重庆大学本科课程

面向对象技术与 UML

Object-Oriented Technique and UML

重庆大学软件工程学院

教材目录 《面向对象技术UML教程》

教材目录 《UML基础、案例与应用（第三版）》

第一部分 基础知识

第二部分 学习案例

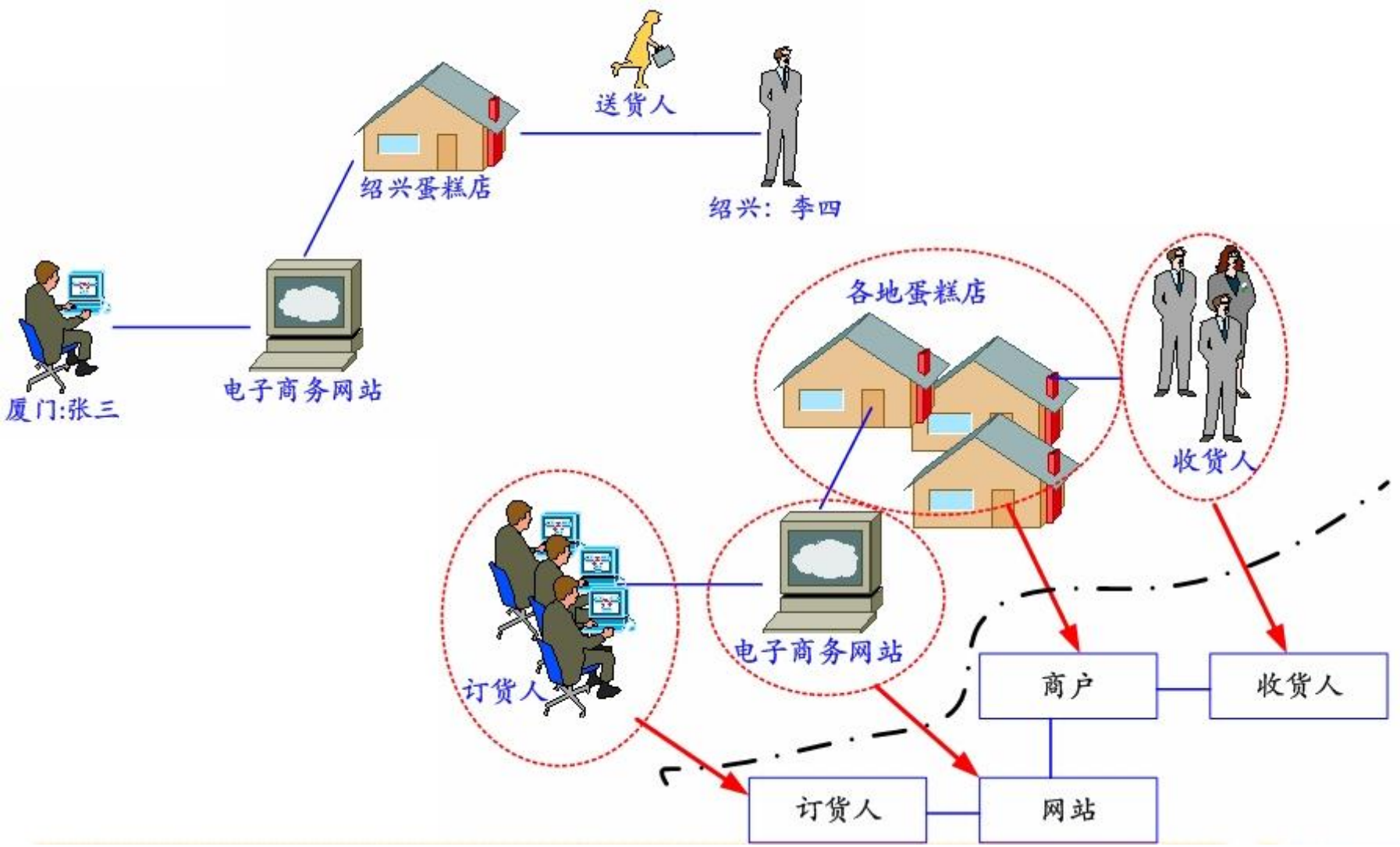
第三部分 高级应用

第4章　关系

- 如何对类之间的关系建模
- 如何可视化类和子类的关系
- 如何表现类之间的依赖

- 本章需要掌握的内容：
  - 什么是类图？类图的UML表示方法？
  - 类之间的关系的UML如何表示？
    - ✓ 关联
    - ✓ 约束
    - ✓ 关联类
    - ✓ 多重性
    - ✓ 限定关联
    - ✓ 自身关联
  - 类与代码如何转换？

类的定义



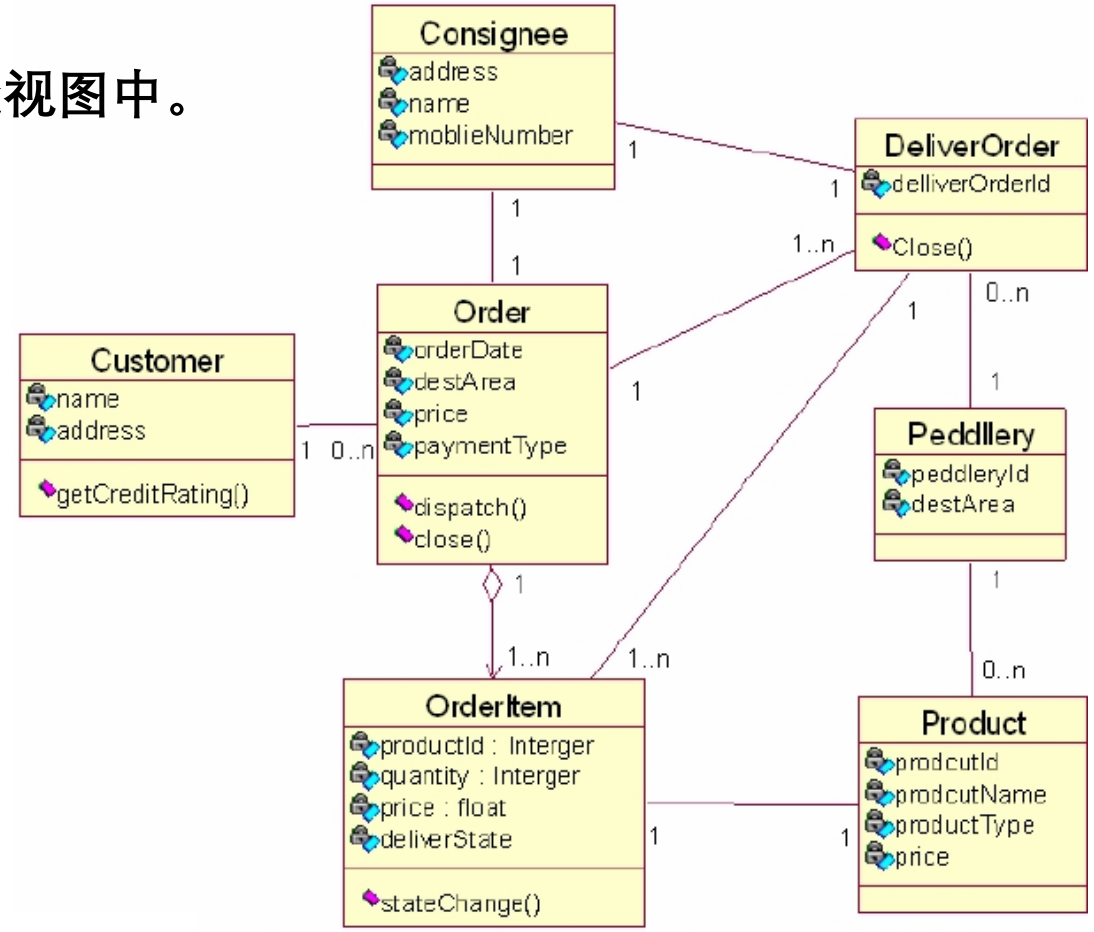
类的定义

面向对象思想

- 每个对象都扮演了一个角色，并为其它成员提供特定的服务或执行特定的行为。
- 在面向对象世界中，行为的启动是通过将“消息”传递给对此行为负责的对象来完成的；同时还将伴随着执行要求附上相关的信息（参数）；而收到该消息的对象则会执行相应的“方法”来实现需求用类和对象表示现实世界，用消息和方法来模拟现实世界的核心思想。

类图

- UML用类图（class diagram）表示类、接口及其关联。
- 类图用于静态对象建模。
- 类图应用在领域建模和概念透视图中。



类图

- 类图(Class diagram)主要用于描述系统的结构化设计。
- 类图包含类和类之间的关系。
- 用类图可以显示出类、接口以及它们之间的静态结构和关系。
- 在系统中，每个类都具有一定的职责，职责指的是类要完成什么样的功能，要承担什么样的义务。
- 类的属性即类的数据职责，类的操作即类的行为职责。设计类是面向对象设计中最重要的组成部分，也是最复杂和最耗时的部分。

A simple model of interacting objects

objects

attributes

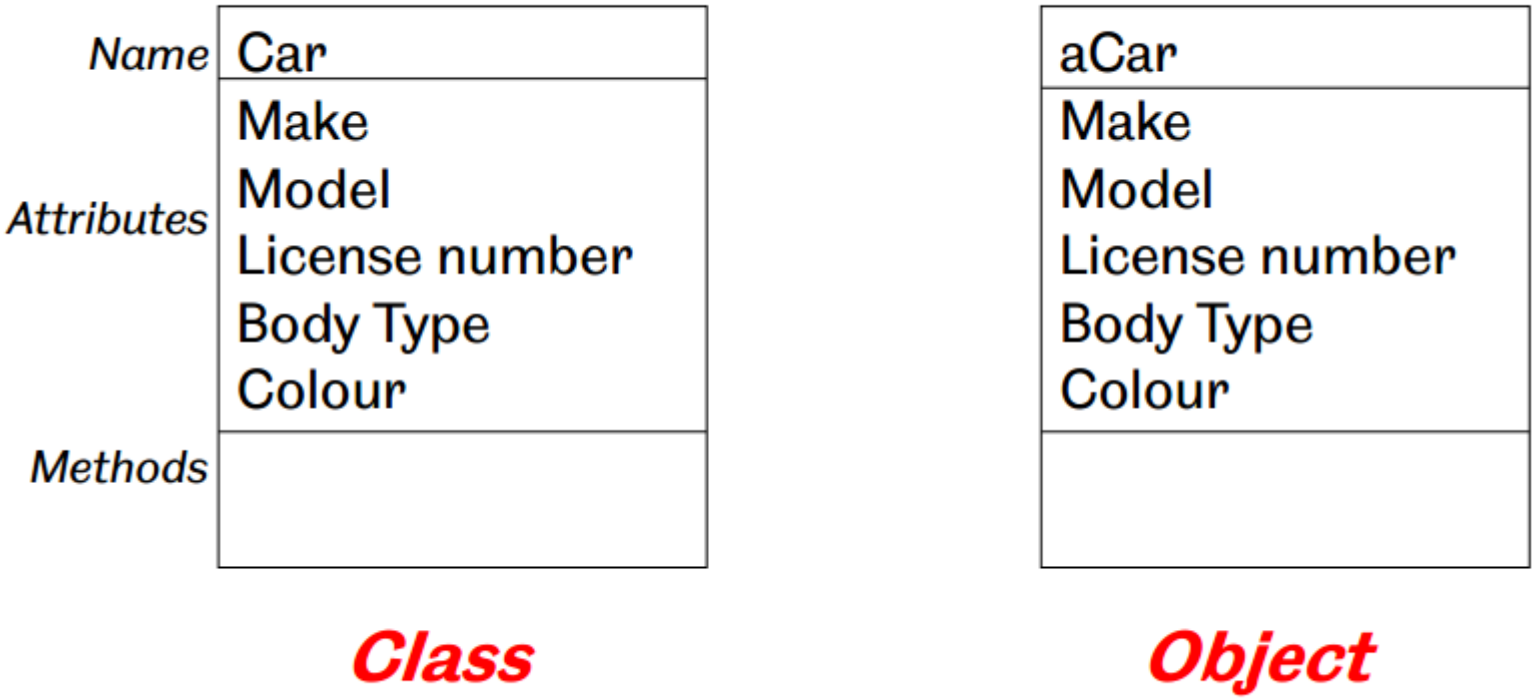
relationships

Name
Address
Age
Drivers license number

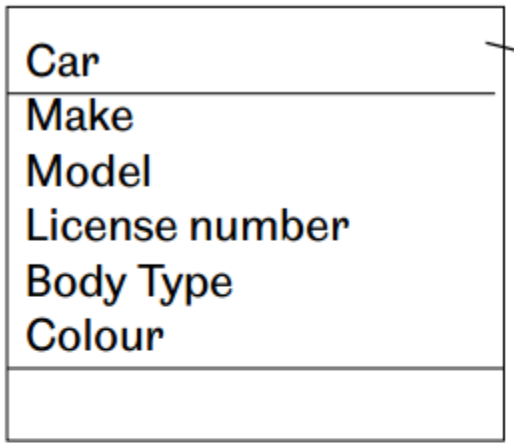
owns

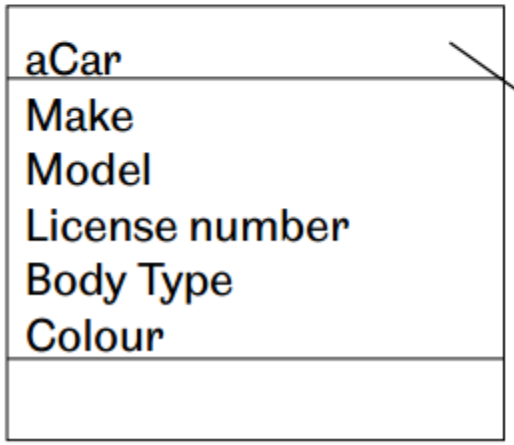
Make
Model
License plate number
Body type
Colour

实例化的类


Name
Car
Attributes
Make
Model
License number
Body Type
Colour
Methods
aCar
Make
Model
License number
Body Type
Colour
Class
Object

程序中如何表示类和对象？



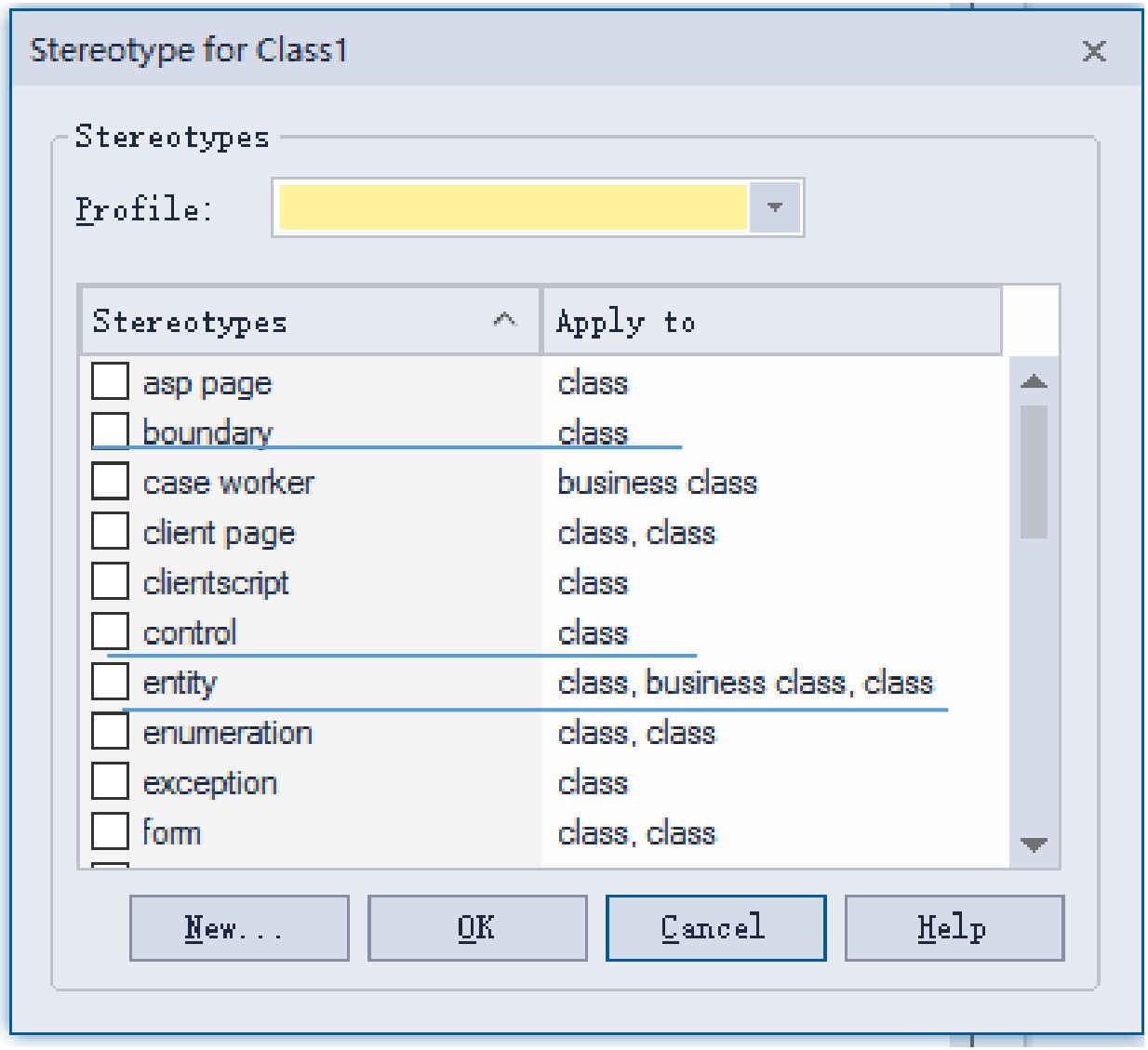



```
C++ CODE
class car{
 public:
    char * Make;
    char * Model;
    int License_number;
    char * Body_type;
    char * Color
     ...
};
...
 void main(void){
    ...
    car aCar;
    ...
 }
```

类的种类

- 类通常分为三种：实体类(Entity Class)、控制类(Control Class)和边界类(Boundary Class)
  - ① 实体类：实体类对应系统需求中的每个实体，它们通常需要保存在永久存储体中，一般使用数据库表或文件来记录，实体类既包括存储和传递数据的类，还包括操作数据的类。实体类来源于需求说明中的名词，如学生、商品等
  - ② 控制类：控制类用于体现应用程序的执行逻辑，提供相应的业务操作，将控制类抽象出来可以降低界面和数据库之间的耦合度。控制类一般是由动宾结构的短语（动词+名词）转化来的名词，如增加商品对应有一个商品增加类，注册对应有一个用户注册类等。
  - ③ 边界类：边界类用于对外部用户与系统之间的交互对象进行抽象，主要包括界面类，如对话框、窗口、菜单等。

在面向对象分析和设计的初级阶段，通常首先识别出实体类，绘制初始类图，此时的类图称为领域模型，包括实体类及其它们之间的相互关系。

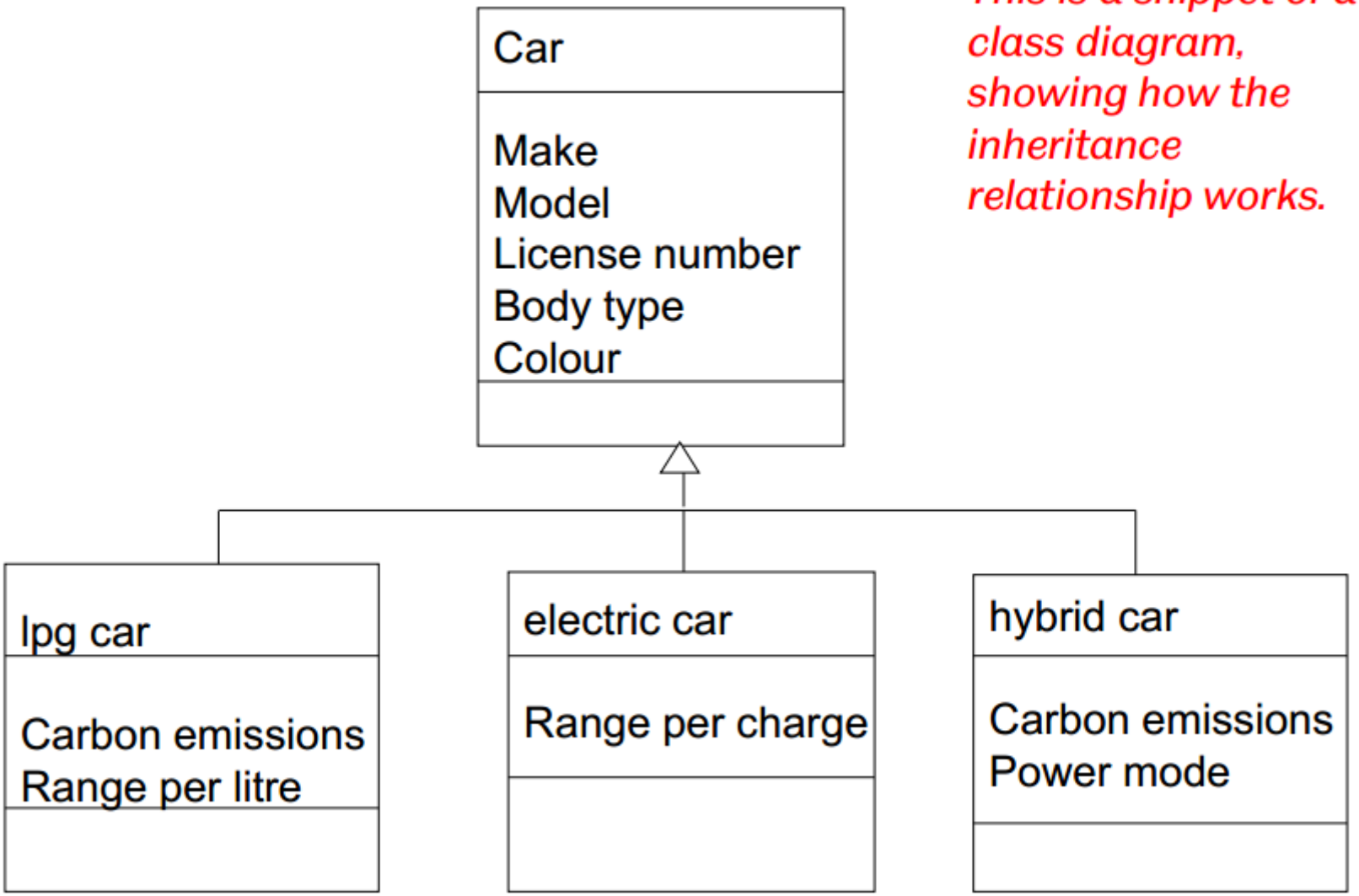

Stereotype for Class1
Stereotypes
Profile:
Stereotypes
Apply to
asp page
class
boundary
class
case worker
business class
client page
class, class
clientscript
class
control
class
entity
class, business class, class
enumeration
class, class
exception
class
form
class, class
New...
OK
Cancel
Help

如何寻找类？

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。

请找出其中的类，并定义相关的操作和类。

类图



This is a snippet of a class diagram, showing how the inheritance relationship works.

关联

◇ 什么是关联（**association**）

当类之间在概念上有连接关系时，类之间的连接叫做关联

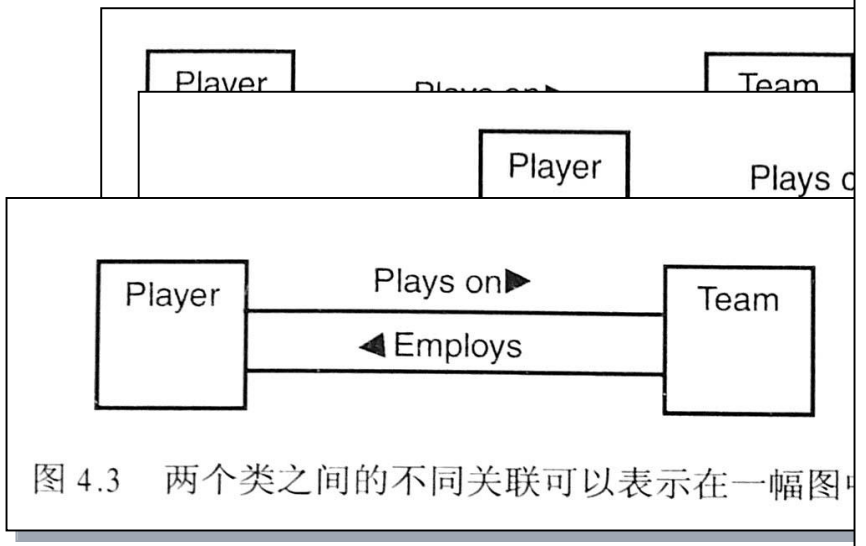


图 4.3　两个类之间的不同关联可以表示在一幅图中

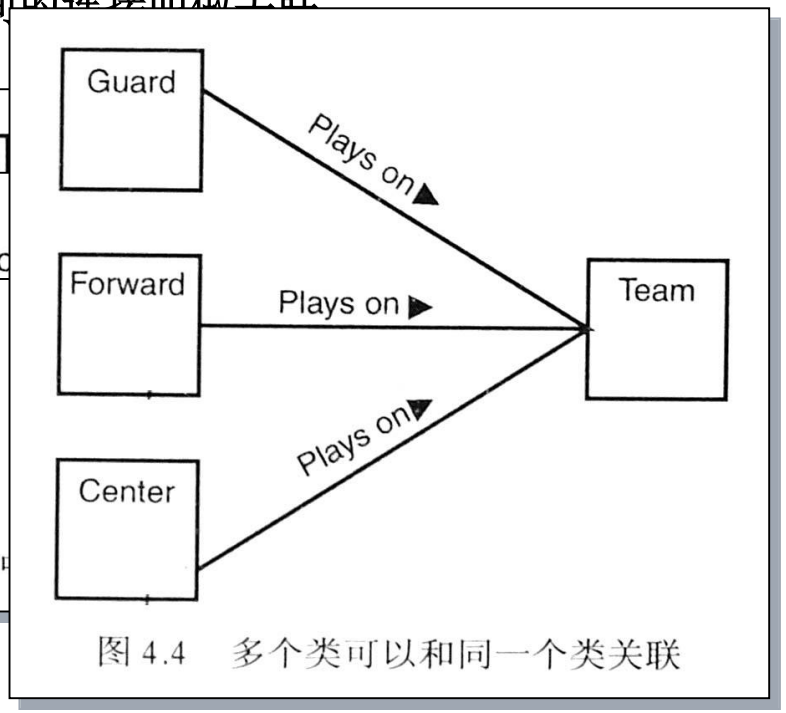


图 4.4　多个类可以和同一个类关联

◇ 符号特征

* 用一条线连接两个类，并把关联的名字放在这个连线上
* 关联的方向用一个实心三角形箭头来指明

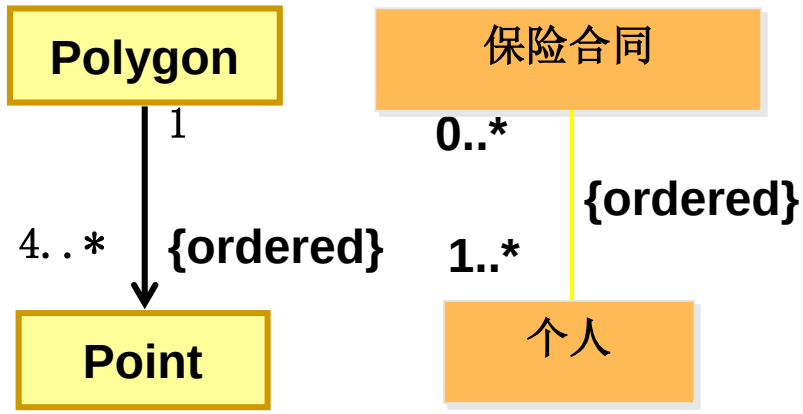
UML中提供了一种简便、统一和一致的约束（constraint），是各种模型元素的一种语义条件或限制。一条约束只能应用于同一类的元素。

◇约束的表示

如果约束应用于一种具有相应视图元素的模型元素，它可以出现在它所约束元素视图元素的旁边。

通常一个约束由一对花括号括起来（{constraint}），花括号中为约束内容

如果一条约束涉及同一种类的多个元素，则要用虚线把所有受约束的元素框起来，并把该约束显示在旁边（如或约束）。



关联

◇ 关联上的约束；关联类；链

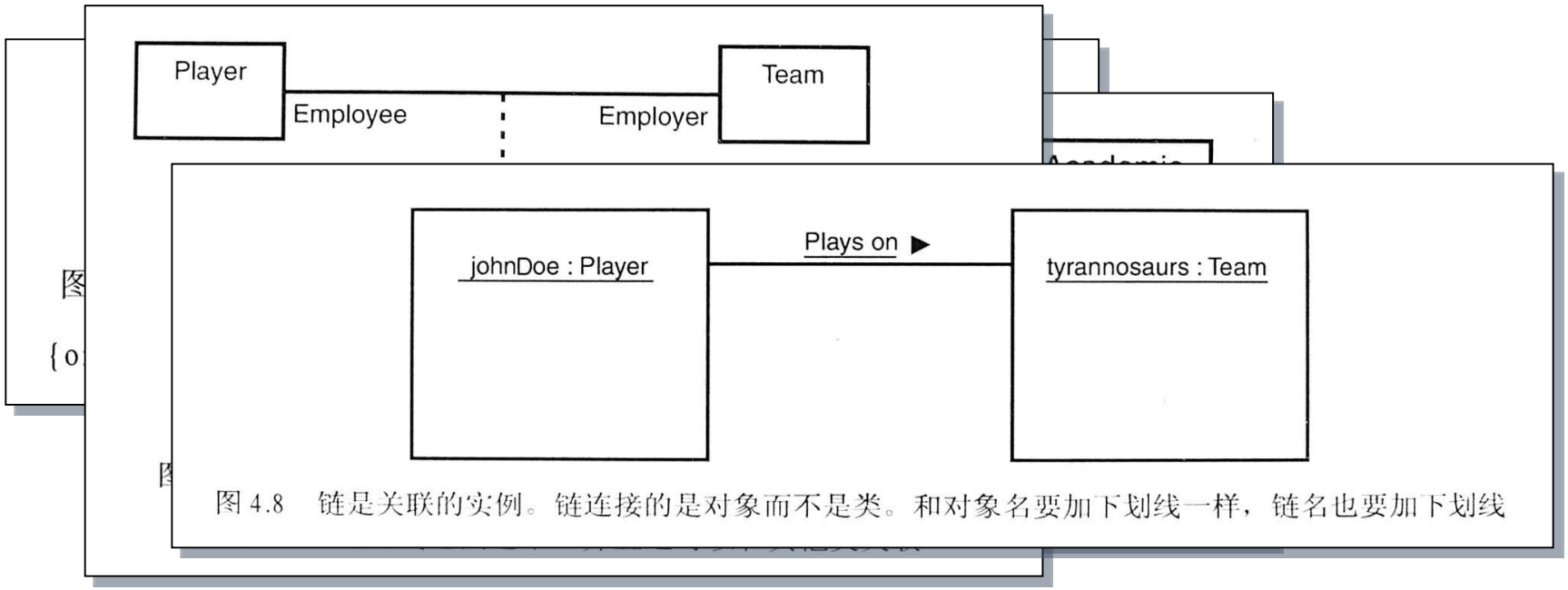


图 4.8　链是关联的实例。链连接的是对象而不是类。和对象名要加下划线一样，链名也要加下划线

◇ 符号特征

* 约束：花括号、虚线（或关系）
* 关联类：用虚线将关联类和对应的关联线连接起来
* 链：用一条线连接两个对象，并把链的名字放在这个连线上

多重性（Multiplicity）

◇ 什么是多重性（multiplicity）

某个类有多个对象可以和另一

图 4.9　多重性说明某个类的多少个对象

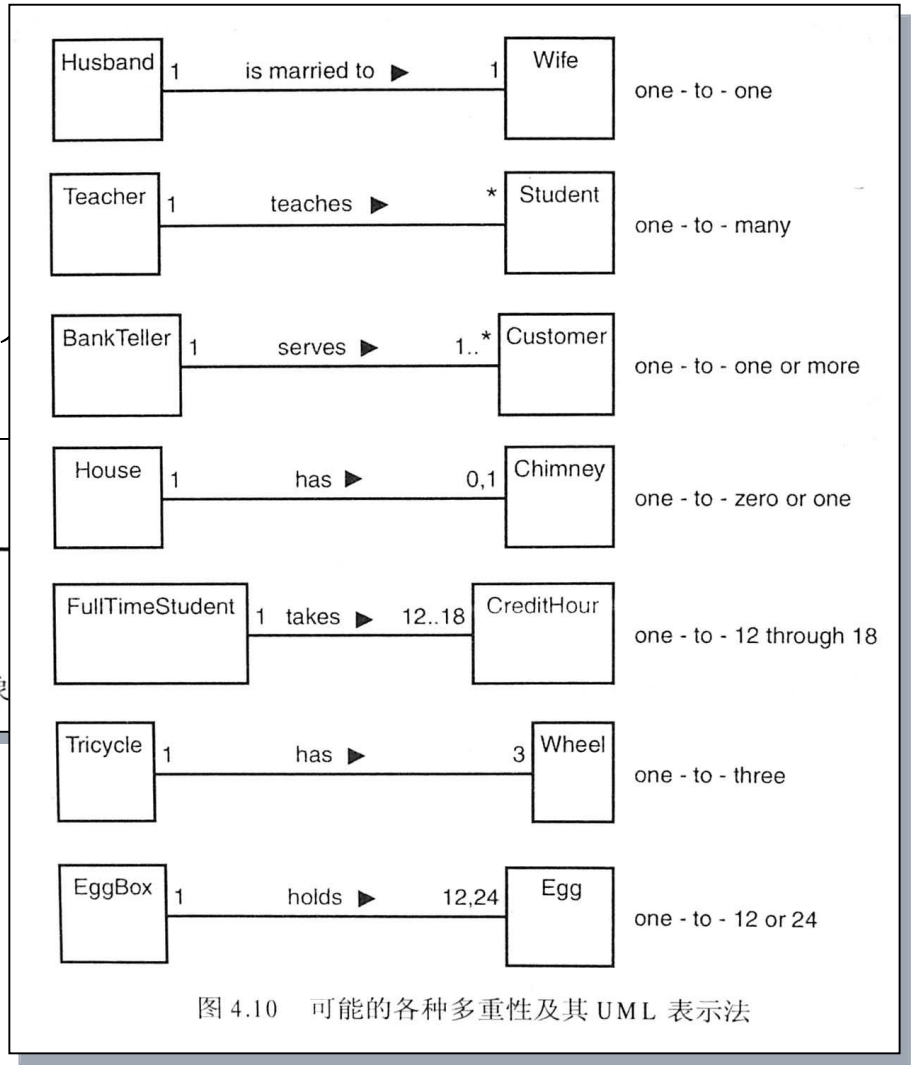


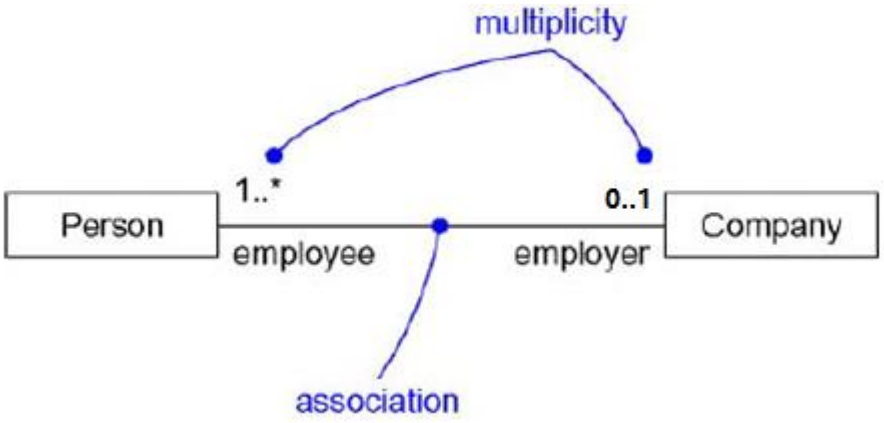
图 4.10　可能的各种多重性及其 UML 表示法

◇ 符号特征

* 在参与关联的类附近的关联线上注明多重性数值

* UML使用“*”来代表许多；“1..*”代表一个或多个；“,”代表或关系

多重性



Multiplicity	Meaning
n (default)	Many
0..0	Zero
0..1	Zero or one
0..n	Zero or more
1..1	Exactly one
1..n	One or more

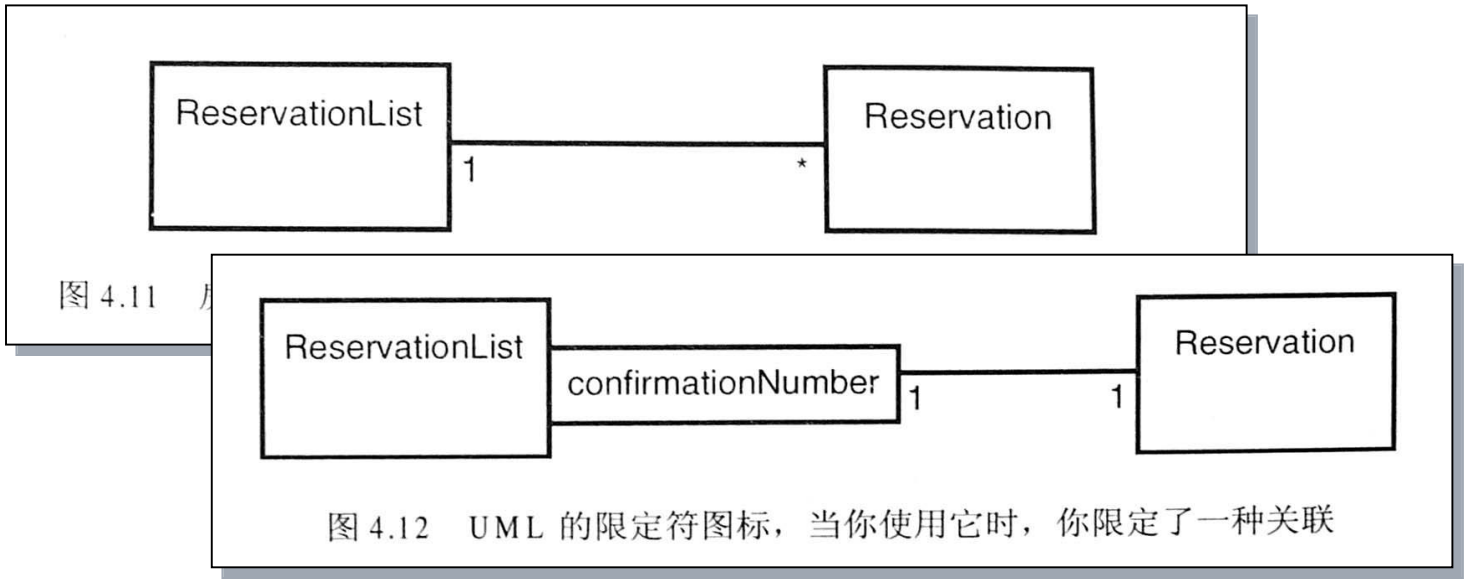
Format	Meaning
<number>	Exactly <number>
<number 1>..<number 2>	Between <number 1> and <number 2>
<number>..n	<number> or more
<number 1>,<number 2>	<number 1> or <number 2>
<number 1> , <number 2> .. <number 3>	Exactly <number 1> or between <number 2> and <number 3>
<number 1> .. <number 2> , <number 3> .. <number 4>	Between <number 1> and <number 2> or between <number 3> and <number 4>

```
public class Person {

    public Company employer;
    public Person(){

    }
}

public class Company {

    public Person[] employee;
    public Company(){

    }
}
```

限定关联

◇ 限定符（**qualifier**）

在**UML**中，标识符**ID**（**identification**）信息叫做限定符。



◇ 符号特征

＊一个小矩形框

自身关联

◇ 自身关联（**reflexive association**）

一个类可能与它自己发生关联，这样的关联被称为自身关联。

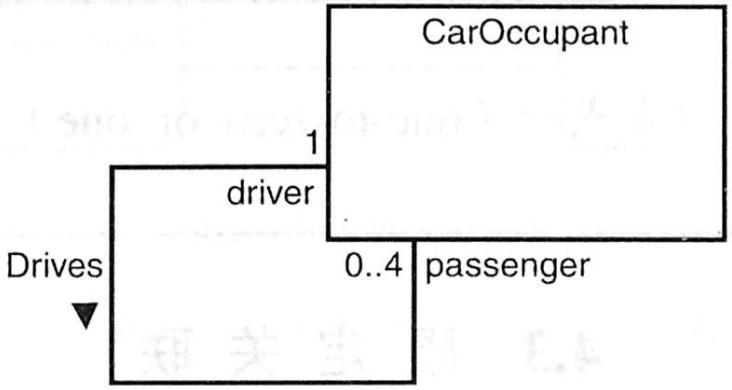
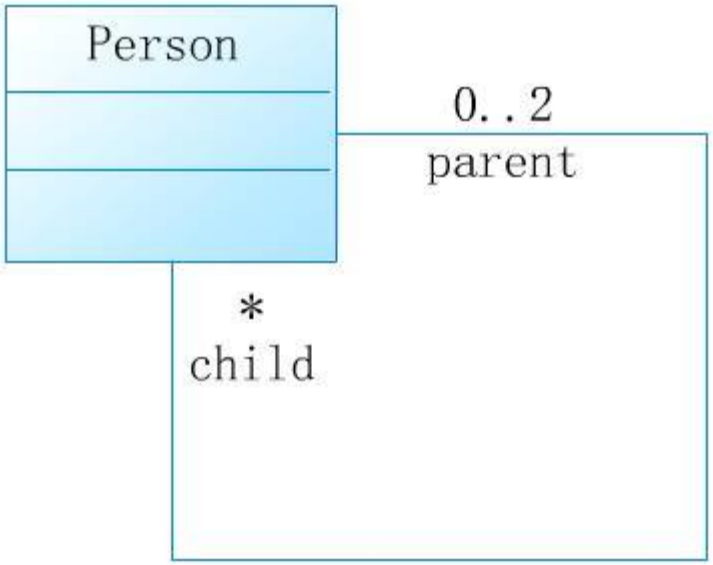


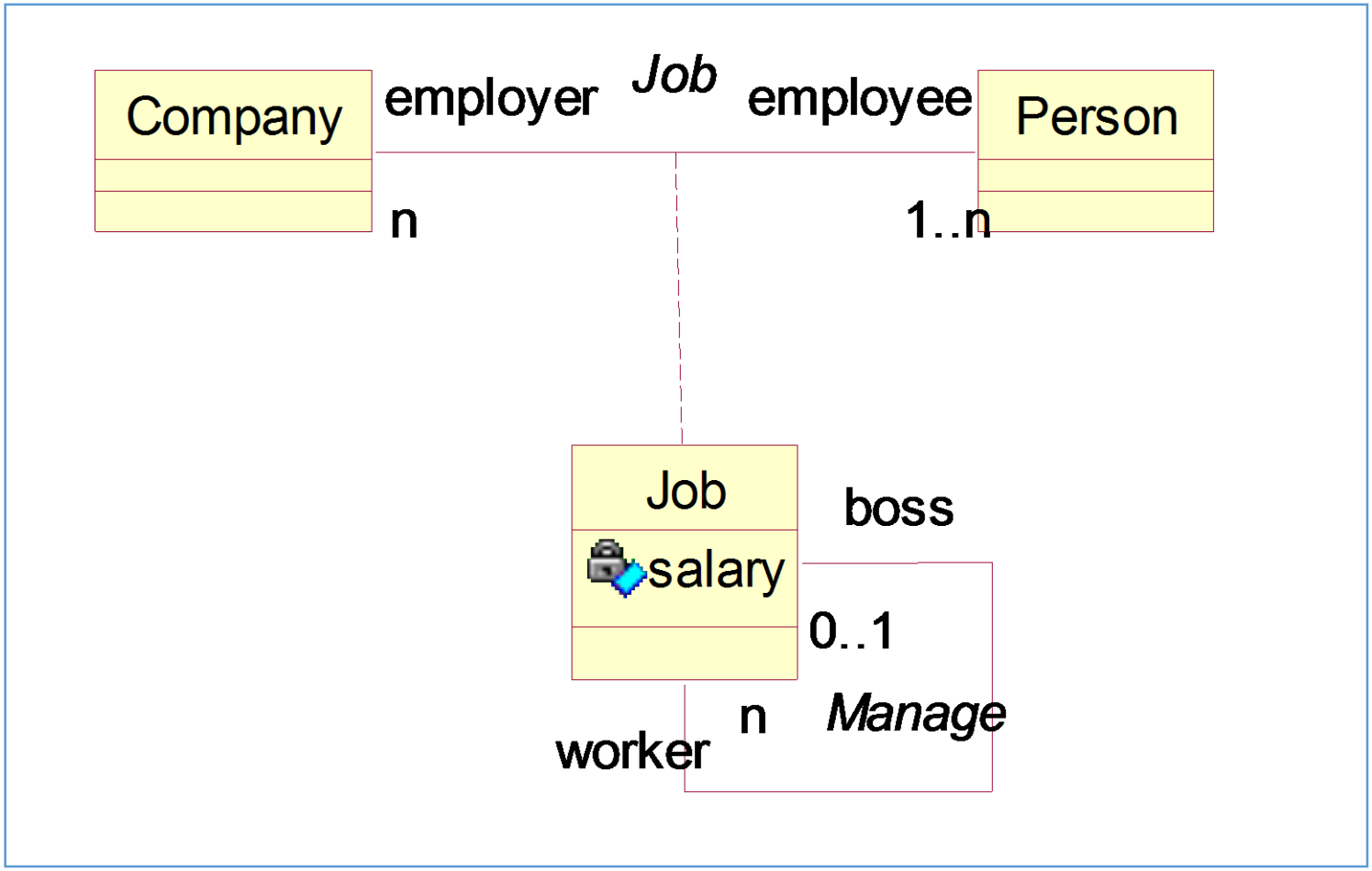
图 4.13　自身关联的关联线从某个类出发又回到其自身。自身关联也可以指明角色名、关联名、关联方向和多重性

自身关联

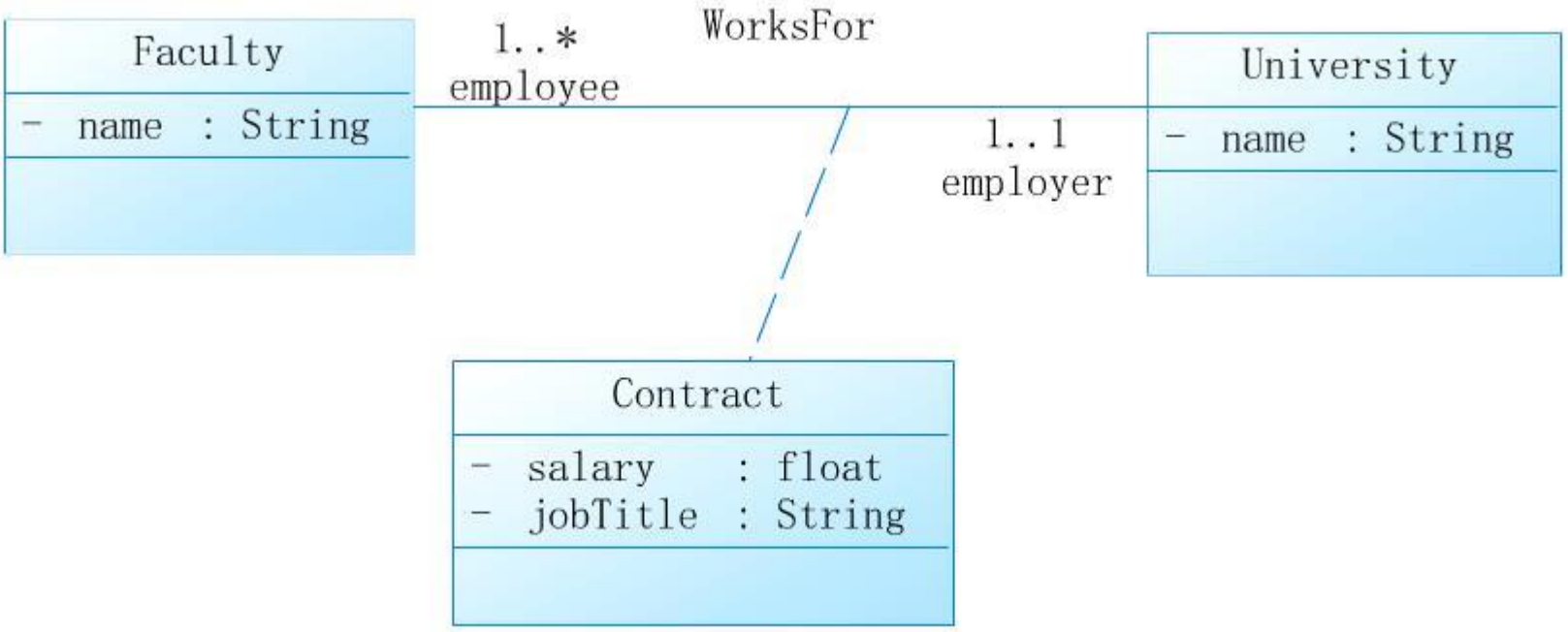



```
public class Person {
    public Person[] parent;
    public Person[] child;
}
```

关联


Company
employer
Job
employee
Person
n
1..n
Job
salary
boss
0..1
n
Manage
worker

关联类



转换为Java代码？

关联类

```
public class Contract {
    private float salary;
    private String jobTitle;
    public Faculty employee;
    public University employer;
}
```

```
public class Faculty {
    private String name;
    public Contract worksFor;
}
```

```
public class University {
    private String name;
    public Contract[] worksFor;
}
```

继承和泛化

◇ 继承（**inheritance**）

如果你知道某物所属的种类，你自然就会知道同类的其他事物也具有该事物的一些特征。在面向对象术语中，这种关系被称为继承。在**UML**中，则被称为泛化（**generalization**）。

泛化关系中，子类继承了父类的行为和含义，子类也可以增加新的行为和含义或覆盖父类的行为和含义。

基类（**base class**）或根类 —— 叶类（**leaf class**）

单继承（**single inheritance**）—— 多继承（**multiple inheritance**）

继承和泛化

Amphibian n. 两栖动物
Mammal n. 哺乳动物
Reptile n. 爬行动物；

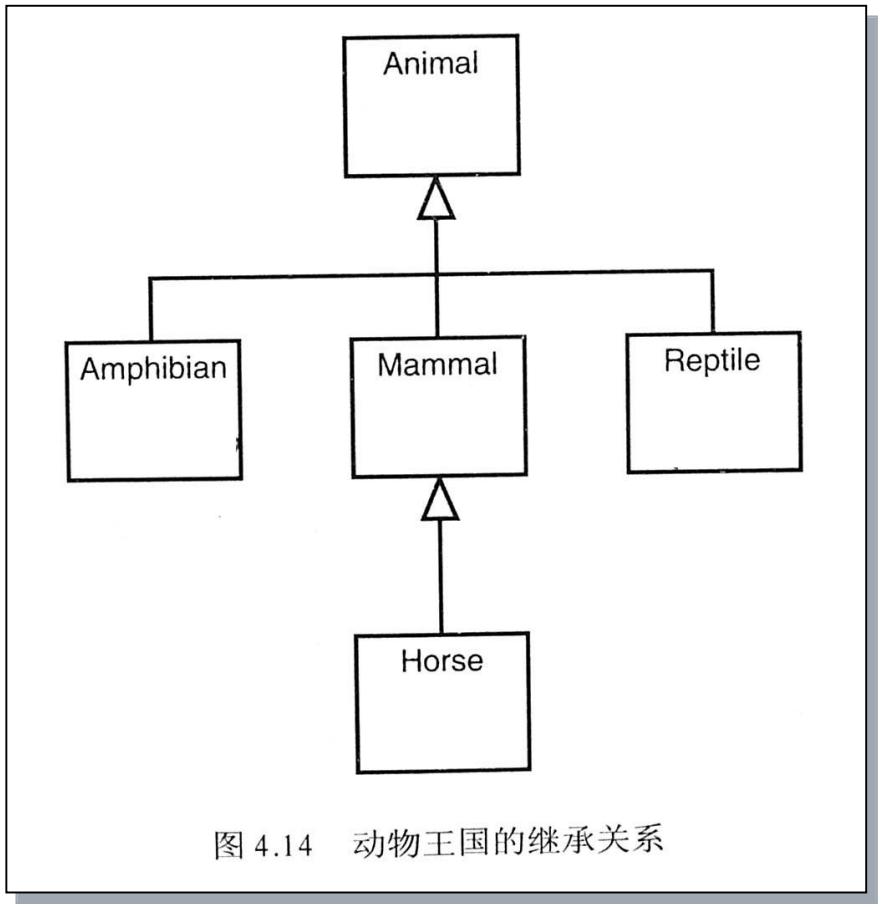


图 4.14　动物王国的继承关系

◇ 符号特征

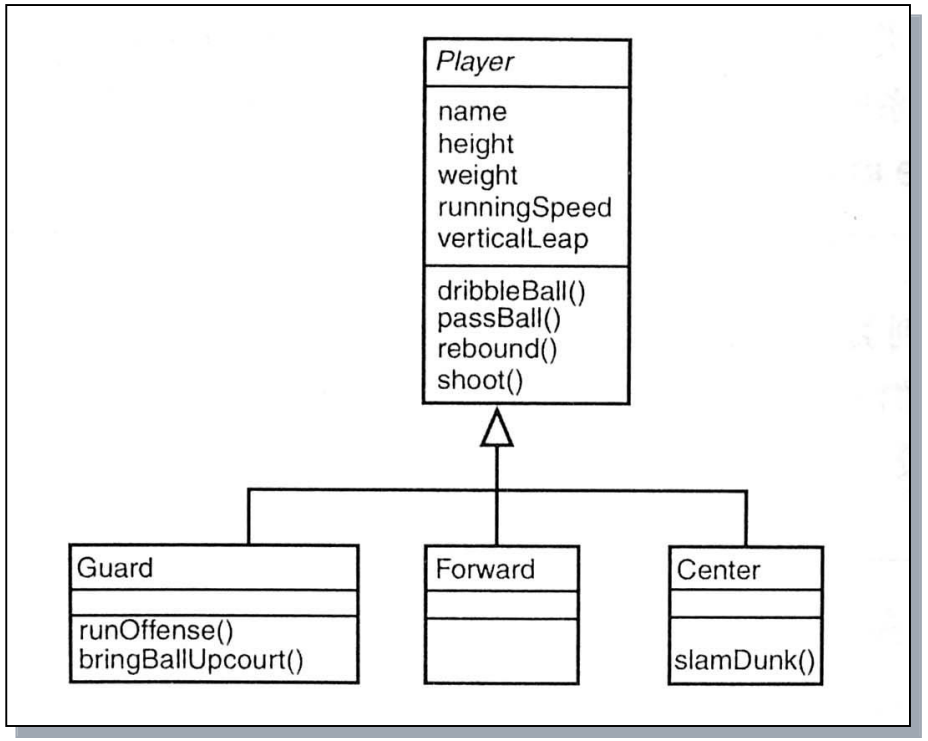
＊指向父类一端带有一个空心三角箭头

继承和泛化

Guard 运球
Forward 传球
Center 抢篮板；

◇ 抽象类（**abstract class**）

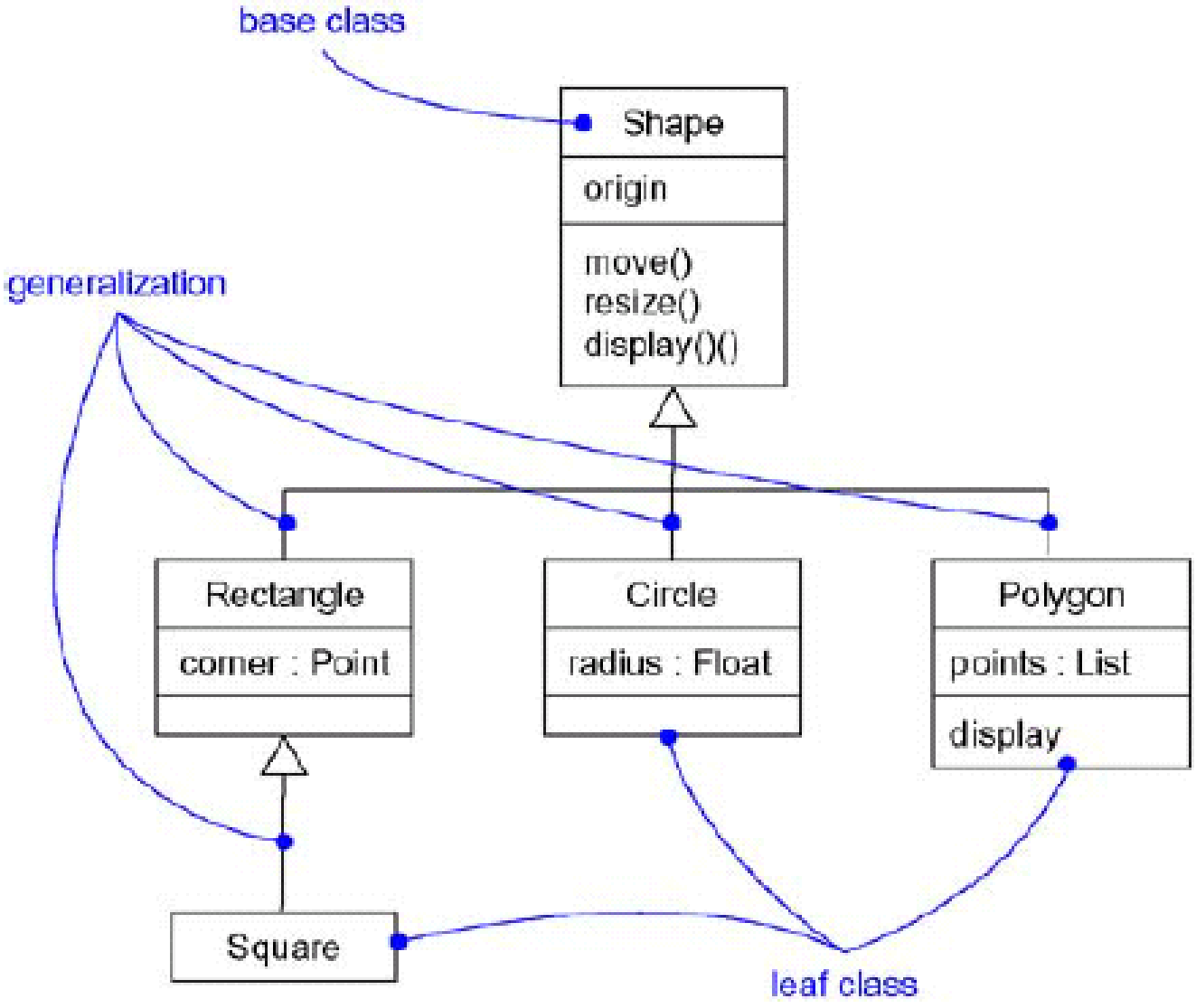
不提供实例对象的类被称为抽象类。



◇ 符号特征

＊类名用斜体书写

泛化关系


base class
Shape
origin
move()
resize()
display()()
generalization
Rectangle
corner : Point
Circle
radius : Float
Polygon
points : List
display
Square
leaf class

依赖

◇ 依赖（**dependency**）

如果一个类使用了另一个类，这种关系称之为依赖。

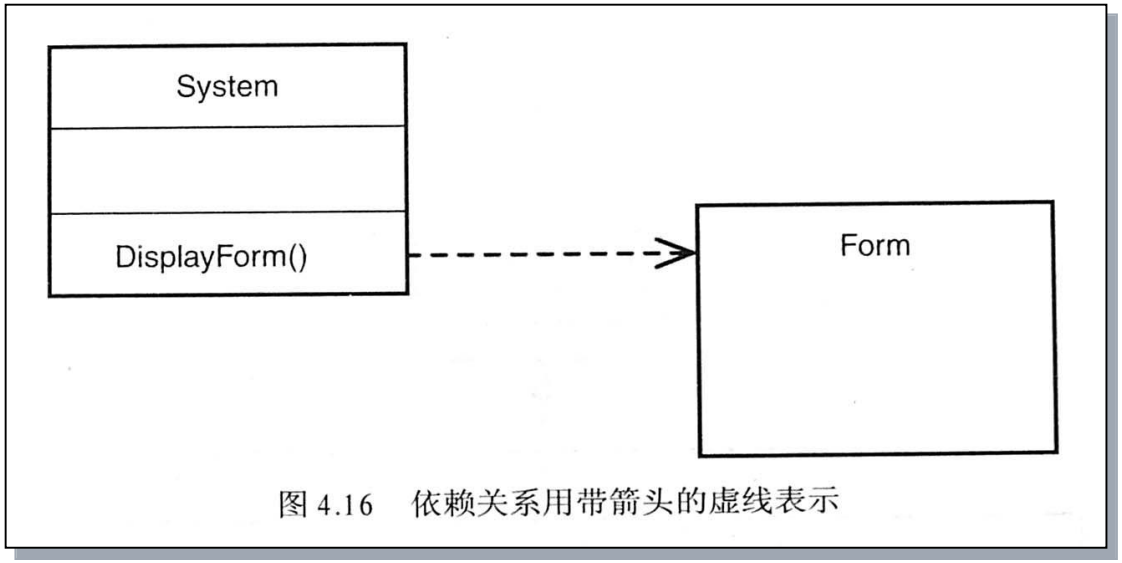
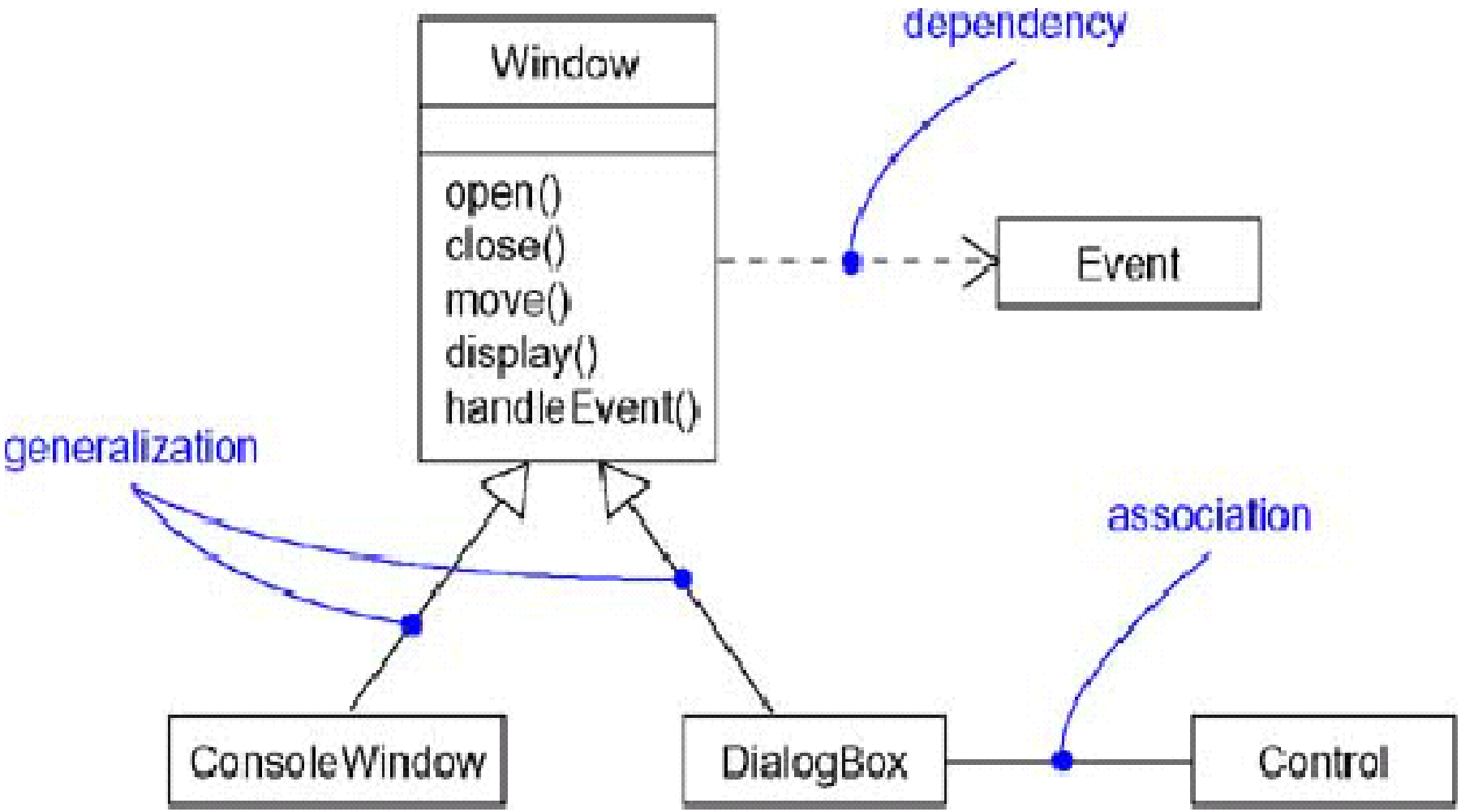


图 4.16　依赖关系用带箭头的虚线表示

◇ 符号特征

* 在有依赖关系的类之间画上一条带箭头的虚线

三种关系和区别


Window
open()
close()
move()
display()
handleEvent()
dependency
Event
generalization
association
ConsoleWindow
DialogBox
Control

类图和对象图

◇ 类图给出的是多个类

◇ 对象图则在某个特定

信息

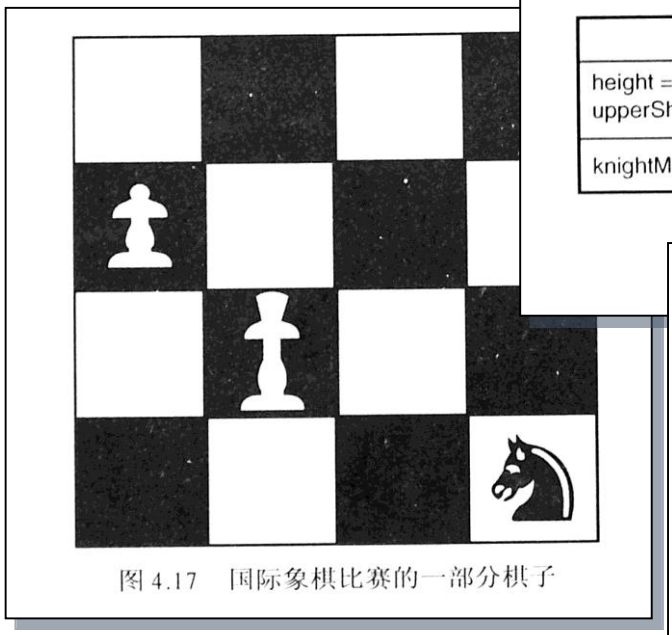
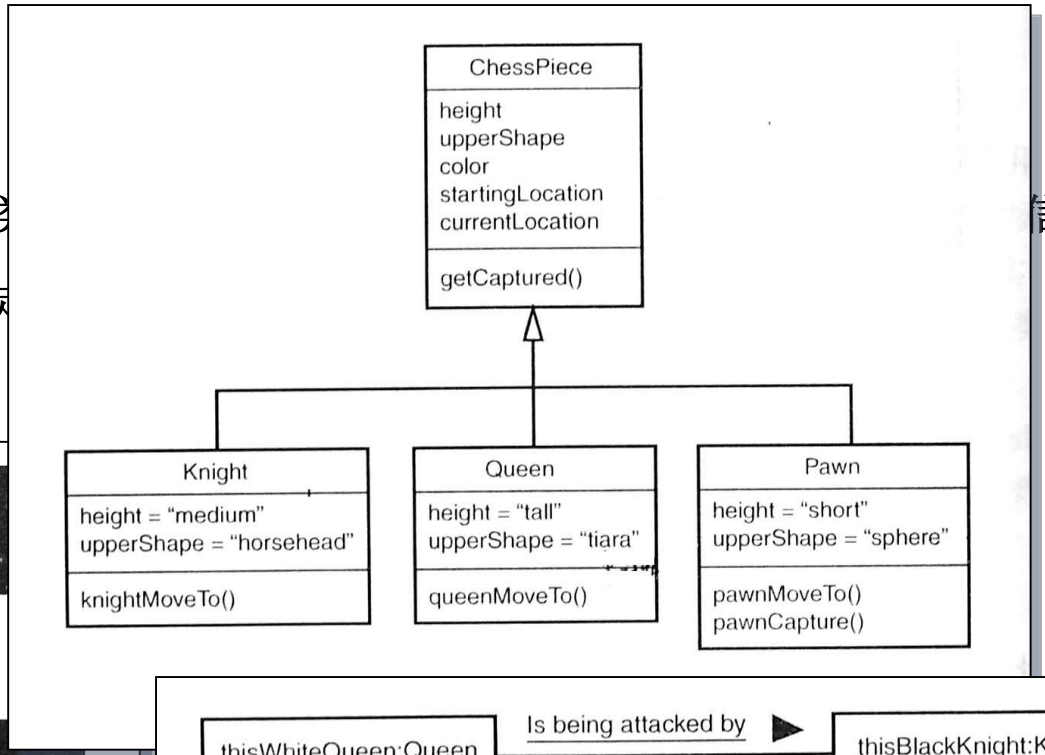


图 4.17　国际象棋比赛的一部分棋子



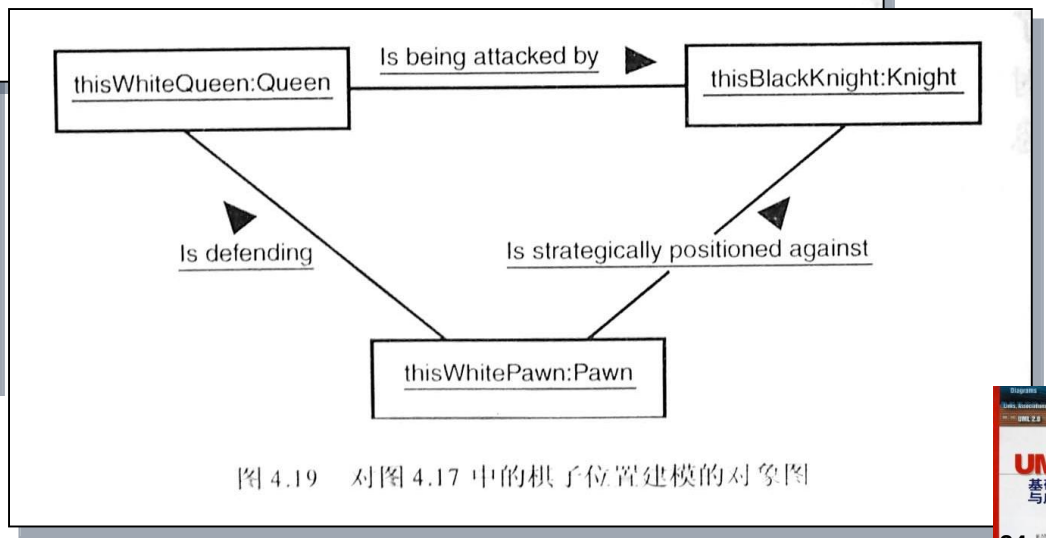
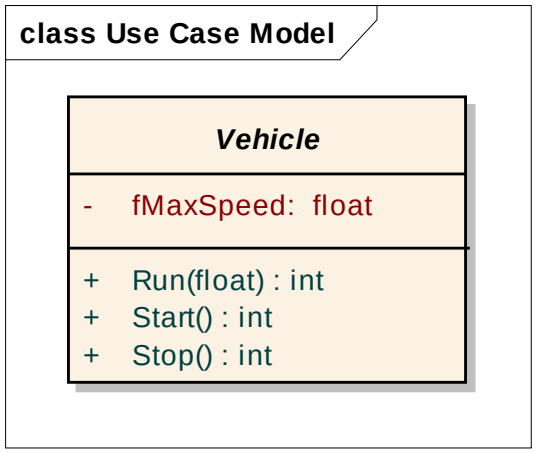


图 4.19　对图 4.17 中的棋子位置建模的对象图

类图与代码的映射

1）类的映射



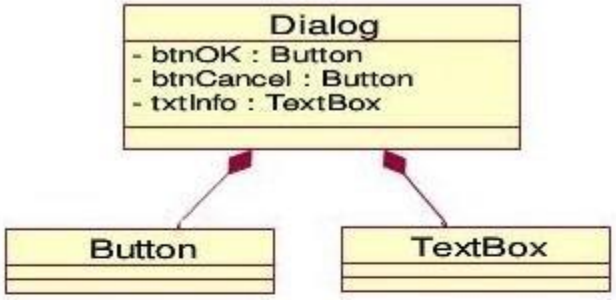
Java代码：

```
public abstract class Vehicle {

        private float fMaxSpeed;
        public Vehicle(){
        }
        public void finalize() throws Throwable {
    //对象释放空间是默认调用此方法
        }
        public abstract int Run(float fSpeed);
        public abstract int Start();
        public abstract int Stop();}
```

C++代码：

```
class Vehicle
{
public:
            Vehicle();
            virtual ~Vehicle();
            virtual int Run(float fSpeed);
            virtual int Start();
            virtual int Stop();

private:
            float fMaxSpeed;
};
```

类图与代码的映射

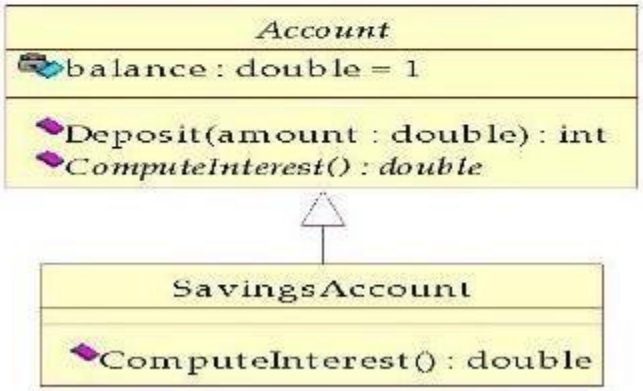


组合关系，代码表现为Dialog的属性有Button和TextBox的对象

C++代码

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

3）泛化关系的映射



C++代码

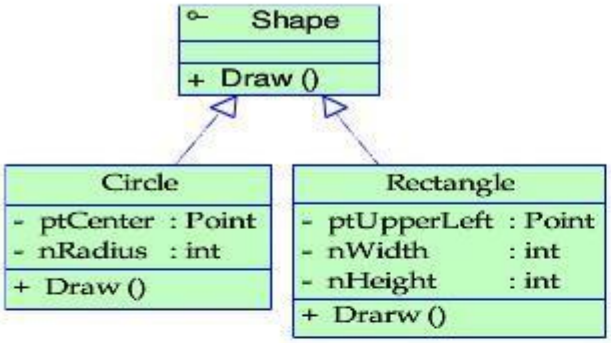
```
class SavingsAccount : public Account
{ };
```

Java代码

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

类图与代码的映射

4. 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

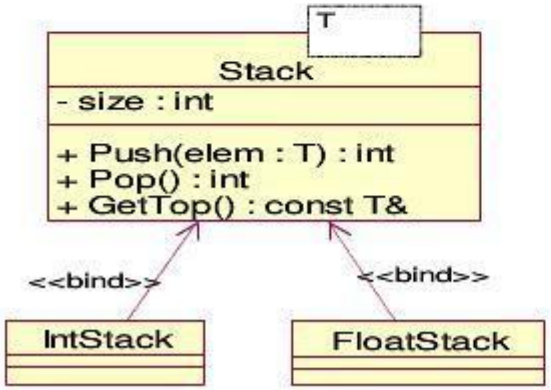
C++代码

```
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;
}
```

5 依赖关系的映射



绑定依赖

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();
};
```

第5章　聚集、组成、接口和实现

- 如何对包含其他类的类建模
- 如何对接口以及与其相关联的类建模
- 可见性的概念

聚集

◇ 聚集（aggregation）

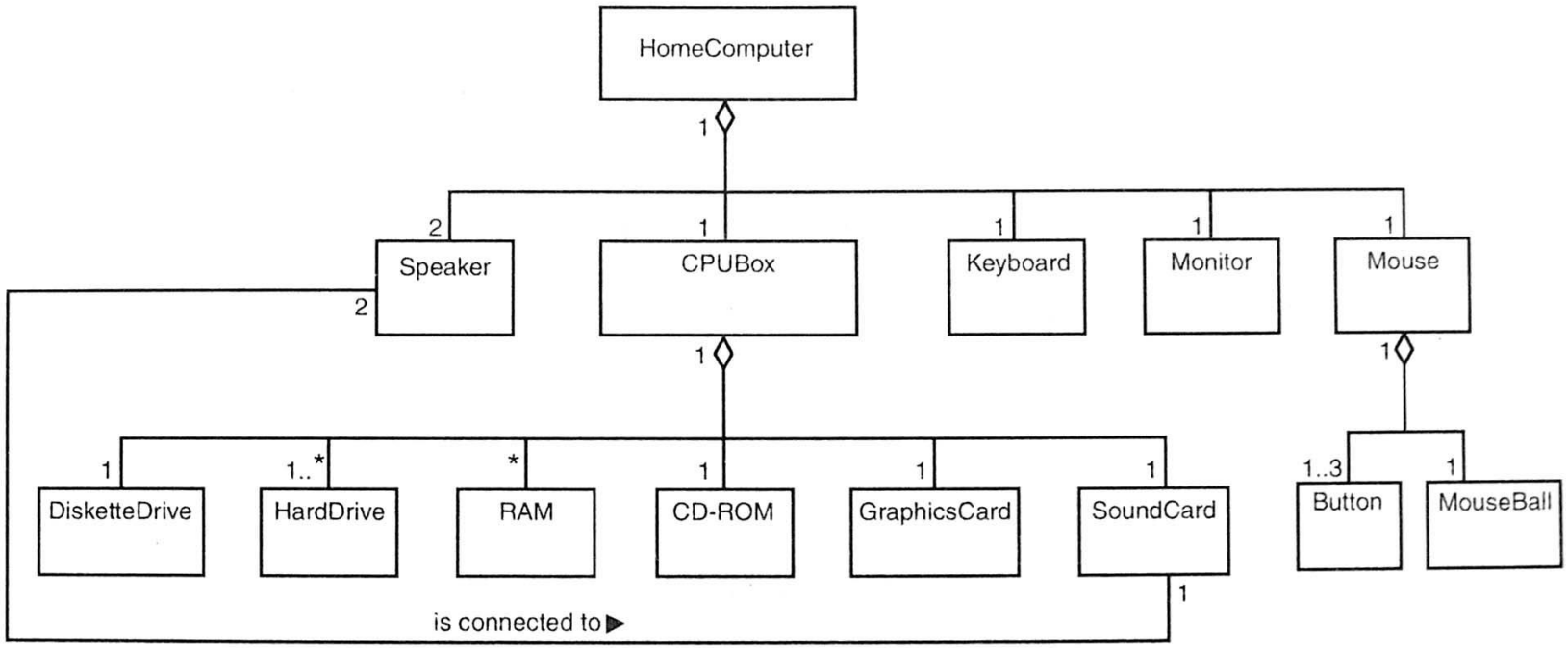


图 5.1　聚集（整体-部分）关联的表示法。关联线上有一个空心菱形箭头，箭头的方向是从部分指向整体

◇ 符号特征

＊关联线上有一个空心菱形箭头，箭头的方向是从部分指向整体。

聚集

◇ 聚集上的约束

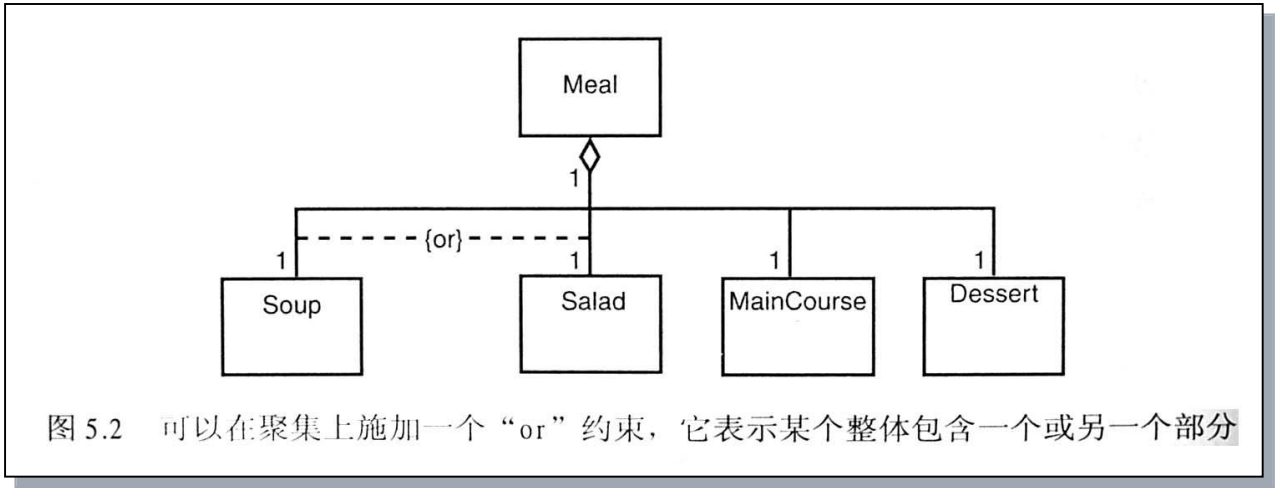


图 5.2　可以在聚集上施加一个“or”约束，它表示某个整体包含一个或另一个部分

组成

◇ 组成

组成是强类型的聚集。聚集中每个部分体只能属于一个整体。

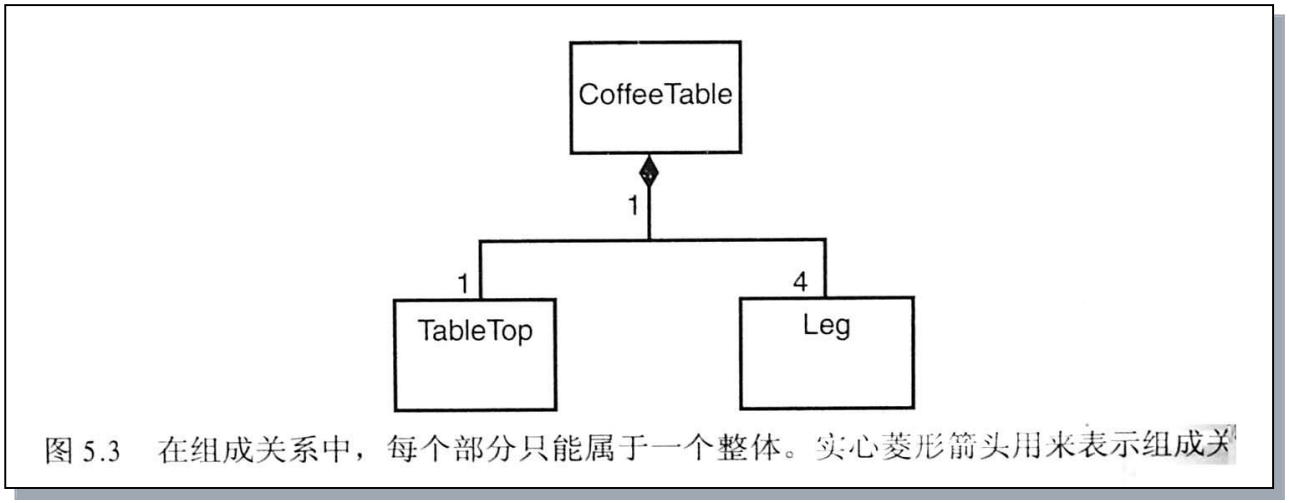


图 5.3　在组成关系中，每个部分只能属于一个整体。实心菱形箭头用来表示组成关

◇ 符号特征

＊关联线上有一个实心菱形箭头，箭头的方向是从部分指向整体。

组成结构图

◇ 组成是展示一个类的构件的一种方式。通过组成结构图可以展示类的内部结构。

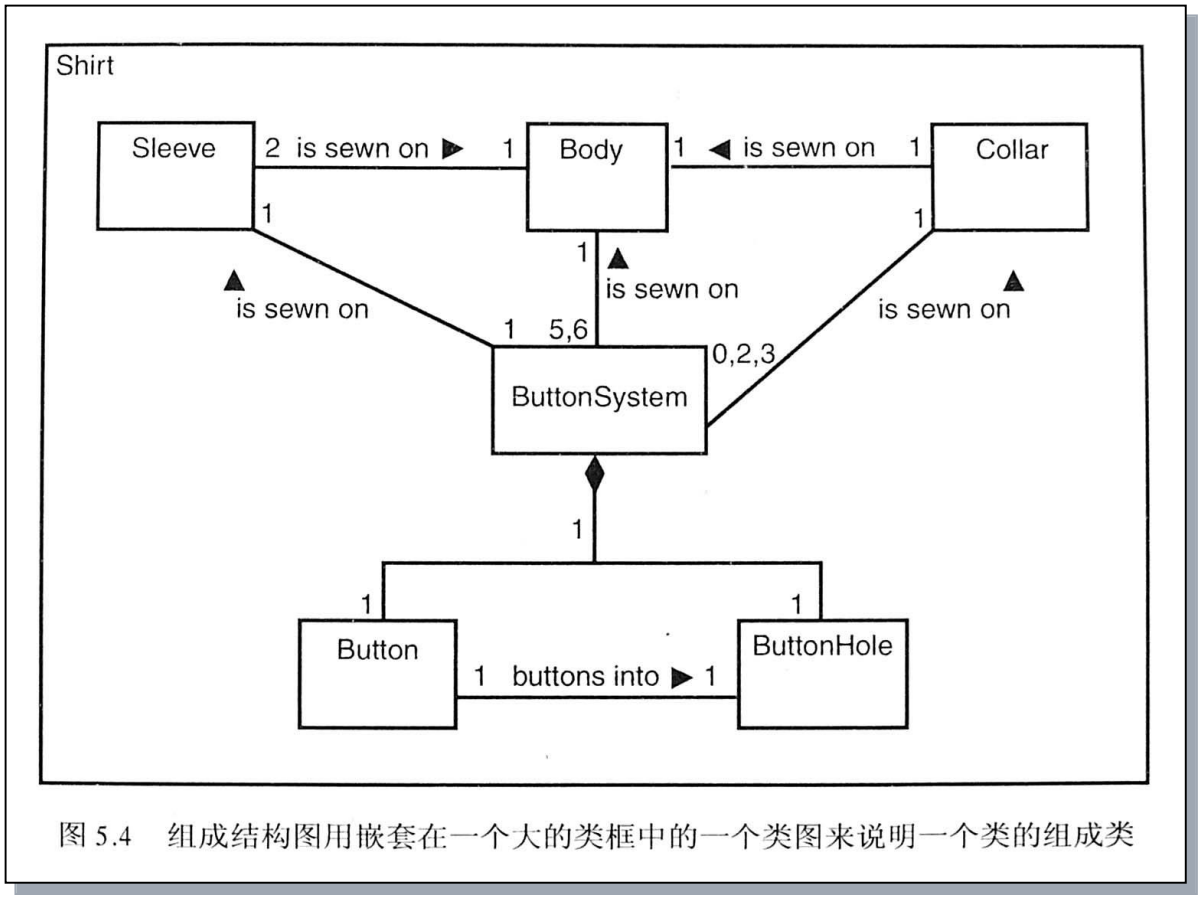
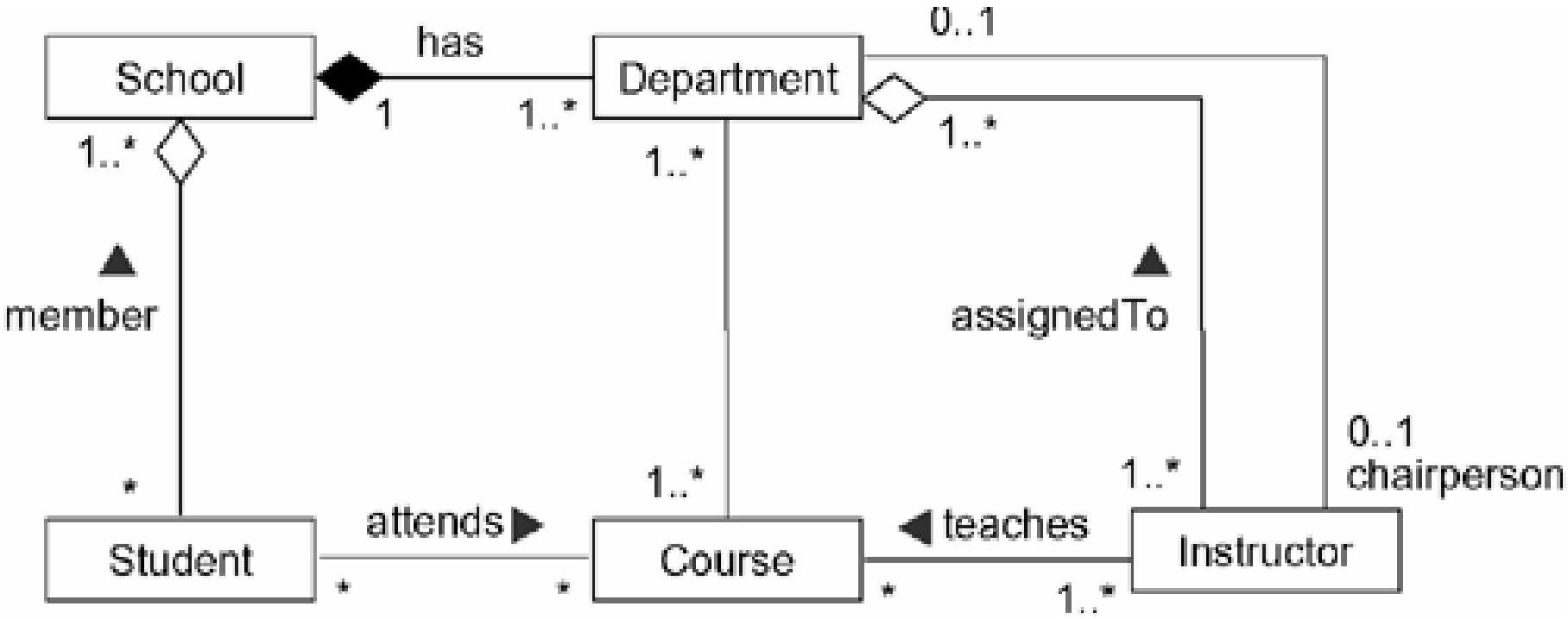


图 5.4　组成结构图用嵌套在一个大的类框中的一个类图来说明一个类的组成类

Example


School
has
Department
0..1
1
1..*
1..*
1..*
1..*
member
assignedTo
*
1..*
1..*
0..1
chairperson
attends
teaches
Student
Course
Instructor
*
*
*
1..*

接口与实现

◇ 接口（**interface**）是描述类的部分行为的一组操作，它也是一个类提供给另一个类的一组操作。

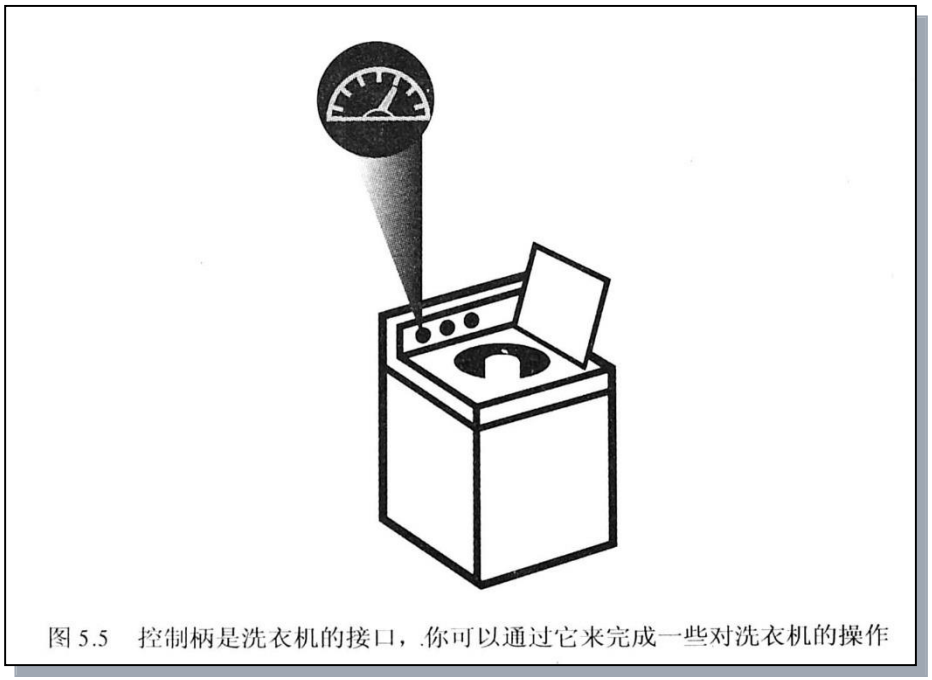


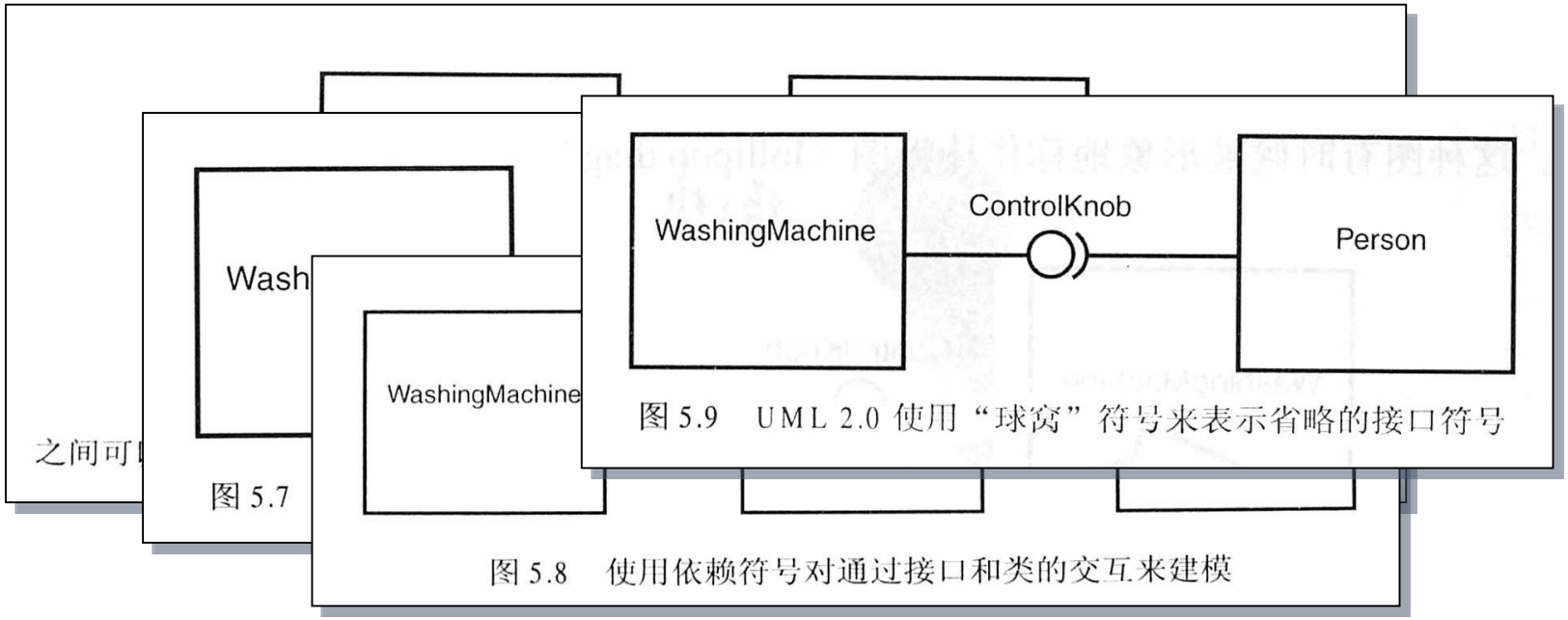
图 5.5　控制柄是洗衣机的接口，你可以通过它来完成一些对洗衣机的操作

◇ 符号特征

* 和类相似，都是用一个矩形图标来代表
* 接口只是一组操作，没有属性

接口与实现

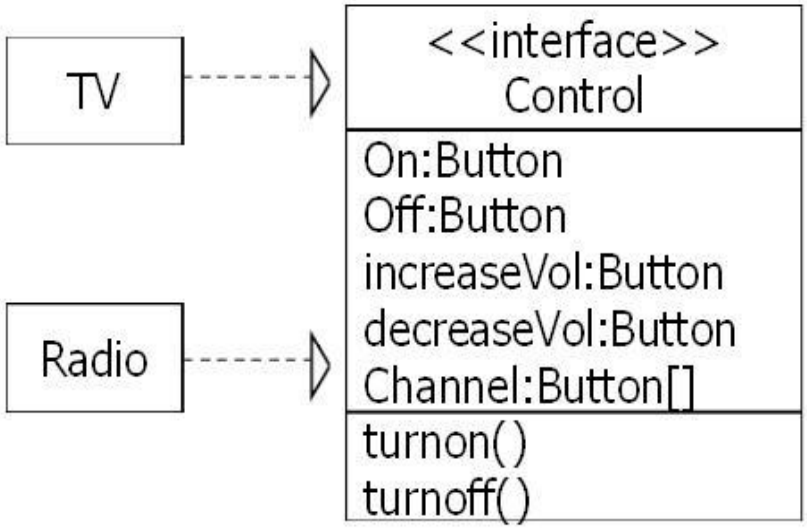
◇ 一个类和它的接口之间的关系叫做实现（**realization**）。

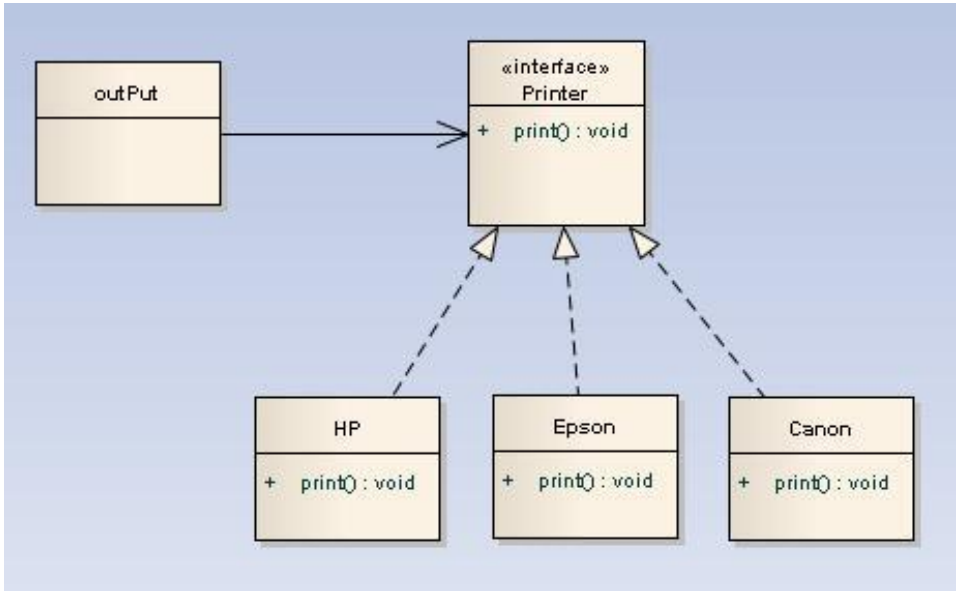


◇ 符号特征

* 和继承符号相似，但它是一个带空心三角形的箭头，箭头的方向指向接口
* 省略表示法是将接口表示为一个小圆圈，并和实现它的类用一条线连起来

接口与实现


TV
Radio
<<interface>>
Control
On:Button
Off:Button
increaseVol:Button
decreaseVol:Button
Channel:Button[]
turnon()
turnoff()


outPut
«interface»
Printer
+ print() : void
HP
+ print() : void
Epson
+ print() : void
Canon
+ print() : void

接口和端口

◇ 端口（**port**）。

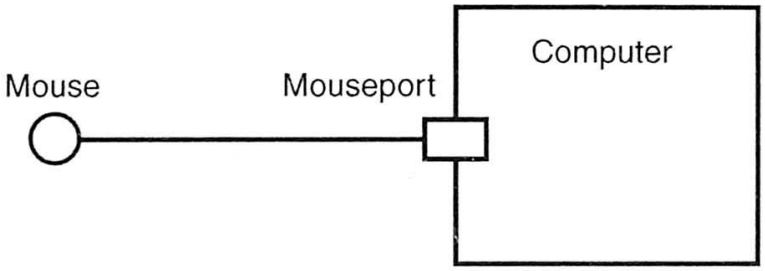


图 5.10　UML 2.0 用来表示端口的符号，它展示了类和环境交互的点

◇ 符号特征

＊ 位于类符号边缘上的一个小方格，这个小方格连接到接口

接口和端口

◇ 可见性（**visibility**）可应用于属性或操作，它说明在给定类的属性和操作（或者接口的操作）的情况下，其他类可以访问到的属性和操作的范围。

◇ 可见性有三个层次（级别）：

公有（**public**）层次上，其他类可以直接访问这个层次中的属性和操作

受保护（**protected**）层次上，只有继承类这些属性和操作的子类可以访问最初类的属性和操作

私有（**private**）层次上，只有最初的类才能访问这些属性和操作

接口和端口

带钥匙的是”Protected”，带锁的是“Private”

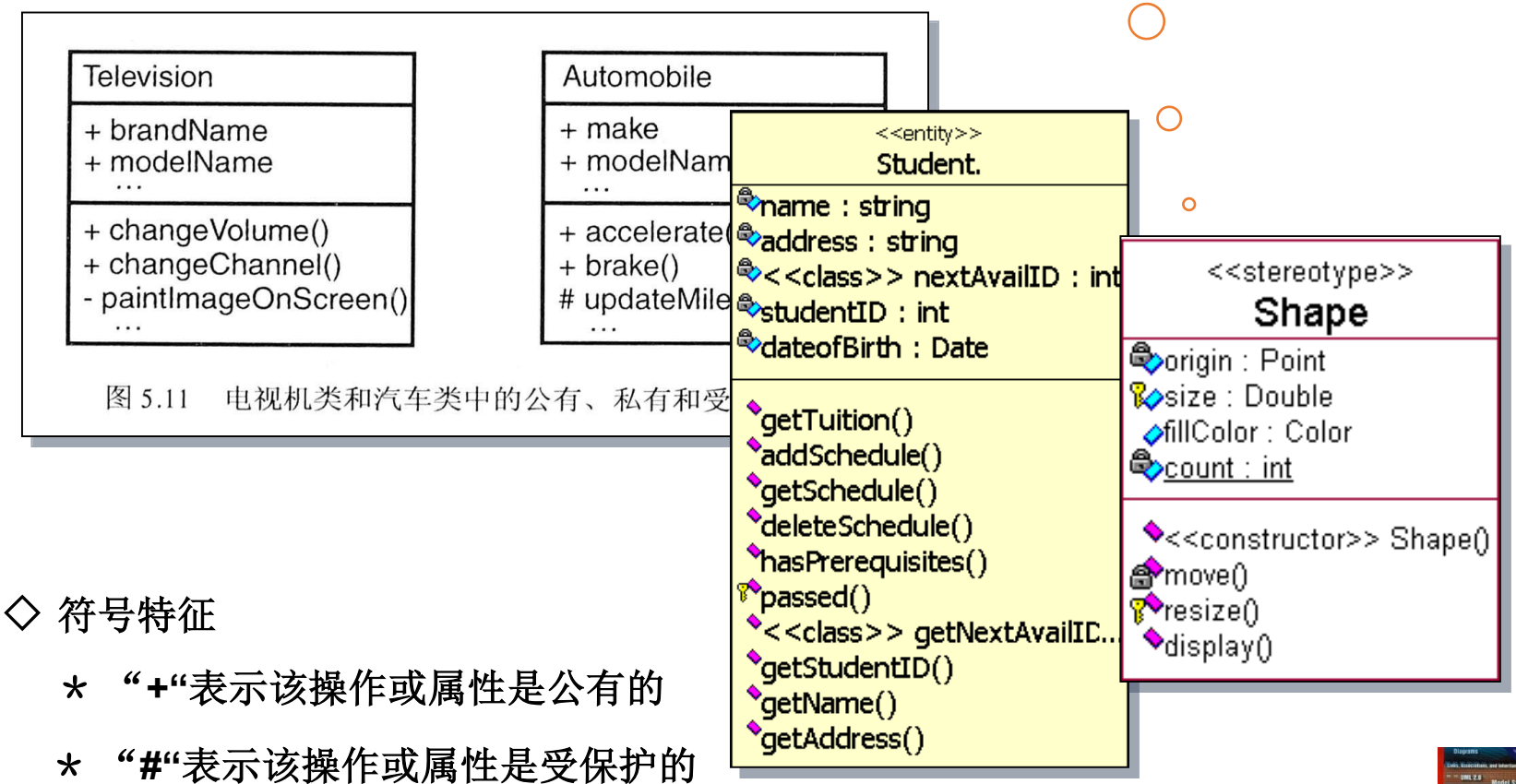


图 5.11　电视机类和汽车类中的公有、私有和受

◇ 符号特征

* “+“表示该操作或属性是公有的
* “#“表示该操作或属性是受保护的
* “-“表示该操作或属性是私有的

1. 在学校中，一个导师可以指导多个研究生，一个研究生可以由多个导师指导，那么导师和研究生之间是（ ）关系。
2. 交通工具与卡车之间是（ ）关系。
3. 公司与部门之间是（ ）关系。
4. 图形与矩形之间是（ ）关系。
5. 参数类及其实例类之间是（ ）关系。

答案 1. 关联 2. 泛化 3. 聚集 4. 泛化 5. 实现

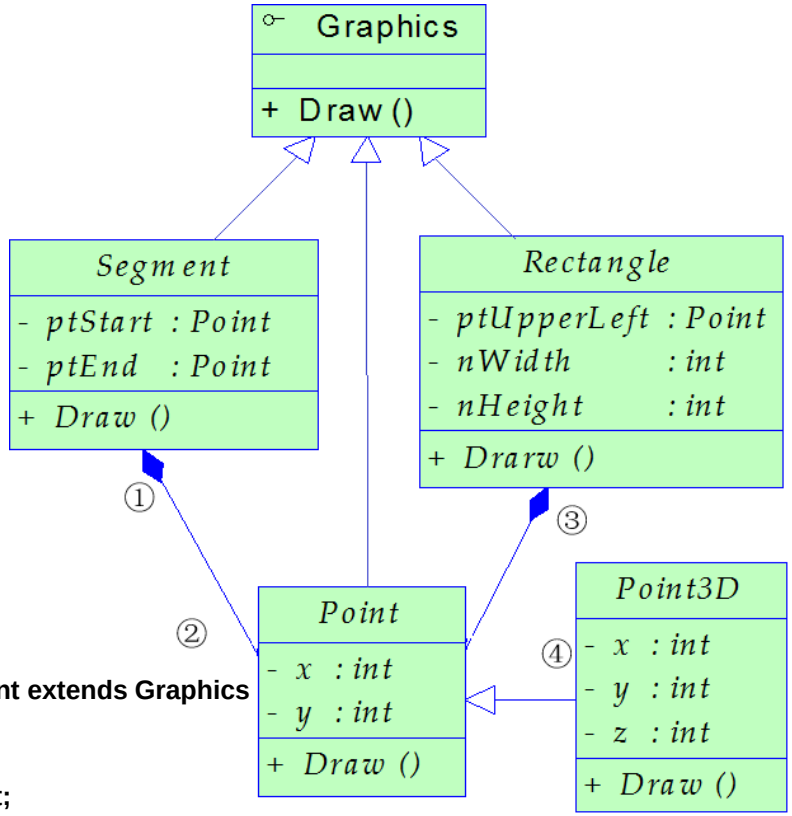
(1)下面哪个关系没有在图中出现

①关联　②泛化　③依赖

(2)下面对图中①②③④四处的多重性的描述哪个不正确

① 0...*　②1　③0...*　④1

(3)下面哪份代码(Java)最接近于图中对Segment的描述




```
public class Segment implements Graphics
{
privatc void Draw();
public Point ptStart;
public Point ptEnd;
}
```

①

```
public class Segment extends Graphics
{
public void Draw();
private Point ptStart;
private Point ptEnd;
}
```

②

```
public class Segment implements Graphics
{
private Point ptStart;
private Point ptEnd;
public void Draw();
}
```

③

```
public class segment implements graphics
{
public void Draw();
private Point ptStart;
private Point ptEnd;
}
```

④