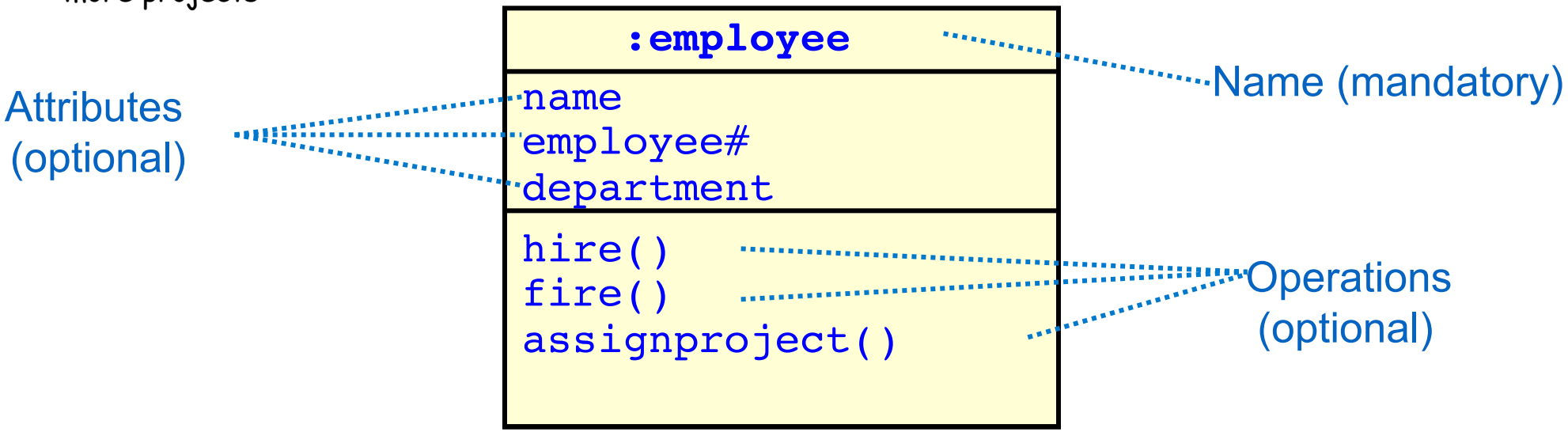
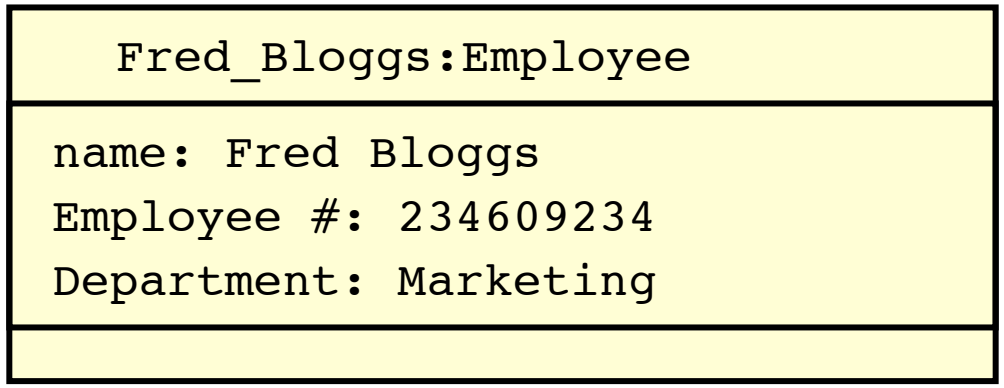
什么是类?

- 类是具有以下特征的对象集合
  - 相同性质(attributes)
  - 相同行为(operations)
  - 相同的对象关系
  - 相同语义(“semantics”)
- 举例
  - employee: has a name, employee# and department; an employee is hired, and fired; an employee works in one or more projects



对象

- 对象是类的实例，表示为：



  - 两个不同的对象可以有相同的属性取值(正如两个同名的人，或者住在同一栋楼的人)

- 对象与其他对象之间发生关联关系

  例如：**Fred_Bloggs:employee** 对象与 **KillerApp:project** 对象相关联

  但这个关联关系要定义为类之间的关系（为什么？）

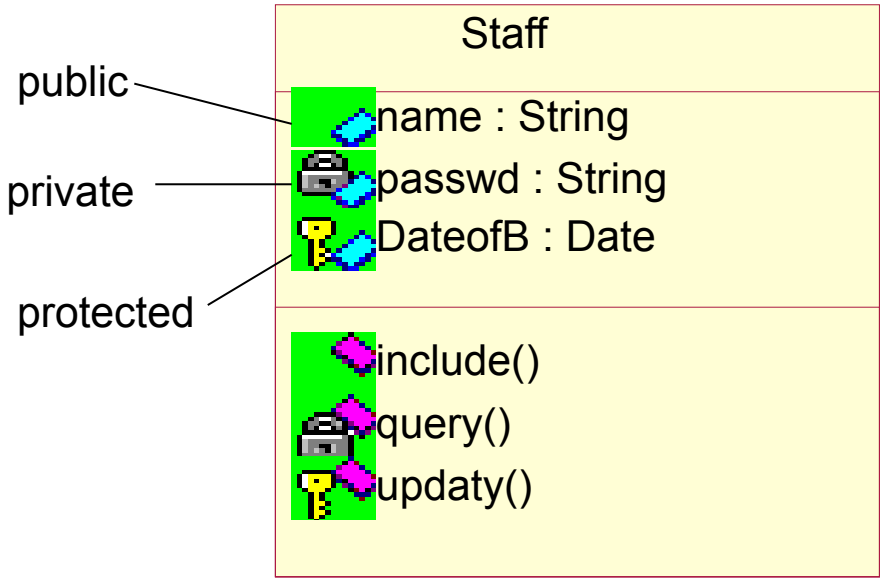
- 注意将属性划归正确的类

  例如：不要将经理姓名和员工编号同时定义为project类的属性 （…为什么？）

类属性定义

- 属性在类图标的属性分隔框中用文字串说明，UML规定属性的语法为：

  [可见性] 属性名 [:类型] [[多样性 [次序]]] [= 初始值] [{约束}]



例：一些属性声明的例子。

+size: Area = (100,100)

#visibility: Boolean = false

+default-size: Rectangle

#maximum-size: Rectangle

-xptr: XwindowPtr

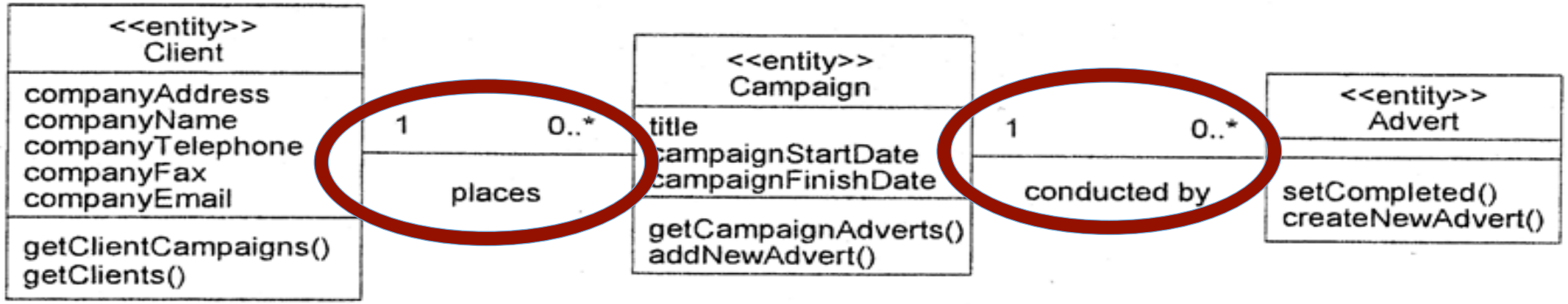
~colors : Color [3]

points : Point [2..*] {ordered}

name : String [0..1]

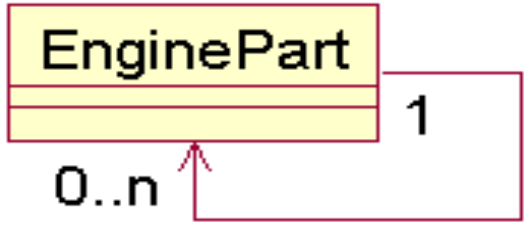
类关系

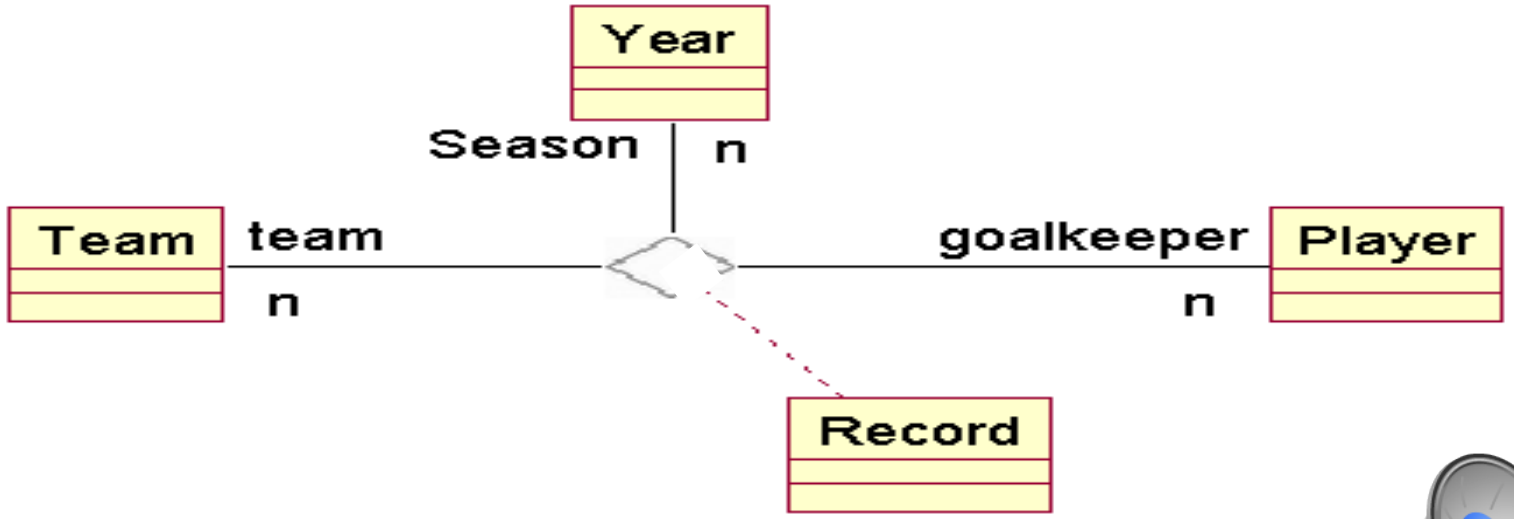
- 对象并非遗世独立，对象间存在千丝万缕的联系
  - UML中，关注以下几种类型的关系：
    - 关联关系（Association）
      - 聚合与组合关系（Aggregation and Composition）
    - 泛化关系（Generalization）
    - 依赖关系（Dependency）
    - 实现关系（Realization）
- 类图描述类和他们之间的关系



关联关系的种类

- 按照关联所连接的类的数量，对象类之间的关联可分为：自返关联，二元关联，N元关联
- 自返关联(reflexive association)又称递归关联(recursive association)，是一个类与本身的关联，即同一个类的两个对象间的联系。
- 二元关联(binary association)：二元关联是在两个类之间的关联。
- N元关联(n-ary association): 是在3个或3个以上类之间的关联。

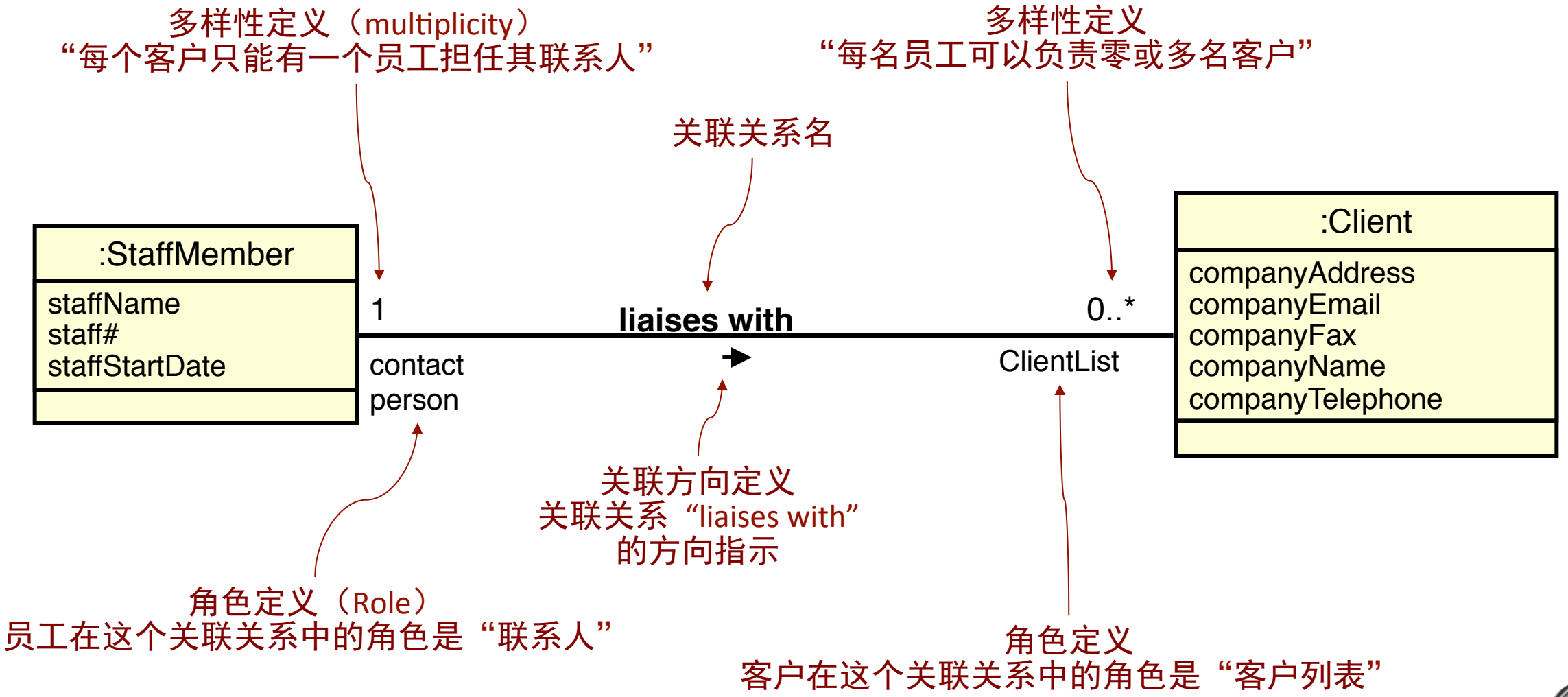




关联关系的“多样性/维度”（Multiplicity）

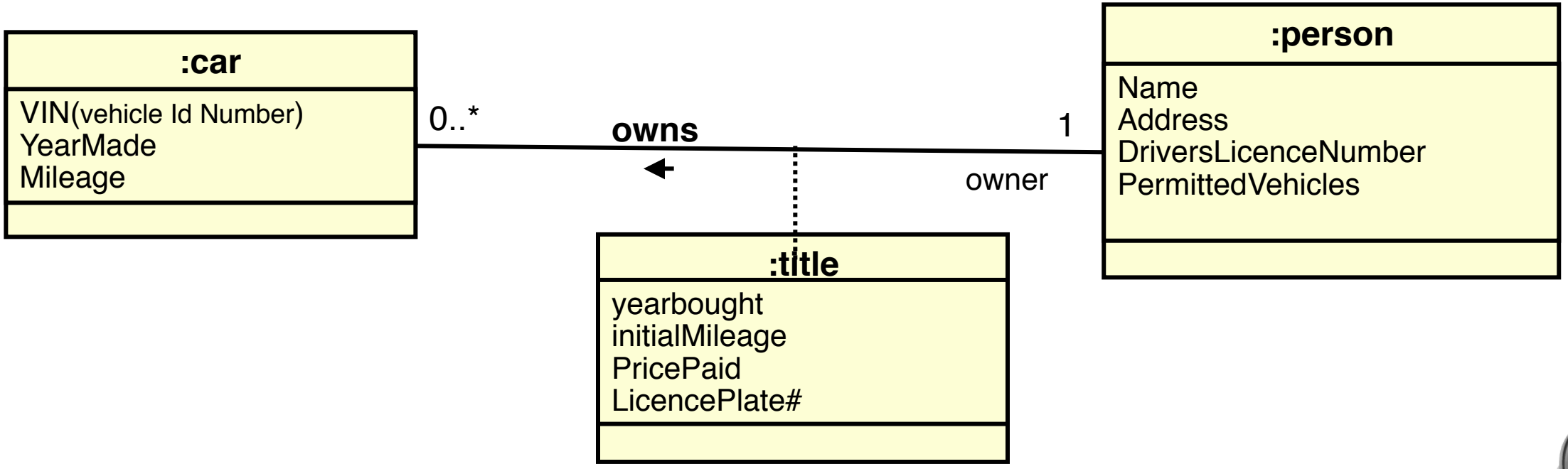
- 定义上页的关联关系时尝试回答以下问题：
  - **市场活动（Campaign）类对象能否在没有员工（StaffMember）类对象管理的时候独立存在？**
    - 如果可以，则员工（StaffMember）类的一端是可选的，多样性/维度定义为 0..*
    - 如果不行，则员工（StaffMember）类是必选的，多样性/维度定义为 1..*
    - 如果必须有，且只能由一个Staff来管理，多样性/维度定义为 1
  - **然后再就另一端问同样的问题**
    - 每位员工都必须管理，且只能管理一项市场活动吗？
    - 如果不是，则市场活动一端的多样性/维度定义为0..*
- 其他的多样性定义举例：
  - 可选的（Optional，0 or 1） 0..1
  - 有且仅有一个（Exactly one，1） 1..1
  - 零或多个（Zero or more） 0..*，*，
  - 一或多个（One or more） 1..*
  - 取值范围（A range of values） 2..6

关联关系图例

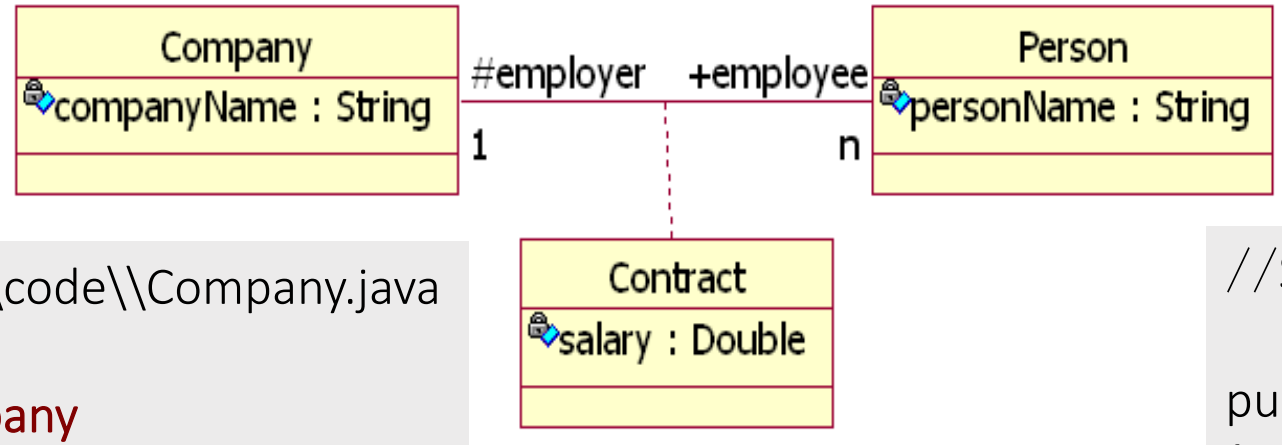

多样性定义（multiplicity）
“每个客户只能有一个员工担任其联系人”
多样性定义
“每名员工可以负责零或多名客户”
关联关系名
:StaffMember
staffName
staff#
staffStartDate
1
liaises with
0..*
contact
person
ClientList
:Client
companyAddress
companyEmail
companyFax
companyName
companyTelephone
关联方向定义
关联关系“liaises with”
的方向指示
角色定义（Role）
员工在这个关联关系中的角色是“联系人”
角色定义
客户在这个关联关系中的角色是“客户列表”

关联类

- 有时要为关联相关信息的存储定义一个专门的类，称为“关联类”
  - …保存与关联关系本身相关的信息
  - …这些信息不属于关联所连接的两端的类
  - 例如，下图中，“title”类的对象中存储的是车主和车辆之间所属关系有关的信息



- 关联类所生成Java代码示例：



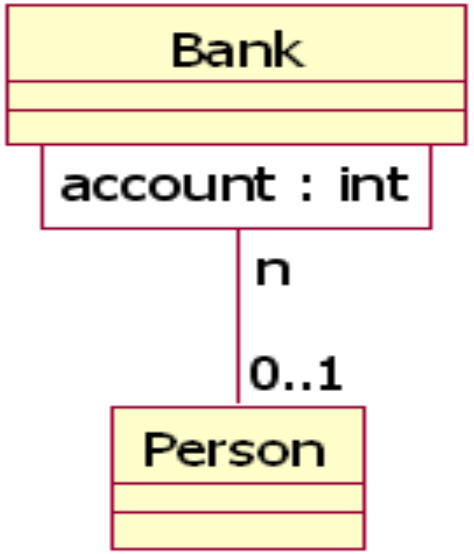

```
//Source file: F:\\code\\Company.java

public class Company
{
   private String companyName;
   public Person employee[];
}
```

```
//Source file: F:\\code\\Contract.java

public class Contract
{
   private Double salary;
}
```

```
//Source file: F:\\code\\Person.java

public class Person
{
   private String personName;
   protected Company employer;
}
```

限定关联 （Qualifier）

在关联端紧靠源类图标处可以有限定符(Qualifier)。带有限定符的关联称为限定关联(Qualified Association)。



- 例：右图是使用限定符的例子。

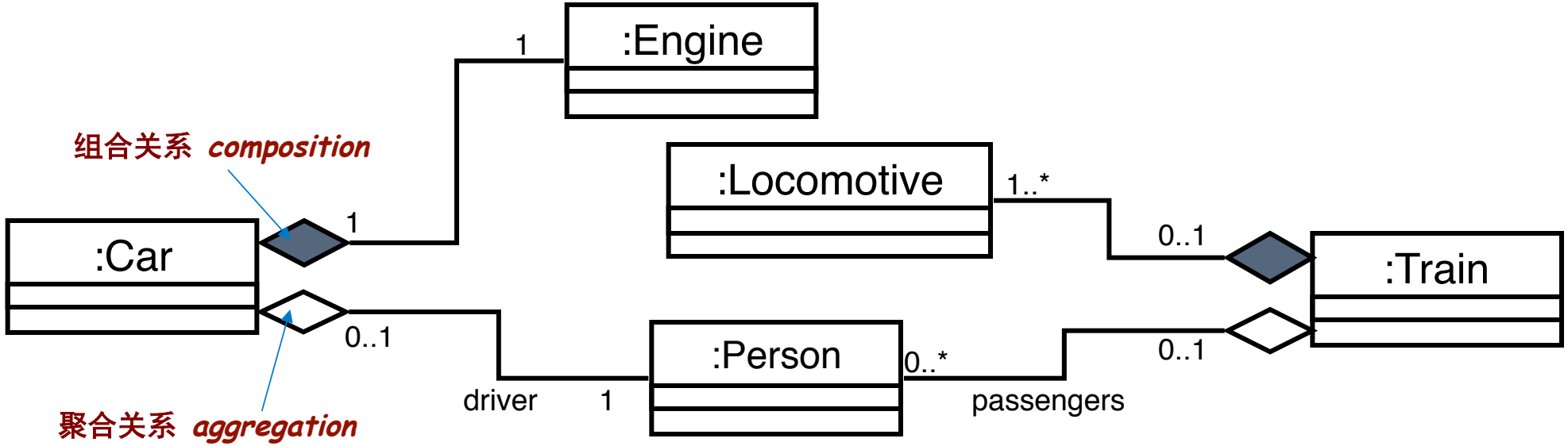
（bank, account） → 0 or 1 person
person → many (bank, account)

说明：

1. 限定符是关联的属性。
2. 限定符的作用是，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。

聚合（Aggregation）与组合（Composition）关系

- 聚合（Aggregation）用于表达一个整体对象与其成员对象之间的关系
  - “**Has-a**” 或是 “**Whole/part**”
- 组合（Composition）用于表达一个整体对象与其组成部分之间的关系
  - 组合关系所表达的整体类与部分类之间的所属关系更强
    - 整体类的对象不存在时，部分类的对象也不存在
    - 整体类对象撤销之前要负责将部分类对象撤销



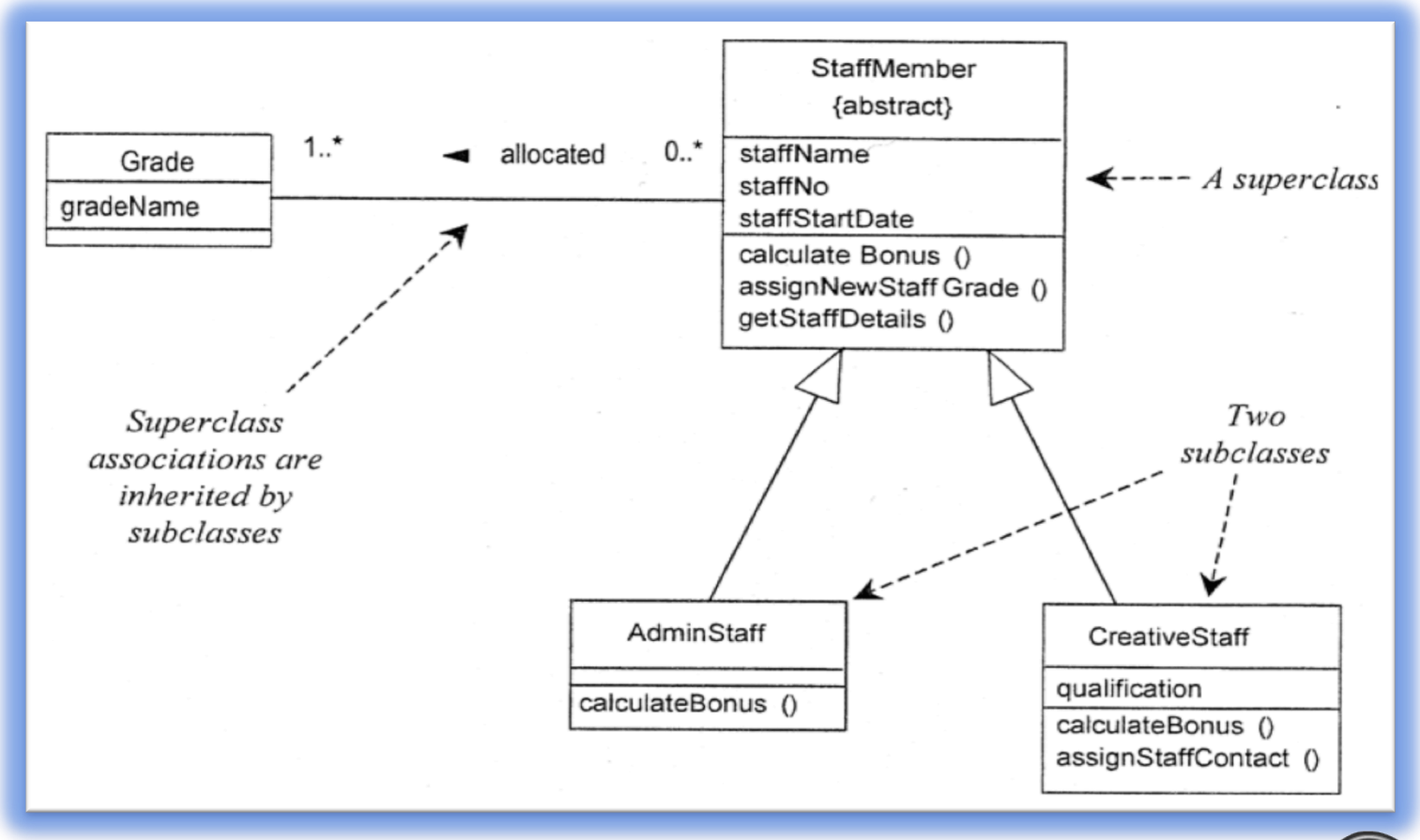
继承/泛化（Inheritance/Generalization）

- 子类继承父类的属性、关联和操作
- 子类可以覆盖继承来的内容

  例如：行政员工**AdminStaff**和专业设计人员**CreativeStaff**的奖金计算方法可以不同

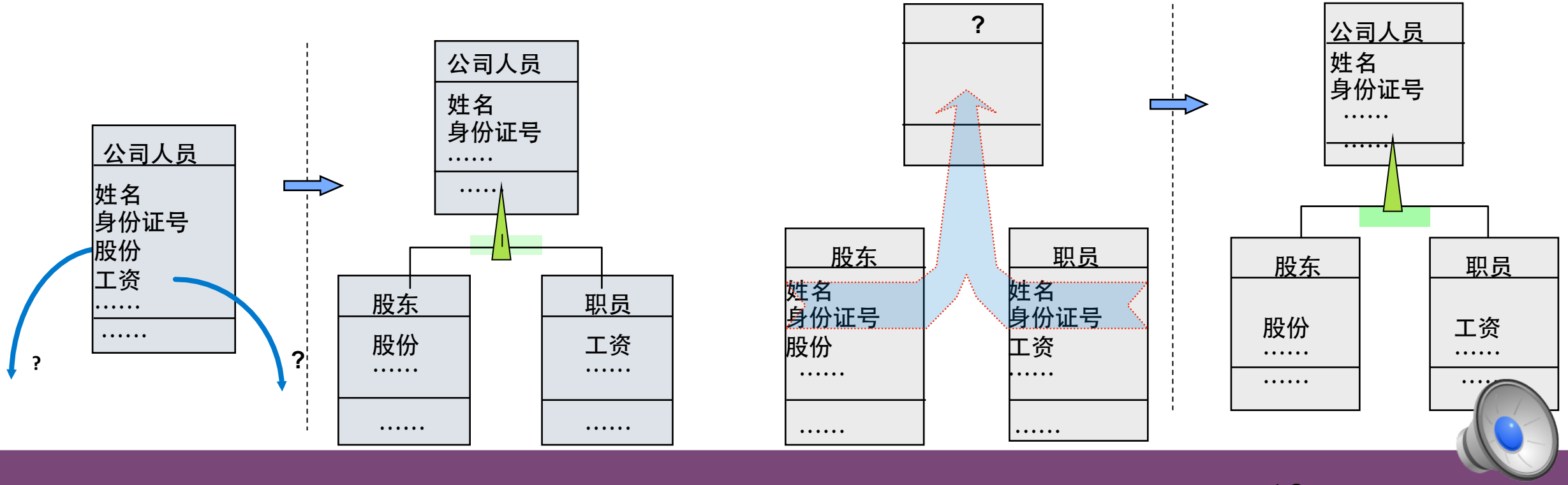
- 父类可以声明为抽象类{**abstract**}，则将不会为它直接创建实例对象
  - 意味着现有子类覆盖了该类对象的全集

  例如，除了行政员工**AdminStaff**和专业设计人员**CreativeStaff**之外，不存在其他类员工

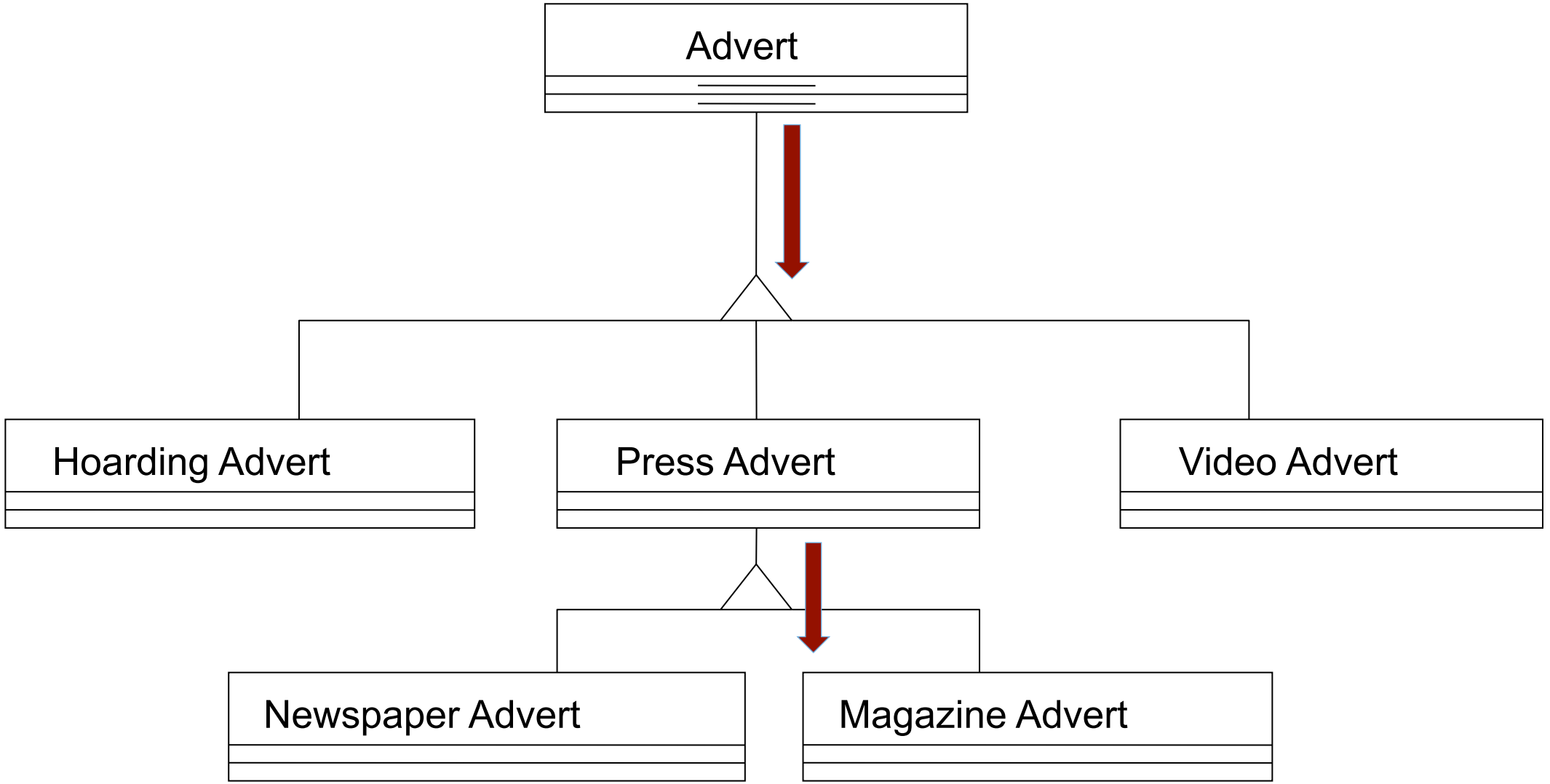


继承/泛化(Inheritance/Generalization)关系的定义

- 继承/泛化关系建模的意义在于系统环境发生变化时便于添加新的子类
- 继承/泛化关系建模的过程
  - **自顶向下**
    - 将某个类分割为属性和操作不同的子类，或者发现关联关系定义的是分类关系“kind of”
  - **自底向上**
    - 为现有的多个具有公共属性及方法的类，定义一个父类



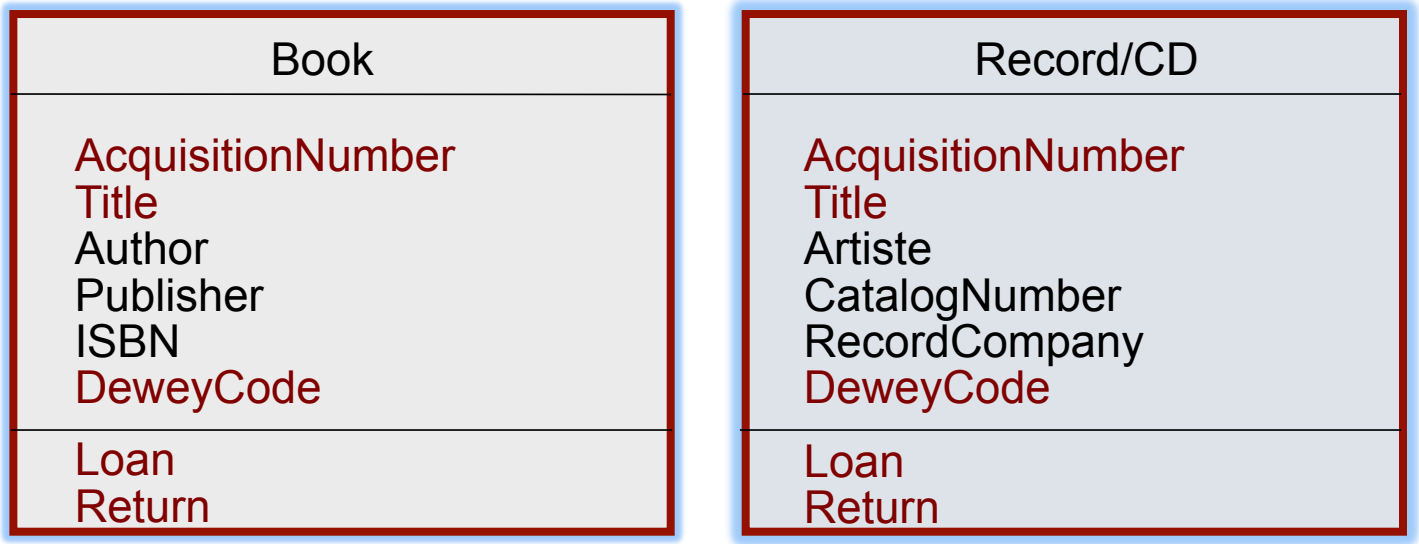
自顶向下定义继承关系

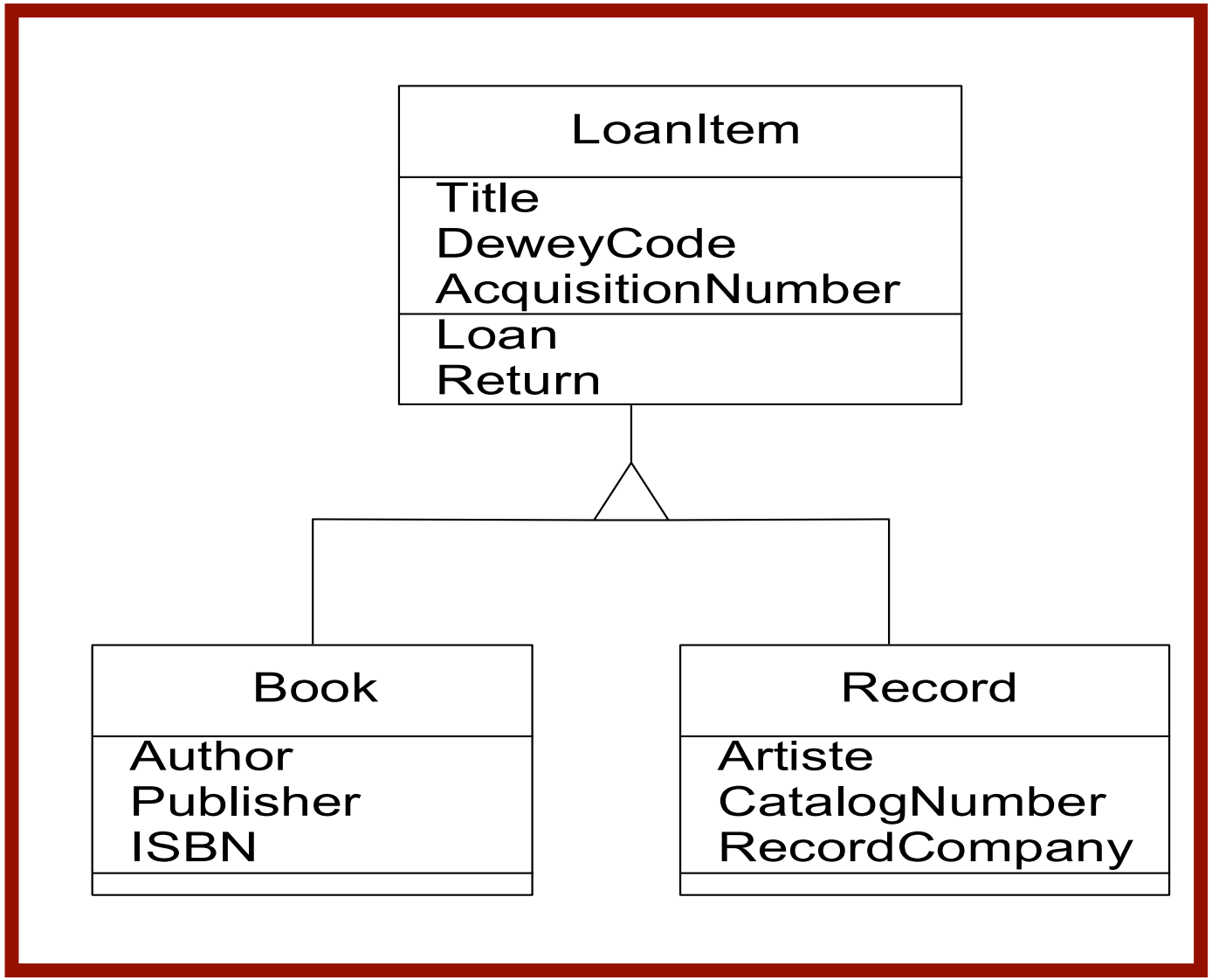

Advert
Hoarding Advert
Press Advert
Video Advert
Newspaper Advert
Magazine Advert

自底向上定义继承关系

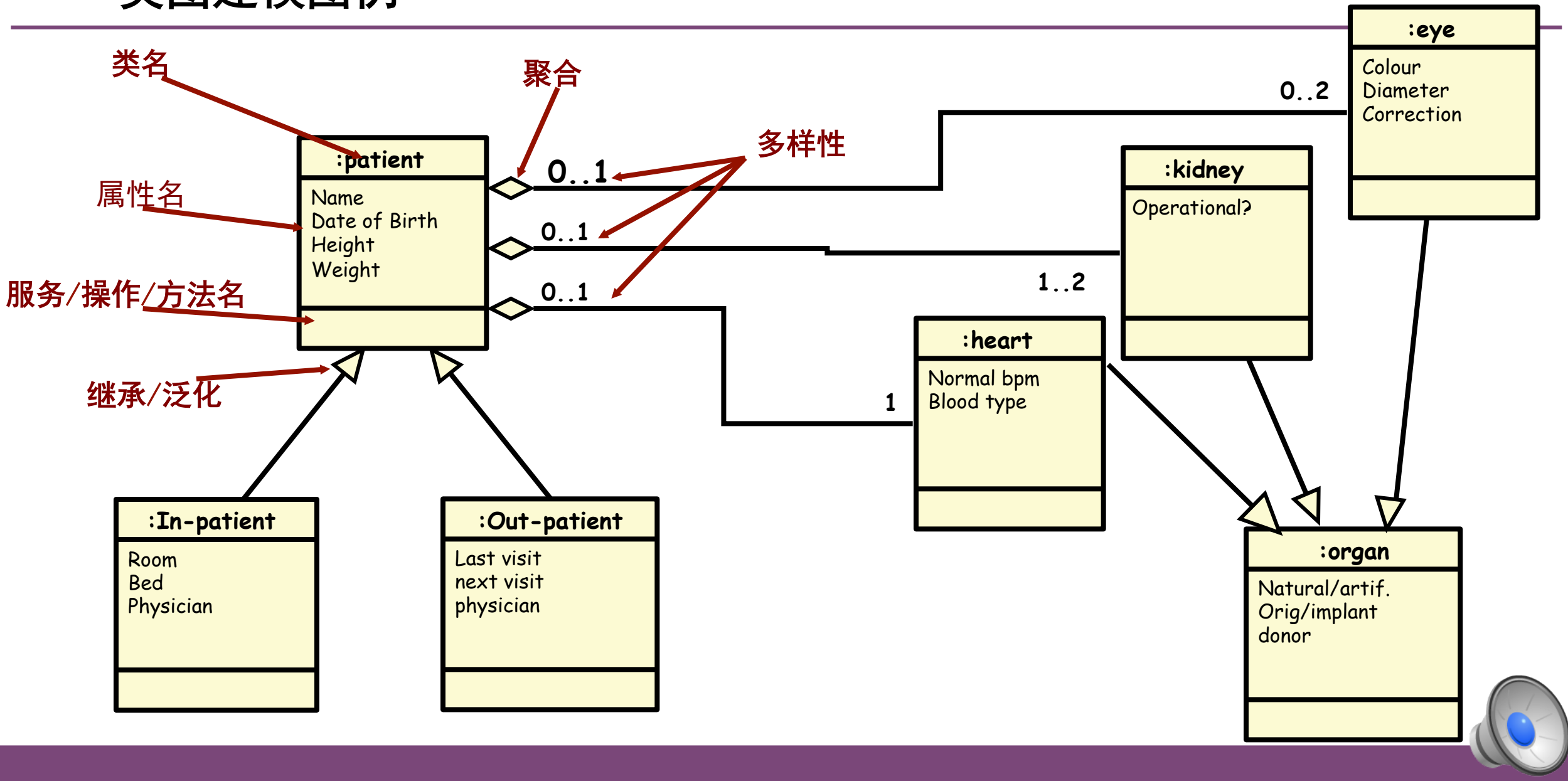
为右图的两个类定义公共父类，

并重画该类图




LoanItem
Title
DeweyCode
AcquisitionNumber
Loan
Return
Book
Author
Publisher
ISBN
Record
Artiste
CatalogNumber
RecordCompany

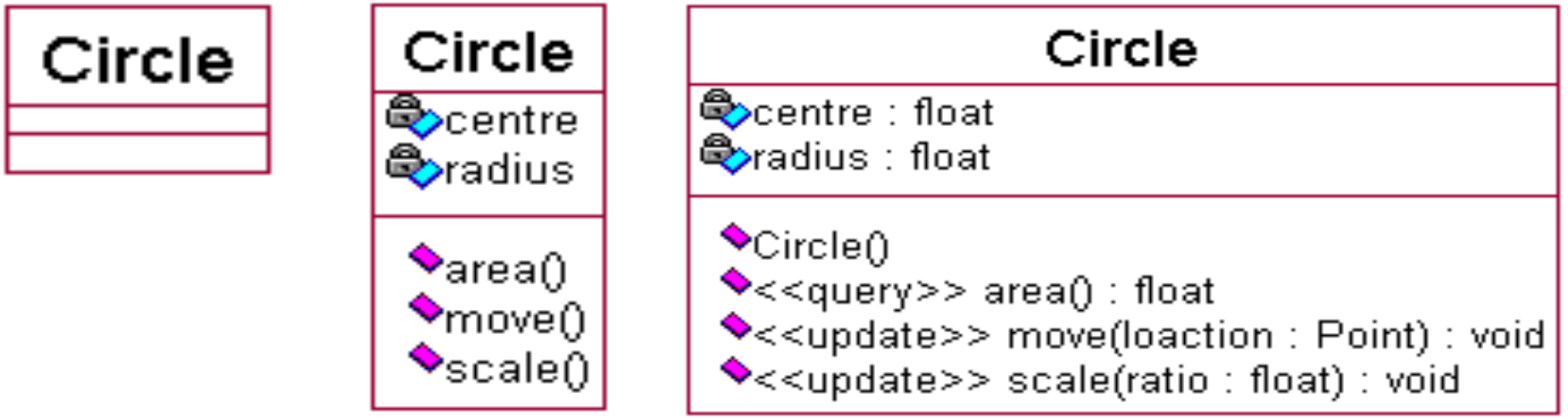
类图建模图例


类名
聚合
多样性
属性名
服务/操作/方法名
继承/泛化
:patient
Name
Date of Birth
Height
Weight
0..1
0..1
0..1
0..2
1..2
1
:eye
Colour
Diameter
Correction
:kidney
Operational?
:heart
Normal bpm
Blood type
:In-patient
Room
Bed
Physician
:Out-patient
Last visit
next visit
physician
:organ
Natural/artif.
Orig/implant
donor

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。一般地，类图可分为三个层次，即概念类，设计说明类和实现类。
- 概念类，设计说明类和实现类的划分最先是由 Steve Cook 和 John Daniels 引入的。

例：类图的三个层次的例子。



概念类　　设计说明类　　实现类

总结建立类图的步骤

1. 研究分析问题领域，确定系统的需求。
2. 发现对象与类，明确它们的含义和责任，确定属性和操作。
3. 发现类之间的关系。把类之间的关系用关联、泛化、聚集、组合、依赖等关系表达出来。
4. 设计类与关系。调整和细化已得到的类和类之间的关系，解决诸如命名冲突、功能重复等问题。
5. 绘制类图并编制相应的说明。

类图建模风格

1：属性名和类型应该一致
2：不要对有关联类的关联命名。
3：静态操作/属性要在实例操作/属性之前列出。
4：以可见性降低的次序列出操作/属性。
5：避免已被语言的命名规范所隐含的版型。
6：总是指明多样性
7：不要对每个依赖关系都建模
8：将子类放在超类的下方
9：小心基于数据的继承
10：按惯例是把整体画在部分的左边