

<<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

版权说明：本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问：<http://www.tushu007.com>

<<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。

<<Java设计模式>>

作者简介

<<Java设计模式>>

书籍目录

序	xv
第1章 绪论	1
为何需要模式	1
为何需要设计模式	2
为何选择Java	3
UML	3
挑战	4
本书的组织	4
欢迎来到Oozinoz公司	6
小结	6
第1部分 接口型模式	
第2章 接口型模式介绍	8
接口与抽象类	8
接口与职责	10
小结	11
超越普通接口	12
第3章 适配器 (Adapter) 模式	13
接口适配	13
类与对象适配器	17
JTable对数据的适配	20
识别适配器	24
小结	25
第4章 外观 (Facade) 模式	27
外观类、工具类和示例类	27
重构到外观模式	29
小结	38
第5章 合成 (Composite) 模式	39
常规组合	39
合成模式中的递归行为	40
组合、树与环	42
含有环的合成模式	47
环的影响	50
小结	51
第6章 桥接 (Bridge) 模式	52
常规抽象：桥接模式的一种方法	52
从抽象到桥接模式	54
使用桥接模式的驱动器	57
数据库驱动	57
小结	59
第2部分 职责型模式	
第7章 职责型模式介绍	62
常规的职责型模式	62
根据可见性控制职责	64
小结	65
超越普通职责	65

<<Java设计模式>>

第8章 单例 (Singleton) 模式67

单例模式机制67

单例和线程68

识别单例70

小结71

第9章 观察者 (Observer) 模式72

经典范例：GUI中的观察者模式72

模型/视图/控制器76

维护Observable对象82

小结84

第10章 调停者 (Mediator) 模式85

经典范例：GUI调停者 (Mediator) 85

关系一致性中的调停者模式89

小结96

第11章 代理 (Proxy) 模式97

经典范例：图像代理97

重新思考图片代理102

远程代理104

动态代理109

小结114

第12章 职责链 (Chain of Responsibility) 模式115

现实中的职责链模式115

重构为职责链模式117

固定职责链119

没有组合结构的职责链模式121

小结121

第13章 享元 (Flyweight) 模式122

不变性122

抽取享元中不可变的部分123

共享享元125

小结128

第3部分 构造型模式

第14章 构造型模式介绍130

构造函数的挑战130

小结132

超出常规的构造函数132

第15章 构建者 (Builder) 模式134

常规的构建者134

在约束条件下构建对象137

可容错的构建者139

小结140

第16章 工厂方法 (Factory Method) 模式141

经典范例：迭代器141

识别工厂方法142

控制要实例化的类143

并行层次结构中的工厂方法模式145

小结147

<<Java设计模式>>

- 第17章 抽象工厂 (Abstract Factory) 模式148
 - 经典范例：图形用户界面工具箱148
 - 抽象工厂和工厂方法153
 - 包和抽象工厂157
 - 小结157
- 第18章 原型 (Prototype) 模式158
 - 作为工厂的原型158
 - 利用克隆进行原型化159
 - 小结162
- 第19章 备忘录 (Memento) 模式163
 - 经典范例：使用备忘录模式执行撤销操作163
 - 备忘录的持久性170
 - 跨会话的持久性备忘录170
 - 小结174
- 第4部分 操作型模式
- 第20章 操作型模式介绍176
 - 操作和方法176
 - 签名177
 - 异常178
 - 算法和多态179
 - 小结180
 - 超越常规的操作181
- 第21章 模板方法 (Template Method) 模式182
 - 经典范例：排序182
 - 完成一个算法186
 - 模板方法钩子188
 - 重构为模板方法模式189
 - 小结191
- 第22章 状态 (State) 模式193
 - 对状态进行建模193
 - 重构为状态模式197
 - 使状态成为常量201
 - 小结203
- 第23章 策略 (Strategy) 模式204
 - 策略建模204
 - 重构到策略模式207
 - 比较策略模式与状态模式211
 - 比较策略模式和模板方法模式211
 - 小结212
- 第24章 命令 (Command) 模式213
 - 经典范例：菜单命令213
 - 使用命令模式来提供服务216
 - 命令钩子217
 - 命令模式与其他模式的关系219
 - 小结220
- 第25章 解释器 (Interpreter) 模式221
 - 一个解释器示例221

<<Java设计模式>>

解释器、语言和解析器233
小结234
第5部分 扩展型模式
第26章 扩展型模式介绍236
面向对象设计的原则236
Liskov替换原则237
迪米特法则238
消除代码的坏味道239
超越常规的扩展240
小结241
第27章 装饰器 (Decorator) 模式242
经典范例：流和输出器242
函数包装器250
装饰器模式和其他设计模式的关系257
小结258
第28章 迭代器 (Iterator) 模式259
普通的迭代259
线程安全的迭代261
基于合成结构的迭代267
小结277
第29章 访问者 (Visitor) 模式278
访问者模式机制278
常规的访问者模式280
Visitor环286
访问者模式的危机290
小结292
附录A 指南293
附录B 答案297
附录C Oozinoz源代码366
附录D UML概览369
参考文献375

章节摘录

版权页：插图：假设其他的开发者编写了一个方法，用来查询车间里所有机器拥有的原料桶的集合。

一旦访问到卸载缓冲池，如果unloadBuffer类的getTubs方法抛出异常，这段代码就会出现异常。

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getTubs和addTub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Machine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 of Demeter，LoD）。

Karl Lieberherr和Ivan 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Machine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 Existing Code（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。

<<Java设计模式>>

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

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:<http://www.tushu007.com>