

1 基本概念

进入Java的世界

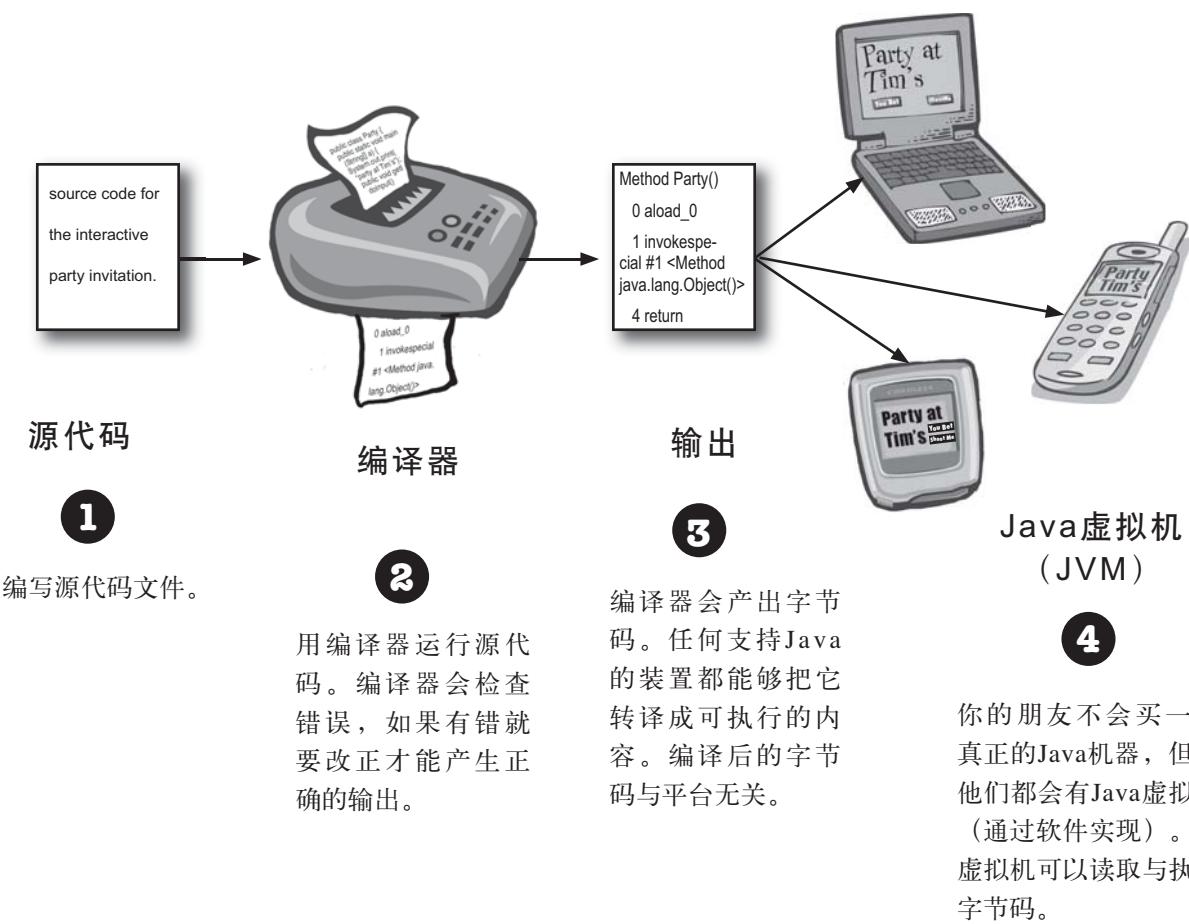


Java将带你进入新领域。它从一开始就以友好的语法、面向对象、内存管理和最棒的跨平台可移植性来吸引程序员。写一次就可以在所有地方执行（write-once/run-anywhere）的特性简直是迷死人了。许多人在投入后才发现有bug要除、功能限制很大、最要命的是运行起来超慢！不过这都是很久以前的事情了。如果你现在才刚刚开始接触Java，那你还真幸运。现在的Java可是又快又有威力。



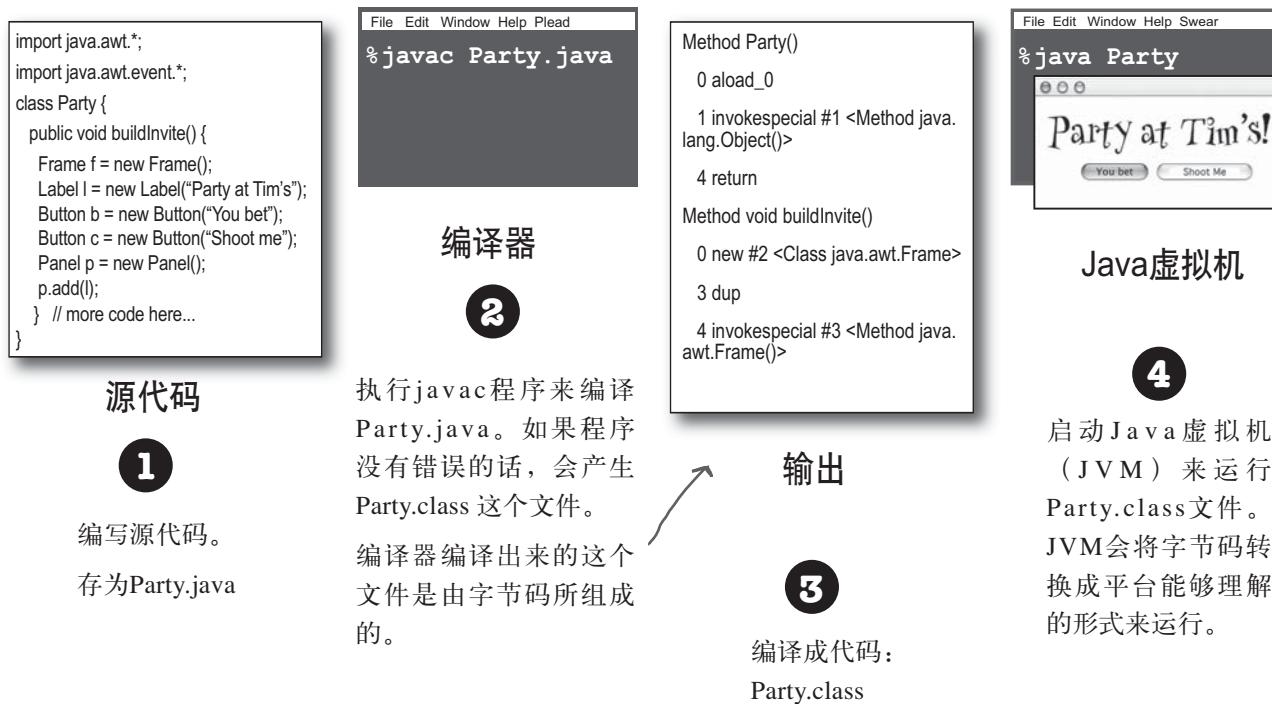
Java的工作方式

它的目标是要让你写出一个应用程序（在此例中是一个交互式派对邀请函系统）且能够在你的朋友所拥有的任何设备上执行。



你要做的事

你会编写源代码文件，用javac编译程序把文件进行编译，然后在某个Java虚拟机上执行编译过的字节码。



(注意：这一页不是练习题，等一下就会让你编写程序，但现在不用，只是要让你知道来龙去脉。)

Java 简史

在 Java 标准函数库中的类 (class) 的数量



Java 简史

Java 1.02
250 个类
龟速
有可爱的logo和名称，非常有趣，但是bug很多，其中applet是重点。

Java 1.1
500 个类
狗速
功能更强、更好用。开始受到欢迎。比较适合开发图形界面。

Java 2
(版本 1.2~1.4)
2300 个类
马速
有时可以达到平台原始(native)速度。可以用来书写正规的企业级应用程序或移动应用程序。有3种版本：Micro Edition (J2ME)、Standard Edition(J2SE)以及 Enterprise Edition (J2EE)。

Java 5.0
(版本1.5及以上)
3500 个类
机器人等级的威力，更容易开发

除了新增数以千计的类之外，Java 5.0 (又称为Tiger)还对语言本身作了许多重大改变，使得它在理论上更容易使用，并含有其他语言中很受欢迎的功能。



看，写Java程序就是这么简单！

猜猜看每一行程序在做什么？（答案在下一页）

```
int size = 27;  
String name = "Fido";  
Dog myDog = new Dog(name, size);  
x = size - 5;  
if (x < 15) myDog.bark(8);  
  
while (x > 3) {  
    myDog.play();  
}  
  
int[] numList = {2,4,6,8};  
System.out.print("Hello");  
System.out.print("Dog: " + name);  
String num = "8";  
int z = Integer.parseInt(num);  
  
try {  
    readTheFile("myFile.txt");  
}  
catch(FileNotFoundException ex) {  
    System.out.print("File not found.");  
}
```

声明一个 integer 类型，名称为 size 的变量并赋值 27

问： 我曾经看过Java 2和Java 5.0，是否有Java 3和4呢？为什么2没有小数点？

最原始的Java版本是Java 1.02（第一次出版的版本），1.02到1.1版都算是“Java”。1.2、1.3和1.4版都叫做“Java 2”。从1.5版开始叫做“Java 5.0”。你偶尔也会看到“Java 5”或“Tiger（开发代号）”的用法，我不知道下一个版本该叫什么。

答： 搞销售的乐趣就在于此，虽然Java的实际版本是从1.1推进到1.2，但因变化幅度大，所以销售部门决定用全新的命名，于是命名为Java 2。Java 3或4从来没有出现过。到了1.5版时，销售部门又认为进步幅度太大（大部分的程序员都同意这么说）而需要全新的名称，所以又冒出一个Java 5.0，用5对应到1.5。



看，写 Java 程序就是这么简单！

```
int size = 27;  
String name = "Fido";  
Dog myDog = new Dog(name, size);  
x = size - 5;  
if (x < 15) myDog.bark(8);  
  
while (x > 3) {  
    myDog.play();  
}  
  
int[] numList = {2,4,6,8};  
System.out.print("Hello");  
System.out.print("Dog: " + name);  
String num = "8";  
int z = Integer.parseInt(num);  
  
try {  
    readTheFile("myFile.txt");  
}  
catch(FileNotFoundException ex) {  
    System.out.print("File not found.");  
}
```

看不懂也无所谓！

后面会有很详细的解说（约40页）。如果你学过类似的语言就会发现这些东西很简单。如果没有的话，接下来就会让你知道……

声明一个integer类型、名称为size的变量并赋初始值为27

声明名称为name的字符串，值为Fido

用name与size声明一个名称为myDog的Dog变量

把size值减5的结果赋给变量x

如果x的值小于15，让狗吠8次

当x的值大于3就持续执行循环

让狗做些执行 play 动作（不管那对狗有什么意义）

{ } 里面就是循环的内容

声明有4个元素的整型数组

把 "Hello" 输出到屏幕上

把 "Dog: " 输出到屏幕上，后面跟着狗的名字 (Fido)

声明字符串变量num并赋值为 8

将字符串 "8" 转换成整数数字 8

试列出可能会出异常状况的指令

读取myFile.txt这个文件

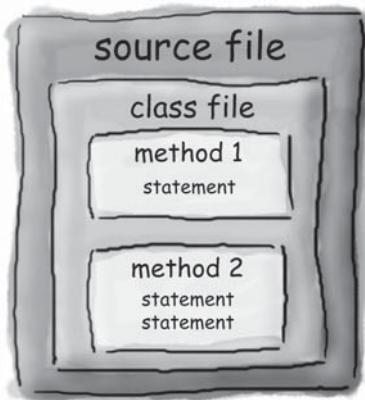
{ }中的指令被视为一体

捕获万一发生的找不到文件异常状况

万一找不到文件就说找不到

{ } 里面是异常的处理程序

Java的程序结构



类存于源文件里面

方法 存于 类 中

语句 (statement) 存于方法中

什么是源文件?

源文件(扩展名为 .java)带有类的定义。类用来表示程序的一个组件，小程序或许只会有一个类。类的内容必须包在花括号里面。

```
public class Dog {  
}  
类
```

什么是类?

类中带有一个或多个方法。在 Dog这个类中，bark方法带有如何“汪汪”的指令。方法必须在类的内部声明。

```
public class Dog {  
    void bark() {  
    }  
}  
方法
```

什么是方法?

在方法的花括号中编写方法应该执行的指令。方法代码是由一组语句所组成，你可以把方法想象成是一个函数或过程。

```
public class Dog {  
    void bark() {  
        statement1;  
        statement2;  
    }  
}  
语句
```

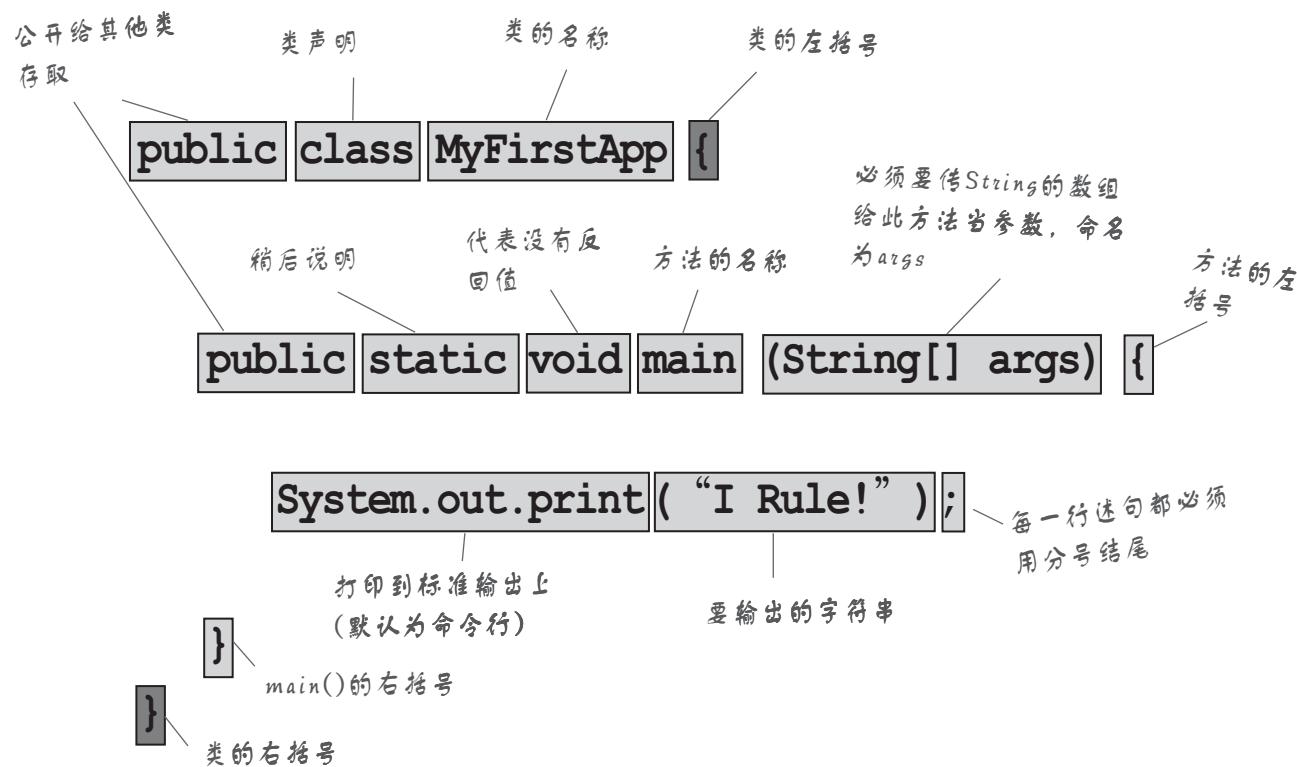
Java的类

剖析类

当Java虚拟机启动执行时，它会寻找你在命令列所指定的类。然后它会锁定像下面这样一个特定的方法：

```
public static void main (String[] args) {  
    // 程序代码写在这里  
}
```

接着Java虚拟机就会执行main方法在花括号间的函数所有指令。每个Java程序最少都会有一个类以及一个main()。每个应用程序只有一个main()函数。



现在还不需要把这些东西背下来，这一章只是热身活动。

编写带有main()的类

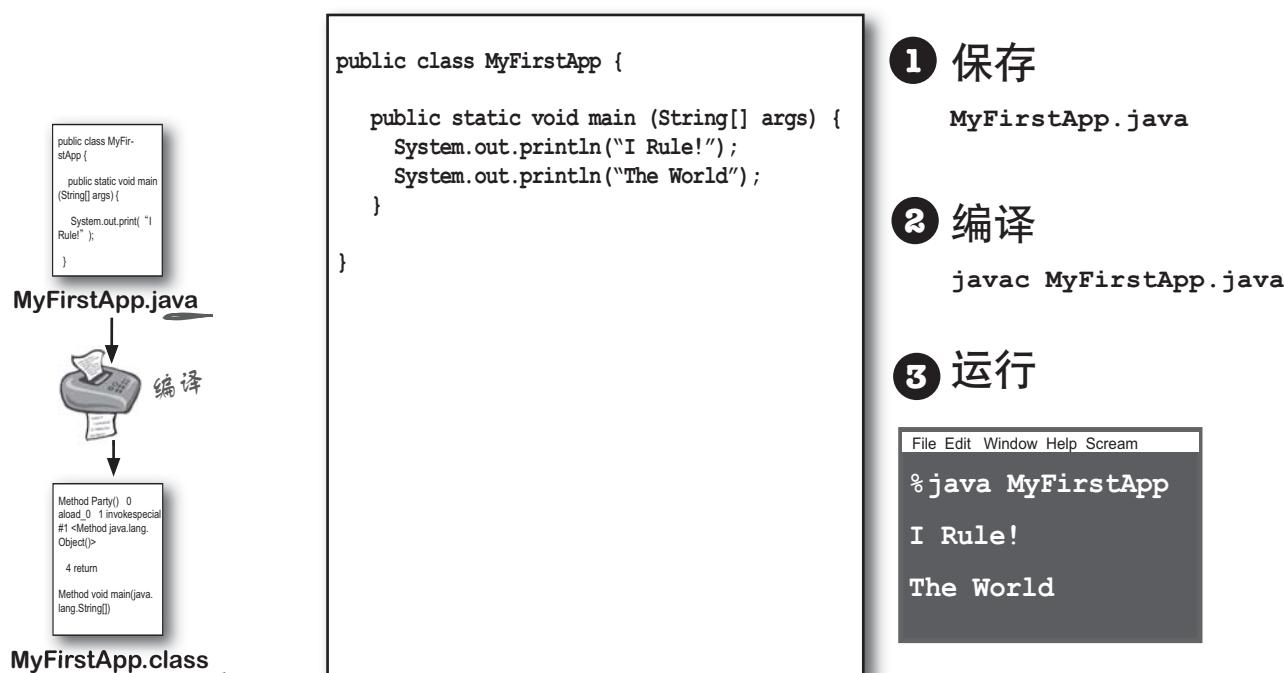
在Java中的所有东西都会属于某个类。你会建立源文件(扩展名为.java),然后将它编译成新的类文件(扩展名为.class)。真正被执行的是类。

要执行程序就代表要命令Java虚拟机(JVM)去“加载Hello这个类,开始执行它的main(),然后一直运行到main的所有程序代码结束为止”。

在第2章中我们会更深入地探讨类的细节,但是现在你只要注意到如何编写与执行Java程序就行了。而这些都与main()有关。

main()就是程序的起点。

不管你的程序有多大(也可以说不管有多少个类),一定都会有一个main()来作为程序的起点。



你能在 main() 中做什么？

你一旦进入main()或其他的方法中，就可以找到乐子。你可以作出任何正常的行为来让编译器（compiler）忙。

你的程序代码可以让Java虚拟机去：



语法
乐趣多

1 做某件事

声明、设定、调用方法等普通语句。

```
int x = 3;
String name = "Dirk";
x = x * 17;
System.out.print("x is " + x);
double d = Math.random();
// 这是注释行
```

2 反复做某件事 for 与 while 的循环 (loop)

```
while (x > 12) {
    x = x - 1;
}

for (int x = 0; x < 10; x = x + 1) {
    System.out.print("x is now " + x);
}
```

3 在适当条件下做某件事

if/else 的条件分支测试

```
if (x == 10) {
    System.out.print("x must be 10");
} else {
    System.out.print("x isn't 10");
}
if ((x < 3) & (name.equals("Dirk"))) {
    System.out.println("Gently");
}
System.out.print("this line runs no matter what");
```

★ 语句是以分号结束。

```
x = x + 1;
```

★ 以两条斜线开始的行是注释。

```
// 我是注释
```

★ 空格符通常无关紧要。

```
x      =      3      ;
```

★ 用名称与类型 (type) 来声明变量（第3章会讨论类型）。

```
int weight;
// 类型: int, 名称: weight
```

★ 类型与方法都必须定义在花括号中。

```
public void go() {
    // 程序代码放在这里
}
```



```
while (moreBalls == true) {
    keepJuggling();
}
```

重复再重复，循环再循环……

Java有3种循环结构：while循环、do-while循环和for循环。稍后你会看到详细的说明，但是先让我们来看一下while循环是怎么工作的。

这个语法简单到你可能已经睡着了。只要while条件为 true，循环块中的程序代码就会一直重复执行。程序代码块是由一对花括号所规范的，所以要重复的区段必须摆在括号中。

循环的关键在于条件测试（conditional test）。在Java中，条件测试的结果是boolean值——不是true就是false。

例如说“如果含笑半步癫比一日丧命散还好吃我就嗑给你看”是个有效的boolean测试，但“如果香蕉我就说芭蕉”并不是个条件测试。条件判断式必须要能够求出真伪值。

简单的boolean测试

你可以用比较运算符（comparison operator）来执行简单的boolean值测试：

< (小于)

> (大于)

== (等于)

注意：赋值运算符（一个等号）与等号运算符（两个等号）并不一样。许多程序员都会不小心犯这个错，但我相信你不会。

```
int x = 4; //给x赋值为4
while (x > 3) {
    // 循环会运行是因为
    // x 大于 3
    x = x - 1; //避免无限循环
}
int z = 27; //
while (z == 17) {
    // 循环不会运行
    // 因为z不等于17
}
```

there are no
Dumb Questions

问：为何所有的东西都得包含在类中？

答：因为Java是面向对象的语言，它不像是以前的程序语言那样。第2章会说明类是对象的蓝图，而Java中的绝大多数东西都是对象。

问：每个类都需要加上一个main()吗？

答：程序不是这样写的。一位大名鼎鼎的程序大师已经解释过一个程序只要一个main来作为运行。

问：其他程序语言可以直接用整数类型测试，我也可以像下面这么做吗：

```
int x = 1;  
while (x) { }
```

答：不行，Java中的integer与boolean两种类型并不兼容。你只能用下面这样的boolean变量来测试：

```
boolean isHot = true;  
while(isHot) { }
```

while循环的范例：

```
public class Loopy {  
    public static void main (String[] args) {  
        int x = 1;  
        System.out.println("Before the Loop");  
        while (x < 4) {  
            System.out.println("In the loop");  
            System.out.println("Value of x is " + x);  
            x = x + 1;  
        }  
        System.out.println("This is after the loop");  
    }  
}  
  
% java Loopy  
Before the Loop  
In the loop  
Value of x is 1  
In the loop  
Value of x is 2  
In the loop  
Value of x is 3  
This is after the loop
```

这是输出

要点

- 语句以分号结束。
- 程序块以{}划出范围。
- 用名称与类型声明变量。
- 等号是赋值运算符。
- 两个等号用来当等式等号运算符。
- 只要条件测试结果为真，while循环就会一直执行块内的程序。
- 把boolean测试放在括号中：

```
while (x == 4) { }
```

条件分支

在Java中if与while循环都是boolean测试，但语义从“只要不下雨就持续……”改成“如果不下雨就……”。

```
class IfTest {
    public static void main (String[] args) {
        int x = 3;
        if (x == 3) {
            System.out.println("x must be 3");
        }
        System.out.println("This runs no matter what");
    }
}

% java IfTest
x must be 3
This runs no matter what
```

← 这是输出

上面的程序只会在x等于3这个条件为真的时候才会列出“x must be 3”。而第二行无论如何都会列出。

我们可以将程序加上else条件，因此可以指定“如果下雨就撑雨伞，不然的话就戴墨镜”。

```
class IfTest2 {
    public static void main (String[] args) {
        int x = 2;
        if (x == 3) {
            System.out.println("x must be 3");
        } else {
            System.out.println("x is NOT 3");
        }
        System.out.println("This runs no matter what");
    }
}
```

```
% java IfTest2
x is NOT 3
This runs no matter what
```

修改后的输出

System.out.print 与

System.out.println

如果仔细观察的话，你会注意到我们将print改成了println。

差别在哪里？

println会在最后面插入换行，若你想让后续的输出以新的一行开始，可以使用println，若是使用print则后续的输出还是会在同一行。

Sharpen your pencil



若程序输出如下：

```
% java DooBee
DooBeeDooBeeDo
```

请填入程序空格部分：

```
public class DooBee {
    public static void main (String[] args) {
        int x = 1;
        while (x < _____) {
            System.out._____("Doo");
            System.out._____("Bee");
            x = x + 1;
        }
        if (x == _____) {
            System.out.print("Do");
        }
    }
}
```

真正的应用

设计真正的应用程序

让我们将你已经学习到的Java技能发挥到真正有用的应用程序上。我们需要一个带有main()的类、一个int与String变量、一个while循环和一个if测试。稍加整理你就能立即创建出有用的程序。但在你偷看示例程序代码之前，先自己想想看你会怎样写一个程序从99数到0。下面是数啤酒瓶童谣的程序：



```
public class BeerSong {  
    public static void main (String[] args) {  
        int beerNum = 99;  
        String word = "bottles";  
  
        while (beerNum > 0) {  
  
            if (beerNum == 1) {  
                word = "bottle"; // 单数的瓶子  
            }  
  
            System.out.println(beerNum + " " + word + " of beer on the wall");  
            System.out.println(beerNum + " " + word + " of beer.");  
            System.out.println("Take one down.");  
            System.out.println("Pass it around.");  
            beerNum = beerNum - 1;  
  
            if (beerNum > 0) {  
                System.out.println(beerNum + " " + word + " of beer on the wall");  
            } else {  
                System.out.println("No more bottles of beer on the wall");  
            } //else结束  
        } //while循环结束  
    } //main方法结束  
} //class结束
```

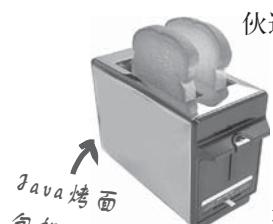
虽然程序可以编译与运行，
但它的输出有点不美观，你
可以试试看改漂亮一点。

阿强的周一早晨

平时阿强的闹钟设定在8:30，但在狂野的周末他会把早上闹钟的声音按掉。此时有Java功能的设备会开始起作用。

首先，闹钟会送出一个信息给咖啡机说：“嗨，这位老兄又赖床了，12分钟后再煮咖啡”。

咖啡机会送出一个信号给烤面包机说：“先不要烤，这家伙还在睡”。



之后闹钟还会送出信号给阿强的手机：“9:00时打给阿强告诉他我们晚一点才会准备好”。

最后，闹钟会送个信号给大头（大头是只狗）的无线颈圈表示去捡报纸回来，但是别想出去散步。

几分钟后闹钟又响了，阿强还是把它按掉，而设备们又开始交谈。第三次阿强还是想要把闹钟按掉，但是闹钟先发制人的送出一个信息给大头的颈圈表示要它跳上床叫人。

终于把阿强叫醒了，他心想幸好Java设备有发挥作用，算是没有白跑一趟光华商场。

面包已经烤好了。



咖啡热腾腾的。



报纸也拿进来了。

又是个美好的早晨。你可以用Java、网络以及Jini技术来装置一栋Java-Enabled的房子。但是要小心所谓的“即插即用”技术（事实上即插即用代表着插入之后随即使用技术支持客服专线）或是“可移植性”平台。阿强的老姐也使用了这些设备，但是效果很差，甚至还具有危险。他的狗在生前也是这么认为……

这个故事有可能是真的吗？是也不是。虽然有各种版本的Java用在PDA、移动电话、智能卡等装置上，但你目前还不太可能找到有Java的烤面包机与狗项圈。然而还是可以通过有Java的计算机等接口来控制其他设备。这就是称为Jini surrogate的架构。你还是可以这样设计梦想豪宅。



开始编写程序



试试看我的高级多功能超级专家术语学习机，它让你说起话来比电视购物频道主持人还要厉害！

```
public class PhraseOMatic {  
    public static void main (String[] args) {  
  
        1 // 你可以随意的加上其他术语  
        String[] wordListOne = {"24/7", "multi-Tier", "30,000 foot", "B-to-B", "win-win", "front-end", "web-based", "pervasive", "smart", "six-sigma", "critical-path", "dynamic"};  
  
        String[] wordListTwo = {"empowered", "sticky", "value-added", "oriented", "centric", "distributed", "clustered", "branded", "outside-the-box", "positioned", "networked", "focused", "leveraged", "aligned", "targeted", "shared", "cooperative", "accelerated"};  
  
        String[] wordListThree = {"process", "tipping-point", "solution", "architecture", "core competency", "strategy", "mindshare", "portal", "space", "vision", "paradigm", "mission"};  
  
        2 // 计算每一组有多少个名词术语  
        int oneLength = wordListOne.length;  
        int twoLength = wordListTwo.length;  
        int threeLength = wordListThree.length;  
  
        3 // 产生随机数字  
        int rand1 = (int) (Math.random() * oneLength);  
        int rand2 = (int) (Math.random() * twoLength);  
        int rand3 = (int) (Math.random() * threeLength);  
  
        4 // 组合出专家术语  
        String phrase = wordListOne[rand1] + " " +  
                      wordListTwo[rand2] + " " + wordListThree[rand3];  
  
        5 // 输出  
        System.out.println("What we need is a " + phrase);  
    }  
}
```

注意：当你自己输入这个程序时，不要在字符串中间换行（也就是“”符号之间），不然的话程序无法通过编译程序。

专家术语学习机

它是如何工作的

简单地讲，它是由3组单字随机挑出来排列组合输出。暂时不用担心你看不懂某一行程序在做什么，毕竟我们才刚开始而已。

- 1.** 第一个步骤是创建出3个String的数组，也就是保存术语的容器。数组的声明和创建是很简单的，下面是一个例子：

```
String[] pets = {"Fido", "Zeus", "Bin"};
```

每个元素放在引号中并彼此间以逗号分开。

- 2.** 为了要在每个数组中能够随机地挑出一个单字，我们得知道每个数组的大小。如果某个数组有14个单字，则我们需要介于0~13的随机数（Java的数组是零基的，第一个元素的索引是0，第二个是1，在有14个元素的数组中最后一个的索引是13）。你可以向数组查询它的长度：

```
int x = pets.length;
```

执行以后 x 的值为 3。

我们可以输出下面
这些言不及义的术
语……

pervasive targeted
process

dynamic outside-
the-box tipping-
point

smart distributed
core competency

24/7 empowered
mindshare

30,000 foot win-win
vision

six-sigma net-
worked portal

- 3.** 我们需要3个随机数。Java本身有一组立即可用的数学方法（可以把它们当作函数）。random()这个方法会返回介于0与1之间的值，所以我们需要将此值乘以数组的元素数量（数组的大小），然后取整数值（第4章会讨论）。如果要对任何浮点数取整数值也是用这样的方法转换数据类型：

```
int x = (int) 24.6;
```

- 4.** 现在我们可以创建专用的术语。选出3个字然后使用“+”这个运算符将字符串对象连接在一起。使用索引数字可以将数组中的元素提取出来：

```
String s = pets[0]; // "Fido"  
s = s + " " + "is a dog"; // "Fido is a dog"
```

- 5.** 最后，我们将结果输出到命令列上，你就可以引用这些术语说出听起来很厉害但是完全没有意义的句子。

编译器与Java虚拟机



Java 虚拟机

今晚的话题：

编译器与JVM争辩谁比较重要？

编译器

什么？你开玩笑吧？这位大婶，我可是 Java 啊。只有我才能让程序运行起来。
你只是产生文件而已。做个文件有什么了不起的，没有我，文件没有用！

请你放尊重点，不然我要叫了。

还有，你得理不饶人，每天老是警告人，
小小一点语法错误也不放水……

对不起，没有我你能运行什么？Java 会设计成这样是有原因的。如果 Java 只是个直译语言，要一边运行一边解译纯文字的程序，我就不相信你能够运行多快！

抗议啦，我又没有说你一点用处都没有。
但说真的，我根本搞不懂你在做什么。
程序员可以直接编写二进制代码给我运行，那你就失业啦，哇哈哈哈……哇哈哈
哈……咳……哇哈哈……

我实在懒得理你。没错，虽然说只要是合格的二进制代码就可以运行，不一定要编译器编译出来的，但实际上不会有人傻成这样的。让程序员直接写出二进制代码就好像要组装计算机的人自己得作出 CPU 一样。还有，你可不可以不要笑得那么难听？

先不管它的笑声问题。你还是没有回答我
你到底有什么用处？

Java 虚拟机

编译器

还记得Java是个强类型的语言吗，这代表我不能容许变量保存类型的数据。这是很关键的类型安全性功能，我能够让大部分的错误在到你那边之前就被抓到。还有……

又不是全部抓光光！我还是会因为遇到将错误类型的数据塞进数组中而不得不抛出异常，并且……

没礼貌，别打断我说话……，是有些数据类型的错误会在运行时发生，但这也是为了要容许动态绑定这样的功能。Java可以在执行期引用连程序员也没有预期会碰到的类型，所以我得留一些运用性。我的工作就是要确保铁定不能跑的东西不会过关。通常我会抓得到错误，例如说把文字字符串除以某个数字这种问题就会被我发现。

OK，当然。但是存取权限的安全问题呢？
还不是靠我把关，而你只不过是作些标点符号的检查罢了。还真谢谢你把这些问题留给我呢。

对不起，大家都知道我才是安全的第一线。我刚刚说的数据类型错误如果没有处理好可是一个漏洞呢。像是违反调用private方法的程序等也是由我检查的。我能够防止人们动到不可以碰的程序代码与其他类的重要数据。如果要把我的功能说完可能要说到天亮。

随你怎么说。我也得做相同的事情，确保不会有人在执行前修改二进制代码。

是啦，如果没有我挡住上述的问题，你老早就挂掉了。没时间了，下回再说吧。

OK，等一下要不要去吃宵夜？

习题



排排看

小朋友，右边是原本写好的程序，但是被饼干大怪兽给弄乱了，英勇的你是不是可以把程序排回原状来产生像下面这样的输出呢？注意到有些括号被吃掉了，请爸爸妈妈一起帮忙补起来！

```
if (x == 1) {  
    System.out.print("d");  
    x = x - 1;  
}
```

```
if (x == 2) {  
    System.out.print("b c");  
}
```

```
class Shuffle1 {  
    public static void main(String [] args) {
```

```
        if (x > 2) {  
            System.out.print("a");  
        }
```

```
        int x = 3;
```

```
        x = x - 1;  
        System.out.print("-");
```

```
    while (x > 0) {
```

输出：

```
File Edit Window Help Sleep  
% java Shuffle1  
a-b c-d
```



我是编译器



这一页的Java程序代码代表一份完整的源文件。你的任务是要扮演编译器角色并判断哪个程序可以编译过关。
如果有问题，哪里需要修改？

B

```
public static void main(String [] args) {  
    int x = 5;  
    while ( x > 1 ) {  
        x = x - 1;  
        if ( x < 3) {  
            System.out.println("small x");  
        }  
    }  
}
```

A

```
class Exerciselb {  
    public static void main(String [] args) {  
        int x = 1;  
        while ( x < 10 ) {  
            if ( x > 3) {  
                System.out.println("big x");  
            }  
        }  
    }  
}
```

C

```
class Exerciselb {  
    int x = 5;  
    while ( x > 1 ) {  
        x = x - 1;  
        if ( x < 3) {  
            System.out.println("small x");  
        }  
    }  
}
```

程序谜题



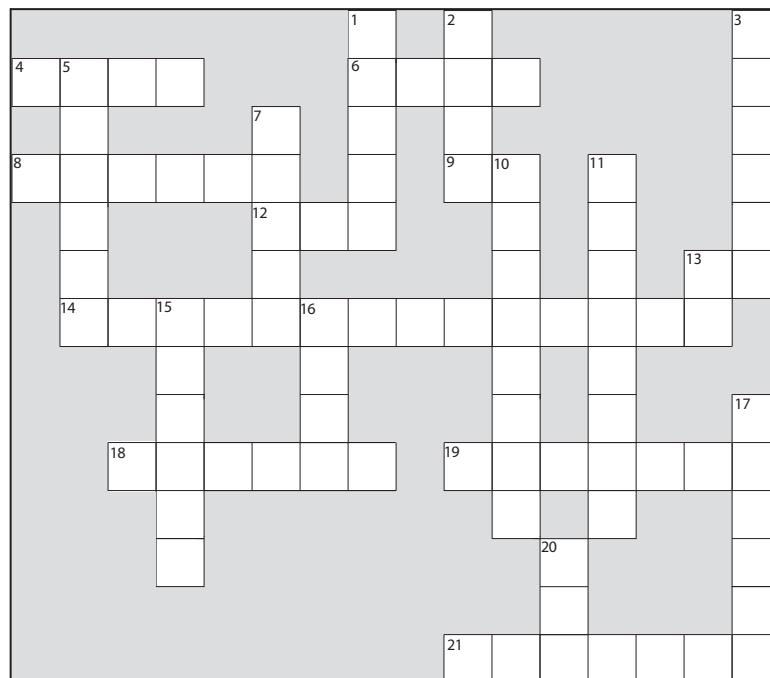
JavaCross 7.0

找点事情让你的大脑动一动。

这是个字谜，几乎所有的单字都在这一章提过。为了提高你的注意力，我们偷偷地放了一些非Java的高科技词汇。

横排提示

4. Command-line invoker
6. Back again?
8. Can't go both ways
9. Acronym for your laptop's power
12. number variable type
13. Acronym for a chip
14. Say something
18. Quite a crew of characters
19. Announce a new class or method
21. What's a prompt good for?



竖排提示

1. Not an integer (or _____ your boat)
2. Come back empty-handed
3. Open house
5. 'Things' holders
7. Until attitudes improve
10. Source code consumer
11. Can't pin it down
13. Dept.of LAN jockeys
15. Shocking modifier
16. Just gotta have one
17. How to get things done
20. Bytecode consumer

(字谜的问题部分保留原汁原味的英文，请自己动手查字典！)



连连看

别翻脸，我不会再提饼干大怪兽了，请你也不要找爸妈帮你作答好吗？下面有段程序代码不见了，你的工作是要把左边的程序代码填入消失的段落以连接到右边的输出。不一定每个输出都会对应到某个程序代码段，有的输出可能会用到两次以上。

```
class Test {
    public static void main(String [] args) {
        int x = 0;
        int y = 0;
        while ( x < 5 ) {
            
            System.out.print(x + " " + y + " ");
            x = x + 1;
        }
    }
}
```

消失的程序代码
段落

程序代码:

输出结果:

y = x - y;

22 46

y = y + x;

11 34 59

```
y = y + 2;
if( y > 4 ) {
    y = y - 1;
}
```

02 14 26 38

02 14 36 48

```
x = x + 1;
y = y + x;
```

00 11 21 32 42

```
if ( y < 5 ) {
    x = x + 1;
    if ( y < 3 ) {
        x = x - 1;
    }
}
y = y + 2;
```

11 21 32 42 53

00 11 23 36 410

02 14 25 36 47

将程序代码划条线
连到正确的输出
结果

程序谜题



泳池迷宫

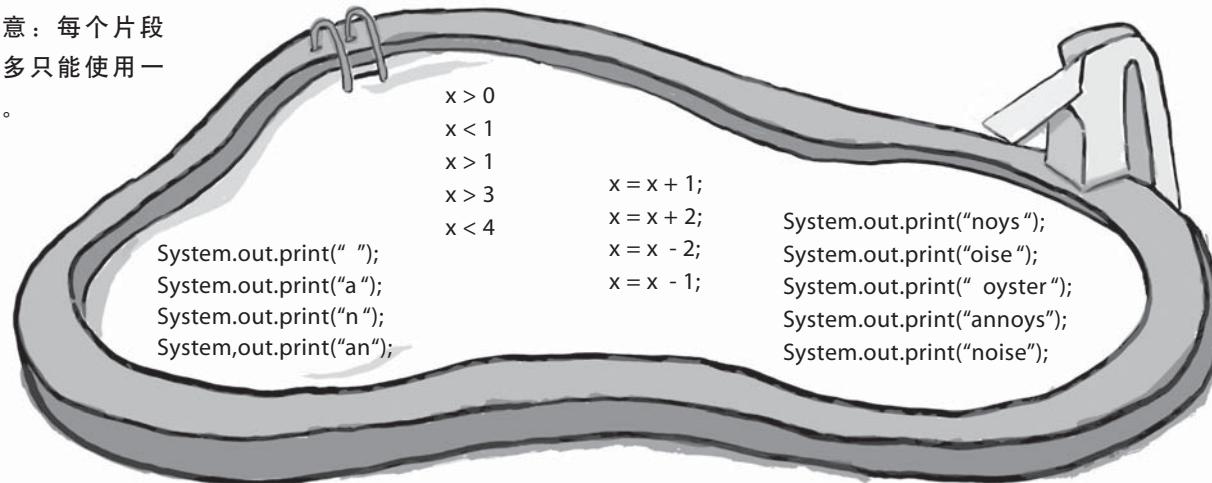


你的任务是要从游泳池挑出程序片段并将它填入右边的空格中。同一个片段不能用两次，且泳池中有些多余的片段。填完空格的程序必须要能够编译与执行并产生出下面的输出。别被骗了，这个问题比看起来的样子要困难的多。

输出：

```
File Edit Window Help Cheat
%java PoolPuzzleOne
a noise
annoys
an oyster
```

注意：每个片段最多只能使用一次。



```
class PoolPuzzleOne {
    public static void main(String [] args) {
        int x = 0;

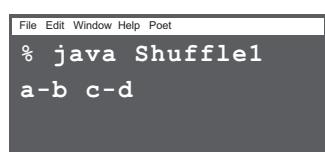
        while ( _____ ) {

            if ( x < 1 ) {
                _____
            }
            _____
            if ( _____ ) {
                _____
            }
            _____
        }
        if ( x == 1 ) {
            _____
        }
        if ( _____ ) {
            _____
        }
        System.out.println("");
    }
}
```



排排看

```
class Shuffle1 {  
    public static void main(String [] args) {  
  
        int x = 3;  
        while (x > 0) {  
  
            if (x > 2) {  
                System.out.print("a");  
            }  
  
            x = x - 1;  
            System.out.print("-");  
  
            if (x == 2) {  
                System.out.print("b c");  
            }  
  
            if (x == 1) {  
                System.out.print("d");  
                x = x - 1;  
            }  
        }  
    }  
}
```



基本概念
class Exerciselb {

```
    public static void main(String [] args) {  
        int x = 1;  
        while ( x < 10 ) {  
            x = x + 1;  
            if ( x > 3) {  
                System.out.println("big x");  
            }  
        }  
    }  
}
```

A

这个程序可以编译与执行，
但如果沒有加入特別标记出
来的这一行，它会无止境地
执行下去！

class Foo {

```
    public static void main(String [] args) {  
        int x = 5;  
        while ( x > 1 ) {  
            x = x - 1;  
            if ( x < 3) {  
                System.out.println("small x");  
            }  
        }  
    }  
}
```

B

如果没有加上类的声
明，这个程序就无法
通过编译！

class Exerciselb {
 public static void main(String [] args) {

```
        int x = 5;  
        while ( x > 1 ) {  
            x = x - 1;  
            if ( x < 3) {  
                System.out.println("small x");  
            }  
        }  
    }  
}
```

C

while 循环必须要在
方法里面！

迷题解答



```

class PoolPuzzleOne {
    public static void main(String [] args) {
        int x = 0;

        while ( X<4 ) {

            System.out.print("a");
            if ( x < 1 ) {
                System.out.print(" ");
            }
            System.out.print("\n");

            if ( X>1 ) {

                System.out.print(" oyster");
                x = x + 2;
            }
            if ( x == 1 ) {

                System.out.print("noys");
            }
            if ( X<1 ) {

                System.out.print("oise");
            }
            System.out.println("");
        }

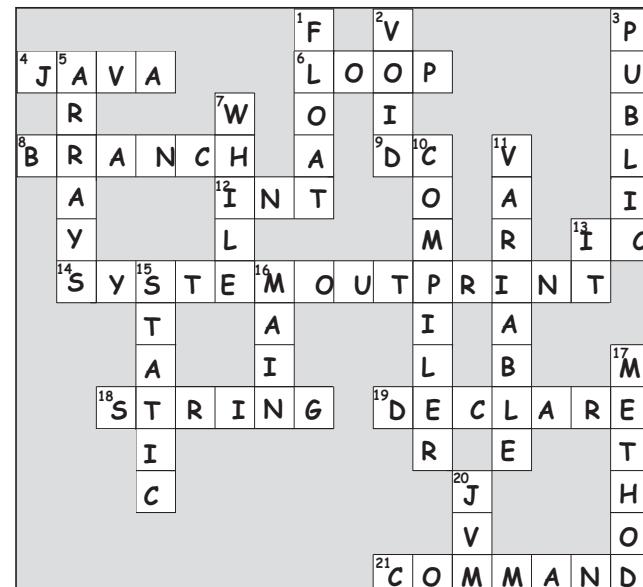
        X=X+1;
    }
}

```

```

File Edit Window Help Cheat
%java PoolPuzzleOne
a noise
annoys
an oyster

```



```

class Test {
    public static void main(String [] args) {
        int x = 0;
        int y = 0;
        while ( x < 5 ) {
            System.out.print(x + " " + y + " ");
            x = x + 1;
        }
    }
}

```

y = x - y;	22 46
y = y + x;	11 34 59
y = y + 2; if(y > 4) { y = y - 1; }	02 14 26 38 02 14 36 48
x = x + 1; y = y + x;	00 11 21 32 42
if (y < 5) { x = x + 1; if (y < 3) { x = x - 1; } y = y + 2;	11 21 32 42 53 00 11 23 36 410 02 14 25 36 47

2 类与对象

拜访对象村



有人告诉我那里遍地都是对象。在第1章中，我们把所有的程序代码放在 main() 里面。事实上，那根本就不是面向对象的做法。我们是调用到一些对象，比如 String 等，但是没有开发出自己设计的对象类型。所以我们要离开过程化的世界，开始建立自己的对象。我们会看到为何 Java 中的面向对象开发是如此的有趣。我们也会看到类与对象的不同，以及对象是如何让你的生活更美好（至少程序设计工作的部分会更好，但对是否能够受到异性的青睐就不一定了）。注意：一旦进入对象村，你就不想再回头。

对象村

椅子大战

(又称： 对象如何改变你的一生)



前，有一间软件小铺，
里面有两个程序员被
指派去设计一个程序。

坏心的老板娘兼项目经理要求两个人比赛，赢的人可以坐上象征身分地位的 Aeron™ 宝椅。程序开发高手阿珠跟面向对象信徒阿花两个人都认为自己赢的可能性很大。

阿珠坐在自己的座位上想着：“这个程序要执行什么动作？我会需要什么样的程序？有了，我需要rotate与playSound！”然后她就开始进行设计的工作。

同时，阿花去倒了一杯咖啡回来，心想：“这个程序有什么样的事物？有什么关键角色？”她首先想到形状体 (shape)。当然啦，她还会想到用户、声响等对象与点击等事件。然而这些对象早就已经建立好了，所以她只需要专注于创建形状体就行了。

接下来就是她们如何设计程序的故事以及你最想知道的答案：“谁赢了 Aeron™ 宝椅？”。



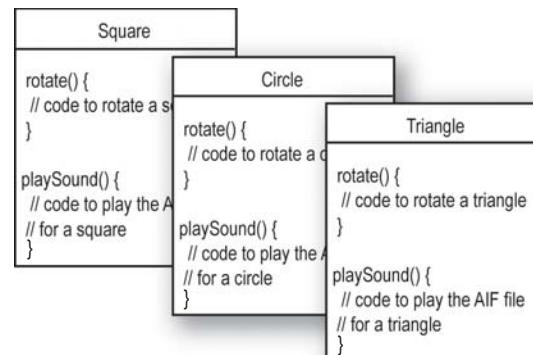
阿珠

如同以往，她准备好要开始设计重要的程序。没几下她就写出了rotate与playSound两个方法。

```
rotate(shapeNum) {  
    // 旋转360°  
}  
playSound(shapeNum) {  
    // 查询播放哪个AIF文件  
    // 播放  
}
```

阿花

阿花分别为3个形状各写出一个类。

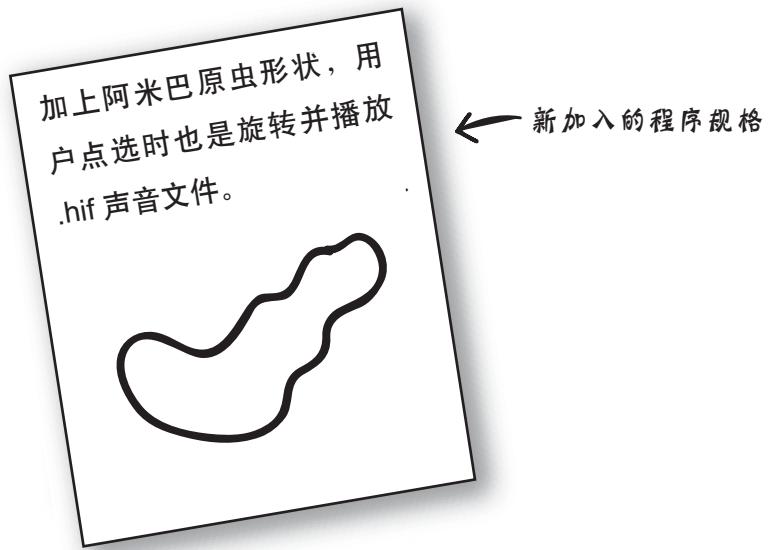


正当阿珠心想说赢得了，开始幻想坐在宝椅上接受大家投以羡慕的眼光……

等一下！老板娘说规格改了。

“OK，技术上来说阿珠赢了……大毛！你给我过来！”老板娘一边追打小孩一边说：“但是我作了一点小小的改变，对你们这种高手来说一定很简单的”。

“说呀，还要改什么？”阿珠与阿花不约而同地盯着角落的折凳看，四只手好像随时准备抄起凳子开始干活，不过想到已经忍了这么久，也不差这一次。



阿珠

原来的rotate程序还可以用；该程序使用一个对应表来找寻特定编号的图形。但是playSound就得要修改。还有.hif文件是什么鬼东西？

```
playSound(shapeNum) {
    // 如果不是阿米巴原虫
    // 查询使用哪个AIF文件
    // 播放
    // 不然
    // 播放amoeba .hif
}
```

虽然修改幅度不大，但是她实在不想去碰已经测试过的程序代码。她应该很清楚，不管项目经理怎么保证，规格就是会不停地改。

阿花

她苦笑一下，耸一耸肩，坐下来写出一个新的类。面向对象让她最喜欢的其中一点就是有时不需动到已经测试好的程序就可以达成新目标。面向对象的适应性与可扩展性让她面对修改时不会觉得太过于痛苦。

Amoeba
<pre>rotate() { // 旋转 } playSound() { // 播放 }</pre>

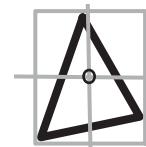
对象村

阿珠悄悄地领先了阿花

但是听到坏心老板娘用失望的口吻说出“唉呀，不对呀，阿米巴原虫不是这样旋转的……”时，阿珠的脸色都变了。

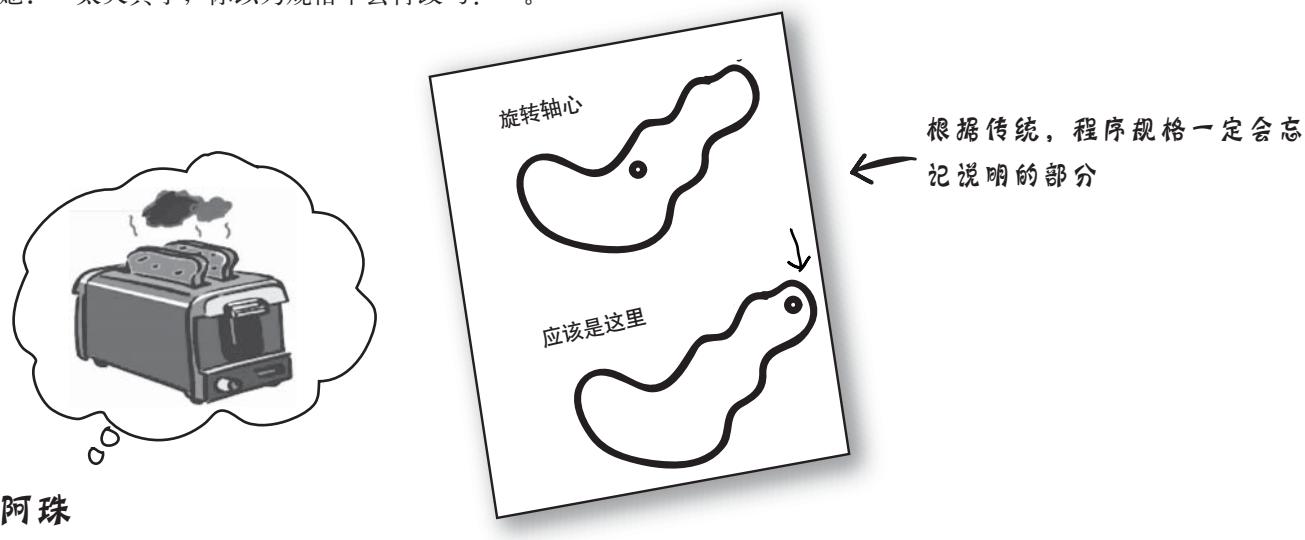
原来，两个人都把旋转的部分写成这样：

- (1) 找出指定形状的外接四边形。
- (2) 计算出四边形的中心点，以此点为轴作旋转。



但是老板娘认为阿米巴原虫应该是要绕着一端旋转，类似秒针那样。

“坏了”，阿珠心里这么想着，眼前浮现出早上烤焦的两片土司。“但我还是可以加上额外的 if/else 到 rotate 程序中硬改给阿米巴原虫，这样应该还好吧”。然而脑海中不断有个声音在提醒她：“太天真了，你以为规格不会再改吗？”。



阿珠

她想到最好还是帮rotate程序加上轴心点的参数。这样就有一堆程序要改。本来已经测试好的东西全部又得重来一遍。

```
rotate(shapeNum, xPt, yPt) {  
    // 如果不是阿米巴  
    // 计算中心点  
    // 然后旋转  
    // 否则  
    // 以xPt和yPt作为旋转中心  
    // 然后旋转  
}
```

阿花

她修改了rotate这个方法，但不是每个都要改，只修改Amoeba这个类而已。其他已经测试、编译过的部分完全没有必要改。该类要作的修改就是加上旋转轴心点的属性（attribute）。

Amoeba

```
int xPoint  
int yPoint  
rotate() {  
    // 使用阿米巴的x和y  
    // 执行旋转  
}  
playSound() {  
    // 播放  
}
```

所以阿花赢了，对吧？

还早，阿珠发现阿花的方法有缺陷。并且她认为假如能够赢得胜利，老板娘的地位早晚也会被她取代，因此她得要扭转形势。

阿珠：“你有重复的程序代码！在4个Shape物体中都有rotate过程”。

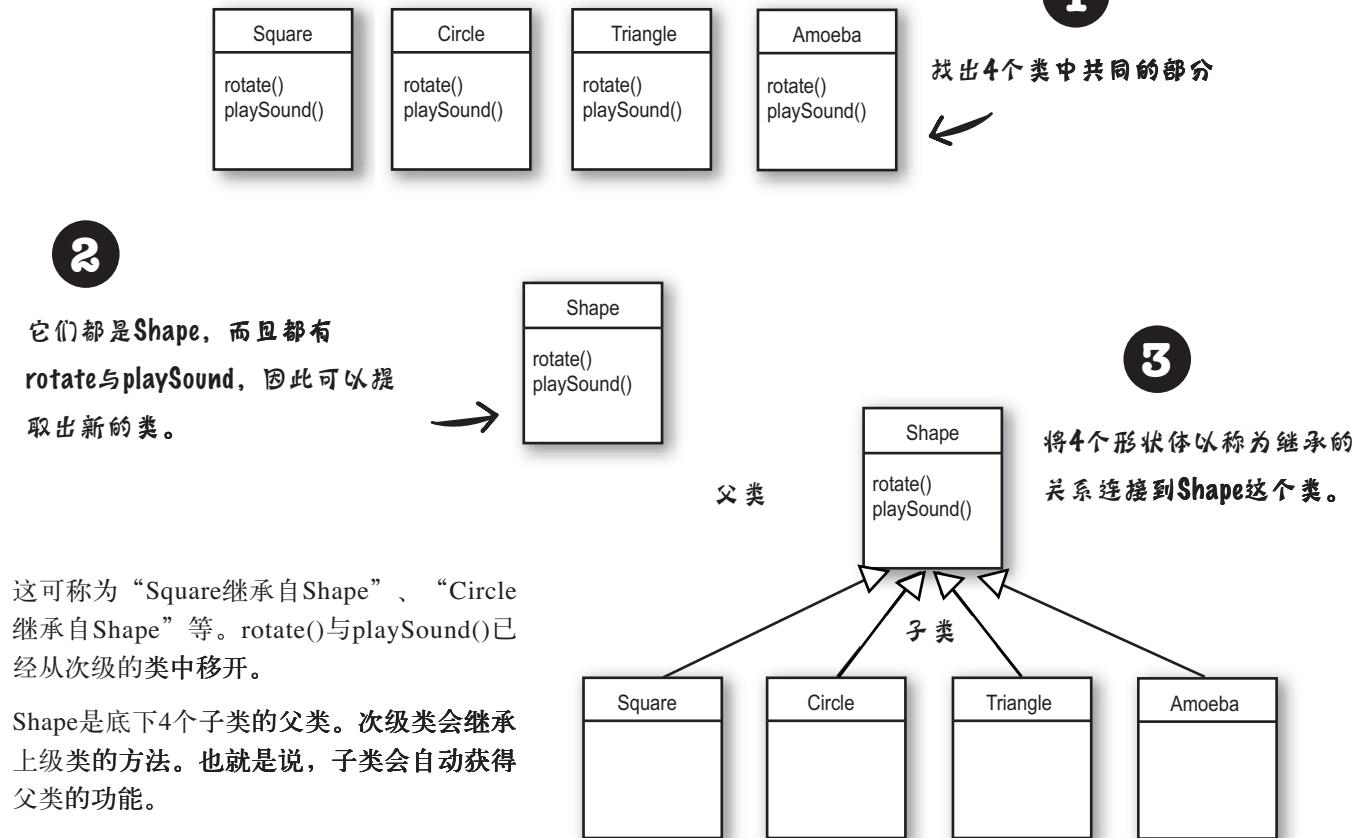
阿花：“那不叫过程，那是方法好吗？还有，物体的正式名称叫做类”。

阿珠：“无所谓。你的设计有问题。这样一来你必须同时维护4个不同的rotate方法。这一点效率都没有”。

阿花：“我猜你一定没看到最终的设计。阿珠，让我告诉你什么叫做面向对象的继承（inheritance）”。



阿珠觉得自己会赢 ↗



对象村

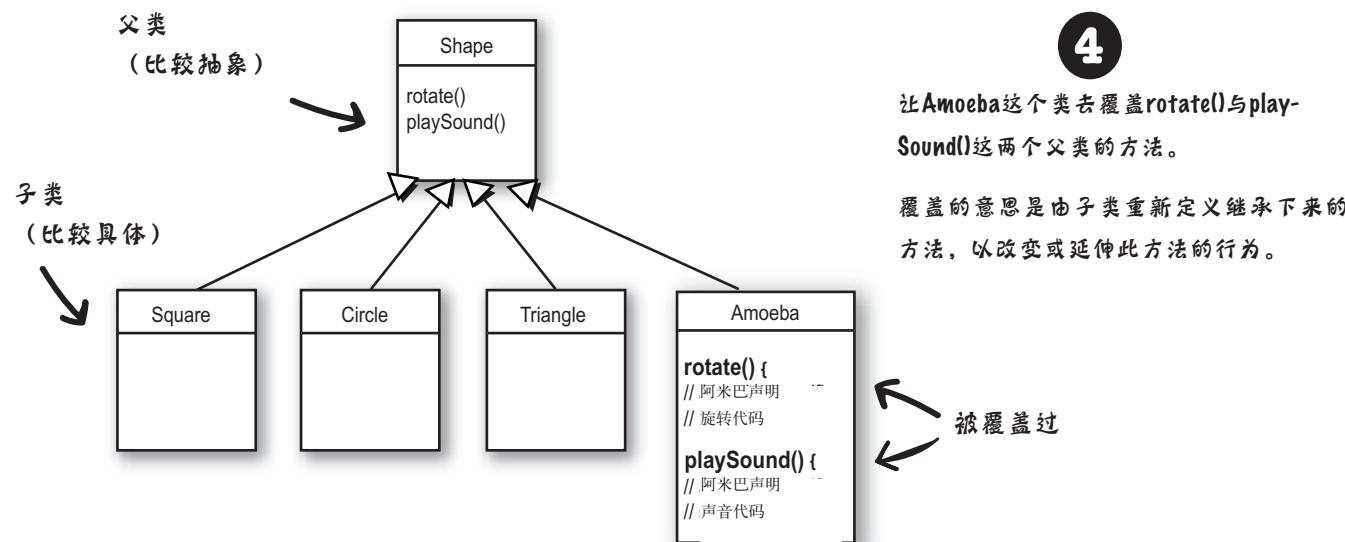
那阿米巴的rotate()要怎么办？

阿珠：“问题不就出在这里吗？阿米巴形状会需要完全不同的rotate与playSound程序？”

阿花：“那叫方法。”

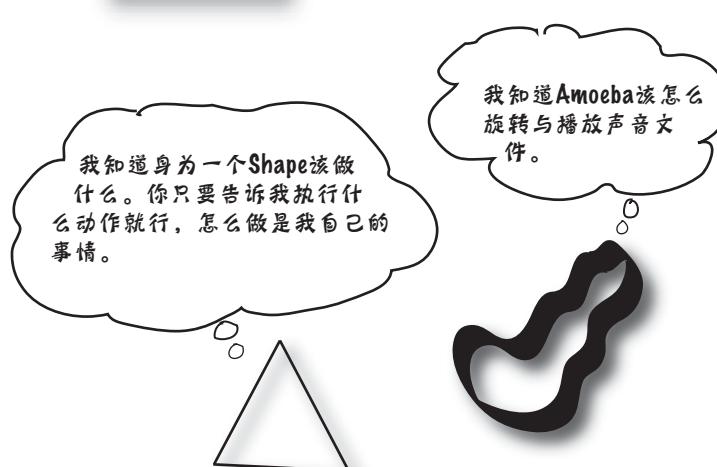
阿珠：“如果阿米巴也是继承自Shape，那旋转的功能不就统统一样吗？”

阿花：“问得好。Amoeba这个类可以覆盖（override）Shape的方法。Java虚拟机会知道在遇到Amoeba时使用不同的rotate()。



阿珠：“你要怎么告诉Amoeba去执行某个动作？你不是得要调用这个过程……呃……方法然后叫它去旋转某个形状体吗？”

阿花：“这就是面向对象的重点。当你要旋转某个三角形的时候，程序代码会调用该三角形对象的rotate()方法。调用的一方不需知道该对象如何执行这个动作。如果要加入新的对象，只要写出该对象类型的类就行，让新的对象维持自己的行为。



剧情真是太紧张了。到底
是谁赢了？



最后是阿娇获胜。

(其实有3个参赛者。附带说明：
阿娇是老板娘的侄女。)

你对面向对象有什么看法？

“它帮助我用更自然的方法设计”。

——Joy, 27岁，软件架构师

“加入新功能时不会搞乱已经写好的程序代码”。

——Brad, 32岁，程序员

“我喜欢它将数据与操作数据的方法摆在同一个类
内”。

——Josh, 22岁，啤酒品尝师

“类可以重复运用在别的应用程序中，当写一个新
类时，可以使人类有足够的扩展性，以便以后用
到”。

——Chris, 39岁，项目负责人

“我不相信Chris刚才所说的，他在近五年内没写过
一行代码”。

——Alex, 35岁，Chris的工作人员

“还有椅子大战外别的游戏吗？”。

——Geena, 26岁，程序员



该刺激一下神经了。

你刚刚读过面向过程程序员与面向对象程序员之间的唇枪舌战。这里面有面向对象关键概念的论述，包括了类、方法和属性。这一章接下来的部分会讨论类与对象。

用你目前所学到的概念来思考与回答下面的问题：

在设计Java的类时有哪些基本的事项要考虑？要对你自己提出哪些问题？如果要列出一份核对清单，你会列出哪些注意事项？

学习秘诀



如果你的思路堵塞了，试着把内容大声地念出来。说话与倾听都会用到大脑的不同位置。虽然与人交谈效果最好，但是对着宠物念也会有帮助。我家的狗狗就是这样学会JSP的。

以对象来思考

当你在设计类时，要记得对象是靠类的模型塑造出来的。你可以这样看：

- 对象是已知的事物
- 对象会执行的动作

<table border="1"><tr><td>ShoppingCart</td></tr><tr><td>cartContents</td></tr><tr><td>addToCart() removeFromCart() checkOut()</td></tr></table>	ShoppingCart	cartContents	addToCart() removeFromCart() checkOut()	已知 执行	<table border="1"><tr><td>Button</td></tr><tr><td>label color</td></tr><tr><td>setColor() setLabel() dePress() unDepress()</td></tr></table>	Button	label color	setColor() setLabel() dePress() unDepress()	已知 执行	<table border="1"><tr><td>Alarm</td></tr><tr><td>alarmTime alarmMode</td></tr><tr><td>setAlarmTime() getAlarmTime() isAlarmSet() snooze()</td></tr></table>	Alarm	alarmTime alarmMode	setAlarmTime() getAlarmTime() isAlarmSet() snooze()	已知 执行
ShoppingCart														
cartContents														
addToCart() removeFromCart() checkOut()														
Button														
label color														
setColor() setLabel() dePress() unDepress()														
Alarm														
alarmTime alarmMode														
setAlarmTime() getAlarmTime() isAlarmSet() snooze()														

对象本身已知的事物被称为：

- 实例变量 (instance variable)

对象可以执行的动作称为：

- 方法 (methods)

实例变量
(状态)
方法
(行为)

<table border="1"><tr><td>Song</td></tr><tr><td>title artist</td></tr><tr><td>setTitle() setArtist() play()</td></tr></table>	Song	title artist	setTitle() setArtist() play()	已知 执行
Song				
title artist				
setTitle() setArtist() play()				

对象本身已知的事物称为实例变量 (instance variable)。它们代表对象的状态 (数据)，且该类型的每一个对象都会独立的拥有一份该类型的值。

所以你也可以把对象当作为实例。

对象可以执行的动作称为方法。在设计类时，你也会设计出操作对象数据的方法。对象带有读取或操作实例变量的方法是很常见的情形。举例来说，闹钟对象会有个变量来保存响铃时间，且会有getTime()与 setTime()这两个方法来存取该时间。

因此说对象带有实例变量和方法，但它们都是类设计中的一部分。



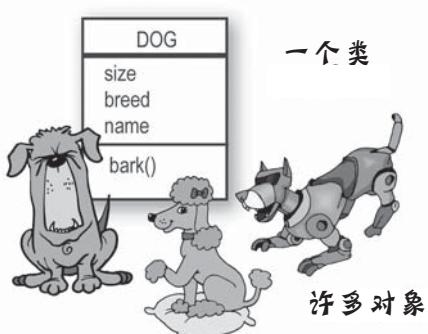
填入电视对象该有的实例变量
和方法。

<table border="1"><tr><td>Television</td></tr><tr><td> </td></tr><tr><td> </td></tr></table>	Television		
Television			

实例变量

方法

到底类与对象两者之间有什么不同？



**类不是对象
(却是用来创建它们的模型)**

类是对象的蓝图。它会告诉虚拟机如何创建某种类型的对象。根据某类创建出的对象都会有自己的实例变量。举例来说，你可以使用按钮类来创建出许多大小、颜色、文字等不同的按钮。



也可以这么说...



对象就好像通讯簿中的一笔数据

通讯簿的每张卡片都有相同的空白字段（实例变量）。填入新的联络人就如同创建新的实例（对象），填入卡片的数据代表联络人的状态。

这个卡片类上的方法就是你会对卡片作的事情：`getTel()`、`changeAddress()`、`deleteCard()`等。

所以每张卡能够执行相同动作，但取出的结果应该是依每张卡片各自独立的。

创建你的第一个对象

要作出哪些东西才会运用对象呢？你需要两个类。一个是要被操作于对象的类（比如说Dog、AlarmClock和Television等），另一个是用来测试该类的类。测试用的类带有main()并且你会在其中建立与存取被测的对象。

本书后续章节的内容会有许多双类的范例。测试用的类会被命名为“受测类名称”+TestDrive。例如说要测试Bungee这个类，我们会作出一个带有main()的BungeeTestDrive。它会创建出Bungee的对象，并使用圆点(.)符号所代表的操作数来存取该对象的变量与方法。看过下面的范例后就应该更清楚了。

1 编写类

```
class Dog {  
    int size;           实例变量  
    String breed;  
    String name;  
  
    void bark() {      方法  
        System.out.println("Ruff! Ruff!");  
    }  
}
```



2 编写测试用的类

```
class DogTestDrive {  
    public static void main (String[] args) {  
        // Dog 测试码  
    }  
}
```

3 在测试用的类中建立对象并存取对象的变量和方法

```
class DogTestDrive {  
    public static void main (String[] args)  
    {  
        Dog d = new Dog();          建立 Dog 对象  
        d.size = 40;                存取该对象的变量  
        d.bark();                  调用它的方法  
    }  
}
```

圆点运算符(.)

此运算符能让你存取对象的状态与行为。

// 建立对象

```
Dog d = new Dog();
```

// 通过操作和调用

method

```
d.bark();
```

// 通过操作数存取属性

```
d.size = 40;
```

创建与测试 Movie 对象

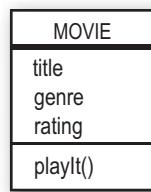


(“外星杀人对象”上映中)

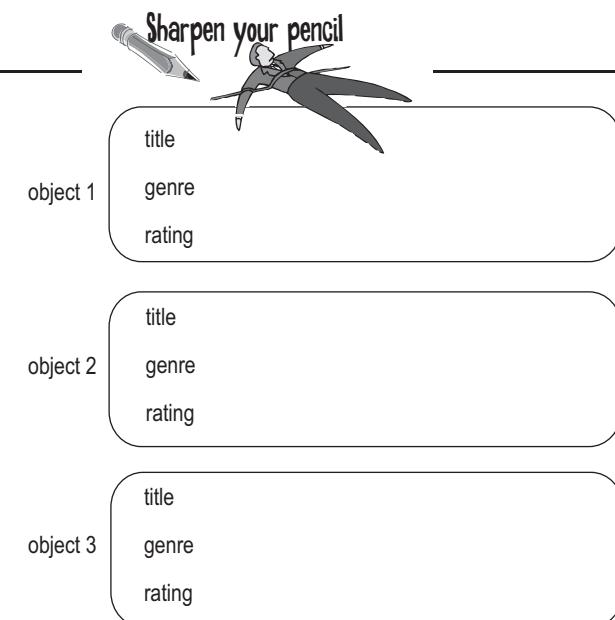
```
class Movie {
    String title;
    String genre;
    int rating;

    void playIt() {
        System.out.println("Playing the movie");
    }
}

public class MovieTestDrive {
    public static void main(String[] args) {
        Movie one = new Movie();
        one.title = "Gone with the Stock";
        one.genre = "Tragic";
        one.rating = -2;
        Movie two = new Movie();
        two.title = "Lost in Cubicle Space";
        two.genre = "Comedy";
        two.rating = 5;
        two.playIt();
        Movie three = new Movie();
        three.title = "Byte Club";
        three.genre = "Tragic but ultimately uplifting";
        three.rating = 127;
    }
}
```



MovieTestDrive这个类会创建出Movie的对象（实例）并使用圆点运算符来设定数据。MovieTestDrive也会调用其中一个对象的方法。将右方的空白处填上main()执行完毕后的对象值。



逃出main()

快离开 main!

只要还呆在main()中，你就是在对象村外。呆在main()中对于一个测试用的程序来说是还好的，但对于货真价实的面向对象应用程序来说，你会需要用对象来与对象交互。

main()的两种用途：

- 测试真正的类
- 启动你的Java应用程序

真正的Java程序只会让对象与对象交互。此处所说的交互是指相互调用方法。上一页与后面的第4章会讨论在独立的TestDrive类中创建与测试其他的类。第6章会看到使用带有main()的类来启动真正的Java应用程序（创建对象并让对象之间产生交互）。

现在先给你看个真正Java应用程序会怎么做“预览”，以下是个小范例。因为我们还在学习Java的初期阶段，能够运用的技巧有限，所以程序不太优雅且无效率。你可能会思考如何将它改善，而这正是我们在后续章节会做的事。别担心看不懂某些部分；这个范例的重点在于示范对象如何与对象互动。

猜数字游戏

摘要：

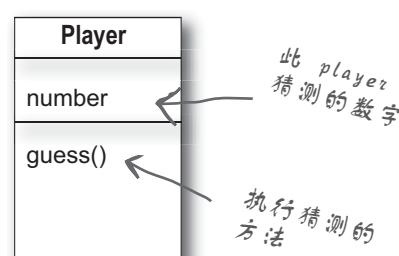
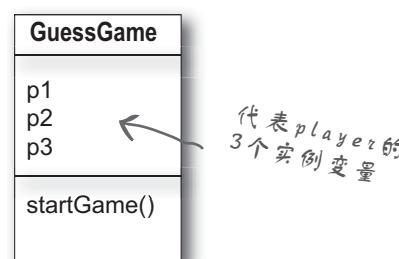
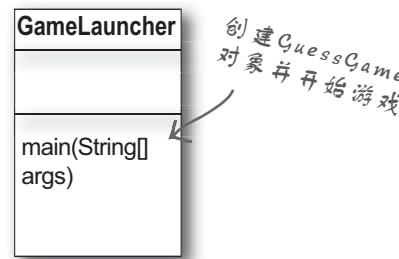
这个游戏涉及到game与player两个对象。game会产生介于0~9之间的随机数字，而3个player对象会猜测该数字（你应该会觉得很无聊）。

类：

`GuessGame.class` `Player.class` `GameLauncher.class`

程序逻辑：

- (1) GameLauncher 这个类带有main()方法，是应用程序的入口点。
- (2) main()中会创建出GuessGame对象，并调用它的startGame()方法。
- (3) startGame()方法是游戏的起点。它会创建3个player，然后挑出要猜测的随机数字。它会要求player猜测并检查结果，过程会被列出来。



类与对象

猜数字

运行猜数字游戏

```
public class Player {  
    int number = 0; // 要被猜的数字  
  
    public void guess() {  
        number = (int) (Math.random() * 10);  
        System.out.println("I'm guessing "  
                           + number);  
    }  
}  
  
public class GameLauncher {  
    public static void main (String[] args) {  
        GuessGame game = new GuessGame();  
        game.startGame();  
    }  
}
```



Java会拾荒

创建对象时，它会被存放在称为堆的内存区域中。不管对象如何创建都会放在此区域中。此区域并非普通的堆；它是可回收垃圾的堆（Garbage-Collectible Heap）。Java会根据对象的大小来分配内存空间。比如说15个实例变量的对象所占用的空间就可能会比只有两个实例变量的对象要大。但对象使用完毕时内存要如何回收呢？Java会主动帮你管理内存！当某个对象被Java虚拟机察觉不再会被使用到，该对象就会被标记成可回收的。如果内存开始不足，垃圾收集器就会启动来清理垃圾、回收空间，让空间能够再次被利用。后面的章节会对此机制有更多的讨论。

输出（每次执行都会不一样）

```
File Edit Window Help Explode  
%java GameLauncher  
I'm thinking of a number between 0 and 9...  
Number to guess is 7  
I'm guessing 1  
I'm guessing 9  
I'm guessing 9  
Player one guessed 1  
Player two guessed 9  
Player three guessed 9  
Players will have to try again.  
Number to guess is 7  
I'm guessing 3  
I'm guessing 0  
I'm guessing 9  
Player one guessed 3  
Player two guessed 0  
Player three guessed 9  
Players will have to try again.  
Number to guess is 7  
I'm guessing 7  
I'm guessing 5  
I'm guessing 0  
Player one guessed 7  
Player two guessed 5  
Player three guessed 0  
We have a winner!  
Player one got it right? true  
Player two got it right? false  
Player three got it right? false  
Game is over.
```

*there are no
Dumb Questions*

问：若需要全局（global）变量或方法时该如何？

答：在Java的面向对象概念中并没有全局变量这回事。然而实际上会有需要方法或常量（constant）可被任何的程序存取。比如说专家术语学习机中到处都在调用的random()方法或圆周率这种常数。第10章会讨论到public与static这些让方法变成类似“global”的修饰符。在任何类中的任何程序都可以存取public static的方法。任何变量只要加上public、static和final，基本上都会变成全局变量可用的常数。

问：如果能做出全局的函数与数据，那又怎么算得上是面向对象呢？

答：首先要注意到任何Java中的事物都必须呆在类中。因此，pi常数或random()方法也必须定义在Math这个类中。而你必须记住这类近似全局的事物在Java中算是例外。它们是非常特殊的情况，不会有多个实例或对象。

问：什么是Java程序？如何进行提交？

答：Java程序是由一组类所组成，其中有一个类会带有启动用的main()方法。因此程序员必须要编写一或多个类并以此提交。若用户没有Java虚拟机则必须一并提交才能让应用程序运行起来。有数种安装程序能够让你集成包装类与不同平台上使用的Java虚拟机到安装光盘上。如此就能让用户同时安装正确版本的Java虚拟机（如果之前没有安装的话）。

问：若有成百上千的类时要如何提交？是否可以包装成类似单一应用程序的形式？

答：数量庞大的个别文件确实会让用户头疼。你可以把所有文件包装进依据pkzip格式来存档的Java Archive – jar文件。在jar文件中可以引入一个简单文字格式的文字文件，它被称为manifest，里面有定义出jar中的哪一个文件带有启动应用程序的main()方法。



要点

- 面向对象设计扩展功能不需改动之前已经测试好的程序代码。
- 所有的Java程序都定义在类中。
- 类如同蓝图描述该类型的对象要如何创建。
- 对象自治；你无需在意它如何完成任务。
- 对象有已知的事物，并能执行工作。
- 对象本身已知道的事物称为实例变量，它代表对象的状态。
- 对象可执行的动作称为方法，它代表对象的行为。
- 创建类时，可能同时会需要创建独立、测试用的类。
- 类可以继承自较为抽象的父类。
- Java的程序在执行期是一组会互相交谈的对象。

习题：我是编译器



我是编译器



这一页的Java程序代码都代表一份完整的源文件。你的任务是要扮演编译器角色并判断哪支程序可以编译通过。如果有问题，哪里需要修改？

A

```
class TapeDeck {  
  
    boolean canRecord = false;  
  
    void playTape() {  
        System.out.println("tape playing");  
    }  
  
    void recordTape() {  
        System.out.println("tape recording");  
    }  
}  
  
class TapeDeckTestDrive {  
    public static void main(String [] args) {  
  
        t.canRecord = true;  
        t.playTape();  
  
        if (t.canRecord == true) {  
            t.recordTape();  
        }  
    }  
}
```

B

```
class DVDPlayer {  
  
    boolean canRecord = false;  
  
    void recordDVD() {  
        System.out.println("DVD recording");  
    }  
}  
  
class DVDPlayerTestDrive {  
    public static void main(String [] args) {  
  
        DVDPlayer d = new DVDPlayer();  
        d.canRecord = true;  
        d.playDVD();  
  
        if (d.canRecord == true) {  
            d.recordDVD();  
        }  
    }  
}
```



排排看

右边是被打散的 Java 程序片段，你是否能够将它们重新排列成为可以编译与运行并产生如同下方的输出结果？注意到有些括号已经遗失，所以你可以认为有需要时自行补上。

```
d.playSnare();
```

```
DrumKit d = new DrumKit();
```

```
boolean topHat = true;  
boolean snare = true;
```

```
void playSnare() {  
    System.out.println("bang bang ba-bang");  
}
```

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

```
if (d.snare == true) {  
    d.playSnare();  
}
```

```
d.snare = false;
```

```
class DrumKitTestDrive {
```

```
d.playTopHat();
```

```
class DrumKit {
```

```
void playTopHat () {  
    System.out.println("ding ding da-ding");  
}
```

```
File Edit Window Help Dance  
% java DrumKitTestDrive  
bang bang ba-bang  
ding ding da-ding
```

迷题



泳池迷宫



你的任务是要从游泳池中挑出程序片段并将它填入右边的空格中。
同一个片段不能使用两次，且游泳池中有些多余的片段。填完空格的程序必须要能够编译与执行并产生出下面的输出。

输出：

```
File Edit Window Help Implode
%java EchoTestDrive
helloooo...
helloooo...
helloooo...
helloooo...
10
```

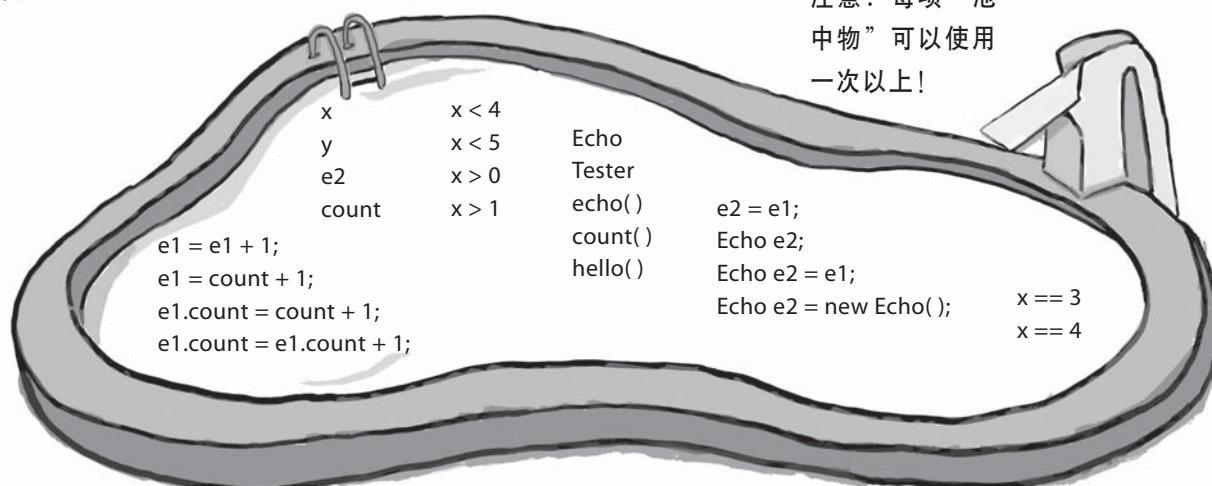
```
public class EchoTestDrive {
    public static void main(String [] args)
    {
        Echo e1 = new Echo();

        _____
        int x = 0;
        while ( _____ ) {
            e1.hello();
            _____
            if ( _____ ) {
                e2.count = e2.count + 1;
            }
            if ( _____ ) {
                e2.count = e2.count + e1.count;
            }
            x = x + 1;
        }
        System.out.println(e2.count);
    }
}
```

```
class _____ {
    int _____ = 0;
    void _____ {
        System.out.println("helloooo... ");
    }
}
```

送分题！

如果最后一行的输出不是 10 而是 24，空格又应该如何填呢？





一组 Java 组件精心打扮出席化装舞会，中场时间有人提议要玩猜猜我是谁的游戏，你可以根据它们对自己的描述来猜测出提示的是哪位。规则是每个组件都得说实话，若某些提示同时对数个组件都为真的话，则将它们全部填入。第一位已经被我们猜出来了。

今晚出席舞会的有：

类 方法 对象 实例变量

我是由.java文件编译出来的 _____ 类

我的实例变量值可以与其他兄弟姐妹不同 _____

我的功能类似模板 _____

我喜欢执行工作 _____

我带有很多方法 _____

我代表“状态” _____

我拥有很多行为 _____

我呆在对象中 _____

我生存于堆上 _____

我被用来创建对象实例 _____

我的状态可以改变 _____

我会声明方法 _____

我可以运行期变化 _____

习题解答



解答

排排看

```
class DrumKit {  
  
    boolean topHat = true;  
    boolean snare = true;  
  
    void playTopHat() {  
        System.out.println("ding ding da-ding");  
    }  
  
    void playSnare() {  
        System.out.println("bang bang ba-bang");  
    }  
  
}  
  
class DrumKitTestDrive {  
    public static void main(String [] args) {  
  
        DrumKit d = new DrumKit();  
        d.playSnare();  
        d.snare = false;  
        d.playTopHat();  
  
        if (d.snare == true) {  
            d.playSnare();  
        }  
    }  
}
```

```
File Edit Window Help Dance  
% java DrumKitTestDrive  
bang bang ba-bang  
ding ding da-ding
```

我是编译器

```
A class TapeDeck {  
    boolean canRecord = false;  
    void playTape() {  
        System.out.println("tape playing");  
    }  
    void recordTape() {  
        System.out.println("tape recording");  
    }  
}
```

```
class TapeDeckTestDrive {  
    public static void main(String [] args) {  
  
        TapeDeck t = new TapeDeck();  
        t.canRecord = true;  
        t.playTape();  
  
        if (t.canRecord == true) {  
            t.recordTape();  
        }  
    }  
}
```

必须要把对象建立起来!

```
class DVDPlayer {  
    boolean canRecord = false;  
    void recordDVD() {  
        System.out.println("DVD recording");  
    }  
    void playDVD (){  
        System.out.println("DVD playing");  
    }  
}
```

```
B class DVDPlayerTestDrive {  
    public static void main(String [] args) {  
        DVDPlayer d = new DVDPlayer();  
        d.canRecord = true;  
        d.playDVD();  
        if (d.canRecord == true) {  
            d.recordDVD();  
        }  
    }  
}
```

必须补上playDVD()这个方法



泳池迷宫

```
public class EchoTestDrive {
    public static void main(String [] args) {
        Echo e1 = new Echo();
        Echo e2 = new Echo(); // 正确答案
        - 或 -
        Echo e2 = e1; // 也可以!
        int x = 0;
        while ( x < 4 ) {
            e1.hello();
            e1.count = e1.count + 1;
            if ( x == 3 ) {
                e2.count = e2.count + 1;
            }
            if ( x > 0 ) {
                e2.count = e2.count + e1.count;
            }
            x = x + 1;
        }
        System.out.println(e2.count);
    }
}

class Echo {
    int count = 0;
    void hello() {
        System.out.println("helloooo... ");
    }
}
```

我是谁？

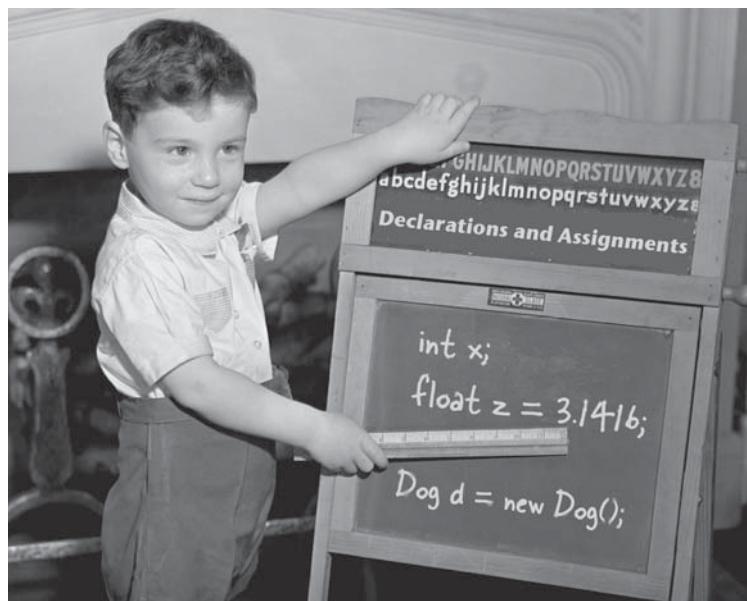
- | | |
|--------------------|----------------------------------|
| 我是由 .java 文件编译出来的 | <i>class</i> |
| 我的实例变量值可以与其他兄弟姐妹不同 | <i>object</i> |
| 我的功能类似模板 | <i>class</i> |
| 我喜欢执行工作 | <i>object, method</i> |
| 我带有很多方法 | <i>class, object</i> |
| 我代表 “状态” | <i>instance variable</i> |
| 我拥有很多行为 | <i>object, class</i> |
| 我呆在对象中 | <i>method, instance variable</i> |
| 我生存于堆上 | <i>object</i> |
| 我被用来创建对象实例 | <i>class</i> |
| 我的状态可以改变 | <i>object, instance variable</i> |
| 我会声明方法 | <i>class</i> |
| 我可以在运行期变化 | <i>object, instance variable</i> |

注意：类 (*class*) 与对象 (*object*) 两者都说有状态与行为。它们是定义在 *class* 中的，我们并未在技术上很精确地界定对象说“有”的意义。

```
File Edit Window Help Assimilate
%java EchoTestDrive
helloooo...
helloooo...
helloooo...
helloooo...
10
```

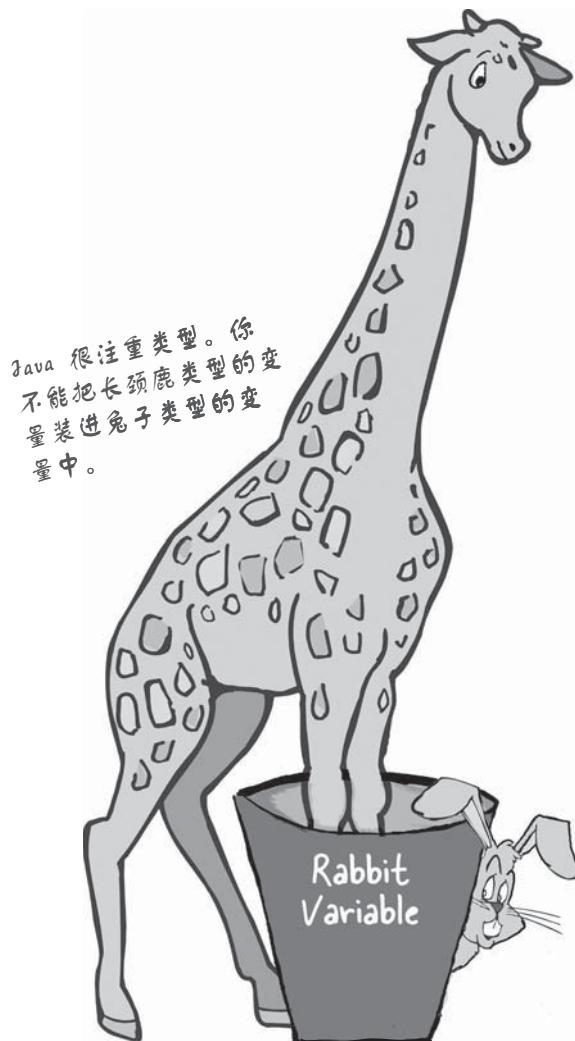

3 primitive主数据类型和引用

认识变量



变量有两种： primitive主数据类型和引用。到目前为止你已经在两处使用过变量——对象的状态（instance variables）与局部（local）变量（声明在方法中的变量）。稍后我们会把变量用于参数（arguments，传递给方法的值）及返回类型（执行方法所返回的值）。你已经看过被声明成primitive整数值（int类型）的变量以及声明成更为复杂如String或数组等类型的东西。但一定还有比这些东西更为复杂的事物。像是PetOwner对象会带有Dog实例变量，或者Car对象带有Engine实例变量。这一章会为你解开Java类型的迷团、探索变量的声明以及研究如何运用变量等议题。最后还会看到垃圾可回收的堆对你本周运势的影响。

声明变量



声明变量

Java注重类型。它不会让你做出把长颈鹿类型变量装进兔子类型变量中这种诡异又危险的举动——如果有人对长颈鹿调用“跳跃”这个方法会发生什么样的悲剧？并且它也不会让你将浮点数类型变量放进整数类型的变量中，除非你先跟编译器确认过数字可以损失掉精确度（例如说舍去所有的小数值）。

编译器会指出大部分的问题：

```
Rabbit hopper = new Giraffe();
```

谢天谢地，这样的程序过不了编译器这关。

为了要让类型安全能够发挥作用，你必须声明所有变量的类型，指定它是个int类型或是个Dog类型。变量有两种口味：清凉的primitive主数据类型与香辣的对象引用。primitive主数据类型用来保存基本类型的值，包括整数、布尔和浮点数等。而对象引用保存的是对象的引用（嗯，这样解释很清楚吧）。

我们会先看primitive主数据类型然后再讨论对象引用真正的意义。先记住下面这条声明变量的规则：

variables must have a type

变量必须拥有类型。另一条规则是必须要有名称。

variables must have a name

```
int count;
```

↑ ↙
 类型 名称

注意：当你读到“X类型的Y对象”时，类型(type)此时与类是相通的同义字。

primitive主数据类型和引用

“我要大杯的摩卡咖啡，不，中杯好了”

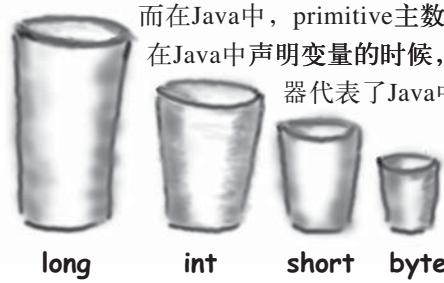
你可以把Java的变量想成是杯子。咖啡杯、茶杯、装满啤酒的泡沫红茶店跟鱼缸一样大的巨无霸杯、电影院贩卖爆米花用的大杯、独享杯、冠军杯、警察杯、有把手的杯以及绝对不能放进微波炉的那种镶有金属的杯。

变量就像是杯子，是一种容器，承装某些事物。

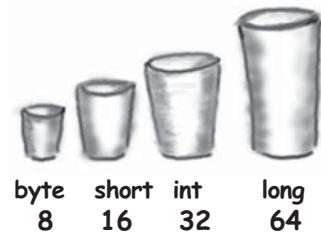
它有大小与类型。在这一章中，我们会先看过承载primitive主数据类型的变量（杯子），稍后再来看装载对象引用的杯子。我们现在以杯子的比喻来看待变量，这是比较简单地说法，后面的讨论将会越来越复杂。

primitive主数据类型如同咖啡馆的杯子，它们有不同的大小，而每种大小都有个名称，像是“小杯”、“大杯”、“重量杯”等。

你或许会在吧台上面看到展示出的杯子，可以借此选择适当的大小：



而在Java中，primitive主数据类型也有不同的大小与名称。当你在Java中声明变量的时候，必须指定它的类型。下图中的4种容器代表了Java中的4种基本整数类型。



每种杯子都可以装载数值，就像你会说“我要小杯的芒果冰沙”，你也要告诉编译器：“请给我一个int变量保存数值90”。其中有个小小的差异——你还得为杯子命名。每种primitive主数据类型变量有固定的位数（杯子大小）。存放数值的primitive主数据类型有下列6种大小：

primitive主数据类型

类型 位数 值域

boolean与char

boolean (Java虚拟机决定) true或false

char 16 bits 0 ~ 65535

数值(带正负号)

integer

byte 8 bits -128 ~ 127

short 16 bits -32768 ~ 32767

int 32 bits -2147483648 ~ 2147483647

long 64 bits -很大 ~ +很大

浮点数

float 32 bits 范围规模可变

double 64 bits 范围规模可变

primitive主数据类型的声明与赋值声明：

```
int x;  
x = 234;  
byte b = 89;  
boolean isFun = true;  
double d = 3456.98;  
char c = 'f';  
int z = x;  
boolean isPunkRock;  
isPunkRock = false;  
boolean powerOn;  
powerOn = isFun;  
long big = 3456789;  
float f = 32.5f;
```

注意“f”。除非加上f，否则所有带小数点的值都会被Java当作double处理。

primitive主数据类型的赋值

小心别溢出来……

要确保变量能存下所保存的值。



你无法用小杯子装大值。

好吧，其实可以，但是会损失某些信息，也就是所说的溢位。当判断到所使用的容器不足以装载时，编译器会试着防止这种情况发生。

举例来说，你无法像下面这样把 int 大小的东西装进 byte 的容器中：

```
int x = 24;  
byte b = x;  
// 不行!
```

为什么不行呢？毕竟byte绝对装得下24这个值。你知、我知，大家都知道这回事，但对编译器来说，你正在将大物体装进小容器中，所以会有溢位的可能。就算你能够用肉眼辨别出这是安全的，但别期待编译器会看着办。

你可以用几种方式来给变量赋值：

- 在等号后面直接打出 (x=12, isGood = true)。
- 指派其他变量的值 (x = y)。
- 上述两种方式的组合 (x = y + 43)。

下面粗体字部分是直接打出值的例子：

```
int size = 32;           int类型的32，名称为size  
char initial = 'j';    char类型的‘j’，名称为initial  
double d = 456.709;     double类型的456.709，名称为d  
boolean isCrazy;         只声明名称为isCrazy的boolean变量，未给值  
isCrazy = true;        赋true值  
int y = x + 456;       名称为y的int类型变量，其值为x与456相加运算的结果
```



编译器不允许将大杯的内容放到小杯中，但反过来呢？可以。

请根据你所知道的 primitive 主数据类型变量类型大小，判断下列哪些赋值是合法的，哪些不合法。因为目前还没有说明所有的规则，所以你得加上自己的判断。提示：编译器在安全性的问题上比较保守。

圈出合法的述句：

1. int x = 34.5;
2. boolean boo = x;
3. int g = 17;
4. int y = g;
5. y = y + 10;
6. short s;
7. s = y;
8. byte b = 3;
9. byte v = b;
10. short n = 12;
11. v = n;
12. byte k = 128;
13. int p = 3 * g + y;

避开关键字 (keyword) !

你已经知道变量需要名称和类型。

你已经知道什么是primitive主数据类型。

但是你知道命名的方法吗？很简单，你可以根据以下规则来帮助类、方法或变量命名（真正的规则在实际上更为复杂，但这些规则就能够保证安全）：

- 名称必须以字母、下划线（_）或 \$ 符号开头，不能用数字开头。
- 除了第一个字符之外，后面就可以用数字。反正不要用在第一个字符就行。
- 只要符合上述两条规则，你就可以随意地命名，但还得要避开 Java 的保留字。

保留字是编译器要辨别的关键字。如果你想要恶搞编译器，就试试看用保留字来命名。

你已经在编写第一个main()的时候就看过几个保留字了：

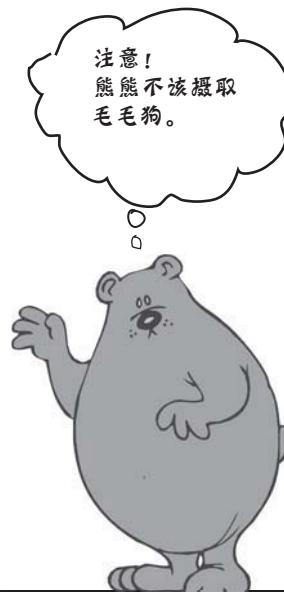
```
public static void
```

不要使用这些字词来
命名你写的东西

而primitive主数据的保留字如下：

```
boolean char byte short int long float double
```

但还有许多我们尚未讨论到的关键字。就算你不需要知道它们的意思，但还是必须清楚不能使用这些字词。现在不论在任何情况下都不要背诵下列的保留字表。留些空间给你的大脑，不然一定会忘记掉其他事情，比如你忘了把车停在哪里。学习到一定程度之后你就自然会记得了。



保留字一览表

boolean	byte	char	double	float	int	long	short	public	private
protected	abstract	final	native	static	strictfp	synchronized	transient	volatile	if
else	do	while	switch	case	default	for	break	continue	assert
class	extends	implements	import	instanceof	interface	new	package	super	this
catch	finally	try	throw	throws	return	void	const	goto	enum

这些是Java的保留字，如果你把它们用在名称上面，编译器会列出混乱的结果。

对象引用

控制Dog对象

你已经知道要如何声明primitive主数据类型变量并赋值给它。但非primitive主数据类型的变量又该如何处理呢？换句话说，对象要怎么处理？

- 事实上没有对象变量这样的东西存在。
- 只有引用（reference）到对象的变量。
- 对象引用变量保存的是存取对象的方法。
- 它并不是对象的容器，而是类似指向对象的指针。或者可以说是地址。但在Java中我们不会也不该知道引用变量中实际装载的是什么，它只是用来代表单一的对象。只有Java虚拟机才会知道如何使用引用来取得该对象。

你无法将对象装进变量中。我们通常会认为说：“我把一个String传入System.out.println()这个方法中”，或者“此方法会返回一个Dog对象”，又或是“我将新创建的Foo对象放进myFoo这个变量中”。

实际情况并不是这样。并没有超巨型的杯子可以放大到能够装载所有的对象。对象只会存在于可回收垃圾的堆上！（本章稍后会有更多的说明）。

虽然primitive主数据类型变量是以字节来代表实际的变量值，但对象引用变量却是以字节来表示取得对象的方法。

你会使用圆点运算符（.）来对引用变量表示：“取得圆点前面的对象，然后求出该对象在圆点后面的事物”。举例来说：

```
myDog.bark();
```

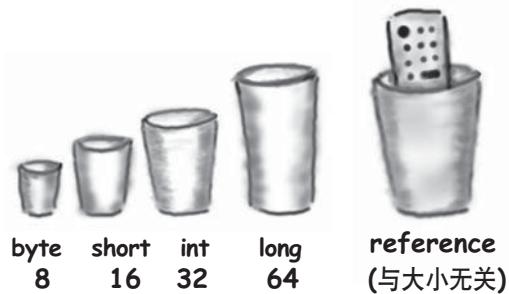
代表名为myDog的变量引用对象上的bark()。你可以把它想成遥控器与上面的按钮。

**Dog d = new Dog();
d.bark();**

把它想成遥控器



把Dog的引用变量
想成是Dog的遥控
器。你可以通过
它来执行工作。



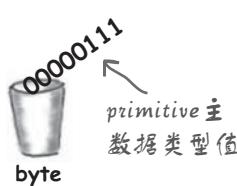
对象引用也只是个变量值

还是会有东西放进杯子中，只是引用所放进去的是遥控器。

Primitive主数据类型变量

`byte x = 7;`

代表数值七的字节被放进变量中(00000111).



引用变量

`Dog myDog = new Dog();`

代表取得 Dog 对象的方法以字节形式放进变量中。

对象本身并没有放进变量中！



对 primitive 主数据类型中的变量来说，变量值就是所代表的值（如 5、-26.7 或 ‘a’）。对引用变量来说，变量值是取得特定对象的位表示法。

你不会知道或在乎某个 Java 虚拟机是如何实现对象引用的。它们当然有可能是指向指针的指针……就算你知道，也无法使用这些字节来实现存取对象以外其他的操作。

我们也不在乎引用变量占用多少个 0 与 1。这与 JVM 以及当时九大行星的排列有关。

对象的声明、创建与赋值有
3个步骤：

1 `Dog myDog` 3 `= new Dog();` 2

1 声明一个引用变量

`Dog myDog = new Dog();`

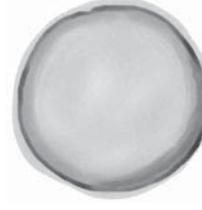
要求 Java 虚拟机分配空间给引用变量，并将此变量命名为 myDog。此引用变量将永远被固定为 Dog 类型。换句话说，它是个控制 Dog 的遥控器，不会是 Cat 或 皮卡丘的遥控器。



2 创建对象

`Dog myDog = new Dog();`

要求 Java 虚拟机分配堆空间给新建立的 Dog 对象（在后面，特别是第 9 章，我们会有更详细的讨论）。



Dog 对象

3 连接对象和引用

`Dog myDog = new Dog();`

将新的 Dog 赋值给 myDog 这个引用变量。换言之就是设定遥控器。



对象引用

there are no
Dumb Questions



Java Exposed

本周的来宾：对象引用（Object Reference）

问： 引用变量有多大？

答： 不知道。除非你跟某个Java虚拟机开发团队的人有交情，不然你是不会知道引用是如何表示的。其内部有指针，但你无法也需要存取。若你是要讨论内存分配的问题时，最需要关心的应该是需要建立多少个对象和引用，以及对象的实际大小。

HeadFirst: 请跟听众说一下对象引用的生活如何？

Reference: 相当单纯，真的。我只是个遥控器，并且可以被设定来控制不同的对象。

HeadFirst: 你是说在运行期间也能控制不同的对象吗？像是引用到狗对象的5分钟后又去引用皮卡丘对象？

Reference: 当然不是了。被声明成什么我就是什么。如果我是个Dog遥控器，就不能指向……啊，对不起，我是说引用到Dog以外的事物。

HeadFirst: 你是说你只能引用单一的Dog？

Reference: 错了，我可以引用某个Dog，5分钟后又去引用另外一个Dog。只要是Dog就行，因为我可以被转换，就像重新设定遥控器一样。除非……算了。

HeadFirst: 说呀，勇敢地说出来。

Reference: 说完天都亮了……先简单说一下好了，如果我被标记成final的话，一旦被指派给某个Dog之后我就不能赋值给这个特定Dog之外的任何事物。也就是说被固定下来了。

HeadFirst: 很好，我还真地不想现在就听到很长很长的故事。那么你能够引用到空指针吗？可以不引用任何东西吗？

Reference: 是可以的，但是我不想谈这个。

HeadFirst: 为什么？

Reference: 这样我就会是个null，这让我很不爽。

HeadFirst: 你是说没有值会让你很不爽就对了？

Reference: null也是个值。这就像你去买个万用遥控器回家，但是家里没有电视。这会让我觉得只是在浪费生命与位数，毫无意义。更糟的是，如果我是某个对象的唯一引用却又被设定成null，这意味着之后将没有其他人能够取得该对象。

HeadFirst: 这会造成困扰吗？

Reference: 还用说吗？我跟某个对象发生关系、有亲密的结合，然后突然间我就再也见不到这个对象，只为了它会被资源处理、垃圾回收。编写程序的人都不会考虑这么多，为什么我就不能当个primitive主数据类型，为什么让我吃到这么好吃的叉烧饭，万一以后吃不到怎么办……（访谈中断）

问： 我可以对引用变量进行运算吗，就像C语言那样？

答： 不行。请跟我重复念一万遍：“Java不是C”。

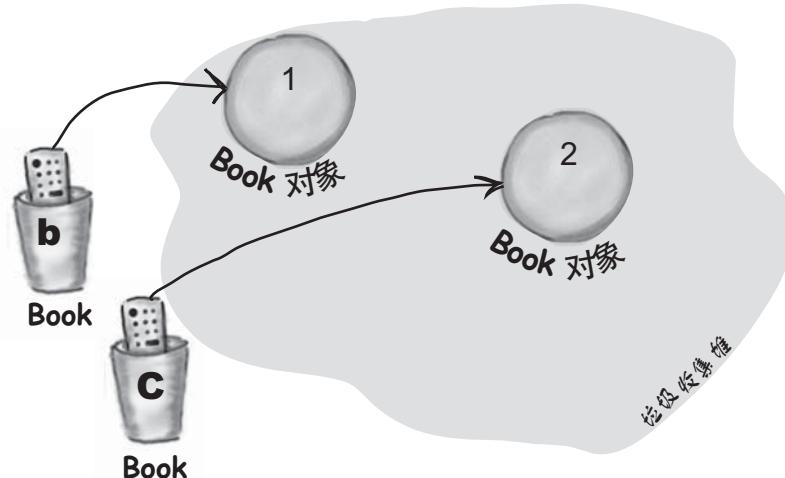
在垃圾收集堆上的生活

```
Book b = new Book();
Book c = new Book();
```

声明两个Book的引用变量并创建两个Book对象，然后将Book对象赋值给引用变量。现在这两个Book对象生活在堆上。

引用数：2

对象数：2



Book d = c;

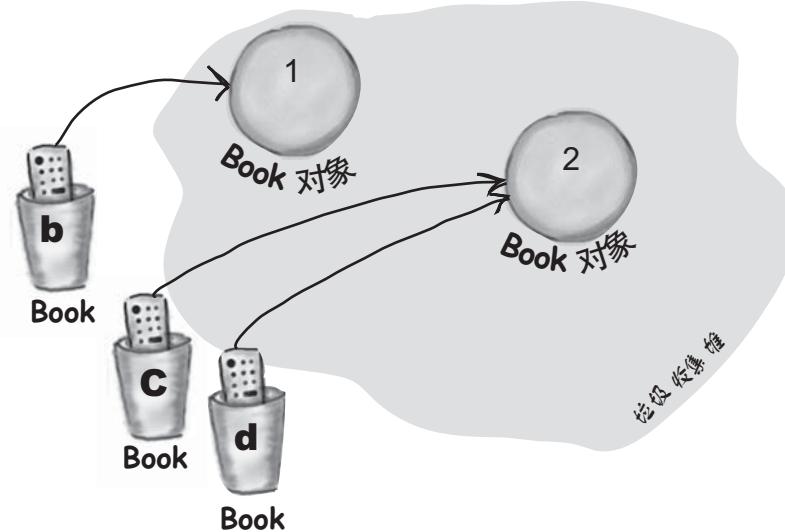
声明新的Book引用变量，但不创建新的Book对象而将变量c的值赋值给变量d。这代表“将c的字节组合拷贝给变量d”。

c与d引用到同一对象。

相同值的两份拷贝。一台电视两个遥控器。

引用数：3

对象数：2



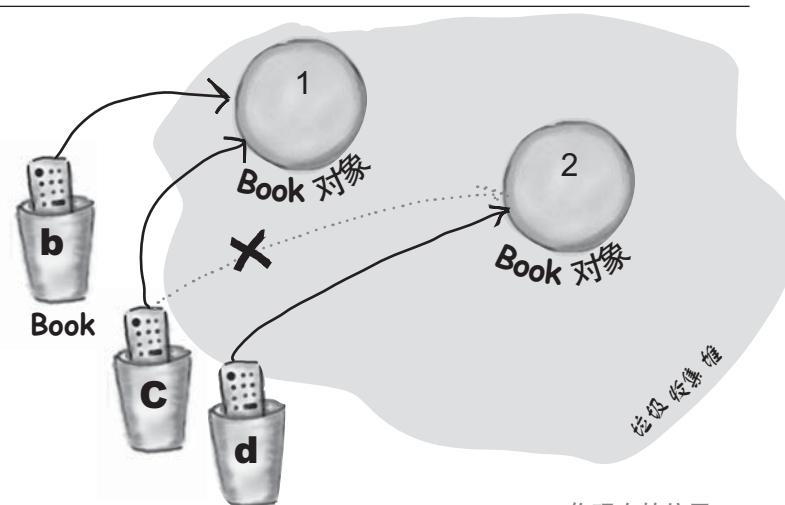
c = b;

把变量b的值赋给变量c。现在你知道这代表什么了。变量b的字节组合被拷贝一份给c。

b与c两者都引用相同的对象。

引用数：3

对象数：2



堆空间

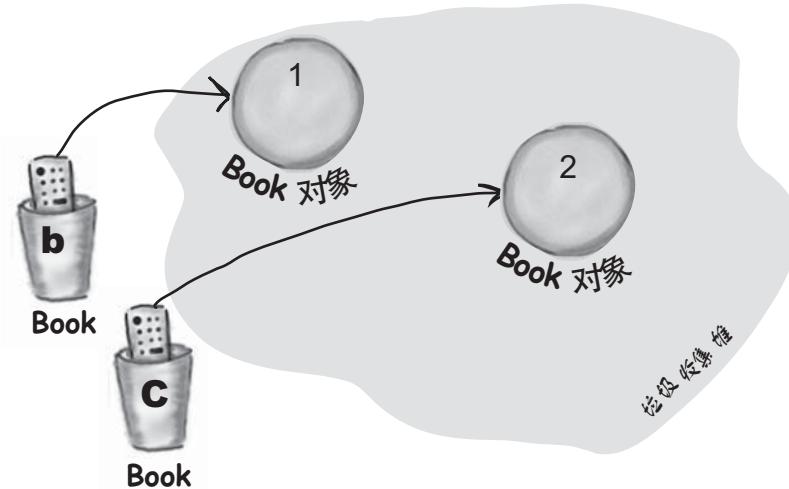
堆上的生与死

```
Book b = new Book();  
Book c = new Book();
```

声明两个Book的引用变量并创建两个Book对象，然后将Book对象赋值给引用变量。现在这两个Book对象生活在堆上。

引用数：2

对象数：2



```
b = c;
```

把变量c的值赋给变量b。两者带有相同的值。

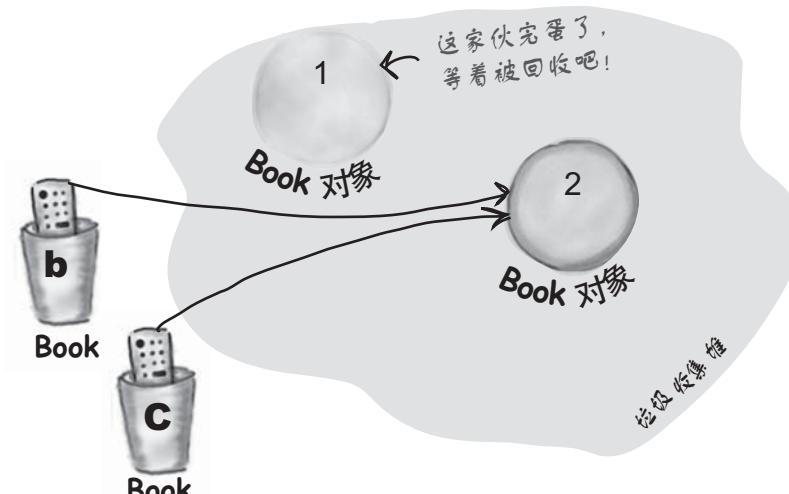
对象1被抛弃且能够作垃圾收集器(GC)。

引用数：2

对象数：2

被抛弃对象数：1

对象1已经没有引用，变成无法存取的。



```
c = null;
```

将null值赋给c。这代表它不再引用任何事物，但还是个可以被指定引用其他Book的引用变量。

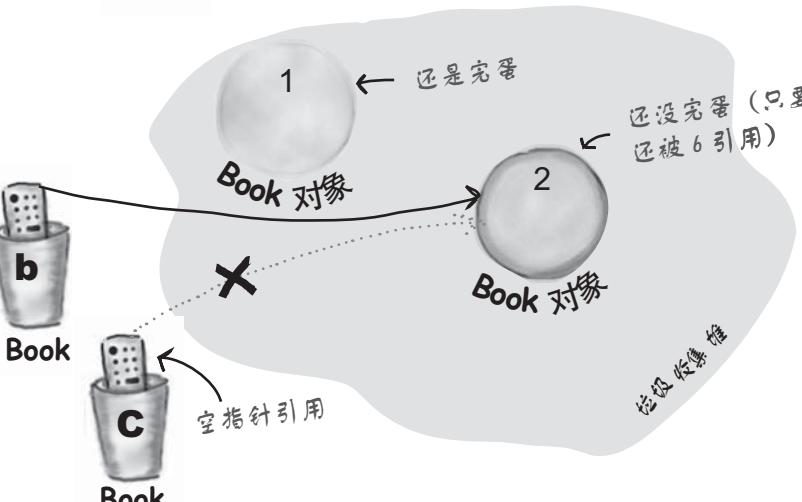
对象2还引用到，所以不能够作垃圾收集器(GC)。

作用中的引用数：1

null引用数：1

可存取对象数：1

被抛弃对象数：1



数组犹如杯架

- 1** 声明一个int数组变量。数组变量是数组对象的遥控器。

```
int[] nums;
```

- 2** 创建大小为7的数组，并将它赋值给之前声明为int[]的变量nums。

```
nums = new int[7];
```

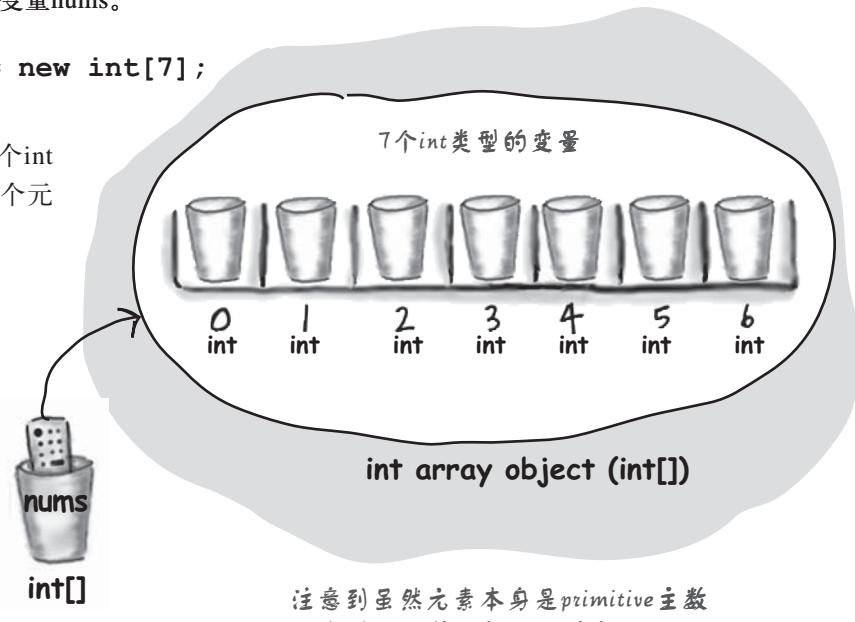
- 3** 赋予int数组的每一个元素一个int值。要记得在int数组中的每个元素皆为int类型的变量。

```
nums[0] = 6;
nums[1] = 19;
nums[2] = 44;
nums[3] = 42;
nums[4] = 10;
nums[5] = 20;
nums[6] = 1;
```

数组也是对象

Java的标准函数库包含了许多复杂的数据结构，比如map、tree和set（见附录B），但如果需要快速、有序、有效率地排列元素时，数组是不错的选择。数组能够让你使用位置索引来快速、随机地存取其中的元素。

数组中的每个元素都是变量。换言之，会是8种primitive主数据类型变量中的1种，不然就是引用变量。可以放进该类型变量中的值都可以当作此类型数组的元素。所以在int类型的数组中，每个元素可以装载一个int。所以在Dog的数组中(Dog[])每个可以装载一个Dog吗？错，要记得引用变量只会保存引用，而不是对象本身。因此Dog数组的元素持有的是Dog的遥控器。当然啦，我们还得创建Dog对象，下一页会来执行这个动作。



注意到虽然元素本身是primitive主数据类型，但数组却是个对象。

在上面的图中有一项要注意的：数组是个对象，不管里面放的是不是primitive主数据类型。

无论被声明来承载的是primitive主数据类型或对象引用，数组永远是对象。但你可以声明出可以装载primitive主数据类型值的数组。换句话说，数组对象可以有primitive主数据类型的元素，但数组本身绝对不会是primitive主数据类型。不管数组带有什么，它一定是对象！

对象的数组

创建Dog数组

- 1 声明一个Dog数组变量。

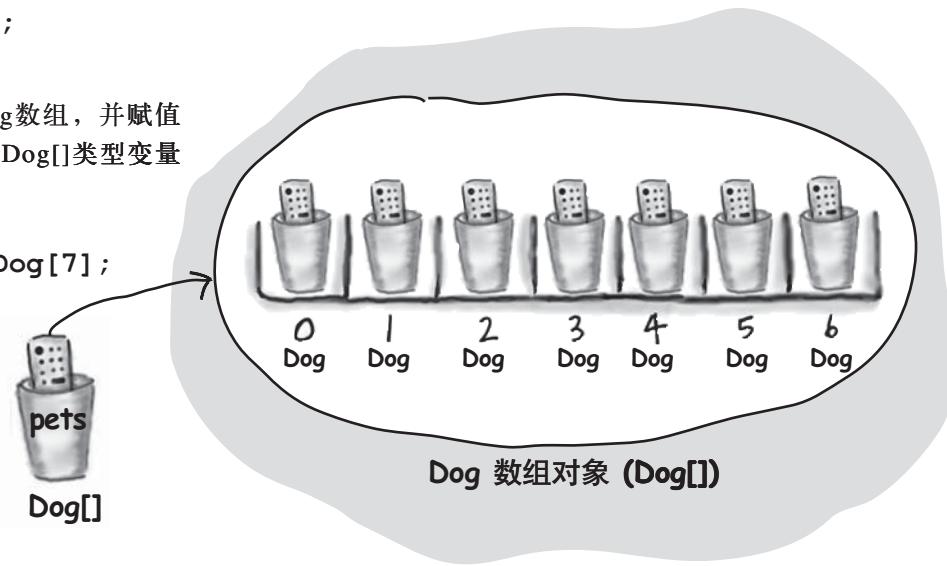
```
Dog[] pets;
```

- 2 创建大小为7的Dog数组，并赋值给前面所声明出的Dog[]类型变量pets。

```
pets = new Dog[7];
```

少了什么？

少了Dog！我们虽然有了对Dog的引用，但缺少实际的Dog对象！



- 3 创建新的Dog对象并将它们赋值给数组的元素。

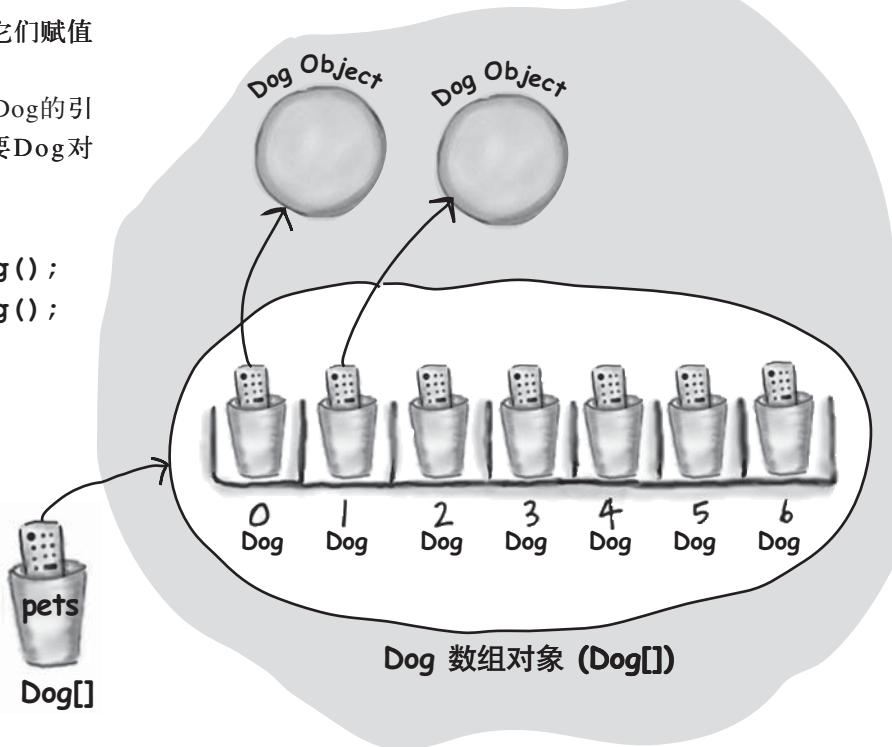
记得Dog数组中只带有Dog的引用变量。我们还是需要Dog对象！

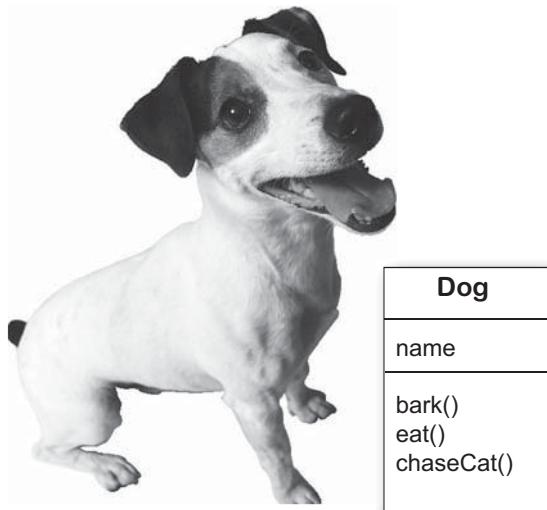
```
pets[0] = new Dog();  
pets[1] = new Dog();
```

Sharpen your pencil

pets[2]目前的值是什么？_____

如何指派pets[3]引用到既有的Dog对象？_____





Java注重类型

一旦数组被声明出来，你就只能装入所声明类型的元素。

举例来说，你不能将Cat放到Dog数组中（如果有人尝试要让数组中的每个元素都汪汪叫一次会出现什么状况？）。素都汪汪叫一次会出现什么状况？）。double也不能放进int数组中。但是你可以将byte放进int的数组中，因为byte可以放进int尺寸的杯子中。这被称为隐含展开（implicit widening，稍后会有更多的说明，现在只需要注意编译器会根据数组声明的类型来防止错误的类型）。

控制Dog

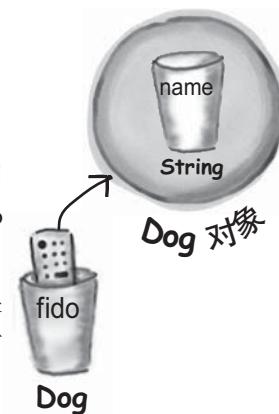
（通过引用变量）

```
Dog fido = new Dog();
fido.name = "Fido";
```

我们创建出Dog对象并使用圆点运算符来操作引用变量fido并存取它的name变量。

我们可以运用fido这个引用来让Dog执行bark()或其他的方法。

```
fido.bark();
fido.chaseCat();
```



如何存取 Dog 数组中的 Dog?

我们知道可以通过圆点运算符来存取 Dog 的实例变量与方法，但数组呢？

我们对数组的操作可以不需要变量名称。只需要数组索引（位置）就可以操作特定对象了：

```
Dog[] myDogs = new Dog[3];
myDogs[0] = new Dog();
myDogs[0].name = "Fido";
myDogs[0].bark();
```

*此处的说明还未运用到封装的概念，我们会在第4章加以讨论。

使用引用

```
class Dog {  
    String name;  
    public static void main (String[] args) {  
        // 创建Dog对象  
        Dog dog1 = new Dog();  
        dog1.bark();  
        dog1.name = "Bart";  
  
        // 创建Dog数组  
        Dog[] myDogs = new Dog[3];  
        // 关门放狗  
        myDogs[0] = new Dog();  
        myDogs[1] = new Dog();  
        myDogs[2] = dog1;  
  
        // 通过数组引用存取Dog  
        myDogs[0].name = "Fred";  
        myDogs[1].name = "Marge";  
  
        // myDog[2]的名字是?  
        System.out.print("last dog's name is ");  
        System.out.println(myDogs[2].name);  
  
        // 逐个对Dog执行bark()  
        int x = 0;  
        while(x < myDogs.length){  
            myDogs[x].bark();  
            x = x + 1;  
        }  
  
        public void bark() {  
            System.out.println(name + " says Ruff!");  
        }  
        public void eat() {}  
        public void chaseCat() {}  
    }
```

数组有个称为length的变量
能够返回元素的数目

Dog 的范例

Dog
name
bark() eat() chaseCat()

输出

```
File Edit Window Help Howl  
%java Dog  
null says Ruff!  
last dog's name is Bart  
Fred says Ruff!  
Marge says Ruff!  
Bart says Ruff!
```



要点

- 变量有两种： primitive主数据类型和引用。
- 变量的声明必须有类型和名称。
- primitive主数据类型变量值是该值的字节所表示的。
- 引用变量的值代表位于堆之对象的存取方法。
- 引用变量如同遥控器，对引用变量使用圆点运算符可以如同按下遥控器按钮般地存取它的方法或实例变量。
- 没有引用到任何对象的引用变量的值为 null值。
- 数组一定是个对象，不管所声明的元素是否为 primitive主数据类型，并且没有 primitive主数据类型的数组，只有装载 primitive主数据类型的数组。



我是编译器



这一页的Java程序代码都代表一份完整的源文件。你的任务是要扮演编译器角色并判断哪支程序可以编译过关。如果有问题，哪里要修改？

A

```
class Books {
    String title;
    String author;
}

class BooksTestDrive {
    public static void main(String [] args) {

        Books [] myBooks = new Books[3];
        int x = 0;
        myBooks[0].title = "The Grapes of Java";
        myBooks[1].title = "The Java Gatsby";
        myBooks[2].title = "The Java Cookbook";
        myBooks[0].author = "bob";
        myBooks[1].author = "sue";
        myBooks[2].author = "ian";

        while (x < 3) {
            System.out.print(myBooks[x].title);
            System.out.print(" by ");
            System.out.println(myBooks[x].author);
            x = x + 1;
        }
    }
}
```

B

```
class Hobbits {

    String name;

    public static void main(String [] args) {

        Hobbits [] h = new Hobbits[3];
        int z = 0;

        while (z < 4) {
            z = z + 1;
            h[z] = new Hobbits();
            h[z].name = "bilbo";
            if (z == 1) {
                h[z].name = "frodo";
            }
            if (z == 2) {
                h[z].name = "sam";
            }
            System.out.print(h[z].name + " is a ");
            System.out.println("good Hobbit name");
        }
    }
}
```

习题



排排看



右边是被打散的Java程序片段，你是否能够将它们重新排列以成为可以编译与运行并产生如同下方的输出结果？注意到有些括号已经遗失，所以你可以在认为有需要时自行补上。

```
int y = 0;  
ref = index[y];  
  
islands[0] = "Bermuda";  
islands[1] = "Fiji";  
islands[2] = "Azores";  
islands[3] = "Cozumel";  
  
int ref;  
while (y < 4) {  
  
    System.out.println(islands[ref]);
```

```
index[0] = 1;  
index[1] = 3;  
index[2] = 0;  
index[3] = 2;  
  
String [] islands = new String[4];  
  
System.out.print("island = ");  
  
int [] index = new int[4];  
y = y + 1;
```

```
File Edit Window Help Bikini  
% java TestArrays  
island = Fiji  
island = Cozumel  
island = Bermuda  
island = Azores
```

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



泳池迷宫



你的任务是要从游泳池中挑出程序片段并将它填入右边的空格中。
同一个片段不能用两次，且泳池中有些多余的片段。填完空格的程序必须要能够编译与执行并产生出下面的输出。

输出：

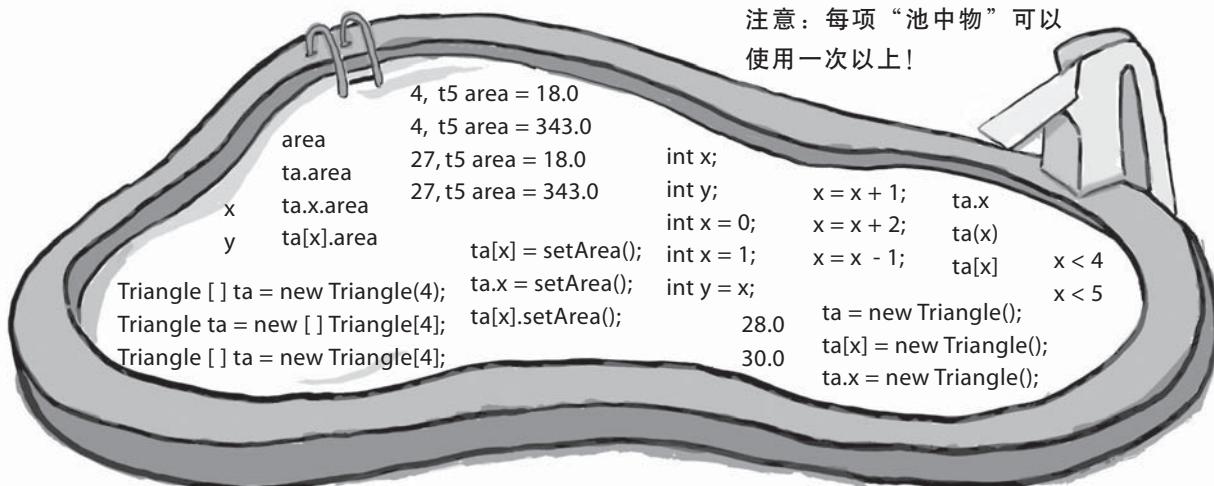
```
File Edit Window Help Bermuda
%java Triangle
triangle 0, area = 4.0
triangle 1, area = 10.0
triangle 2, area = 18.0
triangle 3, area = _____
y = _____
```

加分题！

从池中找出可以填在输出空格部分的片段。

有时我们会为了节省空间而没有采用分离的测试类

```
class Triangle {
    double area;
    int height;
    int length;
    public static void main(String [] args) {
        _____
        while ( _____ ) {
            _____.height = (x + 1) * 2;
            _____.length = x + 4;
            System.out.print("triangle "+x+", area");
            System.out.println(" = " + _____ .area);
        }
        _____
        x = 27;
        Triangle t5 = ta[2];
        ta[2].area = 343;
        System.out.print("y = " + y);
        System.out.println(", t5 area = "+ t5.area);
    }
    void setArea() {
        _____ = (height * length) / 2;
    }
}
```



迷题



连连看

右边有一段Java小程序。执行到“// do stuff”这一行时已经创建一些对象与引用变量。你的工作是要判别哪个引用变量引用到哪个对象。引用变量不一定用到，对象也可能被多个变量引用。对有引用关系的对象画线连接起来。

提示：你可以参考第55页与第56页的做法。

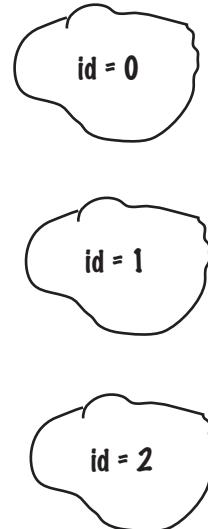
```
class HeapQuiz {  
    int id = 0;  
    public static void main(String [] args) {  
        int x = 0;  
        HeapQuiz [ ] hq = new HeapQuiz[5];  
        while ( x < 3 ) {  
            hq[x] = new HeapQuiz();  
            hq[x].id = x;  
            x = x + 1;  
        }  
        hq[3] = hq[1];  
        hq[4] = hq[1];  
        hq[3] = null;  
        hq[4] = hq[0];  
        hq[0] = hq[3];  
        hq[3] = hq[2];  
        hq[2] = hq[0];  
        // do stuff  
    }  
}
```

将有引用关系的变量与
对象画线连接起来。

引用变量:



堆对象:





阿强少年事件簿：在密室中消失的引用

在一个风雨交加的夜晚，业务部的阿美走进工程部“巡逻”。她知道程序员都还在加班，因为是她要求他们留下来赶一个程序的。她得交给客户一个移动电话用的Java通讯簿管理系统。每个人都知道移动电话的堆内存就跟阿美看得上眼的男人一样少的可怜。就在她突然走向白板时，喧闹的办公室马上就安静下来，因为大家知道她又要增加功能了。她很快在白板上画好新方法的概要图并放下笔来慢慢地扫视全场说到：“帅哥们，来吧，能给我在内存上最有效率方法的人明天就能跟我去夏威夷……帮客户安装程序”。

悬疑！紧张！刺激！到底谁是凶手？到底有没有凶手？



第二天早上，天气晴朗，鸟语花香，阿美穿着夏威夷草裙溜进工程部。“早安啊，帅哥们”，她满脸笑容地说，“谁要跟我远走高飞呢？”暗恋阿美已久的阿强第一个跳到白板前准备展示他熬夜出来的成果。阿美说：“先让我看你怎么处理联络人对象的更新”。阿强很快就把程序展示出来：

```
Contact [] ca = new Contact[10];
while ( x < 10 ) { // 创建10个contact对象
    ca[x] = new Contact();
    x = x + 1;
}
// 执行其余复杂的更新工作
```

“这就是我写的方法”阿强显然对这个方法很满意。接着小明也跳出来，他对阿强说“你不觉得你的写法有点问题吗？”回头又对阿美说“宝贝，看完程序我们就走好吗？”：

```
Contact refc;
while ( x < 10 ) { // 创建10个contact对象
    refc = new Contact();
    x = x + 1;
}
// 执行其余复杂的更新工作
```

“这样写才可以省下引用变量用的宝贵内存啊，学着点……”，小明以胜利者的姿态对着阿强说：“等小孩满月时一定要来啊”。阿美却不这么认为：“小明，你等下辈子吧，阿强我们走，登机前还可以先去喝个饮料……”边说边拉着阿强往等在公司门口的接送车走去。

为什么阿美选择了阿强而不是内存耗用比较少的小明？最后阿强会得逞吗？小明还有什么办法可以从中破坏两人的感情呢？

习题解答



解答

排排看：

```
class TestArrays {  
    public static void main(String [] args) {  
        int [] index = new int[4];  
        index[0] = 1;  
        index[1] = 3;  
        index[2] = 0;  
        index[3] = 2;  
        String [] islands = new String[4];  
        islands[0] = "Bermuda";  
        islands[1] = "Fiji";  
        islands[2] = "Azores";  
        islands[3] = "Cozumel";  
        int y = 0;  
        int ref;  
        while (y < 4) {  
            ref = index[y];  
            System.out.print("island = ");  
            System.out.println(islands[ref]);  
            y = y + 1;  
        }  
    }  
}
```

File Edit Window Help Bikini
% java TestArrays
island = Fiji
island = Cozumel
island = Bermuda
island = Azores

A

```
class Books {  
    String title;  
    String author;  
}  
class BooksTestDrive {  
    public static void main(String [] args) {  
        Books [] myBooks = new Books[3];  
        int x = 0;  
        myBooks[0] = new Books();  
        myBooks[1] = new Books();  
        myBooks[2] = new Books();  
        myBooks[0].title = "The Grapes of Java";  
        myBooks[1].title = "The Java Gatsby";  
        myBooks[2].title = "The Java Cookbook";  
        myBooks[0].author = "bob";  
        myBooks[1].author = "sue";  
        myBooks[2].author = "ian";  
        while (x < 3) {  
            System.out.print(myBooks[x].title);  
            System.out.print(" by ");  
            System.out.println(myBooks[x].author);  
            x = x + 1;  
        }  
    }  
}
```

B

```
class Hobbits {  
    String name;  
}  
public static void main(String [] args) {  
    Hobbits [] h = new Hobbits[3];  
    int z = -1;  
    while (z < 2) {  
        z = z + 1;  
        h[z] = new Hobbits();  
        h[z].name = "bilbo";  
        if (z == 1) {  
            h[z].name = "frodo";  
        }  
        if (z == 2) {  
            h[z].name = "sam";  
        }  
        System.out.print(h[z].name + " is a ");  
        System.out.println("good Hobbit name");  
    }  
}
```



迷宫解答

```

class Triangle {
    double area;
    int height;
    int length;
    public static void main(String [] args) {
        int x = 0;
        Triangle [ ] ta = new Triangle[4];
        while ( x < 4 ) {
            ta[x] = new Triangle();
            ta[x].height = (x + 1) * 2;
            ta[x].length = x + 4;
            ta[x].setArea();
            System.out.print("triangle "+x+", area");
            System.out.println(" = " + ta[x].area);
            x = x + 1;
        }
        int y = x;
        x = 27;
        Triangle t5 = ta[2];
        ta[2].area = 343;
        System.out.print("y = " + y);
        System.out.println(", t5 area = "+ t5.area);
    }
    void setArea() {
        area = (height * length) / 2;
    }
}

```

```

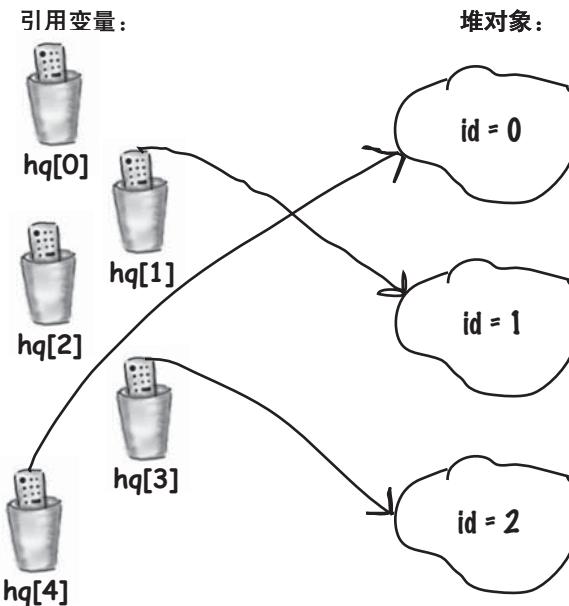
File Edit Window Help Bermuda
%java Triangle
triangle 0, area = 4.0
triangle 1, area = 10.0
triangle 2, area = 18.0
triangle 3, area = 28.0
y = 4, t5 area = 343

```

在密室中消失的引用

因为阿美看出来小明的方法有个重大缺陷。小明所占用的内存是比较少没错，但是除了最后一个Contact对象外其他的都没有办法存取。因为从头到尾只有一个引用变量，此变量最后只能引用到最新建立出的对象。因此小明的程序代码根本不能用。

(至于后来在夏威夷发生了什么事，结果又会变得怎么样，阿美是否会出车祸丧失记忆，阿强是不是私生子，小明会不会被外星人掳走，以上这些问题连我们也没有答案……)



4 方法操作实例变量

对象的行为



危险动作，请勿模仿

(除非你有自残的念头)

状态影响行为，行为影响状态。 我们已经知道对象有状态和行为两种属性，分别由实例变量与方法来表示。但我们还没有看到两者之间的关联。我们已经知道类的每个实例（也就是特定类型的每个对象）可以维持自己的实例变量。某个Dog的名称是“Fido”质量有20kg。另外一个Dog名称为“Killer”，质量有3kg。如果Dog这个类带有一个makeNoise()方法，你觉得哪一只的吠声会比较低沉呢？（假设呜呜两声也算吠的话）。幸好这就是面向对象的重点——行为会依据状态来决定。换句话说，方法会使用到实例变量的值。比如说：“如果狗的质量超过8kg，就发出呜呜的声音，否则……”或是“加2kg”。让我们奔向夕阳、改变状态吧！

对象有状态和行为

记住：类所描述的是对象知道什么与执行什么？

类是对象的蓝图。在编写类时，你是在描述Java虚拟机应该如何制作该类型的对象。你已经知道每个对象有独立的实例变量值。但方法呢？

同一类型的每个对象能够有不同的方法行为吗？

嗯……差不多*。

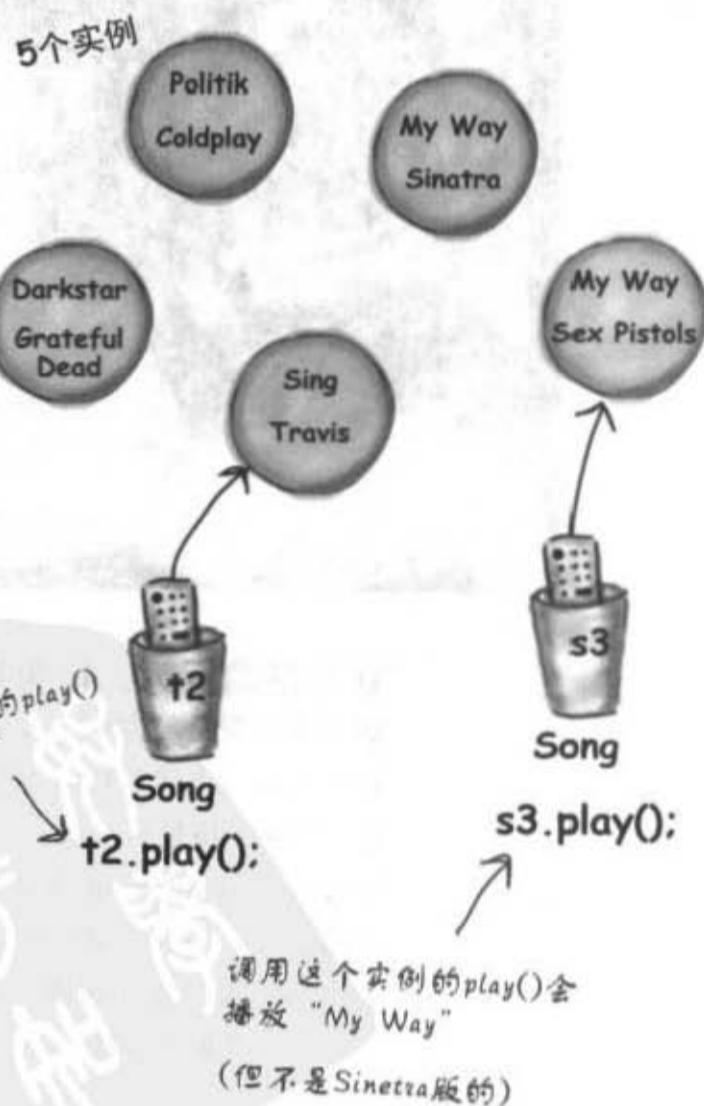
任一类的每个实例都带有相同的方法，但是方法可以根据实例变量的值来表现不同的行为。

Song这个类有title与artist这两个实例变量。play()会播放title值所表示的歌曲。所以调用某个实例的play()可能会播放“Politik”而另一个实例会播放“Darkstar”。然而方法却是相同的：

```
void play() {  
    soundPlayer.playSound(title);  
}
```

```
Song t2 = new Song();  
t2.setArtist("Travis");  
t2.setTitle("Sing");  
  
Song s3 = new Song();  
s3.setArtist("Sex Pistols");  
s3.setTitle("My Way");
```

Song	knows
title artist	
方法 (行为)	does
setTitle() setArtist() play()	



*无懈可击的回答！

大小影响叫声

小型犬的叫声与大型犬不同。

Dog这个类有个称为size的实例变量，bark()会用它来决定使用哪一种声音。

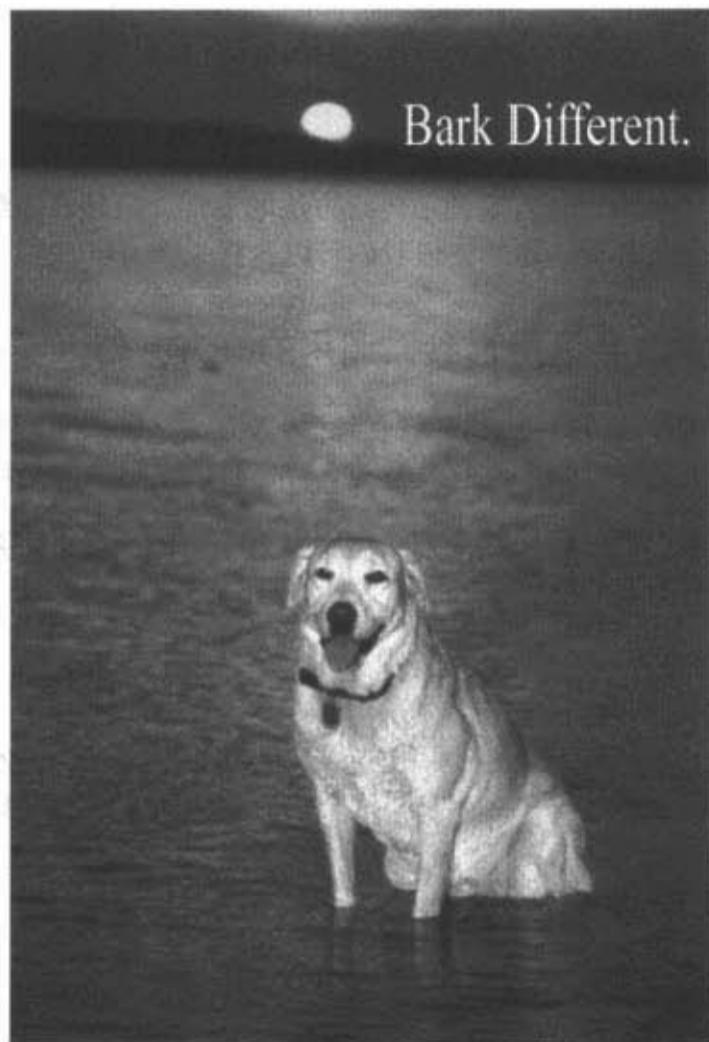
```
class Dog {
    int size;
    String name;

    void bark() {
        if (size > 60) {
            System.out.println("Wooof! Wooof!");
        } else if (size > 14) {
            System.out.println("Ruff! Ruff!");
        } else {
            System.out.println("Yip! Yip!");
        }
    }
}
```

```
class DogTestDrive {

    public static void main (String[] args) {
        Dog one = new Dog();
        one.size = 70;
        Dog two = new Dog();
        two.size = 8;
        Dog three = new Dog();
        three.size = 35;

        one.bark();
        two.bark();
        three.bark();
    }
}
```



你可以传值给方法

如同其他的程序设计语言，你可以传值给方法。举例来说，你可能会要告诉 Dog 对象叫几声：

```
d.bark(3);
```

由于不同的程序设计背景和个人喜好，你可能会用实参（argument）或形参（parameter）来调用传给方法的参数。虽然在正统学院派的信息工程领域中这两者是不同的，但我们可以这样来区分：

方法会运用形参。调用的一方会传入实参。

实参是传给方法的值。当它传入方法后就成了形参。参数跟局部（local）变量是一样的。它有类型与名称，可以在方法内运用。

重点是：如果某个方法需要参数，你就一定得传东西给它。那个东西得是适当类型的值。



你可以从方法中取返回值

方法可以有返回值。每个方法都声明返回的类型，但目前我们都是把方法设成返回 void 类型，这代表并没有返回任何东西。

```
void go() {  
}
```

但我们可以声明一个方法，回传给调用方指定的类型值，如：

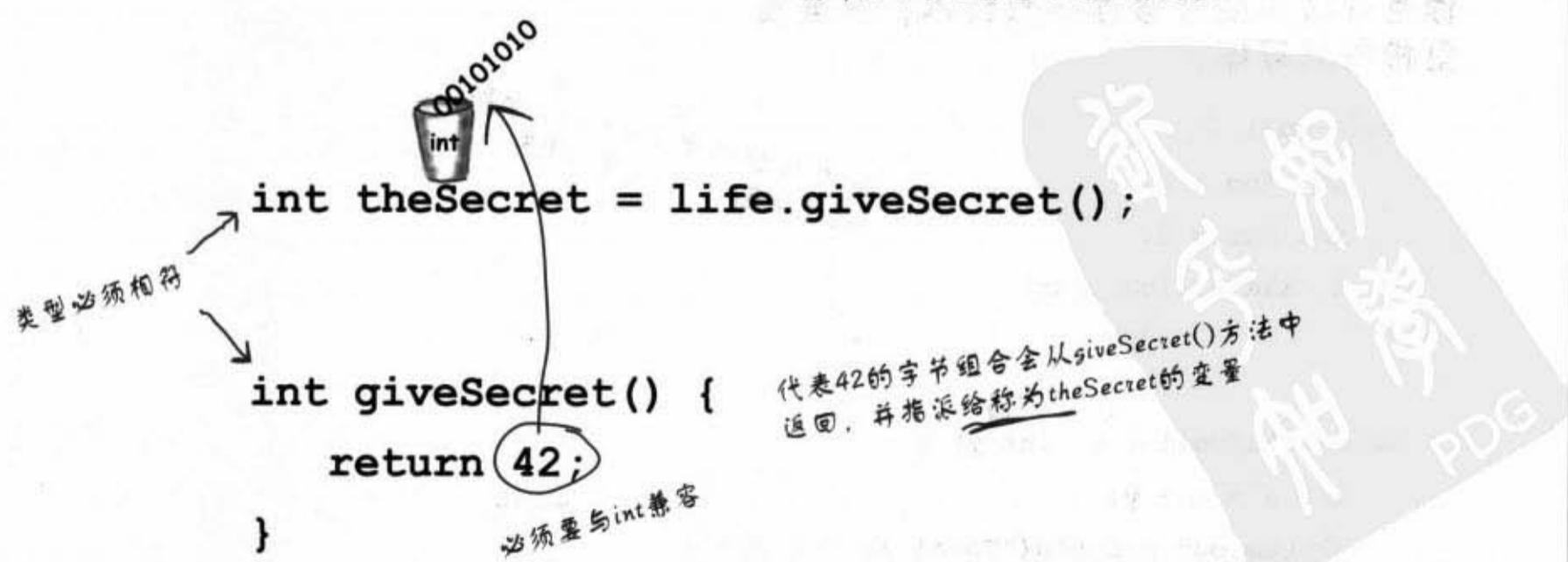
```
int giveSecret() {  
    return 42;  
}
```

如果你将一个方法声明有返回值，你就必须返回所声明类型的值！（或是与声明类型兼容的值。我们会在第7章与第8章讨论多态的时候提到更多的细节）。

说好了要返回，
最好就得返回！



编译器不会让你返回错误的类型



多个参数

你可以向方法中传入一个以上的参数

方法可以有多个参数。在声明的时候要用逗号分开，传入的时候也是用逗号分开。最重要的是，如果方法有参数，你一定要以正确数量、类型和顺序来传递参数。

调用需要两个参数的方法，并传入两个参数：

```
void go() {  
    TestStuff t = new TestStuff();  
    t.takeTwo(12, 34);  
}  
  
void takeTwo(int x, int y) {  
    int z = x + y;  
    System.out.println("Total is " + z);  
}
```

传入的参数会以相同的顺序赋值。
第一个实参会赋给第一个形参，依此类推

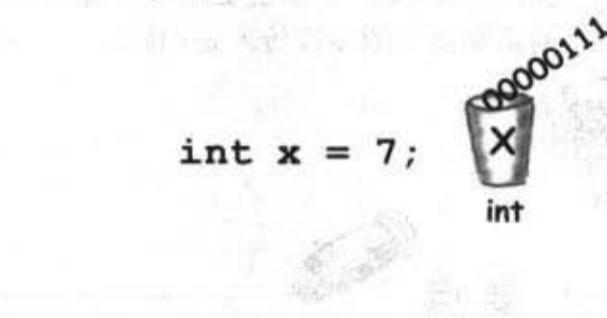
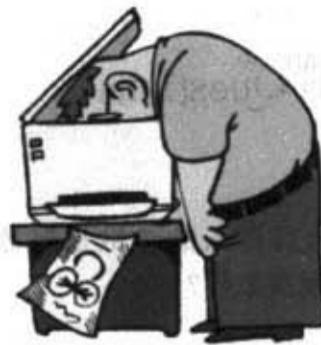
你也可以将变量当作参数传入，只要类型相符就可以：

```
void go() {  
    int foo = 7;  
    int bar = 3;  
    t.takeTwo(foo, bar);  
}  
  
void takeTwo(int x, int y) {  
    int z = x + y;  
    System.out.println("Total is " + z);  
}
```

foo与bar的值会赋给x与y，所以
x的值是7，而y的值是3

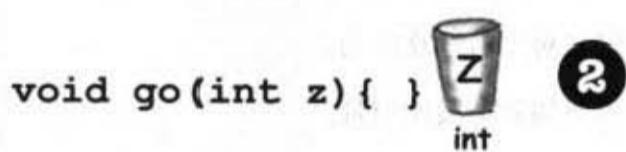
x的值是10

Java是通过值传递的
也就是说通过拷贝传递



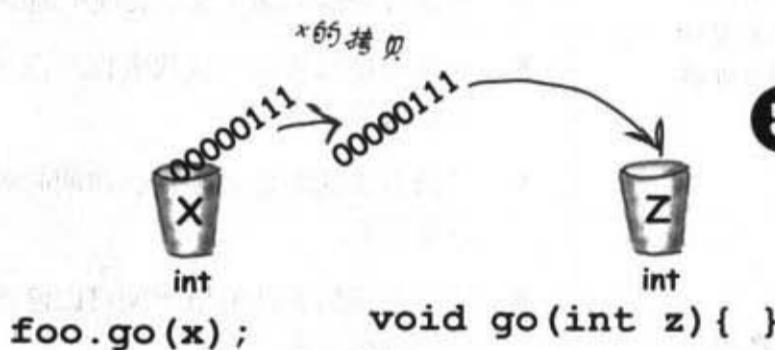
1

声明一个int类型的变量并赋值为7。代表7的字节组合会放进称为x的变量中。



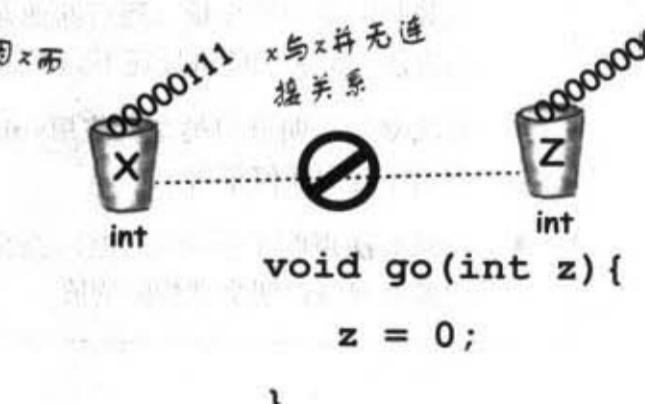
2

声明一个有int参数的方法，参数名称为z。



3

以x为参数传入go()这个方法中。x的字节组合会被拷贝并装进z中。



4

在方法中改变z的值。此时x的值不会改变！传给z的只是个拷贝。

方法无法改变调用方所传入的参数。

*there are no
Dumb Questions*

问：如果想要传入的参数是对象而不是 primitive 主数据类型会怎样？

答：你会在稍后的章节中知道有关这件事情更多的细节，但是你早就知道答案了。在 Java 中所传递的所有东西都是值，但此值是变量所携带的值。还有，引用对象的变量所携带的是远程控制而不是对象本身。若你对方法传入参数，实际上传入的是远程控制的拷贝。

问：方法可以声明多个返回值吗？有没有别的方法可以返回多个值？

答：方法只能声明单一的返回值。若你需要返回 3 个 int 值，就把返回类型说明为 int 的数组，将值装进数组中来返回。如果有混合不同类型的值要返回时，等我们稍后讨论到 ArrayList 时再说。

问：一定要返回所声明的类型吗？

答：你可以返回会被隐含转换成声明类型的其他类型值。例如说用 byte 当作 int 类型的返回。但若声明的类型容器小于想要返回的类型时，必须作明确的转换。

问：我可不可以忽略返回值？

答：Java 并未要求一定要处理返回值。你可以调用返回非 void 类型的方法而不必理会返回值，这代表你要的是方法的行为而不是返回值。你可以不指派返回值。

提醒你：
Java 注重类型！

当返回类型声明成兔子的时候你不能返回长颈鹿。参数也是这样。你不能对取用兔子的参数传入长颈鹿。



要点

- 类定义对象所知及所为。
- 对象所知者是实例变量。
- 对象所为者是方法。
- 方法可依据实例变量来展现不同的行为。
- 方法可使用参数，这代表你可以传入一个或多个值给方法。
- 传给方法的参数必须符合声明时的数量、顺序和类型。
- 传入与传出方法的值类型可以隐含地放大或是明确地缩小。
- 传给方法的参数值可以是直接指定的文字或数字（例如 2 或 ‘c’ 等）或者是与所声明参数相同类型的变量（还有其他东西可以传给方法，但我们的进度还不到那边）。
- 方法必须声明返回类型。使用 void 类型代表方法不返回任何东西。
- 如果方法声明了非 void 的返回类型，那就一定要返回与声明类型相同的值。

运用参数与返回类型

我们已经看过参数与返回类型的工作，接下来就要有效地利用了：来看Getter与Setter。如果要很正式地讨论，你会称他们为Accessors与Mutators。不过这样只是更绕舌而已，由于Getter与Setter较为符合Java的命名习惯，所以我们接下来都会这么叫它们。

Getter与Setter可让你执行get与set。Getter的目的只有一个，就是返回实例变量的值。毫无疑问的，Setter的目的就是要取用一个参数来设定实例变量的值。

```
class ElectricGuitar {

    String brand;
    int numOfPickups;
    boolean rockStarUsesIt;

    String getBrand() {
        return brand;
    }

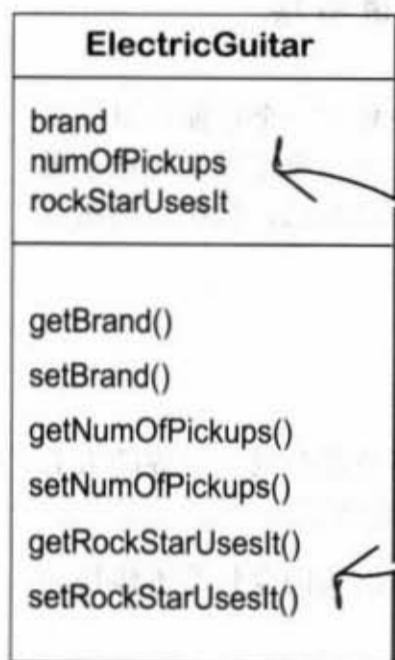
    void setBrand(String aBrand) {
        brand = aBrand;
    }

    int getNumOfPickups() {
        return numOfPickups;
    }

    void setNumOfPickups(int num) {
        numOfPickups = num;
    }

    boolean getRockStarUsesIt() {
        return rockStarUsesIt;
    }

    void setRockStarUsesIt(boolean yesOrNo) {
        rockStarUsesIt = yesOrNo;
    }
}
```



注意：你应该遵循Java的命名标准（naming convention）



封装 (Encapsulation)

不封装可能会很难堪

在此之前我们已经犯了一个在面向对象界最糟糕的错误（这可不像说被发现袜子破了一个洞这种小尴尬而已，我们说的错误可是严重的失礼）。

我们哪里有错呢？

泄露资料！

我们并没有注意到数据会被全世界的人看到，甚至还可以被改动。

你可能经历过暴露出实例变量的不愉快感觉。

暴露的意思是可通过圆点运算符来存取，像是：

```
tehCat.height = 27;
```

你可以把这件事情看做是直接通过远程控制修改Cat的实例变量。若远程控制落入不当之人的手上，变量就可能会成为杀伤力强大的武器。因为你无法防止下面的操作：

```
theCat.height = 0; ← 千万不能让这种事情发生！
```

这一定会很糟糕。所以我们需要创建Setter这个方法给所有的实例变量，并寻求某种方法强制其他程序都必须通过Setter来设定变量而不是直接的存取。



据查所有人都得调用Setter，我们就可以防止Cat被设置成无法接受的高度。

```
public void setHeight(int ht) {
```

```
    if (ht > 9) { ← 这个检查可以  
        height = ht; 确保高度不会  
    } 低于9
```

```
}
```

数据隐藏

要将程序的实现从不良数据改成可以保护数据且让你还能修改数据的方式是很简单的。

所以要如何隐藏数据呢？答案是使用公有与私有这两个存取修饰符（access modifier）。

以下就是封装的基本原则：将你的实例变量标记为私有的，并提供公有的getter与setter来控制存取动作。或许在你有了更多的Java设计与编写经验之后会有些许不同的做法，但是目前这种做法可以维持住安全性。

**将实例变量标记为
private。**

将getters与setters标记为public。

“老王忘记把他的猫封装，后来他的猫就被辗平了……”

（在捷运站听到的鬼故事）



本周的来宾：一个即将被封装的对象引用

HeadFirst： 封装有什么本事？

Object： 嗯，你有没有梦到过面对500个听众时，突然发觉自己没有穿裤子？

HeadFirst： 是有一次。我梦到跟一群模特儿在后宫嬉戏，然后我的裤子……呃，先不谈这个。OK，所以你是说没有封装就像没穿裤子。但是没有露一点出来会不会很不舒服？

Object： 不会吧，大哥。不舒服？很不舒服？哈哈哈哈……哈哈哈……

HeadFirst： 这有什么好笑的？我是很认真的。

Object： 哇哈哈哈哈……（在地上滚来滚去）……哈哈哈……（泪）……哈哈哈……

HeadFirst： 来人啊！叫救护车，快！

Object： 我没事了……哈……啊……快不行了……好了，真的没事了……啊……（深呼吸）。

HeadFirst： 好吧，请告诉我们封装可以怎样保护你的安全。

Object： 封装会对我的实例变量加上绝对领域，因此没有人能够恶搞我的变量。

HeadFirst： 比如说？

Object： 用膝盖想也知道。大部分的实例变量值都有一个适当的范围，比如身高就不可能是负的、佛跳墙是不可能在3分钟之内做好的。

HeadFirst： 我懂你的意思了。那封装是如何设下保护罩的？

Object： 强迫其他的程序一定得经过setter。如此setter就能够检查参数并判断是否可以执行。setter也许可以退回不合理的值、或是抛出Exception、或者自己进行取小数点的动作。重点在于你可在setter中执行任何动作，直接暴露的public实体变量就没有这个能耐。

HeadFirst： 但是我有看过某些setter什么事情也没做，只是把值设给变量而已。这样不是只会增加执行的负担吗？

Object： 这对getter也是一样的，好处是你事后可以改变想法却不会需要改变其他部分的程序。假设说所有人都使用到你的类以及公有变量，万一有一天你发现这个变量需要检查，那不是所有人都要跟着改成调用setter吗？封装的优点就是能够让你三心二意却又不会伤害别人。直接存取变量的效率是比不上这个好处的。

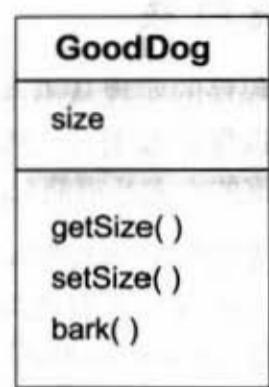
封装 GoodDog

虽然此方法没有加上实质的功能性，但最重要的是允许你能够在事后改变心意：你可以回头把程序改得更安全、更快、更好。

将实例变量设置为 private 的
将 getter 与 setter 设置为 private 的

```
class GoodDog {
    private int size;
    public int getSize() {
        return size;
    }
    public void setSize(int s) {
        size = s;
    }
    void bark() {
        if (size > 60) {
            System.out.println("Woof! Woof!");
        } else if (size > 14) {
            System.out.println("Ruff! Ruff!");
        } else {
            System.out.println("Yip! Yip!");
        }
    }
}

class GoodDogTestDrive {
    public static void main (String[] args) {
        GoodDog one = new GoodDog();
        one.setSize(70);
        GoodDog two = new GoodDog();
        two.setSize(8);
        System.out.println("Dog one: " + one.getSize());
        System.out.println("Dog two: " + two.getSize());
        one.bark();
        two.bark();
    }
}
```



任何有值可以被运用到的地方，都可用调用方法的方式来取得该类型的值。

比如：

int x = 3 + 24;

可以这样改写：

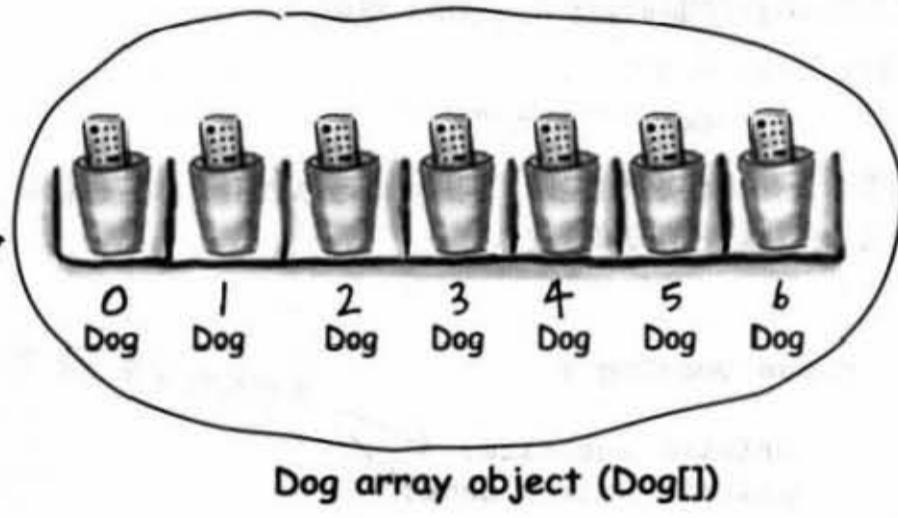
int x = 3 + one.getSize();

数组中对象的行为

数组中的对象就如同其他的对象一样，唯一的差别就是如何取得而已。换言之，不同处就在于你如何取得遥控器。让我们先来尝试调用数组中的Dog对象。

- 1 声明一个装载7个Dog引用的Dog数组。

```
Dog[] pets;
pets = new Dog[7];
```

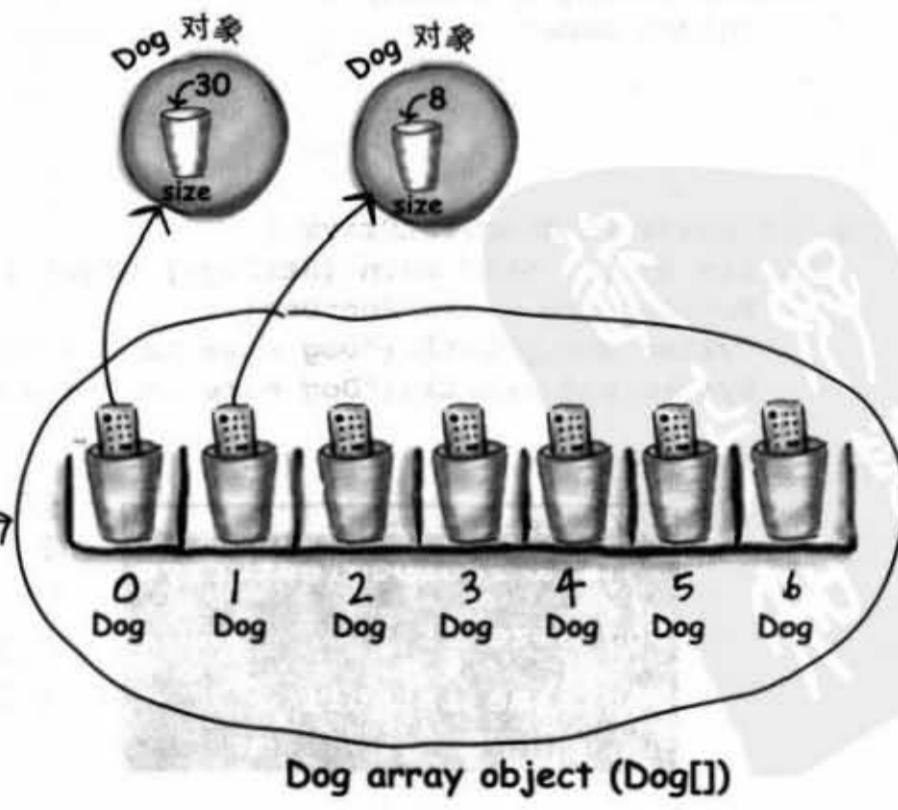


- 2 创建两个Dog对象并赋值为数组的前两项元素。

```
pets[0] = new Dog();
pets[1] = new Dog();
```

- 3 调用这两个Dog对象的方法。

```
pets[0].setSize(30);
int x = pets[0].getSize();
pets[1].setSize(8);
```



声明与初始化实例变量

你已经知道变量的声明至少需要名称与类型：

```
int size;
String name;
```

并且你也知道可以同时初始化（赋值）变量：

```
int size = 420;
String name = "Donny";
```

但如果你没有初始实例变量时，调用getter会发生什么事？也就是说实例变量在初始之前的值是什么？

```
class PoorDog {
    private int size;           ← 声明实例变量但是不给值
    private String name;        ←

    public int getSize() {      ← 会返回什么？
        return size;
    }
    public String getName() {
        return name;
    }
}

public class PoorDogTestDrive {
    public static void main (String[] args) {
        PoorDog one = new PoorDog();
        System.out.println("Dog size is " + one.getSize());
        System.out.println("Dog name is " + one.getName());
    }
}
```

File Edit Window Help CallVet
% java PoorDogTestDrive
Dog size is 0
Dog name is null

实例变量永远都会有默认值。如果你没有明确的赋值给实例变量，或者没有调用setter，实例变量还是会有值！

integers	0
floating points	0.0
booleans	false
references	null

你想这会通过编译吗？
你无需初始实例变量，因为它们会有默认值。数字的primitive（包括char）的预设为0，boolean的预设为false，而对象引用则为null。
(要记得null代表没有操作对象的远程控制，它是个引用而不是对象)

实例变量与局部变量之间的差别

- 1** 实例变量是声明在类内而不是方法中。

```
class Horse {
    private double height = 15.2;
    private String breed;
    // more code...
}
```

- 2** 局部变量是声明在方法中的。

```
class AddThing {
    int a;
    int b = 12;

    public int add() {
        int total = a + b;
        return total;
    }
}
```

- 3** 局部变量在使用前必须初始化。

```
class Foo {
    public void go() {
        int x;
        int z = x + 3;  ←
    }
}
```

无法编译！你可以声明没有值的x，但若要使用时编译器就会给出警示

```
File Edit Window Help Yikes
% javac Foo.java
Foo.java:4: variable x might
not have been initialized
    int z = x + 3;
          ^
1 error
```

局部变量没有默认值！如果在变量被初始前就要使用的话，编译器会显示错误。

*there are no
Dumb Questions*

问： 那方法的参数呢？局部变量的规则也适用于它们身上吗？

答： 方法的参数基本上与局部变量是相同的——它们都是在方法中声明的（精确地说应该是在方法的参数列声明的，但相较于实例变量来说它也算是局部的）。而参数并没有未声明的问题，所以编译器也不可能对这样事情显示出错误。

这是因为如果调用方法而没有赋值参数时编译器就会显示错误。所以说参数一定会被初始化，编译器会确保方法被调用时会有与声明所相符的参数，且参数会自动地被赋值进去。

变量的比较 (primitive主数据类型或引用)

有时你需要知道两个 primitive 主数据类型是否相等。很简单，只要使用 == 这个运算符就可以。有时你想要知道两个引用变量是否引用到堆上的同一个对象。这也很容易，也是使用 == 运算符。但有时你会需要知道两个对象是否真的相等。此时你就得使用 equals() 这个方法。相等的意义要视对象的类型而定。举例来说，如果两个不同的 String 带有相同的字符，它们在意义上是相等的。但对 Dog 来说，你认为尺寸大小或名字一样的 Dog 是相等的吗？所以说是否被视为相等要看对象类型而定。我们会在后面的章节继续探讨对象相等性的部分，但现在我们要知道的是 == 只用来比对两个变量的字节组合，实质所表示的意义则不重要。字节组合要么就是相等，要么就是不相等。

使用 == 来比对 primitive 主数据类型

这个运算式可以用来比较任何类型的两个变量，它只是比较其中的字节组合。

```
int a = 3;
byte b = 3;
if (a == b) { // true }
```

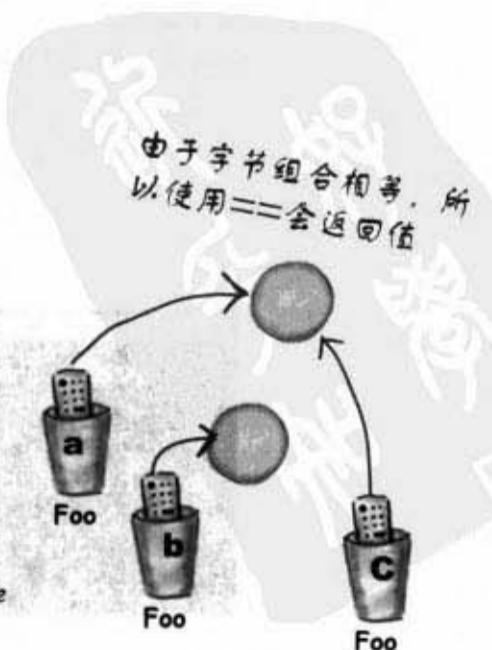
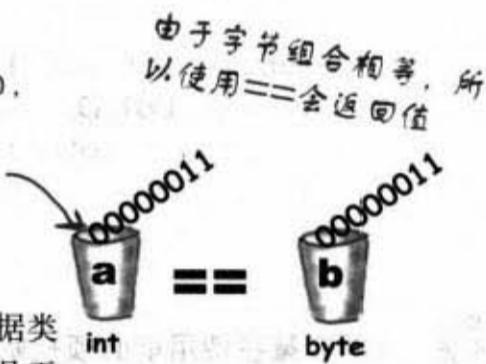
左方有更多的0。
但此处不考虑

使用 == 来判别两个引用是否都指向同一对象。

要记得，这只是在比较字节组合的模样。此规则适用于引用与 primitive 主数据类型。因此 == 运算符对参照相同对象的引用变量会返回值。在此情况下我们还是无法得知字节组合的样式，但可以确定的是所参照的相同的对象。

```
Foo a = new Foo();
Foo b = new Foo();
Foo c = a;
if (a == b) { // false }
if (a == c) { // true }
if (b == c) { // false }
```

a == c is true
a == b is false





Sharpen your pencil

哪些是合法的？

对下面这个方法来说，右边列出的哪几个调用是合法的？

在合法的述句旁边打勾。



```
int calcArea(int height, int width) {
    return height * width;
}
```

```
int a = calcArea(7, 12);
short c = 7;
calcArea(c, 15);
int d = calcArea(57);
calcArea(2, 3);
long t = 42;
int f = calcArea(t, 17);
int g = calcArea();
calcArea();
byte h = calcArea(4, 20);
int j = calcArea(2, 3, 5);
```



练习

我是编译器



这一页的Java程序代码都代表一份完整的源文件。你的任务是要扮演编译器角色并判断哪个程序可以编译过关。如果有问题，哪里要修改？

A

```
class XCopy {
    public static void main(String [] args) {
        int orig = 42;
        XCopy x = new XCopy();
        int y = x.go(orig);
        System.out.println(orig + " " + y);
    }
    int go(int arg) {
        arg = arg * 2;
        return arg;
    }
}
```

B

```
class Clock {
    String time;
    void setTime(String t) {
        time = t;
    }
    void getTime() {
        return time;
    }
}

class ClockTestDrive {
    public static void main(String [] args) {
        Clock c = new Clock();
        c.setTime("1245");
        String tod = c.getTime();
        System.out.println("time: " + tod);
    }
}
```



一组Java组件精心打扮出席化装舞会，中场时间有人提议要玩猜猜我是谁的游戏，你可以根据它们对自己的描述来猜测出提示的是哪位。规则是每个组件都得说实话，若某些提示同时对数个组件都为真的话，则将它们全部填入。

今晚出席舞会的有：

instance variable, argument, return, getter, setter,
encapsulation, public, private, pass by value, method

一个类可以带有很多个

一种方法只能带有一个

可以被隐含地转换

我喜欢private的实例变量

其实就是制作一个拷贝

应该只有setter才能更新

方法可以带很多个

根据定义返回

不应该以实例变量来运用

可以有许多个参数

被定义成采用一个参数

帮忙创建封装

总是单飞



连连看

右边有一个 Java 小程序。其中有两段程序不见了。你的任务是找出下面所列出的程序段与相符的输出。

并非所有的输出都有可对应的程序段，且某些输出可能会被使用多次。画条线将相符的两者连接起来。

程序段

输出

x < 9

14 7

index < 5

9 5

x < 20

19 1

index < 5

14 1

x < 7

25 1

index < 7

7 7

x < 19

20 1

index < 1

20 5

```
public class Mix4 {
    int counter = 0;
    public static void main(String [] args) {
        int count = 0;
        Mix4 [] m4a = new Mix4[20];
        int x = 0;
        while ( [ ] ) {
            m4a[x] = new Mix4();
            m4a[x].counter = m4a[x].counter + 1;
            count = count + 1;
            count = count + m4a[x].maybeNew(x);
            x = x + 1;
        }
        System.out.println(count + " "
                           + m4a[1].counter);
    }
}

public int maybeNew(int index) {
    if ( [ ] ) {
        Mix4 m4 = new Mix4();
        m4.counter = m4.counter + 1;
        return 1;
    }
    return 0;
}
```



泳池迷宫



你的任务是要从游泳池中挑出程序片段并将它填入右边的空格中。同一个片段不能用两次，且泳池中有些多余的片段。填完空格的程序必须要能够编译与执行并产生出下面的输出。

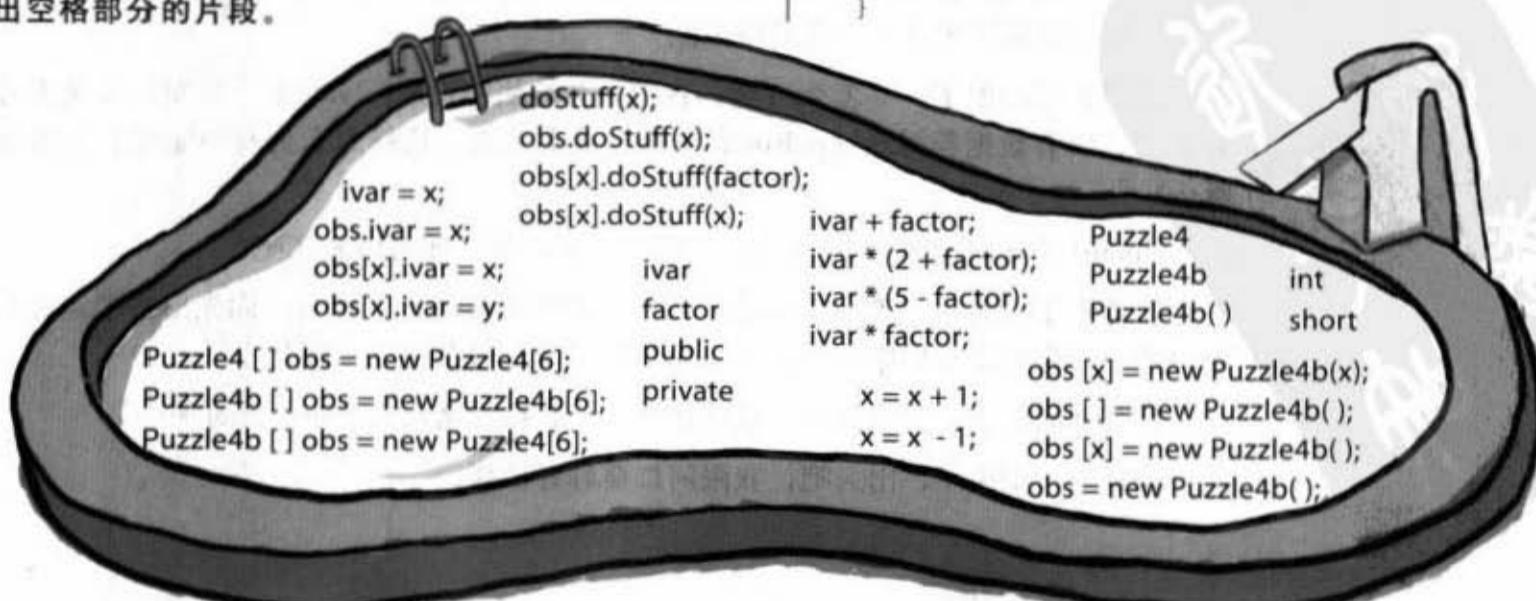
输出

```

File Edit Window Help BellyFlop
%java Puzzle4
result 543345

```

从池中找出可以填在输出空格部分的片段。





达康之道：公私分明

当阿仁发觉有只光电鼠标指在他头上时，整个人都呆住了。他早就知道会面对今天这个状况，但万万想不到的是，拿鼠标的人居然会是傻强，更重要的是这只鼠标居然是无线的。不过这时候也没有办法去想这些事情了。傻强命令阿仁走进琛哥的办公室，琛哥早已经坐在里面等着。阿仁没有把握自己的身份是否已经曝光，这反而让他有点不知道要对琛哥说什么。

“琛哥，”阿仁决定先装傻：“我知道错了，以后我每一行程序都会加批注。”

“五年前，科学园区大门口外的达康公司开张大吉……”琛哥抽了口烟：“我和兄弟们雄心壮志，谁知道开张不到一个月，每天平均被黑客攻击1.3次，一年内挂掉6台服务器。佛祖保佑，算命的说我是‘一将功成万骨枯’，不过我不同意……”没抽两口的烟就被捻熄了。

“出来写程序的，早晚都会有漏洞”琛哥又点起了一根烟，看着傻强：“傻强，你跟我五年多了，你在这几年都很能干，现在有个问题问你，就说如果有兄弟写程序有漏洞，你敢不敢重写？”

傻强不愧是傻的：“当然敢啊，琛哥。”

琛哥：“那你把今天发生的事情说给阿仁听。”

傻强：“漏洞是还没有找到，来攻击我们的黑客倒是抓到了，那个黑客骨头硬，我们把他抓到顶楼足足打了十分钟，十分钟都没有打错一个指令。

琛哥说那个写程序的人很会掩饰，今天谁没有出现，谁就是写出漏洞的人。

仁哥，我好想问你，今天追的show girl漂亮吗？因为你知道，show girl不漂亮那就没劲了。你知道，如果有个人他coding不专心又会翘班去看信息展的话，他就会写出漏洞。是你吧，仁哥？”

阿仁发现到有机会转移目标：“对不起，我是系统分析师……我看过你写的系统登录类，我觉得那里才最有可能出现漏洞……”

傻强开始心虚了：“怎么可能，我把所有方法都设成private了，外面的人是不可能存取的，所有数据都得通过public的实例变量来更新，这样怎么会有问题呢？应该不会呀……”

Bingo！阿仁差点要大叫出来：“琛哥，你看呢，我可以走了吧？”

琛哥点了点头：“等一下再走，我有事情要跟你讨论。傻强，你先出去吧，还有，这一阵子你先不要写程序……，不，你今天起就跟着泰国佬照顾仓库好了”

傻强两眼发直：“琛哥……我没错呀……我跟你这么久了……琛哥……”

琛哥：“别说了，出去吧，我跟阿仁要好好谈谈。”

阿仁的身份会曝光
吗？傻强到底说出了
什么不该说的事？





练习解答

A “XCopy” 编译与运行都没有问题！输出结果是“42 84”。
要记得Java是按值传递（也就是传拷贝的），所以orig这个变量的值
不会被go()方法改变。

```

class Clock {
    String time;
    void setTime(String t) {
        time = t;
    }
    String getTime() {
        return time;
    }
}

class ClockTestDrive {
    public static void main(String [] args) {
        Clock c = new Clock();
        c.setTime("1245");
        String tod = c.getTime();
        System.out.println("time: " + tod);
    }
}

```

注意：setter 要定义出返回的类型

一个类可以带有很多个	instance variables, getter, setter, method
一种方法只能带有一个	return
可以被隐含地转换	return, argument
我喜欢private的实例变量	encapsulation
其实就是制作一个拷贝	pass by value
应该只有setter才能更新	instance variables
方不可以带很多个	argument
根据定义返回	getter
不应该以实例变量来运用	public
可以有许多个参数	method
被定义成采用一个参数	setter
帮忙创建封装	getter, setter, public, private
总是单飞	return

迷宫解答

```

public class Puzzle4 {
    public static void main(String [] args) {
        Puzzle4b [ ] obs = new Puzzle4b[6];
        int y = 1;
        int x = 0;
        int result = 0;
        while (x < 6) {
            obs[x] = new Puzzle4b( );
            obs[x].ivar = y;
            y = y * 10;
            x = x + 1;
        }
        x = 6;
        while (x > 0) {
            x = x - 1;
            result = result + obs[x].doStuff(x);
        }
        System.out.println("result " + result);
    }
}

class Puzzle4b {
    int ivar;
    public int doStuff(int factor) {
        if (ivar > 100) {
            return ivar * factor;
        } else {
            return ivar * (5 - factor);
        }
    }
}

```

输出

```

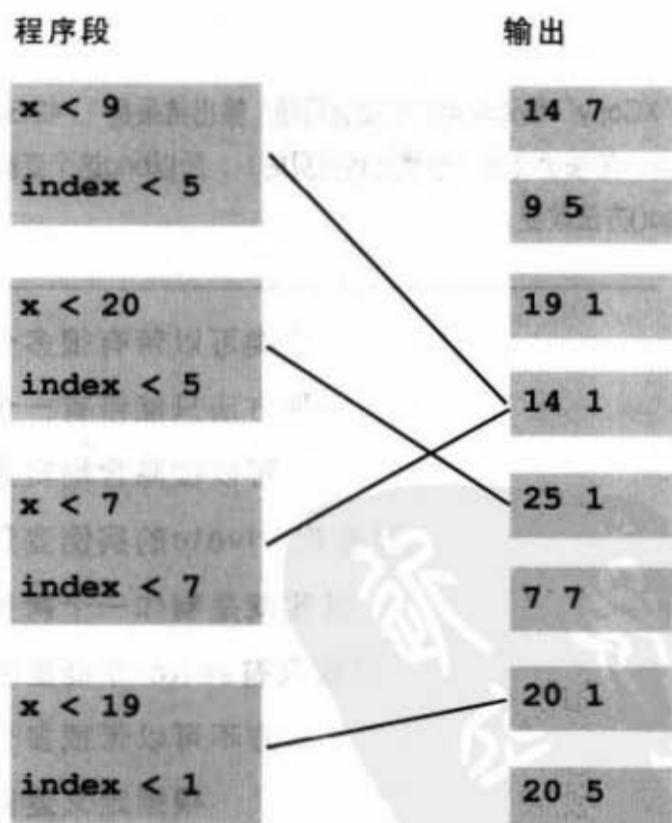
File Edit Window Help BellyFlop
%java Puzzle4
result 543345

```

“达康之道”解析

傻强犯了一个严重的错误：千万别让人知道你的概念是错的。

实例变量应该要标记为private，并通过getter与setter来存取。如此才能有机会确保实例变量值会落在合法的范围内。



5 编写程序

超强力方法



让方法产生更大的作用。我们看过变量、操作过一些对象，并编写了一点程序，但这还不够，我们需要更多的其他操作，例如运算符等。更多的运算符才能让我们执行出比吠两声更有趣的事情。还有，我们需要的不只是while循环，for循环也是专业的配备之一。产生随机数也会很有用。将String转换成int会很酷，最好也要学起来。我们如果能真地从无到有去编写与测试一个实用的程序那会更棒。或许写个game是个好主意。那可是个不小的工程，因此得要花上两章才能完成。这一章会先创建出简单版，然后第6章再来创建出个豪华版。

创建一个类似战舰的游戏： 攻击网站

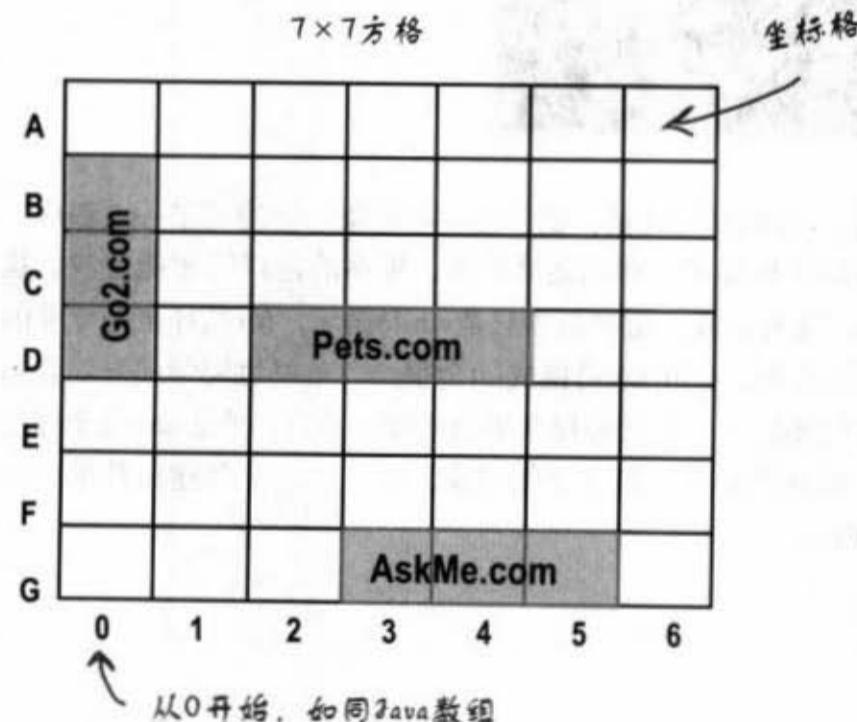
有一种棋盘类的战舰游戏，目标是要猜测对方战舰的坐标，然后轮流开炮攻击，命中数发就可以打沉对方的战舰。

不过我们不喜欢战争，只要打垮这些达康公司就好（因为与商业行为有关，如此一来本书就可以归类在经营企管的费用上）。

游戏目标：以最少的猜测次数打掉计算机所安排的达康公司（Dot Com）网站。计算机会根据你的表现来评分。

初始设置：程序启动后，计算机会在虚拟的 7×7 方格上安排3个达康网站。安排完成后，游戏会要求你开始猜坐标。

进行游戏：因为我们还没有学到图形接口的程序设计，所以这一版会在命令栏上进行。计算机会提示你输入所猜测的位置（格子），你会输入“A3”或“C5”等。计算机会反馈给你命中“Hit”没中“Miss”或击沉“Sunk”等回应。当你清光所有的达康时，游戏会列出你的分数并结束。



你会创建一个攻击网站游戏，它有 7×7 的格子与3间达康公司。每个达康网站占用3个格子。让我们重现网络大崩盘噩梦吧！

游戏进行中的画面

```
File Edit Window Help Sell
%java DotComBust
Enter a guess A3
miss
Enter a guess B2
miss
Enter a guess C4
miss
Enter a guess D2
hit
Enter a guess D3
hit
Enter a guess D4
Ouch! You sunk Pets.com :(
kill
Enter a guess B4
miss
Enter a guess G3
hit
Enter a guess G4
hit
Enter a guess G5
Ouch! You sunk AskMe.com :(
kill
Enter a guess A7
miss
Enter a guess B7
miss
Enter a guess C7
miss
Enter a guess D7
miss
Enter a guess E7
miss
Enter a guess F7
miss
Enter a guess G7
```

首先进行高层设计

我们需要类和方法，但是要哪些类和方法呢？要回答这个问题，得先要取得更多的游戏信息。

首先，我们必须了解游戏的流程。以下是基本思路：

1 玩家启动游戏。

- A** 计算机创建3个达康网站。
- B** 将此3个达康网站停在虚拟战场上。

2 游戏开始。

重复下面的操作直到所有达康网站被歼灭为止。

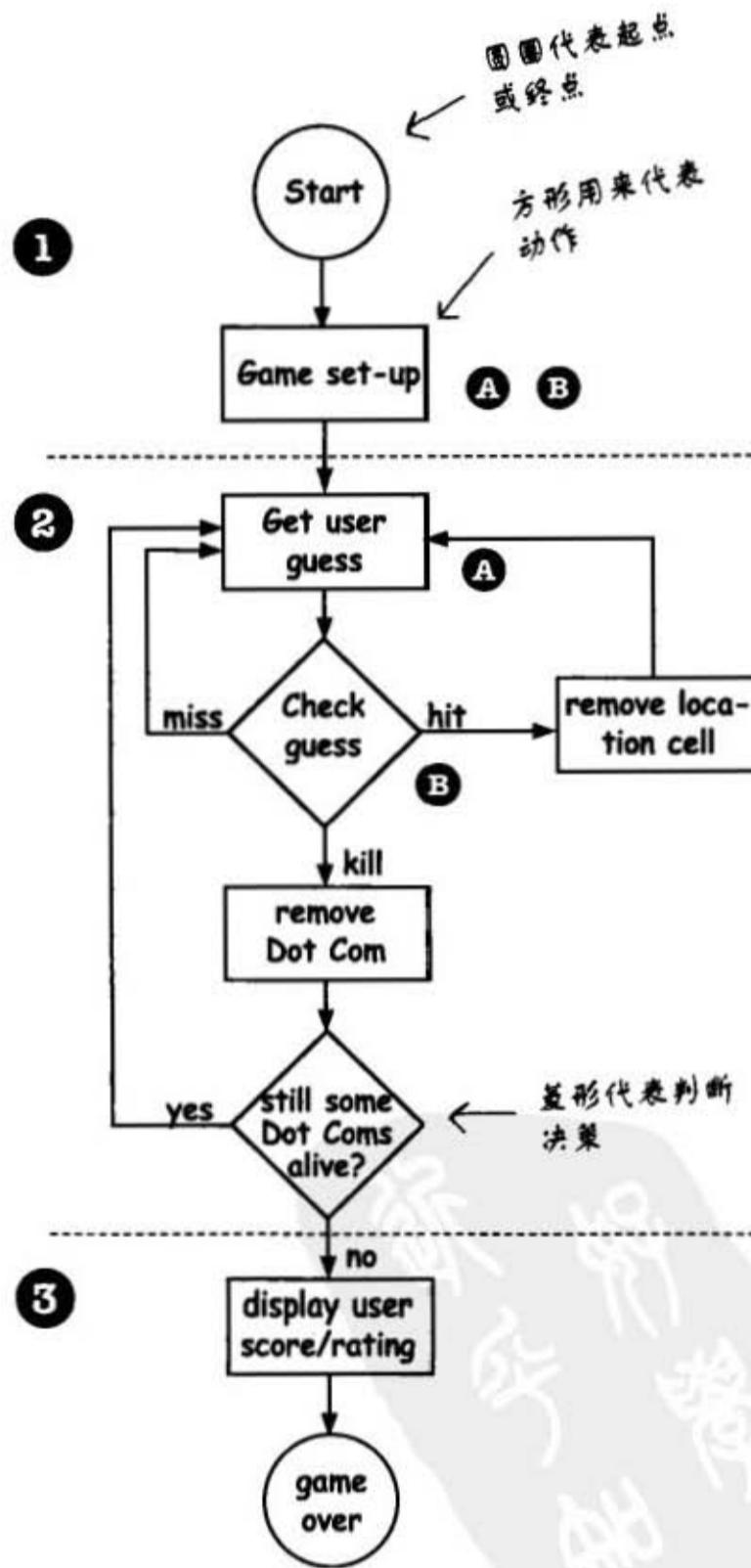
- A** 提示玩家输入坐标。

- B** 检查是否命中、没中或击沉。如果命中就删除格子，击沉就删除达康网站。

3 游戏结束。

根据猜测次数给分。

现在我们对游戏的流程有了了解。接下来的步骤是要设想出需要哪些对象。要记得以面向对象的方式来思考，专注于程序中出现的事物而不是过程。



哇！真正的流程图。

攻击网站游戏

简单的开始

看起来我们至少需要两个类：Game类和DotCom类。但在我们着手开发完整功能版之前，先从一个简单版本开始。简单版称为“Simple Dot Com Game”。这一章会创建出此版本，下一章会进行豪华版的开发工作。

该版本的东西都比较简单。相对于二维方阵，我们只使用横列，并且只设定一家达康公司。

然而游戏的目标仍然相同，因此游戏还是会需要做出DotCom的实例，将它指派在横列上，取得玩家的输入，并在所有的DotCom格子被命中时结束游戏。这个简单版能够作为豪华版的踏脚石。

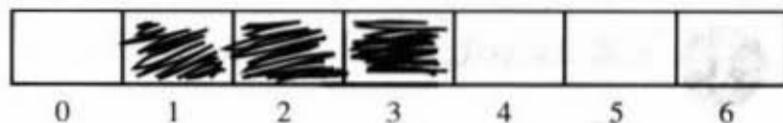
在这个简单版中，Game这个类没有实例变量，且所有的程序代码都在main()中。也就是说程序启动执行main()只会做出一个DotCom的实例，挑出一个位置来放3个连续的格子，要求玩家猜测，检查是否命中，重复这些步骤直到3格都被命中为止。

要知道虚拟的横列是虚拟的。换言之，它并没有出现在程序中，只要玩家与计算机都知道有3个连续的格子会出现在7格的横列中就好，横列并不一定要表现在程序代码中。你也许会想要用有7个int的数组来代表横列，并用其中3个元素代表达康出现的位置，但其实不需这么做。我们只需要3个元素的数组来代表DotCom占据的位置。

1

游戏启动。创建单一的DotCom并指定3个格子在共7格的横列中的位置。

相对于使用“A2”这种表示法，现在的位置只需要数字就可以：

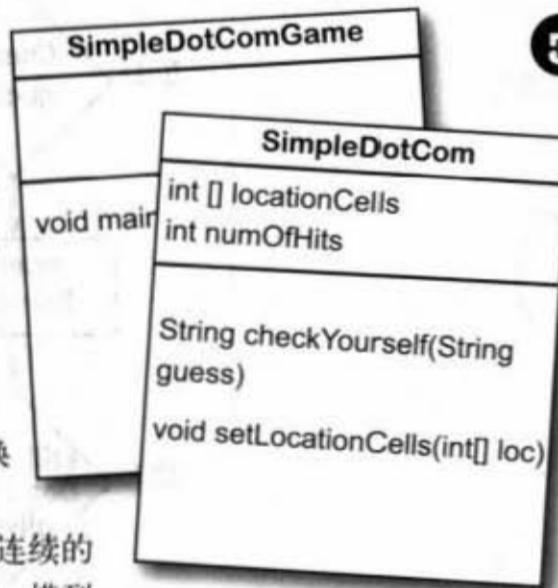


2

开始游戏。提示玩家猜测，然后检查是否命中DotCom的格子。如果是，则递增numOfHits变量的值。

3

游戏结束。3格都命中时游戏结束（当numOfHits的值递增到3），并告诉玩家他们花了多少次才干掉这个DotCom。



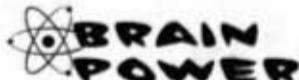
游戏画面

```
File Edit Window Help Destroy
% java SimpleDotComGame
enter a number 2
hit
enter a number 3
hit
enter a number 4
miss
enter a number 1
kill
You took 4 guesses
```

开发类

身为一个程序员，你或许会有编写程序的方法论（methodology）/过程（process）/步骤（approach）。嗯，我们也有。我们的顺序设计是先通过编写程序来帮助你了解和学习我们的想法，并不需要遵循我们实际上怎样写程序的方式。当然啦，实际工作时，你会遵循个人、项目或客户的规范。但我们完全是依照我们自己的想法去做的。当我们在创建Java的类以当作“学习经验”时，程序会像下面这样：

- 找出类应该做的事情。
- 列出实例变量和方法。
- 编写方法的伪码（稍后说明）。
- 编写方法的测试用程序。
- 实现类。
- 测试方法。
- 除错或重新设计。
- 邀请辣妹参加庆功派对（没有成功过）



开始编写程序时，你要决定哪个类先创建出来呢？假设某些类需要同时运用到多个类（如果你遵循良好的面向对象原则，并且没有让单一的类执行太多的任务），又该从哪里开始呢？

我们会为每个类写出下列的事物：

伪码 **测试码** **真实码**

这些标记会用在接下来的几页以显示出我们正在讨论哪个部分。举例来说，下面的标记表示我们正在处理SimpleDotCom类的伪码。

SimpleDotCom 类
伪码 测试码 真实码

伪码

伪码能帮你专注于逻辑而不需要顾虑到程序语法。

测试码

测试用的程序代码。

真实码

实际设计出的真正Java程序代码。



伪码 测试码 真实码

```

SimpleDotCom
int[] locationCells
int numOfHits

String checkYourself(String guess)
void setLocationCells(int[] loc)

```

看过下面的范例你就会对伪码如何运行有个基本的了解。伪码是介于真正的Java程序与正常英语之间的一种语言。伪码大致上包括3部分：实例变量的声明、方法的声明和方法的逻辑。伪码最重要的部分是方法的逻辑，因为它定义出会发生“什么事”，这个部分会在稍后真正编写程序代码时转译成“如何”发生。

DECLARE an int array to hold the location cells. Call it locationCells.

DECLARE an int to hold the number of hits. Call it numOfHits and SET it to 0.

DECLARE a checkYourself() method that takes a String for the user's guess ("1", "3", etc.), checks it, and returns a result representing a "hit", "miss", or "kill".

DECLARE a setLocationCells() setter method that takes an int array (which has the three cell locations as ints (2,3,4, etc.).

METHOD: String checkYourself(String userGuess)

GET the user guess as a String parameter

CONVERT the user guess to an int

REPEAT with each of the location cells in the int array

// **COMPARE** the user guess to the location cell

IF the user guess matches

INCREMENT the number of hits

// **FIND OUT** if it was the last location cell:

IF number of hits is 3, **RETURN** "kill" as the result

ELSE it was not a kill, so **RETURN** "hit"

END IF

ELSE the user guess did not match, so **RETURN** "miss"

END IF

END REPEAT

END METHOD

METHOD: void setLocationCells(int[] cellLocations)

GET the cell locations as an int array parameter

ASSIGN the cell locations parameter to the cell locations instance variable

END METHOD

伪码

测试码

真实码

编写方法的实现部分

开始编写真正可用的方法程序代码

在我们开始编写方法之前，让我们先退回一步来写出测试方法用的程序代码。没错，我们会在有东西可以测试前就先写出测试用的部分！

先编写测试用程序代码的概念来自于极限编程(XP)方法论，这样做会让你能够更容易与更快地写出程序代码。我们并不强制采用极限编程XP，但觉得这个概念真的很不错，并且极限编程XP听起来也很酷。



极限编程(XP)

极限编程(XP)是一种新型的软件开发方法论。它的构想是结合了许多种“程序员真想这么做”的方法而成的。XP的概念于20世纪90年代出现，并已经被从两人工作室到福特汽车等级的大企业所采用。XP的推进力来自于客户会得到他想要的、想要的时候就能够取得甚至在开发过程后期变更规格时也是如此。

XP是由一组被证明有效的施行方法所组成的，这些方法都是被设计来共同运作，但许多人只选择性地实行部分的XP规则。这些方法包括了：

- (1) 多次经常性的规模发布。
- (2) 避免加入规格没有的功能（不管“未来”会用到的功能性有多诱人）。

- (3) 先写测试用的程序。
- (4) 正常工作上下班。
- (5) 随时随地重构(refactor)，也就是改善程序代码。
- (6) 保持简单。
- (7) 双双结伴进行工作，并经常交换伴侣（不是说那个）以便让大家都清楚全局。

建议阅读专门的书籍，以免一知半解地胡乱应用。

伪码 测试码 真实码

为 SimpleDotCom 编写测试码

我们需要写出能够创建 SimpleDotCom 对象并加以测试的程序代码。对 SimpleDotCom 这个类来说，我们真正在意的只有 checkYourself() 方法，然而我们还要实现出 setLocationCells() 方法以便让 checkYourself() 方法能够正确地执行。

先来看下面 checkYourself() 方法这个方法的伪码（setLocationCells() 是个用手肘想也知道的 setter，所以我们不用花太多时间去关心，但真正的应用程序会需要更稳固的 setter，此时就会需要加以测试）。

然后自问：如果 checkYourself() 方法已经写好的话，我要用什么样的测试码才能证明这个方法能够正确地运行？

下面是伪码：

```

METHOD String checkYourself(String userGuess)
GET the user guess as a String parameter
CONVERT the user guess to an int
REPEAT with each of the location cells in the int array
    // COMPARE the user guess to the location cell
    IF the user guess matches
        INCREMENT the number of hits
        // FIND OUT if it was the last location cell:
        IF number of hits is 3, RETURN "Kill" as the result
        ELSE it was not a kill, so RETURN"Hit"
    END IF
    ELSE the user guess did not match, so RETURN "Miss"
    END IF
END REPEAT
END METHOD

```

应该要测试的部分：

- (1) SimpleDotCom 对象的初始化。
- (2) 赋值位置（带有 3 个 int 的数组，像是 {2, 3, 4}）。
- (3) 创建代表玩家猜设的字符串（“2”或“0”等）。
- (4) 传入伪造的玩家猜测来叫用 checkYourself() 方法。
- (5) 列出结果以观察是否正确。

伪码

测试码

真实码

there are no
Dumb Questions

问： 可能是我自己没搞清楚，但是你要如何测试还没有出现的东西呢？

答： 我们从未说过要开始进行测试。编写测试用程序代码的同时确实没有东西可测，因此在写出“stub”程序代码前你应该无法编译。

问： 那我还是不懂，为什么不先写好程序然后再制作测试用的代码？

答： 思索与编写测试用的程序代码能够帮助你了解被测的程序应该要做哪些事情。

当你的实作程序代码完成时，你就能够有准备地地测试代码来进行验证。此外，你心里也有数，若没有先把它做出来，那么以后也不会去做。

理想上，先写出一点点的测试码，然后只编写能够通过该测试的方法就好。之后再编写另一点测试码，再编写新的实现以让测试能够通过。经过如此的循环，你就能够证明新加入的程序代码不会破坏原有已经测试过的部分。

SimpleDotCom 的测试码

```
public class SimpleDotComTestDrive {
    public static void main (String[] args) {
        SimpleDotCom dot = new SimpleDotCom(); // 初始化一个 SimpleDotCom 对象
        int[] locations = {2,3,4}; // 创建带有 dot.com 位置的数组
        dot.setLocationCells(locations); // 调用 dot.com 的 setter
        String userGuess = "2"; // 假的猜测
        String result = dot.checkYourself(userGuess);
        String testResult = "failed"; // 调用被测方法并传入假的数据
        if (result.equals("hit")) {
            testResult = "passed";
        }
        System.out.println(testResult);
    }
}
```

Sharpen your pencil

接下来会继续实现 SimpleDotCom 这个类，当然也会回头修改这个测试用的类。你认为我们还需要加入什么？我们还缺了哪些测试？请写下你的看法。

伪码

测试码

真实码

checkYourself()方法

从伪码到真正的Java程序代码之间并没有完美的对应，你会看到一些调整。伪码让我们对于程序代码需要做什么有比较好的概念，接下来我们就可以看出来要如何做。

观察此程序代码的同时，要想想看如何将它改善。有标记①符号的部分是我们还没有讨论过的语法与功能。后面的章节内容会加以说明。

取得玩家的猜测 **public String checkYourself(String stringGuess) {**

把用户猜测转换成int

 ① int guess = Integer.parseInt(stringGuess); ← 把字符串转换成int

对每个格子重复

 ② for (int cell : locationCells) { ← 以循环对每个格子重复执行

如果猜中

 if (guess == cell) { ← 比较格子与猜测值

递增命中数

 ③ numOfHits++; ← 命中！

 ④ break; ← 已经离开循环，但需

 要判断是否击沉

 } // end if

} // end for

如果命中数为3

 if (numOfHits == locationCells.length) {

返回击沉信息

 result = "kill";

 ← 已经离开循环，但需要判断是否击沉

} // end if

列出信息

 System.out.println(result); ← 将结果显示出来

 return result;

 ← 将结果返回给调用方

} // end method

伪码

测试码

真实码

新功能

这一页说明之前没有讨论过的功能。
其他的细节在后面还会继续探讨。这
足够让你继续下去。

① 将string转换成 int

Java内建类
↓
Integer.parseInt("3")

Integer的一个方法，能够将String解析
采用String参数

② for循环

for (int cell : locationCells) { }

声明出带有数组元素的变量。在循环
的每次循环中，此变量的值都会带有
不同的数组元素，直到跳出循环为
止。

可以把(.)符号读作
in，也就是for each int
in location Cells...

③ 后递增 (post-incre-
ment) 操作符

numOfHits++

++ 代表将其数值加1

这与
numOfHits = numOfHits + 1 的意
思相同

④ 中止指令

break;

无条件立即跳出循环

伪码 测试码 真实码

there are no
Dumb Questions

问：如果传给parseInt()的不是数字会怎样？

答：这个方法只会对代表数字的String产生作用。除此之外，程序就会崩溃（崩溃的意思是抛出一个异常，后面会有讨论异常的章节）。

问：本书前面开头的地方有个for循环的范例与此处的范例大大的不同——是否有两种不同风格的for循环？

答：是的！Java一开始有下面这种for循环：

```
for(int i=0; i < 10; i++) {
    //do something
}
```

但从Java 5.0开始你可以对数组（或其他集合）使用加强版for循环。当然你还是可以对数组使用原始风格的for循环，只是使用加强版会比较好写。

SimpleDotCom与SimpleDotComTester的最终版本

```
public class SimpleDotComTestDrive {
    public static void main (String[] args) {
        SimpleDotCom dot = new SimpleDotCom();
        int[] locations = {2,3,4};
        dot.setLocationCells(locations);
        String userGuess = "2";
        String result = dot.checkYourself(userGuess);
    }
}
```

```
public class SimpleDotCom {
    int[] locationCells;
    int numOfHits = 0;

    public void setLocationCells(int[] locs) {
        locationCells = locs;
    }

    public String checkYourself(String stringGuess) {
        int guess = Integer.parseInt(stringGuess);
        String result = "miss";
        for (int cell : locationCells) {
            if (guess == cell) {
                result = "hit";
                numOfHits++;
                break;
            }
        } // out of the loop
        if (numOfHits == locationCells.length) {
            result = "kill";
        }
        System.out.println(result);
        return result;
    } // close method
} // close class
```

虽然能够编译与运行，但程序其实有些问题，我们稍后会讨论。

执行此程序会有什么结果？

测试用程序代码会创建SimpleDotCom对象并将位置指定为2,3,4。然后它会以2来伪造猜测传给checkYourself()方法。

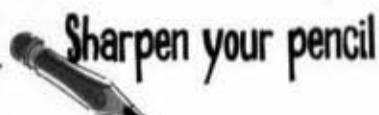
若正确执行会有下面这样的输出：

```
java SimpleDotComTestDrive
hit
```

伪码

测试码

真实码



我们创建出SimpleDoCom类和测试类。但我们确实还没有把游戏作出来。上一页的程序代码以游戏规格为准，写出你的游戏伪码。我们列出了几行当作提示。实际的游戏程序代码在下一页，因此你要作答后才能翻到下一页。
答案应该介于12~18行之间。

METHOD public static void main (String [] args)

DECLARE an int variable to hold the number of user guesses, named numOfGuesses

COMPUTE a random number between 0 and 4 that will be the starting location cell position

WHILE the dot com is still alive :

GET user input from the command line

SimpleDotComGame
需要有下面的功能：

1. 创建出SimpleDotCom对象。
2. 赋值给它。
3. 要求玩家猜测。
4. 检查猜测值。
5. 重复猜测直到击沉为止。
6. 显示玩家的猜测次数。

游戏进行画面：

```
File Edit Window Help Runaway
%java SimpleDotComGame
enter a number 2
hit
enter a number 3
hit
enter a number 4
miss
enter a number 1
kill
You took 4 guesses
```

伪码 测试码 真实码

SimpleDotComGame类的伪码

所有事情都发生在main()中

有些事情必须要依靠信仰。举例来说，有一行伪码说要“从命令行取得输入”。其实我们不会想要自己从无到有地完成这项功能。幸好我们使用的是面向对象。这代表你可以要求其他已经写好的类或对象来完成这件事而不必考虑它是怎么达成的。当你在编写伪码时，应该要假设你总有办法可以做出某些功能，如此才能让你专注于逻辑设计。

```
public static void main (String [] args)
    DECLARE an int variable to hold the number of user guesses, named numOfGuesses, set it to 0.
    MAKE a new SimpleDotCom instance
    COMPUTE a random number between 0 and 4 that will be the starting location cell position
    MAKE an int array with 3 ints using the randomly-generated number, that number incremented by 1,
    and that number incremented by 2 (example: 3,4,5)
    INVOKE the setLocationCells() method on the SimpleDotCom instance
    DECLARE a boolean variable representing the state of the game, named isAlive. SET it to true

    WHILE the dot com is still alive (isAlive == true) :
        GET user input from the command line
        // 检查用户的猜测
        INVOKE the checkYourself() method on the SimpleDotCom instance
        INCREMENT numOfGuesses variable
        // 判断是否击沉
        IF result is "kill"
            SET isAlive to false (which means we won't enter the loop again)
            PRINT the number of user guesses
        END IF
    END WHILE
END METHOD
```

学习技巧



每次使用单边大脑的时间不要太久。连续使用左边大脑30分钟就如同连续使用左臂30分钟一样。周期性地交换以让大脑两侧能够轮流休息。左脑活动包括了循序渐进的工作、解决逻辑问题与分析，而右脑活动包括了隐喻、创造性思考、模式匹配与可视化。

要点

- 你的Java程序应该从高层的设计开始。
- 你通常会在创建新的类时写出下列3种东西：
 - 伪码
 - 测试码
 - 真实码
- 伪码应该描述要做什么事情而不是如何做。
- 使用伪码来帮助测试码的设计。
- 实现方法之前应该要编写测试码。
- 如果知道要执行多少次，应该要使用for循环而不是while循环。
- 使用前置或后置的递增为变量加1（比如`x++`）。
- 使用前置或后置的递减来对变量减1（比如`x--`）。
- 使用`Integer.parseInt()`来取得String的整数值。
- `Integer.parseInt()`只会在所给的String为数字时有作用。
- 使用`break`命令来提前跳出循环。



伪码 测试码 真实码

游戏的main()方法

如同你对SimpleDotCom这个类所做的一样，也要想到如何能够改进这部分的程序代码。有标记数字①的地方是我们会加以讨论的部分。下一页会加以说明。你可能会怀疑我们为何跳过这个类的测试程序，事实上我们不需要此游戏的测试程序。它只有一个方法，所以还有需要另外作一个类来调用这个main()吗？

```

public static void main(String[] args) {      记录玩家猜测次数
    int numOfGuesses = 0;                      ← 的变量
    GameHelper helper = new GameHelper();        ← 我们会写出这个类来取得玩家的输入，现在先假装这是Java提供的

    SimpleDotCom theDotCom = new SimpleDotCom(); ← 创建dot com对象
    int randomNum = (int) (Math.random() * 5);   ← ① 用随机数产生第一格的位置，然后以此制作出数组
    int[] locations = {randomNum, randomNum+1, randomNum+2};

    theDotCom.setLocationCells(locations);       ← 赋值位置

    boolean isAlive = true;                     ← 创建出记录游戏是否继续进行的boolean
                                                变量，这会用在while循环中
    while(isAlive == true) {                   ← ② 取得玩家输入的字符串
        String guess = helper.getUserInput("enter a number"); ← 取得玩家输入的字符串

        String result = theDotCom.checkYourself(guess);           ← 检查玩家的猜测并将结果
        numOfGuesses++;                                         ← 存储在String中
                                                               increment guess count

        if (result.equals("kill")) {                            ← 是否击沉？若击沉，则设定isAlive为
            isAlive = false;                                    ← false并印出猜测次数
            System.out.println("You took " + numOfGuesses + " guesses");
        } // close if
    } // close while
} // close main

```

伪码

测试码

真实码

random()与getUserInput()

有两个部分需要加以说明。这只是个帮助你继续进行下去的解释，在本章的后面有更多关于GameHelper的详细讨论。

① 产生随机数

```
int randomNum = (int) (Math.random() * 5)
```

↑
声明一个保存随机数
的变量

↑
Java内建的类

↑
Math这个类内含的一
个方法

这是辅助性类的一个实例。
它来自于我们还没有讨论到
的GameHelper

这个方法会以String参数
当作提示来要求玩家输入
一个猜测的数字

② 取得玩家输入

```
String guess = helper.getUserInput("enter a number");
```

↑
我们声明出这个变量来保存
玩家输入的猜测值

↑
GameHelper的一个方法，它会
在印出提示字符串后等待玩家
输入

伪码 测试码 真实码

最后一个类：GameHelper

dot com的类已经写好了。

game的类也是。

还剩下helper的类——带有getUserInput()方法的这个。此程序代码会从命令行取得输入。其中有许多细节已经超出这一章所要讨论的范围，因此我们会把这些部分留在后面（第14章）再加以说明。

只要将下列的程序代码拷贝*出来编译成名称为GameHelper的类就可以。将SimpleDotCom、SimpleDotComGame和GameHelper这3个类放到同一个目录中。



现成码

只要看到这个标记 就代表把这个程序代码拿来用就对了。你可以认为它是没有问题的。我们稍后会讨论到它是如何运行的。



我已经预先
“烤”好一些程序
代码，所以你不需要自己
动手。



现成码

```
import java.io.*;
public class GameHelper {
    public String getUserInput(String prompt) {
        String inputLine = null;
        System.out.print(prompt + " ");
        try {
            BufferedReader is = new BufferedReader(
                new InputStreamReader(System.in));
            inputLine = is.readLine();
            if (inputLine.length() == 0) return null;
        } catch (IOException e) {
            System.out.println("IOException: " + e);
        }
        return inputLine;
    }
}
```

*我们知道你很喜欢打字，但是如果你没时间的话，可以到wickedlysmart.com来下载已经打好的Ready-bake 程序代码。

玩游戏

下面就是执行游戏，输入
1, 2, 3, 4, 5, 6的结果，
看起来还不错！

完整的程序交互

(你的画面可能会不同)

```
File Edit Window Help Smile
% java SimpleDotComGame
enter a number 1
miss
enter a number 2
miss
enter a number 3
miss
enter a number 4
hit
enter a number 5
hit
enter a number 6
kill
You took 6 guesses
```

这是什么？bug吗？

下面就是运行游戏，输入1,
1, 1 的结果。

```
File Edit Window Help Faint
% java SimpleDotComGame
enter a number 1
hit
enter a number 1
hit
enter a number 1
kill
You took 3 guesses
```

Sharpen your pencil



危机！

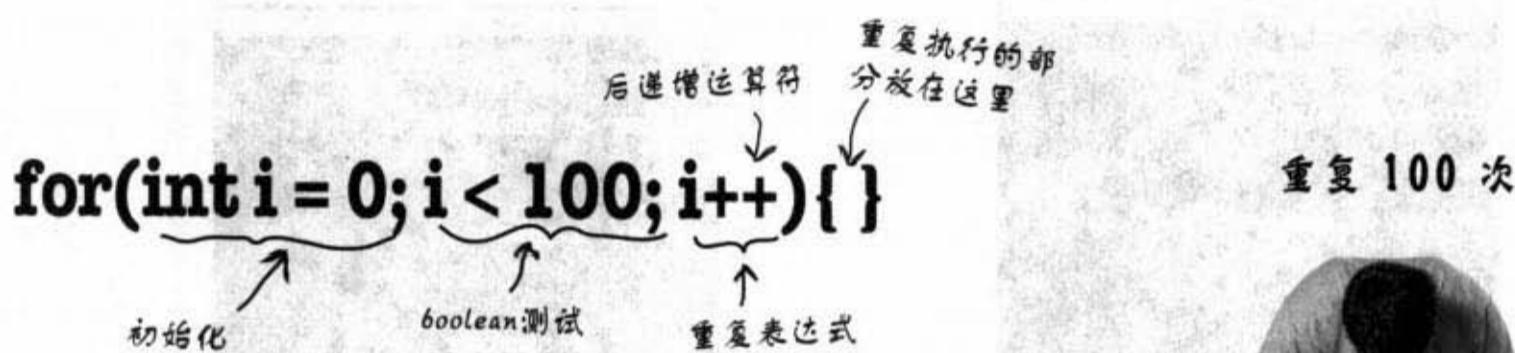
我们能找出bug吗?
我们能修改bug吗?

我们会在下一章回答这些问题 以及更多的信息……你能看出哪里有问题并加以修复吗？

关于for循环

我们已经讨论过本章的游戏程序代码（下一章会继续开发豪华版）。之前还不打算用细节和背景信息等杂事来扰乱你，所以这些东西现在才来讨论。我们会从for循环开始，但如果你是C++程序设计高手，那么可以跳过这几页。

基本（非加强版）的for循环



上面这行程序以中文来说表示：“重复100次”

而编译器会这么认为：

- (1) 创建变量*i*并赋值为0。
- (2) 只要*i*小于100就重复执行。
- (3) 在每趟重复过程最后把*i*加1。

第一段：初始化。

使用这个部分来声明和初始化用在循环体内的变量。你通常会将此变量作为计数器。实际上你可以在这里初始一个以上的变量，但我们稍后才会讨论。

第二段：boolean测试。

测试条件摆在这里。不管写了什么，这里一定要算出一个boolean值（`true`或`false`）。你可以安置`x>-4`这种测试，或者可以调用会返回boolean值的方法。

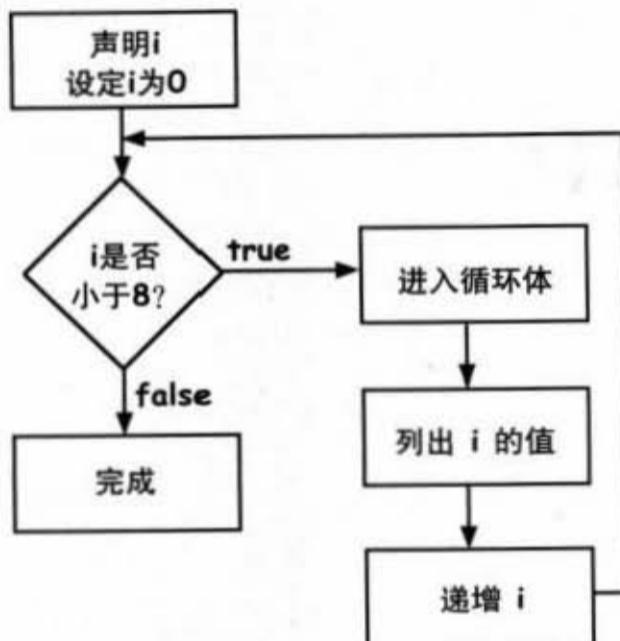
第三段：重复表达式。

在这里安置每趟循环运行完后要执行的项目。要记得这会在运行完一圈之后才会执行。



循环之旅

```
for (int i = 0; i < 8; i++) {
    System.out.println(i);
}
System.out.println("done");
```



比较for循环与while循环的差别

while循环只有boolean测试：它并没有内建的初始化或重复表达式。while适合用在不知道要循环几次的循环上。若你知道要执行几次，则使用for会比较容易阅读。下面是使用while的相同循环：

```
int i = 0; ← 必须得声明并
               初始化计数器
while (i < 8) { 变量
    System.out.println(i);
    i++; ← 将计数器递增
}
System.out.println("done");
```

输出：

```
File Edit Window Help Repeat
%java Test
0
1
2
3
4
5
6
7
done
```

++ --

前置与后置的递增/递减操作符

这是加1或减1的快捷方式。

x++;

这一行与下面有相同的功能：

x = x + 1;

它们都代表：

“将x的目前值加一后存放在x中”

x--;

这一行与下面有相同的功能：

x = x - 1;

加入这种操作符会影响结果。放在变量前面代表先执行加减操作然后再来运用变量的值。前置时只有下面这样的运算才有意义：

int x = 0; int z = ++x;

结果两个变量都是1。但是下面这一行会有不同的结果：

int x = 0; int z = x++;

执行完x是1而z是0。z会被先指派x的值，然后才会执行递增x的操作。

加强版的for

加强版的for循环

从Java 5.0 (Tiger) 开始, Java语言就有称为加强版的for循环, 它能够很容易地逐个运行数组或其他集合 (collection) 的元素。(下一章会讨论其他类型的集合)。这是个很好的强化功能, 因为这是for循环很常见的用途。我们会在讨论非数组的集合时再次看到加强版的for循环。

声明会带有数组单一
元素的循环变量
↓
for (String name : nameArray) { }
↑ ↗ ↘
数组元素的类型必须
与循环变量的类型
匹配 此变量在循环过程
中会带有不同元素
的值 ↑
要被逐个运行的集合

冒号 (:) 代表 "in"
重复执行的部分
放在这里

上面这行程序以中文来说就是：“对nameArray中的每个元素执行一次”而编译器会这么认为：

- (1) 创建名称为name的String变量。
- (2) 将nameArray的第一个元素值赋给name。
- (3) 执行重复的内容。
- (4) 赋值给下一个元素name。
- (5) 重复执行至所有元素都被运行为止。

注意：在其他程序语言背景中，这种循环又称为“for each”或“for in”循环。

第一段：声明循环变量。

使用这个部分来声明与初始化用在循环内容的变量。循环过程中此变量所携带的值会有不同。此变量的类型必须要与数组元素匹配。

第二段：要运行的集合。

这必须是对数组或其他集合的引用。

转换primitive主数据类型

将 String 转换成 int

```
int guess = Integer.parseInt(stringGuess);
```

玩家会在游戏提示时输入他的猜测。此猜测会以String的形式传递给checkYourself()方法。

但格子的位置是int，因此你无法将int与String进行比较。

举例来说，下面这程序无法运行：

```
String num = "2";
```

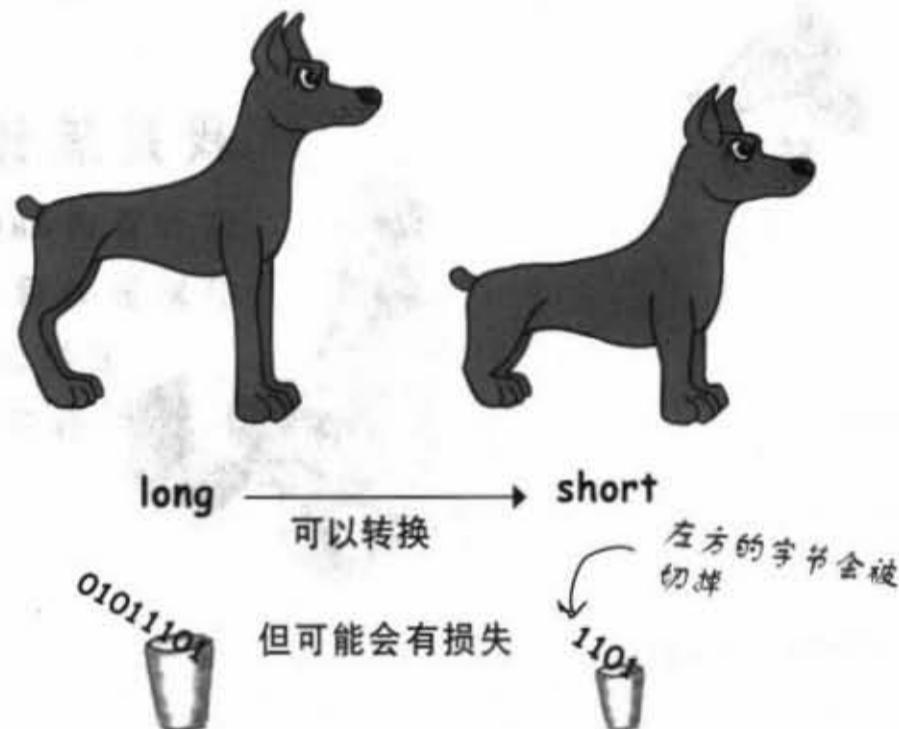
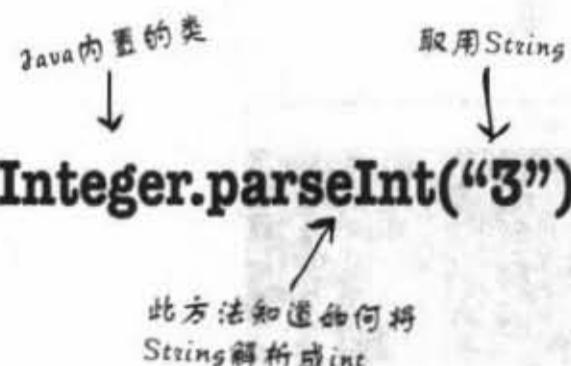
```
int x = 2;
```

```
if (x == num) //这可不行!
```

编译此程序会得到错误信息：

```
operator -- cannot be applied
to int, java.lang.String
if (x == num) { }
```

若要解决这类问题，我们必须要把String的“2”转换成int的2。Java内置有Integer这个类，它有一个方法能够将String所表示的数字转换成实际的数目。



我们在第3章讨论过各种primitive主数据类型的大小，以及小杯子无法装载大杯子的内容物的内容：

```
long y = 42;
```

```
int x = y; // 不能通过编译
```

long比int大，且编译器无法确定long的内容是否可以截掉。若要强制编译器装，你可以使用cast运算符。

```
long y = 42;
```

```
int x = (int) y;
```

前置的类型转换会告诉编译器要将y的值裁剪成int的大小来赋值给x，但这个值可能会很诡异（见附录B）：

```
long y = 40002;
```

```
short x = (short) y; // x的值会是-25534!
```

重点是这样可以通过编译程序。假如说你要取浮点数的整数值：

```
float f = 3.14f;
```

```
int x = (int) f; // x的值会是3
```



我是编译器



这一页的Java程序代码代表一份完整的源文件。你的任务是要扮演编译器角色并判断程序输出会是哪一个？

```
class Output {

    public static void main(String [] args) {
        Output o = new Output();
        o.go();
    }

    void go() {
        int y = 7;
        for(int x = 1; x < 8; x++) {
            y++;
            if (x > 4) {
                System.out.print(++y + " ");
            }
            if (y > 14) {
                System.out.println(" x = " + x);
                break;
            }
        }
    }
}
```

```
File Edit Window Help CM
% java Output
12 14
```

或

```
File Edit Window Help Incense
% java Output
12 14 x = 6
```

或

```
File Edit Window Help Believe
% java Output
13 15 x = 6
```



练习



排排看

右边是被打散的Java程序片段，你是否能够将它们重新排列以成为可以编译与执行并产生如同下方的输出结果？注意到有些括号已经遗失，所以你可以在认为有需要时自行补上。

x++;

if (x == 1) {

System.out.println(x + " " + y);

class MultiFor {

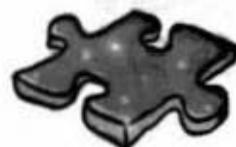
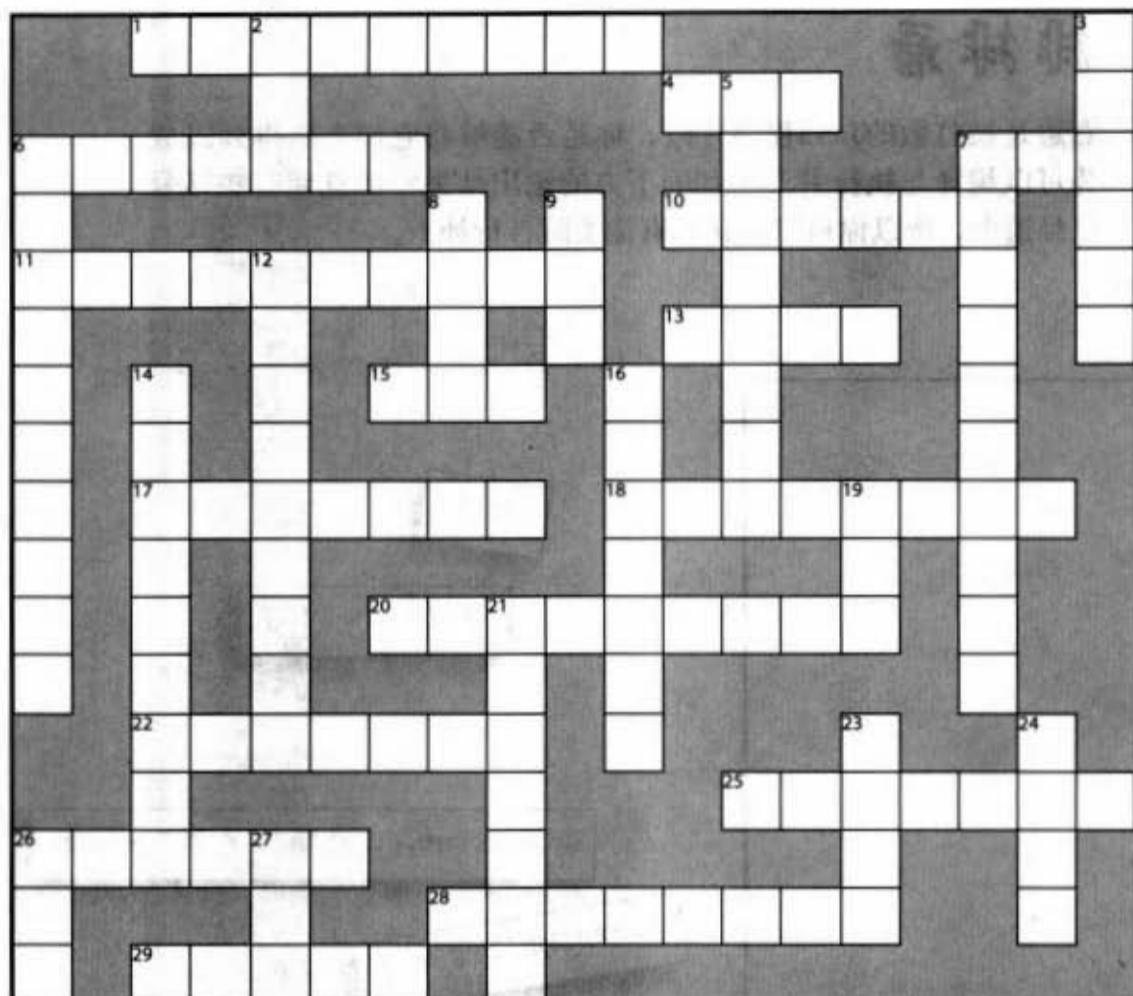
for(int y = 4; y > 2; y--) {

for(int x = 0; x < 4; x++) {

public static void main(String [] args) {

```
File Edit Window Help Raid
% java MultiFor
0 4
0 3
1 4
1 3
3 4
3 3
```

习题



JavaCross

字谜游戏如何能够帮助学习Java呢？这些答案都与Java有关。

横排提示

- 1. Fancy computer word for build
- 2. Automatic toolkit
- 4. Multi-part loop
- 5. Looks like a primitive, but..
- 6. Test first
- 7. Un-castable
- 7. 32 bits
- 8. Math method
- 10. Method's answer
- 11. Converter method
- 11. Precode-esque
- 13. Leave early
- 13. Change
- 15. Towards blastoff
- 15. The big toolkit
- 17. A cycle
- 17. An array unit
- 18. Instance or local

竖排提示

- 2. Increment type
- 3. Class's workhorse
- 5. Pre is a type of _____
- 6. For's iteration _____
- 7. Establish first value
- 8. While or For
- 9. Update an instance variable
- 12. Towards blastoff
- 14. A cycle
- 16. Talkative package
- 19. Method messenger (abbrev.)
- 21. As if
- 23. Add after
- 24. Pi house
- 26. Compile it and _____
- 27. ++ quantity



连连看

下面有一个Java小程序。其中有一段程序不见了。你的任务是找出下面所列出的程序段与相符的输出。

并非所有的输出都有可对应的程序段，且某些输出可能会被使用多次。画条线将相符的两者连接起来。

```
class MixFor5 {
    public static void main(String [] args) {
        int x = 0;
        int y = 30;
        for (int outer = 0; outer < 3; outer++) {
            for(int inner = 4; inner > 1; inner--) {
                 
                y = y - 2;
                if (x == 6) {
                    break;
                }
                x = x + 3;
            }
            y = y - 2;
        }
        System.out.println(x + " " + y);
    }
}
```

程序代码放这里

程序段代码：

x = x + 3;

x = x + 6;

x = x + 2;

x++;

x--;

x = x + 0;

相符的输出：

45 6

36 6

54 6

60 10

18 6

6 14

12 14

把可能的输出和程序代
码连起来。



解答

我是编译器

```
class Output {
    public static void main(String [] args) {
        Output o = new Output();
        o.go();
    }
    void go() {
        int y = 7;
        for(int x = 1; x < 8; x++) {
            y++;
            if (x > 4) {
                System.out.print(++y + " ");
            }
            if (y > 14) {
                System.out.println(" x = " + x);
                break;
            }
        }
    }
}
```

你记住break语句的要素了吗?
它会对输出产生什么样的影响?

File Edit Window Help MotorcycleMaintenance

```
% java Output
13 15 x = 6
```

排排看



亲密度

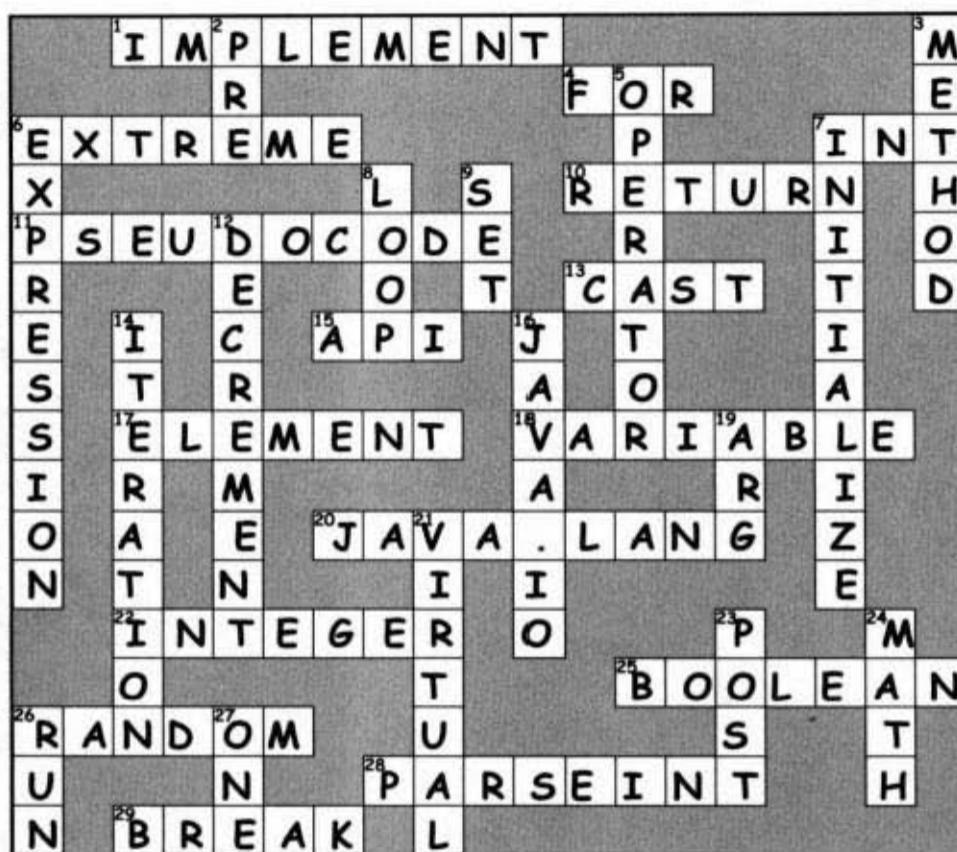
```
class MultiFor {
    public static void main(String [] args) {
        for(int x = 0; x < 4; x++) {
            for(int y = 4; y > 2; y--) {
                System.out.println(x + " " + y);
            }
            if (x == 1) {
                x++;
            }
        }
    }
}
```

如果这个代码块出现在含y的
for循环前，会发生什么?

File Edit Window Help Monopole
% java MultiFor
0 4
0 3
1 4
1 3
3 4
3 3



Java Cross



程序段代码：

x = x + 3;

x = x + 6;

x = x + 2;

x++;

x--;

x = x + 0;

相符的输出：

45 6

36 6

54 6

60 10

18 6

6 14

12 14

6 认识Java的API

使用Java函数库



Java内置有数百个类。如果你知道如何从通称Java API的Java的函数库中选找所需的功能，那么就不用重新发明轮子了。你可以把时间省下来做更有意义的事情。你听说过有些程序员上班总是迟到，而下班又很准时吗？因为他们使用Java API。读完这一章，你也可以这么混。核心Java函数库是由一堆等着被你当作组件使用的类集合而成的。你可以大量运用这些预先创建好的组件来写出你的程序。本书所附已经写好的程序代码让你不需要从无到有慢慢写出功能，但还是需要你自己输入进去。而Java API则根本不用输入，你只需要学会如何运用就好。

有bug

上一章的程序有个bug

正常情况

下面就是运行游戏，输入1, 2, 3, 4, 5, 6的结果，看起来还不错！

完整的程序交互 （你的画面可能会不同）

```
File Edit Window Help Smile
%java SimpleDotComGame
enter a number 1
miss
enter a number 2
miss
enter a number 3
miss
enter a number 4
hit
enter a number 5
hit
enter a number 6
kill
You took 6 guesses
```

出现bug的情况

下面就是执行游戏，输入2, 2, 2的结果。

另一种结果 （哇……）

```
File Edit Window Help Faint
%java SimpleDotComGame
enter a number 2
hit
enter a number 2
hit
enter a number 2
kill
You took 3 guesses
```

在这个版本中，一旦你猜中一格，你就可以持续地猜同一格来击沉dot com！

发生了什么事？

就是这里有问题。
每当玩家猜中某一格时，我们就将计数器加数，而不管之前是否就已经被猜中。

我们需要一种机制来判别之前是否已经被猜中。

```

public String checkYourself(String stringGuess) {
    int guess = Integer.parseInt(stringGuess); ← 把string类型的变量转换成int类型
    String result = "miss"; ← 声明一个变量保存返回结果，默认情况下其值是"miss"

    for (int cell : locationCells) {
        if (guess == cell) { ← 把用户猜测值与数组中的元素比较
            result = "hit"; ← 如果击中的话
            numOfHits++; ←
        }
        break; ← 跳出循环
    } // end if
} // end for

if (numOfHits == locationCells.length) { ← 查看我们当前是否已经"dead"，并把结果改变成"kill"
    result = "kill";
}

System.out.println(result); ← 显示玩家的结果

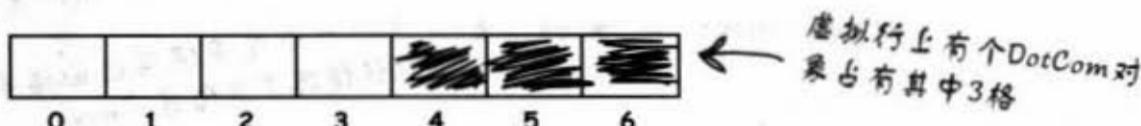
return result; ← 把结果返回给调用的方法
} // end method

```

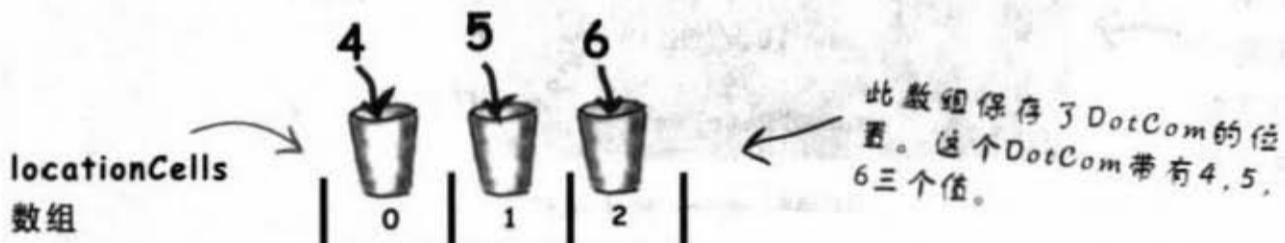
怎么解决？

我们需要一种方法来判别某个格子是否已经被猜中。让我们看看有哪些可能的解法。但首先得看我们已知的部分……

虚拟的行有7个格子，而DotCom会占有其中连续的3格。下面的虚拟列展示出占领4, 5, 6三格的DotCom。

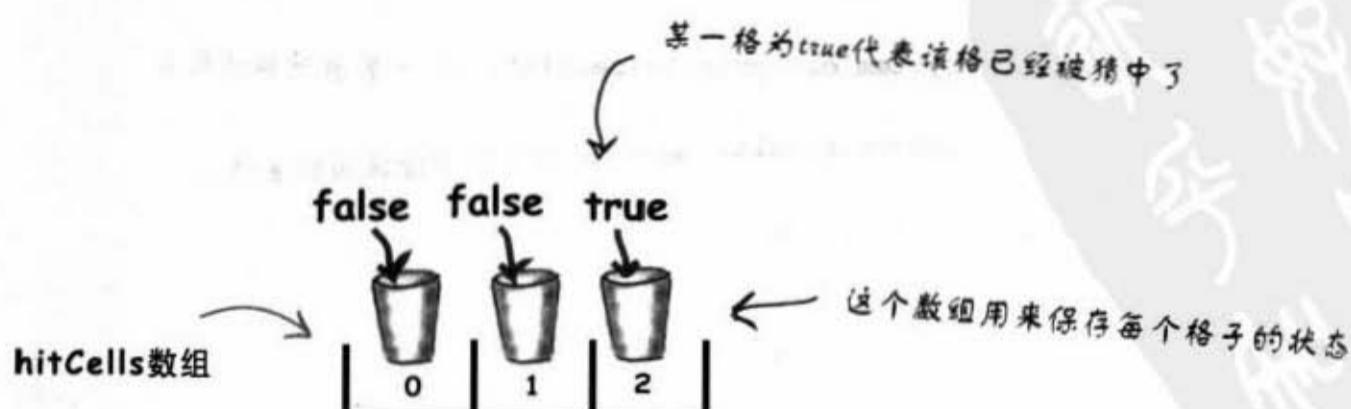


DotCom有个实例变量——一个int数组来保存DotCom对象的位置。



① 方案一

我们可以使用第二个数组。每当玩家猜中某一格时，我们就把相对的那一格设定成true，之后每次猜中都要检查是否在之前就已经被猜过了。



方案一不太优雅

方案一似乎太费时间了。那意味着每次玩家猜中某一格时，你都必须要改变第二个数组的状态，且在之前你还需要检查该数组来判别此格是否已经被命中。这是可行的方案没错，但一定还有更好的方法……

② 方案二

我们可以只动用到原来的数组，办法是将任何被命中的格子改为 -1 。这样只需要维护与检查一个数组。



方案二比较好，但是还不够

方案二比方案一好一点，但不是很有效率。就算有些格子已经被命中了，你还是得逐个搜寻数组中的那些格子。一定还有更好的办法……

③ 方案三

在命中某个格子时就把它删除掉，因此格子就会越来越少。但是数组无法改变大小，因此我们必须作出新的数组并拷贝旧数组的值。



如果数组能够缩小的话，方案三会更好，如此就不用拷贝并重新赋值引用。

原始的伪码：

```

REPEAT with each of the location cells in the int array
    // COMPARE the user guess to the location cell
    IF the user guess matches
        INCREMENT the number of hits
        // FIND OUT if it was the last location cell:
        IF number of hits is 3, RETURN "kill"
        ELSE it was not a kill, so RETURN "hit"
    END IF
    ELSE user guess did not match, so RETURN "miss"
    END IF
END REPEAT

```

如果能够改成这样会更好：

```

REPEAT with each of the remaining location cells
    // COMPARE the user guess to the location cell
    IF the user guess matches
        REMOVE this cell from the array
        // FIND OUT if it was the last location cell:
        IF the array is now empty, RETURN "kill"
        ELSE it was not a kill, so RETURN "hit"
    END IF
    ELSE user guess did not match, so RETURN "miss"
    END IF
END REPEAT

```

如果能够找到一种数组会在删除掉某些元素时自动缩小就好了。这样就不必检查所有的元素，只要查询它是否带有寻找中的值就好。若它还能够让你取出数据而不必管理集合的细节会更好！



别再幻想了

其实真的有这样的集合，但它不是数组，而是个
ArrayList。它是Java函数库中的另一个类。

Java标准版本（除非你用的是小型装置的Micro Edition，不然你用的就是此版）带有数百个预先创建好的类。这就像我们写好给你的程序代码一样，只是说它们已经被编译过了。

这意味着你不用自行输入。

用就对了！

这只是Java函数库数百个类中的一个。

你可以把它们当作自己写的一样使用。

注意：实际上add(Object elem)这个方法比我们此处列出来的更为怪异，这个部分会在本书稍后的章节中解释。现在先当作就是这样。

ArrayList

add(Object elem)	向list中加入对象参数
remove(int index)	在索引参数中移除对象
remove(Object elem)	移除该对象
contains(Object elem)	如果和对象参数匹配返回“true”
isEmpty()	如果list中没有元素返回“true”
indexOf(Object elem)	返回对象参数的索引或-1
size()	返回list中元素的一个数
get(int index)	返回当前索引参数的对象

这只是ArrayList的部分样本而已，实际更为复杂。

ArrayList 的操作

先别管这个 <括号>, 这代表创建出Egg类型的List

① 创建

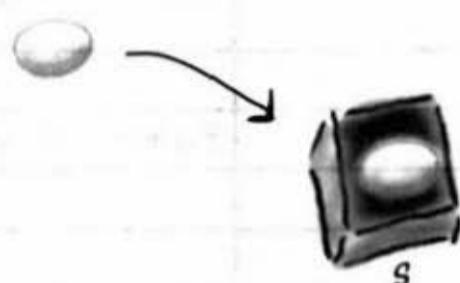
```
ArrayList<Egg> myList = new ArrayList<Egg>();
```

新的ArrayList对象会创建在堆上

② 加入元素

```
Egg s = new Egg();
```

```
myList.add(s);
```

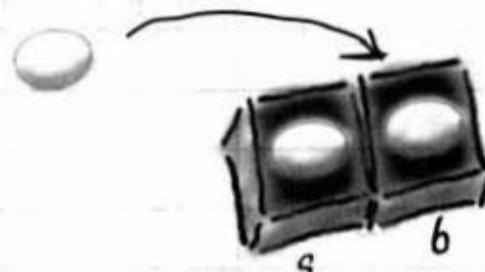


此ArrayList会产生一个“盒子”来放Egg对象

③ 再加入元素

```
Egg b = new Egg();
```

```
myList.add(b);
```



此ArrayList会再弄出一个“盒子”来放新的Egg对象

④ 查询大小

```
int theSize = myList.size();
```

因为myList有两个元素。
size()会返回2

⑤ 查询特定元素

```
boolean isIn = myList.contains(s);
```

因为myList带有s所引用的Egg对象，所以此方法会返回true

⑥ 查询特定元素的位置

```
int idx = myList.indexOf(b);
```

ArrayList为零基的，所以b引用的对象是第二个对象，而indexOf()会返回1。

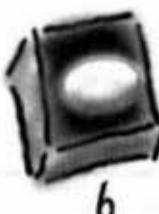
⑦ 判断集合是否为空

```
boolean empty = myList.isEmpty();
```

因为不是空的，isEmpty()会返回false

⑧ 删除元素

```
myList.remove(s);
```



注意，它被缩小了！

数组不够用的时候



观察左方的 ArrayList 程序代码，然后将使用一般数组的相同功能程序代码填入右方空格中。这不是个测验，只是要让你自己动脑思考一下。

ArrayList数组

一般数组

ArrayList<String> myList = new ArrayList<String>();	String [] myList = new String[2];
String a = new String("whoohoo"); myList.add(a);	String a = new String("whoohoo");
String b = new String("Frog"); myList.add(b);	String b = new String("Frog");
int theSize = myList.size();	.
Object o = myList.get(1);	.
myList.remove(1);	.
boolean isIn = myList.contains(b);	.

there are no
Dumb Questions



Java Exposed

本周的来宾：ArrayList

问：我怎么才能知道ArrayList的存在？

答：这个问题其实是在问“我怎么知道API里面有什
么？”。这对于能否成为优秀的Java程序员来说是非常关键的，同时也是让你在不用花太多时间的情况下依然能够创造出应用程序的重点所在。

简单地说就是多花点时间，而更详细的说明会在本章后面的内容中介绍。

问：这个回答等于没有回答。我不仅需要知道Java API带有ArrayList，更重要的是我要知道ArrayList可以解决我的问题！所以我要怎样才能知道有哪些API可以解决哪些问题？

答：年轻人，我能感受到你的热情正在燃烧。如果你能够有耐心一点的话，再过几页我们就会开始讨论这个问题。

HeadFirst: ArrayList其实很像数组吧？

ArrayList: 对不起，我是个对象！

HeadFirst: 如果我没有记错的话，数组也是对象吧。它也是保存在堆上啊。

ArrayList: 没错，但数组总是想要当个ArrayList。我们都应该知道对象有状态与行为对吧？但是你有对数组调用过方法吗？

HeadFirst: 是啊，但是要调用什么功能？我只需要调用放进数组的东西就好，何必调用数组的功能呢？

ArrayList: 是吗？你能从数组中删除元素吗？你在哪里学的Java啊？

HeadFirst: 我可以用Dog d = dogArray[1]来取出元素，不是吗？

ArrayList: 我不是说提取出元素，我说的是删除掉元素，听不懂吗？你那样做只是让d变量引用到数组中的元素所引用的相同对象而已。

HeadFirst: 呃……我听错了。我是没有办法删除元素。

ArrayList: 我可是个高级类呢，所以我可以确实地将引用删除掉，还能够动态地改变我的大小。

HeadFirst: 我真不想继续讨论这个，但是外面有人放话说你只不过是个装模作样的无效率数组罢了。甚至有人还说你根本无法保存primitive主数据类型。那不是很不方便吗？

ArrayList: 这种谎言你也相信？我不是没有效率的数组好不好，我承认有些情况下单纯的数组会比较快，但是谁会为了这一点点效能而放弃了一堆写好又有威力的功能？还有，看看我的弹性再说吧。要保存primitive主数据类型吗？没问题，用个包装类把primitive主数据类型包装起来不就得了吗？甚至在Java 5.0版本上，包装工作会自动进行。我会承认在运用primitive主数据类型的时候数组会比ArrayList快，拜托，现在还有谁在用primitive主数据类型？啊……几点了？我该去健身房了，下次再聊吧！

ArrayList与数组的比较

ArrayList

一般数组

<code>ArrayList<String> myList = new ArrayList<String>();</code>	<code>String [] myList = new String[2];</code>
<code>String a = new String("woohoo");</code>	<code>String a = new String("woohoo");</code>
<code>myList.add(a);</code>	<code>myList[0] = a;</code>
<code>String b = new String("Frog");</code>	<code>String b = new String("Frog");</code>
<code>myList.add(b);</code>	<code>myList[1] = b;</code>
<code>int theSize = myList.size();</code>	<code>int theSize = myList.length;</code>
<code>Object o = myList.get(1);</code>	<code>String o = myList[1];</code>
<code>myList.remove(1);</code>	<code>myList[1] = null;</code>
<code>boolean isIn = myList.contains(b);</code>	<pre>boolean isIn = false; for (String item : myList) { if (b.equals(item)) { isIn = true; break; } }</pre>

这里开始真的很
不一样……

在使用ArrayList时，你只是在运用ArrayList类型的对象，因此就跟运用其他的对象一样，你会使用“.”运算符来调用它的方法。

使用数组时，你会以特殊的数组语法来操作。这样的语法只能用在数组上。虽然说数组也是对象，但是它有自己的规则，你无法调用它的方法，最多只能存取它的length实例变量。

比较 ArrayList 与一般数组

● 一般数组在创建时就必须确定大小

但对于ArrayList来说，你只需要创建出此类型的对象就行。它不需要指定大小，因为它会在加入或删除元素时自动地调整大小。

```
new String[2]      指定大小
new ArrayList<String>()
                  不需要指定大小
```

● 存放对象给一般数组时必须指定位 置

(必须要指定介于0到比length小1之间的数字)

```
myList[1] = b;
          指定索引值
```

如果索引值超越了数组的限制（例如说声明大小为2的数组，然后指派索引值3），程序会在执行期出现错误。

使用ArrayList时，你可以用add(Int, Object)这个形式的方法来指定索引值，或者使用add(Object)的形式来给它自行管理大小。

```
myList.add(b);
          不需指定索引值
```

● 一般数组使用特殊的语法

但ArrayList是个普通对象，所以不会有特殊的语法。

```
myList[1]
          [方括号] 是只用在数组上的特殊语
          法
```

● 在Java 5.0中的ArrayList是参数化的 (parameterized)

虽然我们说ArrayList不像一般数组有特殊的语法，但是它们在Java 5.0中有比较特殊的东西——参数化类型。

```
ArrayList<String>
```

<String>是类型参数。这代表String的集
合，就像说ArrayList<Dog>代表Dog的集合

在Java 5.0之前是无法声明出要存放于ArrayList中元素的类型，它只会是异质对象的集合。现在我们就能用上面列出的语法来声明对象的类型。我们会在讨论Collection的章节对参数化类型作更进一步的探讨。

伪码 测试码 真实码

修改DotCom程序代码

这是有问题的版本：

```

public class DotCom {
    int[] locationCells;
    int numOfHits = 0;

    public void setLocationCells(int[] locs) {
        locationCells = locs;
    }

    public String checkYourself(String stringGuess) {
        int guess = Integer.parseInt(stringGuess);
        String result = "miss";

        for (int cell : locationCells) {
            if (guess == cell) {
                result = "hit";
                numOfHits++;
            }
            break;
        } // out of the loop

        if (numOfHits == locationCells.length) {
            result = "kill";
        }
        System.out.println(result);
        return result;
    } // close method
} // close class

```

我们把类的名称从SimpleDotCom改成
DotCom，但是程序代码与上一章相同

← 有问题的部分，没有考虑到重
复命中同一个格子的问题

伪码

测试码

真实码

全新配方的DotCom类

```

import java.util.ArrayList;
public class DotCom {
    private ArrayList<String> locationCells;
    // private int numHits;
    // 不需要上面这一行了!
    public void setLocationCells(ArrayList<String> loc) {
        locationCells = loc;
    }
    public String checkYourself(String userInput) {
        String result = "miss";
        int index = locationCells.indexOf(userInput);
        if (index >= 0) {
            locationCells.remove(index);
            if (locationCells.isEmpty()) {
                result = "kill";
            } else {
                result = "hit";
            }
        }
        return result;
    }
}

```

Now with
ArrayList
power!

老不管这一行，
稍后说明

将带有String的数组改成承载String的ArrayList

参数有了新名称

判断玩家猜测值是否有出现
在ArrayList中，没有的话全返
回-1

如果索引值大于或等于0，命中！

删除已经命中的格子

如果全部命中清空，
那就表示击沉了！

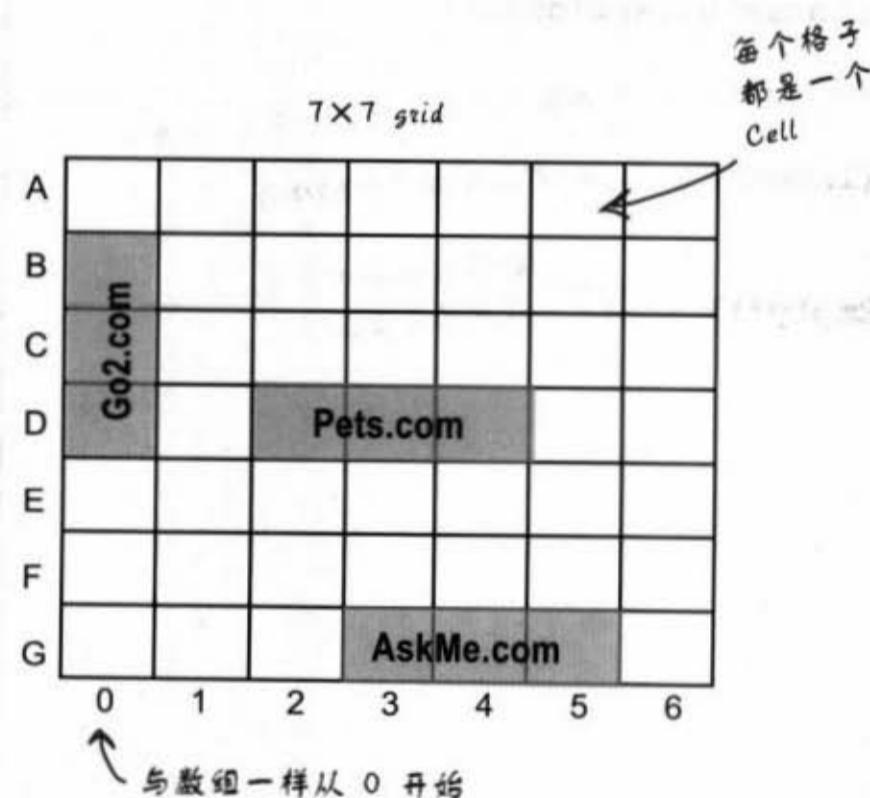
开发真正的 “Sink a Dot Com” 游戏

我们已经创建过简单版的，现在要进行豪华版的开发工作。相对于简单的横列，我们要使用方阵，并且现在有3个DotCom而不是只有一个而已。

游戏目标：以最少的猜测次数打掉计算机所安排的达康公司（Dot Com）网站。计算机会根据你的表现来评分。

初始设置：程序启动后，计算机会在虚拟的 7×7 方格上安排3个达康网站。安排完成后，游戏会要求你开始猜坐标。

进行游戏：因为我们还没有学到图形接口的程序设计，所以这一版会在命令行上进行。计算机会提示你输入所猜测的位置（格子），你会输入“A3”或“C5”等。计算机会回给你命中“Hit”、没中“Miss”或击沉“sunk”等回应。当你清光所有的达康时，游戏会列出你的分数并结束。



你会创建一个攻击网站游戏，它有 7×7 的格子与3间达康公司。每个达康网站占用3个格子。让我们重现网络大崩盘噩梦吧！

游戏进行中的画面

```
File Edit Window Help Sell
%java DotComBust
Enter a guess A3
miss
Enter a guess B2
miss
Enter a guess C4
miss
Enter a guess D2
hit
Enter a guess D3
hit
Enter a guess D4
Ouch! You sunk Pets.com :(
kill
Enter a guess B4
miss
Enter a guess G3
hit
Enter a guess G4
hit
Enter a guess G5
Ouch! You sunk AskMe.com :()
```

要改变什么？

有3个类需要修改：DotCom（以前叫做SimpleDotCom）、游戏的类（DotComBust）与辅助性类（现在先不管）。

A DotCom类

- 增加名称变量

用来保存DotCom的名字，比如说“Pets.com”与“Go2.com”等。如此你就可以在它们被击沉时列出名字。

B DotComBust类(the game)

- 创建出3个DotCom

- 指定DotCom的名称

对每个DotCom的setter调用以设定name这个实例变量。

继续DotComBust类……

- 将DotCom放在方阵上

这个步骤会比简单版更为复杂，因为要考虑到重叠等问题。由于我们不想在这个类中引进复杂的数学问题，所以会把指定DotCom位置的算法放在GameHelper这个已经写好的辅助性类中。

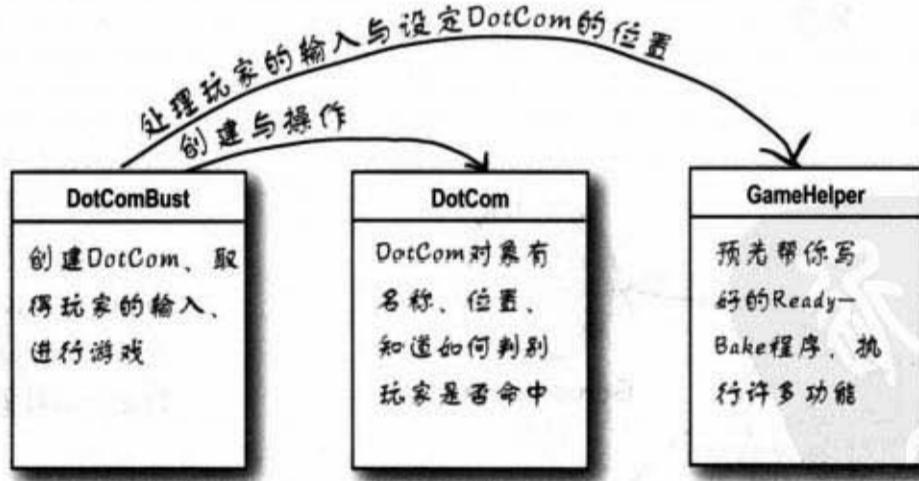
- 每次猜测要检查3个DotCom

- 击沉3个DotCom后才能结束游戏

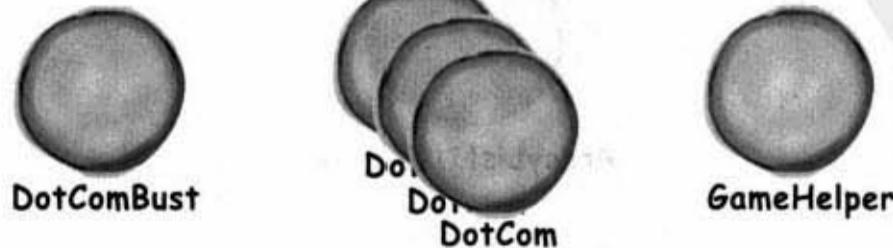
- 脱离main()

简单版之所以会把程序代码放在main()中是因为它是简单版，但是现在我们要写的是豪华复杂版。

3类：

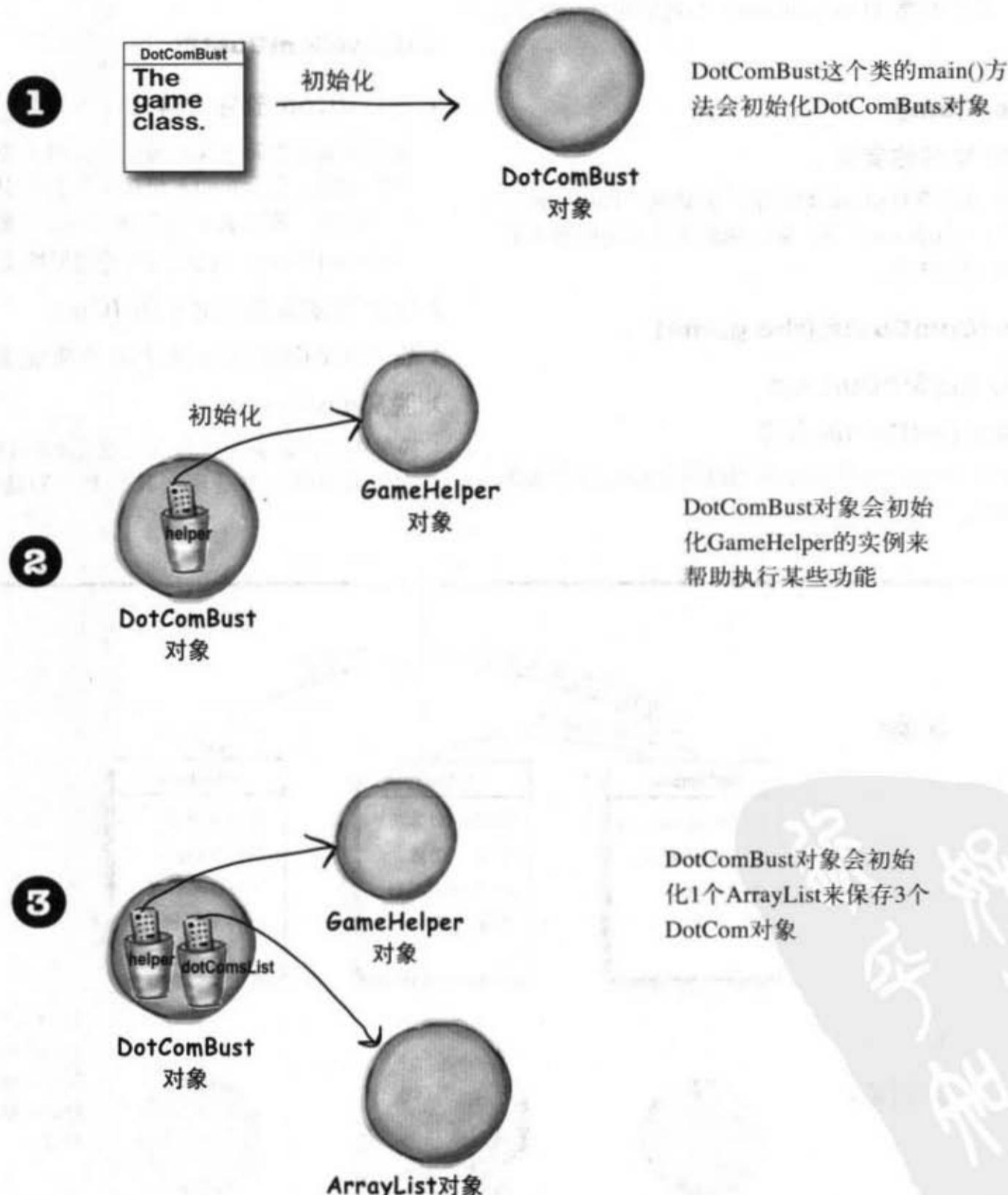


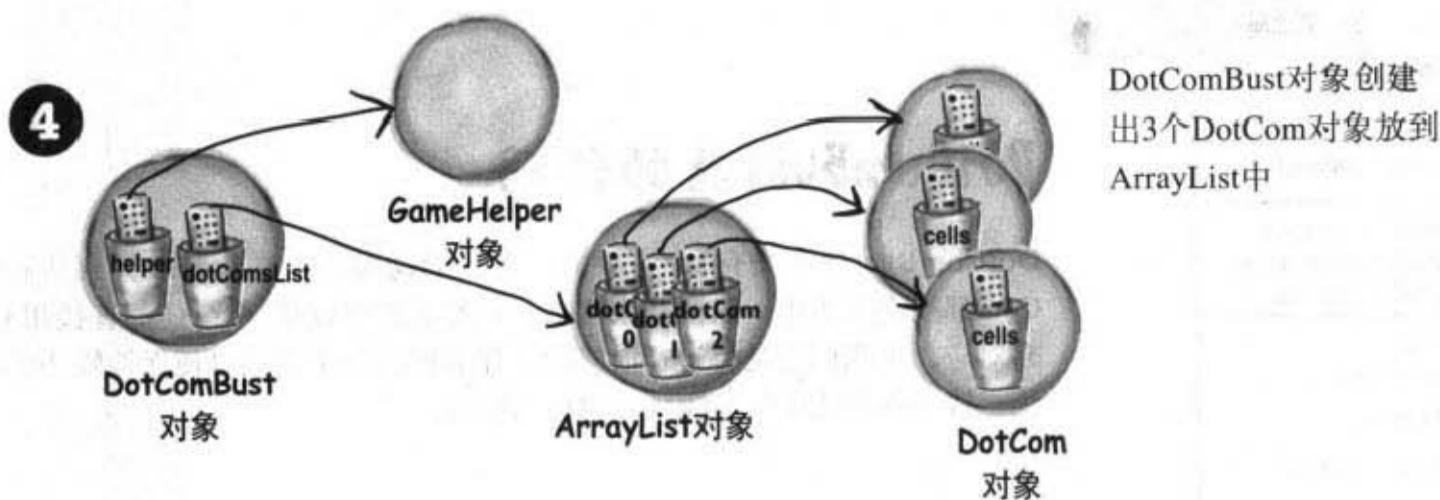
5 对象：



再加上4个
ArrayList：1个给
DotComBust用，其
他3个给DotCom对
象用。

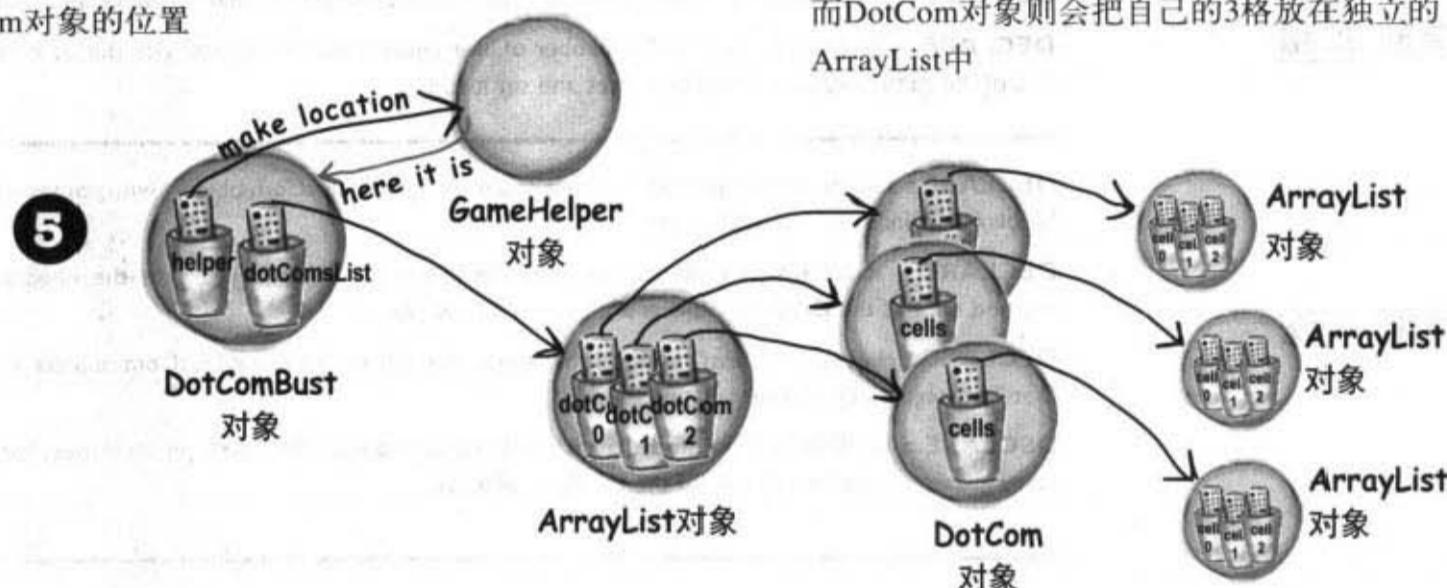
谁在何时做了什么？



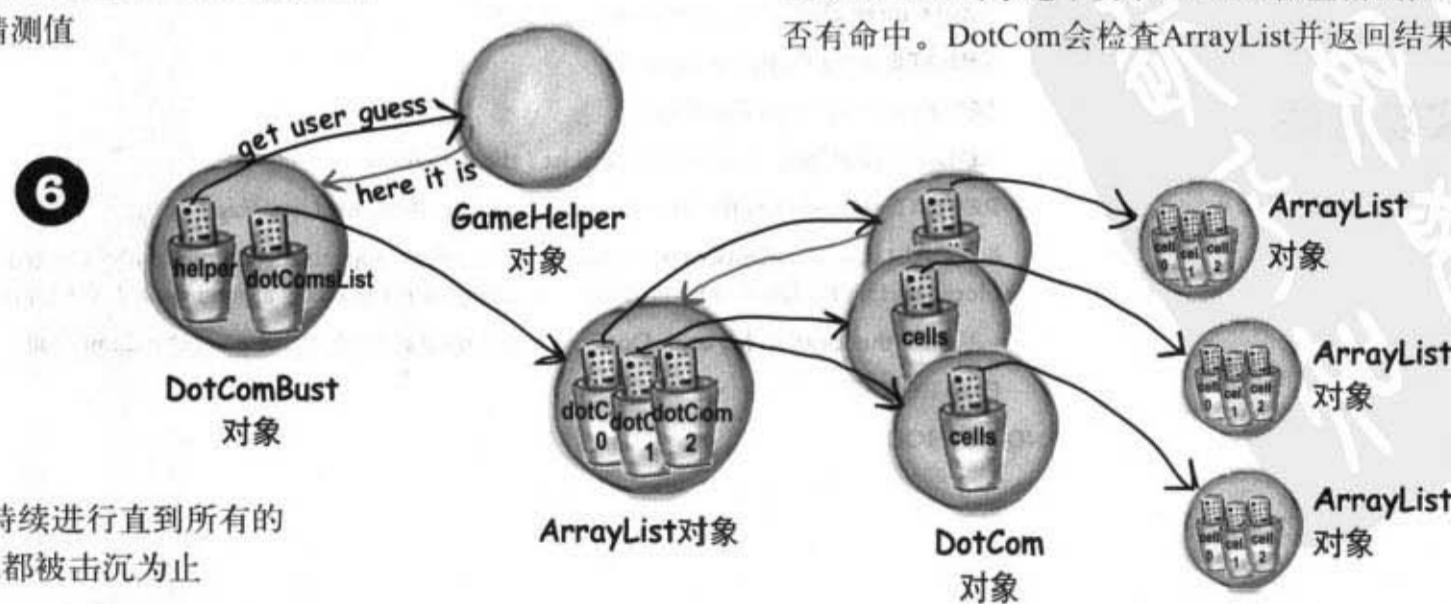


DotComBust对象询问helper对象来设定DotCom对象的位置

DotComBust对象赋值DotCom对象的位置，而DotCom对象则会把自己的3格放在独立的ArrayList中



游戏会持续进行直到所有的DotCom都被击沉为止



伪码 测试码 真实码

DotComBust
GameHelper helper
ArrayList dotComsList
int numOfGuesses
setUpGame()
startPlaying()
checkUserGuess()
finishGame()

声明变量

DotComBust类的伪码

DotComBust这个类有3个主要的任务：启动游戏，进行游戏直到所有的DotCom都被击沉为止，以及结束游戏。虽然我们可以将这3个任务直接用3个方法来对应，但我们还是将进行游戏的工作分割成两个方法以便保持较小的功能模块。较小的方法比较好测试、除错与修改。

声明方法

DECLARE and instantiate the GameHelper instance variable, named helper.

DECLARE and instantiate an ArrayList to hold the list of DotComs (initially three) Call it dotComsList.

DECLARE an int variable to hold the number of user guesses (so that we can give the user a score at the end of the game). Name it numOfGuesses and set it to 0.

DECLARE a setUpGame() method to create and initialize the DotCom objects with names and locations. Display brief instructions to the user.

DECLARE a startPlaying() method that asks the player for guesses and calls the checkUserGuess() method until all the DotCom objects are removed from play.

DECLARE a checkUserGuess() method that loops through all remaining DotCom objects and calls each DotCom object's checkYourself() method.

DECLARE a finishGame() method that prints a message about the user's performance, based on how many guesses it took to sink all of the DotCom objects.

实现方法

METHOD: void setUpGame()

// make three DotCom objects and name them

CREATE three DotCom objects.

SET a name for each DotCom.

ADD the DotComs to the dotComsList (the ArrayList).

REPEAT with each of the DotCom objects in the dotComsList array

CALL the placeDotCom() method on the helper object, to get a randomly-selected location for this DotCom (three cells, vertically or horizontally aligned, on a 7 X 7 grid).

SET the location for each DotCom based on the result of the placeDotCom() call.

END REPEAT

END METHOD

伪码 测试码 真实码

实现方法（续）：

METHOD: void startPlaying()

```

REPEAT while any DotComs exist
    GET user input by calling the helper getUserInput() method
    EVALUATE the user's guess by checkUserGuess() method
END REPEAT
END METHOD

```

METHOD: void checkUserGuess(String userGuess)

```

// find out if there's a hit (and kill) on any DotCom
INCREMENT the number of user guesses in the numOfGuesses variable
SET the local result variable (a String) to "miss", assuming that the user's guess will be a miss.
REPEAT with each of the DotObjects in the dotComsList array
    EVALUATE the user's guess by calling the DotCom object's checkYourself() method
    SET the result variable to "hit" or "kill" if appropriate
    IF the result is "kill", REMOVE the DotCom from the dotComsList
END REPEAT
DISPLAY the result value to the user
END METHOD

```

METHOD: void finishGame()

```

DISPLAY a generic "game over" message, then:
    IF number of user guesses is small,
        DISPLAY a congratulations message
    ELSE
        DISPLAY an insulting one
    END IF
END METHOD

```

Sharpen your pencil

我们要如何将伪码转换成最终的程序代码？我们要先从测试代码开始，然后逐步地创建与测试方法。本书不会持续地列出测试码，所以你应该自行决定要如何测试这些方法。现在问题来了，你

应该要先编写与测试哪个方法呢？尝试看看你是否能够作出测试用的程序代码。看别人写的程序代码或伪码是一回事，自己写出来又是另外一回事。

伪码 测试码 真实码

Sharpen your pencil

```

import java.util.*;
public class DotComBust {
    ① private GameHelper helper = new GameHelper();
    private ArrayList<DotCom> dotComsList = new ArrayList<DotCom>();
    private int numOfGuesses = 0;

    private void setUpGame() {
        // first make some dot coms and give them locations
        DotCom one = new DotCom();
        one.setName("Pets.com");
        DotCom two = new DotCom();
        two.setName("eToys.com");
        DotCom three = new DotCom();
        three.setName("Go2.com");
        dotComsList.add(one);
        dotComsList.add(two);
        dotComsList.add(three);

        System.out.println("Your goal is to sink three dot coms.");
        System.out.println("Pets.com, eToys.com, Go2.com");
        System.out.println("Try to sink them all in the fewest number of guesses");

        for (DotCom dotComToSet : dotComsList) {
            ④ ArrayList<String> newLocation = helper.placeDotCom(3); ⑤
            dotComToSet.setLocationCells(newLocation); ⑥
        } // close for loop
    } // close setUpGame method

    private void startPlaying() {
        while(!dotComsList.isEmpty()) { ⑦
            String userGuess = helper.getUserInput("Enter a guess"); ⑧
            checkUserGuess(userGuess); ⑨
        } // close while
        finishGame(); ⑩
    } // close startPlaying method
}

```

自己批注乐趣多！

把下面的注释与对应的程序代码连接起来，在每个适当的注释前面写下该段程序代码的编号。

每个注释只会使用一次。
所有的注释都会用到。

— 声明并初始化变量 — 取得玩家输入 — 调用这个DotCom的setter方法来指派
 — 列出简短的提示 — 调用checkUserGuess方法 — 判断DotCom的list是否
 — 创建3个DotCom对象并指派名称并置入ArrayList — 对list中的每个DotCom重复一次

伪码 测试码 真实码

```

private void checkUserGuess(String userGuess) {
    numOfGuesses++; ⑪
    String result = "miss"; ⑫
    for (DotCom dotComToTest : dotComsList) { ⑬
        result = dotComToTest.checkYourself(userGuess); ⑭
        if (result.equals("hit")) {
            break; ⑮
        }
        if (result.equals("kill")) {
            dotComsList.remove(dotComToTest); ⑯
            break;
        }
    } // close for
    System.out.println(result); ⑰
} // close method

private void finishGame() {
    System.out.println("All Dot Coms are dead! Your stock is now worthless.");
    if (numOfGuesses <= 18) {
        System.out.println("It only took you " + numOfGuesses + " guesses.");
        System.out.println("You got out before your options sank.");
    } else {
        System.out.println("Took you long enough. " + numOfGuesses + " guesses.");
        System.out.println("Fish are dancing with your options.");
    }
} // close method

public static void main (String[] args) {
    DotComBust game = new DotComBust(); ⑲
    game.setUpGame(); ⑳
    game.startPlaying(); ㉑
} // close method
}

```

千万不要翻到下一页！

只做完这些练习才准翻页。

答案就在下一页。



⑯

— 对list中所有的DotCom重叠

— 这家伙被挂掉了

— 列出结果

— 递增玩家猜测次数的计数

— 列出玩家成绩

— 要求游戏对象启动

— 提前跳出循环

— 先假设没有命中

— 要求DotCom检查是否命中或击沉

— 创建游戏对象

DotComBust 码

伪码 测试码 真实码

```
import java.util.*;
public class DotComBust {

    private GameHelper helper = new GameHelper();
    private ArrayList<DotCom> dotComsList = new ArrayList<DotCom>();
    private int numOfGuesses = 0;

    private void setUpGame() {
        // first make some dot coms and give them locations
        DotCom one = new DotCom();
        one.setName("Pets.com");
        DotCom two = new DotCom();
        two.setName("eToys.com");
        DotCom three = new DotCom();
        three.setName("Go2.com");
        dotComsList.add(one);
        dotComsList.add(two);
        dotComsList.add(three);

        System.out.println("Your goal is to sink three dot coms.");
        System.out.println("Pets.com, eToys.com, Go2.com");
        System.out.println("Try to sink them all in the fewest number of guesses");

        for (DotCom dotComToSet : dotComsList) { ← 对 list 中所有的 DotCom 重复
            ArrayList<String> newLocation = helper.placeDotCom(3); ← 要求 DotCom 的位置
            dotComToSet.setLocationCells(newLocation); ← 调用这个 DotCom 的 setter 方法来指
                派刚取得的位置
        } // close for loop
    } // close setUpGame method

    private void startPlaying() {
        while(!dotComsList.isEmpty()) { ← 判断 DotCom 的 list 是否为空
            String userGuess = helper.getUserInput("Enter a guess"); ← 取得玩家输入
            checkUserGuess(userGuess); ← 调用 checkUserGuess 方法
        } // close while
        finishGame(); ← 调用 finishGame 方法
    } // close startPlaying method
}
```

伪码 流程图 真实码

```

private void checkUserGuess(String userGuess) {
    numOfGuesses++;
    String result = "miss";
    for (DotCom dotComToTest : dotComsList) {
        result = dotComToTest.checkYourself(userGuess);
        if (result.equals("hit")) {
            break;
        }
        if (result.equals("kill")) {
            dotComsList.remove(dotComToTest);
            break;
        }
    } // close for
    System.out.println(result);
} // close method
}

private void finishGame() {
    System.out.println("All Dot Coms are dead! Your stock is now worthless.");
    if (numOfGuesses <= 18) {
        System.out.println("It only took you " + numOfGuesses + " guesses.");
        System.out.println("You got out before your options sank.");
    } else {
        System.out.println("Took you long enough. " + numOfGuesses + " guesses.");
        System.out.println("Fish are dancing with your options");
    }
} // close method
}

public static void main (String[] args) {
    DotComBust game = new DotComBust();
    game.setUpGame();
    game.startPlaying();
} // close method
}

```

列出玩家成绩

↑

↑ 创造游戏对象
↑ 要求游戏对象启动
↑ 要求游戏对象启动游戏的主要循环

↑ 递增玩家猜测次数的计数
↑ 先假设没有命中
↑ 对list中所有的DotCom重复
↑ 要求DotCom检查是否命中或击沉
↑ 提前跳出循环
↑ 这家伙被挂掉了
↑ 列出结果

DotCom 的最终版

```

import java.util.*;

public class DotCom {
    private ArrayList<String> locationCells;
    private String name;
}

public void setLocationCells(ArrayList<String> loc) { ← 更新 DotCom 位置的
    locationCells = loc;
}

public void setName(String n) { ← 基本的 setter 方法
    name = n;
}

public String checkYourself(String userInput) {
    String result = "miss";
    int index = locationCells.indexOf(userInput); ← 使用到 indexOf() 方法！如果玩家猜
    if (index >= 0) {                                         中的话，这个方法会返回它的位
        locationCells.remove(index); ← 置。如果没有的话会返回 -1
        if (locationCells.isEmpty()) { ← 用这个方法来判别是否击沉
            result = "kill";
            System.out.println("Ouch! You sunk " + name + " : ( ");
        } else { ← 列出击沉的信息
            result = "hit";
        } // close if
    } // close if
    return result; ← 返回状态
} // close method
} // close class

```

PDG

超强布尔表达式

到目前为止，我们在if测试或循环中所使用到的布尔表达式都很简单。接下来你会在预先写好的程序代码中看到一些更复杂的布尔表达式。现在先来看一下这些表达式。

“与”和“或”运算符(&&, ||)

假如说你正在编写chooseCamera()方法，它有一些选择相机的规则。或许你会想要选择介于\$50与\$1000之间的相机，但有时你会想要更精确的指定范围。比如说：

“如果价格范围在\$300和\$400之间，就选择X牌相机”

```
if (price >= 300 && price < 400) {
    camera = "X";
}
```

又假如说有10种不同品牌的相机可供选择，你有这样的逻辑来限制品牌：

```
if (brand.equals("A") || brand.equals("B")) {
    // 执行A或B才能进行的工作
}
```

这样的布尔表达式会很大并且很复杂：

```
if ((zoomType.equals("optical") &&
    (zoomDegree >= 3 && zoomDegree <= 8)) ||
    (zoomType.equals("digital") &&
    (zoomDegree >= 5 && zoomDegree <= 12))) {
    // 执行适当的工作
}
```

技术上，你可能会搞不太清楚这些运算符的优先级。与其花时间研究这些规则，不如用括号来让程序代码更容易阅读。

“不等于”运算符(!=和!)

假如说你有这样的规则：有一项规则仅适用于10台相机其中的1台。

```
if (model != 2000) {
    // 非 model 2000 的工作
}
```

或者是这样：

```
if (!brand.equals("X")) {
    // 非X牌的工作
}
```

短运算符(&&, ||)

像&&与||，这些我们已经看过的运算符都称为短运算符。在&&表达式中，左右两边都为true这个表达式才会为true。因此，如果Java虚拟机发现左方的表达式为false，则它不需也不会去计算右方的算式才知道要返回false。||也有相同的特点。所以我们可以用下面这种方式来避免调用内容为null的指针变量的方法：

```
if (refVar != null &&
    refVar.isValidType()) {
    // 执行有效变量的工作
}
```

长运算符(&, |)

&与|运算符使用在boolean表达式时会强制Java虚拟机一定要计算运算符两边的算式。但这两个运算符通常是用来作位的运算。



现成码

这是游戏会使用到的辅助性类。除了取得玩家输入的方法之外，这个类的另外一个最大作用是设置**DotCom**的位置。如果我是你的话，我会先不管这个类，只需要把它编写出来让程序能够编译就好。这个程序用到一些技巧使得它不是很容易理解。

```

import java.io.*;
import java.util.*;

public class GameHelper {

    private static final String alphabet = "abcdefg";
    private int gridLength = 7;
    private int gridSize = 49;
    private int [] grid = new int[gridSize];
    private int comCount = 0;

    public String getUserInput(String prompt) {
        String inputLine = null;
        System.out.print(prompt + " ");
        try {
            BufferedReader is = new BufferedReader(
                new InputStreamReader(System.in));
            inputLine = is.readLine();
            if (inputLine.length() == 0) return null;
        } catch (IOException e) {
            System.out.println("IOException: " + e);
        }
        return inputLine.toLowerCase();
    }

    public ArrayList<String> placeDotCom(int comSize) {
        ArrayList<String> alphaCells = new ArrayList<String>();
        String [] alphacoords = new String [comSize]; // 保存字符串
        String temp = null; // 临时字符串
        int [] coords = new int[comSize]; // 现有字符串
        int attempts = 0; // 目前测试的字符串
        boolean success = false; // 找到适合位置吗?
        int location = 0; // 目前起点

        comCount++;
        int incr = 1; // 现在处理到第n个
        if ((comCount % 2) == 1) { // 水平增量
            incr = gridLength; // 如果是单数号的
        } // 垂直增量

        while ( !success & attempts++ < 200 ) { // 主要搜索循环
            location = (int) (Math.random() * gridSize); // 随机起点
            //System.out.print(" try " + location);
            int x = 0; // 第n个位置
            success = true; // 假定成功
            while (success && x < comSize) { // 查找未使用的点
                if (grid[location] == 0) { // 如果没有使用

```

注意：你可以把 `System.out.print` 这些行的注释去掉，这样会泄漏出位置帮助作弊或者方便测试。



现成码

GameHelper类的程序代码（续）：

```

        coords[x++] = location;           // 储存位置
        location += incr;               // 尝试下一个点
        if (location >= gridSize) {
            success = false;           // 超出下边缘
        }
        if (x > 0 && (location % gridLength == 0)) { // 超出右边缘
            success = false;           // 失败
        }
    } else {                           // 找到已经使用的位置
        // System.out.print(" used " + location);
        success = false;               // 失败
    }
}

} // while结束

int x = 0;                         // 将位置转换成字符串形式
int row = 0;
int column = 0;
// System.out.println("\n");
while (x < comSize) {
    grid[coords[x]] = 1;             // 标识格子已用
    row = (int) (coords[x] / gridLength); // 得到行的值
    column = coords[x] % gridLength;   // 得到列的值
    temp = String.valueOf(alphabet.charAt(column)); // 转换成字符串

    alphaCells.add(temp.concat(Integer.toString(row)));
    x++;
    // System.out.print(" coord "+x+" = " + alphaCells.get(x-1));
}

// System.out.println("\n");

return alphaCells;
}

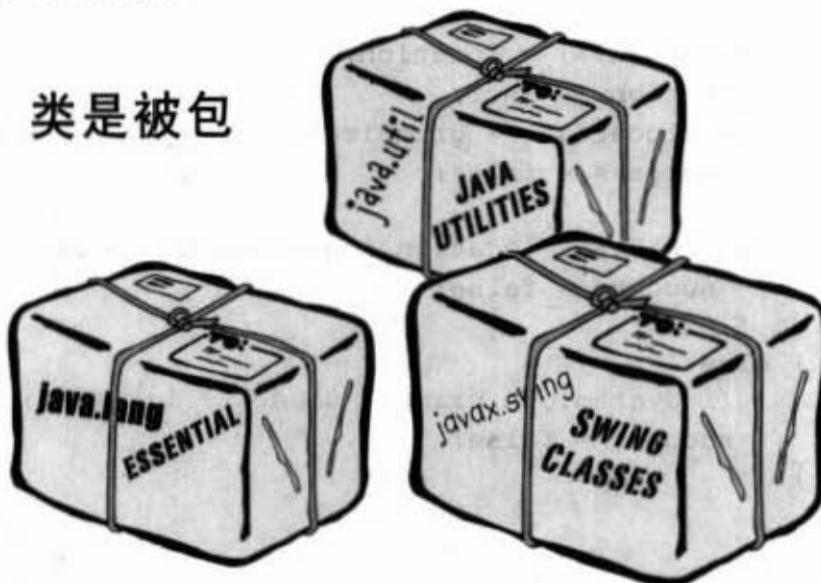
```

这一行会告诉你
DotCom的确实在
位置

使用函数库 (Java API)

我们已经通过了DotComBust游戏的开发过程。这得要感谢ArrayList的帮忙。现在该是来看如何运用Java函数的时候了。

在Java的API中，类是被包装在包中。



要使用API中的类，你必须知道它被放在哪个包中。

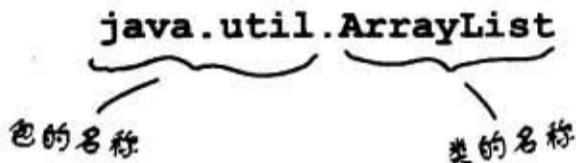
在Java函数库中的每个类都属于某个包。这些包都有个名字，像是javax.swing（里面带有很快就会遇到的Swing接口类）。ArrayList是放在java.util这个包中。顾名思义，java.util放了很多工具类。第16章会讨论有关包的细节，包括如何自制包。

使用来自API的类是很简单的。只要把它们当作是自己写的就好，但是其中还有一个很大的不同：在程序的某个地方你必须要指明函数库类的完整名称，也就是包的名称加上类的名称。

也许你还不清楚，但是实际上已经到了好几个来自API的类。像是System (System.out.println)、String与Math (Math.Random()) 都是属于java.lang这个包。

你必须指明程序代码中所使用到的类的完整名称*

ArrayList并不是它的全名，如同“Mike”也不是全名一样（除非是Madonna或Cher这种）。完整的名称是这样的：



你必须告诉Java想要使用的是哪一个ArrayList。有两种方法可以指定：

A IMPORT

放一个import语句在程序源文件的最前面：

```
import java.util.ArrayList;
public class MyClass { ... }
```

或

B TYPE

或者在程序代码中打出全名。不管在哪里，只要有使用到就打出全名。

声明的时候：

```
java.util.ArrayList<Dog> list = new java.util.ArrayList<Dog>();
```

用在参数的时候：

```
public void go(java.util.ArrayList<Dog> list) { }
```

作为返回类型的时候：

```
public java.util.ArrayList<Dog> foo() { ... }
```

*除非是来自于java.lang这个包中。

*there are no
Dumb Questions*

问： 为何需要全名？这就是包的由来吗？

答： 包之所以很重要有3个原因。首先，它们可以帮助组织项目或函数库相对于一大堆零散的类，以功能来组织会比较好。

其次，包可以制造出名称空间，以便避开相同名称的类。比如说有好几个程序员都设计出Set这个类，我们就可以通过不同的包名称来分辨。

最后，包可以通过限制同一包之间的类才能相互存取以维护安全性。

问： 那要如何防止包的名称也产生冲突？

答： 如果每个程序员都遵循的话，Java的命名传统就能够防止发生这种事情。第16章会有更进一步的探讨。

这个“x”是从哪来的? (javax开头的包代表什么?)

在Java的早期两个版本中(1.02与1.1)，所有随附于Java的类(也就是standard library)都是放在java开头的包中。例如java.lang、java.io、java.util等。

后来出现了一些没有包含在标准函数库中的包。这些被称为扩展的类有两种类型：标准的与非标准的。Sun所认可的称为standard extension，其余实验性质、预览版本或beta版的非标准类则不一定会被认可采用。

标准版的扩展都以javax作为包名称的开头。最早出现的是Swing函数库，它包含的数个包都是以javax.swing开头。从Java 1.2(又称为Java 2)开始，Swing就一并被包含在Java中。

每个人都认为这样很酷，因为如此一来就不必担心用户要如何安装与Swing有关之扩展的步骤。但这些包后来被认可成为标准的一部分，所以Sun在发行1.2版前将开头名称从javax换成了java。很多使用到Swing程序代码的书籍也就这样以新名称印刷发售。

但是非常多的开发者发现这会造成重大的社会写实悲剧，他们之前所写的每一个Swing程序都要跟着改写！想到有多少个import语句是以javax开头就让人痛不欲生……

在最后的关头，开发者终于说服Sun采用“管它的命名传统，先保护程序再说”的方法。所以现在你看到函数库中以javax开头的包就会知道它以前曾经是扩展，后来才取得一个标准名份的。



要点

- ArrayList是个Java API的类。
- 使用add()来新增ArrayList的元素。
- 使用remove()来删除ArrayList中的元素。
- 要寻找某项元素的位置，使用indexOf()。
- 使用isEmpty()来判别ArrayList是否为空。
- 要取得ArrayList的大小，可以使用size()方法。
- 传统的数组可以用length这个变量取得大小。
- ArrayList会自动地调整大小。
- 你可以用参数类型来声明数组内容的类型，例如ArrayList<Button>会声明带有Button类型元素的ArrayList。
- 虽然ArrayList只能携带对象而不是primitive主数据类型，但编译器能够自动地将primitive主数据类型包装成Object以存放在ArrayList中。
- 类会用包来组织。
- 类有完整的名称，那是由包的名称与类的名称所组成的。ArrayList事实上叫做java.util.ArrayList。
- 除了java.lang之外，使用到其他包的类都需要指定全名。
- 也可以在原始程序代码的最开始部分下import指令来说明所使用到的包。

*there are no
Dumb Questions*

问： 使用import会把程序变大吗？编译过程会把包或类包进去吗？

答： 你一定是个C语言程序员。import与C的include并不相同。运用import只是帮你省下每个类前面的包名称而已。程序不会因为用了import而变大或变慢。

问： 那为何我不必import进String类或System类？

答： 要记得java.lang是个预先被引用的包。因为java.lang是个经常会用到的基础包，所以你可以不必指定名称。java.lang.String与java.lang.System是独一无二的class，Java会知道要去哪里找。

问： 我有需要将自己写的类包进包中吗？要怎么做？

答： 在实际应用中，你应该会把类包进包中。第16章会讨论这个问题。现在我们还不需要担心这个。



怕你还没有记得，我们再说一次：



“很高兴知道在java.util这个包中有ArrayList。但是我自己要怎么找呢？”

——Julia, 31岁, 模特



如何查询API?

有两件事情你必须知道：

- ① 库中有哪些类？
- ② 找到类之后，你怎么知道它是做什么的？

1 查阅参考书



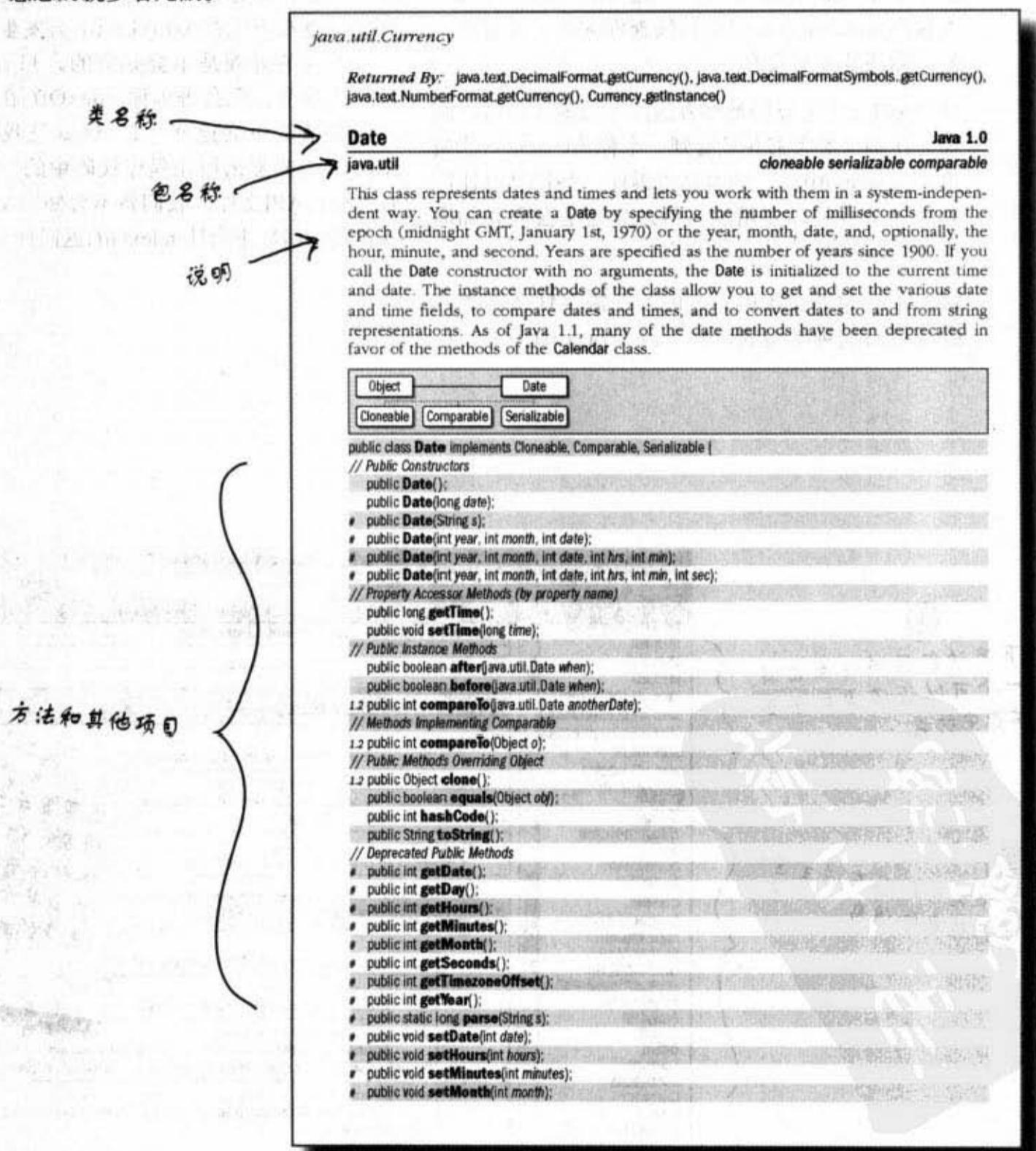
2 查阅HTML API文档

Java 2 Platform Packages	
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.

1 查阅参考书



随意翻阅参考书是找出 Java 函数库有哪些可用类的最佳方式。遇到有意思的就是多看几眼。





2 查阅HTML API文档

Java有个很了不起的在线文件，它很奇怪地被称为Java API。它是Java 5标准版文档的一部分（之前叫做Java 2标准版5.0），并且与Java是分开下载的。如果你联上Internet，并且很有耐心，也可以直接在java.sun.com网站在线查询阅读。相信我，最后你还是会下载的。

这份API文件是查询类与方法细节的最佳参考。假如你在翻阅参考书并且看到一个称为Calendar的类出现在java.util中。书本会告诉你一些信息以让你知道这是否是你要用的，但是你还是会需要知道更多的相关细节。

例如，参考书会告诉你它的方法需要什么参数，以及返回何种类型的数据。以ArrayList为例，在

参考书中你会看到有个称为indexOf()的方法，我们在DotCom这个类中用到它。但你所能知道的也只限于indexOf()取用一个对象参数并返回int类型的索引值，你还是必须要查出最重要的信息：如果对象没有出现在ArrayList中会发生什么事情？光看方法的外观是不会知道的。只有API文件会告诉你细节。它会告诉你indexOf()在找不到相符对象的情况下会返回-1。这就是我们如何知道ArrayList要怎么用在程序代码中的方式。如果没有参考过API文件，我们将不会知道ArrayList对于找不到的情况下会让indexOf()返回什么。

Fields inherited from class java.util.AbstractList

- modCount

Constructor Summary

<code>ArrayList()</code>	Constructs an empty list with an initial capacity of 10.
<code>ArrayList(Collection<? extends E> c)</code>	Constructs a list containing the elements of the specified collection in the order that they are returned by the collection's iterator.
<code>ArrayList(int initialCapacity)</code>	Constructs an empty list with the specified initial capacity.

Method Summary

<code>boolean add(E e)</code>	Appends the specified element to the end of this list.
<code>void add(int index, E element)</code>	Inserts the specified element at the specified position in this list.
<code>boolean addAll(Collection<? extends E> c)</code>	Appends all of the elements in the specified Collection to the end of this list, in the order that they are returned by the specified Collection's iterator.
<code>boolean addAll(int index, Collection<? extends E> c)</code>	Inserts all of the elements in the specified Collection into this list, starting at the specified position.



排排看

右边是被打散的Java程序片段，你是否能够将它们重新排列成为可以编译与运行并产生如同下方的输出结果？注意，你必须查询API文件以找出取用两个参数的add方法的说明。

add(int index, Object o)

```
public static void printAL(ArrayList<String> al) {
```

a.remove(2);

printAL(a);

```
a.add(0, "zero");
a.add(1, "one");
```

printAL(a);

```
if (a.contains("two")) {
    a.add("2.2");
}
```

a.add(2, "two");

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

```
System.out.print(element + " ");
}
System.out.println(" ");
```

```
if (a.contains("three")) {
    a.add("four");
}
```

```
public class ArrayListMagnet {
```

```
if (a.indexOf("four") != 4) {
    a.add(4, "4.2");
}
```

import java.util.*;

printAL(a);

```
ArrayList<String> a = new ArrayList<String>();
```

File Edit Window Help Dance

```
% java ArrayListMagnet
zero one two three
zero one three four
zero one three four 4.2
zero one three four 4.2
```

```
for (String element : al) {
```

```
a.add(3, "three");
printAL(a);
```

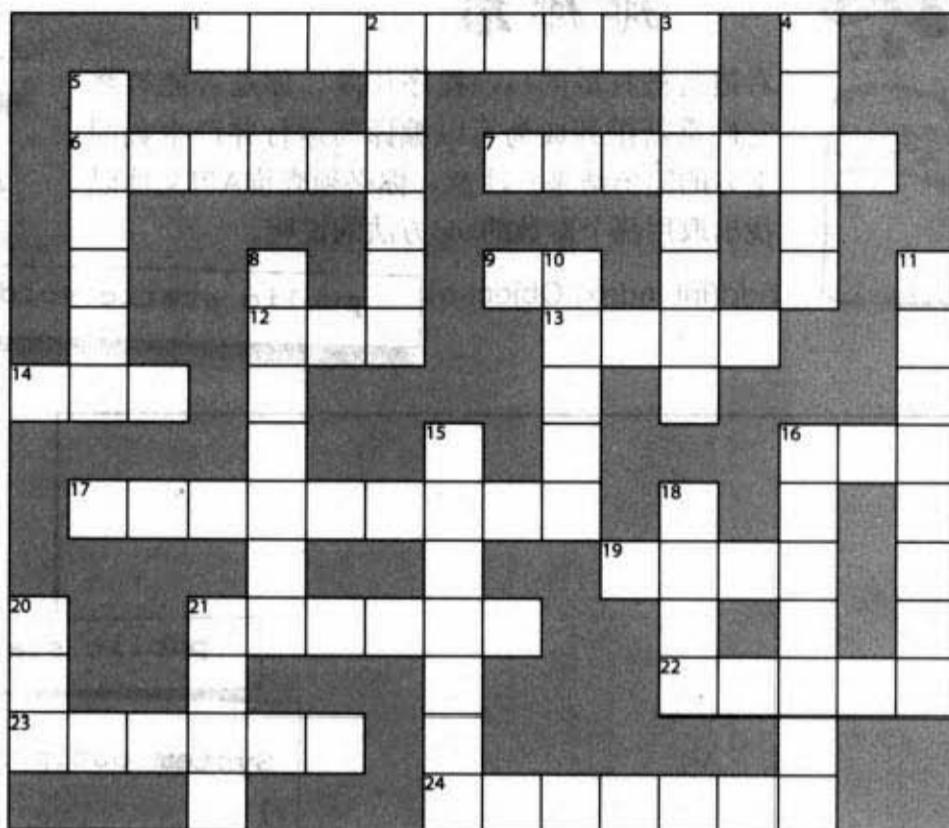


JavaCross 7.0

字谜游戏如何能够帮助学习Java呢？这些答案大部分都与Java以及ArrayList有关。

横排提示：

1. I can't behave
6. Or, in the courtroom
7. Where it's at baby
9. A fork's origin
12. Grow an ArrayList
13. Wholly massive
14. Value copy
16. Not an object
17. An array on steroids
19. Extent
21. 19's counterpart
22. Spanish geek snacks (Note: This has nothing to do with Java.)
23. For lazy fingers
24. Where packages roam



竖排提示：

2. Where the Java action is.
3. Addressable unit
4. 2nd smallest
5. Fractional default
8. Library's grandest
10. Must be low density
11. He's in there somewhere
15. As if
16. dearth method
18. What shopping and arrays have in common
20. Library acronym
21. What goes around

更进一步的提示：

- | | |
|--|-------------------------|
| Across | Down |
| 1. B varieties | 2. What's overridable? |
| 3. Think ArrayList | 4. & 10. Primitive |
| 5. Common primitive | 6. Think ArrayList |
| 7. Think ArrayList | 8. Variables |
| 9. Not about Java - Spanish appetizers | 10. Primitive |
| 11. He's making a | 12. Think ArrayList |
| 13. Wholly massive | 14. Value copy |
| 15. As if | 16. Not an object |
| 17. An array on steroids | 18. Where it's at baby |
| 19. Extent | 20. Library acronym |
| 21. 19's counterpart | 22. Spanish geek snacks |
| 23. For lazy fingers | 24. Where packages roam |



练习解答

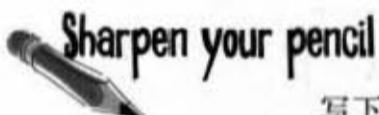
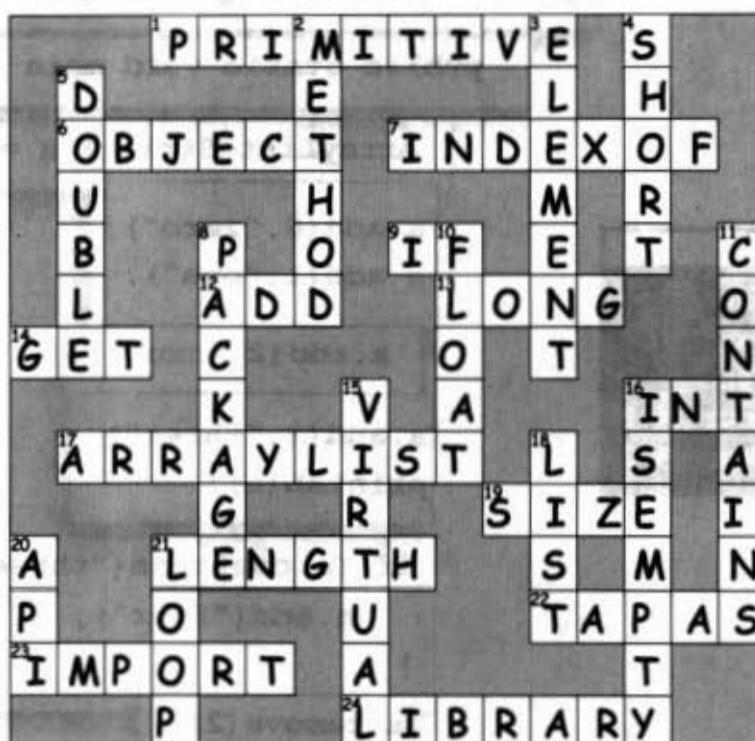
File Edit Window Help Dance

```
% java ArrayListMagnet
zero one two three
zero one three four
zero one three four 4.2
zero one three four 4.2
```

```
import java.util.*;
public class ArrayListMagnet {
    public static void main (String[] args) {
        ArrayList<String> a = new ArrayList<String>();
        a.add(0,"zero");
        a.add(1,"one");
        a.add(2,"two");
        a.add(3,"three");
        printAL(a);
        if (a.contains("three")) {
            a.add("four");
        }
        a.remove(2);
        printAL(a);
        if (a.indexOf("four") != 4){
            a.add(4, "4.2");
        }
        printAL(a);
        if (a.contains("two")) {
            a.add("2.2");
        }
        printAL(a);
    }
    public static void printAL(ArrayList<String> al) {
        for (String element : al) {
            System.out.print(element + " ");
        }
        System.out.println(" ");
    }
}
```



JavaCross 解答



Sharpen your pencil

写下你自己的理解，根据给出答案的每个单词的含义，写下它们简单的、难的或更技术的含义。

Across

1. _____
6. _____
7. _____
9. _____
12. _____
13. _____
14. _____
16. _____
17. _____
19. _____
21. _____
22. _____
23. _____
24. _____

Down

2. _____
3. _____
4. _____
5. _____
8. _____
10. _____
11. _____
15. _____
16. _____
18. _____
20. _____
21. _____

对象村的优质生活

在掌握多态技巧之前，我们的薪水少得可怜，每天又得加班赶工。幸好有多态，现在生活幸福美满，夫妻感情融洽……



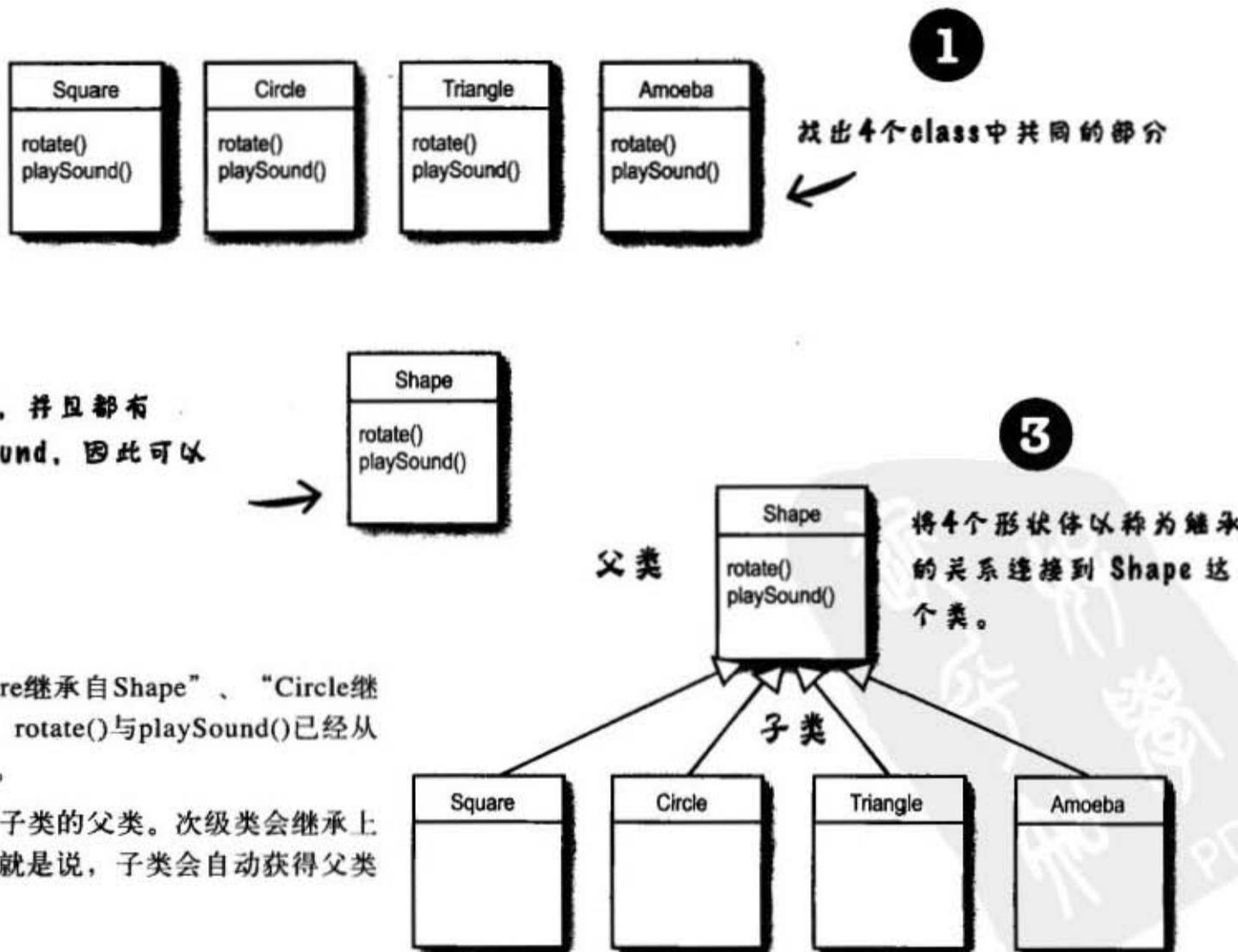
规划程序时要考虑到未来。如果有某种方法能够让你少写点Java程序，多点旅游假期，这对你会有多大的价值？如果你可以写出让他人能够很容易扩充的程序代码呢？你是否对编写出非常有适应性、可以应付最后一刻修改规格的讨厌鬼呢？如果是的话，那你今天真是幸运，因为只要花一个小时，你就可以获得这样的能力。在处理多变的计划时，你会学习到5个更好的设计步骤、3个多态的技巧以及8种让程序更有适应性的方法，此外还有4项对继承的建议。别犹豫了，有这么好的学习机会让你能够获得设计上的自由与程序的适应性，你还不赶快行动？前50名来电者现在还会送你高等抽象的概念！

椅子大战的回顾

还记得第2章的宝椅争夺战吗？我们要从阿珠（面向过程派）与阿花（面向对象派）两人你死我活、惊天地泣鬼神的斗争中查看继承的基本概念。

阿珠：“你有重复的程序代码！4个Shape都有旋转的程序。这样的设计实在很蠢。如此你必须维护4个不同的rotate()方法，这档一点效率都没有”。

阿花：“我猜你一定没有看到最终的设计。阿珠，让我告诉你什么叫做继承”。



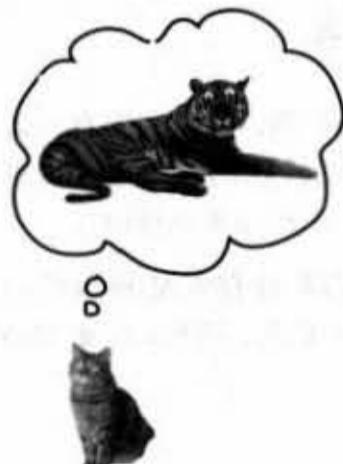
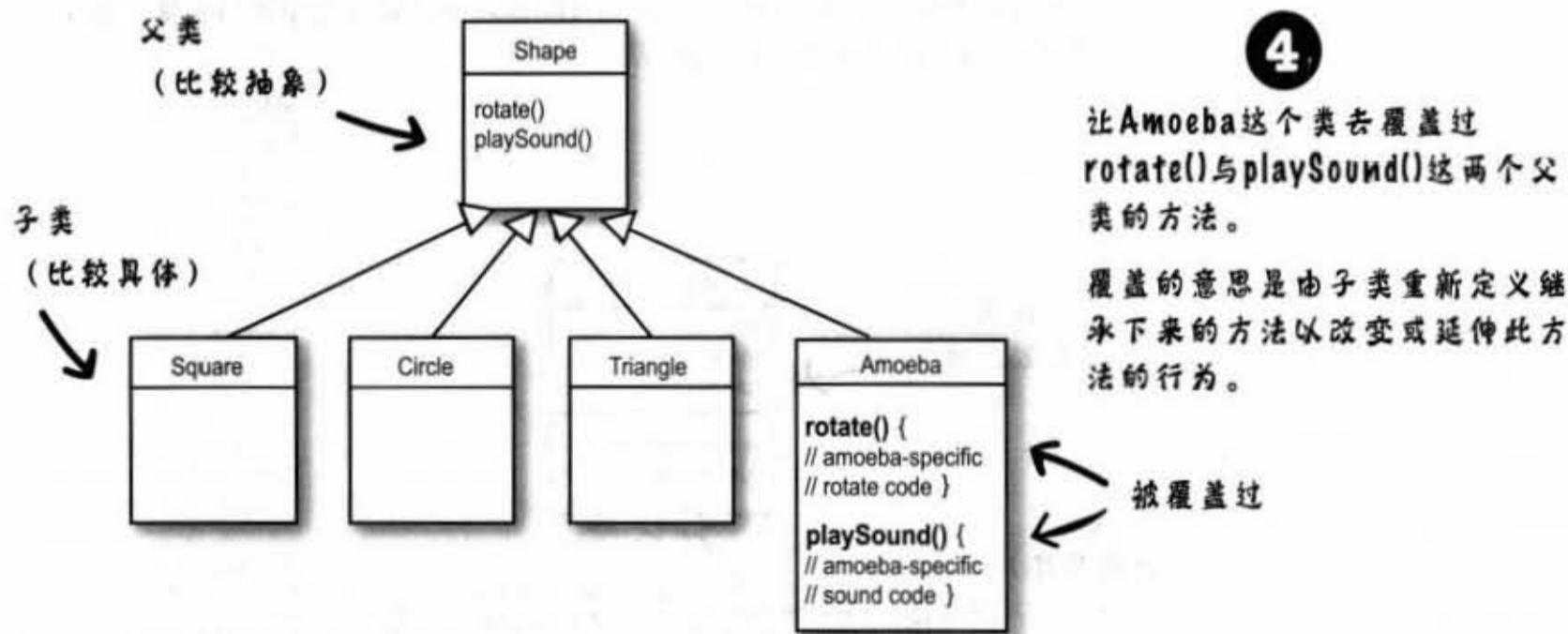
那阿米巴的rotate()要怎么办？

阿珠：“问题不就出在这里吗？阿米巴形状会需要完全不同的rotate与playSound程序”。

阿花：“那叫方法”。

阿珠：“如果阿米巴也是继承自Shape，那旋转的功能不就通通一样吗？”

阿花：“问得好。Amoeba这个类可以覆盖（override）Shape的方法。Java虚拟机会知道在遇到Amoeba时使用不同的rotate()”。



你要如何用继承结构来表示家猫与老虎？被驯养的猫是一种特殊版本的老虎吗？哪一个会是子类？而哪一个才是父类呢？或两者都是某个类的子类呢？

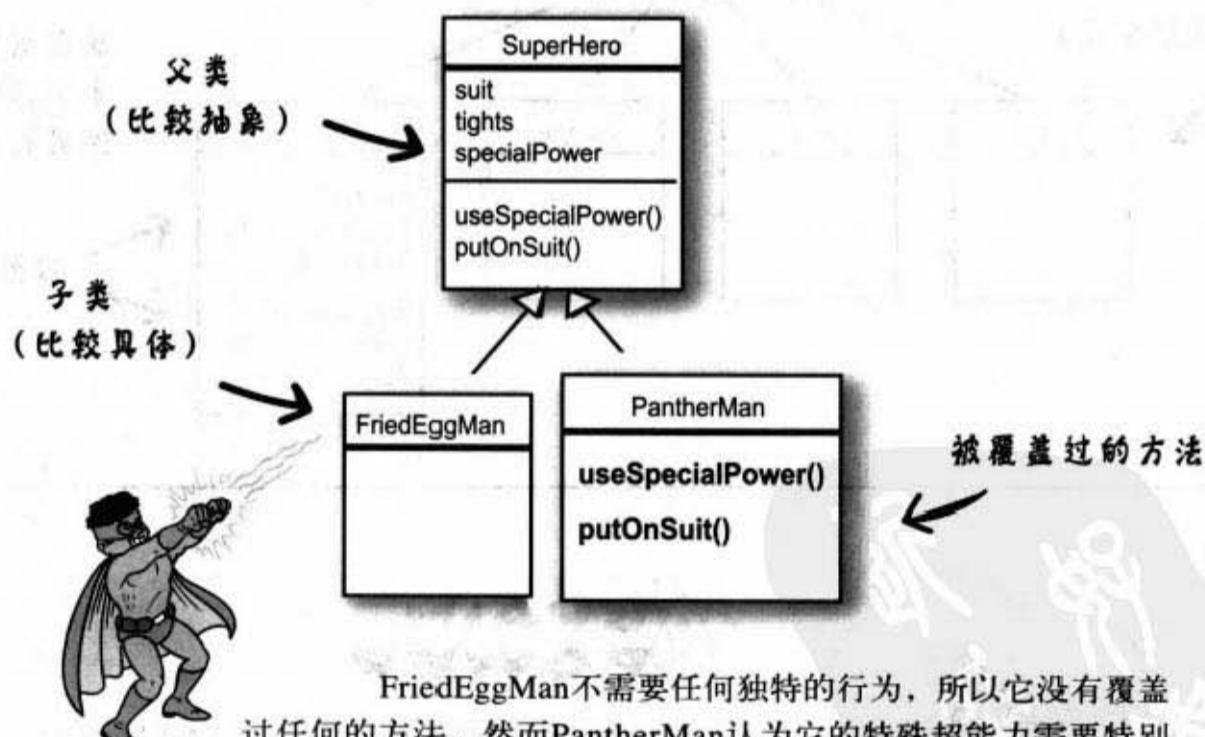
猫一天要睡几小时？方便面应该泡多久才好吃？设计出继承结构之后，会有哪些方法需要被覆盖呢？翻页之前先想一想。

了解继承

在设计继承时，你会把共同的程序代码放在某个类中，然后告诉其他的类说此类是它们的父类。当某个类继承另一个类的时候，也就是子类继承自父类。

以Java的方式说，这是“子类继承父类”。继承的关系意味着子类继承了父类的方法。当我们提及“类的成员”时，成员的意思就是实例变量和方法。

举例来说，如果PantherMan是个SuperHero的子类，则PantherMan会自动地继承SuperHero的实例变量和方法，包括了suit、tights、specialPower、useSpecialPower()等。但PantherMan可以加入自己的方法和实例变量，也可以覆盖掉继承自SuperHero的方法。



FriedEggMan不需要任何独特的行为，所以它没有覆盖过任何的方法。然而PantherMan认为它的特殊超能力需要特别处理过的方法，所以就覆盖掉useSpecialPower()与putOnSuit()。

实例变量无法被覆盖掉是因为不需要，它们并没有定义特殊的行为。PantherMan可以将继承下来的tights设定成紫色，而FriedEggMan可以自行选择白色。

继承的范例

```

public class Doctor {
    boolean worksAtHospital;

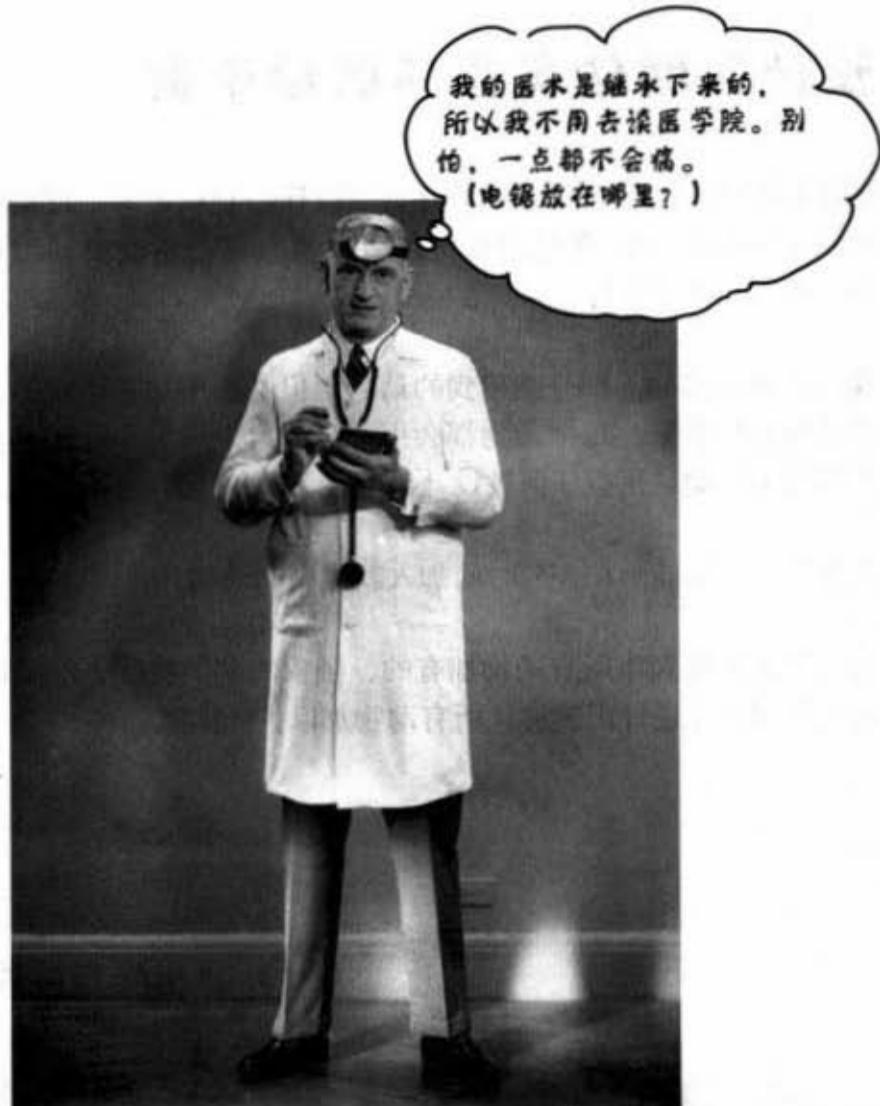
    void treatPatient() {
        // 执行检查
    }
}

public class FamilyDoctor extends Doctor {
    boolean makesHouseCalls;
    void giveAdvice() {
        // 提出诊断
    }
}

public class Surgeon extends Doctor {
    void treatPatient() {
        // 进行手术
    }

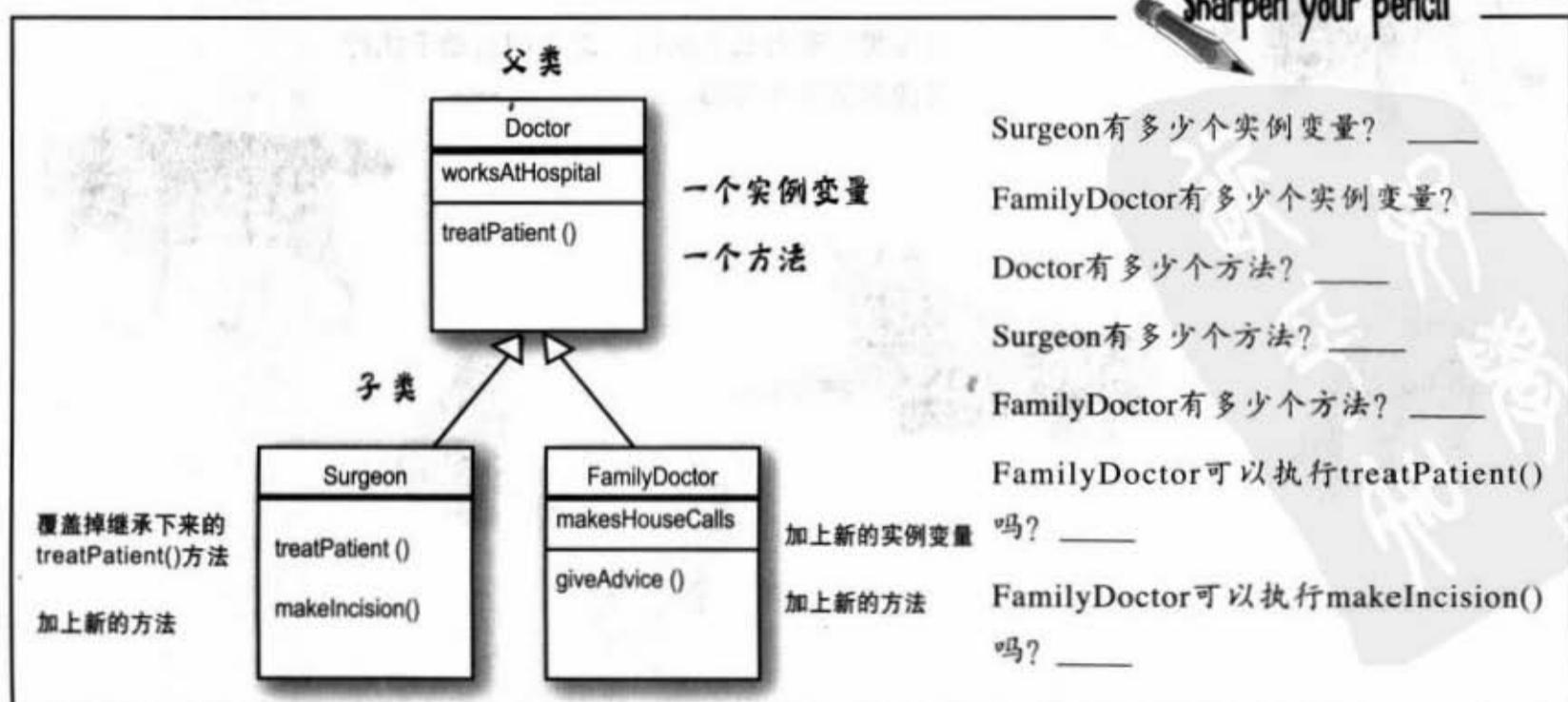
    void makeIncision() {
        // 截肢（好恶心！）
    }
}

```



我的医术是继承下来的，
所以我不用去读医学院。别
怕，一点都不痛。
[电锯放在哪里？]

Sharpen your pencil



设计动物仿真程序的继承树

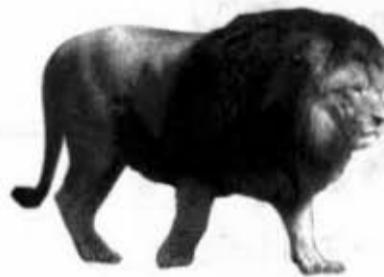
假设你要设计一个仿真系统程序，可以让用户设定将一群动物丢到某种环境中以观察会发生什么事情。现在不需要写出程序，我们只在乎设计。

我们已经被告知一部分会用到的动物，但是并不知道还有多少种动物会加进来。每个动物都会用一个对象来表示，且动物会在环境中活动，执行任何被设计出的行为。

这个程序必须能够在任何时间加入新类型的动物。

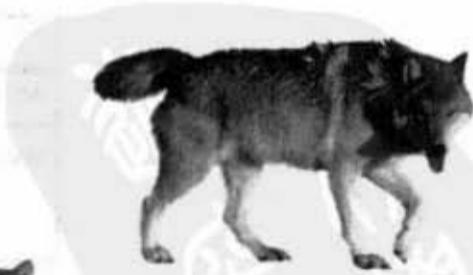
我们首先要辨别出所有动物都有的、抽象的共同特征，然后以这些共同特征设计出能够让所有动物加以扩充的类。

1 找出具有共同属性和行为的对象



这6种动物有什么共同点？这么问可以帮助我们执行第二个步骤。

这些类型有什么相关性？这么问有助于执行第四到第五个步骤。



用继承来防止子类中出现重复的程序代码

我们有5个实例变量：

picture：动物JPEG图像的名称。

food：此动物所吃的食品。现在只有meat和grass两种值。

hunger：代表饥饿程度的int值。它会根据动物吃了多少东西而改变。

boundaries：代表动物活动范围区域的长宽。

location：动物在活动区域中的X与Y坐标。

还有4个方法：

makeNoise()：动物发出声音的行为程序。

eat()：动物遇到食物时的行为程序。

sleep()：睡眠的行为程序。

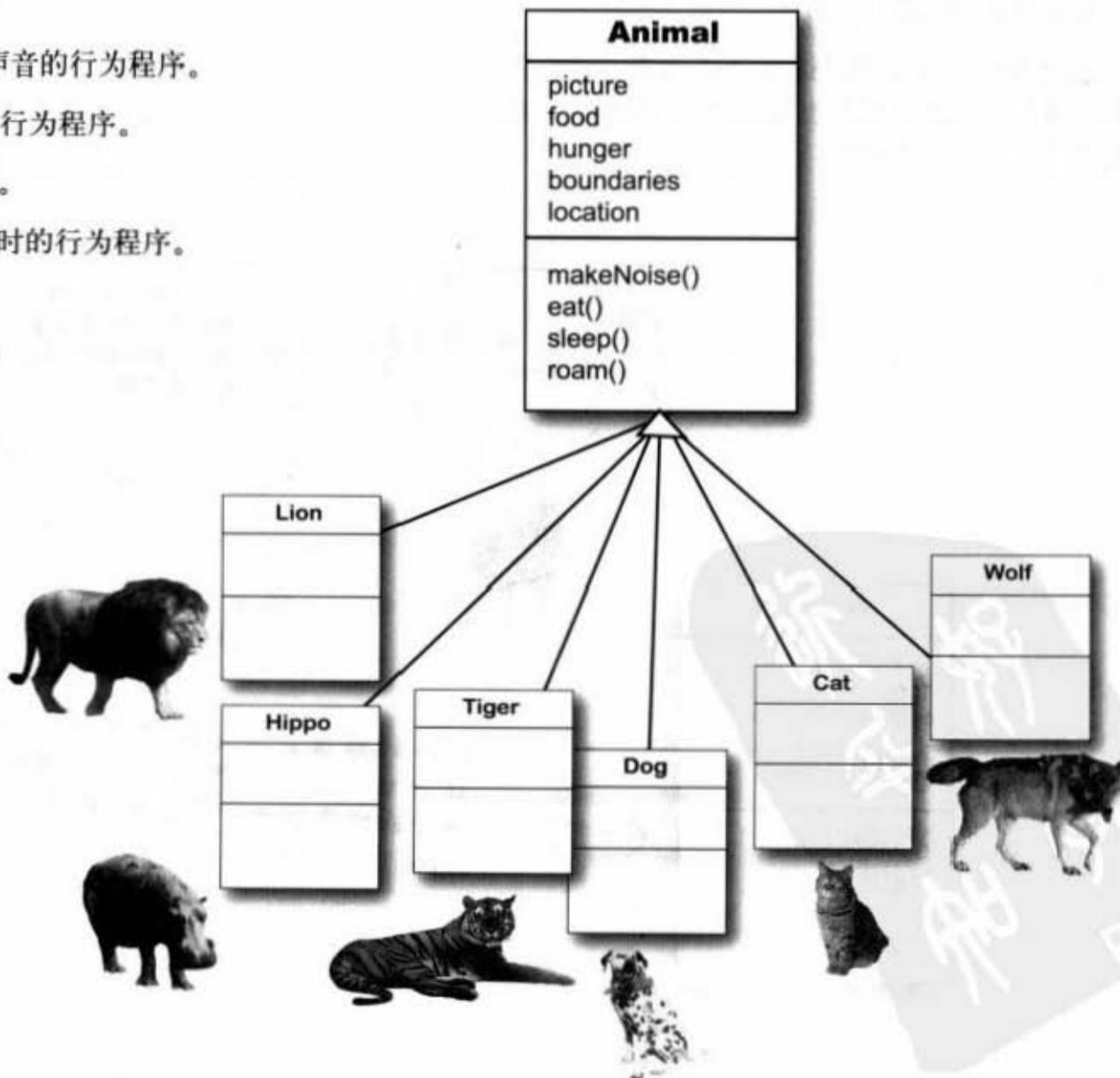
roam()：不在进食或睡眠时的行为程序。

2

设计代表共同状态与行为的类

这些对象都是动物，因此我们可以用Animal作为共同父类的名称。

我们会把所有动物都需要的方法和实例变量加进去。



动物都以同样的方法进食吗？

假设我们都同意一件事：所有Animal类型上的实例变量都公用。狮子的picture带有它的图片路径food的值是meat。猫的图片就是猫、猫的食物是meat（其实有养猫的人都知道猫也会吃草）。所以实例变量没有问题，但是行为程序呢？

我们应该覆盖哪些方法呢？

狮子的叫声会跟河马一样吗？也许你认为一样，但是我们会根据类型设计出不同的行为程序。当然，我们也可以用实例变量来存放声音文件的路径值，而让makeNoise()方法都执行相同动作。但是有时候行为的复杂程度不只是如此而已。

因此就跟阿米巴虫覆盖过rotate()这个方法的例子一样，我们会让某些行为使用各个类自行指定的程序，而不是使用共同的程序。

3

决定子类是否需要让某项行为（也就是方法的实现）有特定不同的运作方式

观察Animal这个类之后，我们认为eat()与makeNoise()应该由各个子类自行覆盖。



寻找更多抽象化的机会

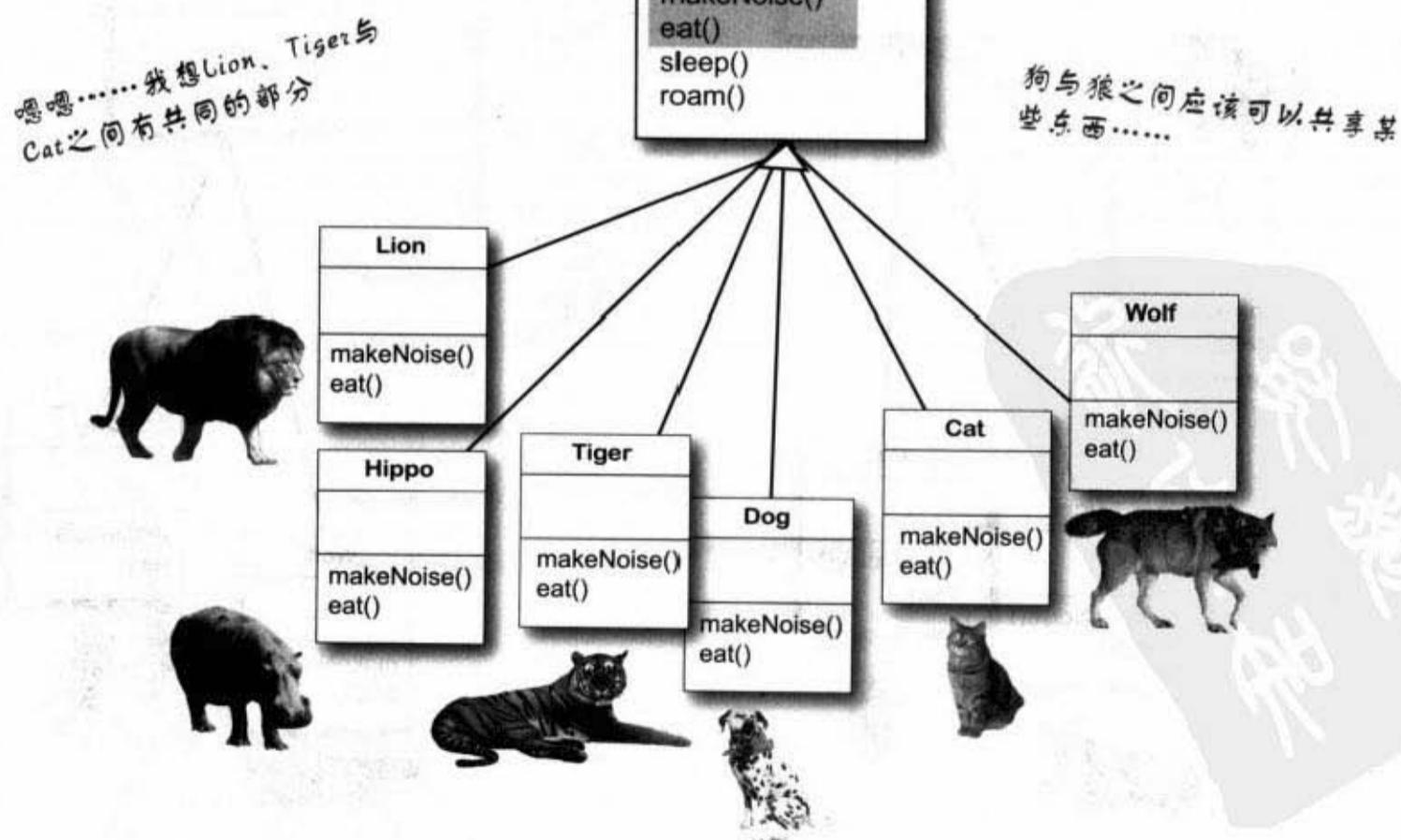
类的继承结构已经大致成型。我们让每个子类都去覆盖掉makeNoise()与eat()这两个方法，因此狗不会喵喵叫（这对双方来说都是种侮辱）、河马也不会抢狮子的食物。

但或许我们还能做更多的设计。我们必须观察Animal的子类找寻是否有可以组织归纳使用共同程序代码的部分。看起来小红帽的好朋友大野狼跟狗有共同的部分。猫、狮子与老虎也有共同的部分。

4

通过寻找使用共同行为的子类来找出更多抽象化的机会

我们观察到Wolf与Dog可能有某些共同的行为，在Lion、Tiger、Cat之间也是。

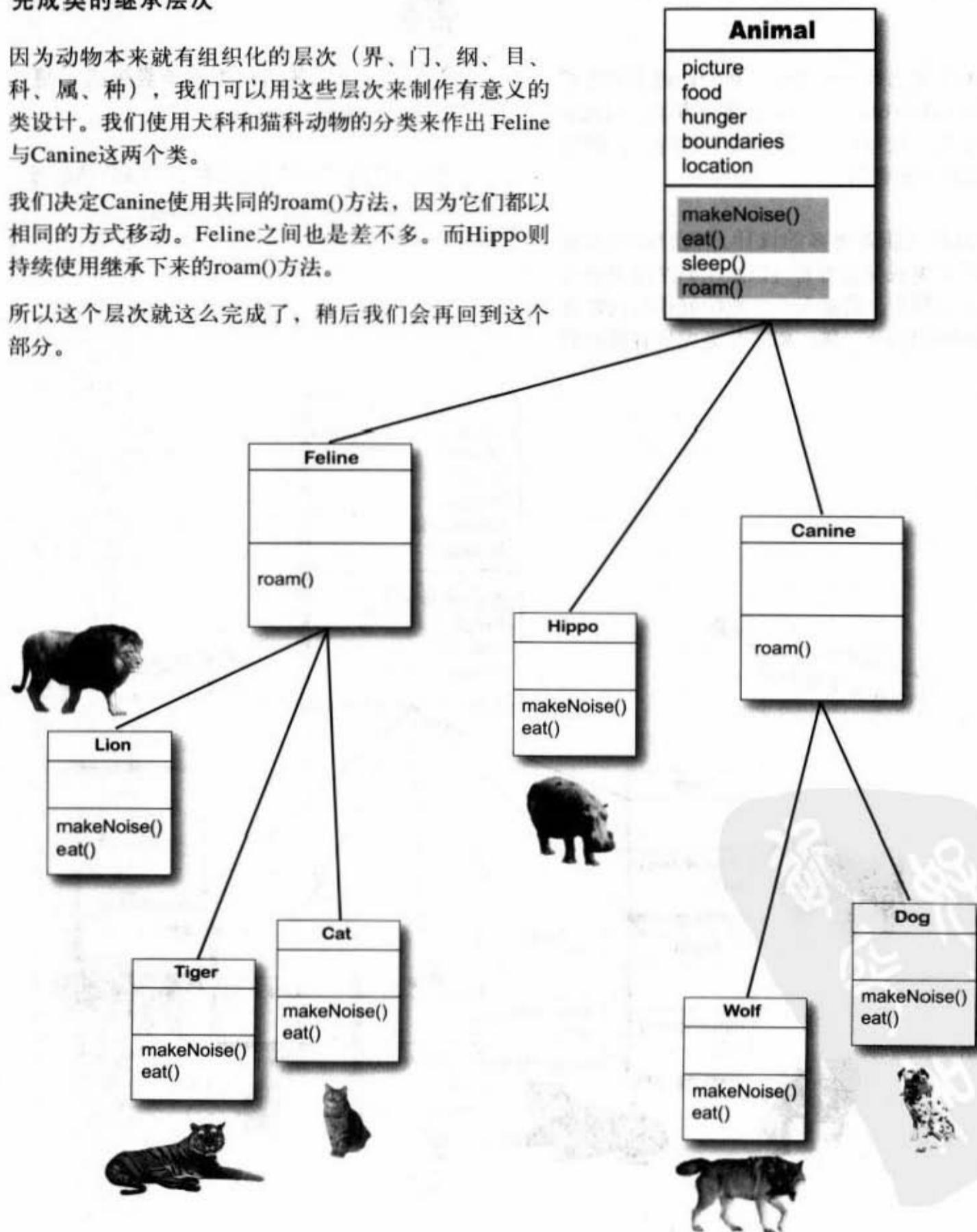


5 完成类的继承层次

因为动物本来就有组织化的层次（界、门、纲、目、科、属、种），我们可以用这些层次来制作有意义的类设计。我们使用犬科和猫科动物的分类来作出 Feline 与 Canine 这两个类。

我们决定 Canine 使用共同的 roam() 方法，因为它们都以相同的方式移动。Feline 之间也是差不多。而 Hippo 则持续使用继承下来的 roam() 方法。

所以这个层次就这么完成了，稍后我们会再回到这个部分。



调用哪个方法？

Wolf这个类有4个方法。其中一个继承自Animal，一个来自Canine（实际上也是覆盖过Animal的方法），还有两个是自己覆盖过的。当你创建出一个Wolf对象并赋给它变量时，你可以使用圆点运算符来调用变量所引用对象的方法。但是这会调用哪个版本的方法呢？

创建大野狼对象

```
Wolf w = new Wolf();
```

调用大野狼的版本

```
w.makeNoise();
```

调用犬科的版本

```
w.roam();
```

调用大野狼版

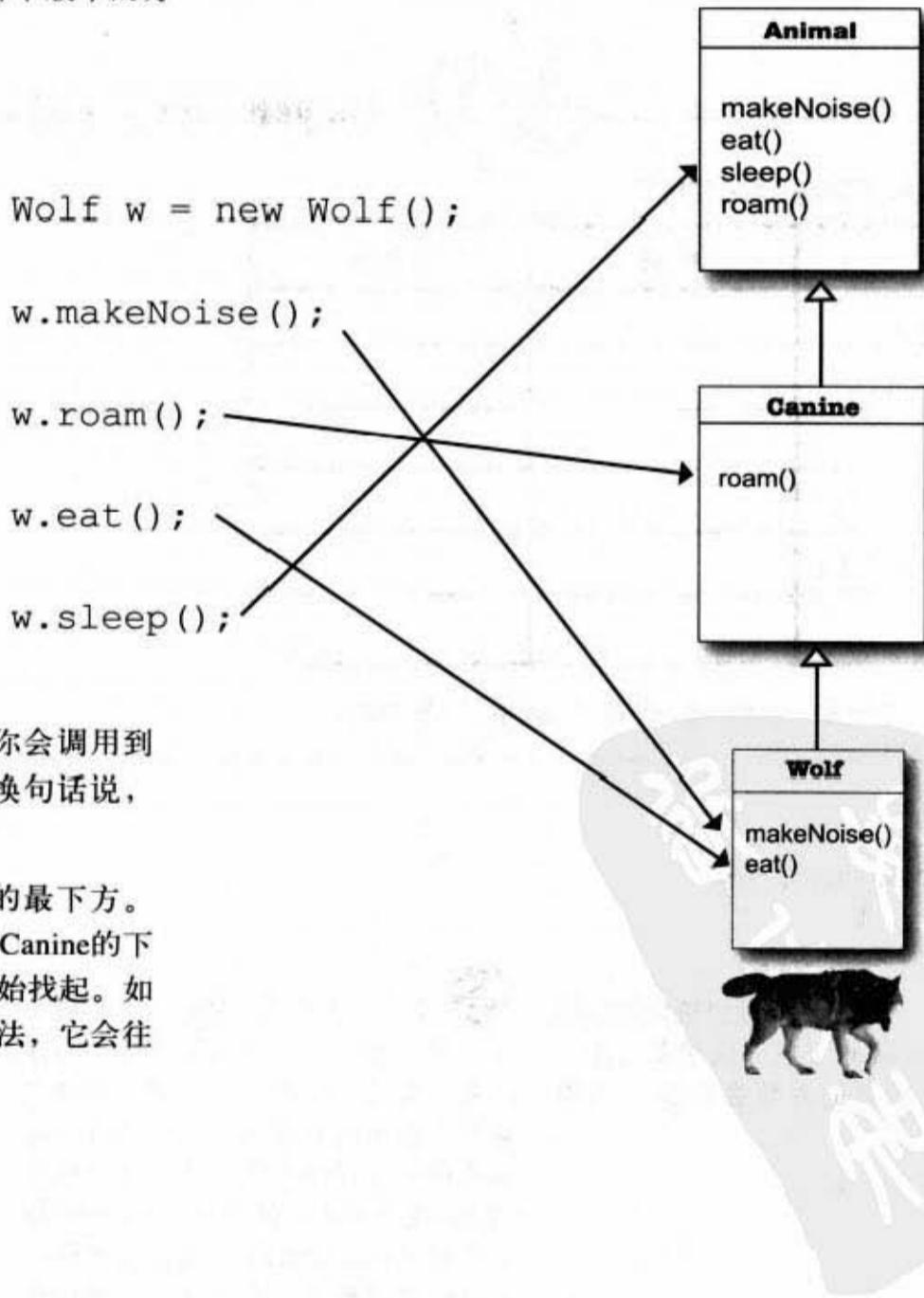
```
w.eat();
```

调用动物版本

```
w.sleep();
```

当你调用对象引用的方法时，你会调用到与该对象类型最接近的方法。换句话说，最低阶的会胜出！

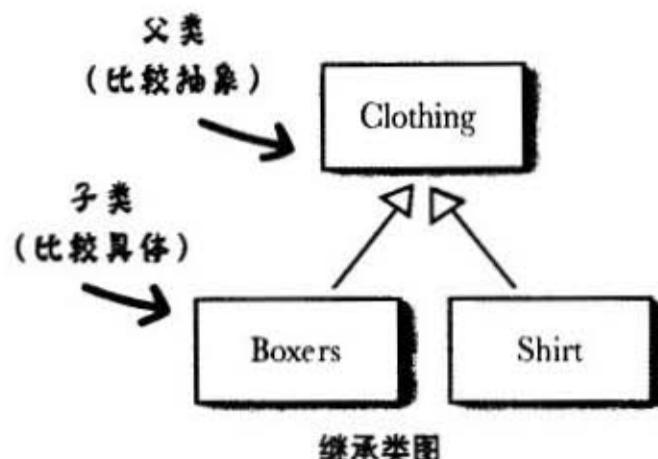
“最低阶”的意思是在层次树的最下方。Canine比Animal低，而Wolf是在Canine的下方，因此Java虚拟机会从Wolf开始找起。如果Java虚拟机找不到Wolf版的方法，它会上寻找直到找到为止。



调用哪个方法？

类	父类	子类
Clothing	---	Boxers, Shirt
Boxers	Clothing	
Shirt	Clothing	

继承表



找出下列类之间的关系填在空格中

类	父类	子类
Musician		
Rock Star		
Fan		
Bass Player		
Concert Pianist		

由上至下为音乐家、摇滚巨星、歌迷、Bass 手、钢琴演奏家。

there are no
Dumb Questions

问：你说Java虚拟机会从继承关系的树形图最下方开始搜索方法，要是没有找到的时候会发生什么事情？

答：好问题！但是你不用担心这件事。编译器会保证引用特定的方法是一定能够被调用到，但在执行期它不会在乎该方法实际上是从哪个类找到的。以Wolf为例，编译器会检查sleep()这个方法，但却不管sleep()实际上是定义在Animal这个类。要记得如果某个类继承了一个方法，它就会有

那个方法。方法在哪里定义对于编译器来说不重要。但在执行期，Java虚拟机就是有办法找到正确的。这个正确的意思是最接近该类型的版本。

“是一个”与“有一个”

当一个类继承自另外一个类时，我们会说这是子类去继承父类。若你想要知道某物是否应该要继承另一物时，则可以用 IS-A 测试来检验。

三角形是一个多边型……嗯，没错。

外科医生是一个医生……OK。

哈啰凯蒂是一个猫……算是吧。

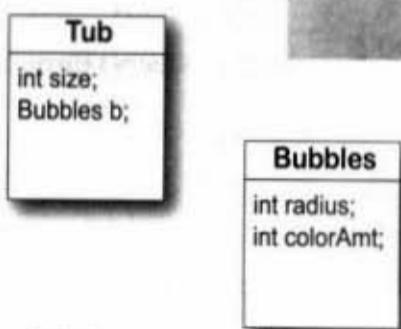
澡盆是一个浴室……失败！

大肠没有洗干净……失败中的失败！

要确认你的设计是否正确，使用这样的测试来加以检验。不合理，表示你的设计有问题。

浴室与澡盆确实有关联，但不是继承上的关系。浴室与澡盆发生的是 HAS-A 的关系。如果“浴室有一个澡盆”成立的话，这表示浴室带有澡盆的实例变量。也就是说浴室会有个澡盆的引用，但是浴室并没有继承过澡盆。

澡盆是个浴室吗？
浴室是个澡盆吗？对我来说两者皆非。澡盆与浴室的关系是一种“A 有一个B”的形式。浴室有一个澡盆。这意味着浴室有个澡盆的实例变量。



浴室有一个澡盆且澡盆有一个泡泡。

但这里没有继承关系。

等一下！还有！

IS-A测试适用在继承层次的任何地方。如果你的继承层次树设计得很好，那么所有的子类都应该通过任一个上层父类的IS-A测试。

如果类Y是继承类X，且类Y是类Z的父类，
那么Z应该能通过IS-A X的测试。

Canine 继承 Animal

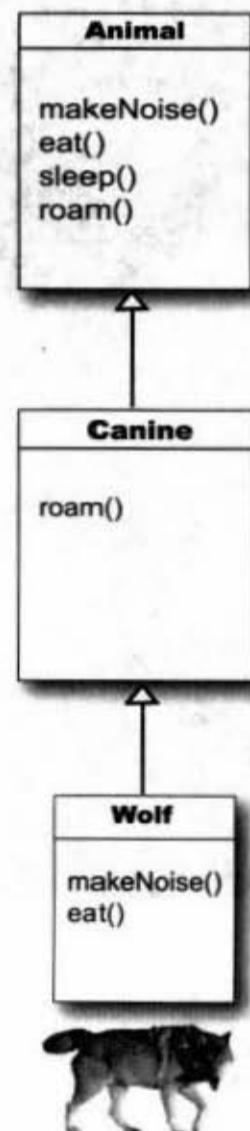
Wolf 继承 Canine

Wolf 继承 Animal

Canine IS-A Animal

Wolf IS-A Canine

Wolf IS-A Animal



就像此处所显示出的继承树，你一定可以说：“Wolf继承Animal”或“Wolf IS-A Animal”。只要Animal位于Wolf之上，Wolf IS-A Animal就一定会成立。

这张Animal继承图说明了：

“Wolf是一个Canine，因此Canine能做的事情Wolf都能做，而Wolf也是个Animal，所以Animal能做的事情Wolf也都能做”。

就算Wolf已经覆盖掉某些来自Animal或Canine的方法也一样。对其他的程序来说，它们只要知道Wolf能够执行这4个方法就行。至于它是怎么做的，或者它是覆盖过哪个类的，则一点都不重要。至少有一件事情可以确定，Wolf一定可以`makeNoise()`、`eat()`、`sleep()`和`roam()`。

如何知道继承设计是对的？

很明显，事情不只是这样而已，但是我们会在下一章讨论更多有关面向对象的概念（最终我们还是得改善这一章所作出的设计）。

虽然如此，我们还是要提出IS-A测试这个建议。如果“X IS-A Y”合理，则这两者或许存在于同一个继承结构下。也有可能两者根本是相同的，或者刚好有相同的行为。

注意：继承概念下的IS-A是个单向的关系！

“三角形是一个多边形”这是合理的，所以你可以从多边形中扩充出三角形。

但是反过来说“多边形是一个三角形”并不合理，所以多边形并不是从三角形中extend出来的。要记得X IS-A Y隐喻着 X 可以做出任何Y可以做的事情（且还可能会做出更多的行为）。



Sharpen your pencil

在合理的叙述旁边打勾

- Oven extends Kitchen**
- Guitar extends Instrument**
- Person extends Employee**
- Ferrari extends Engine**
- FriedEgg extends Food**
- Beagle extends Pet**
- Container extends Jar**
- Metal extends Titanium**
- GratefulDead extends Band**
- Blonde extends Smart**
- Beverage extends Martini**

提示：运用IS-A测试。

谁家的小孩？

there are no Dumb Questions

问： 我们已经看过子类是如何继承父类的方法，但如果父类想要使用子类的方法时该怎么办？

答： 父类不一定要知道它的子类。或许有人会写出一个类并且在多年之后被他人扩充过。回到问题，就算父类与子类是同一个人编写的，继承的方向也不可能反过来。你可以想象小孩继承了父母的遗传，而遗传不会有别的方向。

问： 如果在子类中还打算引用父类的方法然后再加上额外的行为应该怎么办？

答： 这是可行的！事实上这个功能非常重要。扩充本来就有扩充和延伸的意思。

```
public void roam() {  
    super.roam();  
    // my own roam stuff  
}
```

你可以在父类中设计出所有子类都适用的功能实现。让子类可以不用完全覆盖掉父类的功能，只是再加上额外的行为。你可以通过super这个关键词来取用父类。

这会先执行super版的roam()然后再执行sub版自定义的行为或功能



谁开保时捷？又有谁开保肝丸？

(如何知道子类能够继承下来哪些东西？)



子类可以继承父类的成员。这包括了实例变量和方法，然而稍后我们才会提到其他会被继承的东西。父类可以通过存取权限决定子类是否能够继承某些特定的成员。

在这本书中，我们会讨论4种存取权限。下面列出这4种权限，左边是最受限制的，而越往右边限制程度越小：

private default protected public

存取权限 (access level) 控制了谁可以接触什么，这对编写出坚固、设计良好的Java程序来说是很重要的。现在我们先看看public与private两项：

public类型的成员会被继承

private类型的成员不会被继承

当子类把成员继承下来时会把它们当作是自己定义的一样。例如说当某个形状体继承 Shape 时，就会有rotate()与playSound()这两个方法。

任一类的成员包含有自己定义出的变量和方法再加上从父类所继承下来的任何东西。

注意：在附录B和第16章中会看到更多关于default和protected的信息。

你会善用继承吗？你滥用继承了吗？

虽然下面要讨论的规则有些原因是我们现在不会先说明的，但是记住这些规则能够帮助你进行更好的继承设计，这些背后的因素会在后面的章节再加以介绍。

当某个类会比其父类更具有特定意义时使用继承。例如说美国短毛猫是一种特定品种的猫，所以从猫中扩充出美国短毛猫是很合理的。

在行为程序（实现程序代码）应该被多个相同基本类型类所共享时，应该要考虑使用继承。举例来说，方形、圆形、三角形都需要旋转和播放声音，因此将这些功能放在它们的父类上面是很合理的，并且这样也比较好维护和扩充。然而，要注意到虽然继承是面向对象程序设计的一项关键特征，但却不一定是达成重用行为程序的最佳方式。我们会教你如何运用继承，这通常也是不错的选择，但有时常用的“设计模式（design pattern）”也会提出更微妙且更有适应性的选择。如果你不太清楚设计模式是什么，在你能够掌握面向对象程序设计的概念之后，不妨看看《Head First设计模式》，这本书也有中译本。

若两者间的关系对于继承结构来说并不合理，则不要只是因为打算要重用其他类的程序代码而运用继承。例如，在设计钢琴对象时，不能因为想要借用河马对象的发声程序就让这两个八竿子打不着的对象产生继承上的关系。这完全不合理！（应该要创建出发音对象，然后让钢琴与河马都用HAS-A关系来运用此对象才对）。

如果两者间不能通过IS-A测试就不要应用继承关系。一定要确定子类是父类一种更特定的类型才可以。

要点

- 子类是extends父类出来的。
- 子类会继承父类所有public类型的实例变量和方法，但不会继承父类所有private类型的变量和方法。
- 继承下来的方法可以被覆盖掉，但实例变量不能被覆盖掉。
- 使用IS-A测试来验证继承结构的合理性。
- IS-A关系是单方向的，河马是动物，但动物不一定是河马。
- 当某个方法在子类中被覆盖过，调用这个方法时会调用到覆盖过的版本。
- 如果类Y是extends类X，且类Y是类Z的父类，则Z应该能通过IS-A X的测试。

继承到底有什么意义？

通过设计继承的过程你可以累积面向对象经验值。通过提取出一组类间共同的抽象性，你能够排除掉重复的程序代码而将这个部分放在父类中。如此一来，如果有共同的部分需要改变，就只会有一个地方要修改而已，且这样的改变会应用到所有继承此行为的类。修改之后只需要重新编译就行，不必动子类！

换上改变过的父类，则所有扩充过它的类都会自动使用到新的版本。

Java程序只是由一堆类组成的，因此，子类不需要重新编译就能运用到新版本的父类。如果父类没有破坏到子类，万事都会OK（稍后我们会讨论到何谓破坏，现在你可以先想象如果父类修改了方法的参数数目、类型或返回类型会怎么样）。

① 避免了重复的程序代码

在单一的位置定义共同程序代码，然后让子类继承父类的程序代码。当你想要改变这个行为程序时，只需修改这个地方，而子类就会发生同样的改变。

② 定义出共同的协议



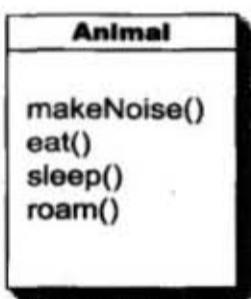
继承让你可以确保某个父型之下所有类都会有父型所持有的全部方法*

也就是说，你会通过继承来定义相关类间的共同协议

当你在父类中定义方法时，它们会被子类继承，这样你就是在对其他程序代码声明：“我所有的子类（比如说 subclass）都能用这些方法来执行这几项工作……”。

也就是说你拟出了一份“合约”。

Animal这个类拟出所有动物子型的共同协议：



你是在声明说所有的Animal都可以执行这4个动作。这也包括了method的参数和返回类型

要记得，当我们说所有的动物时，意思是Animal以及所有继承过Animal的类。也就是说在继承层次上方有Animal的任何一个类。

不过我们还没有讨论到最精采的部分——多态（polymorphism），这要留到最后。

注意下面这段说明，我们会持续在接下来的几页中解释它的意义：

当你定义出一组类的父型时，你可以用子型的任何类来填补任何需要或期待父型的位置。

这段说明很重要……

因为你会体会到多态的好处。

因为我会……

通过声明为父型类型的对象引用来引用它的子型对象。

对我来说……

这是编写出真正具有适应性的程序代码的机会。程序会变得更简洁、更有效率、更简单。程序不但容易开发而且也更容易扩展。

这样你就可以在同事更新程序的同时好好地度个假，他们甚至不需要你的源代码或打扰你就可以继续工作。

你会在下一页看到它是如何运行的。

我们其实也不认识你，但是个人相信上面这样的说法蛮吸引人的。



* “全部方法”的意思是“全部可继承的方法”。更准确的说法是所有public类型的方法，稍后我们还会对这个定义作些微小的更正。

若要观察多态是如何运行的，我们就必须先退回去看一般声明引用和创建对象的方法……

对象声明、创建与赋值的3个步骤：

1 3 2
`Dog myDog = new Dog();`

1 声明一个引用变量

`Dog myDog = new Dog();`

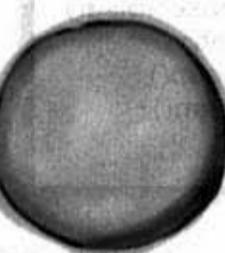
要求Java虚拟机分配空间给引用变量，并将此变量命名为myDog。此引用变量将永远被固定为Dog类型。



2 创建对象

`Dog myDog = new Dog();`

要求Java虚拟机分配堆空间给新建立的Dog对象。



3 连接对象和引用

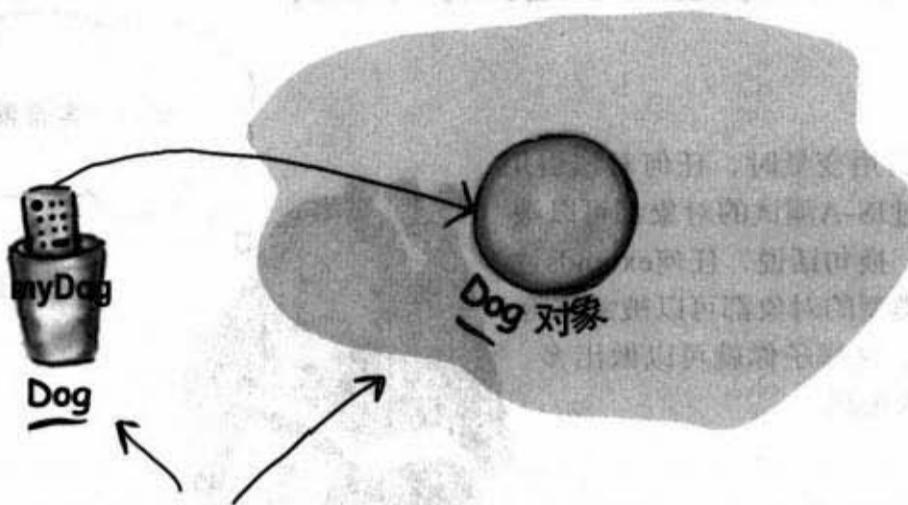
`Dog myDog = new Dog();`

将新的Dog赋值给myDog这个引用变量。换言之就是设定遥控器。



重点在于引用类型与对象的
类型必须相符。

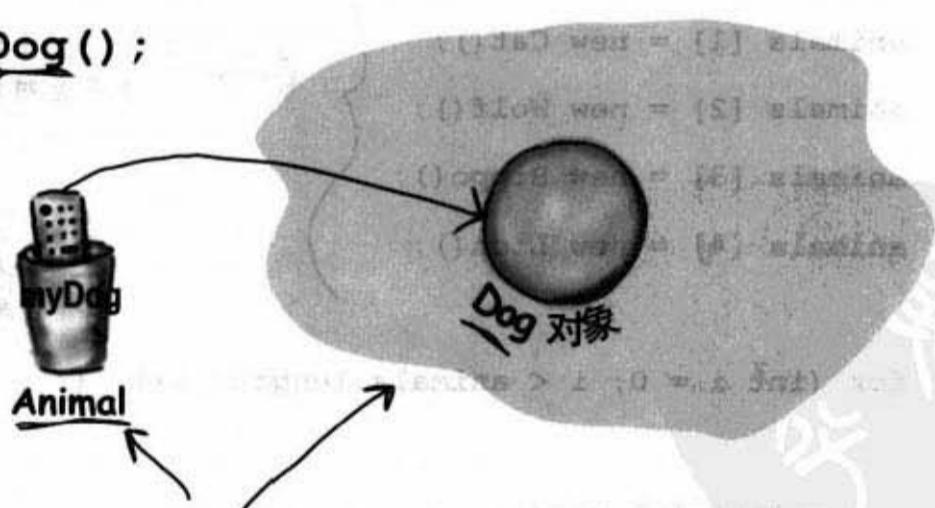
在此例中，两者皆为Dog。



这两者都是相同的类型。引用变量类型是Dog，
而对象也是Dog。

但在多态下，引用与对象可以是
不同的类型。

Animal myDog = new Dog();



这两者的类型不相同。引用变量的类型被
声明为Animal，但对象是Dog。

运用多态时，引用类型可以是实际对象类 型的父类

当你声明一个引用变量时，任何对该引用变量类型可通过IS-A测试的对象都可以被赋值给该引用。换句话说，任何extends过声明引用变量类型的对象都可以被赋值给这个引用变量。这样子你就可以做出多态数组这一类的东西。



OK，或许举个例子会比较清楚。

```
Animal[] animals = new Animal[5];
animals[0] = new Dog();
animals[1] = new Cat();
animals[2] = new Wolf();
animals[3] = new Hippo();
animals[4] = new Lion();

for (int i = 0; i < animals.length; i++) {
```

声明Animal类型的数组。也就是说一个全
保存Animal类型对象的数组

但是注意到这边……你可以放任何Animal
的子类对象进去

这就是多态最强的地方，你可以将
数组的元素逐个调出来当作是Animal

当i为0的时候，这会调用Dog的eat()

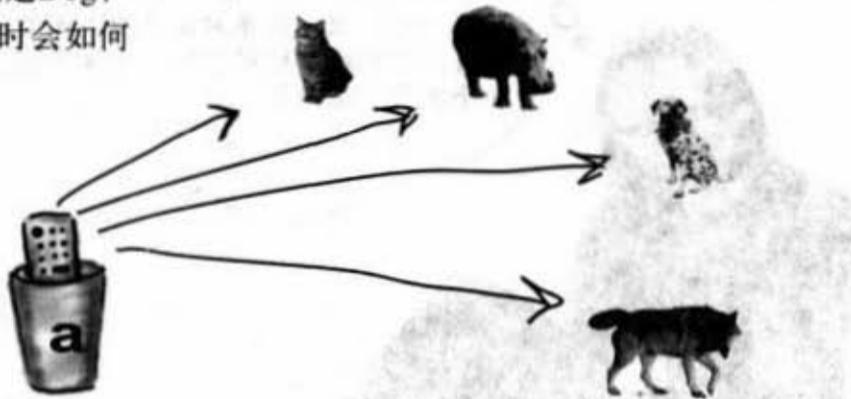
当i为1的时候，这会调用Cat的roam()

不仅如此！还有呢！

继承与多态

参数和返回类型也可以多态！

如果你声明一个父类的引用变量，比如说 Animal，并赋予类对象给它，假设是 Dog，想象一下此变量被当作方法的参数时会如何运作……



```
class Vet {  
    public void giveShot(Animal a) {  
        // do horrible things to the Animal at  
        // the other end of the 'a' parameter  
        a.makeNoise();  
    }  
}
```

* 参数可以用任何Animal的类型对象来传入。执行到makeNoise()的时候，不管它引用的对象到底是什么，该对象都会执行makeNoise()

```
class PetOwner {  
    public void start() {  
        Vet v = new Vet();  
        Dog d = new Dog();  
        Hippo h = new Hippo();  
        v.giveShot(d);  
        v.giveShot(h);  
    }  
}
```

giveShot这个方法可以取用任何一种 Animal。只要所传入的是Animal的子类它都能执行

v.giveShot(d); ← 会执行Dog的makeNoise()

v.giveShot(h); ← 会执行Hippo的makeNoise()



通过多态，你就可以编写出引进新型子类时也不必修改的程序。

还记得Vet这个类吗？如果你使用Animal类型来声明它的参数，则程序代码就可以处理所有Animal的子类。这意味着其他人只要注意到要扩充过Animal就可以利用你的Vet。



为什么多态保证这么做是没有问题的？为什么我们可以放心地假设所有的子类都会跟父类一样的方法可以通过圆点运算符调用？

there are no
Dumb Questions

问：设计子类是否有层次上的限制？最多能够设计几层？

答：如果你观察Java API，你会看到大多数的继承的层次有深度，但是不会很深。大部分不会超过一或两层。但是也有例外，特别是在GUI类这边。之后你就会了解到保持继承树层次少一点是合理的，但实际上并没有严格层数规定（至少你应该不会碰到）。

问：啊，我刚刚想到一件事情，如果你没有办法看到类的源程序代码，但又想要改变该类的方法，是否可以用子类的方式来做呢？用你自己好的代码设计不好的类并覆盖掉它们的方法？

答：可以。这是面向对象一项很了不起的特征。有时这样也可以帮你省下全部重写代码的时间。

问：你能够继承任何一个类吗？就像类的成员一样如果类是私有的你就不能继承？

答：内部类我们还没有介绍到。除了内部类之外，并没有私有类这样的概念。但是有三种方法可以防止某个类被作出子类。

第一种是存取控制。就算类不能标记为私有，但它还是可以不标记公有。非公有的类只能被同一个包的类作出子类。

第二种是使用final这个修饰符(modifier)。这表示它是继承树的末端，不能被继承。

第三种是让类只拥有private的构造程序(constructor，第9章会说明)。

问：你为什么会作出标识final的类？这样有什么好处？

答：一般来说，你不会标识出final。但如果你需要安全——确保方法都会是你写的版本，此时就需要final。

问：可不可以只用final去标识方法而不使用整个类。

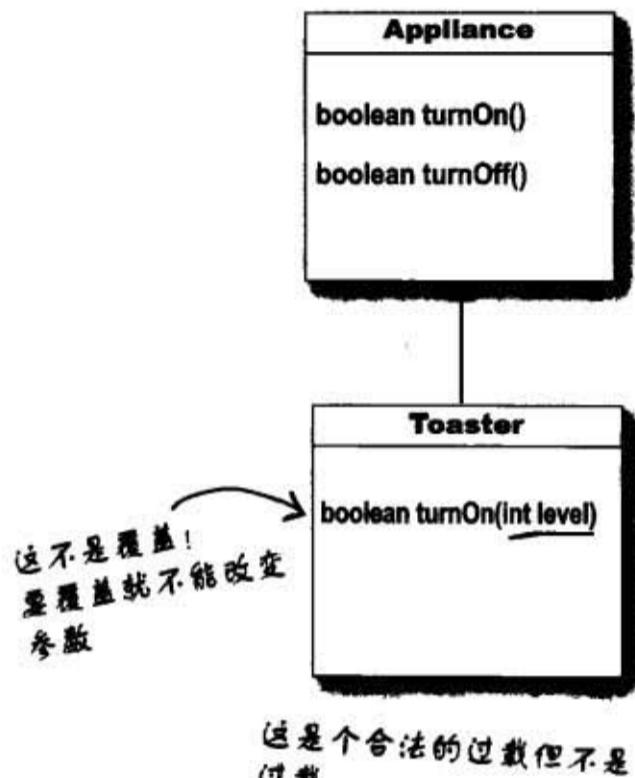
答：如果你想要防止特定的方法被覆盖，可以将该方法标识上final这个修饰符。将整个类标识成final表示没有任何的方法可以被覆盖。

遵守合约：覆盖的规则

当你要覆盖父类的方法时，你就得同意要履约。比如，这个合约表示“我没有参数且返回布尔值”。因此，你所覆盖的方法就必须没有参数且返回布尔值。

方法就是合约的标志。

如果多态运行无误的话，Toaster版覆盖Appliance的方法就会在运行期运行。记住，编译器会寻找引用类型来决定你是否可以调用该引用的特定方法。但在执行期，Java虚拟机寻找的并不是引用所指的类型，而是在堆上的对象。因此若编译器已经同意这个调用，则唯一能够通过的方法是覆盖的方法也有相同的参数和返回类型。不然的话，就算Toaster有个取用int版本的turnOn()，还是会以Appliance引用来调用没有参数的版本。到底运行期会调用哪个版本？答案是Appliance的那个版本。换句话说，在Toaster中的turnOn(int level)这个方法并没有覆盖掉Appliance的版本！



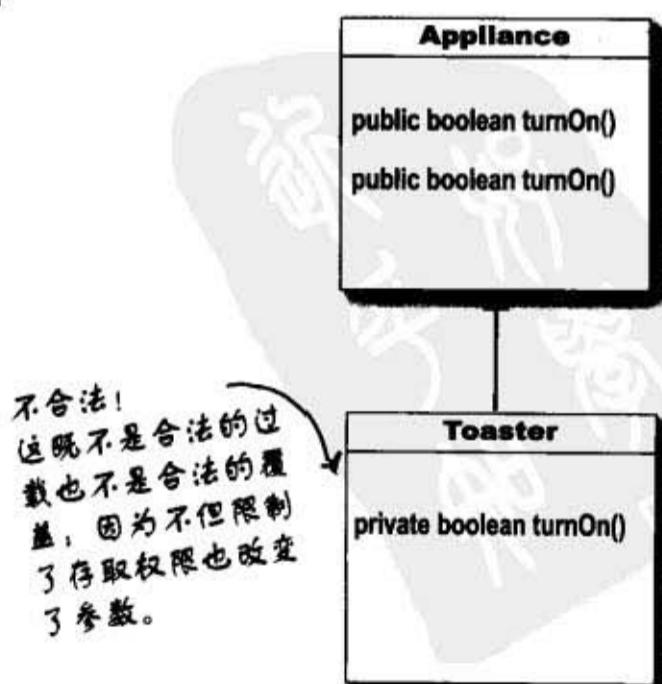
● 参数必须要一样，且返回类型必须要兼容

父类的合约定义出其他的程序代码要如何来使用方法。不管父类使用了哪种参数，覆盖此方法的子类也一定要使用相同的参数。而不论父类声明的返回类型是什么，子类必须要声明返回一样的类型或该类型的子类。要记得，子类对象得保证能够执行父类的一切。

● 不能降低方法的存取权限

这代表存取权必须相同，或者更为开放。举例来说，你不能覆盖掉一个公有的方法并将它标记为私有。这会让它以为在编译期通过的是个公有，然后突然在执行期才被Java虚拟机阻止存取。

目前为止我们看过了私有与公有这两种存取权限。另外两种会在讨论布署的章节和附录B中说明。还有关于覆盖的异常处理会在讨论异常的章节说明。



方法的重载 (overload)

重载的意义是两个方法的名称相同，但参数不同。

所以，重载与多态毫无关系。

重载可以有同一方法的多个不同参数版本以方便调用。比如，如果某个方法需要int，调用方就得将double转换成int然后才能调用。若你有个重载版本取用double参数，则这样对调用方来说就简单多了。这在讨论对象生命周期的章节中关于构造函数一节会有更多的说明。

因为重载方法不是用来满足定义在父类的多态合约，所以重载的方法比较有扩展性。

重载版的方法只是刚好有相同名字的不同方法，它与继承或多态无关。重载的方法与覆盖方法不一样。

● 返回类型可以不同

你可以任意地改变重载方法的返回类型，只要所有的覆盖使用不同的参数即可。

● 不能只改变返回类型

如果只有返回类型不同，但参数一样，这是不允许的。编译器不会让这样的事情过关。就算是重载，也要让返回类型是父类版返回类型的子类。重载的条件是要使用不同的参数，此时返回类型可以自由地定义。

● 可以更改存取权限

你可以任意地设定overload版method的存取权限。

重载的合法范例

```
public class Overloads {
    String uniqueID;

    public int addNums(int a, int b) {
        return a + b;
    }

    public double addNums(double a, double b) {
        return a + b;
    }

    public void setUniqueID(String theID) {
        // lots of validation code, and then:
        uniqueID = theID;
    }

    public void setUniqueID(int ssNumber) {
        String numString = "" + ssNumber;
        setUniqueID(numString);
    }
}
```



练习

连连看

```
a = 6; → 56
b = 5; → 11
a = 5; → 65
```

下面有一个Java小程序。其中有一段程序不见了。你的任务是找出下面所列出的程序段与相应的输出。

并非所有的输出都有可对应的程序段，且某些输出可能会被使用多次。画条线将相符的两者连接起来。

the program:

```
class A {
    int ivar = 7;
    void m1() {
        System.out.print("A's m1, ");
    }
    void m2() {
        System.out.print("A's m2, ");
    }
    void m3() {
        System.out.print("A's m3, ");
    }
}

class B extends A {
    void m1() {
        System.out.print("B's m1, ");
    }
}
```

```
class C extends B {
    void m3() {
        System.out.print("C's m3, "+(ivar + 6));
    }
}

public class Mixed2 {
    public static void main(String [] args) {
        A a = new A();
        B b = new B();
        C c = new C();
        A a2 = new C();
    }
}
```

少了3行
程序代码

程序候选码:

b.m1(); }
c.m2(); }
a.m3(); }

c.m1(); }
c.m2(); }
c.m3(); }

a.m1(); }
b.m2(); }
c.m3(); }

a2.m1(); }
a2.m2(); }
a2.m3(); }

输出:

A's m1, A's m2, C's m3, 6

B's m1, A's m2, A's m3,

A's m1, B's m2, A's m3,

B's m1, A's m2, C's m3, 13

B's m1, C's m2, A's m3,

B's m1, A's m2, C's m3, 6

A's m1, A's m2, C's m3, 13



我是编译器

哪一组A-B两段程序代码插入左边的类中可以通过编译并执行出下方的输出？（A插入类Monster中，B插入类Vampire中）

```
public class MonsterTestDrive {
    public static void main(String [] args) {
        Monster [] ma = new Monster[3];
        ma[0] = new Vampire();
        ma[1] = new Dragon();
        ma[2] = new Monster();
        for(int x = 0; x < 3; x++) {
            ma[x].frighten(x);
        }
    }
}
```

```
class Monster {
    A
}
```

```
class Vampire extends Monster {
    B
}
```

```
class Dragon extends Monster {
    boolean frighten(int degree) {
        System.out.println("breath fire");
        return true;
    }
}
```

```
File Edit Window Help SaveYourself
$ java MonsterTestDrive
a bite?
breath fire
arrrgh
```

1 boolean frighten(int d) {
 A System.out.println("arrrgh");
 return true;
 }

 boolean frighten(int x) {
 B System.out.println("a bite?");
 return false;
 }

2 boolean frighten(int x) {
 A System.out.println("arrrgh");
 return true;
 }

 int frighten(int f) {
 B System.out.println("a bite?");
 return 1;
 }

3 boolean frighten(int x) {
 A System.out.println("arrrgh");
 return false;
 }

 boolean scare(int x) {
 B System.out.println("a bite?");
 return true;
 }

4 boolean frighten(int z) {
 A System.out.println("arrrgh");
 return true;
 }

 boolean frighten(byte b) {
 B System.out.println("a bite?");
 return true;
 }



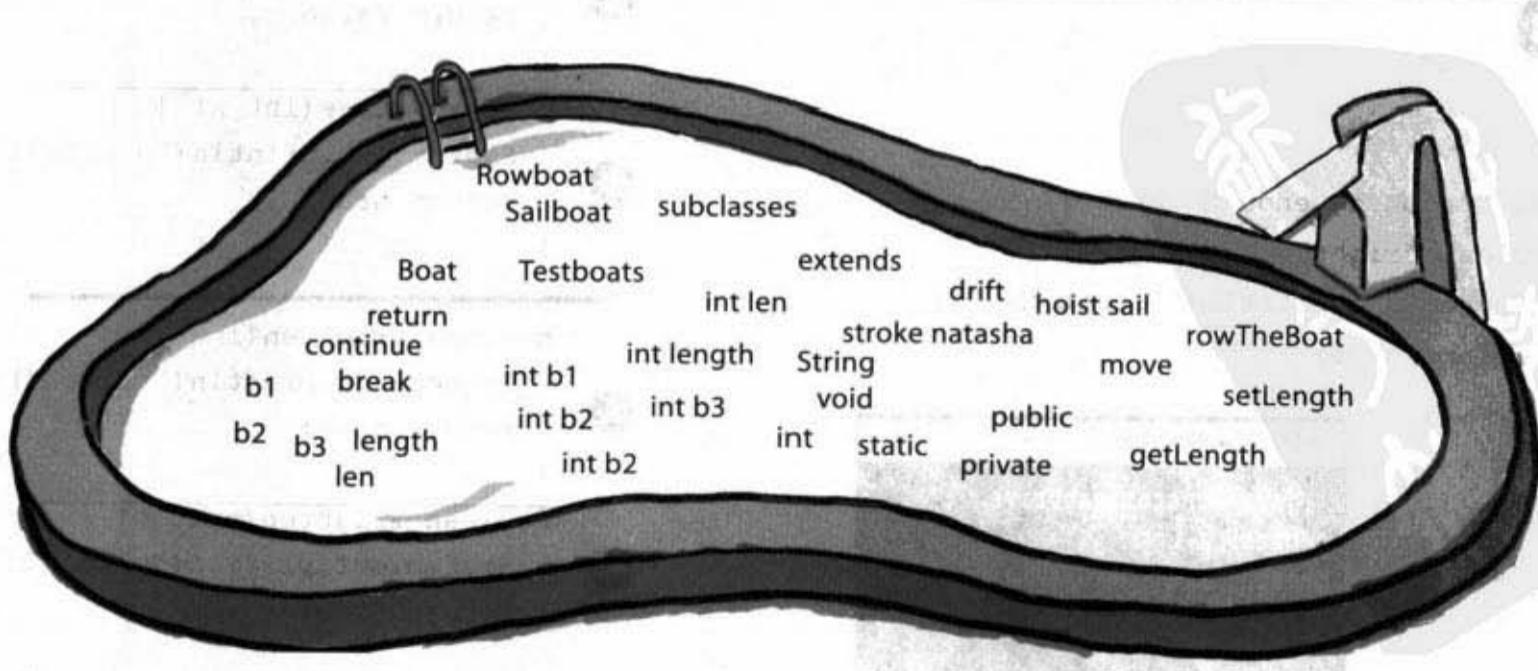
泳池迷宫

你的任务是要从游泳池中挑出程序片段并将它填入右边的空格中。同一个片段可以重复使用，且泳池中有些多余的片段。填完空格的程序必须要能够编译与执行并产生出下面的输出。

```
public class Rowboat {  
    public void rowTheBoat() {  
        System.out.print("stroke natasha");  
    }  
  
    public class {  
        private int ;  
        void ( ) {  
            length = len;  
        }  
        public int getLength() {  
            ;  
        }  
        public move() {  
            System.out.print(" ");  
        }  
    }  
}
```

```
public class TestBoats {  
    main(String[] args){  
        b1 = new Boat();  
        Sailboat b2 = new ();  
        Rowboat = new Rowboat();  
        b2.setLength(32);  
        b1.();  
        b3.();  
        .move();  
    }  
  
    public class Boat {  
        public () {  
            System.out.print(" ");  
        }  
    }  
}
```

输出： **drift drift hoist sail**





我是编译器

第一组可以。

第二组无法通过编译，因为 Vampire 返回的类型是 int。

第三组与第四组可以编译，但是会产生下面这样的输出：

arrrgh

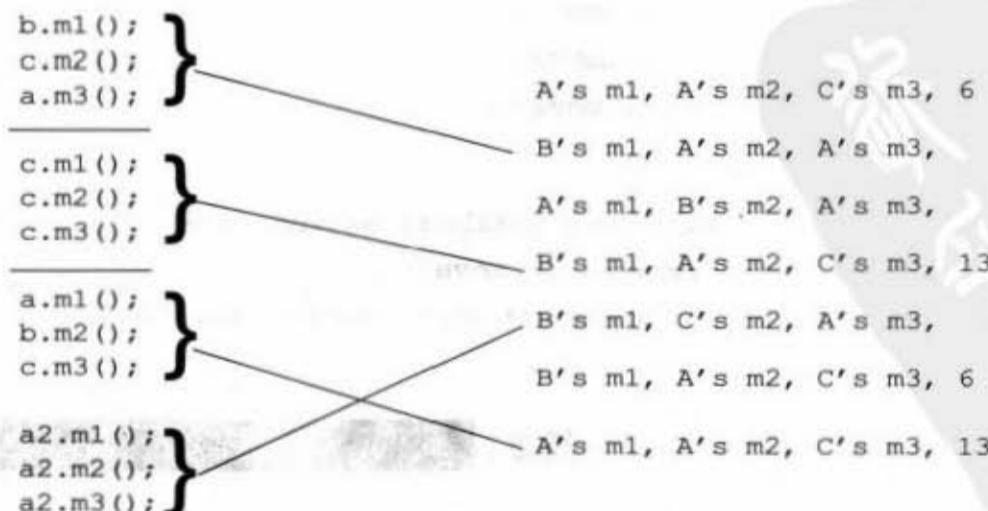
breath fire

arrrgh



解答

连连看



解答



```
public class Rowboat extends Boat {
    public void rowTheBoat() {
        System.out.print("stroke natasha");
    }
}

public class Boat {
    private int length;
    public void setLength ( int len ) {
        length = len;
    }
    public int getLength() {
        return length;
    }
    public void move() {
        System.out.print("drift ");
    }
}

public class TestBoats {
    public static void main(String[] args){
        Boat b1 = new Boat();
        Sailboat b2 = new Sailboat();
        Rowboat b3 = new Rowboat();
        b2.setLength(32);
        b1.move();
        b3.move();
        b2.move();
    }
}

public class Sailboat extends Boat {
    public void move() {
        System.out.print("hoist sail ");
    }
}
```

输出: drift drift hoist sail

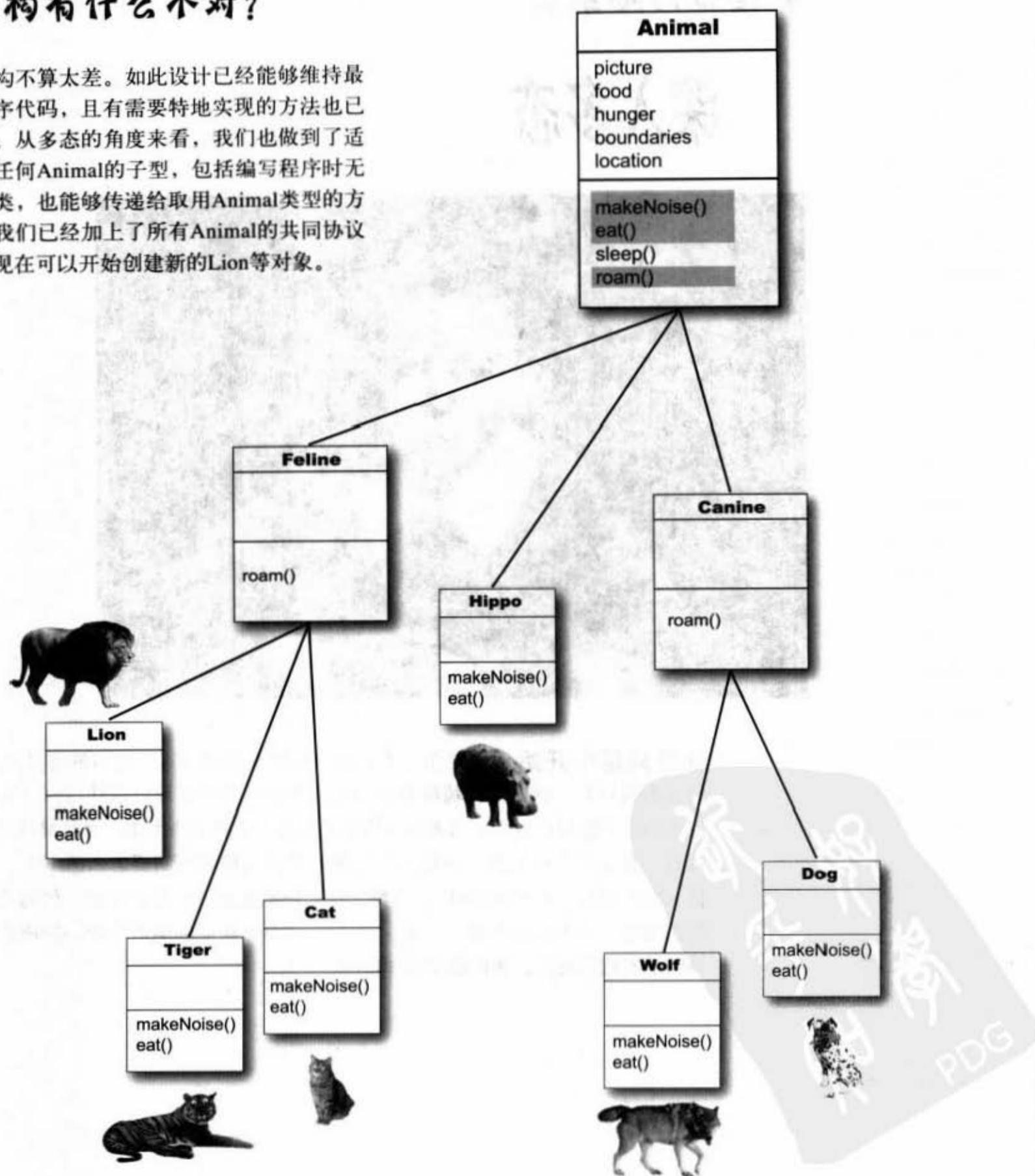
深入多态



继承只是个开始。要使用多态，我们还需要接口（这个接口的意义不是GUI的I所代表的接口）。我们需要超越简单的继承并前进到只有通过设计与编写接口规格才能达成的适应性与扩展性。很多Java功能若没有接口就无法工作，因此就算你不会去设计接口，也还是会使用到。最后你会怀疑如果没有接口机制要怎么活下去。到底接口是什么呢？它是一种100%纯抽象的类。什么是抽象类？它是无法初始化的类。抽象类有什么用途？马上就会告诉你。前一章让Vet这个方法能够运用Animal的所有子类只是多态的基本招式而已。接口是多态和Java的重点。

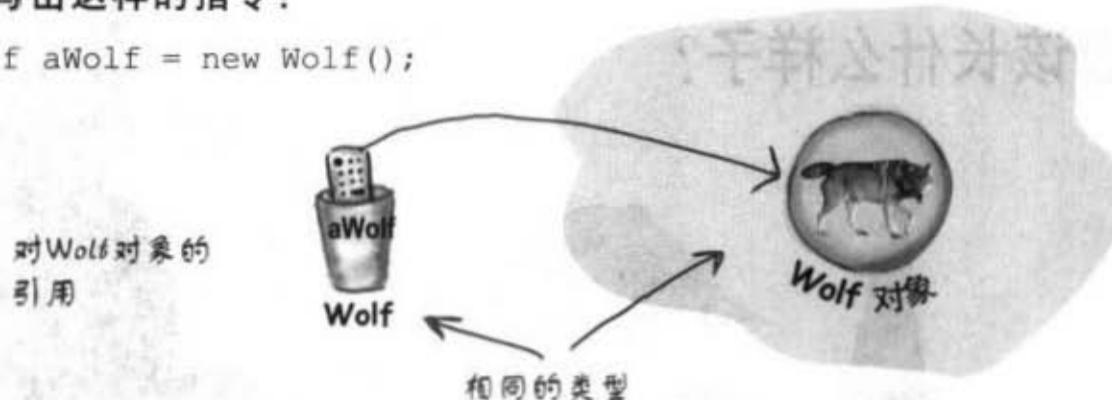
这个结构有什么不对？

这个class结构不算太差。如此设计已经能够维持最少的重复程序代码，且有需要特地实现的方法也已经被覆盖过。从多态的角度来看，我们也做到了适应性，所以任何Animal的子型，包括编写程序时无法想象的种类，也能够传递给取用Animal类型的方法来执行。我们已经加上了所有Animal的共同协议到父类上，现在可以开始创建新的Lion等对象。



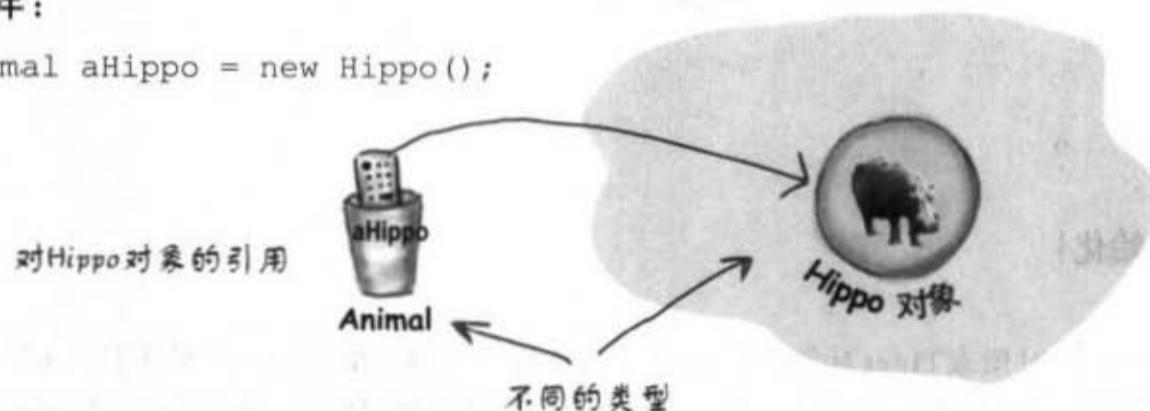
我们可以写出这样的指令：

```
Wolf aWolf = new Wolf();
```



也可以这样：

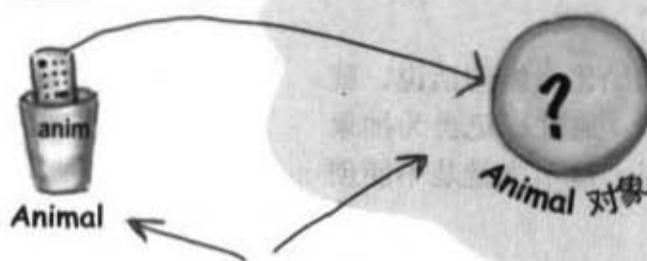
```
Animal aHippo = new Hippo();
```



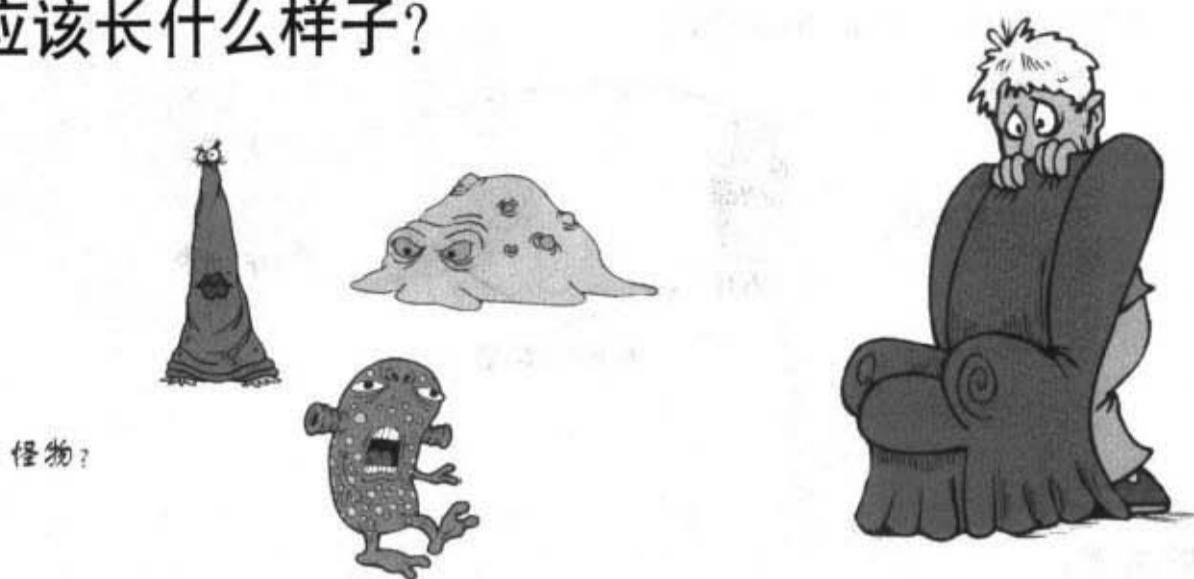
但是这样会很奇怪：

```
Animal anim = new Animal();
```

对 Animal 对象的引用



Animal对象应该长什么样子？



实例变量的值会是……？

有些类不应该被初始化！

创建出Wolf对象或Hippo对象或Tiger对象是很合理的，但是Animal对象呢？它应该长什么样子？什么颜色、大小、几条腿？

尝试创建出Animal类型的对象就好像出人意料之外，在传送组合的过程中发生了一点问题。

那要如何处理这个问题呢？我们一定要有Animal这个类来继承和产生多态。但是要限制只有它的子类才能够被初始化。我们要的是Lion、Hippo对象，而不是Animal对象。

幸好，有个方法能够防止类被初始化。换句话说，就是让这个类不能被“new”出来。通过标记类为抽象类的，编译器就知道不管在哪里，这个类就是不能创建任何类型的实例。

你还是可以用这种抽象的类型作为引用类型。这也就是当初为何要有抽象类型的目的。

当你设计好继承结构时，你必须要决定哪些类是抽象的，而哪些是具体的。具体的类是实际可以被初始化为对象的。

设计抽象的类很简单——在类的声明前面加上抽象类关键词**abstract**就好：

```
abstract class Canine extends Animal {  
    public void roam() {}  
}
```

编译器不会让你初始化抽象类

抽象类代表没有人能够创建出该类的实例。你还是可以使用抽象类来声明为引用类型给多态使用，却不用担心哪个创建该类型的对象，编译器会确保这件事。

```
abstract public class Canine extends Animal
{
    public void roam() { }
}
```

```
public class MakeCanine {
    public void go() {
        Canine c;           } // 这是可以的，因为你可以赋值子类对象给父类的引用，即使父类是抽象的
        c = new Dog();
        c = new Canine();   } // 这个类已经被标记为 abstract，所以过了编译器这一关
        c.roam();
    }
}
```

```
File Edit Window Help BeamMeUp
% javac MakeCanine.java
MakeCanine.java:5: Canine is abstract;
cannot be instantiated
    c = new Canine();
               ^
1 error
```

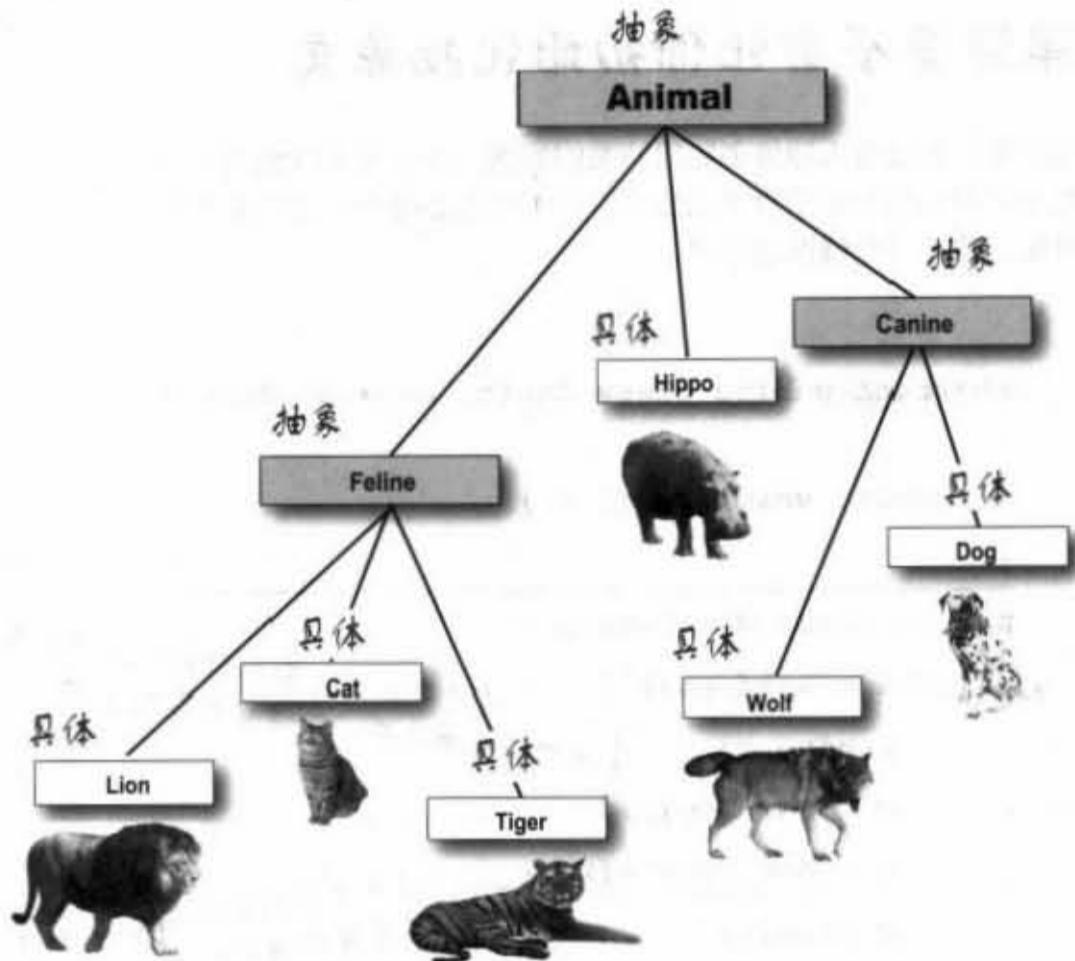
抽象类除了被继承过之外，是没有用途、没有值、没有目的*。

*除了抽象的class可以有static的成员之外，见第10章。

抽象与具体

不是抽象的类就被称为具体类。在 Animal 的继承树下，如果我们创建出 Animal、Canine 与 Feline 的抽象，则 Hippo、Wolf 等就是具体的类。

查阅 Java API 你就会发现其中有很多的抽象类，特别是 GUI 的函数库中更多。GUI 的组件类是按钮、滚动条等与 GUI 有关类的父类。你只会对组件下的具体子类作初始化动作。



抽象或具体？

你怎么知道某个类应该是抽象的？饮料或许是抽象的。那大雕参茸或威士忌是否也应该是抽象的？在继承层次中从哪个点开始才算是具体的？

你会把“提神饮料”设计成具体，还是说它也是个抽象？看起来“保力达 B”才会是具体的。你认为呢？

观察上面的 Animal 继承层次，这些抽象或具体的决定是否合适呢？你会修改这个层次吗？



抽象的方法

除了类之外，你也可以将方法标记为abstract的。抽象的类代表此类必须要被extend过，抽象的方法代表此方法一定要被覆盖过。你或许认为抽象类中的某些行为在没有特定的运行时不会有任何的意义。也就是说，没有任何通用的实现是可行的。想象一下通用的eat()方法会有什么结果？

抽象的方法没有实体！

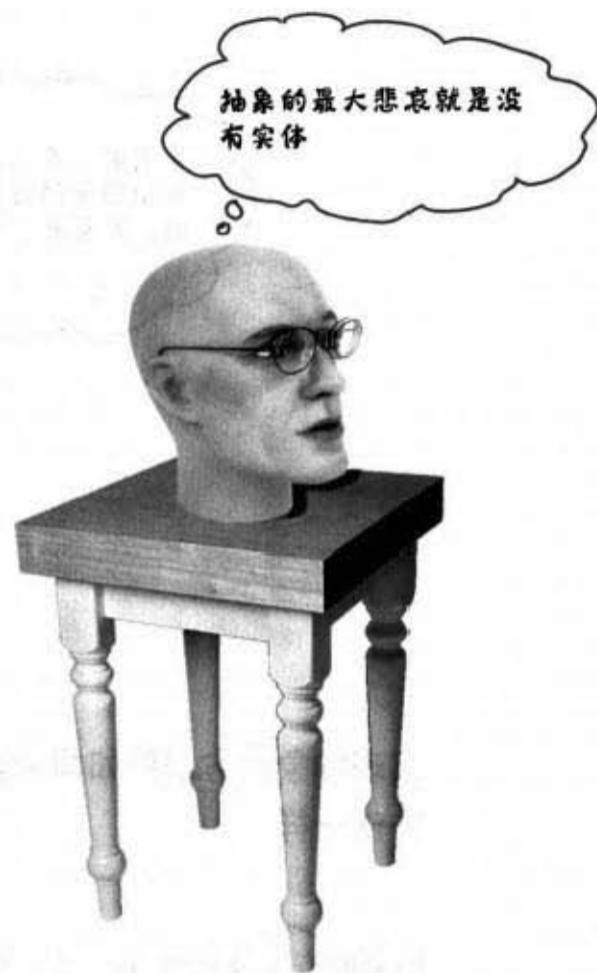
因为你已经知道编写出抽象方法的程序代码没有意义，所以不会含有方法。

```
public abstract void eat();
```

没有方法体！
直接以分号结束

如果你声明出一个抽象的方法，就必须将类也标记为抽象的。你不能在非抽象类中拥有抽象方法。

就算只有一个抽象的方法，此类也必须标记为抽象的。



there are no
Dumb Questions

问：为什么要有抽象的方法？我认为抽象类的重点就在于可以被子类继承的共同程序代码。

答：将可继承的方法体（也就是有内容的方法）放在父类中是个好主意。但有时就是没有办法作出给任何子类都有意义的共同程序代码。抽象方法的意义是就算无法实现出方法的内容，但还是可以定义出一组子型共同的协议。

问：这样做的好处是……？

答：就是多态！记住，你想达成的目标是要使用父型作为方法的参数、返回类型或数组的类型。通过这个机制，你可以加入新的子型到程序中，却又不必重写或修改处理这些类型的程序。想象一下如果不是使用Animal作为Vet的方法参数程序会写成什么样子……你必须为每一种动物写出不同的方法！因此多态的好处就在于所有子型都会有那些抽象的方法。

你必须实现所有抽象的方法



实现抽象的方法就如同覆盖过
方法一样

抽象的方法没有内容，它只是为了标记出多态而存在。这表示在继承树结构下的第一个具体类必须要实现出所有的抽象方法。

然而你还是可以通过抽象机制将实现的负担转给下层。比如说将Animal与Canine都标记为abstract，则Canine就无需实现出Animal的抽象方法。但具体的类，比如说Dog，就得实现出Animal和Canine的抽象方法。

记得抽象类可以带有抽象和非抽象的方法，因此Canine也可以实现出Animal的抽象方法，让Dog不必实现这个部分。如果Canine没有对Animal的抽象类表示出任何意见，就表示Dog得自己实现出Animal的抽象方法。

当我们谈到“你必须实现所有抽象的方法”时，表示说你必须写出内容。你必须以相同的方法签名（名称与参数）和相容的返回类型创建出非抽象的方法。Java很注重你的具体子类有没有实现这些方法。



抽象类与具体类

让我们来对抽象修辞学产生些具体的用途。在中间这行列出一些类。你的任务是想象有哪些应用程序会把它们设计成具体的，又有哪些应用程序会把它们设计成抽象的。我们已经举出几个例子，例如Tree这个类在植物看护应用程序中应该是抽象的，而在高尔夫球场仿真程序中应该会是具体的。

具体	类	抽象
高尔夫模拟	Tree	植物看护
	House	建筑设计程序
卫星影像程序	Town	
	Football Player	教练程序
	Chair	
	Customer	
	Sales Order	
	Book	
	Store	
	Supplier	
	Golf Club	
	Carburetor	
	Oven	

多态的使用

假设我们不知道有ArrayList这种类而想要自行编写维护list的类以保存Dog对象。在第一轮我们只会写出add()方法。我们使用大小为5的简单Dog数组(Dog[])来保存新加入的Dog对象。当Dog对象超过5个时，你还是可以调用add()方法，但是什么事情也不会发生。如果没有越界，add()会把Dog装到可用的数组位置中，然后递增可用索引(nextIndex)。

自己创建的Dog专用list

(这或许是有史以来自行尝试编写ArrayList类型类中最差劲的一个程序)

version
1

```
public class MyDogList {  
    private Dog [] dogs = new Dog[5];  
    private int nextIndex = 0; ← 增加新的Dog就加1  
  
    public void add(Dog d) {  
        if (nextIndex < dogs.length) { } ← 如果没有超出上限就把Dog加进去并列出信息  
            dogs[nextIndex] = d;  
            System.out.println("Dog added at " + nextIndex);  
            nextIndex++; ← 递增计数  
    }  
}
```

糟了，也要写出Cat用的

我们有几个选项：

- (1) 另外创建一个单独的MyCatList类来处理Cat对象，这不好。
- (2) 创建一个单独的DogAndCatList类，用 addCat(Cat c) 与 addDog(Dog d) 来同时处理两个不同的数组实例，这也不好。
- (3) 编写一个不同的AnimalList类让它处理Animal所有的子类。这应该是最好的办法，所以我们就这么处理，以更通用的Animal来取代个别的子类。程序变更得关键部分有特别标出来（逻辑还是一样，只是把Dog换成Animal）。

自己创建的Animal通用list

```

public class MyAnimalList {
    private Animal[] animals = new Animal[5];
    private int nextIndex = 0;
}

public void add(Animal a) {
    if (nextIndex < animals.length) {
        animals[nextIndex] = a;
        System.out.println("Animal added at " + nextIndex);
        nextIndex++;
    }
}

```

别紧张，这不是在创建Animal对象，只是个保存Animal的数组对

MyAnimalList

Animal[] animals
int nextIndex
add(Animal a)

```

public class AnimalTestDrive{
    public static void main (String[] args) {
        MyAnimalList list = new MyAnimalList();
        Dog a = new Dog();
        Cat c = new Cat();
        list.add(a);
        list.add(c);
    }
}

```

```

File Edit Window Help Harm
% java AnimalTestDrive
Animal added at 0
Animal added at 1

```

非Animal呢？何不写个万用类？

你知道这要怎么做。我们可以修改数组的类型，并且调整add()方法的参数，以处理Animal之上的类。那便是更通用、更抽象的一种类。但是真的有这种类吗？我们设计的Animal并没有父类不是吗？

事实上是有的。

还记得ArrayList的方法吗？它们是通过对象这个类型来操纵所有类型的对象。

在Java中的所有类都是从Object这个类继承出来的。

Object这个类是所有类的源头；它是所有类的父类。

如果Java中没有共同的父类，那将无法让Java的开发人员创建出可以处理自定义类型的类，也就是说无法写出像ArrayList这样可以处理各种类的类。

就算你不知道，但实际上所有的类都是从对象给继承出来的。你可以把自己写的类想象成是这样声明的：

```
public class Dog extends Object { }
```

但是Dog本来是从Canine给extends出来的啊！没关系，编译器会知道改成让Canine去继承对象。事实上是Animal去继承对象。

没有直接继承过其他类的类会是隐含的继承对象。

所以就算Dog或Canine没有直接extend对象，还是会通过Animal来继承对象。

version
3

(这只是部分的ArrayList中的方法)

ArrayList

boolean remove(Object elem)

根据索引参数移动对象，如果list中没有元素返回true

boolean contains(Object elem)

如果和对象的参数相匹配的话返回true

boolean isEmpty()

如果list中没有元素返回true

int indexOf(Object elem)

返回对象参数的索引值或-1

Object get(int index)

返回元素在list中的位置

boolean add(Object elem)

向list中增加元素

// more

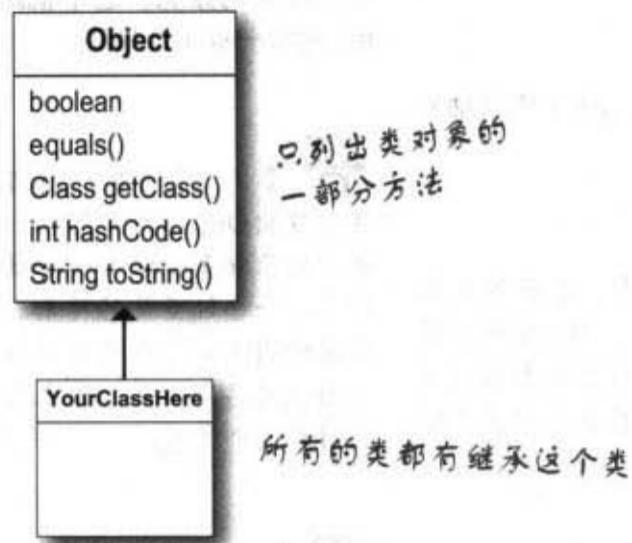
许多ArrayList的方法都用到Object这个终极类型。因为每个类都是对象的子类，所以ArrayList可以处理任何类！

注意：Java 5.0的get()和add()方法与这里所显示的有所不同，但无损于此处要表达的概念。

终极对象有什么？

如果你是Java，那你会想要让每个对象都带有些什么行为？嗯……来个可以判断某对象是否与其他对象相等的方法如何？再加上一个可以说明它是什么类的方法怎样？或许还会需要一个产生对象哈希代码的方法？你可以运用哈希表上的对象（我们会在第17章和附录B中讨论哈希表）。

你知道怎样吗？对象的确有上面所说的方法。那还不是全部的方法，但目前我们只关心这几个。



① equals(Object o)

```
Dog a = new Dog();
Cat c = new Cat();

if (a.equals(c)) {
    System.out.println("true");
} else {
    System.out.println("false");
}
```

```
File Edit Window Help Stop
% java TestObject
false
```

这会让你知道是否两个对象可以认为是“相等”的（见附录B）

③ hashCode()

```
Cat c = new Cat();
System.out.println(c.hashCode());
```

```
File Edit Window Help Drop
% java TestObject
8202111
```

列出此对象的哈希代码，你可以把它想成是一个唯一的ID

② getClass()

```
Cat c = new Cat();
System.out.println(c.getClass());
```

```
File Edit Window Help Faint
% java TestObject
class Cat
```

告诉你此对象是从哪里被初始化的

④ toString()

```
Cat c = new Cat();
System.out.println(c.toString());
```

```
File Edit Window Help LapseIntoComa
% java TestObject
Cat@7d277f
```

列出类的名称和一个我们不关心的数字

there are no
Dumb Questions

问： Object这个类是抽象的吗？

答： 不是。至少不是正式的Java抽象类。因为它可以被所有类继承下来的方法都实现程序代码，所以没有必要被覆盖过的方法。

问： 那是否可以覆盖过Object的方法？

答： 部分可以。但是有些被标记为final，这代表你不能覆盖掉它们。强烈建议你用自己写的类去覆盖掉hashCode()、equals()以及toString()。

问： 如果ArrayList方法是通用的，那ArrayList <DotCom>是什么意思？

答： 限制它的类型。在Java 5.0之前无法限制它的类型。如果你写成ArrayList<Dog>，则此ArrayList受限只能保存Dog的对象。这种新型的语法会在后面的章节有更多的说明。

问： Object类是具体的。怎么会允许有人去创建Object的对象呢？这不就跟Animal对象一样不合理吗？

答： 好问题！为何要允许创建出Object的实例呢？因为有时候你就是会需要一个通用的对象，一个轻量化的对象。它最常见的用途是用在线程的同步化上面（见第15章）。你先当作不会用到这个对象。

问： 所以Object的主要目的是提供多态的参数与返回类型吗？

答： 这个Object类有两个主要的目的：作为多态让方法可以应付多种类型的机制，以及提供Java在执行期对任何对象都有需要的方法的实现程序代码（让所有的类都会继承到）。有一部分的方法是与线程有关，这会在后面的章节说明。

问： 即然多态类型这么有用，为什么不把所有的参数和返回类型都设定成Object类型哪？

答： 啊……想想看这会发生在什么后果。考虑一下何谓“类型安全检查”。它是Java保护程序代码的一项重要机制。在此机制下，你不会意外地要求对象执行错误类型的动作。例如说防止你对Cefiro要求Ferrai的加速动作。

但事实上，你也不用担心会发生这件事，因为当某个对象是以Object类型来引用时，Java会把它当作Object类型的实例。这代表你只能调用由Object类中所声明的方法。若你像下面这样做：

```
Object o = new Ferrai();
o.goFast(); //非法
```

则第二行会无法通过编译。

因为Java是类型检查很强的程序语言，编译器会检查你调用的是否是该对象确实可以响应的方法。换句话说，你只能从确实有该方法的类去调用。同样，这也会在后面的章节说明。

使用 Object 类型的多态引用是会付出代价的……

在你开始以 Object 类型使用所有超适用性参数和返回类型之前，你应该要考虑到使用 Object 类型作为引用的一些问题。注意此处的讨论不涉及制作出 Object 类型的实例，这是在说以 Object 类型作为引用的其他类型。

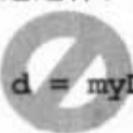
当你把对象装进 ArrayList<Dog> 时，它会被当作 Dog 来输入与输出：

```
ArrayList<Dog> myDogArrayList = new ArrayList<Dog>(); ← 保存 Dog 的 ArrayList
Dog aDog = new Dog(); ← 新建一个 Dog
myDogArrayList.add(aDog); ← 装到 ArrayList 中
Dog d = myDogArrayList.get(0); ← 将 Dog 赋值给新的 Dog 引用变量
```

但若你把它声明成 ArrayList<Object> 时会怎样？如果你打算创建出一个可以保存任何一种 对象的 ArrayList 时，你会如此的声明：

```
ArrayList<Object> myDogArrayList = new ArrayList<Object>(); ← 保存对象的 ArrayList
Dog aDog = new Dog(); ← 新建一个 Dog
myDogArrayList.add(aDog); ← 装到 ArrayList 中
```

如果是这样，当你尝试要把 Dog 对象取出并赋值给 Dog 的引用时会发生什么事？

 **Dog d = myDogArrayList.get(0);** 无法通过编译！对 ArrayList<Object> 调用 get() 方法会返回 Object 类型，编译器无法确认它是 Dog！

任何从 ArrayList<Object> 取出的东西都会被当作 Object 类型的引用而不管它原来是什么。

放进去的对象原来
是足球、鱼、
吉他和汽车



出来后都变成
Object

从 ArrayList<Object> 取出的 Object 都会被当作是 Object 这个类的实例。编译器无法将此对象识别为 Object 以外的事物。

失去狗性

当Dog不再是Dog时

问题在于把所有东西都以多态来当作是Object会让对象看起来失去了真正的本质（但不是永久性的）。Dog似乎失去了犬性。让我们来看一下当我们传入一个Dog给会返回同一个Dog对象的类型引用的方法时会有什么反应。



BAD



```
public void go() {  
    Dog aDog = new Dog();  
    Dog sameDog = getObject(aDog); // ←  
}  
  
public Object getObject(Object o) {  
    return o;  
} // ↑ 返回同一个引用，但是类型已经转换成Object了
```

无法过关！虽然这个方法会返回同一个Dog，但编译器认为这只能赋值给Object类型的变量

GOOD

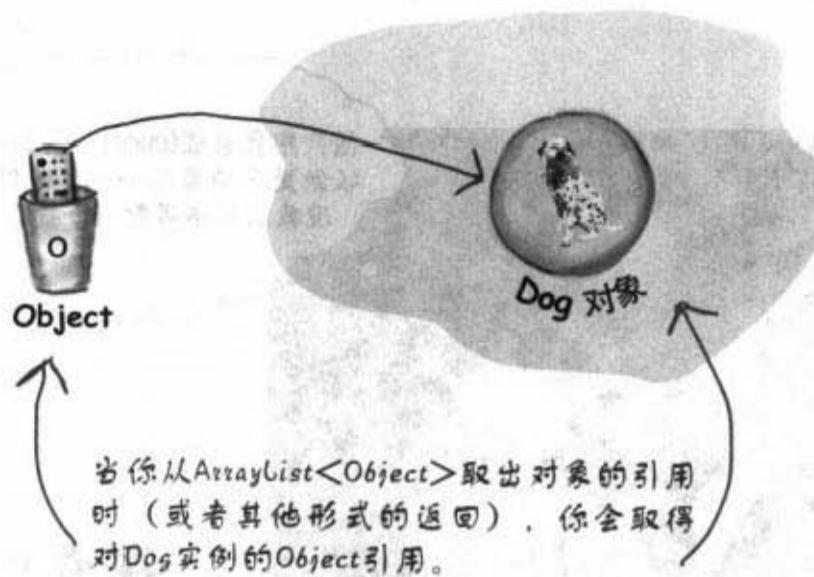


```
public void go() {  
    Dog aDog = new Dog();  
    Object sameDog = getObject(aDog); // ←  
}  
  
public Object getObject(Object o) {  
    return o;  
}
```

这样会过关，因为你可以赋值任何东西给Object类型的引用，并且每个东西都能对Object通过JS-A测试。

Object不会吠

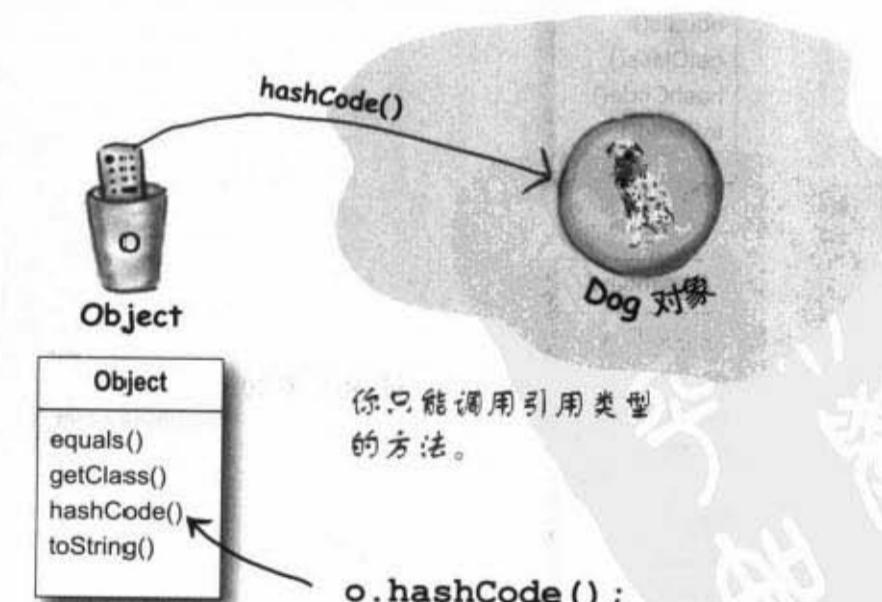
我们已经知道当一个对象被声明为Object类型的对象所引用时，它无法再赋值给原来类型的变量。我们也知道这会发生在返回类型被声明为Object类型的时候，例如前面所提过的ArrayList<Object>。但这意味着什么呢？使用Object引用变量来引用Dog对象会是个问题吗？让我们试着对被编译器认为是Object的Dog调用Dog才有的方法看看：



```
Object o = al.get(index);    没问题。Object本来就有  
int i = o.hashCode();      hashCode()这个方法  
  
有问题! → o.bark(); ← 不能这么做！Object根本就不知道  
                           什么是bark()。就算天知地知你知  
                           我知但编译器就是不知……
```

编译器是根据引用类型来判断有哪些method可以调用，而不是根据Object确实的类型。

就算你知道对象有这个功能，编译器还是会把它当作一般的Object来看待。编译器只管引用的类型，而不是对象的类型。

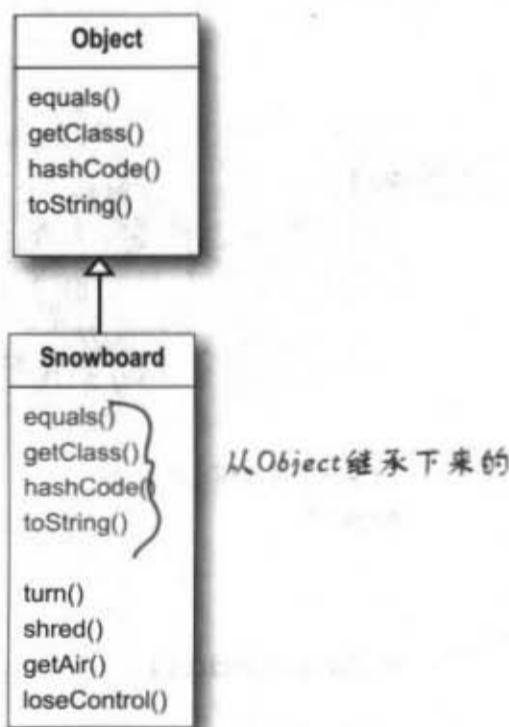


“o”被声明为Object的引用。
所以只能通过o调用Object类中的方法。



探索内部Object

对象会带有从父类继承下来的所有东西。这代表每个对象，不论实际类型，也会是个Object的实例。所以Java中的每个对象除了真正的类型外，也可以当作是Object来处理。当你执行new Snowboard()命令时，除了在堆上会有一个Snowboard对象外，此对象也包含了一个Object在里面。



“多态”意味着“很多形式”

你可以把Snowboard当作Snowboard或者Object

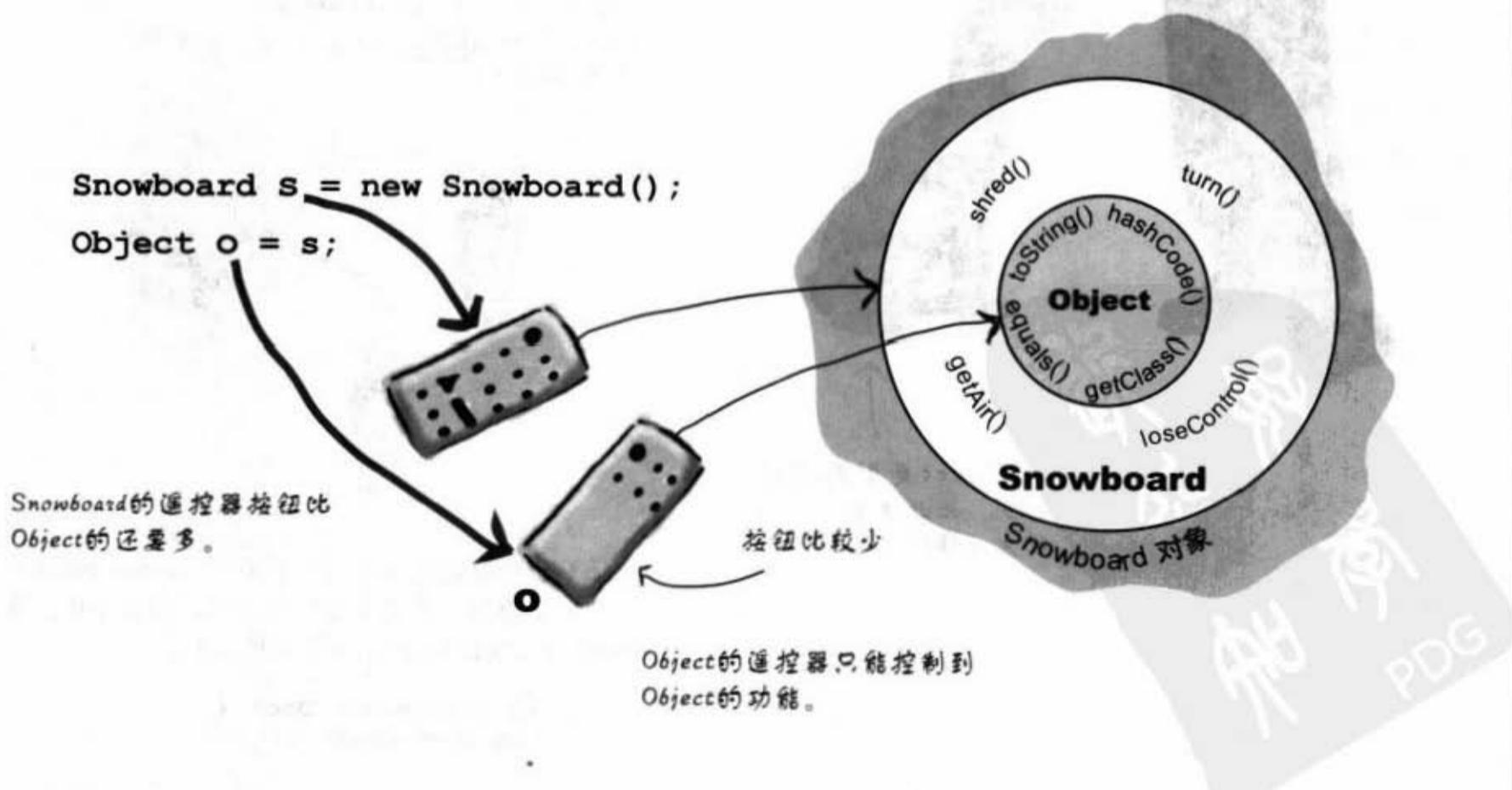
如果引用是个遥控器，则当你在继承树往下走时，会发现遥控器的按钮越来越多。Object的遥控器只有几个按钮而已，但Snowboard的遥控器就会包含有来自Object和自己定义的按钮。越接近具体的类会有越多的按钮。

当然这也不是绝对的，子类也有可能不会加入任何新的方法，而只是覆盖过一些方法罢了。重点在于如果对象的类型是Snowboard，而引用它的却是Object，则它不能调用Snowboard的方法。

当你把对象装进ArrayList<Object>时，不管它原来是什么，你只能把它当作是Object。

从ArrayList<Object>取出引用时，引用的类型只会是Object。

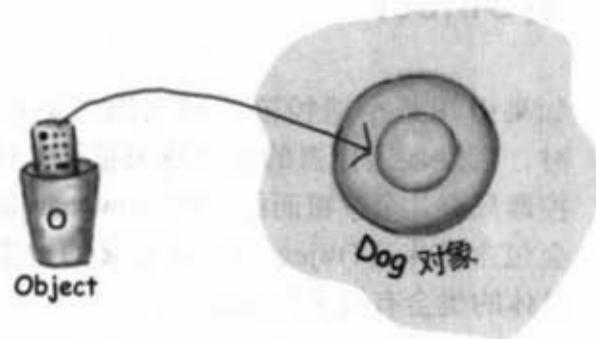
这代表你只会取得Object的遥控器。





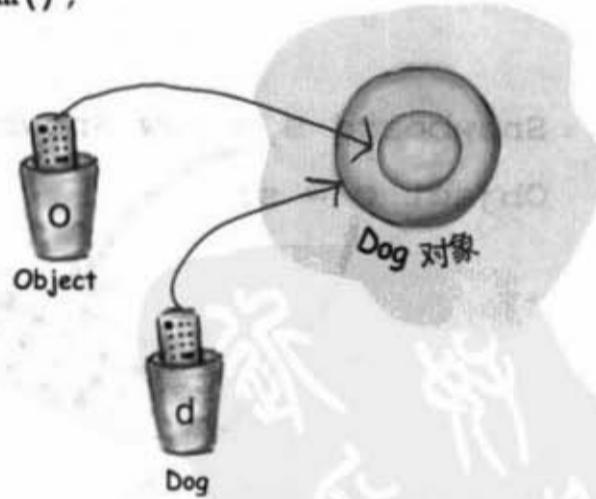
↑
将 Object 类型转换成 Dog，如此才能用 Dog 的方法操作

转换回原来的类型



它还是个Dog对象，但如果你想要调用Dog特有的方法，就必须要将类型声明为Dog。如果你真的确定它是个Dog，那么你就可以从Object中拷贝出一个Dog引用，并且赋值给Dog引用变量。

```
Object o = al.get(index);  
Dog d = (Dog) o; ← 将类型转换成 Dog  
d.roam();
```



如果不能确定它是Dog，你可以使用 instanceof 这个运算符来检查。若是类型转换错了，你会在执行期遇到 ClassCastException 异常并且终止。

```
if (o instanceof Dog) {  
    Dog d = (Dog) o;  
}
```

现在你知道Java是多么注重引用变量的类型。

你只能在引用变量的类确实有该方法才能调用它。

把类的公有方法当作是合约的内容，合约是你对其他程序的承诺协议。



在编写类时，大多数情况下你一定会显露出一些方法给类以外的程序使用。要让方法显露就代表你会让方法能够存取得到，通常这会通过标记成公有来完成。

想象这样的情境：你在编写一个小型会计总账系统给“猪标服装社”使用。你发现有个称为Account的类已经写好并且符合你的需求（上一次帮“宏昌水电行”开发程序时写出来的），因此就把它拿过来用。

它有一个debit()和credit()方法可以用来执行会计的借贷项目，还有getBalance()方法可以计算账户。

所以你可以声明一个变量a引用到Account的实例，然后通过圆点运算符调用a.debit()或a.credit()等。

因为类的合约是这么保证的，所以你在这个类的实例上面一定能找到这些方法。

Account
debit(double amt)
credit(double amt)
double getBalance()

万一想要修改合约呢？

好吧，假如你是个Dog（但是千万别去闻另外一个Dog的屁股，这样很不礼貌），你会发现不只有一份合约，你还继承了所有父类传递下来的方法。

类Canine中的所有元素是你合约中的一部分。

类Animal中的所有元素是你合约中的一部分。

类Object中的所有元素是你合约中的一部分。

根据IS-A测试，你就会是Canine、Animal和Object。

如果有人要编写类似的程序，你大可把定义好的class交给他使用。

但是，如果他还要加上亲热或耍宝等宠物特有的功能要怎么办呢？

现在假设你是设计Dog类的程序设计师。没问题吧？你可以直接把beFriendly()和play()这两个方法加进Dog这个类中。这样做不会让其他用到Dog的程序产生问题，因为你没有更改到其他现有的方法。

你觉得这样的做法（把Pet的方法直接加到Dog上）有没有缺点？



如果你是Dog类的程序设计师，且必须修改Dog类以让它能够执行Pet的动作，那你会怎么办？我们知道直接加入Pet的方法是可行的，并且这也不会对其他程序有影响。

但若Cat也要有Pet的功能怎么办？先不管Java的功能，想象一下你要怎样让Animal可以选择性地带有Pet的行为又不会强迫让狮子老虎都表现成宠物？

停下来想想看，动点脑筋会帮助你消耗能量。

有哪些方法可以在PetShop程序中重用现有的类？

在接下来的几页中，我们会对每种可能的方法逐个介绍。先不用管Java实际的功能是怎么做的。一旦知道各种方法的好处与坏处之后，我们就会知道怎么办了。

① 方法一

采用最简单的做法，把宠物方法加进Animal类中。

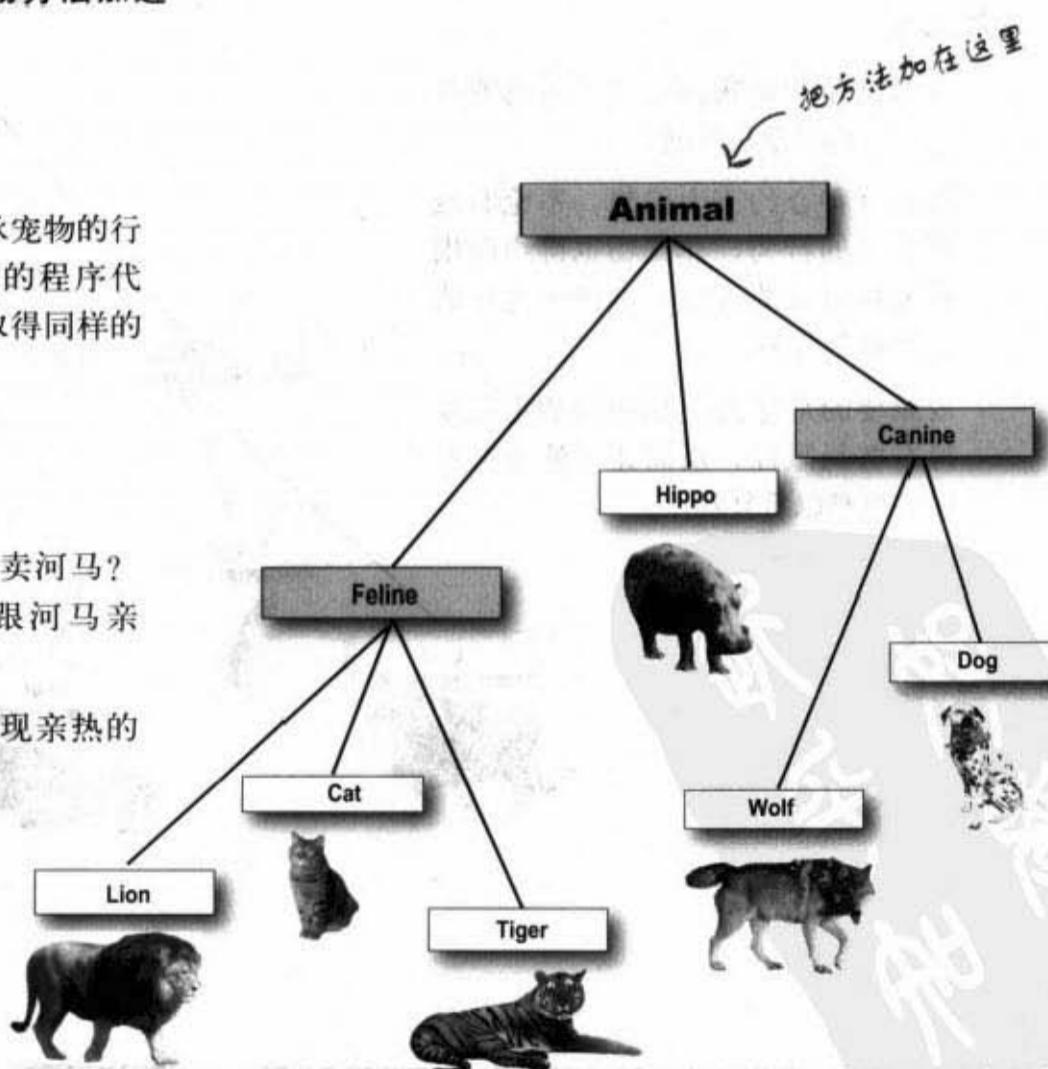
优点：

所有的动物马上就可以继承宠物的行为。不需要改变所有子类的程序代码，而新增加的动物也会取得同样的行为。

缺点：

你什么时候看过宠物店贩卖河马？又是什么时候看到饲主跟河马亲热，带河马去公园散步？

并且我们也知道狗跟猫表现亲热的方式不太一样啊。



② 方法二

采用方法一，但是把宠物的方法设定成抽象的，强迫每个动物子类覆盖它们。

优点：

这样就可以让非宠物类的动物在覆盖这些方法时，作出合理的动作，或者是什么也不作。

缺点：

所有具体的动物都得实现宠物的行为，这样很浪费时间。

并且这种合约不太理想，不论有没有实质的行为，非宠物也得声明出有宠物行为的外观，对狮子老虎的尊严是个打击。

最重要的是这会让Animal的定义变得有些局限性，反而让其他类型的程序更难以重复利用。



③ 方法三

把方法加到需要的地方。

优点：

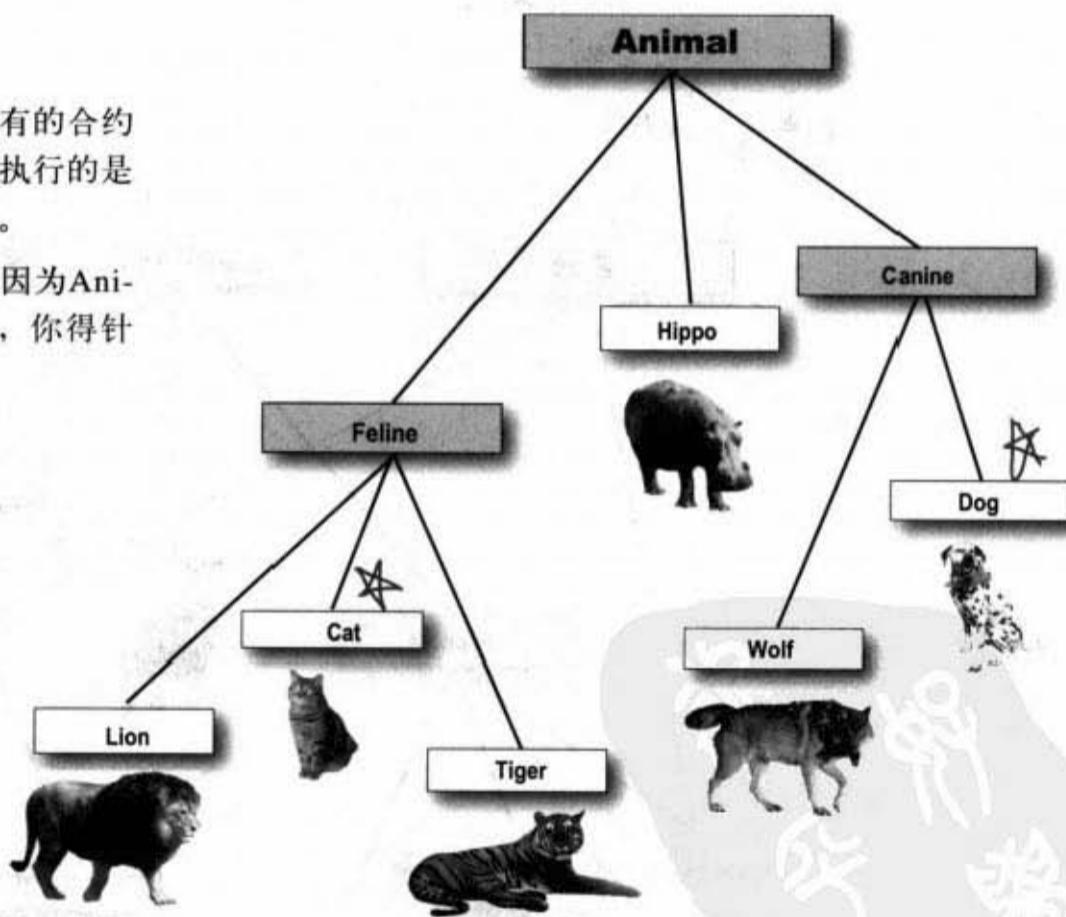
不必担心如何跟河马亲热。只有宠物才会有宠物的行为。

~~把方法写在各个类中~~

缺点：

首先，这样就会失去了物该有的合约保证。你无法确定宠物可以执行的是doFriendly()还是beFriendly()。

其次，多态将无法起作用，因为Animal不会有共同的宠物行为，你得针对个别宠物设计程序。



多重继承？

所以我们真正需要的方法是：

- (1) 一种可以让宠物行为只应用在宠物身上的方法。
- (2) 一种确保所有宠物的类都有相同的方法定义的方法。
- (3) 一种是可以运用到多态的方法。

看起来，我们需要两个上层的父类



Cat可以同时从Animal
和Pet继承

非宠物就不需继承下任
何Pet的方法

“两个父类”这个主意有一个问题……

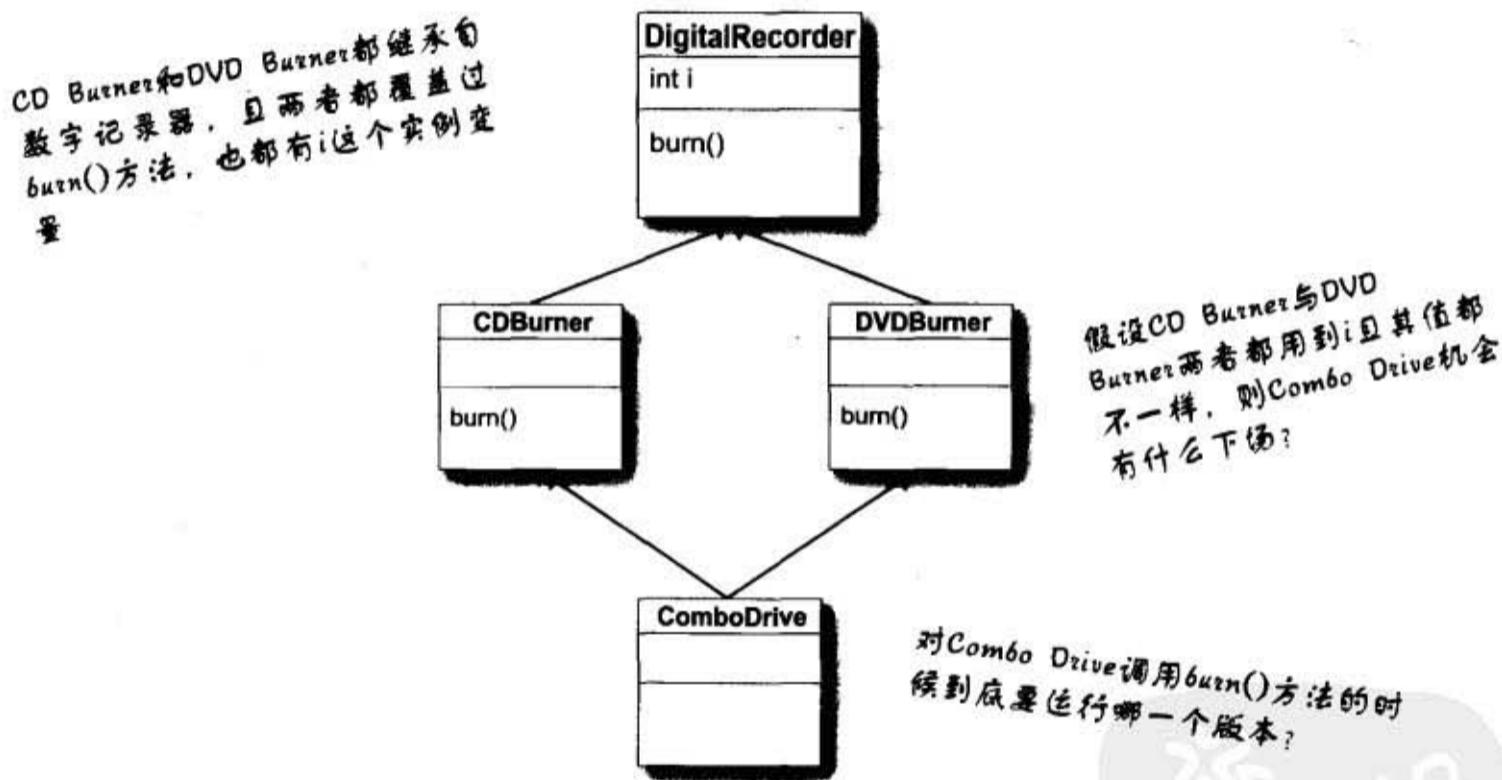
这种“多重继承”可能会很差。

其实Java不支持这种方式，

因为多重继承会有称为“致命方块”的问题。

“致命方块”

(因为这个形状看起来就像扑克牌的方块)



允许致命方块的程序语言会产生某种很糟糕的复杂性问题，因为你必须要有某种规则来处理可能出现的模糊性。额外的规则意味着你必须同时学习这些规则与观察适用这些规则的特殊状况。因此 Java 基于简单化的原则而不允许这种致命方块的出现。好吧，问题还是没有解决……

接口是我们的救星！

Java有个解决方案，使用接口。此处所讨论的不是GUI的接口，也不是“沟通管道”或“存取途径”的接口，我们说的是Java的interface关键词。

此接口可以用来解决多重继承的问题却又不会产生致命方块这种问题。

接口解决致命方块的办法很简单：把全部的方法设为抽象的！如此一来，子类就得要实现此方法，因此Java虚拟机在执行期间就不会搞不清楚要用哪一个继承版本。



Java的接口就好像是100%的纯抽象类。

所有接口的方法都是抽象的，所以任何Pet的类都必须重实现这些方法

接口的定义：

```
public interface Pet {...}
```

↑
使用“interface”取代“class”

接口的实现：

```
public class Dog extends Canine implements Pet {...}
```

↑
使用implements这个关键词。注意
到实现interface时还是必须在某个
类的继承之下

设计与实现 Pet 接口

W/ interface 取代 class

```

public interface Pet {
    public abstract void beFriendly();
    public abstract void play();
}



---


Dog IS-A Animal
Dog IS-A Pet

public class Dog extends Canine implements Pet {
    public void beFriendly() {...}
    public void play() {...}
    public void roam() {...}
    public void eat() {...}
}

```

接口方法带有 public 和 abstract 的意义，这两个修饰符是属于选择性的（我们是为了强调才把它们打出来的，实际上不需要）

接口的方法一定是抽象的，所以必须以分号结束。记住，它们没有内容！

implements 关键词后面跟着接口的名称

必须在这里实现出 Pet 的方法，这是合约的规定

一般的覆盖方法

there are no Dumb Questions

问：等一下！接口并不是真正的多重继承，因为你无法在它里面实现程序代码，不是吗？如果是这样，那还要接口做什么？

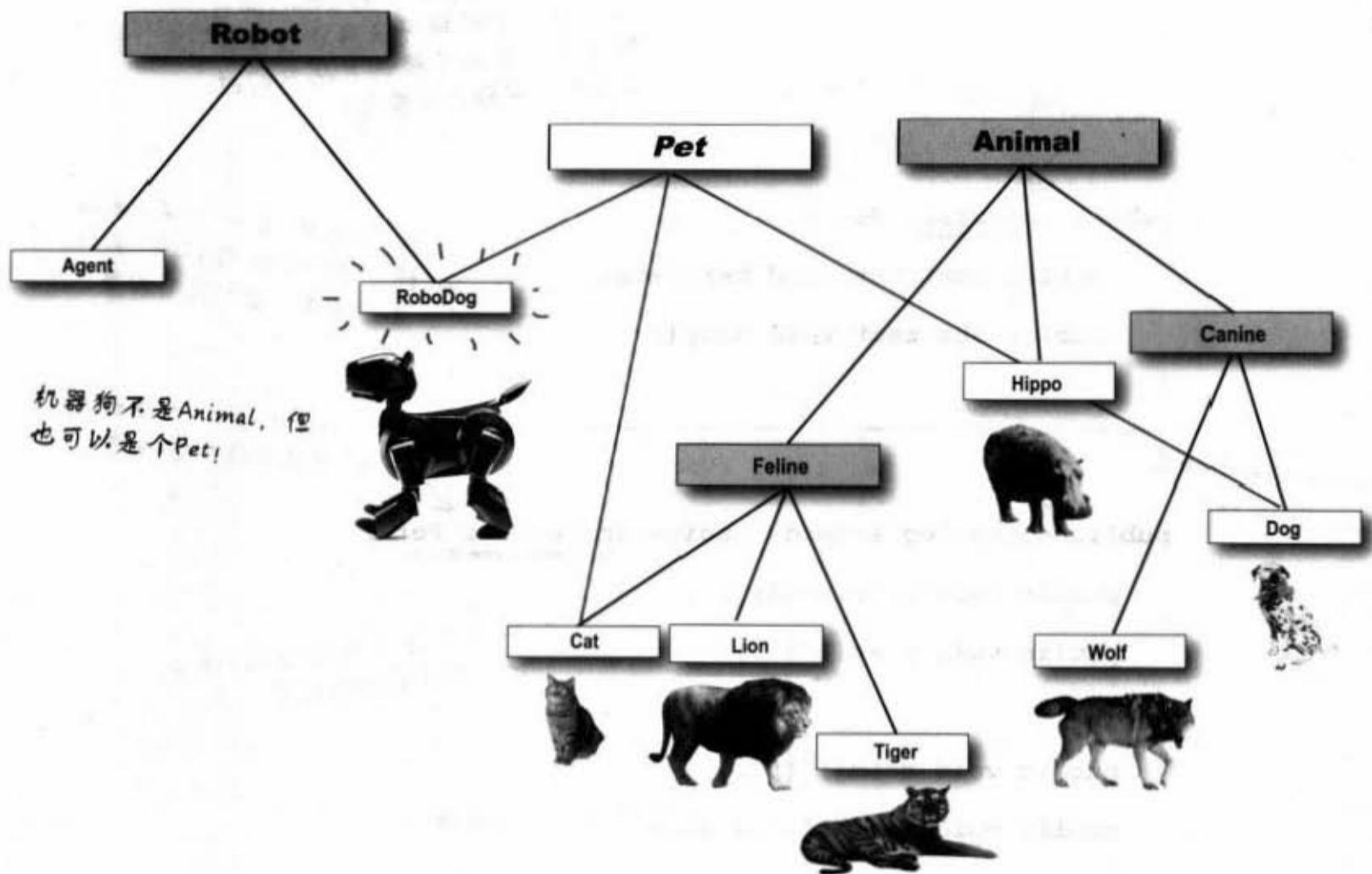
该接口的东西。这么说吧，使用接口你就可以继承超过一个以上的来源。类可以 extend 过某个父类，并且实现其他的接口。同时其他的类也可以实现同一个接口。因此你就可以为不同的需求组合出不同的继承层次。

答：多态、多态、多态。接口有无比的适用性，若你以接口取代具体的子类或抽象的父类作为参数或返回类型，则你就可以传入任何有实现

事实上，如果使用接口来编写程序，你就是在说：“不管你来自哪里，只要你实现这个接口，别人就会知道你一定会履行这个合约”。

大部分良好的设计也不需要在抽象的层次定义出实现细节，我们所需的只是个共同的合约定义。让细节在具体的子类上实现也是很合理的。

不同继承树的类也可以实现相同的接口



当你把一个类当作多态类型运用时，相同的类型必定来自同一个继承树，而且必须是该多态类型的子类。定义为Canine类型的参数可以接受Wolf与Dog，但无法忍受Cat或Hippo。

但当你用接口来作为多态类型时，对象就可以来自任何地方了。唯一的条件就是该对象必须是来自有实现此接口的类。允许不同继承树的类实现共同的接口对Java API来说是非常重要的。如果你想要将对象的状态保存在文件中，只要去实现Serializable这个接口就行。打算让对象的方法以单独的线程来执行吗？没问题，实现Runnable。有概念了吧。后面的章节会有关于

Serializable与Runnable的讨论，现在只要先掌握住这个概念就行。

更棒的是类可以实现多个接口！

通过继承结构，Dog对象IS-A Canine、IS-A Animal、IS-A Object。但Dog IS-A Pet是通过接口实现的机制达成的，并同时也能够实现其他的接口：

```
public class Dog extends Animal implements
Pet, Saveable, paintable { ... }
```



要如何判断应该是设计类、子类、抽象类或接口呢？

- ▶ 如果新的类无法对其他的类通过IS-A测试时，就设计不继承其他类的类。
- ▶ 只有在需要某类的特殊化版本时，以覆盖或增加新的方法来继承现有的类。
- ▶ 当你需要定义一群子类的模板，又不想让程序员初始化此模板时，设计出抽象的类给它们用。
- ▶ 如果想要定义出类可以扮演的角色，使用接口。

调用父类的方法

问：如果创建出一个具体的子类且必须要覆盖某个方法，但又需要执行父类的方法时要怎么办？也就是说不打算完全地覆盖掉原来的方法，只是要加入额外的动作要怎么做？

答：呃……想想看“extends”的字义。设计良好的面向对象要注意到如何编写出必须被覆盖的程序代码。换言之，就是在抽象的类中编写能够共同的实现，让子类加入其余特定的部分。super这个关键词能让你在子类中调用子类的方法。

```
父类的方法，带有子类可以运用的部分  
abstract class Report {  
    void runReport() {  
        // 设置报告  
    }  
    void printReport() {  
        // 输出  
    }  
  
class BuzzwordsReport extends Report {  
  
    void runReport() {  
        super.runReport(); ← 调用父版的方法  
        buzzwordCompliance();  
        printReport();  
    }  
    void buzzwordCompliance() {...}  
}
```

如果在子类中指定下面的命令：

`super.runReport();`

父类的方法就会执行。

super.runReport();

对子类的对象调用会去执行子类覆盖过的方法，但在子类中可以去调用父类的方法



要点

- 如果不想让某个类被初始化，就以abstract这个关键词将它标记为抽象的。
- 抽象的类可以带有抽象和非抽象的方法。
- 如果类带有抽象的方法，则此类必定标识为抽象的。
- 抽象的方法没有内容，它的声明是以分号结束。
- 抽象的方法必须在具体的类中运行。
- Java所有的类都是Object (java.lang.Object) 直接或间接的子类。
- 方法可以声明Object的参数或返回类型。
- 不管实际上所引用的对象是什么类型，只有在引用变量的类型就是带有某方法的类型时才能调用该方法。
- Object引用变量在没有类型转换的情况下不能赋值给其他的类型，若堆上的对象类型与所要转换的类型不兼容，则此转换会在执行期产生异常。

类型转换的例子：Dog d = (Dog) x.getObject(aDog);

- 从ArrayList<Object>取出的对象只能被Object引用，不然就要用类型转换来改变。
- Java不允许多重继承，因为那样会有致命方块的问题。
- 接口就好像是100%纯天然抽象类。
- 以interface这个关键词取代class来声明接口。
- 实现接口时要使用implements这个关键词。

例如：Dog implements Pet

- class可以实现多个接口。
- 实现某接口的类必须实现它所有的方法，因为这些方法都是public与abstract的。
- 要从子类调用父类的方法可以用super这个关键词来引用。

例如：super.RunReport();

问： 还是有点怪怪的，你没有解释为何ArrayList<DOG>返回的引用无需转换，却还是在方法中使用Object而不是Dog。使用ArrayList <Dog>时是否有什么怪招？

答： 说它是怪招一点也不为过。事实上ArrayList根本就不认识Dog，所以不必作类型转换是个怪招没错。

最简单的回答是：编译器帮你做了类型转换！<Dog>对编译器来说是个禁止将Dog类型以外的对象装进ArrayList的标记。就因为这样，所以编译器也很清楚将从此ArrayList中取出的对象转换为Dog类型是绝对安全的。

但这里面还有很多细节，我们会在讨论Collection的章节加以说明。

习题：图呢？



练习

这是挑战艺术天分的好机会。下面左方有一组类和接口的声明。你的任务是在右边画出类的图表。第一张图已经帮你画好了。使用虚线来表示实现并以实线来表示继承。

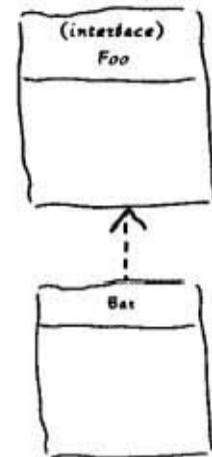
已知：

图呢？

1) public interface Foo { }

public class Bar implements Foo { }

1)



2) public interface Vinn { }

public abstract class Vout implements Vinn { }

2)

3) public abstract class Muffie implements Whuffle { }

public class Fluffie extends Muffie { }

public interface Whuffle { }

3)

4) public class Zoop { }

public class Boop extends Zoop { }

public class Goop extends Boop { }

4)

5) public class Gamma extends Delta implements Epsilon { }

public interface Epsilon { }

public interface Beta { }

public class Alpha extends Gamma implements Beta { }

public class Delta { }

5)

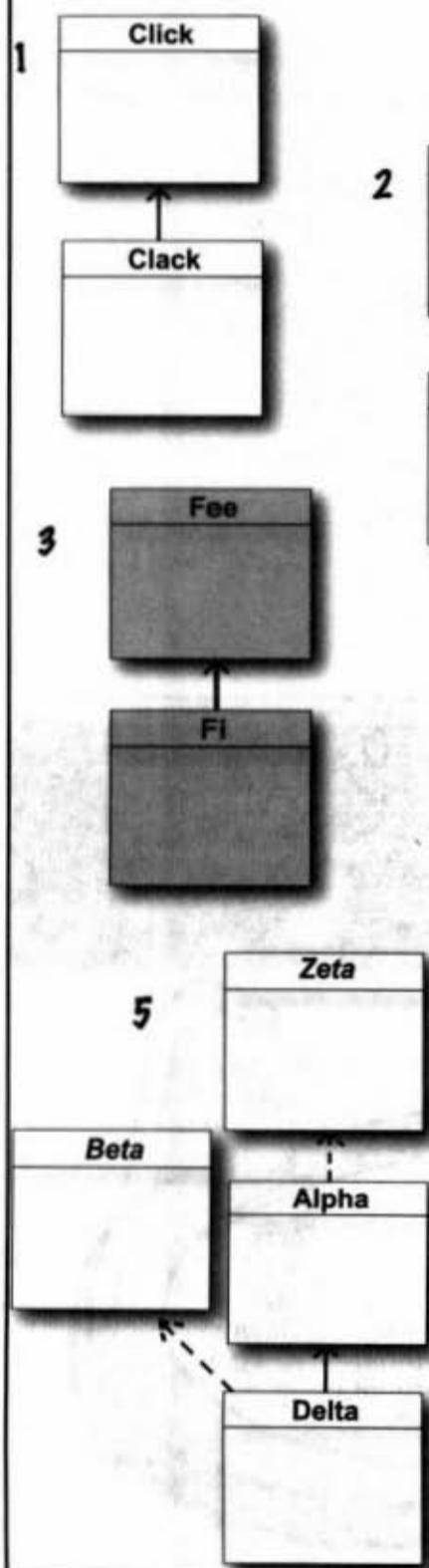


练习

下面左方有一组class和interface的图表。你的任务是在右边写出有效的Java声明。第一张图已经帮你写好声明。

已知：

Java 声明呢？



1) public class Click { }

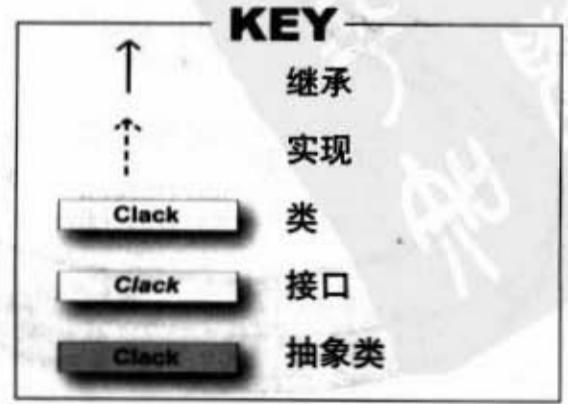
public class Clack extends Click { }

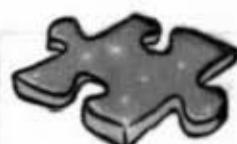
2)

3)

4)

5)





泳池 迷宫



你的任务是从游泳池中挑出程序片段并将它们填入右边的空格中。同一个片段可以重复使用，且泳池中有些多余的片段。填完空格的程序必须要能够编译与执行并产生出下面的输出。

```

    Nose {
    }

abstract class Picasso implements _____ {
    return 7;
}

class _____ {
    class _____ {
        return 5;
    }
}

```

注意：每一个片段
可以重复使用

```

public _____ extends Clowns {
    public static void main(String [] args) {
        _____
        i[0] = new _____
        i[1] = new _____
        i[2] = new _____
        for(int x = 0; x < 3; x++) {
            System.out.println(
                _____
                + " " + _____ .getClass( ) );
        }
    }
}

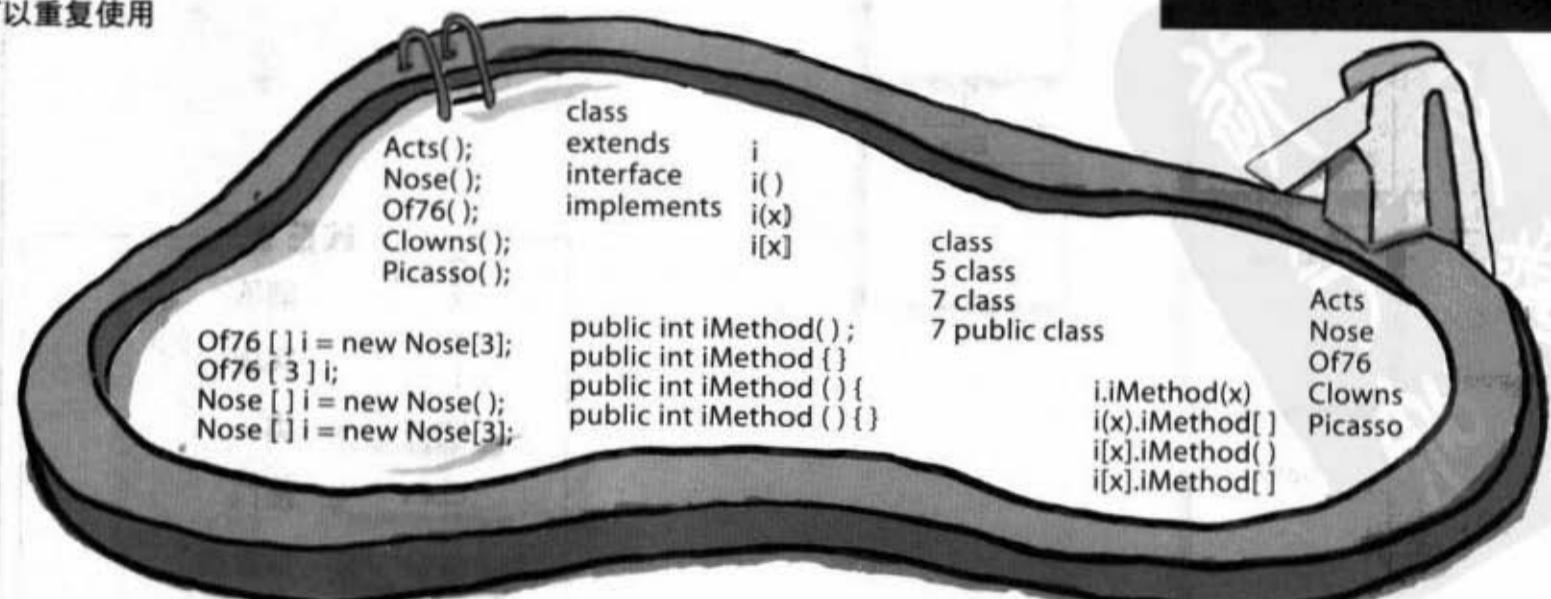
```

输出

```

File Edit Window Help BeAfraid
%java _____
5 class Acts
7 class Clowns
_____ Of76

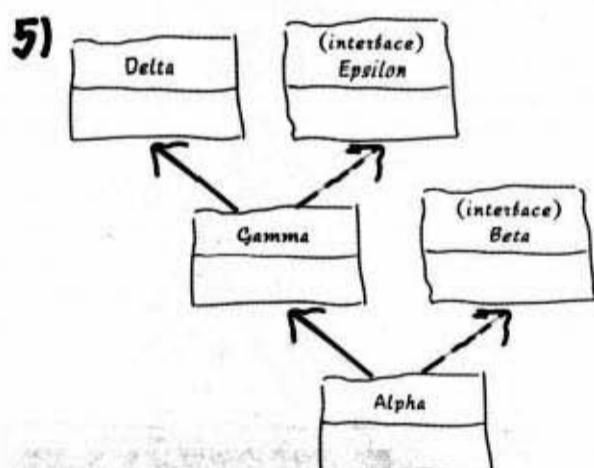
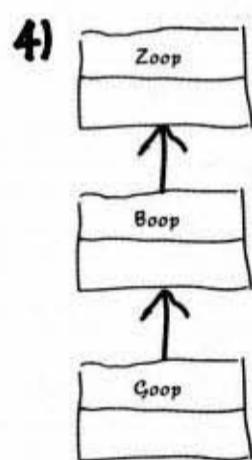
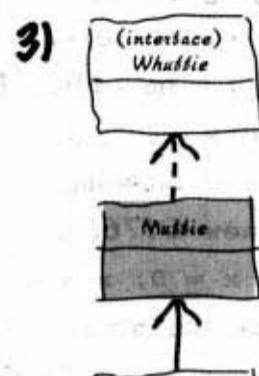
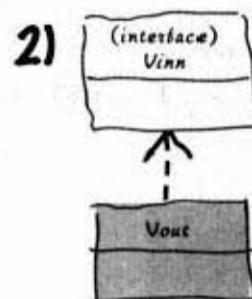
```





练习解答

艺术天分



Java 声明

2) `public abstract class Top { }`
`public class Tip extends Top { }`

3) `public abstract class Fee { }`
`public abstract class Fi extends Fee { }`

4) `public interface Foo { }`
`public class Bar implements Foo { }`
`public class Baz extends Bar { }`

5) `public interface Zeta { }`
`public class Alpha implements Zeta { }`
`public interface Beta { }`
`public class Delta extends Alpha implements Beta { }`



```

interface Nose {
    public int iMethod( );
}

abstract class Picasso implements Nose {
    public int iMethod( ) {
        return 7;
    }
}

class Clowns extends Picasso {}

class Acts extends Picasso {
    public int iMethod( ) {
        return 5;
    }
}

```

```

public class Of76 extends Clowns {
    public static void main(String [] args) {
        Nose [ ] i = new Nose [3];
        i[0] = new Acts();
        i[1] = new Clowns();
        i[2] = new Of76();
        for(int x = 0; x < 3; x++) {
            System.out.println( i [x].iMethod()
                + " " + i [x].getClass());
        }
    }
}

```

输出

```

File Edit Window Help KillTheMime
%java Of76
5 class Acts
7 class Clowns
7 class Of76

```

9 构造器与垃圾收集器

对象的前世今生



忽然一阵阴风吹过来，它还来不及开口，垃圾收集器马上就取走它的性命，我吓得两腿发软，裤底……

对象有生有死。你必须为对象的生命循环周期负责。你决定着对象何时创建，如何创建，也决定着何时销毁对象。其实你不是真的自消灭对象，只是声明要放弃它而已。一旦它被放弃了，冷血无情的垃圾收集器（GC）就会将它蒸发掉、回收对象所占用的内存空间。如果你要编写Java程序，就必须创建对象。早晚你得将它们释放掉，不然就会出现内存不足的问题。这一章会讨论对象如何创建、存在于何处以及如何让保存和抛弃更有效率。这代表我们会述及堆、栈、范围、构造器、超级构造器、空引用等。注意：内容含有死亡成份，12岁以下儿童需由家长陪同观赏。

栈与堆：生存空间

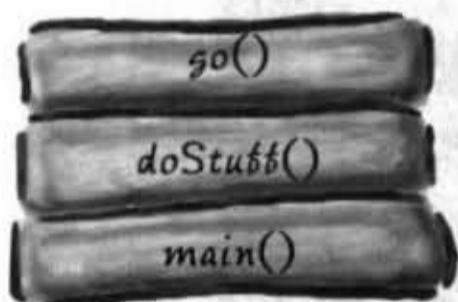
在我们能够了解创建对象真正发生的情况之前，我们必须先退回一步来看。我们需要对生存在Java中的事物更加了解。这代表我们必须对栈与堆有更多的认识。在Java中，程序员会在乎内存中的两种区域：对象的生存空间堆（heap）和方法调用及变量的生存空间（stack）。当Java虚拟机启动时，它会从底层的操作系统取得一块内存，并以此区段来执行Java程序。

至于有多少内存，以及你是否能够调整它都要看Java虚拟机与平台的版本而定。但通常你对这些事情无法加以控制。如果程序设计得不错的话，你或许也不太需要在乎。

我们知道所有的对象都存活于可垃圾回收的堆上，但我们还没看过变量的生存空间。而变量存在于哪一个空间要看它是哪一种变量而定。这里说的“哪一种”不是它的类型，而是实例变量或局部变量。后者这种区域变量又被称为栈变量，该名称已经说明了它所存在的区域。

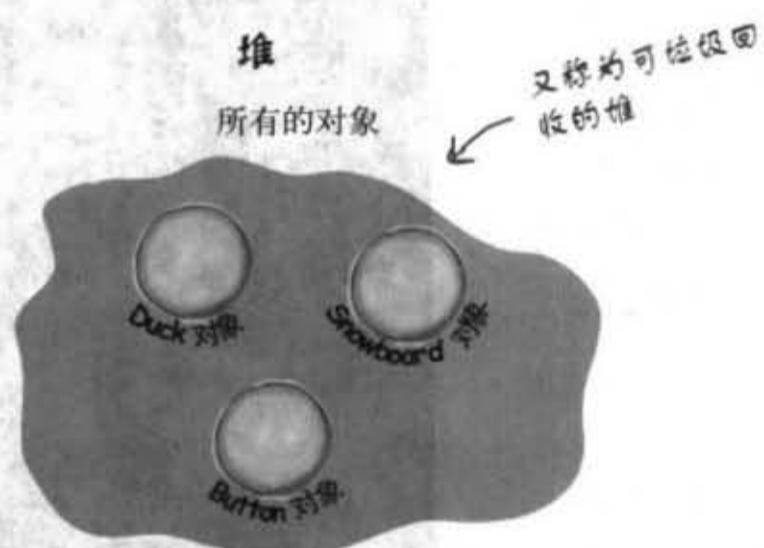
栈

方法调用和局部变量。



堆

所有的对象



实例变量

实例变量是被声明在类而不是方法里面。它们代表每个独立对象的“字段”（每个实例都能有不同的值）。实例变量存在于所属的对象中。

```
public class Duck {
    int size; // 每个Duck对象都会有独立的size
}
```

局部变量

局部变量和方法的参数都是被声明在方法中。它们是暂时的，且生命周期只限于方法被放在栈上的这段期间（也就是方法调用至执行完毕为止）。

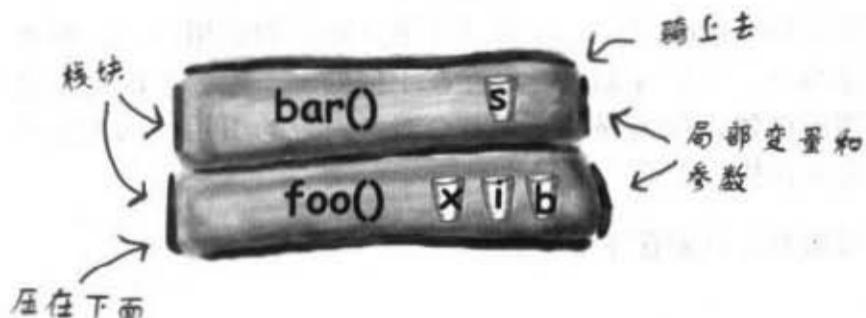
```
public void foo(int x) {
    int i = x + 3; // 参数x和变量i, b都是局部变量
    boolean b = true;
}
```

方法会被堆在一起

当你调用一个方法时，该方法会放在调用栈的栈顶。实际被堆上栈的是堆栈块，它带有方法的状态，包括执行到哪一行程序以及所有的局部变量的值。

栈顶上的方法是目前正在执行的方法（先假设只有一个，第14章有更多的说明）。方法会一直待在这里直到执行完毕，如果foo()方法调用bar()方法，则bar()方法会放在foo()方法的上面。

放了两个方法的栈



栈顶上方法是目前正在执行中的

```
public void doStuff() {
    boolean b = true;
    go(4);
}

public void go(int x) {
    int z = x + 24;
    crazy();
    // 假设还有很多程序代码
}

public void crazy() {
    char c = 'a';
}
```

stack 的情境

左边有3个方法，第一个方法在执行过程中会调用第二个方法，第二个会调用第三个。每个方法都在内容中声明一个局部变量，而go()方法还有声明一个参数（这代表go()方法有两个局部变量）。

① 某段程序代码调用了doStuff()使得doStuff()被放在stack最上方的栈块中。

② doStuff()调用go()。go()就被放在栈顶。

③ go()又调用crazy()使得crazy()现在处于栈顶。

④ 当crazy()执行完成后，它的堆栈块就被释放掉。执行就回到了go()。



有关对象局部变量

要记得非primitive的变量只是保存对象的引用而已，而不是对象本身。你已经知道对象存在于何处——堆。不论对象是否声明或创建，如果局部变量是个对该对象的引用，只有变量本身会放在栈上。

对象本身只会存在于堆上。

```
public class StackRef {  
    public void foof() {  
        barf();  
    }  
  
    public void barf() {  
        Duck d = new Duck(24);  
    }  
}
```

barf()会声明并创造出对
Duck的引用变量d



不论对象是在哪里声明的，它总是运行在堆上

*there are no
Dumb Questions*

问：到底为什么要知道栈与堆的机制？这真地跟我有关系吗？

答：如果想要了解变量的有效范围(scope)、对象的建立、内存管理、线程(thread)和异常处理，则认识栈与堆是很重要的。后面的章节会讨论到有关线程与异常处理的部分，其余的都会在这一章讨论。你无需知道它们是如何实现的，只要能够理解这几页的内容就足够了。

要点

- 我们关心栈与堆这两种内存空间。
- 实例变量是声明在类中方法之外的地方。
- 局部变量声明在方法或方法的参数上。
- 所有局部变量都存在于栈上相对应的堆栈块中。
- 对象引用变量与primitive主数据类型变量都是放在栈上。
- 不管是实例变量或局部变量，对象本身都会在堆上。

如果局部变量生存在栈上，那么实例变量呢？

当你要新建一个CellPhone()时，Java必须在堆上帮CellPhone找一个位置。这会需要多少空间呢？足以存放该对象所有实例变量的空间。没错，实例变量存在于对象所属的堆空间上。

记住对象的实例变量的值是存放于该对象中。如果实例变量全都是primitive主数据类型的，则Java会依据primitive主数据类型的大小为该实例变量留下空间。int需要32位，long需要64位，依此类推。Java并不在乎私有变量的值，不管是32或32,000,000的int都会占用32位。

但若实例变量是个对象呢？如果CellPhone对象带有一个Antenna对象呢？也就是说CellPhone带有Antenna类型的引用变量呢？

当一个新建对象带有对象引用的变量时，此时真正的问题是：是否需要保留对象带有的所有对象的空间？不是这样的。无论如何，Java会留下空间给实例变量的值。但是引用变量的值并不是对象本身，所以若CellPhone带有Antenna，Java只会留下Antenna引用量而不是对象本身所用到的空间。

那么Antenna对象会取得在堆上的空间吗？我们得先知道Antenna对象是在何时创建的。这要看实例变量是如何声明的。如果有声明变量但没有给它赋值，则只会留下变量的空间：

```
private Antenna ant;
```

直到引用变量被赋值一个新的Antenna对象才会在堆上占有空间：

```
private Antenna ant = new Antenna();
```

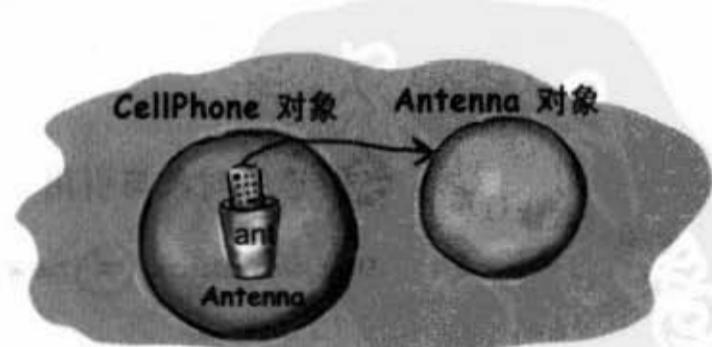


带有两个 primitive 主数据类型的实例变量的对象。变量所需的空间是在对象中。



对象带有引用到 Antenna 对象的变量，但是实际上没有初始 Antenna 对象的情形。

```
public class CellPhone {  
    private Antenna ant;  
}
```



对象带有一个新建出的 Antenna 对象

```
public class CellPhone {  
    private Antenna ant = new Antenna();  
}
```

创建对象的奇迹

现在你已经知道变量和对象的生存空间，我们可以开始更深入对象的创建。要记得声明对象和赋值有3个步骤：声明引用变量、创建对象、连接对象和引用。

但是第二个步骤还是个谜团——新对象的诞生。准备好接收新知识。

3个步骤的回顾：声明、创建、赋值

创造出新的引用变量
给该类型

1 声明引用变量

Duck myDuck = new Duck();



Duck引用

呈现奇迹

2 创建对象

Duck myDuck = new Duck();

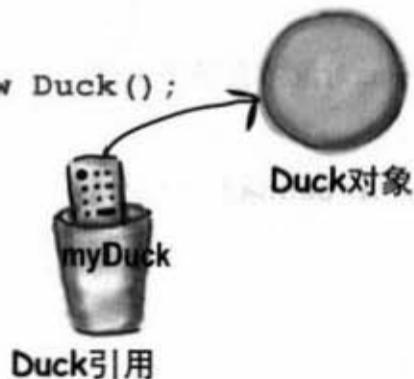


Duck对象

赋值对象给引用

3 连接对象与引用

Duck myDuck = new Duck();



Duck对象

看起来很像是在调用 Duck() 这个方法

```
Duck myDuck = new Duck();
```

看起来像是在调用
Duck()方法

并不是。

我们是在调用 Duck 的构造函数。

构造函数看起很像方法，感觉上也很像方法，但它并不是方法。它带有 new 的时候会执行的程序代码。换句话说，这段程序代码会在你初始一个对象的时候执行。

唯一能够调用构造函数的办法就是新建一个类。（严格说起来，这是唯一在构造函数之外能够调用构造函数的方式，本章稍后会讨论这个部分）。

构造函数带有你在初始化对象时会执行的程序代码。也就是新建一个对象时就会被执行。

就算你没有自己写构造函数，编译器也会帮你写一个。

哪里来的构造函数？

我们没有写啊，难道说这本书有缺页？

你可以帮类编写构造函数，但如果你没有写，编译器会偷偷帮你写！

下面就是编译器写出来的

```
public Duck() {  
}
```

有没有发现少了什么？这跟方法有什么不同之处？

方法有返回类型，构造函数没有返回类型

```
public Duck() {  
    // 构造代码在此  
}
```

一定要与类的名称相同

构造Duck

构造函数的一项关键特征是它会在对象能够被赋值给引用之前就执行。这代表你可以有机会在对象被使用之前介入。也就是说，在任何人取得对象的遥控器前，对象有机会对构造过程给予协助。在Duck的构造函数中，我们没有作出什么有意义的事情，但还是有展示出事件的顺序。



```
public class Duck {  
  
    public Duck() {  
        System.out.println("Quack");  
    }  
}
```

列出一行语句
←

构造函数让你有机会可以介入new的过程

```
public class UseADuck {  
  
    public static void main (String[] args) {  
        Duck d = new Duck();  
    }  
}
```

这样会启动Duck的构造函数
←

```
File Edit Window Help Quack  
% java UseADuck  
Quack
```

Sharpen your pencil

构造函数让你可以在构造过程的步骤中参一脚。你能否想象这有什么用处吗？如果Car是赛车的类，对于右边几个情境你是否能想象出构造函数的用途？可以的话就打个勾。

- 记录已经构造出多少部赛车。
- 记录特定的状态。
- 给实例变量赋值。
- 留下创建对象的证据。
- 将对象加到ArrayList中。
- 创建HAS-A对象。
- _____ (自己想)

新建Duck状态的初始化

大部分的人都是使用构造函数来初始化对象的状态。也就是说设置和给对象的实例变量赋值。

```
public Duck() {
    size = 34;
}
```

这在开发者知道Duck类应该有多大时是没问题的。但如果是要由使用Duck的程序员来决定时应该怎么办？

你可以使用该类的setSize()来设定大小。但这会让Duck暂时处于没有大小数值的状态（实例变量没有默认值），且需要两行才能搞定。下面就是这么做的：

```
public class Duck {
    int size; ← 实例变量

    public Duck() {
        System.out.println("Quack"); ← 构造函数
    }

    public void setSize(int newSize) { ←
        size = newSize; ←
    } ← setter方法
}
```

```
public class UseADuck {

    public static void main (String[] args){
        Duck d = new Duck();
        ←
        d.setSize(42); ← 问题出在这里，Duck在此处已经建立。
        ← 但是却没有size值！你必须依赖Duck
        ← 的用户记得要设定大小。
    }
}
```

*there are no
Dumb Questions*

问：既然编译器会帮你写，那为何还要自己写构造函数？

答：如果你在创建对象时需要有程序代码帮忙初始化，那你就得自己编写构造函数。例如说你需要通过用户的输入来完成对象的创建。另外一个原因与父类的构造函数有关，稍后会讨论这个部分。

问：如何分辨构造函数和方法？

答：Java可以有与类同名的方法而不会变成构造函数。其中的差别在于是否有返回类型。构造函数不会有返回类型。

问：构造函数会被继承吗？

答：不会。我们稍后会讨论到这个部分。

使用构造函数来初始化 Duck 的状态

如果某种对象不应该在状态被初始化之前就使用，就别让任何人能够在没有初始化的情况下取得该种对象！让用户先构造出Duck对象再来设定大小是很危险的。如果用户不知道，或者忘记要执行setSize()怎么办？

最好的方法是把初始化的程序代码放在构造函数中，然后把构造函数设定成需要参数的。

今天的晚餐是……烤鸭！？
对不起，我是天鹅，我真地是
天鹅……

```
public class Duck {
    int size;

    public Duck(int duckSize) {
        System.out.println("Quack");
        size = duckSize;
    }

    System.out.println("size is " + size);
}
```

构造函数加上参数

使用参数的值来设定size这个
实例变量

```
public class UseADuck {

    public static void main (String[] args) {
        Duck d = new Duck(42);
    }
}
```

传值给构造函数

只用一行就可以创
造出新的Duck并且设
定好大小。

```
File Edit Window Help Honk
% java UseADuck
Quack
size is 42
```

Duck的简易饲养方法

一定要有不需参数的构造函数。

如果Duck的构造函数需要一项参数会怎样？想想看。上一页的Duck只有一个构造函数，且它需要一个int型的size参数。这也许不是个问题，但却让程序员感到更为困难，特别是在不知道Duck的大小时。如果有预设的大小让程序员在不知道适当大小时也可以创建出Duck不是更好吗？

想象一下你可以让用户在创建Duck时有两个选项：一个可以指定Duck的大小（通过构造函数的参数），另外一个使用默认值而无需指定大小。

你无法只依靠单一的构造函数就能够很清楚地达到这个目的。要记得，如果某个方法或构造函数有一项参数，你就必须在调用该方法或构造函数的时候传入适当的参数。你没有办法作出一种没给参数时就使用默认值的方法，因为在这个情况下没有给参数就无法通过编译程序。也许你可以用下面这种不太理想的方法取代：

```
public class Duck {
    int size;

    public Duck(int newSize) {
        if (newSize == 0) { ←
            size = 27;
        } else {
            size = newSize;
        }
    }
}
```

如果参数值为0就
使用默认的大小

这代表程序员必须要知道传入0对于创建Duck的构造函数意味着要使用默认的大小而不是真正的0。万一程序员真的做出0大小的Duck怎么办？这样的问题在于传入0的意图无法确实的分辨。

你需要有两种方法来创建出新的Duck：

```
public class Duck2 {
    int size;

    public Duck2() {
        // 指定默认值
        size = 27;
    }

    public Duck2(int duckSize) {
        // 使用参数设定
        size = duckSize;
    }
}
```

知道大小时：

```
Duck2 d = new Duck2(15);
```

不知道大小时：

```
Duck2 d2 = new Duck2();
```

因此这会需要两个构造函数来分辨两种选项。一个需要参数，另外一个不需要参数。如果一个类有一个以上的构造函数，这代表它们也是重载的。

编译器一定会帮你写出没有参数的构造函数吗？No！

你可能会认为如果你只编写有参数的构造函数，编译器会看出你没有无参数的构造函数而帮你弄出一个来。别再相信没有事实根据的说法了，编译器只会在你完全没有设定构造函数时才会调用。

如果你已经写了一个有参数的构造函数，并且你需要一个没有参数的构造函数，则你必须自己动手写！

只要你有自己写的构造函数，不管是哪一种，这都会像是在跟编译器说：“老兄，我自己的构造函数不用你管”。

如果类有一个以上的构造函数，则参数一定要不一样。

这包括了参数的顺序与类型，只要是不一样就可以。这就跟方法的重载是相同的，不过细节会留到其他的章节再讨论。

……嗯，这跟构造函数很像，如果你完全没有写，编译器就会帮你写一个……



重载构造函数的意思代表你有一个以上的构造函数且参数都不相同

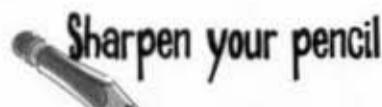
下面列出的构造函数都是合法的，因为参数都不相同。假设说有两个构造函数的参数都是只有一个int，则肯定无法通过编译程序。编译器看的是参数的类型和顺序而不是参数的名字。你可以做出相同类型但是顺序不同的参数。使用String以及int型的参数顺序与使用int以及String型的参数顺序是不同的。

4个不同的构造函数代表
有4种不同的创建方式



要点

- 实例变量保存在所属的对象中，位于堆上。
- 如果实例变量是个对对象的引用，则引用与对象都是在堆上。
- 构造函数是个会在新建对象的时候执行程序代码。
- 构造函数必须与类同名且没有返回类型。
- 你可以用构造函数来初始被创建对象的状态。
- 如果你没有写构造函数，编译器会帮你安排一个。
- 默认的构造函数是没有参数的。
- 如果你写了构造函数，则编译器就不会调用。
- 最好能有无参数的构造函数让人可以选择使用默认值。
- 重载的构造函数意思是超过一个以上的构造函数。
- 重载的构造函数必须有不同的参数。
- 两个构造函数的参数必须不同。
- 实例变量有默认值，原始的默认值是 0/0.0/false，引用的默认值是 null。



猜猜看哪个新建Duck()会用到哪个构造函数？我们已经帮你找到一个。

```
public class TestDuck {
    public static void main(String[] args) {
        int weight = 8;
        float density = 2.3F;
        String name = "Donald";
        long[] feathers = {1,2,3,4,5,6};
        boolean canFly = true;
        int airspeed = 22;

        Duck[] d = new Duck[7];
        d[0] = new Duck();
        d[1] = new Duck(density, weight);
        d[2] = new Duck(name, feathers);
        d[3] = new Duck(canFly);
        d[4] = new Duck(3.3F, airspeed);
        d[5] = new Duck(false);
        d[6] = new Duck(airspeed, density);
    }
}
```

```
class Duck {
    int pounds = 6;
    float floatability = 2.1F;
    String name = "Generic";
    long[] feathers = {1,2,3,4,5,6,7};
    boolean canFly = true;
    int maxSpeed = 25;

    public Duck() {
        System.out.println("type 1 duck");
    }

    public Duck(boolean fly) {
        canFly = fly;
        System.out.println("type 2 duck");
    }

    public Duck(String n, long[] f) {
        name = n;
        feathers = f;
        System.out.println("type 3 duck");
    }

    public Duck(int w, float f) {
        pounds = w;
        floatability = f;
        System.out.println("type 4 duck");
    }

    public Duck(float density, int max) {
        floatability = density;
        maxSpeed = max;
        System.out.println("type 5 duck");
    }
}
```

问：先前你说过最好要有没参数的构造函数以便让用户可以调用使用我们提供默认值的构造函数。但若不可能提供默认值的时候还应该要提供无参数的构造函数吗？

答：没错。有时候有默认值的无参数构造函数是不合理的。在Java API中有些类就没有无参数的构造函数，比如说Color这个类。它是用来设定字型或GUI按钮的颜色。当你制作Color的实例时，该实例会代表特定的颜色。如果你要用到Color，就必须以某种方式指定颜色：

```
Color c = new Color(3, 45, 200);
```

这是使用3个int来代表RGB三色的构造函数，后面讨论Swing的章节会有说明。如果没有给颜色，那Java API的设计人也许可以给你一个预设桃红色，想要吗？如果以这种方式创建Color对象：

```
Color c = new Color();
```

编译器会向你抱怨没有这样的构造函数：

```
File Edit Window Help StopBeingStupid
cannot resolve symbol
:constructor Color()
location: class java.awt.Color
Color c = new Color();
^
1 error
```

构造函数 迷你小回顾：

- 构造函数是在新建类时会执行的程序。

```
Duck d = new Duck();
```

- 构造函数必须与类的名字一样，且没有返回类型。

```
public Duck(int size) { }
```

- 如果你没有写构造函数，则编译器会帮你写一个没有参数的

```
public Duck() { }
```

- 一个类可以有很多个构造函数，但不能有相同的参数类型和顺序，这叫作重载过的构造函数。

```
public Duck() { }
public Duck(int size) { }
public Duck(String name) { }
public Duck(String name, int size) { }
```

研究显示挑战智力大考验最多可以提升神经的大小到 42% 以上。



想到父类吗？

在创建 Dog 的时候，Canine 的构造函数是否应该要执行？

如果父类是抽象的，那它是否有构造函数吗？

接下来会看到这些主题，你现在应该独立地思考一下构造函数与父类之间的关系。

there are no
Dumb Questions

问： 构造函数应该是公有的吗？

答： 不。构造函数可以是公有、私有或不指定的。第 16 章和附录 B 有关于不指定的预设存取权讨论。

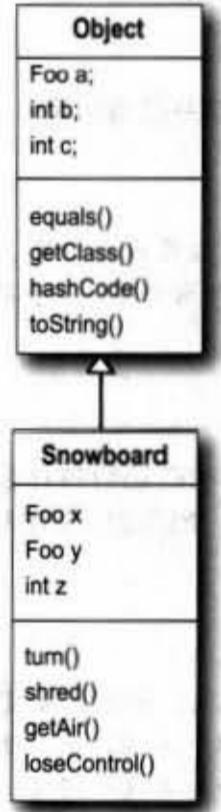
问： 一个私有的构造函数有什么作用？没有人能够调用它，所以也就没有人能够创建该对象？

答： 不是这么说的。私有不是完全不能存取，它代表该类以外不能存取。这听起来很矛盾吧？下一章会讨论这个问题。

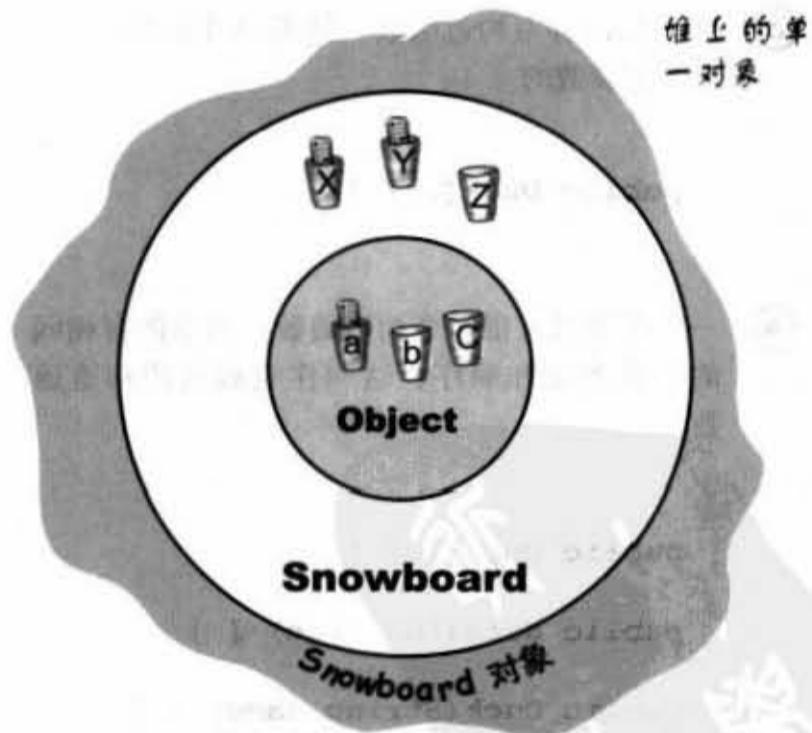
等一下……我们都还没有谈到父类以及继承与构造函数之间的关系

这样才有趣。还记得上一章我们说到Snowboard对象把Object部分包在自己的核心吗？那个讨论的重点在于每个对象不只是保存自行声明的变量，还有从父类来的所有东西（至少会带有Object，因为每个类都有继承过这个类）。

因此在创建某个对象时（完全没有其他方法能够创建对象，只能通过new来产生新对象），对象会取得所有实例变量所需的空间，这当然也包括一路继承下来的东西。想象一下，父类也许会有一个setter以包装私用的变量。但此变量必须有空间。概念上，你可以把整个情境用下面的图来表示，对象就像洋葱是有层次的（请见《史瑞克》），每一层都代表某一级的父类。



Snowboard也有自己的实例变量，因此Snowboard需要自己加上Object的空间。



此图所示只有一个对象。但因为它带有Snowboard与Object的内容，所以这两个类的所有实例变量也会在这里。

父类的构造函数在对象的生命中所扮演的角色

在创建新对象时，所有继承下来的构造函数都会执行。

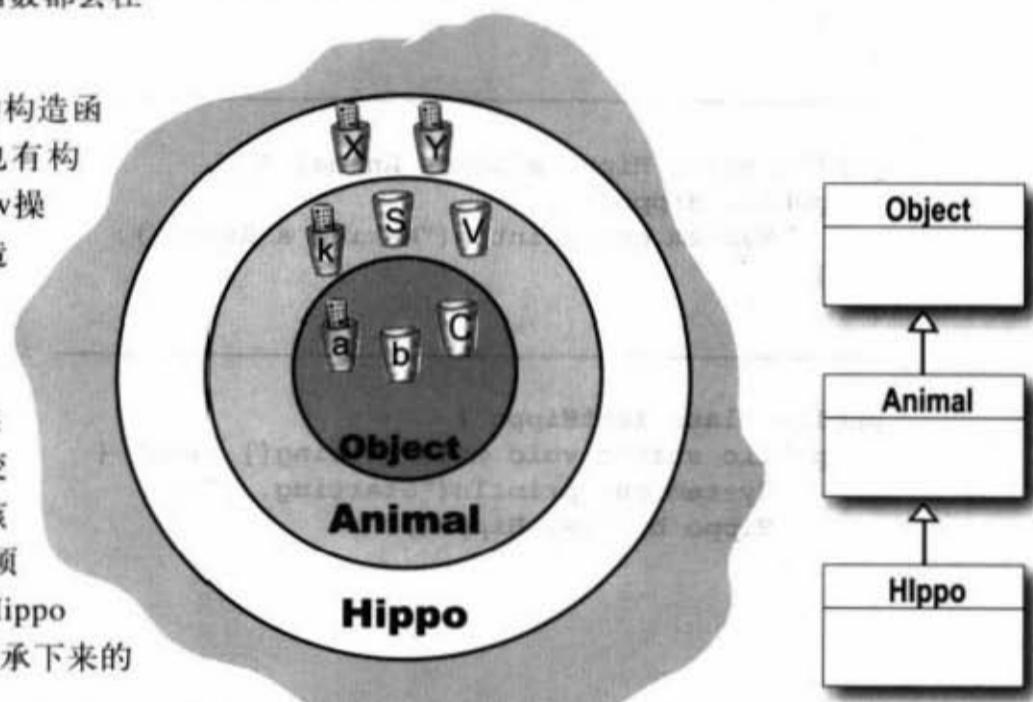
这代表每个父类都有一个构造函数（因为每个类至少都会有一个构造函数），且每个构造函数都会在子类对象创建时期执行。

执行new的指令是个重大事件，它会启动构造函数连锁反应。还有，就算是抽象的类也有构造函数。虽然你不能对抽象的类执行new操作，但抽象的类还是父类，因此它的构造函数会在具体子类创建出实例时执行。

在构造函数中用super调用父类的构造函数的部分。要记得子类可能会根据父类的状态来继承方法（也就是父类的实例变量）。完整的对象需要也是完整的父类核心，所以这就是为什么父类构造函数必须执行的原因。就算Animal上有些变量是Hippo不会用到的，但Hippo可能会用到某些继承下来的方法必须读取Animal的实例变量。

构造函数在执行的时候，第一件事是去执行它的父类的构造函数，这会连锁反应到Object这个类为止。

我们在接下来的几页会看到父类构造函数是如何被调用，以及你如何自行调用它们。你也会知道要如何调用有参数的父类构造函数。



在堆上的Hippo对象

Hippo对象IS-A Animal同时也IS-A Object。如果你要创建出Hippo，也得创建出Animal与Object的部分。

这样的过程被称为“构造函数链（Constructor Chaining）”

创建Hippo也代表创建 Animal与Object

```
public class Animal {
    public Animal() {
        System.out.println("Making an Animal");
    }
}

public class Hippo extends Animal {
    public Hippo() {
        System.out.println("Making a Hippo");
    }
}

public class TestHippo {
    public static void main (String[] args) {
        System.out.println("Starting...");
        Hippo h = new Hippo();
    }
}
```

Sharpen your pencil

哪一个输出才是对的？执行左边的程式代码得到A还是B的结果？

答案就在下面。

A

```
File Edit Window Help Swear
% java TestHippo
Starting...
Making an Animal
Making a Hippo
```

B

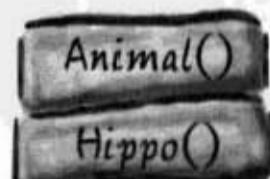
```
File Edit Window Help Swear
% java TestHippo
Starting...
Making a Hippo
Making an Animal
```

- ① 某个程序执行new Hippo()的动作，Hippo()的构造函数进入堆栈最上方的堆栈块

- ② Hippo()调用父类的构造函数导致Animal()的构造函数进入栈顶

- ③ Animal()调用父类的构造函数导致Object()的构造函数进入栈顶

- ④ Object()执行完毕，它的堆栈块被弹出，接着继续执行Animal()的



后一个执行完畢。

器类 A, Hippo()是專門使用的，但其實普

如何调用父类的构造函数？

以Duck的构造函数来说，你也许认为它会直接调用Animal()，但实际上不是这样的：

```
public class Duck extends Animal {
    int size;

    public Duck(int newSize) {
        → Animal(); ← 这不合法,
        size = newSize;
    }
}
```

调用父类构造函数的唯一方法是调用super()。

它看起来会像下面这样：

```
public class Duck extends Animal {
    int size;

    public Duck(int newSize) {
        super(); ← 要这么调用
        size = newSize;
    }
}
```

在你的构造函数中调用super()会把父类的构造函数放在堆栈的最上方。你猜父类的构造函数会做什么？它会调用它的父类构造函数。这会一路上去直到Object的构造函数为止。然后再一路执行、弹出回到原来的构造函数。

如果我们没有调用super() 会发生什么事？

你也许已经猜到了。

编译器会帮我们加上super()的调用。

所以编译器有两种涉入构造函数的方式：

● 如果你没有编写构造函数。

```
public ClassName() {
    super();
}
```

● 如果你有构造函数但没有调用 super()。

编译器会帮你对每个重载版本的构造函数加上下面这种调用：

super();

编译器帮忙加的一定会是没有参数的版本，假使父类有多个重载版本，也只有无参数的这个版本会被调用到。

小孩能够在父母之前出生吗？

如果你把父类想象成子类的父母，那就可以看出来谁是先存在的。父类的部分必须在子类创建完成之前就必须完整地成型。记住，子类对象可能需要动用到从父类继承下来的东西，所以那些东西必须要先完成。父类的构造函数必须在子类的构造函数之前结束。

再看一下252页的堆栈，你会发现Hippo的构造函数是第一个被调用的（在堆栈上的第一个），却也是最后一个完成的！每个子类的构造函数会立即调用父类的构造函数，如此一路往上直到Object。等到Object完成后会回去执行Animal的，然后等Animal完成后又回去执行Hippo剩下的构造函数。这是因为：

对super()的调用必须是构造函数的第一个语句*。



类Boop的可能构造函数

```
R public Boop() {  
    super(); ←
```

这么做是可以的，
因为super()的调用
是第一个命令

```
R public Boop(int i) {  
    super(); ←  
    size = i;  
}
```

```
R public Boop() {  
    } ←
```

这么做也可以，因
为编译器会自动把
super()加到最前面

```
R public Boop(int i) {  
    size = i; ←  
}
```

这就不行了，编译器
不会让你这么做

```
✗ public Boop(int i) {  
    size = i;  
    super(); ←
```

*有异常情况，见256页

有参数的父类构造函数

如果父类的构造函数有参数该怎么办？你能够传值进去吗？如果不行的话，则没有无参数构造函数的类将不能被继承。想象这个情景：所有的动物都有名字。所以Animal这个类有个getName()可以返回name实例变量的值。此实例变量是被标记为私有的，但Hippo这个子类有把getName()继承下来。问题来了：

Hippo有getName()这个方法但是没有name实例变量。Hippo要靠Animal的部分来维持name实例变量，然后从getName()来返回这个值，但Animal要如何取得这个值呢？唯一的机会是通过super()来引用父类，所以要从这里把name的值传进去，让Animal把它存到私有的name实例变量中。

```
public abstract class Animal {
    private String name; ← 每个Animal都会有名字

    public String getName() { ← Hippo会继承这个getter
        return name;
    }

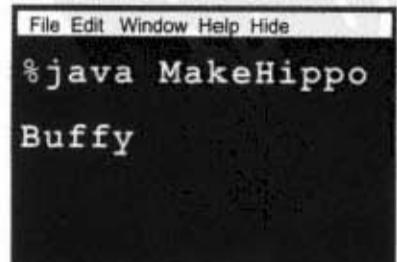
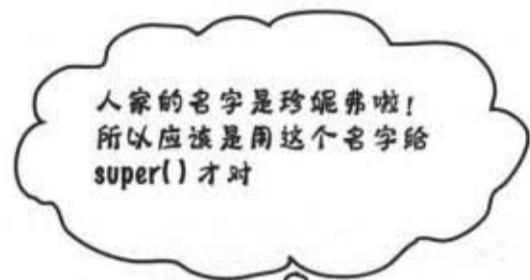
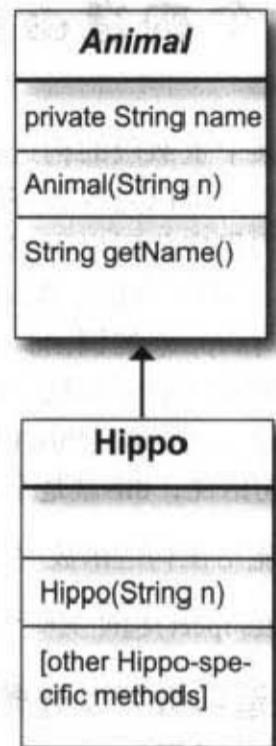
    public Animal(String theName) {
        name = theName; ← 有参数的构造函数，用来
    }
}
```

```
public class Hippo extends Animal {

    public Hippo(String name) {
        super(name); ← 这个构造函数会要求
    } ← 名称
}

← 传递Animal的构造函数
```

```
public class MakeHippo {
    public static void main(String[] args) {
        Hippo h = new Hippo("Buffy"); ← 创建Hippo，传入名字然后
        System.out.println(h.getName()); ← 再列出来看看
    }
}
```



从某个构造函数调用重载版的另一个构造函数

如果有某个重载版的构造函数除了不能处理不同类型的参数之外，可以处理所有的工作，那要怎么办？你不想让相同的程序代码出现在每个构造函数中（维护起来很麻烦），所以你想把程序代码只摆在某个构造函数中（包括对super()的调用）。如此一来，所有的构造函数都会先调用该构造函数，让它来执行真正的构造函数。这很容易，只要调用this()或this(aString)或this(27, x)就行。换句话说，this就是个对对象本身的引用。

this()只能用在构造函数中，且它必须是第一行语句！

这样会跟super()起冲突吗？所以你必须选择：

每个构造函数可以选择调用super()或this()，但不能同时调用！

使用this()来从某个构造函数调用同一个类的另外一个构造函数。

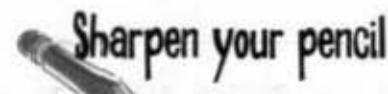
this()只能用在构造函数中，且必须是第一行语句。

super()与this()不能兼得。

```
class Mini extends Car {  
  
    Color color;  
  
    public Mini() {  
        this(Color.Red); ←  
    }  
  
    public Mini(Color c) {  
        super("Mini"); ←  
        color = c; ←  
        // 初始化动作  
    }  
  
    public Mini(int size) {  
        this(Color.Red); ←  
        super(size); ←  
    }  
}
```

无参数的构造函数以默认的颜色调用真正的构造函数
这才是真正的构造函数
有问题！不能同时调用super()和this()，两者只有一个会是第一行语句

```
File Edit Window Help Drive  
javac Mini.java  
Mini.java:16: call to super must  
be first statement in constructor  
        super();  
               ^
```



下面类SonOfBoo的构造函数中有某几个无法通过编译，试试看你能否找到不合法的是哪几个。在有问题的构造函数旁边划条线连到右边的错误信息上。

```
public class Boo {
    public Boo(int i) { }
    public Boo(String s) { }
    public Boo(String s, int i) { }
}
```

```
class SonOfBoo extends Boo {
    public SonOfBoo() {
        super("boo");
    }

    public SonOfBoo(int i) {
        super("Fred");
    }

    public SonOfBoo(String s) {
        super(42);
    }

    public SonOfBoo(int i, String s) {
    }

    public SonOfBoo(String a, String b, String c) {
        super(a,b);
    }

    public SonOfBoo(int i, int j) {
        super("man", j);
    }

    public SonOfBoo(int i, int x, int y) {
        super(i, "star");
    }
}
```



```
File Edit Window Help
%javac SonOfBoo.java
cannot resolve symbol
symbol : constructor Boo
(java.lang.String,java.lang.String)
```

```
File Edit Window Help Yadayadayada
%javac SonOfBoo.java
cannot resolve symbol
symbol : constructor Boo
(int,java.lang.String)
```

```
File Edit Window Help imNotListening
%javac SonOfBoo.java
cannot resolve symbol
symbol:constructor Boo()
```

我们已经了解了对象的诞生过程，但对象会存活多久呢？

对象的生命周期完全要看引用到它的“引用”。如果引用还活着，则对象也会继续活在堆上。如果引用死了（稍后会解释），则对象就会跟着殉情……陪葬……送命……

如果对象生命周期要看引用变量的生命周期而定，那变量到底会活多久？

这又要看它是局部变量或实例变量而定。下面的程序展示出局部变量的生命周期。

```
public class TestLifeOne {  
    public void read() {  
        int s = 42; ← s的范围只限于read()里面。  
        sleep();  
    }  
  
    public void sleep() {  
        s = 7;  
    } ↑ 非法使用！  
}
```



- ① 局部变量只会存活在声明该变量的方法中

```
public void read() {  
    int s = 42;  
    // 's' 只能用在此方法中  
    // 当方法结束时  
    // s 会完全消失
```

变量 s 只能在 read() 方法中。换句话说，此变量的范围只会在所属方法的范围内。其余的程序代码完全见不到 s。

- ② 实例变量的寿命与对象相同。如果对象还活着，则实例变量也会是活的。

```
public class Life {  
    int size;  
  
    public void setSize(int s) {  
        size = s;  
        // 's' 会在方法结束时  
        // 消失，但 size 在类中  
        // 到处都可用  
    }  
}
```

此时 s 变量（这次是方法的参数）的范围同样也只限制在所属的 setSize() 这个方法中。

“life”与“scope”的差别

Life

只要变量的堆栈块还存在于堆栈上，局部变量就算活着。也就是说，活到方法执行完毕为止。

Scope

局部变量的范围只限于声明它的方法之内。当此方法调用别的方法时，该变量还活着，但不在目前的范围内。执行其他方法完毕返回时，范围也就跟着回来。

```
public void doStuff() {
    boolean b = true;
    go(4);
}

public void go(int x) {
    int z = x + 24;
    crazy();
    // 这里有更多的代码
}

public void crazy() {
    char c = 'a';
}
```



- ① doStuff()运行在堆栈，变量b存活于scope中。

- ② 调用go(), x, z, b都活着，但只有b不在它的范围内。

- ③ 调用crazy(), 只有c在它的范围内。

- ④ 完成crazy(), c既不在它的范围内，也没活下来，只有x和z在它的范围内。

当局部变量活着的时候，它的状态会被保存。只要doStuff()还在堆栈上，b变量就会保持它的值。但b变量只能在doStuff()待在栈顶时才能使用。也就是说局部变量只能在声明它的方法在执行中才能被使用。

那引用变量呢？

引用的规则与 primitive 主数据类型相同。引用变量只能在处于它的范围内才能被引用，也就是说除非引用变量是在它的范围内，不然就不能使用对象的遥控器。真正的问题是：

“变量的生命周期如何影响对象的生命周期？”

只要有活着的引用，对象也就会活着。如果某个对象的引用已经不在它的范围内，但此引用还是活着的，则此对象就会继续活在堆上。

如果对对象的唯一引用死了，对象就会从堆中被踢开。引用变量会跟堆栈块一起解散，因此被踢开的对象也就正式的声明出局。关键在于知道何时对象会变成可被垃圾收集器回收的。

一旦对象符合垃圾收集器（GC）的条件，你就无需担心回收内存的问题。如果程序内存不足，GC就会去歼灭部分或全部的可回收对象。你可能还是会遇到内存不足的状况，但这要等到所有可回收的都被回收掉也还不够的时候才会发生。你要注意的是对象用完了就要抛弃，这样才能让垃圾收集器有东西可以回收。如果你把持着对象不放，垃圾收集器也帮不了什么忙。

除非有对对象的引用，否则该对象一点意义也没有。

如果你无法取得对象的引用，则此对象只是浪费空间罢了。

但若对象是无法取得的，GC会知道该怎么做的。那种对象迟早会葬送在垃圾收集器的手上。



当最后一个引用消失时，对象就会变成可回收的。

有3种方法可以释放对象的引用：

① 引用永久性的离开它的范围。

```
void go() {  
    Life z = new Life();  
}
```

z会在方法结束时
消失

② 引用被赋值到其他的对象上。

```
Life z = new Life();  
z = new Life();
```

第一个对象会在z被赋值
到别处时挂掉

③ 直接将引用设定为null。

```
Life z = new Life();  
z = null;
```

第一个对象会在z被赋值
为null时挂掉

对象杀手一号

引用永久性的
离开它的范围



```
public class StackRef {
    public void foof() {
        barf();
    }

    public void barf() {
        Duck d = new Duck();
    }
}
```

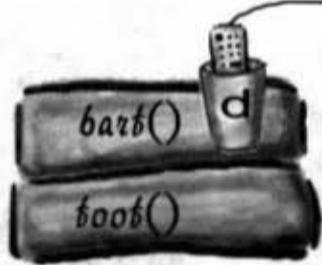
我真地很讨厌这个例子。



- 1 foof()被推到堆栈上，没有声明变量



- 2 barf()被推到堆栈上，创建一个对象以及对它的引用



新的Duck会放在堆上，只要barf()还在运行，d就还活着，所以鸭子也就没事

- 3 barf()执行完毕，因此d也就挂了



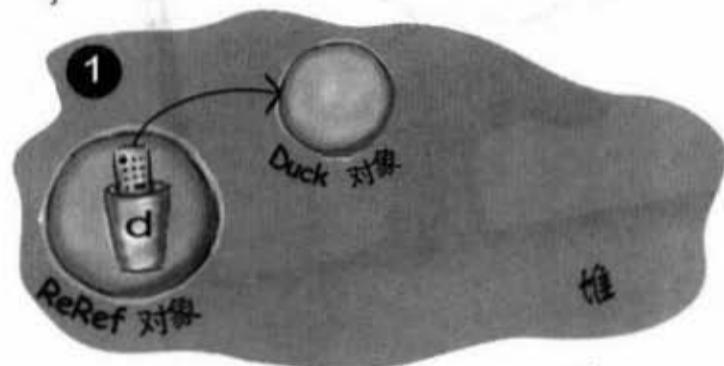
既然d已经不存在了，Duck也就等着要被垃圾收集器回收

对象杀手二号

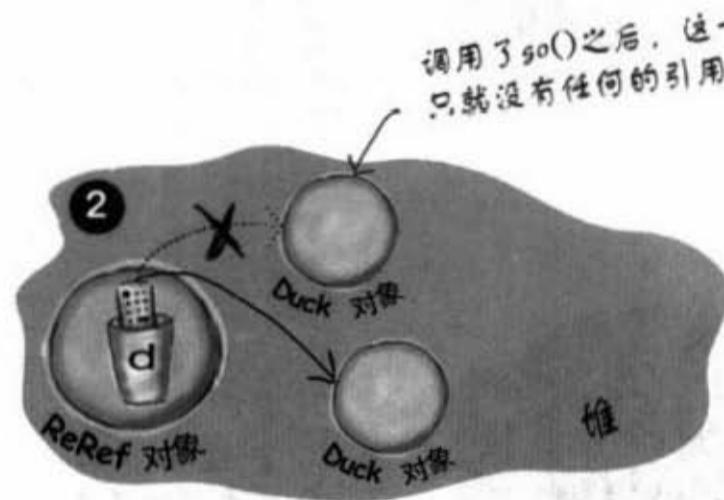
引用被赋值到
其他的对象上



```
public class ReRef {  
    Duck d = new Duck();  
  
    public void go() {  
        d = new Duck();  
    }  
}
```



新的Duck会放在堆上，只要ReRef还活着，d就没事，除非……



既然d引用到其他的Duck，第一个Duck就跟死掉是一样的

考古学家发现四千年前的对象

哇！难道它们不知道可以重置引用吗？也许古人真的很不喜欢内存的管理工作。



物件杀手三号

直接将引用

设定为null



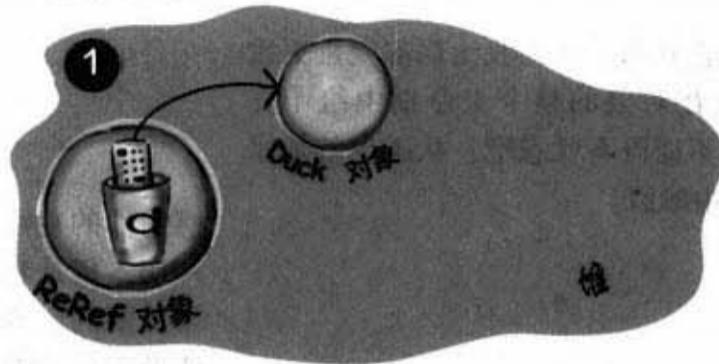
```
public class ReRef {
    Duck d = new Duck();
    public void go() {
        d = null;
    }
}
```

null的真相

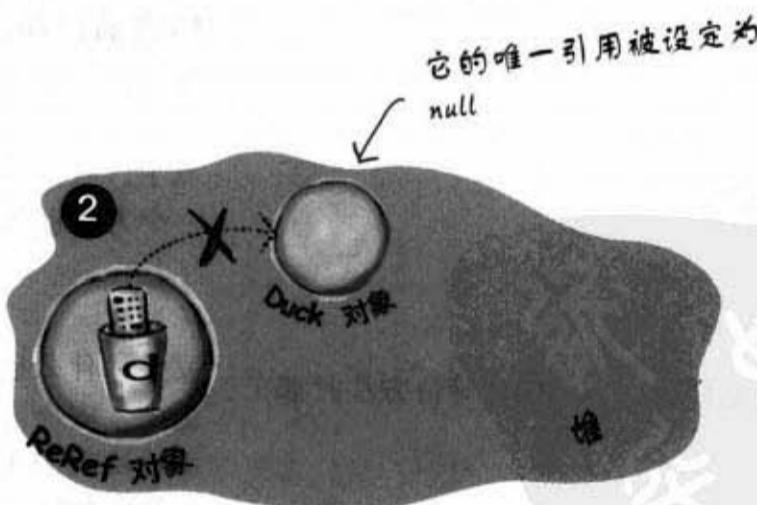
当你把引用设为null时，你就等于是抹除遥控器的功能。换句话说，你会拿到一个没有电视的遥控器。null是代表“空”的字节组合（实际上是什么只有Java虚拟机才会知道）。

如果你真地按下这种遥控器上的按钮，什么事情也不会发生。但在Java上，你是不能对null引用按钮的。因为Java虚拟机会知道（这是运行期，不是编译时的错误）你期待喵喵叫，但是却没有Cat可以执行！

对null引用使用圆点运算符会在执行期遇到NullPointerException这样的错误。后面会有讨论异常的章节。

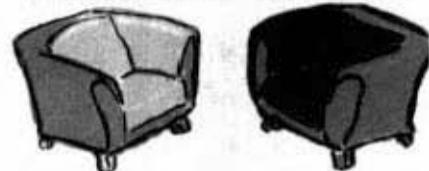


新的Duck会放在堆上，只要ReRef还活着，d就没事，除非……



d被设定成null，就好像没有操作对象的遥控器

Fireside Chats



今晚的话题： 实例变量与局部变量的生死对谈

实例变量

我想应该先从我开始，因为我比局部变量更重要。通常我会在对象的整个生命期中给予支持。毕竟对象不能没有状态吧？状态也就是保持在实例变量中的值。

误会可大了。我了解你在方法中的作用，只是你的命真的太短了。那也就是为何人们把你叫做“临时变量”的原因。

抱歉，我知道了。

我倒不知道这回事，那你们在等待方法时都在做什么？

局部变量

感谢你的观点，也感谢你对对象状态值所做的一切。但是我不想让大家误会。局部变量是相当重要的。容我引用你的说法：“毕竟对象不能没有行为吧？行为也就是保持在方法中的算法”。你也一定很清楚必须要有些局部变量才能让算法运作。

留点口德吧，“临时变量”在我们这边是很轻蔑的说法。我们比较喜欢自称“局部变量”、“栈变量”或者“变量作用域”。

算了。我们是不长命，但有时一个方法调用另一个方法也会让我们待在堆栈上很久。

发呆，什么也不做。但是我们所保存的值不会丢失，安全得很。要等到执行回到我们所处的堆栈块我们才会继续活动，不过这也代表我们又往生命终点迈进了一步。

实例变量

我看过你同伴出殡的新闻报导，看起来蛮惨的。我的意思是当方法执行完成后，堆栈块就直接被清除，那种死法应该很痛吧。

我跟对象一起生存在堆上，离西贡街与公众四方街口不远……嗯，其实是住在给我保存状态的对象里面才对。那附近地段很贵，物价也很高。

对啦，如果我是个猫对象上的实例变量所引用的跳蚤对象，当该变量被设定成null的时候，我就只好等着被收进垃圾堆，我的地方也会让出来给别人用。不过有人跟我说过，这猫不可能会洗澡的。

你不怕遇到警察吗？

局部变量

你还说呢。这叫做被弹出好吗？为什么不来说说你呢？我生存在堆栈上，你老兄生存哪啊？

但你也不一定会很长寿吧？例如说你是猫对象身上的跳蚤实例变量所引用的跳蚤对象。假使今天这猫执行了洗澡的行为，使得跳蚤属性被设定为null，那会发生什么事？

你就这么相信了？如果猫对象只有一个引用呢？假使这个唯一的引用是保存在局部变量中呢？如果声明此局部变量的方法执行完毕，你还不是得像我们一样。老实跟你说，我已经认命了，现在能够活一天就活一天，有机会喝喝酒、吃吃RAM，我就尽量吃喝。

我有一招可以逃过警察，你可以学学。如果你遇到警察的时候，就把……



我是垃圾收集器

假设批注部分实际上会是执行足够长时间的过程调用，下方右边有哪几行程序代码加到左边A位置会使得某一个额外的对象被认为是可以垃圾回收的？

```

public class GC {
    public static GC doStuff() {
        GC newGC = new GC();
        doStuff2(newGC);
        return newGC;
    }

    public static void main(String [] args) {
        GC gc1;
        GC gc2 = new GC();
        GC gc3 = new GC();
        GC gc4 = gc3;
        gc1 = doStuff();
        A
        // 调用更多的方法
    }

    public static void doStuff2(GC copyGC) {
        GC localGC
    }
}

```

1 copyGC = null;
2 gc2 = null;
3 newGC = gc3;
4 gc1 = null;
5 newGC = null;
6 gc4 = null;
7 gc3 = gc2;
8 gc1 = gc4;
9 gc3 = null;



最受欢迎 金对象奖

```

class Bees {
    Honey [] beeHA;
}

class Raccoon {
    Kit k;
    Honey rh;
}

class Kit {
    Honey kh;
}

class Bear {
    Honey hunny;
}

public class Honey {
    public static void main(String [] args) {
        Honey honeyPot = new Honey();
        Honey [] ha = {honeyPot, honeyPot, honeyPot, honeyPot};
        Bees bl = new Bees();
        bl.beeHA = ha;
        Bear [] ba = new Bear[5];
        for (int x=0; x < 5; x++) {
            ba[x] = new Bear();
            ba[x].hunny = honeyPot;
        }
        Kit k = new Kit();
        k.kh = honeyPot;
        Raccoon r = new Raccoon(); ← 新的Raccoon对象
        r.rh = honeyPot; ← 引用它的变量!
        r.k = k;
        k = null;
    } // main函数结束
}

```

下面的程序代码新建出数个对象。你的任务是要找出“最受欢迎”的对象，也就是被最多变量所引用的对象。还有，标记出所有对象以及对它的引用。我们已经先帮你标记出一个对象。



华安 传奇实录



“我们已经测了4次，主要模块的温度还是持续下降”，秋香不耐烦地说着，“上礼拜就已经安装新的温度感应器，散热器（radiator）的读数应该也没有问题，所以我们把焦点放在恒温器（retention bot）上”。华安叹了一口气，想起一开始的时候还以为纳米技术可以帮他们超前进度。现在离发射只剩下5个礼拜，卫星的生存维护装置还是没有办法通过测试。

“你用什么比例来模拟？”，华安问到。

“我知道你要问什么”，秋香回答，“我们也想到了。如果不符合规格，任务控制中心不会验收的。我们必须以2:1的比例执行v3版和v2版的radiator测试单元”，秋香继续说，“整体来说，retention与radiator的比例应该是4:3”。

华安追问：“能源消耗呢？”。秋香想了一下：“那是另外一问题，能源消耗速度比预期的快。我们有另外一组人正在想办法。但是这些无线技术很难将radiator的能源消耗与retention的消耗分离”。

秋香停了一下又继续说：“设计上的整体能源消耗率应该是3:2，也就是radiator的能源消耗会比较快”。

“好吧，既然这样”，华安说，“那我们再看一下仿真程序初始化的程序代码。这个问题一定得要很快地解决！”

```

import java.util.*;
class V2Radiator {
    V2Radiator(ArrayList list) {
        for(int x=0; x<5; x++) {
            list.add(new SimUnit("V2Radiator"));
        }
    }
}

class V3Radiator extends V2Radiator {
    V3Radiator(ArrayList llist) {
        super(llist);
        for(int g=0; g<10; g++) {
            llist.add(new SimUnit("V3Radiator"));
        }
    }
}

class RetentionBot {
    RetentionBot(ArrayList rlist) {
        rlist.add(new SimUnit("Retention"));
    }
}

```

华安 传奇实录 (续)

```

public class TestLifeSupportSim {
    public static void main(String [] args) {
        ArrayList aList = new ArrayList();
        V2Radiator v2 = new V2Radiator(aList);
        V3Radiator v3 = new V3Radiator(aList);
        for(int z=0; z<20; z++) {
            RetentionBot ret = new RetentionBot(aList);
        }
    }
}

class SimUnit {
    String botType;
    SimUnit(String type) {
        botType = type;
    }
    int powerUse() {
        if ("Retention".equals(botType)) {
            return 2;
        } else {
            return 4;
        }
    }
}

```

把程序看了一遍之后，华安露出了得意的笑容。他说：“我想我知道那是怎么回事了。能源消耗这件事情其实也是有关的！”

到底华安看出哪里有问题？那要如何解决这个bug呢？

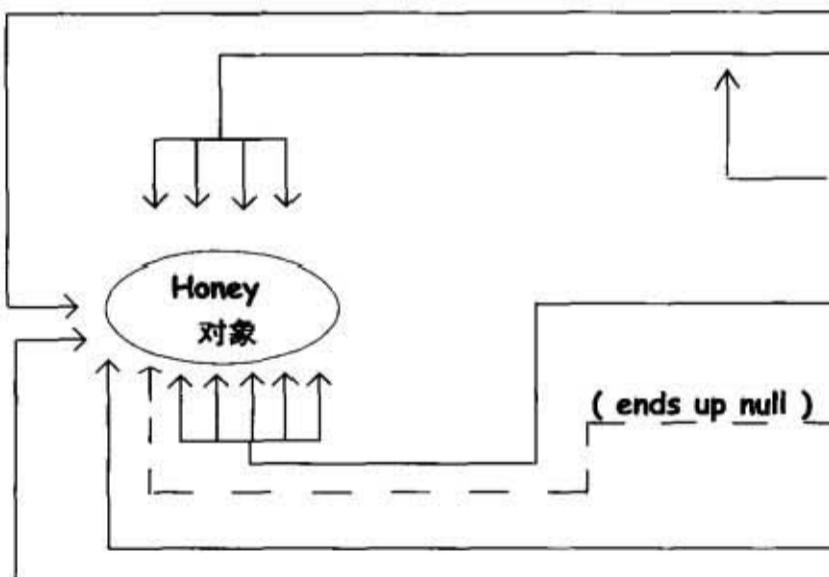


G.C.

- | | | |
|---|----------------|-------------------------------|
| 1 | copyGC = null; | 不行——这一行程序尝试要存取已经不在范围内
的变量。 |
| 2 | gc2 = null; | 可以—— gc2 是该对象唯一的引用。 |
| 3 | newGC = gc3; | 不行——也是超出范围。 |
| 4 | gc1 = null; | 可以—— gc1 是唯一的引用。 |
| 5 | newGC = null; | 不行—— newGC 已经超出范围。 |
| 6 | gc4 = null; | 不行——还有 gc3 引用该对象。 |
| 7 | gc3 = gc2; | 不行——还有 gc4 引用该对象。 |
| 8 | gc1 = gc4; | 可以——重新给对象引用赋值。 |
| 9 | gc3 = null; | 不行—— gc4 还在引用该对象。 |

最受欢迎 金对象奖

要指出Honey这个对象是这个类中最受欢迎的对象应该不难。但要看出这些变量都指向同一个对象其实是要动点脑筋的。



```

public class Honey {
    public static void main(String [] args) {
        Honey honeyPot = new Honey();
        Honey [] ha = {honeyPot, honeyPot,
                      honeyPot, honeyPot};
        Bees b1 = new Bees();
        b1.beeHA = ha;
        Bear [] ba = new Bear[5];
        for (int x=0; x < 5; x++) {
            ba[x] = new Bear();
            ba[x].hunny = honeyPot;
        }
        Kit k = new Kit();
        k.kh = honeyPot;
        Raccoon r = new Raccoon();
        r.rh = honeyPot;
        r.k = k;
        k = null;
    } } // main函数结束
  
```



华安 传奇实录

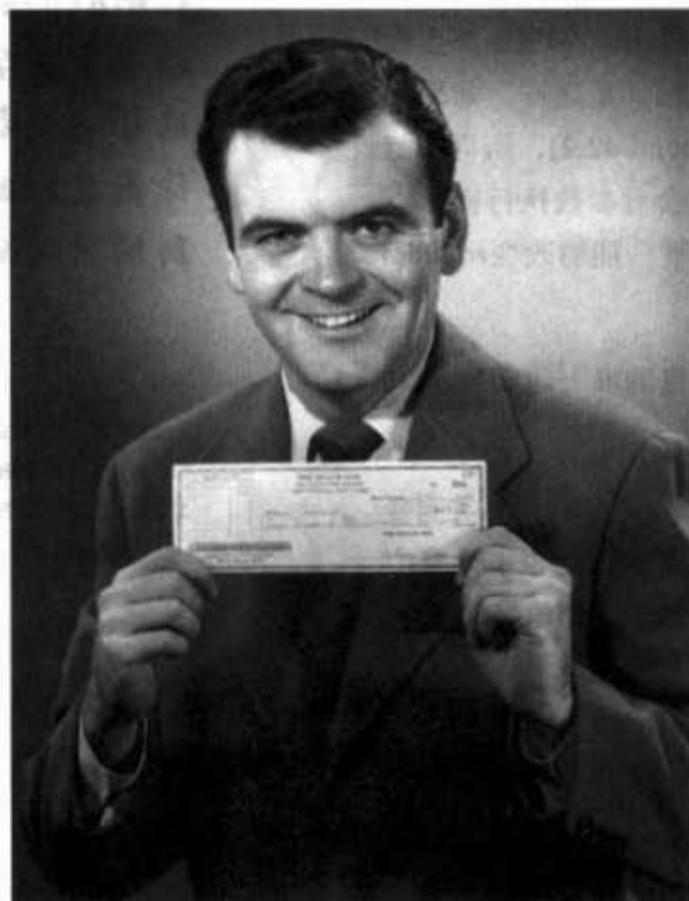
华安发现V2Radiator类的构造函数会取用一个ArrayList参数。这代表每次V3Radiator的构造函数被调用时，它会在对V2Radiator的super()调用中传入一个ArrayList。这样会额外多出5个V2Radiator的SimUnit。如此一来，总体能源消耗会是120而不是秋香预期的100。

因为每个Bot都会创建出SimUnit，所以在SimUnit的构造函数中加上一栏输出就能够很快的发现问题的来源！



10 数字与静态

数字很重要



盘算一下吧。除了primitive主数据类型的运算之外，数字还有其他的工作。你可能需要对数字计算绝对值、取整。或许需要以小数点后两位的打印格式，或者每隔三位数加上逗点以方便阅读。还有日期也是，你可能需要用某种特定的格式来打印日期，或者对日期作运算，比如说“今天起的两个礼拜后”。此外，字符串要如何转换成数字呢？幸好Java API中有很多与数字有关的方法能够很方便地使用。但这些方法多为静态的，因此我们会先从静态的变量和方法开始说起——这也包括了静态的final变量这种Java常数。

Math方法：

最接近全局的方法

虽然在Java中没有东西是全局（global）的。但可以这么想：一种方法的行为不依靠实例变量值。例如Math这个类中的round()方法。它永远都执行相同的工作——取出浮点数（方法的参数）的最接近整数值。永远都是这样。如果你有10000个Math的实例，并且都执行round(42.2)，所得的值永远都会是42。换句话说，这个方法会对参数执行操作，但这操作不受实例变量状态的影响。唯一能够改变round()行为的只有所传入的参数。

你不觉得为了要执行round()而得在宝贵的堆上建立Math的实例是很浪费的事吗？

像round()、abs()、max()等数学运算方法其实不需要实例变量值。事实上也不会有Math的实例变量。因此也不会有空间被它的实例所占用。你猜怎么样？你不需创建Math的实例，实际上你也无法创建。

如果你硬要创建Math的实例：

```
Math mathObject = new Math();
```

会得到下面这样的错误信息：

```
File Edit Window Help IwasToldThereWouldBeNoMath
%javac TestMath
TestMath.java:3: Math() has private access in java.lang.Math
    Math mathObject = new Math();
                           ^
1 error
```

错误信息指出Math的构造函数被标记为私有的！这代表你不能新建Math的对象

在**Math**这个类中的所有方法都不需要实例变量值。因为这些方法都是静态的，所以你无需**Math**的实例。你会用到的只有它的类本身。

```
int x = Math.round(42.2);
int y = Math.min(56,12);
int z = Math.abs(-343);
```



这些方法无需实例变量，因此也不需要特定对象来辨别行为。

非静态方法与静态方法的差别

Java是面向对象的，但若处于某种特殊的情况下，通常是实用方法，则不需要类的实例。static这个关键词可以标记出不需类实例的方法。一个静态的方法代表说“一种不依靠实例变量也就不需要对象的行为”。

非静态方法

```
public class Song {
    String title; ← 实例变量的值会影响到 play() 方法的行为
    public Song(String t) {
        title = t;
    }
    public void play() {
        SoundPlayer player = new SoundPlayer();
        player.playSound(title);
    }
}
```

Song
title
play()

有两个Song
的对象

↑
title的值会决定play()方法
所播放的内容



它会播放Politik
→ s2.play();

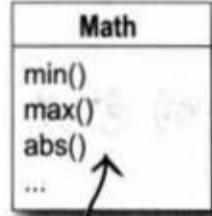
它会播放My Way (没听过
庞克的歌吗？那别说你知
道什么叫庞克)



s3.play();

静态方法

```
public static int min(int a, int b){
    // 返回a与b中较小的值
}
```



没有实例变量

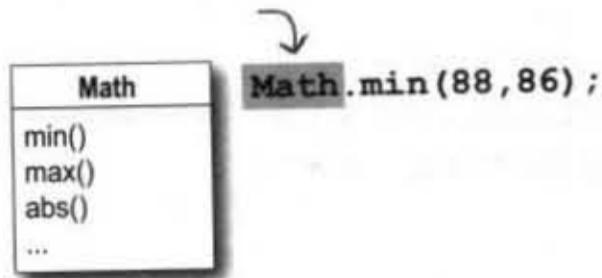
Math.min(42, 36);

直接用类的名字

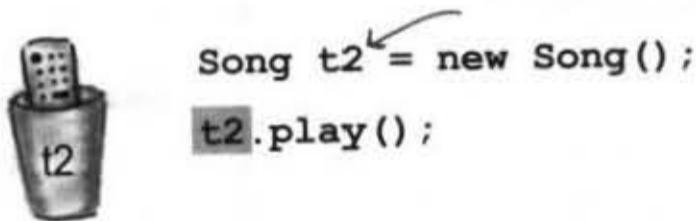


没有对象！绝对没有！

以类的名称调用静态的方法



以引用变量的名称调用非静态的方法



带有静态方法的含义

带有静态的方法的类通常（虽然不一定是这样）不打算要被初始化。在第8章我以已经讨论过抽象类，以及如何用`abstract`这个修饰字来标记类以让它不能被创建出实例。换句话说，抽象的类是不能被初始化的。

但你也可以用私有的构造函数来限制非抽象类被初始化。要记得，被标记为`private`的方法代表只能被同一类的程序所调用。构造函数也是同样意思，这也是`Math`如何防止被初始化的方法。它让构造函数标记为私有，所以你无法创建`Math`的实例。编译器会知道你不能存取这些私有的构造函数。

这并不是说有一个或多个静态的方法的类就不能被初始化。事实上，只要有`main()`的类都算有静态的方法！

通常你会写出`main()`来启动或测试其他的类。从`main()`中创建类的实例并调用新实例上的方法。

因此你可以任意地在类中组合静态与非静态的方法，然而任何非静态的方法都代表必须以某种实例来操作。取得新对象的方法只有通过`new`或者序列化（`deserialization`）以及我们不会讨论的Java Reflection API。除此之外，别无他法。实际上由谁来新建是一个很有意思的问题，稍后我们就会讨论这个部分。

静态的方法不能调用非静态的变量

静态的方法是在无关特定类的实例情况下执行的。如同你在上一页所看到的，甚至也不会有该类的实例出现。因为静态的方法是通过类的名称来调用，所以静态的方法无法引用到该类的任何实例变量。在此情况下，静态的方法也不会知道可以使用哪个实例变量值。

如果你尝试在静态的方法内使用实例变量，编译器会认为：“我不知道你说的是哪个实例的变量！”

静态的方法是不知道堆上有哪些实例的。

如果你要编译下面这段程序代码：

```
public class Duck {
    private int size;

    public static void main (String[] args) {
        System.out.println("Size of duck is " + size);
    }

    public void setSize(int s) {
        size = s;
    }

    public int getSize() {
        return size;
    }
}
```

哪一个Duck?



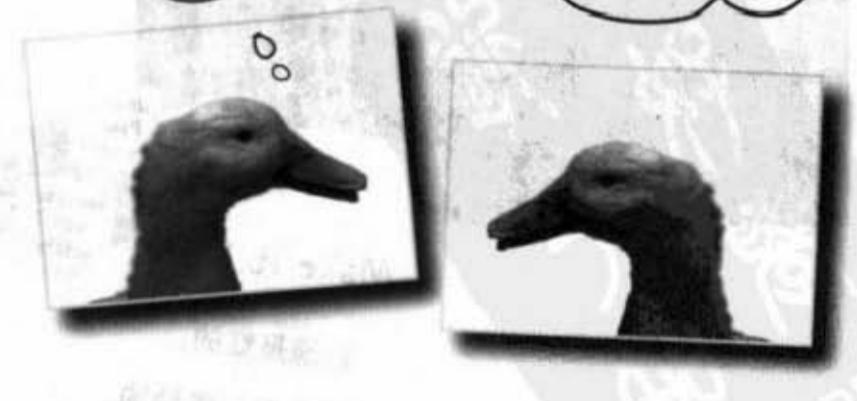
此时我们根本无从得知堆上是否有Duck

碧玉，我觉得他们是在说我的大小

不，逃兰，他们是在说我的大小

你会得到这样的错误：

```
File Edit Window Help Quack
% javac Duck.java
Duck.java:6: non-static variable
size cannot be referenced from a
static context
    System.out.println("Size
of duck is " + size);
^
```



静态的方法也不能调用非静态的方法

非静态的方法做什么工作？它们通常是以实例变量的状态来影响该方法的行为。getName()方法会返回name变量的值。谁的名字？当然是被调用对象的。

这一段无法编译：

```
public class Duck {
    private int size;

    public static void main (String[] args) {
        System.out.println("Size is " + getSize());
    }

    public void setSize(int s) {
        size = s;
    }
    public int getSize() {
        return size;
    }
}
```

调用 getSize() 会需要用到 size 实例变量

老问题，到底要谁的 size？

```
File Edit Window Help Jack-in
% javac Duck.java
Duck.java:6: non-static method
getSize() cannot be referenced
from a static context
    System.out.println("Size
of duck is " + getSize());
```



*there are no
Dumb Questions*

问： 如果从静态方法调用非静态方法，但此非静态方法没有用到实例变量，这样会通过编译吗？

答： 不会。编译器可以知道你有没有使用实例变量，你也知道。但如果现在可以通过，后来却把非静态变量改成会使用实例变量呢？又如果子类去覆盖这个方法成有用到实例变量的版本呢？

问： 我发誓曾经看过以引用变量代替类名称调用静态方法的程序代码，这样对吗？

答： 你确实可以这样做，但合法的事情并不一定都是好事。虽然可以使用类的实例来调用，但这样会产生容易误解的程序代码：

```
Duck d = new Duck();
String[] s = { };
d.main(s);
```

这段程序代码是合法的，但编译器还是会解析出原来的类。使用d来调用main()并不代表main会知道是哪个对象引用所做的调用。如此调用的方法也还是静态的！

静态变量： 它的值对所有的实例来说 都相同

假设你要在执行过程中计算有多少Duck的实例已经被建立出来。你要怎么做？或许可以在构造函数中递增某个实例变量的值？

```
class Duck {
    int duckCount = 0;
    public Duck() {
        duckCount++; 这会在创建Duck对象时
    } 执行递增
}
```

不行，因为duckCount是个实例变量，所以这样做不会成功。每个Duck在初始化的时候duckCount的值都是0。你也许可以调用别的类来计算，不过这样又不太优雅。你需要的是只会有一份拷贝的变量，且所有实例都会用到该拷贝。

这就是静态变量的功用：被同类的所有实例共享的变量。

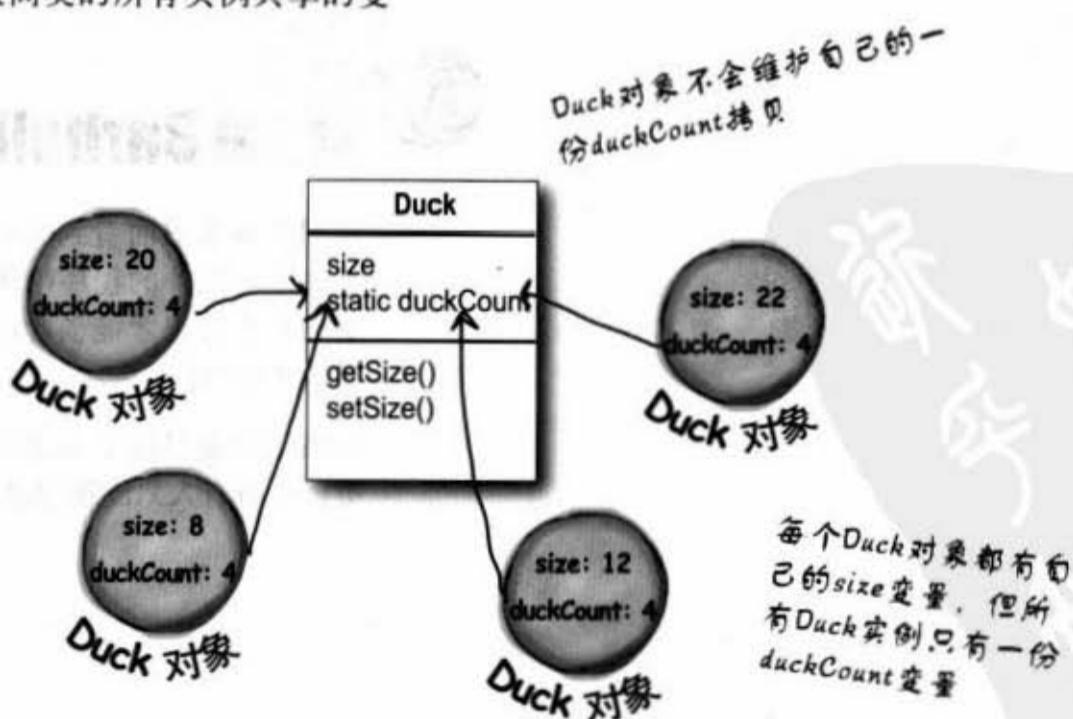
```
public class Duck {
    private int size;
    private static int duckCount = 0;

    public Duck() { 每当构造函数执行的时
        duckCount++; 候，此变量的值就会递
    }

    public void setSize(int s) {
        size = s;
    }

    public int getSize() {
        return size;
    }
}
```

此静态的duckCount变量只会在类第一次载入的时候被初始化。



静态变量



静态变量是共享的。

同一类所有的实例共享
一份静态变量。

实例变量：每个实例一个。

静态变量：每个类一个。



Brain Barbell

我们在本章前面看到私有的构造函数代表这个类不能被本身以外的程序给实例化。也就是说，只有此类的程序代码才能创建出新的实例（到底是鸡生蛋还是蛋生鸡？）。

如果你想要以这个方法编写出只能创建一个实例的类，且所有人只能运用这一份实例，要该怎么做？

静态变量的起始动作

静态变量是在类被加载时初始化的。类会被加载是因为Java虚拟机认为它该被加载了。通常，Java虚拟机会加载某个类是因为第一次有人尝试要创建该类的新实例，或是使用该类的静态方法或变量。程序员其实也可以选择强制Java虚拟机去加载某个类，但你不太需要这么做。大部分的情况下还是让Java虚拟机来决定会比较好。

静态项目的初始化有两项保证：

静态变量会在该类的任何对象创建之前就完成初始化。

静态变量会在该类的任何静态方法执行之前就初始化。

```
class Player {
    static int playerCount = 0;
    private String name;
    public Player(String n) {
        name = n;
        playerCount++;
    }
}
```

```
public class PlayerTestDrive {
    public static void main(String[] args) {
        System.out.println(Player.playerCount);
        Player one = new Player("Tiger Woods");
        System.out.println(Player.playerCount);
    }
}
```

如果你没有给静态变量赋初值，它就会被设定默认值。int会被设定为0。静态变量的默认值会是该变量类型的默认值，就像实例变量所被赋予的默认值一样。

playerCount会在载入类的时候被初始化为0。

像是long或short等primitive主数据类型整数的默认值是0，primitive主数据类型的浮点数默认值是0.0，boolean是false，对象引用是null

静态变量也是通过类的名称来存取



```
File Edit Window Help What?
% java PlayerTestDrive
0 ← 在实例创建之前
1 ← 对象创建之后
```

静态变量会在该类的任何静态方法执行之前就初始化。

静态的final变量是常数

一个被标记为final的变量代表它一旦被初始化之后就不会改动。也就是说类加载之后静态final变量就一直会维持原值。以Math.PI为例：

```
public static final double PI = 3.141592653589793;
```

此变量被标记为public，因此可供各方读取。

此变量被标记为static，所以你不需要Math的实例。

此变量被标记为final，因为圆周率是不变的。

此外没有别的方法可以识别变量为不变的常数（constant），但有命名惯例（naming convention）可以帮助你认出来。

常数变量的名称应该要都是大写字母！

静态初始化程序（static initializer）是一段在加载类时会执行的程序代码，它会在其他程序可以使用该类之前就执行，所以很适合放静态final变量的起始程序。

```
class Foo {
    final static int x;
    static {
        x = 42;
    }
}
```

静态final变量的初始化：

① 声明的时候：

```
public class Foo {
    public static final int FOO_X = 25;
}
```

注意这个命名惯例——
应该都是大写的，并以下划线字符分隔

或

② 在静态初始化程序中：

```
public class Bar {
    public static final double BAR_SIGN;
    > static {
        BAR_SIGN = (double) Math.random();
    }
}
```

这段程序会在类被加载时执行

如果你没有以这两种方式之一来给值的话：

```
public class Bar {
    public static final double BAR_SIGN;
}
```

没有初始化！

编译器会发现这个问题：

```
File Edit Window Help Jack-in
% javac Bar.java
Bar.java:1: variable BAR_SIGN
might not have been initialized
1 error
```

final 不只用在静态变量上……

你也可以用final关键字来修饰非静态的变量，这包括了实例变量、局部变量甚或是方法的参数。不管哪一种，这都代表它的值不能变动。但你也可以用final来防止方法的覆盖或创建子类。

非静态final变量

```
class Foof {
    final int size = 3; ← size将无法改变
    final int whuffle;

    Foof() {
        whuffle = 42; ← whuffle不能改变
    }

    void doStuff(final int x) {
        // 不能改变x
    }

    void doMore() {
        final int z = 7;
        // 不能改变z
    }
}
```

final 的 method

```
class Poof {
    final void calcWhuffle() {
        // 绝对不能被覆盖过
    }
}
```

final 的 class

```
final class MyMostPerfectClass {
    // 不能被继承过
}
```

final的变量代表你不能改变它的值。

final的方法代表你不能覆盖掉该method。

final的类代表你不能继承该类（也就是创建它的子类）。



完了，一切都final了！做什么都没用了！

*there are no
Dumb Questions*

要点

- 静态的方法应该用类的名称来调用，而不是用对象引用变量。
- 静态的方法可以直接调用而不需要堆上的实例。
- 静态的方法是一个非常实用的方法，它不需特别的实例变量值。
- 静态的方法不能存取非静态的方法。
- 如果类只有静态的方法，你可以将构造函数标记为private的以避免被初始化。
- 静态变量为该变量所属类的成员所共享。静态变量只会有一份，而不是每个实例都有自己的一份。
- 静态方法可以存取静态变量。
- 在Java中的常量是把变量同时标记为static和final的。
- final的静态变量值必须在声明或静态初始化程序中赋值：

```
static {
    DOG_CODE = 420;
}
```

- 常量的命名惯例是全部使用大写字母。
- final值一旦被赋值就不能更改。
- final的方法不能被覆盖。
- final的类不能被继承。

问： 静态的方法不能存取非静态的变量，但非静态的方法可以读取静态的变量吗？

答： 当然可以。非静态方法不可以调用该类静态的方法或静态的变量。

问： 为何需要将类标记为final？这不会破坏面向对象的目的吗？

答： 会也不会。将类标记为final的主要目的是为了安全。例如String这个类，假使有人继承过，弄了一个行为很不一致的版本，就会对预期操作String的程序产生很多问题。

问： 如果类已经是final的，再标记final的方法是不是很多余？

答： 不只是多余，而且多了很多。如果一个类不能被子类化，则它的方法根本就无法被覆盖。如果只是想要限制部分的方法不能被覆盖过，那就单独地标记它们为final的就行。



谁是合法的？

就你所学到的static和final知识来看，下面哪些程序可以通过编译？



- public class Foo {

 static int x;

 public void go() {
 System.out.println(x);
 }
 }
- public class Foo2 {

 int x;

 public static void go() {
 System.out.println(x);
 }
 }
- public class Foo3 {

 final int x;

 public void go() {
 System.out.println(x);
 }
 }
- public class Foo4 {

 static final int x = 12;

 public void go() {
 System.out.println(x);
 }
 }
- public class Foo5 {

 static final int x = 12;

 public void go(final int x) {
 System.out.println(x);
 }
 }
- public class Foo6 {

 int x = 12;

 public static void go(final int x) {
 System.out.println(x);
 }
 }

Math 的方法

现在我们已经知道 static 的方法是如何工作的，接着让我们来看一下 Math 的一些方法。这不是全部，API 文件还有像是 sqrt()、tan()、ceil() 等的说明。

Math.random()

返回介于 0.0 ~ 1.0 之间的双精度浮点数。

```
double r1 = Math.random();
int r2 = (int) (Math.random() * 5);
```

Math.abs()

返回双精度浮点数类型参数的绝对值。这个方法有覆盖的版本，因此传入整型会返回整型，传入双精度浮点数会返回双精度浮点数。

```
int x = Math.abs(-240); // 返回240
double d = Math.abs(240.45); // 返回240.45
```

Math.round()

根据参数是浮点型或双精度浮点数返回四舍五入之后的整型或长整型值。

```
int x = Math.round(-24.8f); // 返回-25
int y = Math.round(24.45f); // 返回24
```

↑ 文字直接表示的浮点数都会被当作双精度浮点数，除非后面有加上 f

Math.min()

返回两参数中较小的那个。这有 int、long、float 或 double 的覆盖版本。

```
int x = Math.min(24, 240); // 返回24
double y = Math.min(90876.5, 90876.49); // 返回90876.49
```

Math.max()

返回两参数中较大的那个。这有 int、long、float 或 double 的重载版本。

```
int x = Math.max(24, 240); // 返回240
double y = Math.max(90876.5, 90876.49); // 返回90876.5
```

primitive主数据类型的包装

有时你会想要把primitive主数据类型当作对象来处理。例如在5.0之前的Java版本上，你无法直接把primitive主数据类型放进ArrayList或HashMap中：

```
int x = 32;
ArrayList list = new ArrayList();
list.add(x);
```

除非是用Java 5.0或以上的版本，否则这个命令不会成功

每一个primitive主数据类型都有个包装用的类，且因为这些包装类都在java.lang这个包中，所以你无需去import它们。每个包装类都很好辨别，因为它的名称是照着所包装的类型所设定的，只是将第一个字母改为大写以符合命名惯例。

还有，为了某些没有人知道的理由，API的设计者决定让名称不是完全地符合primitive主数据类型的名字：

Boolean	
Character	
Byte	
Short	
Integer	
Long	
Float	
Double	

注意！这个名称与primitive主数据类型不同，是完整拼出来的

传入primitive主数据类型给包装类的构造函数

包装值

```
int i = 288;
Integer iWrap = new Integer(i);
```

所有的包装类都有类似
这样的方法

解开包装

```
int unWrapped = iWrap.intValue();
```



primitive主数据类型

当你需要以对象方式来
处理primitive主数据类型
时，就把它包装起来。

Java 5.0之前的版本必须
要这么做。



*上面的图听说是个叉烧包



这真的很蠢。你说我不能直接做
出int的ArrayList? 还得要把每个int包
装成Integer对象才行? 既浪费时间又容
易出错……

在Java 5.0以前你得这样做……

没错。在Java 5.0之前的版本上，primitive主数据类型就是原始类型，而对象引用就是对象引用，两者绝无交换使用的方法。要交互使用就得靠程序员进行包装与拆开包装的动作。没有办法能够以primitive主数据类型来直接传入期待对象引用的方法，也没有办法能够把回传对象参考的method值赋值给primitive主数据变量。Integer与int两者间毫无关系可言，只是Integer带有一个int类型的实例变量（以保存Integer所包装的primitive）。你得想办法自己进行这一类转换的工作。

primitive int的ArrayList

无autoboxing

```
public void doNumsOldWay() {
    ArrayList listOfNumbers = new ArrayList();
    listOfNumbers.add(new Integer(3));
    Integer one = (Integer) listOfNumbers.get(0);
    int intOne = one.intValue();
}
```

↑
最后再取出 primitive

创建出ArrayList对象

不能直接加入primitive的3，得先
转换成Integer

返回Object类
型，但你可以
将Object转换成
Integer

autoboxing：不必把primitive主数据类型与对象分得那么清楚

从5.0版开始加入的autoboxing功能能够自动地将primitive主数据类型转换成包装过的对象！

让我们看一下创建int的ArrayList时会发生什么事。

primitive int的ArrayList

有autoboxing

```
public void doNumsNewWay() {
    ArrayList<Integer> listOfNumbers = new ArrayList<Integer>();
    listOfNumbers.add(3);
    int num = listOfNumbers.get(0);
}
```

编译器也会自动地解开Integer对象的包装，因此可以直接赋值给int。

创建Integer类型的ArrayList

虽然ArrayList没有add(int)这样的方法，但编译器会自动帮你包装。

问：为什么不直接声明 `ArrayList<int>`？

答：generic类型的规则是你只能指定类或接口类型。因此`ArrayList<int>`将无法通过编译。但你可以直接把这个包装所对应的primitive主数据类型放进`ArrayList`中，比如说`boolean`类型的放入`ArrayList<Boolean>`中`char`类型的放入`ArrayList<Character>`中。

静态的方法

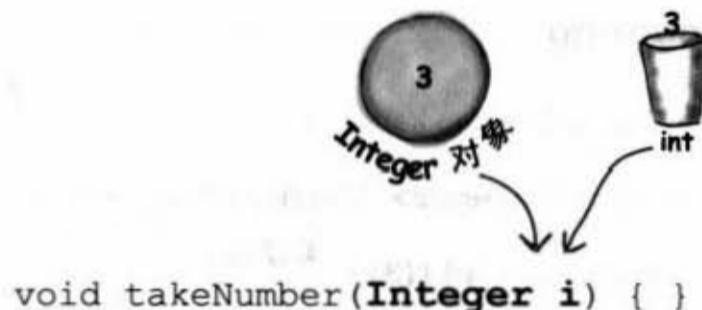
到处都用得到autoboxing

autoboxing不只是包装与解开primitive主数据类型给collection用而已，它还可以让你在各种地方交换地运用primitive主数据类型与它的包装类型。想想看有哪些地方会用到。

autoboxing乐趣多

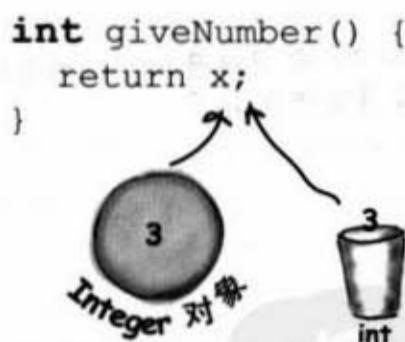
方法的参数

如果参数是某种包装类型，你可以传入相对应的primitive主数据类型，反之亦然。



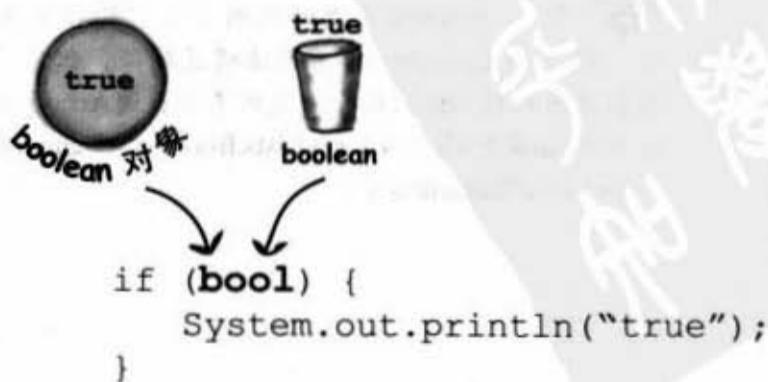
返回值

如果method声明为返回某种primitive主数据类型，你也可以返回兼容的primitive主数据类型或该primitive主数据类型的包装类型。



boolean 表达式

任何预期boolean值的位置都可以用求出boolean的表达式来代替，比如说`4>2`或是Boolean包装类型的引用。



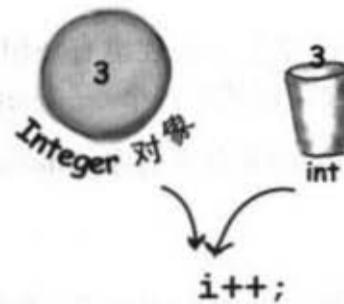
数值运算

这或许是最诡异的。你可以在使用 primitive 主数据类型作为运算子的操作中以包装类型来替换。这代表你可以对 Integer 的对象作递增运算！

```
Integer i = new Integer(42);
i++;
```

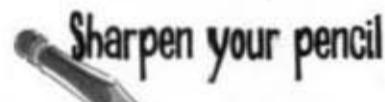
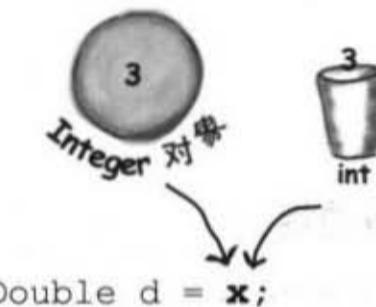
还可以这么做：

```
Integer j = new Integer(5);
Integer k = j + 3;
```



赋值

你可以将包装类型或 primitive 主数据类型赋给声明成相对应的包装或 primitive 主数据类型。



```
public class TestBox {
    Integer i;
    int j;

    public static void main (String[] args) {
        TestBox t = new TestBox();
        t.go();
    }

    public void go() {
        j=i;
        System.out.println(j);
        System.out.println(i);
    }
}
```

右边的程序代码能否通过编译？可以执行吗？如果可以，会有什么结果？

慢慢来，你有足够的时间去想，这会引起我们没讨论过的autoboxing问题。

你得要编译才会有答案，所以我们是在逼你上机操作。来吧！

包装类型的方法

等一下！还有呢！包装也有静态的实用性方法！

除了一般类的操作外，包装也有一组实用的静态方法。

我们在之前已经用过其中一个——Integer.parseInt()。

这个方法取用String并返回给你primitive主数据类型值。

将String转换成primitive主数据

类型值是很容易的：

将“2”解析成2

```
String s = "2";
int x = Integer.parseInt(s);
double d = Double.parseDouble("420.24");

boolean b = new Boolean("true").booleanValue();
```

你可能会以为有Boolean.parseBoolean()吧？其实没有。但是Boolean的构造函数可以用String来创建对象

但若你这么做的话：

```
String t = "two";
int y = Integer.parseInt(t);
```

啊！可以通过编译，但执行时就会出状况

就会在运行期间遇到异常：

```
File Edit Window Help Clue
% java Wrappers
Exception in thread "main"
java.lang.NumberFormatException: two
at java.lang.Integer.parseInt(Integer.java:409)
at java.lang.Integer.parseInt(Integer.java:458)
at Wrappers.main(Wrappers.java:9)
```

解析String的方法或构造函数会抛出NumberFormatException异常。这是运行期间的异常，你应该会处理这种异常。

(下一章会讨论异常)

反过来将 primitive 主数据类型 值转换成 String

有好几种方法可以将数值转换成 String。最简单的方法是将数字接上现有的 String。

```
double d = 42.5;
String doubleString = "" + d;    // " + " 这个操作数是 Java 中唯一有重载过的运算符
```

```
double d = 42.5;
String doubleString = Double.toString(d);
```

Double 这个类的静态方法

如果我要以某种特定的格式列出数字怎么办？比如说 \$56.87 这样……

有没有像 C 语言中的 printf 这种格式化功能呢？



数字的格式化

在Java中，数字与日期的格式化功能并没有结合在输出/输入功能上。通常对用户显示数字是通过GUI来进行的。你会把String放在可滚动的文字区域块或表格中。如果格式化功能只有绑在文字模式输出的命令上，那就没有办法把字符串以比较漂亮的格式输出到GUI上。在Java 5.0之前的格式化功能是通过java.text这个包来处理，但本书已经不屑去提它了。

从Java 5.0起，更多更好更有扩展性的功能是通过java.util中的Formatter这个类来提供的。但你无需自己创建与调用这个class上的方法，因为Java 5.0已经把便利性的功能加到部分的输出/输入类与String上。因此只要调用静态的String.format()并传入值与格式设定就好。

当然，你还是得知道如何提供格式设定，本章会有一些基本的说明，我们会从基本的范例开始，并观察它们是如何运行的（在讨论输出/输入的章节中还会再看过一次）。

将数字以带逗号的形式格式化

```
public class TestFormats {
    public static void main (String[] args) {
        String s = String.format("%, d", 1000000000);
        System.out.println(s);
    }
}
```

↑
有逗号的数字格式

要格式化的数字

↑
格式设定，用来指示应该用哪种形式来输出：这里的逗号是表示数字要以逗号来分隔，并不是说这里有%与d两项参数，千万别弄错了！

解析格式化结构……

基本上来说，格式化由两个主要部分组成（不只是这样，但我们先从简单开始）：

● 格式指令

描述要输出的特殊格式。

● 要格式化的值

不是所有东西都能被格式化。例如，如果你的格式指令适用于浮点数，则你就不能传入 Dog 或看起来很像浮点数的 String。

如果你本就已经熟悉 C/C++ 的
printf()，那这几页说的全部都是
废话！

以这种格式…… 来表现这个值

```
format("%, d", 1000000000);
```

↑

用这个格式将这个参数格式化

这个指令代表什么？

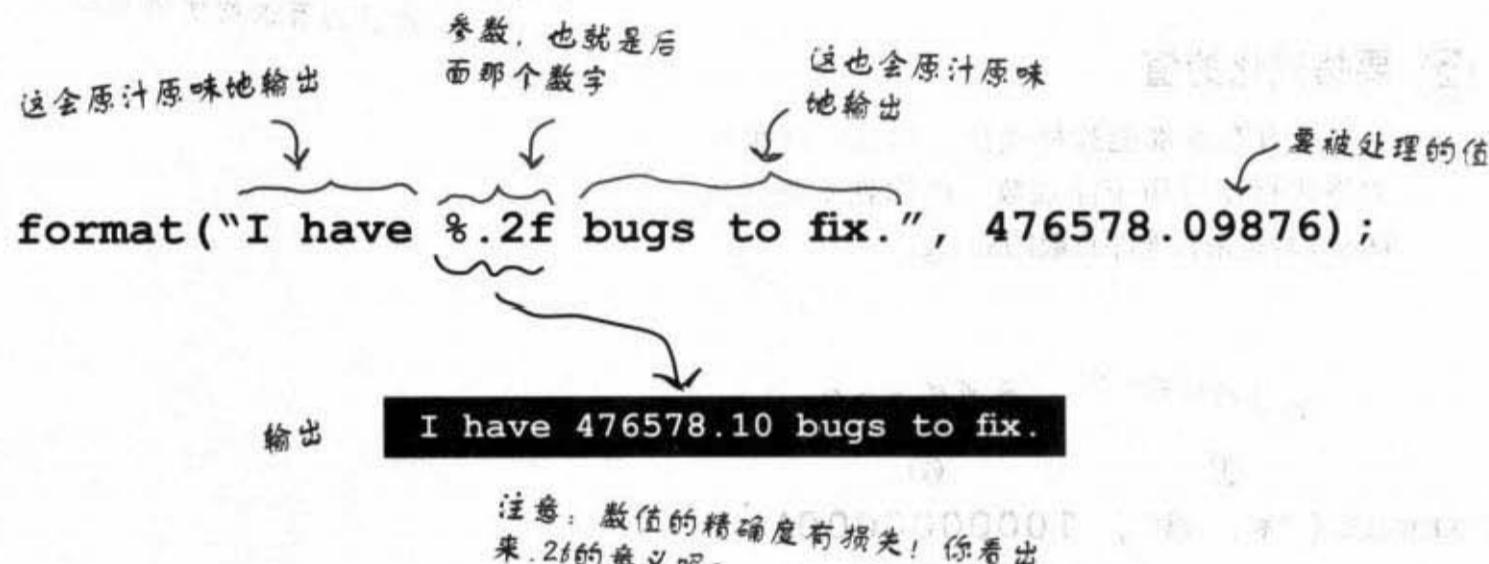
将此方法的第二个参数以第一个参数所表示带有逗号的整数（decimal）方式表示。

它会怎么做？

下一页会对“%， d”作更详细的说明，我们在这里先简略地说明一下。在格式化指令中的%代表一项变量，此变量就是跟在格式化指令后面的参数，其余的字符各有所代表的意义。

%符号代表把参数放在这里

format()方法的第一个参数被称为“格式化串”，它可以带有实际上就是要这么输出而不用转译的字符。当你看到%符号时，要把它想做是会被方法其余参数替换掉的位置。



此例的%符号是第二个参数会放置的位置，“.2f”代表该参数要使用的格式，其他的字都会以原来的方式输出。

加上逗号

```
format("I have %,.2f bugs to fix.", 476578.09876);
```

I have 476,578.10 bugs to fix.

↑
%符号后面的指令被加上逗号之后，
输出也有了变化



格式化语句有自己的一套语法

很明显，%符号后面不可以随便填上任意的字符。%的语法有非常特殊的规则，是用来描述此处所用的格式。

你已经看过两个例子：

%,d：这代表以十进制整数带有逗号的方式来表示。

%.2f：这代表以小数点后两位的方式来格式化此浮点数。

%,.2f：代表整数部分以有逗号的形式表示，小数部分以两位来格式化。

所以问题在于你怎么知道什么字符代表什么意义，以及这些指令字符的使用顺序和时机。

想想看下面这个语句会做出什么样的输出？答案在下一页：

```
String.format("I have %.2f, bugs to fix.", 476578.09876);
```

“格式化说明”的格式

跟在百分号后而包括类型指示（像是d或f）的每个东西都是格式化指令。除非遇到新的百分号，在类型指示之后的一组字符，格式化程序会假设都是直接输出的字符串。这可能吗？要被格式化的参数可以超过一个以上吗？先别管这个，稍后再讨论。现在先来看格式化说明的语法——跟在%后面的那些指令。

格式化说明最多会有5个部分（不包括%符号）。下面的[]符号里面都是选择性的项目，因此只有%与type是必要的。格式化说明的顺序是有规定的，必须要以这个顺序来指定。

% [argument number] [flags] [width] [.precision] **type**

如果要格式化的参数超过一个以上，可以在这些里指定是哪一个；我们稍后会讨论这部分

指定类型的特定选项，例如数字要加逗号或正负号

最小的字符数，注意：这不是总数；输出可以超过此宽度，若不足则会主动补零

精确度，注意前面有个圆点符号
一定是要指定的类型标识

% [argument number] [flags] [width] [.precision] **type**

format("%,6.1f", 42.000);

除了没有argument number之外，其他的项目都用到。

唯一的必填项目是类型

虽然类型是唯一必填的项目，但若指定别的项目，则类型必须是最后一项！类型修饰符有十几种（这不包括日期和时间的，它们有自己的一组），但大部分时间你会使用到%d或%f。且通常你会对%f加上精确度指示来设定所需要的小数长度。

The TYPE is mandatory, everything else is optional.

%d decimal
format("%d", 42);


参数必须能够与 int 相容。

%f floating point
format("%.3f", 42.000000);


参数必须是浮点数类型。

%x hexadecimal
format("%x", 42);


参数必须是byte、short、int、long、BigInteger。

%c character
format("%c", 42);
ASCII的42代表“*”号


参数同上，但不包括BigInteger。

在格式化指令中一定要给的类型，如果还要指定其他项目的话，要把类型摆在最后。

超过一项以上的参数时呢？

如果你要输出像下面这样的字符串：

“The rank is 20,456,654 out of 100,567,890.24.”

但这两个数字来自于不同的变量，该怎么做？把新的参数加到后面，因此你会以3个参数来调用format()而不是两个。并且在第一个参数中，也就是格式化串中，会有两个不同的格式化设定，也就是两个%开头的字符组合。第二个参数会应用在第一个%号上面，第三个参数会用在第二组%组合上。也就是说参数会依照顺序应用在%上面。

```
int one = 20456654;
double two = 100567890.248907;
String s = String.format("The rank is %,d out of %,.2f", one, two);
```

The rank is 20,456,654 out of 100,567,890.25

一项以上的参数会依序
对应到格式化设定

两项都有加逗号，且后者
的小数被限制在两位

当我们讨论到日期的格式化时，你就会看到以同一个参数应用在不同的格式化设定上。这可能有点难以理解，除非你直接看到日期格式化的例子。接下来我们就会讨论如何明确地指定哪个格式化设定要用在哪个参数上。

问：有些地方真的怪怪的，到底可以传多少个参数进去？我是说format()到底有多少个重载的版本？如果有10个参数要用在格式化上面会怎样？

答：问得好！没错，这里有些不一样的东西，且实际上是没有一大堆的重载版format()来取用不同数目排列组合的参数。为了要应付格式化的API，Java语言需要一种新的功能——称为可变参数列表（variable argument list，简称为vararg）。这个部分在附录中有说明，通常你很少会用到这样的功能。

都与数字有关，那日期呢？

如果你要输出这样的字符串：Sunday, Nov 28 2004

看起来好像没什么特别。但是如果说是要把Date类型的变量日期用这样的格式输出呢？Date类型是Java上表示时间用的，而现在你得处理这个类型。

数值与日期时间格式化的主要差别在于日期格式的类型是用“t”开头的两个字符来表示，下面有几个范例：

完整的日期与时间：%tc

```
String.format("%tc", new Date());
Sun Nov 28 14:52:41 MST 2004
```

只有时间：%tr

```
String.format("%tr", new Date());
03:01:47 PM
```

周、月、日：%tA %tB %td

因为没有刚好符合我们要求的输出格式，所以得组合3种形式来产生出所需要的格式：

```
Date today = new Date();
String.format("%tA, %tB %td", today, today, today);
```

这样就得把Date对象传
进去3次

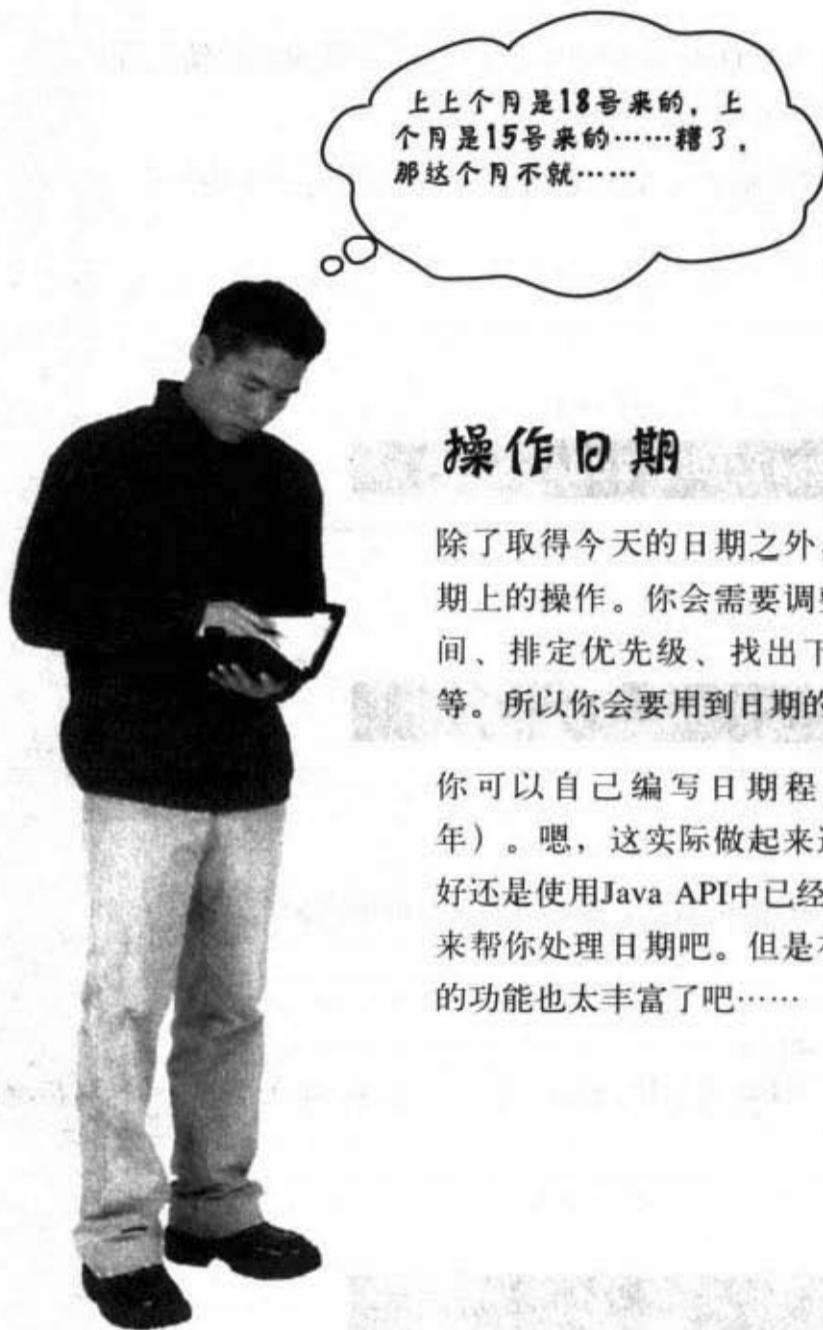
这里的逗号是直接输出的

```
Sunday, November 28
```

同上，但不用重复给参数

```
Date today = new Date();
String.format("%tA, %<tB %<td", today);
```

“<”这个符号是个特殊的指示，用来告诉格式化程序重复利用之前用过的参数



操作日期

除了取得今天的日期之外，你还会遇到很多日期上的操作。你会需要调整日期、计算花费时间、排定优先级、找出下一周期的开始时间等。所以你会需要用到日期的高级操作功能。

你可以自己编写日期程序（别忘记处理闰年）。嗯，这实际做起来还蛮复杂的。所以最好还是使用Java API中已经写好、功能丰富的类来帮你处理日期吧。但是有时你会发现这些类的功能也太丰富了吧……

在时光中前后移动

假如说中国厨艺学院的课程表是从周一到周五。你被十八铜人指定要查出今年每个月最后的上课日……

看来java.util.Date没什么用处

之前我们使用java.util.Date来查询今天的日期，因此从这个类开始寻找适用的功能是很合理的，但当你检查API文件时，你会发现有很多Date的功能被停用了！

但这个类还是很适合用来取得目前的时间。

好消息是API建议到java.util.Calendar上面去寻找其余的功能，因此我们要说：

就用java.util.Calendar来操作日期吧！

Calendar这个API的设计者打算要做全球化的思考。基本的想法是当你要操作日期时，你会要求一个Calendar（通过下一页会讨论的一个静态方法），然后Java虚拟机会赏你一个Calendar的子类实例（实际上Calendar是个抽象的类，所以你能用到的是它的具体子类）。

有意思的是，你所取得的Calendar是符合所在地区（locale）特性的。通常大部分的地区适用公历，但你也有可能处于使用农历或其他特殊格式的情况，此时你可以让Java函数库来处理这一类的日期。

标准的Java API带有java.util.GregorianCalendar，因此我们在书上使用这个历法。通常你也不需要担心你是在使用哪一种Calendar的sub-class，只要专注于Calendar的方法就可以了。

要取得当前的日期时间就用Date，其余功能可以从Calendar上面找。

上一页的老兄是在看信用卡的账单日期……

取得继承过Calendar的对象

你要如何取得抽象类的“实例”呢？当然不行，下面这个例子根本无法运行。

这个不行：

```
Calendar cal = new Calendar();
```

要用这个静态的方法：

```
Calendar cal = Calendar.getInstance();
```

无法通过编译！

这个语法看起来很熟悉——是个对静态方法的调用

等一下！如果不能创建
Calendar的实例，那Calendar
的引用到底引用了谁？



你无法取得Calendar的实例，但是你可以取得它的具体子类的实例

很明显，你不能取得Calendar的实例，因为Calendar是抽象的。但你还是能够不受限制地调用Calendar的静态method，这是因为静态的方法是在类上而不是在某个特定的实例上。所以你对Calendar调用getInstance()会返回给你具体子类的实例。那是某种继承过Calendar（也就是Calendar的多态变化版本）并且会依据合约来响应Calendar应有的方法。

大部分的Java版本都会默认返回一个java.util.GregorianCalendar的实例。

运用Calendar对象

要运用Calendar对象你得先了解几个关键的概念：

- 字段会保存状态——Calendar对象使用许多字段来表示某些事物的最终状态，也就是日期和时间。比如说你可以读取和设定它的year或month字段。
- 日期和时间可以运算——Calendar的方法能够让你对不同的字段作加法或减法的运算，比如说对month字段加一个月或对year减去三年。
- 日期与时间可以用millisecond来表示——Calendar可以让你将日期转换成微秒的表示法，或将微秒转换成日期。（更精确的说法是相对于1970年1月1日的微秒数），因此你可以执行精确的相对时间计算。

运用Calendar对象的范例：

```

Calendar c = Calendar.getInstance();
c.set(2004,1,7,15,40);           ← 将时间设定为2004年1月7日
long day1 = c.getTimeInMillis();   ← 将当前时间转换为以
                                    millisecond表示
day1 += 1000 * 60 * 60;
c.setTimeInMillis(day1);          ← 将c的时间加上一个半小时
System.out.println("new hour " + c.get(c.HOUR_OF_DAY));
c.add(c.DATE, 35);               ← 加上35天，所以c已经
                                    到了2月
System.out.println("add 35 days " + c.getTime());
c.roll(c.DATE, 35);              ← 滚动35天，注意：只有日期字
                                    段会动，月份不会动
System.out.println("roll 35 days " + c.getTime());
c.set(c.DATE, 1);                ← 直接设定DATE的值
System.out.println("set to 1 " + c.getTime());

```

```

File Edit Window Help Time-Files
new hour 16
add 35 days Wed Feb 11 16:40:41 MST 2004
roll 35 days Tue Feb 17 16:40:41 MST 2004
set to 1 Sun Feb 01 16:40:41 MST 2004

```

输出结果

Calendar API的精华

这里只有列出Calendar部分的字段和方法。它是相当大的API，因此这里只能列出常用的部分，一旦你熟悉它的操作之后，其余的部分也会很容易使用。

重要的方法

add(int field, int amount)

加减时间值

get(int field)

取出指定字段的值

getInstance()

返回Calendar，可指定地区

getTimeInMillis()

以毫秒返回时间

roll(int field, boolean up)

加减时间值，不进位

set(int field, int value)

设定指定字段的值

set(year, month, day, hour, minute)

设定完整的时间

setTimeInMillis(long millis)

以毫秒指定时间

// 不只这些……

关键字段

DATE / DAY_OF_MONTH

每月的几号

HOUR / HOUR_OF_DAY

小时

MILLISECOND

毫秒

MINUTE

分钟

MONTH

月份

YEAR

年份

ZONE_OFFSET

时区位移

// 还有更多……

静到最高点！静态的import

这是Java 5.0的新功能：一把双刃剑。有些人很喜欢这个主意，有些人恨死它了。如果你讨厌多打几个字，那你会喜欢这项功能。但它的缺点是会让程序比较难阅读。

基本上，这功能是让你import静态的类、变量或enum（稍后介绍）时能够少打几个字。

**警告：这会让你的
程序更易混淆**

旧式的写法：

```
import java.lang.Math;
class NoStatic {
    public static void main(String [] args) {
        System.out.println("sqrt" + Math.sqrt(2.0));
        System.out.println("tan" + Math.tan(60));
    }
}
```

静态import的语法

使用static import的写法：

```
import static java.lang.System.out;
import static java.lang.Math.*;
class WithStatic {
    public static void main(String [] args) {
        out.println("sqrt" + sqrt(2.0));
        out.println("tan" + tan(60));
    }
}
```

只不过是少打一些字

一 时机与要领

- 如果只会用到一两次，不如不用静态的import，这样程序会比较好阅读。
- 如果会用到很多次，或许用static的import会让程序看起来比较清爽。
- 在静态import的声明中也可以使用.*这样的通用字符。
- 最重要的问题是很容易产生名称的冲突。例如add()到底是要调用哪个的方法？



今晚的话题： 实例变量对静态变量的卑劣指控

实例变量

我不明白为什么要谈这些事情。每个人都知道静态变量只是用来当常数而已。又没有多少常数要用，我猜整个API大约只有四五个吧？真的用过的人也没几个吧。

对啦，全部都是。不过每个人都知道Swing只是个特殊的例子。

是呀，除了少数几个GUI之外，你还能举出任何一个真的大家都在用的静态变量吗？

呃……这也算特殊的例子呀。何况大家只是用它来除错而已。

静态变量

老兄，没有常识也应该要多看电视。你一定很少看API吧？里面有一堆的静态变量呢。甚至有一整个类是专门用来放常数值的。比如说 SwingConstants就全部都是。

或许它是个特例，但也是个很重要的特例啊！还有Color这个类，如果你记住所有颜色的RGB值那会有多恐怖？所以它已经定义好了红橙黄绿等颜色，用起来很方便的。

例如System.out，它是System这个类的一个静态变量。你不必自己创建出System的实例，只要从该class调用out变量就行。

怎样？除错不重要吗？你的豆腐脑一定不知道静态变量比较有效率，共享的类会省下很多内存的！

实例变量

嗯……你该不会忘记了吧？

静态变量不是面向对象的！我们干脆回到古代用算盘来执行计算的工作好了。

就像全局变量一样，许多潇洒帅气的程序员都知道那通常是很负面的事情。

这样就不叫面向对象程序设计了，这叫笨蛋。
你真的是老古董啊。

对啦，偶尔用一下静态变量还算合理，但是我要告诉你，滥用静态变量和方法就是不成熟面向对象程序员的招牌。程序员应该想的是对象的状态而不是类的状态。

这代表程序员还在用程序化的思维想事情，而不是依据对象的状态来进行处理。

是是是，就你自己需要嘛！

静态变量

什么？

别乱扣帽子，不是面向对象怎么了？那是什么？

对不起，我并不是全局变量。根本没有全局变量，我是在面向对象化的类中的！我是对象的自然状态，唯一的差别是我被很有效率地共享。

够了！闭嘴！这不是真的。某些静态变量对系统是非常关键的，不然至少也能够提供便利性。

你说什么？这又跟静态的方法扯上什么关系？

当然，我同意面向对象设计应该要专注于对象，有些问题只是因为菜鸟的因素……当你真的有需要的时候，没有东西能够代替静态。



```
class StaticSuper{
    static {
        System.out.println("super static block");
    }
}
```

```
StaticSuper{
    System.out.println(
        "super constructor");
}
```

```
public class StaticTests extends StaticSuper {
    static int rand;

    static {
        rand = (int) (Math.random() * 6);
        System.out.println("static block " + rand);
    }
}
```

```
StaticTests() {
    System.out.println("constructor");
}
```

```
public static void main(String [] args) {
    System.out.println("in main");
    StaticTests st = new StaticTests();
}
```

我是编译器

这一页的Java程序代码代表一份完整的程序。你的任务是要扮演编译器角色并判断程序输出会是哪一个？



哪个才是它的输出？

可能输出：

```
File Edit Window Help Cling
%java StaticTests
static block 4
in main
super static block
super constructor
constructor
```

可能输出：

```
File Edit Window Help Electricity
%java StaticTests
super static block
static block 3
in main
super constructor
constructor
```



这一章讨论 Java 的静态世界。你的任务是辨别下面的陈述哪些是对的、哪些是错的？

是非题

- (1) 使用Math类的第一个步骤是创建出它的实例。
- (2) 构造函数可以标记为静态的。
- (3) 静态的方法不能存取“this”所引用的对象。
- (4) 最好通过引用变量来调用静态的方法。
- (5) 静态变量可以用来计算类的实例数量。
- (6) 构造函数是在静态变量的初始化之前执行的。
- (7) MAX_SIZE是个合法的静态final变量名称。
- (8) 静态初始化程序会在构造函数之前执行。
- (9) 如果类被标记为final，则它的方法也必须标记为final。
- (10) final的方法只能在它的类被继承时覆盖。
- (11) boolean类型没有包装用的类。
- (12) wrapper是用来把primitive主数据类型包装成对象。
- (13) parseXxx方法都会返回String。
- (14) 格式化的类都在java.format中包。



排排看

下面是被打散的Java程序片段，程序的目的是要以29.52的周期计算满月：上一次的满月是在2004年1月7日。你是否能够将它们重新排列以成为可以编译并执行并产生如同下方的输出结果？注意到有些括号已经遗失，所以你可以在认为有需要时自行补上。

```

long day1 = c.getTimeInMillis();
c.set(2004,1,7,15,40);

import static java.lang.System.out;
static int DAY_IM = 60 * 60 * 24;
("full moon on %tc", c));
(c.format
Calendar c = new Calendar();
class FullMoons {

public static void main(String [] args) {
    day1 += (DAY_IM * 29.52);
    for (int x = 0; x < 60; x++) {
        static int DAY_IM = 1000 * 60 * 60 * 24;
        println
        import java.io.*;
        ("full moon on %t", c));
        import java.util.*;
        static import java.lang.System.out;
        c.set(2004,0,7,15,40); out.println
        c.setTimeInMillis(day1);
        (String.format
Calendar c = Calendar.getInstance();

```

```

File Edit Window Help Howl
% java FullMoons
full moon on Fri Feb 06 04:09:35 MST 2004
full moon on Sat Mar 06 16:38:23 MST 2004
full moon on Mon Apr 05 06:07:11 MDT 2004

```