**Java 编码风格文档**

**1. 代码结构**

**1.1 文件组织**

（1）每个源文件应只包含一个公共类或接口，且文件名与类名相同。

（2）包声明位于文件顶部，其次是导入语句，然后是类或接口声明。

（3）避免使用通配符导入 (import java.util.\*)，除非有明确需要。

**1.2 代码段顺序**

类的代码段组织顺序如下：

（1）静态常量（使用 static final 声明，所有字母大写，单词间用 \_ 分隔）。

（2）静态变量。

（3）实例变量。

（4）构造方法。

（5）公共方法。

（6）受保护的方法。

（7）私有方法。

|  |
| --- |
| 1. public class Example { 2. private static final int MAX\_VALUE = 100; *// 静态常量* 3. private int value; *// 实例变量* 4. public Example() { *// 构造方法* 5. this.value = 0; 6. } 7. public void setValue(int value) { *// 公共方法* 8. this.value = value; 9. } 10. private void helperMethod() { *// 私有方法* 11. System.out.println("Helper method"); 12. } 13. } |

**2. 命名规范**

**2.1 类和接口**

（1）类名使用 PascalCase（首字母大写）。

（2）接口名通常以形容行为的单词命名（如 Runnable）。

|  |
| --- |
| 1. public class UserAccount { } 2. public interface DataProcessor { } |

**2.2 方法和变量**

（1）方法名和变量名使用 camelCase（首字母小写）。

（2）常量名使用 UPPER\_SNAKE\_CASE。

|  |
| --- |
| 1. private int maxRetries; *// 变量* 2. public void processOrder() { } *// 方法* 3. private static final int MAX\_RETRIES = 5; *// 常量* |

**2.3 包名**

（1）全部小写，单词间用点分隔，通常以组织域名的反转形式开始。

|  |
| --- |
| 1. package com.example.myproject; |

**3. 代码格式**

**3.1 缩进**

（1）使用 4 个空格进行缩进，禁止使用 Tab。

**3.2 行长**

（1）每行代码不超过 100 个字符。

**3.3 花括号**

（1）左花括号 { 不换行，与声明保持同一行。右花括号 } 独占一行。

|  |
| --- |
| 1. if (condition) { 2. doSomething(); 3. } else { 4. doSomethingElse(); 5. } |

**3.4 空行**

（1）不同逻辑块之间使用一个空行分隔。

**4. 注释规范**

**4.1 类注释**

（1）使用 Javadoc 注释描述类功能。

|  |
| --- |
| 1. /\*\* 2. \* This class represents a user account in the system. 3. \*/ 4. public class UserAccount { } |

**4.2 方法注释**

（1）方法注释应说明功能、参数和返回值。

|  |
| --- |
| 1. */\*\** 2. \* Calculates the sum of two integers. 3. \* 4. \* @param a the first integer 5. \* @param b the second integer 6. \* @return the sum of a and b 7. \*/ 8. public int add(int a, int b) { 9. return a + b; 10. } |

**4.3 代码内注释**

（1）使用单行注释 // 对复杂逻辑进行解释。

|  |
| --- |
| 1. *// Validate user input before processing* 2. if (input.isValid()) { 3. processInput(input); 4. } |

**5. 异常处理**

（1）捕获特定异常，避免捕获通用异常 (Exception)。

（2）在捕获异常时记录日志，并提供有意义的错误信息。

|  |
| --- |
| 1. try { 2. performOperation(); 3. } catch (IOException e) { 4. logger.error("Error performing operation", e); 5. } |

**6. 测试**

（1）测试方法使用 test 开头，并包含测试目标的描述。

（2）每个测试方法应包含清晰的 Arrange-Act-Assert 三步。

|  |
| --- |
| 1. @Test 2. public void testAdd\_ShouldReturnCorrectSum() { 3. *// Arrange* 4. Calculator calculator = new Calculator(); 5. *// Act* 6. int result = calculator.add(2, 3); 7. *// Assert* 8. assertEquals(5, result); 9. } |

**7. 最佳实践**

（1）避免硬编码，将可配置的值放入配置文件或常量中。

（2）避免深层嵌套结构，通过分解方法和逻辑减少复杂度。

（3）优先使用 StringBuilder 拼接字符串，而不是使用 +。

|  |
| --- |
| 1. StringBuilder sb = new StringBuilder(); 2. sb.append("Hello").append(" World"); |

**8. 工具推荐**

使用 IDE 提供的代码格式化功能（如 IntelliJ IDEA）。