# Program Development in Java Preface

从 farseerfc.wordpress.com 导入

# 程序开发原理

# ——抽象、规格与面向对象设计

Barbara Liskov 、John Guttag 著

杨嘉晨 等译

# 关于翻译风格：

多年来阅读计算机类的著作及译作，感觉总体的困难在于一大堆没有标准译名的技术术语。由于通行于工业界和学术界的还是英文原名和术语，我决定保留大量的英文术语。这样的翻译风格借鉴于台湾著名的译者和作者侯捷先生。对于译与不译的权衡，主要考虑阅读的流畅，以及读者的理解能力，或许难免带有一些主观色彩。

# 前言 Preface

构建产品级质量的程序——可以在很长一段时间内使用的程序——众所周知是极其困难的。本书的目标就是改善程序员解决这项任务的效率。我希望读者在阅读本书之后成为一名好程序员。我相信本书的成功在于改善编程技巧，因为我的学生告诉我这已经发生在他们身上。

怎么才算是一名好程序员？是产生整个程序产品的效率。关键是要在每一阶段减少浪费掉的努力。解决的方法包括：在开始编写代码之前就仔细考虑你的实现方案，通过未雨绸缪的方法来编写代码，使用严格的测试在早期发现错误，以及仔细注意模块化编程，这样当错误出现时，只需要改动极少数代码就可以修正整个程序。本书涉及所有这些领域的技术。

模块化编程(Modularity)是编写好程序的关键。把程序分解成许多小模块，每一个模块通过良好定义的狭窄接口和别的模块交互作用(interact)。有了模块化，可以修正一部分程序中的错误而不考虑程序的其他部分，而且可以仅仅理解一部分程序而不必理解整个程序。没有模块化，程序是一大堆有着错综复杂的相互关系的部分的拼凑。很难去领悟和修改这样一个程序，同样也很难让它正常工作。

因此本书的重点在于创建模块化的程序：怎样把程序组织成一系列精心挑选的模块。本书认为模块化就是抽象(abstraction)。每一个模块意味着一个抽象，比如说指引一系列文档中的关键字的目录，或者在文档中使用目录来查找匹配某个问题的文档的过程。着重强调面向对象编程思想——在程序中使用数据抽象和对象的思想。

这本书使用Java作为它的编程示例的语言。我们没有假定读者已经熟悉Java。尽管可能没什么价值，但是本书中的思想是语言无关的，并且可以在任何语言的编程中使用。

# 怎样使用这本书？ How Can the Book Be Used

本书《程序开发原理》有两种使用方法。其一是作为课本教材，讲述如何用面向对象的方法来设计和实现复杂系统；其二是编程专家使用，帮助他们改善编程技能，增进他们的关于模块化和Object-Oriented(面向对象)设计的知识。

作为教材使用时，本书一般作为第二或第三门程序设计课程。我们已经在MIT使用本书很多年，给大一大二的本科生教授第二门编程课。在这一阶段，学生们已经知道怎样编写小程序。课程在两方面利用这一点：让学生更仔细地思考小程序，以及教他们如何利用小程序作为组件构建大型程序。这本书也可以在专业（如软件工程）后期教学中使用。

建立在本书基础上的课程适合于所有计算机科学专业。尽管许多学生可能永远不会成为真正的大型程序的设计师，他们可以在开发部门工作，在那儿他们负责设计和实现能与整个结构耦合的子系统。模块化设计的子系统是这种任务中心，这对那些从事大型程序设计任务的人来说也同样重要。

# 这本书讲什么？What Is This Book About

通观全篇三分之二的书致力于讨论在构建独立的程序模块时产生的问题，剩下的部分讨论怎样运用这些模块构建大型程序。

## 程序模块Program Modules

这一部分的书集中讨论抽象机制(abstraction mechanism)。它讨论procedure(子程序)和exception(异常)，数据抽象，遍历(iteration)抽象，数据抽象系列(family)以及多态(polymorphic)抽象。

在对抽象的讨论中，三个步骤是重要的。首先是决定被抽象的东西到底是什么：它提供给它的用户哪些行为。创造抽象是设计的关键，因此本书讨论如何在众多选择中挑选，以及怎样才能创造出好的抽象。

第二步是通过为一个抽象制定一个规格(specification)来获取它的意义。如果没有一些描述，一个抽象就会含糊不清，而变得没有使用价值。specification则提供了需要的描述。本书定义了一种specification的格式，讨论了一份好的specification应有的属性，并且提供了许多示例。

第三步是实现抽象。本书讨论怎样设计一份实现，以及在简洁性和执行性能之间怎样权衡利弊。书中强调封装(encapsulation)的重要性以及在一份实现中履行规格中定义的行为的重要性。书中同样提供一些技术——尤其是不变式断言(representation invariant)和抽象函数

(abstraction function)——来帮助读者理解代码和它的原因。不变式断言和抽象函数都实现到尽可能的程度，这对于除错和调试很有用。

关于类型层次(type hierarchy)的材料注重讨论使用它作为抽象的技术——一种把相关联的一组数据抽象归入同一系列的技术。这里很重要的一点是，是否应当将一个类型作为另一个类型的子类。本书定义了替换原则——通过比较子类和父类的specification，来决定是否建立子类关系的方法[1]。

本书同样涉及除错和调试。书中讨论怎样得到足够数量的测试情况，来准备通过黑箱和白箱测试，它同样强调了复查(regression)测试的重要性。

## 编写大型程序 Programming in the Large

本书的其后部分讲解怎样用模块化的方法设计和实现大型程序。它建立在前文有关abstraction和specification的材料的基础之上。

编写大型程序涵盖四个主要议题。首先讲解需求分析——怎样才能领悟程序中需要什么。本书讨论怎样实施需求分析，也讨论书写产生的需求规格的方式，通过使用一种描述程序的抽象阶段的数据模型。使用这种模型将产生一份更为正式的specification，同时它也使需求检查更加严格，这样可以更好的领悟需求。

编写大型程序的第二项议题是程序设计，这通常是一个循序渐进的过程。设计过程围绕构建有用的抽象来组织，这些抽象作为整个程序之中理想的构建组建。这些抽象在设计时被仔细的编写规格，这样当程序实现时，那些实现抽象的模块可以独立地开发。这种设计使用设计笔记编写文档，包括描述整个程序结构的模块间依赖性的图示。

第三项议题是实现和测试。本书讨论了前置设计分析对于实现的必要性，以及怎样进行设计复审。它同样讨论了设计和实现的顺序。这一部分比较了自顶而下与自底而上的组织方式，讨论如何使用驱动程序和占位程序[2](stub)，并且强调了制定一个事先的顺序策略的必要性，以满足开发组织和客户的需求。

本书以一章设计模式(design pattern)结束。一些模式在前面的章节介绍过，比如遍历抽象是算法的主要组建。最后的章节讨论前文中没有涉及到的模式。希望它作为这一教材的介绍。有兴趣的读者可以继续阅读其它书中更完善的讨论[3]。

---

[1] 译注：如果子类的specification包括了所有父类的specification，就是说父类的要求也是子类的要求，或者子类的要求更为严格，那么可以建立父子关系。而替换原则的说法是，对于具有父子关系的类，任何需要一个父类对象的地方，都可以替换为一个子类对象。

[2] 译注：在测试某一组建时，由于其余组建还未实现，这一组建与其余组建的接口衔接部分无法工作。此时可以针对这一组建编写其余组建的占位程序(stub)，预留出接口的衔接代码。占位代码通常不做任何有价值的事情，只报告组建的衔接部位工作正常。

[3] 译注：作者指的是设计模式的开山之作——《Design Patterns—Elements of Reusable Object-Oriented Software》,作者为设计模式界著名的“四人帮”GoF(Gang of Four)。此书详尽讨论了三大类共23个广泛使用的设计模式的适用范围、依存关系、实现细节以及已有的应用领域等问题。书中以C++和Smalltalk为示例语言，不过书中所涉及的模式适用于所有面向对象的语言。