

中文版主页：<https://bookzhou.com>

豆瓣网：<https://book.douban.com/subject/38239120/>

请扫码购买：

清华大学出版社 好店

微信扫一扫查看商品

DEITEL®开发者系列

AI 高效应用：Java 编程实践（第5版）

[美] 保罗·J. 戴特尔（Paul J. Deitel） 著
哈维·M. 戴特尔（Harvey M. Deitel）
周靖 译

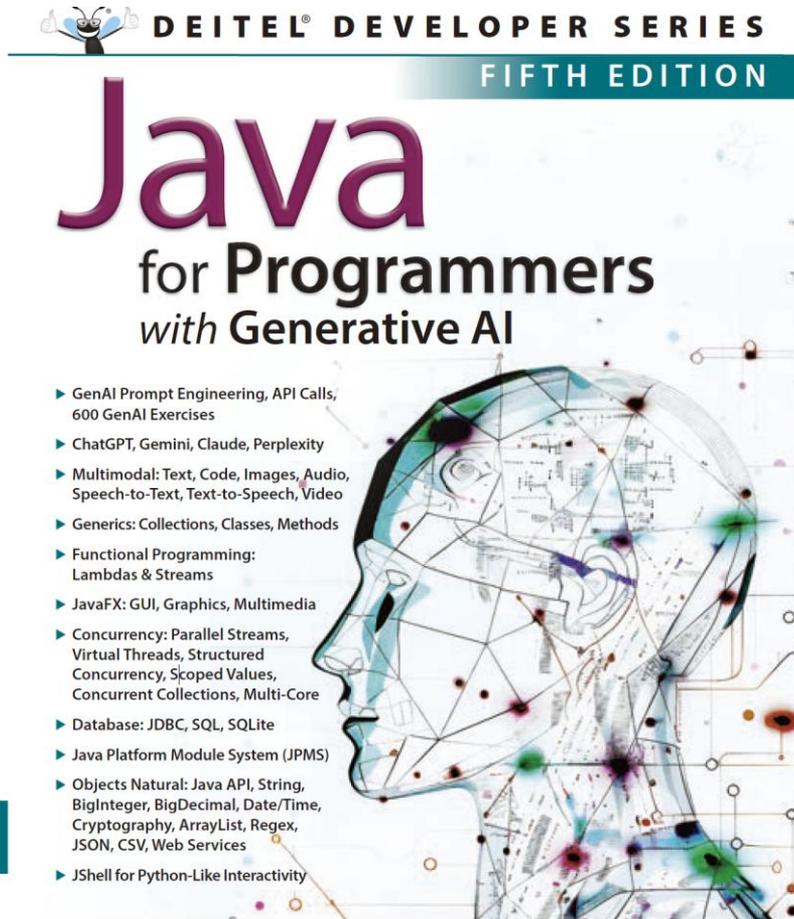
Authorized Translation from the English language edition, entitled JAVA FOR PROGRAMMERS:
WITH GENERATIVE AI 5E BY PAUL DEITEL/ HARVEY M. DEITEL, published by Pearson
Education, Inc, Copyright © 2025 Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any
means, electronic or mechanical, including photocopying, recording or by any information storage
retrieval system, without permission from Pearson Education, Inc.

CHINESE SIMPLIFIED language edition published by TSINGHUA UNIVERSITY PRESS
LIMITED. Copyright © 2025.

AUTHORIZED FOR SALE AND DISTRIBUTION IN THE PEOPLE'S REPUBLIC OF CHINA
ONLY (EXCLUDES TAIWAN, HONG KONG, AND MACAU SAR).

封面



PAUL DEITEL • HARVEY DEITEL

- 生成式 AI 提示工程、API 调用、600 多个生成式 AI 练习
- DeepSeek、ChatGPT、Gemini、Claude、Perplexity
- 多模态：文本、代码、图像、音频、语音转文本、文本转语音、视频
- 泛型：类、接口、方法
- 函数式编程：Lambda 表达式和流
- JavaFX：GUI、图形、多媒体
- 并发：并行流、虚拟线程、结构化并发、作用域值、并发集合、多核
- 数据库：JDBC、SQL、SQLite
- Java 平台模块系统（Java Platform Module System，JPMS）
- 对象：Java API、String、BigInteger、BigDecimal、日期/时间、密码学、ArrayList、正则表达式、JSON、CSV、Web 服务
- 类似于 Python 的交互式解释器 JShell

封底：

Java for Programmers with Generative AI

Features

- GenAI Prompt Engineering, API Calls, 600 GenAI Exercises
- ChatGPT, Gemini, Claude, Perplexity
- Multimodal: Text, Code, Images, Audio, Speech-to-Text, Text-to-Speech, Video
- Generics: Collections, Classes, Methods
- Functional Programming: Lambdas & Streams
- JavaFX: GUI, Graphics, Multimedia
- Concurrency: Parallel Streams, Virtual Threads, Structured Concurrency, Scoped Values, Concurrent Collections, Multi-Core
- Database: JDBC, SQL, SQLite
- Java Platform Module System (JPMS)
- Objects Natural: Java API, String, BigInteger, BigDecimal, Date/Time, Cryptography, ArrayList, Regexp, JSON, CSV, Web Services
- JShell for Python-Like Interactivity

Keep in Touch

- Contact the authors at deitel@deitel.com
- Join the Deitel social media communities
deitel.com/linkedin
facebook.com/DeitelFan
instagram.com/DeitelFan
x.com/deitel
youtube.com/DeitelTV
mastodon.social/@deitel
- For source code and updates, visit:
deitel.com/javafp5

informit.com



The professional programmer's Deitel® guide to Java with integrated generative AI

Written for programmers with a background in another high-level language, in this book, you'll learn **modern Java development** hands on using the **latest Java idioms and features and genAIs**. For more details, see the Preface and its two-page "High-Level Overview."

In the context of **200+ real-world code examples**, you'll quickly master Java fundamentals then move on to arrays, strings, regular expressions, JSON/CSV processing with the Jackson library, private- and public-key cryptography, classes, inheritance, polymorphism, interfaces, dependency injection, exceptions, generic collections, custom generics, functional programming with lambdas and streams, JavaFX GUI, graphics and multimedia, platform threads, virtual threads, structured concurrency, scoped values, building API-based Java genAI apps, database with JDBC and SQLite, the Java Platform Module System and JShell for Python-like interactivity.

Paul Deitel and **Harvey Deitel** are the founders of **Deitel & Associates, Inc.**, the internationally recognized programming languages authoring and corporate training organization. Millions of people worldwide have used Deitel professional books, textbooks, video courses, O'Reilly Online Learning live training (learning.oreilly.com), live corporate training, Pearson+ eTexts (with video) and Revel interactive multimedia courses (with video and autograded assessments) to master Java, Python, C++, C, C#, iOS and Android mobile app development, Internet and web programming, and more.

REVIEWER COMMENTS

"The future of Java programming is here, and this new edition of Deitel is leading the charge! By embracing genAI head-on, the authors are potentially revolutionizing programming education. Through its integrative approach to the use and study of genAI, this book is positioned to be the **leading book in modern Java and its applications**. Indeed, I expect that it should be widely adopted by instructors who want to ingrain in their students an appreciation for the critical role that Java will play in **data science, machine learning, artificial intelligence, and cybersecurity**.

The book's innovative and forward-thinking use of genAI facilitates reader engagement and inspires readers to think critically about the **benefits and limitations of AI as a programming aid**. Chapter 19 could become everyone's favorite new Java book chapter—the **generative AI API-based code examples** are interesting and fun.

All audiences of this book should read the Preface—there's so much to get excited about! It demonstrates, with refreshing transparency and honesty, how much love and care went into the reinvention of an already outstanding Java book by bringing it into a new frontier of what it means to be a programmer in today's world. Bravo! Your Preface statement: "**GenAI has created an ultra-high-level programming capability that will leverage your Java learning experience and ability to produce robust, top-quality Java software quickly, conveniently and economically.**" is a great conclusion to the Preface intro—really helps justify the use of genAI!"

— **Brian Canada, Professor of Computational Science, University of South Carolina Beaufort**

"After reading your whole book, it was fun to read the Preface that wraps everything up at a high level. You have done some amazing work here, and I'm glad to have been a small part of it as a reviewer! I especially appreciate how difficult it must have been to make sure everything was as up to date as possible with the speed at which things change in this field, and the deftness with which you incorporated all the focus on GenAI and data science that's in this book." — **Emily Navarro, Ph.D., Continuing Lecturer, Department of Informatics, University of California, Irvine**

"The generative AI exercises are awesome and reflect the way modern developers work! They are fun and let the reader explore and learn about AI by using AI—how meta. This allows readers to expand their knowledge and get a feel for the AIs' code-related capabilities." — **Jeanne Boyarsky, CodeRanch, Java Champion**

(Continued Inside Back Cover)

ISBN-13: 978-0-13-757473-5

ISBN-10: 0-13-757473-8



9 780137 574735



5 5999

\$59.99 US

本书特色：

- 生成式 AI 提示工程、API 调用、600 多个生成式 AI 练习
- ChatGPT、Gemini、Claude、Perplexity
- 多模态：文本、代码、图像、音频、语音转文本、文本转语音、视频
- 泛型：类、接口、方法
- 函数式编程：Lambda 表达式和流
- JavaFX：GUI、图形、多媒体
- 并发：并行流、虚拟线程、结构化并发、Scoped Value、并发集合、多核
- 数据库：JDBC、SQL、SQLite
- Java 平台模块系统（Java Platform Module System，JPMS）
- 对象：Java API、String、BigInteger、BigDecimal、日期/时间、密码学、

ArrayList、正则表达式、JSON、CSV、Web 服务

- 类似于 Python 的交互式解释器 JShell

保持联系

- 联系作者: deitel@deitel.com
- 加入 Deitel 的社交媒体:
deitel.com/linkedin
facebook.com/DeitelFan
instagram.com/DeitelFan
x.com/deitel
youtube.com/DeitelTV
mastodon.social/@deitel
- 英文版源代码和更新: deitel.com/javafp5
- 中文版源代码和更新: bookzhou.com

Deitel® Java 专业程序员指南：融合生成式 AI 技术

本书专为具备其他高级语言编程经验的开发者编写，通过 200 多个真实世界的代码示例，助您动手学习现代 Java 开发，掌握最新的 Java 惯用法、特性以及生成式 AI 的应用。更多详情请参阅前言及其“高级概览”一节。

通过 200 多个真实世界的代码示例，您将快速掌握 Java 基础知识，然后逐步学习数组、字符串、正则表达式、使用 Jackson 库进行 JSON/CSV 处理、公钥和私钥密码学、类、继承、多态、接口、依赖注入、异常处理、泛型集合、自定义泛型、使用 Lambda 表达式和流进行函数式编程、JavaFX GUI、图形和多媒体、平台线程、虚拟线程、结构化并发、作用域值、构建基于 API 的 Java 生成式 AI 应用、使用 JDBC 和 SQLite 进行数据库操作、Java 平台模块系统以及类似 Python 的交互式 JShell。

保罗·戴特尔 (Paul Deitel) 和哈维·戴特尔 (Harvey Deitel) 是 Deitel & Associates 公司创始人，这是一家国际知名的编程语言著作和企业培训机构。全球数百万人通过 Deitel 的专业书籍、教材、视频课程、O'Reilly 在线学习直播培训 (learning.oreilly.com)、企业现场培训、Pearson+ eTexts (带视频) 和 Revel 交互式多媒体课程 (带视频和自动评分) 来掌握 Java、Python、C++、C、C#、iOS 和 Android 移动应用开发、互联网和 Web 编程等技术。

溢美之辞：

Java 编程的未来已至，Deitel 这部全新力作正引领时代潮流！作者以开拓性姿态拥抱 AI，有望彻底革新编程教育范式。通过将 AI 的应用与研究深度融合，本书势必成为现代 Java 及其应用领域的标杆之作。事实上，我期待本书能被广大教育者广泛采用——他们将借此帮助学生深刻理解 Java 在数据科学、机器学习、人工智能及网络安全等领域所扮演的关键角色。

书中对 AI 的创新性前瞻应用，极大地激发了读者参与感，并引导其批判性思考 AI 作为编程辅助工具的优势与局限。第 19 章或将成为全书亮点——基于 AI API 的代码示例既趣味十足又引人入胜。

所有读者都应认真阅读前言——其中亮点纷呈！它以令人耳目一新的坦率笔触，淋漓尽致地展现了这部卓越 Java 著作如何倾注匠心完成蜕变：引领读者叩响当代程序员的全新境界之门。这份革新之作，当得一声喝彩！前言中那句“AI 创造了一种超高层次的编程能力，这将极大地提升你的 Java 学习体验，并使它能够快速、便捷、经济地开发出健壮、高质量的 Java 软件”堪称点睛之笔——它精准诠释了 AI 的核心价值！

——布莱恩·坎达 (Brian Canada)，南卡罗来纳大学计算机科学教授

通读全书后，再回头阅读高屋建瓴的前言，感受尤为深刻。作者在其中完成了令人赞叹的总结工作，而我很荣幸能作为审稿人为此略尽绵薄之力！尤其值得赞赏的是，鉴于该领域技术迭代速度极快，作者仍确保书中内容尽可能与时俱进，同时巧妙融入对 AI 与数据科学的重点关注——其中的难度不言而喻。

——艾米丽·纳瓦罗 (Emily Navarro) 博士，加利福尼亚大学欧文分校信息学系特聘讲师

这些生成式 AI 练习非常出色，真实还原了现代开发者的工作方式！它们不仅趣味十足，更让读者在使用 AI 的过程中探索和学习 AI。通过这种方式，读者既能拓展知识边界，又能亲身体会 AI 在代码领域的强大能力。

——珍妮·博亚斯基 (Jeanne Boyarsky)，CodeRanch, Java Champion

(接封底内页)

封底内页

Generative AI Innovations in Java for Programmers, 5e Fully Coded GenAI Case Studies	— Reviewer Comments (Continued from Back Cover)
<p>Chapter 19 presents the following code examples that interact with OpenAI APIs: Text Summarization, Sentiment Analysis, Accessible Image Descriptions, Language Detection by Translation, Java Code Generation, Named-Entity Recognition & Structured Outputs, Speech-to-Text, Text-to-Speech, Image Generation, Creating Closed Captions for a Video, Moderation.</p> <p>GenAI Prompt Exercises</p> <p>We find the complete list of all the book's approximately 600 genAI exercises (a 100+ page PDF) in ChatGPT, Gemini, Claude and Perplexity, asking them to categorize the kinds of things we do in those exercises. Next, we fed their categorized lists to the four genAIs, asking them to summarize the summaries, and we chose the best one—Claude in this case:</p> <ul style="list-style-type: none">• Code Generation and Implementation—Writing new Java programs from specifications; Implementing specific features, algorithms and APIs; Creating test programs and practical applications; Generating solutions for basic and advanced tasks.• Code Refactoring and Enhancement—Modernizing code; Improving code structure, readability, and maintainability; Converting between different approaches while maintaining functionality; Improving performance.• Educational Content—Creating tutorials, exercises, and learning materials; Further exploring complex concepts; Developing programming exercises; Writing comprehensive documentation and guides.• Technical Analysis—Analyzing code behavior and feature implementations; Comparing different approaches, tools, and frameworks; Evaluating trade-offs in design decisions; Breaking down complex technical concepts.• Best Practices and Standards—Implementing coding standards and design patterns; Addressing security considerations; Optimizing performance; Following Java development best practices.• Technology Evaluation—Comparing libraries, tools, and frameworks; Assessing the pros and cons of different approaches; Making informed technology choices; Exploring new features and updates.• Debugging and Error Handling—Finding and fixing syntax and logical errors; Implementing exception handling; Improving fault tolerance; Preventing common pitfalls.• API and Library Integration—Working with Java APIs and external libraries; Understanding API features and capabilities; Implementing integration techniques; Creating API documentation and tutorials.• Real-world Applications—Developing practical user cases and industry applications; Creating interactive applications (IGUs, games, multimedia); Implementing real-world scenarios; Building sample projects.• Performance Optimization—Analyzing and improving performance; Optimizing resource usage; Conducting benchmarks; Implementing efficiency improvements.• Creative Development—Exploring multimodal applications; Creating visualizations; Generating test scenarios and sample data; Developing unique user cases. <p>GenAI API-Based Java Programming Exercises</p> <p>Chapter 19: Building API-Based Java Generative AI Applications, suggests challenging project exercises like creating genAI multimedia apps that can debate one another and using genAIs to build and solve crossword puzzles. We fed the 94 exercises into the genAIs, asking for a categorized summary of them, then summarized the summaries. Here's what Claude produced:</p> <ul style="list-style-type: none">• Multimodal Applications—Building comprehensive tools that combine text, images, audio, speech and video capabilities; Creating integrated experiences like interactive books; Developing multimedia educational content.• Text-Based Applications—Document processing (indexing, summarization, exploration); Creative writing (stories, poetry, dialogues); Language tools (translation, text rewriting); Professional document creation (resumes, presentations); Structured outputs.• Image Processing Applications—Generative art and design (logos, fashion, floor plans); Technical visualization (UML, diagrams); Image analysis and recognition.• Audio and Music Applications—Speech processing (transcription, voice cloning); Music generation (MIDI, Magenta AI); Multilingual audio applications; Podcast and audio content analysis.• Educational Tools—Programmatic tutors (java, coding exercises); Subject-specific learning aids (math); Course content creation; Interactive educational experiences.• Games and Puzzle Applications—Puzzle generation and solvers; Interactive game development.• Video—Investigating and experimenting with generative AI video creation tools.• Chatbot Development—Character-based chat experiences; Specialized domain experts.• Research and Analysis Tools—Medical applications (researching drug discovery and personalized medicine); AI capability exploration; Text detector and analysis; Educational research.• Creative Applications—Children's book creation; Interactive storytelling; Artistic content generation; Creative writing.• Practical Tools and Utilities—Document generators; Translation services; Content summarizers; Professional tools (resume filters, presentation creator).	<p>■ Brian Canada, Professor of Computational Science, University of South Carolina Beaufort</p> <p>"Excellent coverage of the latest Java language features. The Shings chapter sets the stage for important applications in NLP (in critical ingredient to making genAI systems work) and cryptography (is critical ingredient to information security) that are introduced by objects-natural case studies in Chapters 7 (private-key cryptography) and 11 (public-key cryptography). The objects-natural case study we chose BigInteger is especially pertinent as it is a time when cybersecurity is more important than ever and programmers need to develop an appreciation for secure software engineering practices.</p> <p>Real-World Modeling with Inheritance, Polymorphism and Interfaces covers a lot but I feel smarter after reading it and going through the associated genAI exercises! These are clearly written to prompt AI to guide the reader to "fill in the gaps" and provide application cases where those techniques are used.</p> <p>The objects-natural case study on <i>ArrayList</i> provides a "road map" at the Java Collections Framework showing how data structure values problems caused by the limitations of Java arrays. The Java Collections Framework chapter does an admirable job covering most key data structures and how Java's built-in collections implement them. This section on <i>hash tables</i> is informative. The coverage of newer JVM techniques, such as <i>concurrency factory methods</i>, is appropriate.</p> <p>The genAI exercises are highly valuable and fun, giving the reader new tools for filling in details and thinking critically about how and why certain practices are followed in Java in particular and object-oriented programming in general. Great job covering "checked" and "unchecked" exceptions. Suggest mechanisms for "hallmarking your code" with validation logs so that certain exceptions aren't thrown in the first place. The genAI prompt exercises on the types of OpenAI API messages posted to the chat completion API were very helpful for cementing my understanding. Accessing descriptions of the book's photos to get school my research group was handy scrubbing the surface of identifying objects in images—things have come a long way! The closed captions done is cool!</p> <p>A broad introduction to fundamental OOP inheritance and polymorphism concepts. The use of interfaces to implement runtime polymorphism logically follows a similar discussion involving abstract classes. I like the discussion of the relative benefits of composition vs. inheritance. Accompanies the best lesson of creating <i>right vs. loose coupling and dependency injection</i> in just a few pages. Great to see the <i>Builder</i> design pattern in action.</p>
	<p>■ Ron Ven, Java Developer, Special Agent at Team Rocket's IT, co-author <i>Virtual Threads, Structured Concurrency, and Scaled Values: Explore Java's New Threading Model</i></p> <p>"This is the best book for learning Java that I have come across in 25+ years of Java experience. I've suggested using it to train developers in my company and in the companies where I'm currently. A detailed explanation of the core Java language. Excellent discussion of inheritance and polymorphism. Includes how to use and incorporate AI in your daily software-development process. Using tools like ChatGPT, Gemini, Claude and Perplexity is now an essential skill to gain a competitive advantage. Awesome Chapter 19 that clearly illustrates the power of AI API-based app development. A clear explanation of Java concurrency. The <i>Big O</i> discussion teaches the cost and consequence of design decisions. Great discussion of creating custom classes. Excellent intro to the <i>Java Platform Module System</i>. Best intro to JSR that I've seen. I like the detailed discussion of how streams and lambdas work and the thorough explanation of exceptions."</p>
	<p>■ Emily Navarra, Ph.D., Continuing Lecturer, Department of Informatics, University of California, Irvine</p> <p>"The book's 'objects-natural' approach introduces substantial subject-matter before diving into custom class creation in later chapters. I like the objects-natural case study with the <i>date/time</i> example—a good forced dive into the Java API documentation. I like that inheritance, interfaces, and composition are covered in the same chapter, as it is a logical progression that tells the story of what is the best approach for an OOP Java program. The <i>Building Java Generative AI API-Based Applications</i> chapter taught me things I did not know. I appreciate how AI body was used to help with the chapter and the text was transparent about it. The 94 end-of-chapter genAI exercises are exciting, creative, and fun. I really enjoyed the <i>Fiber</i> chapter—a highlight in a lot of the latest technologies, such as JSR, CF, networking APIs, and cryptography. Great job introducing and motivating generic class and method concepts and bringing in JSR to support the practical application of Java's generic collection data structures."</p>
	<p>■ Jeanne Boyarsky, CodeRanch, Java Champion</p> <p>"Here the latest Java features to write code in the most modern ways. GenAI can read documents (Prompt genAIs with doc documents), UML class diagrams, asking them to generate the corresponding Java class definitions. I like the genAI 'why questions' and its 'general guidelines to explore to why Java constructs cannot return values.' I don't know GenAI would tell me to be the ones if I didn't ask or could explain how the Java method class works using <i>body</i>, <i>body</i>, <i>body</i> diagrams made of characters. Cool! Great list of NLP applications! The objects-natural NLP topics and cryptography security examples are excellent. They fit the reader use being knowledge while also learning real-world principles. Includes a substantial practical application. The <i>LinkedList</i> module dependency graph is a great real example. The progression of naming the various examples with no precedents, then one, then two is a fun way to show the <i>ServiceLoader</i> really gets products at runtime! I like the number and variety of JSR examples and contexts. I like how lambdas and method references are presented. Focuses on the most common generative use cases and makes them easy to understand. An objects-natural case study on <i>BigDecimal</i>—interesting discussion of how <i>BigDecimal</i> bankers' rounding works for processing monetary amounts. I like the concurrency examples—the applications of virtual threads, structured concurrency and scaled values were good."</p>

溢美之辞：

本书透彻讲解了最新的 Java 语言特性。字符串章节为自然语言处理（NLP，AI 运作的关键要素）和密码学（信息安全的核心要素）的重要应用奠定了基础，这些应用在第 7 章（私钥密码学）和第 11 章（公钥密码学）的“对象自然”案例学习中得到了详细阐述。在网络安全比以往任何时候都更加重要、程序员亟须培养安全软件工程实践意识的当下，关于 **BigInteger** 类的“对象自然”案例学习尤其具有现实意义。

关于继承、多态和接口的真实世界建模内容丰富，通读并完成相关的生成式 AI 练习后，我感觉自己都变聪明了！这些练习设计得非常巧妙，通过提示词引导 AI 辅助读者“填补知识空白”，同时结合技术应用场景实例，帮助读者深入理解并掌握这些核心概念的实操方法。

关于 **ArrayList** 的“对象自然”案例学习，让我们得以对 **Java 集合框架**（**Java Collections Framework**）“先睹为快”——它清晰展现了该数据结构如何有效解决 **Java 数组** 的局限性。**Java 集合框架** 章节出色地重写了大多数关键数据结构，并详细阐释了 **Java 内置集合** 对这些结构的实现方式。书中关于哈希表的部分内容翔实，对较新的 **JDK 技术**（如便捷工厂方法）的介绍也恰到好处。

生成式 AI 练习既极具价值又充满趣味，为读者提供了填补细节的新工具，并引导他们批判性地思考 **Java 特定实践** 及面向对象编程通用原则背后的逻辑。书中对“受检”（checked）和“非受检”（unchecked）异常的对比讲解十分透彻，并提出通过验证逻辑“强化代码防

护”的机制，从源头避免某些异常的抛出。关于传递给“聊天完成 API”的 OpenAI API 消息类型的生成式 AI 练习，对巩固理解非常有帮助。书中对海滩照片的 AI 描述令人惊叹！在研究生阶段，我的研究小组在图像目标识别方面仅能触及皮毛——如今技术发展已一日千里！字幕演示也十分精彩！

本书对面向对象编程中继承和多态的基本概念进行了广泛介绍。关于使用接口实现运行时多态的讨论，在逻辑上很好地衔接了对抽象类的类似探讨。我非常喜欢书中对组合与继承相对优势的讨论，能够在短短几页内完成紧耦合与松耦合、依赖注入等复杂概念的阐述，实属难能可贵。很高兴看到 Builder（建造者）设计模式的实际应用。

——布莱恩·坎达（Brian Canada），南卡罗来纳大学计算机科学教授

这是我在 25 年 Java 开发经历中遇到的最优秀的 Java 学习书籍。我已推荐我所在的公司及提供咨询服务的企业使用本书来培训开发者。书中不仅对 Java 核心语言进行了翔实的解析，对继承与多态的探讨也尤为出色，更教会了读者如何在日常软件开发中应用并整合 AI 技术。如今，掌握 DeepSeek 和 ChatGPT 等 AI 已成为构建竞争优势的必备技能，而精彩的第 19 章则清晰展现了基于 AI API 的应用程序开发的强大潜力。

书中对 Java 并发机制的阐释简明易懂，关于 Big O 表示法的讨论则深入剖析了设计决策的成本与影响。自定义类的讲解细致入微，Java 平台模块系统（Java Platform Module System, JPMS）的介绍堪称典范，更是我所见过的最出色的 JShell 入门指南。此外，书中对流和 Lambda 表达式工作原理的详细探讨，以及对异常处理的透彻解析，均展现了极高的专业水准。

——罗恩·维恩（Ron Veen），Java 开发者，Team Rockstars IT Special Agent，《虚拟线程、结构化并发和作用域值：探索 Java 新线程模型》（*Virtual Threads, Structured Concurrency, and Scoped Values: Explore Java's New Threading Model*）作者之一

本书采用“对象自然”教学法，在深入讲解后续章节的自定义类创建之前，先介绍了大量现成的 Java 类及对象交互。我尤其欣赏日期/时间示例的“对象自然”案例学习——它堪称一次对 Java API 文档的“强制性”深度探索，有效引导读者掌握查阅官方文档的能力。书中将继承、接口和组合置于同一章讲解，这种逻辑递进的方式清晰展现了构建面向对象 Java 程序的最佳实践路径。

第 19 章让我收获了许多全新知识。作者坦诚展现了如何借助 AI 工具辅助撰写本章内容，这种透明化的处理方式令人赞赏。章末的 94 个生成式 AI 练习充满创意与趣味，激发读者

主动探索。关于文件处理的章节同样亮点十足，融合了 JSON、CSV、网络 API 和密码学等前沿技术，充分体现了 Java 在现代数据处理场景中的多元应用。此外，本书在引入泛型类和方法概念时，巧妙结合计算机科学理论支撑 Java 泛型集合数据结构的实际应用，实现了理论与实践的有机统一。

——艾米丽·纳瓦罗（Emily Navarro）博士，加利福尼亚大学欧文分校信息学系特聘讲师

本书采用最新 Java 特性，以最前沿的方式编写代码。生成式 AI 甚至能够解读图表！（向生成式 AI 提示 Account 类的 UML 类图，即可生成对应的 Java 类声明。）我尤其欣赏书中向生成式 AI 提出的“为什么”问题，例如“要求生成式 AI 解释为什么 Java 构造函数不能返回值”。令人惊喜的是，即使未主动询问，生成式 AI 也会主动提示错误修复方案，甚至能用纯文本逐行绘制字符图表，解释 Java 方法调用栈的工作原理——这太酷了！书中罗列的 NLP 应用场景十分精彩，而以“对象自然”方式呈现的 NLP、正则表达式和密码学安全示例更是堪称典范：读者既能运用字符串知识，又能从中学习真实世界的技术原理。

书中对 JavaFX 进行了大篇幅专业讲解：关于如何在 Scene Builder 中设置 JavaFX 应用程序的分步演示堪称教科书级别，JavaFX 布局表格的解析也十分透彻。值得称赞的是，JavaFX 章节还涵盖了辅助功能设计的内容，体现了对全场景开发的考量。模块系统的介绍同样可圈可点——在重写基础概念（甚至延伸更多细节）的同时，始终保持通俗易懂的叙述风格。书中纳入模块化应用程序的结构解析，VideoPlayer 模块依赖图更是极具参考价值的真实案例。通过“无提供者→单个提供者→两个提供者”的递进式服务示例运行演示，生动展现了 ServiceLoader 在运行时动态选择提供者的机制，这种讲解方式趣味与专业性兼具！

JShell 示例与练习的丰富性和多样性令人印象深刻，Lambda 表达式和方法引用的呈现方式也恰到好处。本书聚焦于最常见的泛型应用场景，将复杂概念拆解为易于理解的知识单元。关于 BigDecimal 的“对象自然”案例学习——对银行家舍入法（四舍六入五成双）处理货币金额的探讨，既贴合实际开发场景又体现技术深度。并发编程部分同样亮点频现：对虚拟线程、结构化并发和作用域值的阐释精准到位，充分展现了 Java 在现代高性能编程中的核心优势。

——珍妮·博亚斯基（Jeanne Boyarsky），CodeRanch，Java Champion

Generative AI Innovations in Java for Programmers, 5e

Fully Coded GenAI Case Studies

Chapter 19 presents the following code examples that interact with OpenAI's APIs: Text Summarization, Sentiment Analysis, Accessible Image Descriptions, Language Detection & Translation, Java Code Generation, Named-Entity Recognition & Structured Outputs, Speech-to-Text, Text-to-Speech, Image Generation, Creating Closed Captions for a Video, Moderation.

GenAI Prompt Exercises

We fed the complete list of all the book's approximately 600 genAI exercises (a 100+ page PDF) to ChatGPT, Gemini, Claude and Perplexity, asking them to categorize the kinds of things we do in those exercises. Next, we fed their categorized lists to the four genAIs, asking them to summarize the summaries, and we chose the best one—Claude in this case:

- **Code Generation and Implementation**—Writing new Java programs from specifications. Implementing specific features, algorithms and APIs. Creating test programs and practical applications. Generating solutions for basic and advanced tasks.
- **Code Refactoring and Enhancement**—Modernizing code. Improving code structure, readability, and maintainability. Converting between different approaches while maintaining functionality. Improving performance.
- **Educational Content**—Creating tutorials, exercises, and learning materials. Further exploring complex concepts. Developing programming exercises. Writing comprehensive documentation and guides.
- **Technical Analysis**—Analyzing code behavior and feature implementations. Comparing different approaches, tools, and frameworks. Evaluating trade-offs in design decisions. Breaking down complex technical concepts.
- **Best Practices and Standards**—Implementing coding standards and design patterns. Addressing security considerations. Optimizing performance. Following Java development best practices.
- **Technology Evaluation**—Comparing libraries, tools, and frameworks. Assessing the pros and cons of different approaches. Making informed technology choices. Exploring new features and updates.
- **Debugging and Error Handling**—Finding and fixing syntax and logical errors. Implementing exception handling. Improving fault tolerance. Preventing common pitfalls.
- **API and Library Integration**—Working with Java APIs and external libraries. Understanding API features and capabilities. Implementing integration techniques. Creating API documentation and tutorials.
- **Real-world Applications**—Developing practical use cases and industry applications. Creating interactive applications (GUIs, games, multimedia). Implementing real-world scenarios. Building sample projects.
- **Performance Optimization**—Analyzing and improving performance. Optimizing resource usage. Conducting benchmarks. Implementing efficiency improvements.
- **Creative Development**—Building multimodal applications. Creating visualizations. Generating test scenarios and sample data. Developing unique use cases.

GenAI API-Based Java Programming Exercises

Chapter 19, Building API-Based Java Generative AI Applications, suggests challenging project exercises like creating genAI multimedia apps that can debate one another and using genAIs to build and solve crossword puzzles. We fed the 94 exercises into the genAIs, asking for a categorized summary of them, then summarized the summaries. Here's what Claude produced:

- **Multimodal Applications**—Building comprehensive tools that combine text, image, audio, speech and video capabilities. Creating integrated experiences like interactive books. Developing multimedia educational content.
- **Text-Based Applications**—Document processing (indexing, summarization, exploration). Creative writing (stories, poetry, debates). Language tools (translation, tone rewriting). Professional document creation (resumes, presentations). Structured outputs.
- **Image Processing Applications**—Generative art and design (logos, fashion, floor plans). Technical visualization (UML diagrams). Image analysis and recognition.
- **Audio and Music Applications**—Speech processing (transcription, voice cloning). Music generation (MIDI, Magenta AI). Multilingual audio applications. Podcast and audio content analysis.
- **Educational Tools**—Programming tutors (Java, coding exercises). Subject-specific learning aids (math). Course content creation. Interactive educational experiences.
- **Gaming and Puzzle Applications**—Puzzle generators and solvers. Interactive game development.
- **Video**—Investigating and experimenting with generative AI video creation tools.
- **Chatbot Development**—Character-based chat experiences. Specialized domain experts.
- **Research and Analysis Tools**—Medical applications (researching drug discovery and personalized medicine). AI capability exploration. Text detection and analysis. Educational research.
- **Creative Applications**—Children's book creation. Interactive storytelling. Artistic content generation. Creative writing.
- **Practical Tools and Utilities**—Document generators. Translation services. Content summarizers. Professional tools (resume filters, presentation creators).

提供完整代码的生成式 AI 案例学习

第 19 章提供了以下与 OpenAI API 交互的完整代码示例：文本摘要、情感分析、无障碍图像描述、语言检测与翻译、Java 代码生成、命名实体识别与结构化输出、语音转文本、文本转语音、图像生成、为视频创建隐藏式字幕以及内容审核。

生成式 AI 练习

我们将书中约 600 个生成式 AI 练习的完整列表（一份超百页的 PDF 文档）输入到 ChatGPT、Gemini、Claude 和 Perplexity 中，让它们对这些练习涉及的内容进行分类。随后，我们将分类结果再次输入这 4 个 AI，生成最终摘要并选出最优版本——以下为 Claude 的分类结果。

- **代码生成与实现**：根据需求规范编写全新 Java 程序，涵盖特定语言特性、算法及 API 的实现，创建测试程序与实际应用，为基础及高级任务提供解决方案。
- **代码重构与增强**：实现代码现代化，优化代码结构、可读性和可维护性，在保持功能性的前提下完成不同实现方法间的转换，并提升程序性能。
- **教育内容开发**：创建教程、练习及学习材料，深入探索复杂概念，设计编程练习，撰写全面的文档与指南。
- **技术分析**：分析代码行为与功能实现逻辑，对比不同方法、工具及框架，评估设计决策中的权衡取舍，拆解复杂技术概念。
- **最佳实践与标准**：贯彻编码标准与设计模式，处理安全相关考量，优化性能表现，遵循 Java 开发最佳实践。
- **技术评估**：对比各类库、工具及框架，评估不同方案的优缺点，辅助做出明智的技术选型，探索新功能与更新。
- **调试与错误处理**：排查并修复语法及逻辑错误，实现异常处理机制，提升系统容错能力，预防常见开发陷阱。
- **API 与库集成**：使用 Java 原生 API 及外部库，理解 API 功能与特性，实现集成技术，创建 API 文档与教程。
- **真实场景应用开发**：开发实际用例与行业应用，构建交互式程序（如图形界面、游戏、多媒体应用等），复现真实世界场景，创建示例项目。
- **性能优化**：分析并提升程序性能，优化资源使用效率，执行基准测试，实施效率改进措施。
- **创意开发**：构建多模态应用程序，生成可视化效果，创建测试场景与示例数据，开发独特用例。

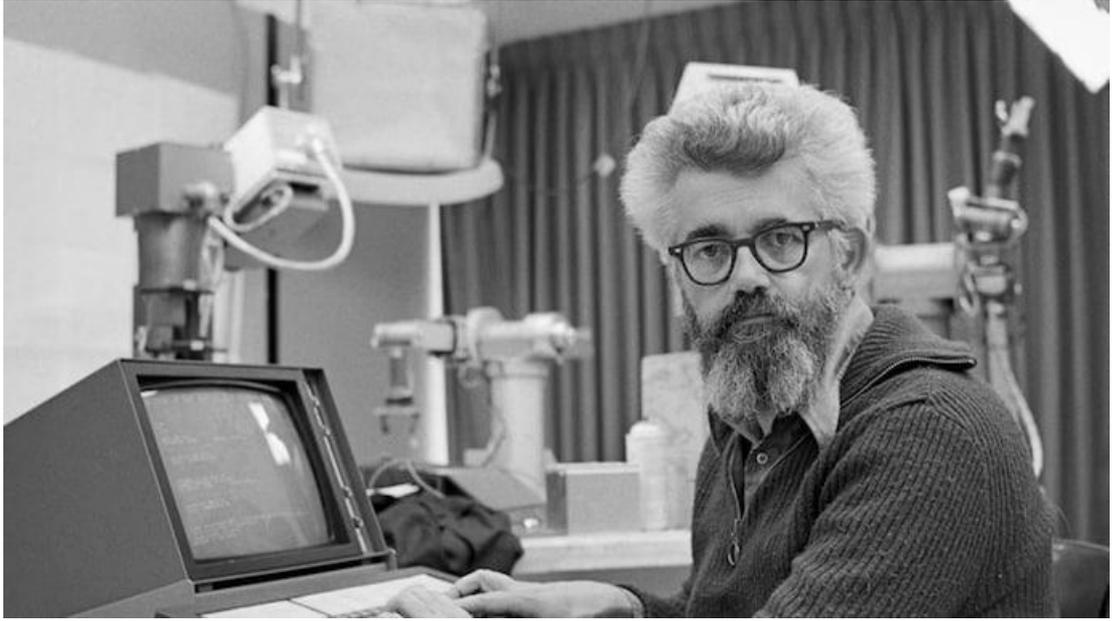
基于生成式 AI API 的 Java 编程练习

第 19 章提出了一些具有挑战性的项目练习，例如创建可以相互辩论的生成式 AI 多媒体应用程序，以及使用生成式 AI 构建和解决填字游戏。我们将这 94 道题提供给 AI，要求它们提供分类摘要，然后我们对这些摘要进行了总结。以下是由 Claude 生成的内容。

- **多模态应用程序**：构建结合文本、图像、音频、语音和视频功能的综合工具；创建如互动书籍等集成体验；开发多媒体教育内容。
- **基于文本的应用程序**：文档处理（索引、摘要、探索）；创意写作（故事、诗歌、辩论）；语言工具（翻译、语气重写）；专业文档创建（简历、演示文稿）；结构化输出。
- **图像处理应用程序**：生成式艺术与设计（标志、时尚、平面图）；技术可视化（UML 图）；图像分析与识别。
- **音频和音乐应用程序**：语音处理（转录、语音克隆）；音乐生成（MIDI、Magenta AI）；多语言音频应用程序；播客和音频内容分析。
- **教育工具**：编程导师（Java、编码练习）；特定学科学习辅助工具（数学）；课程内容创建；交互式教育体验。
- **游戏和拼图应用程序**：拼图生成器和解谜器；交互式游戏开发。

-
- **视频**：调查和实验生成式 AI 视频创建工具。
 - **聊天机器人开发**：基于角色的聊天体验；专业领域专家。
 - **研究和分析工具**：医疗应用（研究药物发现和个性化医疗）；AI 能力探索；文本检测和分析；教育研究。
 - **创意应用程序**：儿童书籍创作；互动故事讲述；艺术内容生成；创意写作。
 - **实用工具和实用程序**：文档生成器；翻译服务；内容摘要器；专业工具（简历筛选器、演示文稿创建器）。

献辞



谨以此书，追思斯坦福先贤约翰·麦卡锡。

公首创“人工智能”之嘉名，更燃思想之炬，光照后人前行之路。

昔日畅想，今已成与我辈共事之智者。愿此书所授，能承公之遗泽，续写华章，是为纪念。

——保罗·戴特尔

——哈维·戴特尔

目录

译者序：AI 赋能的 Java 学习之道	50
前言	53
丰富多彩的真实世界 Java 应用	55
本书面向的读者	56
“实时编码”教学方法	57
高级概览	57
编程范式	57
第 1 部分：导论	57
第 2 部分：编程基础	57
第 3 部分：面向对象编程	58
第 4 部分：数据结构、泛型集合、Lambda 表达式与流	58
第 5 部分：JavaFX 图形用户界面、图形与多媒体	59
第 6 部分：高级主题	59
附录	60
Deitel.com 博客主题	60
“对象自然”教学法	61
什么是“对象自然”教学法？	61
大量可用的免费类库	61
生成式 AI 教学法	61
使用生成式 AI 的两种方法	62
我们使用的生成式 AI 工具	62
生成式 AI 在教育中的角色	63

通过生成式 AI 增强学习体验	63
我们是如何使用生成式 AI 的?	64
给生成式 AI 用户的建议	64
幻觉与错误	65
为什么要使用付费服务.....	65
本书的主旨	65
特殊的第 19 章	66
生成式 AI 练习	67
基于生成式 AI API 的 Java 编程练习	67
生成式 AI 教学法的要点	68
<i>本书增改内容</i>	68
<i>本书特色</i>	70
语法着色与高亮显示.....	70
正文中的代码.....	70
学习目标	70
插图与图表	70
强调 Java 的标准用法.....	70
把握现代 Java 技术脉络.....	70
<i>数百个现代示例与生成式 AI 练习</i>	71
性能优化	71
数据实战体验.....	71
安全 Java 编程.....	71
隐私保护	71
伦理	71
编程智慧	72

本书导读.....	72
第 1 部分：导论	72
第 2 部分：编程基础.....	73
第 3 部分：面向对象编程.....	75
第 4 部分：数据结构、泛型集合、Lambda 表达式与流	77
第 5 部分：JavaFX 图形用户界面、图形与多媒体	78
第 6 部分：高级主题.....	79
附录	81
<i>重要的开发者网站：GitHub 与 StackOverflow</i>	<i>82</i>
<i>Deitel.com 网站与 Deitel 博客.....</i>	<i>83</i>
本书使用的软件.....	83
Java 文档链接	83
有问题在哪里寻找答案?	84
生成式 AI	84
与作者沟通	84
与译者沟通	84
<i>加入 Deitel & Associates 公司的社交媒体社区.....</i>	<i>84</i>
致谢.....	85
审稿人	85
特别感谢布莱恩·戈茨.....	86
特别感谢罗伯特·菲尔德.....	86
谷歌搜索	86
Grammarly	87
生成式 AI	87

甲骨文、Gluon 和 OpenJFX 社区.....	87
作者简介.....	87
Deitel & Associates 公司简介.....	88
准备工作.....	90
获取代码示例.....	90
Java 开发工具包(JDK).....	91
Windows 用户.....	91
macOS 用户.....	92
Java 的发布节奏.....	93
Java 集成开发环境(IDE).....	93
IntelliJ IDEA 社区版.....	93
Eclipse IDE for Java Developers.....	94
Visual Studio Code.....	94
JavaFX SDK.....	94
下载 JavaFX SDK.....	95
从 ZIP 文件中解压缩 JavaFX SDK.....	95
Scene Builder.....	96
生成式 AI.....	97
注册账号.....	97
免费版与付费版.....	97
使用 AI 聊天机器人.....	98
第 1 部分 导论.....	99
第 1 章 概述、Java 应用程序实操演练与生成式 AI.....	99
1.1 导读.....	99
1.2 硬件.....	102

1.3 Java.....	104
1.4 面向对象简单回顾.....	106
1.5 Java API 和开源库.....	109
1.6 典型的Java 程序开发环境.....	110
1.7 使用JDK 运行Java 程序.....	114
1.7.1 使用 javac 和 java 命令编译并运行程序	114
1.7.2 直接使用 java 命令运行程序	117
1.8 互联网、万维网、云计算和物联网.....	118
1.8.1 互联网：网络的网络.....	118
1.8.2 万维网：使互联网更好用.....	119
1.8.3 云计算.....	119
1.8.4 物联网.....	120
1.8.5 边缘计算.....	120
1.8.6 混搭.....	121
1.9 元宇宙.....	121
1.9.1 虚拟现实(VR).....	122
1.9.2 增强现实(AR).....	122
1.9.3 混合现实(MR).....	123
1.9.4 区块链技术.....	123
1.9.5 比特币与加密货币.....	124
1.9.6 以太坊平台.....	124
1.9.7 非同质化代币（NFT）	124
1.9.8 Web3.....	125
1.10 软件开发技术.....	125

1.11 数据分析与数据科学.....	126
1.12 大数据到底有多大?	127
1.12.1 大数据分析.....	130
1.12.2 数据科学与大数据正在改变世界：应用场景	131
1.13 人工智能(AI)——计算机科学与数据科学的交汇点.....	132
1.13.1 人工智能.....	132
1.13.2 通用人工智能(AGI).....	133
1.13.3 人工智能里程碑事件	133
1.13.4 机器学习.....	134
1.13.5 深度学习.....	134
1.13.6 强化学习.....	135
1.14 生成式 AI	135
1.15 小结.....	141
专题：生成式 AI 练习.....	142
第 2 章 Java 编程入门.....	146
2.1 导读.....	146
“对象自然”学习方法.....	146
2.2 第一个 Java 程序：显示单行文本.....	147
2.2.1 编译程序.....	152
2.2.2 执行程序.....	153
2.3 修改第一个程序.....	154
2.4 使用 printf 输出文本.....	156
2.5 另一个程序：整数相加.....	157
2.6 算术运算.....	162
2.7 决策：相等性和关系操作符.....	165

2.8 “对象自然”案例学习：创建和使用Java API String 类的对象.....	169
2.9 小结.....	173
专题：生成式AI 练习.....	173
第2部分 编程基础.....	174
第3章 控制语句（上）.....	174
3.1 导读.....	174
3.2 控制结构.....	174
3.2.1 顺序结构.....	175
3.2.2 选择语句.....	176
3.2.3 循环语句.....	176
3.2.4 控制语句小结.....	177
3.3 if 单选语句.....	177
3.4 if...else 双选语句.....	178
3.4.1 嵌套 if...else 语句.....	179
3.4.2 代码块.....	180
3.4.3 条件操作符(?:).....	182
3.5 while 循环语句.....	182
3.6 计数器控制的循环.....	184
3.6.1 实现算法.....	185
3.6.2 整数除法和截断.....	187
3.6.3 算术溢出.....	187
3.6.4 输入校验.....	188
3.7 哨兵值控制的循环.....	188
3.7.1 实现哨兵值控制的循环.....	189

3.7.2 基元类型之间的显式和隐式转换.....	192
3.7.3 浮点数的格式化.....	192
3.8 嵌套控制语句.....	194
3.8.1 问题陈述.....	194
3.8.2 实现程序.....	195
3.9 复合赋值操作符.....	198
3.10 递增和递减操作符.....	199
3.11 基元类型.....	202
3.12 “对象自然”案例学习: 超大整数.....	203
3.13 小结.....	207
专题: 生成式AI 练习.....	207
第4章 控制语句(下)	209
4.1 导读.....	209
4.2 计数器控制的循环的本质.....	209
4.3 for 循环语句.....	211
4.4 for 语句示例.....	214
4.5 用for 累加偶数.....	215
4.6 复利计算.....	216
4.7 do...while 循环语句.....	220
4.8 switch 多选语句.....	221
4.9 break 和continue 语句.....	228
4.10 条件和逻辑操作符.....	231
4.10.1 条件 AND(&&)操作符.....	231
4.10.2 条件 OR()操作符	232
4.10.3 复杂条件的短路求值.....	233

4.10.4 逻辑 AND(&)和逻辑 OR()操作符	233
4.10.5 逻辑 XOR(^)操作符	234
4.10.6 逻辑非(!)操作符	234
4.10.7 示例：生成条件/逻辑操作符真值表	235
4.11 “对象自然”案例学习：使用 <code>BigDecimal</code> 类进行精确货币计算	237
4.12 小结.....	243
专题：生成式 AI 练习	243
第 5 章 方法.....	245
5.1 导读.....	245
5.2 声明方法.....	246
5.3 方法的声明和使用注意事项.....	251
5.4 案例学习：随机数生成.....	251
5.5 案例学习：博弈游戏和枚举.....	257
5.6 作用域.....	263
5.7 方法重载.....	266
5.8 <code>Math</code> 类：静态方法与变量.....	268
5.9 <code>Java API</code> 包	271
5.10 方法调用栈与激活记录.....	273
5.11 参数提升与强制类型转换.....	277
5.12 “对象自然”案例学习： <code>Java</code> 日期/时间 API	279
5.12.1 日期/时间操作	280
5.12.2 以不同的区域设置运行程序	285
5.13 小结.....	286
专题：生成式 AI 练习	287

第 6 章 数组与 ArrayList	288
6.1 导论.....	288
6.2 基元类型与引用类型.....	289
6.3 数组.....	290
6.4 创建和声明和创建.....	291
6.5 数组的初始化.....	292
6.6 数组初始化器.....	294
6.7 计算数组元素的值.....	295
6.8 数组元素求和.....	296
6.9 使用条形图可视化数组数据.....	297
6.10 数组元素作为计数器使用.....	299
6.11 分析调查结果.....	301
6.12 异常处理.....	303
6.12.1 try 语句.....	303
6.12.2 catch 块.....	304
6.13 增强 for 语句：数组元素求和.....	305
6.14 将数组传递给方法.....	307
6.15 传值和传引用.....	310
6.16 多维数组.....	311
6.17 可变长度参数列表.....	316
6.18 命令行参数.....	317
6.19 Arrays 类.....	320
6.20 “对象自然”案例学习：集合和 ArrayList 类.....	323
6.21 小结.....	328
专题：生成式 AI 练习.....	329

第 7 章 字符串、NLP 和正则表达式：生成式 AI 基础	331
7.1 导论.....	331
7.2 字符和字符串基础.....	332
7.3 <i>String</i> 类.....	333
7.3.1 创建 <i>String</i> 对象	333
7.3.2 <i>String</i> 的 <code>length</code> 、 <code>charAt</code> 和 <code>getChars</code> 方法.....	335
7.3.3 比较字符串	336
7.3.4 在字符串中定位字符和子串	343
7.3.5 从字符串中提取子串	345
7.3.6 拼接字符串	346
7.3.7 其他 <i>String</i> 方法	347
7.3.8 <i>String</i> 的 <code>valueOf</code> 方法	349
7.4 <i>StringBuilder</i> 类.....	351
7.4.1 创建 <i>StringBuilder</i> 对象	351
7.4.2 <code>length</code> 、 <code>capacity</code> 、 <code>setLength</code> 和 <code>ensureCapacity</code> 方法.....	352
7.4.3 <code>charAt</code> 、 <code>setCharAt</code> 、 <code>getChars</code> 和 <code>reverse</code> 方法.....	353
7.4.4 <code>append</code> 方法	355
7.4.5 插入和删除方法.....	357
7.5 <i>Character</i> 类.....	359
7.5.1 字符测试和大小写转换方法	359
7.5.2 字符/数字转换	361
7.5.3 <i>Character</i> 类的其他方法.....	362
7.6 字符串词元化.....	363
7.7 自然语言处理(NLP)简介：生成式 AI 基础.....	365

7.8 “对象自然”案例学习：NLP 中的正则表达式.....	366
7.8.1 将整个字符串与模式匹配.....	367
7.8.2 替换子串.....	372
7.8.3 使用 Pattern 和 Matcher 类查找匹配项.....	372
7.8.4 为训练 AI 与 LLM 准备文本数据.....	376
7.9 “对象自然”安全案例学习：pMa5tfEKwk59dTvC04Ft1IFQz9mEXnkfYXZwxk4ujGE=.....	379
7.10 小结.....	388
第 3 部分 面向对象编程.....	389
第 8 章 使用自定义类进行现实世界建模.....	389
8.1 导论.....	389
8.2 实例方法和赋值/取值方法.....	390
8.2.1 定义了实例变量和赋值/取值方法的 Account 类.....	390
8.2.2 在 AccountTest 类中创建和使用 Account 对象.....	393
驱动类 AccountTest.....	394
8.2.3 编译和执行包含多个类的程序.....	396
8.2.4 Account 类的 UML 类图.....	397
8.2.5 关于 AccountTest 类的二三事.....	398
8.2.6 私有实例变量与公共赋值/取值方法的软件工程实践.....	399
8.3 默认和显式实例变量初始化.....	400
8.4 Account 类：使用构造函数初始化对象.....	401
8.4.1 声明自定义构造函数来初始化类的对象.....	401
8.4.2 AccountTest 类：初始化新的 Account 对象.....	402
8.5 具有余额的 Account 类.....	404
8.5.1 具有实例变量 balance 的 Account 类.....	404
8.5.2 AccountTest 类.....	406

8.6 案例学习: 洗牌和发牌模拟.....	409
8.7 案例学习: Time 类.....	414
8.8 控制对成员的访问.....	419
8.9 使用 this 来引用当前对象的成员.....	420
8.10 案例学习: Time 类的重载构造函数.....	423
8.11 默认构造函数和无参构造函数.....	430
8.12 赋值和取值方法注意事项.....	431
8.13 组合.....	433
8.14 枚举类型.....	436
8.15 垃圾回收.....	439
8.16 类的静态成员.....	440
8.17 静态导入.....	444
8.18 final 实例变量.....	445
8.19 包访问权限.....	447
8.20 记录类.....	448
8.20.1 switch 表达式的模式匹配.....	451
8.20.2 在 switch 表达式中解构记录.....	453
8.21 小结.....	454
第 9 章 使用继承、多态性和接口进行真实世界建模.....	456
9.1 导论.....	456
9.2 超类和子类.....	458
9.3 超类和子类之间的关系.....	461
9.3.1 创建和使用 SalariedEmployee 类.....	461
9.3.2 创建 SalariedEmployee 和 SalariedCommissionEmployee 类继承层次结构..	466

9.4 Object 类.....	472
9.5 多态性简介：多态电子游戏.....	473
9.6 演示多态行为.....	474
9.7 抽象类和方法.....	477
9.8 案例学习：使用多态工资系统.....	479
9.8.1 抽象超类 Employee	481
9.8.2 具体子类 SalariedEmployee.....	482
9.8.3 具体子类 CommissionEmployee	483
9.8.4 多态处理、instanceof 操作符和向下转型	485
9.9 final 方法和类.....	490
9.10 构造函数调用实例方法时的注意事项.....	491
9.11 创建和使用接口.....	492
9.11.1 开发 Payable 继承层次结构	494
9.11.2 Payable 接口	495
9.11.3 Invoice 类	495
9.11.4 修改 Employee 类以实现 Payable 接口	497
9.11.5 使用 Payable 接口多态地处理发票和员工.....	498
9.11.6 一些常见的 Java API 接口.....	500
9.12 其他接口特性.....	501
9.12.1 默认接口方法.....	502
9.12.2 静态接口方法.....	502
9.12.3 函数式接口.....	502
9.12.4 私有接口方法.....	503
9.13 针对接口编程，而非针对实现编程.....	503
9.13.1 CompensationModel 接口	505

9.13.2 Employee 中包含 CompensationModel——组合和依赖注入	505
9.13.3 CompensationModel 的实现	506
9.13.4 测试 CompensationModel 层次结构.....	509
9.13.5 依赖注入的优势.....	511
9.13.6 接口与抽象类.....	511
9.14 密封类和接口.....	512
9.15 私有构造函数.....	515
9.16 受保护成员.....	516
9.17 小结.....	517
专题：生成式 AI 练习.....	518
第 10 章 深入异常处理.....	520
10.1 导论.....	520
10.2 示例：未进行异常处理的除以零.....	522
10.3 示例：处理 ArithmeticException 和 InputMismatchException	525
10.4 Java 异常层次结构.....	530
10.5 受检异常与非受检异常.....	531
10.6 finally 块.....	534
10.7 栈展开和从异常中获取信息.....	537
10.8 链式异常.....	540
10.9 声明自定义异常.....	543
10.10 前置条件和后置条件.....	544
10.11 断言.....	545
10.12 try-with-resources 语句：自动资源释放.....	547
10.13 catch 处理程序中的未命名变量.....	548

10.14 小结.....	549
专题: 生成式AI 练习.....	550
第 11 章 文件、I/O 流、JSON 序列化与 CSV 文件.....	552
11.1 导论.....	552
11.2 文件和流.....	553
11.3 使用 NIO 类和接口获取文件和目录信息.....	556
11.4 顺序文本文件.....	560
11.4.1 创建顺序文本文件.....	560
11.4.2 从顺序文本文件读取数据.....	564
11.4.3 更新顺序文件.....	565
11.5 案例学习: JSON 序列化.....	566
11.5.1 序列化 List<Account>.....	568
11.5.2 反序列化 List<Account>.....	570
11.6 案例学习: 处理来自 Web 服务的 JSON 响应.....	572
11.7 案例学习: 创建和读取 CSV 文件.....	582
11.7.1 创建 CSV 文件.....	583
11.7.2 读取 CSV 文件.....	584
11.8 案例学习: 读取和分析包含泰坦尼克号灾难数据集的 CSV 文件.....	587
11.9 对象自然安全案例学习: RSA 公钥密码学.....	595
11.10 小结.....	603
专题: 生成式AI 练习.....	604
第 4 部分 数据结构、泛型集合、Lambda 表达式与流.....	606
第 12 章 泛型集合.....	606
12.1 导论.....	606
12.2 集合概述.....	607

12.3 类型包装类.....	608
12.4 装箱与拆箱.....	609
12.5 列表.....	609
12.5.1 ArrayList 和 Iterator	610
12.5.2 LinkedList.....	614
12.5.3 集合视图和数组的 asList 方法.....	618
12.6 Collections 的方法一览.....	620
12.6.1 sort 方法.....	621
12.6.2 shuffle 方法	624
12.6.3 reverse、fill、copy、max 和 min 方法.....	627
12.6.4 binarySearch 方法	629
12.6.5 addAll、frequency 和 disjoint 方法	631
12.7 PriorityQueue 类和 Queue 接口.....	632
12.8 哈希表.....	634
12.9 集合.....	636
12.10 映射.....	639
12.11 用于创建不可变集合的便捷工厂方法.....	642
12.12 并发集合.....	646
12.13 小结.....	646
专题：生成式 AI 练习.....	647
第 13 章 深入泛型类与方法	648
13.1 导论.....	648
13.2 泛型方法的动机.....	648
13.3 泛型方法：实现和编译时转换.....	651

13.3.1 泛型方法 <code>printArray</code>	651
13.3.2 泛型方法的类型参数区域.....	651
13.3.3 测试泛型方法.....	652
13.3.4 编译时类型擦除.....	653
13.4 其他编译时转换问题：方法的类型参数用作返回类型.....	654
13.5 重载泛型方法.....	658
13.6 泛型类.....	659
13.6.1 参数化类型.....	659
13.6.2 实现泛型 <code>Stack</code> 类.....	659
13.6.3 测试泛型 <code>Stack</code> 类.....	661
13.6.4 重构测试程序以使用泛型方法操作 <code>Stack<E></code> 对象.....	665
13.7 泛型方法中的通配符.....	667
13.7.1 对 <code>List<Number></code> 求和	667
13.7.2 使用通配符类型实参重新实现 <code>sum</code> 方法	669
13.8 小结.....	672
专题：生成式 AI 练习.....	673
第 14 章 使用 Lambda 表达式和流进行函数式编程	674
14.1 导论.....	674
14.2 流和归约.....	675
14.2.1 使用 <code>for</code> 循环对 1~10 的整数求和	676
14.2.2 使用 <code>for</code> 进行外部迭代容易出错	676
14.2.3 使用流和归约进行求和.....	676
14.2.4 内部迭代.....	678
14.3 映射和 Lambda 表达式.....	679
14.3.1 Lambda 表达式.....	680

14.3.2 Lambda 表达式语法.....	680
14.3.3 中间操作和终端操作.....	682
14.4 过滤.....	684
14.5 元素如何在流水线中移动.....	685
14.6 方法引用.....	687
14.6.1 创建由随机数构成的 <code>IntStream</code>	688
14.6.2 使用 <code>forEach</code> 和方法引用对每个流元素执行任务.....	688
14.6.3 使用 <code>mapToObj</code> 将整数映射为字符串对象.....	689
14.6.4 使用 <code>collect</code> 连接字符串.....	689
14.7 <code>IntStream</code> 的操作.....	690
14.7.1 创建 <code>IntStream</code> 并显示其值.....	690
14.7.2 终端操作 <code>count</code> 、 <code>min</code> 、 <code>max</code> 、 <code>sum</code> 和 <code>average</code>	691
14.7.3 终端操作 <code>reduce</code>	692
14.7.4 对 <code>IntStream</code> 中的值进行排序.....	695
14.8 函数式接口.....	695
14.9 深入 Lambda 表达式.....	697
14.10 <code>Stream<Integer></code> 的操作.....	699
14.10.1 创建 <code>Stream<Integer></code> 、排序并收集结果.....	700
14.10.2 过滤流并存储结果供后续使用.....	701
14.10.3 过滤和排序流并收集结果.....	701
14.10.4 对先前收集的结果进行排序.....	702
14.11 <code>Stream<String></code> 的操作.....	702
14.11.1 将字符串映射为大写.....	703
14.11.2 过滤字符串并进行不区分大小写的升序排序.....	703

14.11.3 过滤字符串并进行不区分大小写的降序排序	704
14.12 Stream<Employee>的操作.....	704
14.12.1 创建并显示 List<Employee>	705
14.12.2 过滤工资在指定范围内的员工	706
14.12.3 按多个字段对员工排序	709
14.12.4 将 Employee 映射到唯一的姓氏字符串	711
14.12.5 按部门对员工进行分组	712
14.12.6 统计每个部门的员工数量	713
14.12.7 计算总工资和平均工资	714
14.13 从文件创建 Stream<String>	715
14.14 随机值流.....	719
14.15 无限流.....	721
14.16 关于接口的补充说明.....	722
14.17 小结.....	723
专题：生成式 AI 练习	724
第 5 部分 JavaFX GUI、图形与多媒体	726
第 15 章 JavaFX GUI (上)	726
15.1 导论.....	726
15.2 JavaFX Scene Builder	728
15.3 JavaFX 应用程序窗口结构.....	728
15.4 “欢迎”应用程序：显示文本和图像.....	730
15.4.1 启动 Scene Builder 并创建 Welcome.fxml 文件.....	730
15.4.2 将图片文件添加到 Welcome.fxml 所在的文件夹中	732
15.4.3 创建 VBox 布局容器	732
15.4.4 配置 VBox 布局容器	732

15.4.5 添加并配置标签.....	733
15.4.6 添加和配置 ImageView.....	734
15.4.7 预览 Welcome GUI.....	735
15.5 小费计算器：事件处理入门.....	735
15.5.1 演示小费计算器.....	736
15.5.2 技术概述.....	738
15.5.3 构建应用程序的 GUI.....	741
15.5.4 TipCalculator 类.....	748
15.5.5 TipCalculatorController 类使用 Lambda 表达式实现事件处理程序.....	750
15.6 其他 JavaFX 章节中涵盖的特性.....	754
15.7 小结.....	755
专题：生成式 AI 练习.....	755
第 16 章 JavaFX GUI (下)	757
16.1 导论.....	757
16.2 在场景图中布局节点.....	757
16.3 “绘图”应用程序：单选按钮、鼠标事件和形状.....	760
16.3.1 技术概述.....	760
16.3.2 创建 Painter.fxml 文件.....	763
16.3.3 构建 GUI.....	763
16.3.4 Application 的 Painter 子类.....	766
16.3.5 PainterController 类.....	767
16.4 “颜色选择器”应用程序：属性绑定和属性侦听器.....	771
16.4.1 技术概述.....	772
16.4.2 构建 GUI.....	773

16.4.3 Application 的 ColorChooser 子类	775
16.4.4 ColorChooserController 类	776
16.5 “封面查看器”应用程序：使用 JavaFX 集合的数据驱动 GUI	779
16.5.1 技术概述.....	779
16.5.2 构建 GUI.....	780
16.5.3 Application 的 CoverViewer 子类	781
16.5.4 CoverViewerController 类.....	781
16.6 “封面查看器”应用程序：自定义 ListView 单元格.....	784
16.6.1 技术概述.....	785
16.6.2 复制 CoverViewer 应用程序.....	785
16.6.3 自定义 ListView 单元格布局	785
16.6.4 CoverViewerController 类.....	787
16.7 FileChooser 和 DirectoryChooser 对话框.....	788
16.8 其他 JavaFX 功能和 JavaFX 可访问性.....	795
16.9 JavaFX 更新.....	796
16.10 JavaFX 资源和库.....	798
16.11 小结.....	800
专题：生成式 AI 练习	800
第 17 章 JavaFX 图形与多媒体.....	802
17.1 导论.....	802
17.2 使用 CSS 控制字体.....	803
17.2.1 用 CSS 设置 GUI 样式.....	803
17.2.2 XML 标记简介	807
17.2.3 从 FXML 中引用 CSS 文件.....	810
17.2.4 指定 VBox 的样式类.....	810

17.2.5 以编程方式加载 CSS.....	811
17.3 显示二维形状.....	812
17.3.1 使用 FXML 定义二维形状.....	812
17.3.2 用 CSS 设置二维形状样式.....	815
17.4 折线、多边形和路径.....	819
17.4.1 GUI 和 CSS.....	819
17.4.2 PolyShapesController 类.....	820
17.5 变换.....	824
17.6 使用 Media、MediaPlayer 和 MediaView 播放视频.....	826
17.6.1 “视频播放器” GUI.....	828
17.6.2 VideoPlayerController 类.....	829
17.7 过渡动画.....	834
17.7.1 TransitionAnimations.fxml	834
17.7.2 TransitionAnimationsController 类.....	836
17.8 时间轴动画.....	840
17.9 使用 AnimationTimer 定义逐帧动画.....	843
17.10 CSS 过渡.....	846
17.11 在画布上绘图.....	850
17.12 三维形状.....	856
17.13 FXGL：使用 JavaFX 进行游戏编程.....	861
17.14 小结.....	862
专题：生成式 AI 练习.....	863
第 6 部分：高级主题.....	864
第 18 章 并发：从平台线程到虚拟线程	864

18.1 导论.....	864
18.2 使用日期/时间API 进行sort/parallelSort 计时.....	867
18.3 顺序流与并行流.....	870
18.4 使用Executor 框架创建和执行平台线程.....	874
18.5 Project Loom 概述.....	879
18.6 使用Executor 框架创建和执行虚拟线程.....	880
18.6.1 使用 Thread-Per-Task Executor 启动虚拟线程	880
18.6.2 启动虚拟线程的其他方法.....	883
18.7 探查平台线程与虚拟线程的性能.....	885
18.7.1 Future 和 Callable 接口	886
18.7.2 我们使用的线程处理方法.....	886
18.7.3 启动大量并发任务并计时	887
18.7.4 测试不同的线程处理方法.....	888
18.7.5 性能分析.....	889
18.8 结构化并发和作用域值.....	892
18.9 线程同步概述.....	902
18.10 使用ArrayBlockingQueue 实现生产者/消费者关系.....	903
18.10.1 同步和状态依赖性.....	904
18.10.2 ArrayBlockingQueue 类.....	904
18.10.3 生产者/消费者示例概述	905
18.10.4 Buffer 接口	905
18.10.5 BlockingBuffer 类.....	906
18.10.6 BlockingBufferTest 类.....	907
18.10.7 有界缓冲区.....	910
18.10.8 其他并发集合.....	911

18.11 JavaFX 中的多线程.....	912
18.11.1 在工作线程中执行计算：斐波那契数.....	913
18.11.2 处理中间结果：埃拉托色尼筛法.....	919
18.12 小结.....	926
专题：生成式 AI 练习.....	927
第 19 章 构建基于 API 的 Java 生成式 AI 应用.....	930
19.1 导论.....	930
19.2 OpenAI API.....	932
19.2.1 OpenAI 的部分模型.....	932
19.2.2 OpenAI API 功能.....	933
19.2.3 获取 OpenAI 开发者帐户.....	933
19.2.4 OpenAI 开发者 API 密钥.....	933
19.3 设置 Java 环境.....	934
19.3.1 Simple-OpenAI 开源库：从 Java 连接到 OpenAI.....	935
19.3.2 IntelliJ 基于 Maven 的 Java 项目.....	935
19.4 通过聊天补全生成文本.....	936
19.4.1 文本摘要.....	936
19.4.2 情感分析.....	940
19.4.3 无障碍图像描述.....	941
19.4.4 语言检测和翻译.....	944
19.4.5 代码生成.....	946
19.4.6 命名实体识别(NER)和结构化输出.....	950
19.5 语音合成和语音识别.....	953
19.5.2 文本转语音.....	954

19.6 图像生成.....	956
19.7 视频.....	959
19.7.1 生成视频隐藏式字幕.....	959
19.7.2 Sora	962
19.7.3 其他视频 AI	963
19.8 审核.....	964
19.9 <i>OpenAIUtilities</i> 类.....	966
19.9.1 建造者设计模式.....	967
19.9.2 package 和 import 语句	967
19.9.3 嵌套 record 类 Message	968
19.9.4 SimpleOpenAI 客户端对象	969
19.9.5 chat 方法.....	969
19.9.6 辅助方法 makeChatMessages.....	971
19.9.7 describeImage 方法	971
19.9.8 translate 方法.....	973
19.9.9 chatWithStructuredOutput 方法	973
19.9.10 speechToText 方法	976
19.9.11 textToSpeech 方法	977
19.9.12 image 方法.....	978
19.9.13 speechToVTT 方法	980
19.9.14 checkPrompt 方法	981
19.10 小结.....	982
专题: 生成式 AI 练习.....	983
AI 开发项目	984
第 20 章 使用 JDBC 和 SQLite 访问数据库.....	999

20.1 导论.....	999
20.2 关系数据库.....	1000
20.3 设置 SQLite RDBMS.....	1001
20.4 books 数据库.....	1002
20.4.1 SELECT 查询	1007
20.4.2 WHERE 子句.....	1007
20.4.3 ORDER BY 子句	1009
20.4.5 INSERT INTO 语句	1011
20.4.6 UPDATE 语句	1012
20.4.7 DELETE FROM 语句	1013
20.5 使用 JDBC 连接和查询数据库.....	1014
20.5.1 驱动程序自动发现.....	1015
20.5.2 连接到数据库.....	1016
20.5.3 创建用于执行查询的语句	1017
20.5.4 执行查询.....	1017
20.5.5 处理查询的 ResultSet	1017
20.6 查询 books 数据库.....	1018
20.6.1 构建 GUI.....	1019
20.6.2 DisplayQueryResultsController 类.....	1021
20.7 PreparedStatement.....	1026
20.7.1 使用 PreparedStatement 的通讯录应用.....	1027
20.7.2 Person 记录类	1028
20.7.3 PersonQueries 类.....	1028
20.7.5 AddressBookController 类.....	1033

20.8 存储过程.....	1038
20.9 事务处理.....	1039
20.10 小结.....	1039
专题: 生成式AI 练习.....	1040
第 21 章 Java 平台模块系统(JPMS).....	1042
21.1 导论.....	1042
21.2 模块声明.....	1047
21.2.1 requires	1048
21.2.2 requires transitive: 隐式可读性	1048
21.2.3 exports 和 exports...to.....	1049
21.2.4 uses.....	1049
21.2.5 provides...with	1049
21.2.6 open、opens 和 opens...to	1049
21.2.7 受限关键字.....	1051
21.3 模块化的Welcome 应用程序.....	1051
21.3.1 Welcome 应用程序的结构	1052
21.3.2 Welcome 类.....	1055
21.3.3 module-info.java	1056
21.3.4 模块依赖关系图.....	1056
21.3.5 编译模块.....	1056
21.3.6 从模块的分解文件夹运行应用程序	1059
21.3.7 将模块打包到模块化 JAR 文件中	1059
21.3.8 从模块化 JAR 文件运行 Welcome 应用程序.....	1060
21.3.9 延伸阅读: 类路径与模块路径.....	1061
21.4 创建和使用自定义模块.....	1061

21.4.1 导出包以便在其他模块中使用	1062
21.4.2 使用另一个模块中包里的类	1063
21.4.3 编译和运行示例	1065
21.4.4 将应用程序打包到模块化 JAR 文件中	1066
21.4.5 强封装和可访问性	1066
21.5 深入模块依赖关系图	1067
21.5.1 java.sql	1067
21.5.2 java.se	1068
21.5.3 错误：带有循环的模块图	1070
21.6 将代码迁移到模块	1071
21.6.1 未命名模块	1072
21.6.2 自动模块	1072
21.6.3 使用 jdeps 工具分析 Java 依赖性	1073
21.7 模块中的资源	1076
21.7.1 依赖多个模块	1078
21.7.2 为反射而开放模块	1078
21.7.3 编译模块	1078
21.7.4 运行模块化应用	1079
21.7.5 模块依赖关系图	1079
21.8 使用 jlink 创建自定义运行时	1080
21.8.1 仅包含 java.base 的自定义运行时	1081
21.8.2 为 Welcome 应用创建自定义运行时	1082
21.8.3 使用自定义运行时执行 Welcome 应用程序	1083
21.8.4 在自定义运行时上使用模块解析器	1083

21.9 服务和 <code>ServiceLoader</code>	1084
21.9.1 服务提供者接口	1086
21.9.2 加载和使用服务提供程序	1087
21.9.3 使用模块指令和服务消费者	1090
21.9.4 运行没有服务提供者的应用程序	1091
21.9.5 实现服务提供者	1091
21.9.6 <code>provides...with</code> 模块指令和声明服务提供者	1092
21.9.7 运行带有一个服务提供者的应用程序	1093
21.9.8 实现第二个服务提供者	1093
21.9.9 使用两个服务提供者运行应用程序	1095
21.10 小结	1096
专题: 生成式 AI 练习	1096
第 22 章 递归和 Big O 表示法	1097
22.1 导论	1098
22.2 递归概念	1098
22.3 递归示例: 阶乘	1098
22.4 递归示例: 斐波那契数列	1101
22.5 递归与迭代	1105
22.6 汉诺塔	1107
22.7 分形	1110
22.7.1 科赫曲线分形	1110
22.7.2 (选读)案例学习: Lo 羽毛分形	1111
22.7.3 (选读)Fractal 应用程序的 GUI	1113
22.7.4 (选读)FractalController 类	1115
22.8 递归回溯	1121

22.9 Big O 表示法.....	1122
22.9.1 O(1)算法.....	1122
22.9.2 O(n)算法	1122
22.9.3 O(n ²)算法	1123
22.10 常见的 Big O 表示法.....	1124
22.11 小结.....	1124
附录 A JShell 交互式 Java 入门.....	1126
A.1 导论.....	1126
A.2 JShell 简介.....	1128
A.2.1 启动 JShell 会话	1128
A.2.2 执行语句	1129
A.2.3 显式声明变量	1130
A.2.4 列出并执行之前的代码片段.....	1133
A.2.5 表达式求值和变量隐式声明.....	1134
A.2.6 使用隐式声明的变量	1135
A.2.7 查看变量的值	1136
A.2.8 重置 JShell 会话	1136
A.2.9 编写多行语句	1136
A.2.10 编辑代码片段	1137
A.2.11 退出 JShell	1141
A.3 JShell 中的命令行输入.....	1141
A.4 声明和使用类.....	1142
A.4.1 在 JShell 中创建类	1142
A.4.2 显式声明引用类型的变量.....	1143

A.4.3 创建对象	1143
A.4.4 操作对象	1144
A.4.5 为表达式创建一个有意义的变量名	1145
A.4.6 保存和打开代码段文件	1146
A.5 使用 JShell 自动补全	1147
A.5.1 自动补全标识符	1147
A.5.2 自动补全 JShell 命令	1148
A.6 探索类的成员和查看文档	1149
A.6.1 列出 Math 类的静态成员	1149
A.6.2 查看方法的参数	1150
A.6.3 查看方法的文档	1151
A.6.4 查看 public 字段的文档	1151
A.6.5 查看类的文档	1152
A.6.6 查看方法的重载版本	1152
A.6.7 探索特定对象的成员	1153
A.7 声明方法	1155
A.7.1 前向引用未声明的方法——声明 displayCubes 方法	1155
A.7.2 声明一个之前未声明的方法	1156
A.7.3 测试 cube 并替换它的声明	1156
A.7.4 测试更新后的 cube 方法和 displayCubes 方法	1157
A.8 异常	1158
A.9 导入类和将包添加到类路径	1158
A.10 使用外部编辑器	1162
A.11 JShell 命令摘要	1162
A.11.1 在 JShell 中获取帮助	1164

A.11.2 /edit 命令: 附加功能	1164
A.11.3 /reload 命令	1165
A.11.4 /drop 命令	1166
A.11.5 反馈模式	1166
A.11.6 其他可用/set 配置的 JShell 功能	1168
A.12 代码片段编辑快捷键	1169
A.13 JShell 如何为交互式使用重新解释 Java	1170
A.14 IDE 的 JShell 支持	1170
A.15 小结	1170
自测题(附答案)	1171
附录 B 格式化输出	1189
B.1 导论	1189
B.2 使用 printf 进行输出	1189
B.3 整数格式化	1190
B.4 浮点数格式化	1191
B.5 字符串和字符格式化	1193
B.6 其他转换字符	1193
B.7 字段宽度与精度	1195
B.8 格式化标志	1197
B.9 格式字符串中用于显式定位的参数索引	1201
B.10 转义序列	1201
B.11 在内存中格式化字符串	1202
B.12 小结	1203
专题: 生成式 AI 练习	1204

附录 C 数制系统	1205
C.1 导论.....	1205
C.2 将二进制数缩写为八进制数和十六进制数.....	1210
C.3 八进制数和十六进制数转换为二进制数.....	1211
C.4 从二进制、八进制或十六进制转换为十进制.....	1211
C.5 从十进制转换为二进制、八进制或十六进制.....	1212
C.6 负二进制数：补码表示法.....	1214

译者序：AI 赋能的 Java 学习之道

与戴特尔父子结缘，始于其 2003 年版的《Python 编程金典》。犹记当年，我与他们通过电子邮件，就书中诸多问题进行了深入而坦诚的沟通。那时我便已深为其写作风格所折服，他们真正做到了手把手引领初学者步入编程之门。

当然，彼时的作品也并非完美，因其面向新手，行文稍显繁复。我当时便将主要精力投入语言的再组织，力求行文之凝练。

二十余载光阴流转，今朝幸得清华大学出版社之邀，翻译戴特尔最新版 Java 教材，心中既有志忑，亦有期待，只因新书围绕 AI 这一前沿主题展开。待到样书在手，所有疑虑烟消云散。戴特尔的书如今已全面革新其叙述风格，不仅语言精炼，更经过了极为全面的技术审查。纵观全书，除却些许微不足道的笔误，我几乎未发现任何技术谬误。更重要的是，它已然成为一部革命性的教科书，彻底改变了一门主流编程语言的学习范式。

传统学习，我们按图索骥，沿书中路径由浅入深，依作者指引完成练习。而今，除却这些基本流程，AI 辅助学习的引入，使得传授知识的不再仅仅是作者与书本，更有无数 AI 导师随时待命。书中任何未解之惑，心中些许未明之疑，皆可随时唤出 AI 探寻究竟。AI 不仅能答疑解惑，更能积极扩展，助您深彻理解某一主题。

书本上的内容有穷尽之日，而与 AI 的对话却无止境。您可以沿一缕线索深挖不辍，直至对一个主题了然于心。AI 亦可助您编写代码。面对任何可用 Java 程序解决的问题，都可以先行思索，构想自己的实现，再让 AI 挥毫一番。两相对照，高下立判，自己的差距（或优势）一目了然。例如，诚如第 18 章所述，以朴素递归法计算斐波那契数，其代价之高昂人尽皆知，故而至少应以迭代法代之。然则，尚有更迅捷之算法否？垂问 AI，它便会贴心地告知您矩阵快速幂法与快速倍增法，甚至会为您奉上具体的 Java 代码实现。凡此种种，皆为书中未述之进阶知识，却与斐波那契算法紧密相连。在同一个知识点上进行这样的深度挖掘，正是构建“知识树”并实现高效学习的关键所在。

为了强调这一点，本书中文版的配套资源 (<https://bookzhou.com>) 中，便在 `ch22/genAIexercises` 目录下提供了一个名为“快速斐波那契.java”的程序。该程序由 DeepSeek 根据以下提示生成，能够瞬间算出第 20 万个斐波那契数：“写一个 Java 程序，用最快的算法计算斐波那契数，要求在合理时间内，计算最多第 20 万个斐波那契数。用户输入一个值 n ，程序计算第 n 个斐波那契数。第 0 个是 0，第 1 个是 1，第 2 个是 1，第 3 个是 2，以此类推”。

不仅如此，译者还创建了一个算法演示网站 (<https://fm2gp-chinese.bookzhou.com>)，专门用于直观对比不同时间复杂度的斐波那契数列算法，此举正是为了再次强调在某个兴趣点上深度钻研的重要性。

本书的组织结构久经考验。二十多年前，戴特尔便从《Python 编程金典》一书开始，不断探索教授编程语言的最佳方式。如今，这套方法已经成熟，形成了“对象自然”（Objects-Natural）与“实时代码”（Live-Code）两大核心教学理念。前者指在学习如何自定义类之前，先直接复用现有的类，避免一开始就陷入细节的泥潭；后者指所有代码示例都可以立即运行，是“鲜活”的代码，而非“死板”的陈列。关于这两大教学法的具体阐述，请参见本书的“前言”。

建议您先阅读“前言”，再根据后续“准备工作”部分的指引搭建好自己的 Java 开发环境。然后，通过第 1 章快速了解 Java 语言并亲手运行您的第一个 Java 程序。最后，从第 2 章开始，就可以正式进入系统的学习了。

第 19 章或许是本书的灵魂所在，它完全围绕调用 API 来开发 AI 应用展开。这些程序可实现文本摘要、情感分析、无障碍图像描述、语言检测与翻译、Java 代码生成、命名实体识别与结构化输出、语音转文本、文本转语音、图像生成、为视频创建隐藏式字幕以及内容审核等诸多尖端功能。在章末的练习中，您甚至会得到用 AI 开发 Java 游戏的指导。这一切都妙趣横生，且与当今技术潮流完全契合。

英文版既已如此出众，中文版自当精益求精。本书中文版具有以下特色：

- 全部代码均已中文化，包括屏幕/GUI 界面和代码注释；
- 不易访问的国外资源已替换为国内对应资源；
- 不易访问的国外 AI 已替换为国内 AI，后者在某些方面甚至更具优势；
- 全书共三百余条“译注”，为您提供补充说明，助您规避诸多“陷阱”；
- 新增了一些“生成式 AI 练习”；
- 本书中文版主页（<https://bookzhou.com>）将与英文版主页（<https://deitel.com/javafp5>）同步更新，并提供独具特色的内容，例如如何顺畅地调用 OpenAI API；
- 本书中文版 GitHub 项目（<https://github.com/transbot/JavaForProgrammers5e-Chinese>）提供了全书源代码和补充资料，并将随时保持更新。

翻译本书的过程中，我不仅学到了许多新的知识（特别是 Java 23 和 Java 24 的新特性），也对 Java 编程有了更深入的理解。作者的语言简洁明了，逻辑严谨，每一个概念都讲解得非常透彻。此外，本书的实用性也给我留下了深刻的印象。作者不仅讲授理论知识，更注重实际应用，通过大量的案例和练习，帮助读者将所学内容应用于实际项目中。这种教学方法不仅提高了学习效果，也让读者在实践中不断成长。

作为一部经典的 Java 教材，本书不仅教会了我们如何编写代码，更让我们明白了编程的本质与意义。无论您是刚刚接触 Java 的新手，还是经验丰富的开发者，都能从本书中获得新的启发和收获。希望这本书能够成为广大读者编程生涯中的良师益友，帮助大家在这个世界里不断探索，不断进步。

编程之路，道阻且长，但只要心中有光，便能一路前行。愿我们每一位读者都能借助于这

本书的帮助，找到属于自己的编程之道。

最后，请允许我以本书“献辞”作为结语：

谨以此书，追思斯坦福先贤约翰·麦卡锡。

公首创“人工智能”之嘉名，更燃思想之炬，光照后人前行之路。

昔日畅想，今已成与我辈共事之智者。愿此书所授，能承公之遗泽，续写华章，是为纪念。

——周靖，2025年，北京

前言

今天，我们正处于计算领域重大转变的风口浪尖。AI 和加速计算的交汇将重新定义未来。

——黄仁勋，英伟达 CEO^①

欢迎阅读《AI 高效应用：Java 编程实践（第 5 版）》。在前言中，我们将揭示本书的“灵魂”——提升效能。本版通过三种前沿互补的教学方法，助您以更快的速度、更广的维度、更深的层次来掌握 Java 编程。相较于传统学习方式，您将能完成更具挑战性的任务，并为职业生涯中即将面临的编程与软件开发挑战做好充分准备：

三大核心教学法，层层突破编程能力：

- **「代码教代码」——夯实基础的沉浸式学习**：我们沿用经典的“实时编码”（Live-Code）教学法。通过沉浸于数百个带实际输出的完整可运行代码示例，您将在掌握 Java 基础知识的同时，充分释放学习效能。每一行代码都直击应用场景，让抽象概念在具体实现中自然落地。
- **「对象教对象」——面向对象编程的革新实践（本版新增）**：全新的“对象自然”教学法将引领您通过 Java 庞大类库及 GitHub 等开源仓库中的现成功能类，学习面向对象编程。无需从零构建，直接调用成熟类库完成复杂任务，在实践中理解封装、继承与多态的本质，快速提升编写高价值程序的能力。
- **「生成式 AI 教生成式 AI」——掌握未来编程的核心引擎（本版新增）**：作为本版革命性内容，您将系统学习生成式 AI 这一提升创造力的关键技术。通过示例与练习，您将学会运用提示词生成原创文本、Java 代码、图像、音频、语音、音乐乃至视频。最终，你不仅会成为 AI 的使用者，更会成为智能程序的构建者。

合著者哈维·戴特尔的个人轶事

20 世纪 70 年代，一位朋友建议我在当地大学教授计算机导论课程。在讲解编程语言类型层次（从低级机器语言到中级汇编语言再到高级语言）时，我曾亲自演示过每类语言的编程。我向学生解释：越往层次结构的上层走，编程效能越高——您将远离原始硬件细节，贴近应用领域，以实际问题为导向思考。这些语言的单个语句在层级中每上升一层，就更强大一分：它们不仅能提升您快速解决复杂问题的能力，更能产出设计更精良的解决方案。

^① 黄仁勋，英伟达 COMPUTEX 2024 主题演讲，2024 年 6 月 2 日。原文是 “Today, we’re at the cusp of a major shift in computing. The intersection of AI and accelerated computing is set to redefine the future.” 官方中文视频：<https://www.bilibili.com/video/BV1mm421N7iN>。

这种“层级跃升带来效能革命”的理念，正是本书的核心脉络——从代码到对象，再到生成式 AI，每一次迭代都在重塑编程的边界，助您在技术浪潮中始终占据先机。

学生们充满好奇，追问这一切将走向何方。我当时答道：“可以合理预测，未来会出现更高级别的编程技术，帮助人们更快、更好地解决更具挑战性的现实问题。随着你们编写的编程语句愈发强大，终有一天，编程将高度贴近应用领域——届时只需告诉计算机需求，而无需提供日益复杂的构建细节，计算机便会为你生成所需软件！”

那是 20 世纪 70 年代的场景。随后在八九十年代，面向对象编程——Java 的核心编程模型——横空出世。对象源自被称为“类”的“蓝图”，几乎任何名词或概念都能用类来表示。真实世界中这类事物不胜枚举，程序员开始构建庞大的类库，进一步提升了软件开发效率。20 世纪 90 年代中期，早期 Java 版本问世时，其免费特性与数百个预构建的强大类库尤为引人注目，这使程序员能够在图形、GUI、并发、网络、数据结构、数据库等关键领域快速开发重要应用。这些类中许多包含数百甚至数千行代码，但程序员创建类的对象通常只需编写一行简洁代码，再用几句简短语句即可让对象“大显身手”。这标志着一种更接近问题领域的高级编程能力。

与此同时，人工智能领域自 20 世纪 50 年代诞生以来持续快速演进。然而，直到谷歌 2017 年发表“注意力就是你所需要的一切”（*Attention is All You Need*）研究论文^①、定义 Transformer 模型，当前的 AI 热潮才真正兴起。这一模型正是当今大语言模型的根基，为本书中将要使用的生成式 AI 聊天机器人和工具提供强大支撑。

2022 年 11 月，OpenAI 发布 ChatGPT，五天内用户突破百万，两个月内达一亿。从未有其他应用能如此迅速普及，同时深刻提升用户生产力，并极大增强人们让计算机完成真正有趣且强大任务的能力。

我们在全书中深度整合了生成式 AI。对 Java 开发者而言，重中之重在于，生成式 AI 能够出色地编写、文档化、解释、调试、纠正、优化 Java 代码并提升其性能。生成式 AI 并非完美无缺——它们会犯错，有时还会产生“幻觉”——因此必须仔细监控。我们将在全书约 600 个生成式 AI 练习和示例中展示这些编码能力。生成式 AI 创造了一种超高层次的编程能力，这将极大地提升你的 Java 学习体验，并使你能够快速、便捷、经济地开发出健壮、高质量的 Java 软件。^②

^① 译注：论文原版请参见 <https://arxiv.org/abs/1706.03762>。

^② 译注：现在提到“AI”时，一般都是指“生成式 AI”，而不是早几年的“预测式 AI”。为了保持严谨，本书大多数时候都保留了“生成式 AI”的说法。

丰富多彩的真实世界 Java 应用

本书为读者提供了友好、现代、代码密集且面向案例的 Java 入门教程。作为全球最流行的编程语言之一^{①②}，Java 因其平台独立性、健壮性及全面的标准库与第三方类库支持而在众多应用领域备受青睐。从最小的物联网设备（例如全球联网的传感器）到最大的基于云的企业计算平台（例如华为云），Java 被用于构建各种软件。以下是一些有趣的 Java 应用案例：^③

- 《我的世界》(*Minecraft*)——有史以来最受欢迎的电子游戏之一^④；
- 金融交易平台（例如 LSEG Real-Time SDK 7^⑤）；
- 美国联邦航空管理局（FAA）航班与机场信息 API^⑥；
- Android 移动应用开发^⑦——Android 作为全球领先的移动设备操作系统，占据 73.5% 的全球市场份额；^⑧
- 美国国家航空航天局（NASA）火星探测器软件；^⑨
- 基因 DNA 和 RNA 测序数据分析工具。^⑩

本书用到的流行生成式 AI 聊天机器人（DeepSeek、ChatGPT、Gemini、Claude 和

^① PYPL 编程语言流行度指数，<https://pypl.github.io/PYPL.html>。

^② Tiobe 指数，<https://www.tiobe.com/tiobe-index>。

^③ 以下内容是 ChatGPT 对“列出 Java 的有趣用途，例如火星探测器”的回复。我们进行了一定的编辑。

^④ 维基百科，<https://zh.wikipedia.org/wiki/Minecraft>。

^⑤ LSEG 开发者社区，“Real-Time SDK - Java”。<https://developers.lseg.com/en/api-catalog/real-time-opnsrc/rt-sdk-java>。

^⑥ 美国联邦航空管理局（FAA）的 GitHub 仓库：<https://github.com/Federal-Aviation-Administration>。

^⑦ Android 开发者社区，<https://developer.android.com>。

^⑧ “Mobile Operating System Market Share Worldwide”，StatCounter，<https://gs.statcounter.com/os-market-share/mobile/worldwide>。

^⑨ “Java runs remote-controlled Mars rover”，作者 Philip Ross，CNET。<https://www.cnet.com/tech/tech-industry/java-runs-remote-controlled-mars-rover>。

^⑩ “Getting Started with GATK4”，Broad 研究所的 GATK 文档。<https://gatk.broadinstitute.org/hc/en-us/articles/360036194592-Getting-started-with-GATK4>。

Perplexity) 还列出了其他 Java 应用场景：^①

- 桌面 GUI 应用程序（我们将用第 15 章～第 17 章介绍的 JavaFX 开发这种应用程序）；
- 跨平台 2D 和 3D 游戏开发库（例如我们将在一篇博客文章中介绍的 FXGL）；
- 嵌入式系统（例如智能卡、蓝光播放器、有线电视机顶盒、打印机及汽车“信息娱乐”系统）；
- 科学与教育软件；
- 基于 Web 的应用程序（例如 Spring 和 Spring Boot 框架）；
- 高性能服务器端软件；
- 银行与金融领域的企业级解决方案；
- 大规模分布式系统；
- 机器人技术；
- 大数据技术（例如用于存储和处理海量数据的 Hadoop）。

本书面向的读者

本书采用模块化架构（详见稍后的“高级概览”一节），适用于多个读者群体：

- 参加初级和中级职业 Java 编程培训课程的读者，课程内容设计基于 ACM（美国计算机协会）和 IEEE（电气电子工程师学会）这两个权威专业机构所推荐的课程大纲；^②
- 熟悉其他编程语言、需要为新项目学习 Java 的开发人员；
- 以及已掌握部分 Java 知识、需要学习最新 Java 特性与编程惯用法以应对新项目的程序员。

此外，本书还重写了 Oracle Java SE（标准版）开发人员认证考试的核心备考主题。^{③④}

^① 这些是各种 AI 对“简单地列出 Java 语言的主要应用场景”的回复，我们进行了一定的编辑。

^② ACM 和 IEEE 联合发布的《计算机科学课程指南 2023》（*Computer Science Curricula 2023*，简称 CS2023）。这是一份权威的计算机科学教育标准文件，旨在为全球高校的计算机专业课程设置提供指导。

^③ 珍妮·博亚尔斯基（Jeanne Boyarsky）与斯科特·塞利科夫（Scott Selikoff）合著的 *OCP Oracle Certified Professional Java SE 21 Developer Study Guide* 一书，Wiley 出版社，2024 年。（珍妮·博亚尔斯基也是本书的技术审校之一。）

^④ “Exam: Java SE 21 Developer Professional (1Z0-830)”，<https://mylearn.oracle.com/ou/exam/java-se-21-developer-professional-1z0-830/40805/139080/220555>。

“实时编码”教学方法

本书的核心是 Deitel 特色的**实时编码**（Live-Code）教学方法。与仅提供代码片段的方式不同，你将通过 200 多个完整编写的真实世界示例和案例学习来动手研习 Java，并附有来自计算机科学、数据科学、人工智能等领域的实时输出结果。请阅读“前言”之后的“准备工作”，了解如何下载这些示例代码，并设置自己的 Windows 或 macOS 计算机以运行它们。

第 1 章的 1.7 节介绍了如何使用免费、开源的 OpenJDK 来编译和运行这些代码示例。在阅读本书的同时，请积极地执行每个程序，这会使你的 Java 学习体验更加生动真实。我们还 [在 https://deitel.com/javafp5](https://deitel.com/javafp5) 提供了以下流行的 Java 开发工具的快速入门指南：

- JetBrains IntelliJ IDEA 社区版
- Eclipse IDE for Java Developers
- Microsoft Visual Studio Code

高级概览

编程范式

过程式（第 2 章～第 7 章）、面向对象（第 8 章～第 10 章）、泛型（第 12 章和第 13 章）、函数式（第 14 章）、并发（第 18 章）及由生成式 AI 增强（第 19 章）。

第 1 部分：导论

第 1 章：概述、Java 应用程序实操演练与生成式 AI——互联网与万维网；云技术；物联网（IoT）；元宇宙；AI/生成式 AI 简介；JDK 快速入门；27 个生成式 AI 练习。

第 2 章：Java 编程入门——Java 基础知识；输入/输出；基元类型；算术运算；判断（决策）语句；“对象自然”案例学习：String 类；12 个生成式 AI 练习。

第 2 部分：编程基础

第 3 章：控制语句（上）——if、if/else 和 while 语句；计数器控制/哨兵控制的迭代；嵌套控制语句；“对象自然”案例学习：用于超大整数的 BigInteger；16 个生成式 AI 练习。

第 4 章：控制语句（下）——for、do/while、switch、break 和 continue 语句；逻辑操作符；“对象自然”案例学习：用于精确货币计算的 BigDecimal；13 个生成式 AI 提示。

第 5 章：方法——自定义方法；随机数；模拟；“对象自然”案例学习：日期/时间 API；

16 个生成式 AI 练习。

第 6 章：数组与 ArrayList——数据结构；内置数组；“对象自然”案例学习：用 ArrayList 实现可动态调整大小的数组；28 个生成式 AI 练习。

第 7 章：字符串、NLP 与正则表达式：生成式 AI 基础——String、Character 和 StringBuilder 类；自然语言处理（NLP）简介；两个“对象自然”案例学习：一是使用正则表达式进行文本模式匹配，二是使用 AES 私钥加密保护数据；22 个生成式 AI 练习。

第 3 部分：面向对象编程

第 8 章：使用自定义类进行真实世界建模——创建有价值的类；Account 类案例学习；洗牌和发牌模拟；Time 类案例学习；控制类成员的访问权限；构造函数；赋值和取值方法；数据校验；抛出异常；static 和 final 类成员；record 类；33 个生成式 AI 练习。

第 9 章：使用继承、多态性和接口进行真实世界建模——继承层次结构；运行时多态性；is-a（继承）关系与 has-a（组合）关系；接口；针对接口编程，而非针对实现编程；依赖注入；sealed 类和接口；31 个生成式 AI 练习。

第 10 章：深入异常处理——用 try 语句捕获并处理异常；Throwable 层次结构；受检（checked）与非受检（unchecked）异常^①；try-with-resources 语句^②；30 个生成式 AI 练习。

第 11 章：文件、I/O 流、JSON 序列化与 CSV 文件——数据持久化；文本文件与二进制文件；NIO 类；获取文件/目录信息；Formatter 类；JSON 序列化；使用 java.net.http 特性^③调用 OpenWeatherMap Web 服务；CSV 文件格式；泰坦尼克号灾难 CSV 数据集与基础数据分析；“对象自然”案例学习：使用 RSA 公钥密码学保障数据安全并保护用户隐私；38 个生成式 AI 练习。

第 4 部分：数据结构、泛型集合、Lambda 表达式与流

第 12 章：泛型集合——来自 Java 集合框架（Java Collections Framework）的预打包数据结

^① 译注：受检异常是编译器强制检查的异常，必须被捕获或者在方法声明中添加 throws 子句。相反，非受检异常可自由处理。关于异常分类的核心在于“责任归属”：受检异常强制调用方处理可能的错误状态，属于方法契约的一部分；非受检异常则是程序运行时无法预料的故障。有趣的是，所有异常在编译时都会被编译器“看到”，区别在于编译器对受检异常会强制验证处理逻辑，对非受检异常则放任自流。

^② 译注：自 Java 7 引入的异常处理机制。在 try 后面的括号内声明多个资源，用分号隔开。它的优点在于，在 try 块执行完毕后，无论是否发生异常，资源都会被自动关闭。

^③ 译注：自 Java 11 引入。

构；预建的泛型数据结构；30 个生成式 AI 练习。

第 13 章：深入泛型类与方法——实现自定义泛型方法和自定义泛型类；编译时类型安全；24 个生成式 AI 练习。

第 14 章：使用 Lambda 表达式和流进行函数式编程——使用 Lambda 表达式和流管道更快、更简单地编写某些类型的程序，同时有望减少 bug 数量；不可变性；第 18 章会讲解如何在多核架构上通过并行化流管道来提升性能，37 个生成式 AI 练习。（可以接着学习第 22 章）。

第 5 部分：JavaFX 图形用户界面、图形与多媒体

第 15 章：JavaFX GUI (上)——用 Scene Builder 进行简单的拖放式 GUI 设计；布局；控件；事件处理；“欢迎”程序；“小费计算器”程序，13 个生成式 AI 练习。

第 16 章：JavaFX GUI (下)——其他布局和控件；鼠标、RadioButton 和属性变化事件；数据绑定；“绘图”程序；“封面查看器”程序；自定义控件外观；FileChooser 和 DirectoryChooser 对话框；21 个生成式 AI 练习。

第 17 章：JavaFX 图形与多媒体——用于自定义 JavaFX 节点外观和文本字体的层叠样式表 (CSS)；2D 和 3D 形状；移动、旋转和缩放节点变换 (node transformation)；显示、播放和暂停视频；使用 Transition 和 Timeline 动画逐渐更改属性；使用 AnimationTimer 实现逐帧动画；使用 CSS 过渡 (CSS transition) 简单地实现动画效果；用 Canvas 进行鼠标绘图；31 个生成式 AI 练习。

第 6 部分：高级主题

第 18 章：并发：从平台线程到虚拟线程——多核编程；创建和管理多个任务；性能案例学习；使用 Java 日期/时间 API 探查顺序与并行排序性能；预包装的并行算法；经典的生产者/消费者关系；更容易使用、更不容易出错、更高级的并发特性；Loom 项目：轻量级虚拟线程、结构化并发、作用域值 (scoped value)^①；51 个生成式 AI 练习。

第 19 章：构建基于 API 的 Java 生成式 AI 应用程序——能理解文本、代码、图像、语音和视频输入，并生成文本、代码、图像、语音和视频的多模态生成式 AI；使用 OpenAI API 创建能够总结文本、分析文本情感、为图像创建可访问描述、检测文本语言并将文本翻译成其他语言、生成 Java 代码、对文本执行命名实体识别并获得结构化 JSON 输出、将语音转录为文本、从文本合成为语音、为视频生成字幕等的应用程序；21 个生成式 AI 练习和 73 个基于 API 的编程项目。

^① 译注：也称为“范围值”。

第 20 章：使用 JDBC 和 SQLite 访问数据库——在数据库中存储持久数据；SQLite 数据库管理系统；SQL 的 CRUD（创建、读取、更新、删除，简称“增删查改”）操作；JDBC API；连接到数据库；从数据库中检索数据；JavaFX TableView；JDBC PreparedStatement^①；19 个生成式 AI 练习。

第 21 章：Java 平台模块系统(JPMS)——创建自定义包和模块；声明模块依赖项；指定模块导出哪些包；定义模块提供的或使用的服务；反射和启用反射访问；将模块与资源捆绑；31 个生成式 AI 练习。

第 22 章：递归与 Big O 表示法——递归和迭代版的阶乘计算；递归斐波那契计算；递归创建“羽毛分形”^②的 JavaFX 应用程序；基于文本的算法可视化；丰富的递归练习；11 个生成式 AI 练习。（可以在学完第 4 部分后直接学习本章。）

附录

附录 A：JShell 交互式 Java 入门——使用 Java 友好的命令行 REPL（读取-求值-输出循环）来探索、发现和实验。这就好比在 Java 中拥有了类似于 Python 的交互性。^③本附录的许多内容都可以配合本书早期的章节阅读。

附录 B：格式化输出——讲解如何对输出和内存字符串进行格式化。总结了全书已经讨论的格式化特性，并进行了补充。

附录 C：数制系统——介绍二进制（基数 2）、八进制（基数 8）、十进制（基数 10）和十六进制（基数 16）数制系统。

Deitel.com 博客主题

序列化集合；灵活的构造函数体；位运算；带标签的 break 和 continue 语句；JavaFX Subscription（订阅）API；基于 FXGL（FX 游戏库）的 JavaFX 游戏开发入门，涉及动画、碰撞检测、粒子效果等。本书英文版主页 <https://deitel.com/javafp5> 将汇总所有相关的 Java 博客文章链接。

^① 译注：PreparedStatement 对象用于存储预编译的 SQL 语句。预编译后的 SQL 会被数据库缓存，重复执行时无需重新解析和优化，性能更高。

^② 译注：羽毛分形通常由一条主线段开始，递归地在主线段两侧按一定角度和比例生成分支线段，最终形成羽毛状的对称结构。这类似于经典的科赫（Koch）雪花或树形分形，但形状更接近羽毛。

^③ 译注：.NET Core 现在也内置了 C# REPL 支持，允许像 Python 一样交互式执行 C# 代码，简化开发和调试流程。

“对象自然”教学法

面向对象编程书籍传统上采用“晚期对象”（late object）或“早期对象”（early object）的教学法。但这些书中所谓的“早”与“晚”，并非针对“对象”本身，而是指在什么时候教你学习开发自定义类（即构建对象所依据的“蓝图”）。我们曾使用这两种方式编写过多种流行的面向对象编程语言的教材。

什么是“对象自然”教学法？

在撰写 *Intro to Python for Computer Science and Data Science: Learning to Program with AI, Big Data and the Cloud* 这本教材时，我们注意到尽管教学方式符合“晚期对象”模型，但实际上它更具独特性。我们将这种方法总结为“对象自然”（Objects-Natural）教学法，并在后续所有面向对象编程语言教材和专业书籍中采用该方法。

与“晚期对象”类似，“对象自然”教学法也从编程基础内容开始讲解，如数据类型、变量、操作符/运算符、控制语句、方法、数组和字符串等，这些内容均安排在前几章——全部在开始创建自定义类之前。然而，在“对象自然”教学法中，学生会大量练习使用功能强大的现有类，快速创建这些类的对象（通常只需一行代码），并通过简单的 Java 语句让它们执行各种操作。

这正是使用 Java 等成熟的面向对象语言最具吸引力的方面之一。在学习第 8 章的自定义 Java 类以及第 9 章的继承、多态性和接口之前，学生就已经能够熟练使用这些现成的对象了。

大量可用的免费类库

我们强调使用 Java 生态系统中数量庞大的免费、有价值的类库。这些类库主要来自：

- Java API——也就是 Java 标准库；
- 免费的第三方 Java 库，它们通常由开源社区开发。

我们鼓励你在 GitHub 等网站上探索大量由别人写的 Java 代码。阅读由其他程序员编写的代码是学习编程的一种非常高效的方式。

生成式 AI 教学法

在本书创作过程中，生成式 AI 不仅是不可或缺的核心环节，也将成为读者与书中内容互动的重要方式。我们将深入探讨如何将生成式 AI 融入本书的架构设计，它将如何革命性地提升 Java 教育，以及读者应如何在保持审慎批判思维的同时，高效利用这些强大的工具。

在本书的撰写过程中，我们始终将生成式 AI 视为强化内容质量的强大辅助工具，而非作

者工作的替代品。同样，我们也鼓励读者将生成式 AI 用于提升学习与软件开发技能，而非依赖其替代自主思考。我们应用生成式 AI 的体验既富有启发性又充满乐趣——它仿佛一个由多元专家组成的全天候支持团队，当然，这一切仍需我们这些人类的细致监督。

使用生成式 AI 的两种方法

本书采用以下两种方式来集成生成式 AI：

- **提示工程**：除了第 19 章外，我们主要关注通过优化提示（prompt）来提升与生成式 AI 工具的交互效果。可以将这种方式理解为学习一种新的“编程语言”。在提示工程中，需要设计精确的指令或问题，引导 AI 生成所需的结果。还可以迭代优化提示词，不断调整以提高结果的准确性与相关性——提示越清晰，结果越理想。掌握优秀的提示技巧需要大量实践，而这本书将为读者提供丰富的练习机会。
- **结合 AI API 的 Java 编程与提示工程**：第 19 章则融合了提示工程与基于 AI API 的代码开发，重点展示生成式 AI 如何协助你构建和优化完整的 Java 应用程序。我们使用的是 OpenAI 的 API，但读者完全可以换成其他 AI 产品。^①

第 19 章的示例和练习将更深入地探讨如何使用生成式 AI 进行代码生成与分析，并指导读者构建基于 AI API 的 Java 生成式 AI 应用程序。读者将学会如何编写有效的提示词来生成代码片段、调试并优化程序，甚至创建完整的 Java 应用程序。

我们使用的生成式 AI 工具

我们的工作依赖于多个生成式 AI 模型，其中包括 4 个主要的生成式 AI 聊天机器人：^②

- OpenAI ChatGPT
- Google Gemini
- Anthropic Claude
- Perplexity

我们在每个任务中都使用了所有这些工具，因为它们产生的结果有一部分相同，也有一部分不同。换言之，每种工具都有其独特价值。事实上，即使向同一个生成式 AI 输入相同的提示，在不同时间点也可能得到不同结果。每种工具都有其优势与局限。随着技术的发

^① 译注：本书中文版通过第三方来间接使用 OpenAI API，详情请访问本书中文版主页（<https://bookzhou.com>）。

^② 译注：此处列出的是本书英文版所使用的 AI。中文版读者建议使用 DeepSeek、豆包、Kimi 等易于访问的 AI。

展，这些能力与限制也在不断变化。未来，它们的能力很有可能会显著提升。^①

此外，我们还使用了 OpenAI 的 DALL-E（现已整合到 ChatGPT 中）进行图像生成，以及 Sora 进行视频生成。许多生成式 AI 提供商提供了免费试用版本。虽然它们的功能可能不如付费版本强大，但也能带来不错的效果。

生成式 AI 在教育中的角色

生成式 AI 将在职业教育与高等教育领域扮演至关重要的角色，尤其在编程语言教学场景中优势显著。这类技术如同一支全天候在线的虚拟专家团队，能以极低甚至零成本为学生提供即时协助。

当前，不同组织机构与教育机构对生成式 AI 的应用态度存在明显分歧：部分机构就像本书一样积极接纳；部分机构则明确禁止使用；还有一些机构在设置监管措施的前提下允许合理使用。我们鼓励你尽早熟悉这类工具——它们极有可能成为职业生涯中不可或缺的辅助。当然，务必严格遵守所在单位或院校制定的相关政策。

通过生成式 AI 增强学习体验

生成式 AI 可以从以下几个方面提升学习体验：

- 从多个角度解释复杂概念；
- 提供交互式调试支持；
- 当概念理解不清晰时，辅助定位需要复习的内容；
- 通过多样化讲解方式适配不同学习风格；
- 为自己编写的代码提供个性化反馈。

我们建议读者在学习中注重动手实践，将生成式 AI 作为强化学习的工具。书中的“生成式 AI 练习”小节将帮助你：

- 学习“提示工程”——设计和优化自己输入的提示，以获得最佳结果；
- 通过输入提示来探索生成式 AI 的能力边界；
- 了解不同生成式 AI 对提示的响应细节程度；
- 发现解决问题的新思路；
- 对比不同生成式 AI 的优势与局限；
- 理解生成式 AI 可能犯错，有时甚至还会产生“幻觉”，并掌握检测与应对方法；
- 注意到生成式 AI 可能会生成不准确的结果；
- 获得书中知识点的进一步阐释；

^① 译注：已经显著提升了，目前包括 DeepSeek 在内的主流 AI 均推出了“深度思考”功能，准确性大增。

-
- 在需要更多细节时拓展自己的知识面。

我们是如何使用生成式 AI 的？

本书在创作过程中深度整合了生成式 AI 技术，但并非由其直接撰写书稿（特定注明的场景除外）或代码示例，而是借助其优化本书的研究、开发、文字润色及校对等环节。具体应用场景包括：

- 锤炼生成式 AI 提示工程技巧，以获取更优质的输出结果；
- 开展头脑风暴；
- 从书中内容提取要点，生成 PowerPoint 幻灯片草稿，助力教师备课与授课；
- 验证内容准确性；
- 核查书中 Java 编程惯用法及代码规范是否符合当前行业标准；^①
- 检查语法、行文流畅度及表达清晰度；
- 基于内容生成选择题草稿；
- 仅使用书中已讲解的技术，为练习生成答案草稿；
- 直接根据内容生成营销文案草稿；
- 直接根据内容生成博客文章草稿；
- 重构代码以符合编码规范（例如代码缩进、注释相关性及密度等）；
- 代码重构（例如重命名变量、方法和类，将重复代码提取为独立函数等）。

多数情况下，有效的提示工程是关键，通过反复优化对话提示词，往往能显著提升输出质量。

给生成式 AI 用户的建议

我们鼓励读者将生成式 AI 视作一种学习工具，而非单纯的“答案生成器”。正如精通编程需要理解语言和库的功能，有效运用生成式 AI 同样需要通过实践不同的“提示”来培养技能。具体建议如下：

- **记录提示与结果**：持续跟踪使用的提示及其输出质量，这不仅有助于梳理有效策略与无效尝试，还能帮助识别不同生成式 AI 平台的细微差异。久而久之，你将积累一套宝贵的提示库及预期结果，为未来编程工作提供有力支撑。
- **体验多平台工具**：尝试使用多种生成式 AI（包括新兴的那些），通过对比输出结果来理解各种工具的能力差异。
- **探索提示策略**：尝试不同的提示构造方法，找出最能达成目标的策略——不同的编程

^① 参考罗伯特·C. 马丁（Robert C. Martin）的 *Clean Code: A Handbook of Agile Software Craftsmanship* 一书，Addison-Wesley, 2009。中文版名为《代码整洁之道》。

挑战往往需要差异化的提示思路。

- **迭代优化提示**：若生成结果未达预期，调整提示后再次尝试。例如，部分工具倾向于生成流畅专业的文本，而另一些则更擅长深入剖析主题。
- **使用生成式 AI 进行探索与发现**：借助生成式 AI 探索新思路、洞察潜在模式与见解。其广泛的分析能力可处理海量数据并整合为可操作信息，既能提升创造力，也能辅助决策。这种“发现能力”可突破自己现有的知识边界，激发创新思维，助力拓展认知维度与智慧深度。

幻觉与错误

生成式 AI 存在固有局限性。这类技术可能产生“幻觉”，即看似自信地给出实则捏造的答案。要求它们执行任务时，其输出结果有时会包含谬误，例如生成无法正常运行的 Java 代码。在本书中，我们既展示了生成式 AI 的成功交互，也呈现了其不理想的案例。例如，尽管生成式 AI 协助我们在“数制系统”附录中拓展了十六进制的创新应用场景，但在处理一些看似直截了当的任务时却显得力不从心。早在第 2 章，我们就遇到了这样的情况：即使是要求生成式 AI 编写简单的 Java 语句，也可能遭遇失败。我们特意将部分此类失败案例收录于书中，作为富有价值的教学实例，旨在警示读者：生成式 AI 未必正确无误，务必谨慎使用。这也再次凸显了人工监督与验证的极端重要性——尤其是在商业决策和关键任务等核心环节。

为什么要使用付费服务

尽管免费的生成式 AI 很有价值，但付费服务往往能提供更为强大的功能、更优的可用性 & 可靠性。为生成式 AI 提供强大能力的大语言模型，其训练成本本身就极为高昂，因此许多生成式 AI 工具也提供付费模式。例如，单是训练 OpenAI 的 GPT-4 和谷歌的 Gemini，成本就分别达到了 7800 万美元和 1.91 亿美元——这还不包括相关开发人员的薪酬及其他开销。^①我们鼓励读者在条件允许时使用免费或低成本工具，甚至可以让生成式 AI 推荐当下最佳的免费版本。然而，专业开发者为了获得最佳成效，最终往往会选择工具的付费版本。

本书的主旨

本书的主旨是助你为未来做好准备——在个人生活与职业生涯中，生成式 AI 将成为不可或缺的存在。通过在全书贯穿生成式 AI 的应用案例与实践练习，我们旨在帮助你熟悉这些工具，同时理解其优势与潜在风险。在探索生成式 AI 能力的过程中培养适度的审慎态度，将使你能够更高效地驾驭这些技术。

^① Stanford Institute for Human-Centered Artificial Intelligence (HAI), AI Index Report 2024, May 2024。

https://hai-production.s3.amazonaws.com/files/hai_ai-index-report-2024-smaller2.pdf。

特殊的第 19 章

第 19 章展示了如何通过 OpenAI API 交互来开发完整 Java 示例应用程序：

- **文本摘要 (19.4.1 节)**：调用 OpenAI 聊天 API，对 Deitel 视频转录内容生成摘要段落及要点列表。此类文本响应称为“聊天补全”（chat completion）；
- **情感分析 (19.4.2 节)**：请求 OpenAI 聊天 API 分析转录文本的情感倾向，并解释生成式 AI 是如何推导出结论的。
- **无障碍图像描述 (19.4.3 节)**：该多模态应用程序将上传图像，要求 OpenAI 聊天 API 对 4.3 节的 for 循环 UML 图和阿鲁巴（Aruba）海滩^①风景照片进行详细描述。
- **语言检测与翻译 (19.4.4 节)**：要求 OpenAI 聊天 API 将文本翻译为指定的目标语言（自动检测源语言）。示例完成了英语和西班牙语/日语的双向翻译，展示了生成式 AI 当前强大的多语言翻译能力。
- **Java 代码生成 (19.4.5 节)**：调用 OpenAI 聊天 API 来生成 Java 代码，模拟投掷 6 亿次一枚六面骰子，并汇总结果。
- **命名实体识别与结构化输出 (19.4.6 节)**：演示如何生成指定格式的结构化 JSON 响应。该应用执行了一个称为“命名实体识别”（named entity recognition, NER）的“自然语言处理”（natural language processing, NLP）任务。该任务对文本中的日期、时间、数量、地点等实体进行定位和分类。为了使结果更容易处理，我们向 API 指定了确切的 JSON 响应格式，并使用第 11 章介绍的 Jackson 开源库处理结果。
- **语音转文字 (19.5.1 节)**：该多模态应用（语音+文本）将语音文件上传至 OpenAI 的 Whisper 模型，完成音频到文本的转录并返回结果。
- **文字转语音 (19.5.2 节)**：该多模态应用（语音+文本）将文本上传至 OpenAI 的 TTS-1 模型进行语音合成，示例使用同一音色将英、西、日三种语言文本转换为语音。
- **图像生成 (19.6 节)**：展示如何调用 Dall-E API 来生成图像，包括：黑色背景下霓虹色日本动漫风格的哈瓦那犬，以及梵高、达·芬奇风格的图像。
- **视频字幕生成 (19.7.1 节)**：该多模态无障碍应用（语音+文本+视频）提取视频音轨并转录为带时间戳的 JSON 文本，再通过聊天补全生成 VTT 格式的字幕。^②文中展示了 VTT 文本及视频字幕截图。
- **内容审核 (19.8 节)**：使用 OpenAI 的“内容审核 API”来评估文本提示，从而检测骚扰、仇恨言论、暴力、自残及色情等有害或不当内容。

^① 译注：阿鲁巴（Aruba）是一个位于加勒比地区的岛屿。鲁巴是荷兰王国四个构成国之一。由于气候干燥，常见热带植物数量不多，岛上的景象与一般人对加勒比海所抱持的既有印象大异其趣。

^② 维基百科：<https://zh.wikipedia.org/wiki/WebVTT>。

生成式 AI 练习

我们将书中约 600 个生成式 AI 练习的完整列表（一份超百页的 PDF 文档）输入到 ChatGPT、Gemini、Claude 和 Perplexity 中，让它们对这些练习涉及的内容进行分类。随后，我们将分类结果再次输入这 4 个 AI，生成最终摘要并选出最优版本——以下为 Claude 的分类结果：

- **代码生成与实现**：根据需求规范编写全新 Java 程序，涵盖特定语言特性、算法及 API 的实现，创建测试程序与实际应用，为基础及高级任务提供解决方案。
- **代码重构与增强**：实现代码现代化，优化代码结构、可读性和可维护性，在保持功能性的前提下完成不同实现方法间的转换，并提升程序性能。
- **教育内容开发**：创建教程、练习及学习材料，深入探索复杂概念，设计编程练习，撰写全面的文档与指南。
- **技术分析**：分析代码行为与功能实现逻辑，对比不同方法、工具及框架，评估设计决策中的权衡取舍，拆解复杂技术概念。
- **最佳实践与标准**：贯彻编码标准与设计模式，处理安全相关考量，优化性能表现，遵循 Java 开发最佳实践。
- **技术评估**：对比各类库、工具及框架，评估不同方案的优缺点，辅助做出明智的技术选型，探索新功能与更新。
- **调试与错误处理**：排查并修复语法及逻辑错误，实现异常处理机制，提升系统容错能力，预防常见开发陷阱。
- **API 与库集成**：使用 Java 原生 API 及外部库，理解 API 功能与特性，实现集成技术，创建 API 文档与教程。
- **真实场景应用开发**：开发实际用例与行业应用，构建交互式程序（如图形界面、游戏、多媒体应用），复现真实世界场景，创建示例项目。
- **性能优化**：分析并提升程序性能，优化资源使用效率，执行基准测试，实施效率改进措施。
- **创意开发**：构建多模态应用程序，生成可视化效果，创建测试场景与示例数据，开发独特用例。

基于生成式 AI API 的 Java 编程练习

第 19 章提出了一些具有挑战性的项目练习，例如创建可以相互辩论的生成式 AI 多媒体应用程序，以及使用生成式 AI 构建和解决填字游戏。我们将这 94 道题提供给 AI，要求它们提供分类摘要，然后我们对这些摘要进行了总结。以下是由 Claude 生成的内容：

- **多模态应用程序**：构建结合文本、图像、音频、语音和视频功能的综合工具；创建如互动书籍等集成体验；开发多媒体教育内容。
- **基于文本的应用程序**：文档处理（索引、摘要、探索）；创意写作（故事、诗歌、辩论）；语言工具（翻译、语气重写）；专业文档创建（简历、演示文稿）；结构化输出。

-
- **图像处理应用程序**：生成式艺术与设计（标志、时尚、平面图）；技术可视化（UML图）；图像分析与识别。
 - **音频和音乐应用程序**：语音处理（转录、语音克隆）；音乐生成（MIDI、Magenta AI）；多语言音频应用程序；播客和音频内容分析。
 - **教育工具**：编程导师（Java、编码练习）；特定学科学习辅助工具（数学）；课程内容创建；交互式教育体验。
 - **游戏和拼图应用程序**：拼图生成器和解谜器；交互式游戏开发。
 - **视频**：调查和实验生成式 AI 视频创建工具。
 - **聊天机器人开发**：基于角色的聊天体验；专业领域专家。
 - **研究和分析工具**：医疗应用（研究药物发现和个性化医疗）；AI 能力探索；文本检测和分析；教育研究。
 - **创意应用程序**：儿童书籍创作；互动故事讲述；艺术内容生成；创意写作。
 - **实用工具和实用程序**：文档生成器；翻译服务；内容摘要器；专业工具（简历筛选器、演示文稿创建器）。

生成式 AI 教学法的要点

- **尝试多种生成式 AI**：对比不同工具的优势和劣势。
- **精通提示工程**：设计高效提示以获取最佳结果，并建立个人的优质提示库。
- **警惕幻觉**：始终审慎核查生成式 AI 的结果。
- **善用生成式 AI 的优势**：将其作为 Java 学习的辅助工具，而非替代品。
- **遵守所在组织的生成式 AI 使用政策**。

本书增改内容

下面列出了本书关键的新增与更新内容：

- **“对象自然”案例学习**：第 1 章深入浅出地介绍了面向对象编程的基本概念和术语。在早期章节中，你将使用现有的 Java API 类创建并使用强大的对象，无需了解如何编写 Java 类或这些类的具体实现。要等到第 8 章，我们才会开始创建自定义类。我们新增了大量“对象自然”案例学习，它们涉及：**String** 类（第 2 章）：用于超大整数的 **BigInteger** 类（第 3 章）；用于精确货币计算的 **BigDecimal** 类（第 4 章）；日期/时间 API（第 5 章）；大小可动态调整的 **ArrayList** 类（第 6 章）；用于查找文本模式的正则表达式（第 7 章）；对称加密 **AES**（第 7 章）和非对称加密 **RSA**（第 11 章）；其他章节也广泛使用了 Java API 和开源库中类的对象。
- **约 600 个生成式 AI 练习**：大多数小节的末尾都包含“生成式 AI 练习”（共约 450 题），你可以通过提示工程与生成式 AI 进行交互，从而拓展或巩固对当前小节所讲述的主题的理解。此外，许多章的末尾都有一个“专题：生成式 AI 练习”（共约 150 题）。你会感觉像有数十位 Java 专家随时为自己解答学习中遇到的问题，帮助自己深入探索

感兴趣的主题，甚至协助自己编写和调试 Java 代码。更妙的是，这些 AI 之间还在不断竞争，不断推出新版本，只为向你呈现最好的结果。

- **第 19 章：构建基于 AI API 的 Java 生成式 AI 应用程序：**本章展示了如何在写代码时与 OpenAI 的 API 进行交互，涵盖了文本摘要、情感分析、图像描述（用于“无障碍访问”）、多语言翻译、Java 代码生成、命名实体识别、语音转文字、文字转语音、视频字幕、图像生成等主题。另外，章末的 94 个练习涵盖了更多主题。
- **涵盖最新的 Java 语言特性、库特性和编程惯用法：**为了保持书籍内容随 Java 新版本发布而更新，我们将持续在英文版博客（<https://deitel.com/blog>）上发布新代码示例和解释，并链接到本书英文版主页（<https://deitel.com/javafp5>）。^①
- **简化现代面向对象编程：**新增关于 record（记录）类、sealed（密封）类/接口以及一个关于“针对接口编程，而非针对实现编程”的案例学习，重点在于组合（composition）^②和依赖注入。
- **增强的文件处理功能：**介绍如何使用 Jackson 开源库来输入输出 JSON 和 CSV 格式的数据，这些格式在当今的数据科学应用中非常流行。
- **调用流行的 Web 服务：**介绍如何使用 java.net.http 包提供的 Web 联网功能来调用如 OpenWeatherMap 这样的服务。
- **特别专题：数据可视化与泰坦尼克号灾难数据集。**介绍使用 CSV 格式的泰坦尼克号灾难数据集进行数据分析的基础知识。
- **密码学专题：**包括对称加密（AES）和非对称加密（RSA），这两项技术在计算机隐私和安全领域至关重要。
- **数据清洗专题：**介绍为训练自然语言处理（NLP）和生成式 AI 模型而准备文本时所需的简单数据清洗步骤，这是数据科学的关键技术之一。
- **第 1 章新增了对元宇宙及其相关技术的讲解：**简要探讨了元宇宙及其相关技术——虚拟现实（VR）、增强现实（AR）、混合现实（MR）、区块链、加密货币、非同质化代币（NFT）和 Web3——以及软件开发技术、大数据，还有人工智能与计算机科学和数据科学的交叉领域。
- **增强了 JavaFX GUI、图形、动画和视频主题：**这些技术将成为未来元宇宙的核心。我们介绍了 JavaFX 新的 CSS 过渡效果，它们简化了动画实现。
- **JavaFX 游戏开发简介：**在第 17 章中介绍使用开源 FXGL 库进行 JavaFX 游戏开发。
- **JShell：**Java 的命令行 REPL（读取-求值-输出循环）环境，便于快速探索和实验 Java 语言特性与库，提供类似于 Python 的交互式体验。
- **并发、并行与多核性能更新：**针对 Project Loom 的关键技术进行了更新，这些技术包括虚拟线程、结构化并发和作用域值等。

^① 译注：中文版主页（<https://bookzhou.com>）会同步更新。

^② 译注：即 has-a 关系。

-
- **强化了对数据库主题的讲解**：使用流行的 SQLite 数据库管理系统命令行工具交互式地讲解 SQL。示例现在使用 SQLite 数据库。我们将旧版的 Swing JTable 替换为 JavaFX TableView，提供更好的外观、性能和定制能力。
 - **提供大量增改的现代示例。**
 - **自 Java 17 引入的 RandomGenerator**：该类提供了更强的随机数生成能力，适用于模拟场景。我们在安全场景中使用的是 SecureRandom，例如在加密示例中。
 - **大量编程技巧**：数百条来自数十年编程教学经验的编程技巧已完全整合进本书正文。
 - **应用程序编程案例学习**：我们对现有的一些案例学习进行了增强，并新增了聚焦于 AI 与数据科学的案例学习，它们涉及随机数生成模拟、自然语言处理（NLP）、生成式 AI 与启发式编程等。

本书特色

语法着色与高亮显示

本书电子版采用与大多数 IDE 和代码编辑器相似的语法颜色对 Java 代码进行着色。本书印刷版则通过油墨的浓淡对语法进行高亮显示。

正文中的代码

正文中的代码采用等宽字体，例如，`int x = 5;`。

学习目标

每章开头列出该章的学习目标。

插图与图表

本书包含丰富的表格、线条图、UML 图、程序代码和程序输出。

强调 Java 的标准用法

我们严格采用最新版《Java 语言规范》（<https://docs.oracle.com/javase/specs>）中的标准术语。此外，我们通过大量跨领域的实际开发案例（涵盖计算机科学、人工智能、数据科学等），系统展示了 Java 语言的标准实践方法。

把握现代 Java 技术脉络

为确保内容与时俱进，我们系统性地研读了数千份最新技术资料，包括学术论文、技术白皮书、书籍、官方文档、博客文章、论坛讨论、在线研讨会及视频教程。同时，我们通过谷歌快讯（Google Alerts）持续监测数百个与 Java、通用计算、人工智能及数据科学相关

的关键词，实时捕捉技术演进趋势。

数百个现代示例与生成式 AI 练习

我们的代码示例和生成式 AI 练习涵盖了当前计算领域的热门话题。

性能优化

我们关注满足现代应用性能需求的技术和策略。

数据实战体验

第 11 章中，你将处理真实世界的文本数据，包括著名的泰坦尼克号灾难数据集（CSV 格式），该数据集常用于数据分析的入门学习。^①

安全 Java 编程

构建能够抵御病毒和其他恶意软件攻击的工业级系统极具挑战性。在当今的互联网环境下，这类攻击的发生可以是瞬时且全球性的。从开发周期伊始就将安全性融入软件，能够显著减少系统漏洞。我们依据甲骨文公司的《Java SE 安全编码指南》（<https://www.oracle.com/java/technologies/javase/seccodeguide.html>）对本书内容进行了审核，并遵循了与本书技术层级相匹配的安全编码实践。

隐私保护

在 ACM/IEEE 制定的计算机科学、信息技术和网络安全课程指南中，“隐私”一词被提及数百次。^②每个程序员都必须重视隐私问题和相关考量。本书第 7 章和第 11 章将分别探讨密钥密码学（对称加密）和公钥密码学（非对称加密）技术。

伦理

在 ACM/IEEE 制定的计算机科学、信息技术和网络安全课程指南中，“伦理”一词被提及超过百次。^③在第 1 章和第 19 章的若干生成式 AI 练习中，我们将通过互联网搜索与生成式 AI 来研究伦理议题。具体包括：探索智能助手（如亚马逊 Alexa、苹果 Siri 及生成式 AI 语

^① 该数据集名为“TitanicSurvival”，可以访问 <https://vincentarelbundock.github.io/Rdatasets> 来获取。作者是弗兰克·E. 哈勒尔（Frank E. Harrell, Jr.）和托马斯·卡森（Thomas Cason）。

^② “Curricula Recommendations”，<https://www.acm.org/education/curricula-recommendations>。

^③ “Curricula Recommendations”，<https://www.acm.org/education/curricula-recommendations>。

音功能)涉及的隐私与伦理问题,同时分析围绕 OpenAI ChatGPT^①、Dall-E^②等生成式 AI 引发的行业热议与争议。

编程智慧

全书整合了数百条编程技巧,它们贯穿于正文内容之中。

- **良好编程实践**: 强调有助于编写更清晰、更易理解、更易维护程序的技术。
- **常见错误**: 指出常见的编程错误,以降低你犯错的可能性。
- **性能优化建议**: 突出可以提升程序运行速度或减少内存占用的机会。
- **软件工程观察**: 强调影响软件系统构建(尤其是大型系统)的架构与设计问题。
- **界面风格观察(第15章~第17章)**: 介绍图形用户界面(GUI)的设计规范。这些观察帮助你设计出符合行业标准、美观且用户友好的 GUI。

本书导读

此前的“高级概览”一节已对本书的模块化结构进行了概述。本节将作进一步的详述。阅读本节时,建议同时参阅“高级概览”的内容。

本书前几章致力于帮助读者奠定坚实的 Java 基础,后续各章则循序渐进地引入中高级编程主题。书中将深入探讨以下 6 种编程范式:

- 过程式编程(第2章~第7章): 采用我们独有的“对象自然”教学法加以强化;
- 面向对象编程(第8章~第10章);
- 泛型编程(第12章~第13章);
- 函数式编程(第14章);
- 并发编程(第18章);
- 以及这一版新增的生成式 AI 应用程序编程(第19章)。

无论你是需要熟悉教材内容的学生,是正在规划课程的教师,还是在项目准备阶段考虑研读哪些章节的专业软件开发人员,这一节都将助你做出明智的决策。

第1部分: 导论

第1部分和第2部分(第1章~第7章)以友好的、“示例导向”的方式讲解传统编程入门主题。

^① “Introducing ChatGPT”, <https://openai.com/blog/chatgpt>。

^② “Dall-E 2”, <https://openai.com/product/dall-e-2>。

第 1 章：概述、Java 应用程序实操演练与生成式 AI

本章探讨技术趋势与互联网概念，为第 2 章～第 22 章的 Java 编程内容及众多案例学习和练习奠定基础。我们将讨论软件开发过程中可能用到的技术，并介绍 Java API——它们是现成的、可复用的、高质量和高性能的功能模块，能帮助开发者避免“重复造轮子”。

我们还探讨了互联网、万维网、云技术、物联网（IoT）、新兴的元宇宙，以及生成式 AI 领域令人振奋的新进展。我们简要提及诸多当前热门话题，包括开源软件、虚拟化、模拟技术、Web 服务、多核硬件架构、多线程、自然语言处理（NLP）、数据科学、健壮的安全编程、密码学、GitHub、StackOverflow、技术论坛、区块链、同质化代币（NFT）、加密货币（如比特币和以太坊）、通用人工智能（AGI）等，为现代 Java 应用程序开发奠定基础。

本章演示了如何使用 OpenJDK 版本的 Java 开发工具包（JDK）来编译和执行 Java 代码。我们探讨了“大数据”到底有多大，以及它增长的速度有多快。最后，本章以人工智能（AI）的一个简介收尾——这是计算机科学与数据科学的关键交汇点。本章包含 27 个生成式 AI 练习。

第 2 章：Java 编程入门

本章系统介绍了 Java 基础知识并展示关键语言特性，涵盖输入输出机制、基本数据类型、算术操作符及其优先级以及判断（决策）语句等内容。作为“对象自然”教学法的一部分，2.8 节的“对象自然”案例学习将演示如何创建并使用 Java API 的 String 类对象——无需掌握自定义类的开发方法或 String 类的具体实现细节。本章包含 12 个生成式 AI 练习。

第 2 部分：编程基础

第 3 章：控制语句（上）

本章介绍了 Java 的控制语句。你将学习使用 if 和 if...else 选择语句、计数器控制和哨兵控制的 while 循环语句，以及递增、递减和赋值操作符。3.12 节的“对象自然”案例学习演示了如何使用 Java API 的 BigInteger 类创建远超计算机原生整数大小的的大整数。本章包含 16 个生成式 AI 练习。

第 4 章：控制语句（下）

本章介绍了更多 Java 控制语句，包括：for、do...while、switch、break 和 continue。还介绍了逻辑操作符。4.11 节的“对象自然”案例学习演示了如何使用 Java API 的 BigDecimal 类执行精确的货币计算。本章包含 13 个生成式 AI 练习。

第 5 章：方法

本章系统性地介绍自定义方法，探讨随机数生成与模拟技术，并在全书首个随机数模拟案例学习中实际应用这些技术，实现一款经典的掷骰子游戏。我们将讲解 Java API 的随机数生成功能，而在第 7 章和第 11 章中，将进一步使用 Java API 生成不可预测的“非确定性”随机数——这类随机数在密码学等安全应用场景中至关重要。此外，本章还将剖析方法调用栈与激活记录如何为方法的调用/返回机制提供底层支持。

5.12 节的“对象自然”案例学习将介绍 Java 日期/时间 API 的核心特性。该 API 具备强大的日期时间管理能力，支持日期时间计算，并能根据用户的语言、国家和时区（即区域设置）进行本地化格式显示。我们将构建一个程序，允许用户输入出生日期和时间（不必非常精确），进而推算用户出生当天是星期几以及当前的年龄。本章包含 16 个生成式 AI 练习。

第 6 章：数组与 ArrayList

本章开启数据结构的入门教学，系统介绍用于存储列表和表格数据的内置数组。读者将学习数组的定义与初始化、元素访问、方法传递机制，以及数组排序、查找和多维数组操作。我们还将讲解 Arrays 类，其包含用于常见数组操作的工具方法。6.20 节的“对象自然”案例学习将探讨如何创建和操作 Java 集合框架中 ArrayList<E>集合类的对象。

Java API 提供了丰富的预定义集合（即数据结构），可在内存中存储相关对象组。这些类无需开发者了解底层数据存储机制，即可通过高效且经过验证的方法实现数据的组织、存储和检索。这不仅大幅缩短程序开发时间，还能帮助构建更健壮的应用程序。本章包含 28 个生成式 AI 练习。

第 7 章：字符串、NLP 与正则表达式：生成式 AI 基础

本章系统讲解 String、Character 和 StringBuilder 类的核心特性，同时引入自然语言处理（NLP）——这正是 OpenAI ChatGPT、Google Gemini、Anthropic Claude、Perplexity 等强大生成式 AI 的技术基石。本章包含两个“对象自然”案例学习。

- 7.8 节：介绍如何利用 String 类的内置特性以及 java.util.regex 包的特性进行正则表达式模式匹配与文本替换。在演示正则表达式基础用法后，我们将通过其执行简单数据清洗操作，为 NLP 和生成式 AI 模型训练做数据预处理。
- 7.9 节：本节的标题“pMa5tfEKwk59dTvC04Ft1IFQz9mEXnkfYXZwxk4ujGE=”看似乱码，实则暗藏玄机！该案例学习延续本书对安全与隐私的关注，聚焦当今互联世界中至关重要的密码学技术。密码学每日在幕后保障互联网通信与数据存储的安全性和私密性。本节将介绍基于 AES（高级加密标准）算法的对称加密，而第 11 章将通过广为人知的 RSA 算法讲解非对称加密。

本章包含 22 个生成式 AI 练习。

第 3 部分：面向对象编程

第 8 章：使用自定义类进行真实世界建模

本章将开始系统性地讲解现代面向对象编程。Java 具有可扩展性——开发者创建的每个类都会成为一种新类型，用于对象实例化。在第 8 章和第 9 章中，读者将学习 Java 构建实用类及操作类对象的核心功能，首先展示了一个创建和使用简单银行账户类的案例学习。在随机数模拟案例学习中，将结合字符串集合、随机数生成与模拟技术，实现一个基于文本的卡牌洗牌发牌程序。该案例可通过 JavaFX（第 15 章～第 17 章）及维基共享资源库（Wikimedia Commons）的免费卡牌图片进行可视化重构：

https://commons.wikimedia.org/wiki/Category:SVG_English_pattern_playing_cards

通过 `Time` 类案例学习及多个扩展类，我们将深入探讨类构建机制、类成员访问控制、构造函数初始化等内容。演示数据验证及异常抛出机制以处理问题场景，设计包含赋值（`set`）和取值（`get`）方法的类来操作实例变量，并讨论“组合”（`has-a`）关系——即类包含其他类对象的引用作为成员。此外，还会深入解析 `enum`、`static` 和 `final` 关键字，讲解 Java 的垃圾回收机制（如何回收未使用的对象），揭示同一包中类的特殊关系。我们还引入了 `record` 类来创建存储相关数据的不可变对象，并结合 `switch` 表达式的模式匹配特性展开应用演示。本章包含 33 个生成式 AI 练习。

第 9 章：使用继承、多态与接口进行现实世界建模

本章聚焦于继承体系中类间关系，以及这些关系所支持的强大运行时多态处理能力（实现“泛化编程”的便捷性）。在多态性案例学习中，读者将实现 `Employee` 类层次结构，并在执行多态工资计算的应用程序中实践。我们将区分继承（`is-a`）与组合（`has-a`）关系——继承常导致类间高度耦合，增加系统维护难度。同时引入接口（`interface`）概念，它特别适用于为不相关类赋予通用功能，使类对象能以多态方式被处理。

通过在不同 `Employee` 类与 `Invoice` 类中实现的 `Payable` 接口，我们演示了：实现同一接口的类对象，可对接口方法调用做出多态响应。此外，讲解了当前主流编程原则，如“针对接口编程，而非针对实现”和“组合优先于继承”，并通过组合与依赖注入技术重构 `Employee` 类层次结构，创建松耦合的类以提升系统可维护性。本章还将介绍密封类与接口，它们允许设计者控制哪些类可继承超类或实现接口。本章包含 31 个生成式 AI 练习。

第 10 章：深入异常处理

本章延续第 6 章的异常处理内容，展开更深入的讨论。异常处理在**关键业务**（`business-`

critical) 应用开发中至关重要，系统故障可能导致公司运营中断及财务损失。这些业务的例子包括：

- 电子商务平台（如京东、淘宝、亚马逊、eBay 等网上商城）
- 客户关系管理（CRM）软件（如 SalesForce、HubSpot、Zoho 等）

在**关键任务**（mission-critical）系统中，异常处理更是关乎重大经济损失、人身伤害甚至生命安全。这些系统的例子包括：

- 空中交通管制系统；
- 医院生命维持系统；
- 调度警察、消防员及急救人员（emergency medical technician, EMT）的应急响应系统；
- 以及军事系统。

使用 Java 组件时，开发者不仅需要了解其“正常运行”时的行为，还需掌握异常情况下抛出的异常类型。

本章将探讨异常使用场景，演示 Java 的 try 语句捕获处理异常的机制，并介绍 Throwable 异常层次结构、受检（checked）与非受检（unchecked）异常^①、异常链、自定义异常、前置条件、后置条件、断言以及用一个 catch 块捕获多个异常的方法。

此外，还将讲解 try-with-resources 语句，它通过在代码块结束时自动释放文件、网络连接、数据库连接等资源来防止“资源泄漏”。该语句在后续章节中广泛使用，确保系统资源在无需使用时及时释放。本章包含 30 个生成式 AI 练习。

第 11 章：文件、I/O 流、JSON 序列化与 CSV 文件

本章演示了使用文件进行数据持久化的技术，是这一版的重要更新内容。我们首先介绍 Java 如何以编程方式处理文件，探讨文本文件与二进制文件的区别，演示如何使用 NIO 类和接口检索文件与目录信息，并通过 Formatter 对象写入顺序文本文件，通过 Scanner 对象读取文件内容。

接下来，通过多个案例学习，演示如何在流行的数据格式之间转换对象。第一个案例介绍 JavaScript 对象表示法（JavaScript Object Notation, JSON）——一种人机可读的文本格式。

^① 译注：受检异常是编译器强制检查的异常，必须被捕获或者在方法声明中添加 throws 子句。相反，非受检异常可自由处理。关于异常分类的核心在于“责任归属”：受检异常强制调用方处理可能的错误状态，属于方法契约的一部分；非受检异常则是程序运行时无法预料的故障。有趣的是，所有异常在编译时都会被编译器“看到”，区别在于编译器对受检异常会强制验证处理逻辑，对非受检异常则放任自流。

使用主流开源 Jackson 库将对象的 JSON 表示写入文件（序列化），并读取 JSON 以重建对象（反序列化）。第二个 JSON 案例使用 Java API 的 `java.net.http` 包的网络功能，调用 OpenWeatherMap.org 提供的 Web 服务，该服务返回城市天气报告的 JSON 格式数据，然后使用 Jackson 处理响应并显示天气报告。

随后介绍 CSV（逗号分隔值）文件格式及使用 Jackson 库读写 CSV 的方法。CSV 在大数据、数据分析、数据科学领域广泛应用，也常用于自然语言处理、机器学习、深度学习等人工智能场景——这些均为生成式 AI 的核心技术。我们通过 Jackson 库读取泰坦尼克号灾难数据集（记录 1912 年首航撞冰山沉没事故中乘客信息及幸存情况），结合部分数据引入数据分析基础——数据科学的核心技术之一。

在最后一个“对象自然”案例学习中，我们延续了安全性主题，优化了 7.9 节的私钥加密示例：使用 RSA 公钥密码学技术对 AES 私钥进行加密和解密，确保加密消息接收方安全获取私钥。此类技术对数据安全与用户隐私保护至关重要。本章包含 38 个生成式 AI 练习。

第 4 部分：数据结构、泛型集合、Lambda 表达式与流

第 12 章：泛型集合

本章进一步探讨 Java 泛型集合框架，该框架始于第 6 章中介绍的泛型 `ArrayList` 集合。Java 集合框架包含众多预建的泛型数据结构，我们将介绍用于声明每种集合类型的功能的接口、实现这些接口的各种类、处理集合对象的方法，以及用于遍历集合的迭代器机制。读者将了解到，Java API 提供了丰富的常用数据结构，无需自行创建——绝大多数数据结构需求均可通过复用这些 API 功能来予以满足。

我们介绍了便捷工厂方法，它们用于创建小型的“不可变”集合。在学习完第 14 章后，读者将能够以更优雅的方式重构本章的许多示例，并且这些重构更易于并行化，从而提升多核系统的性能。而在第 18 章中，将进一步学习如何利用 Java 的并发集合与并行流操作来优化多核性能。本章包含 30 个生成式 AI 练习。

第 13 章：深入泛型类与方法

本章介绍如何编写自定义泛型类和泛型方法，它们类似于第 12 章中介绍的泛型集合。我们将探讨泛型如何使编译器能够在编译时检测类型不匹配的情况，这一特性被称为“编译时类型安全”。我们还将实现一个泛型方法和一个泛型类。本章包含 24 个生成式 AI 练习。

第 14 章：使用 Lambda 表达式和流进行函数式编程

本章介绍了函数式编程的基本概念。我们使用 Lambda 表达式和流来编写某些类型的程序，使其比以往的技术更快、更简单、更简洁，并有望减少 bug 数量。本章采用模块化设计，

各小节可灵活插入或跳过，每个小节都对应书中前面章节的内容。考虑到 Lambda 和流的便捷性和强大功能，我们在第 14 章后的关键示例中均集成了相关技术。事实上，本章许多小节可提前至书中其他章节讲解：建议学生在学完第 6 章后立即阅读 14.1 节~14.7 节，专业开发人员则可在学完第 4 章后优先学习 14.1~14.5 节。

我们将演示如何优化之前各章已实现的任务逻辑。掌握本章内容后，读者将能够巧妙重构书中的大量示例。在第 18 章中，还将学习如何并行化流操作（即同时执行多个任务），借助多核架构提升性能——这正是 Lambda 表达式和流的核心目标之一。此外，你会了解到：当并行任务修改程序状态（如变量值）时，确保任务正确执行极具挑战性。因此，本章会强调“不可变”原则——不修改处理中的数据源或其他程序状态。本章包含 37 个生成式 AI 练习。学完本章后，可以衔接学习第 22 章。

第 5 部分：JavaFX 图形用户界面、图形与多媒体

第 15 章：JavaFX GUI（上）

从本章开始，我们将介绍关于 JavaFX 图形用户界面（GUI）、图形、多媒体和游戏开发。本章的重点是 GUI 开发——作为用户与应用程序交互的友好机制，优质的 GUI 能赋予应用独特的“视觉体验”。通过提供一致且直观的界面组件，用户可快速熟悉新应用并提升使用效率。本章介绍了 JavaFX GUI 基础知识，涵盖了布局、控件与事件处理程序，将使用 Lambda 表达式实现相关功能。读者还将借助免费的 Scene Builder 工具，通过简单的鼠标拖放操作来快速创建两个 GUI 界面。本章包含 13 个生成式 AI 练习。

第 16 章：JavaFX GUI（下）

本章继续介绍 JavaFX 更多关于 GUI 布局和控件的知识。我们将演示鼠标事件与 RadioButton（单选钮）事件的处理方式，讲解如何设置用于响应控件属性变化（例如 Slider 值的变化）的事件处理程序，以及形状绘制、变量与控件绑定（实现数据变化时控件自动更新）和 GUI 组件外观自定义等技术。读者还将在程序中使用文件选择器（FileChooser）和目录选择器（DirectoryChooser）对话框，方便地选择文件或目录。注意，第 18 章将展示如何处理长时间运行的任务，确保 JavaFX GUI 始终保持灵活响应状态。本章包含 21 个生成式 AI 练习。

第 17 章：JavaFX 图形与多媒体

本章系统介绍 JavaFX 的核心图形与多媒体功能，涵盖以下关键技术：

- **层叠样式表 (CSS)**：用于自定义 JavaFX 节点的外观与字体样式，该技术源自 Web 开发中数十年的成熟实践，可精确且优雅地控制界面风格。

- **二维形状**：支持线段、矩形、圆形、椭圆、弧线、折线、多边形及自定义路径等形状绘制。
- **JavaFX 节点变换**：例如平移（移动）、旋转和缩放（调整大小）等操作。
- **视频处理**：播放视频并控制其回放（例如播放和暂停）。
- **Transition（过渡）和Timeline（时间线）动画**：随时间动态改变节点的属性值（例如旋转角度、大小和位置）。
- **逐帧动画**：利用 AnimationTimer 实现帧级动画控制。
- **CSS 过渡**：简化某些动画效果。
- **Canvas（画布）**：支持用鼠标绘图。
- **三维图形**：提供立方体、圆柱体、球体等基础 3D 模型渲染能力。

我们还介绍了用于 JavaFX 游戏开发的 FXGL 库，并计划在我们的英文版博客中进行更深入的讲解。本章包含 31 个生成式 AI 练习。

第 6 部分：高级主题

第 18 章：并发：从平台线程到虚拟线程

本章聚焦于通过并发和多核编程提高应用程序的性能和响应能力。作为全书最重要的章节之一，本章介绍了 Java 构建和管理多个任务的功能，并已根据最新的 Java 技术和惯用法进行更新，特别是引入了革命性的 Project Loom 技术。

本章展示了几个多线程和多核系统性能案例学习。在“顺序与并行排序性能分析”示例中，我们演示了如何使用预封装的并行算法创建多线程程序，从而在多核计算机架构上显著提升运行速度。例如，我们分别使用顺序排序和并行排序对一亿个值进行排序，并使用 Java 的日期/时间 API 测量性能提升。结果显示，在配备 20 核 Intel Core Ultra 7 265K 处理器的电脑上，并行排序的速度比顺序排序快了将近 1000%。我们还展示了一个并行与顺序流处理的对比示例，并再次使用日期/时间 API 展示性能提升。CompletableFuture 示例演示了长时间计算的顺序与并行执行过程。

在“使用 ArrayBlockingQueue 实现生产者/消费者关系”案例学习中，我们讨论了经典的生产者-消费者模型，并演示如何使用预定义的 ArrayBlockingQueue 类来实现它。我们强调，并发编程难以正确实现，因此应优先使用更高层次、更易用、更不容易出错的并发特性。

我们新增了对 Project Loom 的讲解，该项目通过引入更轻量、更简单的虚拟线程（在 Java 21 中正式推出），增强了 Java 的线程模型。这些虚拟线程相比传统的平台线程占用更少的内存和处理器资源，使得应用程序可以启动大量并发线程，从而满足当今大规模并行应用的需求。Project Loom 降低了编写并发程序的复杂性，使并发更容易使用。我们还介绍了 Project Loom 的新特性：结构化并发（structured concurrency）允许程序员将多个线程视为

一个工作单元（称为作用域），从而更容易管理线程、处理错误并回收不再需要的资源；作用域值（`scoped value`）简化了同一作用域内虚拟线程之间的数据共享，减少对复杂线程同步技术的需求。本章包含 51 个生成式 AI 练习。

第 19 章：构建基于 API 的 Java 生成式 AI 应用程序

本章涵盖多个炫酷且完整的 Java 生成式 AI 应用程序，所有程序均基于 API 开发，完整列表详见之前的“特殊的第 19 章”小节。在之前各章，我们主要引导读者通过生成式 AI 的网页界面手动输入提示，并要求大家研究、比较和运用其输出结果，书中未展示 AI 针对这些提示的具体反馈，相关效果需读者在实操过程中自行观察。

生成式 AI 刚问世时，通常仅接收文本提示并生成单一类型的输出，如文本或图像。如今，许多生成式 AI 已具备多模态能力，能够理解文本、图像、音频甚至视频等多种形式的输入，并据此生成融合文本、图像、音频和视频的复合输出。例如，向生成式 AI 发送语音音频文件，并指令其完成语音转文字、显示转录内容、分析情感倾向（积极、消极或中性）、翻译成其他语言、合成对应语言的语音（例如，让外国人说中国话），甚至依据生成的文本创建图像和视频——这些功能在几年前对大多数开发者而言几乎难以实现。

本章借助 Simple-OpenAI 开源库，帮助 Java 程序员便捷地与 OpenAI 的 API 交互，从而构建功能强大的多模态 Java 生成式 AI 应用程序。读者将认识到，提示工程同样是使用生成式 AI API 进行编程的关键环节。在本章的示例与练习中，你将运用生成式 AI 构建涵盖文本、语音、代码、图像和视频的多样化应用程序。本章包含 94 个生成式 AI 练习和基于 API 的编码练习。

第 20 章：使用 JDBC 和 SQLite 访问数据库

本章探讨用于便捷存储和组织持久数据的数据库管理系统（DBMS）。在简述了数据库概念后，读者将通过流行的开源 SQLite DBMS 的命令行工具，学习结构化查询语言（SQL）的基础知识，实时执行 SQL 查询以对 SQLite 数据库进行创建、读取、更新、删除（CRUD）等基本操作，并即时查看执行结果。

接下来，我们将使用 Java 的 JDBC API 以编程方式与 SQLite 数据库交互。JDBC 具有可移植性，因此本章展示的代码可在众多流行的开源和专有数据库管理系统中运行。我们将演示数据库连接、执行更新和检索数据的查询，以及在 JavaFX 的 `TableView` 中显示查询结果的方法。本章重点介绍一个基于 JavaFX 的、数据库驱动的通讯录应用程序，该程序演示了如何使用预编译语句（`prepared statement`）实现可复用、更安全的参数化查询。读者将在练习中为该应用程序添加更新和删除功能。

正如生成式 AI 可以为我们的编写 Java 代码一样，它们也能够生成 SQL 代码，这一点将在本章的生成式 AI 练习中得到体现。本章包含 19 个生成式 AI 练习。

第 21 章：Java 平台模块系统(JPMS)

本章介绍模块机制。作为比包更高层级的聚合单元，“模块”支持对代码进行精准组织。默认情况下，模块在封装实现细节的同时，还允许开发者显式控制向客户端代码公开的内容。这一特性有助于开发者在构建、维护和演进大型软件系统时提升效率。我们首先讨论 JPMS 的设计目标，随后演示自定义包与模块的创建流程，包括声明模块依赖关系、指定模块向其他模块显式公开的包，以及定义模块提供或消费的服务。此外，还将探讨如何控制模块间的反射访问——它允许程序在运行时检查和操作类及其成员。例如，JavaFX 就是利用反射将程序的 GUI 控件与处理用户交互的 Java 代码进行绑定。本章包含 31 个生成式 AI 练习。

第 22 章：递归与 Big O 表示法

本章探讨计算机科学中的经典核心主题。首先通过阶乘和斐波那契数列计算，介绍递归方法的概念——即直接或间接调用自身的方法。我们将开发一个 JavaFX 应用程序，生成自定义的“羽毛分形图”，这是一种通过递归重复模式生成的几何图形。此外，本章还将以通俗易懂的方式介绍 Big O 表示法，该表示法用于衡量算法解决问题的复杂度与所需处理数据量之间的关系。本章包含 11 个生成式 AI 练习，可在学完第 14 章后衔接学习。

附录

附录 A：JShell 交互式 Java 入门

本附录为选读内容，介绍 Java 提供的 JShell——一个友好的命令行 REPL（读取-求值-输出循环）环境，支持快速探索、发现和实验 Java 语言特性与库——好比在 Java 中拥有了类似于 Python 的交互性。若教师希望从课程初始阶段引入 JShell，附录中许多小节可与书中早期章节结合讲授。

JShell 通过 REPL 机制替代传统的编辑-编译-执行流程，无需编写完整程序，只需输入 JShell 命令或 Java 代码片段。当输入代码片段时，JShell 会立即执行以下循环操作：

- 读取；
- 求值（计算）；
- 输出（显示或打印）有助于查看代码执行结果的消息；
- 如此循环，以处理下一个代码片段。

本附录以丰富示例为核心，帮助学生快速掌握 JShell 基础知识。JShell 的即时反馈特性既能持续吸引注意力，又能加速学习与软件开发进程。学生将发现 JShell 易于使用且充满趣味，不仅能帮助读者更快速深入地学习 Java 特性，还能验证这些特性是否按预期工作。对于教师而言，JShell 能够鼓励学生深度探索并优化学习过程；对于专业开发者，JShell 则有

助于快速原型设计关键代码段，以及发现和试验新 API。

附录 B：格式化输出

本附录介绍了如何对输出和内存字符串进行格式化。总结了全书已经讨论的格式化特性，并进行了补充。

附录 C：数制系统

本附录讨论了二进制（基数 2）、八进制（基数 8）、十进制（基数 10）和十六进制（基数 16）数制系统。

重要的开发者网站：GitHub 与 StackOverflow

准备[成为程序员]的最佳途径就是编写程序并研究其他人已经写好的优秀程序。我自己以前就是去翻计算机科学中心的垃圾桶，找出他们的操作系统的程序清单。^①

——威廉·H. 比尔·盖茨（微软创始人）

你可能正在使用 StackOverflow 和 GitHub 这样的知名平台。如果还没有，强烈建议立即开始体验。

- StackOverflow 是全球最受欢迎的编程问答社区之一。我们遇到的绝大多数技术问题，很可能已经在这里被讨论过。无论是调试代码还是学习最佳实践，在这里都能找到高质量的解答。事实上，我们在编写本书时，许多复杂问题的 Google 搜索结果首位都来自 StackOverflow。
- GitHub（现由微软运营^②）是一个绝佳的开源代码平台。可以在这里探索海量的开源项目，借鉴其中的代码，甚至贡献自己的代码以回馈开源社区。正如开源精神所定义的：“开源软件是指源代码公开的软件，任何人都能查看、修改并改进其设计。”GitHub 目前拥有超过 1.5 亿开发者，托管着 4.2 亿个代码仓库，涵盖数百种编程语言。

^① 引自苏珊·拉默斯（Susan Lammers）所著《编程大师访谈录》（*Programmers at Work: Interviews With 19 Programmers Who Shaped the Computer Industry*），微软出版社，1986 年。

^② Microsoft, “Microsoft to Acquire GitHub for \$7.5 Billion”, *Microsoft News*, June 4, 2018, <https://news.microsoft.com/source/2018/06/04/microsoft-to-acquire-github-for-7-5-billion/>。

①仅 2024 年，开发者就提交了 52 亿次代码贡献。②平台提供强大的版本控制工具，帮助开发者高效管理开源和私有项目。其中，Java 开源社区尤为活跃，超过 26 万个 Java 代码仓库持续更新。③我们鼓励你深入阅读并实际运行这些开源 Java 代码，这是学习编程的高效方式，也与我们所倡导的、现在更进一步由生成式 AI 增强的“实时编码”和“对象自然”教学理念高度契合。

Deitel.com 网站与 Deitel 博客

在本书英文版主页 <https://deitel.com/javafp5> 上，我们提供了：

- 本书的读者评价；
- 指向本书 GitHub 仓库的链接，用于下载示例代码；
- 本书后续更新信息。

如需了解如何下载代码示例和设置 Java 开发环境，请参见本书前言之后的“准备工作”。

当我们在 <https://deitel.com/blog> 发布与 Java 相关的博客文章时，会将每篇文章链接到本书英文版主页。

本书使用的软件

本书所需的大部分软件都可以免费下载，只是第 19 章的一些示例使用了付费服务。请参见前言之后的“准备工作”获取下载链接。我们使用免费的 Open JDK 在 macOS 和 Windows 上编写 Java 示例代码。另外，我们使用免费的 JetBrains IntelliJ IDEA 社区版 IDE 开发了一个工具类，供第 19 章的示例使用。

Java 文档链接

本书包含大量引用资料，涵盖撰写手稿时研究的论文、文章、白皮书、视频、博客、在线文档等。若需获取更多信息与见解，你可能需要访问下面列出的资源：

① “Let’s build from here: The complete developer platform to build, scale, and deliver secure software”，<https://github.com/about>。

② “Octoverse 2024: The state of open source software”，<https://octoverse.github.com>。

③ “Java”，<https://github.com/topics/java>。

-
- 最新版官方 Java 文档，网址是 <https://docs.oracle.com/en/java/javase/index.html>；
 - Java API 文档，网址是 <https://docs.oracle.com/en/java/javase/24/docs/api/index.html>。注意，由于 Java 每 6 个月发布新版本，因此访问对应 API 文档时，将链接中的 24 替换为新版本号（如 25、26 等）即可；
 - 我们还频繁地引用了最新的 Java SE（标准版）规范，它同时提供 HTML 和 PDF 格式，可以访问 <https://docs.oracle.com/javase/specs> 来获取；
 - 最后，可以访问 <https://openjdk.org/projects/jdk> 来跟踪未来的版本更新。

有问题在哪里寻找答案？

流行的 Java 和通用编程在线论坛包括：

- <https://stackoverflow.com>
- <https://www.reddit.com/r/java/>
- <https://dev.to/t/java>

此外，厂商通常会为其工具和库提供论坛。许多库托管在 <https://github.com> 上。一些库维护者会在 GitHub 项目页面的“Issues”区域提供支持。

生成式 AI

你可以便捷地向生成式 AI 提问并获得即时回复。建议咨询多个 AI 并对比结果。需注意，生成式 AI 可能出错或产生幻觉，需通过在线搜索验证结果。尽管撰写手稿时多次遇到这些问题，但与生成式 AI 合作提升了我们创建高效教学内容的能力。我们始终期待它们对提示的反馈。

一定要把 AI 用起来！

与作者沟通

阅读本书时，如果有任何问题，可以通过 deitel@deitel.com 联系我们。我们会尽快回复。

与译者沟通

本书中文版读者可以访问译者博客（<https://bookzhou.com>）来联系译者并获取中文版资源，包括中文版源代码和勘误等。

加入 Deitel & Associates 公司的社交媒体社区

读者可以通过以下社交媒体平台关注 Deitel 的最新动态：

- LinkedIn (<https://linkedin.com/company/deitel-&-associates>)
- Facebook (<https://facebook.com/deitelfan>)
- Instagram (<https://instagram.com/DeitelFan>)
- X (@deitel 或 <https://x.com/deitel>)
- YouTube (<https://youtube.com/DeitelTV>)
- Mastodon (<https://mastodon.social/@deitel>)

致谢

感谢芭芭拉·戴特尔（Barbara Deitel）为项目投入大量时间进行技术研究。我们有幸与培生（Pearson）专业的出版团队合作。特别感谢培生 IT 与专业图书部门执行编辑劳拉·勒温（Laura Lewin）的指导、智慧与热忱，她带领团队负责我们所有纸质版和数字版专业图书的出版工作。杰克琳·苏格拉卡姆（Jackleen Sougrakpam）和艾琳·沙利文（Erin Sullivan）分别负责招募本书的行业与学术审稿人，并全程管理审稿流程；朱莉·纳希尔（Julie Nahil）统筹了纸质版与数字版的制作工作；阿什利·桑托拉（Ashley Santora）、凯特琳·科丁顿（Kaitlyn Coddington）与詹内尔·福什勒（Jenell Forschler）从版权角度协助确认可被培生接受的生成式 AI 工具及输出内容。丘蒂·普拉瑟特西特（Chuti Prasertsith）从培生授权资源库中选取封面素材并设计多版方案，杰西卡·戴特尔（Jessica Deitel）以专业设计理念协助最终封面的选定。

审稿人

我们衷心感谢各位审稿人的辛勤付出——包括 Oracle Java 团队成员、Oracle Java 社区的 Java Champion、行业专家及学界翘楚。他们对文本与程序进行了细致审校，并为优化内容呈现提供了诸多宝贵建议。在第五版的修订中，他们协助我们完善了全新的生成式 AI 教学法与自然对象教学法。书中若存在任何疏漏，均由我们负责。

特别感谢 JavaFX 专家约翰·沃斯（Johan Vos）和卡尔·迪亚（Carl Dea）以及前几版的吉姆·韦弗（Jim Weaver）、乔纳森·贾尔斯（Jonathan Giles）和西蒙·里特（Simon Ritter），在第 15 章～第 17 章的 JavaFX GUI 和图形/多媒体内容上给予的专业指导。

第 5 版专业与学术审稿人名单如下：

- 珍妮·博亚斯基（Jeanne Boyarsky）：CodeRanch 成员、Java Champion，合著了《OCP Oracle 认证专家 Java SE 21 开发人员学习指南》（*OCP Oracle Certified Professional Java SE 21 Developer Study Guide*）。
- 卡尔·迪亚（Carl Dea）：某全球商业咨询服务公司首席软件开发人员，合著了《JavaFX 9 实例教程》（*JavaFX 9 by Example*）。

-
- 特里莎·吉 (Trisha Gee): Java Champion。
 - 西蒙·罗伯茨 (Simon Roberts): Dancing Cloud Services, LLC 总裁。
 - 何塞·安东尼奥·冈萨雷斯·塞科 (José Antonio González Seco): 西班牙安达卢西亚议员。
 - 罗恩·维恩 (Ron Veen) 与大卫·弗莱明克斯 (David Vlijmincx): 分别为 Team Rockstars IT 开发人员与 JPoint 高级软件开发人员, 合著了《虚拟线程、结构化并发与作用域值: 探索 Java 新线程模型》(*Virtual Threads, Structured Concurrency, and Scoped Values: Explore Java's New Threading Model*) 以及《云原生开发与向 Jakarta EE 迁移》(*Cloud-Native Development and Migration to Jakarta EE*)。
 - 约翰·沃斯 (Johan Vos): Gluon 联合创始人兼 CTO, 负责云产品, Java Champion, 合著了《JavaFX 现代客户端权威指南: 跨平台移动与云开发 (基于 JavaFX 21 与 23 更新)》(*The Definitive Guide to Modern Java Clients with JavaFX: Cross-Platform Mobile and Cloud Development Updated for JavaFX 21 and 23*)。
 - 布莱恩·加拿大 (Brian Canada): 南卡罗来纳大学博福特分校计算科学教授。
 - 鲍勃·迈尔斯 (Bob Myers): 佛罗里达州立大学计算机科学系。
 - 艾米丽·纳瓦罗 (Emily Navarro): 加州大学欧文分校信息学系特邀讲师。

特别感谢布莱恩·戈茨

Oracle Java 语言架构师、《Java 并发编程实战》(*Java Concurrency in Practice*) 合著者布莱恩·戈茨 (Brian Goetz) 多年来持续审阅我们的 Java 相关内容。他的深刻见解与建设性意见, 极大地深化了我们对 Java 并发编程及面向对象编程精髓的理解。他还耐心指导我们“正确实现”技术细节。能成为他的学生, 我们深感荣幸。

特别感谢罗伯特·菲尔德

甲骨文 (Oracle) 公司 JShell 架构师罗伯特·菲尔德 (Robert Field) 在 JShell 工具开发期间审阅了本书的 JShell 附录内容, 并对我们提出的技术问题、遇到的错误反馈及改进建议逐一回复。由 JShell 的核心开发者亲自审校内容, 且见证部分建议被纳入产品优化, 这对我们而言是一次极为珍贵的专业体验。

谷歌搜索

感谢谷歌 (Google) 公司, 它的搜索引擎回答了我们的查询, 每次都可以在几分之一秒内完成, 无论白天还是晚上, 而且还免费。这是我们过去 20 年的研究过程中所享受到的最好的生产力提升工具, 没有之一。

Grammarly

我们的所有（英文版）稿件现在都会使用 Grammarly 的付费版本。这个工具被描述成“人工智能驱动的写作助手”，能帮助你“写出大胆、清晰、没有错误的文章”。Grammarly 还提供了强大的免费工具，可以集成到几种流行的 Web 浏览器、Microsoft Office 365 和 Google Docs 中。

生成式 AI

在本书的创作过程中，我们使用了 OpenAI ChatGPT、Google Gemini、Anthropic Claude 的付费版本，以及 Perplexity 的免费版本，用于研究 Java 新主题、核实技术细节、检验代码是否符合最新 Java 编程规范及注释要求、提出内容优化建议、确保知识体系的全面重写等。通过这些工具获取的大量信息，已融入书中约 600 个生成式 AI 练习及基于 API 的编码练习中，其中第 19 章集中呈现了主要的基于 API 的编码案例学习。^①

甲骨文、Gluon 和 OpenJFX 社区

感谢甲骨文（Oracle）公司、Gluon 及 OpenJFX 社区，三方协同推动 JavaFX 技术的持续演进，每次 Java SDK 新版本发布时，均会同步推出对应的 JavaFX 版本。同时，感谢 Gluon 提供便捷易用的拖放式 Scene Builder GUI 开发工具。

欢迎来到因生成式 AI 而焕新的 Java 编程世界！三十年来，我们持续耕耘于 Java 专业与学术内容的创作，不断推出迭代版本。愿你在阅读本书时，既能获得充实而富有挑战的学习体验，感受探索之趣；同时也能尽情领略现代 Java 软件开发的独特魅力。

作者简介

保罗·戴特尔是 Deitel & Associates 公司 CEO 兼 CTO，拥有 44 年计算机行业的工作经验，毕业于麻省理工学院。作为全球最有经验的编程语言培训师之一，他从 1992 年以来，就一直为软件开发人员教授专业课程。他向 Deitel & Associates 公司的院校、行业、政府和军事客户提供了数以百计的编程课程，这些客户包括加州大学洛杉矶分校（UCLA）、思科（Cisco）、IBM、西门子（Siemens）、Sun Microsystems（现在的甲骨文公司）、戴尔（Dell）、Fidelity、肯尼迪航天中心的 NASA、美国国家强风暴实验室（National Severe Storm Laboratory）、白沙导弹发射场（White Sands Missile Range）、Rogue Wave Software、波音（Boeing）、彪马（Puma）和 iRobot 等。他与合著者哈维·M. 戴特尔博士合作打造了

^① 译注：本书中文版使用的免费 AI 包括但不限于：DeepSeek、通义千问、Google Gemini、豆包和 Kimi。使用的付费 AI 是 Open AI（通过第三方中转，详情请参见本书中文版主页 <https://bookzhou.com>）。

全球最畅销的编程语言教材、专业书籍、视频和互动多媒体电子学习内容，做过许多线上和线下的培训以及虚拟/现场培训的主持人。

哈维·M. 戴特尔博士是 Deitel & Associates 公司主席兼首席战略官，拥有超过 64 年的计算机行业工作经验。戴特尔博士在麻省理工学院获得电子工程学士和硕士学位，在波士顿大学获得数学博士学位——在计算机专业从这些专业分离出去之前，戴特尔博士已经学习过计算机知识。他具有丰富的大学教学经验。在与儿子保罗 1991 年创立 Deitel & Associates 公司之前，戴特尔博士是波士顿大学计算机科学系主任并拥有终身教职。他们的出版物享有国际声誉并被翻译成日文、德文、俄文、西班牙文、法文、波兰文、意大利文、简体中文、繁体中文、韩文、葡萄牙文、希腊文、乌尔都文和土耳其文。戴特尔博士为许多大公司、学术机构、政府部门和军事机构提供了数百场专业编程培训。

译者简介

周靖 (<https://bookzhou.com>) 是资深软件专家，Microsoft MVP，拥有超过 20 年的 C++、C# 和 Java 开发经验。随着 AI 时代的到来，他也喜欢上了使用 Python 进行大语言模型的开发。周靖不仅是一位技术精湛的程序员，还是一个热情的烹饪爱好者，特别钟爱云贵川菜系，深受擅长于朴实地教大家做饭的王刚老师的启发和影响。周靖多年来出版了多部经典著作的中译本，其中包括畅销的《Windows 核心编程》(Windows via C++)《Visual C# 从入门到精通》《C# 本质论》《CLR via C#》《学习 C++20》和《机器学习与人工智能实战》等，并发表了许多精彩的技术文章。在业余时间，周靖还在出版社的邀请下，审校和重译了一些重点书籍，其中包括《代码大全 2》纪念版和《实时渲染 4》(RTR4) 等经典之作。周靖的贡献不仅限于技术层面，他对烹饪的热爱以及对技术传播的热情使他在多个领域都取得了显著的成就。他希望通过自己的努力，激励更多的人投身于编程和技术的学习之中。

Deitel & Associates 公司简介

Deitel & Associates 公司由保罗 (Paul Deitel) 和哈维 (Harvey Deitel) 创立，是一家国际知名的内容创作和企业培训组织，擅长计算机编程语言、对象技术、移动应用开发以及互联网和 Web 软件技术。公司培训的客户包括一些全球最大型的公司、政府机构、军事部门和学术机构。公司提供讲师现场指导的培训课程，为世界各地的客户提供虚拟或现场培训，并在 O'Reilly Online Learning (前称是 Safari Books Online) 上为培生教育提供线上培训 (<https://learning.oreilly.com>)。

通过与培生公司 50 年的出版合作，Deitel & Associates 以印刷和电子书形式发行了大量行业领先的编程专业书籍和大学教科书、LiveLessons 视频课程、O'Reilly Online Learning 实时培训课程和 Revel 交互式多媒体大学课程。

如需联系 Deitel & Associates 公司和作者或要求提供全球范围内的虚拟或现场由讲师指导

的培训方案，请发送电子邮件到以下邮箱：

deitel@deitel.com

要想进一步了解 Deitel 的虚拟和现场企业培训，请访问以下网站：

<https://deitel.com/training>

个人如需购买 Deitel 的书籍，请访问以下网站：

<https://amazon.com>

<https://barnesandnoble.com/>

准备工作

在使用本书之前，请先了解本书采用的约定，并设置好计算机以编译和运行示例程序。若操作说明发生变更，我们将在以下网址发布更新：

<https://deitel.com/javafp5>

获取代码示例

本书英文版示例代码可以从 GitHub 仓库下载，地址为：

<https://github.com/pdeitel/JavaForProgrammers5e>

中文版 GitHub 仓库：

<https://github.com/transbot/JavaForProgrammers5eChinese>

单击绿色的“<> Code”按钮，然后从弹出菜单中选择“Download ZIP”。这会下载 `JavaForProgrammers5e-main.zip`（英文版）或 `JavaForProgrammers5eChinese-main.zip`（中文版）压缩包。接下来，请解压该压缩包的内容：

- 在 macOS 上请双击该 ZIP 文件并解压其内容，进入解压后的 `JavaForProgrammers5e-main` 或 `JavaForProgrammers5eChinese-main` 文件夹，将其中的 `examples` 文件夹拖放到“文稿”文件夹中；
- 在 Windows 上请双击该 ZIP 文件并解压其内容，进入 `JavaForProgrammers5e-main` 或 `JavaForProgrammers5eChinese-main` 文件夹，然后将 `examples` 文件夹拖放到“文档”文件夹中。

本书的所有说明和代码示例都假设这些示例位于你的用户账户的“文稿”或“文档”文件夹下的一个名为 `examples` 的子文件夹中，其结构如下：

```
文稿（或文档）
├── examples
│   ├── ch01
│   ├── ch02
│   ├── ch03
│   └── ...
```

如果将 `examples` 文件夹放到其他位置或对其重命名，则需要相应地更新我们提供的说明和代码路径。

Java 开发工具包(JDK)

本书示例使用的是 Java 24 版本的 Java 开发工具包 (JDK)。①JDK 包含用于编译 Java 程序的命令 `javac`、用于运行 Java 程序的命令 `java`，以及其他工具。我们在第 1 章的 1.7 节中介绍了如何编译和运行 Java 程序。

我们使用的是免费的开源版本 OpenJDK，下载地址如下：

<https://adoptium.net/temurin/releases/>

该网站由 Eclipse 基金会管理，提供适用于多种操作系统和硬件平台的下载版本。下面将介绍如何为 Windows 和 macOS 获取 OpenJDK。也可以使用更高版本的 JDK 与本书配合使用。

Windows 用户

请访问 <https://adoptium.net/temurin/releases/>，首先单击“**All Versions**”，并选择以下选项：

- 在“**Operating System**”（操作系统）下选择 **Windows**；
- 在“**Architecture**”（体系结构）下选择 **x64**；
- 在“**Windows**”下选择 **JDK**；
- 在“**Version**”（版本）下选择 **24**（或更高版本）。②

接着，请单击与“.msi”字样对应的下载图标来下载 OpenJDK 安装文件。下载完成后，双击已保存到“**下载**”文件夹中的安装程序，它的文件名类似于：

```
OpenJDK24U-jdk_x64_windows_hotspot_24.0.1_9.msi
```

请双击该安装文件，在随后出现的对话框中执行以下步骤：

1. 单击“**下一步**”进入“**最终用户许可协议**”。必须接受许可协议才能安装该软件。阅读协议后，请勾选“**我接受许可协议中的条款**”，然后单击“**下一步**”；
2. 选择是为自己安装 JDK，还是为该计算机上的所有用户安装，然后单击“**下一步**”；
3. 默认情况下，安装程序会选中前两个选项（修改 `PATH` 变量值和关联 `jar` 文件）。我们建议保留默认设置和安装位置。单击“**下一步**”，然后单击“**安装**”；

① 译注：截至 2025 年 6 月，最新版本是 Java 24/JDK 24。随着 2025 年 9 月 Java 25 的发布，将在本书中文版主页（<https://bookzhou.com>）上更新最新的 OpenJDK 下载和安装教程。

② 译注：本书英文版基于 Java 23，但使用版本 24 完全没有问题。因此，本书中文版以 Java 24 为准。但是，当你拿到本书中文版时，根据每 6 个月发布 Java 新版本的规律，Java 至少迭代到了版本 25（这同时会是一个 LTS 版本）。请根据自己的实际情况选择 23 或更高的版本，这不会影响本书的学习。

-
4. 在“用户帐户控制”（UAC）对话框中，单击“是”以允许安装程序对系统进行更改并安装 JDK；
 5. 安装完成后，单击“完成”。

为了确认 Java 是否正确安装，在 Windows 任务栏的搜索框中输入“终端”，然后在搜索结果中选择“终端”，随后会打开一个 Windows PowerShell 窗口。在该窗口中输入命令：

```
java --version
```

然后按 Enter 键执行，结果如下所示：

```
openjdk 24.0.1 2025-04-15
OpenJDK Runtime Environment Temurin-24.0.1+9 (build 24.0.1+9)
OpenJDK 64-Bit Server VM Temurin-24.0.1+9 (build 24.0.1+9, mixed mode, sharing)
```

macOS 用户

请访问 <https://adoptium.net/temurin/releases>，首先单击“All Versions”，并选择以下选项：

- 在“Operating System”（操作系统）下选择 macOS；
- 在“Architecture”（体系结构）下，如果是 Apple 芯片（M 系列处理器）的 Mac，则选择 aarch64；否则为 Intel 处理器的 Mac 选择 x64；
- 在“macOS”下选择 JDK；
- 在“Version”（版本）下选择 24（或更高版本）。

接着，请单击与“.pkg”字样对应的下载图标来下载 OpenJDK 安装文件。下载完成后，双击已保存到“下载”文件夹中的安装程序。如果是使用 M 系列处理器的 Mac，它的文件名类似于：

```
OpenJDK24U-jdk_aarch64_mac_hotspot_24.0.1_9.pkg
```

如果是使用 Intel 系列处理器的 Mac，文件名则类似于：

```
OpenJDK24U-jdk_x64_mac_hotspot_24.0.1_9.pkg
```

请双击该安装文件，在随后出现的对话框中执行以下步骤：

1. 单击“继续”进入“最终用户许可协议”。必须接受许可协议才能安装该软件。阅读协议后，请勾选“我接受许可协议中的条款”，然后单击“继续”；
2. 单击“安装”，如果询问系统密码，请输入后单击“安装软件”；
3. 安装完成后，单击“关闭”。

为了确认 Java 是否正确安装，请在“访达”（Finder）中选择顶部菜单栏的“前往”|“实用工具”，然后打开“终端.app”。在“终端”窗口中输入以下命令：

```
java -version
```

然后按 Return 或 Enter 键执行，结果如下所示：

```
openjdk 24.0.1 2025-04-15
OpenJDK Runtime Environment Temurin-24.0.1+9 (build 24.0.1+9)
OpenJDK 64-Bit Server VM Temurin-24.0.1+9 (build 24.0.1+9, mixed mode, sharing)
```

Java 的发布节奏

自 2017 年 Java 9 发布以来，Java 开始采取每 6 个月一次的发布节奏。新的 Java 版本会在每年的 3 月和 9 月发布，这样可以更快地推出新的语言和类库特性。每两年会有一个版本被指定为**长期支持**（Long-Term Support, LTS）版本。这种版本主要面向需要稳定性和长期维护支持的企业用户。非 LTS 版本的支持周期较短，旨在鼓励开发者尽早采用新技术和创新特性。目前最新的 LTS 版本是 Java 21，下一个 LTS 版本将是 Java 25，预计于 2025 年 9 月发布。

Java 集成开发环境(IDE)

Java 开发领域存在多种**集成开发环境**（Integrated Development Environments, IDE），这类软件集成了代码编辑器、编译器、调试器等软件开发必备工具，助力开发者构建应用程序。由于不同 IDE 的使用流程存在差异，本书多数示例采用 JDK 自带的命令行工具。第 1 章将提供 JDK 的一个“实战演练”（test-drive），同时可在本书英文版主页（<https://deitel.com/javafp5>）获取以下主流 IDE 的入门指引：

- JetBrains IntelliJ IDEA 社区版
- Eclipse 基金会的 Eclipse IDE for Java Developers
- 微软 Visual Studio Code 跨平台编辑器（通过插件支持 Java 开发）^①

上述 IDE 在 Java 开发社区中占据主要市场份额，均支持 Windows、macOS 和 Linux 系统。注意，在安装任何工具前，请先按之前的说明完成 OpenJDK 的安装。

IntelliJ IDEA 社区版

推荐通过 JetBrains Toolbox 应用程序安装 IntelliJ IDEA 社区版，下载地址如下：

<https://jetbrains.com/toolbox-app>

^① 译注：Visual Studio Code 是本书中文版首选 IDE。

网站会自动识别操作系统并显示对应安装程序，请单击“下载”按钮获取安装文件。^①下载完成后，在“下载”文件夹中找到安装文件，双击并按提示完成 Toolbox 应用程序的安装。

安装好之后，请找到 Toolbox 应用程序图标来打开它的窗口。

- macOS 用户可以在屏幕右上角找到 Toolbox 图标。
- Windows 用户可以在屏幕右下角的任务栏托盘中找到 Toolbox 图标。可能需要单击那个向上的小箭头才能看到所有托盘图标。

在 Toolbox 窗口中，向下滚动找到 IntelliJ IDEA Community Edition，然后单击旁边的“安装”按钮即可开始安装。

Eclipse IDE for Java Developers

Eclipse 基金会在以下网址提供了 Eclipse IDE 的分步安装指令：

<https://eclipse.org/downloads/packages/installer>

在步骤 3，请选择安装 Eclipse IDE for Java Developers，然后按剩余的指令完成安装。

Visual Studio Code

微软在以下网址提供了 Visual Studio Code 的下载和安装指令：

<https://code.visualstudio.com/docs/setup/windows>

macOS 用户请访问以下网址：

<https://code.visualstudio.com/docs/setup/mac>

安装好 Visual Studio Code 后，请访问以下网址来了解如何配置 Java 开发环境：^②

<https://code.visualstudio.com/docs/java/java-tutorial>

JavaFX SDK

第 15 章～第 17 章将使用 JavaFX 构建图形用户界面（GUI）、图形和多媒体应用程序。需要注意的是，JavaFX 需独立于 JDK 进行安装，因其并未包含在 JDK 中。JavaFX 由包括甲

^① 译注：可以利用网站顶部的选项将语言更改为“简体中文”。

^② 译注：本书中文版主要使用 VS Code 和 IntelliJ IDEA 社区版开发。

骨文在内的多家企业及个人组成的联盟与 JDK 并行开发，其中甲骨文公司负责监督 Java 的整体开发进程。每个新 Java 版本发布时，都会同步推出对应的新版 JavaFX。

下载 JavaFX SDK

要使用 JavaFX，首先需要安装 JavaFX 软件开发工具包（SDK），可以从以下网址下载：

<https://gluonhq.com/products/javafx>

在 Downloads（下载）区域，请选择以下选项：

- **JavaFX version (JavaFX 版本)**：选择与你所用的 JDK 版本相对应的版本（撰写本书时为 JavaFX 23.0.2）；^①
- **Operating System (操作系统)**：选择自己使用的操作系统。
- **Architecture (架构)**：使用 Apple M 系列处理器的 Mac 用户请选择 aarch64；使用 Intel 处理器的 Mac 用户和 Windows 用户请选择 x64。
- **Type (包类型)**：选择 SDK。

接着，请单击 Download（下载）按钮，以下载适用于你的平台的 SDK。随后会下载一个 ZIP 文件，文件名类似于：

```
openjfx-24.0.1_OS-architecture_bin-sdk.zip
```

其中，OS 是操作系统名称，而 *architecture* 是所选择的架构名称。

从 ZIP 文件中解压缩 JavaFX SDK

接着需要解压缩下载的 ZIP 文件：

- **macOS**：双击 ZIP 文件以解压缩内容。这会在“下载”文件夹中创建一个名为 `javafx-sdk-24.0.1` 的文件夹（版本号取决于下载的具体版本）。
- **Windows**：双击 ZIP 文件，然后将 `javafx-sdk-24.0.1` 文件夹拖放到“下载”文件夹中（版本号同样取决于下载的具体版本）。

为了完成安装，还必须设置 `PATH_TO_FX` 环境变量。如随后的两小节所示，具体步骤在 Windows 和 macOS 上有所区别。^②

^① 译注：中文版读者请选择 24.0.1，因为我们安装的是 JDK 24，而不是原著使用的 JDK 23。

^② 译注：原著将 JavaFX SDK 存储在“下载”文件夹的某个子文件夹中，个人认为这不是一个好的位置。请自行选择一个合适的文件夹来存储该 SDK，并相应地修改 `PATH_TO_FX` 环境变量的设置。

在 Windows 上配置 PATH_TO_FX 环境变量

在 Windows 上，请按照以下步骤操作：

1. 在任务栏的“搜索”框中输入“环境变量”；
2. 单击搜索结果中的“编辑系统环境变量”；
3. 在“系统属性”对话框中，单击“环境变量”按钮；
4. 在“用户变量”区域，单击“新建...”；
5. 在“变量名”框中输入 `PATH_TO_FX`，在“变量值”框中输入如下路径，注意将 *User* 替换为你的用户账户名称；如有必要，请将 24.0.1 替换为你实际安装的 JavaFX 版本号：`C:\Users\User\Downloads\javafx-sdk-24.0.1\lib`；
6. 单击“确定”保存环境变量；
7. 再次单击“系统属性”对话框中的“确定”按钮关闭窗口。

如果当前打开了任何命令提示符（Command Prompt）或 PowerShell 窗口，请关闭并重新打开它们，以便让新的环境变量生效。

在 macOS 上配置 PATH_TO_FX 环境变量

在 macOS 上，请按照以下步骤操作：

1. 打开“访达”（Finder），选择顶部菜单栏中的“前往”|“实用工具”，然后打开“终端.app”；
2. 在“终端”窗口中输入以下命令来判断当前使用的 shell：`echo $SHELL`；
3. 如果上一步显示的是“/bin/zsh”，那么执行：`open -e ~/.zshrc`；否则执行：`open -e ~/.bash_profile`；
4. 启动文件编辑器并打开指定的文件后，请在文件末尾添加下面这一行代码，注意将 *User* 替换为你的用户名；如有必要，请将 24.0.1 替换为你实际安装的 JavaFX 版本号：`export PATH_TO_FX="/Users/User/Downloads/javafx-sdk-24.0.1/lib"`。完成后，保存文件并关闭编辑器。

Scene Builder

在第 15 章～第 17 章的各个 JavaFX 示例中，我们将使用免费的 Scene Builder 工具，以鼠标“拖放”方式创建应用程序的图形用户界面（GUI）。要下载 Scene Builder 的安装程序，首先请访问以下网址：

<https://gluonhq.com/labs/scene-builder/>

然后，单击 Download Now（立即下载）按钮。接着，单击和自己的操作系统对应的

Download（下载）按钮。下载完成后，双击安装程序，然后按照提示完成 Scene Builder 的安装。

生成式 AI

如“前言”中的“生成式 AI 教学法”一节所述，在学习本书的过程中，你将充分利用生成式 AI。在 Deitel & Associates 公司，我们使用了多种生成式 AI，其中包括以下 4 款流行的 AI 聊天机器人：^①

- OpenAI ChatGPT (<https://chatgpt.com>)
- Google Gemini (<https://gemini.google.com/app>)
- Anthropic Claude (<https://claude.ai>)
- Perplexity (<https://perplexity.ai>)

注册账号

ChatGPT 和 Perplexity 支持免注册直接免费使用，但仅注册后才会保存历史对话记录。Google Gemini 的免费版本需登录 Google 账号方可使用；Claude 的免费版本则需要提供电子邮箱，系统会向该邮箱发送登录链接。

免费版与付费版

我们使用的是这些 AI 聊天机器人的付费版本，不过，它们均配备功能强大的免费版本，能够契合多样化的使用需求。免费版本在功能上存在一定限制，具体如下：^②

- **查询次数限制**：在特定时间段内，免费版本允许的查询次数相对较少，这可能会对需要频繁获取信息的用户造成不便；
- **提示内容限制**：免费版本对于输入的提示内容长度有所限制，难以完整、详细地表述复杂问题或需求；
- **文件上传限制**：免费版本通常允许上传的文件数量较少，甚至不支持文件上传功能，这在处理涉及文档分析等任务时功能不足；
- **响应速度受限**：相较于付费版本，免费版本的响应速度往往较慢，用户需要等待更长时间才能获得回复；
- **功能支持局限**：对于最新且最强大的功能，免费版本可能仅提供有限支持，甚至完全

^① 译注：此处列出的是本书英文版所使用的 AI。建议中文版读者选用 DeepSeek、豆包、Kimi 等易于访问的 AI。其中，DeepSeek 不仅功能强大，而且完全免费，在此特别推荐使用。

^② 译注：这些限制只是针对本书英文版使用的 AI。中文版所用的 AI 并无这些限制。

不支持，无法让用户体验到聊天机器人的全部优势。

使用 AI 聊天机器人

本书将提供数百条生成式 AI 提示（prompt），因此你会高频使用这些 AI 聊天机器人。访问每个聊天机器人的网页时，页面会显示输入框，可在其中输入提示语并提交。某些 AI 聊天机器人还支持语音输入功能。若想了解某个聊天机器人的能力，可尝试提交以下提示：

- 你能做什么？
- 你有收费版吗？和免费版有什么区别？

随后，聊天机器人会生成并显示回答。重要提醒：请务必核查生成式 AI 提供信息的准确性！

第 1 部分 导论

第 1 章 概述、Java 应用程序实操演练与生成式 AI

学习目标：

- 了解 Java 的优势；
- 初步认识 Java 应用程序编程接口（Java API），其包含大量可复用组件；
- 了解典型的 Java 程序开发环境；
- 体验 Java 开发工具包（JDK）的实操；
- 回顾互联网、Web、云计算和物联网（IoT）的基础知识；
- 了解面向对象技术的核心概念，这些概念将在“对象自然”案例学习（第 2 章～第 7 章和第 11 章）中应用，在面向对象编程章节（第 8 章～第 9 章）中深入探讨，并在后续各章中全面应用；
- 理解并发编程如何在多核处理器上实现性能最大化；
- 初步接触大数据与数据科学技术；
- 了解计算领域的前沿发展，包括元宇宙（Metaverse）、人工智能（AI）和生成式 AI（Generative AI）。

1.1 导读

欢迎来到 Java 的世界——这是全球最受欢迎且历史悠久的编程语言之一。本书将为你提供深入且注重实践的 Java 编程学习体验。

本章介绍了一系列术语和概念，为你从第 2 章开始学习 Java 编程奠定基础。我们初步介绍了面向对象编程，并讲解了全书将使用的相关术语。此外，我们还对 Java 进行了概述，讨论了 Java 应用程序编程接口（Java API）以及各种 Java 开源库——这些资源能帮助你避免“重复造轮子”。我们在 Java 开发工具包（JDK）的框架下展示了典型的 Java 程序开发环境，并通过实操一个 Java 程序（1.7 节）帮助你准备好运行书中的示例。也可以使用众多 Java 集成开发环境（IDE）中的任意一个来运行我们的程序。我们在以下网址提供了几个流行 IDE 的入门指引：^①

<https://deitel.com/javafp5>

^① 译注：Visual Studio Code 和 IntelliJ IDEA 社区版是本书中文版首选 IDE。

我们回顾了互联网、Web、云计算和物联网（IoT）的基础知识，还介绍了元宇宙及其相关技术，包括区块链、加密货币、NFT、Web3、增强现实、虚拟现实和混合现实。我们简要介绍了几种现代软件开发技术。最后，我们探讨了大数据、数据科学和人工智能（AI）技术——包括机器学习、深度学习，以及令人兴奋的生成式 AI（如 DeepSeek、OpenAI ChatGPT、Google Gemini 和 Anthropic Claude）。在本书中，你将借助这些以及其他生成式 AI 提升 Java 学习体验，并从本章最后的“专题：生成式 AI 练习”开始尝试一些非常酷的实践！

本书的架构

在开始深入学习本书之前，建议对本书的架构有一个“高屋建瓴”的认识，以掌控在学习 Java 这种大型、复杂和强大的编程语言时的方向。为此，我们建议先查看下面这些内容。

- 在“前言”部分，“高级概览”以简洁的形式展示了本书的模块化结构。“前言”还列出了本书的主要特色；
- 在本书英文版主页（<https://deitel.com/javafp5>）中，我们提供了来自 Java 专家和大学教授的书评，他们审阅过本书出版前的手稿；
- “前言”展示了本书大纲，解释了我们教授现代 Java 编程的方法。我们介绍了本书前几章采用的“对象自然”方法，它鼓励使用少量的、简单的 Java 语句来指挥强大的类执行重要的任务，同时无需创建自定义类——要一直等到第 8 章，才会开始创建自定义 Java 类。请务必阅读前言中的“本书导读”一节，它描述了每一章的重点主题，并列出了书中较为重要的案例学习和生成式 AI 练习。在阅读导读时，可以同时参阅前面所说的“高级概览”。

Java 的广泛应用

目前，全球有数十亿台个人电脑正在运行，而以计算机为核心的移动设备数量则更为庞大。据多方估算，全球约有 700 万~900 万名 Java 开发人员，他们所编写的 Java 代码运行在 600 亿个已安装的 Java 虚拟机（JVM）上。^①此外，移动设备的爆发式增长为移动应用开发创造了巨大机遇。表 1.1 列出了一些运行 Java 的设备类型。

表 1.1 运行 Java 的设备

门禁控制系统	楼宇控制系统	台式电脑
--------	--------	------

^① Oracle, “Java”, <https://www.oracle.com/java/>。

飞控系统	分线盒	电子书阅读器
自动取款机	复印机	游戏机
汽车	信用卡	GPS 导航系统
蓝光播放机	CT 扫描仪	家用电器
家庭安保系统	光学传感器	智能手机
物联网网关	停车计时器	智能笔
笔记本电脑	POS 机	平板电脑
电灯开关	打印机	逻辑控制器
机器人	电视机	医疗设备
路由器	恒温器	移动电话
服务器	交通卡	核磁共振成像设备 (MRI)
智能卡	电视机顶盒	网络交换机
智能电表	车辆诊断系统	

Java 标准版

本书第 5 版基于 Java 标准版 (Java Standard Edition, Java SE) 编写, 该版本包含运行桌面和服务器应用程序所需的全部功能。^①Java 支持如下所示的多种编程范式, 我们将在本书中逐一探讨:

- 过程式编程 (第 2 章~第 7 章)
- 面向对象编程 (第 8 章~第 10 章)
- 泛型编程 (第 12 章~第 13 章)
- 函数式编程 (第 14 章)
- 并发编程 (第 18 章)
- 生成式 AI 辅助编程 (第 19 章)

^① Oracle, “Java SE at a Glance”, <https://www.oracle.com/java/technologies/java-se-glance.html>。

Jakarta EE 和 Java ME

由于 Java 在各类应用场景中被广泛使用，因此它还有另外两个版本。其中，Jakarta EE（前身为 Java 企业版，即 Java EE）主要面向大型分布式网络应用和基于 Web 的应用程序开发，适用于支撑大型机构的核心业务系统。^①而 Java 微型版（Java ME）作为 Java SE 的子集，则专门针对资源受限的嵌入式设备应用场景，例如智能手表、医疗设备、电视机顶盒、智能电表等设备的程序开发。

1.2 硬件

由于硬件与软件技术的迅猛发展，计算成本正大幅下降。数十年前，那些可能占据一个巨大的大房间、价值数百万美元的计算机，如今已被刻蚀在比指甲还小的硅芯片上，许多芯片的成本可能仅需几美元。有意思的是，硅是地球上储量最丰富的材料之一——普通沙子中就含有制造芯片所需的硅元素。硅芯片技术让计算变得如此经济，以至于计算机和计算机化设备已成为日用品。

摩尔定律、多核处理器和并发编程

如今，许多个人电脑（PC）每秒可执行数万亿次计算^{②③}——这远超人类一生所能完成的计算量。对于大多数产品和服务，人们往往预期每年至少会有小幅涨价，但在计算机与通信领域（尤其是支撑这些技术的硬件），情况却恰恰相反。多年来，硬件成本持续快速下降。

几十年来，计算机处理能力大致每两年就以较低成本翻倍，这一显著趋势被称为“摩尔定律”（Moore's Law），其命名源于英特尔联合创始人戈登·摩尔（Gordon Moore）——他在 20 世纪 60 年代首次发现了这一规律。英特尔是当今计算机及嵌入式系统（如智能家电、家庭安防系统、机器人、智能交通路口等）处理器的主要制造商。摩尔定律及相关理论尤其适用于以下方面：

^① 译注：Java EE，Java 平台企业版（Java Platform Enterprise Edition），之前称为 Java 2 Platform, Enterprise Edition (J2EE)，2018 年 3 月更名为 Jakarta EE。

^② Tobias Mann, “Intel’s Lunar Lake Mobile Chips Boast 120 TOPS of AI Performance”, *The Register*, September 3, 2024, https://www.theregister.com/2024/09/03/intels_lunar_lake_mobile_chips/.

^③ Tobias Mann, “Apple’s M4 AI Chip: A Leap in Performance”, *The Register*, October 31, 2024, https://www.theregister.com/2024/10/31/apple_m4_ai_chip/.

- 计算机用于存储程序 and 数据的内存容量；
- 用于保存程序 and 数据的辅助存储容量；
- 处理器速度——即计算机执行程序完成任务的速度。

尽管 NVIDIA 和 ARM 的高管曾表示摩尔定律已不再适用，但英特尔首席执行官却称其“依然有效”，并指出英特尔正推进制造工艺创新，以在单个芯片上集成更多晶体管。^①如今，计算机处理能力仍在持续提升，但更依赖多核处理器等新型处理器设计。

多核处理器和性能

今天，大多数计算机都使用多核处理器，这意味着在一个集成电路芯片上集成了多个处理器核心。英特尔的一些处理器拥有多达 128 个核心。^②我们主要的测试电脑配备了 20 核 Intel Core Ultra 7 265K 处理器。苹果最新的 M4 Max 处理器则配备了 16 核 CPU、40 核 GPU 以及 16 核 NPU。AMD 甚至提供了 192 核的 CPU。^③核心数量还会继续呈增长态势。如今，用于高端游戏电脑和 AI 应用的最强大的 GPU 通常拥有数千个核心。例如，英伟达 RTX 5090 拥有 21 760 个 CUDA 核心！

在多核系统中，不同处理器可以处理任务的不同部分，从而使任务能更快地完成。为了充分利用多核架构的优势，我们需要编写多线程应用程序（第 18 章）。如果一个程序将任务分割成多个独立的线程，那么只要有足够数量的内核，多核系统就能并行运行这些线程。

随着多核系统的普及，业界对多线程技术的关注正迅速升温。Java 从诞生之初便支持多线程编程，并且多年来持续引入更高级别的功能，以简化多线程应用的开发流程。第 18 章将深入探讨如何创建与管理多线程 Java 应用程序。这一章结合大量代码示例，详细阐释了 Project Loom 带来的最新简化技术，包括虚拟线程、结构化并发以及作用域值。

多年来的计算能力发展

数据规模正日益庞大，处理数据所需的计算能力也在迅猛增长。如今，处理器性能通常以 FLOPS（floating-point operations per second，每秒浮点运算次数）衡量。20 世纪 90 年代，最快超级计算机的速度以千兆次浮点运算（gigaflops， 10^9 FLOPS）为单位；到 90 年代末，

^① Christian Hetzner, “Intel CEO Pat Gelsinger Dismisses Nvidia’s AI Chip Dominance: Moore’s Law Is Alive and Well”, *Observer*, June 2024, <https://observer.com/2024/06/intel-ceo-pat-gelsinger-nvidia-ai-chip-rivalry>.

^② 译注：英特尔至强 6（Xeon 6）处理器。

^③ 译注：旗舰级 AMD EPYC（霄龙）9965 处理器。

英特尔推出了首批万亿次浮点运算（teraflop, 10^{12} FLOPS）级超级计算机；2008年，IBM发布了首台达到千万亿次浮点运算（petaflop, 10^{15} FLOPS）级的超级计算机。如今，全球最快的超级计算机每秒可执行超过100京次（quintillion, 10^{18} FLOPS）指令！^①截至2024年11月，美国劳伦斯利弗莫尔国家实验室的El Capitan超级计算机以174.2京次/秒（1.742 exaflops）的速度位居世界第一——这意味着它能在1秒内为地球上每个人完成约2.15亿次计算。^②

目前研发中的量子计算机，理论运算速度可达现有计算机的1800京倍（18 quintillion 倍）！这一数字超乎想象：一台量子计算机1秒内完成的计算量，将超过人类有史以来所有计算设备的运算总量。2024年12月，谷歌发布的Willow量子计算芯片仅用5分钟就完成了一项任务，而传统计算机需10 septillion年（ 10^{24} 年）才能完成。这种近乎无限的计算能力可能对区块链^③加密货币（如比特币）造成冲击，工程师已开始重新思考区块链技术，并着手研发“后量子密码学”以应对未来算力的爆发式增长。^④

与此同时，计算成本持续下降，尤其是处理器架构与云计算技术的进步推动了这一趋势。过去人们常问：“本地系统需要多少算力才能应对峰值处理需求？”如今，思维已转变为：“能否在云端快速获取高负载计算任务所需的临时资源？”——用户只需为完成特定任务消耗的资源付费。

1.3 Java

微处理器革命迄今为止最重要的贡献，在于推动了个人电脑和智能手机的发展。微处理器还深刻影响了智能消费电子设备，包括数量庞大的**物联网**（Internet of Things, IoT）设备。早在20世纪90年代初，Sun Microsystems公司就意识到这一趋势，并于1991年资助了由詹姆斯·高斯林（James Gosling）领导的企业内部研究项目，最终开发出一种基于C++的

^① 译注：1京=1亿亿。

^② 译注：达到每秒100亿亿（100京）次的超级计算机称为“E级超算”。目前，我国已有“神威”E级原型机、“天河三号”E级原型机和曙光E级原型机3个不同技术路线的原型机系统完成交付。

^③ 维基百科，<https://zh.wikipedia.org/wiki/区块链>。

^④ Dr. Reji Thomas, “Quantum-Secure Blockchain: Preparing For The Era Of Quantum Computing”, November 26, 2024, <https://www.forbes.com/councils/forbestechcouncil/2024/11/26/quantum-secure-blockchain-preparing-for-the-era-of-quantum-computing/>。

面向对象编程语言——Sun将其命名为Java。^①作为世界上历史最悠久、应用最广泛的高级编程语言之一，Java具备“一次编写，到处运行”（Write Once, Run Anywhere, WORA）的特性，可用于开发适用于各种计算机系统和智能设备的程序。这种跨平台可移植性极具技术优势。

互联网的蓬勃发展使Java受到商业界的关注。如今，Java的应用场景覆盖大型企业级应用开发、Web服务器功能增强、消费电子设备（手机、智能手机、有线电视机顶盒等）应用开发、机器人软件编程等领域。它曾是开发安卓（Android）智能手机和平板电脑应用的初始语言，目前仍是Android开发的两大核心语言之一。^②2010年，Sun Microsystems被甲骨文（Oracle）公司收购。

历经30余年发展，Java已成为最常用的通用编程语言之一，因此现存大量采用旧式编程风格的代码——这类代码被称为“遗留代码”。本书将重点强调现代Java编程规范。

你可以访问以下网址，查看整理好的“史上25个最伟大的Java应用”列表：

<https://www.infoq.cn/article/btftsylfwyzdnf4vwwvpp>

此外，从本章开始，每章结尾的“专题：生成式AI练习”都整合了相关实践内容。例如，练习1.2要求通过生成式AI梳理Java在现代编程语言中仍然保持领先地位的原因。

Java的版本发布机制

自2017年Java 9发布以来，Java开始采取每6个月一次的发布节奏。新的Java版本会在每年的3月和9月发布，这样可以更快地推出新的语言和类库特性。每两年会有一个版本被指定为**长期支持**（Long-Term Support, LTS）版本。这种版本主要面向需要稳定性和长期维护支持的企业用户。非LTS版本的支持周期较短，旨在鼓励开发者尽早采用新技术和创新特性。

JEP、JSR与JCP机制解析

本书鼓励你在线研究Java技术细节，过程中可能会遇到以下常见缩写：

- **JEP (JDK 增强提案, JDK Enhancement Proposals)**: Oracle通过JEP收集Java社区对语言、API和工具改进的建议，用于规划未来Java标准版（Java SE）、企业版

^① Paul Krill, “So Why Did They Decide to Call It Java?”, *InfoWorld*, May 23, 2014, <https://www.infoworld.com/article/2170044/so-why-did-they-decide-to-callit-java-3.html>.

^② 维基百科, <https://zh.wikipedia.org/wiki/Kotlin>.

(Java EE)、微型版 (Java ME) 平台及对应 JSR 的发展路线图。要获取完整 JEP 列表, 请访问 <https://openjdk.org/jeps/0>;

- **JSR (Java 规范请求, Java Specification Requests)**: JSR 是 Java 平台功能的技术规范正式描述。每个新功能需通过 JSR 提交, 并经评审和批准流程后才能纳入 Java。要获取完整 JSR 列表请访问: <https://www.jcp.org/en/jsr/all>;
- **JCP (Java 社区流程, Java Community Process)**: JCP 负责开发 JSR。JCP 专家组创建 JSR 并公开, 以供审阅和反馈。要获取更多 JCP 信息请访问: <https://www.jcp.org/en/home/index>。

1.4 面向对象简单回顾

尽管对新型高性能软件的需求激增, 但快速、准确且经济地构建软件仍是一个难以实现的目标。**对象** (object), 或者更准确地说——如第 8 章所述——对象所源自的**类** (class), 本质上是可复用的软件组件。从日期对象、时间对象到音频对象、视频对象, 再到汽车对象、人物对象等, 不一而足。几乎任何名词都可以通过**特性** (attribute, 如名称、颜色、尺寸等名词)^①和**行为** (behavior, 如计算、移动、通信等动词) 合理地表示为软件对象。软件开发人员发现, 采用模块化的面向对象设计与实现方法, 相较于早期技术, 能显著提升开发团队的效率——这是因为面向对象程序通常更易于理解、测试、调试、修正和演进。

汽车作为对象

让我们从一个简单的类比开始。我们都知道, 开车时踩下油门踏板可以让汽车加速。但在此之前必须发生什么呢? 显然, 在驾驶汽车之前, 必须有人先完成车辆的**设计**。汽车的设计通常始于工程图纸, 类似于描绘房屋构造的**蓝图**, 这些图纸中就包含了油门踏板的设计。踏板向司机**隐藏**了使汽车加速的复杂机械结构——就像刹车踏板隐藏了减速机制, 方向盘隐藏了转向机制一样。这使得对发动机、制动和转向系统知之甚少甚至一无所知的人, 也能轻松驾驶汽车。

而要让汽车能够行驶, 还必须根据工程图纸完成**制造** (build)。一辆完工的汽车会有一个真实的油门踏板来实现加速功能, 但这还不够——汽车不会自行加速 (幸好如此!), 需要司机主动**踩下**油门踏板来指示汽车执行加速动作。

^① 译注: 在 Java 和 C#等语言中, property 和 attribute 具有不同的含义, 不要都说成“属性”。本书第 9 章会详细说明。

方法和类

让我们借助汽车的例子来介绍一些关键的面向对象编程概念。在程序中执行某项任务需要用到**方法**（method），方法中封装了完成该任务的程序语句，并对用户隐藏这些语句——这就如同汽车的油门踏板向司机隐藏了加速的机械原理。在 Java 中，我们通常会创建一个称为**类**（class）的程序单元，用于容纳执行该类任务所需的一组方法。例如，一个表示银行账户的类，必须包含向账户**存款**的方法、从账户**取款**的方法，以及**查询**当前余额的方法。类就好像是汽车的工程图纸，其中包含了油门踏板、刹车踏板、方向盘等部件的设计。

实例化

就像汽车在行驶前需要依据工程图纸完成制造一样，程序在执行类中方法定义的任务前，必须基于类创建一个对象。这个创建过程称为**实例化**，而所创建的对象则称为该类的一个**实例**。

复用

就像汽车的工程图纸能被多次复用以批量制造汽车一样，一个类也可以被多次复用，从而创建多个对象。在构建新类和程序时复用现有类，能够节省大量的时间和精力。这种复用还有助于构建更可靠、更高效的系统，因为现有的类和组件通常已经经过全面的测试、调试和性能优化。正如可互换零件的概念对工业革命至关重要，可复用类对于对象技术推动的软件革命同样意义重大。

消息和方法调用

驾驶汽车时，踩下油门踏板实际上是向汽车发送了一条指令，让它执行一个任务——也就是“加速”。类似地，在面向对象编程中，我们也会向对象发送消息。每一条这样的消息都通过调用对象的一个方法来实现，告诉该方法去完成它的任务。例如，程序可以调用某个银行账户对象的 `deposit`（存款）方法，为账户余额加上指定的金额。

特性和数据成员

汽车除了具备完成驾驶任务的能力，还拥有诸多**特性**，例如车身颜色、车门数量、油箱油量、当前时速以及总行驶里程（即里程表读数）。与驾驶能力一样，这些特性在汽车工程图纸中被作为设计的一部分明确标注（例如要求配备里程表和油量表）。驾驶过程中，这些特性会随车辆同步存在——每辆车都独立维护自身的特性状态。例如，每辆车知晓自己油箱的剩余油量，却无法得知其他车辆的油量情况。

类似地，对象在程序中使用时会携带自身特性，这些特性被定义为类的组成部分。以银

行账户对象为例，它可能包含一个表示账户余额的 `balance` 特性。每个银行账户对象仅知晓其所代表账户的余额，而不涉及银行其他账户的余额信息。对象的特性由类的**实例变量**（instance variable）来定义。

封装和信息隐藏

类将特性和方法**封装**到由其创建的对象中——对象的特性与方法是紧密关联的。对象之间可以相互通信，但通常不允许了解其他对象的内部实现细节，这些细节被隐藏在对象自身之中。正如我们将看到的，这种**信息隐藏**是良好软件工程的关键所在。

继承

通过继承，可以快速、方便地创建新类。新类接收了现有类的特征，并可对其进行自定义，从而增加自己的独特特征。在我们的汽车类比中，“敞篷车”显然也是一种“汽车”，但它的车顶可以伸缩。

接口

Java 支持**接口**（interface）——它是一组方法声明的集合，用于定义对象能执行的操作，但不规定具体实现方式。以汽车为例，若存在一个“基础驾驶功能”接口，其中包含方向盘、油门踏板和刹车踏板的操作定义，驾驶员只需掌握这套通用的转向、加速、刹车接口，即可驾驶不同类型的汽车，即便各厂商对这些系统的内部实现存在差异。

一个类可以**实现**零个或多个接口，每个接口都可以包含独立的方法，这就像汽车同时实现了“基础驾驶功能”“收音机控制”“空调控制”等不同接口。如同汽车制造商对功能的差异化实现，Java 中的类也能对接口方法进行定制化编程。例如，某软件系统定义了“备份”接口，其中包含 `save`（保存）和 `restore`（恢复）方法，不同类可根据备份内容类型（程序、文本、音频、视频等）和存储设备类型，为这些方法提供不同的实现逻辑。

面向对象分析与设计

你很快就会用 Java 语言来写程序。那么，如何为程序创建代码？也许，像许多程序员一样，你会直接启动电脑，然后直接开始敲代码。这种方法可能适合小程序（就像本书早期章节介绍的那些）。但是，如果要求你创建一个软件系统来控制一家大银行的数千台自动取款机呢？或者，如果要求你在一个由数千名软件开发人员组成的团队中工作，建立下一代空中交通管制系统呢？对于如此庞大和复杂的项目，肯定不是直接坐下来开始敲代码那么简单。

为了创建最好的解决方案，应遵循一个详细的分析过程来确定项目的需求（即定义系统应

该做什么)，并开发一个满足这些需求的设计（即决定系统应该如何做）。理想情况下，在写任何代码之前，都应经历这个过程并仔细审查设计（同时让其他软件专业人士审查这个设计）。^①如果这个过程涉及到从面向对象的角度分析和设计系统，就称其为**面向对象分析与设计**（Object-Oriented Analysis and Design, OOAD）过程。像 Java 这样的语言就是面向对象的。用这样的语言编程称为**面向对象编程**（Object-Oriented Programming, OOP）。

统一建模语言(UML)

尽管存在多种面向对象分析与设计（OOAD）流程，但已形成一种统一的图形化语言用于表达各类 OOAD 流程的结果，并得到广泛应用。**统一建模语言**（Unified Modeling Language, UML）是目前面向对象系统建模中使用最广泛的图形化表示方法。我们将在第 3 章和第 4 章首次介绍 UML 图，并在后续深入讲解面向对象编程时持续使用，直至第 10 章。

1.5 Java API 和开源库

你可以从头开始创建程序所需的每个类、接口和方法。然而，大多数 Java 程序员会利用 Java 类库中丰富的现有类、接口和方法集合（这些类库也称为 Java API，即应用程序编程接口，Application Programming Interface）。因此，学习 Java 编程实际上包含两个部分：

- 学习 Java 语言本身；
- 学习如何使用 Java API 中的类、接口和方法，以及其他公开可用的开源库中的相关组件。

使用 Java API 的类和方法而非自行编写，能够提升程序的可靠性和性能——因为这些组件经过精心设计，可确保正确运行且执行高效，同时也能缩短程序开发时间。

开源库

有大量开源 Java 库可帮助你通过编写少量代码实现重要功能。在 GitHub 上，开发者贡献了超过 27 万个 Java 代码仓库：^②

<https://github.com/topics/java>

^① 译注：软件需求是软件开发中的关键议题，译者在以下网址整理了相关书籍供拓展学习：

<https://bookzhou.com/category/软件需求/>。

^② 译注：2025 年 8 月的数据。

此外，以下页面提供了精选的流行 Java 库和资源列表，值得你浏览探索：

- <https://github.com/akullpp/awesome-java>
- <https://github.com/Vedenin/useful-java-links>
- <https://github.com/mhrimaz/AwesomeJavaFX>

Java 模块化开发方法

进行 Java 编程时，通常需要将自己的代码与以下预制模块结合使用：

- Java API 的类、接口和方法；
- 开源 Java 库的类、接口和方法；
- 你自己创建的类、接口和方法（学习第 8 章和第 9 章后）；
- 他人开发并提供的类、接口和方法。

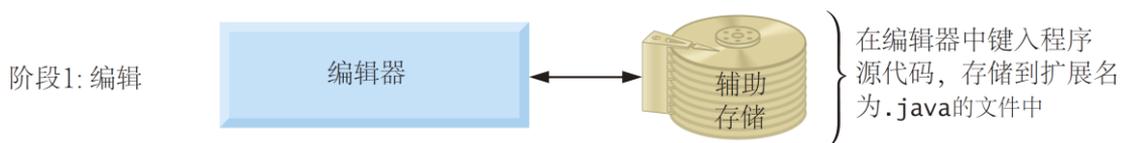
自己创建类、接口和方法的优势在于可以完全掌握其工作原理，缺点是设计、开发、调试和性能优化过程耗时较长。本书将始终使用现有的 Java API 和各类开源库，以提升编程效率并避免“重复造轮子”。这种方式称为软件复用，也是我们“对象自然”教学法的核心环节。

1.6 典型的 Java 程序开发环境

现在我们来解释创建和执行 Java 程序的步骤。通常包含 5 个阶段：编辑、编译、加载、校验和执行。我们将在 Java 开发工具包（JDK）的环境下讨论这些阶段。关于在 Windows 和 macOS 系统上下载及安装 JDK 的操作指南，请参阅前言之后的“准备工作”部分。

阶段 1：创建程序

在阶段 1，我们使用编辑器程序（简称为编辑器）编辑源代码文件。



我们可以在编辑器中键入 Java 程序的源代码，进行任何必要的更正并将其保存到系统中。Java 源代码文件名以 .java 扩展名结尾，表示该文件包含 Java 源代码。

Windows 和 macOS 分别提供了基本的编辑器“记事本”（Notepad）和“文本编辑”（TextEdit）。一些更强大的编辑器包括：

- jEdit (<https://jedit.org>): 跨平台编辑器, 用 Java 编写;
- Sublime (<https://www.sublimetext.com>): 跨平台编辑器;
- Notepad++ (<https://notepad-plus-plus.org>): 仅提供 Windows 版本;
- TextMate (<https://textpad.com>): 仅提供 macOS 版本。

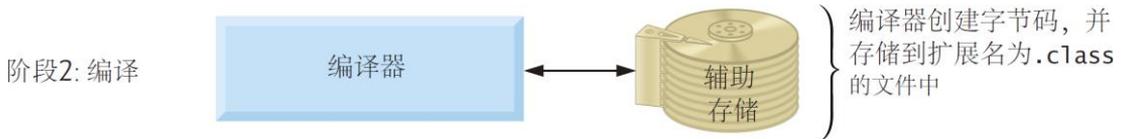
集成开发环境 (integrated development environment, IDE) 提供了支持软件开发过程的所有工具, 其中包括编辑器、用于查找导致程序执行不正确的逻辑错误的调试器等等。最流行的 Java IDE 如下所示:

- JetBrains 的 IntelliJ IDEA 社区版;
- Eclipse 基金会的 Eclipse IDE for Java Developers;
- Microsoft Visual Studio Code 跨平台代码编辑器——从技术上讲, 这是功能非常强大的一款代码编辑器, 提供了包括 Java 开发在内的许多编程语言的插件。

我们已经在“前言”后的“准备工作”部分讨论了每个工具的安装。在本书英文版主页 (<https://deitel.com/javafp5>) 上, 我们展示了如何执行本书的 Java 程序以及如何使用这些 IDE 开发新的 Java 程序。

阶段 2: 将 Java 程序编译为字节码

在阶段 2, 我们可以使用 `javac` 命令 (Java 编译器) 来编译程序。



例如, 要编译名为 `Welcome.java` 的程序, 可以在系统的命令窗口 (参见 1.7 节) 中键入:

```
javac Welcome.java
```

如果程序编译成功, 编译器会生成一个名为 `Welcome.class` 的 .class 文件。IDE 通常会为你自动执行这个操作。如果编译器检测到错误, 那么必须返回阶段 1 并进行更正。第 2 章将讨论编译器可以检测到的错误类型。

执行 `javac` 时, 如果收到诸如“错误的命令或文件名”、“`javac`:未发现命令”或“`'javac'`不是内部或外部命令, 也不是可运行的程序或批处理文件”之类的消息, 则说明 JDK 未正确安装。请仔细查看“准备工作”中的安装说明。

Java 编译器将 Java 源代码转换为**字节码** (bytecode), 字节码表示要在执行阶段 (阶段 5) 中执行的任务。**Java 虚拟机** (Java Virtual Machine, JVM) ——JDK 的一部分, 也是 Java 平台的基础——将执行字节码。**虚拟机** (Virtual Machine, VM) 是一种模拟计算机的软件, 但会向与之交互的程序隐藏底层操作系统和硬件。如果在多个计算机平台上实现了相同的

VM，那么为该类型的 VM 编写的应用程序就可以移植到所有这些平台上并运行。JVM 是使用最广泛的虚拟机之一。

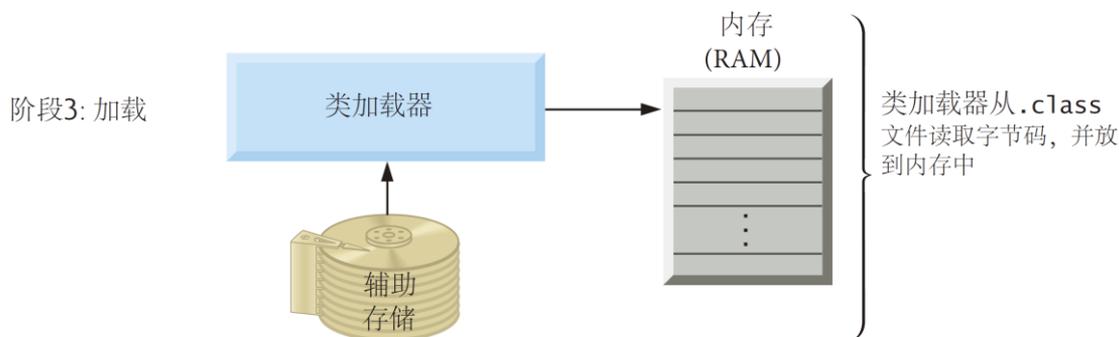
与依赖特定计算机硬件的**平台相关**机器语言指令不同，字节码指令是**平台无关**的。因此，Java 字节码具备**可移植性**（portable）——相同的字节码指令无需重新编译源代码，即可在任何包含 JVM（且该 JVM 支持编译字节码的那个 Java 版本）的平台上执行。JVM 通过 `java` 命令调用。例如，要执行名为 `Welcome` 的 Java 程序，可以在命令窗口中输入以下命令：

```
java Welcome
```

该命令会调用 JVM，由 JVM 启动执行应用程序所需的步骤，从而开始阶段 3。IDE 通常会提供如“运行”（Run）这样的菜单项或按钮，替用户完成 `java` 命令的调用操作。

阶段 3：将程序加载到内存中

在阶段 3，JVM 将程序放到内存中以执行它——这称为**加载**。

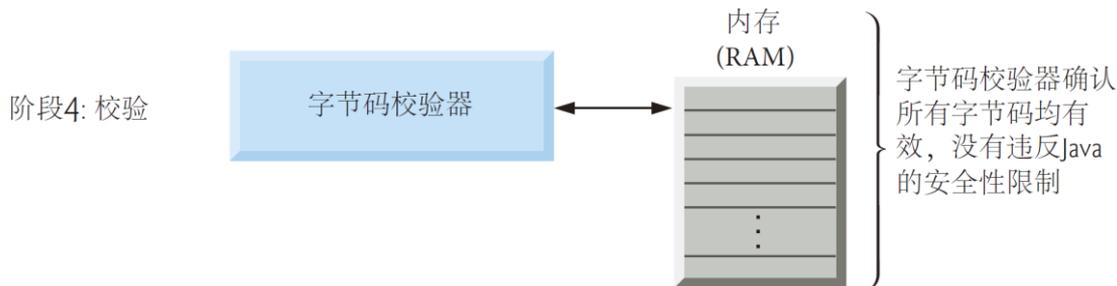


JVM 的类加载器获取包含程序字节码的 `.class` 文件，并将它们传输到 RAM。它还会加载程序中使用的由 Java 提供的任何 `.class` 文件。^①

阶段 4：字节码校验

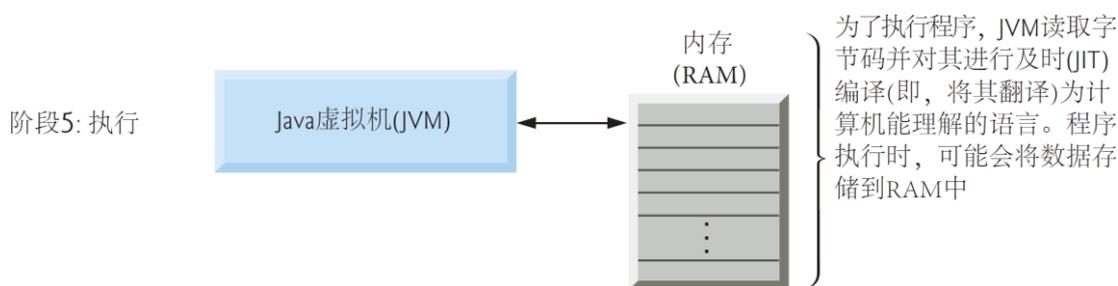
在阶段 4，当加载类时，**字节码校验器**会检查它们的字节码，以确保它们有效并且不违反 Java 的安全性限制。

^① 译注：例如，如第 2 章所述，基于本书的“对象自然”方法，会自动加载导入的任何现有的 Java 类。



阶段 5: 执行

在阶段 5, JVM 执行字节码以执行程序指定的操作。



在早期的 Java 版本中, JVM 通过解释方式执行字节码。由于 JVM 需逐条解释并执行字节码, 多数程序的运行速度较慢。而现代计算机架构已能支持多条指令并行执行。如今的 JVM 通常采用解释与**即时编译** (just-in-time, JIT) 相结合的方式执行字节码: 在解释字节码的过程中, JVM 会对其进行分析以定位“热点”——即频繁执行的字节码片段。对于这些热点代码, 即时编译器 (如 OpenJDK 的 HotSpot 编译器) 会将其转换为目标计算机的机器语言。当 JVM 再次遇到这些已编译的代码时, 便会直接执行速度更快的机器语言指令。

因此, Java 程序的执行实际上经历了两个编译阶段: ^①

- **前端编译**: 将 Java 源代码转换为字节码, 确保代码可在不同平台的 JVM 上移植;
- **即时编译 (JIT)**: 在程序运行时将热点字节码转换为目标机器的本机指令, 提升执行效率。

^① 译注: 在这一技术实现层面, 微软的 C# 语言与 Java 展现出惊人的相似性。C# 源代码首先会被编译为中间语言 (Intermediate Language, IL), 也称为 MSIL (Microsoft Intermediate Language) 或 CIL (Common Intermediate Language)。IL 与 Java 字节码类似, 属于平台无关的中间表示形式, 需通过公共语言运行时 (Common Language Runtime, CLR) 执行。如需深入理解 CLR 的运行机制, 可参阅《深入 CLR》(原《CLR via C#》) 一书, 获取更多技术细节: <https://bookzhou.com/2024/10/07/1667/>。

执行时可能发生的问题

程序可能无法一次成功运行。鉴于本书讨论的各种错误，前面的每个阶段都可能失败。例如，正在执行的程序可能会尝试除以零——这在 Java 中对于整数算术是一种非法运算。^①这将导致 Java 程序显示错误消息。如果发生这种情况，就必须回到编辑阶段，进行必要的更正，然后再次执行其余阶段，以确定更正是否解决了问题。

诸如除以零之类的错误会在程序运行时发生，因此它们被称为**运行时错误**或**执行时错误**。致命的运行时错误会导致程序立即终止，而无法成功完成其工作。非致命的运行时错误允许程序运行到完成，但通常会产生不正确的结果。

1.7 使用 JDK 运行 Java 程序

在阅读本节内容前，请先参阅前言之后的“准备工作”部分，确保已将本书示例存放至 Documents/examples 文件夹，并完成 JDK 的安装。本节将使用 JDK 提供的工具来编译并运行首个 Java 程序——一个猜数游戏。^②该程序会随机选择 1~1000 的一个数字，提示你进行猜测：若猜中则游戏结束；若猜错，程序会提示猜测值高于或低于正确数字。游戏对猜测次数不设限制，但如果掌握了二叉查找技术，那么总猜测次数不应超过 10 次。

1.7.1 使用 javac 和 java 命令编译并运行程序

本节介绍运行 Java 程序的传统方式：通过 javac 命令编译程序，再使用 java 命令运行。而 IDE 通常提供更便捷的操作方式，只需单击按钮、选择菜单项或按下键盘快捷键，即可完成编译与运行的全流程操作。

步骤 1：导航到 ch01 文件夹

首先在自己的系统上打开一个终端（命令行）窗口。

- 在 macOS 中，在“访达”（Finder）中选择顶部菜单栏的“前往”|“实用工具”，然后打开“终端.app”。
- 在 Windows 11 中，在任务栏的搜索框中输入“终端”，在搜索结果中选择“终端”以

^① 译注：在 Java 中，浮点数除以零遵循 IEEE 754 标准，结果是：正数 / 0.0 → Infinity（正无穷大）；负数 / 0.0 → -Infinity（负无穷大）；0.0 / 0.0 → NaN（Not a Number，非数字）。

^② 此处暂不讲解该 Java 程序的源代码。本节旨在演示如何编译和运行现有程序，后续章节的示例均可按此方式执行。

打开 Windows PowerShell 窗口。^①

接着，使用 `cd` 命令导航到本书示例文件夹的 `ch01` 子文件夹。

- 在 macOS 终端窗口中，输入以下命令并按 Enter 键：

```
cd ~/Documents/examples/ch01
```

- 在 Windows PowerShell 窗口（或“命令提示符”窗口）中，输入以下命令并按 Enter 键：

```
cd ~\Documents\examples\ch01
```

在后续步骤中，我们将通过 Windows PowerShell 窗口演示使用 JDK 命令行工具编译和运行程序的操作，并呈现交互过程。需要注意的是，这些操作在 macOS 和 Windows 系统是共通的。

步骤 2：编译程序

在运行猜数字游戏之前，必须使用 `javac` 命令编译其 Java 源代码，如下所示：

```
PS C:\Users\trans\Documents\examples\ch01> javac GuessTheNumberGame.java
PS C:\Users\trans\Documents\examples\ch01>
```

如果程序编译没有错误，`javac` 命令会创建一个名为 `GuessTheNumberGame.class` 的 `.class` 文件。其中包含了程序的字节码。^②

步骤 3：运行程序

要运行程序，请使用 `java` 命令，后跟 `.class` 文件的名称，但不用带 `.class` 扩展名，如下所示：

```
PS C:\Users\trans\Documents\examples\ch01> java GuessTheNumberGame
我有一个 1 到 1000 之间的数字。
你能猜出是多少吗？
请输入你的第一个猜测。
?
```

^① 译注：无论 Windows 10 还是 Windows 11，都可以按 `Ctrl+Esc`，输入 `cmd` 并按 Enter 键，在“命令提示符”窗口中执行后续操作。

^② 译注：执行 `dir` 命令，即可查看当前目录下的文件。

应用程序显示“请输入你的第一个猜测。”然后在下一行显示一个问号(?)作为提示。

步骤 4：输入第一个猜测

在提示符下，输入 **500** 并按 Enter 键，如下所示：

```
PS C:\Users\trans\Documents\examples\ch01> java GuessTheNumberGame
我有一个 1 到 1000 之间的数字。
你能猜出是多少吗？
请输入你的第一个猜测。
? 500
太小了。请再试一次。
?
```

程序显示“太小了。请再试一次。”这是因为输入的数字小于程序选择的数字。^①

步骤 5：输入另一个猜测

在下一个提示符中，输入 **750** 并按 Enter 键，如下所示：

```
PS C:\Users\trans\Documents\examples\ch01> java GuessTheNumberGame
我有一个 1 到 1000 之间的数字。
你能猜出是多少吗？
请输入你的第一个猜测。
? 500
太小了。请再试一次。
? 750
太大了。请再试一次。
?
```

应用程序显示“太大了。请再试一次。”这是因为输入的数字大于程序选择的数字。

步骤 6：输入更多猜测

继续玩游戏，直到输入正确的数字。一旦猜对，应用程序就会显示“太棒了！你猜中了这个数字！”。

```
? 625
太小了。请再试一次。
? 687
```

^① 我们在设计该程序时，故意使其每次运行都选择同一个“随机”数，这样您就能使用与我们一致的猜测逻辑。

太大了。请再试一次。

? 656

太大了。请再试一次。

? 640

太小了。请再试一次。

? 648

太大了。请再试一次。

? 644

太大了。请再试一次。

? 642

太棒了！你猜中了这个数字！

是否想再玩一次（输入 y 或 n）？

步骤 7：再次玩游戏或退出程序

猜中正确的数字后，应用程序会询问是否要再玩一次。输入 **y** 会导致应用程序选择一个新数字并开始一个新游戏。输入 **n** 会终止应用程序并返回命令窗口。

最多只需猜 10 次

猜测 1~1000 的数字最多只需 10 次尝试。若从 500 开始猜测（假设其并非正确答案），根据猜测值与正确答案的大小关系，可立即排除一半数字。例如在演示中，500 小于正确值，因此排除 1~500，仅剩余 501~1000 这 500 个数字。每次猜测应取剩余范围的中间值：下一次猜测 750 时，发现其大于正确值，于是排除 750~1000，将范围缩小至 501~749，即前一次剩余 500 个值的一半。若持续选择剩余范围的中间点，最多 10 次猜测即可将范围缩至唯一值，即正确答案。这种不断对分的技术正是经典的二叉查找算法的核心逻辑。

1.7.2 直接使用 java 命令运行程序

可以直接使用 `java` 命令运行单一源代码文件的程序，例如 1.7.1 节的猜数字游戏，如下所示：^①

```
java GuessTheNumberGame.java
```

直接执行源代码时，`java` 命令会在内存中编译程序并立即运行，与 `javac` 不同的是，该命令不会生成包含程序字节码的 `.class` 文件。

^① “JEP 330: Launch Single-File Source-Code Programs”，<https://openjdk.org/jeps/330>。

甚至可以运行多个源代码文件的程序（如第 8 章开始构建的自定义类程序）。^①此时，所有源文件会在内存中完成编译，然后执行程序。这些特性使得 Java 程序在快速测试场景下更为方便好用。

1.8 互联网、万维网、云计算和物联网

20 世纪 60 年代末，美国国防部高级研究计划局（Advanced Research Projects Agency, ARPA）推出一项计划，旨在将约 12 所受其资助的大学和研究机构的主计算机系统连成网络。这些计算机将通过通信线路连接，传输速度约为每秒 50 000 比特。对于绝大多数仍在通过电话线以每秒 110 比特的速度连接计算机的人来说，这一速度堪称惊人。学术研究由此迎来重大突破。ARPA 实施的网络很快被称为**阿帕网**（ARPANET），即互联网的前身。

如今，最快的民用互联网速度已达每秒数十亿比特（千兆比特）。当前速度纪录由日本国家信息通信技术研究所保持，其下载速度达到每秒 402 太比特（万亿比特），约为美国普通民用每秒 250 兆比特宽带速度的 160 万倍。^②

事情的发展与最初的计划大相径庭。尽管阿帕网让研究人员实现了计算机联网，但其主要价值却体现在通过后来被称为电子邮件（e-mail）的方式实现快速通信。即便在今天的互联网中，电子邮件、即时消息、文件传输和社交媒体（如微信、YouTube、Facebook、WhatsApp、Instagram、抖音、Reddit 和 X）仍让全球数十亿人得以便捷沟通。

阿帕网中计算机通信遵循的协议（规则集）被称为**传输控制协议**（Transmission Control Protocol, TCP）。它确保由顺序编号的“数据包”组成的消息能从发送方正确传递至接收方，保证数据完整到达并按序重组。

1.8.1 互联网：网络的网络

在阿帕网发展的同时，世界各地的组织纷纷建立自有网络，用于内部及组织间通信，催生了多样化的网络硬件与软件。当时的一大挑战是如何实现不同网络间的互通信——ARPA 通过开发**互联网协议**（Internet Protocol, IP）解决了这一问题。该协议创建了“网络的网络”（network of networks）架构，即当今互联网的基础形态。这组协议组合现称为 TCP/IP。每台接入互联网的设备都拥有唯一的 IP 地址，作为 TCP/IP 通信中设备间定位的数字标识。

企业很快意识到可以借助互联网优化运营并提供更优质的客户服务，因此投入巨资拓展和

^① “JEP 458: Launch Multi-File Source-Code Programs”，<https://openjdk.org/jeps/458>。

^② Paulius Grinkevičius, “Japan Breaks Internet Speed Record, Hitting 402 Terabits per Second”, *CyberNews*, July 4, 2024, <https://cybernews.com/tech/internet-connection-speed-world-record>。

强化网络布局。这一趋势推动通信运营商、硬件及软件供应商展开激烈竞争，以满足激增的基础设施需求。其结果是：互联网带宽（通信线路的信息承载能力）大幅提升，而硬件成本则急剧下降。

1.8.2 万维网：使互联网更好用

万维网（World Wide Web，简称 WWW 或 Web）是与互联网相关的软硬件集合，它支持计算机用户查找和浏览包含文本、图形、动画、音频及视频等多元内容的文档。1989年，欧洲核子研究中心（CERN，European Organization for Nuclear Research）的蒂姆·伯纳斯-李（Tim Berners-Lee）开始研发**超文本标记语言**（HyperText Markup Language，HTML）——这是一种通过“超链接”的文本文档实现信息共享的技术。他还编写了**超文本传输协议**（HTTP，HyperText Transfer Protocol）等通信协议，构成其新型超文本信息系统的核心架构，并将该系统命名为“万维网”。

1994年，伯纳斯-李创立了**万维网联盟**（W3C，World Wide Web Consortium），该组织致力于 Web 技术的开发。W3C 的核心目标之一是确保 Web 对所有人普遍可用，无论其能力、语言或文化背景如何。

1.8.3 云计算

如今，越来越多的计算任务正在从本地计算机和移动设备转移到“云”，这就是所谓的**云计算**。它借助分布在全球互联网上的远程服务器集群来运行软件和处理数据。云计算的核心优势之一是其弹性。它允许用户根据实际需求，随时、动态地增加或减少计算资源。与自行购置并维护大量硬件以应对偶发的峰值需求相比，这种按需付费的模式能极大地节省存储与计算成本，具备更高的成本效益。此外，云计算还将管理计算资源的繁重工作，如软件安装与升级、安全防护、数据备份及灾难恢复等，都转移给了专业的服务提供商。这不仅进一步节省了资金，更重要的是降低了企业的运维成本和技术门槛。

在日常生活中，人们使用的各类应用程序，在很大程度上依赖基于云的服务。这些服务依托大规模的计算资源集群（涵盖计算机、处理器、内存、存储等）以及数据库，它们通过互联网彼此通信，并与用户使用的应用程序进行数据交互。通过互联网提供自身访问权限的服务被称为**Web 服务**。截至 2025 年 8 月，在云服务市场中，亚马逊 Web 服务（AWS，市场份额达 30%）、微软 Azure（占 20%）、谷歌云平台（GCP，占 13%）以及阿里云（占 4%），这四家总共占据了 67% 的份额。^①

^① Statista, “Worldwide Market Share of Leading Cloud Infrastructure Service Providers”, November 1, 2024, <https://www.statista.com/chart/18819/worldwide-marketshare-of-leading-cloud-infrastructure-service-providers/>.

1.8.4 物联网

互联网已不仅是计算机的网络，更是一个**物联网**（Internet of Things, IoT）。“物”是指任何拥有 IP 地址、能通过互联网自动发送（有时也接收）数据的设备。这类“物”包括：

- 配备通行费支付转发器和导航 GPS 的汽车；
- 自主机器人；
- 监控车库车位可用性的监测器；
- 植入人体的心脏监护仪；
- 监测能源使用、天气、水质和交通状况的智能传感器；
- 烟雾和一氧化碳探测器；
- 辐射探测器；
- 仓库中的物品追踪器；
- 可跟踪用户移动轨迹的移动应用程序；
- 根据天气预报和家庭活动自动调节室温的恒温器；
- 智能手表、健身追踪器、智能眼镜等可穿戴设备；
- 智能家电、照明系统、门铃和门锁；
- 以及其他更多设备。

预计到 2025 年，活跃的物联网设备将达 416 亿台。^①

1.8.5 边缘计算

许多物联网设备应用于**边缘计算**（edge computing）场景，其核心特征是数据处理、分析和存储发生在数据生成设备上或附近，即网络的边缘地带。^②这种模式“支持更快的处理速度与实时决策，这对需要即时响应的应用至关重要，例如物联网设备、自动驾驶汽车或边缘计算机”。^③以自动驾驶汽车为例，即便失去互联网连接，仍可通过边缘计算能力做出确保安全驾驶的决策。

边缘计算的应用领域极为广泛，涵盖医疗保健、制造业、零售业、清洁能源、智能电网、交通运输、自动驾驶汽车、智慧城市与基础设施、工业自动化与过程控制、国防、航空航

^① Dell Technologies, “Internet of Things and Data Placement”, <https://infohub.delltechnologies.com/en-us/l/edge-to-core-and-the-internet-of-things-2/internet-of-things-and-data-placement/>.

^② 维基百科, <https://zh.wikipedia.org/zh-hans/边缘计算>。

^③ Advantech, “Edge Computing: Enhancing Real-Time Data Processing and Decision-Making”, September 26, 2024, <https://www.advantech.com/en-us/resources/industry-focus/edge-computing/>.

天、农业、游戏、虚拟现实、视频监控等诸多领域。^{①②}

1.8.6 混搭

混搭 (Mashup)^③作为一种应用开发方法，允许开发者通过组合（通常免费的）互补 Web 服务与各类信息源，快速构建出功能强大的软件。早期混搭案例之一的 housingmaps.com，曾将 craigslist.org 的房产列表与谷歌地图结合，用于展示特定区域内待售或出租房屋的位置。

本书“对象自然”方法的核心重点之一是从第 2 章开始的大规模软件复用——读者将使用易用且功能强大的现有类库完成关键任务。同理，在构建自有 Web 服务前，建议优先复用互联网上大量现成（且多为免费）的强大服务。以下在线目录收录了数十到数千个可以直接使用的 Web 服务：

- **GitHub 公共 API 页面** (<https://github.com/public-apis/public-apis>)：涵盖 51 个类别，罗列数百个公开可用的 Web 服务；
- **Google APIs Explorer** (<https://developers.google.com/apis-explorer>)：无需编写代码即可了解并试用近 300 个 Google Web 服务；
- **Free Public APIs** (<https://www.freepublicapis.com>)：收录数百个免费公开 API；
- **Rapid** (<https://rapidapi.com>)：号称全球最大的 API 中心，列出了数千个免费及付费 API，同时提供工具帮助企业构建和分发自有 API。

1.9 元宇宙

元宇宙 (Metaverse) 一词最早出现于尼尔·斯蒂芬森 (Neal Stephenson) 1992 年的小说《雪崩》(Snow Crash) 中。^④书中主角弘 (Hiro) “存在于一个电脑生成的宇宙中，他的电脑通过护目镜绘制出这个宇宙，并通过耳机将其传输给他……这个虚构的空间被称为元宇宙。”^⑤高德纳咨询公司 (Gartner) 指出，元宇宙是由虚拟技术增强的物理和数字现实融

^① Corvalent, “7 Essential Edge Computing Use Cases for Today's Industries”, <https://corvalent.com/news/7-essential-edge-computing-use-cases-for-industries/>。

^② Daffodil Software, “Top 24 Industrial Applications of Edge Computing”, April 3, 2023, <https://insights.daffodilsw.com/blog/applications-of-edge-computing>。

^③ 译注：Mashup 应用在业界有多种译法，包括混搭应用、混合应用、混聚应用等。

^④ 译注：豆瓣读书，《雪崩》，<https://book.douban.com/subject/3816895/>。

^⑤ 维基百科，<https://zh.wikipedia.org/wiki/元宇宙>。

合而成的集体虚拟空间。”^①本节将介绍几种关键的元宇宙技术，1.13 节则会探讨人工智能（AI）——它通过辅助生成逼真的数字替身、环境、自然语言对话等内容，提升元宇宙的沉浸感。

1.9.1 虚拟现实(VR)

虚拟现实（Virtual Reality, VR）通常指模拟的三维（3D）环境，用户可通过以下方式与之互动：

- **非沉浸式 VR**：借助电脑屏幕在二维空间中观看 3D 环境；
- **半沉浸式 VR**：利用大屏幕电视或视频墙等大型显示器，在二维空间中观看 3D 环境；
- **完全沉浸式 VR**：用户通过头戴式显示器（如 VR 头显）和手持控制器等设备与 3D 环境互动，产生身临其境的代入感。

主流虚拟现实设备包括 Meta Quest 3、Meta Quest Pro、索尼 PlayStation VR2 和 HTC Vive XR Elite 等。

1.9.2 增强现实(AR)

增强现实（Augmented Reality, AR）通过智能手机、平板电脑、智能眼镜等内置摄像头的设备，将实时生成的图像、声音、视频、动画等计算机内容叠加到现实场景中，实现虚拟与现实的同步呈现。

AR 技术最知名的案例之一是热门手机游戏《宝可梦 Go》（Pokémon Go）。在游戏中，宝可梦角色会被投射到用户通过移动设备 GPS 定位的现实世界地点。该游戏自发布十年以来，其开发商 Niantic 仍在持续探索新玩法以维持玩家活跃度。^②

下面列出了 AR 的其他应用场景。

- **体育赛事**：在棒球、高尔夫、冰球等赛事直播中，通过显示轨迹线帮助观众追踪球的运动路径；美国国家橄榄球联盟（National Football League, NFL）直播会在球场上叠加彩色线条，标注争球线和首攻所需到达的位置。
- **导航领域**：导航应用在实景地图和街景上叠加实时导航与交通信息。
- **家居装饰**：通过移动应用预览房间粉刷或添置家具后的效果。
- **零售行业**：服装零售商借助 AR 技术展示顾客试穿特定服饰的虚拟形象。
- **文化展览**：博物馆通过显示展品相关信息，增强展览的互动性。

^① “什么是元宇宙”，<https://www.gartner.com/cn/information-technology/articles/what-is-a-metaverse>。

^② “MLB 2025”，*Pokémon GO Live*, February 12, 2025, <https://pokemongolive.com/post/mlb-2025>。

目前，多家企业已推出或正在研发 AR 眼镜及头显设备。^{①②}谷歌、三星和高通正合作开发 Android XR——这是一款融合数字与物理世界的安卓移动操作系统扩展，配套的计算机芯片及 AI 驱动的眼镜、头显设备也在同步研发中。^③

1.9.3 混合现实(MR)

混合现实（Mixed Reality）与增强现实（AR）原理相似，但更进一步：它允许用户与投射到真实视野中的虚拟对象进行交互，从而融合了 AR 与 VR 的技术特性。与 VR 类似，优质的混合现实体验通常需要头戴式设备支持。而**扩展现实**（Extended Reality, XR）是一个更广泛的概念，涵盖了虚拟现实（VR）、增强现实（AR）和混合现实（MR）的全部范畴。

1.9.4 区块链技术

会计系统通过账本记录交易，而**区块链**（blockchain）^{④⑤}是一种基于软件的分布式账本，数据存储在互联网节点中。它将交易信息加密封装成**区块**（block），区块按时间顺序链接，新块始终添加至链尾。每个区块创建后**不可篡改**（immutable），因此区块链构成了永久的交易记录。由于区块链采用去中心化架构，没有单一主体拥有控制权——这种特性为**去中心化应用**（decentralized apps, dApps）奠定了基础，也是元宇宙中数字所有权的技术核心。

区块链是近年来热门数字技术的底层支撑，包括加密货币（1.9.5 节）和非同质化代币（1.9.7 节）等。其应用场景还包括：^⑥

^① Will Greenwald, “The Best Smart Glasses”, *PCMag*, January 16, 2025, <https://www.pcmag.com/picks/the-best-smart-glasses>.

^② Prakhar Khanna and Kerry Wan, “CES 2025: The 8 Most Advanced Smart Glasses We Tried and Were Impressed By”, *ZDNet*, January 10, 2025, <https://www.zdnet.com/article/ces-2025-the-8-most-advanced-smart-glasses-we-tried-and-were-impressed-by/>.

^③ Scott Stein, “I Tried Google and Samsung's Next-Gen Android XR Headsets and Glasses, and the Killer App Is AI”, *CNET*, December 12, 2024, <https://www.cnet.com/tech/computing/i-tried-google-and-samsungs-next-gen-android-xr-headsets-and-glasses-and-the-killer-app-is-ai/>.

^④ 维基百科, <https://wuu.wikipedia.org/wiki/区块链>。

^⑤ Adam Hayes, “Blockchain Facts: What Is It, How It Works, and How It Can Be Used”, September 16, 2024, <https://www.investopedia.com/terms/b/blockchain.asp>.

^⑥ Sam Daley, “33 Blockchain Applications and Real-World Use Cases”, March 2, 2023, <https://builtin.com/blockchain/blockchain-applications>.

-
- **去中心化金融 (decentralized finance , DeFi)**：如借贷、资金转移等应用；
 - **智能合约**：自动化执行的合约协议；
 - **更安全的物联网 (IoT) 应用**：提升设备交互安全性；
 - **数字身份认证**：防范欺诈的安全身份体系；
 - **医疗信息管理**：加密共享医疗数据；
 - **商业物流追踪**：供应链全流程溯源；
 - **知识产权保护**：数字内容确权。

1.9.5 比特币与加密货币

“加密货币是通过密码学技术保护的数字或虚拟货币。”^①这类资产存储在数字钱包中，通常依托区块链技术实现安全、去中心化的交易处理。2009 年诞生的**比特币** (Bitcoin) 是最早、最知名且市值最高的加密货币，它基于区块链构建，采用去中心化架构设计。

1.9.6 以太坊平台

以太坊 (Ethereum) 是可编程的区块链平台，支持在其网络上构建和部署去中心化应用 (dApps) —— 这些应用利用区块链存储数据或控制业务逻辑。其核心特性 “智能合约” 可用于实现借贷平台、去中心化交易所、保险产品、众筹系统等场景。以太坊的原生加密货币是以太币 (Ether)，市值位居全球第二。

1.9.7 非同质化代币 (NFT)

非同质化代币 (non-fungible token, NFT) 是基于区块链的加密唯一通证，与数字或物理内容绑定，提供所有权证明。其应用场景包括：数字艺术品、限量收藏品、游戏与元宇宙中的虚拟资产、实体房地产、知识产权、数字图书等。

以下是一些知名的 NFT 艺术品交易案例：^②

- **最贵集体购买 NFT**：艺术家 Pak^③ 的《吞并》 (The Merge) 以 9180 万美元被 29 983 名

^① Jake Frankenfield, “Cryptocurrency Explained With Pros and Cons for Investment”, February 4, 2023, <https://www.investopedia.com/terms/c/cryptocurrency.asp>。

^② Casey Anderson, “Most Expensive NFT,” NFT Evening”, December 30, 2024, <https://nftevening.com/most-expensive-nft/>。

^③ 译注：匿名艺术家 Pak 是业界享有盛誉的动态特效设计师，也是其创办的 Undream 工作室的创始人和 “AI 策展活动 Archillect” 的首席设计师。他曾与数百个大品牌和工作室合作，有超过 25 年的数字艺术创作经验，这使得 Pak 成为了加密艺术界的翘楚。

买家认购，每人持有部分权益；

- **最贵单买家 NFT**：数字艺术家 Beeple^①的《每一天：前 5000 天》（*Everydays: The First 5000 Days*）以 6930 万美元成交。

1.9.8 Web3

如同元宇宙，Web3 被视作互联网的下一代形态。^②两者因共享部分技术常被混淆，《福布斯》（*Forbes*）将 Web3 定义为“去中心化互联网——基于区块链等分布式技术构建，而非集中在私人或企业服务器上”。^③其核心愿景是：通过区块链及其关联技术（如 NFT、加密货币、智能合约），赋予用户对数据和在线交互的更大控制权。

1.10 软件开发技术

在软件开发过程中，经常会遇到以下流行术语：

- **重构 (Refactoring)**：在保持程序功能正确性的前提下，对代码进行重组优化，使其更清晰易维护，有时还能提升性能。许多集成开发环境（IDE）内置自动重构工具，可批量完成代码重写工作；
- **设计模式 (Design Pattern)**：构建灵活可维护的面向对象软件的一些成熟模式。该领域致力于梳理这些重复出现的结构，鼓励开发者复用模式，以更高效地开发高质量软件，节省资金、时间和精力；
- **软件开发工具包 (Software Development Kit, SDK)**：开发者用于编写应用的工具、库和文档集合，可帮助创建针对特定操作系统、游戏平台、云平台等的应用程序。例如，Java 开发工具包（Java Development Kit, JDK）即为典型的 SDK；
- **版本控制系统 (Version Control System, VCS)**：用于管理和追踪软件开发过程中代码变更的工具。Git 作为 GitHub 的底层技术，是目前使用最广泛的版本控制系统，93% 的开发者都在使用；
- **敏捷软件开发 (Agile Software Development)**：采用小步快跑的开发模式，开发团队根据用户在软件未完成时的反馈持续优化。这种方法帮助团队在需求或目标变化时快速适应；

^① 译注：Beeple，本名为迈克·温科尔曼（Mike Winkelmann），是来自美国南卡罗来纳州的数字艺术家。

^② David Essex, Sean Michael Kerner, and Alexander S. Gillis, “What Is Web 3.0 (Web3)? Definition, Guide and History”, *TechTarget*, <https://www.techtarget.com/whatis/definition/Web-30>。

^③ Bernard Marr, “The Important Difference Between Web3 And The Metaverse”, <https://www.forbes.com/sites/bernardmarr/2022/02/22/the-important-difference-between-web3-and-the-metaverse/>。

-
- **测试驱动开发 (Test-Driven Development , TDD)**: 敏捷开发的重要实践, 开发者先编写测试用例, 再进行代码开发, 并在过程中持续验证代码通过所有测试, 确保软件满足系统需求;
 - **微服务 (Microservice)**: 一种软件架构风格, 将应用拆分为多个小型独立服务协同工作, 每个服务专注于特定功能, 通常以 Web 服务形式实现。例如, 网飞 (Netflix) 流媒体服务通过微服务处理视频播放、推荐、内容发现、用户认证、计费等功能。尽管网飞未开放微服务接口, 但其在开源平台 (<https://netflix.github.io>) 公开了许多软件和架构设计;
 - **容器化 (Containerization)**: 将软件打包为“容器”的技术。这里的“容器”是一种轻量级可移植单元, 包含代码、库和依赖项等运行所需资源。容器化通过虚拟化技术屏蔽底层平台差异, 确保容器在不同环境中一致运行。Docker 是目前最主流的容器化平台。

1.11 数据分析与数据科学

本节的内容是由我们向生成式 AI 提问“什么是数据分析与数据科学”后, 综合回答并修改而成, 主要参考了 DeepSeek 的回答。

统计学

统计学 (Statistics) 是数据分析和数据科学的根基, 它提供了一套系统的方法来收集、整理、分析和解释数据。通过统计方法, 我们可以计算数据的集中趋势 (如均值、中位数)、离散程度 (如方差、标准差), 并进行假设检验来判断数据间的显著性差异。无论是简单的业务分析还是复杂的机器学习模型, 都依赖于统计学原理来确保结论的可靠性。

数据分析

数据分析 (data analytics) 的主要目标是从现有数据中提取有价值的见解, 帮助企业和组织做出更明智的决策。它侧重于回答具体的业务问题, 比如“上季度销售额下降的原因是什么?” 或“哪些因素影响了客户满意度?” 常用的技术包括描述性统计 (如数据汇总)、探索性数据分析 (EDA) 以及数据可视化 (如图表、仪表盘)。工具方面, 数据分析师通常使用 Excel 进行初步处理, SQL 查询数据库, Python (Pandas、Matplotlib) 或 R 进行更深入的分析, 以及 BI 工具 (如 Tableau、Microsoft Power BI) 来呈现结果。

数据科学

数据科学 (data science) 的涵盖面比数据分析更广, 它结合了统计学、计算机科学和领域专业知识, 旨在通过算法和模型从数据中预测未来趋势或实现自动化决策。数据科学家不

不仅要分析数据，还要构建和优化机器学习模型，处理大规模数据集（包括结构化数据和非结构化数据，如文本、图像）。常见应用包括推荐系统、自然语言处理（NLP）、计算机视觉和金融风控等。技术栈通常包括 Python（Scikit-learn、TensorFlow/PyTorch）、大数据工具（如 Spark、Hadoop）以及云计算平台（如 AWS、GCP）。

数据分析与数据科学的区别

数据分析与数据科学具有以下区别：

- **目标不同**：数据分析侧重于解释历史数据，回答“发生了什么”和“为什么发生”；数据科学则聚焦预测未来，解决“将会发生什么”和“如何优化”的问题；
- **方法不同**：数据分析主要依赖统计分析和可视化；数据科学则使用机器学习、深度学习和高级算法建模；
- **数据规模不同**：数据分析通常处理中小型结构化数据；数据科学常涉及大规模、复杂的数据（如文本、图像、视频）；
- **输出不同**：数据分析产出报告、仪表盘；数据科学产出预测模型或 AI 应用；
- **技能要求不同**：数据分析需要 SQL、Excel 和基础统计；数据科学要求编程、算法和数学建模能力。

尽管两者有区别，但在实际工作中往往紧密协作：数据分析发现问题，数据科学提供解决方案。

1.12 大数据到底有多大？

我们常听到“大数据”的说法，但有没有想过具体有多“大”？全球数据量正以惊人速度增长，而大数据应用处理的正是这些海量的数据。表 1.2 总结了常见的大数据字节单位体系。

表 1.2 大数据存储单位对照表

单位	符号	字节换算	直观类比（近似值）
千字节	KB	$1024^1 = 1024$ 字节	简单文本文件（约 2 页文档）
兆字节	MB	$1024^2 = 1\,048\,576$ 字节	标准音乐文件（约 1 首 MP3）
吉字节	GB	$1024^3 = 1\,073\,741\,824$ 字节	高清电影（约 1 部）

			1080P 影片)
太字节	TB	$1024^4 = 1\,099\,511\,627\,776$ 字节	大型图书馆藏书数字化 (约 500 万册文本)
拍字节	PB	$1024^5 = 1\,125\,899\,906\,842\,624$ 字节	全球学术论文数据 (约 10 亿篇文献)
艾字节	EB	$1024^6 = 1\,152\,921\,504\,606\,846\,976$ 字节	2022 年全球互联网月均流量 (约 50 万部 4K 电影)
泽字节	ZB	$1024^7 = 1\,180\,591\,506\,881\,476\,736\,000$ 字节	人类历史语音数据 (约 1000 亿小时通话)
尧字节	YB	$1024^8 = 1\,208\,925\,819\,614\,629\,174\,706\,176$ 字节	全球传感器年数据量 (2025 年预计值)

对于计算机科学家和数据科学家而言，掌握数据的获取、存储、处理及价值提取方法，与编写高效程序同等重要——在生成式 AI 蓬勃发展的当下，这一能力甚至更为关键。

预计到 2025 年底，全球每天将产生 182 泽字节 (ZB) 的数据，约合 1.82×10^{23} 字节 (18.2 万艾字节，1 艾字节 = 10^{18} 字节)，而这一数字到 2028 年预计将翻倍还要多。^①互联网是这类数据的主要来源之一，后续小节描述了现实生活中常见数据单位的具体示例。

兆字节(MB)

1 兆字节 (megabyte, MB) 大约为一百万字节 (2^{20})。高质量 MP3 格式的音频文件每分钟需要 1MB~2.4MB。数码相机拍摄的 JPEG 格式照片每张大约需要 3MB~7MB。在 iPhone 上，1080p 视频以每秒 30 帧 (30 FPS) 的速度录制，每分钟需要 65MB，而 4K 视频以每秒 60 帧 (60 FPS) 的速度录制，则每分钟需要 440MB。

吉字节(GB)

1 吉字节 (gigabyte, GB) 等于 1024 兆字节 (2^{30} 字节)。一张 DVD 最多可存储 8.5 GB 的数据，可容纳 141 小时的 MP3 音频、数百张来自 1600 万像素相机的照片、7.7 分钟的 1080p

^① Statista, “Volume of Data/Information Created, Captured, Copied, and Consumed Worldwide from 2010 to 2023, with Forecasts from 2024 to 2028 (in Zettabytes)”, <https://www.statista.com/statistics/871513/worldwide-data-created/>.

视频（30FPS）或 2.33 分钟的 4K 视频（60FPS）。目前最高容量的超高清蓝光光盘（Ultra HD Blu-ray）最多可以存储 100 GB 的视频。流媒体播放高度压缩的 4K 电影，每小时可能消耗 7 GB~10 GB 的流量。

太字节(TB)

1 太字节（TB）等于 1024 吉字节（ 2^{40} 字节）。当前台式机硬盘的最大容量可达 36 TB，可存储 50.4 年的 MP3 音频、数十万张来自 1600 万像素相机的照片、131.4 小时的 1080p 视频（30 FPS）或 39.7 小时的 4K 视频（30 FPS）。

拍字节、艾字节和泽字节

2021 年，全球有近 50 亿人在线，每天产生了约 2.5×10^{18} 字节（即 2500PB 或 2.5EB）的数据。^①预计到 2025 年，活跃的物联网设备将达到 416 亿台，^②全球数据总量预计将达 163 泽字节（ZB）——相当于地球上每个人每分钟产生约 360MB 的数据。这相当于每秒新增 51.6PB 数据、每分钟 3100PB、每小时 186EB、每天约 446EB（接近半个泽字节）的数据。这意味着每天会产生数十亿小时的 4K 视频或数万亿张照片！

其他大数据统计数据

想要实时感受大数据的规模，可以访问 <https://www.worldometers.info>，查看包括当天 Google 搜索次数、推文（现称为 X 平台上的“帖子”）数量和电子邮件发送量等各类统计数据。Domo 公司的图表信息页面“数据永不眠”（Data Never Sleeps）展示了 2022 年每分钟生成的数据量（这些数字自那以后仍在持续上升），包括：

- Instagram 上发布 6.6 万张照片；
- Facebook 发布 170 万条动态；
- Snapchat 上分享 243 万段快照（snap）；
- 流媒体视频观看时长 100 万小时；
- Zoom 会议累计时长 10.5 万小时。

^① Aditya Rayaprolu, “How Much Data Is Created Every Day in 2025?”, <https://techjury.net/blog/how-much-data-is-created-every-day>。

^② Dell Technologies, “Internet of Things and Data Placement”, <https://infohub.delltechnologies.com/en-us/1/edge-to-core-and-the-internet-of-things-2/internet-of-things-and-data-placement/>。

处理全球数据需要大量电力

2017年的一篇文章指出，处理全球数据的能源消耗正以每年 20% 的速度增长，约占全球用电量的 3%~5%。^①麻省理工学院斯隆管理学院最近的研究显示，到 2030 年，数据中心的耗电量可能占全球电力消耗的 21%。^②

另一大电力消耗源是基于区块链的加密货币比特币（Bitcoin）。仅一次比特币交易的耗电量，就相当于美国家庭一周的平均用电量。这种高能耗源于比特币“矿工”验证交易数据有效性的过程。尤其考虑到人们对区块链应用（不仅限于加密货币）的巨大兴趣，当前的能源消耗水平已难以持续。目前，区块链社区正致力于改进这一问题。此外，生成式 AI 也是高能耗应用领域之一。例如，ChatGPT 每天的能耗相当于为 18 万户美国家庭提供电力支持。^③

1.12.1 大数据分析

“数据分析”一词最早出现于 20 世纪 60 年代，不过人类运用统计学分析数据的历史可追溯至几千年前的古埃及时期。^④“大数据”（big data）这一概念则约在 1987 年被正式提出。

大数据的四个核心“V”特征如下：

1. **Volume (体量)**：全球数据正以指数级规模增长；
2. **Velocity (速度)**：数据生成、组织内流转及更新的速度均在快速提升；
3. **Variety (多样性)**：如今的数据不仅包含字符、数字、标点和特殊符号，还涵盖图像、音频、视频，以及家庭、企业、车辆、城市中日益增多的物联网传感器所产生的数据；
4. **Veracity (真实性)**：数据是否有效？是否完整准确？在做出关键决策时能否信赖这

^① “Tsunami of data could consume one fifth of global electricity by 2025”，December 11, 2017, <https://www.theguardian.com/environment/2017/dec/11/tsunami-of-data-could-consume-fifth-global-electricity-by-2025>。

^② Beth Stackpole, “AI Has High Data Center Energy Costs — but There Are Solutions”，MIT Sloan School of Management, January 7, 2025, <https://mitsloan.mit.edu/ideas-made-to-matter/ai-has-high-data-center-energy-costs-there-are-solutions>。

^③ Gordon, Cindy, “ChatGPT and Generative AI Innovations Are Creating Sustainability Havoc”，*Forbes*, March 12, 2024, <https://www.forbes.com/sites/cindygordon/2024/03/12/chatgpt-and-generative-ai-innovations-are-creating-sustainability-havoc/>。

^④ “A Brief History of Data Analysis”，December 7, 2021, <https://www.flydata.com/blog/a-brief-history-of-data-analysis/>。

些数据？其真实性如何？

数字数据存储技术已发展到容量极大、成本低廉且体积小巧的程度，使得我们能够便捷且经济地保留所有生成的数字数据——这才是真正意义上的“大数据”。

1.12.2 数据科学与大数据正在改变世界：应用场景

数据科学领域正快速发展，因其产生的成果在个人生活与商业场景中均产生重要影响。表 1.3 列举了诸多数据科学与大数据的应用案例。实践表明，大数据分析已助力企业提升利润、改善客户关系，甚至让全球多支体育团队赢得更多比赛与冠军头衔。在美国国家橄榄球联盟（NFL）的职业赛事直播中，解说员常提及教练团队借助数据分析进行四档进攻决策。^①

表 1.3 数据科学的示例应用场景

协助残障人士	人类基因组测序	乘车共享
自动关闭字幕	身份盗窃预防	机器人理财顾问
脑部成像	改进库存控制	自动驾驶汽车
癌症诊断/治疗	智能助手	情感分析
计算机视觉	基于位置的服务	相似度检测
犯罪预防	恶意软件检测	智能家居
CRISPR 基因编辑	市场营销分析	智能电表
作物产量提升	医疗设备监控	智能恒温器
客户服务代理	自然语言翻译	智能交通控制
数据可视化	个性化医疗	垃圾邮件检测
诊断医学	钓鱼诈骗消除	股票市场预测

^① 译注：在美式橄榄球（NFL）比赛中，第四档进攻决策指球队在最后一次进攻机会（4th Down）时面临的战术选择：继续强攻（争取首攻或达阵）、弃踢（交出球权但迫使对方远离得分区）或尝试射门（直接得分）。现代教练团队会借助数据分析评估不同选择的胜率，从而优化战术布置，这也是大数据在体育领域的典型应用之一。

动态驾驶路线	污染减少	文本摘要
电子健康记录	精准医疗	盗窃检测
情绪检测	预防医学	趋势发现
面部识别	预防疾病暴发	视觉产品搜索
欺诈检测	推荐系统	语音识别
健康结果改善	降低能源使用	天气预报

1.13 人工智能(AI)——计算机科学与数据科学的交汇点

当婴儿第一次睁开双眼，真的“看见”父母的脸了吗？它能理解“脸”，乃至简单形状的概念吗？婴儿必须通过观察去“认知”周遭世界，而这正是名为“机器学习”的人工智能（AI）技术所做之事——观察海量数据并从中学习。

如今，人工智能技术应用广泛：从游戏交互、计算机视觉场景落地，到赋能自动驾驶汽车上路、助力机器人习得新任务；从辅助医疗诊断、实现语音近实时跨语言翻译，到依托大型文本数据库训练出大语言模型，构建能流畅精准回应各类提问的聊天机器人……谁能想到，短短数年前，人工智能驱动的自动驾驶汽车竟会获准上路，甚至渐成常态？

1.13.1 人工智能

人工智能（AI）有以下两种定义：^{①②}

- 计算机科学的一个分支，聚焦于在计算机中模拟智能行为；
- 机器模仿人类智能行为的能力。

^① “artificial intelligence”，Merriam-Webster.com，<https://www.merriam-webster.com/dictionary/artificial%20intelligence>。

^② 维基百科，<https://zh.wikipedia.org/wiki/人工智能>。

“AI”这一术语由约翰·麦卡锡（John McCarthy）在 1956 年提出。^①如今，基于人工智能的系统已广泛渗透到我们的日常生活里。常见例子有计算机视觉、语音识别、自然语言翻译、推荐系统（像抖音推荐你可能感兴趣的视频，京东推荐你可能心仪的商品）、自动游戏系统、自动驾驶汽车，还有近些年涌现的令人瞩目的工具，比如：

- **聊天机器人**：如 DeepSeek、OpenAI ChatGPT、Google Gemini、Anthropic Claude 以及 Perplexity，它们能几乎像人类交流一样自然流畅地对话；
- **图像生成器**：像 OpenAI 的 DALL·E（现已整合到 ChatGPT 中）、谷歌的 Imagen（现已整合到 Gemini 中）、Midjourney、Stable Diffusion、Adobe 的 Firefly 等，可依据文字描述生成原创图像；
- **视频生成器**：例如 OpenAI 的 Sora、Adobe 的 Firefly Video、谷歌的 Veo 3、Meta 的 MovieGen 以及 Runway 的 Gen-4，能够按照文字描述产出原创视频。

在过去几十年里，人工智能曾长期处于“有问题但难寻解决方案”的状态。原因在于，一旦某个问题被攻克，人们就会说：“哦，那不算真正的智能，不过是个告诉计算机该做什么的程序罢了。”然而，随着机器学习（1.13.4 节）、深度学习（1.13.5 节）等人工智能技术的发展，我们不再预先为特定问题编写解决方案。而是让计算机从数据（通常是海量数据）中学习来解决问题。众多最有趣、最具挑战性的问题，正借助深度学习得以探索解决。

1.13.2 通用人工智能(AGI)

人工智能的最终目标是实现**通用人工智能**（Artificial General Intelligence, AGI），^②它像人类一样，能够在广泛的多种任务中理解、学习并应用知识，而无需额外训练——基本上就是一种“能够完成人类所能完成的任何任务”的人工智能。

专家们对 AGI 是否会成为现实，以及何时会实现存在分歧：有人认为会在 2026 年实现，有人认为要等到 2030 年代，也有人认为要等到 2300 年，甚至永远无法实现。

1.13.3 人工智能里程碑事件

多个 AI 里程碑事件引起了公众的关注和想象，也让大众开始相信 AI 是真实存在的，并促使企业思考如何将其商业化：

^① 译注：1956 年夏天，一群数学家和计算机科学家占领了达特茅斯学院数学系所在大楼的顶层。在大约 8 周的时间里，他们想象着一个新研究领域的可能性。约翰·麦卡锡（John McCarthy）是当时是达特茅斯大学的一名年轻教授，他在为研讨会写提案时创造了“人工智能”一词，他说研讨会将探索这样的假设：（人类）学习的每一个方面或智能的任何其他特征原则上都可以被精确描述，以至于可以用机器来模拟。

^② 维基百科，<https://zh.wikipedia.org/wiki/通用人工智能>。

-
- 1997 年，IBM 开发的深蓝（DeepBlue）计算机系统，在正式赛事中战胜国际象棋世界冠军加里·卡斯帕罗夫（Gary Kasparov），成为首台在标准竞赛规则下击败现役棋王的计算机。IBM 为深蓝输入数十万局大师级对局，它能以“暴力枚举”方式，每秒评估多达 2 亿步棋！这正是大数据应用的典型实践。凭借此成就，IBM 斩获卡内基梅隆大学 1980 年设立的 Fredkin 奖金（10 万美元）——该奖项专为奖励“首个击败世界象棋冠军的计算机开发者”而设。
 - 2011 年，IBM 的 Watson 在奖金百万美元的《危险边缘》（Jeopardy!）赛事中，击败两位人类顶尖选手。它同时调用数百种语言分析技术，从 2 亿页内容（含完整维基百科）里检索正确答案，需占用 4 TB 磁盘存储空间。Watson 依托 AI 机器学习与强化学习技术完成训练，展现出强大的语义理解与知识检索能力。
 - 围棋（Go）起源于数千年前的中国，因其约 10^{170} 种可能的棋盘状态，被视为史上最复杂棋类之一（对比参考：已知宇宙原子总数约为 10^{80} 个）。2015 年，谷歌 DeepMind 研发的 AlphaGo，凭借深度学习技术与双神经网络，击败欧洲围棋冠军樊麾——围棋的复杂度远超国际象棋，此胜利标志 AI 突破复杂策略游戏边界。
 - 2017 年，谷歌基于 AlphaGo 技术推出了通用版的 AI AlphaZero，可自主学习新棋类规则，能在不需要专家数据的前提下采用自博弈的方式进行训练。2017 年 12 月，它仅用不到 4 小时，就通过强化学习掌握国际象棋规则，并在 100 局对弈中，零封世界顶级国际象棋程序 Stockfish 8（全胜或平局）；仅训练围棋 8 小时后，又在与“前辈”AlphaGo 的 100 局较量中赢下 60 局，展现出强大的“自学+归纳”能力。
 - 2023 年，谷歌推出基于 AlphaZero 的 AlphaDev，聚焦“发现更高效计算机科学算法”，尤其针对排序算法突破。其成果显著：为 LLVM libc++ 排序库带来新算法，使短序列排序性能提升最高 70%，25 万以上元素的长序列排序也能提速约 1.7%——AI 开始深度赋能基础编程工具，重塑计算机科学底层逻辑。

1.13.4 机器学习

作为人工智能的子领域，**机器学习**（machine learning, ML）赋予了计算机无需显式编程就能自主学习的能力。其运作模式通常是：借助预定义算法，利用训练数据构建模型，这些模型随后可用于开展预测工作。

举个例子，若给机器学习模型提供带标签的训练数据，数据里包含合法邮件和垃圾邮件（也叫 spam），模型就能从这些数据中学习规律，之后用于判断从未“见过”的新邮件是否属于垃圾邮件。像微软 Outlook、谷歌 Gmail 这类邮件应用，正是运用此类技术，把垃圾邮件自动移出你的收件箱。

1.13.5 深度学习

作为机器学习的子领域，**深度学习**（deep learning）的核心在于构建人工神经网络，其目

标是“模拟人类大脑的神经网络，使计算机能以类人方式学习与决策”。^①

“深度”一词源于人工神经网络的软件“层数”——层数越多，网络深度越高。当数据在各层间传递时，网络会从训练数据中不断学习进阶知识。

如今，得益于大数据的可获取性、强劲的处理性能、更快的网络速度，以及并行计算机软硬件的发展，越来越多的组织和个人能够开发资源密集型的深度学习解决方案，并在计算机视觉、语音识别、语音合成及新兴的生成式 AI 应用（1.14 节）等领域取得了令人瞩目的成果。

1.13.6 强化学习

作为机器学习的子领域，**强化学习**（reinforcement learning）的算法通过与环境交互实现学习，这一过程类似于人类的学习模式——比如游戏玩家尝试新游戏，或是婴儿学习走路、辨认父母。该算法会构建一个“智能体”，其学习逻辑为：尝试执行任务→接收成败反馈→调整策略→再次尝试，核心目标是最大化奖励值。具体而言，智能体做出正确行为时获得正向奖励，执行错误操作则受到负向奖励（惩罚）。它会基于这些反馈信息规划下一步动作，持续优化策略以追求总体奖励的最大化。

1.14 生成式 AI

生成式 AI（generative AI）描述的是可以根据用户提供的输入提示生成新内容的算法，这些内容包括文本、图像、音频和视频等。本书提到的“AI”基本上都是指“生成式 AI”。

聊天机器人

DeepSeek、OpenAI ChatGPT、Google Gemini、Anthropic Claude 和 Perplexity 都是基于 AI 的聊天机器人，你可以用自然语言与它们对话。例如，可以在提示框中输入以下内容：

- 告诉我 Java 编程语言的历史；
- “元宇宙”这个术语是谁提出的？
- 总结这篇文章的内容：（后跟文章链接）。

所有聊天机器人随后都会生成看起来像是由人类写成的回答。

^① Jason Brownlee, “What is deep learning?”, November 16, 2023, <https://machinelearningmastery.com/what-is-deep-learning>。

多模态聊天机器人

当生成式 AI 刚问世时，它们通常只能接收文本指令并生成单一类型的输出，如文本或图像。而如今，许多生成式 AI 聊天机器人已具备**多模态**交互能力——不仅能理解文本，还能处理图像、音频甚至视频输入，并能融合文本、图像、音频、视频等多种形式生成输出。

例如，用户可上传一段语音文件，指示 AI 完成以下连贯操作：将语音转录为文本→展示转录内容→分析情感倾向（积极/消极/中性）→翻译成另一种语言→合成对应语言的语音，甚至能根据生成的文本进一步创作图像和视频！这些功能在几年前连资深开发者都难以实现，如今却已向公众开放。我们将在第 19 章编写这样的 AI 应用程序。

图灵测试

与 AI 对话时，可能会产生与真人交流的错觉。这种“人机智能边界模糊”的现象，早在 1950 年就被艾伦·图灵（imitation game）以“模仿游戏”^①理论预见到——即如今广为人知的图灵测试，其核心是判断计算机是否具备真正的智能表现。

图灵测试的设定如下：三个终端彼此隔离，其中两个终端由人类通过键盘操作，第三个终端则由计算机控制。测试者通过文本向另外两个终端提问，人类参与者与计算机分别回答。若测试者无法分辨回答源自人类还是机器，则认定该计算机展现出智能行为。这一理论框架为人工智能的智能性评估奠定了基础，至今仍被视为衡量人机交互水平的重要参照。

生成式 AI 可以做什么

我们曾向 AI 提问：“请列出 AI 能生成的所有内容类型”。然后我们让它对冗长的回答进行了总结，主要包括以下几个方面：

- **文本**：文章、论文、故事、诗歌、歌词、代码、摘要、营销文案、学术写作、电子邮件、教学指南等；
- **视觉/图像**：数字插画、概念艺术、角色设计、逼真照片、标志、品牌元素、信息图表、AI 增强图形等；
- **音频**：AI 创作音乐、音效、背景音、文本转语音、旁白、语音克隆、合成语音等；
- **视频**：AI 生成动画、解释视频、文本生成视频、深度伪造视频、虚拟角色等；
- **三维建模**：游戏、模拟、虚拟现实中的 3D 模型、数字雕塑、动画、特效等；
- **编程与软件**：自动代码生成、编程辅助、聊天机器人、虚拟助手、自动化脚本、机器学习算法等；

^① 译注：可以观看同名电影了解详情，<https://movie.douban.com/subject/10463953/>。

- **交互类**：聊天机器人对话、动态叙事、个性化测验、调查问卷、互动网站和教育工具等；
- **数据相关**：AI生成的数据分析报告、可视化图表、表格、合成数据集等；
- **教育**：学习指南、课程计划、测验题、教科书章节、教程、互动学习练习等；
- **多模态**：结合语音与图形的 AI 视频、互动模拟、融合文本、图像与用户互动的虚拟环境等；
- **艺术与创意**：漫画、小说、混合媒体艺术、风格迁移作品、定制商品设计、AI 剧本等；
- **其他杂项**：AI 生成菜谱、健身计划、每日正能量语句、产品名称、口号、个性化推荐、法律文档模板等。

这些功能让内容创作变得更加普及，几乎人人皆可参与。

大语言模型

大语言模型（Large Language Model, LLM）是一类能够理解并生成自然语言的人工智能系统。其“大”的定义源于模型架构——通常以包含数十亿甚至数万亿参数的人工神经网络（神经网络）形式实现。

LLM 的训练依托海量文本数据（如文章、书籍、互联网内容、代码等）。在神经网络的学习过程中，模型会持续调整参数以优化输出效果，这些参数正是 LLM 理解和生成语言模式的核心机制。

这类模型具备两大显著特征：一是能生成与人类写作高度相似的文本，部分先进模型还支持持续学习与动态适应，随时间推移不断提升性能；二是擅长处理多元自然语言任务，既包括翻译、分类、摘要、问答、情感分析、创意写作等基础应用，也能高效实现代码生成、代码解释、错误定位、代码重构等专业开发需求。

Transformer

Transformer 是一种专为处理序列数据（如文本）而设计的神经网络架构。它的核心优势在于，不仅能够精准捕捉句子中相距较远词语间的语义关联，还能有效识别出各个词语的相对重要性。当前主流的大语言模型（LLM）普遍采用了**生成式预训练 Transformer**（Generative Pre-trained Transformer, GPT）架构，通过融合多种人工智能技术，在海量文本数据上进行训练。除了 GPT 之外，典型的 Transformer 技术还包括谷歌的 BERT（Bidirectional Encoder Representations from Transformer），其主要侧重于稍后讲述的自然语言理解（NLU）任务。

自注意力机制

自注意力机制（self-attention）是 Transformer 模型在自然语言处理领域的核心机制之一。它模仿了人类理解语言的方式：当我们在分析句子中的某个词语时，其周围词语所提供的上下文信息对于准确把握其含义至关重要。

例如，“好”这个词通常带有积极的含义，但如果前面加上“不”字，即“不好”，其整体语义立刻转变为负面。又如“bank”一词，当它与“loan”（贷款）一同出现时，我们理解为“银行”；而当它与“river”（河流）搭配时，则意指“河岸”。在教育场景中，“test bank”（题库）一词，其“bank”的含义也显然不同于“银行”或“河岸”，特指方便教师创建测验的资源集合。类似地，像“破冰”这样的短语，在社交场合中使用时，其意义也从字面上的行为转变为打破尴尬气氛的比喻含义。

Transformer 正是通过自注意力机制来深入分析句子中每个词语与其周围词语之间的关联。由于并非所有词语对句子的整体意义贡献相同，自注意力机制能够为这些词语赋予不同的权重，从而将更多的“注意力”集中在那些对理解当前词语或整个句子最关键的部分上。

与人类逐字阅读的习惯不同，Transformer 能够并行处理和分析大量的词语，从而在更宏观的层面上建立复杂的语义联系，实现对整体文本含义的更深刻理解。

词元、上下文和上下文窗口

词元（token）是大语言模型（LLM）中的基本处理单元。一个词元可以是一个完整的单词、一个汉字、单词的一部分（例如，英文单词“running”可能被分解为“run”和“ning”）或者一个标点符号。**词元化**（tokenization）是将这些词元转换为模型能够理解和处理的数值形式的过程。

词元的**上下文**指的是位于某个词语、汉字或短语之前（有时也包括之后）的一系列相关词元。模型通过理解这些上下文信息，能够生成更加准确和恰当的响应。

上下文窗口则是指模型在一次处理中可以同时考虑的最大词元数量，这包括了用户输入和模型输出的总和。上下文窗口越大，意味着模型在与你进行对话时能够“记住”和利用的上下文信息越多，其生成的响应也就越相关、越连贯。然而，更大的上下文窗口也意味着更高的计算和存储成本，因此生成有用响应的成本也会相应增加。这就是为什么许多生成式 AI 的免费版通常会限制上下文窗口的大小，或者在给定时间段内限制用户的查询次数。

不同生成式 AI 聊天机器人的上下文窗口大小存在显著差异。以 DeepSeek、OpenAI ChatGPT、Google Gemini、Anthropic Claude 和 Perplexity 为例，它们的上下文窗口大小从 DeepSeek 免费网页版的 128 000 个词元到 Gemini 付费版的 1 000 000 个词元不等。这些差

异直接影响了模型处理长文本和维持长时间对话连贯性的能力。^①

提示和提示工程

提示工程是设计指令（或称“提示”）以从生成式 AI 获得最佳响应的过程。本书许多章节都以“生成式 AI 练习”结尾，我们将重点关注提示工程，旨在帮助你更有效地与这些 AI 工具进行交互。可以将此视为学习一门新的、独特的“编程语言”：通过精心设计的提示，你能够编写精确的指令或查询，从而引导 AI 生成所需的输出。

向 AI 提问时，请务必注意以下几点：

- **明确指定语言和版本**：要求 AI 编写 Java 代码或回答有关 Java 功能的问题时，请务必在提示中明确包含“Java”字样。此外，请要求它们根据你正在使用的 Java 版本，采用最新的 Java 惯用法来编写代码；
- **记录与反思**：务必记录你的提示及其产生的结果质量。这对于理解何种提示有效、何种无效至关重要。同时，这也有助于你识别不同生成式 AI 平台之间的细微差别，从而逐步建立起一个有价值的提示库和预期结果库；
- **多平台尝试**：积极尝试使用多种生成式 AI（包括不断涌现的新平台），并比较它们生成的结果，以了解它们各自独特的能力和特点；
- **策略多样化**：尝试不同的提示策略，观察哪些策略能帮助你实现所需的结果。请记住，不同的编程挑战可能需要采用不同的提示策略；
- **迭代优化**：如果生成式 AI 的结果不尽如人意，请及时调整提示并重新尝试。更优质的提示往往能带来更出色的结果。有些生成式 AI 能够更深入地理解并处理某个主题。

向某些 AI 提问时，按 Enter（或 Return）键会立即提交你的提示。而对于另一些 AI，可能需要明确地单击一个按钮来提交。若想在提交提示的情况下将光标移动到新行，请按 Shift+Enter 或 Shift+Return。

在本书第 2 章及后续章节中，当我们要求你将 Java 代码提供给 AI 时，建议直接从 Java 源代码文件中复制并粘贴代码。大多数生成式 AI 也支持上传一个或多个文件，但请注意，这有时是仅供付费订阅用户使用的功能。

学习如何创建高质量的提示需要大量的实践经验。你可以参阅各种在线指南来构建更有效

^① 译注：不要小看 DeepSeek 免费的 12 万多的上下文窗口，它能完整分析约 4000 行的 Java 项目（含依赖关系），支持跨文件理解（如同时处理 Controller+Service+DAO 层），还能理解 300 页技术文档（如 JSR 规范全文）。至于 Gemini 付费版那么大的上下文窗口，则非常适合写长文，100 万字的小说应该不在话下。有兴趣的读者可以考虑开源项目 SillyTavern（酒馆），网址是 <https://github.com/SillyTavern/SillyTavern>。

的提示，甚至可以直接要求 AI 本身来帮助你优化提示。DeepSeek 提供了优秀的提示库示例 (<https://api-docs.deepseek.com/prompt-library>)，供你参考学习。

生成式 AI 轶事

尽管生成式 AI 功能强大，但用户在使用时仍需保持审慎。在 2023 年 11 月至 12 月期间，我们观察到 ChatGPT 在执行过去能够生成详细响应的任务时，表现出“懒惰”的倾向，不再提供详细解决方案，而是转为提供执行任务的步骤指引。有研究人员甚至提出了“寒假假设”，推测 ChatGPT 可能因为西方假期期间人们工作量减少而“学会”减少自己的工作量。

除了这种“懒惰”现象，AI 聊天机器人有时还会出现错误，甚至产生“幻觉”；换言之，它们会自信地呈现不正确的信息，并将其当作事实。例如，我们曾询问多个 AI “这些 Java 语句输出什么？”，并提供了 5 个简单的语句来输出以下三行星号：

```
****
*****
*****
```

然而，AI 典型的响应却是显示一大串星号 (*****)，或者垂直地每行仅显示一个星号，这与预期结果不符。

另一个有趣的错误案例发生在 2024 年 8 月左右，当时有人发现，向 ChatGPT 的语音模式提问“strawberry 中有多少个 R？”时，它会给出错误的口头回答“strawberry 中有两个 R”。然而，将相同的问题以文本形式输入，却能得到正确的答案“strawberry 中有三个 R”。即便在 2025 年 3 月 30 日我们重复这个实验，ChatGPT 的语音模式依然给出错误答案。当我们纠正它说：“不。strawberry 中有三个 R”时，ChatGPT 的语音回复才更正道：“让我们看看：S-T-R-A-W-B-E-R-R-Y。你是对的，有三个 R！”^①

关于“幻觉”的一个典型例子是，生成式 AI 曾被发现会凭空编造法庭案件。你可以尝试使用以下提示进行研究：“给我提供法庭案件的引文，其中法官斥责律师引用 AI 生成的不存在的法庭案件。”

另一个来自我们自身经验的幻觉案例，发生在 ChatGPT 早期阶段。当时我们尝试其文本内容生成能力，要求它编写各种计算机术语的定义，并为每个定义提供 4 个参考文献，其中

^① 译注：这是一个有趣的例子，AI 真的是人工“智能”吗？答案是否定的，它确实存在一个“上限”。归根结底，大模型处理文本时，会“机械”地将单词拆分成更小的词元 (token)，例如：Strawberry → "Str" + "aw" + "berry"。"berry" 中的两个 "r" 被合并为一个词元，导致模型无法感知它们是两个独立字母，所以才提出只有 2 个 r 的结论。

一个必须是维基百科。总的来说，它生成的定义质量尚可，并且提供了指向正确维基百科页面的直接链接。然而，无一例外的是，我们要求的其他三个参考文献完全是捏造的——包括虚假的人名、引文、网站，有时甚至是真实的网站但页面根本不存在。尽管如今的生成式 AI 相较早期已有了显著改进，但你务必始终验证它们提供的信息。

此外，互联网上的数据本身可能存在偏差。当生成式 AI 接受训练时，它们可能会习得并反映这些偏差。本书将在生成式 AI 练习 1.23 和第 19 章的某个小节中对这些问题进行更详细的讨论。

NLP 和 NLU

自然语言处理（Natural Language Processing, NLP）与**自然语言理解**（Natural Language Understanding, NLU），是助力生成式 AI 实现与人类语言交互的两项关键技术。

NLP 专注于计算机对人类语言的处理与分析过程，它将文本拆解为有意义的部分，例如单词、汉字、句子，并剖析它们之间的关联。不妨把 NLP 视作一套辅助计算机读取并理解文本的体系，就如同此刻你阅读和处理这些文字的方式。

NLU 则着重帮助计算机理解语言所蕴含的意义及所处的上下文情境，而不只是局限于单个单词的认知。例如，若向生成式 AI 询问“今天天气如何？”之后，紧接着又问“我该带伞吗？”，生成式 AI 便会借助 NLU 技术，识别出你的第二个问题与天气相关，并且明确你正在询问关于防雨的措施。

NLP 与 NLU 共同构成了生成式 AI 理解用户提示，并生成类人回应的基石。倘若缺失这两项技术，与生成式 AI 的交互就仿佛是在尝试与一个熟知某种语言里所有字词，却不明白字词如何组合表意、实际含义为何的人对话一般。

1.15 小结

本章介绍了为你在本书中将要学习的 Java 编程能力奠定基础的术语和概念。我们讨论了摩尔定律，并指出它在几十年里准确预测的持续不断的巨大改进似乎正在结束。因此，硬件设计人员已转向多核架构，你将使用第 18 章讨论的多线程技术对其进行编程。我们介绍了 Java，并鼓励你在开发新软件时避免“重复造轮子”，而是使用 Java API 和开源 Java 库中的现有软件组件。

我们简要回顾了面向对象以及类、对象、接口、继承和多态的概念，为第 2 章～第 7 章和第 11 章中面向对象的案例学习以及第 8 章～第 9 章中面向对象编程的详细论述奠定了基础。

我们在 Java 开发工具包（JDK）的背景下讨论了典型的 Java 程序开发环境，然后使用其工

具实际体验了一个 Java 程序，为你运行本书的示例程序和你将编写的原创程序做好准备。你也可以使用某个 Java 集成开发环境（IDE）运行我们的程序——我们在 <https://deitel.com/javafp5> 上提供了几种流行 IDE 的快速入门指南。

我们讨论了互联网、Web、云计算和物联网（IoT）。我们介绍了元宇宙和相关的关键技术——区块链、加密货币、同质化代币（NFT）、Web3、增强现实（AR）、虚拟现实（VR）和混合现实（MR）。我们介绍了各种软件开发技术，包括重构、设计模式、软件开发工具包（SDK）、版本控制系统（VCS）、敏捷软件开发、测试驱动开发（TDD）、微服务和容器化。

我们介绍了大数据分析，然后讨论了大数据和从千字节（KB）到尧字节（YB）的各种数据大小。最后，我们介绍了人工智能（AI）——计算机科学和数据科学的交叉点——介绍了各种 AI 里程碑，并概述了几种 AI 技术，包括通用人工智能（AGI）、机器学习、深度学习、强化学习和生成式 AI，你将在本书中利用这些技术来增强 Java 学习体验。

从第 2 章开始，你将开始学习并实际使用 Java 进行编程。

专题：生成式 AI 练习

对于以下每一个问题，请尝试多个生成式 AI，以确定哪一个表现最佳。始终验证其输出结果。请记住，它们有时会犯错，甚至产生“幻觉”，杜撰出虚假的信息，因此可能生成不可靠的内容。

1.1（2010 年“闪电崩盘”事件分析）2010 年 5 月 6 日，美国股市突发“闪电崩盘”，大盘在数分钟内暴跌，蒸发超 1 万亿美元市值后又快速回升。请使用 AI 技术调研此次事件的具体成因，并探讨其暴露的系统性问题（如算法交易风险、市场监管漏洞、技术依赖性等）。

1.2（Java 语言竞争力分析）请 AI 梳理 Java 在现代编程语言中仍被优先选择的核心优势（可从跨平台特性、生态成熟度、企业级开发支持、社区活跃度、性能优化机制等维度展开）。

1.3（Java 版本迭代机制评估）请 AI 阐释 Java 每 6 个月发布新版本的设计逻辑，并分析该机制的利弊（优势如快速迭代、功能更新及时；潜在问题如兼容性挑战、企业适配成本等）。同时探讨此节奏对关键业务软件开发企业带来的技术挑战（如版本迁移风险、长期维护成本、第三方库兼容问题等）。

1.4（AI 响应变异性调查）当用户在不同时间间隔（数秒、数分钟、数小时、数天及以上）重复提问同一问题时，AI 的回答往往存在显著差异。请 AI 解释该现象的技术成因（可从模型随机性、上下文理解差异、数据更新机制、算法优化策略等角度分析）。

1.5 (DeepSeek 功能调研): 向 DeepSeek (<https://chat.deepseek.com>) 询问其用于处理文本、图像、音频和视频 (若有相关功能) 的多模态能力教程。并请它给出可以构建的多模态应用类型的建议。^①

1.6 (豆包功能调研): 向豆包 (<https://www.doubao.com>) 询问其用于处理文本、图像、音频和视频 (若有相关功能) 的多模态能力教程。并请它给出可以构建的多模态应用类型的建议。

1.7 (通义千问功能调研): 向通义千问 (<https://www.tongyi.com>) 询问其用于处理文本、图像、音频和视频 (若有相关功能) 的多模态能力教程。并请它给出可以构建的多模态应用类型的建议。

1.8 (Kimi 功能调研): 向 Kimi (<https://www.kimi.com>) 询问其用于处理文本、图像、音频和视频 (若有相关功能) 的多模态能力教程。并请它给出可以构建的多模态应用类型的建议。

1.9 (投身 AI 领域的职业规划): 向 AI 咨询, 若想投身 AI 领域开启职业生涯, 应提前做好哪些准备, 包括但不限于需掌握的专业知识、要培养的技能素养, 以及应积累的实践经验等方面。同时, 让 AI 梳理该领域正在催生哪些全新的工作机会, 如 AI 工程师、AI 内容审核专员等岗位的职责与前景。

1.10 (技术革新与就业变迁的关联): 部分人担忧 AI 的兴起会导致大量岗位流失。向 AI 索要其他重大技术革新的实例, 比如互联网诞生、Google 搜索问世等。这些技术在发展进程中, 虽淘汰了一些旧有工作岗位, 但也同步创造出诸多全新的就业机会, 分析其中的演变逻辑与规律。

1.11 (AI “幻觉” 现象剖析与应对策略): 向 AI 询问, 为何它们有时会产生 “幻觉”, 即生成看似合理却与事实不符的信息。同时, 请 AI 提供有效检测这类 “幻觉” 信息的方法, 以及在实际应用中处理该问题的最佳操作指南, 以确保从 AI 获取的内容真实可靠。

1.12 (生成式 AI 的趣闻轶事盘点): 要求 AI 列举一些生成式 AI 在实际应用或开发过程中发生的趣闻轶事, 如因指令理解偏差生成令人捧腹的内容, 或者意外创造出别具一格艺术作品的故事等。

1.13 (探寻免费或可试用的 AI 资源): 要求 AI 梳理目前市面上存在的免费 AI 产品, 或者那些虽设有付费版本, 但也向用户开放了免费使用权限, 以及提供免费试用期的 AI 工具, 例如免费使用基础功能的绘图 AI、有一定免费调用额度的文本生成 AI 等。

^① 译注: 这 4 个练习都可以这样向 AI 提问: 你有处理文本、图像、音频和视频 (若有相关功能) 等的多模态能力吗? 请给出教程, 并给出我可以构建的多模态应用类型的建议。

1.14（视频生成）以下是一些支持有限免费视频生成的 AI：

- **快手的可灵 AI**：每天登录可领取 66 灵感值，可生成多个 5 秒视频，能生成 3 分钟内的超高清视频（1080p/30 帧），支持“AI 唱跳”功能，还可设置视频首尾帧与智能对口型。
- **字节跳动的即梦 AI**：每日赠送一定积分，支持图文生视频、动态漫画生成及 AI 智能运镜，独创“动效画板”，可自定义物体运动轨迹，生成人物多动作连贯视频。
- **智谱清影**：能在 30 秒内生成 10 秒、4K、60 帧视频，支持多种风格和背景音乐，适用于个人创作和专业制作，目前功能对所有用户免费开放，可不限次数生成视频。
- **白日梦 AI**：首次注册登录可领取 1000 积分，基于先进自然语言处理技术，支持用户输入文本内容快速生成最长可达 6 分钟的视频，具备文本到视频智能转换、丰富角色库、多样化视频风格生成、自动化分镜设计等功能。

要求 AI 提供由每项服务制作的演示链接。观看这些演示并进行评论。

1.15（免费文本转语音服务）以下是一些具有代表性的免费文本转语音服务：

- **讯飞听见**：新用户注册可获赠 300 分钟免费转写时长，支持 10 几种语言及方言的文本转语音，提供“情感主播”（如活泼、温柔等音色）和专业场景音（如新闻播报、客服语音）。其语音合成技术可模拟真人语气停顿，适合短视频配音、有声书制作等场景。
- **腾讯智影**：支持每日 5000 字以内的文本转语音，提供 20 几种音色（如卡通童声、粤语主播、英语外教等），并可自定义语速、语调。特色功能“智能配音”支持上传文本后自动匹配场景音效，适合自媒体内容快速生成。

要求 AI 提供由每项服务制作的演示链接。观看这些演示并进行评论。

1.16（检测 AI 生成内容）要求 AI 查找用于检测 AI 生成内容的免费工具，以及提供免费版本或试用期的工具。需注意该任务存在技术难点，检测结果常出现误判，同时请对使用此类工具的伦理问题展开评论。

1.17（伦理）要求 AI 撰写一份关于 AI 伦理的教程，并梳理使用 AI 时涉及的伦理问题。

1.18（负面因素）要求 AI 列举 AI 技术的负面因素。

1.19（监管）要求 AI 列出当前各国政府针对 AI 的监管措施（需说明不同国家和地区的政策差异）。

1.20（安全）要求 AI 撰写一份关于 AI 安全的教程。

1.21（智商测试）要求 AI 列出可能出现在智商测试中的问题类型。

1.22（轶事：光速决策）在美国内战期间，电报被用于近乎实时的通信。据传，亚伯拉

罕·林肯（Abraham Lincoln）曾有一则轶事：一位年轻的军官兴奋地向他报告，战场正在铺设电报线路，并告诉总统，现在他能以光速做出决策。而更为年长且睿智的林肯回应道：“但你也会以光速做出错误的决策。”

我们让 AI 来追溯这则轶事的来源。以下是 DeepSeek 的回答：

“您提到的这句话通常被认为是亚伯拉罕·林肯在内战期间对电报使用的评论，但它更像是后人对其观点的转述，而非直接引用。其核心思想是：电报的快速通信虽然能加速决策，但也意味着错误的决定同样能以光速传播。常见的转述版本是：‘有了电报，错误的决策现在也能以光速传递。’这反映了林肯对新技术的深刻洞察——尽管电报在战时极大地提升了通信和指挥效率，但它也使得错误决策的影响更快扩散。电报确实是那个时代的关键工具，但它在带来便利的同时，也放大了决策的风险。”

AI 是一项新兴技术，其影响深远，需审慎考量。与林肯的电报轶事相比，它的一大优势在于能提升决策质量。你可以尝试向 AI（如 DeepSeek）提问：“如何帮助我提高决策质量？”探索其在这方面的潜力。

1.23（AI 偏差问题）互联网数据存在大量偏差，且这些数据正用于训练大语言模型，导致模型可能习得这些偏差。询问 AI 如何避免从训练数据中拾取偏差。

1.24（AI 版权问题）询问 AI 如何确保其生成的图像为原创内容，且不侵犯版权。

1.25（智能体）要求 AI 解释“智能体”的定义，并列举若干典型案例。

1.26（“人在环中”机制）询问 AI 在使用其输出结果（尤其是业务关键型和任务关键型场景）时，保持“人在环中”的重要性。^①

^① 译注：人在环中（human-in-the-loop, HITL）是一种人机交互机制，即将个人的判断决策融入 AI 系统流程中去，构成流程的运行闭环。

第 2 章 Java 编程入门

学习目标：

- 编写、编译并运行 Java 程序；
- 使用输入和输出语句；
- 声明、初始化并使用基元类型的变量；
- 使用算术操作符；
- 理解算术操作符的优先级；
- 编写判断语句（如 `if`、`if...else` 等）；
- 使用关系和相等性操作符来构建条件；
- 利用 AI 增强 Java 学习体验；
- 在学习自定义类之前，通过创建和使用 Java API 的 `String` 类对象，体会“对象自然”学习法的价值。

2.1 导读

本章将带你开启 Java 编程之旅。我们从简单的程序入手，这些程序会在屏幕上输出信息。随后将展示一个较复杂的程序，它能从用户处输入（即读取）两个数字，求和并显示结果。你将学习执行算术运算，并保存计算结果供后续使用。本章最后一个示例演示了条件判断语句，它比较两个数字，根据结果显示相应信息。

在整章的学习过程中，你将利用 AI 增强 Java 学习体验。本章的程序将使用 JDK 自带的命令行工具进行编译和运行。若倾向于使用集成开发环境（IDE），可访问本书英文版主页（<https://deitel.com/javafp5>），获取主流 IDE 的入门指南。

“对象自然”学习方法

在程序开发中，有许多经过精心设计与测试的现成类可供使用。通过创建和调用这些类的对象，我们能够以极简的代码完成重要任务。这些类主要来源于 Java API 和免费的第三方库（通常由围绕主流编程语言建立的大型开源社区开发）。

为了让你在本书初期就能体验这种编程模式，在学习创建自定义类之前，我们会先引导你创建并使用 Java API 现有类的对象。这称为“对象自然”（Objects-Natural）学习法/教学法。本章最后一个示例将创建和使用 `String` 类的对象。而在第 8 章～第 9 章，你将学习如何构建有价值的自定义类，供自己或其他程序员复用。

2.2 第一个 Java 程序：显示单行文本

来看一个显示单行文本的简单程序。图 2.1 展示了程序代码及其输出结果。我们添加行号用于讲解——行号本身并非代码的一部分。该示例体现了 Java 程序的多个重要特性。注意，第 7 行执行了程序的核心操作：在屏幕上显示文本“欢迎学习 Java 编程!”。

```
1 // 图 2.1: Welcome1.java
2 // 文本输出程序
3
4 public class Welcome1 {
5     // main 方法开始程序执行
6     public static void main(String[] args) {
7         System.out.println("欢迎学习 Java 编程!");
8     } // 结束 main 方法
9 } // 结束 Welcome1 类
```

欢迎学习 Java 编程!

图 2.1 文本输出程序

注释

在程序中插入注释可以记录代码功能并提升可读性。编译器会忽略注释内容，因此运行程序时注释不会触发任何操作。本书约定每个程序开头会添加注释来标注对应的图号和文件名，便于在示例文件夹（参见“前言”后的“准备工作”部分）中定位文件。例如，第 1 行的注释表明可以在 `examples/ch02/fig02_01` 文件夹中找到 `Welcome1.java`。**单行注释**以 `//` 开头，延伸至行尾，位置不局限于行首，也可出现在行内（如第 5 行、第 8 行和第 9 行）。按照约定，第 2 行注释用于描述程序功能。

Java 还支持传统的**多行注释**，例如：

```
/* 这是一条多行注释，
   可以跨越多行。*/
```

多行注释以 `/*` 开始，以 `*/` 结束，中间内容会被编译器忽略。部分组织要求在每个源代码文件开头使用多行注释，以说明程序功能、开发者姓名及最后修改时间等。

Javadoc 注释

第三种注释称为 **Javadoc 注释**，以 `/**` 和 `*/` 作为分隔符，中间的内容会被编译器忽略。Javadoc 注释允许将程序文档直接嵌入代码中，是业界更推荐的注释方式。`javadoc` 工具（属于 JDK 的一部分）会读取 Javadoc 注释，并利用它们生成可在 Web 浏览器中查看的程

序文档。^{①②}为节省篇幅，本书的代码将全面使用单行注释（即//注释）。

语法错误

若遗漏注释分隔符，将触发**语法错误**——此类错误源于编译器检测到违反 Java 语言语法规则的情况。这些规则如同自然语言的语法规则，例如英语、西班牙语、中文、印地语等语言中关于句子结构的规则体系。语法错误会造成**编译错误**，因其属于编译器可识别的错误类型，这种程序在编译时会失败。程序唯有在完全不存在语法错误的前提下，方可成功完成编译过程。

利用空白

空行（如第 3 行）、空格字符和制表符统称为**空白**（whitespace），其作用是提升程序的可读性。编译器会忽略所有形式的空白。

声明类

第 4 行开始声明 `Welcome1` 类。每个 Java 程序至少要声明一个类。^③关键字 `class` 用于引入类声明，后跟类名（如 `Welcome1`）。**关键字**（keyword）是 Java 语言保留的特殊标识符，全部由小写字母组成。我们声明的类大多以 `public` 关键字开头，它表示该类具有公共访问权限。关于 `public` 类和非 `public` 类的详细区别将在第 8 章讨论。表 2.1 列出了 Java 的全部关键字。

表 2.1 Java 关键字

分类	关键字
常规关键字	<code>abstract</code> 、 <code>assert</code> 、 <code>boolean</code> 、 <code>break</code> 、 <code>byte</code> 、 <code>case</code> 、 <code>catch</code> 、 <code>char</code> 、

^① “How to Write Doc Comments for the Javadoc Tool”，<https://oracle.com/technical-resources/articles/java/javadoc-tool.html>。

^② “The javadoc Command”，<https://docs.oracle.com/en/java/javase/24/docs/specs/man/javadoc.html>。

^③ 译注：在 Java 中，每个程序必须至少声明一个类，这是因为 Java 是一种面向对象的编程语言，所有代码都必须位于类的内部。对于可独立运行的程序（如 “Hello World”），这个类还需要包含一个特定的 `main` 方法作为程序的入口点。

	class、continue、default、do、double、else、enum、extends、final、finally、float、for、if、implements、import、instanceof、int、interface、long、native、new、package、private、protected、public、return、short、static、strictfp、super、switch、synchronized、this、throw、throws、transient、try、void、volatile、while、_(下划线)
上下文关键字 (在特定的程序上下文中使用)	exports、module、non-sealed、open、opens、permits、provides、record、requires、sealed、to、transitive、uses、var、when、with、yield
语言当前未使用但保留的关键字	const、goto

public 类的文件名

`public` 类必须存储在文件名与类名一致（包括拼写和大小写）且扩展名为 `.java` 的文件中，例如 `Welcome1` 类需存储在 `Welcome1.java` 文件里。若 `public` 类的文件名与类名加上 `.java` 后缀在拼写或大小写方面存在差异，编译时将触发错误。

类名和标识符

按照惯例，类名标识符中的每个单词首字母大写。例如，`DollarAmount` 这个类名，第一个单词 `Dollar` 以大写字母 `D` 开头，第二个单词 `Amount` 以大写字母 `A` 开头。这种命名规范称为驼峰命名法，因为其中的大写字母就像骆驼的驼峰一样突出。^①

类名属于标识符，它由字母、数字、下划线（`_`）和美元符号（`$`）组成，且不能以数字开头，也不能包含空格。像 `Welcome1`、`$value`、`_value`、`m_inputField1` 和 `button7` 都是合法的标识符。名称 `7button` 则是非法的，因为它以数字开头。`input field` 也非法，因为它包含空格。`Java` 区分大小写，即大写字母和小写字母被视为不同字符，所以 `value` 和 `Value` 是两个不同但均合法的标识符。

^① 译注：更准确地说，这是“大驼峰命名法”。在编程命名规范中，`Pascal` 命名法（`Pascal Case`）和大驼峰命名法（`Upper Camel Case`）是同一种风格，即所有单词首字母大写（如 `DollarAmount`），常见于类名、接口名等。而小驼峰命名法（`Lower Camel Case`）要求第一个单词的首字母小写（如 `getBalance`），多用于方法或变量名。稍后的 2.5 节就会用到小驼峰命名法。

类主体

左大括号{（第 4 行末尾）标志着类主体的开始。相应地，每一个类的声明都必须以右大括号}结束（第 9 行）。

第 5 行~第 8 行进行了缩进。在每个类声明中，类的主体部分应在左右大括号之间整体缩进一个层级。这种缩进规范有助于突显类的结构，提升代码可读性。本书示例采用三个空格作为一个缩进层级，不过许多程序员更习惯使用两个或四个空格。无论选择哪种方式，关键是要在整个项目中保持一致。集成开发环境（IDE）通常会自动处理代码缩进，默认使用四个空格。你既可以通过按 Tab 键进行缩进，也可以在 IDE 的设置中自定义 Tab 键对应的空格数量。

如果大括号没有正确配对，将会引发语法错误。为避免此类问题，建议在输入左大括号{后，立即输入对应的右大括号}，然后将光标定位到两个大括号之间，进行适当缩进后再开始编写类的主体内容。在大多数 IDE 中，当你输入左大括号{时，IDE 会自动添加右大括号}，并将光标置于中间，同时完成缩进操作。这种自动功能可以有效减少因大括号不匹配导致的编译错误。

声明方法

第 6 行是每个程序的执行起点。标识符 main 后面的圆括号表示它是一个被称为**方法**（method）的程序组成单元。类声明中通常包含一个或多个方法。其中必须有一个名为 main 的方法，它称为程序的**入口点**，并且必须按照第 6 行所示的形式定义，否则程序无法执行。^①

方法用于执行特定任务，任务完成后可返回信息。我们会在 8.2.5 节阐释关键字 static 的作用。关键字 void 表示该方法不返回任何信息，后续我们会接触到有返回值的方法。现阶段编写程序时，可以照抄 main 方法的首行写法（第 6 行）。第 6 章将深入探讨圆括号中的 String[] args。

方法的左大括号（第 6 行末尾）标志着方法体^②的起始，对应的右大括号（第 8 行）则标志着方法体的结束。方法体应在左右大括号之间整体缩进一个层级，如第 7 行所示，这有助于突显方法结构，增强代码可读性。

^① 译注：Java 目前已经有了一个处于预览阶段的特性，称为“实例 main”，它可以第 6 行简化为 void main() {}。

^② 译注：你说成“方法主体”也完全没有问题。不过一般都说成“方法体”。那么，可不可以将“类主体”说成“类体”呢？答案是可以，只要程序员之间交流时能够理解就行。

使用 `System.out.println` 执行输出

第 7 行的代码用于显示双引号 (") 之间的内容，但引号本身不会被显示出来。引号连同其之间的内容被称为**字符串** (string) 或**字符串字面值** (string literal) ^①。编译器不会忽略字符串中的空白字符 (如空格、制表符等，但空行除外)。注意，使用双引号包裹的字符串不能跨越多行 (换言之，不允许换行)。^②

`System.out` 是**标准输出流对象**，由 Java 系统预定义，用于在程序运行时向标准输出设备输出文本内容。^③从技术实现来看：

- 在 Microsoft Windows 系统中，标准输出通常对应命令提示符 (Command Prompt) 或 PowerShell 窗口；
- 在 Linux/macOS 系统中，对应终端 (Terminal) 或 Shell 窗口。

许多程序员简单地将这个窗口称为“命令行”或“控制台”，其本质是通过文本交互方式执行命令的交互式环境。

`System.out.println` 方法用于显示单行文本。^④第 7 行圆括号中的字符串就是传递给该方法的**实参** (argument)。当 `System.out.println` 完成其任务后，它会将输出光标 (即下一个字符将要显示的位置) 移动到下一行的开头。这类似于在编辑器中输入文本时按下 Enter 键——光标会移动到文档中下一行的开头。

第 7 行整行内容 (含分号) 称为**语句** (statement)。方法通常由若干语句构成，用于执行具体任务。大多数语句以分号结尾。^⑤

^① 译注：字面值 (literal) 是直接输入在代码中的值，包括数字和字符串值。也称为直接量、字面量或文字常量。

^② 稍后会讨论如何利用“文本块”来格式化多行字符串。

^③ 译注：`System.out` 是 Java 中 `java.lang.System` 类的一个公共静态字段，它的类型是 `java.io.PrintStream`。这意味着 `System.out` 是一个 `PrintStream` 类的实例，你可以通过它调用 `PrintStream` 类中定义的所有方法，例如 `print()`、`println()` 和 `printf()`，从而向标准输出 (通常是控制台) 打印数据。

^④ 译注：`println` 中的 `ln` 就是 `line` (行) 的意思。

^⑤ 译注：更准确地说，Java 中的所有可执行语句 (如赋值、方法调用) 都必须以分号结尾。

提高可读性：在右大括号后使用单行注释

目前，我们暂时在每一个结束方法声明和类声明的右大括号之后都添加一条单行注释。例如，第 8 行指明了 `main` 方法的右大括号，第 9 行指明了 `Welcome1` 类的右大括号。从第 3 章开始，我们将不再使用这种结尾注释。

生成式 AI 练习

1. 请将以下代码提供给生成式 AI，并询问这段代码会输出什么内容：^①

```
System.out.println("*");
System.out.println("*****");
System.out.println("**");
```

将结果与如下输出进行对比：

```
*
*****
**
```

2.2.1 编译程序

现在可以编译并运行这个程序了。这里我们将使用 Java 开发工具包（Java Development Kit, JDK）提供的命令行工具完成操作。假设本书示例代码已按“准备工作”部分的说明，存储在用户账户“文档”文件夹^②下的 `examples` 子文件夹中。

切换到正确的文件夹

首先打开一个命令窗口（参见 1.7 节），然后切换到存储了该程序的文件夹。操作系统通常使用 `cd` 命令来切换目录。在 Windows 上，输入以下命令并按 `Enter` 键来进入 `fig02_01` 文件夹：

```
cd %userprofile%\Documents\examples\ch02\fig02_01
```

在 Linux/macOS 上，则改为使用以下命令：

```
cd ~/Documents/examples/ch02/fig02_01
```

^① 译注：本书有几百个这样的“生成式 AI 练习”，请随意选择自己喜欢的一个 AI，推荐 DeepSeek、豆包、Kimi 这些容易访问的 AI。

^② 译注：macOS 上是“文稿”文件夹。

编译.java 文件

要编译此 Java 程序，请输入以下命令并按 Enter 键执行：

```
javac Welcome1.java
```

若程序无语法错误，该命令将生成名为 `Welcome1.class` 的类文件，其中包含与平台无关的 Java 字节码，用于表示程序内容。以后使用 `java` 命令在特定平台上运行程序时，Java 虚拟机（JVM）会将这些字节码转换为底层操作系统和硬件可识别的指令

错误消息

在编译过程中，如果报告以下错误消息：

```
“类 Welcome1 是公共的，应在名为 Welcome1.java 的文件中声明”
```

这可能意味着以下两个问题之一：

- 文件名与文件中的 `public` 类名不一致；
- 或者在编译命令中拼错了类名。

有时，故意“破坏”一个正常运行的程序有助于熟悉编译器的错误提示。尽管错误消息通常会给出问题的大致方向，但未必能精准定位代码中的具体错误。例如，尝试从图 2.1 的程序中删除一个分号或大括号，然后重新编译，观察系统报告的错误消息。

若编译器提示语法错误，问题未必出现在错误消息指示的行。建议先检查报错行，若未发现问题，再向前追溯几行代码。每条编译错误消息都会显示文件名和行号，例如 `Welcome1.java:6` 表示错误发生在该文件的第 6 行，消息的其余部分会详细说明错误原因。

生成式 AI 练习

1. 从图 2.1 的程序中移除一个大括号，然后提示 AI 代码不完整，要求它找出并修复错误。将修复的代码与图 2.1 进行比较。

2.2.2 执行程序

要执行编译好的 `Welcome1` 程序，请输入以下命令并按 Enter 键：

```
java Welcome1
```

注意，执行时无需输入 `.class` 文件扩展名。此时 JVM 会启动并加载 `Welcome1.class` 文件，继而调用 `Welcome1` 类的 `main` 方法。随后，`main` 方法中的第 7 行会输出：

```
欢迎学习 Java 编程！
```

2.3 修改第一个程序

本节将修改图 2.1 的示例程序，演示如何使用多个语句输出单行文本，以及如何使用一个语句输出多行文本。

用多个语句输出单行文本

Welcome2 类（图 2.2）使用两个语句（第 7 行~第 8 行）产生如图 2.1 所示的输出。

```
1 // 图 2.2: Welcome2.java
2 // 用多个语句输出单行文本
3
4 public class Welcome2 {
5     // main 方法开始程序执行
6     public static void main(String[] args) {
7         System.out.print("欢迎学习");
8         System.out.println("Java 编程!");
9     } // 结束 main 方法
10 } // 结束 Welcome2 类
```

欢迎学习 Java 编程!

图 2.2 用多个语句输出单行文本

这个程序与图 2.1 中的程序类似，因此我们仅讨论其变化之处。第 2 行的单行注释说明了该程序的用途。第 4 行开始声明 Welcome2 类。在 main 方法中，第 7 行~第 8 行的两个语句共同构成了一行文本输出。其中，第一个语句使用 `System.out.print` 方法显示字符串。每个 `print` 或 `println` 方法都会从上一个输出位置继续显示字符，但与 `println` 不同的是，`print` 方法在显示完实参内容后不会将光标移至下一行的开头——程序后续输出的字符将紧跟在 `print` 方法最后显示的字符之后。因此，第 8 行实参中的第一个字符（字母“J”）会直接出现在第 7 行显示的最后一个字符（即字符串右引号前的“习”）之后。

用一个语句输出多行文本

利用换行符 `\n`，可在单个语句中输出多行文本——该符号会将输出光标定位到下一行开头。如图 2.3 所示的程序实现了 4 行文本输出，它的大多数内容与图 2.1 和图 2.2 中的程序基本一致。

```
1 // 图 2.3: Welcome3.java
2 // 用一个语句输出多行文本
3
4 public class Welcome3 {
```

```

5 // main 方法开始程序执行
6 public static void main(String[] args) {
7     System.out.println("欢迎\n学习\nJava\n编程!");
8 } // 结束 main 方法
9 } // 结束 Welcome3 类

```

```

欢迎
学习
Java
编程!

```

图 2.3 用一个语句输出多行文本

第 7 行实现了 4 行文本的输出。通常情况下，字符串中的每个字符会按双引号内的顺序显示，但语句中三次出现的 `\n` 字符组合并不会在屏幕上实际显示。反斜杠 `\` 是**转义字符**（escape character），对 `System.out.print` 和 `System.out.println` 方法具有特殊含义。当字符串中出现反斜杠时，它会与下一个字符组成**转义序列**（escape sequence），例如 `\n` 转义序列表示换行符（newline）。输出过程中遇到 `\n` 时，输出光标会移动到下一行开头。

表 2.2 总结了常见转义序列，完整列表可参阅《Java 语言规范》的 3.10.7 节“Escape Sequences”。^①

表 2.2 常见转义序列

转义序列	说明
<code>\n</code>	换行符，将屏幕光标移动到下一行开头
<code>\t</code>	水平制表符。将屏幕光标移动到下一个制表位
<code>\\</code>	反斜杠。输出一个反斜杠字符
<code>\"</code>	双引号。输出一个双引号字符。例如： <pre>System.out.println("\"引文\"")</pre> 将输出：“引文”

^① *The Java Language Specification*, Java SE 24 Edition, Section 3.10.7, “Escape Sequences”, <https://docs.oracle.com/javase/specs/jls/se24/html/jls-3.html#jls-3.10.7>。

生成式 AI 练习

1. 要求 AI 简化以下代码：

```
System.out.print("#");
System.out.println("###");
System.out.println("#####");
System.out.print("####");
System.out.println("##");
```

然后运行它返回的代码。输出是否如下所示？

```
####
#####
#####
```

注意，在撰写本书时，我们使用的所有 AI 都未能正确处理这个问题，即使我们尝试调整提示也没能解决。因此，不要盲目相信 AI 生成的结果，始终进行事实核验。^①

2.4 使用 printf 输出文本

`System.out.printf` 方法用于输出格式化数据，其中“`printf`”的“`f`”代表“formatted”（格式化）。图 2.4 的程序演示了如何通过 `printf` 将“欢迎学习”和“Java 编程!”分别输出至不同行。

```
1 // 图 2.4: Welcome4.java
2 // 用 System.out.printf 方法输出多行文本
3
4 public class Welcome4 {
5     // main 方法开始程序执行
6     public static void main(String[] args) {
7         System.out.printf("%s\n%s\n", "欢迎学习", "Java 编程!");
8     } // 结束 main 方法
9 } // 结束 Welcome4 类
```

欢迎学习 Java 编程!

图 2.4 用 `System.out.printf` 方法输出多行文本

^① 译注：问题在于，AI 没有意识到 `print` 和 `println` 的区别。即使你提醒它注意这个区别，输出结果也不一定正确。有的 AI 甚至没有意识到可以用一个语句输出所有这些内容。不过，AI（如 DeepSeek）在开启了“深度思考”后，基本都能给出正确的答案。

第 7 行调用 `System.out.printf` 输出程序内容。该方法调用包含三个实参。当方法需要两个或更多实参时，需将其置于逗号分隔的列表中，且每个逗号后保留一个空格以提升可读性。

`printf` 方法的第一个实参是**格式字符串**（format string），由**固定文本**（fixed text）和**格式说明符**（format specifier）组成。其中固定文本的输出规则与 `print/println` 一致，而格式说明符作为值的占位符，用于指定输出数据的类型。

格式说明符以百分号%开头，后跟表示数据类型的字符。例如，`%s` 是字符串的占位符。第 7 行的格式字符串规定 `printf` 需输出两个字符串，每个字符串后紧跟一个换行符。在本例中，第一个`%s` 会被替换为“欢迎学习”，第二个`%s` 替换为“Java 编程!”，最终程序输出两行文本。

值得注意的是，此处未使用前文介绍的换行转义符`\n`，而是采用仅能在格式字符串中使用的`%n` 格式说明符。本书后续章节将详细讲解 `printf` 的格式化输出用法，附录 B 也会深入探讨输出格式化规则。

生成式 AI 练习

1. 要求 AI 编写一个 Java `printf` 语句来输出以下内容：

```
#  
##  
###  
####  
#####
```

将 AI 生成的代码与以下语句进行对比：^①

```
System.out.printf("#%n##%n###%n####%n#####%n");
```

2.5 另一个程序：整数相加

图 2.5 的程序读取（或称“输入”）两个整数（如-22、7、0 和 1024 等），把它们存储到变量中，求和并显示结果。注意，在程序之后展示的输出结果中，由用户输入的内容（本例

^① 译注：作者在这里给出的代码是 AI 的首选解决方案，使用了“`printf` + 字符串拼接”技术，只是和图 2.4 演示的不一致。许多 AI 会同时提供多个解决方案，其中一个可能就是图 2.4 采用的“纯 `printf` 格式控制”方案。

是 45 和 72) 加粗显示。

```
1 // 图 2.5: Addition.java
2 // Addition (加法) 程序要求用户输入两个整数, 求和并显示结果
3 import java.util.Scanner; // 程序使用了 Scanner 类
4
5 public class Addition {
6     // main 方法开始程序执行
7     public static void main(String[] args) {
8         // 创建一个 Scanner 对象从用户获取输入
9         Scanner input = new Scanner(System.in);
10
11         System.out.print("输入第一个整数: "); // 提示
12         int number1 = input.nextInt(); // 从用户处读取第一个数
13
14         System.out.print("输入第二个整数: "); // 提示
15         int number2 = input.nextInt(); // 从用户处读取第二个数
16
17         int sum = number1 + number2; // 求和并将结果存储到变量 sum 中
18
19         System.out.printf("两数之和是%d\n", sum); // 显示 sum
20     } // 结束 main 方法
21 } // 结束 Addition 类
```

```
输入第一个整数: 45
输入第二个整数: 72
两数之和是 117
```

图 2.5 要求用户输入两个整数, 求和并显示结果

import 语句

Java 的一大优势在于拥有丰富的预定义类集合, 可以直接复用这些类而无需“重复造轮子”。这些类按功能组织成**包** (package) ——即具有相关性的类的命名集合。而所有包又统称为 **Java 应用程序编程接口** (Java Application Programming Interface, Java API)。

第 3 行的 import 语句指示编译器定位程序中要使用的类。本例表明程序使用了 java.util 包中的预定义 Scanner 类 (稍后会详细讲解), 进而确保编译器能正确校验类的使用方式。

所有 import 语句都必须出现在文件中的类声明之前。若将 import 语句置于类内部或类声明之后, 就会导致语法错误。若遗漏必要的 import 语句, 编译时会报错。此时需检查 import 语句是否正确, 且类名拼写 (包括大小写) 是否准确。

声明并创建一个 Scanner 来获取用户输入

变量必须先声明后使用。声明变量时需指定名称和类型。程序可通过变量名来访问内存中

存储的值。按照惯例，变量名使用驼峰命名法（camel-case），即首字母小写，后续单词首字母大写，例如：`firstNumber`。^①变量名可以是任何合法标识符——由字母、数字、下划线（`_`）和美元符号（`$`）组成的一串字符，不能以数字开头，也不能包含空格。变量的类型决定了该变量在内存中存储的数据种类。和其他语句一样，变量声明语句也以分号（`;`）结尾。

`main` 中的第 9 行就是一个变量声明语句，它声明了 `Scanner` 类型的一个名为 `input` 的变量。`Scanner`（来自 `java.util` 包）允许程序读取数据（如数字和字符串），数据来源可以是用户输入或者文件。在使用 `Scanner` 之前，必须先创建它的一个对象（此处为 `input`）并指定数据来源。

第 9 行的等号（`=`）表示对变量 `input` 进行初始化，等号右侧表达式 `new Scanner(System.in)` 为初始值。该表达式通过关键字 `new` 创建 `Scanner` 对象，用于读取用户的输入。`System.in` 是标准输入对象，使程序能读取用户输入的字符数据，`Scanner` 会将这些数据转换为程序可用的数据类型（如 `int`）。

提示用户输入

第 11 行使用 `System.out.print` 方法向用户显示提示消息“输入第一个整数：”，旨在引导用户输入数据。值得注意的是，这里我们选用 `print` 而非 `println`，这样做的好处是用户的输入会直接显示在提示消息的同一行，从而使输出界面更为整洁。此外，提示消息在冒号后特意添加了一个空格，进一步提升了程序输出的可读性。

如 2.2 节所述，以大写字母开头的标识符通常代表类名。这里的 `System` 类便是其中之一，它来自 `java.lang` 包。Java 语言的特性之一是会自动将 `java.lang` 包导入到每一个程序中，因此，`java.lang` 包中的类是唯一无需通过 `import` 语句显式导入的 Java API 类。

声明变量以存储整数并从用户获取一个整数

第 12 行声明一个 `int` 类型的 `number1` 变量来存储整数值，例如 72、-1127 和 0 等。`int` 类型的取值范围是 -2 147 483 648 到 +2 147 483 647。注意，Java 程序中的整数不能包含空格或逗号等千位分隔符。例如，代码中逗号的作用是分隔列表中的各项（参考第 19 行向 `System.out.printf` 方法传递的实参）。若想提升可读性，可以用下划线（`_`）作为数字分隔符，例如用 `60_000_000` 表示 60000000。^②

^① 译注：更准确地说，这是“小驼峰命名法”。

^② 译注：只能在代码中这样写。用户在终端窗口输入一个整数时，不能输入 `60_000_000`，只能输入 `60000000`。

`float` 和 `double` 类型用于表示实数（带小数点的数字），如 3.4、0.0 和 -11.19。`char` 类型则用于表示字符数据。它可以表示如下所示的数据：

- 一个字母（如 A 或 z）；
- 一个数字（0~9）；
- 一个特殊字符（如 * 或 %）；
- 或者一个转义序列（如 `\t` 表示制表符，`\n` 表示换行符）。

`int`、`float`、`double` 和 `char` 都是**基元类型**。^①它们的名字都是语言的“关键字”，必须全部小写。其他基元类型还有：`boolean`、`byte`、`short` 和 `long`。关于各种数值基元类型的取值范围，可以参阅《Java 语言规范》的 4.2 节“Primitive Types and Values”。^②

第 12 行的等号（=）表示变量 `number1` 应在声明时通过 `input.nextInt()` 初始化。换言之，使用 `Scanner` 对象 `input` 的 `nextInt` 方法来获取用户输入的一个整数。此时，程序会等待用户输入一个数字，并按 Enter 键提交给程序。

我们的程序目前假设用户会输入一个有效整数。如果输入无效，程序将遭遇逻辑错误并终止。在后续章节中，我们会讨论如何让程序变得更健壮，使其能够处理这类错误，从而实现容错机制。

获取第二个整数

第 14 行提示用户输入第二个整数。第 15 行声明 `int` 变量 `number2`，并将其初始化为从用户那里读取的另一个整数。

在计算中使用变量

第 17 行声明 `int` 变量 `sum`，并将其初始化为 `number1 + number2` 的结果。这里的 + 是二元

^① 译注：本书使用“基元类型”而不是“基本类型”。“基元类型”（`primitive type`）是指语言内置的、不可再分的类型。一个基元值不会与其他基元值共享状态。它们是语言的核心部分，不需要借助外部或自定义的类即可直接使用。提到“基元类型”时，通常暗示这些类型具有高效的性能，因为它们直接由编译器或运行时环境支持，而不需要额外的封装或对象开销。相反，如果说成“基本类型”，有时会产生歧义。

^② *The Java Language Specification*, Java SE 24 Edition. Section 4.2, “Primitive Types and Values”, <https://docs.oracle.com/javase/specs/jls/se24/html/jls-4.html#jls-4.2>。

操作符，因其有两个操作数：`number1` 和 `number2`。^①语句中包含计算的部分被称为**表达式**（`expression`）。事实上，表达式是指语句中具有一个值的任何部分。例如，表达式 `number1 + number2` 的值就是两个 `int` 变量之和。类似地，表达式 `input.nextInt()`（第 12 行和第 15 行）的值就是用户输入的整数。在二元操作符两侧添加空格可以使操作符更醒目，使程序更易读。

显示计算结果

完成计算后，第 19 行通过 `System.out.printf` 方法输出变量 `sum` 的值。其中，格式说明符 `%d` 是一个用于 `int` 类型值的占位符，字母 `d` 代表“十进制整数”，即 `decimal (base 10) integer`。格式字符串中除了 `%d` 之外的其他部分都是固定文本。因此，`printf` 方法会先输出“两数之和是”，紧接着输出 `sum` 变量的值（替换了 `%d` 的位置），最后输出一个换行符。^②

实际上，第 17 行（求和）和第 19 行（输出）可以合并为一个更简洁的语句：

```
System.out.printf("两数之和是%d\n", (number1 + number2));
```

围绕表达式 `number1 + number2` 的圆括号在此处并非必需，其作用仅仅是为了强调整个表达式的计算结果将作为 `%d` 占位符的替代值进行输出。

Java API 文档

我们引入的每个新的 Java API 类都注明了它所属的包名。这些信息有助于你在 Java API 文档中查找对应的包和类描述。Java API 在线文档地址：^③

<https://docs.oracle.com/en/java/javase/24/docs/api/index.html>

以下网页概述了如何使用 API 文档：

^① 译注：`operator` 在本书均称为“操作符”，而不是“运算符”。类似地，`operand` 称为“操作数”，而不是“运算子”。

^② 译注：2.4 节讲过，换行转义符本应写成 `\n`，它只有在格式字符串中才能写成 `%n`。`%n` 是格式字符串唯一以 `%` 开头但不是格式说明符的“固定文本”。

^③ 译注：目前甲骨文（Oracle）官方未提供完整 Java API 中文文档。虽然 Sun 公司（后被甲骨文收购）曾组织过部分核心类库的中文翻译工作，且存在第三方社区提供的中文资源，但这些翻译往往存在更新滞后的问题。考虑到 Java 技术迭代迅速，建议开发者优先使用官方英文文档（内容准确且实时更新），可借助现代翻译工具辅助阅读，逐步培养直接查阅英文技术文档的习惯。另外，当你拿到本书中文版时，应该可以将文档网址中的 24 更换为 25 了。

<https://docs.oracle.com/en/java/javase/24/docs/api/help-doc.html#overview>

也可以下载文档以便离线使用：

<https://www.oracle.com/java/technologies/javase-jdk24-doc-downloads.html>

用不同的语句声明和初始化变量

每个变量在参与计算（或其他表达式）之前，都必须已经存储了一个值。第 12 行的语句既声明了变量 `number1`，又用用户输入的值初始化了它。有的时候，我们也可以选择先在一个语句中声明变量，然后在另一个语句中给它赋值。

例如，第 12 行可以改写成以下两个语句：

```
int number1; // 声明 int 变量 number1
number1 = input.nextInt(); // 将用户输入的整数赋给 number1
```

第一个语句仅仅声明了 `number1` 这个变量，但并没有给它任何初始值。第二个语句则使用了赋值操作符（`=`）将用户输入的整数赋给了 `number1`。可以把这个语句理解为“`number1` 得到了 `input.nextInt()` 的值”。

请记住，赋值操作前，等号右侧的所有表达式都会先被求值，然后才会将结果存储到左侧的变量中。

生成式 AI 练习

1. 要求 AI 编写一个 Java 程序，该程序从用户处获取三个整数，计算它们的和并显示结果。将生成的代码与本节的加法程序对比，运行并验证结果。若出现错误，将错误消息发送给 AI 并要求修正，然后重新测试。
2. 要求 AI 创建一个简明教程，介绍如何使用 Java API 文档

2.6 算术运算

表 2.3 总结了算术操作符。

表 2.3 算术操作符

运算	算术操作符	代数式	Java 语言的表达式
----	-------	-----	-------------

加	+	$f + 7$	$f + 7$
减	-	$p - c$	$p - c$
乘	*	bm 或 $b \cdot m$	$b * m$
除	/	$x / y, \frac{x}{y}$ 或 $x \div y$	x / y
求余	%	$r \text{ mod } s$	$r \% s$

需注意，此处运用了多个代数中未出现的符号。星号（*）表示乘法，百分号（%）为求余操作符（稍后将详细讲解）。表中的算术操作符均为二元操作符——每个操作符均对两个操作数进行运算。例如，表达式 $f + 7$ 中包含二元操作符+以及两个操作数 f 和 7 。

整数除法

分子和分母都是整数的整数除法会生成一个整数商。例如，表达式 $7/4$ 的求值结果是 1 ，表达式 $17/5$ 的求值结果是 3 。整数除法结果中的任何小数部分都被截断（换言之，被丢弃），不会进行四舍五入。

求余操作符(%)

求余操作符%（又称模除操作符）用于生成除法运算后的余数。具体来说， $x \% y$ 表示 x 除以 y 的余数。例如， $7 \% 4$ 的结果为 3 ， $17 \% 5$ 的结果为 2 。该操作符主要适用于整数操作数，但也可用于其他算术类型。其常见应用包括判断整数的奇偶性，或判断一个数是否为另一个数的倍数。

直线形式的算术表达式

算术表达式必须写成**直线形式**（straight-line form），以便于在计算机中输入程序。因此，像“ a 除以 b ”这样的表达式必须写成 a / b ，这样所有的常量、变量和操作符都呈直线排列。以下代数式通常不为编译器所接受：

$$\frac{a}{b}$$

用圆括号对子表达式进行分组

圆括号在 Java 表达式中的用法和在代数式中一样。例如，为了求 b 与 c 之和，结果再与 a 相乘，可以写成 $a * (b + c)$ 。如果表达式包含嵌套圆括号，例如 $(a * (b + c))$ ，那么最内层圆括号中的表达式（本例是 $b + c$ ）会首先求值。

操作符优先级规则

Java 在算术表达式中应用各个操作符的顺序由以下**操作符优先级规则**决定，这些规则大体与代数中的规则一致：

1. 首先应用乘法、除法和求余操作符。如果一个表达式包含多个这样的操作符，它们按从左到右的顺序应用。我们说这三个操作符具有相同的优先级；
2. 然后应用加法和减法操作符。如果一个表达式包含多个这样的操作符，它们按从左到右的顺序应用。加法和减法操作符也具有相同的优先级；
3. 最后应用赋值操作符。

当我们说 Java 语言从左到右应用某些操作符时，实际说的是操作符的**分组**（grouping），有时也称为操作符的**结合性**（associativity）。例如，在以下表达式中：

$a + b + c$

加法操作符（+）从左到右分组，等同于 $(a + b) + c$ 。大多数相同优先级的 Java 操作符都从左到右分组。以后会提到一些从右到左分组的操作符。表 2.4 总结了操作符优先级规则。

表 2.4 操作符优先级规则

操作符	运算	求值顺序（优先级）
* / %	乘法 除法 求余	最先求值。如果有多个此类操作符，它们将从左到右进行计算
+ -	加法 减法	其次求值。如果有多个此类操作符，它们将从左到右进行计算
=	赋值	最后求值

生成式 AI 练习

1. 要求 AI 解释 Java 在执行以下语句时操作符的求值顺序，并指出在语句执行完毕后 x 的值：

```
int x = 2 % 2 + 2 * 2 - 2 / 2;
```

验证 AI 回复的是以下求值顺序：

```
% * / + -
```

最终赋给 x 的值是 3。

2.7 决策：相等性和关系操作符

条件是一种表达式，其值可为 true（真）或 false（假）。if 语句会依据条件的值来做出决策（判断）。在 if 语句中，我们通过相等性操作符（==和!=）以及关系操作符（>、<、>=和<=）来构建条件。表 2.5 对这些操作符进行了总结。

表 2.5 相等性和关系操作符

代数运算符	Java 相等性或关系操作符	示例条件	条件的含义
<i>相等性操作符</i>			
=	==	x == y	x 等于 y
≠	!=	x != y	x 不等于 y
<i>关系操作符</i>			
>	>	x > y	x 大于 y
<	<	x < y	x 小于 y
≥	>=	x >= y	x 大于或等于 y

\leq	<code><=</code>	<code>x <= y</code>	x 小于或等于 y
--------	--------------------	------------------------	-----------

两个相等性操作符（`==`和`!=`）的优先级相同，且整体低于关系操作符（`>`、`<`、`>=`、`<=`）。相等性操作符从左向右分组（结合）。所有关系操作符也具有相同的优先级，也从左向右分组（结合）。

使用 if 语句

图 2.6 的程序使用 6 个 `if` 语句对用户输入的两个整数进行比较。若某个 `if` 语句的条件为 `true`，其主体内的输出语句就会执行；否则，将跳过该主体。程序借助 `Scanner` 获取用户输入的整数，并将其存储于变量 `number1` 和 `number2` 中。随后，利用全部 6 个相等性和关系操作符对这两个数进行比较，并且仅显示结果为 `true` 的比较情况。我们给出了用户输入不同数值组合时的三个示例输出。

```

1 // 图 2.6: Comparison.java
2 // 使用 if 语句、关系操作符和相等性
3 // 操作符比较两个整数
4 import java.util.Scanner; // 程序使用了 Scanner 类
5
6 public class Comparison {
7     // main 方法开始程序执行
8     public static void main(String[] args) {
9         // 创建一个 Scanner 对象从用户获取输入
10        Scanner input = new Scanner(System.in);
11
12        System.out.print("输入第一个整数: "); // 提示
13        int number1 = input.nextInt(); // 从用户处读取第一个数
14
15        System.out.print("输入第二个整数: "); // 提示
16        int number2 = input.nextInt(); // 从用户处读取第二个数
17
18        if (number1 == number2) {
19            System.out.printf("%d == %d\n", number1, number2);
20        }
21
22        if (number1 != number2) {
23            System.out.printf("%d != %d\n", number1, number2);
24        }
25
26        if (number1 < number2) {
27            System.out.printf("%d < %d\n", number1, number2);
28        }
29
30        if (number1 > number2) {
31            System.out.printf("%d > %d\n", number1, number2);

```

```

32     }
33
34     if (number1 <= number2) {
35         System.out.printf("%d <= %d\n", number1, number2);
36     }
37
38     if (number1 >= number2) {
39         System.out.printf("%d >= %d\n", number1, number2);
40     }
41 } // 结束 main 方法
42 } // 结束 Comparison 类

```

```

输入第一个整数: 777
输入第二个整数: 777
777 == 777
777 <= 777
777 >= 777

```

```

输入第一个整数: 1000
输入第二个整数: 2000
1000 != 2000
1000 < 2000
1000 <= 2000

```

```

输入第一个整数: 2000
输入第二个整数: 1000
2000 != 1000
2000 > 1000
2000 >= 1000

```

图 2.6 使用 if 语句、关系操作符和相等性操作符比较两个整数

在 Comparison 类的 main 方法中，第 10 行创建了一个 Scanner 对象，用于读取用户输入的数据。第 12 行~第 13 行提示用户输入第一个整数，然后读取该值并存储到 int 类型的变量 number1 中。第 15 行~第 16 行提示用户输入第二个整数，读取后存储到 int 类型的变量 number2 中。第 18 行~第 20 行通过比较 number1 和 number2 的值来判断二者是否相等，若相等则在第 19 行显示用==分隔的两个数字。从第 22 行、第 26 行、第 30 行、第 34 行和第 38 行开始的 if 语句，分别使用!=、<、>、<=和>=操作符对 number1 和 number2 进行比较。当某个 if 语句的条件为 true 时，其对应的语句块会输出一行文本，明确提示该条件的判断结果为 true。

代码块

在图 2.6 的程序中，每个 `if` 语句的主体都只有一个语句。为了提高可读性，该语句被缩进。IDE 通常会自动帮你缩进。另外要注意的是，我们将每个主体语句都用一对大括号 `{}` 包裹，这形成了一个**代码块或块**（`block`）。

在代码块仅包含单个语句时，无需使用大括号 `{}` 包裹；若代码块包含多个语句，则必须用大括号包裹。需要注意的是，若忘记为多语句代码块添加大括号，后续会引发错误。为避免此类问题，建议始终用大括号包裹 `if` 语句的代码块。

在 `if` 语句条件的右圆括号后直接添加分号，通常属于逻辑错误（虽非语法错误）。此时分号会使 `if` 语句的主体变为**空语句**（即不执行任何操作），导致无论条件是否为 `true`，`if` 语句都不会执行任何动作。更严重的是，原本看似属于 `if` 语句主体的缩进语句，实际上会在 `if` 语句之后无条件执行——这种逻辑错误常导致程序输出错误结果。

空白

在图 2.6 的程序中，请注意空白的使用规范。由于编译器会忽略大部分空白（包括空行、空格和制表符），因此可按个人习惯将语句拆分到多行，并添加适当空白，这不会改变程序语义。但需注意，标识符和字符串不可拆分。理想情况下，语句应保持简洁，但实际应用中未必总能实现。

若将语句拆分到多行，需选择自然的断点——例如逗号分隔列表的逗号之后，或长表达式的操作符之后。当语句跨越多行时，从第二行开始，每行均需与首行保持相同缩进量，直至语句结束。

生成式 AI 练习

1. 要求 AI 找出并修复以下 `if` 语句中的错误。

```
if (c < 7); {
    System.out.println("c 小于 7");
}
if (c => 7) {
    System.out.println("c 大于或等于 7");
}
```

AI 的回复应大致如下：在第一个 `if` 语句中，条件判断 `if (c < 7)` 后面多了个分号。这会让 `if` 语句的主体被认定为空语句，大括号里的 `System.out.println("c 小于 7")` 和 `if` 语句就没有关联了，不管条件是真是假，它都会执行。要解决这个问题，得把右圆括号后的分号删掉。在第二个 `if` 语句中，条件中所用的 `=>` 不是合法的关系操作符。在 Java 中，表示“大于或等于”得用 `>=`。所以得把 `=>` 替换成 `>=`，才能让这个条件判断符合语法规则。

2.8 “对象自然” 案例学习：创建和使用 Java API String 类的对象

在本书中，我们始终强调从 Java API 及 Java 开源社区的各类库中使用有价值的现有类。你将重点学习以下内容：了解可用的库有哪些，针对应用程序需求选择合适的库，利用现有库中的类创建对象，并让这些对象发挥其功能。我们所说的“对象自然”是指，即使在第 8 章学习创建自定义类之前，你也能轻松创建并使用功能强大的对象。

实际上，我们之前已经使用过 Java 对象。例如，我们曾通过 `System.out` 对象显示程序输出，还将 `Scanner` 对象与 `System.in` 结合使用来获取用户输入。这种对象封装了输入和输出机制。`System.out` 和 `System.in` 对象由 `System` 类自动创建，而 `Scanner` 对象则是通过 `new` 操作符创建的。本节将介绍如何创建和操作 Java API 中 `String` 类的对象。^①

使用 `String` 类

类自己是不能执行的。一个 `Person`（人）对象可以告诉一个 `Car`（汽车）对象需要做什么（开快点、开慢点、左转、右转等）来“驱动”这个 `Car` 对象，同时无需知道汽车的内部工作机制。同样，`main` 方法可以通过调用 `String` 对象的方法来“驱动”它，同时不必知道该类具体是如何实现的。从这个意义上说，下面这个程序中的 `main` 可以称为驱动程序（`driver program`^②）。如图 2.7 所示，`main` 函数“驱动”了几个 `String` 对象。

```
1 // 图 2.7: StringTest.java
2 // String 类的驱动/测试程序
3 public class StringTest {
4     public static void main(String[] args) {
5         String s1 = "生日";
6         String s2 = "快乐";
7         String s3 = ""; // 创建一个空字符串
8
9         // 显示三个字符串及其长度
10        System.out.printf("s1: \"%s\"; 长度: %d\n", s1, s1.length());
11        System.out.printf("s2: \"%s\"; 长度: %d\n", s2, s2.length());
12        System.out.printf("s3: \"%s\"; 长度: %d\n\n", s3, s3.length());
13
14        // 比较字符串
15        System.out.println("字符串比较结果:");
```

^① `String` 类的其他功能将在后续章节中逐一介绍。第 7 章将深入探讨 `String` 类，并演示其更多方法。

^② 译注：这里的 `driver` 是个双关语，也可以理解为“驾驶员”，即之前说的开车的“人”。另外，`driver program` 也可以理解为“测试程序”。8.2 节会更全面地讨论“驱动程序”或“驱动类”。

```

16     System.out.printf("s1.equals(\生日\"): %b%n", s1.equals("生日"));
17     System.out.printf("s2.equals(s1): %b%n", s2.equals(s1));
18
19     // 测试 String 的 isEmpty 方法
20     System.out.println("测试 s3.isEmpty():");
21
22     if (s3.isEmpty()) {
23         System.out.println("s3 为空; 让我们向 s3 赋值");
24         s3 = s1 + s2; // 连接 (拼接) s1 和 s2, 将结果赋给 s3
25         System.out.printf("s3: \"%s\"; 长度: %d%n", s3, s3.length());
26     }
27
28     // 测试字符串是否以 "生" 开头, 或者是否以 "乐" 结尾
29     System.out.printf("s1.startsWith(\生\"): %b%n",
30         s1.startsWith("生"));
31     System.out.printf("s2.startsWith(\生\"): %b%n",
32         s2.startsWith("生"));
33     System.out.printf("s1.endsWith(\乐\"): %b%n", s1.endsWith("乐"));
34     System.out.printf("s2.endsWith(\乐\"): %b%n", s2.endsWith("乐"));
35 } // 结束 main 方法
36 } // 结束 StringTest 类

```

```

s1: "生日"; 长度: 2
s2: "快乐"; 长度: 2
s3: ""; 长度: 0

```

字符串比较结果:

```

s1.equals("生日"): true
s2.equals(s1): false

```

测试 s3.isEmpty():

```

s3 为空; 让我们向 s3 赋值
s3: "生日快乐"; 长度: 4

```

```

s1.startsWith("生"): true
s2.startsWith("生"): false
s1.endsWith("乐"): false
s2.endsWith("乐"): true

```

图 2.7 String 类的驱动/测试程序

创建 String 对象

对于每一个字符串字面值, Java 编译器都会创建 Java API String 类的一个对象, 并用该字

面值中的字符来初始化它。^①因此，第 5 行~第 7 行创建了三个 `String` 对象：`s1` 包含"生日"；`s2` 包含"快乐"；`s3` 包含空字符串（""）。^②在之前的示例中声明 `int` 变量时，编译器清楚 `int` 的含义——它是 Java 内置的基元类型。同理，编译器也清楚 Java API 中 `String` 类的相关特性，例如如何从字符串字面值创建 `String` 对象。

和 `System` 类一样，`String` 类也属于 `java.lang` 包，因此会自动导入到每个程序中。当类被正确打包后，其他程序员便可以重复使用这些类。这就是使用 Java 这种支持面向对象编程的语言的最大优势之一——它们拥有丰富且强大的预定义类库。例如，只需用 `import` 语句导入 Java API 中的类，任意程序都可以复用它们，具体可以参考图 2.5 和图 2.6 中 `Scanner` 类的运用。

String 的 length 方法

第 10 行~第 12 行输出每个字符串及其长度。`String` 类的 `length` 方法用于返回特定 `String` 对象中存储的字符数量。^③若要为某个具体对象调用该方法，需按以下格式：先指定对象名（如 `s1`），接着是点操作符（`.`）^④，然后是方法名（`length`）和一对圆括号。注意，一对空圆括号表明 `length` 方法无需额外实参即可完成其任务。

比较 String 对象

可以使用 `String` 类的 `equals` 方法将字符串与其他字符串进行比较，该方法返回 `true` 或 `false`：^⑤

- 第 16 行将 `s1` 的内容与字符串字面值"生日"进行比较——两个字符串包含相同的字符，

^① 其他大多数类都要求使用 `new` 操作符来创建它们的对象。图 2.5 和图 2.6 在创建 `Scanner` 类的对象时，就是这样做的。

^② 译注：这里的“空”是指“empty”，而不是“null”。虽然在中文里也说成“空”，但 `null` 在语言中有特殊的含义。简单地说，`empty` 字符串是长度为 0 的一个真实存在的 `String` 对象，而 `null` 字符串是一个指向 `String` 对象的引用变量，其值为 `null`。这意味着这个引用变量当前没有指向内存中的任何 `String` 对象。

^③ 译注：对于常见英文和中文文本，`String.length()` 返回的值通常等于字母或汉字数量。但如果字符串包含代理对（如部分生僻字、表情符号），返回值可能大于实际字符数。

^④ 译注：也称为“成员访问操作符”。

^⑤ 注意，不能使用 `==` 操作符比较 `String` 对象的内容。以后在讲解面向对象编程时，会具体解释原因。（译注：简单地说，`==` 操作符比较的两个对象的“地址”，而不是其实际内容。）

因此 `equals` 返回 `true`。

- 第 17 行将 `s2` 的内容与 `s1` 的内容进行比较——两个字符串包含不同的字符，因此 `equals` 返回 `false`。

注意，字符串比较区分大小写。^①第 16 行和第 17 行的 `printf` 调用使用了 `%b` 格式说明符，这是一个用于条件值的占位符。^②该格式说明符会使 `printf` 将条件值显示为单词 `"true"` 或 `"false"`。

String 的 isEmpty 方法

第 22 行调用了 `String` 的 `isEmpty` 方法，如果字符串为空（即字符串长度为 0），那么该方法会返回 `true`。由于对象 `s3` 被初始化为空字符串（`""`），所以它确实为空，因此会执行 `if` 的主体语句。

字符串连接

第 24 行使用 `+` 操作符使两个字符串 `s1` 和 `s2` “相加”，并将结果赋给 `s3`——这被称为**字符串连接**（或称**字符串拼接**）。在赋值完成后，`s3` 将包含 `s1` 的字符加上 `s2` 的字符，也就是“生日快乐”。第 25 行输出 `s3` 的内容和长度，以证明字符串连接成功。

String 的 startsWith 和 endsWith 方法

第 29 行~第 34 行演示了 `String` 的 `startsWith` 和 `endsWith` 方法，如果字符串以指定子字符串开头或结尾，那么它们分别返回 `true`；否则分别返回 `false`。这些字符串比较也是区分大小写的：

- 第 30 和 32 行证明 `s1` 以子字符串“生”开头，但 `s2` 不以“生”开头。
- 第 33 和 34 行证明 `s1` 不以“乐”结尾，但 `s2` 以“乐”结尾。

生成式 AI 练习

1. 要求 AI 写一个 Java 程序，提示用户分别输入姓氏和名字，然后输出“欢迎学习 Java，”后

^① 第 7 章会讲到，`String` 执行的是“字典序”比较，基于字符串中每个字符的数值形式（更准确地说，是它们的 `Unicode` 值）。

^② 译注：所谓“条件值”，就是条件表达式的值。`%b` 中的“b”代表 `boolean`（布尔值），`%b` 格式说明符将值格式化为布尔值形式的字符串，即 `"true"` 或者 `"false"`。

跟用户输入的姓氏、名字和一个感叹号 ("!"). 运行生成的代码, 确保它能正常工作。生成式 AI 引入了 `Scanner` 哪个新的方法来读取用户输入的字符串?

2.9 小结

本章创建了一些非常基本的 Java 程序。首先, 我们通过 `System.out` 的 `print`、`println` 和 `printf` 方法在屏幕上显示信息; 接着, 加法程序借助 `Scanner` 对象来获取用户输入。在此期间, 我们讨论了算术操作符 (+、-、*、/和%), 以及操作符优先级规则如何决定表达式中的各个操作符的求值顺序。我们简要提到了其他基元类型。此外, 还介绍了 `if` 语句的用法——通过相等性操作符 (==和!=) 及关系操作符 (<、<=、>和>=) 构成条件, 以此实现逻辑判断。

最后, 我们通过创建功能强大的 Java API 类 `String` 的对象并与之交互, 介绍了“对象自然”学习方法。整章借助 AI 增强了 Java 学习体验。后续章节将创建并使用许多现有类的对象, 以极简的代码完成重要任务。第 8 章~第 9 章将利用 Java 的特性来构建有价值的自定义类。展望第 3 章, 我们将介绍控制语句, 这些语句指定了程序执行操作的顺序。

专题：生成式 AI 练习

对于以下每一个问题, 请尝试多个生成式 AI, 以确定哪一个表现最佳。始终验证其输出结果。请记住, 它们有时会犯错, 甚至产生“幻觉”, 杜撰出虚假的信息, 因此可能生成不可靠的内容。

2.1 (添加 Java 代码注释) 要求 AI 为未注释的代码添加注释, 然后检查注释是否贴切。

2.2 (优化 Java 代码注释) 要求 AI 优化 Java 程序中的注释。

2.3 (对比自己的代码和 AI 生成的代码) 对于你编写的每个程序, 都要求生成式 AI 实现相同功能。将其生成的代码自己写的代码对比。请告知 AI 你目前已学到的 Java 特性, 并要求其代码仅限于使用这些特性。

第 2 部分 编程基础

第 3 章 控制语句（上）

学习目标：

- 使用 `if` 和 `if...else` 选择语句实现分支逻辑；
- 使用 `while` 循环语句实现程序指令的重复执行；
- 掌握计数器控制循环、哨兵值控制循环及嵌套控制结构的使用；
- 熟练运用复合赋值操作符及递增/递减操作符；
- 理解基元数据类型的跨平台特性；
- 继续利用 AI 增强 Java 学习体验；
- 通过一个案例学习继续我们的“对象自然”方法，创建和操作任意大小的整数。

3.1 导读

本章和第 4 章将介绍结构化编程的理论和原则。这些概念在构建类和操作对象方面至关重要。

我们对 `if` 语句展开了更细致的探讨，同时介绍了 `if...else` 和 `while` 语句。还引入了复合赋值操作符，以及递增、递减操作符。另外，探讨了 Java 基元数据类型的可移植性问题。最后，将延续“对象自然”方法，通过一个案例学习来演示如何创建和操作超大的整数（这类整数可表示的值，突破了 Java 基元整数类型所支持的有限取值范围）。

3.2 控制结构

如第 2 章所述，程序中的语句通常按照编写顺序依次执行，这一过程称为**顺序执行**（`sequential execution`）。而 Java 的各类语句允许指定下一个执行语句不必是序列中的下一个语句，此为**控制转移**（`transfer of control`）。

20 世纪 60 年代，人们逐渐认识到，无节制使用控制转移是软件开发团队面临诸多难题的根源。当时大多数编程语言中使用的 `goto` 语句成为众矢之的，该语句允许将控制转移至程序中任意位置。^①

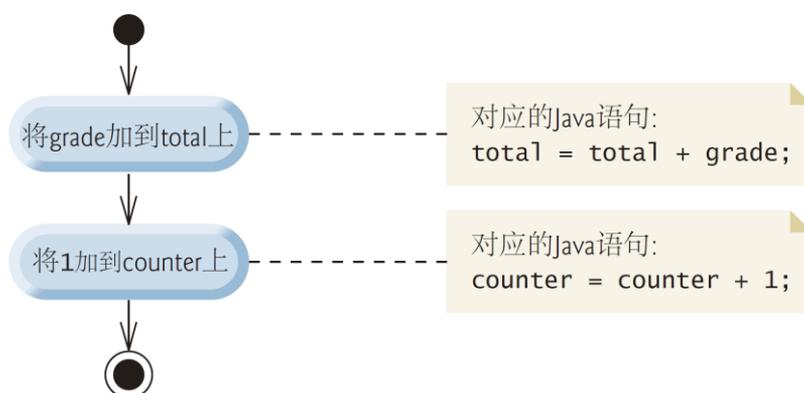
^① Java 没有 `goto` 语句；但是，`goto` 被 Java 保留，不应在程序中用作标识符。使用它会导致编译错误。

伯姆和贾可皮尼的研究表明，程序完全可以在不使用任何 `goto` 语句的情况下编写。那个时代的程序员面临着向“无 `goto` 编程”风格转变的挑战，“结构化编程”一词几乎成为“消除 `goto`”的代名词。直到 20 世纪 70 年代，大多数程序员才开始认真对待结构化编程，其成果令人瞩目。软件开发团队报告称，开发周期缩短，系统按时交付率提升，软件项目预算内完成率更高。这些成功的关键在于，结构化程序更清晰、更易于调试和修改，且从根本上更不易出错。

根据伯姆和贾可皮尼的研究成果，所有程序仅需三种控制结构即可编写——顺序结构、选择结构和循环结构。^①在介绍 Java 控制结构的实现时，我们将采用《Java 语言规范》的术语，将其称为“控制语句”。

3.2.1 顺序结构

顺序结构 (sequence structure) 是 Java 的内置特性。在无特殊指令的情况下，计算机将按照 Java 语句的编写顺序依次执行，即遵循顺序执行逻辑。以下 UML 活动图展示了一个典型的顺序结构，其中的两个计算是按顺序进行的。



Java 允许在顺序结构中采取任意数量的行动（也称为操作）。很快就会看到，在任何可以放置一个行动的地方，都可以按顺序放置多个行动。

UML 活动图建模的是软件系统中某个部分的 **workflow** (workflow)。这样的工作流可能是算法的一部分，就像上图的顺序结构。活动图由多种符号组成，例如行动状态符号（圆角矩形）、菱形和小圆圈。这些符号用**过渡箭头** (transition arrow) 连接，代表活动的流程，也就是行动/操作的顺序。

活动图有助于开发和表示算法，能清晰展示控制结构的运行机制。在第 3 章和第 4 章中，

^① 译注：这就是结构化程序理论，也称为伯姆-贾可皮尼理论或 Böhm-Jacopini 理论，1966 年在科拉多·伯姆 (C. Bohm) 及朱塞佩·贾可皮尼 (G. Jacopini) 的论文中提出。论文网址是 <https://dl.acm.org/doi/10.1145/355592.365646>。

我们将使用 UML 活动图来展示各种控制语句的工作流（也称为控制流）。

上述活动图包含两个**行动状态**（action state），每个都包含指定了具体操作的一个**行动表达式**（action expression），例如“将 grade 加到 total 上”或者“将 1 加到 counter 上”。其他可能的行动还有计算或输入/输出等。活动图中的箭头代表**过渡**（transition），即行动状态所代表的行动的发生顺序。程序为了实现如图所示的操作，会先将 grade（成绩）累加到 total（总分）上，再对 counter（计数器）执行加 1 操作。

活动图顶部的“实心圆”（●）代表**初始状态**（initial state）——即程序在执行所建模的行动之前，工作流的开始。图底部的“空心圆包实心圆”（◎）代表**最终状态**（final state），即程序执行完操作之后，工作流的结束。

顺序结构活动图还包括右上角折叠的矩形。这些是 UML 注释（类似于 Java 注释），它对图中符号的作用进行解释。一条虚线将每个注释与它所描述的元素连接起来。上图的 UML 注释描述了该图与每个行动状态的 Java 代码的关系。注意，我们一般情况下不在活动图中展示 Java 代码。要想更多地了解 UML，请访问 <https://uml.org>。

3.2.2 选择语句

Java 有三种类型的**选择语句**（selection statement）。其中，if 语句（3.3 节）在条件为 true（真）时执行（选择）一个行动（或行动组），在条件为 false（假）时跳过该行动（或行动组）。if...else 语句（3.4 节）在条件为 true 时执行一个行动（或行动组），在条件为 false 时执行另一个行动（或行动组）。switch 语句（4.8 节）根据一个表达式的值，从多个不同的行动（或行动组）中选择一个来执行。

if 语句被称为**单选语句**（single-selection statement），因其选择或忽略了一个行动（或行动组）。if...else 语句被称为**双选语句**（double-selection statement），因其在两个不同的行动（或行动组）之间选择。switch 语句则被称为**多选语句**（multiple-selection statement），因其从多个不同的行动（或行动组）中选择一个。

3.2.3 循环语句

Java 提供了 4 种用于在**循环继续条件**（loop-continuation condition）为 true 时重复执行语句的**循环语句**（iteration statement），包括 while（3.5 节）、for（4.3 节）、do...while（4.7 节）和增强 for（6.13 节）。while 和 for 语句执行其主体内的行动（或行动组）零次或多次。如果循环继续条件一开始就为 false，那么它的行动（或行动组）一次都不会执行。相反，do...while 语句执行其行动（或行动组）至少一次。

关键字

if、else、switch、while、do 和 for 都是 Java 关键字。我们已经在 2.2 节展示了完整的

Java 关键字列表。同时也可以参见《Java 语言规范》的 3.9 节“Keywords”。^①

3.2.4 控制语句小结

Java 语言只有三种控制结构，从现在起，我们把它们称为**控制语句**：

- 顺序
- 选择（if、if...else 和 switch）
- 循环（while、for、do...while 和增强 for）

每个控制语句均可通过活动图建模，图中包含代表控制语句入口和出口的初始状态与最终状态。**单入/单出**（single-entry/single-exit）控制语句通过将两个控制语句的出口与下一个控制语句的入口连接，即采用控制语句**堆叠**（stacking）技术，可轻松构建具有良好可读性的程序。除此之外，控制语句的另一种连接方式是控制语句**嵌套**（nesting），即将一个控制语句置于另一个控制语句内部。

3.3 if 单选语句

2.7 节简单介绍了 if 单选语句。程序使用选择语句在备选的行动路线中进行选择。例如，假设一次考试的及格线是 60 分，以下 Java 语句判断条件 `grade >= 60` 是否为 `true`。

```
if (grade >= 60) {  
    System.out.println("及格");  
}
```

如果条件为 `true`，就打印“及格”，然后顺序执行下个语句。如果条件为 `false`，就忽略主体中的输出语句，顺序执行下个语句（若存在）。这个选择语句第二行的缩进虽然可选，但强烈建议加上以提高程序的可读性。

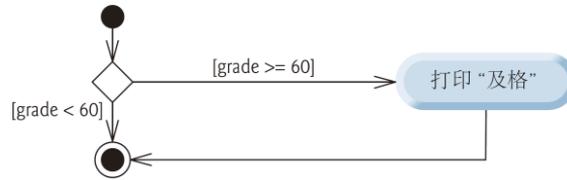
boolean 数据类型

在 2.7 节中，我们用关系或相等性操作符创建条件。Java 中的条件必须求值为 `true` 或 `false`，两者均为 Java 关键字。Java 专门提供了 `boolean` 基元数据类型，这种类型的变量只能容纳 `true` 和 `false` 两个值之一。

^① *The Java Language Specification*, Java SE 24 Edition, Section 3.9, “Keywords”, <https://docs.oracle.com/javase/specs/jls/se24/html/jls-3.html#jls-3.9>。

if 语句的 UML 活动图

下图展示了上述单选 if 语句的控制流。



这张图包含了活动图中最重要的符号：菱形，或者称为**决策符号**（decision symbol），它表示要做出一个决定。 workflow 沿着与该符号关联的**保护条件**（guard condition）所决定的路径继续。保护条件要么为 `true`，要么为 `false`。每个从决策符号引出的过渡箭头都附有一个保护条件（在箭头旁边的方括号中指定）。保护条件为 `true`，workflow 就进入过渡箭头所指向的行动状态。上图显示，如果成绩（grade）高于或等于 60 分（即条件为 `true`），程序会打印“及格”，然后过渡到活动的最终状态。如果成绩低于 60 分（即条件为 `false`），程序则立即过渡到最终状态，而不打印消息。if 语句是一种单入/单出的控制语句。

3.4 if...else 双选语句

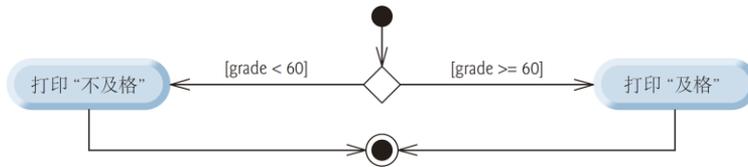
只有当条件为 `true` 时，if 单选语句才执行指定的行动。if...else 双选语句则允许当条件为 `true` 时执行一个行动，当条件为 `false` 时执行另一个行动。例如，以下 Java 语句在 grade 大于或等于 60 时打印“及格”，在小于 60 时打印“不及格”。

```
if (grade >= 60) {
    System.out.println("及格");
}
else {
    System.out.println("不及格");
}
```

无论何种情形，打印操作完成后，程序将按顺序执行下一个语句（若存在）。为了提升代码可读性，需在程序中统一采用缩进规范：当存在多个缩进层级时，每级缩进的空格数应保持一致。本书示例以三个空格作为一个缩进层级，且集成开发环境（IDE）通常会自动完成代码缩进。

if...else 语句的 UML 活动图

下图展示了上述 if...else 语句的控制流。



同样地，该 UML 活动图从初始状态开始，包含过渡箭头、决策和行动状态，并以一个最终状态结束。

生成式 AI 练习

1. 向 AI 提供以下代码，要求它识别并修复错误：

```

if (count > 10) {
    System.out.println("count 大于 10");
}
else
    System.out.println("count 小于或等于 10");
}
  
```

请确认 AI 识别出了 `else` 关键字后缺少左大括号`{`这一语法错误。

3.4.1 嵌套 if...else 语句

程序可以将 `if...else` 语句放到其他 `if...else` 语句中来测试多个条件，这样创建的是**嵌套 if...else 语句**。例如，以下嵌套 `if...else` 语句对考试成绩大于或等于 90 分的学生打印“A”，对 80~89 分的学生打印“B”，对 70~79 分的学生打印“C”，对 60~69 分的学生打印“D”，对其他所有成绩打印“F”。我们用阴影来强调嵌套。

```

if (grade >= 90) {
    System.out.println("A");
}
else {
    if (grade >= 80) {
        System.out.println("B");
    }
    else {
        if (grade >= 70) {
            System.out.println("C");
        }
        else {
            if (grade >= 60) {
                System.out.println("D");
            }
            else {
                System.out.println("F");
            }
        }
    }
}
}
  
```

如果变量 `grade` 大于或等于 90，那么嵌套 `if...else` 语句中的前 4 个条件都为 `true`，但只有第一个 `if...else` 语句的 `if` 主体语句会执行。在该语句执行后，“最外层” `if...else`

语句的整个 `else` 部分会被跳过。上述 `if...else` 语句也可以写成以下形式：

```
if (grade >= 90) {
    System.out.println("A");
}
else if (grade >= 80) {
    System.out.println("B");
}
else if (grade >= 70) {
    System.out.println("C");
}
else if (grade >= 60) {
    System.out.println("D");
}
else {
    System.out.println("F");
}
```

后一种写法避免了向右的深度缩进，这种缩进往往会占用源代码行的大量空间，导致不得不换行，从而使代码可读性变差。在编写嵌套 `if...else` 语句时，要确保已经测试了所有可能的情况。

3.4.2 代码块

若 `if` 或 `else` 的主体包含多个语句，那么需要用大括号`{}`将其包裹。如 2.7 节所述，大括号内的语句构成了**代码块**。在方法中，凡可放置单个语句的位置，均可放置包含多个语句的代码块。本书遵循的规范是始终用大括号包裹控制语句的主体，以规避所谓的“空悬 `else`”（`dangling-else`）逻辑错误。

下例在 `if...else` 语句的 `else` 子句中包含了一个多语句块：

```
if (grade >= 60) {
    System.out.println("及格");
}
else {
    System.out.println("不及格");
    System.out.println("你必须重修这门课");
}
```

如果 `grade` 小于 60，那么程序会执行 `else` 主体中的两个语句，即打印下面两行文本：

```
不及格
你必须重修这门课
```

如果不用大括号包裹 `else` 子句中的两个语句，那么以下语句：

```
System.out.println("你必须重修这门课");
```

会跑到 `else` 子句的主体外面，无论 `grade` 是否小于 60 都会执行，这属于逻辑错误。

语法错误和逻辑错误

编译器可以捕获语法错误，例如代码块中遗漏大括号{或}的情况。逻辑错误（如计算错误）则在程序运行时才显现其影响。其中，**致命逻辑错误**（fatal logic error）会导致程序失败并提前终止；^①**非致命逻辑错误**（nonfatal logic error）则允许程序继续执行，但会导致程序产生错误的结果。

空语句

在任何可以放置单个语句的地方，都可以放置一个代码块。类似地，在任何可以放置单个语句的地方，都可以放置一个空语句。如 2.7 节所述，空语句就是一个分号（;），它没有任何作用。

若在 `if` 或 `if...else` 语句的条件后误加分号（;），单选结构的 `if` 语句会因空语句而导致逻辑错误，而双选结构的 `if...else` 语句中若 `if` 部分包含实际代码块，则会引发语法错误。^②

生成式 AI 练习

1. 向 AI 提供以下代码并要求其识别并修复错误：

```
if (grade < 60)
    System.out.println("不及格");
else
    System.out.println("及格");
    System.out.println("你真不错! ");
```

将 AI 修正后的代码和解释与以下内容对比：由于两个缩进语句未被大括号包裹，所以 `else` 后仅第一个缩进语句属于 `else` 的主体。因此，无论学生成绩是否及格，代码都会打印“你真不错！”。此外，即使是单语句的 `if` 或 `else` 主体（如本例中的 `if` 部分），也建议用大括号包裹，以避免后续添加语句时因省略大括号而导致逻辑错误。正确代码应为：

^① 译注：例如，栈溢出（常见于未设定终止条件的无限递归）就是一种“致命逻辑错误”。一旦发生，JVM 会终止当前线程的执行（如果是主线程，则整个程序退出）。

^② 译注：若在双选（双分支）`if...else` 的条件后添加分号，如 `if (x > 10); { ... } else { ... }`，编译器会报错。因为分号在此处构成一个独立的空语句，导致 `if` 条件与后续代码块割裂，破坏了 `if...else` 的语法结构（`if` 条件后直接跟代码块时，中间不应插入独立语句）。

```
if (grade < 60) {
    System.out.println("不及格");
}
else {
    System.out.println("及格");
    System.out.println("你真不错! ");
}
```

3.4.3 条件操作符(?:)

可以使用**条件操作符** (?:) 来代替简单的 `if...else` 语句。这可以使代码更短、更清晰。条件操作符是 Java 唯一的**三元操作符** (即需要三个操作数的操作符)。操作数和?:符号共同构成一个**条件表达式** (conditional expression)。例如, 以下语句打印 `println` 方法的实参的求值结果。该实参就是一个条件表达式。

```
System.out.println(grade >= 60 ? "及格" : "不及格");
```

- ?左侧的操作数是条件, 即 `grade >= 60`, 它求值为 `true` 或 `false`;
- ?和:之间的操作数是条件为 `true` 时整个条件表达式的求值结果, 即字符串"及格"。
- :右侧的操作数则是条件为 `false` 时整个条件表达式的求值结果, 即字符串"不及格"。

所以, 上述语句用条件操作符来执行和 3.4 节第一个 `if...else` 语句一样的功能。注意, 条件操作符的优先级较低, 所以一般将整个条件表达式放到一对圆括号中。

生成式 AI 练习

1. 要求 AI 编写一个 `System.out.printf` 语句, 判断 `int` 变量 `number` 的值是奇数还是偶数, 并输出判断结果。告诉 AI 应假设变量 `number` 已存在且包含值 7。将它生成的代码与以下语句进行比较, 并确认其代码输出这里展示的结果:

```
System.out.printf("%d 是%s\n",
    number, number % 2 == 0 ? "偶数" : "奇数");
```

```
7 是奇数
```

3.5 while 循环语句

循环语句指定程序在条件为 `true` 时重复执行某些操作。下例展示了如何使用 **while 循环语句** 查找大于 50 的 3 的最小幂。以下代码执行后, 变量 `product` 将包含最终结果。

```
int product = 3;
while (product <= 50) {
    product = 3 * product;
}
```

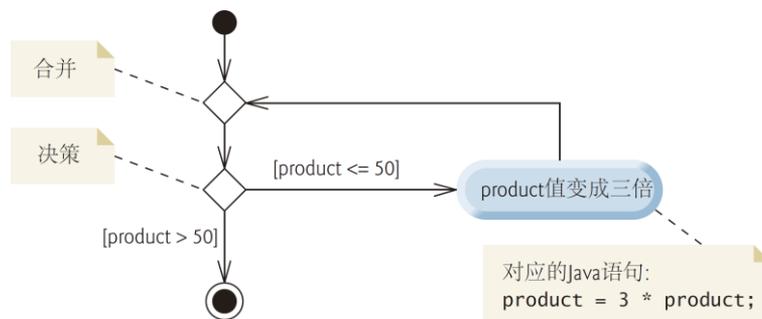
首先, 我们将 `product` 初始化为 3。然后, `while` 语句按如下方式执行:

1. 程序测试条件 `product <= 50`，因为 `product` 为 3，所以该条件表达式的求值结果为 `true`。主体语句将 `product` 乘以 3，并将结果 9 赋给 `product`。循环的第一次迭代完成；
2. 程序再次测试条件，结果为 `true`，因为 `product` 现在为 9。主体将 `product` 设为 27。循环的第二次迭代完成；
3. 程序再次测试条件，结果为 `true`，因为 `product` 现在为 27。主体将 `product` 设为 81。循环的第三次迭代完成；
4. 程序再次测试条件，结果最终为 `false`，因为 `product` 现在为 81，不符合“小于或等于 50”的条件，因此循环终止。

变量 `product` 现在包含 81——是大于 50 的 3 的最小幂。注意，`while` 语句的主体可以是单个语句，也可以是一个（代码）块。

while 语句的 UML 活动图

以下 UML 图展示了上述 `while` 循环语句的控制流。



该 UML 活动图引入了**合并符号**（merge symbol）。UML 将合并符号和决策符号都表示为菱形。合并符号将两个活动流连接成一个。在上图中，合并符号连接了来自初始状态和行动状态的过渡，使两者都流向决定循环是否应该开始（或继续）执行的决策。

可以根据进出的过渡箭头的数量来区分决策与合并符号。决策符号有一个箭头指向菱形，有两个或更多箭头离开菱形，代表该决策可能发生的状态过渡。另外，每个离开决策符号的箭头都有一个保护条件。相反，合并符号有两个或更多过渡箭头指向它，但只有一个箭头离开它，表示它合并了多个活动流以继续活动。与合并符号关联的过渡箭头都没有保护条件。^①

该图清晰呈现了 `while` 语句的迭代流程。从行动状态（圆角矩形）引出的过渡箭头指向“合并”，程序执行流随后从“合并”跳转回每次迭代起始时需检测的“决策”环节。

^① 译注：3.3 节讲过，“保护条件”的求值结果要么为 `true`，要么为 `false`。

`while` 语句持续执行，直至保护条件 “`product <= 50`” 变为 `false`，此时 “`product > 50`” 这一保护条件变为 `true`。之后，`while` 语句抵达最终状态，程序将按顺序执行后续语句。

生成式 AI 练习

1. 要求 AI 编写一个 Java 程序，找出大于一百万的第一个 2 的幂。它是否使用了我们在以下代码中展示的技术？如果不是，请相应地调整提示，以帮助 AI 生成与此处显示的代码类似的代码。^①

```
1 // 3.5 节, 生成式 AI 练习: Power.java
2 // 找出大于一百万的第一个 2 的幂
3 public class Power {
4     public static void main(String[] args) {
5         int product = 2;
6         int n = 1;
7
8         while (product <= 1_000_000) {
9             product = 2 * product;
10            n = n + 1;
11        }
12
13        System.out.printf("%s%d\n%s%d%s\n",
14            "比一百万大的第一个 2 的幂是", product,
15            "它等于 2 的", n, "次方");
16    }
17 }
```

比一百万大的第一个 2 的幂是 1048576 它等于 2 的 20 次方
--

3.6 计数器控制的循环

考虑以下问题陈述：

一个班有 10 名学生，全班进行了一次测验。你可以获得这次测验的成绩（0~100 的整数）。计算此次测验的班级平均成绩。

班级平均成绩等于所有学生成绩之和除以学生人数。程序必须输入每个学生的成绩，累加

^① 译注：具体如何调整提示？其实只需告诉 AI 你希望的输出格式（包括要显示的文本），并要求它使用 `printf` 输出即可。另外要注意的是，本书配套资源（<https://bookzhou.com>）提供了这种 AI 练习的源代码，具体目录是 `genAIexercises`。

所有成绩，求平均值并输出结果。我们使用**计数器控制的循环**（counter-controlled iteration）来一次输入一个成绩。这个技术要求使用一个称为**计数器**的变量（所以也可称为**控制变量**）来控制语句执行的次数。计数器控制的循环也称为**确定性循环**（definite iteration），因为在循环开始执行之前，循环迭代的次数是已知的。在本例中，当计数器超过 10 时，迭代终止。本节将介绍实现该算法的 Java 程序（图 3.1）。

3.6.1 实现算法

在图 3.1 的程序中，ClassAverage 类的 main 方法允许用户输入 10 个成绩，然后计算并显示平均成绩。

```
1 // 图 3.1: ClassAverage.java
2 // 使用计数器控制的循环计算班级平均成绩
3 import java.util.Scanner; // 程序使用了 Scanner 类
4
5 public class ClassAverage {
6     public static void main(String[] args) {
7         // 创建一个 Scanner 对象从用户获取输入
8         Scanner input = new Scanner(System.in);
9
10        // 初始化阶段
11        int total = 0;           // 初始化用户输入成绩之累加和
12        int gradeCounter = 1; // 初始化下一个输入的成绩的编号
13
14        // 在处理阶段，我们使用计数器控制的循环
15        while (gradeCounter <= 10) { // 循环 10 次
16            System.out.print("输入成绩: "); // 提示输入下一个成绩
17            int grade = input.nextInt(); // 输入成绩
18            total = total + grade; // 将 grade 累加到 total 上
19            gradeCounter = gradeCounter + 1; // 计数器递增 1
20        }
21
22        // 终止阶段
23        int average = total / 10; // 整数除法的结果也是整数
24
25        // 显示总成绩和平均成绩
26        System.out.printf("%n 全部 10 个成绩的总和是%d\n", total);
27        System.out.printf("班级平均成绩是%d\n", average);
28    }
29 }
```

```
输入成绩: 67
输入成绩: 78
输入成绩: 89
输入成绩: 67
输入成绩: 87
输入成绩: 98
输入成绩: 93
```

```
输入成绩：85
输入成绩：82
输入成绩：100
```

```
全部 10 个成绩的总和是 846
班级平均成绩是 84
```

图 3.1 使用计数器控制的循环计算班级平均成绩

main 方法中的局部变量

第 8 行声明并初始化 `Scanner` 变量 `input`，它用于读取由用户输入的值。第 11 行、第 12 行、第 17 行和第 23 行分别声明 `int` 变量 `total`、`gradeCounter`、`grade` 和 `average`。变量 `grade` 用于存储用户的输入。

这些声明都在 `main` 方法的主体中。在方法主体（一个代码块）内声明的变量是所谓的**局部变量**（local variable），其**作用域**（scope）从声明位置开始，到方法（代码块）的右大括号结束。所有局部变量在使用前都必须声明。局部变量不能在声明它的方法外部访问。在 `while` 循环体内声明的 `grade` 变量只能在这个代码块中使用。

初始化阶段：初始化变量 `total` 和 `gradeCounter`

第 11 行~第 12 行的赋值语句将 `total` 初始化为 0，将 `gradeCounter` 初始化为 1。这些初始化操作在变量用于计算前完成。

所有局部变量在表达式中使用前都必须初始化，否则会引发编译错误。用于存储累加结果的变量（本例是 `total`）通常初始化为 0，计数器变量（本例是 `gradeCounter`）则根据使用场景初始化为 0 或 1。本书将通过不同的示例说明 0 和 1 的具体使用时机。

处理阶段：从用户输入读取 10 个成绩

第 15 行表明，当 `gradeCounter` 的值小于等于 10 时，`while` 语句将持续循环（也称迭代）。只要该条件为 `true`，`while` 循环就会重复执行第 16 行~第 19 行的语句：

- 第 16 行显示提示消息“输入成绩:”；
- 第 17 行读取用户输入的成绩并赋给变量 `grade`；
- 第 18 行将新输入的成绩累加到 `total` 上，替换其原值；
- 第 19 行使 `gradeCounter` 递增 1（替换其原值），表示已处理一个成绩并准备好接收下一个输入。

随着 `gradeCounter` 不断递增，其值最终将超过 10，此时循环条件（第 15 行）变为 `false`，循环终止。

结束阶段：计算并显示班级平均成绩

循环终止后，会执行以下操作：

- 第 23 行计算平均值并将结果赋给变量 `average`；
- 第 26 行使用 `System.out` 的 `printf` 方法显示文本“全部 10 个成绩的总和是”，后跟 `total` 的值；
- 第 27 行使用 `printf` 显示文本“班级平均成绩是”，后跟 `average` 的值。

当执行到第 28 行时，程序终止。

3.6.2 整数除法和截断

第 23 行的平均值计算会产生整数结果。在示例输出中，成绩总和为 846，除以 10 本应得到 84.6——这是一个浮点数（带小数点的数字）。但由于 `total` 和 10 均为整数类型，第 23 行 `total / 10` 的计算结果为整数 84。两个整数相除执行的是整数除法，计算中的小数部分会被直接截断（即丢失）。下一节将介绍如何从平均值计算中获取浮点结果。若误以为整数除法是四舍五入（而非直接截断），可能导致计算结果与预期不符。

3.6.3 算术溢出

在图 3.1 的程序中，第 18 行将用户输入的每个成绩累加到 `total` 变量上：

```
total = total + grade; // 将 grade 累加到 total 上
```

即使如此简单的语句也可能出问题——若两个整数相加的结果过大，超出了 `int` 变量所能存储的范围，就会发生**算术溢出**（`arithmetic overflow`）。这会导致未定义的行为，可能会产生不可预料的结果。图 2.5 的加法程序在第 17 行也存在同样的问题，该行计算用户输入的两个 `int` 值之和：

```
int sum = number1 + number2; // 求和并将结果存储到变量 sum 中
```

在 Java 中，常量 `Integer.MIN_VALUE` 和 `Integer.MAX_VALUE` 分别表示一个 `int` 变量可以存储的最小值和最大值。其他基元数值类型也有类似的常量。可以在 Java 官方文档中查找这些常量的具体值。`Integer` 类的官方文档地址是：^①

<https://docs.oracle.com/en/java/javase/24/docs/api/java.base/java/lang/Integer.html>

^① 译注：`Integer` 类是基元类型 `int` 的**包装类**。`Integer` 类型的对象包含一个 `int` 类型的字段。此外，该类提供了多种将 `int` 转换为 `String`、将 `String` 转换为 `int` 的方法，以及处理 `int` 时有用的其他常量（如本例所说的 `Integer.MIN_VALUE` 和 `Integer.MAX_VALUE`）和方法（例如，`doubleValue()` 方法将当前 `Integer` 对象中存储的 `int` 值转换成 `double`）。

3.6.4 输入校验

每当程序从用户那里接收输入时，可能会出现各种各样的问题。例如，图 3.1 的第 17 行是：

```
int grade = input.nextInt(); // 输入成绩
```

它假设用户输入 0~100 的整数成绩。但是，用户完全可能输入：

- 小于 0 的整数；
- 大于 100 的整数；
- 超出 int 变量可存储值的范围的整数；
- 包含小数点的数字；
- 包含跟数字无关的字母或特殊符号的值。

工业级程序必须对所有可能的错误情况进行测试，以确保输入有效。输入成绩的程序应通过**范围检查**（range checking）验证输入值，确保其在 0~100 之间。对于任何超出范围的值，可要求用户重新输入。若程序需要特定值集合的输入（如非连续的产品代码），则需确保每个输入均与集合中的值匹配。

生成式 AI 练习

1. 要求 AI 修正以下代码，确保显示 1~4 的 count 值：

```
int count = 1;
while (count < 4) {
    count = count + 1;
    System.out.printf("count 的值是: %d\n", count);
}
```

将 AI 输出的结果与以下代码进行比较：

```
int count = 0;
while (count < 4) {
    count = count + 1;
    System.out.printf("count 的值是: %d\n", count);
}
```

```
count 的值是: 1
count 的值是: 2
count 的值是: 3
count 的值是: 4
```

3.7 哨兵值控制的循环

下面对 3.6 节的班级平均成绩问题进行了一般化：

开发一个求班级平均成绩的程序，每次运行都能处理输入的任意数量的成绩。

在上一个班级平均成绩例子中，问题陈述中限死了学生数量，所以有多少个成绩（10个）是事先知道的。而在一般化后，我们并不知道用户在程序执行过程中会输入多少个成绩。程序必须能处理任意数量的成绩。

解决这个问题的一个方案是使用**哨兵值**（sentinel value）来指示“数据输入结束”。用户输入所有有效的成绩，最后输入一个哨兵值，表示没有更多成绩需要输入了。

哨兵值的选择必须得当，不能和任何要实际计算的输入值混淆。成绩肯定全是非负的整数，所以-1就是一个可接受的哨兵值。例如，用户连续输入 95, 96, 75, 74, 89 和-1，程序就可以正常地计算前 5 个输入值的平均值。由于-1 是哨兵值，所以不参与实际的平均值计算。

注意，用户可能在没有输入任何有效成绩之前就输入-1，这会导致成绩数量为零。所以，必须在计算平均成绩前测试这种情况。根据 Java 标准，在整数除法中除以零会抛出运行时异常 `ArithmeticException`。^①因此，在执行除法（/）或求余（%）计算时，由于右操作数可能为零，所以应测试这种情况并予以相应的处理（例如，显示一条错误消息），而不是让计算继续。

3.7.1 实现哨兵值控制的循环

图 3.2 的程序实现了哨兵控制的循环。图 3.2 的大部分代码与图 3.1 相同，因此这里主要关注新增内容。^②虽然用户输入的每个成绩均为整数，但平均值计算结果很可能是浮点数（如 84.6），而 `int` 类型无法表示此类数值。为了存储浮点数，可以使用 `float` 和 `double` 这两种基元类型：

- `float` 变量存储的是**单精度浮点数**，其小数点右侧最多保留 7 位有效数字；
- `double` 变量存储的是**双精度浮点数**，占用内存为 `float` 类型的两倍，最多可保留 15

^① 译注：在 Java 中，浮点数除以零遵循 IEEE 754 标准，结果是：正数 / 0.0 → `+Infinity`（正无穷）；负数 / 0.0 → `-Infinity`（负无穷）；0.0 / 0.0 → `NaN`（Not a Number）。

^② 译注：若使用 Visual Studio Code 来运行本书的代码，注意由于图 3.2 使用了和图 3.1 一样的类名（`ClassAverage`），因此可能需要按 `Ctrl+Shift+P`，输入“Java:清理”，并执行“Java:清理 Java 语言服务器工作区”，消除图 3.1 的影响。

位有效数字（约为 float 类型精度的两倍）。^①

```
1 // 图 3.2: ClassAverage.java
2 // 使用哨兵控制的循环计算班级平均成绩
3 import java.util.Scanner; // 程序使用了 Scanner 类
4
5 public class ClassAverage {
6     public static void main(String[] args) {
7         // 创建一个 Scanner 对象从用户获取输入
8         Scanner input = new Scanner(System.in);
9
10        // 初始化阶段
11        int total = 0; // 初始化用户输入成绩之累加和
12        int gradeCounter = 0; // 初始化用户迄今为止输入的成绩数量
13
14        // 处理阶段
15        // 提示用户输入并读取输入的成绩
16        System.out.print("输入成绩, 或者输入-1 退出: ");
17        int grade = input.nextInt();
18
19        // 一直循环迭代, 直到用户输入哨兵值 (-1)
20        while (grade != -1) {
21            total = total + grade; // 将 grade 累加到 total 上
22            gradeCounter = gradeCounter + 1; // 计数器递增 1
23
24            // 提示用户输入下一个成绩并读取
25            System.out.print("输入成绩, 或者输入-1 退出: ");
26            grade = input.nextInt();
27        }
28
29        // 终止阶段
30        // 如果用户输入至少一个成绩...
31        if (gradeCounter != 0) {
32            // 将一个操作数转型为浮点类型, 从而执行浮点计算
33            double average = (double) total / gradeCounter;
34
35            // 显示总成绩和平均成绩 (两位小数精度)
36            System.out.printf("%n 全部%d 个成绩的总和是%d\n",
37                gradeCounter, total);
38            System.out.printf("班级平均成绩是%.2f\n", average);
```

^① 译注: 作者的说法虽然正确, 但不全面。对于 Java 的 float 和 double 类型, 有多少位有效数字由 IEEE 754 二进制浮点表示法的精度决定, 而非单纯看十进制的小数部分。例如, 执行 `System.out.println(1.12345678f)` 将输出 1.1234568, 这符合作者的说法, 小数点右侧保留了 7 位。但执行 `System.out.println(123.12345678f)` 将输出 123.12346。换言之, “有效数字”是指所有可靠数字 (包括整数和小数部分), 而非仅小数部分。

```

39     }
40     else { // 没有输入成绩, 输出对应的消息
41         System.out.println("没有输入任何成绩");
42     }
43 }
44 }

```

```

输入成绩, 或者输入-1 退出: 97
输入成绩, 或者输入-1 退出: 88
输入成绩, 或者输入-1 退出: 72
输入成绩, 或者输入-1 退出: -1

```

```

全部 3 个成绩的总和是 257
班级平均成绩是 85.67

```

图 3.2 使用哨兵值控制的循环计算班级平均成绩

由于程序源代码中的所有浮点字面值（如 7.33 和 0.0975）默认均以 double 类型表示，因此大多数程序员选择使用 double 类型。这些值被称为**浮点字面值**（floating-point literal）。

①Java 浮点类型的具体取值范围请参阅《Java 语言规范》的 4.2 节“Primitive Types and Values”。②

对比哨兵值控制的循环与计数器控制的循环的程序逻辑

第 12 行将 gradeCounter 初始化为 0，因为尚未输入任何成绩。记住，我们使用哨兵控制的循环来输入成绩，因此仅当用户输入有效成绩时，才会递增变量 gradeCounter。第 33 行声明 double 变量 average，目的是将班级平均成绩存储为浮点数。

将此程序中哨兵控制的循环与图 3.1 计数器控制的循环进行对比：

- 在计数器控制的循环中，已经提前锁死了用户输入的成绩数（10 个）；
- 在哨兵值控制的循环中，图 3.2 的第 16 行~第 17 行会在进入 while 循环前提示用户输入第一个值。该值决定了控制流是否进入 while 循环体：若条件为 false，说明用户第一次便输入了哨兵值（未输入任何成绩），因此 while 循环体不会执行；若条件为 true，循环体将 grade 累加到 total 上并递增 gradeCounter，然后通过第 25 行~

① 译注：字面值（literal）是直接代码中输入的值，包括数字和字符串值。也称为直接量、字面量或文字常量。

② *The Java Language Specification*, Java SE 24 Edition, “4.2. Primitive Types and Values”,

<https://docs.oracle.com/javase/specs/jls/se24/html/jls-4.html#jls-4.2>。

第 26 行提示用户输入下一个成绩并读取。此时程序控制到达循环右大括号（第 27 行），因此会重新测试 `while` 循环的条件（第 20 行），根据最新输入的成绩决定是否再次执行循环体。

注意，下一个成绩总是在程序测试 `while` 语句的条件之前从用户处输入。这使程序能在处理成绩（即把它累加到 `total` 上）之前判断刚才输入的值是否为哨兵值。若 `grade` 包含哨兵值，那么循环终止且不会将 `-1` 累加到 `total` 上。在哨兵值控制的循环中，应显示消息来提醒用户哨兵值的存在。

循环终止后，程序将执行第 31 行~第 42 行的 `if...else` 语句。条件（第 31 行）用于判断是否输入过任何成绩：若未输入，就在 `else` 子句中显示“没有输入任何成绩”。`if...else` 执行完毕后，整个程序终止。

3.7.2 基元类型之间的显式和隐式转换

如果至少输入了一个成绩，第 33 行将计算这些成绩的平均值。之前在讲解图 3.1 的程序时，我们提到整数除法会截断小数部分，产生一个整数结果。因此，即使变量 `average` 被声明为 `double` 类型，但是如果像下面这样计算平均值：

```
double average = total / gradeCounter;
```

那么由于 `total` 和 `gradeCounter` 都是整数，程序还是会执行整数除法，导致商的小数部分在赋给 `average` 之前就已经丢失了。

大多数平均值都是浮点数。若想使用整数值来执行浮点计算，必须临时性地将这些值视为浮点数。这可以通过**一元强制类型转换操作符**（unary cast operator）^①来实现。图 3.2 的第 33 行使用了 `(double)` 转型操作符，它为其右侧的操作数 `total` 创建一个临时的浮点数副本。这称为**显式转换**或**转型**（type casting）。请注意，`total` 变量本身存储的仍然是整数。

经过转换，现在的计算就变成了一个浮点数（`total` 的临时 `double` 副本）除以一个整数 `gradeCounter`。在包含多种类型的算术表达式中，Java 会通过一个称为**提升**（promotion）或**隐式转换**（implicit conversion）的过程来处理操作数。具体来说，当一个 `int` 和一个 `double` 值进行计算时，`int` 值的一个副本会被提升为 `double` 类型。因此，`gradeCounter` 的值也被提升为 `double` 类型，然后程序执行浮点除法，并将最终结果赋给 `average`。本章稍后会讨论所有基元类型，并在 5.11 节更深入地探讨类型提升规则。

3.7.3 浮点数的格式化

第 36 行~第 37 行显示了变量 `gradeCounter` 和 `total` 的值，随后第 38 行显示了 `double` 变

^① 译注：以后将“强制类型转换”简称为“转型”。

量 `average` 的值。这里，`printf` 方法使用 **%f 格式说明符** 来格式化浮点数。默认情况下，`%f` 会以小数点后 6 位的精度来显示 `double` 类型的值。不过，我们可以在 `%` 和 `f` 之间指定一个自定义的精度。在示例运行中，三个成绩的平均值为 `85.666666...`，一个无限循环小数。为了控制输出格式，我们使用了格式说明符 `%.2f`。其中的 `.2` 就代表我们自定义的精度，它会将数值四舍五入到百分位，也就是保留两位小数。因此，最终的输出结果为 `85.67`。

关于浮点数精度的说明

浮点数的表示并不总是百分百精确，但它们在许多应用中都非常有用。例如，当我们说“正常”体温是 `37°C` 时，实际上一般不需要极高的精度。温度计显示 `37°C` 时，其物理上的精确值可能是 `36.9999473210643...`。对于大多数涉及体温的应用来说，简单地称这个数为 `37` 就足够了。

浮点数常常因除法运算而产生，就像本例中的班级平均成绩计算一样。在常规算术中，`10` 除以 `3` 的结果是 `3.3333333...`，其中数字 `3` 会无限重复。计算机为此类数值分配的存储空间是固定的，因此它存储的浮点值必然是一个近似值。

由于浮点数天生就不精确，因此我们通常优先使用 `double` 类型而非 `float`，因为 `double` 变量能更精确地表示浮点数。所以，本书主要使用 `double`。如果想当然地认为浮点数是完全精确的，就可能导致计算错误。在某些情况下，`float` 和 `double` 的精度甚至都无法满足需求。对于需要精确浮点数计算的场景（如货币计算），Java 提供了 `BigDecimal` 类（位于 `java.math` 包中）。我们将在 4.11 节的“对象自然”案例学习中深入探讨它。

生成式 AI 练习

要求 AI 编写一个 Java 程序，该程序应读取用户输入的任意数量的整数摄氏温度，然后计算并显示这些温度的浮点数平均值。请告诉 AI 应假定用户输入的温度范围在 `-135~100` 摄氏度之间，并使用哨兵值 `999` 作为终止输入的标志。

将 AI 生成的代码与以下程序进行比较。如果 AI 使用了我们尚未介绍过的技术，请尝试调整你的提示，给出更具体的细节，引导它仅使用自己已掌握的技术来生成代码。例如，明确告诉它使用 `while` 循环，并在循环之前输入第一个温度值，在循环体的末尾输入后续的温度值。

```
1 // 3.7 节, 生成式 AI 练习: AverageTemperature.java
2 // 计算几个摄氏温度的平均值
3 import java.util.Scanner;
4
5 public class AverageTemperature {
6     public static void main(String[] args) {
7         Scanner input = new Scanner(System.in);
8     }
```

```

9      int total = 0;
10     int count = 0;
11
12     System.out.print("输入一个摄氏温度, 或者输入 999 退出: ");
13     int temperature = input.nextInt();
14
15     while (temperature != 999) {
16         if (temperature >= -135 && temperature <= 100) {
17             total += temperature;
18             count++;
19         } else {
20             System.out.println("温度超出范围 (-135~100 摄氏度), 请重新输入。");
21         }
22         System.out.print("输入一个摄氏温度, 或者输入 999 退出: ");
23         temperature = input.nextInt();
24     }
25
26     if (count > 0) {
27         double average = (double) total / count;
28         System.out.printf("平均温度是: %.2f\n", average);
29     } else {
30         System.out.println("没有输入任何温度");
31     }
32 }
33 }

```

```

输入一个摄氏温度, 或者输入 999 退出: 12
输入一个摄氏温度, 或者输入 999 退出: -22
输入一个摄氏温度, 或者输入 999 退出: 17
输入一个摄氏温度, 或者输入 999 退出: 999
平均温度是: 2.33

```

3.8 嵌套控制语句

如前所述, 控制语句可以按顺序堆叠(连接)在一起。在这个案例学习中, 我们将讨论控制语句的另一种结构化连接方式: 一个控制语句**嵌套**在另一个控制语句中。

3.8.1 问题陈述

考虑以下问题陈述:

大学开设了一门课程, 为学生参加该州的房地产经纪人执照考试做准备。去年, 完成该课程的学生中有 10 人参加了考试。学校想知道学生在考试中的表现如何。你需要写一个程序来汇总结果。已经拿到了这 10 名学生的名单。如果学生通过了考试, 名字旁边会写 1; 如果没有通过考试, 则会写 2。

程序应该对考试的结果进行分析，具体如下。

1. 输入每个考试结果（即 1 或 2）。每次程序要求输入下一个考试结果时，就在屏幕上显示“输入结果(1 = 通过, 2 = 未通过)”；
2. 统计每种考试结果的数量；
3. 显示考试结果的摘要，列出通过和未通过考试的学生人数；
4. 如果超过 8 名学生通过了考试，就输出消息：“给老师发奖金!”。

分析问题陈述

仔细分析问题陈述，我们得出以下几点结论：

- 该程序必须处理 10 名学生的考试结果（通过或未通过）。由于考试结果数量是已知的，因此可以使用计数器控制的循环；
- 每个考试结果只能是 1 或 2。当程序读取每个结果时，它必须判断该结果是 1 还是 2。在我们的算法中，首先会检查是否为 1；如果结果不为 1，则我们默认它是 2；
- 程序使用两个计数器来跟踪考试结果，一个用于统计通过考试的学生人数，另一个用于统计未通过考试的学生人数；
- 在程序处理完所有结果后，它必须判断是否有超过 8 名学生通过了考试。

3.8.2 实现程序

图 3.3 通过一个计数器控制的循环实现了该程序，并展示了两次示例执行过程。其中，第 11 行~第 13 行及第 19 行声明了处理考试结果所需的变量。建议在声明局部变量时就将其初始化，这是一种良好的编程习惯，有助于避免因尝试使用未初始化变量而导致的编译错误。需要强调的是，虽然 Java 并不强制要求在声明变量时就进行初始化，但它严格规定：任何局部变量在表达式中使用之前都必须被赋值，否则将导致编译错误。

```
1 // 图 3.3: Analysis.java
2 // 使用嵌套控制语句分析考试结果
3 import java.util.Scanner; // class uses class Scanner
4
5 public class Analysis {
6     public static void main(String[] args) {
7         // 创建一个 Scanner 对象从用户获取输入
8         Scanner input = new Scanner(System.in);
9
10        // 声明的同时初始化变量
11        int passes = 0;
12        int failures = 0;
13        int studentCounter = 1;
14
15        // 使用计数器控制的循环处理 10 名学生的考试结果
16        while (studentCounter <= 10) {
```

```

17         // 提示用户输入值
18         System.out.print("输入结果(1 = 通过, 2 = 未通过): ");
19         int result = input.nextInt();
20
21         // if...else 嵌套在 while 中
22         if (result == 1) {
23             passes = passes + 1;
24         }
25         else {
26             failures = failures + 1;
27         }
28
29         // 递增 studentCounter, 使循环最终能终止
30         studentCounter = studentCounter + 1;
31     }
32
33     // 终止阶段; 准备并显示结果
34     System.out.printf("通过数: %d\n 未通过数: %d\n", passes, failures);
35
36     // 判断是否超过 8 名学生通过考试
37     if (passes > 8) {
38         System.out.println("给老师发奖金!");
39     }
40 }
41 }

```

```

输入结果(1 = 通过, 2 = 未通过): 1
输入结果(1 = 通过, 2 = 未通过): 2
输入结果(1 = 通过, 2 = 未通过): 1
通过数: 9
未通过数: 1
给老师发奖金!

```

```

输入结果(1 = 通过, 2 = 未通过): 1
输入结果(1 = 通过, 2 = 未通过): 2
输入结果(1 = 通过, 2 = 未通过): 1
输入结果(1 = 通过, 2 = 未通过): 2
输入结果(1 = 通过, 2 = 未通过): 1
输入结果(1 = 通过, 2 = 未通过): 2
输入结果(1 = 通过, 2 = 未通过): 2
输入结果(1 = 通过, 2 = 未通过): 1
输入结果(1 = 通过, 2 = 未通过): 1
输入结果(1 = 通过, 2 = 未通过): 1

```

通过数: 6 未通过数: 4

图 3.3 使用嵌套控制语句分析考试结果

`while` 语句（第 16 行~第 31 行）总共循环 10 次。在每次循环迭代中，程序都会输入并处理一份考试成绩。`while` 循环内部嵌套了一个 `if...else` 语句（第 22 行~第 27 行），用于处理每次输入的成绩：如果输入值为 1，则将 `passes` 变量加 1；否则，程序假定输入值为 2，并将 `failures` 变量加 1。第 30 行将循环计数器 `studentCounter` 加 1，随后程序返回第 16 行，再次对循环条件进行判断。当输入 10 个成绩后，循环终止。第 34 行会显示通过和未通过考试的总人数。最后，第 37 行~第 39 行的 `if` 语句会判断通过考试的学生是否超过 8 名，如果条件为 `true`，则输出消息：“给老师发奖金!”。

图 3.3 展示了该程序的两次示例执行过程。在第一次执行中，第 37 行的条件为 `true`（通过人数超过 8 人），因此程序输出了发放奖金的消息。

生成式 AI 练习

1. 要求 AI 简化以下程序，将其中嵌套的 `if...else` 语句替换为一个使用了三元操作符的条件表达式（参见 3.4.3 节）的单行语句。执行 AI 生成的代码，确保其产生相同的结果。

```
1 // 3.8 节, 生成式 AI 练习: OddOrEven.java
2 // 判断输入的整数是奇数还是偶数
3 import java.util.Scanner;
4
5 public class OddOrEven {
6     public static void main(String[] args) {
7         Scanner input = new Scanner(System.in);
8         int counter = 1;
9
10        while (counter <= 3) {
11            System.out.print("输入一个整数: ");
12            int number = input.nextInt();
13
14            if (number % 2 == 0) {
15                System.out.printf("%d 是偶数\n", number);
16            }
17            else {
18                System.out.printf("%d 是奇数\n", number);
19            }
20
21            counter = counter + 1;
22        }
23    }
```

```

输入一个整数: 2
2 是偶数
输入一个整数: 3
3 是奇数
输入一个整数: 5
5 是奇数

```

3.9 复合赋值操作符

可以使用**复合赋值操作符**（compound assignment operator）来简化赋值表达式。例如，以下语句：

```
c = c + 3; // 使 c 的值递增 3
```

可以使用加法复合赋值操作符+=简写为：

```
c += 3; // 更简洁的写法
```

+=操作符将右侧表达式的值与左侧变量的值相加，结果存回左侧的变量中。因此，赋值表达式 `c += 3` 将 3 加到 `c` 上，结果存回 `c`，从而实现了使 `c` 递增 3 的效果。复合赋值操作符会对右侧表达式的值执行隐式类型转换，以匹配操作符左侧变量的类型。因此，如果左侧变量是 `int` 类型，而右侧表达式的结果是 `double` 类型，那么该浮点值的小数部分会被截断（即不会四舍五入）。^①表 3.1 展示了算术复合赋值操作符和示例表达式，并解释了它们的作用。

表 3.1 算术复合赋值操作符

操作符	示例表达式	解释	赋值结果
<i>假设: int c = 3, d = 5, e = 4, f = 6, g = 12;</i>			
+=	<code>c += 7</code>	<code>c = c + 7</code>	将 10 赋给 c
--	<code>d -= 4</code>	<code>d = d - 4</code>	将 1 赋给 d
*=	<code>e *= 5</code>	<code>e = e * 5</code>	将 20 赋给 e

^① *The Java Language Specification*, Java SE 24 Edition, Section 15.26.2, “Compound Assignment Operators”, <https://docs.oracle.com/javase/specs/jls/se24/html/jls-15.html#jls-15.26.2>。

/=	f /= 3	f = f / 3	将 2 赋给 f
%=	g %= 9	g = g % 9	将 3 赋给 g

生成式 AI 练习

1. 要求 AI 回答在执行了以下代码后，变量 `y` 的值是多少：

```
int y = 5;
y += 3.8;
```

确认 AI 回答 `y` 最终的值是 8。你可能会预期该赋值语句会导致编译错误，但事实并非如此。复合赋值操作符会隐式地将右侧表达式的值转换为与左侧变量相同的类型。在本例中，3.8 是一个 `double` 值，但由于 `y` 是 `int`，因此 3.8 会被截断为整数 3，而不是四舍五入。因此，`y` 最终的值是 $5 + 3 = 8$ 。

3.10 递增和递减操作符

表 3.2 总结了一元递增和递减操作符 `++` 和 `--`，它们分别使数值变量的值加 1 或减 1。和二元操作符不同，根据约定，一元操作符与操作数间不留空格。^①

表 3.2 一元递增和递减操作符

操作符	名称	执行顺序	示例（假设 num = 5）
<code>++num</code>	前缀递增	先加 1，再使用新值	<code>int a = ++num;</code> → a=6, num=6
<code>num++</code>	后缀递增	先使用当前值，再加 1	<code>int a = num++;</code> → a=5, num=6
<code>--num</code>	前缀递减	先减 1，再使用新值	<code>int a = --num;</code> → a=4, num=4
<code>num--</code>	后缀递减	先使用当前值，再减 1	<code>int a = num--;</code> → a=5, num=4

^① 译注：避免在复杂表达式中过度使用前缀/后缀操作符（如 `a = ++i + i++;`），因为它们会导致代码可读性下降。

对比前缀递增和后缀递增

图 3.4 的程序演示了++递增操作符的前缀和后缀版本的区别。递减操作符（--）与之类似。

```
1 // 图 3.4: Increment.java
2 // 前缀递增和后缀递增操作符
3
4 public class Increment {
5     public static void main(String[] args) {
6         // 演示后缀递增操作符
7         int c = 5;
8         System.out.printf("后缀递增之前的 c: %d\n", c); // 打印 5
9         System.out.printf("对 c 进行后缀递增: %d\n", c++); // 打印 5
10        System.out.printf("后缀递增之后的 c: %d\n", c); // 打印 6
11
12        System.out.println(); // 跳过一行
13
14        // 演示前缀递增操作符
15        c = 5;
16        System.out.printf("前缀递增之前的 c: %d\n", c); // 打印 5
17        System.out.printf("对 c 进行前缀递增: %d\n", ++c); // 打印 6
18        System.out.printf("前缀递增之后的 c: %d\n", c); // 打印 6
19    }
20 }
```

后缀递增之前的 c: 5 对 c 进行后缀递增: 5 后缀递增之后的 c: 6 前缀递增之前的 c: 5 对 c 进行前缀递增: 6 前缀递增之后的 c: 6
--

图 3.4 前缀递增和后缀递增操作符

第 7 行将变量 `c` 初始化为 5，并在第 8 行输出这个初始值。第 9 行输出表达式 `c++` 的值。该表达式对变量 `c` 执行的是后缀递增，所以会先输出 `c` 的原始值（5），再递增 `c`（变成 6）。因此，第 9 行输出的是 `c` 的初始值（5）。第 10 行则输出 `c` 的新值（6），以证明该变量的值确实在第 9 行递增了。

第 15 行将 `c` 的值重置为 5，并在第 16 行输出该值。第 17 行输出表达式 `++c` 的值。该表达式对变量 `c` 执行的是前缀递增，所以会先递增它的值，再输出 `c` 的新值（6）。第 18 行再次输出 `c` 的值，证明在执行了第 17 行后，`c` 的值仍然是 6。

简化递增语句

可以使用算术复合赋值操作符以及递增和递减操作符来简化代码。以图 3.3 的三个赋值语句（第 23 行、第 26 行和第 30 行）为例：

```
passes = passes + 1;
failures = failures + 1;
studentCounter = studentCounter + 1;
```

它们可用复合赋值操作符简化为：

```
passes += 1;
failures += 1;
studentCounter += 1;
```

甚至可以用前缀递增操作符进一步简化为：

```
++passes;
++failures;
++studentCounter;
```

或者使用后缀版本：

```
passes++;
failures++;
studentCounter++;
```

如果变量递增或递减自成一个语句，那么前缀和后缀版本的效果一样（无论递增还是递减）。只有当变量用在一个更大的表达式中时，前缀版本和后缀版本才有不同的效果。

生成式 AI 练习

1. 要求 AI 解释以下代码的作用：

```
1 // 3.10 节, 生成式 AI 练习: Mystery.java
2 public class Mystery {
3     public static void main(String[] args) {
4         int x = 1;
5         int total = 0;
6
7         while (x <= 10) {
8             int y = x * x;
9             System.out.println(y);
10            total += y;
11            ++x;
12        }
13
14        System.out.printf("总和为%d\n", total);
15    }
16 }
```

将它的回答与以下结果进行比较（输出 1~10 的平方，最后输出这些平方值之和）

```
1
4
9
16
25
36
49
64
81
100
总和为 385
```

3.11 基元类型

Java 严格要求必须为所有变量都声明一个类型。^①这一点与 C 和 C++ 不同，后两者中的**基元类型**（primitive type）在不同计算机平台上的大小可能不一致，程序员因此常常需要为不同平台编写不同版本的程序。例如，在 C 或 C++ 中，一个 `int` 类型在一种系统上可能占用 16 位（2 字节），在另一种系统上可能是 32 位（4 字节），甚至在其他系统上是 64 位（8 字节）。相比之下，Java 通过规定所有基元类型的大小在任何平台上都保持不变，解决了这个问题。例如，Java 的 `int` 类型在所有环境下始终是 32 位（4 字节）。这种跨平台的一致性 is Java 的重要特性之一。

Java 的 8 种基元类型（`boolean`、`byte`、`char`、`double`、`float`、`int`、`long` 和 `short`）都具有跨平台一致性，在支持 Java 的所有计算机上，其大小和行为完全相同。要具体了解这些类型的取值范围，请参阅《Java 语言规范》的 4.2 节：“Primitive Types and Values”。^②

为了从根本上保证程序的**可移植性**（portability），Java 的设计者在两个关键领域采用了国际公认的标准：

- **字符格式**：遵循 Unicode 标准，确保文本在不同系统上都能被一致地处理（<https://unicode.org>）
- **浮点数表示**：遵循 IEEE 754 标准，确保浮点数运算在任何硬件上都得到相同的结果

^① 第 14 章在讲解 Lambda 表达式的时候，会看到一个例外。（译注：Lambda 表达式严格来说并不是变量类型声明的例外，但它确实引入了更灵活的类型推断机制，使得在某些情况下可以隐式声明类型。不过，这并不意味着 Lambda 逃避了 Java 的类型系统，编译器在背后仍然严格维护着类型安全。）

^② *The Java Language Specification*, Java SE 24 Edition, Section 4.2, “Primitive Types and Values”, <https://docs.oracle.com/javase/specs/jls/se24/html/jls-4.html#jls-4.2>。

(<https://standards.ieee.org/ieee/754/6210>)

这种对标准化设计的坚持，是 Java 实现“一次编写，到处运行”（Write Once, Run Anywhere, WORA）承诺的核心基石之一。

生成式 AI 练习

1. 要求 AI 编写一个 Java 程序，显示 Java 中所有基元整型^①的最小值和最大值，然后运行生成的程序。^②

3.12 “对象自然” 案例学习：超大整数

随着互联网通信和联网设备的爆炸式增长，数据隐私与安全变得前所未有的重要。几千年来，**密码学**（cryptography）一直被用于对数据进行**编码**（encoding），使未经授权的用户难以（理想情况下完全无法）读取，这在当今万物互联的世界中至关重要。我们访问的大多数网站都使用 HTTPS（HyperText Transfer Protocol Secure，安全超文本传输协议）来对网络交互进行**加密**（encrypt）和**解密**（decrypt）。

有些应用需要超出 long 范围的整数

Java 的基元类型 long 是一种 64 位有符号整数，可存储从 -9 223 372 036 854 775 808 到 9 223 372 036 854 775 807 的值，这是一个高达 19 位的十进制数。然而，许多现代密码学算法所执行的计算，涉及的整数远远超出了 long 的存储极限。

当前最流行的加密/解密方案之一是 **RSA 公钥密码学**（RSA Public-Key Cryptography），它被广泛用于保护互联网传输的数据。^③工业级的 RSA 实现会使用由数百位数字构成的巨大质数，其大小远非 long 类型所能容纳。RSA 之所以安全，一个关键原因在于：即便对于当今最强大的超级计算机，想要分解这些巨大质数的乘积也需要极其漫长的时间。^④此外，公钥密码学也被用于保护比特币等加密货币背后的区块链技术。^⑤我们将在本书后续章节

^① 译注：包括 byte、short、int、long 和 char。

^② 译注：本书中文版配套资源提供了示例程序 IntegerLimits.java，详情请访问 <https://bookzhou.com>。

^③ 维基百科，<https://zh.wikipedia.org/wiki/RSA> 加密算法。

^④ 译注：除非真正的量子计算机问世。

^⑤ Sarah Rothrie, “How Blockchain Cryptography Is Fighting the Rise of Quantum Machines,” December 28, 2018, <https://coincentral.com/blockchain-cryptography-quantum-machines/>。

深入探讨公钥密码学。

超出 long 范围的整数需要自定义编程

那么，我们该如何处理具有任意位数的大整数呢？当需要处理超出 long 范围的整数时，就需要进行自定义编程。我们可以利用 Java 的类机制（1.4 节提及，第 8 章详述）来开发一个能处理大整数运算的自定义类——但这可能需要数月的工作量。

而“对象自然”的编程思想鼓励我们，首先去检查 Java API 或其他免费的开源库中，是否已存在类似的“大整数”类。

使用 Java API 中的 java.math.BigInteger 类

幸好，Java 已经有现成的类可用于创建和操作大整数。在本例中，我们将演示来自 Java API java.math 包的 BigInteger 类，它的 API 文档网址是：

<https://docs.oracle.com/en/java/javase/24/docs/api/java.base/java/math/BigInteger.html>

在图 3.5 的“对象自然”示例程序中，我们将直接使用 BigInteger 类的对象，而无需了解其内部实现细节。^{①②}我们只需导入该类（第 4 行），用它创建 BigInteger 对象（第 12 行～第 15 行），然后进行计算并显示结果（第 23 行～第 30 行）。

```
1 // 图 3.5: UsingBigInteger.java
2 // 使用 Java API BigInteger 类的对象来
3 // 创建和操作超大整数
4 import java.math.BigInteger;
5
```

^① 这就像驾驶汽车，无需精通发动机、转向、刹车及传动系统的内部工作原理，我们就能正常地把车开走。同样，可以先高效地使用 BigInteger 类。在对 Java 的理解加深后，或许会产生兴趣去研究它的源代码（大约 5000 行，包含大量 Javadoc 注释），从而学习其实现细节。在本书后续的面向对象编程章节中，你将学到足以创建自己的大整数类的技术。

^② 你可能会反驳：“不行，我坚持要了解我使用的每个 Java 特性是如何实现的。”那么，请稍作思考：你正在使用 Java 编译器将代码翻译成字节码，然后由 Java 虚拟机（JVM）将其转换为你计算机使用的机器语言。在此过程中，你了解编译器的具体实现吗？你确定它能将你的每个程序都准确无误地翻译，从而保证运行正确、结果无误吗？事实是，我们每天都在信任并使用着我们并未完全理解其内部细节的复杂系统。这正是现代软件工程中“抽象”的核心思想——它让我们能站在巨人的肩膀上，高效地构建更强大的应用。

223 372 036 854 775 807)。你会看到，本例中 `BigInteger` 的每一项计算结果都远超此值。Java 的每种基元类型（如 `long`）在 `java.lang` 包中都有一个对应的包装类（如 `Long`），这些类定义了 `MIN_VALUE` 和 `MAX_VALUE` 常量，分别表示该基元类型的最小值和最大值。

初始化 `BigInteger` 对象

可以使用以下方式初始化 `BigInteger` 对象：

1. **使用字符串初始化**：如果一个 `BigInteger` 的初始值超出了 `long` 的范围，那么可以用一个字符串来表示这个巨大的值，并用它初始化 `BigInteger` 对象。第 12 行～第 13 行处理一个 30 位整数时采用了这种方式。
2. **使用整数值初始化（静态方法 `valueOf`）**：第 14 行～第 15 行演示了如何使用基元类型的整数值来初始化 `BigInteger` 对象。类通过方法来完成其任务。在此之前，我们都是类的某个对象上调用方法（例如 `System.out` 对象的 `printf` 方法）。但是，有些方法直接在类本身上调用，而不是在类的实例（对象）上调用，这类方法称为**静态方法**（`static method`）。调用静态方法的语法是：*类名.方法名(参数值)*。

`BigInteger` 类的静态方法 `valueOf` 接收一个基元类型的整数值作为实参，并返回一个用该值初始化的新 `BigInteger` 对象。第 14 行用 `long` 类型的最大值初始化了 `value2`。注意数字末尾的 `L` 后缀，它告诉编译器这是一个 `long` 类型的值。如果没有 `L`，编译器会默认将其视为 `int`，而该值远大于 `int` 的最大值（2 147 483 647），会导致编译错误。^①第 15 行用 `int` 值 17 初始化了 `value3`。第 18 行～第 20 行显示了这三个 `BigInteger` 对象的初始值。

`printf` 的 `%s` 格式说明符如何识别 `BigInteger`？

在 Java 中，每个基元类型的值和对象都有其字符串表示形式。使用 `printf` 和 `%s` 格式说明符时，Java 会自动将目标值或对象转换为字符串。对于对象，Java 是通过调用该对象的 `toString` 方法来实现这一点的。同样，使用 `System.out.print` 或 `println` 显示对象时，`toString` 方法也会被自动调用。

执行 `BigInteger` 算术运算

`BigInteger` 类提供了 `add`（加）、`subtract`（减）、`multiply`（乘）和 `divide`（除）等常见的算术方法。如第 23 行～第 30 行所示，`BigInteger` 可以正确处理任意位数的整数运算。每个计算结果的位数都超出了 Java 基元整型的表示能力，例如，第 28 行的乘法运算生成了一个 48 位的整数。

^① 第 14 行可以直接使用 `Long.MAX_VALUE`，而不必写字面值 `9_223_372_036_854_775_807L`。

可靠的内容。

3.1（控制语句）要求 AI 出一些入门级的 Java `if` 和 `while` 控制语句练习题，做这些题并实际运行程序。要求每道题的解决方案不超过 50 行代码。

3.2（补全缺失代码）从一个程序中删除部分代码，然后告诉 AI 这个程序应该实现什么功能，并要求它补全缺失的代码。

3.3（超大整数）要求 AI 撰写一篇关于 `BigInteger` 的实际应用的教程。

附录 A JShell 交互式 Java 入门

学习目标：

- 使用 JShell 快速探索、发现和试验 Java 语言及 API 功能，改善自己的学习和软件开发过程；
- 启动 JShell 会话；
- 执行代码片段；
- 声明变量；
- 表达式求值；
- 编辑现有的代码片段；
- 声明并使用类；
- 将代码片段保存到文件；
- 打开一个 JShell 代码片段文件并执行其中的代码；
- 自动补全代码和 JShell 命令；
- 显示方法的参数和重载；
- 在 JShell 中使用 Java API 文档；
- 声明并使用方法；
- 前向引用尚未声明的方法；
- 了解 JShell 如何包装异常；
- 导入自定义包以便在 JShell 会话中使用；
- 控制 JShell 的反馈级别。

A.1 导论

我们接触过的许多老师都表示，在入门编程课程中，他们更喜欢使用支持 REPL（读取-求值-输出循环）的语言。Java 的 JShell 提供了丰富的 REPL 体验。许多第三方已将 JShell 和相关的交互式开发工具构建到了主流 IDE 中，如 IntelliJ、Eclipse、NetBeans 等。^①

什么是 JShell？

JShell 的神奇之处是什么？其实很简单。它提供了一个快速且友好的环境，使你可以迅速地探索、发现和试验 Java 语言的功能及其包罗万象的库。像 JShell 中的 REPL 这样的工具已经存在了几十年。在 20 世纪 60 年代，最早的 REPL 之一使得 LISP 编程语言的便捷交互式开发成为可能。那个时代的学习者，比如本书的作者之一哈维·戴特尔（Harvey Deitel），

^① 译注：.NET Core 现在也内置了 C# REPL 支持，同样允许交互式执行 C# 代码，简化开发和调试流程。

发现它使用起来既快速又有趣。

JShell 用其读取-求值-输出循环取代了编辑-编译-执行的周期。无需编写完整的程序，只需编写 JShell 命令和 Java 代码片段。输入一个代码片段时，JShell 会立即读取它、求值并打印结果，以帮助你观察代码的效果。然后，它会循环此过程，为下一个代码片段再次执行。若需完成大量示例和练习，就会体会到 JShell 及其即时反馈如何帮助自己保持专注、提高效率并加快学习和软件开发过程。

让代码不再“冷冰冰”

如你所知，我们在本书强调“实时代码”^①教学方法的^①价值。其重点在于，你接触到的都是完整的、可实际运行的程序。JShell 将这种方法带到了单个代码片段的级别。每输入一行代码，代码就真正地“活”了起来。当然，输入代码片段时，仍然会偶尔出错。JShell 会逐个代码片段地向你报告编译错误。例如，可以利用这个功能来测试常见的编程错误，并观察错误发生时的状态。

代码片段的种类

代码片段 (snippet) 可以是表达式、单个语句、多行语句以及更大的实体，如方法和类。JShell 支持除少数特性外的所有 Java 功能，但也有一些独特地方，旨在促进 JShell 的探索-发现-试验能力。在 JShell 中，方法不必在类中，表达式和语句不必在方法中，并且不需要 main 方法（其他差异见 A.13 节）。与完整程序漫长而重复的编辑、编译和执行周期相比，省去这些基础结构可以为你节省大量时间。而且，由于 JShell 会自动显示表达式和语句的求值结果，因此不需要像本书的传统 Java 代码示例中那样使用那么多的打印语句。

通过自动补全进行探索

我们详细介绍了**自动补全** (auto-completion) 功能，这是一个能加快编码过程的关键探索功能。输入名称的一部分（类、方法、变量等）并按下 Tab 键后，JShell 会自动补全该名称，或提供一个以已输入内容开头的所有可能名称的列表。然后，就可以轻松地显示方法参数和描述这些方法的文档。

快速原型设计

专业开发人员通常使用 JShell 进行实验和快速原型设计，而不是一开始就进行完整的软件

^① 译注：即“live-code”，也可以理解为“活代码”。

开发。一旦创建并测试了一个小的代码块，就可以将其粘贴到更大的项目中。

本附录的组织方式

本附录是选读内容。对于那些想使用 JShell 的人，本附录被设计为一系列与前面特定章节同步的单元。每个单元都以类似于“本节可以在学习完第 2 章后阅读”的语句开头。因此，应该首先通读第 2 章，然后阅读本附录的相应小节。

针对第 2 章的 JShell 自测题与答案

在学习本附录的过程中，请在 JShell 中执行每个代码片段和命令，以确认各项功能是否如文中所述。A.2 节~A.3 节设计为在学习第 2 章后阅读。读完这些小节后，建议立即完成本附录末尾的数十个自测题。JShell 鼓励你“边做边学”，因此这些练习将指导你编写并测试代码片段，以实践第 2 章讲述的许多 Java 功能。这些自测题小而精，每个练习后面都紧跟着答案，以帮助你快速熟悉 JShell 的各种功能。

A.2 JShell 简介

[本节可以在学习完第 2 章后阅读。]

如第 2 章所述，要创建一个可以运行的 Java 应用程序，我们需要：

1. 创建一个包含 main 方法的类；
2. 在 main 方法中声明程序运行时要执行的语句；
3. 编译程序并修复出现的任何编译错误——这个步骤必须重复进行，直到程序编译无误；
4. 运行程序以观察结果。

通过在完成每个表达式或语句时就自动编译和执行代码，JShell 消除了以下开销：

- 创建一个包含你希望测试的代码的类；
- 编译该类；
- 执行该类。

相反，现在可以专注于交互式地发现和试验 Java 的语言和 API 特性。如果输入了无法编译的代码，JShell 会立即报错。然后，可以使用 JShell 的编辑功能快速修复并重新执行代码。

A.2.1 启动 JShell 会话

要启动 JShell 会话，请在系统的命令行中执行 `jshell` 命令：

- 在 Microsoft Windows 上，打开命令提示符或 PowerShell 窗口。

- 在 macOS 上，打开终端窗口。

这将启动一个新的 JShell 会话，显示以下消息和 `jshell>` 提示符：

```
| 欢迎使用 JShell -- 版本 24.0.1
| 要大致了解该版本，请键入：/help intro

jshell>
```

上面的第一行表明当前正在使用 JDK 的 24.0.1 版本。^①JShell 会在提示消息前加上竖线 (|)。现在就可以开始输入 Java 代码或 JShell 命令了。

A.2.2 执行语句

[学习本附录时，请在 `jshell>` 提示符中输入和本书完全一致的代码和 JShell 命令，以确保你的输出与我们的匹配。]

JShell 有两种输入类型：

- Java 代码，JShell 文档将其称为代码片段 (snippet)；
- JShell 命令。

在本节和 A.2.3 节中，我们从 Java 代码片段开始。后续小节将介绍 JShell 命令。

可以在 `jshell>` 提示符下输入任何表达式或语句，然后按 Enter 或 Return 键来执行代码并立即观察其结果。我们在图 A.1 中重新列出了图 2.1 的程序。为了演示 `System.out.println` 的工作方式，该程序需要许多行代码和注释，你需要编写、编译和执行这些代码。即使没有注释，也仍然需要 5 行代码（第 4 行和第 6 行～第 9 行）。

```
1 // 图 A.1: Welcome1.java
2 // 文本输出程序
3
4 public class Welcome1 {
5     // main 方法开始程序执行
6     public static void main(String[] args) {
7         System.out.println("欢迎学习 Java 编程!");
8     } // 结束 main 方法
9 } // 结束 Welcome1 类
```

```
欢迎学习 Java 编程!
```

图 A.1 文本打印程序

^① 译注：具体版本号取决于你安装的 Java。注意，Java 25 将在 2025 年 9 月发布。

在 JShell 中，可以直接执行第 7 行的语句，而无需完整地创建 `Welcome1` 类及其 `main` 方法，如下所示：

```
jshell> System.out.println("欢迎学习 Java 编程!")
欢迎学习 Java 编程!

jshell>
```

在本例中，JShell 在初始的 `jshell>` 提示符和你输入的语句下面显示该代码片段的命令行输出结果。按照惯例，我们用粗体表示用户输入。

注意，这里没有语句的终止分号（`;`）。注意，JShell 仅为独立语句^①自动添加终止分号。^②例如，如果语句位于大括号（`{}`和`}`）内，那么我们必须手动添加分号。

第二个 `jshell>` 提示符之前之所以出现空行，是连续两个换行符的结果。`println` 方法会生成一个，JShell 在每个 `jshell>` 提示符之前又始终会添加一个。若使用 `print` 而不是 `println` 命令，就可以消除这个空行，如下所示：

```
jshell> System.out.print("欢迎学习 Java 编程!")
欢迎学习 Java 编程!
jshell>
```

JShell 会跟踪你输入的所有内容，这对于重新执行先前的语句和修改语句以更新其任务非常有用。稍后就会讲到，按下箭头键即可重复之前输入的内容。

A.2.3 显式声明变量

几乎所有可以在典型的 Java 源代码文件中声明的内容，也可以在 JShell 中声明（A.13 节讨论了无法使用的一些特性）。例如，可以按如下方式显式声明一个变量：

```
jshell> int number1
number1 ==> 0

jshell>
```

输入变量声明时，JShell 会显示变量的名称（本例为 `number1`），后跟 `==>`（表示“具有值”）和变量的初始值（`0`）。如果没有显式指定初始值，那么变量将初始化为其类型的默认值。在本例中，`int` 变量的默认值为 `0`。

^① 译注：一行完整的、可独立运行的语句。

^② 不要求分号是 JShell 为了方便交互而重新解释标准 Java 的一个例子。本附录会演示其中的一些例子，并在 A.13 节中进行总结。

变量可以在声明的同时初始化，下面让我们重新声明 `number1`：

```
jshell> int number1 = 30
number1 ==> 30

jshell>
```

JShell 显示 `number1 ==> 30`，表明 `number1` 现在的值为 `30`。在当前的 JShell 会话中，若声明的新变量与之前声明过的变量同名，那么 JShell 会将第一个声明替换为新的声明。^①由于 `number1` 之前已声明，因此可以简单地地为 `number1` 重新赋值，如下所示：

```
jshell> number1 = 45
number1 ==> 45

jshell>
```

JShell 中的编译错误

变量必须先声明然后才能在 JShell 中使用。以下 `int` 变量 `sum` 的声明尝试使用一个名为 `number2` 的变量，但后者尚未声明，因此 JShell 报告一个编译错误，指示编译器无法找到名为 `number2` 的变量。

```
jshell> int sum = number1 + number2
| 错误：
| 找不到符号
|   符号： 变量 number2
|   位置：类
| int sum = number1 + number2;
|                               ^-----^

jshell>
```

错误消息使用符号 `^-----^` 来突出显示语句中的错误。之前声明的变量 `number1` 没有报错。但上述代码片段存在一个编译错误，因此它是无效的。不过，JShell 仍然将该代码片段作为 JShell 会话历史记录的一部分进行维护，其中包括有效的代码片段、无效的代码片段和你输入的命令。很快就会了解到，可以轻松地召回这个无效的代码片段并再次执行它。JShell 的 `/history` 命令显示当前会话的历史记录——即你输入的所有内容：

```
jshell> /history

System.out.println("欢迎学习 Java 编程!")
```

^① 允许重新声明现有变量，这是 JShell 为了方便交互而重新解释标准 Java 的另一个例子。此行为与 Java 编译器处理现有变量的新声明的方式不同，后者属于“重复声明”，会造成编译错误。

```
System.out.print("欢迎学习 Java 编程!")
int number1
int number1 = 30
number1 = 45
int sum = number1 + number2
/history
```

修复错误

JShell 可以轻松修复之前的错误并重新执行代码片段。下面通过首先声明值为 72 的 `number2` 来修复前面的错误：

```
jshell> int number2 = 72
number2 ==> 72

jshell>
```

后续代码片段就能正常使用 `number2` 了。稍后，我们将重新执行“声明并将 `sum` 初始化为 `number1 + number2`”的代码片段。

召回并重新执行之前的代码片段

现在，`number1` 和 `number2` 已声明，我们可以正确声明 `int` 变量 `sum` 了。按上箭头键和下箭头键，即可切换之前输入的代码片段和 JShell 命令。在本例中，可以按三次上箭头键来召回之前失败的声明，而不必重新输入。JShell 以相反的顺序召回之前的输入——输入的最后一个代码片段或命令首先被召回。因此，第一次按上箭头键时，`jshell>` 提示符下会出现以下内容：

```
jshell> int number2 = 72
```

第二次按向上箭头键时，会出现 `/history` 命令：

```
jshell> /history
```

第三次按向上箭头键时，会出现 `sum` 先前的声明：

```
jshell> int sum = number1 + number2
```

现在就可以按 `Enter` 键来重新执行声明和初始化 `sum` 的代码片段：

```
jshell> int sum = number1 + number2
sum ==> 117

jshell>
```

JShell 将 `number1` (45) 和 `number2` (72) 的值相加，将结果存储到新的 `sum` 变量中，然后

输出 `sum` 的值（117）。

A.2.4 列出并执行之前的代码片段

可以使用 JShell 的 `/list` 命令查看所有之前有效的 Java 代码片段的列表——JShell 按照它们的输入顺序来显示，如下所示：

```
jshell> /list

 1 : System.out.println("欢迎学习 Java 编程!")
 2 : System.out.print("欢迎学习 Java 编程!")
 4 : int number1 = 30;
 5 : number1 = 45
 6 : int number2 = 72;
 7 : int sum = number1 + number2;

jshell>
```

每个有效的代码片段都由一个连续的代码片段 ID 标识。在本例中，ID 为 3 的代码片段缺失，因为我们用 ID 为 4 的代码片段替换了如下所示的原始变量声明：

```
int number1
```

`/list` 命令没有列出 `/history` 所显示的全部内容。如前所述，如果省略了语句的终止分号，JShell 会在后台自动插入该分号。执行 `/list` 时，只有声明语句（代码片段 4、6 和 7）才自动插入分号。代码片段 1 只是一个表达式。如果用终止分号输入它，它就变成了一个**表达式语句**。

按 ID 执行代码片段

可以通过输入 `/id` 来执行任何之前的代码片段，其中 `id` 是代码片段的 ID。例如，下例输入 `/1`：

```
jshell> /1
System.out.println("欢迎学习 Java 编程!")
欢迎学习 Java 编程!

jshell>
```

JShell 首先会显示当前会话中 ID 为 1 的代码片段，然后执行它并显示结果。

重新执行多个单独的代码片段或代码片段范围

也可以一次重新执行多个语句。例如，以下命令将重新执行 ID 从 `start` 到 `end`（包括 `end`）的代码片段范围：

```
/start-end
```

以下命令将重新执行单独的代码片段或代码片段范围：

```
/id1 id2 id3 ...
```

在本例中，每个 ID 既可以是单独的代码片段 ID，也可以是一个 ID 范围。

重新执行最后一个代码片段

要重新执行上一个代码片段，请使用 `!/`，如下所示：

```
jshell> !/  
System.out.println("欢迎学习 Java 编程!")  
欢迎学习 Java 编程!
```

```
jshell>
```

JShell 为我们执行的每个有效代码片段分配一个 ID，因此即使以下语句已作为代码片段 1 存在于此会话中：

```
System.out.println("欢迎学习 Java 编程!")
```

JShell 也会创建一个新的代码片段，其 ID 为序列中的下一个 ID（在本例中，最后两个代码片段的 ID 为 8 和 9）。可以执行 `/list` 命令来确认代码片段 1、8 和 9 是相同的，如下所示：

```
jshell> /list  
  
1 : System.out.println("欢迎学习 Java 编程!")  
2 : System.out.print("欢迎学习 Java 编程!")  
4 : int number1 = 30;  
5 : number1 = 45  
6 : int number2 = 72;  
7 : int sum = number1 + number2;  
8 : System.out.println("欢迎学习 Java 编程!")  
9 : System.out.println("欢迎学习 Java 编程!")
```

```
jshell>
```

A.2.5 表达式求值和变量隐式声明

在 JShell 中输入一个表达式时，它会对其进行求值，隐式地创建一个变量，并将表达式的

求值结果赋给该变量。隐式变量的名称为`$#`，其中`#`是新代码片段的 ID，如下例所示：^①

```
jshell> 11 + 5
$10 ==> 16

jshell>
```

上述命令对表达式 `11 + 5` 进行求值，并将结果值（16）赋给隐式声明的变量`$10`，这是因为之前已经有 9 个有效的代码片段（尽管因为我们重新声明变量 `number1` 而删除了一个）。JShell 推断`$10`的类型是 `int`，这是因为表达式 `11 + 5` 将两个 `int` 值相加，结果是一个 `int`。表达式还可以包括一个或多个方法调用。让我们再次用 `/list` 列出当前的代码片段，如下所示：

```
jshell> /list

1 : System.out.println("欢迎学习 Java 编程!")
2 : System.out.print("欢迎学习 Java 编程!")
4 : int number1 = 30;
5 : number1 = 45
6 : int number2 = 72;
7 : int sum = number1 + number2;
8 : System.out.println("欢迎学习 Java 编程!")
9 : System.out.println("欢迎学习 Java 编程!")
10 : 11 + 5

jshell>
```

注意，隐式声明的变量`$10`在列表中仅显示为 `10`，没有`$`。

A.2.6 使用隐式声明的变量

可以在表达式中使用隐式声明的变量。例如，以下命令将 `number1`（45）和`$10`（16）相加的结果赋给现有的变量 `sum`：

```
jshell> sum = number1 + $10
sum ==> 61

jshell>
```

现在的代码片段列表如下所示：

```
jshell> /list
```

^① 隐式声明的变量是 JShell 为了方便交互而重新解释标准 Java 的另一个例子。在常规 Java 程序中，所有变量都必须显式声明。

```
1 : System.out.println("欢迎学习 Java 编程!")
2 : System.out.print("欢迎学习 Java 编程!")
4 : int number1 = 30;
5 : number1 = 45
6 : int number2 = 72;
7 : int sum = number1 + number2;
8 : System.out.println("欢迎学习 Java 编程!")
9 : System.out.println("欢迎学习 Java 编程!")
10 : 11 + 5
11 : sum = number1 + $10
```

```
jshell>
```

A.2.7 查看变量的值

可以随时通过输入变量的名称并按 Enter 键来查看变量的值，如下所示：

```
jshell> sum
sum ==> 61
```

```
jshell>
```

这样做不会创建另一个隐式声明的变量。

A.2.8 重置 JShell 会话

可以执行 `/reset` 命令从 JShell 会话中清除之前的所有代码，如下所示：

```
jshell> /reset
| 正在重置状态。
```

```
jshell> /list
```

```
jshell>
```

后续的 `/list` 命令没有产生任何输出，从而确认之前的所有代码片段都已清除。注意，JShell 经常会显示一些提示消息，例如：

```
| 正在重置状态。
```

它们可以很好地帮助你熟悉 JShell。在 A.11.5 节中，我们将演示如何更改 JShell 反馈模式，使其更详细或更简洁。

A.2.9 编写多行语句

接着尝试输入一个 `if` 语句来判断 45 是否小于 72。首先将 45 和 72 存储到隐式声明的变量中，如下所示：

```
jshell> 45
```

```
$1 ==> 45

jshell> 72
$2 ==> 72

jshell>
```

然后输入 `if` 语句的开头部分：

```
jshell> if ($1 < $2) {
...>    _
```

JShell 知道这个 `if` 语句不完整，因为输入了左大括号，但没有提供主体或右大括号。因此，JShell 显示延续提示符 `...>`，可以在其中输入更多控制语句。它还将第二行缩进 4 个空格。在本例中，我们人为添加一个下划线来指示光标当前所在位置。以下代码完成并求值该 `if` 语句：

```
jshell> if ($1 < $2) {
...>    System.out.printf("%d < %d%n", $1, $2);
...> }
45 < 72

jshell>
```

出现第二个延续提示符是因为 `if` 语句仍然缺少右大括号 (`}`)。输入右大括号时，JShell 会自动将其与“`if`”关键字中的“`i`”对齐。在 `if` 语句的主体中，`System.out.printf` 语句末尾的语句终止分号 (`;`) 是必需的。JShell 为每个多行代码片段（例如 `if` 语句）分配一个代码片段 ID。现在的代码片段列表如下所示：

```
jshell> /list

1 : 45
2 : 72
3 : if ($1 < $2) {
    System.out.printf("%d < %d%n", $1, $2);
}

jshell>
```

A.2.10 编辑代码片段

有的时候，我们可能想基于当前 JShell 会话中的现有代码片段来新建一个。例如，假设要创建一个 `if` 语句来判断 `$1` 是否大于 `$2`。执行此任务的语句如下：

```
if ($1 > $2) {
    System.out.printf("%d > %d%n", $1, $2);
}
```

它与 A.2.9 节中的 `if` 语句几乎完全相同，因此最好的做法是编辑现有语句，这比从头输入容易得多。编辑代码片段时，JShell 会将编辑后的版本保存为一个新的代码片段，其 ID 为序列中的下一个 ID。

编辑单行代码片段

编辑代码片的步骤如下所示：

1. 按上箭头键来定位它；
2. 按箭头键导航到代码片段中需要更改的位置；
3. 在代码片段中进行更改；
4. 然后按 `Enter` 键来求值。

我们将在 A.12 节总结用于编辑代码片段的键盘快捷键。

编辑多行代码片段

对于较大的多行代码片段（例如 `if` 语句），还可以使用 JShell 的 `/edit` 命令来打开 **JShell Edit Pad**（JShell 编辑板），以便编辑整个代码片段。在 JShell Edit Pad 中，显示了到目前为止输入的所有有效代码片段。如果只想编辑特定的代码片段，请指定具体的代码片段 ID，如下所示：

```
/edit id
```

因此，`/edit 3` 命令将在 JShell Edit Pad 中显示 A.2.9 节中的 `if` 语句：



还可以编辑代码片段组和代码片段范围，如 A.2.4 节所述。注意，JShell Edit Pad 不会显示代码片段 ID。另外，这是一种模态窗口。换言之，一旦打开，便无法在 JShell 提示符下输入代码片段或命令，除非关闭 JShell Edit Pad 窗口。

JShell Edit Pad 仅支持基本编辑功能，它支持执行以下操作：

- 单击以将光标插入到特定位置，随后可以开始输入文本；
- 使用键盘上的箭头键移动光标；
- 拖动鼠标以选择文本；
- 按 `Delete` 或退格键删除文本；

- 使用操作系统的键盘快捷键来剪切、复制和粘贴文本；
- 输入文本，包括与当前正在编辑的代码片段分开的新代码片段。

在本例中，请在 `if` 语句的第一行和第二行中，选择每个小于操作符 (`<`)，并将其更改为大于 (`>`) 操作符，然后单击“Accept”按钮，以创建一个包含编辑后的代码的新 `if` 语句。一旦单击“Accept”，JShell 就会立即执行新的 `if` 语句并显示结果（如果有的话）。由于 `$1`（45）不大于 `$2`（72），因此 `System.out.printf` 语句不会执行。^①

如果想立即返回到 JShell 提示符，而不是单击“Accept”，那么可以单击“Exit”以执行编辑后的代码片段并关闭 JShell Edit Pad。单击“Cancel”会关闭 JShell Edit Pad 并放弃自上次单击“Accept”或自从启动 JShell Edit Pad 以来所做的任何更改（如果尚未单击“Accept”的话）。

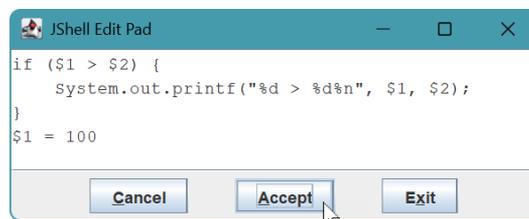
如果更改或创建多个代码片段，然后单击“Accept”或“Exit”，那么 JShell 会将 JShell Edit Pad 中当前的内容与之前保存的代码片段进行比较。然后，它会执行每个修改过的或者新增的代码片段。

通过 JShell Edit Pad 添加新的代码片段

为了演示 JShell Edit Pad 确实在单击“Accept”时立即执行代码片段，下面让我们在 JShell Edit Pad 中的 `if` 语句之后输入以下语句来将 `$1` 的值更改为 `100`：

```
$1 = 100
```

然后单击“Accept”。



每次修改变量值时，JShell 都会立即显示该变量的名称和新值，如下所示：

```
jshell> /edit 3
$1 ==> 100
```

请单击“Exit”以关闭 JShell Edit Pad 并返回到 `jshell>`提示符。

^① 这里可以用一个 `if...else` 语句在条件为 `false` 时输出一些东西。但是，本节旨在配合第 2 章使用，而在第 2 章中，我们只介绍了单选 `if` 语句。

下面列出了当前的代码片段——注意，每个多行 `if` 语句只有一个 ID：

```
jshell> /list

1 : 45
2 : 72
3 : if ($1 < $2) {
    System.out.printf("%d < %d\n", $1, $2);
  }
4 : if ($1 > $2) {
    System.out.printf("%d > %d\n", $1, $2);
  }
5 : $1 = 100

jshell>
```

再次执行新的 `if` 语句

以下命令使用更新后的 `$1` 值重新执行新的 `if` 语句（ID 4）：

```
jshell> /4
if ($1 > $2) {
  System.out.printf("%d > %d\n", $1, $2);
}
100 > 72

jshell>
```

条件 `$1 > $2` 现在为 `true`，因此会执行 `if` 语句的主体语句。当前的代码片段列表如下：

```
jshell> /list

1 : 45
2 : 72
3 : if ($1 < $2) {
    System.out.printf("%d < %d\n", $1, $2);
  }
4 : if ($1 > $2) {
    System.out.printf("%d > %d\n", $1, $2);
  }
5 : $1 = 100
6 : if ($1 > $2) {
    System.out.printf("%d > %d\n", $1, $2);
  }

jshell>
```

A.2.11 退出 JShell

要终止当前的 JShell 会话，可以执行 `/exit` 命令或者按键盘快捷键 `Ctrl+d`（或 `control+d`）。

A.3 JShell 中的命令行输入

[本节可以在学习完第 2 章和本附录的前面几节之后阅读。]

在第 2 章中，我们使用 `Scanner` 对象执行命令行输入，如下所示：

```
Scanner input = new Scanner(System.in);

System.out.print("输入第一个整数: ");
int number1 = input.nextInt();
```

上述代码创建一个 `Scanner`，提示用户输入，然后使用 `Scanner` 的 `nextInt` 方法读取一个值。程序将等待用户输入一个整数并按 `Enter` 键，然后才会继续执行下一个语句。屏幕交互如下所示：

```
输入第一个整数: 45
```

本节将演示在 JShell 中如何实现这样的交互。

创建 Scanner

启动一个新的 JShell 会话或 `/reset` 当前会话，然后创建一个 `Scanner` 对象，如下所示：

```
jshell> Scanner input = new Scanner(System.in)
input ==> java.util.Scanner[delimiters=\p{javaWhitespace}+ ...
\E][infinity string=\Q∞\E]

jshell>
```

这里不需要使用 `import` 语句来导入 `Scanner`。JShell 会自动导入 `java.util` 包和其他一些包（完整列表请参见 A.9 节）。创建一个对象时，JShell 会显示它的文本表示形式。`input ==>` 右边的符号就是 `Scanner` 的文本表示形式（可以安全地忽略这些内容）。

提示输入并读取值

接下来，让我们提示用户输入，如下所示：

```
jshell> System.out.print("输入第一个整数: ")
输入第一个整数:
jshell>
```

该语句的输出会立即显示，然后是一个新的 `jshell>` 提示符。现在，请执行输入语句：

```
jshell> int number1 = input.nextInt()
-
```

JShell 会等待输入。输入光标（这里用下划线表示）位于 `jshell>` 提示符下方，而不是像第 2 章中那样位于“输入第一个整数:”的右侧。现在，请输入一个整数并按 `Enter` 键将其赋给 `number1`——上述代码段的执行现已完成，因此会显示一个新的 `jshell>` 提示符，如下所示：

```
jshell> int number1 = input.nextInt()
45
number1 ==> 45

jshell>
```

虽然可以在 JShell 中使用 `Scanner` 进行命令行输入，但这颇为繁琐，而且在大多数情况下都是不必要的。上述交互的目标是将一个 `int` 存储到变量 `number1` 中。完全可以在 JShell 中使用简单的赋值操作来完成：

```
jshell> int number1 = 45
number1 ==> 45

jshell>
```

总之，我们在 JShell 中通常会直接执行赋值，而不是使用命令行来输入一个值。

A.4 声明和使用类

[本节可以在学习完第 8 章之后阅读。]

A.2 节演示了 JShell 的基本功能。本节将创建一个类并操作该类的对象。我们将使用图 8.1 中那个版本的 `Account` 类。

A.4.1 在 JShell 中创建类

启动一个新的 JShell 会话（或 `/reset` 当前会话），然后声明 `Account` 类。在本例中，我们省略了图 8.1 中的注释。

```
jshell> public class Account {
...>     private String name;
...>
...>     public void setName(String name) {
...>         this.name = name;
...>     }
...>     public String getName() {
...>         return name;
...>     }
```

```
...> }
...> }
| 已创建 类 Account
```

```
jshell>
```

一旦输入类的右大括号，JShell 就会识别出完整的类，并显示以下提示消息：

```
| 已创建 类 Account
```

然后，它会显示一个新的 `jshell>` 提示符。注意，`Account` 类主体中的所有分号都是必需的。

为了节省时间，可以将现有的源代码文件直接加载到 JShell 中，而不是一行一行地输入类的代码，这具体将在 A.4.6 节解释。虽然可以在自定义类（和其他类型）上指定像 `public` 这样的访问修饰符，但 JShell 实际上会忽略顶级类型（即，未嵌套在其他类型中的类型）上的所有访问修饰符。

查看已声明的类

可以执行 `/types` 命令来查看到目前为止已声明的类的名称，如下所示：^①

```
jshell> /types
| class Account

jshell>
```

A.4.2 显式声明引用类型的变量

以下代码创建 `Account` 变量 `account`：

```
jshell> Account account
account ==> null

jshell>
```

引用类型变量的默认值为 `null`。

A.4.3 创建对象

我们可以新建对象的同时初始化它。以下命令创建一个名为 `account` 的 `Account` 变量，并使用一个新对象对其进行初始化：

^① `/types` 命令将显示你在当前会话中声明的所有类型，包括 `class`、`record`、`interface` 和 `enum`。

```
jshell> account = new Account()
account ==> Account@28c97a5
```

```
jshell>
```

`Account@28c97a5` 这个奇怪的符号是新 `Account` 对象的默认文本表示形式。如果一个类提供了一个自定义的 `toString` 方法，那么这里就会输出它的自定义文本表示形式。`@`符号后面的值是对象的哈希码。

用 `Account` 对象初始化一个隐式 `Account` 变量

如果只用表达式 `new Account()` 来创建一个对象，JShell 会将该对象赋给一个 `Account` 类型的隐式变量，如下所示：

```
jshell> new Account()
$8 ==> Account@6d5380c2
```

```
jshell>
```

注意，该对象的哈希码（`6d5380c2`）有别于之前的 `Account` 对象的哈希码（`28c97a5`）。两个哈希码通常都不同，但不能保证。

查看声明的变量

可以使用 JShell 的 `/vars` 命令查看到目前为止已声明的所有变量，如下所示：

```
jshell> /vars
|   Scanner input = java.util.Scanner[delimiters=\p{javaWhitespace}+
| [position=2][match valid=true][need input=false][source closed=false]
| [skipped=false][group separator=\x{2c}][decimal separator=\x{2e}]
| [positive prefix=][negative prefix=\Q-\E][positive suffix=][negative suffix=]
| [NaN string=\QNaN\E][infinity string=\Q∞\E]
|   int number1 = 45
|   Account account = Account@28c97a5
|   Account $8 = Account@6d5380c2
```

```
jshell>
```

对于每个变量，JShell 会输出类型和变量名，后跟一个等号和变量的文本表示形式。

A.4.4 操作对象

一旦拥有了一个对象，就可以调用它的方法。事实上，我们已经在之前的代码段中调用了 `System.out` 对象的 `println`、`print` 和 `printf` 方法。

以下命令设置 `account` 对象的 `name` 字段（该字段表示当前账户的“姓名”）：

```
jshell> account.setName("赵敏")
```

```
jshell>
```

`setName` 方法具有 `void` 返回类型，因此不返回值，并且 JShell 中也不会出现额外的输出。

以下命令获取 `account` 对象的 `name`：

```
jshell> account.getName()
```

```
$10 ==> "赵敏"
```

```
jshell>
```

`getName` 方法返回一个 `String`。调用返回值的方法时，JShell 会将该值存储到一个隐式声明的变量中。在本例中，`$10` 的类型被推断为 `String`。当然，完全可以将上述方法调用的结果赋给一个显式声明的变量。

在语句中使用方法的返回值

如果在一个更大的语句中调用方法，那么返回值将直接在该语句中使用，而不是被存储下来。例如，以下命令使用 `println` 来显示 `account` 对象的 `name`：

```
jshell> System.out.println(account.getName())
```

```
赵敏
```

```
jshell>
```

A.4.5 为表达式创建一个有意义的变量名

可以为之前赋给隐式变量的值指定一个有意义的变量名。例如，请按上箭头键召回以下代码段：

```
jshell> account.getName()
```

然后按 `Shift + Tab, v` 键，即同时按 `Shift` 和 `Tab` 键，然后松开这些键，再按 `v` 键。^①JShell 会推断表达式的类型并开始一个变量声明。由于 `account.getName()` 返回一个 `String`，因此 JShell 会在表达式之前插入 `String` 和一个等号 (`=`)，如下所示：

```
jshell> account.getName()
```

```
jshell> String _ = account.getName()
```

JShell 还会将光标（如上面的下划线_所示）定位在 `=` 之前，以方便你直接输入变量名，如下所示：

^① 译注：千万不要同时按 `Shift`、`Tab` 和 `v`。

```
jshell> String name = account.getName()
name ==> "赵敏"
```

```
jshell>
```

按下 Enter 键后，JShell 会执行新的代码段并获取的值存储到指定的变量中。

A.4.6 保存和打开代码段文件

可以将一个会话的有效代码段保存到一个文件中。以后可以根据需要将其加载到 JShell 会话中。

将代码段保存到文件

/save 命令只保存有效的代码段，如下所示：

```
/save 文件名
```

默认情况下，该文件将在启动 JShell 的文件夹中创建。要将文件存储到不同的位置，请指定文件的完整路径。

从文件加载代码段

保存了代码段后，可以使用 /open 命令重新加载，如下所示：

```
/open 文件名
```

它会执行文件中保存的每个代码段。

使用 /open 加载 Java 源代码文件

甚至可以使用 /open 打开现有的 Java 源代码文件。例如，假设要试验如图 8.1 所示的 Account 类（就 A.4.1 节所做的那样），可以通过从源文件 Account.java 加载该类来节省时间，而不是将它的代码一行一行地输入到 JShell 中。在命令窗口中，可以先切换到到 Account.java 所在的文件夹，执行 JShell，然后使用以下命令将类声明加载到 JShell 中：

```
/open Account.java
```

对于不在当前文件夹中的文件，也可以通过指定该文件的完整路径名来加载。A.9 节演示了如何在 JShell 中使用现有的已编译类。

```
indexOf(          intern()          isBlank()
isEmpty()        lastIndexOf(       length()
lines()          matches(          notify()
notifyAll()      offsetByCodePoints( regionMatches(
repeat(          replace(          replaceAll(
replaceFirst(    resolveConstantDesc( split(
splitWithDelimiters( startsWith(       strip()
stripIndent()   stripLeading()    stripTrailing()
subSequence(     substring(        toCharArray()
toLowerCase(    toString()        toUpperCase(
transform(       translateEscapes() trim()
wait()
```

```
jshell> greeting.
```

在列表后面，还显示了一个新的 `jshell>` 提示符，其中包含到目前为止输入的内容。该列表包含 `String` 类自己声明的方法以及所有 `Java` 类都具有的几个 `Object` 方法（参见第 9 章）。在方法名称列表中，后跟 `()` 的方法不需要参数，而那些只跟着 `()` 的方法则需要至少一个参数，或者是可以使用不同数量和/或类型的参数调用的重载方法（参见 5.7 节）。

例如，假定要使用 `String` 的 `endsWith` 方法来检查 `greeting` 是否以“好”结束。由于只有一个方法以“en”开头，所以可以输入 `en` 然后按 `Tab` 键来自动补全 `endsWith`，如下所示：

```
jshell> greeting.endsWith(
```

JShell 会自动插入方法调用的左括号。然后，可以自行完成整个代码段，如下所示：

```
jshell> greeting.endsWith("好")
$2 ==> true
```

```
jshell>
```

A.5.2 自动补全 JShell 命令

JShell 命令也可以自动补全。如果输入 `/`，然后按 `Tab` 键，那么会显示 JShell 命令的完整列表，如下所示：

```
jshell> /
/!      /?      /drop    /edit    /env      /exit
/help   /history /imports /list    /methods /open
/reload /reset  /save    /set     /types   /vars
```

<再次按 `Tab` 可查看提要>

```
jshell> /
```

此时如果输入 `h` 并按 `Tab` 键，那么 JShell 会只显示以 `/h` 开头的命令，如下所示：

```
jshell> /h
/help      /history
```

```
floorMod(      fma(          getExponent(  hypot(
incrementExact( log(          log10(        log1p(
max(           min(          multiplyExact( multiplyFull(
multiplyHigh(  negateExact(  nextAfter(   nextDown(
nextUp(       pow(          random()     rint(
round(        scalb(        signum(      sin(
sinh(         sqrt(         subtractExact( tan(
tanh(         toDegrees(    toIntExact(  toRadians(
ulp(          unsignedMultiplyHigh(
jshell> Math.
```

JShell 自动补全会显示所有可以出现在点 (.) 右侧的内容。本例输入了类名和一个点 (.), 因此 JShell 只显示该类的静态成员。没有跟任何圆括号的名称 (E、PI 和 TAU) 是该类的静态变量。其他所有名称都是该类的静态方法:

- 任何后跟一对圆括号的方法名 (本例只有 `random`) 不需要参数;
- 任何只跟一个左圆括号的方法名需要至少一个参数或者被重载。

可以很容易地查看像 PI 和 E 这样的常量的值, 如下所示:

```
jshell> Math.PI
$1 ==> 3.141592653589793

jshell> Math.E
$2 ==> 2.718281828459045
```

A.6.2 查看方法的参数

假设我们想测试 Math 的 `pow` 方法, 但不知道所需的参数。为此, 我们可以输入以下内容:

```
jshell> Math.p
```

然后按 `Tab` 键来自动补全名称 `pow`, 如下所示:

```
jshell> Math.pow(
```

没有其他方法以 "pow" 开头, 所以 JShell 会插入左括号来表示方法调用的开始。接下来, 可以再一次 `Tab` 键来查看方法的参数, 如下所示:

```
jshell> Math.pow(
$1 $2
```

签名:

```
double Math.pow(double a, double b)
```

<再次按 `Tab` 可查看文档>

```
jshell> Math.pow(
```

JShell 显示的内容如下所示:

- 当前会话中的变量（\$1 和\$2）可以被用作 pow 的实参；
- 方法的返回类型、名称和完整参数列表，表明该方法需要两个 double 参数；
- "<再次按 Tab 可查看文档>";
- 包含到目前为止输入内容的下一个 jshell>提示符。

A.6.3 查看方法的文档

JShell集成了 Java API 文档，所以可以在 JShell 中方便地查看文档，而不必使用单独的 Web 浏览器。假想在补全代码段之前了解更多关于 pow 的信息。可以再按一次 Tab 键来查看方法的 Java 文档（被称为 javadoc）。我们删掉了一些文档文本，并用省略号（...）代替以节省空间——在你自己的 JShell 会话中，请尝试这些步骤来阅读完整文本。

```
jshell> Math.pow(
double Math.pow(double a, double b)
Returns the value of the first argument raised to the power of the second argument.
Special cases:
    * If the second argument is positive or negative zero, then the result is 1.0.
...
<再次按 Tab 可查看下一页>
jshell> Math.pow(
```

对于冗长的文档，JShell 会显示部分文档，后面跟着以下消息：

```
<再次按 Tab 可查看下一页>
```

可以按 Tab 键来查看下一页文档。下一个 jshell>提示符将包含到目前为止输入的代码段。

A.6.4 查看 public 字段的文档

可以通过按 Tab 键来了解更多关于类的 public 字段的信息。例如，如果输入 Math.PI，然后按 Tab 键，那么 JShell 将显示以下内容：

```
jshell> Math.PI
签名:
Math.PI:double

<再次按 Tab 可查看文档>
```

其中包括 Math.PI 的类型，并指出可以再次按 Tab 键来查看文档。再次按 Tab 键后会显示以下内容：

```
jshell> Math.PI
Math.PI:double
The double value that is closer than any other to pi ( $\pi$ ), the ratio of the circumference of a circle to its diameter.
jshell> Math.PI
```

并且下一个 `jshell>` 提示符将包含到目前为止输入的代码段。

A.6.5 查看类的文档

也可以输入一个类名然后按 `Tab` 键来查看该类的完整限定名。例如，输入 `Math` 然后按 `Tab` 键会输出以下内容：

```
jshell> Math
Math          MathContext
```

```
签名:
java.lang.Math
```

```
<再次按 Tab 可查看文档>
jshell> Math
```

这表明 `Math` 类在 `java.lang` 包中。再次按 `Tab` 键会显示该类的文档，如下所示（这里只列出的开头部分内容）：

```
jshell> Math
java.lang.Math
The class Math contains methods for performing basic numeric operations such as the
elementary exponential, logarithm, square root, and trigonometric functions.
...
```

```
<再次按 Tab 可查看下一页>
jshell> Math
```

在本例中，有更多的文档内容可以查看，因此可以按 `Tab` 键翻页。无论是否查看剩余的文档，`jshell>` 提示符都包含到目前输入的代码段。

A.6.6 查看方法的重载版本

许多类都有重载的方法。按 `Tab` 键来查看一个重载方法的参数时，`JShell` 会显示完整的重载列表，其中包含每个重载的参数。例如，`Math.abs` 方法有如下所示的 4 个重载：

```
jshell> Math.abs(
$1 $2

签名:
int Math.abs(int a)
long Math.abs(long a)
float Math.abs(float a)
double Math.abs(double a)
```

```
<再次按 Tab 可查看文档>
jshell> Math.abs(
```

再次按 Tab 键来查看文档时，JShell 会输出方法第一个重载的文档，如下所示：

```
jshell> Math.abs(  
int Math.abs(int a)  
Returns the absolute value of an int value.If the argument is not negative, the argument  
is returned.  
...  
  
<按 Tab 可查看下一文档>  
jshell> Math.abs(  
int Math.abs(int a)  
Returns the absolute value of an int value.If the argument is not negative, the argument  
is returned.  
...
```

然后可以再次按 Tab 键来查看列表中下一个重载的文档。同样，无论是否查看剩余的文档，jshell>提示符都包含到目前为止输入的代码段。

A.6.7 探索特定对象的成员

A.6.1~A.6.6 节的探索功能也适用于特定对象的成员。让我们启动一个新的会话，然后创建并探索一个 String 对象，如下所示：

```
jshell> String dayName = "星期一(Monday)"  
dayName ==> "星期一(Monday)"  
  
jshell>
```

为了查看 dayName 对象的方法，请输入"dayName."，然后按 Tab 键，如下所示：

```
jshell> dayName.  
charAt(          chars()          codePointAt(  
codePointBefore( codePointCount(  codePoints()  
compareTo(      compareToIgnoreCase( concat(  
contains(       contentEquals(   describeConstable()  
endsWith(      equals(          equalsIgnoreCase(  
formatted(     getBytes(        getChars(  
getClass()     hashCode()       indent(  
indexOf(       intern()         isBlank()  
isEmpty()     lastIndexOf(    length()  
lines()       matches(        notify()  
notifyAll()   offsetByCodePoints( regionMatches(  
repeat(       replace(        replaceAll(  
replaceFirst( resolveConstantDesc( split(  
splitWithDelimiters( startsWith(    strip()  
stripIndent() stripLeading()  stripTrailing()  
subSequence(  substring(     toCharArray()  
toLowerCase( toString()     toUpperCase(  
transform(    translateEscapes() trim()  
wait(  
  
jshell> dayName.
```

探索 toUpperCase

让我们研究一下 `toUpperCase` 方法。继续输入 `"toU"`（注意大小写）并按 `Tab` 键来自动补全它的名称，如下所示：

```
jshell> dayName.toUpperCase(
```

再按 `Tab` 键来查看该方法的参数，如下所示：

```
jshell> dayName.toUpperCase(
签名:
String String.toUpperCase(Locale locale)
String String.toUpperCase()
```

<再次按 `Tab` 可查看文档>

```
jshell> dayName.toUpperCase(
```

该方法有两个重载。现在可以按 `Tab` 键来阅读每个重载的信息，或者通过指定适当的参数（如果有的话）来选择想要使用的那个。在本例中，我们将使用无参版本来创建一个包含“星期一(MONDAY)”的新 `String`，所以我们只需输入方法调用的右括号并按 `Enter` 键，如下所示：

```
jshell> dayName.toUpperCase()
$2 ==> "星期一(MONDAY)"

jshell>
```

探索 substring

假设想要创建一个新 `String` 对象“DAY”，即隐式变量 `$2` 的字符的子集。为此，`String` 类提供了重载方法 `substring`。首先，输入 `"$2.subs"` 并按 `Tab` 键来自动补全该方法的名称，如下所示：

```
jshell> $2.substring(
```

然后，按 `Tab` 键来查看该方法的重载，如下所示：

```
jshell> $2.substring(
签名:
String String.substring(int beginIndex)
String String.substring(int beginIndex, int endIndex)
```

<再次按 `Tab` 可查看文档>

```
jshell> $2.substring(
```

接着，再次按 `Tab` 键来查看第一个重载的文档，如下所示：

```
jshell> $2.substring(
```

```
String String.substring(int beginIndex)
Returns a string that is a substring of this string. The substring begins with the
character at the specified index and extends to the end of this string.
...
<按 Tab 可查看下一文档>
jshell> $2.substring(
```

文档表明，方法的第一个重载允许获得从特定字符索引（即位置）开始并持续到字符串末尾的子串。但是，我们希望获得的子串"DAY"并没有持续到字符串末尾，因此再按一次 Tab 键来查看第二个重载的文档，如下所示：

```
jshell> $2.substring(
String String.substring(int beginIndex, int endIndex)
Returns a string that is a substring of this string. The substring begins at the specified
beginIndex and extends to the character at index endIndex - 1. Thus the length of the
substring is endIndex-beginIndex.
...
Parameters:
beginIndex - the beginning index, inclusive.
endIndex - the ending index, exclusive.
...
<再次按 Tab 可查看所有可能的输入提示；可能的输入提示总计：557>
jshell> $2.substring(
```

该重载符合我们的要求，它获取从指定起始索引到指定结束索引减 1 处的子串。现在，我们可以完成对 `substring` 的调用并按 Enter 键来查看结果，如下所示：

```
jshell> $2.substring(7, 10)
$3 ==> "DAY"

jshell>
```

A.7 声明方法

[本节可以在学习完第 5 章以及本附录前面的内容之后阅读。]

可以使用 JShell 来制作方法的原型。例如，假设要编写代码来显示 1~10 的整数的立方。为了方便讨论，我们将定义两个方法，如下所示：

- `displayCubes` 方法内部将循环迭代 10 次，每次都调用 `cube` 方法；
- `cube` 方法将接收一个 `int` 值并返回该值的立方。

A.7.1 前向引用未声明的方法——声明 `displayCubes` 方法

让我们从 `displayCubes` 方法开始。启动一个新的 JShell 会话或 `/reset` 当前会话，然后输入以下方法声明：

```
void displayCubes() {
    for (int i = 1; i <= 10; ++i) {
        System.out.printf("%d 的立方是%d%n", i, cube(i));
    }
}
```

完成方法声明时，JShell 将显示以下提示：

```
| 已创建方法 displayCubes(), 不过, 它无法调用, 直至方法 cube(int)已声明
```

输入方法体的左大括号后，JShell 会在每一行显示延续提示符 (...>)，直到通过输入右大括号完成方法声明。此外，尽管 JShell 显示"已创建方法 displayCubes()"，但它同时也指出该方法在声明 "cube(int)" 之前不能调用。这在 JShell 中是允许的，它认识到 displayCubes 依赖于一个未声明的方法 (cube) —— 这被称为**前向引用** (forward referencing) 一个未声明的方法。一旦定义了 cube，就可以调用 displayCubes。

A.7.2 声明一个之前未声明的方法

接着，让我们声明 cube 方法，但故意通过返回实参的平方而不是立方来制造一个逻辑错误，如下所示：

```
jshell> int cube(int x) {
...>     return x * x;
...> }
| 已创建方法 cube(int)
```

此时，可以使用 JShell 的 /methods 命令来列出当前 JShell 会话中声明的方法，如下所示：

```
jshell> /methods
| void displayCubes()
| int cube(int)

jshell>
```

A.7.3 测试 cube 并替换它的声明

现在 cube 方法已经被声明，让我们用实参 2 来测试它，如下所示：

```
jshell> cube(2)
$3 ==> 4

jshell>
```

根据该方法当前的实现，它返回 4 (即 $2 * 2$)。然而，该方法的目的是计算其实参的立方，所以结果应该是 8 ($2 * 2 * 2$)。可以编辑 cube 的代码段来纠正这个问题。这一次让我们直接在 JShell 中编辑 cube 的代码段，而不是使用 JShell Edit Pad。按上箭头键三次来重新显示 cube 的定义，如下所示：

```
jshell> int cube(int x) {  
...>     return x * x;  
...> }
```

然后使用箭头键将光标移动到分号并编辑语句，使该方法如下所示，然后按 Enter 键：

```
jshell> int cube(int x) {  
...>     return x * x * x;  
...> }  
| 已修改方法 cube(int)  
  
jshell>
```

JShell 确认 `cube` 方法已被修改。

A.7.4 测试更新后的 `cube` 方法和 `displayCubes` 方法

现在 `cube` 方法已被正确声明，让我们使用实参 2 和 10 来再次测试它，如下所示：

```
jshell> cube(2)  
$5 ==> 8
```

```
jshell> cube(10)  
$6 ==> 1000
```

```
jshell>
```

该方法正确地返回了 2 的立方（即 8）和 10 的立方（即 1000），并将结果存储到隐式变量 `$5` 和 `$6` 中。

现在，让我们测试 `displayCubes`。输入 "di" 并按 Tab 键，JShell 会自动补全该名称，包括方法调用的完整一对圆括号，这是因为 `displayCubes` 不接收任何参数，如下所示：

```
jshell> displayCubes()  
1 的立方是 1  
2 的立方是 8  
3 的立方是 27  
4 的立方是 64  
5 的立方是 125  
6 的立方是 216  
7 的立方是 343  
8 的立方是 512  
9 的立方是 729  
10 的立方是 1000  
  
jshell>
```

A.8 异常

[本节可以在学习完第 6 章和本附录前面的内容之后阅读。]

6.12 节介绍了 Java 的异常处理机制，演示了如何捕获当数组索引越界时发生的异常。在 JShell 中，不需要手动捕获异常——它会自动捕获每个异常并显示关于它的信息，然后显示下一个 JShell 提示符，所以可以继续会话。这对于在常规 Java 程序中必须捕获的受检异常（10.5 节）尤其重要。如你所知，在常规程序中为了捕获异常，需要将代码包装在 `try...catch` 语句中。通过自动捕获所有异常，JShell 使我们可以更容易地试验抛出受检异常的方法。

在以下新的 JShell 会话中，我们声明一个 `int` 值的数组，然后演示有效和无效的数组访问表达式。

```
jshell> int[] values = {10, 20, 30}
values ==> int[3] { 10, 20, 30 }

jshell> values[1]
$2 ==> 20

jshell> values[10]
| 异常错误 java.lang.ArrayIndexOutOfBoundsException: Index 10 out of
| bounds for length 3
|         at (#3:1)

jshell>
```

代码段 `values[10]` 试图访问一个越界元素，这会导致 `ArrayIndexOutOfBoundsException`。即使没有将代码包装在 `try...catch` 中，JShell 也会捕获该异常并显示它的 `String` 表示形式。这包括异常的类型和一条错误消息（在本例中，是无效的索引 10），后面跟着一个栈跟踪，指示问题发生的位置。以下符号：

```
| at (#3:1)
```

表明该异常发生在 ID 为 3 的代码段的第 1 行。5.10 节讨论了方法调用栈。“栈跟踪”显示了当异常发生时方法调用栈上的方法。在一个典型的栈跟踪中，包含几个像这里输出的“at”行——每个栈帧一行。在显示栈跟踪之后，会出现下一个 `jshell>` 提示符。第 10 章详细讨论了栈跟踪。

A.9 导入类和将包添加到类路径

[本节可以在学习完第 11 章以及本附录前面的内容之后阅读。]

JShell 可以帮助我们在程序中使用类和类库之前进行试验。可以从 Java 的 API 和我们添加

到 JShell 的类路径和模块路径的第三方库中导入类型。几个包被 Java 开发者如此普遍地使用，以至于 JShell 会自动导入它们。要改变这一行为，可以使用 JShell 的以下命令：

```
/set start
```

具体将在 A.11 节中介绍。

查看导入的包

本节假设在 ch11 示例文件夹下执行 jshell。JShell 的 `/imports` 命令列出了当前会话的导入声明。开始一个新的 JShell 会话时，以下包会被自动导入：

```
jshell> /imports
| import java.io.*
| import java.math.*
| import java.net.*
| import java.nio.file.*
| import java.util.*
| import java.util.concurrent.*
| import java.util.function.*
| import java.util.prefs.*
| import java.util.regex.*
| import java.util.stream.*

jshell>
```

`java.lang` 包的内容在 JShell 中始终可用，在任何 Java 源代码文件中亦是如此。

向类路径添加包

也可以导入自己的或第三方的包，以便在 JShell 中进行实验。JShell 的 `/env -class-path` 命令将包添加到 JShell 的 CLASSPATH（类路径）中，CLASSPATH 告诉 JShell 其他包的位置。

^①然后，可以使用 `import` 声明在 JShell 中实验包的内容。

在 JShell 中使用 Jackson

第 11 章介绍了 Jackson 开源库，它用于将对象序列化为 JSON 格式。我们准备将它添加到 JShell 的 CLASSPATH 中，然后在 JShell 中将一个对象序列化为 JSON 字符串。如果当前有一个打开的 JShell 会话，请先执行 `/exit` 终止它。然后，将目录切换到 ch11 示例文件夹，并启动一个新的 JShell 会话。

^① JShell 还提供了 `-module-path` 和 `-add-modules` 选项来添加模块，这样就可以从这些模块中导入包。

将包的位置添加到 CLASSPATH

ch11 文件夹的 `libs` 子文件夹包含我们在第 11 章中使用的 Jackson 库组件。以下命令将此包添加到 Windows 上的 CLASSPATH 中——确保你输入完整的类路径，没有空格。

```
jshell> /env -class-path libs/jackson-core-2.18.1.jar;libs/
jackson-databind-2.18.1.jar;libs/jackson-annotations-2.18.1.jar
| 正在设置新选项并还原状态。

jshell>
```

如果使用 macOS，请用冒号 (:) 替换分号 (;)。`.jar` (Java 归档) 文件是包含已编译的 Jackson 库 Java 代码的 ZIP 文件。我们在这里使用了 `libs` 文件夹的相对路径，但是也可以指定绝对路径。

从包中导入类

现在可以导入 `ObjectMapper` 以便在 JShell 中使用。以下命令演示了如何导入 `ObjectMapper` 类并显示当前会话中的完整导入列表：

```
jshell> import com.fasterxml.jackson.databind.ObjectMapper;

jshell> /imports
| import java.io.*
| import java.math.*
| import java.net.*
| import java.nio.file.*
| import java.util.*
| import java.util.concurrent.*
| import java.util.function.*
| import java.util.prefs.*
| import java.util.regex.*
| import java.util.stream.*
| import com.fasterxml.jackson.databind.ObjectMapper

jshell>
```

创建一个要序列化为 JSON 的 List<Account>

让我们创建一个 `List<Account>` 以便在 JShell 中序列化，如下所示：

```
jshell> List<Account> accounts = List.of(
...> new Account(100, "张无忌", 24.98),
...> new Account(200, "赵敏", 345.67));
accounts ==> [Account[accountNumber=100, name=张无忌, balance=24. ...
name=赵敏, balance=345.67]]
```

```
jshell>
```

使用导入的 ObjectMapper 类

现在可以使用 `ObjectMapper` 类。首先创建一个 `ObjectMapper` 并利用 JShell 的自动补全功能来显示可用方法的列表。共有 247 个自动补全项，所以这里只显示其中几个，并用省略号表示还有更多。

```
jshell> var mapper = new ObjectMapper();
mapper ==> com.fasterxml.jackson.databind.ObjectMapper@6c49835d

jshell> mapper.
<再次按 Tab 可查看所有可能的输入提示；可能的输入提示总计：247>
jshell> mapper.
acceptJsonFormatVisitor(          activateDefaultTyping(
activateDefaultTypingAsProperty(  addHandler(
addMixIn(                          addMixInAnnotations(
...
jshell> mapper.
```

接着，使用 `ObjectMapper` 的 `writerWithDefaultPrettyPrinter` 方法来获取一个 `ObjectWriter`，它可以输出带有缩进的 JSON 文本，以提高可读性。

```
jshell> var prettyWriter = mapper.writerWithDefaultPrettyPrinter()
prettyWriter ==> com.fasterxml.jackson.databind.ObjectWriter@711f39f9

jshell>
```

最后，我们使用 `ObjectWriter` 的 `writeValueAsString` 方法来获取 `accounts` 的 JSON 表示，并显示返回的 `String`。

```
jshell> System.out.println(prettyWriter.writeValueAsString(accounts))
[ {
  "accountNumber" : 100,
  "name" : "张无忌",
  "balance" : 24.98
}, {
  "accountNumber" : 200,
  "name" : "赵敏",
  "balance" : 345.67
} ]

jshell>
```

A.10 使用外部编辑器

A.2.10 节演示了用于编辑代码片段的 JShell Edit Pad。此工具仅提供简单的编辑功能。许多程序员喜欢使用更强大的编辑器。可以使用 JShell 的 `/set editor` 命令指定自己喜欢的外部编辑器。

例如，执行如下所示的 JShell 命令：

```
jshell> /set editor notepad.exe
| 编辑器设置为: notepad.exe

jshell>
```

会将 `notepad.exe` 设置为当前 JShell 会话的代码片段编辑器。`/set editor` 命令的参数依赖于具体的操作系统。例如，在 macOS 上，可以使用内置的 `nano` 文本编辑器，命令如下

```
/set editor nano
```

使用自定义文本编辑器编辑代码片段

使用自定义编辑器时，每次保存代码片段，JShell 都会立即重新求值任何已更改的代码片段，并在 JShell 中输出其结果（但不包括代码片段本身）。

保留编辑器设置

可以为以后的 JShell 会话保留自己的编辑器设置，如下所示：

```
/set editor -retain 编辑器启动命令
```

将 JShell Edit Pad 恢复为默认编辑器

如果不保留自定义编辑器，后续新启动的 JShell 会话还是会使用 JShell Edit Pad。如果保留了自定义编辑器，可以使用以下命令将 JShell Edit Pad 恢复为默认编辑器：

```
/set editor -retain -default
```

A.11 JShell 命令摘要

表 A.1 总结了基本的 JShell 命令。本附录已经介绍了其中许多命令。其他命令在本节中讨论。

表 A.1 基本的 JShell 命令

命令	描述
/help 或/?	显示 JShell 的命令列表
/help intro	显示 JShell 的简要介绍
/help shortcuts	显示 JShell 快捷键说明
/list	默认情况下，列出在当前会话中输入的有效代码片段。要列出所有代码片段，请使用 <code>/list -all</code>
/!	调用并重新求值最后一个代码片段
/id	调用并重新求值具有指定 <code>id</code> 的代码片段。也可以指定几个单独的代码片段或代码片段范围，各项之间用空格分隔
/-n	调用并重新求值之前的某个代码片段——对于 <code>n</code> ，1 代表最后一个代码片段，2 代表倒数第二个，依此类推
/edit	默认情况下，打开一个 JShell Edit Pad 窗口，其中包含在当前会话中输入的有效代码片段。A.10 节讨论了如何配置外部编辑器
/save	将当前会话的有效代码片段保存到指定文件
/open	打开指定的代码片段文件，将片段加载到当前会话，并对加载的片段进行求值
/vars	显示当前会话的变量及其对应的值
/methods	显示当前会话中已声明方法的签名
/types	显示当前会话中已声明的类型
/imports	显示当前会话的 <code>import</code> 声明
/exit	终止当前 JShell 会话
/reset	重置当前 JShell 会话，删除所有代码片段
/reload	重新加载一个 JShell 会话，并执行其中的有效代码片段（参见 A.11.3 节）
/drop	从当前会话中删除指定的代码片段（参见 A.11.4 节）

<code>/env</code>	更改 JShell 环境，例如添加包或模块，以便可以在 JShell 中使用它们的类型
<code>/history</code>	列出在当前 JShell 会话中输入的所有内容，包括所有代码片段（有效的、无效的或被覆盖的）和 JShell 命令—— <code>/list</code> 命令仅列出代码片段，不列出 JShell 命令
<code>/set</code>	设置各种 JShell 配置选项，例如响应 <code>/edit</code> 命令时使用的编辑器、JShell 提示符的文本、会话启动时要指定的 <code>import</code> 等（参见 A.11.5~A.11.6 节）

A.11.1 在 JShell 中获取帮助

JShell 的帮助文档直接通过 `/help` 或 `/?` 命令来集成——`/?` 只是 `/help` 的简写。要快速了解 JShell，请执行以下命令：

```
/help intro
```

要显示 JShell 的命令列表，请执行以下命令：

```
/help
```

要获得关于给定命令选项的更多信息，请执行以下命令：

```
/help 命令
```

例如，如果执行以下命令：

```
/help /list
```

那么将显示 `/list` 命令更详细的帮助文档。类似地，执行以下命令：

```
/help /set start
```

将显示 `/set` 命令的 `start` 选项的更详细的帮助文档。要查看 JShell 中的快捷键列表，请执行以下命令：

```
/help shortcuts
```

A.11.2 `/edit` 命令：附加功能

之前已经讨论了如何使用 `/edit` 命令在 JShell Edit Pad 中加载所有有效的代码片段、具有指定 ID 的代码片段或具有指定名称的方法。除此之外，还可以指定要编辑的任何变量、方法或类型声明的标识符。例如，如果当前 JShell 会话包含一个名为 `Account` 的类的声明，那么可以执行以下命令将该类加载到 JShell Edit Pad 中：

```
/edit Account
```

A.11.3 /reload 命令

JShell 的 `/reload` 命令可以重新执行当前会话中的所有有效代码片段。以 A.2.9 节~A.2.10 节的会话为例：

```
jshell> /list

1 : 45
2 : 72
3 : if ($1 < $2) {
    System.out.printf("%d < %d%n", $1, $2);
  }
4 : if ($1 > $2) {
    System.out.printf("%d > %d%n", $1, $2);
  }
5 : $1 = 100
6 : if ($1 > $2) {
    System.out.printf("%d > %d%n", $1, $2);
  }

jshell>
```

以下命令一次性重新加载整个会话的代码片段：

```
jshell> /reload
| 正在重新启动并还原状态。
-: 45
-: 72
-: if ($1 < $2) {
    System.out.printf("%d < %d%n", $1, $2);
  }
45 < 72
-: if ($1 > $2) {
    System.out.printf("%d > %d%n", $1, $2);
  }
-: $1 = 100
-: if ($1 > $2) {
    System.out.printf("%d > %d%n", $1, $2);
  }
100 > 72

jshell>
```

每个重新加载的代码片段都以 `-:` 开头，并且在 `if` 语句的情况下，输出（如果有）会立即出现在每个 `if` 语句之后。如果不想在重新加载时列出代码片段，可以使用 `/reload` 命令的 `-quiet` 选项，如下所示：

```
jshell> /reload -quiet
| 正在重新启动并还原状态。
45 < 72
100 > 72
```

```
jshell>
```

在这种情况下，只会显示输出语句的结果。然后，可以使用 `/list` 命令查看重新加载的代码片段。

A.11.4 `/drop` 命令

可以使用 JShell 的 `/drop` 命令，后跟代码片段 ID 或标识符，从当前会话中删除一个代码片段。在以下新的 JShell 会话中，我们声明了变量 `x` 和方法 `cube`，然后通过其代码片段 ID 删除 `x`，并通过其标识符删除 `cube`。

```
jshell> int x = 10
x ==> 10

jshell> int cube(int y) {return y * y * y;}
| 已创建 方法 cube(int)

jshell> /list

  1 : int x = 10;
  2 : int cube(int y) {return y * y * y;}

jshell> /drop 1
| 已删除 变量 x

jshell> /drop cube
| 已删除 方法 cube(int)

jshell> /list

jshell>
```

A.11.5 反馈模式

JShell 有几种反馈模式，用于确定每次交互后显示的内容。要更改反馈模式，请使用 JShell 的 `/set feedback` 命令，如下所示：

```
/set feedback 模式
```

其中，*模式*可以是 `concise`、`normal`（默认）、`silent` 或 `verbose`。本附录之前所有 JShell 交互都使用了 `normal` 模式。

反馈模式 verbose

首先开启一个新的 JShell 会话，并在其中使用 `verbose` 模式，Java 新手可能会更喜欢这种模式。

```
jshell> /set feedback verbose
| 反馈模式: verbose

jshell> int x = 10
x ==> 10
| 已创建 变量 x : int

jshell> int cube(int y) {return y * y * y;}
| 已创建 方法 cube(int)

jshell> cube(x)
$3 ==> 1000
| 已创建暂存变量 $3 : int

jshell> x = 5
x ==> 5
| 已分配给 x : int

jshell> cube(x)
$5 ==> 125
| 已创建暂存变量 $5 : int

jshell>
```

请注意，额外的反馈提供了以下详细描述：

- 创建了变量 `x`；
- 创建了方法 `cube`；
- 在第一次调用 `cube` 时创建了暂存变量 `$3`；
- 将一个 `int` 赋给变量 `x`；
- 在第二次调用 `cube` 时创建了暂存变量 `$5`。

反馈模式 concise

接着，我们 `/reset` 会话，然后将反馈模式设置为 `concise`，并重复之前的会话。

```
jshell> /set feedback concise
jshell> int x = 10
jshell> int cube(int y) {return y * y * y;}
jshell> cube(x)
$3 ==> 1000
jshell> x = 5
```

```
jshell> cube(x)
$5 ==> 125
jshell>
```

注意，显示的唯一反馈是每次调用 `cube` 的结果。如果发生错误，也会显示其反馈。

反馈模式 `silent`

最后，我们 `/reset` 会话，然后将反馈模式设置为 `silent`，并重复之前的会话。

```
jshell> /set feedback silent
-> int x = 10
-> int cube(int y) {return y * y * y;}
-> cube(x)
-> x = 5
-> cube(x)
-> /set feedback normal
| 反馈模式: normal

jshell>
```

在这种情况下，`jshell>`提示符变为`->`，并且只会显示错误反馈。如果从 `Java` 源文件中复制了代码并想将其粘贴到 `JShell` 中，同时不想为每一行都显示反馈，就可以考虑使用此模式。

A.11.6 其他可用 `/set` 配置的 `JShell` 功能

我们已经演示了如何使用 `/set` 命令来设置外部代码片段编辑器和反馈模式。`/set` 命令支持使用以下命令创建自定义反馈模式：

- `/set mode`
- `/set prompt`
- `/set truncation`
- `/set format`

`/set mode` 命令创建自定义反馈模式。然后，可以使用其他三个命令来自定义 `JShell` 反馈的各个方面。这些命令的细节超出了本附录的范围。`JShell` 为上述每个命令内置的帮助文档包含更多信息。例如，可以执行 `/help /set mode` 查看详情。

自定义 `JShell` 启动

A.9 节列出了 `JShell` 在每次会话开始时导入的常用包的集合。我们可以使用 `JShell` 的 `/set start filename` 命令来提供一个包含 `Java` 代码片段和 `JShell` 命令的文件，当会话因为 `/reset` 或 `/reload` 命令而重启时，该文件将在当前会话中使用。也可以使用以下命令删除所有启动代码片段：

```
/set start -none
```

或者使用以下命令返回默认的启动代码片段：

```
/set start -default
```

在这三种情况下，指定的设置仅适用于当前会话，除非同时使用了 `-retain` 选项。例如，以下命令指示后续所有 JShell 会话都应加载指定的启动代码片段和命令文件：

```
/set start -retain filename
```

还可以使用以下命令为以后的会话恢复默认设置

```
/set start -retain -default
```

A.12 代码片段编辑快捷键

除了图 A.11 中的命令外，JShell 还支持许多用于编辑代码的键盘快捷键，可以使用它们快速跳转到行的开头或结尾，或在行中的单词之间跳转等。表 A.2 总结了一些可用的快捷键。

表 A.2 JShell 快捷键

快捷键	描述
Ctrl + a	光标移动到行首
Ctrl + e	光标移动到行尾
Alt + b (Windows) 或 Option + b (macOS)	光标向后移动一个单词
Alt + f (Windows) 或 Option + f (macOS)	光标向前移动一个单词
Ctrl + r	搜索上一个命令或代码片段，其中包含在按 Ctrl + r 之后输入的字符
Ctrl + t	反转光标左侧的两个字符
Ctrl + k	剪切从光标到行尾的所有内容
Ctrl + u	剪切从行首到光标位置（但不包括光标处的字符）的所有内容
Ctrl + w	删除从光标到前一个空格的文本
Alt + d (Windows) 或 Option + d (macOS)	删除从光标到后一个空格的文本

A.13 JShell 如何为交互式使用重新解释 Java

为了方便交互，JShell 在某些方面重新解释了标准 Java，如下所示：

- 不需要 `main` 方法；
- 独立（standalone）语句不需要分号；
- 变量不需要在类或方法中声明；
- 方法不需要在类的主体中声明；
- 语句不一定要在方法内部；
- 重新声明变量、方法或类型会删除先前的声明，并将其替换为新的声明，而 Java 编译器通常会报错；
- 不需要捕获异常，但如果需要测试异常处理，则还是可以这样做；
- JShell 忽略顶级访问修饰符（`public`、`private`、`protected`、`static` 和 `final`）——只允许 `abstract`（第 9 章）作为类修饰符；
- `synchronized` 关键字被忽略；
- 不允许 `package` 语句和 `module` 语句。

A.14 IDE 的 JShell 支持

许多 Java IDE 都提供了对 JShell 的支持，包括 IntelliJ IDEA、Eclipse IDE for Java Developers、NetBeans 和 Visual Studio Code。每个 IDE 都以自己的方式启动 JShell 和运行代码片段。有些 IDE 包括代码语法着色，并支持以不同的方式处理名称自动补全。

生成式 AI 练习

1 要求 AI 提供在你喜欢的 IDE 中使用 JShell 的分步教程。要求 AI 解释你的 IDE 中的 JShell 与命令行中的 JShell 之间的主要区别。

A.15 小结

在本附录中，我们使用 JShell——Java 的交互式 REPL——来进行探索、发现和实验。我们启动了一个 JShell 会话，并使用了各种类型的代码片段，包括语句、变量、表达式、方法和类，所有这些都无需声明一个包含 `main` 方法的类来执行。

我们列出了当前会话中的有效代码片段，并使用上下箭头键来召回并执行了先前的代码片段和命令。我们还学习了如何列出当前会话的变量、方法、类型和导入。我们清除了当前的 JShell 会话以移除所有现有的代码片段，并指出可以将代码片段保存到文件，并在以后重新加载。

我们演示了 JShell 的代码和命令自动补全功能，探索了一个类的成员，并在 JShell 中直接显示了文档。我们探索了 `Math` 类，演示了如何列出其静态成员，如何查看方法的参数和重载形式，以及如何查看方法的文档和公共字段的文档。我们还探索了 `String` 对象的方法。

我们声明了方法，并对一个尚未声明的方法进行了前向引用，然后在会话后期声明了该方法，接着我们学习了可以返回并执行第一个方法。我们还将一个方法声明替换为了一个新的方法——事实上，任何对变量、方法或类型的声明都可以被替换。

我们演示了 JShell 能捕获所有异常，并会简单地显示一个栈跟踪，然后便显示下一个 `jshell>` 提示符，以便您可以继续会话。我们从一个包中导入了一个已有的已编译类，然后在 JShell 会话中使用了该类。

接着，我们总结并演示了各种其他的 JShell 命令。我们配置了一个自定义的代码片段编辑器，查看了 JShell 的帮助文档，重新加载了一个会话，从会话中删除了代码片段，配置了反馈模式等等。我们还列出了一些额外的键盘快捷键，用于在 `jshell>` 提示符下编辑当前的代码片段。

最后，我们讨论了 JShell 如何为交互式使用而对标准 Java 进行重新诠释，并提到了 IDE 对 JShell 的支持。

自测题(附答案)

鼓励读者在阅读 A.2 节~A.3 节后，使用 JShell 完成自测题 A.1~A.43。我们提供了所有这些题目的答案，以帮助读者快速熟悉 JShell。

A.1 确认当使用 `System.out.println` 显示一个字符串字面值（如“生日快乐！”）时，不会显示引号（`""`）。用分号结束语句。

答案：

```
jshell> System.out.println("生日快乐!");
生日快乐!

jshell>
```

A.2 重复练习 A.1，但删除语句末尾的分号，以证明独立语句的分号在 JShell 中是可选的。

答案：

```
jshell> System.out.println("生日快乐!")
生日快乐!

jshell>
```

A.3 确认 JShell 不会执行//行尾注释。

答案:

```
jshell> // 注释不会执行

jshell>
```

A.4 确认用多行注释（以/*和*/分隔）包裹的可执行语句不会执行。

答案:

```
jshell> /* 开始多行注释
...>   System.out.println("欢迎学习 Java 编程!")
...>   结束多行注释 */

jshell>
```

A.5 在 JShell 中输入以下内容时会发生什么:

```
/* 不完整的多行注释
System.out.println("欢迎学习 Java 编程!")
/* 结束多行
注释 */
```

答案: 没有编译错误, 因为第二个/*被认为是第一个多行注释的一部分。

```
jshell> /* 不完整的多行注释
...> System.out.println("欢迎学习 Java 编程!")
...> /* 结束多行
...> 注释 */

jshell>
```

A.6 证明用空格缩进代码不会影响语句的执行。

答案:

```
jshell> System.out.println("A")
A

jshell>   System.out.println("A") // 缩进 3 个空格
A

jshell>     System.out.println("A") // 缩进 6 个空格
A

jshell>
```

A.7 在 JShell 中将以下每个变量声明为 int 类型, 哪些有效, 哪些无效?

- a) first
- b) first number
- c) first1
- d) 1first

答案: a)有效; b)无效 (不允许空格); c)有效; d)无效 (不能以数字开头)。

```
jshell> int first
first ==> 0
```

```
jshell> int first number
| 错误:
| 需要';'
| int first number;
|           ^
```

```
jshell> int first1
first1 ==> 0
```

```
jshell> int 1first
| 错误:
| 需要 '.class'
| int 1first
|       ^
| 错误:
| 不是语句
| int 1first
|       ^--^
| 错误:
| 意外的类型
|   需要: 值
|   找到: 类
| int 1first
| ^--^
```

```
jshell>
```

A.8 证明字符串面值中的大括号不必成对出现。

答案:

```
jshell> "字符串中不匹配的大括号{是允许的"
$1 ==> "字符串中不匹配的大括号{是允许的"
```

```
jshell>
```

A.9 在 JShell 中输入以下每个代码片段时会发生什么:

- a) System.out.println("一切正常")
- b) System.out.println("遗漏了什么吗?)
- c) System.out.println("又遗漏了什么吗? ")

答案:

```
jshell> System.out.println("一切正常")
一切正常

jshell> System.out.println("遗漏了什么吗? )
|  错误:
|  未结束的字符串文字
|  System.out.println("遗漏了什么吗? )
|                                     ^

jshell> System.out.println"又遗漏了什么吗? ")
|  错误:
|  需要';'
|  System.out.println"又遗漏了什么吗? ")
|                                     ^
|  错误:
|  找不到符号
|  符号:   变量 println
|  位置:  类型为 java.io.PrintStream 的变量 out
|  System.out.println"又遗漏了什么吗? ")
|  ^-----^

jshell>
```

A.10 在一个新会话中，演示在 `System.out.print` 之后，下一个 `print` 的结果出现在同一行中，紧跟在前一个结果之后。提示：为了证明这一点，请重置当前会话，输入两个 `System.out.print` 语句，使用以下两个命令将代码片段保存到一个文件中，然后重新加载并重新执行它们：

```
/save mysnipets
/open mysnipets
```

`/open` 命令加载 `mysnipets` 文件的内容，然后执行它们。

答案:

```
jshell> System.out.print("生日")
生日
jshell> System.out.print("快乐")
快乐
jshell> /save mysnipets

jshell> /open mysnipets
生日快乐
jshell>
```

A.11 证明在 `System.out.println` 之后，下一个 `println` 的结果在下一行的左侧显示。提示：为了演示这一点，请重置当前会话，输入这两个 `System.out.println` 语句，使用以

下两个命令将代码片段保存到一个文件中，然后重新加载并重新执行它们：

```
/save mysnipets1
/open mysnipets1
```

`/open` 命令加载 `mysnipets` 文件的内容，然后执行它们。

答案：

```
jshell> System.out.println("生日")
生日

jshell> System.out.println("快乐")
快乐

jshell> /save mysnipets1

jshell> /open mysnipets1
生日
快乐

jshell>
```

A.12 证明可以重置一个 JShell 会话，以删除之前的所有代码片段，并从头开始，而无需退出 JShell 并启动一个新的会话。

答案：

```
jshell> int x = 10
x ==> 10

jshell> int y = 20
y ==> 20

jshell> x + y
$3 ==> 30

jshell> /reset
| 正在重置状态。

jshell> /list

jshell>
```

A.13 使用 `System.out.println` 来证明转义序列 `\n` 导致向输出发送一个换行符。请使用字符串 `"欢迎\n学习\nJShell!"`。

答案：

```
jshell> System.out.println("欢迎\n学习\nJShell!")
```

```
欢迎
学习
JShell!
```

```
jshell>
```

A.14 证明转义序列 `\t` 导致向输出发送一个制表符。注意，具体的输出取决于当前系统如何设置制表符。请使用字符串“前\t后\n前\t\t后”。

答案：

```
jshell> System.out.println("前\t后\n前\t\t后")
前      后
前              后

jshell>
```

A.15 实验在一个字符串中包含单个反斜杠 (`\`) 的结果。请确保反斜杠后面的字符不会创建一个有效的转义序列。

答案：

```
jshell> System.out.println("不正确的转义\序列")
|  错误:
|  非法逃逸 符
|  System.out.println("不正确的转义\序列")
|                               ^

jshell>
```

A.16 显示一个包含 `\\` 的字符串（记住，`\\` 是反斜杠的转义序列）。显示了多少个反斜杠？

答案：两个。

```
jshell> System.out.println("显示反斜杠\\")
显示反斜杠\\

jshell>
```

A.17 使用转义序列 `\"` 来显示包含引号的字符串。

答案：

```
jshell> System.out.println("\"两端的引号是字符串的一部分\"")
"两端的引号是字符串的一部分"

jshell>
```

A.18 在 JShell 中执行以下代码时会发生什么？

```
System.out.println("生日快乐!\r 赵敏")
```

答案: \r 将输出光标移回至当前行的开头, 因此"赵敏"会覆盖"生日"。

```
jshell> System.out.println("生日快乐!\r 赵敏")
赵敏快乐!
```

```
jshell>
```

A.19 考虑以下语句:

```
System.out.printf("%s\n%s\n", "欢迎学习", "Java 编程!")
```

分别故意引入以下错误, 观察会发生什么。

- a) 省略参数列表周围的圆括号。
- b) 省略逗号。
- c) 省略其中一个%s\n 序列。
- d) 省略其中一个字符串 (即, 第二个或第三个参数)。
- e) 将第一个%s 替换为%d。
- f) 将字符串"欢迎学习"替换为整数 23。

答案:

a)

```
jshell> System.out.printf"%s\n%s\n", "欢迎学习", "Java 编程!"
| 错误:
| 需要';'
| System.out.printf"%s\n%s\n", "欢迎学习", "Java 编程!"
|                               ^
| 错误:
| 找不到符号
|   符号:   变量 printf
|   位置: 类型为 java.io.PrintStream 的变量 out
| System.out.printf"%s\n%s\n", "欢迎学习", "Java 编程!"
| ^-----^
```

```
jshell>
```

b)

```
jshell> System.out.printf("%s\n%s\n" "欢迎学习" "Java 编程!")
| 错误:
| 需要')'或','
| System.out.printf("%s\n%s\n" "欢迎学习" "Java 编程!")
|                               ^
```

```
jshell>
```

c)

```
jshell> System.out.printf("%s%n", "欢迎学习", "Java 编程!")
欢迎学习
$2 ==> java.io.PrintStream@69d0a921
```

```
jshell>
```

d)

```
jshell> System.out.printf("%s%n%s%n", "欢迎学习")
欢迎学习
| 异常错误 java.util.MissingFormatArgumentException: Format specifier '%s'
|     at Formatter.format (Formatter.java:2760)
|     at PrintStream.format (PrintStream.java:1183)
|     at PrintStream.printf (PrintStream.java:1081)
|     at (#3:1)
```

```
jshell>
```

e)

```
jshell> System.out.printf("%d%n%s%n", "欢迎学习", "Java 编程!")
| 异常错误 java.util.IllegalFormatConversionException: d != java.lang.String
|     at Formatter$FormatSpecifier.failConversion (Formatter.java:4662)
|     at Formatter$FormatSpecifier.printInteger (Formatter.java:3200)
|     at Formatter$FormatSpecifier.print (Formatter.java:3155)
|     at Formatter.format (Formatter.java:2761)
|     at PrintStream.format (PrintStream.java:1183)
|     at PrintStream.printf (PrintStream.java:1081)
|     at (#4:1)
```

```
jshell>
```

f)

```
jshell> System.out.printf("%s%n%s%n", 23, "Java 编程!")
23
Java 编程!
$5 ==> java.io.PrintStream@69d0a921
```

```
jshell>
```

A.20 在一个新的 JShell 会话中输入 `/imports` 命令时会显示什么?

答案:

```
jshell> /imports
| import java.io.*
| import java.math.*
| import java.net.*
| import java.nio.file.*
| import java.util.*
```

```
| import java.util.concurrent.*
| import java.util.function.*
| import java.util.prefs.*
| import java.util.regex.*
| import java.util.stream.*
```

```
jshell>
```

A.21 首先导入 `Scanner` 类，然后创建一个用于从 `System.in` 读取的 `Scanner` 对象 `input`。最后执行下语句，并由用户输入字符串 `"hello"`，会发生什么？

```
int number = input.nextInt()
```

答案：

```
jshell> import java.util.Scanner
```

```
jshell> var input = new Scanner(System.in)
input ==> java.util.Scanner[delimiters=\p{javaWhitespace}+] ...
\E][infinity string=\Q∞\E]
```

```
jshell> int number = input.nextInt()
```

```
hello
```

```
| 异常错误 java.util.InputMismatchException
|       at Scanner.throwFor (Scanner.java:964)
|       at Scanner.next (Scanner.java:1619)
|       at Scanner.nextInt (Scanner.java:2284)
|       at Scanner.nextInt (Scanner.java:2238)
|       at (#3:1)
```

```
jshell>
```

A.22 在一个新的或 `/reset` 的 JShell 会话中，重做题 A.21，但不导入 `Scanner` 类，以证明 `java.util` 包已经在 JShell 中导入。

答案：（略）

A.23 实验若不在 `Scanner` 输入操作之前显示一条有意义的提示消息，告诉用户要输入什么，那么会有什么结果。输入以下语句：

```
var input = new Scanner(System.in)
int value = input.nextInt()
```

答案：JShell 似乎挂起，因为它静悄悄地等待用户输入一个值并按下 `Enter` 键。

```
jshell> var input = new Scanner(System.in)
input ==> java.util.Scanner[delimiters=\p{javaWhitespace}+] ...
\E][infinity string=\Q∞\E]
```

```
jshell> int value = input.nextInt()
```

A.24 证明不能将 `import` 语句放到一个类中。

答案:

```
jshell> class Demonstration {
...>   import java.util.Scanner;
...> }
| 错误:
| 非法的类型开始
|   import java.util.Scanner;
|   ^
| 错误:
| 需要<标识符>
|   import java.util.Scanner;
|   ^

jshell> import java.util.Scanner

jshell> class Demonstration {
...> }
| 已创建 类 Demonstration

jshell>
```

A.25 在一个新的会话中，分别声明 `String` 和 `int` 类型的变量 `id` 和 `ID` 来证明标识符是区分大小写的。使用 `/list` 命令来输出表示单独变量的两个代码片段。

答案:

```
jshell> String id = "周芷若"
id ==> "周芷若"

jshell> int ID = 413
ID ==> 413

jshell> /list

1 : String id = "周芷若";
2 : int ID = 413;

jshell>
```

A.26 证明对于以下初始化语句来说:

```
String month = "四月"
int age = 26
```

它们确实会使用指定的值来初始化变量。

答案:

```
jshell> String month = "四月"
month ==> "四月"
```

```
jshell> month
month ==> "四月"
```

```
jshell> int age = 26
age ==> 26
```

```
jshell> age
age ==> 26
```

```
jshell> /vars
|   String month = "四月"
|   int age = 26
```

```
jshell>
```

A.27 实验以下操作：

- a) 在最大 int 值 2147483647 上加 1;
- b) 从最小 int 值 -2147483648 中减 1。

答案：

```
jshell> 2147483647 + 1
$1 ==> -2147483648
```

```
jshell> -2147483648 - 1
$2 ==> 2147483647
```

```
jshell>
```

A.28 证明像 1234567890 这样的大整数等同于带有下划线分隔符的对应整数，即 1_234_567_890。

- a) 1234567890 == 1_234_567_890
- b) 打印每个值以确认得到相同的结果
- c) 将每个值除以 2 以确认得到相同的结果

答案：

```
jshell> 1234567890 == 1_234_567_890
$1 ==> true
```

```
jshell> System.out.println(1234567890)
1234567890
```

```
jshell> System.out.println(1_234_567_890)
1234567890
```

```
jshell> 1234567890 / 2
$4 ==> 617283945
```

```
jshell> 1_234_567_890 / 2
$5 ==> 617283945
```

```
jshell>
```

A.29 证明在算术表达式中的操作符周围添加空格不会影响该表达式的值。例如，以下表达式是等价的：

```
17+23
17 + 23
```

用以下条件来证明这一点

```
(17+23) == (17 + 23)
```

答案：

```
jshell> (17+23) == (17 + 23)
$1 ==> true
```

```
jshell>
```

A.30 证明以下语句中 `number1 + number2` 周围的圆括号是不必要的：

```
System.out.printf("总和为%d\n", (number1 + number2))
```

答案：

```
jshell> int number1 = 10
number1 ==> 10
```

```
jshell> int number2 = 20
number2 ==> 20
```

```
jshell> System.out.printf("总和为%d\n", (number1 + number2))
总和为 30
$3 ==> java.io.PrintStream@5387f9e0
```

```
jshell> System.out.printf("总和为%d\n", number1 + number2)
总和为 30
$4 ==> java.io.PrintStream@5387f9e0
```

```
jshell>
```

A.31 声明 `int` 变量 `x` 并将其初始化为 `14`，然后证明后续赋值 `x = 27` 会覆盖 `x` 先前的值。

答案：

```
jshell> int x = 14
x ==> 14
```

```
jshell> x = 27
x ==> 27
```

```
jshell>
```

A.32 证明打印以下变量的值不会修改其值：

```
int y = 29
```

答案：

```
jshell> int y = 29
y ==> 29
```

```
jshell> System.out.println(y)
29
```

```
jshell> y
y ==> 29
```

```
jshell>
```

A.33 基于以下变量声明：

```
int b = 7
int m = 9
```

a) 证明在 Java 中，不能像代数中那样连续写变量名（如 `bm`）来执行乘法。

b) 证明在 Java 表达式中，要写表达式 `b * m` 来将两个操作数相乘。

答案：

```
jshell> int b = 7
b ==> 7
```

```
jshell> int m = 9
m ==> 9
```

```
jshell> bm
| 错误:
| 找不到符号
|   符号: 变量 bm
|   位置: 类
|   bm
|   ^^
```

```
jshell> b * m
```

\$3 ==> 63

jshell>

A.34 基于以下表达式来证明整数除法会产生整数结果:

a) $8 / 4$

b) $7 / 5$

答案: a) 2; b) 1

jshell> $8 / 4$

\$1 ==> 2

jshell> $7 / 5$

\$2 ==> 1

jshell>

A.35 执行以下每个整数除法时会发生什么 (参见 10.2 节)?

a) $0 / 1$

b) $1 / 0$

c) $0 / 0$

d) $1.0 / 0$

e) $0.0 / 0.0$

答案:

jshell> $0 / 1$

\$1 ==> 0

jshell> $1 / 0$

```
| 异常错误 java.lang.ArithmeticException: / by zero  
|         at (#2:1)
```

jshell> $0 / 0$

```
| 异常错误 java.lang.ArithmeticException: / by zero  
|         at (#3:1)
```

jshell> $1.0 / 0$

\$4 ==> Infinity

jshell> $0.0 / 0.0$

\$5 ==> NaN

jshell>

A.36 证明以下表达式具有不同的求值结果:

a) $(3 + 4 + 5) / 5$

b) $3 + 4 + 5 / 5$

因此，如果想要先计算 $3 + 4 + 5$ ，再将结果除以 5，则第一个表达式中的圆括号是必需的。

答案：

```
jshell> (3 + 4 + 5) / 5
$1 ==> 2
```

```
jshell> 3 + 4 + 5 / 5
$2 ==> 8
```

```
jshell>
```

A.37 手动计算以下表达式的值（注意遵守操作符优先级规则），在 JShell 中验算。

$5 / 2 * 2 + 4 \% 3 + 9 - 3$

答案：

```
jshell> 5 / 2 * 2 + 4 % 3 + 9 - 3
$1 ==> 11
```

```
jshell>
```

A.38 对两个相同的值 7 测试两个相等性和四个关系操作符。例如， $7 == 7$ 、 $7 < 7$ 等。

答案：

```
jshell> 7 == 7
$1 ==> true
```

```
jshell> 7 != 7
$2 ==> false
```

```
jshell> 7 < 7
$3 ==> false
```

```
jshell> 7 <= 7
$4 ==> true
```

```
jshell> 7 > 7
$5 ==> false
```

```
jshell> 7 >= 7
$6 ==> true
```

```
jshell>
```

A.39 使用值 7 和 9 重做题 A.38。

答案:

```
jshell> 7 == 9
$1 ==> false
```

```
jshell> 7 != 9
$2 ==> true
```

```
jshell> 7 < 9
$3 ==> true
```

```
jshell> 7 <= 9
$4 ==> true
```

```
jshell> 7 > 9
$5 ==> false
```

```
jshell> 7 >= 9
$6 ==> false
```

```
jshell>
```

A.40 使用值 11 和 9 重做题 A.38。

答案:

```
jshell> 11 == 9
$1 ==> false
```

```
jshell> 11 != 9
$2 ==> true
```

```
jshell> 11 < 9
$3 ==> false
```

```
jshell> 11 <= 9
$4 ==> false
```

```
jshell> 11 > 9
$5 ==> true
```

```
jshell> 11 >= 9
$6 ==> true
```

```
jshell>
```

A.41 证明在 `if` 语句中，若不小心在条件的右圆括号后添加了一个分号，那么可能导致逻辑错误。

```
if (3 == 5); {
    System.out.println("3 等于 5");
}
```

答案:

```
jshell> if (3 == 5); {
...>     System.out.println("3 等于 5");
...> }
3 等于 5

jshell>
```

A.42 给定以下声明:

```
int x = 1
int y = 2
int z = 3
int a
```

在执行了以下语句后, a、x、y 和 z 的值分别是多少?

```
a = x = y = z = 10
```

答案:

```
jshell> int x = 1
x ==> 1
```

```
jshell> int y = 2
y ==> 2
```

```
jshell> int z = 3
z ==> 3
```

```
jshell> int a
a ==> 0
```

```
jshell> a = x = y = z = 10
a ==> 10
```

```
jshell> /vars
|   int x = 10
|   int y = 10
|   int z = 10
|   int a = 10
```

```
jshell>
```

A.43 手动计算以下表达式的值, 在 JShell 中验算。

$(3 * 9 * (3 + (9 * 3 / (3))))$

答案:

```
jshell> (3 * 9 * (3 + (9 * 3 / (3))))  
$1 ==> 324
```

```
jshell>
```

附录 B 格式化输出

学习目标：

- 使用 `printf` 方法格式化输出；
- 格式化十进制、八进制和十六进制整数；
- 使用定点记数法和科学记数法格式化浮点数；
- 格式化字符串和字符；
- 使用字段宽度和精度进行格式化；
- 使用各种 `printf` 格式化标志来对齐数据、显示整数的进制、用 0 填充字段、以特定于区域设置的千位分隔符显示数值，以及在圆括号中显示负值；
- 使用参数索引来指定每个参数在格式字符串中的位置；
- 在内存中格式化字符串，而不是直接显示格式化输出。

B.1 导论

本附录将介绍 `printf` 方法和 `Formatter` 类（来自 `java.util` 包）的格式化特性。`Formatter` 类默认将数据格式化为内存中的字符串（`String`），但也可以将格式化输出发送到指定目标，例如文件。`printf` 方法使用 `Formatter` 对象将格式化输出发送到标准输出。本书已经讨论了许多 `printf` 特性。在本附录中，我们将总结这些特性并介绍一些新特性，例如使用参数索引在格式字符串中定位参数，以及使用各种标志（`flag`）来显示数字和字符串。我们已经在 5.12 节“对象自然”案例学习中简要讨论了日期/时间格式化，并将在生成式 AI 练习 B.2 中进一步探索。

B.2 使用 `printf` 进行输出

使用 `printf` 方法可以实现精确的输出格式化。Java 从 C 语言中借鉴并增强了此特性。`printf` 方法可以执行以下格式化功能，本附录将对每一项进行讨论：

- 将浮点值四舍五入到指定的小数位数；
- 对齐一系列数字，使小数点垂直对齐；
- 输出右对齐和左对齐；
- 在输出行的精确位置插入字符面值；
- 以指数格式表示浮点数；
- 以八进制和十六进制格式表示整数；
- 使用固定大小的字段宽度和精度来显示所有类型的数据。

虽然也可以用 `printf` 方法来日期和时间，但应首选 `DateTimeFormatter` 类（参见 5.12 节）。

每次调用 `printf` 时，都要提供一个格式字符串作为其第一个参数，该字符串描述了输出格式。格式字符串可以包含**固定文本**（fixed text）和**格式说明符**（format specifier）。固定文本会由 `printf` 直接输出，就像 `System.out` 的 `print` 或 `println` 方法一样。每个格式说明符都是一个值的占位符，并指定了要输出的数据类型。格式说明符还可以包含可选的格式化信息。

在其最简单的形式中，每个格式说明符都以一个百分号（%）开头，后跟一个代表待输出值的数据类型的**转换字符**（conversion character）。例如，格式说明符 `%s` 是 `String` 的占位符，而格式说明符 `%d` 是 `int` 值的占位符。可选的格式化信息（例如参数索引、标志、字段宽度和精度）则在百分号和转换字符之间指定。我们稍后将逐一演示。

B.3 整数格式化

表 B.1 总结了**整数转换字符**——附录 C 将介绍二进制、八进制、十进制和十六进制数制系统。

表 B.1 整数转换字符

转换字符	描述
<code>d</code>	显示十进制（基数 10）整数
<code>o</code>	显示八进制（基数 8）整数
<code>x</code> 或 <code>X</code>	显示十六进制（基数 16）整数， <code>x</code> 表示在输出中使用大写字母

图 B.1 的程序使用每个转换字符来打印一个整数。默认情况下不显示加号，但会显示减号。本附录后面（图 B.8）会解释如何强制显示加号。

```
1 // 图 B.1: IntegerConversionTest.java
2 // 使用整数转换字符
3
4 public class IntegerConversionTest {
5     public static void main(String[] args) {
6         System.out.printf("%d\n", 26);
7         System.out.printf("%d\n", +26);
8         System.out.printf("%d\n", -26);
9         System.out.printf("%o\n", 26);
10        System.out.printf("%x\n", 26);
11        System.out.printf("%X\n", 26);
12    }
```

```
13 }
```

```
26
26
-26
32
1a
1A
```

图 B.1 使用整数转换字符

`printf` 方法具有以下形式：

```
printf(format-string, argument-list);
```

其中，*format-string*（格式字符串）描述了输出格式，而可选的 *argument-list*（参数列表）包含与格式字符串中的每个格式说明符对应的值。一个格式字符串中可以有多个格式说明符。

第 6 行～第 8 行中的每个格式字符串都指定 `printf` 应首先输出一个十进制整数（`%d`），然后输出一个换行符（`%n`）。在格式说明符的位置，`printf` 将用格式字符串后的第一个参数的值来替换。如果格式字符串包含多个格式说明符，则在后续每个格式说明符的位置，`printf` 都用参数列表中的下一个参数的值来替换。第 9 行的 `%o` 格式说明符指定以八进制格式输出整数。第 10 行的 `%x` 格式说明符指定以带有小写字母的十六进制格式输出整数。第 11 行的 `%X` 格式说明符则指定以带有大写字母的十六进制格式输出整数。

B.4 浮点数格式化

表 B.2 总结了浮点数转换字符。

表 B.2 浮点数转换字符

转换字符	描述
e 或 E	以指数记数法显示浮点值。E 表示在输出中使用大写字母
f	以定点格式显示浮点值
g 或 G	基于值的大小，以定点格式 <code>f</code> 或指数格式 <code>e</code> 显示浮点值。如果浮点值小于 10^{-3} 或大于等于 10^7 ，则以 <code>e</code> （或 <code>E</code> ）打印。否则，该值以格式 <code>f</code> 打印。G 表示在输出中使用大写字母

a 或 A	以十六进制格式显示浮点数。A 表示在输出中使用大写字母
-------	-----------------------------

转换字符 e 和 E 以**计算机科学计数法**（computerized scientific notation，也称为**指数记数法**）显示浮点值。指数记数法是数学中使用的科学计数法的计算机等效形式。例如，值 150.4582 在数学中以科学计数法表示为 1.504582×10^2 ，在 Java 中则以指数记数法表示为 1.504582e+02，意思是 1.504582 乘以 10 的 2 次方（e+02）。e 是 exponent 的缩写，即“指数”。

默认情况下，使用转换字符 e、E 和 f 打印的值在小数点右侧输出 6 位精度（例如，1.045921）。其他精度必须显式指定。对于使用转换字符 g 打印的值，精度代表着除指数之外显示的总位数。默认为 6 位数字（例如，12345678.9 显示为 1.23457e+07）。转换字符 f 始终在小数点左侧打印至少一位数字。转换字符 e 和 E 在指数前打印小写 e 和 大写 E，并且始终在小数点左侧打印一位数字。如果正在格式化的值具有比精度更多的有效数字，则会发生舍入。

转换字符 g（或 G）根据浮点值的大小以 e（E）或 f 格式打印。例如，值 0.0000875、87500000.0、8.75、87.50 和 875.0 使用转换字符 g 打印为 8.750000e-05、8.750000e+07、8.750000、87.500000 和 875.000000。值 0.0000875 使用 e 记数法，因为该值小于 10^{-3} 。值 87500000.0 也使用 e 记数法，因为该值大于 10^7 。图 B.2 的程序演示了浮点转换字符的用法。

```
1 // 图 B.2: FloatingNumberTest.java
2 // 使用浮点转换字符
3
4 public class FloatingNumberTest {
5     public static void main(String[] args) {
6         System.out.printf("%e\n", 12345678.9);
7         System.out.printf("%e\n", +12345678.9 );
8         System.out.printf("%e\n", -12345678.9);
9         System.out.printf("%E\n", 12345678.9);
10        System.out.printf("%f\n", 12345678.9);
11        System.out.printf("%g\n", 12345678.9);
12        System.out.printf("%G\n", 12345678.9);
13    }
14 }
```

```
1.234568e+07
1.234568e+07
-1.234568e+07
1.234568E+07
12345678.900000
1.23457e+07
1.23457E+07
```

图 B.2 使用浮点转换字符

B.5 字符串和字符格式化

转换字符 `c` 和 `s` 分别用于打印单个字符和字符串。转换字符 `c` 和 `C` 需要一个 `char` 类型的实参。转换字符 `s` 和 `S` 可以接收一个 `String` 或任何 `Object` 作为实参，因为任何对象都可以通过其 `toString` 方法转换为 `String`。当使用转换字符 `C` 和 `S` 时，输出会以大写字母显示。图 B.3 的程序使用转换字符 `c`、`s` 和 `S` 来显示字符、字符串和对象。第 7 行中，当一个 `int` 常量被赋给一个 `Integer` 对象时，会发生自动装箱。第 13 行使用转换字符 `s` 输出一个 `Integer` 实参，这会隐式调用其 `toString` 方法来获取整数值。也可以使用 `%d` 格式说明符来输出一个 `Integer` 对象，在这种情况下，`Integer` 对象中的 `int` 值将被自动拆箱并输出。使用 `%c` 来打印字符串会导致 `IllegalFormatConversionException`，因为 `String` 无法被转换为单个字符。

```
1 // 图 B.3: CharStringConversion.java
2 // 使用字符和字符串转换字符
3 public class CharStringConversion {
4     public static void main(String[] args) {
5         char character = 'A'; // 初始化 char 变量
6         String string = "This is also a string"; // 初始化 String 变量
7         Integer integer = 1234; // 初始化 Integer 变量发生自动装箱
8
9         System.out.printf("%c\n", character);
10        System.out.printf("%s\n", "This is a string");
11        System.out.printf("%s\n", string);
12        System.out.printf("%S\n", string);
13        System.out.printf("%s\n", integer); // 隐式调用 toString
14    }
15 }
```

```
A
This is a string
This is also a string
THIS IS ALSO A STRING
1234
```

图 B.3 使用字符和字符串转换字符

B.6 其他转换字符

表 B.3 总结了其他转换字符——`b`、`B`、`h`、`H`、`%`和 `n`。

表 B.3 其他转换字符

转换字符	描述
b 或 B	对于 <code>boolean</code> 或 <code>Boolean</code> 类型的值，打印 <code>"true"</code> 或 <code>"false"</code> 。这些转换字符也可以格式化任何引用的值：如果引用非空，那么输出 <code>"true"</code> ；否则输出 <code>"false"</code> 。B 表示在输出中使用大写字母
h 或 H	以十六进制格式打印对象哈希码的字符串表示形式。如果对应的实参为 <code>null</code> ，那么打印 <code>"null"</code> 。H 表示在输出中使用大写字母
%	打印百分号字符
n	打印特定于平台的换行符（例如，在 Windows 上是 <code>\r\n</code> ，在 macOS/Linux 上是 <code>\n</code> ）

在图 B.4 的程序中，第 7 行~第 8 行使用 `%b` 来打印 `boolean`（或 `Boolean`）值 `false` 和 `true`。第 9 行将一个字符串与 `%b` 关联，由于该字符串非空，因此输出 `true`。第 10 行将一个 `null` 对象与 `%B` 关联，由于 `test` 为 `null`，因此输出 `FALSE`。第 11 行~第 12 行使用 `%h` 来打印字符串 `"hello"` 和 `"Hello"` 的哈希码的字符串表示形式。这些值可在 `HashMap`（参见第 12 章）中用于存储或定位字符串。由于是两个不同的字符串（一个以小写字母开头，另一个以大写字母开头），因此它们具有不同的哈希码。第 13 行使用 `%H` 以大写字母打印 `null`。最后两个 `printf` 语句（第 14 行~第 15 行）使用 `%%` 在字符串中打印 `%` 字符，并使用 `%n` 打印一个特定于平台的换行符（行分隔符）。

```

1 // 图 B.4: OtherConversion.java
2 // 使用 b、B、h、H、%和 n 转换字符
3
4 public class OtherConversion {
5     public static void main(String[] args) {
6         Object test = null;
7         System.out.printf("%b\n", false);
8         System.out.printf("%b\n", true);
9         System.out.printf("%b\n", "测试");
10        System.out.printf("%B\n", test);
11        System.out.printf("\hello\的哈希码是%h\n", "hello");
12        System.out.printf("\Hello\的哈希码是%h\n", "Hello");
13        System.out.printf("null 的哈希码是%H\n", test);
14        System.out.printf("在格式字符串中打印一个%%\n");
15        System.out.printf("打印一个换行符\n 下一行从这里开始");
16    }
17 }

```

```
false
```

```
true
true
FALSE
"hello"的哈希码是 5e918d2
"Hello"的哈希码是 42628b2
null 的哈希码是 NULL
在格式字符串中打印一个%
打印一个换行符
下一行从这里开始
```

图 B.4 使用 b、B、h、H、%和 n 转换字符

注意，在格式字符串中使用%而不是%%打印字面百分号字符，可能会导致难以发现的逻辑错误。当%出现在格式字符串中时，它后面必须跟有字符串中的转换字符。单个百分号可能会意外地后跟一个合法的转换字符，从而导致逻辑错误。

B.7 字段宽度与精度

打印数据字段的大小由字段宽度指定。^①如果字段宽度大于要打印的数据，则默认情况下，数据在该字段中右对齐。我们将在 B.8 节中讨论左对齐。可以在格式说明符中的%和转换字符之间插入一个表示字段宽度的整数（例如，%4d）。图 B.5 打印了两组各 5 个数字，并将那些位数少于字段宽度的数字进行右对齐。如果值的宽度超过了字段宽度，那么字段宽度会自动增加以适应该值。负号在字段中会占用一个字符位置。此外，如果未指定字段宽度，那么数据将按其实际需要的位数打印。字段宽度可用于除行分隔符（%n）外的所有格式说明符。若未提供足够大的字段宽度来处理待打印的值，可能会挤占其他正在打印的数据，并产生令人困惑的输出。一定要了解自己的数据！

```
1 // 图 B.5: FieldWidthTest.java
2 // 在字段中右对齐整数
3
4 public class FieldWidthTest {
5     public static void main(String[] args) {
6         System.out.printf("%4d\n", 1);
7         System.out.printf("%4d\n", 12);
8         System.out.printf("%4d\n", 123);
9         System.out.printf("%4d\n", 1234);
10        System.out.printf("%4d\n\n", 12345); // 太大
11
12        System.out.printf("%4d\n", -1);
```

^① 译注：也可以将 field 理解为“域”，将 field width 理解为“域宽”。

```

13     System.out.printf("%4d%n", -12);
14     System.out.printf("%4d%n", -123);
15     System.out.printf("%4d%n", -1234); // 太大
16     System.out.printf("%4d%n", -12345); // 太大
17 }
18 }

```

```

1
12
123
1234
12345

-1
-12
-123
-1234
-12345

```

图 B.5 在字段中右对齐整数

`printf` 方法还提供了指定数据打印精度的功能。精度对于不同类型有不同的含义。当与浮点转换字符 `e` 和 `f` 一起使用时，精度是小数位数。当与转换字符 `g`、`a` 或 `A` 一起使用时，精度是要打印的最大有效数字个数。当与转换字符 `s` 一起使用时，精度是要从字符串中写入的最大字符数。要使用精度，请在百分号和转换说明符之间放置一个小数点 (`.`)，后跟一个代表精度的整数。图 B.6 演示了格式字符串中的精度。当打印的浮点值所指定的精度小于其原始值的小数位数时，该值会被四舍五入。此外，格式说明符 `%.3g` 表示用于显示该浮点值的总位数为 3。因为该值小数点左边有三位数字，所以它被四舍五入到了个位。

```

1 // 图 B.6: PrecisionTest.java
2 // 为浮点数和字符串指定精度
3 public class PrecisionTest {
4     public static void main(String[] args) {
5         double f = 123.94536;
6         String s = "一二三四五六七八九十一二三四五";
7
8         System.out.printf("使用浮点数精度%n");
9         System.out.printf("\t%.3f%n\t%.3e%n\t%.3g%n", f, f, f);
10
11        System.out.printf("使用字符串精度%n");
12        System.out.printf("\t%.11s%n", s);
13    }
14 }

```

```

使用浮点数精度
123.945

```

<pre> 1.239e+02 124 使用字符串精度 一二三四五六七八九十一 </pre>

图 B.6 为浮点数和字符串指定精度

字段宽度和精度可以组合使用。为此，需要在百分号和转换字符之间先指定字段宽度，后跟一个小数点，再指定精度，如以下语句所示：

```
printf("%9.3f", 123.456789);
```

该语句会显示 123.457，小数点右侧保留三位数字，并在一个 9 位字段中右对齐，该数字前会留出两个空格。

B.8 格式化标志

有多个**标志**（flag）可与 printf 方法一起使用，以补充其输出格式化功能。表 B.4 总结了可以在格式字符串中使用的标志。

表 B.4 格式化标志

标志	描述
-（减号）	在指定的字段内将输出左对齐
+（加号）	在正值前显示加号，在负值前显示减号
空格	在未使用+标志打印的正值前打印一个空格
#	与八进制转换字符 o 一起使用时，为输出值附加前缀 0；与十六进制转换字符 x 一起使用时，为输出值附加前缀 0x
0（零）	用前导零来填充字段
,（逗号）	使用特定于区域设置的千位分隔符（例如，美国使用 ','）来显示十进制数和浮点数
(将负数放到一对圆括号中

很有用。由于负号，值-547 在输出中前面没有空格。

```
1 // 图 B.9: SpaceFlagTest.java
2 // 在非负数加打印一个空格
3
4 public class SpaceFlagTest {
5     public static void main(String[] args) {
6         System.out.printf("% d%n% d%n", 547, -547);
7     }
8 }
```

```
547
-547
```

图 B.9 在非负数之前打印空格

图 B.10 使用#标志为八进制值附加前缀 0，为十六进制值附加前缀 0x。

```
// 图 B.10: PoundFlagTest.java
// 将#标志与转换字符 o 和 x 一起使用

public class PoundFlagTest {
    public static void main(String[] args) {
        int c = 31;    // 初始化 c

        System.out.printf("#o%n", c);
        System.out.printf("#x%n", c);
    }
}
```

```
037
0x1f
```

图 B.10 将#标志与转换字符 o 和 x 一起使用

图 B.11 结合使用+、0 和空格标志，首先在宽度为 9 的字段中以+号和前导零打印 452，然后仅使用 0 标志在宽度为 9 的字段中打印 452，最后仅使用空格标志在宽度为 9 的字段中打印 452。

```
1 // 图 B.11: ZeroFlagTest.java
2 // 使用 0（零）标志打印来填充前导零
3
4 public class ZeroFlagTest {
5     public static void main(String[] args) {
6         System.out.printf("%+09d%n", 452);
7         System.out.printf("%09d%n", 452);
```

```

8      System.out.printf("% 9d\n", 452);
9    }
10 }

```

```

+00000452
00000452
  452

```

图 B.11 使用 0（零）标志打印来填充前导零

图 B.12 使用逗号（,）标志显示带有千位分隔符的十进制数和浮点数。

```

// 图 B.12: CommaFlagTest.java
// 使用逗号（,）标志显示带有千位分隔符的数字

public class CommaFlagTest {
    public static void main(String[] args) {
        System.out.printf("%,d\n", 58625);
        System.out.printf("%,.2f\n", 58625.21);
        System.out.printf("%,.2f\n", 12345678.9);
    }
}

```

```

58,625
58,625.21
12,345,678.90

```

图 B.12 使用逗号（,）标志显示带有千位分隔符的数字

图 B.13 使用(标志将负数放到一对圆括号。50 由于是正数，所以未放到圆括号中。

```

1 // 图 B.13: ParenthesesFlagTest.java
2 // 使用(标志将负数放到圆括号中
3
4 public class ParenthesesFlagTest {
5     public static void main(String[] args) {
6         System.out.printf("(d\n", 50);
7         System.out.printf("(d\n", -50);
8         System.out.printf("(.1e\n", -50.0);
9     }
10 }

```

```

50
(50)
(5.0e+01)

```

图 B.13 使用(标志将负数放到圆括号中

B.9 格式字符串中用于显式定位的参数索引

参数索引 (argument index) 是后跟美元符号 (\$) 的一个可选的整数, 用于指示在参数列表中的位置。参数索引使程序员能够指定每个参数的确切位置。对于支持多语言的软件, 重新排序参数对于翻译后的格式字符串很有用, 因为口语通常以不同的方式对单词进行排序。参数索引使我们使用不同的格式字符串, 而无需更改 printf 的其他参数的顺序。参数索引还有助于避免重复提供参数值。图 B.14 使用参数索引按相反的顺序打印参数值。

```
1 // 图 B.14: ArgumentIndexTest.java
2 // 使用参数索引重新排序输出
3
4 public class ArgumentIndexTest {
5     public static void main(String[] args) {
6         System.out.printf(
7             "未重新排序的参数列表: %s %s %s %s\n",
8             "第一", "第二", "第三", "第四");
9         System.out.printf(
10            "重新排序后的参数列表: %4$s %3$s %2$s %1$s\n",
11            "第一", "第二", "第三", "第四");
12     }
13 }
```

未重新排序的参数列表: 第一 第二 第三 第四
重新排序后的参数列表: 第四 第三 第二 第一

图 B.14 使用参数索引重新排序输出

B.10 转义序列

我们使用 printf 输出的大多数字符面值都可以直接包含在格式字符串中。然而, 存在一些“问题”字符, 例如用于界定格式字符串本身的引号 (")。各种控制字符, 如换行符和制表符, 都必须用转义序列来表示。表 B.5 总结了这些转义序列。

表 B.5 转义序列

转义序列	描述
\' (单引号)	输出单引号 (') 字符
\" (双引号)	输出双引号 (") 字符
\\ (反斜杠)	输出反斜杠 (\) 字符

<code>\b</code> （退格）	将光标在当前行上回退一个位置
<code>\f</code> （换页）	将光标移动到下一页的开头
<code>\n</code> （换行）	将光标移动到下一行的开头
<code>\r</code> （回车）	将光标移动到当前行的开头，后续输出将覆盖当前行的内容
<code>\t</code> （水平制表符）	将光标移动到下一个水平制表符的位置

B.11 在内存中格式化字符串

到目前为止，我们只是演示了如何显示格式化的输出。如果想创建格式化的字符串，而不是显示格式化的文本，该怎么做呢？`Formatter` 类（来自 `java.util` 包）允许在内存中格式化字符串，或将格式化的字符串发送到指定的目标（例如磁盘上的文件）。事实上，`printf` 方法正是使用了 `Formatter` 对象来提供其格式化输出功能。默认情况下，`Formatter` 会在内存中创建一个字符串。图 B.15 首先使用一个 `Formatter` 在内存中构建一个格式化的字符串，然后将其输出。

```

1 // 图 B.15: FormatterTest.java
2 // 使用 Formatter 类格式化输出
3 import java.util.Formatter;
4
5 public class FormatterTest {
6     public static void main(String[] args) {
7         // 创建一个 Formatter 对象并格式化输出
8         Formatter formatter = new Formatter();
9         formatter.format("%d = %#o = %#X", 10, 10, 10);
10
11         System.out.println(formatter.toString());
12     }
13 }

```

10 = 012 = 0XA

图 B.15 使用 `Formatter` 类格式化输出

第 8 行使用默认构造函数创建了一个 `Formatter` 对象，因此该对象将在内存中构建一个字符串。`Formatter` 还提供了其他构造函数，允许将格式化数据输出到其他目标位置。欲知详情，请访问以下网址：

<https://docs.oracle.com/en/java/javase/24/docs/api/java.base/java/util/Formatter.html>

第 9 行调用 `format` 方法来格式化输出。与 `printf` 类似，`format` 接收一个格式字符串和一个参数列表，但 `format` 会将格式化的输出发送到其构造函数所指定的目标位置，默认情况下是在内存中创建一个字符串。第 11 行调用该 `Formatter` 的 `toString` 方法以字符串形式获取格式化后的数据，然后我们将其显示出来。

String 的静态方法 `format` 方法和实例方法 `formatted`

`String` 类也提供了两个简化在内存中创建格式化字符串的方法。

- `String` 的静态方法 `format` 接收一个格式字符串和要格式化的值，并返回一个格式化后的 `String`。例如，以下语句创建了与第 9 行相同的格式化字符串：

```
String s = String.format("%d = %o = %#x", 10, 10, 10);
```

- 字符串字面值本身就是一个 `String` 对象，因此可以直接在它上面调用 `String` 的方法。`String` 类的实例方法 `formatted` 与 `format` 类似，但它是直接在格式字符串上调用的。只需将要格式化的值传递给它即可，例如：

```
String s = "%d = %o = %#x".formatted(10, 10, 10);
```

`String` 类的 `formatted` 方法也可以在文本块上调用。例如，以下语句使用一个文本块作为格式字符串：

```
String book = ""
    书名: "%s"
    版次: %d
    Copyright: %d"".formatted("Java How to Program", 12, 2025);
```

打印 `book` 这个格式化后的文本块将显示以下内容：

```
    书名: "Java How to Program"
    版次: 12
    Copyright: 2025
```

B.12 小结

本附录总结了如何使用各种格式字符和标志来显示格式化的输出，以及如何在内存中创建格式化的字符串。我们使用格式字符 `d`、`o`、`x` 和 `X` 来输出整数；使用格式字符 `e`、`E`、`f`、`g` 和 `G` 来输出浮点数。我们使用字段宽度和精度来格式化文本。我们通过将数字格式字符与 `+`、`-`、空格、`#`、`0`、逗号和 `()` 等标志相结合来格式化数值。我们使用参数索引来指定每个参数在格式字符串中的位置。最后，我们演示了如何在内存中格式化字符串和文本块，首先使用了 `Formatter` 类及其 `format` 方法，然后使用了更便捷的 `String` 类及其 `format` 和 `formatted` 方法。

专题：生成式 AI 练习

对于以下每一个问题，请尝试多个生成式 AI，以确定哪一个表现最佳。始终验证其输出结果。请记住，它们有时会犯错，甚至产生“幻觉”，杜撰出虚假的信息，因此可能生成不可靠的内容。

B.1（格式字符串的安全性）要求 AI 解释为什么 Java 程序永远不应将用户输入的字符串作为格式字符串提供给 `printf`、`Formatter` 的 `format` 方法或 `String` 类的 `format` 与 `formatted` 方法。要求其提供简单的代码示例，说明如何防止它们所描述的问题。

B.2（格式化日期和时间）要求 AI 创建一个基于代码的教程，展示 Java 最常用的 `DateTimeFormatter` 功能。

B.3（特定于区域设置的格式化）要求 AI 创建一个基于代码的教程，展示 Java 中特定于区域设置的格式化。

B.4（在圆括号中显示负值）要求 AI 解释为什么有时需要像图 B.13 中那样在圆括号中显示负值，而不是为其附加减号前缀。

附录 C 数制系统

学习目标：

- 学习数制系统的概念，例如基数、位值和符号值；
- 了解数制系统中的位置表示法；
- 使用以二进制、八进制和十六进制数制系统表示的数字；
- 将二进制数缩写为八进制数，或更简洁地缩写为十六进制数；
- 将八进制数和十六进制数转换为二进制数；
- 在十进制数及其二进制、八进制和十六进制的等价值之间进行相互转换；
- 学习二进制算术以及如何使用补码表示法来表示负的二进制数；
- 了解反码和补码表示法之间的关系；
- 写二进制、八进制和十六进制数的 Java 数字字面值。

C.1 导论

本附录将介绍程序员所使用的数制系统。若软件项目需要与机器级硬件密切交互，那么这就是一个相当重要的主题。这种项目包括操作系统、计算机网络软件、编译器、数据库系统以及要求高性能的应用程序。

当我们在 Java 程序中写一个整数（例如 227 或 -63）时，该数字被假定为十进制（基数 10）数制系统。十进制数制系统中的数字是 0、1、2、3、4、5、6、7、8 和 9。最低位的数字是 0，最高位是 9，比基数 10 小 1。在内部，计算机使用二进制（基数 2）数制系统，它只有数字 0 和 1。其最低位的数字是 0，最高位是 1，比基数 2 小 1。

二进制数使用起来很麻烦，因为它们通常比其等价的十进制数长得多。因此，另外两个数制系统——八进制（基数 8）数制系统和十六进制（基数 16）数制系统——之所以流行，主要是因为它们可以方便地缩写二进制数。十六进制数也常用于指定颜色代码（例如网页安全色^①）、在图形学中使用以及用于缩写内存地址。在 7.9 节中，我们在讨论私钥密码学时使用了 Base64 编码。^②

在八进制数制系统中，数字的范围是 0~7。由于二进制和八进制数制系统的数字都少于十进制，因此它们的数字与十进制中相应的数字相同。

十六进制数制系统需要 16 个数字。其最低位的数字是 0，最高位数字的值等同于十进制的

^① “Web Safe Color Chart”，<https://htmlcolorcodes.com/color-chart/web-safe-color-chart/>。

^② 维基百科，<https://zh.wikipedia.org/wiki/Base64>。

15, 比基数 16 小 1。按照惯例, 字母 A 到 F 表示对应于十进制值 10 到 15 的十六进制数字。十六进制字母 A~F 不区分大小写, 本附录将使用大写字母。因此, 在十六进制中, 我们可以有如下所示的几种数字:

- 876 仅由类似十进制的数字组成;
- 8A55F 由数字和字母组成;
- FFE 仅由 A~F 范围内的字母组成。

有的时候, 一个十六进制数会碰巧拼写出一个常见的单词, 例如 FACE 或 FEED, 这对于习惯于使用十进制数的程序员来说可能看起来很奇怪。

表 C.1 总结了二进制、八进制、十进制和十六进制数制系统的数字。

表 C.1 不同数制系统使用的数字

二进制数字	八进制数字	十进制数字	十六进制数字
0	0	0	0
1	1	1	1
	2	2	2
	3	3	3
	4	4	4
	5	5	5
	6	6	6
	7	7	7
		8	8
		9	9
			A 或 a (十进制值 10)
			B 或 b (十进制值 11)
			C 或 c (十进制值 12)
			D 或 d (十进制值 13)

			E 或 e (十进制值 14)
			F 或 f (十进制值 15)

表 C.2 列出了这些数制系统的基数、最低位和最高位。

特性	二进制	八进制	十进制	十六进制
基数	2	8	10	16
最低位	0	0	0	0
最高位	1	7	9	F

每个数制系统都使用**位置表示法** (positional notation) —— 写入数字的每个位置都有不同的**位值** (positional value)。例如，在十进制数 937 中 (9、3 和 7 被称为**符号值**，即 symbol value)，我们说 7 在个位，3 在十位，9 在百位。所有这些位置都是基数 (10) 的幂次。这些幂次从 0 开始，当我们从右向左移动数字时，它们会递增 1，如表 C.3 所示。

表 C.3 十进制数制系统中的位值

十进制数字	9	3	7
位置名称	百位	十位	个位
位值	100	10	1
位值，作为基数 (10) 的幂次	10^2	10^1	10^0

对于具有更多数字的十进制数，左侧的下一个位置将是：

- 千位 (10 的 3 次方)；
- 万位 (10 的 4 次方)；
- 十万位 (10 的 5 次方)；
- 百万位 (10 的 6 次方)；
- 千万位 (10 的 7 次方)；

- 等等。

在二进制数 101 中，最右边的 1 在个位，0 在二位，最左边的 1 在四位。每个位置都是基数（2）的幂次，并且当我们从右向左移动时，这些幂次从 0 开始并递增 1，如表 C.4 所示。因此， $101 = 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 4 + 0 + 1 = 5$ 。

表 C.4 二进制数制系统中的位值

二进制数字	1	0	1
位置名称	四位（fours）	二位（twos）	个位（ones）
位值	4	2	1
位值，作为基数（2）的幂次	2^2	2^1	2^0

对于更多位数的二进制数，左侧的下一个位置将是：

- 八位（2 的 3 次方）；
- 十六位（2 的 4 次方）；
- 三十二位（2 的 5 次方）；
- 六十四位（2 的 6 次方）；
- 等等。

在八进制数 425 中，5 在个位，2 在八位，4 在六十四位。每个位置都是基数（8）的幂次，并且当我们从右向左移动时，这些幂次从 0 开始并递增 1，如表 C.5 所示。

表 C.5 八进制数制系统中的位值

八进制数字	4	2	5
位置名称	六十四位（sixty-fours）	八位（eights）	个位（ones）

位值	64	8	1
位值，作为基数 (8)的幂次	8^2	8^1	8^0

对于更多位数的八进制数，左侧的下一个位置将是：

- 五百一十二位（8的3次方）；
- 四千零九十六位（8的4次方）；
- 三万二千七百六十八位（8的5次方）；
- 等等。

在十六进制数3DA中，我们说A在个位，D在十六位，3在二百五十六位。每个位置都是基数（16）的幂次，并且当我们从右向左移动时，这些幂次从0开始并递增1，如表C.6所示。

表 C.6 十六进制数制系统中的位值

十六进制数字	3	D	A
位置名称	二百五十六位 (two-hundred-and-fifty-sixes)	十六位 (sixteens)	个位 (ones)
位值	256	16	1
位值，作为基数 (16)的幂次	16^2	16^1	16^0

对于更多位数的十六进制数，左侧的下一个位置将是：

- 四千零九十六位（16的3次方）；
- 六万五千五百三十六位（16的4次方）；
- 等等。

C.2 将二进制数缩写为八进制数和十六进制数

八进制数和十六进制数主要用于缩写冗长的二进制表示。表 C.7 强调了一个事实，即冗长的二进制数可以用比二进制数系统更高基数的数制系统简洁地表示。

表 C.7 对比不同的数制系统

十进制 (基数 10)	二进制 (基数 2)	八进制 (基数 8)	十六进制 (基数 16)
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

八进制和十六进制数制系统与二进制数制系统的一个重要关系是，它们的基数（分别为 8 和 16）是二进制基数（2）的幂。下面展示了一个 12 位二进制数及其对应的八进制和十六进制数，让我们研究如何利用这种关系方便地将二进制数缩写为八进制或十六进制。

二进制数	八进制等价值	十六进制等价值
100011010001	4321	8D1

通过将 12 位二进制数分成每组三个连续的位，并将这些组写到八进制数的对应数字下方，就可以轻松地将二进制数转换为八进制数。

100	011	010	001
4	3	2	1

每组三位下的八进制数字，与表 C.7 中该三位二进制数的八进制等价值完全对应。

从二进制转换为十六进制时，可以观察到类似的关系。将 12 位二进制数分成每组 4 个连续的位，并将这些组写在十六进制数的相应数字下方，如下所示：

1000	1101	0001
8	D	1

每组 4 位下的十六进制数字，与表 C.7 中该 4 位二进制数的十六进制等价值完全对应。

C.3 八进制数和十六进制数转换为二进制数

上一节讲述了如何通过二进制数字进行分组，并简单地将其重写为等价的八进制或十六进制数字，从而将二进制数转换为八进制和十六进制的等价值。此过程可以反向应用，以生成给定八进制或十六进制数的二进制等价值。

例如，八进制数 653 可以通过将 6 写为其 3 位二进制等价值 110，将 5 写为其 3 位二进制等价值 101，将 3 写为其 3 位二进制等价值 011，来简单地转换为 9 位二进制数 110101011。

十六进制数 FAD5 可以通过将 F 写为其 4 位二进制等价值 1111，将 A 写为其 4 位二进制等价值 1010，将 D 写为其 4 位二进制等价值 1101，将 5 写为其 4 位二进制等价值 0101，来简单地转换为 16 位二进制数 1111101011010101。

C.4 从二进制、八进制或十六进制转换为十进制

我们习惯于使用十进制，因此经常需要将二进制、八进制或十六进制数转换为十进制，以了解该数的“真正”大小。C.1 节的多个表格以十进制表示位值。要将一个数从其他基数转换为十进制，请将每个数字的十进制等价值乘以其位值，然后将这些乘积相加。例如，

二进制数 110101 可按如下方式转换为十进制数 53:

符号值:	1	1	0	1	0	1
位值:	32	16	8	4	2	1
乘积:	1*32=32	1*16=16	0*8=0	1*4=4	0*2=0	1*1=1
总和:	= 32 + 16 + 0 + 4 + 0 + 1 = 53					

要将八进制数 7614 转换为十进制数 3980, 我们使用相同的技术, 这次使用八进制相应的位值, 如下所示:

符号值:	7	6	1	4
位值:	512	64	8	1
乘积:	7*512=3584	6*64=384	1*8=8	4*1=4
总和:	= 3584 + 384 + 8 + 4 = 3980			

要将十六进制数 AD3B 转换为十进制数 44347, 我们使用相同的技术, 这次使用十六进制相应的位值, 如下所示:

符号值:	A	D	3	B
位值:	4096	256	16	1
乘积:	10*4096=40960	13*256=3328	3*16=48	11*1=11
总和:	= 40960 + 3328 + 48 + 11 = 44347			

Java 中的二进制、八进制和十六进制字面值

Java 支持直接在代码中写二进制、八进制和十六进制的字面值:

- 一个像 110101 (十进制 53) 这样的二进制值, 可以在 Java 代码中以 `0b` 作为前缀写成二进制字面值, 如 `0b110101`。可以在任何数值字面值中使用下划线 (`_`) 来分隔数字以提高可读性, 如 `0b110_101`;
- 一个像 7614 (十进制 3980) 这样的八进制值, 可以在 Java 代码中以 `0` 作为前缀写成八进制字面值, 如 `07614`;
- 一个像 AD3B (十进制 44347) 这样的十六进制值, 可以在 Java 代码中以 `0x` 或 `0X` 作为前缀写成十六进制字面值, 如 `0xAD3B` 或 `0XAD3B`。

Java 会将二进制、八进制和十六进制的字面值转换为其十进制等价值。为了确认这一点, 可以使用 `System.out` 的打印方法来输出上述列表中的八进制、二进制和十六进制字面值。此外, 每个整数类型的包装类都提供了静态方法, 允许将整数值转换为二进制、八进制或十六进制 `String`。

C.5 从十进制转换为二进制、八进制或十六进制

C.4 节中的转换遵循了位置表示法的约定。从十进制转换为二进制、八进制或十六进制也遵循这些约定。

十进制转换为二进制

假设要将十进制数 57 转换为二进制。首先从右到左写出各列的十进制位值，直至到达第一个位值大于该十进制数的列。我们不需要那一列，因此将其舍弃。于是，我们首先写下：

位值： 64 32 16 8 4 2 1

然后，舍弃位值为 64 的列，得到：

位值： 32 16 8 4 2 1

接着，从最左边的列开始向右处理。用 32 除 57，商 1 余 25，所以在 32 这一列写入 1。接着，用 16 除余数 25，商 1 余 9，在 16 这一列写入 1。用 8 除余数 9，商 1 余 1。用后续两列的位值（4 和 2）来除余数 1 时，商均为 0，所以在 4 和 2 这两列写入 0。最后，用 1 除余数 1，商 1，所以在 1 这一列写入 1。结果如下：

位值： 32 16 8 4 2 1
符号值： 1 1 1 0 0 1

因此，十进制数 57 等价于二进制数 111001。

十进制转换为八进制

要将十进制数 103 转换为八进制，首先从右到左写出各列的十进制位值，直至到达第一个位值大于该十进制数的列，舍弃该列。于是，我们首先写下：

位值： 512 64 8 1

然后，舍弃位值为 512 的列，得到：

位值： 64 8 1

接着，从最左边的列开始向右处理。用 64 除 103，商 1 余 39，所以在 64 这一列写入 1。用 8 除余数 39，商 4 余 7，在 8 这一列写入 4。最后，用 1 除余数 7，商 7，没有余数，所以在 1 这一列写入 7。结果如下：

位值： 64 8 1
符号值： 1 4 7

因此，十进制数 103 等价于八进制数 147。

十进制转换为十六进制

要将十进制数 375 转换为十六进制，首先从右到左写出各列的十进制位值，直至到达第一个位值大于该十进制数的列，舍弃该列。于是，我们首先写下：

位值: 4096 256 16 1

然后, 舍弃位值为 4096 的列, 得到:

位值: 256 16 1

接着, 从最左边的列开始向右处理。用 256 除 375, 商 1 余 119, 所以在 256 这一列写入 1。用 16 除余数 119, 商 7 余 7, 在 16 这一列写入 7。最后, 用 1 除余数 7, 商 7, 没有余数, 所以在 1 这一列写入 7。结果如下:

位值: 256 16 1
符号值: 1 7 7

因此, 十进制数 375 等价于十六进制数 177。

C.6 负二进制数: 补码表示法

本附录到目前为止主要讨论正数。本节解释了计算机如何使用**补码表示法** (two's complement notation) 来表示负二进制数。首先解释如何生成二进制数的补码, 然后展示为什么它表示给定二进制数的负值。

Java 使用 32 位 `int`。假设执行以下语句:

```
int value = 13;
```

`value` 的 32 位表示形式是:

```
00000000 00000000 00000000 00001101
```

为了生成 `value` 的负数, 我们首先通过应用 Java 的按位取反操作符 (`~`) 来生成它的**反码** (one's complement):

```
onesComplementOfValue = ~value; // 反码
```

在内部, `~value` 现在是 `value` 中每一位反转后的结果, 即 1 变成 0, 0 变成 1, 如下所示:

```
value:  
00000000 00000000 00000000 00001101
```

```
~value (即 value 的反码):  
11111111 11111111 11111111 11110010
```

为了生成 `value` 的补码, 我们将 1 加到 `value` 的反码上, 结果如下:

```
value 的补码:  
11111111 11111111 11111111 11110011
```

如果这实际上等于 -13, 就应该能将它加到二进制数 13 上, 并得到结果 0。让我们试试:

```

00000000 00000000 00000000 00001101
+11111111 11111111 11111111 11110011
-----
00000000 00000000 00000000 00000000

```

注意，最左边那一列的 **1 进位** (carry bit) 被丢弃，最终的结果确实是 0。如果将一个数的反码加到这个数上，结果必然是全 1。得到全零结果的关键是补码比反码多 1。加 1 导致每一列的和为 0，进位为 1。进位一直向左移动，直到它从最左边的位被丢弃。因此，结果数的十进制位全是 0。

计算机在执行减法时，例如：

```
x = a - value;
```

实际上是通过将 value 的补码加到 a 上来实现，如下所示：

```
x = a + (~value + 1);
```

假定 a 为 27，value 和之前一样为 13。如果 value 的补码实际上是 value 的负数，那么将 value 的补码加到 a 上应该得到结果 14。让我们试试：

```

a(即 27)      00000000 00000000 00000000 00011011
+ (~value + 1) +11111111 11111111 11111111 11110011
-----
                00000000 00000000 00000000 00001110

```

这确实是与 14 等价的二进制值。