面向对象理论、方法及应用

申贵成 陈蕾 孙媛 杨玺 编著

兵器工业出版社

北京市属市管高等学校人才强教计划资助项目
北京物资学院科技创新平台资助项目

面向对象理论、方法及应用

申贵成　陈蕾　孙媛　杨玺　编著

兵器工业出版社

内 容 简 介

本书在阐述面向对象理论的基础上，采用大量的程序设计案例来说明面向对象的基本概念与应用方法，在内容组织上，侧重知识体系的完整和知识结构的合理组织，严格按照理论、方法及应用的阶梯，让读者能够循序渐进地学习，以掌握面向对象的基本理论，同时辅之以实例，掌握面向对象的程序分析、设计以及系统开发中的应用。在内容的编排上，按照面向对象的基本理论、类与对象、继承与多态、异常处理、输入与输出、图形界面设计、Java 数据库的顺序，从抽象到具体，符合读者的学习习惯。

本书既可作为高等院校计算机科学与技术、信息管理与信息系统、电子商务等相关专业的本科生教材，也可以作为软件系统开发人员尤其是 Java 程序员的参考书或培训教材。

图书在版编目（CIP）数据

面向对象理论、方法及应用/申贵成等编著. —北京：兵器工业出版社，2008.10

ISBN 978-7-80248-003-2

Ⅰ. 面… Ⅱ. 申 Ⅲ. 面向对象语言—程序设计—高等学校—教材 Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2008）第 141841 号

出版发行：兵器工业出版社
发行电话：010－68962596，68962591
邮　　编：100089
社　　址：北京市海淀区车道沟 10 号
经　　销：各地新华书店
印　　刷：北京市登峰印刷厂
版　　次：2008 年 10 月第 1 版第 1 次印刷

责任编辑：常小虹
封面设计：李　晖
责任校对：郭　芳
责任印制：赵春云
开　　本：787×1092　1/16
印　　张：18
字　　数：445 千字
定　　价：22.00 元

前　言

面向对象理论自20世纪70年代诞生以来，在信息技术领域发挥着巨大的作用，已经深入到方法学、数据库、人工智能方面。

本书旨在介绍和解释面向对象的基本理论与方法，并讨论这些理论与方法如何通过程序设计来进行体现。通过本书的阅读，读者可以提高对面向对象理论的理解，并能较好地利用这些理论来进行程序设计，尤其是进行Java语言程序设计，从而积累程序设计的经验，从理论上提高自己使用这些理论的自觉性。

基于上述思想，我们精心编写了本书。书中通过大量的实例，由浅入深、循序渐进、由点到面、系统地讲解了Java程序设计各方面的知识。

第1章系统讲述了面向对象理论的基本概念、发展过程，并简要介绍该理论对信息技术的重大影响，重点讲述了对程序设计语言的影响。

第2章系统讲述了类与对象的关系，介绍了对象如何创建、使用以及消亡的过程，在此基础之上讲述了如何定义类，重点讲述方法的定义过程以及如何使用类的方法。

第3章系统讲述了继承、多态与封装的概念，通过利用继承的方法，实现代码的重用，通过利用封装的方法，实现对属性、方法的限制性开放，使得用户知道他们该知道的，无须知道他们不应该知道的。

第4章系统讲述了异常处理的理论，介绍了如何处理异常、创建自己的异常类以及如何扔出异常的方法，确保系统的安全，避免系统的崩溃。

第5章系统讲述了输入与输出流的基本概念，讲解了如何利用系统定义的输出类与输入流类，利用这些类进行文件操作。

第6章系统讲述了可视化程序设计的基本方法，主要是讲解如何利用已有的类来创建自己的对象、使用这些对象，并能够在这些类的基础上再次进行扩展。

第7章系统讲述了JAVA数据库的基本理论以及SQL语言的语法规则，介绍了JDBC的常用类，利用JDBC接口进行数据库操作，包括插入、删除以及更改记录等方法。

本书的每一章都是按照理论、方法与应用的顺序编写，通过对这一过程的不断重复，加深读者对面向对象理论的理解。

在本书的编写过程中，参考了大量的相关文献和著作，在此向这些文献的作者致以诚挚的谢意，并欢迎读者对书中的错误与不足提出宝贵意见。

编著者

2008年9月于北京

目　录

第1章　面向对象的基本概念

1.1　引　　言

长久以来，人们一直在解决这样一个不合理的现象，即我们认识一个系统的过程和方法同我们用于分析、设计和实现一个系统的过程和方法不一致。

我们对一个系统的认识是一个渐进过程，是在继承了以往的有关知识的基础上、多次迭代往复而逐步深化的。在这种认识的深化过程中，既包括了从一般到特殊的演绎，也包括了从特殊到一般的归纳。而目前我们用于分析、设计和实现一个系统的过程和方法大部分是瀑布型，即后一步是实现前一步所提出的需求，或者是进一步发展前一步所得出的结果。因此，当越接近系统设计（或实现）的后期时，如要对系统设计（或实现）的前期的结果做修改就越加困难了。同时也只有在系统设计的后期才能发现在前期所铸成的一些差错。当系统越大、问题越复杂时，由于这种对系统的认识过程和对系统的设计过程不一致所引起的困扰也就越大。

当一个有数十年实践经验的工程师在回首往事时，经常把自己以往所完成的一些工程称为“遗憾”工程。这是因为当系统设计开始时（概念设计阶段），用户对系统的需求是比较笼统和抽象的，而那时设计者的设计自由度较大，设计者可选用当时已有的原理、方法、工具、环境、构件和器件去设计系统。但是随着时间的推进，用户对系统的需求是越来越细致和具体了。但设计者对系统进行修改的自由度却越来越小了。还应该看到，随着时间的推进，原理、方法、工具、环境、构件和器件的水平是日益提高的，而设计者采用新技术的自由度却受到了限制，如图1-1所示。因此，系统越大、周期越长，成为遗憾工程的可能性就越大。

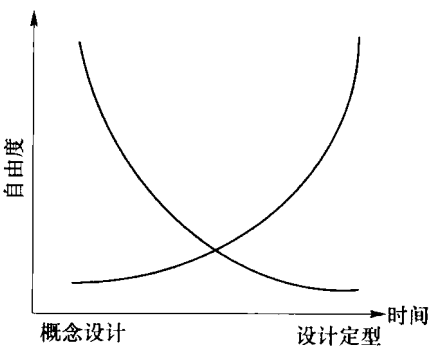

图1-1　用户与系统开发者之间的矛盾

为了解决上述这种不合理的对象，就应使我们分析、设计和实现一个系统的方法尽可能地接近我们认识一个系统的方法，换言之，就应使描述问题的问题空间和解决问题的方法空间在结构上尽可能一致，也就是使我们分析、设计和实现系统的方法学原理和我们认识客观世界的过程尽可能地一致。这就是面向对象方法学的出发点和所追求的基本原则。

和人们认识世界的规律一样，面向对象的基本方法学认为：客观世界是由许多各种各样的对象（事物）组成的，每种对象都有各自的内部状态和运动规律，不同对象间的相互作用和联系构成了各种不同的系统，构成了我们所面对的客观世界。当我们设计和实现一个客

观系统时，如果能够在满足需求条件下把系统设计成是由一些不可变的（相对固定的）部分所组成的最小集合，则这个设计就是优秀的，而这些不可变的（相对固定的）部分就被看成是一些不同的对象。

根据上述对面向对象的基本方法学的粗浅介绍，今后我们所讨论的对象至少应当具有以下特征：

（1）模块性。一个对象是一个可以独立存在的实体。从外部看这个模块，只了解这个模块具有哪些功能，至于这个模块的内部状态，以及如何实现这些功能的细节都是隐藏在模块的内部的。一个模块的内部状态是不受（或很少受到）依赖外界的影响的。同时一个模块的内部状态的改变也不会影响到其他模块的内部状态。因此，各模块间的依赖性是很小的。所以，各种模块才有可能较为独立地为各个系统所选用。

（2）继承性和类比性。人们是通过对客观世界中的各种对象进行分类及合并等方法而来认识世界的，每个具体的对象都是在它所属的某一类对象（类）的层次结构中占据一定的位置。因此，下一个层次的对象应该具有上一层次对象的某些属性，在面向对象方法学中，我们把它称为是一层次的对象继承了上一层次对象的某些属性。另一方面，当人们发现一些不同的对象具有某些相同的属性时，也常常把它们归并成一个类，在面向对象方法学中，我们把它称为是通过对象间的类比而实现了归类。

（3）动态连接性。在客观世界中，由于存在各式各样的对象以及它们之间的相互连接和作用，从而构成了各种不同的系统。因此，我们把对象和对象间所具有的一种统一、方便、动态地连接和传递消息的能力与机制称为动态连接性。

（4）易维护性。任何一个对象都是把如何实现本对象功能的细节隐藏在该对象的内部。因此，无论是完善本对象的功能，还是改正功能实现的细节，都被局限于该对象的内部，而不会传播给外部，这就增强了对象和整个系统的易维护性。

1.2 面向对象技术的形成、现状与发展

20 世纪 70 年代末，面向对象方法学的一些基本概念已在系统工程领域内萌发出来了。如对于系统中的某个模块或构件可表示为问题空间的一个对象或一类对象。到了 80 年代，面向对象的程序设计方法得到了很快的发展，并显示出其强大的生命力。因此是面向对象技术在系统工程、计算机、人工智能等领域得到了广泛的应用。

在 70 年代末，一些系统分析和设计人员已从实践中归纳出一系列符合面向对象方法学原理的一些分析及设计的步骤，只是尚未得到充分和有效的软件工具的支持，这些步骤是：

（1）确定构成该系统的各个组成部分（即对象）及它们的属性；

（2）确定每一组成部分（对象）应完成的功能；

（3）建立每一组成部分（对象）与其他成分（也是对象）的相互关系；

（4）建立各个组成部分（即对象）间的通信关系和接口形式；

（5）进一步协调和优化各个组成部分的性能及相互间的关系，使得该系统成为是由不同的组成部分（即不同的对象）的最小集合所组成的；

（6）分析、设计及实现每个组成部分（即对象）的功能实现细节。

程序设计语言的发展过程是抽象程度不断演变和提高的过程。对程序设计语言的发展影

响较大的几次抽象是过程抽象、语法抽象、数据抽象和进程抽象，但实际上更具有本质意义的是对知识的抽象。70 年代末，具有面向对象方法雏形的软件系统 SIMULA 诞生了。SIMULA 允许用户创建一个面向对象的系统，但它是使用传统的面向数据/过程的语言（ALGOL）编写的。不久，在 1976 年推出 SmallTalk－72，1978 年推出了 SmallTalk－76，1981 年推出了 SmallTalk－80，由此逐步发展和完善了面向对象的程序设计语言的概念。例如，在 SmallTalk－72 中，列表结构和控制结构都具有了对象的概念，但类的概念尚未形成。在 SmallTalk－76 中就有了类的概念，并且用面向对象技术的观点去说明程序接口的性质。在 Xero Learning Research Group 所研制的 SmallTalk－80 系统，则是较全面地体现了面向对象程序设计语言的特征。由于 SmallTalk－80 的推广使用，使得面向对象的方法和原理很快地推广到与信息技术有关的其他各个领域。

在此还必须提到 Ada 语言及其环境在推广面向对象方法和技术上所起的作用。Ada 是美国国防部为解决 20 世纪 60 年代末到 70 年代初的软件危机而投入了巨大的人力和物力所研制的一种语言和环境，目前 Ada 已经成为世界上许多国家的标准军用语言，并且已扩展到非军用的其他各个领域。Ada 并非是一种标准的面向对象的语言，但由于 Ada 所具有的模块化、信息隐藏、数据抽象、并发任务执行等特点，所以 Ada 所倡导的是一种非常接近面向对象方法学原理的系统设计思想。因此，由于 Ada 的推广和应用，以及它所取得的良好效果，也强有力地推动了面向对象技术的发展。目前人们已经普遍认为：Ada 是一种基于对象（Object-Based）的语言。

除了 SmallTalk 和 Ada 外，我们还应提到的就是 Modula－2 了。在 Pascal 语言推出后的 10 年，于 1978 年定义一种 Modula－2 的语言，并且于 1979 年首次研制出 Modula－2 的编译程序。在这种语言中特别强调了模块的概念，并且还提出了模块是可嵌套的特性。在每一个模块内（也可是在不同的模块内）的各个对象间的联系是通过 IMPORT 和 EXPORT 功能实现的。显然，Modula－2 所提出的这些概念也对后来的面向对象技术的发展做出了贡献。

从 20 世纪 80 年代起，人们基于以往已提出的有关信息隐藏和抽象数据类型等概念，以及由 Modula－2、Ada 和 SmallTalk 等语言所奠定的基础，再加上客观需求的推动，逐步地发展和建立起较完整的面向对象的软件系统（Object Oriented Software System，OOS）的概念和机制。按照面向对象的软件系统的方法学，我们对一些将要用于面向对象技术中的基本概念给出以下定义：

（1）信息（*Information*）是对事物的一种表示和描述；

（2）软件（Software）是描述信息处理的信息；

（3）软件系统（Software System）是一种处理工具；

（4）程序设计环境（Programming Environment）是一个用于设计、生产和使用软件系统的环境；

（5）对象（Object）是一个由信息及有关对它进行处理的描述所组成的包；

（6）消息（Message）是对某种对象处理的说明；

（7）类（Class）是对一个或几个相似对象的描述；

（8）实例（Instance）是被某一个特定的类所描述的一个对象，因此，每一个对象都是某个类的实例，而类是对各个实例的全部相似性的描述；

（9）方法（Method）是描述对象对消息的响应。

对于上述的这些概念，本书将在以下的各个章节给出更为精确的定义和详细的说明。但是我们已经初步认识到，对象是一种普遍实用的基本逻辑结构，是一个以有组织的形式含有信息的实体。它既可以表示一个抽象的概念，也可以表示一个具体的模块，既可以表示软件，也可以表示硬件。于是，面向对象的方法学既提供了一个分析、设计和实现系统的统一方法，又提供了描述、设计和实现硬件和软件系统的统一框架。

把以上所述的这种面向对象的软件系统的基本概念和运行机制应用到其他的各个领域后，就得到了一系列相应领域的面向对象的技术和应用，这在20世纪80年代得到了很大的发展：

（1）面向对象的设计（Object Oriented Design，OOD）。系统的设计过程可看成是把系统所要求的问题分解为一些对象及对象间传递消息的过程。

（2）面向对象的语言（Object Oriented Language，OOL）。在这种语言中，可以把数据和处理数据的过程结合为一个对象。对象即可以像数据一样被处理，又可以像过程一样描述处理的流程和细节。例如，在SmallTalk－80中，把一切都看成是对象；在Ada中，把包（Package）看成是对象；在Modula－2中，把模块看成是对象。

（3）面向对象的数据库（Object Oriented DataBase，OODB）。把对象作为存取和检索的单位，把传统数据库定义的数据定义语言和数据操作语言融为一体，这种概念也是构成语义数据库和知识库的基础。

（4）面向对象的智能程序设计（Object Oriented Intelligent Programming）。把知识系统中的智能实体作为对象，一个对象所具有的知识是该对象的静态属性，一个对象所具有的知识处理方法则是该对象的智能行为的体现。

（5）面向对象的体系结构（Object Oriented Architecture）。能支持对于对象的描述、存储和处理的计算机体系结构。一些新的计算机体系结构都在试图能支持面向对象的程序设计和软件系统的概念，如某些多处理器系统、神经网络计算机及连接机制等。

1.3 基本概念

用计算机解决问题时需要用程序设计语言对问题的求解加以描述，实质上，软件是问题求解的一种表现形式。显然，如果软件能够直接地表现求解问题的方法，则软件不仅易于被人理解，而且易于维护和修改，从而提高了软件的可靠性和可维护性。此外，如果能按人们通常的思维方式来建立问题域的模型，则可以提高公共问题域中的软件模块化和重用化的可能性。面向对象的方法学的基本原则是：按人们通常的思维方法建立问题域的模型，涉及尽可能自然地表现求解方法的软件。

为了实现上述基本原则，必须建立直接表现组成问题域的事物以及这些事物间的相互联系的概念，还必须建立适应人们一般思维方式的描述范式。在面向对象的设计方法中，对象和传递消息分别是表现事物及事物间相互联系的概念。类和继承是适应人们一般思维方式的描述范式。方法是允许作用于该对象上的各种操作。这种对象、类、消息和方法的程序设计方式的基本点在于对象的封装性和继承性。通过封装能够将对象的定义和对象的实现分开，通过继承能够体现类与类之间的关系，以及由此带来的动态绑定和实体的多态性，从而构成了面向对象的基本特征。

1.3.1 对象

客观世界的问题都是由客观世界的实体及实体间的相互关系构成的，我们把客观世界的实体称之为问题空间（问题域）的对象。显然，对象不是固定的。一本书可以是一个对象，一个图书馆也可以是一个对象。可见，世界上的各个事物都是由各种对象组成的，任何事物都是对象，是某一个对象类的一个元素。复杂的对象可由相对比较简单的对象以某种方法组成，甚至整个世界也可以从一些最原始的对象开始，经过层层组合而成。从这个意义上讲，整个客观世界就是一个最复杂的对象。

本质上，我们用计算机解题时借助某种语言规定对计算机实体施加某种动作，以此动作的结果去映射解，我们把计算机实体称为解空间（求解域）的对象。

从动态的观点来看，对象的操作就是对象的行为。问题空间对象的行为是极其丰富多彩的，而解空间对象的行为极其死板。因此，只有借助于极其复杂的算法才能操纵解空间对象而得到解。传统的程序设计语言限制了程序员定义解空间对象。而面向对象语言提供了对象概念，这样，程序员就可以自己去定义解空间对象。

从存储的角度来看，对象是一片私有存储，其中有数据也有方法。其他对象的方法不能直接操纵对象的私有数据，只有对象私有的方法才可操纵它。

从对象的实现机制来看，对象是一台自动机，其中私有数据表示了对象的状态，该状态只能由私有的方法改变它。每当需要改变对象的状态时，只能由其他对象向该对象发送消息，对象响应消息后按照消息模式找出匹配的方法，并执行该方法。

在面向对象的设计中，对象是应用域中的建模实体。所有对象在外观上都表现出相同的属性，即固有的处理能力和通过传递消息实现的统一的联系方式。

在面向对象的程序设计中，对象是系统中的基本运行实体。换句话说，对象是具有特殊属性和行为方式的实体，对象占有存储空间且具有传统设计元的数据，如数字、数组、字符串等。给对象分配存储单元就确定了给定时刻对象的状态。与每一个对象相关的方法定义该对象上的操作。

1.3.2 消息和方法

如何要求对象完成一定的处理工作？对象间如何进行联系？所有这一切都只能通过传递消息来实现。消息用来请求对象执行某一处理或回答某些信息的要求；消息统一了数据流和控制流。某一对象在执行相应的处理时，它可以通过传递消息请求其他对象完成某些处理工作或回答某些信息；其他对象在执行所要求的处理活动时，同样可以通过传递消息与别的对象联系。因此，程序的执行是靠在对象间传递消息来完成的。

发送消息的对象称为发送者，接受消息的对象称为接受者。消息中只包含发送者的要求，它告诉接受者需要完成哪些处理，但并不指示接受者应该怎样完成这些处理。消息完全由接受者解释，接受者独立决定采用什么方式完成所需的处理；不同的对象对于形式相同的消息可以有不同的解释，能够做出不同的反应。对于传来的消息，对象可以返回相应的回答消息，但这种返回并不是必须的，这与子程序的调用/返回有着明显的不同。

消息的形式用消息模式刻画，一个消息模式定义了一类消息，它可以对应内容不同的消息。对于同一消息模式的不同消息，同一个对象做的解释和处理都是相同的，只是处理的结

果可能不同。对象固有处理能力按消息分类，一个消息模式定义对象的一种处理能力。这种处理能力是通过该模式及消息引用表现出来的。所以，只要给出对象的所有消息模式及相应于每一个消息模式的处理能力，也就定义了一个对象的外部特性。消息模式不仅定义了对象所能够受理的消息，而且还定义了对象的固有处理能力，它是定义对象接口的唯一信息。使用对象只需了解它的消息模式，所以对象具有极强的黑盒性。

把所有对象分成各种对象类，每个对象类都定义一组所谓的方法，它们实际上可视为允许作用于该类对象上的各种操作。

当一个面向对象的程序运行时，一般要做三件事：首先，根据需要创建对象；其次，当程序处理信息或响应来自用户的输入时要从一个对象传递消息到另外一个对象；最后，若不再需要该对象时，应删除它并回收它所占用的存储单元。

由此可见，面向对象的设计方法放弃了传统语言中控制结构的概念，以往的一切控制结构的功能都可以通过对象及其相互间传递消息来实现。

1.3.3 类和层次

在面向对象程序设计中，对象是程序的基本单位，相似的对象可以和传统语言中的变量与类型关系一样，归并到一类中去。程序员只需定义一个类对象就可以得到若干个实例对象了。

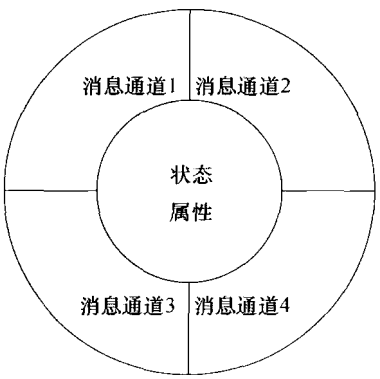

图 1-2 对象模型

具体来说，类由方法和数据组成，它是关于对象性质的描述，包括外部特性和内部特性两个方法，如图 1-2所示。类通过描述消息模式及其相应的处理能力来定义对象的外部特性，通过描述内部状态的表现形式及固有处理能力的实现来定义对象的内部实现。

一个类实质上定义的是一种对象类型，它描述了属于该类型的所有对象的性质。例如，Integer 是一个类，它描述了所有证书的性质。对象是在执行过程中由其所属的类动态生成的，一个类可以生成多个不同的对象。同一个类的所有对象具有相同的性质，即其外部特性和内部特性都是相同的。一个对象的内部状态只能由其自身来修改，任何别的对象都不能改变它。因此，同一个类的对象虽然在内部状态的表现形式上相同，但它们可以有不同的内部状态，这些对象并不完全一模一样。

从理论上来讲，类是一个 ADT（抽象数据类型）的实现。信息隐藏原则表明类中的所有数据是私有的。类的公共借口是由两种类型的类方法组成。一种是返回有关实例状态的抽象辅助函数，另外一种是用来改变实例状态的变换过程。

一个类的上层可有超类（Superclass），下层可以有子类（Subclass），形成一种层次的结构。这种层次结构的一个重要特点是继承性，一个类（直接）继承其超类的全部描述。这种继承具有传递性，即如果 C1 继承 C2，C2 继承 C3，则 C1 继承了 C3。所以，一个类实际上继承了层次结构中在其上面的所有类的描述。因此，属于某个类的对象除具有该所描述的特性外，还具有层次结构中该类上面所有类描述的全部特性。

在类的层次结构中，一个类可以有多个子类，也可以有多个超类。因此，一个类可以直

接继承多个类，这种继承方法称为多重继承（Multiple Inheritance）。如果限制一个类至多只能有一个超类，则一个至多只能继承一个类，这种继承方式称为单重继承或简单继承（Single Inheritance），如图 1－3 所示。在简单继承情况下，类的层次结构为树结构，而多重继承是网状结构。

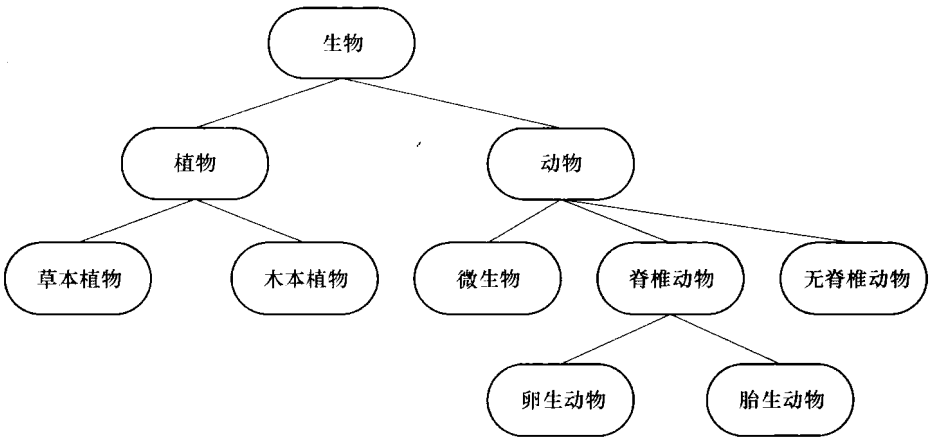

图 1－3　类的层次结构

抽象类（Abstract Class）是一种不能建立实例的类。抽象类将有关的类组织在一起，提供一个公共的根，其他一系列的子类从这个根派生出来。抽象类刻画了公共行为的特征并将这些特征传给它的子类。通常一个抽象类只描述与这个类有关的操作接口；或是这些操作的部分实现，完整的实现被留给一个或几个子类。有时，抽象类描述了这个类的完整实现，但只有在将这个类和其他的类组合在一个新的类中时它才有用。抽象类已为一个特定的选择器集合定义了方法，并且这些方法服从某种语义，所以抽象类的通常用途是用来定义一种协议（或概念）。

例如，类 Person，如果它只用来说明 Teacher 和 Student 所具有的公共特征，如名字、地址以及对它们的方法，单独的 Person 实例在实际中是不具有意义的，只有和它的子类 Teacher 和 Student 结合起来才能用来描述一个部门中的具体对象，因此我们可以认为 Person 是一个抽象类。

判别一个类是否是抽象类，最容易的办法是使用它。如果有必要建立一个类，但是你只使用从它那儿继承过来的子类，则这个类是抽象类。例如食肉动物表示一个概念，从它可以得出更具体的推论，例如狮子和老虎。你绝不会遇到一只动物，简单地认为是食肉动物，它总是一个具体种类的食肉动物。

一个抽象类包含了操作接口但没有实现，就定义了一种协议，如果包含有某种实现，则抽象了用缺省实现定义了一种协议。通过从多个父类中继承定义，几个协议可被组合在一起。

综上所述，类是对一组对象的抽象，它将该种对象所具有的共同特征（包括操作特征和存储特征）集中起来，由该种对象所共享。在系统构成上，则形成了一个具有特定功能的模块和一种代码共享的手段。

1.3.4 继承

继承性（Inheritance）是一种自动的共享类、子类和对象中的方法和数据的机制。每个对象都是某个类的实例，一个系统中类对象是各种封闭的。如果没有继承性机制，则类对象中数据和方法就可能出现大量重复。为了使读者更清楚地理解继承性是什么，以及它是如何提供我们所期望的好处的，下面将进一步讨论继承关系，并用一个简单的例子加以解释。

当类 Y 继承 X 时，就表明类 Y 是 X 的子类，而类 X 是 Y 的超类。类 Y 由两部分组成，继承部分和增加部分。继承部分是从 X 继承来的，增加部分是专为 Y 编写的新代码。由语言规则可定义 X 的成员映射成 Y 的继承成员，再加上程序员专为 Y 写的代码，就构成了 Y。映射可以简单地等同，即 Y 的继承部分完全等同于 X 的成员。不过，继承映射可以比这种简单的等同更为丰富。例如，在进行 X 到 Y 映射后，程序员可以对 X 的性质重新命名、实现、复制和置空等。

如同任何语言性质一样，程序员应合理地使用继承性。如使用多重继承性，则有时会使系统变得较为复杂，从而导致继承路径查找时间也较大。当然，随着网络技术的发展，多重继承性也许略占上风。

每一种面向对象的语言都提供一套用于继承的机制。语言级提供的关键字用来表示期望映射的种类，某些映射还要求附加一些信息。例如，如果在进行 X 到 Y 映射是要重新实现 X 的某一成员，那么必须给出重新实现的代码。

继承关系常称“即是”（is a）关系。这是因为当类 Y 继承 X 时，由继承性可知 Y 现在已经具备 X 的全部性质，所以 Y 即是 X。无疑 Y 也可能包含了 X 中没有的特性，它具有比 X 更多的性质。

因此，继承关系常用于反映抽象和结构。Rectangle（矩形）是一种特殊类型的 Polygon（多边形），这很容易从继承关系中获得。当 Rectangle 继承 Polygon 时，Rectangle 获得了 Polygon 的全部性质。另外，Polygon 是 Closed Figure（封闭图形），所以 Rectangle 也继承了 Closed Figure 的所有性质。

图形系统中最高层的图形具有像素宽度、颜色以及平移和旋转等特性，这些是在根类 Figure（图形）中定义的。另外，Closed Figure 有边界和区域，Polygon 增加了 Number of Sides 属性。所以，根据继承性，Rectangle 有像素宽度、平移过程和计算区域函数等。当然一些性质可直接继承，也有一些应在继承时做某种修改再用。如，类 Figure 定义的 Set-Pixel-Width 可能适合 Rectangle，但是，Caculate-Area 过程应重新定义，以便给出更有效的实现。

可见，在面向对象的图形系统中，继承性减轻了增加新图形原语的工作。同样地，面向对象方法利用继承性也有助于开发快速原型。继承性是实现可重用成分构造软件系统的最有效的特性，它不仅支持系统的可重用性，而且还促进系统的可扩充性。

继承性封装是一种信息隐藏技术，用户只能见到对象封装界面上的信息，对象内部对用户是隐藏的。封装的目的在于将对象的使用者和对象的设计者分开，使用者不必知道行为实现的细节，只需用设计者提供的消息来访问该对象。

封装的定义为：

（1）一个清楚的边界，所有的对象的内部软件的方位被限定在这个边界内；

（2）一个接口，这个接口描述这个对象和其他对象之间相互的作用；