## **1. 代码结构与组织**

**1.包命名  
a. 必须使用小写字母和域名反转的方式（如 com.example.project）。  
b. 推荐按功能模块划分包（如 com.example.project.service）。  
c. 允许使用单数形式命名包（如 util 而非 utils）。**

**2.类命名**

**a. 必须使用大驼峰命名法（如 UserService）。**

**b. 推荐使用名词或名词短语，避免使用动词（如 Configuration 而 非 Configure）。**

**c. 允许使用缩写（如 HTMLParser），但需确保广泛可理解。**

**3.接口命名  
a. 必须使用大驼峰命名法。**

**b. 推荐使用形容词（如 Serializable）或 I 前缀（如 IUserRepository）c. 允许单一方法接口使用 \*able 后缀（如 Runnable）。**

**4.方法命名**  
a. 必须使用小驼峰命名法（如 getUserName）。  
b. 推荐使用动词或动词短语（如 calculateTotalPrice）。  
c. 允许使用缩写（如 init() 代替 initialize()），但需保持一致性。

****5.常量命名****  
a. 必须使用全大写字母和下划线分隔（如 MAX\_CONNECTIONS）。  
b. 推荐定义在接口或类中，避免使用全局常量。  
c. 允许使用静态导入（static import）提高可读性。

****6.变量命名****  
a. 必须使用小驼峰命名法。  
b. 推荐使用有意义的名称（如 customerId 而非 cid）。  
c. 允许循环变量使用单字母（如 i, j）。

****7.类文件结构****  
a. 必须按照固定顺序组织：类注释 → 成员变量 → 构造方法 → 普通方法。  
b. 推荐将相关方法分组（如 getter/setter 放在一起）。  
c. 允许使用内部类，但需控制嵌套深度不超过 2 层。

****8.文件编码****  
a. 必须使用 UTF-8 编码。  
b. 推荐在 IDE 中统一配置文件编码。  
c. 允许使用 ASCII 编码的特殊场景需在注释中说明。

****9.源文件命名****  
a. 必须与公共类名完全一致（包括大小写）。  
b. 推荐每个文件只包含一个公共类。  
c. 允许内部类或私有类与主类放在同一文件中。

****10.注释规范****  
a. 必须为公共类、方法和字段添加 Javadoc 注释。  
b. 推荐使用 Markdown 格式编写复杂注释。  
c. 允许使用 TODO/FIXME 标记待处理事项，但需定期清理。

## **2. 代码风格**

****11.缩进与空格****  
a. 必须使用 4 个空格缩进，禁止使用 Tab。  
b. 推荐在二元运算符两侧添加空格（如 a + b）。  
c. 允许省略方法参数列表中的空格（如 method(a,b)）。

****12.行长度****  
a. 必须限制每行不超过 120 个字符。  
b. 推荐在运算符后换行（如 result = a + b +  
c + d;）。  
c. 允许长字符串不换行，但需使用字符串连接符（+）。

****13.空行分隔****  
a. 必须在类成员之间使用空行分隔（如变量与方法之间）。  
b. 推荐在方法内部逻辑块之间使用空行分隔。  
c. 允许连续多个空行，但需保持代码整洁。

****14.括号使用****  
a. 必须在条件语句和循环语句中使用括号（如 if (condition) { ... }）。  
b. 推荐在复杂表达式中使用括号明确优先级（如 (a + b) \* c）。  
c. 允许省略单语句块的括号（如 if (a) return b;），但需保持一致性。

****15.访问修饰符****  
a. 必须明确指定访问修饰符（如 private, protected, public）。  
b. 推荐优先使用最小访问权限（如 private 而非 protected）。  
c. 允许省略默认访问修饰符（package-private），但需确保符合设计意图。

****16.枚举类****  
a. 必须使用大驼峰命名法。  
b. 推荐为枚举值添加描述属性（如 ACTIVE("Active Status")）。  
c. 允许在枚举中定义方法和构造函数。

****17.Lambda 表达式****  
a. 必须在参数列表中省略类型（如 (a, b) -> a + b）。  
b. 推荐使用简洁语法（如 a -> a \* 2）。  
c. 允许在复杂逻辑中使用大括号和返回语句。

****18.Stream API****  
a. 必须避免过长的 Stream 链式调用（超过 5 个操作）。  
b. 推荐使用并行流（parallelStream()）时注意线程安全。  
c. 允许使用 Stream 处理集合操作，但需确保性能合理。

****19.异常处理****  
a. 必须捕获并处理受检异常（Checked Exception）。  
b. 推荐自定义业务异常类（如 UserNotFoundException）。  
c. 允许使用 try-with-resources 语句管理资源。

****20.泛型使用****  
a. 必须使用类型通配符（如 List<? extends Number>）。  
b. 推荐在方法签名中使用泛型（如 public <T> T getValue()）。  
c. 允许使用原始类型（Raw Type），但需添加 @SuppressWarnings 注解。

## **3. 性能与安全**

****21.字符串操作****  
a. 必须在循环中使用 StringBuilder 代替 + 操作符。  
b. 推荐在单线程环境下使用 StringBuilder，多线程使用 StringBuffer。  
c. 允许在简单场景下直接使用字符串连接（如 String message = "Hello " + name;）。

****22.集合操作****  
a. 必须在初始化集合时指定初始容量（如 new ArrayList<>(100)）。  
b. 推荐使用 Arrays.asList() 创建固定大小的列表。  
c. 允许使用 Collections.unmodifiable\* 方法创建不可变集合。

****23.单例模式****  
a. 必须使用枚举实现单例（如 public enum Singleton { INSTANCE; }）。  
b. 推荐使用双重检查锁定（Double-Checked Locking）实现懒加载。  
c. 允许使用静态内部类实现单例（如 Bill Pugh 方法）。

****24.线程安全****  
a. 必须在多线程环境中使用线程安全的集合（如 ConcurrentHashMap）。  
b. 推荐使用 java.util.concurrent 包中的工具类（如 ExecutorService）。  
c. 允许使用 synchronized 关键字，但需避免过度同步。

****25.序列化****  
a. 必须为可序列化类添加 serialVersionUID。  
b. 推荐实现 writeObject() 和 readObject() 方法控制序列化过程。  
c. 允许使用 Externalizable 接口替代 Serializable。

****26.日志使用****  
a. 必须使用 SLF4J 作为日志门面，Logback 或 Log4j 作为实现。  
b. 推荐使用参数化日志（如 log.info("User {} logged in", username)）。  
c. 允许在测试环境使用 System.out.println，但禁止在生产代码中使用。

****27.资源管理****  
a. 必须关闭所有打开的资源（如文件、网络连接）。  
b. 推荐使用 try-with-resources 语句自动关闭资源。  
c. 允许手动调用 close() 方法，但需在 finally 块中执行。

****28.反射使用****  
a. 必须谨慎使用反射，避免破坏封装性。  
b. 推荐在框架开发中使用反射，业务代码中尽量避免。  
c. 允许使用反射访问私有成员，但需添加 @SuppressWarnings("unchecked") 注解。

****29.SQL 注入防护****  
a. 必须使用预编译语句（PreparedStatement）执行 SQL。  
b. 推荐使用 ORM 框架（如 Hibernate、MyBatis）避免手动拼接 SQL。  
c. 允许在特殊场景下使用原生 SQL，但需严格过滤用户输入。

****30.敏感信息处理****  
a. 必须加密存储密码等敏感信息（如使用 BCrypt）。  
b. 推荐在日志中屏蔽敏感信息（如使用脱敏工具）。  
c. 允许在开发环境使用明文配置，但需在生产环境加密。

## **4. 测试与工具**

****31.单元测试****  
a. 必须使用 JUnit 5 编写单元测试。  
b. 推荐使用 Mockito 模拟依赖对象。  
c. 允许使用 TestNG 等其他测试框架。

****32.代码覆盖率****  
a. 必须确保核心业务代码覆盖率达到 80% 以上。  
b. 推荐使用 JaCoCo 生成覆盖率报告。  
c. 允许忽略测试难以覆盖的代码（如私有方法）。

****33.静态代码分析****  
a. 必须集成 SonarQube 进行代码质量检查。  
b. 推荐使用 Checkstyle、PMD 和 FindBugs 进行额外检查。  
c. 允许在 CI/CD 流程中自动触发静态分析。

****34.依赖管理****  
a. 必须使用 Maven 或 Gradle 管理依赖。  
b. 推荐明确指定依赖版本（如 <version>1.2.3</version>）。  
c. 允许使用依赖范围（如 provided、test）控制传递依赖。

****35.版本控制****  
a. 必须使用 Git 作为版本控制系统。  
b. 推荐遵循 GitFlow 或 GitHub Flow 工作流程。  
c. 允许使用 SVN 等其他版本控制系统，但需符合团队规范。

****36.构建工具****  
a. 必须使用 Maven 或 Gradle 构建项目。  
b. 推荐配置自动化构建脚本（如 mvn clean install）。  
c. 允许使用 Ant 等其他构建工具，但需保持一致性。

****37.代码格式化****  
a. 必须使用统一的代码格式化配置（如 Google Java Format）。  
b. 推荐在 IDE 中自动应用格式化规则。  
c. 允许手动格式化代码，但需定期检查一致性。

****38.注释规范****  
a. 必须为公共 API 添加 Javadoc 注释。  
b. 推荐使用 @param、@return 和 @throws 标签完善注释。  
c. 允许使用 TODO/FIXME 标记待处理事项，但需定期清理。

****39.代码审查****  
a. 必须通过代码审查（Code Review）才能合并代码。  
b. 推荐使用 GitHub Pull Request 或 GitLab Merge Request 进行审查。  
c. 允许使用工具自动化检查常见问题（如 SonarLint）。

****40.文档维护****  
a. 必须更新代码变更对应的文档（如 README、Wiki）。  
b. 推荐使用 AsciiDoc 或 Markdown 编写技术文档。  
c. 允许使用 Swagger 等工具自动生成 API 文档。

## **5. 最佳实践**

****41.设计模式****  
a. 必须合理使用设计模式（如单例、工厂、观察者）。  
b. 推荐避免过度设计，优先使用简单方案。  
c. 允许在复杂场景下使用高级模式（如责任链、策略）。

****42.不可变对象****  
a. 必须创建不可变对象（如 String、Integer）以提高线程安全性。  
b. 推荐使用 final 关键字修饰类和字段。  
c. 允许在必要时创建可变对象，但需明确文档说明。

****43.依赖注入****  
a. 必须使用依赖注入（如 Spring 框架）管理组件。  
b. 推荐通过构造函数注入依赖。  
c. 允许使用 setter 方法注入可选依赖。

****44.性能优化****  
a. 必须避免创建不必要的对象（如在循环中重复创建相同对象）。  
b. 推荐使用基本数据类型（如 int）代替包装类（如 Integer）。  
c. 允许在性能关键代码中使用原生方法（如 System.arraycopy()）。

****45.异常处理****  
a. 必须捕获特定异常而非通用异常（如 catch (IOException e) 而非 catch (Exception e)）。  
b. 推荐记录异常堆栈信息（如 logger.error("Error occurred", e)）。  
c. 允许在必要时抛出运行时异常（如 IllegalArgumentException）。

****46.集合操作****  
a. 必须使用增强 for 循环或 Stream API 遍历集合。  
b. 推荐使用 for-each 循环代替传统 for 循环。  
c. 允许使用迭代器（Iterator）在需要修改集合时遍历。

****47.字符串处理****  
a. 必须使用 equals() 方法比较字符串内容。  
b. 推荐使用 equalsIgnoreCase() 进行不区分大小写的比较。  
c. 允许使用 == 比较字符串常量（如 "abc" == "abc"）。

****48.日期与时间****  
a. 必须使用 Java 8 的 java.time 包（如 LocalDateTime）代替 Date 和 Calendar。  
b. 推荐使用 DateTimeFormatter 进行日期格式化。  
c. 允许在遗留代码中使用旧的日期 API，但需避免新增使用。

****49.断言使用****  
a. 必须使用断言（assert）验证内部不变量。  
b. 推荐在开发和测试环境启用断言（-ea 参数）。  
c. 允许在生产环境禁用断言以提高性能。

****50.代码复用****  
a. 必须避免代码重复，提取公共逻辑到工具类或基类。  
b. 推荐使用组合（Composition）而非继承。  
c. 允许在必要时使用继承，但需控制层次深度不超过 3 层。