# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖

识别对象之间的关系

# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖

## 一、识别继承（泛化）


### 1、识别继承（泛化）关系的策略

**（1）学习当前领域的分类学知识**

因为问题域现行的分类方法往往比较正确地反映事物的特征、类别以及各种概念的一般性和特殊性。按照问题域已有的分类方法，可以找出一些与它对应的继承关系。

**（2）按常识考虑事物的分类**

如果问题域没有可供参考的现行分类方法，可以按照自己的常识，从各种不同的角度考虑事物的分类，从而发现继承关系。

例如对于“人员”可以从以下几种角度去分类：

- 青年人、成年与老年；
- 男人与女人；
- 黄种人、白种人和黑种人；
- 在职人员与离退休人员；
- 正式职工与临时工；
- 。。。。。。

从不同的角度考虑问题域中事物的分类，可以形成一些建议一般-特殊关系的初步设想，从而启发自己发现一些确实需要的一般-特殊关系。

# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖

## (3) 使用继承的定义

使用两种思路去发现继承关系：

(a) 一种思路是把每个类看作是一个对象集合，分析这些集合之间的包含关系，如果一个类是另一个类的子集（例如“职员”是“人员”的子集，“轿车”是“汽车”的子集），则它们应组织到同一个一般-特殊关系中。

(b) 看一个类是不是具有另一个类的全部特征，这又包括以下两种情况：

- 一种是建立这些类时已经计划让某个类继承另一个类的全部属性与操作，现在应建立继承关系来落实；
- 另一种是起初只是孤立地建立每个类，现在发现一个类中定义的属性与操作全部在另一个类中重新出现，此时应考虑建立继承关系，把后者作为前者的特殊类，以简化其定义。

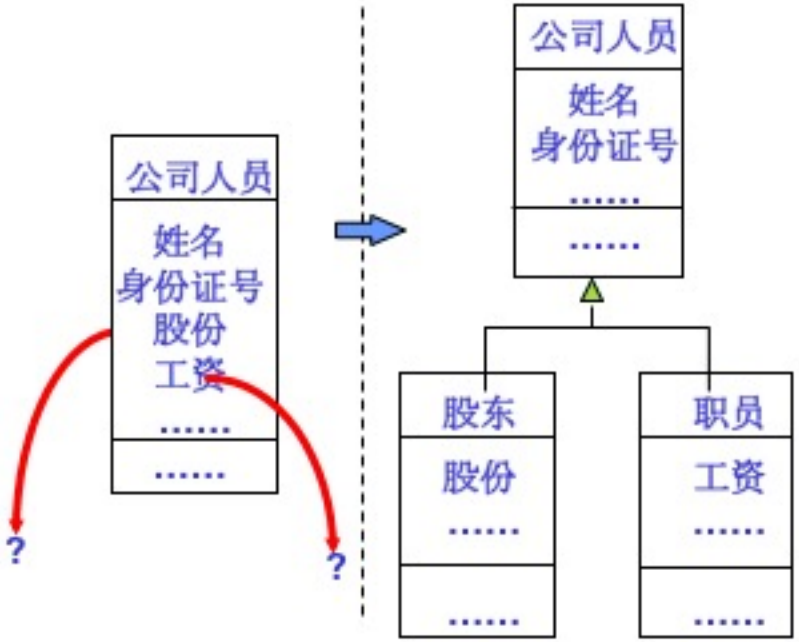
# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖

## (4) 考察属性与操作的适用范围

对系统中的每个类，从以下两方面考虑它们的属性与操作：

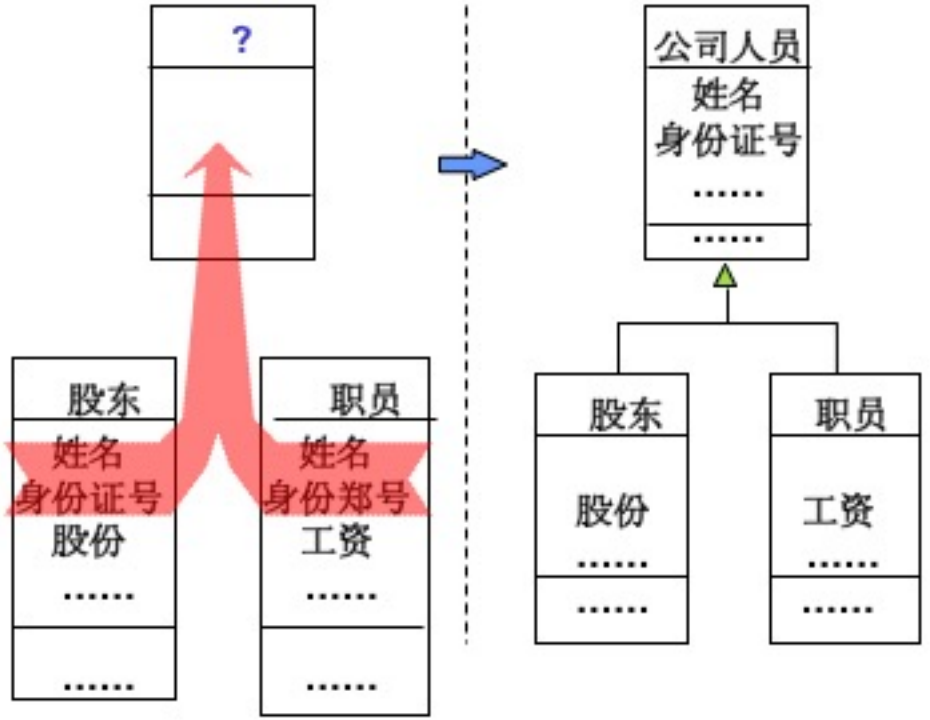
- **看一个类的属性与操作是否适合这个类的全部对象。**如果某些属性或操作只适合该类的一部分对象，说明应从这个类中划分出一些特殊类，建立继承关系。



# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖

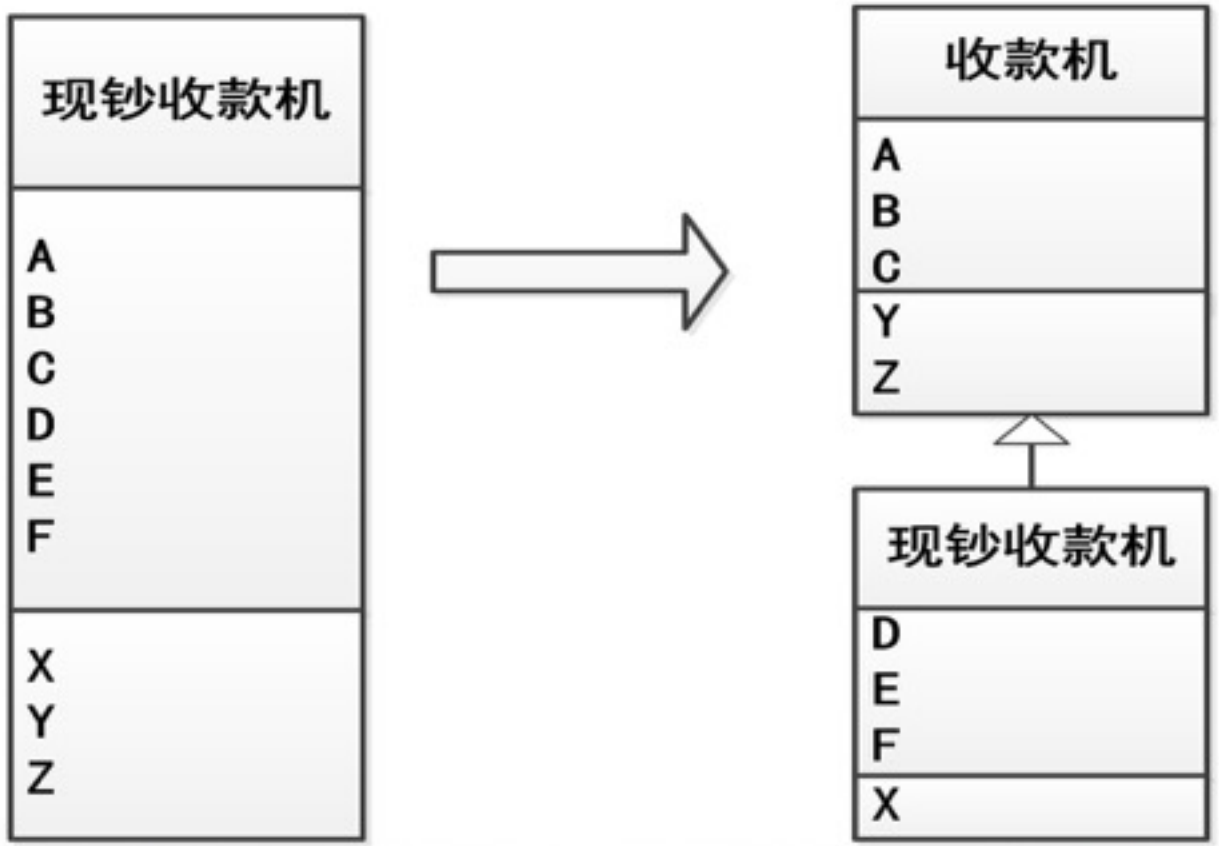
- **检查是否有两个（或更多）的类含有一些共同的属性与操作。**如果有则考虑，若把这些共同的属性与操作提取出来，能否构成一个在概念上包含原先那些类的一般类，形成一个继承关系。



# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖

（5）考虑领域范围内的复用

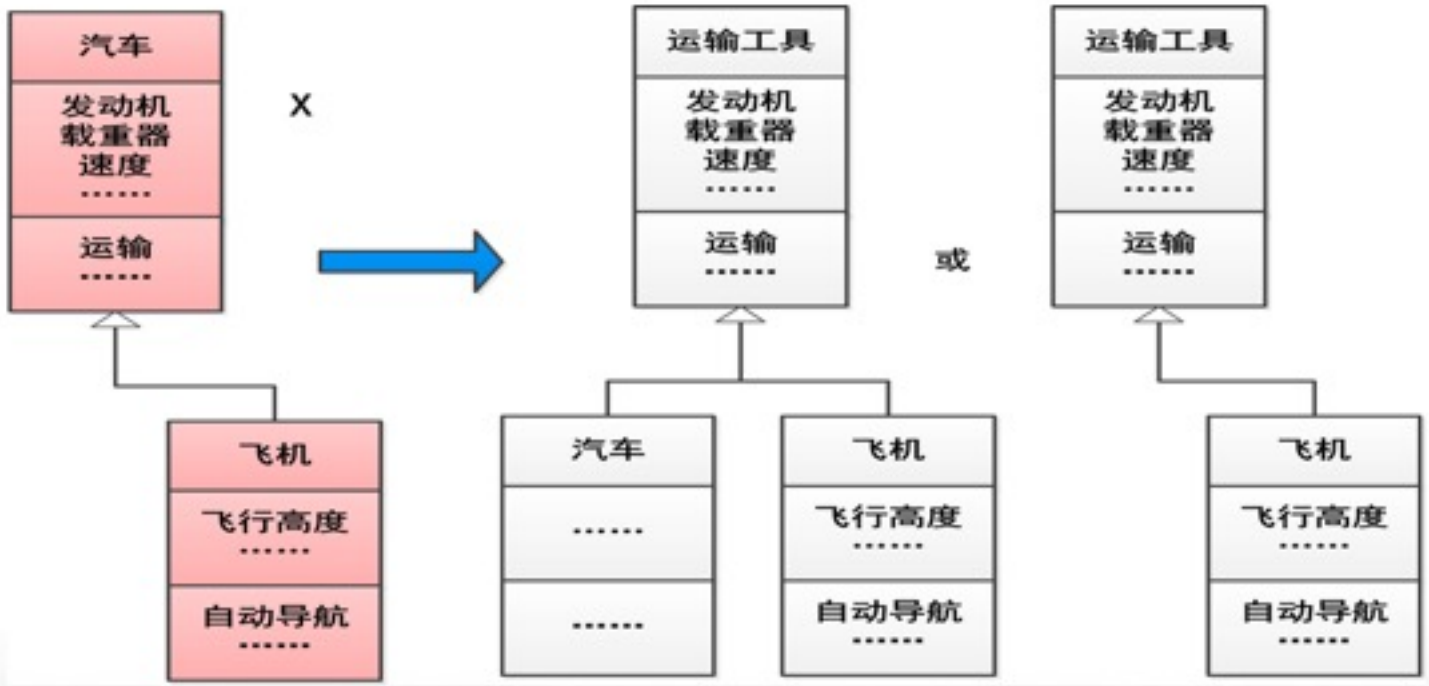


# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖

## 2、审查与调整

(1) 问题域是否需要这样的分类？（例：书—线装书）
(2) 系统责任是否需要这样的分类？（例：职员—本市职员）
(3) 是否符合分类学的常识？（用“is a kind of ”去套）




(4) 是否构成了继承关系？（确实继承了一些属性或操作）

例如：航标船

北京大学

# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖

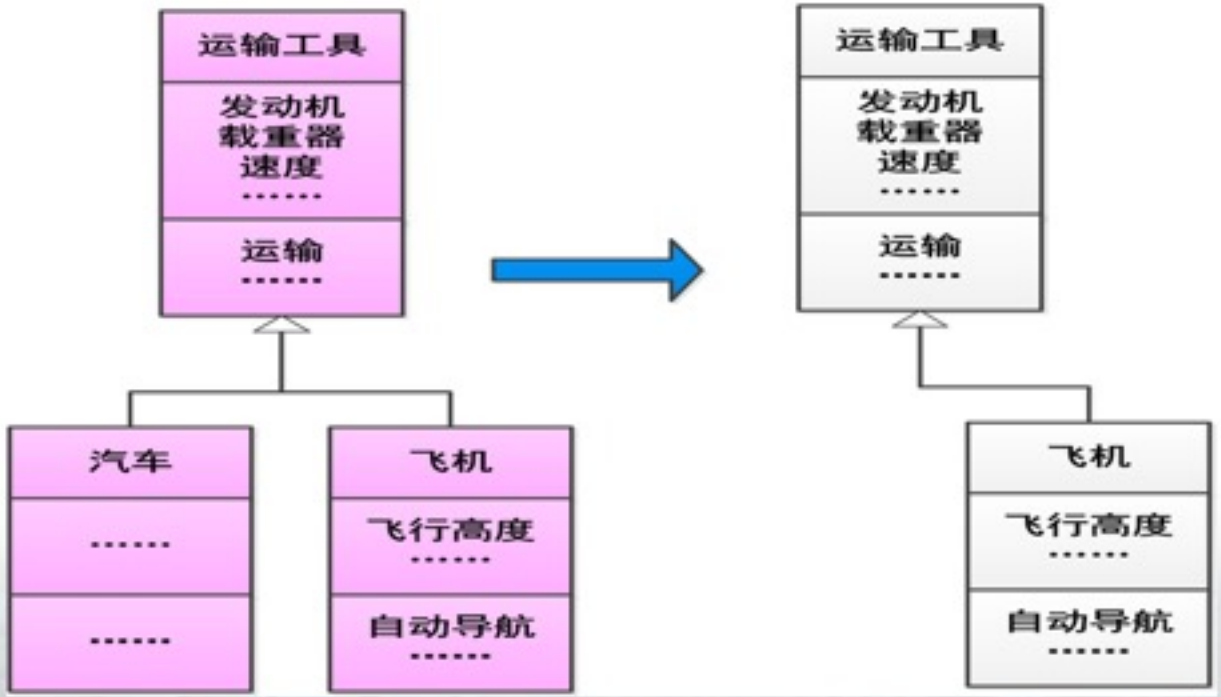
## 3、继承关系的简化

（1）从一般类划分出太多的特殊类，使系统中类的设置太多，增加了系统的复杂性；
（2）建立过深的继承层次，增加了系统的理解难度和处理开销。

对继承关系的运用要适度

重点考察以下情况：

（1）取消没有特殊性的特殊类



# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖

（2）增加属性简化继承关系（某些特殊类之间的差别可以由一般类的某个属性值来体现，而且除此之外没有太多的不同）

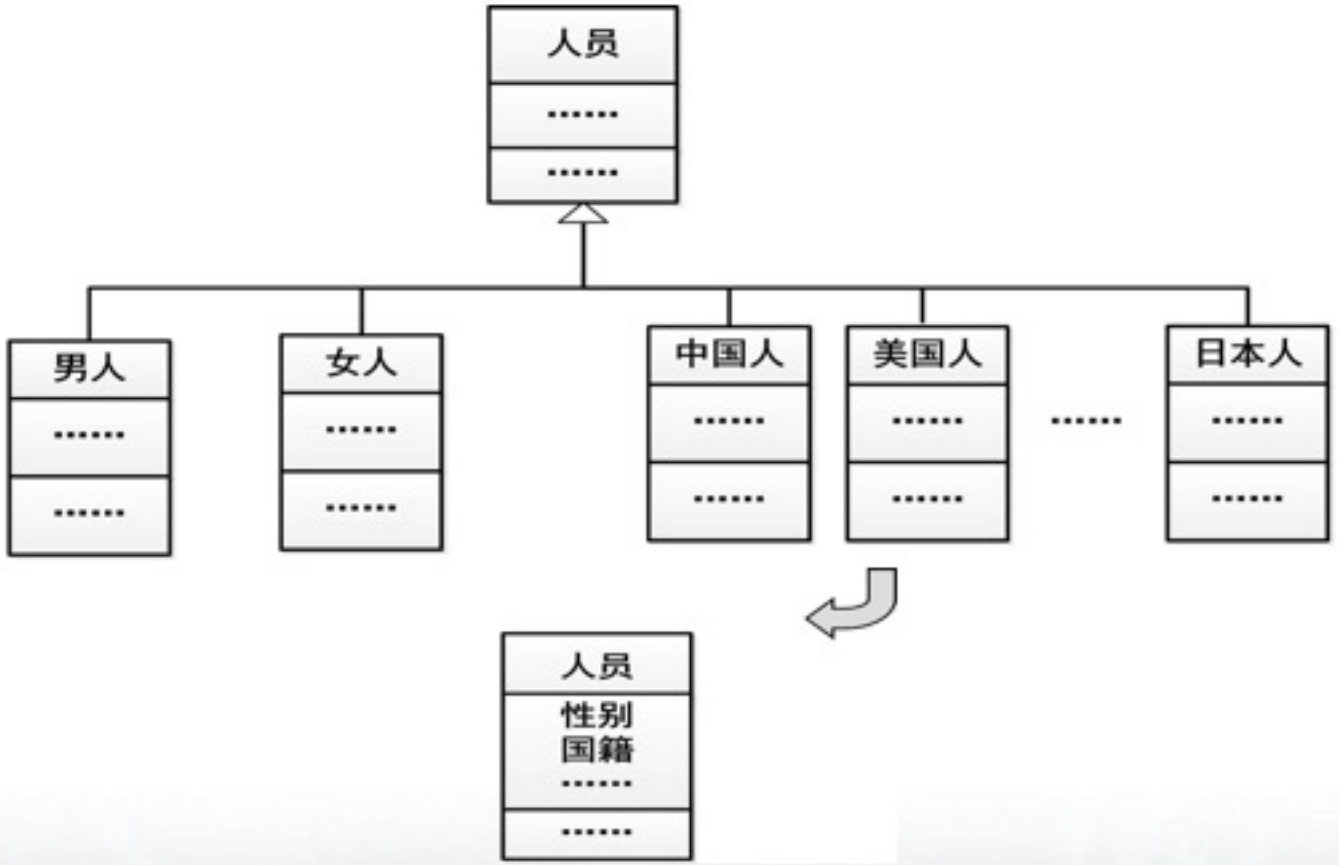



图 通过增加属性简化继承关系

# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖

## （3）取消用途单一的一般类，减少继承层次

**一般类存在的理由：**

- 有两个或两个以上的特殊类
- 需要用它创建对象实例
- 有助于软件复用

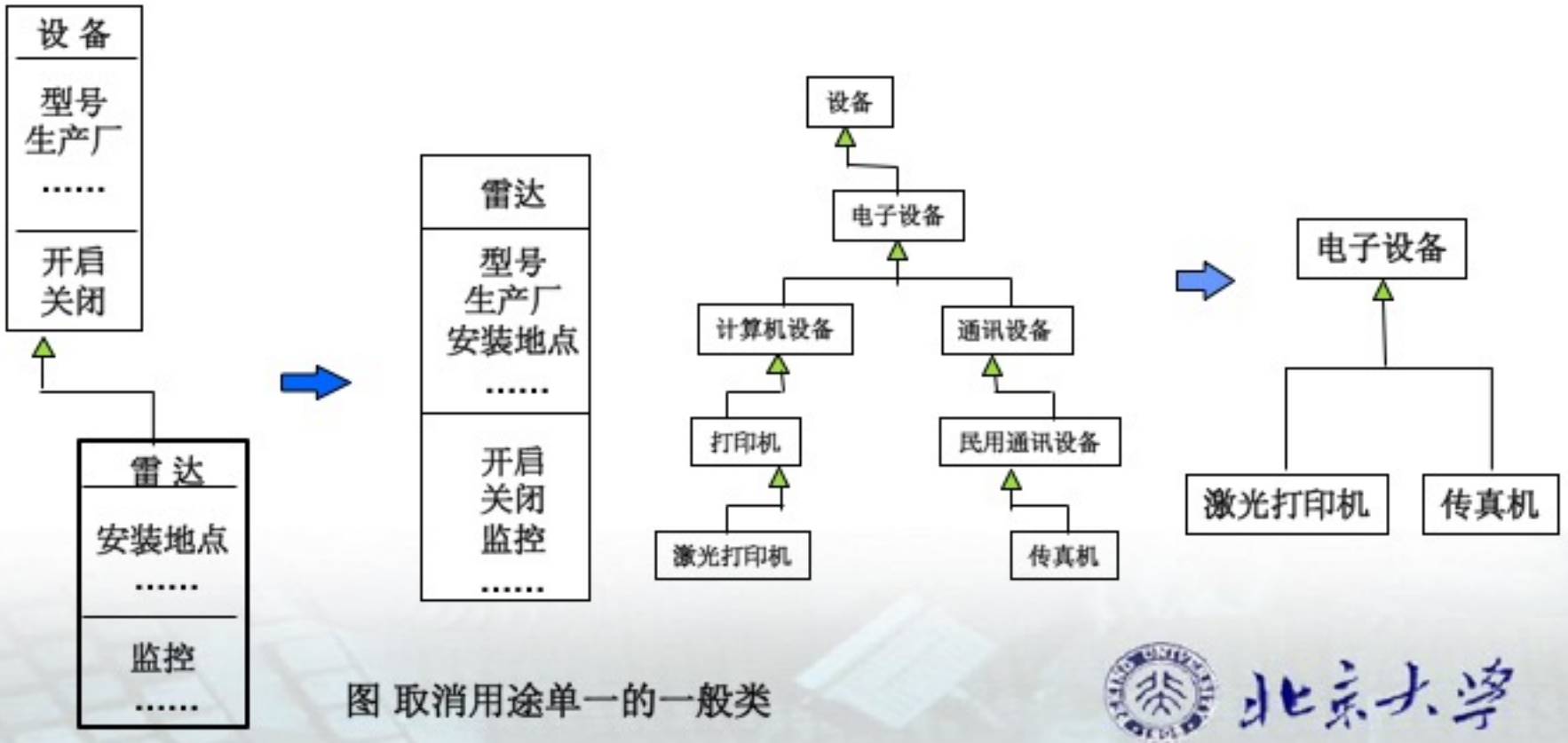



图 取消用途单一的一般类

# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖

## 4、调整对象层和特征层

定义继承的活动，将使分析员对系统中的对象和类及其特征有更深入的认识，在很多情况下，随着继承的逐步建立，需要对类图的对象层和特征层进行某些修改，包括增加、删除、合并或分开某些类，以及增、删某些属性与操作或把它们移到其他类。

识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖

# 二、识别关联


## 1、识别关联的策略

（1）认识对象之间的静态联系

考虑问题域和系统责任——哪些类的对象实例之间的关系需要在系统中表达。

（2）认识关联的属性与操作

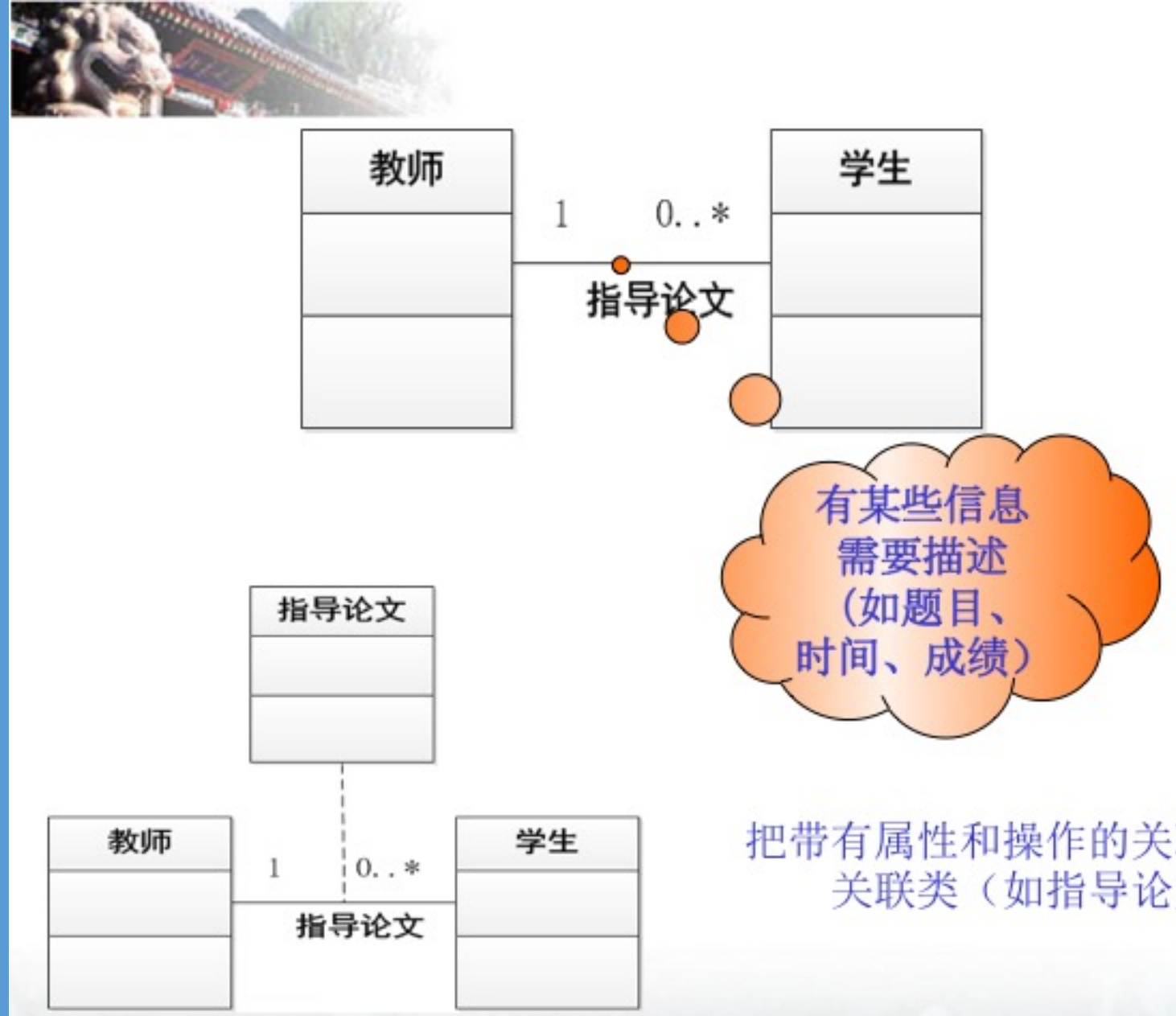
对于考虑中的每一种关联，进一步分析它是否应该带有某些属性和操作。就是说，是否含有一些仅凭一个简单的关联不能充分表达的信息。

例：在教师和学生的“指导毕业论文”的关联中，是否需要给出毕业论文的题目、答辩时间、成绩等属性信息？

如果有，则可先在关联线上附加一个关联类符号来容纳这些属性和操作，或在两个类之间插入一个类来描述这些属性与操作。

# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖



# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖

（3）分析并表示关联的多重性

对于每个关联，从连接线的每一端看本端的一个对象与另一端的几个对象发生连接，把结果标注在连接线的另一端。

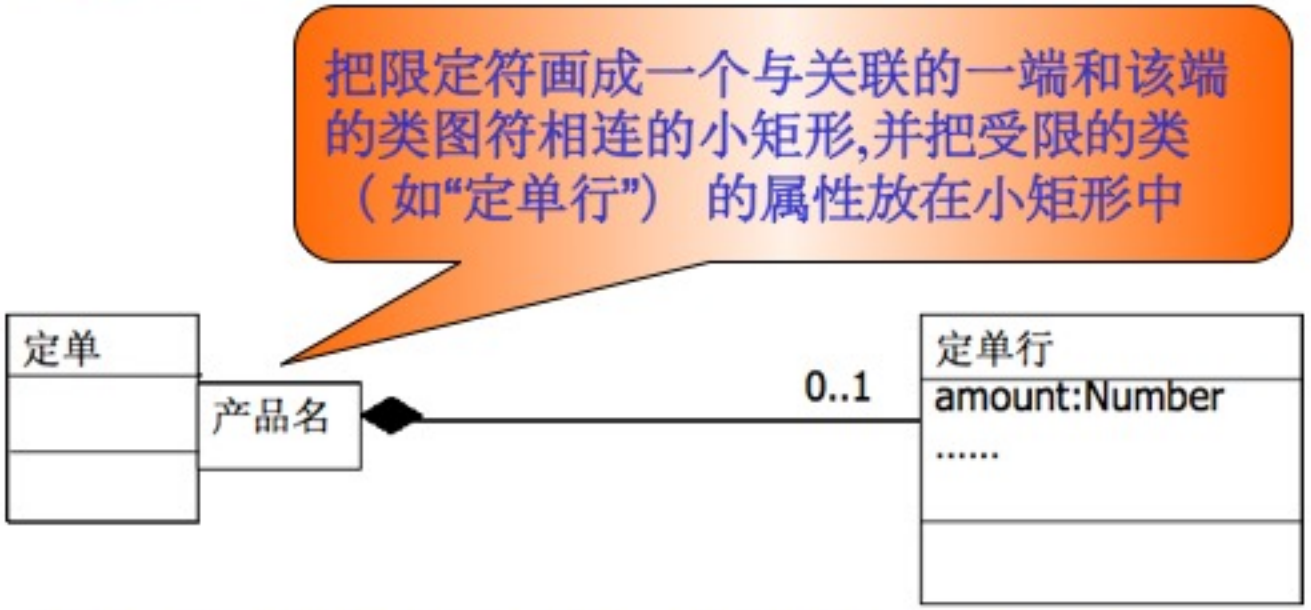
（4）进一步分析关联的性质

若需要，使用关联角色和限定符，以详细描述关联的性质。

限定符的值用于确定该关联的另一端类的对象.即给定类的一个对象,并指定限定符内的属性值,能选定另一端类的一个对象或一组对象.

# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖



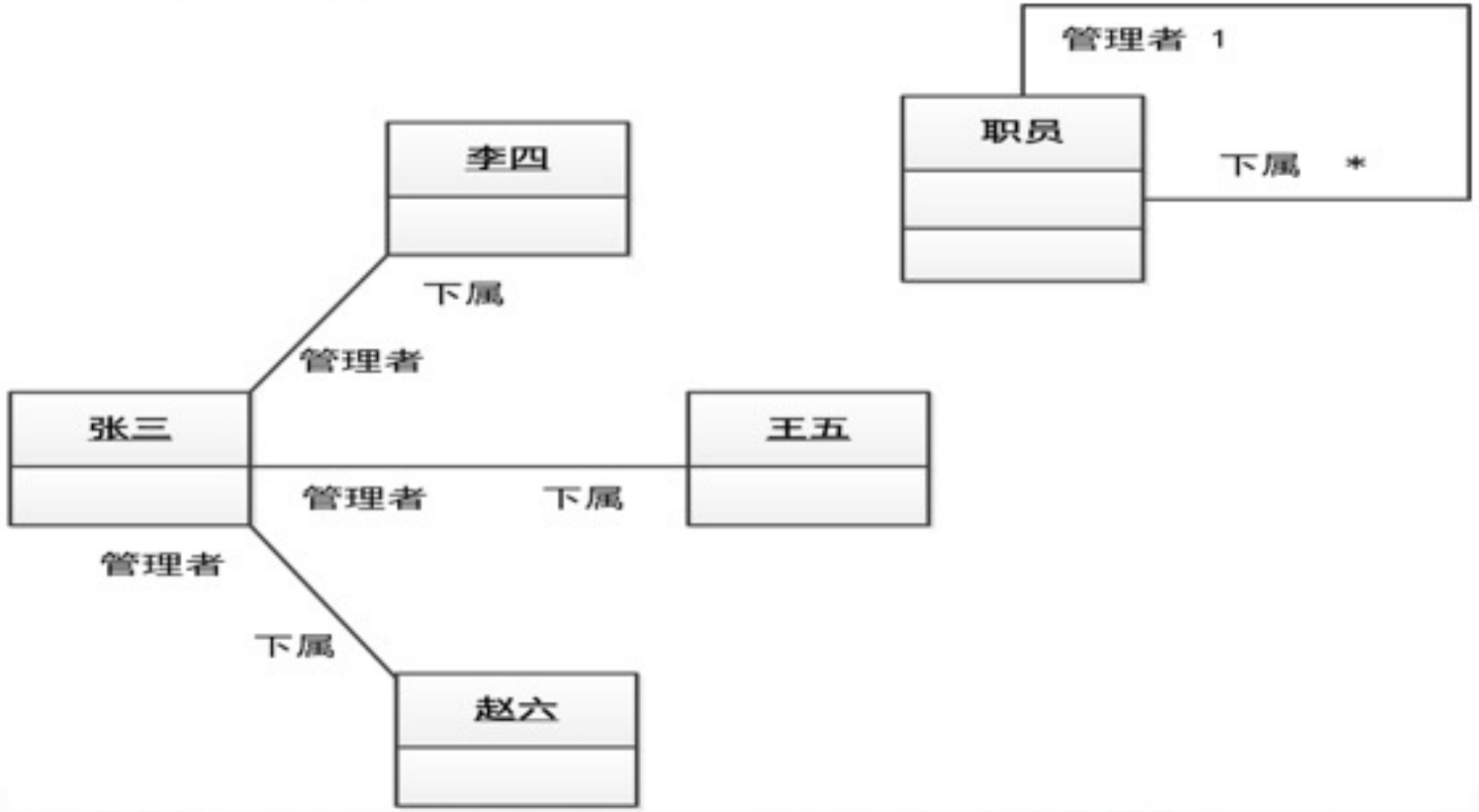

通常产品定单由定单行和一些其他描述信息组成。上图使用带有限定符的组合描述定单、定单行以及它们之间的关系。

对于一份定单，并指定了产品名，在另一端可能有或没有一个定单与其对应。如果没有这个限定符，给定一份定单，对应的定单行可能有许多。

# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖

**关联角色**:一个类参与一个关联的角色标识。



# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖

## 2、命名与定位命名：

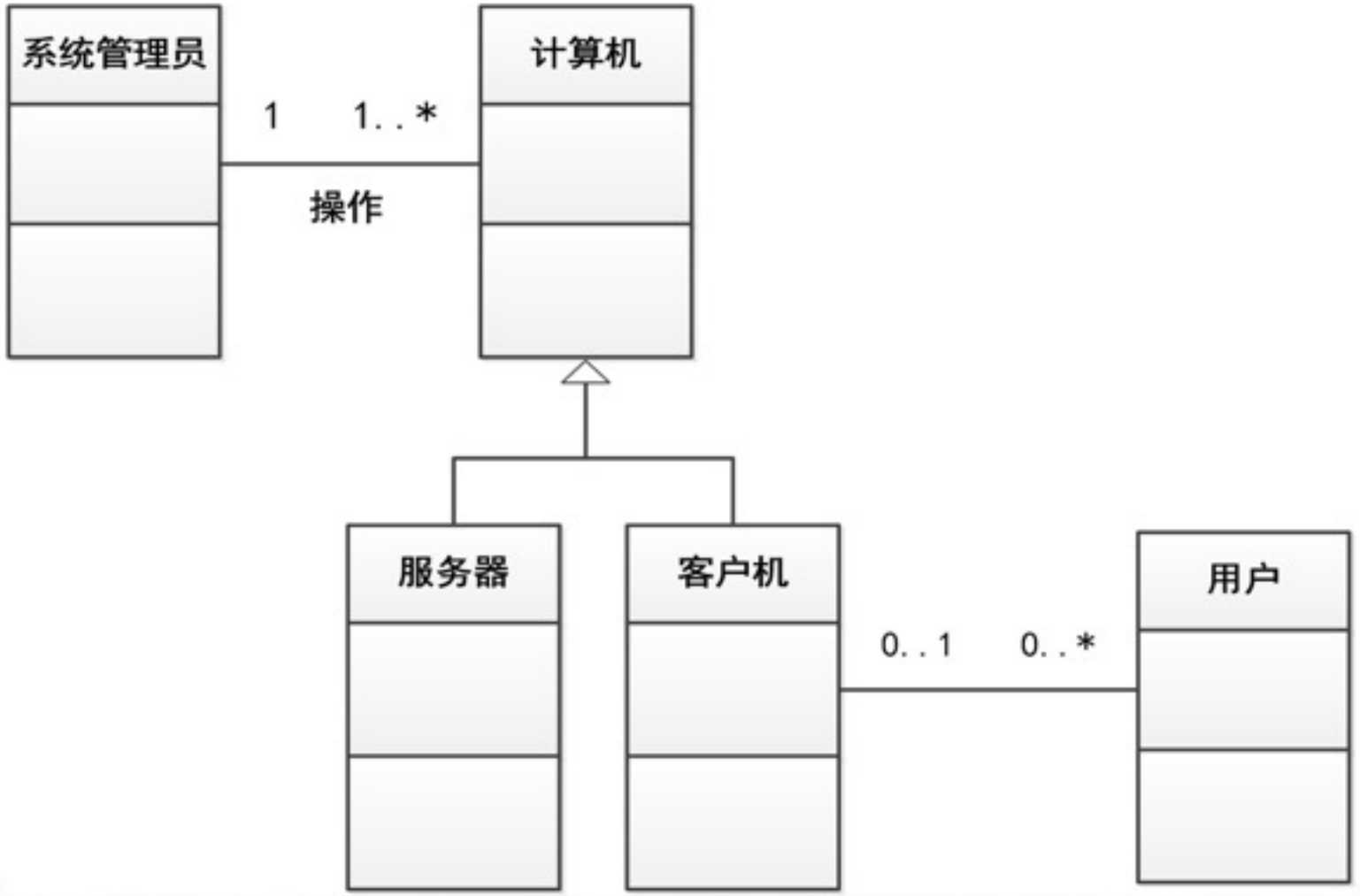
当关联线的语义很清晰时，则关联的命名可缺省。否则，关联可用动词或动宾结构命名。

**定位问题：**当连接线的某一端是一个继承结构时，要考虑连接线画到结构中的哪个类符号上。

**原则：如果这个关联适应结构中的每个类的对象，则画到一般类上，如果只适应其中某些特殊类，则画到相应的特殊类上。**

# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖



# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖

3、调整对象层和特征层

在建立关联的过程中可能增加一些新的类，要把这些新增加的类补充到类图的对象层中，并建立它们的类规约。

对于每一个关联，要给出其有关性质的说明，至少要说明它所表示的实际意义.

# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖

## 三、识别聚合


### 1、识别聚合的策略

(1) 物理上的整体事物和它的组成部分

例：机器、设备和它的零部件

(2) 组织机构和它的下级组织及部门

例：公司与子公司、部门

(3) 团体（组织）与成员

例：公司与职员

(4) 一种事物在空间上包容其它事物

例：生产车间与机器

(5) 抽象事物的整体与部分

例：学科与分支学科、法律与法律条款

(6) 具体事物和它的某个抽象方面

例：人员与身份、履历

(7) 在材料上的组成关系

例如，面包由面粉、糖和酵母组成，汽车是由钢、塑料和玻璃组成。

# 识别对象之间的关系

- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖

## 2、审查与筛选

（1）是否属于问题域？

例：公司职员与家庭

（2）是不是系统责任的需要？

例：公司与工会

（3）部分对象是否有一个以上的属性？

例：汽车与轮胎（规格）

（4）是否有明显的整体-部分关系？

例：学生与课程

## 3、调整对象层和属性层

定义聚合关系的活动可能发现一些新的对象类，或者从整体对象的类定义中分割出一些部分对象的类定义，应把它们加入到对象层中，并给出它们的详细说明。

识别对象之间的关系

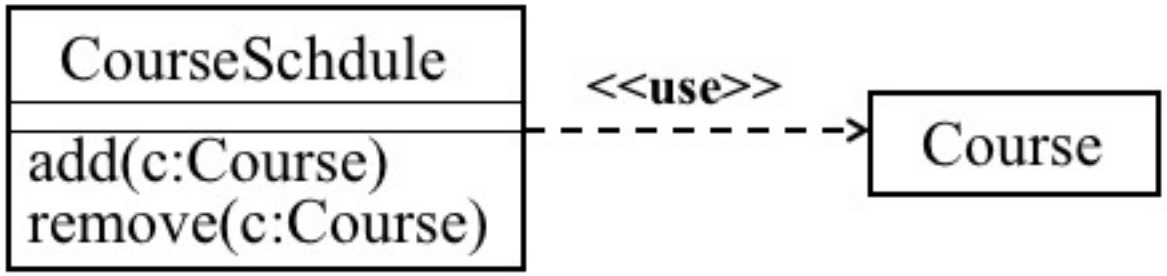
- 识别继承（泛化）
- 识别关联
- 识别聚合
- 识别依赖

# 四、识别依赖


依赖是一种使用关系，用于描述一个事物（如类Window）使用另一事物（如类Event）的信息和服务。

❶在大多数情况里，使用依赖来描述一个类使用另一个的操作；

❷如果被使用的类发生变化，那么另一个类的操作也会受到影响；




建议：在初步建立类之间的关系时，可以暂时使用依赖。在最终的类图中，若能用其他关系明确地指明类之间关系的含义，就不要使用依赖。