A Chapman & Hall Book

CRC Press
Taylor & Francis Group

JAVA Programming Exercises

Volume Two: Java Standard Library

Christian Ullenboom

Java Programming Exercises

Take the next step in raising your coding skills and dive into the intricacies of Java Standard Libraries. You will continue to raise your coding skills, and test your Java knowledge on tricky programming tasks, with the help of the pirate *Captain CiaoCiao*. This is the second of two volumes which provide you with everything you need to excel in your Java journey, including tricks that you should know in detail as a professional, as well as intensive training for clean code and thoughtful design that carries even complex software.

Features:

- 149 tasks with commented solutions on different levels
- For all paradigms: object-oriented, imperative, and functional
- Clean code, reading foreign code, and object-oriented modeling

With numerous best practices and extensively commented solutions to the tasks, these books provide the perfect workout for professional software development with Java.

Java Programming Exercises

Volume Two: Java Standard Library

Christian Ullenboom

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

Designed cover image: Mai Loan Nguyen Duy, Rheinwerk Verlag GmbH

First edition published 2025
by CRC Press
2385 NW Executive Center Drive, Suite 320, Boca Raton FL 33431

and by CRC Press
4 Park Square, Milton Park, Abingdon, Oxon, OX14 4RN

CRC Press is an imprint of Taylor & Francis Group, LLC

ISBN: 978-1-032-80117-9 (hbk)
ISBN: 978-1-032-79801-1 (pbk)
ISBN: 978-1-003-49555-0 (ebk)

DOI: 10.1201/9781003495550

Typeset in Times
by codeMantra

Access the Support Material: https://routledge.com/9781032798011

Contents

About the Author

Christian Ullenboom started his programming journey at the tender age of ten, typing his first lines of code into a C64. After mastering assembler programming and early BASIC extensions, he found his calling on the island of Java, following his studies in computer science and psychology. Despite indulging in Python, JavaScript, TypeScript, and Kotlin vacations, he remains a savant of all things Java.

For over 20 years, Ullenboom has been a passionate software architect, Java trainer (check out http://www.tutego.com), and IT specialist instructor. His expertise has resulted in a number of online video courses and reference books:

- *Java: The Comprehensive Guide* (ISBN-13: 978-1493222957)
- *Java ist auch eine Insel: Java programmieren lernen mit dem umfassenden Standardwerk für Java-Entwickler* (ISBN-13: 978-3836287456)
- *Java SE 9 Standard-Bibliothek: Das Handbuch für Java-Entwickler* (ISBN-13: 978-3836258746)
- *Captain CiaoCiao erobert Java: Das Trainingsbuch für besseres Java* (ISBN-13: 978-3836284271)
- *Spring Boot 3 und Spring Framework 6: Professionelle Enterprise-Anwendungen mit Java* (ISBN-13: 978-3836290494)

Christian Ullenboom has been spreading Java love through his books for years, earning him the coveted title of Java Champion from Sun (now Oracle) way back in 2005. Only a select few—about 300 worldwide—have achieved this status, making him a true Java superstar.

As an instructor, Ullenboom understands that learning by doing is the most effective way to master a skill. So, he has compiled a comprehensive catalog of exercises that accompany his training courses. This book features a selection of those exercises, complete with documented solutions.

His roots are in Sonsbeck, a small town in the Lower Rhine region of Germany.

Introduction

Many beginners in programming often ask themselves, "How can I strengthen my skills as a developer? How can I become a better programmer?" The answer is simple: study, attend webinars, learn, repeat, practice, and discuss your work with others. Many aspects of programming are similar to learning new skills. Just as a book can't teach you how to play a musical instrument, watching the *Fast and the Furious* movie series won't teach you how to drive. The brain develops patterns and structures through repeated practice. Learning a programming language and a natural language have many similarities. Consistent use of the language, and the desire and need to express and communicate in it (just as you need to do so to order a burger or a beer), leads to gradual improvement in skills.

Books and webinars on learning a programming language are available, but reading, learning, practicing, and repeating are just one aspect of becoming a successful software developer. To create effective software solutions, you need to creatively combine your knowledge, just as a musician regularly practices finger exercises and maintains their repertoire. The more effective your exercises are, the faster you will become a master. This book aims to help you progress and gain more hands-on experience.

Java 21 declares more than 2300 classes, about 1400 interfaces, around 140 enumerations, approximately 500 exceptions, and a few annotation types and records are added to this. However, in practical terms, only a small subset of these types proves to be relevant. This book selects the most important types and methods for tasks, making them motivating, and following Java conventions. Alternative solutions and approaches are also presented repeatedly. The goal is to make non-functional requirements clear because the quality of programs is not just about "doing what it should." Issues such as correct indentation, following naming conventions, proper use of modifiers, best practices, and design patterns are essential. The proposed solutions aim to demonstrate these principles, with the keyword being *Clean Code*.

PREVIOUS KNOWLEDGE AND TARGET AUDIENCE

The book is aimed at Java developers who are either new to Java or are already advanced and wish to learn more about the Java SE standard libraries. The intended audience includes:

- Computer science students
- IT specialists
- Java programmers
- Software developers
- Job applicants

The book is centered around tasks and fully documented solutions, with detailed explanations of Java peculiarities, good object-oriented programming practices, best practices, and design patterns. The exercises are best solved with a textbook, as this exercise book is not a traditional textbook. A useful approach is to work through a topic with a preferred textbook before attempting the exercises that correspond to it.

DOI: 10.1201/9781003495550-1

The first set of tasks are designed for programming beginners who are new to Java. As you gain more experience with Java, the tasks become more challenging. Therefore, there are tasks for both beginners and advanced developers.

The Java Standard Edition is augmented by numerous frameworks and libraries. However, this exercise book does not cover specific libraries or Java Enterprise frameworks like Jakarta EE or Spring (Boot). There are separate exercise books available for these environments. Additionally, the book does not require the use of tools like profiling tools, as these are beyond the scope of the book.

WORKING WITH THE BOOK

The task book is organized into different sections. The first section covers the Java language, followed by selected areas of the Java standard library, such as data structures or file processing. Each area is accompanied by programming tasks and "quiz" questions that contain surprises. Each section starts with a small motivation and characterization of the topic, followed by the exercises. Additional tips and hints are provided for particularly challenging assignments, while other exercises offer optional extensions for further exploration.

The majority of exercises are independent of each other, making it easy for readers to dive in anywhere. However, in the chapter on imperative programming, some tasks build on each other to develop a larger program, and the same goes for the chapter on object-oriented programming. The problem definitions make this clear, and more complex programs help to provide context for understanding different language characteristics. Furthermore, a more complex program can motivate readers to continue.

The exercises are rated with one, two, or three stars to indicate their complexity, although this rating is subjective to the author.

- 1 star ★: Simple exercises, suitable for beginners. They should be easy to solve without much effort. Often only transfer of knowledge is required, for example, by writing down things that are in a textbook differently.
- 2 stars ★★: The effort is higher here. Different techniques have to be combined. Greater creativity is required.
- 3 stars ★★★: Assignments with three stars are more complex, require recourse to more prior knowledge, and sometimes require research. Frequently, the tasks can no longer be solved with a single method, but require multiple classes that must work together.

THE SUGGESTED SOLUTIONS

The task book provides at least one suggested solution for each problem. The term "sample solution" is not used to avoid implying that the given solution is the best one and that all other solutions are useless. Readers are encouraged to compare their solutions with the proposed solution and can be satisfied if their solution is more elegant. All proposed solutions are commented, making it possible to follow all steps well.

The suggested solutions are compiled at the end of each chapter to reduce the temptation to look into a solution directly after the task, which takes the fun out of solving the task. The suggested solutions can also be found on the website https://github.com/ullenboom/captain-ciaociao. Some solutions contain comments of the type //tag::solution[], which marks the parts of the solutions printed in the book.

USE OF THE BOOK

To become a software developer, you must master the art of turning problems into code, and that's where practice and role models come in. While there are plenty of exercises available online, they're often disorganized, poorly documented, and outdated. That's where this book shines, by offering a systematic approach to tasks and well-thought-out solutions. Studying these solutions and reading code in general helps the brain develop patterns and solutions that can be applied to future coding challenges. It's like reading the Bible; you need to read to understand and learn. Surprisingly, many software developers write code without bothering to read others' code, which can lead to confusion and misunderstanding. Reading good code elevates our writing skills by building patterns and solutions that our brains unconsciously transfer to our own code. Our brains form neuronal structures independently based on templates, and the quality of the input we receive matters greatly. Therefore, we should only feed our brains with good code, as bad solutions make for bad models. The book covers important topics such as exception handling or error handling, discussing the correct input values, identifying erroneous states, and how to handle them. In software, things can and will go wrong, and we must be prepared to deal with the less-than-perfect world.

It's easy for developers to get stuck in their ways of writing code, which is why it's important to explore new approaches and "expand our vocabulary", so to speak. For Java developers, libraries are their vocabulary, but too many enterprise Java developers write massive, non-object-oriented code. The solution is to continuously improve object-oriented modeling, which is precisely what this book demonstrates. It introduces new methods, creates new data types, and minimizes complexity. Additionally, functional programming is becoming increasingly important in Java development, and all solutions in this book take advantage of modern language features.

While some solutions may appear overly complex, the tasks and proposed solutions in this book can help developers improve their ability to concentrate and follow through with steps. In practice, the ability to concentrate and quickly comprehend code is crucial for developers. Often, developers must join a new team and be able to understand and modify unfamiliar source code, and possibly fix bugs. Those who wish to expand upon existing open-source solutions can also benefit from honing their concentration skills through these exercises.

In addition to its emphasis on the Java programming language, syntax, libraries, and object orientation, this book provides numerous side notes on topics such as algorithms, the historical evolution of programming, comparisons to other programming languages, and data formats. These additional insights and perspectives offer readers a more well-rounded understanding of software development beyond just the technical aspects.

If you're looking for one more reason to add this book to your collection, it doubles as a fantastic sleep aid!

REQUIRED SOFTWARE

While solving a task with just a pen and paper is possible in theory, modern software development requires the proper use of tools. Knowing programming language syntax, object-oriented modeling, and libraries is just the tip of the iceberg. Understanding the JVM, using tools like Maven and Git for version management, and becoming proficient in an IDE are all crucial aspects of professional software development. Some developers can even perform magic in their IDE, generating code and fixing bugs automatically.

USED JAVA VERSION IN THE BOOK

While Java 8 remains prevalent in enterprise settings, it's crucial for learners to become acquainted with the latest language features. Accordingly, whenever feasible, the suggested solutions in this book leverage Java 17. Not only is this version equipped with Long-term Support (LTS), but runtime environment providers also offer extensive support, ensuring that the release retains its relevance for an extended period.

JVM

If we want to run Java programs, we need a JVM. In the early days, this was easy. The runtime environment first came from Sun Microsystems, later from Oracle, which took over Sun. Today, it is much more confusing. Although a runtime environment can still be obtained from Oracle, the licensing terms have changed, at least for Java 8 up to Java 16. Testing and development are possible with the Oracle JDK, but not in production. In this case, Oracle charges license fees. As a consequence, various institutions compile their own runtime environments from the OpenJDK, the original sources. The best known are *Eclipse Adoptium* (https://adoptium.net/), *Amazon Corretto* (https://aws.amazon.com/de/corretto), *Red Hat OpenJDK* (https://developers.redhat.com/products/openjdk/overview) and others such as those from *Azul Systems* or *Bellsoft*. There is no specific distribution that readers are required to follow.

Development Environment

Java source code is just plain text, so technically a simple text editor is all you need. However, relying solely on Notepad or vi for productivity is like trying to win a race on a tricycle. Modern integrated development environments support us with many tasks: color highlighting of keywords, automatic code completion, intelligent error correction, insertion of code blocks, visualization of states in the debugger, and much more. It is therefore advisable to use a full development environment. Four popular IDEs are: *IntelliJ*, *Eclipse*, *Visual Studio Code*, and *(Apache) NetBeans*. Just like with Java runtime environments, the choice of IDE is left to the reader. Eclipse, NetBeans, and Visual Studio Code are all free and open-source, while IntelliJ Community Edition is also free, but the more advanced IntelliJ Ultimate Edition will cost you some cash.

Halfway through the book, we delve into implementing project dependencies using Maven in a few places.

CONVENTIONS

Code is written in `fix width font`, filenames are *italicized*. To distinguish methods from attributes, methods always have a pair of parentheses, such as in "the variable `max` contains the maximum" or "it returns `max()` the maximum". Since methods can be overloaded, either the parameter list is named, as in `equals(Object)`, or an ellipsis abbreviates it, such as in "various `println(…)` methods". If a group of identifiers is addressed, * is written, like `print*(…)` prints something on the screen.

In the suggested solutions, there are usually only the relevant code snippets, so as not to blow up the book volume. The name of the file is mentioned in the listing caption, like this:

VanillaJava.java

```
class VanillaJava { }
```