1 命名规范

* 1. 强制

包名必须全部小写，多层包名之间用句点分隔，如com.google.examples.project。包名应与其在文件系统中的位置一致，以便定位和组织代码。

类名、接口名采用大驼峰命名，即每个单词的首字母大写，其余字母小写，且不能使用下划线或其他分隔符。例如：MyClass、MyInterface。类名应清晰、准确地描述其功能和职责，避免使用模糊或易混淆的名称。

方法名、变量名、参数名采用小驼峰命名，即第一个单词首字母小写，后续单词首字母大写，如 myMethodName、myVariableName。方法名应体现其行为和功能，变量名应能够准确表示其存储的值的含义。

常量名全部大写，单词之间用下划线分隔，如MAX\_VALUE。常量一旦定义，其值不应被修改。

常量名应避免使用缩写，除非缩写在项目中已被广泛认知和使用。

* 1. 推荐

为项目中的包名添加公司或组织的前缀，以避免包名冲突。例如，阿里巴巴的包名常以com.alibaba开头。

类名和接口名的长度应尽量控制在合理范围内，避免过长或过短。一般建议类名不超过 20 个字符，接口名不超过 15 个字符，以便于阅读和理解。

方法名应尽量简洁明了，能够准确传达其功能。避免使用过于复杂的逻辑操作名称，对于较长的方法名，可以考虑将其拆分为多个更小的方法。

1.3 允许

在某些特定情况下，如为了与外部库或框架保持一致，可以适当调整命名风格，但应在团队内进行沟通和说明。

2代码格式

2.1 强制

每个类或接口的定义必须以package声明开头，且 package 声明应位于文件的第一行，后跟导入语句（如果有）。导入语句应按照 Java 的标准导入顺序进行排列，即java包、javax包、第三方包和项目内部包，每个部分内部按字母顺序排列。

代码块必须使用大括号{}包裹，即使代码块中只有一行代码，也不能省略大括号。大括号的左括号应与代码块的起始语句位于同一行，右括号应与起始语句对齐。

禁止使用制表符（Tab）进行缩进，统一使用 4 个空格进行缩进。这样可以确保代码在不同的编辑器和环境中具有一致的显示效果。

一行代码的最大长度不得超过 100 个字符。如果一行代码过长，应进行合理换行，换行时应使用 continuation indentation，即在新的行中使用比正常缩进多 4 个空格的缩进。

2.2 推荐

在代码块之间添加适当的空行，以增强代码的可读性，将逻辑上相关的代码段分隔开来。

对于较长的表达式或参数列表，可以考虑在每个参数或操作数之前添加一个空格，以提高代码的可读性。

在运算符（如+、-、\*、/等）周围添加适当的空格，以增强代码的可读性。

注释应与代码对齐，保持一致的缩进。例如，在方法声明的上方添加注释时，注释的起始位置应与方法声明的起始位置对齐；在代码块内添加注释时，注释的缩进应与代码块的缩进一致，以便于阅读和理解代码。

3 异常处理

3.1 强制

不得使用return语句或break语句来跳过异常处理。每个可能抛出受检异常的方法都必须显式地使用try-catch块或在方法签名中使用throws声明，以确保异常得到妥善处理。

禁止捕获Throwable或Exception类，而应仅捕获具体的异常类型。这样可以避免掩盖非预期的错误，并且能够更精确地处理不同的异常情况。

在捕获异常时，必须对异常进行适当的处理，如记录日志、释放资源、重试操作或转换为其他异常等。不得捕获异常后直接return或break，而不进行任何处理，导致异常被忽略。

最终的资源（如文件流、网络连接等）必须在try-with-resources语句中或finally块中正确关闭，以防止资源泄漏。

3.2 推荐

定义项目自己的异常层次结构，以便于对业务逻辑中的不同类型的异常进行分类和处理。

在抛出异常时，提供有意义的错误消息和相关上下文信息，以便于调试和问题定位。

对于可能被多个调用者处理的异常，可以考虑使用异常链，将原始异常作为新的异常的原因，以便于保留完整的错误堆栈信息。

异常处理的粒度应合理，避免过于粗放或过于细致。对于不同类型的异常，应根据其对程序的影响程度和处理方式进行分类处理。例如，对于可恢复的异常，可以尝试进行重试或采取其他补救措施；对于不可恢复的严重异常，应及时记录日志并通知相关人员进行处理，避免程序继续运行而产生不可预知的后果。

4 并发编程

4.1 强制

在多线程环境中，必须确保共享变量的正确性和可见性，通过使用适当的同步机制（如synchronized关键字、volatile变量或显式的锁对象）来避免线程安全问题。对于共享对象的读写操作，必须保证其原子性、可见性和有序性。

禁止使用过时的线程通信方式，如Thread类中的suspend和resume方法，而应使用现代的并发工具类（如java.util.concurrent包中的类）来实现线程间的通信和协作。

在使用线程池时，必须合理配置线程池的参数，如核心线程数、最大线程数、任务队列大小等，以避免线程池耗尽或任务积压等问题。同时，应确保线程池中的任务能够正确处理异常，并且在任务完成后释放相关资源。

线程池必须配置合理的拒绝策略，如CallerRunsPolicy、AbortPolicy等，以应对任务提交过多或线程池资源耗尽的情况。不同的拒绝策略适用于不同的业务场景，应根据实际情况选择合适的拒绝策略，确保程序在高并发情况下能够稳定运行，避免因任务积压或线程池不可用而导致系统崩溃。

4.2 推荐

优先使用不可变对象或线程安全的集合类（如ConcurrentHashMap、CopyOnWriteArrayList等）来避免线程安全问题，而不是手动进行同步操作。

在设计并发程序时，遵循“线程局部存储”、“锁分段”、“无锁编程”等设计原则，以提高程序的并发性能和可伸缩性。

对于复杂的并发场景，可以考虑使用高级的并发框架，如CompletableFuture、ExecutorService等，以简化并发编程的复杂性。

5 资源管理

5.1 强制

必须确保所有外部资源（如文件、数据库连接、网络连接等）在使用完毕后被正确关闭，以避免资源泄漏。可以使用try-with-resources语句或finally块来确保资源的关闭。

在使用数据库连接时，必须确保每个数据库操作都有对应的连接关闭操作。禁止将数据库连接作为全局变量或单例对象使用，以避免连接泄漏和资源耗尽。

对于有限的资源，如线程池、连接池等，必须进行合理的管理和监控，避免资源耗尽或过度使用。在资源不足时，应提供适当的错误处理和资源回收机制。

禁止重复分配资源，如多次打开同一个文件流或多次获取同一个数据库连接而未正确关闭。在分配资源之前，应先检查资源是否已经被分配，并且在使用完毕后及时关闭资源，以避免资源泄漏和资源竞争等问题，确保资源的有效利用和程序的稳定性。

5.2 推荐

使用资源池化技术来管理外部资源，以提高资源的利用率和性能。例如，使用数据库连接池、对象池等技术。

在使用文件资源时，尽量避免频繁打开和关闭文件，可以采用缓冲流或文件池等方式来提高文件读写的效率。

6 安全性

6.1 强制

禁止在日志或异常信息中直接输出敏感数据，如用户密码、信用卡号等。在记录日志或处理异常时，应对敏感数据进行脱敏处理或替换为占位符，以保护用户的隐私和数据安全，防止敏感信息泄露给未经授权的人员。

7 性能优化

7.1 推荐

根据预估大小合理设置集合类（如ArrayList、HashMap等）的初始容量，以减少集合在动态扩容时的性能开销。如果能够提前预估集合中元素的数量，设置合适的初始容量可以避免多次扩容操作，提高程序的运行效率。