**System Analysis and Design**

**Homework Assignment 6 (due July 5)**

1. What is Responsibility-Driven Design? How can we do Responsibility-Driven Design?（胡洁珏）

**什么是职责驱动设计？**

答：职责驱动设计指思考软件对象设计和大型构架时，考虑其职责、角色和协作的一种方式。在RDD中，我们认为软件对象具有职责，即对其所作所为的抽象。驱动设计是通过为每个类分配明确定义的职责来设计类的过程。 此过程可用于确定哪个类应该实现应用程序功能的哪个部分。

**怎样进行职责驱动设计？**

答：RDD也包括了协作的思想，在做RDD时，要考虑对象间的协作关系。职责借助于方法来实现，该方法既可以单独动作，也可以与其他方法和对象协作。

1. Why we say that RDD is a general metaphor for thinking about OO software design?（胡洁珏）

答：RDD是思考OO软件设计的一般性隐喻。把软件对象想象成具有某种职责的人，他要与其他人协作以完成工作。RDD使我们把OO设计看作使职责对象进行协作的共同体。

1. What is design pattern? Why design patterns are important in software design?（胡若晴）

**什么是设计模式？**

答：在软件工程中，软件设计模式是对软件设计中给定上下文中常见问题的通用、可重用解决方案。它不是可以直接转换为源代码或机器代码的完成设计。它是关于如何解决可以在许多不同情况下使用的问题的描述或模板。设计模式是程序员在设计应用程序或系统时可以用来解决常见问题的形式化最佳实践。

**为什么设计模式在软件设计中十分重要？**

答：作为软件开发人员，我们经常通过一些属性来评估我们的代码，比如我们的代码有多干净、有表现力、占用更少的内存和速度。但是我们经常忽略的最重要的问题是我们需要能够在以后轻松更改任何内容。我们的代码应该足够灵活，以便更改成本不会太高。所以设计模式是涵盖这些属性的最佳实践。设计模式的核心包括以下 5 条规则：

1. **它们是经过验证的解决方案**

因为设计模式经常被许多开发人员使用，所以我们可以肯定它们是有效的。不仅如此，我们还可以确定它们经过多次修订，并且可能实施了优化。

1. **它们易于重复使用**

设计模式记录了一个可重用的解决方案，该解决方案可以修改以解决多个特定问题。因为它们与特定问题无关。

例如，考虑迭代器设计模式，它可以在整个 STL 中重用，尽管容器和算法发生了变化。迭代器就像容器和算法之间的粘合剂一样工作。

**C．它们富有表现力**

设计模式可以非常优雅地解释大型解决方案。例如，访问者模式用于对一系列/一组类执行新操作。因此，标准库采用了这种单一功能即[std::visit](https://en.cppreference.com/w/cpp/utility/variant/visit)算法的模式。

**D．它们简化了沟通**

当开发人员熟悉设计模式时，他们可以更轻松地相互交流关于给定问题的潜在解决方案。

如果你与多个开发人员团队中的同事一起工作，请就设计模式与他们达成一致，因为它们可以帮助你更好地解决问题。另外对于软件的维护，你应该遵循这样的程序，这样你的维护操作会更快更高效。

1. **它们防止了重构代码的需要**

如果应用程序是用设计模式编写的，通常情况下您以后不需要重构代码，因为将正确的设计模式应用于给定的问题已经是最佳解决方案。

如果这样的解决方案是更新，它们可以被任何优秀的软件开发人员无缝应用，并且不会引起任何问题。

1. What are the problems and the corresponding solutions of the following GRASP design patterns? ( you may just select 4 of them to answer the question.)（丁维力）

Creator  
Information Expert  
Low Coupling  
Controller  
High Cohesion  
Polymorphism  
Indirection  
Pure Fabrication  
Protected Variations

**答：**

**Creator：**

问题：谁应该负责产生类的实例？

解决方案：如果符合下面的一个或者多个条件，则可将创建类A实例的职责分配给类B：

B包含A；

B聚合A；

B拥有初始化A的数据并在创建类A的实例时将数据传递给类A；

B记录A的实例；

B频繁使用A。

      此时，我们称类B是类A对象的创建者。如果符合多个条件，类B聚合或者包含类A的条件优先。

**Information Expert：**

问题：给对象分配职责的通用原则是什么？

解决方案：将职责分配给拥有履行一个职责所必需信息的类，即信息专家。

**Low Coupling：**

问题：怎样支持低的依赖性，减少变更带来的影响，提高重用性？

解决方案：分配一个职责，使得保持低耦合度。类A和类B之间的耦合关系体现如下：A具有一个B类型的属性；A调用B的方法；A的方法包含对B的引用，如方法参数类型为B或返回类型为B；A是B的直接或者间接子类；B是一个接口，A实现了该接口。低耦合模式鼓励在进行职责分配时不增加耦合性，从而避免高耦合可能产生的不良后果。在进行类设计时，需要保持类的独立性，减少类变更所带来的影响，它通常与信息专家模式和高内聚模式一起出现。为了达到低耦合，我们可以通过如下方式对设计进行改进：

在类的划分上，应当尽量创建松耦合的类，类之间的耦合度越低，就越有利于复用，一个处在松耦合中的类一旦被修改，不会对关联的类造成太大波及；

在类的设计上，每一个类都应当尽量降低其成员变量和成员函数的访问权限；

在类的设计上，只要有可能，一个类型应当设计成不变类；

在对其他类的引用上，一个对象对其他对象的引用应当降到最低。

**Controller：**

问题：谁应该负责处理一个输入系统事件？

解决方案：把接收或者处理系统事件消息的职责分配给一个类。这个类可以代表：

整个系统、设备或者子系统；

系统事件发生时对应的用例场景，在相同的用例场景中使用相同的控制器来处理所有的系统事件。

1. What is Use Case Realization? What is the prime input to use case realizations? What are the main tasks we need do during use cage realization ?（李浩天、杨宗恺）

**什么是用例实现？**

答：用例实现基于协作对象，如何在设计模型中实现某个用例。在面向对象理论体系中，协作被定义为“多个对象为了完成某个目标而进行的交互”。更确切的说是实现某个用例中的某个场景。因此也叫场景实现。

**用例实现的主要输入是什么？**

答：用例。

**在实现过程中我们需要做的主要任务是什么？**

答：找到用例中的对象，然后找到它们的创建者，确定它们之间的交互，确定可能应用的模式。

1. Why should it be based on the GRASP patterns to make the choices and decisions during the design of a use case realization with objects?（李浩天）

答：

因为用例实现的过程需要构建UML类图，因此需要思考每个类在用例实现中起到的作用和职责，需要合理地调度和分配任务给各个类，并且保证各个类之间的关系恰当，持续高效地运作。而GRASP模式正是决定这个系统有多少对象，实现一个场景需要什么类来协作，每个类要承担什么职责地指导原则，因此需要遵循其指导。

1. What artifacts may be the UP implementation model? What artifacts should be used as the inputs for the code generation process ?（崔子潇）

**什么工件可能是UP实现模型？**

答：UP的实现模型需要类图、DCD、交互图（通信图或顺序图）。

**应该使用哪些工件作为代码产生过程的输入？**

答：类图中类之间的关联关系、交互图中传递的消息应当成为代码产生过程中的输入。

1. During the mapping design to code, what kinds of translations and what creative work we should do?（崔子潇）

**答：**

代码的创建需要类和接口的定义以及方法的定义。因此，要由DCD和类图来创建接口和类，由交互图创建方法。DCD中描述了类和接口的名称、超类、操作的特征标记和类的属性等，可以借助这些完成类的定义。交互图中的一系列消息可以转换成方法定义中的一系列语句。

有些类具有一对多的关系，这时需要采用集合类的方法（数组等数据结构）来实现。有些类需要设置异常和错误处理，在设计和建模中的异常处理映射到代码中。此外，还要关注实现的顺序，要从耦合度最低的类开始，逐渐实现耦合度更高的类。最后，还应注意代码与测试相结合。

1. What is test-driven development ? What basic rhythm we have suggested for test-first development?（莫振威）

**什么是测试驱动开发？**

答：测试驱动开发（test-driven development）是迭代和敏捷XP方法提倡的优秀实践，（与大部分XP实践一样）也适用于UP，该实践也称为测试优先开发。

**对于测试优先开发，我们建议的基本节奏是什么？**

答：TDD的基本规律是编写一小段测试代码，然后再编写一小段产品代码，保证其通过测试，然后再编写更多的测试代码，以此类推。

关键点：首先编写测试，然后再编写预计要测试的代码。

1. What are the goals of refactory? List several refactoring methods we may usually use in our coding process.（莫振威）

**重构的目标是什么？**

答：重构（refactory）是重写或重新构建已有代码的结构化和规律性方法，但不会改变已有代码的外在行为，而是采用一系列少量转换的步骤，并且每一步都结合了重新执行的测试。

重构的活动和目标只是为了得到优秀的代码：

* 去除冗余代码
* 改善清晰度
* 使过长的方法变得较短
* 去除硬编码的字面常量……

**列出几种我们在编程过程中常用的重构方法。**

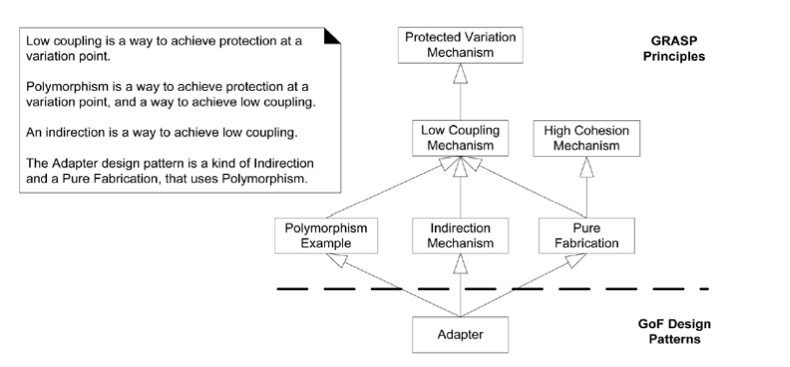
答：我们在编程过程中通常使用的几种重构方法：

|  |  |
| --- | --- |
| 重构 | 描述 |
| 提炼方法（extract method） | 将较长的方法转换为短小的方法，其中将原有方法中的部分内容分解为私有的帮助方法（helper method） |
| 提炼常量（extract constant） | 使用常量变量（constant variable）替换字符变量 |
| 引入解释变量（提炼局部变量的特化） | 将表达式的部分或完整结果置入临时变量，该变量的名字应该能够说明其目的 |
| 使用工厂方法代替构造器调用 | 例如在Java中，调用创建对象的帮助者方法（helper method）来代替对新操作和构造器的调用 |

1. Explain how GoF adapter pattern support GRASP *Protected Variations* in terms of GRASP terminology.（丁维力）

**答：**

适配器通过使用应用接口和多态性的间接对象来更改外部接口或第三方包，依次支持受保护变量。



1. What are the connections between Factory pattern and GRASP pure Fabrication pattern?（胡若晴）

**答：**

GRASP在创造者模式下，创建对象是面向对象系统中最普遍的活动之一，因此，确定一个分配创建对象的通用职责非常重要。如果职责分配合理,设计就能降低耦合，提高设计的清晰度性和重用性。通常情况下，如果对象的创建过程不是是很复杂,则根据上述原则，由使用的类来创建对象。但是如果创建过程非常复杂，而且可能需要重复使用对象实例或者对象从外部注入一个对象实例，此时，可以委托一个专门的工厂类来辅助创建对象。创建者需与各种工厂模式(简单工厂模式、工厂方法模式和抽象工厂模式)相对应。

1. For solving the external services with varying interfaces problem, what patterns may be used in your design?（陈家龙）

答：

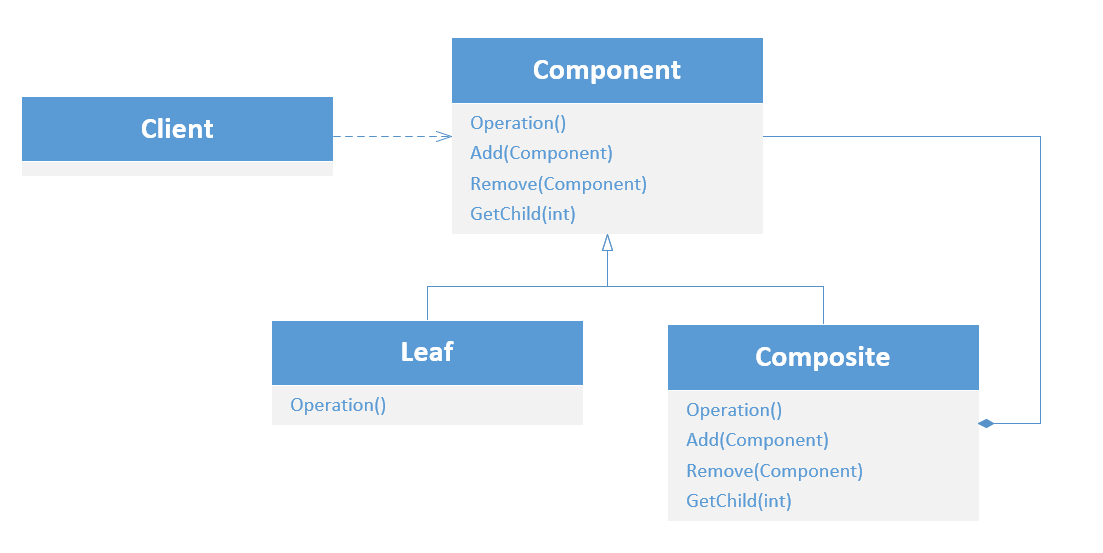
由于外部服务的接口是多样的，而内部已经实现的接口是唯一的，因此可以首先考虑**适配器模式**，为每一种外部接口定义一种适配器实现。适配器分为两种：类适配器和对象适配器，前者通过继承adaptee 类，并实现目标方法的方式实现；后者通过委派的方式实现，即每一个适配器的实例都持有 adaptee 对象。

但是使用适配器模式的问题在于谁来创建适配器，如果由当前领域对象来创建适配器，就会使得模型的内聚降低。此时可以通过使用**简单工厂模式**解决这个问题。简单工厂模式专门用于创建实例，有利于分离职责，隐藏创建时的复杂逻辑，还可以引入存储管理机制。

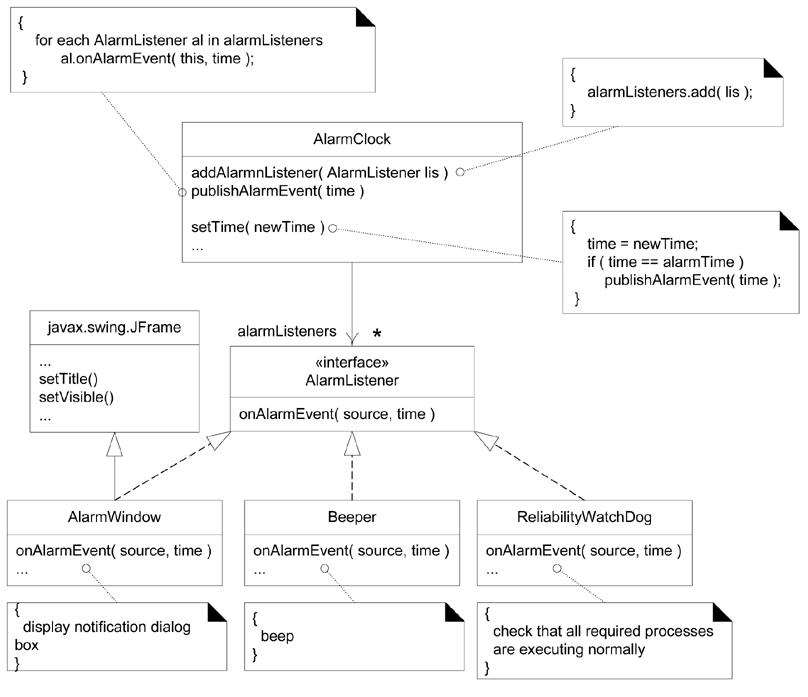
最后，我们希望确保内部的接口都是由同一个服务对象提供的，所有适配器也应该由同一个工厂生产出，因此可以对实现内部接口的类和工厂类使用**单例模式**。

综上所述，可以这么设计这个问题：通过工厂模式，针对不同的外部接口生产不同的适配器（适配器模式），工厂本身使用单例模式创建。

1. Try to draw a class diagram for the structure of the general GoF Composite pattern.（陈家龙）

答：

1. What pattern is illustrated by the following diagram? Simply explain what and how the problem is solved.（杨宗恺）



答：

策略（Strategy）模式。

首先主要的实例类是AlarmClock，代表一个闹钟，有一个成员变量alarmListeners，是一个闹钟监听器数组，可以存放多个闹钟监听器，闹钟里面有三个方法：

addAlarmListener(AlarmListener lis)用来添加一个闹钟监听器；

pubishAlarmEvent(time)使所有的闹钟监听器触发闹钟监听器执行事件；

setTime(newTime)为闹钟设置新的时间，如果时间等于响铃时间，那么直接触发闹钟监听器执行事件；

这个闹钟由闹钟监听器AlarmListener进行监听，对有关这个闹钟的各种事件做出反应，AlarmListener是一个接口，有AlarmWindow、Beeper、ReliabilityWatchDog三个实现，AlarmWindow是一个图形界面，用来显示监听到事件时的通知对话框，Beeper用来监听到特定事件时发出嘟嘟声，ReliabilityWatchDog用来检查所有必需的进程是否正常执行，保证监听器正常工作。

闹钟要解决的问题是：当设定一个响铃时间给这个闹钟，到达这个响铃时间时，闹钟能相应的作出应有的响应。

解决问题的方式：

首先为闹钟添加闹钟监听器，闹钟由闹钟监听器进行监听，当时间time到达响铃时间alarmtime时，闹钟监听器监听到这一事件时，立刻做出反应，使用它的实现，使闹钟发出响声，并显示通知对话框，达到闹钟在指定时间响铃完成应有动作的效果。

1. **Revise the collection of all your artifacts you have done for your course project and submit the revised collection in a separate zipped file.**

具体见压缩文件。