经 典 原 版 书 库

设计模式

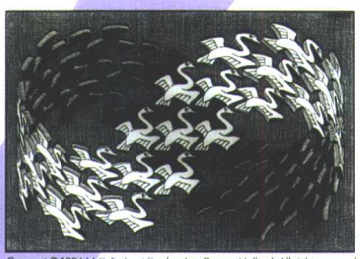
可复用面向对象软件的基础

（英文版）

Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

（美）Erich Gamma Richard Helm Ralph Johnson John Vlissides 著

机械工业出版社
China Machine Press

Pearson Education
培生教育出版集团

经典原版书库

(英文版)

设计模式

可复用面向对象软件的基础

Design Patterns

Elements of Reusable Object-Oriented Software

(美) Erich Gamma Richard Helm
Ralph Johnson John Vlissides 著

机械工业出版社
China Machine Press

本书版权登记号：图字：01-2001-5020

图书在版编目（CIP）数据

设计模式：可复用面向对象软件的基础（英文版）/（美）伽玛（Gamma, E.）等著. - 北京：机械工业出版社，2002.3

（经典原版书库）

ISBN 7-111-09507-3

Ⅰ.设… Ⅱ.伽… Ⅲ.面向对象语言-程序设计-英文 Ⅳ.TP312

中国版本图书馆CIP数据核字（2001）第081806号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：华 章

北京昌平奔腾印刷厂印刷 · 新华书店北京发行所发行

2002年3月第1版第1次印刷

850mm × 1168mm 1/32 · 13.25印张

印数：0 001-5 000 册

定价：38.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专诚为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：针对本科生的核心课程，剔抉外版菁华而成“国外经典教材”系列；对影印版的教材，则单独开辟出“经典原版书库”；定位在高级教程和专业参考的“计算机科学丛书”还将保持原来的风格，继续出版新的品种。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

“经典原版书库”是响应教育部提出的使用原版国外教材的号召，为国内高校的计算机教学度身订造的。在广泛地征求并听取丛书的“专家指导委员会”的意见后，我们最终选定了这30多种篇幅内容适度、讲解鞭辟入里的教材，其中的大部分已经被M.I.T.、Stanford、U.C. Berkley、C.M.U.等世界名牌大学采用。丛书不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzedu@hzbook.com
联系电话：（010）68995265
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周克定	周傲英	孟小峰	岳丽华	范　明
郑国梁	施伯乐	钟玉琢	唐世渭	袁崇义
高传善	梅　宏	程　旭	程时端	谢希仁
裘宗燕	戴　葵			

To Karin
—E.G.

To Sylvie
—R.H.

To Faith
—R.J.

To Dru Ann and Matthew
Joshua 24:15b
—J.V.

Praise for *Design Patterns: Elements of Reusable Object-Oriented Software*

"This is one of the best written and wonderfully insightful books that I have read in a great long while...this book establishes the legitimacy of patterns in the best way: not by argument but by example."

— **Stan Lippman,** ***C++ Report***

"...this new book by Gamma, Helm, Johnson, and Vlissides promises to have an important and lasting impact on the discipline of software design. Because *Design Patterns* bills itself as being concerned with object-oriented software alone, I fear that software developers outside the object community may ignore it. This would be a shame. This book has something for everyone who designs software. All software designers use patterns; understanding better the reusable abstractions of our work can only make us better at it."

— **Tom DeMarco,** ***IEEE Software***

"Overall, I think this book represents an extremely valuable and unique contribution to the field because it captures a wealth of object-oriented design experience in a compact and reusable form. This book is certainly one that I shall turn to often in search of powerful object-oriented design ideas; after all, that's what reuse is all about, isn't it?"

— **Sanjiv Gossain,** ***Journal of Object-Oriented Programming***

"This much-anticipated book lives up to its full year of advance buzz. The metaphor is of an architect's pattern book filled with time-tested, usable designs. The authors have chosen 23 patterns from decades of object-oriented experience. The brilliance of the book lies in the discipline represented by that number. Give a copy of *Design Patterns* to every good programmer you know who wants to be better."

— **Larry O'Brien,** ***Software Development***

"The simple fact of the matter is that patterns have the potential to permanently alter the software engineering field, catapulting it into the realm of true elegant design. Of the books to date on this subject, *Design Patterns* is far and away the best. It is a book to be read, studied, internalized, and loved. The book will forever change the way you view software."

— **Steve Bilow,** ***Journal of Object-Oriented Programming***

"*Design Patterns* is a powerful book. After a modest investment of time with it, most C++ programmers will be able to start applying its "patterns" to produce better software. This book delivers intellectual leverage: concrete tools that help us think and express ourselves more effectively. It may fundamentally change the way you think about programming.

— **Tom Cargill,** ***C++ Report***

Preface

This book isn't an introduction to object-oriented technology or design. Many books already do a good job of that. This book assumes you are reasonably proficient in at least one object-oriented programming language, and you should have some experience in object-oriented design as well. You definitely shouldn't have to rush to the nearest dictionary the moment we mention "types" and "polymorphism," or "interface" as opposed to "implementation" inheritance.

On the other hand, this isn't an advanced technical treatise either. It's a book of **design patterns** that describes simple and elegant solutions to specific problems in object-oriented software design. Design patterns capture solutions that have developed and evolved over time. Hence they aren't the designs people tend to generate initially. They reflect untold redesign and recoding as developers have struggled for greater reuse and flexibility in their software. Design patterns capture these solutions in a succinct and easily applied form.

The design patterns require neither unusual language features nor amazing programming tricks with which to astound your friends and managers. All can be implemented in standard object-oriented languages, though they might take a little more work than *ad hoc* solutions. But the extra effort invariably pays dividends in increased flexibility and reusability.

Once you understand the design patterns and have had an "Aha!" (and not just a "Huh?") experience with them, you won't ever think about object-oriented design in the same way. You'll have insights that can make your own designs more flexible, modular, reusable, and understandable—which is why you're interested in object-oriented technology in the first place, right?

A word of warning and encouragement: Don't worry if you don't understand this book completely on the first reading. We didn't understand it all on the first writing! Remember that this isn't a book to read once and put on a shelf. We hope you'll find yourself referring to it again and again for design insights and for inspiration.

This book has had a long gestation. It has seen four countries, three of its authors' marriages, and the birth of two (unrelated) offspring. Many people have had a part in its development. Special thanks are due Bruce Anderson, Kent Beck, and André Weinand for their inspiration and advice. We also thank those who reviewed drafts

of the manuscript: Roger Bielefeld, Grady Booch, Tom Cargill, Marshall Cline, Ralph Hyre, Brian Kernighan, Thomas Laliberty, Mark Lorenz, Arthur Riel, Doug Schmidt, Clovis Tondo, Steve Vinoski, and Rebecca Wirfs-Brock. We are also grateful to the team at Addison-Wesley for their help and patience: Kate Habib, Tiffany Moore, Lisa Raffaele, Pradeepa Siva, and John Wait. Special thanks to Carl Kessler, Danny Sabbah, and Mark Wegman at IBM Research for their unflagging support of this work.

Last but certainly not least, we thank everyone on the Internet and points beyond who commented on versions of the patterns, offered encouraging words, and told us that what we were doing was worthwhile. These people include but are not limited to Jon Avotins, Steve Berczuk, Julian Berdych, Matthias Bohlen, John Brant, Allan Clarke, Paul Chisholm, Jens Coldewey, Dave Collins, Jim Coplien, Don Dwiggins, Gabriele Elia, Doug Felt, Brian Foote, Denis Fortin, Ward Harold, Hermann Hueni, Nayeem Islam, Bikramjit Kalra, Paul Keefer, Thomas Kofler, Doug Lea, Dan LaLiberte, James Long, Ann Louise Luu, Pundi Madhavan, Brian Marick, Robert Martin, Dave McComb, Carl McConnell, Christine Mingins, Hanspeter Mössenböck, Eric Newton, Marianne Ozkan, Roxsan Payette, Larry Podmolik, George Radin, Sita Ramakrishnan, Russ Ramirez, Alexander Ran, Dirk Riehle, Bryan Rosenburg, Aamod Sane, Duri Schmidt, Robert Seidl, Xin Shu, and Bill Walker.

We don't consider this collection of design patterns complete and static; it's more a recording of our current thoughts on design. We welcome comments on it, whether criticisms of our examples, references and known uses we've missed, or design patterns we should have included. You can write us care of Addison-Wesley, or send electronic mail to `design-patterns@cs.uiuc.edu`. You can also obtain softcopy for the code in the Sample Code sections by sending the message "send design pattern source" to `design-patterns-source@cs.uiuc.edu`. And now there's a Web page at `http://st-www.cs.uiuc.edu/users/patterns/DPBook/DPBook.html` for late-breaking information and updates.

Mountain View, California E.G.

Montreal, Quebec R.H.

Urbana, Illinois R.J.

Hawthorne, New York J.V.

August 1994

Foreword

All well-structured object-oriented architectures are full of patterns. Indeed, one of the ways that I measure the quality of an object-oriented system is to judge whether or not its developers have paid careful attention to the common collaborations among its objects. Focusing on such mechanisms during a system's development can yield an architecture that is smaller, simpler, and far more understandable than if these patterns are ignored.

The importance of patterns in crafting complex systems has been long recognized in other disciplines. In particular, Christopher Alexander and his colleagues were perhaps the first to propose the idea of using a pattern language to architect buildings and cities. His ideas and the contributions of others have now taken root in the object-oriented software community. In short, the concept of the design pattern in software provides a key to helping developers leverage the expertise of other skilled architects.

In this book, Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides introduce the principles of design patterns and then offer a catalog of such patterns. Thus, this book makes two important contributions. First, it shows the role that patterns can play in architecting complex systems. Second, it provides a very pragmatic reference to a set of well-engineered patterns that the practicing developer can apply to crafting his or her own specific applications.

I'm honored to have had the opportunity to work directly with some of the authors of this book in architectural design efforts. I have learned much from them, and I suspect that in reading this book, you will also.

Grady Booch

Chief Scientist, Rational Software Corporation

Guide to Readers

This book has two main parts. The first part (Chapters 1 and 2) describes what design patterns are and how they help you design object-oriented software. It includes a design case study that demonstrates how design patterns apply in practice. The second part of the book (Chapters 3, 4, and 5) is a catalog of the actual design patterns.

The catalog makes up the majority of the book. Its chapters divide the design patterns into three types: creational, structural, and behavioral. You can use the catalog in several ways. You can read the catalog from start to finish, or you can just browse from pattern to pattern. Another approach is to study one of the chapters. That will help you see how closely related patterns distinguish themselves.

You can use the references between the patterns as a logical route through the catalog. This approach will give you insight into how patterns relate to each other, how they can be combined with other patterns, and which patterns work well together. Figure 1.1 (page 12) depicts these references graphically.

Yet another way to read the catalog is to use a more problem-directed approach. Skip to Section 1.6 (page 24) to read about some common problems in designing reusable object-oriented software; then read the patterns that address these problems. Some people read the catalog through first and *then* use a problem-directed approach to apply the patterns to their projects.

If you aren't an experienced object-oriented designer, then start with the simplest and most common patterns:

- Abstract Factory (page 87)
- Adapter (139)
- Composite (163)
- Decorator (175)
- Factory Method (107)
- Observer (293)
- Strategy (315)
- Template Method (325)

It's hard to find an object-oriented system that doesn't use at least a couple of these patterns, and large systems use nearly all of them. This subset will help you understand design patterns in particular and good object-oriented design in general.

Contents

Chapter 1

Introduction

Designing object-oriented software is hard, and designing *reusable* object-oriented software is even harder. You must find pertinent objects, factor them into classes at the right granularity, define class interfaces and inheritance hierarchies, and establish key relationships among them. Your design should be specific to the problem at hand but also general enough to address future problems and requirements. You also want to avoid redesign, or at least minimize it. Experienced object-oriented designers will tell you that a reusable and flexible design is difficult if not impossible to get "right" the first time. Before a design is finished, they usually try to reuse it several times, modifying it each time.

Yet experienced object-oriented designers do make good designs. Meanwhile new designers are overwhelmed by the options available and tend to fall back on non-object-oriented techniques they've used before. It takes a long time for novices to learn what good object-oriented design is all about. Experienced designers evidently know something inexperienced ones don't. What is it?

One thing expert designers know *not* to do is solve every problem from first principles. Rather, they reuse solutions that have worked for them in the past. When they find a good solution, they use it again and again. Such experience is part of what makes them experts. Consequently, you'll find recurring patterns of classes and communicating objects in many object-oriented systems. These patterns solve specific design problems and make object-oriented designs more flexible, elegant, and ultimately reusable. They help designers reuse successful designs by basing new designs on prior experience. A designer who is familiar with such patterns can apply them immediately to design problems without having to rediscover them.

An analogy will help illustrate the point. Novelists and playwrights rarely design their plots from scratch. Instead, they follow patterns like "Tragically Flawed Hero" (Macbeth, Hamlet, etc.) or "The Romantic Novel" (countless romance novels). In the same way, object-oriented designers follow patterns like "represent states with objects"

and "decorate objects so you can easily add/remove features." Once you know the pattern, a lot of design decisions follow automatically.

We all know the value of design experience. How many times have you had design *déjà-vu*—that feeling that you've solved a problem before but not knowing exactly where or how? If you could remember the details of the previous problem and how you solved it, then you could reuse the experience instead of rediscovering it. However, we don't do a good job of recording experience in software design for others to use.

The purpose of this book is to record experience in designing object-oriented software as **design patterns**. Each design pattern systematically names, explains, and evaluates an important and recurring design in object-oriented systems. Our goal is to capture design experience in a form that people can use effectively. To this end we have documented some of the most important design patterns and present them as a catalog.

Design patterns make it easier to reuse successful designs and architectures. Expressing proven techniques as design patterns makes them more accessible to developers of new systems. Design patterns help you choose design alternatives that make a system reusable and avoid alternatives that compromise reusability. Design patterns can even improve the documentation and maintenance of existing systems by furnishing an explicit specification of class and object interactions and their underlying intent. Put simply, design patterns help a designer get a design "right" faster.

None of the design patterns in this book describes new or unproven designs. We have included only designs that have been applied more than once in different systems. Most of these designs have never been documented before. They are either part of the folklore of the object-oriented community or are elements of some successful object-oriented systems—neither of which is easy for novice designers to learn from. So although these designs aren't new, we capture them in a new and accessible way: as a catalog of design patterns having a consistent format.

Despite the book's size, the design patterns in it capture only a fraction of what an expert might know. It doesn't have any patterns dealing with concurrency or distributed programming or real-time programming. It doesn't have any application domain-specific patterns. It doesn't tell you how to build user interfaces, how to write device drivers, or how to use an object-oriented database. Each of these areas has its own patterns, and it would be worthwhile for someone to catalog those too.

1.1 What Is a Design Pattern?

Christopher Alexander says, "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice" [AIS+77, page *x*]. Even though Alexander was talking about patterns in buildings and towns, what he says is true about object-oriented design patterns. Our solutions are expressed in terms of objects and interfaces instead of walls