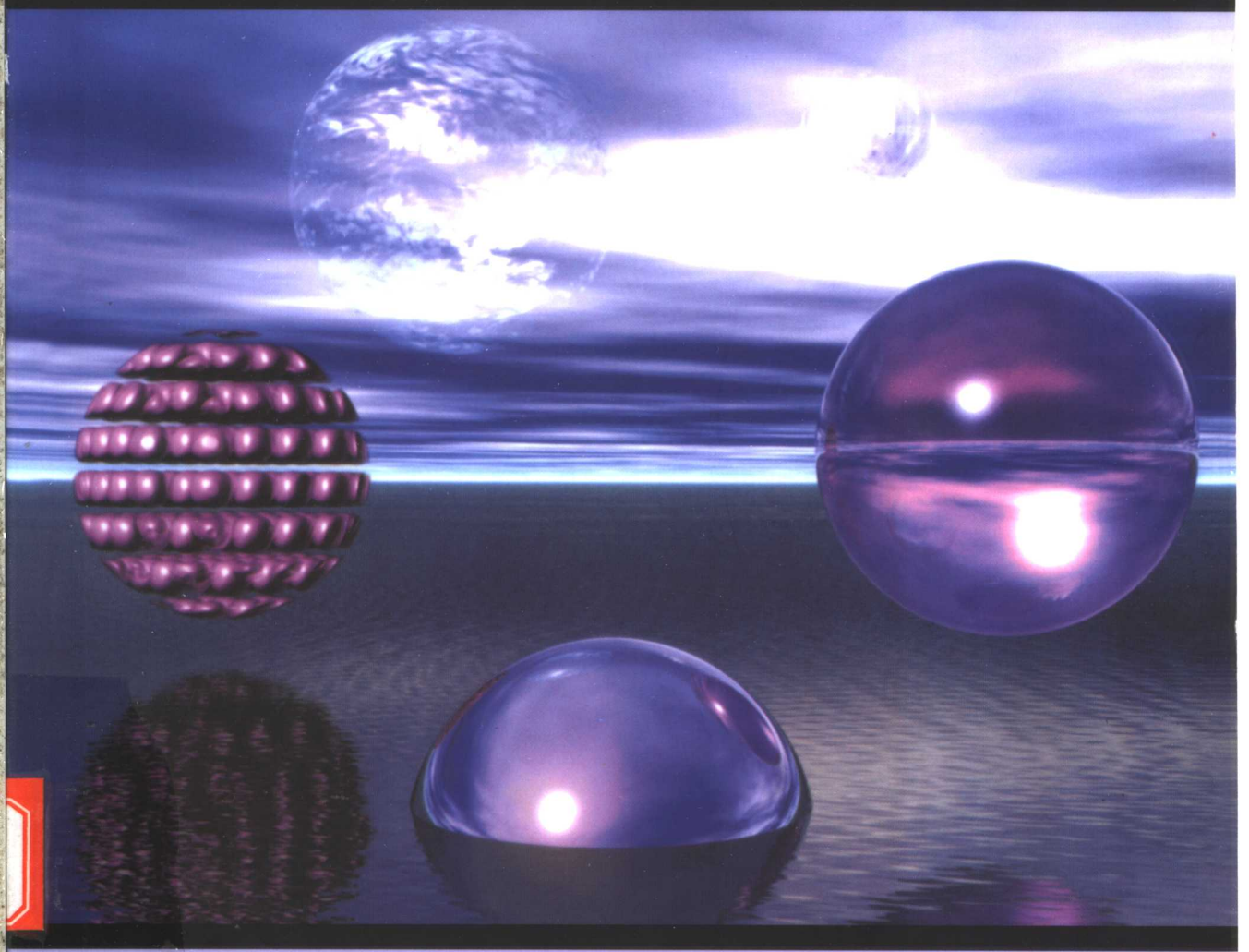
PEARSON
Prentice Hall

UML与Java面向对象开发

Practical Object-Oriented Development with UML and Java

（影印版）



[美] Richard C. Lee，William M. Tepfenhart 著

Pearson Education

清华大学出版社

UML 与 Java 面向对象开发

（影印版）

Practical Object-Oriented Development with UML and Java™

Richard C. Lee
William M. Tepfenhart

清华大学出版社
北京

图书在版编目(CIP)数据

UML 与 Java 面向对象开发=Practical Object-Oriented Development with UML and Java/（美）李，（美）泰芬哈特著. —影印本. —北京：清华大学出版社，2004.4
ISBN 7-302-08206-5

I. U… II. ①李… ②泰… III. ①面向对象语言，UML—程序设计—英文 ②Java 语言—程序设计—英文 IV. TP312

中国版本图书馆 CIP 数据核字（2004）第 015572 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　　编**：100084
社总机：010-6277 0175　　**客户服务**：010-6277 6969
责任编辑：郭福生　常晓波
封面设计：立日新
印 刷 者：清华大学印刷厂
装 订 者：三河市金元装订厂
发 行 者：新华书店总店北京发行所
开　　本：180×230　**印 张**：31.25
版　　次：2004 年 4 月第 1 版　2004 年 4 月第 1 次印刷
书　　号：ISBN 7-302-08206-5/TP・5923
印　　数：1~3000
定　　价：59.00 元

前　言

本书是为从事大型系统开发的繁忙的专业软件分析师和开发人员编写的。如果你没有时间参加课程培训，而又希望跟上统一建模语言(UML)和 Java 的面向对象技术的发展，那么本书正好是一本非常合适的自学教程。它将帮助你了解面向对象分析、面向对象设计和面向对象编程之间的差别。本书的目标是：

- 为你介绍如何利用 Java 建立面向对象的应用程序，并根据业务需求进行正确的决策。
- 澄清与面向对象技术有关的基本概念。
- 为学生和专业人员提供有足够深度的学习资料，帮助他们跟上这个领域的技术发展。
- 揭开面向对象技术的神秘面纱，关注它作为一种软件工程工具所具有的实用性。
- 提供利用面向对象技术分析、设计和编写程序的实用方法。
- 介绍如何利用 Java 实施面向对象技术。
- 找到现有文献中的应用实践与理论的平衡点。

要深入了解面向对象的重要概念和问题，不一定需要了解计算机科学和高等数学。即使在有关编程的章节中，也不需要 Java 背景；而是用图的形式表示 Java 代码的编写过程。

面向对象技术

假设我们都是大型系统的软件开发人员。我们得到了许多用不同语言编写的代码，这些语言代表了多种软件技术。这里出现的软件革新在过去的 30 年中我们并未经历过。因此，从不平凡的角度，可以说，我们相信面向对象的技术是我们使用过的最重要的软件技术。

为什么这样说呢？因为面向对象改变了我们构建软件的方法，也改变了应用程序通过多个厂商提供的计算机在世界范围的网络间沟通交流的方式。此外，对象模型正在改变着我们设计业务过程的方式和我们认识一个企业的方式。

多数企业都在重新设计自己的系统，以迎接目前由 Internet 带来的业务挑战。面向对象技术在这个过程中扮演了重要角色，它提供了可体现业务过程、工序、策略的模型以及推动设计的规则。利用将模型转换为运行系统的工具，可以推动再设计过程的实施。

随着市场和商业环境的改变，这些系统也需要重新生成，通过更新模型和使用这些工具可以反映这些变化。较之于在过去几十年中使用的其他方法，可靠的软件工程实践为我们提供了更深入和更迅速的方法。

这是一个常识，即面向对象的技术对软件起到了深远的影响，也就是说，面向对象技术的机制正在成为软件结构和计算机硬件设计的芯片设计的主体。这些看法来自于：

- 熟悉高级的面向对象模型，为软件设计人员提供了真实世界的可编程组件，从而降低了软件开发成本。
- 面向对象技术共享和重用代码的能力缩短了应用程序的开发周期。
- 通过编程抽象机制使修订工作局部化，并将修订所带来的影响减至最小，从而加速开发过程，使软件更可靠、更健壮。
- 其复杂度管理能力可以解决更困难的应用程序问题。

面向对象的概念的集合就是对现实世界建模的一个工具集。这个面向对象的工具集为开发人员提供了复杂度管理的最佳途径。有些面向对象的概念可帮助开发人员提高软件的灵活性和可维护性。

为什么使用统一建模语言(UML)？

作为面向对象技术的使用者，我们知道，如果使用得当，那么应用所有的方法都会得到相同或相似的模型。但是不同的建模语言符号，可能会妨碍进展。统一建模语言(UML)已经成为了一种工业标准，它集成了各种不同的建模符号，形成了一套单一的建模语言符号系统。这就为我们选择 UML 提供了足够充分的理由。

UML 是对分析和设计模型建立文档的一种语言。它向我们提供了用于获得对解决实际的业务问题有价值的多数概念或机制所必需的所有绘画图标。另外，它还提供了对模型建档十分重要的所有必要的图表。最后，它是一种可发展的语言，允许我们扩充并非由 Rational Software Corporation 的 Grady Booch、James Rumbaugh 和 Ivor Jacobson 组成的著名小组定义的机制符号。

UML 并不是本书的主题。本书只是将其作为分析和设计模型建档的方法进行了介绍，本书的主题是开发设计模型的方法。所有对 UML 的介绍和讨论都是根据在其中获得的信息和如何获得信息的方式进行的。

为什么介绍 Java？

准确地说，Java 是一种排他性的面向对象的编程语言，较之于多范型(multi-paradigm)编程语言 C++，其排他性限制了它的使用。但是 Java 有一个优点使它超越了一般性的限制。实际上，Java 是运行在 Java 虚拟机(Java VM)上的。这就使得 Java 程序能够运行

在任何实施了 Java 虚拟机的机器上。从而使开发人员不必再为几种硬件和操作系统的不同组合设计和开发具有相同功能的软件。

乍看之下，这对于使用虚拟机的推动作用可能并不明显。一方面，厂商们现在关注于工具和产品的开发，他们会将投资放在一个方向，而不是五个或六个方向。这就是说，他们强调更强功能性的实现（我们称之为更大的商业价值）。这对重用也产生了影响。开发人员可以在运行任何操作系统的任何硬件上开发程序库，且不必针对平台差异进行修改就可重用代码。编程错误不会出现在某个指定平台的某个版本的代码中，也不会出现在其他版本中。重用相同代码的基础越广泛，就意味着用更少的时间和更低的成本实现的组件的可靠性越强。分析工作可着重于商业价值；设计工作应关注于更大的灵活性和可维护性；实现和测试工作现在可侧重质量、可靠性和性能方面。这种变化的最终结果就是花更少的钱得到更好的代码。

如果分析过 Sun 提供的大量的 Java 库(框架)，就可体会到这些优点。现在有一套很好的通用实用程序库、高性能的图形用户界面库(如 SWING)以及供开发人员使用的特定用途的商业类库。较之于这些库的 C++版本，它们更加全面，提供了更多的功能，并且可以更容易地引入到最终的产品中。因为这些库在所有的项目中都可用，并且在手册中进行了详细的记录，所以这些库得到广泛应用。因此，其中的应用技巧非常易于掌握。较之于 C++版本，它具有完全不同的应用程序编程接口(API)。在 C++领域中，开发人员可能是某个平台的某种产品的专家，但他不了解其他平台的其他任何产品。实际上，没有几个程序员能够同时编写基于 PC、Mac 和 Unix 平台的代码。

在现代网络浏览器中直接引入 Java 虚拟机，使得从 Internet 上下载 Java 程序，并且在浏览器中运行成为可能。这有助于为浏览器环境提供更强的功能性，并且产生了一类新的应用程序。可以说，如果没有 Java，我们就不可能在通过 Internet 为用户提供的功能性方面获得现在的成绩。Java 在现代 WWW 上应用的广泛程度，在出现一种新的能够提供更强抽象能力和同样广泛的平台支持的技术（现在还不可知）之前是不可能削弱的。

面向对象技术的应用

我们不是面向对象技术的纯粹主义者，也不是理论家。我们只是希望通过好的方法达到两个非常重要的商业目标的开发人员，这两个目标是：更低的开发成本和更短的开发周期。我们相信可靠性、可维护性和灵活性这几个技术目标对于满足上述商业目标至关重要。

我们使用面向对象技术的方法是希望通过管理软件开发工作的复杂度，以使软件具有良好的可靠性、可维护性和灵活性。复杂度管理是达到这些目标，特别是我们的商业目标的关键。我们发现，要在复杂的问题域中进行复杂度管理，开发人员需要了解对象、

类、关系和规则是如何适应对象范型的。在建立最复杂的问题域时，我们发现了对象、类和对象中的许多关系。此外，我们需要获得这些域中的规则(策略)。因此，我们必须利用非常丰富的静态建模技术，以捕捉数据(对象)关系。

许多面向对象的技术专家认为关系是“不好的”，因为它违反了封装原则。从我们的角度看，它能够帮助我们管理问题域的复杂度，还有助于达到我们的商业目标。我们喜欢使用它，并希望这个领域能够得到更多的机制和语言支持。在关于声明语义的第 9 章，我们介绍了那些应作为模型的组成部分、而非某个特殊子系统的延伸的规则和策略。

利用机制帮助我们建立复杂问题域模型，与选择 UML 作为建模语言，选择 Java 作为编程语言是一致的。UML 和 Java 允许我们定义有助于构建更可管理的软件所需的机制。

我们讨论了捕捉模型过程特征的行为(动态和静态)和多态性。利用有限状态机或其他状态模型有助于在定时、同步和中断寻址时进行过程复杂度管理。我们还介绍了用于错误恢复管理的例程（这是一个很重要的题目，因为错误恢复是程序逻辑组成部分的一半）。这些方面在大多数面向对象的书籍中通常是被忽略的。

我们相信，成功构建大型的面向对象系统的关键是，需要开发人员和程序员了解比在多数面向对象的书籍中所介绍的更多的内容。构建大型系统需要使用由某些面向对象的技术专家提出，但并未被所有人接受的机制。专业的开发人员在成为多产的团队成员之前，至少需要了解如何处理问题域的这些特征。本书不会让你成为专家，你仍然需要系统开发方面的专家或顾问。根据 80/20 规则，本书的 80%能够使你成为多产的开发人员，并了解专家解决的 20%的问题。

本书并没有介绍面向对象技术的最新趋势和潮流，包括对象设计模式、标准模板库以及分布式对象计算。尽管这些内容十分有趣，但并不符合本书的宗旨：为开发人员提供一种实用框架，这种框架能够让不熟悉面向对象编程的开发人员尽快熟悉这种技术。

最后，我们并不同意多数专家所认为的，面向对象的技术是一种成熟的技术。我们认为它正在趋于成熟。面向对象技术具有巨大的潜力，能够帮助我们进行早期技术(基于过程、功能和规则的技术)所不具备的复杂度管理工作。我们看到在面向对象技术和 Java 中有许多不同的抽象机制合并(集成)到了一个真正强大的技术中。这种合并仍未完成，但在 Java 中远远比在其他面向对象的技术中所获得的成果要多。

本书内容的编排

我们带领读者以理性方式应用面向对象的技术和方法。这并不是绝对的规则。我们的目标是让你了解在 Java 中开发软件和编程时所使用的好的面向对象的概念和好的设计原则。

我们设计将本书编写为一本自学指南，应当按顺序阅读。我们采用了 Richard 多年以来在面向对象的概念和基本技巧的教学中所使用的方法；但是我们并不提倡将之作为构建面向对象的系统的方法。每章都讨论了应用面向对象技术的一个重要步骤。大多数章节以逐步介绍的指南或秘诀结束。我们希望读者只以这些步骤作为指导；应当相信常识而不是盲目地遵照后面介绍的步骤。

第 1 章，介绍对复杂度控制机制的抽象，并建立面向对象的概念——面向对象是抽象机制的现代版本。

第 2 章，介绍面向对象的基本原则。

第 3 章，通过用例方法开发一个规范模型，作为开发过程的开始。

第 4 章，通过识别对象/类/接口，启动分析模型的开发过程。

第 5 章，介绍如何通过识别属性(数据)和与对象相关的服务来区别“真”对象和“假”对象。

第 6 章，演示捕捉对象行为的过程。

第 7 章，介绍如何识别和描述动态行为。

第 8 章，介绍几种不同的关系(通用/特定、链接和对象集等)，它们可用于组织系统中所有的对象。

第 9 章，介绍如何将声明性事实(declarative fact)引入到关于对象知识的面向对象模型中和用于实施的基于规则的机制中。

第 10 章，回顾分析模型，重新构建分析模型以考虑 helper 类。

第 11 章，介绍几种开发设计模型的元素。[1]

第 12 章，介绍 Java 语言编程。

第 13 章，介绍如何用 Java 实现类和接口。

第 14 章，介绍静态行为的实现过程。

第 15 章，介绍动态行为的实现过程。

第 16 章，介绍通用/特定(generalization/specialization)的实现过程。

第 17 章，介绍其他关系的实现过程。

附录 A，概要介绍统一模型语言。

附录 B，简要介绍 Java。

附录 C，比较 Java 和 C++的异同。

[1] 从系统的角度看，设计是非常复杂的，并且是一个独立于面向对象技术的课题。但是，第 14～22 章从如何实现分析概念的角度介绍了几种设计元素。在所有关于实现的章节中都介绍了一些习惯用语和设计模式。

本书的使用方法

本书主要针对经验丰富的软件开发人员和大学的高年级学生。以有相当能力的程序员所需要的工科课程中的教学内容为基础。本书内容可在为期两周的课程中介绍。第一周介绍前 11 章，第二周介绍余下的部分。这种课程的讲授一般要结合一个项目，即让学生开发完成一个计算机游戏项目，而不是完成一些家庭作业。第一周的课程结束时，学生们应完成游戏的设计。第二周的课程结束时，学生应全部实现了他们的设计。

我们选择基于项目的方法进行教学有几个原因。首先，我们发现家庭作业要么在有效地传达所学概念的重要意义方面价值不大，要么过于复杂，以至于不能在合理的时间内完成。第二，这种范型的价值最好从这些概念的相应的应用中了解，而这只能通过项目才能做到。第三，实际的项目不像能够在一天编写完成的简单的小程序那样，它能够让人对实际项目的完成有一个感性认识。第四，项目是在小组中开发的，在程序开发过程中会出现讨论、决策和撤销决定的过程。第五，对于多数大学生来说，这将成为具有实际规模的第一个程序，他们要进行详细说明、分析和设计工作。最后，选择一个大小合适的项目，可以让学生们掌握所有的关键概念。

在学校中使用的典型项目是一个大型的冒险游戏，游戏中的角色要在虚拟世界中寻宝、与魔怪和坏人搏斗，最终达到目标。这些游戏有代表性地引入了一百个类和同样多的关系。这些游戏中有多种不同的地形、武器、魔怪、财宝和角色。随着网络游戏的普及，许多项目组选择开发多人玩的游戏。多数情况下，项目组开发的游戏具有与许多商业产品相同的复杂度。

项目组的合理构成是三到四个学生。太大的项目组会需要太多的时间用于达成一致意见，而过小的项目组又会力不从心。项目组在上课时开展项目工作，这样指导教师就可以检查进度，并回答与概念应用有关的问题，所以每个班级的人数要适量。学生每周都要进行项目工作，这样才能按时完成项目，应根据课程安排作业。

下面是建议的课程作业安排。假定是标准的 15 周的时间表，第 15 周考试。这个安排表将在学期开始安排大量时间用于定义游戏和开发用例。有时课程内容会比项目组完成的作业提前三周。已经证明这是有好处的，因为这有助于学生少犯一般性错误，比如混淆了属性和关联或者带有对象属性的对象状态。

Preface

Practical Object-Oriented Development with UML and Java is for busy professional software analysts and developers who work on large systems. If you do not have time to take a class and need to get up-to-speed on object-oriented technology using unified modeling language (UML) and Java, then this book is a self-teaching guide for you. It will help you understand the differences between object-oriented analysis, object-oriented design, and object-oriented programming. Our goals are to

- Teach you to build an object-oriented application using Java and make the right trade-off decisions to meet your business needs
- Clarify the basic concepts associated with object-oriented technology
- Supply sufficient depth in coverage for students and practitioners entering the field to get them up-to-speed
- Expose some of the myths surrounding object-oriented technology while focusing on its practicality as a software engineering tool
- Provide a practical approach to analysis, design, and programming in object-oriented technology
- Show how to implement object-oriented technology using Java
- Balance theory with application practices in the existing literature

You do not have to know computer science or advanced mathematics to understand the important object-oriented concepts and issues in depth. Even the programming chapters do not require a background in Java; they illustrate how working code in Java is produced.

OBJECT-ORIENTED TECHNOLOGY

We are software developers of large systems. We have delivered code written in several dozen programming languages representing a half-dozen software technologies. There have been few software revolutions that we have not experienced over the last 30 years. So it is from some nontrivial perspective that we say that it is our belief that object-oriented technology is the most important software technology with which we have worked.

Why do we say this? Well, object-orientation has changed the way we build software and the way applications intercommunicate over worldwide networks and across multi-vendor computers. Moreover, the object model is changing the way we design business processes and the way we think about an enterprise.

Most enterprises are in the process of redesigning themselves to meet current business challenges introduced by the Internet. Object-orientation is playing a major role in this effort by providing a model that captures the business processes, procedures, policies, and rules that facilitate design. The use of tools that translate the model into an operational system speeds implementation of the redesign. As market or business conditions change, these systems should be regenerated to reflect these changes by updating the model and using these tools. Solid software engineering practices have taken us farther and faster than any other approach in previous decades.

It is a common belief that object-oriented technology has put a dent in the software crisis, meaning that the mechanisms of object-oriented technology are becoming for software what the bolts and beams are for construction design and what the chip is for computer hardware design. This belief stems from the following:

- The proficiency of a higher-level object-oriented model provides the software designer with real-world, programmable components, thereby reducing software development costs.
- Its capability to share and reuse code with object-oriented techniques reduce time to develop an application.
- Its capability to localize and minimize the effects of modifications through programming abstraction mechanisms allows for faster enhancement development and provides more reliable and more robust software.
- Its capability to manage complexity allows developers to address more difficult applications.

The collection of object-oriented concepts is a tool set for modeling reality. This object-oriented tool set gives developers the best means of managing the complexity. Certain object-oriented concepts help developers produce flexible and maintainable software.

WHY UNIFIED MODELING LANGUAGE?

As practitioners of object-oriented technology, we know that all the methods, if practiced properly, result in the same or a similar model. Different modeling language notations, however, can be impediments to progress. The unified modeling language (UML) has become an industrial standard that has integrated different modeling notations into a single modeling language notation. This is reason enough to have chosen the UML.

UML is a language for documenting our analysis and design models. It gives us all the drawing icons necessary to capture most of the concepts or *mechanisms* that we find valuable in solving real business problems. Also, it provides all the necessary diagrams that are vital for documenting our models. Finally, it is a living language that gives us the ability to extend the notation for mechanisms not yet defined by the distinguished group of Grady Booch, James Rumbaugh, and Ivor Jacobson at Rational Software Corporation.

UML is not the central subject of this book. It is presented as a means of documenting the analysis and design models that are developed as a result of the methods that are the central subject of this book. All of the figures of UML are presented and discussed in terms of what information is captured within them and how that information is captured.

WHY JAVA?

It is true that Java is exclusively an object-oriented programming language and that this exclusivity tends to limit its use compared to the multi-paradigm programming language C++. Yet Java has one benefit that far outweighs any general limitations. In particular, Java runs on the Java Virtual Machine (Java VM). This allows a Java program to run on any machine that has an implementation of the Java VM running on it. This frees developers from having to design and implement the same functionality for several different combinations of hardware and operating systems.

There are positive consequences to the use of the virtual machine that may not be apparent at first glance. For one, vendors can now focus on development of tools and products knowing that they have to invest development dollars on only one implementation and not five or six. This means that they can emphasize the realization of greater functionality (read that as greater business value). This impacts reuse efforts as well. One can develop libraries on any hardware running any operating system and reuse the code without modification for platform differences. Programming errors will not appear in one version of the code for a given platform and not in another. The broader base of reuse of the same code means that greater reliability of components can be achieved in less time and cost. Analysis can focus on business value; design can focus on greater flexibility and maintainability; and implementation and testing can now focus on quality, reliability, and performance. The net result of this change in focus is better code for less money.

These benefits are seen when one looks at the large number of Java libraries (frameworks) that are now available from Sun. There is now a good set of general utility libraries, a high-performance graphical user interface library (e.g., SWING), and libraries of special-purpose business classes available to developers. Compared to the C++ versions of these libraries, they are far more sophisticated, provide much greater functionality, and are more easily incorporated into a final product. Because these libraries are available across all projects and are well documented in other books, they are widely used. Hence, expertise in their use is readily available.

This is to be compared to the C++ versions that have entirely different application programming interfaces (APIs). In the C++ world, a developer may be an expert for one product on one platform and know nothing about other products for other platforms. Few programmers have actually written code for the PC, Mac, and Unix platforms.

The direct incorporation of the Java VM in modern web browsers makes it possible for Java programs to be downloaded from the Internet and run from within the browser. This has helped provide greater functionality within the browser environment and has spawned a new class of applications. It is safe to say that it would have been impossible for us to have achieved the recent gains in functionality being delivered to users via the Internet without Java. Java programs now appear as both client and server applications. The widespread use of Java in the modern World Wide Web is unlikely to diminish until a new (as yet unrecognized) technology provides a greater set of abstractions and the same broad platform support.

OUR APPROACH TO OBJECT-ORIENTED TECHNOLOGY

We are not object-oriented purists, and neither are we theorists. We are developers willing to use any good idea that will help us achieve two very critical business goals: lower development cost and reduced time-to-market for enhancements. We believe that these technical objectives—reliability, maintainability, and flexibility—are critical to meeting these business goals.

Our approach to using object-oriented technology is to mange the complexity of developing software so it is reliable, maintainable, and flexible. Managing complexity is the key to achieving these objectives and, thus, our business goals. To manage complexity in complex problem domains, we find that the developers are required to know how objects, classes, relationships, and rules fit into the object paradigm. When we model most complex problem domains, we find objects, classes, and many relationships among objects. In addition, we need to capture the rules (policies) within that domain. Thus, we have to use very rich static modeling techniques to capture the data (object) relationships.

Many object-oriented experts consider relationships as "bad" because they violate the encapsulation principle. From our perspective, it helps us manage the complexity of the problem domain and helps us to achieve our business goals. We gladly use it, and we look for more mechanisms and language support in this area. In Chapter 9 on declarative semantics we write that rules and policies should be captured as an integral part of our model and not in special subsystem extensions.

Using mechanisms to help us model complex problem domains is consistent with our choice of UML as our modeling language and Java as our programming language. Both UML and Java allow us to define any needed mechanism that helps us to build more manageable software.

We discuss behaviors (dynamic and static) and polymorphism for capturing the procedural aspects of the model. The use of finite state machine or some other state model helps us manage procedural complexity while addressing timing, synchronization, and interrupts. We also present exceptions for managing error recovery (an important topic because error recovery can comprise half of a programs logic). These areas are generally ignored or overlooked by most object-oriented books.

We believe the key to success in building large object-oriented systems requires that developers and programmers know more than what is taught in most object-oriented books. Building large systems requires using mechanisms promoted by some object-oriented experts but not accepted by all. Professional developers need to at least understand how these aspects of the problem domain can be handled before they can be productive team members. This book will not make you an expert. You still need experts or consultants to develop the system. By applying the 80/20 rule, this book provides the 80 percent that can make you productive and understand how the experts solve the difficult 20 percent.

In this book we do not cover the latest trends or fads in object-oriented technology, including object design patterns, the standard template library, and distributed object computing. Although they are interesting, we are not convinced that they contribute significantly to our goal of providing a practical framework for enabling developers new to object-oriented programming to get up-to-speed as soon as possible.

Finally, we do not agree with most experts that object-oriented technology is a mature technology. We believe it is maturing. Object-oriented technology has the enormous potential to help us manage complexity that did not exist with the earlier technologies (procedural, functional, rule-based, etc.). We see in object-oriented technology and Java many different abstraction mechanisms merging (integrating) into a truly powerful technology. This merging is not yet complete, but it is far more complete in Java than in any other endeavor in object-oriented technology.

ORGANIZATION OF THE BOOK

We take the reader through our rational in applying object-oriented techniques and methods. These are not a set of absolute laws. Our goal is to make you think about good object-oriented concepts and good design principles when developing software and programming in Java.

We have written and designed this book to be a self-teaching guide that should be read in sequential order. We have adopted a method that Richard has used for years teaching object-oriented concepts and basic skills; however, we do not advocate this as a method for building object-oriented systems. Each chapter discusses a major step of our approach to object-oriented technology. Most chapters conclude with a step-by-step guide or recipe. We hope the reader will use these steps only as a guide; always rely on common sense rather than following prescribed steps blindly.

Chapter 1 introduces abstraction as a mechanism for controlling complexity and establishes object-orientation as the modern inheritor of the long line of abstraction mechanisms.

Chapter 2 presents the basic principles of object-orientation.

Chapter 3 begins the process of development by using the use-case approach to develop a specification model.

Chapter 4 begins the process of developing an analysis model by identifying objects/classes/interfaces.

Chapter 5 describes how to differentiate between "real" objects and "false" objects by identifying attributes (data) and services associated with the object.

Chapter 6 demonstrates how to capture objects' behavior.

Chapter 7 describes how to identify and describe dynamic behavior.

Chapter 8 describes the various relationships (generalization/specialization, link, object aggregation, etc.) that are available for organizing all the objects in the system.

Chapter 9 describes how to incorporate declarative facts into the object-oriented model about object knowledge and a rule-based mechanism for their implementation.

Chapter 10 reviews the analysis model and restructures it to take into account helper classes.

Chapter 11 addresses some elements of developing the design model.[1]

Chapter 12 presents programming in the Java language.

Chapter 13 introduces how classes and interfaces are implemented using Java.

Chapter 14 describes how static behavior can be implemented.

Chapter 15 describes how dynamic behavior can be implemented.

Chapter 16 describes how generalization/specialization can be implemented.

Chapter 17 describes how additional relations can be implemented.

Appendix A presents a summary guide of the unified modeling language.

Appendix B presents a summary of Java.

Appendix C presents a comparison of Java and C++.

USING THIS BOOK

This book is primarily targeted at experienced software developers and upper-level college students. It is based on the material taught in industrial courses attended by

[1] Design is very complex and, from a system perspective, is a separate topic from object-oriented technology. However, Chapters 14 through 22 present design material from the perspective of how the analysis concepts can be implemented. Idioms and design patterns are presented in all of the implementation chapters.

competent programmers. The material of this book is presented in two courses of one-week duration. The first week covers the first 11 chapters while the second week covers the remainder of the book. This course has always been taught utilizing a project in which the students develop a computer game rather than a homework-based approach. At the end of the first course, students have a design for a game. At the end of the second course, students have a fully functional implementation of their design.

We have chosen the project-based approach for several reasons. First, we have found that homework problems either are too trivial to effectively communicate the significance of the concepts or are too complex to be performed in a reasonable period of time. Second, the value of this paradigm is best learned from the consistent application of the concepts that can only be achieved via a project. Third, a substantial project gives a sense of real accomplishment as the projects are not simple little programs that can be finished in a single day of programming. Fourth, the project is developed in a team context with the periods of discussion, decision making, and reversals of decisions that actually occur when developing a program. Fifth, for most university students this will be the first program of substantial size that they will have to specify, analyze, and design. Finally, the selection of a project of suitable scale enables the student to master all of the key concepts.

The typical project that is employed in a university environment is a large adventure game in which characters explore some virtual world picking up treasures, fighting monsters or villains, and achieving some final objective. These games typically incorporate a hundred classes and just as many relationships. These games include many different kinds of terrain, weapons, monsters, treasures, and characters. With the widespread use of networked games, many project teams have chosen to develop multiplayer games. In most cases, the games developed by project teams have the same levels of complexity as many commercially developed products.

A reasonable project team consists of three or four students. A larger team spends too much time coming to agreement and a smaller team tends to become overwhelmed. The team works on the project during class time so that the instructor can review progress and answer questions concerning the application of the concepts, so class size is kept to a manageable size. The students have to work on the project weekly so as to complete the project on time, and activities are scheduled to correspond with lectures.

Following is a suggested schedule of course activities. It assumes a standard 15-week schedule with the final exam given in week 15. A key feature of this schedule is that it allows generous time early in the semester to define the game and develop the use cases. Lectures occasionally precede the activities performed by the team by as much as three weeks. This has been found to be advantageous because it prevents students from making common mistakes, such as confusing attributes and associations or object state with object attributes.

Acknowledgments

We owe so much to many people. The impetus of this book came from requests from people who wanted a book that presented more of an engineering approach to the development of Java programs. We appreciates their persistence and their encouragement, ignoring the perils to our personal lives.

Because we are basically developers (not researchers, academics, or writers), we have leveraged off the work of object-oriented researchers (who originated all the ideas) and object-oriented writers (who presented these ideas to us in earlier writings). We simply apply these ideas to building real applications in a useful way. To all the originators of the ideas, concepts, mechanisms, and techniques and to the greater object-oriented writers before us, we acknowledge them; without them, this book would not have been possible.

Theories and ideas are wonderful; however, to practitioners, experience is the best teacher. We could not produced this book without our experiences in applying object-oriented technology and the methods to real projects. We thank our many bosses, present and past, who had the courage to let us do leading-edge (and many times bleeding-edge) software development. Without their support, we would not have been able to test what we have written. We also thank reviewers William McQuain of Virginia Tech University and Michael Huhns of the University of South Carolina.

Richard Lee thanks the multitude of people who have worked for him and who were the pioneers in applying the ideas written in this book on real projects. They shared with him both the excitement and the misery of being "first" to apply Java technology to large projects in their respective companies. To all of you, Richard owes his thanks.

William Tepfenhart thanks his colleagues at Monmouth University for their support while putting together this book. Most importantly, he appreciates the patience of his family and their willingness to accept that there were times when deadlines took priority.

RICHARD C. LEE
WILLIAM M. TEPFENHART