Extracted from:

Pragmatic Unit Testing in Java 8 with JUnit

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Pragmatic Unit Testing in Java 8 with JUnit

Jeff Langr

with Andy Hunt
& Dave Thomas

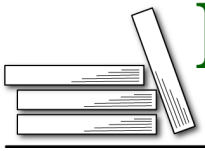
edited by
Susannah Davidson Pfalzer

Pragmatic Unit Testing in Java 8 with JUnit

Jeff Langr

with Andy Hunt
Dave Thomas

The Pragmatic Bookshelf
Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *https://pragprog.com*.

The team that produced this book includes:

Susannah Davidson Pfalzer (editor)
Potomac Indexing, LLC (indexer)
Eileen Cohen (copyeditor)
Dave Thomas (typesetter)
Janet Furlow (producer)
Ellie Callahan (support)

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-94122-259-1
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—March 2015

Preface

The notion of programmers writing tests to verify their own code was a shock to many in 2003, despite the fact that JUnit had been around for five years at the time. The first edition of *Pragmatic Unit Testing in Java with JUnit* appeared in 2003, providing a friendly overview to this new brave world of programmer unit testing.

Over a decade later, unit testing is a skill expected of most developers, even more so in Java shops. Don't be surprised if you interview at a company and they ask how you test your code. They might also ask whether or not you test-drive your code, use things known as mock objects, or have any thoughts about how to deal with legacy dependency challenges.

Getting a job is one benefit of learning about unit testing. A better benefit is that you'll improve the quality of the software you ship. Approach unit testing with an open mind, and you might even decide to change the way you build code.

Why Unit Testing

Unit testing is when you (a programmer) write test code to verify *units* of code. The size of a unit isn't precisely defined, so we'll view a unit as a small bit of code that exhibits some useful behavior in your system. A unit on its own usually doesn't represent complete end-to-end behavior. It instead represents some small subset of that end-to-end-behavior.

We're coding in Java, so we write our unit tests in Java, too. We run these unit tests through JUnit, a tool that marks our tests as passing or failing.

Here are a few whens and whys for writing unit tests:

- You just finished coding a feature and want to ensure that it works as you expect.
- You want to document a change so that you and others later understand the choices you coded into the system.

- You need to change code and want to make sure your forthcoming changes don't break any existing behavior.
- You want to understand the current behavior of the system.
- You want to know when third-party code no longer behaves as you expect.

Most important, good unit tests increase your confidence to ship your production system. You still need *integration* and/or *acceptance* tests, which verify end-to-end behavior. We focus only on unit tests in this book.

After reading this book, you'll be off and writing lots of unit tests in no time. Take care: it's easy to create lots of costly-to-maintain tests that provide little value. This book teaches you better practices for unit testing so that your investment in it keeps paying off.

Who This Book Is For

This book is a fast-paced introductory book for Java programmers new to unit testing. Although it doesn't cover every last detail about unit testing, you'll learn everything you need to dive into testing your production systems.

You should already be familiar with Java programming and comfortable with getting around in your IDE of choice.

What You Need

To follow along and code the examples shown in this book, you'll need the following three pieces of software:

- Java,[1] of course. Most any version will work, though the examples in this book use Java 8.
- An IDE. The examples in this book were built using Eclipse,[2] but you can use IntelliJ IDEA,[3] NetBeans,[4] vi, Emacs, or pretty much any editor.
- JUnit.[5] JUnit is integrated with the major three IDEs (Eclipse, IntelliJ, and NetBeans), so you won't need to install it if you go the IDE route. The examples in this book use JUnit 4.11. If you're using an older version of Java, JUnit 4.x should work for any version of Java from 1.5 on. (For even older versions of Java, you'll need to use JUnit 3.8, which sports a different interface than presented here.)

1. https://java.com/download
2. http://eclipse.org/downloads/
3. http://www.jetbrains.com/idea/download/
4. https://netbeans.org/downloads/
5. https://github.com/junit-team/junit/wiki/Download-and-Install

If your team uses TestNG, another unit-testing tool, the vast majority of this book still directly applies to your world. TestNG is close to being a proper superset of JUnit, and you'll find it trivial to translate JUnit tests to TestNG. The meatier part of the book is the set of good practices you'll learn, not the tool specifics themselves.

Refer to the individual product sites for details on how to download, install, and configure the development tools.

How to Use This Book

This book is divided into four main sections:

- Unit-Testing Foundations provides you with a starter set of information about writing basic tests in JUnit. You'll learn how to incorporate JUnit into your project, you'll write a sample test, you'll write a couple of more-realistic tests, you'll learn about JUnit organization and assertions, and you'll pick up a few core quality practices for unit testing.
- Mastering Manic Mnemonics! presents a trilogy of acronyms for improving the quality of your unit testing: the FIRST properties of good tests, the Right-BICEP for determining what to test, and the CORRECT way of exploring boundary conditions.
- The Bigger Design Picture focuses on the relevance of design to unit testing and vice versa. You'll refactor in the small, in the large, and in your tests; and you'll learn how to use mock objects to deal with troublesome dependencies.
- The Bigger Unit-Testing Picture discusses a handful of larger concerns in unit testing. You'll learn about the disciplined unit-testing practice of test-driven development. You'll be presented with some examples of testing more-interesting code challenges. And you'll find some suggestions for introducing unit testing in a team environment.

If you're brand-new or reasonably new to unit testing, we recommend that you work through the book front-to-back.

If you're more experienced with unit testing, you might be able to skip all of Unit-Testing Foundations. However, Chapter 2, *Getting Real with JUnit*, on page ? introduces the iloveyouboss application that's used throughout the book, so you might want to skim that chapter to get a bit of familiarity with the small codebase.

Otherwise, feel free to pick up any chapter that strikes your interest and move around from there. You'll find numerous links to take you elsewhere when we reference an interesting topic.

Code and Online Resources

You'll find gobs of Java code throughout the book, almost all of which is included in the source distribution. You can download the source from the official Pragmatic Unit Testing book page.[6]

Code snippets that can be found as part of the distribution appear with the path and filename immediately above the chunk of code. For example:

iloveyouboss/3/test/iloveyouboss/ScoreCollectionTest.java

```
public class ScoreCollectionTest {
   @Test
   public void test() {
   }
}
```

You'll find that snippet of code in the ScoreCollectionTest.java file in the source distribution, in the iloveyouboss/3/test/iloveyouboss directory. If you're reading this as an ebook, you can click the filename header to go directly to the code.

The code snippets you see in the book aren't manually copied from the source base; they're extracted from the source automatically. That means that the source should be in sync with what you see here. However, due to IDE configuration settings, you might see some minor differences between your code and the source code in the book. Most notably, this book uses the wildcard form for import statements (for example, import java.util.*), whereas you might have configured your IDE to show explicit import statements, one per class (for example, import java.util.List).

To reduce a bit of code clutter, we've omitted package statements from code listings.

You'll find a number of additional resources for the book at its official Pragmatic Bookshelf page.[7]

Your best route to success is to work along with the code examples yourself rather than simply read them.

Acknowledgments

We'd like to thank all the reviewers who helped with this version of the book, and thank again all those involved with the production of the first edition.

6. https://pragprog.com/titles/utj2/source_code
7. https://pragprog.com/book/utj2/pragmatic-unit-testing-in-java-8-with-junit

Many thanks to Susannah Pfalzer, who again provided excellent feedback and guidance. Thanks to Andy Hunt and Dave Thomas for blazing the trail with the first edition, and also for their wisdom in shaping this edition.

A sincere thank you to Mario Aquino, Rusty Bentley, Terry Birch, Kelly Brant, John Cater, Brad Collins, Jeremy D. Frens, Derek Graham, Alexander Henry, Rod Hilton, Eric Jutrzenka, Andy Keffalas, Richard Langlois, Mark Latham, Harold Meder, Fahmida Y. Rashid, Sam Rose, Ray Santos, Bas Stoker, Charley Stran, and Colin Yates, for your valued feedback.

Thanks in advance to those of you who provide feedback after initial publication—and we'll try to make sure you see your name here. Books today are living, breathing documents.

If you bought the first edition of this book over ten years ago, thank you and we still love you! Hopefully you've been "test infected"[8] ever since, and if so, you probably don't need another introductory book, but welcome back anyway. You'll probably find a few new nuggets that pay for the low, low price of admission.

Jeff Langr
jeff@langrsoft.com
January 2015

8. See http://junit.sourceforge.net/doc/testinfected/testing.htm.