数据结构

主编　陈仲珊

上海交通大学出版社

内容提要

本书以 Java 为基础，通过丰富的实例讲解数据结构的相关知识。全书共 9 章，包括 Java 与面向对象程序设计、数据结构与算法基础、线性表、栈与队列、递归、树、图、查找和排序。本书采用 Java 论述数据结构（组织大量数据的方法）并进行算法分析（算法运行时间的估计），通过大量的实例为读者展示了如何使用数据结构实现有效的算法，并分析和测试了算法的性能，本书可作为计算机相关专业的教学用书，也可作为相关技术人员培训或工作的参考用书。

数据结构
SHUJU JIEGOU

主　　编：陈仲珊　王珊珊　罗　春
地　　址：上海市番禺路 951 号
出版发行：上海交通大学出版社
电　　话：6407 1208
邮政编码：200030
印　　制：北京荣玉印刷有限公司
经　　销：全国新华书店
开　　本：889 mm × 1194 mm　1/16
印　　张：16
字　　数：371 千字
版　　次：
印　　次：
书　　号：
定　　价：58.00 元

版权所有　侵权必究
告读者：如发现本书有印装质量问题请与印刷厂质量科联系
联系电话：010-6020 6144

前　言

随着信息技术的发展和推广，各行各业都采用计算机进行数据存储和处理，而数据结构是计算机存储、组织数据的方式，因此了解数据结构与算法显得尤为重要。对于计算机科学与技术、计算机信息管理与应用、电子商务等专业的高校学生，以及从事 IT 行业的从业者而言，要想更好、更有效地使用计算机，充分发挥计算机的性能，就必须学习和掌握数据结构的有关知识。

人们常常将 Java 与 C++ 比较，并经常把 Java 看成一种比 C++ 更安全、更具有可移植性并且更容易使用的语言。因此，这使得它成为讨论和实现基础数据结构的一种优秀的核心语言。随着计算机的速度越来越快，对于能够处理大量输入数据的程序的需求变得日益迫切。可是，由于在输入量很大的时候，低效率变得非常明显，因此这又要求对低效率问题给予更仔细的关注。通过在实际编程之前对算法的分析，可以确定某个特定的解法是否可行。希望本书可以教授读者良好的程序设计技巧和算法分析能力，使读者能够以更高的效率开发出程序。

本书采用 Java 论述数据结构（组织大量数据的方法）并进行算法分析（算法运行时间的估计），通过大量的实例为读者展示了如何使用数据结构实现有效的算法，并分析和测试了算法的性能。本书分 9 章为读者讲解数据结构的相关知识，以及用 Java 实现各种算法的方法，具体内容包括 Java 与面向对象程序设计、数据结构与算法基础、线性表、栈与队列、递归、树、图、查找和排序。

本书在编写上具有以下特色：

（1）通过章前的“学习目标”“知识导图”明确本章要学习的内容，帮助读者做好学习准备；通过文中的“算法说明”等，丰富知识内容，提高读者的学习兴趣。

（2）在精练的语言的基础上，充分利用丰富的实例讲解数据结构与算法知识，内容由浅入深、循序渐进，符合初学者的认知规律。

（3）计算机技术发展很快，本书着重于当前的最新知识和主流技术的讲解，使读者学到的知识和技术都与行业密切联系，做到学有所用。

此外，编者还为广大一线教师提供了服务于本书的教学资源库，有需要者可致电 13810412048，或发邮件至 2393867076@qq.com 领取。

为与代码格式保持一致，本书科技符号均用正体表示。

本书可作为计算机相关专业的教学用书，也可作为相关技术人员培训或工作的参考用书。由于编写时间仓促，网络技术发展迅猛，书中存在的不足和疏漏之处，敬请广大读者批评指正，在此表示衷心的感谢。

编　者
2022 年 5 月

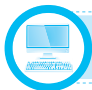
目　录

第 1 章　Java 与面向对象程序设计 / 1

1.1　Java 语言基础知识……2
1.1.1 基本数据类型及运算……2
1.1.2 流程控制语句……3
1.1.3 字符串……5
1.1.4 数组……6
1.2　Java 的面向对象特性……8
1.2.1 类与对象……8
1.2.2 继承……10
1.2.3 接口……12
1.3　异常……14
1.4　Java 与指针……15

第 2 章　数据结构与算法基础 / 17

2.1　数据结构……18
2.1.1 基本概念……18
2.1.2 抽象数据类型……20
2.2　算法及性能分析……23
2.2.1 算法……23
2.2.2 时间复杂性……23
2.2.3 空间复杂性……27
2.2.4 算法时间复杂度分析……27
2.2.5 最佳、最坏与平均情况分析……30
2.2.6 均摊分析……32



第 3 章　线性表 / 35

3.1　线性表及抽象数据类型……36
3.1.1 线性表定义……36
3.1.2 线性表的抽象数据类型……36
3.1.3 List 接口……38
3.1.4 Strategy 接口……40
3.2　线性表的顺序存储与实现……41
3.3　线性表的链式存储与实现……47
3.3.1 单链表……47
3.3.2 双向链表……51
3.3.3 线性表的单链表实现……53
3.4　两种实现的对比……58
3.4.1 基于时间的比较……58
3.4.2 基于空间的比较……59
3.5　链接表……59
3.5.1 基于节点的操作……59
3.5.2 链接表接口……60
3.5.3 基于双向链表实现的链接表……61
3.6　迭代器……65

第 4 章　栈与队列 / 69

4.1　**栈**……70
4.1.1 栈的定义及抽象数据类型……70
4.1.2 栈的顺序存储实现……72
4.1.3 栈的链式存储实现……73
4.2　**队列**……75
4.2.1 队列的定义及抽象数据类型……75
4.2.2 队列的顺序存储实现……76
4.2.3 队列的链式存储实现……80
4.3　**堆栈的应用**……81
4.3.1 进制转换……81
4.3.2 括号匹配检测……82
4.3.3 迷宫求解……84

第 5 章　递归 / 89

5.1　**递归与堆栈**……90
5.1.1 递归的概念……90
5.1.2 递归的实现与堆栈……92
5.2　**基于归纳的递归**……93
5.3　**递推关系求解**……95
5.3.1 求解递推关系的常用方法……95
5.3.2 线性齐次递推式的求解……98
5.3.3 非齐次递推关系的解……98
5.3.4 Master Method……99
5.4　**分治法**……101
5.4.1 分治法的基本思想……101
5.4.2 矩阵乘法……104
5.4.3 选择问题……106

第 6 章　树 / 109

6.1　**树的定义及基本术语**……110
6.2　**二叉树**……113
6.2.1 二叉树的定义……113
6.2.2 二叉树的性质……114
6.2.3 二叉树的存储结构……116
6.3　**二叉树基本操作的实现**……121
6.4　**树、森林**……128
6.4.1 树的存储结构……128
6.4.2 树、森林与二叉树的相互转换……131
6.4.3 树与森林的遍历……133
6.4.4 由遍历序列还原树结构……134
6.5　**Huffman 树**……135
6.5.1 二叉编码树……135
6.5.2 Huffman 树及 Huffman 编码……136

第 7 章　图 / 141

7.1　**图的定义**……142
7.1.1 图及基本术语……142
7.1.2 抽象数据类型……146
7.2　**图的存储方法**……149
7.2.1 邻接矩阵……149
7.2.2 邻接表……150
7.2.3 双链式存储结构……151
7.3　**图 ADT 实现设计**……159

7.4 图的遍历 …… 161
7.4.1 深度优先搜索 …… 161
7.4.2 广度优先搜索 …… 164
7.5 图的连通性 …… 166
7.5.1 无向图的连通分量和生成树 …… 166
7.5.2 有向图的强连通分量 …… 167
7.5.3 最小生成树 …… 168
7.6 最短路径 …… 174
7.6.1 单源最短路径 …… 174
7.6.2 任意顶点间的最短路径 …… 180
7.7 有向无环图及其应用 …… 181
7.7.1 拓扑排序 …… 181
7.7.2 关键路径 …… 184

第 8 章　查找 / 189

8.1 查找的定义 …… 190
8.1.1 基本概念 …… 190
8.1.2 查找表接口定义 …… 191
8.2 顺序查找与折半查找 …… 191
8.3 查找树 …… 195
8.3.1 二叉查找树 …… 195
8.3.2 AVL 树 …… 203
8.3.3 B - 树 …… 213
8.4 哈希 …… 218
8.4.1 哈希表 …… 219
8.4.2 哈希函数 …… 220
8.4.3 解决冲突 …… 221

第 9 章　排序 / 225

9.1 排序的基本概念 …… 226
9.2 插入排序 …… 227
9.2.1 直接插入排序 …… 227
9.2.2 折半插入排序 …… 229
9.2.3 希尔排序 …… 229
9.3 交换排序 …… 231
9.3.1 起泡排序 …… 231
9.3.2 快速排序 …… 233
9.4 选择排序 …… 235
9.4.1 简单选择排序 …… 235
9.4.2 树型选择排序 …… 237
9.4.3 堆排序 …… 238
9.5 归并排序 …… 242
9.6 基于比较的排序方法的对比 …… 244
9.7 在线性时间内排序 …… 246
9.7.1 计数排序 …… 246
9.7.2 基数排序 …… 247

参考文献 …… 248

第 1 章 Java 与面向对象程序设计

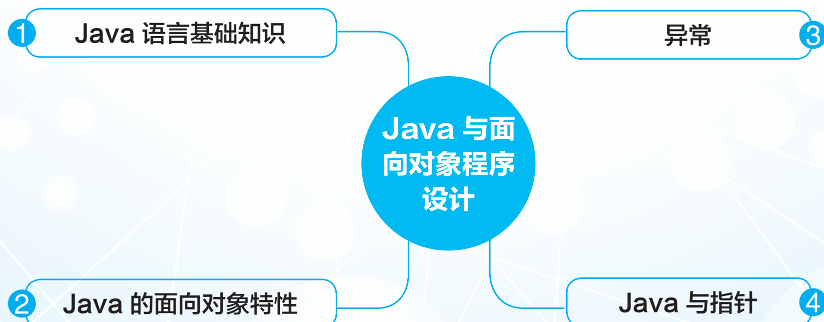
学习目标

（1）掌握 Java 语言基础知识。

（2）了解 Java 面向对象的特性，并能熟练使用。

（3）掌握 Java 中的异常和指针。

知识导图



笔记

本章导读

在这一章中向读者简要介绍有关 Java 的基础知识。Java 语言是一种广泛使用并且具有许多良好的如面向对象、可移植性、健壮性等特性的计算机高级程序设计语言，在这里对 Java 的介绍不可能面面俱到，因此在第 1 章中只对 Java 代码的相关知识进行介绍。熟悉 Java 的读者可以不阅读本章。

1.1 Java 语言基础知识

1.1.1 基本数据类型及运算

在 Java 中每个变量在使用前均必须声明它的类型。Java 共有八种基本数据类型：四种整型，两种浮点型，一种字符型，以及用于表示真假的布尔类型。各种数据类型的细节如表 1-1 所示。

表 1-1　Java 数据类型

类型	存储空间 /bit	范围
int	32	[−2147483648,2147483647]
short	16	[−32768,32767]
long	64	[−9223372036854775808, 9223372036854775807]
byte	8	[−128,127]
float	32	[−3.4E38,3.4E38]
double	64	[−1.7E308,1.7E308]
char	16	Unicode 字符
boolean	1	True,False

声明一个变量[①]时，应先给出此变量的类型，随后写上变量名。在声明变量时，一行中可以有多个变量，并且可以在声明变量的同时对变量进行初始化。例如：

```
int i;
double x, y = 1.2;
char c = 'z';
boolean flag;
```

在程序设计中，常常需要在不同的数字数据类型之间进行转换。图 1-1 给出了数字类型间的合法转换。

①为和代码中的正斜体格式一致，本书中的科技符号统一使用正体表示。

图 1-1 中 6 个实箭头表示无信息损失的转换，而 3 个虚箭头表示的转换则可能会丢失精度。

有时在程序设计中也需要进行在图 1-1 中没有出现的转换，在 Java 中这种数字转换是可以进行的，不过信息可能会丢失。在可能丢失信息的情况下进行的转换是通过强制类型转换来完成的。其语法是在需要进行强制类型转换的表达式前使用圆括号，圆括号中是需要转换的目标类型。例如：

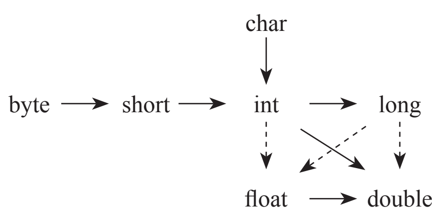


图 1-1　数字类型间的合法转换

```
double x = 7.8;
int n = (int)x;        //x 等于 7
```

Java 使用常见的算术运算符 +、–、*、/ 进行加、减、乘、除的运算。当除法运算符 / 用于两个整数时，是进行整数除法。整数的模（求余）运算使用 % 运算符。对整型变量一种最常见的操作就是递增与递减运算，与 C/C++ 一样，Java 也支持递增和递减运算。例如：

```
int n = 7, m = 2;
double d = 7;
n = n / m;             //n 等于 3
d /= m;                //d 等于 3.5
n--;                   //n 等于 2
int a = 2 * n++;       //a 等于 4
int b = 2 * ++m;       //b 等于 6
```

此外，Java 还具有完备的关系运算符，如 = =（是否相等），<（小于），>（大于），< =（小于等于），> =（大于等于），!=（不等于）；并且 Java 使用 && 表示逻辑与，|| 表示逻辑或，! 表示逻辑非；以及 7 种位运算符 &（与）、|（或）、^（异或）、~（非）、>>（右移）、<<（左移）、>>>（高位填充 0 的右移）。

最后 Java 还支持一种三元运算符 ?: ，这个运算符有时很有用。它的形式为

```
condition ? e1 : e2
```

这是一个表达式，在 condition 为 true 时返回值为 e1，否则为 e2。例如：

```
min = x < y ? x : y;
```

则 min 为 x 与 y 中的较小值。

1.1.2 流程控制语句

计算机高级语言程序设计中共有 3 种流程结构，分别是顺序、分支、循环。其中，分

支与循环流程结构需要使用固定语法的流程控制语句来完成。

Java 中有两种语句可用于分支结构，一种是 if 条件语句，另一种是 switch 多选择语句。条件语句的形式如下：

```
if (condition) statement1 else statement2
```

当 if 后的条件 condition 的值为 true 时执行 statement1 中的语句，否则执行 statement2 中的语句。

多选择语句的形式为

```
switch (integer expression){
    case value1: block1;break;
    case value2: block2; break;
    …
    case valueN: blockN;break;
    default: default block;
}
```

switch 语句从与选择值相匹配的 case 标签处开始执行，一直执行到下一个 break 处或者 switch 的末尾。如果没有相匹配的 case 标签，而且存在 default 子句，那么执行 default 子句。如果没有相匹配的 case 标签，并且没有 default 子句，则结束 switch 语句的执行，执行 switch 后面的语句。

Java 中的循环语句主要有 3 种，分别是 for 循环、while 循环、do…while 循环。

for 循环的形式为

```
for (initialization; condition; increment) statement;
```

for 语句循环控制的第一部分通常是对循环变量的初始化，第二部分给出进行循环的测试条件，第三部分则是对循环变量的更新。

while 循环的形式为

```
while (condition) statement;
```

while 循环首先对循环条件进行测试，只有在循环条件满足的情况下才执行循环体。

do…while 循环的形式为

```
do statement while (condition);
```

与 while 循环不同的是，do…while 循环首先执行一次循环体，当循环条件满足时再继续进行下一次循环。

1.1.3 字符串

字符串是指一个字符序列。在 Java 中没有内置的字符串类型，而是在标准 Java 库中包含一个名为 String 的预定义类。每个被一对双引号括起来的字符序列均是 String 类的一个实例。字符串可以使用如下方式定义：

```
String s1 = null;      //s1 指向 null
String s2 = "";        //s2 是一个不包含字符的空字符串
String s3 = "Hello";
```

Java 允许使用符号“+”把两个字符串连接在一起。当连接一个字符串和一个非字符串时，后者首先被转换成字符串，然后进行连接。例如：

```
s3 = s3 + "World!";          //s3 为 "HelloWorld!"
String s4 = "abc" + 123;     //s4 为 "abc123"
```

Java 的 String 类包含许多方法，其中多数非常有用，表 1-2 给出了常用的一些方法。

表 1-2　JAVA String 类常用方法及说明

返回值类型	方法及说明
char	charAt(int index) Returns the char value at the specified index
int	compareTo(String anotherString) Compares two strings lexicographically
int	compareToIgnoreCase(String str) Compares two strings lexicographically, ignoring case differences
boolean	endsWith(String suffix) Tests if this string ends with the specified suffix
boolean	equals(Object anObject) Compares this string to the specified object
boolean	equalsIgnoreCase(String anotherString) Compares this String to another String, ignoring case considerations
int	indexOf(String str) Returns the index within this string of the first occurrence of the specified substring
int	lastIndexOf(String str) Returns the index within this string of the right most occurrence of the specified substring
int	length() Returns the length of this string
boolean	startsWith(String prefix) Tests if this string starts with the specified prefix
String	substring(int beginIndex) Returns a new string that is a substring of this string

笔记

续表

返回值类型	方法及说明
String	substring(int beginIndex, int endIndex) Returns a new string that is a substring of this string
char[]	toCharArray() Converts this string to a new character array
String	toLowerCase() Converts all of the characters in this String to lower case using the rules of the default locale
String	toString() This object (which is already a string!) is itself returned
String	toUpperCase() Converts all of the characters in this String to upper case using the rules of the default locale
String	trim() Returns a copy of the string, with leading and trailing white space omitted

如果读者需要进一步了解有关 String 提供的其他方法及方法完成的功能，可以通过在线 API（应用程序接口）文档了解相关信息，从中可以查到标准库中所有的类及方法。API 文档是 Java SDK 的一部分，以 HTML 格式显示。 JDK1.5.0 的 API 文档地址为 http://java.sun.com/j2se/1.5.0/docs/api/index.html。

1.1.4 数组

数组是用来存放一组具有相同类型数据的数据结构。可以通过整型下标来访问数组中的每一个值。数组可以通过在某种数据类型后面加上 [ ] 来定义，在此之后跟上变量名就可以定义相应类型的数组变量了。例如：

```
int[] a;
```

还可以使用另一种方法定义数组，例如：

```
int a[];
```

以上这两种方法的定义是等价的。在这里只定义了一个整型数组变量 a，但是还没有将 a 真正地初始化为一个数组。为将一个数组初始化可以使用 new 关键字，也可以使用赋值语句进行初始化。数组一旦被创建，它的大小就不能改变。

例如：

```
// 将 a 初始化为大小为 10 的整型数组
a = new int[10];
// 将 b 初始化为大小为 4 的整型数组，并且 4 个分量的值分别等于 0，1，2，3
int[] b = {0,1,2,3};
```

数组的下标从 0 开始计数，到数组大小减 1 结束。在 Java 中不能越过数组下标的范

围去访问数组中的数据。例如：

```
a[10] = 10;
```

如果越过数组的下标访问数据，则会产生一个名为 ArrayIndexOutOfBoundsException 的运行时错误。这种错误可以通过在访问某个下标的数组元素前检查数组的大小来避免。数组的大小可以通过数组的变量 length 返回。例如：

```
for (int i=0;i<a.length;i++)
    a[i] = i;
```

由于在 Java 中数组实际上是一个类，因此两个数组变量可以指向同一个数组。请读者预测以下这段代码的运行结果。

```
int[] a = {1,1,1};
int[] b = a;
for (int i=0;i<b.length;i++)
    b[i]++;
for (int i=0;i<a.length;i++)
    System.out.print(a[i]);
```

在这段代码中对数组 b 的每个分量加 1，但是在输出数组 a 的每个分量时，可以发现 a 的每个分量都发生了变化，都为 2。其原因是赋值语句 int[] b = a; 只是将对于数组 a 的引用传递给变量 b，此时数组变量 a、b 实际上指向同一个数组空间。图 1-2 说明了这段代码运行时的情况。

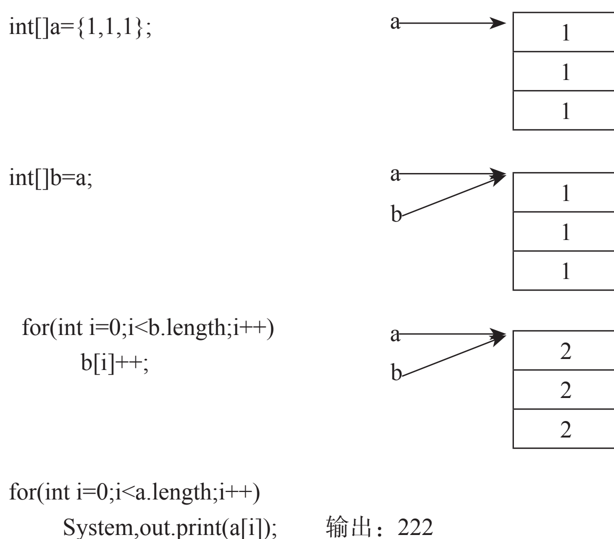


图 1-2　两个数组变量指向同一个数组空间

若要得到两个不同的但每个分量的值均相等的数组，可以使用循环语句或 System 类中的 arraycopy 方法来完成。

同样当数组作为方法的参数传递时，也是传递的对于数组的引用，因此在方法中对数组进行的操作会影响到原来的数组。例如：

```
public void changeArray(int[] a)
{
    for (int i=0;i<a.length;i++)
            a[i] = a[i] * 2;
}
```

那么如下代码的运行结果为 444。

```
int c = {2,2,2};
changeArray( c );
for (int i=0;i<c.length;i++)
    System.out.print(c[i]);
```

1.2 Java 的面向对象特性

面向对象程序设计（object-oriented programming，OOP）是目前主流的程序设计方法，它已经取代了以前基于过程的程序设计技术。面向对象程序设计主要是指在程序设计中采用抽象、封装、继承等设计方法。

1.2.1 类与对象

在面向对象思想中，抽象决定了对象的对外形象、内部结构以及处理对象的外部接口，其关键是处理对象的可见外部特征。抽象主要是从现实世界中抽象出合理的对象结构。

封装性是保证软件部件具有优良的模块性的基础。在 Java 中，最基本的封装单元是类，一个类定义了将由一组对象所共享的行为（数据和代码）。一个类的每个对象均包含它所定义的结构与行为，这些对象就像是一个模子铸造出来的。所以对象也叫作类的实例。

1. 类的定义

在定义一个类时，需要指定构成该类的代码与数据，类所定义的对象叫作成员变量。操作数据的代码叫作成员方法，方法定义怎样使用成员变量。这意味着类的行为和接口要由操作数据的方法来定义。

由于类的用途是封装复杂性，所以类的内部有隐藏实现复杂性的机制。所以 Java 中提供了私有和公有的访问模式，类的公有接口代表外部的用户应该知道或可以知道的每件东西。私有的方法和数据只能通过该类的成员代码来访问。

在 Java 中类的定义是通过关键字 class 来实现的。例如：

```
public class People{
    private String name;
    private String id;
    //Constructor
    public People(){
        this("","");
}

public People(String name,String _id){
        this.name = name;
        id = _id;
}

public void sayHello(){
        System.out.println("Hello! My name is " + name);
}
public void sayHello(String name){
        System.out.println("Hello,"+name+ "! My name is " +this.name);
}

//get & set methods
public void setName(String name){
        this.name = name;
}
public void setId(String id){
        this.id = id;
}
public String getName(){
        return this.name;
}
public String getId(){
        return this.id;
    }
}
```

代码中使用 class 关键字定义了一个名为 People 的类，class 前面的 public 关键字表示这个类似的一个公有类，可被访问。People 类中使用了 this 关键字，this 关键字主要有两个作用，一是表示对隐式参数的引用，二是调用类中的其他构造方法。

在类的内部首先通过 private 关键字定义了两个私有的成员变量，由于这两个成员变量是私有的，因此为了能够在类的外部获取或修改这些信息，定义了四个 get、set 方法。并且 People 类定义了两个构造方法，构造方法是一种特殊的方法，其作用是构造并初始化对象，在一个类中构造方法可以定义多个。要构造一个新的对象只需要在构造方法前使用 new 关键字就可以了。例如：

```
People jack = new People("Jack","0001");
```

此外在类中还定义了两个 sayHello 方法，以便实现与外界的互操作。

2. 使用现有类

Java 提供的大量的预定义类可供使用，同时程序中可能还会使用第三方提供的或者自己编写的属于其他包的类，使用这些类会给编写程序带来巨大的便利。

如果在某个类中需要使用其他包中的类，可以使用 import 关键字将需要使用的类在类的定义开始之前引入。

例如，在 People 类中还可以定义一个新的成员变量 birthday，而日期可以使用 Java 提供的预定义类 Calendar 实现。以下代码给出了实现方法。

```
import java.util.Calendar;
public class People {
    private String name;
    private String id;
    private Calendar birthday;
    // 构造方法
    ……
    //sayHello 方法、get & set 方法
    ……
}//end of class
```

1.2.2 继承

继承是子类自动获取父类的数据和方法的机制，这是类之间的一种关系。在定义和实现一个类的时候，可以在一个已经存在的类的基础之上来进行，把这个已经存在的类所定义的内容作为自己的内容，并加入若干新的内容。

例如，学生也是人，那么他（她）也有 name、id、birthday 等属性，也可以和外界进行 sayHello 方法定义的互操作，但是学生是一群特定的人群，他们还具有一些特定的属性以及特定的和外界进行互操作的方法。在这种情况下就需要使用继承，可以定义一个新的类 Student，然后向它添加功能。但是新的类可以重用 People 类中已有的成员变量和方法。抽象地说，Student 类和 People 类是一个明显的“is-a”关系：每个学生都是人。“is-a”关系就是继承的特点。

在 Java 中使用 extends 关键字来实现继承。例如下面的代码定义了一个新的类 Student，它继承了最初定义的 People 类。

```
public class Student extends People{
private String sId; // 学号
//Constructor
public Student() {
    this("","","");
}
public Student(String name,String id,String sId){
    super(name,id);
    this.sId = sId;
}

public void sayHello(){
    super.sayHello();
    System.out.println("I am a student of department of computer science.");
}
//get & set method
public String getSId(){
    return this.sId;
}
public void setSId(String sId){
    this.sId = sId;
}
}
```

代码中使用了 super 关键字，super 关键字主要有两个作用，一是调用父类的构造方法，二是调用父类的方法。

extends 关键字表明使用它构造出来的类是从一个现有的类衍生出来的。现有类称为父类，而新的类称为子类。父类与子类相比并不具有更多的属性和功能，子类比父类具有更多的属性和功能。“is-a”规则表明子类的每个对象都是父类的对象，例如每个学生都是人。因此，无论何时只要在程序中需要一个父类对象时都可以使用一个子类的对象来替代它，反过来则不行。

例如，可以把子类的对象赋给父类变量：

```
People p = new Student("Bob","0002","2006137129");
```

如果想要把对某个类的对象引用转换为对另一个类的对象引用，需要用圆括号把目标

类名括起来，然后放到需要转换的对象引用之前。例如：

```
Student s = (Student)p;
```

当然这种转换并不是一定能够完成，如果是不能完成的情况，程序在运行时会抛出异常。为了使转换在允许的情况进行，可以使用 instanceof 关键字。例如：

```
if ( p instanceof Student)
    Student s = (Student)p;
```

在 Java 中有一个非常特殊的预定义类，那就是 Object 类。在 Java 中 Object 类是所有类的祖先，每个类都由 Object 类扩展而来。在定义类时如果不指定父类，则 Java 会自动把 Object 类作为要定义类的父类。例如 People 类就是 Object 类的子类。因此可以使用 Object 类的变量引用任意类型的对象。例如：

```
Object obj = new People("Jack","0001");
```

在 Java 中不支持多继承。Java 对于多继承大部分功能的实现是通过接口机制来完成的。

1.2.3 接口

接口是 Java 实现多继承的一种机制，一个类可以实现一个或多个接口。接口是一系列方法的声明，是一些方法特征的集合，一个接口只有方法的特征没有方法的实现，因此这些方法可以在不同的地方被不同的类实现，而这些实现可以具有不同的行为。简单地说，接口不是类，但是定义了一组对类的要求，实现接口的某些类要与接口一致。

在 Java 中使用 interface 关键字来定义接口。例如：

```
public interface Compare {
    public int compare(Object otherObj);
}
```

Compare 接口定义了一种操作 compare，该操作应当完成与另一个对象进行比较的功能。它假定某个实现这一接口的类的对象 x 在调用该方法时，例如 x.compare(y)，如果 x 小于 y，则返回负数，相等返回 0，否则返回正数。

让类实现一个接口需要使用 implements 关键字，然后在类中实现接口所定义的方法。例如：

```
public class Student extends People implements Compare{
private String sId; // 学号
//Constructor
public Student() {
```

```
        this("","","");
    }
    public Student(String name,String id,String sId){
        super(name,id);
        this.sId = sId;
    }

    public void sayHello(){
        super.sayHello();
        System.out.println("I am a student of department of computer science.");
    }
    //get & set method
    public String getSId(){
        return this.sId;
    }
    public void setSId(String sId){
        this.sId = sId;
    }

    //implements Compare interface
    public int compare(Object otherObj){
        Student other = (Student)otherObj;
        return this.sId.compareTo(other.sId);
    }
    }//end of class
```

代码中 Student 类实现了 Compare 接口，并且实现了 compare 方法，这里假定通过两个学生学号的字典顺序完成对两个学生的比较，学号字典顺序在前的学生小于学号字典顺序在后的学生。

需要注意的是在 Java 中接口不是类，因此不能使用 new 实例化接口。但是虽然不能通过 new 构造接口对象，但是可以声明接口变量。并且只要类实现了接口，就可以在任何需要该接口的地方使用这个接口的对象。例如：

```
Compare com = new Student("Cary","0003","2006137101");
```

反之，也可以将一个接口变量转换为对某个类的对象的引用，不过此时要进行强制转换，并且不一定能够完成，情况和从父类引用到子类引用的转换是一样的。例如：

```
Student s = (Student)com;
```

笔记

1.3 异常

对于程序运行时碰到的异常情况，Java 使用了一种被称为“异常处理”的机制来进行处理。在 Java 中一个异常对象总是 Throwable 子类的实例。图 1-3 所示为 Java 异常继承层次结构。

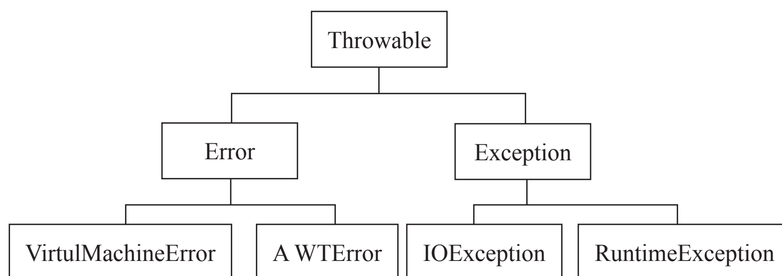


图 1-3 Java 异常继承层次结构

在 Java 程序设计中，常常关注 Exception 这个分支体系，而 Exception 中一类是从 RuntimeException 衍生出来的子类，以及不是从它衍生出来的其他异常类。一般来说，由编程导致的错误会引起 RuntimeException，例如数组下标越界、错误的类型转换、访问空指针等错误会导致不同类型的 RuntimeException。

在程序中可能会碰到任何标准异常都不能很好描述的异常情况。此时，可创建自己的异常类，创建自己的异常类只需要继承 Exception 类或 Exception 的子类就可以了。

在程序中如果碰到了异常的情况，可以有两种方法来处理这个异常，一种是由方法本身捕获这个异常并进行相应的处理，使用 try…catch 结构；另一种是将这个异常从方法中抛出，使用 throws 及 throw 关键字。例如：

```
public void method1(){
    try{
            //statement may cause exception
            ……
    }catch(ExceptionType e){
            //deal with exception
            ……
    }
}
```

再如：

```
Public void method2() throws ExceptionType {
    ……
    if (exception condition) throw instance of ExceptionType;
    ……
}
```

1.4 Java 与指针

尽管在 Java 中没有显示使用指针并且也不允许程序员使用指针，然而实际上对象的访问就是使用指针来实现的。一个对象会从实际存储空间的某个位置开始占据一定数量的存储体。该对象的指针就是一个保存了对象的存储地址的变量，并且这个存储地址就是对象在存储空间中的起始地址。在许多高级语言中指针是一种数据类型，而在 Java 中是用对象的引用来替代的。

考虑前面已经定义的 People 类，以及下列语句：

```
People p = null;
q = new People("Jack","0001");
```

这里创建了两个对于对象引用的变量 p 和 q 。变量 p 初始化为 null，null 是一个空指针，它不指向任何地方，也就是说它不指向任何类的对象，因此 null 可以赋值给任何类的对象的引用。变量 q 是一个对于 People 类的实例的引用，操作符 new 的作用实际上是为对象开辟足够的内存空间，而引用 p 是指向这一内存空间地址的指针。

为此请读者考虑如下代码的运行结果。

```
People p1 = new People("David","0004");
People p2 = p1;
p2.setName("Denny");
System.out.println(p1.getName());
```

这段代码中对 People 类的对象引用 p2 的 name 成员变量进行了设置，使其值为字符串“ Denny”。但是很容易会发现在输出 p1 的成员变量 name 时并不是输出“ David”，而是“Denny”。原因是 p1 与 p2 均是对对象的引用，在完成赋值语句“People p2 = p1;”后，p2 与 p1 指向同一存储空间，因此对于 p2 的修改自然会影响到 p1。图 1-4 清楚地说明了这段代码运行的情况。

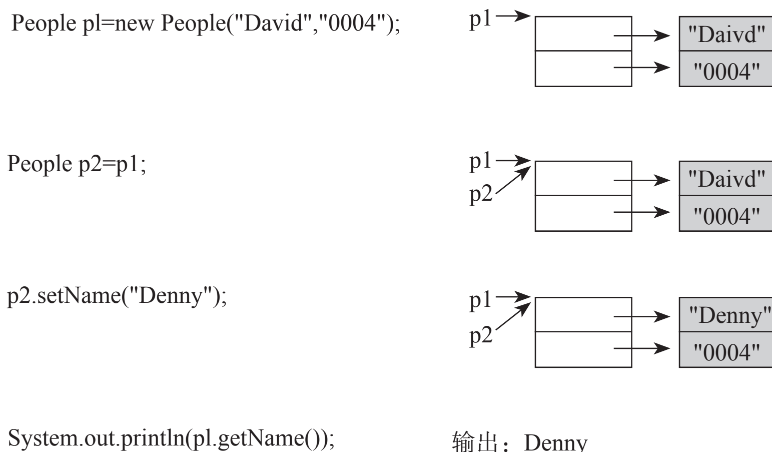


图 1-4　两个对象引用变量指向同一个存储空间

请读者继续考虑以下代码的运行结果。

```
People p1 = new People("David","0004");
People p2 = new People("David","0004");
System.out.println(p1= = p2);
```

在这里虽然 p1 与 p2 的所有成员变量的内容均相同，但是由于它们指向不同的存储空间，因此，输出语句输出的结果为 false。图 1-5 说明了 p1 与 p2 的指向。

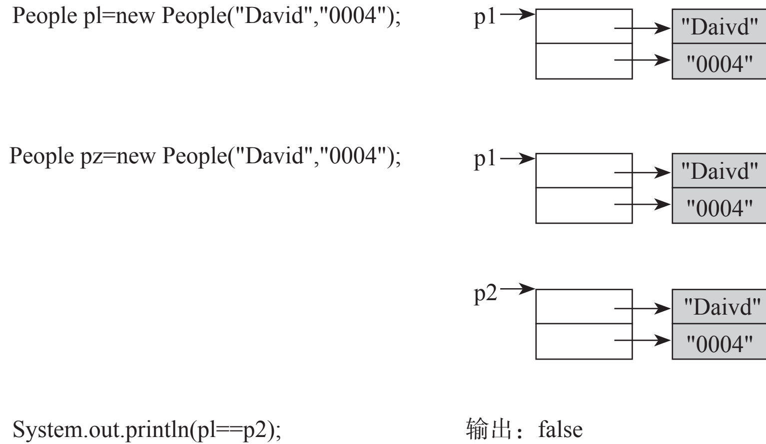


图 1-5　p1 与 p2 指向不同存储空间

可见如果希望完成对象的拷贝，使用一个简单的赋值语句是无法完成的。要达到这一目的可以通过实现 Cloneable 接口并重写 clone 方法来完成。如果希望判断两个对象引用是否一致，可以通过覆盖继承自 Object 类的 equals 方法来实现。