4 方法操作实例变量

对象的行为

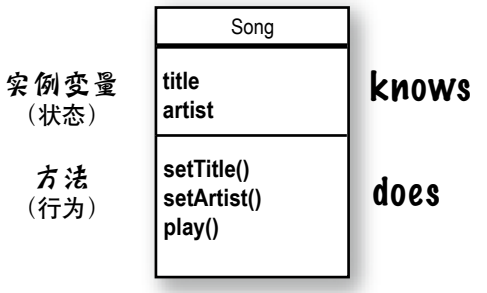
危险动作，请勿模仿

（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！

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？



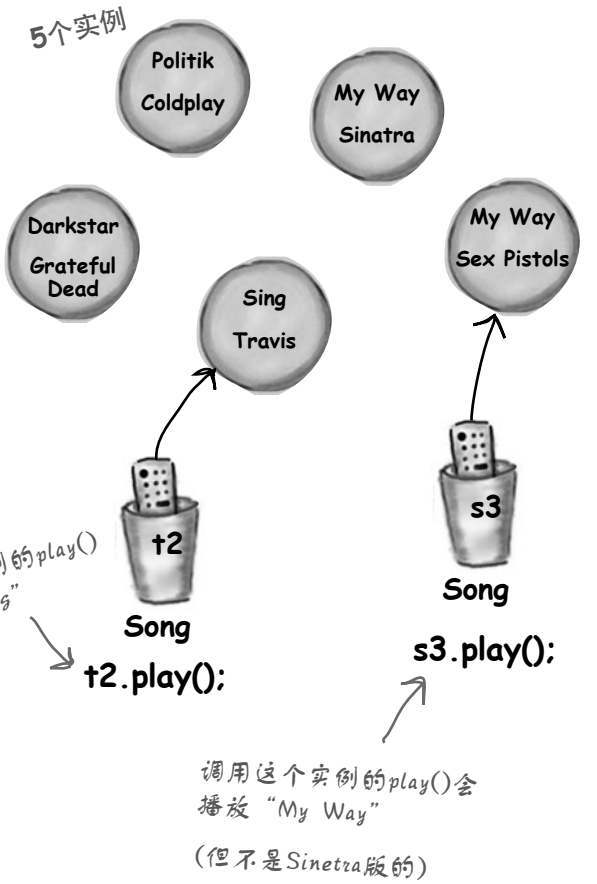
同一类型的每个对象能够有不同的方法行为吗？

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);
}
```



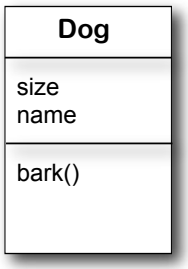

```
Song t2 = new Song();
t2.setArtist("Travis");
t2.setTitle("Sing");
Song s3 = new Song();
s3.setArtist("Sex Pistols");
s3.setTitle("My Way");
```

*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。

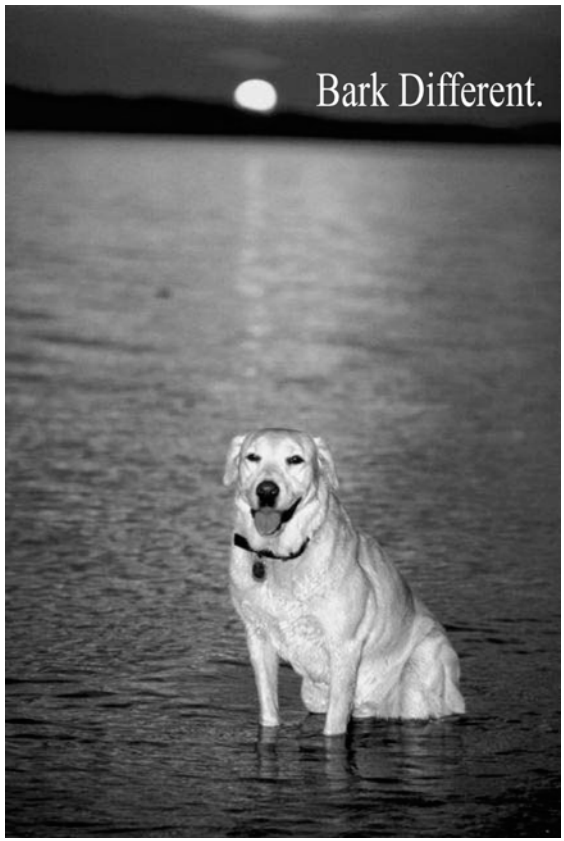
Dog这个类有个称为size的实例变量，bark()会用它来决定使用哪一种声音。




```
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();
   }
}
```

```
File Edit Window Help Playdead
%java DogTestDrive
Wooof! Wooof!
Yip! Yip!
Ruff!  Ruff!
```



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：

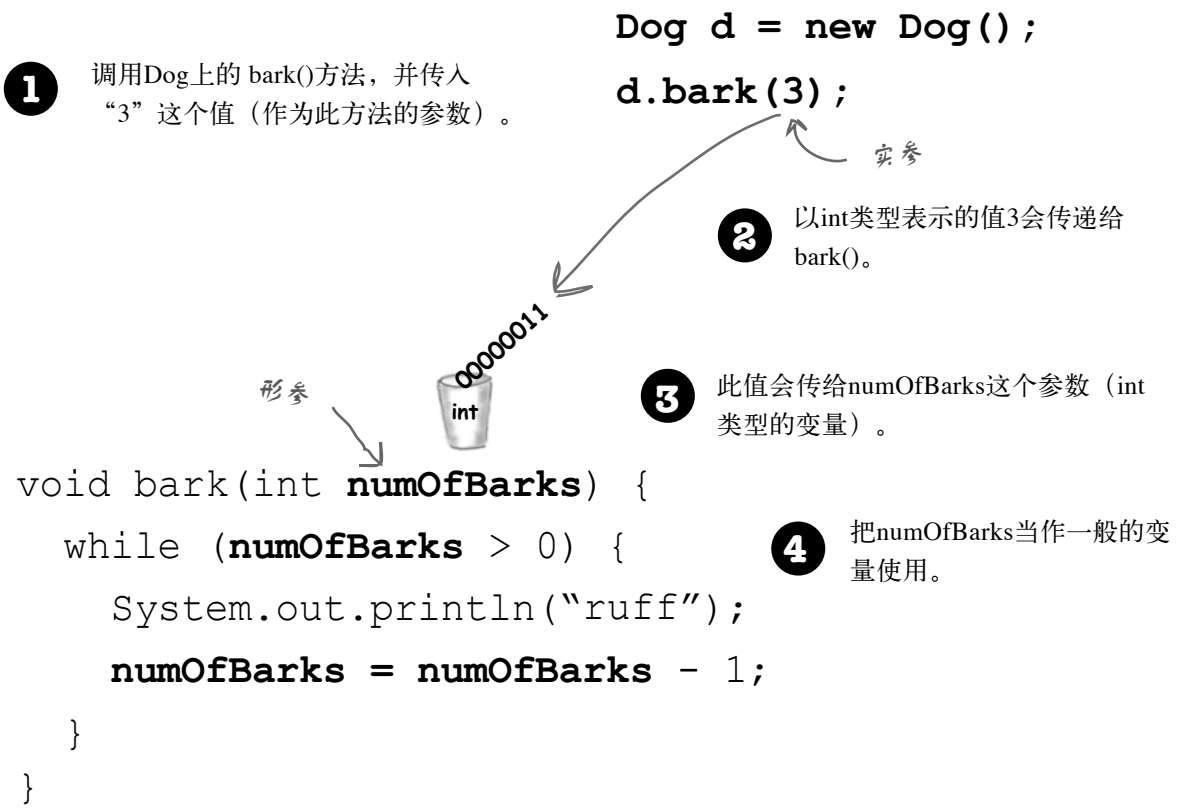
```
d.bark(3);
```

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() {
}
```

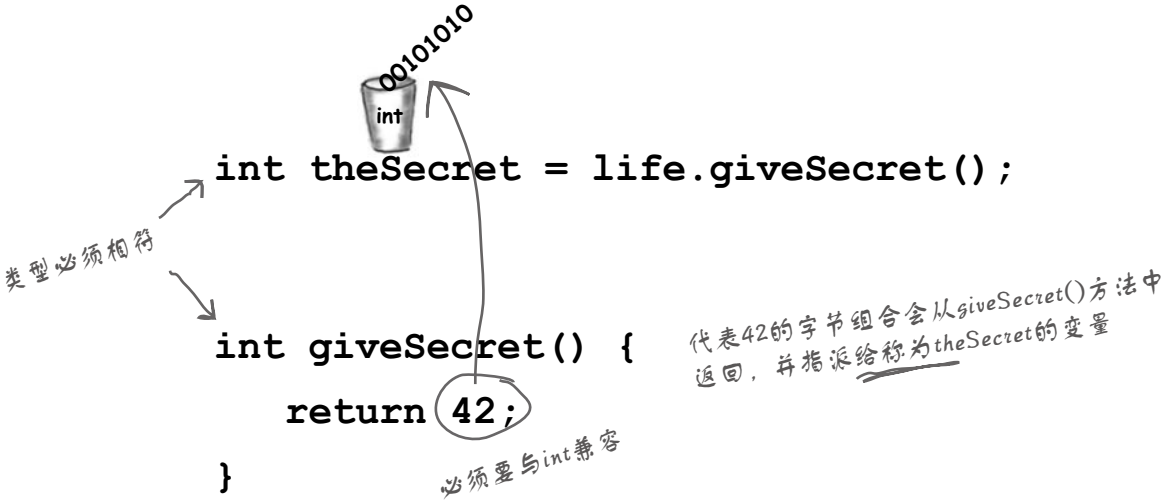
但我们可以声明一个方法，回传给调用方指定的类型值，如：

```
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）。

编译器不会让你返回错误的类型

说好了要返回，最好就得返回！

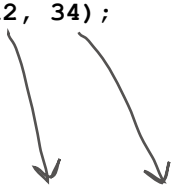


你可以向方法中传入一个以上的参数

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：

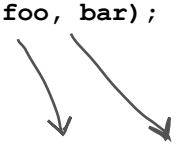
```
void go() {
   TestStuff t = new TestStuff();
   t.takeTwo(12, 34);
}

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的值会是7，而y的值会是3

x的值会是10

Java是通过值传递的

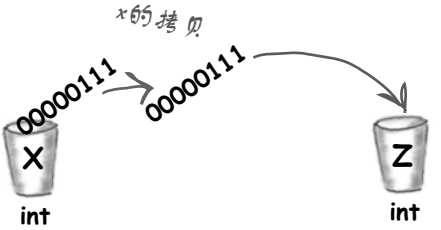
也就是说通过拷贝传递

```
int x = 7;
```

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。

```
void go(int z){ }
```

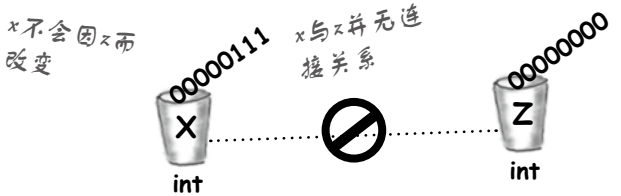
2 声明一个有int参数的方法，参数名称为z。




```
foo.go(x);
```

```
void go(int 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。




```
void go(int z){
    z = 0;
}
```

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。

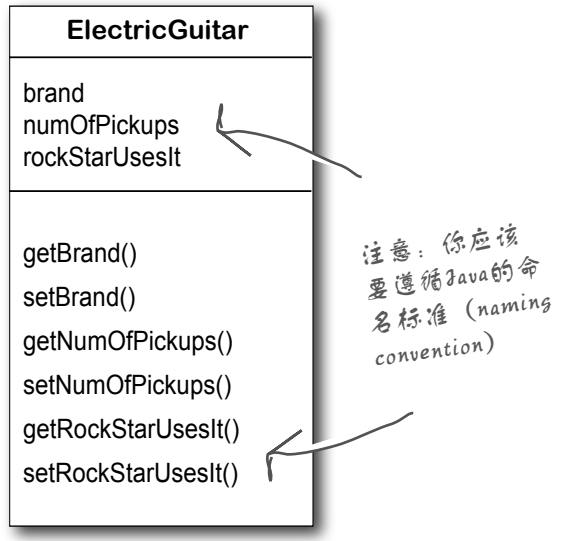
要点

- 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Accessor与Mutator。不过这样只是更饶舌而已，由于Getter与Getter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。毫无意外的，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;
   }
}
```

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。

强迫所有人都得调用Setter，我们就可以防止Cat被设定成无法接受的高度。

```
public void setHeight(int ht) {
    if (ht > 9) {
        height = ht;
    }
}
```

这个检查可以确保高度不会低于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）

Java Exposed

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。

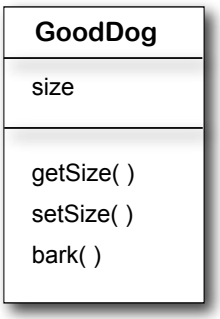
HeadFirst：我懂你的意思了。那封装是如何设下保护罩的？

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。重点在于你可于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



将实例变量设定为private的

将getter与setter设定为private的

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。

```
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("Wooof! Wooof!");
    } 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：

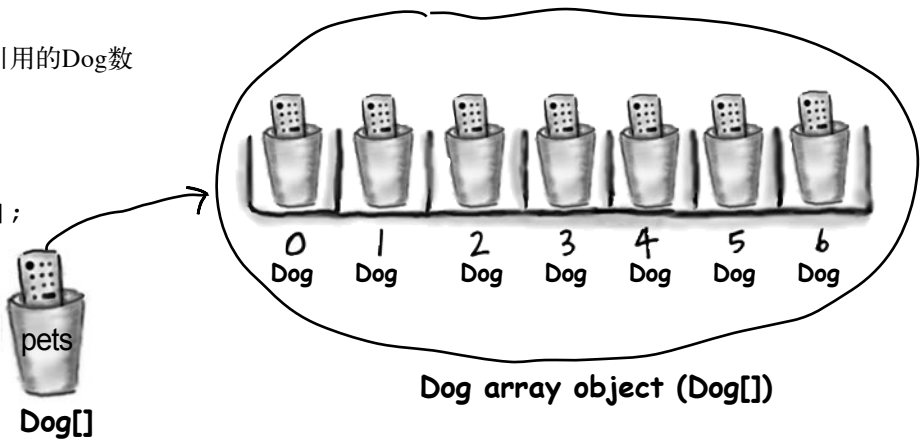
```
int x = 3 + one.getSize();
```

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];
```



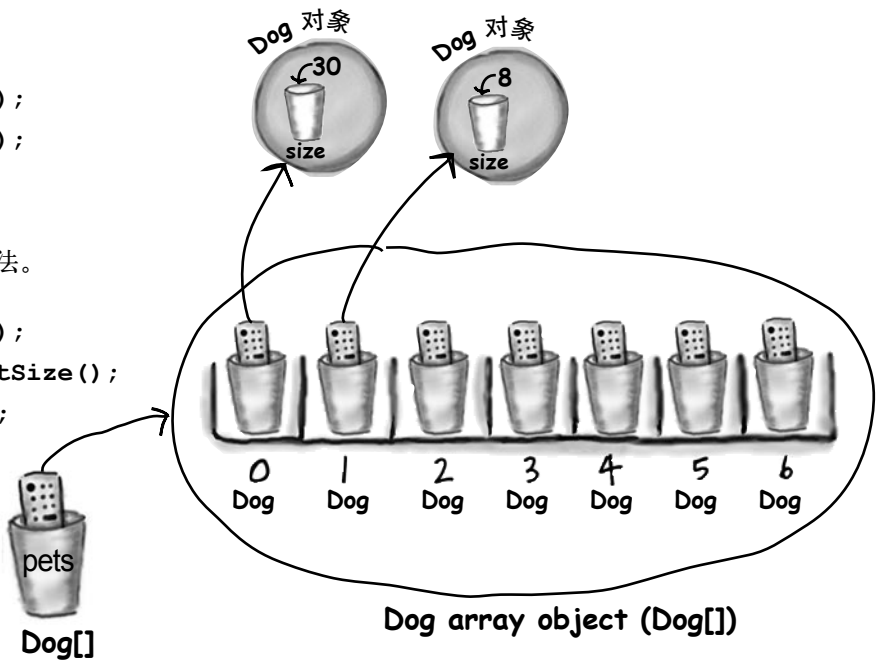
---

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？

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

```
class PoorDog {

   private int size;
   private String name;

   public int getSize() {
      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());
   }
}
```

声明实例变量但是不给值

会返回什么？

你想这会通过编译吗？

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

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，它们在涵义上是相等的。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==来比较两个primitive主数据类型，或者判断两个引用是否引用同一个对象。

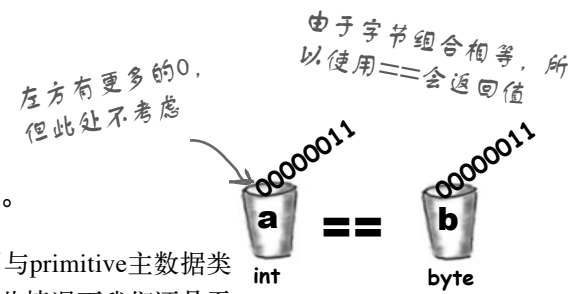
使用equals()来判断两个对象是否在意义上相等。

（像是两个String对象是否带有相同的字节组合）

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。

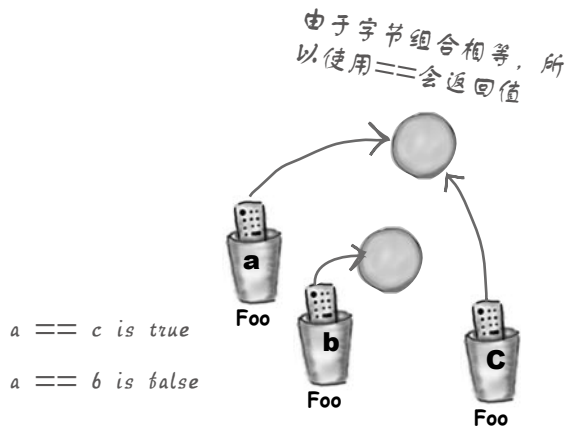
```
int a = 3;

byte b = 3;

if (a == b) { // true }
```

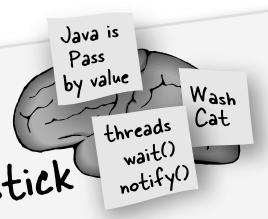


使用==来判别两个引用是否都指向同一对象。

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 }
```





Make it Stick

玫瑰是红的，
有些事情是讲天分的，
以值传递就是以拷贝传递。

噢，你以为写个烂诗很容易吗？那你写个新诗来给我瞧瞧。其实只要你写个够好的诗，那你就永远不会忘记这件事。

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);
  }
}
```

练习

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。

```
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;
  }
}
```

程序段	输出
`x < 9` `index < 5`	`14 7`
	`9 5`
`x < 20` `index < 5`	`19 1`
	`14 1`
`x < 7` `index < 7`	`25 1`
	`7 7`
`x < 19` `index < 1`	`20 1`
	`20 5`

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
```

```
public class Puzzle4 {
  public static void main(String [] args) {

    _____________________________________
    int y = 1;
    int x = 0;
    int result = 0;
    while (x < 6) {

      _____________________________
      _____________________________
      y = y * 10;

      ___________________
    }
    x = 6;
    while (x > 0) {

      ___________________
      result = result + _____________________
    }
    System.out.println("result " + result);
  }
}

class ___________ {
  int ivar;
  _________  ______ doStuff(int __________) {
    if (ivar > 100) {
      return _____________________________
    } else {
      return _____________________________
    }
  }
}
```

从池中找出可以填在输出空格部分的片段。

```
doStuff(x);
obs.doStuff(x);
obs[x].doStuff(factor);
obs[x].doStuff(x);

ivar = x;
obs.ivar = x;
obs[x].ivar = x;
obs[x].ivar = y;

ivar
factor
public
private

ivar + factor;
ivar * (2 + factor);
ivar * (5 - factor);
ivar * factor;

Puzzle4
Puzzle4b
Puzzle4b( )

int
short

Puzzle4 [ ] obs = new Puzzle4[6];
Puzzle4b [ ] obs = new Puzzle4b[6];
Puzzle4b [ ] obs = new Puzzle4[6];

x = x + 1;
x = x  - 1;

obs [x] = new Puzzle4b(x);
obs [ ] = new Puzzle4b( );
obs [x] = new Puzzle4b( );
obs = new Puzzle4b( );
```

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。

阿仁的身份会曝光吗？傻强到底说出了什么不该说的事？

“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，就说如果有个兄弟写程序有漏洞，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。”

练习解答

B

```
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

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。

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。