面向对象程序设计技术及应用

MIANXIANGDUIXIANG CHENGXUSHEJI JISHU JI YINGYONG

王丽珍 李卫宏 姚小宁 姜 跃 编著

云南科技出版社

前　言

面向对象技术被人们誉为软件技术的一场革命，它与传统的结构化软件技术相比具有许多优点。应用面向对象技术可以有效地提高软件的开发效率和质量，改善软件的可靠性和重用性。今天，面向对象的概念和应用已经渗透到计算机的许多领域，如程序设计与开发、CAD/CAM、图形处理、动态仿真、数据库系统、用户界面、分布式系统、人工智能、体系结构等。面向对象技术已经成为计算机及相关专业人员必须掌握的一项重要技术。

然而，编者在教学和工程实践中发现，不少技术人员在使用面向对象编程语言(OOPL)进行软件开发时，对面向对象的概念和机制理解得并不透彻，有些甚至停留在传统的结构化软件设计上，这是非常遗憾的。为此，编者在总结多年研究生课程《面向对象技术》教学及工程实践中应用 Turbo Vision (Borland C++的 DOS 人机界面构架)的经验，撰写本书，意在使面向对象技术的基本概念（对象、类、继承、消息、方法等）、特殊机制(重置、多态、动态联编、类属类、多重继承等）及基本手段（类库、构架等）经过本书以实例分析，深入浅出而又不乏严谨地介绍，以深入读者之心，并通过下篇 Turbo Vision 的具体实践，引导读者一步步从传统的结构化软件技术走进面向对象软件技术的大门。

当然，选择 Turbo Vision 程序设计作为面向对象程序设计示例，不仅只考虑到将它作为一个面向对象程序设计技术的实践环境，若仅为此，我们可能选择 object windows (Borland C++的 Windows 人机界面构架)作为示例。更主要的考虑是介绍 Turbo Vision 程序设计方面的文献太少，系统地介绍 Turbo Vision 程序设计的参考书几乎没有，而 DOS 应用程序的开发需要在今后相当一段时间内仍然存在。为一举两得，我们断然选择相对“冷”的 Borland C++构架——Turbo Vision 作为实践和研究面向对象程序设计技术的示例。相信您能从书中得到意想不到的收获。

本书由云南大学计算机系王丽珍副教授、姚小宁硕士、解放军某部工程师李卫宏硕士、云南财贸学院姜跃教师编写，王丽珍副教授汇编和校审了全书。

由于我们的水平有限，书中难免有错误和不妥之处，敬请批评指正。

编　者

目 录

上 篇 面向对象程序设计

下 篇　OOP 技术应用示例——Turbo　Vision 程序设计

上篇

面向对象程序设计

第一章　为什么面向对象程序设计

第一节　引 言

进入 20 世纪 90 年代，在计算机领域中，软件的开发和研究人员面临着不容乐观的形势。一方面，随着微电子技术的突飞猛进，硬件技术的发展十分迅速，应用对软件系统在规模与复杂程度上的要求在不断提高；另一方面，软件与硬件能力的差距却至少有两代处理机之多，而且这种滞后状况正呈逐渐增大之势。传统的软件开发方法学越来越难以满足使用者对日益庞大而复杂的软件系统的需求，使软件开发步履维艰。因此，如何提高软件能力已经成为制约软件系统发展的关键因素。

虽然还无法预言能否在不远的将来从根本上解决软件能力问题，但必须提高软件能力这一点是肯定的。朝着这一目标，一种新的程序设计范型已经出现，这就是面向对象程序设计（OOP），与之对应的是面向对象程序设计语言（OOPLs）。

第二节　软件能力所面临的问题

所谓软件能力（software capability）是指软件开发过程中所采用的软件技术、软件工具和抽象层次等因素对开发目标所起的作用。此概念是一个相对概念，当产品的规模和复杂程度改变时，同样的技术（工具，抽象层次）就呈现出不同的作用；当产品特征不变时，不同的技术（工具，抽象层次）也将呈现出不同的作用。当然，要求软件产品来适应软件能力是不容乐观的，所以只有让软件能力去适应应用的变化。

21 世纪，计算机正不断成为越来越多人生活中不可缺少的一部分，特别是新的应用对软件开发提出了更为复杂的要求，需要采用更为复杂的数据结构和软件系统结构来适应更广范围应用的需求。因此，这种外部推动力迫使软件开发能力必须有一个飞跃。

软件开发能力所面临的主要问题是如何提高软件质量和软件生产率。从汇编语言到高级程序设计语言，标志着软件质量和软件生产率的一次质的飞跃。造成这次飞跃的决定性因素是编译理论和技术的扩充和完善，实现了从高级语言到机器代码的自动变换。自 20 世纪 70 年代起，为了解决“软件危机”，人们从结构化程序设计的研究中，提出了一些基本原则和方法，例如模块化、数据抽象、E—R 模型、数据流方法等，并应用到软件系统的分析和设计上，促进了软件工程技术的发展。但软件开发迫切需要解决的问题，如生产率低下，软件重用和可扩充能力差，可维护性低等，并未从根本上获得解

决。而且许多软件开发还是没能摆脱手工作坊式的生产过程，大量程序员仍然像工匠一样在生产软件。成千上万花费了大量心血编制的各种软件，随着应用需求的扩大和变化很难再得到利用，因此很少能见到寿命超过 10 年的软件。

造成软件开发水平的落后，不能完全归结为硬件性能的迅速提高，也不能归结于应用水平发展的加速度，其根本原因是软件能力的不适应性。这种不适应性越来越激化了软件缺乏足够的保证来与能力相当的硬件匹配这一尖锐矛盾。

提高软件能力是改变软件开发水平落后的根本出路。而面对软件能力所面临的问题，采用新的程序设计范型，使软件开发产生质的飞跃，成为一种必然的选择。

第三节　选择新的程序设计范型

程序设计范型（programming　paradigm）是人们在程序设计时所采用的基本方式模型。例如：过程程序设计，结构化程序设计，函数程序设计和逻辑程序设计等，都是不同的程序设计范型。一种程序设计范型可以认为是一类程序设计语言的基础，是执行设施的集合，或者是关于计算机系统的思考方式，它体现了一类语言的主要设计风格。程序设计范型一般都有代表性的程序设计语言，如过程程序设计与 FORTRAN，结构化程序设计与 Pascal 和 C，函数程序设计与 Lisp，逻辑程序设计与 Prolog 等。这其中，影响范围最大就非结构化程序设计 SP(Structured Programming)莫属了。因此，在认识面向对象程序设计以前，我们认为首先应回顾一下传统的结构化程序设计（SP）及其设计思想、结构与特点，以便在以后更好地理解什么是面向对象程序设计（Object-Oriented Programming），简称 OOP。

SP 诞生于 20 世纪 60 年代，是对当时所爆发的“软件危机”的挑战，而提出的以“自顶向下，逐步求精”为设计思想的程序设计方法与技术。SP 强调了功能抽象和模块化，熟悉 Pascal 或 C 的读者都了解，在 Pascal 或 C 中程序(program)是由一些过程(procedure)或子程序(subroutine)组成的集合。这些过程通过参数传递数据。这些做法纵然有一些优点，但从本质上看，SP 仍是一种面向过程的设计方法。它把数据和在数据上允许的操作分离为相互独立的实体，用数据描述问题空间的客体，程序代码则体现处理加工这些数据的算法。这样，程序员在编制程序时，必须时时刻刻考虑要处理的数据结构和数据类型，对不同的数据格式（结构和类型）即使要做同样的处理，或者对于相同的数据格式要做很相似的处理都必须编写不同的程序。可见，用传统的 SP 方法设计出来的软件，可重用的部分很少。此外，这种现象还引出了一些其他的问题，诸如软件的稳定性、可修改性、可维护性都很低。所以，采用新的程序设计范例，就成了一种必然的选择。

面向对象程序设计范型将计算看作是一个系统的开发过程。系统由独立对象的集合组成，对象既包含着数据，也包含着作用于这些数据的操作。这样的操作过程虽然也是一段程序，但在 OOP 中被重新命名为“方法”，以区别通常意义下的过程。一个对象中的数据代表了它的状态，方法则代表了它的行为，外界要改变一个对象的状态，也就是对它所包含的数据施加操作，只能向这个对象发出请求（面向对象中称之为“消息”），

然后由这个对象的相应的方法来改变状态，这体现了对象的封装性。方法对外界只提供接口，不暴露实现细节，数据结构也对外界屏蔽，体现了“信息隐蔽”。以上可以看到，面向对象的程序虽然也是用数据和操作来表示客观世界的实体：数据表示实体的结构，数据值表示实体的状态，操作表示实体的行为，但是与传统的程序有着本质的区别。传统的过程为中心的程序设计范型，使得实体的结构表示（即数据）依赖于行为表示（即过程），而 OOP 则相反。

具有共同特性和共同用途的一组对象，被抽象成“类”。新创建的类可“继承”已有类的数据与方法，使直接重用变为现实。一组有继承关系的类构成了类层次结构。因此，OOP 提供了更高层次的抽象，可用于建立预定义的类库，甚至可以提供针对某种特定应用的类库，称之为构架。这样 OOP 把重用和抽象这两个概念结合在一起，利用两者的优势，成为提高软件能力的武器。在 20 世纪 90 年代，一些程序员已经受益于 OOP，但是在软件开发中 OOP 还是没有很好的得以普及。一部分原因是没有充分理解、认识到 OOP 的优势，或者说是虽然已经使用了面向对象程序设计开发工具，但编制出的程序仍然是传统的、结构化的，并没有体现出 OOP 的真正特性与机制。当然，我们必须看到，要实现从以过程为中心的传统程序设计范型，转为以对象为中心的 OOP，还需要时间。因此，本书的一个重要目的就是帮助读者完成上述范型的平滑的转移，充分认识到 OOP 的概念、方法和机制。一旦软件开发者从 OOP 中得到了经验和收益，那么这种范型的转移将逐步完成。

概括起来，对于面向对象这一新的程序设计范型，我们可以说它既保留了原有范型的优点，还有其他范型所缺乏的或不具备的特点。如建立在类及其继承性基础之上的重用能力；可以应付复杂的大型的软件开发；便于扩充与维护，抽象层次更高，具有较高的生产率。正是由于面向对象的这些特点，它已不仅仅是局限于程序设计领域，并逐步渗透到了软件的开发与设计，数据库，图形处理，CAD，系统模拟以及知识库的组织与管理等计算机的各个领域。可以肯定地说，这种新的范型得到广泛的理解和使用之后，必将有力地推动软件工业的新的发展。

第四节　采用 OOP 的现实可能性

OOP 的形成并不是最近几年才有的，实际上已有二十几年的历史了，其根源可以追溯到 20 世纪 60 年代后期，当时在挪威计算中心的 Kristen Nygaard 和 Ole-Johan Dahl 合作开发了一种叫做 Simula 67 的仿真语言。Simula 67 首次引入了类，协同程序和子类的概念，非常类似今天的 OOPLs。它证实了基于类的语言具有很强的造型能力，以及将数据和操作封装在一起是有益的。它的出现开始使人们承认，如果能对一组对象的公共特性（也就是类和类层次结构）进行程序设计，就可以节省程序设计的工作量，提高程序的重用水平。

以后，在 20 世纪 70 年代中期 Alan Key 在施乐公司的 Palo Alto 研究中心开发了第一

个完整的、健全的 OOPL: Smalltalk，以后在此基础上又进行了改进和扩充，形成了 Smalltalk-80。该语言的每一个元素都被作为一个对象来实现，从语言本身，程序设计环境以及程序设计风格都是面向对象的。因此即使今天，Smalltalk 仍被认为是最纯的面向对象程序设计语言。到 20 世纪 80 年代中期，OOPLs 已经呈百花齐放的局面，形成了面向对象语言的几个类别。一类是纯面向对象的语言，如 Smalltalk，Eiffel；另一类是混合型面向对象语言，即在传统的结构化设计语言中增加面向对象的机制，如 C++，Objective-C。还有一类是与人工智能语言相结合而形成的，如 Flavors，LOOPs。今天，随着网络技术的广泛应用，又出现了适应于网络应用的 Java 语言。图 1—1 描述了面向对象程序语言的发展历史。

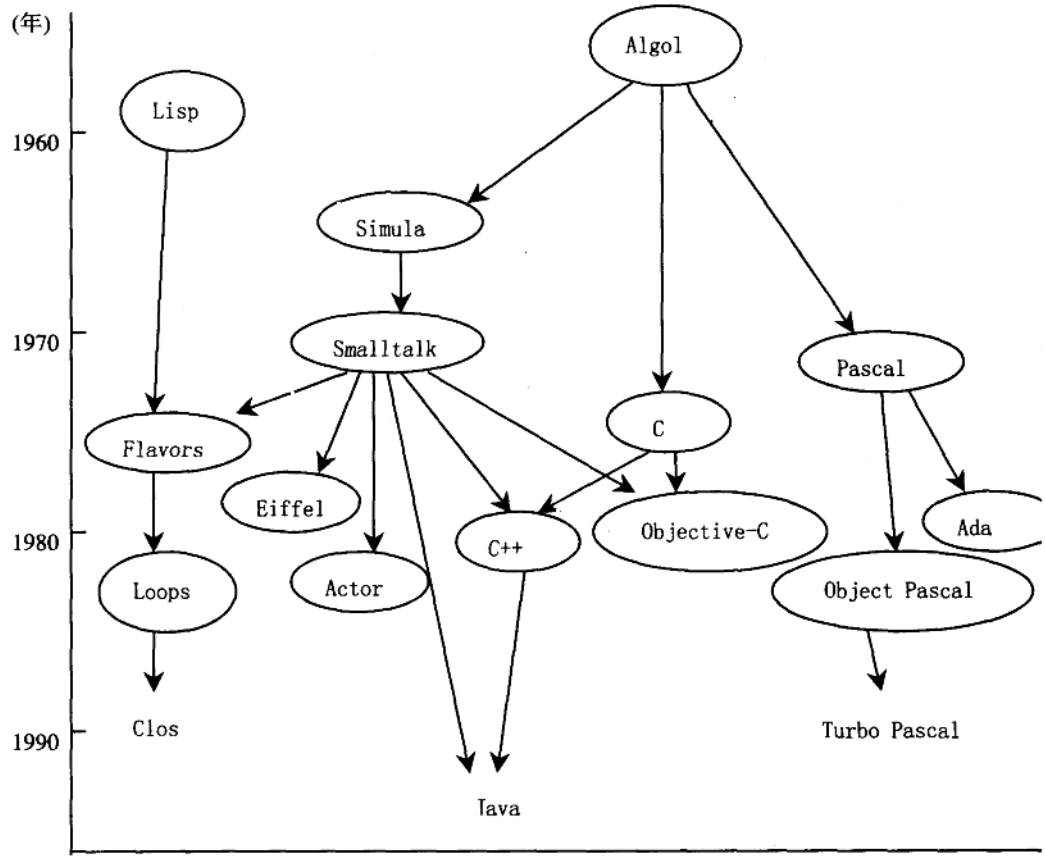

图 1—1　面向对象程序设计语言发展史

虽然 OOP 的一些基本概念的形成及运用已经很早就出现了。但引入一种新的程序设计范型并被广泛的接受是一件很不容易的事情，这不仅有技术的因素，还有经济和社会因素在起作用。

从图 1–1 中，大家可以发现虽然早在 20 世纪 60 年代、70 年代就出现了 Simula 和

Smalltalk 这样的 OOPLs，但为什么在相当一段时间未受到软件产业的注意，而是直至 20 世纪 80 年代中后期，才开始得到足够的重视呢？这是由诸多原因造成的。一个原因是当时能充分理解并使用它们的人长期没有超出大学里研究人员的范围，没有在社会上发生实在的经济效益之前，很难说服软件产业采用一个全新的范型；另外一个重要的原因是它们对计算机平台的性能需求很高，例如需要大量的存储空间和很高的运算速度。这样对于商业软件的开发者来说，在经济上是不合算的。到了 20 世纪 80 年代中后期，随着大规模集成电路的出现，硬件的迅速发展逐步排除了采用 OOP 在性能价格比方面的障碍，就连现在的个人计算机也能满足 OOP 的基本要求。高性能的工作站，高质量的图形显示设备，有丰富工具集并提供良好支持的开发环境，都使得软件产业有了采用 OOP 的基本条件。

另外，一个新的程序设计范型是否受到欢迎，是否有生命力还要必须考虑到能否充分地利用已有的资源和开发环境。例如，能够在通用的机器上运行，可以在常用的操作系统的支持下工作，不和传统断然割裂等。OOP 的一个基本特征，就是对体系结构和支持软件环境没有突变的要求，因而不存在难以重用现有资源的问题。这样从经济与社会因素来看，OOP 是可行的。再从技术的角度来看，OOP 的基本机制是对象、类、消息，基本特征是继承性、封装性、多态性。OOP 用对象为客观事物造型，认为任何事物都是对象，复杂的对象可由简单的对象以某种方式组合而成，这和人们习惯的思考方法比较一致；把所有对象都划分成各种对象类（简称类），每个对象类都定义了一组数据（表示对象的静态属性）和一组方法（允许施加的操作）。按照子类（派生类）与父类（基类）的关系，把若干个对象类组成一个层次结构的系统，在这种层次结构中，下层的派生类具有和上层的基类相同的特征（数据和方法），这种现象称为继承（inheritance）。但如果在派生类中对某些特征又做了重新描述，则在派生类中的这些特征将以新描述的为准，即低层的特性将屏蔽高层的同名特性，这种层次分明，层次间的关系明确，正是按照客观世界本来的规律行事，也是应用所希望的。

综上所述，正因为目前具备了技术、经济和社会方面的客观条件，又有应用的推动，加之软件产业自身发展生产力的要求，使得 OOP 具有成为新一代的主流范型的现实可能。

第五节　小 结

在这一章里，我们从如何提高软件能力出发引出为什么需要 OOP，又从技术、经济和社会等各方面讨论了采用 OOP 的现实可能性，其目的就是让读者对 OOP 有一个初步的认识，同时了解一些 OOP 的基本思想和方法，以便准确地把握 OOP 的本质，更好地运用于自己的研究和开发实践中。在第二、第三章，我们将具体地介绍 OOP 的基本概念及其在 OOPLs 中的体现。

第二章　面向对象程序设计的基本概念

第一节　引 言

OOP 作为一种程序设计范型，主要解决程序设计方面的问题，因此与程序设计语言的发展密切相关。事实上，最早 OOP 的一些概念正是由一些特定语言机制体现出来的。

早在 20 世纪 50 年代后期，程序员在编制 Fortran 的大型软件时出现了变量名在不同的程序部分发生冲突的问题。这引起了 Algol 语言设计者的注意，并决定在 Algol 语言中采用“阻挡”（Barriers）来隔开程序段中的变量名，这样就产生了 Algol60 以“Begin...End”为标识的程序块，并以在程序块中分设局部变量来防止与其他程序块中同名变量的冲突。这是首次在编程语言中提出保护（Protection）或封装（Encapsulation）的概念。至今，我们仍能从一些高级语言如 Pascal，C，Ada 等中看到应用这种概念的痕迹。

到了 20 世纪 60 年代中后期，Simula-67 语言的设计者采用了 Algol 语言的程序概念，并最早开始使用数据密封，它的 CLASS 结构和以后的抽象数据类型基本是一样的。更重要的是在这个语言中首先提出了“类”的概念，所以，现在的面向对象程序设计语言被公认为发源于 Simula。进入到 20 世纪 70 年代，随着微型计算机的问世，人们设想在个人计算机上用简单的方法处理各种形式复杂的信息，在 Xerox 公司的 Palo Alto 研究中心，Key 和其他研究人员在借鉴了 Lisp 以及从 Simula 吸取核心概念——类之后，研制了 Smalltalk 语言。“Smalltalk”这个名字源自“talk　small（少说话）”，意思是你可以通过很少的工作量就完成许多任务。Smalltalk 在系统设计中强调对象概念的统一，引入对象、对象类、方法、实例等概念和术语，采用动态联编和单继承机制。它还建立了以 OO 编程语言为核心，集各种软件开发为一体，建立 OO 设计环境，配有很强的图形功能和多窗口用户界面。Smalltalk 现在被公认为是最纯的面向对象程序设计语言，它也是第一次采用“面向对象（OO）”一词，所以说，Smalltalk 作为一种新的、纯粹的面向对象程序设计语言，同时又体现和发展了面向对象方法的许多概念和机制，为面向对象程序设计奠定了基础。正是通过 Smalltalk-80 的研制，使人们注意到面向对象方法所具有的模块化、信息封蔽与隐藏、数据抽象、继承性、多态性等独特之处，这些优异特性为解决大型软件管理，提高软件可靠性、可重用性、可扩充性和可维护性提供了有利的手段与途径。自从面向对象程序设计语言出现，发展到现在的多种的 OOPLs，比较典型的有美国施乐公司（Xerox）的 Smalltalk-80，Smalltalk/v（微机版），美国 PPI 公司的 Objective-C，美国 Interactive　Softwaring　Engineering 公司的 Eiffel，以及 C++（包括 Turbo　C++，

Borland C++，Microsoft C++，Visual C++)。在此我们要特别提一下 C++，它是由 Bell 实验室的 B.Stroustrup 着手在 C 语言的基础上加以扩展，引入了 Simula-67 的主要特性，使之成为一个面向对象语言。由于在此前 C 语言已经在 20 世纪 80 年代成为通用的开发语言，并被广大的程序设计人员所接受，不仅仅可以用于微机，而且可以用于范围很广的计算机结构和环境。因此，在 C 的基础上扩展而成的 C++，虽然不是纯粹的面向对象的语言，却拥有 OO 的多种特性，并继承了 C 的语言特性，与 C 完全兼容，并保持内部一致性，高效率等特点，使得大量采用 C 语言开发的编程人员，系统、环境容易向 C++扩展，促成了程序设计范型的平滑转移。这样也使 C++唯一有可能成为标准化的面向对象程序设计语言。因此在短短的几年，C++就获得了广泛的应用。

以上我们从技术引用的角度谈了面向对象程序设计语言的发展历程。其目的就是使读者了解到面向对象的一些基本概念的由来。可以看出，它们并不是被凭空地、任意地组合到一起的，而是经过研究人员的不断改进与优化，才形成了今天集众家之所长的面向对象语言机制。在本节的开始，我们就提出了特定的 OOP 概念都是通过 OOPLs 中特定的语言机制来体现的，所以在下面的章节我们通过 C++语言的描述，从程序设计的角度来讨论这些概念，使读者更清楚地了解 OOP 的基本内涵。

第二节　对象及类

长期以来，人们一直在设法争取使描述问题的问题空间与解决问题的求解空间在结构上尽可能的保持一致，也就是使我们分析、设计和实现系统的方法同我们认识客观世界的过程尽可能一致。

客观世界中存在着各种各样的事物，我们不可能在很短的时间内把这些事物都完全认识。实际上，人们对客观世界的认识是一个不断积累，渐进的过程，是在继承了以往有关知识的基础上，多次迭代往复而逐步深化的思维组织模式。这样的思维组织模式是多年来人类在考虑问题，以及解决问题时做到条理化、层次化、客观化的一般性经验总结。这其中既包括了从一般到特殊的演绎手段（如继承等），也包括了从特殊到一般的归纳形式（如类等）。正如大英百科全书指出：人类在认识和理解现实世界时，思维过程大都分成三个方面来进行组织。

（1）从现实世界中区分出特定的客体及其属性。例如，从一棵树的大小或空间位置关系，来与其他的客体相区别。

（2）对客体的整体和组成部分加以区分。例如，区分一棵树和组成这棵树的树根、树枝、树叶。

（3）对不同种类的客体给出表示，也就是形式化，然后在此基础上加以区分。例如，对树和石头这两类客体分别给出形式表示，来区分这两类对象。

在这些思维过程中，演绎和归纳都是在实体间的“相似”这一关系上进行的。客观事物发展过程中存在着同和变异。只有同才能有所继承；只有变异，事物才能够向前发

展。当人们去认识客观事物和改造客观事物时，“首先是按照相似原理和关系，把所要研究的问题区分成一定的相似系统与类别”，“分类之后，进一步对事物进行详细的解剖和分析”。解剖分析后再进行“综合优化”，并在事物的运动中和运动的相互关系中去考察客观事物的静态相似和动态相似的关系，宏观相似和微观相似的关系，纵向相似和横向相似的关系。例如：今天我们看到一辆轿车，明天又看到一辆卡车和客车，它们除了外型、颜色的不同之外，都有发动机、传动装置、变速设备、刹车装置等相似的特征，这样，我们可以构造一个类——“汽车”，（如图 2—1）这是认识过程中一种由特殊到一般的归纳能力。另外一种功能是由一般到特殊的演绎功能，也就是一种分类方法，一个按照面向对象方法设计的系统的构造恰好符合这一演绎过程。例如：交通工具的分类可由图 2—2 表示，交通工具按不同的功能可分为水上的、陆地的、天上的，分别为轮船、汽车、飞机，对这些类别再加以一些限制，就可以再深化，如飞机按不同用途和外型，又可划分成客机、战斗机。这就是人类处理自然问题的自然思维方式。

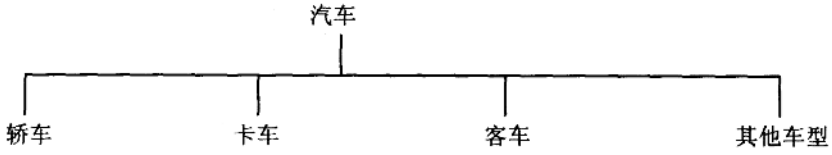

图 2—1　汽车类的归纳

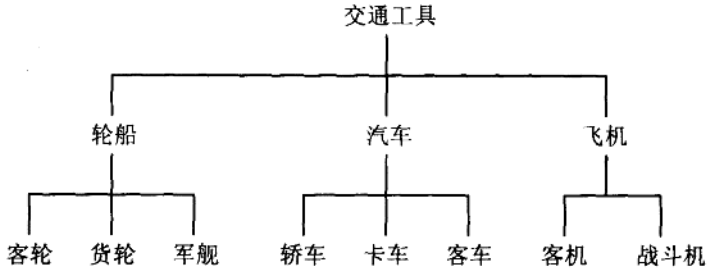

图 2—2　交通工具的演绎（分类）

为了更加直接、自然地模拟上述人类处理问题的自然思维方式，在程序设计方法学中引入了类和对象的概念，借此来形象地表达人类的思维组织模式，就形成了面向对象程序设计方法学。

面向对象方法学认为，客观世界是由各种各样的“对象”所组成，任何事物都是对象，每个对象都有自己的运动规律和内部状态，每一对象都属于某个“对象类”，都是该对象类的一个组成部分。复杂的对象可以是由相对比较简单的各种对象以某种方式而构成。不同对象的组合及相互作用就构成了我们要研究、分析和解决的客观系统。那么究竟何为对象和类呢？面向对象程序设计中的对象和类与我们现实世界中的对象、类的概念有无区别呢？

一、对 象（Object）

客观世界都是由实体及实体间的相互关系构成，我们把客观世界的所有实体都看成是对象。就是说世界上的一切事物小到一个原子、一个人，大到地球，乃至整个宇宙都可以认为是对象。从哲学概念上讲，它们都是客观对象。我们大脑中存在的一些概念和认识，也可以抽象为对象，是主观对象。不管什么对象都有它的属性，比如某人的姓名、年龄、性别、住址、电话号码等都是这个人的属性。对象的属性值可因施加于该对象的行为动作而改变。例如，一个人若搬家了，就要相应改变其“住址”、“电话号码”等属性值。另外，若改变对象的属性及状态，如一个电视机有音量、色彩、亮度、对比度、频道等属性，若想改变色彩、音量、频道时，只能通过电视机面板上或遥控器上的特定操作来实现，而不可能允许用户直接对电视机内部的元器件进行操作。这其实就是一种封装，它的主要思想就是将对象的属性，以及在这些属性上允许的操作对外界进行屏蔽。

外部实体对象在计算机系统中的内部表示被称为软件对象，简称对象。在面向对象的程序设计中，“对象”是系统中的基本运行实体。也就是说，“对象”是具有特殊属性（数据）和行为方式（方法）的实体。

根据以上对对象的介绍，一个对象至少应当具备以下的特征：

（1）模块性。一个对象是一个可以独立存在的实体。从外界只能了解到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都是被“屏蔽”在模块的内部。一个模块的内部状态是不受外界的影响的。同时，一个模块内部的状态改变也不会影响其他模块的内部状态，因此各模块间的依赖性是很小的。

（2）继承性和类比性。人们是通过对客观世界的各种对象进行分类及合并等方法来认识世界的。每个具体的对象都是在它所属的某一个对象类的类层次结构中占一定的位置。因此，下一层次的对象应具有上一层次对象的属性，在面向对象方法学中称为继承。继承性的概念在下面一节将被详细介绍。另外，利用类比性，我们可以把一些具有相同或相似属性的不同对象归并成一个类，这就称为通过对象间的类比实现归类。

（3）对象标识性。对象具有唯一的识别功能，在建立对象时，必须赋以对象唯一的标识符，用来唯一而且永久地标识该对象。

（4）动态连接性。在客观世界中，由于存在着各式各样的对象以及它们之间的相互连接作用。这些相互之间的作用是动态发生的。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之为动态连接性。

以上我们从概念的角度上谈了对象，在本章第四节我们将从功能方面对对象做进一步探讨。

二、类

“类”是日常生活中的一个常见术语，字典中解释为“许多相似或相同事物的综合”，“物以类聚，人以群分”就是分类的意思。由“类”这个词可以派生出许多词组，如类型、类比、同类等等。由此可以看出“类”不是自然存在的，根据我们前面所讲，类是通过人们在日常生活中观察客观事物不断积累总结出的一些事物的共同性质，与其他事物加以区分。这些共同性质的集中体现就是一个“类”的特征。

类就是对一组客观对象的抽象，它将该组对象所具有的共同特征集中起来，以说明该组对象的共同能力和性质。在面向对象系统中，人们并不是去描述逐个的对象，而是将注意力集中于具有相同属性的一类对象，抽象出这样一类对象的共同结构和行为。具有相同或相似属性和操作的一组对象的一般描述，称之为对象类，简称类；类整体地代表一组对象，对外屏蔽内部结构和实现。属于同一类的对象可以共享类所提供的描述。

在面向对象程序设计中，类是关于对象性质的描述，包括外部特性和内部实现两个方面。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，通过描述内部壮态的表现形式及固有能力的实现来定义对象的内部实现。类的形式如图 2–3 所示。

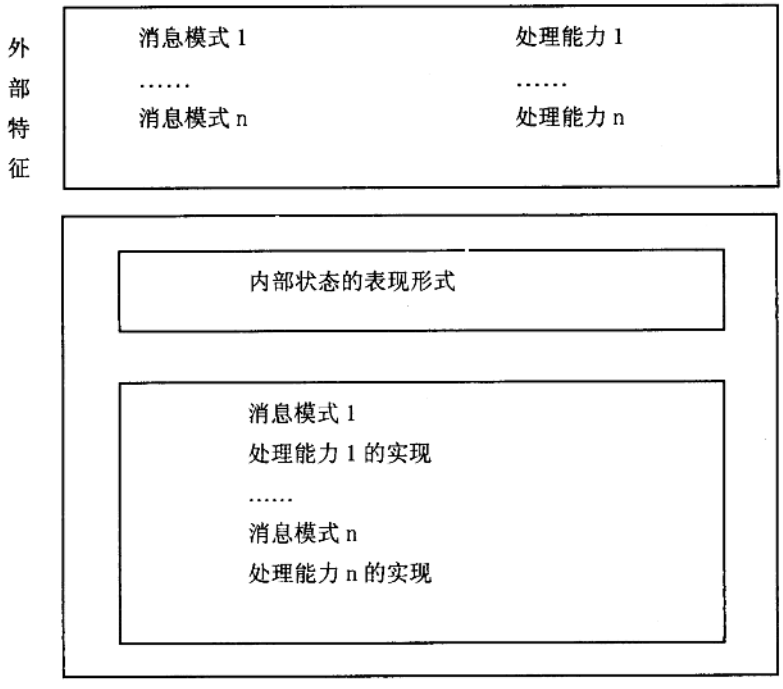

图 2—3　类的形式

那么如何用数据类型将图 2—3 类的形式很好地描述出来呢？用传统的简单数据类型显然是不行的，因为若用整型、数组等简单的数据类型表示一个实体对象时，人们很难将它与它的计算表示关联起来，两者之间的概念相差太大，不可能对这样一个相对复杂

的形式进行描述，所以就引入了抽象数据类型（Abstract Data Type，简称 ADT）的概念。

抽象是一个普通的概念，就是对一个复杂事物或系统的简化描述，它集中注意力于事物或系统的本质方面，而忽略其细节。数据抽象最本质的一点就是把数据类型的表示和它的实现加以分离。从图 2—3 中可以看出处理能力的表示是在外部特性中，它的实现则在内部实现中表示，把表示和实现分离开来。

抽象数据类型（ADT）的定义为：一个值集和作用在值集上的操作集。抽象数据类型可分为 4 个部分，如图 2—4 所示。语法和语义构成 ADT 的说明，让使用者了解这个抽象数据类型的特征。表示和算法构成 ADT 的实现，展示这个 ADT 是怎样做成的。ADT 的语法说明规定了所有操作符或过程名，被操作对象的数目及其类型，以及返回值的类型。ADT 的语义说明是表示类型的行为及其意义，避免出现语义上的错误。实现的表示是指出 ADT 的值在存储中如何表示。算法则说明操作是怎样实现的，严格地给出对表示的使用和操作。使用 ADT 不需要知道是怎样实现，有关实现的知识不能直接被使用，这是 ADT 的特点，也是最关键的地方。

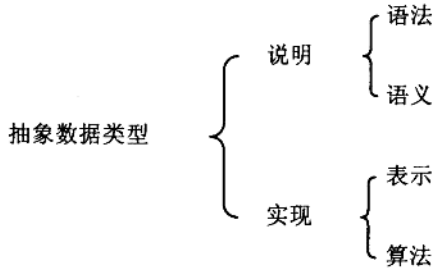

图 2—4 抽象数据类型的结构

在面向对象程序设计中，类实现了抽象数据类型。首先，将具有相同语义特性（即结构特性、操作特性、约束特性）的一组对象组成类之后，就可进一步将这些对象的共性加以抽取，并进行统一的说明，从而减少了各个对象对共性的重复说明。其次，类将数据结构上的抽象与功能上的抽象结合起来，实现了传统方法所不具备的更高级的抽象。这些功能可以在下一小节的实际例子中体现出来。

所有类被组织成一个有根的有向无环图或一个层次结构（类层次结构）。一个类可以从其分层结构中直接或间接祖先那里继承所有的属性和方法。一个类的上层可以有超类（Supeclass），下层可有子类（Subclass），形成一种层次结构。

三、对象及类在 C++中的实现

以上两小节，我们已经初步搞清楚了对象，以及类的来历和在面向对象程序设计方法中的重要意义。但这些概念是如何在面向对象程序设计语言中实现的呢？这一节我们将通过一个例子来具体说明对象及类在 C++语言中的表述。