类　图

内容

- 类图概述
- 类图基本概念
- 类图建模方法

内容

- 类图概述
  - 类图基本概念
  - 类图建模方法

类图概述

- 在面向对象的建模技术中，类、对象和它们之间的关系是最基本的建模元素。对于一个想要描述的系统，其类模型、对象模型以及它们之间的关系揭示了系统的结构。

类图概述

- 类图描述了系统中的类及其相互之间的关系，其本质反映了系统中包含的各种对象的类型以及对象间的各种静态关系。

类图概述

- 类图是面向对象系统建模中最常用的图。
- 类图是定义其它图的基础。
- 在类图的基础上可以使用其它图进一步描述系统其它方面的特征。

类图概述

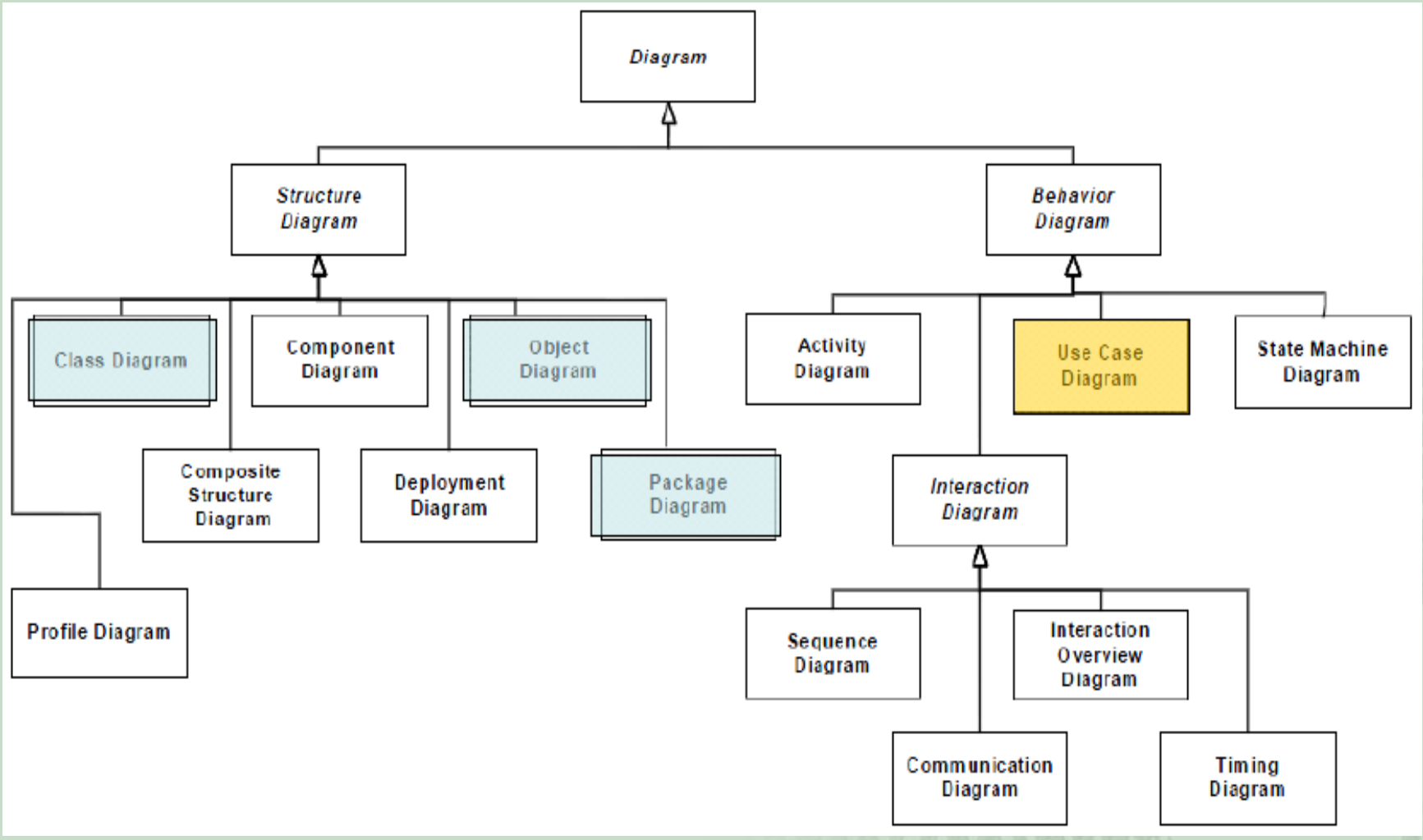


Diagram
Structure Diagram
Behavior Diagram
Class Diagram
Component Diagram
Object Diagram
Activity Diagram
Use Case Diagram
State Machine Diagram
Composite Structure Diagram
Deployment Diagram
Package Diagram
Interaction Diagram
Profile Diagram
Sequence Diagram
Interaction Overview Diagram
Communication Diagram
Timing Diagram

内容

- 类图概述
- **类图基本概念**
- 类图建模方法

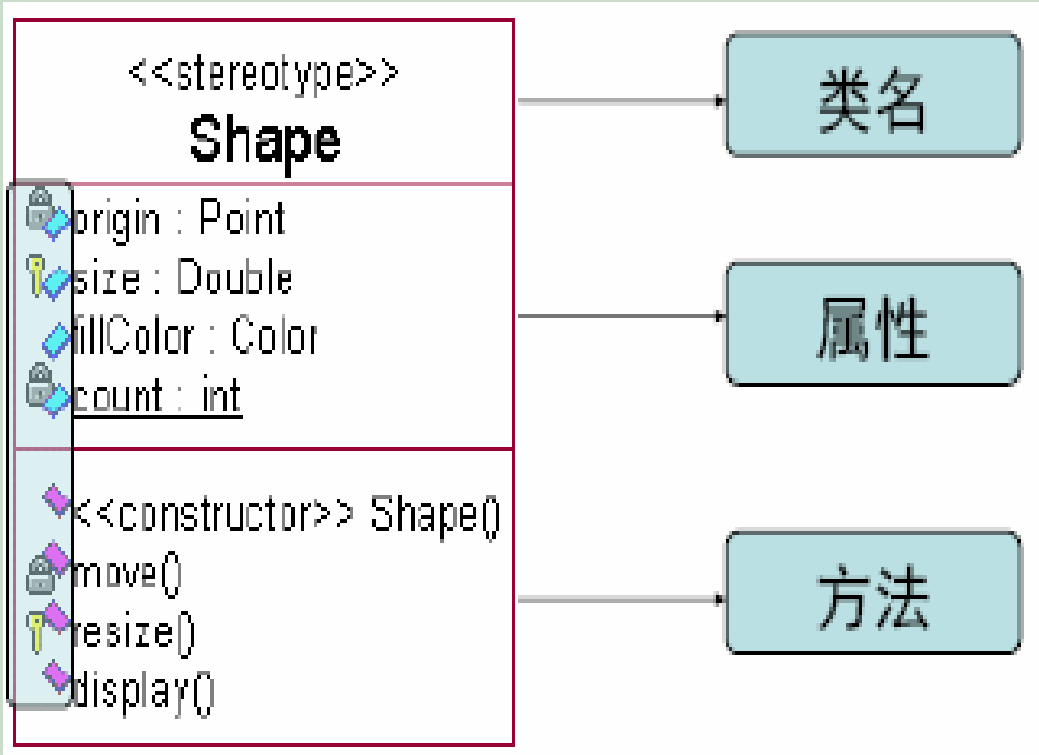
类

- 类是对现实世界实体的抽象，捕获并规定对系统而言比较重要的特征，同时隐藏那些无关的信息。
- 类是具有相似结构、行为和关系的一组对象的描述符（James Rumbaugh）。
- 类描述一组具有相同特征的对象。

类的表示

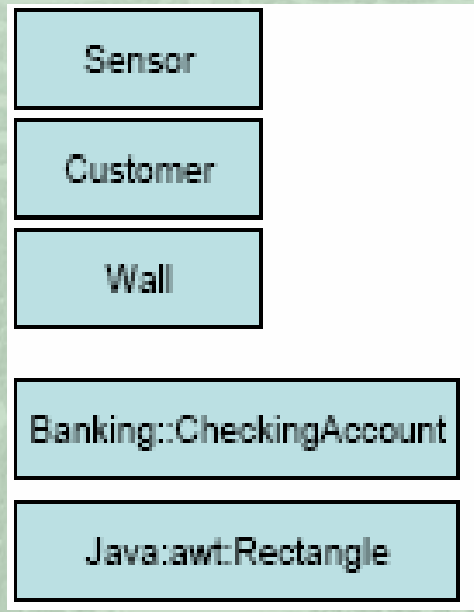
- 类的UML语法非常丰富。
- UML类符号包含三个分栏：名称分栏、属性分栏和操作分栏。
- 唯一强制部分是名称分栏，其它分栏是可选的，依赖于图的目的。

类的表示


<<stereotype>>
Shape
origin : Point
size : Double
fillColor : Color
count : int
<<constructor>> Shape()
move()
resize()
display()
类名
属性
方法

类的名称

- 命名建议
  - 应尽量用应用领域中的术语
  - 应明确、无歧义，以利于开发人员与用户之间的相互理解和交流
  - 一般而言，类的名字是名词
  - 每个单词以大写字母打头，避免使用特殊记号
- 命名分类
  - simple name，简单的类的名字
  - path name，包括包名



类的属性

- 属性是对类的对象的静态特性的抽象。
- 属性在类图标的属性分隔框中用文字串说明。
- 属性的描述格式为：

[可见性] 属性名[:类型] ['[' 多重性[次序] ']'] [= 初始值] [{特性}]

属性的可见性

- 属性可以具有不同的可见性。
- 可见性描述了该属性对于其它类的对象是否可见以及可引用。

属性的可见性

- UML支持的可见性类型
  - +:Public
  - #:Protected
  - -:Private
  - ~:Package
- Rational Rose中可见性的表示
  -  Public
  -  Protected
  -  Private
  -  Implementation

可见性

- private: 只有该类的操作才能访问该类带有私有可见性的特征。
- protected:只有该类及其子类的操作才能访问该类带有保护可见性的特征。
- package: 与该类处在相同包中的或者是在嵌套子包中的任何元素能够访问该类带有包可见性的特征。
- public:能够访问该类的任何元素能够访问该类带有公共可见性的特征。

属性的名称

- 每个属性都要有一个名称以区别于类中其它的属性。
- 属性名由描述所属类特性的名词或名词短语组成。

属性的类型

- 属性具有类型，用来说明该属性是什么数据类型。
- 在UML中，类的属性可以使用任何类型，如简单类型、其它类。

属性的初始值

- 属性可以设定初始值。
- 设定初始值可以保护系统的完整性，防止漏掉取值或被非法值破坏系统的系统的完整性；也可以为用户提供易用性。

属性的多重性

- 多重性提供了一种精确的方法表达特定的与参与关系中的“事物数目”有关的业务约束。

属性的多重性

- 属性的多重性
  - 0..1　　0个或1个
  - 1　　只能1个
  - 0..*　　0个或多个
  - *　　0个或多个
  - 1..*　　1个或多个
  - 3　　只能3个
  - 0..5　　0到5个
  - 5..15　　5到15个

属性的特性

- 特性是用户对该属性性质的一个约束说明。
- 例如{只读}这样的属性说明该属性的值不能修改。
- Name:String=“COSE SEU.”{readOnly}

类属性声明的实例

- +size: Area = (100,100)
- #visibility: Boolean = false
- +defaultSize: Rectangle
- #maximumSize: Rectangle
- colors : Color [3]
- points : Point [2..* ordered]
- name : String [0..1]

属性的选择

- 类的属性应能描述特定的对象
- 只有系统感兴趣的特征才包含在类的属性中

类的操作（operation）

- 类的操作是对类的对象的动态特性的抽象，描述类的对象所能做的事。
- 操作在类图标的操作分隔框中用文字串说明。
- 操作的描述格式如下：

  [可见性] 操作名[(参数列表)] [:返回类型] [{特性}]

操作的名称

- 操作名由描述所属类的对象的行为的动词或动词短语组成。

操作的可见性

- 操作可以具有不同的可见性。
- 可见性描述了该操作对于其它类的对象是否可见以及可调用。

操作的参数表

- 参数表是一些按顺序排列的参数名称和类型，定义了操作的输入。
- 参数表是可选的。
- 参数可以具有默认值。

操作的返回类型

- 返回类型指定操作的返回值类型。
- 返回类型是可选的。

操作签名

- 操作名称、所有参数的类型以及返回类型构成了操作签名。

类操作声明的实例

- +display (): Location
- +hide ()
- #create ()
- -attachXWindow(xwin: XwindowPtr)

范围

- 目前，我们看到的对象具有自己的属性和操作，这些属性和操作一般具有实例范围。
- 然而，有时为类的所有对象定义共享的属性和操作是有用的，如对象创建操作，这些属性和操作具有类范围。
- 类范围特征为类的所有对象提供了一组全局特征。
- 类范围特征表示为加上下划线的特征说明。

属性的范围

- 实例范围：

--默认时，属性具有实例范围

--该类的每个对象获得实例范围属性的一份拷贝

--因此，每个对象都具有实例范围的不同属性值

- 类范围：

--属性可以被定义为类范围

--该类的每个对象共享相同的类范围属性

--因此，每个对象具有相同的类范围属性值

操作的范围

- 实例范围：

--默认时，操作具有实例范围

--实例范围操作的每个调用作用于该类的特定实例

--你不能调用实例范围的操作，除非你已有可用的该类实例。显然这意味着你不能使用类的实例范围操作去创建该类的第一个对象

- 类范围：

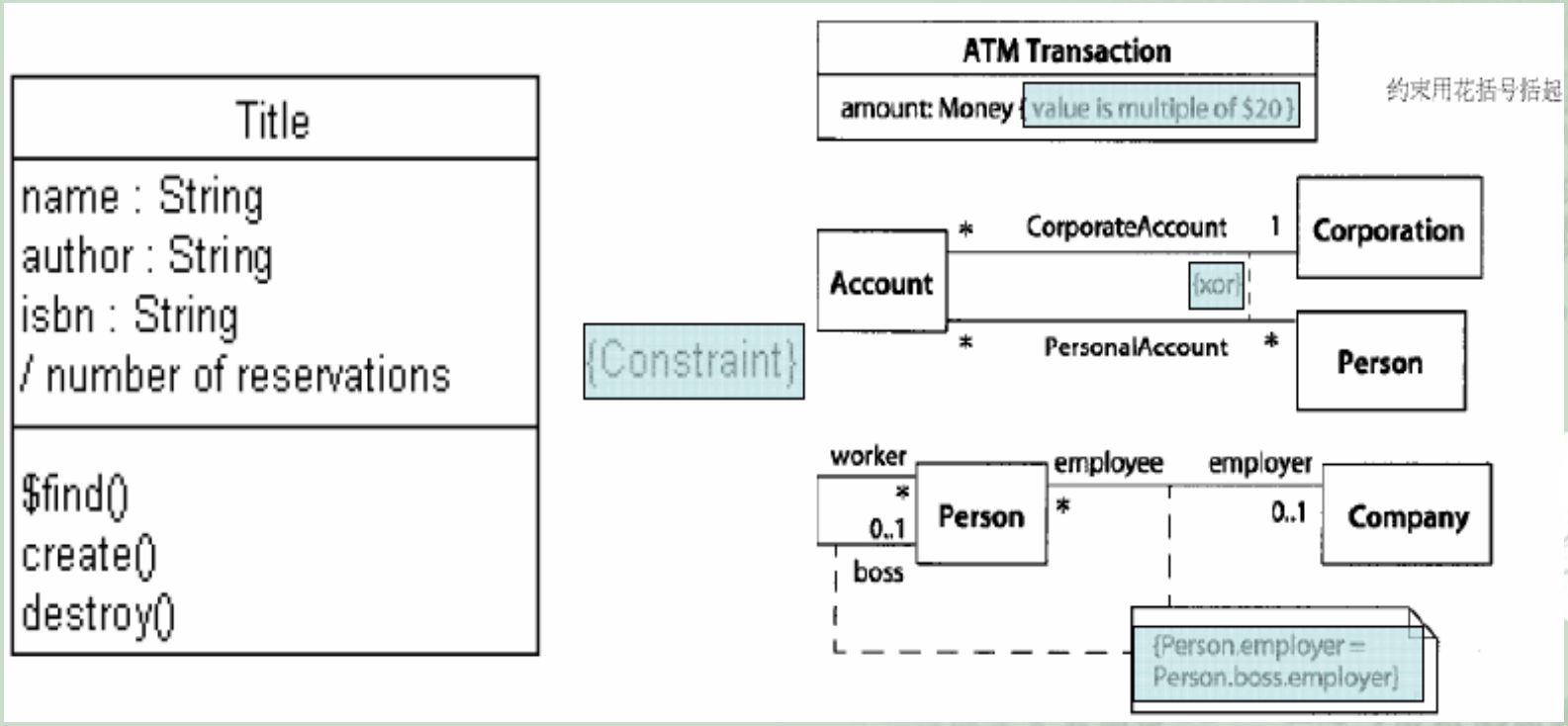
--操作可以被定义为类范围

--类范围操作的调用不作用于该类的任何特定实例，可以看作作用于该类本身。

--你能够调用类范围操作，甚至没有类的实例存在，这对于对象创建操作是理想的

类的约束

- 约束指定类所要满足的一个或者多个规则
- 在UML中，约束是用花括号扩起来的自由文本



类的一些种类

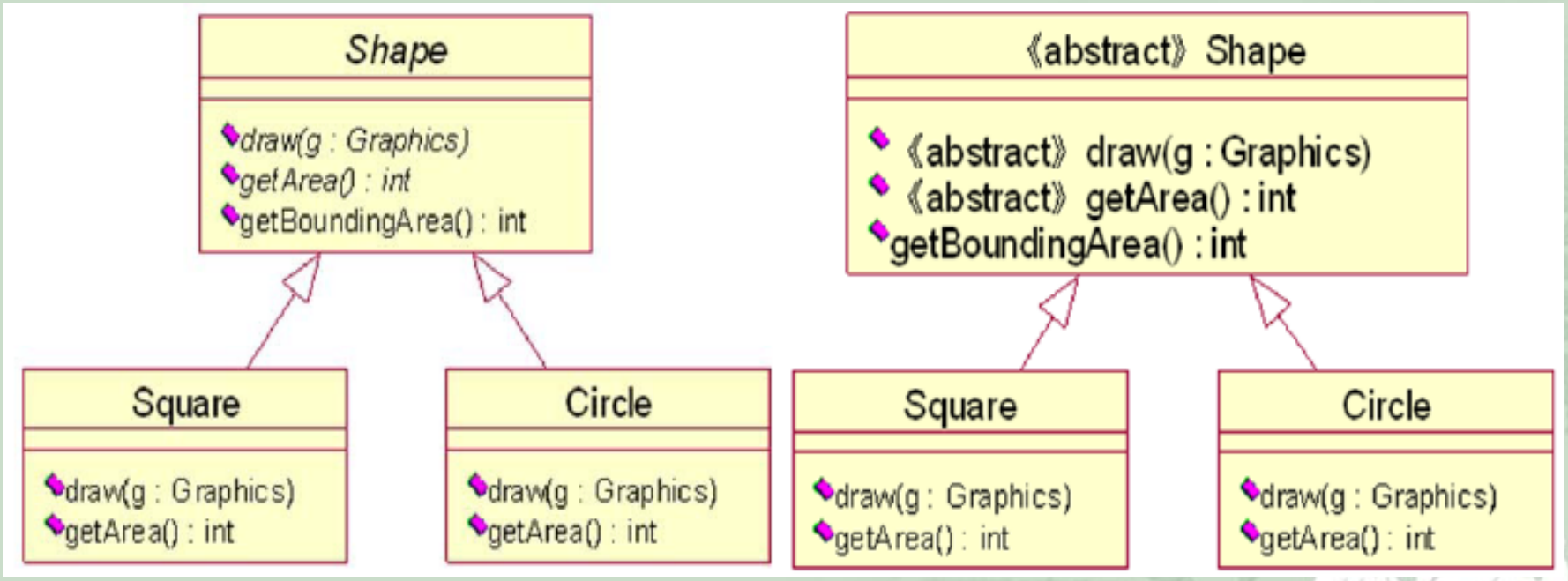
- 抽象类
- 接口
- 关联类
- 模板类
- 主动类
- 嵌套类

抽象类（abstract class）

- 缺乏实现的操作称为抽象操作，包含抽象操作的类是抽象类。
- 抽象类是不完整的，不能被直接实例化。不能创建一个属于抽象类的对象。

抽象类（abstract class）

- UML中的表示：用斜体书写或用构造型

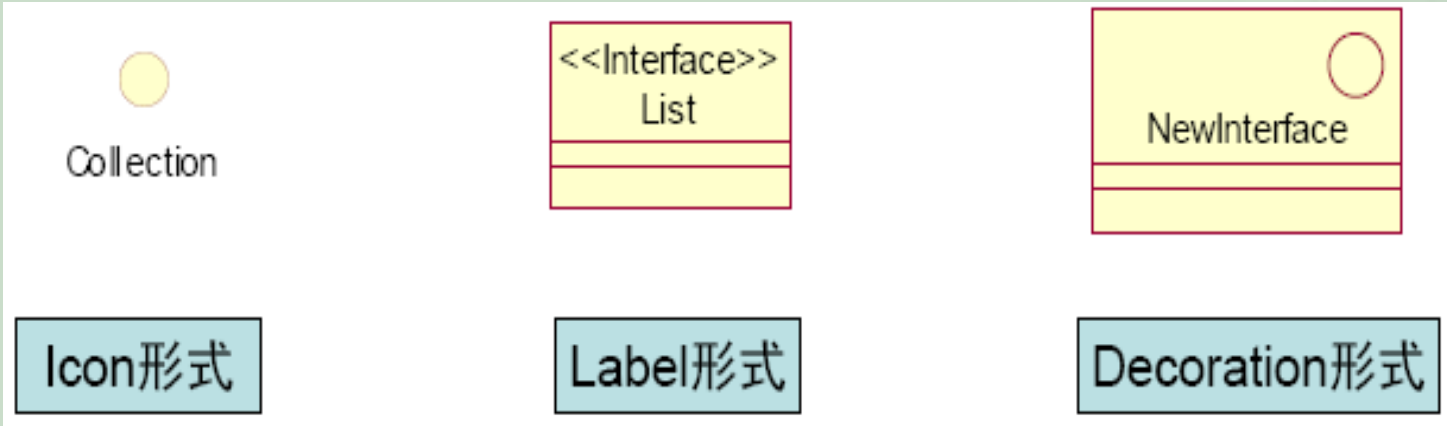


接口（interface）

- 接口是一种类似于抽象类的机制
- 接口是一个没有具体实现的类，不包含属性和操作实现
- 接口说明了操作的命名集合，是功能性规格说明
- 接口主要定义操作签名，不包含实现，这留给实现接口的类、子系统或构件。
- 使用接口是为了将规格说明和实现相分离

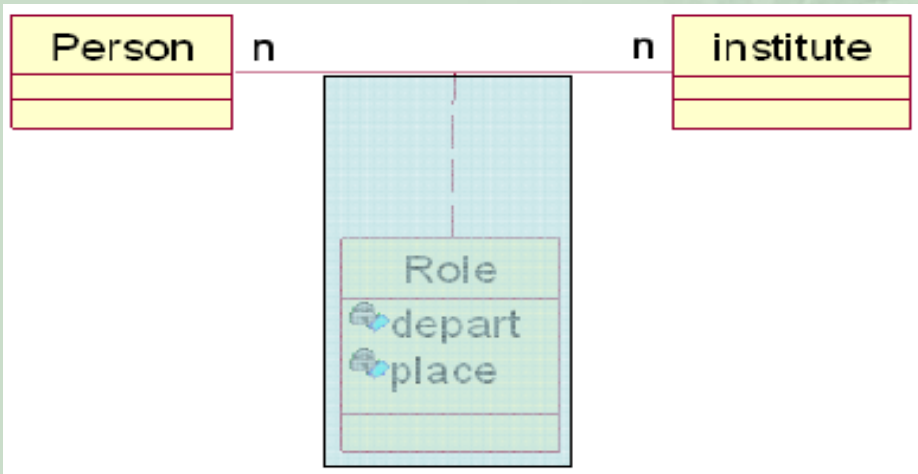
接口（interface）

- UML中的表示方法：



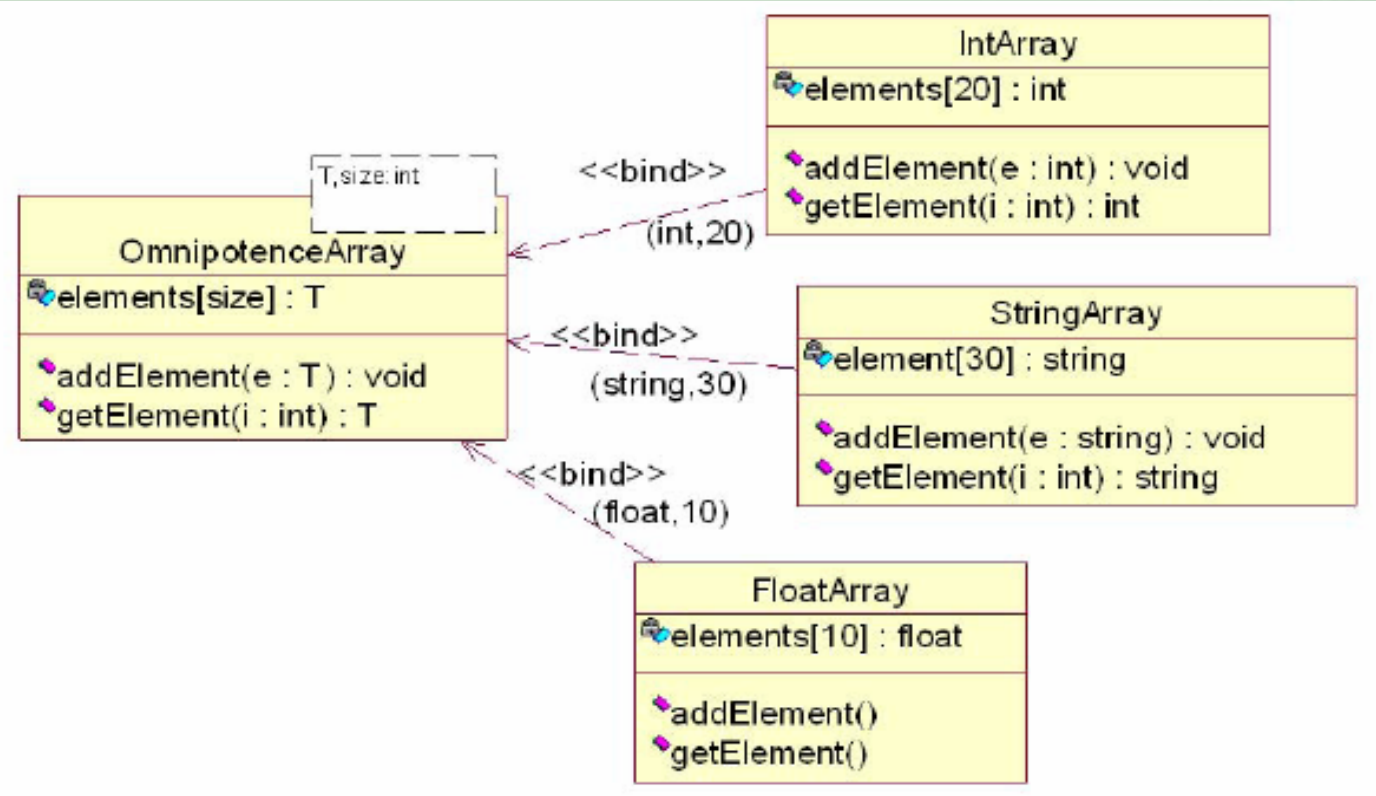
关联类（ association class）

- 面向对象建模中存在这样的问题：
  两个类之间存在多对多关系，有些属性不容易归到某一个类中。
- 关联类既是关联又是类
- 关联类表示方法：



模板类（parameterized class）

- 可以根据占位符或参数来定义类，而不用说明属性、方法返回值和方法参数的实际类型



主动类（active class）

- 一种特殊的类
- 主动类的实例称为主动对象，一个主动对象拥有一个控制线程并且能够控制线程的活动，具有独立的控制权
- 主动类的表达方式
  - 在类的两边加垂直线
  - 可以使用版型

嵌套类（nested class）

- 在一个类内部定义另一个类
- 嵌套类存在于外层类的名字空间中，因此只有外层类或外层类的对象才能访问它或它的实例
- 嵌套类的表示

关系

- 关系是事物之间的语义联系。

类之间的关系

- 泛化
- 关联
- 聚合，组合
- 实现
- 依赖

泛化（generalization）

- 泛化是一般事物和特殊事物之间的关系

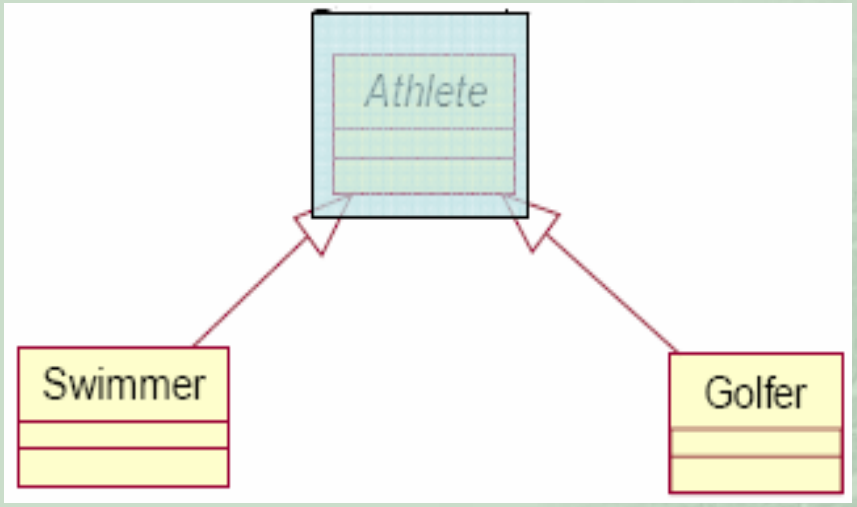
--特殊事物完全符合一般事物

--能够在一般事物出现的地方替换成特殊事物

--通过从特殊事物泛化或从一般事物特化可以创建泛化层次

泛化（generalization）

- UML中的表示方法：用一头为空心三角形的连线表示，由特殊指向一般



关联（association）

- 关联是类间的一种语义联系

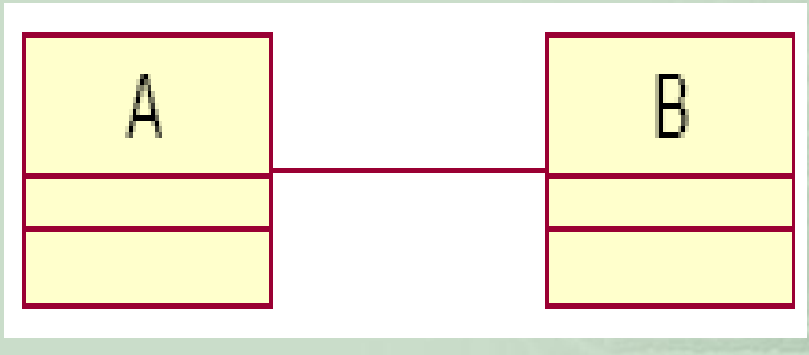
--它是对具有共同的结构特性、行为特性、关系和语义的链接(link)的描述

--如果在两个对象之间存在链接，那么这些对象的类之间必然存在关联

--链接是关联的实例，就像对象是类的实例一样

关联（association）

- UML中的表示方法：在类图中，关联用一条把类连接在一起的实线表示

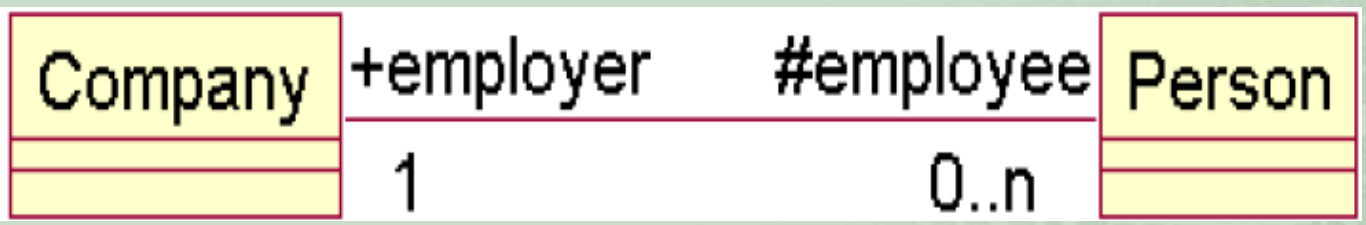


关联的名称

- 用来描述关联
- 通常是动词或者动词短语
- 命名应该有助于理解模型
- 可以前缀或后缀一个小黑箭头表明阅读的方向

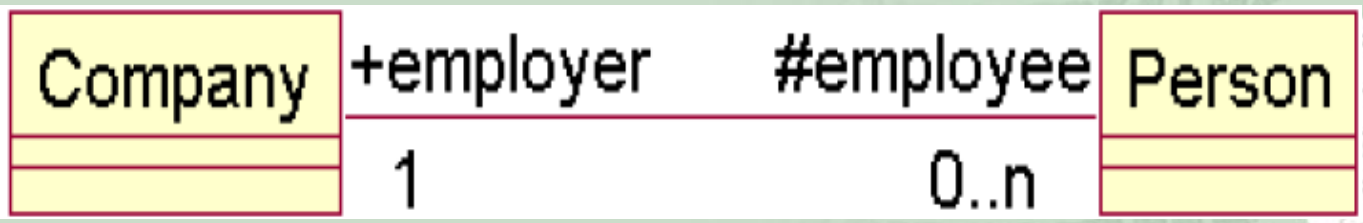
关联的角色

- 关联两端的类可以某种角色参与关联
- 应该用名词或名词短语描述角色的语义



关联的多重性

- 多重性表明在任意时刻关系所能够涉及的对象数目
- 对象可去可留，但多重性约束任意时刻对象的数目
- 没有默认的多重性：如果多重性没有显式地表示出来，那么多重性没有确定



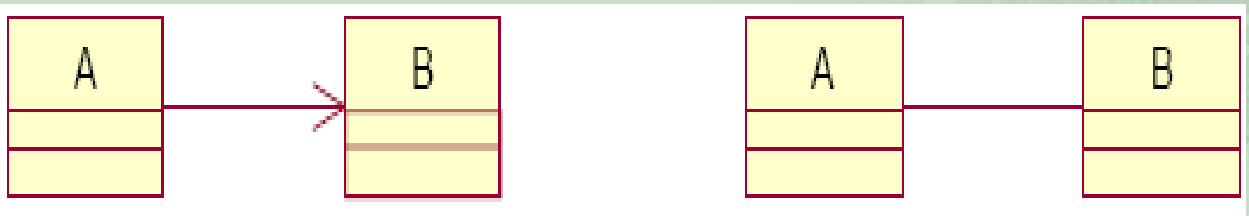
关联的多重性

在UML中，多重性可以用以下格式表示：

- 0..1
- 0..n
- 1 (1..1的简写)
- 1..n
- * (即0..n)
- 7
- 3, 6..9
- 0 (0..0的简写)（表示没有实例参与关联，一般不用）

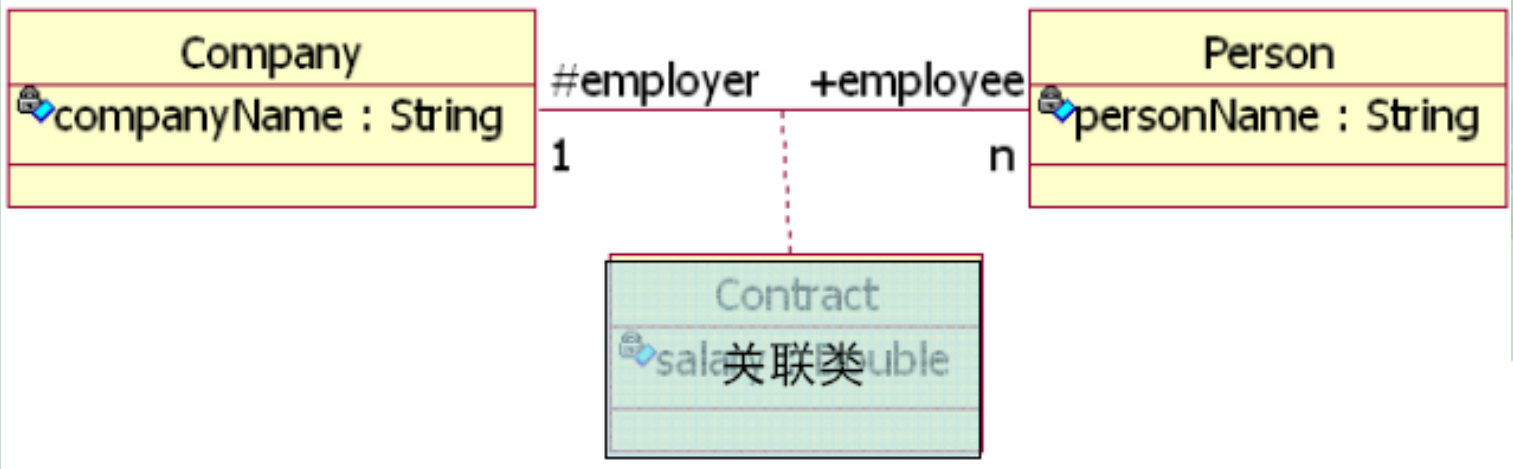
关联的导航性

- 导航性表示能够在箭头的方向上遍历关系
- 用关系端部的箭头表示，如果没有箭头，那么导航性是双向的



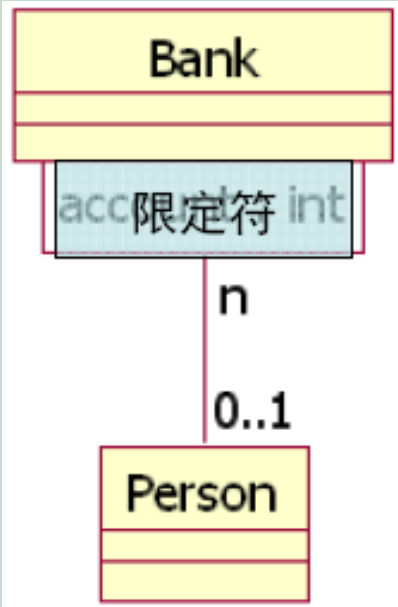
关联类

- 关联类既是关联又是类
- 通过关联类可以进一步描述关联的属性、操作，以及其它信息



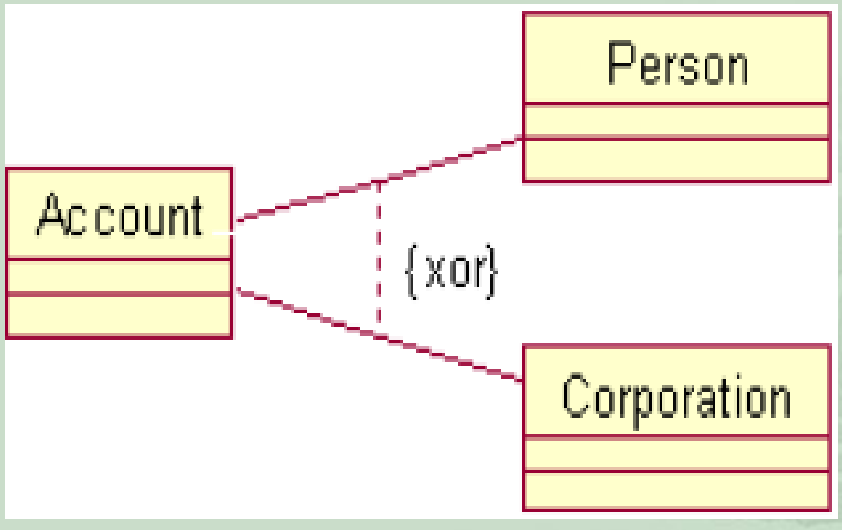
限定关联

- 带有限定符（qualifier）的关联称为限定关联（qualified association）
- 限定符的作用：给定关联一端的一个对象和限定符值，可确定另一端的一个对象或对象集



关联的约束

- 对于关联可以加上一些约束，以规定关联的含义



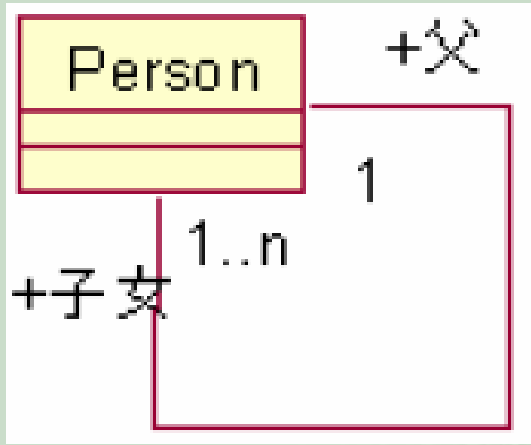
关联的种类

按照关联所连接的类的数量分：

- 自反关联
- 二元关联
- N元关联

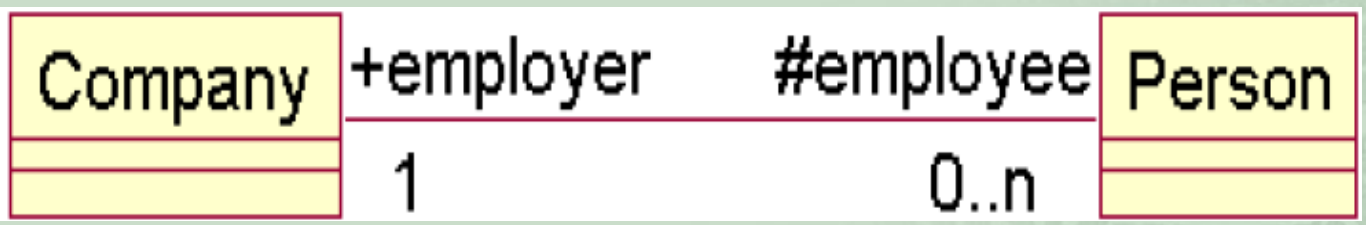
自反关联

- 是一个类与自身的关联



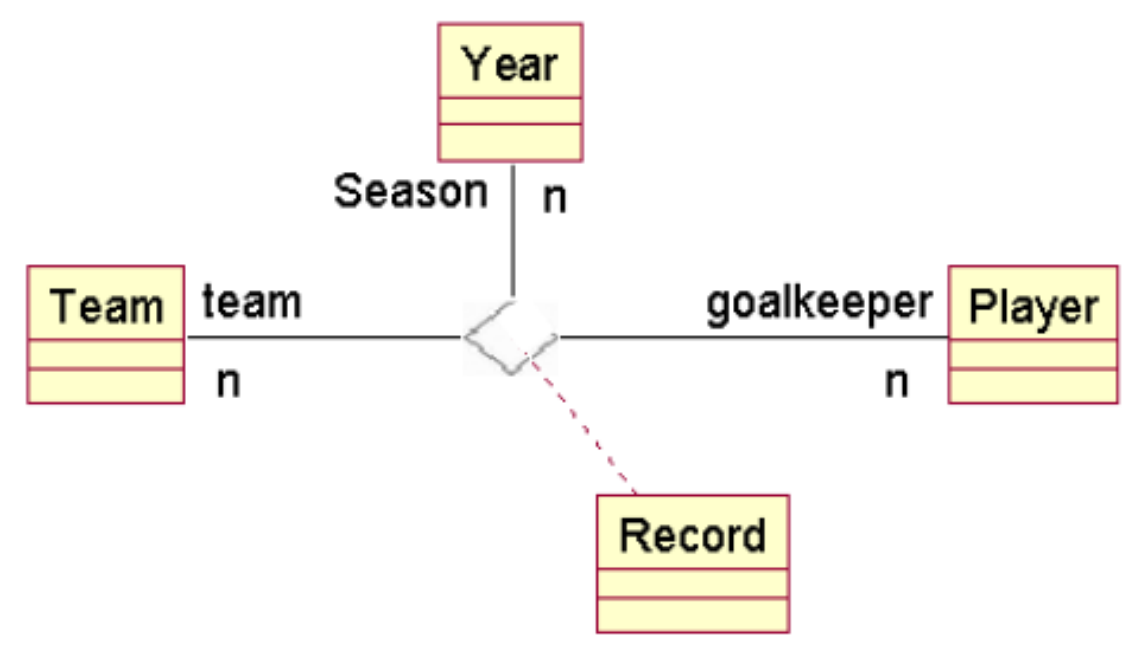
二元关联

- 两个类之间的关联



N元关联

- 是在3个或者3个以上类之间的关联
- N元关联没有限定符，聚合，组合等概念



聚合（aggregation）

- 聚合是一种特殊形式的关联，表示类之间的整体与部分的关系
- 部分可以独立于聚合而存在
- 部分可以由几个聚合来共享所有权
- 整体总是知道部分，但如果从整体到部分的关系是单向的，那部分就不知道整体

聚合

- 聚合就像计算机和它的外围设备：

--计算机只是很松散地和外部设备关联

--外围设备可有可无

--外围设备可以被其它计算机所共享

--外围设备不被任何特定计算机“拥有”

聚合（aggregation）

- UML中的表示：空心菱形+实线

组合（composition）

- 组合是一种强形式的聚合
- 部分每次只能属于一个整体
- 整体唯一地负责处理它的部分，包括创建和销毁
- 如果整体销毁，必须将它所有的部分销毁或把处理权交给其它对象
- 组合关系中的整体与部分一般具有同样的生存期

组合（composition）

- 组合就像是树和树叶：

--树叶只能被一棵树所拥有

--树叶不能由几棵树所共享

--当树死去时，它的树叶也随之死去

组合（composition）

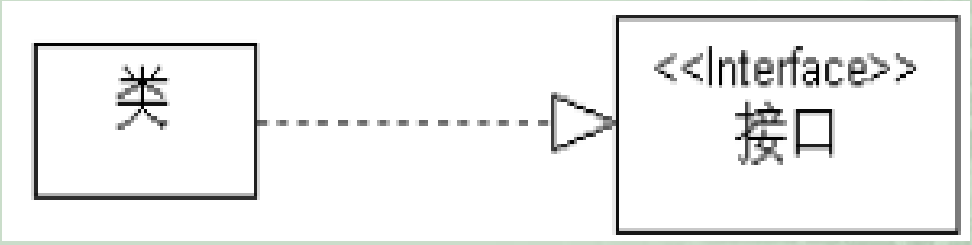
- 组合关系的表示：用实心菱形+实线

聚合和组合的区别

- 聚合表示较弱的整体/部分关系，组合表示较强的整体/部分关系
- 聚合关系中，代表部分事物的对象可以属于多个聚合对象，具有共享性，而组合关系中，代表部分事物的对象不可以属于多个聚合对象，不具有共享性
- 聚合关系中，代表整体事物的对象与代表部分事物的对象，有各自独立的生存期，而组合关系中，有完全一致的生存期

实现

- 类与被类实现的接口
- 协作与被协作实现的用例
- 使用带空心三角形箭头的虚线表示



依赖

- 依赖表示两个或多个模型元素之间语义上的关系，对一个元素的改变可能影响其它元素。
- 在类中，依赖由各种原因引起，如：
  - 一个类向另一个类发消息
  - 一个类是另一个类的数据成员类型
  - 一个类是另一个类的某个操作参数类型
  - ……

依赖

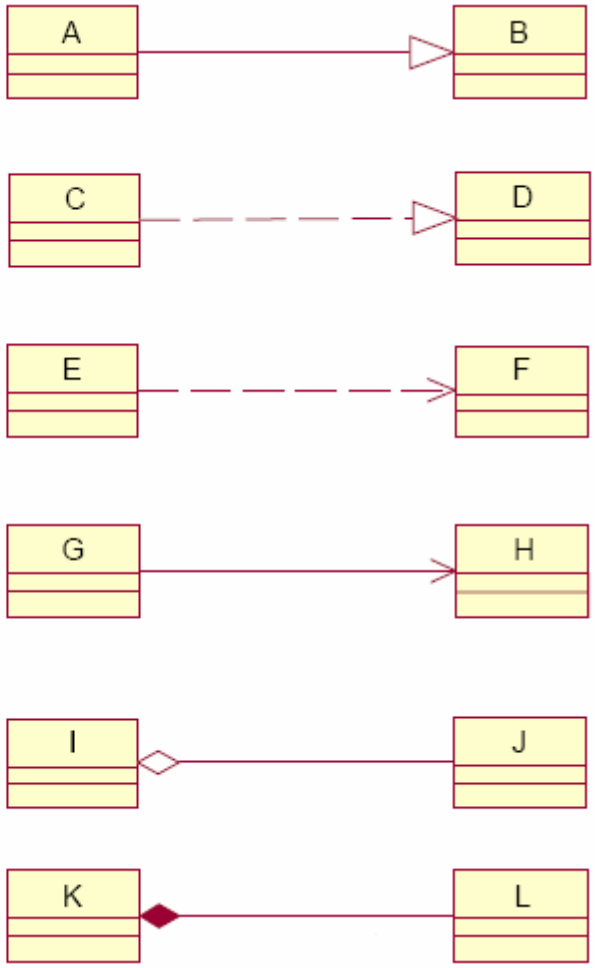
- 依赖关系的表示：虚线+箭头

类之间关系小结

- 泛化和实现
- 较为清晰
- 关联、聚合、组合、依赖
- 在代码层面不能完全区分
- 强弱关系：组合>聚合>关联>依赖

类之间各种关系的表示小结

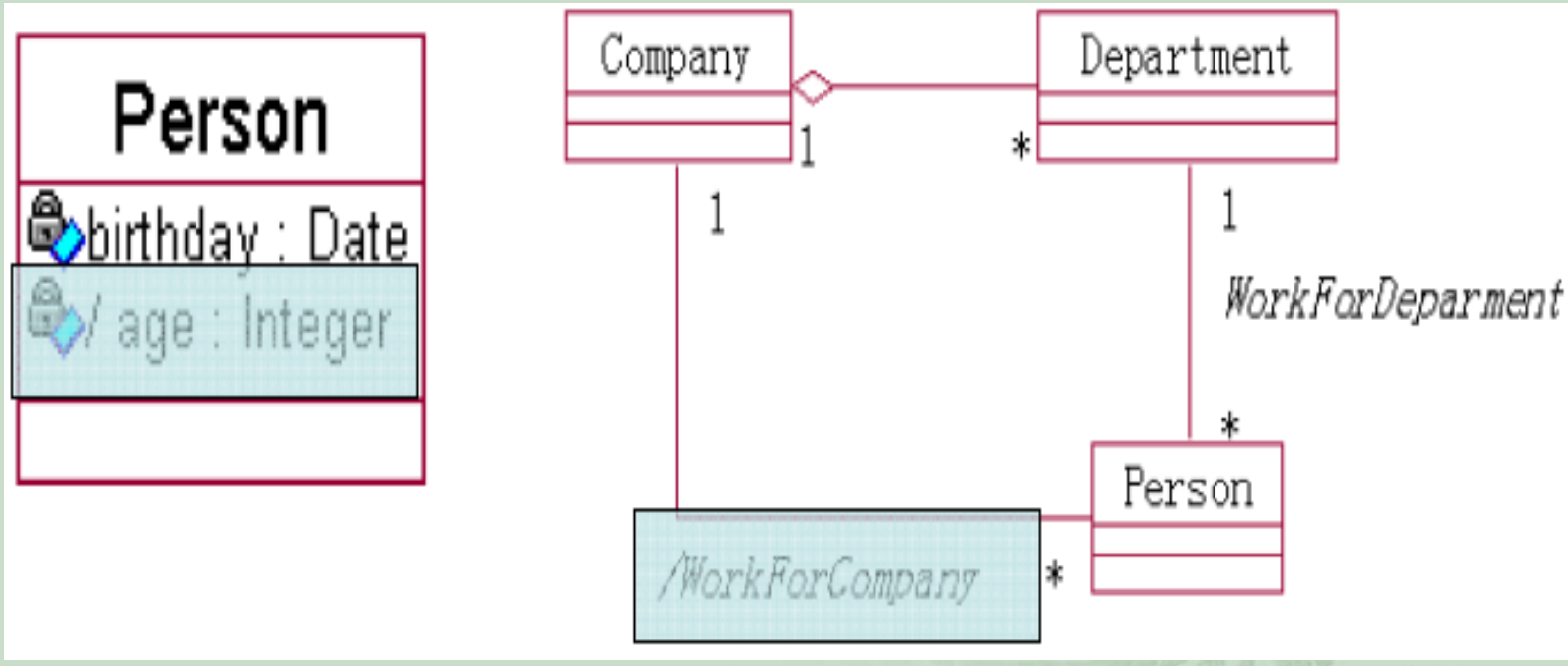
- 泛化
- 实现
- 依赖
- 关联
- 聚合
- 组合



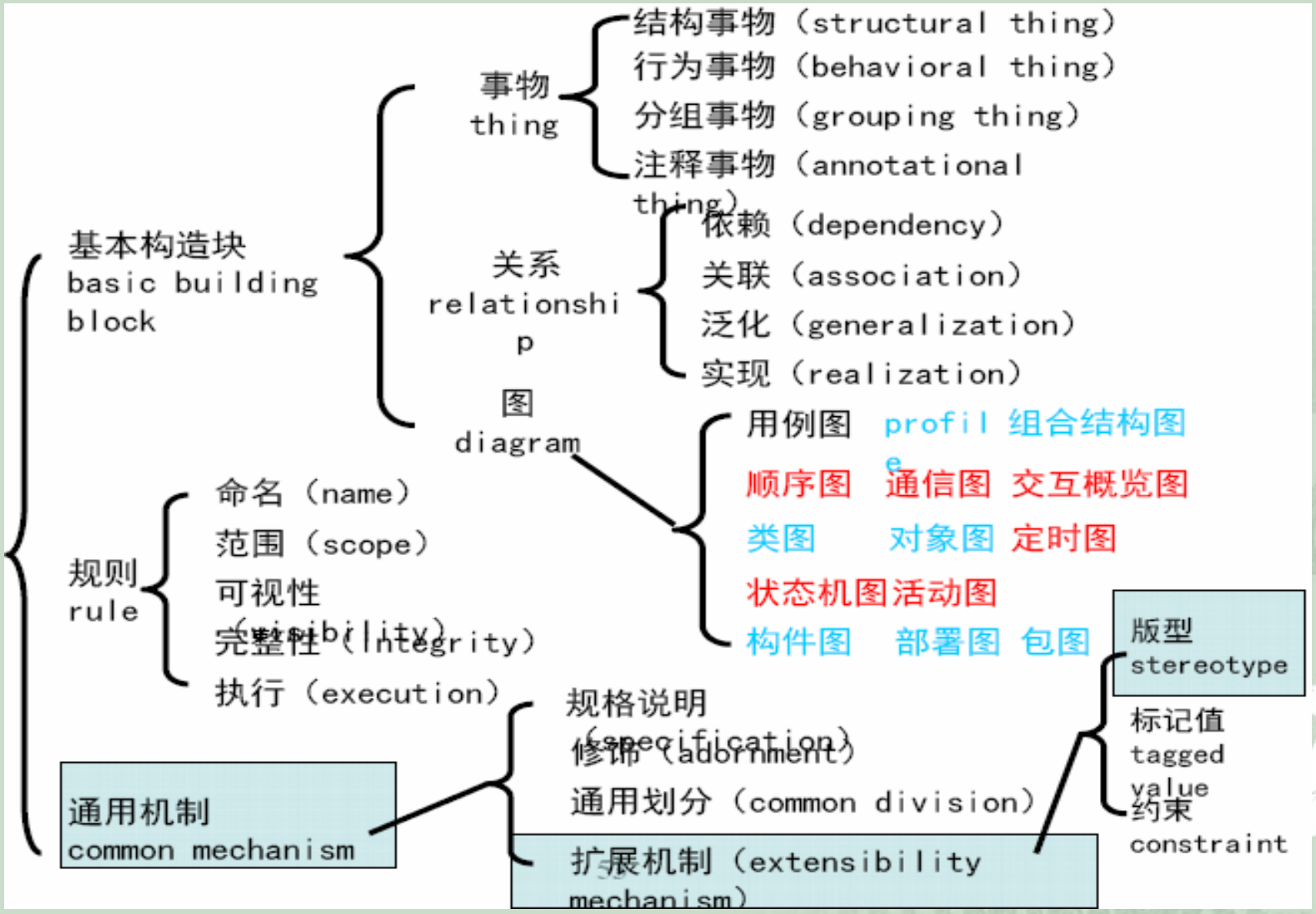
派生属性和派生关联

- 派生属性(derived attribute)和派生关联(derived association)：可以从其它属性和关联计算推演得到的属性和关联
- 在UML中表示方法：在派生属性和派生关联的名字前加斜杠“/”

派生属性和派生关联的示例和说明



版型


基本构造块
basic building block
事物
thing
结构事物（structural thing）
行为事物（behavioral thing）
分组事物（grouping thing）
注释事物（annotational thing）
关系
relationship
依赖（dependency）
关联（association）
泛化（generalization）
实现（realization）
图
diagram
用例图 profile 组合结构图
顺序图 通信图 交互概览图
类图 对象图 定时图
状态机图 活动图
构件图 部署图 包图
规则
rule
命名（name）
范围（scope）
可视性（visibility）
完整性（integrity）
执行（execution）
通用机制
common mechanism
规格说明（specification）
修饰（adornment）
通用划分（common division）
扩展机制（extensibility mechanism）
版型
stereotype
标记值
tagged value
约束
constraint

版型的相关说明

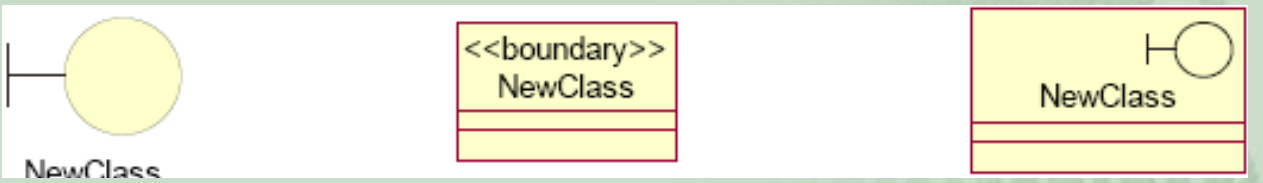
- 版型可以应用于所有的模型元素
- 版型示例：

三种主要的类版型

- 边界类（boundary class）
- 实体类（entity class）
- 控制类（control class）

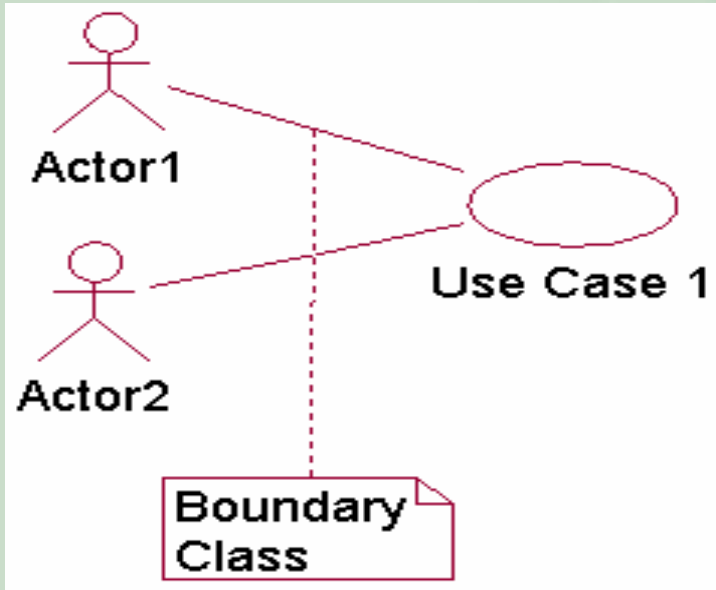
边界类（boundary class）

- 边界类位于系统与外界的交界处，如：
  - 窗体类
  - 报表类
  - 描述通信协议的类
  - 直接与外设交互的类
  - 直接与外部系统交互的类……
- 边界类的表示：



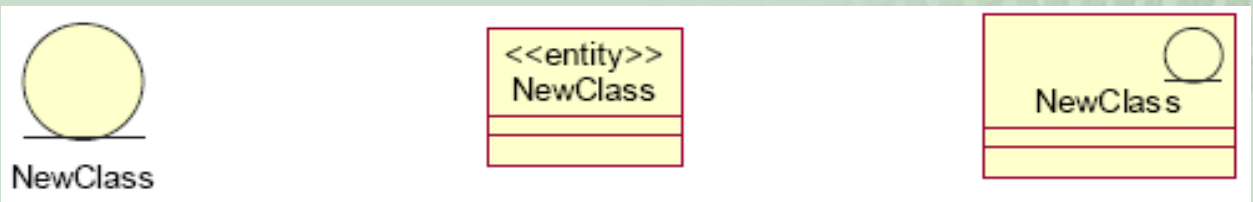
边界类的说明

- 可以通过用例图确定需要的边界类
- 每个参与者-用例对至少要有一个边界类，但并非每个参与者-用例对要生成唯一边界类，多个参与者可以用同一个边界类



实体类（entity class）

- 实体类描述要保存到持久存储体中的信息，如：
  - 数据库
  - 各种形式的数据文件中的信息……
- UML中实体类的表示：

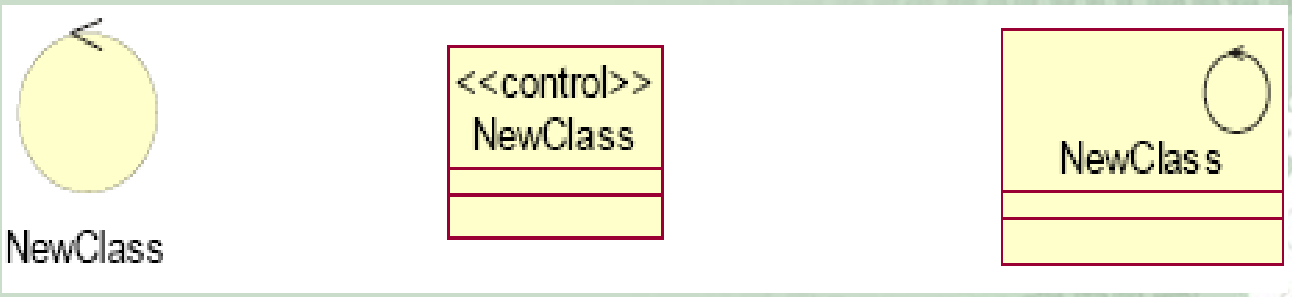


实体类的说明

- 实体类可以通过事件流和交互图发现
- 实体类通常用领域术语命名
- 通常，每个实体类在数据库中有相应的表，实体类中的属性对应数据库中表的字段

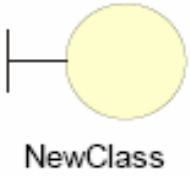
控制类（control class）

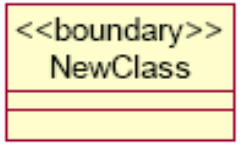
- 控制类是主要负责控制其它类工作的类，如：
  - 主程序类
  - 主窗体类.....
- UML中控制类的表示：

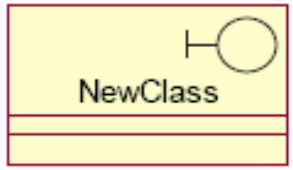


边界类、实体类、控制类

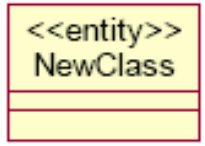
- 边界类

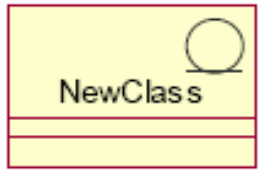




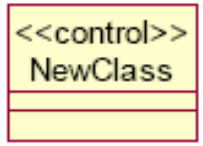


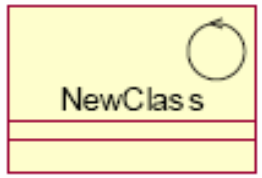
- 实体类





- 控制类





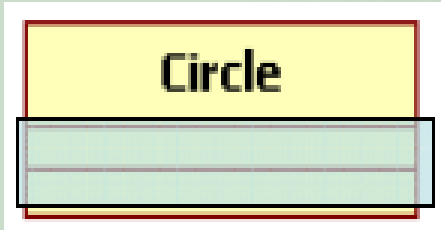
类图的抽象层次

在软件开发的不同阶段使用的类图具有不同的抽象层次：

- 概念层（conceptual）类图
- 说明层（specification）类图
- 实现层（implementation）类图

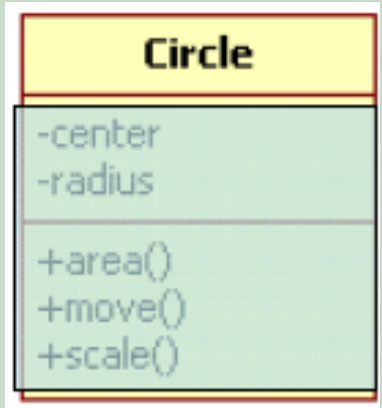
概念层类图

- 描述应用领域中的概念：一般这些概念与类有很自然的联系，但未必有直接的映射关系
- 独立于程序设计语言，类的描述可能没有或有少量属性和操作名称：



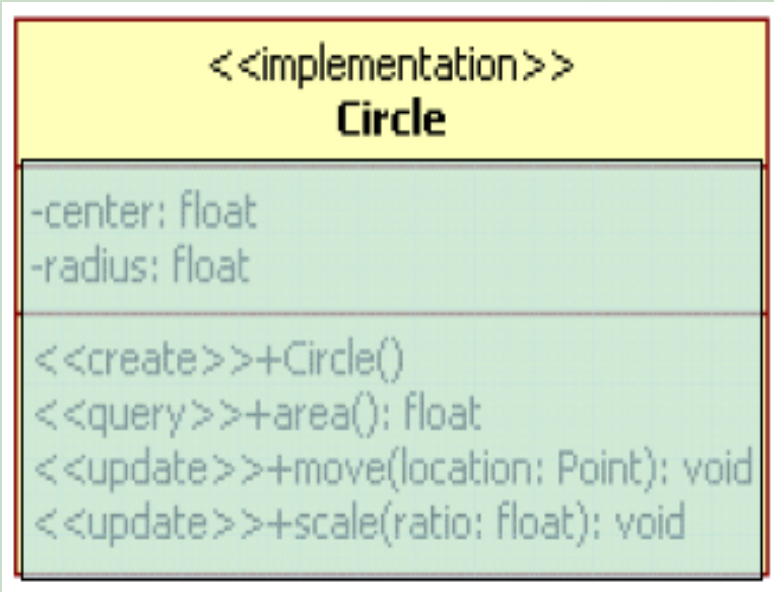
说明层类图

- 描述软件的接口部分，不是实现部分：接口可能因为实现环境等不同而有不同的实现
- 主要列出属性和操作名称

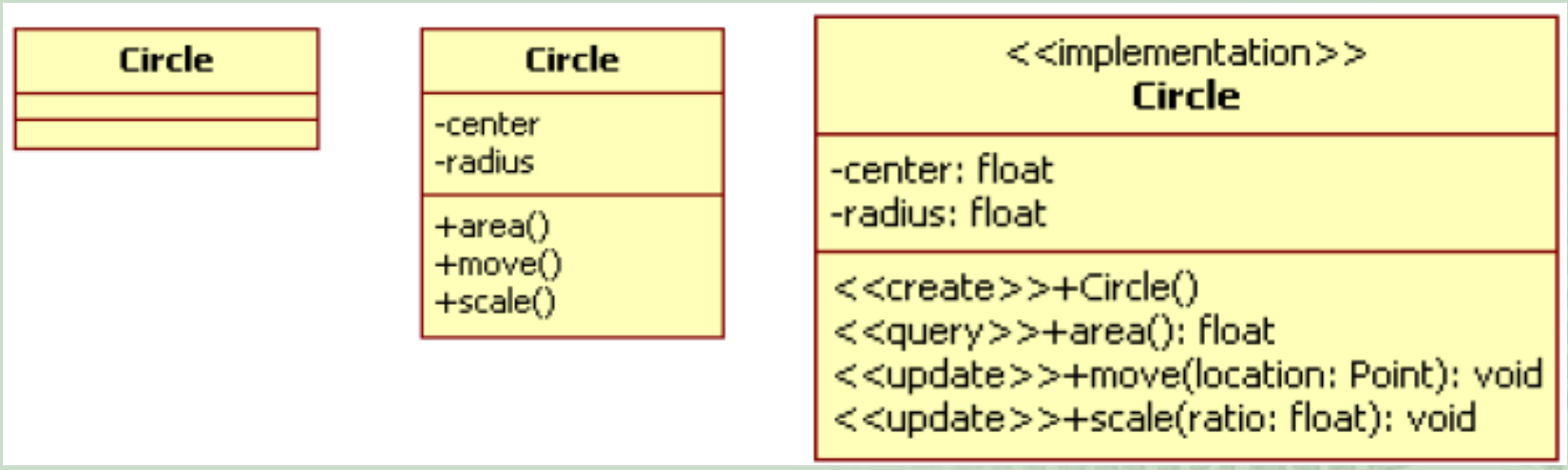


实现层类图

- 考虑类的实现问题
- 提供类的实现细节



3个层次类图对比


Circle
Circle
-center
-radius
+area()
+move()
+scale()
<<implementation>>
Circle
-center: float
-radius: float
<<create>>+Circle()
<<query>>+area(): float
<<update>>+move(location: Point): void
<<update>>+scale(ratio: float): void

类图的抽象层次的说明

- 类图的三个层次之间没有一个清晰的界限
- 三个层次的观点并不是UML的组成部分，但它们对于画类图或者阅读类图非常有用
- 指明类图层次的方法
- 版型：

《implementation class》

内容

- 类图概述
- 类图基本概念
- 类图建模方法

寻找分析类的方法

- 不存在找出恰当分析类的简单算法。
- 存在导致良好结果的经验技术。

寻找分析类的方法

- 名词/动词分析
- CRC分析
- 考虑类的其它来源

名词/动词分析

- 名词/动词分析是分析文本尝试找出类、属性和职责的非常简单的方法
- 基本上，文本中的名词和名词短语暗示类或类的属性，动词和动词短语暗示职责或类的操作。
- 合适的信息来源是：用例、补充规约、术语表和任何其它信息源
- 需要与领域专家交流
- 需要仔细分析，不能机械应用

用例中的名词

Main Success Scenario (or Basic Flow):

1. **Customer** arrives at a **POS checkout** with **goods** and/or **services** to purchase.
2. **Cashier** starts a new **sale.**
3. **Cashier** enters **item identifier.**
4. System records **sale line item** and presents **item description, price,** and running **total.** Price calculated from a set of price rules.

Cashier repeats steps 2-3 until indicates done.

5. System presents total with **taxes** calculated.
6. Cashier tells Customer the total, and asks for **payment.**
7. Customer pays and System handles payment.
8. System logs the completed **sale** and sends sale and payment information to the external **Accounting** (for accounting and **commissions)** and **Inventory** systems (to update inventory).
9. System presents **receipt.**
10. Customer leaves with receipt and goods (if any).

Extensions (or Alternative Flows):

7a. Paying by cash:

1. Cashier enters the cash **amount tendered.**
2. System presents the **balance due,** and releases the **cash drawer.**
3. Cashier deposits cash tendered and returns balance in cash to Customer.
4. System records the cash payment.

CRC分析

- CRC分析是有力和有趣的头脑风暴技术

--把问题域中重要事物书写在便笺上

--每个便笺具有三个分栏：

  --类：包含该类的名称

  --职责：包含该类的职责列表

  --协作方：包含与该类协作的类列表

--过程：头脑风暴

  --要求团队成员命名问题域的事物，并写在便笺上

  --要求团队成员阐述事物的职责，并写在便笺上

  --要求团队成员识别可能一起工作的类，并写在便笺上

考虑类的其它来源

- 存在很多其它潜在的、可供考虑的分析类来源。可在真实世界中寻找分析类，例如物理客体、文档记录、到外部世界的接口以及概念实体。

分析类分类表

Conceptual Class Category	Examples
physical or tangible objects	*Register* *Airplane*
specifications, designs, or descriptions of things	*ProductSpecification* *FlightDescription*
places	*Store* *Airport*
transactions	*Sale, Payment* *Reservation*
transaction line items	*SalesLineItem*
roles of people	*Cashier* *Pilot*
containers of other things	*Store, Bin* *Airplane*
things in a container	*Item* *Passenger*
other computer or electro-mechanical systems external to the system	*CreditPaymentAuthorizationSystem* *AirTrafficControl*
abstract noun concepts	*Hunger* *Acrophobia*

分析类分类表

organizations	*SalesDepartment* *ObjectAirline*
events	*Sale, Payment, Meeting* *Flight, Crash, Landing*
processes (often *not* represented as a concept, but may be)	*SellingAProduct* *BookingASeat*
rules and policies	*RefundPolicy* *CancellationPolicy*
catalogs	*ProductCatalog* *PartsCatalog*
records of finance, work, contracts, legal matters	*Receipt, Ledger, EmploymentContract* *MaintenanceLog*
financial instruments and services	*LineOfCredit* *Stock*
manuals, documents, reference papers, books	*DailyPriceChangeList* *RepairManual*

POS候选分析类

- Register (cat list)
- Item (noun)
- Store (cat list)
- Sale (noun)
- Payment (noun)
- ProductCatalog(cat list)
- ProdSpecification(cat list)
- SalesLineItem(noun)
- Cashier
- Customer
- Manager

设计类

- 设计类有下列来源：

--问题域：

--分析类的精化

--一个分析类可变为零个、一个或是多个设计类

--解域：

--实用类库、中间件、GUI库、重用构件

--设计模式

--特定实现的细节

--……

关联的获取

- 名词/动词分析
- CRC分析
- 考虑关联的其它来源

关联的分类表

Category	Examples
A is a physical part of B	*Drawer — Register (or more specifically, a POST)* *Wing — Airplane*
A is a logical part of B	*SalesLineItem — Sale* *FlightLeg—FlightRoute*
A is physically contained in/on B	*Register — Store, Item — Shelf* *Passenger — Airplane*
A is logically contained in B	*ItemDescription — Catalog* *Flight— FlightSchedule*
A is a description for B	*ItemDescription — Item* *FlightDescription — Flight*
A is a line item of a transaction or report B	*SalesLineItem — Sale* *Maintenance Job — Maintenance-Log*
A is known/logged/recorded/reported/captured in B	*Sale — Register* *Reservation — FlightManifest*

关联的分类表

A is a member of B	*Cashier — Store* *Pilot — Airline*
A is an organizational subunit of B	*Department — Store* *Maintenance — Airline*
A uses or manages B	*Cashier — Register* *Pilot — Airplane*
A communicates with B	*Customer — Cashier* *Reservation Agent — Passenger*
A is related to a transaction B	*Customer — Payment* *Passenger — Ticket*
A is a transaction related to another transaction B	*Payment — Sale* *Reservation — Cancellation*
A is next to B	*SalesLineItem — SalesLineItem* *City— City*
A is owned by B	*Register — Store* *Plane — Airline*
A is an event related to B	*Sale — Customer; Sale — Store* *Departure — Flight*

关联的获取

分析模型中应该包含的有用的高优先级的一些关联：

- A是 B 物理上或逻辑上的组成部分.
- A 物理上或逻辑上包含于B.
- A在B中记录.

POS的候选关联

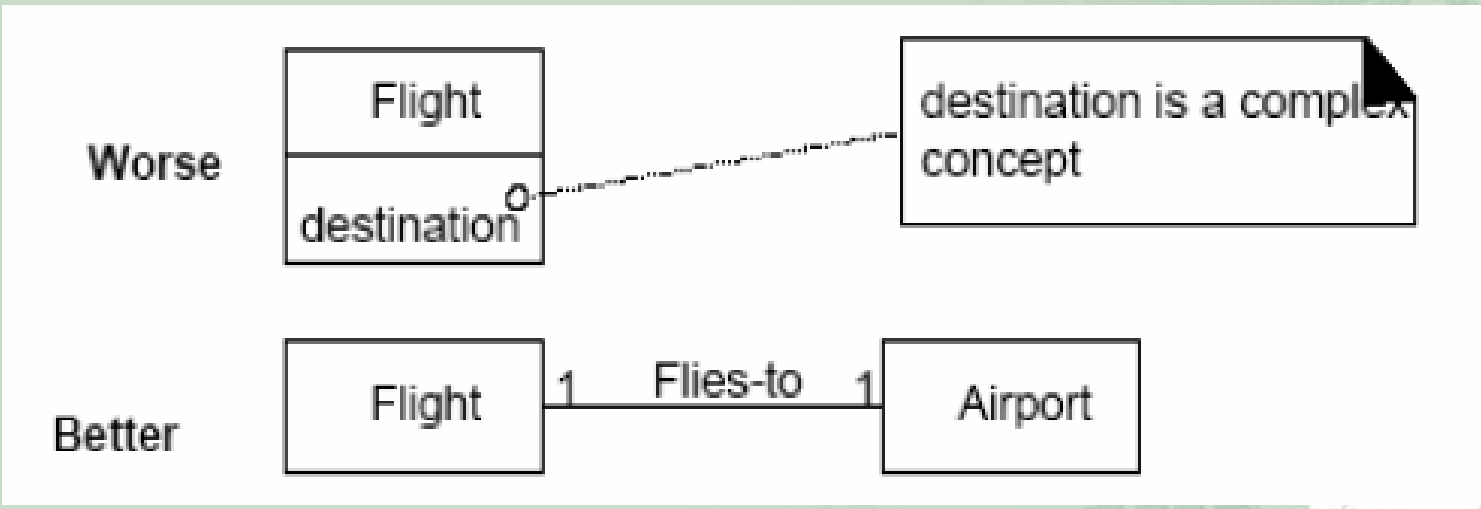
- **A** is subpart/member of **B**. (SaleLineItem-Sale)
- **A** uses or manages **B**. (Cashier –Register)
- **A** is transaction related to **B**. (Payment -Sale)
- **A** is next to **B**. (SaleLineItem-SaleLineItem)
- **A** is an event related to **B**. (Sale-Store)
- ……

属性的获取

- 属性的来源与类相似

属性还是类

- 如果一个元素似乎没有什么特别重要的信息或者具有很少令人感兴趣的行为，那么倾向于建模为属性；反之，则倾向于建模为关联的类



操作的获取

- 在确定了需求和产生领域模型之后,为软件类添加方法,并且定义对象间的消息传递,从而实现需求.
- 决定哪些方法位于哪些地方以及对象间如何交互是非常重要的.

操作的获取

- 当交互图创建时, 实际上已经为对象分配了职责,反映到交互图就是该对象发送相应的消息到不同类的对象.
- 类图的方法栏表示了职责分配的结果,职责具体由这些方法来实现.

操作的获取

- 对象职责的分配：创造性工作，与创建交互图实现用例并行

  --用例中的动词或动词短语

  --CRC分析

  --GRASP

  --……

操作的获取

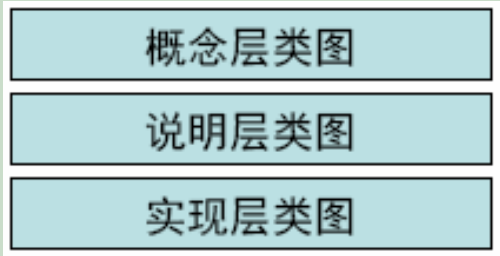
- GRASP (**G**eneral **R**esponsibility **A**ssignment **S**oftware **P**atterns)
  --Information Expert(信息专家)
  --Creator(创建者)
  --Low Coupling(低耦合)
  --High Cohesion(高内聚)
  --Controller(控制器)
  --Polymorphism(多态)
  --Indirection(中介)
  --Pure Fabrication(纯虚构)
  --Protected Variations(受保护变化)

类图建模

- 研究分析问题域，确定系统需求
- 确定类，明确类的含义和职责，确定属性和操作
- 确定类之间的关系
- 细化类和类之间的关系，解决命名冲突、功能重复等问题
- 绘制类图，并给元素必要的文字说明

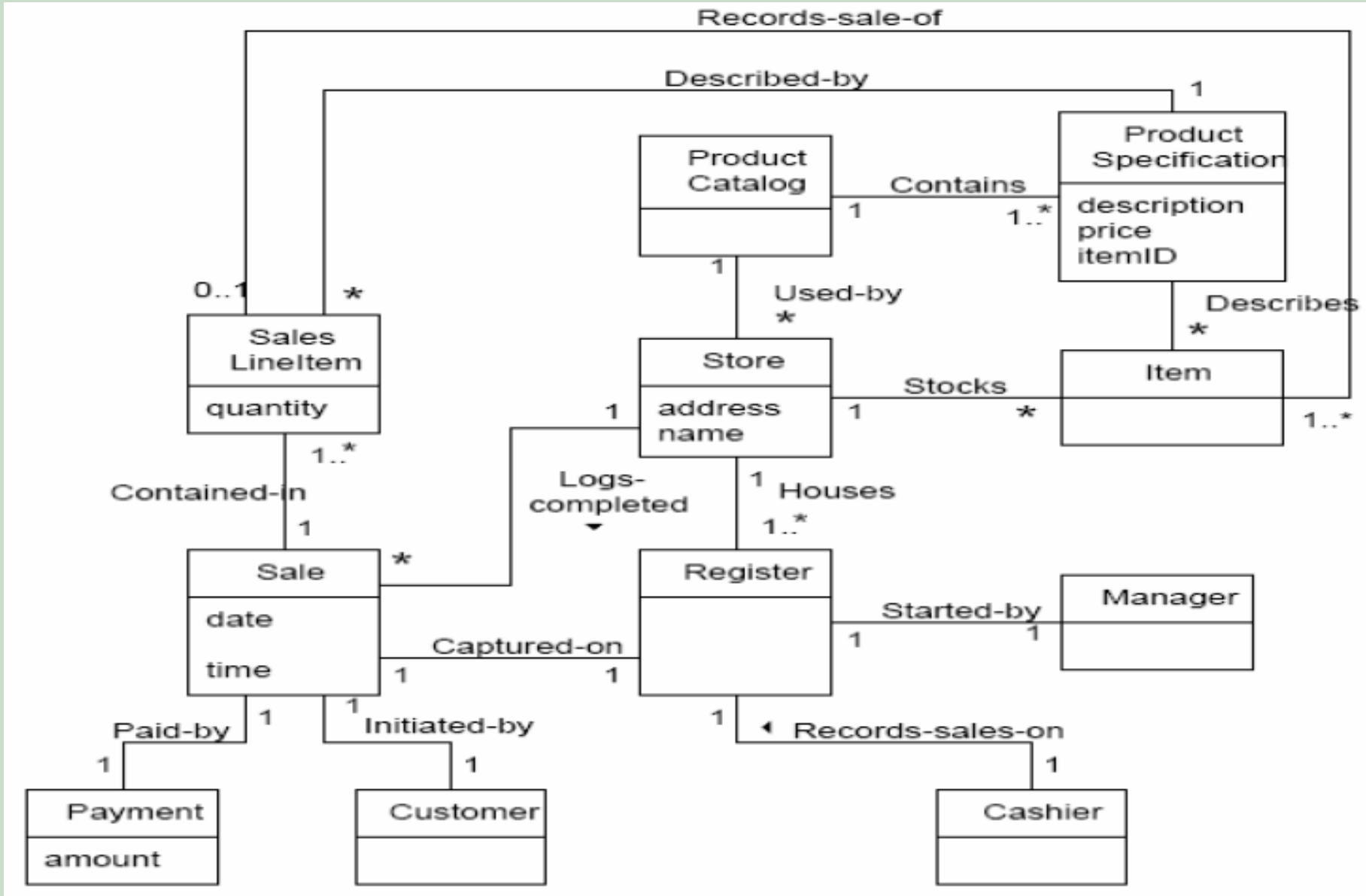
类图建模时的几个建议

- 不要试图使用所有的符号
  - 真正需要时才使用
- 不要过早陷入细节
  - 分析阶段
  - 设计阶段
  - 实现阶段

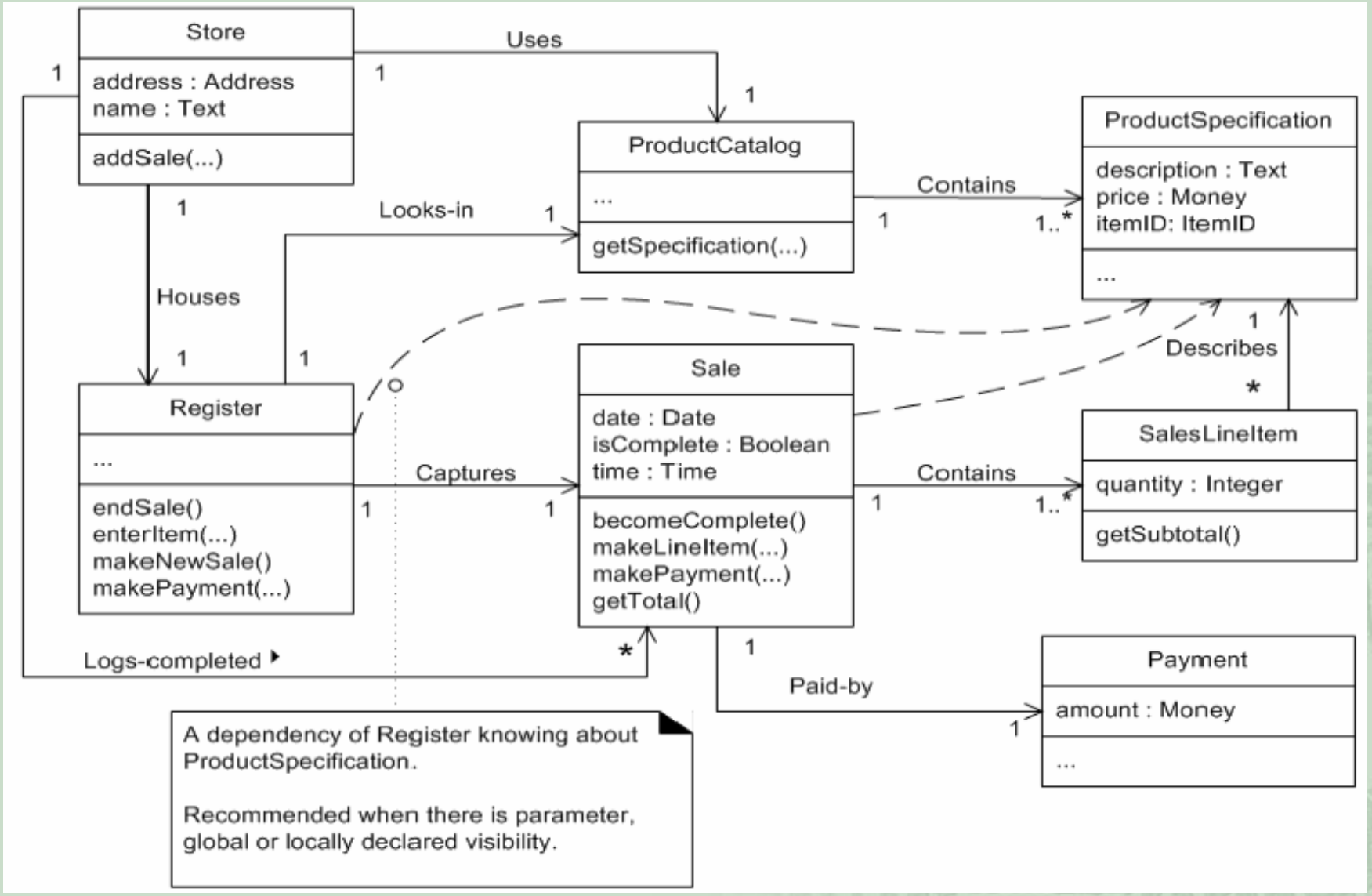


- 类图构造完成之后，应考虑
  - 模型是否反映问题域的真实情况
  - 模型及其中建模元素的职责是否明确
  - 模型中元素的粒度是否合适

POS分析类图局部


Records-sale-of
Described-by
1
Product Specification
description
price
itemID
Product Catalog
Contains
1
1..*
1
Used-by
*
Describes
*
0..1
*
Sales LineItem
quantity
1..*
Store
address
name
1
Stocks
1
*
Item
1..*
Contained-in
1
Logs-completed
1
Houses
1..*
Sale
*
date
time
Register
Started-by
1
1
Manager
Captured-on
1
1
1
Paid-by
1
1
Initiated-by
1
1
Records-sales-on
1
Payment
amount
Customer
Cashier

POS设计类图局部


Store
address : Address
name : Text
addSale(...)
Uses
ProductCatalog
...
getSpecification(...)
ProductSpecification
description : Text
price : Money
itemID: ItemID
...
Contains
Looks-in
Houses
Register
...
endSale()
enterItem(...)
makeNewSale()
makePayment(...)
Captures
Sale
date : Date
isComplete : Boolean
time : Time
becomeComplete()
makeLineItem(...)
makePayment(...)
getTotal()
Describes
SalesLineItem
quantity : Integer
getSubtotal()
Contains
Logs-completed
Paid-by
Payment
amount : Money
...
A dependency of Register knowing about ProductSpecification.
Recommended when there is parameter, global or locally declared visibility.

类图与领域分析（domain analysis）

- 建立类图的过程就是对领域及其解决方案的分析和设计过程
- 在获取类的时候有时候必须与领域专家合作
  - 对研究领域进行仔细分析，抽象出领域中的概念，定义其含义及相互关系，分析出系统类，并用领域中的术语为类命名

领域分析

- 领域分析
  - 是通过对某一领域的已有应用系统、理论、技术、开发历史等的研究，来标识、收集、组织、分析和表示领域模型及软件体系结构的过程
  - 是指特定应用领域中公共需求的标识、分析和规约，即发现或创建那些可广泛应用的类，其目的使它们在应用域中多个项目间能被复用
  - 是软件工程师了解背景信息的过程
- 没有坚实的领域分析，任何重大的软件项目都不应当进行

分析类与分析模型

- 分析类表示问题域中简洁、良好定义的抽象。

--问题域是产生软件系统需求的域。

--分析类应该以清晰的无歧义的方式映射到真实世界业务概念。

--在分析中常常需要将混淆或不恰当的业务概念澄清为能够形成分析类基础的事物。

--分析类捕捉抽象的本质，忽略实现细节。

- 分析模型只包含分析类，不包含从设计角度产生的类。

良好的分析类

- 名称反映目的
- 简洁抽象建模问题域的一个特定元素
- 清晰地映射到问题域中的可识别的特征
- 具有小的、良好定义的职责集合
- 高内聚
- 低耦合

分析类的经验法则

- 每个类支持适量的职责
- 不存在独立的类
- 当心很多非常小的类
- 当心“伪类”
- 当心万能类
- 避免深度继承树

面向对象设计的原则

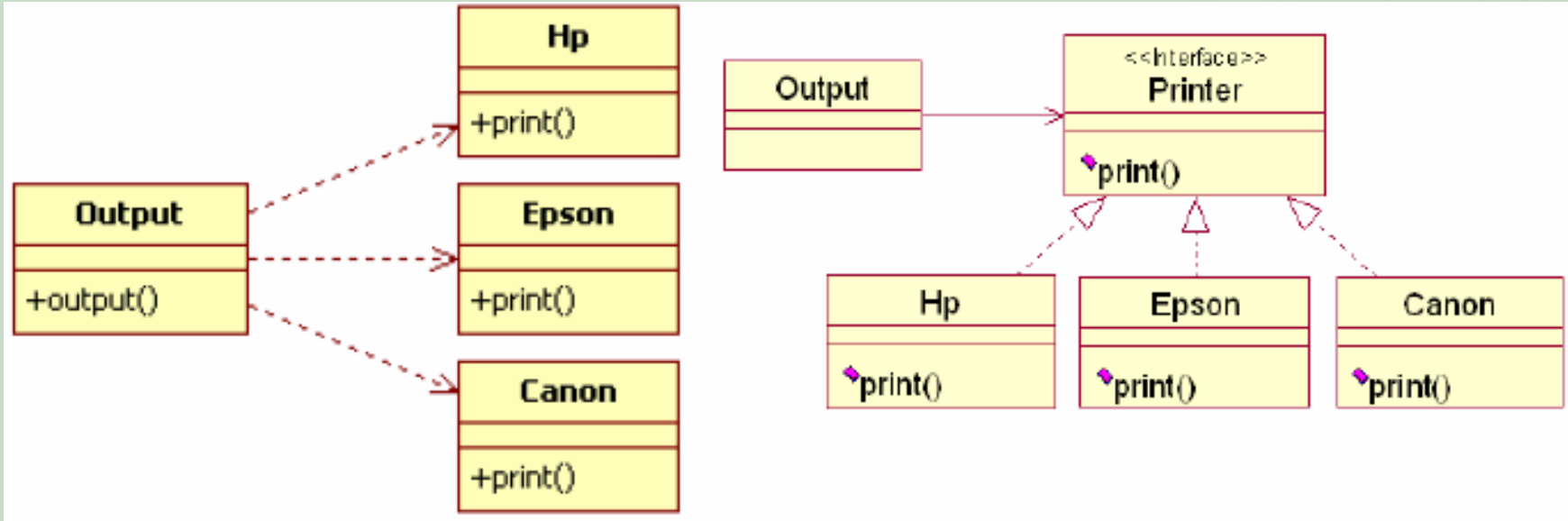
- 开闭原则：Open/Closed Principle, OCP
- Liskov替换原则：Liskov Substitution Principle, LSP
- 依赖倒置原则：Dependency Inversion Principle, DSP
- 接口分离原则：Interface Segregation Principle, ISP

开闭原则

- Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification. (Bertrand Meyer)
- 软件实体在扩展性方面应该是开放的，而在更改性方面应该是封闭的
- 为了满足开闭原则，设计时应尽量使用接口进行封闭, 采用抽象机制, 并利用OO中的多态技术

开闭原则示例

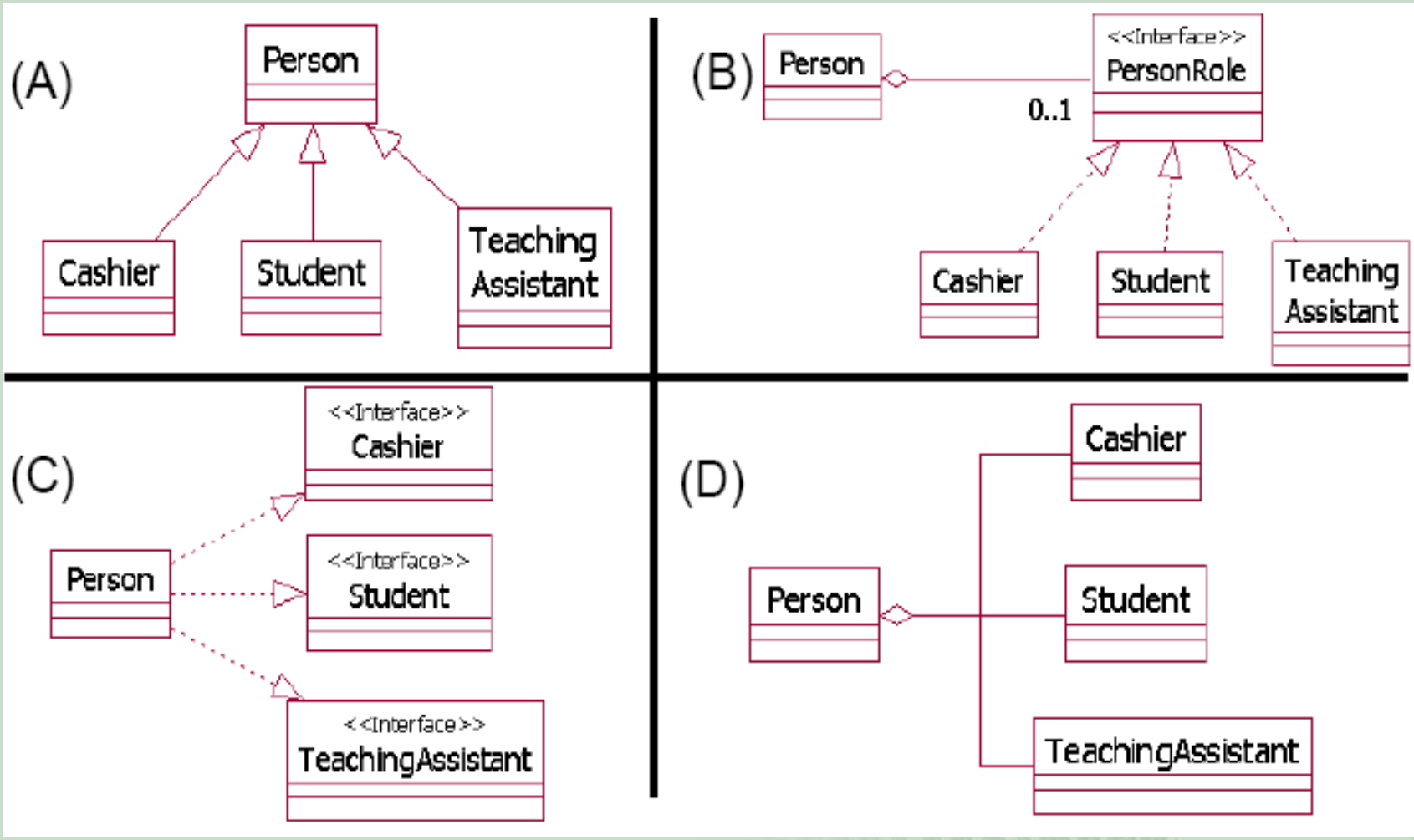
- 打印输出设计



开闭原则示例

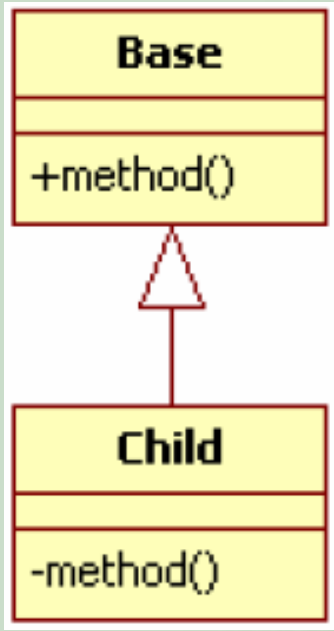
- “一个研究生在软件学院做助教（teaching assistant），同时也在校园的餐厅打工做收银员（cashier），也就是说，这个研究生有三种角色：学生，助教和收银员，但在同一时刻只能有一种角色”根据上面的陈述，哪种设计是最合理的？

开闭原则示例


(A)
Person
Cashier
Student
Teaching Assistant
(B)
Person
0..1
<<Interface>>
PersonRole
Cashier
Student
Teaching Assistant
(C)
<<Interface>>
Cashier
Person
<<Interface>>
Student
<<Interface>>
TeachingAssistant
(D)
Cashier
Person
Student
TeachingAssistant

Liskov替换原则

- Subtypes must be substitutable for their base types. (Liskov,1987)
- 子类可以替换父类出现在父类能出现的任何地方

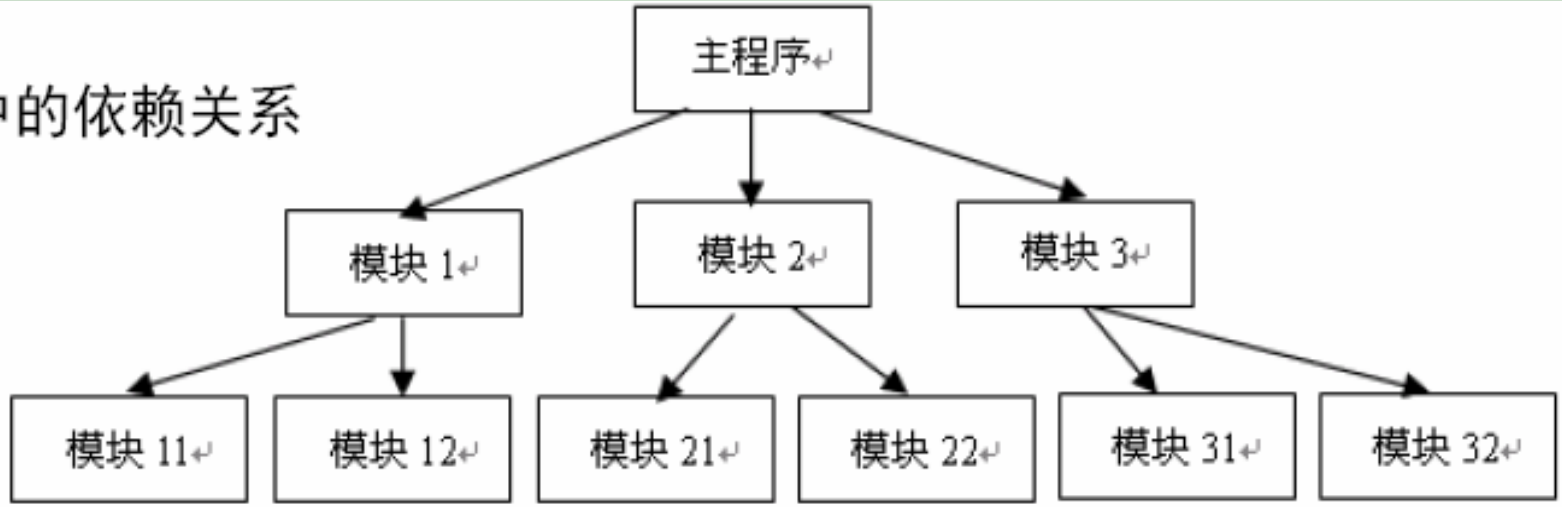


依赖倒置原则

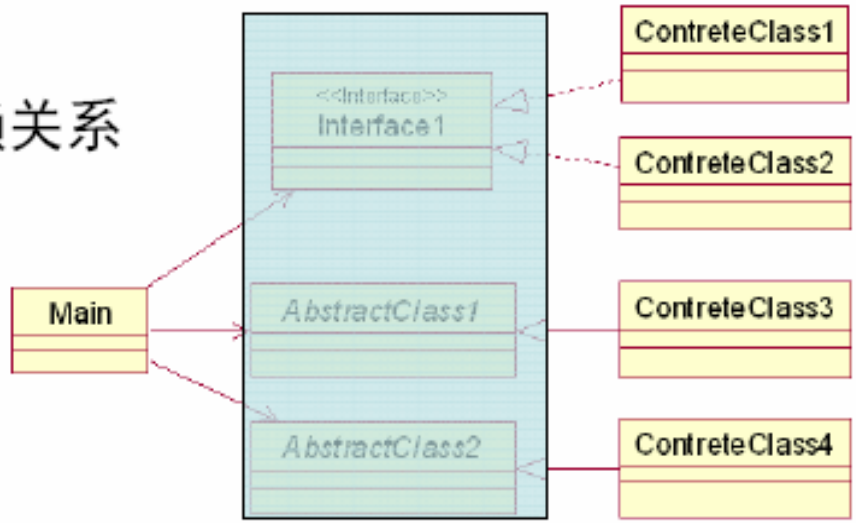
- 依赖关系应该是尽量依赖接口(或抽象类)，而不是依赖于具体类
- 抽象不应该依赖于细节, 细节应该依赖于抽象
- 要针对接口编程, 不要针对实现编程

不同设计方式依赖关系的比较

SD中的依赖关系

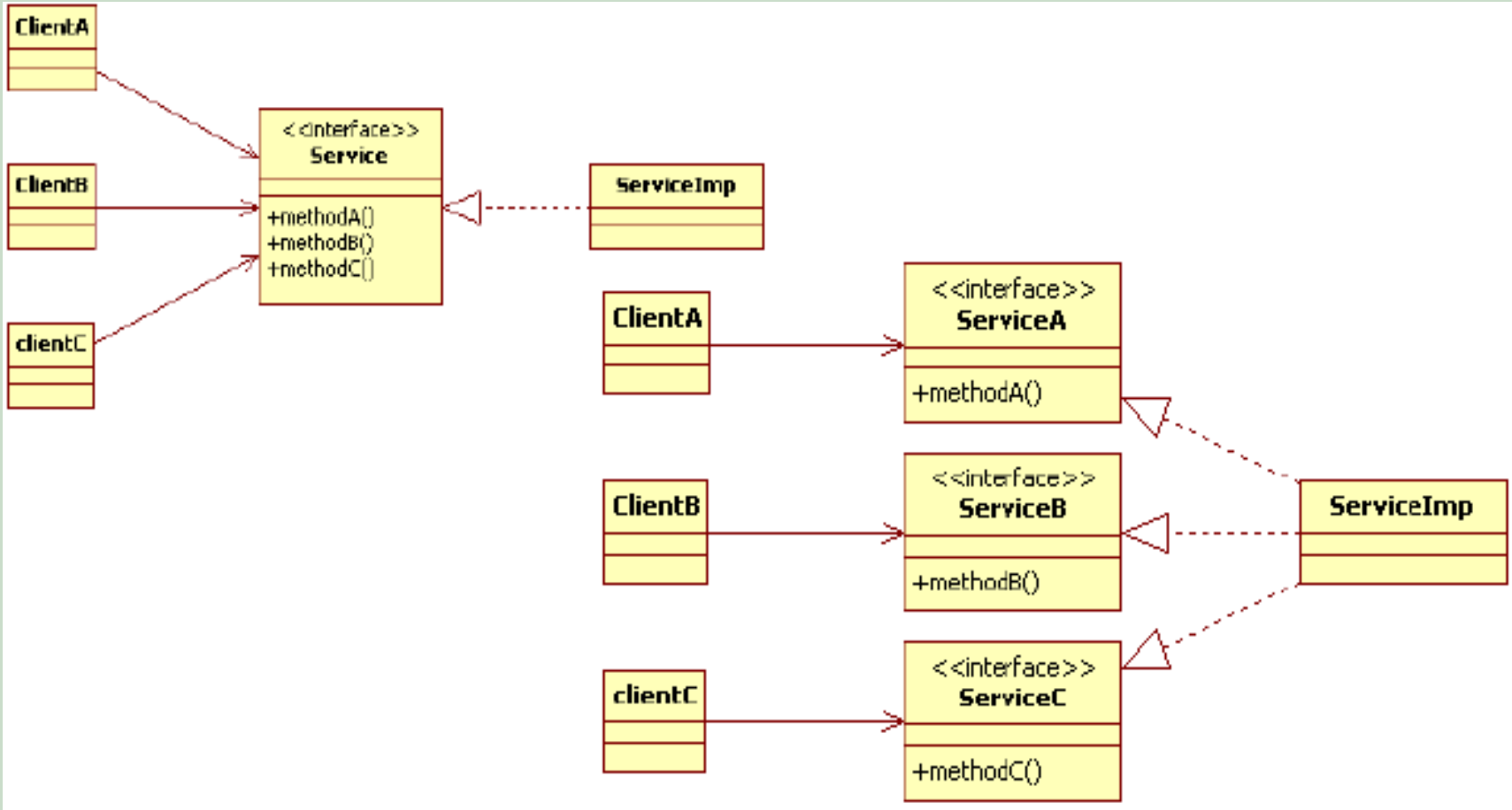


OOD中的依赖关系



接口分离原则

- 使用多个专门的接口比使用单一的总接口要好



类图建模实例-需求描述

- 小王是一个爱书之人，家里各类书籍已过千册，而平时又时常有朋友外借，因此需要一个个人图书管理系统。该系统应该能够将书籍的基本信息按计算机类、非计算机类分别建档，实现按书名、作者、类别、出版社等关键字的组合查询功能。在使用该系统录入新书籍时系统会自动按规则生成书号，可以修改信息，但一经创建就不允许删除。该系统还应该能够对书籍的外借情况进行记录，可对外借情况列表打印。另外，还希望能够对书籍的购买金额、册数按特定时间周期进行统计

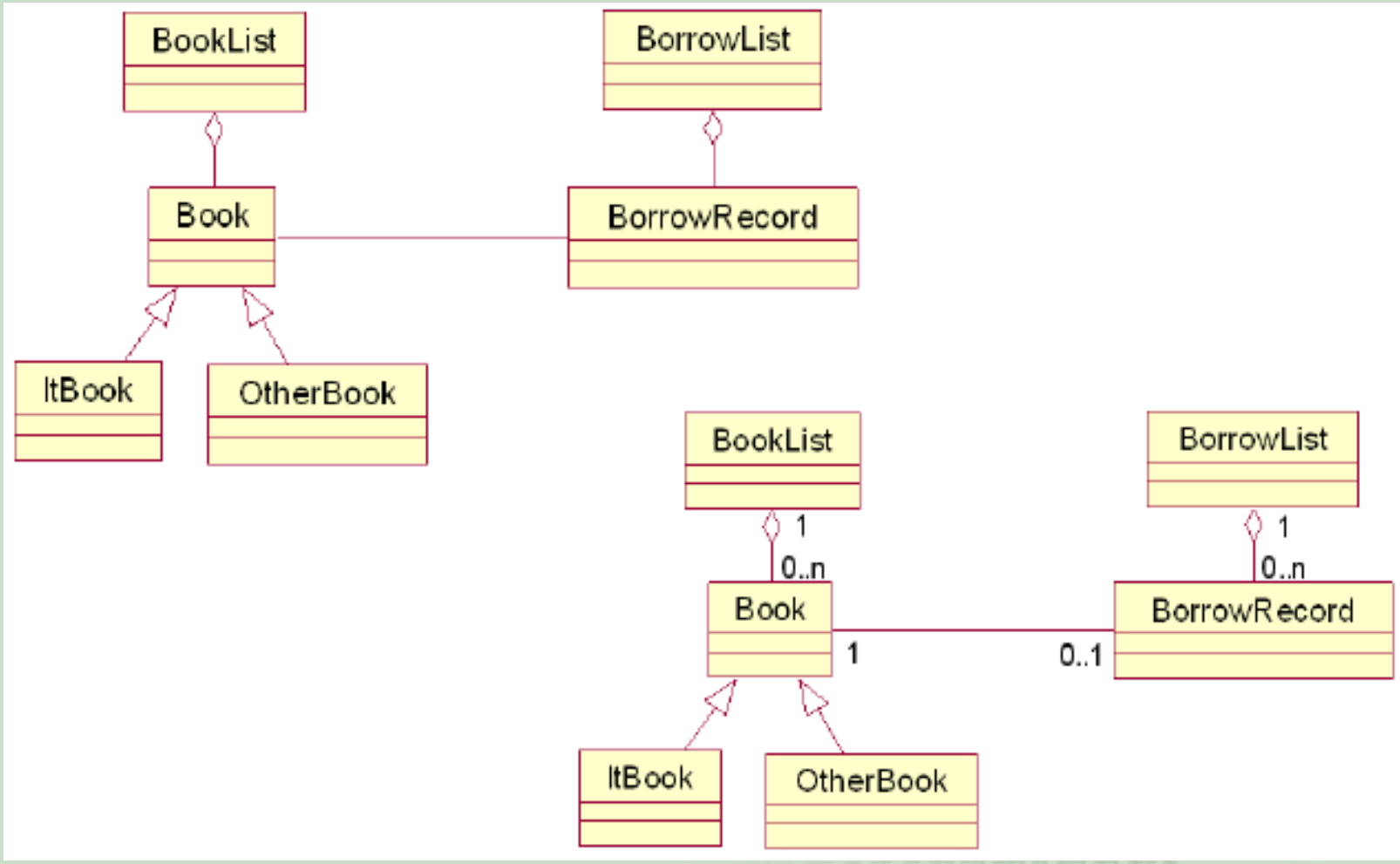
类图建模实例-发现类

小王是一个爱书之人，家里各类书籍已过千册，而平时又时常有朋友外借，因此需要一个个人图书管理系统。该系统应该能够将书籍的基本信息按计算机类、非计算机类分别建档，实现按书名、作者、类别、出版社等关键字的组合查询功能。在使用该系统录入新书籍时系统会自动按规则生成书号，可以修改信息，但一经创建就不允许删除。该系统还应该能够对书籍的外借情况进行记录，可对外借情况列表打印。另外，还希望能够对书籍的购买金额、册数按特定时间周期进行统计

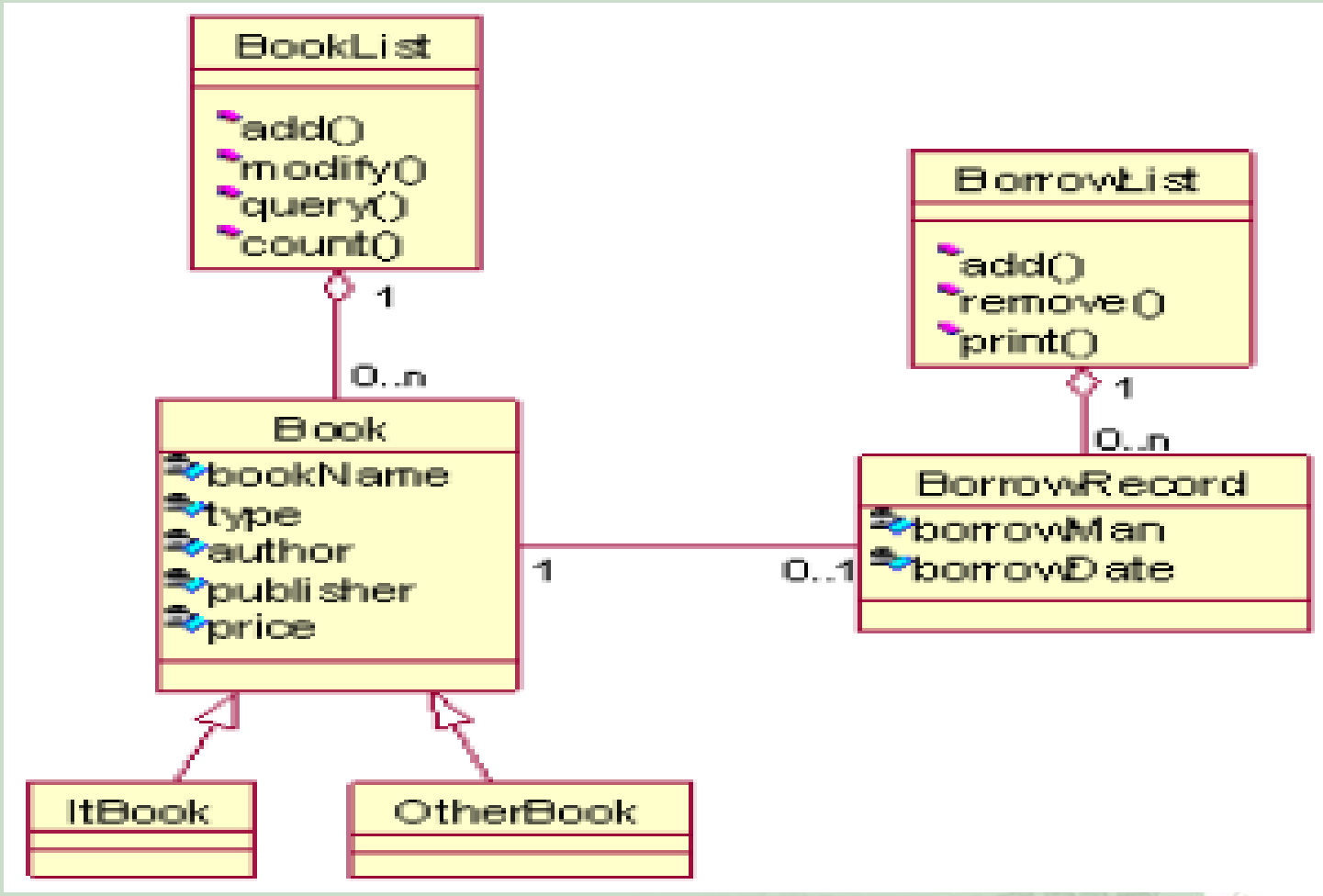
类图建模实例-初步确定类

- 书籍
- 计算机类书籍
- 非计算机类书籍
- 借阅记录
- 借阅记录列表
- 书籍列表

类图建模实例-确定类关系



类图建模实例


BookList
add()
modify()
query()
count()
1
0..n
Book
bookName
type
author
publisher
price
1
0..1
BorrowList
add()
remove()
print()
1
0..n
BorrowRecord
borrowMan
borrowDate
ItBook
OtherBook

UML类图建模风格

- 属性名和类型应该一致
- 如果属性的名字是customerNumber，其值是字符串，则存在不一致的问题
- 不要对有关联类的关联命名
- 关联类的名字已经充分描述关联了，这时关联不需要额外的修饰指明它的名字

UML类图建模风格

- 不要对每个依赖关系都建模
- 仅在能增加交流价值的情况下，才需对两个类之间的依赖关系建模

UML类图建模风格

- UML建模元素的连线：

  --正交风格：直角相交的直线

  --倾斜风格：斜线

  --坚持一种风格，除非混合两种风格可以增加图的可读性