Tutorial 9: Factory Design Pattern

[Experimental Objective]

Learn how to refactor interface-oriented code to simple factory design pattern, static factory design pattern and factory method design pattern respectively. In the process, you can appreciate the role of factory model in practical project.

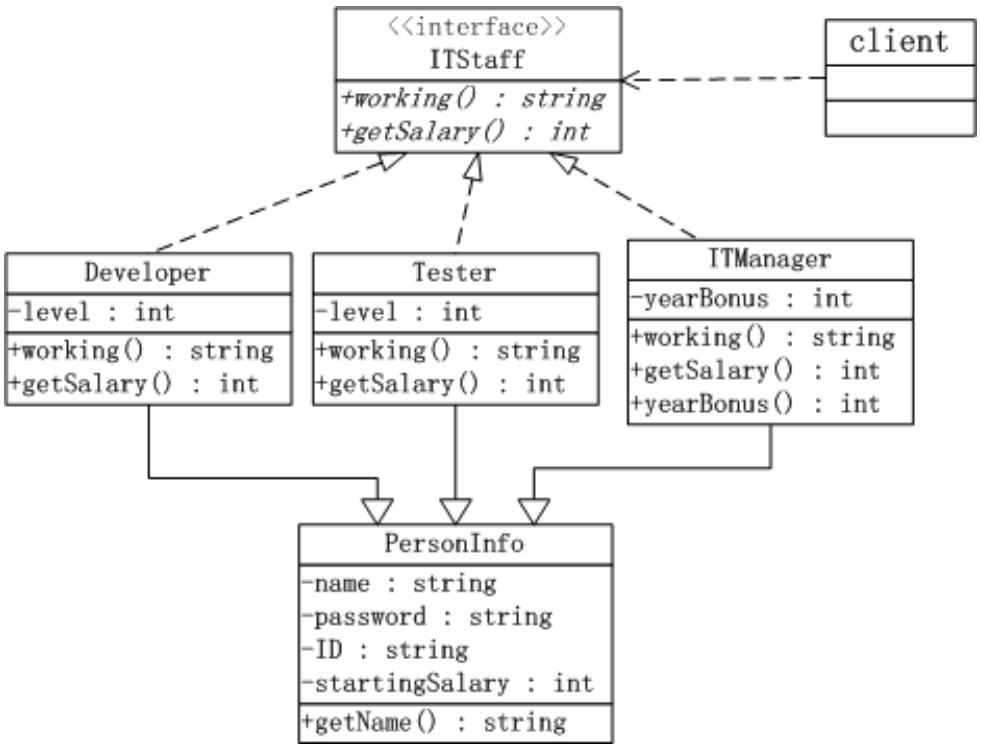
[Introduce of source code]

1. Requirement introduction:

In development department, the types of staff and their salary are shown in following table.

ITStaff	Description of work (**working**)	His salary（**getSalary**）
Developer	Coding	10000+level * 2000
Tester	Testing	8000+level * 1500
ITManager	IT Manager	30000 (he has additional bonus)

2. Class diagram:



3. How to use it?
We can use it in command window.
Input 1 means an instance of ITManager has been created.
Input 2 means an instance of Developer has been created.

Input 3 means an instance of Tester has been created.
Input 4 means print out all user information by order of salary.
Input 5 means print out all user information by order of working.
Input 0 can stop the program.

```
1 2 3 2 1 4 5 0
All information:
Testing     name: 10003 Tester   , salary: 11000
Coding      name: 10004 Developer, salary: 12000
Coding      name: 10002 Developer, salary: 14000
IT Manager  name: 10005 ITManager, salary: 30000, bonus in the end of year 21000
IT Manager  name: 10001 ITManager, salary: 30000, bonus in the end of year 0
All name:
Coding      name: 10004 Developer, salary: 12000
Coding      name: 10002 Developer, salary: 14000
IT Manager  name: 10005 ITManager, salary: 30000, bonus in the end of year 21000
IT Manager  name: 10001 ITManager, salary: 30000, bonus in the end of year 0
Testing     name: 10003 Tester   , salary: 11000
```

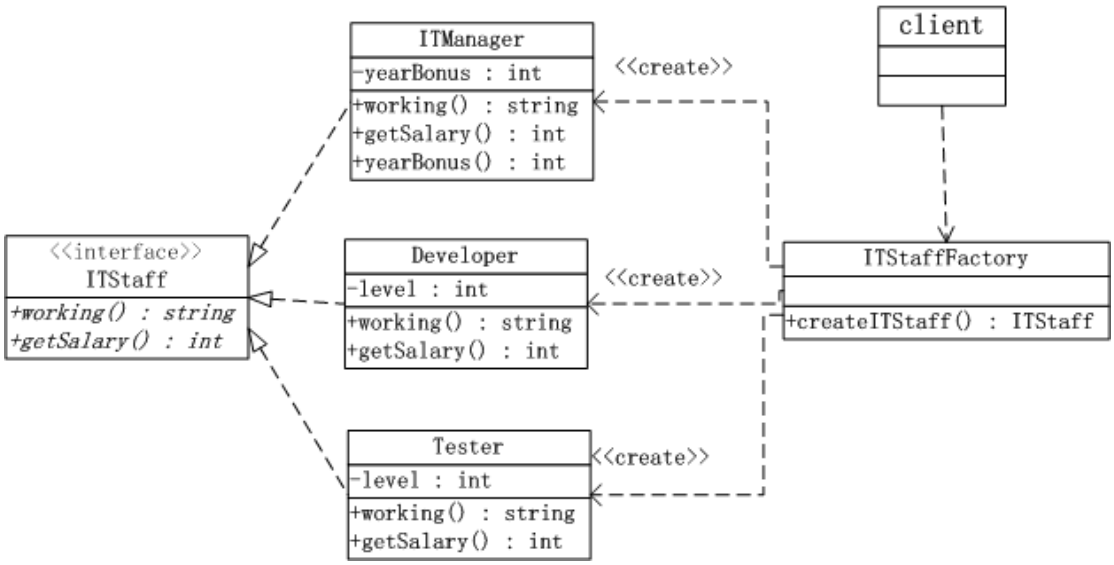
4. Design problems if we only use interface.

How can we do if an interface are given to us without understand which classes can implement it?
Consequently, in interface-oriented programming, clients not only need to understand interface but also need to understand for which classes can implement such interface. Actually, the implementation classes should be encapsulated by interface and isolated form client, in other words, the client does not need to know what the specific implementation classes (ITManager, Developer, Tester) of the interface (ITStaff) are.

Simple Factory

To solve the previous problem, we can provide a class, the function of which is to provide functions which can create instances. The class can be regarded as factory, and a factory can be an interface, abstract class or a class.

The main internal implementation of the simple factory is to “select the appropriate implementation class” to create the instance. Since we want to realize the select operation, we need to pass parameters, which represent different choices. This condition and parameters can be derived from client.



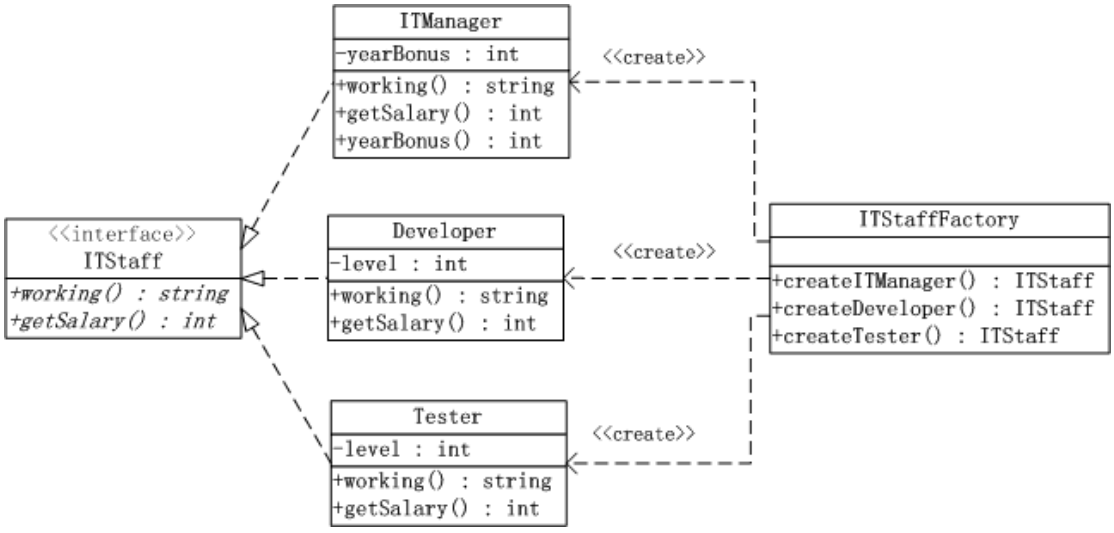
Task 1

Create a package named **SimpleFactory**, then copy your source code here, and modify them to simple factory design pattern.

Hints: Create a class named **ITStaffFactory**, in which there is a method named **createITStaff()**, and we need to passing parameters in this method to distinguish which implementation class we need to instantiate.

Static Factory

The difference between static factory and simple factory is that several static methods should be defined in factory to instantiate different objects. The client needs to understand the definition of those static methods.



Task2:

Create a package named **staticFactory**, then copy your source code here, and modify them to static factory design pattern.

Hints: Create a class named **ITStaffFactory**, in which you need to design three static methods **createITManager(), createDeveloper()** and **createTester()** and the return value of those three methods is a **ITStaff** type.

Disadvantage of simple factory:

1. Increase the complexity of client usage.
2. Inconvenient to expand sub-factories.

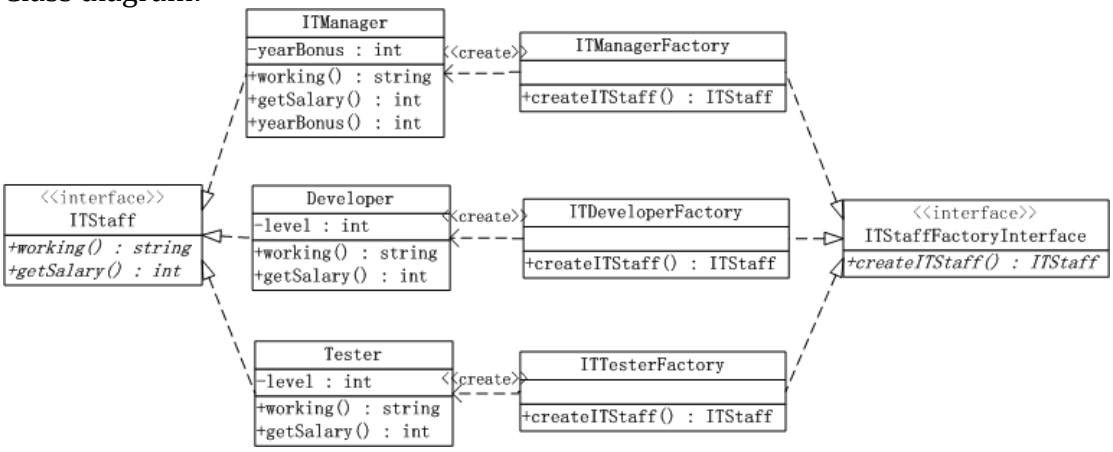
Factory method

For simple factory, the process of creating instance is in the factory class by the way like "selective implementation", while for factory method, it will transfer the process to subclasses. The factory is an abstract class, and the methods inside are also abstract rather than concrete implementations.

In factory method:

1. Abstract factory: It is the core of the factory method pattern. It can be an interface that a concrete factory must implement, or it can be a parent class that concrete factories need to inherit.
2. Concrete factory: It can create objects for the corresponding specific product.
3. Abstract product (ITStaff): It is an interface or a parent class to be implemented or inherited by concrete product classes.
4. Concrete product (ITManager etc.): The instance of which is created by corresponding concrete factory.

Class diagram:



Task3: Create a package named **factoryMethod**, then copy your source code here.

1. Modify it to factory method design pattern.
   Hints: Create an interface named **ITStaffFactoryInterface**, and then created three concrete factory classes to implements the interface.
   Those three factory can be named as follows:
   ITManagerFactory
   DeveloperFactory
   TesterFactory

2. Adding an concrete class named **ArtDesigner** and we can describe it as follows:
   Attributes:
   level: the value is from 1 to 5
   Starting Salary: 7000

name: userID+" "+"ArtDesigner"
For other attributes are similar to Developer or Tester classes.

Method:

Constructor and tostring method are similar to which in Developer or Tester classes.
working: It needs to return "Art Design"
getSalary: startSalary+ level*1500.

3. Input type for client:

Input 1 means an instance of ITManager has been created.
Input 2 means an instance of Developer has been created.
Input 3 means an instance of Tester has been created.
Input 4 means an instance of ArtDesigner has been created.
Input 5 means print out all user information by order of salary.
Input 6 means print out all user information by order of working.
Input 0 can stop the program.

```
1 2 3 4 5 6 0
All information:
Testing     name: 10003 Tester   , salary: 12500
Art Design  name: 10004 ArtDesigner, salary: 13000
Coding      name: 10002 Developer, salary: 20000
IT Manager  name: 10001 ITManager, salary: 30000, bonus in the end of year 9000
All name:
Art Design  name: 10004 ArtDesigner, salary: 13000
Coding      name: 10002 Developer, salary: 20000
IT Manager  name: 10001 ITManager, salary: 30000, bonus in the end of year 9000
Testing     name: 10003 Tester   , salary: 12500
```