云南大学软件学院 实验报告

课程： 软件工程概论 学期： 2024 - 2025学年 第一学期 任课教师： 莫启

专业： 软件工程 学号： 20221120044 姓名： 唐嘉骏 成绩：

**实验4**

（1）首先详细地阐述抽象工厂模式，然后利用熟悉的编程语言（如Java、Python等）实现抽象工厂模式。

抽象工厂模式是一种创建型设计模式，其核心思想是创建一个负责创建具体工厂对象的工厂接口，而抽象工厂中会定义一些相关功能的接口，这样使得抽象工厂模式可以创建出针对一组相关对象的工厂，而无需明确指定具体的类。每个具体的工厂类可以按照工厂方法生产出每个接口的具体产品。这样可以将客户端与具体产品的生产过程隔离开来，实现解耦。抽象工厂模式在创建一族产品时非常有效。

代码：

ITaxFactory

1. **package** com.krowfeather.rggl;
3. **public** **interface** ITaxFactory {
4. Double calculateTax(Double baseSalary);
5. }

IBonusFactory

1. **package** com.krowfeather.rggl;
3. **public** **interface** IBonusFactory {
4. Double calculateBonus(Double bonus);
5. }

ChineseTaxFactory

1. **package** com.krowfeather.rggl;
3. **public** **class** ChineseTaxFactory **implements** ITaxFactory {
5. @Override
6. **public** Double calculateTax(Double baseSalary) {
7. **return** baseSalary \* 0.15;
8. }
9. }

ChineseBonusFactory

1. **package** com.krowfeather.rggl;
3. **public** **class** ChineseBonusFactory **implements** IBonusFactory {
4. @Override
5. **public** Double calculateBonus(Double baseSalary) {
6. **return** baseSalary \* 0.05;
7. }
8. }

USATaxFactory

1. **package** com.krowfeather.rggl;
3. **public** **class** USATaxFactory **implements** ITaxFactory {
4. @Override
5. **public** Double calculateTax(Double baseSalary) {
6. **return** baseSalary \* 0.05;
7. }
8. }

USABonusFactory

1. **package** com.krowfeather.rggl;
3. **public** **class** USABonusFactory **implements** IBonusFactory{
4. @Override
5. **public** Double calculateBonus(Double bonus) {
6. **return** 0.15 \* bonus;
7. }
8. }

BaseSalary

1. **package** com.krowfeather.rggl;
3. **public** **class** BaseSalary {
4. **private** Double baseSalary=10.0;
5. **public** Double getBaseSalary() {
6. **return** baseSalary;
7. }
9. **public** **void** setBaseSalary(Double baseSalary) {
10. **this**.baseSalary = baseSalary;
11. }
12. }

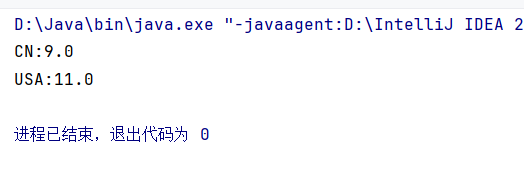
Client

1. **package** com.krowfeather.rggl;
3. **public** **class** Client {
4. ITaxFactory taxFactory;
5. IBonusFactory bonusFactory;
6. BaseSalary baseSalary;
7. **public** Client(ITaxFactory taxFactory, IBonusFactory bonusFactory, BaseSalary baseSalary) {
8. **this**.taxFactory = taxFactory;
9. **this**.bonusFactory = bonusFactory;
10. **this**.baseSalary = baseSalary;
11. }
12. **public** Double calculateSalary() {
13. Double tax = taxFactory.calculateTax(baseSalary.getBaseSalary());
14. Double bonus = bonusFactory.calculateBonus(baseSalary.getBaseSalary());
15. **return** baseSalary.getBaseSalary() - tax + bonus;
16. }
17. }

Main

1. **package** com.krowfeather.rggl;
3. **public** **class** Main {
4. **public** **static** **void** main(String[] args) {
5. ITaxFactory taxFactory = **new** ChineseTaxFactory();
6. IBonusFactory bonusFactory = **new** ChineseBonusFactory();
7. BaseSalary baseSalary = **new** BaseSalary();
8. Client client = **new** Client(taxFactory, bonusFactory, baseSalary);
9. System.out.println("CN:"+client.calculateSalary());
11. ITaxFactory taxFactory1 = **new** USATaxFactory();
12. IBonusFactory bonusFactory1 = **new** USABonusFactory();
13. BaseSalary baseSalary1 = **new** BaseSalary();
14. Client client1 = **new** Client(taxFactory1, bonusFactory1, baseSalary1);
15. System.out.println("USA:"+client1.calculateSalary());
16. }
17. }

测试：



1. 首先详细地阐述适配器模式，然后利用熟悉的编程语言（如Java、Python等）实现适配器模式。

适配器模式是一种结构型设计模式，它通过引入一个适配器类，将一个接口转换成客户端期望的另一个接口，使得原本因接口不兼容而无法一起工作的类能够协作。使用适配器模式可以使得代码复用性增强。具体的，适配器可以使用继承和组合两种方式实现，继承方式是通过让适配器去继承需要适配的类同时适配器实现接口的方法，在方法中调用父类的具体方法；组合方式是让适配器去组合需要适配的类，让需要适配的类成为其成员变量，这样适配器去实现接口则变为直接调用需要适配的类的方法即可。

1. 继承方式

Target

1. **package** com.krowfeather.rggd;
3. **public** **interface** Target {
4. **void** request();
5. }

Adapter

1. **package** com.krowfeather.rggd;
3. **public** **class** Adapter **extends** Adaptee **implements** Target  {
4. @Override
5. **public** **void** request() {
6. **super**.specificRequest();
7. }
8. }

Adaptee

1. **package** com.krowfeather.rggd;
3. **public** **class** Adaptee {
4. **public** **void** specificRequest() {
5. System.out.println("Specific request");
6. }
7. }

Client

1. **package** com.krowfeather.rggd;
3. **public** **class** Client {
4. Target target;
5. **public** Client(Target target) {
6. **this**.target = target;
7. }
8. **public** **void** request() {
9. target.request();
10. }
11. }

Main

1. **package** com.krowfeather.rggd;
3. **public** **class** Main {
4. **public** **static** **void** main(String[] args) {
5. Client client = **new** Client(**new** Adapter());
6. client.request();
7. }
8. }

测试：



1. 组合方式

Client,Adaptee,Target与继承方法相同

Adapter

1. **package** com.krowfeather.rggd;
3. **public** **class** Adapter **implements** Target  {
4. Adaptee adaptee;
5. **public** Adapter(Adaptee adaptee) {
6. **this**.adaptee = adaptee;
7. }
8. @Override
9. **public** **void** request() {
10. adaptee.specificRequest();
11. }
12. }

Main

1. **package** com.krowfeather.rggd;
3. **public** **class** Main {
4. **public** **static** **void** main(String[] args) {
5. Client client = **new** Client(**new** Adapter(**new** Adaptee()));
6. client.request();
7. }
8. }

测试：

