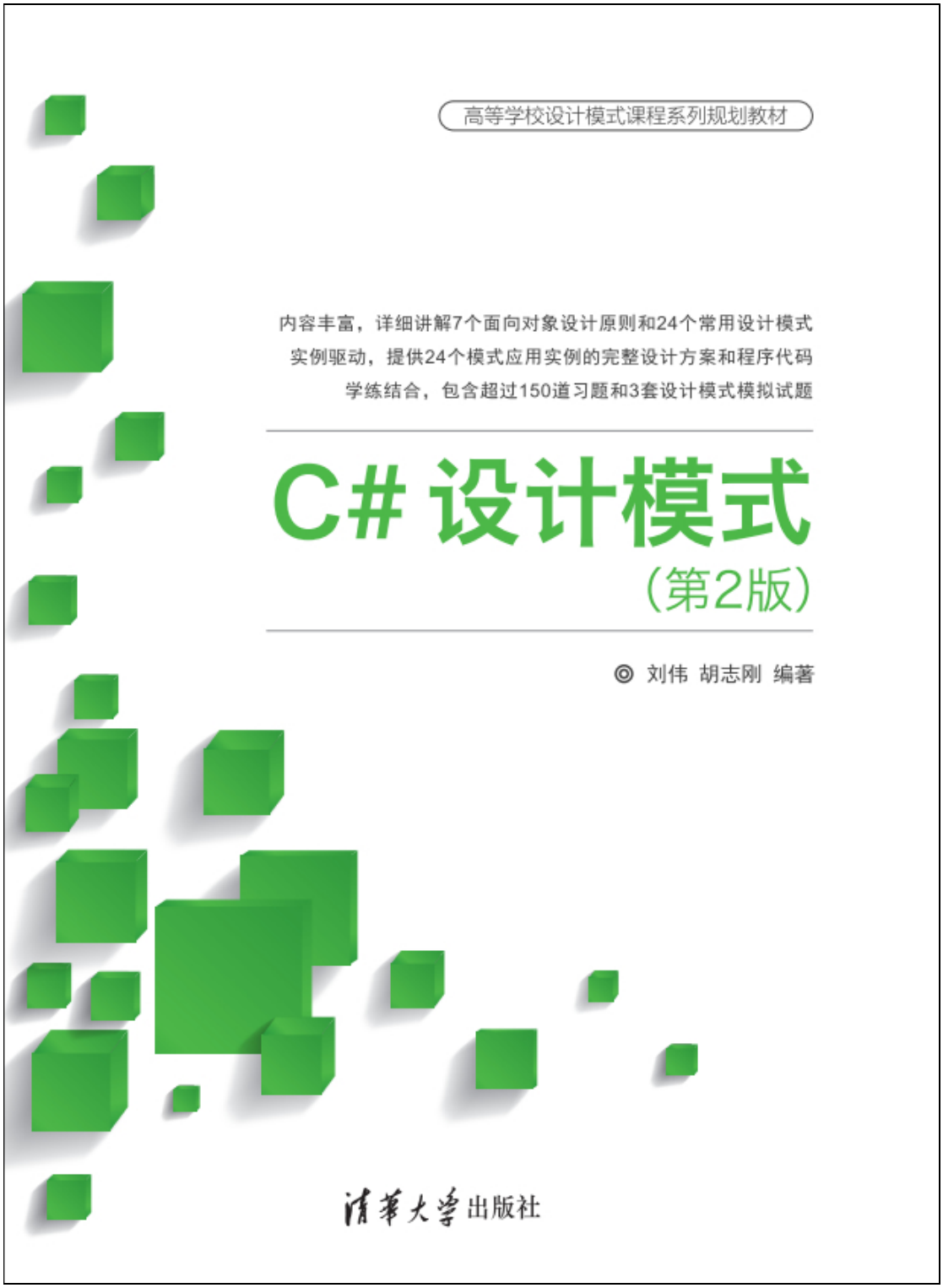
刘伟，胡志刚. C#设计模式（第 2 版）. 北京：清华大学出版社, 2018.

ISBN: 9787302485704



《C#设计模式（第 2 版）》模拟试题参考答案及评分标准

【说明：本参考答案中“综合应用题”使用 Java 代码实现，解答思路与 C#完全一致！如有意见和建议，请通过电子邮箱 weiliu_china@126.com 与作者联系！】

模拟试题一

一、选择题（每题 2 分，共 30 分）

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
C	A	A	D	B	C	D	B
(9)	**(10)**	**(11)**	**(12)**	**(13)**	**(14)**	**(15)**	
A	B	C	A	C	B	D	

二、填空题（每题 1 分，共 10 分）

1. 里氏代换原则
2. 迪米特法则
3. 简单工厂
4. 建造者
5. 外观
6. 享元
7. 职责链
8. 解释器
9. 状态
10. 访问者

三、综合应用题（每题 12 分，共 60 分）

1. 开闭原则的定义：软件实体应该对扩展开放，对修改关闭。【4 分】

说明：【8 分】

工厂方法模式代码片段如下：

```
//抽象工厂类
public abstract class Factory {
    public abstract Product createProduct();
}
//具体工厂类
public class ConcreteFactoryA extends Factory {
    public Product createProduct() {
        return new ConcreteProductA();
    }
}
```

在客户端存在如下代码片段：

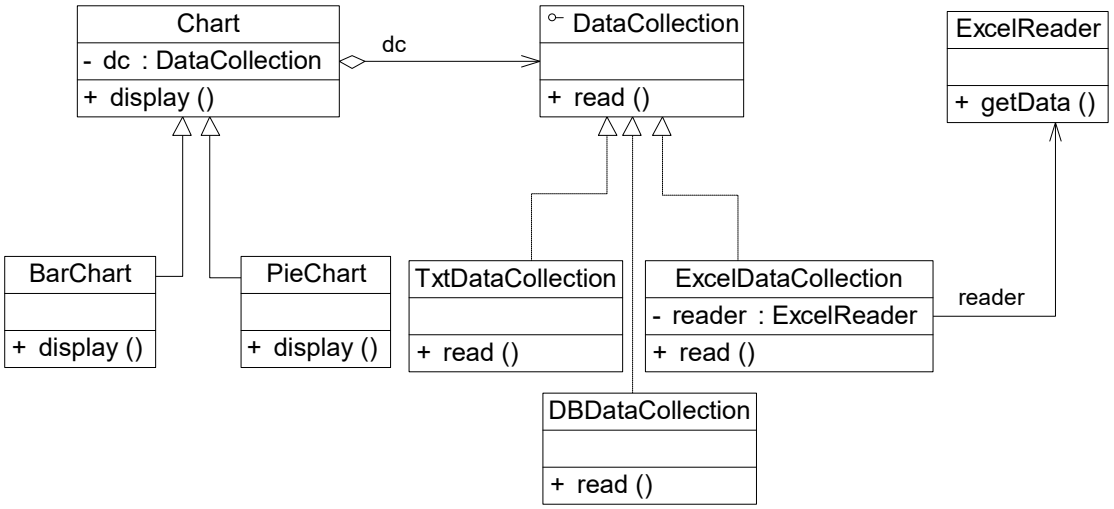
```
……
Factory factory;
factory = new ConcreteFactoryA();  //可通过反射和配置文件来实现，修改具体工厂类无须修改源代码
Product product;
product = factory.createProduct();
```

在工厂方法模式中，引入了抽象工厂类，例如上面代码中的 Factory，所有具体工厂类

都是 Factory 的子类（例如 ConcreteFactoryA），具体工厂类负责创建具体的产品。如果要新增一个具体产品 ConcreteProductB，只需要对应增加一个新的具体工厂类 ConcreteFactoryB 作为 Factory 类的子类，并实现在 Factory 中声明的 createProduct()方法即可，无须修改已有工厂类和产品类的源代码；在客户端代码中需要将 ConcreteFactoryA 改为 ConcreteFactoryB，如果采取反射和配置文件等方式来实现的话，只需修改配置文件中存储的类名即可，客户端代码也无须修改，完全符合开闭原则。

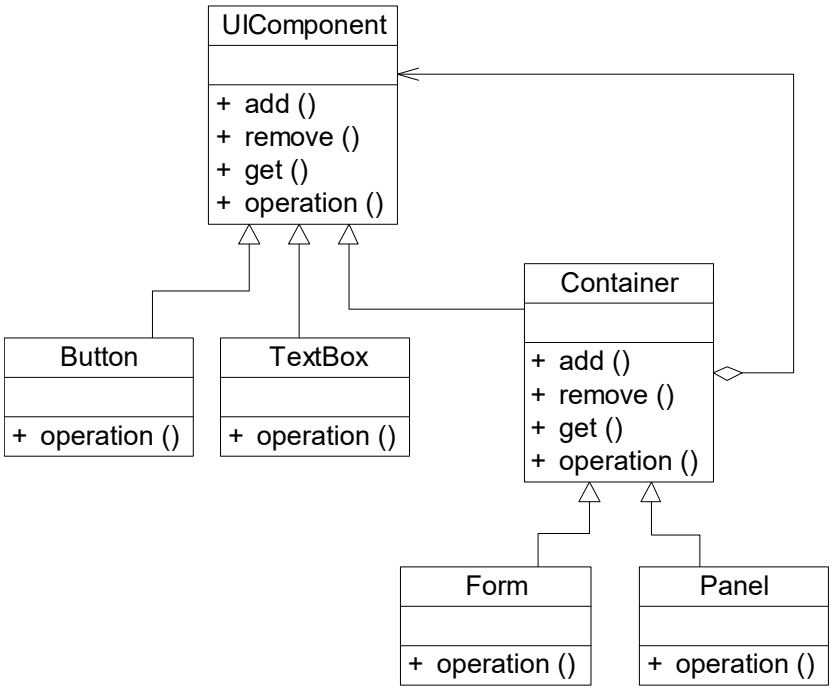
2. 可选择适配器(Adapter)模式和桥接(Bridge)模式来设计该模块。【4 分】

结构图：【8 分】



【注：可以不用标注类里面的方法和属性。】

3. 结构图：【8 分】



【注：可以不用标注类里面的方法和属性，也可以将 Form 和 Panel 直接作为 UIComponent 的子类，但必须分别持有对 UIComponent 的关联引用。】

组合模式的适用场景：【4 分，可酌情计分】

(1) 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，客户端可以一致地对待它们。

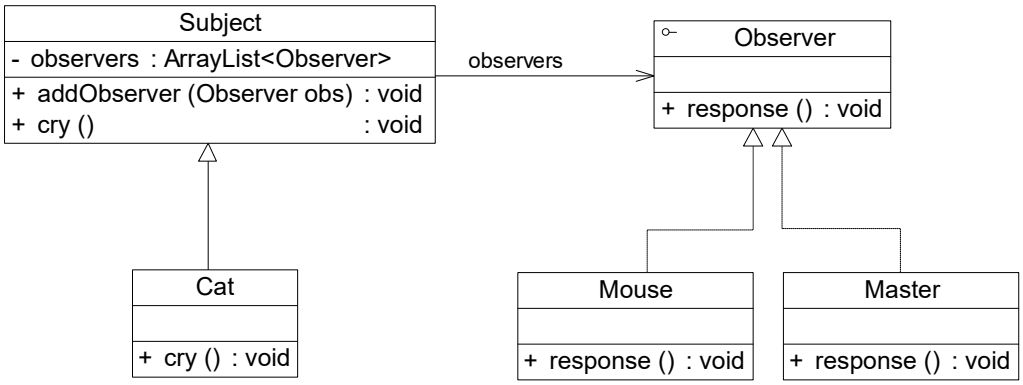
(2) 在一个使用面向对象语言开发的系统中需要处理一个树形结构。

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

4. 可选择观察者模式。【2 分】

观察者模式的定义：定义对象之间的一种一对多依赖关系，使得每当一个对象状态发生改变时，其相关依赖对象皆得到通知并被自动更新。【2 分】

结构图：【8 分】

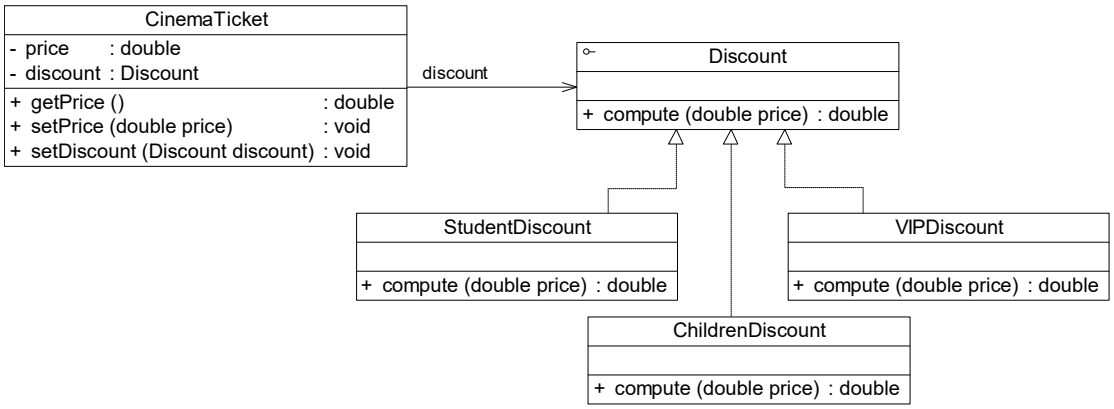


【注：可以不用标注类里面的方法和属性，抽象观察目标类 Subject 可以省略。】

5. 可选择策略模式。【2 分】

策略模式的定义：定义一系列算法类，将每一个算法封装起来，并让它们可以相互替换，策略模式让算法独立于使用它的客户而变化。【2 分】

结构图：【8 分】



【注：可以不用标注类里面的方法和属性。】

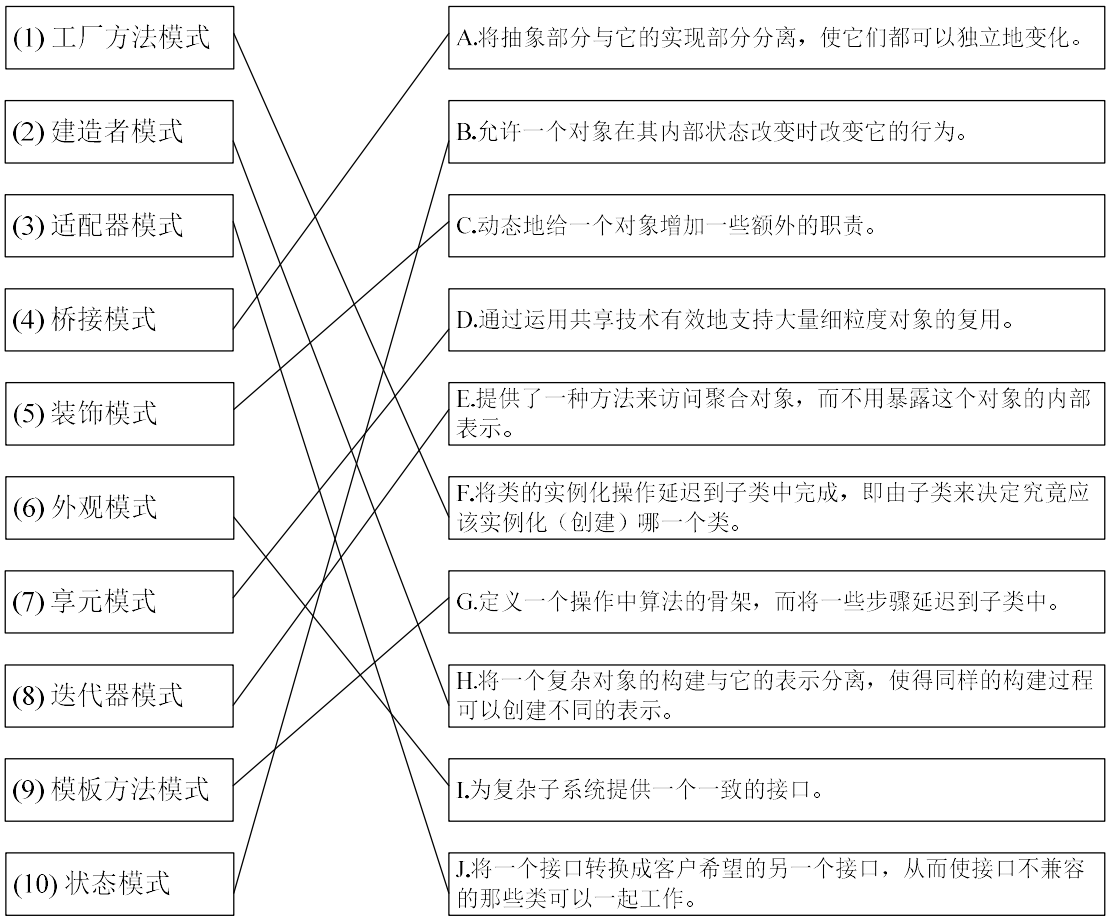
模拟试题二

一、选择题（每题 2 分，共 20 分）

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
C	B	D	C	A	C	D	B	D	A

二、连线题（每题 10 分，共 20 分）

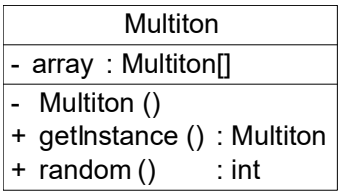
1. 每连接正确一组计 1 分。



2. 每连接正确一组计 1 分。

三、综合应用题（每题 10 分，共 60 分）

1. 多例模式。参考类图如下所示：



多例模式(Multiton Pattern)是单例模式的一种扩展形式，多例类可以在系统中存在有限多个实例，而且必须自行创建和管理这些实例，并向外界提供自己的实例，可以通过静态集合对象来存储这些实例。

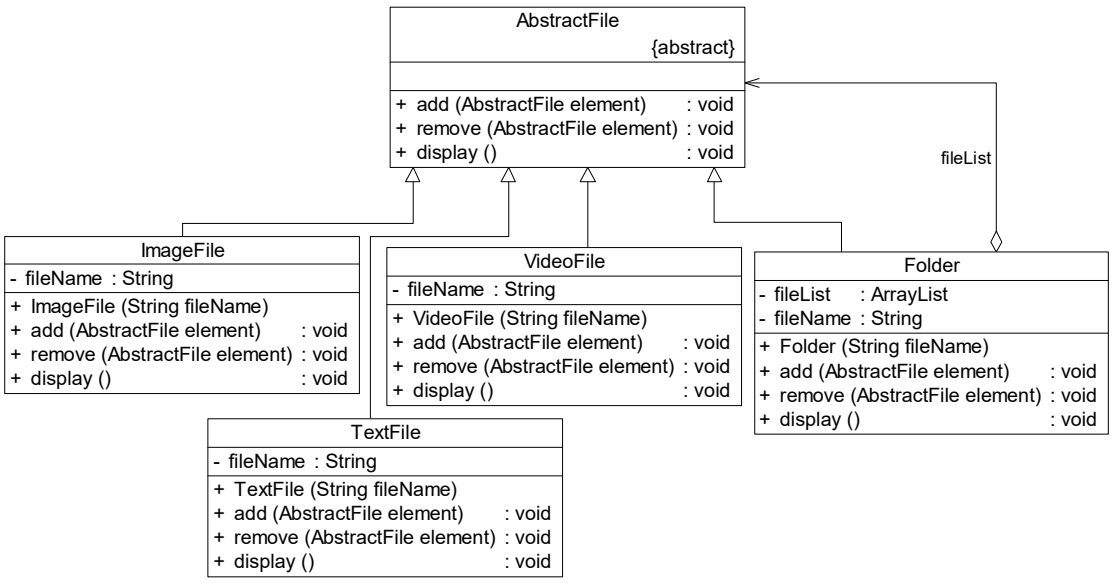
Multiton 的实现代码如下：

```
import java.util.*;

public class Multiton
{
    //定义一个数组用于存储四个实例
    private static Multiton[] array = {new Multiton(), new Multiton(), new Multiton(), new Multiton()};
    //私有构造函数
    private Multiton()
    {
```

```
    }
    //静态工厂方法，随机返回数组中的一个实例
    public static Multiton getInstance()
    {
        return array[random()];
    }
    //随机生成一个整数作为数组下标
    public static int random()
    {
        Date d = new Date();
        Random random = new Random();
        int value = Math.abs(random.nextInt());
        value = value % 4;
        return value;
    }
    public static void main(String args[])
    {
        Multiton m1,m2,m3,m4;
        m1 = Multiton.getInstance();
        m2 = Multiton.getInstance();
        m3 = Multiton.getInstance();
        m4 = Multiton.getInstance();

        System.out.println(m1==m2);
        System.out.println(m1==m3);
        System.out.println(m1==m4);
    }
}
```

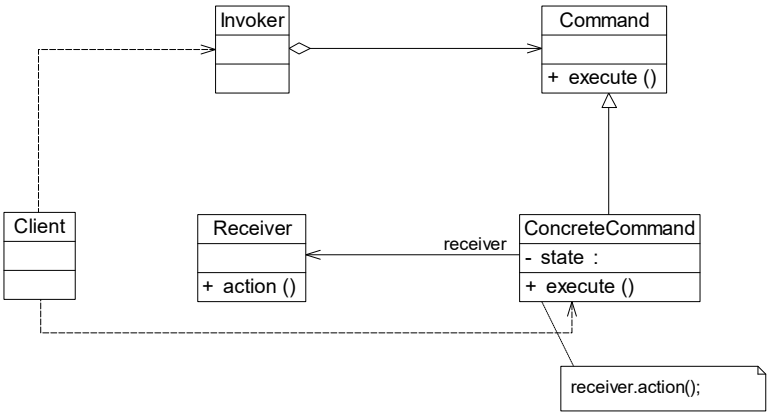
【注：类图为可选项，只要实现代码正确即可得满分 10 分。】

2. Windows 下文件目录结构是一个树形结构，可以使用组合模式来进行设计。【4 分】参考 UML 类图如下所示：【6 分】



3. 电视机遥控器蕴含了命令模式和迭代器模式等设计模式。【2 分】

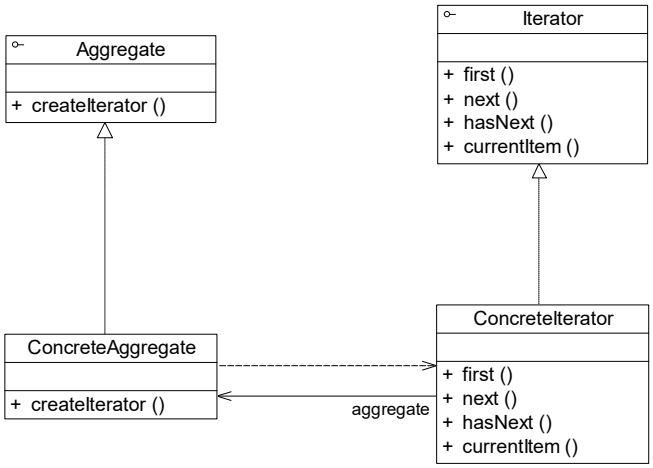
命令模式结构图：【2 分】



命令模式适用场景：【2 分】

系统需要将请求调用者和请求接收者解耦，使得调用者和接收者不直接交互；系统需要在不同的时间指定请求、将请求排队和执行请求；系统需要支持命令的撤销操作和恢复操作；系统需要将一组操作组合在一起形成宏命令。

迭代器模式结构图：【2 分】



迭代器适用场景：【2 分】

访问一个聚合对象的内容而无须暴露它的内部表示；需要为一个聚合对象提供多种遍历方式；为遍历不同的聚合结构提供一个统一的接口，在该接口的实现类中为不同的聚合结构提供不同的遍历方式，而客户端可以一致性地操作该接口。

4. 避免使用 case 和 if 等条件语句的设计模式有工厂方法模式、状态模式、策略模式等。【2 分】

下面通过策略模式来加以说明：【8 分】

如果在一个数据处理软件中可以使用多种方式来存储数据，如数据库存储、XML 文件存储、Excel 文件存储等，在没有使用策略模式之前代码片段如下：

```
public class DataHandler
{
    ……
    public void saveData(int i)
    {
        ……
        switch(i)
        {
            case(0): //使用数据库存储数据
            break;
            case(1): //使用 XML 文件存储数据
            break;
            case(2): //使用 Excel 文件存储数据
            break;
            ……
        }
        ……
    }
    ……
}
```

在上述代码中，客户端在调用 DataHandler 类的 saveData()方法时，需要传入一个参数，通过参数来确定使用哪种数据存储方式，在代码中将出现的冗长复杂的 switch…case…语句，导致 saveData()方法非常庞大，不利于维护和测试。除此之外，如果需要增加新的数据存储方式还需要修改源代码，必须增加新的 case 语句，违反了开闭原则。因此可以使用策略模式进行重构。

在策略模式中，我们可以将每一种数据存储方式封装在一个具体的策略类中，客户端针对抽象策略类编程，通过配置文件将具体策略类类名存储在其中，如果需要修改或增加策略类只需修改配置文件即可，无须修改源代码，通过策略模式重构后的代码片段如下：

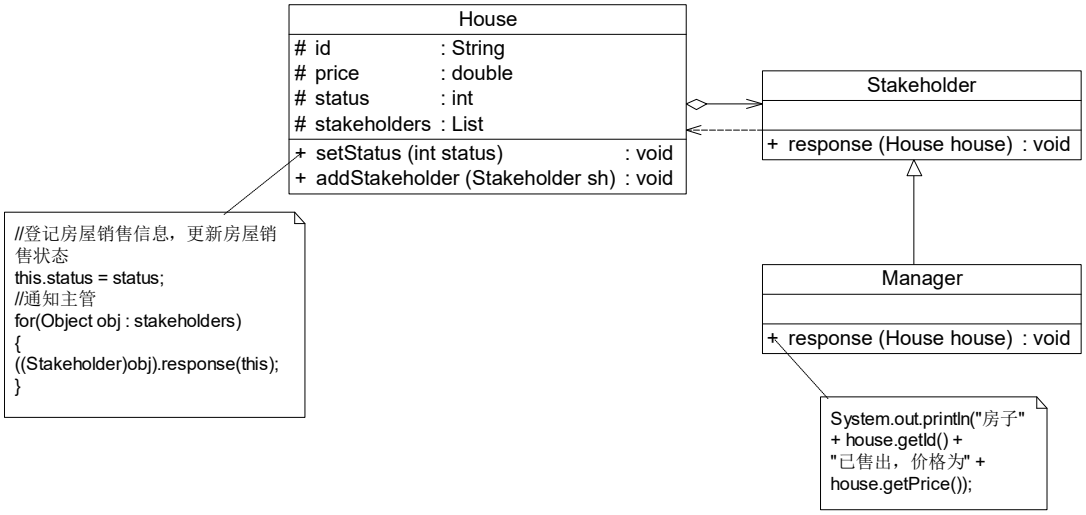
```
public class DataHandler       //环境类
{
    private Strategy strategy;
    public void setStrategy(Strategy strategy)
    {
        this.strategy = strategy;
```

```
    }
    public void saveData()
    {
        strategy.saveData();
    }
}

public abstract class Strategy      //抽象策略类
{
    public abstract void saveData();
}

public class DBStrategy extends Strategy      //具体策略类
{
    public void saveData()
    {
        //使用数据库存储数据
    }
}

public class XMLStrategy extends Strategy      //具体策略类
{
    public void saveData()
    {
        //使用 XML 文件存储数据
    }
}

public class ExcelStrategy extends Strategy      //具体策略类
{
    public void saveData()
    {
        //使用 Excel 文件存储数据
    }
}
```

【注：解答过程不需要这么详细，只需要简单的代码说明和文字描述即可。】

5. 可以选择使用观察者模式。【2 分】

参考类图和核心代码如下：【8 分】



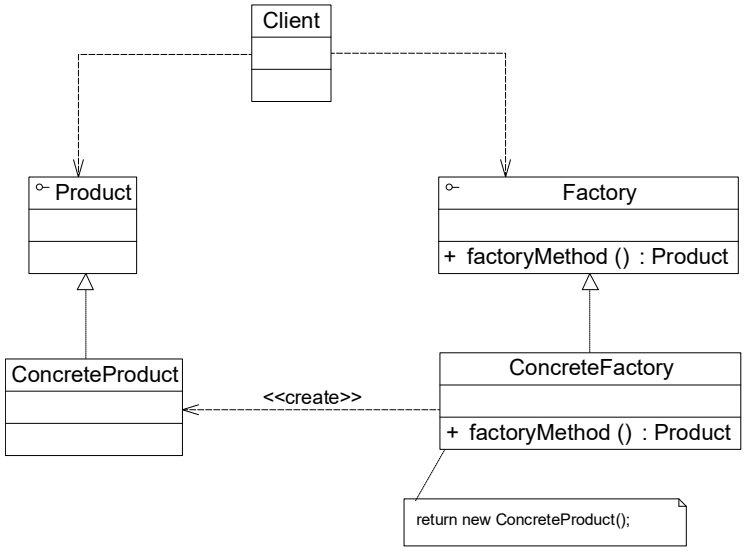
在类图中，House 类作为观察目标，相关人员类 Stakeholder 作为抽象观察者，其子类 Manager（主管）作为具体观察者，Manager 实现了在 Stakeholder 中声明的 response()方法，当房屋售出时，房屋的状态 status 将发生变化，在 setStatus()方法中调用观察者的 response()方法，即主管将收到相应消息，此时应用了观察者模式。

【注：代码可以提取出来单独编写，核心代码为观察目标的通知方法和观察者的响应方法。】

6. 依赖倒转原则的定义：高层模块不应该依赖低层模块，它们都应该依赖抽象。抽象不应该依赖于细节，细节应该依赖于抽象。也可以定义为：要针对接口编程，不要针对实现编程。【4 分】

可结合工厂方法模式、抽象工厂模式、建造者模式、适配器模式、桥接模式、策略模式等具有抽象层的模式，下面以工厂方法模式为例来进行说明。

工厂方法模式结构图：【3 分】



工厂方法模式代码片段如下：【3 分】

```
//抽象工厂类
public abstract class Factory {
    public abstract Product createProduct();
```

```
}
//具体工厂类
public class ConcreteFactoryA extends Factory {
    public Product createProduct() {
        return new ConcreteProductA();
    }
}
```

在工厂方法模式中，定义了抽象工厂类 Factory，所有的具体工厂类都是 Factory 的子类，具体工厂类负责创建具体的产品对象。

客户端代码片段如下：

```
……
Factory factory;
Product product;

//通过反射和配置文件创建工厂对象 factory
product = factory.createProduct();
```

客户端代码中，针对抽象工厂和抽象产品编程，使用抽象工厂类型来声明工厂对象，使用抽象产品类型来声明产品对象，程序在运行时，具体工厂对象将覆盖抽象工厂对象，可以采用反射结合配置文件的方式来创建具体工厂对象。所有出现工厂和产品的地方都是针对抽象编程，而不是针对具体工厂和产品编程，应用了依赖倒转原则，这样做的好处是更换和新增具体产品时，只需要修改存储在配置文件中的具体工厂类类名即可，无须修改源代码，使得系统符合开闭原则。开闭原则是目标，依赖倒转原则是手段。

【注：重点是客户端代码要针对抽象编程】

模拟试题三

一、判断题（每题 1 分，共 20 分）

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
错	对	对	错	错	错	对	对	对	错
(11)	**(12)**	**(13)**	**(14)**	**(15)**	**(16)**	**(17)**	**(18)**	**(19)**	**(20)**
对	对	错	对	对	对	错	错	对	对

二、【本题共 15 分】

开闭原则的定义：软件实体应当对扩展开放，对修改关闭。【3 分】

如何实现开闭原则：抽象化是开闭原则的关键，提供相对稳定的抽象层和灵活的具体层。【3 分】

简单工厂模式：违背了开闭原则，增加新的产品时需要修改工厂类。【1 分】

简单工厂模式的结构图：【2 分】

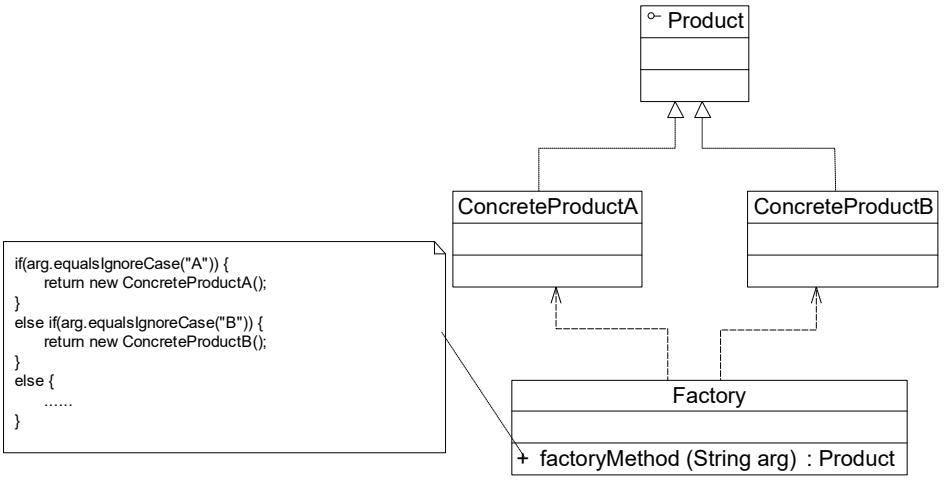


图 1 简单工厂模式结构图

工厂方法模式：符合开闭原则，增加新的产品只需对应增加一个新的具体工厂类，无需修改源代码。【1 分】

工厂方法模式的结构图：【2 分】

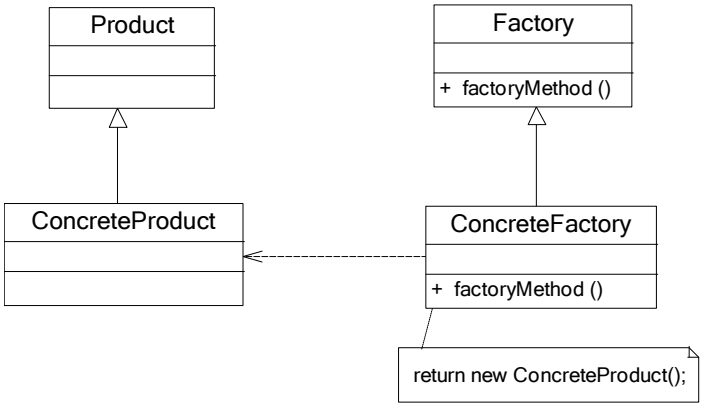


图 2 工厂方法模式结构图

抽象工厂模式：具有开闭原则的倾斜性，增加新的产品族符合开闭原则，增加新的产品等级结构违背开闭原则。【1 分】

抽象工厂模式的结构图：【2 分】

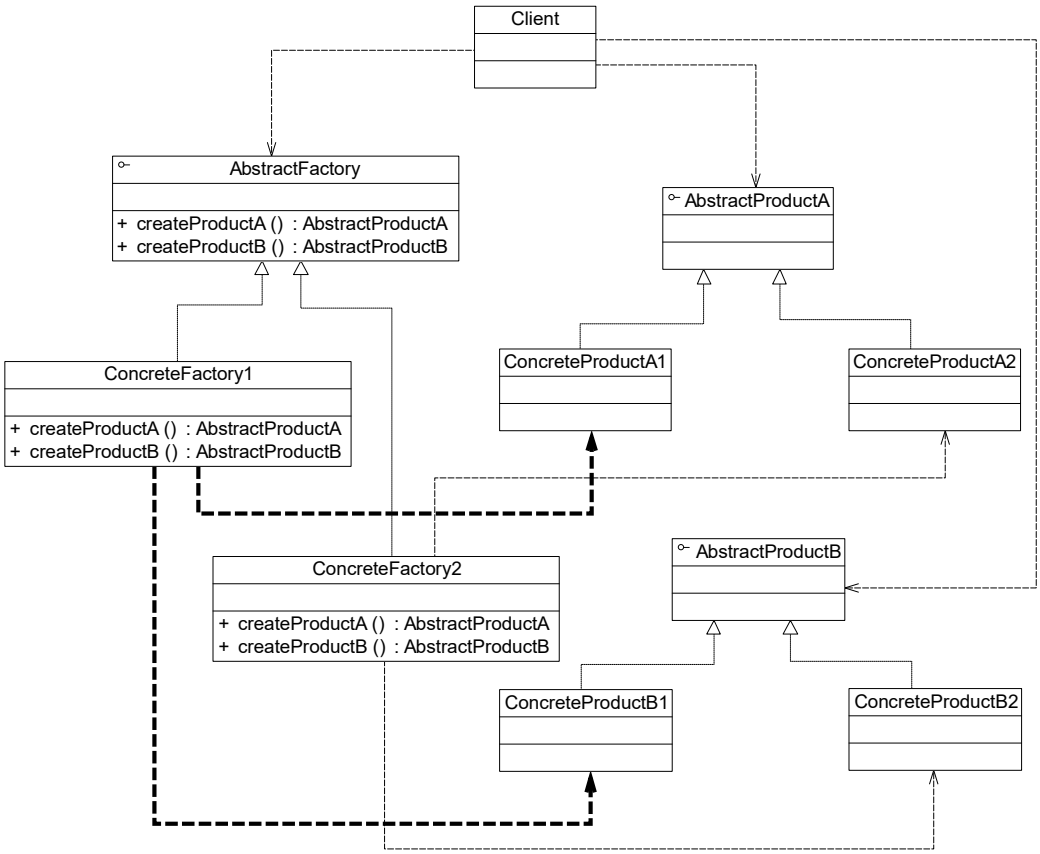


图 3 抽象工厂模式结构图

三、【本题共 10 分】

存在的问题：将两个变化维度耦合在一起，违背了单一职责原则，导致类的个数增加，系统庞大，扩展困难。【4 分】

设计模式选择：桥接模式。【2 分】

重构之后的设计方案结构图：【4 分】

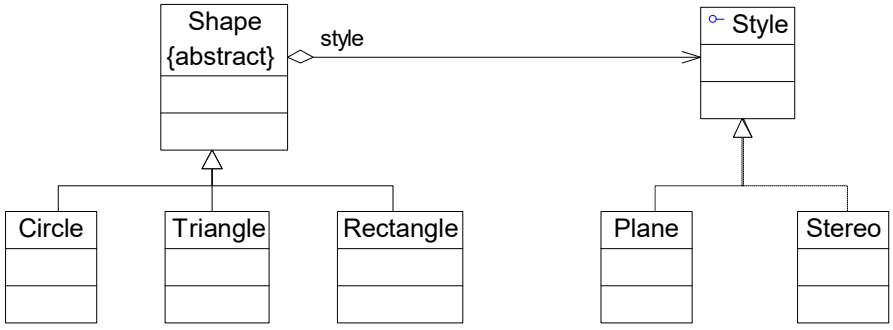


图 4 使用桥接模式重构后的结构图

四、【本题共 15 分】

设计模式的选择：单例模式和外观模式。【4 分，每个设计模式 2 分】

单例模式的定义：确保一个类只有一个实例，并提供一个全局访问点来访问这个唯一实例。【3 分】

外观模式的定义：为子系统中的一组接口提供一个统一的入口。外观模式定义了一个高

层接口，这个接口使得这一子系统更加容易使用。【3 分】

解决方案结构图：【5 分】

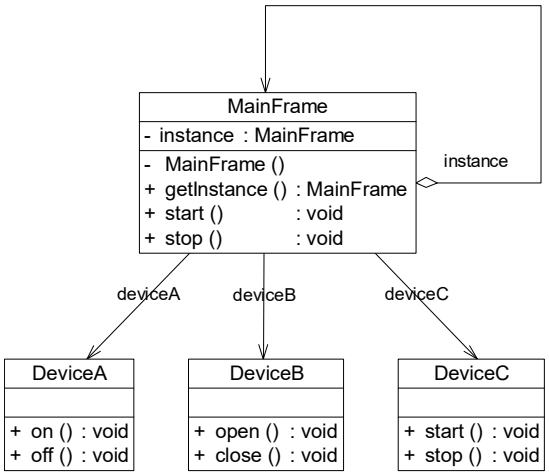


图 5 单例模式和外观模式联用解决方案结构图

五、【本题共 15 分】

设计模式的选择：组合模式和观察者模式。【4 分，每个设计模式 2 分】

组合模式的定义：组合多个对象形成树形结构以表示具有部分-整体关系的层次结构。组合模式让客户端可以统一对待单个对象和组合对象。【3 分】

观察者模式：定义对象之间的一种一对多依赖关系，使得每当一个对象状态发生改变时，其相关依赖对象都得到通知并被自动更新。【3 分】

解决方案结构图：【5 分】

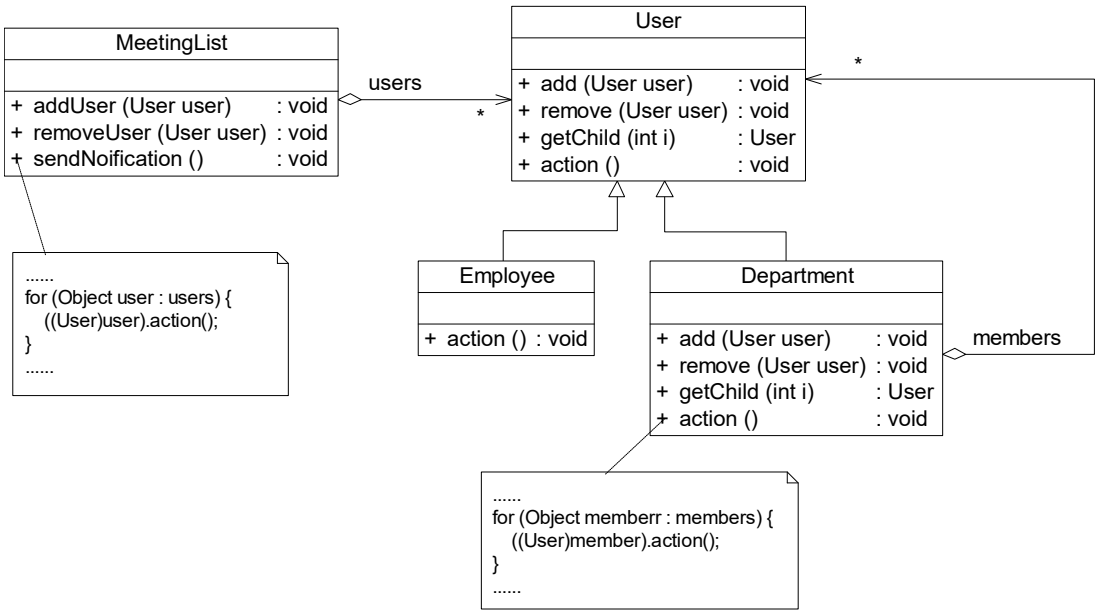


图 6 组合模式和观察者模式联用解决方案结构图

六、【本题共 25 分】

【问题 1：6 分】

创建型模式主要用于创建对象。【2 分】

结构型模式主要用于处理类或对象的组合。【2 分】

行为型模式主要用于描述类或对象怎样交互和怎样分配职责。【2 分】

【问题 2：9 分】

创建型模式：单例模式、抽象工厂模式、原型模式。【共 3 分，每个 1 分】

结构型模式：适配器模式、组合模式、代理模式。【共 3 分，每个 1 分】

行为型模式：命令模式、职责链模式、策略模式。【共 3 分，每个 1 分】

【问题 3：10 分】

针对设计要求(1)，可选择策略模式【2 分】，对应的解决方案结构图如下【3 分】：

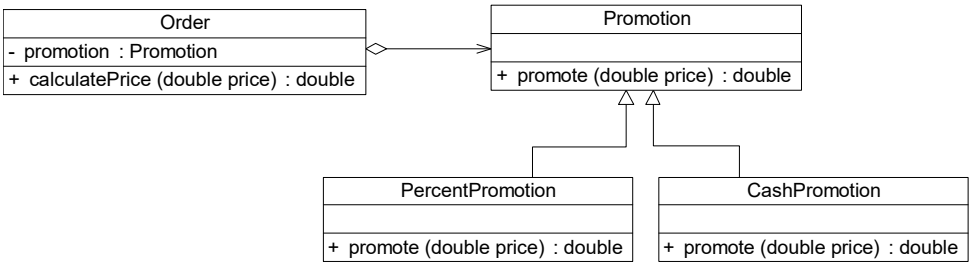


图 7 策略模式解决方案结构图

针对设计要求(2)，可选择适配器模式【2 分】，对应的解决方案结构图如下【3 分】：

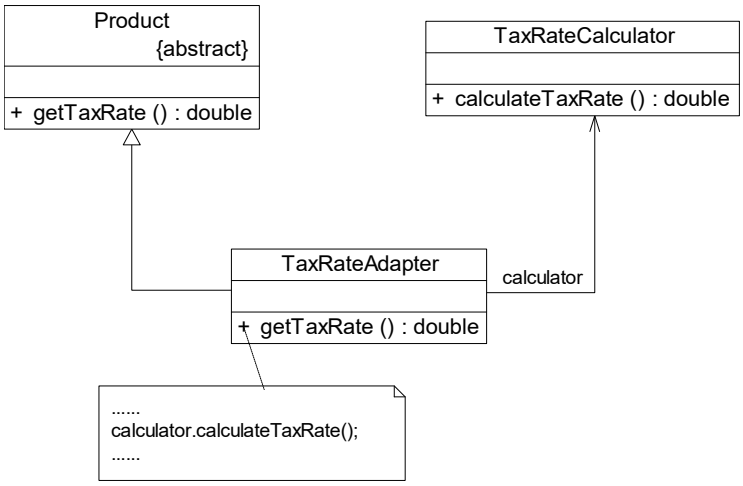


图 8 适配器模式解决方案结构图