C++ 语言大全

彭木昌 安瑞飞 赵学龄 郑小军 编著

国防工业出版社

C++语言大全

彭木昌　安瑞飞　赵学龄　郑小军　编著

国防工业出版社

(京)新登字 106 号

图书在版编目(CIP)数据

C++语言大全/彭木昌等编著。—北京：国防工业出版社，1993
ISBN 7-118-01159-2

Ⅰ.C
Ⅱ.彭…
Ⅲ.①C++语言　②程序语言-C++
Ⅳ.TP312

C++语言大全
彭木昌 安瑞飞 赵学龄 郑小军　编著
责任编辑　鹿啟炳　　王　宁
*
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 100044)
新华书店经售
国防工业出版社印刷厂印装
*
787×1092 毫米　16 开本　　印张 25 1/2　　590 千字
1993 年 9 月第一版　　1993 年 9 月第一次印刷　　印数:0 001—4000 册

ISBN 7-118-01159-2/TP・152　　定价:27.50 元

前　言

自 90 年代初以来，软件与硬件之间的差距至少有两代处理器之隔，并且这种差距还在增大。不断增加的复杂性、多样性和相互关联性是当今大多数软件系统的共同特征。计算机正朝着分布式处理、并行处理、网络化和软件生产工程化的方向发展，而传统的软件开发方法已不能满足需要。面向对象以一种全新的设计和构造软件的思维方法，开拓了程序设计方法史上的新世纪。

面向对象方法的基本思想是：对问题域进行自然的分割，以更接近人类思维的方式建立问题域模型，以便于对客观的信息实体进行结构模拟和行为模拟，从而使设计出的软件尽可能直接地描述现实世界，构造出模块化的、可重用的、维护性好的程序，并能控制软件的复杂性和降低费用。在面向对象方法中，对象作为描述信息实体的统一概念，它把数据和对数据的操作融为一体，是一个自足模块，通过方法、消息、类、继承、封装和实例化等机制构造软件系统，并为软件重用提供强有力的支持。

C++ 是当今最受欢迎的面向对象语言，因为它既融合了面向对象的能力，又保留了 80 年代最流行的 C 语言的许多重要特性，C 程序员可以很容易地转向 C++，并扩大其编程知识，增加其编程能力，提高其编程水平，而不必头学习一种新的语言。不过如果用与 C 语言相同的编程方法来编写 C++ 程序的话，那么，这本身就意味着失败。这并不是说如此编写的程序不能运行，而是说未能充分地利用 C++，未能发挥面向对象无以伦比的优越性。面向对象技术将从根本上改变程序员的工作方法，加速产生下一代软件的速度。

与许多其他新技术仅仅局限于研究机构所截然不同的是，面向对象已为大多数软件领域所接受。目前，在国外的许多商业软件公司中，成千上万的程序员已经在使用 C++ 开发产品。在我国，C++ 也逐渐流行起来，并大有迅猛发展的趋势。这是一股促进软件产业化的洪流。

本书集国内外最新技术与作者的研究成果和经验之大成，力图把面向对象方法学与 C++ 语言紧密结合，体现既系统又实用两大特色。由于面向对象技术正处在不断发展和完善阶段，再加上作者水平所限，本书不足之处在所难免，欢迎读者批评指正。

最后，作者感谢在本书撰写和定稿中给予了热情支持，付出了辛勤劳动的鹿啟炳同志，还要感谢王秀清、黄文华、雷励、阎怀君等同志的密切合作。

作　者

于北京系统工程研究所

1993 年 6 月

目　录

第一部分　面向对象技术

第二部分 C++语言

第三部分　常用算法

第四部分　实例研究

第一部分　面向对象技术

第一章　面向对象的基本概念和特征

1.1　引　　言

面向对象不仅仅是一种新的程序设计技术，而且是一种全新的设计和构造软件的思维方法，它使计算机解决问题的方式更加类似于人类的思维方式，更能直接地描述客观世界。

大家知道，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述形式。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，从而可提高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。面向对象方法学的基本原则是：按人们通常的思维方式建立问题域的模型，设计出尽可能自然地表现求解方法的软件。

和人们认识世界的规律一样，面向对象方法学认为：客观世界是由许多各种各样的对象组成的，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就构成了各种不同的系统，构成了我们所面对的客观世界。可见，对象是一种普遍适用的基本逻辑结构，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，当然也就既可以表示软件，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。

为了实现面向对象的基本原则，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(paradigm)。在面向对象方法学中，对象(object)和传递消息(message passing)分别是表现事物及事物间相互联系的概念。类(class)和继承(inheritance)是适应人们一般思维方式的描述范式。方法(method)是允许作用于该类对象上的各种操作。这种对象、类、消息和方法的程序设计范式的基本点在于对象的封装性(encapsulation)和继承性。通过封装能将对象的定义和对象的实现分开，通过继承能体现类与类之间的关系，以及由此带来的动态聚束(dynamic binding)和实体的多态性(polymorphism)，从而构成了面向对象的基本特征。

1.2　抽象数据类型

在面向对象的程序设计中，我们把一个实体内的数据及其操作所形成的描述称为…

个对象。对象中的数据由域组成，域可以引用其他的对象，与这个对象有关的操作刻画了该对象的行为。每个对象有一个类型，我们称这个对象为该类型的一个实例，类型提供了对相似的对象进行分类并归纳这些对象的共同特征的一种手段。一个抽象数据类型是将类型及其有关的操作集合封装在一起的一个数据模型。

支持抽象数据类型的语言必须能够支持信息隐蔽，也就是说，一个对象只能通过为这个抽象数据类型定义的接口进行访问和修改；用于实现这个抽象数据类型及其操作的内部实现细节、数据结构和存储表示，对访问和处理这个对象的使用者是不可见的。这表明对象具有公共接口，以及这些接口的私有表示和实现。这些抽象类型的基本观点很简单。考虑在一个传统程序设计语言中的一个基本类型，例如 C 中的整型，语言提供了有限个数的操作，如(+、*、- 表示整数的加、乘和减操作，还有余和商操作等。)用户简单地使用这些操作处理整数，一个整数的内部位串表示(如使用十六位二进制补码)是完全隐蔽的。处理整数的程序可以很容易地移植到对整数使用完全不同的内部表示的系统中，并能够正确地编译和执行。

使用抽象数据类型的语言将这种方法扩展到程序中的每个对象或数据上。抽象操作接口来进行，这就是所谓的封装概念，我们将在 1.7 节作详细地介绍。

使用抽象数据类型，为这个整数集合定义一系列操作，对这个集合的访问和更新只能通过由这些操作提供的接口来进行。在内部，一个整数集合可以用链表形式来表示，或表示为一个整数数组，或使用其他的表示形式，尽管内部表示形式发生变化，而在使用这个整数集合时都通过接口来进行，那么这个集合对使用者就没有什么不同，使用者也不必因为内部表示发生变化而更改他的程序。

抽象数据类型是面向对象程序设计中组织程序的主要原则。一个类型结构设计完善的程序将减少并局部化类型之间的依赖，因此增强了软件的维护性。

1.3 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，一家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式组成，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，整个客观世界可认为是一个最复杂的对象。

本质上，用计算机解题是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之为解空间(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是极其死板的。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。

从存贮的角度来看，“对象”是一片私有存储，其中有数据也有方法（例程或单个指令)。其他对象的方法不能直接操纵该对象的私有数据，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特性，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。

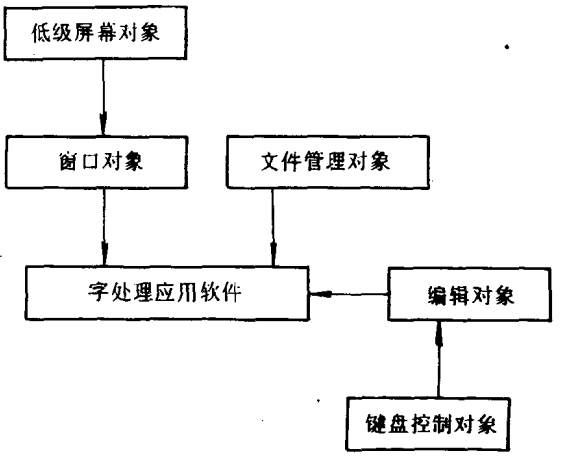

图 1.1　使用对象编程的工作方式

在面向对象程序设计中，“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（数据）和行为方式（方法）的实体，是将各种数据和对这些数据进行操作的各种函数约束在一起的一种语言结构。它是记录概念的具有变革性的拓延，记录允许我们将各种数据组织在一个程序中，而对象允许我们将数据和方法连结在一个独立的程序块中。

由于对象包括了数据和方法，它们就像微型的、独立的程序，这就允许将它们构造成程序块，建立更加复杂的对象。这很像晶体管元件可以用来构造电路一样。例如，假定你需要建立一个字处理应用软件，你可以建立一系列对象，而不必编写一系列函数来处理各种所需的任务，如图 1.1 所示。这个程序分成几个对象，如低级屏幕对象、窗口对象、文件管理对象等等。这种表达形式的优点是什么呢？很容易将一个程序的操作部分分隔开。由于可以把对象设计成相互之间独立工作方式，这就使得对程序的维护更加容易。因为程序是由对象的组合构成的，而不是单个函数组成的，对函数的维护和修改是一件十分困难的事情。另外，某些对象，如低级屏幕对象和键盘控制对象是用来建立其他对象的，这有助于隐藏低级屏幕对象的细节，并且允许我们以搭积木的方式来构造各种对象。

1.4　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，如果需要，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，接收消息的对象称为接收者。消息中只包含发送者的要求，它告诉接收者需要完成哪些处理，但并不指示接收者应该怎样完成这些处理。消息完全由接收者解释，接收者独立决定采用什么方式完成所需的处理。一个对象能够接收不同形式、不同内容的多个消息；相同形式的消息可以送往不同的对象；不同的对象对于形式

相同的消息可以有不同的解释，能够做出不同的反映。对于传来的消息，对象可以返回相应的回答信息，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，可以对应于内容不同的消息。例如，定义“+an Integer”为实体“3 ”的一个消息模式，那么“+4”、“+5”等都属于该消息模式的消息。对于同一消息模式的不同消息，同一个对象所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。

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。

1.5 类和类层次

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(instance)对象了。

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，它描述了所有整数的性质(包括整数的算术运用和大小比较的实现)，“3 ”、“5 ”和“10”等这些具体整数都是 Integer 这个类的对象，都具备算术运算和大小比较的处理能力。

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实现都是相同的。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够改变它。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，这些对象并不是完全一模一样的。

一个类的上层可以有超类，下层可以有子类，形成一种层次结构。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(间接)继承 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全部描述。因此，属于某个类的对象除具有该类所描述的特性外，还具有层次结构中该类上面所有类描述的全部性质。

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，这种继承方式称为多重继承(如图 1.2(a) 所示)。如果限制一个类至多只能有一个超类，则一个类至多只能直接继承一个类，这种继承方式称为单重或简单继承(如图 1.2(b) 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。

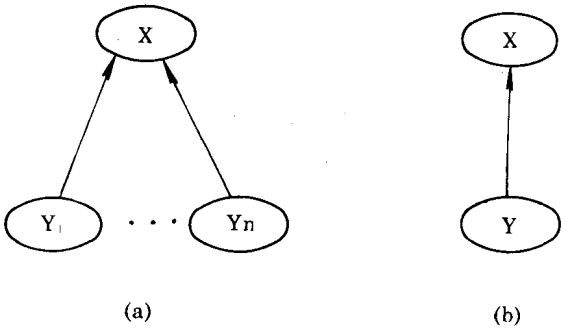

图 1.2　继承方式(X 是 Y，Y1，Y2，...，Yn 的超类)

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方法接口；或是这些方法的部分实现，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)。

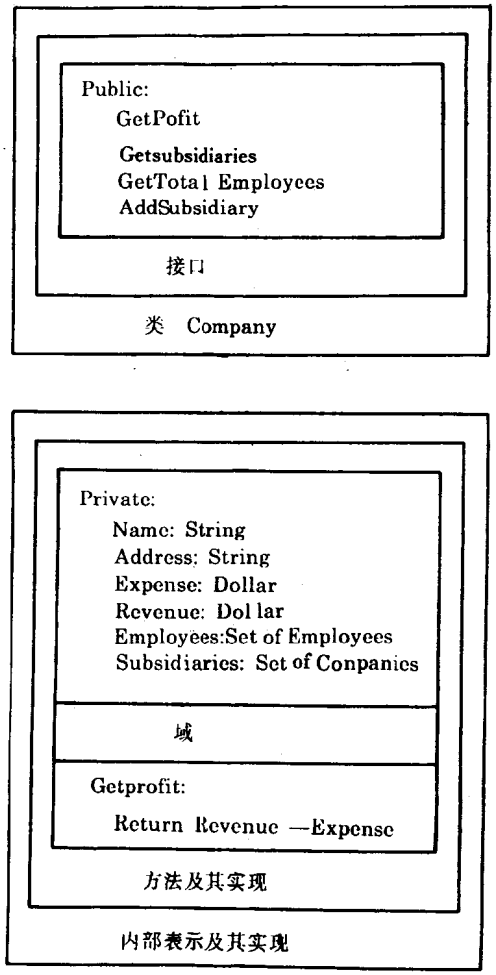

图 1.3　公司表示为一个类 Company

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、地址以及对它们的访问，单独的 Person 实例在实际中是没有意义的，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，最简单的办法是使用它。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，从它可以得出更多具体的推论，如狮子和老虎。你决不会遇到一只动物就简单地认为是食肉的，它总是一个具体种类的食肉动物。

一个抽象类包含了方法接口但没有实现，这定义了一种协议。如果包含有某种实现，

则抽象类用缺省实现定义了一种协议。通过从多个父类中继承定义，可以把几个协议组合在一起。

每个类变量(即对象)表示这个类的一个实例。如果定义几个对象具有相同的类，它们常表示包含有不同值的一个集合。图 1.3 是一个把公司表示成为一个类的例子。例如，IBM，SUN，DEC 等都是类 Company 的实例，它们具有不同的名字、地址等状态，只有通过给定组成其接口的方法集合才能处理它们。

可见，一个类至少包括类名、用于处理这个类的实例的接口、内部表示和方法的内部实现等。

我们假定存在一个由这些方法使用的隐含的目标对象，方法可以带有参数。直观上讲，目标对象是消息即方法的接受者。换句话说，一个方法每次调用适用于一个且仅有一个这个类的实例。这个实例就是目标对象。上例的目标对象是这些公司中的一个：IBM，SUN 或 DEC。下面给出类 Company 的方法的几个描述：

(1)名字：GetProfit

结果：返回当前财政年度公司的利润。

(2)名字：GetSubsidiary

结果：返回目标父公司所有的子公司的集合。

(3)名字：AddSubsidiary

参数：类 Company 的一个实例，将成为目标父公司的一个新的子公司。

副作用：在当前的目标父公司的子公司集合中插入一个新的子公司。

类 Company 的其他元素包括：

(4)内部表示：Name，Address，Expense，Revenue，Employees 等。

(5)内部实现：在后面我们会看到，方法是通过访问和更新内部表示的值来实现的。

因此，一个类包含对象状态的描述(内部表示)，以及实现这个类的方法的代码。要注意，表示类的实例的内部状态的域值是完全不同的，属于特定的对象。例如，IBM 的内部表示由它特定的 Expense，Revenue，Employees，Subsidiaries 等组成。因此，对一个类的每个实例，必须为其分配存贮以保存内部表示。

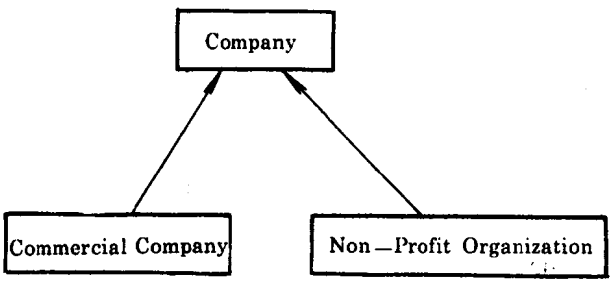

图 1.4　一个公司的类层次

然而，所有的实例共享实现方法的代码。因此，只有一组代码用于实现 GetProfit，GetTotalEmployees，AddSubsidiary 以及其他的方法。正如我们前面说过的，这些方法与一个目标对象一起被访问，并带有参数。面向对象的系统知道如何将相应的方法应用于目标对象，而不破坏它们的内部状态。这非常类似于传统程序设计语言中的过程调用。

下面举例说明类层次。盈利公司和非盈利组织有大量的共同之处，两类机构都包含有名称、地址、人员等信息。非盈利组织有更具体的信息，如政府许可、赞助商等；同样，盈利

公司也有一些特殊内容，如纳税减免等。

因此，为获得非盈利组织和盈利公司的共同信息和行为，我们可以建立一个类 Company，并允许类 CommercialCompany 和 Non－ProfitOrganization 从它那儿继承。CommercialCompany 和 Non－ProfitOrgination 的每个实例也是 Company 的一个实例。图 1.4 说明了继承层次。但注意有的 Company 的实例既不是 CommercialCompany 的实例也不是 Non－ProfitOrgination。例如，我们可以有一个公司由其他几个盈利公司拥有，但不是它们中任何一个公司的子公司，也不产生盈利，并且不是非盈利组织。

CommercialCompany 和 Non－ProfitOrgination 可称为 Company 的子类，Company 称为 CommercialCompany 和 Non－ProfitOrgination 的父类。子类和父类的关系是可传递的。例如，如果 SemiconductorFirm 是 CommercialCompany 的一个子类，则由传递性，它也是 Company 的一个子类。这表明它继承了 Company 和 CommercialCompany 的行为和表示。当然，除此之外，它还继承了 CommercialCompany 特有的行为和表示，这些行为和表示是 Company 的实例所没有的。

从这个例子可以看出继承有两个方面：

(1)结构：这表明一个类，如 Non－ProfitOrganization，它是 Company 的一个子类的实例，将从 Company 继承域，如 Name，Address 等的值。

(2)行为：类 Company 的方法，如 AddSubsidiary ，EvaluateBudge 等，子类 Non－ProfitOrganization 和 CommercialCompany 将继承它们。这说明我们可以使用方法名 AddSubsidiary 向 Non－ProfitOrganization 的一个实例 NPO 发送消息，并以 NPO 作为目标对象执行类 Company 中的方法 AddSubsidiary。

综上所述，类是对一组对象的抽象，它将这些对象所具有的共同特征(包括操作特征和存贮特征)集中起来，由该种对象所共享。从系统构成的角度来看，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。

1.6 继承性

继承性是自动地共享类、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自封闭的。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。继承性是实现从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。

继承是一种非常自然和有力的组织信息的方法。前面已经简单地介绍了继承的概念，下面从继承域、继承方法和多重继承三个方面进一步阐述继承性。

1. 继承域

一个对象的类通过定义对象的域描述对象的内部结构。一个子类的实例必须和它的父类的实例一样保存有同样类型的信息，子类可以继承父类的域，也可以定义自己所特有的域。

由此可见，一个类有两种类型的使用方法：

(1)建立类的实例，并通过与该类有关的方法处理这个实例 —— 称作实例化使用。

(2)类的子类继承父类的方法(行为)和表示 —— 称作继承使用。