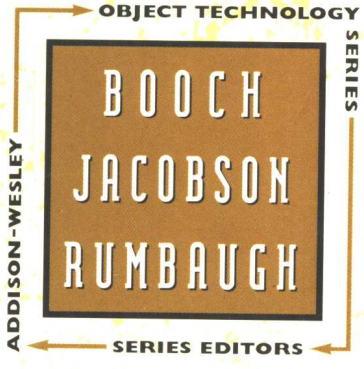
UML与面向对象设计影印丛书

UML面向对象设计基础

FUNDAMENTALS OF OBJECT-ORIENTED DESIGN IN UML

MEILIR PAGE-JONES 编著



科学出版社
www.sciencep.com

UML与面向对象设计影印丛书

UML 面向对象设计基础

Meilir Page-Jones 编著

科学出版社

北 京

内 容 简 介

UML 已成为描述面向对象设计符号的事实上的标准。本书介绍了面向对象软件设计的基本概念、符号表示、术语、准则以及原理等内容，其中第一部分(第 1 章和第 2 章)介绍了面向对象的基本概念以及面向对象编程的发展过程，第二部分(第 3 章至第 7 章)对 UML 进行了系统的介绍，第三部分(第 8 章至第 14 章)较深入地介绍面向对象设计的原理。最后一章(第 15 章)对软件构件的优缺点作了分析。

本书可供面向对象技术的程序员、设计人员、系统工程师或技术经理使用。

ISBN 0-201-69946-x

图字：01-2003-2542

图书在版编目(CIP)数据

UML 面向对象设计基础=Fundamentals of Object-Oriented Design in UML/（美）约翰斯（Johns,M.P.）著. —影印本. —北京：科学出版社，2003

ISBN 7-03-011409-4

Ⅰ.U… Ⅱ.约… Ⅲ.面向对象语言，UML—程序设计—英文 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2003）第 030819 号

策划编辑：李佩乾/责任编辑：李佩乾
责任印制：吕春珉/封面制作：东方人华平面设计室

科学出版社 出版
北京东黄城根北街16号
邮政编码:100717
http://www.sciencep.com

双青印刷厂 印刷
科学出版社发行 各地新华书店经销

*

2003 年 5 月第 一 版 开本：787×960 1/16
2003 年 5 月第一次印刷 印张：29 3/4
印数：1—2 000 字数：569 000

定价：50.00 元

（如有印装质量问题，我社负责调换<环伟>）

影印前言

随着计算机硬件性能的迅速提高和价格的持续下降，其应用范围也在不断扩大。交给计算机解决的问题也越来越难，越来越复杂。这就使得计算机软件变得越来越复杂和庞大。20 世纪 60 年代的软件危机使人们清醒地认识到按照工程化的方法组织软件开发的必要性。于是软件开发方法从 60 年代毫无工程性可言的手工作坊式开发，过渡到 70 年代结构化的分析设计方法、80 年代初的实体关系开发方法，直到面向对象的开发方法。

面向对象的软件开发方法是在结构化开发范型和实体关系开发范型的基础上发展而来的，它运用分类、封装、继承、消息等人类自然的思维机制，允许软件开发者处理更为复杂的问题域和其支持技术，在很大程度上缓解了软件危机。面向对象技术发端于程序设计语言，以后又向软件开发的早期阶段延伸，形成了面向对象的分析和设计。

20 世纪 80 年代末 90 年代初，先后出现了几十种面向对象的分析设计方法，其中，Booch, Coad/Yourdon、OMT 和 Jacobson 等方法得到了面向对象软件开发界的广泛认可。各种方法对许多面向对象的概念的理解不尽相同，即便概念相同，各自技术上的表示法也不同。通过 90 年代不同方法流派之间的争论，人们逐渐认识到不同的方法既有其容易解决的问题，又有其不容易解决的问题，彼此之间需要进行融合和借鉴；并且各种方法的表示也有很大的差异，不利于进一步的交流与协作。在这种情况下，统一建模语言(UML)于 90 年代中期应运而生。

UML 的产生离不开三位面向对象的方法论专家 G. Booch、J. Rumbaugh 和 I. Jacobson 的通力合作。他们从多种方法中吸收了大量有用的建模概念，使 UML 的概念和表示法在规模上超过了以往任何一种方法，并且提供了允许用户对语言做进一步扩展的机制。UML 使不同厂商开发的系统模型能够基于共同的概念、使用相同的表示法，呈现彼此一致的模型风格。1997 年 11 月 UML 被 OMG 组织正式采纳为标准的建模语言，并在随后的几年中迅速地发展为事实上的建模语言国际标准。

UML 在语法和语义的定义方面也做了大量的工作。以往各种关于面向对象方法的著作通常是以比较简单的方式定义其建模概念，而以主要篇幅给出过程指导，论述如何运用这些概念来进行开发。UML 则以一种建模语言的姿态出现，使用语言学中的一些技术来定义。尽管真正从语言学的角度看它还有许多缺陷，但它在这方面所做的努力却是以往的各种建模方法无法比拟的。

从 UML 的早期版本开始，便受到了计算机产业界的重视，OMG 的采纳和大公司的支持把它推上了实际上的工业标准的地位，使它拥有越来越多的用户。它被广泛地用

于应用领域和多种类型的系统建模，如管理信息系统、通信与控制系统、嵌入式实时系统、分布式系统、系统软件等。近几年还被运用于软件再工程、质量管理、过程管理、配置管理等方面。而且它的应用不仅仅限于计算机软件，还可用于非软件系统，例如硬件设计、业务处理流程、企业或事业单位的结构与行为建模，等等。

在 UML 陆续发布的几个版本中，逐步修正了前一个版本中的缺陷和错误。即将发布的 UML2.0 版本将是对 UML 的又一次重大的改进。将来的 UML 将向着语言家族化、可执行化、精确化等理念迈进，为软件产业的工程化提供更有力的支撑。

本丛书收录了与面向对象技术和 UML 有关的 12 本书，反映了面向对象技术最新的发展趋势以及 UML 的新的研究动态。其中涉及对面向对象建模理论研究与实践的有这样几本书：《面向对象系统架构及设计》主要讨论了面向对象的基本概念、静态设计、永久对象、动态设计、设计模式以及体系结构等近几年来面向对象技术领域中的新的理论知识与方法；《用 UML 进行用况对象建模》主要介绍了面向对象的需求阶段、分析阶段、设计阶段中用况模型的建立方法与技术；《高级用况建模》介绍了在建立用况模型中需要注意的高级的问题与技术；《UML 面向对象设计基础》则侧重于经典的面向对象理论知识的阐述。

涉及 UML 在特定领域的运用的有这样几本：《UML 实时系统开发》讨论了进行实时系统开发时需要对 UML 进行扩展的技术；《用 UML 构建 Web 应用程序》讨论了运用 UML 进行 Web 应用建模所应该注意的技术与方法；《面向对象系统测试：模型、视图与工具》介绍了将 UML 应用于面向对象的测试领域所应掌握的方法与工具；《对象、构件、框架与 UML 应用》讨论了如何运用 UML 对面向对象的新技术——构件-框架技术建模的方法策略。《UML 与 Visual Basic 应用程序开发》主要讨论了从 UML 模型到 Visual Basic 程序的建模与映射方法。

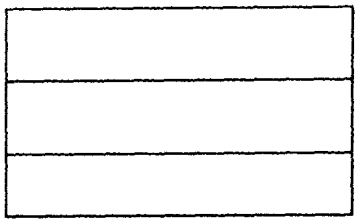
介绍面向对象编程技术的有两本书：《COM 高手心经》和《ATL 技术内幕》，深入探讨了面向对象的编程新技术——COM 和 ATL 技术的使用技巧与技术内幕。

还有一本《Executable UML 技术内幕》，这本书介绍了可执行 UML 的理念与其支持技术，使得模型的验证与模拟以及代码的自动生成成为可能，也代表着将来软件开发的一种新的模式。

总之，这套书所涉及的内容包含了对软件生命周期的全过程建模的方法与技术，同时也对近年来的热点领域建模技术、新型编程技术作了深入的介绍，有些内容已经涉及到了前沿领域。可以说，每一本都很经典。

有鉴于此，特向软件领域中不同程度的读者推荐这套书，供大家阅读、学习和研究。

北京大学计算机系　蒋严冰 博士



Foreword

Objects have become the ubiquitous building blocks of modern software, and object orientation is the pervasive paradigm of contemporary software-engineering practice. Books on object-oriented this or object-oriented that are discounted by the dozen, but when *What Every Programmer Should Know About Object-Oriented Design*, the first edition of this book, was published, it was immediately recognized as an original, insightful, and valuable contribution from one of the most consistently lucid thinkers and readable authors in software development today.

This newly revised and retitled second edition extends the foundation, expands the material, and updates the notation to create a reference of both immediate and lasting value. It is filled with fresh insights on object-oriented development, from the uses and abuses of inheritance, to how to model problematic data relationships in object classes. It is vintage Page-Jones, meaning up-to-the-minute and unflaggingly intelligent.

The author has been in the front-line trenches as a consultant and designer for decades, and the hard-won lessons show on every page of this book. I have been in the trenches with him, most recently collaborating on a very large-scale project with an initial use-case model of more than 340 use cases! He is, as the

reader will learn, above all else, a pragmatist whose attention to fundamentals and detail is reflected in his analysis and design work as well as in his writing.

The truth is, Meilir is a gifted teacher who has a knack for taking complex and often misunderstood ideas and casting a conceptual light on them that makes them stand out in sharp relief from the confusing shadows. He can take a barrowload of problems and wrap them up in a single archetypal example that makes it all seem so obvious that the rest of us are left wondering how we could ever have failed to see. What do you do when milking time arrives in the object-oriented dairy farm? Do you send a message to the **Cow** object to milk itself or a message to the **Milk** to un-cow itself? A moment's reflection and the need for an event manager to coordinate the milking becomes crystal clear. His clarifying examples, such as this one from a conference panel presentation or the "**Person** owns **Dog**" conundrum, have become part of the essential folklore of object orientation.

Indeed, this book aptly demonstrates how the long-established principles of sound design already in wide use by practicing professionals can carry over into and be adapted for developing object-oriented systems in the most advanced new languages and challenging contexts. Building on such fundamentals, the book maintains a relentlessly pragmatic focus based on real-world experience, distilling the essence of that experience into compact examples that will guide the developer, whether novice or old hand, toward better object-oriented software solutions.

Meilir draws on extensive experience with object-oriented development, as a consultant, as a teacher, and as a methodologist. He was codeveloper of the Synthesis method, one of the early systematic approaches to object-oriented analysis and design, and we were collaborators on the creation of the influential Uniform Object Notation, whose features can be found today reflected and incorporated into numerous object-oriented methods and notations. The legacy of our work can even be recognized in the Unified Modeling Language (UML) that has been adopted as a de facto industry standard and is used to illustrate and clarify examples throughout this book.

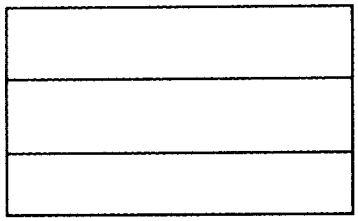
Here you will find everything you need to begin to master the fundamentals of object-oriented design. Not only are the basic techniques for designing and building with objects explained with exceptional clarity, but they are illustrated with abundant examples, and elaborated with discussions of the do's and don'ts of good object-oriented systems. The rest is up to you.

September 1999
Rowley, Massachusetts

Larry Constantine
Coauthor of *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*
(Reading, Mass.: Addison-Wesley, 1999)

"You say you want some evolution.
Well, you know, I'm doing what I can."

—Charles Darwin, *On the Origin of Species*



Preface

People who reviewed this book in its draft form had several questions for me, questions that perhaps you share. Let me address some of them.

I'm a programmer. Why should I care about design?

Everyone who writes code also *designs* code—either well or badly, either consciously or unconsciously. My goal in writing this book is to encourage O.O. professionals—and their number increases annually—to create good object-oriented designs consciously and prior to coding. To this end, I introduce notation, principles, and terminology that you and your colleagues can use to evaluate your designs and to discuss them meaningfully with one another.

Will this book teach me an O.O. programming language?

No. Although I occasionally swoop down close to code, this isn't a book on object-oriented programming.

But if I'm learning an object-oriented language, will this book help?

Yes, it will. If you don't currently know an object-oriented programming language, you can begin your object-oriented knowledge with Chapter 1. Knowing the key concepts of object orientation will speed your learning an object-oriented language and, I hope, boost your morale as you move into unfamiliar territory. The later chapters of the book, on sound design, will also help you in getting your early programs to work successfully.

On the other hand, if you're already an experienced object-oriented programmer, you can use Parts II and III of the book to enhance the design skills that are vital to your being a rounded, professional software designer or programmer.

Why aren't the code examples in this book in C++?

I've written the code in this book in a language of my own devising, which is a blend of four popular languages: C++, Eiffel, Java, and Smalltalk. I did this because there are two kinds of programmers: those who are fluent in C++ and those who aren't. If you're a C++ aficionado, then you'll find the code a breeze to translate into C++. If you're not familiar with C++, then you might have found the language's arcane syntax distracting. Some examples are given in Java because it's more accessible to a non-Java programmer than C++ is to a non-C++ programmer. I'd like you to feel welcome in this book whatever your programming language might be.

Why isn't this book devoted to the design of windows, icons, and menus?

There are two reasons: First, I don't believe that object orientation is useful only for the design of graphical user interfaces. Second, there are many books on the market devoted solely to the topic of object-oriented window design. I want this book to cover topics that are not well covered by other object-oriented books. However, in Chapter 7, I offer some notation for window-navigation design.

Is this book about a methodology?

No. As you know, a development methodology contains much more than design. For example, there's requirements analysis, library management, and so on. Also, a true methodology needs to explain how the various development activities fit together. A lot of stuff!

So, instead of turning out a book as diffuse as many other books on object orientation, I decided to focus on a single topic: object-oriented design.

You've said a lot about what this book isn't about. What is it about?

It's about the fundamental ideas, notation, terminology, criteria, and principles of object-oriented software design. Object-oriented software is software that comprises objects and the classes to which they belong. An object is a software construct in which operations (which are like functions or procedures) are organized around a set of variables (which are like data). A class implements a type that defines the group of objects belonging to that class.

The above modest sentences hold some surprising implications for software designers and programmers, implications that arise from the design concepts of inheritance, polymorphism, and second-order design. But, since you asked a specific question, let me give you a specific answer.

Part I of the book (Chapters 1 and 2) provides an introduction to object orientation. Chapter 1 summarizes the key concepts and demystifies "polymorphism," "genericity," and all the other O.O. jargon. Chapter 2 sets object orientation into the framework of previous developments in software. If you're already familiar with object orientation (perhaps by having programmed in an object-oriented language), then you can skip or skim Part I.

Part II (Chapters 3 to 7) covers Unified Modeling Language (UML), which has become the *de facto* standard notation for depicting object-oriented design. In passing, Part II also illustrates many of the structures that you find in object-oriented systems. Chapter 3 introduces UML for depicting classes, along with their attributes and operations. Chapter 4 covers UML for associations, aggregate and composite objects, and hierarchies of subclasses and superclasses. Chapter 5 sets out UML for messages (both sequential and asynchronous), while Chapter 6 covers UML for state diagrams. Chapter 7 reviews UML for system architecture and the windows that form a human interface.

Part III (Chapters 8 to 14) covers object-oriented design principles in some depth. Chapter 8 sets the scene with the crucial notions of connascence and level-2 encapsulation. Chapter 9 explores the various domains that "classes come from" and describes different degrees of class cohesion. Chapters 10 and 11 are the central pillars of Part III, applying the concepts of state-space and behavior to assess when a class hierarchy is both sound and extendable.

Chapter 12 offers some light relief, as it examines designs taken from real projects, including both the subtle and the absurd. (Chapter 12 is really about the dangers of abusing inheritance and polymorphism.) Chapter 13 looks at some ways of organizing operations within a given class, and it explains design techniques, such as mix-in classes and operation rings, that will improve class reusability and maintainability.

Chapter 14 takes a stab at the old question: "What makes a *good* class?" In answering this question, Chapter 14 describes the various kinds of class interface, ranging from the horrid to the sublime. A class with an exemplary interface will be a worthy implementation of an abstract data-type. If the class also obeys the fundamental principles laid out in earlier chapters, then it will be as robust, reliable, extensible, reusable, and maintainable as a class can ever be.

Chapter 15 rounds off the book by examining the characteristics, together with the advantages and disadvantages, of software components. In tracing the development of an object-oriented component for a business application, I recall some of the object-oriented principles of the previous chapters.

Although I've added plenty of examples, diagrams, and exercises to reinforce what I say in the main text, I must admit that the material in Part III gets tough at times. Nevertheless, I decided not to trivialize or dilute important issues. Some aspects of object-oriented design *are* difficult and to suggest otherwise would be misleading.

Does this book cover everything *in object-oriented design?*

I very much doubt it. Each day, I learn more about object orientation, and I'm sure you do, too. Indeed, it would be a dull world if a single book could tell us everything about object-oriented design and leave us with nothing more to learn. And not everything in this book may be completely true! I certainly changed my mind about one or two things after writing my previous books, as I became older and wiser—well older, anyway.

So, although I think that I've covered many important design principles in this book, if you're serious about object orientation you should continue to read as much as you can and always challenge what you read.

Do you offer courses on object-oriented design?

Yes. My firm, Wayland Systems, offers several courses on object-oriented topics. Our curriculum continually changes, so check out **www.waysys.com** for our latest offerings.

Bottom-line, as they say: Is this book for me?

What kind of question is that? You expect me to say, "No!"? But seriously, folks, this book's for you if you are—or are about to become—a programmer, designer, systems engineer, or technical manager on a project using object-oriented techniques. Even if you're a beginner to object orientation, you can glean a lot from this book by reading Part I, practicing some object-oriented programming, and then returning to Parts II and III.

You should also read this book if you're a university student or professional programmer who has mastered the techniques of standard procedural programming and is looking for wider horizons. Much of the book's material is suitable for a final-year computer-science or software-engineering course in object orientation.

But, whatever your role in life, I hope that you enjoy this book and find it useful. Good luck!

September 1999
Bellevue, Washington
meilir@waysys.com

Meilir Page-Jones

Contents