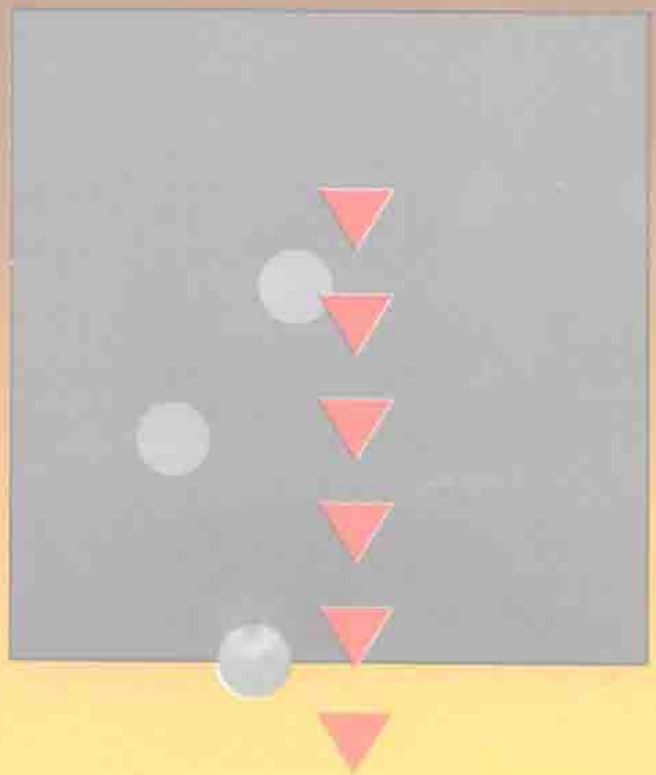
对象技术丛书

面向对象的应用开发环境



与

VisualAge C++

IBM

[美] M. 卡雷尔-比利亚德 等著

科学出版社

对象技术丛书

面向对象的应用开发环境与 VisualAge C++

〔美〕M.卡雷尔-比利亚德 等著

邵丹华 张波 译

科学出版社

1998

内容简介

本书分三部分。第一部分介绍了可视化程序设计和面向对象技术的概念和术语，并对 VisualAge C++开发环境作了初步介绍。第二部分引入了一个应用程序示例，为 VisualAge C++完成并实现应用，分析和设计了该应用程序的静态和动态模型。第三部分是本书主体，介绍如何使用 VisualAge C++以及各种工具完成应用程序的开发过程。

本书适用于软件开发人员和程序员，也适用作有关课程的教学参考书。

M. Carrel-Billiard, et al.

OBJECT-ORIENTED APPLICATION DEVELOPMENT WITH VISUALAGE FOR C++ FOR OS/2

Authorized translation from English language edition

Published by Prentice Hall PTR

图书在版编目（CIP）数据

面向对象的应用开发环境与 VisualAge C++/（美）卡雷尔-比利亚德，M.（Carrel-Billiard, M.）等著；邵丹华，张波译．-北京：科学出版社，1998.7

（对象技术丛书）

书名原文：Object-Oriented Application Development with VisualAge for C++ For OS/2

ISBN 7-03-006610-3

Ⅰ.面… Ⅱ.①卡… ②邵… ③张… Ⅲ.C 语言-程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（98）第 08143 号

图字：01-98-0556 号

科学出版社 出版

北京东黄城根北街 16 号

邮政编码：100717

北京双青印刷厂 印刷

新华书店北京发行所发行 各地新华书店经售

*

1998 年 8 月第 一 版 开本：787×1092 1/16

1998 年 8 月第一次印刷 印张：15 1/4

印数：1-3 000 字数：341 000

定价：28.00 元

《对象技术丛书》前言

国内有关面向对象语言和技术的书籍已经出版了不少。为什么还要再出这样一套丛书？道理很简单，这是社会的需要。既是高等学校进行软件工程教学和实践的需要，也是广大从事软件开发和应用的工程技术人员的需要。有人认为，能够用C++编写程序就是掌握了对象技术。这是一种误解。近代软件工程发展正面临社会传统的结构化范型到面向对象范型的转移，这需要有新的语言、新的系统和新的方法学的支持，对象技术就是这种新范型的核心技术。对象技术正发展成为当代乃至面向下一世纪软件工程发展的主流技术。只有真正深刻理解对象技术的内涵，才能在实践活动中运用自如，进入一个新的境界。本丛书旨在向国内读者全面和深刻地介绍面向对象技术，包括面向对象的语言、方法学、体系结构、技术标准和产品应用等。丛书部分专题由国内有关对象技术专家撰写，部分专题经IBM授权同意，由该公司的技术资料翻译而成。由于对象技术尚在蓬勃发展阶段，各种对象技术的新系统和新产品不断涌现。本丛书没有预先确定一套完整的书目清单，根据先求实不求全的原则，按照对象技术的发展和国内读者的需要，将陆续选定出版的专题。丛书编委会负责指导和组织专题的编写和翻译工作。

希望这套丛书能对我国从事对象技术教学和研究开发的科技人员有所帮助，并能为我国对象技术的应用和软件产业的进步起到推动作用。一本书、一套书的作用总是很有限的，但是它所激发的社会响应却可能很大。这正是我们所追求的。

《对象技术丛书》编委会

1997年12月

译者的话

随着软件技术的发展，面向对象的程序设计（Object-Oriented Programming）越来越得到广泛的应用，它为软件的发展带来了很多好处：可复用性、抽象性、封装性等。对于程序员来说，尽管很多高级语言都具有面向对象机制，但是，C++尤其受到青睐。这不仅是因为它充分支持面向对象中的抽象、封装、继承等机制，更因为它的基础——C语言受到了广泛的技术支持。

在软件开发中，简便实用的图形化用户界面和事件驱动的程序设计风格改变了原有的编程模式，因此，可视化程序设计成了必然的趋势。目前，市场上出现了一些可视化的C++程序设计工具，它们可以让程序员在屏幕上直接设计图形用户界面。然而，它们所能够做的也就到此为止了。对于开发中剩下的业务逻辑和数据访问事务处理必须由程序员手工完成。如果使用 VisualAge for C++，程序员就可以继续可视化地工作下去，它所提供的组件不仅可以建立图形用户界面，而且可以建立整个应用程序，这其中包括数据库的访问和添加多媒体功能。

在 VisualAge for C++中采用了 IBM 的 VisualAge Smalltalk 中的可视化程序设计构造技术，是 CSet++的后续产品。它所支持的组合技术能够可视化地创建和集成部件来构造复杂的应用程序。它提供了一个强有力的开发框架，用户可以通过鼠标选择应用程序所需要的可视部件，并把这些部件拖放到自由区域来布置应用程序的界面。使用这种办法就可以在建立业务逻辑和调用编译器之前看到这个应用程序的图形用户界面了。

同时，在 VisualAge for C++的使用过程中可以逐步学习和提高对象技术。从创建图形用户界面、仍旧调用已有的代码出发，逐渐地进行可视化编程，复用自己的图形用户界面，最后就可以使用它的更高级的功能来创建自己的部件了。

在本书中，介绍了一个真正的应用程序——Visual Realty。它支持房地产经纪人管理各种房地产信息和客户信息。作者通过一步一步地介绍这个例子的开发过程来细致地指导用户如何使用 VisualAge for C++。作者从初学者的角度出发，耐心细致地介绍了开发过程中的每一步，可以让用户详细完整地开发所有内容。

在本书的翻译、校对过程中得到了许多朋友的热情帮助，其中包括黄涛、倪彬、钱军、王栩、钟华、金蓓弘、周桓、吴晓斌、付远彬、丁向武、王劲松、孙红艳、李冬、虞海江、武小鹏、杜小尧、宛海鹏、葛鹏、李欣、贺志军、郭胜辉等，在此一一表示感谢。由于译者的水平有限，其中必有不当之处，恳请读者批评指正。

导　言

欢迎走进可视化程序设计世界。有了 VisualAge for C++，你就可以投身到一个全新的程序设计潮流中去。如果你刚刚得到 IBM VisualAge for C++，并且希望自己建立起第一个有意义的应用程序，那么，本书恰好适合于你。事实上，本书就是通过例子来指导你学习 VisualAge for C++的。有了 VisualAge for C++，应用程序的构造变得前所未有地容易。即使是最复杂的应用程序，也能够依靠在 IBM Open 类中已经预定义好的大量部件构造出来。本书告诉你如何运用 VisualAge for C++来实现一个已经使用面向对象方法完成了分析和设计工作的软件系统。本书介绍了一种面向对象的软件开发方法——可视化建模技术（Visual Modeling Technique），并且描述了如何运用这种方法来建立一个真正的应用程序，使它具备关系数据库的支持、视频和声音处理能力，以及用于真正直观的图形用户界面的大量图形控制。

本书的独特之处

本书详细介绍了开发一个应用程序的全过程：从需求描述一直到使用 VisualAge for C++进行编码。首先，本书将指导你使用可视化建模技术建立静态模型和动态模型。然后，使用可视编制器（Visual Builder）来可视化地翻译这些模型并且自动产生源代码。本书不是一本单纯介绍方法学的书，也不是一本单纯介绍程序设计的书，而是这两方面内容的结合！

本书第一次手把手地教你去完成应用程序的完整的开发过程。因此，作为一名分析员-设计员-开发者，请作好准备到“可视化程序设计”世界中去漫游吧！

本书的组织

本书由三部分组成。第一部分（第一，二章）介绍了与可视化程序设计和面向对象技术相关的概念和术语，并且对 VisualAge for C++开发环境作了初步介绍。第二部分（第三，四章）引入了一个应用程序示例。在此部分中，主要分析和设计了该应用程序的静态模型和动态模型，以便顺利地使用 VisualAge for C++来完成实现。第三部分（第五～十章）构成了本书的主体部分。这一部分将教你如何使用 VisualAge for C++以及各种工具从最底层开始来完成该应用程序的开发过程。

- □ 第一章，“VisualAge for C++和应用开发”
  第一章将引导你走进应用开发的可视化时代。由此，可以了解到近几年来出现的软件构造的新潮流，以及 VisualAge for C++是怎样迎接新的挑战的。
- □ 第二章，“VisualAge for C++环境入门”
  第二章介绍了 VisualAge for C++软件包中的所有工具以及它们的特色。这一章虽然并不是用户指南，但是介绍了一些关键性内容，以便使你能够应用这些工具进行应用开发。
- □ 第三章，“工作的分析员”

在这一章和下一章中，我们让你扮演一个小说家的角色。也就是说，把实现一个精巧的软件系统之前的分析和设计阶段比作在编写一本畅销书之前所作的准备性工作。本章集中介绍应用程序的分析阶段。

□ 第四章，“工作的设计员”

本章所强调的是设计阶段。

□ 第五章，“设置开发环境”

本章介绍的是为构造具有良好结构的软件所需要进行的一些准备性工作。它指导你如何在 WorkFrame/2 环境中正确地初始化你的新项目。

□ 第六章，“使用数据存取编制器（Data Access Builder）映射关系数据库”

本章以及后面的两章主要介绍可视编制器。在本书的编写过程中，我们非常高兴能与这个工具打交道。如果了解了我们是怎样使用这个工具来成功地实现了示例应用程序的时候，那么，你也会激动起来的。如果能够根据我们的指导，一步一步地重复这一实现过程，那么你就会得到最大的收益。在这一章中，你将通过数据存取编制器使你的应用程序具有持久性，并且能够把对象存储在关系数据库中。

□ 第七章，“创建可视部件”

本章将指导你运用 VisualAge for C++ 提供的可视部件来开发你的应用程序中的图形用户界面。我们的示例程序使用了系统提供的绝大部分部件，这样，你就会得到指导和提示以便更好地运用它们。

□ 第八章，“创建不可视部件”

与其他图形用户界面生成系统不同，Visual Builder 允许你将业务对象实现为非可视部件。在这一章中，我们将告诉你如何开发在示例应用程序中所需要的非可视部件。

□ 第九章，“连接部件”

一旦建立了可视部件和非可视部件，那么就可以在它们之间作出图形化的连接了。在这一章，我们将介绍怎样连接不同的部件，激发从一个对象到另一个对象的消息传递，这样，应用程序就可以运转起来了。然后，让系统去自动生成这个应用程序的 C++ 源代码并编译它们！在以上这三章中，我们所要介绍的是如何把详细设计中的静态模型和动态模型映射到 VisualAge for C++ 中去。

□ 第十章，“如果希望对可视编制器了解得更多……”

如果好奇心还没有得到满足或如果想知道更多的技术细节，那么你应该继续读下去。这一章回答了一些以前没有问到的问题，比如：通知框架是什么？能复用原有的程序代码吗？

目　录

第一部分　VisualAge for C++环境介绍

第二部分　使用 VisualAge for C++开发应用程序

第三部分　构建 Visual Realty 应用

第 一 部 分
VisualAge for C++环境介绍

过去的那些美好的日子哪儿去了？那时，计算机供应商不仅提供了大型机和相应的操作系统，而且还提供了相应的软件工具，以扩充系统的基本配置；销售商们只需为他们所喜爱的二三个客户提供支持，过着舒适的日子；应用程序的开发者把主要精力集中在数据库事务以及业务逻辑的实现上，而不必去关心用户界面(这是因为终端就像打字机，而那些提供输入的倒霉的人们就像系统的外部设备一样，与应用程序紧密地绑在一起)；程序员是软件的权威，他们能容忍编辑器中成千行的只读代码。然而，这些美好的日子已经过去了。

现在，让我们来看一看时代的巨大变化吧！很多人在工作之余，要用他们喜爱的程序设计语言编写代码，修改配置文件，调整他们个人的开发环境。新技术提供了漂亮的图形化用户接口。因此，程序员必须能够开发出能够适应需求以及环境变化的程序。其中最主要的问题是，软件和硬件供应商在推出自己的产品的时候很少关心这些产品之间的联系。

的确，过去的那些美好的日子已经一去不复返了。现在我们所需要的是新的工具和新的技术——它们所支持开发的应用程序能够运行在不同的平台上而且能够很容易地被修改以适应新的需求。否则，软件危机永远不会结束。

第一章　VisualAge for C++和应用开发

在考察制造业的时候，我们发现很多制造商都是用部件来建造他们的产品的，像螺钉、螺母这样的标准化元件在任何地方都能买到。工厂可以把相同的部件安装在不同的产品上。例如，汽车制造商在他们所有型号的汽车上安装同样的观后镜和同样的离合器。在生产真正的产品之前，工程师们是通过建立实物模型来寻找可能出现的结构上的问题的。

然而，在软件行业中，我们却发现很多新产品都是由最底层建起的。这其中有很多的原因，比如，出现了一种能在某个特定领域内方便地解决问题的程序设计语言；一个应用程序需要一个新的数据库系统来支持，而这一变化导致了要在已有软件上作出很大的改动；拥有更好思想的新的开发人员的到来，或者原来的一位开发人员带着他所有的、没有文档化的经验离开了公司。

目前，软件市场上还没有标准化的软件模块可以让程序员使用。现在有很多函数集合（即程序库）可以帮助我们解决很多软件领域的问题，比如数据库、网络、通讯、图形用户界面。但是，如果程序员想要使用这些库函数，就要费力地去翻阅很多卷的使用手册来查找每个所要用到的函数，以及相应的参数。另外，如果程序员混合使用来自不同供应商的库函数，那么他们经常会遇到兼容性问题，比如名字的重复问题。

VisualAge for C++既没有除去底层的函数库，也不能够阻止库函数供应商使用相同的名字，但是它可以支持你去建立起设计良好的模型以及能够在不同的应用程序、不同的软件和硬件平台上多次复用的软件部件。

1.1　可视化程序设计

在过去的10年中，软件设计者为个人计算机和工作站提供了丰富的操作系统，为用户提供了各种各样的图形用户界面。同时，软件开发者也开始让他们的应用程序逐步适应这种新环境。

从用户的角度来看，图形化用户界面有很多显而易见的优点：

- □ 用户不必再去键入具有很多参数和选项的命令行
- □ 用户可以更直观地控制应用程序
- □ 用户可以同时看到多个角度的视图
- □ 应用程序看起来很漂亮而且具有一致的界面

但是，程序员必须与成百的新函数打交道，以开发图形用户界面，而且必须使用一种新的程序设计方法：事件驱动程序设计。

在事件驱动程序设计风格中,程序员向图形元素发送消息。当事件发生时,图形系统就向程序员提供的函数发送消息。这样,从开发者的角度来看,图形用户界面的不足之处也是显而易见的:

□ 必须很快学会这些新观念

□ 增加了开发的复杂性

□ 由此导致开发时间的增加

为了缩短学习周期和开发时间,各个软件公司都推出了辅助工具来支持程序员可视化地开发应用程序。这样,程序员就不必从打开编辑器开始编写程序了。他们可以在屏幕上直接设计图形用户界面。当然,不仅是图形用户界面,程序员还应该能够在其中加入业务逻辑和数据访问事务处理。但问题是,对于绝大多数现有工具来说,程序员不能继续用可视化方法开发下去了,他们必须手工地编写剩下的代码。但是如果使用 VisualAge for C++,程序员就可以继续可视化地工作下去了,因为他们所拥有的组件不仅可以建立图形用户界面,而且可以建立整个应用程序,这其中包括数据库的访问和添加多媒体功能。

在运用 VisualAge for C++这一具有强大功能的工具之前,必须熟悉面向对象的开发方法。这种方法是在软件开发变得越来越复杂的情况下出现的。图形用户界面、远程数据访问和底层通讯协议,以及迅速地重新编写已有代码使其能够适应新的硬件/软件环境,这些都是外界向我们提出的挑战。我们需要借助方法上和工具上的帮助来理解和对付这些软件开发中的复杂问题。

以前我们采用的是把巨大的程序分解为过程的设计方法。而今天软件专家们把软件开发看作是对象的组合。对象方法简化了我们对问题的理解,并且能够帮助我们把这些理解转换成软件。而本来就具有对象概念的面向对象的程序设计语言则可以支持程序员顺利地完成这一转换过程。

程序员不一定要采用面向对象的程序设计语言,他们可以使用过程式语言来实现对象及其行为。但是,过程式语言具有一定的危险性,它们不能阻止程序员随意地访问对象。程序员可以在每个模块执行的时候直接修改对象的数据。因此,当一个对象要改变它的行为或数据结构时,就必须改动很多模块。

在面向对象的程序设计语言中,程序只能通过特定的模块访问对象的数据。因此,当发生变化时,他们知道在哪里进行改动。另外,因为面向对象方法包含了程序员在编写程序代码之前必须经历的分析和设计阶段,所以他们就不会写出结构很差的代码。在本书中,我们介绍了面向对象方法的所有阶段,并告诉你如何在 VisualAge for C++的帮助下运用这个方法。

1.2 对象探讨

在本部分中,我们将介绍一些在本书中要用到的面向对象的术语和概念。这里并没有关于面向对象软件开发方法的更深入的讨论以及所有面向对象术语的定义。如果你对于面向对象方法的更广泛的含义有兴趣,我们建议你查阅我们在相关出版物部分所列出

的书目。

1.2.1 对 象

按照 Webster's Dictionary 的定义：

对象是指在现实世界中能感知到的东西，特别是指触觉或视觉上感知到的东西。

实际上，按照这个定义，我们会有一大堆对象！让我们以一辆轿车为例。如果你让两个人描绘同一辆轿车，他们可能会根据自己的经验，观察事物的方法以及兴趣，给出两个完全不同的回答。一个热情的司机会告诉你这辆轿车的发动机和其他关于轿车的内部工作细节；一个从来没有开过轿车的人会告诉你这辆车的颜色，它的估计长度、宽度、高度，以及其他任何能够看得到的方面；但是没有人会给出一个包含这辆车所有特性的"正确"的描述。即使是一位了解这辆车所有元件的工人也不可能完全正确地描绘它，因为他不会知道它当前的里程和磨损的轮胎橡胶的总量。现实世界中的对象具有无穷多的属性和用途，一辆轿车既可以用来作为驾驶工具，也可以有其他的用处，比如作为狗窝。

在驾驶学校，老师在描绘一辆轿车的时候，会主要强调它的功能并解释怎样把握方向盘、操纵杆、踏板以及怎样看懂仪表板上的各个指示器。从软件技术术语的角度，我们可以说老师解释的是这辆轿车的界面，不同的输入参数是怎样影响这辆轿车的动作的，以及驾驶员如何理解这辆轿车的输出值。

所以，我们看到了现实世界中的对象有各种各样的特性：

□ 属性：比如颜色、长度、宽度、高度、重量

□ 接口：比如方向盘、踏板、门把手

□ 功能或行为：比如发动、刹车

到现在为止，对于怎样实现从现实世界到计算机虚拟世界的转换我们已经有所了解了。当然，面向对象方法本身是不能找出这种转换的，但它通过一种巧妙的、透明的方式使得这种转换更加方便。过程式的程序设计语言提供了最基本的数据类型，比如整数、浮点数、字符和指针。另外，还有复合数据类型，即结构或记录，它可以把具有简单类型或复合类型的元素存储起来。面向对象语言提供了一种特殊的数据类型，它可以存储一个对象的所有属性并且保证只有通过它自己的函数或已定义好的接口才能操纵对象的属性。

在过程式的方法中，一个对象通常被分成包含属性的结构和与属性相关的函数和操作。使用过程式语言的不足之处在于不可能阻止程序员直接操纵其他对象的数据。虽然这种操作并不是最初设计的，但是当受到时间上的限制的时候，程序员更倾向于这样作。因此，不同对象的过程和数据结构经常相互依赖，这就使得复用几乎是不可能的了，而且还增加了扩充已有软件系统时的复杂性。

在计算机环境中，对象一旦设计好了，它就是离散的了，它拥有有限个数的属性以及预先定义好的操作。虽然，软件对象代表的是现实世界中的对象，但是它的实现是针对特定领域问题的。让我们再一次把轿车作为例子吧。对于一个支持轿车销售的应用程序来说，设计者实现的对象具有对于销售有重要意义的属性：例如价格、马力、颜色。这些属性都具有相当高的层次，因为顾客通常并不关心生产这辆所耗费的铁和铝的总量等底层的细节信息。但是在一个支持轿车生产过程的应用程序中，设计者必须赋予轿车对象更多

的属性,因为工程师关心的是生产轿车中的重要细节信息。在这两个应用程序中,轿车这个对象都只是采用了它在现实世界中的属性的一部分。

1.2.2 类

如果我们参照 Webster's Dictionary 的定义,我们会发现:

类是一个集合或群体,其中的成员至少共享一个属性。

这个定义使得类的成员具有很大的不确定性。让我们以动物类为例,其中的所有成员都有"具有器官"这个属性。而生物学告诉我们,人也同样有这个属性,那么按照 Webster' Dictionary 的定义,人也属于同样的类。这恐怕不对吧?另外一个例子是:轿车是一个类,其中的所有成员都有"四个轮子"这个属性,滑冰鞋也有"四个轮子",那么它们也属于这个类。这又错了吧?

这些例子告诉我们,找到正确的抽象层次是多么的困难。当然,在这里很显然,我们选择了错误的层次,动物和人的确属于同一个类,但那是一个更一般的类:生物。滑冰鞋和轿车也属于一个更为一般的类:四轮机械。不难看出,我们可以使用比描述对象更为抽象的术语来描述类。

我们从科学界选择的第一个关于类的例子并非偶然。因为生物学家确实把生命世界划分成几类,比如:哺乳动物、冷血动物、微生物。化学家也对物质进行了分类,比如有机物、无机物。而物理学家则是按照固态、液态、气态来划分的。我们了解到在科学界的每个领域中都是用类把我们的复杂的世界划分为可以理解的集合的。按照我们当前兴趣的焦点,分类可以是非常详细的。

在计算机领域中,我们用类使软件问题具有结构化。因此,类应该被看作是这样的集合,其中的成员共享某些属性或提供相似的功能。我们看一下轿车这个对象和类的例子。"汽车"类是所有汽车的集合,每个汽车都提供了相同的功能和相似的属性和接口。这是非常显然的,因为有经验的驾驶员可以操纵每一个交付给他们的汽车,即使以前他从未见过这种型号的车。"汽车"类描述了所有汽车所共同拥有的一般功能。"一辆汽车"这个对象是"汽车"类的一个独立的实例,比如某人的车或在某辆指定地点停放的汽车。

1.2.3 继 承

在现实生活中,我们非常了解术语"继承"的含义。比如,我们总是梦想着通过继承得到一笔遗产。但是在面向对象中,它的含义却完全不同。

严格地说,面向对象中的继承与生物学中的继承的概念更相近,因为它不是指一个类把它的特性传递给另一个类,并且传递下去;而是说,新类最初的样子和行为看起来像它的父类,并且新类和父类这两个类是同时存在的。类的设计者通过增加和改变最初的属性或功能来使这个新类具有特殊性。这样,这个类就可以满足它的设计要求了。可以把这个工作比作一个工程师想设计一种新型轿车。这个工程师并不是从一张白纸作起,而是选择一个已有的轿车作为原型,在这里或那里修改它。比如,他会去掉原有车顶以设计一种可以敞篷的车顶。

工程师们对新型汽车的设计是通过绘图或 CAD 程序辅助完成的。另外他还要考虑

到去掉车顶会导致一系列的相应变化。我们假定一个没有顶篷的轿车需要与有顶篷轿车不同的底盘、前门,以及其他部分。然而,当一个继承类要接受这么大的变化的时候,我们并不推荐使用继承机制。继承的主要优点在于对已有类的复用。

看一下图 1.1,"带篷汽车"类是"汽车"类的特殊化,而"汽车"类是"带篷汽车"类的一般化。而"汽车"类的泛化就应该是"交通工具"类了。继承意味着特殊化的类具有它的祖先类的所有特性,也就是说,特殊化类的行为就像它的祖先类一样,并且拥有同样的属性。实际上,如果从一个已有的父类导出一个新类,那么得到的是这个父类的一个拷贝。但用户可以对这个导出类增加新的属性、成员函数或改变其已有的成员函数。比如"打开车顶"这一操作对于"带篷汽车"类已不再有意义了,而摇柄也没有出现在"汽车"类。

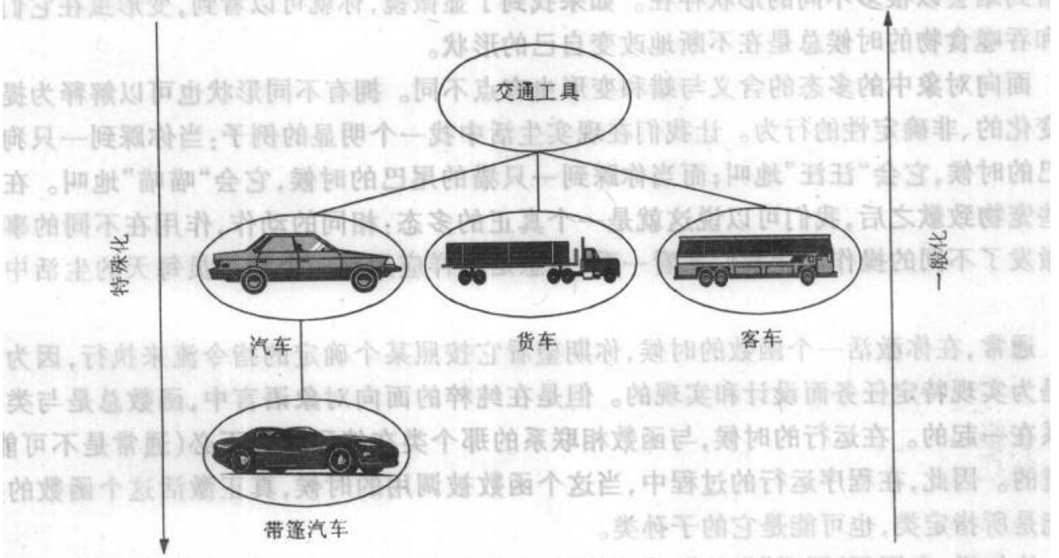


图 1.1 继承

导出类能够继承它祖先类的所有功能,这为代码复用提供了手段。公共行为的修改只需一次就可以完成,即只需修改祖先类的功能即可。

1.2.4 封 装

轿车功能的实现细节对驾驶员来说是不可见的,驾驶员只需知道怎样与接口打交道就可以了。

轿车驾驶员应该小心地操纵发动机并正确地移动变速杆。对于初学者来说,在驾驶学校的最初的学习生活中最大的困难是压住离合器,移动变速杆,放开离合器,并再次压紧离合器这一操作过程。在一个装有自动传动装置的轿车中,驾驶员可以在不使用另外的踏板的情况下移动变速杆。变速杆的变化是被封装在加速过程内部的。因此驾驶员只需操纵油门就可以了。驾驶员还知道,只要踩住刹车,轿车就会减速,而他并不知道刹车板启动的是什么形式的刹车装置。

面向对象的程序设计语言同样提供了封装功能。对象公开的只有它们的接口,而不是内部实现。函数调用者不必去关心相应的算法是怎样实现的,他们只关心函数的行为。

1.2.5 多 态

术语多态在 Webster's Dictionary 中的定义是:

对于相同的人种或种族,由于遗传上发生变化而导致了不同个体之间特征上的差异。

的确,这个定义看起来非常奇怪。因此,让我们再查一下多态这个词(英文为 Polymorphism)的希腊字根:poly 的意思是很多或多个,morph 的意思是形状。因此,我们把多态定义为:

一个事物或有机体能够以多种形态存在的能力。

在现实世界中,我们可以找到像蜡或变形虫这样的例子。在参观蜡像馆的时候,你就会看到蜡会以很多不同的形状存在。如果找到了显微镜,你就可以看到,变形虫在它们移动和吞噬食物的时候总是在不断地改变自己的形状。

面向对象中的多态的含义与蜡和变形虫有点不同。拥有不同形状也可以解释为提供可变化的、非确定性的行为。让我们在现实生活中找一个明显的例子:当你踩到一只狗的尾巴的时候,它会"汪汪"地叫;而当你踩到一只猫的尾巴的时候,它会"喵喵"地叫。在向这些宠物致歉之后,我们可以说这就是一个真正的多态:相同的动作,作用在不同的事物上激发了不同的操作。让我们来看一看,多态是怎样应用到一个程序员每天的生活中去的。

通常,在你激活一个函数的时候,你期望看它按照某个确定的指令流来执行,因为它就是为实现特定任务而设计和实现的。但是在纯粹的面向对象语言中,函数总是与类相联系在一起的。在运行的时候,与函数相联系的那个类在编码时是不必(通常是不可能)知道的。因此,在程序运行的过程中,当这个函数被调用的时候,真正激活这个函数的类可能是所指定类,也可能是它的子孙类。

比如说,在调用"图形"类的"画"函数时,我们所期望的是一个实际的对象把它自己画在屏幕上。但是,我们并不关心在运行时刻这个实际的对象是"圆"类的实例,还是"矩形"类的实例,这两个类都是"图形"类的子孙类。显然,"画"这个函数对于不同的类来说差别是很大的。比如,在这个将来应用的版本中,又增加了一个或多个子孙类,如"三角"类,它也提供了自己的"画"函数。最好的情形是这样:"画"函数的调用者并不知道这个新类的出现。

开发者要与很多的类和对象打交道。其中,一部分类和对象是现实世界中对象的直接映象。而其他的类和对象则用来表示在实现业务逻辑和与用户或外部设备通信时所需要的服务。在大的软件应用中,对象间的相关和交互作用既是难以描绘的,也是极为复杂的。因此,我们必须找出办法以弄清楚这种复杂性。没有这些方法,开发者们很快就会发现他们处于"有了对象却无法面向"的尴尬境地。

1.2.6 面向对象方法

在 Webster's Dictionary 中,"方法"的定义是:

有序、系统地组织安排、排序或其他相似的东西。

事实上,我们需要一种系统化安排的工作模型来定义、求精、实现、维护和文档化复杂