

高级程序设计语言



孙吉鹏 杜泽林
林子童 徐卫霞
谷一滕 鲍 伟
袁郭苑 张晓敏



由衷地为你开始了你的第一门编程语言的学习感到开心，在国外，以 Java 为起手作为第一门高级程序设计语言的大学并不多，但这并不意味着 Java 不好，相反，他们都是以仰视的态度来看待这门语言，认为它是大项目的代名词，因为 Java 这门语言，是 Android 应用，金融服务器，网站，大数据 Hadoop 的一些基础语言，是企业级使用最多的语言。它又是一种纯面向对象的语言，你可能现在不清楚这个词是什么意思，但随着学习的深入你会发现面向对象开发技术的优雅与强大。一开始就接触这种强大的语言，比 Python 复杂，却比 C++ 省心，这是对你的挑战，更是对你的收获。

但是，注意，这门课的名称叫做高级程序设计，并不叫 Java，它的目的是将你带入编程世界，不同编程语言之间总是殊途同归，掌握了一门语言后，你会发现一通百通，所以不要囿于一门语言的边界，重在体会编程的思想，把它当作工具，开始你对计算机世界的探索。

我们希望，通过智库的努力，能够帮助现在的你不要再像当时我们探索时那么迷茫，但更希望你们不要因为智库带来的安逸，失去主动探索的能力。去突破我们，突破课本的边界吧，这个世界，需要你我一起去探索未知，希望走在我们铺的小路上的你，能为后来人踏出新的大道！

Welcome!

第一章 导论

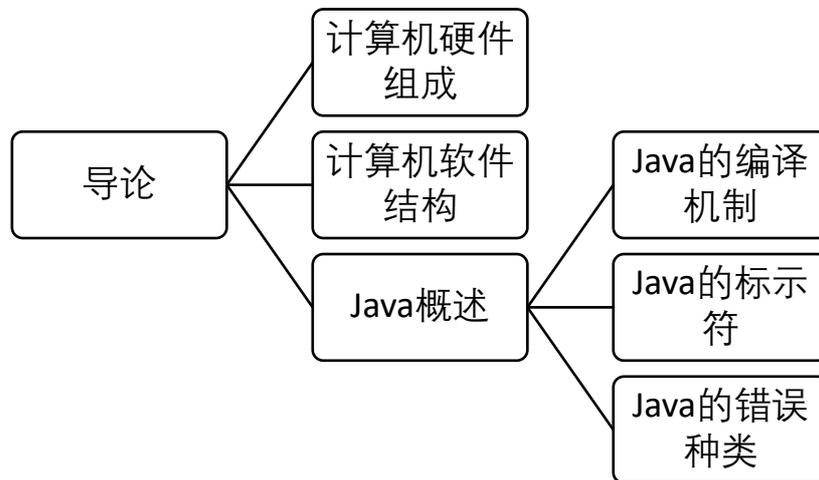
作者：孙吉鹏

1.1 本章概述

本章作为软件工程专业的第一门专业课——高级程序设计语言（Java）的第一章，一定是肩负了介绍计算机基本知识和 Java 基本特点的重任。但是这一章的内容更多的是大学四年其他专业课的简单介绍（计算机组成原理、操作系统、计算机网络、面向对象开发技术、软件工程、编译原理等），我们作为这个专业的初学者在这门课中对这章知识不需要深究其原理，记住常识知识即可，更重要的是以这章知识为景区入口的大地图，概览风光，心怀期待，剩下的交给以后的日子用脚步去亲自丈量体验！

好，让我们出发！

1.2 知识框架



1.3 要点解析

1.3.1 计算机的硬件组成部分

CPU, 主存储器, 辅助存储器, IO 设备

计算机计算机, 顾名思义一开始是想用来计算的, 我们平时用的计算器包含什么? 算数的电路 (控制器和运算器, 现代计算机里包装在一起称 CPU) 和输入按键, 显示屏 (IO) 和保存中间结果的部分 (主存储器)。如果我想保存更多的历史结果和数据怎么办? 于是加入了断了电也能用的大容量存储器 (辅助存储器)。这就是计算机的组成部分, 这么设计不是拍脑袋想出来的, 是有原因和思路的哦!

1.3.2 计算机的软件组成结构

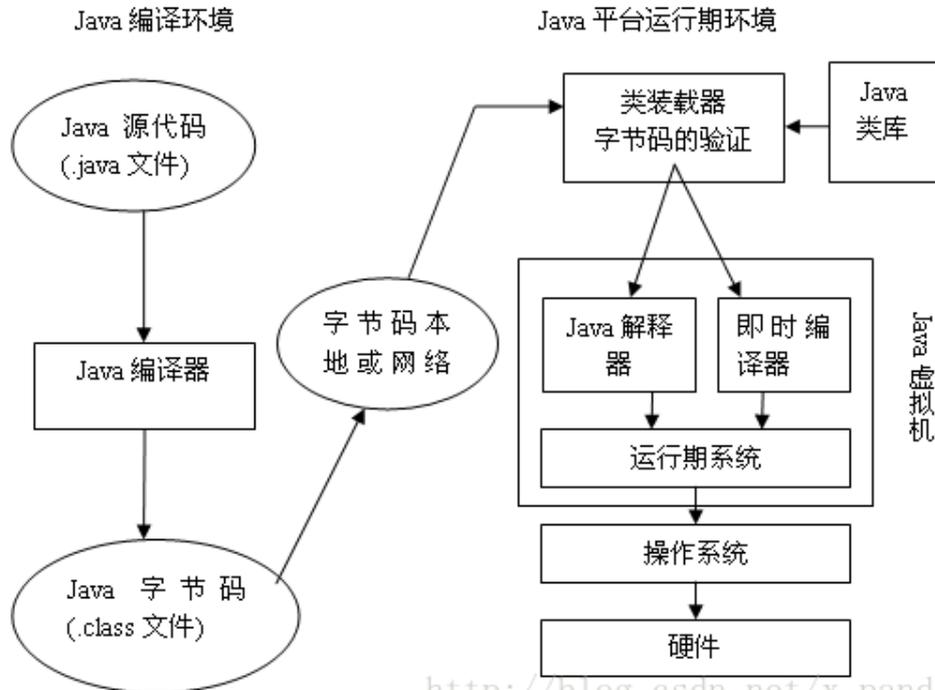
软件是用来帮助人方便地使用计算机硬件的, 高级计算机的软件分为专门与硬件打交道, 为上层各种软件提供服务的操作系统层和功能各异的应用软件层。这种分层帮助应用软件的开发更为便捷。

1.3.3 Java 是个软件么? Java 是用来干什么的?

你可能不假思索地回答是啊, 我都装了 Eclipse 了, 它就用来写程序的呗! 要这么说, Eclipse 是个软件, 它是更好的帮助你写程序的, 自带了语法检查, 拼写补全, 显示结果等辅助功能, 但没了 Eclipse 就没法用 Java 语言了么? 当然不是, 大不了用记事本写选择用 Java 编译运行就可以啊, 只是没了那些辅助功能了么。所以 Java 是什么? 是种创造其他程序的方式, 是种将你能看得懂的自然语言单词命令映射成机器看得懂的 01 二进制指令的映射方式。C 是什么? C++ 是什么? Python 是什么? 都是这个答案, 用心体会。

1.3.4 Java 是如何将你看得懂的语言转化成机器执行的语言的？

你在编辑器里写的 Java 代码叫 Source Code (源代码)，后缀名为 .java。Java Compiler (Java 编译器) 负责将你的源代码统一转化成一种中间形式的代码，叫 bytecode (字节码) 后缀名为 .class，Java 为了保证让你写的代码可以不受 CPU 类别不同的限制在哪里都能运行于是加入了 Java 虚拟机，然后再由 Java 虚拟机里的专门的字节码解释器 (Bytecode interpreter) 和字节码编译器 (Bytecode compiler) 转换成不同机器的机器码。



http://blog.csdn.net/x_panda

1.3.5 一个简单可运行的 Java 程序的结构是什么？

class 中套着方法，一个工程只能有一个名叫 main 的方法，表示计算机从这个方法里的内容开始执行。具体的书写规则会在后面章节加深理解。

```

// comments about the class
public class MyProgram
{
    // comments about the method
    public static void main (String[] args)
    {
    }
}

```

method body

method header

1.3.6 注释 (Comments) 是什么，怎么写？

注释是写程序时为了帮助自己和其他编程人员理解的内容，里面可以写任何内容，因为编译时编译器会忽略其内容直接编译执行代码，所以不会影响机器执行。提倡写注释，在软件项目中写好关键注释会给你加分哦！

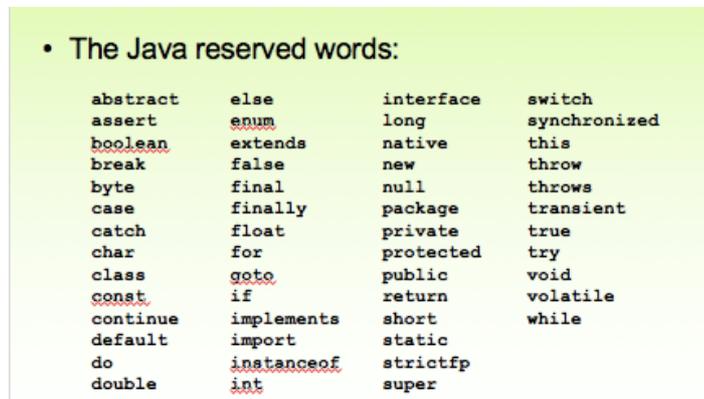
用//表示这一行都为注释内容

用/* 和 */符号把任意行的内容作为注释。

1.3.7 标示符 (Identifiers) 是什么，有什么命名规则？

标示符就是源程序中的一个一个单词，它的规则是规定下来的，需要记忆。

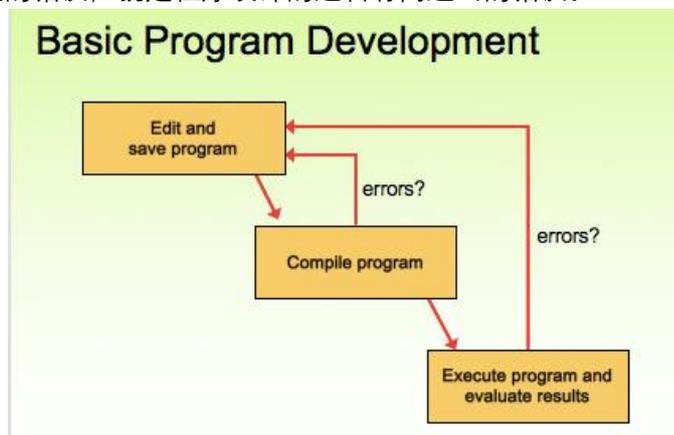
- (1) 在 Java 中，由数字，字母，下划线，\$构成。
- (2) 在 Java 中，不能由数字开头。
- (3) 在 Java 中，大小写敏感，即大小写有区别
- (4) reserved words (保留字) 只能用来表示其特定作用 (遇到时再细提)，不能用做其他作用。



1.3.8 标示符 (Identifiers) 是什么，有什么命名规则？

错误大体分为三类：

- (1) 语法错误(compile-time errors 编译时错误)
编写时语法不准确，命名不守规则等，这些都是编译器可以发现的错误，写时报的错。
- (2) 运行时错误(run-time errors)
语法没错误但是出现除零啊等算数逻辑错误。
- (3) 逻辑错误(logical errors)
没有以上的明显的错误，就是程序设计的逻辑有问题出的错误。



1.4 经典例题

1. 下列选项中符合 Java 命名规则的标识符是 (D)。
A. Student@sdu B. #myname
C. 2SUN D. _endline
2. 下列关于计算机系统和 Java 编程语言的说法, 正确的是 (C)
A. 计算机是由硬件、操作系统和软件组成, 操作系统是缺一不可的组成部分。
B. Java 语言编写的程序源代码可以不需要编译直接在硬件上运行。
C. 在程序中书写注释不会影响程序的执行, 可以多写一些详细的注释。
D. Java 的集成开发环境 (IDE), 如 Eclipse, 是开发 Java 语言必需的软件工具。
3. 程序结束运行后结果为 200, 正确结果应为 100, 这属于什么错误? (D)
A. 语法错误 B. 编译错误 C. 运行错误 D. 逻辑错误

第二章 数据类型和表达式

作者：杜泽林

2.1 本章概述

如果将一个完整的软件项目比作现实世界里的高楼大厦，其中最基础的部分就是数据类型和表达式，这二者好比大楼的砖瓦。一道程序的效率高低往往是由选取的变量类型和表达式所决定的。从熟悉各种变量和表达式开始，一窥计算机与软件的魅力。

2.2 从"Hello World!"开始

每个程序员都是从输出一句"Hello World!"开始学习一门语言，但是不同的语言有不同的输出语句，Java 中的方式是 `System.out.println ("Hello World!");`。

不要小瞧了这行简单的代码，通过它可以认识到 Java 的很多特点。首先这是一句经典的函数调用，其次它涉及到最经典的字符串变量。下面将为大家介绍 Java 的函数调用和字符串类型变量。

2.2.1 Java 函数调用

各种各样的方法 (method) 被放在不同的类 (object) 中，要调用这些方法需要使用 `Object.Method(parameters);` 这样的语句，Object 是类名，Method 是方法名，parameters 是参数。输出"Hello World!"时需要用到 `println` 方法，`println` 方法在 `System.out` 类中，"Hello World!"就是方法的参数。

需要注意，很多方法需要先将对应的类实例化后才可以调用，不用实例化即可用类名调用的方法是静态方法。这其中的原因在学习类与方法时大家就会了解了，现在大家可以保持好奇但不必过度纠结。

2.2.2 String 类型

可以这样讲，字符串 (String) 类型是最经典的数据类型。String 类型的成功是多方面的，它不仅很容易的拆分和拼接，还可以与其他数据类型很容易的转化。所以熟悉 String 类型的使用是很有必要的。

(1) 字符拼接

String 类型的拼接使用 "+" 即可。但是要注意与算数加法区分。

以下是经典例题：

```
System.out.println ("24 and 45 concatenated: " + 24 + 45);  
System.out.println ("24 and 45 added: " + (24 + 45));
```

输出是：

```
24 and 45 concatenated: 2445  
24 and 45 added: 69
```

分析：

第一句只做了字符串拼接，24 和 25 都当做字符串处理。

第二句先做了算数加法，再做字符串拼接。

再看一道两者结合的例题：

如果 `int x=20, y=5`，则语句 `System.out.println(x+y +""+(x+y)+y);` 的输出结果是 (D)
A. 2530 B. 55 C. 2052055 D. 25255

(2) 转义字符

<u>Escape Sequence</u>	<u>Meaning</u>
<code>\b</code>	backspace
<code>\t</code>	tab
<code>\n</code>	newline
<code>\r</code>	carriage return
<code>\"</code>	double quote
<code>'</code>	single quote
<code>\\</code>	backslash

很多时候我们希望字符串的输出格式规范一些，这时我们可以用一系列转义符来实现我们的需求。当 Java 看到字符串中的“\”符号时，它会知道这是 要对字符串的格式进行操作了，所以“\”被称为转义符。但是这样我们要 想简单的输出“\”就需要用“\\”来表达了。

大家思考一下下面的实例：

```
System.out.println ("Roses are red,\n\tViolets are blue,\n" +  
    "Sugar is sweet,\n\tBut I have \"commitment issues\", \n\t" +  
    "So I'd rather just be friends\n\tAt this point in our " +  
    "relationship.");
```

它的输出应该是：

```
Roses are red,  
    Violets are blue,  
Sugar is sweet,  
    But I have "commitment issues",  
    So I'd rather just be friends  
    At this point in our relationship.
```

理解了这个例子，转义字符的使用就没有问题了。

2.3 基本数据类型和表达式

前面我们已经接触到了一种 String 类型的变量，还有多种性质各异的变量需要我们学习。有专门用来表示数据的也有专门用来表示真假值的。表达式是和各种变量紧密相关的。特别注意的是：Java 的基本数据类型只有 8 个，分别是：byte, short, int, long, float, double, char, boolean。String 类型并不是基本数据类型。

2.3.1 数值类型

表示数据的变量有一个大家族，有 byte, short, int, long 以及 float, double。

(1) 整数

整数类型的表示相对简单，就是将日常中使用的十进制转化为二进制数。

byte, short, int, long 都属于整数类型变量，但是它们的表达数据的能力有些区别。

byte：占 8 位 bit (1 字节) 表达范围：-128~127

short：占 16 位 bit (2 字节) 表达范围：-32768~32767

int：占 32 位 bit (4 字节) 表达范围：-2,147,483,648~2,147,483,647

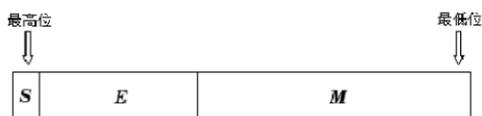
long：占 64 位 bit (8 字节) 表达范围：-9 x 10¹⁸~9 x 10¹⁸

因为需要表示正负数，所以每种类型的第一位数用 0 表示正数，1 表示负数。除此以外，正数比负数少一个，因为有 0 需要表示。所以一个占 Mbit 的数，它的表示范围是 -2^(M-1)~2^(M-1)-1。

(2) 浮点数

浮点数的表示和整数类型大不相同，它更接近于我们使用的科学计数法。

一个 32 位的浮点数在计算机中表示如下图所示：



$$n = (-1)^s \times m \times 2^e$$

其中 s 是符号位，e 是指数位，m 是低数位。

float, double 都是浮点数类型，它们的表达能力比正数类型大许多。float 是 32 位，也称为单精度浮点数。double 是 64 位，也称为双精度浮点数。关于浮点数的使用和表示，今后的课程《计算机组成原理》有详细的分析，这里不深入解读。

(3) 数据类型转换

不同的数据类型之间可以相互转换，表达能力小的变成大的自然没什么问题，但是表达能力大的变成小的会出问题，要谨慎使用。

Widening Conversions		Narrowing Conversions	
From	To	From	To
byte	short, int, long, float, or double	byte	char
short	int, long, float, or double	short	byte or char
char	int, long, float, or double	char	byte or short
int	long, float, or double	int	byte, short, or char
long	float or double	long	byte, short, char, or int
float	double	float	byte, short, char, int, or long
		double	byte, short, char, int, long, or float

2.3.2 其他类型

除了表示数据，我们也经常需要记录诸如判断的真假或单位字符的内容。Java 中提供的 `boolean` 和 `char` 类型就是具有这种表达能力的数据类型。

`boolean` 类型只有两种值：`true` 和 `false`。在条件语句中经常使用。

`char` 类型只保存一位字符：比如 `'a'`、`'b'`。在字符串处理时经常使用。

关于 `char` 类型的补充：

类型转换的表中出现了 `char` 类型，而其它都是正规的数值类型，很多同学一定有这种疑惑——Java 的 `char` 类型中保存的是字符还是数字？Java 字符存储使用 Unicode 编码，每个字符对应 2 个字节（16 位）的二进制数字，Java 要根据这串数字在字库中找到对应字符。因此 `int a = 'a'`；这个语句可以得到字符 `'a'` 的 Unicode 编码，也就是说，`char` 类型的数值转换其实是 Unicode 编码被转换成不同数值类型。

2.3.3 表达式

(1) 变量赋值

最基本的表达式就是为变量赋值的表达式。变量赋值使用的符号是 `=`，它将右边的值赋给左边。这个值要符合左边数据类型的规范。另外如果变量前有 `final` 关键字的约束，那么这个变量一经赋值其值便无法被第二次赋值。在之后学习类与方法时也会遇到 `final` 关键字，届时要注意二者的区别。

很容易被人们忽略的地方：`float A=41.3f`；数字最后的 `f` 一定要有，告诉 Java 这个浮点数是 `float` 类型的。因为 Java 默认的小数（浮点数）的表示方式是 `double`，而 `double` 的表达能力大于 `float`，如果去掉 `f`，相当于将 `double` 的值赋给 `float`，这样是不恰当的。并且浮点数的赋值可以用科学计数法来表示。

典型例题：

下面赋值语句中正确的是 (A)

A. `double d=5.3e12;` B. `float f=11.1;`

C. `int i=0.0;` D. `Double oD=3;`

这道题值得大家记住它。其中的 A、B 选项很容易迷惑大家，A 是浮点数的科学计数法，B 没有 `f` 标记，这是大家容易忽视的知识点。而 D 选项是没有注意数据类型声明的大小写。

(2) 牵涉数据类型转换的运算

Java 中表达式的使用和现实世界中的加减乘除没有分别，运算优先级也么什么不同，但是由于多种数据类型的加入，其中必然牵扯到变量的转换，因此我们需要了解其中的一些特性。

需要注意的有以下几点：

1. 两个数据类型相同的数作运算，结果和他们的数据类型相同。因此整数类型的结果只保留整数部分。
2. 两个数据类型不同的数作运算，先将表达能力小的转换成大的，再作运算，结果是表达能力大的那种类型。

结合下面的例子来理解：

```
public static void main(String[] args){
    int count = 12;
    double sum = 490.27;
    System.out.println(sum / count);
}
```

分析：先将 count 转成 double 类型，再计算 sum/count。输出的结果是 double 类型，输出：40.85583333333333。

3. 在运算中显式的类型转换，数据类型与转换类型相同，其精度也于转换类型相同。

结合下面的例子来理解：

```
public static void main(String[] args){
    int total = 50;
    int count = 6;
    float result = (float) total / count;
    System.out.println(result);
}
```

分析：total 和 count 都是 int 类型，如果不强制类型转换，那么输出的结果将是 8，整数之后的部分会丢失。而强制转换之后小数点之后的部分被保留下来，且精度为 float 类型，输出：8.333333。

注意：这一部分内容只是看讲解可能印象并不深刻，建议大家设计几个小例子，亲自动手试一试。

经典例题：

如下 Java 语句 `double x=2.0; int y=4; x/=++y;` 执行后，x 的值是 (C)

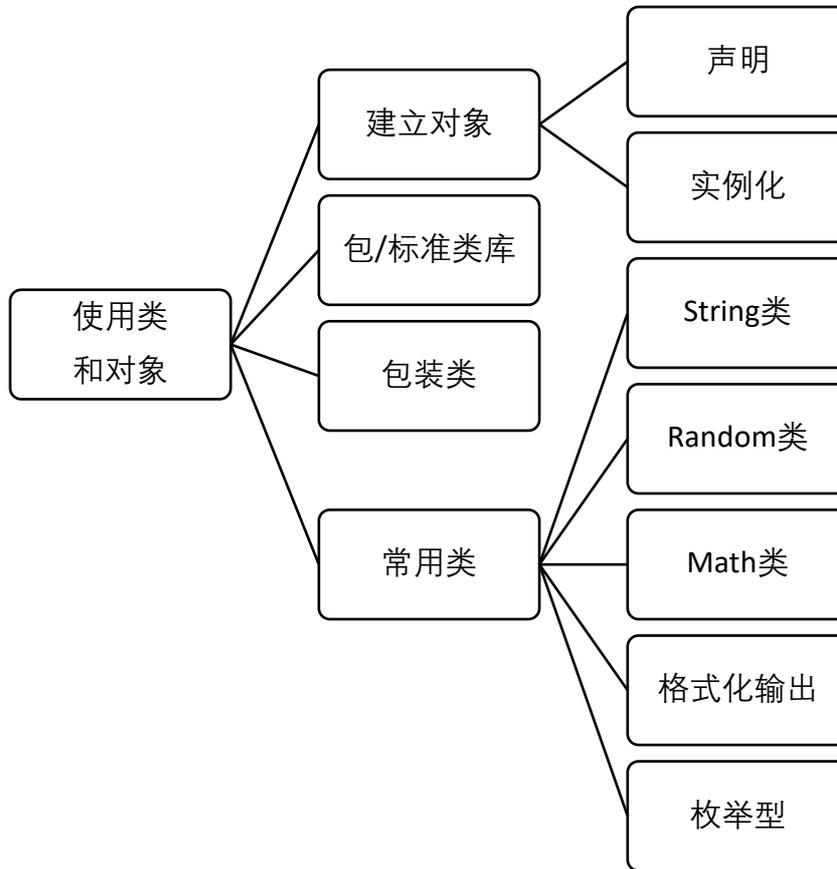
A. 0.5 B. 2.5 C. 0.4 D. 2.0

分析：x/=++y 是表达式的简写方式，将它展开写成：`x=x/(y+1)`，x 是 double 类型，y 是 int 类型，结果应该是 double 类型，结果等于 $2/5=2.5$ 。

第三章 使用类和对象

作者：林子童

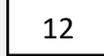
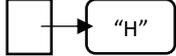
前面讨论了类和对象的基本关系以及基本数据类型，接下来要深入讨论关于类和对象的概念，即 java 的其他常用的类与方法。



3.1 建立对象

3.1.1 对象引用变量的声明和初始化

java 中，变量名代表一个基本数据类型的值或者一个对象，下面来区分二者。基本数据类型以 int 为例，而对象则以常用的 String 类型为例。

声明		初始化	
int num;	num 	num=12;	num 
String name;	name 	name=new String("H");	name 

注意：

1) 基本类型变量名代表的是该类型的一个值，而对象变量名保存的则是保存了该对象真实数据的地址，所以也称为对象引用变量

2) 经过声明的变量开始并没有存放任何数据（没有初始化）。必须将变量初始化才能使用。当然也可以将其设置为 null，表示不指向任何对象，空

3) 对象初始化后，可用对象名.方法(参数)来引用该对象的方法。没有参数也要保留括号以标志方法。其中某些方法还会有返回值，比如 name.length()就是返回该 String 对象的长度，需要有一个 int 对象来存储这个返回值，即 count= name.length();

4) new 运算符相当调用类的一种特殊方法：构造方法，该方法名与类名相同，会初始化一个新的对象将参数传递进去以后返回该对象引用变量的地址。

5) 声明和初始化可以直接合并为一步操作，与基本类型相同，如上面的例子：

```
String name=new String("H");
```

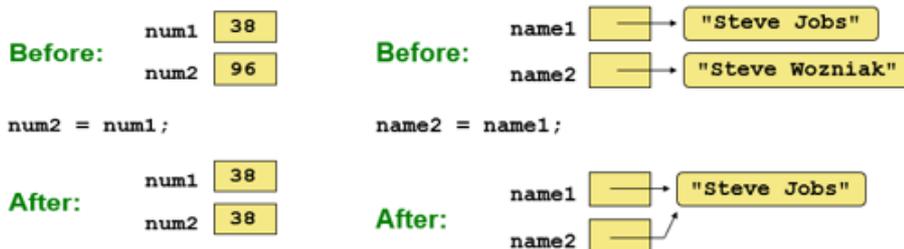
6) String 虽然不是基本类型，但是因为常用所以可以直接和基本类型一样声明：String city="Beijing";

3.1.2 别名

由于基本类型变量名代表的是该类型的值，而对象引用变量名代表的是地址，所以在出现 classA=classB 的赋值语句时，会有不同的结果。

基本类型：

对象引用变量：



可以看到，基本类型在赋值以后只是将 num2 内存单元中的数据修改成了 num1 中的，二者还是不同的内存单元，但是对象引用变量则是在赋值后二者指向了同一个内存单元，而 name2 原本指向的那个对象因为没有了引用而用不会被使用，该对象称为垃圾，会被 java 的垃圾回收机制处理释放该资源。

这两种类型赋值时的本质是一样的，在 `classA=classB` 时都是 `classB` 中内容替换 `classA` 中内容，只不过对象引用变量中内容是某个对象的引用（也就是某个对象的地址），所以才会出现不同。

对于上面 `name1` 和 `name2`，它们引用相同对象而变量名不同所以互为对方别名。

3.2 String 类

一旦建立 `String` 对象，该对象就是不可修改的，所以 `String` 对象是不可变的，所有 `String` 类的方法都是返回新的 `String` 对象而不是修改原来的对象。

课本上有 `String` 类常用的方法以及一个示例，可以用于初学者了解语言的运行过程，但是通常学习 `java` 并不要求掌握这些方法，只需要在编译器中会使用即可。初学者更应该学习的是调用方法的机理。

当然，在了解机理以后熟练掌握常用方法能大大提升编程能力，请各位读者量力而行（以下其他类的要求也同 `String` 类）

3.3 包

`java` 语言有一个标准类库支持，可以根据需要直接使用。类库的类被划分成若干个包，每个包都有自己特有的功能。当用户想要使用某个功能的时候，可以在程序最开头使用 `import` 声明将该包中的某个类引入程序，这个类就变成了可以使用的类。

如果用户想要使用一个包中的许多类，可以直接将整个包引入。如：

```
import java.util.Scanner; //引入 java.util 包中的 Scanner 类
import java.util.*;      //引入 java.util 包中所有类
```

这里再明晰一下类的概念。在前面的学习中，我们接触到了基本类型和 `String` 类型，它们都有自己对应的方法。而实际上在 `java` 中有很多其他的或者来自标准类库或者人为定义的一些类，如 `pen` 或者 `dog`。这些将在第四章展开讨论，这里只需要明晰类的概念即可。

关于类库/包，只需要了解概念，实际应用中借助编译器就可以使用。当然如果对于常用的包比较熟悉更好。

注意，`java.lang` 包是最基本的包，可视为语言的基本扩展，所以会被自动导入程序，包中任何类如 `String`，`System` 都可以直接使用而不需要 `import`。

3.4 Random 类

在 `java.util` 包中，能产生一个伪随机数

使用示例：

- 1) `Random rand=new Random();`//rand 是个能随机数的“种子”/生成器
- 2) `int num1=rand.nextInt();`//num1 是 rand 随机产生的一个 `int` 值
- 3) `num1=rand.nextInt(10);`//num1 是 rand 随机产生的一个 0-9 区间的 `int` 值
- 4) `num1=rand.nextInt(10)-1;`//num1 是 rand 随机产生的一个 -1-8 区间的 `int` 值
- 5) `float num2=rand.nextFloat();`//num2 是 rand 随机产生的一个 `[0.0,1.0)` 浮点数
- 6) `float num2=rand.nextFloat()*6;`//num2 是 rand 随机产生的一个 `[0.0,6.0)` 的浮点数

数

3.5 Math 类

这个类中有着大量基本数学函数，定义在 `java.lang` 包中。Math 类所有方法都是静态方法，可以直接使用 `类名.方法(参数)` 调用而不需要实例化类的对象。第 6 章详细讨论静态方法。Math 类也同上只需了解不用把所有方法都记下来。具体使用方法如下：

```
int a=Math.abs(-3); //a 的值为-3 的绝对值
```

3.6 格式化输出

为了使显示的信息看起来格式编排清晰。

3.6.1 NumberFormat 类

提供了通用的数据格式化能力，它的方法也是静态方法。

有 `getCurrencyInstance` 获取当前货币格式和 `getPercentInstance` 获取百分率格式两种方法。以第一种为例进行演示：

```
NumberFormat fmt1=NumberFormat.getCurrencyInstance();//fmt1 设置为当前地区  
货币值格式对象
```

```
System.out.println("价格 : "+fmt1.format(11));//将 11 以 fmt1 的格式输出  
结果 : 价格 : ¥11
```

3.6.2 DecimalFormat 类

使用的不是静态类而是 `new` 一个模式然后进行使用

```
DecimalFormat fmt2=new DecimalFormat("0.###");  
System.out.println("The price is "+fmt2.format(3.32));;  
结果 : The price is 3.320
```

3.6.3 printf 方法

示例：

```
String name="Mike";  
System.out.printf("ID: 232323\tname:%s",name);  
这句话中，%s 所在位置会被 name 替换，最终输出 ID: 232323 name:Mike
```

%s 表示 `string` 类型数据，当然还有其他数据。这种语法更多的用在 `c` 中，在 `java` 中保留但不建议使用，可自行上网学习

3.7 枚举型

为变量设定可取值范围，被认为是一种安全类型。

一个枚举型被认为是一种特殊的类，每个枚举值都有自己对应的枚举序列

```
enum time{winter,spring,summer,fall}其中对应的序列之分别是 0,1,2,3
```

使用示例：

```
time t1=time.summer; //声明并实例化一个 time 对象  
int i=t1.ordinal; //i 表示 t1 的序列号 2  
String s=t1.name; //s 表示 t1 的名字 summer
```

3.8 包装类

包装类是一种对象类，允许将基本类型数据作为对象管理，其主要作用之一就是实现基本类型之间的转换，从而将不同的数据类型转换为一种类型，方便使用和管理。

如：将一个 `String` 类型与 `int` 类型数据进行相加时，需要让二者成为同一类型才可以执行操作。这两种类型之间的转换时最常见的，最好记下方法：

`String` 转为 `int`：`int a=Integer.parseInt(str);`

`int` 转为 `String`：有三种办法：

1) `String s = String.valueOf(i);`

2) `String s = Integer.toString(i);`

3) `String s = "" + i;`

每一个基本类型都对应着一个包装类，而且二者之间可以完成自动包装，也就是互相赋值是不会报错的。但是这仅限于基本类型和包装类，基本类型和对象之间的赋值通常是不兼容的。

3.9 例题

1. 下列在 Java 语言中关于数据类型和包装类的说法，正确的是 (B)

A. 基本（简单）数据类型是包装类的简写形式，可以用包装类替代基本（简单）数据类型。

B. `long` 和 `double` 都占了 64 位（64bit）的存储空间。

C. 默认的整数数据类型是 `int`，默认的浮点数据类型是 `float`。//浮点是 `double`

D. 和包装类一样，基本（简单）数据类型声明的变量中也具有静态方法，用来完成进制转化等。

2. 下列关于包（package）的描述，正确的是 (D)

A. 包（package）是 Java 中描述操作系统对多个源代码文件组织的一种方式。

B. `import` 语句将所对应的 Java 源文件拷贝到此处执行。

C. 包（package）是 Eclipse 组织 Java 项目特有的一种方式。//java 类库可以存在于 java 的任何一种开发环境

D. 定义在同一个包（package）内的类可以不经 `import` 而直接相互使用。

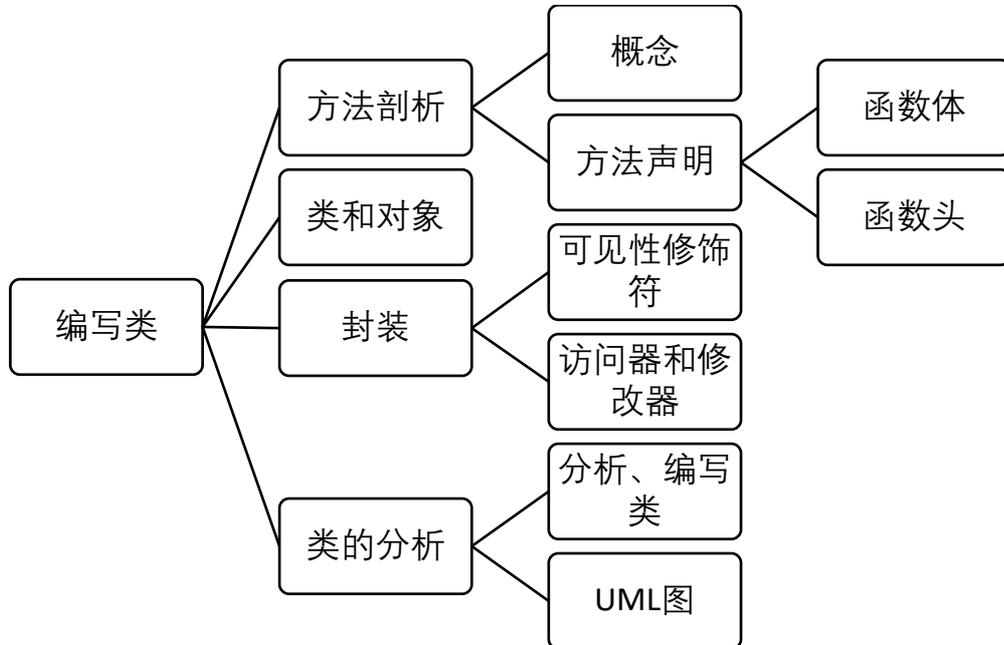
3.10 写在后面

本书前四章都是对概念的基本介绍，学的时候可能一头雾水，但是在多多编程后心中许多疑惑就会解开，真实的编程会告诉你为什么需要这样那样。多多动手熟悉 eclipse，熟悉每一种学过的典型方法，就会发现这些枯燥的概念其中的奥妙之处。

第四章 编写类

作者：徐卫霞

了解对象和类的关系、区别；编写类时最初需要熟悉类的结构（数据、方法）、基础知识（返回值、参数列表等），之后需要能够根据类的特点自己设计它所包含的变量、方法。



4.1 基本概念

类与对象：类是一类相似事物的总和，而对象则是具体存在的一个事物，比如人是个类，而你就是一个对象；又或者学生是一个类，而你是学生这个类的一个对象。

对象：对象是类的一次实现，是有状态和行为的。

对象的状态：由内部的属性（变量）表示，行为由函数实现，例如：学生张三是学生类的一个对象，它有一些状态，比如学习的状态，这可以用一个变量来表示（例如用 `boolean` 类型的变量 `study` 表示是否学习，`true` 表示正在学习，`false` 为玩耍状态）。

对象的行为：以学生张三为例，它存在上课这个行为，所以可以定义一个函数 `Learn ()` 来表示它上课的行为，每次调用这个函数就表示张三进行了上课这个行为。

注意：对象的行为可能会改变对象的状态，上面的例子中当张三执行上课这个函数时它就会改变张三的状态，改变成学习状态。

类的编写：因为我们平时用到的类很多都不是 `Java` 类库的预定义类，所以需要我们自己进行类的编写来实现我们的需求。

4.2 类的分析

对象存在状态和行为，所以作为一类对象的抽象事物，类也存在状态和行为，即数据和方法，类的数据和方法称为类的成员。

4.2.1 编写类的一般方法

(1) 声明类包含的各种变量

(2) 编写类的各种方法，方法中可能会修改类的变量。类的方法包括构造方法和常规方法。写方法时需要考虑方法的返回值、参数等。

4.2.2 概念

构造方法：和常规方法的编写方法没有差别，不同的是它的方法名称跟类名称相同、方法不能有返回值（`void` 返回值也不可以），创建一个对象时会调用它来初始化一个对象（一般是用来初始化对象的一些变量值等）。不必为每个类都创建构造方法，因为每个类都有一个不带参数的默认构造方法。

实例数据：类中的属性（变量），每次创建一个对象时会为这个对象的实例数据分配内存空间，这就是为什么同一个类的不同对象都具有自己的状态。变量的声明位置定义了这个变量的可见范围，变量的可见范围确定了变量在程序中已被引用的区域。当变量声明在类的级别（不是在某个方法内部声明），类内的所有方法都能引用这个变量。

4.2.3 实例分析

(1) 分析

以 `Die` 类（骰子）为例：因为 `Die` 不是 `Java` 类库的预定义类，所以需要我们自己进行类的编写，需要编写类的数据和方法：

① 类的数据声明：分析骰子的各种状态，它有最大面值和当前面值，所以需要两个变量来表示这两个状态。

② 类的方法的声明：骰子具有的行为有旋转、设置当前面值、获取当前面值，所以需要三个方法来分别实现这三个行为。除了类本身的各种行为以外，还可能会声明一些方法用于信息的输出，例如下面要看的方法 `toString()`。

(2) 用代码来编写类

具体代码如下：

```

//*****
// Die.java      Author: Lewis/Loftus
//
// Represents one die (singular of dice) with faces showing values
// between 1 and 6.
//*****

public class Die
{
    private final int MAX = 6; // maximum face value

    private int faceValue; // current value showing on the die

    //-----
    // Constructor: Sets the initial face value.
    //-----
    public Die()
    {
        faceValue = 1;
    }

    //-----
    // Rolls the die and returns the result.
    //-----
    public int roll()
    {
        faceValue = (int)(Math.random() * MAX) + 1;
        return faceValue;
    }

    //-----
    // Face value mutator.
    //-----
    public void setFaceValue (int value)
    {
        faceValue = value;
    }

    //-----
    // Face value accessor.
    //-----
    public int getFaceValue()
    {
        return faceValue;
    }

    //-----
    // Returns a string representation of this die.
    //-----
    public String toString()
    {
        String result = Integer.toString(faceValue);

        return result;
    }
}

```

Die 类包含两个变量（实例数据）：整型常量 MAX 来表示骰子的最大面值，整型变量 faceValue 来表示骰子当前面的数值，这两个实例数据的声明在类的级别，所以 Die 类内的所有的方法都可以引用它们。final 修饰 MAX 表示 MAX 是一个常量（一次执行过程中不能被修改），private 修饰符在后面会讲到。

Die 类包含构造方法和普通方法。它的构造方法是方法名和类名 Die 相同的方法，也就是方法 Die()，它实现的是把当前面的数值初始化成 1。当使用 new 运算符来创建一个 Die 对象的时候，系统会调用 Die 类的构造方法 Die() 完成 facevalue 变量初始化为 1。常规方法包括骰子自身的一些行为和用于输出信息的方法。其中骰子自身的行为包括 roll(), setFaceValue(), getFaceValue() 三个，这跟之前分析的骰子的行为是一致的。而 toString() 函数要执行的操作就是返回 facevalue 变量的值。它的使用情况：把一个 Die 对象传递给 print 方法或者 println 方法的时候，系统会自动调用 toString 方法，输出 toString 方法的返回值，这会在之后的学习中进行分析，此处先了解一下。

注意：Die 类跟之前例子中的类没有什么不同，唯一的区别在于 Die 是我们自己编写的，而其他的是 Java 标准类库提供的。

4.2.4 UML 图

UML：统一建模语言，最流行的一种描述面向对象程序设计的符号体系。

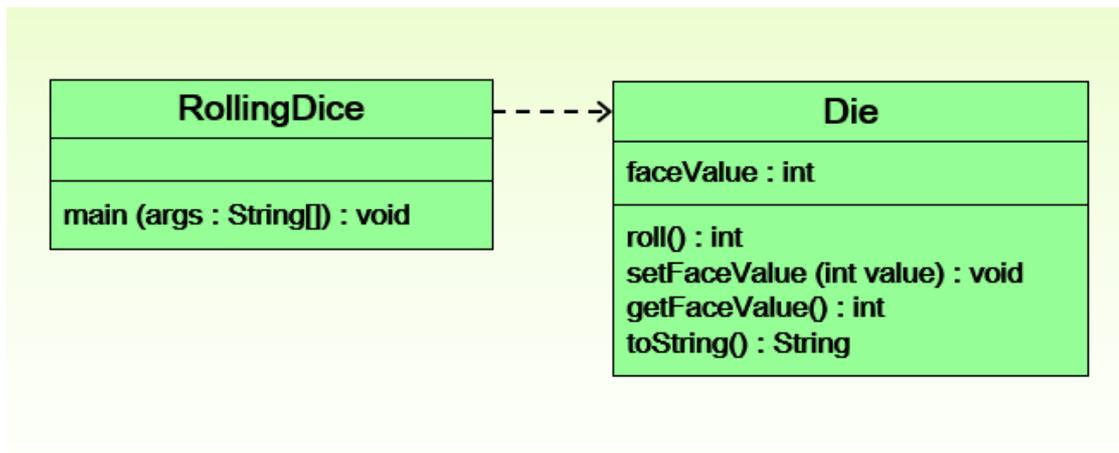
UML 类图：描述类的结构及其类间的关系的 UML 图。

用法：UML 类图中每个类用矩形表示，其中由三部分组成：类名、属性（数据）、操作（方法）。

类间关系：以类 A，类 B 为例

- ①“A uses B”：A 使用 B 的方法
- ②“A has B”：类 A 内部有 B 类型的变量
- ③“A is B”：A 类属于 B 类，例如人类属于生物这个类

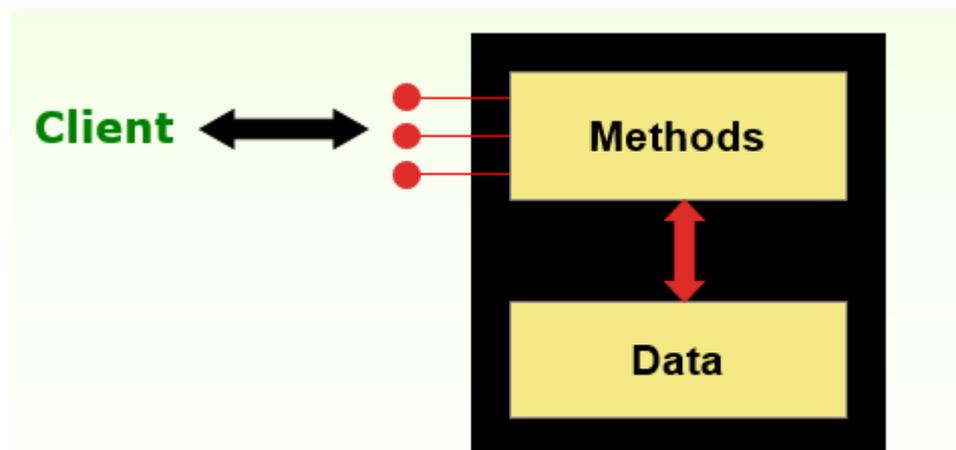
例：下图中 RollingDice 类调用了 Die 的方法，用带箭头的虚线表示，是“use”关系。



4.3 封装

封装：对象必须是独立运行的，内部变量必须经过自己的方法修改，类之外的代码难于甚至无法访问和修改在类内部声明的变量，这种特性叫做封装，java 中通过修饰符（java 保留字）实现封装。

图示说明：client 想要获取对象的数据，必须通过调研对象的方法来获得数据，而不能直接获取数据。



4.3.1 可见性修饰符

Java 中可见性修饰符包括 `private` 和 `public`。

Public：修饰的方法和变量是公开可见的，所有类都可见（可以引用），称为服务方法，UML 图中在变量前面加“+”表示 `public` 可见性。

Private：修饰的方法和变量则只能在类的内部可见，类外不可见（之前的 `Die` 类的 `MAX`, `faceValue` 变量都是 `private` 修饰，所以只在 `Die` 类内可见）。`Private` 修饰的方法只在类的内部可见，所以它一般辅助类内的其他方法工作，也称为支持方法，UML 图中在变量前面加“-”表示 `private` 可见性。

4.3.2 访问器和修改器

访问器方法：类似 `getFaceValue()` 这样的方法，调用这种方法可以得到相关数据。

修改器方法：类似 `setFaceValue()` 这样的方法，调用这种方法可以用于修改实例数据。

规范：一般来说，若变量名为 `Height`，则它的访问器方法和修改器方法分别为 `getHeight()`、`setHeight()`。

4.4 方法剖析

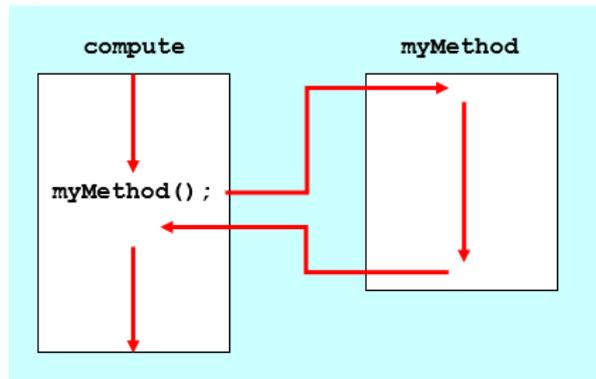
深入剖析类中方法声明的细节。

4.4.1 概念

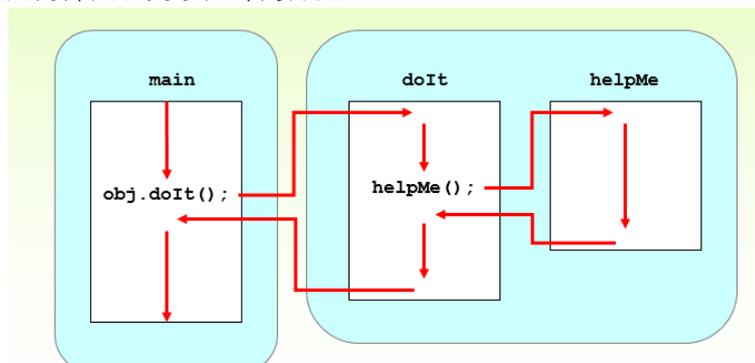
方法声明：调用本方法时要执行的代码。

方法调用：当一个方法被调用时，执行控制流就转移到该方法，并逐条执行该方法中的语句，方法执行完后返回到调用该方法的位置继续往下执行，图解：

情况一：只调用一次方法：



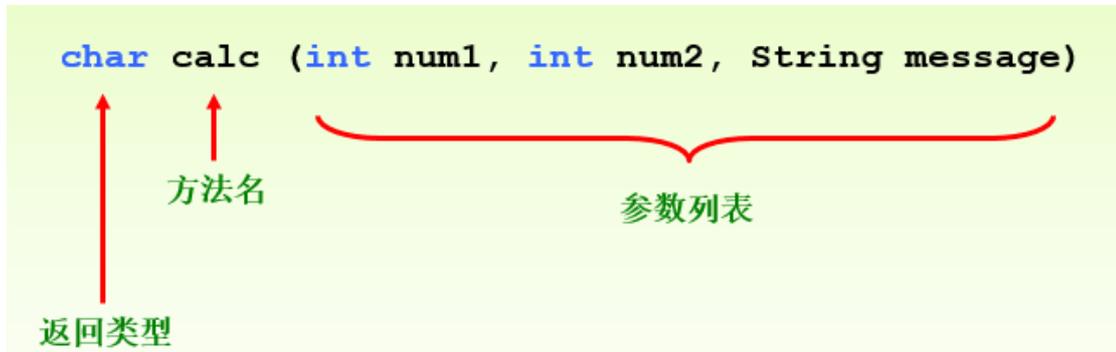
情况二：调用方法内部又调用了另外的方法：



4.4.2 方法声明

函数（方法）包括函数头、函数体两部分，所以函数的编写也需要按这两部分进行编写。

(1) 函数头



①返回类型：函数要返回给调用位置的数据，若没有返回值则写 `void` 作为返回值类型。

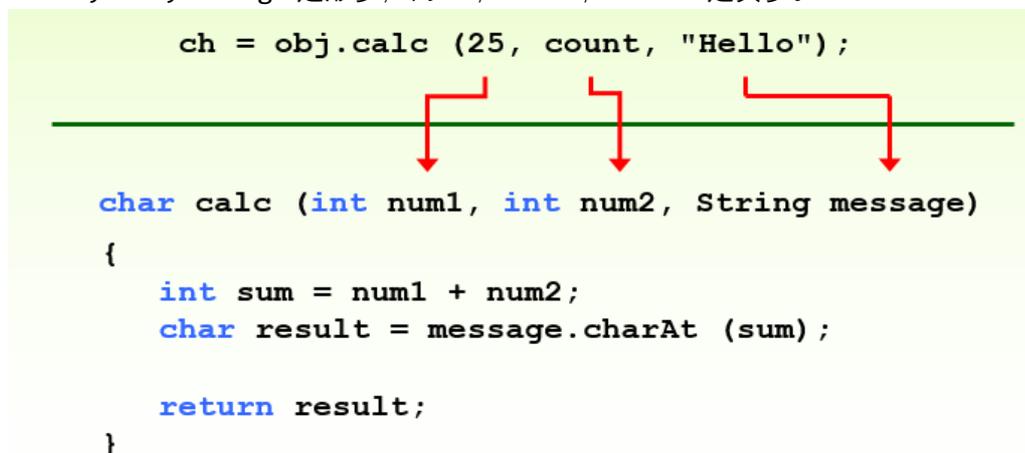
②参数：

实际参数：一次方法调用中，实际传递给方法的参数是实际参数（实参，也称为方法的变元）。

形式参数（形参）：是方法声明时的参数，方法调用时形参的初始值由调用该方法时传递的实参赋予。

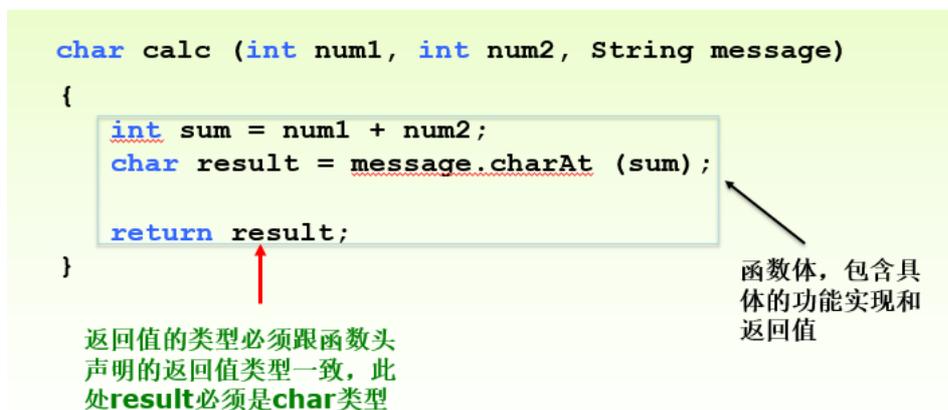
③参数列表：调用这个函数时需要给出的数据，给出的参数列表（数据的类型、排列顺序等）需要跟函数的声明时的参数列表完全一致，若该方法没有参数也需要在方法名后面写一个空括号。

图解：num1,num2,message 是形参，而 25, count, "Hello"是实参。



(2) 函数体

① 返回语句：存在返回值时需要有返回语句 `return`。当执行返回语句时控制立即返回到调用该方法的位置并继续往下执行。返回语句由 `return` 和后续的表达式组成，表达式确定了要返回的值，并且类型和函数头规定的返回值类型一致。一般一个方法只有一个 `return` 语句，写在方法的最后一行。图解：



② 局部数据

局部数据：声明位置在方法内部的变量，相对于实例数据（类级别的）。局部数据只能在声明它的方法内部使用，方法外不可见。

注意：当一个类内存在实例数据和局部数据名称相同时，在局部数据对应的方法内部使用的是局部数据，其他区域都是实例数据，但是这种情况应当避免，因为会造成混淆。

4.5 例题参考

1.

下列关于Java类中方法的定义，正确的是（D）

- 1.若代码执行到return语句，则将当前值返回，而且继续执行return语句后面的语句。
- 2.只需要对使用基本数据类型定义的属性使用getter和setter，体现类的封装性。
- 3.方法的返回值只能是基本数据类型。
- 4.在同一个类中定义的方法，允许方法名称相同而形参列表不同，并且返回值数据类型也不同。

注解：代码执行到 `return` 语句后会立即返回到调用该方法的位置并继续执行，并不会继续执行 `return` 后面的语句。

Getter 和 setter 的使用不仅仅是针对于基本数据类型，而是类内的变量，通过对类内变量的 `get` 和 `set` 实现封装。

方法的返回值类型没有特殊规定，只需要返回值类型和函数头中声明的返回值类型一致。

2.

下列关于Java中类的构造方法的描述，正确的是（B）

- A. 构造方法的返回类型为void
- B. 可以定义一个类而在代码中不写构造方法。
- C. 在同一个类中定义的重载构造方法不可以相互调用。
- D. 子类不允许调用父类的构造方法。

注解：构造方法没有返回值，void 类型也不可以；系统给各个类都有默认的无参构造方法，所以可以不写构造方法。子类调用父类构造方法可以通过 super 实现。

3.

```
public class Shape{
private String Description ;
    Shape (String desc) {
        Description=desc;
    }
}
```

给出属性 Description 的 get 和 set 方法：

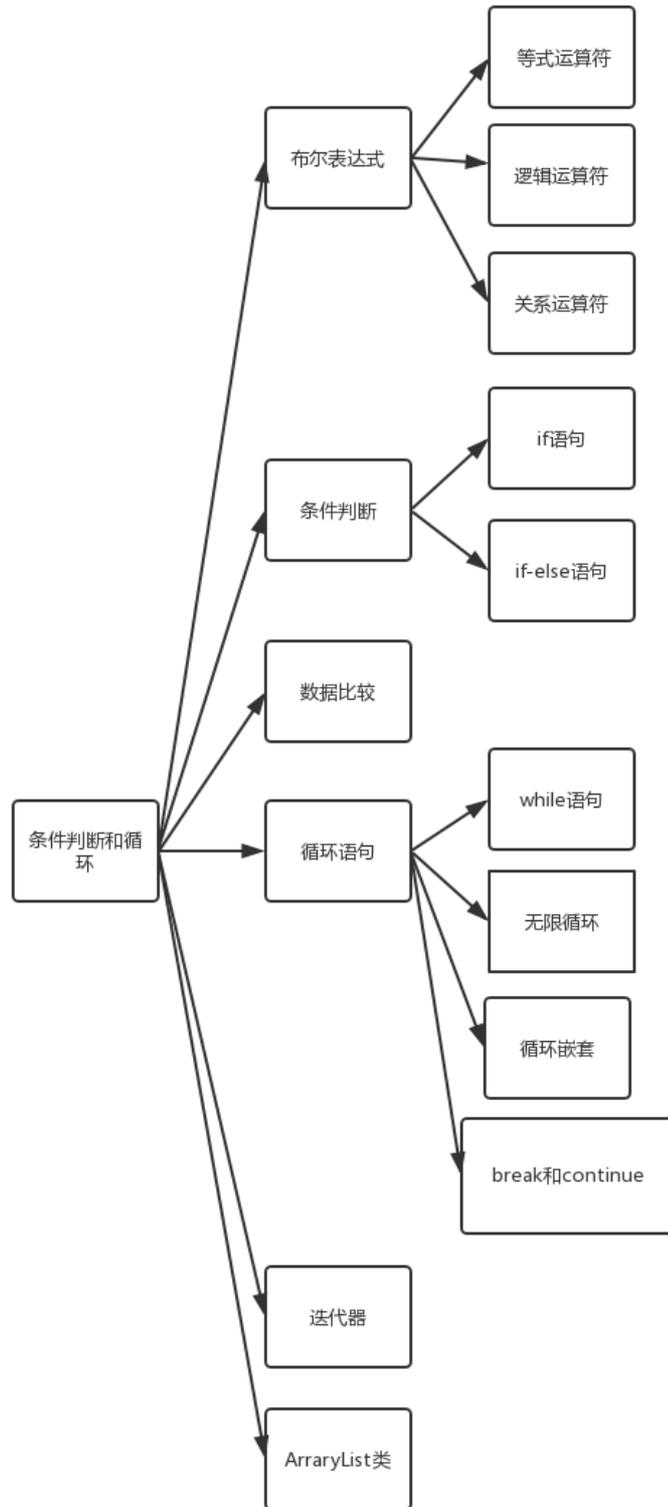
注解：set 和 get 方法的结构已经在 Die 类中大致给出。可参照编写代码。参考代码如下：

```
public void setDescription(String desc)
{
    Description=desc;
}
public String setDescription()
{
    return Description;
}
```

第五章 条件判断和循环

作者：谷一滕

所有的程序设计语言都提供了语句，用于描述下一步要执行的操作。其中一些语句还可以多次重复执行某些操作。本章将讨论这种类型的 Java 关键字。



5.1 布尔表达式

在程序运行中，语句的执行顺序称为**控制流**，控制流默认顺序执行。在一个给定的方法内，可以通过**条件语句**与**循环语句**控制程序执行流程。条件语句允许选择下一步执行的语句，而该决定的产生，基于布尔表达式（也称为条件）。

5.1.1 等式运算符和关系运算符

运算符	运算符意义
==	等于
!=	不等于
<	小于
<=	小于或等于
>	大于
>=	大于或等于

注：等式运算符和关系运算符的优先级比算术运算符低。注意“==”与“=”的区别。

5.1.2 逻辑运算符

运算符	描述	举例	结果
!	逻辑非	!a	a 为真，则!a 为假；a 为假，则!a 为真
&&	逻辑与	a&& b	a、b 同时为真，则表达式的值为真，否则为假
	逻辑或	a b	a 为真或 b 为真或者同时为真，则表达式的值为真，否则为假

注：“&&”和“||”具有“短路”性，即左边的操作数已经足以确定整个运算的结果，那么右边的操作数就不会参与运算。

5.2 if 语句

if 为 java 保留字，其结构为：

```
if(condition)
    statement;
```

其中，condition 为布尔表达式，其运算结果只能为 true 或 false。如果 condition 的结果为 true，则执行 statement 中的语句，否则跳 过。

注：规范使用缩进可以增加代码可读性，而对编译器本身没有影响。

例：

```
if(a > b)
    a = a + b;
```

上例中，如果 a 大于 b，赋值语句就会执行，否则跳过赋值语句。注意 statement 语句中缩进的使用。

5.2.1 if-else 语句

其结构为：

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

其中，condition 的结果为 true，则执行 statement1 中的语句，否 则执行 statement2 中的语句。

例：

```
if(a > b)
    a = a + b;
else
    a = a - b;
```

解释略。

5.2.2 使用语句块

当计算出布尔条件的结果时，为了可以做更多的事情（需要执行多行代码）。使用语句块来替换任意一条语句。

例：

```
if (a == b)
{
    a = a + b;
    b = b + a;
}
else
{
    a = a - b;
    b = b - a;
}
```

不论布尔条件结果如何，都会执行对应的两条语句。

5.2.3 if 语句的嵌套

即一条 if 语句中嵌入另一条 if 语句。

例：

```
if (a < b)
    if (a < c)
        min = a;
    else
        min = c;
else
    if (b < c)
        min = b;
    else
        min = c;
```

这是一个求出三个数字 a, b, c 中最小值的代码，好好体会一下。通过正确的缩进代码有助于我们理解 if-else 匹配的关系。另一个定义是：else 子句和它前面最近的没有匹配项的 if 语句相匹配。上述代码等价于：

```
if (a < b)
{
    if (a < c)
        min = a;
    else
        min = c;
}
else
{
    if (b < c)
        min = b;
    else
        min = c;
}
```

我们在逻辑比较复杂时，可以通过使用语句块的方法来帮助我们理解。

5.3 数据比较

数据类型不同，比较时它们采用的策略也有所不同。

5.3.1 浮点数比较

比较浮点数 (`float`) 时 (`double` 也同理)，等号 (`==`) 的定义是只有两个浮点数的二进制数位都相等时，这两个浮点数的值才相等。即便它们差的十分小。一个策略是设置误差标准。比如误差标准使用 `0.00001` (`1e-5`)。如果两个浮点数很近似，以至于它们之间的差比指定的误差标准还小，就认为它们相等。例：

```
if (Math.abs(f1 - f2) < 1e-5)
    System.out.println ("Essentially equal");
```

5.3.2 字符比较

由于字符以 Unicode 字符集为标准，所以我们可以依据其定义的顺序，表示字符的大小。部分 Unicode 码：

字符	Unicode 码
0-9	48-57
A-Z	65-70
a-z	97-122

5.3.3 比较对象

比较 `String` 对象时，要使用 `equals` (判断内容相等) 方法或 `compareTo` (用于判断字典序，具体方式：按 Unicode 字符集，从左至右逐位比较，此时，若一个字符串是另一个字符串的前缀，则短字符串在长字符串前) 方法。虽然使用 `==` 比较 `String` 对象是合法的，但实际上是比较它们的引用 (即存储值的磁盘地址)。例：

```
if (name1.equals(name2))
    System.out.println ("Same name");
```

判断 `name1` 与 `name2` 存储的内容是否相同

```
int result = name1.compareTo(name2);
```

当 `name1` 字典序在 `name2` 之前，返回负数

当 `name1` 字典序等于 `name2` 的，返回 `0`

当 `name1` 字典序在 `name2` 之后，返回正数

注：比较其他对象时，根据情况需要自行定义 `equals` 方法与 `compareTo` 方法，之后才可使用。

5.4 while 语句

循环语句可用于多次重复执行一条语句。`while` 语句就是一种循环语句。结构为：

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

在每一次循环中，判断 `condition`，如果 `condition` 结果为 `true`，则 `statement` 语句就会被执行，如果 `condition` 结果为 `false`，则终止执行。

例：

```
int count = 1;
while (count <= 5)
{
    System.out.println (count);
    count++;
}
```

该循环语句会依次打印出 1-5 的值，每次循环打出一个值，然后计数器加一。当计数器增加到 6 时，条件为 false，跳出循环。

5.4.1 无限循环

当循环语句的判断条件不能达到 false，循环就会永远执行下去，或者直到这个程序被终止。

例：

```
int count = 1;
while (count <= 25)
{
    System.out.println (count);
}
```

因为循环中 count 的值永远为 1，这就会导致 count 的值始终小于 25，因此循环会无限进行下去。

5.4.2 循环嵌套

一个循环体中包含另一个循环。外层循环每执行一次，内层循环就会执行一轮指定次数的完整循环。

例：

```
int count1, count2;
count1 = 1;
while (count1 <= 10)
{
    count2 = 1;
    while (count2 <= 20)
    {
        System.out.println ("Here");
        count2++;
    }
    count1++;
}
```

外层代码共循环 10 次，在外层代码的每一次循环中，内层代码就会循环 20 次，因此一共会打印出 200 次 "Here"。

5.4.3 break 和 continue 语句

break 和 continue 语句会影响程序中的条件控制和循环。循环体中出现 break 语句时，将终止循环的进行；出现 continue 语句时，会跳过其后的循环体语句，并开始执行一次新的循环。

例：

```
int count = 1;
while(true)
{
    count++;
}
```

```

    if(count>20)
        break;
}

```

该语句中看似是一个无限循环，但由于 count 在每次循环中加一，当 count 大于 20 时，if 语句的内容就会执行，即 break 语句，这也就跳出无限循环。

```

int ans = 0;
while(ans <= 25)
{
    if(ans%2==0)
        continue;
    System.out.println (ans);
}

```

该语句用于输出 0-25 之间所有的奇数，在 if 语句中，判断 ans 是否为偶数，如果为偶数，执行 continue 语句，即直接进行下一次循环（跳过 continue 后的循环语句），不然就会执行输出语句，即将奇数打印出来。

5.5 迭代器

迭代器是用来逐次处理集合中一项元素的一种对象。它使用了 Java 中的 Iterator 接口（第七章）。

5.5.1 迭代器对象的基本方法

hasNext 方法：返回布尔值，表示集合中是否还有至少一个元素要处理。

next 方法：用于取得集合中的下一个要处理的元素。

hasNext 与 next 方法的变形：如 Scanner 类中的 hasNextInt 和 hasNextDouble 方法，用于判断下一项输入是否符合特定类型。Scanner 类中的 nextInt 和 nextDouble 用于获得特定的类型值。

5.6 ArrayList 类

ArrayList 类用于管理对象集，属于 Java 标准类库的 java.util 包。可以通过索引值来查找对应对象，也可以使用循环语句遍历列表中的各个对象。一个 ArrayList 对象可以根据需要来增加自身长度，既可以向其插入元素，也可以将其存储的元素移除。起始下标从零开始。

结构为：

```
ArrayList<Object> arraryList= new ArrayList<Object>();
```

注：Object 为存储对象的类型。ArrayList 存储的是对象的引用，因此它不可以存储基本类型的变量（如 int,double 等）。但可以使用包装类解决此问题，如创建 ArrayList<Integer>对象或 ArrayList<Double>对象。

ArrayList 类包含方法有：

```

boolean add (E obj)
void add (int index, E obj)
Object remove (int index)
Object get (int index)
boolean isEmpty()
int size()

```

具体用法参照 Java API

5.7 例题

下列语句序列执行后，x 的值是：

```
int a=3, b=4, x=5;
if(++a==b)
    x=++a*x;
```

- A. 5 B. 20 **C. 25** D. 35

选出下列关于 break、continue 和 return 的说法中正确的选项。

- A. 对于多重嵌套循环，break 跳出所有循环而到达最外层，顺序执行后面的程序。
 B. 在循环语句中，continue 语句和 break 语句作用相同。
 C. break 语句只在循环语句中使用。
 D. 在循环中执行 return 语句，将跳出所有循环。

二、读程序题，写出下列各段程序的执行结果（30 分）

1. public class TwoOne {

```
    public static void main(String[] args) {
```

```
        int number = 21;
```

```
        while(number>0)
```

```
        {
```

```
            System.out.print(number++%3+"\t");
```

```
            number/=3;
```

```
        }
```

```
    }
```

```
}
```

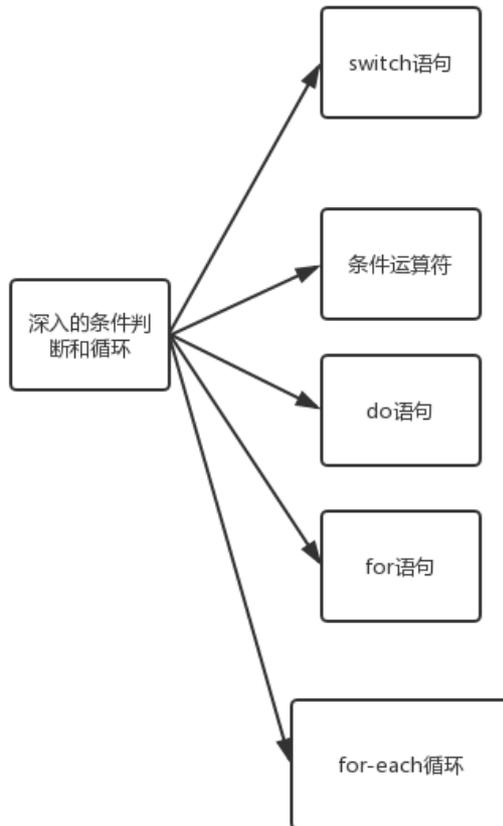
(6分)

```
21 22 7 8 2 3 1 2 0
0 1 2 1
```

第六章 深入的条件判断和循环

作者：谷一滕

本章主要讨论了 Java 中进行判断、决策和循环的另外几种语句。



6.1 switch 语句

Java 中的另一种条件语句是 `switch` 语句，该语句可使程序从多条执行路径中选择一条来执行。虽然使用多个 `if` 语句可以达到同样效果，但是 `switch` 的逻辑和可读性都要更强一些。

结构为：

```
switch ( expression )
{
    case value1 :
        statement-list1
    case value2 :
        statement-list2
    case value3 :
        statement-list3
    case ...
default:
    statement-default
}
```

`switch` 和 `case` 都是保留字。该代码将 `expression` 中的内容与 `case` 中的 `value` 值进行比对，如果它们相等则执行对应的 `statement` 代码。并且继续比较下面的 `case` 中的值，重复上述操作。最后执行 `default` 中的代码（`default` 可以不使用）。通常在每个 `case` 中的 `statement` 代码最后加入 `break` 语句，使 `switch` 语句一旦匹配到相应的 `case`，执行完对应的 `statement` 代码就跳出的操作。

例：

```
int a,b,c;
switch (option)
{
    case 'A':
        a++;
        break;
    case 'B':
        b++;
        break;
    case 'C':
        c++;
        break;
}
```

判断 `option` 中的字符与哪一个 `case` 中的字符相等，并使对应的值加一。加一完毕后就立即跳出 `switch` 语句。注意：`switch` 语句开始的表达式的运算结果必须是 `char`、`byte`、`short` 或 `int` 类型，而且每一个 `case` 语句中的表达式必须为常量。

6.2 条件运算符

条件运算符在某些方面与 `if-else` 语句相似。它是三元运算符。结构为：

```
condition ? expression1 : expression2
```

`condition` 是一个布尔条件，其后为用冒号分隔的两个表达式。如果 `condition` 为 `true`，则执行 `expression1` 的代码，否则执行 `expression2` 中的代码。它的特点是可读性较差，但是书写简便。

例：

```
larger = ((num1 > num2) ? num1 : num2);
```

该代码用于获得 `num1` 与 `num2` 中较大的数。

6.3 do 语句

与 while 语句相似，do 语句也重复执行循环体内语句，直到控制条件变为 false。但是 do 语句至少会循环一次。结构为：

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

先执行 statement 语句，再判断循环条件是否成立。

例：

```
int count = 0;
do
{
    count++;
    System.out.println (count);
} while (count < 5);
```

打印出数字 1-5。

6.4 for 语句

用于循环执行已经确切知道具体循环次数的情况。结构为：

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

initialization：初始化变量

condition:循环条件

increment:增量

例：

```
for (int count=1; count <= 5; count++)
    System.out.println (count);
```

输出数字 1-5。首先初始化变量 count，之后判断循环条件，成立的话就输出数字，最后执行增量操作，如此循环。

6.4.1 for-each 循环

用于处理一个迭代器对象中的各项元素。结构为：

```
for(Object obj : Iterator)
    statement;
```

Object 为 Iterator 迭代器中数据的类型。其语句等价于：

```
Object obj;
while(Iterator.hasNext())
{
    obj = Iterator.next();
    statement;
}
```

6.5 例题

1. 将下列代码转化为条件运算符

```
if (val <= 10)
    System.out.println("It is not greater than 10.");
else
    System.out.println("It is greater than 10.");
```

答: `val<=10?System.out.println("It is not greater than 10."):System.out.println("It is greater than 10.");`

代码片段如下, 执行后的输出结果是什么?

```
int i=11,j=5;
switch(i/j) {
    case 3:    j+=i;
    case 2:    j+=2;
    case 4:    j+=4; break;
    case 1:    j +=1;
}
System.out.pirnthn(j);
```

运行结果是:

A. 6 B. 7 C. 11 D. 22

第七章 面向对象设计

作者：鲍伟

7.1 软件开发过程

7.1.1 软件开发的四个基本过程

- 建立软件需求
- 软件设计
- 实现软件设计
- 软件测试

7.2 识别类和对象

7.2.1 识别类

识别可能的类的方法之一就是识别软件需求中所讨论的实体对象，所谓的实体可以简单理解为名词。

类代表了一组由类似行为的对象，例如学生类 `Student`，动物类 `Animal`。

识别类中的另一个关键是，应该把某一个事物表示成一个对象还是另一个对象的属性，这个要依赖于你的具体问题做出合理分析，没有标准的答案。

7.2.2 类的职责

每一个类代表一个具有某些行为的对象，对象的行为由这个类的方法来定义。类的行为所执行的动作体现了程序的功能，通常使用动词来给行为和方法命名。例如 `Student` 类中的 `study` 方法，可以表示学生学习的行为。

(前两小节的内容会在以后软件工程和面向对象的课程中深入讲解，这里有个概念就可以)

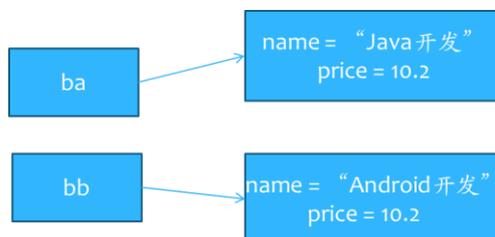
7.3 静态类成员

前面已经接触过 `Math` 类了，这个类的方法都是静态的。声明静态成员（一个类的成员包括类的方法和属性）使用 `static` 修饰符。

7.3.1 静态变量

讨论静态变量之前先理清两个概念：类（`class`）和实例（`instance`，有时也称为对象 `Object`）。类是生成实例的一个模板，实例是类的一个具体体现。类是一个抽象概念，反映实例的总体特征。

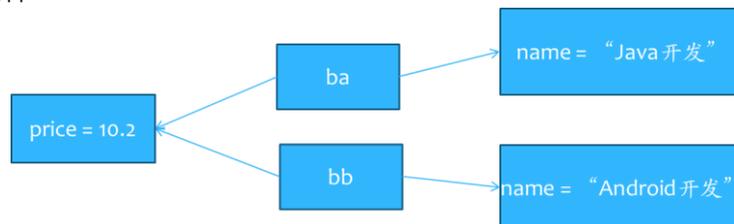
这里我们拿 `Book` 类举例，`Book` 类中有两个属性 `name` 和 `price`。现在有两本书 `ba`（`name = "Java 开发"`，`price = 10.2`）和 `bb`（`name = "Android 开发"`，`price = 10.2`），对于 `Book` 而言，它是一个类，是一个抽象的概念；但是那些具体的书 `ba` 和 `bb` 就是实例，`name` 和 `price` 两个变量的值在每个实例都有自己的变量副本，即每个对象都有自己不同的变量存储空间。对于这样的变量也被称为实例变量。



静态变量有时被称为类变量，特点就是静态变量对于所有的类的实例共享，对于类的所有实例只存在一个静态变量实体，因此修改静态变量的值将直接导致其他所有对象的值修改。接着上面的例子，如果我们把价格设为静态变量会如何（可以通过添加 `static` 关键字将实例变量变为静态变量）：

```
private static double price = 10.2;
```

那么结果会是这样：



可以看出实例变量在每个对象中都是有自己的副本的，尽管值可能是一样的；但是静态变量则是所有的类共同使用一个副本，这样也就很容易理解为什么修改一个对象的静态变量值其他对象的值也会随之改变。（大家可以将书本例 7.2 的类通过像编者这样的图形进行分析进一步理解。）

静态变量是在程序第一次引用的静态变量时分配的空间。注意，在方法的内部声明的局部变量不能具有静态属性。

7.3.2 静态方法

使用 `static` 修饰的方法成为静态方法，静态方法可以通过类名直接引用，例如 `Math` 中的方法。我们使用静态方法的原因是有些情况并不需要实例化对象来进行对应的操作。Java 程序的 `main` 方法必须使用 `static` 声明，这样是为了解释器执行 `main` 方法的时候不用实例化含 `main` 方法的类。

静态方法不能引用实例变量，因为静态方法不受实例化对象的控制，即：可以在没有实例化对象的时候也可以访问。在静态方法中引用实例变量可能存在两个问题：一是没有实例，二是不知道引用哪个实例，所以是不被允许的。

7.4 类间关系

软件系统中的类之间最基本的三种关系是依赖，聚合和继承。

7.4.1 依赖关系

依赖关系前面已经提到了很多的例子。它通常体现为一个类的方法调用另一个类的方法，这样就建立了“使用”关系（use-it）

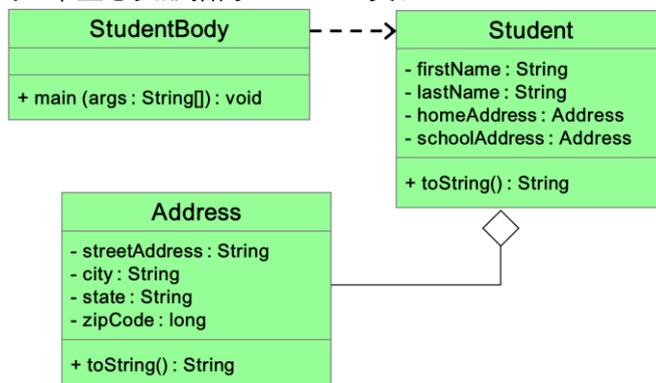
某些情况下，一个类依赖于本身，即一个类的对象于本类的其他对象交互。典型的代表就是 `String` 类中的 `concat` 方法。

原则上应该最小化类之间的依赖关系强度，类之间的依赖关系越小，软件系统代码修改所带来的影响和错误就会越少。

7.4.2 聚合关系

一个聚合对象由其他的对象组成，形成一种“has-a”关系。聚合关系是依赖关系的特殊类型，聚合体对象依赖组成它的各个对象。例如人由身体，头，胳膊等组成，这也是一种聚合关系。

聚合关系的 UML 图表示如下：这里以课本例 7.5-7.7 为例：`Address` 组成了 `Student` 类的一部分，则由 `Address` 类一个空心菱形指向 `Student` 类。



7.4.3 this 引用

`this` 引用可用于引用当前正在运行的对象。

例如：`Student` 有一个方法 `study`，`study` 中使用了 `this`。则：

```

xiaomin.study();
zhiku.study();
  
```

在第一个调用中，`this` 表示的就是 `xiaomin`，第二个调用中 `this` 表示的就是 `zhiku`。

`this` 的妙用在这里还得不到体现，但是在之后的程序编写中确实相当的关键，这里大家先记下这个概念：`this` 表示的是当前正在运行的对象。

在构造方法中我们可以使用 `this` 来避免对于含义相同的变量要给出不同命名以示区别的问题。

```

public Student(String name){
    this.name = name;
}
  
```

`this.name` 表示的就是这个类的 `name`，等于后面的 `name` 就是传进来的参数。

7.5 接口

Java 接口是一组常量和抽象方法的集合。所谓的抽象方法就是没有实现只有定义的方法。因为没有具体的实现，所以接口不能被实例化。

7.5.1 接口基本概念

抽象方法声明前面可以使用保留字 `abstract`，接口中的方法通常可以不加上 `abstract`，因为接口中的方法默认都是 `public` 和 `abstract` 修饰的。

定义一个接口的时候使用保留字 `interface`，例如：

```
public interface Animal{
    public void run();
    public void eat(Food f);
}
```

由于抽象方法是没有具体实现的，所以在方法声明之后紧接着一个分号 (;)，而不是像正常方法那样使用大花括号 ({})。因为花括号里的内容认为是具体实现，即使花括号中什么也没有写，也被认为是一种实现，即空实现。

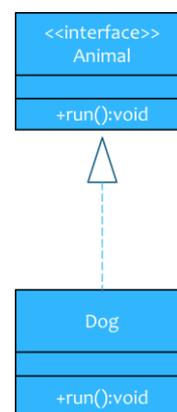
一个类可以实现一个接口，实现接口的类在类的头部要使用保留字 `implements` 再加上接口名，例如：

```
public class Dog implements Animal
```

如果一个类声明要实现某个接口，它就必须提供接口中所有方法的实现代码，否则编译器出错。当然，这并不意味着实现接口的类中只能有接口定义的方法，实现接口的类还可以有自定义的方法。

接口的作用并不同于具体类那样，具体类关注的是实现的细节，而接口的作用是提供一组规范，规定实现它的类应该具有怎样的行为，而不在意这个行为的具体实现。这种规范性也可以从必须实现所有的方法中体现出来。

一个类可以实现多个接口，声明时用逗号 (,) 将多个接口名隔开，此时需要实现所有接口中的所有方法。实现类和接口之间的 UML 表示如图：实现类一个虚线空心三角箭头指向接口。



7.5.2 Comparable 接口

`Comparable` 接口是 Java 类库中定义好的一个接口，里面只有一个方法 `compareTo`，这个方法需要一个对象作为参数，返回值为整型值。对于 `obj1.compareTo(obj2)`，一般定义 `obj1` 大于 `obj2` 返回正数，`obj1` 和 `obj2` 相等时返回 0，`obj1` 比 `obj2` 小时返回负值。

`String` 类就实现了这个方法，我们可以调用 `String` 的 `compareTo` 方法比较两个字符串，比较的依据就是 Unicode 编码的字典序。

7.5.3 Iterator 接口

`Iterator` 接口中定义的是迭代器中的方法，最重要的是 `hasNext` 和 `next` 方法，迭代器是处理遍历元素的对象。`hasNext` 判断是否有下一个需要遍历的对象，`next` 则返回下一个对象。

7.6 枚举型类

枚举型是一种特殊的类，枚举型的值是对象，枚举值是枚举类型的实例。既然是一种类，那么在原先的枚举类型中还可以增加属性和方法。

```
public enum Season
{
    winter ("December through February"),
    spring ("March through May"),
    summer ("June through August"),
    fall ("September through November");
    private String span;
    Season (String months)
    {
        span = months;
    }
    public String getSpan()
    {
        return span;
    }
}
```

7.7 方法设计

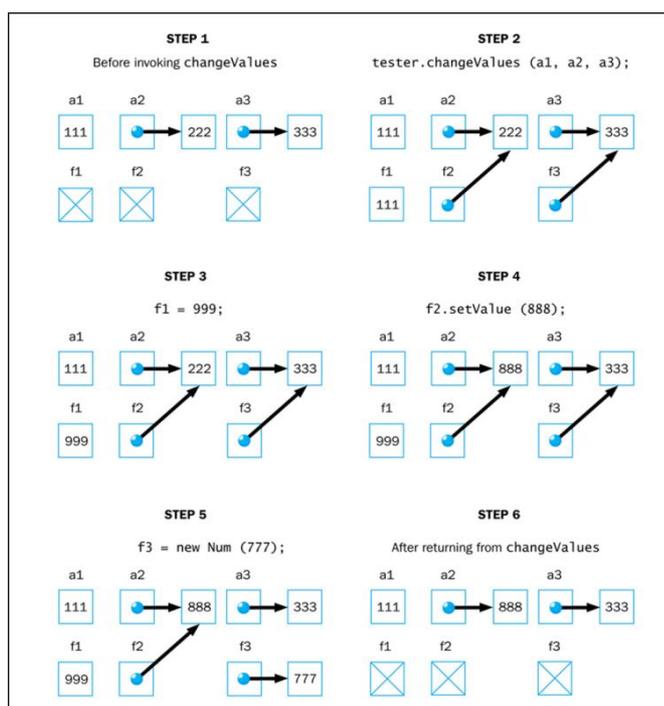
7.7.1 方法分解

有时候一个对象提供的服务可能过于复杂，这个时候一个方法不适合实现，可以将这个方法分解为多个方法，然后相互调用来实现这个复杂的服务。

7.7.2 方法参数的传递方式

Java 中采用的是传值的方式传递，参数传递实际上类似赋值语句，即实际参数保存值的副本赋给形式参数，对于基本数据类型，如 `int`，`double` 等，形参只是实参的一个副本，所以形参的改变不影响实参的值。但是对于对象而言，对象的保存值是对象的地址空间，所以赋值以后形参和实参指向同一个地址，形参和实参相互成为对方的别名。对于形参的改变会影响实参的值。

下图是课本例 7.15-17 的图解，第二步中是形参赋值之后的情况，第三步 `f1=999`，所以 `f1` 的值变为 999，但是 `a1` 的值并没有改变；第四步中，`f2.setValue(888)`，因为 `f2` 和 `a2` 是对象，赋值之后指向同一个地址，对于 `f2` 值的修改也修改了 `a2` 的值；第五步中刚开始的时候 `f3` 和 `a3` 指向同一个地址，`f3 = new Num(777)` 之后给 `f3` 赋了一个新的对象，所以 `f3` 指向了另一块地址。第六步中，方法调用结束，形参被释放，这个时候 `f3` 因为指向的是另一块地址，所以并没有影响到 `a3` 的值，但是 `a2` 因为在方法调用中地址里的值被修改，所以调用结束后值实际被修改了。



7.8 方法重载

方法名以及参数个数，参数类型和参数顺序成为方法的签名。只要方法的签名不同，两个方法就是不同的方法。所谓的方法重载就是在方法名相同的情况下使用不同的方法签名区别不同方法。

例如：

```
void write(char c)
void write(int n , char c)
void write(int n1 ,int n2)
```

这几个就是方法重载。在实际调用的时候会依据参数的类型调用具体的 write 方法。

注意，方法的返回值并不是方法签名的一部分，不能依据返回值重载。

构造方法也是可以重载的，这样可以使用多种方式创建对象。

7.9 测试

黑盒测试：将被测试的事物当成一个黑盒子，对于输入集合，通过黑盒得到的输出都是预期的效果，则任务这个黑盒是成功的。

白盒测试：也称玻璃测试，用于测试一个方法的内部结构和实现。

7.10 例题

1. 静态变量和实例变量的区别？

静态变量是类变量，对于所有的类的实例，所有实例只保存一个共享的地址，一旦一个类修改了静态变量，其他类都得到修改。静态变量是在第一次引用的时候分配空间。

实例变量，对于每个实例都保存一个副本，其他类的修改不影响本类变量的值。实例变量在每次实例化的时候分配空间。

2. 下列关于 static 关键字的说法，正确的是（ ）。

- A. 关键字 static final 在修饰 int 类型变量的时候，必须同时定义变量的初始值为 0。
- B. 关键字 static 放在类前面做修饰符时，表示该类中的所有方法都为静态方法。
- C. 关键字 static 只能用来修饰类里面的实例变量而不能用来修饰方法里的局部变量。
- D. 关键字 static 主要用来修饰 main 方法，表明 main 方法是类里面独一无二的。

答案：C

解析：

A: static final 是用来定义常量的，但是初始值是自己定义的，所以没有必要必须是 0。

B: 普通的类不能用 static 修饰，只有内部类才可以，但是内部静态类和普通类没什么区别，static 的作用只是让它可以不实例化外部类直接实例化自身，其他的定义和普通类没有区别，所以和静态方法没有任何联系。

C: 局部变量的作用范围只在方法之内，方法结束之后就会释放的，不能是静态的。

D: main 方法确实独一无二，但是这和 static 修饰没有任何关系。并不是被 static 修饰就独一无二。

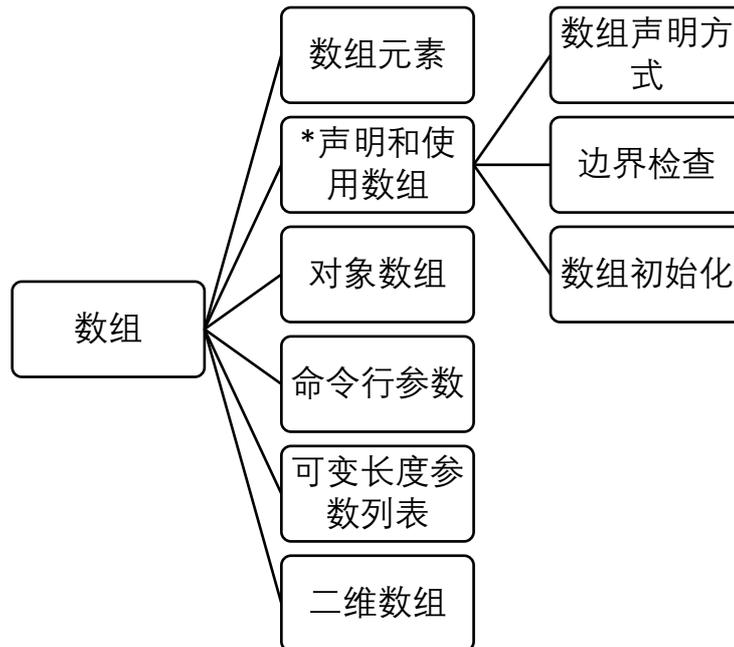
第八章 数组

作者：袁郭苑

经过前面对基本语法和类等相关概念的学习，大家已经掌握了一定的 java 编程能力与技巧。

现在让我们思考一个问题，编程离不开变量，科学合理的组织存储变量能够大大地简化我们的工作，那么，如何存储一系列相同类型的元素呢？能不能有快速调用一系列元素中某个确定的元素的方法呢？

带着自己的思考，让我们一起来学习第一个较为复杂的数据结构---数组（array）。它为我们提供了简单方便的声明及引用方法，是我们以后编程工作中组织数据的必不可少的好工具。



8.1 数组元素

8.1.1 数组

数组 (array) 是一种简单而功能强大的程序设计语言结构，用于分组和组织数据。(一个数组可以容纳多个可独立访问值的变量)

*一个数组中存储的多个元素应该是同一类型的，可以是原始类型 (byte, short, int, long, boolean, char, float, double)，也可以是引用类型。

*数组存储在一段连续的内存空间里，它的每个元素所占的空间大小是相同的，所以可以利用数组在内存空间的起始地址和每个元素所占空间的大小直接定位某元素在内存空间中的位置。

8.1.2 数组的索引

数组的每个值存在于数组中特定的、具有编号的位置。对应每个位置的编号称为索引或下标 (index)。

*索引从 0 开始，所以具有 N 个值的数组索引为 0 至 (N-1)。

访问数组值的方式：数组名[索引值]，例如访问 height 数组的第 9 个值表示为：height[8]

8.2 声明数组和使用数组

8.2.1 数组的声明

在 java 中，数组是对象，要建立数组，必须声明数组引用变量，然后可以用 new 运算符实例化数组，为数组分配内存空间。(java 中数组是必须实例化的对象)。

有 11 个整型值元素的数组 height 的声明：

```
int[] height = new int[11];
```

8.2.2 两种等价的数组声明方式

```
int[] grades;
```

```
int grades[];
```

P.S.第一种声明方式更具有一致性，也使得声明的数据类型更清楚。

8.2.3 边界检查

一旦一个数组被创建，它就有了一个固定的大小。

索引运算符自动执行边界检查 (是否满足 0 至 N-1)，从而保证引用数组操作的有效性。

如果越界，会抛出 ArrayIndexOutOfBoundsException 异常。

8.2.4 大小差一 (off-by-one) 错误

常见于 for 循环中边界条件控制不当，例如：

```
for (int index=0; index <= 100; index++)
```

```
    codes[index] = index*50 + epsilon;
```

其中会遍历到元素 index[100]

8.2.5 数组的 length 常量

length 是一个 public 常量，保存了数组的长度。例如 2.1 中声明的 height 数组的长度 height.length 的值为 11。

*数组的 length 值并非索引的最大值 (比索引最大值大一)

8.2.6 数组元素的遍历

除了一般形式的 for 循环（如 2.4 中的例子）之外，对于数组的遍历还有一种常用的形式，例如要遍历 int 类型的 scores 数组中的每个元素并依次打印出来，实现代码如下：

```
for(int score : scores)
    System.out.println(score);
```

P.S. 这种形式的遍历只适用于从数组的第一个元素（索引值为 0）开始的遍历；而且不能被用来设置数组的值

8.2.7 数组初始化

使用初始值表可以用于实例化一个数组对象，这时不再需要用 new 运算符创建数组对象。以下为一个示例，创建了有 8 个元素的数组，并用指定值进行初始化：

```
int[] scores = {87, 98, 85, 87, 84, 96, 81, 83};
*初始值表中每个值的类型必须匹配数组元素的类型
```

8.3 对象数组

8.3.1 存储基本类型数据数组 vs 存储对象引用数组

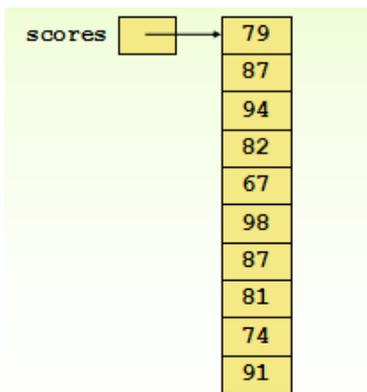


图 1

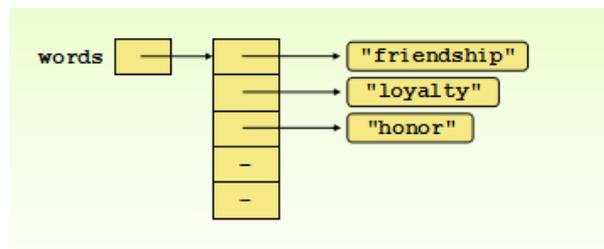


图 2

图 1 为存储基本类型数据的数组，数组的名称是一个对象引用变量；图 2 为存储对象引用的数组，数组的名称是一个对象引用变量，数组里存储着对象的引用。

P.S. 实例化对象数组只是为保存对象引用而预留空间，对每个数组元素所代表的对象必须分别实例化。

*一个对象数组实际上是一个对象引用数组

*字符串对象可以表示为字符串常量，所以 String 类型的对象数组可以用初始值表对数组进行初始化，初始值表中的每个 String 对象就是一个字符串常量，代码示例如下：

```
String[] verbs = {"play", "work", "eat", "sleep"};
```

8.4 命令行参数

命令行参数是给程序提供输入信息的一种方式，调用解释器时关于命令行的信息将存放在 args 数组中供程序使用。

命令行参数存储在 String 对象的数组（通常称为 args）中，并将传递给 main 方法。

8.5 可变长度参数列表

一个 java 方法可定义为参数个数可变的方法，示例代码如下：

```
public double average (int ... list)
{
    // whatever
}
```

在以上代码的参数列表中，省略号表示该方法接收的参数个数是可变的，int 表示元素的类型，所接受的参数将自动存入数组 list。方法内按正常的数组处理参数。

8.6 二维数组

8.6.1 二维数组

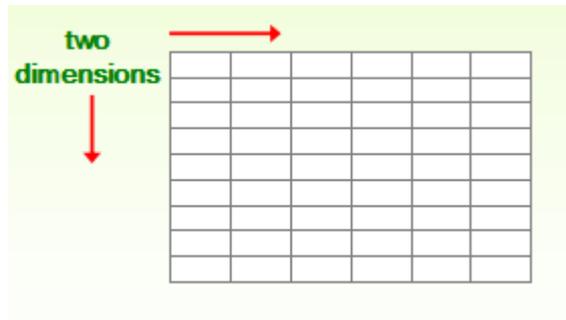


图 3

图 3 所示，二维数组有二维的值，常视为由行和列构成的表。

二维数组的声明示例如下，12 行 50 列：

```
int[][] table = new int[12][50];
```

二维数组的引用示例如下，引用第 3 行第 6 列元素：

```
int value;
value = table[3][6];
```

存在一行里的元素，可以仅用一个索引来引用，示例如下，引用第一行元素：

```
int[] value;
value = table[1];
```

所以从某种意义上讲，二维数组是数组的数组（an array of arrays）

8.7 例题解析

1.

关于下面代码 `int[] x=new int[25];` 描述正确的是（C）

- A. `x[25]` 存放了数据 “\0”。
- B. `x[24]` 存放了数据 “\0”。
- C. 若访问 `x[25]`，程序将抛出异常。
- D. `x[1]` 访问此数组的第一个元素。

解答：new 运算符实例化数组，为数组分配内存空间。对于 A，x[25] 存在访问越界问题，因为 x 数组的最后一个元素是 x[24]，所以访问 x[25] 将抛出 `ArrayIndexOutOfBoundsException` 异常，故 A 错误，C 正确，另外元素所在的内存空间没有被赋特定的初值，其存放的应该是“0”，而不是“\0”，所以 B 错误，x[1] 访问数组的第 2 个元素，所以 D 错误。

2.

读程序题：（写出以下程序的运行结果）

```
public class ReadTwo {
    public static final int N=4;
    public static void main(String[] args) {
        int[][] m = new int[N][N];
        Calc(m, N);
        Print(m);
    }
    public static void Calc(int[][] m, int num) {
        int k = N;
        for (int i = 0; i <= num / 2; i++) {
            for (int j = i; j < num - i; j++)
                m[i][j] = k++;
            for (int j = i + 1; j < num - i; j++)
                m[j][num - i - 1] = k++;
            for (int j = num - i - 2; j >= i; j--)
                m[num - i - 1][j] = k++;
            for (int j = num - i - 2; j > i; j--)
                m[j][i] = k++;
        }
    }
    public static void Print(int[][] m) {
        for (int i = 0; i < m.length; i++) {
            for (int j = 0; j < m[i].length; j++)
                System.out.print(m[i][j]+"\\t");
            System.out.println();
        }
    }
}
```

解答：这道题目其实更偏重于对 for 循环条件和逻辑的考察，但是借用二维数组的数据结构来表达，我们要理解一般的（即规则的二维数组）二维数组 a 的 `a.length` 表示的是 a 的行数，`a[0].length` 表示的是 a 的列数，再根据循环条件进行分析，不难得出程序运行结果为：

4	5	6	7
15	16	17	8
14	19	18	9
13	12	11	10