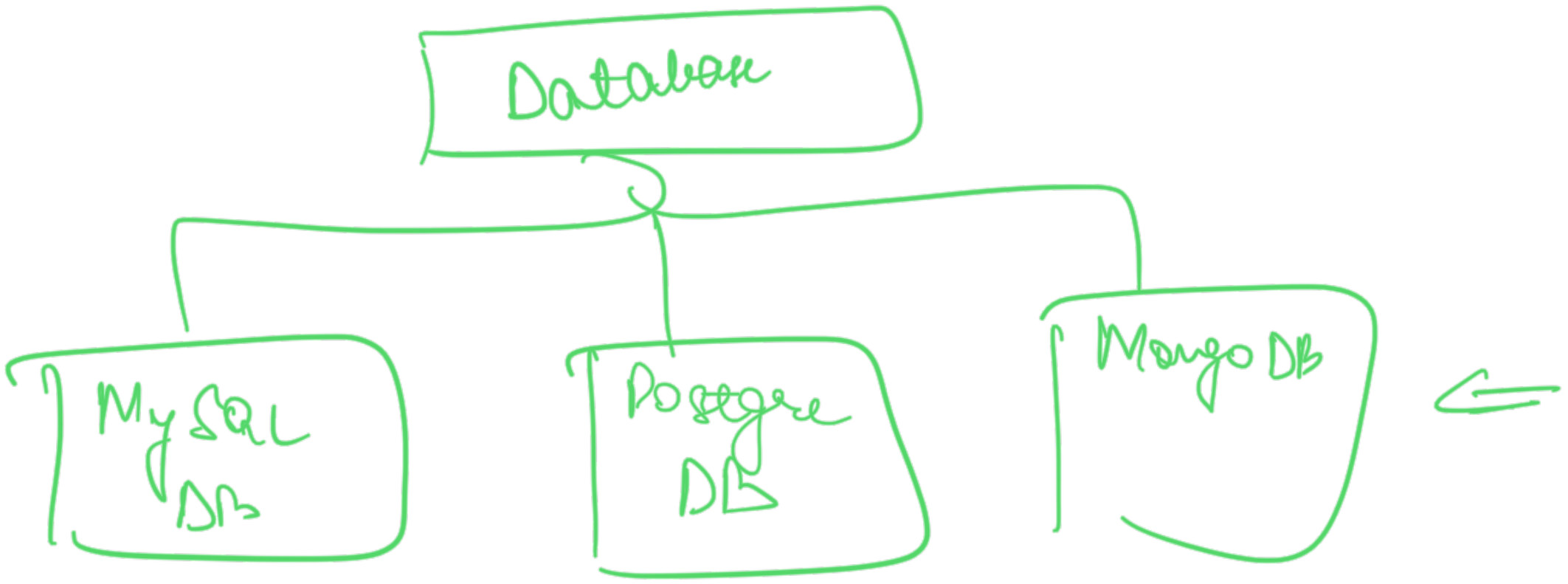
(Theme. WHITE)

Theme = Theme Factory. getTheme for Type (Theme. MATERIAL) {

3

1

DatabaseURL = "mongodb+srv://..."



abstract time.

≒ DatabaseFactory . getDBForURL (url) {

if url starts with (mysql://)

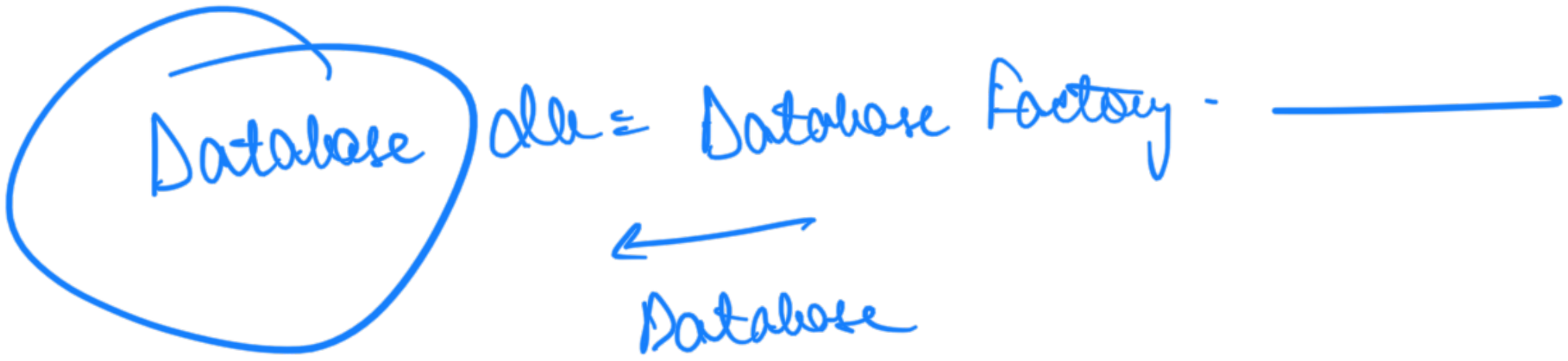
return new MySQLDBC()

(psql://)

PostgreDBC()

Code to an interface. Never code to an implementation.

Dependency Inversion



dependent of Database interface

Factory Method DP

Abstract factory DP

Prototype DP

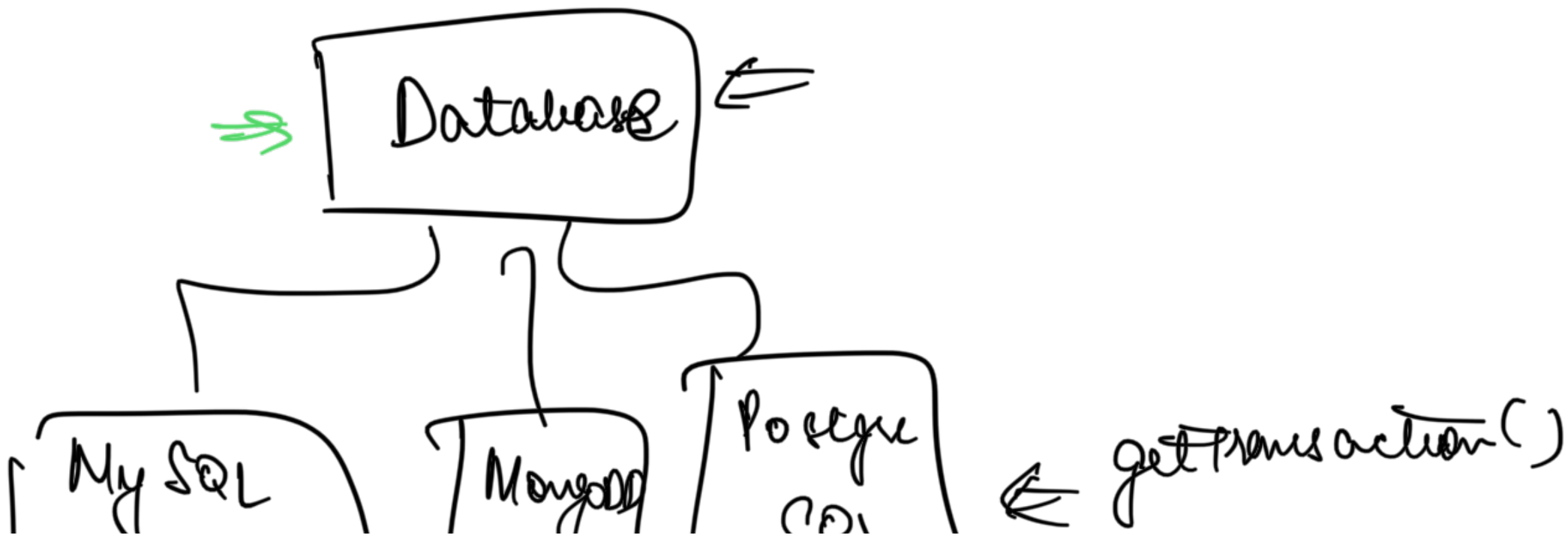
interface Database {

getConnection() ⇒ String

Factory Method → getTransaction() → Transaction

getUrl() → String

getConnect() → int



Transaction

Transaction

Transaction

Database

SAP

getConn()

getQuery()

getTransaction()

getCat

getConnection

Database : Transaction

MySQL : MySQL Trans

Mongo DB : MongoDB Transac

A method in an interface / abstract

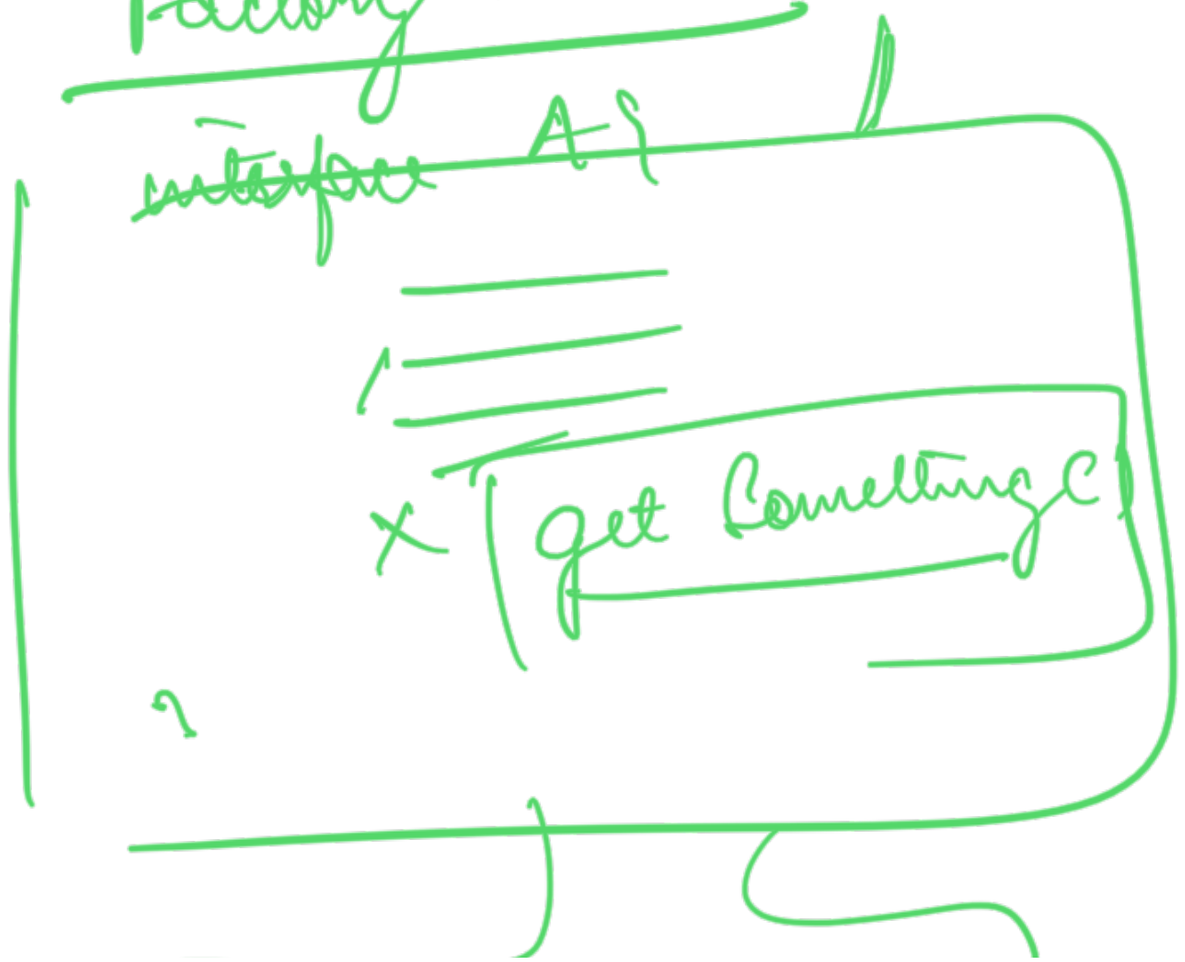
class whose imp in the child class

ONLY involves creating & returning an object of one of subclass' imp of parent

is known as Factory Method.

Practical factory

Factory Method




```
DatabaseFactory {

    static getDBForType( ) {

}
```

)

Abstract Factory DB

ORIGINAL

interface Database {

getURL()
getConnUnit()
}

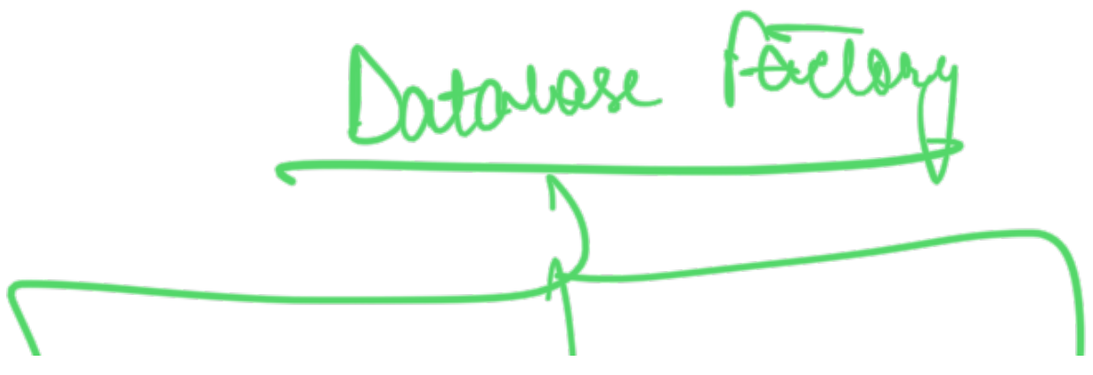
getConn()
getTransactn()
getQuery()

FINAL !

interface Database {
    getConn()
    getConnDetails
}

interface DatabaseFactory {
    Conn getConn
    Trans getTrans
    Query getQuery
}

A class with only factory methods



MySQL DB fac

Mongo DBfac

Postgr DBFa

interface Theme

getName()

getAuthor()

getSource()

getButton(): Button

getMenuBar(): MenuBar

getDropdown(): Drop

related to theme

Material Butto

Material Men

Factory Method

Theme {
getName()
getAuthor()
getSource()
}

ThemeElements() {
getButton()
getMenus()
getDropdown()
}

Material UI Theme

Material Theme Elements

Factory

Factory Method

interface IS

Abstract Factory

class XFactory {

X getXBy_() {

X getXBy_() {

}

interface DB {

Query getQuery()

MySQLDB

Query () {
return MySQLQuery()
}

interface DB
ElementFactory {
}

Only purpose is
to get an object
of corresponding type
in imp