# <<Java设计模式>>

## 图书基本信息

书名：<<Java设计模式>>

13位ISBN编号：9787121178269

10位ISBN编号：7121178265

出版时间：2012-9

出版时间：电子工业出版社

作者：[美]梅特斯克（Metsker,S.J.）,[美]维克（Wake,W.J.）

译者：张逸,史磊

# <<Java设计模式>>

## 内容概要

本书通过一个完整的Java项目对经典著作Design Patterns一书介绍的23种设计模式进行了深入分析与讲解，实践性强，却又不失对模式本质的探讨。
本书创造性地将这些模式分为5大类别，以充分展现各个模式的重要特征，并结合UML类图与对应的Java程序，便于读者更好地理解。
全书给出了大量的练习，作为对读者的挑战，以启发思考，督促读者通过实践练习的方式来掌握设计模式。
同时，作者又给出了这些练习的参考答案，使读者可以印证比较，找出自己的不足，提高设计技能。

## 作者简介

# <<Java设计模式>>

## 书籍目录

## 章节摘录


一旦访问到卸载缓冲池，如果unloadBuffer类的9etTubs0方法抛出异常，这段代码就会出现问题。
这严重违反了LSP：当你将unloadBuffer对象当做Machine对象来使用时，程序可能会崩溃！
假设不抛出异常，我们需要简单地忽略对unloadBuffer类中9etTubs0和addTubO的调用。
这一做法依然违反了LSP，因为在你给机器添加一个原料桶时，这个原料桶可能会消失！
违反LSP并不一定是设计缺陷。
针对Oozinoz公司的这种情况，需要权衡一下，让Machi ne类拥有大多数机器的行为和违反LSP原则，究竟哪个价值更大。
重要的一点是意识到LSP，并且清楚为什么其他设计可能违反了LSP。
迪米特法则 在20世纪80年代后期，美国东北大学Demeter Project的成员尝试编辑了一些规则，用于确保健康的面向对象编程。
项目团队将这些规则称为迪米特法则（Law ofDemeter，LoD）。
KarlLieberherr和Ian Holland在Assuring Good Style for Object—Oriented Programs一文中全面地总结了这一规则。
声明认为：非正式地说，迪米特法则要求每个方法只能给有限的对象发送消息，包括参数对象、[this]伪变量，以及[this]的直接子部分。
文章随后给出了该法则的正式定义。
相对于完全理解迪米特法则的意图，识别设计是否违反该法则更加容易。
假设你有一个Material Manager对象，该对象有一个方法接收一个Tub对象作为参数。
Tub对象有一个Location属性，该属性返回一个Machi ne对象，用以表示桶被放在哪个位置。
假设在Material Manager的方法中，想要知道机器是否可用，你可能会写如下代码：如果这个挑战仅仅让你觉得形如a.b.c的表达式是错误的，那就降低了迪米特法则的价值。
事实上，Lieberherr和Holland希望迪米特法则能够进一步确定无误地回答这样的问题：是否可以遵循某种公式或法则来写出更好的面向对象程序？
这篇阐释迪米特法则的早期论文非常值得我们拜读。
就像Liskov替换原则一样，如果知道并遵循这些规则，它就会帮助你写出更好的代码，一旦你的设计违背了这些原则，就能够及时获知。
你会发现，遵循这些指导原则，扩展性就能够自然而然产生好的代码。
但是，对于很多开发者而言，面向对象的开发依然是一门艺术。
对代码库的艺术扩展源于艺术家们的实践，而这些大师们依然在不断地改进他们的艺术。
重构就是诸多技艺中的一种工具，它可以在不改变既有功能的前提下，改善代码的质量。
消除代码的坏味道 你可能寄希望于Liskov替换原则与迪米特法则，能够永远地防止你写出拙劣的代码。
不过，更实际的做法是运用这些准则来帮助发现代码的坏味道，然后修复它。
这是一种通用的实践：先写出可工作的代码，然后找出代码的问题，并且修复它，以提升代码质量。
但是该如何准确地识别问题呢？
答案就是找到代码的坏味道。
在Refactoring：Improving the Design of ExitingCode（由Fowler等人在1999年编写）一书中描述了22种坏味道，并给出了相应的重构手法。

## 编辑推荐

《Java设计模式(第2版)》适合各个层次的Java开发人员与设计人员阅读，也可以作为学习Java与设计模式的参考读物或教材。

## 版权说明