

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：实验二 结构型设计模式实验

实验、实践地点： 软件学院C2实验室

专业班级： 软件2028班 学号： 2020007657

学生姓名： 常兆海

指导教师： 曹锐

2023年 3月 28日

目录

[1 实验目的和要求： 3](#_Toc131972235)

[1.1 目的： 3](#_Toc131972236)

[1.2 要求： 3](#_Toc131972237)

[2 实验平台: 4](#_Toc131972238)

[3 主要实验内容及结果: 5](#_Toc131972239)

[3.1 实验内容 5](#_Toc131972240)

[3.1.1 适配器模式的运用 5](#_Toc131972241)

[3.1.2 组合模式的运用 9](#_Toc131972242)

[3.1.3 外观模式的运用 13](#_Toc131972243)

[3.2 心得体会 18](#_Toc131972244)

# 实验目的和要求：

## 目的：

* 综合实例，熟练绘制常见的结构型设计模式结构图。
* 结合实例，使用 Java 实现常见的结构型设计模式。
* 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。

## 要求：

* 独立完成实验
* 书写实验报告书

# 实验平台:

JDK-1.8 idea—2022.3

# 主要实验内容及结果:

## 实验内容

### 适配器模式的运用

#### 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）

#### 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。

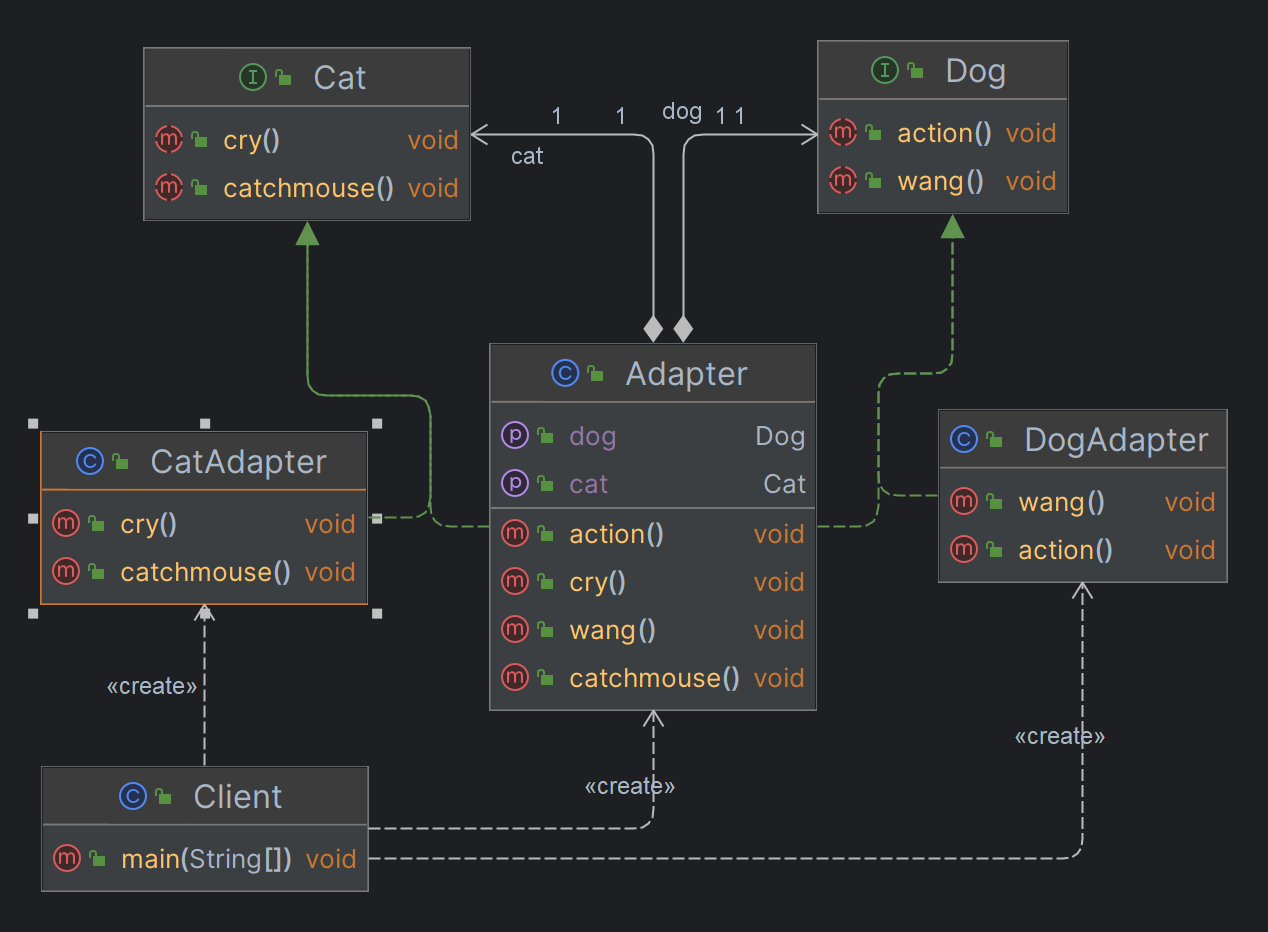
#### 案例总结：

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

#### 结果：

##### 类图



##### 代码实现

* Adapter类

public class Adapter implements Cat, Dog{  
 private Cat cat;  
 private Dog dog;  
 public void setCat(Cat cat) {  
 this.cat = cat;  
 }  
 public void setDog(Dog dog) {  
 this.dog = dog;  
 }  
 @Override  
 public void cry() {  
 System.out.print("猫学");  
dog.wang();  
 }  
 @Override  
 public void catchmouse() {  
cat.catchmouse();  
 }  
 @Override  
 public void wang() {  
dog.wang();  
 }  
 @Override  
 public void action() {  
 System.out.print("狗学");  
 cat.catchmouse();  
 }  
}

* cat接口

public interface Cat {  
 void cry();  
 void catchmouse();  
}

* dog接口

public interface Dog {  
 void wang();  
 void action();  
}

* CatAdapter类

public class CatAdapter implements Cat{  
 @Override  
 public void cry() {  
 System.out.println("常兆海·s Cat cry");  
 }  
 @Override  
 public void catchmouse() {  
 System.out.println("常兆海·s Cat catchmouse");  
 }  
}

* DogAdapter类

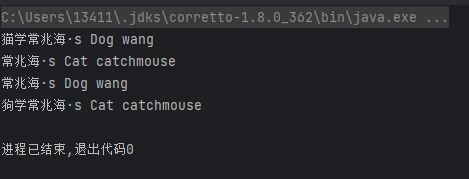
public class DogAdapter implements Dog{  
 @Override  
 public void action() {  
 System.out.println("常兆海·s Dog action");  
 }  
 @Override  
 public void wang() {  
 System.out.println("常兆海·s Dog wang");  
 }  
}

Client类

public class Client {  
 public static void main(String[] args) {  
 //必须要把猫狗都实例化出来 不然会报空指针异常！！！  
 Cat cat = new CatAdapter();  
 Dog dog = new DogAdapter();  
 Adapter adapter = new Adapter();  
 adapter.setCat(cat);  
 cat = adapter;  
 adapter.setDog(dog);  
 cat.cry();  
 cat.catchmouse();  
 dog = adapter;  
 dog.wang();  
 dog.action();  
 }  
}

##### 运行结果

猫学常兆海·s Dog wang  
常兆海·s Cat catchmouse  
常兆海·s Dog wang  
狗学常兆海·s Cat catchmouse  
  
进程已结束,退出代码0



### 组合模式的运用

#### 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）。

#### 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。

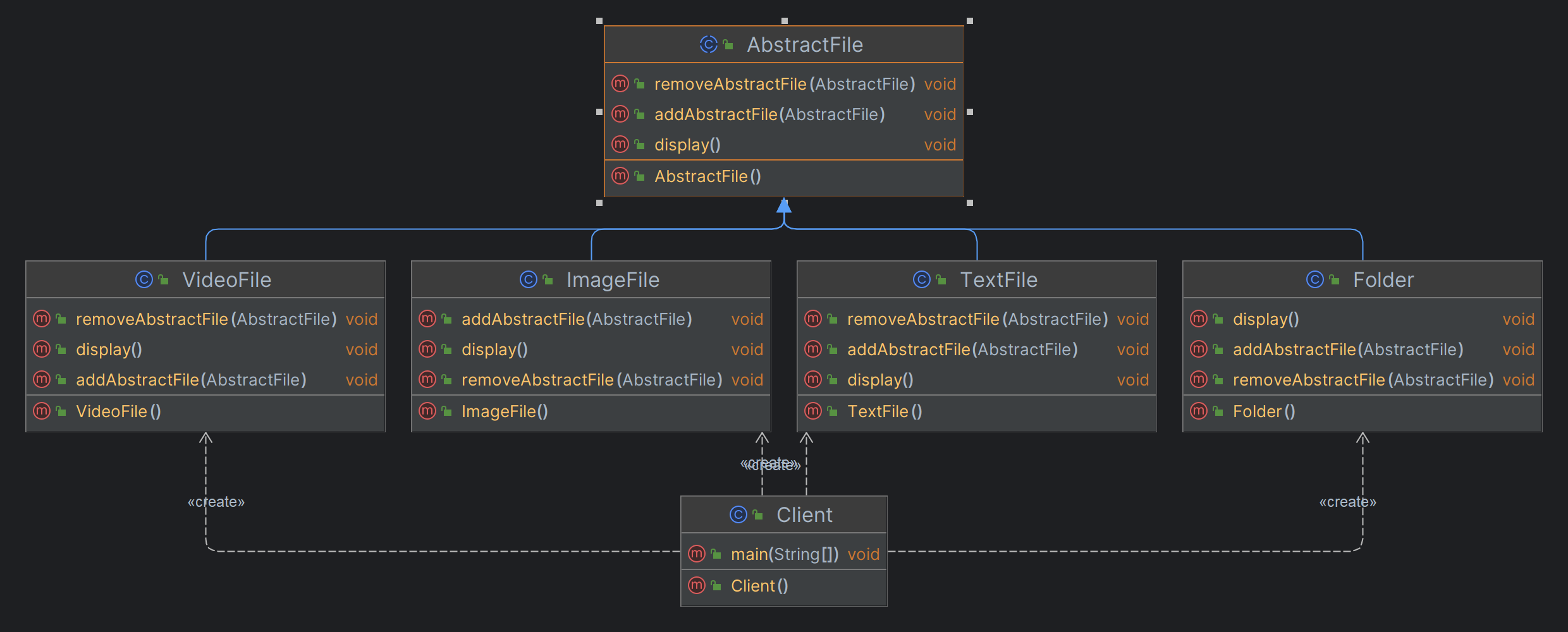
#### 案例总结：

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

#### 结果：

##### 类图



##### 代码实现

* AbstractFile类

public abstract class AbstractFile {  
 public void addAbstractFile(AbstractFile abstractFile){}  
 public void removeAbstractFile(AbstractFile abstractFile){}  
 public void display(){}  
}

* Folder类

public class Folder extends AbstractFile{  
 private ArrayList arrayList=new ArrayList();  
 @Override  
 public void addAbstractFile(AbstractFile abstractFile) {  
 arrayList.add(abstractFile);  
 }  
 @Override  
 public void removeAbstractFile(AbstractFile abstractFile) {  
 arrayList.remove(abstractFile);  
 }  
 @Override  
 public void display() {  
 for (Object o:arrayList  
 ) {  
  
 ((AbstractFile)o).display();  
  
 }  
 }  
}

* ImageFile类

public class ImageFile extends AbstractFile{  
 @Override  
 public void addAbstractFile(AbstractFile abstractFile) {  
 System.out.println("ImageFile add Error");  
 }  
  
 @Override  
 public void removeAbstractFile(AbstractFile abstractFile) {  
 System.out.println("ImageFile remove Error");  
 }  
  
 @Override  
 public void display() {  
 System.out.println("常兆海ImageFile display！！");  
 }  
}

* TextFile 类

public class TextFile extends AbstractFile{  
 @Override  
 public void addAbstractFile(AbstractFile abstractFile) {  
 System.out.println("TextFile add Error");  
 }  
  
 @Override  
 public void removeAbstractFile(AbstractFile abstractFile) {  
 System.out.println("TextFile remove error");  
 }  
  
 @Override  
 public void display() {  
 System.out.println("2028班 TextFile display!!");  
 }  
}

* VideoFile类

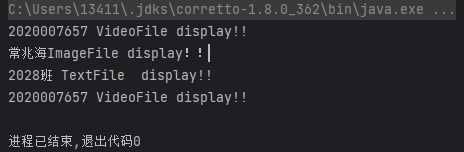
public class VideoFile extends AbstractFile{  
 @Override  
 public void addAbstractFile(AbstractFile abstractFile) {  
 System.out.println(" VideoFile add Error");  
 }  
  
 @Override  
 public void removeAbstractFile(AbstractFile abstractFile) {  
 System.out.println(" VideoFile remove Error");  
 }  
  
 @Override  
 public void display() {  
 System.out.println("2020007657 VideoFile display!!");  
 }  
}

* 客户端类

public class Client {  
 public static void main(String[] args) {  
 AbstractFile file1,file2,file3,file4,folder1,folder2,folder3;  
 file1=new ImageFile();  
 file2=new VideoFile();  
 file3=new TextFile();  
 file4=new VideoFile();  
 folder1=new Folder();  
 folder2=new Folder();  
 folder3=new Folder();  
 folder1.addAbstractFile(folder2);  
 folder1.addAbstractFile(file1);  
 folder1.addAbstractFile(folder3);  
 folder2.addAbstractFile(file2);  
 folder3.addAbstractFile(file3);  
 folder3.addAbstractFile(file4);  
folder1.display();  
 }  
}

##### 运行结果

2020007657 VideoFile display!!  
常兆海ImageFile display！！  
2028班 TextFile display!!  
2020007657 VideoFile display!!  
  
进程已结束,退出代码0



### 外观模式的运用

#### 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）

#### 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

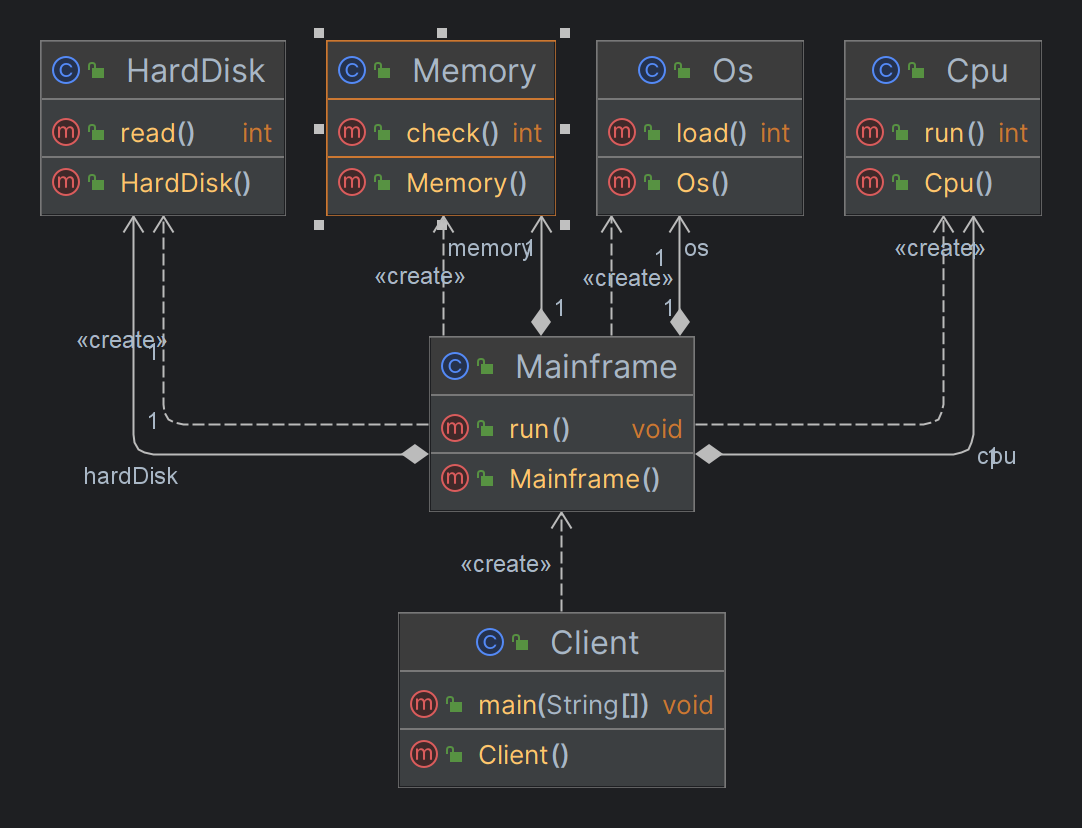
#### 案例总结：

#### 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

#### 结果：

##### 类图



##### 代码实现

* cpu类

public class Cpu {  
 public int run(){  
// return ((int)(10 \* Math.random())) % 2;//0,1随机数  
Random a=new Random();  
 int n6 = a.nextInt(20);  
 int m1 = 9;  
 if(n6 < 5){  
 m1 = 0;  
 }else{  
 m1 = 1;  
 }  
 System.out.println("Cpu"+m1);  
 return m1;  
 }  
}

* HardDisk类

public class HardDisk {  
 public int read(){  
 //0,1随机数  
// return ((int)(10 \* Math.random())) % 2;//0,1  
 //0,1概率数  
 Random a=new Random();  
 int n6 = a.nextInt(20);  
 int m1 = 9;  
 if(n6 < 4){  
 m1 = 0;  
 }else {  
 m1 = 1;  
 }  
 System.out.println("HardDisk"+m1);  
 return m1;  
 }  
}

* Mainframe类

public class Mainframe {  
 private Cpu cpu=new Cpu();  
 private Memory memory=new Memory();  
 private HardDisk hardDisk=new HardDisk();  
 private Os os=new Os();  
 public void run(){  
 if (cpu.run()==1&&memory.check()==1&&hardDisk.read()==1&&os.load()==1) {  
 System.out.println("运行正常");  
 }else {  
 System.out.println("运行异常");  
 }  
 }  
}

* Memory类

public class Memory {  
 public int check(){  
 //0,1随机数  
// return ((int)(10 \* Math.random())) % 2;//0,1  
 Random a=new Random();  
 int n6 = a.nextInt(20);  
 int m1 = 9;  
 if(n6 < 4){  
 m1 = 0;  
 }else {  
 m1 = 1;  
 }  
 System.out.println("MEmory"+m1);  
 return m1;  
 }  
}

* Os类

public class Os {  
 public int load() {  
 //0,1随机数  
// return ((int) (10 \* Math.random())) % 2;//0,1  
  
 Random a=new Random();  
 int n6 = a.nextInt(20);  
 int m1 = 9;  
 if(n6 < 5){  
 m1 = 0;  
 }else {  
 m1 = 1;  
 }  
 System.out.println("os"+m1);  
 return m1;  
 }  
}

* 客户端类

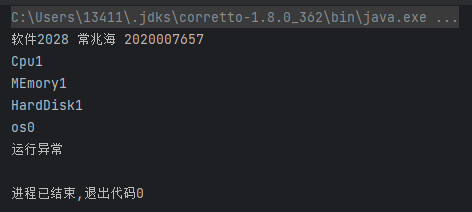
public class Client  
{  
 public static void main(String[] args) {  
 Mainframe mainframe=new Mainframe();  
 System.out.println("软件2028 常兆海 2020007657");  
 mainframe.run();  
 }  
}

##### 运行结果

失败：

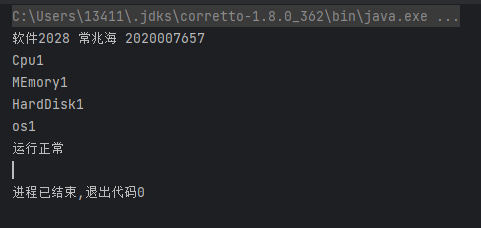
软件2028 常兆海 2020007657  
Cpu1  
MEmory1  
HardDisk1  
os0  
运行异常

进程已结束,退出代码0



成功：

软件2028 常兆海 2020007657  
Cpu1  
MEmory1  
HardDisk1  
os1  
运行正常



## 心得体会

设计模式是前人结合实践总结出来的一套行之有效的理论，能够将实际业务需求转换为技术实现时，使系统更具有维护性、扩展性。在学习设计模式时，我不禁赞叹它巧妙的思路和优雅的方式。通过本此实验的学习，更好的理解了结构型设计模式，将理论学习中出现的不足在本次实践中得到了很好的补充。