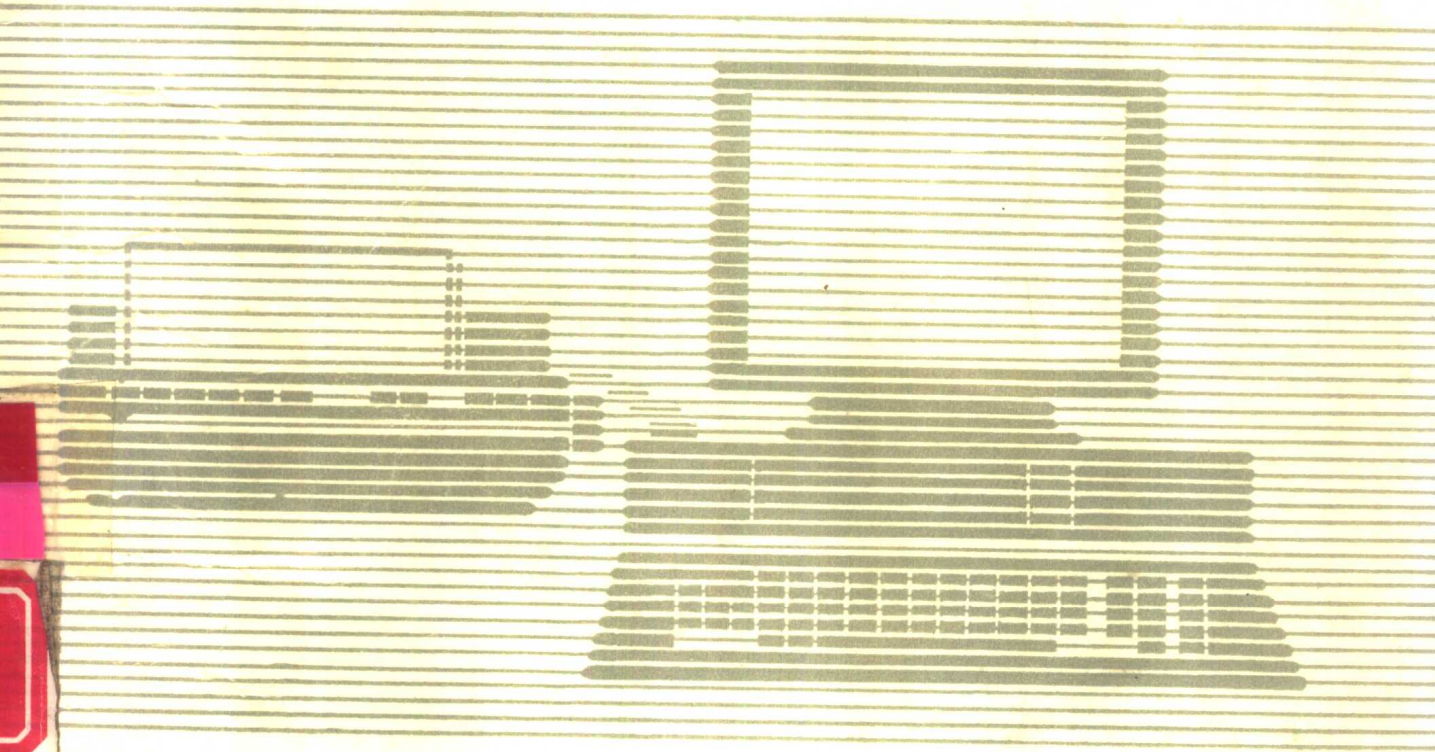
Turbo Pascal (5.5版)

程序设计指南

〔美〕BORLAND公司 著 陈彪 李宏 许伟 译



上海科学普及出版社

IBM-PC软件

Turbo Pascal(5.5版)程序设计指南

[美] BORLAND公司著

陈彪 李宏 许伟 译

上海科学普及出版社

Turbo Pascal 5.5 Object-oriented Programming Guide

Borland International, Inc.

责任编辑：徐丽萍
封面设计：毛增南

Turbo Pascal (5.5版)程序设计指南
〔美〕 BORLAND公司著
陈 彪 李 宏 许 伟 译
上海科学普及出版社出版
（上海曹杨路500号 邮政编码 200063）

新华书店上海发行所发行 上海市印刷七厂一分厂印刷
开本 787×1092 1/16 印张 6.75 字数 150000
1991年2月第1版 1991年2月第1次印刷

ISBN 7-5427-0289-0/TP·18 定价：6.00元

内 容 提 要

Turbo Pascal (5.5)版除了具有以前版所有的特性之外，还提供了面向对象的程序设计方法。本书包含了Turbo Pascal 5.5版中关于面向对象这种新特性的信息，至于有关Turbo Pascal的语法、编程、调试、安装等信息，请参见《Turbo Pascal(5.0版)使用和参考手册》(上海科普版)。

本书内容包括:面向对象的程序设计方法;面向对象的调试;Turbo Pascal 5.5语言定义;对于Turbo Pascal覆盖管理程序的改进;面向对象特性的具体实现等。

读者对象:IBM-PC用户、程序员、大专院校有关专业师生。

目　　录

第一章　概　述

面向对象的程序设计方法是一种强有力而且高效的程序设计方法，用它取代传统的编程方法已成为一种趋势。Turbo Pascal 5.5版本提供了这样的面向对象的程序设计方法，而且还具备了Turbo系列软件产品所惯有的那种高速度。Turbo Pascal 5.5版本除了提供以前Turbo Pascal版本中所具有的全部特性之外，还提供了如下一些很有发展前途的编程手段。

- 既提供了静态对象以获得最大的效率，又提供了动态对象以获得最大的灵活性。
- 静态手段和虚拟手段。
- 构造手段和释放手段用于创建和释放对象(这可以节约编程时间并提高代码的可读性)。
- 对象常量——静态对象的初始化工作是自动进行的。
- 更快的编译速度——Turbo Pascal 5.5版本的编译器比以前版本的编译器以更快的速度编译源代码。
- 进一步改进了覆盖管理程序，以减少磁盘的输入输出操作，使得覆盖管理更为高速。
- 进一步改进了Help屏幕，使用户可以从Help信息所包含的例子中直接截取你认为有用的程序片段。
- 丰富的随机指示信息，引导你熟悉Turbo Pascal的集成化开发环境。
- Turbo Pascal 5.5 中关于面向对象的扩展是在 Larry Tesler 的《Object Pascal Report》和Bjarne Stroustrup的《The C++ Programming Language》这两本著作的影响下进行的。

第一节　本书简介

本书所介绍的是Turbo Pascal 5.5版本中提供的面向对象编程这种新特性。至于有关Turbo Pascal的其它信息，请参见上海科学普及出版社出版的《Turbo Pascal (5.0版)使用和参考手册》(下同)。

- 下面是对本书中各章节和附录的一个简单说明。
- 第一章：面向对象的程序设计方法

  这一章向你介绍了面向对象程序设计方法所包含的一些重要概念——对象和记录有什么不同、把数据和代码包裹起来有哪些好处、多形性、静态对象实例和动态对象实例的对比等等。该章中还给出了一些实例，通过这些实例来进一步介绍面向对象程序设计方法的核心思想。
- 第二章：面向对象的调试

  这一章介绍了Turbo Debugger中为了支持Turbo Pascal 5.5版本面向对象编程这一新特性而做的改进。包括对对象观察窗口和对象层次结构观察窗口等的介绍。
- 第三章：Turbo Pascal 5.5语言定义

这一章给出了Turbo Pascal 5.5版本中有关面向对象扩展部分的形式定义。

- 第四章：覆盖
  这一章讨论了对Turbo Pascal覆盖管理程序的改进。
- Turbo Pascal的内部实现
  这一章介绍了Turbo Pascal 5.5版本中面向对象特性的内部实现方法。
- 附录A：新增加的出错信息和修正后的出错信息
  该附录列出了针对Turbo Pascal 5.5版本中面向对象的这种扩展而新增加的编译出错信息和警告信息。
- 附录B：术语词典

第二节 安 装

你要做的第一件事就是在你的系统上安装Turbo Pascal 5.5版本软件。你的Turbo Pascal 5.5版本软件中包括了运行集成化开发环境和命令行编译版本所需要的一切文件和程序。名为INSTALL的程序专门用于把Turbo Pascal 5.5版本安装到你的系统上，它既可以把Turbo Pascal 5.5版本安装在软盘上也可以把它安装在硬盘上。

INSTALL程序引导你逐步完成整个安装过程，你只要遵循屏幕上对每一步所给出的指导信息就行了。你应该仔细阅读这些指示信息。如果你把Turbo Pascal安装在软盘上而不是硬盘上，你必须先准备好四张空的、容量各为360k的软盘。

按如下步骤运行INSTALL程序：

1.把Turbo Pascal 5.5版本软件包中标记为Installation Disk的软盘插入A驱动器中。

2.键入A:，后随回车。

3.键入INSTALL，后随回车。

从这里开始，只要一直遵循INSTALL在屏幕上给出的指示信息就行了，直到完成整个安装过程。

在试用了Turbo Pascal集成化开发环境后，你可能还想改变一些任选项的值，为此只要运行TINST。在《Turbo Pascal(5.0版本)使用和参考手册》的附录中对该程序有详细的介绍。

特别说明

- 如果你使用了INSTALL的Upgrade(升级)任选项，那么Turbo Pascal 5.5版本将覆盖所有Turbo Pascal 5.0版本中的同名文件。
- 如果你把图形文件装入另一个子目录(例如是C:\TP\BGI)，那么请记住在调用InitGraph时，必须指明关于驱动程序和字体文件的全名路径。例如：
  InitGraph(Driver, Mode, "C:\TP\BGI");
- 如果GRAPH.TPU不在当前目录中，那么你必须通过Options/Directories/Unit Directories命令(或者命令行编译中的/u任选项)，把它的全路径名加入到单元目录中，从而可以编译一个BGI程序。
- INSTALL和TINST这两个程序都可以接受命令行任选项。如果你对阅读由IN-

STALL和TINST程序给出的信息感到有困难，可以通过任选项使它们工作在黑、白显示模式下。如:

A: INSTALL/B　强迫INSTALL运行在BW80(黑白80列)模式下

A: TINST/B　强迫TINST运行在BW80(黑白80列)模式下

如果你使用的是一个LCD屏幕或者一个彩色图形适配器和一个单色屏幕或者组合屏幕，你就必须使用/B任选项。《Turbo　Pascal(5.0版本)使用和参考手册》中还会告诉你如何强迫集成化开发环境工作在LCD屏幕(或者CGA加上一个单色屏幕，或者组合屏幕)上。

第三节　随机的帮助信息

你可以通过随机的帮助得到关于集成化开发环境和语言本身的有关辅助信息。无论光标是处于菜单上还是在某个窗口内，只要按下F1功能键就可以进入随机帮助屏幕再按一下F1功能键便可得到关于随机帮助系统的索引目录。

只有当光标处于Edit(编辑)窗口内时，才可以得到关于语言本身的辅助信息。为了得到关于语言的辅助信息，先在Edit(编辑)窗口内把光标停在需要查询的对象上，然后按下Ctrl—F1，你就可以得到关于Pascal保留字的句法或者某个过程或函数的参数以及使用的辅助信息，你还可以把例子中的某个片段载入你的程序，并得到编译时应该使用的开关值等等。

为了从辅助信息所包含的例子中把有关的程序片段载入你的程序中，只要遵循以下几个步骤即可:

1. 一旦你找到所需拷贝的辅助信息后，按下C键。该命令激活光标，从而你可以把光标定位在辅助屏幕上的任何位置。
2. 在把光标定位到你想拷贝内容的开始位置后，按下B键开始拷贝，然后使用↑、↓、←和→这四个箭头键把光标移动到所需拷贝内容的末尾(与此同时所需拷贝的正文内容会增亮)。如果再次按下B键便把当前光标所处的位置重新设为拷贝的开始。
3. 键入回车键便结束整个截取过程，而且被截取的内容被置入你所编辑的文件中。
4. 被截入编辑窗口的正文部分仍被标记为一个光块，你可以方便地对它施行移动，拷贝等块操作。

第四节　如何与Borland公司联系

在阅读本书和使用了Turbo　Pascal之后，你若还想进一步与Borland公司获得联系以获取技术上的支持，我们建议你遵循如下的步骤:

- 最好的方法是通过CompuServe进入Borland的专题讨论:在CompuServe的主菜单上键入Go　BPROGA，然后通过菜单进入Section　2，把你的问题或者建议留在其中让有关人员进行处理。
- 如果你愿意，可以写信寄往如下的地址:
  Technical Support Department

Borland International
P.O. Box 660001
1800 Green Hill Road
Scotts Valley, CA 95066—0001

注意:如果你的信中附有程序,它必须约束在100行以内。我们建议你把它放在软盘上提供给我们,同时包括所有必须的支持文件,以及一步一步的操作过程以重复所出现的问题。在你决定请求技术支持之前,你应该也保留该软盘的一个拷贝,并确信我们可以通过你给我们的软盘重复你所遇到的问题。

你也可以直接给Technical Support Department来电,电话号码是(408)438—5300。为了使我们能尽快地帮助你解决所遇到的问题,在你打电话之前应该具备如下的信息:

- 产品名和版本名
- 产品系列号
- 计算机制造者和型号
- 所使用的操作系统和版本号

第五节　关于Turbo Pascal 5.5版本的进一步信息

Borland公司推荐如下的书/盘软件包:

Werner Fiebel.《Turbo Pascal DiskTutor》Borland Osborne/McGraw－Hill Programming Series. Osborne/McGraw－Hill, 1989.

该软件包包括了Turbo Pascal DiskTutor工具和Turbo Pascal 5.5版本编译器。

第二章　面向对象的程序设计方法

面向对象的程序设计方法(Object—Oriented Programming，简称为OOP)是与我们解决问题的方法极为相近的一种编程方法．它是编程方法的一种自然发展与以往的结构化程序设计方法相比它显得更结构化;与以往的数据抽象和细节隐藏方法相比它显得更模块化和抽象．一种面向对象的程序设计语言应具有三大性质:

- 包裹性: 把一个记录类型与对其操作的函数和过程结合起来形成一个新的数据类型——对象类型．
- 继承性: 定义一个对象，然后由它建立一个派生对象的层次结构，每个派生对象都可以访问其祖先对象的代码和数据．
- 同形性: 一个对象层次结构中的对象可共享同一个操作名，层次结构中的每个对象对该操作都有适合于其本身的实现．

Turbo Pascal 5.5版本的扩展部分向你提供了面向对象的程序设计语言所具有的全部功能: 进一步的结构化和模块化，更高层次的抽象以及语言本身所具有的可重用性．所有这些特性使你的代码更结构化，更易于扩展和维护．

面向对象的程序设计方法也向你提出了挑战，它要求你放弃多年来你一直认为是标准的编程方法和习惯．一旦你做到这一点，面向对象的程序设计方法就是一个简单、直截了当而且功能强大的工具,.它可以解决以往困扰传统软件开发的许多问题．

注意:如果你曾经使用别的语言进行过面向对象的程序设计，你应该把过去对面向对象的程序设计方法的印象抛在一边，并且重新以Turbo Pascal 5.5版本所包含的概念来学习面向对象的程序设计方法所具有的特点．面向对象的程序设计方法并不只是一种编程方法，它包含了一系列的思想．除Turbo Pascal 5.5以外，C＋＋和Smalltalk这两种编程语言也都具有面向对象的特点，但Turbo Pascal 5.5版本更接近于C＋＋．Smalltalk 是一个翻译器，而Turbo Pascal从一开始就是一个纯粹的代码编译器．代码编译器处理问题的方法与代码翻译器的处理方法有所不同，而且编译的速度要比翻译快得多．Turbo Pascal一开始就是作为一个产品开发工具，而不是作为一个研究工具出现的．

如果你目前对面向对象的程序设计方法还一无所知，那就再好不过了．因为在过去的几年里，概念上存在太多的混淆、太多的误解以及那么多一知半解的人的高谈阔论已经把一切都搞乱了，因此最好尽量抛弃以前从别人那里得到的关于面向对象的程序设计方法的那些知识．学习和掌握面向对象的程序设计方法的最佳(也是唯一的)途径是: 坐下来亲自实践一下．

第一节　何为对象(Object)

对象到底是指什么?环顾一下你的周围，可能你看到了你带来的用于午餐后享用的一个苹果．假定你想用软件术语来描述这个苹果，你可能首先把它分成几部分，然后再加以描述:用S代表苹果皮，用J来表示苹果所包含的液体果汁，用F来表示苹果的重量，用D来

表示苹果所包含的种子数等等。

最好不要采用上面所介绍的那种方法，而采用下面介绍的方法。假定你自己是一个画家，你看到一个苹果并且画下了这个苹果。关于苹果的画并不是真正的苹果，其实画只是平面上的一个符号，但是它不象上面把苹果分成几部分的方法那样用几个数以及一些相互独立的数据项来描述苹果，相反，画中的苹果仍是一个整体，而且体现了各部分之间的必然联系。

这个苹果可以被分成几个部分，但是一旦它被分成几个部分之后，它就不再是一个苹果了。如果把每个部分都集中起来，那么部分与整体之间以及部分与部分的关系就显得更为自然，这就是"包裹"，这个概念非常重要，后面我们还要详细讲述。

对象就是将特性与行为结合在一起，它是刻画我们所生活的这个世界中元素的特性和行为的数据抽象，它是至今为止最高级的数据抽象。

另一个同等重要之处是：对象可以继承其"祖先对象"的特性和行为。你可以感觉得到这是一个很显著的飞跃。继承性也许就是面向对象的Turbo Pascal与以前版本的Turbo Pascal在程序设计方法上最大的不同之处。

第二节　继承性(Inheritance)

科学的目标在于描述整个世界，而许多科学研究工作仅仅是创建一棵家谱树。当昆虫学家发现一种未知的昆虫时，他们最关心的是新发现的昆虫在包含所有已知昆虫的体系表中应处于哪个位置。植物、鱼类、爬行动物、化学元素、亚原子粒子以及外部宇宙空间都有类似的体系表，它们看上去都象是家谱树：顶部是单独的总体类别，它下面是一些子类别，子类别又可拥有子类别，直至不可分为止。

例如，昆虫类中有两个分支：有可见翅膀的昆虫和有隐藏翅膀或没有翅膀的昆虫。在带翅膀的昆虫类别下又有许多类别：蝴蝶类、蝇类等等。这些类别又有许多的子类别，进一步这些子类别下面又可拥有更多的子类别。

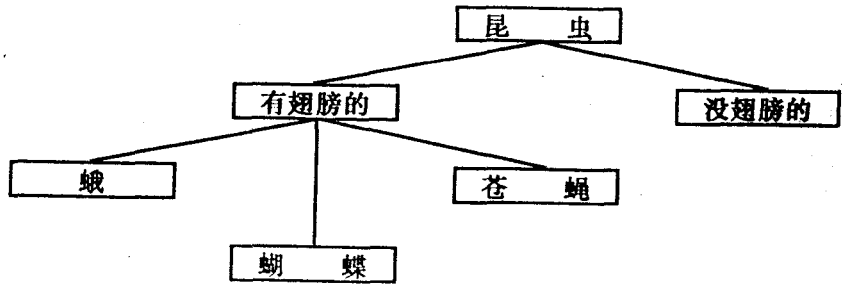


图 2.1　　昆虫分类图的某个局部

分类处理的方法叫做分类学。它与面向对象程序设计方法的继承机制非常相似。

在确定新的动物或对象的类别时，科学家们常提出这样的问题：它与总类别下其它成

员的相同之处是什么，不同之处又在哪里。每个不同的类别都有与之紧密相连的一组行为和特性。科学家们从一个种类的家谱树的顶部出发，然后开始形成分枝，其间不断地重复上面提出的那些问题。最高层是最一般性的，问题也是最简单的:有没有翅膀?每一层比上一层更为具体，最后达到这样的阶段：需要根据昆虫后腿第三枝节上毛发的数目来决定昆虫的类别——这已是非常具体了。

值得牢记的重要一点是：一旦在一个类别中定义了一个特性之后，该类别下的所有子类别也都包含了这个特性。因此当你确定一个昆虫属于双翅目时，你不必再另外指出它有一对翅膀了，因为属于双翅目下的昆虫都具有该目所具有的特性(有一对翅膀)。

不久你就会认识到，面向对象的程序设计方法主要就是建立一棵数据结构的家谱树。在诸如Pascal之类的传统语言上，面向对象的程序设计方法加入了一种很重要的机制，借助于它，一种类型可以继承另一种更为简单、一般的类型所具有的特性，这个强有力的机制就是继承性。

第三节　对象：具有继承性的记录

在Pascal中，一个对象与一个记录非常相似。记录是用于把一些相关的数据包裹在同一个名字之下。在图形环境中，你也许会把图形屏幕上任一位置的X、Y坐标集中起来，形成一个名为Location的记录：

```
Location = record
  X, Y : Integer;
end;
```

这里Location是一个记录类型，也就是说它是编译程序用于创建记录变量的一个样板。一个Location类型的变量称为Location类型的一个实例。Pascal中"实例"这个词偶尔被用到，但对于运用面向对象的程序设计方法的编程者来说，这个概念一刻也离不开。因此，你最好开始习惯于用类型和类型的实例这样的术语来思考问题。

可以从两个角度来看Location类型：当你需要独立看待X、Y坐标时，你可以把它们单独看作记录的两个域;另一方面，当你想把X、Y坐标结合起来用以表示屏幕上的一个位置时，你可以把它们看成为一个整体，即Location类型。

假定你想在由Location记录所描述的屏幕位置上显示一个点，在Pascal中你可能会加入一个布尔变量域，用以标识在给定的位置上的象素是否是暗的。从而形成该记录类型的一个新的域：

```
Point = record
  X, Y : Integer;
  Visible : Boolean;
end;
```

你或许更聪明些在Point类型内创建一个Location类型的域:

```
Point = record
  Position : Location;
  Visible : Boolean;
end;
```

上面这种做法完全正确，而且Pascal的编程者总是这么做的。但这种方法没有促使你去考虑在新的软件中你所处理的对象的自然特性。你应该思考如下的问题:屏幕上的点和屏幕上的位置有什么不同?答案是: 点是具有明暗特性的位置。再注意一下这句话的含义:点也是一种位置，只不过它在位置的基础上又增加了某种特性。

点的定义意味着它对应于一个位置，面向对象的程序设计方法能够识别出这种特殊的关系。因为所有的点都必须包含一个位置，因此Point类型是Location类型的一个派生类型。Point类型继承了Location类型所具有的一切，并且加入了Location类型所不具备的一些新的内容，从而使Point成为不同于Location的一种新类型。

一种类型继承另一种类型特性的这种过程叫做继承性，继承者叫做"派生类型"，被继承者叫做"祖先类型"。

我们所熟悉的Pascal的记录类型不具有继承性，Turbo Pascal 5.5版本对Pascal语言做了扩充以支持继承性。扩充之一是提供了一种新的数据结构，这种新的数据结构与记录有关，但功能要强得多。这种新的数据结构类型通过新的保留字object进行定义。一个对象类型可以用Pascal的记录定义成完整、独立的类型;也可以定义成已定义的对象类型的派生类型，方法是把"祖先类型"的名字放在保留字object之后的括号里。

在前面的图形例子中，两个相关的对象类型可这样定义:

```
type
  Location = object
    X, Y : Integer;
  end;
  Point = object(Location)
    Visible : Boolean;
  end;
```

注意: 这里使用了括号来表示继承性。

这里Location是祖先类型，Point是派生类型。正如你将在后面看到的，这个过程可以一直继续下去:你可以定义Point的派生类型和Point派生类型的派生类型，以此类推。运用面向对象的程序设计方法进行编程时，一大部分工作就是在于建立对象的层次结构，以表达实际问题中对象的家谱树。

最终可追溯到Location类型的类型都叫做Location类型的派生类型。但Point类型是Location的"直接派生"，同样，Location类型是Point的"直接祖先"。一个对象类

型就象是DOS的一个子目录，它可以具有任意数目的直接派生，但只能有一个直接祖先。

正如定义所表示的，对象是联系紧密的记录。两者最明显的不同之处是对象定义中使用了新的保留字object，此外还存在许多别的区别。某些区别非常微妙，这你可在后面体会到。

例如，尽管Location类型中的X域和Y域并没有在Point类型中出现，但Point类型借助于继承性还是拥有这两个域。你可以象使用Location的X域一样使用Point的X域。

一、对象类型的实例(Instance)

对象类型实例的说明方法与Pascal中变量的说明方法相同，它们也可以说明成静态变量或者在堆上进行空间分配的指针。

```
type
  PointPtr = ^Point;
var
  StatPoint : Point;      {该变量马上可以使用}
  DynaPoint : PointPtr;   {该变量在使用前必须先用New分配空间}
```

二、一个对象的域

访问一个对象的数据域就如同访问一个普通记录的域一样，可以通过with语句或者借助于加前缀。例如:

```
MyPoint.Visible := False;
with MyPoint do
begin
  X := 341;
  Y := 42;
end;
```

现在你只要记住:对继承的域可以象对该类型中定义的域一样进行访问(读完本章后这一点会变得很自然)。对于一个对象所继承的域来说，并不因为是继承的而需要特殊对待。例如，尽管X域和Y域并没有出现在Point类型的定义中(它们由Location类型继承而来)，但你可以象在Point中定义它们那样使用它们。如:

```
MyPoint.X := 17;
```

三、好的实践和差的实践

虽然你可以直接访问一个对象的域，但这样做并不十分合适。面向对象程序设计方法要求尽可能地隐蔽对象的域，因此好的编程实践应尽可能避免对对象域的直接访问。乍一看你会觉得这种约束过于严格和武断，但这正是本章我们所要介绍的面向对象的程序设计方法的许多内容中的一部分。不久，你就可以体会到这种什么是好的编程实践的新定义的内在含义。当然在这之前我们还要做一些准备工作。现在你只要牢记:应尽量避免对对象域的直接

访问。

既然要避免直接访问对象的域，那么到底应该如何访问对象的域呢?也就是说怎样得到对象域的值以及怎样对它们进行赋值。

答案是通过"手段"对一个对象的数据域进行访问。一个"手段"是指在对象内部说明的并且与对象紧密相连的一个过程或函数。

一个对象的数据域描述了一个对象包含些什么内容，一个对象的手段则描述了该对象能做些什么。

第四节　手段(Method)

手段是面向对象的程序设计方法最显著的特性之一。要马上熟悉这个概念并非十分容易，让我们从数据结构的初始化这方面开始来逐步熟悉它。数据结构的初始化对于结构化程序设计来说是必须的一步，考虑如下定义记录的初始化:

```
Location = record
  X,Y : Integer;
end;
```

大多数编程者会使用 with 语句对 X、Y 域赋以初值:

```
var
  MyLocation : Location;
with MyLocation do
begin
  X := 17;
  Y := 42;
end;
```

这样做是可以的，但这些初始化工作只与特定的实例相关，如果有多个 Location 类型的实例需要初始化，你就需要使用多个 with 语句对它们进行做相同的处理。很自然你可以进一步建立一个初始化过程，它对由参数传入的 Location 对象类型的实例使用一个 with 语句:

```
procedure InitLocation(var Target : Location;NewX,NewY : Integer);
begin
  with Target do
  begin
    X := NewX;
    Y := NewY;
```