外观模式实验

**【实验目的及要求】**

1. 掌握外观模式的结构与实现；
2. 学会使用外观模式开发应用程序；
3. 明白外观模式的扩展应用。

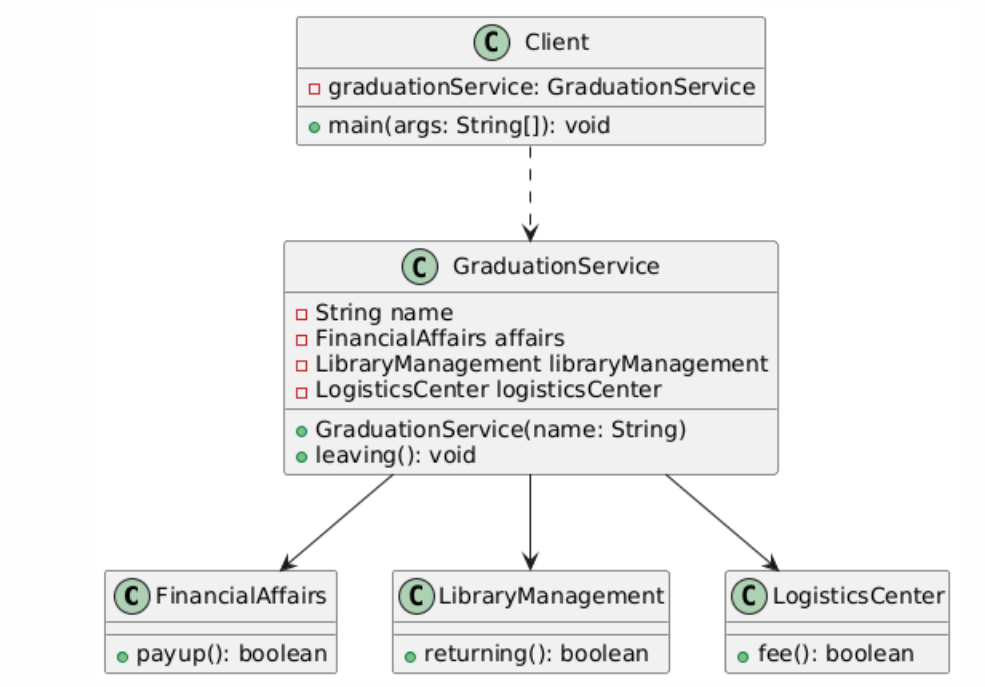
**【实验内容】**

**毕业生在离校前需要办理离校手续，包括在财务处进行费用结算、归还图书馆借阅图书、到后勤服务中心缴纳水电费。其中财务处费用结算由已经由FinancialAffairs类的payup()提供，图书归还由LibraryManagement类的****returning()实现，水电费由LogisticsCenter类的fee()提供。现在开发一个毕业离校系统，为了降低系统的耦合度，方便客户端的调用，请使用外观模型实现上述功能。**

**【实验步骤】**

1. **类图**

**根据下面的代码画出类图。**

****

1. **实现代码**

**(1)子系统角色**

**FinancialAffairs类**

**package** design.facade;

**public** **class** FinancialAffairs {

//财务处进行费用缴清审查

**public** **boolean** payup() {

System.***out***.println("在财务处的进行学费等费用结算！");

**return** **true**;

}

}

**LibraryManagement类**

public class LibraryManagement {  
 public boolean returning() {  
 System.*out*.println("归还图书馆借阅图书！");  
 return true;  
 }  
}

**LogisticsCenter类**

**public class LogisticsCenter {  
 public boolean fee() {  
 System.*out*.println("到后勤服务中心缴纳水电费！");  
 return true;  
 }  
}**

**(2)外观角色**

**GraduationService类**

**package** design.facade;

**public** **class** GraduationService {

**private** String name;

**private FinancialAffairs affairs;**

**private LibraryManagement libraryManagement;**

**private LogisticsCenter logisticsCenter;**

**public GraduationService(String name) {**

**this.name = name;**

**affairs = new FinancialAffairs();**

**libraryManagement = new LibraryManagement();**

**logisticsCenter = new LogisticsCenter();**

**}**

**public void leaving() {**

**System.out.println("------------"+name+"同学正在办理离校手续----------");**

**affairs.payup();**

**libraryManagement.returning();**

**logisticsCenter.fee();**

**}**

}

**(5)测试类（客户端）**

**Test**

**package** design.facade;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

GraduationService graduation = **new** GraduationService("张三");

//办理离校手续

graduation.leaving();

}

**【问题】**

1. **上面的外观类违背了什么原则？如何修改上面的代码解决该问题。**

**开闭原则。因为若是需要增加、删除或更换与外观类交互的子系统类，就必须修改外观类或客户端的代码。新增抽象外观类，更改功能直接新增一个具体外观类，新增配置文件和处理读取配置文件的工具类，其余基本不变，除了让外观具体类继承抽象外观类之外。**

**public abstract class AbstractGraduationService {  
 public abstract void leaving();  
}**

**public class GraduationService extends AbstractGraduationService{**

**//省略  
}**

**<?xml version="1.0"?>  
<config>  
<className>com.fancy.experiment12\_appearance.GraduationService</className>**

**<name>李四</name>  
</config>**

**package com.fancy.experiment12\_appearance;  
  
import org.w3c.dom.\*;  
import javax.xml.parsers.\*;  
import java.io.\*;  
import java.lang.reflect.\*;  
  
public class XMLUtil {  
  
 public static Object getBean() {  
 try {  
 // 创建DOM文档对象  
 DocumentBuilderFactory dFactory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder builder = dFactory.newDocumentBuilder();  
 Document doc;  
 doc = builder.parse(new File("src//com//fancy//experiment12\_appearance//config.xml"));  
  
 // 获取包含类名的文本节点  
 NodeList nl = doc.getElementsByTagName("className");  
 Node classNode = nl.item(0).getFirstChild();  
 String cName = classNode.getNodeValue();  
  
 // 获取学生姓名  
 NodeList nameList = doc.getElementsByTagName("name");  
 Node nameNode = nameList.item(0).getFirstChild();  
 String name = nameNode.getNodeValue();  
  
 // 通过类名生成实例对象并将其返回  
 Class<?> c = Class.*forName*(cName);  
 Constructor<?> constructor = c.getConstructor(String.class); // 获取带有一个 String 参数的构造器  
 Object obj = constructor.newInstance(name); // 传入 name 参数实例化对象  
  
 return obj;  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
}**

**public class Test {  
 public static void main(String[] args) {  
 // 通过 XMLUtil 获取 GraduationService 的实例  
 GraduationService graduation = (GraduationService) XMLUtil.*getBean*();  
  
 if (graduation != null) {  
 // 调用 leaving 方法办理离校手续  
 graduation.leaving();  
 } else {  
 System.*out*.println("未能加载 GraduationService 类");  
 }  
 }  
}**

**【总结】**

|  |  |
| --- | --- |
| 实验总结  （不少于100字） | 本次实验通过实现毕业离校系统，深入理解了外观模式的结构与应用。外观模式通过简化复杂子系统的接口，为客户端提供了一个统一的接口，使得客户端不需要了解各个子系统的细节，能够更加方便地调用系统功能。在实验中，我们使用了 GraduationService 类作为外观类，简化了财务结算、图书归还、水电费用缴纳等多个子系统的操作。通过该实验，掌握了如何使用外观模式封装复杂系统，并且实现了系统的扩展性。比如，在外观类中，我们可以轻松地添加新的子系统或修改现有的实现而不影响客户端的调用，这体现了外观模式的良好扩展性。此外，实验还演示了如何动态加载外观类及其构造参数，通过反射机制和配置文件的结合，提升了系统的灵活性和可维护性。 |
| 学号 | 22154A206 |
| 姓名 | 冯艳芳 |
| 成绩 |  |