国外计算机科学经典教材

Mc Graw Hill Education

Java Structures

Data Structures in Java for the Principled Programmer

Second Edition

数据结构 Java语言描述

(第2版)

(美) Duane A. Bailey 著

李 化 高树静 译

清华大学出版社

国外计算机科学经典教材

数据结构 Java 语言描述

(第 2 版)

(美) Duane A. Bailey 著

李 化 高树静 译

清华大学出版社

北 京

Duane A.Bailey
Java Structures — Data Structures in Java for the Principled Programmer Second Edition
EISBN: 0-07-112163-3

图书在版编目(CIP)数据

数据结构 Java 语言描述(第 2 版)/(美) 贝利(Bailey, D. A.)著；李化译.—北京：清华大学出版社，2004.9
书名原文：Java Structures — Data Structures in Java for the Principled Programmer Second Edition
(国外计算机经典教材)
ISBN 7-302-08983-3
I. 数… II. ①贝… ②李… III. ①数据结构 ②Java 语言—程序设计 IV. ①TP311.12 ②TP312
中国版本图书馆 CIP 数据核字(2003)第 043774 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
组稿编辑：曹 康
文稿编辑：侯 彧
封面设计：康 博
版式设计：康 博
印 刷 者：北京季蜂印刷有限公司
装 订 者：北京市密云县京文制本装订厂
发 行 者：新华书店总店北京发行所
开 本：185×260 **印张**：26.5 **字数**：678 千字
版 次：2004 年 9 月第 1 版 2004 年 9 月第1 次印刷
书 号：ISBN 7-302-08983-3/TP・6353
印 数：1～5000
定 价：48.00 元

出 版 说 明

近年来，我国高等学校的计算机学科教育进行了较大的改革，急需一批门类齐全、具有国际水平的计算机经典教材，以适应当前的教学需要。引进国外经典教材，可以了解并吸收国际先进的教学思想和教学方法，使我国的计算机学科教育能够与国际接轨，从而培育更多具有国际水准的计算机专业人才，增强我国信息产业的核心竞争力。Pearson、Thomson、McGraw-Hill、Springer、John Wiley 等出版集团都是全球最有影响的图书出版机构，它们在高等教育领域也都有着不凡的表现，为全世界的高等学校计算机教学提供了大量的优秀教材。为了满足我国高等学校计算机学科的教学需要，我社计划从这些知名的国外出版集团引进计算机学科经典教材。

为了保证引进版教材的质量，我们在全国范围内组织并成立了“清华大学计算机外版教材编审委员会”（以下简称“编委会”），旨在对引进教材进行审定、对教材翻译质量进行评审。“编委会”成员皆为全国各类重点院校教学与科研第一线的知名教授，其中许多教授为各校相关院、系的院长或系主任。“编委会”一致认为，引进版教材要能够满足国内各高校计算机教学与国际接轨的需要，要有特色风格，有创新性、先进性、示范性和一定的前瞻性，是真正的经典教材。为了保证外版教材的翻译质量，我们聘请了高校计算机相关专业教学与科研第一线的教师及相关领域的专家担纲译者，其中许多译者为海外留学回国人员。为了尽可能地保留与发扬教材原著的精华，在经过翻译和编辑加工之后，由“编委会”成员对文稿进行审定，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。

由于时间紧迫和能力所限，本套外版教材在出版过程中还可能存在一些不足和遗憾，欢迎广大师生批评指正。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，共同为我国高等学校的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

前　言

前言基本概念

- 学习本书的方法
- 原理

小时候，你的母亲可能给你买过比较有教育意义的玩具，如积木、万能工匠(Tinker Toy)[1]，或者拼装玩具等。这些玩具都是有教育意义的：它们可以培养我们的空间思维能力，并且教会我们逐渐构造复杂的结构。你可以开发出能粘在一起的模块，并形成指导建造过程的规则。

阅读本书的过程就像是玩有教育意义的玩具。你可以把编写程序当作是艺术创作的过程，只是你已经从玩积木发展成为编写程序。除了你能编写的程序的复杂性没有限制之外，用于积木搭建结构的规则同样适用于编写程序。

当编写比较大的程序时，用来维护程序中数据的数据结构决定着程序运行过程中的时间和空间复杂度。此外，较大的程序也需要时间去编写。使用不同的数据结构实际上将会影响编写程序所花费的时间。选择错误的数据结构会使程序运行效率很差，或者很难甚至是不可能有效地实现。

因此，编写程序时有一部分时间是花费在选择数据结构上的。理想情况下，你会通过分析和比较它们不同的优缺点得出结论。本书的重点集中在现代编程环境中传统数据结构的创建和分析上，这里的编程环境是指 Java 编程语言或者简称 Java。

自述

每一章都有特定的主题，许多主题都与特定的数据结构相关。我们将要研究的数据结构是从 Java 应用程序中抽象出来的，可以从 Internet[2]上获得。另外还有一些主题涉及商业应用方面。在所有主题中，有一些是数学方面的，有一些是哲学上的，但它们都很好地考虑到了编程过程。

我们所介绍的主题并不是涵盖一切的，一些有用的数据结构没有包含在本书中。因为我们学习的是编写数据结构的原理，按照本书的内容编排进行阅读，就可以自己编写出新的更好的数据结构。

本书中最重要的部分可能是每章后面的一组练习题。所有这些练习题都值得认真思考。有一些问题我会在本书的后面提供一些合理的提示或者答案。为什么应该思考这些问题呢？熟能生巧。我可以教你如何骑独轮脚踏车，但是如果你从来都不练习，那就永远也学不会。当学习并理解了这些问题之后，你将会发现自己的设计和分析能力都得到了提高。

有时候我们会在本书中提出一些问题，这些问题没有答案(有时候它们会作为正式的问题在每章后面出现，那里会有答案)，在阅读过程中要仔细思考这些问题。手边准备纸和笔有助于“思考”这些问题。

---

1 所有的商标都是经过认证的。

2 要想获得更多的信息，可以登录网站 http://www.cs.williams.edu/JavaStructures。

本书简练并切中要点。大多数人都对实验感兴趣，我们将会留出尽可能多的时间来解问题，仔细阅读代码，并练习编写程序。在通读每一章的时候，你还会发现联机阅读源代码是很有帮助的。

另外一点，与其他许多图书一样，本书所涉及到的是一项正在进行的工作，最新的思想可能不会包含在本书中。如果还有什么疑问，可以登录网站获取更新的信息。在网站上也会发现使用 javadoc 从代码生成的每个数据结构的联机文档。为了获得最新的详细信息，最好是阅读联机文档，以及其他没有正式出现在本书中的许多数据结构的文档。

原理的作用

确实，在本书中有一些策略性的评论。这些内容在刚开始可能是微不足道的，可以越过。然而，如果作为一个程序员和一个计算机科学工作者想要提高工作技巧，我建议你还是阅读一下。有时候这些评论是相当重要的，可能会成为原理：

原理 1
一个理性的程序员会很好地理解一条原理，对它形成一个看法。

自测题
这些问题的答案参见附录 A.1。

(1) 在哪里可以找到自测题的答案？

(2) 大型程序的特点有哪些？

(3) 你会阅读整本书吗？

(4) 原理就是真理吗？

本章问题
奇数问题的答案参见附录 A.2。

(1) 所有的奇数问题都有答案，你是在哪里找到这些问题的答案的(提示：参见附录 A.2)？

(2) 假设你是一个经验丰富的程序员，你会给新入门的程序员哪 5 条重要建议？

(3) 登录与本书相关的网站(http://www.cs.williams.edu/JavaStructures)，查看提供给你的资源。

(4) 本书中描述了下面哪些数据结构？(参见附录 D)：BinarySearchTree、BinaryTree、BitSet、Map、Hashtable、List？

(5) 登录网站 http://www.javasoft.com，查看 Java 的开发者 Sun 公司所提供的资源。

(6) 查看关于 Sun 公司的 java.util 包的文档(参见 http://www.javasoft.com 中的内核 API 文档)。这个包中包含了下面哪些数据结构：BinarySearchTree、BinaryTree、BitSet、Map、Hashtable、List？

(7) 查看一下附近的图书馆或者书店中关于 Java 的参考书。

(8) 编写一个 Java 应用程序，写一行代码来学习如何使用本地的 Java 编程环境(提示：阅读附录 B)。

(9) 寻找本地关于 Structure 包的文档。如果没有发现，记住在 Internet 的 http://www.cs.williams.edu/JavaStructures 上有同样的资料。

(10) 寻找与 Structure 包一起以电子形式发布的示例。这些示例中的许多将会在后述章节中讨论。

目　录

第 1 章　面向对象的方法

本章基本概念

- 数据结构
- 抽象数据类型
- 对象
- 类
- 接口

计算机科学并不像其他许多学科一样，有着悠久的历史。其他学科都有比较完备的范例和方法，而计算机科学仍然在为一个重要的问题争论不休：编写程序的最佳方法是什么？直到今天我们也没有找到最好的答案。计算机语言设计者关注的焦点是设计一种使用起来简单但又能够准确高效地描述出大型程序细节的编程语言。Java 开发正是这样一种努力的结果。

在全书中我们都把精力集中在用面向对象(object-oriented)的编程方法设计数据结构上。程序员运用这种方式，设计出称为类(class)的结构模板。这个模板可以用来构造实例(instance)或对象(object)。在面向对象程序中大部分语句所做的就是，通过向对象传送消息，让对象报告或者改变它们的状态。那么运行一个程序就是指构造和协调对象。这就是类似于 Java 这样的语言所谓的面向对象特性。

除了很小的编程项目外，对大部分项目而言，抽象(abstraction)是一种很好的编写程序的工具。在包括 Pascal、Scheme，以及 C 语言在内的许多编程语言中，程序实现的细节都隐藏在它的过程或者函数中。这种方法叫做过程抽象(procedural abstraction)。在面向对象的编程中，数据结构实现的细节都隐藏在它的对象中。这种方法涉及到了数据抽象(data abstraction)。许多现代编程语言使用面向对象来支持基本的数据抽象。在本章中，我们将回顾数据抽象的详细内容以及对象的接口(interface)设计。

1.1　数据抽象和封装

如果从马萨诸塞州的 Neville 面包店购买一个油炸圈饼，你可以不必了解它的成分，只要知道这是一个油炸圈饼就可以了。油炸圈饼是圆的，类似于面包，是甜的。油炸圈饼中的特定成分不是我们所关心的。当然 Neville 女士可以自由选用任一种甜料，只要油炸圈饼的味道不变就行。油炸圈饼的成分列表和它的构造不是我们所关心的细节问题。

同样，为了使用一个数据结构而去了解它是如何实现的，是完全没有必要的。例如，我们大多数人都熟悉字符串或者数组的运作或者语义(semantic)，然而却很难去描述它们的机制：是不是在数组中连续的所有位置，在计算机内存中的位置也是相邻的？还是离得很远？其实这并不重要，只要数组或者字符串像我们所需要的一样工作，我们就满意了。对数组或者字符串是如何实现的知道得越少，对特定的实现的依赖性也越小。抽象考虑这个问题的另外一个方法是认为数据结构遵守一个隐含的“契约”：一个字符串是字符的一个有序列表，或者一个数组的

元素可以以任何顺序访问。只要“契约”的所有条款都满足，数据结构的实现者可以自由地以任意一种合理方法去实现它。由于不同的实现者习惯于采用不同的方法，任何帮助隐藏实现细节的做法，或者任何使用抽象的方法，都会为编程提供良好的环境。

在使用正确的情况下，面向对象的编程方法允许程序员将对用户很重要的细节和仅对实现重要的细节区分开来。在本书后续的章节中，我们将会讨论数据结构的一般行为。例如，在 9.1 节中，将会学习一些数据结构，允许用户只删除最近加入的记录。这样的行为是我们对数据结构如何工作的固有的抽象认识。尽管还没有讨论这些数据结构是如何实现的，这种数据结构的独特行为方式我们还是欢迎的。那些对数据结构的用户十分重要的抽象细节，如方法的抽象语义，组成了它的契约(contract)或者接口(interface)。接口描述了数据结构的抽象行为。我们大多都会同意，尽管字符串和数组是非常相似的数据结构，但是它们的行为是不同的：我们可以缩小或者扩展一个字符串，但是却不能用同样的方法直接对数组进行操作；可以直接打印一个字符串，然而打印一个数组却意味着实际上要打印它的每一个元素。这些不同点表明了它们具有不同的抽象行为，在设计它们的接口时存在着很大的不同。

对用户隐藏的不重要的细节问题是实现的一部分。我们可以决定字符串将用一组字符以及一个附加的字符计数构成(图 1-1)。除此以外，也可以通过使用一个特殊的字符串结束标志结束字符串，以此隐式地指定字符串的长度，这个字符串结束标志是专用的。这两种方法都可以得到令人满意的效果，但是它们之间存在着折中。第一种实现方法(叫做计数字符串，counted string)显式地存储它的长度，而第二种实现方式(叫做终结字符串，terminated string)的字符串长度却是隐含的。所以就需要使用较长的时间确定终结字符串的长度，因为必须要找到字符串结束标志。而另一方面，终结字符串的大小仅仅受到可用内存空间的限制，而计数字符串的长度却取决于可以存储在它的长度域内的整数范围(通常只能有几百个字符)。如果数据结构实现者能够隐藏这些细节，用户就可以集中精力去做他们重要的设计工作。随着应用程序的不断成熟，对象都采用固定的接口，这样就允许考虑采用不同的方式实现对象。

例如 Java 这样的语言中的数据抽象允许一个数据结构确定它自己的状态。数据结构能够保持它自己的状态而不需要程序员去考虑。例如，如果必须将两个独立的字符串连接成为一个字符串，则可能必须发出一个新内存分配的请求。幸运的是，由于字符串可以自己完成对自身的操作，用户不必分心去管理内存。

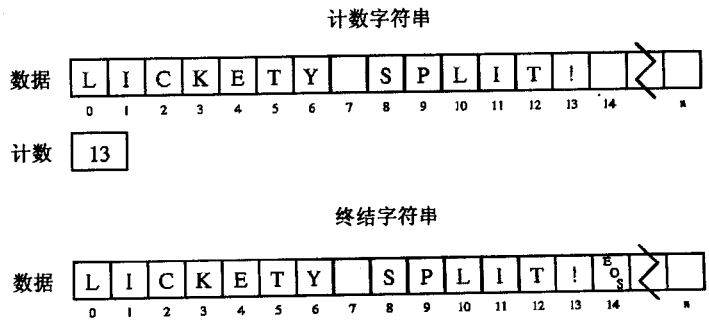

图 1-1　实现字符串的两种方法。计数字符串显式地记录了它的长度。终结字符串的长度由字符串的结束标识确定

1.2 对象模型

为了创建一个设计良好的对象，必须时刻记住设计方法。在前面曾经提到过，我们可能经常需要对程序中使用的并被对象管理的数据进行可视化。每一个对象都管理着决定它的状态的数据。例如屏幕上的一个点，有两个坐标；每张病例卡都有姓名，受益人清单，既往病史，以及一份给保险公司的证明书；遗传物质有碱基对序列。为了维护一致状态，可以想像程序只通过向对象传递消息或者方法调用(method call)来操作对象中的数据。字符串可能会收到这样一个消息："给出你的长度"，而病例卡可能会收到"更改保险费"的消息。字符串消息只简单地访问信息，而病例卡方法可能还要以一致的方式更改该病例卡内以及其他与此相关的对象中的多条信息。如果直接更改给保险公司的证明书，可能就会忘记修改与每个受益人相关的类似的证明书。对于包含复杂数据结构的大型应用程序而言，将一个复杂对象从一个状态改变为另外一个状态需要执行许多操作，想要记住协调所有这些操作是很困难的。因此我们选择由数据结构的设计者提供给我们一个方法，这个方法能够使对象在各个状态之间进行变换；当对象收到一个高层消息时，作为响应，这个方法就会被激活。

因此本书将会集中讨论两个重要的主题：(1)如何实现和评价具有逻辑上非常复杂的方法的对象，(2)如何使用所创建的对象。这些对象就代表着我们最感兴趣的数据结构(data structure)。有时候我们会开发控制结构(control structure)，开发这些结构的目的是为了控制对其他对象的操作。控制结构是一个很重要的概念，将会在第 7 章中详细讨论。

1.3 面向对象术语

在 Java 中，数据抽象是通过把对象(类的实例)中的数据进行封装(encapsulation)来实现的。类似于其他语言中的记录(record)一样，对象具有域(field)。与记录不同的是，对象还包含方法(method)。域和方法可以被声明为公共的(public)，这样做意味着它们可以从类的接口中看到；也可以声明为受保护的(protected)，这种情况下它们只能被类中的代码访问[1]。下面计算两个整数比率的简单对象是一个典型的类声明示例：

```
public class Ratio
{
   protected int numerator;   // numerator of ratio
   protected int denominator; //denominator of ratio

   public Ratio(int top, int bottom)
   // pre: bottom != 0
   // post: constructs a ratio equivalent to top::bottom
   {
      numerator = top;
      denominator = bottom;
      reduce();
   }
```

1 这种说法并不完全正确，在附录 B 中有关于实际情况的讨论。

```
public int getNumerator()
// post: return the numerator of the fraction
{
   return numerator;
}

public int getDenominator()
// post: return the denominator of the fraction
{
   return denominator;
}

public double getValue()
// post: return the double equivalent of the ratio
{
   return (double)numerator/(double)denominator;
}

public Ratio add(Ratio other)
// pre: other is nonnull
// post: return new fraction--the sum of this and other
{
   return new Ratio(this.numerator*other.denominator+
                    this.denominator*other.numerator,
                    this.denominator*other.denominator);
}

protected void reduce()
// post: numerator and denominator are set so that
// the greatest common divisor of the numerator and denominator is 1
{
   int divisor = gcd(numerator,denominator);
   numerator /= divisor;
   denominator /= divisor;
}

protected static int gcd(int a, int b)
// pre: 0 <= a,b
// post: computes the greatest integer value that divides a and b
{
   if (a ==0) {
      if (b ==0) return 1;
      else return b;
   }
   if (b < a) return gcd(b,a);
   return gcd(b%a,a);
}

public String toString()
// post: returns a string that represents this fraction
{
```