

|  |
| --- |
| 实验报告  成绩（满分100） |
|  |

**《软件设计》实验报告**

**2019-2020学年第一学期**

姓名：刘文博

学号：201716040224

专业：软件工程

班级：1702班

2019年10月

**实验项目3．结构型模式实验（6学时）**

**1．实验内容**

使用结构型模式设计软件系统。

**2．基本要求**

（1）熟悉7种结构型模式。

（2）使用结构型模式设计软件

**3．支撑的课程目标**

本实验项目可以支撑“课程目标2：能够在软件项目开发过程中,培养学生的创新精神”和“课程目标3：能够运用设计模式设计软件系统，支持复杂工程问题的求解”。

本实验使学生能分析软件系统，在软件项目的设计过程中使用结构型模式来设计软件，达到课程目标的要求。

**4.实验题目**

**1.**实现一个双向适配器实力,似的猫(Cat)可以学狗(Dog)叫,狗可以学猫抓老鼠,绘制相应类图并使用代码编程模拟.

**2.** 一个电器工厂可以产生多种类型的电器，如海尔工厂可以生产海尔电视机、海尔空调等，TCL工厂可以生产TCL电视机、TCL空调等，相同品牌的电器构成一个产品族，而相同类型的电器构成了一个产品等级结构，模拟该场景，设计类图并编程实现

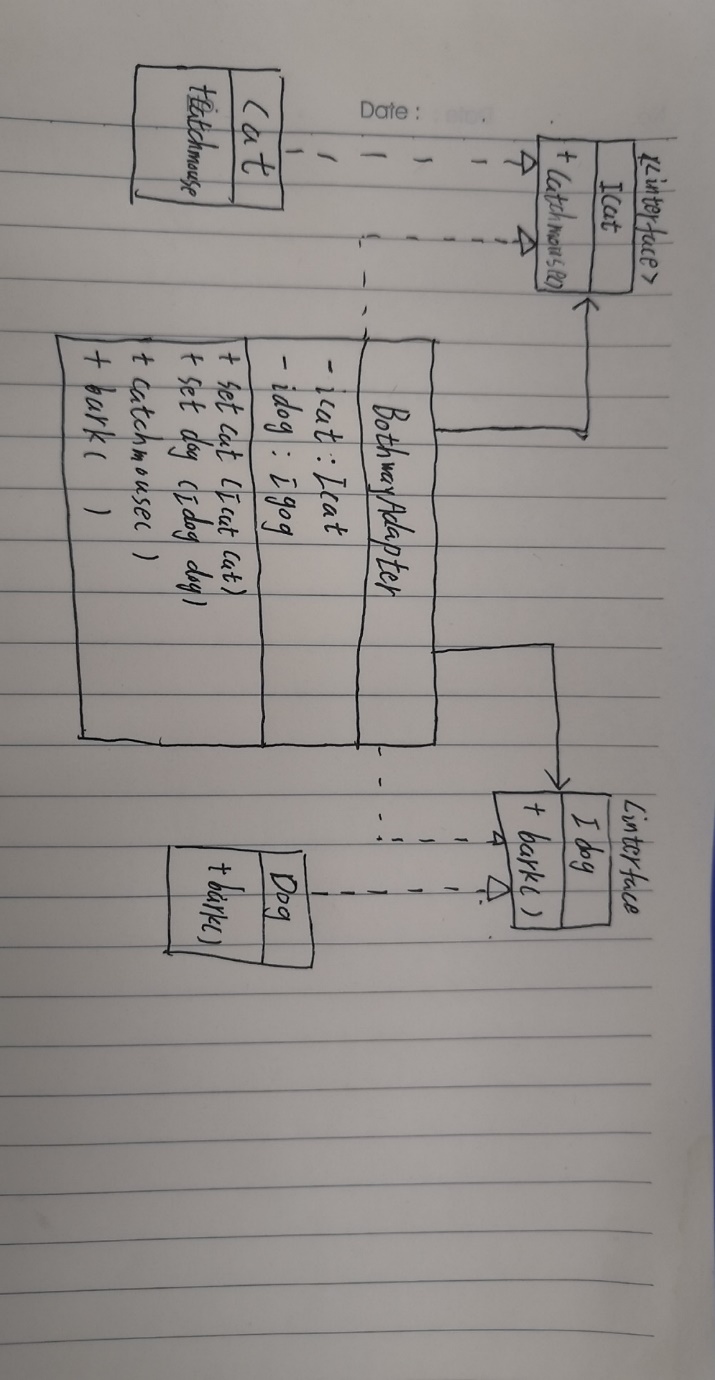
**3.** 在计算机主机（MainFrame）中，只需要按下主机的开机按钮(on())，就可以调用其他硬件设备和软件的启动方法，如内存(Memory)的自检(check())，CPU的运行(run())，硬盘(Harddisk)的读取(read())，操作系统(OS)的载入(load())等，如果某一过程发生错误，则计算机启动失败，使用外观模式模拟该过程绘制类图并编程实现。

（1）实现给出外观模式结构视图。

（2）给出该实例类图及代码实现。

**5.实验方法、实验过程及实验分析**

**1.类图**



**抽象类:**

public interface ICat {

void catlook();

void catchmouse();

}

public interface IDog {

void look();

void bark();

}

**实体类:**

public class Cat implements ICat {

@Override

public void catlook() {

System.out.println("我是一只猫!");

}

@Override

public void catchmouse() {

System.out.println("我会抓老鼠!");

}

}

public class Dog implements IDog {

@Override

public void look() {

System.out.println("我是一只狗!");

}

@Override

public void bark() {

System.out.println("汪汪汪!!!!");

}

}

**适配器:**

public class TwowayAdapter implements ICat, IDog {

ICat cat ;

IDog dog ;

public TwowayAdapter(ICat cat) {

this.cat = cat;

}

public TwowayAdapter(IDog dog) {

this.dog = dog;

}

@Override

public void bark() {

cat.catchmouse();

}

@Override

public void catchmouse() {

dog.bark();

}

@Override

public void look() {

System.out.println("我是一只狗!");

}

@Override

public void catlook() {

System.out.println("我是一只猫!");

}

}

**测试类:**

public class Client {

public static void main(String[] args) {

Cat cat = new Cat();

Dog dog = new Dog();

cat.catlook();

cat.catchmouse();

dog.look();

dog.bark();

System.out.println("\*\*\*\*\*\*\*\*\*添加适配器\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

ICat cat1 = new TwowayAdapter(dog);

cat1.catlook();

cat1.catchmouse();

IDog dog1 = new TwowayAdapter(cat);

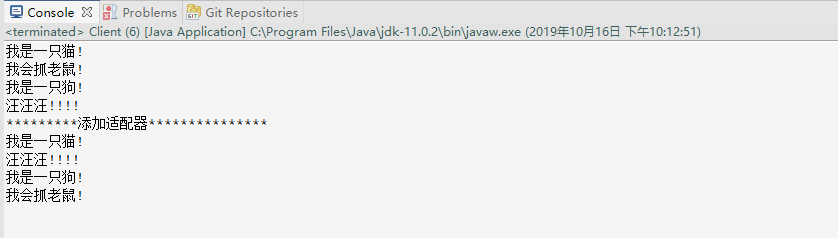
dog1.look();

dog1.bark();

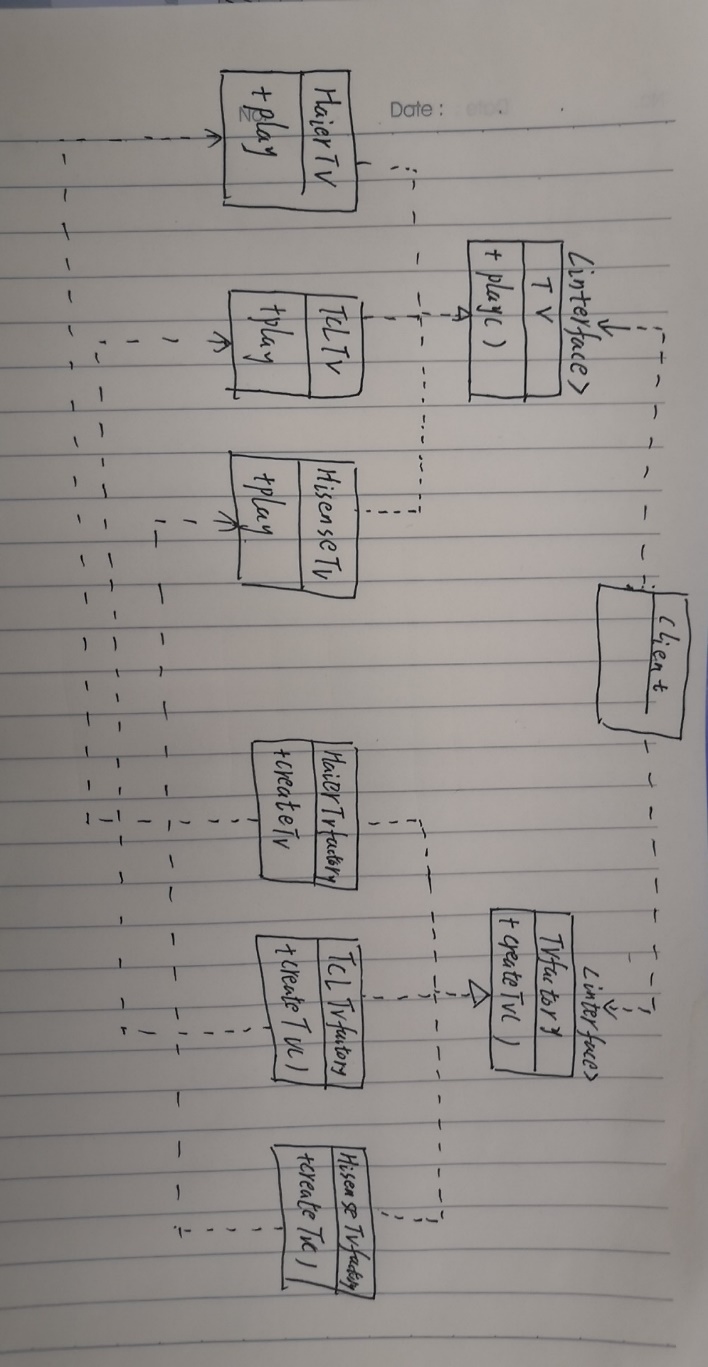
}

}

**运行结果:**



**2.类图**



**抽象产品类:**

public interface Television {

public void play();

}

public interface AirConditioner {

public void changeTemperature();

}

**抽象工厂类:**

public interface EFactory {

public Television produceTelevision();

public AirConditioner produceAirConditioner();

}

**具体产品类:**

public class HaierTelevision implements Television {

public void play() {

System.out.println("海尔电视机播放中.");

}

}

public class TCLTelevision implements Television {

public void play() {

System.out.println("TCL电视机播放中.");

}

}

public class HaierAirConditioner implements AirConditioner {

public void changeTemperature() {

System.out.println("海尔空调控温中");

}

}

public class HaierAirConditioner implements AirConditioner {

public void changeTemperature() {

System.out.println("TCL空调控温中");

}

}

**具体工厂类:**

public class HaierFactory implements EFactory {

public Television produceTelevison() {

return new HaierTelevision();

}

public AirConditioner produceAirConditioner {

return new HaierAirConditioner();

}

}

public class TCLFactory implements EFactory {

public Television produceTelevison() {

return new TCLTelevision();

}

public AirConditioner produceAirConditioner {

return new TCLAirConditioner();

}

}

**测试代码:**

public class Client {

public static void main(String[] args) {

Television tv;

AirConditioner airConditioner;

EFactory factory = new HaierFactory();

tv = factory.produceTelevision();

airConditioner = factory.produceAirConditioner();

tv.play();

airConditioner.changeTemperature();

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

factory = new TCLFactory();

tv = factory.produceTelevision();

airConditioner = factory.produceAirConditioner();

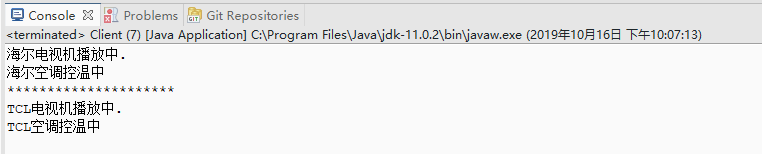
tv.play();

airConditioner.changeTemperature();

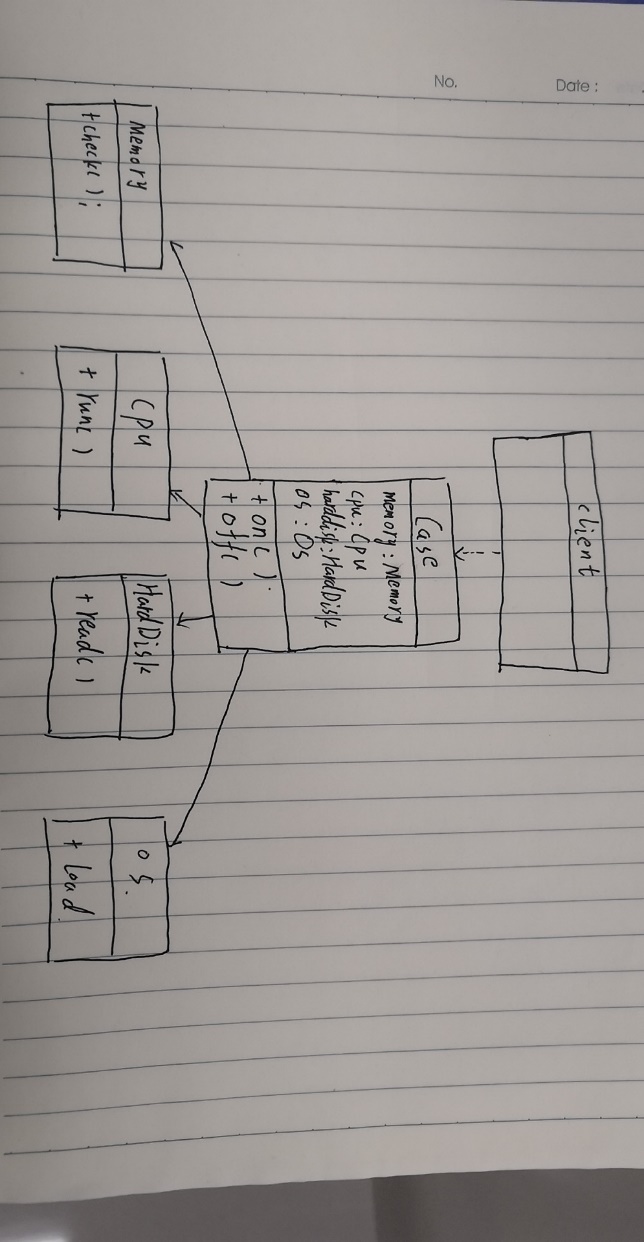
}

}

**实验结果:**



**3.类图**



**实体类:**

public class CPU {

void run() {

System.out.println("CPU开始运行......");

}

}

public class HardDisk {

void read() {

System.out.println("开始从硬盘读取数据......");

}

}

public class Memory {

void check() {

System.out.println("内存开始自检......");

}

}

public class OS {

void load() {

System.out.println("系统开始加载......");

}

}

public class Mainframe {

CPU cpu = new CPU();

Memory memory = new Memory();

OS os = new OS();

HardDisk harddisk = new HardDisk();

public void on() {

cpu.run();

memory.check();

harddisk.read();

os.load();

System.out.println("系统已经启动");

}

public void off() {

System.out.println("系统已经关闭");

}

}

**测试类:**

public class Client {

public static void main(String[] args) {

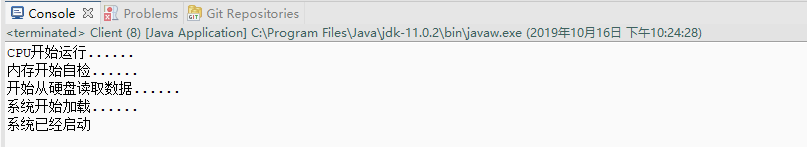
Mainframe mainfram = new Mainframe();

mainfram.on();

}

}

**实验结果:**



**6．实验总结**

对象适配器和类适配器其实算是同一种思想，只不过实现方式不同。 根据合成复用原则，组合大于继承， 所以它解决了类适配器必须继承src的局限性问题，也不再强求dst必须是接口。 同样的它使用成本更低，更灵活。

抽象工厂模式最大的好处是易于交换产品系列，由于具体工厂类， 在一个应用中只需要在初始化的时候出现一次，这就使得改变一个应用的具体工厂变得非常容易，它只需要改变具体工厂即可使用不同的产品配置。不管是任何人的设计都无法去完全防止需求的更改，或者项目的维护，那么我们的理想便是让改动变得最小、最容易，例如我现在要更改以上代码的数据库访问时，只需要更改具体的工厂即可。

抽象工厂模式的另一个好处就是它让具体的创建实例过程与客户端分离，客户端是通过它们的抽象接口操作实例，产品实现类的具体类名也被具体的工厂实现类分离，不会出现在客户端代码中。就像我们上面的例子，客户端只认识IUser和IDepartment，至于它是MySQl里的表还是Oracle里的表就不知道了。

通过引入外观模式，外观类，客户端代码只和外观类直接通信，代码变得简单。实现了子系统与客户之间的松耦合关系。客户端依然可以直接访问子系统。但是违反了开闭原则