

Tutorial 10: Abstract Factory and Singleton

[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.

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
```

Abstract Factory Design Pattern

1. The problem in simple factory:

If the client does not understand the parameters of the methods in two factories, what will happen?

Suppose, we try to modify the code in **client** as follows:

```
StaffDao staffDao=staffFactory.createStaffDao(1);
ComputerDao computerDao=computerFactory.createComputerDao(2);
```

The result will be:

```

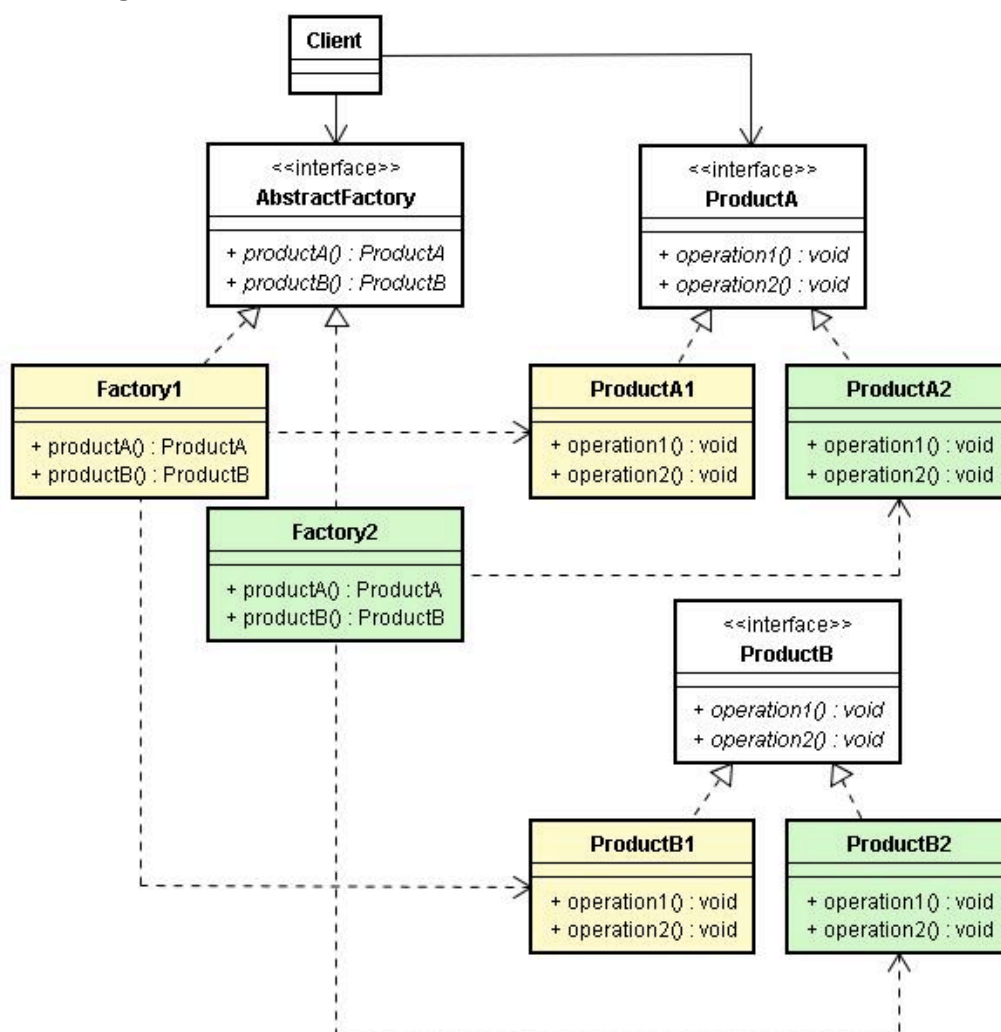
1 2 3 4 5 6 0
insert staff into Mysql database successfully
update Staff in Mysql database successfully
delete Staff in Mysql database successfully
insert computer into SqlServer database successfully
update computer in SqlServer database successfully
delete computer in SqlServer database successfully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

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.

2. What is the abstract factory can do?

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.

3. 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 create two concrete classes including **MysqlDaoFactory** and **SqlServerDaoFactory** to implement the interface **DaoFactory**

Singleton Design Pattern

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



3. Sample Code of Singleton

```

public class Singleton {
    private static Singleton instance = null;

    private Singleton() {}

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

```

or

```

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.

Here is a sample code of instantiate a class from properties file.

```
public class demo {
    public String toString(){
        return "hello world";
    }

    public static void main(String[] args){
        Properties prop=new Properties();
        try{
            InputStream in=new BufferedInputStream(new
FileInputStream("src/resource.properties"));
            prop.load(in);

            try {
                Class clz=Class.forName(prop.getProperty("classname"));
                demo object= (demo)clz.getDeclaredConstructor().newInstance();
                System.out.println(object);
            }catch (ClassNotFoundException e) {
                e.printStackTrace();
            }catch (InstantiationException e) {
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            } catch (InvocationTargetException e) {
                e.printStackTrace();
            } catch (NoSuchMethodException e) {
                e.printStackTrace();
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Task 2:

- 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.

Dependency Injection

We can also use the way of dependency injection to separate two layers. The process is that, when declaring an interface, we need to declare its implement class and register them into a hash map. Here you need to refactor your code based on Task1.

Task3:

Step1: Create an Interface named **InjectFactory** as follows.

```
public interface InjectFactory {

    <T> void register(Class<T> serviceType, Class<? extends T>
implementationType);

    <T> T resolve(Class<T> serviceType);
}
```

Step2: Implement the factory by using following code:

```
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Modifier;
import java.util.HashMap;
import java.util.Map;

public class InjectFactoryImpl implements InjectFactory {
    Map<Class<?>, Class<?>> map = new HashMap<>();

    @Override
    public <T> void register(Class<T> serviceType, Class<? extends T>
implementationType) {
        if (Modifier.isAbstract(implementationType.getModifiers())
|| Modifier.isInterface(implementationType.getModifiers())) {
            throw new IllegalArgumentException();
        }
        if (serviceType != null) {
            map.put(serviceType, implementationType);
        }
    }

    @Override
    public <T> T resolve(Class<T> serviceType) {
        try {
            return (T)
map.get(serviceType).getDeclaredConstructor().newInstance();
        } catch (InstantiationException | IllegalAccessException |
InvocationTargetException | NoSuchMethodException e) {

```

```

        e.printStackTrace();
    }
    return null;
}
}

```

Step3. Design injection process as follows:

```

public static InjectFactory injectProcess() {
    InjectFactory injectFactory = new InjectFactoryImpl();
    injectFactory.register(DaoFactory.class, MysqlDaoFactory.class);
    return injectFactory;
}

```

Step4. Modify the Client:

```

public static void main(String[] args) {
    InjectFactory factory= injectProcess();
    DaoFactory daoFactory = factory.resolve(DaoFactory.class);
    ComputerDao computerDao = daoFactory.createComputerDao();
    StaffDao staffDao = daoFactory.createStaffDao();

    test(staffDao, computerDao);
}

```

It is only a sample example for this tutorial, see more in the forth assignment.

(designed by Yueming Zhu)