

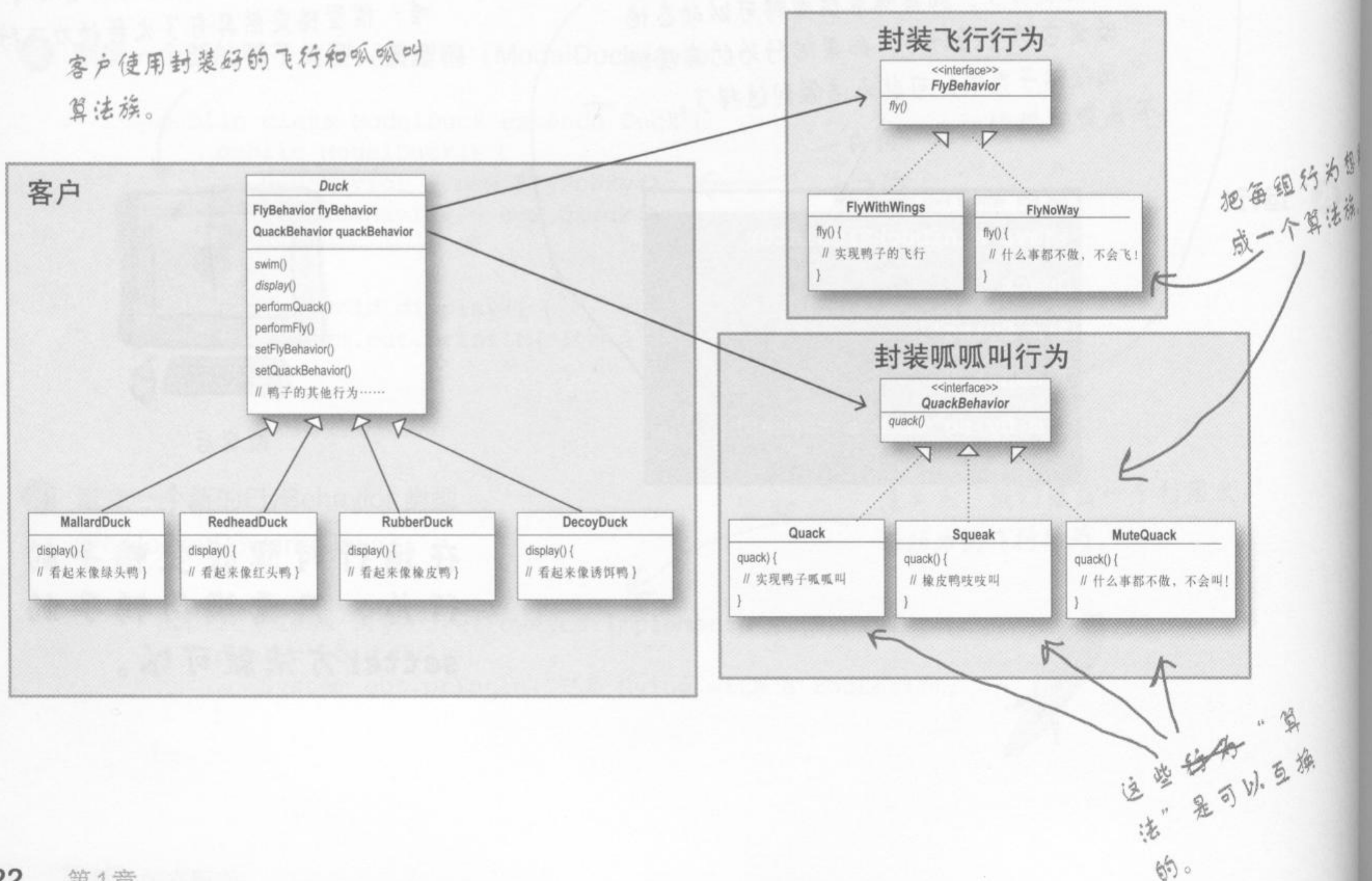
## 封装行为的大局观

好，我们已经深入研究了鸭子模拟器的设计，该是将头探出水面，呼吸空气的时候了。现在就来看看整体的格局。

下面是整个重新设计后的类结构，你所期望的一切都有：鸭子继承Duck，飞行行为实现FlyBehavior接口，呱呱叫行为实现QuackBehavior接口。

也请注意，我们描述事情的方式也稍有改变。不再把鸭子的行为说成是“一组行为”，我们开始把行为想成是“一族算法”。想想看，在SimUDuck的设计中，算法代表鸭子能做的事（不同的叫法和飞行法），这样的做法也能很容易地用于用一群类计算不同州的销售税金。

请特别注意类之间的“关系”。拿起笔，把下面图形中的每个箭头标上适当的关系，关系可以是IS-A（是一个）、HAS-A（有一个）或IMPLEMENTS（实现）。





## “有一个”可能比“是一个”更好。

“有一个”关系相当有趣：每一鸭子都有一个FlyBehavior和一个QuackBehavior，好将飞行和呱呱叫委托给它们代为处理。

当你将两个类结合起来使用，如同本例一般，这就是组合（composition）。这种做法和“继承”不同的地方在于，鸭子的行为不是继承来的，而是和适当的行为对象“组合”来的。

这是一个很重要的技巧。其实是使用了我们的第三个设计原则：



### 设计原则

多用组合，少用继承。

如你所见，使用组合建立系统具有很大的弹性，不仅可将算法族封装成类，更可以“在运行时动态地改变行为”，只要组合的行为对象符合正确的接口标准即可。

组合用在“许多”设计模式中，在本书中，你也会看到它的诸多优点和缺点。



### BRAIN POWER

鸭鸣器（duckcall）是一种装置，猎人用鸭鸣器模拟出鸭叫声，以引诱野鸭。你如何实现你自己的鸭鸣器，而不继承Duck类？



### 大师与门徒……

大师：蚱蜢，告诉我，在面向对象的道路上，你学到了什么？

门徒：大师，我学到了，面向对象之路承诺了“复用”。

大师：继续说……

门徒：大师，借由继承，好东西可以一再被利用，所以程序开发时间就会大幅减少，就好像在林中很快地砍竹子一样。

大师：蚱蜢呀！软件开发完成“前”以及完成“后”，何者需要花费更多时间呢？

门徒：答案是“后”，大师。我们总是需要花许多时间在系统的维护和变化上，比原先开发花的时间更多。

大师：蚱蜢，这就对啦！那么我们是不是应该致力于提高可维护性和可扩展性上的复用程度呀？

门徒：是的，大师，的确是如此。

大师：我觉得你还有很多东西要学，希望你再深入研究继承。你会发现，继承有它的问题，还有一些其他方式可以达到复用。



## 讲到设计模式……



恭喜你，学会第一个模式了！

你刚刚用了你的第一个设计模式：也就是策略模式（Strategy Pattern）。不要怀疑，你正是使用策略模式改写SimUDuck 程序的。多亏这个模式，现在系统不担心遇到任何改变，主管们可以勾画他们的赌城狂欢之旅了。

为了介绍这个模式，我们走了很长的一段路。下面是此模式的正式定义：

**策略模式**定义了算法族，分别封装起来，让它们之间可以互相替换，此模式让算法的变化独立于使用算法的客户。

当你需要给朋友留下深刻的印象，或是想影响关键主管的决策时，请使用“这个”定义。






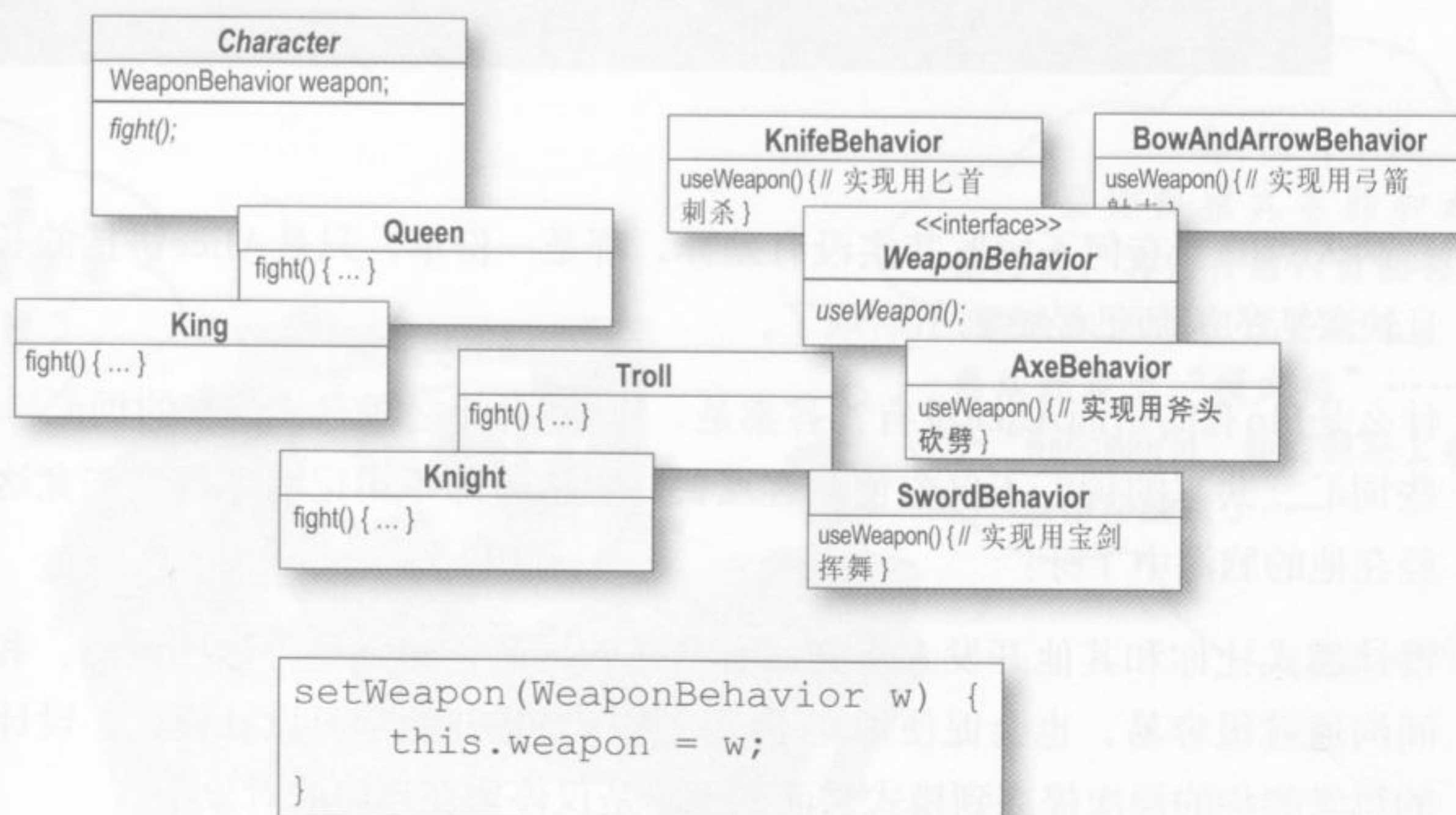
## 设计谜题

在下面，你将看到一堆杂乱的类与接口，这取自一个动作冒险游戏。你将看到代表游戏角色的类和角色可以使用的武器行为的类。每个角色一次只能使用一种武器，但是可以在游戏的过程中换武器。你的工作是要弄清楚这一切……

(答案在本章结尾处)

你的任务：

- 1 安排类。
- 2 找出一个抽象类、一个接口，以及八个类。
- 3 在类之间画箭头。
  - a. 继承就画成这样（“extend”）。 
  - b. 实现接口就画成这样（“implement”）。 
  - c. “有一个”关系就画成这样。 
- 4 把setWeapon()方法放到正确的类中。





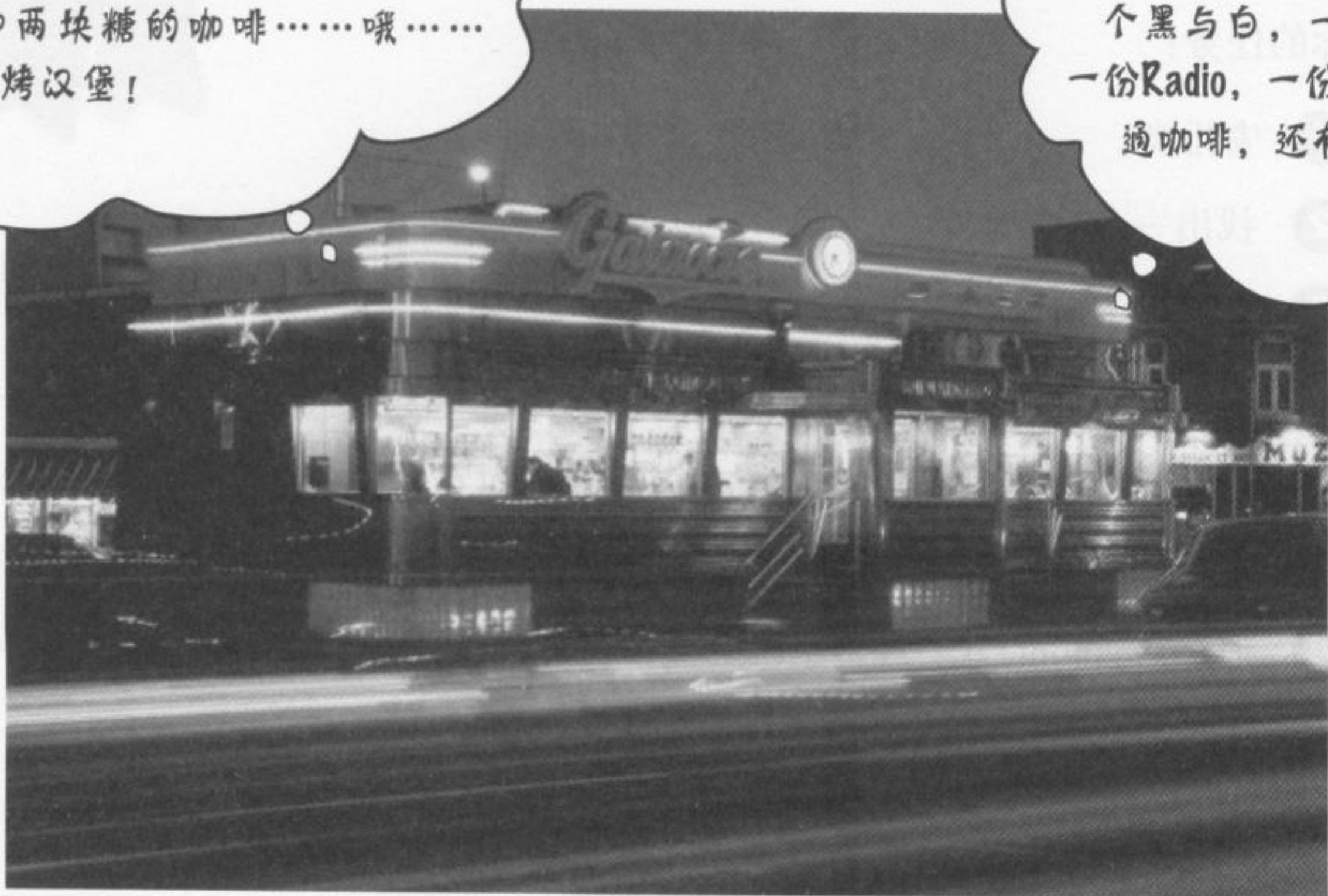
## 在附近餐厅中无意间听到……

Alice

我要一份涂了奶酪及果酱的白面包、加了香草冰淇淋的巧克力汽水、夹了培根的火烤起司三明治、鲔鱼色拉吐司、香蕉船（有冰淇淋和香蕉片）、一杯加了奶精和两块糖的咖啡……哦……还有一个烧烤汉堡！

Flo

给我一份C.J.怀特，一个黑与白，一份杰克·班尼，一份Radio，一份主厨船，一个普通咖啡，还有给我烧一个！



这两人点的餐有何不同？其实没有差异，都是一份单，只是Alice讲话的长度多了一倍，而且快餐店的厨师已经感到不耐烦了。

什么是Flo有的，而Alice没有？答案是，Flo和厨师之间有“共享的词汇”，Alice却不懂这些词汇。共享的词汇不仅方便顾客点餐，也让厨师不用记太多事，毕竟这些餐点模式都已经在他的脑海中了呀！

设计模式让你和其他开发人员之间有共享的词汇，一旦懂得这些词汇，和其他开发人员之间沟通就很容易，也会促使那些不懂的程序员想开始学习设计模式。设计模式也可以把你的思考架构的层次提高到模式层面，而不是仅停留在琐碎的对象上。



# 在办公室隔间中无意间听到……

我建立了这个广播类。它能够追踪所有的倾听对象，而且任何时候只要有新资料进来，就会通知每个倾听者。最棒的是，倾听者可以随时加入此广播系统，甚至可以随时退出。这样的设计方式相当动态和松耦合。

Rick



## BRAIN POWER

除了面向对象设计和在餐厅点餐之外，你还能够想到有哪些例子需要共享词汇？（暗示：想一想汽车修理工、木工、大厨、航管）利用这些行话进行沟通的质量如何？

你能否想到OO设计的什么方面，能够和模式名称匹配的？“策略模式”这个名字是否传神？

Rick，你只要说使用了“观察者模式”我们就懂了。



没错，如果你用模式名称和大家沟通，其他开发人员能够马上且清楚地知道你在说些什么。但是也请不要从此染上“模式病”……以后连写一个“HelloWorld”都能够扯上模式，那就代表你已经病了……



## 共享模式词汇的威力

你使用模式和他人沟通时，其实“不只是”和他人共享“行话”而已。

共享的模式词汇“威力强大”。当你使用模式名称和其他开发人员或者开发团队沟通时，你们之间交流的不只是模式名称，而是一整套模式背后所象征的质量、特性、约束。

模式能够让你用更少的词汇做更充分的沟通。当你用模式描述的时候，其他开发人员便很容易地知道你对设计的想法。

将说话的方式保持在模式层次，可让你待在“设计圈子”久一点。使用模式谈论软件系统，可以让你保持在设计层次，不会被压低到对象与类这种琐碎的事情上面。

共享词汇可帮你的开发团队快速充电。对于设计模式有深入了解的团队，彼此之间对于设计的看法不容易产生误解。

共享词汇能帮助初级开发人员迅速成长。初级开发人员向有经验的开发人员看齐。当高级开发人员使用设计模式，初级开发人员也会跟着学。把你的组织建立一个模式使用者的社区。

“我们使用策略模式实现鸭子的各种行为。”这句话也就是告诉我们，鸭子的行为被封装进入一组类中，可以被轻易地扩充与改变。如果需要，甚至在运行时也可以改变行为。

想想看，有多少次的设计会议中，你们一不小心就进入了琐碎的实现细节的讨论上。

当你的团队开始利用模式分享设计想法与经验，你等于是建立了一个模式使用者的社区。

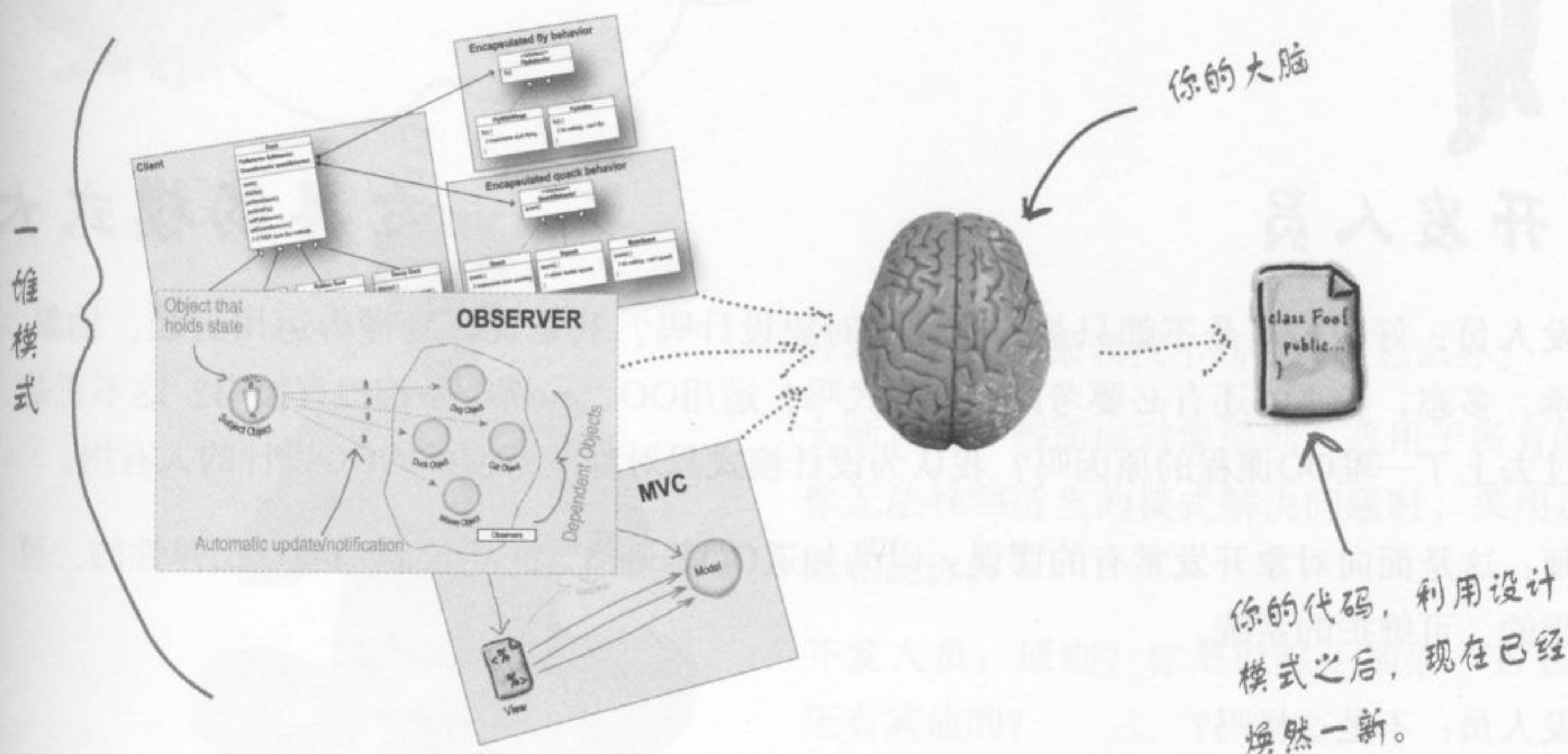
考虑在你的组织内发起一个设计模式研讨会，说不定在学习的过程中，就开始得到回报了……



## 我如何使用设计模式？

我们全都使用别人设计好的库与框架。我们讨论库与框架、利用它们的API编译成我们的程序、享受运用别人的代码所带来的优点。看看Java API及它所带来的功能：网络、GUI、IO等。库与框架长久以来，一直扮演着软件开发过程的重要角色，我们从中挑选所要的组件，把它们放进合适的地方。但是……库与框架无法帮助我们将应用组织成容易了解、容易维护、具有弹性的架构，所以需要设计模式。

设计模式不会直接进入你的代码中，而是先进入你的“大脑”中。一旦你先在脑海中装入了许多关于模式的知识，就能够开始在新设计中采用它们，并当你的旧代码变得如同搅和成一团没有弹性的意大利面一样时，可用它们重做旧代码。



### there are no Dumb Questions

**问：** 如果设计模式这么棒，为何没有人建立相关的库呢？那样我们就不必自己动手了。

**答：** 设计模式比库的等级更高。设计模式告诉我们如何组织类和对象以解决某种问题。而且采纳这些设计并使它们适合我们特定的应用，是我们责无旁贷的事。

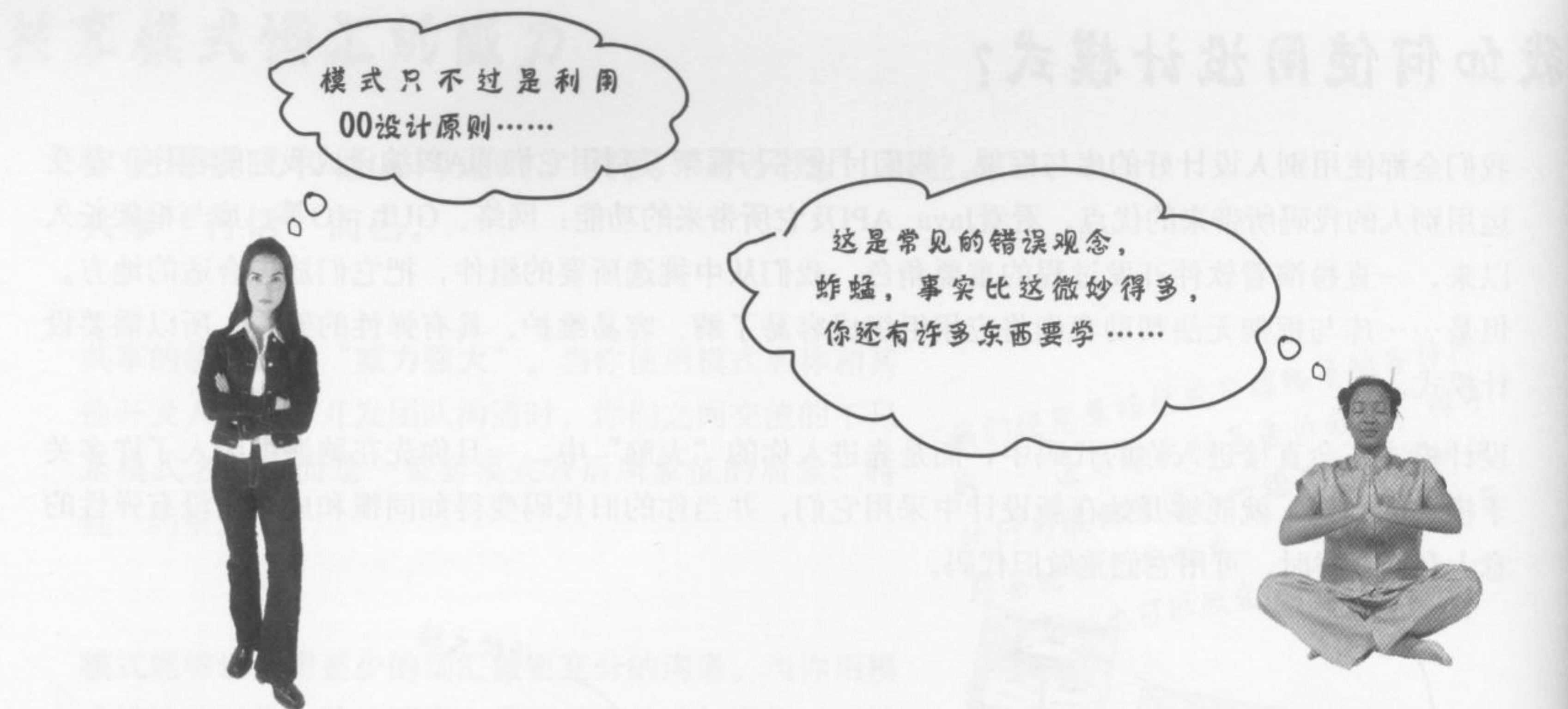
**问：** 库和框架不也是设计模式吗？

**答：** 库和框架提供了我们某些特定的实现，让我们的代码可以轻易地引用，但是这并不算是设计模式。有些时候，库和框架本身会用到设计模式，这样很好，因为一旦你了解了设计模式，会更容易了解这些API是围绕着设计模式构造的。

**问：** 那么，没有所谓设计模式的库？

**答：** 没错，但是稍后你会看到设计模式类目。你可以在应用中利用这些设计模式。





## 怀疑的开发人员

## 友善的模式大师

开发人员：好吧！但是不都只是好的面向对象设计吗？我是说，我懂得运用封装、抽象、继承、多态，我真的还有必要考虑设计模式吗？运用OO，一切不是都很直接吗？这不正是我过去上了一堆OO课程的原因吗？我认为设计模式只对那些不懂好的OO设计的人有用。

大师：这是面向对象开发常有的谬误：以为知道OO基础概念，就能自动设计出弹性的、可复用的、可维护的系统。

开发人员：不是这样吗？

大师：不是！要构造有这些特征的OO系统，事实证明只有通过不断地艰苦实践，才能成功。

开发人员：我想我开始了解了，这些构造OO系统的隐含经验于是被收集整理出来……

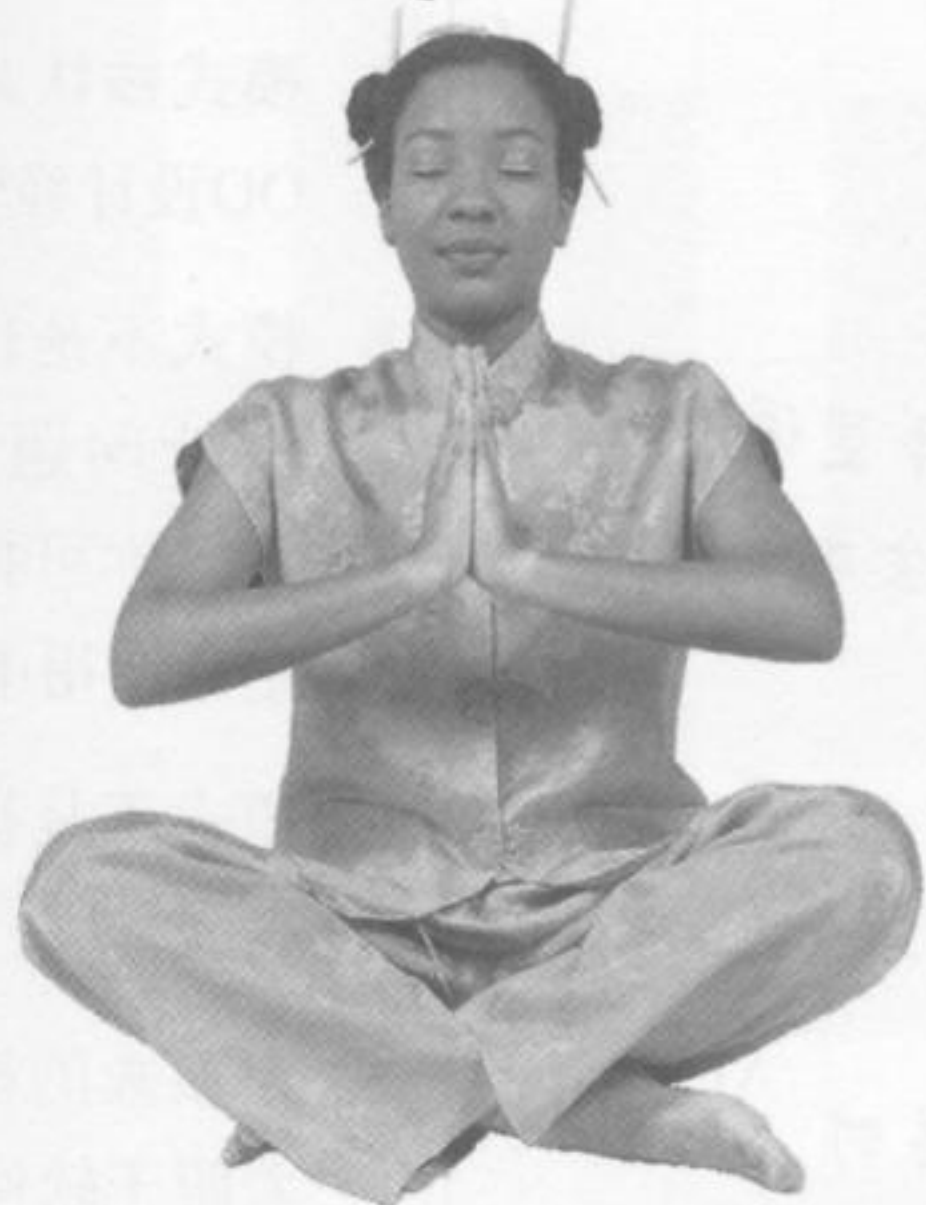
大师：……是的，被整理成了一群“设计模式”。

开发人员：那么，如果知道了这些模式，我就可以减少许多体力劳动，直接采用可行的模式吗？

大师：对，在一定程度上可以这么说。不过要记住，设计是一门艺术，总是有许多可取舍的地方。但是如果你能采用这些经过深思熟虑，且经受过时间考验的设计模式，你就领先别人了。



记住，知道抽象、继承、多态这些概念，并不会马上让你变成好的面向对象设计者。设计大师关心的是建立弹性的设计，可以维护，可以应付变化。



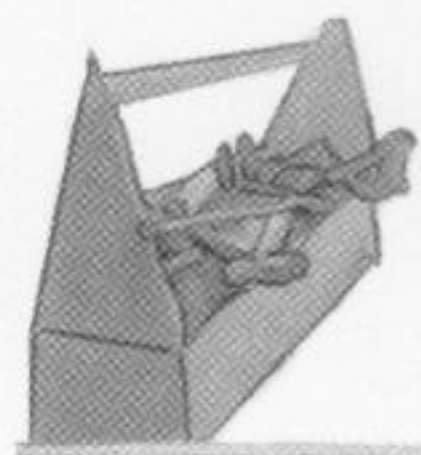
开发人员：如果我找不到模式，怎么办？

大师：有一些面向对象原则，适用于所有的模式。当你无法找到适当的模式解决问题时，采用这些原则可以帮助你。

开发人员：原则？你是说除了抽象、封装……之外，还有其他的？

大师：是的，建立可维护的OO系统，要诀就在于随时想到系统以后可能需要的变化以及应付变化的原则。





## 设计工具箱内的工具

你几乎快要读完第1章了！你已经在你的设计工具箱内放进了几样工具，在我们进入第2章之前，先将这些工具一一列出。

### OO基础

抽象

封装

多态

继承

我们假设你知道OO基础包括了多态的用法、继承就像按契约进行设计、封装是如何运作的。如果你觉得脑袋有一点生锈了，快快拿出你的《Head First Java》复习，然后再把这一章读一遍。

### OO原则

封装变化

多用组合，少用继承

针对接口编程，不针对实现编程

我们会在后续的内容中更详细地看看这些原则，还会再多加一些原则到清单上。

### OO模式

策略模式——定义算法族，分别封装起来，让它们之间可以互相替换，此模式让算法的变化独立于使用算法的客户。

阅读本书时，时时刻刻要思考着：模式如何仰赖基础与原则。

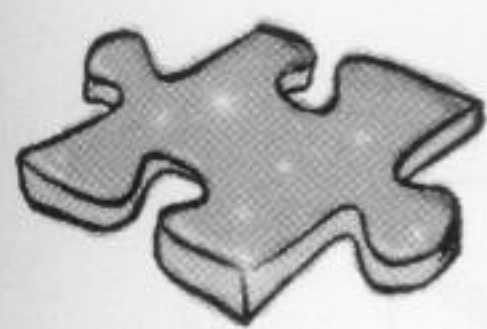
学了一个，还有更多！

### 要点



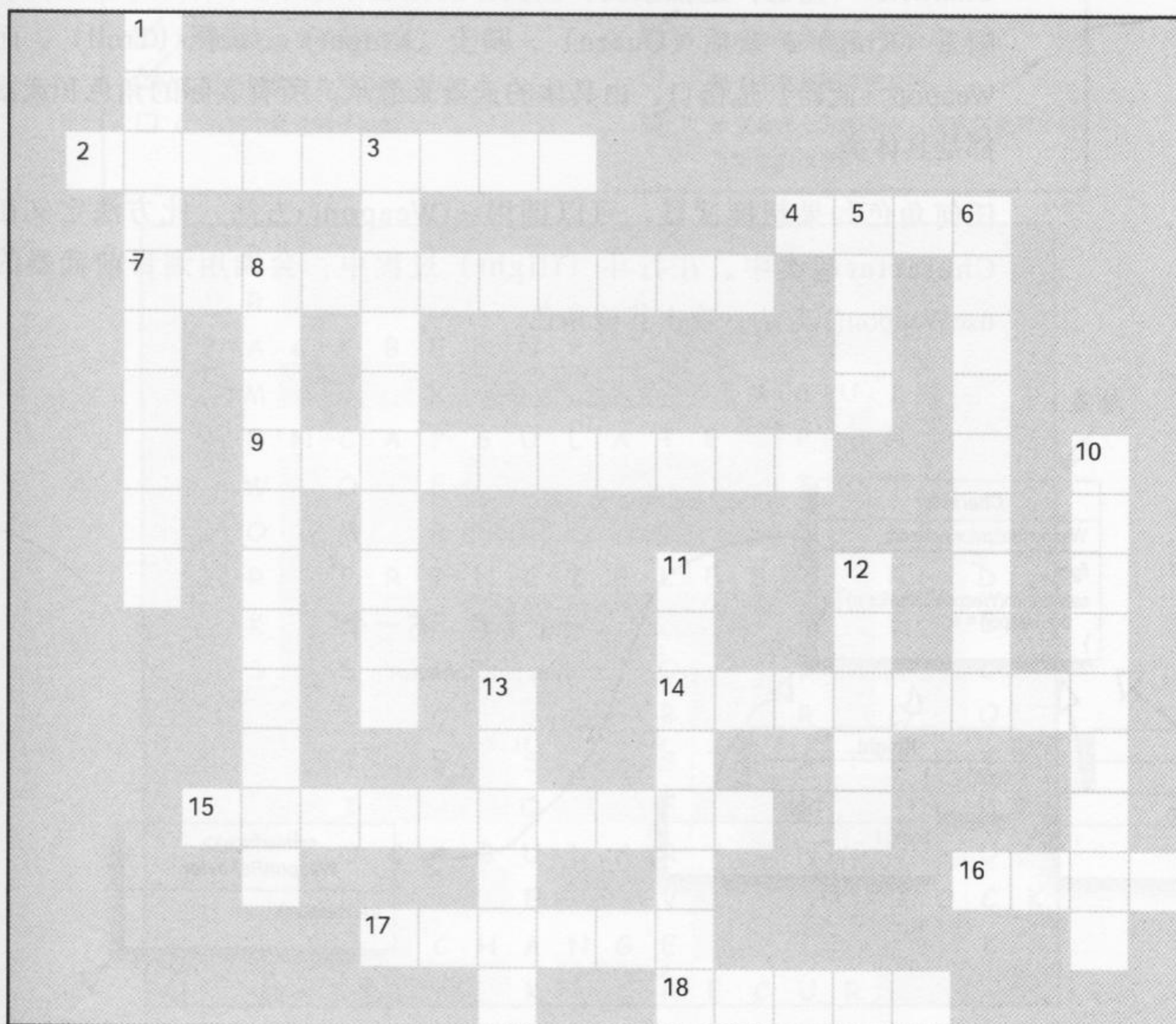
- 知道OO基础，并不足以让你设计出良好的OO系统。
- 良好的OO设计必须具备可复用、可扩充、可维护三个特性。
- 模式可以让我们建造出具有良好OO设计质量的系统。
- 模式被认为是历经验证的OO设计经验。
- 模式不是代码，而是针对设计问题的通用解决方案。你可把它们应用到特定的应用中。
- 模式不是被发明，而是被发现。
- 大多数的模式和原则，都着眼于软件变化的主题。
- 大多数的模式都允许系统局部改变独立于其他部分。
- 我们常把系统中会变化的部分抽出来封装。
- 模式让开发人员之间有共享的语言，能够最大化沟通的价值。





让标准填字游戏，动动你的右脑。

这是一个标准的纵横填字游戏，所有的词都来自本章。



### 横排提示：

2. Grilled cheese with bacon
4. Duck demo was located where
7. \_\_\_\_\_ what varies
9. Most patterns follow from OO \_\_\_\_\_
14. Pattern that fixed the simulator
15. Patterns give us a shared \_\_\_\_\_
16. Design patterns \_\_\_\_\_
17. Development constant
18. Patterns \_\_\_\_\_ in many applications

### 竖排提示：

1. High level libraries
3. Learn from the other guy's \_\_\_\_\_
5. Java IO, Networking, Sound
6. Program to this, not an implementation
8. Favor over inheritance
10. Duck that can't quack
11. Rick was thrilled with this pattern
12. Patterns go into your \_\_\_\_\_
13. Rubberducks make a \_\_\_\_\_

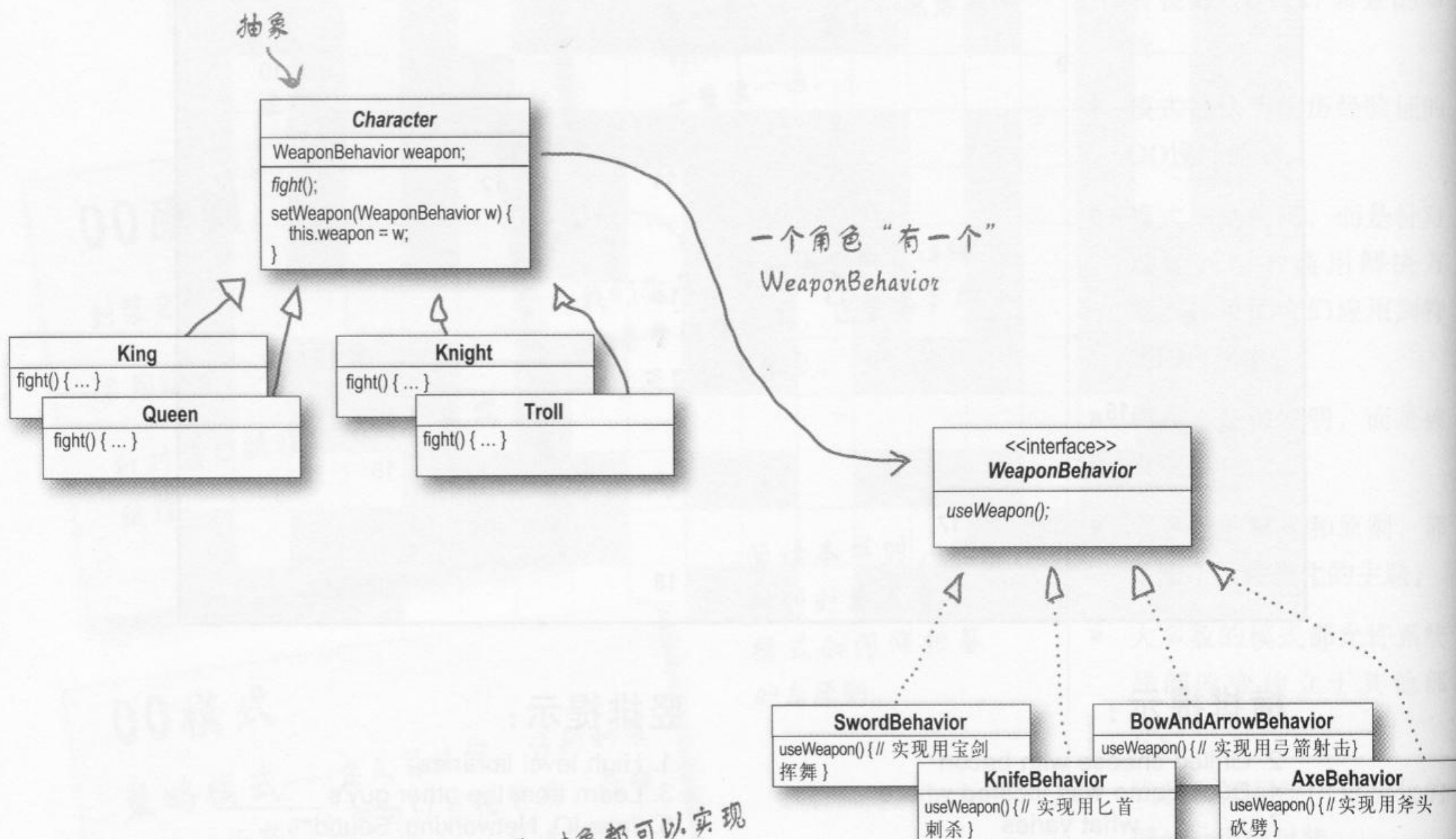




## 设计谜题解答

Character（角色）是抽象类，由具体的角色来继承。具体的角色包括：国王（King）、皇后（Queen）、骑士（Knight）、妖怪（Troll）。而 Weapon（武器）是接口，由具体的武器来继承。所有实际的角色和武器都是具体类。

任何角色如果想换武器，可以调用 setWeapon() 方法，此方法定义在 Character 超类中。在打斗（fight）过程中，会调用到目前武器的 useWeapon() 方法，攻击其他角色。





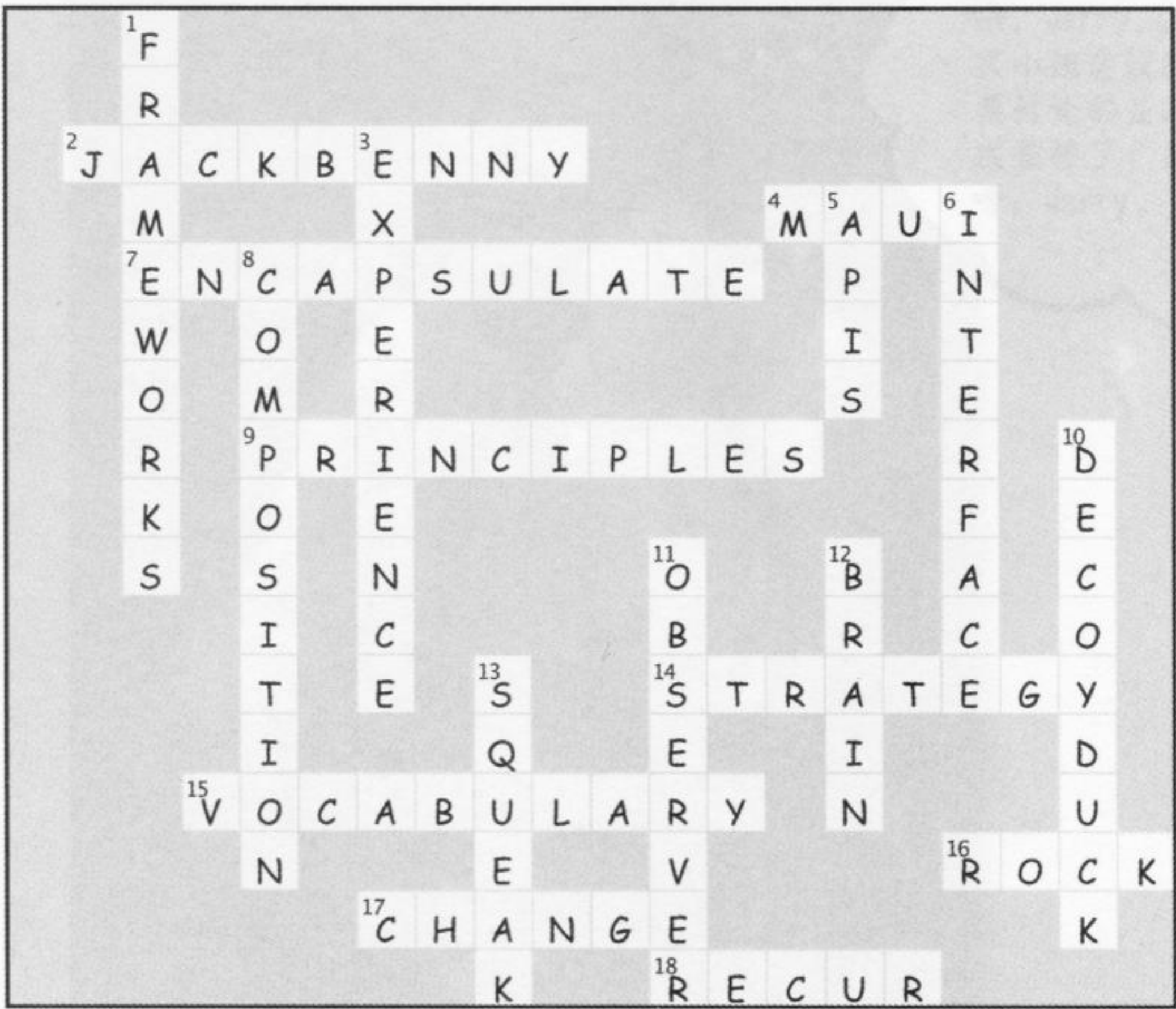
# 答案



## Sharpen your pencil

利用继承来提供Duck的行为，这会导致下列哪些缺点？（多选）

- ☒ A. 代码在多个子类中重复。
- ☒ B. 运行时的行为不容易改变。
- ☐ C. 我们不能让鸭子跳舞。
- ☒ D. 很难知道所有鸭子的全部行为。
- ☐ E. 鸭子不能同时又飞又叫。
- ☒ F. 改变会牵一发而动全身，造成其他鸭子不想要的改变。



## Sharpen your pencil

驱动改变的因素很多。找出你的软件中需要改变代码的地方，一一列出来。下面是我们的答案，你的答案可能和我们不一样。

我们的顾客或用户决定要别的做法，或者想要新功能。  
我的公司决定采用别的数据库产品，又从另一家厂商买了数据，这造成数据格式不兼容。  
唉！  
唔！应对技术改变，我们必须更新代码，适用于新协议。  
我们学到了足够的构建系统的知识，希望回去把事情做得更好。







## 2 观察者 (Observer) 模式

# 让你的对象 知悉现况



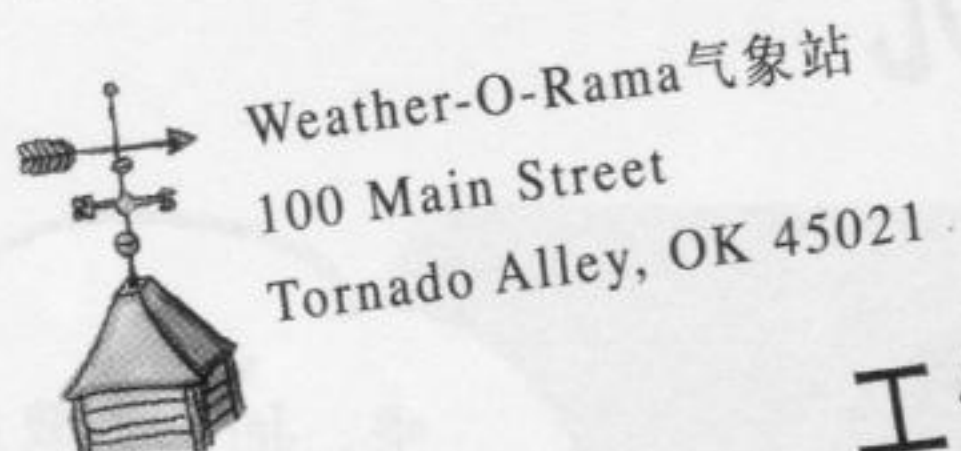
喂，Jerry，我正在通知大家，模式小组会议改到周六晚上，这次要讨论的是观察者模式，这个模式最棒了！超级棒！你一定要来呀，Jerry。

有趣的事情发生时，可千万别错过了！有一个模式可以帮你的对象知悉现况，不会错过该对象感兴趣的事。对象甚至在运行时可决定是否要继续被通知。观察者模式是JDK中使用最多的模式之一，非常有用。我们也会一并介绍一对多关系，以及松耦合（对，没错，我们说耦合）。有了观察者，你将会消息灵通。



恭喜你！

你的团队刚刚赢得一纸合约，负责建立  
Weather-O-Rama公司的下一代气象站——  
Internet气象观测站。



## 工作合约

恭喜贵公司获选为敝公司建立下一代Internet气象观测站！  
该气象站必须建立在我们专利申请中的WeatherData对象  
上，由WeatherData对象负责追踪目前的天气状况（温度、  
湿度、气压）。我们希望贵公司能建立一个应用，有三种  
布告板，分别显示目前的状况、气象统计及简单的预报。  
当WeatherObject对象获得最新的测量数据时，三种布告板  
必须实时更新。

而且，这是一个可以扩展的气象站，Weather-O-Rama气象  
站希望公布一组API，好让其他开发人员可以写出自己的  
气象布告板，并插入此应用中。我们希望贵公司能提供这  
样的API。

Weather-O-Rama气象站有很好的商业营运模式：一旦客  
户上钩，他们使用每个布告板都要付钱。最好的部分就是，  
为了感谢贵公司建立此系统，我们将以公司的认股权支付  
你。

我们期待看到你的设计和应用的alpha版本。

真挚的

*Johnny Hurricane*

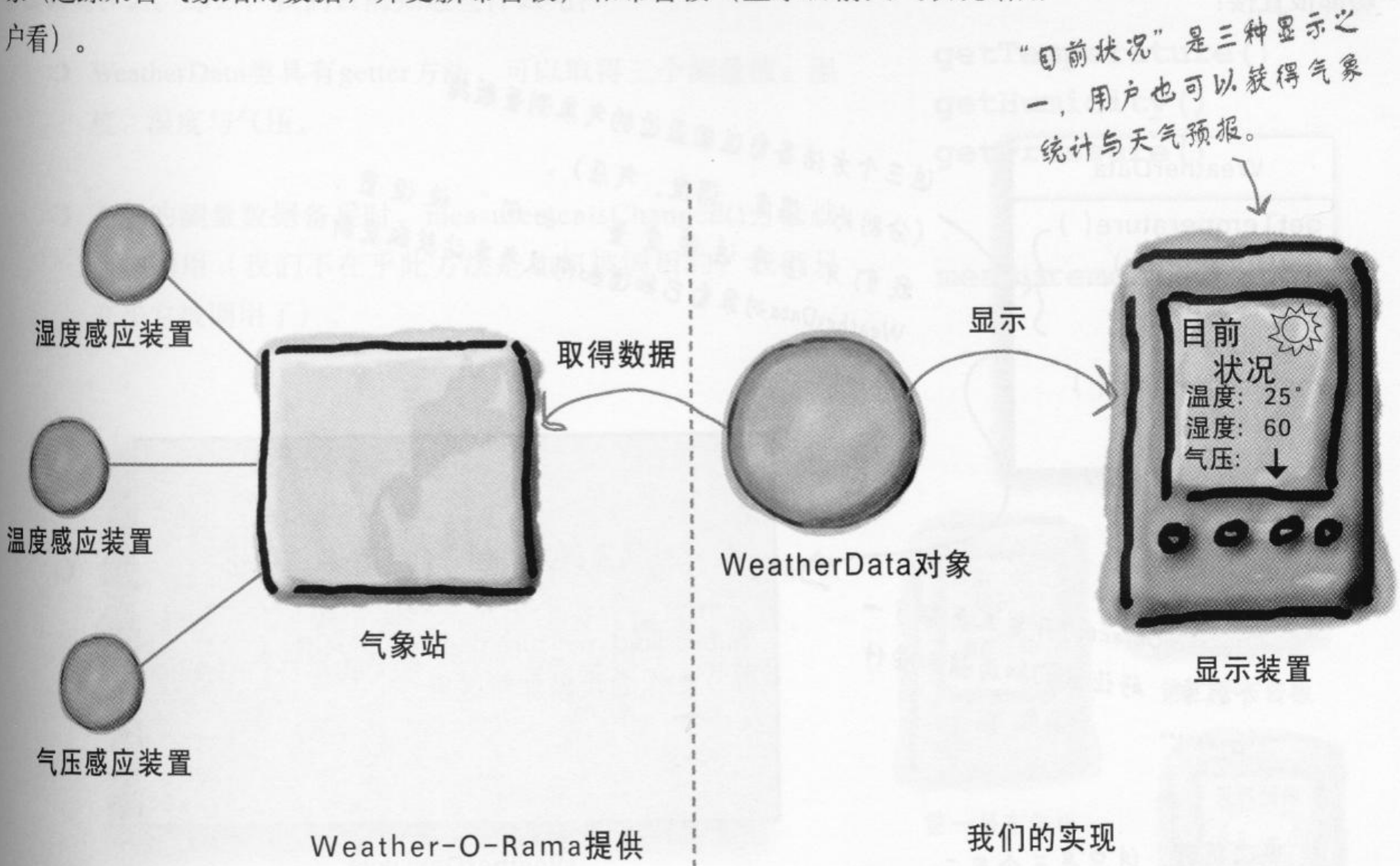
Johnny Hurricane——Weather-O-Rama气象站执行长

附注：我们正通宵整理WeatherData源文件给你们。



# 气象监测应用的概况

此系统中的三个部分是气象站（获取实际气象数据的物理装置）、WeatherData对象（追踪来自气象站的数据，并更新布告板）和布告板（显示目前天气状况给用户看）。



WeatherData对象知道如何跟物理气象站联系，以取得更新的数据。WeatherData对象会随即更新三个布告板的显示：目前状况（温度、湿度、气压）、气象统计和天气预报。

如果我们选择接受这个项目，我们的工作就是建立一个应用，利用WeatherData对象取得数据，并更新三个布告板：目前状况、气象统计和天气预报。