图书在版编目（CIP）数据

实用数据结构教程：Java语言描述 / 周大庆编著. —北京：人民邮电出版社，2007.5
（高等院校计算机教材系列）

ISBN 978-7-115-15907-6

Ⅰ. 实... Ⅱ. 周... Ⅲ. ①数据结构－高等学校－教材②JAVA语言－程序设计－高等学校－教材 Ⅳ. TP311.12 TP312

中国版本图书馆CIP数据核字（2007）第027312号

内 容 提 要

本书以面向对象语言Java作为描述语言，系统介绍如何用面向对象的方法来设计和实现传统的数据结构，内容包括数组、链表、栈、队列、表、二叉树、优先队列、堆、集合、映射、散列表、树和图等基本数据结构，以及插入、删除、遍历、查找、归并和排序等基本算法。本书突出了抽象数据类型的概念，提供了大量精心设计的示例程序，不仅讲述了常用数据结构的具体实现，而且抽象出一般的设计原则。

本书选材精当、结构新颖、深入浅出、简明实用，可作为高等院校计算机专业和相近专业本科生“数据结构”课程的教材或参考书，也可供计算机应用领域的工程技术人员参考。

高等院校计算机教材系列

实用数据结构教程：Java 语言描述

◆ 编　　著　周大庆
责任编辑　杨海玲

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街 14 号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京顺义振华印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：800 × 1000　1/16
印张：15.25
字数：355 千字　　2007 年 5 月第 1 版
印数：1- 4 000 册　　2007 年 5 月北京第 1 次印刷

ISBN 978-7-115-15907-6/TP

定价：28.00 元

读者服务热线：(010)88593802　印装质量热线：(010)67129223

前　言

“数据结构”是计算机专业的基础与核心课程之一，同时面向对象方法已经成为目前系统开发和程序设计的主流方式，而Java语言是目前使用最为广泛的面向对象程序设计语言之一。本书以Java为描述语言，系统地介绍如何用面向对象的方法来设计和实现传统的数据结构，内容包括数组、链表、栈、队列、表、二叉树、优先队列、堆、集合、映射、散列表、树和图等基本数据结构，以及插入、删除、遍历、查找、归并和排序等基本算法。对每一种数据结构，除了详细阐述其基本概念和可能的实现方式以外，还对每一种操作都给出Java语言的算法描述。

本书中解决问题的典型步骤是：首先对问题的需求进行分析，抽象出一个适当的数学模型，即抽象数据类型（ADT）；然后设计一个规格说明来描述ADT应支持的操作，并用Java接口来表示这个规格说明；最后选择合适的数据结构来实现ADT规格说明并分析和比较各种不同实现的代价。这种做法清楚地区分了ADT的接口与实现，体现了软件工程中分离关注点的重要原则。

全书分为12章和3个附录。第1章介绍抽象数据类型（ADT）的概念，以及怎样用Java接口表示ADT的规格说明，并且选择不同的数据结构来实现ADT规格说明。第2章介绍算法，阐述怎样分析算法的效率并介绍递归算法。

第3章和第4章讲解两个简单和普遍存在的数据结构——数组和链表，以及与它们有关的插入、删除、查找、归并和排序算法。第5章和第6章介绍栈、队列、表ADT和它们各自可供选择的实现。第7章主要介绍二叉树和二叉查找树及其相关算法的实现。第8章介绍优先队列ADT及其可供选择的实现，重点介绍了堆数据结构、用堆实现优先队列和堆排序。作为优先队列的应用，还介绍了赫夫曼编码树。第9章介绍集合与映射ADT及其各种可供选择的实现。第10章介绍散列表及其插入、删除和查找操作的实现。第11章介绍树的表示与应用。第12章介绍图ADT及其实现，以及与图有关的一些算法。

附录A给出了分析算法的效率所需要的一些数学知识。附录B总结了对本书很重要的一些Java特性，有些高级特性可能在初级程序设计课程中被省略了。附录C提供了一些课程实验项目。

本书覆盖了ACM/IEEE-CS计算学科教程CC2001（后来演变成CC2005）中关于数据结构和算法方面的大部分知识单元，主要包括：

PF2　算法与问题求解
PF3　基本数据结构
PF4　递归
AL1　基本算法分析
AL3　基本计算算法
DS5　图与树

本书可作为本科生一个学期（64学时）的教学内容。如果精简一部分内容（如第9章集合与映射、第12章中的迷宫问题等），则可以在48学时内讲完。

本书假定读者已预先修过Java初级程序设计课程，需要掌握的基本知识主要有表达式、语句、对象和类，以及接口、异常和内部类等。读者若不熟悉这些主题可以随时参考附录B。

某些数学主题（如指数、对数、级数求和与递归关系）是分析算法效率时需要的。这些数学主题大多数包含在高中数学课程中。读者遇到不熟悉的数学问题时，可以查阅附录A中的相关内容。

本书的主要特点如下：

- 把面向对象程序设计方法与数据结构和算法有机地结合了起来。
- 强调抽象数据类型的概念，说明如何用“规格说明”定义抽象数据类型的操作，并且清楚地区分抽象数据类型的规格说明和它的基于合适的数据结构与算法的实现。
- 用精心设计的Java代码表示算法，说明如何分析它们的效率，并且尽量避免涉及高深的理论和数学知识。
- 配有大量的插图，便于学生直观地理解数据结构与算法。
- 每一章都附有适量的习题（全部是算法设计题），帮助学生加深对各章内容的认识。

本书所有示例程序均在J2SE v1.4.2 SDK下编译通过并能够正确运行。本书的全部示例程序和教学用的电子教案都可以从www.turingbook.com网站上下载。

在本书编写过程中，参考了近年出版的多种国外大学数据结构教科书，其中最主要的列在“参考文献”中。

本书可作为高等院校计算机专业和相近专业的教材或参考书，也可供从事计算机应用的工程技术人员参考。

由于作者水平有限，书中难免存在错误或不足之处，欢迎读者批评指正。

周大庆

2007年1月

目　录

第1章 绪 论

本章要点

- 数据结构与数据类型。
- 抽象数据类型：ADT的规格说明，ADT的实现，Java中ADT的规格说明与实现。
- 串抽象数据类型：串ADT的规格说明，串ADT的实现。

1.1 数据结构与数据类型

数据（data）是计算机加工处理的对象，*数据结构*（data structure）是用某种方法组织起来的数据的集合。例如，数组就是一种常见的数据结构，它有时也被称为*静态数据结构*（static data structure），这是因为在创建一个数组时其大小是固定的。另外还有许多*动态数据结构*（dynamic data structure），如链表和二叉树等，它们的大小是可变的，可以在任何时候扩大或者缩小。通常，可以通过插入、删除和查找等操作来处理数据结构中的数据。

数据可以划分为不同的数据类型。*数据类型*（data type）可以描述为：

- 一个值（value）的集合。
- 一个*数据表示法*（data representation），它是该数据类型中所有的值公用的。
- 一个操作（operation）的集合，每一个操作都可以统一地应用于该数据类型中所有的值。

每一种程序设计语言都提供了一些*内置数据类型*（built-in data type），它们的值、表示法和操作是由语言本身定义的。Java提供的内置数据类型包括整型、浮点型、字符型、布尔型和字符串型等。

整型（int）是-2147483648（-2^{31}）~2147483647（$2^{31}-1$）范围内所有整数的集合。一个整数用32位2的补码表示，有加、减、乘、除和取模等操作。

字符串型（string）是所有可能的字符序列的集合。一个字符串存储在一个字符数组中，它具有字符串连接、求长度以及求指定位置的字符等操作。

程序设计语言的内置数据类型不可能满足所有应用的需求，它还必须提供引入新数据类型的手段。要引入新的数据类型，必须确定它的值、数据表示法和操作。

在Java中，引入新数据类型的手段是*类声明*（class declaration）。类的*对象*（object）是新数据类型的实例，类的*实例变量*（instance variable）确定了新数据类型的数据表示方法，类的*构造方法*（constructor）和*方法*（method）是新数据类型的操作。

类的每一个对象都包含所有实例变量的一份副本，构造方法用于创建和初始化新的对象。类的方法分为两种：查看但不改变对象的状态的方法称为*访问者*（accessor）或*获取者*（getter），改变对象的状态的方法称为*修改者*（modifier）或*设置者*（setter）。构造方法和修改者方法一起可以产生数据类型的所有的值。

例1.1 Counter数据类型。

假设我们要引入一个名为Counter（计数器）的新数据类型。计数器维护一个整数值，初始值为0，以后每调用一次inc()方法就增加1。计数器还应该提供一个getValue()方法，用于返回当前值。在这个例子中，inc()是一个修改者方法，getValue()是一个访问者方法。程序清单1-1显示了Counter类的声明。

程序清单1-1 Counter类声明

```
public class Counter {

    private int value;

// Constructor
    public Counter()
    { value = 0; }

// Modifier
    public void inc()
    { value++; }

// Accessor
    public int getValue()
    { return value; }
}
```

如果应用程序需要用到分别统计偶数个数和奇数个数的两个计数器，可以像下面这样创建两个Counter类的对象：

```
Counter even = new Counter();
Counter odd = new Counter();
```

当输入的整数k是偶数时，even计数器增加1；当输入的整数k是奇数时，odd计数器增加1。实现上述功能的语句如下：

```
if (k % 2 == 0) even.inc();
else odd.inc();
```

在设计类成员时，应掌握以下原则：

- 类的实例变量应该是私有的，这样就使得类的客户（即使用类的程序）不能直接访问这

些实例变量。

- 提供公有的方法以允许类的客户访问或修改私有实例变量，但这种访问或修改受到了程序员所写的方法实现的限制。
- 仅在类中调用的辅助方法应该是私有的。

以上原则体现了封装（encapsulation）和信息隐藏（information hiding）的思想，即一个设计良好的对象应该封装与其需要执行的任务相关的所有状态和行为，同时尽可能多地隐藏内部实现细节（见图1-1）。

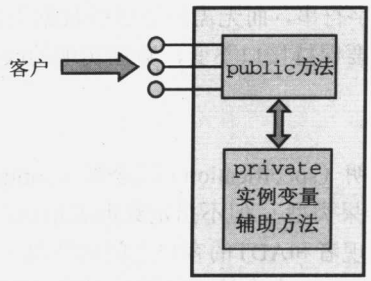

图1-1 客户与类成员的关系示意图

1.2 抽象数据类型

在上一节中提到，整型是某个范围内所有整数的集合；每个整数用32位补码表示，并且具有一些预定义的操作。但是，使用整数时，并不需要知道整数数据是如何表示的，也不需要知道整数的操作是如何实现的，只需要知道如何使用整数的操作就可以了。整数的存储方式以及具体的操作过程如何实现与整数的使用无关。

对字符串来说也是如此，我们可能不知道字符串在系统内部是如何表示的以及串操作是如何实现的，但这并不妨碍我们正确地使用字符串。

抽象（abstraction）就是要求人们关注“做什么”而不是“怎样做”的过程。在使用数据时，我们把注意力放在想要利用或者针对这些数据去做什么，而不是操心如何实现这些任务以及如何表示这些数据。例如，对两个整数进行加或乘运算时，我们并不关心它们在计算机中如何存储；同样，连接两个字符串时并不需要知道它们的内部表示。这种把数据的使用与它的实现分离开来的做法称为数据抽象（data abstraction）。

数据抽象通过抽象数据类型实现。抽象数据类型（abstract data type, ADT）可以描述为：

- 一个值的集合。
- 一个操作的集合，每一个操作都可以统一地应用于所有这些值。

它不包括数据表示法，而且这里的操作是脱离了具体实现的抽象操作（abstract operation）。换句话说，抽象数据类型是指隐藏了数据表示法并且不涉及操作的实现细节的数据类型。

抽象数据类型是一组数据及其上定义的一组操作的集合，这些操作构成了这些数据的接口，

客户正是通过接口操作数据的。而数据的实现（implement）包括它的内部表示和基于这些表示的操作的实现。抽象数据类型提供给客户的仅仅是数据的接口而屏蔽了它的实现（见图1-2）。

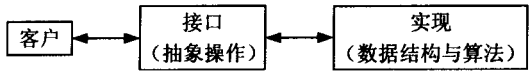

图1-2 客户、接口与实现的关系

使用抽象数据类型有两个主要优点。首先，客户不需要了解详细的实现细节就可以使用它，比如，我们可以直接使用整数和字符串，而无需知道这些数据类型的实现；其次，由于对客户屏蔽了数据类型的实现，因此，只要保持接口不变，数据实现的改变不会影响客户的使用。

1.2.1 ADT 的规格说明

每个ADT应该有一个规格说明（specification）或合约（contract），它指定ADT的每一个操作（包括操作的名字、参数类型和结果类型），但不指定数据表示法，也不指定用于实现操作的算法。

ADT的规格说明是ADT的实现者和ADT的客户之间的协议（agreement）。实现者负责提供符合ADT规格说明的实现，但不关心ADT未来是如何被使用的；而客户仅仅是用ADT规格说明中指定的操作处理ADT的值，但不关心ADT是如何实现的。

1.2.2 ADT 的实现

要实现一个ADT，必须完成两项任务：

- 确定数据表示法，它必须适用于所有可能的值，并且应该是私有的。
- 对每个可能的操作确定一个算法，这些算法必须适合于已选择的数据表示法，所有辅助操作（不在规格说明中出现的操作）应该是私有的。

1.2.3 Java 中 ADT 的规格说明与实现

在Java语言中，可以用两种方法表示ADT的规格说明。

第一种方法：ADT的规格说明用*抽象类声明*（abstract class declaration）表示，ADT的实现用继承该抽象类的*具体类声明*（concrete class declaration）表示。

第二种方法：ADT的规格说明用Java*接口*（interface）表示，ADT的实现用实现该接口的类表示。

这两种方法本质上是一样的，因为Java接口实际上是一种特殊的抽象类。本书将全部采用Java接口方法。

例1.2 `Date` ADT的规格说明与实现。

`Date` ADT描述一个日期（年、月、日），假设有如下应用需求：

- 数据值必须能表示所有过去、现在和未来的日期。
- 可以由年(y)、月(m)和日(d)构造日期。

- 可以按ISO格式“*y-m-d*”显示日期。
- 可以把当前日期向未来推进*n*天。

程序清单1-2是用Java接口表示的ADT规格说明。

程序清单1-2 用Java接口表示的Date ADT规格说明

```
public interface Date {

// Modifier
    public void advance(int n);

// Accessor
    public String toString();
}
```

Date接口中声明的两个方法分别用于把当前日期向未来推进*n*天和返回用ISO格式表示的日期字符串。

我们可以采用两种不同的方法实现Date ADT。程序清单1-3中的Date1类是第一个实现。

程序清单1-3 Date ADT的第一个实现

```
public class Date1 implements Date {
    private int year;
    private int month;
    private int day;

// Constructor
    public Date1(int y, int m, int d) {
        if (m < 1 || m > 12 ||
            d < 1 || d > length(m, y))
            throw new IllegalArgumentException("Improper date");
        year = y;
        month = m;
        day = d;
    }

// Modifier
    public void advance(int n) {
        day += n;
        int last;
        while (day > (last = length(month, year))) {
            day -= last;
            month++;
            if (month > 12) { month = 1; year++; }
        }
    }

// Accessor
    public String toString() {
        return year + "-" + month + "-" + day;
    }

// Auxiliary methods
    private int length(int m, int y) {
```

```
            switch (m) {
                case 1: case 3: case 5: case 7:
                case 8: case 10: case 12:
                    return 31;
                case 4: case 6: case 9: case 11:
                    return 30;
                case 2: return (isLeap(y) ? 29 : 28);
                default: return -1;
            }
        }

        private boolean isLeap(int y) {
            return (y%4 == 0 && (y%100 != 0 || y%400 == 0));
        }
    }
```

Date1类实现了Date接口的两个方法，该类还包括三个私有的实例变量year、month和day，以及一个构造方法。注意，这里的day表示在一个月中是第几天。此外，该类还包括两个辅助方法length()和isLeap()。

将接口与实现分离的好处之一是只要接口保持不变，客户就不会受实现改变的影响。例如，我们可以写出Date接口的另一个实现Date2，这个类的实例变量只有year和day。这里的day表示在一年中的第几天，例如，元旦是一年中的第1天，平年的最后一天是第365天。程序清单1-4是Date2的类声明。

程序清单1-4 Date ADT的第二个实现

```
public class Date2 implements Date {
    private int year;
    private int day;

// Constructor
    public Date2(int y, int m, int d) {
        if (m < 1 || m > 12 ||
            d < 1 || d > length(m, y))
          throw new IllegalArgumentException("Improper date");
        for (int i = 1; i < m; i++)
            d += length(i, y);
        year = y;
        day = d;
    }

// Modifier
    public void advance(int n) {
        day += n;
        int last;
        while (day > (last = (isLeap(year) ? 366 : 365)))
        { day -= last; year++; }
    }

// Accessor
    public String toString() {
        int m = 1, d = day;
        int last;
        while (d > (last = length(m, year)))
```

```
            { d -= last; m++; }
            return year + "-" + m + "-" + d;
        }

    // Auxiliary methods
    // The declarations of length() method and isLeap() method are the same as the Date1's
    }
```

1.3 串抽象数据类型

串（string）是字符的序列，这些字符有连续的下标（index）。子串（substring）是串中字符的子序列。串的长度（length）是其中字符的个数。空（empty）串的长度为0。

1.3.1 串 ADT 的规格说明

程序清单1-5是用Java接口表示的规格说明。接口中所有的方法都不改变当前串对象的内容，所以它们全部是访问者方法。

程序清单1-5 串ADT的规格说明

```
public interface MyString {

// Accessors
    public int length();
    // Return the length of this string.

    public char charAt(int i);
    // Return the character at index i in this string.

    public boolean equals(MyString that);
    // Return true if and only if this string is equal to that.

    public int compareTo(MyString that);
    // Return -1 if this string is lexicographically less than that,
    // or 0 if this string is equal to that,
    // or +1 if this string is lexicographically greater than that.

    public MyString substring(int i, int j);
    // Return the substring of this string consisting of
    // the characters whose indices are i, ..., j-1.

    public MyString concat(MyString that);
    // Return the string obtained by concatenating this string and
    // that.
}
```

1.3.2 串 ADT 的实现

我们可以用串的长度n加上n个字符的数组表示串，如图1-3a所示；或者用串的长度n加上n个字符的单向链表表示串，如图1-3b所示。通常用数组表示法更好一些，因为这种串是固定的，不能插入和删除字符。

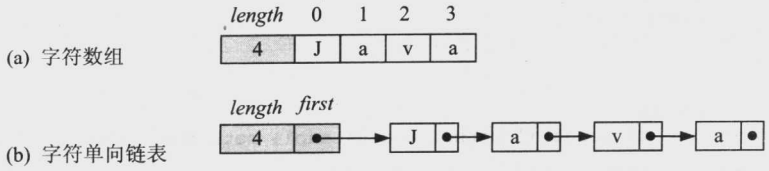

图1-3 串的不同表示法

程序清单1-6是用数组实现串的Java代码，用名为ImmutableString的类表示，它实现了MyString接口。

程序清单1-6 用数组实现串的Java代码

```
public class ImmutableString implements MyString {

    private char[] array;

// Constructors
    public ImmutableString()
    { array = new char[0]; }

    public ImmutableString(char[] that) {
        array = new char[that.length];
        for (int i = 0; i < that.length; i++)
            array[i] = that[i];
    }

// Accessors
    public int length()
    { return array.length; }

    public char charAt(int i) {
        if (i < 0 || i >= array.length)
            throw new IndexOutOfBoundsException();
        return array[i];
    }

    public boolean equals(MyString that) {
        ImmutableString other = (ImmutableString) that;
        if (array.length != other.length())
            return false;
        for (int i = 0; i < array.length; i++)
            if (array[i] != other.array[i])
                return false;
        return true;
    }

    public int compareTo(MyString that) {
        ImmutableString other = (ImmutableString) that;
        int len1 = array.length;
        int len2 = other.length();
        for (int i = 0; i < Math.min(len1, len2); i++) {
            int comp = array[i] - other.array[i];
            if (comp != 0) return comp;
        }
        if (len1 > len2) return +1;
        else if (len1 < len2) return -1;
```

```
        else return 0;
    }

    public MyString substring(int i, int j) {
        if (i < 0 || i >= array.length ||
            j < 0 || j >= array.length || i > j)
          throw new IndexOutOfBoundsException();
        int len = j - i;
        char[] newArray = new char[len];
        for (int k = 0; k < len; k++)
          newArray[k] = array[i + k];
        return new ImmutableString(newArray);
    }

    public MyString concat(MyString that) {
        ImmutableString other = (ImmutableString) that;
        int len = array.length + other.length();
        char[] newArray = new char[len];
        int i = 0, j = 0;
        while (j < array.length)
          newArray[i++] = array[j++];
        j = 0;
        while (j < other.length())
          newArray[i++] = other.array[j++];
        return new ImmutableString(newArray);
    }

    public String toString()
    { return new String(array); }
}
```

习题

1.1 编写一个名为Person的类，它包含表示人的名字和电话号码的两个数据字段。要求此类提供修改电话号码的方法，还要求有一个toString()方法，以返回由人的名字和电话号码组成的字符串。

写一个测试程序测试这个类。

1.2 编写一个名为Point的类以描述平面上的一个点，它包含点的x坐标和y坐标这两个数据字段。要求提供对其中任何一个数据字段进行设置与获取的方法，以及求两点之间距离的方法，还要求有一个toString()方法，以返回表示点的(x, y)坐标的字符串。

写一个测试程序测试这个类。

1.3 编写一个名为Rational（有理数）的类以执行分数运算，它包含numerator（分子）和denominator（分母）这两个数据字段。要求提供以下public操作：

- 两个分数相加。
- 两个分数相减。
- 两个分数相乘。
- 两个分数相除。
- 判断两个分数是否相等。

- 比较两个分数的大小。
- 以*a*/*b*的格式返回表示分数的字符串，其中*a*为分子，*b*为分母。

注意，分数的分母不能为0，并且所有的运算结果应该化为最简分数。

写一个测试程序测试这个类。

1.4 设计一个Coin ADT以描述一枚硬币。要求能表示硬币的两种状态，即正面朝上和反面朝上；随机抛掷硬币，并用一个方法来判断是否正面朝上；此外，用一个toString()方法返回表示硬币当前状态的字符串。写出用Java接口表示的ADT规格说明，并用两种不同的方法实现这个ADT。例如，可以用整数0和1表示正面和反面，或者用布尔值true和false表示正面和反面。

写一个程序来模拟简单的抛硬币游戏。要求用户猜测硬币会是正面朝上还是反面朝上，然后抛掷硬币，将结果告诉用户，并宣告用户的猜测是否正确。

1.5 设计一个Card ADT以描述一张扑克牌。要求能用指定的花色（suit）和点数（rank）生成一张扑克牌；用两个方法分别获取花色和点数；并用equals()方法判断两张牌是否相等，用compareTo()方法比较两张牌的大小；最后用一个toString()方法返回表示花色和点数的字符串。写出用Java接口表示的ADT规格说明，并用两种不同的方法实现这个ADT。例如，可以用整数0~3表示花色（梅花、红心、方块、黑桃），并且用整数0~12表示点数（2、3、……、J、Q、K、A）；或者直接用整数0~51表示全部52张牌。

编写一个测试程序测试这两种实现方法（按顺序生成52张牌，并显示结果）。

1.6 设计一个Complex ADT以描述复数。复数具有如下形式：

```
realPart + imaginaryPart * i
```

其中realPart为实部，imaginaryPart为虚部，i为-1的平方根。要求提供以下public操作：

- 取复数的实部。
- 取复数的虚部。
- 两个复数相加。
- 两个复数相减。
- 两个复数相乘。
- 两个复数相除。
- 按照格式(*a*, *b*)返回表示复数的字符串，其中*a*为实部，*b*为虚部。

写出用Java接口表示的ADT规格说明，并用两种不同的方法实现这个ADT，例如，可以用直角坐标表示复数，也可以用极坐标表示复数。无论采用哪一种实现方法，都必须提供由实部和虚部构造复数的构造方法。

编写一个测试程序测试这两种实现方法。

第2章 算 法

本章要点

- 问题、算法和程序。
- 算法的代价。
- 算法分析：规模与基本操作，运行时间和增长率，最佳、最差和平均情况。
- 大O符号：大O的定义，大O的性质，大O的计算。
- 空间代价。
- 递归算法。

2.1 问题、算法和程序

问题（problem）是一个需要完成的任务。算法（algorithm）是解决问题的步骤，它以一组值作为输入，并产生另一组值作为输出。程序（program）是算法的具体实现。计算机按照程序逐步执行算法，实现对问题的求解。

算法具有如下基本性质：

- 有限性：一个算法必须总是在执行有限步之后结束。
- 确定性：算法中的每一步都必须具有确切的含义，不会产生二义性。
- 可行性：算法中描述的操作都必须足够基本，可以通过执行有限次基本操作来实现算法。
- 输入：一个算法可以有零个或多个输入，这些输入取自某个特定的对象集合。
- 输出：一个算法可以有一个或多个输出，这些输出是与输入有着某些特定关系的量。

算法与程序有许多相同之处，但它们之间也有重大的区别。首先，算法可以由人或机器执行，而程序必须由机器执行。其次，算法可以用任何一种适当的语言或符号表示，而程序必须用程序设计语言表示。最后，算法可能是抽象的，而程序必须详细而且准确。

在用程序实现算法时，有许多程序设计语言可供选择，本书采用Java语言。当然，你也可以选择Pascal、C和C++等。