经　典　原　版　书　库

# 面向对象软件开发原理

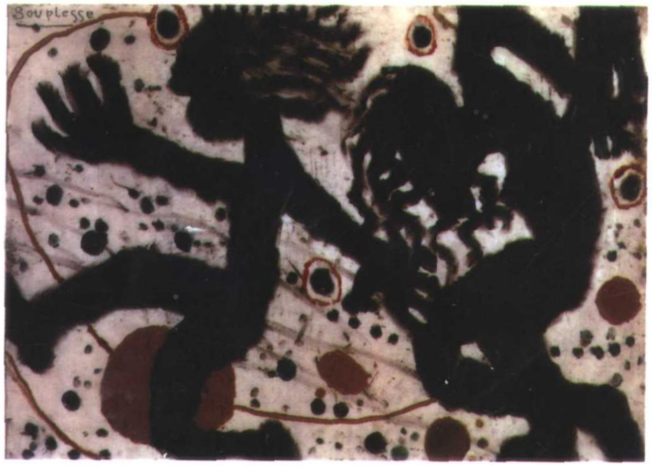
（英文版·第2版）

Principles of

# Object-Oriented Software Development

2nd Edition

Anton Eliëns



附赠 CD-ROM

Addison-Wesley

（荷）Anton Eliëns　著

机械工业出版社
China Machine Press

经 典 原 版 书 库

# 面向对象软件开发原理

（英文版·第2版）

# Principles of Object-Oriented Software Development

(Second Edition)

（荷）Anton Eliëns 著

机械工业出版社
China Machine Press

**本书版权登记号：图字：01-2003-1014**

**图书在版编目（CIP）数据**

面向对象软件开发原理（英文版·第2版）/（荷）艾伦（Eliëns, A.）著. -北京：机械工业出版社，2003. 4

（经典原版书库）

书名原文：Principles of Object-Oriented Software Development, Second Edition

ISBN 7-111-11907-X

Ⅰ. 面…　Ⅱ. 艾…　Ⅲ. 面向对象语言－软件开发－英文　Ⅳ. TP311. 52

中国版本图书馆CIP数据核字（2003）第023801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：杨海玲

北京瑞德印刷有限公司印刷·新华书店北京发行所发行

2003年4月第1版第1次印刷

787mm × 1092mm 1/16 · 33.5印张

印数：0 001 - 3 000册

定价：59.00 元（附光盘）

# 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年开始，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：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。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

[illegible]

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。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。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、有的历经三十年而不衰、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）68995264
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）

# Foreword

What an unusual book! I have certainly seen many books on object-oriented software development and even some that have similar coverage, but Anton Eliëns's book is in a different category entirely. As so many books in our field, this one has also had its roots in the development of lecture notes. However, Eliëns took a surprising deviation from the established path of developing notes into books. Instead of inflating the notes, reorganizing the material, and creating the traditional textbook, Eliëns decided to keep the essence of his notes alive.

By condensing the key points into "slides" and keeping these slides as visual anchors all over his text, the reader's experience is truly different. There is a fast track where we just follow the slides: this is what students to with handouts on first encounter. Then there are the deeper modes of reading where we focus in and follow Eliëns's full-text explanations; explanations that are thorough enough where that is important and shallow enough where overwhelming detail wouldn't pay back. Reading through this book and working with it to enhance our understanding is a pleasure.

For a breakdown of the book's structure, I refer to the preface and the foreword to the first edition. However, it is worth noting that Eliëns has improved his text substantially over the first edition. A theme close to my heart has been woven into the text: components. Software architecture, a closely related theme of quickly growing importance, has also found coverage.

The style and didactic quality of the presentation are matched by a wide-ranging selection of topics. Living in times of rapid change and extremely broad diversification of our discipline, we have to value the few books that span significant ranges in an integrative fashion. After all, it is the reader's key challenge to use books like Eliëns's to reconstruct and integrate the vast sea of knowledge fragments out there; and it is with the help of books like Eliëns's that the reader has a chance of achieving this formidable goal. Instructors and lecturers will equally appreciate this book as readily usable for teaching and lecturing tasks.

In the end, for the software developer to be, as well as for the established software developer or the computer scientist with an eye on software development, there is a lot to know from a spectrum of subdisciplines, before we can feel even half-confident about what we are actually doing when developing software. To get there, we need to understand everything from modeling and design techniques, over architecture and components, to implementation detail expressed in specific programming languages. This book is a good starting point for doing so from an object-oriented perspective. (Keep your eyes open for other perspectives and approaches, though!)

*Clemens Szyperski*
*November 1999*

# Foreword to the first edition

This book is an important contribution to object-oriented literature, bridging the gap between the language and software engineering communities. It covers language design issues relating to inheritance, types, polymorphism, and active objects as well as software design paradigms such as the object modeling technique (OMT), the model-view-controller paradigm (MVC) and responsibility-driven design. Its four-part subdivision of the subject matter into design, languages and systems, foundations, and application frameworks nicely balances practice and theory, covering both practical design techniques and foundational models. Its use of C++ as the primary application language, with Smalltalk and Eiffel as additional languages, allows the book to be used in courses with programming assignments in mainstream object-oriented languages.

The overall sense of balance and perspective is matched by an engaging style and a modern treatment of an exceptionally broad range of topics in the body of the book. The conceptually challenging questions at the end of each chapter (with answers in an appendix) are sometimes humorous. For example, the question 'Why do you need friends?', which invites the reader to examine the value of this C++ language construct, is nicely answered by pointing out tradeoffs between efficiency and safety, ending with the admonition 'treat friends with care'.

Object-oriented programming started as a language framework for single-user systems, but is maturing into a technology for heterogeneous, distributed network systems that focus on interoperability and glue for the composition of heterogeneous modules. The notion of structure in object-oriented programming is analogous to, but more complex than, the structure of structured programming. This book reflects the maturation process from single-user to distributed systems technology and provides a bridge from object-oriented concepts of single-user programming to distributed software design concepts.

Basic object-oriented concepts are introduced from the viewpoint of design, thereby motivating language concepts by their role in the software life cycle. The first four chapters provide a gentle introduction to fundamental concepts that yields unexpected insights for the seasoned reader. Chapter 1 examines paradigms of programming and provides a distinctive object-oriented view of the software life cycle, while chapter 2 presents C++, examines its benefits and pitfalls, and compares it to Smalltalk and Eiffel. Chapter 3 on object-oriented design includes an insightful discussion of models, contracts, and specifications that provides a comparative overview and synthesis of alternative approaches to the conceptual foundations of design. Chapter 4 rounds out the section on design with a discussion of testing and metrics for software validation that provides a practical counterpoint to the conceptual focus of earlier chapters.

The topics in the first four chapters are well chosen to provide a foundation for later topics. The chapter on language design principles includes an up-to-date review of models of inheritance and delegation, the chapter on concurrency examines inheritance anomalies, concurrent object models, and principles of distributed programming, while the chapter on composition and collaboration explains callbacks, window management, and event-driven computation. The three chapters on foundations examine, in a substantive but relaxed way, algebraic models for abstract data types, calculi for type polymorphism, and behavioral refinement through subtyping. The two final chapters provide an account of interoperability, standards, library design, requirements engineering, hypermedia links, and heterogeneous systems.

This book covers an unusually broad range of topics in an eminently readable fashion and is unique in its balance between theory and practice and its multifaceted approach. Anton Eliëns demonstrates an up-to-date mastery of the literature and the rare ability to compare, evaluate, and synthesize the work of different software research and development communities. He is to be commended on his skill and versatility in weaving a sequential expository thread through a heterogeneous, distributed domain of subject matter.

*Peter Wegner*
*October 1994*

# Preface

This is a book about object-oriented software development. It reflects the contents of an upper-level undergraduate course on Object-Oriented Programming, given at the Vrije Universiteit Amsterdam.

This was the beginning of the preface of the first edition. It still holds true. However, OO is a rapidly evolving field. As a consequence my book, published in 1994, may have been considered to be outdated from the start. As an example, right after its publication, patterns came into the focus of public interest. As another example, think of the Java wave that has come over us. Clearly, a revised edition was needed in which those subjects, and other subjects, are covered, or, as in the case of CORBA, are covered in more detail.

Another reason is that the field of OO itself has matured considerably. The acceptance of UML as a modeling standard is one example. The increased utilization of CORBA for business-critical applications is another sign that (distributed) object technology is being considered as sufficiently robust.

The availability of new topics in itself is not enough to justify a second edition, since new books have been published in which these topics are covered. You only have to think of the enormous number of books on Java ... A revised second edition of the book is justified however, in my opinion, since the book distinguishes itself from the competition by its approach. Set up as a series of lectures, organized around so-called slides, the book covers a large number of topics, some in depth, some more casually. From an educational point of view, the advantage of this approach is the direct availability of educational material, including the slides to be presented in classroom. For the average reader, moreover, the slides provide an overview which facilitates comprehension and recall.

Finally, another more personal reason for bringing out a revised edition is that both in research and teaching my experience with OO has become more extensive, and I may even dare say that my own thoughts about OO have matured to some extent. In particular, in my group we have developed a multi-paradigm OO framework, which was already introduced in chapter 12 of the first edition, that has been applied in, for example, business process reengineering and collective improvisation on the Web. Although I do not plan to treat any of this material extensively, it does provide a basis for the examples and, moreover, the material (including articles, software and examples) will be available on the accompanying CDROM.

## Features of this book

- The book provides an introduction to object-oriented programming, covering design, languages, and foundational issues. It pays attention to issues such as reuse, component technology, design patterns and in particular the application of object technology in Web applications.

- It contains guidelines for developing object-oriented applications. Apart from practical examples it provides an overview of development methods as well as an introduction to UML, the standard for object-oriented modeling. In particular *design patterns* will act as a recurrent theme, or rather as a perspective from which examples and solutions will be discussed.

- Distributed object technology will be a major theme. The book provides an introduction to CORBA that allows the student to gain hands-on experience with developing CORBA applications. It also provides a discussion of competing technologies, and in particular it will elucidate the distinction between component technology and distributed objects. Examples in Java and C++ will be included.

- Another major theme of the book is to establish precisely the relation between the guidelines and prescriptions emerging from software engineering practice on the one hand, and the constraints and insights originating from theoretical research. In the book attention will be paid to foundational issues as well as the pragmatical solutions the designers of object-oriented languages have chosen to realize their ideas.

- Many of the notions introduced and problems discussed are clarified by short programs, mostly in Java, some in C++. The examples cover GUI development, business process reengineering and Web applications. No extensive knowledge of the programming languages used is required since a brief tutorial on a number of object-oriented programming languages, including C++, Smalltalk, Eiffel and Java, is given in the appendix.

- The material is organized around *slides*. The slides occur in the text in reduced format, but are also available in Powerpoint and Netscape Presentation format. Each slide captures some important notion or concept which is explained and commented upon in the accompanying text. An online Instructor's Guide is available that provides hints for presenting the slides and answers to the questions posed at the end of each chapter.

- The entire book, including the software from the examples and the *Instructor's Guide* is available electronically, on the accompanying CDROM as well as on the Internet. The electronic version contains links to other material on the Internet. The electronic version may be accessed also in *slide mode* that allows for presenting the material in a classroom equipped with a beamer.

**Intended readers** The book will primarily address an academic audience, or IT professionals with an academic interest. Nevertheless, since I am getting more and more involved in joint research with business partners and the development of extra-academic curricula, examples are included that are of more relevance to IT in business. In particular, it contains a section on the deployment of (object-oriented) simulation for business process redesign, and a section on the 3D visualisation of business data using object technology.

This book may be used as the primary text for a course on OO or independently as study or reference material. It may be used by the following categories of readers:

- *students* – as a textbook or as supplementary reading for research or project papers.
- *software engineers* – as (another) text on object-oriented software development.
- *professional teachers* – as ready-made material for a course on object-oriented software development.

Naturally, this is not meant to exclude other readers. For instance, researchers may find the book useful for its treatment of foundational issues. Programmers may benefit from the hints and example programs in Java and C++. Another reason for using this book may be its compact representation of already familiar material and the references to other (often research) literature.

The book is meant to be self-contained. As prior knowledge, however, a general background in computer science (that is, computer languages and data structures as a minimum) is required. To fully understand the sections that deal with foundational issues or formal aspects, the reader must also have some knowledge of elementary mathematical logic.

**Organization** The book is divided into four parts. Each part presents the issues involved in object-oriented programming from a different perspective, which may be characterized respectively as *software engineering and design*, *languages and system development*, *abstract data types and polymorphism*, and *applications and frameworks*.

**Part I: Designing Object-Oriented Systems**

1. *Introduction*: This chapter gives an introduction to the area of object-oriented software development. It gives a global view on the object-oriented life cycle and discusses object orientation as a paradigm of programming. It discusses a number of trends and technologies that have come into the focus of public attention and indicates their relevance to 'object-orientation'.

2. *Idioms and patterns**: This chapter introduces *idioms* and *design patterns* as means to capture recurrent structures and solutions in object-oriented programming. It distinguishes between idioms as solutions tied to a particular language and patterns which are the product of rational design. This chapter contains numerous examples, in Java.

3. *Software engineering perspectives*: This chapter discusses the process of software development and the various modeling perspectives involved in analysis and design. It explains the issues involved in arriving at a proper object model and introduces the notion of *contract* as an instrument to capture the relationships between object classes. In addition, it proposes a method for validation and testing based on *contracts*.

4. *Application development**: In this chapter we develop a complete application and discuss the issues involved in its design and realization. It presents guidelines for (individual) class design, and gives an example of how to derive an implementation from a formal specification.

**Part II: Object-Oriented Languages and Systems**

5. *Object-oriented languages*: This chapter provides a comparison between object-oriented languages, including Smalltalk, Eiffel, C++ and Java. It further discusses a number of alternative languages, included Self and Javascript, each with their own object model, and treats issues such as dynamic inheritance by delegation. synchronous active objects, and meta-level architectures for class-based languages.

6. *Component technology**: This chapter discusses the relation between component technology and distributed object technology, and will give a brief overview of the solutions that are available on the market, including Microsoft COM/ActiveX, JavaBeans, Java RMI and CORBA. It also presents a simple workgroup application and an example of integrating CORBA with an existing software library.

7. *Software architecture**: In this chapter we explore how software architecture affects design and implementation. It treats design patterns for distributed object systems, and looks at the technical issues involved in developing multi-lingual systems. As an example we show how to employ the native interface to embed an existing framework in Java.

**Part III: Foundations of Object-Oriented Modeling**

8. *Abstract data types*: This chapter considers the notion of abstract data types from the perspective of *types as constraints*. It presents an algebraic approach in which objects may be characterized as algebras. Further, it explains the difference between the classical approach of realizing abstract data types in procedural languages and the realization of abstract data types in object-oriented languages. The implications of a more pragmatic conception of types is also discussed.

**9.** *Polymorphism*: This chapter discusses inheritance from a declarative perspective, and gives a precise characterization of the subtype relation. It further discusses the various flavors of polymorphism and presents a type theoretical treatment of genericity and overloading. Also, type calculi that capture data hiding and self-reference are given. These insights are related to the realization of polymorphism in Eiffel, C++ and Java.

**10.** *Behavioral refinement*: This chapter extends the notion of types as constraints to include behavioral properties. It presents an assertion logic for the verification of programs and discusses the operational model underlying the verification of object behavior based on traces. It further gives precise guidelines to determine whether classes that are syntactical subtypes satisfy the behavioral refinement relation. Finally, an overview is given of formal approaches to characterize the behavior of collections of objects.

**Part IV: Object-Oriented Application Frameworks**

**11.** *Business process redesign**: In this chapter we look at the opportunities IT offers in (re)designing business processes. In particular, we look at (object-oriented) simulation as a means to capture the logistical aspects involved in business process modeling, and in addition we look at how simulation models can be made available as to allow decision making, by deploying visualisation and dissemination over the Web.

**12.** *Web applications**: In this chapter we look at how object technology may be applied to the Web. We will look both at client-side extensions and server-side solutions. In particular, we look at systems that employ CORBA in addition to other Web technologies. We also briefly look at another new trend in computing, intelligent, mobile agents, and we argue that agents are a direct derivation from object technology.

**Appendices** The appendices contain brief tutorials on Smalltalk, Eiffel, C++, Java and the distributed logic programming language DLP. They also contain an overview of UML, an overview of CORBA IDL, a tutorial on programming CORBA applications with Orbacus, and suggestions for small and medium-term projects.

**Tracks** For those developing a course on object-oriented programming, the book offers a choice between various tracks, for which the ingredients are sketched below. Also, an indication is given of the sections that contain more advanced material.

| | regular | extended | advanced |
|---|---|---|---|
| programming | 2, 4, 5, 12 | 6, 11 | 7, 8 |
| software engineering | 1, 3, 4, 11 | 8.1-2, 10.1 | 9.1-3, 10.2 |
| theoretical | 1, 3, 8 | 5, 9.1-4 | 9.5-6, 10 |

The *programming track*, consisting of chapters 2, 4, 5 and 12, may be augmented with material from the appendices and chapters 6 and 11. The *software engineering track*, consisting of chapters 1, 3, 4 and 11, may be augmented with material from the theoretical track as indicated. The *theoretical track*, consisting of chapters 8, 9 and 10, may need to be augmented with more general information concerning OOP provided in the other tracks.

**Differences with respect to the first edition** For clarity I have marked the chapters that have been substantially changed with an asterisks.

Adding new topics is one thing, eliminating parts of the book, naturally, is quite another thing. Yet I have chosen to remove the chapters on C++ (previously chapter 2), software engineering issues (chapter 4), concurrency in C++ (chapter 6), composition mechanisms (chapter 7), software libraries (chapter 11) and hypermedia (chapter 12). Some of this material, for example parts of the hypermedia chapter (12), composition mechanisms (7), and software engineering issues (4), will reappear elsewhere. Nevertheless, since some of it is obsolete, and other material does not function well in classroom, it is better to remove it, and allow its space to be taken by other topics.

**Background and motivations** My own interest in object-oriented languages and software development stems from my research on the language DLP, a language integrating logic programming with object-oriented features and parallelism (Eliëns, 1992). When looking for material for a course on object-oriented programming, I could not find a book that paid sufficient attention to foundational and formal aspects. Most of the books were written from a perspective on OOP that did not quite suit my purposes. What I was looking for could to some extent only be found in research papers. As a consequence, I organized my OOP course around a small number of papers, selecting the papers that, to my mind, can be considered as *landmark papers*, papers that have become known as originally presenting some significant notion or insight. The apparent disadvantage of basing a course on OOP on papers is the obvious lack of a unified view, and of a consistent use of terminology. The advantage of such an approach, however, is that students are encouraged to assess the contribution of each paper and to form their own view by comparing critically the different viewpoints expressed in the papers. Personally, I favor the use of original papers, since these somehow show more clearly how the ideas put forward originated. Later, more polished, renderings of these same ideas often lack this quality of 'discovery'.

The idea of organizing a book around slides came quite naturally, as the result of structuring the growing collection of slides, and the wish to maintain the compact representation offered by the slides.

The choice of material reflects my personal preference for foundational issues, in other words, papers that are focused on concepts rather than (mal)practice. The choice of material has also been colored by my interest in (distributed) hypermedia systems, the Web and, to some extent, by my previous work on distributed logic programming. Although the book is certainly not focused on language constructs, modeling issues as well as foundational issues are generally related to existing or conceivable language constructs, and (whenever possible) illustrated by working examples developed for that purpose.

The choice for Java as the main vehicle for presenting the program fragments and examples is motivated simply by the popularity of Java. The presentation of some of the other examples in C++ reflects my belief that C++ must still be considered as a valid programming language for object-oriented software development. However, I also believe that in the (near) future multi-paradigm approaches (extending Java and C++) will play a significant role.

The approach taken in this book may be characterized as *abstract*, in the sense that attention is paid primarily to concepts rather than particular details of a solution or implementation language. By chance, in response to a discussion in my class, I looked up the meaning of *abstract* in a dictionary, where to my surprise I learned that one of its meanings is *to steal, to take away dishonestly*. Jokingly, I remarked that this meaning sheds a different light on the notion of *abstract data types*, but at a deeper level I recognized the extent to which the ideas presented in this book have profited from the ideas originally developed by others. My rendering of these ideas in a more abstract form is, however, not meant to appropriate them in a dishonest way, but rather to give these ideas the credit they deserve by fitting them in a context, a framework encompassing both theoretical and pragmatical aspects of object-oriented computing. As one of the meanings of the adjective *abstract*, the dictionary also lists the word *abstruse* (not easy to understand). There is no need to say that, within the limits of my capabilities, I have tried to avoid becoming abstruse.

Finally, in presenting the material, I have tried to retain a sufficient degree of objectivity. Nevertheless, whenever personal judgments have slipped in, they are meant rather to provoke a discussion than provide a final answer.

**Information** The electronic version can be found at

`http:`*//www.cs.vu.nl/~eliens/online/oo*

For any questions or comments you may contact the author at `eliens@cs.vu.nl` by electronic mail, or at Dr A. Eliëns, Vrije Universiteit, Faculty of Sciences, Division of Mathematics and Computer Science, De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands.

**Contents of the CDROM** The CDROM contains a complete online version of the book, including additional lectures, software and links to resources on the Internet. This online version may be used for presentation in the classroom, using the Netscape Presentation Format, which is supported by Netscape Navigator 4.x or better and by Internet Explorer 4.x or better. For each chapter, the CDROM