

第04讲 UML静态建模

SUST
2022年3月

4.1 类图 and 对象图

- 如何对一个类建模
- 如何表现一个类的特性、职责和约束。
- 小结

- 类图描述系统中类的静态结构。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)。类图描述的是一种静态关系,在系统的整个生命周期都是有效的。
- 对象图是类图的实例,几乎使用与类图完全相同的标识。他们的不同点在于对象图显示类的多个对象实例,而不是实际的类。
- 一个对象图是类图的一个实例。由于对象存在生命周期,因此对象图只能在系统某一段时间段存在。

类图和对象图

- 类图给出了一般性的、定义性的信息：
一个类的特性以及它的属性，以及和这个类关联的其他的类。对象图则在某个特定时刻及时给出了一个类的多个具体实例以及它们如何联系起来等相关信息。

类的可视化表示

在UML中一个矩形表示一个类的图标



WashingMachine

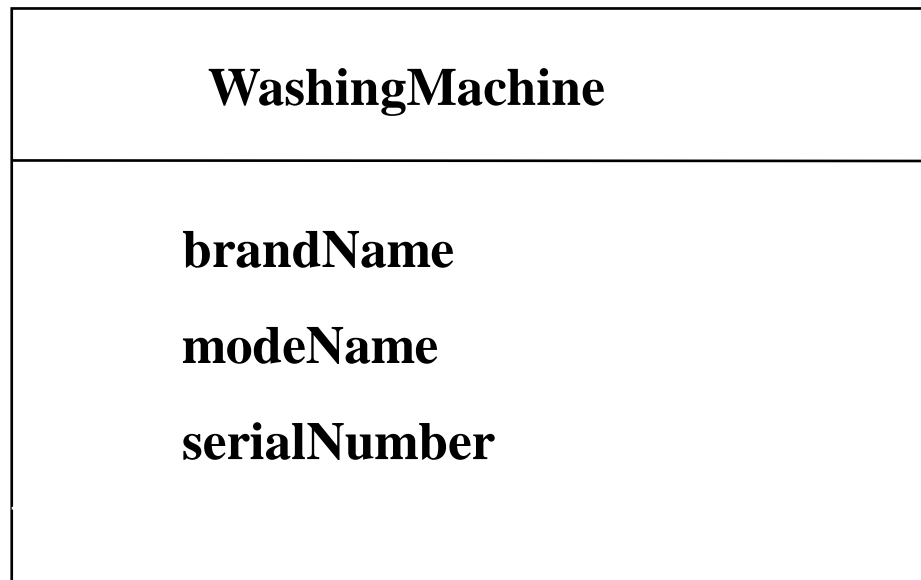
A rectangular box representing a UML class. The text 'WashingMachine' is written inside the box, with the first letter 'W' capitalized and the rest in lowercase.

注：按照UML的约定

- 1、把类名的首字母大写，放在矩形的偏上部。
- 2、如果类名是由两个单词组成，那么将两个单词合并，第二个单词首字母大写。

属性

属性是类的一个特性。它描述了类的对象（也就是类的实例）所具有的一系列特性值。一个类可以具有零个到多个属性。属性名列表放在类名之下，并且和类名之间用分隔号隔开，如图所示。

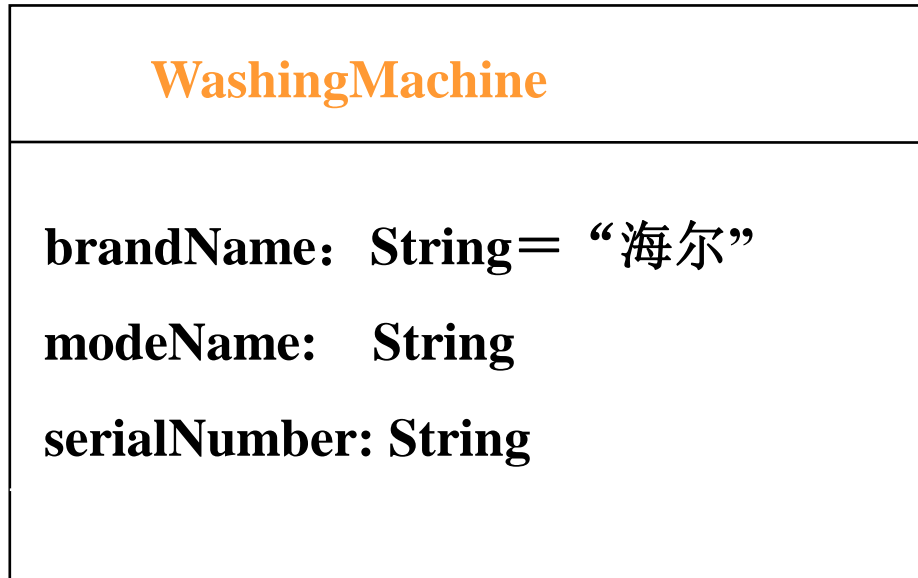


注：按照UML的约定

- 1、单个单词的属性名的小写。
- 2、如果属性名是由多个单词组成，那么将多个单词合并，除了第一个单词外，其它单词的首字母大写。

属性的附加信息

UML还允许指明属性的附加信息。要指明类型，则在属性值后面加上类型名，中间用冒号隔开。还可以为属性指定一个缺省值



对象

- 类的属性在该类的每个对象中都有具体值。下图是一个例子。注意，对象名首写字母小写，后面跟一个冒号，冒号后面是该对象所属的类名，并且整个名字要带下划线。

myWasher:WashingMachine

brandName= “海尔”

modelName= “小神童”

serialNumber= “GL0214”

命名对象或者不命名对象

名字myWasher: WashingMachine是一个命名实例(named instance)。也可以有诸如: WashingMachine这样的匿名实例(anonymous instance)。

操作

操作（operation）是类能够做的事情或者你（或者另一个类）能对类做的事情。操作名列表要放在属性名列表之下，两者之间用分隔线隔开，如图所示。

WashingMachine
brandName modeName serialNumber
addClothes() removeClothes() turnON()

注：按照UML的约定

1、单个单词的操作名的小写。

2、如果操作名是由多个单词组成，那么将多个单词合并，除了第一个单词外，其它单词的首字母大写。

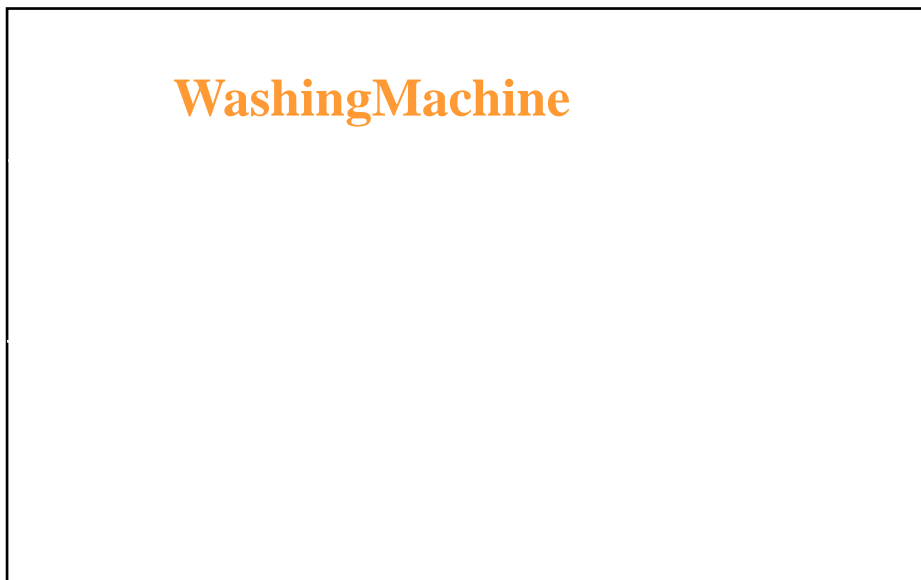
操作的附加信息

你也可以为操作指定附加信息。在操作名后面的括号中可以说明操作所需要的参数和参数类型。有一种操作叫函数（**function**），它在完成操作后要返回一个返回值。可以指明函数的返回值及返回值的类型。

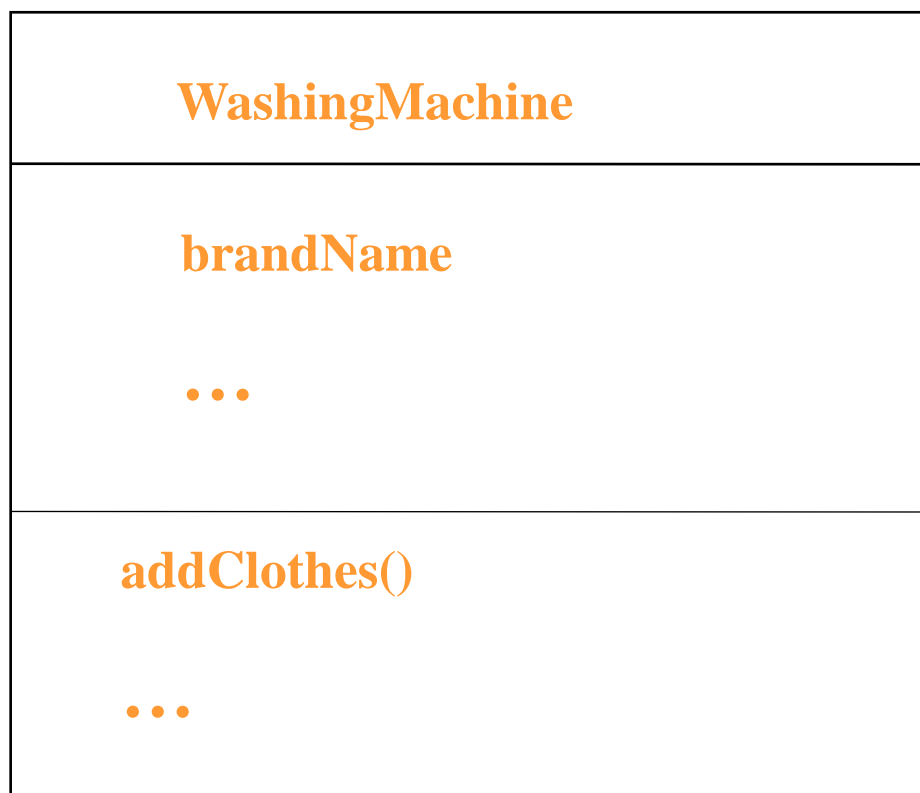
WashingMachine
brandName
modeName
serialNumber
addClothes(C: String)
removeClothes(C: string)
turnON(): Boolean

属性操作的可视化表示

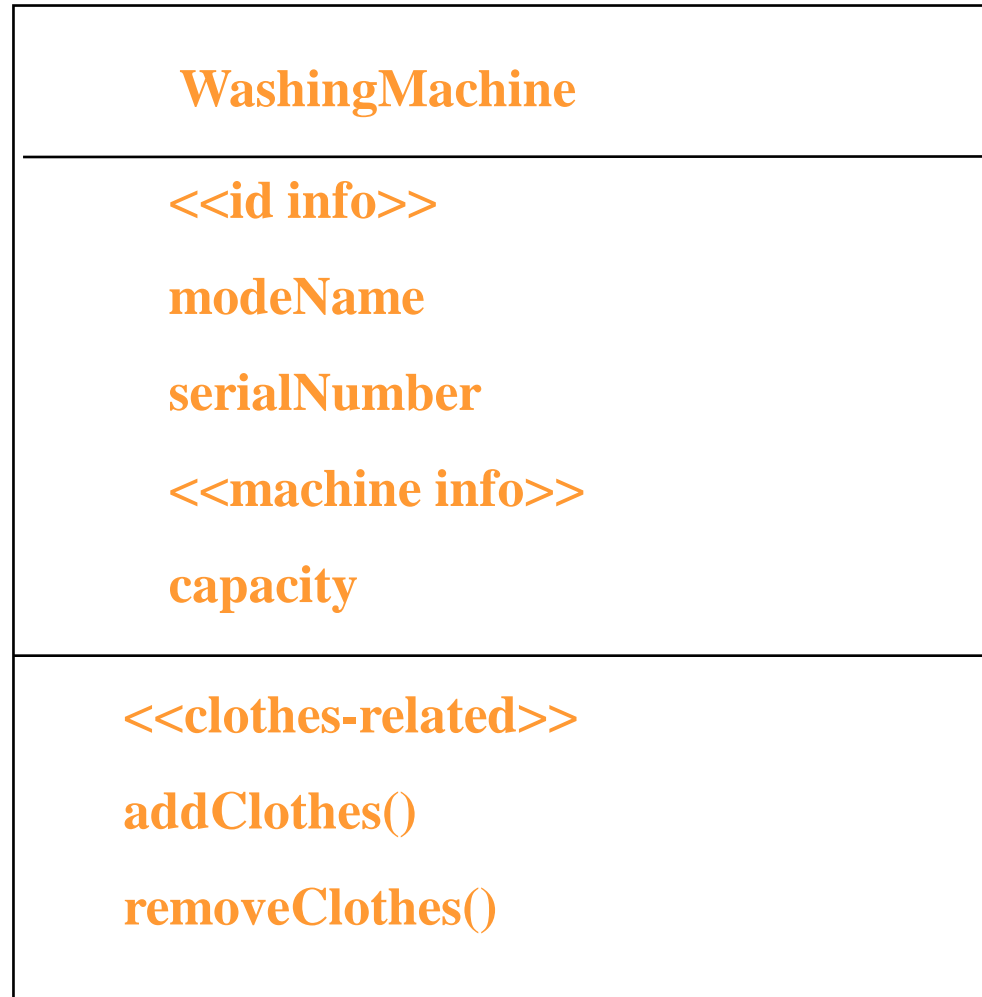
- 1、在实际中，可以只给出类名，而将属性或者操作区（或者两个全都）空着，如图所示：



2、有时可以只显示类的一部分属性和方法，这种省略了一个或多个属性或者操作的表示法叫做类的省略表示法（eliding a class）。图说明了类的省略表示法。

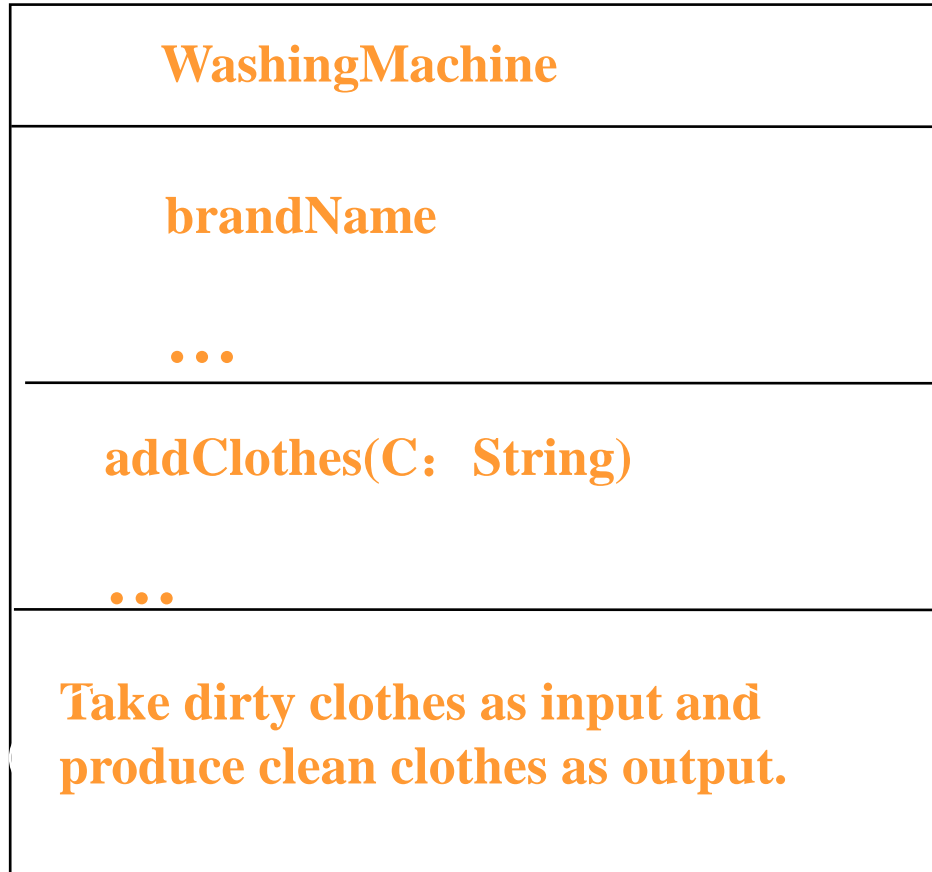


3、如果属性或者操作列表太长，可以用构造型来组织属性或操作列表，以便理解。构造型是UML提供的 扩展机制。



职责和约束

- 职责（**responsibility**）描述类做什么——也就是类的属性和操作能完成什么任务。在图标中，职责在操作区域下面的区域中说明。类的职责是消除二意性的一种非形式化的方法。



职责

更形式化的方式是使用约束（**constraint**），它是用花括号括起来的自由文本。括号中的文本指定了改类所要满足的一个或多个规则。如图所示：

WashingMachine
brandName capacity
addClothes() ...

{capacity=16 or 18 or 20}

约束

附加注释

我们还可以以对类附加注释的形式为类添加更多的信息。
下图中的注释说明了**serialNumber**（序列号）属性引用了政府标准。
记住，注释可以包含图形也可以包含文本。

WashingMachine
brandName modelName serialNumber
addClothes() removeClothes() turnON()

序列号的生成参考国际标准

小结

- 类代表的是领域知识中的词汇和术语。
- UML的类图标是由一个矩形表示。类名字、属性、操作和职责都在区域中有各自的方框。可以使用构造型来组织属性和操作名列表。可以使用类的省略表示法，只表示处类的一部分属性和操作。
- 可以在图标中指定属性的类型和初始值，还可以指明操作执行时所需要的参数和参数的类型。对一个操作来讲，这些附加信息被称为构型。
- 为了减少描述类时的二义性，可以对类施加约束。
- 类表达的是领域知识中的词汇。

5.2 类之间的关系

- 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。

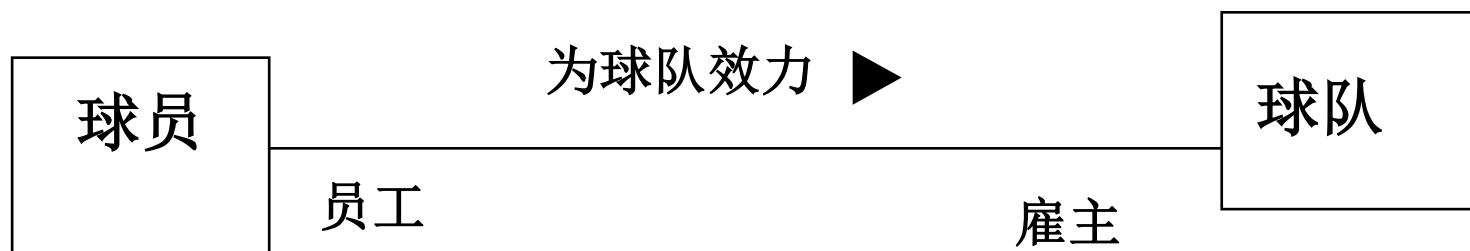
关联（association）

当类之间在概念上有连接关系时，类之间的连接叫做关联

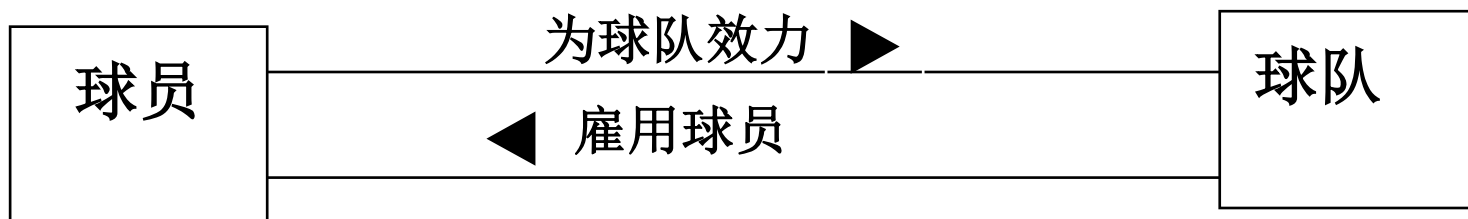
关联的可视化方法是用一条线连接两个类，并把关联的名字（例如“**plays on**”）放在这个连接线之上。表示出关联的方向是很有用的，关联的方向用一个实心三角形箭头来指明。下图说明如何可视化表示队员和球队之间的**plays on**关联。



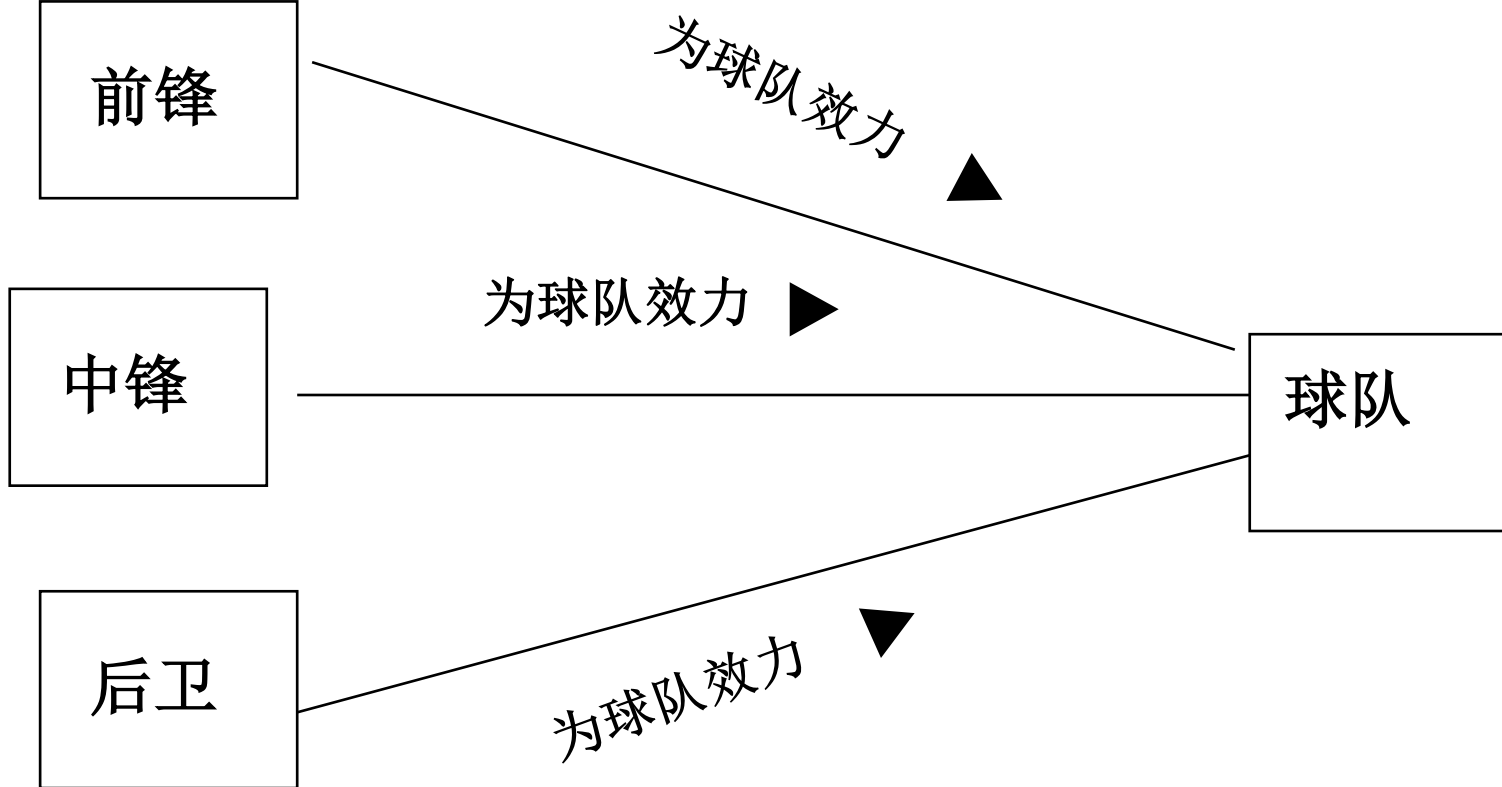
当一个类与另一个类发生关联时，每个类通常在关联中都扮演着某种角色。在队员和球队的关联中，如果球队是职业篮球队，那么它就是队员的雇主（employer）。下图说明了如何表示出这些角色。



- 关联还可以从另一个方向发生：篮球队雇佣（Employ）队员。可以把这两个方向上的关联表示在一个图中，用实心三角形箭头指明各自关联的方向，下图所示。



- 关联也可以好几个类连接同一个类。如图所示的关联图。

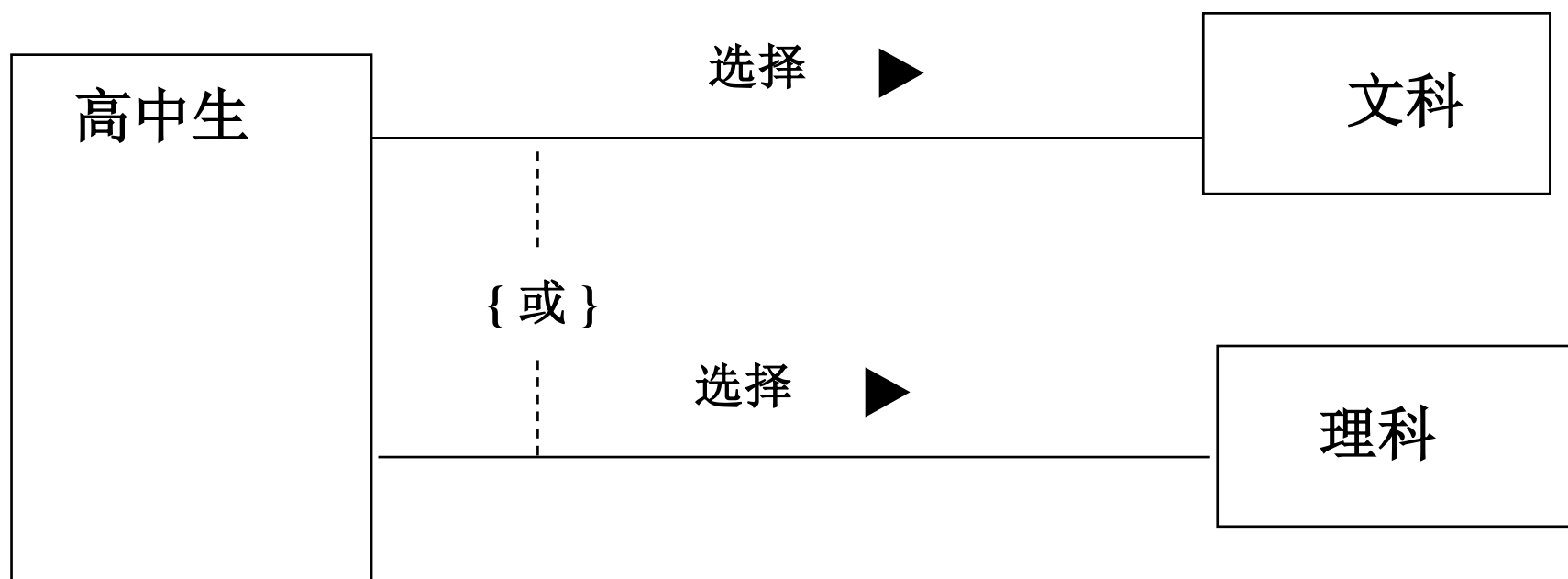


关联上的约束

有时两个类之间的一个关联随后就有一个规则。可以通过关联线附近加注一个约束来说明这个规则。如图所示。

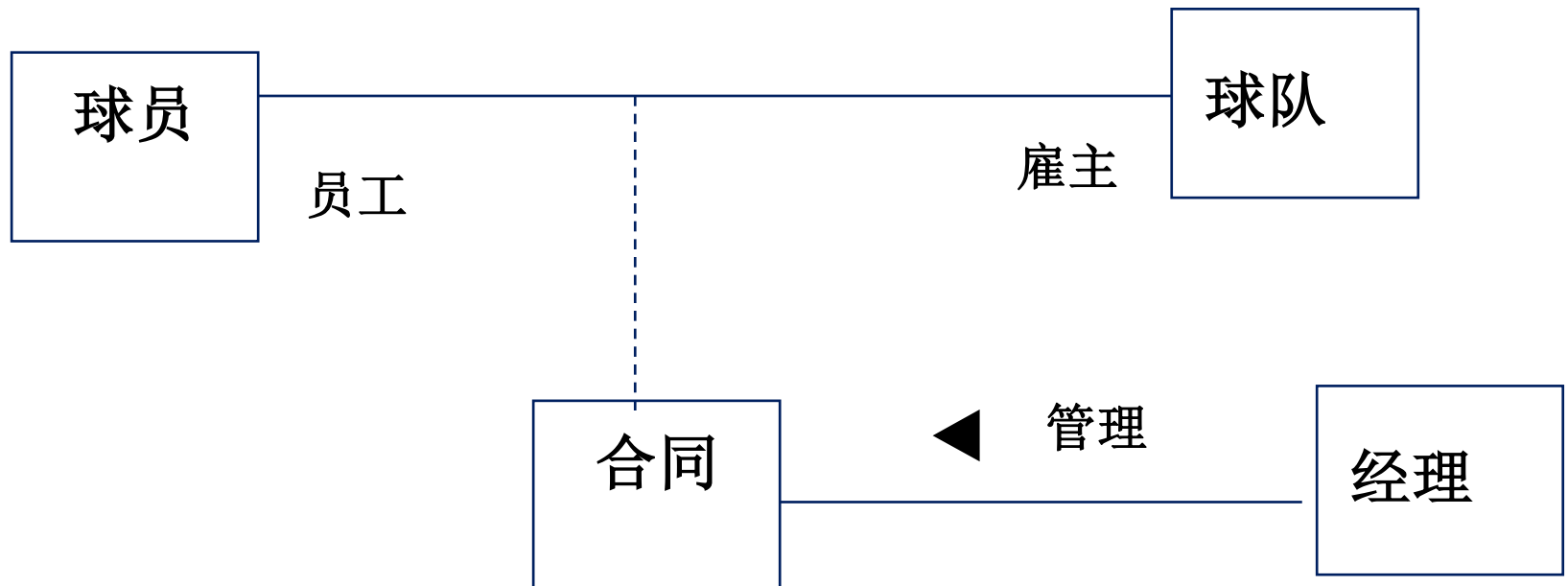


另一种类型的约束关系是Or(或)关系，通过两条关联线之间联一条虚线，虚线之上标注 {or} 来表示这种约束。图是高中生（high school student）选 (choose) 文科，或者选理科时的模型



关联类（association class）

和类一样，关联也可以有自己的属性和操作，此时。这个关联实际上是个关联类。图是球员类和球队类之间的关联对应的关联类：合同关联类。它同时又和经理类发生关联。



链（link）

正如对象是类的实例一样，关联也有自己的实例。如果我们想象要一个特定的队员效力一个特定的球队，那么两者之间的为球队效力关系就叫做一个链，可以用两个对象之间的连线来表示它。和对象的名字要加下划线一样链的名字也要加下划线，如图所示。

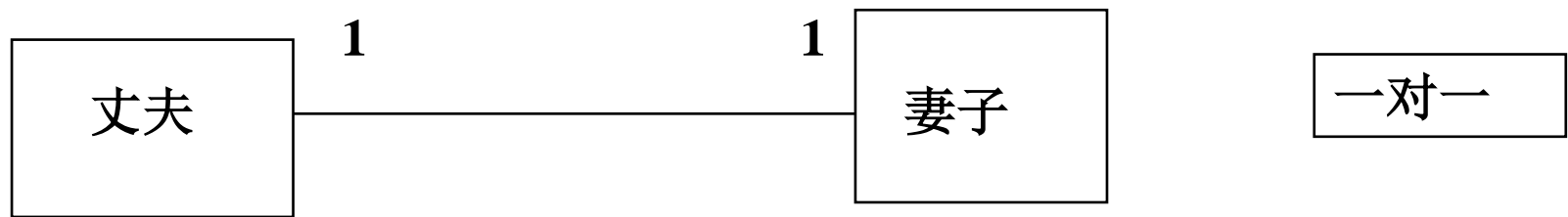


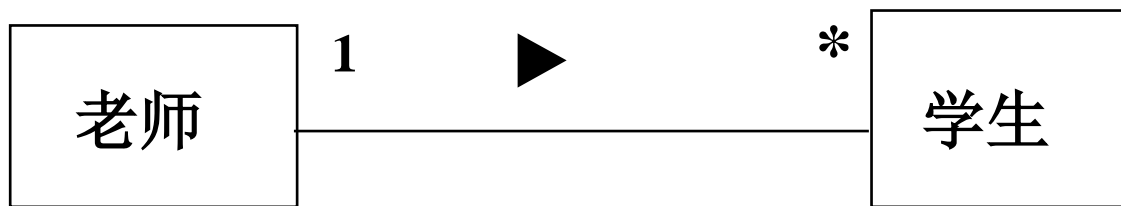
多重性（multiplicity）

- 某个类有多个对象可以和另一个类的单个对象关联。表示多重性方法是在参与关联的类附近的关联线上注明多重性数值，如图所示。

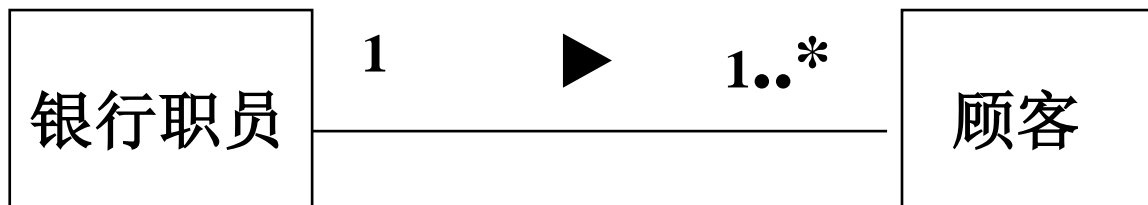


- 实际上存在各种可能的多重性。两个类之间可以是一对一、一对多、一对一或多、一对零或一、一对N或者一对一组选择。UML使用星号(*)来代表许多(more)和多个(many)。在一种语境中,两点代表Or(或)关系,例如“1..*”代表一个或多个。在另一种语境中,Or关系用逗号来表示,例如“5, 10”代表5或10。下图显示了各种可能的多重性表示方法。

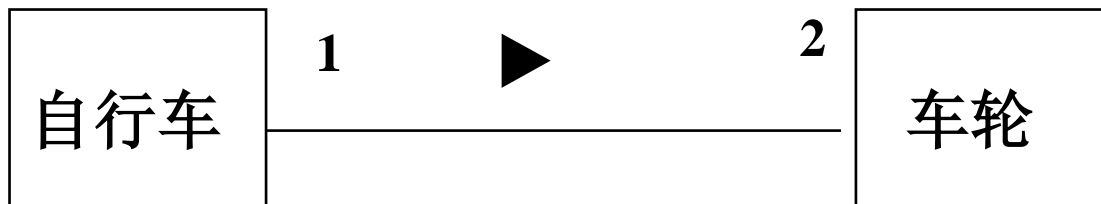




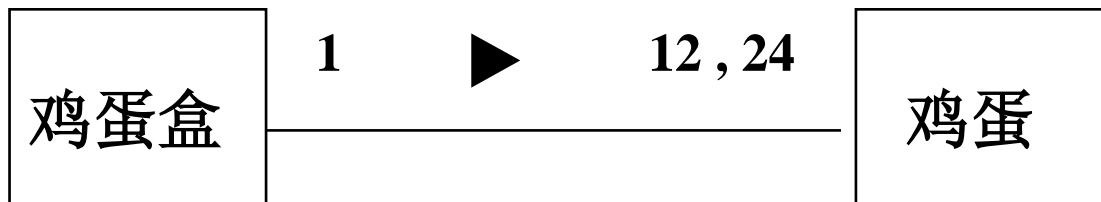
一对多



一对一或多



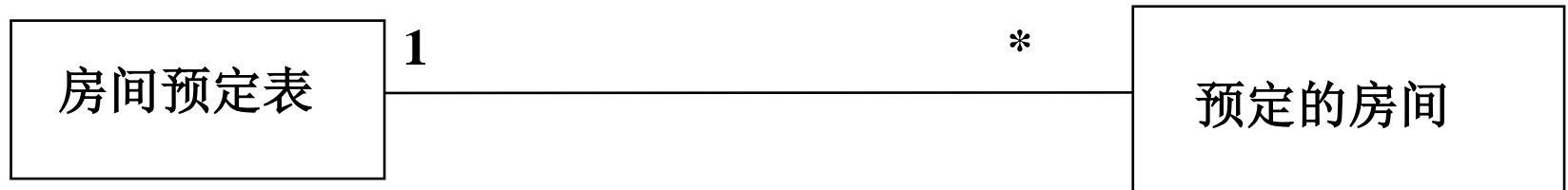
一对二

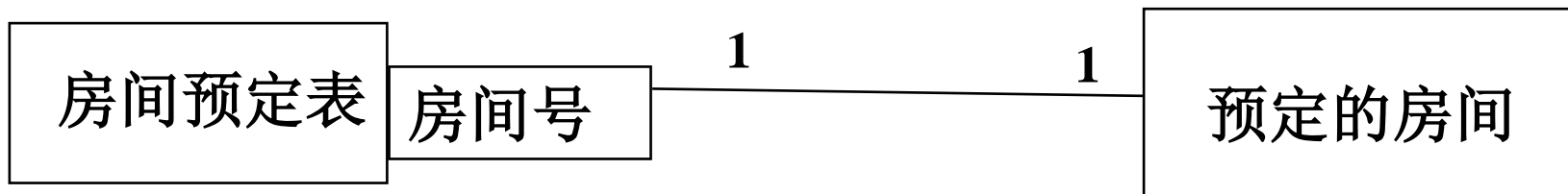


一对12或24

限定关联

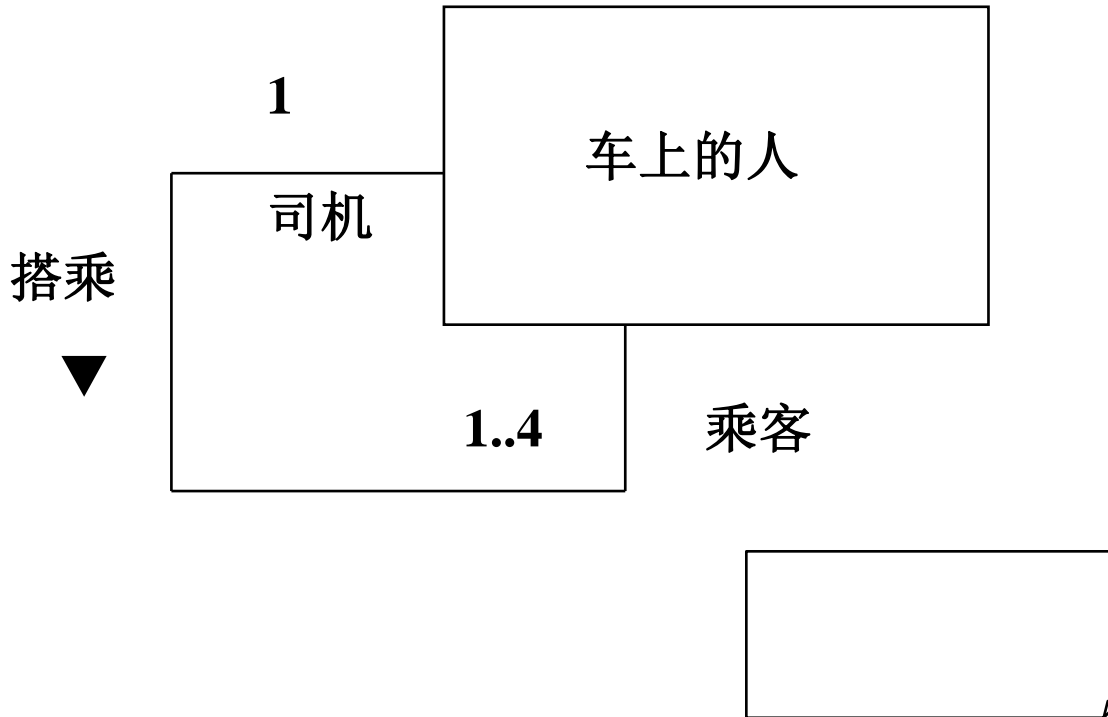
- 当关联的多重性是一对多时，就产生了一个特殊的问题：查找问题。当一个类的对象必须要选择规则中的另一个类的特定对象来满足关联中的角色时，第一个类必须依赖一个具体的属性值来找到正确的对象。这个属性值通常是一个标识符号，在UML中，**ID**（**identification**,标识）信息叫做限定符（**qualifier**）。它的符号是一个小矩形框，把作为一对多重性的一部分的类连在一起。图表示了这种关系。





自身关联（reflexive association）

- 有时，一个类可能与它自身发生关联，这样的关联被称为自身关联。当一个类的对象可以充当多种角色时，自身关联就可能发生。

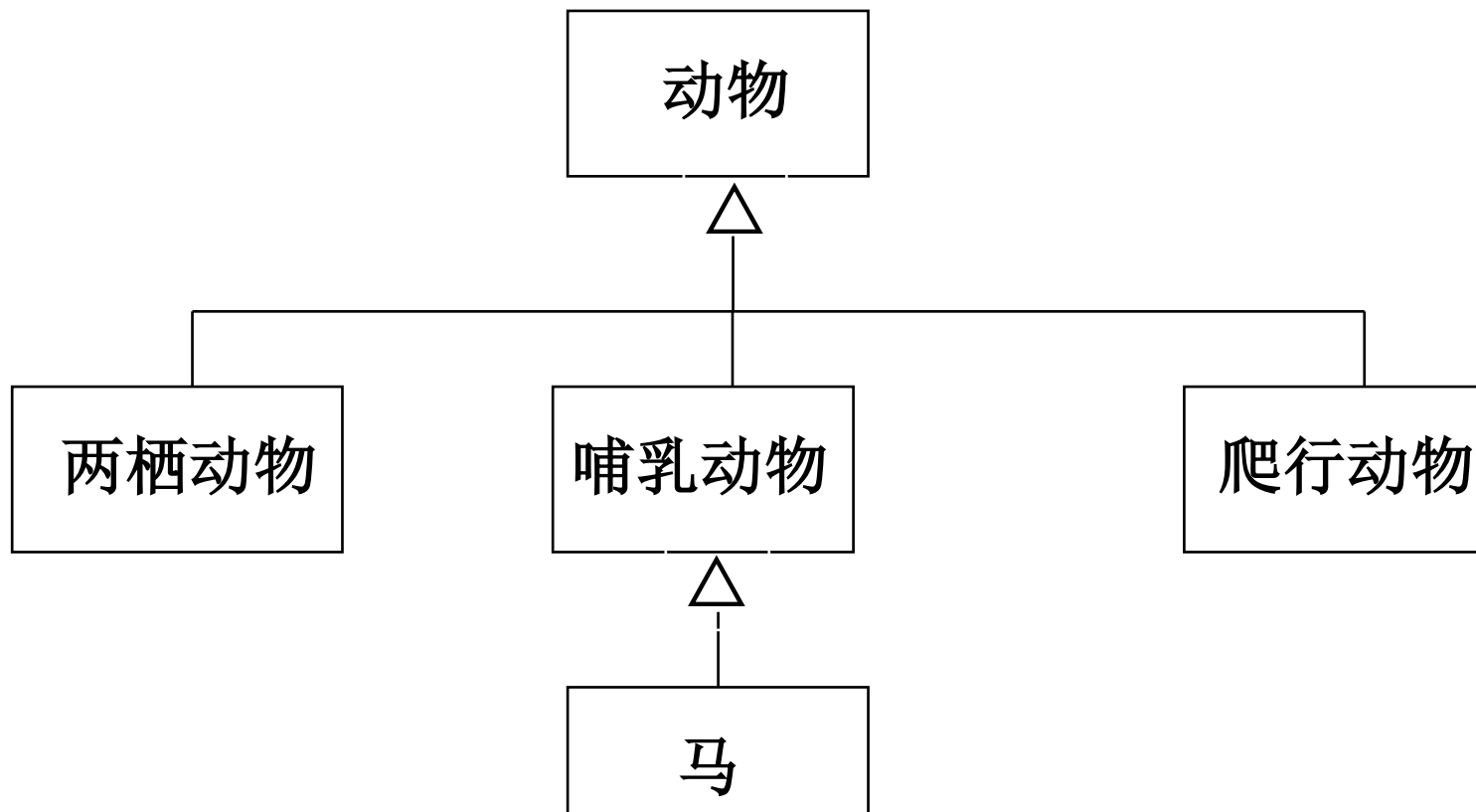


继承（inheritance）和泛化（generalization）

- 面向对象的一个特点是它能够反映日常生活中的常识：如果你知道某事务所属的种类，你就自然会知道同类的其它事物也具有该事物的一些特征。
- 在面向对象的术语中，上述的关系被称为继承,UML中也称它为泛化。一个类,可以继承另一个类的属性和操作。

继承层次并不止是两层：子类还可以是另一个子类的父类。

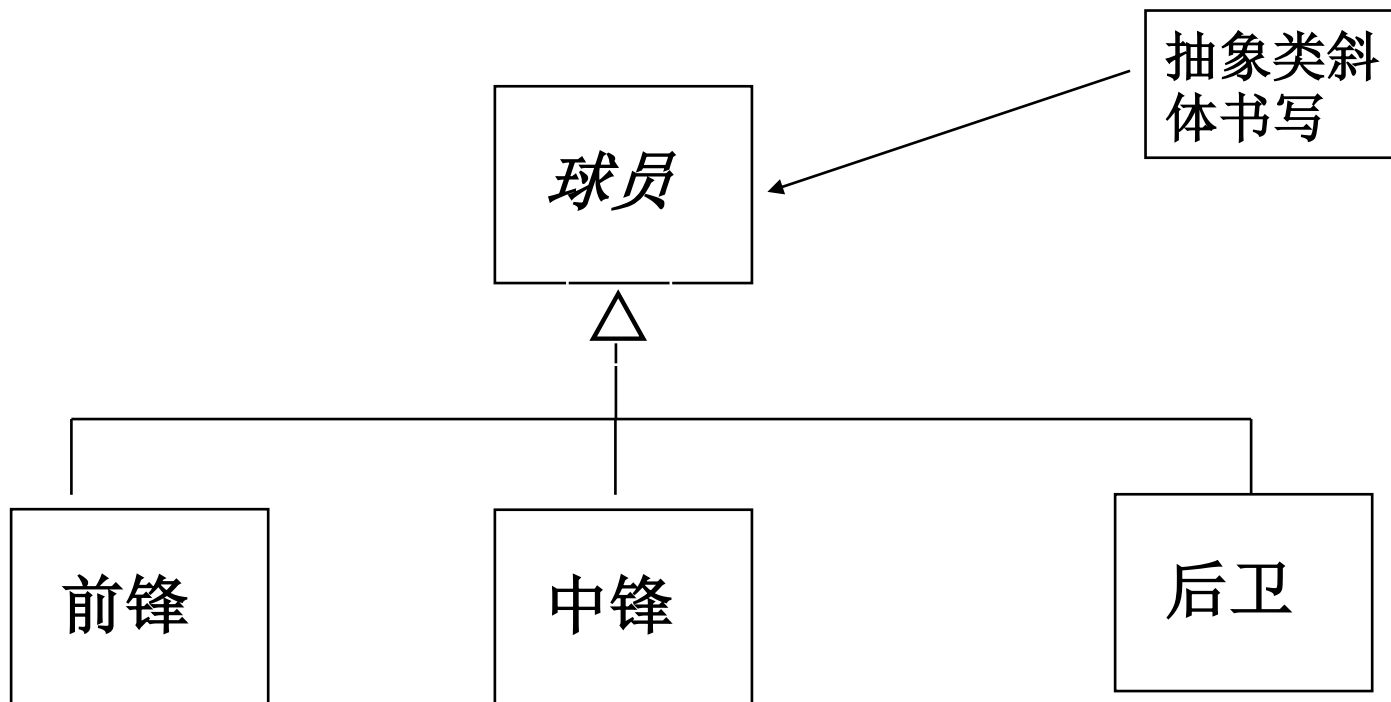
- 在UML中，用父类到子类之间的连线关系来表示继承关系。父类的连线部分，指向父类的一端带有一个空心三角形箭头。
- 注意：在父类中已经指明的属性和操作，在子类中可以不用再指明。
- 子类除了继承父类的属性和操作外，通常也增加了自己的属性和操作。



- 子类除了继承父类的属性和操作外，通常也增加了自己的属性和操作。
- 一个类可能没有父类，这种没有父类的类被称为基类（**base class**）或根类（**root class**）
- 一个类也可以没有子类，没有子类的类被称为叶类（**leaf class**）。如果一个类恰好只有一个父类，这样的继承被称为单继承（**single inheritance**），如果一个类有多个父类，这样的继承就是多继承（**multiple inheritance**）。

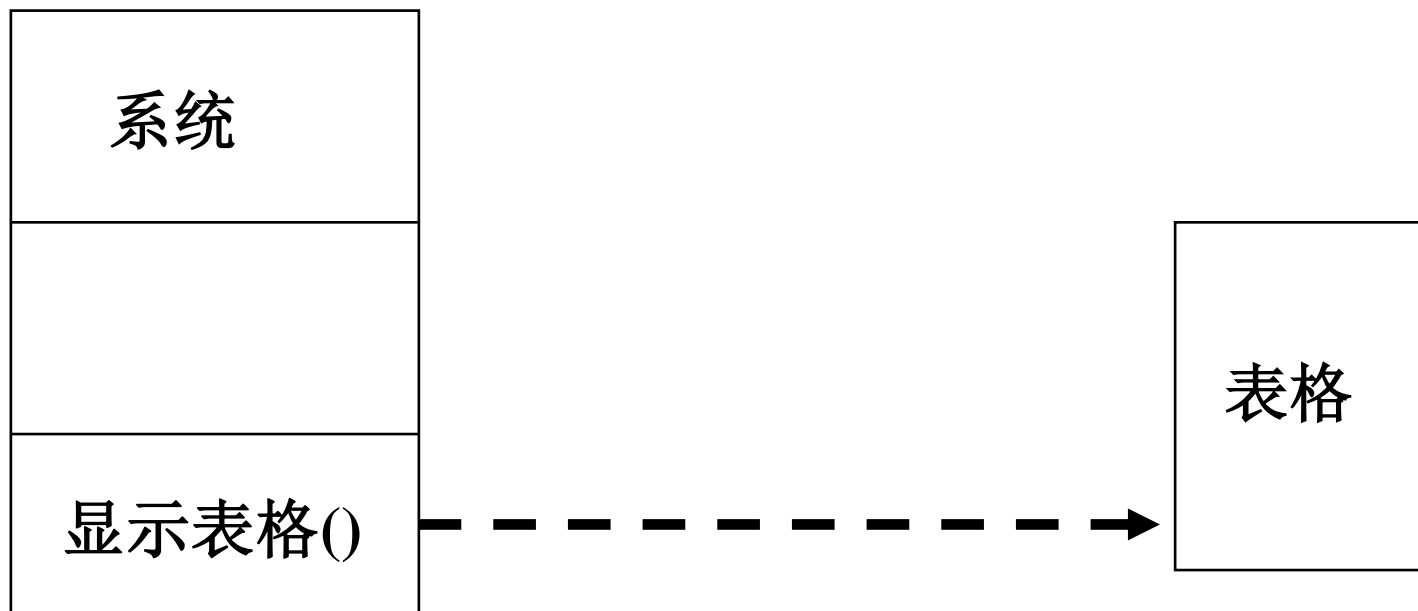
抽象类（abstract class）

- 像不提供实例对象的类被称为是抽象的（abstract）。
- 表明一个类是抽象类的方法是类名用斜体书写。下图示意了抽象类和它的子类的表示方法。



依赖（dependency）

- 另一种类间关系是一个类使用了另一个类，这种关系叫做依赖。最通常的依赖关系是一个类操作的构型中用到了另一个类的定义。
- 依赖关系用带箭头的虚线表示，如图所示。



小结

- 离开了类之间的关系，类模型仅仅只是一堆代表领域词汇的杂乱矩形方框。关系说明这些词汇表达的概念之间的连接，这样才能完整地说明我们所建模的对象。关联是类之间最基础的概念性连接。关联的多重性说明了一个类的多少个对象能够和另一个类的单个对象发生联系。和类一样关联也可以有自己的属性和操作。
- 一个类可以继承其他类的属性和操作。继承了属性和操作的了叫子类，被继承的类叫父类或超类。通过在初步类模型中寻找不同类的共同属性和操作可以发现类之间的继承关系。抽象类只是为了提供其他类继承的基类之用，它本身不产生对象实例。
- 在依赖关系中一个类使用了另一个类。
- 类图给出了类的一般性定义。在某一特定时刻，及时对类的具体实例建模则要使用对象图。

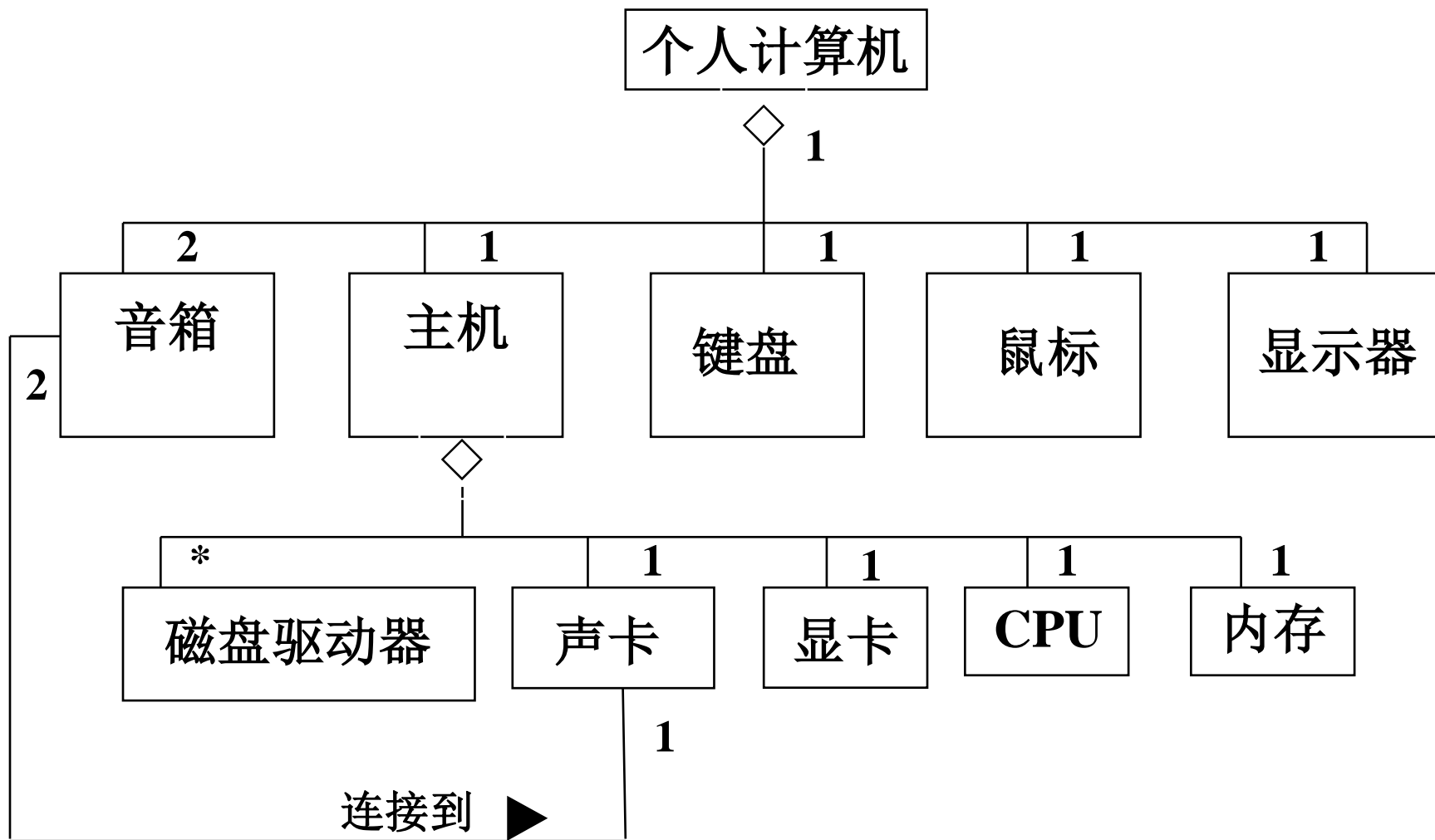
5.3 聚集、组成、接口和实现

- · 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（aggregation）

一个类有时由几个部分类组成的，这种特殊类型的关系被称为聚集。部分类和由它们组成的类之间是一种整体一部分（part-whole）关系。

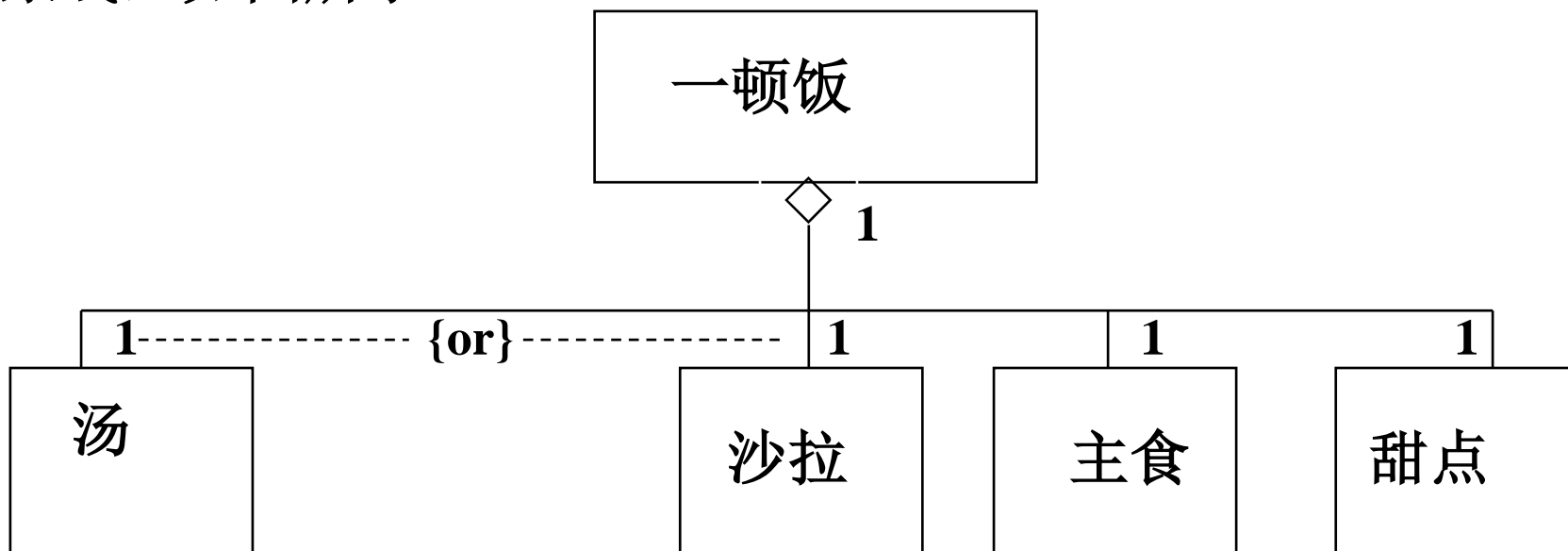
按照聚集关系的表示法，聚集关系构成了一个层次结构。“整体”类位于层次结构的最顶部，以下依次是各个“部分”类。整体和部分之间用带空心菱形箭头的连线连接，箭头指向整体。下图示意了家用计算机系统的组成。



- 在上个例子中的每个部分都属于一个整体，但聚集的关系并不是只有这种情况。
- 例如：在一个家庭影院系统中，电视机和影碟机可以共用一个遥控器，那么这个遥控器既是电视机的组成部分也是影碟机的组成部分。

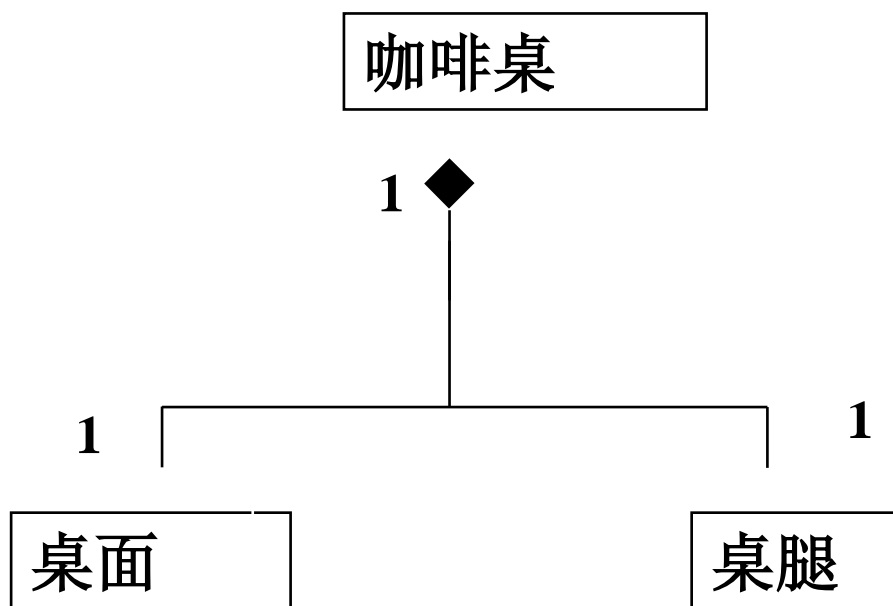
聚集上的约束

有时一个聚集体可能有多种部分体组成，这些部分体之间是“Or”关系。我们在两个整体一部分关系线之间加上一花括号括起来的“Or”来表示这个约束，并用虚线连接这两个关系线，如图所示。



组成

组成是强类型的聚集。聚集(组成)中的每个部分体只能属于一个整体。除了菱形箭头是实心之外，组成和聚集的表示法相同，如图所示。



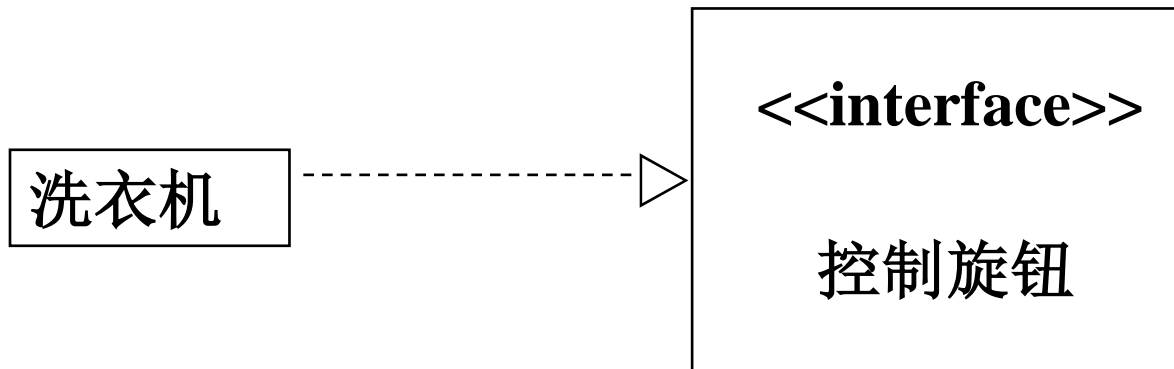
接口（interface）和实现（realization）

接口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。

一个类和它的接口之间的关系叫做实现。如洗衣机保证了它的部分行为能够“实现”控制柄的行为。

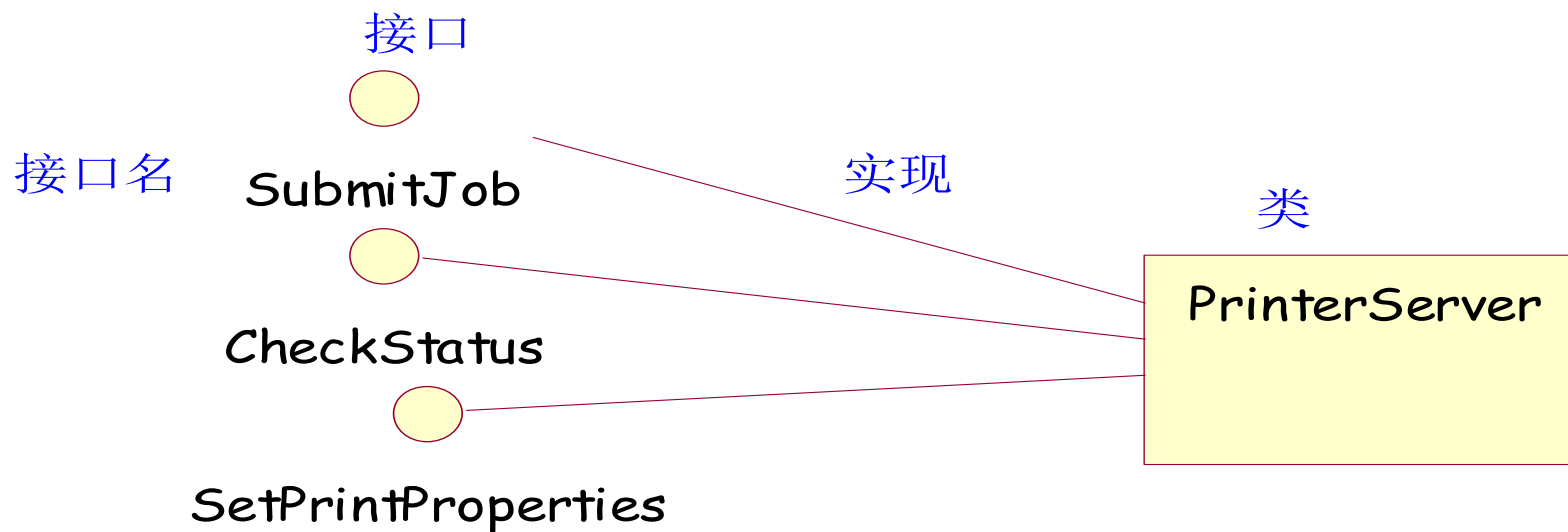
接口的模型表示法和类大致相同，都是用一个矩形图标来代表。和类的不同之处在于，**接口只是一组操作，没有属性**。如果一个类表示了省略了属性，那么怎么区分类和接口呢？一种办法是使用构造型《interface》，把它放在矩形框中接口的名字之上。另一个办法是接口的名字以大写字母“I”开头。

表示类和接口之间的实现关系的符号和继承关系的符号有些相似，只不过它是一个带空心三角形的箭头，箭头的方向指向接口。下图示意了洗衣机和控制旋钮之间的实现关系。



- 另一种表示法（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，如图所示。这种图有时候形象地被称为棒棒糖图（lollipop diagram）。





上图表示了打印机中的用例关系。

- 一个类可以实现多个接口，一个接口也可以被多个类实现。
- 由于我们要依靠接口实现洗衣机的操作，我们把通过接口的交互建模为一种依赖。下图示意了这种关系。

注意：图中使用依赖符号，对通过接口和类的交互来建模。



- 实现关系将一种模型元素（如类）与另一种模型元素（如接口）连接起来，其中接口只是行为的说明而不是结构或者实现。客户必须至少支持提供者的所有操作（通过继承或者直接声明）。虽然实现关系意味着要有像接口这样的说明元素，它也可以用一个具体的实现元素来暗示它的说明（而不是它的实现）。
- 泛化和实现关系都可以将一般描述与具体描述联系起来。泛化将在同一语义层上的元素连接起来（如，在同一抽象层），并且通常在同一模型内。实现关系将在不同语义层内的元素连接起来（如，一个分析类和一个设计类；一个接口与一个类），并且通常建立在不同的模型内。在不同发展阶段可能有两个或更多的类等级，这些类等级的元素通过实现关系联系起来。两个等级无需具有相同的形式，因为实现的类可能具有实现依赖关系，而这种依赖关系与具体类是不相关的。
- 实现关系用一条带封闭空箭头的虚线来表示，且与泛化的符号很相像。
- 用一种特殊的折叠符号来表示接口（无内容）以及实现接口的类或构件。
 - 。接口用一个圆圈表示，它通过实线附在表示类元的矩形上。

接口和端口（port）

- 鼠标是如何连接到计算机的？沿着鼠标后面的电缆，在计算机的后面，你会看到一个端口，也就是鼠标接入的地方。当然你的计算机也可能由一系列端口，包括一个并行端口以及一个或多个**USB**端口。计算机正是通过这些端口和外界的环境交互。
- **UML2.0**提供了一个符号用来对这些交互点建模。如图所示，端口符号是位于类符号边缘上的一个小方格，这个小方格连接到接口。



可见性（visibility）

- 可见性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。可见性由3种层次（级别）。
- 1、在公有（**Public**）层次上其他类可以直接访问这个层次中的属性和操作。
- 2、在受保护(**Protected**)层次上，只有继承了这些属性和操作的子类可以访问最初类的属性和操作。
- 3、在私有(**Private**)层次上，只有最初的类才能访问这些属性和操作。
- 实现关系意味着接口中的所有操作都是公有的。下图示意了前面提到的电视机类和汽车类中的公有、受保护的和私有操作。

Television

+brandName

+modelName

...

+changeVolume()

+changeChannel()

-paintImageOnScreen()

...

Automobile

+brandName

+modelName

...

+accelerate()

+brake()

#updateMileageCount()

...

说明：属性或操作前面有“+”号，该属性或操作是公有的

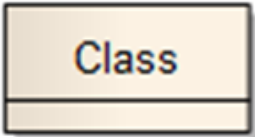

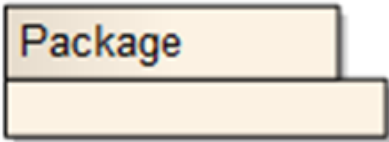

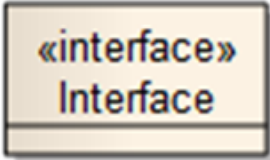



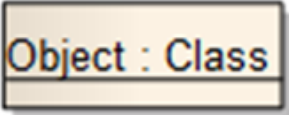


属性或操作前面有“#”号，该属性或操作是受保护的

属性或操作前面有“-”号，该属性或操作是私有的

小结

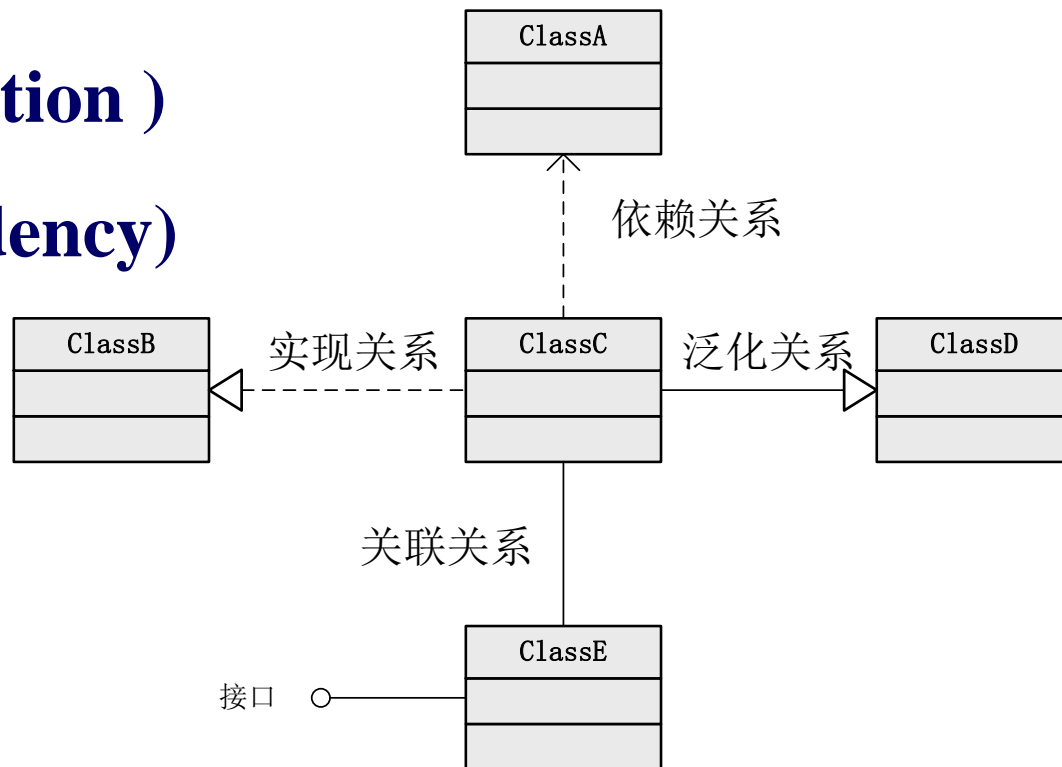
- 聚集是一个整体部分关联：“整体”类是由“部分”类组成的。组成是一种强类型的聚集，因为组成体的部分体只能属于一个整体。聚集关系的菱形箭头是空心的，而组成关系的菱形箭头是实心的。
- 组成结构图通过展示嵌入在一个类中的那些类，使得该类的内部结构变得可见。
- 实现是类和接口之间的一个关联，接口是可供其他类使用的一个操作集。接口用没有属性的类表示。实现关系用一条虚线连接类和接口，虚线靠近接口的一端带有一个空心三角形箭头指向接口。另一种表示实现的方法是用一条直线连接小圆圈，小圆圈表示接口。
- UML2.0增加了一个符号来表示端口。类通过端口和它的环境交互。这个符号是一个位于类符号边缘上的小方格，他和接口相连。
- 在可见性术语中，接口中的所有操作都是公有的，以使任何类都可以访问接口中的操作。

小结

结构元素	图形符号	关系元素	图形符号
类 (Class)		依赖 (Dependency)	
包 (Package)		关联 (Association)	
接口 (Interface)	 或  Interface	聚合 (Aggregation)	
		组合 (Composition)	
对象 (Object)		泛化 (Generalization)	
		实现 (Realization)	

小结

- 1. 关联(Association)
- 2. 泛化(Generalization)
- 3. 实现(Realization)
- 4. 依赖(Dependency)



小结

UML 符号	描述	Java访问控制符
+	表示具有公共可见性，可以被所有的类访问和使用	public
#	表示受保护的可见性，经它修饰的属性和方法可以被同一个包中的其他类、不同包中该类的子类以及该类自身访问和引用	protected
~	表示包级可见性，只能被同一个包中的其他类访问或引用，不在同一个包中的类不能访问它	default
-	表示私有可见性，经它修饰的属性和方法只能被该类自身所访问，它对属性和方法提供了最高级别的保护	private

类图实践

