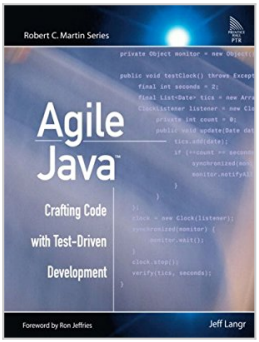

Agile Java™: Crafting Code with Test-Driven Development

By Jeff Langr

Agile Java™: Crafting Code with Test-Driven Development By Jeff Langr

Shows how Java and TDD integrate throughout the entire development lifecycle, helping you leverage efficient development techniques. This work also shows how to translate oral requirements into practical tests, and then how to use those tests to create reliable, high-performance Java code that solves real problems.

Agile Java™: Crafting Code with Test-Driven Development

By Jeff Langr

Agile Java™: Crafting Code with Test-Driven Development By Jeff Langr

Shows how Java and TDD integrate throughout the entire development lifecycle, helping you leverage efficient development techniques. This work also shows how to translate oral requirements into practical tests, and then how to use those tests to create reliable, high-performance Java code that solves real problems.

Agile Java™: Crafting Code with Test-Driven Development By Jeff Langr Bibliography

- Sales Rank: #1267391 in Books
- Published on: 2005-02-24
- Original language: English
- Number of items: 1
- Dimensions: 9.10" h x 1.60" w x 7.10" l, 2.75 pounds
- Binding: Paperback
- 792 pages

Introduction

I am a software craftsman.[1] I have spent much of my software development career trying to quickly build solutions to problems. At the same time I have tried to ensure that my solutions demonstrate carefully crafted code. I strive for perfection in code, yet I know that it is unattainable, particularly with the constant pressure from business to get product out the door. I take modest pride in the code I build daily, yet when I look at the code I wrote the day before, I wonder, "What the heck was I thinking?" This is what keeps the craft challenging to me—the constant desire to figure out how to do things a little better the next time, and with a little less pain than this time.

Agile Java represents a successful approach to learning and mastering Java development. It is based on the way I have learned to best teach programming and to learn new programming languages myself: Using test-driven development (TDD), a technique that introduces a large amount of low-level feedback. This feedback allows you to more quickly see the results of your actions. Using TDD, you will learn how to craft Java code to produce solid object-oriented designs and highly maintainable high-quality systems.

I have used TDD in production systems for over four years and am still amazed at what it does for me. It has improved the quality of my code, it has taught me new things each week, it has made me more productive. I have also created and taught language courses using TDD, both at my own company and at Object Mentor (which continues to teach their language courses with this approach).

Prior to learning TDD, I spent more than fifteen years learning, developing in, and teaching languages the "classic" way—without using tests to drive the development. The student builds and executes example code. The student obtains feedback on what the code is teaching by viewing the output from code execution. While this is a perfectly valid approach, my anecdotal experience is that using it results in less-than-ideal retention of language details.

In contrast, the high volume of rapid feedback in TDD constantly reinforces correct coding and quickly points out incorrect coding. The classic code-run-and-observe approach provides feedback, but at a much slower rate. Unfortunately, it is currently the predominant method of teaching programming.

Others have attempted more innovative approaches to teaching. In the 1990s, Adele Goldberg created a product known as LearningWorks designed for teaching younger students. It allowed a user to directly manipulate visual objects by dynamically executing bits of code. The user saw immediate results from their actions. A recent Java training tool uses a similar approach. It allows the student to execute bits of code to produce visual effects on "live" objects.

The problem with approaches like these is that they are bound to the learning environment. Once you complete the training, you must still learn how to construct your own system from the ground up, without the use of these constrained tools. By using TDD as the driver for learning, you are taught an unbounded technique that you can continue to use in your professional software development career.

Agile Java takes the most object-oriented approach feasible. Part of the difficulty in learning Java is the "bootstrapping" involved. What is the minimum you must learn in order to be able to write classes of some substance?

Most books start by teaching you the prototypical first Java program—the "hello world" application. But it is quite a mouthful: class Hello { public static void main(String args) { System.out.println("hello world"); } }. This brief program contains at least a dozen concepts that you must ultimately learn. Worse, out of those dozen concepts, at least three are nonobject-oriented concepts that you are better off learning *much* later.

In this book, you will learn the right way to code from the start and come back to fully understand "hello world" later in the book.[2] Using TDD, you will be able to write good object-oriented code immediately. You'll still have a big initial hurdle to get over, but this approach keeps you from having to first understand not-very-object-oriented concepts such as static methods and arrays. You will learn all core Java concepts in due time, but your initial emphasis is on objects.

Agile Java presents a cleaner break from the old way of doing things. It allows you to pretend for a while that there was never a language called C, the syntactical basis for Java that has been around for 30 years. While C is a great language, its imprint on Java left a quite a few constructs that can distract you from building good object-oriented systems. Using Agile Java, you can learn the right way of doing things before having to understand these legacies of the Java language.

Who Is This Book For?

I designed Agile Java for new programmers who want to learn Java as their first language. The book can also be effective for programmers familiar with TDD but new to Java, or vice versa. Experienced Java developers may find that going through Agile Java presents them with a new, and I hope better, way of approaching things.

This edition of Agile Java covers Java 2 Standard Edition (J2SE) version 5.0.

Sun has made available dozens of class libraries, or APIs (application programming interfaces), that enhance the core Java language. Some examples: JMS (Java Messaging Service) provides a definition for standard messaging-based solutions. EJBs (Enterprise Java Beans) provide a means of building component-based software for large enterprise solutions. JDBC (Java DataBase Connectivity) supplies a standard interface for interacting with relational databases. About a dozen of the advanced APIs are collectively known as J2EE (Java 2 Enterprise Edition). Many of the APIs require entire books for comprehensive coverage. There are dozens of books on J2EE.

This book covers only a few of the additional APIs at an introductory level. Technologies that are used pervasively in the majority of enterprise applications, such as logging, JDBC, and Swing, are presented in Agile Java. Some of the information (for example, logging) will teach you all you need to know for most applications. Other lessons (for example, Swing and JDBC) will teach you a basic understanding of the technology. These lessons will provide you with enough to get started and will tell you where to go for more information.

If you are developing mobile applications, you will be using J2ME (Java 2 Micro Edition). J2ME is a version of Java geared toward environments with limited resources, such as cell phones. J2ME has some significant limitations compared to J2SE. This book does not discuss anything specific with respect to J2ME. However, most of the core techniques and concepts of Java development are applicable to the J2ME environment.

In order to use any of these add-on Java technologies, you must first understand the core language and libraries provided in J2SE. Agile Java will help you build that knowledge.

What This Book Is Not

Agile Java is not an exhaustive treatise on every aspect of the Java language. It instead provides an agile approach to learning the language. Rather than giving you all the fish, I teach you how to fish and sometimes where to find the fish. Agile Java will teach you the majority of the core language concepts. Indeed, upon completing the core fifteen lessons, you will be able to produce quality production Java code. However, there

are bound to be a few esoteric language features and nuances that I do not cover in the book.

One way to become familiar with the dusty corners of the language is to peruse the Java Language Specification (JLS). The second edition of the language specification is available at http://java.sun.com/docs/books/jls. This edition covers versions of Java up to but not including J2SE 5.0. A third edition of the JLS is in the works at the time I write this. You can find a maintenance review version at http://java.sun.com/docs/books/jls/java_language-3_0-mr-spec.zip .

For additional understanding of what is in the Java API library, the Java API documentation and actual source code is the best place to go.

Agile Java is not a certification study guide. It will not prepare you for a certification exam. A good book on certification will teach you how to take a test. It will also teach you how to decipher (deliberately) poorly written code that shouldn't have been written that way in the first place. Agile Java will teach you instead how to write professional Java code.

Agile Java doesn't attempt to coddle you. Learning to program is a significant challenge. Programming involves thinking and solving problems—it is not an easy undertaking for idiots or dummies. I've tried to avoid insulting your intelligence in this book. That being said, I have tried to make Agile Java an enjoyable, easy read. It's a conversation between you and me, much like the conversations you will continue to have in your professional development career. It's also a conversation between you and your computer.

TDD Disclaimer

Some developers experienced in TDD will note stylistic differences between their approach and my approach(es) in Agile Java. There are many ways to do TDD. None of these techniques are perfect or ordained as the absolute right way to do things. Do whatever works best for you. Do what makes the most sense, as long as it doesn't violate the basic tenets set forth in Agile Java.

Readers will also find areas in which the code could be improved.[3] Even as a beginning developer, you no doubt will encounter code in Agile Java that you don't like. Let me know. Send in your suggestions, and I may incorporate them in the next edition. And fix your own implementations. *You can improve almost any code or technique*. Do!

After the first lesson in Agile Java, the tests appear wholesale, as if they were coded in one fell swoop. This is not the case: Each test was built assertion by assertion, in much smaller increments than the book can afford to present. Keep this in mind when writing your own code—one of the most important aspects of TDD is taking small, small steps with constant feedback. And when I say small, I mean small! If you think you're taking small steps, try taking even smaller steps.

How to Use This Book

The core of Agile Java is fifteen lessons of about 30 pages each. You will start with baby steps in Java, TDD, and OO. You will finish with a strong foundation for professional Java development.

The core lessons are sequential. You should start at Lesson 1 and complete each lesson before proceeding to the next. Once you have completed the core lessons, you should have a solid understanding of how to build robust Java code.

If you haven't completed the fifteen core lessons, you should not assume you know how to write good Java code! (Even if you have completed the lessons, you're not an expert . . . yet.) Each lesson builds upon the

previous lessons. If you stop short of completing the lessons, your lack of full understanding may lead you to construct poor code.

Each lesson starts with a brief overview of the topics to be discussed. The remainder of the lesson is a narrative. I describe each language feature in the text, specified by test code. I demonstrate each feature in a corresponding code implementation. I have interspersed discussions of TDD technique, OO principles, and good development practices in these lessons.

I've supplied three additional lessons to cover a few more Java topics. Two of the lessons present an introduction to Java's tool for user interface development, Swing. These two lessons will provide you with enough information to begin building robust user interface applications in Java. But the bigger intent is to give you some ideas for how to build them using TDD. The third additional lesson presents an overview for a number of Java topics, things that most Java developers will want to know.

For the most effective learning experience, you should follow along with the lessons by entering and executing each bit of test and implementation code as I present it. While the code is available for downloading (see the next paragraph), I highly recommend that you type each bit of code yourself. Part of doing TDD correctly is getting a good understanding of the rhythm involved in going back and forth between the tests and the code. If you just download the code and execute it, you're not going to learn nearly as much. The tactile response of the keyboard seems to impart a lot of learning.

However, who am I to make you work in a certain way? You may choose to download the code from http://www.LangrSoft.com/agileJava/code. I've organized this code by lesson. I have made available the code that is the result of working each lesson—*in the state it exists by the end of the lesson*. This will make it easier for you to pick up at any point, particularly since many of the examples carry through to subsequent lessons.

Exercises

Each of the fifteen core lessons in Agile Java has you build bits and pieces of a student information system for a university. I chose this single common theme to help demonstrate how you can incrementally build upon and extend existing code. Each lesson also finishes with a series of exercises. Instead of the student information system, the bulk of the exercises, provided by Jeff Bay, have you build bits and pieces of a chess application.

Some of the exercises are involved and quite challenging. But I highly recommend that you do every one. The exercises are where the real learning starts—you're figuring out how to solve problems using Java, without my help. Doing all of the exercises will give you a second opportunity to let each lesson sink in.

Conventions Used in This Book

Code flows with the text, in order to make it part of the conversation. If I refer to code "below," it's the next piece of code as you continue to read; code "above" appears in recent prior text.

Code (below) appears in a non-proportional font:

```
this.isCode();
```

Smaller portions of a large block of code may appear in **bold**. The **bold** indicates either new code that relates to the current example or code that has particular relevance:

```
class B { public void thisIsANewMethod() {</b>  }</b>}
```

Code appearing directly in text, such as this.someCode(), also appears in a non-proportional font. However, the names of classes, such as CourseSession, appear in the same font as the rest of the text.

I often use ellipses in the code samples. The ellipses indicate code that already exists in a class but is not relevant to the current discussion or example:

```
class C { private String interestingVariable;</b> ... private void
someInterestingMethod() {</b>  }</b> ...
```

The dialog in Agile Java alternates between expressing ideas in English and expressing them in code. For example, I may refer to the need to cancel a payroll check or I might choose to say that you should send the cancel message to the PayrollCheck object. The idea is to help you start making the necessary connections between understanding requirements and expressing them in code.

Class names are by convention singular nouns, such as Customer. "You use the Customer class to create multiple Customer objects." In the name of readability, I will sometimes refer to these Customer objects using the plural of the class name: "The collection contains all of the Customers loaded from the file system."

New terms initially appear in *italics*. Most of these terms appear in the Glossary (Appendix A).

Throughout Agile Java, you will take specifications and translate them into Java code. In the tradition of agile processes, I present these specifications using informal English. They are requirements, also known as *stories*. A story is a promise for more conversation. Often you will need to continue a dialog with the presenter of the story (sometimes known as the customer) in order to obtain further details on a story. In the case of Agile Java, if a story isn't making sense, try reading a bit further. Read the corresponding tests to see how they interpret the story.

I have highlighted stories throughout Agile Java with a "storytelling" icon. The icon emphasizes the oral informality of stories.

Your best path to learning the craft of programming is to work with an experienced practitioner. You will discover that there are many "secrets" to programming. Some secrets are basic concepts that you must know in order to master the craft. Other secrets represent pitfalls—things to watch out for. You must bridge these challenges, and remember what you learned, to be a successful Java programmer.

I've marked such critical points with a bridge icon (the bridge is crossing over the pitfalls). Many of these critical points will apply to development in any language, not just Java.

FOOTNOTES

1. See McBreen2000.
2. Don't fret, though: You'll actually start with the "hello world" application in order to ensure you can compile and execute Java code. But you won't have to understand it all until later.
3. A constant developer pair would have helped.

Read Agile Java™: Crafting Code with Test-Driven Development By Jeff Langr for online ebook

Agile Java™: Crafting Code with Test-Driven Development By Jeff Langr Free PDF d0wnl0ad, audio books, books to read, good books to read, cheap books, good books, online books, books online, book reviews epub, read books online, books to read online, online library, greatbooks to read, PDF best books to read, top books to read Agile Java™: Crafting Code with Test-Driven Development By Jeff Langr books to read online.

Online Agile Java™: Crafting Code with Test-Driven Development By Jeff Langr ebook PDF download

Agile Java™: Crafting Code with Test-Driven Development By Jeff Langr Doc

Agile Java™: Crafting Code with Test-Driven Development By Jeff Langr Mobipocket

Agile Java™: Crafting Code with Test-Driven Development By Jeff Langr EPub