# 第一章 上升到面向对象

## [√]1.什么是抽象，如何进行抽象？

* 抽象揭示本质特征
* 抽象根据使用目的的不同，强调不同特征（*忽略其他的特征*）

## [√]2.什么是封装，通过封装如何实现信息隐藏和数据抽象？

* 封装是指隐藏具体的实现
* 保护私有属性，保证外界访问的合法，实现了信息隐藏
* 调用操作就可以实现对数据的访问，从而将数据抽象为行为

## [√]3.什么是泛化，什么是多态，它们之间有什么关系？

* 泛化是类之间的关系，一个类共享另外一个或多个类的结构和行为*（泛化应该就体现在继承中吧，子类自然共享父类的结构和行为）*
* 多态是在同一接口下表现多种行为*（接口便实现了多态，结合LSP，可以表现不同子类的行为）*

关系：

* 通过泛化建立类之间的**抽象层次结构**
* 通过上层抽象（*如接口*）多态调用底层实现

## [√]4.什么是分层，分层和分解有何不同？

* 分层是根据不同的目标，建立不同的抽象级别层级
* *（将分解的范围调小）*实现在不同抽象的层次上对系统进行分解
* 目的在于简化对系统的理解

分层和分解的不同：

* 分解是在系统的同一个层次上进行
* 分层是在不同的抽象层次上进行
* 大规模系统开发，一般是先分层，再分解

## 5. 与传统结构化方法相比，面向对象技术的优势主要体 现在哪些方面？

主要包括以下几个方面的与优势：

1. 沟通：在计算机中模拟现实世界的事和物；
2. 稳定：较小的需求变化不会导致系统结构大的改变；
3. 复用：提高质量，减少成本；
4. 改善软件结构，提高软件灵活性；增加可扩展性；支持增量式开发，支持大型软 件开发等

## 6. 什么是对象，什么是类，说明它们之间的区别和联系

1. 对象是一个实体，这个实体具有明确定义的边界和标识，并且封装了状态和行为；
2. 类就是对象的抽象描述，这些对象共享相同的属性、操作、关系和语义。
3. 类是对象的抽象，而对象是类的实例，是具体的；通过类可以构造具体的对象。

## 7. 什么是分解，结构化分解和面向对象分解有何不同？

1. 分解是指将单个大规模复杂系统划分为多个不同的小构件；分解后的构件通过抽 象和封装等技术形成相对独立的单元，这些单元可以独立地设计和开发，从而实现化繁为简、 分而治之，以应对系统的复杂性，减少软件开发成本。
2. 结构化分解中，通过函数、模块等进行功能分解，实现模块化设计。通过耦合和 内聚来判断分解的合理性，将系统分解为多个高内聚、低耦合的模块。而面向对象的分解则 是在类和对象分解的基础上，进一步考虑类之间依赖程度、复用问题和稳定性等问题，进行 合理的打包和分层，从而形成更加复杂的分解结构。

## 8. 什么是复用，在软件开发的哪些阶段可以进行复用？

1. 复用是借助于已有软件的各种有关知识建立新的软件的过程，以缩减软件开发和维护的成本；
2. 系统开发的各个阶段都可能涉及到复用，如**代码复用**、**设计复用**、**架构复用**、**需求复用**和**领域复用**

# 第二章 可视化建模技术

## [√]1.通过建模技术，可以达到哪些目标？

（1）可视化：模型有助于按照所需的样式可视化系统；

（2）描述：模型能够描述系统的结构和行为；

（3）构造：模型提供构造系统的模板提高质量，减少成本；

（4）文档化：模型可以文档化设计决策。

## [√]2.在系统建模过程中，需要遵循哪些基本原则？

（1）选择合适的模型；

（2）模型具有不同的精确程度；

（3）最好的模型是与现实相联系的；

（4）需要从多个视角创建不同的模型，单一的模型是不够的。

## [√]3.哪些情况下，适合使用 UML 进行系统建模？

（1）项目采用的 **OO 方法论**；

（2）提高项目**开发人员之间交流效率**，准确抓住问题本质；

（3）**系统的规模和设计都比较复杂**，需要用图形抽象地表达复杂的概念，增强设计的 灵活性、可读性和可理解性，以便暴露深层的次设计问题、降低开发风险。

（4）**需要记录已成功项目**、产品的公共设计方案，在开发新项目时可以参考、复用过 去的设计，以节省投入，提高开发效率和整体成功率。

（5）有必要采用一套通用的图形语言和符号体系描述组织的业务流程和软件需求，促进业务人员、软件开发人员之间一致、高效地交流。

## [√]4.UML 中的事物之间主要存在哪些基本关系？

UML 中的事物之间主要 4 类基本关系

* *依赖：弱语义关系：一个变化影响另一个。*
* *关联：弱语义关系；两者有语义联系。*
* *泛化：一般/特殊；（特殊元素）子元素可替代父元素（一般元素）。*
* *实现：契约关系；一个事物描述另一个事物需实现的契约。*

## [√]5.什么是 UML 架构中的视图，和 UML 图有什么区别和联系？

* 视*（角）*图是某个视角的模型
* 每种视图面对**不同用户**，实现**不同目标**
* 每种视图提供不同UML模型

区别，联系：

* UML 图是特定的 UML 模型，视图由不同的 UML 图组成。
* 根据视图的用户及目标，选择不同的UML图

# 第三章 业务建模

## [√]1.什么是业务建模，软件开发过程中为什么要进行业务建模？

* *业务建模是****分析，理解业务****以生成业务模型的的****建模方法集****。*
* *业务建模有助于建立业务模型与系统模型的对应关系,以保证模型可以满足业务需求*

## [√]2.什么是业务用例模型，业务用例模型主要包括哪些内容？

* *业务用例模型是业务建模的****核心***
* *用来说明希望达成的****业务功能，确定组织的各个角色和可交付工件****。*
* *业务用例模型主要包括****业务用例****和****业务参与者****，主要目的说明如何进行这个业务。*

## \*3.活动图中的动作节点什么条件下可以执行，有哪些种类的动作节点？

（1）当动作结点**所有**的**对象流**和**控制流**的前提条件**都**满足时，才创建动作的一次执行。

（2）根据动作执行所涉及的功能不同，可以划分为不同类别的动作，包括**基本功能**、 **行为调用**、**通信动作**和**对象处理**等不同类型的动作节点

*条件：所有对象流控制流的前提条件满足*

*种类：基本功能，行为调用，通信动作，对象处理*

## \*4.什么是活动图中的控制节点，通过哪类控制节点可以进行并发行为建模？

（1）控制节点是一种特殊的**活动节点**，用于在**动作节点**或**对象节点**之间**协调流程**，表示某一种**控制动作**。

（2）根据**分叉和汇合**这两个控制节点可以对并发执行和同步控制行为进行建模。

## \*5.活动图中的边可以设定哪些执行参数？

（1）可以为活动边设定执行条件、关联动作和权重等信息。

（2）执行条件为真时才能通过该活动边进入下一个动作关联的动作。

（3）关联动作表示在进入下一个动作节点之前需要提前执行的动作。

（4）权重规定了转移发生时输入对象的最小数目（常量或表达式），缺省为全部输入对象

## \*6.什么是活动分区，一般什么情况下对活动进行分区？

（1）活动分区用于识别具有相同特性的一组动作，这些动作被放入相同的区间。 （2）可以使用不同的分区规则进行分区，并没有严格的规范。

（3）在业务模型或需求中，往往按照组织机构的单位或系统角色进行分区，一个单位或角色负责分区中所有节点的行为。而在设计模型中，可以按照不同的类（或构件）进行分区，一个类（或构件）负责执行该分区中所有节点的行为。

## [√]7.什么是业务对象模型，业务对象模型主要包括哪些内容？

业务对象模型：

* **从业务人员内部的观点定义了业务用例*（展示业务用例内部）***
* 确定了业务人员以及他们处理和使用的对象（“业务类和对象”）之间应该具有的静态和动态关系 *（展示业务人员与对象之间的关系）*
* 业务对象模型主要包括：
  + **业务工人**
  + **业务实体**
  + **业务用例实现**等

# 第四章

## [√]1.什么是涉众，涉众和参与者有何区别和联系？

* 涉众就是跟当前用例有利益关系的系统内外部人员和组织
* 普通人或部门的参与者一般是涉众
* 外系统的用户*（如果有）*往往作为当前用例的涉众
* 从涉众的角度来看，**用例实际上是涉众之间所达成的契约**，并以参与者为达成特定目标和系统交互的方式演绎。把用例比作一台戏，参与者和系统就是这台戏的演员，而涉众则是观众，戏的好坏由观众来评价。

## [√]2.什么是用例的前置条件和后置条件，它们有什么作用，定义时需要注意什么？

* 前置条件：执行前满足，是入口限制，满足条件才可以执行该用例
* 后置条件：执行后应满足的条件，用于确保涉众理解用例结果的

需要注意：

* 用例使用者将这些条件作为**附加价值**的时候才使用
* 要求系统是**可以感知**的
* 前置条件要在用例执行前就可以感知到

## [√]3.什么是用例的事件流，描述事件流是需要注意什么？有哪几种事件流，它们之间有何区别和联系？

事件流：

* 是**参与者与系统交互的过程**
* 只需要描述**需求部分**
* 使用用户和开发人员**都理解**的用例的功能

需要注意：

* + **使用业务语言**：使用用户平时所使用的语言进行描述。 
  + 重点描述参与者与系统交互的**信息**。 
  + 不使用[例如]、[等]这样的**不清晰**的表达。 
  + **不**要过多的考虑**界面细节**。 
  + 不要描述系统**内部处理细节**，要描述从系统外部所看到的活动。 
  + 要明确描述用例的**开始和结束**：一般事件流的第一句话表明该用例在何时

区别联系：

* 有基本事件流和备选事件流
* 最核心的是基本事件流，是最一般情况下用例发生的路径
* 备选事件流代表用例的分支异常情况，分离自基本事件流的某一步

## [√]4.用例模型中，可以定义哪几种用例关系，它们有何不同？

* 包含关系，扩展关系，泛化关系
* 包含关系：一个用例包含其他用例，从多个用例中提取**公共部分**单独作为用例
* 扩展关系：扩展用例接受**基用例的委托**，处理其无法处理的**特殊情况**
* 泛化关系
  + 表明一种继承层次，特化用例**继承泛化用例的所有属性行为**，并参与其各种关系
  + 实现了**更高层次**的**需求复用**
  + **泛化用例描述通用的行为，特化用例描述其他的行为**

## [√]5.什么是扩展点，扩展点有什么作用？

* 是一个，定义在基用例中的，**特定条件**
* 每个扩展用例至少与一个扩展点**相连**
* 满足特定条件，就触发对应的扩展用例，扩展了基用例的行为*（所以是扩展点了）*

## [√]6.如何对用例进行分级，高优先级的用例有何特征？

* 需要结合一下来考虑
  + 项目**业务特点**
  + 开发团队**技术特点**
* 特征
  + 对系统架构有重要影响的用例*(不做这个其他没法做呀)*
  + 体现系统核心业务流程的用例*（系统的核心就在这儿）*
  + 存在开发风险的用例*（有风险就需要额外关照了）*
  + 涉及新技术或者需要创新的用例 *（有难度就需要额外关照了）*
  + 能够尽快投入使用并带来直接经济效益的用例*（当然先做赚钱的了）*

## [√]7.有哪些用例的分包策略，一般如何进行用例分包？

* 基于**业务主题**分包
* 按照**参与者**
* 基于**开发团队**
* 基于**发布情况**
* 一般先根据业务主题，再考虑开发团队和发布情况

# 第五章 用例分析

## [√]1.分析模型主要包括什么内容？

* 是对分析形成目标制品的总称
* 两个层次
  + 架构分析
  + 用例分析
* 两类模型
  + 静态模型
  + 动态模型

## [√]2.什么是用例实现？它和用例之间有何区别和联系？

* 分析（设计）模型中系统用例的表达式
* 通过对象交互方式描述用例是怎么实现的

区别，联系：

* 用例实现是用例的实现
* 用例实现将用例和分析模型、设计模型中的类联系起来，描述用例需要哪些类实现

## [√]3.什么是架构模式，有哪些典型的架构模式？

* 架构模式是那些在**开发过程中积累**下来，并经过**实践验证行之有效的**、**可复用**的**软件架构**。*（与设计模式定义相仿）*
* 它表示了对软件系统的一个**基础结构组织形式**。
* 它提供了一套**预定义子系统**，详细说明它们的职责，并且包括组织它们之间的规则和指南。

典型架构：

* 针对系统软件
  + **层**
  + **管道**
  + **过滤器**
  + **黑板**
* 针对交互式软件
  + **模型视图控制器**模式

## **[√]4.什么是 B-C-E 三层架构？**

* B-C-E 三层架构是对 MVC 架构的另一种表述，
* 将系统划分为三层，分别处理 3 类业务逻辑。
* B 表示**边界层**，负责处理**系统与参与者的交互**；
* C 为**控制层**，处理系统的**控制逻辑**；
* E为**实体层**，负责管理**系统使用的信息**。

## [√]5.顺序图中主要有哪些元素，绘制顺序图的基本过程是什么？

* 顺序图是一种 UML 交互图，表示对象的交互，**由一组对象和它们之间的消息传递组成**， **强调消息的时间顺序**。
* 主要包括
  + **对象**
  + **对象生命线**
  + **消息**
  + **执行发生**等

步骤

1. **放置对象**：从已识别的参与用例的分析类中构造相应的对象放置到顺序图中；
2. **描述交互**：从参与者开始，按照用例事件流（或场景）的叙述，将系统行为转化为对象间的消息；
3. **验证行为**：从后往前，验证对象的行为序列，确保每一个对象能够实现该行为序列。

## [√]6.什么是边界类，什么是控制类，如何识别这两种分析类？

* 边界类是从那些系统和外界进行交互的对象中归纳和抽象出来，代表了**系统与外部参与者交互的边界**。
* 控制类封装控制系统上层的**边界类**和下层的**实体类**之间的交互行为，是整个用例行为的**协调器**。
* 对边界类识别的基本原则:为每一对参与者/用例确定一个边界类。
* 对控制类识别的基本原则:为每个用例确定一个控制类。

## [×]7.什么是关键抽象，如何识别关键抽象？

* 关键抽象是一个通常**在需求上被揭示的概念**，系统必须能够对其处理。
* 它来源于业务，**体现了业务的核心价值**，**即业务需要处理哪些信息**；
* 这些信息所构成的**实体**即可作为初步的**实体分析类**。
* 关键抽象来自于业务领域，**领域专家**可以很清楚地提供业务系统的初始关键抽象**候选集合**
* 在此基础上，再结合**业务对象模型**、**需求**和**词汇表**等业务文档资料补充和完善。

## 8.什么是架构机制，什么是分析机制，有哪些典型的分析机制？

* 架构机制是对**通用问题**的**决策**、**方针**和**实践**，它描述了针对一个经常发生的问题的一种通用解决方案。通过有效地应用架构机制，可以使项目组内部以相同的方式对待这些问题， 并复用相同的解决方案来实现复用。
* 分析机制是架构机制在分析阶段的表述，它以与实现无关的方式捕获解决方案的关键部分。它们可能表示结构模式或行为模式，也可能表示这两者。它们主要用于在分析过程中向 设计人员提供复杂行为的简短表示，从而减少分析的复杂性并提高分析的一致性。
* 典型的分析机制
  + 持久性
  + 分布
  + 安全性

## 9.什么是实体类，如何有效地识别实体类？

* **实体类代表了系统的核心概念**，来自于对业务中的实体对象的归纳和抽象，用于记录系统所需要维护的数据和对这些数据的处理行为。 实体类是用来表示业务信息的名词，

因此识别实体类的基本思路是:

* 分析**用例事件流**中的**名词**、**名词短语**，找出所需的实体信息。
* 架构分析中的**关键抽象**是识别实体类最重要的来源
* 而更多的实体类还需要从**用例事件流**、**业务模型**、**词汇表**等业务和需求的载体中获得。

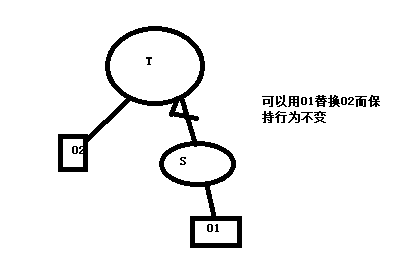
# 第六章

## [√]1.什么是面向对象的设计原则，它和设计质量有何联系？

* 是面向对象设计的**基本思想**
* 是面向对象设计**评价**的**价值观体系**
* 是构造**高质量软件**的出发点

## [√]2.什么是 Liskov 替换原则（LSP，里式替换原则），该原则有什么用？

什么是LSP：

* *(是什么)*是**继承**层次设计需遵循的原则
* 

有什么用：

* 指导**继承层次的设计**，如果任何情况下子类与基类都是可以互换的，才称继承的使用是合适的

## [√]3.什么是开放-封闭原则（OCP），该原则与 Liskov 替换原则有何联系？

* 软件模块对**扩展**开放
  + 需求变化，可以对模块进行扩展*（比如可以在原有类基础上派生出一个子类，在子类中修改；既保留了原有行为，又增加了新的行为；在基于里式替换原则，便可以实现直接用派生类替换原有类）*
* 软件模块对**修改**封闭
  + 修改时不应修改模块的源代码*（比如上例，可以派生，但不应修改原有类）*

联系

* **LSP**是OCP成为可能的主要原则之一
* *（说明LSP的性质起的作用）*子类型的可替代性使模块无需修改就可以扩展
* *（进一步解释上句，怎么应用LSP）*定义一个**抽象基类**建立软件系统的基本结构*（这个抽象基类相当于LSP中的T，派生的子类相当于S），*在这个结构上，扩展派生类即可应对需求变更

## [√]4.什么是单一职责原则（SRP），什么时候使用该原则？

* 指一个类应只有一类功能相关的职责
* 该职责主要用于类职责分配
* 如果职责过多，过多的变化原因导致类频繁变化
* 在类设计中应使用SRP，从而实现高内聚的类*（高内聚：尽可能类的每个成员方法只完成一件事（最大限度的聚合））*

## [√]5.什么是接口隔离原则，什么时候使用该原则？

什么是：

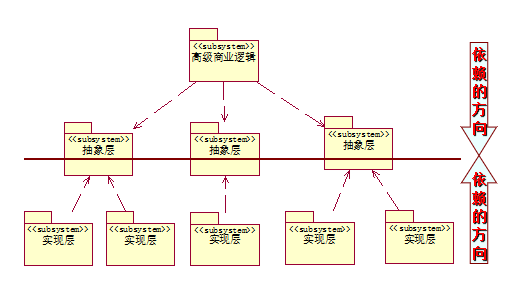
* 使用多个专门的接口比使用单一的总接口要好
* 一个类依赖性应建立在最小接口上

什么时候使用：

* 接口设计时使用
* 每个角色由一个特定接口代表，而不是多个角色

## [√]6.什么是依赖倒置原则，在该原则中如何理解抽象层的设计？

* 高层模块不应依赖于底层模块，两者依赖于抽象
* 抽象不应依赖于细节，细节依赖于抽象



如何理解抽象层设计？

* 系统建立抽象层的过程即是透过现象找本质*（特征）*的过程
* 通过对本质特征的描述(*本质特征不容易变更，因而稳定*)可以建立稳定的系统结构
* 职责分配过程，需要对职责进行抽象，在抽象层次上理解分配职责

# 第七章

## [√]1.什么是模式，什么是设计模式，它们之间有何区别和联系？

模式：

* 对成功经验的总结，复用
* 是通用的解决方案
* （因为通用且成功，且在成功经验的基础上进行了一定的抽象）可以重复使用

区别联系：

* 设计模式：用于构建设计阶段,定义类或对象之间的结构行为
* 模式：用于各种领域，设计模式是其中的一种

## [√]2.什么是 GOF 模式，有哪些典型的 GOF 模式？

GOF:

* 四人帮（组）在《设计模式-可复用面向对象软件的基础》一书中提出的23种设计模式

典型模式：

* 按范围可以分为类模式和对象模式，
* 按目的可以分为创建型，结构型，行为型模式
* 典型的有：
  + **工厂方法**
  + **抽象工厂**
  + **单例**
  + **适配器**
  + **组合**
  + **命令**
  + **状态**
  + **策略**。

## [√]3.面向对象的设计原则和设计模式之间有何区别和联系？

* 设计原则是指导思想
* 设计模式是遵循了这个思想

## [√]4.什么是通用职责分配模式（GRASP），有哪些典型的通用职责分配模式？

职责分配模式：

* 是进行类职责分配的**原则**和**模式** （原则和模式这两个说法，根据课本注解应该都可以）
* 结合面临的不同问题，给出**具体分配规则**
* 用于指导**用例设计**期间的职责分配过程

典型类职责分配模式：

* 创建者
* 信息专家
* 低耦合
* 控制类
* 高内聚
* 多态
* 纯虚构
* 中介
* 受保护变化

## [√]5.什么是迪米特准则？在什么时候使用该准则？

迪米特准则：

* 是职责分配模式
* 给出一个方法内向哪些对象发送消息的限制

使用：

规定在一个方法中，消息只能发往下面的对象：

* 对象本身；
* 该方法的一个参数；
* 对象本身的属性；
* 对象本身的一个属性集合中的元素；
* 该方法内部创建的对象。

# 第八章

## [√]1. 什么是包，有哪些包设计原则？

包是将模型分组的机制（可以比作一个文件夹）。

* 包是一个容器，用来包含其他 UML元素（文件夹包含文件）；
* 为内部元素提供命名空间，外界通过包名来访问其内部元素（通过加文件夹名才能访问到文件）

设计原则（总的原则应该是归类存放吧，一个包中各个类行为一致比较好操作，一个文件夹都放相同类的文件也好管理）

* 复用发布等价原则 （一个包不能同时包含可复用和不可复用的元素，为的是改动不可复用元素时不影响可复用元素）
* 共同复用原则（一个包中的类要共同复用，不能只复用其中若干个）
* 共同封闭原则（一个包中的类要共同封闭，不能只封闭其中若干个）
* 无环依赖原则（依赖不能形成环，否则修改一个会影响其他的包）
* 稳定依赖原则（不稳定的包应该依赖稳定的包， 就像盖房要建立在坚实的地基上一样）
* 稳定抽象原则（抽象包稳定，不稳定包具体，意思是软件抽象部分不应随意变动）

## [√]2. 什么是子系统，它和包有什么区别和联系？

子系统(性质跟类类似，同样隐藏内部属性，同样通过方法提供服务)：

* 一种特殊的包
* 完全封装，不对外公开内部元素
* 实现接口定义的行为，外部通过接口获取服务

区别，联系：

* 普通包无语义的
* 子系统有与类相似行为特征

## [√]3. 什么是接口，接口和相应的子系统之间是什么关系？

接口：

* 定义了一个集合
* 这个集合包含**类**，**子系统**或构建所实现的操作 （子系统和类都可以用接口，接口是他们输出操作的方式）
* 接口只有定义，没有实现

接口与相应子系统之间的关系：

* 实现关系，子系统实现接口的操作

## [√]4. 如何进行软件架构设计，架构设计时需要考虑哪些方面的问题？

*如何进行：*

*在****分析阶段****和****设计阶段****都有进行*

* *分析阶段：*
  + ***基础架构的选型***
  + *确定核心的****分析机制***
* *设计阶段：针对分析阶段的****备选架构****各个方面进行****详细的定义****，具体工作包括：*
* ***确定核心元素****: 在架构中高层中，以****分析类****为出发点*
* ***引入外围元素：****在架构中低层中，以****分析机制****为出发点，确定满足分析类要求的设计机制*
* ***优化组织结构：****按高内聚，低耦合等设计原则，完善架构的层次和内容*
* ***定义设计后的组织结构：****还应考虑设计完成后****系统实现****，****运行****，以及****部署****阶段的****组织结构***

## [√]5. 什么是设计元素，面向对象设计中有哪些设计元素？

*设计元素：****直接指导实现编码的模型元素***

*主要元素有：*

* *设计类*
* *子接口*
* *主动类*
* *事件*
* *信号*

# 第九章

## [√]1. 什么是关联的导航性，如何设计导航性？

*导航性：*

* *指关联的方向*
* *描述了源类的任何对象到目标类的一个或多个对象****访问权限***
* *消息仅能在箭头方向上传递*
* *分析阶段默认是双向的导航*
* *设计阶段根据需要设计单向的导航*
* *没有导航性方向也就没有耦合，故使用单方向导航可以降低耦合性，减少维护成本*
* *A需要B，或A需要向B发送消息，则A-> B*

## [√]2. 什么是类间的组合关系，和聚合关系有何区别和联系？

*组合关系：*

* *是特殊的聚合关系，部分不脱离整体存在*
* *整体不存在，部分没有意义*

*与聚合的区别和联系：*

* *聚合表示引用关联，整体有部分的引用，部分可以独立存在*
* *组合表示值关联，整体拥有部分的值，负责部分的创建删除*

## [√]3. 什么是类间的依赖关系，哪些情况下定义为依赖关系？

*依赖：*

* *一个类对象使用另一个类对象的信息和服务*
* *被使用对象的变化可能会影响到使用对象*

*几种情况：*

* *参数引用*
* *局部声明引用*
* *全局引用*

## [√]4. 子系统设计主要包括哪些工作？

*子系统需要完成的工作：*

* *将子系统行为分配给子系统元素：通过交互图将接口操作的集合分配给子系统元素*
* *描述子系统内部的设计元素：在交互图基础上，定义每个设计元素的结构和关系，*
* *定义子系统间的依赖关系*

## [√]5. 在类设计阶段，针对三种分析类的有什么不同的设计策略？

*边界类：*

* *边界类分为用户界面和系统接口*
* *系统接口*
  + *通过定义为子系统和接口来实现*
  + *通过子系统设计完成内部设计流程*
* *界面类设计*
  + *需要研究具体与用户交互的场景，设计满足要求的最终用户界面*
  + *往往依赖界面开发工具*
  + *从界面元素布局等人机工程学方面考虑问题*

*实体类：*

* ***实体类本身职责的明确性觉****得大多数实体类可以直接作为初始设计类*
* *实体类的****持久性架构机制和数据库设计原则****对实体类设计方案产生影响*
* ***性能方面可能要求实体类重构***

*控制类：*

* *明确控制类****是否有必要存在****，不含业务逻辑或处理流程的控制类没有必要存在*
* *可以从****提供公共控制类****和****分解复杂控制类****两个方面来处理现有的控制类*

## [√]6. 什么是类的操作，什么是类的方法，它们有何区别和联系？

类的操作：

* 是类的行为特征
* 是对于特定请求做出应答的规范

类的方法

* 描述操作如何实现

区别，联系：

* 操作描述类对外提供的接口，是类的外在行为，通过定义操作明确参数和返回值等接口细节（感觉像是C++中的.h中的声明）
* 方法关注内部实现算法的设计（方法是具体实现，像是C++ .cpp中的函数实现）

## [√]7. 用例设计和用例分析有什么区别和联系？

*用例设计（总结的说，****用架构设计出的元素****替换****用例分析的结果，并加以实现，就产生了用例设计****）：*

* *是****用例分析****的延续*
* *利用架构设计给的****设计元素****，设计机制等素材*
* *将****分析结果****用****设计元素****替换和实现*

*区别，联系：*

* *建模方法相同*
  + *以交互图分析动态场景*
  + *类图描述静态结构*
* *出发点不同*
  + *用例分析使用****分析类****的概念分配职责*
  + *用例设计从设计元素，设计机制（来自架构设计），结合设计原则和模式，进行职责分配*
* *关注点不同*
  + *用例设计关注职责如何实现，目标类提供什么操作才可以响应消息*
  + *用例分析是目标类需提供什么样的职责*