

《系统分析与设计》

System Analysis and Design

任课教师： 范 国 祥

电 话： 0451-86418876-811(O)
13199561265(微信同号)

邮 箱： fgx@hit.edu.cn

哈工大计算学部/
国家示范性软件学院
软件工程教研室

2022. 03



本章主要内容

1. 面向对象设计概述
2. 系统设计
3. 包的设计
4. 对象设计
5. 面向对象设计总结



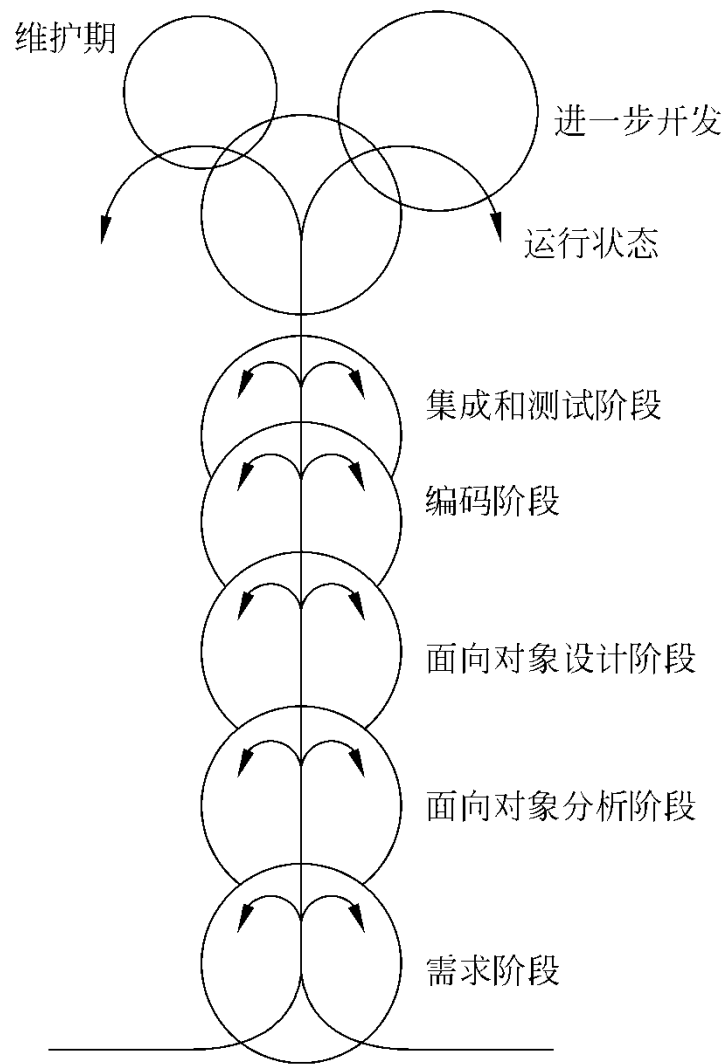
本章主要内容

1. 面向对象设计概述
2. 系统设计
3. 包的设计
4. 对象设计
5. 面向对象设计总结



面向对象的设计

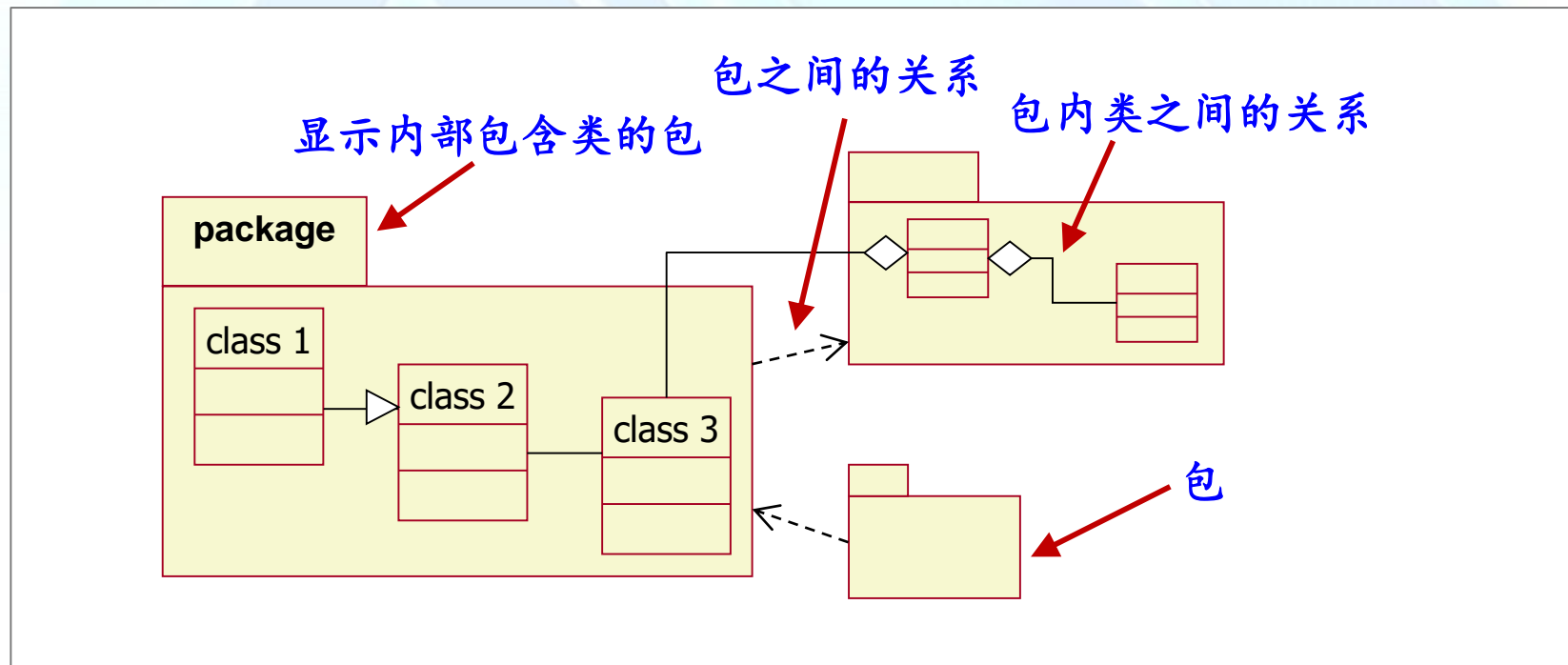
- 传统的结构化方法：
分析阶段与设计阶段分得特别清楚，分别使用两套完全不同的建模符号和建模方法
- 面向对象的设计(OOD):
OO各阶段均采用统一的“对象”概念，各阶段之间的区分变得不明显，形成“无缝”连接
- 因此，OOD中仍然使用“类、属性、操作”等概念，是在OOA基础上的进一步细化，更加接近底层的技术实现





面向对象设计中的基本元素

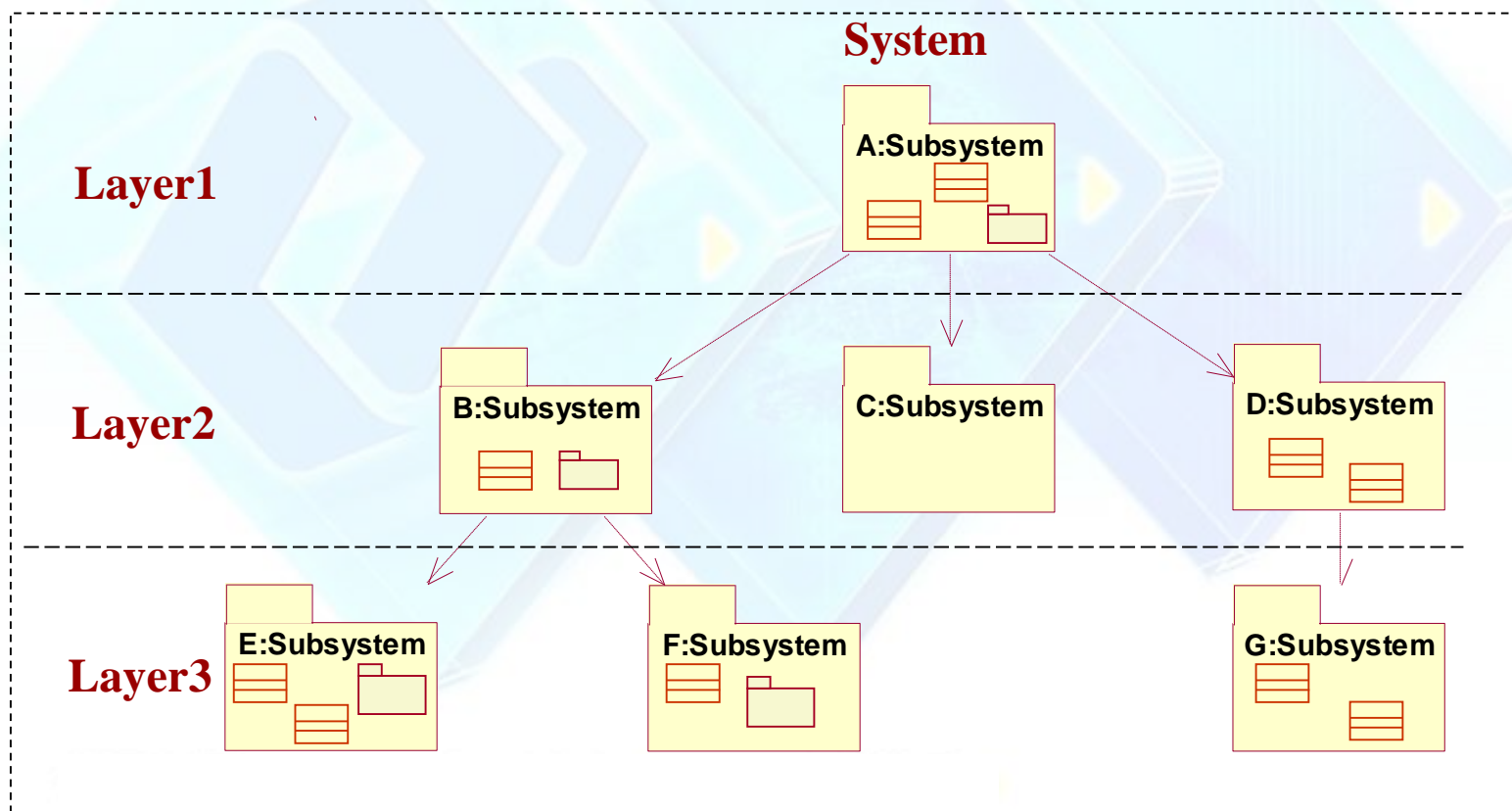
- 基本单元：设计类(design class)
 - 对应于OOA中的分析类(analysis class)
- 为了系统实现与维护过程中的方便性，将多个设计类按照彼此关联的紧密程度聚合到一起，形成大粒度的包(package)





面向对象设计中的基本元素

- 一个或多个包聚集在一起，形成子系统(sub-system)
- 多个子系统，构成完整的系统(system)



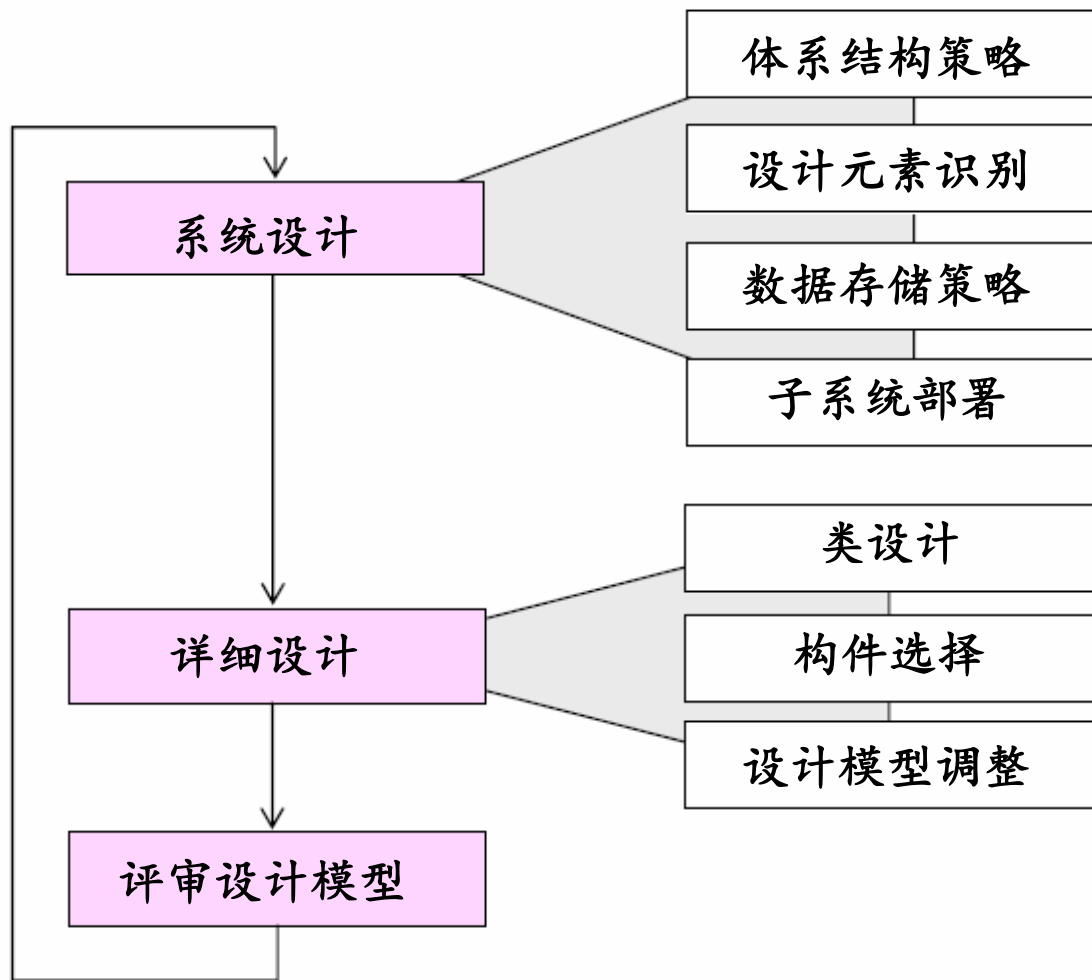


面向对象的设计的两个阶段

- 系统设计(System Design)
 - 相当于概要设计（即设计系统的体系结构）
 - 选择解决问题的基本途径（框架、平台、工具…）
 - 决策整个系统的结构与风格（拓扑结构、软件体系结构…）
- 对象设计(Object Design)
 - 相当于详细设计（即设计对象内部的具体实现）
 - 细化需求分析模型
 - 识别新的对象
 - 在系统所需的应用对象与可复用的商业构件之间建立关联：
 - 识别系统中的应用对象
 - 调整已有的构件
 - 给出每个子系统/类的精确规格说明



面向对象设计的过程





本章主要内容

1. 面向对象设计概述
2. 系统设计
3. 包的设计
4. 对象设计
5. 面向对象设计总结



系统设计概述

- 设计系统的体系结构 -- 在课程《软件体系结构》重点讲述
 - 选择合适的分层体系结构策略，建立系统的总体结构：分几层？每层的功能分别是什么？
- 识别设计元素 -- 详见后续章节
 - 识别“设计类” (design class)、“包” (package)、“子系统” (sub-system)
- 部署子系统 -- 在课程《软件体系结构》重点讲述
 - 选择硬件配置和系统平台，将子系统分配到相应的物理节点，绘制部署图 (deployment diagram)
- 定义数据的存储策略 -- 详见后续章节
- 检查系统设计



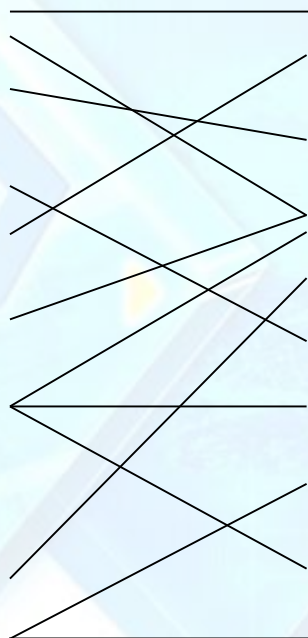
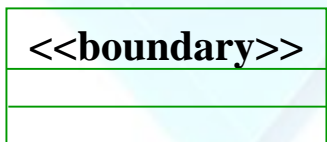
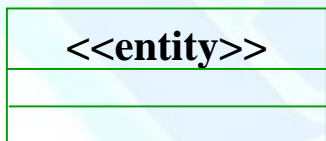
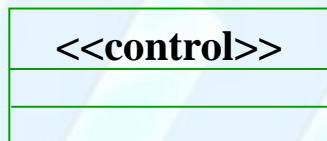
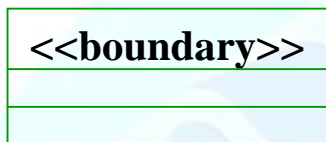
本章主要内容

1. 面向对象设计概述
2. 系统设计
3. 包的设计
4. 对象设计
5. 面向对象设计总结

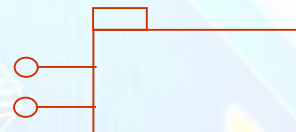
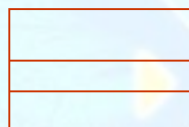


识别设计元素

分析类



设计元素



多对多映射



确定设计元素的基本原则


- 如果一个“分析类”比较简单，代表着单一的逻辑抽象，那么可以将其**一对一的映射为“设计类”**
 - 通常，主动参与者对应的边界类、控制类和一般的实体类都可以直接映射成设计类
- 如果“分析类”的职责比较复杂，很难由单个“设计类”承担，则应该将其**分解为多个“设计类”，并映射成“包”或“子系统”**
- 将设计类分配到相应的“包”或“子系统”当中
 - 子系统的划分应该符合高内聚低耦合的原则



图书管理系统：识别设计元素

类型	分析类	设计元素
	<i>LoginForm</i>	“设计类” <i>LoginForm</i>
	<i>BrowseForm</i>	“设计类” <i>BrowseForm</i>

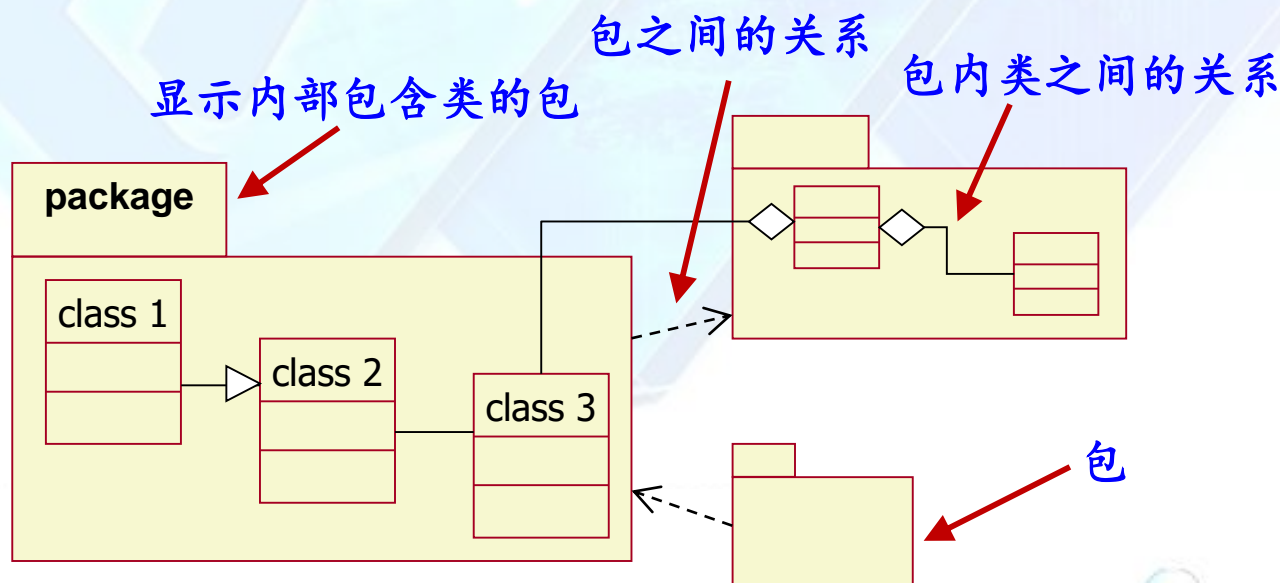
	<i>MailSystem</i>	“子系统接口” <i>IMailSystem</i>
	<i>BrowseControl</i>	“设计类” <i>BrowseControl</i>
	<i>MakeReservationControl</i>	“设计类” <i>MakeReservationControl</i>

	<i>BorrowerInfo</i>	“设计类” <i>BorrowerInfo</i>
	<i>Loan</i>	“设计类” <i>Loan</i>



绘制包图(package diagram)

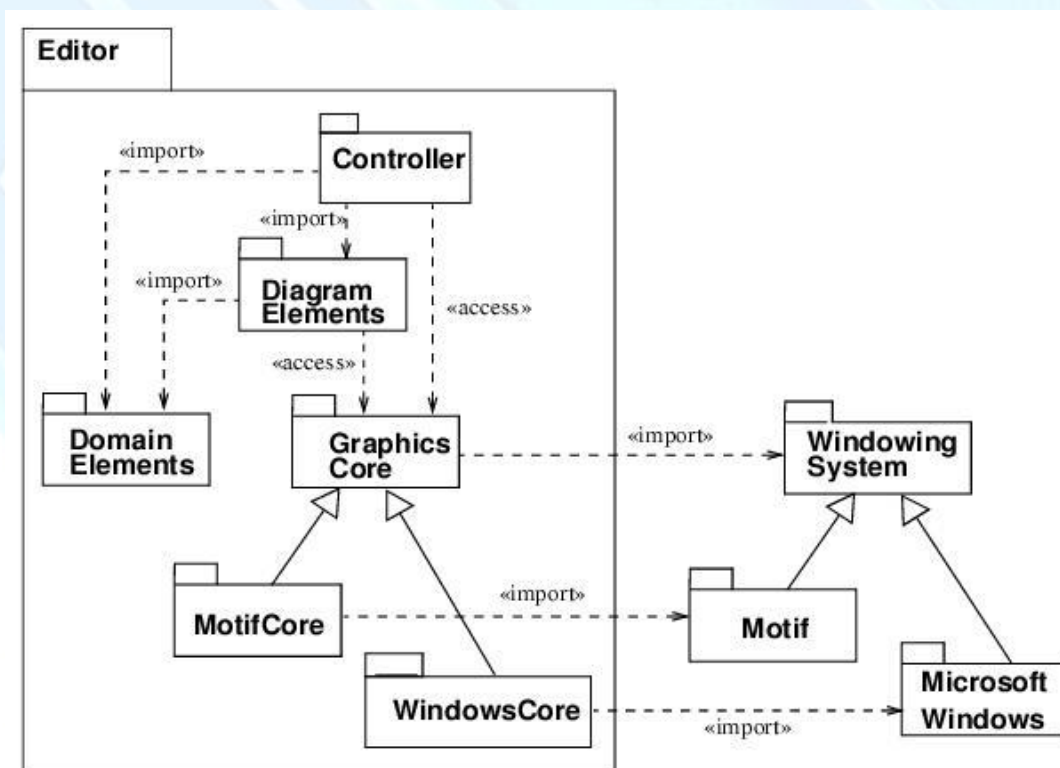
- 对一个复杂的软件系统，要使用大量的设计类，这时就必要把这些类分组进行组织
- 把在语义上接近且倾向于一起变化的类组织在一起形成“包”，既可控制模型的复杂度，有助于理解，而且也有助于按组来控制类的可见性
- 结构良好的包是松耦合、高内聚的，而且对其内容的访问具有严密的控制





包之间的关系

- 类与类之间存在的“聚合、组合、关联、依赖”关系导致包与包之间存在依赖关系，即**包的依赖(dependency)**
- 类与类之间存在的“继承”关系导致包与包之间存在继承关系，即**包的泛化(generalization)**





模型管理视图——包图

- 包图(Package Diagram)是在 UML 中用类似于文件夹的符号表示的模型元素的组合
 - UML包图提供了组织元素的方式，包能够组织任何事物：
 - 类、其它包、用例等
 - 系统中的每个元素都只能为一个包所有
 - 一个包可嵌套在另一个包中
- 注：UML中的包为广义概念，不等同于Java包的狭义概念
- 类、接口、组件、节点、协作、用例、其他包
 - 使用包图可以将相关元素归入一个系统
 - 一个包中可包含附属包、图表或单个元素

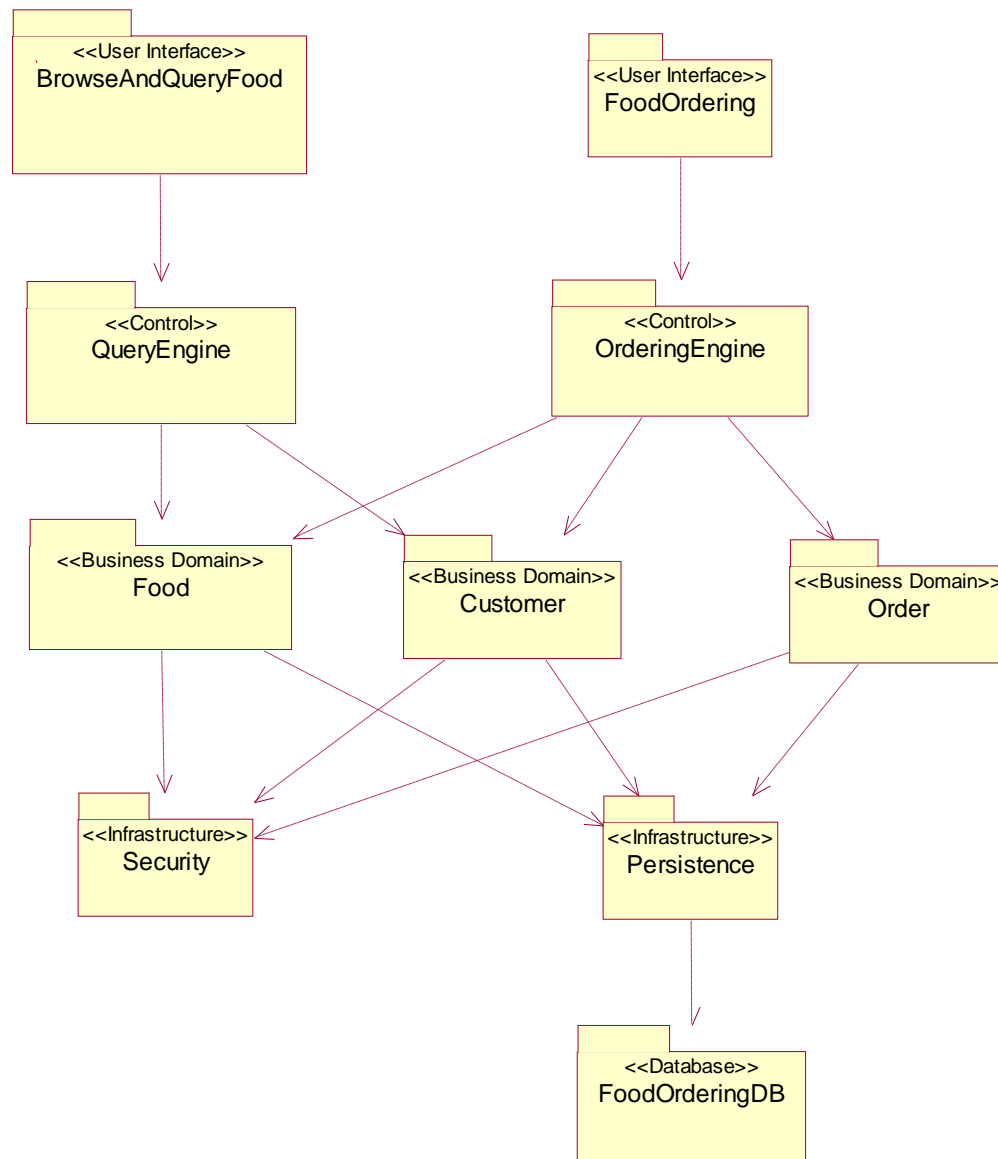


绘制包图的方法

- 分析设计类，把概念上或语义上相近的模型元素纳入一个包
- 可以从类的功能相关性来确定纳入包中的类：
 - 如果一个类的行为和/或结构的变更要求另一个相应的变更，则这两个类是功能相关的
 - 如果删除一个类后，另一个类便变成是多余的，则这两个类是功能相关的，这说明该剩余的类只为那个被删除的类所使用，它们之间有依赖关系
 - 如果两个类之间大量的频繁交互或通信，则这两个类是功能相关的
 - 如果两个类之间有一般/特殊关系，则这两个类是功能相关的
 - 如果一个类激发创建另一个类的对象，则这两个类是功能相关的
- 确定包与包之间的依赖关系(<<import>>、<<access>>等)
- 确定包与包之间的泛化关系
- 绘制包图

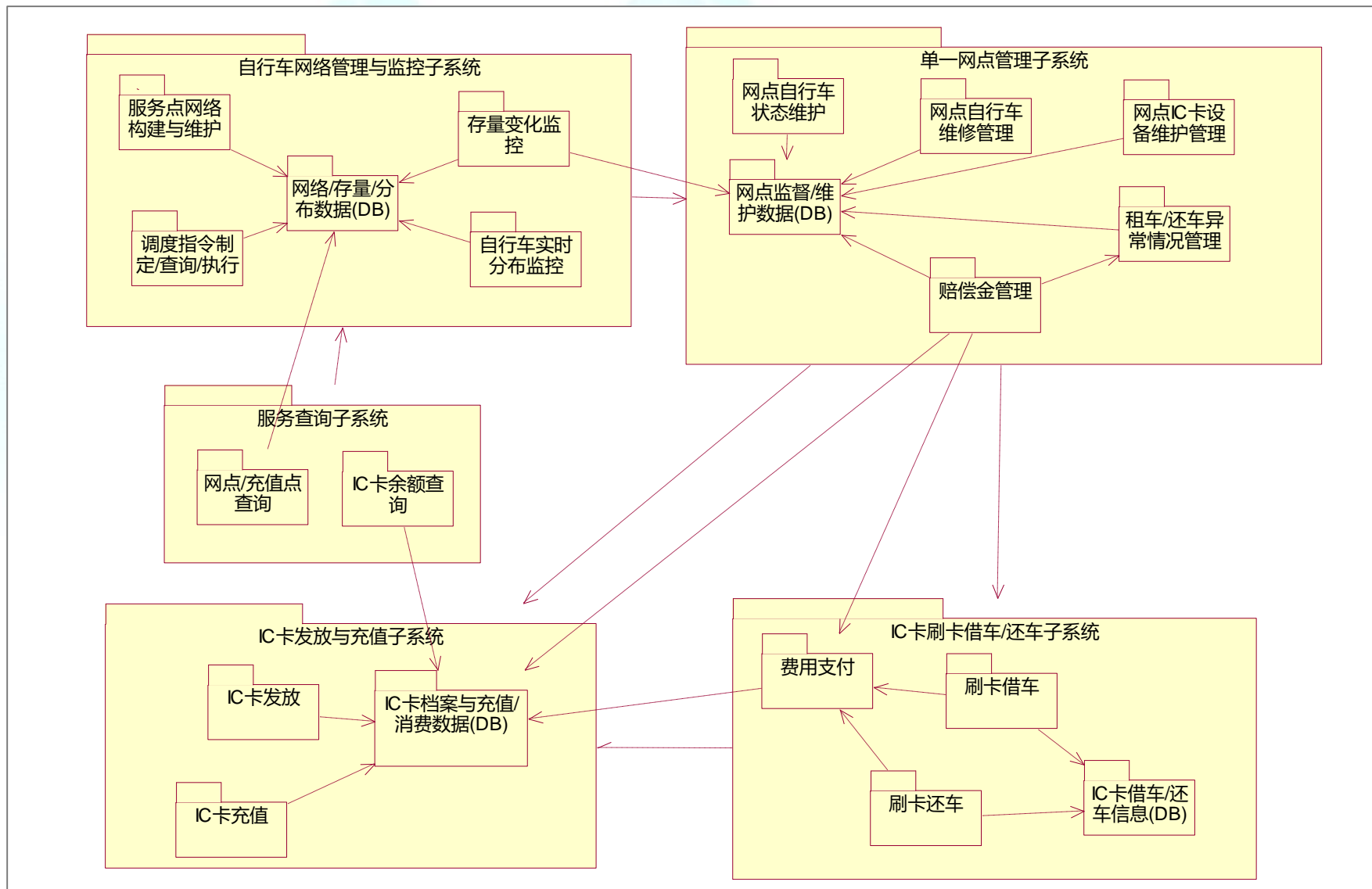


包图示例1



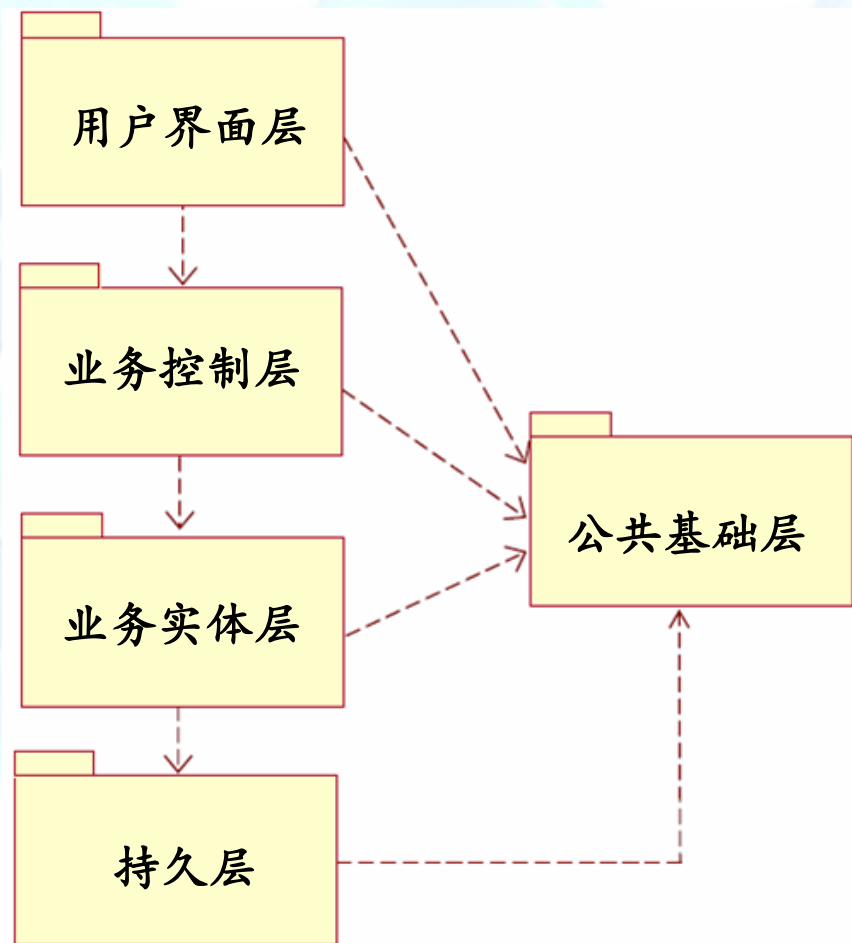


包图示例2



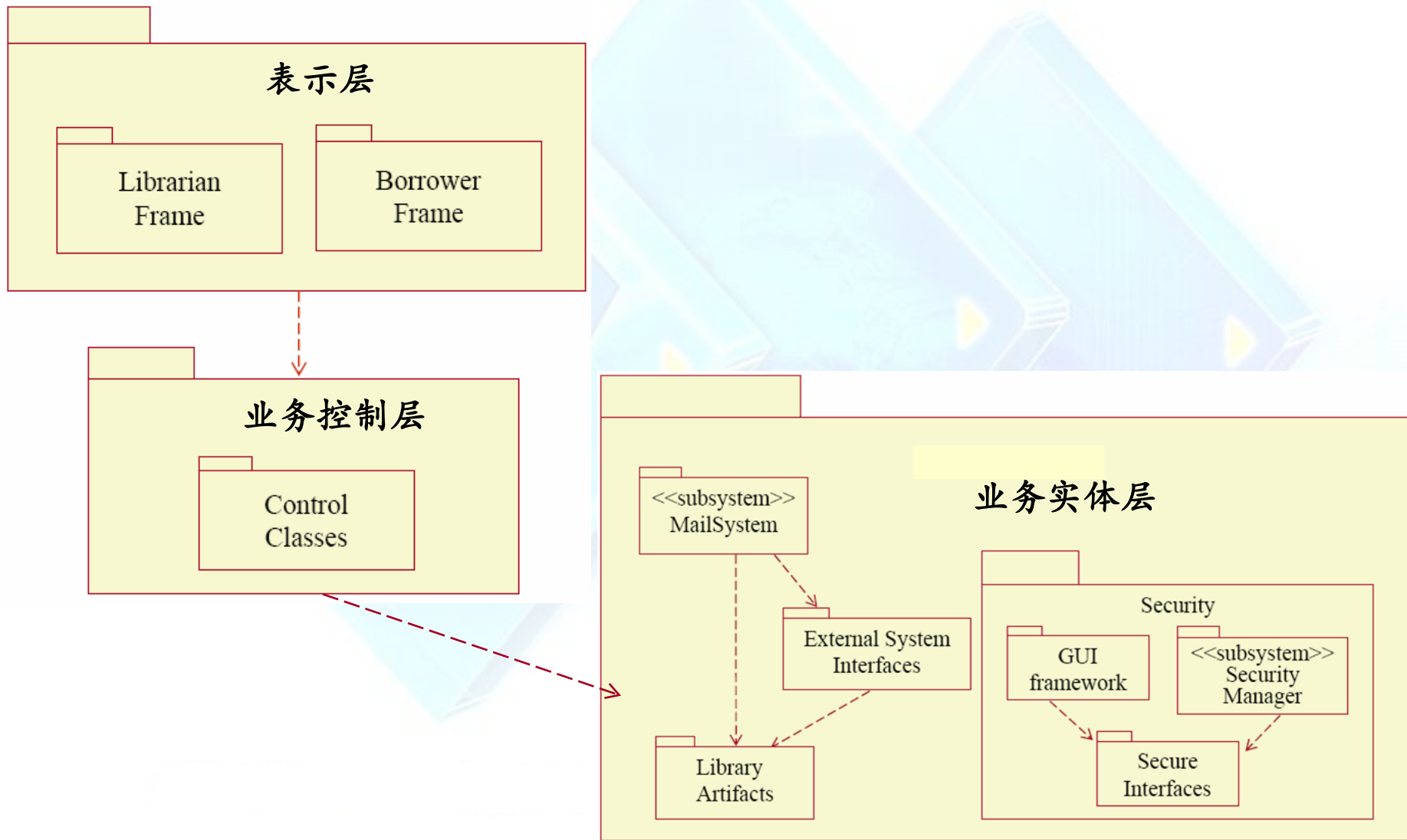


按实现技术划分包的分层结构





图书管理系统：软件体系结构





JAVA 中的 “package”

```
java.io.InputStream is = java.lang.System.in;
```

```
java.io.InputStreamReader isr= new java.io.InputStreamReader(is);
```

```
java.io.BufferedReader br = new java.io.BufferedReader(isr);
```

```
import java.lang.System;
```

```
import java.io.InputStream;
```

```
import java.io.InputStreamReader;
```

```
import java.io.BufferedReader;
```

```
InputStream = System.in;
```

```
InputStreamReader isr = new InputStreamReader(is);
```

```
BufferedReader br = new BufferedReader(isr);
```



JAVA中的“package”

package cn.edu.hit.cs;

public class A{ }

package cn.edu.hit.cs;

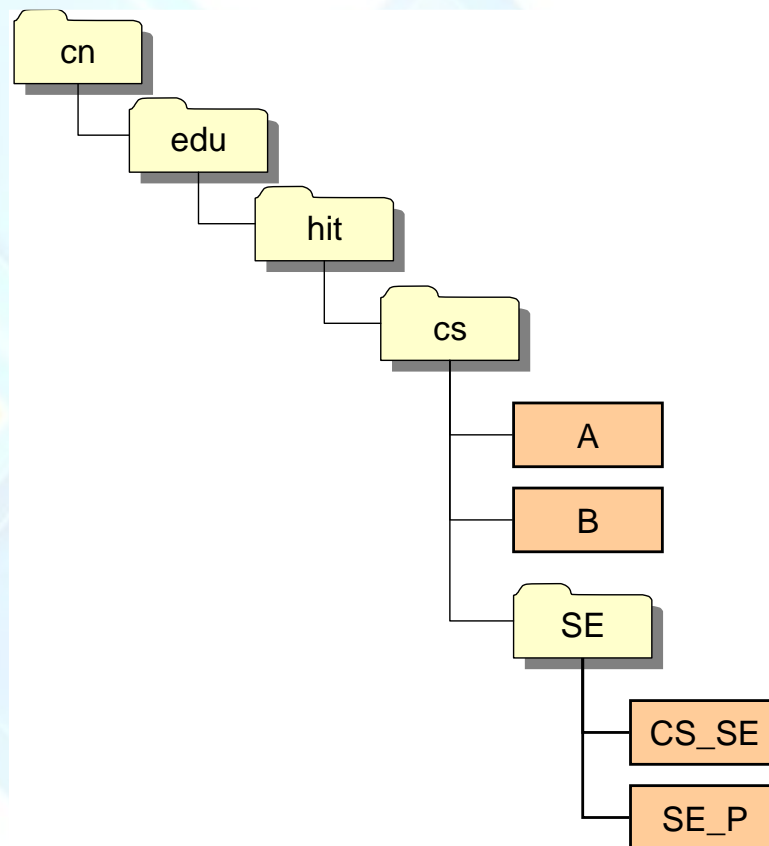
public class B{ }

package cn.edu.hit.cs.se;

public class CS_SE{ }

package cn.edu.hit.cs.se;

public class CS_SE_P{ }





检查系统设计

■ 检查“正确性”

- 每个子系统都能追溯到一个用例或一个非功能需求吗？
- 每一个用例都能映射到一个子系统吗？
- 系统设计模型中是否提到了所有的非功能需求？
- 每一个参与者都有合适的访问权限吗？
- 系统设计是否与安全性需求一致？

■ 检查“一致性”

- 是否将冲突的设计目标进行了排序？
- 是否有设计目标违背了非功能需求？
- 是否存在多个子系统或类重名？



检查系统设计

■ 检查“完整性”

- 是否处理边界条件？
- 是否有用例走查来确定系统设计遗漏的功能？
- 是否涉及到系统设计的所有方面(如硬件部署、数据存储、访问控制、遗留系统、边界条件)？
- 是否定义了所有的子系统？

■ 检查“可行性”

- 系统中是否使用了新的技术或组件？是否对这些技术或组件进行了可行性研究？
- 在子系统分解环境中检查性能和可靠性需求了吗？
- 考虑并发问题了吗？



本章主要内容

1. 面向对象设计概述
2. 系统设计
3. 包的设计
- 4. 对象设计**
5. 面向对象设计总结



对象设计概述

- 对象设计
 - 细化需求分析和系统设计阶段产生的模型
 - 确定新的设计对象
 - 消除问题域与实现域之间的差距
- 对象设计的主要任务
 - 精化类的属性和操作
 - 明确定义操作的参数和基本的实现逻辑
 - 明确定义属性的类型和可见性
 - 明确类之间的关系
 - 整理和优化设计模型



对象设计的基本步骤

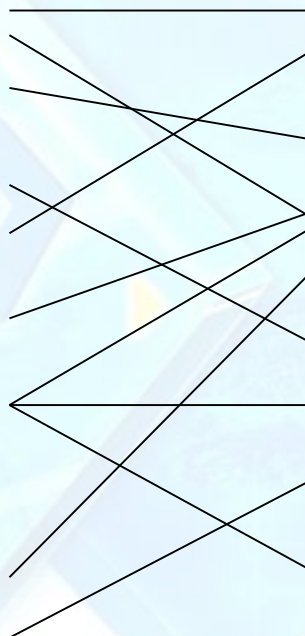
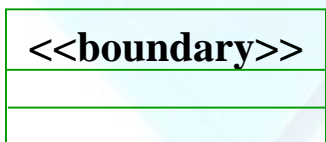
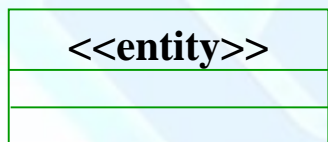
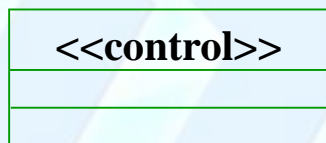
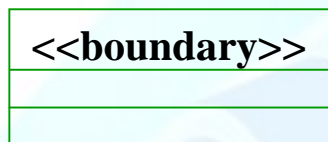
1. 创建初始的设计类
2. 细化属性
3. 细化操作
4. 定义状态
5. 细化依赖关系
6. 细化关联关系
7. 细化泛化关系



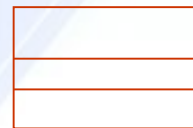
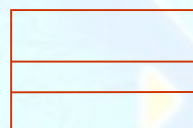
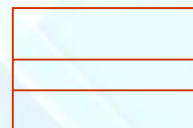


1. 创建初始的设计类

分析类



设计类





2. 细化属性

■ 细化属性

- 具体说明属性的名称、类型、缺省值、可见性

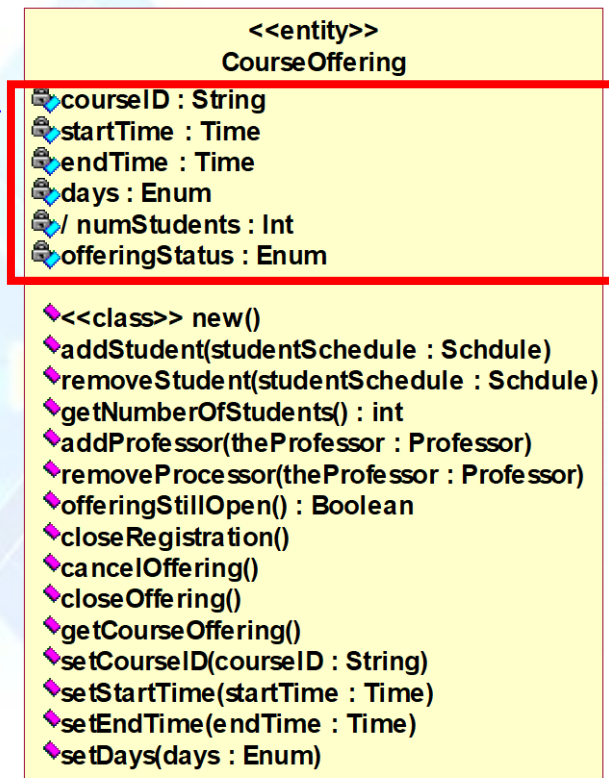
visibility attributeName : Type = Default

- **Public: +** 对能看到该类的任何元素都可见
- **Private: -** 只对本类的任何元素可见
- **Protected: #** 对该类及子孙类的任何元素可见
- **Package: ~** 对同包中的任何元素可见

■ 属性的来源:

- 类所代表的现实实体的**基本信息**
- **描述状态的信息**
- **描述该类与其他类之间关联的信息**
- **派生属性(derived attribute):** 该类属性的值需要通过计算其他属性的值才能得到, 属性前面加 “/” 表示

例如: 类CourseOffering中的“学生数目” / numStudents : int





细化属性

■ 基本原则

- 属性命名符合规范(名词组合, 首字母小写)
- 尽可能将所有属性的可见性设置为`private`
- 仅通过`set`方法更新属性
- 仅通过`get`方法访问属性
- 在属性的`set`方法中, 实现简单的有效性验证, 而在独立的验证方法中实现复杂的逻辑验证



关于状态属性

- 状态由一个或多个属性来表示（**实体类中经常含有状态属性**）
 - 对“订单”类来说，可以设定一个状态属性“订单状态”，取值为enum{未付款、取消、已付款未发货、已发货、已确认收货未评价、买方已评价、双方已评价、...}
 - 也可以由多个属性来表示状态：是否已付款、是否已发货、是否已确认、买方是否已评价、卖方是否已评价、...
 - 每个状态属性的类型多数用boolean
- 大部分状态属性，可以由该类的其他属性的值进行逻辑判断得到
 - 若订单处于“未付款”状态，则该订单的“订单变迁记录”中一定不包含有付款信息
 - 若它处于“买家已评价”状态，则它的“买家评价”属性一定不为空
- 从一个**状态值**到另一个状态值的**变迁**，一定是由该实体类的**某个操作**所导致的
 - 订单从不存在到变为“未付款”，是由new操作导致的状态变化
 - 订单从“已发货”到“已确认”的变化，是由“确认收货”操作所导致的状态变化
 - 检查状态变迁的条件，可以判断你为实体类所设计的操作是否完整



关于关联属性

- 两个类之间有association关系，意味着需要永久管理对方的信息，需要在程序中能够从类1的object“导航”(navigate)到类2的object
 - 例如：“订单”类与“买家”类产生双向关联，意即一个订单对象中需要能够找到相应的“买家”对象，反之买家对象需要知道自己拥有哪些订单对象
 - 订单类中有一个关联属性buyer，其数据类型是“买家”类
 - 买家类中有一个关联属性OrderList，其数据类型是“订单”类构成的序列
- 关联属性不只是一个ID，而是一个或多个完整的对方类的对象
- 务必与数据库中的“外键”区分开：
 - 例如：订单table中有一个外键（只是一个ID值）：买家ID(字符串类型)，靠它与买家table联系起来
- 不要用关系数据模式的设计思想来构造类的属性
- 关联属性的目标：
 - 在程序运行空间内实现object之间的导航，而无需经过数据库层存取



3. 细化操作

- 找出满足基本逻辑要求的操作：
 - 从actor出发，分别思考需要类的哪些操作
- 补充必要的辅助操作：
 - 初始化类的实例、销毁类的实例 — Student(...), ~Student()
 - 验证两个实例是否等同 — equals()
 - 获取属性值(get操作) — getXXX()
 - 设定属性值(set操作) — setXXX(...)
 - 将对象转换为字符串 — toString()
 - 复制对象实例 — clone()
 - 用于测试类的内部代码的操作 — main()
 - 支持对象进行状态转换的操作 —
- 细化操作时，要充分考虑类的“属性”与“状态”是否被充分利用：
 - 对属性进行CRUD
 - 对状态进行各种变更



细化操作

- 给出完整的操作描述：
 - 确定操作的名称、参数、返回值、可见性等
 - 应该遵从程序设计语言的命名规则(动词+名词, 首字母小写)
- 详细说明操作的内部实现逻辑
 - 复杂的操作过程采用伪代码或者绘制程序流程图/活动图方式
- 在给出内部实现逻辑之后, 可能需要:
 - 将各个操作中公共部分提取出来, 形成独立的新操作



细化操作

■ 操作的形式:

visibility opName (param : type = default, ...) : returnType

■ 一个例子: CourseOffering

— Constructor

<<class>>new ()

— Set attributes

+ setCourseID (courseID:String)

+ setStartTime (startTime:Time)

+ setEndTime (endTime:Time)

+ setDays (days:Enum)

— Others

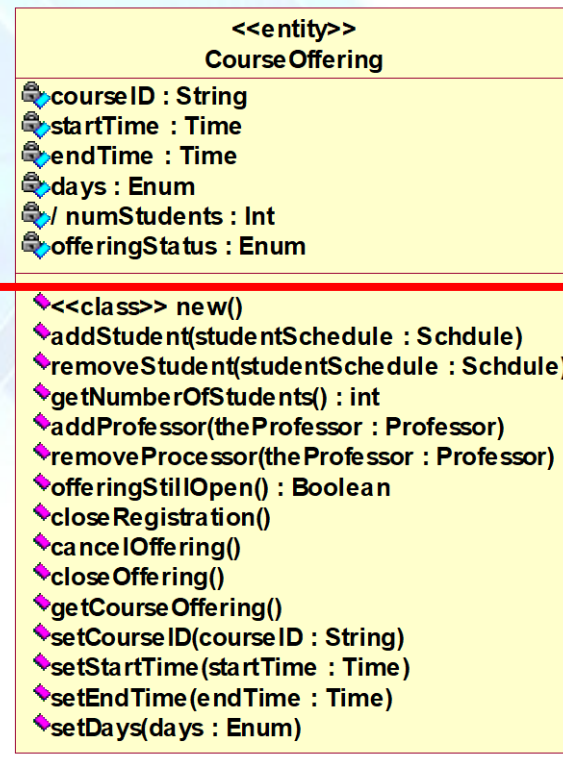
+ addProfessor (theProfessor:Professor)

+ removeProfessor (theProfessor:Professor)

+ offeringStillOpen () : Boolean

+ getNumberOfStudents () : int

... ..





细化操作

■ 一个例子: *BorrowerInfo*类

— 构造函数

`<<class>> + new ()`

— 属性赋值

`+ setName(name:String)`

`+ setAddress(address:String)`

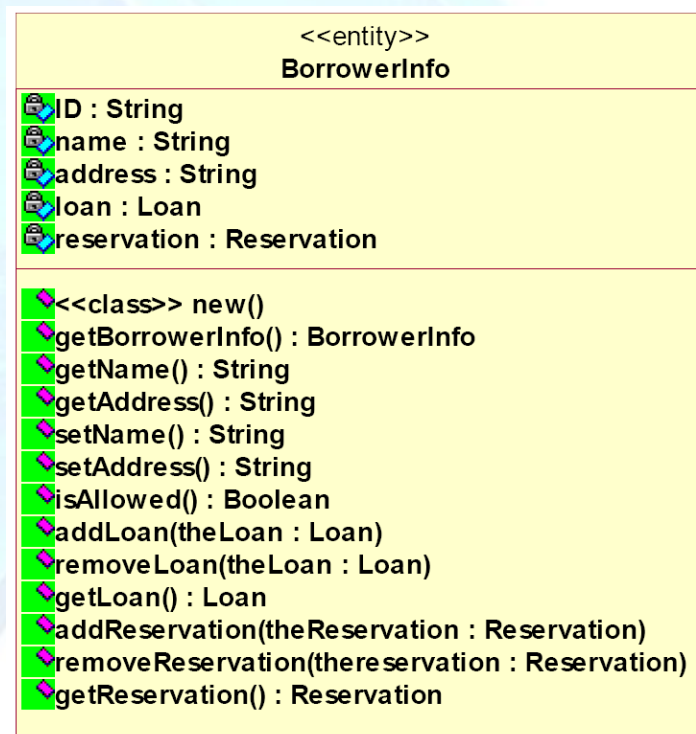
— 其他

`+ addLoan(theLoan:Loan)`

`+ removeLoan(theLoan:Loan)`

`+ isAllowed() : Boolean`

... ..





细化操作

new ()

```
offeringStatus := unassigned;  
numStudents := 0;
```

addProfessor (theProfessor : Professor)

```
if offeringStatus = unassigned {  
    offeringStatus := assigned;  
    courseInstructor := theProfessor;  
}  
else errorState ( );
```

removeProfessor (theProfessor : Professor)

```
if offeringStatus = assigned {  
    offeringStatus := unassigned;  
    courseInstructor := NULL;  
}  
else errorState ( );
```

closeOffering ()

```
switch ( offeringStatus ) {  
    case unassigned:  
        cancelOffering ( );  
        offeringStatus := cancelled;  
        break;  
    case assigned:  
        if ( numStudents < minStudents )  
            cancelOffering ( );  
        offeringStatus := cancelled;  
        else  
            offeringStatus := committed;  
        break;  
    default: errorState ( );  
}
```



4* 定义状态

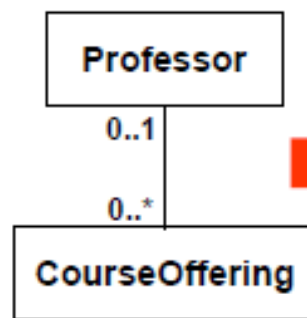
- 目的：
 - 设计一个对象的状态是如何影响其行为的
 - 绘制对象状态图
- 需要考虑的要素：
 - 哪些对象有状态？
 - 如何发现对象的所有状态？
 - 如何绘制对象状态图？
- Example: CourseOffering

numStudents < maximum

Open

numStudents ≥ maximum

Closed



Professor Exists

Assigned

Professor
Doesn't Exist

Unassigned



5. 细化类之间的关系

- 细化关系：关联关系、依赖关系、继承关系、组合和聚合关系
- “继承”关系很清楚
- 在对象设计阶段，需要进一步确定详细的关联关系、依赖关系和组合/聚合关系等
- 不同对象之间的可能连接：四种情况：

— 全局(Global)

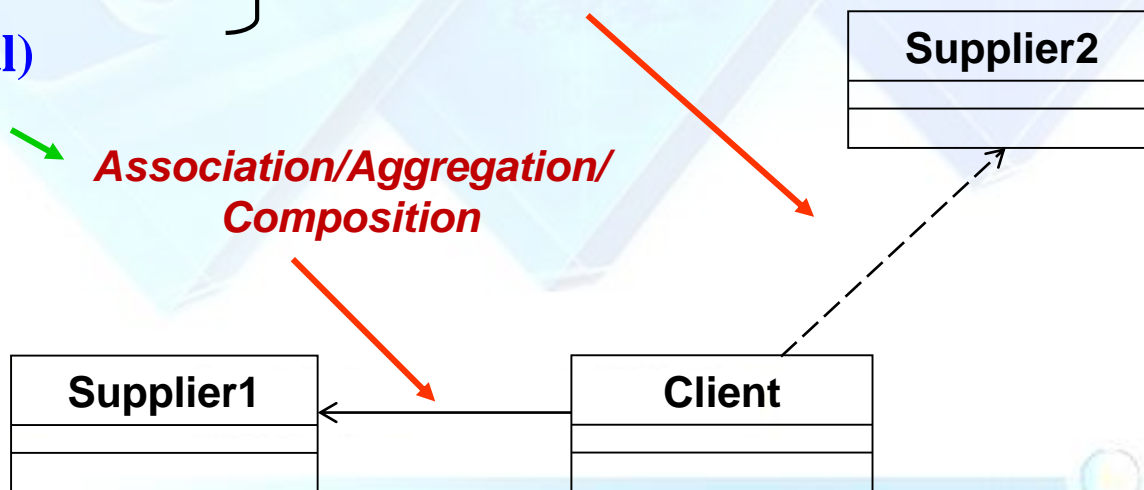
— 参数(Parameter)

— 局部(Local)

— 域(Field)

} **Dependency**

**Association/Aggregation/
Composition**

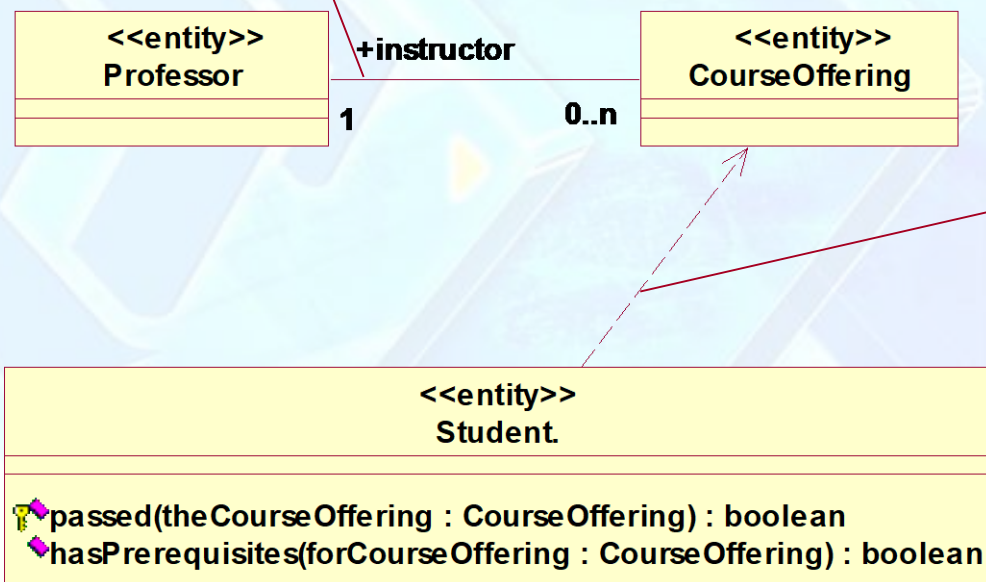




定义类之间的关系：示例

Field reference

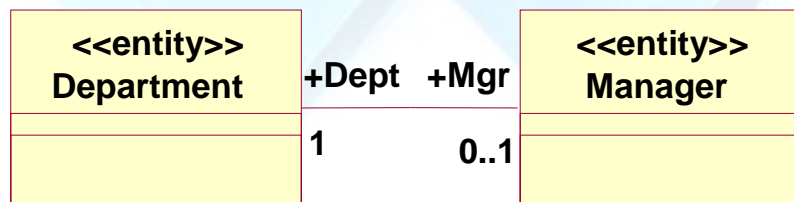
Parameter reference





定义关联关系 (Association/Composition/Aggregation)

- 根据“多重性”进行设计 (multiplicity-oriented design)
- 情况1: Multiplicity = 1 或 Multiplicity = 0..1
 - 可以直接用一个单一属性/指针加以实现, 无需再作设计
 - 若是双向关联:
 - Department类中有一个属性: +Mgr: Manager
 - Manager类中有一个属性: +Dept: Department
 - 若是单向关联:
 - 只在关联关系发出的类中增加关联属性



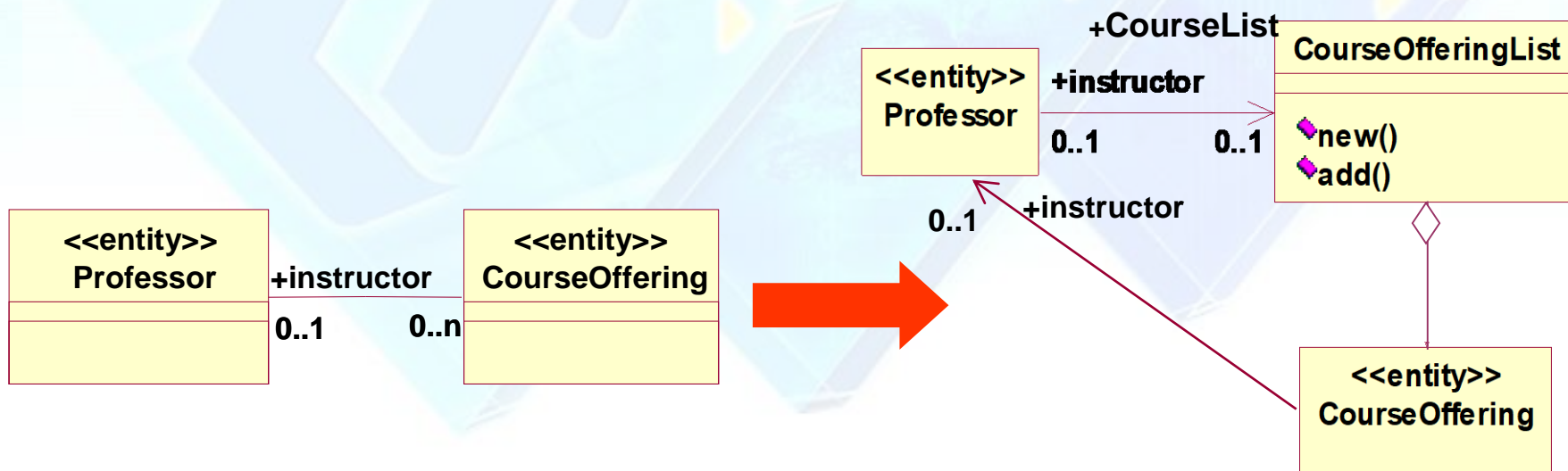
设计阶段需要将这种
“关联属性”增加到
属性列表中, 并更新
操作列表

分析阶段则不需要在
属性列表中加入“关
联属性”



定义关联关系 (Association/Composition/Aggregation)

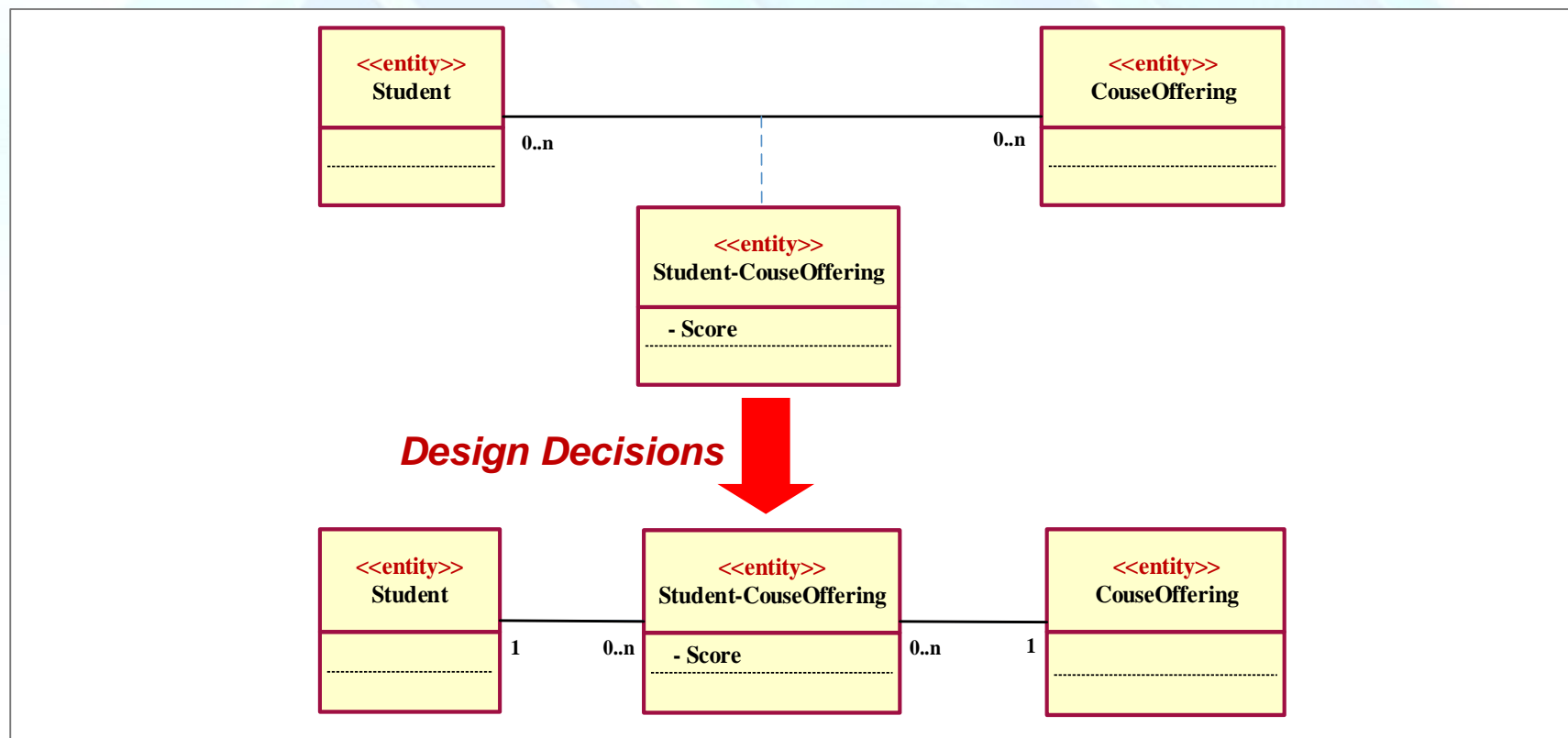
- 根据“多重性”进行设计(multiplicity design)
- 情况2: Multiplicity > 1
 - 无法用单一属性/指针来实现, 需要引入新的设计类或能够存储多个对象的复杂数据结构(例如链表、数组等)
 - 将1:n转化为若干个1:1





定义关联关系 (Association/Composition/Aggregation)

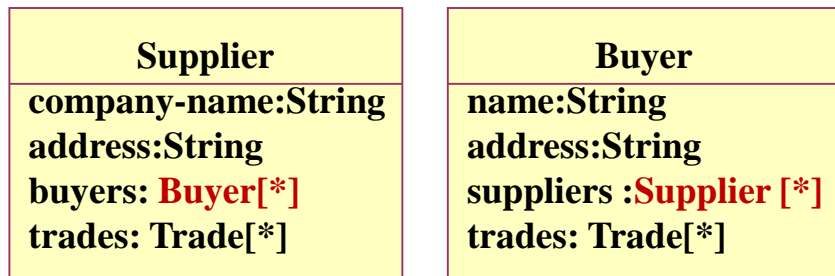
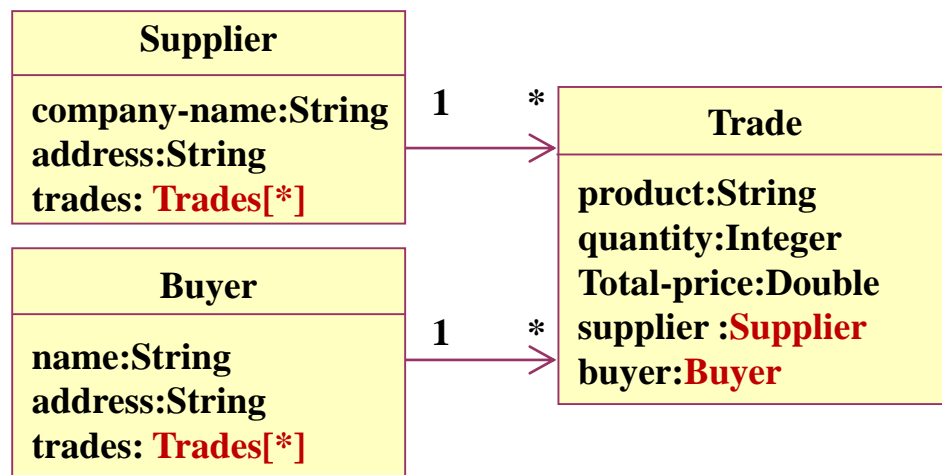
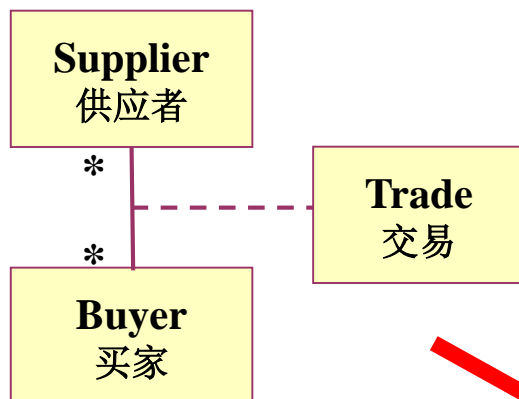
- 情况3: 有些情况下, 关联关系本身也可能具有属性, 可以使用**关联类**将这种关系建模
- 举例: 学生**Student**与开设课程**CourseOffering**之间的关联关系





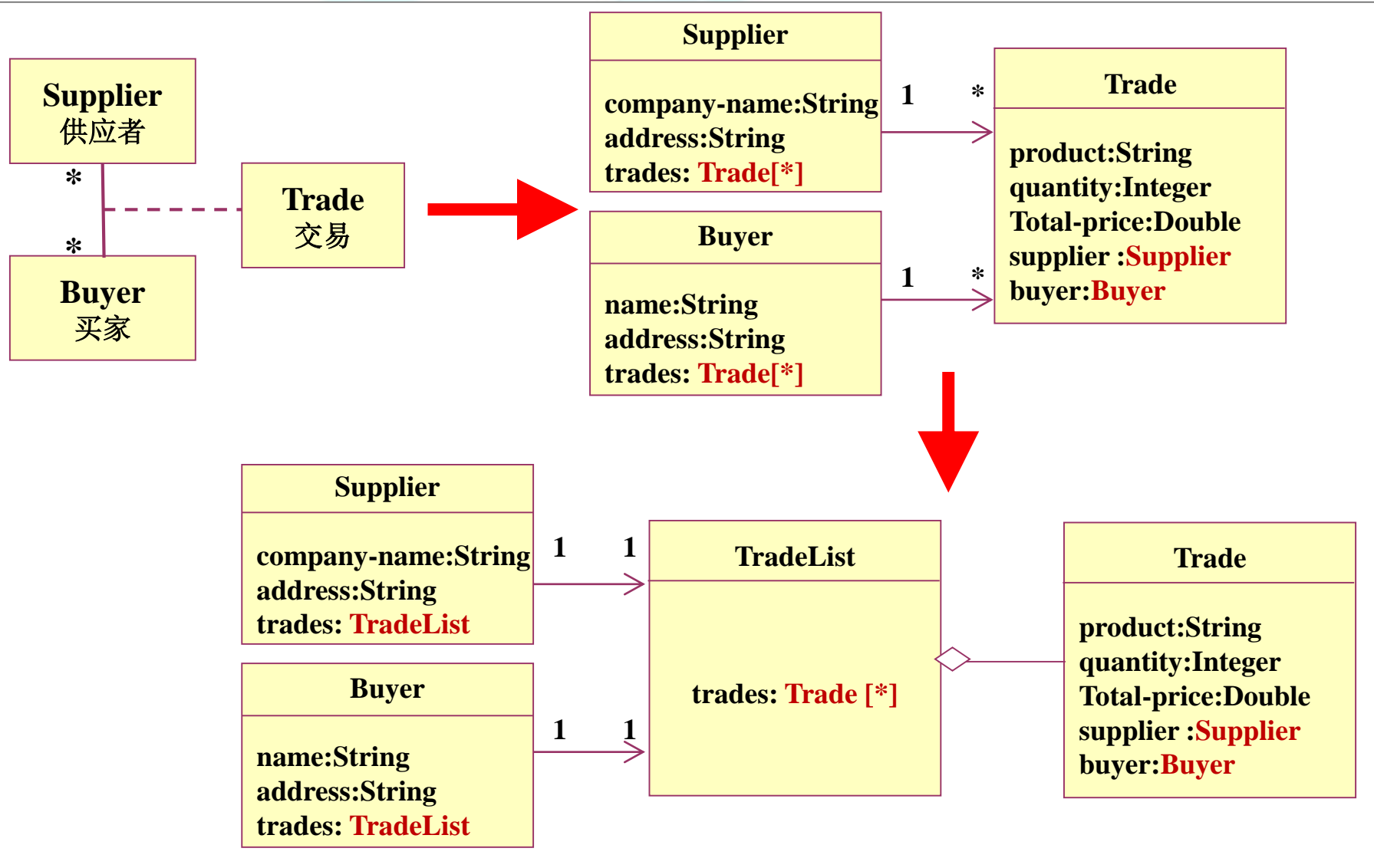
定义关联关系 (Association/Composition/Aggregation)

使用关联类和不使用关联类的对比分析





定义关联关系 (Association/Composition/Aggregation)





引入“辅助类”简化类的内部结构

- 辅助实体类，是对从用例中识别出的核心实体的补充描述，目的是使每个实体类的属性均为简单数据类型：
 - 何谓“简单数据类型”？编程语言提供的基本数据类型(int, double, char, string, boolean, list, vector等，以及其他实体类)
 - 目的：使用起来更容易
- 例如对“订单”类来说，需要维护收货地址相关属性，而收货地址又由多个小粒度属性构成(收货人、联系电话、地址、邮编、送货时间)
 - **办法1**：这五个小粒度属性直接作为订单类的五个属性
实际上，这五个属性通常总是在一起使用，该办法会导致后续使用的麻烦
 - **办法2**：构造一个辅助类“收货地址”，订单类只保留一个属性，其类型为该辅助类
 - 在淘宝系统中，恰好还有用例是“增加、删除、修改收货地址”，故而设置这样一个辅助类是合适的
 - 这两个实体类之间形成**聚合关系**(收货地址可以独立于订单而存在)



引入“辅助类”简化类的内部结构

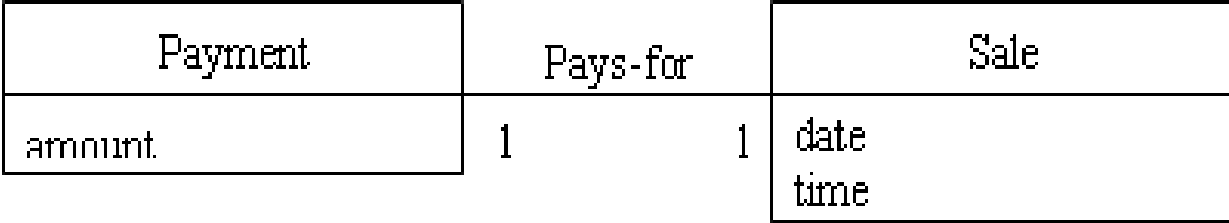
- 仍以订单类为例，订单的物流记录是一个非常复杂的结构体，由多行构成，每行又包含“时间(datetime)、流转记录(text)、操作人(text)”等属性，故而可以将“订单物流记录”作为一个实体类，包含着三个简单属性，而订单类中维护一个“订单流转记录(list)”属性，该属性是一个集合体，其成员元素的类型是“订单流转记录”这个实体类
- 这两个实体类之间形成组合关系(没有订单，就没有物流记录)
- 当查询订单的流转记录时，使用getXXX操作获得这个list属性，然后遍历每个要素，从中分别取出这三个基本属性即可



领域类图/分析类图 → 设计类图

Domain Model

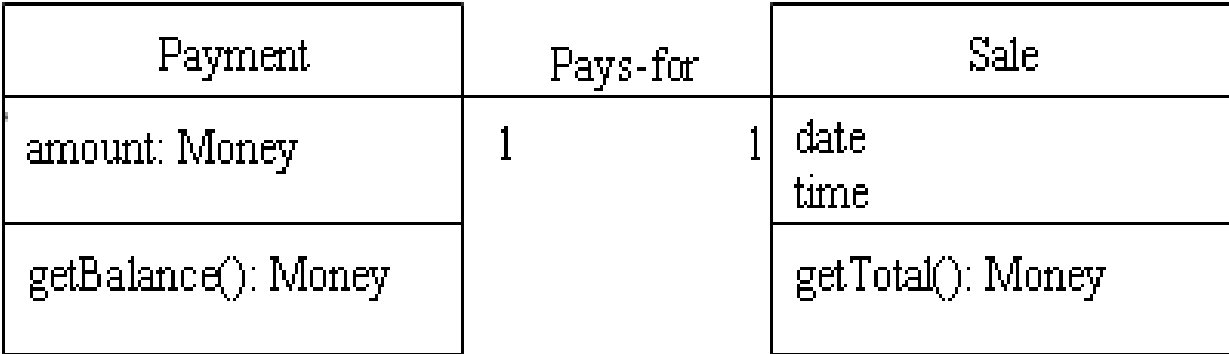
A payment in the domain model is a concept.



模块化 / OOP / 设计模式, etc!

Design Model

A payment in the design model is a software class.





本章主要内容

1. 面向对象设计概述
2. 系统设计
3. 包的设计
4. 对象设计
5. 面向对象设计总结



面向对象设计总结

- 系统设计
 - *包图(package diagram) → 逻辑设计
 - 部署图(deployment diagram) → 物理设计
- 对象设计
 - 设计类图(design class diagram) → 更新分析阶段的类图，对各个类给出详细的设计说明
 - *状态图(statechart diagram) → 使用设计类来更新分析阶段的状态图
 - *时序图(sequence diagram) → 使用设计类来更新分析阶段的时序图
 - 关系数据库设计方案(RDBMS design)
 - 用户界面设计方案(UI design)