Design Pattern About Factory and Abstract Factory

> Designed by Zhu Yueming

Part 1 Abstract Factory

Experimental Objective

- According to source code, learn what is the benefit of abstract factory, in which circumstance using abstract factory is better, and how to use abstract factory design pattern in different design layer.
- Then exercise the basic usage of singleton design pattern.
- Finally, exercise how to separate the dao layer from service layer in different ways.

Introduce of source code

1. Requirement introduction

There are two different kinds of databases in a project, the one is Mysql, and the other is SqlServer. In the project, there are two entity classes (**Staff** and **Computer**), and we need to do insert, update and delete operations of those two entities in both two databases respectively.

The original code is using simple factory design pattern. For example the `ComputerFactory` can return an instance of `MysqlComputerDao` or `SqlServerComputerDao` by passing different parameter. Accordingly, the client needs to generate two instances for two different factories, and then whether we can get correct instance of `StaffDao` and `ComputerDao` is determined by passing correct parameter.

In this tutorial, the task is that how can we get the instance of `StaffDao` and `ComputerDao` in a better way.

2. How to use it

After we get two instances, we can do a simple test:

```
1 2 3 4 5 6 0
insert staff into Mysql database successfully
update Staff in Mysql database successfully
delete Staff in Mysql database successfully
insert computer into Mysql database successfully
update computer in Mysql database successfully
delete computer in Mysql database successfully
```

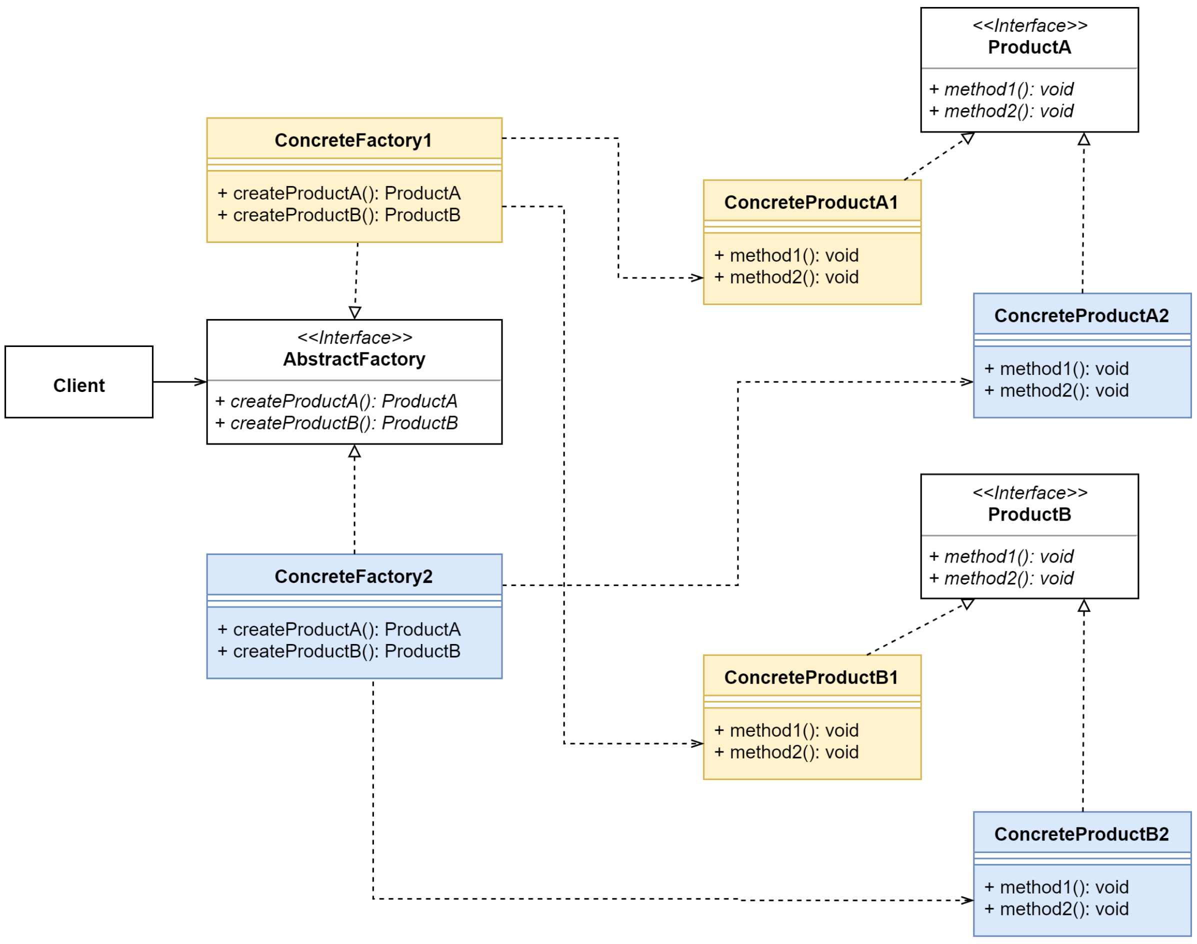
3. Disadvantage analysis

What we want is to interact with only one database for one time, however if we pass a wrong parameter, we might interact the staff information with one database as the same time interact the computer information with another database

Abstract Factory

The abstract factory design pattern can provide a way to encapsulate a group of individual factories that have a common theme without specifying their concrete classes. Which means, the products created by only one concrete factory can not from different themes.

1. Class Diagram



Task 1

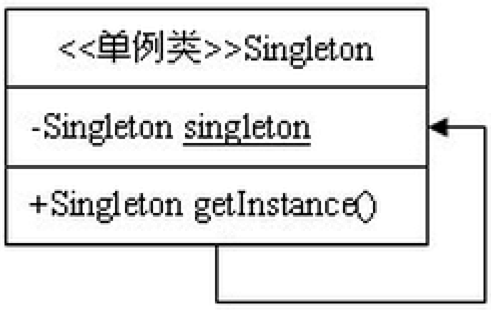
In `abstractFactory` package, please modify your code by using abstract factory design pattern in `dao` layer, and make sure that it can match the client.

Hints: You need to created two concrete classes including `MysqlDaoFactory` and `SqlServerDaoFactory` to implements the interface `DaoFactory`

Refactoring Abstract Factory by Singleton

The Singleton Pattern ensures a class has only one instance and provides a global point of access to that instance.

1. Class Diagram



2. Sample code

```java
public class Singleton {
    private static Singleton instance = null;

    private Singleton() {}

    public static synchronized Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}
```

or

```java
public class Singleton {
    private static Singleton instance = new Singleton();

    private Singleton() {}

    public static Singleton getInstance() {
        return instance;
    }

}
```

Using reflection to separate two layers

If we want to switch database from one to another, we need to instantiate another concrete factory accordingly, in the process the source code needs to be changed. A better way is that, we can define the name of concrete factory into configure file, and then using reflection to instantiate an instance, so that we can switch the database only by changing the configure file instead of modifying the source code. In this case, defining two concrete factories are duplicated, only one is enough.

1. Design a class named `PropertiesReader`

```
import java.io.*;
import java.util.Properties;

public class PropertiesReader {
    public static Properties readProperties(String url){
        Properties prop=new Properties();
        try{
            InputStream in=new BufferedInputStream(new FileInputStream(url));
            prop.load(in);
            in.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
        return prop;
    }
}
```

2. Create a properties file named `resources.properties`

```
StaffDao= MysqlStaffDao
ComputerDao = MysqlComputerDao
```

3. Design a java class named `DaoFactoryImpl`

```
public class DaoFactoryImpl implements DaoFactory{
    @Override
    public StaffDao createStaffDao() {
        return null;
    }

    @Override
    public ComputerDao createComputerDao() {
        return null;
    }
}
```

Complete all codes in `DaoFactoryImpl` , and modify it to be a singleton. Which concrete product it would be created is according to the data in properties file.

Hint:

How to create instance according to the String name of class?

```
String className = "MysqlStaffDao";
Class clz = Class.forName(className);
Object instance = clz.getDeclaredConstructor().newInstance();
```

Task 2

According to the promp above, in **singleton** package, please merge two concrete factories into one concrete factory (`DaoFactoryImpl`), and you need make sure that the instantiate process for `StaffDao` and `ComputerDao` is by reflection, and the class name of concrete factories are from `resource.properties`.

Modify your concrete factory (`DaoFactoryImpl`) to be a **singleton**.

Your code need to match the client.