

Pragmatic Unit Testing in Java 8 with JUnit

Jeff Langr

with Andy Hunt
Dave Thomas

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

Contents

Foreword	ix
Preface	xi

Part I — Unit-Testing Foundations

1. Building Your First JUnit Test	3
Reasons to Write a Unit Test	3
Learning JUnit Basics: Our First Passing Test	4
Arrange, Act, and Assert Your Way to a Test	10
Is the Test Really Testing Anything?	12
After	12
2. Getting Real with JUnit	13
Understanding What We're Testing: The Profile Class	13
Determining What Tests We Can Write	15
Covering One Path	17
Tackling a Second Test	19
Initializing Tests with @Before Methods	19
How Ya Feelin' Now?	22
After	23
3. Digging Deeper into JUnit Assertions	25
Assertions in JUnit	25
Three Schools for Expecting Exceptions	31
After	34
4. Organizing Your Tests	35
Keeping Tests Consistent with AAA	35
Testing Behavior Versus Testing Methods	36
Relationship Between Test and Production Code	37

The Value of Focused, Single-Purpose Tests	40
Tests as Documentation	41
More on @Before and @After (Common Initialization and Cleanup)	43
Green Is Good: Keeping Our Tests Relevant	45
After	47

Part II — Mastering Manic Mnemonics!

5. FIRST Properties of Good Tests	51
FIRST It Helps to Remember That Good Tests Are FIRST	51
[F]IRST: [F]ast!	52
F[!]RST: [!]solate Your Tests	56
FI[R]ST: Good Tests Should Be [R]epeatable	57
FIR[S]T: [S]elf-Validating	59
FIRS[T]: [T]imely	61
After	62
6. What to Test: The Right-BICEP	63
[Right]-BICEP: Are the Results Right?	63
Right-[B]ICEP: Boundary Conditions	65
Remembering Boundary Conditions with CORRECT	67
Right-B[!]CEP: Checking Inverse Relationships	68
Right-BI[C]EP: Cross-Checking Using Other Means	70
Right-BIC[E]P: Forcing Error Conditions	71
Right-BICE[P]: Performance Characteristics	71
After	73
7. Boundary Conditions: The CORRECT Way	75
[C]ORRECT: [C]onformance	76
C[O]RRECT: [O]rdering	77
CO[R]RECT: [R]ange	78
COR[R]ECT: [R]eference	85
CORR[E]CT: [E]xistence	86
CORRE[C]T: [C]ardinality	87
CORREC[T]: [T]ime	89
After	91

Part III — The Bigger Design Picture

8.	Refactoring to Cleaner Code	95
	A Little Bit o' Refactor	95
	Finding Better Homes for Our Methods	98
	Automated and Manual Refactorings	100
	Taking Refactoring Too Far?	102
	After	105
9.	Bigger Design Issues	107
	The Profile Class and the SRP	107
	Extracting a New Class	109
	Command-Query Separation	114
	The Cost of Maintaining Unit Tests	115
	Other Design Thoughts	118
	After	121
10.	Using Mock Objects	123
	A Testing Challenge	123
	Replacing Troublesome Behavior with Stubs	125
	Changing Our Design to Support Testing	128
	Adding Smarts to Our Stub: Verifying Parameters	128
	Simplifying Testing Using a Mock Tool	130
	One Last Simplification: Introducing an Injection Tool	131
	What's Important to Get Right When Using Mocks	133
	After	134
11.	Refactoring Tests	135
	Searching for an Understanding	135
	Test Smell: Unnecessary Test Code	137
	Test Smell: Missing Abstractions	138
	Test Smell: Irrelevant Information	140
	Test Smell: Bloated Construction	142
	Test Smell: Multiple Assertions	143
	Test Smell: Irrelevant Details in Test	144
	Test Smell: Misleading Organization	146
	Test Smell: Implicit Meaning	147
	Adding a New Test	148
	After	149

Part IV — The Bigger Unit-Testing Picture

12.	Test-Driven Development	153
	The Primary Benefit of TDD	153
	Starting Simple	154
	Adding Another Increment	157
	Cleaning Up Our Tests	158
	Another Small Increment	161
	Supporting Multiple Answers: A Small Design Detour	162
	Expanding the Interface	164
	Last Tests	166
	Tests As Documentation	167
	The Rhythm of TDD	169
	After	169
13.	Testing Some Tough Stuff	171
	Testing Multithreaded Code	171
	Testing Databases	180
	After	186
14.	Testing on a Project	187
	Coming up to Speed	187
	Getting on the Same Page with Your Team	188
	Convergence with Continuous Integration	190
	Code Coverage	192
	After	196
A1.	Setting Up JUnit in IntelliJ IDEA and NetBeans	197
	IntelliJ IDEA	198
	NetBeans	202
	Index	207

Foreword

Some time after Dave Thomas and I (Andy Hunt) wrote *The Pragmatic Programmer* and the first edition of *Programming Ruby*, we turned our attention to the most basic needs of modern software developers.

We came up with the idea of *The Pragmatic Starter Kit*, three books covering the most fundamental needs of a team: version control, unit testing, and automated build and test. These were the first three books we'd write and publish as the Pragmatic Bookshelf.

These topics are still fundamental and critical to any team's success, but a lot has changed over the last dozen years or so. Version-control technology has moved from centralized CVS and Subversion to a distributed model in Git. Automated build and related tools have become more scripted and more sophisticated, and testing has evolved from a hard-sell afterthought to a widely embraced approach via test-driven development.

Now Jeff Langr has taken on the task of updating and expanding our original unit-testing treatise for the modern world. The principles are the same, but the tools have gotten better, and I'd like to think the whole approach to software development has become more realistic, more professional, and—dare I say it?—more pragmatic. Jeff will show you the way.

Testing was always a poor name for this particular programming activity. The very name makes it sound like it's something separate from coding, separate from design, and separate from debugging.

It's not.

Your programming-language compiler/interpreter verifies that your source code is syntactically valid: that it makes at least some sort of sense according to the syntax of the language. But the compiler can't really tell what your code *does* and so can't help to determine if the code is correct or not.

Unit testing lets you specify what the code does and verifies that the code does it. Unit testing has become a marvelous intersection of design, coding, and debugging.

If you haven't gotten huge value from your testing yet, then this book will help you. Whether you're brand-new to the ideas here, or just trying to get the most benefit from unit testing, this book will help you.

Enjoy!

Andy Hunt

Publisher, The Pragmatic Bookshelf

Raleigh, NC