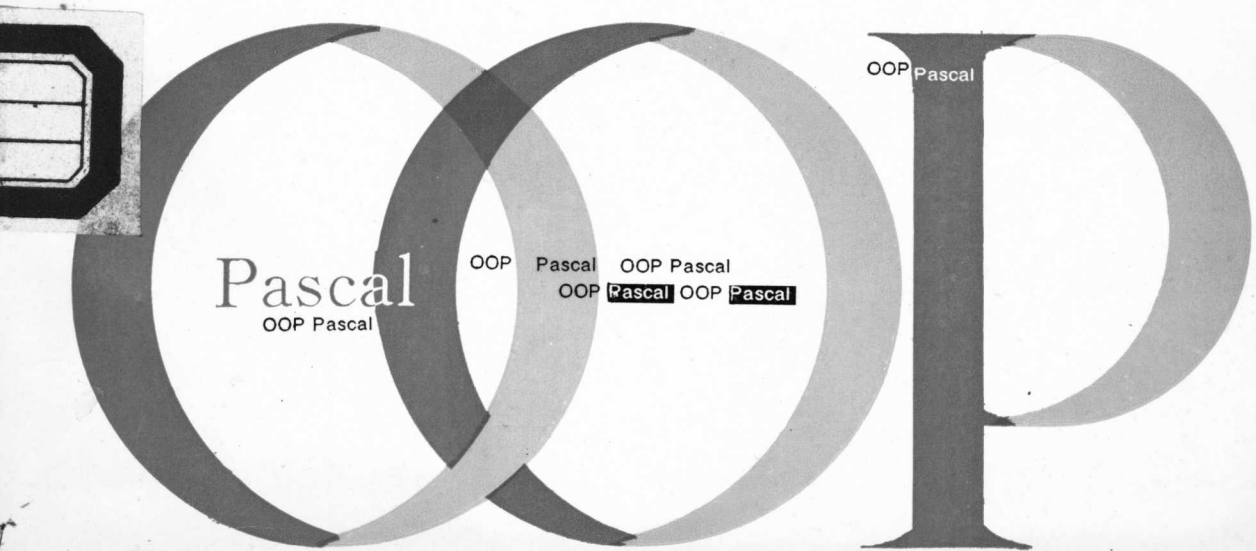
OOP Pascal

高级图形用户界面编程指南及技巧

金罗军 [比]Rijsman Bruno W A 邹红



OOP Pascal
高级图形用户界面编程指南和技巧

金罗军
[比] Rijsman Bruno W A
邹 红

华中理工大学出版社

内容简介

本书由三部分组成：第一部分基础篇详细地介绍了面向对象的程序设计方法，这种方法在国外被誉为“九十年代的程序设计方法”；第二部分指南篇，探讨了面向对象的程序设计方法在设计高级图形用户界面程序过程中，必须注意的事项和方法；第三部分技巧篇给出了有关图形界面设计的常用技巧。

本书内容新颖，实用性强，适用面广，书中所有的源程序清单均附有详细的说明，以便于读者理解和阅读。本书可作为高等学校计算机专业的教学用书。

本书配有软盘一张。

OOP　Pascal

高级图形用户界面编程指南和技巧

金罗军

[比] Rijsman Bruno W A

邹　红

责任编辑　李凤英

*

华中理工大学出版社出版发行

（武昌喻家山）

新华书店湖北发行所经销

石首市第二印刷厂印刷

*

开本：787×1092　1/16　印张：14.75　字数：326 000

1993 年 12 月第 1 版　1993 年 12 月第 1 次印刷

印数：1—2000

ISBN7-5609-0883-7/TP・103

定价：4.25 元

（鄂）新登字第 10 号

前　言

面向对象程序设计技术(OOP)是当今软件工程程序设计的潮流。正如70年代模块化程序设计广泛流传一样，面向对象程序设计技术正逐渐在世界范围内流行开来，在国外被誉为“90年代的程序设计方法”。为此，Turbo Pascal在原先的基础上增加了支持对象的功能，以适应程序设计风格的新潮流。

我们在使用OOP Pascal进行高级图形用户界面软件系统的开发过程中，查阅了大量的资料，做了大量的实际工作，积累了一定的经验和教训，并深感OOP Pascal的功能强大。为此，我们觉得有必要把在实际工作中的体会公之于众，以便广大同仁在今后软件开发过程中有所受益。

本书共分三部分：第一部分是基础篇，阐述了OOP Pascal程序设计的基本概念、原理及其它一些软件设计基础知识；第二部分是指南篇，详细地介绍了OOP Pascal的编程原则，并给出了许多具体的实例；第三部分是技巧篇，用大量的编程实例，介绍了设计高级图形用户界面的编程技巧。

本书内容新颖，实用性强，对OOP Pascal程序设计的初学者和有一定编程经验的软件程序设计人员都有一定的参考和借鉴作用。

在本书的编写和出版过程中，我们得到了许多老师和同学的帮助，在此表示感谢。

本书的出版得到了华中理工大学出版社的大力支持，在此表示真诚的谢意。

尤其感谢裴先登教授、刘本喜副教授、赵永龙副教授、谢长生、胡亦鸣、吴海良、梅松、王立环等老师，他们为我们提供了良好的工作条件。感谢文兴林、谢佩杰、黄伟等同学；感谢艾里·加穆卡玛，OPIO·OPEDI等外国朋友。

我们在编写过程中，虽然作出了很大的努力，尽量使文笔通顺、概念清晰，但书中仍不免会有很多错误，恳请广大读者谅解并指正。

本书配有软盘，盘中除了附有书中给出的例子的程序清单外，还有一些相当实用、有趣的演示实例。需者请与华中理工大学出版社软件编辑室联系。

编者

1993年5月

目　录

第一部分　基础篇

第二部分 指南篇

第三部分 技巧篇

第一部分

基 础 篇

这一部分主要讲述 OOP 程序设计方法的基本概念、原理。同时，这一部分也是进入指南篇和技巧篇的基础，是 OOP 程序设计所不可缺少的。

主要内容

- 面向对象程序设计技术
- OOP Pascal 概要
- 面向对象程序的调试技术
- 程序设计要素

第一章　面向对象的程序设计

本章概要地阐述面向对象的程序设计方法(Object Oriented Programming)中的重要概念、方法和思想,读者可以很快地浏览一遍。通读完本书后再回过头来仔细推敲本章的内容,将会加深对面向对象程序设计方法的认识。

第一节　传统程序设计方法的局限性

一般的程序员都有这样的体会:当程序达到一定规模时,为了修改一个小的错误常常能引发出多个大的错误。问题就出在目前所采用的传统程序设计方法上。

现行的程序设计方法以 70 年代荷兰学者 Dijkstra 提出的"结构化程序设计"使用得最为广泛。结构化程序设计方法把面向机器代码的传统程序抽象为三种基本的程序结构,即顺序结构、选择结构和重复结构,并提出了自顶向下、逐步求精、模块化程序等设计原则。按这种规范构成的模块是高度功能性的,内聚力很强。但模块中的数据处于实现功能和从属地位。因此各模块与数据间的相关性就差,无论把数据分散在各个模块里还是作为全局量放在总控模块里,模块之间的粘合力都是比较大的。在顺序程序设计中,粘合力的大小还不是致命的问题,但如果各模块并发执行,则极易导致程序系统的混乱。

任何一个特定的应用问题,它的问题域基于现实世界,它的解域由软、硬件的组合来实现。H. Ledgard 提出了一种模型来描述一个典型的编程任务,如图 1.1.1 所示。

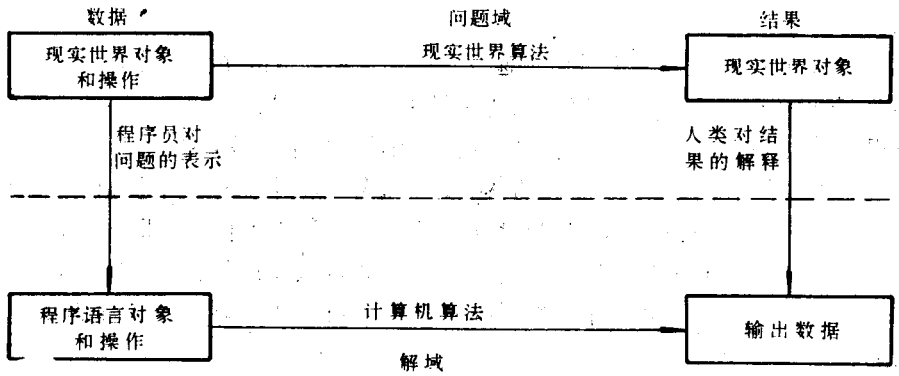


图 1.1.1　一个典型的编程任务的模型

该模型说明了在问题域中,有一个现实世界对象的集合,每一对象有一相应的操作集合。这些对象可能像存折、支票一样简单,也可能像宇宙飞船那样复杂。在问题域中,还有一些现实世界算法,对这些对象进行操作,并将其转换成目标对象。例如,一个现实世界结果可以是收支平衡的支票簿或宇宙飞船的导航波束等。

不管什么时候建立一个软件系统,要么完全用软件模拟一个现实世界问题,要么,将现实世界对象转换成软件和硬件,得到现实世界的结果。不论怎样实现,解域必须和问题域并行。

为了达到上述目标，首先要求程序语言为程序员提供表示现实世界对象的机制，实质上，是由程序员抽象问题域中的对象，并将这些抽象用软件实现。其次，还要求提供转换这些软件对象的计算机算法，同时，程序员必须抽象现实世界运算逻辑。最后，运行这些算法产生某种形式的输出，或者是产生某种物理动作，如控制表面运动。很明显，如果解域的映像与问题域的抽象越接近，我们就越易于达到下列目标：易修改、高效率、高可靠性和易于理解等。然而我们所知道的现实世界都是通过抽象而获得的，如果解域和问题域相差很远，那么就必须在现实世界的抽象转换上花费很多的脑力和体力，从而增加了解的复杂性。

研究表明，人类语言有两个基本组成元素：名词词组和动词词组。对程序语言而言，也是如此，即它们为实现对象（名词词组）的操作（动词词组）提供结构。目前大多数语言本质上都是命令型的，即它们有相当丰富的运算集，然而对现实世界对象的抽象却常常无能为力。同时，正如我们所知，这些语言的拓扑结构表明它们的结构都是相当扁平（FLAT）的，但是现实世界既不是平面也不是有序的，而是多维的和高度并发的。传统的程序语言尤其是汇编语言，加宽了问题域和解域之间的界限。而传统的程序设计方法如自顶向下结构化设计方法和数据结构设计方法，也不能很好地解决上述问题。

自顶向下设计技术本质上是强制性的，即迫使程序员把注意力集中在解域的操作上，而不考虑数据结构的设计。用这种方法设计的系统，数据被强制为全程的数据，同时，由于条件分支的出现错综复杂，使得系统可靠性不高。再出现数据的改变将影响整个结构（使整个结构波动），因此系统的灵活性也不好。数据结构设计则走向另一极端，它强调对象，而把操作看成是全程的。

采用上述设计方法，要么可以纯粹有一个功能化的解从而避免对现实世界对象抽象；要么可以有一个非常清晰的数据结构，但其操作运算却是相当模糊不清的，其结果就像是试图只通过动词或名词用英语交流一样。最起码，必须从问题域到解域进行抽象转换，在最坏的情形下，必须进行物理模型的转换。无论怎样，这些方法得到的解域与问题域都相隔遥遥，不能满足我们的需要。

在传统的程序设计中，程序是由传递参数的程序和函数的集合组成的，每个过程处理自己的参数，并可能返回值，因此传统的面向过程的程序设计是以过程为中心的。程序员被迫基于过程组织模块，这必然导致程序的结构与应用领域中的结构差异很大。

传统的程序设计方法的另一个大的缺点是围绕各种软件系统中关键结构的作用域与可见性。许多重要的函数和过程的实现主要地取决于关键的数据结构。如果一个或多个这样的数据结构发生了变化，这种变化将波及到许多方面，许多函数和过程必须重写。有时几个关键的数据结构发生了变化，将导致整个软件系统的崩溃。随着软件规模和复杂性的增长，这种缺陷日益明显。

此外，当代的软件应用领域正从传统的科学计算和事务处理扩展到了医疗、办公、人工智能、CAD/CAM 等方面，所需处理的数据也已从简单数字和字符串发展为记录在各种条件介质上并具有多种格式的多媒体数据，如正文、图形、影像、声音等等，因此数据量与数据类型空前激增，许多程序的规模和复杂性均接近或达到了用结构化程序设计方法无法管理的程度。

为了最大限度地利用已有的资源和减少新任务开发的工作量，需要有一种比以往程序开发方法抽象能力更强的新方法，在这样的背景下就诞生了面向对象的程序设计方法。

第二节　面向对象的程序设计

面向对象的程序设计的最根本目的就是使程序员更好地理解和管理庞大而复杂的程序，它在结构化程序设计的基础上完成进一步的抽象。这种在设计方法上更富层次的抽象正是目前软件开发的特点。

“面向对象”的程序设计源于 SmallTalk 语言，该语言是 1972 年美国 Xerox 公司 Palo Alto 研究中心为快速处理各种信息而在 Alto 个人计算机上研制的软件，于 1983 年正式发行 SmallTalk(V2.0)，SmallTalk 被认为是单纯的面向对象的语言。该语言只有一种数据类型——对象。但严格地讲，SmallTalk 只是一个程序设计环境，用环境来反映面向对象设计的原则。也许是因为非 Von Neumamn 机的效率问题限制了其表达能力(事实上 Alto 机的软件从来未出售过)，但 SmallTalk 所开创的面向对象的程序设计方法把结构化程序设计的抽象层次又增高了一层，对程序设计方法产生的影响是极大的。继 SmallTalk 之后，又相继诞生了许多面向对象的程序设计语言，如 Eiffel，Neon，C^{++}，Object-C 等，非面向对象程序设计语言如 Ada，Modula-2，Pascal 等都引进了部分面向对象的机制。

面向对象的程序设计绝非是要摒弃结构化程序设计方法，相反它是在吸收结构化程序设计的优点的基础上，引进了一些全新的强有力的概念，从而开创了程序设计工作的新天地。按照传统的程序设计方法，各个不同的编程者编制的程序由于缺乏可交换性和标准化，不仅效率低，可靠性差，而且程序的维护也相当困难。针对这一现状，面向对象的程序设计方法把重复使用性视为软件开发的中心问题，通过装配可重复使用软件来产生软件，而不是像目前编程方法那样通过调用函数库中的函数来产生软件。

理想情形是：将现实世界的抽象直接映像到解的体系结构中，并且和人类的语言一样尽量在现实模型中对对象和操作进行平衡。我们把这种方法称为面向对象的实现方法，这样命名是为了强调它既不是纯粹的强制性的方法，也不是说明性的方法。相反，它把软件对象的重要性看作是作用者(actor)，每一个都有自己的应用操作集合。

面向对象的实现与传统的方法有本质上的不同。传统方法分解一个系统的基本标准通常是系统中的每个模块代表整个操作的一个步骤。对面向对象而言其标准是：系统中的每个模块代表问题域中的一个对象或一类对象，抽象和信息隐藏组成了所有面向对象实现的基础。

鉴于对象在这种方法中的重要性，在此解释一下对象的意义。首先，一个对象是有状态的实体，即具有某个值，例如在一辆汽车的导航控制系统中，可把刹车、节流阀、加速器和发动机看作对象，它们都和现实模型相关。其次，一个特定对象的行为由它承受的和施加的动作来定义。同样在导航控制系统中，对节流阀的有意义的操作包括增加或降低其值，然后节流阀必须能作用于发动机，并改变其速度。最后，每个对象实际上是某个对象类的特例，例如，一个给定的导航控制系统——特定的发动机，这个发动机实际上是一个对象类的特例，该对象类中都是发动机。可以是六缸的或八缸的发动机，但它们都有相似的行为。这样做的好处是通过把具有共同属性的对象归于同一类，可以在此类中进行局部方法设计。

在任何情况下，都不可能对问题域有完美的认识。人们对事物的理解是一个反复上升的过程。当我们已进入解的深层次时，很可能发现有新的问题是以前没注意到的。但是，因为解直接映像到问题域，新发现的问题域一般并不影响解的体系结构。使用面向对象的方法，一般可以将改变的范围限制在和问题域中的对象并行的解域模型中。

因此，一个实现现实模型的程序可看作是相互作用的对象的集合。可采用以下步骤，使用面向对象方法设计一个系统：

1）标记对象和它们的属性；

2）标记影响对象的每个操作和每个对象必须初始化的操作；

3）建立每个对象和其它对象相关的清晰度；

4）建立每个对象的界面；

5）实现每个对象。

■ 标记对象

第一步，标记对象和它们的属性，包括识别问题域中的主要行为者、中介者和服务者，加上现实模型中的规则。通常，这一步标记的对象来自于描述问题空间的名词。我们也可能发现一些相似的对象。在这种情况下，必须建立很多实例的对象类。例如，可以揭示一些类型的名词词组：

1）普通名词命名一类实体（如桌子、终端、传感器）。

2）物质名词和度量单位命名共同具有的数量、活性、物质或质量（如水、事物、燃料）。

3）专属名词和直接参考名词命名特定的实例（如热传感器、我的桌子、紧急开关）。

普通名词，物质名词和度量单位，不标识特定的实例，但是标识一类对象，称其为抽象数据类域。

■ 标记操作

该标记步骤用来描述每个对象或对象类的行为。建立对象的语义学，使得对象的操作或对对象实施的操作具有某种意义。在这一步还通过标记对时间和空间的约束来建立动态行为。例如可以为操作规定一个时间顺序。

■ 建立可见度

当建立每个对象和其它对象相关的可见度时，我们标识对象之间和对象类之间的静态依赖性，即一个确定对象“看见的”和“未看见的”。这一步的目的是从现实模型中捕捉对象的拓扑结构。

■ 建立接口

建立每个对象的接口，就是用某种合适的形式符号表示法来产生一个模型的详细说明。这一步用来获取已建立的每个对象或对象类的静态语义。该说明也作为对象和它的“顾客”之间的合同。换句话说，接口形成对象的外视图和内视图之间的界限。

■ 实现每个对象

在第五步即最后一步，包括为每个对象或对象类选择一合适的表示，并实现以上步骤中的接口。这包括合成，分解，或两者均有。有时候，一个对象由一些辅助对象组成。在这种情况下，我们可反复运用该方法，进一步分解对象。多数时候，一个对象可通过合成来实现，在现有的低层对象或对象类上建立。一个系统的原型建立后，开发者可以选择将所有的对象的实现延期到以后。在这种情况下，开发者依赖于对象的详细描述（带有适当的粗线条实现），来实验

系统的体系结构和行为，类似地，开发者可以在对象生命期中用一些替换表示来测试各种实现的行为。

我们看看这种方法是怎样支持软件工作原理的。很明显，面向对象的实现方法支持抽象和信息隐含，因为该方法的基础是直接映像现实模型到解域。并且，用 OOP 设计语言，可以隐藏操作细节和对象的表示。

这种方法也提供了把一个系统分解为模块的有效策略，使用这个策略，只要求解域与现实世界匹配即可。并且，我们有了统一的符号表示来选择对象和操作。当然，和任何开发策略一样，必须依赖于某些管理工具来保证设计的完整性和确定性。但是有了面向对象方法提供的强有力的结构，这项工作简单多了。

小　结

1)传统的程序设计方法不能有效地实现抽象和信息隐含，它们通常不适应于有并发性的问题域，并且不能响应问题域中的变化。面向对象实现方法能缓和这些问题。

2)用面向对象方法，系统的体系结构围绕着对应于现实模型的对象和对象类的概念来 组织。

3)对象是具有状态的实体，特点在于它吸收和发起的操作，是对象类的一个实例。

第二章 OOP Pascal概要

面向对象的程序设计方法(OOP)与人们日常处理问题所采用的方法基本类似，它从早期的程序设计语言自然发展而来，比以前的结构化程序更为结构化，其数据抽象和信息隐藏更为抽象、模块化。面向对象的程序设计语言有以下几个特点：

1) 封装性：将记录与处理该记录的过程和函数组合成新的数据类型——对象。

2) 继承性：定义对象，然后使用对象构造子对象序列，子对象可以继承并访问其所有的父对象的代码和数据。

3) 多态性：为动作赋予一个可由对象序列共享的名称，对象序列中的各个对象均以适合自身的方式实现该操作。

Turbo Pascal 语言的扩展，已使得它具有面向对象程序设计的能力：更为结构化、更为模块化、更为抽象化，而且语言本身具有可再用性。所有这些使得代码的结构化程度有所提高，也更容易维护。

面向对象的程序设计方法要求程序员放弃多年养成的程序设计思路和习惯。相对于传统方法所难于解决的问题，OOP 提供了一种简单、直观而高级的工具，因此，这个牺牲还是值得的。

第一节 OOP 的基本概念

在面向对象的程序设计方法中，用户通过将任务分成许多易于管理的、与应用概念密切相关的对象来完成程序设计。

■ 类

用来描述具有相同结构和行为的对象群体，它描述了一个数据集及一个过程或函数的集合。因而类由以下两部分组成：

1)静态部分，即数据，通常称之为域，由名字及数值构成。域刻划了对象在运行时的状态。

2)动态部分，即过程，又称作方法，它表示此类所描述的对象的共同行为特征。方法处理对象域并且规定了能由对象完成的动作。

按上述模式，类定义由两部分组成：域的定义；方法的定义。

方法选择符一起组成方法表(Method Dictionary)，有一点很重要，必须要能明确区分各个选择者，即方法名和方法体(也即定义要完成操作的过程和函数)。事实上虽然我们可为同一个方法名定义不同的方法体，但那是一种特殊情形，大多数情况下，程序员必须保证不要让用户混淆它们。

■ 对象实例

类定义了用以产生其实际表示——对象实例的样板。因而，对象实例是根据类而生成的一个具体对象。类可以被认为是允许生成许多拷贝的模板。既然方法表示了对象实例的共同行

为特征，因而对类中的方法进行复制是毫无用处的。另外，如果类的对象实例都具有相同的域，但是这些域可以因为是不同的对象实例而具有不同的值，也就是说，虽然同一个类的对象实例的域都一样，但它们可以赋以不同的值。

对象和类是面向对象的程序设计语言中最基本且重要的概念。对象直接对应于应用概念，可用于表示各种信息，如数值、字符串、矩阵、堆栈等都可以是对象。

类是一个用户定义的类型。它给出了构造和使用某一特定的对象所必需的所有信息。类提供了数据隐藏、数据结构化、用户定义类型之间的隐含转换、用户控制存储管理、控制重载操作符等功能。

注意

对 Turbo Pascal，它将“类”称作“对象”，并称“对象”为“对象的实例”，这种命名有别于所有其它 OOP 语言和工具。

在面向对象的程序设计中，“类”就是用户产生对象的软件样板，类发展了数据的类型，它们不仅定义了决定对象状态的数据域，而且指定了对象的功能。它类似于 Pascal 中的数据类型。对象则类似于 Pascal 中的传统变量。

■ 继承性

继承性允许程序员创建一类对象后，继续创建其特例的对象。建立某一存在的特例类称为建立子类。新类是存在类的子类，而已存在的类称为新类的超类(或父类)。子类从其超类中继承所有的例式变量和方法。子类还可以增加例式变量、类变量及适用于特例对象的方法，此外子类可使超类的方法无效，也可在超类的方法中增加新的特性。当新方法使用与旧方法相同名称时，旧方法便失效了。

继承性可使程序员通过描述新类和已存在的类之间的差别来创建特例对象的新类而不必重抄相同的那部分代码。继承性机制提供了代码的共享，大大节省了程序设计的工作量，这对缩短应用软件的研制周期，无疑是十分有用的。

同时这也是一种有效的知识共享(Knowledge-sharing)机制。类应该当作是一个知识库，由它可以尽可能地定义其它的类。父类中的最一般的知识(东西)可逐步分解、细化，并由它的子类去定义。

故，一子类是对其父类(超类)的更加专门化的描述。同时子类分享其父类的变量和方法，很明显，应该尽量避免父子类之间无用的拷贝，但是从概念上讲似乎父类的所有信息都拷贝到了子类中，子类则继承父类的所有信息。

类的具体化有以下两种方式：

1)增加法，即子类增加新变量、新方法，用以描述其特有的属性。

2)替换法，当父类的方法不足以描述新类的特征时，可以替换原来的方法，对方法进 行重新定义。

■ 继承图

继承关系将一个类与其父类连接在一起，按照这种关系构成的类层次和动物世界的食物链有几分类似，我们用继承图(Inheritance Graph)来表示这种层次关系。图 1.2.1 便是这样的一个例子。

这种情形其实是棵树，根节类表示最一般的类，称为 Objects 类(大多数面向对象的语言

都是这样称呼),该类包含了所有类的共同部分,即如何打印一个对象,以及如何搜索一个对象类等,较高层次的类由基础类组成。

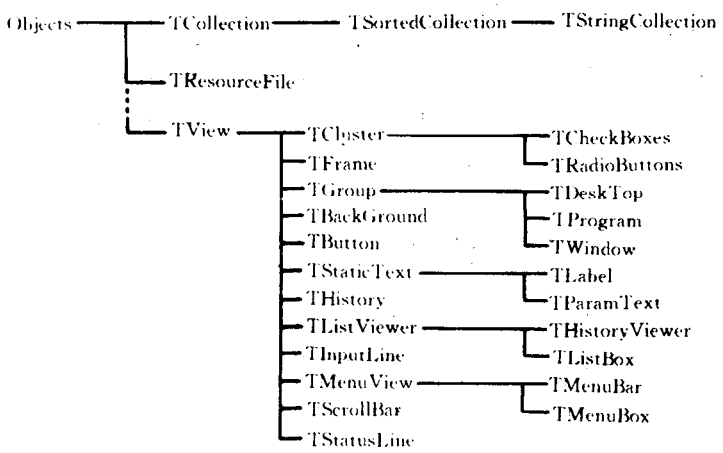


图 1.2.1 一个继承图实例

继承关系是可传递的:较低层次的类继承较高层次类的信息,离树叶越近则越具体化。按上述概念,术语超类(父类)代表了它所有子类的共同特性。

用类的观点重新构造世界能够实现真正的模块化,将世界分成一个个相互独立的由对象构成的模块,修改其中的任何一部分,对别的部分的影响都相当小。可以从继承图看出这一点。每棵子树类分享该树根节类的信息,对这个根节类进行修改,应该只影响到它的子类。

例如对 TMenuView 进行修改,只会影响到 TMenuBar 和 TMenuBox,而不会影响到 TView 及其它类。

■ 方法

方法实际上就是能定义的过程,类为它提供了存储的空间。通过发送消息即可调用方法完成预定的功能。方法存在于类中以节省空间,因为所有的某一类对象都有相同的方法集。

■ 信息隐藏

信息隐藏对保证系统的可靠性极其重要,它减少了软件之间的相互依赖关系,这对大型软件来说尤为重要。因为各软件模块(在面向对象的程序设计语言中实际上是对象)之间的依赖性愈少,其正确性就愈易保证,同时也使软件系统的局部的改动与其它部分无关。由于面向对象的程序设计,对象的状态是体现在其自身的私有存储单元中,只有该对象的操作程序才能访问这些私有存储单元,而程序的其它部分是不能对它们进行操作的,这就保证了软件系统的模块化和局部化。仔细设计的模块接口允许改变内部的数据结构和过程而不影响其它软件模块的实现。

注意

Turbo Pascal 并不提供数据保护手段,但这不妨碍信息隐藏的实现。我们将在 指南篇中详细介绍如何实现数据隐藏。