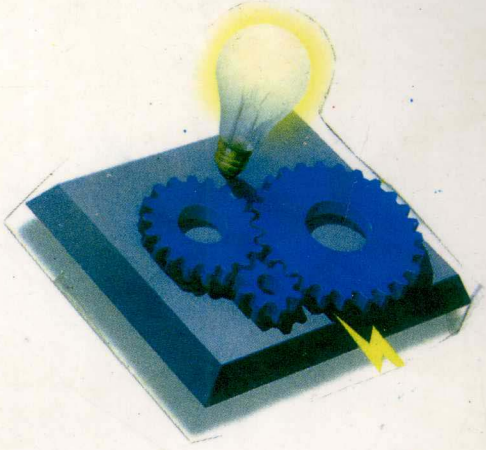
BORLAND C++

BORLAND C++

计算机程序设计语言系列丛书

田志良
杨秀国
丁志强
王丽珍 编著

面向对象程序设计循序渐进



学苑出版社

BORLAND C++面向对象
程序设计循序渐进

田志良
杨秀国
丁志强
王丽珍 编著
熊可宜 审校

学苑出版社
1994

(京)新登字 151 号

内 容 简 介

面向对象程序设计是近年来迅速发展的一个研究领域，被认为是程序设计方法学方面的一场实质性革命，它对软件工程学、认知科学、系统工程、人工智能和信息科学等学科都有重要意义。本书共有九章，采用循序渐进，深入浅出的方法以大量上机通过的示例来阐述 C++面向对象的基本概念和主要特征，并以典型应用范例为例来论述面向对象的分析和设计方法。具有易学易懂的特点。本书针对学习 C++程序设计和软件开发的广大初级和中级读者，适宜作为有关课程的入门教程与参考资料和培训班教材，以及从事计算机开发应用的广大科技人员的参考书。

欲购本书的用户，请直接与 8721 信箱联系，邮编：100080，电话：2562329。

计算机程序设计语言系列丛书

BORLAND C++面向对象程序设计循序渐进

编　　著：田志良　杨秀国　丁志强　王丽珍
审　　校：熊可宜
责任编辑：徐建军
出版发行：学苑出版社　　**邮政编码**：100032
社　　址：北京市西城区成方街 33 号
印　　刷：北京市朝阳区小红门印刷厂印刷
开　　本：787×1092　1/16
印　　张：8.25　　**字数**：186 千字
印　　数：1～5000 册
版　　次：1994 年 1 月北京第 1 版第 1 次
ISBN 7－5077－0807－1/TP・18
本册定价：10.00 元

前　言

面向对象是一种认识方法学，它既提供了从一般到特殊的演绎手段（如继承），又提供了从特殊到一般的归纳形式（如类等）。面向对象也是一种程序设计方法学。它基于信息隐藏和抽象数据类型等概念，把系统中所有资源都视为“对象”，每个对象封装数据和方法，而方法实施对数据的处理。由于面向对象技术的目标是使程序员在计算机上解决问题的方式更加类似于人类的活动，也就是使我们分析，设计和实现系统的方法学同我们认识客观世界的过程尽可能一致，因而引起计算机界极大的兴趣和关注。

当前计算机正朝着分布式过程、并行处理、网络化和软件生产工程化的方向发展，而面向对象方法学作为实现这些目标的关键技术之一，是同计算机领域的总体发展相一致的，因此被人们称之为主宰 90 年代的软件开发方法，是程序设计方面的一场实质性革命。

本书不在于贪大求全，面面俱到，旨在循序渐进，深入浅出地阐述有关面向对象的基本概念和主要特征，为读者在今后的编程实践中充分应用这些特性打下坚实基础。其次，本书列举大量示例，通过这些示例来讲解面向对象的概念和特征，具有易懂易学易于掌握的特点；第三，每章都按照什么叫类（对象，继承・・・），为何引入类（对象，继承・・・），怎样设计类（对象，继承・・・），引入类（对象，继承・・・）的益处是什么和应用实例这样的模式进行介绍。使读者知其然又知其所以然，贵在学以致用；第四，本书还叙述了面向对象的分析和设计方法并与传统的 SP 方法进行比较。特别是本书的最后一章给出了一个应用实例——用 OOP 技术实现嵌入式汉字系统，读者仔细阅读后将对怎样用 OOP 技术开发软件有所受益。

一九九四年一月

目　录

第一章 概 述

通过本章的学习，你将会了解：

· 面向对象的程序设计方法 OOP 与传统的结构化程序设计 SP 的主要区别是什么，以及什么是面向对象的程序设计。

· OOP 技术的几个基本概念

· OOP 技术的若干主要特征

从而为今后进一步的学习打下基础。

1.1 什么是 OOP

要弄清什么是面向对象的程序设计，首先应了解和回顾一下传统的结构化程序设计方法及其设计思想，程序结构与特点，以便比较对照二者之间的差异并领会 OOP 确是 SP 的更高层次的继承、丰富、发展和突破。

1.1.1 什么是 SP

SP(Structure Programming)是 60 年代诞生的、针对当时爆发的所谓“软件危机”的挑战、而在 70 年代——80 年代遍及全球成为所有软件开发设计领域、每个程序员都广为采用和熟知的传统的结构化程序设计方法与技术之简称。它的产生和发展形成了现代软件工程学的基础。

SP 的总的设计思路是：

· 自顶向下，层次化

· 逐步求精，精细化

其程序结构是按功能划分基本模块为树型结构，使模块间的关系尽可能简单、独立，并从而可单独验证模块的正确性。每一模块均由顺序、选择和循环三种基本结构组合而成。综言之，此即所谓“模块化”。参见说明实例图示 1.1：

因此，SP 的程序的基本特点是：

· 按层次组织模块(战略上划分战役)

· 每一模块只有一个入口，一个出口(每一战役尽可能简单、明确)

· 代码和数据分离(战术实现上程序＝数据结构＋算法)

SP 实现上述战略战术的具体方法和技术是：使用局部变量与子程序。

SP 的优点可以概括为：子程序对程序其它部份没有或尽可能少有副作用，从而为共享代码奠定基础。由于 SP 方法的模块化分解与功能抽象，自顶向下，分而治之的手段及技术路线从而能有效地将一个较复杂、中大规模的程序系统的设计任务分成许多易于控制、处理、可独立编程的子任务、子程序模块。对于这些子程序或模块中的每一个都有一个清晰的抽象界面，它只能说明——对应用程序设计者来说只需了解它做什么(What to do)而不必说明——了解它如何去做(How to do)。

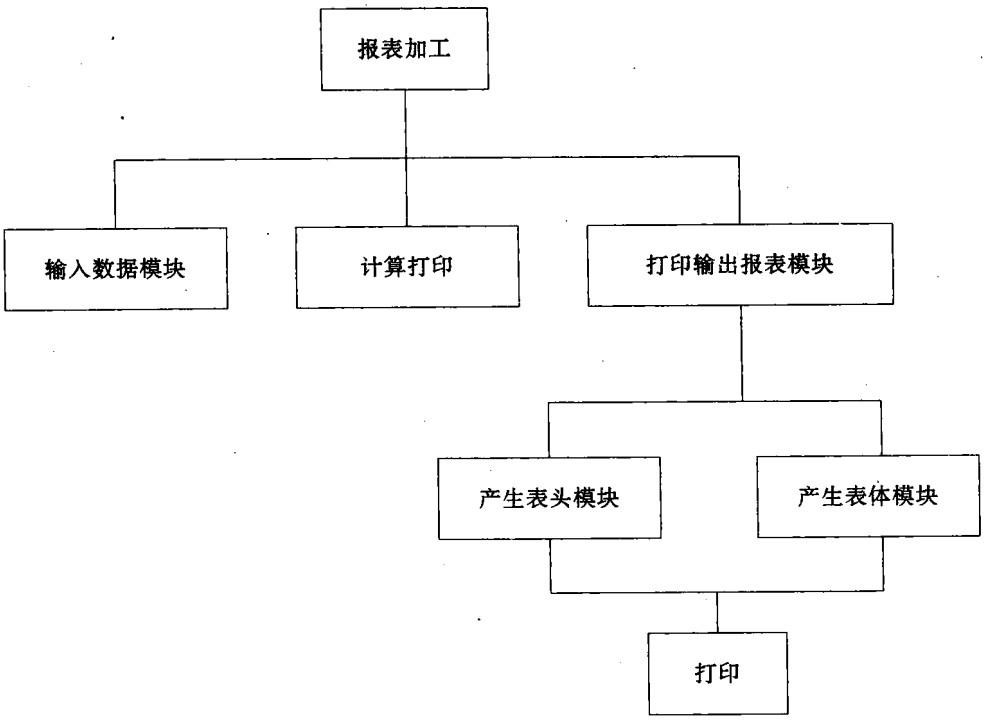


图1.1 报表加工处理程序的模块化

但从本质上说，由 PASCAL 和 C 这样的程序设计语言推动的传统的 SP 仍是一种面向数据/过程的设计方法，它把数据和过程分离为相互独立的实体，用数据代表问题空间中的客体借以表达实际问题中的信息；程序代码则体现/ 用于处理加工这些数据的算法。程序员在编程时，必须时时刻刻考虑所要处理的数据结构和类型，对不同的数据格式(结构和类型)即使要作同样的处理计算，或者对于相同的数据格式要作不同的处理都必须编写不同的程序，可见，使用传统 SP 方法设计出来的程序或系统，可重用的成分很少。另一方面，当把数据和代码作为不同的分离实体时，总存在着用错误的数据调用正确的程序模块，或用正确的数据调用了错误的程序模块的危险。因而，使数据与程序始终保持相容/一致，已成为程序员的一个沉重负担，这就是为什么在开发一个大型软件或应用系统的过程中，如果用户在工程后期对数据格式或实现方案提出任何改变请求(甚至如果负责设计数据结构或某一模块的人中途改变了某个数据结构或输入输出方案而又未及时通知其它开发人员)，那么，经常会发生下列景象："什么！要改变需求？天啊！请看看文挡，还有您的签字！这将需要大量时间重新设计和重新安排计划。"开发者气极败坏，对着用户继续说："若您们一定要如此，拿钱来！请再付重新开发费用！"变化摧垮了一切，前功尽弃！

1.1.2 什么是 OOP

为克服和解决当今许多大型软件工程项目和系统设计都接近或达到了 SP 方法所难以控制处理和适应其变化的上述种种矛盾及问题而产生的 OOP 方法与技术既吸取了 SP 的一切

优点和长处，同时又正视和顺应现实世界由物质和意识两部分组成，映射到面向对象的解空间就是：具体事物——对象和抽象概念——类；而一个对象无非就是这样一个实体，它具有一个名字标识，并带有自身的状态（属性）和自身的功能（行为）——世界上小至一粒米、大至一个宇宙的所有事物（对象）就是如此奇妙地简单！这正是面向对象方法和技术所追求的目标——将世界上的问题求解尽可能简单化。事实上，用计算机求解的问题都是现实世界中的问题，它们无非都由一些相互具有联系的，并处于不断运动变化的事物即对象所组成。因此每个具体对象都可用两个特征来把握，即：描述事物动态行为的可施于这些数据上的有限操作（功能模块）。也就是说，应当把数据结构和对数据进行的操作放在一起，作为一个相互依存不可分割的整体（用 OOP 的术语即“对象类”，下面括号中的术语亦然）来处理（“封装”、“信息隐蔽”、“数据抽象”…），并且要考虑不同事物即对象类之间的联系（“消息”、“通讯”、“协议”…）和事物即对象类的重用性，变化性（“继承”“多态性”…），这才更符合客观世界的本来面目。

概言之，OOP 与传统的忽略了数据与程序之间有不可分割内在联系的面向数据或面向过程的传统 SP 不同，它除了包纳 SP 的一切优特点与机制外，同时又是一个引入了若干强有力的、更能反映事物本质的新概念、新机制，从而开创了一个程序程序设计新天地的强大新方法技术。

OOP 的基本原理是，用问题领域的模型来模拟大千世界，从而设计出尽可能直接、自然地表示向题求解方法的软件，这样的软件系统由对象组成，而对象则是完整反映客观世界事物具有不可分割的静态属性（“数据结构”）与动态行为（“方法”）的，并且它们是既有联系，又有变化发展的实体。

OOP 方法所涉及的新概念，在理论上的最一般性（广义）的对应物及其描述其实是如此的简单和容易理解：

- 对象——对客观世界事物的表示或描述，世界上任何具体事物均可称为对象。
- 类——一组对象的抽象定义（“抽象概念”、“抽象数据类型”）。
- 方法——对应于对象的能力。
- 消息——客观世界中对象之间通讯的途径。
- 继承——对象间具有相同性（可重用部分）和差异变化的关系。

基于以上概念，面向对象就成了一个作为现实世界映射物的封闭缩微世界，一个对象具有自身的属性（“私有数据类型”）和可为自已或别人做的工作功能（“方法”、“操作”、或“成员函数”），它能通过发送消息与其它对象进行通讯，协同完成任务。这样，这个世界就活了。

正如著名计算机专家 Rentsch 于 1982 年就预言到的“我认为，80 年代问世的面向对象程序设计就象 70 年代的结构化程序一样，每个人都将会喜欢它，每个软件商都将开发他们的软件来支持它，每个管理人员都将付出代价来应用它，每个程序员都将以不同的方法来实践它，但，却没有人能精确地讲清楚它。”确实，OOP 并不是用一两句话就可以精确定义描述清楚的一套方法与技术（上面我们也仅只是对其基本思想与原则作了一些简单的论述），除非读完全书、除非循序渐进地深入理解了本书将要讲述的所有概念和内容后，你才会惊喜地说：“我终于搞清和会用 OOP 了！它确实是个好东西！！”

然而这里，目前暂时我们也只能非形式化地、笼统地这样告诉你：

OOP 是 SP，信息掩蔽，知识表示，并行处理领域等概念的继承和发展。OOP 的特点是把系统设计成将所需求解的问题分解成一些对象及对象间传递消息的符合客观世界规律的自然的

过程。OOP 方法使程序员摆脱具体的数据格式和过程的束缚，将精力集中到对要处理的对象的设计和研究上，从而大大减少了软件开发的复杂度。OOP 包括了功能抽象的抽象数据、信息隐蔽即封装等机理，使对象的内部实现与外界隔离，从而提供了更理想的模块化机制，大大减少了程序间的相互干扰和副作用。OOP 的抽象数据类型——对象类及继承，则为我们提供了理想的高可重用性的软件成份和机制。

此外，在人工智能领域中，若用 OOP 方法表示知识，则更接近于自然的客观世界的知识表示和认识论，因而不仅能表达描述非常复杂的客观事物和知识，而且具有模块性强、结构化程度高，便于分层实现，有利于实际开发及维护。因此 OOP 方法和技术的优特点将非常适合知识处理、知识库、专家系统等人工智能领域和数据库、CAD、图形处理（多媒体技术）、系统模拟与构造等大型复杂软件工程化开发。

1.2 OOP 技术的基本概念

C++语言及环境中的 OOP 有五个基本概念，这五个基本概念术语是构成整个 C++ OOP 方法与技术的柱石。它们是：

- 对象（object）
- 消息（Message）
- 方法（Method）
- 类（class）
- 继承（Inheritence）

下面我们将分别介绍它们的含义（定义）、组成和性质。

1.2.1 对象

什么是对象？对象是私有数据及可以对这些数据施加的操作结合在一起所构成的独立实体。

在 1.1.1 节中我们已经说过，从传统的 SP 观点来看，数据和处理它们的代码（操作过程）是两个不同的独立实体，它们之间的正确联系、选择与匹配需要应用系统的设计者时刻考虑、操心和进行统一。而在 OOP 中，一对象则是由私有数据和其上的一组操作代码组成的一个统一体（参见图 1.2）。对象的动作取决于发送给该对象的消息表达式，消息告诉对象要求完成的功能（What to do），并激活该功能，这意味着对象具有自动“知道”如何完成相应操作代码（how to do）的“智能”选择机制，正是这一选择机制把 SP 中应用系统程序员或用户作出的选择操作数与相应操作函数代码匹配的负担转移给了系统设计员，正是这一与传统 SP 风格有本质区别的对消息请求自动选择操作的小小变化，却蕴函着 OOP 技术的全部威力。

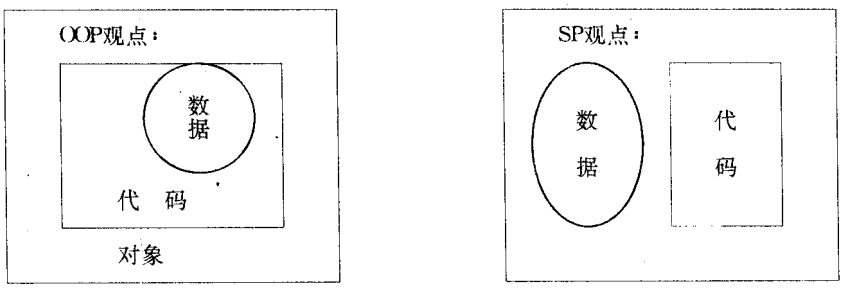


图 1.2 SP 与 OOP 代码和数据的关系——
SP 中的数据和代码是两个分离的独立实体
OOP 中的数据和代码被集中统一为一实体如同“软件集成块”

上面所谓的“智能化”的选择机制其实并不神秘，只不过是在对象的操作(过程或函数)表中对消息操作名进行按名搜索而已，对应 C++来说，执行一个消息表达式就是触发选择机制在相应对象的调度表(内存控制块)中进行搜索，找到后便转向对应于该消息的成员函数代码中运行。事实上，除了上述选择匹配机制是新东西外，下面这些术语的含义和传统的 SP 相应名词及说明是完全对应的：

OOP：		SP：
对象	←→	内存中的数据块(数据和代码共存的内存区域模块)
对象标识符	←→	指向上述数据的指针
向对象发送消息	←→	使成员函数作用于数据

在 OOP 中，数据和代码是紧密结合在一起的计算机中的一内存分块。但局部于该对象中的数据只可以由对象中的代码块来访问，即数据是对象私有的，不能被其它对象影响和修改。“私有的”同义语即“受保护的”。简单地说，一个对象从不关心其它对象的内部细节，而只关心它自己。对象之间的通讯仅只是以“做什么”的消息发送为契机的，并且认为接受消息的对象是知道如何去做的。若将对象间的对话拟人化的话，那便是“不要问我怎么做，我只告诉你需要做什么，你是做这类事的专家，你自己做就是了，我决不干涉你的内部事务(私事)。”

对象可以非常简单，也可以十分复杂。通常复杂对象由简单的对象层层嵌套组合而成。

今后我们将从 C++程序实例中看到，对象从形式上只是系统程序员、应用程序员或用户所定义的类这种类型的变量，当用户定义了一个对象，就创造出了一种有如此丰富内涵的新的抽象数据类型。这内涵丰富到就象在宿主计算机上拥有数据+ 代码并可相互通讯、被外壳封装保护，互不干扰破坏的具有若干功能行为的一台较小计算机(有些资料称之为“软件集成块”或“软插件”——Software_IC)。因此，对象可以说是构成和支撑整个 OOP 最重要的细胞与基石。除上述基本组成、关系、机理特性外，总之，对象还有以下性质和优点：

△ **模块独立性**：逻辑上看，一个对象是独立存在的模块，从外部看这模块，只需了解它具有哪些功能，至于它如何实现这些功能和使用哪些局部数据来完成它们的细节则“隐蔽”在模块内部。这意味着模块内部状态不受外界干扰改变，也不会殃及到其它模块。即模块间依赖性极小或几乎没有；各模块可独立为系统所组合选用，也可被程序员重用，而不必担心波及或破坏别的模块。

△ **动态连接性**：客观世界中各式各样的对象，并不是孤立存在的，正是它们之间的相互作用、联系和连接，才构成了世间各种不同的系统。同时，在OOP中，通过消息(传送)激活机制，把对象之间动态联系连接起来，便称为对象的动态连接性，它使整个机体运转起来了，这个世界变活了。

△ **易维护性**：由于对象的功能被"隐蔽"和好象被一层封装壳保护在对象内部，故无论是修改，还是完善功能以及实现的细节都被囿于该对象内部，不会播及到外部去，这就增加了对象和整个系统的易维护性。

1.2.2 消息

什么是消息？消息是要求某个对象执行其中某个功能操作的规格说明。

更通俗地讲，OOP中的术语"消息"只不过是现实世界中的"请求"、"命令"等日常生活用语的同义词。

通常，OOP的一条消息由消息选择器("消息操作"或"消息名")及若干变元(0个或多个实参)和接受消息的对象三部份组成。例如：

消息	对象	响应
at:1 put:H	"hello"	["Hello"]
print:GOOD MORNING!	disktop	在disktop输出设备上输出GOOD MORNING!

上例消息的选择器"at: put:"具有两个变元："1"和"H"。

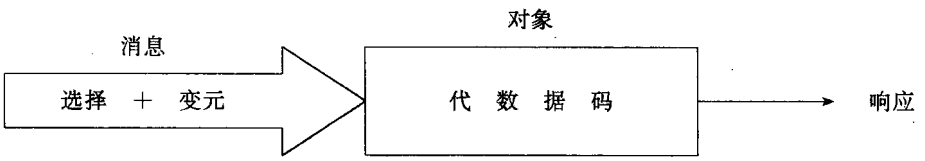


图1.3 消息激活对象自动完成请求功能

从程序设计语言的角度上来说，除前面曾经提到过的消息具备提供(激活)对象如1.2.1中所述的动态选择机制外，消息表达式从形式上看类似于普通的函数调用。如两者都带有变元，并返回值(响应结果)，调用时其环境都被压栈保护，执行时，调用者被挂起，在调用完成后再恢复运行环境等等，但二者仍在以下三个方面有所区别：

1. 函数调用可以带或不带参量，但消息至少带一个参量；它指明接收该消息的对象。如上例中的"hello"或"disktop"。消息选择器则负责告诉该对象应做些什么。

2. 消息选择器或消息名(消息操作)类似于函数名，但二者之间的本质差别在于："函数名仅只代表一段可执行的代码，而消息操作名的具体功能实现的选定取决于接收消息的对象。

3. 函数调用是过程式(面向过程)的即"how to do"，而消息则是说明式的，即"What to do"，具体"如何做(how to do)"由对象根据接收到的消息自行确定。

又如一个开方函数sqrt唯一标识了对一浮点数类型的自变量进行开方运行的代码，调用者必须保证自变量为正浮点数类型。但当sqrt为一消息操作名时，只能表示是对一个量的开方功能，而具体是哪一种类型的量还取决于该消息的受体对象，即具体执行哪一段开方处理的相应操作代码，由对象(消息接收者)自行确定，与用户(消息发送者或调用者)无关。在后面1.

4 中，我们还会看到更具体的 C++中用消息激活(驱动)对象的例子，而进一步理解领会消息和对象之间的依存关系。

一般，我们把发送消息的对象称为发送者，接收消息的对象称接收者(在 SP 中则相应于调用者和被调用者)。对象间的联系，只能通过传送消息即执行消息表达式来进行。对象也只有在收到消息时，才被激活，被激活后的对象代码将"知道"如何去操作它的私有数据完成所发送的消息要求的功能。

此外，消息还有以下三个性质：

1. 同一对象可接收不同型式内容的多个消息，产生不同的响应。例如：

对象	消息	响应
7	+1	〔8〕
7	edcdicHexValue	〔F7〕

2. 相同形式的消息可以送给不同对象，所做出的响应是截然不同的：

消息	对象	响应
at:1 put:N	"name"	["Name"]
at:2 put:23	(1045)	[(12345)]

3. 消息发送不考虑具体接受者，对象可以响应，但也可以不响应。即这种响应不是必须的，这与子程序调用必须要返回主程序也有所不同。

1.2.3 方法

什么是方法？OOP 中的术语"方法"对应于对象的能力，即它是实现对象所具有的功能操作的代码段。是响应消息的"方法"。

C++中，方法即是类中定义的成员函数，它只不过是该类对象所能执行的操作的算法实现。

方法与消息是一一对应的，每当对象收到一个消息，它除了能用其"智能化"的选择机制知道和决定应该去做什么(What to do)外，还要知道和决定该怎样做(How to do)。而方法正是与对象相连决定怎么做的操作执行代码。所以方法是实现每条消息具体功能的手段。

每个类中一般包含若干个方法，每个方法(即 C++ 的成员函数)由方法名(函数名+参数表)和说明该成员函数的算法实现的一段代码所组成。除了它必须被包含在类中并紧紧依附于类(动态运行时即紧紧依附于该对象)以外，形式上，它与一般的函数说明没有什么两样。

类中的方法即成员函数用以说明如何实施对应消息所要求的操作，当某类的实例即一具体对象接到了它所能接受的消息(在 C++中即成员函数调用表达式)后，它就会将控制转到以该类作样板模块(模板)而产生的实例对象的内存拷贝块中的相应功能代码段去执行，简单的说，即调用相应的方法(成员函数)去完成消息所传送发出的功能请求。

因此，例如一个消息若要求一窗口(一个类的对象实例)给出其中心位置，则相应的类的定义中就应有如何通过取窗口对角线的中点求实现该消息的要求的方法(成员函数)说明。

1.2.4 类

什么是类？类是对一组对象的抽象归纳概括，更确切地说，类是对一组具有相同数据成员和相同操作成员的对象的定义/说明。而每个对象都是某个类的一个具体实例。

在 OOP 中，每个对象由一个类来定义或说明，类可以看作生产具有相同属性和行为方式对象的模板。与类或分类就是“物以类聚，人以群分”的意思一样，“类”就是具有相似性质的事物的同类特征的集中。在面向对象系统中，我们一般就是根据对象的相似性（包括相似的存储特征——数据成员说明和相似的操作特征——成员函数说明）来组织类的。简言之，按照对象的相似性，我们把对象分成一些类和子类，故相似对象的集合即称为“类”。对 C++程序员而言，类实际只是一种对象类型。它描述属于该类型的具有相同结构和功能的对象的一般性质。而结构类型则只是将相关信息（仅只是不同类型的数据集）集中在一起的一个 C 中的变量集。

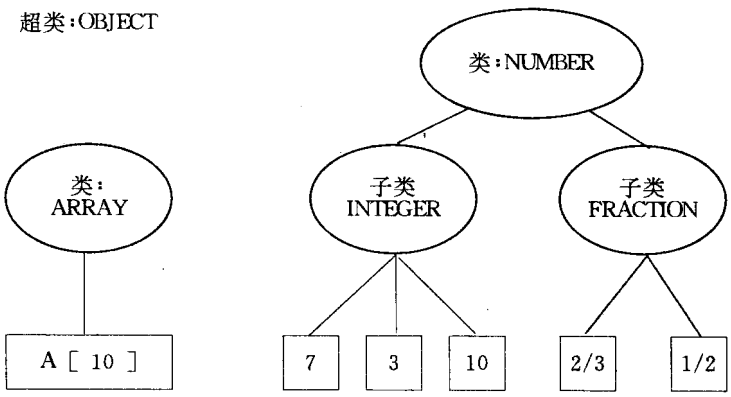


图1.4 对象的类和子类的区别

例如，如图 NUMBER 是一个类，它描述了所有整数的性质，包括整数的四则运算以及大小比较运算，"7"、"3"、"10"等具体的整数便都是 INTEGER 这个子类的对象。"2/3、"1/2"、"1/3"...则是子类 FRACTION 的对象。

类有上下层次之分。譬如所有的对象都属于超类 OBJECT。而上例中的类 NUMBER 则属于类 OBJECT，并且它有子类 INTEGER 和 FRACTION。

每个对象都是一个类或子类的变量，即该类的实例。例如"1/3"是子类 FRACTION 的一个实例。

从理论上讲，类(Class)是一组同类对象的抽象，它将该种对象的共同特征，包括操作特征和存储特征都集中封装起来。由属于此类的对象共享。例如，讨论食肉动物时，我们并不只是特指老虎和狮子，而只是抽象出它们的共同特性，比如喜欢吃肉。更具体地，又如，若为屏幕上的点设计了一个类：

```
Class point {
    x,y;                    // 坐标,私有数据
    new( );                 // 生成一个点
    delete( );              // 删除一个点      } 操作码
    move( );                // 移动点
    highlight( );           // 增强点的亮度
```

则可以说明 Point 类的两个对象 a,b 如下：

```
Point a(100,50), b(30,60);
```

Point 中定义的数据是抽象数据(类型),而对象 a,b 中的数据则是对应着内存中的具体数据单元拷贝块,以其初始化时的具体值和存储于不同单元块而区分不同的对象。这些具体的不同的点 a,b,... 便具有类 Point 中操作代码所描述的行为,诸如删除点、移动点、使该点为高亮度点等等。

1.2.5 继承

什么是继承?继承是对象类间的一种相关关系,并具有以下关键内容:

- 类间共享的特征(包括共享程序代码(操作)和数据)。
- 类间的细微区别或新增部份(包括非共享程序代码(操作)和数据)。
- 类间的层次结构。

继承机制既是一个对象类就获得另一对象类特征的过程,也是一个以分层分级结构组织、构造和重用类的工具。它是解决客观对象"相似但又不同"的妙法。

具体地,若类 B 继承类 A 时,则属于类 B 中的对象便具有类 A 的一切性质(数据属性)和功能(操作代码)。我们称被继承类 A 为基类或父类,而称继承类 B 为 A 的派生类或子类。因此,要描述或构造一个新类 B,只须去继承一个与之有共同特征的父类 A,再描述与父类不同的少量特征,即增加一些新的数据成员和成员函数即可。于是,类 B 便由承袭来的和新添加的两部份组成,其中继承部份来自于 A 及 A 的先辈类,增加部份则是专为类 B 所编写的数据或功能代码。

继承机制有以下优特点:

1. 能清晰体现相似类间的层次结构关系
2. 能减小代码和数据的重复冗余度,大大增强程序的重用性
3. 能通过增强一致性来减少模块间的接口和界面,大大增强程序的易维护性。
4. 继承是能自动传播代码的有力工具。
5. 继承还是在一些比较一般的类的基础上构造、建立和扩充新类的最有效的手段。

如果没有继承概念的支持,则 OOP 中所有的类就象一盘各自为政、彼此独立的散沙,每次软件开发都要从"一无所有"开始。

C++中,除一般 OOP 方法的树型层次结构的单一继承外,还支持多重继承。对单一继承,每个子类只能有一个父类(基类),而对多继承,每个子类则可以有多个父类(是网络状层次结构)。如图 1.5 所示。

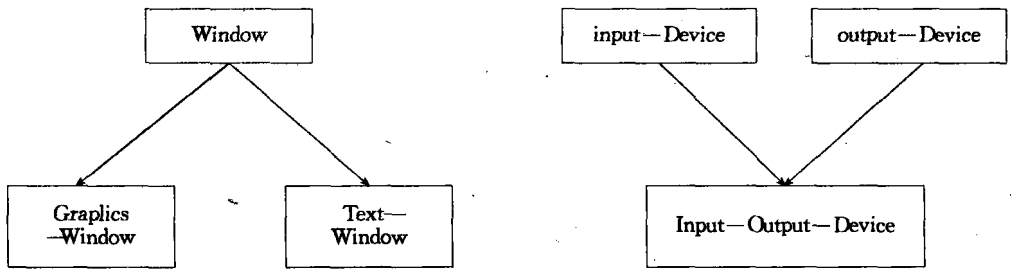


图1.5 单一继承和多重继承

1.3 OOP技术的特征

1.3.1 封装性

OOP中的封装性是一种信息掩蔽技术，它使系统设计员能够清晰地标明他们所提供的服务界面，用户和应用程序员则只看得见对象提供的操作功能(即封装面上的信息)，看不到其中的数据或操作代码细节。从用户或应用程序员的角度讲，对象提供了一组服务，而服务的具体实现即对象的内部却对用户屏蔽。如图示1.6：

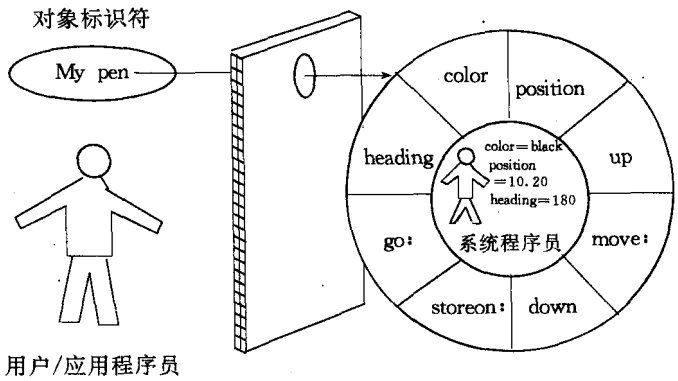


图1.6 从用户和系统设计员看封装(用户只能看见对象提供的服务，而看不到服务的具体实现，只有系统设计员才能看见封闭在对象中数据及其过程。)

对象的这一封装机制的目的在于将对象的设计者分开，使用者不必知道对象行为实现的细节，只须用设计者提供的消息命令对象去做就是了。

总之，也可以这样来定义封装：

- 一个清楚的边界，对象的所有私有数据、内部程序即方法(成员函数)细节被圈定在这个边界内。
- 一个接口，这个接口描述了对象之间的相互作用，请求和响应，它就是消息。
- 对象内部实现代码受到封装壳的保护，其它对象不能直接修改本对象拥有的(私有)数据和代码，只有通过本对象类的内部代码来修改。这就是OOP中对象的封装性。它避免了程序过程(函数)间的相互干扰。否则，仅仅由于重名引起的冲突就可能导致用错误的函数去处理正确的数据，或用正确的函数去处理错误的数据甚至二者兼错。这对于多人协作来设计一个大型软件项目更是如此。

1.3.2 多态性

OOP支持多态性，它指的是相同的函数调用为不同的对象接收时，可导致不同的行为。利用多态性，我们可以把函数的不同的实现细节留给接受函数调用的对象，而程序中则用同一函数名进行一般形式的调用。例如，消息函数调用Print( )被发送到一图形对象时和将其发送到一正文对象时的结果肯定不一样。

又如，假定现有三个虚函数：

abs( ) 返回整数绝对值
labs( ) 返回 long 的绝对值
fabs( ) 返回 double 的绝对值

——尽管这三个函数执行几乎相同的操作。

在 SP 中，须用三个不同的函数说明及调用来表示和使用它们，即大同小异的函数也必须有不同的函数名。于是，虽然仅只是一个求绝对值的运算行为，程序员必须至少记住三件事(三个函数)。在 C++中则改变了这一传统规定，可以用一个名字来表示和访问这三个函数而不会出现混乱，即如：

```
#Include <Iostream.h>
  Int abs (int i) { Return i< 0 ? -i:i;}
  double abs (double d) { Return d< 0 ? -d:d;}
  long abs (long l) { Return l< 0 ? -l:l;}
main( )
  { cout << abs(-10) <<"\n";
    cout << abs(-11.0) <<"\n";
    cout << abs(-9L) <<"\n";
   }
```

它们是不同的函数，却使用同一名字 abs，当程序中调用 abs( )时，OOP 编译器会根据自变量类型自动录找正确的那个函数进行运算，这就将要记住三件事减少为只须记住一件事。从应用的角度看，就好象一个函数支持三种不同的运算(功能)一样，这就是函数的多态性。

1.3.3 动态连接

连接是把例程地址送给例程调用者的过程，如：

Call ABC(1,2,3) ← PROC ABC(x,y,z)

在运行前的编译时便以目标代码的形式与系统完成连接称为静态连接，而在运行时进行上述连接的则称为动态连接。

传统的 PASCAL 与 C 语言仅支持静态连接，而面向对象的程序设计语言 Smalltalk、Objective-C 和 C++都在其实现中使用了动态编译技术，因而允许以目标代码的形式在运行过程中才与系统进行连接。这使得系统的灵活性和方便性大大增强。

因此我们常说，OOP 在这一方面(指以动态连接取代静态连接)比传统的 SP 有明显的优势。

1.3.4 消息驱动

OOP 中的消息驱动(激活)机制不同于 SP 中的子程序调用，这是因为，子程序调用者与被调调用者有控制与被控制关系；并且凡调用必有返回。而消息驱动则是当一个对象想激活另一个对象的某个能力时，它把请求(命令)以消息的形式传送给接受者就算了事，至于接受对象如何行动，如何处理该消息，完全是接受者自行决定的私事，其行为既不受发送者控制，也不一定有求必应和必有返回。其优点和灵活性是：不一定要知道消息来自何方，接受对象收到此消