TURING
图灵程序
设计丛书

Java示例代码

图解设计模式

【日】结城浩 著
杨文轩 译

原版连续畅销12年、重印25次！

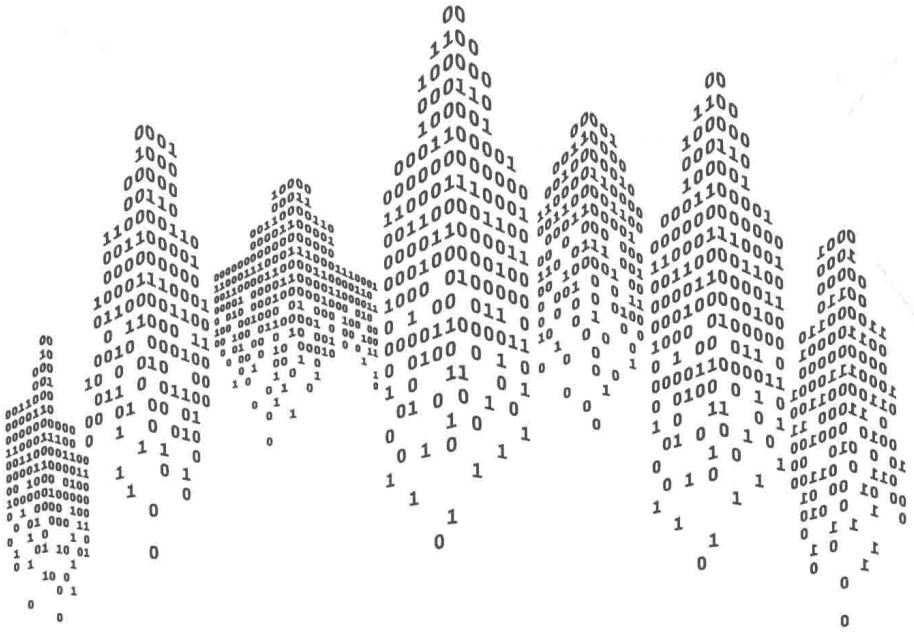
194张图表轻松理解GoF的23种设计模式
《程序员的数学》《数学女孩》作者结城浩又一力作

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

TURING
图灵程序
设计丛书

图解设计模式

【日】结城浩 著
杨文轩 译



人民邮电出版社
北 京

图书在版编目（CIP）数据

图解设计模式 /（日）结城浩著；杨文轩译. -- 北京：人民邮电出版社，2017.1
（图灵程序设计丛书）
ISBN 978-7-115-43949-9

Ⅰ. ①图… Ⅱ. ①结… ②杨… Ⅲ. ①Java语言－程序设计－图解 Ⅳ. ①TP312.8-64

中国版本图书馆CIP数据核字（2016）第264954号

内 容 提 要

本书以浅显易懂的语言逐一说明了GoF的23种设计模式。在讲解过程中，不仅搭配了丰富的图片，而且理论结合实例，用Java语言编写代码实现了设计模式的程序，让程序真正地运行起来，并提供了运用模式解决具体问题的练习题和答案。除此以外，本书在必要时还对Java语言的功能进行补充说明，以加深读者对Java的理解。本书适合对面向对象开发感兴趣、对设计模式感兴趣的人以及所有Java程序员阅读。

◆ 著　[日]结城浩
译　杨文轩
责任编辑　傅志红
执行编辑　高宇涵　侯秀娟
责任印制　彭志环
◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路11号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
三河市中晟雅豪印务有限公司印刷
◆ 开本：787×1092　1/16
印张：24.75
字数：682千字　2017年1月第1版
印数：1-4 000册　2017年1月河北第1次印刷
著作权合同登记号　图字：01-2016-3942号

定价：79.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广字第8052号

站在巨人的肩上

Standing on Shoulders of Giants

iTuring.cn

站在巨人的肩上

Standing on Shoulders of Giants

iTuring.cn

译者序

提起设计模式，GoF 的《设计模式：可复用面向对象软件的基础》一书可谓是设计模式世界的的“圣经”，几乎无人不知，无人不晓。不过，一来该书实际上源自 4 位作者的博士论文，学术性较强，初学者很难透彻理解书中内容。二来，虽说设计模式只是设计思想，不依赖于任何编程语言，但是各种编程语言的特性终究是不同的，而该书中的示例代码又是基于 C++ 和 Smalltalk 的。因此，对于使用 Java 语言编程的开发者来说，当然还是最希望能够阅读通过 Java 语言的示例代码来讲解设计模式的图书。

本书是结城浩先生除《程序员的数学》《图解密码技术（第 3 版）》《数学女孩》系列之外的又一力作，初版于 2001 年 6 月发行。当时，日本还没有通俗易懂地讲解设计模式的图书。就这一点而言，本书堪称日本第一。许多日本 IT 工程师在攻读硕士和博士学位时都学习过本书。如今，15 年过去了，本书历经多次重印，仍位居销售排行榜前列，足见其在日本 IT 类图书中的地位。

当然，在这 15 年间，IT 界也发生了翻天覆地的变化，各种开源框架层出不穷，机器学习大兴其道。但是，在面向对象编程中，设计模式的重要性却不曾改变。与以前一样，在大规模的企业系统开发中，Java 和 C# 仍处于主导地位。在这种大规模系统的开发中，设计模式可以帮助我们实现系统结构化，很好地支撑起系统的稳定性和可扩展性。而本书内容经典，时至今日仍然适用，作为设计模式的入门图书，非常适合于初学设计模式的开发者。

本书特色如下：

- 讲解了 23 种设计模式

本书对 GoF 书中的 23 种设计模式全部进行了讲解。通过了解这些模式，我们可以知道在哪些情况下应当使用哪种设计模式。在编程时，如果能够预测到系统中的某处可能发生什么样的变化，然后提前在系统中使用合适的设计模式，就可以帮助我们以最少量的修改来应对需求变更。设计模式是由前人的知识和经验浓缩而成的，是帮助我们快速提高开发水平的捷径。

- 讲解了对接口的理解

接口的使用方法是 Java 等面向对象编程语言的重要部分，只是满足于知道接口的基本语法是不行的。本书可以帮助我们加深对接口的重要性和使用方法的理解。

- 讲解了可复用代码的写法

需求变更是令所有开发者都会感到头疼的问题。当发生需求变更时，我们总是希望需要修改的代码能尽量集中在一起，不想大范围地修改代码。另外，我们也经常希望在新系统中沿用之前已经测试过的代码。本书就将教我们如何编写可复用的代码。

不过，设计模式是一把双刃剑。正确地使用它可以提高系统的适应性，误用则会反过来降低系统的适应性。下面的学习方法有助于我们尽快地掌握设计模式：

1. 了解设计模式

首先通过阅读图书和文章了解设计模式。除了阅读本书以外，还可以参考本书附录中介绍的许多讲解和讨论设计模式的优秀图书和文章。

2. 动手体验设计模式

自己动手编写示例代码，观察代码运行结果。在这个过程中，注意用心去感受代码。

3. 在项目中实践

当认为时机成熟时，可以尝试在项目中运用设计模式。遇到阻力时，可以用书中的知识和自己的理解去说服其他开发人员和项目经理。

4. 总结经验教训

误用设计模式并不可怕，可怕的是一错再错。在每次误用设计模式后都应当总结经验教训，这样才能真正地提高对设计模式的理解。

5. 与其他开发者交流讨论

与其他开发人员，特别是与经验丰富的开发人员交流讨论是快速掌握设计模式的行之有效的方法之一。在讨论候选的几种设计模式到底哪种更好的过程中，时常会出现“一语惊醒梦中人”的情况。

在此衷心希望各位读者朋友们能够爱上设计模式。

杨文轩

2016 年 10 月

引言

大家好，我是结城浩。欢迎阅读《图解设计模式》。

想必大家在编写程序的时候，也曾遇到“咦，好像之前编写过类似的代码”这样的情况。随着开发经验的增加，大家都会在自己的脑海中积累起越来越多的“模式”，然后会将这些“模式”运用于下次开发中。

Eric Gamma、Richard Helm、Ralph Johnson、John Vlissides 等 4 人将开发人员的上述“体会”和“内在积累”整理成了“设计模式”。这 4 人被称为 the Gang of Four，简称 GoF。

GoF 为常用的 23 种模式赋予了“名字”，并按照类型对它们进行了整理，编写成了一本书，这本书就是《设计模式：可复用面向对象软件的基础》（请参见附录 E 中的 [GoF]）。

大家应当都知道，当多个模块组合在一起工作时，接口是非常重要的。其实，这条原则不仅仅适用于计算机，也适用于人。当多位开发人员一起工作的时候，“人”这个接口也非常重要，而这个接口的基础就是“语言”。特别是脱离具体代码、只讨论程序的大致结构时，语言和图示就显得尤其重要。比如，另外一位开发人员提出的改进方案与我的方案究竟是否相同？是不是大框架相同而细节不同呢？如果有无限的时间与耐力，这些问题都是可以通过反复讨论解答出来的。但是，如果借助设计模式的术语来表达想法，我们就可以更加轻松地比较两人的观点，进而使讨论进行得更加顺利。

设计模式为开发人员提供了有益且丰富的词汇，让开发人员可以更容易地理解对方所要表达的意思。

本书将对 GoF 的 23 种设计模式逐一进行讲解，让那些面向对象的初学者也可以很轻松地理解这些设计模式。本书并非仅仅给出枯燥的设计模式理论，还会用 Java 语言编写实现了设计模式的示例程序，并让程序真正地运行起来。我们学习设计模式，并不是为了遥远的将来而打算，而是将它当作是一种有益的技巧，因为它可以帮助我们从全新的角度审视我们每天所编写的代码，从而帮助我们开发出更易于复用和扩展的软件。

本书的特点

◆用 Java 语言编写可实际运行的程序

我们会编写 Java 程序代码来实现 GoF 的 23 种设计模式。为了方便大家通读这些代码，所有的代码都只有 100 行左右，非常精简。而且，所有的代码中都没有“以下代码省略”的部分，且都经过笔者自己编译并运行过。

◆模式名称的讲解

设计模式的名称原本不是汉语，而是英语。开发人员如果不精通英语，就无法由设计模式的名称直接联想到它的作用。因此，本书还会讲解各设计模式的名称是什么意思，以及怎样用汉语表达。这样一来，那些不擅长英语的开发人员也可以很轻松地掌握设计模式。

◆模式之间的关联与练习题

设计模式不需要死记硬背。要想掌握模式，必须要多练习，比如试着在阅读程序时识别出模式，在编写程序时运用模式。因此，必须了解模式之间的关联，并练习运用模式解决具体问题。本书为大家设计了用于学习设计模式的练习题和答案。

◆ Java 语言的相关信息

本书不仅会讲解设计模式，还会向读者展示一些信息以帮助大家深入理解 Java。带有Java符号的内容表示这部分是和 Java 语言相关的信息。

◆模式插图

如果只阅读文字讲解内容，很难掌握这些模式。在本书中，我们在每章的首页中都放了一张图片来直观地展示所要学习的模式，这样可以帮助大家更加轻松地掌握模式。

本书的读者

本书适合以下读者阅读。

- 对面向对象开发感兴趣的人
- 对设计模式感兴趣的人
  （特别是阅读了 GoF 的著作但是难以理解的人）
- 所有 Java 程序员
  （特别是对抽象类和接口的理解不充分的人）

阅读本书需要掌握 Java 语言的基本知识。具体而言，至少需要理解类和接口、字段和方法，并能够编译和运行 Java 源代码。

虽然本书讲解的是设计模式，但必要时也会对 Java 语言的功能进行补充说明，因此读者还可以在阅读本书的过程中加深对 Java 的理解。特别是对于那些对抽象类和接口的目的理解不充分的读者来说，本书具有很大的参考价值。

此外，即使不了解 Java 语言也没关系。如果了解 C++ 语言，同样可以轻松理解本书中的内容。

如果想从零开始学习 Java 语言，建议读者在阅读本书前，先阅读笔者的拙作《Java 语言编程教程（修订版）》[①]（请参见附录 E [Yuki03]）。

另外，建议学习完本书的读者再去学习一下《图解设计模式：多线程》[②]（请参见附录 E[Yuki02]）。

本书的结构

本书结构如下所示，各章基本上与 GoF 设计模式的章节相对应。但是笔者对设计模式的分类与 GoF 不同，因此章节划分也不尽相同。关于 GoF 对设计模式的分类，请参见附录 C。

- 在**第 1 部分“适应设计模式”**中，我们将学习一些比较容易理解的设计模式，并以此来适应

① 原书名为『改訂版　Java 言語プログラミングレッスン』，尚无中文版。——译者注

② 原书名为『Java 言語で学ぶデザインパターン入門　マルチスレッド編』，人民邮电出版社即将引进出版。——译者注

设计模式的概念。

- 在第 1 章“Iterator 模式——一个一个遍历”中，我们将要学习从含有多个元素的集合中将各个元素逐一取出来的 Iterator 模式。
- 在第 2 章“Adapter 模式——加个‘适配器’以便于复用”中，我们将要学习 Adapter 模式，它可以用来连接具有不同接口（API）的类。

- 在**第 2 部分“交给子类”**中，我们将学习与类的继承相关的设计模式。
  - 在第 3 章“Template Method 模式——将具体处理交给子类”中，我们将要学习在父类中定义处理框架，在子类中进行具体处理的 Template Method 模式。
  - 在第 4 章“Factory Method 模式——将实例的生成交给子类”中，我们将要学习在父类中定义生成接口的处理框架，在子类中进行具体处理的 Factory Method 模式。
- 在**第 3 部分“生成实例”**中，我们将学习与生成实例相关的设计模式。
  - 在第 5 章“Singleton 模式——只有一个实例”中，我们将要学习只允许生成一个实例的 Singleton 模式。
  - 在第 6 章“Prototype 模式——通过复制生成实例”中，我们将要学习复制原型接口并生成实例的 Prototype 模式。
  - 在第 7 章“Builder 模式——组装复杂的实例”中，我们将要学习通过各个阶段的处理以组装出复杂实例的 Builder 模式。
  - 在第 8 章“Abstract Factory 模式——将关联零件组装成产品”中，我们将要学习像在工厂中将各个零件组装成产品那样生成实例的 Abstract Factory 模式。
- 在**第 4 部分“分开考虑”**中，我们将学习分开考虑易变得杂乱无章的处理的设计模式。
  - 在第 9 章“Bridge 模式——将类的功能层次结构与实现层次结构分离”中，我们将要学习按照功能层次结构与实现层次结构把一个两种扩展（继承）混在一起的程序进行分离，并在它们之间搭建桥梁的 Bridge 模式。
  - 在第 10 章“Strategy 模式——整体地替换算法”中，我们将要学习 Strategy 模式，它可以帮助我们整体地替换算法，使我们可以更加轻松地改善算法。
- 在**第 5 部分“一致性”**中，我们将学习能够让两个看上去不同的对象的操作变得统一，以及在不改变处理方法的前提下增加功能的设计模式。另外，我们还要学习“委托”。
  - 在第 11 章“Composite 模式——容器与内容的一致性”中，我们将要学习让容器和内容具有一致性，从而构建递归结构的 Composite 模式。
  - 在第 12 章“Decorator 模式——装饰边框与被装饰物的一致性”中，我们将要学习让装饰边框与被装饰物具有一致性，并可以任意叠加装饰边框的 Decorator 模式。
- 在**第 6 部分“访问数据结构”**中，我们将学习能够漫步数据结构的设计模式。
  - 在第 13 章“Visitor 模式——访问数据结构并处理数据”中，我们将要学习在访问数据结构的同时重复套用相同操作的 Visitor 模式。
  - 在第 14 章“Chain of Responsibility 模式——推卸责任”中，我们将要学习可以处理连接在一起的多个对象中某个地方的 Chain of Responsibility 模式。
- 在**第 7 部分“简单化”**中，我们将学习可以让类关系简单的设计模式。
  - 在第 15 章“Facade 模式——简单窗口”中，我们将要学习 Facade 模式，该模式并不是单独地控制那些错综复杂地关联在一起的多个类，而是通过配置一个窗口类来改善系统整体的可操作性。

  - 在第 16 章“Mediator 模式——只有一个仲裁者”中，我们将要学习可以不与多个复杂的类打交道，而是准备一个窗口，然后通过与这个窗口打交道来简化程序的 Mediator 模式。
- 在**第 8 部分“管理状态”**中，我们将学习与状态相关的设计模式。
  - 在第 17 章“Observer 模式——发送状态变化通知”中，我们将要学习将状态发生变化的类和发送状态变化通知的类分开实现的 Observer 模式。
  - 在第 18 章“Memento 模式——保存对象状态”中，我们将要学习可以保存对象现在的状态，并可以根据情况撤销操作，将对象恢复到以前状态的 Memento 模式。
  - 在第 19 章“State 模式——用类表示状态”中，我们将要学习用类来表现状态，以减少 `switch` 语句的 State 模式。
- 在**第 9 部分“避免浪费”**中，我们将学习可以避免浪费、提高处理效率的设计模式。
  - 在第 20 章“Flyweight 模式——共享对象，避免浪费”中，我们将要学习当多个地方有重复对象时，通过共享对象来避免浪费的 Flyweight 模式。
  - 在第 21 章“Proxy 模式——只在必要时生成实例”中，我们将要学习除非必须“本人”处理，否则就只使用代理类来负责处理的 Proxy 模式。
- 在**第 10 部分“用类来表现”**中，我们将学习用类来表现特殊东西的设计模式。
  - 在第 22 章“Command 模式——命令也是类”中，我们将要学习用类来表现请求和命令的 Command 模式。
  - 在第 23 章“Interpreter 模式——语法规则也是类”中，我们将要学习用类来表现语法规则的 Interpreter 模式。

本书中的示例代码

示例代码的获取方法

本书的示例代码可以从以下网址下载（点击“随书下载”）：

```
http://www.ituring.com.cn/book/1811
```

详细信息请参见附录 B。

从 Main 类启动示例代码

在 Java 中，只要类中定义了以下方法，就可以将该类作为程序的起点：

```
public static void main(string[])
```

但是在本书中，为了使读者能够更容易理解代码，各章的示例程序都使用 `Main` 类作为程序的起点。

关于本书中术语的注意事项

接口和 API

接口这个术语有多个意思。

一般而言，在提到“某个类的接口”时，多是指该类所持有的方法的集合。当想要对该类进行某些操作时，需要调用这些方法。

但是在 Java 中，也将“使用关键字 `interface` 声明的代码”称为接口。

这两个“接口”的意思有些相似，在使用时容易混乱，因此本书中采用以下方式加以区分。

- 接口（API）：通常的意思（API 是 application programming interface 的缩写）
- 接口：使用关键字 `interface` 声明的代码

模式、类和角色

在本书中，**模式**这个词表示设计模式的意思。例如，“GoF 一共在书中整理了 23 种模式”指的就是“GoF 一共在书中整理了 23 种设计模式”。另外，我们会将名为 Memento 的设计模式简称为“Memento 模式”。

类是指 Java 中的类，即以 `class` 关键字定义的程序。例如，在书中会有“这段程序中定义的是 `Gamer` 类”这种描述；而“`Memento` 类”则是指在程序上用 `class Memento { ... }` 定义的代码。

角色是本书中特有的说法。它是指模式（设计模式）中出现的类、接口和实例在模式中所起的作用。例如，在书中会有“由 `Gamer` 类扮演 Originator 角色”这种描述。当然，也存在角色的名字与类和接口的名字不一致的情况。

此处的内容很繁琐，但是当大家阅读本书时，就会理解笔者在这里想要表达的意思了。

致谢

首先需要向整理出设计模式的 Eric Gamma、Richard Helm、Ralph Johnson、John Vlissides 这 4 人表示感谢。

然后，还要向阅读笔者拙作，包括图书、连载杂志和电子邮件杂志的读者们表示感谢。另外，还要向笔者 Web 主页上的朋友们表示感谢。

笔者在编写本书的原稿、程序以及图示的过程中，也同时将它们公布在了互联网上，以供大家评审。在互联网上招募的评审人员不限年龄、国籍、性别、住址、职业，所有交流都是通过电子邮件和网络进行的。在此，笔者要向参与本书评审的朋友们表示感谢，特别是对给予了我宝贵意见、改进方案，向我反馈错误以及一直鼓励我的以下各位表示我最真挚的感谢（按五十音图顺序排列）：

新真千惠、池田史子、石井胜、石田浩二、井芹义博、宇田川胜俊、川崎昌博、榊原知香子、砂生贵光、佐藤贵行、铃木健司、铃木信夫、竹井章、藤森知郎、前田恭男、前原正英、三宅喜义、谷内上智春、山城俊介。

此外，对其他参与了评审工作的人员也一并表示感谢。

另外，还要向软银出版股份有限公司的图书总编野泽喜美男和第一图书编辑部的松本香织表示感谢。当我们一起商量这本书的选题时，他们都表示“这一定会是一本好书”，这让我倍受鼓舞。

最后要感谢我最爱的妻子和两个儿子，以及总是精神满满地支持我的岳母大人。

结城浩

2001 年 3 月 于武藏野

写于“修订版”前

《图解设计模式》一书自 2001 年初版发行以来，承蒙各位读者的厚爱，在此再次向各位读者表达我最真挚的感谢。

在这次“修订版”中，笔者重新全面地审视了本书的内容和表述。在修订中，也参考了读者朋友们发送给我的无数反馈意见和建议，真心谢谢你们。希望本书也能在读者朋友的工作和学习中发挥些许作用。

结城浩

2004 年 6 月

关于本书官网

读者可以从以下网址获取本书的最新信息：

http://www.hyuki.com/dp

该网址是作者本人运营的网站之一。

问答 Web

请将您读完这本书后的感想及意见发送到以下网址。

http://www.ituring.com.cn/book/1811

关于 UML

UML

UML 是让系统可视化、让规格和设计文档化的表现方法，它是 Unified Modeling Language（统一建模语言）的简称。

本书使用 UML 来表现各种设计模式中类和接口的关系，所以我们在这里稍微了解一下 UML，以方便后面的阅读。但是请大家注意，在说明中我们使用的是 Java 语言的术语。例如讲解时我们会用 Java 中的“字段”（field）取代 UML 中的“属性”（attribute），用 Java 中的“方法”（method）取代 UML 中的“操作”（operation）。

UML 标准的内容非常多，本节只对书中使用到的 UML 内容进行讲解。如果想了解更多 UML 内容，请访问以下网站。UML 的规范书也可以从该网站下载。

- UML Resource Page

  `http://www.omg.org/uml/`

类图

UML 中的**类图**（Class Diagram）用于表示类、接口、实例等之间相互的静态关系。虽然名字叫作类图，但是图中并不仅仅只有类。

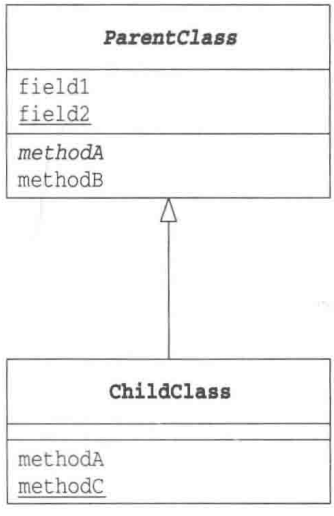
类与层次结构

图 0-1 展示了一段 Java 程序及其对应的类图。

图 0-1　展示类的层次关系的类图

```
abstract class ParentClass {
    int field1;
    static char field2;
    abstract void methodA();
    double methodB() {
        // ...
    }
}

class ChildClass extends ParentClass {
    void methodA() {
        // ...
    }
    static void methodC() {
        // ...
    }
}
```



该图展示了 ParentClass 和 ChildClass 两个类之间的关系，其中的空心箭头表明了两者之间的层次关系。箭头**由子类指向父类**，换言之，这是表示继承（extends）的箭头。

ParentClass 是 ChildClass 的父类，反过来说，ChildClass 是 ParentClass 的子类。父类也称为基类或超类，子类也称为派生类。

图中的长方形表示类，长方形内部被横线自上而下分为了如下 3 个区域。

- **类名**
- **字段名**
- **方法名**

有时，图中除了会写出类名、字段名和方法名等信息外，还会写出其他信息（可见性、方法的参数和类型等）。反之，有时图中也会省略所有不必要的项目（因此，我们无法确保一定可以根据类图生成源程序）。

abstract 类（抽象类）的名字以斜体方式显示。例如，在图 0-1 中 ParentClass 是抽象类，因此它的名字以斜体方式显示。

static 字段（静态字段）的名字带有下划线。例如，在图 0-1 中 field2 是静态字段，因此名字带有下划线。

abstract 方法（抽象方法）的名字以斜体方式显示。例如，在图 0-1 中 methodA 是抽象方法，因此它以斜体方式显示。

static 方法（静态方法）的名字以下划线显示。例如，在图 0-1 中 ChildClass 类的 methodC 是类的静态方法，因此它的名字带有下划线。

▶▶ 小知识：Java 术语与 C++ 术语

Java 术语跟 C++ 术语略有不同。Java 中的字段相当于 C++ 中的成员变量，而 Java 中的方法相当于 C++ 中的成员函数。

▶▶ 小知识：箭头的方向

UML 中规定的箭头方向是从子类指向父类。可能会有人认为子类是以父类为基础的，箭头从父类指向子类会更合理。

关于这一点，按照以下方法去理解有助于大家记住这条规则。在定义子类时需要通过 extends 关键字指定父类。因此，子类一定知道父类的定义，而反过来，父类并不知道子类的定义。只有在知道对方的信息时才能指向对方，因此箭头方向是从子类指向父类。

接口与实现

图 0-2 也是类图的示例。该图表示 PrintClass 类实现了 Printable 接口。为了强调接口与抽象类的相似性，本书的类图中会以斜体方式显示接口的名字。不过在其他书的类图中，接口名可能并非以斜体显示。空心箭头代表了接口与实现类的关系，箭头**从实现类指向接口**。换言之，这是表示实现（implements）的箭头。

UML 以 <<interface>> 表示 Java 的接口。

图 0-2 展示接口与实现类的类图

```
interface Printable {
    abstract void print();
    abstract void newPage();
}

class PrintClass implements Printable {
    void print() {
        // ...
    }
    void newPage() {
        // ...
    }
}
```

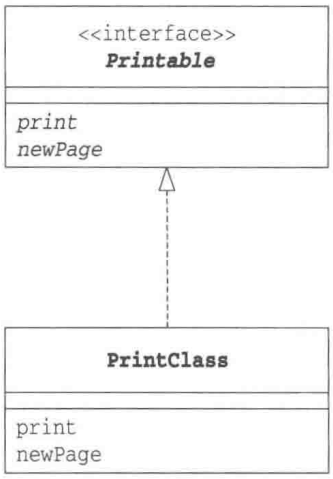
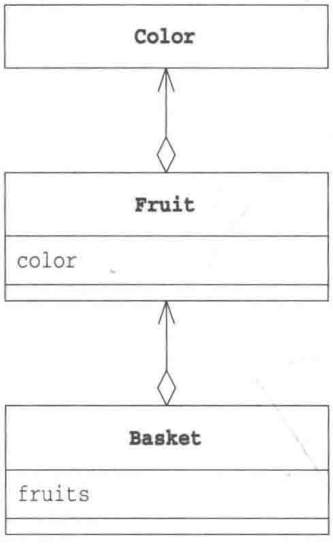


图 0-3 展示聚合关系的类图

```
class Color {
    // ...
}

class Fruit {
    Color color;
    // ...
}

class Basket {
    Fruit[] fruits;
    // ...
}
```



聚合

图 0-3 也是类图的示例。

该图展示了 Color（颜色）、Fruit（水果）、Basket（果篮）这 3 个类之间的关系。Basket 类中的 fruits 字段是可以存放 Fruit 类型数据的数组，在一个 Basket 类的实例中可以持有多个 Fruit 类的实例；Fruit 类中的 color 字段是 Color 类型，一个 Fruit 类实例中只能持有一个 Color 类的实例。通俗地说就是在篮子中可以放入多个水果，每个水果都有其自身的颜色。

我们将这种“持有”关系称为**聚合**（aggregation）。只要在一个类中持有另外一个类的实例——无论是一个还是多个——它们之间就是聚合关系。就程序上而言，无论是使用数组、java.util.Vector 或是其他实现方式，只要在一个类中持有另外一个类的实例，它们之间就是聚合关系。

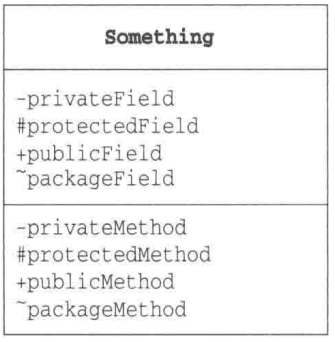
在 UML 中，我们使用带有空心菱形的实线表示聚合关系，因此可以进行联想记忆，将聚合关系想象为在空心菱形的器皿中装有其他物品。

可见性（访问控制）

图 0-4 也是类图的示例。

图 0-4 标识出了可见性的类图

```
class Something {
    private   int privateField;
    protected int protectedField;
    public    int publicField;
    int packageField;
    private   void privateMethod() {
    }
    protected void protectedMethod() {
    }
    public    void publicMethod() {
    }
    void packageMethod() {
    }
}
```



该图标识出了方法和字段的可见性。在 UML 中可以通过在方法名和字段名前面加上记号来表示可见性。

“+”表示 public 方法和字段，可以从类外部访问这些方法和字段。

“-”表示 private 方法和字段，无法从类外部访问这些方法和字段。

“#”表示 protect 方法和字段，能够访问这些方法和字段的只能是该类自身、该类的子类以及同一包中的类。

“~”表示只有同一包中的类才能访问的方法和字段。

类的关联

可以在类名前面加上黑三角表示类之间的关联关系，如图 0-5 所示。

图 0-5 类的关联