Click to verify

When left unchecked, a Java codebase can accumulate bugs, memory leaks, unhandled exceptions, and barely comprehensible code fragments. So, you need to perform Java code reviews to keep your application efficient, scalable, and secure.One problem, though, is that most teams loathe large-scale reviews. We believe thats because they dont follow a proper structure, and as a result, the process becomes unwieldy and overwhelming.So, allow us to offer you a framework and some practices to make Java code review more manageable and efficient. Well also provide a detailed Java code review checklist, examples of widespread issues, and tools that streamline (and automate) various checks.Why Is a Code Review in Java Critical for Projects?A Java code review is a systematic assessment of your source code to ensure it meets established standards and coding practices. It involves inspecting the code for errors, performance problems, security gaps, syntax issues, and documentation clarity.In Java projects, regular code reviews have several purposes:Improved code quality: Reviews help adhere to uniform coding standards, naming conventions, and documentation style, resulting in efficient, error-free, and maintainable code.Performance enhancement: Addressing inefficient threads, memory-consuming objects, and collections helps improve application speed.Less waste: Code assessments provide an opportunity to mentor junior team members, reinforce best practices, and improve their skill levels. This ensures you make the most of your team and foster ongoing professional development to encourage long-term commitment to growing with the company.Increased code maintainability: Properly structured code leads to less technical debt and fewer unnecessary dependencies, making components more modular.Reduced vulnerabilities: Reviews help identify logical errors, memory leaks, unhandled exceptions, security flaws, and compliance issues.Even experienced developers and testers tend to make mistakes or overlook certain practices. A comprehensive review can keep your codebase cleaner and easier to maintain.Java Code Review Checklist Code review is one of those things everyone agrees is important but few teams actually enjoy doing. Its time-consuming, sometimes awkward, and often feels like a chore. But when done right? Its one of the highest-leverage practices you can have in a Java team.This guide is for teams who want to get serious about code quality without slowing everything down. Youll find real-world best practices, a practical checklist, and the kinds of things Java devs actually deal with in reviews. Whether your team is still growing or already shipping complex systems, this is for you.Why Java Code Reviews Still Matter (a Lot)Lets get one thing out of the way: Java isnt going anywhere. Its still one of the most-used backend languages in the world. But it also comes with real complexity verbose syntax, threading, exceptions, OOP patterns, dependency injection you get the idea.Thats exactly why Java code review matters.A single method can span 80 lines and look totally fine at first glance. But dig deeper and you might find a silent performance hit, a concurrency risk, or just something that breaks your teams internal conventions. Reviews catch those before they become production issues.But they also do more than that:They help everyone understand more of the codebaseThey onboard new devs fasterThey reduce technical debtThey prevent only Alice knows how this works situationsAnd maybe most importantly they help teams build shared trust. That matters.What Great Java Code Reviews Look LikeBefore we get to the checklist, lets talk about what good actually looks like. Because a code review is more than nitpicks and green checkmarks.1. Theyre focusedGreat reviews dont try to fix the entire codebase. They focus on the current change. They ask:Does this PR do what its supposed to do?Is it safe, readable, and maintainable?Are there edge cases were missing?2. Theyre specificRename this var isnt helpful.This variable name doesnt reflect what it actually holds maybe call it parsedResult instead of temp? is. Give context. Help your teammates understand why somethings worth changing.3. They teach (and learn)The best reviews are conversations. Not lectures. If youre reviewing, your goal isnt to catch mistakes its to help the author make the code better, and to share knowledge in the process. Ask questions. Point to docs. Offer alternatives.Java Code Review: The Essential ChecklistYou dont need to reinvent the wheel every time. This checklist covers what most reviewers should look for in any Java codebase. Copy, adapt, share with your team. Logic and RequirementsDoes the code do what the story or ticket says it should?Are edge cases handled (nulls, invalid input, concurrency)?Could this break existing functionality? Code StructureAre methods short and single-purpose?Are classes doing only one thing?Is the code easy to read and reason about? Style and ConventionsAre naming conventions followed (camelCase, PascalCase, constants in UPPER_SNAKE_CASE)?Is formatting consistent?Are there unnecessary comments or TODOs left? Error HandlingAre exceptions handled properly?Are we catching the right exceptions or swallowing stack traces?Are we logging sensitive information? Performance and ResourcesAny obvious inefficiencies (e.g., inside loops, unnecessary object creation)?Are streams, maps, and collections used correctly?Are resources (files, DB connections) being closed? TestsAre there tests for this change?Do the tests cover both expected and unexpected input?Would this break existing tests? Security (especially in web apps)Are inputs validated and sanitized?Are passwords/tokens stored or logged?Any potential injection or race condition risks?Common Mistakes to Avoid in Java Code ReviewEven experienced devs fall into these traps. Recognizing them early saves time (and frustration). The it compiles, ship it mindsetPassing tests are great. But they dont catch:Bad variable namesCode thats impossible to extendLogic that sort of works until it doesntYour job as a reviewer isnt to rubber-stamp. Its to think critically. Giant PRsWeve all seen them. 500+ lines. 9 files. Impossible to review well.Smaller is better. Aim for 100300 line diffs. Make it easy to give thoughtful feedback. Style nitpickingIf your team has autoformatting tools (and you should), dont waste time arguing about braces or spacing. Focus on logic, design, and impact. Reviewer-as-gatekeeperYour job isnt to prove youre smarter. Its to help your teammate ship better code. If somethings wrong, point it out respectfully and offer help when you can.Tips to Make Java Code Reviews Smoother (For Everyone)You can have the best checklist in the world, but if your process is broken, nobodys going to enjoy doing reviews.Here are some battle-tested tips:Keep PRs smallSeriously. This changes everything. Reviewers will give better feedback, faster.Review early, review oftenDont let PRs sit for two days. Give feedback while the context is fresh. If youre the author, request review as soon as tests pass.Use review tools properlyLeverage GitHub/GitLab features:Leave inline commentsSuggest changes directlyTag specific teammatesWrite good PR descriptionsContext matters. A one-liner like Fix bug doesnt help. Briefly explain whats changing and why. Link to related issues if relevant.Celebrate improvementsSaw a smart solution or a thoughtful refactor? Say so. Reviews arent just about criticism theyre a chance to recognize great work.Java Code Review FAQWhat if I dont understand the code Im reviewing?Thats okay. Ask questions. A good PR should be easy to follow. If its not, thats a signal not a you problem.How long should I spend reviewing a PR?Depends on size and complexity. A 200-line change? Maybe 1520 minutes. Set a timer if needed. More than that? Its probably too big.Who should review the code?Someone familiar with the domain is ideal. But everyone should be able to review most PRs. Rotate reviewers to avoid silos.What if we disagree in a review?Disagree respectfully. Use data or reference guides. Involve a third teammate if needed. Dont let it become personal.Cant AI do code review now?AI helps a lot but doesnt fully replace humans. It wont always know your internal conventions or business logic. Think of it as a power-up, not a substitute.Final ThoughtsJava code review isnt just a gatekeeping process. Its a way to build better software, together.If you want your team to move faster and keep quality high, investing in reviews is one of the best decisions you can make. Start with the checklist above. Avoid the common traps. Keep your PRs small, your comments kind, and your focus on long-term maintainability.And if your current review process feels like a mess? Thats normal. Most teams improve it gradually. You dont need a perfect system just one that keeps evolving.Thanks for reading. Now go review that PR in your backlog. Want to go deeper? Try using tools like Kodus to automate part of your Java code review workflow and free up your team to focus on the stuff that really matters.Your future self will thank you. In modern software development, code reviews are a crucial component in delivering high-quality software. They ensure that applications are maintainable, efficient, and bug-free. Without a structured review process, you may face performance issues and introduce bugs and inconsistencies that could put your team at great risk in delivery.This article will help you understand how important code review is and what you should include in a code review checklist for Java projects.Role of the Code Review ProcessIn Java development, maintaining high-quality code is an essential factor in delivering high-quality software. The code review process serves as a quality assurance measurement to discover bugs, improve code structure, and ensure coding standards. It also enhances team collaboration and reduces technical debt, leading to more sustainable source code.Why Are Code Reviews Crucial for Java Projects?Java is a highly complex programming language with a strong object-oriented nature designed to enhance a software applications reliability, scalability, and maintainability, which could be compromised without a rigid code review process. Here are several reasons why code reviews are essential for Java projects.Java applications enforce object-oriented programming that heavily relies on class hierarchies, inheritance, and encapsulation.Poorly structured classes and long inheritance trees or improper methods typically lead to code that is tightly coupled and difficult to maintain.Large-scale enterprise applications require long-term maintainability.Most Java applications are built for large-scale enterprise systems, which require long-term maintainability.When code isnt reviewed, it can become inefficient, bloated, and extremely challenging to refactor over time.Performance optimization is a key factor in Java.To build high-performance systems, memory usage, CPU usage and responsiveness must be optimized. Incorrect implementation of these key factors could cause significant issues such as memory leakages and inefficient loops.Code reviews help identify inefficient use of data structures, excessive use of object references, and nested loops.Exception handling in Java requires mindful reviews.Java is a programming language that promotes a strong checked exception mechanism, which significantly enhances the reliability of an application. However, it also has a flaw, as it might lead to an excessive use of try-catch blocks without logging them.A good code review process not only handles these exceptions well but also uses custom exception classes where necessary and standardized logging mechanisms like SLF4J or Log4J to debug.Early discovery of security vulnerabilities in Java is essential.Java is widely used in security-critical sectors like banking, finance systems, government applications, and e-commerce engines.A well-processed code review helps to ensure that applications are secure, use proper encryption hashing, and avoid unsafe Java features like insecure deserialization of objects.Javas multi-threading model can be complex and error-prone.Core features, such as concurrent programming and multi-threading models, can introduce deadlocks, synchronization issues, and race conditions if not handled carefully.Code reviews can resolve these by correctly using synchronized blocks, deadlocks, and concurrent collections while ensuring thread safety in shared resources.The Ultimate Java Code Review ChecklistAs we now have a clear idea of the importance of code review guidelines, lets see what should be on a Java code review checklist.1. Code Readability and MaintainabilityValidate that the code is easy to read and understand.Validate whether the variables and methods used adhere to Java naming conventions.Ensure comments are clear, relevant, concise, and consistent in formatting and indentation.// Badint x = 1;if (x == 1) { System.out.println("Yes");}// Goodint userStatus = 1;if (userStatus == 1) { System.out.println("User is active");}2. Adherence to Coding StandardsValidate that the code aligns with the standard Java conventions like indentation, line length, use of spaces, etc.Ensure consistency in the naming and organization of classes, methods, and variables.// Badpublic class myclass{public void getdata( ) {System.out.println("data");}}// Goodpublic class MyClass { public void getData() { System.out.println("data"); }}3. Error Handling and Exception ManagementValidate that exceptions are well handled with meaningful messages.Ensure a clear strategy exists for exception propagation, logging, and recovery.// Badtry { int result = 10 / 0;} catch (Exception e) { e.printStackTrace();}// Goodtry { int result = 10 / 0;} catch (ArithmeticException e) { System.err.println("Cannot divide by zero: " + e.getMessage()); // Log exception properly here}4. Code Performance, Optimization, and EfficiencyEnsure that loops, redundant computations, and resource management are implemented efficiently.Suggest refinements for code performance without sacrificing readability.Look for potential memory leaks or excessive object creation.// Bad (redundant loopList names = getNames();for (int i = 0; i < names.size(); i++) { System.out.println(names.get(i));}// GoodList names = getNames();for (String name : names) { System.out.println(name);}5. Unit Tests and Test CoverageEnsure that tests cover both happy paths and edge cases.Validate that unit tests are independent, automated, and run consistently.Ensure meaningful assertions are used, and always check if the expected outcome is met.6. Code Modularity and ReusabilityEnsure the methods/classes used are small and have a single responsibility.Ensure that the redundant code has been refactored into reusable methods.// Badpublic void calculateTax() { double tax = salary * 0.18; System.out.println("Tax: " + tax);}// Goodpublic double calculateTax(double salary) { return salary * 0.18;}Ensure that design patterns like Factory and Singleton are used where applicable.7. API and Database InteractionValidate that API calls are handled asynchronously.Ensure SQL queries are optimized for enhanced performance.Check if proper indexing and caching mechanisms are used.SonarQube: One of the most popular tools available in the market for continuous code inspection. Integrates seamlessly with most development environments and automatically analyzes Java code for bugs and vulnerabilities.Qodo Gen: An IDE plugin that generates context-aware code and tests, offers smart code suggestions, and helps developers write better tests directly within their environment.Qodo Merge: A Git-integrated agent that enhances pull request workflows by reviewing code, highlighting issues, and generating AI-powered suggestions for improvements.Checkstyle: An SCA tool designed to validate adherence to Java coding standards. This highly configurable tool can be integrated into your build process to detect style violations automatically.PMD: An SCA tool used to identify common programming flaws in Java code, such as unused variables, empty catch blocks, and overly complex methods.FindBugs: Another SCA tool focusing on analyzing bytecode to identify potential errors. It also helps detect performance issues and security vulnerabilities.JDepend: A tool used to measure the quality of the design of Java code. It calculates metrics related to the structure and dependencies of Java packages and classes.JaCoCo: A code coverage tool for Java that measures and generates reports on which parts of the code are covered and not covered by unit tests.Benefits of Regular Code ReviewsFewer bugs in production: Early discovery of bugs helps resolve them earlier than fixing them post-release, which is very costly.Enhanced code quality: Scrutinizing the code by multiple developers prevents issues from going unnoticed in production.Better team collaboration: Developers learn new techniques, best practices, and solutions from one another.Faster development cycle: Discovering and resolving defects or issues helps expedite the delivery cycle as it reduces the complexity of debugging them in a later stage.Increased maintainability: Well-reviewed clean code is more modular, reusable, and easier to understand. This leads to better long-term maintainability.Code Review Pitfalls and How to Avoid Them1. Rushing Through Code ReviewsReviewers sometimes rush through code due to time pressure, leading to missed bugs or poor-quality approvals.How to avoid:Set aside time specifically for code reviews.Encourage smaller, incremental pull requests to keep reviews manageable.2. Overlooking Small IssuesWhile searching for big bugs or design flaws, its easy to skip over small but important issues like naming, duplication, or formatting.How to avoid:Minor issues impact maintainability; treat them seriously.Use a checklist to consistently catch both big and small problems.3. Focusing Only on SyntaxGetting stuck on minor formatting or stylistic changes can slow progress and create tension.How to avoid:Prioritize functionality, readability, and logic over nitpicking.Use automated tools to handle formatting issues.4. Ignoring Coding StandardsWithout consistent standards, code becomes hard to read and maintain across the team.How to avoid:Follow agreed-upon coding conventions.Use linters and formatters to automate standard checks.5. Not Providing Constructive FeedbackNegative or vague comments like this is bad arent helpful and can demotivate developers.How to avoid:Give clear, specific suggestions for improvement.Focus on the code, not the coder. Keep feedback respectful and helpful.Final ThoughtsCode reviews in Java projects arent just about discovering syntax errors. They help ensure key factors in a project like scalability, performance, security, and maintainability. Since Java is widespread in mission-critical applications across multiple industries, neglecting code review may cause significant impact through costly defects and long-term issues.FAQs1. Why is code review critical for Java projects?Code reviews help catch bugs early, improve code quality, ensure consistency, and promote knowledge sharing across the team. In Java projects, they also help maintain strict type safety and adherence to object-oriented design principles.2. How do I ensure Java code readability during a review?Look for clear naming of variables and methods, proper indentation, consistent formatting, and meaningful comments. Encourage short, focused methods and avoid deeply nested logic.3. How can I enforce Java coding standards in my team?Use tools like Checkstyle, PMD, or SonarQube to automatically flag violations. Set up shared formatting rules in the IDE and make coding standards part of the review checklist.4. How do I review Java code for proper unit testing?Check if tests cover both normal and edge cases. Ensure they are isolated and repeatable and use clear assertions. Also, verify the use of mocking frameworks like Mockito for testing dependencies. Join the DZone community and get the full member experience. Join For Free Clean CodeChecklist ItemCategoryUse Intention-Revealing NamesMeaningful NamesPick one word per conceptMeaningful NamesUse Solution/Problem Domain NamesMeaningful NamesClasses should be small!ClassesFunctions should be small!FunctionsDo one ThingFunctionsDon't Repeat Yourself (Avoid Duplication)FunctionsExplain yourself in codeCommentsMake sure the code formatting is appliedFormattingUse Exceptions rather than Return codesExceptionsDon't return NullExceptions* Reference: ItemCategoryMake class final if not being used for inheritanceFundamentalsAvoid duplication of codeFundamentalsRestrict privileges: Application to run with the least privilege mode required for functioningFundamentalsMinimize the accessibility of classes and membersFundamentalsDocument security related informationFundamentalsInput into a system should be checked for valid data size and rangeDenial of ServiceAvoid excessive logs for unusual behaviorDenial of ServiceRelease resources (Streams, Connections, etc) in all casesDenial of ServicePurge sensitive information from exceptions (exposing file path, internals of the system, configuration)Confidential InformationDo not log highly sensitive informationConfidential InformationConsider purging highly sensitive from memory after useConfidential InformationAvoid dynamic SQL, use prepared statementInjection InclusionLimit the accessibility of packages,classes, interfaces, methods, and fieldsAccessibility ExtensibilityLimit the extensibility of classes and methods (by making it final)Accessibility ExtensibilityValidate inputs (for valid data, size, range, boundary conditions, etc)Input ValidationValidate output from untrusted objects as inputInput ValidationDefine wrappers around native methods (not declare a native method public)Input ValidationTreat output from untrusted object as inputMutabilityMake public static fields final (to avoid caller changing the value)MutabilityAvoid exposing constructors of sensitive classesObject ConstructionAvoid serialization for security-sensitive classesSerialization DeserializationGuard sensitive data during serializationSerialization DeserializationBe careful caching results of potentially privileged operationsSerialization DeserializationOnly use JNI when necessaryAccess Control * Reference: ItemCategoryAvoid excessive synchronizationConcurrencyKeep Synchronized Sections SmallConcurrencyBeware the performance of string concatenationGeneral ProgrammingAvoid creating unnecessary objectsCreating and Destroying Objects* Reference: ItemUse checked exceptions for recoverable conditions and runtime exceptions for programming errorsExceptionsFavor the use of standard exceptionsExceptionsDon't ignore exceptionsExceptionsCheck parameters for validityMethodsReturn empty arrays or collections, not nullsMethodsMinimize the accessibility of classes and membersClasses and InterfacesIn public classes, use accessor methods, not public fieldsClasses and InterfacesMinimize the scope of local variablesGeneral ProgrammingRefer to objects by their interfacesGeneral ProgrammingAdhere to generally accepted naming conventionsGeneral ProgrammingAvoid finalizersCreating and Destroying ObjectsAlways override hashCode when you override equalsGeneral ProgrammingAlways override toStringGeneral ProgrammingUse enums instead of int constantsEnums and AnnotationsUse marker interfaces to define typesEnums and AnnotationsSynchronize access to shared mutable dataConcurrencyPrefer executors to tasks and threadsConcurrencyDocument thread safetyConcurrencyValid JUnit / JBehave test cases existTesting* Reference: Code AnalysisCategoryChecklist ItemCheck static code analyzer report for the classes added/modifiedStatic Code Analysis code style Java (programming language) Opinions expressed by DZone contributors are their own. Related Image SourceCode reviews are an essential part of the software development process, especially in Java, a language known for its resilience and cross-platform capabilities. Effective code reviews lead to higher code quality, reduced bugs, and improved team collaboration. This article aims to provide actionable insights and best practices for conducting Java code reviews efficiently and effectively.Understanding the Importance of Code ReviewsJava, being a statically typed language, brings a level of complexity and strictness in its code structure. Code reviews in this context serve multiple purposes:Improving Code Quality: Javas syntax and feature set, including its object-oriented nature, demand a thorough understanding to write efficient and error-free code. Reviews help in maintaining high standards by scrutinizing the use of Java-specific features like stream APIs, exception handling, and memory management.Detecting Bugs Early: Javas exception handling mechanism, while strong, can often lead to complex error-handling code. During reviews, such errors are more likely to be caught, reducing the likelihood of runtime exceptions.Knowledge Sharing Among Team Members: Code reviews in Java offer a platform for less experienced developers to learn from more seasoned colleagues, especially regarding Javas vast standard library and best practices.Benefits of Regular Code ReviewsConducting regular code reviews yields numerous advantages:Faster Onboarding of New Team Members: Regular reviews create an environment of continuous learning, making it easier for new team members to get up to speed with the codebase and the teams coding conventions.Consistent Code Style and Standards: Java projects benefit from a consistent style and adherence to best practices, which is more achievable through regular code reviews. Tools like Checkstyle or PMD can be used alongside manual reviews to enforce these standards.Enhanced Team Collaboration and Morale: Code reviews encourage an atmosphere of collective ownership and responsibility. This collaborative approach not only improves the code quality but also boosts team morale and cohesion.In-Depth Look at the BenefitsRisk Mitigation: In complex systems, especially those written in Java, the risk of introducing bugs that can have far-reaching consequences is high. Regular code reviews act as a checkpoint to mitigate these risks.Performance Optimization: Javas performance can be significantly influenced by how the code is written, especially in terms of memory management and efficient use of collections. Through code reviews, more experienced developers can guide others on optimizing code for better performance.Adherence to Best Practices: Javas evolution introduces new features and best practices. Code reviews are an excellent way for a team to stay updated and incorporate these practices into their daily work.Setting Up a Code Review ProcessImplementing an effective code review process is a critical step in ensuring the success of Java development projects. A well-defined process not only streamlines the review activities but also ensures consistency and effectiveness across the team.Defining Clear Objectives and GuidelinesBefore diving into code reviews, its important to set clear objectives and guidelines for what the process should achieve.Establishing Code Review Objectives: Define what your team aims to accomplish with code reviews. This could include improving code quality, ensuring code conforms to project standards, sharing knowledge among team members, and identifying potential bugs early.Creating a Checklist for Java Code Reviews: A checklist acts as a guide for reviewers, ensuring that they consistently cover all essential aspects of the code. For Java, this might include checking for memory leaks, proper use of Java Collections, adherence to object-oriented principles, and effective exception handling.Tools and Technologies for Code ReviewsChoosing the right tools and technologies is pivotal for facilitating an efficient code review process.Utilizing Platforms like GitHub, GitLab, or Bitbucket: These platforms offer built-in code review tools that allow for inline comments, discussions, and approvals. They also integrate with continuous integration tools, making it easier to automate certain aspects of code reviews.Code Review Tools Specific to Java: Tools like Crucible and Review Board offer additional features tailored for Java development, such as deeper integration with Java IDEs (e.g., IntelliJ IDEA or Eclipse) and specific code analysis capabilities.Establishing a Review WorkflowA structured workflow is essential for efficient and effective code reviews.Define the Review Process: Establish who will review the code, when reviews will occur, and how feedback will be communicated. This includes deciding whether to use a lightweight approach (e.g., over-the-shoulder reviews) or a more formal process (e.g., pull requests).Integrating Code Reviews into the Development Cycle: Ensure that code reviews are a regular part of the development cycle, not an afterthought. This integration helps in catching issues early and reduces the burden of addressing multiple issues at a later stage.Training and OnboardingFor the code review process to be successful, team members need to be properly trained.Educate the Team: Provide training on how to conduct effective code reviews, including how to give and receive feedback constructively.Onboarding New Members: Make sure new team members understand the code review process and are comfortable with the tools and standards used by your team.Best Practices for Conducting Code ReviewsEffective code reviews are not just about finding bugs but also about improving code quality, enhancing knowledge sharing, and fostering a collaborative team environment. Here are some best practices specifically tailored for Java code reviews:Reviewing Code EffectivelyEffective code review goes beyond just spotting errors. It involves understanding the codes purpose and ensuring it aligns with the projects goals and coding standards.Focus on Logic and Structure, Not Just Syntax: While syntax errors are important, pay more attention to the overall logic and structure of the code. Check if the code is well-organized, modular, and follows Javas best practices like proper use of classes and interfaces, and efficient memory management.Understanding Context: Its crucial to understand the context of the changes. Reviewers should familiarize themselves with the relevant parts of the codebase to provide more informed feedback.Automated Code Analysis Tools: Utilize tools like SonarQube or FindBugs for static code analysis. These can automatically detect potential bugs, code smells, and security vulnerabilities, allowing reviewers to focus on more complex issues.Giving and Receiving FeedbackThe way feedback is given and received can significantly impact the effectiveness of code reviews.Constructive Feedback: Feedback should be constructive and aimed at improving the code. Avoid personal comments and focus on the code and its improvement.Examples of Constructive Feedback in Java Code Reviews: For instance, if you notice an inefficient use of collections, suggest an alternative approach and explain why its more efficient.Encouraging Discussion: Foster a culture where developers feel comfortable discussing suggestions and explaining their reasoning.Handling Larger Code Bases and Complex ChangesReviewing large commits or complex code changes can be challenging but is common in Java projects due to the nature of the language.Strategies for Reviewing Large Commits: Break down large commits into smaller, manageable parts. If a commit touches multiple areas, consider having multiple reviewers with expertise in each area.Tackling Complex Code: For complex changes, it might be beneficial to have a walkthrough or a pair programming session. This helps in understanding the rationale behind the changes and ensures that the reviewer and the author are on the same page.Continuous Learning and ImprovementCode reviews are a learning process for both authors and reviewers.Learning from Each Review: Encourage team members to view code reviews as learning opportunities. Discussing different approaches and solutions helps in broadening knowledge and skills.Regularly Reviewing the Review Process: Periodically assess the effectiveness of your code review process and make adjustments as necessary. This can involve changing the tools used, the way feedback is given, or the checklist items.Common Pitfalls and How to Avoid ThemCode reviews, while essential, can sometimes be fraught with challenges that can hinder their effectiveness. Recognizing and addressing these common pitfalls is crucial for a successful code review process, especially in Java development, which often involves complex code structures.Overcoming Common Challenges in Code ReviewsEvery team encounters obstacles in code reviews. Addressing these head-on improves the process and outcomes.Avoiding Nitpicking: Its easy to get caught up in minor issues that dont significantly impact the overall quality of the code. Focus on issues that genuinely matter for the projects success. Use a linter or automated code formatter to handle trivial style issues.Balancing Between Overly Critical and Too Lenient Reviews: Finding the right balance in code reviews is crucial. Be thorough but also practical. Not every code piece needs to be perfect; it needs to be good enough for the projects current stage and context.Time Management and EfficiencyEfficiently managing time during code reviews ensures that they are effective without becoming a bottleneck in the development process.Setting Realistic Timeframes for Reviews: Allocate a reasonable amount of time for code reviews and stick to it. Lengthy reviews can be counterproductive.Streamlining the Review Process Without Compromising Quality: Develop a process that is thorough yet efficient. This can include using checklists, automated tools, and dividing large reviews among multiple reviewers.Handling Disagreements ConstructivelyDisagreements are inevitable in code reviews, but they can be handled constructively.Fostering a Positive Environment for Discussion: Encourage open and respectful discussions. Disagreements should be about the code, not the coder.Finding Compromises: When disagreements arise, focus on finding a compromise that aligns with the project goals. Sometimes, this might involve deferring to a team lead or voting on a particular approach.Recognizing and Adapting to Different Skill LevelsIn a team, developers will have varying levels of expertise, which can impact the review process.Tailoring Feedback to the Developers Skill Level: Adjust the depth and nature of your feedback based on the developers experience. Offer more guidance and explanations to less experienced developers.Encouraging Continuous Learning: Promote an environment where developers of all levels can learn and improve through code reviews. This might involve pairing less experienced developers with more seasoned ones during reviews.ConclusionEffective Java code reviews are a crucial component of the software development process. They not only enhance code quality and reduce the likelihood of bugs but also foster a culture of collaboration and continuous learning within development teams. By understanding the importance of code reviews, setting up a thorough review process, adhering to best practices, and avoiding common pitfalls, teams can significantly improve their Java development efforts.Remember, the key to successful code reviews lies in constructive feedback, efficient processes, and a commitment to ongoing improvement. As technologies and practices evolve, so should your approach to code reviews. Regularly revisiting and refining your code review practices will ensure they remain effective and aligned with your teams goals.Through mindful implementation and continuous refinement of these practices, teams can reap the full benefits of code reviews, resulting in strong, efficient, and maintainable Java applications.Oracles Java DocumentationGitHubs Pull Request GuidelinesSonarQube This Java code review checklist is not only useful during code reviews, but also to answer an important Java job interview question,Q. How would you go about evaluating code quality of others work?This can judge a candidates real experience & technical hows.You also learn a lot from peer code reviews. What has been written well? Why was it done this way? Could this have been written differently?, etc. This is one of the benefits of volunteering to review code via open-source project contribution.FunctionalityChecklistDescription/exampleFunctionality is implemented in a simple, maintainable, and reusable manner.Keep in mind some of the design principles like SOLID design principles, Dont Repeat Yourself (DRY), and Keep It Simple ans Stupid (KISS).Also, think about the OO concepts A PIE. Abstraction, Polymorphism, Inheritance, and Encapsulation. These principles and concepts are all about accomplishing Low coupling and High cohesion.Apply functional programming (FP) paradigm where it makes more sense.ChecklistDescription/exampleUse of descriptive and meaningful variable, method and class names as opposed to relying too much on comments.E.g. calculateGst(BigDecimal amount), BalanceLoader.java, etc.Bad: List list;Good: List users;Class and functions should be small and focus on doing one thing. No duplication of code.E.g. CustomerDao.java for data access logic only, Customer.java for domain object, CustomerService.java for business logic, and CustomerValidator.java for validating input fields, etc.Similarly, separate functions like processSalary(String customerCode) will invoke other sub functions with meaningful names likeevaluateBonus(String customerCode),evaluateLeaveLoading(String customerCode), etcFunctions should not take too many input parameters.Bad: processOrder(String customerCode, String customerName, String deliveryAddress, BigDecimal unitPrice, int quantity, BigDecimal discountPercentage);Good: processOrder(CustomerDetail customer, OrderDetail order);where CustomerDetail is a value object with attributes like customerCode, customerName, etc.Use a standard code formatting template.Share the template across the development team.Declare the variables with the smallest possible scope.For example, if a variable tmp is used only inside a loop, then declare it inside the loop, and not outside.Dont preserve or create variables that you dont use again.E.g. instead of boolean removed = myItems.remove(item); return removed;Do: return myItems.remove(item);Omit needless and commented out code. No System.out.println statements either.You have source control for the history. Use proper logging frameworks like slf4j and logback for logging.FundamentalsChecklistDescription/exampleMake a class final and the object immutable where possible.Immutable classes are inherently thread-safe and more secured. For example, the Java String class is immutable and declared as final.Minimize the accessibility of the packages, classes and its members like methods and variables.E.g. private, protected, and public access modifiers.Code to interface as opposed to implementation.Bad: ArrayList names = new ArrayList();Good: List names = new ArrayList();Use right data types.For example, use BigDecimal instead of floating point variables like float or double for monetary values. Use enums instead of int constants.Avoid finalizers and properly override equals, hashCode, and toString methods.The equals and hashCode contract must be correctly implemented to prevent hard to debug defects.Write fail-fast code by validating the input parameters.Apply design by contract.Return an empty collection or throw an exception as opposed to returning a null. Also, be aware of the implicit autoboxing and unboxing gotchas.NullpointerException is one of the most common exceptions in Java.Key Areas like Security, Exception Handling, Performance, Memory/Resource leaks, Concurrency, etcChecklistDescription/exampleDont log sensitive data.Security.Clearly document security related information.Security.Sanitize user inputs.Security.Favor immutable objects.Security.Use Prepared statements as opposed to ordinary statements.Security to prevent SQL injection attack.Release resources (Streams, Connections, etc).Security to prevent denial of service attack (DoS) and resource leak issues.Dont let sensitive information like file paths, server names, host names, etc escape via exceptions.Security and Exception Handling.Follow proper security best practices like SSL (one-way, two-way, etc), encrypting sensitive data, authentication/authorization, etc.Security.Use exceptions as opposed to return codes.Exception Handling.Dont ignore or suppress exceptions. Standardize the use of checked and unchecked exceptions. Throw exceptions early and catch them late.Exception Handling.Write thread-safe code with proper synchronization and use of immutable objects. Also, document thread-safety.Concurrency.Keep synchronization section small and favor the use of the new concurrency libraries to prevent excessive synchronization.Concurrency and Performance.Reuse objects via flyweight design pattern.Performance.Presence of long lived objects like ThreadLocal and static variables holding references to lots of short lived objects.Memory Leak and PerformanceBadly constructed SQL, REGEX, etc.Performance. E.g. Cartesian joins in SQL and back tracking regular expressions.Inefficient Java coding and algorithms in frequently executed methods leading to death by thousand cuts.PerformanceOther general programmingChecklistDescription/exampleFavor using well proven frameworks and libraries as opposed to reinventing the wheel by writing your own.E.g. Apache commons libraries, Google Gauva libraries, Spring libraries, XML/JSON libraries, etc.Presence of JUnit and JBehave test cases.Check the test coverage and quality of the unit tests with proper mock objects to be able to easily maintain and run independently/repeatedly.Test only a unit of code at a time (e.g. one function).Unit tests must be independent of each other. They should run independently.Set up should not be too complicated.Mockout external states and services that you are not asserting. For example, retrieving data from a database.Avoid unneccessary assertions.Start with functions that have the fewest dependencies, and work your way up.Write unit tests for negative scenarios like throwing exceptions, negative values, null values, etc.Dont have try/catch inside unit tests. Use throws Exception statement in test case declaration itself.Dont have any System.out.println(..)Ensure that the unit tests are written properly.Dont write unit tests for the sake of writing one.Presence of hard coded config values.Externalize configuration data in a .properties file. Sensitive information like password must be encrypted.Presence and implementation of non functional requirements like archiving, auditing, and purging data and application monitoring where required.It is easy to ignore these non functional requirements. Doing code reviews (or peer reviews) for your Java codebase? Here's our collection of tips to make you an even better reviewer.What should be on your code review checklist?Understandability Just based on the term code review , you may immediately think of "looking for bugs" when hearing the term. While this is on our list, we'd argue reading the code with a focus on understandability (and readability) may be even more valuable!Naming: As you know, naming things is hard, so be sure to double-checkHelpful commentsOpportunities to introduce (or use) utility methodsFor example, let's imagine we encounter the following code snippet during our review:String url = getUrl();if (url.startsWith("https://")) { url = "https://" + url;}With a helper method in place, we can reduce this toString url = StringUtils.ensureStartsWith(getUrl(), "https://");While this may not seem like much, small improvements like this add up quickly. This means there is less code to maintain, and (hopefully) less bugs as you can rely on "battle-tested" parts of your code base.Consistency Doing things consistently helps you and your team to find your way around different the areas of your codebase - even the ones that you've never worked on before.There are quite a few aspects around consistency, and maybe of them can (and should) be checked by linters, not humans.But even with tools in place, there are still things you can look out for in your code review.Bugs This one is obvious. Of course, code reviews are a big opportunity to spot bugs early and before they land in production.Let's consider this example:public boolean isValidUrl(String url) { return url.startsWith("http://");}As an attentive reviewer, you'll probably see that this method will return false for URLs that use HTTPS, which is probably not what we want.So take your time, focus on the code, and think about edge cases or scenarios that may not be handled correctly the code yet.Tests Were any tests added (or adapted) with the code changes? If not, this may be a good opportunity to clarify with the implementor if they can add additional tests, for example using JUnit (unit tests) or Selenium (end-to-end tests).Sure, this may mean that the feature / bugfix will not be merged right away. But this will prevent regression bugs, and everyone on your team will feel more confident with the stability of your software.Performance Let's consider this simple example:for (File inputFile : listInputFiles()) { Parser parser = createParser(getOutputFolder());parser.parseFile(inputFile);}Reading the code carefully, we can see that parser is re-created on every loop iteration, but it is not dependent on the loop at all (for the sake of this example, let's assume that parseFile() has no side effects we need to worry about).This means that as a reviewer, you could suggest to move the createParser() call out of the loop.What should NOT be on your code review checklist?Now that we know what to look for during a code review, what should we (probably) not put onto our review checklist? Anything that can be automated This is probably the most important rule of code reviews:Automate all checks that can be automatedThis includes things like: Nitpicking Nitpicking (sometimes abbreviated with NIT) means pointing out trivial or unimportant things during code reviews.While there is no clear definition of what exactly is (not) nitpicking, let's just bring a little common sense. Code reviews are important, but they shouldn't slow you down unnecessarily.Here's an hypothetical example: public static void readInputFiles(String folder) {}Please don't be like that Summary Code reviews for your Java code can be a super power if everybody involved understands what (and what NOT) to look for.Focus on consistency, performance, testing, and other topics that you (as a human) can judge best. Have automation in place where possible, and don't waste everyone's time with useless discussions about trivial things. Are you looking for a Java code review checklist?According to a survey, developers spend 45% of their time fixing bugs rather than writing new code. Strict code reviews allow developers to spend less time resolving bugs and delivering new features.Developing a Java application requires a deep understanding of object-oriented programming, Java language, tools like Eclipse IDE, Java Standard Library, Git version control, and frameworks like Hibernate, Spring Boot, etc. You also require skills in database management, like JDBC (Java Database Connectivity) and ORM (object-relational mapping), application security practices, performance optimization, etc.If you dont have a professional team with this relevant expertise to take on the task, then submit a request for a complimentary discovery call, and one of our tech account managers who managed similar projects will contact you shortly.Lets discuss Java code review checklist items to ensure a well-written, functional, and maintainable software application. Get a complimentary discovery call and a free ballpark estimate for yourproject Trusted by 100x of startups and companies like Java Code Review ChecklistCheck off the following when performing a Java code review:Functional checks ensure the software works as intended. Review the following:The code logic correctly implements the required functionality. Check input validation and output accuracy. The code should validate input values to avert unexpected behavior, data corruption, etc., and verify the actual output matches the expected outcome. There are no functional regressions or inconsistencies in case of changes to the existing code. The functionality is implemented in a simple and reusable manner. Check for the following design principles:DRY (Dont Repeat Yourself), SOLID (Single-responsibility Principle, Open-closed Principle, Liskov Substitution Principle, Interface Segregation Principle, and Dependency Inversion Principle), etc. Observe object-oriented principles of Abstraction, Polymorphism, Inheritance, and Encapsulation.You should check for the following to ensure a clean code:Follow Java code conventions to enhance code readability. All package names in Java are written in lower case, all constants in upper case, variables in CamelCase, etc. Use meaningful and descriptive names instead of relying on comments to understand code. For example, calculateTax(BigDecimal amount).There should be no code duplication. Classes and methods should focus on one functionality. Declare variables with the smallest scope. E.g., if a variable is used