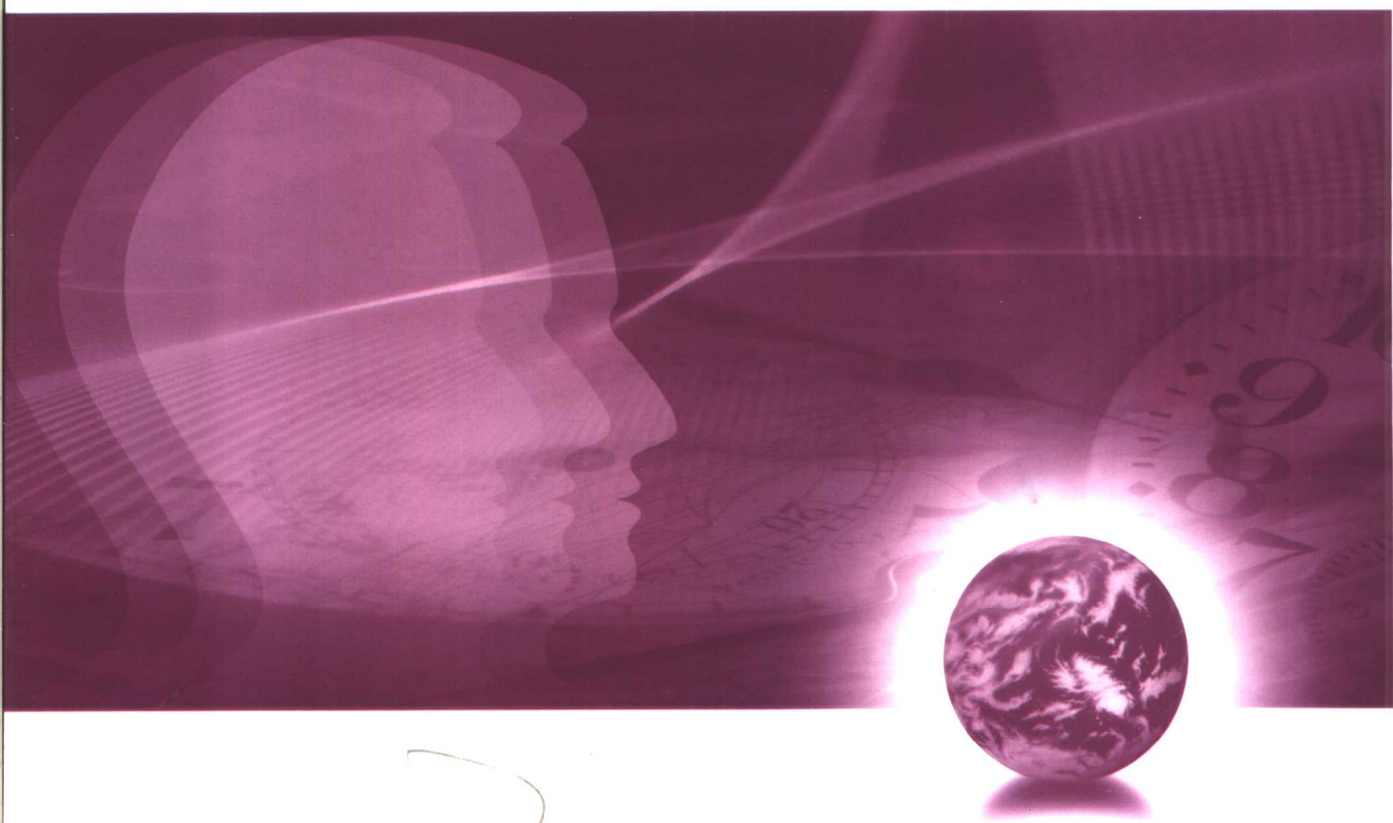
高等院校计算机科学与技术
“十五”规划教材

面向对象程序设计语言C++



陈文宇 编著

机械工业出版社
CHINA MACHINE PRESS

高等院校计算机科学与技术“十五”规划教材

面向对象程序设计语言 C++

陈文宇　编著

机械工业出版社

C++语言是一种扩充了面向对象成分的 C 语言，它保持了 C 语言的简洁和高效，又支持面向对象的程序设计。

全书共十章，详细介绍了面向对象的思想和方法，C++语言的主要概念和基本语法，面向对象的三个核心概念——封装、多态性和继承性在C++语言中的实现方式，以及 C++语言的某些扩充功能：如模板、命名空间、例外处理等，并用直观的方法讲述了面向对象的设计技术。

本书可作为大专院校、培训班和自考班教材，也可供从事计算机软件开发和应用的人员参考。

图书在版编目（CIP）数据

面向对象程序设计语言C++/陈文宇编著．—北京：机械工业出版社，2004.1

高等院校计算机科学与技术“十五”规划教材

ISBN 7-111-13714-0

Ⅰ．面… Ⅱ．陈… Ⅲ．C语言—程序设计—高等学校—教材

Ⅳ．TP312

中国版本图书馆CIP数据核字（2003）第120720号

机械工业出版社（北京市百万庄大街22号 邮政编码100037）

策　　划：胡毓坚

责任编辑：时　静

责任印制：闫　焱

北京交通印务实业公司印刷 · 新华书店北京发行所发行

2004年1月第1版 · 第1次印刷

787mm×1092mm 1/16 · 20.25印张 · 499千字

0001—5000册

定价：28.00元

凡购本图书，如有缺页、倒页、脱页，由本社发行部调换

本社购书热线电话（010）68993821、88379646

封面无防伪标均为盗版

出 版 说 明

信息技术高度普及的今天，具备一定层次的信息技术素养成为社会素质教育的一个重要目标，由此对高等院校的计算机专业教育提出了更高更新的要求。教育水平提高的关键是教学质量，那么对教学质量有直接影响的教材建设就成为了计算机专业教育的根本，为重中之重。

适逢高等院校计算机专业教育改革的关键时期，为配合相关的教材建设，机械工业出版社会同全国在该领域内享誉盛名、具备雄厚师资和技术力量的高等院校，包括清华大学、上海交通大学、南京大学、成都电子科技大学、东南大学、西安电子科技大学、解放军理工大学、北京科技大学等重点名校，组织了多位长期从事教学工作的骨干教师，集思广益，对当前高等院校的教学现状开展了广泛而深入的研讨，继而紧密结合当前技术发展需要并针对教学改革所提出的问题，精心编写了这套面向普通高等院校计算机专业的系列教材，并陆续出版。

本套教材内容覆盖了普通高等院校计算机专业学生的必修课程，另外还恰如其分地添加了一些选修课程，总体上分为基础、软件、硬件、网络和多媒体五大类。在编写过程中，对教学改革力度比较大、内容新颖以及各院校急需的并且适应社会经济发展的新教材，优先选择出版。

本套教材注重系统性、普及性和实用性，力求达到专业基础课教材概念清晰、深度合理的标准，并且注意与专业课教学的衔接；专业课教材覆盖面广、深浅适中，在体现相关领域最新发展的同时注重理论联系实际。全套教材体现了教育改革的最新思想，可作为高等院校计算机科学与技术专业的教学用书，同时也是培训班和自学使用的最佳教材。

机械工业出版社

前　言

C++语言是C语言面向对象的扩展，与C语言具有良好的兼容性，自1983年发表以来，引起很大的反响，在商业上取得了巨大的成功，是目前应用最广泛的一种面向对象的语言。

1998年公布了C++语言的国际标准版本，本书将以此标准为基础进行阐述。

学习C++语言，不仅要掌握其语言成分，更重要的是要学习一种与传统结构化程序设计完全不同的程序设计方法。因此，本书分为两大部分：面向对象的设计方法和C++语言的主要语言特性。

本书内容安排如下：第1章介绍面向对象的基本知识；第2章介绍C++语言和C语言除面向对象以外的其他不同的语言成分；第3章介绍面向对象的核心概念——数据封装的思想和实现；第4章介绍面向对象的核心概念——多态性的思想和实现；第5章介绍面向对象的核心概念——继承的思想和实现；第6章介绍C++语言的流库；第7章介绍类属的思想和实现；第8章介绍面向对象设计技术；第9章介绍C++语言的命名空间和C++语言的例外处理。

本书是作者在大量阅读关于C++语言和面向对象方面的参考书和近10年的实际教学的基础之上编著而成的。

在此，谨对张松梅老师、参阅文献的作者和翻译人员表示衷心感谢。

由于水平有限，书中难免有错误之处，敬请广大读者批评指正。

作　者

目　录

第1章 引 论

C++语言是一种面向对象语言，它所支持的面向对象的概念容易将问题空间直接映射到程序空间，为程序员提供了一种与传统结构程序设计十分不同的思维方式。因此，学习C++语言面临两个问题，如何建立面向对象的思维方式?如何用C++语言编程?亦即需要学习面向对象的设计方法和使用C++语言的编程方法。

面向对象的设计方法是尚在探索的领域，本章将从面向对象的目标、面向对象的核心概念、面向对象的思想和方法几个方面，给读者以面向对象概念的总体印象，并通过参考文献[26]中所述的CRC（Class_Responsibility_Collaborator）方法，介绍一种面向对象的设计方法。

1.1 面向对象的目标

传统的程序设计方法是模块化（或结构化）的程序设计方法，具体步骤为：

1）整个软件系统功能逐步细化为多个小的功能——功能划分。

2）多个小的功能对应由一个模块（如函数、过程、分程序、子程序等）来实现。

3）多个模块合作完成较大的功能，所有模块的合作完成整个软件系统的功能。

对于传统的程序设计，在设计和实现（编程）阶段考虑的是模块，程序本身也是由模块构成的，称之为面向模块的。

什么是面向对象的呢？下面先来介绍一下数据类型。

数据类型是一个抽象的概念，包含一组数据的定义和一组对该组数据操作的定义。

数据类型分为三种：简单数据类型、用户定义数据类型、抽象数据类型。

① 简单数据类型：语言本身提供的，如整型，包含所有的整数和对整数的操作，对于一个整数的成分（二进制表示）是不可见的，也不可直接操作。

② 用户定义数据类型：以简单数据类型为基础，它包含的数据成分是多个简单数据类型的数据（或已经定义好的其他用户定义数据类型的数据），可以对数据成分进行直接操作。

在一个高级程序设计语言的程序中，对于简单数据类型和用户定义数据类型，数据的定义和操作是分开的，只是在对数据进行操作时，需要检查该操作是否符合对应类型允许的操作（即类型检查）。

③ 抽象数据类型：在定义数据时，必须同时定义对数据的操作；它的成分（简单数据类型数据或用户定义数据类型数据）是不可见的，也不可直接操作，必须通过类型提供的操作进行访问。

抽象数据类型是从更一般的信息隐蔽（Information Hiding）原则派生出来的。抽象数据类型隐蔽了表示的细节，通过过程（或方法）来访问抽象数据对象。对象的表示是被保护的，防止了任何外界对它的直接操作。要对抽象数据对象进行访问，只能通过在抽象数据类型中提供操作才能进行。

对于类型的使用，必须通过类型的实际例子（简称实例或实体，即原来所说的变量或常

量）的使用来体现。

面向对象语言中的对象是“将某组数据和使用该组数据的一组基本操作封装在一起，而成的一个实体”。实际上就是抽象数据类型的一个实例，如图 1-1 所示。

对象和抽象数据类型的关系，就像整型变量和整型的关系。

面向对象的基本想法就是把要构造的系统表示为对象的集合。这里所说的系统，不仅应考虑为程序和软件系统，而且应广泛解释为计算模型、CAD/CAM 中的成分模型以及人工智能中的知识表示形式等等。

面向对象的思想与解决计算机软件面临的两大课题有关。一个问题是：怎样克服软件的复杂性；另一个问题是：怎样将现实世界模型在计算机中自然地表示出来（还需要包括它们之间的关系）。

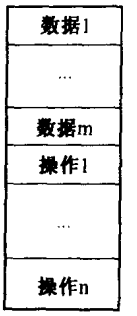


图 1-1　对象的模型结构

客观世界的问题都是由客观世界的实体及其相互间的联系构成的，把客观世界的实体称之为问题空间的对象。显然，“对象”因问题的特殊性是不固定的，一本书可以是一个对象，一个人也可以是一个对象，每一个对象都有自己的运动规律和内部状态，不同对象之间的相互作用和互相通信构成了完整的客观世界。因此，从思维模型的角度，面向对象很自然地与客观世界相对应。

从软件的角度，计算可以看作仿真。考虑一个雇员数据库，库中每个记录都代表一个雇员，可以说这是对公司某些方面的一种仿真或模拟。同样地，一个操作系统的数据结构常反映了真实世界中某些对象的状态。例如，可以用来反映这样一个事实：一个磁盘或正处于奇偶校验状态，或系统中的设备正分配给某个特定的作业。

如果每个被仿真的实体都由一个数据结构来表示，并且将相关的操作信息封装进去，仿真将被简化，可以方便地刻画对象的内部状态和运动规律。面向对象就是这样一种适用于直观模型化的设计方法。这意味着系统设计者从现实世界所得到的图像，或设计者头脑中形成的模型里所出现的物理图像与构成系统的一组对象之间有近乎一对一的对应关系。这一思想非常利于实现大型的软件系统。

作为克服软件复杂性的手段，在面向对象中，利用了如下对象的性质：

（1）将密切相关的数据和过程封装起来定义为一个实体。

（2）定义了一个实体后，即使不知道此实体的功能是怎样实现的，也能使用它们（这一点类似于库函数：对于使用库函数的人而言，不必关心库函数的实现细节，只要知道该函数的原型，就可以正确地使用他们）。

这相当于软件工程和程序设计方法论中的抽象化、抽象数据类型和信息隐藏等概念所具有的性质。它把系统中所有资源，如数据、模块以及系统都看作对象。每个对象把一组数据和一组过程封装在一起，这组过程对这组数据进行处理。使用这一方法，设计人员可以依照自己的意图创建自己的对象，并将问题映射到该对象上。该方法直接、自然，可以使设计人员把主要精力放在系统一级，而对细节问题可以较少地关心。

使用对象将信息局部化，并使程序结构与设计结构相吻合的优点，有利于在完善和维护阶段对软件进行修改，也有利于其他人（非设计人员）来清除软件错误。程序员容易确定程序的哪些部分依赖于正要修改的片断，而且正在修改的部分对其他部分影响很小。这对大型、

复杂软件的维护和改进是很重要的。

面向对象设计非常注重设计方法，因为它要产生一种与现实具有自然关系的软件系统，而现实就是一种模型。实际上，用面向对象方法编程的关键是模型化。程序员的责任是构造现实的软件模型。此时，计算机的观点是不重要的，而现实生活的观点才是最重要的。

因此，可以将面向对象的目标归纳为：对试图利用计算机进行问题求解和信息处理的领域，尽量使用对象的概念，将问题空间中的现实模型映射到程序空间，由此所得到的自然性可望克服软件系统的复杂性，从而得到问题求解和信息处理的更高性能。

将一组性质和功能相同的数据归为一类，使用一个抽象数据类型来定义他们。对于相互间还有联系或关系的数据，还要考虑他们之间的联系或关系。

在进一步叙述面向对象设计方法之前，先了解一下面向对象的几个核心概念。

1.2 面向对象语言的核心概念

在问题空间中，将客观世界的实体称为对象，不同对象之间的相互作用和互相通信构成了完整的客观世界。如何将问题空间的这一思维模型直接映射到程序空间，也就是说，面向对象语言提供什么概念来支持这一思维模型，纵观诸多面向对象的程序设计语言，最核心的概念是数据封装、继承和多态性。

1.2.1 数据封装

使用传统设计方法的程序员往往有着共同的弱点。用一个较为夸张的例子来说，当人们问及这些程序员时间的时候，他们会详细地解释钟表的概念。这是由他们使用计算机的经验所致。这里，必须搞清楚的概念是，告诉别人时间和让他人理解时间的概念是两个不同的知识领域。比方说，某人有一块手表，并常常告诉别人现在是几点了，这并不等于说他知道这块表的内部工作原理。手表是一个对象，只需知道它提供目前的时间，无需了解它如何运转。数据封装的概念反映的就是这一简单原理。

抽象是指对于一个系统的简化描述。对于使用系统的人员，不会去关心该系统的组成和工作的原理；他们所关心的是该系统具有什么样的功能，如何去使用该系统（既系统提供什么样的接口，让人们使用）。当然，对于该系统的实现人员，需要关心的是该系统的一切情况。

抽象的原则，运用在计算机领域，称之为“信息隐蔽”原则；在面向对象的程序设计语言中，使用数据封装机制实现信息隐蔽。

数据封装将一组数据和这组数据有关的操作集合封装在一起，形成一个能动的实体，称为对象。用户不必知道对象行为的实现细节，只需根据对象提供的外部特性接口访问对象。例如，可能使用一个数组来存储在屏幕上画一个字符所需要的字形信息：int font[num]；如何显示、缩小、放大、增加亮度和设置字符颜色呢?在传统 C 语言中，常用的解决办法是，将数据结构和相关的函数放入一个可编译的源文件中，把数据和函数作为模块看待。这当然是朝正确的方向迈出了一步。但这还不够好，在数据和函数之间没有明确的关系。若不用上述提供的相关函数也能直接操纵 font 的数据，这样可能导致一些问题。假如你决定用链表取代数组，而另一位做同一工作的程序员则可能认为，他有更好的方法来访问这些字形数据，

于是他编写了一些自己能直接操纵这些数组的函数。可问题是，那里已经没有数组了。当程序联调的时候，将产生错误。实际上，在数据 font 和操纵它的函数（显示、缩小、放大、旋转等）之间存在着明确的关系。数据 font 以及这些函数的实现细节对使用者并不重要，重要的是这些函数的界面，使用者只需根据这些函数的界面（函数名及其参数）和函数的功能进行访问。那么，在设计、实现、维护和重用程序时就有很大的帮助。

C++语言通过建立一个合适的 Font 类，将字形数据（称为数据成员）和函数（称为成员函数）结合在一起，形成一个新的抽象数据类型，称为类类型（class）。

```
class Font
{
  字形数据成员;
  操作字形数据的成员函数;
};
```

在这样建立的类 Font 中，能确保字形数据只能由类中的成员函数进行访问和处理。在任何时候，都可以自由地把字形数据从数组变为链表，只需改变成员函数的实现细节。由于这些成员函数的界面不改变，系统其他部分的程序（及使用者）就不会由于改动而受到影响。

类的概念将数据和与这组数据有关的操作集合封装在一起，建立了一个定义良好的接口，人们只关心其使用，不关心其实现细节。这反应了抽象数据类型的思想。

类本身还是一组对象共同属性和操作的抽象。类代表了一般性，而该类的每一个对象代表了具体性。

1.2.2 继承

继承是面向对象语言的另一个重要的概念。在客观世界中，存在着整体和部分的关系（a part of）、一般和特殊的关系（is a 或 a kind of）。

继承实现了一般和特殊的关系，解决了软件的重用性和扩充性问题。

举一个比喻的例子。例如，昆虫学家对昆虫的分类如图 1-2 所示。

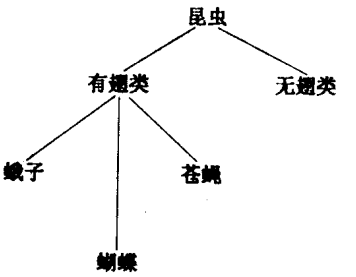


图 1-2 昆虫分类图示

在试图对一些新的动物或对象进行分类之前，关心的问题是：它与其他一般的类有多少相似之处？差别有多大？每一个不同的类都由一组描述它的行为和特征组成，最高层（根结点）是最普遍的，问题也最简单：有翅膀还是没翅膀？每一层都比它之前的层更具体。一旦某个特征定义下来，所有在它之下的种类都要包含该特征。所以，一旦确定苍蝇为昆虫有翅类中的一种，就不必指出苍蝇是昆虫，有一对翅膀，因为苍蝇从它的目中继承了这个特征。

在面向对象语言中，类功能支持这种层次机制。除了根结点外，每个类都有它的超类（superclass），又称父类或基类。除了叶结点外，每个类都有它的子类（subclass），又称派生类。一个子类可以从它的基类那里继承所有的数据和操作，并扩充自己的特殊数据和操作。基类抽象出共同特征，子类表达其差别。有了类的层次结构和继承性，不同对象的共同性质只需定义一次，用户就可以充分利用已有的类，符合软件重用的目标。

再看一个实际例子，将一个人当成对象，对于人的定义（如名字、年龄、性别等）和对人的操作（取名字、打印一个人的各种信息等）已经封装在 Person 类中了；下面将学生当成对象，学生是一类特殊的人，基本的信息和操作已经 Person 中定义过了，为了避免重复定义，C++语言允许用户将学生类 Student 说明为 Person 类的继承类。学生类 Student 和 Person 类的继承关系如图 1-3 所示。

这样，Student 类就可以不必在考虑对于人的基本信息的定义和操作了，继承机制允许学生类从人类中继承人的所有数据和操作，学生类只需要考虑学生特殊的信息的定义和对学生本身的操作即可。

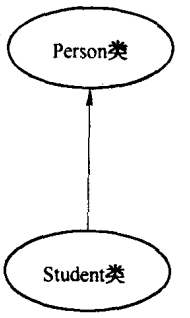


图 1-3　学生类和人类的继承关系

1.2.3　多态性

面向对象另外一个核心概念是多态性。所谓多态，是指一个名字（或符号）具有多种含义。

函数的名字有两个作用：其一，代表了该函数的函数体（一段代码）；其二，代表了该函数的功能。在传统的语言中（如 C 语言），不允许函数有同名的情况，考虑的是函数名的第一个作用；而在面向对象的程序设计语言中，如果发现多个函数的功能是一致的，尽管他们确实是不同的函数，但允许它们具有相同的函数名字，即存在同名函数。

下面让我们来考察多态性问题的一个类比问题。当一位汽车司机为避免撞车刹车时，他关心的是快速刹车（效果），而不关心刹车是鼓式刹车还是盘式刹车（实现方法的细节）。这里，刹车的使用与刹车的结构是分离的概念，可能有多种结构的刹车，它们的使用方法是相同的。相同的使用方法（相同界面）对应于不同种类的刹车结构（多种实现），这反映了多态性的思想。

与此类似，用户在使用函数编程时，关心的是该函数的功能及其使用界面，并不需要了解该函数的使用界面与函数的哪一种实现方法相匹配（binding）。也就是说，在设计这一级上，软件人员只关心“施加在对象上的动作是什么”，而不必牵涉“如何实现这个动作”以及“实现这个动作有多少种方法”的细节。在面向对象语言中，重载（或称为重载 overload）表达了最简单的多态性。比较容易理解的是函数重载。例如：

```
void fun(int, int, char);
int fun(char, float);
void fun(int, int);
void fun(float, float);
```

这几个函数都具有相同的函数名 fun，但其参数不同，它们的函数体也可以完全不同，编译能根据其函数参数的不同而自动选择相应的函数体进行匹配。因此，一个函数名代表了多种函数的实现（函数体）。

对于函数重载，若函数调用（界面）与哪一个函数体（函数实现）相匹配，是在编译时确定的，称为早期匹配（early binding）（或静态联编）。如果函数调用与哪一个函数体的匹配是在运行时动态进行的，称之为晚期匹配（lately binding）（或动态联编）。一般来说，早期匹配执行速度比较快，晚期匹配提供灵活性和高度的问题抽象。

C++语言中的虚特性的函数提供了晚期匹配带来的良好特征。函数重载强调的是函数名相同，函数参数和函数体不相同（编译能根据参数的差别进行识别和匹配）。虚特性的函数则强调单界面多实现版本的方法，亦即函数名、返回类型、函数参数的类型、顺序、个数完全相同，但函数体可以完全不同，这在编译阶段是无法识别的，只能由系统在运行时刻动态地寻找所需的函数体进行匹配。

在 C++语言中，通过继承关系，基类及其派生类之间构成一个树形结构（多重继承为图结构），树中的每个类（基类或派生类）都可以说明一个具有虚特性的函数。那么在这个类及其派生类中都可以定义这个虚函数的不同实现，但要求这些不同实现必须遵守相同的函数界面，否则虚特性丢失。使用时，系统能在运行时刻动态地寻找所需的实现版本。具体细节见第 5 章。

C++语言中的继承机制和虚函数使得软件可扩充性更为自然。可以继承基类拥有的所有特性，然后加进派生类所需要的特殊东西，使派生类对象以相似的方式工作并具有新的功能。派生类定义的类型及其虚函数版本是有规则的功能等级的真正扩充。由于这是语言设计的一部分，不是编程时才有的特性，所以很容易实现软件的可扩充性。

1.3 按对象方式思维

本节介绍参考文献[26]中所述的一种面向对象设计方法，称为 CRC 方法。

到现在为止，已经建立了这样的思想基础：面向对象是一种仿真技术，面向对象设计产生一种与现实具有自然关系的软件系统。那么，怎样用面向对象方法进行设计?怎么按对象方式进行思维?

若把软件设计理解成一个可操作模型，面向对象设计就是一种最自然的途径。模型化的外部世界都是由对象（传感器、设备、飞机、雇员、工资单、税收等）组成，软件对象只不过反映了这些外部对象。因此，面向对象设计人员通常不花费时间在学术上讨论寻找对象的方法，而是围绕这些模型化的物理世界的对象来组织模型。

面向过程的程序设计方法鼓励全局地看问题，典型的如结构程序设计，它强调系统功能的逐步细化。这种方法开发的程序，其中每个子程序都知道自己来自何处，将去哪里。而在对象程序设计中，每个对象严格地是局部的。每个对象管理自己的实现，并且不干涉互相的事。用于人，这是一种非常自私的哲学："各人自扫门前雪，不管他人瓦上霜"。用于模块，这是得到分散式结构的一个必要的需求：一个操作实现的变化只影响包含该操作的对象，一个新类型对象的增加在大多数情况下将使其他对象完全不受影响。

一个旋转图形的例程（routine）没有必要知道所有的图形类是什么。正是这种具有局部性质的对象共同工作的集体效应完成了计算工作。因此，学习面向对象设计，需要人们将思维方式从使用完整的知识来全局地看问题，转变到仅根据局部图像寻找对象，并将对象进行组合的方法上来。这就要求集中精力判断哪一个对象应做哪一部分计算，仅仅当对象做正确的工作时，才能将注意力移到其他问题上，诸如对象怎样被描述，它们怎样被继承等。

设计对象需要做许多小决策，怎样将计算部分分配给不同的对象，称之为分散责任（distribution responsibility），这是设计决策的基础，其余都是次要的。CRC 方法帮助设计者分散责任，直到设计的最后阶段才考虑问题的总体。

面向对象的 CRC 方法从以下三个方面来表述对象。

（1）Class name（对象取名）。给对象取名，实际上就是对需要处理的问题空间中包含的不同性质的数据进行分类。在讨论设计模型、实现模型和用户模型时都要使用这些对象（所以称为面向对象的）。在设计早期仔细地选择对象的名字，用直观的语言表达如何组织系统，为以后设计的正确性奠定基础。

（2）Responsibilities（责任）。每个对象完成设计中的一个小目标的功能，它实现了整个系统状态的某一部分。这些责任可以用带活跃动词的短语来描述。就像给对象取名一样，仔细选择描述责任的词也是很有价值的设计活动。

（3）Collaborators（合作者）。每个对象可能会依赖其他对象来完成其责任。这个对象所依赖的对象集合称为它的合作者。重要的合作者能帮助设计者弄清楚怎样将责任在系统中分散。

CRC 设计方法将继承的决策推迟到设计后期甚至实现阶段。初学者如果首先过分地关注继承，那么他将花去太多的时间来得到比较完善的类分层，以致于忽略了用户的需求和实现过程的实际需要。CRC 方法将继承这个有价值的资源放入实现者的手中，实现者能用之来减少代码的行数。继承还有其他有效的使用，诸如协议规格或对象分类。不管怎样选择使用继承，推迟继承策略直到已将系统状态分解为对象之后，故保证能最好地利用继承。

CRC 方法也忽略了对象描述的问题，到了某一阶段，描述决策将显得很重要：是嵌入进去还是用指针指向，用堆栈实现还是用堆分配技术等等。有了稳定的责任分布会使这些决策比较容易，因为你已知道，每个对象所涉及的存储信息是什么。

如何判定“是否已经找到合适的对象？”，没有一定的准则，这里提出几条判定方法，仅供参考。较好的 CRC 对象应该具有如下特性：

（1）从实际比喻中取一个一致性的名字。好的名字常会使设计更清晰。例如，不要将一种画称为“Paint”，而另一种画称为“ColorValue”，要保持一致性的名字，不然会导致混乱。

（2）具有简明扼要的责任。随着对象设计的成熟，不应该用一个对象保存数据，而用另一个对象来处理这些数据。对象应具有责任。

（3）具有较小的、但实用的一组合作者。一个对象，如果能同所有的对象打交道，则认为该对象是全局性的象征。这属于过程式的思维，应该力图避免。分解这样的对象，或者把它们的责任分散到已存在的其他对象，或者创建几个新对象取代之。

（4）责任不要太多。一个对象承担的责任太多，就需要寻找更简明的方法来表述对象的责任。例如，创建其他对象来分担一些责任。

较差的 CRC 对象具有如下特征：

（1）没有任何责任。设计结束时，不承担任何计算负担的对象应该删除。

（2）用“Manager”取名的对象。像“Manager”、“Object”或“Thing”这样的词作为对象的名字，没有表达出任何语义信息。应消除这种“噪音”字，寻找更好的、能表达对象正在做什么的词为对象取名。例如，应使用 Process_Scheduler，而不用 ProcessManager；应使用 Figure，而不用 DrawingObject。

（3）使用词“has”、“hold”或“uses”来表达责任。这些词仅提供描述而不提供行为。表达责任的词应尽可能解释为什么其他对象还需维持。例如，在一个 Bitmap 对象中，可能