System Analysis and Design

Homework Assignment 6

学号：19335286 姓名：郑有为

1. What is Responsibility-Driven Design? How can we do Responsibility-Driven Design?

- **什么是Responsibility-Driven Design**：Responsibility-Driven Design，即职责驱动设计，其内在含义是考虑如何给相互协作的对象分配职责，主要关注职责、角色和协作。其中职责分为行为和认知两种类型。
- **对象的行为职责**，包括：
  - 自身执行的一些行为，如创建对象或计算
  - 初始化其他对象中的动作
  - 控制和协调其他对象中的活动
- **对象的认知职责**，包括：
  - 对私有封装数据的认知
  - 对相关对象的认知
  - 对其能够导出或计算的事物的认知
- **如何做Responsibility-Driven Design**：进行职责驱动设计主要的任务就是给类分配职责，要注意职责与方法的区别：职责是一种抽象，而方法实现了职责。当我们在绘制UML交互图时，就是在决定职责的分配。GRASP中的基本原则可以指导在分配职责时可做的选择。

2. Why we say that RDD is a general metaphor for thinking about OO software design?

- 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设计看作使有职责对象进行协作的共同体。

3. What is design pattern? Why design patterns are important in software design?

- **设计模式的概念**：
  - 设计模式代表了最佳的实践，通常被有经验的面向对象的软件开发人员所采用。设计模式是软件开发人员在软件开发过程中面临的一般问题的解决方案。这些解决方案是众多软件开发人员经过相当长的一段时间的试验和错误总结出来的。
  - 开发者以结构化的形式对开发过程的问题、解决方案和命名进行描述使其系统化得到的原则和习惯用法称为设计模式，它可应用于新的情景。
- **设计模式重要的原因**：
  - 设计模式对一种问题和解决方案进行命名，支持将概念条理化低组织为我们的理解和记忆，便于沟通。
  - 设计模式就像盖房子首先要搭建好一个框架 ，这个框架就是设计模式，框架能规范建造房子的过程，也省去了不少麻烦，同理，在我们编写程序的时候也需要用设计模式，这样减少了时间的开发，节省了人力和无力，使代码减少了冗余。
  - 使用设计模式可以提高代码可重用性、可扩展性、灵活性，让代码更容易被他人理解、保证代码可靠性。

4. 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.)

- 名称：Creator 创建者
  - 问题：谁创建了A?
  - 解决方案：如果以下条件为真时（越多越好），将创建类A实例的职责分配给类B：

        - B“包含”或组合聚集了A。
        - B记录A。
        - B紧密地使用A。
        - B具有A的初始化数据。
- 名称：Information Expert 信息专家
    - 问题：给对象分配职责的基本原则是什么？
    - 解决方案：把职责分配给具有完成该职责所需信息的那个类，信息包括关于其他对象的信息、对象自身的状态、对象周围的环境、对象能够导出的信息等等。
- 名称：Low Coupling 低耦合
    - 问题：如何减少因变化产生的影响？
    - 解决方案：分配职责以使不必要的耦合保持在较低水平，用该原则对可选方案进行评估。
- 名称：Controller 控制器
    - 问题：在UI层之上首先接收和协调（“控制“）系统操作的对象是什么？
    - 解决方案：把职责分配给能代表下列选择之一的对象：
        - 能够代表全部”系统“、“根对象”、运行软件的设备或主要的子系统（外观控制器的变体）
        - 代表发生系统操作的用例场景（用例或会话控制器）
- 名称：High Cohesion 高内聚
    - 问题：怎样使得对象保持有内聚、可理解和可管理，同时具有低耦合的附加作用？
    - 解决方案：职责分配应该保持高内聚，以此来评估备选方案。内聚是元素职责的相关性和集中度的度量。如果元素具有高度相关的职责，并且没有过多的工作，那么该元素具有高内聚性。

5. 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 ?

- **用例实现**：描述某个用例基于协作对象如何在设计模型中实现。更精确地说，设计者能够描述用例的一个或多个场景的设计，其中的每个设计都称为用例实现（或场景实现）。
- **用例实现的主要输入：**用例就是用例实现的主要输入。用例文本和补充性规格说明、词汇表、UI原型、报表原型等所述的相关需求向开发者提供需要构建的全部内容。
- **在用例实现过程中的主要任务是**：
    - 找出用例中的对象。
    - 找出它们的创建者。
    - 确定对象之间的交互关系。
    - 确定可能应用的模式。

6. 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?

- **设计具有对象的用例实现过程中要基于GRASP模式做出选择和决定的原因**：用例实现描述某个用例基于协作对象在设计模型中的实现。在对象设计中，设计对象交互和职责分配是核心，这些设计决策对对象软件系统的清晰度、可扩展性和可维护性具有重大影响，同时也对构件复用的程度和质量具有影响。而GRASP模式总结了面向对象设计最常用的原则，具有对象的用例实现基于GRASP模式做选择和决定，将有助于得到具有上述优点的设计模型。

7. What artifacts may be the UP implementation model? What artifacts should be used as the inputs for the code generation process ?

- UP实现模型包括的制品包括业务建模，需求和设计。业务建模具体为领域模型。需求具体为用例模型，设想，补充性规则说明，词汇表，业务规则。设计具体为设计模型，软件架构文档，数据模型。
- 在设计工作中创建的UML制品（交互图和DCD）可以作为代码生成过程中的输入。

8. During the mapping design to code, what kinds of translations and what creative work we should do?

- 在设计工作中完成了一些制定决策和创造性的工作。在后续对这些示例生成代码的讨论中，会看到一些相对机械的转换过程。
- 然而一般来说，编程工作并非微不足道的代码生成步骤，事实恰恰相反！实际上，在设计建模中产生的结果只是不完整的第一步，在编程和测试的过程中，会做出很多的变更并且要发现和解决无数细节的问题。
- 如果做得好，那么可以将OO设计建模中形成的思想和理解作为良好的基础，提高编程的优雅性和健壮性从而应对编程中遇到的新问题。但是，要对编程中存在的变化和偏差有所预计和计划。这是在迭代和进化式方法中的关键（也是实际的态度）。

9. What is test-driven development ? What basic rhythm we have suggested for test-first development?

- 测试驱动开发是迭代和敏捷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。在TDD风格的OO单元测试中，要在测试类之前编写测试代码，并且开发者要为几乎所有的产品代码编写单元测试代码。
- 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，依此类推。

10. What are the goals of refactory? List several refactoring methods we may usually use in our coding process.

- 重构的目标：
  - 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
- 重构的方法：
  - 提炼方法：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者方法。
  - 提炼常量：使用常量变量替换字面常量。
  - 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来代替对新操作和构造器的调用（隐藏细节）

11. Explain how GoF adapter pattern support GRASP *Protected Variations* in terms of GRASP terminology.

- 适配器支持防止变异，因为它通过应用了接口和多态的间接对象，改变了外部接口或第三方软件包。
- 低耦合是在变化点实现保护的方式。
- 多态是在变化点实现保护的方式，并且也是实现低耦合的方式。
- 间接性是实现低耦合的方式。
- 适配器设计模式是一种间接性和纯虚构，并且使用了多态，通过这些方法来防止变异。

12. What are the connections between Factory pattern and GRASP pure Fabrication pattern?

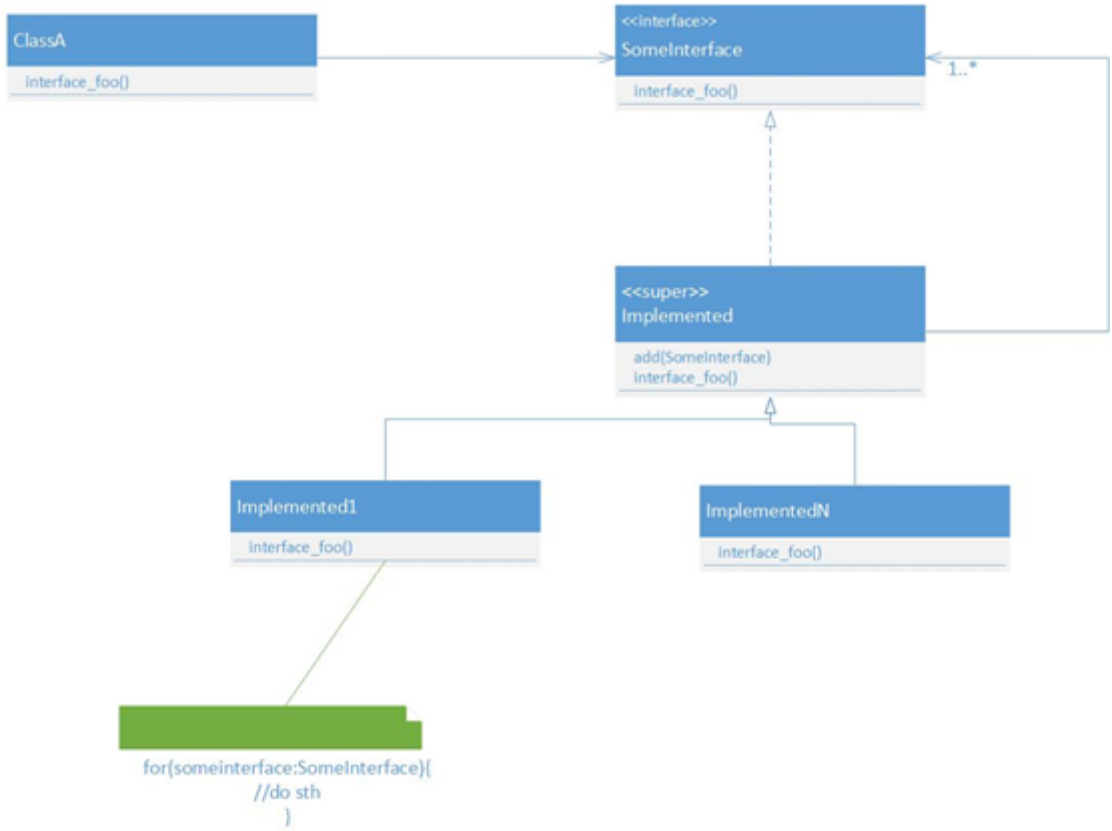
- 纯虚构是GRASP模式，其中的软件类是由设计者任意创建（虚构）的，而不是来源于领域模型。
- 基本设计原则中有：设计要保持关注分离。也就是说，将不同关注分离或模块化为不同领域以确保内聚。基本上，这是对GRASP高内聚原则的应用。因此，选择领域对象来创建适配器不能支持关注分离的目标，也降低了内聚。

- 在这种情况下，常用的替代方案是使用工厂模式，其中定义纯虚构的“工厂对象”来创建对象。
    - 问题：当有特殊考虑（例如存在复杂创建逻辑、为了改良内聚而分离创建职责等）时，应该由谁来负责创建对象？
    - 解决方案：创建称为工厂的纯虚构对象来处理这些创建职责。
    - 工厂方法返回对象的类型时接口而不是类，因此工厂能够返回接口的任何实现。

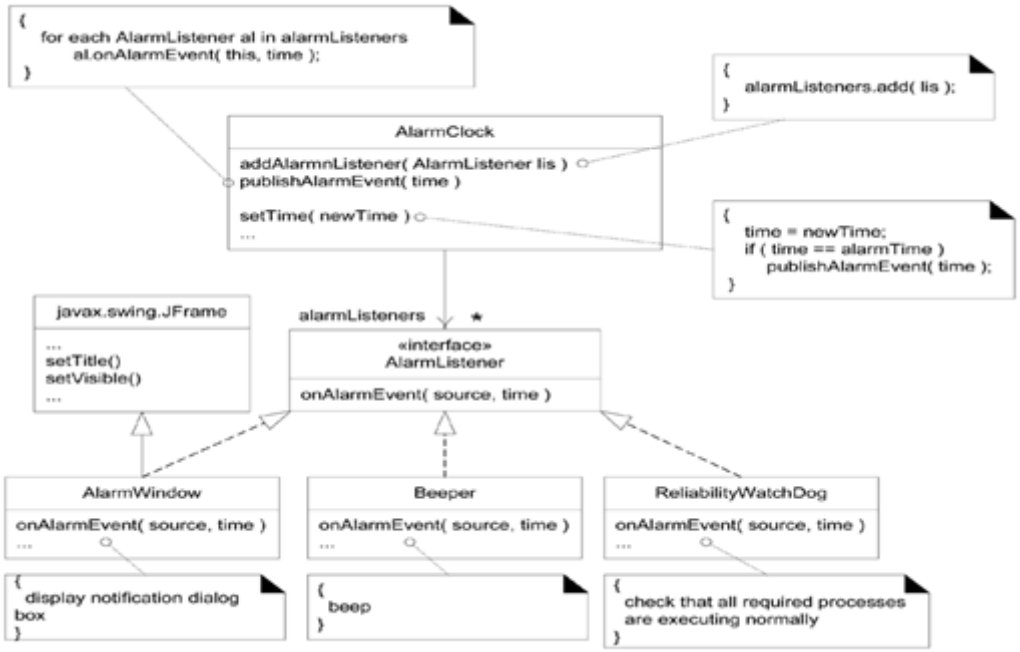
13. For solving the external services with varying interfaces problem, what patterns may be used in your design?

- 多态

14. Try to draw a class diagram for the structure of the general GoF Composite pattern.



15. What pattern is illustrated by the following diagram? Simply explain what and how the problem is solved.



- 使用了观察者模式。
    - 解决的问题：不同的订阅者对象要对发布者的事件作出反应，但是订阅者和生产者之间的低耦合要被保持。
    - 解决方法：发布者不知道各个订阅者的对象，只对接口耦合，各个发布者实现接口并使发布者将自己加入到发布者维护的订阅者列表中。当一个事件到达时，发布者通过遍历订阅者列表并调用接口来使得订阅者对事件感知。

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