Creational Design Patterns

creating things

Agenda

1. factory
   - Practical factory
   - factory Method
   - Abstract factory
2. Prototype

Factory : place for creation of things
: where will we create thing

↓

code unit that leads to creation of

objects

String typeOfDB = "mySQL"
"postgreSQL"
"MongoDB"

MySQLDatabase
PostgreSQLDB
MongoDB

Database

String typeOfDB = ";

Database db ;

= new MySQLDBC
new Postg
new MongoDB

0

Database.db:

if typeOf DB == 'MySQL'

db = new MySQLDB

elif typeOf DB == "Postgr"

db = new Postgre

Scaler

config.txt → typeOf DB = mySQL

User Service

Second responsibility

Database db;

Open (conn

if typeOfDB == -

Duplicate Code

SRP

Database Factory

Create Database of Correct type

CreateDatabase ForType(String type)

if type == "mySQL"

if else are a part of logic

return new MySQL();

type == "pSQL"

return new PSQL.

y

type == "MySQL"

return MySQL.getInstance

Benefits of Factory

When to use

⇐ Factory always return a new obj

⇒ Whenever I have an interface for whom there are multiple implementations,

I might need to select one of those implementations based on value of some variable.

Rather than having lot of if else in my codebase.

Create a class and ~~that~~ the only purpose of that class will be what object to create.

Factory Method Design Pattern

Connected Class

Database

MySQL DB

2

Url

Username

password

Connect();

list();

CreateQuery();

executeQuery();

Method

Query

CreateQuery();

Subclass of

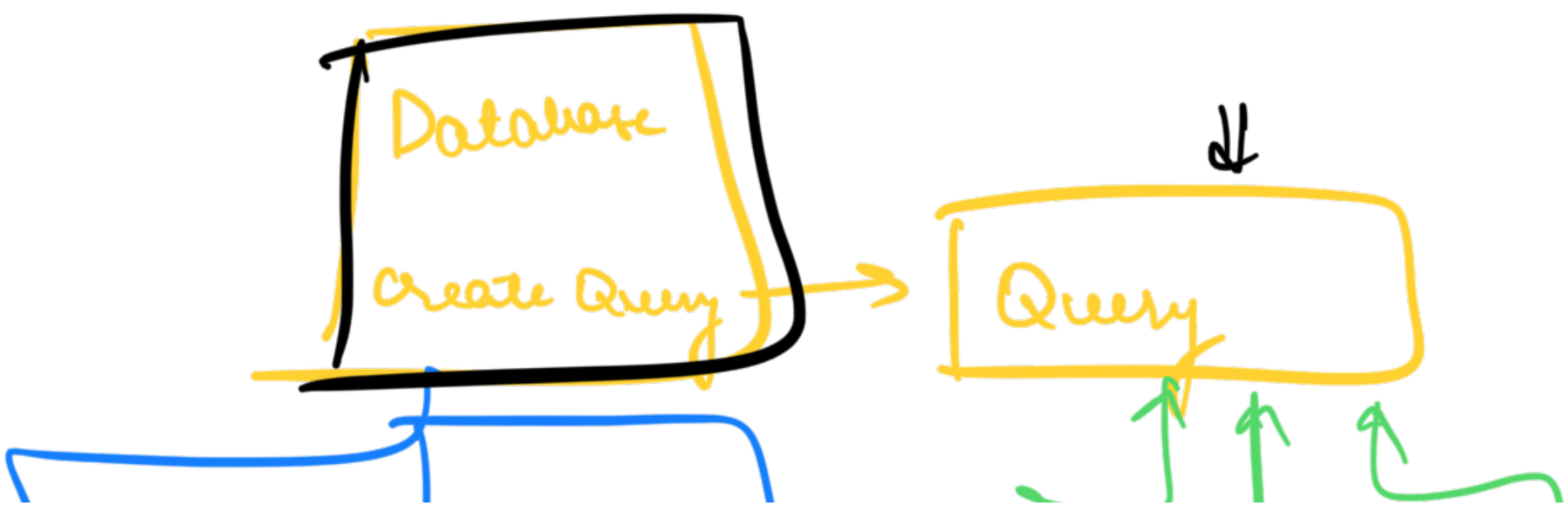
Query

Database db = new Database();

= new MySQLDatabase();

Query q = db.createQuery();

MySQL Query



abstract Database {

abstract createQuery();

}

connect()

class MySQL Database extends Database {

Factory Method

```
    createQuery() {
        return new MySQLQuery
    }
}
```

class PSQL Database extends Database

```
    String url = "psql://" ——————>
    int port = 5432

    createQuery() {
        return new PSQLDB()
    }
}
```

SRP

Database de = MySQL Database

db. create Quer

Abstract Factory Design Pattern

Client creator =

- Using Concrete -

createButton() createButton() createButton()

createDropdown()

create Slider()

interface Theme {

interface / abs class

createButton() ⇒ Button

createDropdown() ⇒ DD

create Shadow () → Shadow

create Menu () → Menu

create Toolbar () → Toolbar

}

class MaterialUITheme implements Theme {

create Button () → MatUIButton

create Menu () → MatUIMenu

create DB → MatUIDB

Theme theme = new iOSTheme();

theme.createButton();

theme.createDAC();

Theme {

name:

primaryColor;

author:

createButton()

Factory

Abstract

Product

create

component

createButton()

createMenu()

createDD()

createNavbar()

ThemeCon

changePrimaryColor()

updateAuthor()

}

Meltdown

ThemeAC

violation of SRP

Theme {

name

pColor

author

changePColor

interface ThemerComponentFactory {

createButton()

createMenu()

createDD()

createToolbar

update Author

factory
Method

createThemeCompFactory → ThemeCompFacto

}

MaterialTheme

MaterialTheme for

}