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

责任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。

依据职责驱动设计的原则，我们在分析系统的时候，必须为每一个模块、包、对象类，特别是对象类，定义一个明确的职责。进行职责驱动设计，应遵循 GRASP 软件设计模式，即包括创建者、控制器、信息专家、低耦合、高内聚、多态、纯虚构、间接性和防止变异九个模式。

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

面向对象的设计鼓励将世界视为合作和协作代理的系统。在面向对象的系统中，工作是通过一个对象向另一个对象发送请求以执行其操作之一、显示其某些信息或两者兼而有之来完成的。责任驱动设计使用类-责任-协作者捕获设计决策。协作者与他们参与的角色的职责相关联。在角色、类和职责被捕获之后，对象之间的特定交互被设计，通常使用语言隐喻对象协作.

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

General principles and idiomatic solutions for creating software • It is usually codified in a structured format describing the problem and solution • In OO design, a pattern is a named description of a problem and solution that can be applied to new contexts • GRASP defines nine basic OO design principles or basic building blocks in design

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.)

Creator

Information Expert

Low Coupling

Controller

High Cohesion

Polymorphism

Indirection

Pure Fabrication

Protected Variations

Creator Pattern • Problem: Who should be responsible for creating a new instance of some class? • Solution: assign class B the responsibility to create an instance of A if one or more are true: – B contains or aggregates A – B records A – B closely uses A – B has the initializing data for A

Information Expert • Problem: What is a basic principle by which to assign responsibilities to objects? • Solution: Assign a responsibility to the class that has the information needed to fulfill it.

Controller A controller is the first object beyond the UI layer that is responsible for receiving or handling a system operation message. System operations were first explored during the analysis of SSD. These are the major input events upon our system • For example, the “End Sale” button in a POS system or the “Spell Check” button on a word processor. • This is a delegation pattern

High Cohesion • Cohesion is a measure of how strongly related the responsibilities of an element are • A class with low cohesion does many unrelated things. – It is hard to understand, hard to reuse, hard to maintain, and delicate – constantly affected by change • In POS system,

if Register creates the payment, this is less cohesive than if Sale does it

5. What is Use Case Realization?

描述如何在设计模型中根据协作对象实现特定用例，UML 图是说明用例实现的通用语言，我们可以在这个用例实现设计工作中应用原则和模式。

What is the prime input to use case realizations?

用例是用例实现的主要输入，在补充规范、词汇表、UI 原型、报表原型等中表达的用例文本和相关需求，都会告知开发人员需要构建什么。

What are the main tasks we need do during use cage realization ?

查看用例建议的对象，确定谁应该创建它们，决定他们应该如何互动，确定正在使用的模式

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?

因为程序编码时至少要先做一些启动的初始化，同时在建模时，最后在做启动初始化，然后再来考虑系统用例实现所需的初始化，而 GRASP 模式可以完美的解决这些问题。

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

指的是实际的程序：可以包括源代码、数据库的定义、JSP、ASP 等等。

使用 UML artifacts （ 包括交互图和 DCD ）作为代码生成过程的输入。

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

translations:

* 第一步是设计，设计模型和类图
* 在程序实现和测试的步骤中, 会有大量的变化出现, 这时能发现细节的问题并解决它们。
* 过程中对于生成的领域的理解是你写代码的基础。
* 同时还要预料到，在程序实现时会有大量的设计上的变化和偏差。

creative work:

* 类和接口的定义；方法的定义
* 从 DCD 创建类
* 从交互图 创建方法
* 通过一些 list、array 等类型将类综合起来。
* 用接口声明类
* 做异常和错误的处理
* 通过协作图可以导出方法的代码
* 考虑实现的先后顺序，先从低耦合的类开始实现。

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

测试驱动开发是敏捷开发中的一项核心实践和技术，也是一种设计方法论。TDD 需要在写代码之前先进行单元测试，并为所有的单元书写测试代码。

其基本规则是：小步推进，小步代码测试

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

重构的目标：

1. 移除重复代码
2. 增加清晰度

3. 缩短方法
4. 清除文字常量

重构方法：

1. 提取方法
2. 抽取常量
3. 引入解释性变量
4. 以工厂函数取代构造函数

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

适配器通过使用应用接口和多态性的间接对象，支持与更改外部接口或第三方包有关的受保护的变体

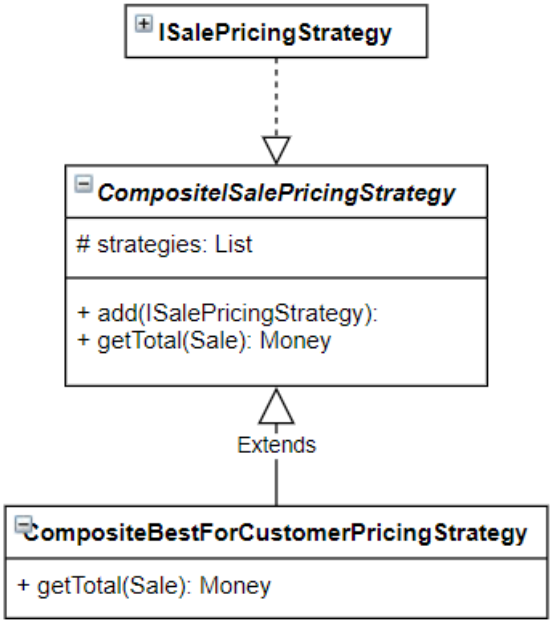
12. What are the connections between Factory pattern and GRASP pure Fabrication pattern?

Factory pattern 中定义一个 GRASP pure Fabrication 工厂对象来创建对象.

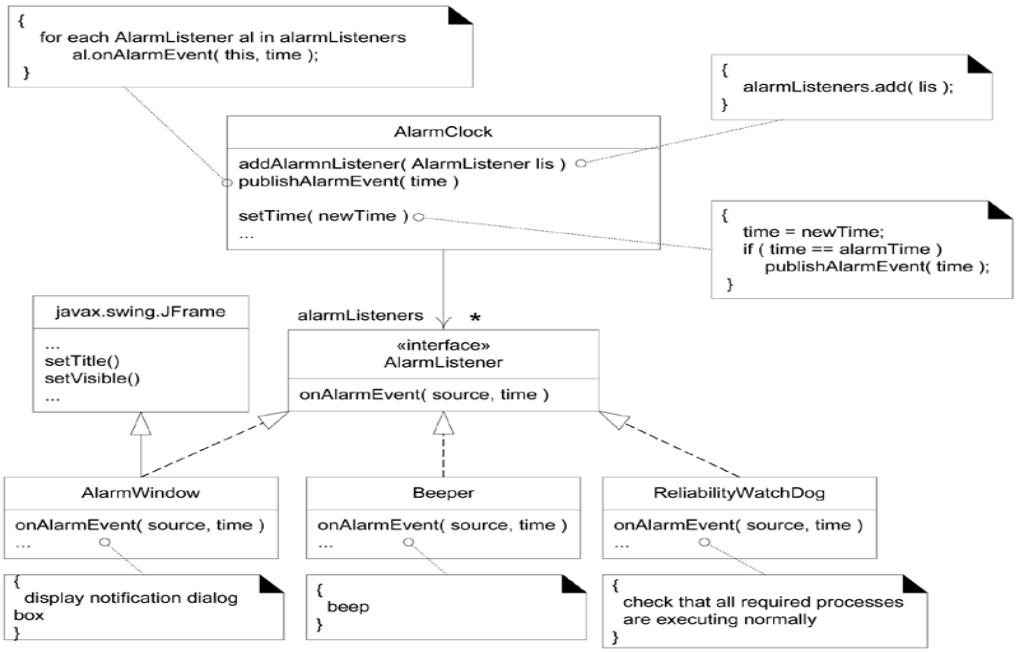
13. For solving the external services with varying interfaces problem, what patterns may be used in your design?

可以使用 Adapter 设计模式来解决具有不同接口的外部服务问题，通过中间适配器对象将组件的原始接口转换为另一个接口。

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.



组成部分：

中间 AlarmClock 是主体，表明这个图是描述这个 AlarmClock 的。包含三个方法：addAlarmnListener方法添加alarmListener。pubilshAlarmEvent方法对于每一个alarmListener都去调用函数 onAlarmEvent 方法。SetTime 方法，设置时间。

下面是每一个具体的 alarmListener 的方法，有一个表面 interface。里面有一个方法ononAlarmEvent，这里面有三个硬件设备会对这个函数起反应：AlarmWindow，控制一个类似于显示屏的东西，还有一些别的设定这个显示屏的函数。Beeper 一个用于提醒用户的发声设备，当时间到了以后会“beep~”的响。最后一个是 ReliabilityWatchDog 是一个可靠的硬件设备或软件，用于检测所有的进程运行正常。

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.