国外经典教材·计算机科学与技术

PEARSON Addison Wesley

Data Abstraction and Problem Solving with Java: Walls and Mirrors, Updated Edition

数据结构与Java教程

(美) Frank M. Carrano
Janet J. Prichard 著
文家焱 刘伟杰 黄丽姬 译

Pearson Education

清华大学出版社

国外经典教材·计算机科学与技术

数据结构与 Java 教程

（美） Frank M. Carrano
Janet J. Prichard 著

文家焱 刘伟杰 黄丽姬 译

清华大学出版社
北京

内 容 简 介

本书详细介绍了数据间的逻辑关系、存储方式和相关运算。帮助学生逐步学会分析和解决程序设计问题。举例说明了在问题求解过程中类和抽象数据类型的作用，论述了抽象数据类型的主要用途，并在许多实例和习题中使用了递归方法。

图书在版编目（CIP）数据

数据结构与 Java 教程/（美）卡里诺（Carrano，F. M.），（美）普里查德（Prichard，J. J.）著；文家焱，刘伟杰，黄丽姬译.—北京：清华大学出版社，2004.11
（国外经典教材·计算机科学与技术）
书名原文：Data Abstraction and Problem Solving with Java:Walls and Mirrors,Updated Edition
ISBN 7-302-09732-1
Ⅰ.数… Ⅱ.①卡…②普…③文…④刘…⑤黄… Ⅲ.①数据结构－教材②JAVA 语言－程序设计－教材
Ⅳ.TP311.12
中国版本图书馆 CIP 数据核字（2004）第 105320 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：010-62770175　**客户服务**：010-62776969
文稿编辑：徐　刚
封面设计：久久度文化
印 刷 者：北京市昌平环球印刷厂
装 订 者：三河市新茂装订有限公司
发 行 者：新华书店总店北京发行所
开　　本：185×260　**印张**：32.75　**字数**：790 千字
版　　次：2004 年 11 月第 1 版　2004 年 11 月第2 次印刷
书　　号：ISBN 7-302-09732-1/TP·6729
印　　数：3001～4000
定　　价：55.00 元

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，急需一批门类齐全、具有国际先进水平的计算机经典教材，以适应当前我国计算机科学的教学需要。通过使用国外先进的经典教材，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，从而培育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，我们从国外知名的出版集团 Pearson 引进这套“国外经典教材·计算机科学与技术”教材。

作为全球最大的图书出版机构，Pearson 在高等教育领域有着不凡的表现，其下属的 Prentice Hall 和 Addison Wesley 出版社是全球计算机高等教育的龙头出版机构。清华大学出版社与 Pearson 出版集团长期保持着紧密友好的合作关系，这次引进的“国外经典教材·计算机科学与技术”教材大部分出自 Prentice Hall 和 Addison Wesley 两家出版社。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，成立了一个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 Pearson 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，全部来自于对应专业的高校教师或拥有相关经验的 IT 专家。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

国外经典教材·计算机科学与技术

编 审 委 员 会

译 者 序

Java 程序设计语言是 Sun 公司最重要的产品之一，经过多年的发展，现在已经真正成长为严谨、主流的开发语言，已经有越来越多的程序开发人员喜欢上这种简单易懂的面向对象语言，此外，Java 程序设计语言在服务器及手机中的应用也是非常成功的。由于跨平台与网络功能将是未来软件的发展趋势，而 Java 在此方面具有得天独厚的优势，因此，我们在选择语言时，不得不考虑 Java。正如 Don Bailey 所说的“我赞成作者使用 Java 来实现抽象数据类型的决定”。

本书以“墙和镜子”为主题，帮助读者澄清数据抽象（墙）与递归（镜子）这两个挑战性的概念：数据抽象将模块的实现细节同程序的其他部分相隔离，并将实现细节隐藏起来，非常像一堵墙把你同邻居隔开；递归是一种特殊的技术，通过解决类型完全相同的更小问题而解决大问题，非常像镜子中的影像随着每次反射而变得越来越小。本书通过揭示计算机科学家所用的方法和思维过程，帮助学生获得问题求解和程序设计的能力。

本书可以作为高等院校的计算机课程的教材，也可以在介绍数据结构或高级语言程序设计及其问题求解课程中结合使用本书。本书要求读者具备 Java 或其他语言的基础知识，或者有教师帮助他们通过附录 A 学习 Java。

全书的翻译出版是集体工作的结晶。文家焱、刘伟杰、黄丽姬、柳赐佳、周莎莎、施晓东、施惠琼、蔡桂凌、施琳琼、陈华、柳晁锦、柳晁惠、施卓成、张琼雯、张庭辉、方杰等负责全书的翻译工作，柳小艳、孔颂燕、梁锦伦等负责全书的审校工作，施金庭、柳聿荫、施群肖和缪彩珠等负责全书的录入和排版工作。全书最后由施平安负责统稿。

在翻译过程中，我们对本书中出现的所有术语和难词难句都进行了仔细推敲和研究，然而疏漏和争议之处在所难免，望广大读者提出宝贵的意见。

译者

前　言

本书源于 Paul Helman 和 Robert Veroff 编著的 *Intermediate Problem Solving and Data Structures: Walls and Mirrors* 一书（该书 1986 年由 Benjamin/Cummings 有限出版公司出版）。本书基于原书的组织结构和基本观点，从原著中选取一些技术、文本、插图及习题。Helman 和 Veroff 教授引入了两个非常强大的类比：墙（wall）和镜子（mirror），这使我们更容易理解和学习计算机科学。

本书重点是数据抽象和其他问题求解工具。考虑到本科课程的多样性，本书覆盖了相当广泛的内容，以使其适合其他课程。例如，可以在介绍数据结构或高级语言程序设计及其问题求解课程中使用本书。本书目标是在数据抽象、面向对象程序设计和问题求解方法方面给学生打下坚实的基础。

致　学　生

已经有成千上万的学生阅读并学习过本书。“墙”和“镜子”表示贯穿于本书的两种基本的问题求解方法。数据抽象将模块的实现细节同程序的其他部分相分离，并将实现细节隐藏起来，非常像一堵墙把你同邻居隔开。递归是一种重复的技术，通过解决类型完全相同的更小的问题而解决大问题，非常像镜子中的影像随着每次反射而变得越来越小。

本书假定读者具备某些基本的 Java 知识。一些读者可能需要重温 Java 语言，甚至从头进行学习（参考附录 A）。读者需要掌握 if 和 switch 选择结构；for、while 和 do 循环结构；类、方法和参数；数组；字符串和文件。除了附录 A 外，本书还在第 3 章和第 8 章讨论 Java 类。我们还假设读者不具备第 2 章和第 5 章中论述的递归方法的使用经验。

方　法

本书详细说明了 Java 语言的优势和不足，同时继续采用对初级水平的学生容易理解的教学方法和材料。

背景知识

本书假设读者已具备 Java 或其他语言的基础知识，并有教师帮助他们通过附录 A 过渡到 Java。除了该附录外，本书正式论述了 Java 类，包括类、继承、多态性、接口和包的基本概念。尽管本书介绍了这些内容以及抽象数据类型（ADT）的实现，但重点仍然是 ADT，而不是 Java。这些在面向对象程序设计的环境下介绍，但它假设未来的课程将详细

讨论面向对象设计和软件工程，因此重点仍是数据抽象。

适应性

本书内容丰富。读者可以选择想学的内容，并按照自己的计划学习。各章的从属关系进一步说明了章节的学习顺序。

第 I 部分中，各章可以根据学生的背景知识选择。其中有 3 章大篇幅地介绍了数据抽象和递归。数据抽象和递归都很重要，但到底应先讲哪部分却意见不一。尽管递归的章节在前，数据抽象的章节在后，但这一顺序可以调整。

第 II 部分中，各章讲解顺序也可灵活处理。例如，可在介绍栈（第 6 章）之前或之后介绍类关系（第 8 章）；也可以在学完第 5 章后引入算法效率与排序（第 9 章）；可以在队列之前介绍树，或在表之前介绍图，或在学完表格后学习散列法、平衡二叉树和优先级队列（其他顺序亦可），也可在课程的早些时候学习外部方法（第 14 章），例如，在学完第 9 章中的归并排序后学习外部排序。

数据抽象

抽象数据类型的设计及应用是始终贯穿本书的问题求解方法。我们用几个实例说明了如何设计 ADT，这是整个解决方案设计的一部分。我们首先用伪代码定义了所有 ADT，然后在论述这些 ADT 的实现之前，在简单应用中使用。ADT 与实现它的数据结构之间的差异仍然在整个讨论的最前面论述。本书一开始就解释了封装和 Java 类，并使学生明白 Java 类如何在 ADT 的客户程序中将已实现的数据结构隐藏起来。抽象数据类型包括：表、栈、队列、树、表格、堆和优先级队列，它们构成了课程的基础。

问题求解

本书强调了计算机科学家所用的方法和思维过程，帮助学生掌握问题求解和程序设计的能力。学习开发、分析和实现解决方案的方法同学习算法一样重要。

针对示例问题，提出解决方案的分析方法，在贯穿本书的解题方案设计过程中，使用了算法、数据结构以及递归逐步求精。

本书很早就介绍了 Java 引用和链表的处理，并在构建数据结构时使用了它们。本书还初步介绍了算法的数量级分析方法。这种方法先非正式地考虑，然后更量化地考虑基于数组和基于引用的数据结构的优缺点。各种方案的选择及实现是问题求解的核心课题。

最后研究程序设计风格、前置条件和后置条件的编档、辅助调试手段和循环不变量，它们都是实现和验证解决方案的重要手段。整本书都论述这些主题。

应用

本书围绕主题给出了一些典型应用。例如，折半检索、快速排序和归并排序算法是递

归的重要应用，另外还介绍了数量级分析。在平衡二叉树搜索、散列法和文件索引这样一些课题中，又继续讨论搜索问题。在外部文件一节中还将再次讨论搜索和排序问题。

识别和计算代数表达式的算法，在递归环境下首先介绍，然后在栈的应用中再次讨论。其他应用包括：作为回溯法实例的八皇后问题、作为队列应用实例的事件驱动仿真，以及作为栈和队列应用实例的图搜索和遍历。

概　述

为便于学习和方便地剪裁内容以适应实际需要，本书进行了精心规划。

组织

本书内容分成两部分：第 1 章到第 11 章通常是一学期课程的核心，第 1 章或第 2 章可作为学生的复习材料。第 11 章到第 14 章的内容取决于这门课程在整个课程体系中的地位。

第 I 部分是问题求解方法。第 1 章强调程序设计和软件工程的主要问题。下一章讨论递归，该章是为那些对递归这一重要课题不了解的学生准备的。递归地思考问题的能力是计算机科学家应具备的最有用的技能之一，对帮助人们更好地理解问题的性质通常具有重大的价值。这一章详尽地讨论递归，在第 5 章将再次深入地讨论。全书广泛使用递归，包括从简单的递归定义到语言识别、搜索、排序等算法。第 3 章详细论述数据抽象和抽象数据类型。讨论完 ADT 规范说明和用途后，本章讨论 Java 类、接口和异常，并用它们来实现 ADT。第 4 章在讨论 Java 引用变量和链表时介绍了附加实现工具。

第 II 部分是使用抽象数据类型进行问题求解。第 II 部分继续研究数据抽象的用途。这一部分首先详细说明了栈、队列、二叉树、二叉查找树、表、堆和优先级队列等的基本抽象数据类型，然后用类实现它们。在实例中使用了抽象数据类型，并对它们的实现进行了比较。第 8 章通过进一步讨论继承、包、类与类之间的关系和迭代器，扩展了 Java 类的覆盖内容。第 9 章中，通过引入数量级分析和相应表示法，形式化地描述了前面讨论过的算法的效率。本章研究了几种查找和排序算法，包括递归的归并排序和快速排序的效率。

第 II 部分还讨论一些高级课题。如平衡查找树（2-3、2-3-4、红黑和 AVL 树）和散列法，它们被当成表格的实现方式来研究的。我们对这些实现进行了分析，以确定各自支持的最优操作。最后讨论外部直接访问文件中的数据存储。修改了归并排序算法以便对数据排序，使用外部散列法和 B-树索引对其进行搜索。这些搜索算法是内部散列法和 2-3 树的推广。

补充材料

本书读者可以在线（*www.aw.com/cssupport*）获得下列补充材料。

- 源代码：读者可以使用本书中出现的所有 Java 类、方法和程序

- 勘误表：我们努力做到不犯错误，但错误是难免的。诚邀读者指出更多的问题
- 某些教师可得到补充材料。请联系当地销售代理或发邮件 aw.cse@aw.com
- 教师手册。该手册包含教学提示、示例教学大纲及所有习题的解答
- 习题库。包含大量多选题、判断题和简答题
- PowerPoint 讲义

目　录

第 I 部分　问题求解方法

第 I 部分的 5 章主要讨论问题求解方法的技巧，这奠定了本书其他章节的基础。第 1 章首先叙述了一个良好的解决方案的特点以及实现良好解决方案的方法。这些方法强调了抽象、模块化和信息隐藏。第 I 部分的其他章节讨论方案设计中的数据抽象、在实现过程中使用的 Java 类和引用以及作为问题求解策略的递归。

第 1 章　程序设计与软件工程基本原理

本章概述了几个基本原理，它们是解决大型程序复杂度的基础。这些论述强调了程序设计的基本原理，同时也说明了编写精心设计、良好编档的程序可以避免许多消耗。本章还简要地讨论算法和数据抽象问题，指出这些问题与本书主题——问题求解与程序设计技能的相关性。在后续章节，重点从程序设计的基本原理转到数据的组织与使用方法上。即使讨论重点落到这些新方法上，仍应注意解决方案如何遵循本章讨论的基本原理。

1.1　问题求解与软件工程

你最近一次写程序是从哪里开始的？读完问题描述并等待片刻后，大多数缺乏经验的程序员自然开始了编码。显然，他们的目标是使程序运行，最好能得到正确结果。因此，运行程序，检查错误信息，插入分号，修改逻辑，删去分号，祈祷……，还有其他那些折磨人的方法，直到程序工作正常。他们的大部分时间也许是花在检查语法和程序逻辑上了。当然，程序设计技能比他们第一次写程序时强多了，但使用刚才的方法，程序员真能编写出大型程序吗？也许能，但有更好的方法。

要点提示：*没有方案设计就进行编码，将增加调试时间。*

应该明确的是，特大型软件开发项目通常需要一组程序员而不仅仅是一个程序员。团队工作需要整体规划、组织和交流。毫无计划的程序设计方法不能很好地为其他程序员提供服务，而且存在严重的浪费。幸运的是，计算机科学的一个分支，即软件工程，促进了计算机程序开发方法的发展。

要点提示：*软件工程使程序开发更容易。*

尽管计算机科学的第一门课程一般强调程序设计问题，但本书的重点在于另一个更明

确的问题，即问题求解。本章首先概述问题求解过程和解决问题的各种方法。

1.1.1 什么是问题求解

此处的“问题求解”指理解问题陈述和开发解题程序的整个过程。这一过程需要经过许多阶段，从理解问题开始，通过设计概念性解决方案，一直到用计算机程序实现这个解决方案。

要点提示：解决方案定义了算法和存储数据的方法。

到底什么是解决方案呢？通常，解决方案由两部分构成：算法以及存储数据的方法。算法是在有限时间内，对解题方法的分步说明。算法经常对大量数据进行操作，例如：一个算法可能将一个新的数据加入某个集合中，或从某个集合中删除数据，或在某个数据集合中查询。

也许，对解决方案的这个描述会给人留下一种虚假的印象，在问题求解过程中，人的所有聪明才智都用来研究算法了，而如何存储数据仅仅起支撑作用。这种印象远远不是实际情况，实际上，要做的事情比单纯地存储数据多得多。在构造解决方案时，必须对数据集进行精心组织，以便按照算法要求的方式很方便地操作数据。事实上，本书主要内容论述组织数据的方法。

在设计已知问题的解决方案时，可以利用几种简化任务的方法。本章简单介绍这些方法，后续章节将更详细地讨论。

1.1.2 软件的生命期

优秀软件的开发是一个漫长、持续的过程，这个过程称为软件的生命期。该过程从最初的一种思想开始，包括程序的编写和调试，以及连续多年不断地改正与完善最初的软件。图 1.1 把软件生命期的 9 个阶段描绘成水车（这里要感谢 Raymond L. Paden，他建议我把“轮子”改成“水车”）上的扇形。这种安排表明软件生命期的各阶段是周期的一部分，而不仅仅是个线性表。尽管软件生命期从规定问题开始，但通常可以从任意一个阶段转到其他阶段，例如，测试程序时可能需要修改问题的定义或设计方案。还应注意图中围绕文档核心的 9 个阶段。文档并没有像人们期望的那样分段，而是与软件生命期的各阶段结成一体。

这是典型的软件生命期的各阶段。尽管各阶段都重要，但我们只详细讨论与本书最相关的。

第 1 阶段：规格说明。给定软件用途的最初描述后，必须清楚地规定问题的各个方面。通常，描述问题的人并不是程序员，因此最初对问题的描述可能不确切。因而，规格说明阶段要求精确化、细节化原始问题的描述，同时还要与程序员和非程序员沟通。

下面给出书写软件的规格说明时必须回答的几个问题。输入数据是什么？什么是合法数据？什么是非法数据？谁将使用软件？应该使用什么样的用户接口？需要检查什么错

误并给出什么样的错误信息？什么假设是可能的？有特殊情况吗？输出格式是什么？什么文档是必需的？程序将来很可能要增加什么功能？

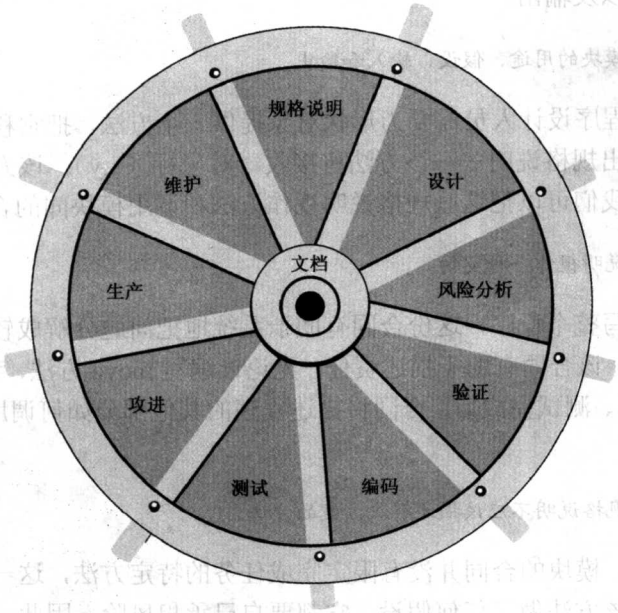

图 1.1　软件生命期比喻成能从一个阶段转到其他阶段的水车

要点提示： 要让问题的描述精确而具体。

一种增进人们之间的交流和澄清软件规格说明的方法是编写原型程序（prototype program），用它来模拟软件产品的部分行为，例如：一个简单、甚至是低效的程序，可以演示用户接口以便我们分析，此时发现困难或改变想法要比在程序设计中，甚至是程序设计完成后处理要好得多。

要点提示： 原型程序可以澄清问题。

以前的工作已经陈述了程序的规格说明。这些规格说明的某些方面也许不清楚，你不得不进一步澄清，但往往是在编写程序规格说明时缺少实际经验。

第 2 阶段：设计。一旦完成规格说明阶段，就必须设计问题的解决方案。大多数人发现，很难一次就为整个程序设计出规模和复杂度适中的解决方案。简化问题求解过程的最好方法是将大问题分解成若干小的、容易处理的部分。最终的程序将包含模块（module）。模块是自包含的代码单元。一个模块可能是一个单独的方法，或是几个方法和代码块。模块的设计应尽可能独立，即低耦合，接口部分当然除外。接口负责各模块间的通信。我们可以开发与其他模块相互独立的模块。各模块应该执行一项明确定义的任务，即应该是高内聚的。因此，模块化描述了程序模块的高内聚、低耦合的组织方式。

要点提示： 低耦合模块是相互独立的。高内聚模块执行一项明确定义的任务。

在设计阶段，明确规定各模块的功能很重要，明确规定各模块间的数据流也很重要。

例如，针对各模块应明确如下问题：在执行模块前，模块可用的数据是什么？模块做了哪些假设？已经发生了什么动作以及模块执行后数据看起来是什么样子？因此，应规定各模块的假设、输入以及输出。

要点提示： 规定各模块的用途、假设、输入和输出。

例如，如果程序设计人员需要为形状对象提供一个方法，把它移动到屏幕上的一个新地方，则可以写出规格说明——该方法将接收一对坐标（x,y）；该方法将把形状移动到屏幕上的新地方，我们可以把这些规格说明视作方法和调用模块间的合同条款。

要点提示： 把规格说明视作一种契约。

如果独立编写整个程序，这份合同有助于系统地把问题分解成较小的任务。如果程序是一个团队项目，该合同有助于描述责任，无论谁编写 move 方法，都必须履行这份合同。move 方法的编写、测试完成后，合同将描述程序的其他部分如何调用 move 方法以及相应结果。

要点提示： 模块的规格说明不应该描述解决方案的方法。

然而应注意，模块的合同并没有限定完成任务的特定方法，这一点很重要。如果程序的另外一部分对该方法做了任何假设，它都要自己承担风险。因此，以后如果有人采用不同的 move 算法来移动屏幕上的对象，应该根本不需要修改程序的其他部分。只要新方法遵守原始的合同，程序的其余部分显然不应改变。

要点提示： 方法的规格说明应包括明确的前提和结果。

以上论述对你来讲应该不是什么新内容。尽管以前你可能没有显式地使用过“合同”这个术语，但该概念应该熟悉。在书写函数的前提及结果时，实际就是在写合同，前提说明方法开始时必须存在的条件，而结果说明方法结束时的条件。例如，依据上述合同，以伪码形式书写的 move 方法如下所示：

```
move(x,y)
   // 把形状移到屏幕上的一个新位置
   // 前提：调用程序提供一对(x,y)，其中x和y都是整数
   // 结果：形状移到新位置
```

这些特定的前提和结果实际上是不完善的，很可能就是合同初稿。例如，“移动”是意味着相对于前一个位置移动，还是移动到新位置呢？x 和 y 的取值范围是多少？在实现该方法时，可能假定“移动”的意思是把形状移动到一个新位置，并且 x 和 y 的取值范围是 0 到 100。设想一下，如果另一个人试图使用 move 方法相对于前一个位置，移动(-5,-5)，则可能会遇到困难。除非在文档中修改前提与结果，否则该用户不知道相关的假设。修改后的规格说明如下：

```
move(x, y)
   // 把一个形状移动到屏幕上的横纵坐标
   // 前提：调用程序提供一对横纵坐标，横坐标和纵坐标都是整数
   // 0<=x<=MAX_XCOOR，0<=y<=MAX_YCOOR
   // MAX_XCOOR和MAX_YCOOR是类常量，规定最大坐标值
```

```
// 结果：形状移动到某个新位置
```

写前提时，首先描述方法的形参，说明方法使用的类名常量，最后列出编写方法时所做的假设。同样，在写结果时，首先描述方法的输出结果。如果是求值方法，就是方法返回的值，然后描述已经发生的其他动作（人们倾向于不加区别地使用 parameter 和 argument，但我们还是用 parameter 表示形参，而用 argument 表示实参）。

要点提示：以上是规格说明的草案初稿以及修改后的规格说明。

缺乏经验的程序员容易忽视文档准确的重要性，特别是当他们同时作为一个小程序的设计师、程序员和用户时。如果你设计了 move 方法，但没有写下契约条款，此后实现该方法时还能记着这些契约条款吗？编写完 move 方法数周后还能记住如何使用它吗？为了想起那些东西，你是愿意重新分析 Java 代码呢，还是愿意阅读一组简单的前提和结果呢？随着程序规模的增大，无论你是惟一的程序员还是程序员小组中的一员，良好文档的重要性都与日俱增。

要点提示：准确的文档是必需的。

不应忽略已实现了某些所需模块或者方法的可能性。Java 使软件组件更容易重用，这些软件组件通常按类库的形式进行组织，把类组织成包含已编译代码的软件包。也就是说，不需要总是访问方法的 Java 代码。Java 的应用程序编程接口（Application Programming Interface，API）就是这样一个集合。例如，大家都知道如何使用 Java API 包 java.lang.Math 中包含的静态方法 sqrt，而不必使用 sqrt 方法的源代码，因为它已经是预编译过的。而且，大家还知道，如果传给 sqrt 一个 double 类型的表达式，它将以 double 类型返回表达式值的平方根。即使不知道 java.lang.Math.sqrt 的实现方法，仍可以使用它。此外，java.lang.Math.sqrt 也可能不是用 Java 语言编写的。关于 java.lang.Math.sqrt 有那么多未知的东西，但只要知道它的规格说明，在程序中仍可以随意地使用它。

要点提示：设计时可以结合现有的软件组件。

如果在过去，程序设计阶段所花的时间很少或根本没有花时间，就必须改变这一习惯。设计阶段的最终结果应该是一个解决方案，而且这种解决方案非常易于转化为某种特定程序设计语言的结构。在设计阶段花费适量时间，编写和调试程序阶段将会节省更多的时间。

本章稍后将继续讨论设计。

第 3 阶段：风险分析。编写软件必然伴随着风险。一些风险对所有软件项目都是相同的，而另一些则是某个特定项目特有的。一些风险可以预测，而另一些是不可知的。风险可能影响项目的进度、费用、商业成功或者人们的健康与生活。你可以排除或减小某些风险，而另一些则不能。有方法可以侦测、评价和处理软件产品制作的风险。如果在后续课程中学习软件过程的话，将学习这些方法。风险分析的结果将影响生命期的其他阶段。

要点提示：可以预测和处理一些风险，但不是全部。

第 4 阶段：验证。理论上，可以证明算法的正确性。尽管这一领域的研究是不完备的，

但探讨一些验证过程方面的问题是有益的。

断言是算法在某一点的特定条件的声明。前提和结果只是方法在开始和结束时的条件断言。不变量是在算法特定位置始终成立的条件。循环不变量是算法中循环执行前后始终成立的条件。正如我们将看到的，循环不变量有助于写出正确的循环。使用不变量，可以在开始编码之前检查错误，进而减少调试和测试时间。总之，使用不变量可以节约时间。

证明算法的正确性与证明几何定理类似，例如,为了证明某个方法是正确的，必须从其前提，类似于几何中的公理和假设开始，根据结论来展示算法步骤。为此，要考虑算法的每一步，并说明断言间的推导关系。

要点提示：可以证明一些算法的正确性。

通过证明单个语句的正确性，可以证明语句序列、函数，直至整个程序是正确的。例如，要证明如果断言 A_1 成立且语句 S_1 执行，则断言 A_2 成立。同样，假若已经证明断言 A_2 和语句 S_2 执行，可得到断言 A_3 成立，则可以得出这样的结论：如果断言 A_1 成立，则先后执行语句 S_1 和 S_2 将得到断言 A_3。以这种方式继续下去，最终能够证明程序是正确的。

显然，如果在验证过程中发现错误，可以修改算法并且可能要修改问题的规格说明。因此，通过使用不变量，可能在编码之前使算法包含的错误更少些。结果，在调试程序上花费的时间也较少。

可以形式化地证明像条件、循环和赋值语句这样的特定结构是正确的。一种重要的方法使用循环不变量证明迭代算法的正确性。例如，我们将证明下述简单循环计算了数组 item 中前 *n* 个元素之和，在此循环开始执行前，*sum* 等于 0，*j* 等于 0。循环执行一次后，*sum* 等于 item[0]，*j* 等于 1。总的来讲，*sum* 是 item[0]到 item[j-1]这些元素的和。

```
// 计算item[0],item[1],…,item[n-1]的和, n > = 1
int  sum = 0;
int  j = 0;
while (j < n ) {
   sum += item[j] ;
   ++j ;
} // while 语句结束
```

要点提示：上述声明是此循环的不变量，请体会循环不变量的作用。

一个正确循环的不变量，在下述各个地方成立：

- 一开始时，初始化后，循环开始执行前
- 循环的每次迭代之前
- 循环的每次迭代之后
- 循环结束以后

上述的循环例子中，这些地方如下：

```
int sum = 0;
int j = 0;
              <— 不变量在这里成立
while (j < n )
              <— 不变量在这里成立
{
   sum += item[j] ;
```