**https://www.adservio.fr/post/java-code-review-checklist**

**1. Ensure the code follows standard naming conventions**

Meaningful naming conventions help ensure the readability and maintainability of the application.

As such, ensure variable, method, and class names convey the subject:

*addPerson()*

Could be clarified to:

*addEmployee()*

Use all lower cases for package names and use reversed Internet domain naming conventions:

*org/companyname/appname*

Class names should start with Capitals:

*Employee, Student,*

Variable and method names should use CamelCase:

*employeeList, studentName, displayEmployees()*

**2. Make sure it handles constants efficiently**

Constants help improve memory as they are cached by the JVM. For values that are reused across multiple places, create a constant file that holds static values.

Favor database-driven values over dynamic values. Also, use ENUMs to group constants.

**3. Check for proper clean Up**

It is common during development to use procedures that help with your coding and debugging (hard coded variables, for example). It is good practice to remove these items and others such as:

* Console print statements
* Unnecessary comments
* Use @deprecated on method/variable names that aren’t meant for future use

**4. Handle strings appropriately**

If you need to perform a lot of operations on a [String](https://stackabuse.com/string-vs-stringbuilder-vs-stringbuffer-in-java/), use StringBuilder or StringBuffer.

Strings are immutable, whereas [StringBuilder](https://docs.oracle.com/javase/tutorial/java/data/buffers.html) and StringBuffer are mutable and can be changed. Additionally, a new String object is created for every concatenation operation.

Rather than creating multiple items, using a mutable object is preferred.

**5. Optimize to use switch-case over multiple If-Else statements**

Rather than using multiple if-else conditions, use the cleaner and more readable switch-case.

Doing so makes the logic cleaner and optimizes the app's performance.

switch(expression) {

 case x:

// code block

   break;

case y:

  // code block

   break;

 default:

   // code block

}

**6. Ensure the code follows appropriate error handling procedures**

The **NullPointerException**is one of the most common and frustrating exceptions.

However, they can be avoided by performing a null check on a variable before operating on it.

The best practice is to use checked exceptions for recoverable operations and use runtime exceptions for programming errors.

Another area to evaluate during a Java code review is to ensure all exceptions are wrapped in custom exceptions.

In this way, the stack trace is preserved, making it easier to debug when things go wrong.

Also, it should declare specific checked exceptions that the method throws rather than generic ones. Doing so doesn’t give you the option to handle and debug the issue appropriately.

**Avoid this:**

public void hello() throws Exception { //Incorrect way

}

**Try this instead:**

public void hello() throws SpecificException1, SpecificException2 { //Correct way

}

Use the try-catch block for exception handling with appropriate actions taken in the catch block.

Also, use a finally block to release resources, such as database connections, in the finally block. This allows you to close the resource gracefully.

**7. Avoid unnecessary comments in code?**

Comments should not be used to explain code. If the logic is not intuitive, it should be rewritten. Use [comments](https://stackoverflow.blog/2021/07/05/best-practices-for-writing-code-comments/)to answer a question that the code can’t.

Another way to state it is that the comment should explain the “why” versus “what” it does.

**8. Validate that the code follows separation of concerns**

Ensure there is no duplication. Each class or method should be small and focus on one thing.

**For example:**

EmployeeDao.java - Data access logic

Employee.java - Domain Logic

EmployeerService.java - Business Logic

EmployeeValidator.java - Validating Input Fields

**9. Does the code rely on frameworks rather than custom logic when possible?**

When time is of the essence, reinventing the wheel is time wasted. There are plenty of proven frameworks and libraries available for the most common use cases you may need.

Examples include Apache Commons libraries, Spring libraries, and XML/JSON libraries.

**10. Make sure variables don’t cause memory leaks**

Creating a bunch of unnecessary variables can overwhelm the heap and lead to [memory leaks](https://docs.oracle.com/javase/10/troubleshoot/troubleshoot-memory-leaks.htm#JSTGD266) and cause [performance](https://www.adservio.fr/post/bringing-performance-to-java-developers) problems. This occurs when an object is present in the heap but is no longer used, and the garbage collection cannot remove it from memory.

Example:

**Avoid This**

*boolean removed = myItems.remove(item); return removed;*

**Try This Instead**

*return myItems.remove(item);*

https://deepsource.io/blog/java-code-review-guidelines/