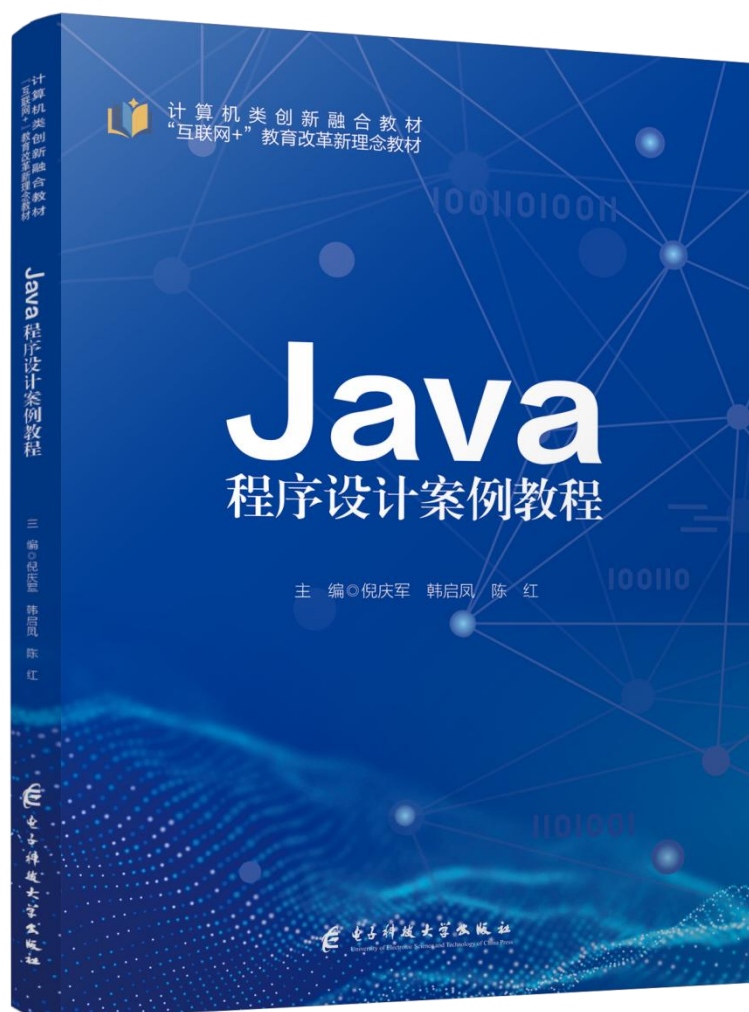


Java 程序设计案例教程



类目：计算机类

书名：Java 程序设计案例教程

主编：倪庆军 韩启凤 陈 红

出版社：电子科技大学出版社

开本：大 16 开

书号：978-7-5770-2186-7

使用层次：通用

出版时间：2026 年 5 月

定价：49.80 元

印刷方式：双色

是否有资源：有

策划编辑 万晓桐
责任编辑 万晓桐
封面设计 万晓桐

计算机类创新融合教材
“互联网+”教育改革新理念教材



计算机类创新融合教材
“互联网+”教育改革新理念教材

Java

程序设计案例教程

Java 程序设计案例教程

主 编 倪庆军 韩启凤 陈 红

Java

程序设计案例教程

主 编 倪庆军 韩启凤 陈 红



定价：49.80元

电子工业出版社

电子工业出版社



计算机类创新融合教材
“互联网+”教育改革新理念教材

10011010011

Java

程序设计案例教程

主 编◎倪庆军 韩启凤 陈 红
副主编◎周大勇 张凤林 刘 玮
李玉峰 邵 雪 王 炜
魏 雯 周 睿 席 洁
赵翊凡

100110

1101001



电子科技大学出版社
University of Electronic Science and Technology of China Press

· 成都 ·

图书在版编目 (CIP) 数据

Java 程序设计案例教程 / 倪庆军, 韩启凤, 陈红主编. -- 成都: 成都电子科大出版社, 2026. 5. -- ISBN 978-7-5770-2186-7

I. TP312.8

中国国家版本馆 CIP 数据核字第 2025BX0414 号

Java 程序设计案例教程

Java CHENGXU SHEJI ANLI JIAOCHENG

倪庆军 韩启凤 陈 红 主编

策划编辑 万晓桐

责任编辑 万晓桐

责任校对 罗 雅

责任印制 梁 硕

出版发行 电子科技大学出版社

成都市一环路东一段 159 号电子信息产业大厦九楼 邮编 610051

主 页 www.uestcp.com.cn

服务电话 028-83203399

邮购电话 028-83201495

印 刷 三河市祥达印刷包装有限公司

成品尺寸 210 mm×285 mm

印 张 15.5

字 数 441 千字

版 次 2026 年 5 月第 1 版

印 次 2026 年 5 月第 1 次印刷

书 号 ISBN 978-7-5770-2186-7

定 价 49.80 元

版权所有, 侵权必究

前言 PREFACE

党的二十大报告明确指出“必须坚持科技是第一生产力、人才是第一资源、创新是第一动力”，“加快建设网络强国、数字中国，加快发展数字经济”。在数字化浪潮席卷全球的今天，Java 作为全球应用得最广泛的编程语言之一，已成为推动信息技术创新、赋能产业数字化转型的核心工具。学习 Java 程序设计，不仅能掌握一门实用技术，还能响应国家“科技自立自强”号召、投身数字中国建设。

本书以培养“懂技术、能创新、有担当”的新时代编程人才为目标，系统讲解了 Java 程序设计的核心知识与实战技能。本书以“项目驱动、案例导向”为核心理念，构建了从开发环境搭建到高级特性应用的完整学习体系，涵盖 Java 基础语法、接口与抽象类、流程控制、数组、面向对象编程、异常处理、常用 API、集合框架及 I/O 流等九大核心项目；通过任务描述、任务分析、任务实施、项目复盘的闭环设计，结合丰富的案例与实训任务，帮助读者夯实编程基础、提升实战能力，为投身数字中国建设、赋能产业数字化转型奠定坚实基础。

本书特色鲜明，主要体现在以下几个方面：

1. 思政融合，价值引领

将“科技报国”“工匠精神”“开源协作”等思政元素自然融入技术讲解，在传授编程技能的同时，引导读者树立正确的技术价值观与社会责任感，契合新时代人才培养要求。

2. 项目驱动，层层递进

以九大项目贯穿全书，每个项目围绕真实场景设计，从基础到进阶逐步深入，将复杂知识拆

解为可操作的阶段性任务，帮助读者明确学习路径，实现“知识输入”到“能力输出”的转化。

3. 案例丰富，贴近实战

精选电商后台、成绩管理、文件处理等高频应用场景案例，配套“项目实训”“码到功成”模块，涵盖基础练习、综合应用与拓展挑战，助力读者学以致用，积累工程经验。

4. 工具全面，紧跟趋势

除核心语法外，本书还介绍了集合框架、异常处理、常用 API、I/O 流等企业级开发必备技能，帮助读者快速对接实际开发需求。

本书既可作为计算机相关专业的教材，也适合 Java 初学者及开发人员参考。

在本书的编写过程中，编者参考、引用了部分相关书籍和资料，在此向原作者表示衷心的感谢，如涉及版权问题请及时联系我们。由于涉及面较宽泛，限于作者水平，书中难免会出现一些疏漏之处，恳请广大师生在使用过程中批评指正，并提出宝贵的意见和建议，以便我们在之后不断改进和完善。

目录 CONTENTS

项目一

初探Java——环境配置与首程编写 / 1

- 任务一 了解 Java 特点 / 2
- 任务二 配置 Java 开发环境 / 9
- 任务三 编写首个 Java 程序 / 13
- 任务四 掌握 Java 程序结构 / 17

项目三

流程控制——结构与分支 / 57

- 任务一 灵活使用分支结构 / 58
- 任务二 掌握循环与跳转语句 / 69
- 任务三 定义与调用方法 / 80

项目二

数据基石——Java类型与操作 / 25

- 任务一 解析关键字与标识符 / 26
- 任务二 管理变量与常量 / 29
- 任务三 转换数据类型 / 32
- 任务四 运用运算符与表达式 / 44

项目四

数组世界——从一维到二维 / 87

- 任务一 初探一维数组 / 89
- 任务二 掌握一维数组排序 / 98
- 任务三 解密二维数组 / 105

项目五

面向对象基础——类与继承 / 114

- 任务一 初识类与对象 / 115
- 任务二 详解继承机制 / 129
- 任务三 实现多态性 / 136

项目六

面向对象进阶——抽象、接口与异常 / 146

- 任务一 掌握抽象类 / 147
- 任务二 设计与应用接口 / 150
- 任务三 处理异常情况 / 155

项目七

Java实用工具——常用API / 167

- 任务一 精通字符串操作 / 168
- 任务二 生成随机数与处理日期 / 174
- 任务三 使用包装类 / 183

项目八

数据容器——集合框架 / 192

- 任务一 应用集合接口 / 193
- 任务二 探究集合遍历、排序与泛型机制 / 204

项目九

数据通道——I/O流 / 216

- 任务一 掌握 I/O 流分类与操作 / 217
- 任务二 学习文件处理方法 / 234

参考文献 / 242

项目一

初探 Java——环境配置与首程编写

项目全景

在数字化浪潮席卷全球的当下，Java 凭借其跨平台特性、稳健性与广泛的生态应用，成为互联网、大数据、企业级开发等领域的核心编程语言。从电商平台的后台架构到移动应用的开发，从云计算服务到人工智能算法落地，Java 的身影无处不在。对于编程入门者而言，掌握 Java 的基础体系、搭建开发环境并编写首段代码，是叩开编程世界大门的关键一步。为帮助读者夯实 Java 的入门基础，本项目将围绕 Java 特点、开发环境配置、首个 Java 程序编写及程序结构等内容展开详细介绍。

学习目标

■ 知识目标

1. 掌握 Java 的发展史、核心特点及体系分类，理解 Java 程序的工作机制。
2. 熟悉 JDK 的下载安装流程与环境变量配置方法，了解 Java 程序编译运行的原理。
3. 掌握 Java 源程序的编写规范、编译步骤及程序结构的核心概念。

■ 能力目标

1. 能够独立完成 Java 开发环境的搭建与测试，排查环境配置中的常见问题。
2. 能够编写简单的 Java 源程序，完成编译与运行，并解决基础语法错误。
3. 能够运用 Java 程序结构知识组织代码，合理使用包、主函数参数及代码块。

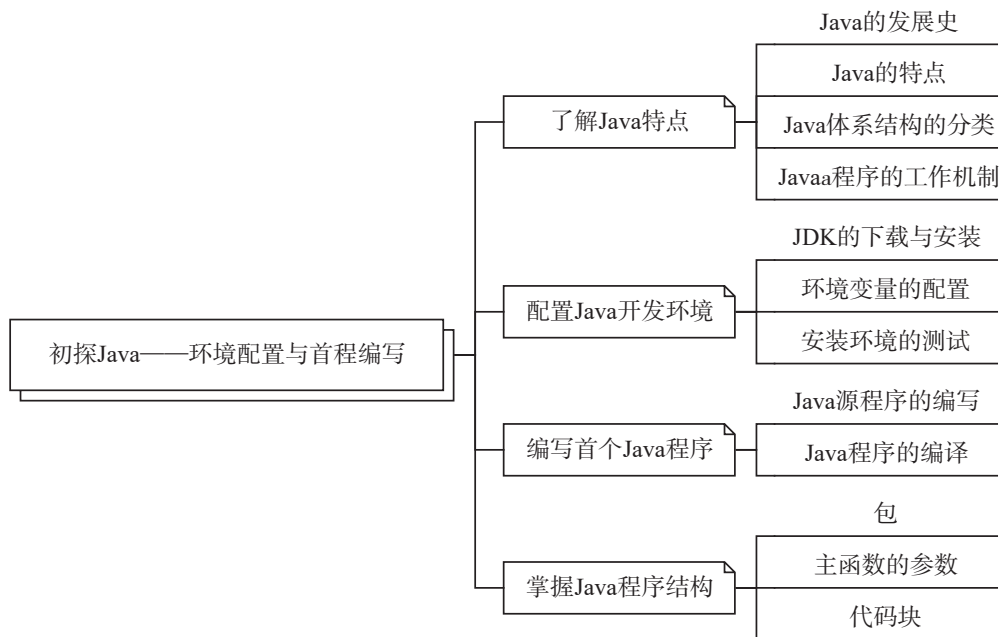
■ 编程素养

1. 培养严谨的编程思维与代码规范意识，提升问题排查与解决的能力。
2. 增强自主学习与实践探索的能力，形成系统化的知识梳理习惯。
3. 树立团队协作中的代码可读性与可维护性意识。

思政赋能

1. 从 Java 技术的发展历程中理解创新与坚持的价值，培养科技报国的使命感。
2. 通过规范编程训练，树立求真务实、精益求精的工匠精神。
3. 认识开源生态的协作理念，培养开放包容、互利共赢的团队价值观。

编程脑图



任务一 了解 Java 特点

任务描述

小王在刚入职的一家互联网公司担任 Java 开发工程师助理。他参加了公司组织的新人技术培训。培训中，导师要求每位新人用一段话向团队介绍“为什么 Java 能在企业级开发中长期占据主流地位”。小王查阅资料后了解到 Java 具有跨平台、安全性高、自动内存管理等优势，但对“一次编写，到处运行”背后的机制仍感模糊——他不清楚 JVM、字节码和垃圾回收如何协同工作，也不理解为何 Java 能比 C++ 更安全。面对即将进行的 5 分钟分享，小王意识到必须深入理解 Java 的核心特点与工作机制。那么，要准确、专业地阐述 Java 的优势，小王需要系统掌握哪些关键知识点？

任务分析

小王需要掌握 Java 语言的核心特性及其技术实现原理。首先，理解“平台无关性”的实现依赖于 Java 虚拟机（JVM）和字节码机制：源代码编译为与硬件无关的 .class 文件，由不同平台上的 JVM 解释执行。其次，掌握 Java 的安全机制，如无指针设计、字节码校验、类加载隔离等，避免内存非法访问。再次，了解垃圾自动回收机制如何替代手动内存管理，减少内存泄漏风险。最后，还需熟悉 Java 的面向对象特性、多线程支持及高性能表现，并能结合 Java SE/EE/ME 体系说明其在不同场景下的适用性。

任务实施

一、Java 的发展史

Java 作为全球广泛应用的面向对象程序设计语言，其诞生与发展始终紧扣技术变革与市场需求。

（一）Java 语言的起源与早期发展历程

从早期嵌入式应用的探索到成为跨平台开发的核心工具，其演进轨迹清晰地反映了编程语言的发展规律。

1. Java 语言的诞生背景与命名演变

20 世纪 90 年代，全球电子产品进入快速迭代期，提升设备智能化水平成为行业竞争的核心焦点。在这一技术浪潮下，太阳计算机（Sun Microsystems）公司为抢占嵌入式应用市场先机，组建了以詹姆斯·高斯林（James Gosling）为核心的“格林”（Green）项目小组。

该小组的核心目标是研发适用于家电产品的嵌入式应用新技术。经过持续攻关，该小组于 1991 年成功开发出一种面向对象的编程语言，最初将这种面向对象的编程语言命名为 Oak。1995 年，为适配更广泛的应用场景并规避命名版权问题，该语言正式更名为 Java，标志着其商业化发展的开端。

2. 早期关键版本与核心组件发布

Java 语言的早期发展以核心版本迭代和基础生态构建为主要特征，关键版本的发布为其后续普及奠定了坚实基础。

1996 年 1 月，Sun 公司正式推出 Java 1.0 版本，该版本包含两大核心组件：Java 运行时环境（Java runtime environment, JRE）和 Java 开发工具包（Java development kit, JDK）。JRE 为 Java 程序提供了运行所需的基础环境，JDK 则集成了编译、调试等开发必需的工具，二者共同构成了 Java 开发与运行的完整支撑体系。

1998 年 12 月，JDK1.2 的发布成为 Java 发展史上的重要里程碑。Sun 公司同步启用“Java 2”这一品牌名称，不仅对语言语法和 API 进行了大幅扩充，还构建了 J2SE、J2EE、J2ME 三大应用平台，覆盖了桌面应用、企业级应用和移动嵌入式应用等多个领域，极大地拓展了 Java 的应用边界。

（二）Java 的所有权更迭与发展态势

进入 21 世纪后，Java 语言历经所有权变更与技术革新，在保持跨平台核心优势的同时，通过灵活的版本更新策略和特性升级，持续适配云计算、人工智能等新兴技术场景，始终稳居主流编程语言

言行列。

1. 所有权变更与版本更新策略调整

2009 年，Oracle（甲骨文）公司完成对 Sun 公司的收购，Java 及相关平台工具正式纳入 Oracle 的技术生态体系。Oracle 延续了 Java 的开源路线与商业化并行策略，持续投入资源进行技术迭代与生态完善，确保了语言发展的连续性与稳定性。

2017 年 9 月，Oracle 发布 JDK 9 版本，同步宣布调整 JDK 的更新频率为每半年发布一个新版本，形成“功能版本 + 长期支持版本（LTS）”的双轨发布模式。这一调整既满足了开发者对新特性的快速需求，又通过长期支持版本为企业级用户提供了稳定的生产环境保障，平衡了创新与稳定的双重诉求。

2. 关键长期支持版本与最新技术突破

长期支持版本（LTS）作为 Java 生态的核心支柱，每 3 年左右发布一次，提供长达数年的安全更新与技术支持，成为企业级应用的首选版本。

从技术演进来看，Java 8.0 引入的 Lambda 表达式、StreamAPI 等特性重塑了代码编写范式；Java 11.0 实现了 HTTP Client 标准化与 ZGC 垃圾回收器的实验性引入；Java 17.0 推出密封类特性，增强了代码的安全性与可维护性；Java 21.0 则以虚拟线程、Record 模式等重磅特性，大幅提升了高并发场景下的性能表现。

2025 年，Oracle 发布 Java 25.0 版本，该版本定位为未来长期支持版本，重点强化了对人工智能与机器学习工作负载的支持能力。其核心特性包括作用域值、灵活构造器、紧凑源文件等，其中作用域值有效解决了传统 ThreadLocal 的内存泄漏与数据篡改问题，在虚拟线程环境下具备更优的内存开销与安全性；向量 API 的持续孵化则进一步提升了计算密集型应用的执行效率。此外，Java 25.0 移除了对 32 位 x86 架构的支持，优化了分代 Shenandoah 垃圾回收机制，同时承诺提供至少 8 年的商业支持，为企业在云原生与 AI 时代的技术选型提供了可靠保障。



编程拓知

计算机语言的发展史

计算机语言是人与计算机通信的工具，由指令组成，主要功能是实现人机交互。其发展伴随着计算机软硬件进步，历经了下面三个阶段。

一是机器语言，用二进制代码表示指令，是计算机唯一能直接识别、理解和执行的语言，无须编译，运行效率高、速度快，但难读、难懂、难记，不利于开发，被称为低级语言或第一代语言。

二是汇编语言，面向微处理器等编程器件，用简单字母和单词表示指令，不同计算机指令集不同。它与计算机相关性强、运行效率较高，但可读性和移植性差，应用范围窄，被称为中级语言或第二代语言。

三是高级语言，接近人类自然语言，与硬件无关，有特定符号和语法规则。它数量繁多，分为面向过程和面向对象语言，可读性强、易于学习，但需编译，运行速度相对较慢。

二、Java 的特点

Java 是一种高级、通用的面向对象程序设计语言，语法与 C、C++ 相近但组织结构差异显著。

它以类为程序基本处理单位，设计初衷偏向实用生产力而非单纯研究，让开发者在学习与运用中能快速体会其便捷性与强大功能。以下为 Java 语言的核心特性。

（一）语法设计特性

语法设计的核心目标是简化开发流程，让不同基础的开发者都能快速上手。Java 在借鉴经典编程语言优势的同时，剔除冗余复杂的设计，形成了简洁、清晰的语法体系。

1. 简单易学

基于 C 语言和 C++ 构建，保留两者核心实用的语法元素，移除多继承、指针、操作符重载等难以理解、易混淆的内容，语法结构更简洁，降低了学习与掌握的难度。

2. 语法兼容

语法风格与 C、C++ 相近，多数有编程基础的开发者能快速衔接，减少跨语言学习的适配成本。

（二）编程思想特性

面向对象是 Java 语言的核心设计思想，改变了传统编程的逻辑架构，为大规模、复杂软件的开发提供了高效解决方案。

1. 纯粹面向对象

以对象为中心、消息为驱动，提供类、接口、继承等核心原语，仅支持类的单继承，支持接口多继承及类与接口的实现机制。

2. 动态绑定支持

全面支持动态绑定机制，相较于 C++ 仅对虚函数使用动态绑定的设计，进一步提升了代码的灵活性与扩展性。

3. 简化存储管理

自动处理对象引用，开发者无须关注底层存储管理，可将精力集中在核心功能研发上，有效地提升开发的效率与软件质量。

（三）运行机制特性

Java 的运行机制突破了传统编程语言的平台限制，凭借独特的编译与解释方式，实现了“一次编写，到处运行”的核心优势。

1. 解释型执行

源程序编译后生成字节码文件（.class），而非直接执行的机器语言指令，由 Java 虚拟机（JVM）解释执行，适配不同的硬件与操作系统。

2. 平台无关性

字节码文件依赖 Java 虚拟机运行，针对 Windows、Linux 等不同操作系统安装对应虚拟机，即可实现程序跨平台运行，无须修改源代码。

3. 结构中立

编译后的字节码格式与体系结构无关，只要安装 Java 运行时系统，就能在任意处理器上运行，增强了程序的兼容性与部署灵活性。

4. 系统可移植性

严格规定基本数据类型长度，编译器基于 Java 实现，解释器采用标准 C 编写，确保 Java 系统本身具备较强的可移植性。

初学者常误以为“Java 跨平台”意味着无须关注底层环境，实则不然。JVM 虽屏蔽了操作系统差异，但不同厂商和版本在性能、GC 策略、安全补丁上存在显著差异。建议在企业项目中优先选用官方长期支持（LTS）的版本，并明确 JDK 发行商，避免因版本碎片化导致的部署异常或兼容问题。此外，开发与生产环境应保持 JDK 版本一致，防止“本地能跑，线上报错”。

（四）安全稳健特性

Java 通过一系列底层设计与机制，从根源上减少程序运行中的安全隐患与异常问题，保障软件的稳定可靠。

1. 安全机制完善

语言层面删除指针、手动释放内存等易引发非法操作的功能；编译器执行代码校验、指针操作检测、类型转换检测等多重测试；字节码校验确保无堆栈溢出、参数类型正确等；类装载通过命名空间区分本机类与网络资源类，避免特洛伊木马病毒。

2. 稳健性保障

采用强类型机制，搭配异常处理机制，能有效捕获并处理运行中的异常；垃圾自动回收机制可动态回收无引用对象的内存，无须开发者手动管理，减少了内存泄漏风险。

（五）功能扩展特性

Java 内置多种实用功能，支持多任务并行处理与动态调整，能满足不同场景下的开发需求，尤其适用于复杂系统构建。

1. 多线程支持

内置 Thread 类与 Runnable 接口，原生支持多线程功能，可在一个程序中同时执行多个小任务，提升交互性能与实时控制能力。

2. 分布式处理

提供网络应用编程接口（含 URL、Socket 等类库），支持 Internet 应用开发，可通过特定 URL 便捷访问网络对象，操作如同访问本地文件系统。

3. 动态适配能力

支持动态载入运行所需的类资源，可通过本地环境或网络获取类资源，能灵活适应动态变化的运行环境。

4. 高性能表现

相较于传统解释型高级脚本语言，Java 运行效率更高，且随着即时编译（JIT）技术的发展，运行速度已逐步接近 C++。

三、Java 体系结构的分类

Java 的内涵并非局限于程序设计语言本身，而是涵盖了一套完整的程序开发平台环境。这套环境集成了开发与运行 Java 软件所需的各类资源，包括编译器等开发工具、丰富的软件库以及核心的 Java 虚拟机。基于不同的应用场景与市场需求，Java 平台衍生出三个核心版本，各版本定位明确，为不同领域的开发提供针对性支持。

Java 平台的三个版本分别对应桌面系统、企业级服务、嵌入式设备三大核心领域，形成了覆盖

从微型设备到大型服务器的全场景技术体系。这种差异化的版本设计，使得软件开发、服务提供商及设备生产商能够根据具体需求，选择最适合的技术方案，提升开发效率与产品适用性。

（一）Java 标准版

Java 标准版（Java standard edition, Java SE）是整个 Java 平台的核心与基础，为其他版本提供技术支撑。该版本具备开发和部署多场景 Java 应用程序的能力，应用范围覆盖桌面端、服务器、嵌入式设备及实时运行环境。从核心组件来看，Java SE 主要包含 Java 开发工具包（JDK）、Java 运行时环境（JRE），以及一系列核心类库，这些类库涵盖用户界面（UI）、集合框架、异常处理、多线程、输入/输出流（I/O 流）、数据库编程、网络编程等关键功能模块。

（二）Java 企业版

Java 企业版（Java enterprise edition, Java EE）专为解决企业级应用开发、部署与管理过程中的复杂问题而设计。作为构建于 Java SE 之上的高级版本，它完整保留了 Java SE 的核心特性，同时针对企业级需求拓展了丰富的技术支持。其核心技术体系包括企业级 JavaBean（enterprise Java bean, EJB）、Servlet、Java 服务器页面（Java server pages, JSP）、可扩展标记语言（eXtensible markup language, XML）等，能够满足大型企业应用在安全性、可扩展性、事务管理等方面的严苛要求。

（三）Java 微型版

Java 微型版（Java micro edition, Java ME）是面向嵌入式消费电子设备的轻量级 Java 平台，主要适配机顶盒、移动电话、个人数字助理（personal digital assistant, PDA）等设备。该版本的核心组成包括适配嵌入式环境的 Java 虚拟机，以及一系列标准化的 Java 应用程序接口（application program interface, API）。嵌入式装置根据运算能力与电力供应可分为两类：一类是运算能力有限、电力资源紧张的设备，如 PDA、手机；另一类是运算能力较强、电力供应稳定的设备，如冷风机、电冰箱、机顶盒。针对这些设备的特性，Java ME 配备了专属类库，涵盖用户界面、安全模型、内置网络协议等模块，同时支持动态下载联网及离线应用程序，满足嵌入式设备的多样化需求。

四、Java 程序的工作机制

Java 语言的核心设计思想聚焦于跨平台特性与安全性保障，基于这一目标，Java 衍生出了 Java 虚拟机、Java 字节码以及垃圾自动回收机制这三大关键核心技术。

（一）Java 虚拟机

平台无关性堪称 Java 最为关键的特性，而实现这一特性的关键支撑便是 Java 虚拟机（Java virtual machine, JVM）。从底层原理层面剖析，Java 虚拟机本质上是以 Java 字节码作为指令组的软 CPU。图 1-1 呈现了 Java 程序的运行过程。从该图能够清晰地看出，在服务器端，开发者首先编写 Java 源程序，并将其以 .java 文件的形式进行存储；其次，Java 编译器会把 .java 文件编译成字节码，并保存为 .class 文件；最后将 .class 文件

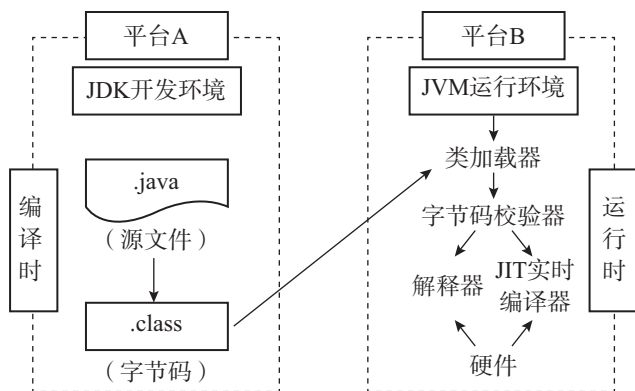


图 1-1 Java 程序的运行过程

存放在 Web 服务器之上。在客户端，用户访问服务器端的主页，下载 Java 程序，随后依靠本地 Java 虚拟机对 .class 文件进行解释与执行。

Java 虚拟机涵盖了类加载器、字节码校验器以及 JIT 实时编译器。类加载器的主要职责是获取从网络获取的或者存储于本地机器上的类文件字节码。字节码校验器则会对这些类文件的格式进行严格检查，以此确保在程序运行过程中不会出现破坏内存的行为。Java 解释器负责将字节码解释并翻译成机器码，而 JIT 实时编译器同样能够将字节码转换为本地机器码，并且它能够使原本采用解释执行方式的虚拟机提升至编译时的运行效率。

（二）Java 字节码

Java 源程序经过 Java 编译器编译后所生成的产品为字节码文件（.class）。这种字节码与机器码存在本质区别，它并非真实 CPU 能够直接执行的机器代码，因此也被称为“伪代码”。字节码文件需要在执行期间由 JDK 将其编译成能够被本地机器识别的机器码，然后交由操作系统执行。由此可见，Java 的跨平台特性不仅体现在源代码层面，还体现在可执行文件层面。只要目标操作系统安装了 JDK，就能够执行任意的字节码文件，进而实现“Write Once, Run Anywhere”的理想状态。

（三）垃圾自动回收机制

Java 语言的一个突出特点在于引入了垃圾自动回收机制，这一机制成功解决了令 C++ 程序员颇为头疼的内存管理难题。它使得 Java 程序员在编写程序时，无须再考虑内存空间的释放问题，所有相关工作均由 Java 的垃圾自动回收机制来完成。垃圾回收机制能够有效地防止内存泄漏，同时高效地利用空闲内存。所谓内存泄漏，指的是内存空间在使用完毕后未被回收，即内存中某个对象的生命周期超出了程序对其所需的时间长度。

计算机仅能识别二进制数据，无法直接执行 Java 源文件。因此，需要借助一个“翻译官”，将 Java 源文件“翻译”成计算机能够识别的格式。

以下通过分析一段简单的 Java 程序，来熟悉其构成元素：

```
public class Example1_1 {  
    public static void main(String[ ] args) {  
        System.out.println(" 欢迎新同学 !");  
    }  
}
```

以上是一个入门级 Java 程序，其结构可拆解如下：

程序以 public class Example1_1 开始，定义了一个名为 Example1_1 的公共类。

在 Java 中，类名必须与保存的文件名一致，否则编译会出错。

类内部包含 public static void main 方法，这是 Java 程序的入口点。

各修饰符的含义如下：

public：表示方法可被外部访问；

static：表示方法属于类本身，无须创建对象即可调用；

void：表示方法无返回值；

String[] args：是一个字符串数组参数，用于接收程序运行时传入的命令行参数。

执行语句如下：

主方法内部只有一条语句：`System.out.println(" 欢迎新同学 !");`

这是 Java 的标准输出语句，功能是在控制台打印双引号中的内容，并自动换行。

论码启思

Java 通过“编译为字节码 + VM 解释执行”的机制实现了跨平台能力，但这也带来了性能开销的质疑。有人认为：“虽然现代硬件性能强大，但是 JVM 的解释执行和垃圾回收增加了复杂性，不如直接使用编译型语言，如 Go 或 Rust。”请讨论这种观点是否合理。

任务二 配置 Java 开发环境

任务描述

小王作为新入职的 Java 开发工程师助理，被安排在自己的办公电脑上配置 Java 开发环境，以便参与后续的代码学习与简单功能开发。他按照公司提供的文档下载了 JDK 1.8 安装包并顺利完成安装，但在配置系统环境变量时，误将 JRE 的 bin 路径而非 JDK 的 bin 路径添加到 Path 中。结果在命令行中执行 `javac` 命令时提示“不是内部或外部命令”，而 `java -version` 却能正常显示版本信息。导师指出问题出在环境变量配置不完整，并要求小王重新检查 JDK 目录结构和 Path 设置。那么，要正确完成 Java 开发环境的搭建，小王需要掌握哪些关于 JDK 组成与环境变量配置的关键知识？

任务分析

小王需要理解 JDK 与 JRE 的区别，明确 JDK 包含编译工具而 JRE 仅支持运行；掌握 JDK 安装后的目录结构，特别是 bin 文件夹的作用；准确配置系统 Path 环境变量，必须指向 JDK 的 bin 目录而非 JRE 的；能够通过 `java -version` 和 `javac -version` 等命令验证开发环境是否完整可用；熟悉 Windows 系统环境变量的设置流程，避免路径拼写错误或权限问题。

任务实施

一、JDK 的下载与安装

JDK 作为 Java 开发的核心工具支撑，其功能构成与版本演进是掌握 Java 开发的基础前提。明确 JDK 与 JRE 的关系，能够为后续安装操作提供清晰的逻辑依据，确保开发环境搭建的准确性与高效性。

（一）JDK 与 JRE 的核心概念

JDK 全称为 Java 开发工具包，是 Java 开发工作的基础工具集合，内部集成了 Java 编译工具、运行工具、文档生成工具及文件打包工具等核心组件，为 Java 程序的开发、调试与部署提供全面支持。从版本发展来看，1996 年 Sun 公司正式发布 JDK 1.0，开启了 Java 技术的应用历程，后续又相继推出 JDK 1.1、JDK 1.2 等迭代版本。在当前的开发场景中，JDK 1.6/Java 6.0、JDK 1.7/Java 7.0 及 JDK 1.8/Java 8.0 因具备良好的兼容性与稳定性，仍被广泛应用。

JRE 即 Java 运行时环境，其核心功能是为 Java 程序提供运行支撑，仅包含 Java 运行所需的基础工具，不具备程序编译能力。需要重点说明的是，JDK 安装包中已内置完整的 JRE 组件，因此在环境搭建过程中，只需完成 JDK 的安装即可满足开发与运行双重需求，无须单独安装 JRE，此举可有效简化环境配置流程，提升操作便捷性。

掌握 JDK 与 JRE 的核心区别后，即可进入实际的下载与安装环节。不同操作系统对应的 JDK 安装文件存在差异，但安装与配置的核心逻辑基本一致，以下将以 64 位 Windows 10 操作系统和 JDK 1.8 为具体场景，详细演示完整操作流程。

（二）JDK 的下载操作

JDK 安装文件的获取需通过官方渠道保障安全性与完整性，下载前需精准匹配操作系统版本与所需 JDK 版本，这是确保后续安装流程顺利推进的基础。

用户可直接访问 Oracle 官方网站的 JDK 下载专区，根据自身计算机的操作系统类型及系统位数（32 位或 64 位），选择对应的 JDK 版本进行下载。本演示案例中，结合 64 位 Windows 10 操作系统的需求，选取 JDK 1.8 作为安装版本，对应的安装文件名称为“jdk-8u40-windows-x64.exe”。下载完成后，通过双击该安装文件启动 JDK 安装程序，进入安装向导界面。

完成安装文件的下载与启动后，根据安装向导提示完成对应设置即可。

（三）JDK 的安装流程

JDK 的安装流程设计简洁，若对安装路径无特殊要求，采用默认配置即可完成安装。整个过程分为定制安装设置、安装进度监控及安装完成三个核心环节，以下将逐一说明。

1. 定制安装界面设置

在安装向导的初始界面中单击“下一步”按钮，系统将跳转至 JDK 定制安装界面。该界面左侧展示了三个核心功能模块，单击任意模块即可查看对应的功能说明。开发者可根据实际开发需求选择需要安装的模块，对于常规开发场景而言，保持默认的模式选择即可满足需求，无须额外修改。

2. 安装进度监控与完成

完成定制安装设置后，再次单击“下一步”按钮，进入 JDK 安装进度界面，界面将实时显示当前的安装进度与完成百分比。JDK 的安装通常在数分钟内即可完成。安装进度达到 100% 后，系统将自动跳转至安装完成界面，此时单击“关闭”按钮，即可结束整个 JDK 安装流程。

JDK 安装完成后，其安装目录中包含多个核心文件夹，这些文件夹承载着不同的功能，明确各文件夹的作用有助于后续开发工作的高效开展。

（四）JDK 安装目录核心构成

JDK 安装完成后，在指定的安装路径下会生成两个核心文件夹，分别为 jdk 文件夹和 jre 文件夹。若采用默认安装路径，可通过访问“C:\Program Files\Java”路径找到这两个文件夹，文件夹名称

中包含的具体版本数字与所下载的 JDK 版本保持一致。其中，jdk 文件夹作为 Java 开发环境的核心目录，包含多个关键子文件夹，其功能如下。

1. bin 文件夹

bin 文件夹用于存放各类可执行程序，核心程序包括 javac.exe、java.exe 及 javadoc.exe 等，是 Java 开发与运行的核心工具集合。

2. include 文件夹

include 文件夹包含一系列 C 语言头文件，由于 JDK 的底层部分通过 C 语言与 C++ 实现，因此在 JDK 启动及运行过程中，需要引入这些头文件以保障相关功能的正常调用。

3. jre 文件夹

jre 文件夹作为 Java 运行时环境的根目录，内部集成了 Java 虚拟机（JVM）及 Java 程序运行所需的各类基础类库，为 Java 程序的运行提供底层环境支撑。

4. lib 文件夹

“lib”为“library”的缩写，是 Java 类库的核心存储目录，包含了 Java 开发过程中常用的类库文件，为程序开发提供丰富的 API 支持。

论码启思

既然 JDK 已内置 JRE，那么为什么 Oracle 仍单独提供 JRE 下载？在仅需运行 Java 程序的场景下，安装完整 JDK 是否会造成资源浪费？请结合 JDK 与 JRE 的功能差异及实际使用场景进行讨论。

二、环境变量的配置

JDK 安装操作完成后，系统无法自动识别其运行路径，因此需要手动配置 Path 系统环境变量。该配置可使操作系统在任意目录下识别 Java 相关命令，为后续 Java 程序开发、编译及运行提供基础支持。

环境变量的配置需遵循固定操作流程，核心在于准确定位系统环境变量设置入口，并将 JDK 安装目录中的关键路径添加至 Path 变量中。配置工作的第一步是进入系统环境变量管理界面，这需要通过桌面系统图标逐步操作以打开对应设置窗口，确保后续配置操作在正确的环境中进行。以下为具体的环境变量配置步骤。

（一）打开环境变量对话框

在电脑桌面找到“此电脑”图标，用鼠标右键点击该图标。在弹出的快捷菜单中，依次点击“属性”选项、“高级系统设置”按钮，随后在弹出的窗口中点击“环境变量”按钮。完成上述操作后，将显示“环境变量”对话框，在该对话框的系统变量列表中找到并选中“Path”变量。

成功选中 Path 变量后，需要进入变量编辑界面完成路径添加，这一步是环境变量配置的核心环节，路径的准确性直接影响配置的效果。

（二）添加 JDK 路径至 Path 变量

在“环境变量”对话框中点击“编辑”按钮，进入“编辑环境变量”对话框。在该对话框中点击“新建”按钮，此时会出现一个空白的路径输入框，在输入框中准确填入此前 JDK 安装目录下 bin 文件夹的完整路径“C:\Program Files\Java\jdk1.8.0_40\bin”。当路径输入完成后，点击对话框中的“确定”按钮，本次 Path 变量的编辑操作就完成了。

配置 Java 环境时，务必确认 JDK 位数（32 位 /64 位）与操作系统及后续要集成的 IDE 保持一致。若混用 64 位 JDK 与 32 位 IDE，可能导致 IDE 无法识别 JDK 或启动失败。此外，建议不要将 JDK 安装在含有中文或空格的路径中，推荐使用纯英文无空格的路径。配置 Path 时应直接复制 bin 目录完整路径，避免手动输入错误。配置完成后务必重启命令行窗口，否则新环境变量不会生效。

三、安装环境的测试

在开发环境搭建完成后，需通过特定操作验证其是否配置成功。这一环节主要通过系统命令行工具，检查核心组件 JDK 的安装状态与版本一致性，确保后续开发工作能正常启动。

（一）JDK 版本验证操作

版本验证是环境测试的核心步骤，需借助 Windows 系统自带的“命令提示符”工具执行具体命令，具体操作流程为：

打开“命令提示符”窗口，可通过 Windows 开始菜单搜索“cmd”或按下“Win+R”组合键输入“cmd”后回车启动。在弹出的命令行窗口中，输入命令 `java -version`（注意：java 与 -version 之间需保留一个空格），随后按下键盘上的“Enter”键。命令执行后，窗口将显示当前系统配置的 Java 版本信息。

（二）验证结果核对与补充测试

完成版本信息查询后，需对结果进行核对，同时可通过补充测试进一步确认环境稳定性，确保无潜在配置问题。

1. 结果核对

将命令行显示的 JDK 版本信息，与前期安装时选择的 JDK 版本进行逐一比对，确认版本号、位数（32 位或 64 位）等关键信息完全一致，若存在差异则需重新检查安装与配置步骤。

2. 补充测试

若版本核对无误，可进一步测试环境完整性。进入 JDK 安装目录下的“bin”文件夹，在命令行中执行该文件夹内的其他基础命令，观察命令是否能正常执行。

3. 结果判定

此时无须深入理解各命令的具体功能，只需关注命令执行后是否出现错误提示。若所有测试命令均能正常运行且无错误提示，即可判定前期开发环境的搭建与配置工作已顺利完成，环境可投入后续开发使用。



编程拓知

现代 Java 开发环境工具演进

传统 Java 开发依赖手动下载 JDK、配置 Path，过程烦琐且易出错。随着 DevOps 和多版本并行开发需求增长，开发者逐渐转向自动化环境管理工具。其中，SDKMAN! 和 jEnv 成为

主流选择。SDKMAN! 适用于 Linux/macOS，支持一键安装、切换多个 JDK 发行版，并通过 `sdk install java 21 - oracle` 等命令实现版本管理；jEnv 则专为 macOS 优化，可为不同项目自动绑定指定 JDK 版本。

在 Windows 平台上，Chocolatey 或 Scoop 包管理器也能简化 JDK 安装。更进一步，现代 IDE 已内置 JDK 探测与管理功能，可自动识别本地多个 JDK 并按项目设置使用。此外，容器化技术（Docker）让“环境即代码”成为可能，开发者可通过 Dockerfile 指定基础镜像，彻底规避本地环境差异问题。

这些工具不仅提升效率，还解决了“团队成员 JDK 版本不一致”导致的构建失败问题。对于企业级项目，推荐结合 CI/CD 流水线统一 JDK 版本，并在 `.tool-versions` 或 Dockerfile 中固化环境依赖，实现真正的“一次配置，处处一致”。

任务三 编写首个 Java 程序

任务描述

小王作为新入职的 Java 开发工程师助理，被导师安排完成一个“员工打卡信息展示”的入门练习任务。他需要使用记事本在本地编写一个名为 `EmployeeCheckIn.java` 的 Java 程序，要求定义工号和姓名两个字符串变量，并通过 `System.out.println()` 方法在控制台输出打卡成功提示及个人信息。但在保存文件后，小王在执行 `javac` 命令时遇到了“类名与文件名不一致”的编译错误；再次修改后又因忘记分号导致语法错误。最终，他成功运行程序，却对为何必须将 `public` 类名与文件名保持一致、`main` 方法为何格式固定等问题感到困惑。那么，要避免这些常见错误并理解 Java 程序的基本结构，小王需要掌握哪些核心知识点？

任务分析

小王需要掌握 Java 源文件命名规则——`public` 类名必须与文件名完全一致；掌握 `main` 方法的标准签名及其作为程序入口的唯一性；熟悉 Java 基本语法规则，如语句以英文分号结尾、严格区分大小写；了解编译与运行命令的正确使用方式；同时需遵循基础编程风格以提升代码可读性。这些是编写、编译和运行首个 Java 程序的基础保障。

任务实施

一、Java 源程序的编写

Java 源程序是开发 Java 应用的基础，其编写质量直接影响后续编译与运行效率。掌握规范的编

写方法、理解源程序的组成结构，是入门 Java 开发的核心步骤。以下将从基础编写流程入手，逐步拆解源程序的关键要素与实践规范。

Java 开发环境搭建并配置完成后，即可通过编写简单程序，系统掌握源程序的编写、编译及解释执行全流程。由于 JDK 未内置专属 Java 编辑器，开发者可选用记事本、UltraEdit 等文本编辑器，或 IntelliJ IDEA、Eclipse 等集成开发工具进行编辑。下文将以 Windows 系统自带的记事本为工具，演示第一个 Java 源程序的编写过程。

首先新建文本文件，在文件中输入如下代码：

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!!!");  
    }  
}
```

完成代码输入后，将文件以“HelloWorld.java”为名保存。

（一）Java 源文件

Java 源文件以“.java”作为固定扩展名，类（class）是源文件的核心组成部分。一个源文件中可以包含多个类定义，但最多只能有一个类使用 public 关键字修饰，且此时源文件的文件名必须与该 public 修饰的类名完全一致，包括字母的大小写状态。这一规则是 Java 编译器识别主类的重要依据，若违反将直接导致编译失败。

论码启思

为什么 Java 规定一个源文件中最多只能有一个 public 类，且文件名必须与该类名一致？这一设计对大型项目中的类管理、编译效率或模块化有何潜在影响？

（二）方法

一个 Java 类中可定义多个方法以实现不同功能，其中 main() 方法是 Java 程序的唯一执行入口，其格式固定且不可随意修改，标准格式如下：

```
public static void main(String[] args){...}
```

（三）Java 语法规范

Java 语言严格区分字母大小写，例如，关键字“String”与小写“string”代表完全不同的含义，前者是字符串类的标准名称，后者若未定义则会被识别为语法错误。同时，Java 语句以英文半角输入法下的分号“;”作为结束标志，缺少分号或使用全角分号都会导致编译报错。

编程锦囊

初学者常因使用中文标点导致编译失败。务必确保所有代码符号均使用英文半角输入法输入。建议在编辑器中开启“显示不可见字符”功能，便于快速识别隐藏的格式问题。此外，保存 .java 文件时应确认编码为 UTF-8，避免中文注释或字符串出现乱码。

（四）Java 注释

注释是代码的“说明文档”，合理的注释不仅能提升代码可读性，还能降低团队协作成本与后续维护难度，是专业开发的必备规范。Java 提供了多种注释方式以适应不同场景的需求。

为确保源程序的可读性、提升团队开发效率及增强系统可维护性，Java 开发者应编写简洁明了、含义准确的注释。Java 支持的注释标记主要有以下三种，见表 1-1 所列。

表 1-1 Java 支持的注释标记

类别	注释标记
单行注释	以 “//” 开头，注释内容仅作用于当前行，常适用于对局部代码的简短说明
多行注释	以 “/*” 开头、“*/” 结尾，可对多行内容进行注释，适用于对代码块的功能说明
文档注释	以 “/” 开头、“*/” 结尾，除实现多行注释功能外，还可通过 Javadoc 工具提取为正式文档，常适用于类、方法的说明

（五）编程风格

为增强程序的可读性与可维护性，Java 开发者应遵循统一的编程风格，其核心规范见表 1-2 所列。

表 1-2 Java 开发者应遵循的规范

类别	说明
缩进	采用 4 个空格作为标准缩进单位，通常可通过按一次 “Tab” 键实现，确保代码层次清晰
代码块标识	使用大括号 “{ }” 包裹代码块，多个代码块可嵌套使用。嵌套时需保证同一层次代码垂直对齐，内层代码相对外层代码缩进 4 个空格，符合视觉逻辑
空格使用	在关键字、变量、常量等操作数之间的二目或三目运算符前后，均需添加一个空格，如 “int a=10;” 而非 “int a=10;”，提升代码可读性
语句换行	建议一行仅编写一条语句，避免将多条短语句压缩在同一行，确保每条语句的独立性与可调试性

〔例 1-1〕

很多工作单位目前都实行打卡考勤制度来统计每个员工的上下班信息，请编写一个程序显示员工的个人打卡信息。

◆ 示例分析

可以利用前面介绍过的 System.out.println() 方法，将员工的个人打卡信息显示出来。

◆ 程序实现

```
public class Example1_1 {
    public static void main(String[ ] args) {
        String id="137";           // 设置工号
        String name=" 马 × 轩 ";    // 设置姓名
        System.out.println(" 早上好 ! 您已打卡成功 !");
        System.out.println(" 工号 :"+id);
        System.out.println(" 姓名 :"+name);
    }
}
```

◆ 结果呈现

早上好！您已打卡成功！

工号:137

姓名:马 × 轩

二、Java 程序的编译

编写完成的 Java 源程序（.java 文件）无法直接运行，需通过编译器转换为虚拟机可识别的字节码文件（.class 文件），这种转换过程即为编译。编译环节是连接源程序与运行结果的关键步骤，其操作流程与错误处理需要重点掌握。

Java 程序的编译需通过 JDK 提供的 javac 命令实现，具体操作步骤及注意事项如下。

（一）启动命令提示符

打开 Windows 系统的“命令提示符”窗口，通过“cd”命令切换至 HelloWorld.java 文件所在的目录。例如，若文件保存在“D:\JavaDemo”路径下，需输入“cd D:\JavaDemo”并按“Enter”键确认。

（二）执行编译命令

在目标目录下输入编译命令“javac HelloWorld.java”，按“Enter”键执行编译操作。

编译结果分为以下两种情况。

1. 编译成功

命令提示符窗口无错误提示，此时在源文件所在目录会生成一个名为“HelloWorld.class”的文件。该文件为 Java 字节码文件，是虚拟机可执行的文件格式，其文件名与源文件中 public 类名保持一致。

2. 编译失败

命令提示符窗口会显示具体错误信息，多数错误源于代码输入问题。例如，单词拼写错误、字母大小写不规范、缺少分号或标点符号使用全角等。此时需返回源文件检查并修正对应问题，重新执行编译命令。

（三）执行字节码文件

编译成功后，需通过 Java 命令执行字节码文件。在命令提示符窗口中继续输入“java HelloWorld”，按“Enter”键后，控制台将输出“Hello, World!!!”，表明程序运行成功。



编程拓知

Java 项目的演进路径

虽然使用记事本和 javac 命令能帮助初学者理解 Java 编译运行机制，但真实企业项目早已采用自动化构建工具。Maven 和 Gradle 不仅管理依赖，还规范了项目结构、编译流程与测试执行。例如，一个标准 Maven 项目会自动生成符合约定的目录，并通过 pom.xml 声明所需库版本，彻底避免手动配置 classpath 的烦琐与错误。此外，IDE 深度集成这些工具，使得开发者只需点击“Run”即可完成编译、打包与执行全过程。因此，在掌握基础命令行操作后，应尽快过渡到使用构建工具和 IDE，以贴近工业级开发实践。

任务四 掌握 Java 程序结构

任务描述

新入职一家互联网公司的小王，被安排参与一个内部工具类库的重构项目。项目负责人要求他将原先散落在无名包中的多个工具类按功能重新组织到合理的包结构中，并确保其他模块能正确引用这些工具类。小王在整理过程中发现部分类使用了 `java.util.Date` 和 `java.sql.Connection`，但不清楚这些类分别属于哪个标准包，也不清楚如何通过 `import` 语句或完整路径正确引用它们。此外，他还需要为新创建的工具包编写测试主类，其中包含 `main` 方法接收命令行参数以验证不同工具的功能。那么，小王要顺利完成这项任务，需要掌握哪些关于 Java 程序结构的核心知识点？

任务分析

小王需要理解 Java 包的作用与命名规范，能根据功能对类进行合理分包；掌握标准类包的用途及典型类；熟练使用 `package` 语句创建包，并通过 `import` 语句或全限定名引用其他包中的类；明确 `main` 方法的签名要求及其参数 `args[ ]` 的用途，能编写可接收命令行参数的入口程序。此外，还需理解无名包的局限性，避免在正式项目中使用。

任务实施

一、包

在 Java 开发体系中，包（Package）是类的松散集合形式，其核心作用是对各类文件进行系统化组织，功能类比于磁盘存储中的目录（文件夹）。无论是 Java 语言自带的标准类文件，还是开发者自主编写的类文件，均需归属到特定包结构中。包的管理机制为类提供了多层次命名空间，不仅有效规避了类的命名冲突问题，还解决了类文件的组织混乱难题，显著提升了类的可复用性与可维护性。

（一）Java 中常用的标准类包

JDK（java development kit）内置了大量实用类，这些类共同构成标准 API（application programming interface，应用程序编程接口）。为便于调用，这些类会依据功能属性分类封装至不同包中。随着 JDK 版本的迭代升级，标准类包的功能覆盖范围持续拓展，使用便捷性也不断提升。

Java 标准类均承载于标准包结构内，以下为开发中常用的标准类包及其功能说明。

1. java.lang 包

java.lang 包包含 Java 语言最基础的核心类，典型代表有 System、Math、String、Integer、Float 等。出于使用便捷性考虑，程序调用这些类时无须通过 import 语句导入，可直接使用。

2. java.awt 包

java.awt 包封装了用于构建图形化用户界面的基础类，如 Frame（窗口）、Button（按钮）、TextField（文本框）等。借助这些类，开发者能够构建满足需求的图形交互界面。

3. javax.swing 包

javax.swing 包是 java.awt 包的功能扩展，提供了更丰富、美观且功能强大的 GUI 组件，对应类包括 JFrame、JButton、JTextField 等。与 java.awt 包的组件相比，javax.swing 包的组件具备更高的灵活性与易用性。

4. java.util 包

java.util 包提供了多种实用工具类，功能涵盖系统特性定义、日期与日历相关操作、字符串解析等，是日常开发中高频使用的工具类集合。

5. java.io 包

java.io 包封装了数据流 I/O 操作的核心类，支持磁盘文件的创建、读取、写入等基础 I/O 操作，为数据持久化与传输提供支撑。

6. java.sql 包

java.sql 包提供了支持通过标准结构化查询语言（structured query language，SQL）方式访问数据库的类与接口，是 Java 程序实现数据持久层交互的基础。

（二）包的创建及包中类的引用

在实际开发中，若程序未明确指定包名，系统会默认将其归入无名包。无名包内的类可实现相互引用，但无法被其他包中的 Java 程序调用。对于简单测试程序而言，不特意指定包名可能不会产生明显影响；但在复杂应用开发场景中，若缺乏包对类的管理，将导致程序结构混乱，大幅增加开发与维护成本。

以下将简要介绍包的创建方法及使用规范。

1. 创建包

将编写完成的类按功能分类归入对应包中，以便在其他应用程序中复用，是面向对象程序设计的基本要求。开发者可通过 package 语句将类指定至特定包内，其通用语法格式如下：

```
package 包名；
```

需特别说明的是，package 语句必须置于源程序的起始位置（注解行与空行除外）。包名需符合 Java 标识符命名规则，开发习惯上通常采用全小写字母命名。包名支持多级结构，例如 Java 标准类包 java.util、java.sql 均采用此类命名方式。从物理存储角度而言，创建包本质上是在当前文件夹下创建以包名命名的子文件夹，并将类的字节码文件存入其中；若为多级包名，则以包名中的“.”作为文件夹分隔符，依次创建嵌套子文件夹。

2. 引用类包中的类

在程序的引用包中，类的方式通常分为以下两种。

第一种方式是通过 import 语句导入所需类，其通用语法格式如下：

```
import 包名.*; // 可以使用包中所有的类
```

或

```
import 包名.类名; // 只加载包中类名指定的类
```

程序中，import 语句需置于 package 语句之后；若程序中未包含 package 语句，则 import 语句应放在源程序开头。一个程序可包含多个 import 语句，以满足从不同包中引用类的需求。

第二种方式是在程序中直接通过“包名.类名”的完整路径引用所需类，其通用语法格式如下：

```
包名.类名
```

例如，通过以下语句可以直接使用 java.util 工具包中的 Date 类创建日期对象：

```
java.util.Date date1=new java.util.Date( );
```

上述语句通过指定类的完整包路径，调用系统提供的 Date 类完成了日期对象的实例化。

编程锦囊

在实际开发中，应避免使用无名包，即使对于简单测试也建议显式声明包名。无名包不仅无法被其他包引用，还可能导致 IDE 或构建工具识别异常。更重要的是，良好的包结构是团队协作和代码可维护性的基础。建议从项目初期就规划包层次，如按“com.公司名.项目名.模块名”规范命名。

二、主函数的参数

在 Java 应用程序中，main() 方法是程序的入口点，即程序运行时首先执行的方法。该方法与普通方法存在显著差异，具备严格的规范要求：方法名称必须为 main，访问修饰符与返回值类型必须为 public static void，且必须接收一个字符串数组作为参数。

main() 方法的字符串参数数组 args[]，核心作用是接收命令行输入的参数，各参数之间以空格作为分隔符。作为 Java 程序的特殊规范，main() 方法的限制条件较多，其运行原理需结合 Java 虚拟机（java virtual machine, JVM）相关知识才能深入理解。

论码启思

你正在开发一个简单的命令行工具，需要通过 args[] 接收搜索关键词、文件路径、是否区分大小写等参数，你会如何设计参数的格式，以确保程序能准确解析不同类型的输入？

三、代码块

代码块是 Java 中的基础概念，在实际开发中应用广泛。其定义为使用大括号“{ }”包裹的一段代码，根据所处位置及声明关键字的差异，可分为普通代码块、构造块、静态代码块与同步代码块四类。以下重点介绍前三种代码块。

（一）普通代码块

普通代码块是直接与方法或语句中定义的代码块，例如：

```
public class CodeDemo01
{
    public static void main(String[] args) {
        {int x=30;
        System.out.println (" 普通代码块 -->x="+x);
        }
        intx=100;
        System.out.println (" 代码块之外 -->x=" +x);}}}
```

运行结果如下：

```
普通代码块 -->x=30
代码块之外 -->x=100
```

（二）构造块

构造块是直接定义在类内部的代码块，不依赖于方法或其他结构，例如：

```
class daimakuai_1
{{
    System.out.println ("1. 构造块。");
    public daimakuai_1( )
    {System.out.println ("2. 构造方法 ");}}
public class CodeDemo02
{
    public static void main (String[] args)
    {
        new daimakuai_1( );
        new daimakuai_1( );
    }
}}
```

运行结果如下：

```
1. 构造块。
2. 构造方法
1. 构造块。
2. 构造方法
```

从运行结果可以得出核心结论：构造块的执行优先级高于构造方法，且每次实例化类对象时，构造块中的代码都会被执行一次。这一特性使得构造块适用于封装所有对象实例化时都需执行的通用逻辑。

（三）静态代码块

静态代码块是通过 `static` 关键字修饰的代码块，具备静态成员的特性，例如：

```

class Demo
{
    {System.out.println ("1. 构造块 ");}
    static {System.out.println ("0. 静态代码块 ");}
    public Demo( ) {System.out.println ("2. 构造方法 ");}
}

public class CodeDemo03
{
    static {System.out.println (" 在主方法所在类中定义的代码块。");}
    public static void main (String[ ] args) {
        new Demo( );
        new Demo( );new Demo( );}
}

```

运行结果如下：

在主方法所在类中定义的代码块。

0. 静态代码块

1. 构造块

2. 构造方法

1. 构造块

2. 构造方法

1. 构造块

2. 构造方法

从程序运行结果中可以发现以下三点：其一，静态代码块的执行优先级高于主方法；其二，类内部的静态代码块执行优先级高于构造块；其三，无论实例化多少对象，静态代码块仅在类加载时执行一次。基于这些特性，静态代码块常被用于初始化类的静态成员或执行类级别的通用初始化逻辑。



项目复盘

本项目围绕“初探 Java——环境配置与首程编写”展开，系统介绍 Java 入门的核心内容：首先梳理 Java 的发展脉络、核心特点、体系分类及程序工作机制；其次详解 Java 开发环境的搭建流程，包括 JDK 的下载安装、环境变量配置及环境测试方法；再次指导首个 Java 程序的编写与编译，涵盖源程序规范、编译运行步骤；最后剖析 Java 程序结构，讲解包的使用、主函数参数及代码块的应用。通过本项目的学习，读者可以夯实 Java 入门基础，掌握环境搭建与基础编程的核心技能，从而为后续深入学习 Java 开发奠定坚实基础。



项目实训

学习完本项目的内容后，请完成表 1-3 中实训信息的填写。

表 1-3 实训信息

任务名称	员工信息管理程序开发				
姓名		班级		学号	
学时			日期		
所需设备					
任务目的					

一、任务描述

开发一个简单的员工信息管理程序，需按规范构建代码结构并实现核心功能：首先声明 `com.xxx.employee` 包以组织代码，通过定义员工类封装姓名、工号、部门信息并实现输出逻辑；利用构造块初始化员工部门信息，借助静态代码块输出程序标题；主方法接收命令行参数作为员工入职时间，整合所有信息后在控制台完整输出；同时代码需严格遵循缩进、注释等编程规范，确保编译运行无错误，最终形成结构清晰、功能完整的员工信息管理程序。

二、分组讨论

全班学生以 3 ~ 6 人为一组，选出各自组长。组长对组员进行任务分工并将分工情况记录在表 1-4 中。

表 1-4 分工情况

小组成员	姓名	学号	任务分工
组长			
组员			

问题：

1. 如何通过包结构避免类命名冲突？在多人协作开发中包的设计原则是什么？
2. 主函数参数 `args` 的实际应用场景有哪些？如何处理多个命令行参数的解析？
3. 静态代码块与构造块的执行顺序对程序初始化有何影响？如何合理利用二者完成数据初始化？

三、实施过程

在实际操作过程中，将操作内容及实施过程中遇到的问题和解决办法记录在表 1-5 中。

表 1-5 问题与解决办法

序号	操作内容	遇到的问题	解决办法
1			
2			
3			

3. 静态代码块的执行特点是（ ）。

- A. 每次创建对象时执行
- B. 优先级低于构造方法
- C. 类加载时仅执行一次
- D. 需要手动调用才能执行

二、填空题

1. Java 程序的入口方法是_____，其参数类型为_____。
2. 在 JDK 安装目录中，存放编译工具 javac.exe 的文件夹是_____。
3. Java 中适用于导入其他包中类的关键字是_____。

三、操作题

1. 检查自己电脑上的 Java 开发环境是否配置成功，是否能正常运行 Java 命令。
2. 写一个输出“Hello Java”的简单 Java 程序，完成编译并运行，查看结果。
3. 手动配置 Java 的 Path 环境变量，确保 Java 命令在任意目录下都能执行。

项目二

数据基石——Java 类型与操作

项目全景

在 Java 开发体系中，类型与操作是构建程序的核心骨架，直接决定着数据的存储方式、运算逻辑及代码的健壮性。从金融系统的精准数值计算，到电商平台的用户数据处理，几乎所有业务场景都依赖对关键字、变量、数据类型及运算符的灵活运用。若基础环节出现疏漏，轻则导致程序报错，重则引发数据安全隐患。为帮助读者吃透 Java 类型与操作的底层逻辑，本项目将围绕解析关键字与标识符、管理变量与常量、转换数据类型、运用运算符与表达式等内容展开详细介绍。

学习目标

■ 知识目标

1. 掌握 Java 关键字的分类及使用禁忌，熟记标识符的语法规则与命名风格。
2. 理解变量与常量的本质区别，掌握其声明、赋值及作用域规则。
3. 熟悉 Java 基本数据类型的特性，掌握数据类型转换的条件与方法。

■ 能力目标

1. 能够根据业务需求合理定义变量、常量，规范使用标识符命名。
2. 能够灵活运用运算符构建表达式，实现数值计算与逻辑判断。
3. 能够识别并解决数据类型不匹配、运算溢出等常见问题。

■ 编程素养

1. 培养严谨的代码编写习惯，注重标识符命名的可读性与规范性。
2. 提升逻辑思维能力，能够拆解复杂运算需求并转化为代码实现。
3. 树立工程化思维，关注代码的可维护性与扩展性。

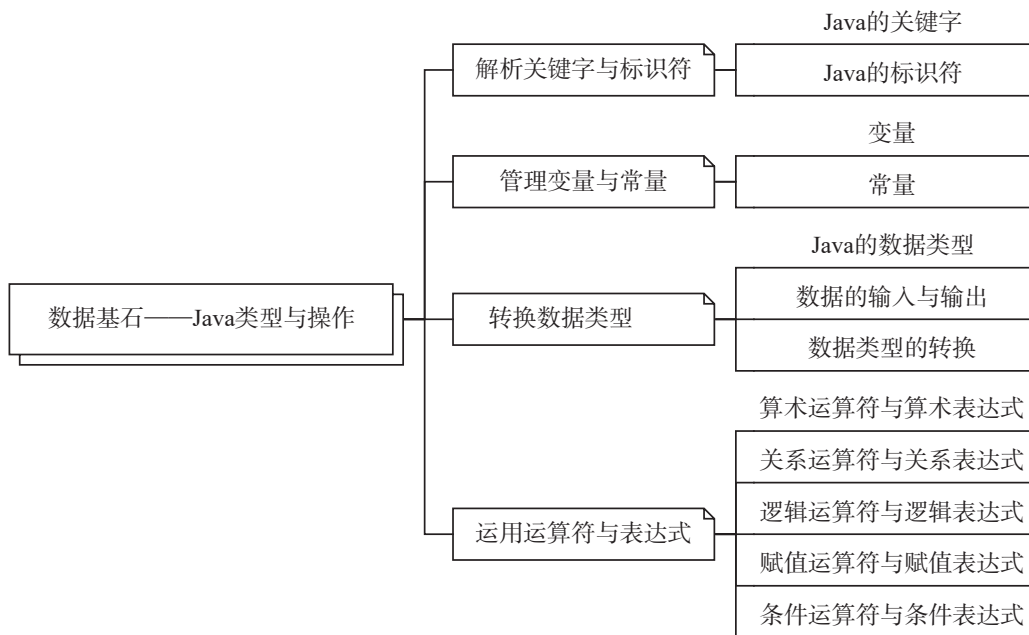


思政赋能

1. 强化规则意识，理解编程语言语法规则背后的严谨性与逻辑性。
2. 培养精益求精的工匠精神，追求代码的高效性与准确性。
3. 树立数据安全意识，认识到规范数据处理对系统稳定的重要性。



编程脑图



任务一 解析关键字与标识符



任务描述

小王作为新入职的 Java 开发工程师助理，正在参与公司内部一个用户信息管理模块的代码编写。他在定义变量时，尝试使用“class”作为变量名存储用户所属班级，结果编译报错；随后又用“User Name”命名一个字符串变量，再次失败。导师指出他违反了 Java 标识符的命名规则，并要求他重写代码，同时为包、类、方法和常量采用符合团队规范的命名风格。小王意识到自己对关键字与标识符的理解存在盲区，不清楚哪些词不能用、哪些字符合法，以及如何规范命名才能提升代码可读性。那么，要避免此类低级错误并写出专业级代码，小王需要系统学习哪些关于关键字与标识符的核心知识？

任务分析

小王需要掌握 Java 关键字的基本概念及其不可重用性，明确所有保留关键字不得用于自定义命名；理解标识符的构成规则——只能由字母、数字、下画线和美元符号组成，且首字符不能为数字；熟悉 Java 对大小写的敏感性及 Unicode 字符支持；更重要的是，需掌握行业通用的命名风格：包名全小写、类名用帕斯卡命名法、变量与方法用驼峰命名法、常量全大写加下画线。这些规范不仅是语法要求，更是团队协作和代码可维护性的基础。

任务实施

一、Java 的关键字

关键字是 Java 编程语言中由系统预先定义的特殊标识符，这类标识符在程序运行过程中被赋予特定功能与含义，用户在编写代码时不得将其用作自定义标识符。

Java 语言中的常用关键字见表 2-1 所列。

表 2-1 Java 中的常用关键字

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

编程拓知

为何 const 和 goto 在 Java 中“存在却不使用”

在 Java 关键字列表中，const 和 goto 虽被列为保留字，但从未在语言中实现具体功能。这是 Sun 公司在 Java 设计初期为保持语法一致性及预留未来扩展空间而采取的策略。例如，C/C++ 中的 const 用于定义常量，但 Java 选择了 final 来承担这一职责；而 goto 因易导致“面条式代码”而被摒弃，仅保留关键字形式以防止开发者误用为标识符。这种“保留但禁用”的做法体现了语言设计的前瞻性——既避免破坏现有代码兼容性，又为潜在语法演进留出通道。值得注意的是，随着 Java 持续引入新模式，未来仍可能激活某些保留字。开发者应了解这些“幽灵关键字”的存在意义，避免将其用于命名，同时关注 JEP（JDK enhancement proposal）中对关键字体系的调整动向。

二、Java 的标识符

在 Java 程序开发过程中，类名、变量名、方法名、类型名、数组名及文件名等元素均需通过特定符号进行标识，这类用于标识程序元素的符号被称为标识符。从本质上来说，标识符是程序中各类元素的专属命名标识。

（一）Java 标识符的语法规则

1. 构成规则

标识符由字母、数字、下画线（_）和美元符号（\$）组成，语言层面未对其长度设置上限。

2. 首字符限制

标识符的第一个字符不能是数字，仅可使用字母、下画线或美元符号开头。

3. 关键字禁用

自定义标识符不得使用 Java 语言中的关键字（关键字为系统预定义且具有特殊含义的标识符）。

4. 大小写敏感

Java 标识符严格区分大小写，例如，“Hello”与“hello”会被判定为两个不同的标识符。

Java 语言采用 Unicode 字符集进行字符编码，该字符集可支持最多 65 535 个字符的标识。因此，Java 标识符中允许使用的字符范围覆盖 Unicode 字符集的全部内容，包括拉丁文、中文汉字、日文假名及其他多种语言的字符。

例如，以下都是合法的字符。

age、_value、user_name、Hello、hello、\$salary、studentName、姓名、类 1

编程锦囊

虽然 Java 允许使用中文、日文等 Unicode 字符作为标识符，但在实际工程中应严格避免。非 ASCII 字符会降低代码可移植性，既可能导致部分编辑器或构建工具解析异常，也影响国际化团队协作。建议始终使用英文命名，确保代码清晰、通用且符合主流开发规范。

（二）标识符命名风格

为提升源程序的可读性与系统的长期可维护性，程序开发中除需保证标识符命名符合语法规则外，还应遵循以下行业通用的命名风格。

1. 语义化命名

标识符的命名需要尽可能体现其用途与含义，实现“见名知意”，便于开发者快速理解其功能。

2. 包名规范

包名全部由小写字母构成，不使用大写字母或特殊符号分隔。

3. 类名与接口名规范

采用“帕斯卡命名法”，即每个单词的首字母大写，其余字母小写。

4. 变量名与方法名规范

采用“驼峰命名法”，即第一个单词全小写，从第二个单词开始首字母大写。

5. 常量名规范

全部字母采用大写形式，单词之间通过下画线分隔。

论码启思

在多人协作项目中，如果一名开发者使用 \$ 开头的标识符命名业务变量，而另一名开发者坚持只用字母开头，哪种做法更合理？请从可读性、框架惯例、团队规范等角度展开讨论。

任务二 管理变量与常量

任务描述

小王作为新入职的 Java 开发工程师助理，被安排协助编写一个计算员工月度绩效奖金的小程序。他在代码中定义了一个变量名为“class”用于存储员工所属部门类别，结果编译报错；随后又尝试在方法外直接使用未声明的变量 score 进行计算，再次失败。导师指出他混淆了变量声明规则、关键字禁用原则以及作用域概念，并要求他改用 final 定义圆周率 PI 作为常量，同时确保所有变量在使用前正确初始化且命名规范。面对接连出错，小王意识到自己对 Java 中变量与常量的基础管理机制理解不足。那么，要写出合法、清晰且可维护的代码，小王需要系统掌握哪些关于变量与常量的核心知识？

任务分析

小王需要掌握 Java 变量与常量的基本管理规则。第一，理解变量必须先声明后使用，且声明需指定数据类型并以分号结尾；第二，严格遵守变量命名规范——不能使用关键字、首字符不能为数字、仅允许字母/数字/下划线/\$ 组成，并区分大小写；第三，明确变量作用域规则：块内定义的变量仅在该代码块内有效；第四，掌握常量通过 final 关键字定义，一旦赋值不可更改，且命名应采用全大写加下划线的风格；第五，了解初始化与赋值的区别，避免使用未初始化的局部变量。

任务实施

一、变量

在 Java 程序设计中，变量是指在程序运行过程中其存储的值能够被修改的量，它是程序中用于承载数据的基本单元。例如，a、a1、name 等符合标识符命名规则的符号，都可以作为合法的变量名。

（一）变量的声明与使用说明

1. 变量声明的前置要求

Java 语言对变量的使用有明确的语法约束，要求开发者在使用任何一个变量之前，必须先对该

变量的数据类型进行显式声明。这一规定确保了编译器能够准确识别变量的存储格式与运算规则，避免出现数据类型混乱的问题。

2. 变量声明语句的语法完整性

在 Java 语法体系中，对一个变量的声明本身就构成一条完整的 Java 语句。根据 Java 语句的语法规则，所有完整语句的结尾都必须使用分号（;）作为终止符，变量声明语句也不例外。

3. 变量名的命名规则

变量名作为标识符的一种，其命名需严格遵循以下规则，以确保语法合法且具备良好的可读性，见表 2-2 所列。

表 2-2 变量名的命名规则

规则	说明
首字符规则	变量名必须以字母开头，后续字符则可以是字母、数位的任意组合。这里的“字母”和“数位”并非仅指英文字母和阿拉伯数字，而是基于 Unicode 字符集的广义定义
字符范围定义	在 Java 语境中，“字母”涵盖了 Unicode 字符集中所有被判定为字母属性的字符，“数位”也包括 0～9 阿拉伯数字之外、Unicode 字符集中具有数位属性的其他字符，这使得变量名可以支持多语言字符
禁用字符与大小写特性	变量名中不允许包含加号（+）、版权符号（©）以及空格等特殊字符；同时，Java 变量名严格区分大小写，如“UserName”与“username”会被识别为两个完全不同的变量
长度限制说明	从语法层面来看，Java 语言并未对变量名的长度设置明确上限，实际开发中可根据业务需求与可读性要求灵活定义，但通常不建议使用过长的变量名
Unicode 字符判定方法	若需精确确认某个 Unicode 字符是否可作为变量名的首字符或后续字符，可借助 Java 核心类库中 Character 类提供的 isJavaIdentifierStart() 方法和 isJavaIdentifierPart() 方法进行校验
保留字禁用规则	变量名绝对不能使用 Java 语言的保留字（即关键字），这些保留字是 Java 系统预定义的、具有特定语法含义的标识符，如“class”“int”等均不可用作变量名

4. 多变量的批量声明方式

为简化代码编写，Java 允许在同一条语句中完成多个同类型变量的声明操作，多个变量名之间只需使用逗号（,）进行分隔，语句结尾仍以分号终止。例如，“int a,b,c;”就实现了对三个 int 类型变量 a、b、c 的批量声明。

（二）变量的赋值和初始化

在 Java 编程语言中，变量获取初始值的方式主要分为两种：赋值操作与初始化操作。这两种方式的核心实现载体均为赋值语句，二者的本质区别在于赋值语句的执行时机。

其中，赋值操作是指在变量完成声明之后，通过单独的赋值语句为其赋予具体数值；而初始化操作则是将赋值语句与变量声明语句合并执行，在声明变量的同时完成数值赋予，是一种“声明与赋值同步完成”的特殊赋值形式。

以下为两种方式的具体代码示例，两者在程序执行效果上完全等效。

1. 初始化方式（声明与赋值同步）

int a=10; 该语句同时完成变量 a 的声明与赋值，属于变量初始化过程

2. 单独赋值方式（声明与赋值分离）

int a; 第一步：仅声明 int 类型变量 a，此时变量暂未赋值
a=10; 第二步：通过独立赋值语句为变量 a 赋予数值 10

编程锦囊

局部变量在使用前必须显式初始化，否则编译器会报错；而类成员变量若未初始化，Java 会自动赋予默认值。但依赖默认值易引发逻辑错误，建议始终显式初始化所有变量，提升代码健壮性与可读性。

（三）变量的有效范围

变量的有效范围，在 Java 编程领域也被称为“变量的作用域”，其核心定义为：变量从被声明定义的位置开始，到其作用边界结束为止的这段代码区域，在该区域内变量能够被正常引用和使用。

变量的作用域具有明显的“定义位置关联性”——在程序中不同位置声明的变量，其作用域范围存在本质差异。从通用规则来看，在某一程序块内部声明定义的变量，其有效作用域通常仅限于该程序块内部。超出该程序块的代码区域，将无法直接引用该变量，若强行调用，则会导致编译失败。

论码启思

在多人协作项目中，有人主张将所有魔法数字全部替换为 final 常量，也有人认为局部临时使用的数字无须提取。你认为在什么场景下必须使用常量？什么情况下可以直接使用字面量？请结合可读性、维护成本和性能影响等展开讨论。

二、常量

常量是指在程序运行全过程中数值始终保持固定不变的量，其值一旦确定便无法被程序的后续执行逻辑修改，因此也常被称为“最终量”。标识常量的核心作用是通过一个特定的标识符来指代具体的常数值，这种定义方式能提升代码的可读性与维护性，其通用语法格式如下：

```
final 数据类型 常量名 =value[, 常量名 =value...];
```

在上述格式中，final 属于程序中的保留字，其作用是明确声明后续定义的标识符对应的是常量（即最终量），不可被二次修改；数据类型需指定该常量所属的具体类型，且必须为程序支持的基本数据类型之一；常量名是符合标识符命名规则的名称，用于唯一对应并代表常数值 value，在程序中所有需要使用该 value 的场景，均可以通过引用该常量名来替代直接书写数值。

例如，定义语句“final double PI=3.1415926;”中，便创建了一个名为 PI 的标识常量，该常量的数据类型为 double（双精度浮点型），其固定值设定为 3.141 592 6。



final 变量解析

在 Java 中，并非所有用 final 修饰的变量都是“编译时常量”。只有满足以下三个条件的 final 变量才被视为编译时常量：

- (1) 是基本数据类型或 String 类型；
- (2) 在声明时直接初始化；
- (3) 初始值是常量表达式。

例如，`final int MAX=100;` 是编译时常量，编译器会将其值内联到所有使用处，提升性能；而 `final int value=getValue( );` 则是运行时常量，仅保证不可变性，无法内联。这一区别影响着 switch 语句的 case 标签、类加载行为及 JIT 优化策略。理解这一点有助于开发者合理设计配置常量，避免误以为所有 final 变量都能参与编译期优化。

任务三 转换数据类型



任务描述

小王作为新入职的 Java 开发工程师助理，被安排编写一个简单的用户 BMI（身体质量指数）计算程序。他在代码中定义了身高为 int 类型、体重为 double 类型，并尝试将身高直接除以 100 后与体重运算。然而程序输出结果异常——BMI 值远高于预期。经导师检查，发现他未对 int 类型的身高进行显式转换，导致整数除法截断了小数部分。此外，他还误将布尔值“是/否”用 String 存储，而非 boolean 类型。面对这些问题，小王意识到自己对 Java 数据类型的选择与转换机制理解不足。那么，要正确处理不同类型数据的运算与赋值，小王需要掌握哪些关于数据类型及其转换的核心知识？



任务分析

小王需要掌握 Java 基本数据类型的分类与适用场景，明确 byte、short、int、long、float、double、char、boolean 各自的取值范围和内存占用；理解自动类型转换（隐式转换）的规则，即“小范围类型可自动转为大范围类型”，并牢记转换方向：byte/short/char → int → long → float → double；掌握强制类型转换的语法与风险，尤其是大类型转小类型可能导致的数据溢出或精度丢失；同时需了解浮点数精度问题，知道在高精度计算中应使用 BigDecimal；此外，还需正确选择数据类型，避免语义错误。

任务实施

一、Java 的数据类型

表 2-3 展示了一份个人基本资料表，表中包含了姓名、身高、体重、血型及职业属性等多类信息。这些数据从性质上存在明显差异，计算机语言为了实现对数据的高效处理与管理，会依据数据的本质特征进行分类，每一类具有相同本质特征的数据集合都被定义为一种“数据类型”。数据类型不仅明确了数据的核心性质与取值边界，还规定了数据在计算机内存中的存储格式，以及能够对该类数据执行的运算逻辑和操作方式，是程序设计中数据处理的基础依据。

表 2-3 个人基本资料

姓名	身高 /cm	体重 /kg	血型	是不是教师
王 × 磊	177	52.5	A	是
陈 × 设	162	78.5	O	是
李 ×	170	80.0	AB	否
孙 × 小	169	49.5	B	否

在 Java 语言的体系中，数据类型被清晰地划分为两大核心类别，即基本数据类型和引用数据类型。其中，基本数据类型包含八种具体类型，引用数据类型则包含五种具体类型，各类数据类型的层级关系与具体分类如图 2-1 所示。以下重点围绕使用频率最高的基本数据类型展开详细阐述。

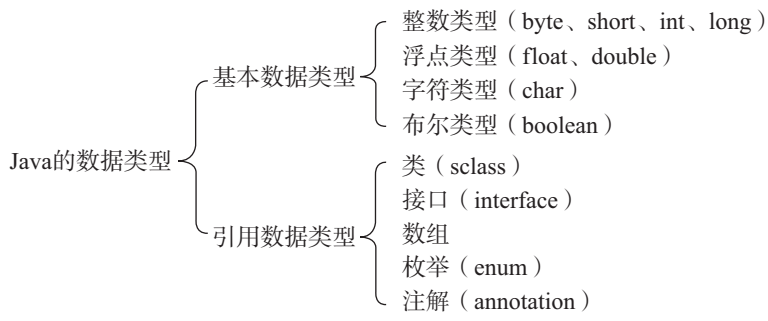


图 2-1 Java 的数据类型

(一) 整型

整型是用于表示无小数部分数值的数据类型，其取值范围覆盖正整数、负整数以及零，是程序开发中描述计数、编号等整数场景的核心类型。在 Java 程序中，整型数据通常以两种形态存在：用于表示固定数值的整型常量和用于存储可变化整数值整型变量。

1. 整型常量

整型常量是程序中直接出现的固定整数值，在 Java 中支持二进制、八进制、十进制和十六进制四种表示形式，每种形式都有明确的语法规则，以确保编译器能够准确识别，具体说明如下。

(1) 二进制表示

由数字 0 和 1 组成的连续数字序列，为与其他进制区分，必须以前缀“0b”或“0B”开头，如 0b01010001、0B01110111。

(2) 八进制表示

以数字 0 作为固定前缀，后续数字只能是 0 ~ 7（包含 0 和 7）范围内的整数，如 017、0236。

(3) 十进制表示

最符合人类日常使用习惯的表示形式，由 0 ~ 9（包含 0 和 9）的整数组成，无须特殊前缀，如 167、5698。

(4) 十六进制表示

以前缀“0x”或“0X”标识，后续字符可由 0 ~ 9 的数字以及 A ~ F（或 a ~ f，字母大小写不敏感）的字母组成，如 0X145、0x512。

2. 整型变量

整型变量是用于存储整数数据的内存单元，其核心特征由所占内存空间的大小决定。根据内存占用字节数的不同，Java 中的整型变量被细分为 byte（字节型）、short（短整型）、int（基本整型）和 long（长整型）四种具体类型，它们的内存占用量与取值范围存在明确差异，具体参数见表 2-4 所列。

表 2-4 整型

类型名	字节数	取值范围
byte	1	$-2^7 \sim 2^7-1$
short	2	$-2^{15} \sim 2^{15}-1$
int	4	$-2^{31} \sim 2^{31}-1$
long	8	$-2^{63} \sim 2^{63}-1$

四种整型变量的具体使用规范与特性说明如下。

(1) byte 型（字节型）

byte 型变量通过关键字“byte”进行定义，语法上支持一次性定义多个变量，可同时为部分或全部变量赋初始值，未赋值的变量会被赋予默认值 0。作为内存占用最小的整型类型，byte 型仅占用 1 个字节（8 位）内存，其取值范围限定为 -128 ~ 127。取值范围较窄，在使用 byte 型变量时需特别注意数据溢出问题——当赋值或运算结果超出该范围时，会导致程序运行错误，因此仅适用于存储小范围整数场景。示例代码如下：

```
byte x=25, y=-56, z; // 定义 byte 型变量 x、y、z，并赋初值给 x、y
```

(2) short 型（短整型）

short 型变量通过关键字“short”定义，系统为其分配 2 个字节（16 位）内存，取值范围扩展至 -32 768 ~ 32 767，相比 byte 型能存储更大范围的整数。但需注意，该类型的取值范围仍有限制，在处理较大整数时需避免数据溢出。short 型变量的定义语法与 byte 型变量类似。

(3) int 型（基本整型）

int 型是 Java 中使用得最广泛的整型类型，通过关键字“int”定义，占用 4 个字节（32 位）内存，取值范围为 -2 147 483 648 ~ 2 147 483 647，足以满足大多数整数处理场景的需求。无论是日常的计数、编号，还是中等规模的数值运算，int 型都是首选类型。其定义示例如下：

```
int x=345, y=-678, z;
```