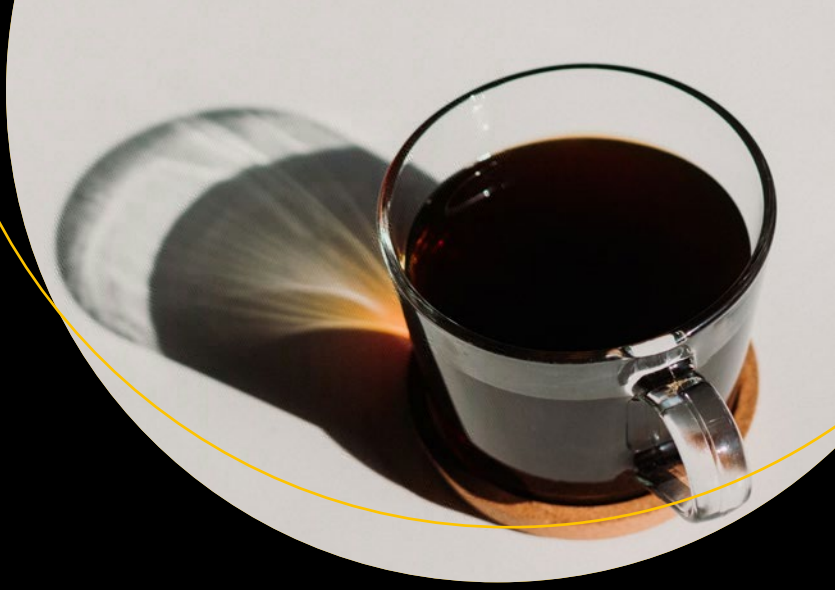

More Java 17

An In-Depth Exploration of the Java Language and Its Features

—

Third Edition

—

Kishori Sharan
Peter Späth

Apress®

More Java 17

An In-Depth Exploration of the Java Language and Its Features

Third Edition

Kishori Sharan

Peter Späth

Apress®

More Java 17: An In-Depth Exploration of the Java Language and Its Features

Kishori Sharan
Montgomery, AL, USA

Peter Späth
Leipzig, Sachsen, Germany

ISBN-13 (pbk): 978-1-4842-7134-6
https://doi.org/10.1007/978-1-4842-7135-3

ISBN-13 (electronic): 978-1-4842-7135-3

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Steve Anglin
Development Editor: Matthew Moodie
Coordinating Editor: Mark Powers

Cover designed by eStudioCalamar

Cover image by Ben Kolde on Unsplash (www.unsplash.com)

Distributed to the book trade worldwide by Apress Media, LLC, 1 New York Plaza, New York, NY 10004, U.S.A. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484271346. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

To Paulina

Table of Contents

About the Authors

Kishori Sharan works as a senior software engineer lead at IndraSoft, Inc. He earned a master of science degree in computer information systems from Troy University, Alabama. He is a Sun-certified Java 2 programmer and has over 20 years of experience in developing enterprise applications and providing training to professional developers using the Java platform.

Peter Späth graduated in 2002 as a physicist and soon afterward became an IT consultant, mainly for Java-related projects. In 2016, he decided to concentrate on writing books on various aspects, but with the main focus set on software development. With two books about graphics and sound processing, three books for Android app development, and several books about Java and Jakarta EE development, the author continues his effort in writing software development–related literature.

About the Technical Reviewers

Massimo Nardone has more than 25 years of experience in security, web/mobile development, cloud, and IT architecture. His true IT passions are security and Android. He has been programming and teaching how to program with Android, Perl, PHP, Java, VB, Python, C/C++, and MySQL for more than 20 years. He holds a master of science degree in computing science from the University of Salerno, Italy.

He has worked as a CISO, CSO, security executive, IoT executive, project manager, software engineer, research engineer, chief security architect, PCI/SCADA auditor, and senior lead IT security/cloud/SCADA architect for many years. His technical skills include security, Android, cloud, Java, MySQL, Drupal, Cobol, Perl, web and mobile development, MongoDB, D3, Joomla, Couchbase, C/C++, WebGL, Python, Pro Rails, Django CMS, Jekyll, Scratch, and more.

He worked as a visiting lecturer and supervisor for exercises at the Networking Laboratory of the Helsinki University of Technology (Aalto University). He holds four international patents (PKI, SIP, SAML, and Proxy areas). He is currently working for Cognizant as head of cyber security and CISO to help both internally and externally with clients in areas of information and cyber security, like strategy, planning, processes, policies, procedures, governance, awareness, and so forth. In June 2017, he became a permanent member of the ISACA Finland Board.

Massimo has reviewed more than 45 IT books for different publishing companies and is the co-author of *Pro Spring Security: Securing Spring Framework 5 and Boot 2-based Java Applications* (Apress, 2019), *Beginning EJB in Java EE 8* (Apress, 2018), *Pro JPA 2 in Java EE 8* (Apress, 2018), and *Pro Android Games* (Apress, 2015).

Satej Kumar Sahu works in the role of Senior Enterprise Architect at Honeywell. He is passionate about technology, people, and nature. He believes through technology and conscientious decision making, each of us has the power to make this world a better place. In his free time, he can be found reading books, playing basketball, and having fun with friends and family.

Introduction

How This Book Came About

My first encounter with the Java programming language was during a one-week Java training session in 1997. I did not get a chance to use Java in a project until 1999. I read two Java books and took a Java 2 programmer certification examination. I did very well on the test, scoring 95%. The three questions that I missed on the test made me realize that the books that I had read did not adequately cover details of all the topics. I made up my mind to write a book on the Java programming language. So I formulated a plan to cover most of the topics that a Java developer needs to use Java effectively in a project, as well as to become certified. I initially planned to cover all essential topics in Java in 700–800 pages.

As I progressed, I realized that a book covering most of the Java topics in detail could not be written in 700–800 pages. One chapter alone that covered data types, operators, and statements spanned 90 pages. I was then faced with the question, "Should I shorten the content of the book or include all the details that I think a Java developer needs?" I opted for including all the details in the book, rather than shortening its content to maintain the original number of pages. It has never been my intent to make lots of money from this book. I was never in a hurry to finish this book because that rush could have compromised the quality and coverage. In short, I wrote this book to help the Java community understand and use the Java programming language effectively, without having to read many books on the same subject. I wrote this book with the plan that it would be a comprehensive one-stop reference for everyone who wants to learn and grasp the intricacies of the Java programming language.

One of my high-school teachers used to tell us that if one wanted to understand a building, one must first understand the bricks, steel, and mortar that make up the building. The same logic applies to most of the things that we want to understand in our lives. It certainly applies to an understanding of the Java programming language. If you want to master the Java programming language, you must start by understanding its basic building blocks. I have used this approach throughout this book, endeavoring to build upon each topic by describing the basics first. In the book, you will rarely find a

topic described without first learning about its background. Wherever possible, I tried to correlate the programming practices with activities in daily life. Most of the books about the Java programming language available on the market either do not include any pictures at all or have only a few. I believe in the adage "A picture is worth a thousand words." To a reader, a picture makes a topic easier to understand and remember. I have included plenty of illustrations in the book to aid readers in understanding and visualizing the concepts. Developers who have little or no programming experience have difficulty in putting things together to make it a complete program. Keeping them in mind, I have included over 390 complete Java programs that are ready to be compiled and run.

I spent countless hours doing research when writing this book. My main sources were the Java Language Specification, whitepapers and articles on Java topics, and Java Specification Requests (JSRs). I also spent quite a bit of time reading the Java source code to learn more about some of the Java topics. Sometimes, it took a few months of researching a topic before I could write the first sentence on it. Finally, it was always fun to play with Java programs, sometimes for hours, to add them to the book.

Introduction to the Second Edition

I am pleased to present the second edition of the *Java Language Features* book. It is the second book in the three-volume "Beginning Java 9" series. It was not possible to include all JDK9 changes in one volume. I have included JDK9-specific changes at appropriate places in the three volumes, including this one. If you are interested in learning only JDK9-specific topics, I suggest you read my *Java 9 Revealed* book (ISBN 9781484225912). There are several changes in this edition, as follows:

- I added the following five chapters to this edition: Implementing Services, The Module API, Breaking Module Encapsulation, Reactive Streams, and Stack Walking.
- Implementing services in Java is not new to JDK9. I felt this book was missing a chapter on this topic. A chapter covers in detail how to define services and service interfaces and how to implement service interfaces using JDK9-specific and pre-JDK9 constructs. This chapter shows you how to use them and provides statements in a module declaration.

- Another chapter covers the Module API in detail, which gives you programmatic access to modules. This chapter also touches on some of the advanced topics, such as module layers. The first volume of this series covered basics on modules, such as how to declare modules and module dependence.
- The following chapter covers how to break module encapsulation using command-line options. When you migrate to JDK9, there will be cases requiring you to read the module's internal APIs or export non-exported packages. You can achieve these tasks using command-line options covered in this chapter.
- Reactive Streams is an initiative for providing a standard for asynchronous stream processing with non-blocking backpressure. It is aimed at solving the problems processing a stream of items, including how to pass a stream of items from a publisher to a subscriber without requiring the publisher to block or the subscriber to have an unbounded buffer. One more chapter covers the Reactive Streams API, which was added in JDK9.
- A new chapter covers the Stack-Walking API, which was added in JDK9. This API lets you inspect the stack frames of threads and get the class reference of the caller class of a method. Inspecting a thread's stack and getting the caller's class name were possible before JDK9. The new Stack-Walking API lets you achieve this easily and efficiently.
- I received several emails from the readers about the fact that the books in this series do not include questions and exercises, which are needed mainly for students and beginners. Students use this series in their Java classes, and many beginners use it to learn Java. Due to this popular demand, I spent over 60 hours preparing questions and exercises at the end of each chapter. My friend Preethi offered her help and provided the solutions.

Apart from these additions, I updated all the chapters that were part of the first edition. I edited the contents to make them flow better, changed or added new examples, and updated the contents to include JDK9-specific features.

It is my sincere hope that this edition will help you learn Java better.

Introduction to the Third Edition

The third edition is the second author Peter Späth's work. Pleasantly taking over much of Kishori Sharan's efforts, the original text was substantially shortened by omitting a couple of chapters, and instead adding API-related topics from the book *Java APIs, Extensions and Libraries*, again from Kishori Sharan. In addition, all topics covered were hovered to Java 17, in order to maximize the benefit for the reader facing contemporary Java projects and wishing to use the new features included with the JRE 17.

Caution Oracle changed the licensing with JDK8. You must enter a paid program if you plan to use Oracle's JRE or JDK for commercial projects. If you want to avoid this, consider using OpenJDK.

Structure of the Book

This book contains 14 chapters. The first seven chapters contain language-level topics of Java such as annotations, reflection, generics, lambda expressions, streams, etc. The chapters introduce Java topics in increasing order of complexity. The subsequent six chapters introduce some of the more important Java APIs and modules, like network programming, remote method invocation, scripting, and more. The last chapter, "Miscellanea," gives the rationale for chapters omitted in this edition compared to the previous one.

In the appendix, solution hints to the exercises are provided.

Audience

This book is designed to be useful to anyone who wants to learn the Java programming language. If you are a beginner, with little or no programming background in Java, you are advised to read one of the beginning-level Java books from Apress, and also the online Java documentation including the Java tutorial will help. This book contains topics of various degrees of complexity. As a beginner, if you find yourself overwhelmed while reading a section in a chapter, you can skip to the next section or the next chapter and revisit it later when you gain more experience.

If you are a Java developer with an intermediate or advanced level of experience, you can jump to a chapter or to a section in a chapter directly. If a section covers an unfamiliar topic, you need to visit that topic before continuing the current one.

If you are reading this book to get a certification in the Java programming language, you need to read almost all of the chapters, paying attention to all of the detailed descriptions and rules. Most of the certification programs test your fundamental knowledge of the language, not the advanced knowledge. You need to read only those topics that are part of your certification test. Compiling and running the Java programs included with the book will help you prepare for your certification.

If you are a student who is attending a class on the Java programming language, you should read the chapters of this book selectively. Some topics, such as lambda expressions, collections, and streams, are used extensively in developing Java applications, whereas other topics are infrequently used. You need to read only those chapters that are covered in your class syllabus. I am sure that you, as a Java student, do not need to read the entire book page by page.

How to Use This Book

This book is the beginning, not the end, of learning the Java programming language. If you are reading this book, it means you are heading in the right direction to learn the Java programming language, which will enable you to excel in your academic and professional career. However, there is always a higher goal for you to achieve, and you must constantly work hard to achieve it. The following quotations from some great thinkers may help you understand the importance of working hard and constantly looking for knowledge with both your eyes and mind open.

> *The learning and knowledge that we have, is, at the most, but little compared with that of which we are ignorant.*
>
> —Plato

> *True knowledge exists in knowing that you know nothing. And in knowing that you know nothing, that makes you the smartest of all.*
>
> —Socrates

Readers are advised to use the API documentation for the Java programming language as much as possible while reading this book. The Java API documentation includes a complete list of everything available in the Java class library. You can download (or view) the Java API documentation from the official website of Oracle Corporation at www.oracle.com.

While you read this book, you need to practice writing Java programs. You can also practice by tweaking the programs provided in the book. It does not help much in your learning process if you just read this book and do not practice writing your own programs. Remember that "practice makes perfect," which is also true in learning how to program in Java.

Source Code and Errata

Source code for this book can be accessed by clicking the **Download Source Code** button located at www.apress.com/9781484271346.

CHAPTER 1

Annotations

In this chapter, you will learn:

- What annotations are
- How to declare annotations
- How to use annotations
- What meta-annotations are and how to use them
- Commonly used annotations that are used to deprecate APIs, to suppress named compile-time warnings, override methods, and declare functional interfaces
- How to access annotations at runtime
- How to process annotations in source code

All example programs in this chapter are a member of a `jdojo.annotation` module, as declared in Listing 1-1.

Listing 1-1. The Declaration of a jdojo.annotation Module

```
// module-info.java
module jdojo.annotation {
    exports com.jdojo.annotation;
}
```

What Are Annotations?

Before I define annotations and discuss their importance in programming, let's look at a simple example. Suppose you have an `Employee` class, which has a method called

K. Sharan and P. Späth, *More Java 17*, https://doi.org/10.1007/978-1-4842-7135-3_1

setSalary() that sets the salary of an employee. The method accepts a parameter of the type double. The following snippet of code shows a trivial implementation for the Employee class:

```
public class Employee {
    public void setSalary(double salary) {
        System.out.println("Employee.setSalary():" +
            salary);
    }
}
```

A Manager class inherits from the Employee class. You want to set the salary for managers differently. You decide to override the setSalary() method in the Manager class. The code for the Manager class is as follows:

```
public class Manager extends Employee {
    // Override setSalary() in the Employee class
    public void setSalary(int salary) {
        System.out.println("Manager.setSalary():" +
            salary);
    }
}
```

There is a mistake in the Manager class, when you attempt to override the setSalary() method. You'll correct the mistake shortly. You have used the int data type as the parameter type for the incorrectly overridden method. It is time to set the salary for a manager. The following code is used to accomplish this:

```
Employee ken = new Manager();
int salary = 200;
ken.setSalary(salary);
Employee.setSalary():200.0
```

This snippet of code was expected to call the setSalary() method of the Manager class, but the output does not show the expected result.

What went wrong in your code? The intention of defining the setSalary() method in the Manager class was to override the setSalary() method of the Employee class, not to overload it. You made a mistake. You used the type int as the parameter type in the

`setSalary()` method, instead of the type `double` in the `Manager` class. You put comments indicating your intention to override the method in the `Manager` class. However, comments do not stop you from making logical mistakes. You might spend, as every programmer does, hours and hours debugging errors resulting from this kind of logical mistake. Who can help you in such situations? Annotations might help you in a few situations like this.

Let's rewrite your `Manager` class using an annotation. You do not need to know anything about annotations at this point. All you are going to do is add one word to your program. The following code is the modified version of the `Manager` class:

```
public class Manager extends Employee {
    @Override
    public void setSalary(int salary) {
        System.out.println("Manager.setSalary():" +
            salary);
    }
}
```

All you have added is an `@Override` annotation to the `Manager` class and removed the "dumb" comments. Trying to compile the revised `Manager` class results in a compile-time error that points to the use of the `@Override` annotation for the `setSalary()` method of the `Manager` class:

```
Manager.java:2: error: method does not override or
            implement a method from a supertype
        @Override
        ^
1 error
```

The use of the `@Override` annotation did the trick. The `@Override` annotation is used with a non-static method to indicate the programmer's intention to override the method in the superclass. At the source code level, it serves the purpose of documentation. When the compiler comes across the `@Override` annotation, it makes sure that the method really overrides the method in the superclass. If the method annotated does not override a method in the superclass, the compiler generates an error. In your case, the `setSalary(int salary)` method in the `Manager` class does not override any method in the superclass `Employee`. This is the reason that you got the error. You may realize that

using an annotation is as simple as documenting the source code. However, they have compiler support. You can use them to instruct the compiler to enforce some rules. Annotations provide benefits much more than you have seen in this example. Let's go back to the compile-time error. You can fix the error by doing one of the following two things:

- You can remove the `@Override` annotation from the `setSalary(int salary)` method in the `Manager` class. It will make the method an overloaded method, not a method that overrides its superclass method.
- You can change the method signature from `setSalary(int salary)` to `setSalary(double salary)`.

Since you want to override the `setSalary()` method in the `Manager` class, use the second option and modify the `Manager` class as follows:

```
public class Manager extends Employee {
    @Override
    public void setSalary(double salary) {
        System.out.println("Manager.setSalary():" +
            salary);
    }
}
```

Now the following code will work as expected:

```
Employee ken = new Manager();
int salary = 200;
ken.setSalary(salary);
Manager.setSalary():200.0
```

Note that the `@Override` annotation in the `setSalary()` method of the `Manager` class saves you debugging time. Suppose you change the method signature in the `Employee` class. If the changes in the `Employee` class make this method no longer overridden in the `Manager` class, you will get the same error when you compile the `Manager` class again. Are you starting to understand the power of annotations? With this background in mind, let's start digging deep into annotations.