**简答题检索表**

1. **设计模式与面向对象【P3】**
2. 什么是设计模式？它包含哪些基本要素？
3. 设计模式如何分类？每一类设计模式有何特点？
4. 设计模式具有哪些优点？
5. 除了设计模式以外，目前有不少人在从事“反模式”的研究，请查阅相关资料，了解何谓“反模式”以及研究“反模式”的意义。
6. 请查阅相关资料，了解在JDK中使用了哪些设计模式，在何处使用了何种模式，至少列举两个。
7. 有人将面向对象设计原则简单地归为3条：①封装变化点；②对接口进行编程；③多使用组合，而不是继承。请查阅相关资料并结合本章所学内容谈谈对这3条原则的理解。
8. 结合本章所学的面向对象设计原则谈谈对类和接口“粒度”的理解。
9. 结合面向对象设计原则分析正方形是否为长方形的子类？
10. 面向对象软件设计中，给一个类或对象增加行为（新功能）的方式有哪些？请举例说明。
11. 为什么优先使用对象组合而不是类继承。
12. 你认为面向对象方法中类设计的难点是什么，如何应对？
13. 你认为设计模式能否增强软件复用性？请列举两个模式并结合实例进行说明。
14. 请阐述你对开闭原则的理解，以及如何设计能达到开闭原则的要求
15. 请举例说明你对白箱复用和黑箱复用的理解。
16. GOF设计模式有几种类型，分别包括哪些模式。
17. 简述什么是硬编码，并说明硬编码有什么缺点？
18. 在面向对象设计时，采用水平关联（含依赖、组合、聚合、普通关联等）的方法或者继承都可以适应变化。请说明两种方式在适应变化上各自的优缺点。
19. **在面向对象编程中往往提倡尽量不使用条件判断语句（例如 if…else…语句或者switch…case…语句）。选择一种有助于避免使用条件判断语句的设计模式，结合文字说明和代码片段来解释它是如何避免的。**
20. 结合一种合适的设计模式谈谈对依赖倒转原则的理解，要求给出依赖倒转原则的定义并结合代码片段及结构图进行说明。
21. **什么是开闭原则？简要说明如何实现开闭原则。分别讨论3种工厂模式（简单工厂模式、工厂方法模式、抽象工厂模式）是否支持开闭原则，并结合类图加以说明。**
22. **创建型模式【P8】**
23. 请举例说明你对创建型设计模式的理解。
24. 抽象工厂模式和原型模式有哪些相同和不同，可以联合使用么？
25. 简述工厂方法模式和抽象工厂模式的基本思想，并说明两者有何异同？
26. 请举例说明什么是简单工厂方法？什么是单件（单例）模式？分别是如何实现的？
27. 分析并理解饿汉式单例与懒汉式单例的异同。
28. 什么是双重检查锁定？为什么要进行双重检查锁定？ Java 如何实现双重检查锁定？
29. 分别采用双重检查锁定和 IoDH方式实现8.3节的负载均衡器。

8、**结合工厂方法模式，谈谈对开闭原则的理解。要求在解答时给出开闭原则的定义并结合代码片段进行说明，编程语言不限。**

9、使用Java语言编程实现一个多例模式（Multiton Pattern），确保系统中某个类的对象只能存在有限多个，例如两个或3个。

10、比较深克隆与浅克隆的异同。

1. **结构型模式【P11】**
2. 请举例说明哪些情况下可以使用组合模式。
3. 请举例说明在应用标准外观模式时可能产生的问题，以及对应的解决方案。
4. 请举例说明桥接模式如何将抽象部分与它的实现部分分离，使它们可以独立地变化。
5. 简述组合模式和装饰模式的基本思想，并说明两者有何导同。
6. 请具体说明你认为文档编辑器是如何处理大量的字符对象的。
7. 简述类适配器模式和对象适配器模式的基本思想，并说明两者有何异同？
8. 简述外观模式和中介者模式的基本思想，并说明两者有何异同？
9. 变化整体上分为接口变化和实现变化。若这两类变化同时存在，通常应首先采用哪种设计模式分离稳定与变化部分？举例说明如何设计。
10. 线性聚集关系是一种现实中常见的逻辑关系，如一个A聚集多个B，一个B聚集多个C，……。使用合成模式可以以统一的接口访问各部分。请给出本例应用合成模式的设计类图。子类聚集父类是该模式的特点之一，请再说出也具有该特点的其它几个模式的名称（至少2个）。
11. 在对象适配器中一个适配器能否适配多个适配者?如果能，应该如何实现？如果不能，请说明原因。如果是类适配器呢？
12. 如果系统中存在两个以上的变化维度，是否可以使用桥接模式进行处理？如果可以，系统该如何设计？
13. 在组合模式结构图中，如果聚合关联关系不是从 Composite 到 Component,而是从 Composite 到Leaf，如图所示，会产生怎样的结果？
14. 半透明装饰模式能否实现对同一个对象的多次装饰？为什么？
15. 毕业生通过职业介绍所找工作，请问该过程蕴含了哪种设计模式，绘制相应的类图。

**四、行为型模式【P15】**

1、请给出一个适合使用策略模式的示例场景，并说明与不使用策略模式相比，有哪些优点？

2、请给出一个适合使用状态模式的示例场景，并说明与不使用状态模式相比，有哪些优点？

3、简述状态模式和策略模式的基本思想，并说明两者有何异同？

4、现在大多数软件都有撤销（Undo）和重做（Redo）的功能，请问这种功能一般是采用什么设计模式实现的，并举例说明是如何实现的。【文本编辑软件的撤销与重做功能】

5、为达到某些设计目的，设计时可以同时组合使用多个设计模式。请举出一个同时组合使用策略模式和对象适配器模式。要求给出例子的应用场景、设计方案。

6、Java语言中的异常处理机制是职责链模式的一个应用实例，编写一个包含多个catch子句的程序，理解异常处理的实现过程，并判断此处使用的是纯的职责链模式还是不纯的职责链模式。

7、在策略模式中一个环境类Context能否对应多个不同的策略等级结构？如何设计？

8、在模板方法模式中，钩子方法如何实现子类控制父类的行为？

**9、电视机遥控器的设计原理中蕴含了哪些设计模式，列举两个，绘制这两种设计模式的结构图并简单论述其适用场景。**

**一、设计模式与面向对象**

**1、什么是设计模式？它包含哪些基本要素？**

【参见《Java设计模式》P5】

**2、设计模式如何分类？每一类设计模式有何特点？**

【参见《Java设计模式》P6-P7】

**3、设计模式具有哪些优点？**

【参见《Java设计模式》P9-P10】

**4、除了设计模式以外，目前有不少人在从事“反模式”的研究，请查阅相关资料，了解何谓“反模式”以及研究“反模式”的意义。**

反模式(AntiPatterns)是指那些导致开发出现障碍的负面模式，即在软件开发中普遍存在、反复出现并会影响到软件成功开发的不良解决方案。反模式是关注于负面解决方案的软件研究方向，揭示出不成功系统中存在的反模式有利于在成功系统中避免出现这些模式，有助于降低软件缺陷和项目失败出现的频率。反模式清晰定义了大部分人在软件开发过程中经常会犯的一些错误，根据视角的不同，可分为开发性反模式、架构性反模式和管理性反模式。

**5、请查阅相关资料，了解在JDK中使用了哪些设计模式，在何处使用了何种模式，至少列举两个。**

【详见文件A】

**6、有人将面向对象设计原则简单地归为3条：①封装变化点；②对接口进行编程；③多使用组合，而不是继承。请查阅相关资料并结合本章所学内容谈谈对这3条原则的理解。**

【参见《实验》P30】

**7、结合本章所学的面向对象设计原则谈谈对类和接口“粒度”的理解。**

【参见《实验》P30】

**8、结合面向对象设计原则分析正方形是否为长方形的子类？**

【参见《实验》P30】

**9、面向对象软件设计中，给一个类或对象增加行为（新功能）的方式有哪些？请举例说明。**

①继承机制：使用继承机制是给现有类添加功能的一种有效途径，通过继承一个现有类可以使得子类在拥有自身方法的同时还拥有父类的方法，但是这种方法是静态的，用户不能控制增加行为的方式和时机。

②关联机制：将一个类的对象嵌入另一个对象中，由另一个对象来调用嵌入对象的行为同时扩展其行为，我们称另一个对象为装饰器。

**10、为什么优先使用对象组合而不是类继承。【与17结合】**

参照面向对象设计原则之组合/聚合复用原则，应尽量使用组合/聚合，不要使用类继承。

**（1）通过组合/聚合复用的优缺点**

**优点**：

①新对象存取成员对象的唯一方法是通过成员对象的接口；

②这种复用是黑箱复用，因为成员对象的内部细节是新对象所看不见的；

③这种复用更好地支持封装性；

④这种复用实现上的相互依赖性比较小；

⑤每一个新的类可以将焦点集中在一个任务上；

⑥这种复用可以在运行时间内动态进行，新对象可以动态的引用与子对象类型相同的对象。

⑦作为复用手段可以应用到几乎任何环境中去。

**缺点: 就是系统中会有较多的对象需要管理。**

**（2）通过继承来进行复用的优缺点**

优点：

①新的实现较为容易，因为基类的大部分功能可以通过继承的关系自动进入派生类。

②修改和扩展继承而来的实现较为容易。

**缺点**：

①继承复用破坏封装性，因为继承将基类的实现细节暴露给派生类。由于基类的内部细节常常是对于派生类透明的，所以这种复用是透明的复用，又称“白箱” 复用。

②如果基类发生改变，那么派生类的实现也不得不发生改变。

③从基类继承而来的实现是静态的，不可能在运行时间内发生改变，没有足够的灵活性。

**11、你认为面向对象方法中类设计的难点是什么，如何应对？**

【奔腾答案】难点：设计可复用的类要求必须找到相关的对象，以适当的粒度归类，在定义类的接口和继承层次，建立对象间的基本关系。你的设计应该对手头的问题有针对性，同时对将来的问题和需求也要有足够的通用性。你也希望避免重复设计或尽可能少做重复设计。应对方法：学习设计模式（面向对象软件的设计经验），从而更加简单方便地复用成功的设计和体系结构。

【PPT】类设计的难点在于——变化的存在。包括职责的变化（接口、功能的变化）和实现的变化（数据表示、行为的变化）。变化的适应方式：修改既有代码的缺点（既有代码的获取问题、改变实现的顾虑、改变接口的顾虑）和扩展既有代码（继承方式扩展、关联方式扩展、依赖方式扩展、聚合方式扩展、组合方式扩展）

**12、你认为设计模式能否增强软件复用性？请列举两个模式并结合实例进行说明。**

设计模式能增强软件复用性。

①适配器模式：将一个类的接口转换威客户希望的另外一个接口，增强了软件复用性，比如 windows 下的软件和 linux下的软件接口不兼容，如果想要不修改源代就使用的话，那么可以使用适配器模式定义新的接口，接口里面具体功能还是原来的已有接口的方法，从而解决接口不兼容问题，实现了软件复用性。

②装饰模式：动态的给对象添加一些职责。它也可以增强软件复用性，比如你已经有多个功能比较少的类，你想要一个功能复杂，更强的类，那么就可以使用该模式，将简单的类复合起来，组合出复杂的功能。

**13、请阐述你对开闭原则的理解，以及如何设计能达到开闭原则的要求**

**理解**：软件实体（模块、类、方法）应对扩展开放，对修改关闭，可以让软件系统可复用，并易于维护。

**设计方法**：

①对软件系统中的不变部分加以抽象，在面向对象的设计中，可以把这些不变的部分加以抽象成不变的接口，这些不变的接口可以应对未来的扩展；

②接口的最小功能设计原则。根据这个原则，原有的接口要么可以应对未来的扩展；不足的部分可以通过定义新的接口来实现；

③模块之间的调用通过抽象接口进行，这样即使实现层发生变化，也无需修改调用方的代码。

④接口可以被复用，但接口的实现却不一定能被复用。接口是稳定的，关闭的，但接口的实现是可变的，开放的。可以通过对接口的不同实现以及类的继承行为等为系统增加新的或改变系统原来的功能，实现软件系统的柔性扩展。 简单地说，软件系统是否有良好的接口（抽象）设计是判断软件系统是否满足开闭原则的一种重要的判断基准现在多把开闭原则等同于面向接口的软件设计。

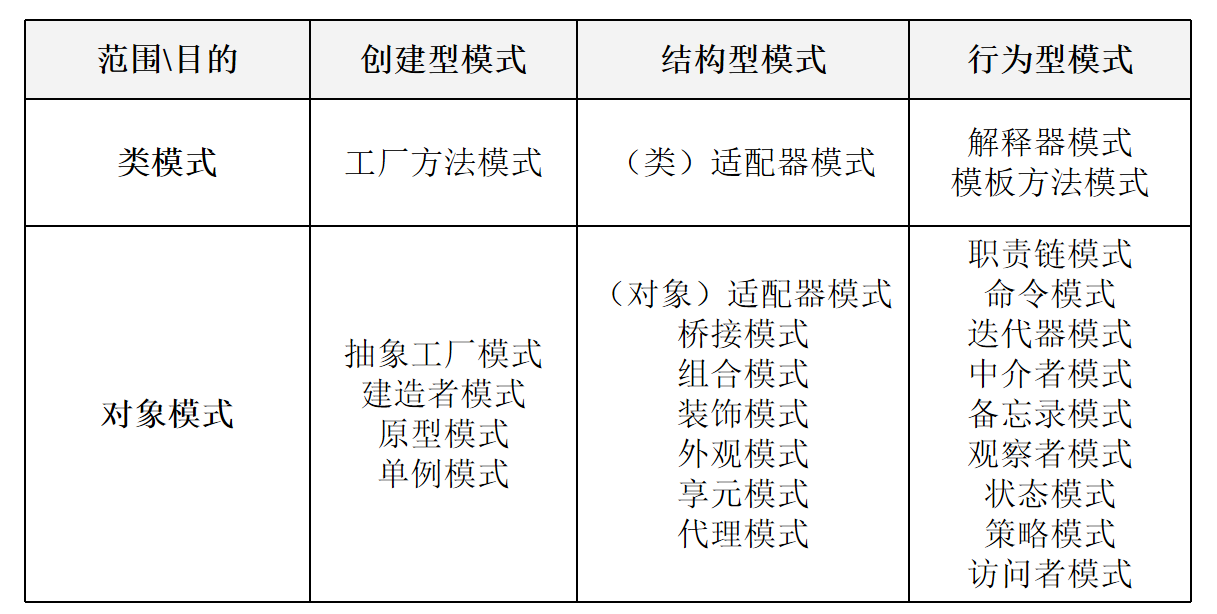
**14、请举例说明你对白箱复用和黑箱复用的理解。**

**白箱复用**指的是在了解内部实现细节的情况下进行代码复用，典型的例子是继承。在面向对象编程中，子类通过继承父类的成员变量和方法来进行复用。这种方式的优点是复用力度大，但缺点是子类对父类的实现细节有较高的依赖度，可能会导致耦合性增加。

**黑箱复用**指的是不需要了解内部实现细节，通过接口进行复用，典型的例子是组合。在这种方式下，一个对象通过包含另一个对象来实现功能复用，而不关心其内部实现。这种方式降低了耦合性，但复用力度相对较低。

**15、GOF设计模式有几种类型，分别包括哪些模式。**

根据目的分为三种：创建型、结构型和行为型；根据范围分为两种：类模式和对象模式。



**16、简述什么是硬编码，并说明硬编码有什么缺点？**

硬编码（Hardcoding或Hard-coding）是指在编写程序时，直接将具体的值（如字符串、数字、路径等）写入源代码中，而不是通过变量、配置文件、数据库查询或其他动态方法来获取这些值。

**缺点**：

①灵活性差：硬编码的数据或逻辑无法在运行时更改，导致程序缺乏灵活性和可配置性。

②可维护性低：修改硬编码的值需要直接更改源代码，可能引入错误，并且增加了维护的成本和难度。

③复用性差：硬编码的逻辑往往与具体实现耦合紧密，难以在不同的项目或模块中复用。

④可读性差：硬编码的数据和逻辑混杂在代码中，降低了代码的可读性和可理解性。

**17、在面向对象设计时，采用水平关联（含依赖、组合、聚合、普通关联等）的方法或者继承都可以适应变化。请说明两种方式在适应变化上各自的优缺点。【与10结合】**

**（1）水平关联（组合）**

**优点**

①不破环封装性，类之间松耦合，彼此相对独立

②具有较好的可扩展性

③在运行时，整体对象可以选择不同类型的局部对象

④整体可以对局部进行包装，封装局部类的接口提供新的接口

**缺点**

①整体类不能自动获得和局部类同样的接口

②创建整体类的对象时需要创建所有局部类的对象

**（2）继承关系**

**优点**

①子类能自动继承父类的接口

②子类对象创建时，无需创建父类对象

**缺点**

①破坏封装性，父类子类之间紧耦合，子类依赖父类实现，缺乏独立性

②支持扩展，但往往以增加系统结构为代价

③不支持动态继承，在运行时子类无法选择不同的父类

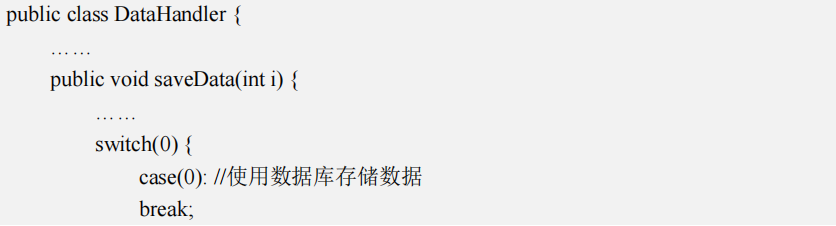
④子类不能改变父类的接口

**18、在面向对象编程中往往提倡尽量不使用条件判断语句（例如 if…else…语句或者switch…case…语句）。选择一种有助于避免使用条件判断语句的设计模式，结合文字说明和代码片段来解释它是如何避免的。**

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。

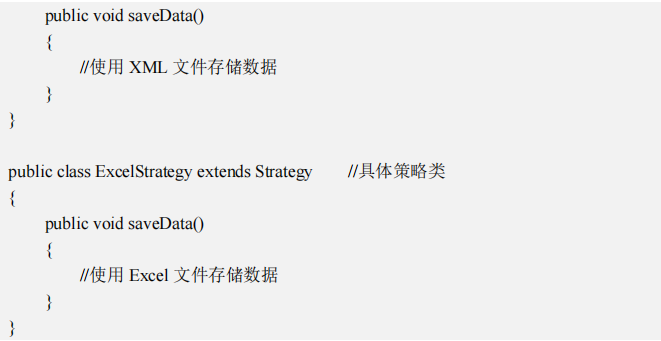
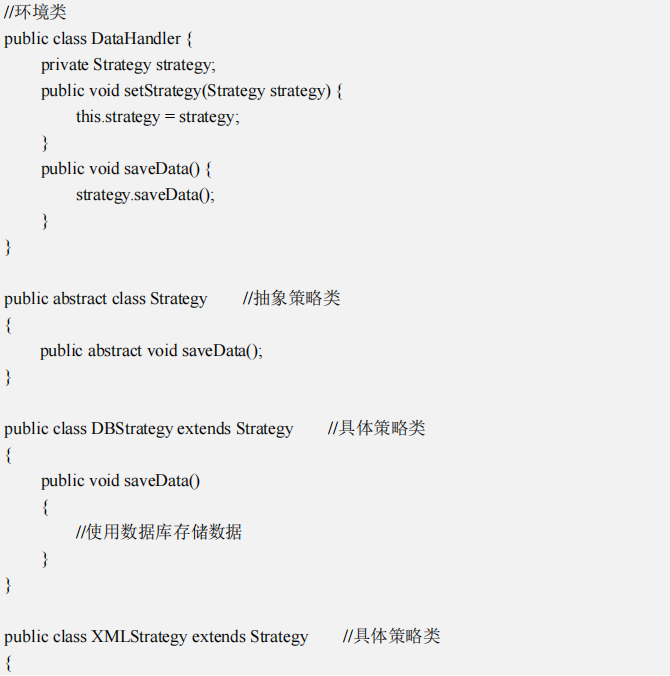
下面通过策略模式来加以说明：

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：



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：

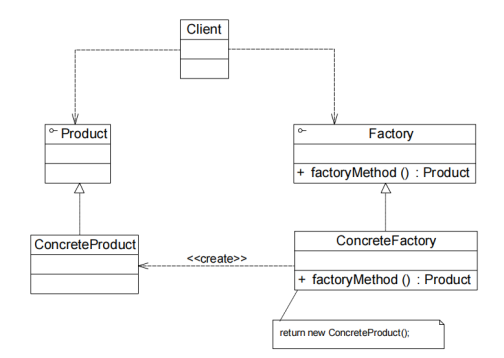


**19、结合一种合适的设计模式谈谈对依赖倒转原则的理解，要求给出依赖倒转原则的定义并结合代码片段及结构图进行说明。**

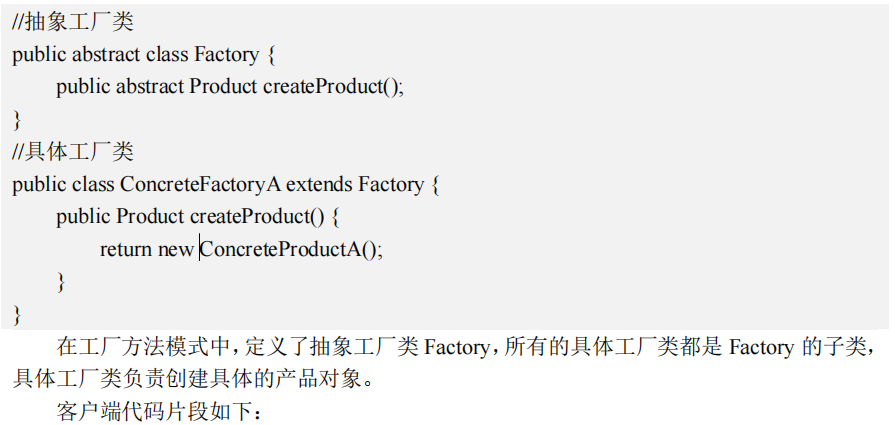
依赖倒转原则的定义：高层模块不应该依赖低层模块，它们都应该依赖抽象。抽象不应该依赖于细节，细节应该依赖于抽象。也可以定义为：要针对接口编程，不要针对实现编程。

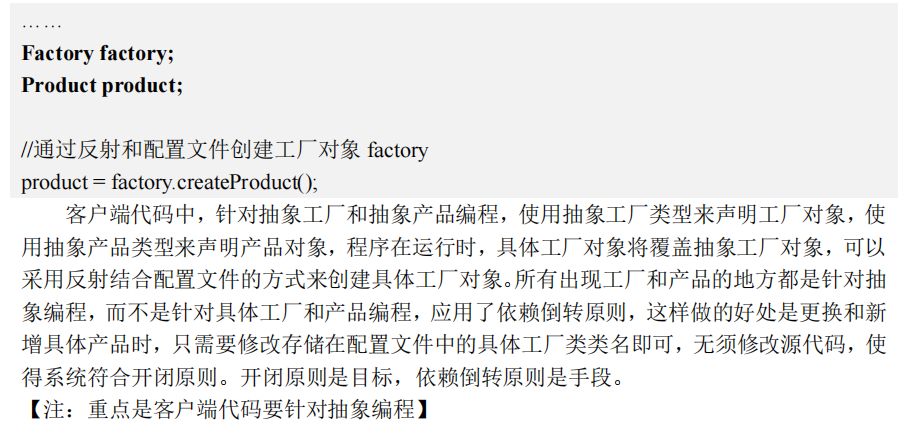
可结合工厂方法模式、抽象工厂模式、建造者模式、适配器模式、桥接模式、策略模式等具有抽象层的模式，下面以工厂方法模式为例来进行说明。

工厂方法模式结构图：



工厂方法模式代码片段如下：





**20、什么是开闭原则？简要说明如何实现开闭原则。分别讨论3种工厂模式（简单工厂模式、工厂方法模式、抽象工厂模式）是否支持开闭原则，并结合类图加以说明。**

【实验P188】

**二、创建型模式**

**1、请举例说明你对创建型设计模式的理解。**

创建型模式（Creational Pattern）关注的是对象的创建，将创建对象（类的实例化）的过程进行了抽象和封装，分离了对象创建和对象使用。作为客户程序仅仅需要去使用对象，而不再关心创建对象过程中的逻辑。帮助一个系统独立于如何创建、组合和表示它的那些对象。一个类创建型模式使用继承改变被实例化的类，而一个对象创建型模式将实例化委托给另一个对象。

主要特点：

·客户不知道对象的具体类是什么（除非看源代码）

·隐藏了对象实例是如何被创建和组织的

例如：一个人想吃苹果时，仅关心所获得的苹果对象，至于苹果是怎么来，超市买来的，或者果园摘来的，他并不需要关心。

两种创建型模式：

·类创建型模式：使用继承改变被实例化的类。

·对象创建型模式：将对象实例化委托给另一个对象。

1. **抽象工厂模式和原型模式有哪些相同和不同，可以联合使用么？**

**相同**：功能类似，都是用来获取一个新的对象实例的。

**不同之处**：原型模式是通过克隆的方案来创造出新对象实例，抽象工厂模式则更关注如何创建产品族，至于如何创建产品族中的每个产品对象实例，则不是很关注。

**两者关注点不同，所以可以联合使用**，例如，在抽象工厂模式里，具体创建产品族中每一个产品的新对象实例时，可以通过该产品原型对象的克隆来创建。

1. **简述工厂方法模式和抽象工厂模式的基本思想，并说明两者有何异同？**

**基本思想：**

工厂方法模式：定义一个创建对象的接口，但由子类决定要实例化的具体类是哪一个。这样，使得一个类的实例化延迟到其子类。这种模式使用继承将对象的创建延迟到子类。

抽象工厂模式：提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。这种模式使用对象组合来创建对象家族。

**异同：**

相同点：都是创建型设计模式，都是用于创建对象，提供一种将对象创建与使用分离的机制。

不同点：工厂方法模式通过继承来创建对象，通常只创建一种产品；抽象工厂模式通过对象组合来创建对象家族，可以创建多种相关联的产品。【工厂模式针对的是一个产品等级结构 ，抽象工厂模式针对的是面向多个产品等级结构的。】

1. **请举例说明什么是简单工厂方法？什么是单件（单例）模式？分别是如何实现的？**

①简单工厂模式可以根据参数的不同返回不同类的实例。【例子见文件B】

②单例模式是确保某一个类只有一个实例，并且能自行实例化并向这个系统提供这个实例。【例子见文件B】

1. **分析并理解饿汉式单例与懒汉式单例的异同。**

【参见《Java设计模式》P111-P114（主要是114），可从延迟加载、线程安全、响应时间等角度进行分析与对比】

1. **什么是双重检查锁定？为什么要进行双重检查锁定？ Java 如何实现双重检查锁定？**

【参见《Java设计模式》P112-P113】

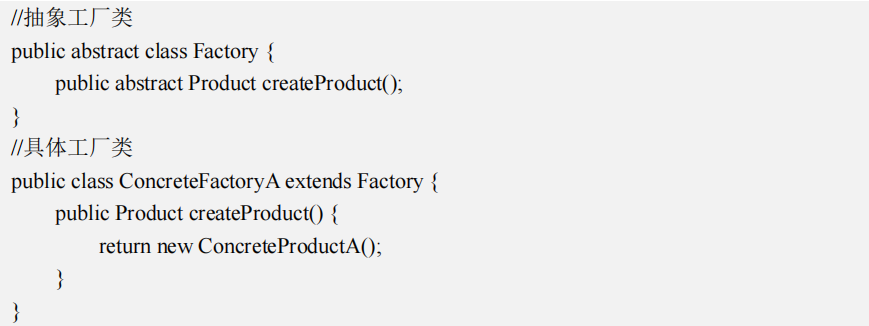
**7、分别采用双重检查锁定和IoDH方式实现8.3节的负载均衡器。**

【双重检查锁定方式实现代码参见《Java设计模式》P113；IoDH 方式实现代码参见P114】

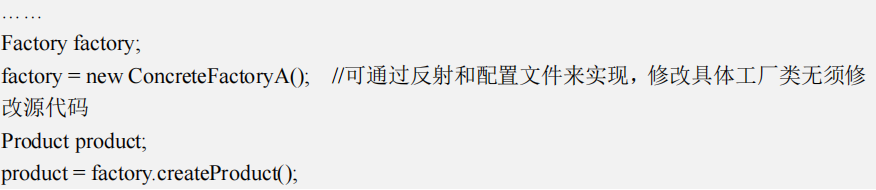
**8、结合工厂方法模式，谈谈对开闭原则的理解。要求在解答时给出开闭原则的定义并结合代码片段进行说明，编程语言不限。**

开闭原则的定义：软件实体应该对扩展开放，对修改关闭。

工厂方法模式代码片段如下：



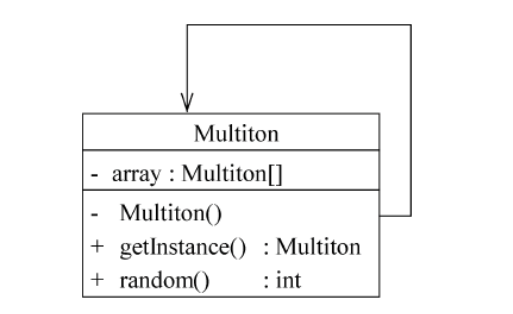
在客户端存在如下代码片段：



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。

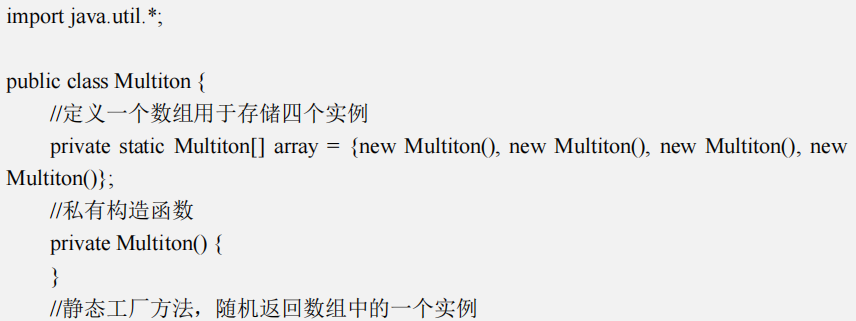
**9、使用Java语言编程实现一个多例模式（Multiton Pattern），确保系统中某个类的对象只能存在有限多个，例如两个或3个。**

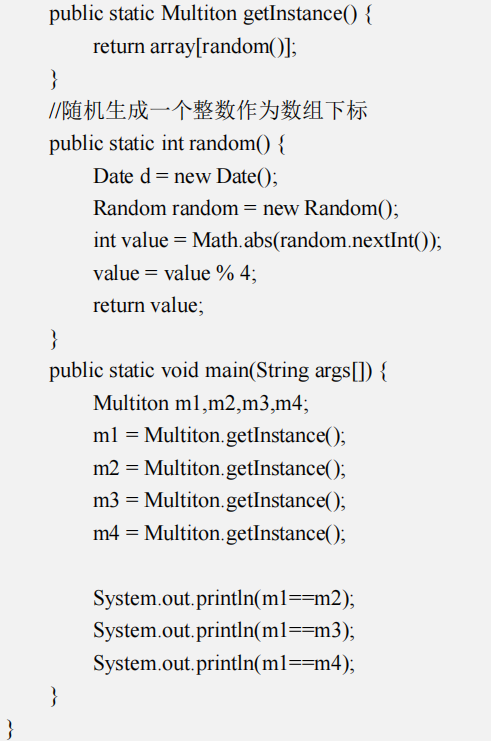
多例模式。参考类图如下所示：



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：





**10、请比较深克隆和浅克隆的异同。**

浅克隆：创建一个新对象，新对象的属性值和原来对象完全相同，对于非基本类型属性，仍指向原有属性所指向的对象。

深克隆：创建一个新对象，属性中引用的其他对象也会被克隆，不再指向原有对象。

异同点：

相同点：深浅克隆都会在堆中新创建一个对象；

不同点：区别在于对象属性引用的对象是否也进行了克隆（递归性的，深的克隆，浅的不克隆）；

**三、结构型模式**

**1、请举例说明哪些情况下可以使用组合模式。**

组合模式适用于以下情况：

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。

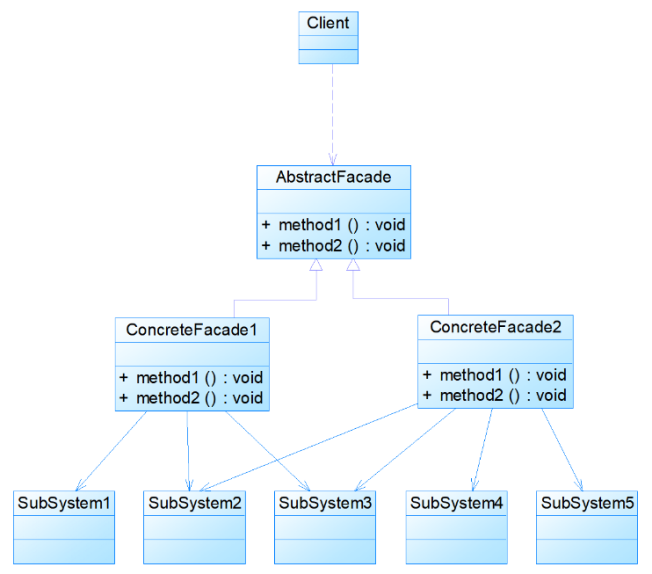
③在一个系统中能够分离出叶子对象和容器对象，而且它们的类型不固定，需要增加一些新的类型。

例如：操作系统中的目录结构、应用软件中的菜单、办公系统中的公司组织结构等等，都可以使用组合模式。

**2、请举例说明在应用标准外观模式时可能产生的问题，以及对应的解决方案。**

【PPT，不过需要补充一个例子】在标准的外观模式结构图中，如果需要增加、删除或更换与外观类交互的子系统类，必须修改外观类或客户端的源代码，这将违背开闭原则。

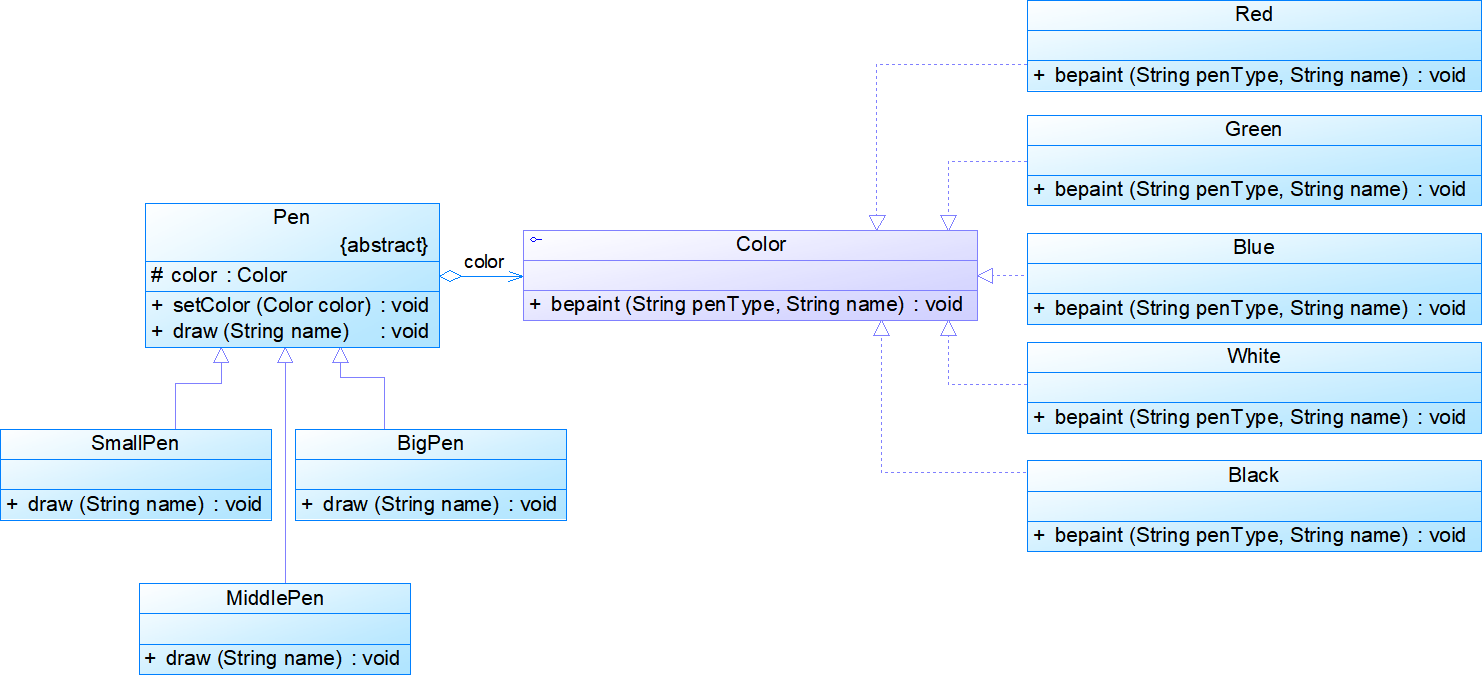
因此可以通过引入抽象外观类来对系统进行改进，在一定程度上可以解决该问题。在引入抽象外观类之后，客户端可以针对抽象外观类进行编程，对于新的业务需求，不需要修改原有外观类，而对应增加一个新的具体外观类，由新的具体外观类来关联新的子系统对象，同时通过修改配置文件来达到不修改任何源代码并更换外观类的目的。



**3、请举例说明桥接模式如何将抽象部分与它的实现部分分离，使它们可以独立地变化。**

与多重/多层继承方案不同，桥接模式将两个独立变化的维度设计为两个独立的继承等级结构，并且在抽象层建立一个抽象关联，该关联关系类似一条连接两个独立继承结构的桥。 通常情况下，我们将具有两个独立变化维度的类的一些普通业务方法和与之关系最密切的维度设计为“抽象类”层次结构（抽象部分），而将另一个维度设计为“实现类”层次结构（实现部分）。

例如：对于毛笔而言，由于型号是其固有的维度，因此可以设计一个抽象的毛笔类，在该类中声明并部分实现毛笔的业务方法，而将各种型号的毛笔作为其子类；颜色是毛笔的另一个维度，由于它与毛笔之间存在一种“设置”的关系，因此我们可以提供一个抽象的颜色接口，而将具体的颜色作为实现该接口的子类。在此，型号可认为是毛笔的抽象部分，而颜色是毛笔的实现部分。



**4、简述组合模式和装饰模式的基本思想，并说明两者有何导同。**

**组合模式**:

基本思想是将对象组合成树形结构表示部分-整体的层次结构，使客户端对单个对象和组合对象的使用具有一致性。例如，在绘图应用中，组合模式允许将单个图形对象和组合图形对象作为一个整体来处理。

**装饰模式**:

基本思想是动态地给对象添加一些额外的职责，就增加功能来说，装饰模式相比生成子类更为灵活。例如，在咖啡订单系统中，装饰模式允许动态地为咖啡对象添加配料，如牛奶和糖。

**异同点**:

相同点: ①都属于结构型模式②两者都用来扩展对象的功能，且不通过生成子类来实现。

不同点: ①组合模式关注的是部分和整体的关系，而装饰模式关注的是给对象动态地添加职责。②组合模式相较于装饰模式，实现对象往往含有树形结构，并且引入了抽象容器构建类使得客户端能以一致的方式处理树形结构中的叶子节点和容器节点。

**5、请具体说明你认为文档编辑器是如何处理大量的字符对象的。**

文档编辑器通常采用**享元模式**来处理大量的字符对象。享元模式通过共享来有效地支持大量细粒度对象，用享元池存相关对象，从而减少内存消耗。例如，对于文档中的每个字符对象，享元模式可以使相同字符共享一个实例，同时保存字符的位置等状态信息。

**6、简述类适配器模式和对象适配器模式的基本思想，并说明两者有何异同？**

**基本思想**

类适配器模式中，适配器类实现了目标抽象类接口并继承了适配者类，并在目标抽象类的实现方法中调用所继承的适配者类的方法；

对象适配器模式中，适配器类继承了目标抽象类并定义了一个适配者类的对象实例，在所继承的目标抽象类方法中调用适配者类的相应业务方法。

**异同**

相同点：都是用来解决接口不兼容问题，使得一个类可以使用另一个类的接口。

不同点：类适配器模式通过继承实现适配，编译时确定适配关系；对象适配器模式通过组合实现适配，运行时可以动态更换被适配对象。

①实现方式不同。类适配器：类适配器通过多重继承实现适配，它继承了目标接口和被适配类；对象适配器：对象适配器通过组合的方式实现适配，它包含一个被适配类的实例，并通过调用该实例的方法来实现目标接口的方法。

②由于适配器类是适配者类的子类，因此可以在适配器类中置换一些适配者的方法，使得适配器的灵活性更强。

③一个对象适配器可以把多个不同的适配者适配到同一个目标，也就是说，同一个适配器可以把适配者类和它的子类都适配到目标接口。

**7、简述外观模式和中介者模式的基本思想，并说明两者有何异同？**

**基本思想**

外观模式提供了一个统一的接口，用来访问子系统中的一群接口。外观模式定义了一个高层接口，使得子系统更加容易使用。它通过简化接口，隐藏子系统的复杂性，让客户端与子系统之间的交互变得更加简洁和清晰。

中介者模式定义了一个中介对象来封装一系列对象之间的交互。中介者通过使各对象不需要显式地相互引用，从而使其耦合松散。各对象通过中介者进行通信，避免了对象之间的直接引用关系，降低了系统的复杂性。

**异同**

相同点：都用于简化对象之间的交互，降低对象之间的耦合度；都通过引入一个中间层（外观或中介者）来控制对象之间的复杂交互。

不同点：

①外观模式多用来封装一个子系统内部的多个模块，目的是向子系统外部提供简单易用的接口，也就是说外观模式封装的是子系统外部和子系统内部模块间的交互；而中介者模式是提供多个平等的同事对象之间交互关系的封装，一般是用在内部实现上。

②外观模式是实现单向的交互，是从子系统外部来调用子系统内部，不会反着来，而中介者模式实现的是内部多个模块间多向的交互。

③中介者模式的目的主要是松散多个模块之间的耦合，把这些耦合关系全部放到中介者中去实现；而外观模式的目的是简化客户端的调用，这点和中介者模式也不同。

**8、变化整体上分为接口变化和实现变化。若这两类变化同时存在，通常应首先采用哪种设计模式分离稳定与变化部分？举例说明如何设计。**

在接口变化和实现变化同时存在的情况下，通常首先采用的是桥接模式（Bridge Pattern）来分离稳定与变化的部分。桥接模式通过将抽象部分（接口）与实现部分分离，使它们可以独立变化，从而更灵活地适应变化需求。

桥接模式的设计思想

桥接模式将类的抽象层次与实现层次分离，使得两者可以独立地扩展和变化。通常通过组合关系，将抽象部分和实现部分分开，使抽象部分依赖一个实现接口，并在实现层次中定义多种实现，从而满足不同需求变化。

示例：绘图应用中的形状和颜色

假设我们在设计一个绘图应用程序，需要绘制各种形状，并且每种形状可以有不同的颜色。在这个需求中，形状和颜色都可能发生变化。桥接模式可以帮助我们将“形状”作为抽象部分，“颜色”作为实现部分，将这两个变化的维度分开处理，使得可以在不影响形状的情况下更换颜色，也可以在不影响颜色的情况下增加新形状。

【奔腾：工厂方法、桥接模式、策略模式都可以。

所谓的变化部分我们可以理解为接口或者类的扩展，而呈现给客户的只是一个统一的接口，用户不必关心内部如何实现，只要每次调用同一个方法，只传不同即可，而针对接口的变化，采用以上结合适配器模式即可转换关键代码部分。】

**9、线性聚集关系是一种现实中常见的逻辑关系，如一个A聚集多个B，一个B聚集多个C，……。使用合成模式可以以统一的接口访问各部分。请给出本例应用合成模式的设计类图。子类聚集父类是该模式的特点之一，请再说出也具有该特点的其它几个模式的名称（至少2个）。**

合成模式就是组合模式的别名。

装饰模式、解释器模式、职责链模式（存疑）

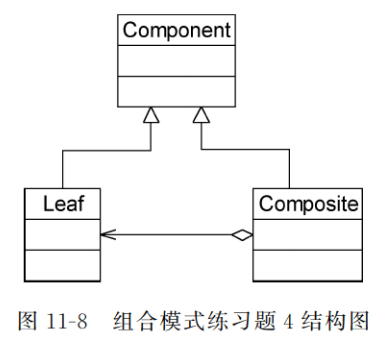
**10、在对象适配器中一个适配器能否适配多个适配者?如果能，应该如何实现？如果不能，请说明原因。如果是类适配器呢？**

在对象适配器中，适配器与适配者之间是关联（组合）关系，一个适配器能够对应多个适配者类，只需要在该适配器类中定义对多个适配者对象的引用即可；在类适配器中，适配器与适配者是继承关系，一个适配器能否适配多个适配者类取决于该编程语言是否支持多重类继承，例如 C++语言支持多重类继承则可以适配多个适配者，而 Java、C#等语言不支持多重类继承则不能适配多个适配者。

**11、如果系统中存在两个以上的变化维度，是否可以使用桥接模式进行处理？如果可以，系统该如何设计？**

桥接模式可以处理存在多个独立变化维度的系统，每一个独立维度对应一个继承结构，其中一个为“抽象类”层次结构，其他为“实现类”层次结构，“抽象类”层次结构中的抽象类与“实现类”层次结构中的接口之间存在抽象耦合关系。

**12、在组合模式结构图中，如果聚合关联关系不是从 Composite 到 Component,而是从 Composite 到Leaf，如图所示，会产生怎样的结果？**



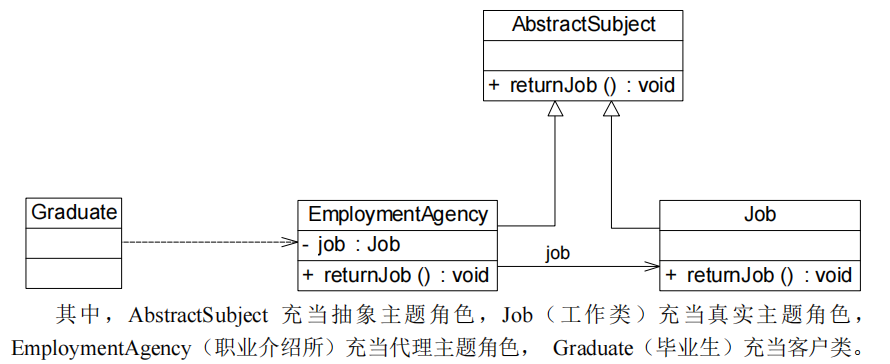
结果：容器（Composite）对象中只能包含叶子（Leaf）对象，不能再继续包含容器对象，导致无法递归构造出一个多层树形结构。

**13、半透明装饰模式能否实现对同一个对象的多次装饰？为什么？**

不能实现对用一个对象的多次装饰。因为在半透明装饰模式中，使用具体装饰类来声明装饰之后的对象，具体装饰类中新增的方法并未在抽象构件类中声明，这样做的优点在于装饰后客户端可以单独调用在具体装饰类中新增的业务方法，但是将导致无法调用到之前装饰时新增的方法，只能调用到最后一次装饰时具体装饰类中新增加的方法，故对同一个对象实施多次装饰没有任何意义。

**14、毕业生通过职业介绍所找工作，请问该过程蕴含了哪种设计模式，绘制相应的类图。**

**代理模式**。参考类图如下所示：



**四、行为型模式**

**1、请给出一个适合使用策略模式的示例场景，并说明与不使用策略模式相比，有哪些优点？**

在软件系统中，要实现如查找/排序的功能，有许多算法可以实现。一种常用的方法是硬编码在一个类中，如需要提供多种查找算法，可以将这些算法写到一个类中，在该类中提供多个方法，每一个方法对应一个具体的查找算法；当然也可以将这些查找算法封装在一个统一的方法中，通过if…else…等条件判断语句来进行选择。这两种实现方法我们都可以称之为硬编码，如果需要增加一种新的查找算法，需要修改封装算法类的源程序，更换查找算法，也需要修改客户端调用代码。在这个算法类中封装了大量查找算法，该类代码将较复杂，维护较为困难。

使用策略模式，定义一系列算法，将每一个算法封装起来，并让它们可以相互替换，可以让算法独立于使用它的客户而变化，提供了对“开闭原则”的完美支持，提供了管理相关的算法族的办法，提供了一种算法的复用机制，提供了一种可以替换继承关系的办法，也可以避免多重条件选择语句。

**2、请给出一个适合使用状态模式的示例场景，并说明与不使用状态模式相比，有哪些优点？**

状态模式例子：水，它可以凝固成冰，也可以受热蒸发后变成水蒸汽，水可以流动，冰可以雕刻，蒸汽可以扩散。在不同的状态有不同的名字以及不同的行为。所以最适合状态模式。

状态模式优点:

①封装了状态的转换规则，在状态模式中可以将状态的转换代码封装在环境类或者具体状态类中，可以对状态转换代码进行集中管理，而不是分散在一个个业务方法中；

②将所有与某个状态有关的行为放到一个类中，只需要注入一个不同的状态对象即可使环境对象拥有不同的行为；③允许状态转换逻辑与状态对象合成一体，而不是提供一个巨大的条件语句块，状态模式可以让我们避免使用庞大的条件语句来将业务方法和状态转换代码交织在一起；

④可以让多个环境对象共享一个状态对象，从而减少系统中对象的个数。

**3、简述状态模式和策略模式的基本思想，并说明两者有何异同？**

**基本思想：**

状态模式：允许对象在其内部状态发生改变时改变其行为。状态模式将与状态相关的行为封装到不同的状态对象中，使得状态的转换和状态相关行为的变化独立于对象本身。

策略模式：定义一系列算法，并将每个算法封装起来，使它们可以互相替换。策略模式使得算法在客户端代码中独立变化，而不影响使用这些算法的客户端。

**异同：**

相同点：都是行为型模式，都涉及将行为封装到独立的类中，并且可以动态地改变这些行为。

不同点：

①状态模式是基于对象状态的变化来改变行为，关注的是状态转换和行为变化的关系；策略模式是基于算法的选择来改变行为，关注的是在不同算法之间进行选择。

②使用策略模式时，客户端需要知道所选的具体策略是哪一个，而使用状态模式时，客户端无须关心具体状态，环境类的状态会根据用户的操作自动转换。

③策略模式中，具体策略类无须关心环境类；而状态模式中具体状态往往需要维护一个环境类的引用，以便通过该引用实现状态的切换，因此环境类和状态类之间存在一种双向的关联关系。

④可以通过环境类状态的个数来决定是使用策略模式还是状态模式。如果系统中某个类的对象存在多种状态，不同状态下行为有差异，而且这些状态之间可以发生转换时使用状态模式；如果系统中某个类的某一行为存在多种实现方式，而且这些实现方式可以互换时使用策略模式。

**4、现在大多数软件都有撤销（Undo）和重做（Redo）的功能，请问这种功能一般是采用什么设计模式实现的，并举例说明是如何实现的。**

【类图和代码详见文件B“命令模式-文本编辑器”】

撤销（Undo）和重做（Redo）功能通常使用命令模式（Command Pattern）来实现。【主要是命令模式，可以结合备忘录模式】

在命令模式中，可以通过调用一个命令对象的execute()方法来实现对请求的处理，如果需要撤销(Undo)请求，可通过在命令类中增加一个逆向操作来实现。如果需要重做(redo)请求，则可以重新调用命令对象对应的execute()方法。

具体举例如下：

①执行命令：

·当执行一个命令（如添加操作）时，将这个命令对象保存到 undo 栈中。

·该命令执行完成后，将其从 redo 栈中移除（如果存在的话）。

②撤销（Undo）操作：

·从 undo 栈中取出最近执行的命令对象。

·调用该命令对象的 undo 方法，这通常是原操作的反向操作（如将添加操作改为删除操作）。

·将该命令对象保存到 redo 栈中。

③重做（Redo）操作：

·从 redo 栈中取出最近撤销的命令对象。

·调用该命令对象的 execute 方法，重新执行该命令。

·将该命令对象保存回 undo 栈中。

除了通过一个逆向操作来实现撤销(Undo)外，还可以通过保存对象的历史状态来实现撤销，后者可使用备忘录模式(Memento Pattern)来实现。【在命令操作的时候记录下操作前后的状态，然后在命令撤销和重做的时候，直接使用相应的备忘录对象来恢复状态就可以了。】

**5、为达到某些设计目的，设计时可以同时组合使用多个设计模式。请举出一个同时组合使用策略模式和对象适配器模式。要求给出例子的应用场景、设计方案。**

【详见文件B模式联用】

**6、Java语言中的异常处理机制是职责链模式的一个应用实例，编写一个包含多个catch子句的程序，理解异常处理的实现过程，并判断此处使用的是纯的职责链模式还是不纯的职责链模式。**

【答】异常处理使用的是不纯的职责链模式，存在一个 try 语句中的异常最终没有被任何 catch语句处理的情况，即一个请求可能最终不被任何处理者对象接收并处理。

**7、在策略模式中一个环境类Context能否对应多个不同的策略等级结构？如何设计？**

【答】一个环境类可以对应多个不同的策略等级结构。在环境类中维持对每一个策略等级结构中抽象策略类的引用即可，在程序运行时再分别从每一个策略等级结构中选择一个具体策略对象注入到环境类中。

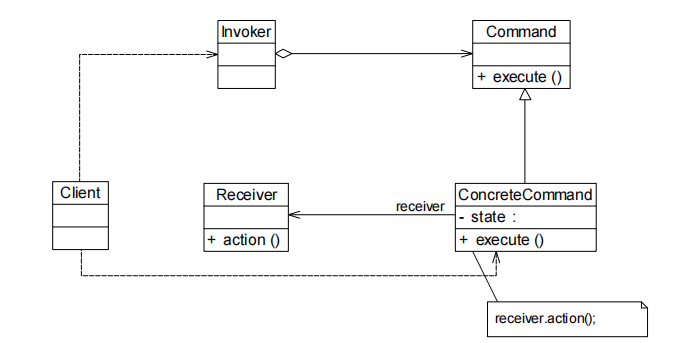
**8、在模板方法模式中，钩子方法如何实现子类控制父类的行为？**

【答】由于钩子方法通常返回一个 boolean 类型的值，并以此来判断是否执行某一基本方法，因此在子类中可以通过覆盖钩子方法来决定是否执行父类中的某一方法，从而实现子类对父类 行为的控制。【在模板方法模式中，由于面向对象的多态性，子类对象在运行时将覆盖父类对象，子类中定义的方法也将覆盖父类中定义的方法，因此程序在运行时，具体子类的基本方法将覆盖父类中定义的基本方法，子类的钩子方法也将覆盖父类的钩子方法，从而可以通过在子类中实现的钩子方法对父类方法的执行进行约束，实现子类对父类行为的反向控制。】

**9、电视机遥控器的设计原理中蕴含了哪些设计模式，列举两个，绘制这两种设计模式的结构图并简单论述其适用场景。**

电视机遥控器蕴含了命令模式和迭代器模式等设计模式。

命令模式结构图：



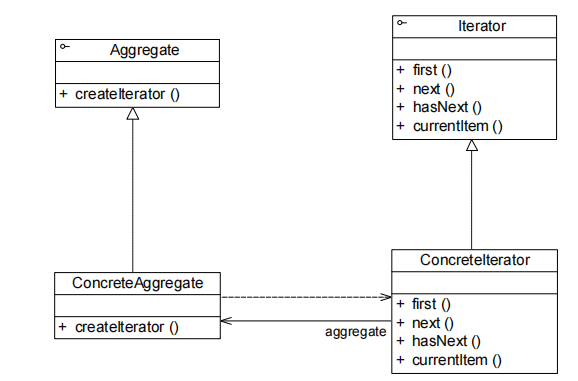
命令模式适用场景：

①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；

②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；

③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：



迭代器适用场景：

①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；

②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。