# Java技术管理规范

## 1. 编码规范

### 1.1 命名规则

#### 1.1.1 类名

## 强制：类名首字母大写，采用驼峰命名法，如CustomerManager。类名应清晰表达其功能和用途，避免使用缩写，除非该缩写是广泛认可的，如HTTP。

## 1.1.2 方法名

## 强制：方法名首字母小写，采用驼峰命名法，如getCustomerName。方法名应准确描述其行为，避免模糊不清的命名，如doSomething。

## 1.1.3 变量名

## 强制：变量名首字母小写，采用驼峰命名法，如customerName。局部变量名应简洁明了，类成员变量应以m开头，如mCustomerName。

## 1.1.4 常量名

## 强制：常量名全大写，单词间用下划线分隔，如MAX\_SIZE。常量应在类中集中定义，避免在代码中硬编码。

## 1.2 代码格式

## 1.2.1 缩进

## 强制：使用4个空格进行缩进，禁止使用制表符。保持代码的整齐和一致性，便于阅读。

## 1.2.2 大括号

## 强制：大括号前不换行，大括号后换行。如if (condition) {，}单独一行。

## 1.2.3 空行

## 推荐：方法之间、逻辑块之间添加空行，增强代码的可读性。避免过多空行，以免浪费空间。

## 1.2.4 空格

## 推荐：在操作符两侧添加空格，如a = b + c。在括号外添加空格，如if (condition) {。

## 1.3 注释规范

## 1.3.1 类注释

## 强制：每个类必须有注释，描述类的功能、用途和主要方法。注释应简洁明了，避免冗长。

## 1.3.2 方法注释

## 强制：每个公共方法必须有注释，描述方法的功能、参数、返回值和异常。注释应详细准确。

## 1.3.3 代码注释

## 推荐：复杂逻辑代码块应添加注释，解释代码的目的和实现方式。避免在简单代码上添加不必要的注释。

## 1.4 编码实践

## 1.4.1 单一职责原则

## 强制：类和方法应遵循单一职责原则，避免一个类或方法承担过多职责。每个类和方法应专注于一个功能。

## 1.4.2 代码复用

## 推荐：尽量复用代码，避免重复代码。通过提取公共方法或类来提高代码的可维护性。

## 1.4.3 异常处理

## 强制：合理使用异常处理机制，避免捕获通用异常Exception。应捕获具体的异常类型，并在必要时抛出自定义异常。

## 1.4.4 日志记录

## 推荐：在关键位置添加日志记录，便于问题排查和系统监控。日志应包含足够的信息，如时间、日志级别、消息等。

## 2. 代码审查

## 2.1 审查流程

## 2.1.1 提交代码

## 强制：开发人员完成代码后，提交代码至代码仓库，并发起代码审查请求。提交的代码应经过本地测试，确保无明显错误。

## 2.1.2 审查人

## 强制：代码审查由至少1名其他开发人员完成，审查人应具备相关技术能力。审查人应认真审查代码，提出问题和改进建议。

## 2.1.3 审查结果

## 强制：审查结果分为通过、不通过两种。不通过的代码，开发人员需根据审查意见修改后重新提交。

## 2.2 审查内容

## 2.2.1 代码规范

## 强制：审查代码是否符合编码规范，包括命名、格式、注释等。不符合规范的代码应要求修改。

## 2.2.2 代码质量

## 强制：审查代码的可读性、可维护性和性能。避免复杂逻辑和低效代码，确保代码质量。

## 2.2.3 功能实现

## 强制：审查代码是否实现了预期功能，是否存在逻辑错误。通过测试用例验证功能的正确性。

## 2.2.4 安全性

## 强制：审查代码是否存在安全漏洞，如SQL注入、跨站脚本攻击等。确保代码的安全性。

## 3. 版本管理

## 3.1 分支策略

## 3.1.1 主分支

## 强制：主分支（如master或main）用于存放稳定版本的代码。只有经过充分测试的代码才能合并到主分支。

## 3.1.2 开发分支

## 强制：开发分支（如develop）用于日常开发。开发人员在开发分支上进行代码开发，定期合并到主分支。

## 3.1.3 特性分支

## 推荐：为每个新特性创建独立的特性分支（如feature-x）。特性开发完成后，合并到开发分支。

## 3.1.4 修复分支

## 推荐：为每个修复任务创建独立的修复分支（如fix-y）。修复完成后，合并到主分支和开发分支。

## 3.2 提交规范

## 3.2.1 提交信息

## 强制：提交信息应清晰、简洁，描述提交的内容和目的。遵循一定的格式，如[模块]：描述。

## 3.2.2 提交频率

## 推荐：定期提交代码，避免长时间不提交。每次提交应包含一个完整的功能或修复。

## 3.2.3 提交内容

## 强制：提交的代码应经过本地测试，确保无编译错误和明显逻辑错误。避免提交无关的文件，如临时文件。

## 3.3 合并规范

## 3.3.1 合并条件

## 强制：只有经过代码审查通过的代码才能合并到主分支。合并前应确保代码与主分支的兼容性。

## 3.3.2 合并方式

## 推荐：优先使用rebase合并，保持提交历史的清晰。在必要时可以使用merge合并。

## 3.3.3 合并记录

## 强制：合并操作应记录详细信息，包括合并人、合并时间等。便于后续的版本追溯。

## 4. 测试规范

## 4.1 单元测试

## 4.1.1 测试覆盖率

## 强制：单元测试覆盖率应达到80%以上。重点测试核心功能和复杂逻辑。

## 4.1.2 测试用例

## 强制：每个方法应有对应的单元测试用例。测试用例应覆盖正常情况和异常情况。

## 4.1.3 测试框架

## 推荐：使用JUnit等成熟的单元测试框架。测试框架应支持断言、Mock等功能。

## 4.2 集成测试

## 4.2.1 测试范围

## 强制：集成测试应覆盖系统的主要功能模块。确保模块之间的交互正确无误。

## 4.2.2 测试环境

## 推荐：使用与生产环境类似的测试环境。测试环境应包含完整的数据和配置。

## 4.2.3 测试方法

## 推荐：采用黑盒测试和白盒测试相结合的方法。黑盒测试关注功能实现，白盒测试关注内部逻辑。

## 4.3 性能测试

## 4.3.1 性能指标

## 强制：明确性能测试的指标，如响应时间、吞吐量等。根据业务需求设定合理的性能指标。

## 4.3.2 测试工具

## 推荐：使用JMeter等性能测试工具。测试工具应支持多种协议和场景。

## 4.3.3 性能优化

## 强制：根据性能测试结果进行优化。优化应从代码、数据库、服务器等多方面入手。

## 4.4 安全测试

## 4.4.1 安全漏洞

## 强制：重点测试常见的安全漏洞，如SQL注入、跨站脚本攻击等。确保系统不存在安全漏洞。

## 4.4.2 安全工具

## 推荐：使用OWASP等安全测试工具。安全工具应支持多种安全漏洞检测。

## 4.4.3 安全策略

## 强制：制定安全策略，明确安全测试的流程和标准。安全策略应涵盖开发、测试、上线等各个环节。

## 5. 代码优化

## 5.1 代码效率

## 5.1.1 避免冗余代码

## 强制：避免重复代码，通过提取公共方法或类来提高代码复用性。减少代码冗余，提高代码可维护性。

## 5.1.2 使用合适的数据结构

## 推荐：根据实际需求选择合适的数据结构，如ArrayList、LinkedList等。避免使用不恰当的数据结构导致性能问题。

## 5.1.3 减少不必要的计算

## 强制：避免在循环中进行不必要的计算，将计算结果缓存起来复用。减少计算量，提高代码执行效率。

## 5.2 代码可读性

## 5.2.1 代码简洁性

## 推荐：保持代码简洁，避免过于复杂的逻辑。通过合理的分解和封装，使代码更易于理解。

## 5.2.2 代码一致性

## 强制：保持代码风格的一致性，遵循统一的编码规范。避免在同一项目中出现多种不同的代码风格。

## 5.2.3 代码注释

## 推荐：在复杂逻辑处添加必要的注释，解释代码的目的和实现方式。注释应简洁明了，避免冗长。

## 5.3 代码安全性

## 5.3.1 输入验证

## 强制：对所有输入进行严格的验证，防止非法输入导致的安全问题。如对用户输入的字符串进行长度和格式验证。

## 5.3.2 权限控制

## 推荐：合理设置代码的访问权限，避免不必要的权限暴露。如将类成员变量设置为私有，并通过公共方法进行访问。

## 5.3.3 敏感信息处理

## 强制：对敏感信息进行加密处理，避免明文存储或传输。如对用户的密码进行加密存储。

## 6. 文档规范

## 6.1 开发文档

## 6.1.1 系统架构文档

## 强制：编写系统架构文档，描述系统的整体架构和各模块之间的关系。架构文档应包含架构图和详细的文字说明。

## 6.1.2 接口文档

## 强制：编写接口文档，描述系统的对外接口和内部模块之间的接口。接口文档应包含接口的名称、参数、返回值和调用方式。

## 6.1.3 数据库设计文档

## 强制：编写数据库设计文档，描述数据库的表结构和关系。数据库设计文档应包含数据库表的创建语句和数据字典。

## 6.2 测试文档

## 6.2.1 测试计划

## 强制：编写测试计划，描述测试的范围、方法、资源和时间安排。测试计划应明确测试的目标和策略。

## 6.2.2 测试用例

## 强制：编写详细的测试用例，描述测试的步骤、输入、预期结果和实际结果。测试用例应覆盖系统的各种功能和场景。

## 6.2.3 测试报告

## 强制：编写测试报告，描述测试的结果、发现的问题和建议。测试报告应包含测试的总结和改进建议。

## 7. 团队协作

## 7.1 沟通协作

## 7.1.1 定期会议

## 推荐：定期召开项目会议，沟通项目进度和问题。会议应有明确的议程和记录，确保团队成员之间的信息同步。

## 7.1.2 即时通讯

## 推荐：使用即时通讯工具进行日常沟通，及时解决问题。即时通讯工具应支持文件共享和屏幕共享功能。

## 7.1.3 代码共享

## 强制：通过代码仓库共享代码，确保团队成员能够及时获取最新的代码。代码仓库应支持版本控制和代码审查功能。

## 7.2 知识共享

## 7.2.1 技术分享

## 推荐：定期组织技术分享会，分享最新的技术知识和经验。技术分享会应有明确的主题和内容。

## 7.2.2 文档共享

## 强制：将项目相关的文档共享给团队成员，确保团队成员能够随时查阅。文档共享应支持版本控制和权限管理。

## 7.2.3 代码复审

## 推荐：定期进行代码复审，发现和改进代码中的问题。代码复审应有明确的规则和流程。

## 8. 项目管理

## 8.1 项目规划

## 8.1.1 项目目标

## 强制：明确项目的最终目标和阶段性目标。项目目标应具体、可衡量、可实现、相关联和有时限。

## 8.1.2 项目计划

## 强制：制定详细的项目计划，包括任务分解、时间安排和资源分配。项目计划应明确每个任务的负责人和完成时间。

## 8.1.3 风险评估

## 推荐：对项目可能面临的风险进行评估，并制定相应的应对措施。风险评估应涵盖技术风险、人员风险和市场风险等。

## 8.2 项目监控

## 8.2.1 进度跟踪

## 强制：定期跟踪项目的进度，及时发现和解决问题。进度跟踪应有明确的指标和方法。

## 8.2.2 质量监控

## 强制：对项目的质量进行监控，确保项目符合质量标准。质量监控应涵盖代码质量、测试质量和文档质量等。

## 8.2.3 成本控制

## 推荐：对项目的成本进行控制，确保项目在预算范围内完成。成本控制应涵盖人力成本、设备成本和材料成本等。

## 8.3 项目收尾

## 8.3.1 项目验收

## 强制：按照项目验收标准进行项目验收，确保项目达到预期目标。项目验收应有明确的验收标准和流程。

## 8.3.2 项目总结

## 推荐：对项目进行总结，总结项目的经验教训和改进建议。项目总结应涵盖项目的技术总结、管理总结和团队总结等。

## 8.3.3 知识移交

## 强制：将项目相关的知识和文档移交给相关人员，确保项目的顺利交接。知识移交应有明确的移交清单和确认流程。