♥ 30+ Java Code Review Checklist Items

Posted on February 17, 2015 by Arulkumaran Kumaraswamipillai — No Comments ↓

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?

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.

Functionality

Checklist	**Description/example**

Functionality is implemented in a simple, maintainable, and reusable manner.	Keep in mind some of the design principles like **SOLID** design principles, Don't 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.

Clean code

Checklist	**Description/example**

Use 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<String> 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 like evaluateBonus(String customerCode), evaluateLeaveLoading(String customerCode), etc

Functions 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.
Don't preserve or create variables that you don't 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.

Fundamentals

Checklist	**Description/example**
Make 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, default, and public access modifiers.

Code to interface as opposed to implementation.	**Bad**: ArrayList<String> names = new ArrayList<String>();
	
	 Good: List<String> names = new ArrayList<String>();

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, etc

Checklist	**Description/example**
Don't 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.
Don't 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.**
Don't 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	**Concurrency** and **Performance**.

use of the new concurrency libraries to prevent excessive synchronization.	
Reuse objects via flyweight design pattern.	**Performance**.
Presence of long lived objects like ThreaLocal and static variables holding references to lots of short lived objects.	**Memory Leak** and **Performance**
Badly 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.	**Performance**

Other general programming

Checklist	**Description/example**
Favor 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	Check the test coverage and quality of the unit tests with

cases.	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 independendtly. • 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. • Don’t have try/catch inside unit tests. Use throws Exception statement in test case declaration itself. • Don’t have ant System.out.println(…..)
Ensure that the unit tests are written properly.	Don’t 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.

Popular Member Posts

- ♦ 11 Spring boot interview questions & answers
  905 views
- ♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers
  816 views
- 001A: ♦ 7+ Java integration styles & patterns interview questions & answers
  427 views
- 18 Java scenarios based interview Questions and Answers
  409 views
- ♦ 7 Java debugging interview questions & answers
  324 views
- 01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers
  311 views
- 01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints
  304 views
- ♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers
  301 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

251 views

♦ Object equals Vs == and pass by reference Vs value

234 views

42 Like Share Tweet submit reddit 3 G+1 102 Share

Bio | **Latest Posts**

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ Understanding Java locks, multi-threading, and synchronized keyword

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers ›

Posted in Code Quality, Coding

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category ▼

© Disclaimer