软件建模与设计

授课班级 ：B220417-19，B220400

授课安排：QQ群、超星学习通，课堂

授课时间：周二第8-9节{第1-16周}

金惠颖

第4讲: 静态建模

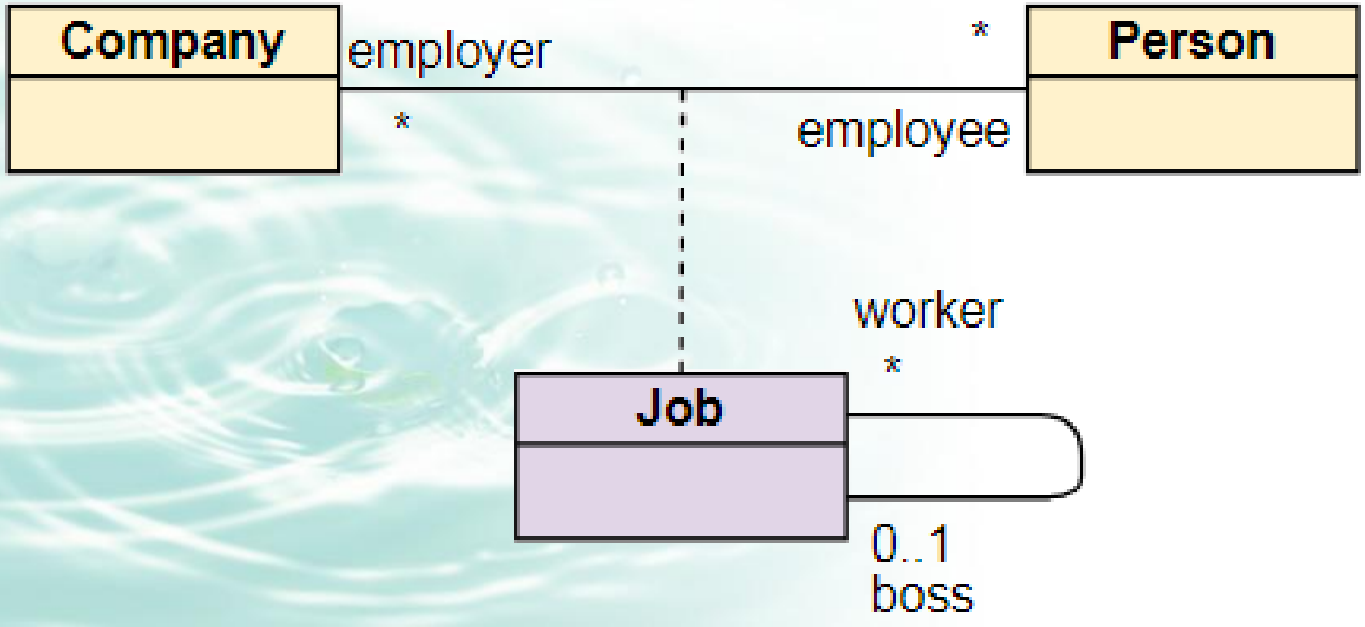
4.1 类和对象

4.2 关系

4.3 类图

4.4 对象图

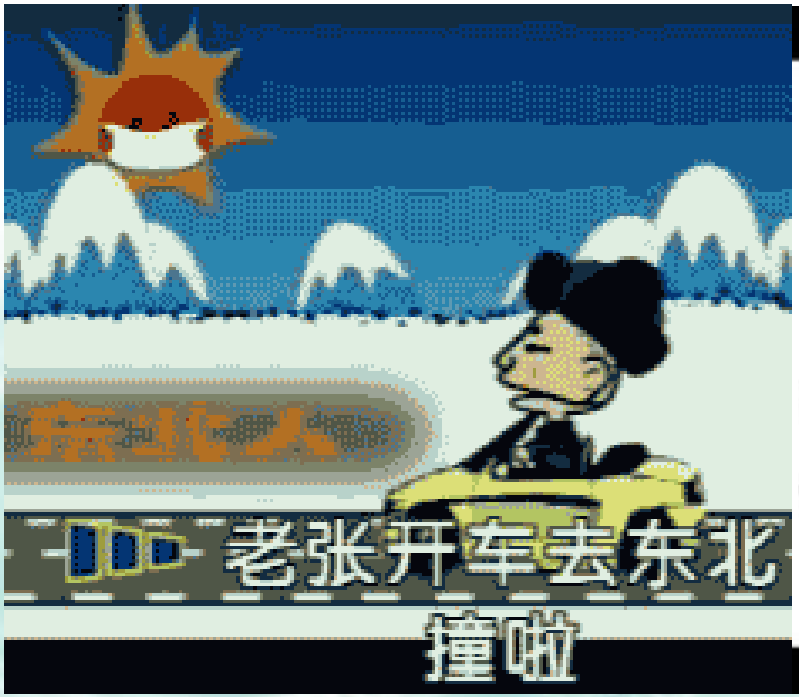
4.5 包图（模型管理图）



第4讲: 静态建模

4.1 类和对象

- 面向对象思想：经典实例 **“老张开车去东北”**



第4讲: 静态建模

4.1 类和对象

- 面向对象思想：经典实例“老张开车去东北”

➢ 程序1（本质上非OO—面向对象）

```
public class Test1 {
        public static void main(String[] args) {
                System.out.println("老张开车去东北");
        }
}
```

第4讲: 静态建模

4.1 类和对象

- 面向对象思想：经典实例“老张开车去东北”

➢ **类：程序2（本质上非OO）**

```
public class Test2 {
        public static void main(String[] args) {
                String driverName = "老张";
                String vehicle = "车";
                String targetPlace = "东北";
                System.out.println(driverName + "开" +
                vehicle + "去" + targetPlace);
        }
}
```

第4讲: 静态建模

4.1 类和对象

- 面向对象思想：经典实例“老张开车去东北”

➢ **程序3（本质上非OO）**

```
public class Test3 {
        public static void main(String[] args) {
                String driverName = "老张";
                String vehicle = "车";
                String targetPlace = "东北";
                go(driverName, vehicle, targetPlace);
        }
        public static void go(String driverName, String
                vehicle, String targetPlace) {
                System.out.println(driverName + "开" +
                vehicle + "去" + targetPlace);
        }
}
```

第4讲: 静态建模

4.1 类和对象

- 面向对象思想：经典实例“老张开车去东北”
- **类：对具有相同数据和操作的相似对象的抽象定义**
  - 名词：老张→Driver；车→Car；东北→Address
    - ✓ 属性（数据）
    - ✓ 操作（方法）
- **封装：实现了对象的主动性和并行性**
  - 对象是数据处理的主体，接收消息，处理自己的数据
  - 不同对象各自独立处理自己的数据，本质上具有并行工作的属性

第4讲: 静态建模

4.1 类和对象

- 面向对象思想：经典实例“老张开车去东北”

➢ 程序4（OO）：Driver类

```
public class Driver {
        private String name;
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public void drive(Car c) {
                c.go(new Address(“东北”));
        }
        public void drive(Car c, Address dest) {
                c.go(dest);
        }
}
```

第4讲: 静态建模

4.1 类和对象

- 面向对象思想：经典实例“老张开车去东北”

➢ **程序4（OO）：Car类**

```
public class Car{
        public void go(Address dest) {
                System.out.println("一路哼着歌，冒着烟，
                        去了" + dest.getName());
        }
}
```

第4讲: 静态建模

4.1 类和对象

- 面向对象思想：经典实例“老张开车去东北”

程序4（OO）：Address类

```
public class Address {
        private String name;
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public Address(String name) {
                super();
                this.name = name;
         }
}
```

第4讲: 静态建模

4.1 类和对象

- 面向对象思想：经典实例“老张开车去东北”

➢ **程序4（OO）：主程序**

```
public class Test {
        public static void main(String[] args) {
                Driver d = new Driver();
                Address ad = new Address("北京");
                d.setName("老张");
                System.out.println(d.getName());
                d.drive(new Car());
                d.drive(new Car(), ad);
        }
}
```

结果显示

- 老张
- 一路哼着歌，冒着烟，去了东北
- 一路哼着歌，冒着烟，去了北京

第4讲: 静态建模

4.1 类和对象

- 面向对象思想：经典实例 **“老张开车去东北”**

```
public class Driver {
    private String name;
    public Driver(String name) {
        super();
        this.name = name;}
    public String getName() {
        return name;    }
    public void setName(String name) {
        this.name = name; }
    public void drive(Car c) {
        c.go(new Address(“东北”));   }
    public void drive(Car c, Address dest)
    {
        c.go(dest);
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        Driver d = new Driver("老张");
        Car c = new Car();
        Address ad = new Address("北京");
        System.out.println(d.getName());
        d.drive(c, ad);
    }
}
```

协议的一部分

消息

服务

第4讲: 静态建模

4.1 类和对象

- 面向对象思想：经典实例“老张开车去东北”
- **程序5（OO）：扩展1**
- **问题：Car → Train，Plane**
- **方案：增加抽象类Vehicle**

```
public abstract class Vehicle {
        public abstract void go(Address dest);
}
```

第4讲: 静态建模

4.1 类和对象

- 面向对象思想：经典实例“老张开车去东北”

➢ **程序5（OO）：扩展1—修改Driver类**

```
public class Driver {
        private String name;
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public void drive(Vehicle v) {
                v.go(new Address(“东北”));
        }
        public void drive(Vehicle v, Address dest) {
                v.go(dest);
        }
}
```

第4讲: 静态建模

4.1 类和对象

- 面向对象思想：经典实例 **“老张开车去东北”**
- **程序5（OO）：扩展2**
- **需求变化：**
  - ✓ 老张乘火车、乘飞机、骑着扫把去北京?
- **修改程序**
  - ✓ 增加Vehicle类的子类Train、Plane、Broom，并在子类中**重写**go方法 （重载）
  - ✓ 修改主程序Test

第4讲: 静态建模

4.1 类和对象

- 面向对象思想：经典实例“**老张开车去东北**”
  - **类**
  - **对象**
  - **封装**
  - **消息**
  - **接口（协议）**
  - **服务**
  - **继承**
  - **重载**
  - **多态**

第4讲: 静态建模

4.1 类和对象

- 类（class）
  - **对具有相同数据和相同操作的一组相似对象的抽象定义**
  - **理解**
    - 对具有相同属性（不同值）和行为的一组对象的描述
    - 支持继承的抽象数据类型，是**对象实例化的模版**
    - 一个对象就是类的一个实例
    - 类定义了对象的特征，分为**属性**和**操作**
      - ✓ **属性**：定义类实例存储的数据，是对客观实体所具有共同性质的抽象，类实例化的每个对象都有自己特有的属性值
      - ✓ **操作**：也称方法，定义类实例的动态行为（算法）

第4讲: 静态建模

4.1 类和对象

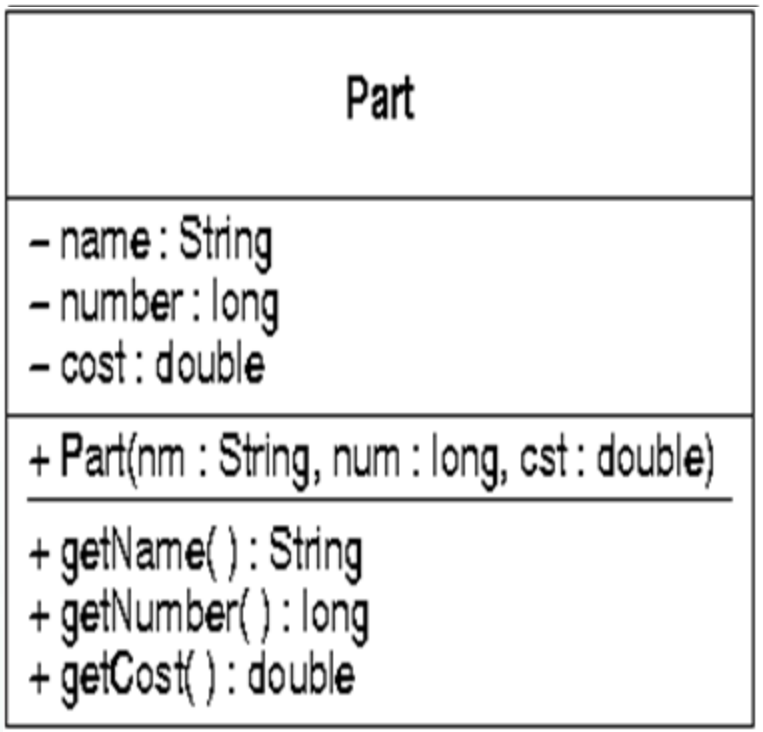
- **库存管理系统**
  - 某制造业的装配车间，通过订购零件来组装复杂产品，现需设计一个“库存管理系统”辅助车间库存管理
  - 功能
    - 维护零件的库存信息：零件出入库管理
    - 记录产品（组件）的组装结构
    - 以产品（组件）为单位：
      - ✓ 计算产品（组件）的零件总价格
      - ✓ 打印产品（组件）的零件清单
  - 零件特征：
    - 零件编号number : long;
    - 零件名称name : string;
    - 零件单价cost : double;

第4讲: 静态建模

4.1 类和对象

- 初步的零件候选类

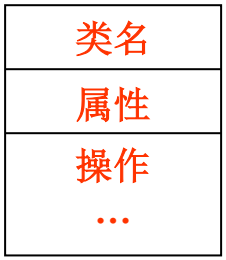
```
public class Part {
    private String name ;
    private long number ;
    private double cost ;

    public Part(String nm,
      long num, double cost)
      {
      name = nm ;
      // etc
    }
    public String getName( )
      { return name; }
     // etc
}
```



第4讲: 静态建模

4.1 类和对象

- **类（Class）的UML图标**
  - **用实线矩形框表示，矩形框中含有若干分隔框，分别包含类的名字、属性、操作、约束以及其他成分等。**
    - ✓ 类名可以是简单名，也可以是路径名。
    - ✓ 属性框中包含类的属性。属性是类的命名的性质，它描述类性质的实例所能具有的值。
    - ✓ 操作框中包含类的操作。操作实现类的服务功能，它可以被本类的对象请求执行，从而发生某种行为。



第4讲: 静态建模

4.1 类和对象

- **类（Class）的UML图标：属性**
  - 属性（Attribute）是类的命名的性质，属性在类图标的属性分隔框中用文字串说明。
  - 属性有在类中唯一的属性名或标识符。
  - 冒号“:”后跟属性值的数据类型。
  - 属性名后跟的方括号中的内容是可选项目。
    - ✓ **多重性**（Multiplicity）用多值表达式表示，其值是该类的每个实例的属性值的个数。
    - ✓ 多值表达式的格式为：integer，integer，… 或 低界..高界

属性
可视性 属性名 [多重性]：类型 = 初始值

第4讲: 静态建模

4.1 类和对象

- **类（Class）的UML图标：属性的可视性**
  - **可视性（Visibility）用以下可视性标记表示：**
    - ✓ + （公共），# （保护），– （私用），~（包）
  - **可视性也可以用以下关键字表示：**
    - ✓ public（公共）、protected（保护）、private（私用）、package（包）
  - **若可视性标记为“+”或“public”，则为公共属性，可以被外部对象访问。**
  - **若可视性标记为“#”或“protected”，则为保护属性，可以被本类或子类的对象访问。**
  - **若可视性标记为“–”或“private”，则为私用属性，不可以被外部对象访问，只能为本类的对象使用。**
  - **若可视性标记为“~”或“package”，则为包属性，只对同一包中其他类的对象可见。**
  - **可视性可以缺省，表示该属性不可视。**

第4讲: 静态建模

4.1 类和对象

- **类（Class）的UML图标：属性的属主范围**
  - **若属性的属主范围是实例，则该类的每一个实例对象都有一个自己的该属性的值**
  - **若属性的属主范围是分类符，则对于该类本属性只有一个值，该类的每一个实例对象都持有此唯一的属性值。**
    - ✓ **属性标注下划线**

第4讲: 静态建模

4.1 类和对象

- **类的UML图标：操作（类的行为特征或动态特征）**
  - **一个类可以有多个操作，也可以没有一个操作；没有一个操作的类常用于表达接口或数据表。**
  - **操作用文字串说明，在类中有唯一的操作名或标识符。**
  - ✓ **参数列表是可选项目，即一个操作可以有参数，也可以没有参数，由逗号分隔的操作的形式参数组成，其格式为：**

    参数名： 类型 = 缺省值，…
  - ✓ **返回列表由逗号分隔的操作的返回值类型表达式组成，其格式为：返回类型 或 返回名字 = 类型，…**

操作

可视性 操作名 （参数列表）：返回列表 （性质）

第4讲: 静态建模

4.1 类和对象

- **类的UML图标：操作（类的行为特征或动态特征）**
  - **操作的可视性的表示方法和含义与属性中相同。**
  - **操作也有属主范围的区分，它的含义和表示与属性的属主范围相同，分类符属主范围的操作标注下划线。**
  - ✓ **对象的构造操作（构造函数）必须带有下划线，表示它的属主范围是分类符。构造操作也可以用构造型<<constructor>>标示。**
  - ✓ **操作定义的最后花括号{}中的性质，是一个文字串，说明该操作的一些有关信息。性质是一个可选项。**

📖 ***注意区别术语“操作”和“方法（Method）”：操作是被对象调用的一个过程，而方法是过程体，这在有多态性的情况下二者是有所区别的。***

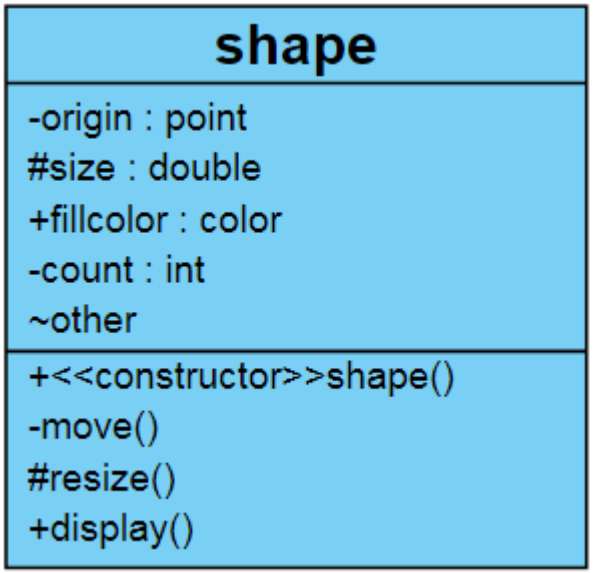
第4讲: 静态建模

4.1 类和对象

- **类的UML图标：总结**

➢ **类（Class）**

- 具有相似结构、行为和关系的一组对象的描述
- 符号
- 组成
  - ✓ 名称：类的唯一标识
  - ✓ 属性：描述类的静态特征
  - ✓ 操作：说明类所提供的服务
  - ✓ 职责：定义类的责任和义务
  - ✓ 约束：指明类满足的规则



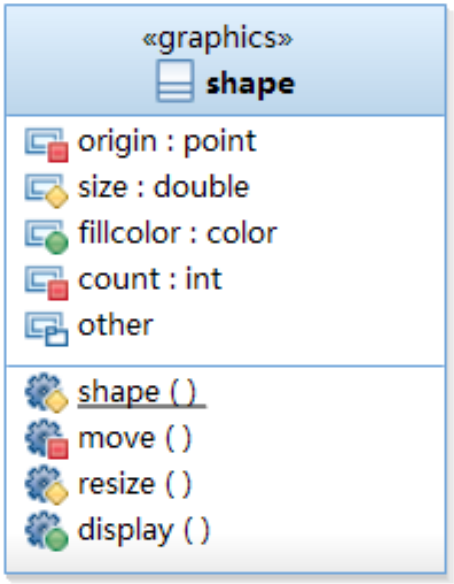
第4讲: 静态建模

4.1 类和对象

- **类的UML图标：总结**

➢ **属性**

- [可视性]属性名[:类型]['['多重性[次序]']'][=初始值][{特性}]
  - ✓ 可视性：可访问性
  - ✓ 多重性：属性值个数格式
  - ✓ 次序：属性值顺序
  - ✓ 特性：属性约束



➢ **操作**

- [可视性]操作名[(参数列表)][:返回类型][{特性}]

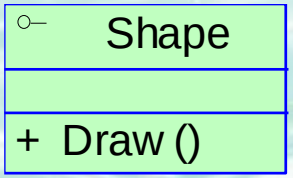
第4讲: 静态建模

4.1 类和对象

- **类的UML图标：接口类、抽象类、模板类**
  - **接口：一组操作的集合，只有操作的声明而没有实现**
    - ✓ 标准图形、变体图形，没有属性
    - ✓ 不要求实现类和接口类概念本质上是一致的
  - **抽象类：不能被实例化的类，一般至少包含一个抽象操作**
    - ✓ 类名、抽象操作名均为斜体
  - **模版类：一种参数化的类，在编译时把模版参数绑定到不同的数据类型，从而产生不同的类；模版化的类具有相同行为，但数据类型不同。**

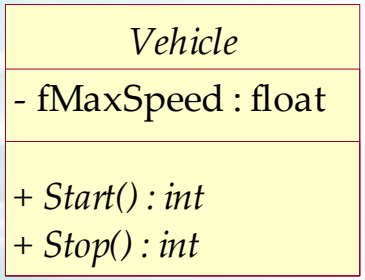
Shape

(标准图形)

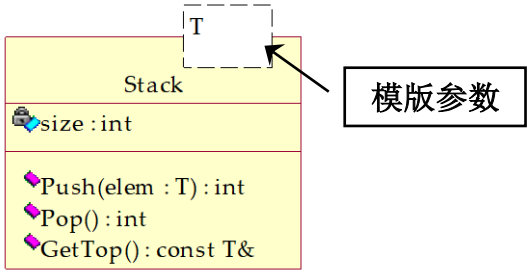


(变体图形)

接口



抽象类

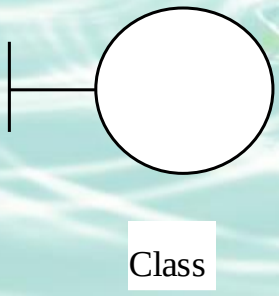


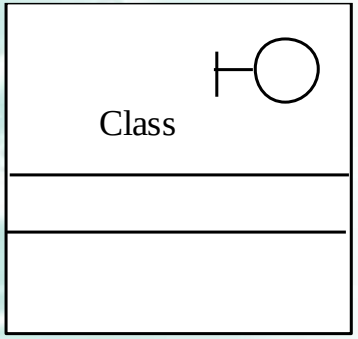
模版类

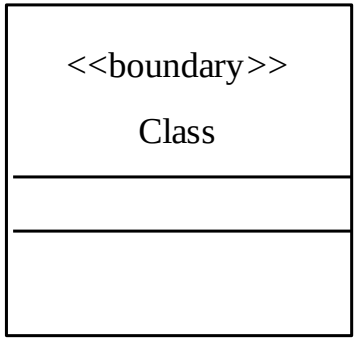
第4讲: 静态建模

4.1 类和对象

- **类的UML图标：分析类**
  - **边界类（boundary）**
    - ✓ 边界类处理系统环境与系统内部之间的通信，为用户或另一个系统（即参与者）提供了接口，例如窗体、对话框、报表、与外部设备或系统交互的类等
    - ✓ 边界类可以通过用例确定，因为参与者必须通过边界类参与用例
    - ✓ 边界类的UML符号表示：



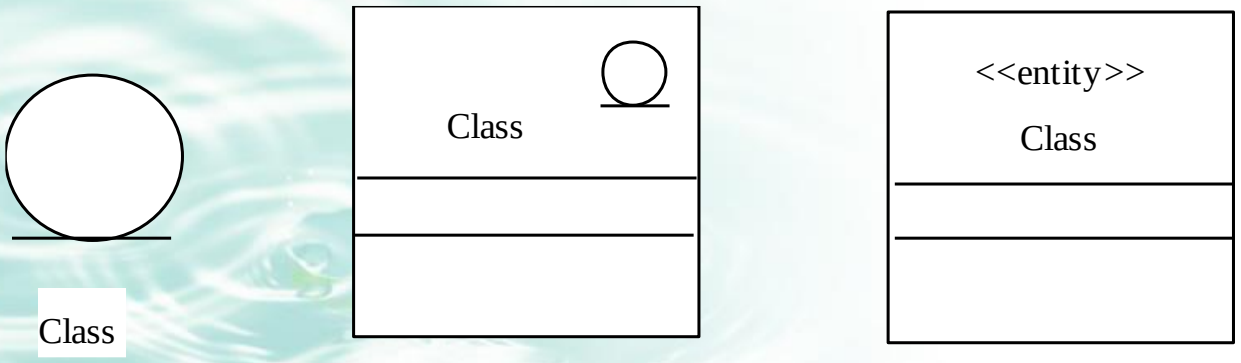




第4讲: 静态建模

4.1 类和对象

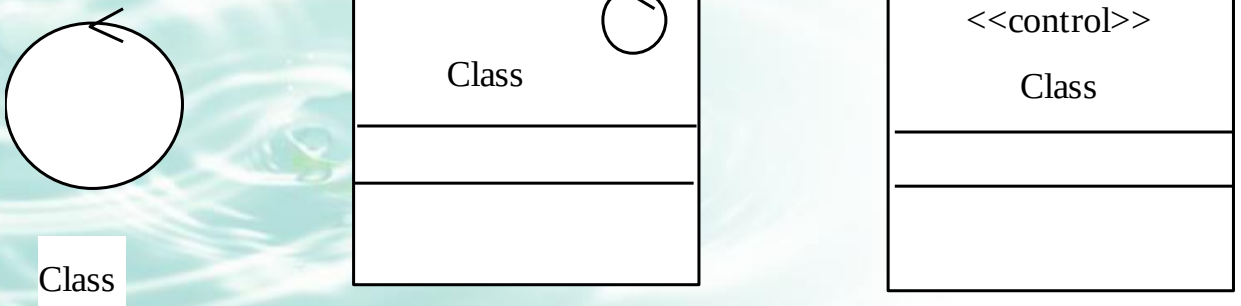
- **类的UML图标：分析类**
  - **实体类（entity）**
    - ✓ 实体类是模拟必须被存储的信息和其关联行为的类。
    - ✓ 保存永久信息，最终可能映射数据库中的表和字段
    - ✓ 实体类的UML符号表示：



第4讲: 静态建模

4.1 类和对象

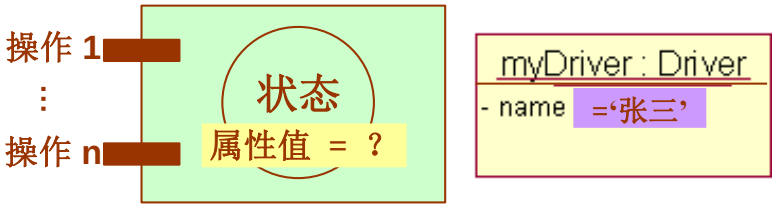
- **类的UML图标：分析类**
  - **控制类 （control）**
    - ✓ 控制类是用来为特定于一个或多个用例的控制行为建模的类。
    - ✓ 一个用例有一个控制类，协调其他类工作和控制总体逻辑流程，例如：命令处理程序
    - ✓ 控制类的UML符号表示：



第4讲: 静态建模

4.1 类和对象

- **对象**
  - **由描述属性的数据和施加在这些数据的所有操作封装在一起的统一体**



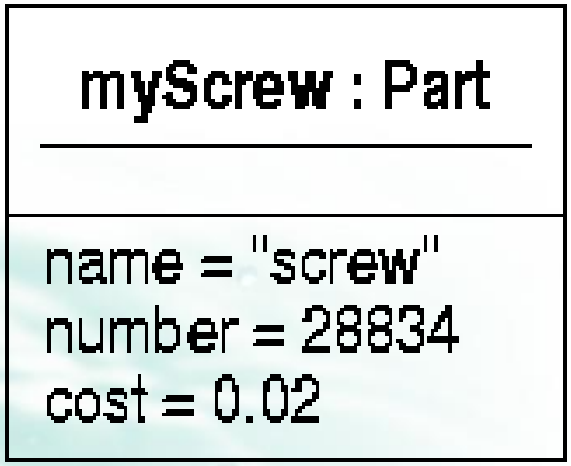
  - **理解**
    - ✓ 对象是对问题域中某个客观实体的抽象，是对属性值和操作的封装
    - ✓ 形式化的定义：对象 ::= <ID,MS,DS,MI>
      - ID → 对象的表示或名字
      - MS → 对象中操作的集合
      - DS → 对象的数据结构
      - MI → 对象受理的消息集合，即对外的接口

第4讲: 静态建模

4.1 类和对象

- **库存管理系统：创建一个对象**

```
Part myScrew = new Part(“screw”, 28834, 0.02);
```

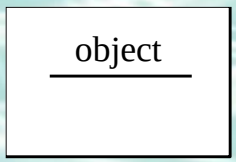


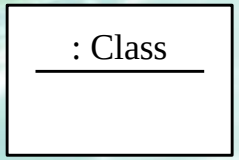
第4讲: 静态建模

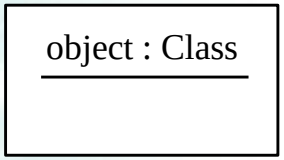
4.1 类和对象

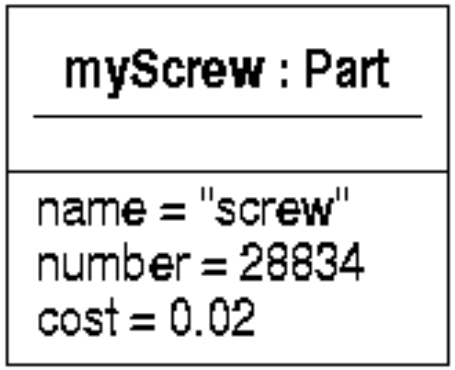
- **对象（Object）的UML图标**
  - **对象是唯一的，可以标识的；每个对象都是不同的，即使它具有相同的属性。**
  - **对象的UML图标用实线矩形框表示，含有若干分隔框。**
    - ✓ 对象名分隔框中包含对象名，其格式为：

      对象名:类名
    - ✓ 对象属性分隔框含有该对象的属性和属性值的列表（状态列表），表示该对象的并发状态。
    - ✓ 对象图标不含有操作框。









第4讲: 静态建模

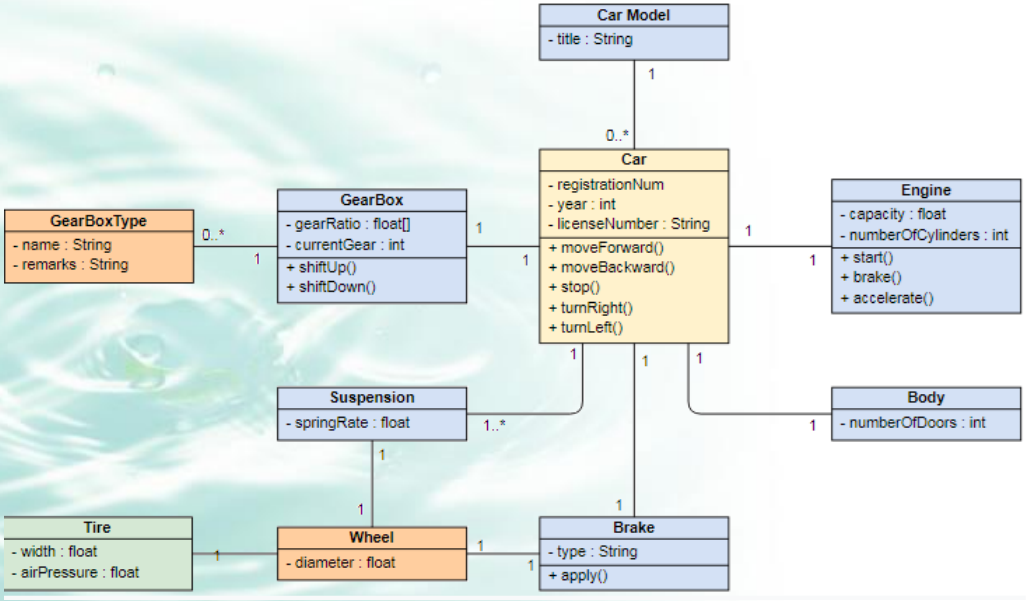
4.1 类和对象

4.2 关系

4.3 类图

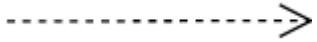
4.4 对象图

4.5 包图（模型管理图）



第4讲: 静态建模

4.2 关系

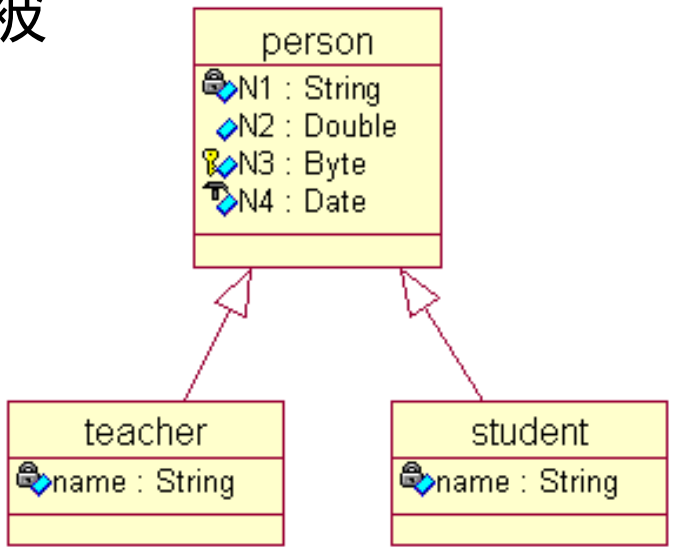
- 类之间的关系
  - **泛化（Generalization）** 
  - **实现（Realization）** 
  - **依赖（Dependence）** 
  - **关联（Association）**



第4讲: 静态建模

4.2 关系

- 泛化（Generalization）
  - **一般元素与特殊元素的关系。**
  - **目的**
    - ✓ 子类继承、共享父类的属性和操作。
    - ✓ 可以将子类的实例用于任何父类被声明使用的地方，实现多态。
  - **继承**
    - ✓ 父类的公共（public）和保护（protected）特性被子类继承。
    - ✓ 父类派生出的子类表面相同，行为不同。

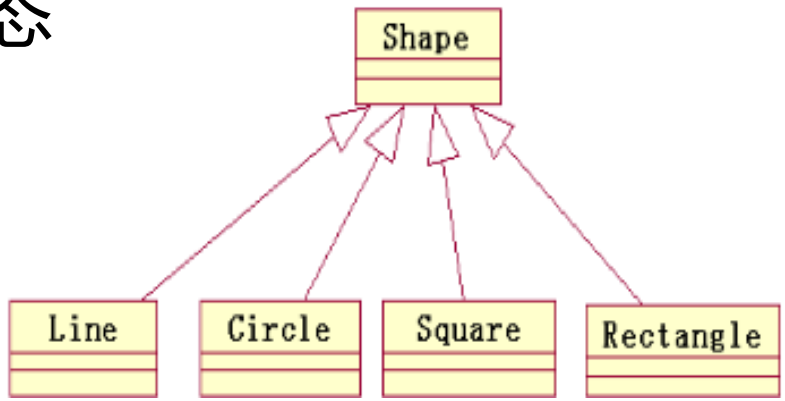


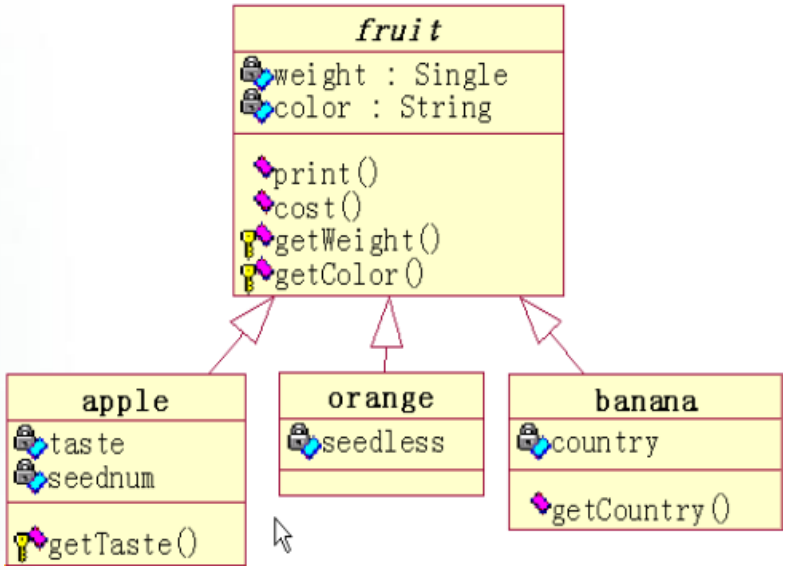
第4讲: 静态建模

4.2 关系

- 泛化（Generalization）：多态
  - ✓ 每个子类的实现方法各不相同，但外界的调用是一样的。
  - ✓ 例如：

```
Shape *myShape;
Line *myLine
myLine = new Line( );
myShape = myLine;
myShape.draw( );
```





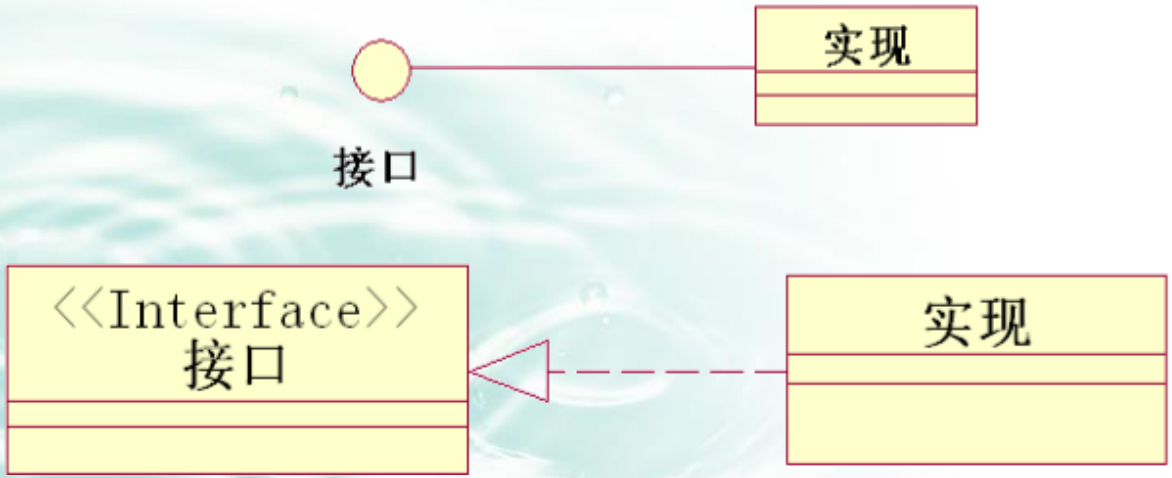
第4讲: 静态建模

4.2 关系

- 实现（Realization）

一个元素完成另外一个元素的操作功能。

- ✓ **例如：接口类及其实现。**
- ✓ **接口没有属性，方法只声明不实现，由实现类具体定义方法的实现部分。**



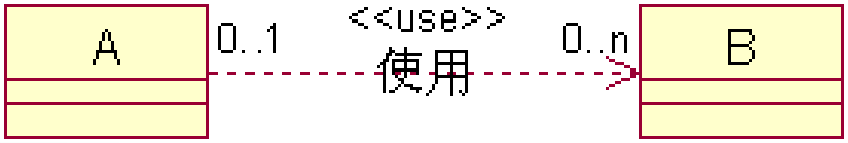
第4讲: 静态建模

4.2 关系

- 依赖（Dependence）

模型元素之间语义上的关系

- 一个元素 b 的改变会影响另外一个元素 a，则存在依赖关系“a依赖于b”。
- 关联、实现和泛化在本质上都是依赖关系。



- 分类
  - ✓ 使用依赖：使用(use)、调用(call)、参数(parameter)、发送(send)、实例化(instantiate)。
  - ✓ 抽象依赖：跟踪(trace)、精化(refine)、派生(derive)。
  - ✓ 授权依赖：访问(access)、导入(import)、友元(friend)。
  - ✓ 绑定依赖：绑定(bind)。

第4讲: 静态建模

4.2 关系

- 关联（Association）

关联用一条把类连接在一起的实线表示。

- ✓ **一个关联至少有两个关联端。**
- ✓ **每个关联端连接到一个类，关联端是有序的。**
- ✓ **关联线旁可以标出关联的名字。**
- ✓ **关联线端的箭头表示关联的方向，从源类指向目标类，箭头起关联的导航作用。**

第4讲: 静态建模

4.2 关系

- 关联（Association）
  - 关联可以是单向的或双向的（单向关联或双向关联），如果该关联是双向的，就不必标出方向箭头。
  - 在关联端可有多重性标记，规定该类中有多少个对象参与该关联。
  - 在关联的类图标旁可以标出类的角色名（Role），角色表示被关联的类参与关联的特定的行为。

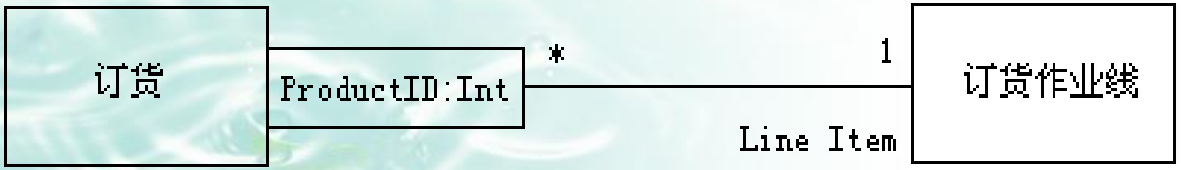
第4讲: 静态建模

4.2 关系

- 关联（Association）：限定关联

  带有限定符的关联称为限定关联（Qualified Association）。

  - ✓ **限定符的值确定如何划分和标识该关联的目标类的对象。**
  - ✓ **源类的一个带有限定符值的对象，唯一地选择目标类的一个划分。**
  - ✓ **目标类的每一个对象只能是某一个划分的成员。**



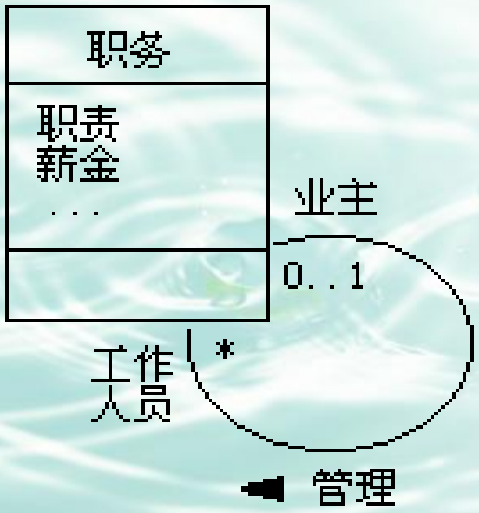
第4讲: 静态建模

4.2 关系

- 关联（Association）： **自关联**

  自关联又称递归关联（Self Association），是一个类与本身的关联，即一个类的两个对象间的联系。

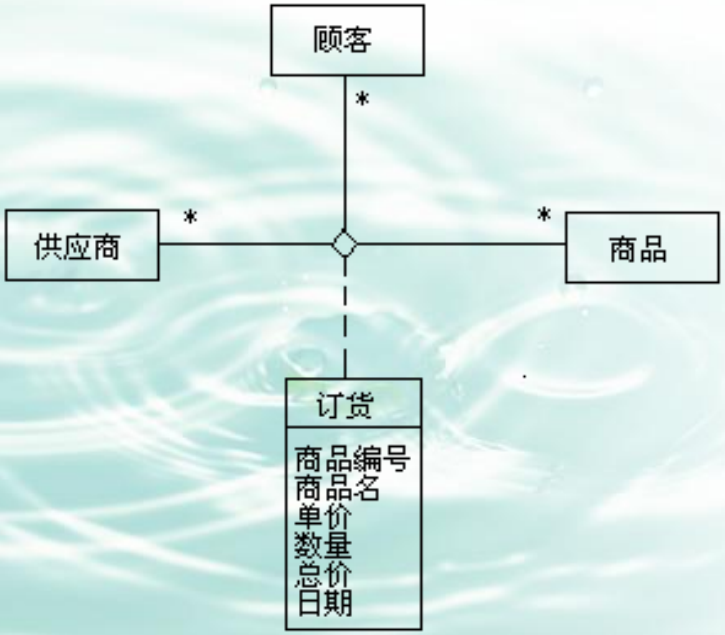
  - ✓ **自关联虽然只有一个被关联的类，但有两个关联端，每个关联端的角色不同。**
  - ✓ **例如，类“职务”存在自关联“管理”：**



第4讲: 静态建模

4.2 关系

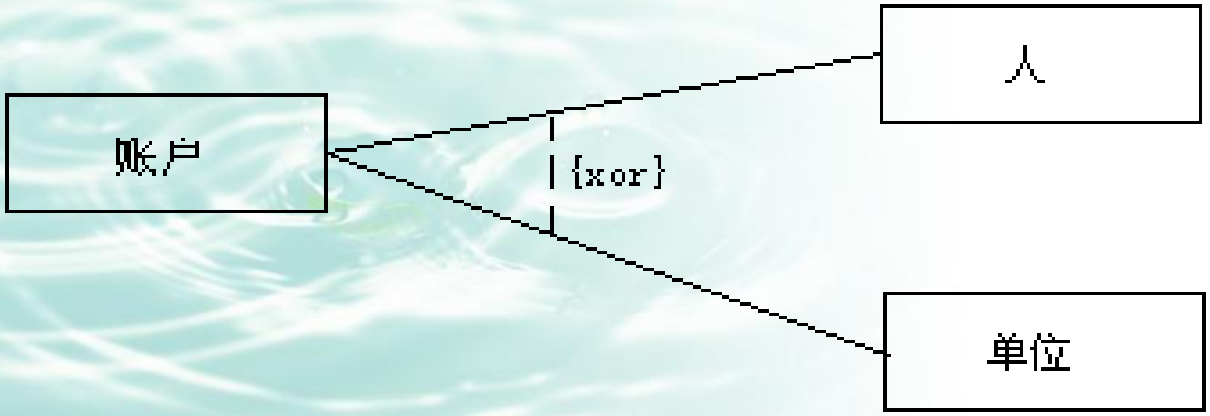
- 关联（Association）： N元关联
  - **二元关联是在两个类之间发生的关联。**
  - **N元关联是在3个或多个类之间发生的关联，N元关联的每一个实例是被关联的类的对象的多元组。**
  - ✓ **在类图上用一个菱形连接互相关联的类表示N元关联。**



第4讲: 静态建模

4.2 关系

- 关联（association）：约束
  - 关联可以加上一些约束，以规定关联的含义。
  - 约束的字符串括在花括号{}内。
  - UML定义了一些约束可施加在目标关联端上，如“implicit”、“ordered”、“changeable”、“addonly ”、“xor”等。
  - 例：具有xor约束的关联，代表一组关联的互斥的情况。

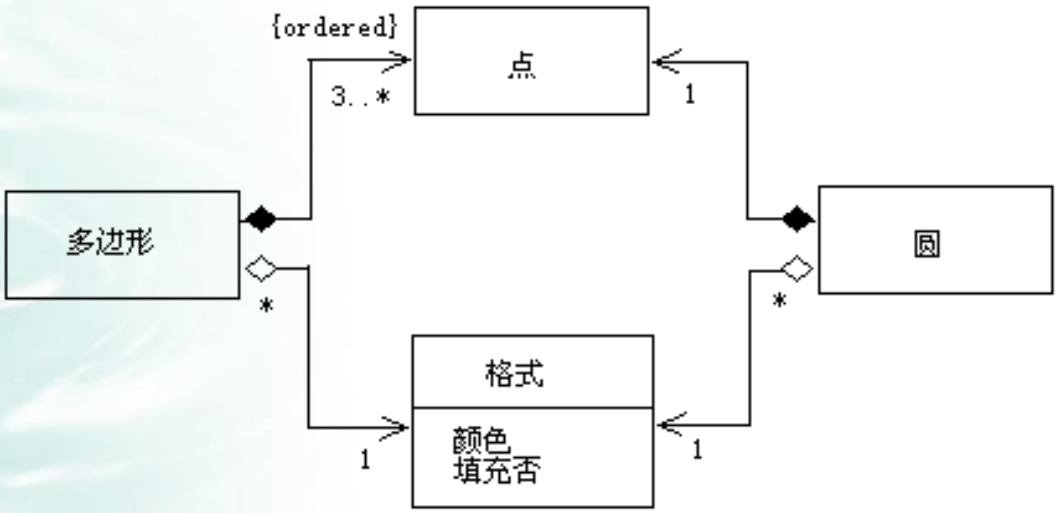


第4讲: 静态建模

4.2 关系

- 关联（association）：聚合（Aggregation）
  - ✓ 表示事物的部分/整体关系的较弱的情况。聚合也称为“has-a”联系。
  - ✓ 在关联线端加一个小**空心菱形**表示聚合，菱形连接代表整体事物的类，称之为聚合类，另一个关联端连接代表部分事物的类。

例：圆和多边形是图形格式的两个聚合类

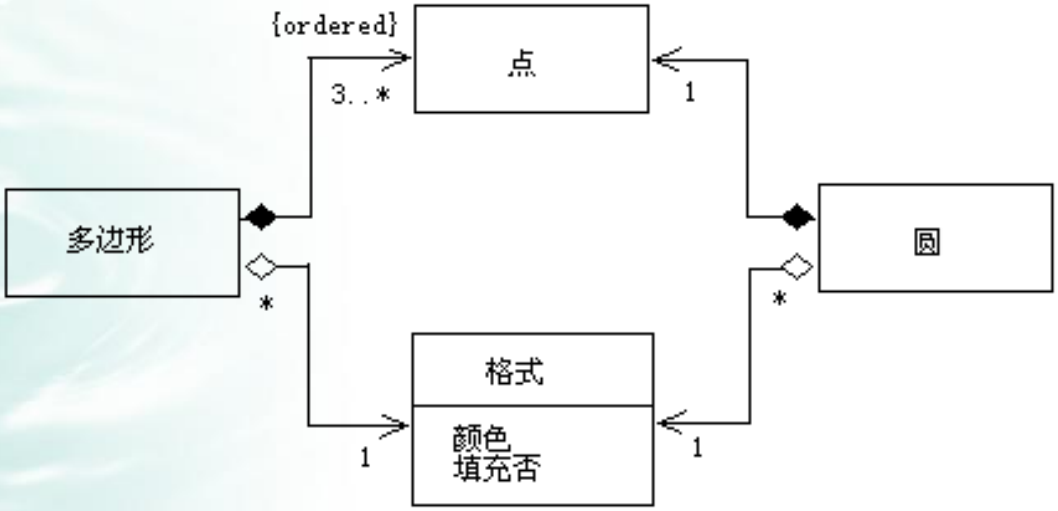


第4讲: 静态建模

4.2 关系

- 关联（association）：组合（Composition）
  - ✓ 表示事物的部分/整体关系的较强的情况。组合也称为“contains-a”联系。
  - ✓ 在关联线端加一个小**实心菱形**表示组合，菱形连接代表整体事物的类，称之为组合类，另一个关联端连接代表部分事物的类。

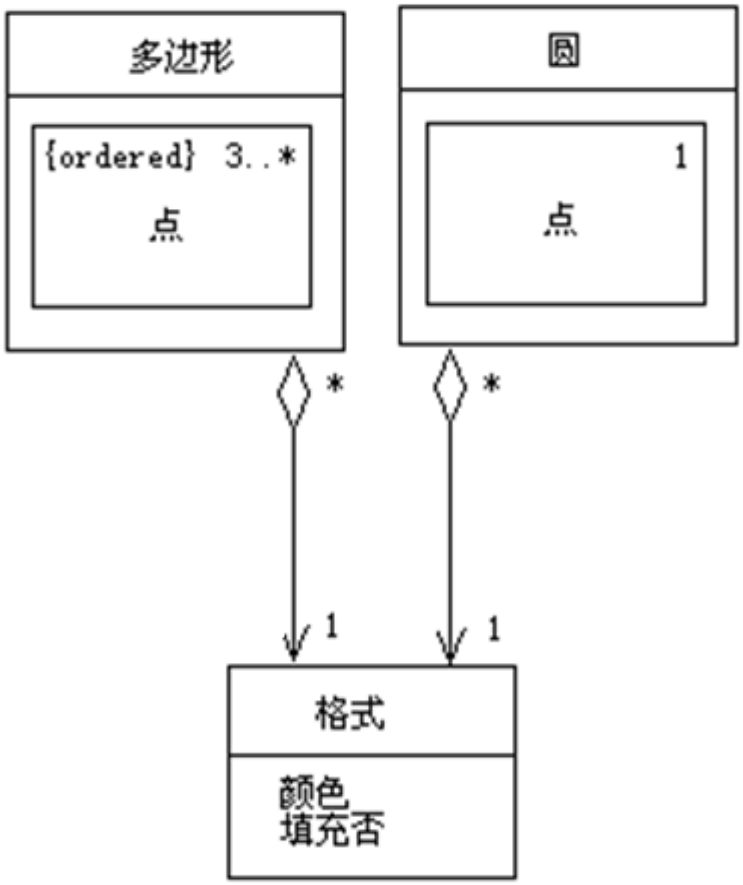
例：圆由点组成，“圆”是组合类，“点”是成分类；多边形也是由点组成的，是一个组合类。



第4讲: 静态建模

4.2 关系

- 关联（association）：组合（Composition）
  - 聚合与组合表示的部分/整体结构关系对系统建模具有重要的作用：简化了对象的定义、支持软件重用。
  - 组合的另一种表示方式：把成分类放在它的组合类的属性框中，在其右上角可以标出多重性标记。成分类的名字可以按格式写为：“角色名:类名”。

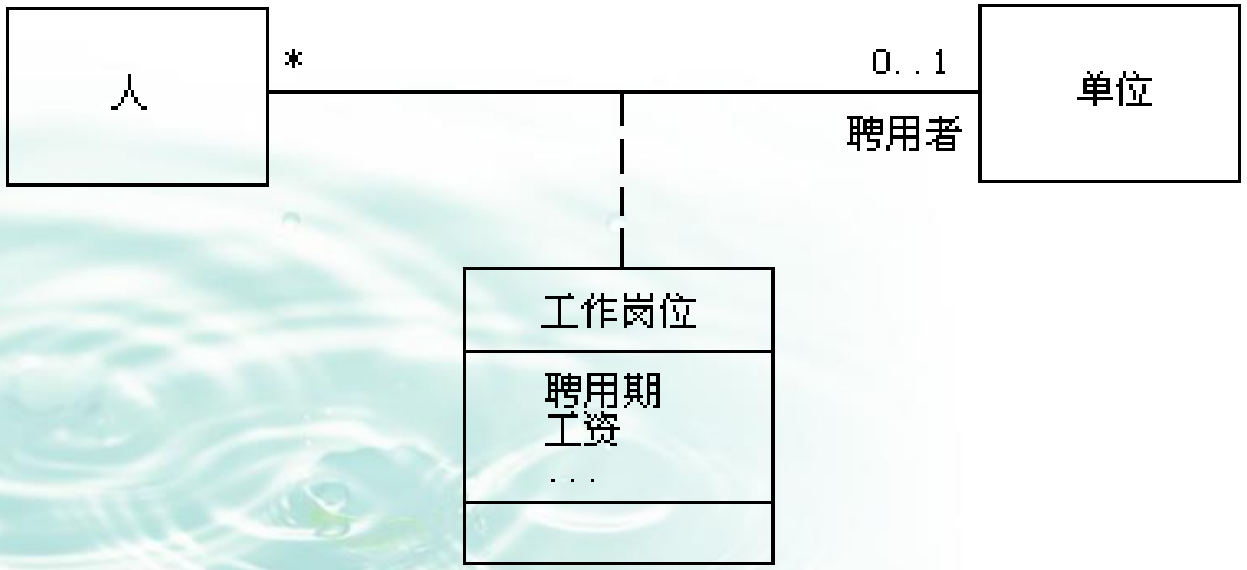


第4讲: 静态建模

4.2 关系

- 关联（Association）：关联类（Association class）

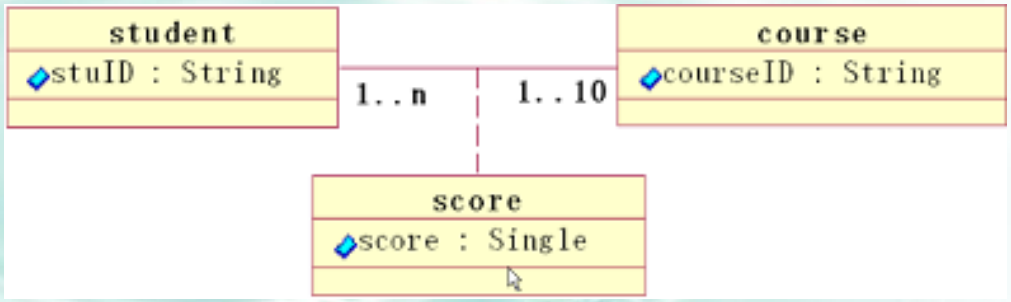
  关联本身也有特性，通过关联类（Association Class）可以进一步描述关联的属性、操作，以及其他信息。

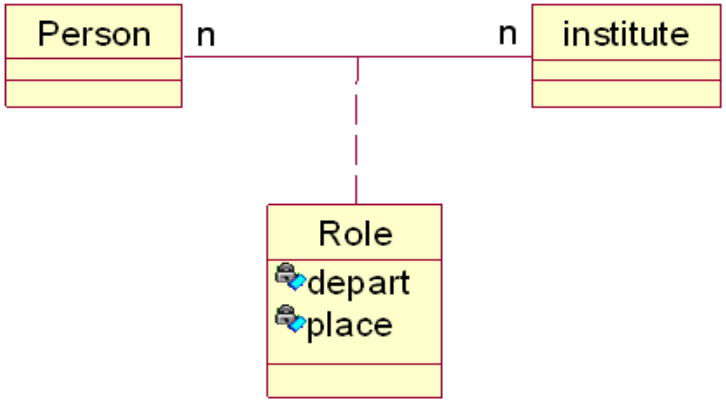


第4讲: 静态建模

4.2 关系

- 关联（Association）：关联类（Association class）
  - ✓ 既是关联又是类，有属于关联类的属性
  - ✓ 两个类之间具有多对多的关系，并且有些属性不属于关联两端任何一个类

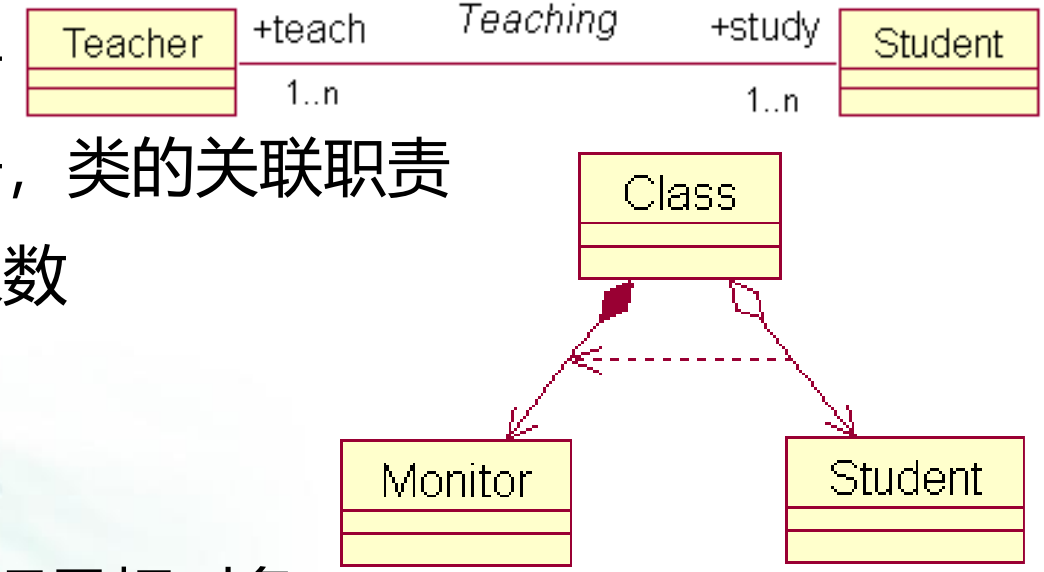


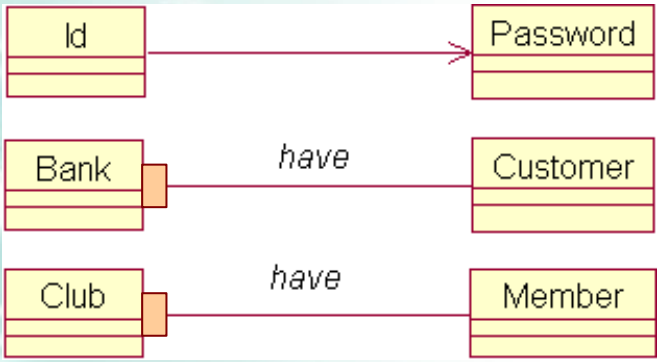


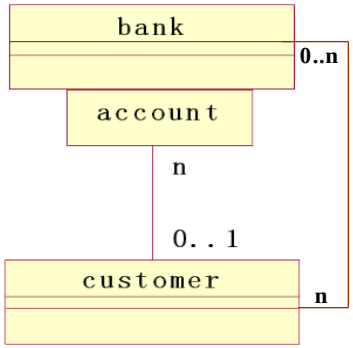
第4讲: 静态建模

4.2 关系

- 关联（association）总结
  - 名称：动词或动词短语
  - 角色：名词或名词短语，类的关联职责
  - 多重性：类关联的对象数
  - 约束/依赖：
  - 导航性：
    - ✓ 表示源对象可以访问目标对象
  - 限定符







第4讲: 静态建模

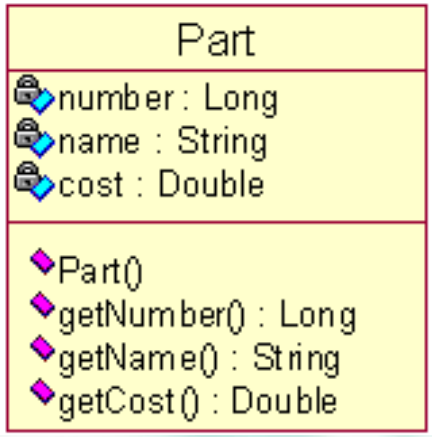
4.2 关系

- 关联（association）总结
  - 双向关联
  - **单向关联**
  - 限定关联
  - 自关联
  - N元关联
  - 聚合关联
    - ✓ 弱的整体与部分关系
  - 组合关联
    - ✓ 强的整体与部分关系
  - 关联类

第4讲: 静态建模

4.2 关系

- 关联（association）：单向关联
  - **代码与UML类图**



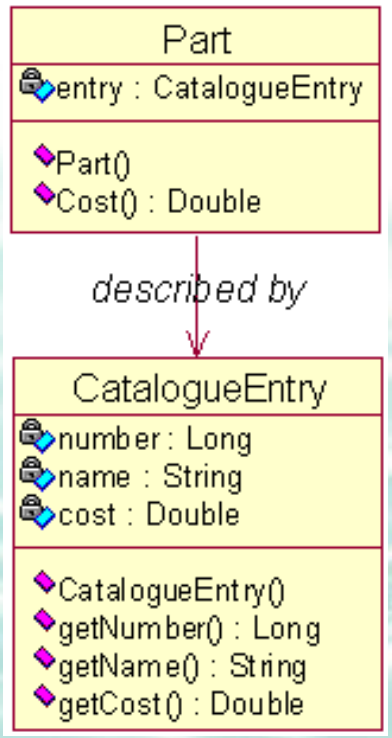

```
public class Part {
        private long number;
        private string name;
        private double cost;
        public Part(long num, string nm, double cst) {
                name = nm ;
                number = num ;
                cost = cst ; }
        public string getName() {
                return name; }
        public long getNumber() {
                return number; }
        public double getCost() {
                return cost; }
    }
```

  - **问题**
    - ✓ 同一产品组件，包含多个同类型零件，零件对象中的数据信息重复
    - ✓ 为避免这种数据重复，需要分离Part类

第4讲: 静态建模

4.2 关系

- 关联（association）：单向关联
  - ✓ 类Part与类CatalogueEntry的单向关联




```
public class Part {
        private CatalogueEntry entry ;
        public Part(CatalogueEntry e) {
                entry = e ; }
        public double cost() {
                return entry.getCost; }
    }
```

```
public class CatalogueEntry {
        private long number ;
        private string name ;
        private double cost ;
        public CatalogueEntry(long num, string nm, double cst) {
                name = nm ;
                number = num ;
                cost = cst ; }
        public string getName() {
                return name; }
        public long getNumber() {
                return number; }
        public double getCost() {
                return cost; }
    }
```

第4讲: 静态建模

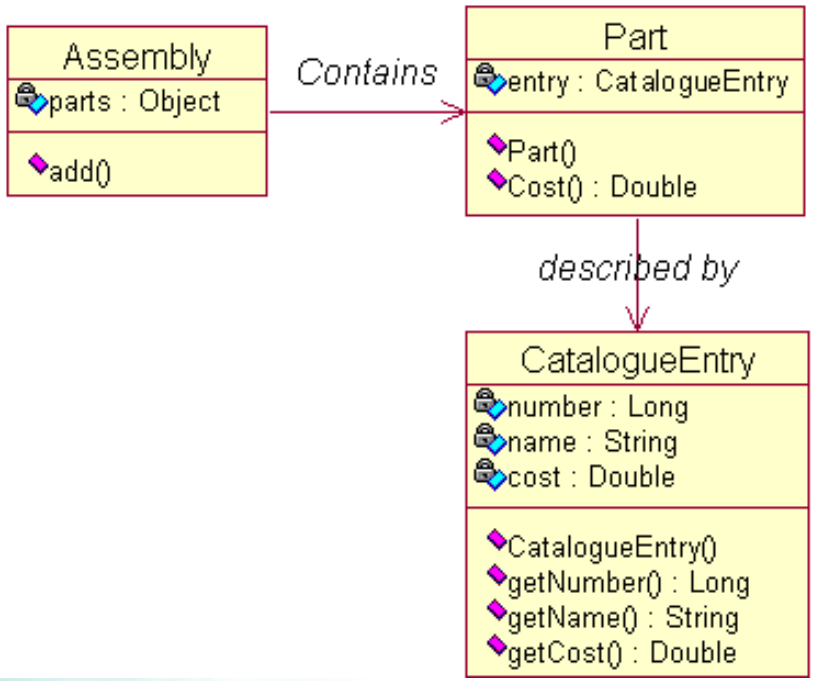
4.2 关系

- 关联（association）：单向关联
  - ✓ **将产品（组件）看作是若干个零件的组合向量**

```
public class Assembly {
      private Vector parts = new Vector() ;
      public void add(Part p) {
            parts.addElement(p) ; }
   }
```

```
public class Part {
      private CatalogueEntry entry ;
      public Part(CatalogueEntry e) {
            entry = e ; }
      … …
   }
```

```
public class CatalogueEntry {
      private long number ;
      private string name ;
      private double cost ;
      public CatalogueEntry(... …) {
            name = nm ;
            number = num ;
            cost = cst ; }
      … …
   }
```



第4讲: 静态建模

4.2 关系

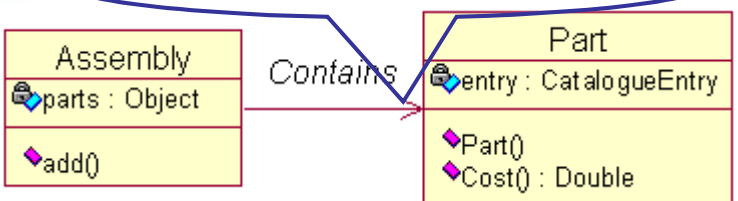
- 关联（association）：单向关联
  - **复杂视角**
    - ✓ 若干个零件组成子组件。
    - ✓ 若干个子组件+若干个零件组成更高一级的子组件。
    - ✓ ……
    - ✓ 产品是若干复杂子组件+零件的组合。
  - **问题**
    - ✓ 现有的类设计不满足这种视角，Assembly类只能包含Part类。
    - ✓ 引入**多态机制**

      无论是程序设计语言，还是UML，都是强类型的，不允许链接任意类的对象。

```
public class Assembly {
      private Vector parts = new Vector() ;
      public void add(Part p) {
            parts.addElement(p) ; }
  }
```

```
public class Part {
      private CatalogueEntry entry ;
      public Part(CatalogueEntry e) {
            entry = e ; }
      ……
  }
```

约束为只能包含 **Part** 类



第4讲: 静态建模

4.2 关系

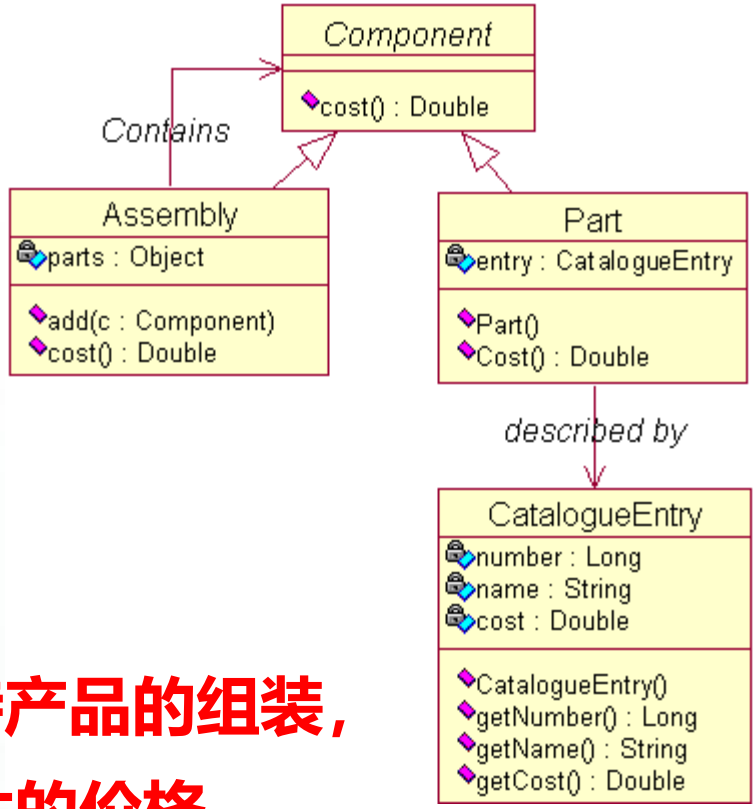
- 关联（association）：单向关联
- **多态机制**

引入一个***抽象类***，表达Assembly类的链接范围，真正链接的可以是若干个抽象类的特化。

```
public abstract class Component {
      public abstract double cost() ;
   }
```

```
public class Assembly extends Component {
      private Vector parts = new Vector() ;
      public void add(Component c) {… …}
      public double cost() {…算法 A …}
   }
```

```
public class Part extends Component {
      private CatalogueEntry entry ;
      public Part(CatalogueEntry e) {
            entry = e ; }
      public double cost() {…算法 B…}
   }
```



- **注：不但解决了从任意视角看待产品的组装，还可以计算产品任意子组件的价格。**

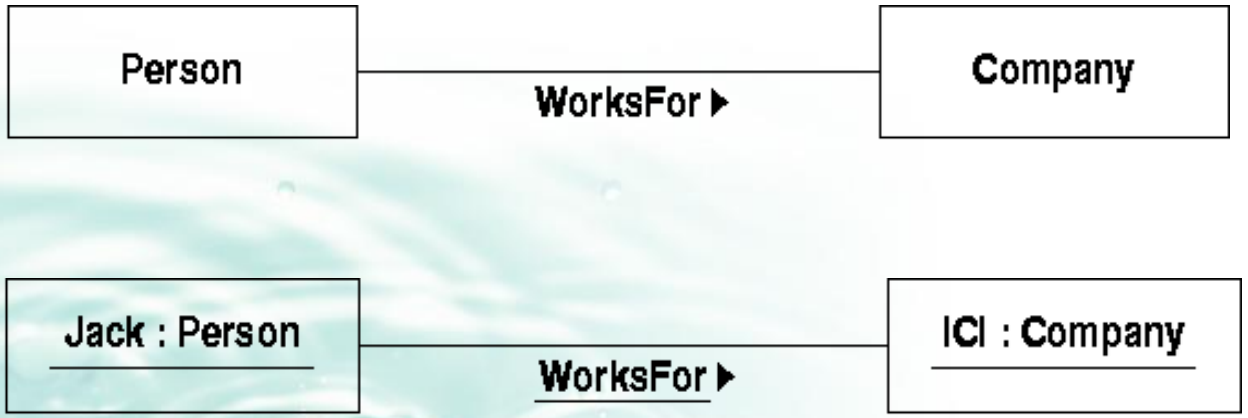
第4讲: 静态建模

4.2 关系

- 对象之间的关系

  链接（Link）

  ✓ **意味着导航，代表一个对象对另一对象的引用**



第4讲: 静态建模

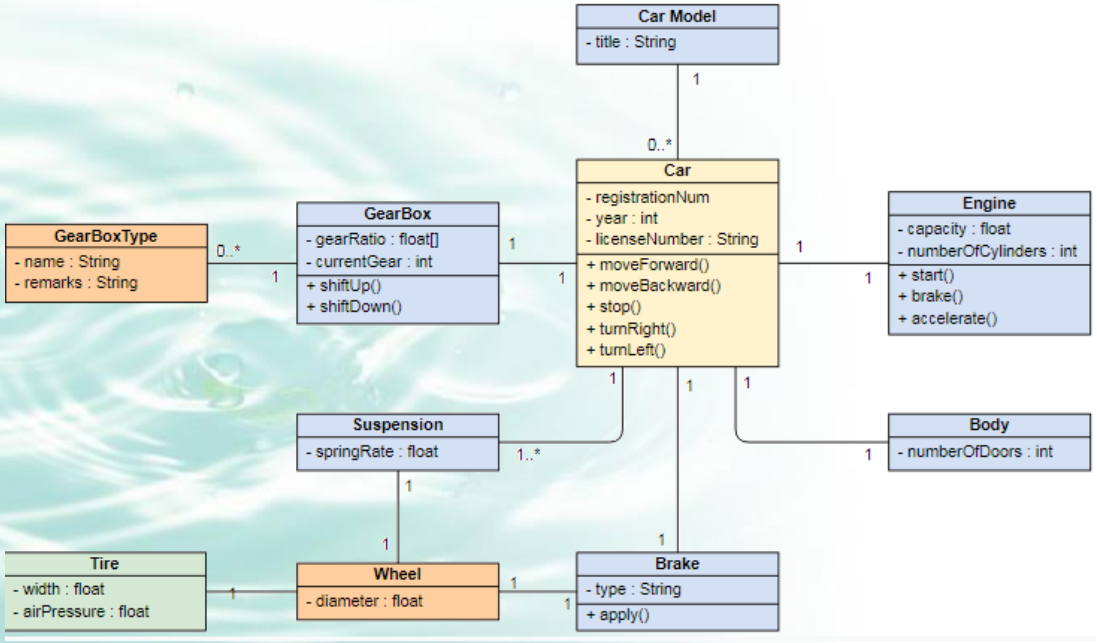
4.1 类和对象

4.2 关系

4.3 类图

4.4 对象图

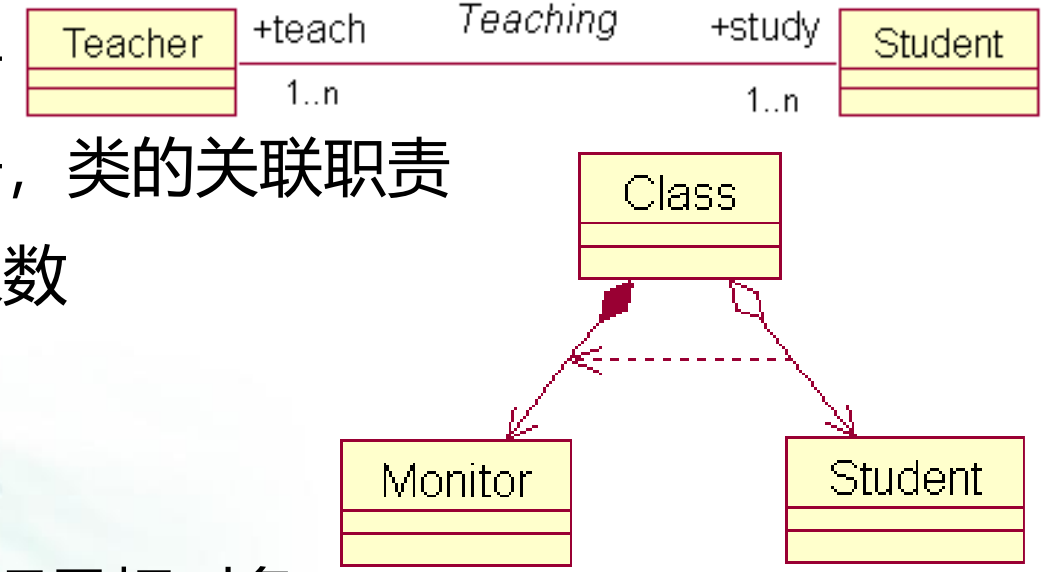
4.5 包图（模型管理图）

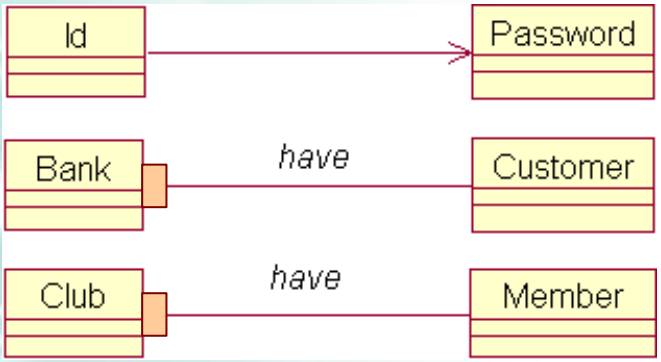


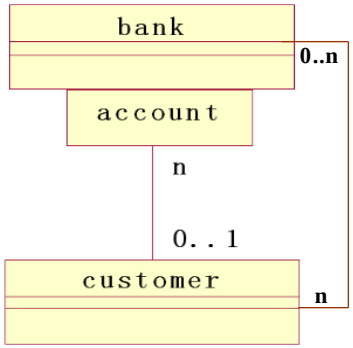
第4讲: 静态建模

4.2 关系

- 关联（association）总结
  - 名称：动词或动词短语
  - 角色：名词或名词短语，类的关联职责
  - 多重性：类关联的对象数
  - 约束/依赖：
  - 导航性：
    - ✓ 表示源对象可以访问目标对象
  - 限定符







第4讲: 静态建模

4.2 关系

- 关联（association）总结
  - 双向关联
  - **单向关联**
  - 限定关联
  - 自关联
  - N元关联
  - 聚合关联
    - ✓ 弱的整体与部分关系
  - 组合关联
    - ✓ 强的整体与部分关系
  - 关联类

第4讲: 静态建模

4.3 类图

- **类图是较广泛使用的UML模型，描述系统的结构**
  - **表述类、协作、接口及其关系。**
  - **类图元素：类、接口、协作、关系、注释、约束等**
    - ✓关系：连接类、协作与接口。
    - ✓注释：对类和接口进行说明。
    - ✓约束：对类和接口进行约束。
  - **类图模型建立过程**
    - ✓寻找类：寻找备选类，从备选类中筛选出候选类。
    - ✓确定类关系，给关联加属性。
    - ✓确定类的职责：类所维护的信息、类提供的行为。

第4讲: 静态建模

4.3 类图

- 例：家庭图书管理
  - 有一个爱书的人，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，但不能够删除记录。该系统还应该对书籍的外借情况进行记录，显示外借情况列表。另外，还希望对书籍的购买金额、册数按特定时限进行统计。

第4讲: 静态建模

4.3 类图

- 例：家庭图书管理
  - 有一个爱书的人，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，但不能够删除记录。
  - ✓ 该系统还应该对书籍的外借情况进行记录，显示外借情况列表。
  - ✓ 另外，还希望对书籍的购买金额、册数按特定时限进行统计。

第4讲: 静态建模

4.3 类图

- 例：家庭图书管理：**寻找分析类**

➢ **寻找备选类**

✓ 名词、名词短语→备选类列表

➢ **从备选类中筛选出候选类**

✓ 人、朋友、家→系统外概念，无需建模

✓ 图书管理系统、系统→待开发系统，无须建模

✓ 书籍

- 书号、书名、作者、类别、出版社、单价→属性
- 规则→构造函数

✓ 关键字、功能、新书籍、信息、记录等→描述需求，非问题本质，无须建模

第4讲: 静态建模

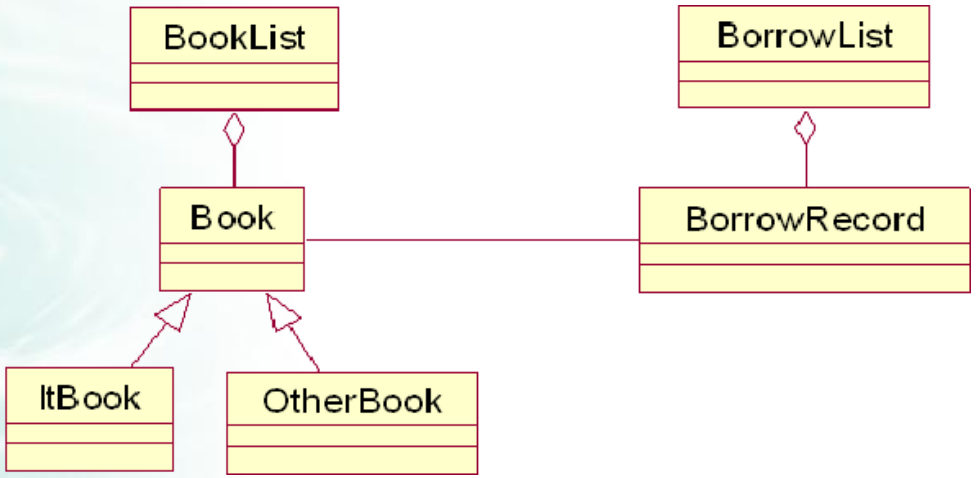
4.3 类图

- 例：家庭图书管理：**寻找分析类**
  - 计算机类书籍、非计算机类书籍
  - 借阅记录、借阅记录列表
    - ✓ 外借情况（主体是朋友）、外借情况列表
  - 书籍列表→执行统计的主体
    - ✓ 购买金额、册数、特定时限，和统计有关
- **候选类**
  - 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

第4讲: 静态建模

4.3 类图

- 例：家庭图书管理：**确定类关系**
  - 泛化：计算机类书籍 (ItBook)、非计算机类书籍(OtherBook) 是书籍类 (Book) 的派生
  - 聚合（弱的整体与部分关系）
    - ✓ “书籍列表”(BookList) 是由“书籍”(Book) 组成。
    - ✓ “借阅记录列表”(BorrowList) 是由“借阅记录”(BorrowRecord) 组成。
  - 双向关联
    - ✓ “借阅记录”和“书籍”
    - ✓ 浏览书籍时，会看到书的外借信息；归还时，从借阅记录也能关联到书籍。



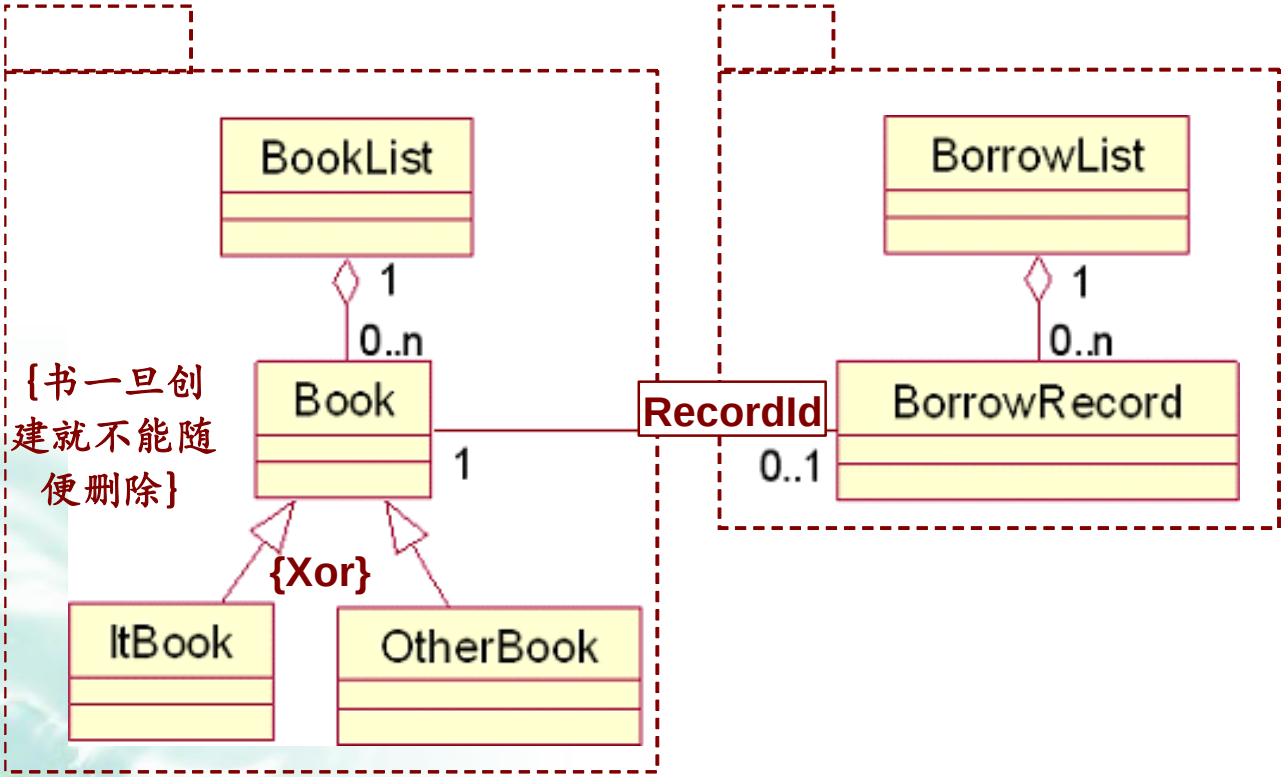
第4讲: 静态建模

4.3 类图

- 例：家庭图书管理：**给关联加属性**

➢ **关联的属性**

- ✓ 名称
- ✓ 角色
- ✓ 多重性
- ✓ 导航型
- ✓ 约束
- ✓ 限定符



➢ **注**

- ✓ 复杂的系统将关联密切的类组成包，以便更好地组织子系统

第4讲: 静态建模

4.3 类图

- 例：家庭图书管理：**类的职责**

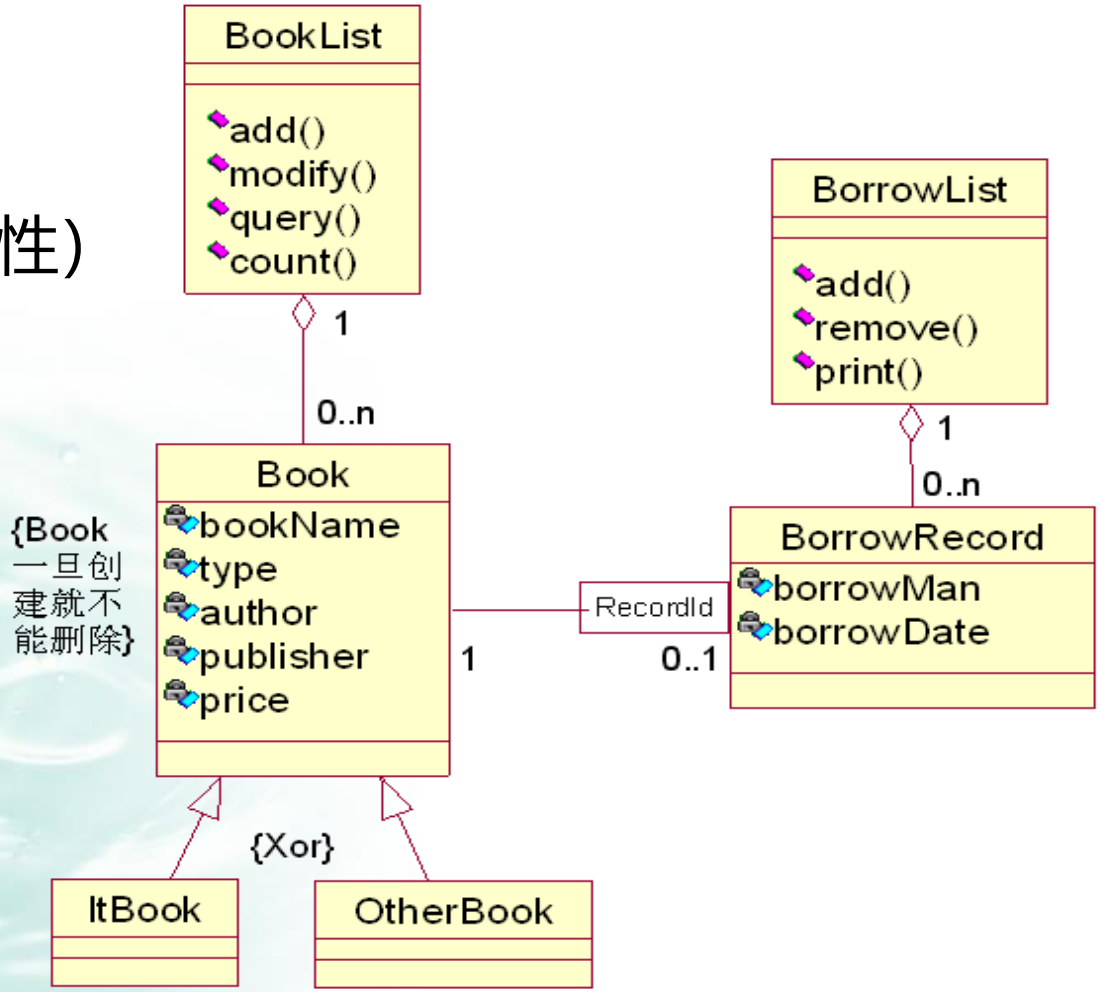
➢ **类职责**

✓ 类所维护的信息

- 成员变量（属性）

✓ 类提供的行为

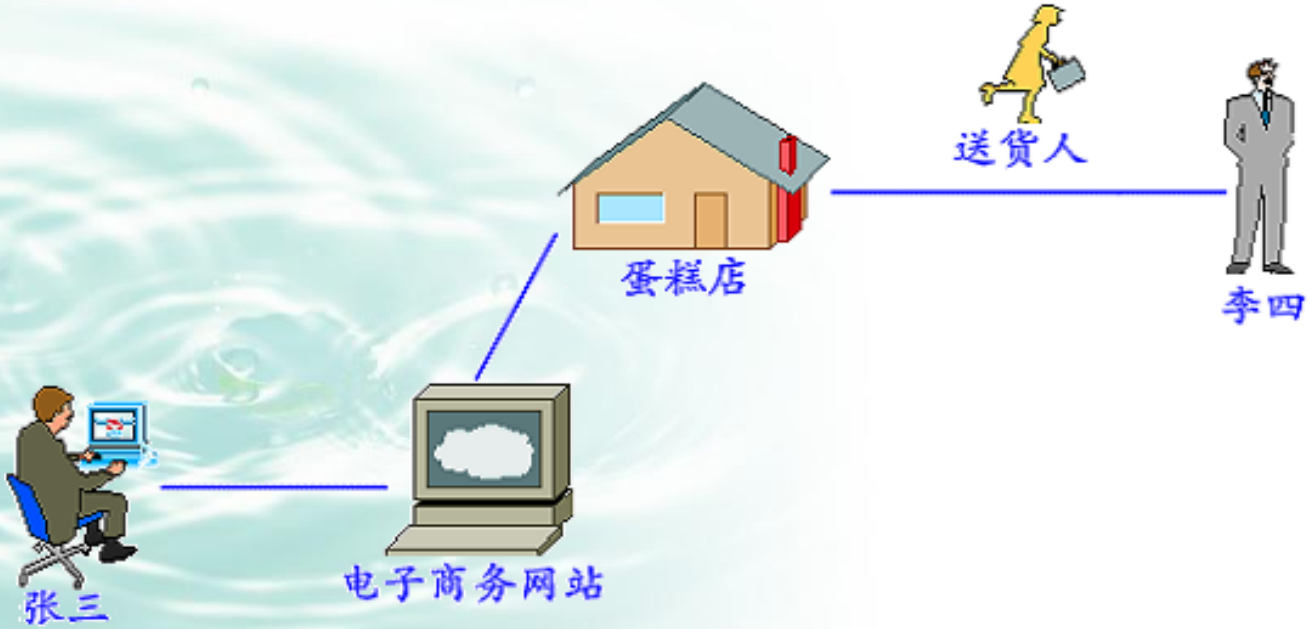
- 成员方法



第4讲: 静态建模

4.3 类图

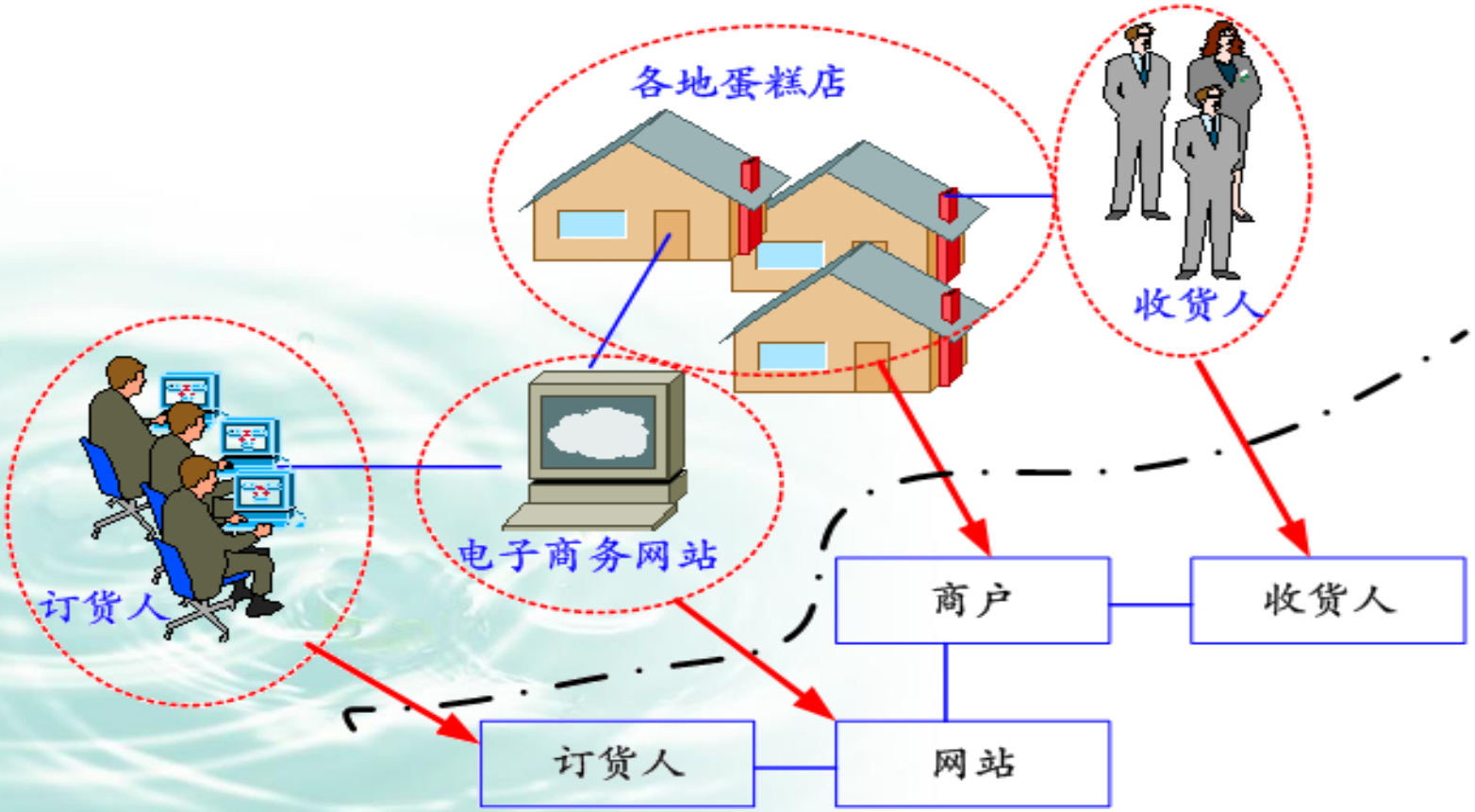
- 练习：**电子商务网站**
  - **需求陈述**
    - 张三给朋友李四（可能在异地）送生日蛋糕
    - 在电子商务网站的蛋糕网店订蛋糕
    - 蛋糕实体店收到订单，派遣送货员给李四送蛋糕



第4讲: 静态建模

4.3 类图

- 练习：**电子商务网站**
  - **抽取业务模型**



第4讲: 静态建模

4.3 类图

- 练习：**电子商务网站**
  - **类**
    - ✓ 顾客（订货人）类：Customer
    - ✓ 收货人类：Consignee
    - ✓ 产品类：Product
    - ✓ 商户类：Peddlery
    - ✓ 订单类：Order
    - ✓ 订单项类：OrderItem
    - ✓ 送货单类：DeliverOrder

第4讲: 静态建模

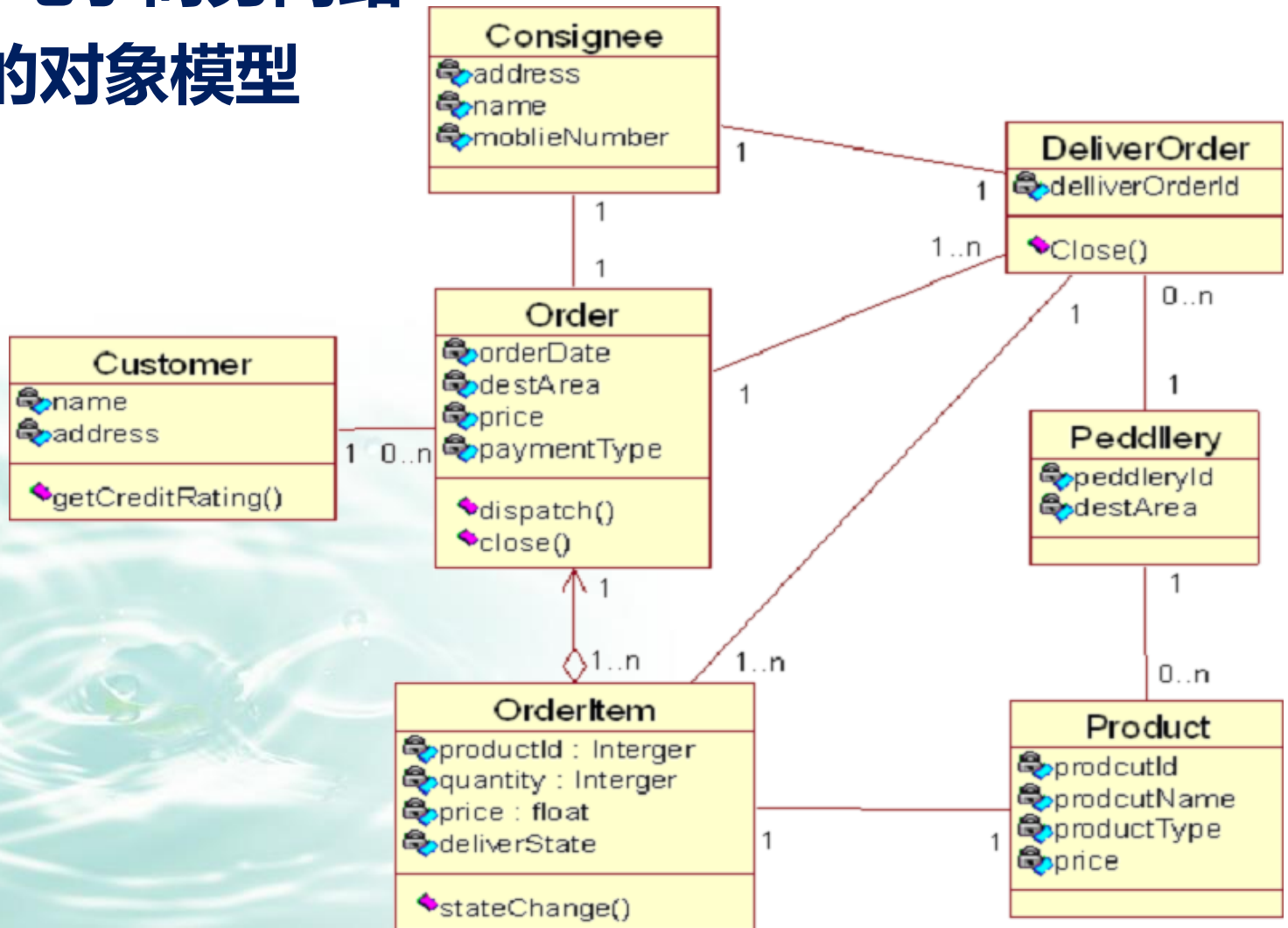
4.3 类图

- 练习：**电子商务网站**
  - **类关系**
    - ✓ 组合关系
      - Order与OrderItem
    - ✓ 关联关系
      - Order与Customer、Consignee、 DeliverOrder
      - DeliverOrder与Order、 OrderItem、Consignee 、Peddlery
      - Product与Peddlery、OrderItem

第4讲: 静态建模

4.3 类图

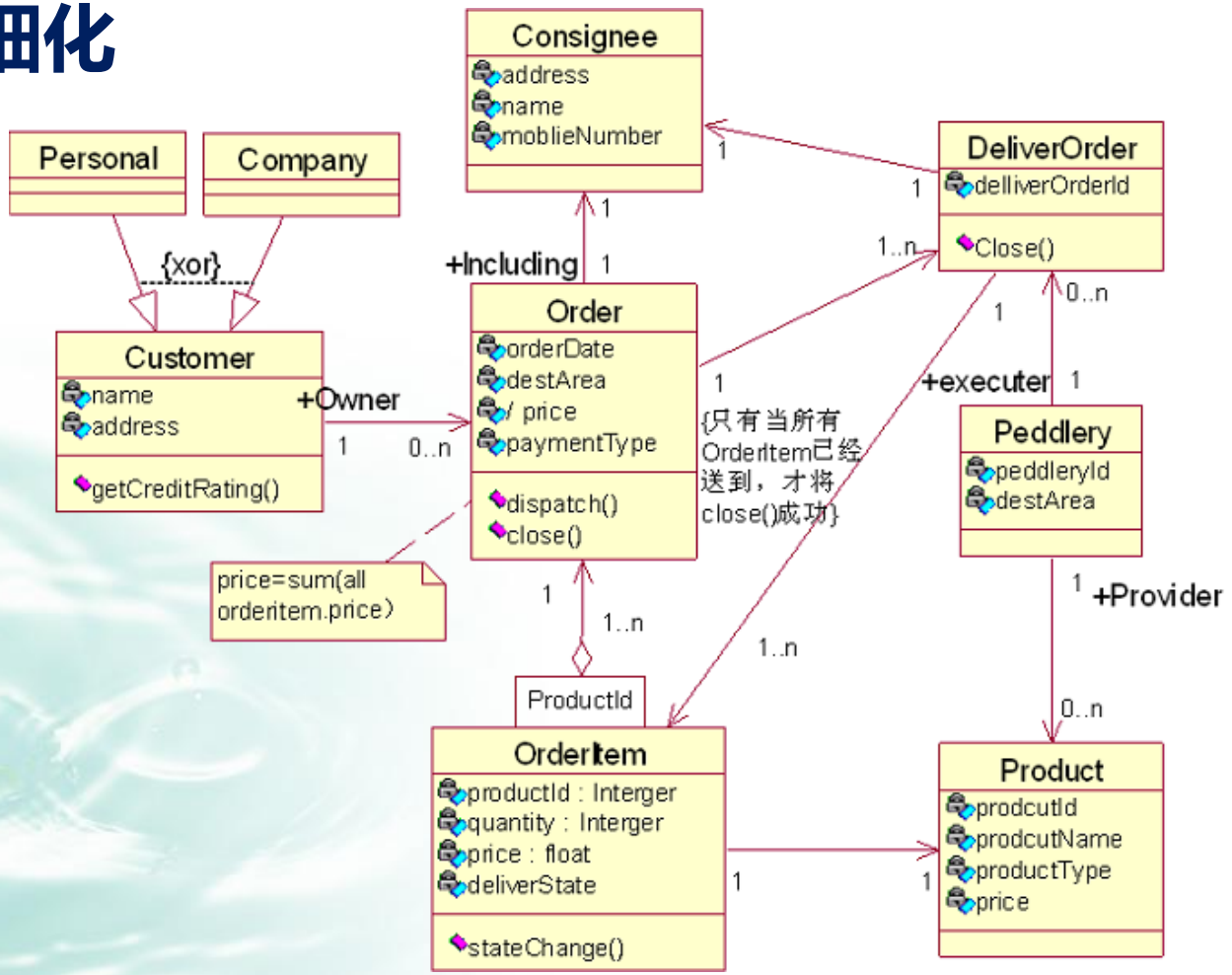
- 练习：**电子商务网站**
  - **初步的对象模型**



第4讲: 静态建模

4.3 类图

- 练习：**电子商务网站**
- **对象模型的细化**
  - ✓ 关联的属性
    - 多重性
    - 导航性
    - 角色
    - 限定符
    - 约束
  - ✓ 类的职责
    - 属性
    - 行为



第4讲: 静态建模

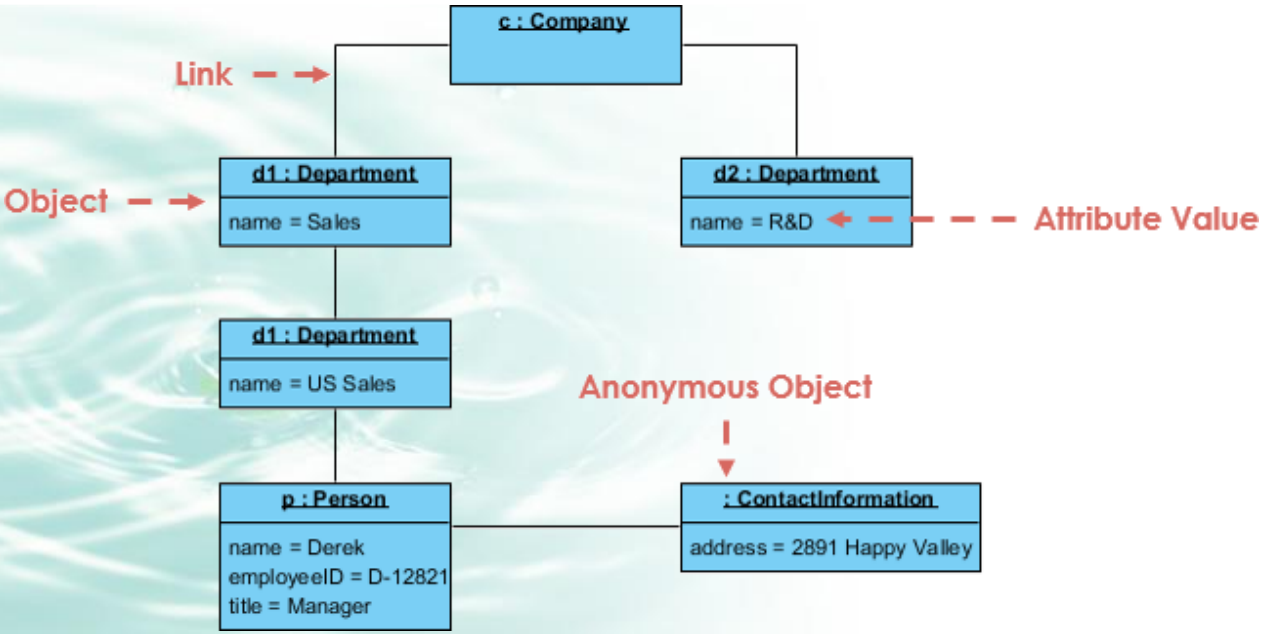
4.1 类和对象

4.2 关系

4.3 类图

4.4 对象图

4.5 包图（模型管理图）



第4讲: 静态建模

4.4 对象图

- 对象图：表示对象和对象之间链接关系的图,类图的实例。
  - 表示对象和对象之间链接关系的图。
  - 表达了系统数据在给定时刻的一个“快照”，代表系统的一个局部状态。
  - 全部对象的链接图，表达了系统的对象模型，对象模型是软件开发OO方法学的核心。

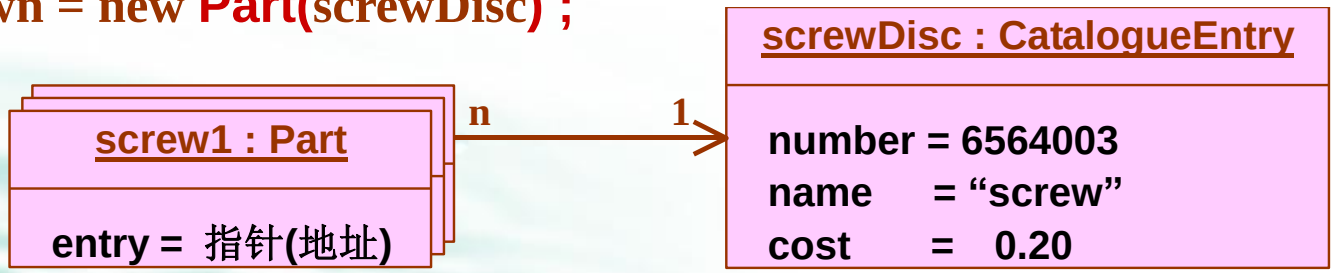
第4讲: 静态建模

4.4 对象图

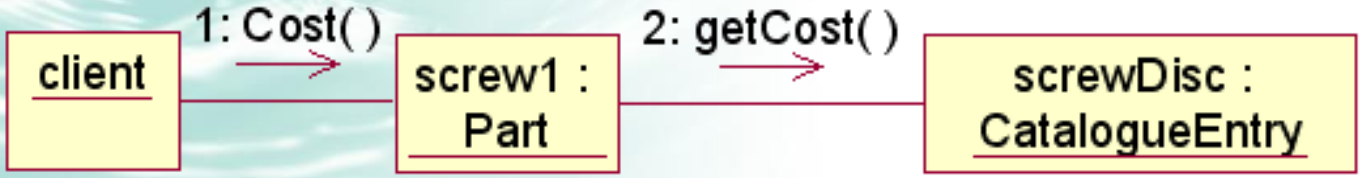
- 例：库存控制系统

➢ 对象创建

```
CatalogueEntry screwDisc = new CatalogueEntry(6564003, “screw”, 0.20) ;

Part screw1 = new Part(screwDisc) ;
    … … … …
Part screwn = new Part(screwDisc) ;
```



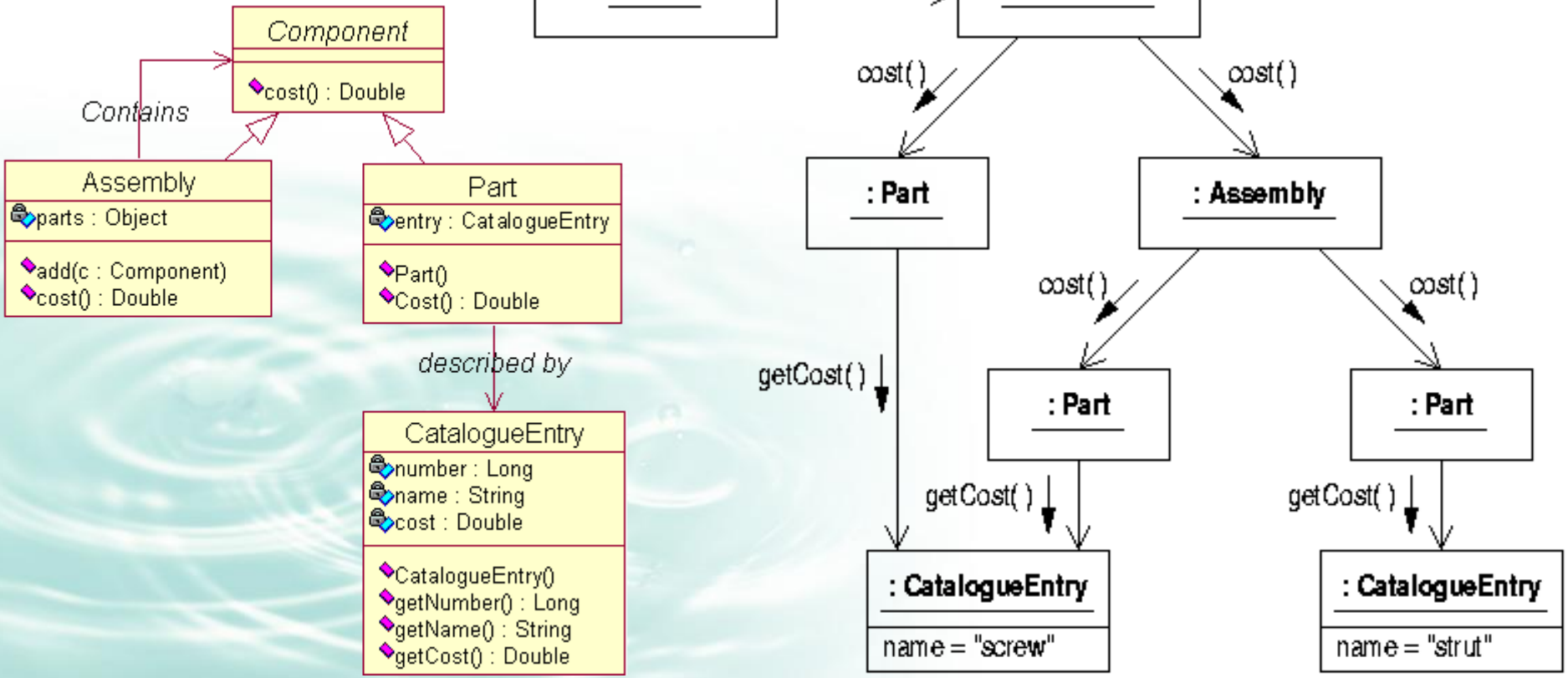
➢ 消息传递（协作图）



第4讲: 静态建模

4.4 对象图

- 例：库存控制系统
  - **计算组件总价格（含计算配件价格的消息传递）**



第4讲: 静态建模

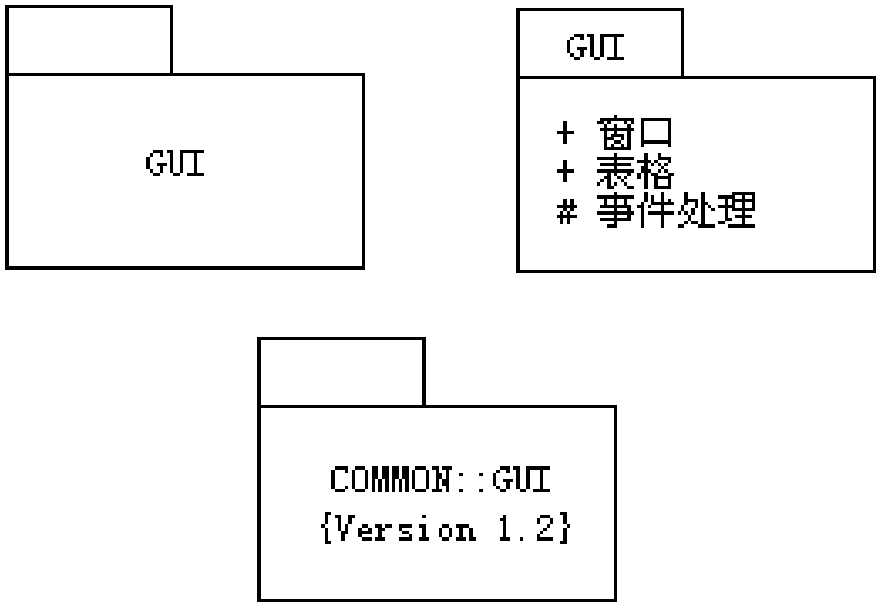
4.1 类和对象

4.2 关系

4.3 类图

4.4 对象图

4.5 包图（模型管理图）



第4讲: 静态建模

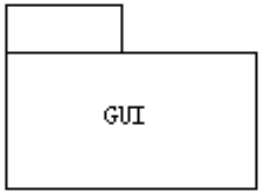
4.5 包图（模型管理图）

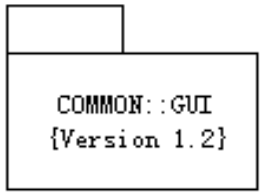
- **在对一个大型的软件系统建立模型时往往需要面对和处理大量的模型元素：类、接口、构件、节点等。**
  - **把语义上相近的、可能一起变更的模型元素组织在同一个包里**，便于理解复杂的系统，控制系统结构各部分间的接缝。
  - 包是一种概念性的模型管理的图形工具，只在软件的开发过程中存在。
  - ✓包可以用于组织一个系统模型
  - ✓一个系统的框架、模型、子系统等也都可以看作是特殊的包。

第4讲: 静态建模

4.5 包图

- **包（Package）是一种对模型元素进行成组组织的通用机制。**
  - 包用于定义一个名字空间（Namespace）或容器（Container），它本身是UML的一种模型元素。
  - 对包中的元素作为一个整体对待，并且控制它们的可视性和存取。
  - 包的图标是一个大矩形（内容框）的左上角带一个小矩形（名字框）
    - ✓ 名字可是一个简单名或路径名。
    - ✓ 在包名之后或之下，可用括在花括号中的文字（约束）说明包的性质，如“{abstract}”、“{version}”等





第4讲: 静态建模

4.5 包图

- **一个包可以拥有一个或多个模型元素，包括类、接口、组件、节点、协同、用例、图等，甚至拥有其他包。**
  - 所有UML的模型元素都可以放入包内。通常，一个包拥有的是类或其他的包。
  - 包与它所含的模型元素之间的关系是一种组合联系，即一个包由一个或多个模型元素组成，每一个模型元素在该包中声明。
    - ✓ 一个模型元素只能为一个包唯一地拥有。一个包消失，它所拥有的全部模型元素也随之消失。
    - ✓ 不同包的模型元素可以同名，但在同一个包中的模型元素不能同名。

第4讲: 静态建模

4.5 包图

- **包的模型元素名前可以有可视性标记。**
  - 分别用“+”、“#”、“-”表示可视性为“公共”、“保护”、“私用”。
- **包只是一种组织模型元素的容器，它没有实例，只有内含的内容，即类、接口等模型元素。**
  - 包纯粹是一种概念性的建模工具，它与组件不同。
  - ✓包只在软件开发过程中存在，类似一个有标签的文件夹，其中包含有包的名字和内容。
  - ✓组件是系统组成部分，既存在于软件开发过程中，也存在于系统运行期间。
  - 包内的模型元素具有较强的内聚性，不同的包的元素之间的耦合性很弱。

第4讲: 静态建模

4.5 包图

- **包可以拥有其他包作为包内的元素，子包又可以拥有自己的子包，这样可以构成一个系统的嵌套结构，以表达系统模型元素的静态结构关系。**
  - 包的嵌套可以清晰地表现系统模型元素之间的相互关系。但嵌套不宜过深，包的嵌套层数一般以2～3层为宜。
  - 包与它的元素的组成关系可以用树形结构表示，这对于表达概念模型是有用的。
  - 对于一个包可视的元素，对于该包的内嵌套子包也是可视的。

第4讲: 静态建模

4.5 包图

- **对于一个包可以加上构造型或标记值说明其特定的性质，如说明包的开发者，包所提供的服务等。**
  - UML预定义的用于包的构造型有：<<facade>>、<<framework>>、<<stub>>、<<subsystem>>、<<system>>等。
    - ✓ 构造型<<facade>>说明一个包仅仅是其他一些包的视图。
    - ✓ 构造型<<framework>>说明一个包代表模型架构。
    - ✓ 构造型<<stub>>说明一个包是另一个包的公共内容的服务代理。
    - ✓ 构造型<<subsystem>>说明一个包代表系统模型的一个独立部分，即子系统。
    - ✓ 构造型<<system>>说明一个包代表系统模型。
    - ✓ 构造型<<facade>>和<<stub>>特别有助于管理大型系统模型。

第4讲: 静态建模

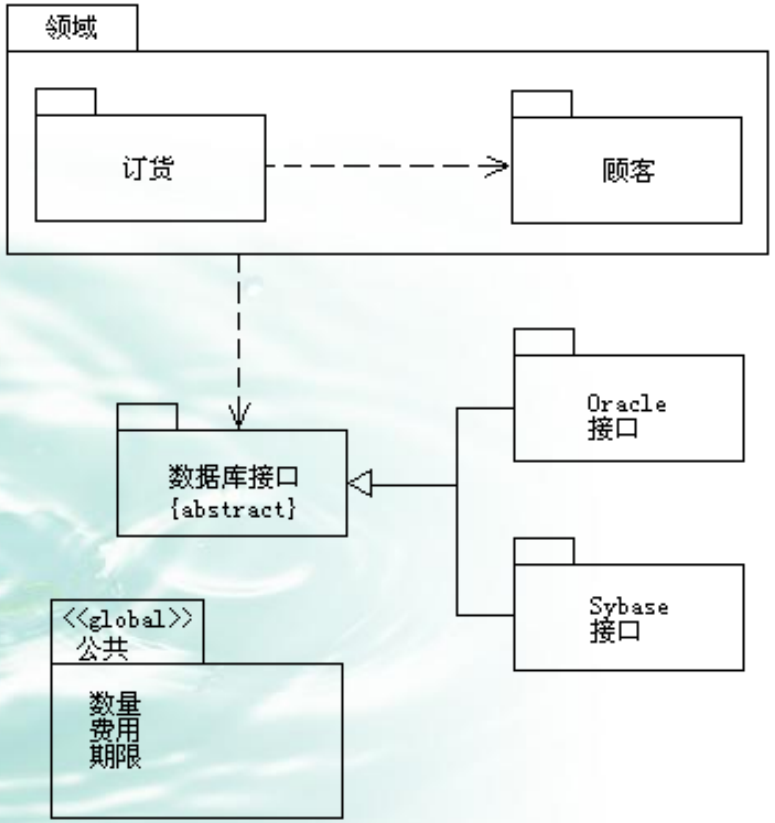
4.5 包图

- **包与包之间的联系主要有两种：依赖和泛化。**
  - 依赖：两个包所含模型元素之间存在着一个或多个依赖
  - ✓ 对于由类组成的包，如果在两个包的任何类之间存在着任何一种依赖，则这两个包之间存在着依赖。
  - ✓ 包的依赖联系同样是用一条虚箭线表示，虚箭线从依赖包（源）指向独立包（目标）。
  - ✓ 包的依赖联系没有传递性。

第4讲: 静态建模

4.5 包图

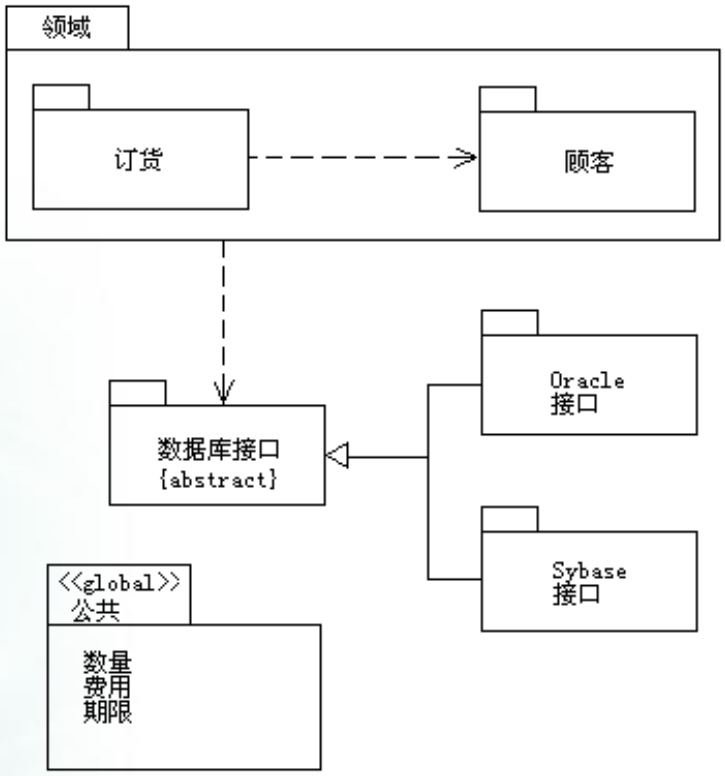
- **包与包之间的联系主要有两种：依赖和泛化。**
  - **依赖：两个包所含模型元素之间存在着一个或多个依赖。**
    - ✓ 示例：“订货”包与“顾客”包之间存在着依赖（嵌套、泛化）



第4讲: 静态建模

4.5 包图

- **包与包之间的联系主要有两种：依赖（尤其是输入依赖）和泛化。**
  - **泛化：特殊性包必须遵循一般性包的接口。**
    - ✓ 一般性包可加上一个性质说明“**{abstract}**”，表明它只不过是定义了一个接口，该接口可以由多个特殊包实现。
    - ✓ 与类的继承相同，特殊包从一般包继承其所含的公共类，并且可重载和添加自己的类。
    - ✓ 特殊包可代替一般包，用在一般包使用的任何地方。



第4讲: 静态建模

课后作业

- 2.2 (第29页)
- 2.5 (第30页)
- 8.2 (第138页)
- 8.3 (第138页)
- 8.5 (第138页)
- 8.9 (第140页)
- 8.17 (第141页)