

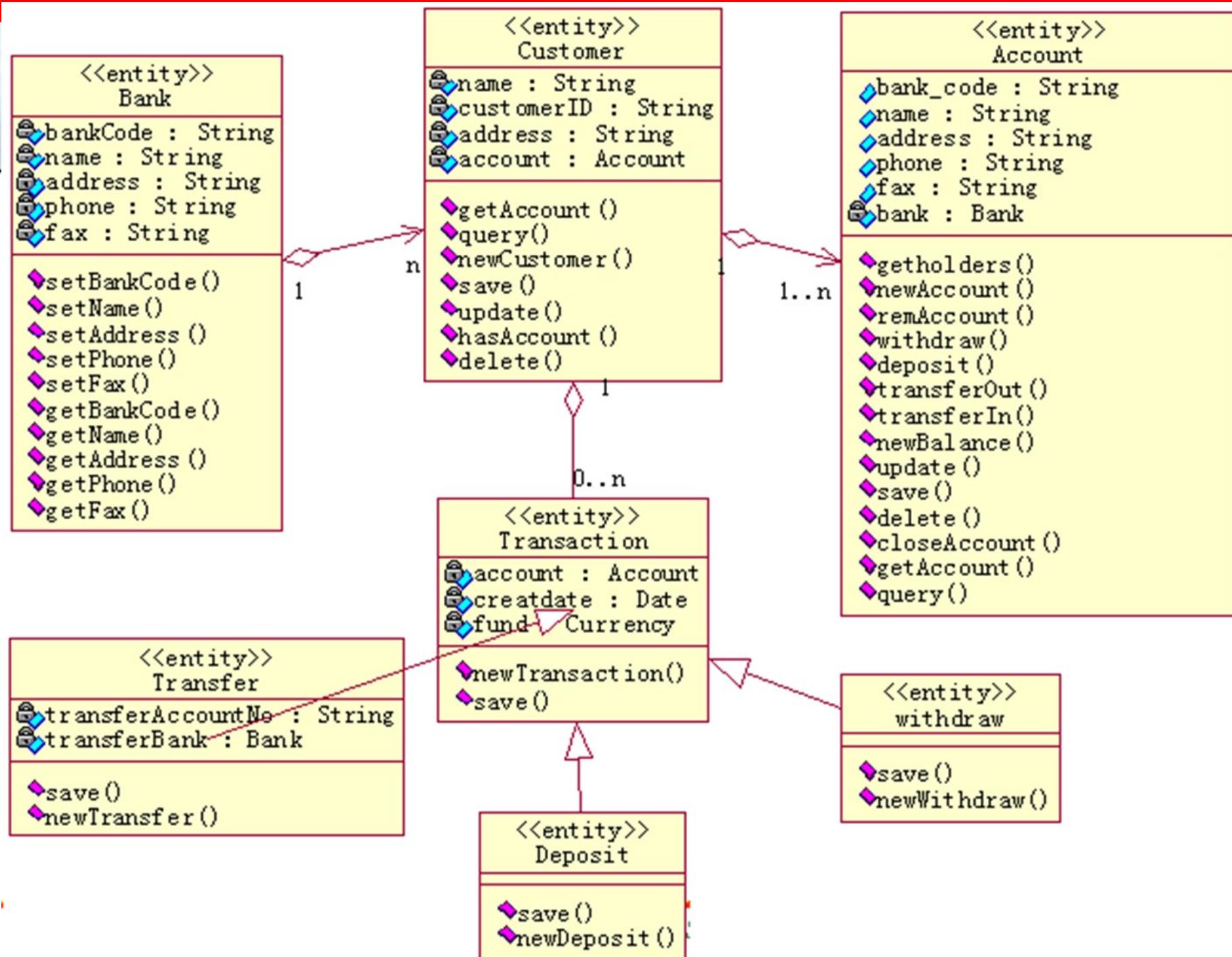
# 类图



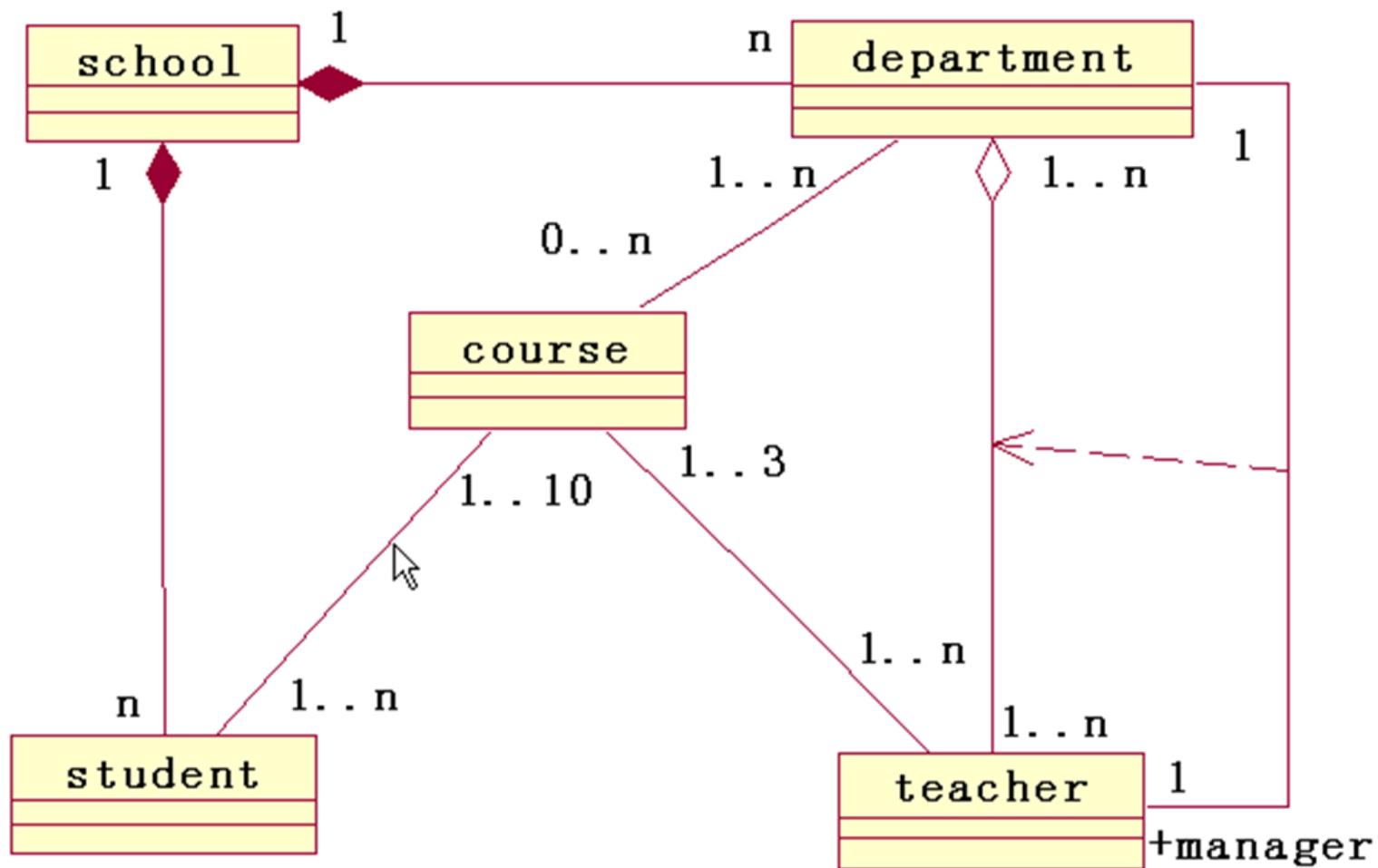
- 为什么要进行软件建模?
  - 软件系统越来越大，任何个人都不可能单独管理这些代码；
  - 没有参加开发的人员无法直接理解程序代码；
  - 我们需要一种描述复杂系统的简单方法。

# 类图的作用

- 类图的作用:类图常用来描述业务或软件系统的组成、结构和关系。我们通常通过下面三种方式使用类图:
  - 对系统词汇建模
    - 为系统的词汇建模实际上是从词汇表中发现类，发现它的责任。
  - 对协作建模
    - 协作是指一些类、接口和其他的元素一起工作，提供一些合作的行为，这些行为不是简单地将元素加在一起就能实现的。
  - 对逻辑数据库模式建模
    - 我们常用类图设计数据库的蓝图。在很多领域，我们想把持久性数据保存到关系数据库或面向对象的数据库中。我们可以用类图为这些数据库模式建立模型。



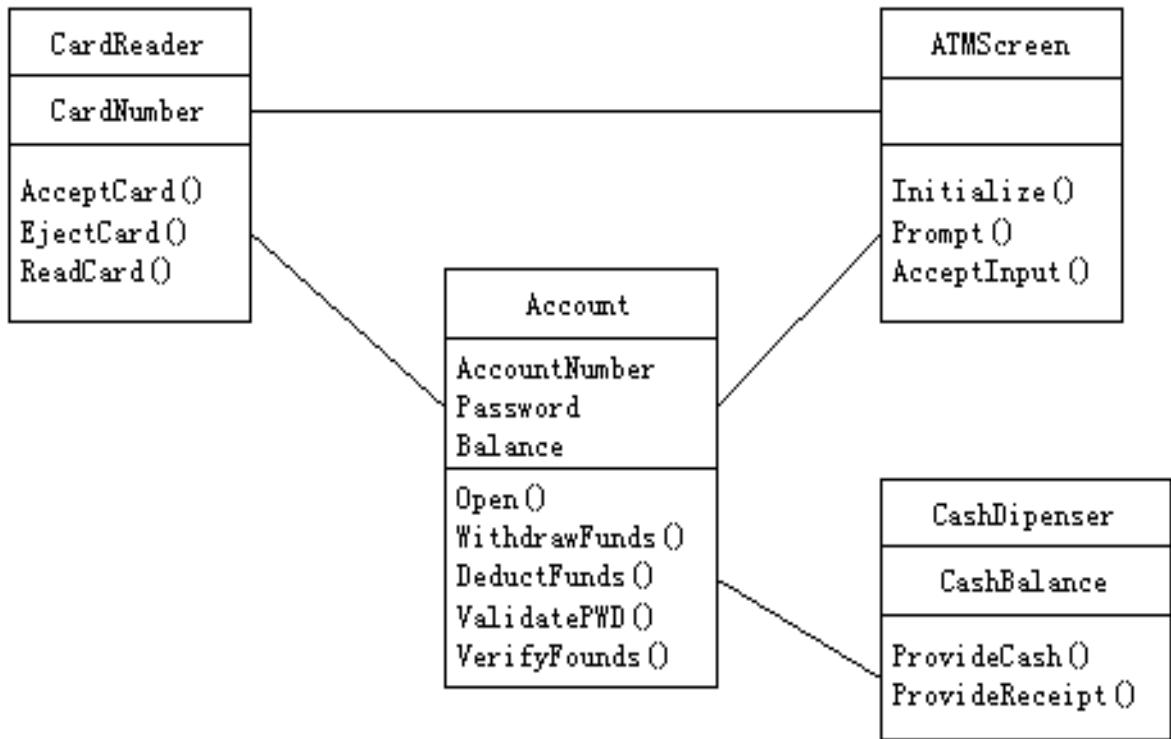
# 某校的类图





- 类图在**UML** 的静态机制中是重要的组成部分，它不但是设计人员关心的核心，更是实现人员关注的重点。建模工具也主要根据类图来产生代码。
- 类图是用来显示系统中的类、接口以及它们之间的静态结构和关系的一种静态模型，它用于描述系统的结构。类图的建模贯穿系统的分析和设计阶段的始终，通常从用户能够理解的用例开始建模，最终到系统开发小组能够完全理解的类。

# 类图概要

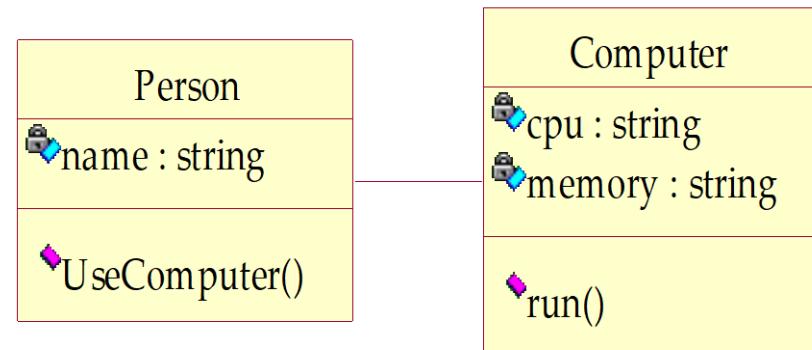
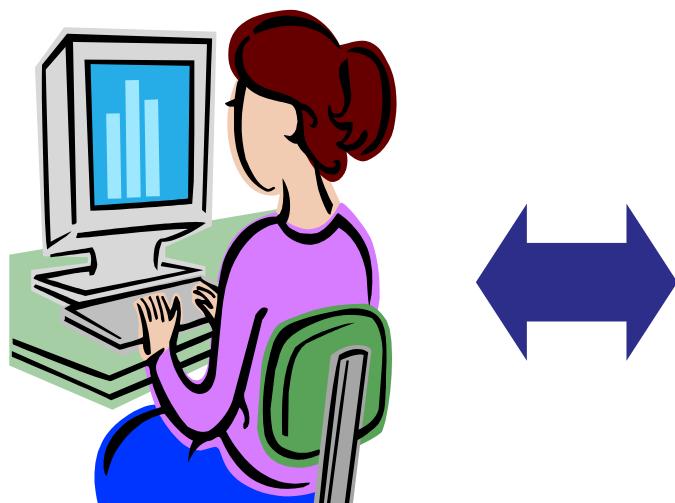


➤ 表示系统中的类和类与类之间的关系，系统静态结构的描述。  
不仅定义系统中的类，表示类之间的联系如关联、依赖、聚合等，也包括类的内部结构（类的属性和操作）

➤ 类图是以**类**为中心来组织的，类图中的其他元素或属于某个类或与类相关联

# 类图概要

- 类图以反映类的结构(属性、操作)以及类之间的关系为主要目的，描述了软件系统的结构，是一种静态建模方法
- 类图中的“类”与面向对象语言中的“类”的概念是对应的，是对现实世界中的事物的抽象





# 类图的组成元素

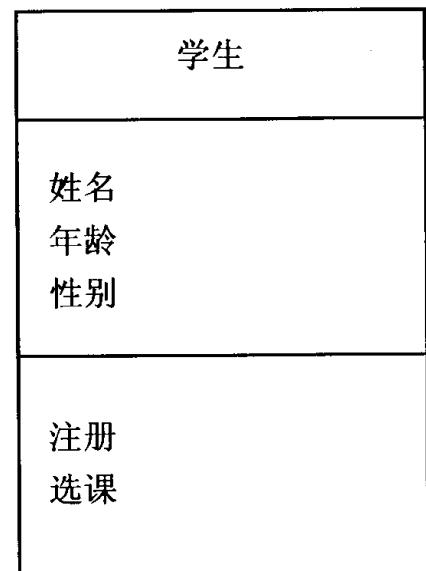
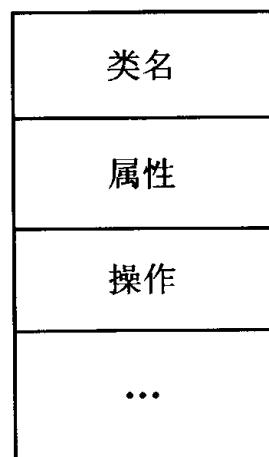
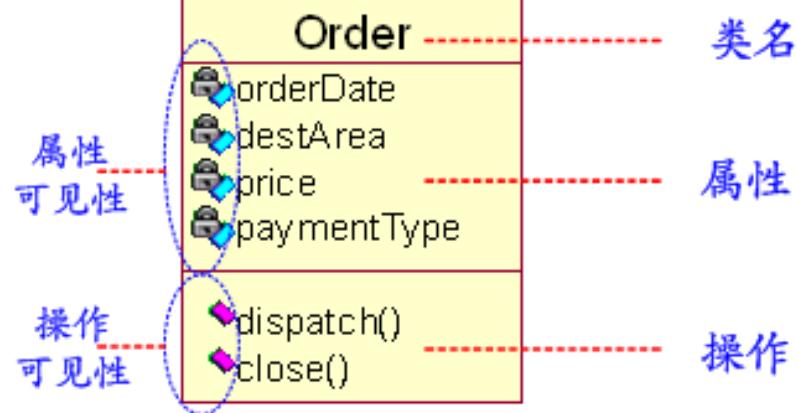
- **类(Class)**: 是具有共同结构特征、行为特征、联系和语义的对象集合的抽象形式。
- **关联（Association）** : 它表示类与类之间的关系。

# 类

- 概念与表示法
  - 对象是具有明确语义边界并封装了状态和行为的实体，由一组属性和作用在这组属性上的一组操作构成，是构成系统的一个基本单位。
  - 类是对一组具有相同属性和操作的一组对象的抽象描述。
    - 一个类的所有对象具有相同的属性，是指所有对象的属性的个数、名称、数据类型都相同，各个对象的属性值则可以互不相同，并且随着程序的执行而变化。
    - 至于操作，对于一个类的所有对象都是一样的，即所有的对象共同使用它们的类定义中给出的操作。

# 类

- UML中，表示一个类，主要是标识它的名称、属性和操作。  
类由一个矩形表示，它包含3栏，在每栏中分别写入类的名称  
、类的属性和类的操作。



类的图形表示和示例



# 类的表示

## – 类的名称

- 每个类都必须有一个有别于其他类的名称, 类名部分是不能省略的。表示方法有两种:
  - 简名单: **Order** (订单), 它就只是一个单独的名称。
  - 全名: 也称为路径名, 就是在类名前面加上包的名称, 例如 **java::awt::Rectangel**、**businessRule::order**等。

<i>Account</i>
-balance: double=1
+Deposite(Amount: double):int +ComputeInterest(): double

<i>Business::Account</i>
-balance: double=1
+Deposite(Amount: double):int +ComputeInterest(): double

# 类的表示



① 名词或名词短语(动词或动词短语表示控制类)

例如：人，桌子，图形，汇总

② 尽可能用明确、简短，业务领域中事物的名称，避免使用抽象、无意义的名词

例如：帐户，订单，事物

③ 用英文，第1个字母大写

例如：**Shape, Person, CheckingAccount**

④ 可分为简单类名，带路径类名

例如：**CheckingAccount**

**Banking::CheckingAccdount**

# 类的表示

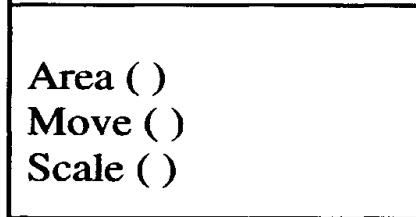
在类图中，根据建模的不同景象，类图标中不一定列出全部的内容。如在建立分析模型或设计模型时，甚至可以只列出类名，在图中着重表达的是类与类之间的联系；在建立实现模型时，则应当在类图标中详细给出类的属性和方法等细节。



(a)



Centre  
Radius



(b)



- centre: float  
- radius: float

+ Circle()  
《query》  
area( ): float  
...  
《update》  
move(location: Point)  
scale(ratio: float)  
...

(c)

# 类的表示

## 1. 属性的含义

**属性(attribute)**: 描述类所表示事物的静态性质。属性描述了类在软件系统中代表的事物（即对象）所具备的特性。

## 2. 属性的格式

[可见性] 属性名[:类型][‘多重性[次序]’][=初始值][{特性}]

表示属性约束说明:

例如: #visibility:Boolean=false{读写}

表示属性” visibility”可读,写

属性描述了类在软件系统中代表的事物（即对象）所具备的特性。类可以有任意数目的属性，也可以没有属性。类如果有属性，则每一个属性都必须有一个名字，另外还可以有其他的描述信息，如可见性、数据类型、默认值等。

<i>Account</i>
<b>-balance: double=1</b>
<b>+Deposite(Amount: double):int</b>
<b>+ComputeInterest(): double</b>

# 类的表示



## 1. 操作的含义

**操作(operation)**:对类的对象所能做的事务的一个抽象，描述类所表示事物的动态性质。

## 2. 操作的格式

[可见性]操作名[(参数列表):返回类型][{特性}]



# 类的表示

## - 类的职责

- 职责指类承担的责任和义务。在矩形框中最后一栏中写明类的职责。

## - 类的约束

- 约束指定了类所要满足的一个或多个规则。在UML中，约束是用花括号括起来的自由文本。

Title
name : String author : String isbn : String / number of reservations
\$find() create() destroy()
Responsibility

Title
name : String author : String isbn : String / number of reservations
\$find() create() destroy()

{Constraint}

借阅者
-借阅证号:int -是否有借阅资源:int +类别:string +姓名:string +性别: string
+借书(): void +还书(): void

{类别=教师 or 学生 or 行政管理人员}



**【例5-1】**在图书管理系统的借阅者类，类名为借阅者，共有5个属性：借阅证号、是否有借阅资源、姓名、性别和类别；操作有借书和还书。

借阅者是类的名称，5个属性中，借阅证号、是否有借阅资源是私有属性（private），类型都为int。姓名、性别和类别属性是公有属性，类型都是string。2个操作都是公有的（public），均没有返回值。



借阅者类用Java实现的程序如下：

```
Public class Borrower
{
    //定义属性
    private int int borrowedId;
    private int orNotCanBorrow;
    public String type;
    public String name;
    public String sex;
    //定义操作
    public void borrowBook()
    {
        .....
    }
    public void ReturnBook()
    {
        .....
    }
}
```

# 课堂提问

- 下列哪个符号在UML中代表类（ ）。
  - A. 矩形
  - B. 菱形
  - C. 圆形
  - D. 椭圆

答案：A

# 课堂提问

- 在**UML**类图中，表示类的三个基本要素不包括（ ）。
  - A. 类名
  - B. 属性
  - C. 方法
  - D. 关系

答案：D

# 类图中的关系



- **关联关系 (Association)**

- 关联(association):** 模型元素之间的一种语义联系,它是对具有共同的结构特性、行为特性、关系和语义的链的描述。
- 关联可以分为单向关联, 双向关联。**



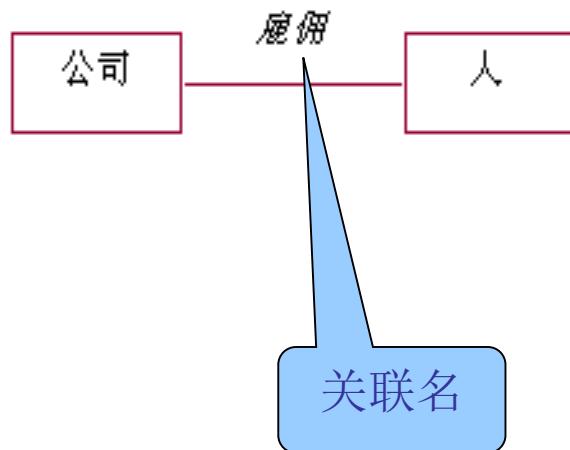
单向关联



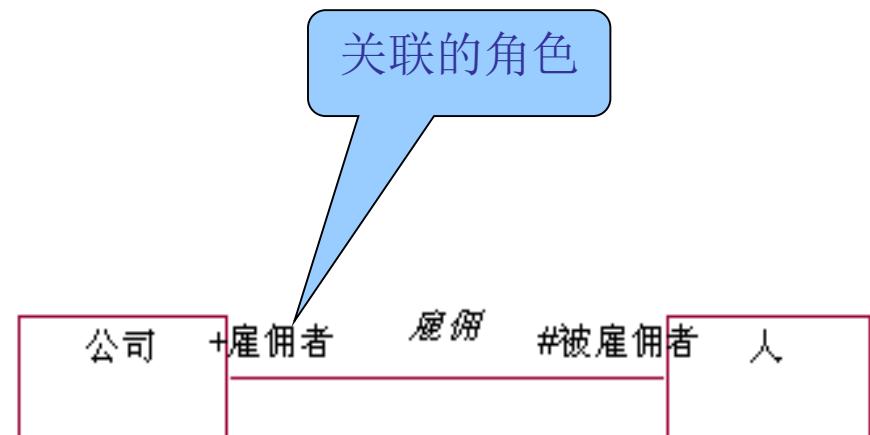
双向关联

# 关联的特性

## (1) 关联名



## (2) 关联的角色

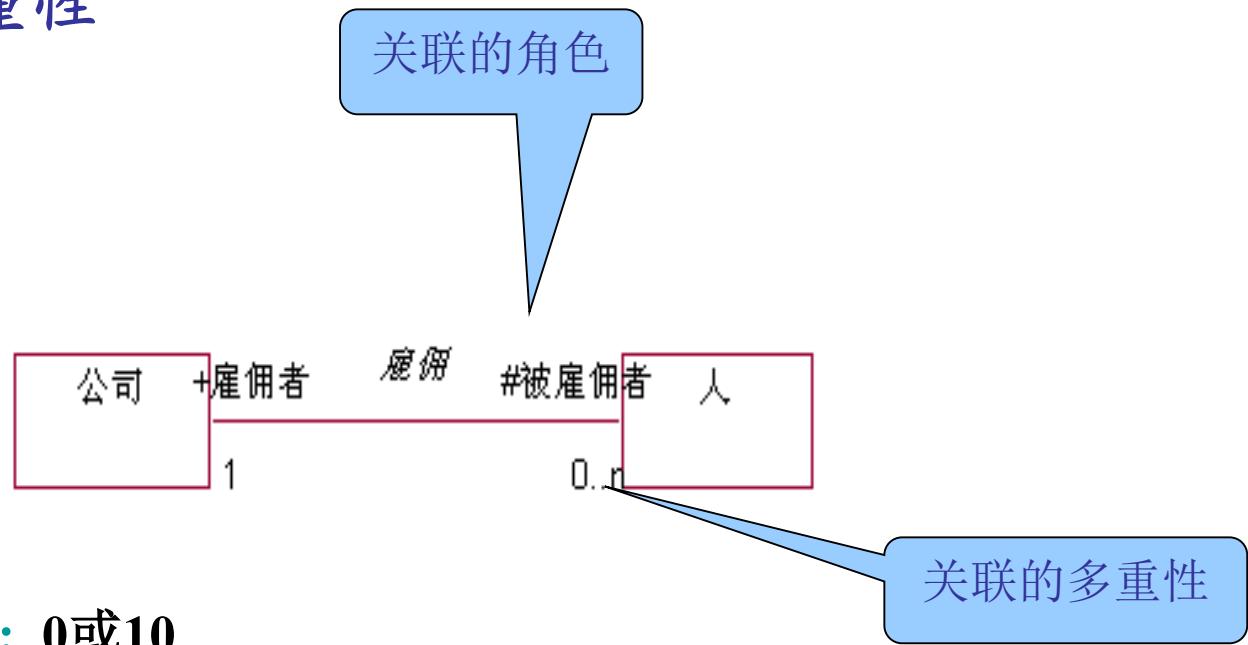


如果关联关系已经清楚，  
就无需关联名

如果关联名与角色名相  
同，则不标出角色名

# 关联的特性

## (3) 关联的多重性



0,10 : 0或10

0..\* : 0到多个

1

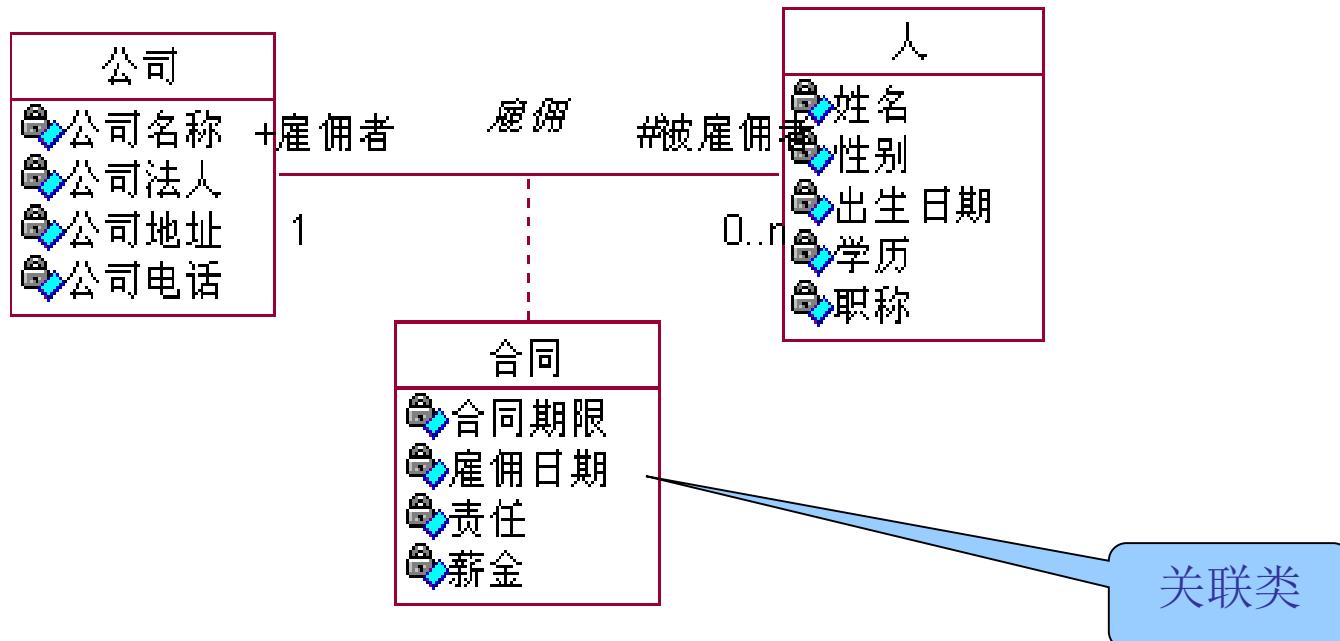
1..n : 1到多

\* : 0到多

# 关联的特性

## (4) 关联类

通过关联类描述关联的属性，操作，及其它信息。

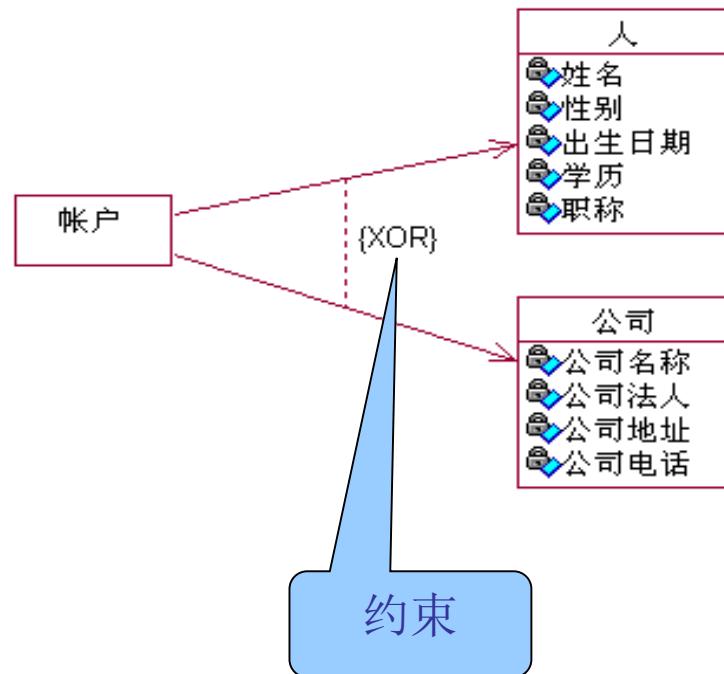


# 关联的特性

## (5) 关联的约束

通过约束加强关联的含义。

例如，“帐户”不能同时与“人”和“公司”有关联。



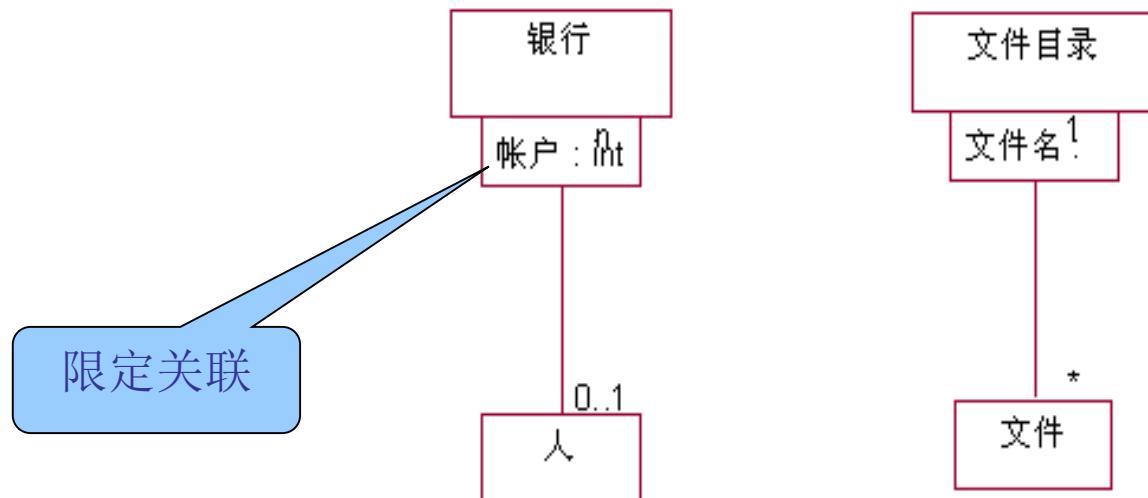
# 关联的特性

## (6) 限定关联

通过限定符来规定关联的限定关系。

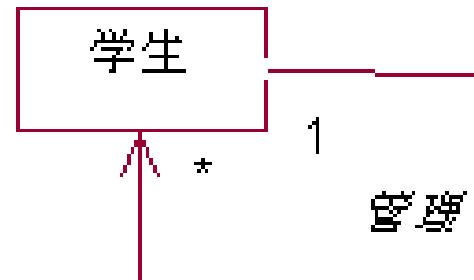
例如“文件目录”通过“文件名”来与具体的文件对象进行关联。

“银行”通过“帐户”与具体的“人”对象进行关联。



# 关联的种类

## ① 一元(自返)关联

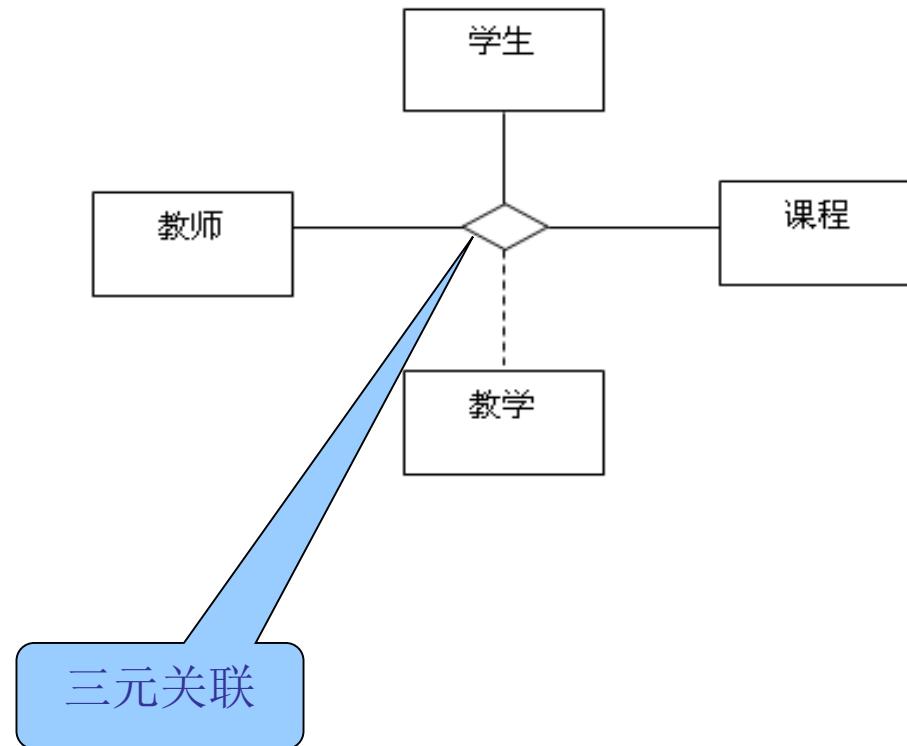


## ② 二元关联



# 关联的种类

## ③ 多元关联



# 练习

“教师”和“学生”两个类之间存在授课关系，一个教师可以教授多个学生，一个学生可以由多个教师授课，标出这两个类的关系。

教师

学生



```
public class A  
{  
    public B theB;  
    .....  
    public A() { }  
}
```

```
public class B  
{  
    .....  
    public B() { }  
}
```

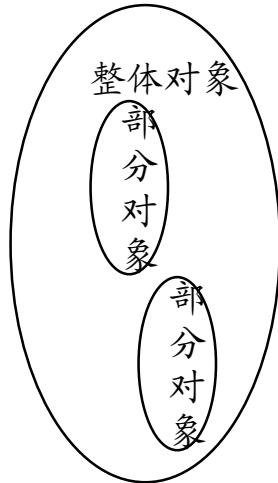


# 类图中的关系

- 一般的关联关系语义较弱。也有两种语义较强，分别是聚合与组合
  - **聚合(aggregation)** 是关联的一种特殊形式，表示整体类和部分分类之间的“整体一部分”关系。
  - **组合**是聚合的一种形式，其部分类的对象和整体类的对象之间有很强的“属于”关系，整体类的对象管理部分类的对象，决定部分类的对象何时属于它，何时不属于它。

# 类图中的关系

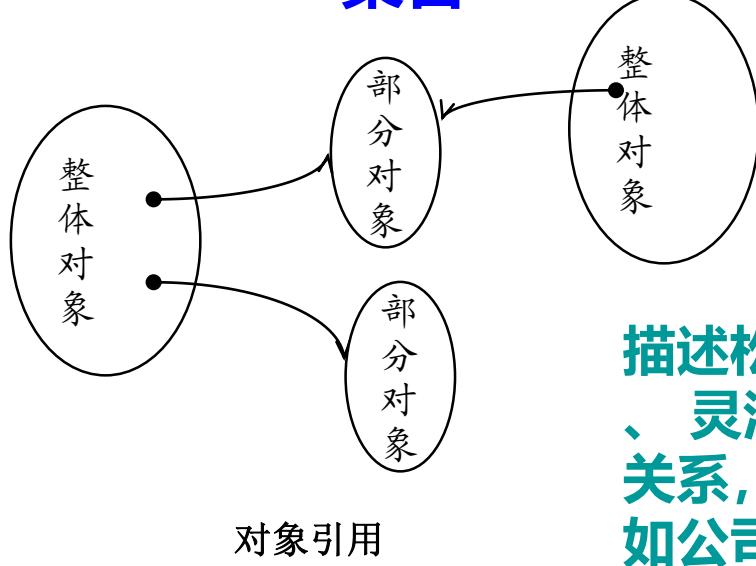
## 组合



嵌套对象

描述紧密、固定的关系，例如汽车与发动机

## 聚合



整体对象  
部分对象

部分对象

Class B {}  
Class A  
{... B b; ...}

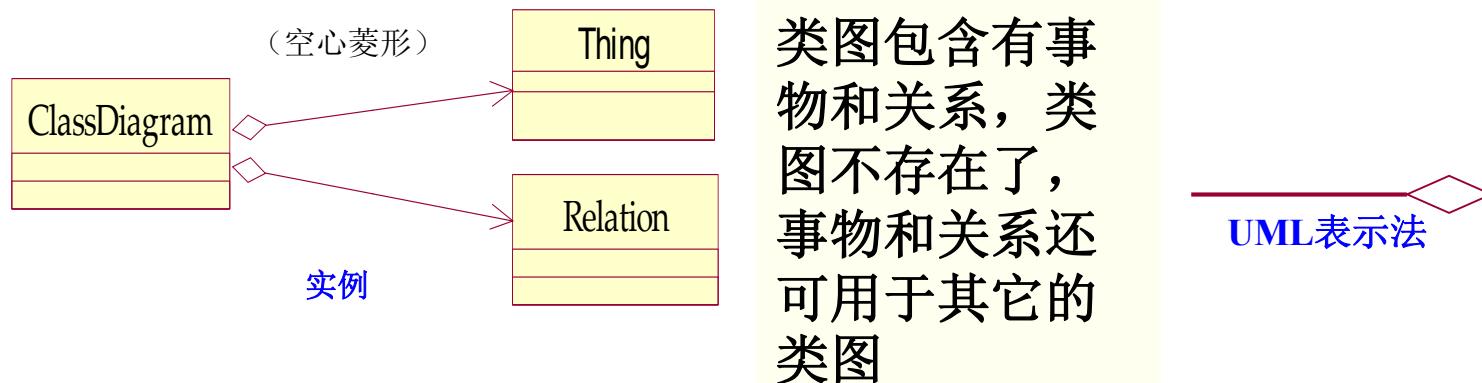
描述松散、灵活的关系，例如公司与法律顾问

类与类之间的聚合关系指的是，一个类的对象实例，以另一个类的对象实例作为其组成部分，是种“a part of”或“has a”；也可理解为，一个类定义引用另一个类定义。

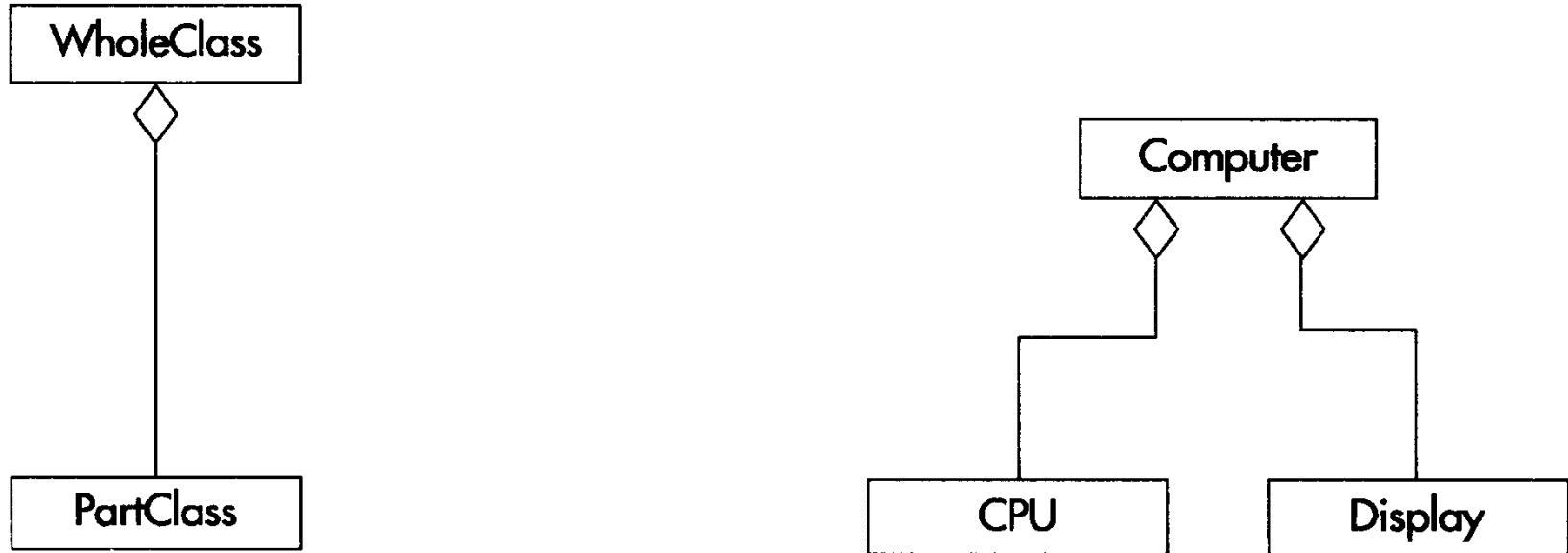
# 类图中的关系

## 聚合关系

- 特殊关联关系，指明一个聚集（整体）和组成部分之间的关系。
- 在聚合中，部分分类可以没有整体类而存在。



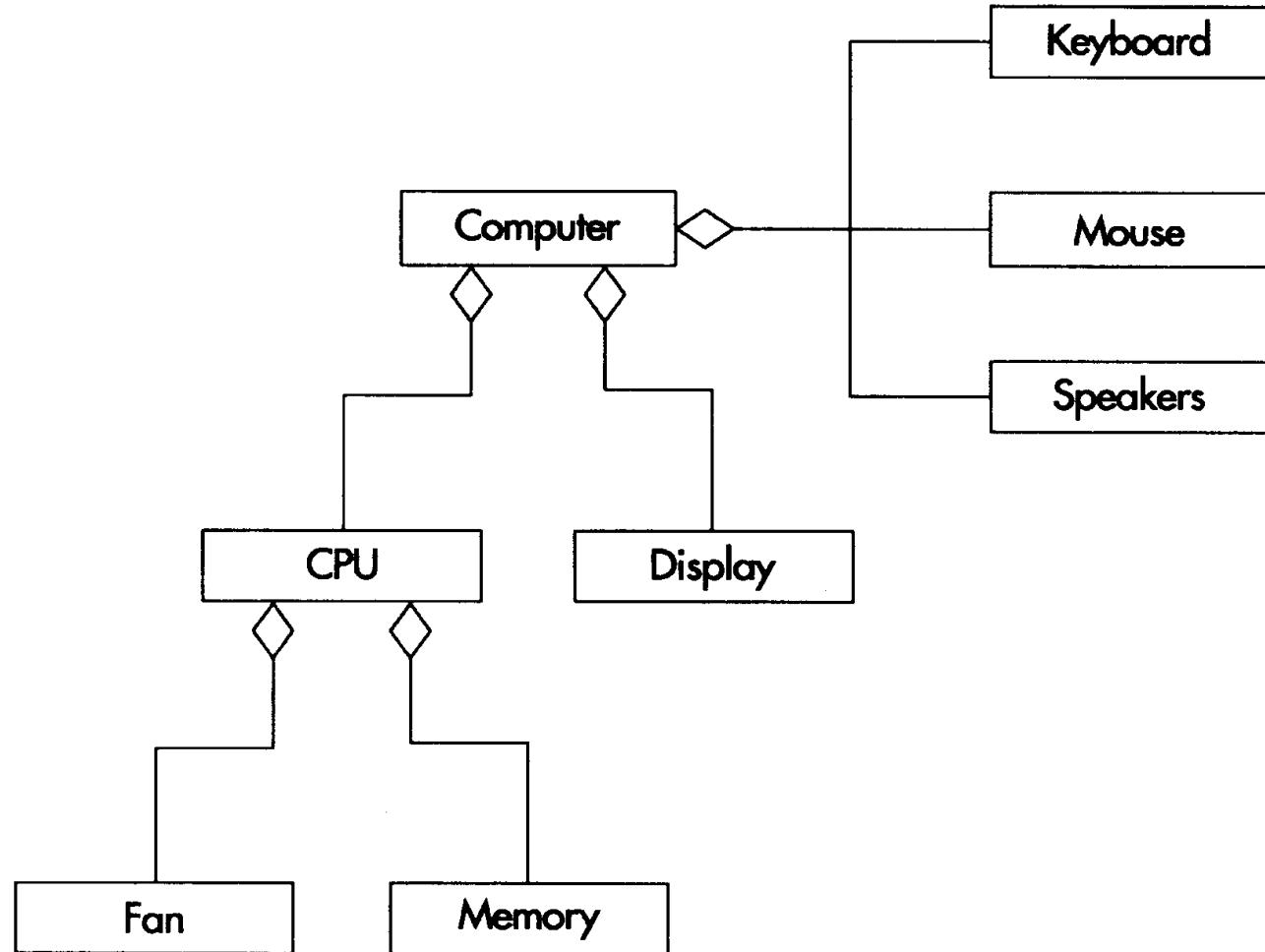
# 类图中的关系



例如，CPU和显示器都可以以独立类的形式存在，但是当它们组成Computer类时，它们就变为整个计算机的组成部分。

# 类图中的关系

通过提供其他的计算机部件，如键盘、鼠标和扬声器来扩展该示例。



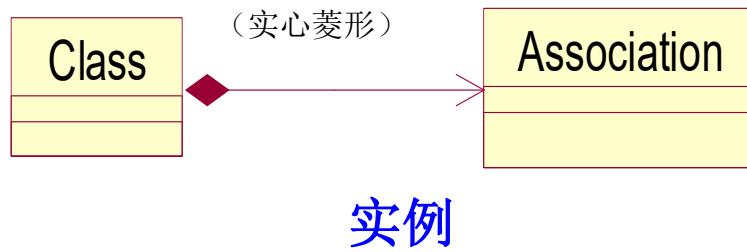
# 类图中的关系



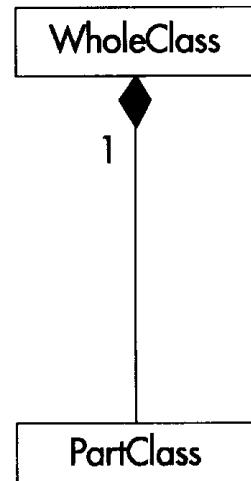
## 组合关系

语义更强的聚合，部分和整体具有相同的生命周期。在组合关联中用来组成整体类的部分类是不能独立存在。整体类由部分类组成，部分类需要整体类才能存在。这种关系意味着销毁整体类将会同时销毁部分类。

UML表示法

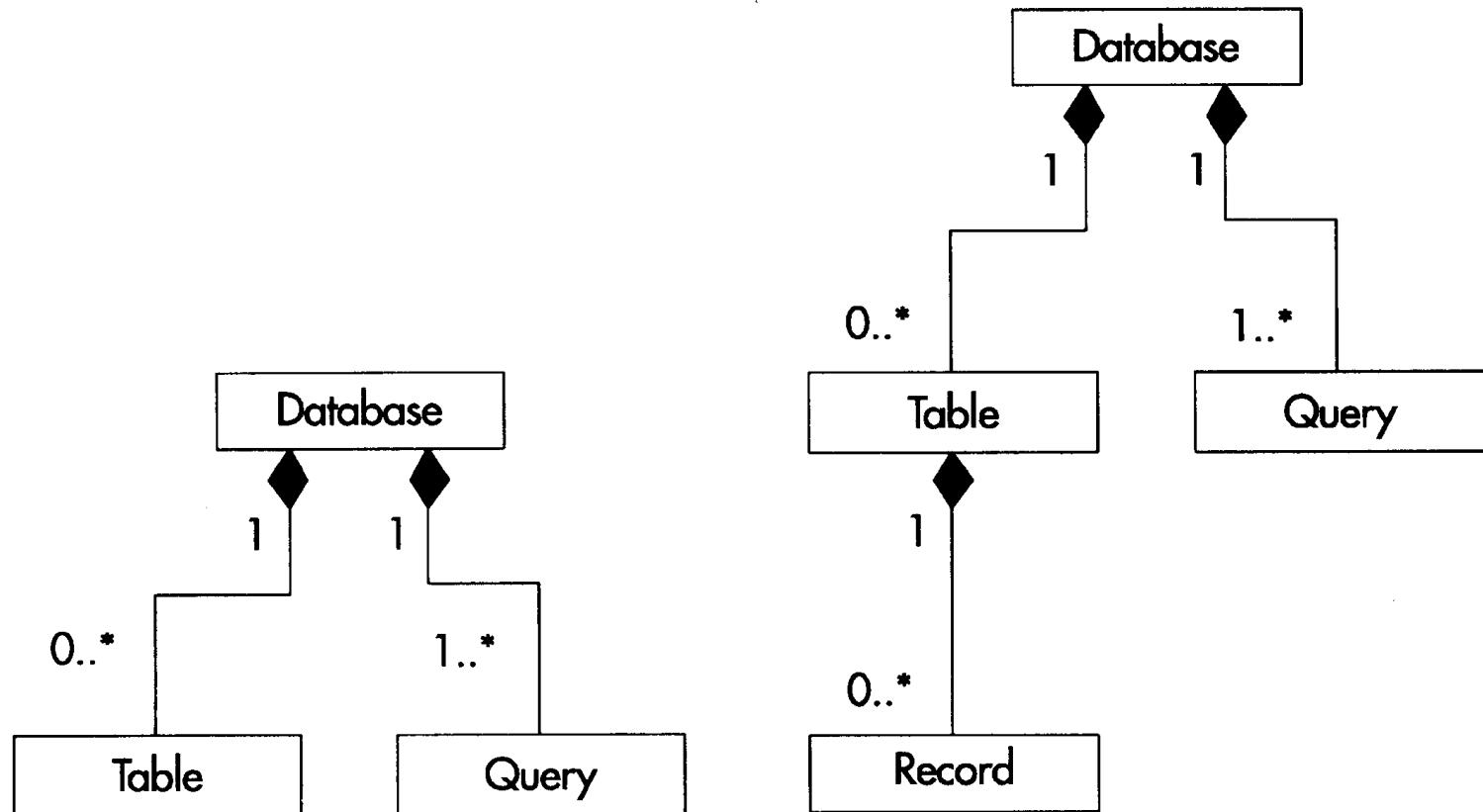


类与关联关系之间有组合关系，  
类不存在了，则相应的关联关系  
也不存在



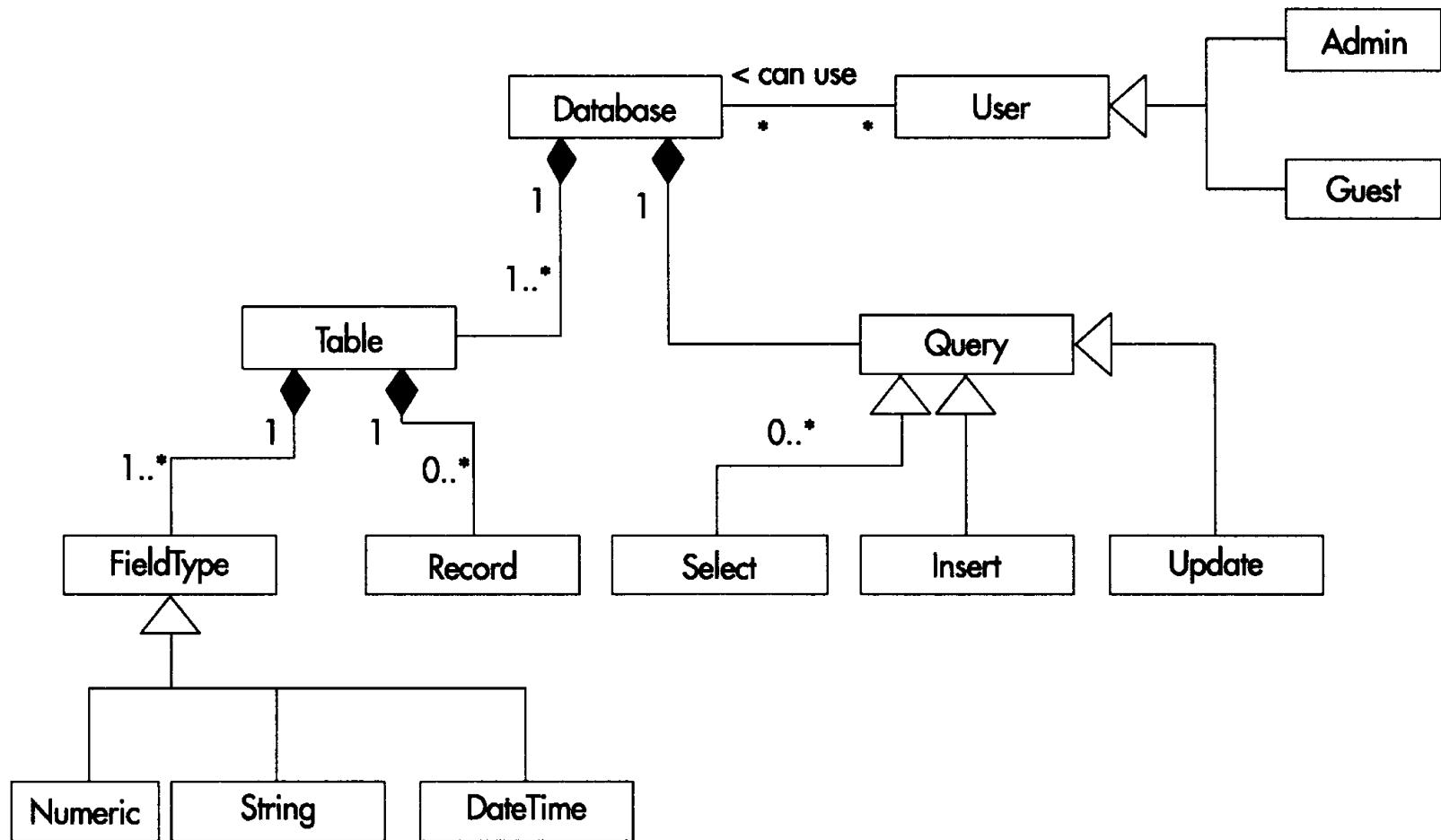
# 类图中的关系

由于组合关联指示的部分类是强制的，对于整体类意味着至少有一个多重性。在下面的示例中，整体类数据库由表和查询组成。这些关联使用组合表示，因为如果没有数据库，表和查询也不会存在。



# 类图中的关系

聚合和组合表示的是类之间的关系，它们可以与泛化结合来进一步扩展类图模型。





## 聚合与组合的区别：

- (1) 聚合松散，组合紧密；
- (2) 一个部分事物对象可以属于多个聚集对象,但一个部分事物对象仅能属于一个组合对象；
- (3) 聚集的对象生命周期可以不同,但组合对象则是同存同亡。

# 课堂提问

- 在UML中，用什么符号表示两个类之间的关联关系（ ）。
  - A. 箭头
  - B. 虚线
  - C. 实线
  - D. 双线

答案：C

# 课堂提问

- 表示一个类包含另一个类，且包含的类不能独立存在，这种关系称为（ ）。
  - A. 关联
  - B. 泛化
  - C. 聚合
  - D. 组合

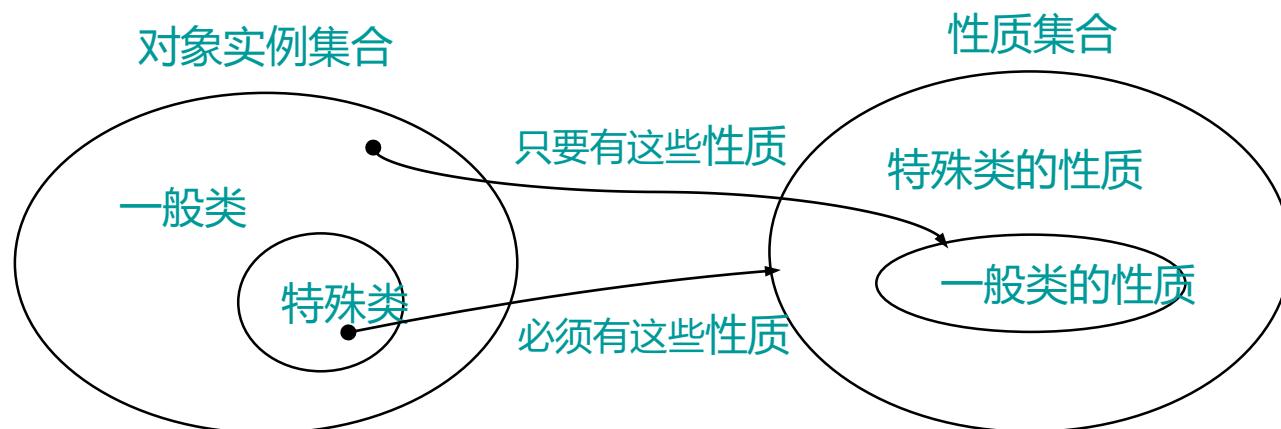
答案：D



- 泛化关系 (Generalization)

- 泛化是较特殊的类和较一般的类之间的直接关系（继承关系），其中较一般的类具有较特殊的类的共同性质，较特殊的类继承较一般的类的性质，且还具有自己的性质，较特殊的类的对象是较一般的类的对象的子集。

理解一般类与特殊类之间的关系



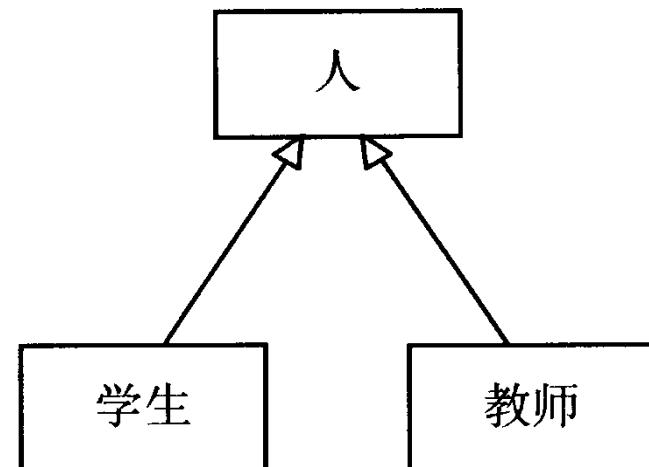
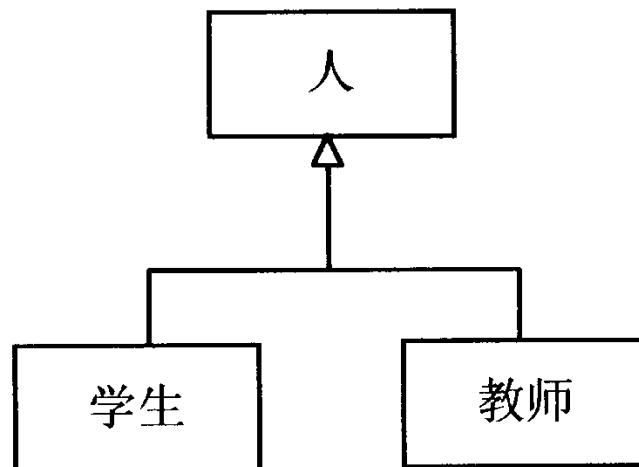
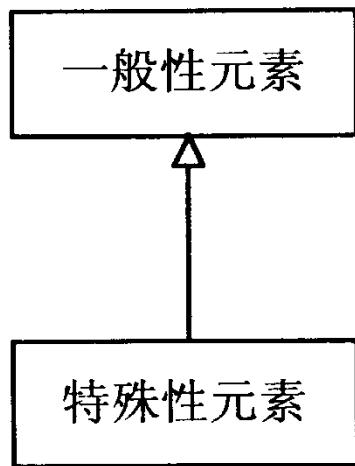
# 类图中的关系



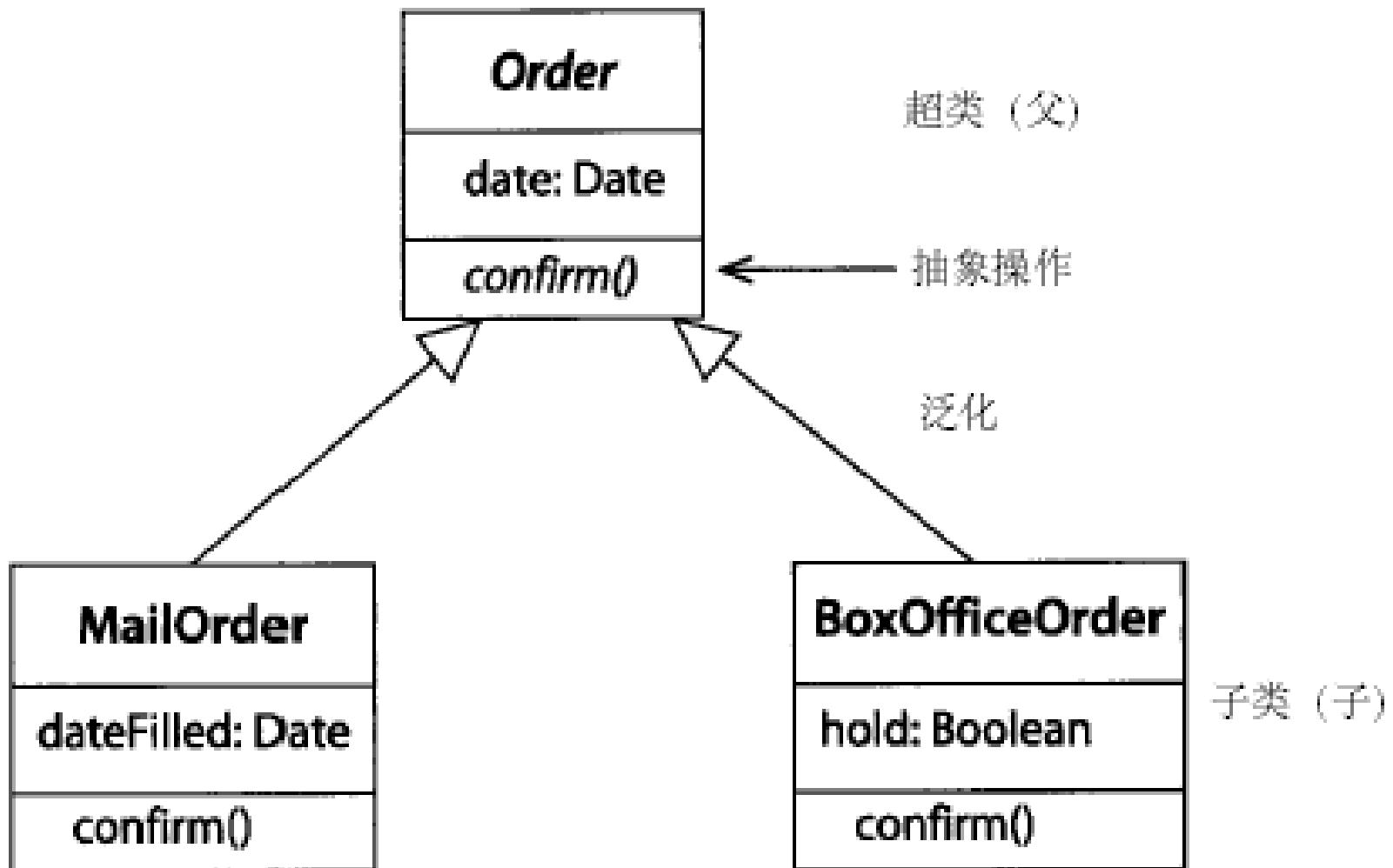
## 泛化关系

UML表示法

- 泛化关系是描述类之间的继承关系。利用泛化来表达类之间的相似性。

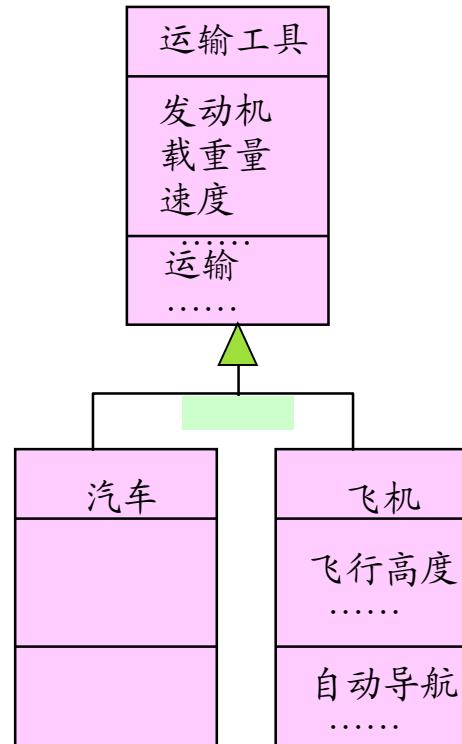
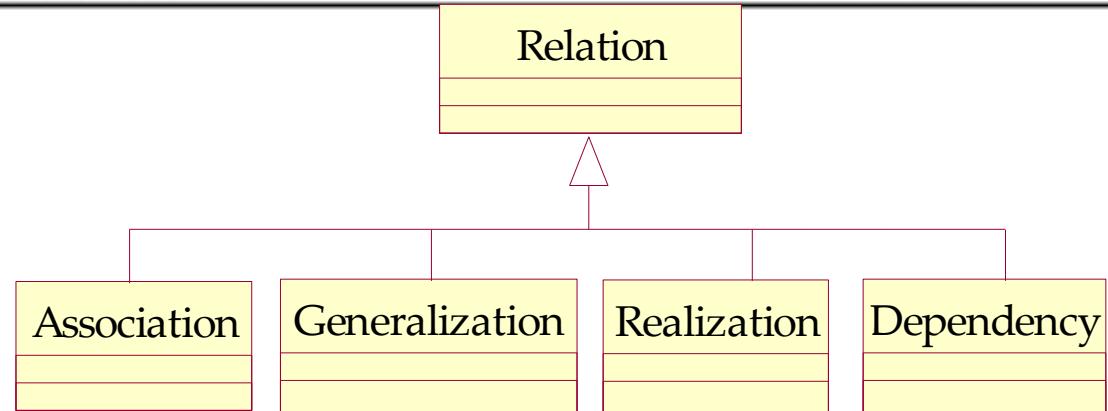
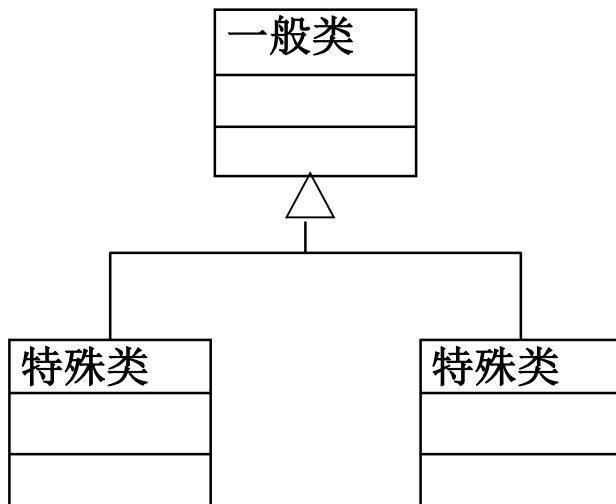


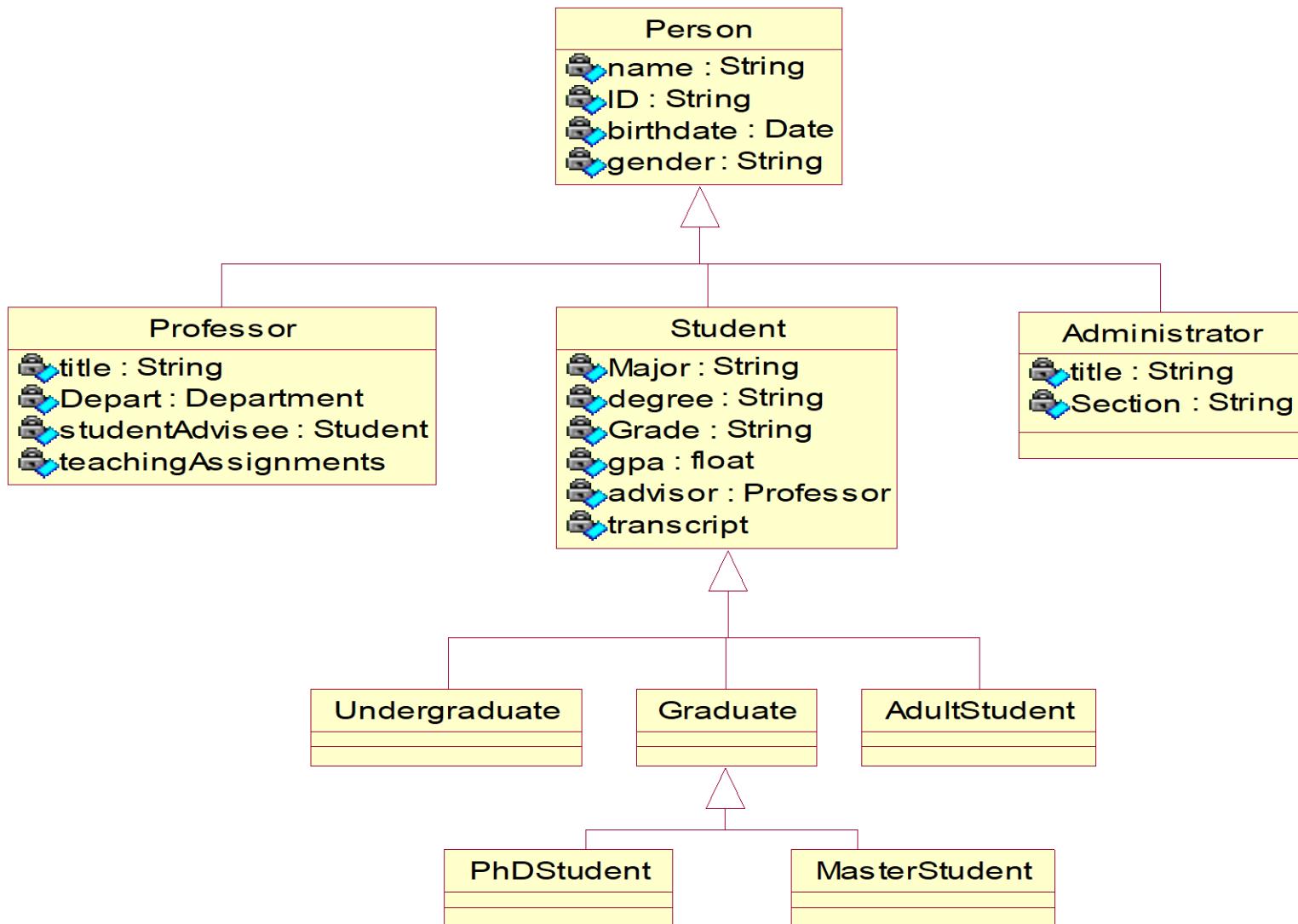
# 类图中的关系

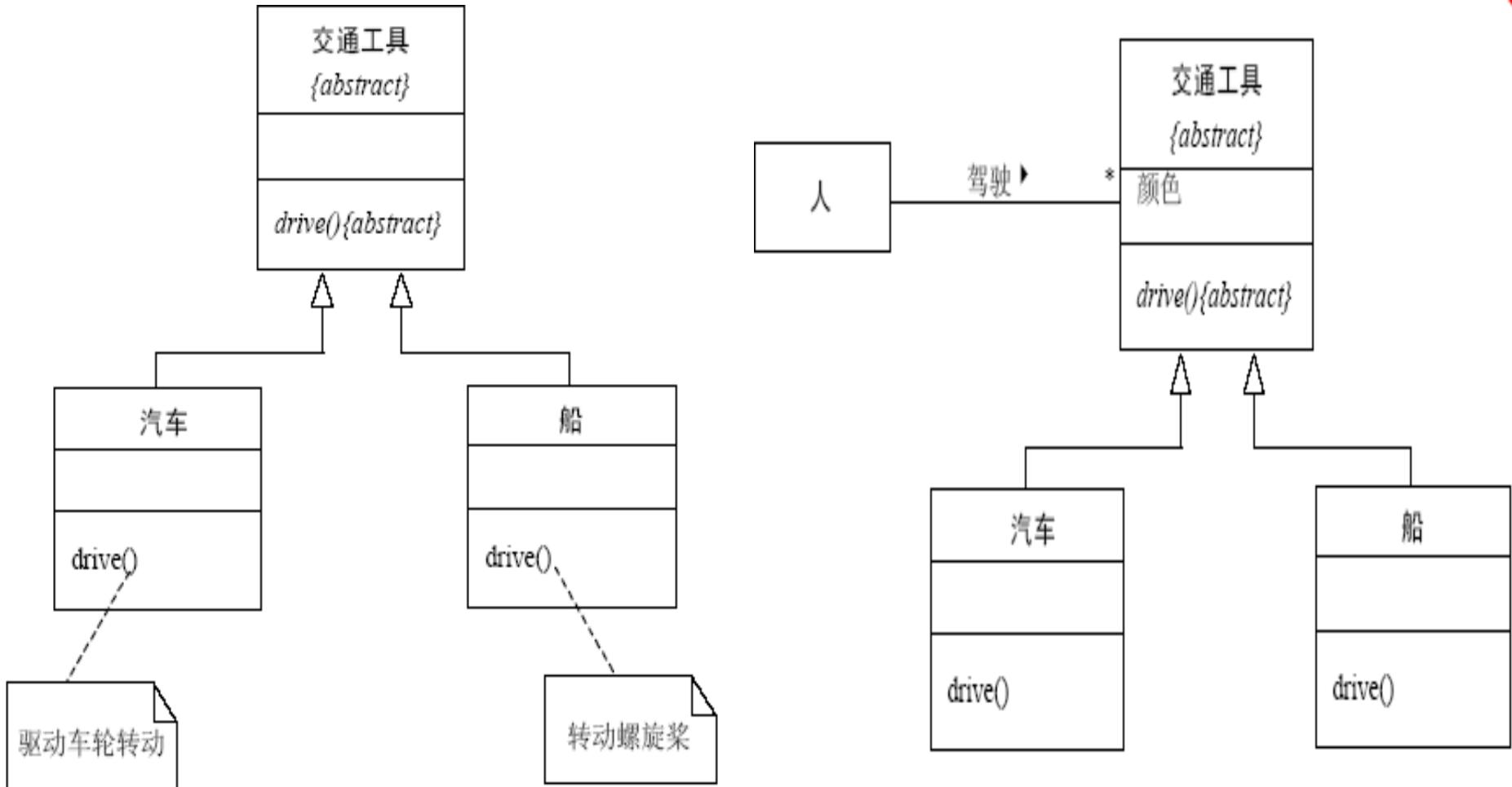




## • 泛化表示法







# 课堂提问

- 表示一个类可以有一个或多个父类，这种关系称为（ ）。
  - A. 关联
  - B. 泛化
  - C. 聚合
  - D. 组合

答案：B

# 类图中的关系



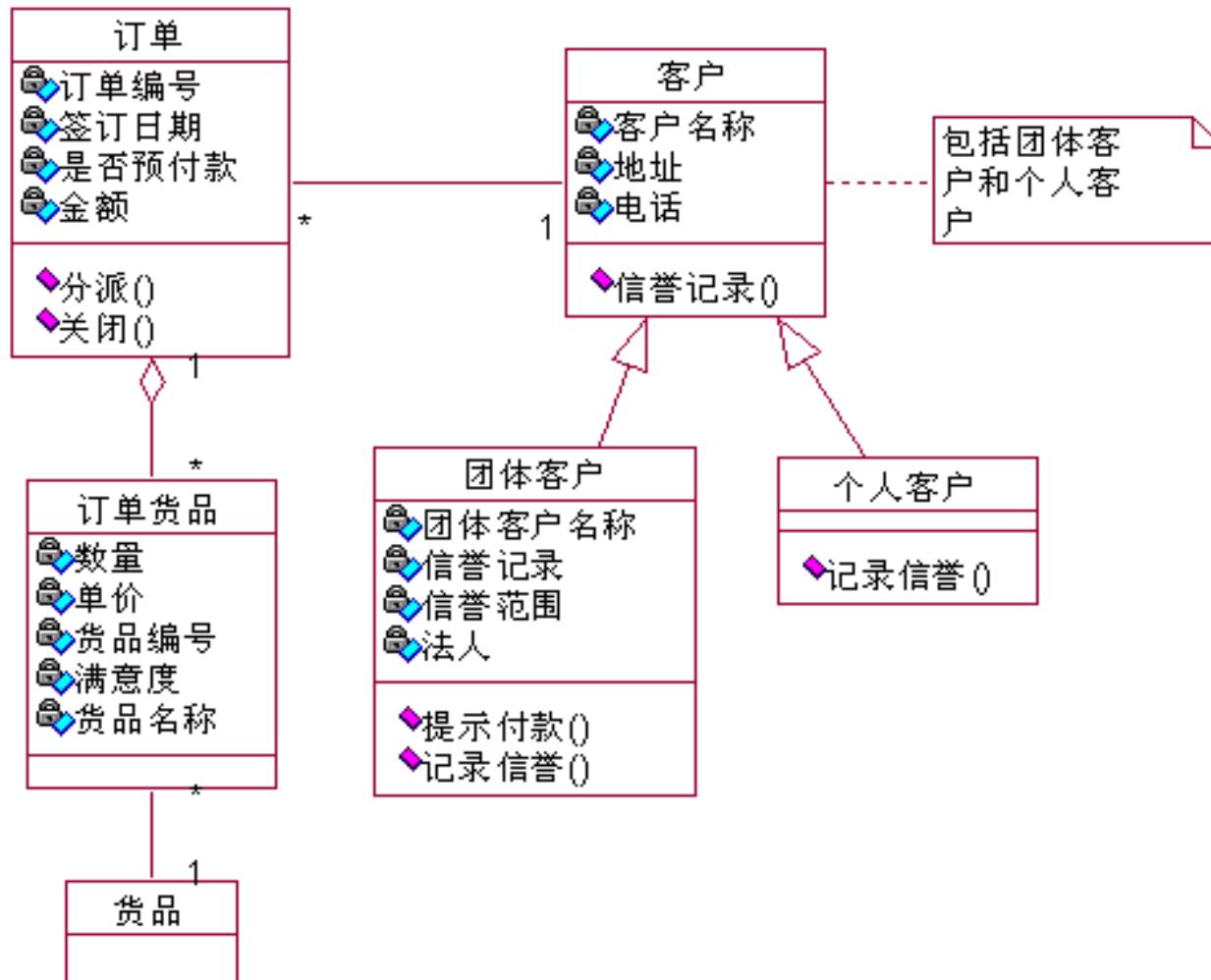
## 练习：阅读一个类图

在这个练习中，将会通过识别到目前为止学习的**UML**标记符来阅读下面的类图，如图所示。

- 1) 指出建模的类。
- 2) 指出所有属性。
- 3) 指出所有显示的操作。
- 4) 指出找到的关联。

# 类图中的关系

## 订货系统的类图

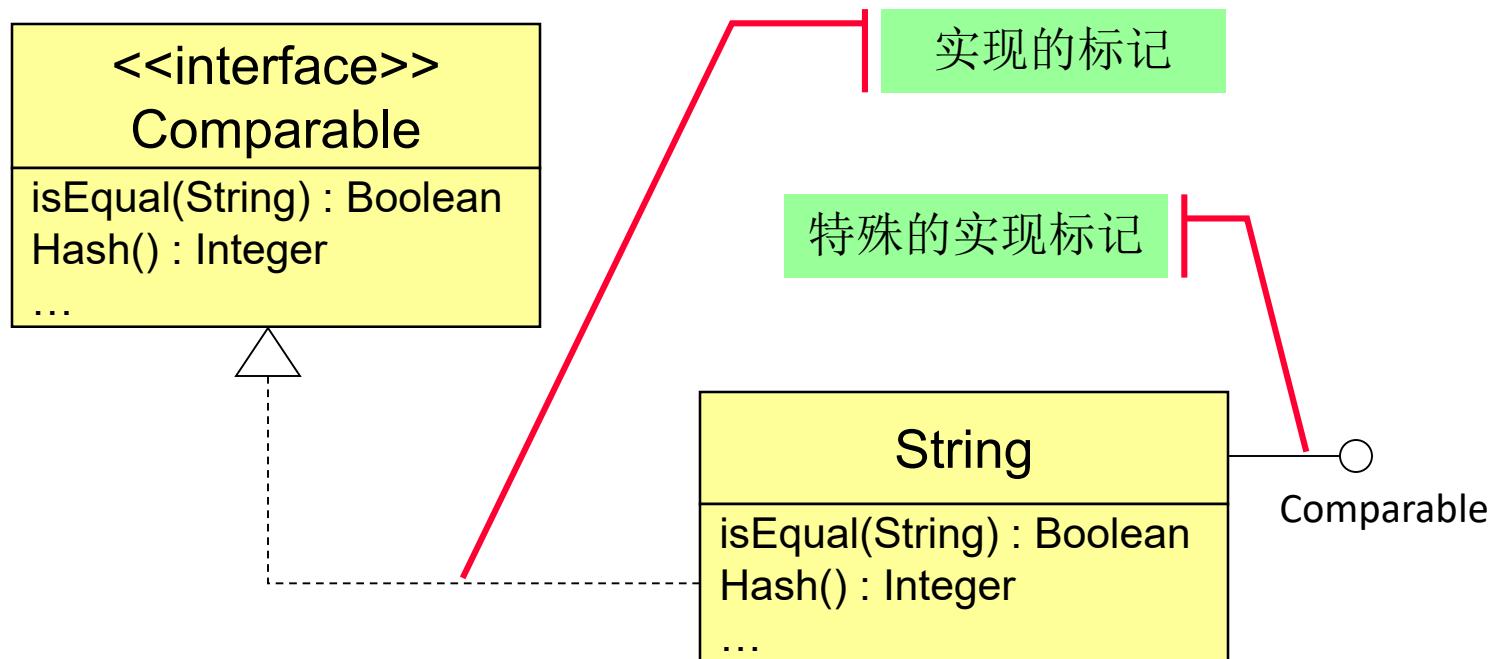


# 类图中的关系



- 实现关系 (Realization)

实现是依赖的一种，但由于它具有特殊意义，所以将它独立讲述。实现关系用于表示同一个事物的两种描述之间的关系。对同一个事物的两种描述建立在不同的抽象层上。例如：分析类和设计类之间就是一种实现关系；接口的实现；用例和实现该用例的协作。

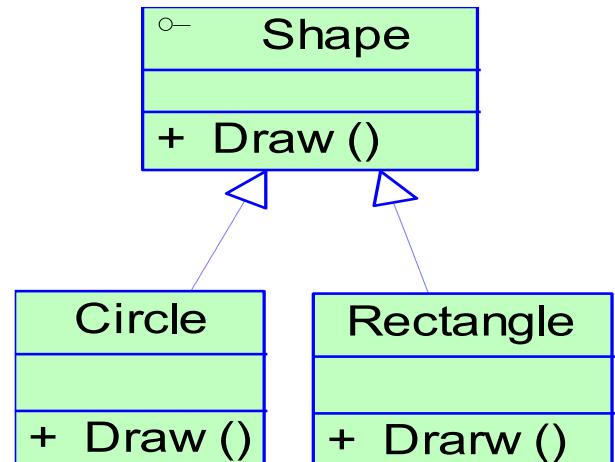


# 类图中的关系



## 实现关系

对应于类和接口之间的关系

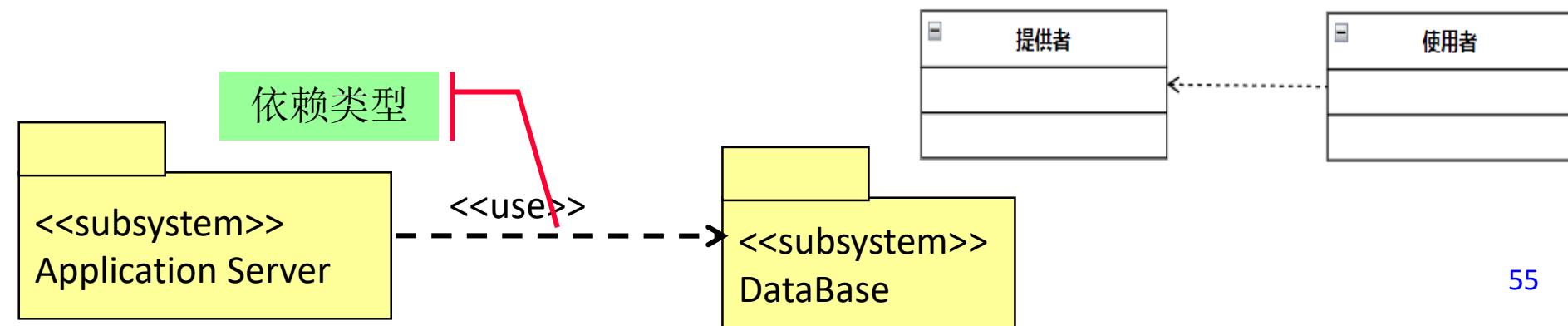


类Circle、Rectangle实现了接口Shape的操作

# 类图中的关系

- 依赖关系 (Dependency)

- 依赖指明两个或两个以上模型元素(类、组合、用例)之间的关系。一个模型元素是独立的，另一个模型元素是非独立的(依赖于独立的模型元素)。独立的模型元素的改变要影响依赖的。
  - 一个类使用另一个类的对象作为操作中的参数；
  - 一个类存取另一个类中的全局对象成员；
  - 一个类调用另一个类中的类作用域操作；
  - 一个类是另一个类的数据成员，都是依赖关系。

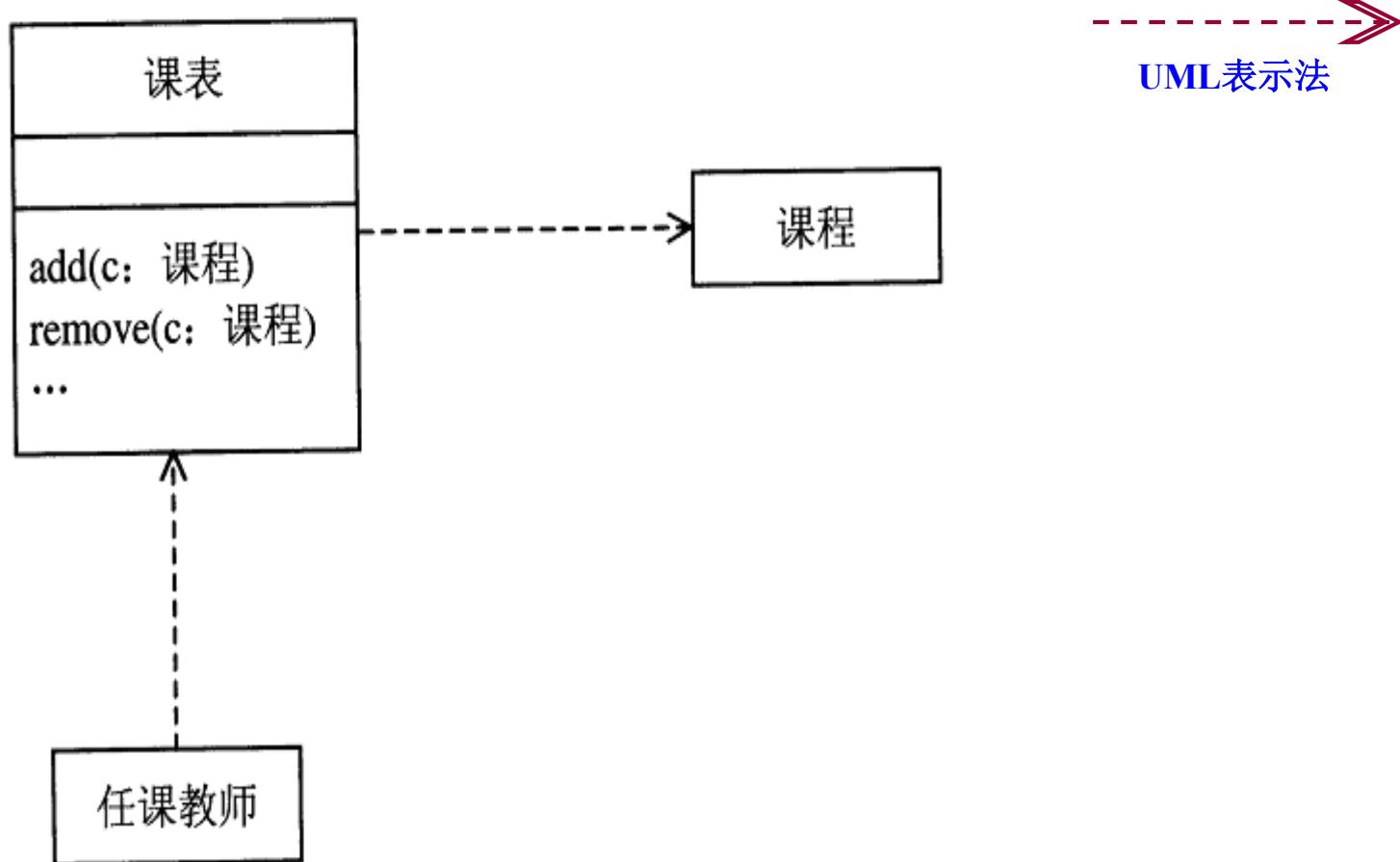


# 类图中的关系



**依赖关系：**描述了一个类的变化对依赖于它的类产生影响的情况。

依赖关系描述类之间的引用关系。





UML定义了4种基本依赖：使用依赖、抽象依赖、授权依赖和绑定依赖。

(1) **使用依赖**。使用依赖是一种非常直接的关系，它通常表示使用者使用服务提供者所提供的服务实现它的行为。

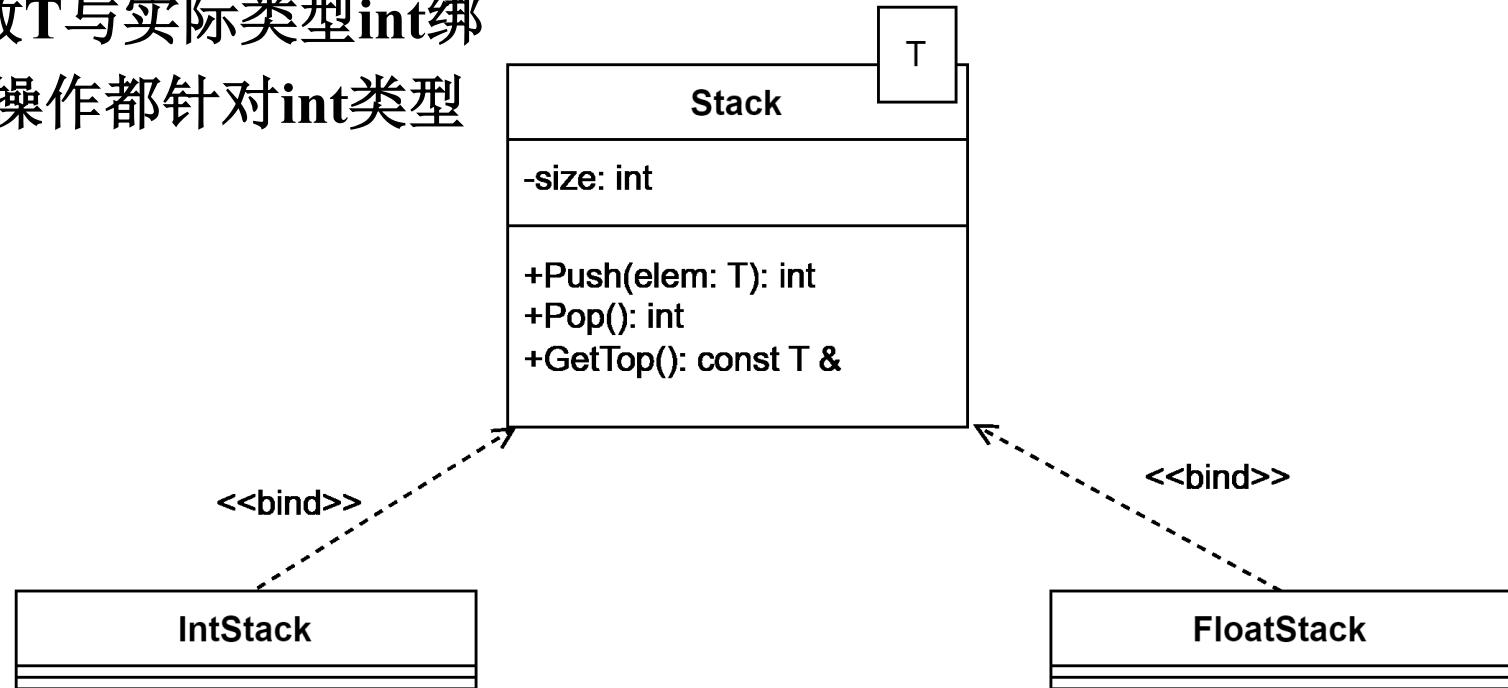
(2) **抽象依赖**。抽象依赖建模表示使用者和提供者之间的关系，它依赖于在不同抽象层次上的事物。共有3种类型的抽象依赖：跟踪依赖（《trace》）、精化依赖（《refine》）和派生依赖（《derive》）。

(3) **授权依赖**。授权依赖表达了一个事物访问另一个事物的能力。提供者可以规定使用者的权限，这是提供者控制和限制对其内容访问的方法。主要有3种类型的授权依赖：访问依赖（《access》）、导入依赖（《import》）和友元依赖（《friend》）。

(4) **绑定依赖**。它表明对目标模板使用给定的实际参数进行实例化。当对模板类的细节建模时，要使用绑定（《bind》）。例如，模板容器类和这个类的实例之间的关系被模型化为绑定依赖。绑定包括一个映射到模板的形式参数的实际参数列表



【例5-2】在如图所示的模板类  
`Stack<T>`定义了栈相关的操作；  
`IntStack`将参数T与实际类型`int`绑定，使得所有操作都针对`int`类型的数据。



依赖关系 (bind) 实现



- UML类图中常见的关系有以下几种：泛化、实现、关联、聚合、组合以及依赖。其中聚合和组合关系都是关联关系的一种。这些关系的强弱顺序不同，排序结果为：**泛化=实现>组合>聚合>关联>依赖**。

# 课堂提问

- 表示一个类与另一个类之间的依赖关系，这种关系称为（ ）。
  - A. 关联
  - B. 泛化
  - C. 聚合
  - D. 依赖

答案：D

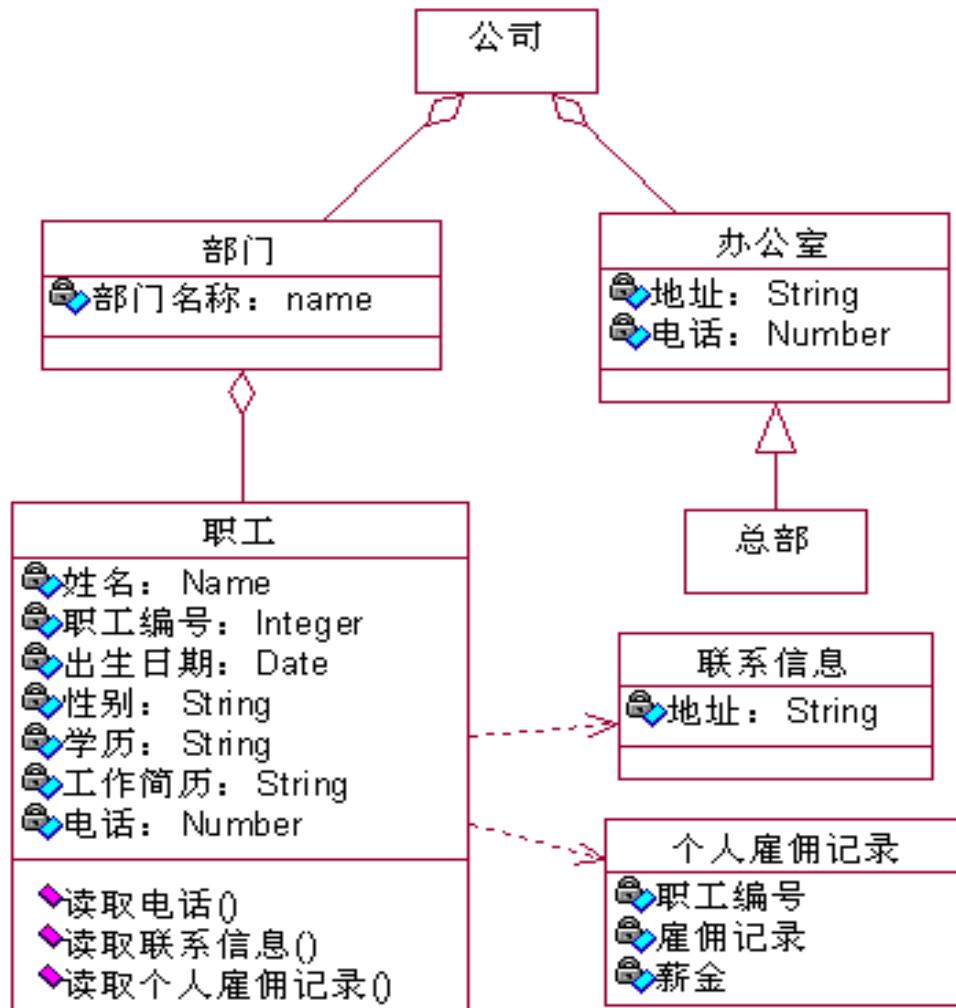
# 课堂提问

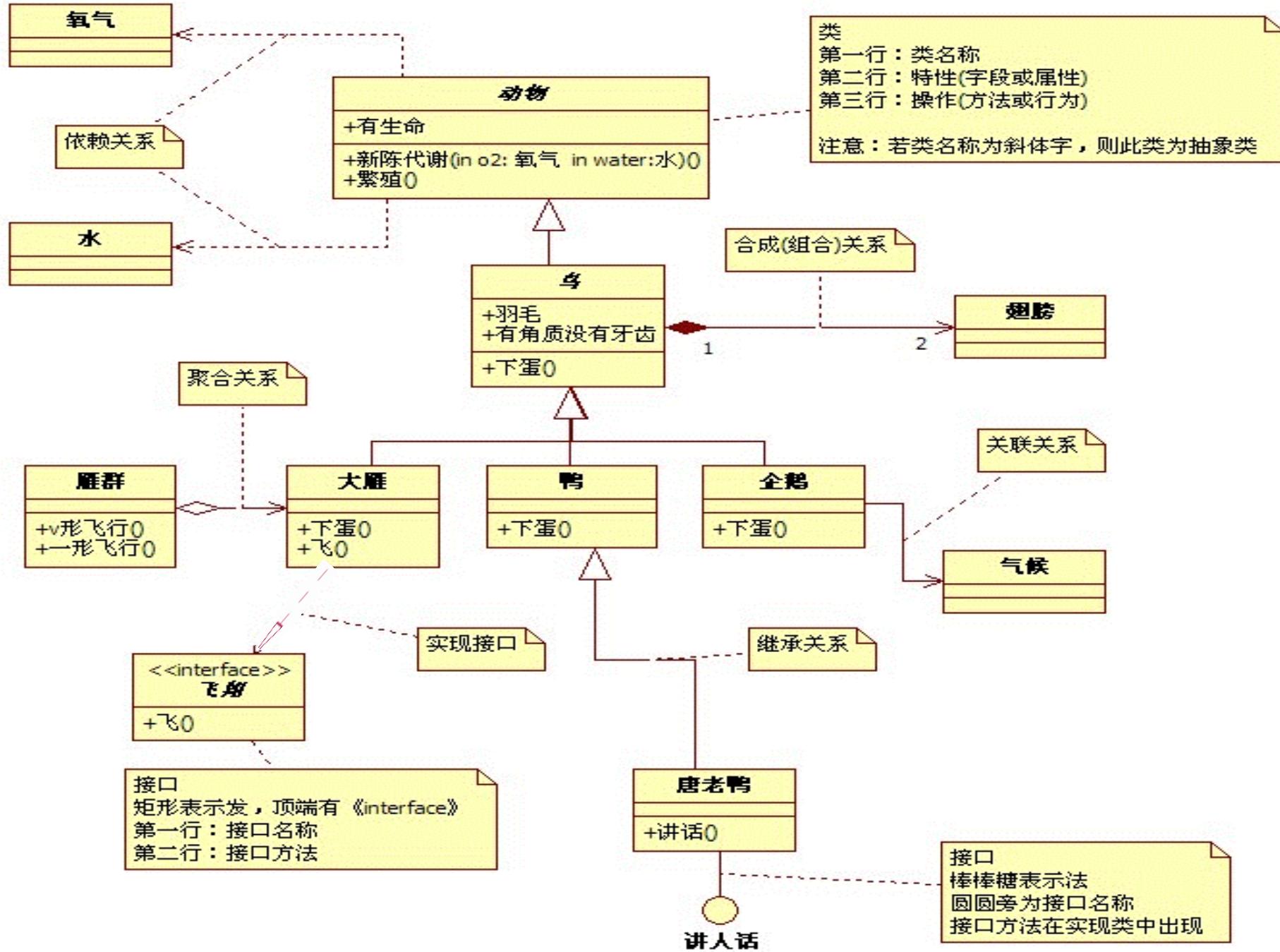
- 在**UML**中，用什么符号表示一个类的依赖关系（ ）。
  - A. 箭头
  - B. 虚线
  - C. 实线
  - D. 双线

答案： B

# 练习：阅读类图

## 公司系统类图





# 类图的建模技术及应用



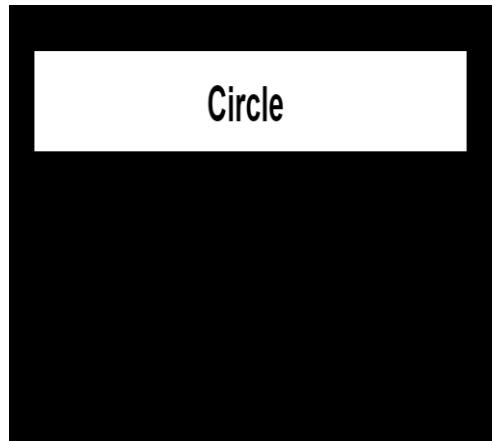
- 在软件开发不同阶段使用具有不同的**抽象层次**的类图，即**概念层**、**说明层**和**实现层**。
- 在**UML**中，从开始的需求分析到最终的设计类，类图也是围绕着这三个层次的观点来进行建模的。
- 类图建模是先建立**概念层**到**说明层**，进而到**实现层**，随着抽象层次的逐步降低并逐步细化的过程。



- 概念层类图

概念层的类图描述的是现实世界中对问题领域的概念理解，类图中表达的类与现实世界的问题领域有着明显的对应关系，类之间的关系也与问题领域中实际事物的关系有着明显的对应关系。在概念层类图阶段很少考虑或者几乎不需要考虑类的实现问题。

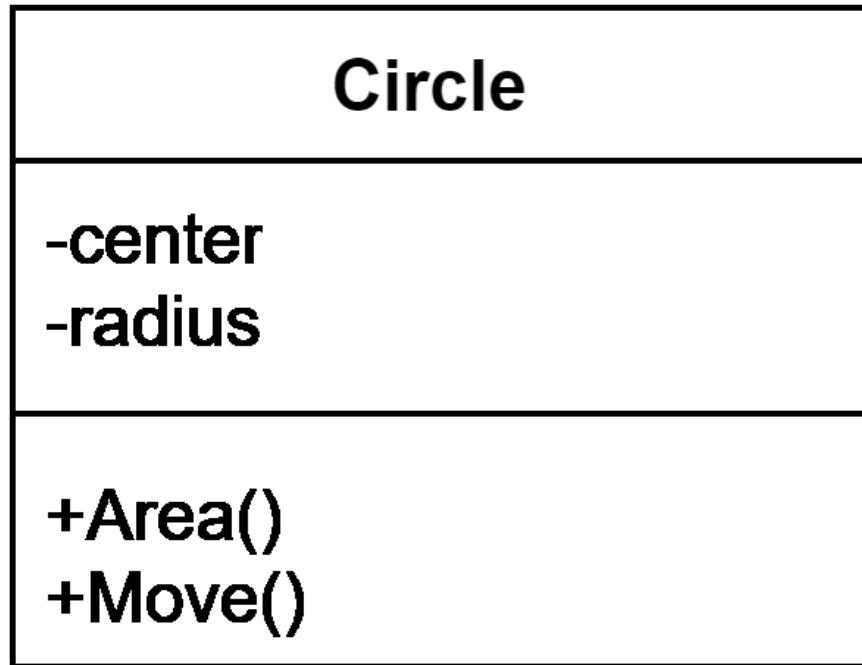
概念层类图中的类和类关系和最终的实现类并不一定有直接和明显的对应关系，在概念层上，类图着重于对问题领域的概念化理解，而不是实现。因此，类名通常都是问题领域中实际事物的名称，并且独立于具体的编程语言。例如，圆形类的概念层类图表示如图所示。





- 说明层类图

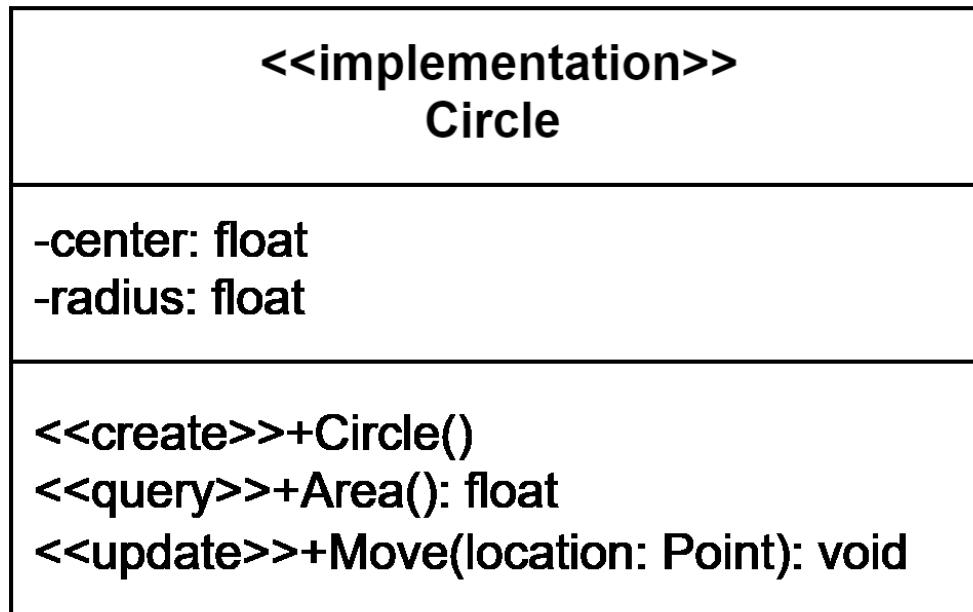
在说明层阶段主要考虑的是类的接口部分，而不是实现部分。这个接口可能因为实现环境、运行特性等有多种不同的实现。





## • 实现层类图

真正需要考虑类的实现问题是在实现层类图阶段。提供实现的细节，在实现层阶段的类的概念才是真正严格意义上的类。它揭示了软件实体的构成情况。说明层的类有助于人们对软件的理解，而实现层的类是最常用的。





使用UML对系统进行建模时，最终的目标是识别出系统中所有必须的类，并分析这些类之间的关系，类的识别贯穿于整个建模过程，分析阶段主要识别问题域相关的类，在设计阶段需要加入一些反映设计思想、方法的类以及实现问题域所需要的类，在编码实现阶段，因为语言的特点，可能需要加入一些其他的类。

其中，建立类图的步骤如下：

- (1) 研究分析问题领域，对系统进行需求分析，确定系统需求。
- (2) 确定系统中的类，明确类的含义和职责，以及确定类型的属性和操作。
- (3) 确定类之间的关系。

# 类的识别方法

对系统进行建模时，对类的识别是一个需要大量技巧的工作，寻找类的一些方法包括：

- 名词识别法；
- 从用例中识别类；
- 使用CRC分析法；
- 领域分析法。

## 名词识别法

名词识别法的关键是识别系统问题域中的实体。对系统进行描述，描述应该使用问题域中的概念和命名，从系统描述中标识名词及名词短语，其中的名词往往可以标识为对象，复数名词往往可以标识为类。

## 从用例中识别类

用例图是对系统进行需求分析建立的图形描述，实质上是一种系统描述的形式，自然可以根据用例描述来识别类。针对各个用例，可以提如下的问题来帮助我们识别系统中的类：

用例描述中有哪些实体？

用例的完成需要哪些实体进行合作？

用例执行过程中会产生并存储了什么信息？

用例要求与之关联的每个角色的输入是什么？

用例反馈与之关联的每个角色的输出是什么？

用例需要操作哪些硬设备？

## 从用例中识别类

CRC（Class， Responsibilities， Collaboration）卡的最大价值在于把人们从思考过程模式中脱离出来，更充分的专注于对象技术。CRC允许整个项目组的人员对系统的设计做出贡献。参与系统设计的人越多，能够收集到的信息也就越多。在CRC会议中，一些人模拟系统和对象交流，把消息传给其他的对象，通过一步步处理，问题很容易就被解决。

CRC卡由三部分组成：类（Class）、职责（Responsibility）和协作（Collaborator）。



## 从用例中识别类

建立类图的过程就是对领域及其解决方案的分析和设计的过程。获取类也是依赖于每个人对领域的了解和理解的过程，有时需要和所涉及领域的专家进行合作，对所要研究的领域进行仔細分析，抽象出领域中的概念、定义、含义及其相互的关系，分析出系统类，并用领域中的术语为类进行命名。因此，领域分析法是：通过对某一领域中的已有应用系统、理论、技术、开发历史等的分析和研究，来标识、收集、组织和表示领域模型及软件体系结构的过程，并得到最终的结果。

# 成绩管理系统的类图建模实例

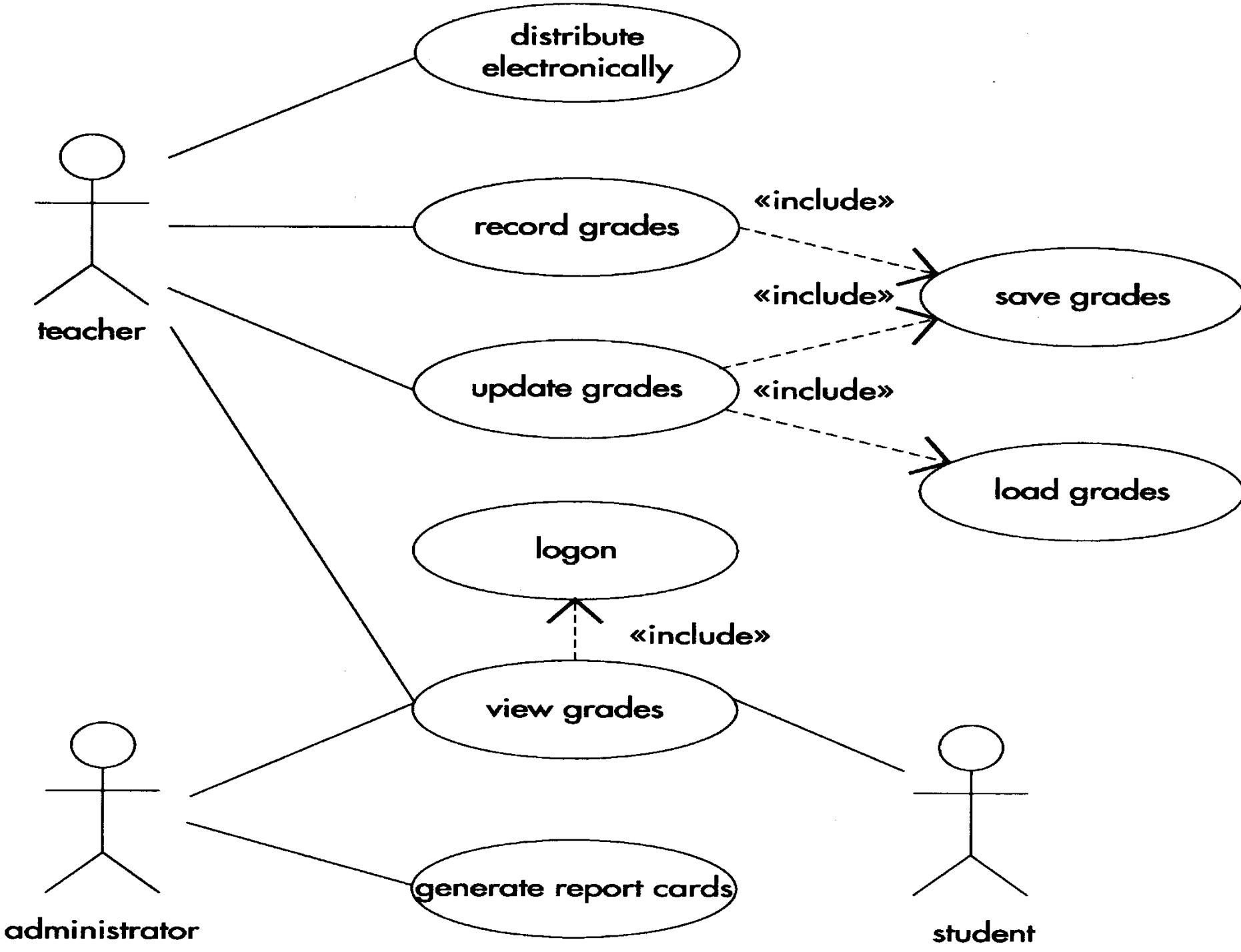


下面以学生管理系统为例，利用第2种方法从用例图的角度来进行系统类图的建模。

创建类图需要两个反复执行的步骤：

- 1) 确定类及其关联。
- 2) 确定属性和操作。

开始创建类图的起点就是用例图。如下面成绩管理的用例图所示。





## 1. 确定类和关联

首先要做的是通过分析用例图确定类及其关联。找到第一批类，确定它们的内容。

在用例图中，首先确定了Grades类和ReportCard类。接下来，通过同时使用参与者名称确定附加的类。这时将会确定Teacher类，Student类和Administrator类。

下面检查用例图并且确定各个功能所属的类：

发布报告卡—Grades类

记录分数—Grades类

更新分数—Grades类

保存分数—Grades类

加载分数—Grades类

登录—？

查看分数—Grades类

生成报告卡—ReportCard类

首先发现的是登录没有所属的类。可以添加一个Logon类来处理Logon用例。



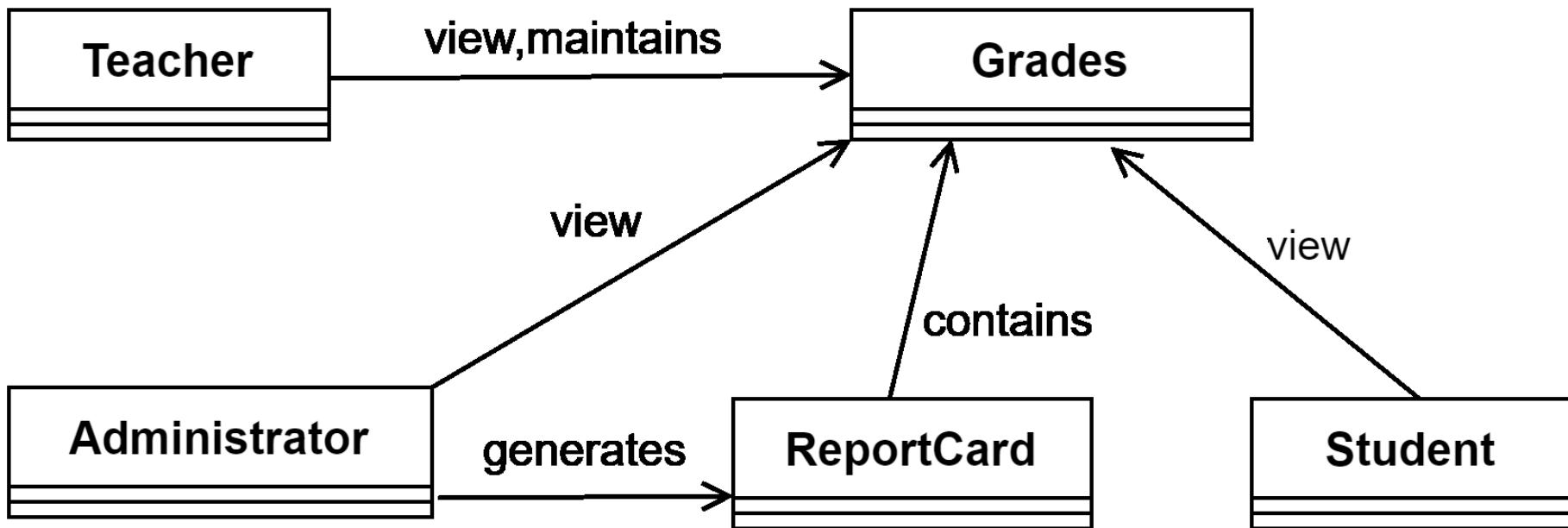
现在可以开始创建类的关联：

Teacher记录、更新、查看 Grades

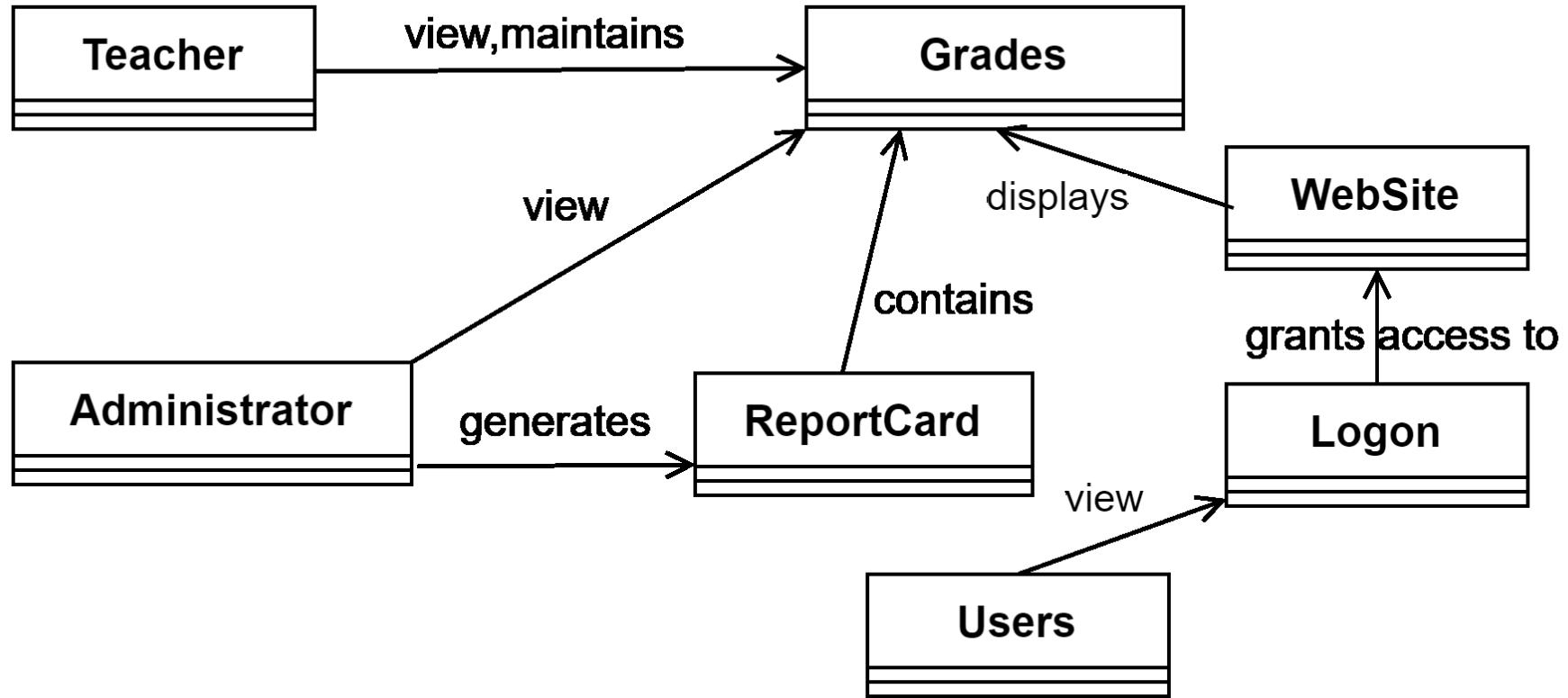
Administrator查看 Grades、生成ReportCards

Student查看Grades

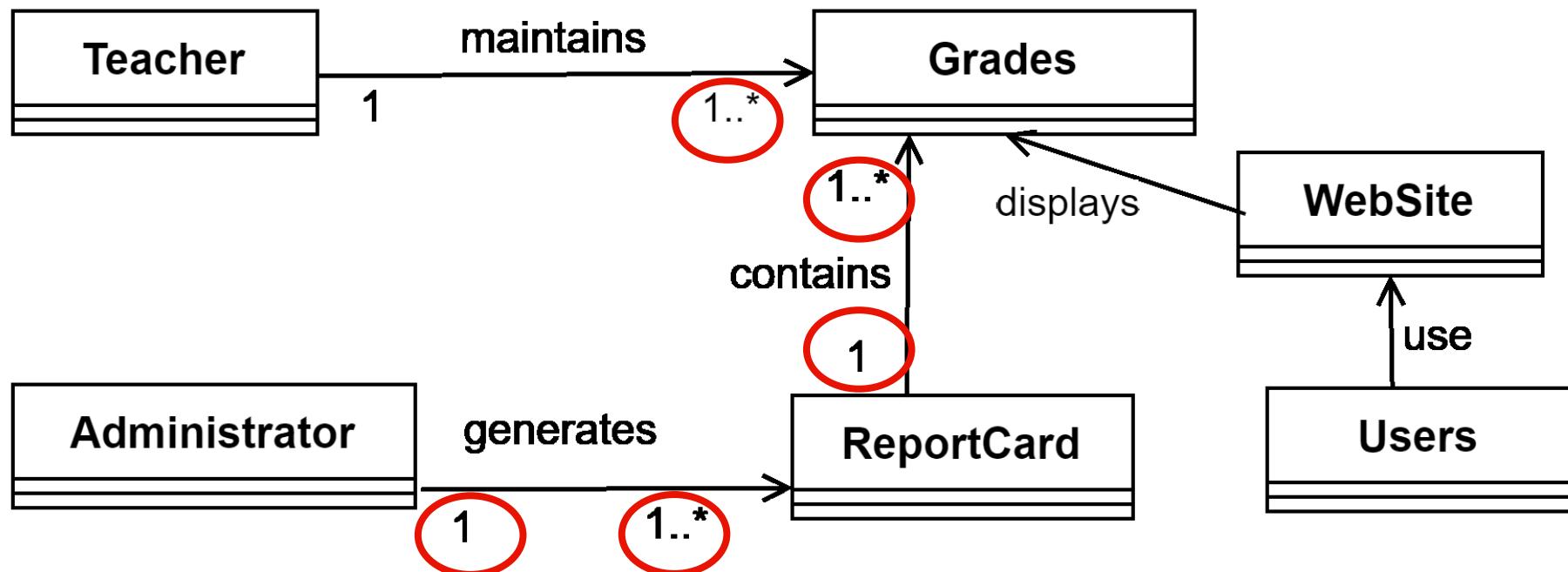
ReportCards包含Grades



## 进一步创建类的关联：增添WebSite类和Logon类



下一步通过添加**多重性**让类图的信息更加详细，并且对类图进行调整以便保证没有冗余的类和关联。

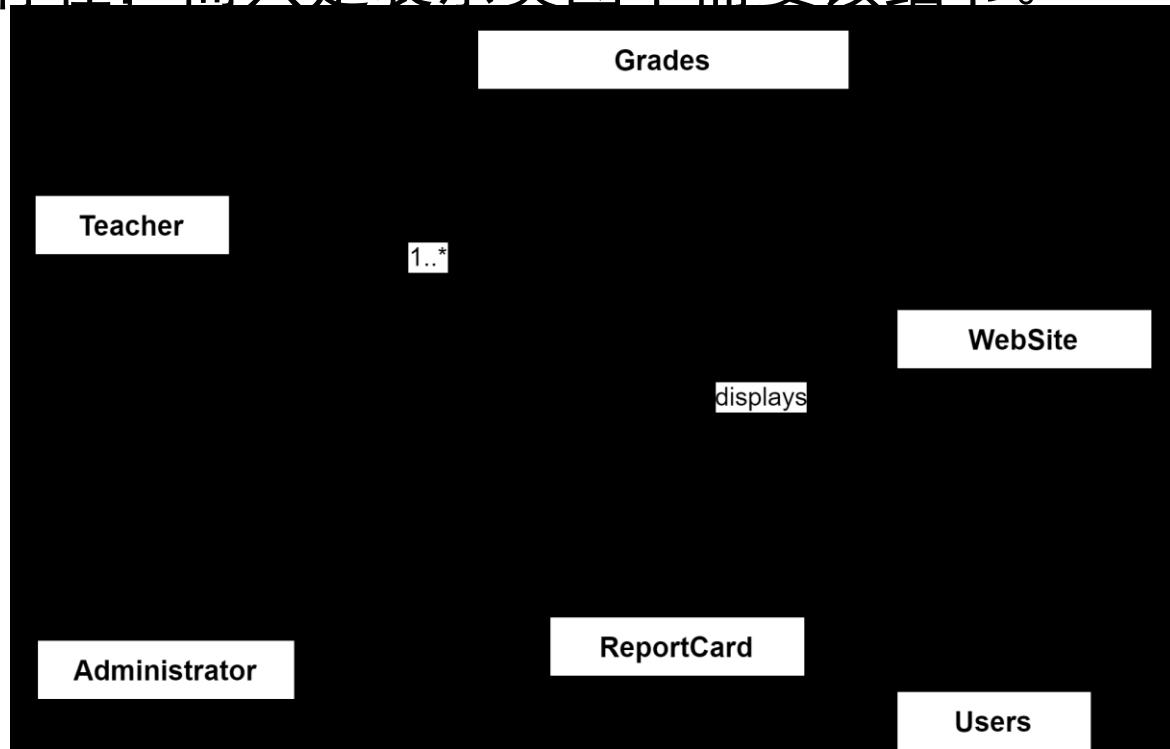




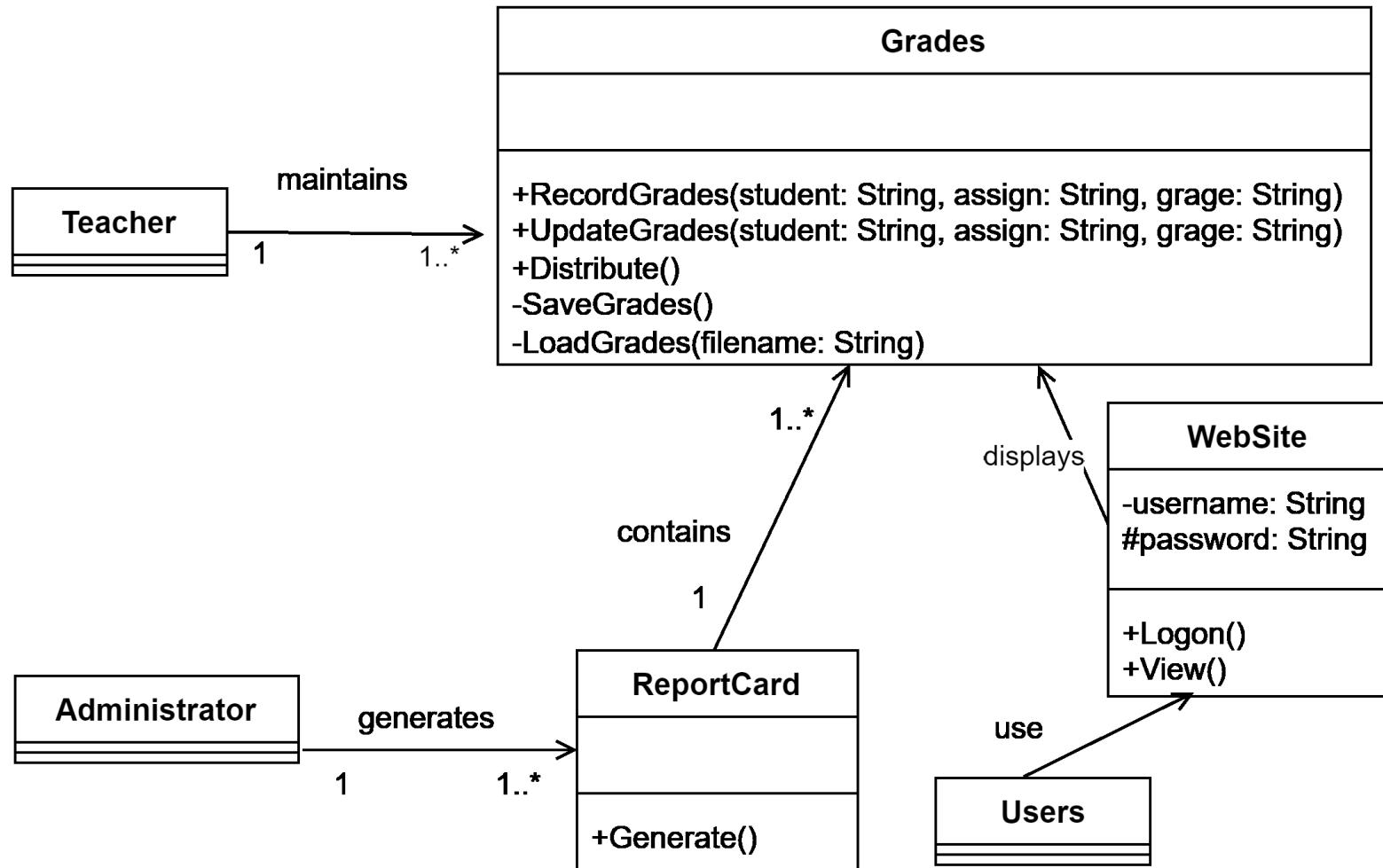
## 2. 确定属性和操作

现在我们已经创建好了类和关联，可以开始添加属性和操作以便提供数据存储和需要的功能来完成系统功能。

在下图中可以看到，表示参与者的类没有显示属性和操作，这并不意味着它们不存在，而只是表示类图不需要该细节。



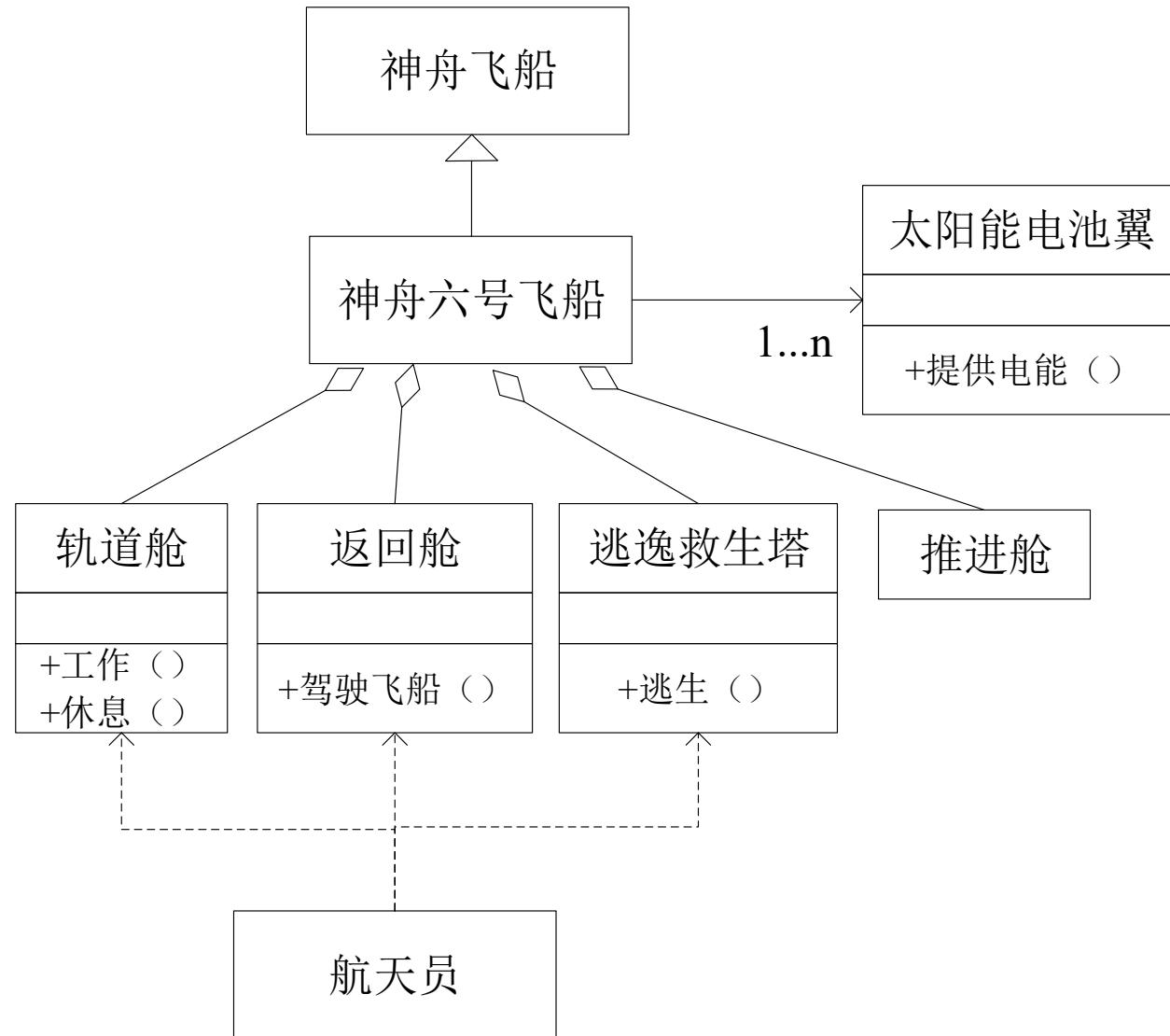
最后，为属性和操作提供参数、数据类型和初始值。如下图所示：



# 课堂练习

请根据以下描述画出该系统的类设计图。

神州六号飞船是神州飞船系列的一种，它由轨道舱、返回舱、推进舱和逃逸救生塔等组成。航天员可以在返回舱内驾驶飞船，轨道舱则是航天员工作和休息的场所。在紧急情况下，可以利用逃逸救生塔逃生。在飞船两侧有多个太阳能电池翼，可以为飞船提供电能。



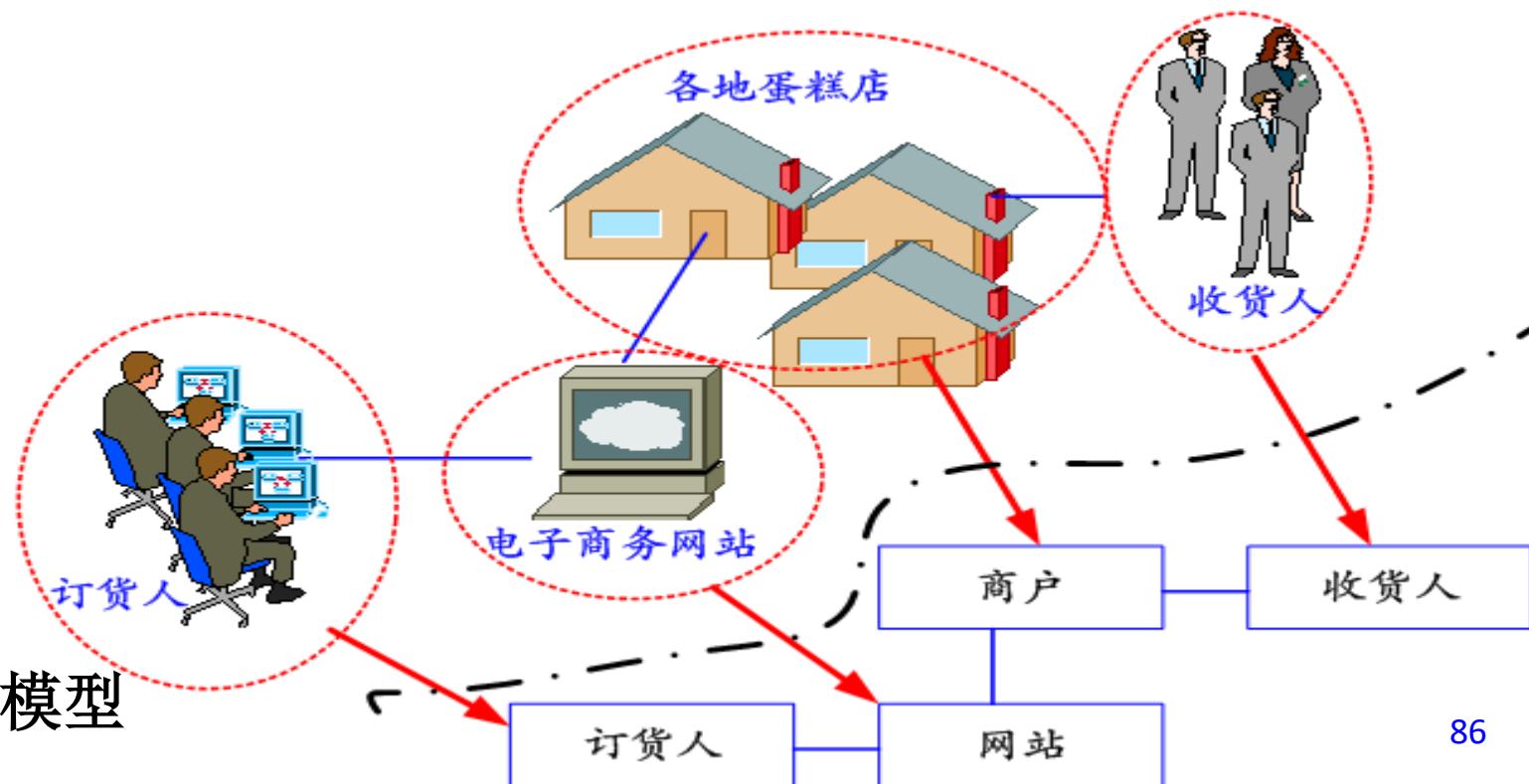
# 电子商务网站的类图建模实例

- 问题陈述。
  - 假设住在厦门的张三要给住在绍兴的朋友李四送一个生日蛋糕，由于它们之间的距离太远，不可能亲自买一个送过去。但解决这个问题并不难，张三登录到一个电子商务网站购买一个，并通过该网站将其送给李四。而这个电子商务网站实际上就是通过绍兴的蛋糕店来完成这个任务的。



- 张三就是一个对象，它可以归位“订货人”这个类中；而绍兴蛋糕店显然也是一个对象，它可以归到“商户”这个类中。
- 例如在上面的例子中，订货人把想完成的事（给李四送蛋糕）委托给电子商务网站，而电子商务网站又委托给具体的商户（绍兴蛋糕店），具体的商户最终通过送货人完成了这个行为。

- 那么在实际的过程中，整个行为是怎么动起来的呢？张三在电子商务网站填写信息，电子商务平台向商户发信息，商户指挥送货人完成该动作。

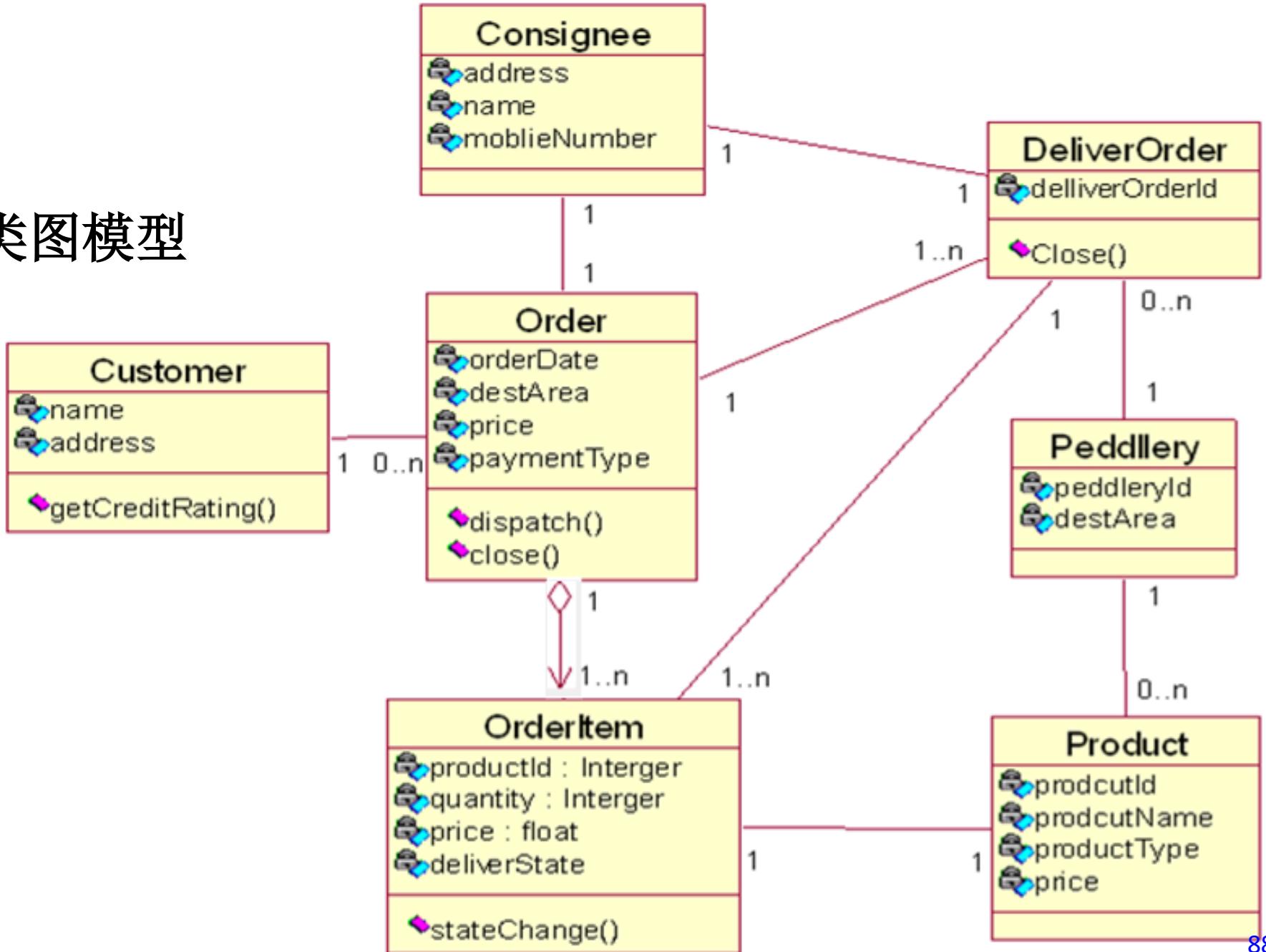


- 建立对象模型



- 发现类
  - 该图包含7个类：**order**（订单），**orderitem**（订单项），**customer**（顾客），**consignee**（收货人），**deliverorder**（送货单），**peddler**（商户），**product**（产品），并且每个类都定义了若干属性和方法。

# 类图模型

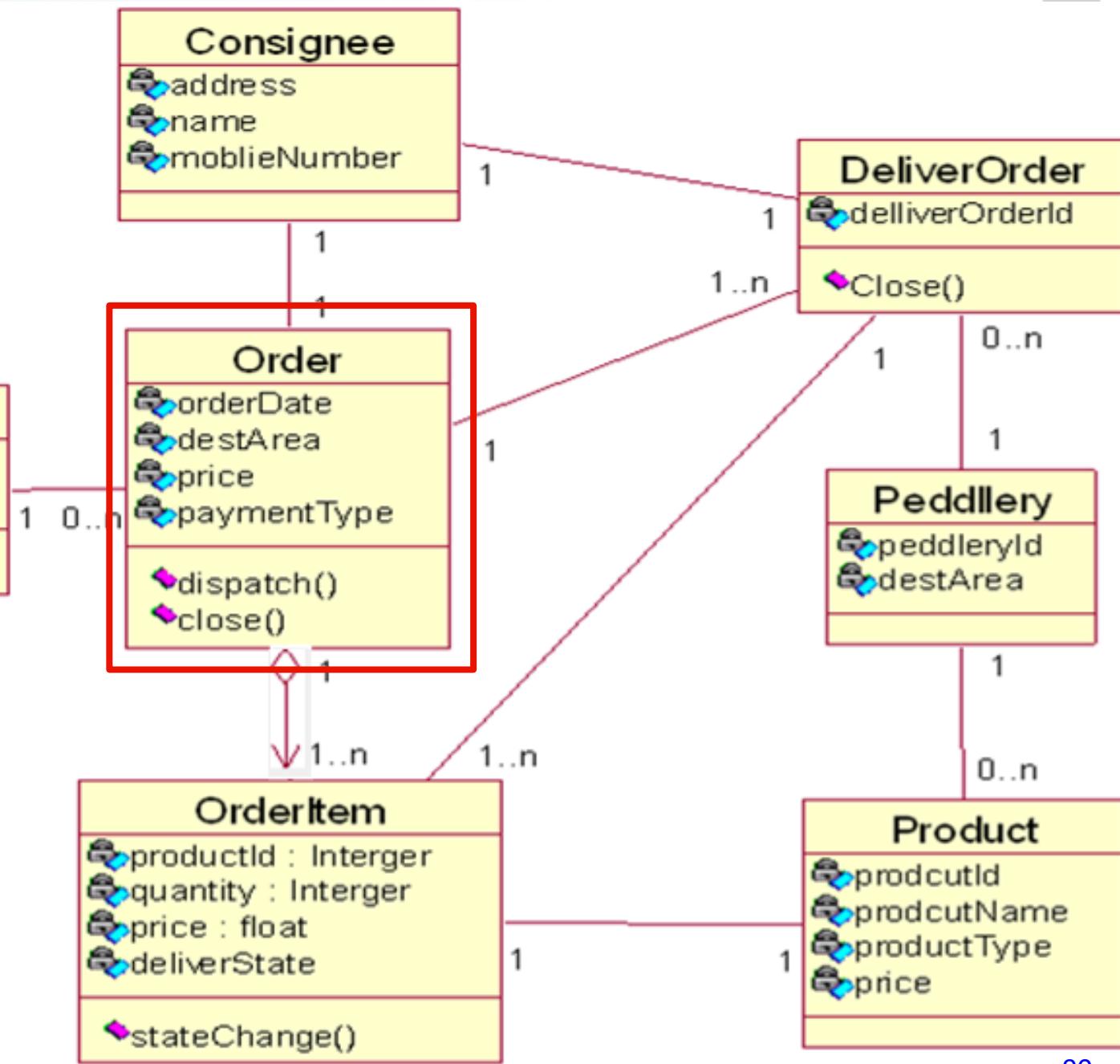




- 分析类关系

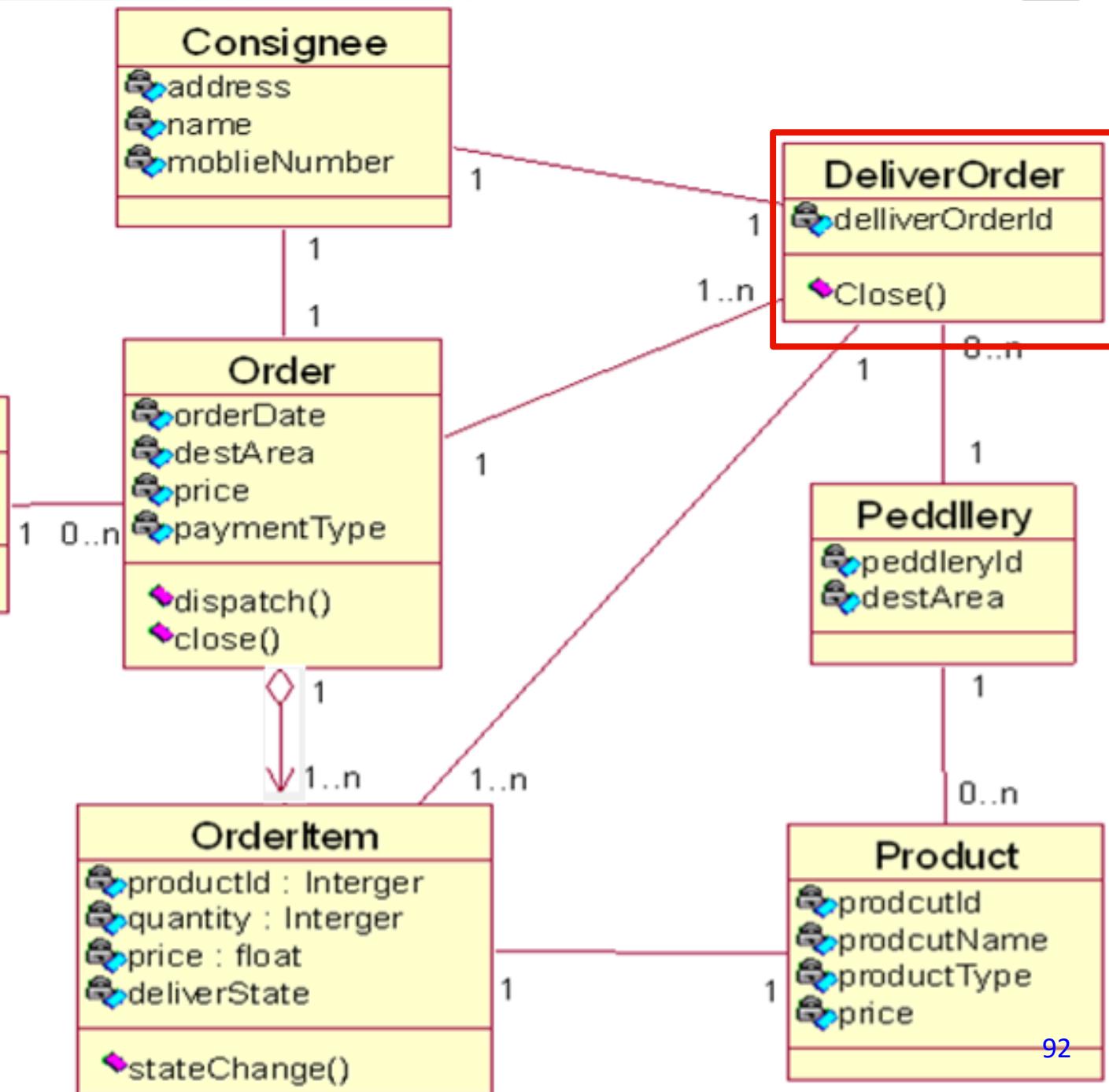
- 关系包含关联（包括聚合、组合两种）、泛化、实现、依赖四种。
- 阅读类图时，从图中最复杂（核心）的类开始阅读。在本例中，这个类就是**order** (订单)。然后逐一地分析该类与其他类之间的关系
- **order**与**orderItem**之间是组合关系，根据箭头的方向可知是**orderItem**组合为**order**。这就是说**order**包含了**orderItem**。显然对于该应用系统而言，独立的**orderItem**是没有意义的。
- **order**和**customer**、**consignee**、**deliverorder**是关联关系，即一个订单和客户、收货人、送货单是相关的。

## 类图模型



- 第二复杂关系的类就是**deliverorder(送货单)**，和它相关的也有4个类：**order**、**orderItem**、**congsinee**、**peddlery**，即表示送货单与订单是相关的，同时还关联到订单项。另外，它与商户、收货人的关联关系也是很显然的。
- 分析完这两个类之后，会发现图中的几乎所有的关系都已经明了，只剩下一个**product（产品）**，与它关联的类是**peddlery**和**orderItem**，显然产品属于某个商户的，订单项中必须指出是哪种商品。

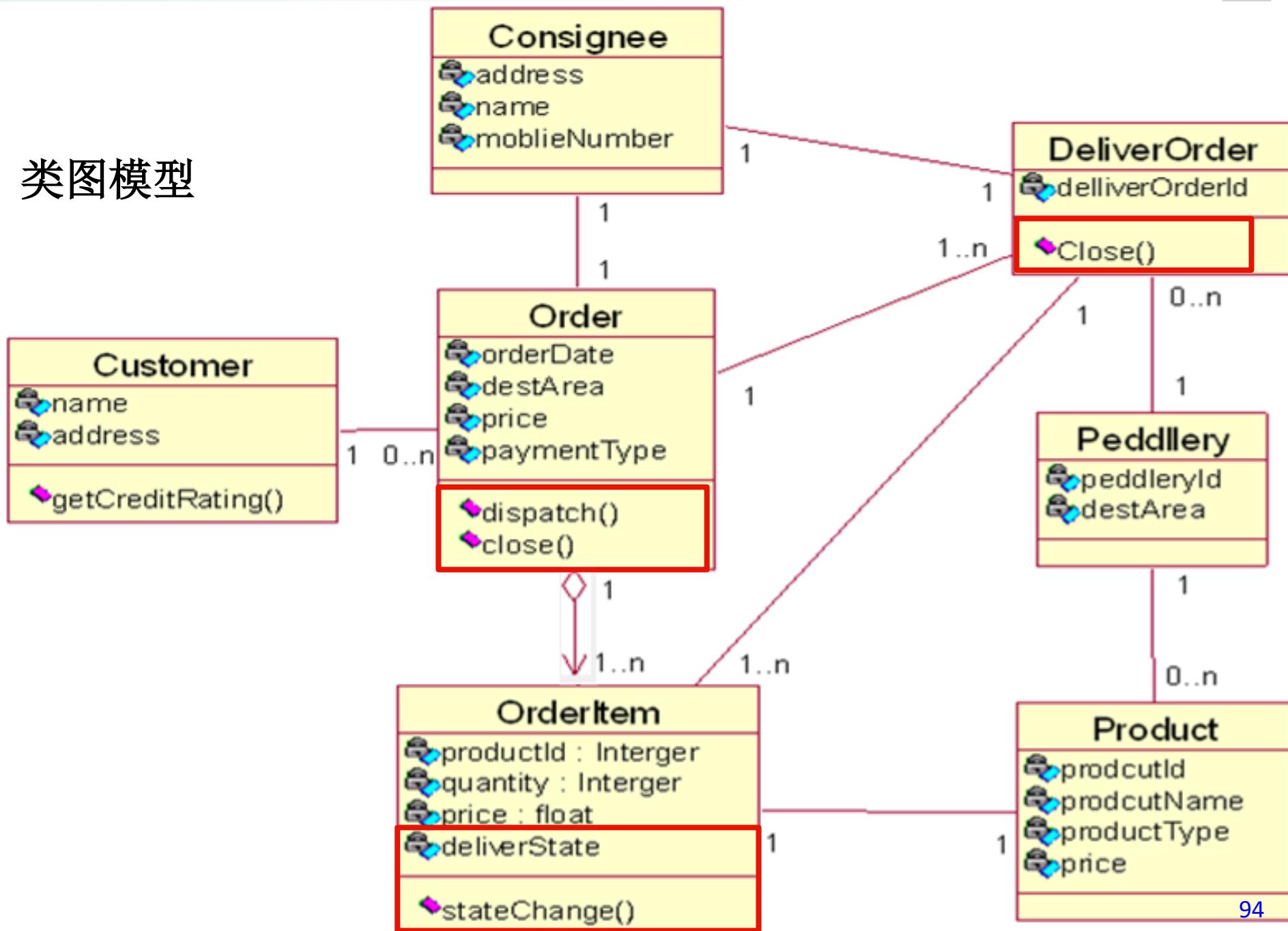
## 类图模型





- 分析关联的多重性
  - 多重性用来说明关联的两个类之间的数量关系
  - **order**类包含了两个方法：**dispatch()**和**close()**，它们分别实现“分拆订单生成送货单”和“完成订单”，而在**deliveorder**类中有一个**close()**方法，它表示“完成订单”。在**orderitem**中有一个**statechange()**方法和**deliverstate**，它们是用来改变其“是否交给收货人”标志位的。

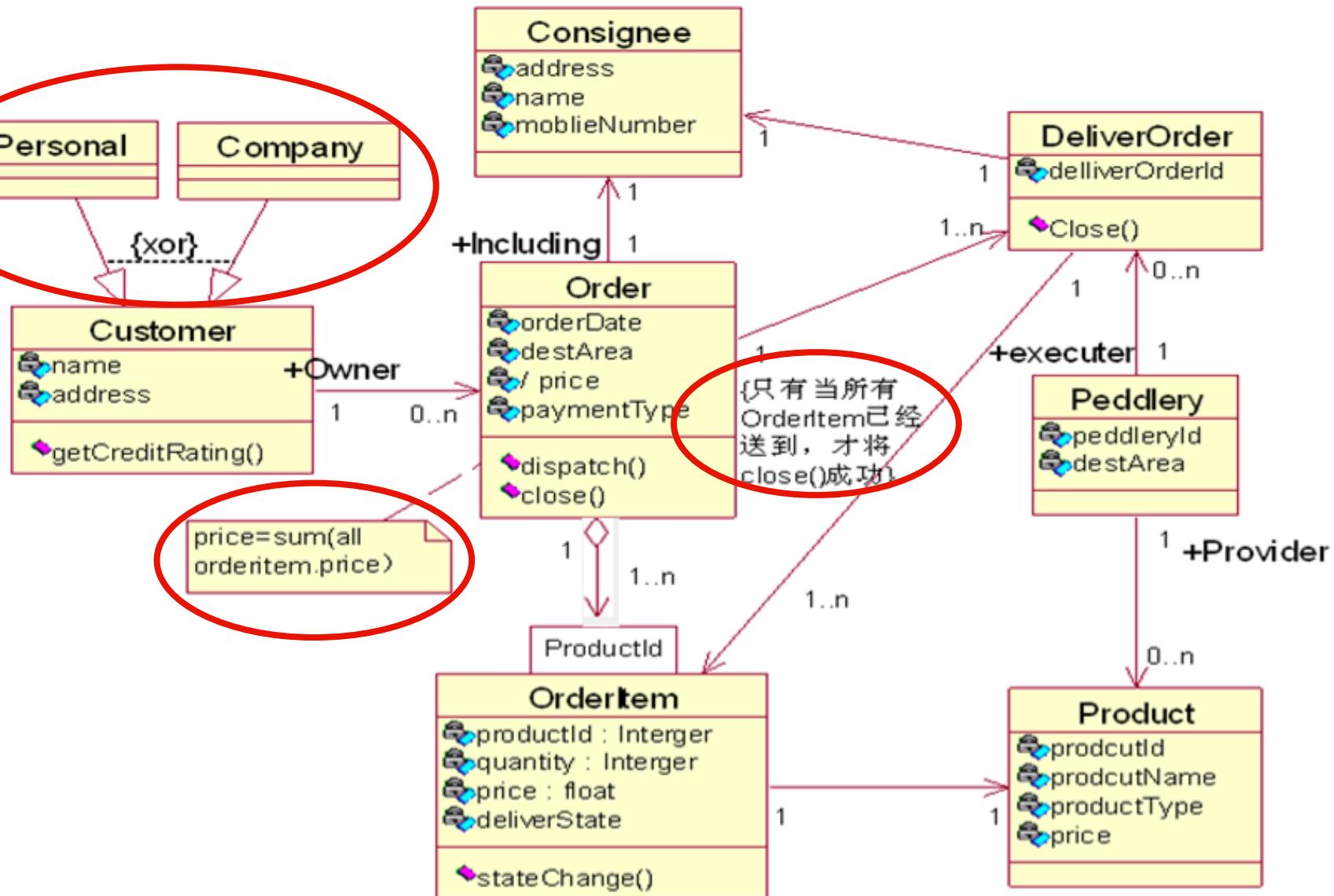
## 类图模型



- 先调用**order**的**dispatch()**方法，它将根据包含的**orderItem**中的产品信息来按供应商分拆成若干个**deliverorder**。商户登录系统后即可获取**deliverorder**，并执行完后调用**close()**方法。这时，将调用**orderItem**的**statechange()**方法来改变其状态。同时，再调用**order**的**close**方法。判断该**order**的所有**ordeltem**是否都送到了，如果是就将其**close ()**掉。

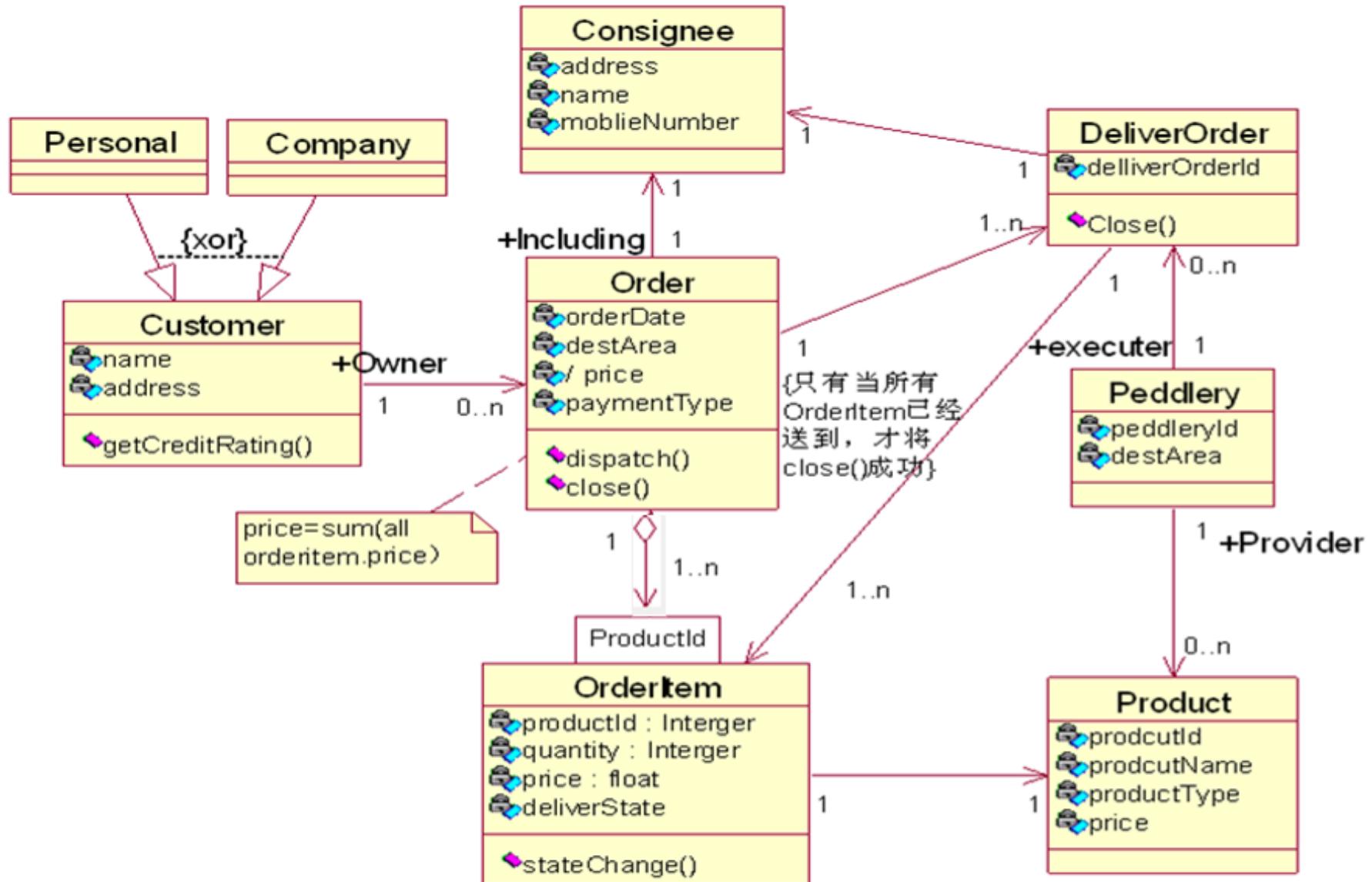
源类	目标类	关联分析	
Customer(1)	Order(0...n)	订单是属于某个客户的，网站的客户可以有0个或多个订单	
Order(1)	Consignee(1)	每个订单只能够有一个收货人	
Order(1)	OrderItem(1...n)	订单是由订单项组成的，至少要有一个订单项，最多可以有n个	
Order(1)	DeliverOrder(1...n)	一个订单有一个或多个送货单	说明：系统根据订单项的产品所属的商户，将其分发给商户，拆成了多个送货单！
DeliverOrder(1)	OrderItem(1...n)	一张送货单对应订单中的一到多个订单项	
DeliverOrder(1)	Consignee(1)	每张送货单都对应着一个收货人	
Peddlery(1)	DeliverOrder(0...n)	每个商户可以有相关的0个或多个送货单	
OrderItem(1)	Product(1)	每个订单项中都包含着唯一的一个产品	
Peddlery(1)	Prodcut(0...n)	产品是属于某个商户的，可以注册0到多个产品	

- 类图精化：除了引入personal和company两个类之外，还增加了关联属性、约束、注释。





- 关联的导航性
  - 关联进一步具体化为导航箭头。对象的消息只能沿着导航箭头的方向发送给另一对象。例如，可以从对象**order**向对象**consignee**导航，但不能从**consignee**向**Order**导航。通俗的说就是在**Order**中可以获取其相应的**Consignee**，而从**Consignee**不能了解**Order**。如果没有箭头，则表示是双向关联的，也就是等价于两端都有箭头，即可互相导航。





- 关联的角色名称
  - 在**customer**和**order**的关联中，**customer**端有一个“**+ owner**”字符串；在**order**到**consignee**; **peddery**到**deliverorder**再到**product**之间也有类似的字符串。这些字符串称为角色名称，表示**customer**是**order**的所有者（**owner**），**order**包含（**including**）了**consignee**, **peddlery**是**product**的提供者，是**deliverorder**的执行者。通过理解这些角色名称，使得类之间的关联关系更加具体化。
- 关联的限定符
  - 在**order**和**orderItem**之间的关系中，**orderItem**端多了一个方框，里面写着**productId**，它在**UML**中称为限定符。存在限定符的关联称为受限关联，用来表示某种限定关系。在本例中，它的用途是说明对于一张订单，每种产品只能有一个订单项。

Personal

Company

Consignee

address  
name  
mobileNumber

+Including 1

{xor}

Customer

name  
address

getCreditRating()

+Owner

1 0..n

Order

orderDate  
destArea  
/ price  
paymentType

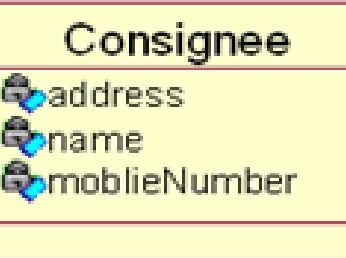
dispatch()  
close()

ProductId

OrderItem

productId : Integer  
quantity : Integer  
price : float  
deliverState

stateChange()



DeliverOrder

delliverOrderId

Close()

Peddler

peddlerId  
destArea

+Provide

0..n

Product

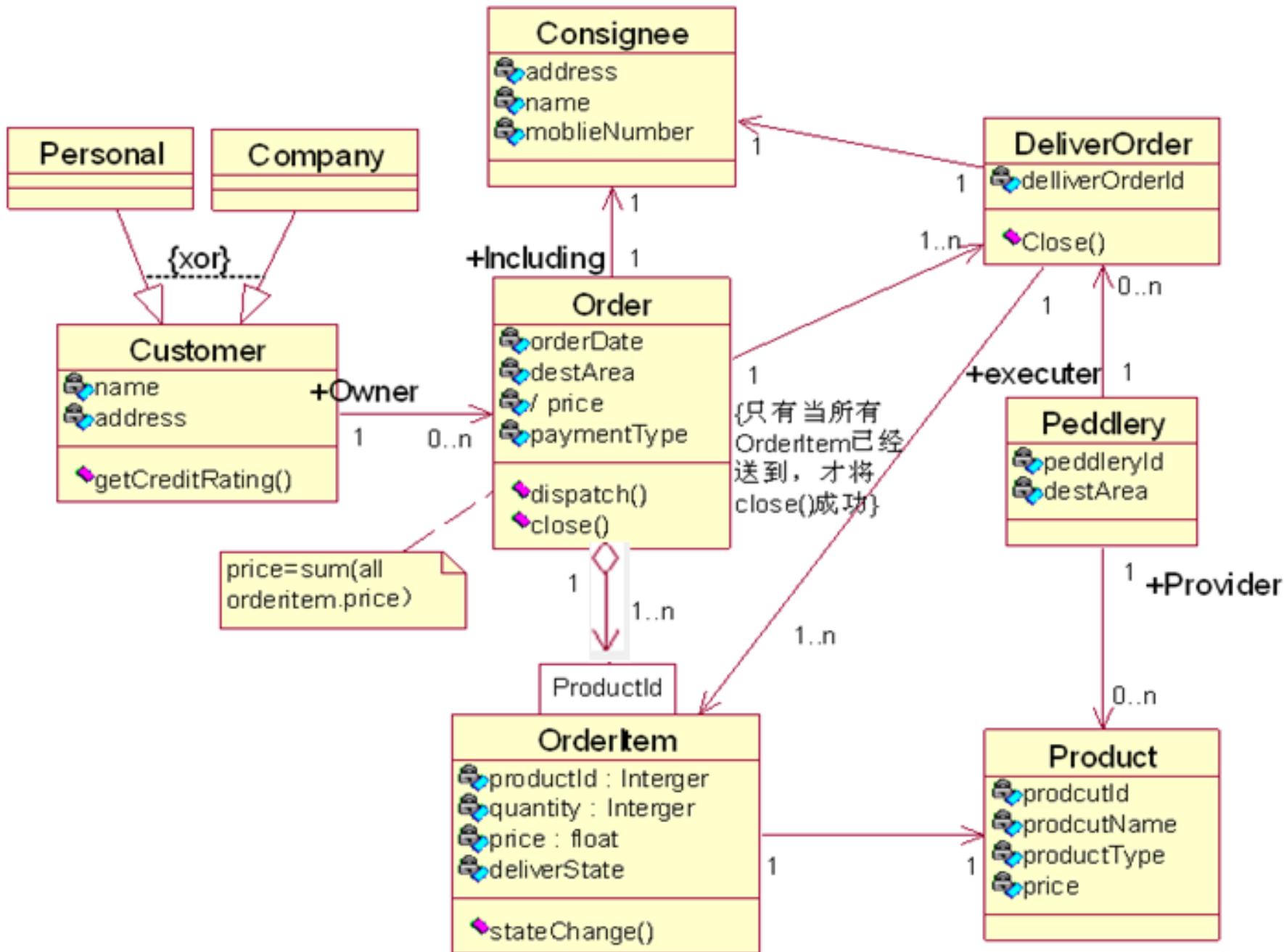
prodctId  
prodctName  
productType  
price

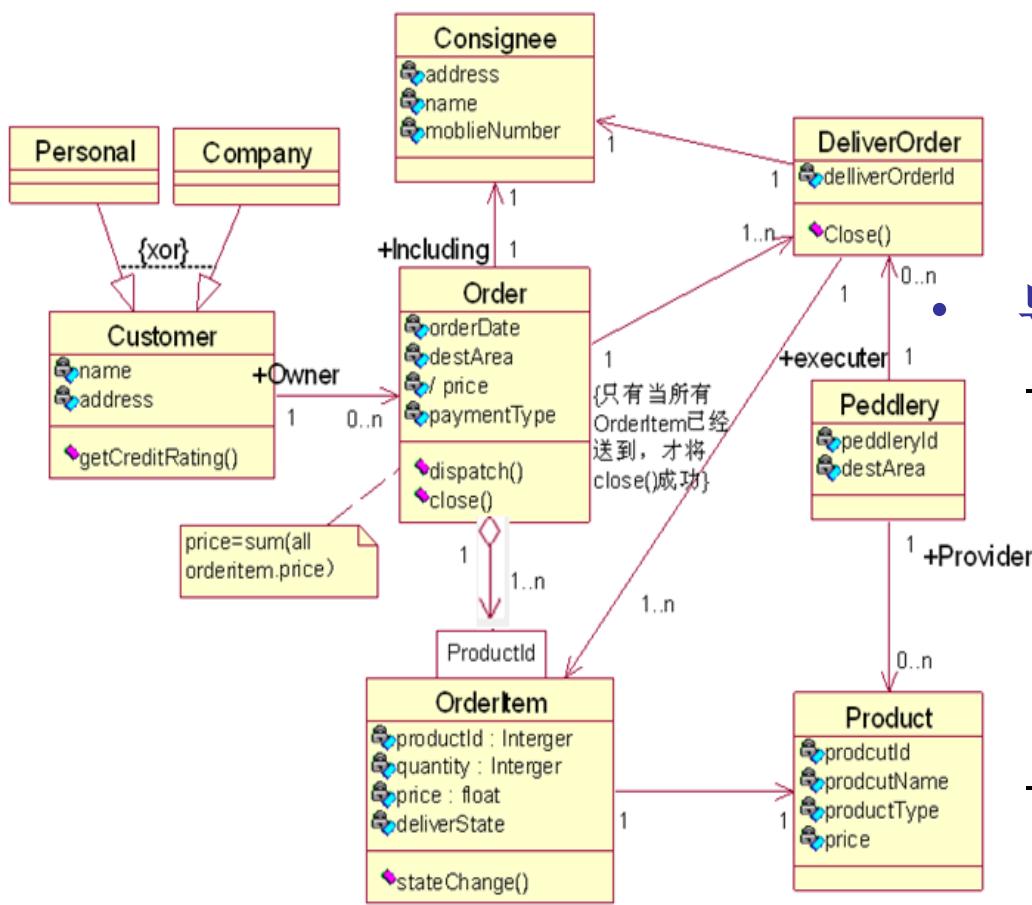


- 约束

- 在**order**类的边上有一个用大括号括起的文本，里面写着“**只有当所有的**orderitem ()** 已经送到，才将**close ()** 成功**”，而在**persongal**和**company**类当中则有一根虚线，上面写着“**{xor}**”。
- 在**UML**类图中。这种以大括号括起来的、放在建模元素外边的字符就是约束。约束可以**自由文本**和**OCL**两种形式表示。在本例中，这两个都是自由文本。

- 前一个约束显然是表示一种规则，也就是说每送完一个**deliverorder**，就会将其所包含的**orderitem**的**deliverstate**修改为**true**；而对于**order**，调用**close()**成功的前提是它所包含的所有**orderitem**的**deliverstate**的值都是**true**才行。
- 而后一种约束则是一种关联间的约束，它表示一个**customer**要么是**personal(个人用户)**，要么是**company(公司客户)**。



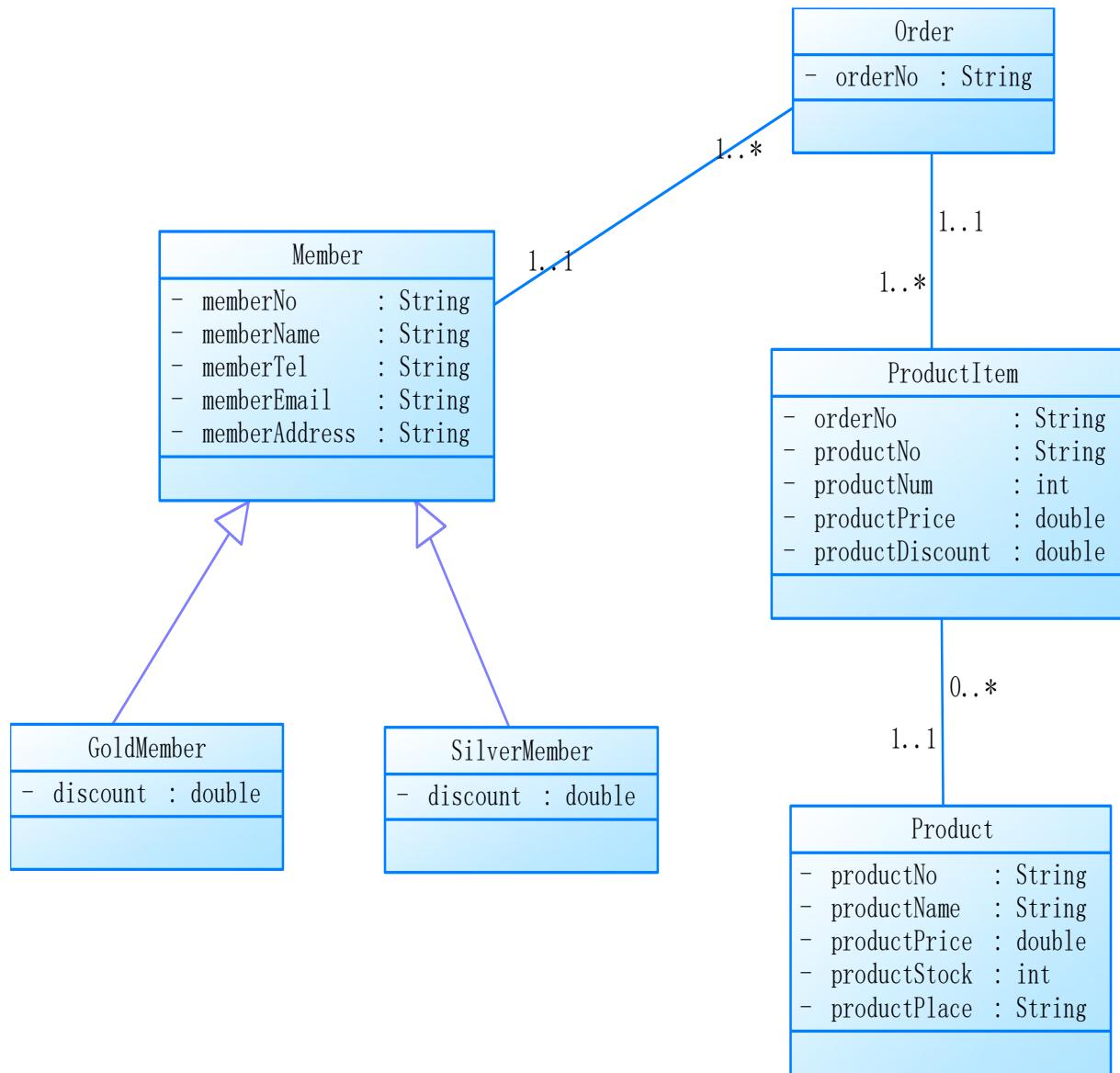


## 导出属性

- 在**Order**类中有一个名为**price**的属性，它前面加了一个“**/**”符号，这在**UML**模型中是用来表示导出（**derived**）属性的。导出属性的是指可以根据其他值计算出来的属性
  - 为了说明**order**类中的这个导出属性是如何计算出来的，还特意为其添加了一个注释。并在注释中写到：**price=sum(all orderitem.price)**，意思是这里的**price**属性的值应该等于所有的**orderitem**类的**price**字段值的总和。

# 课堂练习

根据以下描述设计并绘制类图：某商场会员管理系统包含一个会员类(Member)，会员的基本信息包括会员编号、会员姓名、联系电话、电子邮箱、地址等，会员可分为金卡会员(GoldMember)和银卡会员(SilverMember)两种，不同类型的会员在购物时可以享受不同的折扣；每个会员可以拥有一个或多个订单(Order)，每一个订单又可以包含至少一条商品销售信息(ProductItem)，商品销售信息包括订单编号、商品编号、商品数量、商品单价和折扣等；每一条商品销售信息对应一类商品(Product)，商品信息包括商品编号、商品名称、商品单价、商品库存量、商品产地等。



# 建立类图的基本过程

## 建立领域模型类图步骤：

1. 寻找候选类（名词识别法），去掉不恰当的类。
2. 识别类之间的关联关系
3. 识别类的属性、方法、标注关联关系的多重性。
4. 利用继承组织类
5. 迭代并细化模型：领域类模型在健壮之前需要多次细化。
  - 1) 如果某个类没有属性、操作和关联关系就考虑删除这个类。
  - 2) 如果有属性和操作没有宿主类，就考虑添加新类来存放这些属性和操作

# 建立类图的实例

## 1. 问题陈述

- 邓小平是一个爱书之人，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、册数按特定时限、周期进行统计。

# 建立类图的过程

## 2. 寻找分析类

- 我们以问题陈述为输入信息，采用“名词动词法”寻找分析类。名词动词法的主要规则是从名词与名词短语中提取对象与属性；从动词与动词短语中提取操作和关联。

### 1) 找备选类

- 首先。可以逐字逐句地阅读上面那段需求描述，并将其中的所有**名词及名词短语**列出来，可以得到**备选类列表**。

### 2) 从备选类中筛选出候选类

- 并不是所有的备选类都是适合候选类，有些名词对于要开发的系统来说无关紧要。甚至不属于系统；而有些名词表述的概念则相对较小，适合某个候选类的属性。因此，需要对备选类进行一番筛选，将这些不适合的排除掉。

# 建立类图的过程

- (1)“邓小平”、“人”、“家里”很明显是系统外的概念，无须对其实建模；
- (2)而“个人图书管理系统”、“系统”指的就是将要开发的系统，即系统本身，也无须对其进行建模；
- (3)很明显“书籍”是一个很重要的类，而“书名”、“作者”、“类别”、“出版社”、“书号”则都是用来描述书籍的基本信息的，因此应该作为“书籍”类的属性处理，而“规则”是指书号的生成规则，而书号则是书籍的一个属性，因此“规则”可以作为编写“书籍”类构造函数的指南。
- (4)“基本信息”则是书名、作者、类别等描述书籍的基本信息统称，“关键字”则是代表其中之一，因此无需对其建模；

# 建立类图的过程

- (5)“功能”、“新书籍”、“信息”、“记录”都是在描述需求时使用到的一些相关词语，并不是问题域的本质，因此先可以将其淘汰掉；
- (6)“计算机类”、“非计算机类”是该系统中图书的两大分类，因此应该对其建模，并改名为“计算机类书籍”和“非计算机类书籍”，以减少歧义；
- (7)“外借情况”则是用来表示一次借阅行为，应该成为一个候选类，多个外借情况将组成“外借情况列表”，而外借情况中一个很重要的角色是“朋友”——借阅主体。虽然到本系统中并不需要建立“朋友”的资料库，但考虑到可能会需要列出某个朋友的借阅情况，因此还是将其列为候选类。为了能够更好地表述，将“外借情况”改名为“借阅记录”，而将“外借情况列表”改名为“借阅记录列表”；

# 建立类图的过程

- (8)“购买金额”、“册数”都是统计的结果，都是一个数字，因此不用将其建模，而“特定期限”则是统计的范围，也无需将其建模；不过从这里的分析中，我们可以发现，在该需求描述中隐藏着一个关键类—书籍列表，也就是执行统计的主体。
- 通过上面的分析,得到一个候选列表:

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

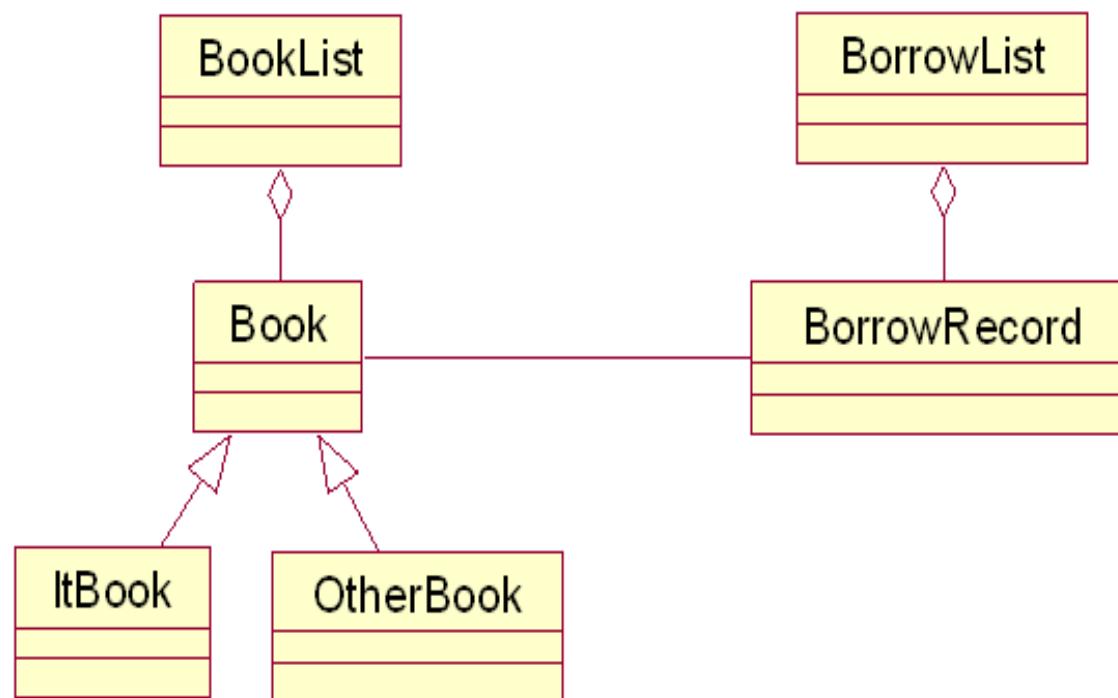
# 建立类图的过程

## 3 确定类关系

- 通过上面的工作，从需求描述中找到了6个相关的类，接下来就是确定类之间的关系。
- 1) 确定类关系
  - 可以发现“计算机书籍（**itbook**）”、“非计算机书籍（**otheritbook**）”与“书籍（**book**）”之间是继承关系；而“书籍列表（**booklist**）”是多个“书籍”组成的，“借阅记录列表（**borrowlist**）”是由多条“借阅记录”组成的。这种组成关系使用于组合还有聚合关系呢？显然，由于本系统的“书籍”是可以独立于“书籍列表”而存在；“借阅记录”也是可以独立于“借阅记录列表”而存在，因此使用聚合更合适一些。还可以发现“借阅记录”和“书籍”是关联的，离开“书籍”，“借阅记录”不存在意义。



- 为了反映和记录这些类之间的关系，可以使用**UML**中的类图将其记录下来。





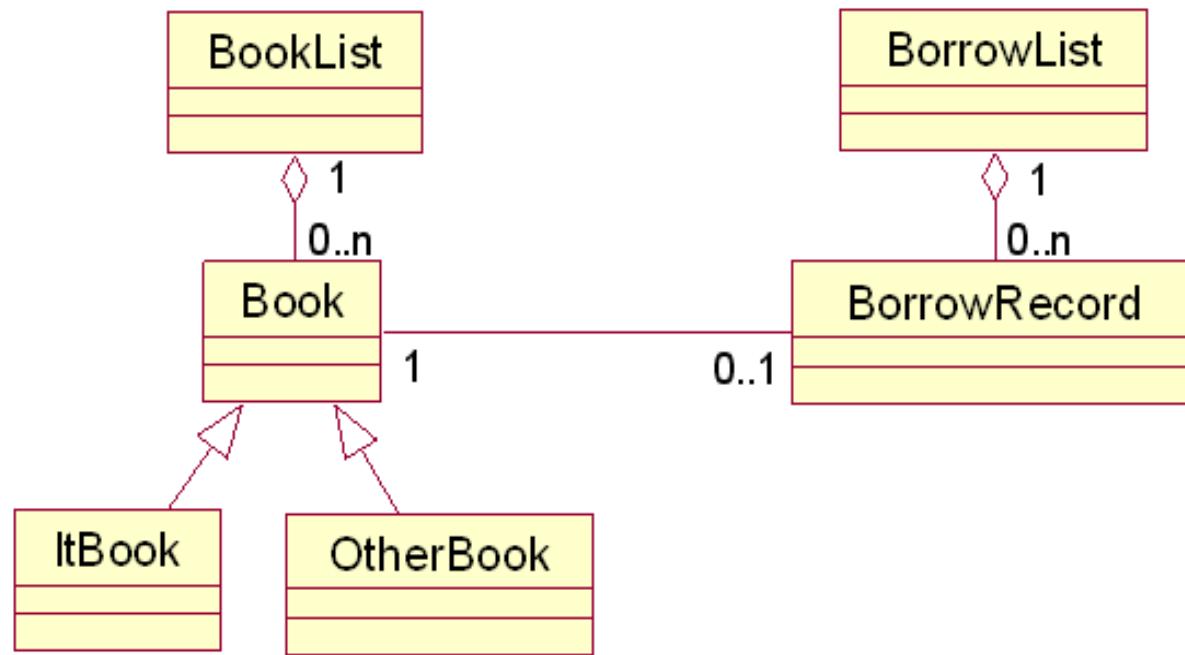
## 2) 给关联添加属性

### (1) 确定关联的多重性

- 例如一本书可以有几条借阅记录，书籍列表指的是多少本书籍，这些问题需要进一步的进行多重性分析，并修改上面所示的类图。
- 因为是个人藏书，因此每本书都是唯一的，没有副本，要么被借出，要么未被借出，因此对于每一本书籍来说，要么只有一条借阅记录，要么没有借阅记录。
- 所有的书籍组成书籍列表，借阅记录列表是由所有的借阅记录组成。
- 通过上面的分析，可以得到信息补充的类图，即可得到类模型。



- 如果系统较大，可以以上面的类图为基础，把关联度紧密的类合成一个包，以便更好的组织子系统。例如，在本例中可以将“书籍列表”、“书籍”、“计算机书籍”、“非计算机书籍”合成一个包，而将“借阅记录”、“借阅记录列表”合成另一个包。但本例比较简单，类相对较少，因此无须进行这样的合成。



## (2) 确定关联的导航性

- 类图中的诸如导航性，角色名，导出属性，限定符及约束等高级属性不是每个类模型都必须加入的。
- 在图中，只有**book**和**booklist**之间的组合关系，**borrowrecord**与**borrowlist**之间的组合关系、**book**与**borrowrecord**之间的关联关系，这三个关系可能存在导航性。
- 组合关系显然已经将类的关系清晰化了，因此无须对其进行导航性描述。根据对需求的理解，**book**与**borrowrecord**之间，应该是一个双向链接。因为，当浏览书籍列表时，会希望看到某本书是否被借出；当有人归还时，希望能从借阅记录中关联到**book**。



### (3) 确定约束

- 根据用户需要，我们有两个地方可以用约束来体现：一是**book**对象创建之后就不能被删除，只能做修改，因此在**book**类边上加上了一条用自由文本写的**约束**。二是一本书要么属于计算机类，要么属于非计算机类。因此要加一个“**{xor}**”约束。

### (4) 确定关联的限定符

- 由于这个系统是“个人图书管理系统”，因此特定的一本书只有一本，所以只能被借一次，因此对于一本书而言，只有一个**Recordid**与其对应，因此将添加一个**Recordid**限定符。把限定符加入图中，再把类的职责(属性和方法)加入到类图后，得到的类图。

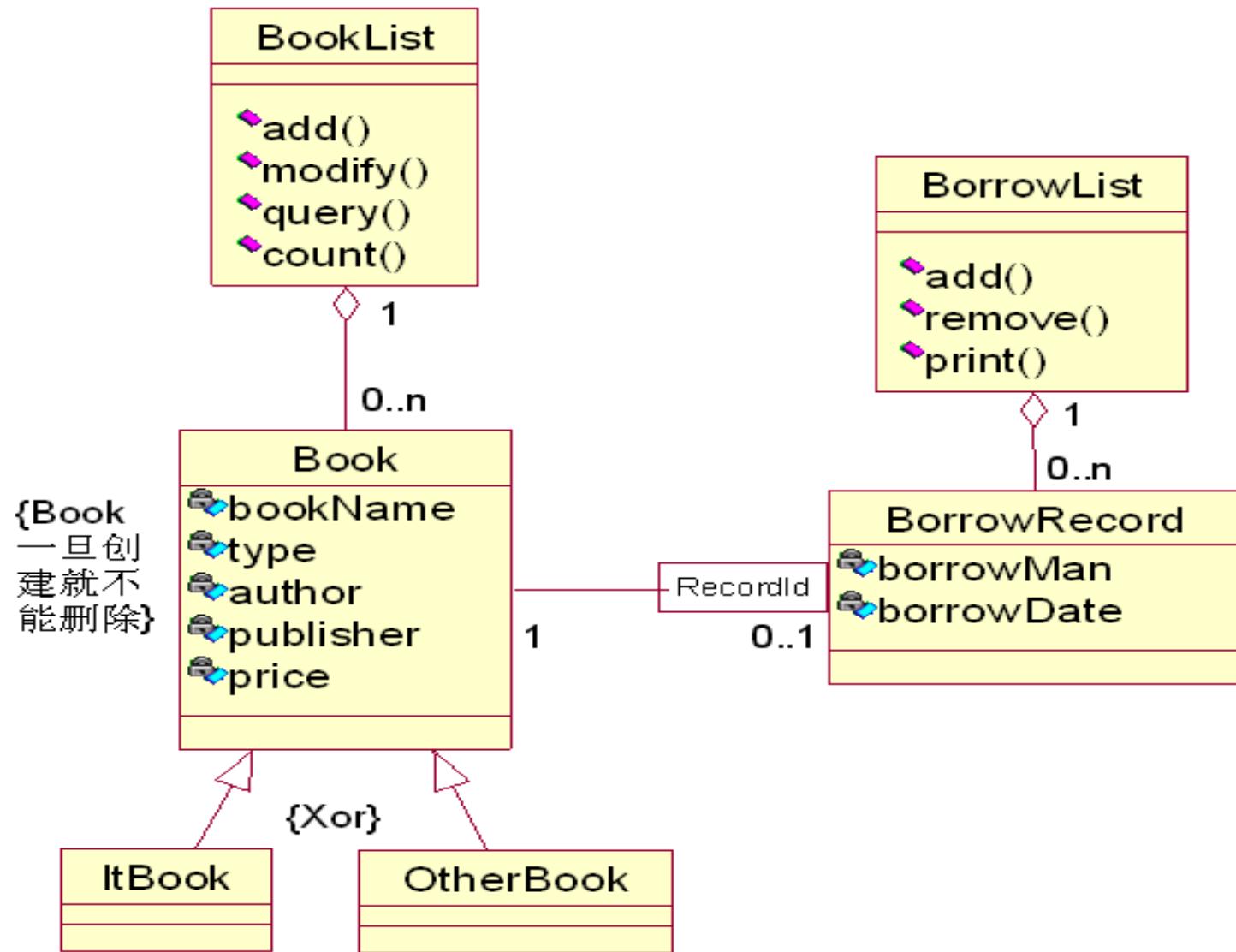


### 3) 给类添加职责

- 当找到了反应问题域本质的主要类，并清理他们之间的关系之后，就可以为这些类添加相应的职责。类的职责包括以下两个内容：类所维护的信息(成员变量)和类提供的行为(成员方法)。
- 在本阶段将主要的成员变量和成员方法标识出来，以便更好的理解问题域。
  - 书籍类：从需求描述中，可找到书名、类别、作者、出版社；同时从统计的需要中，可得知“定价”也是一个关键的成员变量。



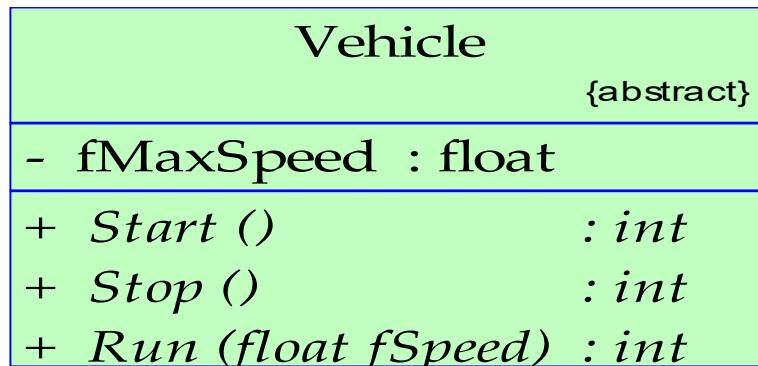
- 书籍列表类：书籍列表就是全部的藏书列表，其主要的成员方法是新增、修改、查询（按关键字查询）、统计（按特定时限统计册数与金额）。
- 借阅记录类：借阅人（朋友）、借阅时间。
- 借阅记录列表类：主要职责就是添加记录（借出）、删除记录（归还）以及打印借阅记录
- 通过上面的分析，我们对这些概念类有了更深入的了解，可以重新修改类，将这些信息加入原先的模型中。同时，把关联的属性加入类模型后。
- 职责（属性，方法）的添加是一个循序渐进的过程，在类分析，类设计时都是逐步对类模型进行完善的。



加入限定符和约束的类图

# 类图与代码的映射

## 类的映射



C++代码

```
class Vehicle
{
public:
    virtual int Start() = 0;
    virtual int Stop() = 0;
    virtual int Run(float
fSpeed) = 0;
private:
    float fMaxSpeed;
};
```

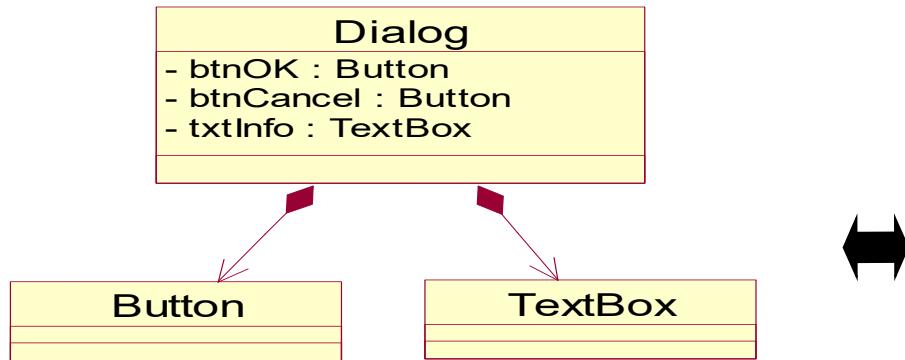
Java代码

```
public abstract class Vehicle
{
    public abstract int Start();
    public abstract int Stop();
    public abstract int Run(float
fSpeed);
    private float fMaxSpeed;
}
```

# 类图与代码的映射



## 关联关系的映射



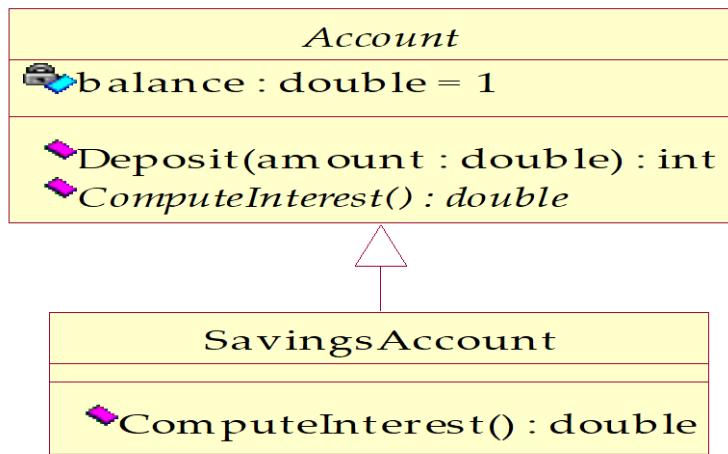
组合关系，代码表现为Dialog的属性有Button和TextBox的对象

C++代码  
class Dialog

```
{  
private:  
    Button btnOK;  
    Button btnCancel;  
    TextBox txtInfo;  
};  
class Button  
{};  
class TextBox  
{};
```

# 类图与代码的映射

## 泛化关系的映射



C++代码

```
class SavingsAccount : public Account
{ };
```

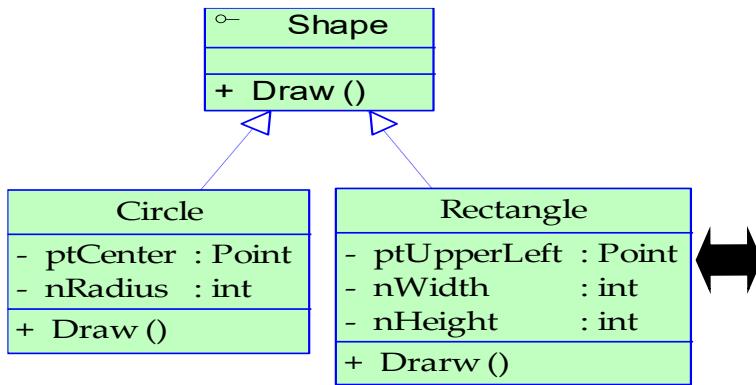


Java代码

```
public class SavingsAccount extends Account
{ }
```

# 类图与代码的映射

## 实现关系的映射



在C++语言里面，使用抽象类代替接口，使用泛化关系代替实现关系  
在Java语言里面，有相应的关键字  
interface、 implements

C++代码

```
class Shape
{
public:
    virtual void Draw() = 0;
};

class Circle : public Shape
{
public:
    void Draw();
private:
    Point ptCenter;
    int nRadius;
};
```

Java代码

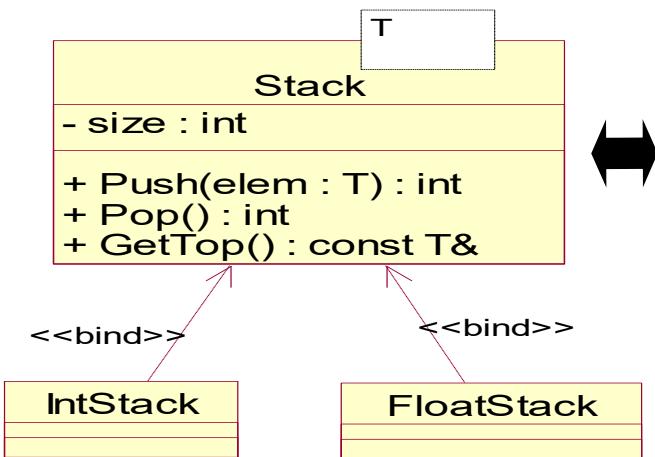
```
public interface Shape
{
    public abstract void Draw();
}
```

```
public class Circle implements Shape
{
    public void Draw();

    private Point ptCenter;
    private int nRadius;
}
```

# 类图与代码的映射

## 依赖关系的映射



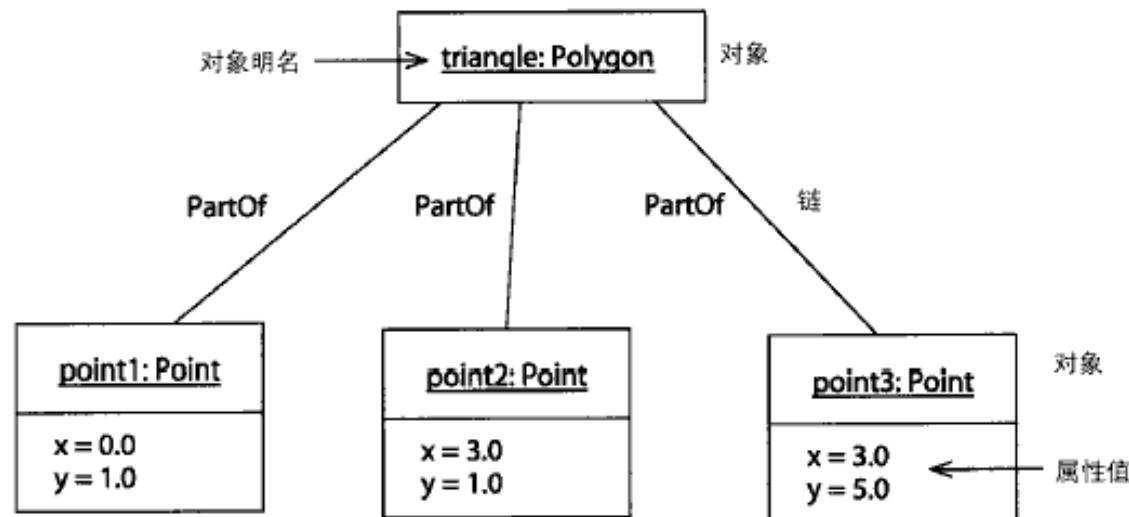
C++代码  
template<typename T>  
class Stack  
{  
private:  
 int size;  
public:  
 int Push(T elem);  
 int Pop();  
 const T& GetTop();  
};  
  
typedef Stack<float>  
**FloatStack**;

C++代码(编译器生成)  
class **FloatStack**  
{  
private:  
 int size;  
public:  
 int Push(float elem);  
 int Pop();  
 const float&  
 GetTop();  
};

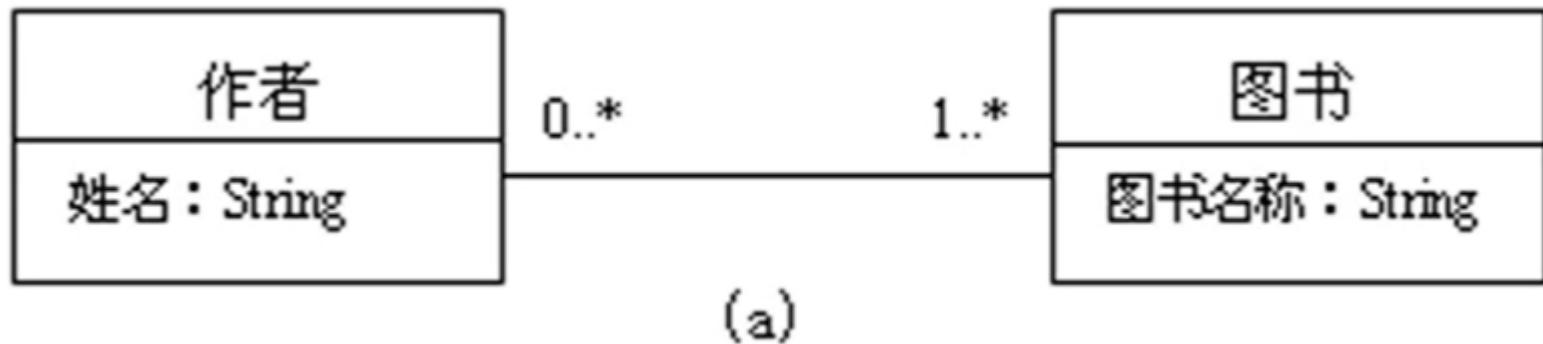
# 对象图

## ◇ 对象图

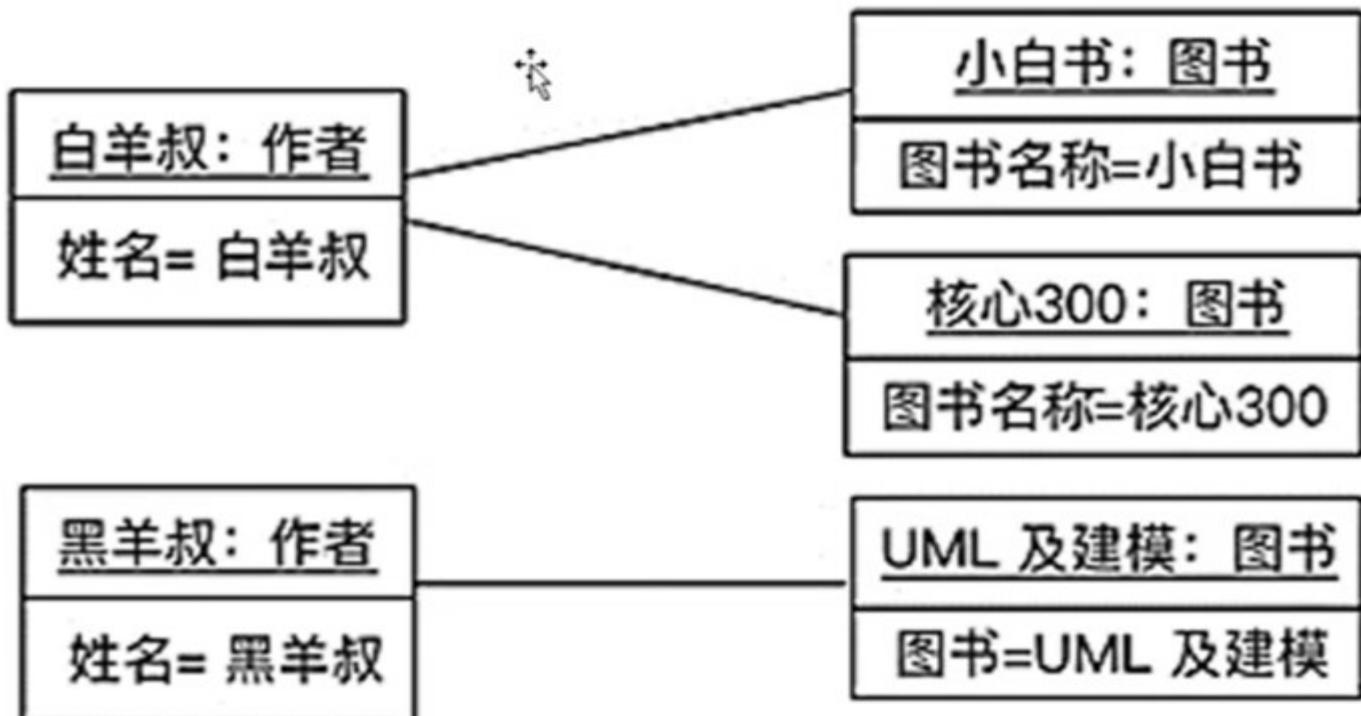
- **对象图(Object Diagram)** 是显示了一组对象和他们之间的关系。使用对象图来说明数据结构，类图中的类或组件等实例的快照。对象图和类图一样，反映了系统的静态过程，但它是以实际的或原型化为基础来表达对象间的关系。
- **对象图**显示某时刻对象和对象之间的关系。一个对象图可看成一个类图的特殊实例，实例和类可在对象图中同时表示。



## 类图



## 对象图





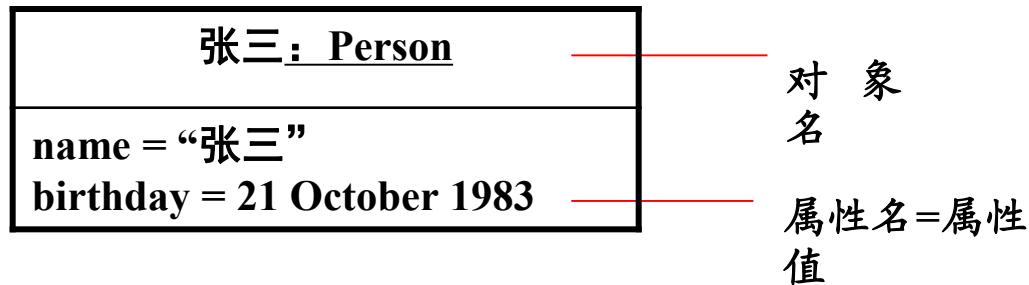
# 对象图

## ◇ 对象图的组成

- 对象图是由**对象**和**链**组成的。对象图的目的在于描述系统中参与交互的各个对象在某一时刻是如何运行的。

# 对象的表示

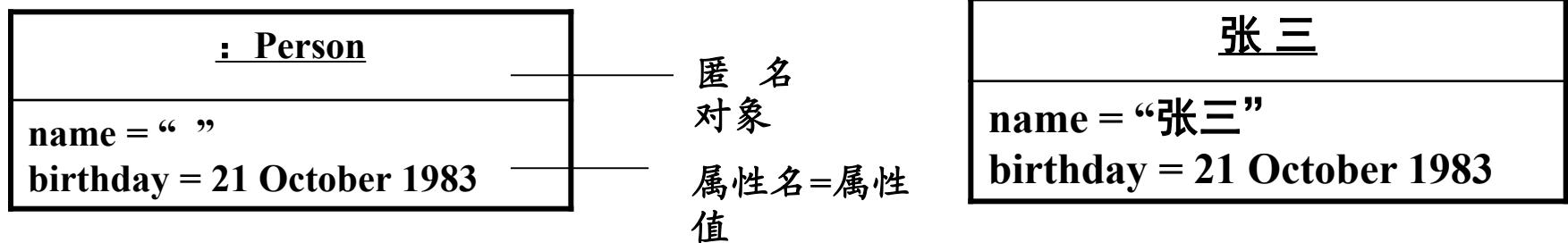
- UML中，表示一个对象，主要是标识它的名称、属性。对象由一个矩形表示，它包含2栏，在第一栏写入对象名，在第二栏列出属性名及属性值，格式如：“属性名=属性值”
  - 对象名有下面三种表示格式，不同点在于第一栏表示对象的格式不同：
    - 对象名：类名
      - 对象名在前，类名在后，用冒号来连接。对象名和类名都加下划线。



# 对象的表示

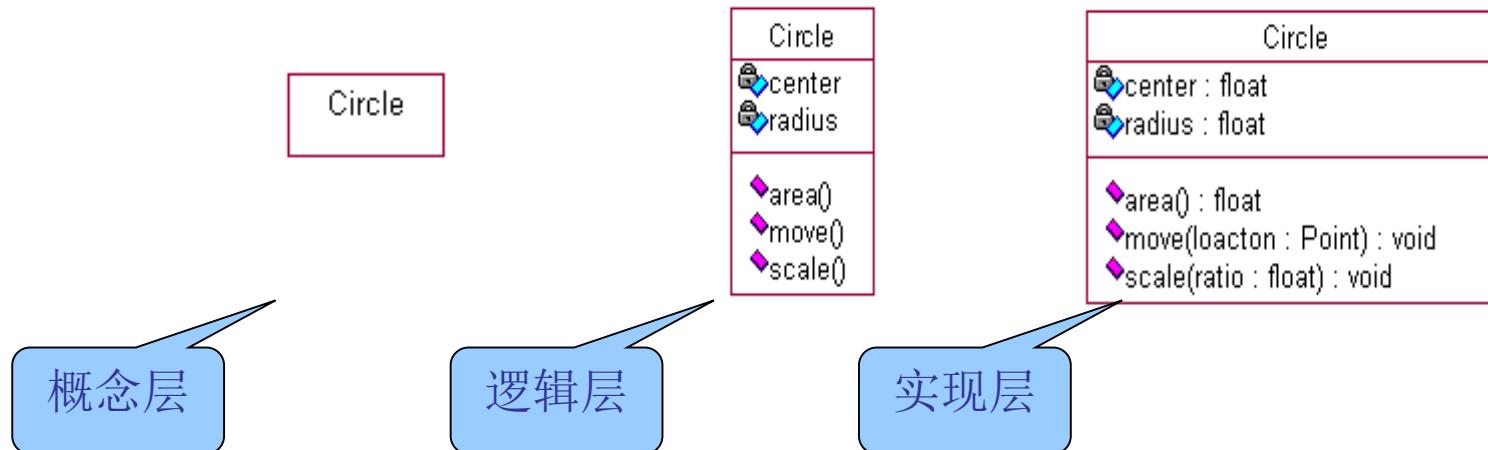


- `:类名`
  - 这是对匿名对象的表示方法。这种格式用于尚未给对象取名的情况，前面的冒号不能省略。
- 对象名
  - 省略格式，即省略掉类名。只有对象名，对象名必须加下划线。



# 对象的表示

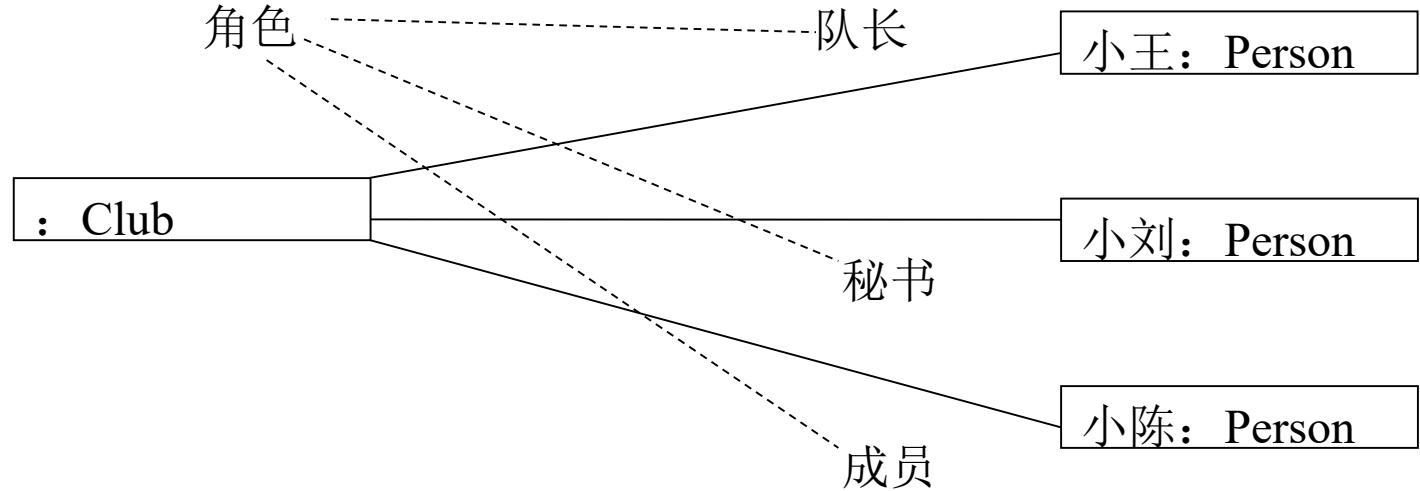
在系统的不同开发阶段，类图可以具有不同的抽象程度。  
随着开发的深入，类图应该越来越详细、具体。  
可以分为：概念层，逻辑层，实现层。



# 链的表示

- 链是两个对象间的语义关系。关联是两个类间的关系。就像对象是类的实例一样，链接是关联的实例。链接分单向链接和双向链接。

## 1. 双向链



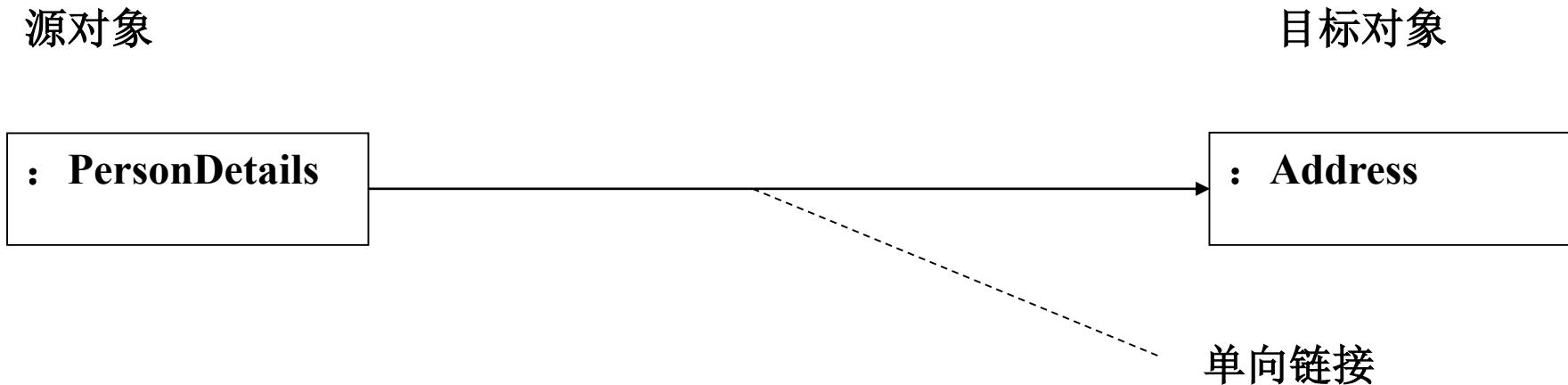
其中，队长、秘书和成员都是角色名称。分别表示小王，小刘，小陈在链接中充当的角色。

# 链接的表示



## 2. 单向链

- 单向链如图所示。



: PersonDetails到: Address的链是单向的，即，对象: PersonDetails可以引用对象: Address，反之不然。



- 阅读对象图的方法
  - 对象图中，对象间的关系称为链。阅读对象图的方法：
    - (1).找出图中所有的类
    - (2).了解每个对象的语义
    - (3).了解对象之间连接含义。