Java 技术管理规范

本文档旨在为 Java 开发团队提供一套统一的编码和技术管理标准，以提高代码质量、可维护性和团队协作效率。规范分为三个级别：

- **强制**：必须无条件遵守的规则，违反这些规则的代码不应被合并到主干分支。
- **推荐**：强烈建议遵守的规则，有助于提升代码质量和可读性，但在特定场景下可酌情处理。
- **允许**：可以选择性采纳的风格或实践，团队可根据偏好和项目需求进行统一。

一、 强制

1.1 命名规范

1. **类名**：必须使用大驼峰命名法（UpperCamelCase）。例如：public class MyClass { ... }
2. **接口名**：必须使用大驼峰命名法（UpperCamelCase）。例如：public interface MyInterface { ... }
3. **方法名**：必须使用小驼峰命名法（lowerCamelCase）。例如：void myMethodName() { ... }
4. **常量名**：必须全部大写，单词间用下划线分隔（CONSTANT_CASE）。例如：static final int MAX_CONNECTIONS = 10;
5. **变量名**：必须使用小驼峰命名法（lowerCamelCase），包括局部变量、参数和非静态字段。
6. **包名**：必须全部小写，单词间用点分隔，通常基于组织域名反写。例如：com.google.common。
7. **泛型类型参数**：单个大写字母，如 T (Type), E (Element), K (Key), V (Value)。

1.2 代码格式

8. **大括号使用**：if, else, for, do, while 语句，即使只有一行代码，也必须使用大括号 {}。
9. **缩进**：每次缩进必须使用 4 个空格，严禁使用 Tab 字符。
10. **行长度**：每行代码长度原则上不超过 120 个字符。
11. **非空检查**：对于可能为 null 的对象，在调用其方法或访问其属性前，必须进行 null 值检查。
12. **异常处理**：try-catch 块中，catch 代码块不能为空。如果确实不需要处理，必

须添加注释说明原因。

13. **Switch 语句**：每个 switch 的 case 块都必须以 break, return, throw 或 continue 结束。如果需要贯穿 (fall-through)，必须有明确的注释说明。

14. **equals() 方法**：比较对象时，必须使用 equals() 方法，而不是 == 操作符（enum 和基本类型除外）。

1.3 注释规范

15. **JavaDoc**：所有 public 的类、接口和方法都必须有 JavaDoc 注释。

16. **废弃标记**：使用 @Deprecated 注解标记已废弃的方法或类，并提供替代方案的说明。

1.4 依赖管理

17. **禁止使用通配符导入**：严禁使用 import java.util.*; 这样的通配符导入，必须明确列出所有导入的类。

二、 推荐

2.1 编程实践

18. **接口导向编程**：优先面向接口编程，而不是具体实现类。例如：List<String> list = new ArrayList<>(); 而不是 ArrayList<String> list = new ArrayList<>();。

19. **使用 final**：对于不会被重新赋值的局部变量、方法参数和字段，推荐使用 final 关键字修饰。

20. **使用 @Override**：覆盖父类或实现接口的方法时，推荐总是使用 @Override 注解。

21. **避免魔法值 (Magic Numbers)**：代码中应避免直接使用未经定义的数字或字符串常量，推荐将其定义为常量。

22. **拆分长方法**：一个方法的代码行数推荐不超过 50 行，逻辑过长的方法应被拆分为多个更小的私有方法。

23. **使用 try-with-resources**：对于实现了 AutoCloseable 接口的资源（如流、连接等），推荐使用 try-with-resources 语句来确保资源被正确关闭。

24. **集合初始化**：推荐在创建集合时指定初始容量，以减少动态扩容带来的性能开销。例如：new ArrayList<>(expectedSize);。

25. **使用 Objects 工具类**：推荐使用 java.util.Objects 类中的方法（如 isNull, nonNull, equals, hashCode）来简化代码和避免 NullPointerException。

26. **日志级别**：根据日志信息的性质选择恰当的日志级别（DEBUG, INFO, WARN, ERROR）。
27. **字符串拼接**：在循环或大量拼接场景中，推荐使用 StringBuilder 或 StringBuffer，而不是 + 操作符。
28. **异常封装**：在捕获底层异常后，如果向上抛出，推荐将其封装在自定义的业务异常或更合适的标准异常中，避免直接暴露底层实现细节。
29. **使用 Optional**：对于可能返回 null 的方法，推荐返回 Optional<T>，以明确告知调用者需要处理缺失值的情况。
30. **Stream API**：对于集合的复杂处理，推荐使用 Stream API 来编写更具声明性和可读性的代码。
31. **工具类设计**：工具类（Helper/Util Class）的构造函数推荐私有化，并将其所有方法声明为 static。

2.2 代码格式与风格

32. **空行**：在方法之间、逻辑代码块之间推荐使用一个空行来增加代码的可读性。
33. **空格**：在二元操作符（如 +, -, =, >）两侧、for 循环的分号后、方法参数的逗号后，推荐使用空格。
34. **static 导入**：推荐对静态常量或静态方法使用静态导入（import static），以简化代码。

三、 允许

3.1 风格与实践选择

35. **Lambda 表达式**：在函数式接口的实现中，允许使用 Lambda 表达式来简化匿名内部类的写法。
36. **方法引用**：如果 Lambda 表达式只是调用一个已存在的方法，允许使用方法引用（Method Reference）。例如：list.forEach(System.out::println);。
37. **三元运算符**：对于简单的 if-else 赋值逻辑，允许使用三元条件运算符 (? :)，但应避免嵌套使用。
38. **链式调用**：对于遵循构建者模式（Builder Pattern）或 Fluent API 设计的类，允许使用链式方法调用。
39. **var 关键字**：在 Java 10+ 项目中，允许使用 var 关键字进行局部变量类型推断，前提是它能让代码更清晰易读，而不是更模糊。
40. **final 的位置**：修饰符的顺序可以遵循约定俗成的顺序，如 public static final。

41. **注释风格**：行内注释（//）和块注释（/* ... */）均允许使用，根据注释内容的多少和场景选择。
42. **this 关键字**：在构造函数或方法中，当参数名与字段名相同时，必须使用 this；在没有歧义的情况下，允许省略 this。
43. **Javadocs 格式**：允许在 Javadocs 中使用 HTML 标签来增强格式，如 <p>, <code>, 。
44. **switch 表达式**：在 Java 14+ 项目中，允许使用 switch 表达式来简化 switch 语句的写法。
45. **Record 类型**：在 Java 16+ 项目中，允许使用 record 关键字来定义不可变的数据载体类。
46. **assert 关键字**：允许在测试代码或内部逻辑中使用 assert 进行断言检查，但不能用于对外的参数校验。