<http://javarevisited.blogspot.sg/2011/08/code-comments-java-best-practices.html>

10 Best Practices to Follow while writing Code Comments

**Comments are an important part of writing code** not only in Java, but whatever programming or scripting language you use. At the same time this is one of the most abused things as well. Both writing no comment and writing too much comment is bad and this has been high lighted by many software gurus e.g. Robert C. Martin in his classic book Clean code. There is a whole chapter dedicated on How to write comments and finding pros and cos of comment. This article is my learning in same direction, here I am going to share with you guys some 0f the rule and best practices I follow while writing comments. Before that let's first see what is the purpose of having comment in the code? Why do we need comment, isn't writing code is enough. Some of the people I have met always argue that we are getting paid for writing code and not comment :).  
  
[code comments best practices in java](http://javarevisited.blogspot.com/2011/08/what-is-polymorphism-in-java-example.html)Anyway in my opinion we all agree with each other that **software spend only 10% time of its life in development and rest of 90% in maintenance**. This 90% part of maintaining the code is where comment can help you immensely. Since no single developer stays till whole life of any product or software and its often new people, who works of already written code. These are the people who read the code and not aware of why a certain piece of code has been written, here comments can help them to understand code quickly and believe me you will get lot of roses from that fellow developer :).    
  
Anyway long story short here are some of the things I try to follow while writing code:

### 10 tips on writing code comments

1) Focus on **readability of code**; assume that you don't have comments to explain the code. Give your method, variables and class meaningful name.  
  
2) **Don't write what code is doing**, this should be left for the code to explain and can be easily done by giving class, variable and method meaningful name. For example:

//calculates square root of given number

//using Newton-Raphson method

public void **abc(int a)**{

       r = a / 2;

       while ( abs( r - (a/r) ) > t ) {

       r = 0.5 \* ( r + (a/r) );

       }

       System.out.println( "r = " + r );

}

Above code is calculating square root using Newton-Raphson method and instead of writing comment you can just rename your method and variable as follows:

public void **squareRoot(int num)**{

       root = num/ 2;

       while ( abs(root - (num/ root) ) > t ) {

       r = 0.5 \* (root + (num/ root));

       }

       System.out.println( " root = " + root );

}

3) **Always write why you are writing this piece of code**, why you are writing this piece of code because this information is not visible until you write them in comments and this is critical to identify any bug or behavior with changing business environment.   
  
4) If you are writing core libraries which will be used by different project and with different teams. Follow **javadoc comment style** and document all assumption and precondition for using your API. Joshua Bloch has also mentioned about writing Java-doc comment in his classic Effective Java, which is worth knowing.  
  
5) **Include JIRA Number and description on comment,** especially if you are modifying an existing piece of code as part of maintenance. This I found extremely useful while comparing different version of code in CVS or SVN. This gives you clear idea why that particular code has been added and whether issue is because of that piece of code or not.  
  
  
6) Always try to **finish your comment in as few words** as possible, one liner comment is best until its explaining *"Why"* part and can't be replaced by code itself. No body likes or has enough time to read longer comment.  
  
7) **Don't write story in comment as your name**, employee id, your department etc because those information can be obtained from CVS commit data in case someone wants to know who has make this change.  
  
8) Always **put comment while committing code in source control repository** and especially why you are adding this piece of code if possible include JIRA or QC Number so that any one can refer JIRA for complete details.  
  
9) If you want upcoming developer to follow certain **standards** or inform about certain things then **include them in the beginning of your class as comment**. E.g. suppose if you are writing serializable class in java then its good to put a serializable alert stating that any new fields addition in this class must implement [serializable interface in java](http://javarevisited.blogspot.com/2011/04/top-10-java-serialization-interview.html) or make it transient etc.  
  
  
10) **Last but not the least give your code to fellow developer** **to understand** as part of code review and ask him how much he understands it.  
  
That’s all from me on code commenting, please share the standard, best practices or your experience with writing comments on code. I believe these are the areas which a junior developer or even we can improve and it’s only possible from learning which each mother's experience.  
Happy weekend :)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

http://javarevisited.blogspot.sg/2011/09/code-review-checklist-best-practice.html

Code Review Checklist and Best practices in Java

**What to review while doing Code review**  
Code Review and Unit testing is one of best development practice I always recommend, strive for and enforce as much as possible. Even just by doing code review and Junit test case always offer positive result it can be improved a lot by constantly learning with our mistakes , others mistakes and by observing how others are doing it. I always try to get my code review by some one with more experience so that I can capture any domain specific scenario which has been missed during think through process and some with less experience so that he can improve his code readability, have a four eye check and most importantly I found that when i explain my code to some one as part of code review I myself discover many things which can be improved or left out.Overall it always add something and improve code quality and reduce bug.

[code review checklist, code review best practices](http://javarevisited.blogspot.com/2011/08/enum-in-java-example-tutorial.html)I also review some one else code and voluntary take part to **improve my code** understanding ability and offer help to others, In this article I will list things which I look while doing code review. These are the things which I have been accumulated over the years but I also look forward to you guys to contribute your experience, best practices for code review and suggest how you guys do code review. These tips are independent of language and equally apply to Java, .NET or C++ code.

### 10 points checklist on Code Review

1) Does Code meet **functional requirement**: first and foremost does code meets all requirements which it should met, point out if anything has been left out.

[](http://2.bp.blogspot.com/-gj7a3WnaxDI/TeJPgQFi0uI/AAAAAAAAAL0/UjLVk0qaAfQ/s1600/2.jpg)

2) Is there any **Side effect of this change**: Some time one change in your system may cause bug in other upstream and downstream system and it’s quite possible that new developer or anyone who is writing code might not be available of that dependency. This often directly related to experience in project and I found that the more you know about system and its environment better you able to figure this out.

3) **Concurrency**: does code is thread-safe? Does it have properly synchronized if using shared resource? Does it free of any kind of deadlock or live-lock? Concurrency bugs are hard to detect and often surfaces in production. *Code review* is one place where you can detect this by carefully understand design and its implementation.

4) **Readability and maintenance**:  does code is readable? Or is it too complicate for some-one complete new. Always give value to readability as code is not just for this time it will remain there for long time and you need to read it many times. Another important aspect is maintenance as most of software spends 90% time in maintenance and only 10% time on development it should be maintainable and flexible in first place. You can verify that whether code is configurable or not, look for any hard coding, find out what is going to be changed in near future etc.

5) **Consistency**: This is part of point 4 but I have made it another separate point because of its importance. This is the best thing you can have in your code which automatically achieves readability. Since many developer and programmer take part in project and they have there own style of coding, it’s in best interest of everybody to form a coding standard and follow it on letter and spirit. For example it’s not good someone using function initialize() and other is using init() for same kind of operation, keep you *code consistent* and it will look better, read better.

6) **Performance**: Another important aspect most important if you are writing high volume low latency electronic trading platform for high frequency trading which strives for micro second latency. Carefully monitor which code is going to execute at start-up and which is going to be executed in loop or multiple times, optimize the code which is going to execute more often.

7) **Exception handling**: Ask does code handles bad input and exception? It should and that too with predefined and standard way which must be available and documented for support purpose. I put this point well above on my chart while doing review because failing on this point can lead your application crash and not able to recover from fault on other system or other part of same application.

8) **Simplicity**: Always see if there is any simple and elegant alternative available at-least give a thought and try. Many times first solution comes in mind is not best solution so giving another thought is just worth it.

9) **Reuse of existing code**: See if the functionality can be achieved by using existing code, advantage of doing this is that you are using tried and tested code which reduce your QA time and also give you more confidence. Introducing new libraries introduce new dependency. I prefer not to try anything fancy until it’s absolutely necessary.

10) **Unit test**: Check whether enough JUnit test cases have been written and cover sufficient percentage of new code. never let you pass the code without Junit test because developer often make excuse of time but believe me its worth to write it.

Not last but least that put a comment on your java file that by whom it has been reviewed, what issue finds are out and status of those. This will make the whole process official and ensures that due diligence would be applied during code review. Its also good to maintain your own **code review checklist** or a project wise code review checklist and use it every time while doing review. There are so many **best practices** but I have only included those which I follow and found interesting but as I had said earlier this is the area which always needs improvement and no body is perfect on that. So please contribute your ideas on **code review** and effective development.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://javarevisited.blogspot.com.au/2013/01/java-best-practices-method-overloading-constructor.html>

Method and Constructor Overloading Best Practices in Java

You need to be careful while overloading a method in Java, especially after introduction of autoboxing in Java 5. Poorly overloaded method not only adds confusion among developers who use that but also they are error prone and leaves your program on compiler's mercy to select proper method. One of the best example of poorly overloaded method is remove() method from ArrayList. There are two versions of remove, first one which takes an Object as argument i.e. remove(Object element) and second one, which takes an index as argument i.e. remove(int index). It worked fine until Java 1.4 where there is clearly a distinction between primitive types and objects type but in Java 1.5, where you can pass an int primitive to a method which accepts an Integer object, creates some nasty problem. Now suppose you have an ArrayList of Integer with values 1, 2 and 3, and you call remove(1) then which method will be called? JVM can interpret 1 as index also or 1 as Integer object also.  
  
  
It's best to avoid issues related to method overloading by following some Java best practices. For those who doesn’t know what is method overloading in Java? method overloading means declaring more than one method with same name but different [method signatures](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html). This is generally done to create methods which does same thing but with different types. For example, one of the most popular example of method overloading is System.out.println() method, which is overloaded to accept different types of parameters like String, double, int etc, see this Java tutorial on [method overloading](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html) and [static vs dynamic binding](http://javarevisited.blogspot.sg/2012/03/what-is-static-and-dynamic-binding-in.html) for more details. By the way all of these Java best practices which are explained in context of method overloading are equally applicable to constructor overloading in Java, because in terms of overloading method and constructors are almost same.

## Java Best Practices - Method Overloading

[Java best practices for method and constructor overloading in Java](http://3.bp.blogspot.com/-K6q0DQ1v-tw/TWu8owBtc2I/AAAAAAAAADA/oBoHDBiJ8ag/s1600/17.jpg)Here are some of the common things which you can remember while overloading method or constructor in Java. These Java best practices are completely based upon experience and you may have some more to add on this list. let’s see my list of Java best practices while overloading method in Java.

**1) Don't overload method which accept same number of parameter with similar types**

Two overloaded method which accept same number of argument with similar types i.e. which follow same [type hierarchy](http://javarevisited.blogspot.sg/2012/12/what-is-type-casting-in-java-class-interface-example.html) is most common mistake while overloading method in Java. For example, find out which version of overloaded method will be invoked in following scenario :

**import** java.util.ArrayList;  
**import** java.util.LinkedList;  
**import** java.util.List;  
  
/\*\*  
 \* Java program to demonstrate some best practice to following while overloading

\* method in Java.This Java program shows a case of confusing method overloading in Java

\*  
 \* @author Javin Paul  
 \*/  
**public** **class** OverloadingTest {  
    
    **public** **static** **void** main(**String** args[]){  
       **List** abc = **new** **ArrayList**();  
       **List** bcd = **new** **LinkedList**();  
        
       ConfusingOverloading co = **new** ConfusingOverloading();  
       co.hasDuplicates(abc); *//should call to ArryList overloaded method*  
       co.hasDuplicates(bcd); *//should call to LinkedList overloaded method*  
    }  
  
    
}  
  
**class** ConfusingOverloading{  
    
    **public** **boolean** hasDuplicates (**List** collection){  
        **System**.out.println("overloaded method with Type List ");  
        **return** **true**;  
    }  
    
    **public** **boolean** hasDuplicates (**ArrayList** collection){  
        **System**.out.println("overloaded method with Type ArrayList ");  
        **return** **true**;  
    }  
    
    
    **public** **boolean** hasDuplicates (**LinkedList** collection){  
        **System**.out.println("overloaded method with Type LinkedList ");  
        **return** **true**;  
    }  
    
}  
  
**Output**  
overloaded method with **Type** **List**  
overloaded method with **Type** **List**

To surprise of some programmers method with argument type List is called both the time, instead of expected method which takes ArrayList and LinkedList, because method overloading is resolved at compile time using static binding in Java. This is also one of the reason, why its important to clearly understand [difference between method overloading and overriding in Java](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html). Here expected case is result of mistaking overloading as overriding, which work on actual object and happens at runtime. To know more about static and dynamic binding in Java , you can also see my post difference between static and dynamic binding in Java.

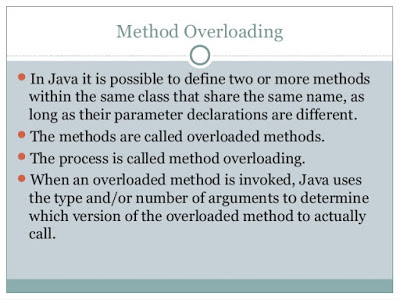
**2) Use radically different types while overloading method in Java**

It's completely legal and there is no ambiguity when two overloaded method accepts radically different types like String and Integer. Though both overloaded method will accept only one parameter, it’s still clear which method is called because both types are completely different to each other. Both programmer and compiler both know which method will be invoked for a particular call. One of the example of this kind of overloading is constructor of java.util.Scanner class which accepts File, InputStream or String as parameter, as shown below :

Scanner(File source)

Scanner(InputStream source)

Scanner(String source)

[](http://2.bp.blogspot.com/-oqQ1WUjQqb8/VXLspowzixI/AAAAAAAAC6Y/vJKmD0omMjA/s1600/method-overloading-21.jpg)

**3) Beware of Autoboxing while overloading method in Java**

Prior to introduction of [Autoboxing and unboxing in Java 5](http://javarevisited.blogspot.sg/2012/07/auto-boxing-and-unboxing-in-java-be.html), method which accept primitive type and object type were radically different and it’s clear which method will be invoked. Now with autoboxing it's really confusing. Clasical example of this kind overloading mistake is ArrayList’s remove() method, which is overloaded to accept index as well as Object. when you store Integer in ArrayList and call remove() method, It’s hard to find out which remove() method will be called, as shown in below example :

**List**<**Integer**> numbers = **new** **ArrayList**<**Integer**>();  
numbers.add(1);  
numbers.add(2);  
numbers.add(3);  
**System**.out.println("numbers: " + numbers);  
numbers.remove(1); *//should remove "1" as element or 2nd element from ArrayList*  
**System**.out.println("numbers: " + numbers);  
  
**Output:**  
numbers: [1, 2, 3]  
numbers: [1, 3]

Many Java programmer expect that Integer(1) object would be removed but since remove() is overloaded, compiler choose remove(int) over remove(Object). Rules of which overloaded method gets chosen in case of autoboxing is complex and hard to remember, so Its best to avoid two overloaded method where one accept Object and other accept primitive type. If by any chance you must have to do this then make sure both of them perform identical function.

Other **Java best practices articles** from Javarevisited Blog

* [Top 10 JDBC Best practices for Java programmer](http://javarevisited.blogspot.sg/2012/08/top-10-jdbc-best-practices-for-java.html)
* [10 Best practices to follow while writing code comments](http://javarevisited.blogspot.sg/2011/08/code-comments-java-best-practices.html)
* [JUnit best practices for better unit testing](http://javarevisited.blogspot.sg/2012/08/best-practices-to-write-junit-test.html)
* [Why use @Override annotation in Java – coding best practice](http://javarevisited.blogspot.com/2012/11/why-use-override-annotation-in-java.html)
* [Java best practice while dealing with passwords in application](http://javarevisited.blogspot.sg/2012/05/best-practices-while-dealing-with.html)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://javarevisited.blogspot.sg/2012/08/top-10-jdbc-best-practices-for-java.html>

Top 10 JDBC Best Practices for Java Programmer

**Java JDBC Best practices**

JDBC Best Practices are some coding practices which Java programmer should follow while writing JDBC code. As discussed in [how to connect to Oracle database from Java](http://javarevisited.blogspot.sg/2012/04/java-program-to-connect-oracle-database.html), JDBC API is used to connect and interact with a Database management System. We have touched some of the JDBC best practices in our last article [4 JDBC Performance tips](http://javarevisited.blogspot.sg/2012/01/improve-performance-java-database.html), On which we have discussed simple tips to improve performance of Java application with database. By using JDBC you can execute DDL, DML and Stored Procedures. JDBC Best practices is probably most significant set of coding practices in Java because it significantly affect performance of Java application. I have seen substantial performance gain by simply following common *JDBC best practices* like running queries with auto commit mode disable. One of the query which we used in our example of JDBC Batch update was taking almost 30 second to finish with auto commit mode enabled but it just took under one second with auto commit mode disable and using explicit commit. This JDBC tutorial is collection of such practices which help you to write better JDBC code and in most cases result in improved performance.

## 10 JDBC Best pratices in Java

[JDBC Best practices Java coding database](http://3.bp.blogspot.com/-K6q0DQ1v-tw/TWu8owBtc2I/AAAAAAAAADA/oBoHDBiJ8ag/s1600/17.jpg)Here is my list of top 10 JDBC best practices in Java which helps to avoid potential error, to get better performance and helps to write robust Java database connection code.

**JDBC Best Practice #1: Use PreparedStatement**

This is by far most popular JDBC practices suggested by everyone who has worked in JDBC API in Java. Indeed PreparedStatement deserve that admiration because of useful services it provides like *prevention from SQL injection*, Precompiled SQL queries and use of bind variables as discussed in [why Use PreparedStatement in Java](http://javarevisited.blogspot.sg/2012/03/why-use-preparedstatement-in-java-jdbc.html)

**JDBC Best Practice #2: Use ConnectionPool**

ConnectionPool as JDBC best practice has already gained recognition and it even become standard now days. Several framework provides in built connection Pool facility like [Database Connection Pool in Spring](http://javarevisited.blogspot.sg/2012/06/jdbc-database-connection-pool-in-spring.html), DBCP and if you are running in managed environment like [J2EE Application Server](http://javarevisited.blogspot.sg/2012/05/5-difference-between-application-server.html) e.g. WAS or JBOSS, Server will provide Connection Pool facility. rational behind this JDBC best practices is that Creating JDBC connection take relatively longer time which can increase overall response time, by caching JDBC connection in pool application can immediately access database.

**JDBC Best Practice #3: Disable auto commit mode**

This is one of those JDBC best practices which provided substantial performance gain in our JDBC batch update example. Its recommended to run SQL query with auto commit mode disable. Rational behind this JDBC best practice is that with auto commit mode disabled you can [group SQL Statement in one transaction](http://javarevisited.blogspot.sg/2011/11/database-transaction-tutorial-example.html) while in case of auto commit mode every SQL statement runs in its own transaction and committed as soon as it finishes. So always run queries with auto commit mode disabled

**JDBC Best Practice #4: Use JDBC Batch Update**

This is another JDBC best practice which is very popular. JDBC API provides addBatch() method to add [SQL queries](http://javarevisited.blogspot.sg/2012/07/subquery-example-in-sql-correlated-vs.html) into batch and executeBatch() to send batch queries for execution. Rational behind this JDBC best practices is that, JDBC batch update potentially reduce number of database roundtrip which result in significant performance gain. So always Use JDBC batch update for insertion and update queries.

**JDBC Best Practice #5: Access ResultSet using column name to avoid invalidColumIndexError**

JDBC API allows to access data returned by [SELECT query](http://javarevisited.blogspot.sg/2011/10/selct-command-sql-query-example.html) using ResultSet, which can further be accessed using either column name or column index. This JDBC best practice suggest using column name over column index in order to [avoid InvalidColumnIndexException](http://javarevisited.blogspot.sg/2012/01/javasqlsqlexception-invalid-column.html) which comes if index of column is incorrect, most common of them is 0, since ResultSet column Index starts from 1, zero is invalid. Also you don't need to change your JDBC access code if order of column changed in SELECT SQL query, which is a major maintenance gain and a robust way to write JDBC code. Some Java programmer may argue that accessing column using index is faster than name, which is true but if you look in terms of maintenance, robustness and readability, I prefer accessing column using name in ResultSet [Iterator](http://javarevisited.blogspot.sg/2011/10/java-iterator-tutorial-example-list.html).

**JDBC Best Practice #6: Use Bind variables instead of String concatenation**

In JDBC Best Practice #1 we have suggest to [use PreparedStatement in Java](http://javarevisited.blogspot.sg/2012/03/why-use-preparedstatement-in-java-jdbc.html) because of better performance. But performance can only be improved if you use *bind variables* denoted by *? or place holders*. which allows database to run same query with different parameter. This JDBC best practices also result in better performance and also provide protection against SQL injection.

**JDBC Best Practice #7: Always close Statement, PreparedStatement and Connection.**

Nothing new on this JDBC Best practice. Its common Java coding practice to close any resource in finally block as soon as you are done with that. JDBC Connection and other JDBC classes are costly resource and should be closed in finally block to ensure release of connection even in case of any SQLException. From Java 7 onwards you can use [Automatic Resource Management (ARM) Block](http://javarevisited.blogspot.sg/2011/09/arm-automatic-resource-management-in.html) to close resources automatically.

**JDBC Best Practice #8: Choose suitable JDBC driver for your application**

There are 4 typs of JDBC driver in Java and it can directly affect the performance of DAO layer. always use latest JDBC Driver if available and prefer [type 4 native JDBC Drivers](http://javarevisited.blogspot.sg/2012/05/different-types-of-jdbc-drivers-in-java.html).

**JDBC Best Practice #9: Use standard SQL statement and avoid using db specific query until necessary**

This is another JDBC best practice in Java which ensures writing portable code. Since most of JDBC code is filled up with [SQL query](http://javarevisited.blogspot.sg/2012/07/subquery-example-in-sql-correlated-vs.html) its easy to start using Database specific feature which may present in [MySQL](http://javarevisited.blogspot.sg/2010/10/frequently-used-mysql-commands-part-3.html) but not in Oracle etc. By using ANSI SQL or by not using DB specific SQL you ensure minimal change in your DAO layer in case you switch to another database.

**JDBC Best Practice #10: Use correct getXXX() method**

This is the last JDBC best practice in this article which suggest using correct getter while getting data from ResultSet to avoid data conversion even though JDBC allows to get any data type using getString()or getObject().

That's all on *JDBC best practices for Java Programmer*, I am sure there are many more JDBC best practices around but these are most common practices which I can think of. let us know if you are familiar with any other JDBC best practice.

Other **Development Best practices** article from Javarevisited blog

[10 best practices to follow while writing code comments](http://javarevisited.blogspot.sg/2011/08/code-comments-java-best-practices.html)

[10 Object oriented design principles Java programmer know](http://javarevisited.blogspot.sg/2012/03/10-object-oriented-design-principles.html)

[How to write production quality code in Java](http://javarevisited.blogspot.sg/2011/09/how-to-write-production-quality-code.html)

[Code review best practices and checklist for programmer](http://javarevisited.blogspot.sg/2011/09/code-review-checklist-best-practice.html)

[Junit Best practices for Java programmer](http://javarevisited.blogspot.sg/2012/08/best-practices-to-write-junit-test.html)

[20 Design pattern interview questions answers in Java](http://javarevisited.blogspot.sg/2012/06/20-design-pattern-and-software-design.html)

You might like:

* [Difference between SOAP and RESTful Web Service in Java](http://javarevisited.blogspot.com/2015/08/difference-between-soap-and-restfull-webservice-java.html)
* [How to Fix java.net.ConnectException: Connection refused: connect in Java](http://javarevisited.blogspot.com/2013/02/java-net-ConnectException-Connection-refused.html)
* [Top 50 Programmer Phone Interview Questions with Answers](http://javarevisited.blogspot.com/2015/02/50-programmer-phone-interview-questions-answers.html)
* [How to create Immutable Class and Object in Java - Tutorial Example](http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html)
* [JUnit best practices for writing better test](http://javarevisited.blogspot.sg/2012/08/best-practices-to-write-junit-test.html)
* [10 best practices to follow while writing code comments in Java](http://javarevisited.blogspot.sg/2011/08/code-comments-java-best-practices.html)
* [4 tips to improve JDBC performance in Java](http://javarevisited.blogspot.sg/2012/01/improve-performance-java-database.html)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://javarevisited.blogspot.sg/2012/06/common-cause-of-javalangnullpointerexce.html>

java.lang.NullPointerException - Common cause of NullPointerException in Java Example

java.lang.NullPointerException or NullPointerException in Java is probably the first Exception you will face in Java. It is true nightmare for beginners in Java but pretty easy to solve once you get familiar with Exception handling in Java. What makes NullPointerException little tricky is its name which has pointer in itself and Java does not support pointers like [multiple inheritance in Java](http://javarevisited.blogspot.in/2011/07/why-multiple-inheritances-are-not.html) . In this article we will see What is NullPointerException in Java, How to solve Exception in thread "main" java.lang.NullPointerException, finding possible cause of Java NullPointerException and how to troubleshoot NPE in Java. Based on my experience once you know little bit about NullPointerException its pretty easy to solve. By the way, as said, "prevention is better than cure", you can [avoid != null check and NullPointerException](http://javarevisited.blogspot.com/2013/05/ava-tips-and-best-practices-to-avoid-nullpointerexception-program-application.html) by following some best practices.

## What is NullPointerException in Java

NullPointerException in Java is nothing but an error or precisely an exception which occurs if we tried to perform any operation on object which is null. In Java reference variable points to object created in heap but when you create a reference variable of type object by default its point to "null" and when you try to call any method on null, try to access any variable on null you will get this null pointer exception. no matter by what reason your reference variable is pointing to null be it not initialized or set to null if you perform any operation it will throw [Exception in thread main : java.lang.NullPointerException](http://java67.blogspot.com/2012/09/what-is-nullpointerexception-in-java.html)

## When does NullPointerException occurs in Java

Javadoc of java.lang.NullPointerException has outlined scenario when it could be occurred:

1) When you call instance method on a null object. you won't get null pointer exception if you call static method or class method on null object because static method doesn't require an instance to call any method.

2) While accessing or changing any variable or field on null object.

3) Throwing null when an Exception is expected to throw.

4) When calling length of array when array is  null.

5) Accessing or changing slots of null just like an array.

6) When you try to synchronize on null object or using null inside synchronized block in Java

we will see examples of NullPointerException for each of above scenario to get it right and understand it better.

## Common cause of NullPointerException in Java as Example

Based upon my experience java.lang.NullPointerException repeats itself on various format, I have collected most common cause of  java.lang.NullPointerException in java code and explained them here, we will use following Trade class for example :

**public** **class** Trade {  
    **private** **String** symbol;  
    **private** **int** price;  
    **public** **static** **String** market;  
  
    **public** Trade(**String** symbol, **int** price) {  
        **this**.symbol = symbol;  
        **this**.price = price;  
    }  
  
    **public** **int** getPrice() {  
        **return** price;  
    }  
  
    **public** **void** setPrice(**int** price) {  
        **this**.price = price;  
    }  
  
    **public** **String** getSymbol() {  
        **return** symbol;  
    }  
  
    **public** **void** setSymbol(**String** symbol) {  
        **this**.symbol = symbol;  
    }  
  
}

1)  Java  NullPointerException while calling instance method on null object

This is probably the most common case of this error, you call method on some object and found that reference is null, always perform null check if you see possibility of null before calling any method on object.

Trade pennyStock = **null**;  
pennyStock.getPrice(); *//this will throw NullPointerException*  
  
**Exception** in thread "main" java.lang.**NullPointerException**  
at test.NullPointerExceptionTest.main(NullPointerExceptionTest.java:23)

2) NullPointerException in Java while accessing field on null reference.

Trade fxtrade = **null**;  
**int** price = fxtrade.price; //here fxtrade is null, you can’t access field here  
  
**Exception** in thread "main" java.lang.**NullPointerException**  
at test.NullPointerExceptionTest.main(NullPointerExceptionTest.java:64)

**3) java.lang.NullPointerException when throwing null as exception.**

If you throw an Exception object and if that is null you will get null pointer exception as shown in below example

**RuntimeException** nullException = **null**;  
**throw** nullException;  
  
**Exception** in thread "main" java.lang.**NullPointerException**  
at test.NullPointerExceptionTest.main(NullPointerExceptionTest.java:74)

4)example of NullPointerException when getting length of an array which is null.

Trade[] bluechips = **null**;  
**int** length = bluechips.length;  //array is null here  
  
**Exception** in thread "main" java.lang.[**NullPointerException**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/NullPointerException.html)  
at test.NullPointerExceptionTest.main(NullPointerExceptionTest.java:85)

5) Example of NPE when accessing element of a null array.

Trade[] bluechips = **null**;  
Trade motorola = bluechips[0]; //array is null here  
  
**Exception** in thread "main" java.lang.**NullPointerException**  
at test.NullPointerExceptionTest.main(NullPointerExceptionTest.java:94)

6) You will also get NullPointerException in Java if you try to synchronize on null object or try to use null object inside synchronized block in Java.

Trade highbetaTrade = **null**;  
**synchronized**(highbetaTrade){  
**System**.out.print("This statement is synchronized on null");  
}

**Exception** in thread "main" java.lang.**NullPointerException**  
at test.NullPointerExceptionTest.main(NullPointerExceptionTest.java:104)

## How to solve NullPointerException in Java

To solve a NullPointerException in Java first we need to find cause, which is very easy just look the stack-trace of NullPointerException and it will show exact line number where NPE has occurred. now go to that line and look for possible object operation like *accessing field*, calling method or throwing exception etc, that will give you an idea which object is null. Now once you found that which object is null job is half done , now find out why that object is null and solve the java.lang.NullPointerException. , This second part always vary sometime you get null object from factory or sometime some other thread might have set it null, though using [Assertion](http://javarevisited.blogspot.sg/2012/01/what-is-assertion-in-java-java.html) in early phase of development you can minimize chances of java.lang.NullPointerException but as I said its little bit related to environment and can come on production even if tested fine in test environment. Its best to avoid NullPointerException by applying careful or defensive coding technique and null safe API methods.

### When in Java Code NullPointerException doesn't come

1) When you access any static method or static variable with null reference.

If you are dealing with static variables or static method than you won't get null pointer exception even if you have your reference variable pointing to null because static variables and method call are bonded during [compile time](http://javarevisited.blogspot.com/2012/03/what-is-static-and-dynamic-binding-in.html) based on class name and not associated with object. for example below code will run fine and not throw NullPointerException because "market" is an [static variable](http://javarevisited.blogspot.com/2011/11/static-keyword-method-variable-java.html) inside Trade Class.

Trade lowBetaTrade = **null**;  
**String** market = lowBetaTrade.market; *//no NullPointerException market is static variable*

## Important points on NullPointerException in Java

1) NullPointerException is an [unchecked exception](http://javarevisited.blogspot.com/2011/12/checked-vs-unchecked-exception-in-java.html) because its extends RuntimeException and it doesn’t mandate try catch block to handle it.

2) When you get NullPointerException look at line number to find out which object is null, it may be object which is calling any method.

3) Modern IDE like Netbeans and Eclipse gives you hyper link of line where NullPointerException occurs

4) You can set an [Exception break point in Eclipse](http://javarevisited.blogspot.com/2011/07/java-debugging-tutorial-example-tips.html) to suspend execution when NullPointerException occurs read 10 tips on java debugging in Eclipse more details.

5) Don't forget to see name of [Thread](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) on which NullPointerException occurs. in multi-threading NPE can be little tricky if some random thread is setting reference to null.

6) Its best to avoid NullPointerException while coding by following some coding best practices or putting null check on database as constraint.

That’s all on **What is java.lang.NullPointerException**, When it comes and how to solve it. In next part of this tutorial we will look on some best java coding practices to avoid NullPointerException in Java.

Other **Java debugging tutorial**

[How to remote debug java application in Eclipse IDE](http://javarevisited.blogspot.com/2011/02/how-to-setup-remote-debugging-in.html)

[How to fix java.lang.ClassNotFoundException in Java](http://javarevisited.blogspot.com/2011/08/classnotfoundexception-in-java-example.html)

[Difference between NoClassDefFoundError and ClassNotFoundException in Java](http://javarevisited.blogspot.com/2011/07/classnotfoundexception-vs.html)

[How to solve bad version number in class file error](http://javarevisited.blogspot.com/2011/12/bad-version-number-in-class-files-cause.html)

[How to write production quality code in Java](http://javarevisited.blogspot.sg/2011/09/how-to-write-production-quality-code.html)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*