国 外 计 算 机 科 学 教 材 系 列

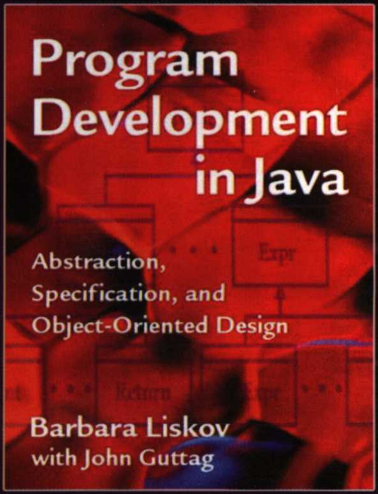
PEARSON Addison Wesley

程序开发原理

——抽象、规格与面向对象设计

Program Development in Java

Abstraction, Specification, and Object-Oriented Design


Program Development in Java
Abstraction, Specification, and Object-Oriented Design
Barbara Liskov
with John Guttag

英文版

[美] Barbara Liskov
John Guttag 著

電子工業出版社
Publishing House of Electronics Industry
http://www.phei.com.cn

[illegible]

程序开发原理

——抽象、规格与面向对象设计

Program Development in Java

Abstraction, Specification, and Object-Oriented Design

[illegible]

[美] [illegible] 著
[illegible]

[illegible]

国外计算机科学教材系列

程序开发原理

—— 抽象、规格与面向对象设计

（英文版）

Program Development In Java

Abstraction, Specification, and Object-Oriented Design

［美］ Barbara Liskov
John Guttag 著

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内 容 简 介

本书由著名编程理论专家所著，是美国麻省理工学院电子工程与计算机科学系的编程实践课程教材。书中讨论了怎样构建具有高可靠性、易于维护和快速修改的软件的开发方法，强调了软件的模块化开发思想，用丰富的实例告诉读者怎样进行模块化并合理地组织各种模块以构成大型软件系统的过程。该书的前8章首次提出编程领域中的一些十分有用的抽象概念，如过程抽象、迭代抽象以及最重要的数据抽象等。此外，本书通过大量的例子，用非形式化的规范来详细定义这些数据抽象，描述模块所需完成的任务，并定义了模块所需的性能边界条件。该书的后7章主要讲述了怎样利用抽象构建大型软件，主要侧重于软件工程的内容，基于类型层次结构提出对于数据抽象的调试、测试、需求分析、自顶向下和迭代的开发过程，还简要介绍了设计模式的概念。

这是一本传授思想的书籍，能使读者透过现象看到本质，从而掌握编写程序的关键。本书非常适合作为软件学院的教材，在低年级即可培养学生对于事物的抽象能力。此外，本书也非常适合软件开发人员参考。

版权贸易合同登记号　图字：01-2006-3398

图书在版编目（CIP）数据

程序开发原理——抽象、规格与面向对象设计 = Program Development In Java: Abstraction, Specification, and Object-Oriented Design /（美）利斯科瓦（Liskov, B.）等著. - 北京：电子工业出版社，2006.6
（国外计算机科学教材系列）
ISBN 7-121-02693-7

Ⅰ.程… Ⅱ.利… Ⅲ.软件开发 - 教材 - 英文　Ⅳ.TP311.52

中国版本图书馆CIP数据核字（2006）第052410号

责任编辑：马　岚
印　　刷：北京市顺义兴华印刷厂
出版发行：电子工业出版社
　　　　　北京市海淀区万寿路173信箱　邮编：100036
经　　销：各地新华书店
开　　本：787 × 980　1/16　　印张：29　　字数：650千字
印　　次：2006年6月第1次印刷
定　　价：58.00元

读者意见调查表

感谢您对电子工业出版社的支持！
为帮助我们进步，请将您的宝贵意见填于下表并寄回我们。

您购买的出版物名称	
先进性和实用性	□很好 □好 □一般 □不太好 □差
图书文字可读性 （光盘使用方便性）	□很好 □好 □一般 □不太好 □差 （□很好 □好 □一般 □不太好 □差）
图书篇幅适宜度 （光盘界面设计）	□很合适 □合适 □一般 □不合适 □差 （□很好 □好 □一般 □不太好 □差）
出版物中差错	□极少 □较少 □一般 □较多 □太多
图书封面（光盘盘面及包装）设计水平	□很好 □好 □一般 □不太好 □差
图书（光盘盘面及包装）印刷装订质量	□很好 □好 □一般 □不太好 □差
纸张质量（光盘材质）	□很好 □好 □一般 □不太好 □差
定价	□很便宜 □便宜 □合理 □贵 □太贵
对宣传工作的感觉	□很好 □好 □一般 □不好 □差
对服务质量的感觉	□很好 □好 □一般 □不好 □差
从何处获取出版物信息	□书目报 □电子社宣传材料 □书店 □他人转告 □网站 □报刊
您认为电子工业出版社应改进的方面	□先进性和实用性 □文字可读性（光盘使用方便性） □篇幅适宜度（光盘界面设计） □出版物中差错 □设计水平 □印刷装订质量 □纸张质量（光盘材质） □定价 □宣传工作 □服务质量
您的具体意见或建议	

读者姓名： 联系方式：

从事工作：□技术研发 □技术管理 □经营管理 □行政管理 □教育培训 □在校学习

表格寄回 邮寄地址：北京市万寿路 173 信箱电子工业出版社外版室 邮政编码：100036 收信人：冯小贝
传真：（010）88254560 收件人：冯小贝
电子信箱：fengxiaobei@phei.com.cn

出 版 说 明

21世纪初的5至10年是我国国民经济和社会发展的重要时期，也是信息产业快速发展的关键时期。在我国加入WTO后的今天，培养一支适应国际化竞争的一流IT人才队伍是我国高等教育的重要任务之一。信息科学和技术方面人才的优劣与多寡，是我国面对国际竞争时成败的关键因素。

当前，正值我国高等教育特别是信息科学领域的教育调整、变革的重大时期，为使我国教育体制与国际化接轨，有条件的高等院校正在为某些信息学科和技术课程使用国外优秀教材和优秀原版教材，以使我国在计算机教学上尽快赶上国际先进水平。

电子工业出版社秉承多年来引进国外优秀图书的经验，翻译出版了“国外计算机科学教材系列”丛书，这套教材覆盖学科范围广、领域宽、层次多，既有本科专业课程教材，也有研究生课程教材，以适应不同院系、不同专业、不同层次的师生对教材的需求，广大师生可自由选择和自由组合使用。这些教材涉及的学科方向包括网络与通信、操作系统、计算机组织与结构、算法与数据结构、数据库与信息处理、编程语言、图形图像与多媒体、软件工程等。同时，我们也适当引进了一些优秀英文原版教材，本着翻译版本和英文原版并重的原则，对重点图书既提供英文原版又提供相应的翻译版本。

在图书选题上，我们大都选择国外著名出版公司出版的高校教材，如Pearson Education培生教育出版集团、麦格劳－希尔教育出版集团、麻省理工学院出版社、剑桥大学出版社等。撰写教材的许多作者都是蜚声世界的教授、学者，如道格拉斯·科默(Douglas E. Comer)、威廉·斯托林斯(William Stallings)、哈维·戴特尔(Harvey M. Deitel)、尤利斯·布莱克(Uyless Black)等。

为确保教材的选题质量和翻译质量，我们约请了清华大学、北京大学、北京航空航天大学、复旦大学、上海交通大学、南京大学、浙江大学、哈尔滨工业大学、华中科技大学、西安交通大学、国防科学技术大学、解放军理工大学等著名高校的教授和骨干教师参与了本系列教材的选题、翻译和审校工作。他们中既有讲授同类教材的骨干教师、博士，也有积累了几十年教学经验的老教授和博士生导师。

在该系列教材的选题、翻译和编辑加工过程中，为提高教材质量，我们做了大量细致的工作，包括对所选教材进行全面论证；选择编辑时力求达到专业对口；对排版、印制质量进行严格把关。对于英文教材中出现的错误，我们通过与作者联络和网上下载勘误表等方式，逐一进行了修订。

此外，我们还将与国外著名出版公司合作，提供一些教材的教学支持资料，希望能为授课老师提供帮助。今后，我们将继续加强与各高校教师的密切联系，为广大师生引进更多的国外优秀教材和参考书，为我国计算机科学教学体系与国际教学体系的接轨做出努力。

电子工业出版社

教材出版委员会

导　读

由美国麻省理工学院Liskov教授和Guttag教授合著的这本书，以简洁生动的语言，清晰流畅的文笔，概括而精练地介绍了抽象、规格和面向对象设计等重要概念，明确展现了以规格为基石的软件开发系统，填补了技术和教育之间缺失的一环。软件教育界认为，在许多软件开发书籍中，本书极具前瞻性和综合思维，凝聚了Liskov教授广博的知识与几十年在软件开发研究方面的经验，其敏锐的观察力和深入浅出的分析，使读者茅塞顿开。应该说，本书是一部软件教育中里程碑式的著作。

按照开放式课程计划，美国麻省理工学院已有900门课程上网，并有望在2007年达到1800门课程上网，本书作为该校电子工程与计算机科学学科的一门核心课程（6.170）的教材，其所有课程资料现已共享于Internet上。因“Liskov替换原则”而知名的Barbara Liskov教授，获得过2004年度冯·诺伊曼奖。在20世纪70年代，Liskov教授使抽象数据结构（ADT）的理论和实现有了重大进展，奠定了面向对象的基础。Liskov教授是美国工程院和艺术科学院的双院士，她在麻省理工学院的多年教学经验以及在人工智能实验室的研究成果，有力地保证了该课程的教学质量。

软件工程不仅是具有战略意义的学科，还提供了自由而广阔的想像空间，其工作的技术难度和艺术性与过去相比已有很大相同，这对年轻一代软件工程师的修养提出了更高的要求。脱出定势思维的桎梏，才能激发自身的无限潜力。正如素描是一切绘画的基础，是学习美术的过程中必须经过的一个阶段（学习美术必须打好素描基础），又如巴赫复调作品是钢琴专业的必修课程，程序开发原理是软件专业工作者必须坚实打好的专业基础。软件工程师的抽象化能力将决定其对软件理解的深度。

软件技术进步了，而软件教育没能跟上来，因此中间出现了一个缺口。学术与企业必须寻找新的平衡点和结合点。在充分理解和认识到程序开发原理的重要性之后，如何把握程序开发原理中的思想和内涵，较好地消化吸收并准确地将教师自己的理解传授给学生，这一问题就明显地摆在了我们这些教师的面前。

超过20年的实践活动表明，大部分软件的规格构建依靠的是非形式化方法的运用。所谓非形式化方法，指的是主要以自然语言来表示、表述并诠释问题，而不采用形式化研究的一种方法。相比于形式化方法，它不需要建立形式系统。自然语言和抽象的数学形式化语言之间存在一条鸿沟，现在尚未找到将两者转换的理想途径。非形式化方法的重要性是不言而喻的，即使在规格的形式化系统构建过程中，也离不开对非形式化方法的运用。

事实上，本书的副标题所谈及的抽象、规格与面向对象设计，比主标题更精确地描述了本书的内涵。它解释了如何进行抽象化，如何设计并对其进行描述，如何根据一种面向对象语言来具体表

达。此外，本书还阐述了怎样把复杂问题分解成较小的子问题并把一个大系统组织成抽象化集合的方法。软件设计的一项重要任务就是把一个程序分解成几个部分。

本书的第一部分（第1章到第8章）的重点集中在抽象机制上。这些章节讨论程序和异常、过程抽象、数据抽象、迭代抽象、多态抽象以及类型层次。这部分内容回答了一些问题，例如什么是抽象化？怎样进行抽象？如何描述抽象？如何实现抽象？如何把它们组织成抽象集合？如何组成类型层次？在关于类型层次的章节中，可以看到以Liskov命名的子类/父类关系替换原则，简称LSP（Liskov替换原则）。本书的第一部分描述了两个用于理解抽象数据类型的工具：表示不变式（representation invariant）和抽象函数（abstraction function）。Liskov替换原则是继承复用的基石，只有当子类可以替换父类，且软件的功能不受影响时，才算真正地被复用，而子类也才能在父类的基础上添加新的行为。教师可以利用这些具有思辨性的章节，让学生认真解读，把握难点，努力突破，朝着各种方向去探寻各种不同解决途径和答案，以提高学生的抽象思维能力。

本书的第二部分（第9章到第14章）描述的是从需求分析开始，经过反复迭代阶段直至测试的整个开发过程。规格抽象机制涵盖了软件开发过程的整个生命周期。在计划、设计、编码实现、测试和维护方面，从根本上影响了大型软件的开发。

本书的第三部分（只包括第15章）解释了设计模式。在20世纪90年代中期，设计模式的思想在面向对象领域中吸引了相当多的注意力。设计模式是建筑上的观念，并可广泛适用于应用领域。这类模式具有精确的理论和不太复杂的定义，它们的本质性描述与现实类似，对于操作实现具有明显的意义，乃是必需的。

本书中还讨论了可变对象和不可变对象之间的区别，但Java并不是表达不可变性观念的理想语言，此类重要议题在面向对象领域里时常被忽略。

Liskov风格是Liskov提出的基本规格书写模式。它提供一个有系统的习惯。富有经验的程序员时常直觉地这样组织其概览（overview）。概览给出了对数据抽象的概要描述。在过程抽象化阶段，Liskov风格使用需求格式（requires clause）阐述了约束条件，而抽象就是在这些约束条件下被定义的。修改格式（modifies clause）列出了被过程修改了的所有输入的名称（也包括隐式输入）。如果某些输入被修改了，就可以说这个过程产生了副作用（side effect）。当没有输入被修改时，可以省略修改格式。不存在修改格式就意味着没有输入被修改。最后，结果格式（effects clause）定义产生了哪种输出，同时也必须定义对象作为输入。在数据抽象化阶段，操作分为4种：生成了新的对象的创建者，根据旧对象创建新对象的生产者（producer），修改其对象的状态的改变者（mutator），提供有关其对象的状态信息的观察者（observer）。

在异常处理阶段中，有时一个异常会被反射（reflect）到其他层面上。其他的可能方法是调用者屏蔽（mask）异常，处理异常本身，然后再继续正常的流程。本书还给出了许多实例来讨论这两种使用方法。

关于类型层次的章节深入讨论了替换原则。替换原则要求子类型规格支持基父类型规格。有3个属性必须得到支持：签名规则（Signature Rule），子类型对象必须拥有父类型的所有方法，并且

子类型方法的签名必须与相应的父类型方法的签名兼容；方法规则（Methods Rule），子类型方法的调用必须“表现得像”相应的父类型方法的调用；属性规则（Properties Rule），子类型必须保存父类型对象的所有属性。

这是一本结构性很好并且非常易读的书籍，特别是它具有思路开阔，层次丰富的特点。生动的数学抽象也使得乏味的技术内容变得明快有趣。喜欢阅读经典教材和专业书籍的年轻朋友们一定能通过阅读本书感受到喜悦，品尝到它的浓香和涩味。愿本书成为读者桌上一杯香浓的咖啡！

林德璋

上海交通大学软件学院

Preface

Constructing production-quality programs—programs that are used over an extended period of time—is well known to be extremely difficult. The goal of this book is to improve the effectiveness of programmers in carrying out this task. I hope the reader will become a better programmer as a result of reading the book. I believe the book succeeds at improving programming skills because my students tell me that it happens for them.

What makes a good programmer? It is a matter of efficiency over the entire production of a program. The key is to reduce wasted effort at each stage. Things that can help include thinking through your implementation before you start coding, coding in a way that eliminates errors before you test, doing rigorous testing so that errors are found early, and paying careful attention to modularity so that when errors are discovered, they can be corrected with minimal impact on the program as a whole. This book covers techniques in all these areas.

Modularity is the key to writing good programs. It is essential to break up a program into small modules, each of which interacts with the others through a narrow, well-defined interface. With modularity, an error in one part of a program can be corrected without having to consider all the rest of the code, and a part of the program can be understood without having to understand the entire thing. Without modularity, a program is a large collection of intricately interrelated parts. It is difficult to comprehend and to modify such a program, and also difficult to get it to work correctly.

The focus of this book therefore is on modular program construction: how to organize a program as a collection of well-chosen modules. The book relates modularity to abstraction. Each module corresponds to an abstraction, such as an index that keeps track of interesting words in a large collection of documents or a procedure that uses the index to find documents that match a particular query. Particular emphasis is placed on object-oriented programming—the use of data abstraction and objects in developing programs.

The book uses Java for its programming examples. Familiarity with Java is not assumed. It is worth noting, however, that the concepts in this book are language independent and can be used to write programs in any programming language.

How Can the Book Be Used?

Program Development in Java can be used in two ways. The first is as the text for a course that focuses on an object-oriented methodology for the design and implementation of complex systems. The second is use by computing professionals who want to improve their programming skills and their knowledge of modular, object-oriented design.

When used as a text, the book is intended for a second or third programming course; we have used the book for many years in the second programming course at MIT, which is taken by sophomores and juniors. At this stage, students already know how to write small programs. The course builds on this material in two ways: by getting them to think more carefully about small programs, and by teaching them how to construct large programs using smaller ones as components. This book could also be used later in the curriculum, for example, in a software engineering course.

A course based on the book is suitable for all computer science majors. Even though many students will never be designers of truly large programs, they may work at development organizations where they will be responsible for the design and implementation of subsystems that must fit into the overall structure. The material on modular design is central to this kind of a task. It is equally important for those who take on larger design tasks.

What Is This Book About?

Roughly two-thirds of the book is devoted to the issues that arise in building individual program modules. The remainder of the book is concerned with how to use these modules to construct large programs.

Program Modules

This part of the book focuses on abstraction mechanisms. It discusses procedures and exceptions, data abstraction, iteration abstraction, families of data abstractions, and polymorphic abstractions.

Three activities are emphasized in the discussion of abstractions. The first is deciding on exactly what the abstraction is: what behavior it is providing to its users. Inventing abstractions is a key part of design, and the book discusses how to choose among possible alternatives and what goes into inventing good abstractions.

The second activity is capturing the meaning of an abstraction by giving a specification for it. Without some description, an abstraction is too vague to be useful. The specification provides the needed description. This book defines a format for specifications, discusses the properties of a good specification, and provides many examples.

The third activity is implementing abstractions. The book discusses how to design an implementation and the trade-off between simplicity and performance. It emphasizes encapsulation and the need for an implementation to provide the behavior defined by the specification. It also presents techniques—in particular, the use of representation invariants and abstraction functions—that help readers of code to understand and reason about it. Both rep invariants and abstraction functions are implemented to the extent possible, which is useful for debugging and testing.

The material on type hierarchy focuses on its use as an abstraction technique—a way of grouping related data abstractions into families. An important issue here is whether it is appropriate to define one type to be a subtype of another. The book defines the substitution principle—a methodical way for deciding whether the subtype relation holds by examining the specifications of the subtype and the supertype.

This book also covers debugging and testing. It discusses how to come up with a sufficient number of test cases for thorough black box and glass box tests, and it emphasizes the importance of regression testing.

Programming in the Large

The latter part of *Program Development in Java* is concerned with how to design and implement large programs in a modular way. It builds on the material about abstractions and specifications covered in the earlier part of the book.

The material on programming in the large covers four main topics. The first concerns requirements analysis—how to develop an understanding of what is wanted of the program. The book discusses how to carry out requirements analysis and also describes a way of writing the resulting requirements specification, by making use of a data model that describes the abstract state of the program. Using the model leads to a more precise specification, and it also makes the requirements analysis more rigorous, resulting in a better understanding of the requirements.

The second programming in the large topic is program design, which is treated as an iterative process. The design process is organized around discovering useful abstractions, ones that can serve as desirable building blocks within the program as a whole. These abstractions are carefully specified during design so that when the program is implemented, the modules that implement the abstractions can be developed independently. The design is documented by a design notebook, which includes a module dependency diagram that describes the program structure.

The third topic is implementation and testing. The book discusses the need for design analysis prior to implementation and how design reviews can be carried out. It also discusses implementation and testing order. This section compares top-down and bottom-up organizations, discusses the use of drivers and stubs, and emphasizes the need to develop an ordering strategy prior to implementation that meets the needs of the development organization and its clients.

This book concludes with a chapter on design patterns. Some patterns are introduced in earlier chapters; for example, iteration abstraction is a major component of the methodology. The final chapter discusses patterns not covered earlier. It is intended as an introduction to this material. The interested reader can then go on to read more complete discussions contained in other books.

Barbara Liskov

Acknowledgments

John Guttag was a coauthor of an earlier version of this book. Many chapters still bear his stamp. In addition, he has made numerous helpful suggestions about the current material.

Thousands of students have used various drafts of the book, and many of them have contributed useful comments. Scores of graduate students have been teaching assistants in courses based on the material in this book. Many students have contributed to examples and exercises that have found their way into this text. I sincerely thank all of them for their contributions.

My colleagues both at MIT and elsewhere have also contributed in important ways. Special thanks are due to Jeannette Wing and Daniel Jackson. Jeannette Wing (CMU) helped to develop the material on the substitution principle. Daniel Jackson (MIT) collaborated on teaching recent versions of the course and contributed to the material in many ways; the most important of these is the data model used to write requirements specifications, which is based on his research.

In addition, the publisher obtained a number of helpful reviews, and I want to acknowledge the efforts of James M. Coggins (University of North Carolina), David H. Hutchens (Millersville University), Gail Kaiser (Columbia University), Gail Murphy (University of British Columbia), James Purtilo (University of Maryland), and David Riley (University of Wisconsin at LaCrosse). I found their comments very useful, and I tried to work their suggestions into the final manuscript.

Finally, MIT's Department of Electrical Engineering and Computer Science and its Laboratory for Computer Science have supported this project in important ways. By reducing my teaching load, the department has given me time to write. The laboratory has provided an environment that enabled research leading to many of the ideas presented in this book.

Contents