

实验报告☑ 实践报告□

课程名称： 软件详细设计R

实验、实践名称： 结构型模式

实验、实践地点： 行勉A105实验室

专业班级： 软件2121班 学号： 2020005731

学生姓名： 张帅

指导教师： 曹若琛

2023年 3 月 28 日

目录

[一、实验目的和要求 3](#_Toc11900)

[二、实验平台 3](#_Toc26479)

[三、主要实验内容及结果 3](#_Toc10770)

[1.适配器模式的运用 3](#_Toc7145)

[1.1 实验内容 3](#_Toc21141)

[1.2 代码实现 4](#_Toc7598)

[1.3 运行结果 6](#_Toc29693)

[2.组合模式的运用 7](#_Toc31600)

[2.1 实验内容 7](#_Toc29285)

[2.2 代码实现 7](#_Toc8265)

[2.3 运行结果 9](#_Toc28227)

[3.外观模式的运用 10](#_Toc3475)

[3.1 实验内容 10](#_Toc9051)

[3.2 代码实现 11](#_Toc6282)

[3.3 运行结果 13](#_Toc8282)

[四、心得体会 13](#_Toc29183)

**一、实验目的和要求**

（1）实验目的

1．综合实例，熟练绘制常见的结构型设计模式结构图。

2. 结合实例，使用 Java 实现常见的结构型设计模式。

3. 通过实验，理解不同结构型设计模式的使用动机，掌握不同结构型设计模式的特点和运用场合，学习如何使用代码实现这些设计模式。

（2）实验要求

1．独立完成实验。

2．书写实验报告书。

**二、实验平台**

使用JDK-11.0.1，IntelliJ IDEA 2020.3.1x64，笔记本电脑。

**三、主要实验内容及结果**

**1.适配器模式的运用**

**1.1 实验内容**

（1）案例背景：

在课堂上我们学习了单向适配器的使用和实现，现在我们需要实现一个双向适配器，编写代码，使用Java 语言实现双向适配器，使猫可以学狗叫，狗可以学猫抓老鼠，请绘制相应类图并实现。（课本 167 页第三题）

（2）实现步骤：

 根据题意，画出双向适配器的类图，类图中应该包含一个适配器类 Adapter；两个抽象类 Cat类和 Dog 类，Cat 类中有发出叫声的方法 cry（）和捉老鼠的方法 catchMouse（），Dog 类中有发出狗叫声的方法 wang（）和动作方法 action（）；两个具体适配者类 ConcreteCat 类和 ConcreteDog 类，两个抽象类互为抽象目标和抽象适配者，如果客户端针对 Cat 类编程，则 Cat 类充当抽象目标，Dog 类充当抽象适配者，ConcreteCat 类充当具体适配者，反之同理。

 根据类图，实现上述类的具体代码以及用户类 Client，由于本题中只有一个适配器类 Adapter，所以不需要通过 XML 文件来改变用户类的操作。

 编译并运行代码，观察是否能让猫发出狗叫声和让狗实现抓老鼠的动作。

（3）案例总结：

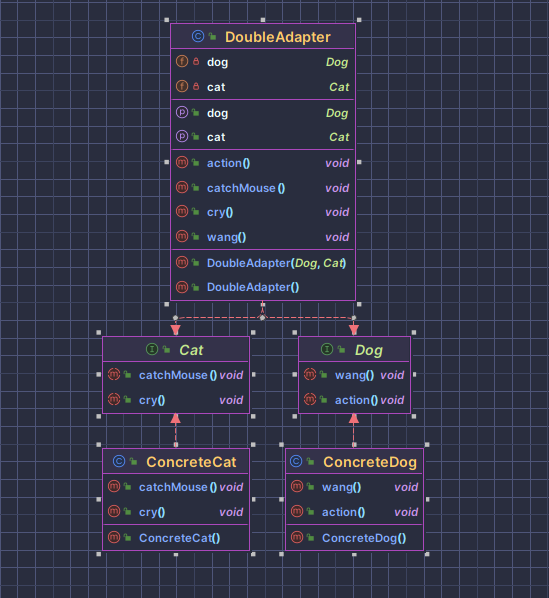
在以下情况下可以使用适配器模式：

 系统需要使用一些现有的类，而这些类的接口丌符合系统的需要，甚至没有这些类的源代码。

 创建一个可以重复使用的类，用于和一些彼此之间没有太大关联的类，包括一些可能在将来引进的类一起工作。

**1.2 代码实现**

绘制类图：



**适配器类 DoubleAdapter：**

package edu.zhshio.adapter.lab.adapter;  
  
import edu.zhshio.adapter.lab.support.Cat;  
import edu.zhshio.adapter.lab.support.Dog;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 上午10:16  
 \*/  
  
public class DoubleAdapter implements Dog, Cat {  
  
 private Dog dog;  
 private Cat cat;  
  
 @Override  
 public void cry() {  
 cat.cry();  
 }  
  
 @Override  
 public void catchMouse() {  
 cat.catchMouse();  
 }  
  
 @Override  
 public void wang() {  
 dog.wang();  
 }  
  
 @Override  
 public void action() {  
 dog.action();  
 }  
  
 public void setDog(Dog dog) {  
 this.dog = dog;  
 }  
  
 public void setCat(Cat cat) {  
 this.cat = cat;  
 }  
  
 public DoubleAdapter(Dog dog, Cat cat) {  
 this.dog = dog;  
 this.cat = cat;  
 }  
  
 public DoubleAdapter() {  
 }  
}

**抽象类 Cat：**

package edu.zhshio.adapter.lab.support;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/13 下午10:15  
 \*/  
  
public interface Cat {  
 void cry();  
 void catchMouse();  
}

**抽象类 Dog：**

package edu.zhshio.adapter.lab.support;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/13 下午10:14  
 \*/  
  
public interface Dog {  
 void wang();  
 void action();  
}

**具体适配者类 ConcreteCat：**

package edu.zhshio.adapter.lab.adaptee;  
  
import edu.zhshio.adapter.lab.support.Cat;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 上午10:14  
 \*/  
  
public class ConcreteCat implements Cat {  
 @Override  
 public void cry() {  
 System.out.println("猫叫");  
 }  
  
 @Override  
 public void catchMouse() {  
 System.out.println("捉老鼠");  
 }  
}

**具体适配者类 ConcreteDog：**

package edu.zhshio.adapter.lab.adaptee;  
  
import edu.zhshio.adapter.lab.support.Dog;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 上午10:14  
 \*/  
  
public class ConcreteDog implements Dog {  
 @Override  
 public void wang() {  
 System.out.println("狗叫");  
 }  
  
 @Override  
 public void action() {  
 System.out.println("狗刨");  
 }  
}

**用户测试类 Client：**

package edu.zhshio.adapter.lab;  
  
import edu.zhshio.adapter.lab.adaptee.ConcreteCat;  
import edu.zhshio.adapter.lab.adaptee.ConcreteDog;  
import edu.zhshio.adapter.lab.adapter.DoubleAdapter;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 上午10:40  
 \*/  
  
public class Client {  
 public static void main(String[] args) {  
 DoubleAdapter doubleAdapter = new DoubleAdapter();  
 doubleAdapter.setCat(new ConcreteCat());  
 doubleAdapter.setDog(new ConcreteDog());  
  
 doubleAdapter.cry();  
 doubleAdapter.catchMouse();  
 doubleAdapter.wang();  
 doubleAdapter.action();  
 }  
}

**1.3 运行结果**



**2.组合模式的运用**

**2.1 实验内容**

（1）案例背景：

在操作系统中，一个文件夹中可能存放着图像文件，视频文件，文本文件，也可能存放其他的文件夹，而对不同类型的文件进行的浏览操作也不一样，使用透明组合模式，绘制类图并编程实现文件的浏览（课本 197 页第二题）。

（2）实现步骤：

 根据题意，画出组合模式的类图，类图中应包含抽象文件类 AbstractFile，具体的图像文件类ImageFile，视频文件类 VideoFile，文本文件类 TextFile 以及文件夹类 Folder，对每个文件都有display（）方法，而对文件夹可以进行 add（）方法和 remove（）方法。

 根据类图，实现上述类的具体代码以及用户类 Client，在用户类中需要将不同类型的文件放入文件夹中。

 编译并运行程序，使程序能够输出对文件的浏览过程。

（4）案例总结：软件详细设计 R 实验指导书在以下情况可以使用组合模式：

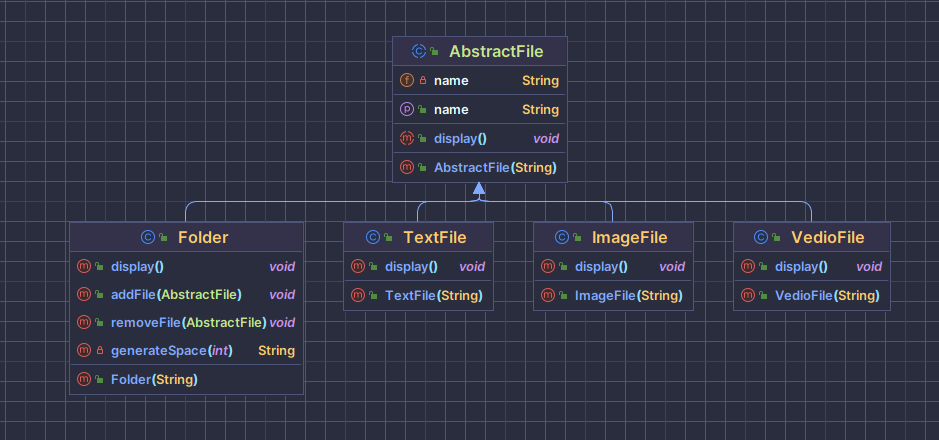
 在具有整体和部分的层次结构中，希望通过一种方式忽略整体与部分的差异，使客户端可以一致的对待他们。

 在使用面向对象语言开发的系统中需要处理一个树形结构。

 在一个系统中能够分离出叶子对象和容器对象，而且他们的类型不固定，需要增加一些新的类型。

**2.2 代码实现**

绘制类图：



**抽象文件类 AbstractFile：**

package edu.zhshio.composite.lab.support;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 上午10:48  
 \*/  
  
public abstract class AbstractFile {  
  
 private String name;  
  
 public abstract void display();  
  
 public String getName() {  
 return this.name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public AbstractFile(String name) {  
 this.name = name;  
 }  
}

**具体的图像文件类ImageFile：**

package edu.zhshio.composite.lab.folder;  
  
import edu.zhshio.composite.lab.support.AbstractFile;  
  
import java.util.\*;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 上午11:02  
 \*/  
  
public class Folder extends AbstractFile {  
  
 private Map<String, AbstractFile> subFiles = new TreeMap<>();  
  
 @Override  
 public void display() {  
 String folderName = getName();  
 System.out.println(folderName);  
 String spaces = generateSpace(folderName.length());  
 subFiles.values()  
 .stream()  
 .forEach(  
 file ->{  
 if (file instanceof Folder) {  
 System.out.print(spaces + "|");  
 System.out.print("--");  
 file.display();  
 } else {  
 System.out.println(spaces + "|");  
 System.out.println(spaces + "--" + file.getName());  
 }  
 }  
 );  
 }  
  
 public void addFile(AbstractFile file) {  
 if (subFiles.containsKey(file.getName())) {  
 throw new RuntimeException("不可在同一文件夹中创建相同名字的文件");  
 }  
 subFiles.put(file.getName(), file);  
 }  
  
 public void removeFile(AbstractFile file) {  
 subFiles.remove(file.getName());  
 }  
  
 private String generateSpace(int length) {  
 StringBuilder sb = new StringBuilder();  
 while (length-- > 0) {  
 sb.append(" ");  
 }  
 return sb.toString();  
 }  
  
 public Folder(String name) {  
 super(name);  
 }  
  
}

**具体的视频文件类 VideoFile：**

package edu.zhshio.composite.lab.files;  
  
import edu.zhshio.composite.lab.support.AbstractFile;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 上午10:58  
 \*/  
  
public class VedioFile extends AbstractFile {  
  
 @Override  
 public void display() {  
 System.out.println("播放视频");  
 }  
  
 public VedioFile(String name) {  
 super(name);  
 }  
  
}

**具体的文本文件类 TextFile：**

package edu.zhshio.composite.lab.files;  
  
import edu.zhshio.composite.lab.support.AbstractFile;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 上午11:01  
 \*/  
  
public class TextFile extends AbstractFile {  
  
 @Override  
 public void display() {  
 System.out.println("展示文本");  
 }  
  
 public TextFile(String name) {  
 super(name);  
 }  
}

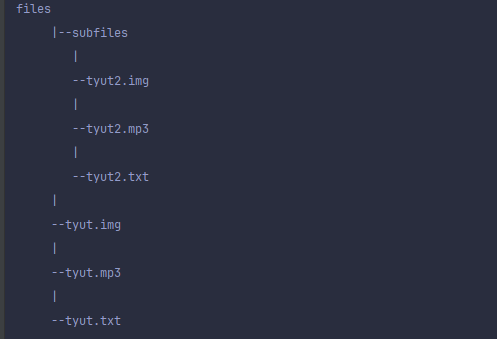
**具体的文件夹类 Folder：**

package edu.zhshio.composite.lab.folder;  
  
import edu.zhshio.composite.lab.support.AbstractFile;  
  
import java.util.\*;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 上午11:02  
 \*/  
  
public class Folder extends AbstractFile {  
  
 private Map<String, AbstractFile> subFiles = new TreeMap<>();  
  
 @Override  
 public void display() {  
 String folderName = getName();  
 System.out.println(folderName);  
 String spaces = generateSpace(folderName.length());  
 subFiles.values()  
 .stream()  
 .forEach(  
 file ->{  
 if (file instanceof Folder) {  
 System.out.print(spaces + "|");  
 System.out.print("--");  
 file.display();  
 } else {  
 System.out.println(spaces + "|");  
 System.out.println(spaces + "--" + file.getName());  
 }  
 }  
 );  
 }  
  
 public void addFile(AbstractFile file) {  
 if (subFiles.containsKey(file.getName())) {  
 throw new RuntimeException("不可在同一文件夹中创建相同名字的文件");  
 }  
 subFiles.put(file.getName(), file);  
 }  
  
 public void removeFile(AbstractFile file) {  
 subFiles.remove(file.getName());  
 }  
  
 private String generateSpace(int length) {  
 StringBuilder sb = new StringBuilder();  
 while (length-- > 0) {  
 sb.append(" ");  
 }  
 return sb.toString();  
 }  
  
 public Folder(String name) {  
 super(name);  
 }  
  
}

**用户测试类 Client：**

package edu.zhshio.composite.lab;  
  
import edu.zhshio.composite.lab.files.ImageFile;  
import edu.zhshio.composite.lab.files.TextFile;  
import edu.zhshio.composite.lab.files.VedioFile;  
import edu.zhshio.composite.lab.folder.Folder;  
  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午5:14  
 \*/  
  
public class Client {  
 public static void main(String[] args) {  
 Folder folder = new Folder("files");  
 Folder subfolder = new Folder("subfiles");  
  
 ImageFile image = new ImageFile("tyut.img");  
 VedioFile vedio = new VedioFile("tyut.mp3");  
 TextFile text = new TextFile("tyut.txt");  
  
 ImageFile image2 = new ImageFile("tyut2.img");  
 VedioFile vedio2 = new VedioFile("tyut2.mp3");  
 TextFile text2 = new TextFile("tyut2.txt");  
  
 subfolder.addFile(image2);  
 subfolder.addFile(vedio2);  
 subfolder.addFile(text2);  
  
 folder.addFile(subfolder);  
 folder.addFile(image);  
 folder.addFile(vedio);  
 folder.addFile(text);  
  
 folder.display();  
  
 }  
}

**2.3 运行结果**



**3.外观模式的运用**

**3.1 实验内容**

（1）案例背景：

在计算机主机（Mainframe）中，只需要按下主机的开机按钮（on（）），即可调用其他硬件设备和软件的启动方法，如内存（Memory）的自检（check（））、CPU 的运行（run（））、硬盘的（HardDisk）的读取（read（））、操作系统（OS）的载入（load（））等，如果某一过程发生错误则计算机启动失败。使用外观模式模拟该过程，绘制类图并编程实现。（课本 230 页第二题）

（2）实现步骤：

 根据题意，画出外观模式的类图，使主机类 Mainframe 充当外观角色，内存类 Memory，CPU 类 CPU，硬盘类 HardDisk 和操作系统类 OS 充当子系统角色。

 根据类图，编写并实现代码。

 编译并运行代码，使代码能够输出模拟出来的电脑开机过程。

（3）案例总结：

在以下情况下可以使用外观模式：

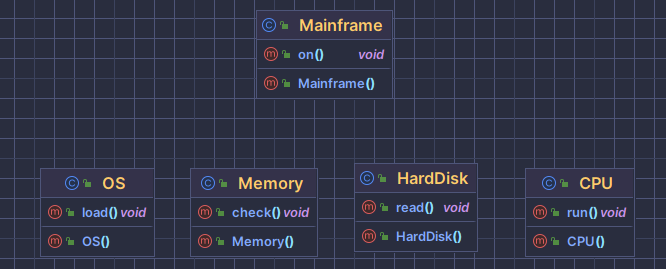
 当要为一个复杂子系统提供一个简单接口的时候可以使用外观模式，该接口可以满足大部分用户需求，用户也可以越过外观类直接访问子系统。

 客户程序和多个子系统之间存在很大的依赖性。引入外观类将子系统与客户以及其他子系统解耦，可以提高子系统的独立性和可移植性。

 在层次化结构中，可以使用外观模式定义系统中每一层的入口，层与层之间不直接产生联系，而是通过外观类建立联系，降低层之间的耦合度。

**3.2 代码实现**

绘制类图：



内存类 Memory：

package edu.zhshio.facade.lab.accessories;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午5:54  
 \*/  
  
public class Memory {  
 public void check() {  
 System.out.println("Memory is performing self-check...");  
 }  
}

CPU 类 CPU：

package edu.zhshio.facade.lab.accessories;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午5:52  
 \*/  
  
public class CPU {  
 public void run() {  
 System.out.println("CPU is running...");  
 }  
}

硬盘类 HardDisk：

package edu.zhshio.facade.lab.accessories;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午5:53  
 \*/  
  
public class HardDisk {  
 public void read() {  
 System.out.println("Hard disk is reading data...");  
 }  
}

操作系统类 OS：

package edu.zhshio.facade.lab.accessories;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午5:54  
 \*/  
  
public class OS {  
 public void load() {  
 System.out.println("Operating system is loading...");  
 }  
}

主机类 Mainframe：

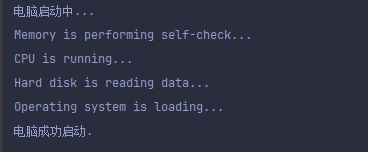
package edu.zhshio.facade.lab.facade;  
  
import edu.zhshio.facade.lab.accessories.CPU;  
import edu.zhshio.facade.lab.accessories.HardDisk;  
import edu.zhshio.facade.lab.accessories.Memory;  
import edu.zhshio.facade.lab.accessories.OS;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午5:54  
 \*/  
  
  
public class Mainframe {  
  
 private CPU cpu;  
  
 private HardDisk hardDisk;  
  
 private Memory memory;  
  
 private OS os;  
  
 public Mainframe() {  
 this.cpu = new CPU();  
 this.hardDisk = new HardDisk();  
 this.memory = new Memory();  
 this.os = new OS();  
 }  
  
 public void on() {  
 System.out.println("电脑启动中...");  
try {  
 memory.check();  
 cpu.run();  
 hardDisk.read();  
 os.load();  
 System.out.println("电脑成功启动.");  
} catch (Exception e) {  
 System.out.println("电脑成功失败.");  
 throw new RuntimeException(e);  
}

}  
  
}

用户测试类 Client：

package edu.zhshio.facade.lab;  
  
import com.sun.tools.javac.Main;  
import edu.zhshio.facade.lab.facade.Mainframe;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午5:57  
 \*/  
  
public class Client {  
 public static void main(String[] args) {  
 Mainframe mainframe = new Mainframe();  
 mainframe.on();  
 }  
}

**3.3 运行结果**



**四、心得体会**

本次做的是结构型模式的实验，实验内容较为基础，参考课本上的例题，独立完成没有遇到什么问题。

适配器模式的运用，课本上的案例是让机器人模仿其他动物叫，单方面地适配也就是单向适配，而这个实验是让我们双向适配，就是说猫可以学狗的一些行为，狗也可以学习猫的一些行为，它们之间是互相学习的，两个抽象类互为抽象目标和抽象适配者。在适配器类中new一个小猫对象和一个小狗对象，并重写它们的所有抽象方法，互相学习的过程就是互相调用对方方法的过程。

组合模式就像课本上的盘子和水果的关系，又或本次实验里的文件夹和文件的关系，一个文件夹里可以有很多文件，也可以有更多的文件夹，访问的时候自然就用到递归进行遍历，俗称套娃模式，一层一层地执行，到最后剩下的就是文件不能再分解，而文件就是我们最终想要的结果，这里一个文件夹里面的所有内容我们是用集合来接收的，以便后续遍历。

最后就是外观模式。外观模式课本上的例子是一个开关和一堆电器直接产生的联系，按下开关可控制全部的电器。这里是主机和软、硬件的控制关系，题目要求，打开主机，主机上的所有软、硬件将全部打开，所以在主机类中，我定义了一个on方法，在该方法里将调用所有设备的功能方法，测试类中直接调用主机的on方法就可以开启所有的软件和硬件设备，当然，题目要求，只要这其中任何一个过程发生错误，计算机将启动失败，所以我用了try/catch 去进行异常抛出，异常输出内容为，计算机启动失败。