Software Modeling with UML2:
Concepts, Sepecifications and Methods

UML2
软件建模：概念、规范与方法

严悍 刘冬梅 赵学龙 编著

国防工业出版社
National Defense Industry Press

UML2 软件建模：概念、规范与方法

严悍　刘冬梅　赵学龙　编著

国防工业出版社
·北京·

内容简介

规范化、可视化的软件建模已成为当今软件技术的主流之一。本书依据统一建模语言UML最新版本的规范，与面向对象编程语言相对应，结合实际工程，深入全面地探讨软件建模的新概念、新规范和新方法。本书共有四个部分。第一部分是概述，简要介绍了面向对象特征、建模的概念以及UML2，作为第1章。第二部分是逻辑结构建模，包括第2章到第5章，探讨了用例、类与接口、关系建模以及其他结构建模。第三部分是行为建模，包括第6章到第8章，分别探讨了交互、状态机和活动。第四部分是体系结构建模，包括第9章和第10章，从结构建模的角度探讨了构件、制品、结点与部署。各章后配有小结和适量的练习题，以方便读者及时总结和提高。

本书可作为计算机相关专业的本科生的教学用书和研究生的参考教材，也可作为软件工程开发维护人员的自学用书和研究人员的参考用书。

图书在版编目(CIP)数据

UML2软件建模：概念、规范与方法/严悍，刘冬梅，赵学龙编著. —北京：国防工业出版社，2009.2
ISBN 978-7-118-06145-1

Ⅰ.U... Ⅱ.①严...②刘...③赵... Ⅲ.面向对象语言，UML－程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字(2009)第004677号

※

国防工业出版社出版发行
(北京市海淀区紫竹院南路23号 邮政编码100048)
腾飞印务有限公司印刷
新华书店经售

*

开本 787×1092 1/16 **印张** 15¾ **字数** 378千字
2009年2月第1版第1次印刷 **印数** 1—4000册 **定价** 32.00元

(本书如有印装错误，我社负责调换)

国防书店：(010)68428422 发行邮购：(010)68414474
发行传真：(010)68411535 发行业务：(010)68472764

前　言

软件日益复杂化是当今软件领域不可回避的问题。建模就是为了更好地理解系统的复杂性。规范化、可视化的软件建模已成为当今软件开发维护的主流技术之一。1997 年 OMG 推出统一建模语言 UML，并逐渐成为软件建模的工业规范。2003 年推出新版本 UML2，添加了许多新概念新规范，之后每年更新，逐步完善。本书依据统一建模语言 UML 最新版本规范，与面向对象编程语言相对应，结合实际工程，深入全面地探讨软件建模的新概念、新规范和新方法。

本书特色在于新规范、探索性和实用性。本书依据统一建模语言 UML 最新版本 2.1.2 规范，探讨新概念、新规范和新方法，并与以前的概念和规范进行比较。本书深入探讨了建模语言与面向对象编程语言(如 C++ 和 Java)之间的关系，分析了两者在语法、语义、语用等方面的区别和映射。几乎每一章都会探讨 UML 概念是否能映射到编程实现以及如何映射。这种探索有助于读者将模型与软件相融合，使建模成为软件工程中的一个有机组成部分。本书注重实用性，采用了大量的实际工程案例，而且将一些设计原则引入到建模方法。例如在关系建模中引入了面向对象设计的五项基本原则，在包的建模中探讨了内聚性原则和耦合性原则。这些内容对于实际软件工程具有实用价值。

本书内容分为四个部分。第一部分是概述，结合编程语言介绍了面向对象特征、模型概念以及 UML2 的各种图，作为第 1 章内容。第二部分是逻辑结构建模，包括第 2 章到第 5 章，探讨了用例、类与接口、关系建模、其他结构建模。其中第 2 章用例比较特殊。虽然用例应归入行为建模的范畴，但在实际软件工程中用例建模往往首先用于需求建模，故此本书将其作为最重要内容放在前面介绍，以方便读者按实际软件工程的过程进行理解和掌握。第三部分是行为建模，包括第 6 章到第 8 章，分别探讨了交互、状态机和活动，这三个部分相对独立。第四部分是体系结构建模，包括第 9 章和第 10 章，从结构建模的角度探讨了构件、制品、节点与部署。各章后配有小结和适量的练习题，以方便读者及时总结和提高。

为了更好地使用本书，对读者有以下几个建议：

(1) 深刻理解建模的目的。图形是人类思考的工具，模型是人们理解复杂性的手段，建模作为一种针对复杂系统的设计技术而被广泛采纳。软件建模的目的是规范化设计、可视化表达、构建与存档。

(2) 结合面向对象编程语言。本书中大量探讨了 C++ 和 Java 编程语言与 UML 建模语言的元素之间的关系。建议读者能同时学习实践 C++ 或者 Java 编程语言。

(3) 结合实际工程，反复实践。软件建模的概念结构庞大、规范复杂多变，同时实践性很

强。不仅要求熟练掌握重要的概念和规范，也要求大量的、反复的建模实践。

模型驱动架构(MDA，Model－Driven Architecture)是未来软件技术发展的一个方向，建模将取代大部分代码编程。基于UML规范的软件建模技术将扮演更加重要的角色。

本书不足之处，竭诚希望广大读者指正。

编　者

2008年11月

目　录

第1章 概 述

基于 UML 的建模技术是面向对象的重要内容，本章简单介绍对象和类的基本概念、面向对象的基本特征，再探讨模型和建模的概念，最后介绍 UML2 的各种图和相关工具。

1.1 一个简单例子

假设描述奥运会及主办城市，可用图 1.1 来描述。

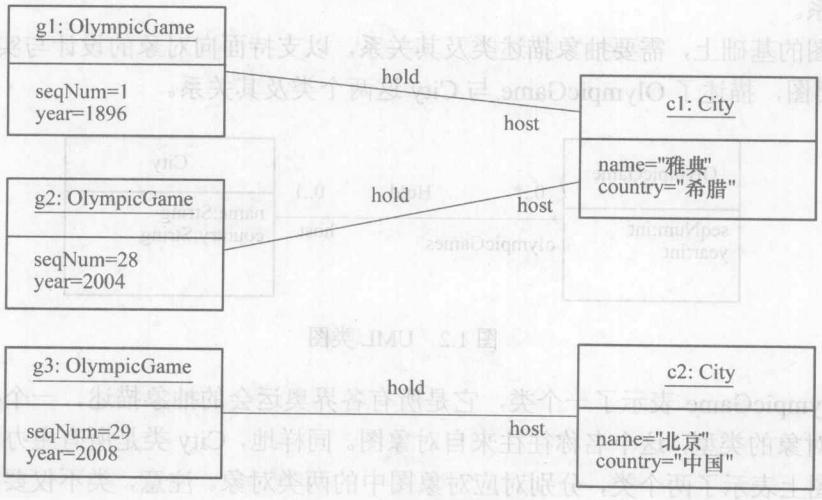

图 1.1 对象图例

每一届奥运会都作为一个对象，图中描述了第 1 届、第 28 届和第 29 届。图中取“OlympicGame”作为所有奥运会对象的类型名称，用冒号和下划线表示它是一个对象，冒号之前可确定对象的名称。届数用一个“seqNum”值来表示，举办年用“year”值来表示。每一个对象都用一个方框围起来，在上端间隔中说明该对象所属的种类，在下端说明各个性质的名称及对应的值。所以图中清楚地描述了 3 届奥运会的情况。

到 29 届为止，每一届奥运会都是由一个城市主办的。图中将每一个城市作为一个对象，取“City”作为所有城市对象的类型名称。每一个城市对象都用一个“name”值来表示其名字，用一个“country”值来表示该城市所在的国家。

当描述一届奥运会与一个主办城市之间的联系时，对象图要用一个实线段来连接两个对象，称为“链 link”，并用标注文字来说明链的细节。图中用“hold”来描述链的名称，图中 3 个链都有同样名称。在靠近“City”对象的一端用“host”来表示东道主。

图 1.1 采用了统一建模语言 UML(Unified Modeling Language)来描述客观事实，也可用其他方式来表示，用表 1.1 来描述同样的事实。

表 1.1　历届奥运会主办城市

seqNum	year	hostCityName	hostCountry
1	1896	雅典	希腊
…	…	…	…
28	2004	雅典	希腊
29	2008	北京	中国

表 1.1 中的每一行对应一届奥运会，每行用 4 列来表示每届奥运会的相关性质。我们经常在各种文献和媒体上见到类似的表结构。这种结构被称为“关系”。对象图与关系表都能表示相同的内容，但两者之间具有很大不同，最明显的差别在于，对象图是基于识别对象、描述对象的类型、性质以及对象间的关系，而关系表是基于行与列的形式来描述实体及其属性、以及实体间的联系。

在对象图的基础上，需要抽象描述类及其关系，以支持面向对象的设计与实现。图 1.2 所示为 UML 类图，描述了 OlympicGame 与 City 这两个类及其关系。

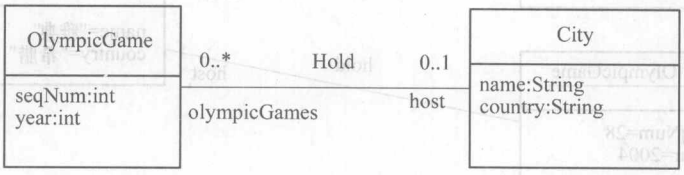

图 1.2　UML 类图

图中 OlympicGame 表示了一个类，它是所有各界奥运会的抽象描述。一个类需要一个名称来标识其对象的类型，这个名称往往来自对象图。同样地，City 类是所有主办城市对象的抽象描述。类图上表示了两个类，分别对应对象图中的两类对象。注意，类不仅要抽象概括已出现的对象，而且要描述将来出现的对象。

每个类除了要描述类型名称之外，还可描述各个性质及类之间的关系。OlympicGame 类描述了两个性质。每个性质都可描述了名称及类型，但不同于对象描述，类中不描述各性质的值，这是因为只有在对象中，各个性质才具有具体的值。采用简单的整数 int 来确定 seqNum 和 year 性质值的类型。同样地，City 类中描述了两个性质，采用字符串 String 来描述这两个性质的值的类型。

图中还描述了两个类之间的一个关系，这是一个关联关系，用实线连接两个类，并命名为 hold，表示奥运会与主办城市之间的一个关联。同时，在 City 一端用 0..1 来表示每一届奥运会最多只有一个主办城市，换言之，一个 OlympicGame 对象对应 0 个或 1 个 City 对象。这是根据对象图描述得到的抽象。

同样，在 OlympicGame 类一端用 0..*表示一个城市可主办多届奥运会，也就是说，一个 City 对象对应 0 个或多个 OlympicGame 对象。由对象图可知，同一个城市可能多次主办不同届的奥运会，如雅典已主办了第 1 届和第 28 届奥运会。

这种定量的约束被称为多重性。对每个关联都要识别并确定关联两端的多重性。

对于一个关联，除了要描述名称和多重性之外，往往需要描述关联两端的角色，来说明相关联的对象各自所起到的作用。图中 hold 关联在 City 一端，用 host 来说明每一届奥运会的主办城市作为东道主。同样地，在 OlympicGame 一端用 olympicGames 来说明东道主所主办的多届奥运会。

对于每个类，不仅可描述类的名称、各个性质，还可描述该类的各个操作，以表示该类所有对象的行为。尽管上面图中没有描述，但我们起码知道一个类的所有对象都具有相同的操作。

以上通过一个对象图和一个类图说明了面向对象建模的一个简单例子。可以知道，模型能反映客观世界的基本规律，也能描述将要设计实现的软件，所以建立模型对于分析和设计具有重要意义。

同时也注意到，图中并未表示各个类的所有信息。例如，每个主办城市可能还要描述人口数量、体育场馆的种类及数量等。一个模型中的一张图只能对系统中某一部分特征在特定抽象层面上进行描述，而不可能既详细又全面地反映所有特征。模型不同于程序代码，单个模型的单个图都是抽象的、片面的、有特定意图的。

因此，每一张图都需要一个准确的命名来表示其意图。对类图 1.2 可命名为“奥运会及主办城市间的关联”。

最后，应注意的是类的名称、性质的名称以及关联的名称都采用英文表示，而且类和关联的名称的首字母应大写。

1.2 对象和类

对象和类是面向对象方法学中的基本概念。

1.2.1 对象的概念

上面例子中，把每座城市都作为一个对象，建立了 City 类来抽象所有城市对象的共性。那么对象究竟是什么概念？一个对象是：

(1) 具有明确定义的边界和唯一标识(identity)的一个实体(entity)。

(2) 状态和行为的一个封装体。

(3) 类的一个实例(instance)。

首先，识别一个对象必须要能区别其他对象，主要是区别一个对象的边界和标识。一座城市作为一个对象，通常有自己的特定名称，这个标识能够明确区别于其他城市。同样每一座城市相对于其他城市，具有明确的地理边界，而不至于混淆。计算机软件在运行时刻，每个对象都有自己的标识，称为 OID(Object Identity)，是对象式语言系统赋予的内部标识。

其次，一个对象包含了属于自己的一些特定状态描述。对象的状态可用性质来表示。一个性质(property)是一个“名=值”的对偶。习惯上，我们将对象内的性质称为属性。如 City 类中的 name。值得注意的是，一个对象所持有的“链”也是该对象的性质。如图 1.1 所示，第 1 届奥运会 g1 和雅典 c1 共同持有一个 hold 链，那么第 1 届奥运会对象 g1 就持有一个 host 名称的性质，该性质的值就是雅典城市对象 c1。

对象的行为用于管理维护对象的状态，可能是改变状态，也可能读取状态。一种行为可用一个操作来表示，定义一个操作需要一个名字加上一组可能的形式参量。例如，任一个

OlympicGame 对象都可定义一个操作 setYear(year: String)来设定其举办年份 year; 可定义一个操作 setHost (host : City)来设定一个主办城市 host 作为其东道主。当执行一个操作时需要确定一个目标对象、一个操作名字加上对应的实际参量。这些操作的定义并非针对某一个对象，而是针对该类所有对象而定义的。在执行某个操作时，必须确定唯一一个目标对象。例如：g1.setYear(1896)，g1.setHost(c1)。这两个操作执行就确定了 g1 对象的年份和东道主。图 1.1 对象图表示了这两个操作的执行效果。

在一个对象上调用一个操作时，应将该对象理解为操作的对象，而不是调用操作的主体。例如，g1.setYear(1896)应理解为“调用 setYear 操作作用于对象 g1”，而不是“对象 g1 调用了 setYear 操作”。调用操作的主体往往是当前执行程序的某个线程。在 UML 中将持有线程的对象称为主动对象，而一般的对象都是被动对象。

在面向对象编程语言中，实例与对象往往混为一谈，但在 UML 模型中，实例是比对象更抽象的概念。一个对象是一个实例，但一个实例并不一定就是一个对象。例如，int 是一个类型，3 是它的一个实例，而 3 并不是一个对象。

一个对象的性质和操作统称为该对象的特征(feature)。

面向对象主要体现为人们观察分析世界的一种思维方式，而不局限于软件设计和编程。将客观世界中的各个事物都看做是对象，承认客观对象具有自己的规律。面向对象意味着“除了对象，别无他物”。首先，识别对象的标识和边界，以区分各个对象；其次，识别对象的各种状态和行为，用性质来描述状态，用操作来描述行为；最后，为对象确定它所属的类。

1.2.2 类的概念

计算机所实现的对象都来自于特定的类。一个对象是类实例化的结果。我们已建立了 City 类，那么所有的城市对象都应是该类实例化所创建的实例。那么类又是什么概念？

一个类是：

(1) 一组对象的抽象描述。

(2) 这组对象具有相同的特征、约束和语义规范。

一个类相对于其对象是抽象的，而对象是具体的实例。一个类必须先描述一个名字，这个名字应来自客观世界中的某种存在形式。类中并不描述单个对象的性质的值，而应描述该类所有对象共有哪些性质和操作。类图 1.3 中分别描述了 OlympicGame 类和 City 类的部分性质和

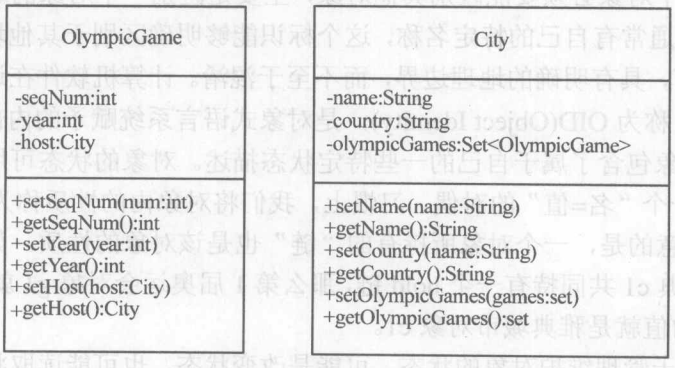

图 1.3 一个类图表示了两个类

操作。性质和操作分别表示在类名下面的两个区间中。注意，图 1.3 不同于图 1.2，用性质代替了关联，即用两个性质代替了原先的 hold 关联。而且添加了一些操作。这些操作都是针对该类的各个性质分别进行写(set)或读(get)的行为。

一个类相对于其对象而言是静态的，而对象是动态的，仅在运行时刻存在。在运行时刻，类不改变，对象是由类经实例化创建出来的，然后调用其操作而改变其状态、执行特定计算并返回结果，最后被撤销。这被称为对象的生命周期。有些对象的生命周期比较短，如一个对话框或者菜单栏是一个对象，通过键盘或鼠标操作结束之后关闭，其生命周期就结束了。而有些对象则有较长的生命周期，如一届奥运会作为一个对象，可能要存在很长时间，这往往需要文件或数据库来存储对象状态。

计算机中实现的对象是对客观对象的一种模拟。我们往往需要从分析客观对象入手，建立业务模型或分析模型，再建立设计模型，逐步抽象到计算机能识别的类。在计算机运行时刻，这些类进行实例化创建计算机对象，人与对象进行交互，对象之间进行交互，最终完成计算并得到结果，这个结果对于客观对象具有模拟作用。这样一个过程应建立在尊重客观事实，正确反映客观规律的基础上。如果计算机中的对象能够正确反映客观对象的规律，就能较好地满足用户的需求，所实现的软件就具有较长的生存周期。同时，人类的认知能力、当前软件实现能力都是有限的。所识别的对象可能与客观对象之间有偏差，也可能不完备；所设计实现的类也可能出现错误，计算机对象所提供的状态和行为可能与用户的要求有差距。这意味着从认识客观对象到建立类是一个循环往复的过程，而不是一蹴而就。

1.3 面向对象特性

下面结合面向对象编程语言(如 C++、Java)，探讨面向对象的几个基本特性：

(1) 封装性(encapsulation)。

(2) 继承性(inheritance)。

(3) 多态性(polymorphism)。

1.3.1 封装性

封装是一种自然的构造，目的是保护内部；同时也是一种人为的构造，目的是方便使用。我们经常使用一个遥控器或者一个简单操作面板来操纵一台空调机。一台空调机就是被封装起来的一个单元，这种封装具有保护内部、方便操作、使用安全等好处。

那么封装是什么概念？封装就是把相关概念组成一个单元，然后可通过一个名字来引用它。

什么是面向对象封装？面向对象封装就是把表示状态的各个性质和对状态的各个操作包装成对象类型，使得对对象状态的存取只能通过封装提供的接口来进行。

在概念上，面向对象封装有三个层面的解释：对象的封装、类的封装和包的封装。

如何理解对象的封装？一个对象封装了自己的状态和行为。在运行时刻，对象的状态表示为一组性质的值，一个对象持有自己的性质的值，而且通过自己的行为对自己的状态施加管理。一般情况下，一个对象不会将自己的性质直接提供给外部来随意改变，因为不安全、易出错。尽管有时说不清究竟会有多大风险，但很少有共有的(public)性质。

一个复杂的对象可能由多个部分组成，每个部分都可持有自己的状态，这种对象往往被称为“复合对象”。这种整体和部分的关系是一种递归结构。无论对象内部有多复杂，从外部看仍然是一个对象。

一个对象对每一个性质都持有一个值，但一个对象对每一个操作并非持有一份拷贝。概念上可以认为一个对象的行为作用在一个对象之上，但在背后的实现中，同一类的多个对象往往共享同一个操作代码，以节省存储空间。所以一个对象所占空间仅为其状态的存储空间。

对象的封装是由类来决定的。类的封装可从三个视图来理解。第一个视图是成员特征。一个类封装了一组性质和一组操作作为该类的特征(feature)。一个类为其各个特征提供了命名空间，各个特征在此空间中具有唯一名称或基调(signature)。基调是指操作的名字加上形式参量。在编程语言中，性质往往表示为变量(如 C++)或域(如 Java)，而操作表示为函数(如 C++)或方法(如 Java)。显然，在同一个类中不会有同名的性质，各操作之间也不会有相同的基调。一个类中的多个特征之间应该是次序无关的。比如，类中的多个性质之间的前后次序应该是无关紧要的。如果一个类中的两个性质之间隐含某种次序，这往往隐含着存在某些错误。

第二个视图是可见性(visibility)，即各个特征可有不同的访问控制，以确定该成员能否被类外访问。根据 UML 规范，可见性说明有以下四种：

(1) private：私有，只有本类内部可访问。简单表示为负号“-”前缀。

(2) public：共有或公共，类外部可访问，只要类是可见的，公有成员就可访问。表示为加号“+”。

(3) protected：受保护的，本类及子类可访问。表示为“#”。

(4) package：包所有，同一个包中的类可访问。表示为“~”。缺省为包所有。

通常类的性质被说明为私有的，而操作说明为公有的，这样只能通过操作来访问或改变对象的状态，可保护关键数据免遭滥用或破坏。所以，一个类中共有部分也称为该类的接口。如图 1.4 所示。

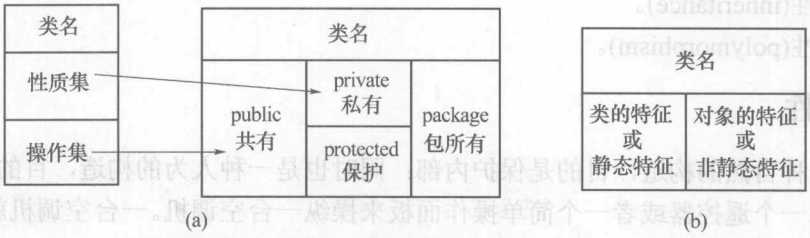

图 1.4　类的封装性的三个视图

(a) 性质通常为私有，操作为共有；(b) 区分是类的特征还是对象的特征。

第三个视图是静态与非静态特征。一个类中除了对象的特征，可能还有类的特征。静态特征即为类的特征，区别于对象的特征，被该类的所有对象所共享。即便该类没有创建任何对象，也能访问其静态特征。静态特征也有可见性说明。静态的操作往往用于管理静态的性质，但静态操作中没有当前对象的概念，也就是说，没有 this 引用或指针来指定当前对象。非静态操作内部都具有 this 引用或指针。在 UML 模型中用下划线表示静态特征。访问一个静态特征应使用类名作为前缀，如“类名.静态特征名”。而非静态特征则为对象的特征。对非静态特征的访问应先区分对象，再确定特征，如“对象名.非静态特征名”。缺省是非静态特征。

类中有一种特殊的操作称为构造器(constructor)，用来描述创建对象时对部分性质的初始化过程。当一个类实例化创建一个对象时，一定有一个构造器得到执行。一个类可定义多个构造器。构造器的说明是类名加形式参量，在运行时将返回新创建的一个对象。看起来类似静态操作，但构造器内部有当前对象的概念，也就是说，具有 this 引用或指针来指定当前所创建的新对象。C++语言中还有析构函数(destructor)，用来描述一个对象在被撤销时自动执行的行为。

包也是一层封装。一个包(package)封装了一组类型，并为这些类型提供了一个命名空间，即一个包中的各类型具有唯一名称。一个包必需命名，包之间可有嵌套结构，形成一个有层次的组织结构，用于管理规模较大的系统或子系统。包中的一个类的完整名称应有包名作为其前缀，UML 采用双冒号进行标识，“包名::子包名::类名”，而 Java 采用点符号。包中的类可选择是否对包外部可见。Java 语言提供了完整的包的机制。C++语言则利用命名空间 namespace 来实现包的封装。

Java 语言自身具有良好的封装性，从包到类型、从类型到其中特征，严格规定其封装性。对 C++语言来说，需要人为设计才具有良好的封装性。其语法允许没有任何封装的变量、函数作为全局特征，而且其命名空间也没有真正的保护作用。

封装性的好处体现在哪里？主要是信息隐藏和状态保持两方面。

1. 封装与信息隐藏

从对象外部来看，良好的封装能隐藏大量细节。隐藏是使用封装将某些信息或实现方法限制在封装结构内部，以约束外部的可见性。隐藏有两种形式：信息隐藏和实现隐藏。信息隐藏就是使封装单元内的信息不被外界察觉，而实现隐藏是指外界不能察觉单元内的实现细节。

信息/实现的隐藏是降低软件复杂性的有效手段。对象外部可把对象看做一个黑盒子，即外部知道对象可以做什么，而不必知道如何做，也不知道内部如何构造。

信息/实现隐藏有以下两个主要优点：

(1) 设计决策局部化。在对象内私有的设计决策的更改对系统其他部分影响很小，甚至没有影响。这样对局部决策的修改对整个系统的影响最小化，这样限制了“修改蔓延”的影响。

(2) 减少信息内容。因对象对外部隐藏实现细节，这使外部用户不会受到对象内部信息特定格式的困扰，这样使对象外部的用户，即其他程序员，就不用干涉对象内部的事情，也防止向对象内部引入不稳定的连接。

2. 封装与状态保持

在对象封装基础上，对象具有保持自己状态的能力。当传统的过程模块(如函数、子程序、过程等)返回到调用方，只是把计算结果返回，而不能保持自己的状态。当同一个过程模块再次调用时与首次调用一样，这样的模块对以前的状态没有记忆。

对象在自身内部封装了状态信息，并保持直到对象被撤销。在对象生命周期中可被多次调用操作，每次状态改变都能保持记忆，而且对象如何保持其状态是隐藏在内部的细节，外部无需知道具体如何处理。

面向对象封装、信息/实现隐藏及状态保持是面向对象的核心思想。

1.3.2 继承性

继承是类与类之间的一种关系，它使开发人员在已有的类的基础上可定义和实现新的类。继承是利用可重用软件构造系统的有效机制。

继承能有效地支持类的重复使用，当需要在系统中增加新特征时，所需的新代码最少，并且当继承性与下面介绍的多态性结合使用时，为修改系统所需变动的源代码最少。

继承性表示较一般的类与较特殊的类之间的关系，如图 1.5 所示。在此关系中，一般性的类(general class)通常被称为“超类”、“基类”、“父类”等，而特殊类(specifc class)被称为“子类”、“衍类”、“派生类”等。简单起见，下面采用“超类”和“子类”的说法。UML 类图中用一个三角箭头和实线从子类指向超类。UML 中将这种关系称为“泛化”或“一般化”。泛化关系可嵌套形成层次结构。

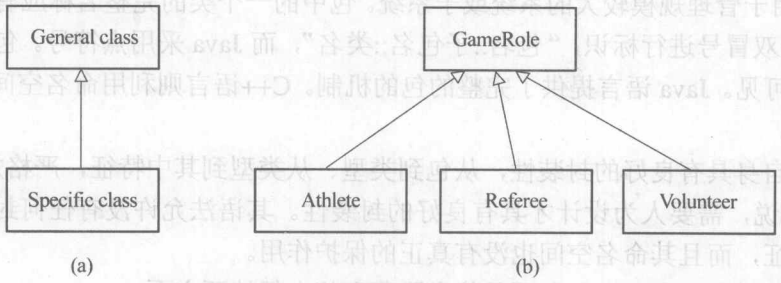

图 1.5　继承性的表示

(a) 继承性表示一般类与特殊类之间的关系；(b) 继承性的例子。

继承性反映自然的分类结构。通常的说法是特殊类“是一种(is-a-kind-of)”一般类。图 1.5 中 GameRole 类表示了奥运会参与者的一般性的角色，而运动员(Athlete)“是一种”奥运会参与者角色，Athlete 就是 GameRole 的一个子类。同样地，裁判员(Referee)和志愿者(Volunteer)也是特殊的奥运会参与者角色。另一种说法是“包括”。例如，奥运会参与者“包括”了运动员、裁判员和志愿者等。

子类继承了超类中定义的特征，而且子类可扩展新特征，且仅描述新特征。继承关系中，子类实质上由两部分组成：继承部分和扩展部分。继承部分是从其超类中继承而来的，而扩展部分是专为子类编写的新特征。扩展部分的设计或多或少地依赖于继承部分。这样使用继承性就能避免大量特征的复制，提供了一种强有力的重用机制，但也存在巨大风险。假如一个类在实现后期才发现继承关系错误，这往往意味着它自己的扩展部分需要重新设计。不仅如此，如果这个类还有几个子类的话，那么这些子类很可能也面临同样的问题，所以建立正确的继承性是极其重要的。

要建立一个正确的继承关系需满足“里氏替代原则”：凡是需要一个超类对象的地方，而实际提供了一个子类对象，总应该是可行的。如果不能满足此原则，那么这个继承关系就有问题了。

在继承结构中，超类自己并不知道它有什么子类，而子类必须知其超类，所以子类依赖于超类。特殊性依赖一般性，具体依赖抽象，这是面向对象设计的基本原则之一。

在继承结构中，子类虽然不能直接访问其超类的私有性质，但子类确实继承了其超类的私有性质，这是因为子类的对象中包含了其超类的所有性质的值。

在继承结构中，当一个子类实例化创建一个对象时，该子类的所有超类也将实例化。当创建一个运动员(Athlete)对象时，不仅 Athlete 类要实例化，而且 GameRole 类也要实例化。事实上，GameRole 类先完成实例化，然后才是 Athlete 类。这意味着，超类的构造器先执行，然后才执行子类的构造器。

继承性允许子类中说明与其超类同名的性质。当通过对象引用来访问某个性质时，在编译