
How to Become a
Good
Java
Developer

INDEX

Become a Good Developer

Developer Guidelines

- **Use SonarLint:** Integrate and utilize SonarLint in your IDE (e.g., Eclipse) for continuous static code analysis and quality checks.
- **Adhere to Standards:** Follow industry-standard coding conventions for Java and any other technologies in use. Maintain consistency across the project.
- **Code Formatting & Cleanup:**
  - Format code before committing (Ctrl+Shift+F in Eclipse).
  - Remove unused imports, variables, comments, and redundant code.
  - Eliminate auto-generated comments unless they add value.
- **Proper Naming Conventions:**
  - Use meaningful, descriptive names for variables, methods, and classes.
  - Follow camelCase for variables/methods and PascalCase for classes.
- **Logging:** Replace System.out.println (SOPL) with a proper logging framework (e.g., SLF4J, Log4j, or java.util.logging). Ensure logs include context for better debugging.
- **Constants:**
  - Store string literals and constant values in enum classes or constant files.
  - Use UPPERCASE_SNAKE_CASE for constant names.
- **Lombok API:** Use Lombok annotations (@Getter, @Setter, @ToString, @Builder, etc.) to reduce boilerplate code.
- Utility Classes:
  - Use private constructors for utility classes.
  - Apply @NoArgsConstructor(access = AccessLevel.PRIVATE) with Lombok.
- **Constructor/Method Parameters:** Limit the number of parameters in constructors or methods to 7. Consider using Builder Pattern for complex objects.
- **OOP Principles:** Always follow Object-Oriented Programming (OOP) principles such as Encapsulation, Abstraction, Inheritance, and Polymorphism.
- **Design Principles:**
  - SOLID principles (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion).
  - Ensure loose coupling and tight cohesion.
- **Use Parent Types:** Declare variables and return types using the

parent type, not the implementation class (e.g., List<Employee> instead of ArrayList<Employee>).

- **Generics:** Use generics for type safety in collections (e.g., List<Employee> employees = new ArrayList<>();).
- **Null Handling:**
  - Avoid returning or assigning null. Use empty objects, default values, or Optional.
  - Use Objects.requireNonNull() or utility methods for null safety.
- **Documentation & Comments:**
  - Use JavaDocs for all public classes, methods, and interfaces.
  - Write clear and concise comments where necessary. Avoid redundant or obvious comments.
- **Avoid Hardcoding:** Avoid hardcoding values in code. Use configuration files, environment variables, or constant classes.
- **Branching & Merging:**
  - Create feature branches for new tasks or bug fixes.
  - Merge changes via pull requests with proper code reviews.
- **Unit Testing:**
  - Write unit tests for all methods using JUnit, TestNG, or other frameworks.
  - Aim for at least 80% code coverage.
- **Exception Handling:**
  - Use meaningful custom exceptions where appropriate.
  - Log exceptions with clear, actionable messages.
- **Dependency Management:**
  - Use Maven or Gradle for dependency management.
  - Avoid adding unnecessary dependencies to reduce bloat.
- **Security:**
  - Sanitize inputs to prevent vulnerabilities like SQL Injection and XSS.
  - Avoid hardcoding sensitive information (e.g., passwords, API keys). Use environment variables or secure vaults.
- **Performance:**
  - Optimize loops and database queries for performance.
  - Use caching mechanisms where appropriate (e.g., Redis, Ehcache).
- **Version Control:**
  - Commit frequently with meaningful commit messages.
  - Avoid committing temporary or debug code.
- **CI/CD Integration:** Ensure code integrates smoothly into the Continuous Integration/Continuous Deployment (CI/CD) pipeline with automated

tests and builds.

- **Code Reviews:**
  - Participate actively in code reviews, providing constructive feedback.
  - Address feedback from peers promptly.
- **Database Best Practices:**
  - Use parameterized queries or ORM tools like Hibernate to avoid SQL Injection.
  - Ensure indexes are used appropriately in the database schema.
- **Thread Safety:** For multi-threaded code, ensure thread safety using appropriate synchronization or thread-safe data structures.
- **API Design:**
  - Follow RESTful API principles (if applicable).
  - Ensure APIs are well-documented using tools like Swagger or OpenAPI.
- **Error Codes:** Define consistent error codes and messages in APIs for client-side handling.
- **Code Metrics:** Regularly monitor code quality metrics like complexity, maintainability, and test coverage.
- **Build Tools:** Automate builds using tools like Maven or Gradle. Ensure proper dependency resolution and versioning.
- **Avoid Overengineering:** Write simple, clean, and maintainable code. Avoid adding unnecessary complexity.
- Code Optimization:
  - Remove redundant or duplicate logic.
  - Use efficient algorithms and data structures.
- **Memory Management:**
  - Close resources like file streams, database connections, and network sockets promptly.
  - Avoid memory leaks by managing references carefully.
- **Internationalization (i18n):** Use resource bundles for messages and texts that may need localization.
- Testing Best Practices:
  - Include integration and end-to-end tests in addition to unit tests.
  - Use mocks/stubs for external dependencies during tests.
- **Build Configurations:** Use separate configurations for development, testing, and production environments.
- **Event-Driven Architecture:** Use message queues or event-driven mechanisms (e.g., RabbitMQ, Kafka) for scalable systems where applicable.

- **Documentation:** Maintain an up-to-date README and developer guide with setup instructions, coding standards, and architecture diagrams.
- **Refactoring:** Periodically review and refactor code to improve readability, maintainability, and performance.

Links to Follow

- https://google.github.io/styleguide/javaguide.html
- https://docs.pmd-code.org/latest/pmd_rules_java.html
- https://www.tatvasoft.com/blog/java-best-practices/
- https://www.oracle.com/java/technologies/javase/codeconventions-introduction.html
- https://blog.jetbrains.com/idea/2024/02/java-best-practices/
- https://www.developer.com/languages/javascript/java-best-practices/
- https://github.com/in28minutes/java-best-practices#why-is-unit-testing-important
- https://www.javaguides.net/
- https://www.codejava.net/
- https://spring.io/projects/spring-boot

---------- The END ----------