

NEW PLAN

软考高级架构师

• — 段 新 征 程 •





大纲介绍

面向对象设计

面向对象相关概念

UML

用例图

类图

顺序图

设计模式

创建型

结构型

行为型

考试方式：上午选择题会有12分，下午一般会出2道大题，30分



设计模式-结构型

结构型设计模式：关注如何将对象和类组合成更大的结构，以便更好地实现系统的整体结构。这些模式有助于解决对象之间的组合、接口和继承等关系，从而提高代码的可扩展性和可维护性

适配器模式（Adapter Pattern）：

适配器模式允许两个不兼容的接口协同工作。就像使用电源适配器将外国电器插头适配到国内插座一样，适配器模式可以使不同的类一起工作。

生活中的例子：

想象你有一台笔记本电脑，但只有一个USB-C接口，而你的打印机只支持USB-A接口。你可以使用USB-C到USB-A的适配器，使两者兼容。

桥接模式（Bridge Pattern）：

桥接模式分离了一个对象的抽象部分和具体部分，使它们可以独立地变化。这个模式就像一座桥，将两个独立的领域连接起来。

生活中的例子：

想象你有一辆汽车，汽车的品牌（如奥迪、宝马）和颜色（如红色、蓝色）是两个独立的变化维度。桥接模式允许你将品牌和颜色抽象出来，使你可以轻松地组合不同品牌和颜色，例如创建一个红色的奥迪或蓝色的宝马。



设计模式-结构型

组合模式（Composite Pattern）：

组合模式允许将对象组织成树状结构，使单个对象和组合对象都可以一致地对待。这就像将多个文件夹和文件组织成文件系统树一样。

生活中的例子：

在操作系统中，文件夹可以包含文件和其他文件夹，创建了一个树状的组织结构，你可以对文件夹和文件执行相似的操作，如复制、删除等。

装饰器模式（Decorator Pattern）：

装饰器模式允许动态地为对象添加新的功能，而无需修改其源代码。这就像在你的家中不断地添加新的家具或装饰来改善它的外观和功能。

生活中的例子：

如果你有一台智能音响，可以通过添加新的语音助手技能（如天气查询或音乐播放）来扩展其功能，而不需要改变音响的核心设计。

外观模式（Facade Pattern）：

外观模式提供了一个简化的接口，用于访问复杂系统中的一组接口。这个模式就像建筑物的外立面，隐藏了建筑内部复杂的结构，使外部用户可以轻松访问。

生活中的例子：

在计算机操作系统中，你通常使用一个图形用户界面（GUI），它提供了一个易于使用的外观，隐藏了底层操作系统的复杂性。你可以点击图标、打开应用程序，而无需了解操作系统的内部工作原理。



设计模式-结构型

享元模式（Flyweight Pattern）：

享元模式旨在最小化内存或计算开销，通过共享尽可能多的相似对象来实现。这就像在共享办公空间中租用一个工作区，多个人可以共享同一空间，减少了资源浪费。

生活中的例子：

在图像编辑软件中，当多个图像元素共享相同的颜色或图案时，可以使用享元模式来减少内存占用。相同的颜色或图案可以被多个图像元素引用，而不是为每个元素都存储一份相同的数据。

代理模式（Proxy Pattern）：

代理模式允许一个对象代表另一个对象进行控制访问。就像你聘请一个房产经纪人代表你购买房产，代理模式可以控制对另一个对象的访问。

生活中的例子：

在计算机网络中，代理服务器充当客户端和目标服务器之间的中间层，用于缓存、过滤或加速请求，以提供更好的访问控制和性能。



设计模式

速记口诀：外侨组员带配饰

结构型设计 模式	定义	记忆关键字
Adapter 适配器模式	将一个类的接口转换成用户希望得到的另一种接口。它使原本不相容的接口得以协同工作	转换，兼容接口
Bridge 桥接模式	将类的抽象部分和它的实现部分分离开来，使它们可以独立的变化	抽象和实现分离
Composite 组合模式	将对象组合成树型结构以表示“整体-部分”的层次结构，使得用户对单个对象和组合对象的使用具有一致性	整体-部分，树形结构
Decorator 装饰模式	动态的给一个对象添加一些额外的职责。它提供了用子类扩展功能的一个灵活的替代，比派生一个子类更加灵活	附加职责
Façade 外观模式	定义一个高层接口，为子系统中的一组接口提供一个一致的外观，从而简化了该子系统的使用	对外统一接口
Flyweight 享元模式	提供支持大量细粒度对象共享的有效方法	细粒度，共享
Proxy 代理模式	为其他对象提供一种代理以控制这个对象的访问	代理控制



设计模式-行为型

行为型设计模式：关注对象之间的通信、职责分配和算法的交互方式。它们帮助我们更好地管理对象之间的关系，使系统更具灵活性和可维护性

责任链模式（Chain of Responsibility Pattern）：

责任链模式就像是传递请求。多个对象依次尝试处理请求，直到有一个对象能够处理为止。

生活中的例子： 在组织中，员工的请假申请可能需要经过多个级别的审批，每个级别的主管可以决定是否批准请假。

命令模式（Command Pattern）：

命令模式就像使用遥控器来控制设备。你将命令封装在遥控器按钮中，然后可以随时执行命令。

生活中的例子： 电视遥控器是命令模式的经典示例。你可以使用遥控器来执行命令，如打开电视、切换频道或调整音量。

解释器模式（Interpreter Pattern）：

解释器模式用于处理语言解释和编译器等领域，它定义了一种语言的语法表示，并提供了解释器来解释这种语法。

生活中的例子： 在编程中，正则表达式就是解释器模式的一个实例，用于匹配和解释字符串模式。



设计模式-行为型

迭代器模式（Iterator Pattern）：

迭代器模式就像是遍历集合。它提供了一种顺序访问集合元素的方法，而无需暴露集合的内部结构。

生活中的例子： 在编程中，你可以使用迭代器来遍历数组、列表或集合中的元素，而无需知道底层数据结构。

中介者模式（Mediator Pattern）：

中介者模式就像是一个中间人协调多个对象之间的交互。它减少了对象之间的直接通信，降低了耦合度。

生活中的例子： 在一个团队中，项目经理可以充当中介者，协调团队成员之间的合作和沟通，以确保项目的顺利进行。

备忘录模式（Memento Pattern）：

备忘录模式就像是保存和还原状态。它允许你保存对象的状态，以便将来可以还原到先前的状态。

生活中的例子： 在文本编辑器中，你可以使用撤销和重做功能，这就是备忘录模式的应用。你可以撤销到之前的状态，然后再次重做。

观察者模式（Observer Pattern）：

观察者模式就像订阅通知。多个观察者（订阅者）订阅主题（发布者），当主题有新信息时，观察者会自动接收通知。

生活中的例子： 许多社交媒体平台使用观察者模式，当你关注某个用户或主题时，系统会将他们的更新信息发送给你。



设计模式-行为型

状态模式（State Pattern）：

状态模式就像人的不同情绪状态。你可以处于高兴、生气或伤心等不同的状态，每种状态下的行为可能不同。

生活中的例子： 在自动售货机中，售货机可能有不同的状态，如待机、售卖中、缺货等。每种状态下，售货机的行为都不同。

策略模式（Strategy Pattern）：

策略模式就像选择不同的解决方案来解决同一个问题。你可以根据情况选择不同的策略，以实现相同的目标。

生活中的例子： 在旅行规划中，你可以选择不同的交通工具（策略），如汽车、火车或飞机，来达到相同的目的地。

模板方法模式（Template Method Pattern）：

模板方法模式就像是定义了一个算法的框架，但允许子类实现其中的一些步骤。

生活中的例子： 在烹饪中，制作一道菜的过程可能包括一些固定的步骤，如准备、烹饪和装盘，但每道菜的具体步骤和材料可能不同。

访问者模式（Visitor Pattern）：

访问者模式就像是访问不同类型的元素。你可以定义不同的访问者来执行不同类型元素的操作。

生活中的例子： 在图书馆中，你可能有不同类型的书籍，如小说、科普书和艺术书。不同类型的书籍可能需要不同类型的读者来浏览和理解。

设计模式

速记口诀：观摩（模）对（迭）策，责令解放（访），戒（介）忘台（态）

行为型设计模式	定义	记忆关键字
Chain of Responsibility 职责链模式	通过给多个对象处理请求的机会，减少请求的发送者与接收者之间的耦合。将接收对象链接起来，在链中传递请求，直到有一个对象处理这个请求	传递请求、职责、链接
Command 命令模式	将一个请求封装为一个对象，从而可用不同的请求对客户进行参数化，将请求排队或记录请求日志，支持可撤销的操作	日志记录、可撤销
Interpreter 解释器模式	给定一种语言，定义它的文法表示，并定义一个解释器，该解释器用来根据文法表示来解释语言中的句子	解释器，虚拟机
Iterator 迭代器模式	提供一种方法来顺序访问一个聚合对象中的各个元素而不需要暴露该对象的内部表示	顺序访问，不暴露内部
Mediator 中介者模式	用一个中介对象来封装一系列的对象交互。它使各对象不需要显式地相互调用，从而达到低耦合，还可以独立的改变对象间的交互	不直接引用

行为型设计模式	定义	记忆关键字
Memento 备忘录模式	在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态，从而可以在以后将该对象恢复到原先保存的状态	保存，恢复
Observer 观察者模式	定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动更新	通知、自动更新
State 状态模式	允许一个对象在其内部状态改变时改变它的行为	状态变成类
Strategy 策略模式	定义一系列算法，把它们一个个封装起来，并且使它们之间可互相替换，从而让算法可以独立于使用它的用户而变化	算法替换
Template Method 模板方法模式	定义一个操作中的算法骨架，而将一些步骤延迟到子类中，使得子类可以不改变一个算法的结构即可重新定义算法的某些特定步骤	
Visitor 访问者模式	表示一个作用于某对象结构中的各元素的操作，使得在不改变各元素的类的前提下定义作用于这些元素的新操作。	数据和操作分离



真题

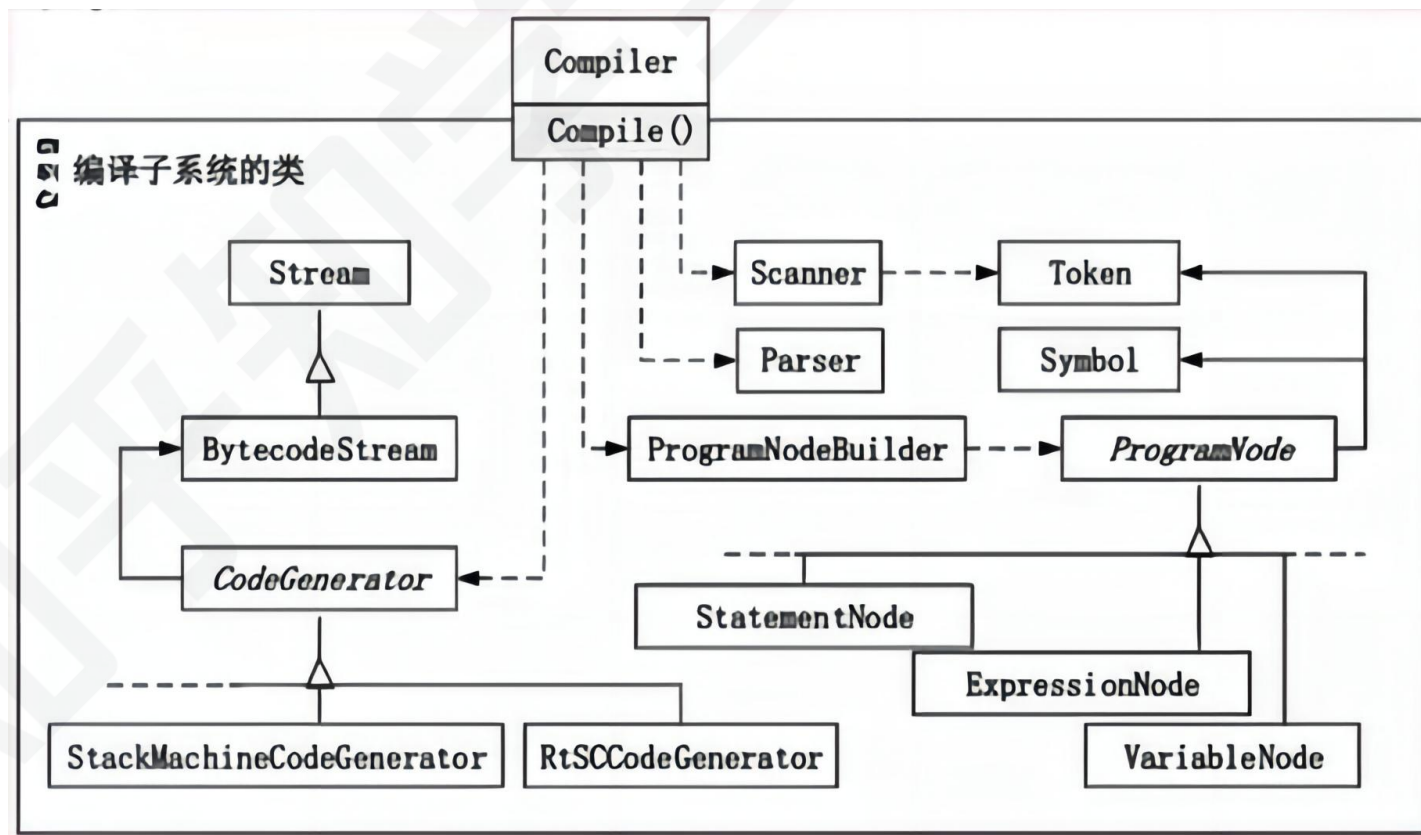
【2014】下图所示为(44)设计模式，适用于(45)。

(44)

- A.适配器(Adapter)
- B.责任链(Chain of Responsibility)
- C.外观(Facade)
- D.桥接(Bridge)

(45)

- A.有多个对象可以处理一个请求，在运行时刻自动确定由哪个对象处理
- B.想使用一个已经存在的类，而其接口不符合要求
- C.类的抽象和其实现之间不希望有一个固定的绑定关系
- D.需要为一个复杂子系统提供一个简单接口



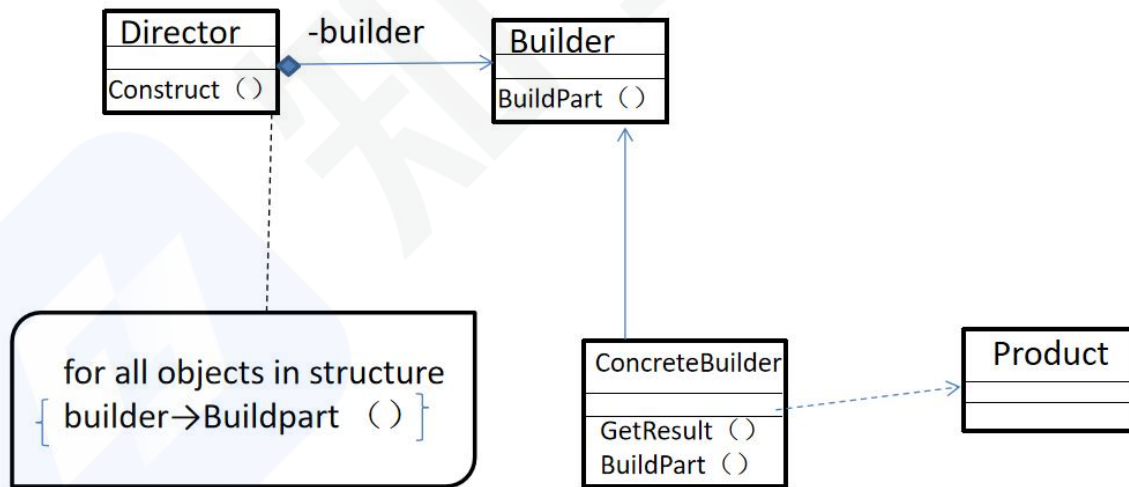
真题

【2014】下图所示为()设计模式, 适用于()。

A.抽象工厂(Abstract Factory)
C.工厂方法(Factory Method)

B.生成器(Builder)
D.原型(Prototype)

- A.一个系统要由多个产品系列中的一个来配置时
B.当一个类希望由它的子类来指定它所创建的对象时
C.当创建复杂对象的算法应该独立于该对象的组成部分及其装配方式时
D.当一个系统应该独立于它的产品创建、构成和表示时





真题

【2015】(45)设计模式能够动态地给一个对象添加一些额外的职责而无需修改此对象的结构；(46)设计模式定义一个用于创建对象的接口，让子类决定实例化哪一个类；欲使一个后端数据模型能够被多个前端用户界面连接，采用(47)模式最适合。

(45)

- A.组合(Composite)
- B.外观(Facade)
- C.享元(Flyweight)D
- D.装饰器(Decorator)

(46)

- A.工厂方法(Factory Method)
- B.享元(Flyweight)
- C.观察者(Observer)
- D.中介者(Mediator)

(47)

- A.装饰器(Decorator)
- B.享元(Flyweight)
- C.观察者(Observer)
- D.中介者(Mediator)



真题

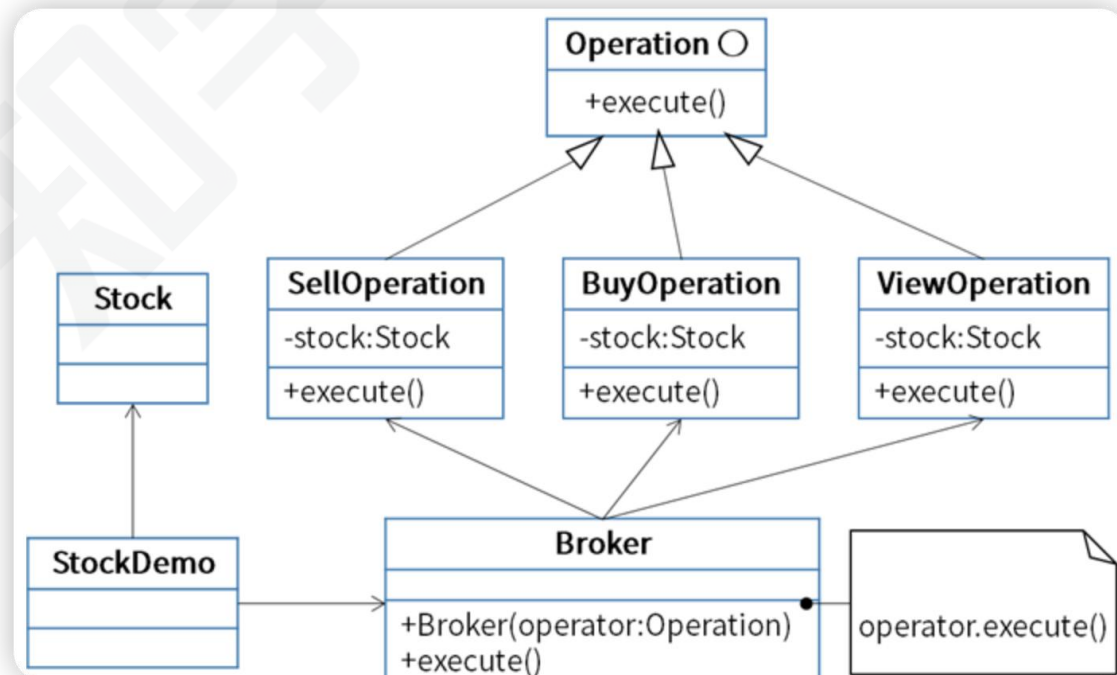
【2021】股票交易中，股票代理(Broker)根据客户发出的股票操作指示进行股票的买卖操作，设计如下所示类图。该设计采用（ ）模式将一个请求封装为一个对象，从而使得以用不同的请求对客户进行参数化；对请求排队或记录请求日志，以及支持可撤销的操作，其中，（ ）声明执行操作的接口。该模式属于（ ）模式，该模式适用于（ ）

- (A) 命令 (Command) (B) 观察者 (Observer)
(C) 状态 (State) (D) 中介者 (Mediator)

- (A) Operation
(B) sellOperation/BayOperation/ViewOperation
(C) Broker (D) Stock

- (A) 结构类型 (B) 结构型对象
(C) 创建类型 (D) 行为型对象

- (A) 一个对象必须通知其他对象，而它又不能假定其他对象是谁
(B) 抽象出需要执行的动作以参数化某对象
(C) 一个对象的行为决定于其状态且必须在运行时刻根据状态改变行为
(D) 一个对象引用其他很多对象并且直接与这些对象通信，导致难以复用该对象



T H E E N D

功不唐捐，玉汝于成！

开 启 新 征 程

