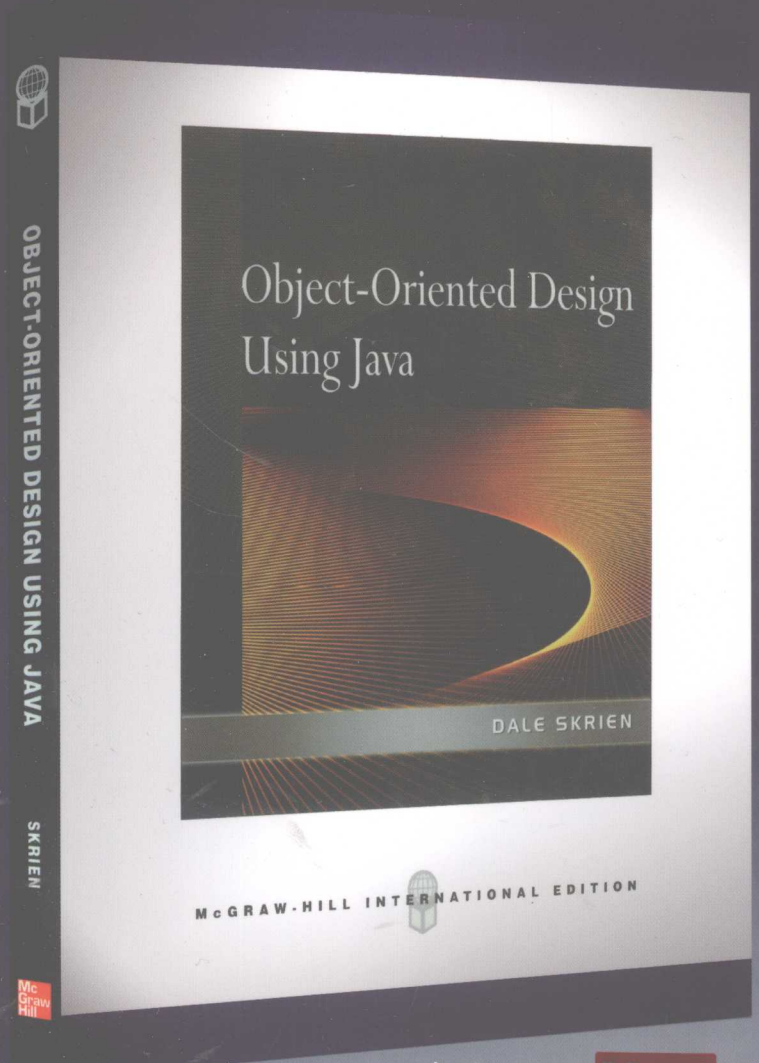
国外计算机科学经典教材

面向对象设计原理与模式 (Java版)

(美) Dale Skrien 著　　腾灵灵 仲婷 译



Object-Oriented Design Using Java

Mc Graw Hill Education

清华大学出版社

国外计算机科学经典教材

面向对象设计原理与模式

(Java 版)

(美) Dale Skrien 著

腾灵灵 仲婷 译

清华大学出版社

北 京

Dale Skrien
Object-Oriented Design Using Java
EISBN: 978-007-126387-0

图书在版编目(CIP)数据

面向对象设计原理与模式(Java 版)/(美)斯科瑞(Skrien, D.)著；腾灵灵，仲婷 译.
—北京清华大学出版社，2009.4
书名原文：Object-Oriented Design Using Java
(国外计算机科学经典教材)
ISBN 978-7-302-19671-6

Ⅰ.面…　Ⅱ. ①斯…②腾…③仲…　Ⅲ.Java 语言—程序设计　Ⅳ.TP312

中国版本图书馆 CIP 数据核字(2009)第 032611 号

责任编辑： 王　军　于　平
装帧设计： 孔祥丰
责任校对： 成凤进
责任印制： 何　芊

出版发行： 清华大学出版社　　**地　　址：** 北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编：** 100084
社　总　机： 010-62770175　　**邮　购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn

印 刷 者： 北京四季青印刷厂
装 订 者： 三河市李旗庄少明装订厂
经　　销： 全国新华书店
开　　本： 185×260　**印　张：** 20.5　**字　数：** 499 千字
版　　次： 2009 年 4 月第 1 版　　**印　　次：** 2009 年 4 月第1 次印刷
印　　数： 1～4000
定　　价： 36.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：(010)62770177 转 3103　　产品编号：029691-01

出 版 说 明

近年来，我国的高等教育特别是计算机学科教育，进行了一系列大的调整和改革，亟需一批门类齐全、具有国际先进水平的计算机经典教材，以适应我国当前计算机科学的教学需要。通过使用国外优秀的计算机科学经典教材，可以了解并吸收国际先进的教学思想和教学方法，使我国的计算机科学教育能够跟上国际计算机教育发展的步伐，从而培养出更多具有国际水准的计算机专业人才，增强我国计算机产业的核心竞争力。为此，我们从国外多家知名的出版机构 Pearson、McGraw-Hill、John Wiley & Sons、Springer、Cengage Learning 等精选、引进了这套“国外计算机科学经典教材”。

作为世界级的图书出版机构，Pearson、McGraw-Hill、John Wiley & Sons、Springer、Cengage Learning 通过与世界级的计算机教育大师携手，每年都为全球的计算机高等教育奉献大量的优秀教材。清华大学出版社和这些世界知名的出版机构长期保持着紧密友好的合作关系，这次引进的“国外计算机科学经典教材”便全是出自上述这些出版机构。同时，为了组织该套教材的出版，我们在国内聘请了一批知名的专家和教授，成立了专门的教材编审委员会。

教材编审委员会的运作从教材的选题阶段即开始启动，各位委员根据国内外高等院校计算机科学及相关专业的现有课程体系，并结合各个专业的培养方向，从上述这些出版机构出版的计算机系列教材中精心挑选针对性强的题材，以保证该套教材的优秀性和领先性，避免出现“低质重复引进”或“高质消化不良”的现象。

为了保证出版质量，我们为该套教材配备了一批经验丰富的编辑、排版、校对人员，制定了更加严格的出版流程。本套教材的译者，全部由对应专业的高校教师或拥有相关经验的 IT 专家担任。每本教材的责编在翻译伊始，就定期不间断地与该书的译者进行交流与反馈。为了尽可能地保留与发扬教材原著的精华，在经过翻译、排版和传统的三审三校之后，我们还请编审委员或相关的专家教授对文稿进行审读，以最大程度地弥补和修正在前面一系列加工过程中对教材造成的误差和瑕疵。

由于时间紧迫和受全体制作人员自身能力所限，该套教材在出版过程中很可能还存在一些遗憾，欢迎广大师生来电来信批评指正。同时，也欢迎读者朋友积极向我们推荐各类优秀的国外计算机教材，共同为我国高等院校计算机教育事业贡献力量。

清华大学出版社

国外计算机科学经典教材

编审委员会

0.1 本书简介

本书是关于面向对象设计(object-oriented，OO)的介绍，主要面向计算机科学二年级或者更高年级的本科生。本书根据“优雅”(elegance)一词讨论了软件的设计和实现，该词将在第 1 章中定义。本书将首先回顾面向对象编程的概念，然后介绍软件设计的基本技术、编码风格、重构、UML 和设计模式的内容。

本书通过多个实例、一个小型案例分析和两个中等规模的案例分析来介绍设计原则和模式。这些原则和模式都是在需要利用它们来解决设计方面的问题时引入的。在本书中，多个实例和案例分析都会从针对问题的一个“明显的”解决方案开始，然后通过讨论该初始解决方案的优缺点，逐步将其优化为更加优雅的解决方案。

本书并不是要致力于成为关于编码和设计优雅性的完整的、权威的“圣经”。实际上，本书是对于在达到优雅性的这个目标的过程中将会涉及到的很多话题的一个介绍。本书为需要进一步了解这些话题的学生提供了进一步学习的参考文献。

本书中所有的讨论和例子都基于 Java 1.5 给出，但是提出的概念和原则几乎都具有通用的面向对象的属性，所以可以适用于利用其他面向对象语言的设计和实现。

本书包括了大量的练习题。从第 2 章到第 9 章，每章平均有 18 道练习题。这些练习题包括考察学生关于该章所介绍内容的理解程度的简单测试，但大部分的练习题包括的内容都无法在正文中加以介绍，否则，就会偏离正文的主题太远，并且导致正文过于庞大。

0.2 写作目的

ACM/IEEE 的模式课程(ACM/IEEE model curriculum，CC2001)提出“编程的介绍性课程通常过于简化了编程的过程，以使其适用于新生，这些课程相对于概念上简化了的编码过程，对于设计、分析和测试强调得太少了。所以，学生掌握了编程技术的表象，掩盖了他们其实缺乏基本技术的事实，这将会限制学生在未来适应各种不同问题的能力，并限制他们适应解决问题的各种背景的能力”[1]。

上面的叙述很好地总结了笔者在过去 20 年讲授计算机科学的过程中遇到的情况，尤其是当笔者所在的系 10 年前使用 Java 作为介绍性课程以及许多高级课程的实现语言时遇

到的情况。在我们的学生使用 Java 完成了 CS1 和 CS2 的课程后，他们知道如何设计和编写可以工作的代码，但是笔者在发现他们并不知道如何设计并编写可读、可维护的代码时，感到十分挫败。同时，当笔者发现这些学生对于 Java 的面向对象特性(继承、接口、静态变量/实例变量、静态方法/实例方法、动态方法调用)的理解十分肤浅时，也非常有挫败感。比如，如果事先向他们提供继承或者接口，他们会使用，但是如果要求他们自己设计软件的时候，他们就不能恰当地使用继承或者接口。

这样的挫败感部分来源于 CS1 和 CS2 课程自身，这些课程需要覆盖太多其他材料，所以在一个学期内没有时间以一种有意义的方式提到面向对象设计和实现的原则。问题之一在于面向对象设计原则事实上在大的应用中比较能体现出自身的优势，但是似乎对于同学们来说，面向对象设计原则对于小的应用，比如我们的 CS1 和 CS2 的课程，就大材小用了。在高级课程中，面向对象设计原则将充分发挥作用以协助大型的编程项目，但是此时，学生们却无法鉴赏好的设计，因为他们几乎没有对于大型项目的经验，无论是好的项目还是坏的项目。

作为一名教授，笔者强烈觉得面向对象设计，包括设计模式的使用，是学生们需要具备的重要的技术。而且，笔者希望能和学生们分享一下对于“优雅”设计的热爱。希望同学们对于软件设计能够培养出一种审美感，使得他们可以分辨什么时候软件“闻起来很糟”(smell bad，由 Martin Fowler[2]提出)，什么时候闻起来很好。希望他们可以将审美感培养到这种程度，即看到优秀的代码时能“全身上下感受到喜悦的颤动”(正如笔者在课程中曾经提出的)，而对拙劣的设计则会“反感到想要逃离”。Donald Knuth[3]说过计算机编程既是一门科学，也是一门艺术，它包含创造力和美感。希望同学们可以向艺术家一样，尝试创建具有美感的对象。当成功创建的时候，他们会对自己的工作感到足够的骄傲，以致于同学们会想要像艺术家一样，在作品上签上自己的名字。

本书正是为了帮助指导学生设计和编写出漂亮的面向对象软件。

0.3 读者对象

本书面向已经学习过编程的介绍性课程(CS1)和数据结构课程(CS2)的本科生，但是这些本科生的课程中还没有涵盖大量关于面向对象设计原则和设计模式的内容。本书可以作为大学二年级学生的、循序渐进的第三课程的主要课本，或者本书也可以作为数据结构课程的补充教材。又或者，本书可以作为关于面向对象软件设计的本科生的高级课程的主要教材或者补充教材。

本书假设学生们已经具有如下的背景：

- 编程经验，例如在 CS1 课程中获得的经验
- 关于数据结构的基本知识，例如在 CS2 课程中学习到的知识
- 关于 Java 基本原理的理解，如果某个课程曾经是用 Java 进行教学的，那么就应该具有这些知识
- 对于下面的 Java 特性有一些熟悉：
  - 泛型(来自 Java 1.5)

- 抛出和捕捉异常
- 内部类
- AWT 或者 Swing 程序包(第 8 章)

如果对这些话题并不熟悉的话，学生可以在课程的合适阶段学习相关内容。本书和一些可选练习题中都提及了 Java 的反射能力以及 Java 线程，但是并不需要提前具备这些知识。

本书使用了四种 UML 图(状态图、顺序图、状态机图和用例图)，但是不需要提前具备 UML 的知识，因为本书会在第一次使用到 UML 的时候，对进行适当的介绍。本书还将在附录 A 中对这四种图进行更深入的介绍。

0.4　教学方法

本书试图通过软件解决方案的演化，指导学生进行面向对象的设计和实现。正如前面提到的，在学生们学习了一系列的介绍性课程之后，他们将知道如何找到解决方案，但是通常却无法分辨不同的解决方案的区别。因此，本书试图首先介绍问题，然后讨论学生很可能首先提出的“明显的”、但是往往是不优雅的解决方案，逐步建立学生对编程过程的理解。然后讨论该解决方案的优缺点，随后给出更好的设计。接着再次讨论新解决方案的优缺点，而这往往会引出更好的设计。通过这种方法，本书在用优雅的方式解决特殊问题的过程中，还将介绍所有重要的软件设计原则和模式。

0.5　本书内容

本书大致可以分为四个部分。

第一部分(第 1 章)十分简短，实际上类似于简要的介绍。它为本书的其他部分提供了出发点和动机，所以对于理解后续章节的内容是十分重要的。

第二部分(第 2~3 章)包含了对面向对象基本概念的回顾，比如对象、类、方法、继承、多态和动态方法调用。这部分还讨论了继承的正确使用和错误使用。

第三部分(第 4~6 章)介绍了设计和实现的优雅性。该部分首先讨论了底层实现的话题，例如编码约定和术语。然后讨论了通用的设计原则。接着介绍一个小规模的案例分析（Money 类的设计）。

第四部分(第 7~9 章)介绍了设计模式。第 7 章介绍了设计模式的概念，并讨论了几个最简单的模式。随后是两个中等规模的案例分析(一个绘图应用程序和一个语言分析器)。这些章节在案例分析中涉及到新的特性时，逐渐会介绍一些新的模式和原则。

除此之外，本书还包含了两个附录。附录 A 讨论了关于本书使用的四个 UML 模式(类、顺序、状态机和用例)的更细节的内容。附录 B 涵盖了 Sun Microsystems [4]提供的 Java 的许多编码约定，尤其是 Javadoc 的标记法。除了 Javadoc 外的大部分编码约定都可能是学生

熟悉的内容，但是出于完整性的考虑，本书还是将这些内容列在了附录 B 中。

下面是关于各章内容的详细信息：

1. 第 1 章是对本书的介绍和总结。本章定义了“优雅的”的软件，即可用的、完整的、健壮的、高效的、可伸缩的、可读的、可重用的、简洁的、可维护的以及可扩充的软件。笔者试图激发学生学习面向对象设计的动力，这样学生才能够领悟学习本书的价值。

2. 第 2 章是关于面向对象编程的基本特性的回顾。第 2 章的目的就是回顾这些学生可能已经学过，但是却没有能够很自如地运用的面向对象的主题。本章提供了学习这些特性的一些动机，这样学生就可以开始培养一种关于面向对象编程相对于非面向对象编程的优点的审美感。

3. 因为实现继承十分复杂并且很容易用错，所以笔者认为需要对此话题进行大量的讨论。笔者将此内容另设了一章，而不是全部填塞在第 2 章中。所以，第 3 章讨论了什么时候应该使用实现继承，以及什么时候不应该使用实现继承。该章首先讨论了继承的 4 个观点，并给出例子来证明，这些观点中大部分都无法独立地显示其充分性。本章还讨论了在什么情况下，应该使用委托来代替继承。

4. 优雅性不仅对于设计阶段十分有用，而且在实现阶段也十分有价值。因此，笔者增加了第 4 章来讨论实现方法时应该遵循的原则，第 4 章也包括一些编码约定。在本书的第 4 章增加这些内容，也是为了便于在后续章节的样例代码和案例分析中涉及到相关内容的时候，可以进行查阅。笔者也希望以一些学生已经熟悉的内容来开始第 4 章的介绍。第 4 章还对 Java 中的 equals 方法和 clone 方法进行了大量的讨论，以展现这样一个观点：当需要遵循一些原则时，就需要十分仔细地对方法进行实现。

5. 第 5 章开始了对于面向对象设计的重点讨论。本章包括了面向对象设计的许多基本原则，对于每个原则，都将介绍其动机以及一个相关例子。

6. 因为这样的设计原则只有在大型(至少不是小型)的例子下才能更好地鉴赏，所以笔者在第 6 章中列举了一个这样的例子作为案例分析。笔者希望能列举一个学生感兴趣的例子，但是该例子同时也要能够引出一些有趣的设计话题。因而，笔者选择 Money 类的设计作为分析的案例。

7. 第 7 章介绍了设计模式。本章通过讨论几个最简单的模式来展示什么是设计模式，以及它们如何在开发优雅的设计中发挥作用。

8. 第 8 章包含了关于绘图程序的一个较大型的案例分析。该应用基于 Kent Beck 以及 Ralph Johnson[5]开发的 HotDraw 架构以及 John Vlissides[6]的部分工作。首先给出了一个简单的绘图程序，然后逐渐增加新的特性。每个特性的优雅设计通常都会引入一个新的设计模式。

9. 最后一章介绍了另一个案例分析。该案例分析是关于 Java 子集的分析器。本章中又介绍了几个新的设计模式，例如，本章在讨论由分析器生成的抽象语法树的使用时引入的访问者(Visitor)模式。

0.6　各章关系

第 1 章应该最先阅读，因为它为后续的章节提供了上下文环境，并且定义了设计和代码的“优雅性”。

第 2 章和第 3 章涉及面向对象的基本概念，如对象、类、多态、动态方法调用和继承。所有后续的章节都将会使用这两章的内容。如果学生对这些话题已经非常熟悉，那么可以略过这两章。

第 4 章和第 5 章是本书的核心内容，所以所有的学生都需要阅读这两章的内容，尽管第 4 章的大部分内容对于某些学生而言，可能是十分熟悉的了。

第 6 章是选读内容。这一章的案例分析使用了第 4 章和第 5 章中讨论过的设计原则。

第 7 章介绍了设计模式。如果学生对这个概念已经非常熟悉，就可以略过这一章。但是学习这一章中介绍的模式对学生来说，还是非常有益的。对于设计模式的动机来源于第 4 章和第 5 章的内容。

第 8 章和第 9 章是选读内容。这两章在两个案例分析的上下文中介绍了一些新的设计模式。这两章依赖于第 4、5、7 章中介绍的内容。

附录 A 和附录 B 是选读内容。在本书第一次使用 UML 图的时候，会简要解释这些图和图标。

0.7　本书和 ACM 模式课程(CC2001)[1]的匹配性

CC2001 课程的两个核心知识单元就是 PL6 和 SE1。这两个单元包括如下主题：

PL6　面向对象设计
- 封装和信息隐藏
- 行为和实现的分离
- 类和子类
- 继承(重写、动态调度)
- 多态(子类型多态与继承)
- 类层次结构
- 集合类和迭代协议
- 对象和方法表的内部表示

SE1　基础设计概念和原则
- 设计模式
- 软件体系结构
- 结构化设计
- 面向对象分析和设计
- 组件级别设计
- 可重用设计

本书实际上包含了这些知识单元的所有内容。本书的案例分析也包含了 PF5 的内容、HC2 的大部分内容以及 PL3 和 PF3 的部分内容。将 CC2001 特别工作组认为是必要的中等规模的团队开发项目，与使用本书的课程相结合是一件很好的做法。

0.8 主要特色

1. 根据代码“优雅性”讨论设计和实现。

2. 使用几个小型和两个中型案例分析来介绍设计原则和模式。在案例分析中需要解决设计方面的问题时，会引入这些原则和模式以解决问题。

3. 许多例子和案例分析都从对问题的一个“明显的”解决方案开始，然后通过讨论这个初始方案的优缺点，将它逐步演化为更加优雅的解决方案。

4. 讨论了如下的话题：

(1) 面向对象的基本概念(类、对象、继承、多态、方法)。

(2) 编码风格和实现的话题，如合适的命名以及文档，包括前置条件、后置条件和不变式。

(3) 基本的面向对象设计原则，实际上包括 CC2001 模式课程中 PL6 和 SE1 知识单元的所有主题。

(4) 设计模式。大部分设计模式均在解决某个问题的背景中引入。

5. 在每章的最后都有大量的各种难度的练习题(总共 144 题，平均每章 18 题)，从考察学生对于该章节知识的理解程度的简单测试，到要求学生对现存的代码进行大量修改或者改善的练习题。

0.9 其他特色

教师用书包含大部分练习题的答案，以及配合本书可以布置给学生的一些中等规模的项目的描述。该书还包括了讲授课程的教学建议。

0.10 参考文献

1. *Computing Curricula 2001, Computer Science Volume.* 2001. [Cited March 28, 2007; available from *http://www.sigcse.org/cc2001/cs-introductory-courses.html.*]

2. Fowler, M., *Refactoring, Improving the Design of Existing Code.* Object Technology Series. 1999. Addison-Wesley. Reading, MA.

3. Knuth, D., *Literate Programming*. 1992. Chicago, IL: University of Chicago Press.

4. Sun Microsystems, I. *Code Conventions for the Java Programming Language.* 1999. [Cited March 28, 2007; available from *http://java.sun.com/docs/codeconv/html/ CodeConvTOC.doc.html.*]

5. Beck, K. and R. Johnson, Patterns generate architectures. In *European Conference on Object-Oriented Programming (ECOOP'94)*. 1994. Bologna, Italy: Springer-Verlag.

6. Vlissides, J., Tooled composite. C__ Report. September 1999.

目录

第 1 章

面向对象设计与实现的优雅性

对于阅读本书的大学生而言，您很可能已经学习了一些计算机科学的相关课程，其中有一门对于编程的介绍性课程。经过认真的学习，您一定掌握了至少一门编程语言的基本编程概念。您学会了诸如循环、赋值和条件语句等语言结构，也学会了编写这些语言结构的一种或多种方法。您更有可能已经理解了计算机科学并不仅仅是编程而已，事实上，编程仅仅是计算机科学工作者的一种工具。

遗憾的是，因为没有时间讲解计算机科学中如此众多的主题，所以您将无法获悉解决同一问题的各种不同方法的优缺点。举例来说，在较低的层次上，存在很多种方法来实现一个结构，例如循环(比如，可以使用 for 循环，while 循环或者递归)，其中的某些方法相对而言会更加优越。在高一点的层次上，存在多种方法来实现一个任务(如归并排序或者选择排序)，其中某些相比而言更为快速。再高一点的层次上，存在各种方法来将一个程序划分为诸如类或者方法的模块，其中某些分割的方法会更加合适。

我们来看一个最后那种情况的例子。请考虑这样一个问题，该问题需要编写一个模块来存储集合，该集合包括一些人以及他们各自的狗的信息。这个集合需要被保存为散列表、数组还是其他的集合形式？这个集合是需要存储 Person 对象还是 Person-Dog 这样的对象组合。Dog 对象是否需要有实例变量指向其主人？而狗的主人又是否需要实例变量指向他所拥有的狗的集合？如果是的话，这又将是一个怎样的集合？这样的 Person 对象还需要存储哪些其他数据？如果这些人是美国公民，是否需要存储他们的社会保障号码？是否需要将这样的人的集合排序以便于查找？如果一个人的某条狗死了，是否需要将其从这个集合中删除，或是将其保留为一个没有狗的人？

很自然地，您会疑惑这些问题的答案是否真的很重要。如果两个设计，一个只有少数几个类和方法，另一个则包含很多类和方法，当这两个设计都能解决问题的时候，具体使用哪个设计是否真的很重要？类似地，如果所有版本的循环都能正确地完成计算，具体使用哪个版本是否真的很重要？如果两个算法都能正确计算，哪个运行更快是否真的很重要，尤其是在考虑到每年硬件处理速度飞速增长的情况下？

答案是“对，这些都很重要”。本章将尝试解释其中的理由。此外，本章将给出适用于编程的设计和实现的一系列标准，以帮助您对程序做出评判。本书的后续章节将给出一些能反映软件工程师的最佳实践的准则，以便您从这些专家的经验中领悟到什么是好的什

么是不好的。

本书从头至尾尝试阐述的不仅是哪些是适用的，并且会讨论是什么使得某些软件很“丑陋”而其他软件很“优雅”或者很“漂亮”。也就是说，本书希望帮助您培养一种设计和编码的审美观。数学家基于优雅性来评价理论的各种证明——两个证明可能都是正确的，但其中一个却比另一个更加漂亮——计算机科学家也可以利用类似的方式来评价软件设计。这或许需要通过扩展审美判断的感知面来做到。但是如果能把审美观培养到如下这种程度就更好了，即，一看到优秀的设计和编码便能欣喜若狂，而面对拙劣的设计则想要逃。

1.1 存在的问题

一个压根没有考虑设计标准就将代码快速拼凑在一起的编程人员可能会这样评价他自己的行为：他知道这段代码都做了些什么，所以不必担心会产生曲解。并且，他还可能认为，这些代码不会被重用或被其他人重读，那么何必还要在设计上花费不必要的时间呢？

对于只由一个人编写并且只使用一次的“快而脏”(quick-and-dirty)的程序，例如简短的 Shell 脚本程序而言，这位编程人员的想法是对的。花费大量的时间来编写最优雅的程序版本并不高效。重要的是这些代码能够正确地工作。然而，必须注意到那些被编程人员认为是“一次性”的代码往往不是一次性的。这些代码最后经常被复制粘贴到另一个程序中，或者成为另一个更广泛、更复杂的程序的核心。在这种情况下，被花费在设计的优雅性上的时间就非常值得了。

设计一个预计将长时间、频繁使用的软件需要时间和精力两方面的投资。比如，考虑一些类库，例如 Java 的 Swing 程序包，这些程序包被应用程序的开发人员频繁地使用。对于这样的类库，如果只能保证其正确性是远远不够的，这些类库的优秀设计同样非常重要。这些类中任何一个类的拙劣设计都将给使用这个程序包的编程人员造成很严重的后果。比如，如果一个有用的或者重要的属性被忽略，编程人员则很可能陷入一个尴尬的境地，自己编写代码，来实现原本是由类库所提供的功能。

对需要很多编程人员参与的大型系统而言，在问题的分析和解决方案的设计上花费大量的时间就更为重要了。在这种情况下，没有任何一个人可以理解程序的所有部分，事实上，每位编程人员只负责程序的一个很小的部分。如果解决方案设计不佳，任何一位编程人员在一行代码上的改动(如错误的修改或功能的增强)，都可能为其他人员所编写的代码中引入新的错误。这样的新错误所造成的代价可能十分微小，但也可能是灾难性的。

最近几年发生的几个问题，有些是灾难性的，有些则付出了巨大的金钱代价，而其他的仅仅给人们带来了些许的苦烦，正如[1-3]中所描述的。

1. 1962 年，Mariner I 号飞船在飞往金星时被人摧毁，仅因为地面计算机系统的一个错误，而让人们错误地认为飞船的助推器发生了故障。

2. 1985~1998 年间有 6 起病人被 Therac25 放射治疗系统过量辐射的事故。事故的部分起因就是该系统的控制软件中存在一个错误。

3. 1990 年，由于 AT&T 公司新安装在 114 台电子交换系统中的软件的某个错误，引起了全美范围的长达 9 小时的信号阻塞，影响了大约五百万次的电话通信。这个错误