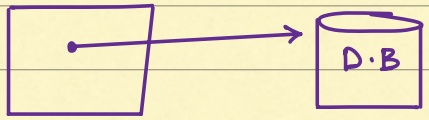
AGENDA

- Factory design Pattern
  - abstract
  - practical
  - factory method

*) FACTORY METHOD:

you have object of DB in user class of project

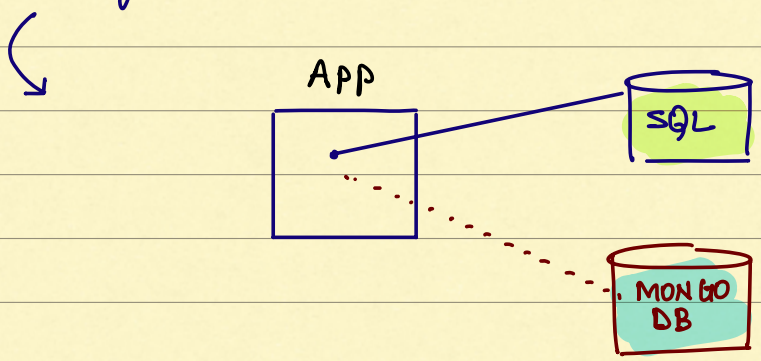



```
class UserService {
    Database db;

    createUser() {
        db.createQuery("INSERT INTO users.....");
        db.execute()
    }

    getUser() {
        db.createQuery("select * FROM ....");
        db.execute()
        .
        .
        .
        .
    }
```

-> 1st we code query | → In Every method.
-> execute query

Q.) Database datatype → i / class / abstract class ???

Mostly (i)



-> Going fwd if you want to use MongoDB / PSQL etc.

PRINCIPLE: D.I.

-> Interface: changing Implementation is easy

Database d= new MySQL();
Database c= new PsQL();

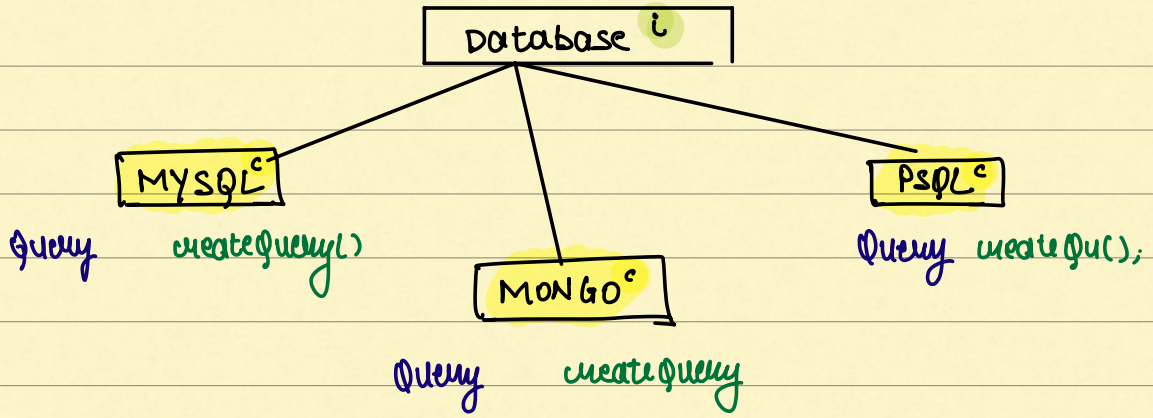
↓

Now:

Q.) what should be Query q ???

should be _____

Reason: Queries for MySQL & MONGODB can be different



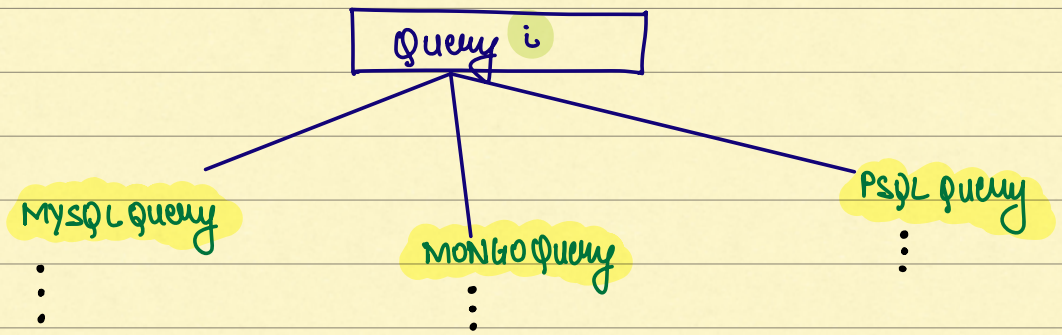
Interface Database {

Query createQuery()

}

Incorporated via Each Implementor.

Query createQuery():



class SQLQuery Implements Query {

Query createQuery() {

.....

}

class MONGOQuery Implements Query {

Query createQuery() {

.....

}

·> similarly → DB can have many methods.

createQuery ()
changeThreadPool()
connect()

createQuery() : returns Object of specific Query.

FACTORY DP:

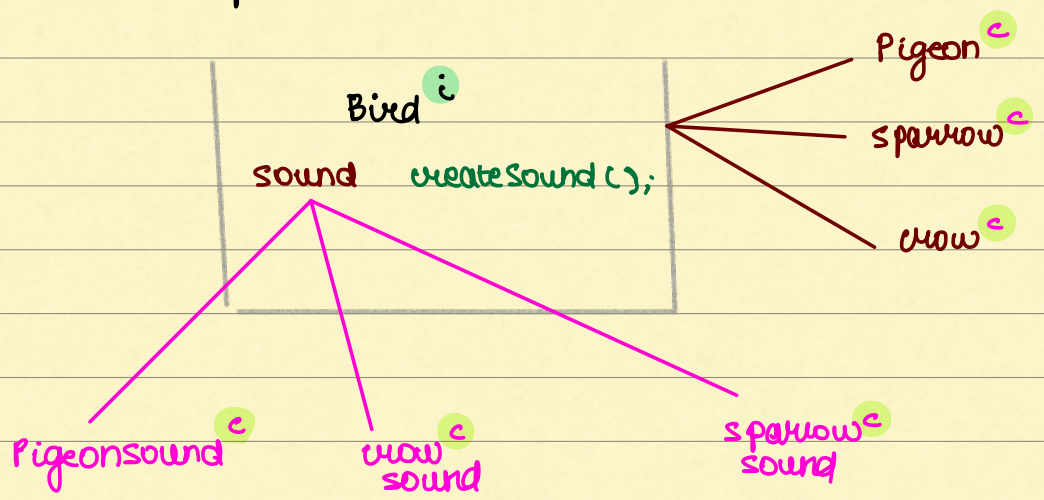
factory Method is a Method in interface which returns instance of another specific class.

*) ADVANTAGE:

·> switching dyb - No Major code change required.

↳ if factory was Not used:
would Need Multiple if-else checks

Another Example:



-> Pigeon / sparrow / crow Implements Bird
-> Pigeonsound / s.s. / c.s. Implements sound.

-> createsound() : factory method
-> Bird can have other methods as well - which are NOT factory methods.

Now, PROBLEM: Increased Responsibility

DB example:

```
interface database {
    Query createQuery();        // factoryMethod
    int   getThreadPoolCount();
    String hostURL();           // Normal Methods
}
```

what if there are more factory methods in DBi ??

② Transaction createTransaction()
③ Execute execute();

→ Now you have ③ factory methods

Responsibility of Databasei increases:

•> Now, db has many factory methods - to get corresponding objects.

∴ S.R.P. is violated.

↓

*) ABSTRACT FACTORY:

divide responsibility

class 1: has attributes / General methods
class 2: has ONLY factory methods

Above Example:

Database Interface

getThreadPoolCount()

hostURL()

↓

ONLY General Methods

Database factory

Query

Transaction

Execute

createQuery

createTransaction

execute

↓

ONLY factory Methods

code structure:

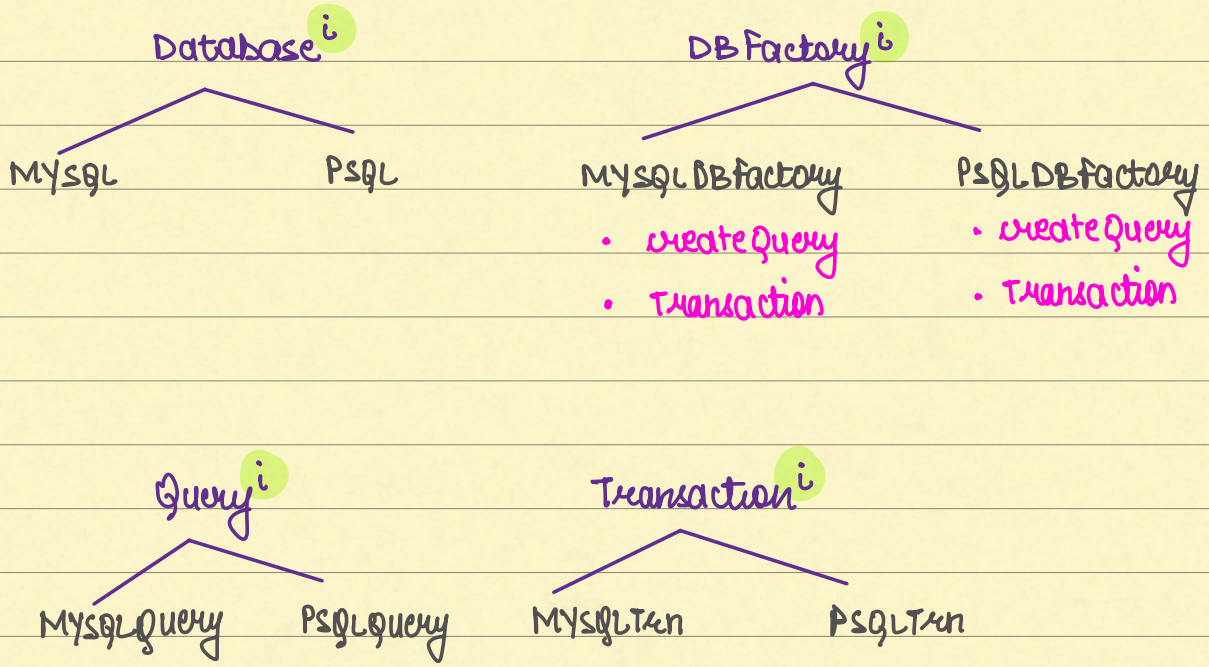
```
class UserService {
    Database db;
    DatabaseFactory dbf;

    ..... createDatabaseFactory() {

        return DatabaseFactory
    }

}
```

Abstract Factory:

if one class/i becomes very large due to many factory methods (breaks SRP) - divide in (2) interfaces

1.) All Normal methods
2.) All factory methods

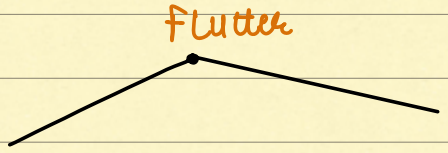
*) COMPLETE DIAGRAM:



Real usecase:

UI / Mobile libraries: Android / iOS

Cross Platform Frameworks: Flutter / React Native



ANDROID

createButton()

iOS

createButton()

NORMAL CODE:

Flutter {

createButton() {

if (platform == iOS)

iOS Button

else if (platform == Android)

android Button

}

NOT CORRECT

createMenu() {

:

}

}

violates: OCP

↳ windows support !!

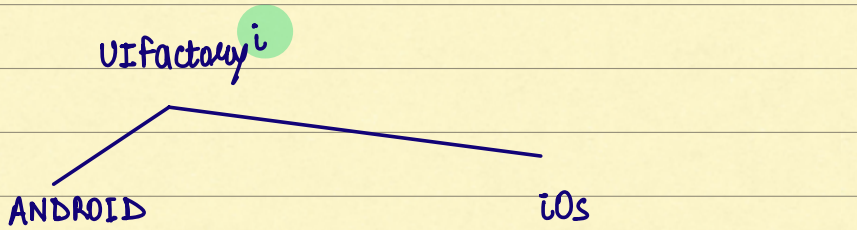
better solution → use Abstract Factory.

class Flutter {

VIFactory createUIFactory()

void setRefreshRate()

}



iButton createButton()
iMenu createMenu()

iButton createMenu()
iMenu createButton()

iButton / iMenu → Interfaces

DEMO

*) PRACTICAL FACTORY:

Mostly used among all factories

eg: DBfactory

i
DBfactory {

Query createQuery()
Transaction createTrans()

}

(↓)
Abstract factory

↳ Returns different objects using diff. methods (v.v. rare)

[you need object of variant by class]

c
DBfactory {

Database createDBByName() {
if (name == "MySQL")
return MySQL DB
else if (name == 'MONGO')
return MONGODB()
}

↓
(Practical factory)

mostly used

Advantage:
→ NO if-else in client code

∴ you can use Practical factory when you need object of same class/Implementation

Usecases:

1.) you need object of any (1) variant of class with Multiple Implementations

Does Practical factory violate OCP ??

DEMO