2013 Java Revisited

Difference between Stack vs Heap in Java

Here are few differences between stack and heap memory in Java:

1) Main difference between heap and stack is that stack memory is used to store [local variables](http://javarevisited.blogspot.com/2012/02/difference-between-instance-class-and.html) and function call, while heap memory is used to store objects in Java. No matter, where object is created in code e.g. as member variable, local variable or class variable,  they are always created inside heap space in Java.

2) Each [Thread in Java](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) has there own stack which can be specified using -Xss JVM parameter, similarly you can also specify heap size of Java program using JVM option -Xms and -Xmx where -Xms is starting size of heap and -Xmx is maximum size of java heap. to learn more about JVM options see my post [10 JVM option Java programmer should know](http://javarevisited.blogspot.com/2011/11/hotspot-jvm-options-java-examples.html).

3) If there is no memory left in stack for storing function call or local variable, JVM will throw java.lang.StackOverFlowError, while if there is no more heap space for creating object, JVM will throw java.lang.OutOfMemoryError: Java Heap Space. Read more about how to deal with java.lang.OutOfMemoryError  in my post [2 ways to solve OutOfMemoryError in Java](http://javarevisited.blogspot.com/2011/09/javalangoutofmemoryerror-permgen-space.html).

4) If you are using [Recursion](http://javarevisited.blogspot.com/2012/12/recursion-in-java-with-example-programming.html), on which method calls itself, You can quickly fill up stack memory. Another difference between stack and heap is that size of stack memory is lot lesser than size of  heap memory in Java.

5) Variables stored in stacks are only visible to the owner Thread, while objects created in heap are visible to all thread. In other words stack memory is kind of private memory of Java Threads, while heap memory is shared among all threads.

That's all on **difference between Stack and Heap memory in Java**. As I said, It’s important to understand what is heap and what is stack in Java and which kind of variables goes where, how you can run out of stack and heap memory in Java etc. Let us know if you are familiar with any other difference between stack and heap memory in java.

**10 XML Interview questions and answers for Java Programmer**Here is my list of some common and frequently asked Interview questions on XML technologies.  Questions on this list is not very tough but touches some important areas of XML technologies e.g. DTD, XML Schema, XSLT transformations, [XPATH evaluation](http://javarevisited.blogspot.sg/2012/12/create-and-evaluate-xpath-java-example-tutorial-program.html), XML binding, XML parsers and fundamentals of XML e.g. namespace, validation, attribute, elements etc.

Question 1: What is XML ?

Answer : XML stands for Extensible Markup language which means you can extend XML based upon your needs. You can define custom tags like <books>, <orders> etc in XML easily as opposed to other mark-up language like HTML where you need to work with predefined tags e.g. <p> and you can not use user defined tag. Though structure of XML can be standardize by making use of DTD and XML Schema. XML is mostly used to transfer data from one system to another e.g. between client and server in enterprise applications.

Question 2: Difference between DTD and XML Schema?

Answer : There are couple of differences between DTD and XML Schema e.g. DTD is not written using XML while XML schema are xml documents in itself, which means existing XML tools like [XML parsers](http://javarevisited.blogspot.sg/2011/12/parse-read-xml-file-java-sax-parser.html) can be used to work with XML schema. Also XML schema is designed after DTD and it offer more types to map different types of data in XML documents. On the other hand DTD stands for Document Type definition and was a legacy way to define structure of XML documents.

Question 3: What is XPath ?

Answer : XPath is an XML technology which is used to retrieve element from XML documents. Since XML documents are structured, XPath expression can be used to locate and retrieve elements, attributes or value from XML files. XPath is similar to SQL in terms of retrieving data from XML but it has it's own syntax and rules. See here to know more about [How to use XPath to retrieve data from XML documents](http://javarevisited.blogspot.sg/2012/12/xpath-tutorial-example-how-to-select-elements.html).

Question 4: What is XSLT?

Answer : XSLT is another popular XML technology to transform one XML file to other XML, HTML or any other format. XSLT is like a language which specifies its own syntax, functions and operator to transform XML documents. Usually transformation is done by XSLT Engine which reads instruction written using XSLT syntax in XML style sheets or XSL files. XSLT also makes extensive use of [recursion](http://javarevisited.blogspot.sg/2012/12/recursion-in-java-with-example-programming.html) to perform transformation. One of the popular example of using XSLT is for displaying data present in XML files as HTML pages. XSLT is also very handy to transforming one XML file into another XML document.

Question 5: What is element and attribute in XML?

Answer : This can be best explained by an example. let's see a simple XML snippet

<Orders>  
  <Order id="123">  
     <Symbol> 6758.T</Symbol>  
     <Price> 2300</Price>  
  <Order>  
<Orders>

In this sample XML id is an attribute of <Order> element. Here <Symbol>, <Price> and <Orders> are also other elements but they don't have any attribute.

Question 6: What is meaning of well formed XML ?

Answer : Another *interesting XML interview question* which most appeared in telephonic interviews. A well formed XML means an XML document which is syntactically correct e.g. it has a root element, all open tags are closed properly, attributes are in quotes etc.  If an XML is not well formed, it may not be processed and parsed correctly by various XML parsers.

Question 7: What is XML namespace? Why it's important?

Answer : XML namespace are similar to [package in Java](http://java67.blogspot.sg/2012/08/what-is-package-in-java-how-to-use.html) and used to provide a way to avoid conflict between two xml tags of same name but different sources. XML namespace is defined using xmlns attribute at top of the XML document and has following syntax  xmlns:prefix="URI". later that prefix is used along with actual tag in XML documents. Here is an example of using XML namespace :

<root xmlns:inst="http://instruments.com/inst"  
  <inst:phone>  
      <inst:number>837363223</inst:number>  
   </inst:phone>  
</root>

Question 8: Difference between DOM and SAX parser ?

Answer : This is another very popular XML interview question, not just in XML world but also on Java world. Main difference between DOM and SAX parser is the way they parse XML documents. DOM creates an in memory tree representation of XML documents during parsing while SAX is a event driven parser. See [Difference between DOM and SAX parser](http://javarevisited.blogspot.sg/2011/12/difference-between-dom-and-sax-parsers.html) for more detailed answer of this question.

Question 9: What is a CDATA section in XML?

Answer : I like this XML Interview questions for its simplicity and importance, yet many programmer doesn't know much about it. CDATA stands for character data and has special instruction for XML parsers. Since XML parser parse all text in XML document e.g. <name>This is name of person</name>  here even though value of tag <name> will be parsed because it may contain XML tags e.g. <name><firstname>First Name</firstname></name>. CDATA section is not parsed by XML parser. CDATA section starts with "<![CDATA[" and finishes with "]]>".

Question 10: What is XML data Binding in Java?

Answer : XML binding in Java refers to creating Java classes and object from XML documents and then [modifying XML documents](http://javarevisited.blogspot.sg/2011/12/parse-xml-file-in-java-example-tutorial.html) using Java programming language. JAXB , Java API for XML binding provides convenient way to bind XML documents with Java objects. Other alternatives for XML binding is using open source library e.g. XML Beans. One of the biggest advantage of XML binding in Java is to leverage Java programming capability to create and modify XML documents.

This list of XML Interview questions and answers are collected from programmers but useful to anyone who is working in XML technologies. Important of XML technologies like XPath, XSLT, XQuery is only going to increase because of platform independent nature of XML and it's popularity of transmitting data over cross platform. Though XML has disadvantage like verbosity and size but its highly useful in web services and transmitting data from one system to other where bandwidth and speed is of secondary concern.

**JDBC Batch INSERT and UPDATE example in Java with PreparedStatement**  
  
JDBC API in Java allows program to batch insert and update data into database, which tends to provide better performance by simple virtue of fact that it reduce lot of database round-trip which eventually improves overall performance. In fact it’s one of [JDBC best practices](http://javarevisited.blogspot.sg/2012/08/top-10-jdbc-best-practices-for-java.html) to insert and update data in batches. For those who doesn’t know what is *batch insert and update*, Java provides several ways to execute SQL queries, one of them is JDBC batch insert and update, on which instead of executing sql query one by one using either Statement or [PreparedSatement](http://javarevisited.blogspot.sg/2012/03/why-use-preparedstatement-in-java-jdbc.html), you execute query in batch and send a batch of query to database for execution instead of single query. Since multiple queries are combined into batch and one batch is sent to database instead of individual queries, it reduce database round trip by factor of batch size. Batch size can be anything but needs to be decided carefully. JDBC specification supports upto 100 but individual database e.g. [Oracle](http://javarevisited.blogspot.sg/2012/12/top-10-oracle-interview-questions-and-answers-database-sql.html), MySQL, Sybase or SQL Server has there own limit on maximum batch size, , normal jdbc batch size ranges from 50 to 100. JDBC API provides addBatch() method to add queries into batch and than later execute them using executeBatch() method. Both Statement andPreparedStatement can be used to execute batch queries in Java. By the way batch insert and update also provide performance boost to [Data access Object or DAO layer](http://javarevisited.blogspot.sg/2013/01/data-access-object-dao-design-pattern-java-tutorial-example.html),  as discussed in our last post [4 ways to improve Performance of JDBC applications](http://javarevisited.blogspot.sg/2012/01/improve-performance-java-database.html).

How to run batch insert and update in JDBC

There are multiple ways you can run batch queries in Java application, You have choice of using plain old JDBC or you can leverage Spring's JdbcTemplate Utility class. Though both Statement and PreparedStatment can execute batch queries, It’s better to use PreparedStatement because of several benefits it provides including improved performance and prevention from SQL injection assuggested on [Why you should use PreparedStatement in Java](http://javarevisited.blogspot.sg/2012/03/why-use-preparedstatement-in-java-jdbc.html). In next section, we will compare performance of same INSERT SQL query when running as without batch and running as batch insert query. In both cases we will use PreparedStatement to make testing similar.

**SQL query without JDBC batch update using PreparedStatement**

Here is an example of running SQL query without using JDBC batch update. Performance of this example can be used to compare how JDBC Batch update perform.

*//query for inserting batch data*  
        **String** query = "insert into employee values (?,?,NULL)";  
        **PreparedStatement** pStatement = conn.prepareStatement(query);  
        
        **long** startTime = **System**.currentTimeMillis();  
        **for**(**int** count = 0; count < 1000; count++ ){  
            pStatement.setString(1, **Integer**.toString(count));  
            pStatement.setString(2, "Employee"+count);  
            pStatement.executeUpdate();  
        }  
        **long** endTime = **System**.currentTimeMillis();  
        **long** elapsedTime = (endTime - startTime)/1000; *//in seconds*  
        **System**.out.println("Total time required to execute 1000 SQL INSERT queries using PreparedStatement without JDBC batch update is :" + elapsedTime);  
  
**Output:**  
Total time required to execute 1000 queries using **Statement** without JDBC batch update is :38

So it took 38 seconds to insert 1000 records on employee table on [MySQL database](http://javarevisited.blogspot.sg/2010/10/frequently-used-mysql-commands-part-1.html) running on localhost. Yes, indeed its quite high but don't bother about absolute number yet, what is important here is to find out whether JDBC batch insert or update gives better performance or not. By the way above example uses PreparedStatement and bind variables to ensure [standard JDBC practices](http://javarevisited.blogspot.sg/2012/08/top-10-jdbc-best-practices-for-java.html) are followed.

**JDBC Batch INSERT example using PreparedStatement**

Now, let’s run same set of SQL query as JDBC batch INSERT. In this example, instead of running every SQL INSERT query as executeUpdate() , we are adding them in a batch using addBatch() method and once we reaches batch size, which is 100 here, we send them to database using executeBatch() method of JDBC API.

**import** java.sql.Connection;  
**import** java.sql.DriverManager;  
**import** java.sql.PreparedStatement;  
**import** java.sql.SQLException;  
  
/\*\*  
  \* **Java program to demonstrate JDBC Batch Insert example**. Inserting data in batch   
  \* seems to improve performance a lot. executeBatch() method of PreparedStatement is  
  \* used to run batch queries in Java JDBC.  
  \*/  
**public** **class** MySQLJdbcExample {  
  
    **public** **static** **void** main(**String** args[]) **throws** **SQLException** {  
        
        *//creating JDBC Connection to mysql database*  
        **String** url="jdbc:mysql://localhost:3306/test";  
        **Connection** conn = **DriverManager**.getConnection(url, "root", "root");  
       *// conn.setAutoCommit(false); keep auto commit false for better performance*  
  
        *//query for inserting batch data*  
        **String** query = "insert into employee values (?,?,NULL)";  
        **PreparedStatement** pStatement = conn.prepareStatement(query);  
        **int** batchSize = 100;  
        
        **long** startTime = **System**.currentTimeMillis();  
        **for** (**int** count = 0; count < 1000; count++) {  
            pStatement.setString(1, **Integer**.toString(count));  
            pStatement.setString(2, "Employee" + count);  
            pStatement.addBatch();  
            
            **if** (count % batchSize == 0) {  
                pStatement.executeBatch();  
            }  
        }  
  
       pStatement.executeBatch() ; *//for remaining batch queries if total record is odd no.*  
        
     *// conn.commit();*  
        pStatement.close();  
        conn.close();  
        **long** endTime = **System**.currentTimeMillis();  
        **long** elapsedTime = (endTime - startTime)/1000; *//in seconds*  
        **System**.out.println("Total time required to execute 1000 queries using PreparedStatement with JDBC batch insert is :" + elapsedTime);  
  
        
        
    }  
}  
  
**Output:**  
Total time required to execute 1000 queries using **PreparedStatement** with JDBC batch insert is :28

So JDBC batch insert and update does gives us better performance over queries running without batches. One of the important thing which I have not used here is, I have not disabled auto commit mode. You should always run [SQL query](http://javarevisited.blogspot.sg/2012/11/how-to-join-three-tables-in-sql-query-mysql-sqlserver.html) with auto-commit mode disabled even with *JDBC Batch insert and update example* and do commit() explicitly. That will further boost performance of your JDBC code. Try running above code with auto commit mode disabled and it won't take even a second to execute.

**Benefits of using JDBC batch update:**

Significant improvement in performance can be achieved by using JDBC batch update and insert. Since in case of batch queries, You effectively reduce database round-trip,  You save a lot of time spent on network latency, which results in better performance of Java application. Always combine JDBC batch insert or update with PreparedStatement to get best of both world and also follow these [Top 10JDBC best practices while writing JDBC code in Java](http://javarevisited.blogspot.sg/2012/08/top-10-jdbc-best-practices-for-java.html). e.g. running SQL query with auto-commit mode disabled.

That's all on how to run JDBC Batch insert and update in Java. We have seen how using JDBC batch insert can improve performance and  how we can execute PreparedStatement queries in batch. Choose batch size based on what suits your application and run query with auto commit mode disabled for better performance.

**Top 5 Concurrent Collections from JDK 5 and 6**Several new Collection classes are added in Java 5 and Java 6 specially concurrent alternatives of standard [synchronized ArrayList](http://javarevisited.blogspot.com/2011/05/example-of-arraylist-in-java-tutorial.html), [Hashtable](http://javarevisited.blogspot.sg/2012/01/java-hashtable-example-tutorial-code.html" \o "Click to open in a new window" \t "_blank) and  [synchronized HashMap](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html) collection classes. Many Java programmer still not familiar with these new collection classes from java.util.concurrent package and misses a whole new set of functionality which can be utilized to build more scalable and high performanceJava application. In this Java tutorial we will some of useful collection classes e.g. [ConcurrentHashMap](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html" \o "Click to open in a new window" \t "_blank), [BlockingQueue](http://javarevisited.blogspot.com/2012/12/blocking-queue-in-java-example-ArrayBlockingQueue-LinkedBlockingQueue.html) which provides some of the very useful functionalities to build concurrent Java application. By the way this is not a comprehensive article explaining each feature of all these concurrent collections, Instead I will just try to list out why they are there, which Collection class they replace or provides alternative for. Idea is to keep it short and simple while highlighting key points of those useful java.util.concurrent collections.

**1. ConcurrentHashMap**

ConcurrentHashMap is undoubtedly most popular collection class introduced in Java 5 and most of us are already using it. ConcurrentHashMap provides a concurrent alternative of [Hashtable or Synchronized Map](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) classes with aim to support higher level of concurrency by implementing fined grained locking. Multiple reader can access the Map concurrently  while a portion of Map gets locked for write operation depends upon concurrency level of Map. ConcurrentHashMap provides better scalability than there synchronized counter part. [Iterator](http://javarevisited.blogspot.com/2011/10/java-iterator-tutorial-example-list.html) of ConcurrentHashMap are [fail-safe iterators](http://javarevisited.blogspot.com/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) which doesn't throw ConcurrencModificationException thus eliminates another requirement of locking during iteration which result in further scalability and performance.

**2. CopyOnWriteArrayList and CopyOnWriteArraySet**

CopyOnWriteArrayList is a concurrent alternative of synchronized List. CopyOnWriteArrayList provides better concurrency than [synchronized](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html)List by allowing multiple concurrent reader and replacing the whole list on write operation. Yes, write operation is costly on CopyOnWriteArrayList but it performs better when there are multiple reader and requirement of iteration is more than writing. SinceCopyOnWriteArrayList Iterator also don't throw ConcurrencModificationException it eliminates need to lock the collection during iteration. Remember both ConcurrentHashMap and CopyOnWriteArrayList doesn't provides same level of locking as Synchronized Collection and achieves [thread-safety by](http://javarevisited.blogspot.com/2012/01/how-to-write-thread-safe-code-in-java.html) there locking and mutability strategy. So they perform better if requirements suits there nature. Similarly,CopyOnWriteArraySet is a concurrent replacement to Synchronized Set. See [What is CopyOnWriteArrayList in Java](http://java67.blogspot.com/2012/09/what-is-copyonwritearraylist-in-java-example-vs-arraylist.html) for more details

**3. BlockingQueue**

BlockingQueue is also one of better known collection class in Java 5. BlockingQueue makes it easy to implement [producer-consumer design pattern](http://javarevisited.blogspot.com/2012/02/producer-consumer-design-pattern-with.html) by providing inbuilt blocking support for put() and take() method. put() method will block if Queue is full while take() method will block if Queue is empty. Java 5 API provides two concrete implementation of BlockingQueue in form of [ArrayBlockingQueue and LinkedBlockingQueue](http://javarevisited.blogspot.com/2012/12/blocking-queue-in-java-example-ArrayBlockingQueue-LinkedBlockingQueue.html), both of them implement FIFO ordering of element. ArrayBlockingQueue is backed by Array and its bounded in nature while LinkedBlockingQueue is optionally bounded. Consider using BlockingQueue to solve producer Consumer problem in Java instead of writing your won [wait-notify code](http://java67.blogspot.com/2012/12/producer-consumer-problem-with-wait-and-notify-example.html). Java 5 also provides PriorityBlockingQueue, another implementation of BlockingQueuewhich is ordered on priority and useful if you want to process elements on order other than FIFO.

**4. Deque and BlockingDeque**

Deque interface is added in Java 6 and it extends Queue interface to support insertion and removal from both end of Queue referred as head and tail. Java6 also provides concurrent implementation of Deque like ArrayDeque and LinkedBlockingDeque. Deque Can be used efficiently to increase parallelism in program by allowing set of [worker thread](http://javarevisited.blogspot.sg/2013/01/threadlocal-memory-leak-in-java-web.html) to help each other by taking some of work load from other thread by utilizing Deque double end consumption property. So if all [Thread](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) has there own set of task Queue and they are consuming from head; helper thread can also share some work load via consumption from tail.

**5. ConcurrentSkipListMap and ConcurrentSkipListSet**

Just like [ConcurrentHashMap](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) provides a concurrent alternative of [synchronized HashMap](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html). ConcurrentSkipListMap and ConcurrentSkipListSet provide concurrent alternative for synchronized version of SortedMap and SortedSet. For example instead of using TreeMap or TreeSet wrapped inside synchronized Collection, You can consider using ConcurrentSkipListMap or ConcurrentSkipListSet fromjava.util.concurrent package. They also implement NavigableMap and NavigableSet to add additional navigation method we have seen in our post [How to use NavigableMap in Java](http://javarevisited.blogspot.sg/2013/01/what-is-navigablemap-in-java-6-example-submap-head-tail.html).

That’s all on this list of concurrent Collection classes from Java 5 and 6. They are added on java.util.concurrent package as concurrent alternative of there synchronized counterpart. It’s good idea to learn these Collection classes along with other popular classes from Java Collection Framework.

[**How to use ConcurrentHashMap in Java**](http://javarevisited.blogspot.com/2013/02/concurrenthashmap-in-java-example-tutorial-working.html)

ConcurrentHashMap in Java is introduced as an alternative of Hashtable in Java 1.5 as part of Java concurrency package. Prior to Java 1.5 if you need a Map implementation, which can be safely used in a concurrent and multi-threaded Java program, than, you only have [Hashtable](http://javarevisited.blogspot.com/2012/01/java-hashtable-example-tutorial-code.html) or [synchronized Map](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) because HashMap is not [thread-safe](http://javarevisited.blogspot.com/2012/01/how-to-write-thread-safe-code-in-java.html). With ConcurrentHashMap, now you have better choice; because, not only it can be safely used in concurrent multi-threaded environment but also provides better performance over Hashtable and synchronizedMap. ConcurrentHashMap performs better than earlier two because it only locks a portion of Map, instead of whole Map, which is the case with [Hashtable and synchronized Map](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html). CHM allows concurred read operations and same time, maintains integrity by synchronizing write operations. We have seen basics of ConcurrentHashMap on [Top 5 Java Concurrent Collections from JDK 5 and 6](http://javarevisited.blogspot.sg/2013/02/concurrent-collections-from-jdk-56-java-example-tutorial.html) and in this Java tutorial, we will learn :

       How ConcurrentHashMap works in Java or how it is implemented in Java.

       When to use ConcurrentHashMap in Java

       ConcurrentHashMap examples in Java

       And some important properties of CHM .

## How ConcurrentHashMap is implemented in Java

ConcurrentHashMap is introduced as an alternative of Hashtable and provided all functions supported by Hashtable with additional feature called "concurrency level", which allows ConcurrentHashMap to partition Map. ConcurrentHashMap allows multiple readers to read concurrently without any [blocking](http://javarevisited.blogspot.com/2012/02/what-is-blocking-methods-in-java-and.html). This is achieved by partitioning Map into different parts based on concurrency level and locking only a portion ofMap during updates. Default concurrency level is 16, and accordingly Map is divided into 16 part and each part is governed with different lock. This means, 16 thread can operate on Map simultaneously, until they are operating on different part of Map. This makes ConcurrentHashMap high performance despite keeping thread-safety intact.  Though, it comes with caveat. Since update operations like put(),remove(), putAll() or clear() is not [synchronized](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html), **concurrent retrieval may not reflect most recent change on Map**.

In case of putAll() or clear(), which operates on whole Map, concurrent read may reflect insertion and removal of only some entries. Another important point to remember is iteration over CHM, [Iterator](http://javarevisited.blogspot.com/2011/10/java-iterator-tutorial-example-list.html) returned by keySet of ConcurrentHashMap are weekly consistent and they only reflect state of ConcurrentHashMap and certain point and may not reflect any recent change. Iterator of ConcurrentHashMap'skeySet area also [fail-safe](http://javarevisited.blogspot.in/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) and doesn’t throw ConcurrentModificationExceptoin..

Default concurrency level is 16 and can be changed, by providing a number which make sense and work for you while creating ConcurrentHashMap. Since concurrency level is used for internal sizing and indicate number of concurrent update without contention, so, if you just have few writers or thread to update Map keeping it low is much better. ConcurrentHashMap also uses ReentrantLock to internally lock its segments.

## ConcurrentHashMap putifAbsent example in Java

ConcurrentHashMap examples are similar to [Hashtable examples](http://javarevisited.blogspot.com/2012/01/java-hashtable-example-tutorial-code.html), we have seen earlier,  but worth knowing is use of putIfAbsent() method. Many times we need to insert entry into Map, if its not present already, and we wrote following kind of code:

synchronized(map){

**if** (map**.**get(key) **==** *null*){

**return** map**.**put(key, value);

  } **else**{

**return** map**.**get(key);

  }

}

Though this code will work fine in [HashMap and Hashtable](http://java67.blogspot.sg/2012/08/5-difference-between-hashtable-hashmap-Java-collection.html" \o "Click to open in a new window" \t "_blank), This won't work in ConcurrentHashMap; because, during put operation whole map is not locked, and while one thread is putting value, other thread's get() call can still return null which result in one thread overriding value inserted by other thread. Ofcourse, you can wrap whole code in [synchronized block](http://java67.blogspot.com/2013/01/difference-between-synchronized-block-vs-method-java-example.html) and make it [thread-safe](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html) but that will only make your code single threaded. ConcurrentHashMap provides putIfAbsent(key, value) which does same thing but atomically and thus eliminates above race condition.

## When to use ConcurrentHashMap in Java

ConcurrentHashMap is best suited when you have multiple readers and few writers. If writers outnumber reader, or writer is equal to reader, than performance of ConcurrentHashMap effectively reduces to [synchronized map](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) or [Hashtable](http://javarevisited.blogspot.com/2012/01/java-hashtable-example-tutorial-code.html). Performance of CHM drops, because you got to lock all portion of Map, and effectively each reader will wait for another writer, operating on that portion of Map. ConcurrentHashMap is a good choice for caches, which can be initialized during application start up and later accessed my many request processing threads. As javadoc states, CHM is also a [good replacement of Hashtable](http://javarevisited.blogspot.sg/2013/02/concurrent-collections-from-jdk-56-java-example-tutorial.html) and should be used whenever possible, keeping in mind, that CHM provides slightly weeker form of synchronization than Hashtable.

### **Summary**

Now we know What is ConcurrentHashMap in Java and when to use ConcurrentHashMap, it’s time to know and revise some important points about CHM in Java.

1. ConcurrentHashMap allows concurrent read and thread-safe update operation.

2. During update operation, ConcurrentHashMap only lock a portion of Map instead of whole Map.

3. Concurrent update is achieved by internally dividing Map into small portion which is defined by concurrency level.

4. Choose concurrency level carefully as a significant higher number can be waste of time and space and lower number may introduce thread contention in case writers over number concurrency level.

5. All operations of ConcurrentHashMap are [thread-safe](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html).

6. Since ConcurrentHashMap implementation doesn't lock whole Map, there is chance of read overlapping with update operations like put() and remove(). In that case result returned by get() method will reflect most recently completed operation from there start.

7. Iterator returned by ConcurrentHashMap is weekly consistent, [fail safe](http://javarevisited.blogspot.com/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) and never throw ConcurrentModificationException. In Java.

8. ConcurrentHashMap doesn't allow null as key or value.

9. You can use ConcurrentHashMap in place of [Hashtable](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html) but with caution as CHM doesn't lock whole Map.

10. During putAll() and clear() operations, concurrent read may only reflect insertion or deletion of some entries.

That’s all on **What is ConcurrentHashMap in Java** and when to use it. We have also seen little bit about internal working of ConcurrentHashMap and how it achieves it’s thread-safety and better performance over Hashtable and synchronized Map. Use ConcurrentHashMap in Java program, when there will be more reader than writers and it’s a good choice for creating cache in Java as well.

**10 Exception handling Best Practices in Java Programming**Exception handling is an important part of writing robust Java application. It’s a non functional requirement for any application, to gracefully handle any erroneous condition like resource not available, invalid input, null input and so on. Java provides several exception handling features, in built in language itself in form of try, catch and [finally keyword](http://javarevisited.blogspot.com/2012/11/difference-between-final-finally-and-finalize-java.html).  Java  programming language also allows you to create new exceptions and throw them using  [throw and throws](http://javarevisited.blogspot.com/2012/02/difference-between-throw-and-throws-in.html) keyword. In reality, Exception handling is more than knowing syntax. Writing a robust code is an art more than science, and here we will discuss few Java best practices related to Exception handling. These [Java best practices](http://javarevisited.blogspot.com/2013/01/java-best-practices-method-overloading-constructor.html) are followed even in standard JDK libraries, and several open source code to better deal with Errors and Exceptions. This also comes as handy guide of writing robust code for Java programmers.

Exception Handling Java Best Practices

Here is my collection of 10 Java best practices to write Exception handling code in Java. There have been both applause and criticism of checked Exception in Java, which is a language feature to force dealing with Exceptions. In this article, we will look to minimize use of checked Exception and learn when to use checked vs unchecked exceptions in Java as well.

**1) Use Checked Exception for Recoverable error and Unchecked Exception for programming error.**

Choosing between checked and unchecked exception is always been confusing for Java programmers. Checked exceptions ensures that you provide exception handling code for error conditions, which is a way from language to enforcing you for writing robust code, but same time it also add lots of clutter into code and makes it unreadable. Also, it seems reasonable to catch exception and do something if you have alternatives or recovery strategies.  See [checked vs unchecked exceptions](http://javarevisited.blogspot.com/2011/12/checked-vs-unchecked-exception-in-java.html) for more information on choosing between checked and RuntimeException in Java.

**2) Close or release resource in finally block**

This is a well known best practice in Java and quite a standard, while dealing with networking and IO classes. Closing resources in finally block guarantees that precious and scarce resource released properly in case of normal and aborted execution, guaranteed by finally block. From Java 7, language has a more interesting [automatic resource management or ARM blocks](http://javarevisited.blogspot.com/2011/09/arm-automatic-resource-management-in.html), which can do this for you. Nevertheless, always remember to close resources in finally block, which is important to release limited resources like FileDescriptors, used in case of both socket and files.

**3) Including cause of Exception in stack-trace**

Many times Java library and open source code wraps one Exception into another, when one exception is thrown due to result of another exception. Its become extremely important to log or print cause of root exception. Java Exception class provides getCause() method to retrieve cause which can be used to provide more information about root cause of Exception. This Java best practice helps a lot while [debugging](http://javarevisited.blogspot.com/2011/07/java-debugging-tutorial-example-tips.html)or troubleshooting an issue. Always remember to pass original Exception, into constructor of new Exception, if you are wrapping one exception into another.

**4) Always provide meaning full message on Exception**

message of Exception is the most important place, where you can point out cause of problem because this is the first place every programmer looks upon. Always try to provide precise and factual information here. For example, compare these two Exception messages for IllegalArgumentException :

message 1: "Incorrect argument for method"

message 2: "Illegal value for ${argument}: ${value}

first one just says that argument is illegal or incorrect, but second one include both name of argument and its illegal value which is important to point out cause of error. Always follow this *Java best practice*, when writing code for handling exceptions and errors in Java.

**5) Avoid overusing Checked Exception**

Checked Exception has there advantage in terms of enforcement, but at same time it also litters the code and makes it unreadable by obscuring business logic. You can minimize this by not overusing checked Exception which result in much cleaner code. You can also use newer Java 7 features like [one catch block for multiple exceptions](http://javarevisited.blogspot.com/2011/07/jdk7-multi-cache-block-example-tutorial.html) and [automatic resource management](http://java67.blogspot.com/2012/09/what-is-new-in-java-7-top-5-jdk-7.html), to remove some duplication.

**6) Converting Checked Exception into RuntimeException**

This is one of the technique used to limit use of checked Exception in many of frameworks like Spring ,where most of checked Exception, which stem from JDBC is wrapped into DataAccessException, an unchecked Exception. This Java best practice provides benefits, in terms of  restricting specific exception into specific modules, like SQLException into [DAO layer](http://javarevisited.blogspot.com/2013/01/data-access-object-dao-design-pattern-java-tutorial-example.html) and throwing meaningful RuntimeException to client layer.

**7) Remember Exceptions are costly in terms of performance**

One thing which is worth remembering is that Exceptions are costly, and can slow down your code. Suppose you have method which is reading from ResultSet and often throws SQLException than move to next element, will be much slower than normal code which doesn't throw that Exception. So minimizing catching unnecessary Exception and moving on, without fixing there root cause. Don’t just throw and catch exceptions, if you can use boolean variable to indicate result of operation, which may result in cleaner and performance solution. Avoid unnecessary Exception handling by fixing root cause.

**8) Avoid empty catch blocks**

Nothing is more worse than empty catch block, because it not just hides the Errors and Exception, but also may leave your [object](http://javarevisited.blogspot.com/2012/12/what-is-object-in-java-or-oops-example.html) in unusable or corrupt state. Empty catch block only make sense, if you absolutely sure that Exception is not going to affect object state on any ways, but still its better to [log](http://javarevisited.blogspot.com/2011/05/top-10-tips-on-logging-in-java.html) any error comes during program execution. This is not a Java best practice, but a most common practice, while writing Exception handling code in Java.

**9) Use Standard Exceptions**

Our ninth Java best practice advise on using standard and inbuilt Java Exceptions. Using standard Exception instead of creating own Exception every now and then is much better in terms of maintenance and consistency. Reusing standard exception makes code more readable, because most of  Java developers are familiar with standard RuntimeException from JDK like, IllegalStateException,IllegalArgumentException or [NullPointerException](http://javarevisited.blogspot.com/2012/06/common-cause-of-javalangnullpointerexce.html), and they will immediately be able to know purpose of Exception, instead of looking out another place on code or docs to find out purpose of user defined Exceptions.

**10) Document Exception thrown by any method**

Java provides [throw and throws keyword](http://java67.blogspot.com/2012/10/difference-between-throw-vs-throws-in.html) to throw exception and in javadoc you have @throw to document possible Exception thrown by any method. This becomes increasingly important if you are writing API or public interface. With proper documentation of Exception thrown by any method you can potentially alert anyone who is using it.

That's all on Java best practices to follow while handing Exceptions in Java. Do let us know what practices you follow while  writing Exception handling code in Java.

**Can You Overload or Override Static methods in Java**Can static method be overridden in Java, or *can you override and overload static method in Java*, is a common Java interview question, mostly asked to 2 years experienced Java programmers. Answer is, No, [you can not override static method in Java](http://java67.blogspot.com/2012/08/can-we-override-static-method-in-java.html), though you can declare method with same signature in sub class. It won't be overridden in exact sense, instead that is called method hiding. But at same time, you can overload static methods in Java, there is nothing wrong declaring static methods with same name, but different arguments. Some time interviewer also ask, *Why you can not override static methods in Java*? Answer of this question lies on time of resolution. As I said in [difference between static and dynamic binding](http://www.blogger.com/javarevisited.blogspot.com/2012/03/what-is-static-and-dynamic-binding-in.html) , static method are bonded during compile time using Type of reference variable, and not Object. If you have using IDE like Netbeans and Eclipse, and If you try to access static methods using an object, you will see warnings. As per Java coding convention, static methods should be accessed by class name rather than object. In short Static method can be overloaded, but can not be overridden in Java. If you declare,  another static method with same signature in derived class than static method of super class will be hidden, and any call to that static method in sub class will go to static method declared in that class itself. This is known as method hiding in Java.

Can Static Method be overridden in Java - See yourself

Let's see an example of trying to override a static method. In this Java Program, we have two classes Parent and Child, both have name() method which is static. Now, As per [rules of method overriding](http://java67.blogspot.com/2012/08/what-is-method-overriding-in-java-example-tutorial.html), if a method is overridden than a call is resolved by type of object during runtime. Which means, in our test class StaticOverrideTest, p.name() in second example, should call Child class' name()method, because reference variable of type Parent is referring an object of Child, but instead it call name() method of Parent class itself. This happens, because static methods are resolved or bonded during compile time, and only information which is available, and used by compiler is type of reference variable. Since p was reference variable of Parent type, name() method from Parent class was called. Now, In order to prove that static method can be hidden, if we call Child.name() or c.name(), it will call name() method from Child class. This means static methods can not overridden in Java, they can only be hidden. This also answers, *Why static method can not be overridden in Java*, because they are resolved during compile time. By the way, this example doesn't show, whether you can overload static method or not, but you can. See this tutorial, for an example of [overloading static method in Java](http://java67.blogspot.sg/2012/08/can-we-overload-static-method-in-java.html).

*/\*\**

*\* Java Program to show that, you can not override static method in Java.*

*\* If you declare same method in subclass then, It's known as method hiding.*

*\**

*\* @author Javin Paul*

*\*/*

public class StaticOverrideTest {

public static void main(String args[]) {

Parent p = new Parent();

p.name(); *// should call static method from super class (Parent)*

*// because type of reference variable*

*// p is Parent*

p = new Child();

p.name(); *// as per overriding rules this should call to child's static*

*// overridden method*. *Since static method can not be overridden*

*// , it will call parent static method*

*// because Type of p is Parent.*

Child c = new Child();

c.name(); *// will call child static method because static method*

*// get called by type of Class*

}

}

class Parent{

*/\**

*\* original static method in super class which will be hidden*

*\* in subclass.*

*\*/*

public static void name(){

System.out.println("static method from Parent");

}

}

class Child extends *Parent*{

*/\**

*\* Static method with same signature as in super class,*

*\* Since static method can not be overridden, this is called*

*\* method hiding. Now, if you call Child.name(), this method*

*\* will be called, also any call to name() in this particular*

*\* class will go to this method, because super class method is hidden.*

*\*/*

public static void name(){

System.out.println("static method from Child");

}

}

Output

static method from Parent

static method from Parent

static method from Child

That's all on this Java interview question guys. Remember, Static methods can not be overridden in Java, but they can be [overloaded](http://javarevisited.blogspot.com.au/2013/01/java-best-practices-method-overloading-constructor.html) and hidden in Java. We have also touched based on What is method hiding in Java, and learned *Why Static method can not be overridden in Java*, since they are bonded during compile time by using type of Class, and not at runtime using Objects.

**Difference between Singleton Pattern vs Static Class in Java**Singleton pattern  vs  Static Class (a class, having all static methods) is another interesting questions, which I missed while blogging about [Interview questions on Singleton pattern in Java](http://javarevisited.blogspot.com/2011/03/10-interview-questions-on-singleton.html). Since both Singleton pattern and static class provides good accessibility, and they share some similarities e.g. both can be used without creating object and both provide only one instance, at very high level it looks that they both are intended for same task. Because of high level similarities, interviewer normally ask questions like, *Why you use Singleton instead of Static Methods,* or Can you replace Singleton with static class, and  what are differences between [Singleton pattern](http://javarevisited.blogspot.com/2012/07/why-enum-singleton-are-better-in-java.html) and [static in Java](http://javarevisited.blogspot.sg/2012/03/mixing-static-and-non-static.html). In order to answer these question, it’s important to remember fundamental difference between Singleton pattern and static class, former gives you an[Object](http://javarevisited.blogspot.com/2012/12/what-is-object-in-java-or-oops-example.html), while later just provide static methods. Since an object is always much more capable than a method, it can guide you when to use Singleton pattern vs static methods.

In this Java article we will learn, where to use Singleton pattern in Java, and when static class is better alternative. By the way, JDK has examples of both singleton and static, and that too very intelligently e.g. java.lang.Math is a [final class](http://javarevisited.blogspot.com/2011/12/final-variable-method-class-java.html) with full of [static methods](http://javarevisited.blogspot.com/2011/11/static-keyword-method-variable-java.html), on the other hand java.lang.Runtime is a Singleton class in Java. For those who are not familiar with Singleton design pattern or static class,  static class is a [Java class](http://javarevisited.blogspot.com/2011/10/class-in-java-programming-general.html), which only contains static methods, good examples of static class is java.lang.Math,which contains lots of utility methods for various maths function e.g. sqrt(). While [Singleton classes](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html) are those, which has only one instance during application life cycle like java.lang.Runtime.

When to use Static Class in place of Singleton in Java

Indeed there are some situations, where static classes makes sense than Singleton. Prime example of this is java.lang.Math which is not Singleton, instead a class with all static methods. Here are few situation where I think using static class over Singleton pattern make sense:

1) If your Singleton is not maintaining any state, and just providing global access to methods, than consider using static class, as static methods are much faster than Singleton, because of [static binding](http://javarevisited.blogspot.com/2012/03/what-is-static-and-dynamic-binding-in.html) during compile time. But remember its not advised to maintain state inside static class, especially in concurrent environment, where it could lead subtle [race conditions](http://javarevisited.blogspot.com/2012/02/what-is-race-condition-in.html) when modified parallel by multiple threads without adequate synchronization.

You can also choose to use static method, if you need to combine bunch of utility method together. Anything else, which requires singles access to some resource, should use Singleton design pattern.

Difference between Singleton vs Static in Java

This is answer of our second interview question about Singleton over static. As I said earlier, fundamental difference between them is, one represent object while other represent a method. Here are few more differences between static and singleton in Java.

1) Static class provides better performance than Singleton pattern, because static methods are bonded on compile time.

2) One more difference between Singleton and static is, ability to override. Since [static methods in Java cannot be overridden](http://java67.blogspot.com/2012/08/can-we-override-static-method-in-java.html), they leads to inflexibility. On the other hand, you can override methods defined in Singleton class by extending it.

3) Static classes are hard to mock and consequently hard to test than Singletons, which are pretty easy to mock and thus easy to test. It’s easier to write [JUnit test](http://javarevisited.blogspot.com/2013/03/how-to-write-unit-test-in-java-eclipse-netbeans-example-run.html) for Singleton than static classes, because you can pass mock object whenever Singleton is expected, e.g. into constructor or as method arguments.

4) If your requirements needs to maintain state than Singleton pattern is better choice than static class, because

maintaining  state in later case is nightmare and leads to subtle bugs.

5) Singleton classes can be [lazy loaded](http://javarevisited.blogspot.sg/2012/12/how-to-create-thread-safe-singleton-in-java-example.html) if its an heavy object, but static class doesn't have such advantages and always eagerly loaded.

6) Many [Dependency Injection framework](http://javarevisited.blogspot.com/2012/12/inversion-of-control-dependency-injection-design-pattern-spring-example-tutorial.html) manages Singleton quite well e.g. Spring, which makes using them very easy.

These are some differences between static class and singleton pattern, this will help to decide between two, which situation arises. In next section we will when to choose Singleton pattern over static class in Java.

Advantage of Singleton Pattern over Static Class in Java

Main advantage of Singleton over static is that former is more object oriented than later. With Singleton, you can use [Inheritance](http://javarevisited.blogspot.com/2012/10/what-is-inheritance-in-java-and-oops-programming.html) and [Polymorphism](http://javarevisited.blogspot.com.au/2011/08/what-is-polymorphism-in-java-example.html) to extend a base class, implement an interface and capable of providing different implementations. If we talk about java.lang.Runtime, which is a Singleton in Java, call to getRuntime() method return different implementations based on different JVM, but guarantees only one instance per JVM, had java.lang.Runtime an static class, it’s not possible to return different implementation for different JVM.

That’s all on difference between Singleton and static class in Java. When you need a class with full OO capability , chose Singleton, while if you just need to store bunch of static methods together, than use static class.

**ReentrantLock Example in Java, Difference between synchronized vs ReentrantLock**ReentrantLock in Java is added on java.util.concurrent package in Java 1.5 along with other concurrent utilities like [CountDownLatch](http://javarevisited.blogspot.com/2012/07/countdownlatch-example-in-java.html), Executors and [CyclicBarrier](http://javarevisited.blogspot.com/2012/07/cyclicbarrier-example-java-5-concurrency-tutorial.html). ReentrantLock is one of the most useful addition in Java concurrency package and several of concurrent collection classes from java.util.concurrent package is written using ReentrantLock, including ConcurrentHashMap, see [How ConcurrentHashMap works in Java](http://javarevisited.blogspot.com/2013/02/concurrenthashmap-in-java-example-tutorial-working.html) for more details. Two key feature of ReentrantLock, which provides more control on lock acquisition is trying to get a lock with ability to interrupt, and a timeout on waiting for lock, these are key for writing responsive and scalable systems in Java. In short, ReentrantLock extends functionality of [synchronized keyword](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html) in Java and open path for more controlled locking in Java.

In this Java concurrency tutorial we will learn :

* What is ReentrantLock in Java ?
* Difference between ReentrantLock and synchronized keyword in Java?
* Benefits of using Reentrant lock in Java?
* Drawbacks of using Reentrant lock in concurrent program?
* Code Example of ReentrantLock in Java?

What is ReentrantLock in Java

On class level,  ReentrantLock is a concrete implementation of Lock [interface](http://javarevisited.blogspot.sg/2012/04/10-points-on-interface-in-java-with.html) provided in Java concurrency package from Java 1.5 onwards.  As per Javadoc, ReentrantLock is mutual exclusive lock, similar to implicit locking provided by [synchronized keyword in Java](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html), with extended feature like fairness, which can be used to provide lock to longest waiting thread. Lock is acquired by lock() method and held by[Thread](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html)until a call to unlock() method. Fairness  parameter is provided while creating instance of ReentrantLock in constructor. ReentrantLock provides same visibility and ordering guarantee, provided by implicitly locking, which means, unlock() happens before another thread get lock().

**Difference between ReentrantLock and synchronized keyword in Java**

Though ReentrantLock provides same visibility and orderings guaranteed as implicit lock, acquired by synchronized keyword in Java, it provides more functionality and differ in certain aspect. As stated earlier,  main difference between synchronized and ReentrantLock is ability to trying for lock interruptibly, and with timeout. [Thread](http://javarevisited.blogspot.com/2012/01/difference-thread-vs-runnable-interface.html)doesn’t need to block infinitely, which was the case with synchronized. Let’s see few more differences between synchronized and Lock in Java.

1) Another significant difference between ReentrantLock and synchronized keyword is fairness. synchronized keyword doesn't support fairness. Any thread can acquire lock once released, no preference can be specified, on the other hand you can make ReentrantLock fair by specifying fairness property, while creating instance of ReentrantLock. Fairness property provides lock to longest waiting thread, in case of contention.

2) Second difference between synchronized and Reentrant lock is tryLock() method. ReentrantLock provides convenient tryLock() method, which acquires lock only if its available or not held by any other thread. This reduce [blocking](http://javarevisited.blogspot.com/2012/02/what-is-blocking-methods-in-java-and.html)of thread waiting for lock in Java application.

3) One more worth noting difference between ReentrantLock and synchronized keyword in Java is, ability to interrupt Thread while waiting for Lock. In case of [synchronized](http://javarevisited.blogspot.com/2012/03/mixing-static-and-non-static.html) keyword, a thread can be blocked waiting for lock, for an indefinite period of time and there was no way to control that. ReentrantLock provides a method called lockInterruptibly(), which can be used to interrupt thread when it is[waiting for lock](http://javarevisited.blogspot.com/2011/05/wait-notify-and-notifyall-in-java.html). Similarly tryLock() with timeout can be used to timeout if lock is not available in certain time period.

4) ReentrantLock also provides convenient method to get List of all threads waiting for lock.

So, you can see, lot of significant differences between synchronized keyword and ReentrantLock in Java. In short, Lock interface adds lot of power and flexibility and allows some control over lock acquisition process, which can be leveraged to write highly scalable systems in Java.

Benefits of ReentrantLock in Java

Most of the benefits derives from the *differences covered between synchronized vs ReentrantLock* in last section. Here is summary of benefits offered by ReentrantLock over synchronized in Java:

1) Ability to lock interruptibly.

2) Ability to timeout while waiting for lock.

3) Power to create fair lock.

4) API to get list of waiting thread for lock.

5) Flexibility to try for lock without blocking.

**Disadvantages of ReentrantLock in Java**

Major drawback of using ReentrantLock in Java is wrapping method body inside [try-finally block](http://javarevisited.blogspot.com/2012/11/difference-between-final-finally-and-finalize-java.html), which makes code unreadable and hides business logic. It’s really cluttered and I hate it most, though IDE like [Eclipse](http://javarevisited.blogspot.com/2013/02/must-override-superclass-method-java-eclipse.html)and Netbeans can add those try catch block for you. Another disadvantage is that, now programmer is responsible for acquiring and releasing lock, which is a power but also opens gate for new subtle bugs, when programmer forget to release the lock in finally block.

Lock and ReentrantLock Example in Java

Here is a complete code example of How to use Lock interface and ReentrantLock in Java. This program locks a method called getCount(), which provides unique count to each caller. Here we will see both synchronized and ReentrantLock version of same program. You can see code with synchronized is more readable but it’s not as flexible as locking mechanism provided by Lock interface.

import java.util.concurrent.locks.ReentrantLock;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

 \* Java program to show, how to use ReentrantLock in Java.

 \* Reentrant lock is an alternative way of locking

 \* apart from implicit locking provided by synchronized keyword in Java.

 \* @author  Javin Paul

 \*/

public class ReentrantLockHowto {

    private final ReentrantLock lock = new ReentrantLock();

    private int count = 0;

     //Locking using Lock and ReentrantLock

     public int getCount() {

        lock.lock();

        try {

            System.out.println(Thread.currentThread().getName() + " gets Count: " + count);

            return count++;

        } finally {

            lock.unlock();

        }

     }

     //Implicit locking using synchronized keyword

     public synchronized int getCountTwo() {

            return count++;

     }

    public static void main(String args[]) {

        final ThreadTest counter = new ThreadTest();

        Thread t1 = new Thread() {

            @Override

            public void run() {

                while (counter.getCount() &lt; 6) {

                    try {

                        Thread.sleep(100);

                    } catch (InterruptedException ex) {

                        ex.printStackTrace();                    }

                }

            }

        };

        Thread t2 = new Thread() {

            @Override

            public void run() {

                while (counter.getCount() &lt; 6) {

                    try {

                        Thread.sleep(100);

                    } catch (InterruptedException ex) {

                        ex.printStackTrace();

                    }

                }

            }

        };

        t1.start();

        t2.start();

    }

}

Output:

Thread-0 gets Count: 0

Thread-1 gets Count: 1

Thread-1 gets Count: 2

Thread-0 gets Count: 3

Thread-1 gets Count: 4

Thread-0 gets Count: 5

Thread-0 gets Count: 6

Thread-1 gets Count: 7

That’s all on What is ReentrantLock in Java, How to use with simple example, and difference between ReentrantLock and synchronized keyword in Java. We have also seen significant enhancement provided by Lock interface over synchronized e.g. trying for lock, timeout while waiting for lock and ability to interrupt thread while waiting for lock. Just be careful to release lock in finally block.

**How to create Immutable Class and Object in Java**Writing or creating immutable classes in Java is becoming popular day by day, because of concurrency and multithreading advantage provided by immutable objects. Immutable objects offers several benefits over conventional mutable object, especially while creating concurrent Java application. Immutable object not only guarantees safe publication of object’s state, but also can be shared among other threads without any external [synchronization](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html). In fact JDK itself contains several immutable classes like [String](http://javarevisited.blogspot.com/2012/03/how-to-compare-two-string-in-java.html), [Integer](http://javarevisited.blogspot.com/2011/08/convert-string-to-integer-to-string.html) and other wrapper classes. For those, who doesn’t know what is immutable class or object, Immutable objects are those, whose state can not be changed once created e.g. java.lang.String, once created can not be modified e.g. trim, uppercase, lowercase. All modification in String result in new object, see [why String is immutable in Java](http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html) for more details. In this Java programming tutorial, we will learn, how to write immutable class in Java or how to make a class immutable. By the way making a class immutable is not difficult on code level, but its the decision to make, which class mutable or immutable which makes difference. I also suggest reading, Java Concurrency in Practice to learn more about concurrencybenefit offered by Immutable object.

What is immutable class in Java

As said earlier, Immutable classes are those class, whose [object](http://javarevisited.blogspot.com/2012/12/what-is-object-in-java-or-oops-example.html) can not be modified once created, it means any modification on immutable object will result in another immutable object. best example to understand immutable and mutable objects are, [String and StringBuffer](http://javarevisited.blogspot.com/2011/07/string-vs-stringbuffer-vs-stringbuilder.html). Since String is immutable class, any change on existing string object will result in another string e.g. replacing a character into String,[creating substring from String](http://javarevisited.blogspot.in/2011/10/how-substring-in-java-works.html" \o "Click to open in a new window" \t "_blank), all result in a new objects. While in case of mutable object like StringBuffer, any modification is done on object itself and no new objects are created. Some times this immutability of String can also cause security hole, and that the reason [why password should be stored on char array instead of String](http://javarevisited.blogspot.com/2012/03/why-character-array-is-better-than.html).

How to write immutable class in Java

Despite of few disadvantages, Immutable object still offers several benefits in multi-threaded programming and it’s a great choice to achieve [thread safety](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html) in Java code. here are few rules, which helps to make a class immutable in Java :

1. State of immutable object can not be modified after construction, any modification should result in new immutable object.

2. All fields of Immutable class should be final.

3. Object must be properly constructed i.e. object reference must not leak during construction process.

4. Object should be final in order to restrict sub-class for altering immutability of parent class.

By the way, you can still create immutable object by violating few rules, like String has its [hashcode](http://javarevisited.blogspot.com/2011/10/override-hashcode-in-java-example.html) in non final field, but its always guaranteed to be same. No matter how many times you calculate it, because it’s calculated from final fields, which is guaranteed to be same. This required a deep knowledge of Java memory model, and can create subtle [race conditions](http://javarevisited.blogspot.com/2012/02/what-is-race-condition-in.html) if not addressed properly. In next section we will see simple example of writing immutable class in Java. By the way, if your Immutable class has lots of optional and mandatory fields, then you can also use [Builder design pattern](http://javarevisited.blogspot.com/2012/06/builder-design-pattern-in-java-example.html) to make a class Immutable in Java.

Immutable Class Example in Java

Here is complete code example of writing immutable class in Java. We have followed simplest approach and all rules for making a class immutable, including it [making class final](http://javarevisited.blogspot.com/2011/12/final-variable-method-class-java.html) to avoid putting immutability at risk due to [Inheritance](http://javarevisited.blogspot.com/2012/10/what-is-inheritance-in-java-and-oops-programming.html) and [Polymorphism](http://javarevisited.blogspot.com/2011/08/what-is-polymorphism-in-java-example.html).

public final class Contacts {

    private final String name;

    private final String mobile;

    public Contacts(String name, String mobile) {

        this.name = name;

        this.mobile = mobile;

    }

    public String getName(){

        return name;

    }

    public String getMobile(){

        return mobile;

    }

}

This Java class is immutable, because its state can not be changed once created. You can see that all of it’s fields are final. This is one of the most simple way of creating immutable class in Java, where all fields of class also remains immutable like String in above case. Some time you may need to write immutable class which includes mutable classes like [java.util.Date](http://javarevisited.blogspot.com/2012/04/difference-between-javautildate-and.html), **despite storing Date into final field it can be modified** **internally,** if internal date is returned to the client. In order to preserve immutability in such cases, its advised to **return copy of original object**, which is also one of the [Java best practice](http://javarevisited.blogspot.co.uk/2012/08/top-10-jdbc-best-practices-for-java.html). here is another example of making a class immutable in Java, which includes mutable member variable.

public final class ImmutableReminder{

    private final Date remindingDate;

    public ImmutableReminder (Date remindingDate) {

        if(remindingDate.getTime() < System.currentTimeMillis()){

            throw new IllegalArgumentException("Can not set reminder” +

                        “ for past time: " + remindingDate);

        }

        this.remindingDate = new Date(remindingDate.getTime());

    }

    public Date getRemindingDate() {

        return (Date) remindingDate.clone();

    }

}

In above example of creating immutable class, [Date](http://javarevisited.blogspot.com/2011/09/convert-date-to-string-simpledateformat.html) is a mutable object. If getRemindingDate() returns actual Date object than despite remindingDate being final variable, internals of Date can be modified by client code. By returning clone() or copy of remindingDate, we avoid that danger and preserves immutability of class.

Benefits of Immutable Classes in Java

As I said earlier Immutable classes offers several benefits, here are few to mention:

1) Immutable objects are by default [thread safe](http://javarevisited.blogspot.com/2012/01/how-to-write-thread-safe-code-in-java.html), can be shared without synchronization in concurrent environment.

2) Immutable object simplifies development, because its easier to share between multiple threads without external synchronization.

3) Immutable object boost performance of Java application by reducing [synchronization](http://java67.blogspot.com/2013/01/difference-between-synchronized-block-vs-method-java-example.html) in code.  
  
4) Another important benefit of Immutable objects is **reusability**, you can cache Immutable object and reuse them, much like String literals and Integers.  You can use [static factory methods](http://javarevisited.blogspot.it/2011/12/factory-design-pattern-java-example.html) to provide methods like valueOf(), which can return an existing Immutable object from cache, instead of creating a new one.

Apart from above advantages, immutable object has disadvantage of creating garbage as well. Since immutable object can not be reused and they are just a use and throw. String being a prime example, which can create lot of garbage and can potentially slow down application due to [heavy garbage collection](http://javarevisited.blogspot.com/2011/04/garbage-collection-in-java.html), but again that's extreme case and if used properly Immutable object adds lot of value.

That's all on **how to write immutable class in Java**. we have seen rules of writing immutable classes, benefits offered by immutable objects and how we can create immutable class in Java which involves mutable fields. Don’t forget to read more about concurrency benefit offered by Immutable object in one of the best Java book recommended to Java programmers, Concurrency Practice in Java.

**How to Compare Two Enum in Java - Equals vs == vs CompareTo**How do I compare two enum in Java? Should I use == operator or equals() method? What is difference between comparing enum with == and equals() method are some of the [tricky Java questions](http://java67.blogspot.com/2012/09/top-10-tricky-java-interview-questions-answers.html). Until you have solid knowledge of Enum in Java, It can be difficult to answer these question with confidence. By the way unlike comparing String in Java, you can use both == and equals() method to compare Enum, they will produce same result because equals() method of Java.lang.Enum internally uses == to *compare enum in Java*. Since every [Enum in Java](http://javarevisited.blogspot.com/2011/12/convert-enum-string-java-example.html) implicitly extends java.lang.Enum ,and since equals() method is declared final, there is no chance of overriding equals method in user defined enum. If you are not just checking whether two enum are equal or not, and rather interested in order of different instance of Enum, than you can use [compareTo() method](http://javarevisited.blogspot.com/2011/11/how-to-override-compareto-method-in.html) of enum to compare two enums. Java.lang.Enum implements Comparable interface and implements compareTo() method. Natural order of enum is defined by the order they are declared in Java code and same order is returned by ordinal() method.

Comparing Enum with equals and ==

As I said earlier, equals method of java.lang.Enum (see below code)  uses == operator to check if two enum are equal. This means, *You can compare Enum using both == and equals method*. Also [equals() method](http://javarevisited.blogspot.com/2011/02/how-to-write-equals-method-in-java.html) is declared final inside java.lang.Enum, so [risk of overriding equals method](http://javarevisited.blogspot.com/2011/12/method-overloading-vs-method-overriding.html). Here is the code of equals from java.lang.Enum class

public final boolean equals(Object other) {

return this==other;

}

By the way there are *subtle* *difference when you compare enum with == and equals method*, which stems from ==  (equality operator) being operator and equals() being method. Some of these points are already discussed in [difference between equals and == in Java](http://javarevisited.blogspot.sg/2012/12/difference-between-equals-method-and-equality-operator-java.html), but we will see them here with respect to comparing Enums in Java.

1) Using == for comparing Enum can prevent NullPointerException

This one is easy to guess, but same time provide worth of money. If you compare any Enum with null, using == operator, it will result in false, but if you use equals() method to do this check, you may get NullPointerException, unless you are using calling equals in right way as shown in [how to avoid NullPointerException in Java](http://javarevisited.blogspot.com/2012/06/common-cause-of-javalangnullpointerexce.html). Look at below code, here we are comparing an unknown Shape object withShape enum which contains CIRCLE, RECTANGLE etc.

private enum Shape{

RECTANGLE, SQUARE, CIRCLE, TRIANGLE;

}

private enum Status{

ON, OFF;

}

Shape unknown = null;

Shape circle = Shape.CIRCLE;

boolean result = unknown == circle; *//return false*

result = unknown.equals(circle); *//throws NullPointerException*

I agree this can be avoided by simply comparing known to unknown i.e. circle.equals(unknown), but this is one of the most common [coding error Java programmers make](http://javarevisited.blogspot.com/2012/02/java-mistake-1-using-float-and-double.html). By using == to compare enum, you can completely avoid it.

2) == method provides type safety during compile time

Another advantage of using == to compare enum is, compile time safety. Equality or == operator checks if both enum object are from same enum type or not at compile time itself, while equals() method will also return false but at runtime. Since it's always better to detect errors at compile time, == scores over equals in case of comparing enum. If you are using [Eclipse](http://javarevisited.blogspot.com/2012/10/eclipse-shortcut-to-remove-all-unused-imports-java.html) or Netbeans, you can detect these error as soon as you type. By the way Netbeans also shows warning when you call equals() method on two incomparable types, but that is completely IDE specific.

3) == should be faster than equals method

This is more from common sense, using operator should be a touch faster than calling method, and than using operator. Though I believe modern JIT compiler might inline equals() method, when you compare two enums in Java. Which means this would not be big difference in terms of performance.But, without any smartness from compiler or [JVM](http://javarevisited.blogspot.com/2011/12/jre-jvm-jdk-jit-in-java-programming.html), I think == should always be touch faster.

Comparing Enums with compareTo method

When we say comparing enum, it's not always checking if two enums are equal or not. Sometime you need to compare them for sorting or to arrange them in a particularly order. We know that we can compare objects using [Comparable and Comparator in Java](http://java67.blogspot.com/2012/10/how-to-sort-object-in-java-comparator-comparable-example.html) and enum is no different, though it provides additional convenience. Java.lang.Enum implements Comparable interface and it's compareTo() method compares only same type of enum. Also natural order of enum is the order in which they are declared in code. As shown on [10 examples of Enum in Java](http://javarevisited.blogspot.com/2011/08/enum-in-java-example-tutorial.html), same order is also maintained by ordinal() method of enum, which is used by EnumSet and EnumMap.

public final int compareTo(E o) {

Enum other = (Enum)o;

Enum self = this;

if (self.getClass() != other.getClass() && *// optimization*

self.getDeclaringClass() != other.getDeclaringClass())

throw new ClassCastException();

return self.ordinal - other.ordinal;

}

If you look last line, it's using ordinal to compare two enum in Java.

That's all on How to compare two enum in Java and difference between == and equals to compare two enums. Though using equals() to compare object is considered [Java best practice](http://javarevisited.blogspot.com/2013/03/0-exception-handling-best-practices-in-Java-Programming.html), comparing Enum using == is better than using equals. Don't forget ordinal() and compareTo() methods, which is also key to get natural order of Enum during comparison.

**Difference between LEFT and RIGHT OUTER Joins in SQL**There are two kinds of OUTER joins in SQL, LEFT OUTER join and RIGHT OUTER join. Main difference between RIGHT OUTER join and LEFT OUTER join, as there name suggest, is inclusion of non matched rows. Sine INNER join only include matching rows, where value of joining column is same, in final result set, but OUTER join extends that functionality and also include unmatched rows in final result. LEFT outer join includes unmatched rows from table written on left of join predicate. On the other hand RIGHT OUTER join, along with all matching rows, includes unmatched rows from right side of table. In short result of LEFT outer join is INNER JOIN + unmatched rows from LEFT table and RIGHT OUTER join is INNER JOIN + unmatched rows from right hand side table. Similar to difference between INNER join and OUTER join, difference between LEFT and RIGHT OUTER JOIN can be better understand by a simple example, which we will see in next section. By the way joins are very popular in SQL interviews, and along with classic questions like [finding second highest salary of employee](http://javarevisited.blogspot.com/2012/12/how-to-find-second-highest-or-maximum-salary-sql.html), Inner join vs outer join or left outer join vs right outer join is commonly asked.

LEFT and RIGHT OUTER Join Example in SQL

In order to understand difference between LEFT and RIGHT outer join, we will use once again use classical Employee and Department relationship. In this example, both of these table are connected using dept\_id, which means both have same set of data in that column, let's see data on these two table.

mysql> select \* from employee;

+--------+----------+---------+--------+

| emp\_id | emp\_name | dept\_id | salary |

+--------+----------+---------+--------+

|    103 | Jack     |       1 |   1400 |

|    104 | John     |       2 |   1450 |

|    108 | Alan     |       3 |   1150 |

|    107 | Ram      |    NULL |    600 |

+--------+----------+---------+--------+

4 rows in set (0.00 sec)

mysql> select \* from department;

+---------+-----------+

| dept\_id | dept\_name |

+---------+-----------+

|       1 | Sales     |

|       2 | Finance   |

|       3 | Accounts  |

|       4 | Marketing |

+---------+-----------+

4 rows in set (0.00 sec)

If you look closely, there is one row in employee table which contains NULL, for which there is no entry in department table. Similarly department table contains a department (row) Marketing ,for which there is no employee in employee table. When we do a LEFT or RIGHT outer join it includes unmatched rows from left or right table. In this case LEFT OUTER JOIN should include employee with NULL as department and RIGHT OUTER JOIN should include Marketing department. Here is example of LEFT and RIGHT OUTER Join in MySQL database :

mysql> select e.emp\_id, e.emp\_name, d.dept\_name from employee e LEFT JOIN department d on e.dept\_id=d.dept\_id;

+--------+----------+-----------+

| emp\_id | emp\_name | dept\_name |

+--------+----------+-----------+

|    103 | Jack     | Sales     |

|    104 | John     | Finance   |

|    108 | Alan     | Accounts  |

|    107 | Ram      | NULL      |

+--------+----------+-----------+

4 rows in set (0.01 sec)

As I said unmatched rows, i.e. row with dept\_id as NULL has included in final result and dept\_name for that row is NULL, as there is no corresponding row for NULL dept\_id in department table. But note that Marketing department is not included in this result. Now, let's see example of RIGHT OUTER JOIN in MySQL, this should include Marketing department but leave out employee with NULL dept\_id.

mysql> select e.emp\_id, e.emp\_name, d.dept\_name from employee e RIGHT JOIN department d on e.dept\_id=d.dept\_id;

+--------+----------+-----------+

| emp\_id | emp\_name | dept\_name |

+--------+----------+-----------+

|    103 | Jack     | Sales     |

|    104 | John     | Finance   |

|    108 | Alan     | Accounts  |

|   NULL | NULL     | Marketing |

+--------+----------+-----------+

4 rows in set (0.00 sec)

As I said, final result set has Marketing department and emp\_id, emp\_name is NULL in that row because there is no employee with dept\_id=4 in employee table.

Difference between LEFT and RIGHT OUTER JOIN in SQL

In short, following are some notable difference between LEFT and RIGHT outer join in SQL :

1) LEFT OUTER join includes unmatched rows from left table while RIGHT OUTER join includes unmatched rows from right side of table.

2) Result of LEFT OUTER join can be seen as INNER JOIN + unmatched rows of left able while result of RIGHT OUTER join is equal to INNER JOIN + unmatched rows from right side table.

3) In ANSI SQL, left outer join is written as LEFT JOIN while right outer join is written as RIGHT JOIN in select sql statements.

4) In Transact-SQL syntax left outer join is written as \*= and right outer join is written as =\*, Sybase database supports both syntax and you can write join queries in both ANSI and T-SQL syntax.

That's all on **difference between LEFT and RIGHT OUTER JOIN in SQL**. We have seen example of RIGHT and LEFT join in MySQL database but since we have used ANSI syntax of OUTER joins, it's for other databases e.g. Oracle, Sybase, SQL Server and PostgreSQL as well. JOIN is a one of the most important and common concept in SQL and you should be good enough to figure out which rows will be included as a result of JOIN statement before actually running that [SELECT query](http://javarevisited.blogspot.com/2011/10/selct-command-sql-query-example.html) against any table. Some time erroneous JOIN query can bring loads of data and potentially may hang your database so beware of it.

## 5 Reasons to Prefer Composition over Inheritance in Java

Just to revise, composition and Inheritance are ways to reuse code to get additional functionality. In Inheritance, a new class, which wants to reuse code, inherit an existing class, known as super class. This new class is then known as sub class. On composition, a class, which desire to use functionality of an existing class, doesn't inherit, instead it holds a reference of that class in a member variable, that’s why the name composition. Inheritance and composition relationships are also referred as IS-A and HAS-A relationships. Because of IS-A relationship, an instance of sub class can be passed to a method, which accepts instance of super class. This is a kind of Polymorphism, which is achieved using Inheritance. A super class reference variable can refer to an instance of sub class. By using composition, you don’t get this behavior, but still it offers a lot more to tilde the balance in its side.  
  
1) One reason of favoring Composition over Inheritance in Java is fact that [Java doesn't support multiple inheritance](http://javarevisited.blogspot.com/2011/07/why-multiple-inheritances-are-not.html). Since you can only extend one class in Java, but if you need multiple functionality like e.g. for reading and writing character data into file, you need Reader and Writer functionality and having them as private members makes your job easy. That’s called composition. If you are following programming for interface than implementation principle, and using type of base class as member variable, you can use different Reader and Writer implementation at different situation. You won’t get this flexibility by using Inheritance, in case of extending a class, you only get facilities which are available at compile time.  
  
2) Composition offers better test-ability of a class than Inheritance. If one class is composed of another class, you can easily create [Mock Object](http://javarevisited.blogspot.sg/2014/04/difference-between-stub-and-mock-object-java-junit.html) representing composed class for sake of testing. Inheritance doesn't provide this luxury. In order to test derived class, you must need its super class. Since unit testing is one of the most important thing to consider during software development, especially in test driven development, composition wins over inheritance.  
  
3) Many object oriented design patterns mentioned by Gang of Four in there timeless classic Design Patterns: Elements of Reusable Object-Oriented Software, favors Composition over Inheritance. Classical examples of this is [Strategy design pattern](http://java67.blogspot.sg/2014/12/strategy-pattern-in-java-with-sample.html), where composition and delegation is used to change Context’s behavior, without touching context code. Since Context uses composition to hold strategy, instead of getting it via inheritance, it’s easy to provide a new Strategy implementation at run-time. Another good example of using composition over inheritance is Decorator design pattern. In [Decorator pattern](http://javarevisited.blogspot.com/2011/11/decorator-design-pattern-java-example.html), we don't extend any class to add additional functionality, instead we keep an instance of the class we are decorating and delegates original task to that class after doing decoration. This is one of the biggest proof of choosing composition over inheritance, since these design patterns are well tried and tested in different scenarios and withstand test of time, keeping there head high.

4) Though both Composition and Inheritance allows you to reuse code, one of the disadvantage of Inheritance is that it breaks encapsulation. If sub class is depending on super class behavior for its operation, it suddenly becomes fragile. When behavior of super class changes, functionality in sub class may get broken, without any change on its part. One example of inheritance making code fragile is method add() and addAll() from [HashSet](http://javarevisited.blogspot.sg/2012/06/hashset-in-java-10-examples-programs.html" \o "Click to open in a new window" \t "_blank). Suppose, If addAll() of HashSet is implemented by calling add() method and you write a sub class of HashSet, which encrypt the content before inserting into HashSet. Since there are only one methods add(), which can insert object into HashSet you override these method and called your encrypt() method by overriding add(). This automatically covers addAll() as well, because addAll() is implemented using add(), it looks very enticing.If you look closely you will see that this implementation is fragile, because its relied on super class behavior. If base class wants to improve performance and implements addAll() without calling add() method, following example will break.

**public** **class** **EncryptedHashSet** **extends** HashSet{

.....

**public** **boolean** **add**(Object o) {

**return** **super**.add(encrypt(o));

}

}

If you have *used Composition in favor of Inheritance* you won't face this problem and your class would have been more robust, because you are not relying on super class behavior any more. Instead you are using super class method for addition part and you will benefit with any improvement in addAll() as shown in below example:

**public** **class** **EncryptedHashSet** **implements** Set{

**private** HashSet container;

**public** **boolean** **add**(Object o) {

**return** container.add(encrypt(o));

}

**public** **boolean** **addAll**(Collection c) {

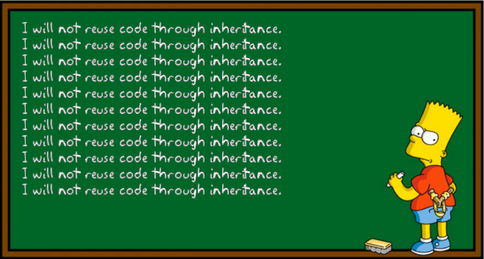
**return** conatainer.add(encrypt(c));

}

.......

}

In short, don't use Inheritance just for the sake of code reuse, Composition allows more flexible and extensible mechanism to reuse code. Make sure you don't forget this rule, if you don't to do this :)

[](http://3.bp.blogspot.com/-yQb2PmC63VY/VJmCl4aBI5I/AAAAAAAACPg/tZY6u4XSbtU/s1600/Why%2Bfavor%2Bcomposition%2Bover%2BInheritance%2Bin%2BJava.png)

5. Another reason of favoring Composition over inheritance is flexibility. If you use Composition you are flexible enough to replace implementation of Composed class with better and improved version. One example is using [Comparator class](http://javarevisited.blogspot.com/2011/06/comparator-and-comparable-in-java.html) which provides compare functionality. if your Container object contains a Comparator instead of extending a particular Comparator for comparing , its easier to change the way comparison is performed by setting different type of Comparator to composed instance, while in case of inheritance you can only have one comparison behavior on runtime, You can not change it runtime.  
  
There are many more reasons to *favor Composition over inheritance*, which you will start discovering once you start using design patterns in Java. In nutshell favoring Composition results in more flexible and robust code than using Inheritance. Though there are certainly some cases where using Inheritance makes much sense like when a genuine parent child relation exists, but most of time it makes sense to favor composition over inheritance for code reuse. There is an item of this topic on my another favorite book, [Effective Java](http://www.amazon.com/dp/0321356683/?tag=javamysqlanta-20), which has also helped me to understand this concept better, you may want to look that as well.

**What is difference in using instanceof and getClass() method for checking type inside equals?**

This question was asked multiple times, sometime by looking at your equals() and hashCode implementation. Well key difference comes from point that instanceof operator returns true, even if compared with sub class e.g. Subclass instanceof Super class is true, but with getClass() it's false. By using getClass() you ensure that your equals() implementation doesn't return true if compared with sub class object. While if you use instanceof operator, you end up breaking symmetry rule for equals which says that if a.equals(b) is true than b.equals(a) should also be true. Just replace a and b with instance of Super class and Sub class, and you will end up breaking symmetry rule for equals() method.

**Why use SLF4J over Log4J for logging in Java**  
Every Java programmers knows that logging is critical for any Java application, especially server side application, and many of them are already familiar with various logging libraries e.g. java.util.logging, Apache log4j, logback, but if you don't know about **SLF4J**, Simple logging facade for Java,  then it's time to learn and use SLF4J in your project. In this Java article, we will learn why using SLF4J is better than using log4j or java.util.logging. It’s been long time, since I wrote [10 logging tips for Java programmer](http://javarevisited.blogspot.com/2011/05/top-10-tips-on-logging-in-java.html),I don’t remember anything I have writing about logging. Anyway, let’s get back to topic, on contrary to all those logging libraries, there is a major difference between them and SLF4J. SLF4J or Simple logging Facade for Java is not really a logging implementation, instead it's an [abstraction layer](http://javarevisited.blogspot.com/2010/10/abstraction-in-java.html), which allows you to use any logging library in back-end. If you are writing API or utility library, which can be used internally or externally, then you really don't want that any client, which uses your library, should also stick with your choice of logging library. Suppose if a project is already using log4j, and you included a library say Apache Active MQ, which has dependency on logback, another logging library, then you need to include them as well, but if Apache Active MQ uses SL4J, you can continue with your logging library, without pain of adding and maintaining new logging framework. In short SLF4J make your code independent of any particular logging API, which is good think for public API developers. Though idea of abstracting logging library is not new and Apache commons logging is already using it, but now SLF4J is quickly becoming an standard for logging in Java world. Let's see couple of more reason to use SLF4J over log4j, logback or java.util.logging.

Prefer SLF4J over Log4J, logback and java.util.Logging

As I said earlier, main motivation of using SLF4J in your code to write log statements is, to make your program, independent of any particular logging library, which might require different configuration than you already have, and introduce more maintenance headache. But apart from that, there is one more feature of SLF4J API, which convinced me to use SL4J over my long time favorite **Log4j**, that is know as place holder and represented as {} in code. Placeholder is pretty much same as %s in [format() method of String](http://javarevisited.blogspot.com/2012/08/how-to-format-string-in-java-printf.html), because it get substituted  by actual string supplied at runtime. This not only reduce lot of String concatenation in your code, but also cost of creating String object. This is true even if you might not need that, depending upon your log level in production environment e.g. String concatenation on DEBUG and INFO levels. Since [Strings are immutable](http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html) and they are created in String pool, they consume [heap memory](http://javarevisited.blogspot.com/2013/04/what-is-maximum-heap-size-for-32-bit-64-JVM-Java-memory.html) and most of the time they are not needed e.g. an String used in DEBUG statement is not needed, when your application is running on ERROR level in production. By using SLF4J, you can defer String creation at runtime, which means only required Strings will be created. If you have been using log4j then you already familiar with a workaround of putting debug statement inside if() condition, but SLF4J placeholders are much better than that.

This is how you would do in Log4j, but surely this is not fun and reduce readability of code by adding unnecessary boiler-plate code.

**if** (logger.isDebugEnabled()) {

logger.debug("Processing trade with id: " + id + " symbol: " + symbol);

}

On the other hand if you use SLF4J, you can get same result in much concise format as shown below :

logger.debug("Processing trade with id: {} and symbol : {} ", id, symbol);

In SLF4J, we don't need String concatenation and don't incur cost of temporary not need String. Instead, we write log message in a template format with placeholder and supply actual value as parameters. You might be thinking about what if I have multiple parameters, well you can either use variable arguments version of log methods or pass them as Object array. This is really convenient and efficient way of logging. Remember, before generating final String for logging message, this method check if a particular log level is enabled or not, which not only reduce memory consumption but also CPU time involved for executing those [String concatenation](http://javarevisited.blogspot.com/2011/07/string-vs-stringbuffer-vs-stringbuilder.html) instruction in advance. Here is the code of SLF4J logger method from it's Log4j Adapter class Log4jLoggerAdapter from slf4j-log4j12-1.6.1.jar.

**public** **void** **debug**(String format, Object arg1, Object arg2) {

**if** (logger.isDebugEnabled()) {

FormattingTuple ft = MessageFormatter.format(format, arg1, arg2);

logger.log(FQCN, Level.DEBUG, ft.getMessage(), ft.getThrowable());

}

}

It's also worth knowing that logging has severe impact on [performance](http://javarevisited.blogspot.sg/2012/01/improve-performance-java-database.html) of application, and it's always advised to only mandatory logging in production environment.

How to use SLF4J with Log4J for logging

Apart from above benefits, I think there is one caveat though, in order to use SLF4J you not only need to include SLF4J API Jar e.g. slf4j-api-1.6.1.jar, but also companion JAR, depending upon which logging library, you are using in backend. Suppose If you want to *use SLF4J, Simple Logging Facade for Java,  along with Lo4J*, you need to include following jars in your [classpath](http://javarevisited.blogspot.sg/2013/02/windows-8-set-path-and-classpath-java-windows-7.html" \o "Click to open in a new window" \t "_blank), depending upon which version of SLF4J and log4J you are using e.g.

 slf4j-api-1.6.1.jar - JAR for SLF4J API

 log4j-1.2.16.jar    - JAR for Log4J API

 slf4j-log4j12-1.6.1.jar - Log4J Adapter for SLF4J

If you are using [Maven](http://javarevisited.blogspot.com/2011/08/increase-heap-size-maven-ant.html) to mange dependency in your project, you can just include SLF4J JAR, and maven will include it's dependent companion JAR. In order to use Log4J along with SLF4J, you can include following dependency in your project's pom.xml

**<dependency>**

**<groupId>**org.slf4j**</groupId>**

**<artifactId>**slf4j-log4j12**</artifactId>**

**<version>**1.6.1**</version>**

**</dependency>**

**<dependency>**

**<groupId>**org.slf4j**</groupId>**

**<artifactId>**slf4j-log4j12**</artifactId>**

**<version>**1.6.1**</version>**

**</dependency>**

By the way, if you are interested in using [variable argument version](http://javarevisited.blogspot.com/2011/09/variable-argument-in-java5-varargs.html) of logger methods, than include SLF4J 1.7 version.

Summary

To summarize this post, I would suggest following reasons are good enough to choose SLF4J over Log4j, commons logging, logback or java.util.logging directly.

1) Using SLF4J in your open source library or internal library, will make it independent of any particular logging implementation, which means no need to manage multiple logging configuration for multiple libraries, your client will going to appreciate this.

2) SLF4J provides place holder based logging, which improves readability of code by removing checks lie isDebugEnabled(), isInfoEnabled() etc.

3) By using SLF4J logging method, you defer cost of constructing logging messages (String), until you need it, which is both memory and CPU efficient.

4) As a side note, less number of temporary strings means less work for [Garbage Collector](http://javarevisited.blogspot.sg/2011/04/garbage-collection-in-java.html), which means better throughput and performance for your application.

These advantages are just tip of iceberg, you will learn about more benefits, when you start using SL4J and reading about it.  I strongly suggest, any new code development in Java, should use SLF4J for logging over any other logging API including log4J.

[**How clone method works in Java?**](http://javarevisited.blogspot.com/2013/09/how-clone-method-works-in-java.html)

The clone() is a tricky method from java.lang.Object class, which is used to create a copy of an Object in Java. The intention of the clone() method is simple, to provide a cloning mechanism, but somehow it's implementation became tricky and has been widely criticized from long time. Anyway, we will not go to classic debate of clone in Java, at least for now; instead, we will try to learn *how clone method works in Java*. To be fair, understating cloning mechanism in Java is not easy and even experienced Java programmer fail to explain how cloning of mutable object works, or a [difference between deep and shallow copy in Java](http://java67.blogspot.sg/2013/05/difference-between-deep-copy-vs-shallow-cloning-java.html). In this three part article, we will first see working of clone method in Java, and in second part we will learn **how to override clone method in Java**, and finally we will discuss *deep copy vs shallow copy* mechanism. The reason I chose to make this a three-part article is to keep the focus on one thing at a time. Since clone() itself is confusing enough, it's best to understand concept one by one. In this post, we will learn what is clone method, what it does and How clone method works in Java. By the way, clone() is one of the few fundamental methods defined by objects, others being equals, hashcode(), toString() along with wait and notify methods.

How clone method works in Java: <http://t.co/vJ0oIT3mWy> [#java](https://twitter.com/hashtag/java?src=hash) [#programming](https://twitter.com/hashtag/programming?src=hash)

— javinpaul (@javinpaul) [May 28, 2015](https://twitter.com/javinpaul/status/604008517373149184)

What is the clone of an object in Java?

An object which is returned by the clone() method is known as a clone of original instance. A clone object should follow basic characteristics e.g. a.clone() != a, which means original and clone are two separate object in Java heap, a.clone().getClass() == a.getClass() and clone.equals(a), which means clone is exact copy of original object. This characteristic is followed by a well behaved, correctly overridden clone() method in Java, but it's not enforced by the cloning mechanism. Which means, an object returned by clone() method may violate any of these rules.  
  
By following the convention of returning an object by calling super.clone(), when overriding clone() method, you can ensure that it follows first two characteristics. In order to follow the third characteristic, you must override equals method to enforce logical comparison, instead of physical comparison exists in java.lang.Object.  
  
For example, clone() method of Rectangle class in this method return object, which has these characteristics, but if you run the same program by commenting equals(), you will see that third invariant i.e. **clone.equals(a)** will return **false**. By the way there are a couple of good items on [Effective Java](http://www.amazon.com/dp/0321356683/?tag=javamysqlanta-20) regarding effective use of clone method, I highly recommend to read those items after going through this article.

How Clone method works in Java

[How Clone method works in Java](http://2.bp.blogspot.com/-wrzDeQGAe1I/TWu8pLuLr4I/AAAAAAAAADE/V017G-6Q61w/s1600/java_logo_50_50.jpg)java.lang.Object provides default implementation of clone() method in Java. It's declared as [protected](http://javarevisited.blogspot.sg/2012/10/difference-between-private-protected-public-package-access-java.html) and native in Object class, so implemented in native code. Since its convention to return clone() of an object by calling super.clone() method, any cloning process eventually reaches to java.lang.Object clone() method. This method, first checks if the corresponding object implements Cloneable interface, which is a marker interface. If that instance doesn't implement Cloneable then it throws CloneNotSupportedException in Java, a checked exception, which is always required to be handled while cloning an object. If an object passes this check, than java.lang.Object's clone() method creates a shallow copy of the object and returned it to the caller.  
  
  
Since Object class' clone() method creates copy by creating new instance, and then copying field-by-field, similar to assignment operator, it's fine for primitives and Immutable object, but not suited if your class contains some mutable data structure e.g. Collection classes like ArrayList or [arrays](http://javarevisited.blogspot.sg/2011/06/converting-array-to-arraylist-in-java.html). In that case, both original object and copy of the object will point to the same object in the heap. You can prevent this by using the technique known as deep cloning, on which each mutable field is cloned separately. In short, here is how clone method works in Java:

1) Any class calls clone() method on an instance, which implements Cloneable and overrides protected clone() method from Object class, to create a copy.

  Rectangle rec = **new** Rectangle(**30**, **60**);

  logger.info(rec);

**try** {

         logger.info("Creating Copy of this object using Clone method");

         Rectangle copy = rec.clone();

         logger.info("Copy " + copy);

    } **catch** (CloneNotSupportedException ex) {

         logger.debug("Cloning is not supported for this object");

    }

2) Call to clone() method on Rectangle is delegated to super.clone(), which can be a custom [superclass](http://java67.blogspot.sg/2013/06/difference-between-this-and-super-keyword-java.html) or by default java.lang.Object

    @Override

**protected** Rectangle **clone**() **throws** CloneNotSupportedException {

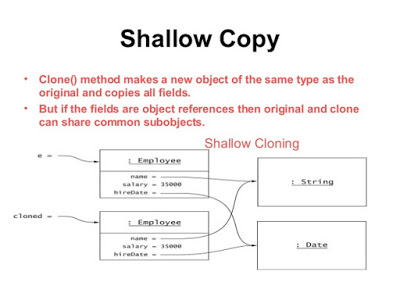
**return** (Rectangle) **super**.clone();

    }

3) Eventually, call reaches to java.lang.Object's clone() method, which verify if the corresponding instance implements Cloneable interface, if not then it throws CloneNotSupportedException, otherwise it creates a field-by-field copy of the instance of that class and returned to the caller.

So in order for clone() method to work properly, two things need to happen, a class should implement Cloneable interface and should*override clone() method of Object class*.  
  
By the way this was this was the simplest example of overriding clone method and how it works, things gets more complicated with real object, which contains mutable fields, arrays, collections, [Immutable object](http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html), and primitives, which we will see in [second part](http://javarevisited.blogspot.sg/2015/01/java-clone-tutorial-part-2-overriding-with-mutable-field-example.html) of this **Java Cloning tutorial** series.

Here is how  a shallow copy of an object looks like:

[](http://1.bp.blogspot.com/-rojtWvPpiiQ/VgvbR1CshxI/AAAAAAAAD2I/rgHImvatmls/s1600/Shallo%2BCopy%2Bin%2BJava%2Busing%2Bclone.jpg)

Java clone() method Example

In this article, we have not seen complexity of overriding clone method in Java, as our Rectangle class is very simple and only contains primitive fields, which means shallow cloning provided by Object's **clone()** method is enough. But, this example is important to understand the process of Object cloning in Java, and *how clone method works*.

### Things to Remember - Clone method in Java

1) The clone() method is used to create a copy of an object in Java. In order to use clone() method, class must implement java.lang.Cloneable [interface](http://javarevisited.blogspot.com/2012/04/10-points-on-interface-in-java-with.html) and override protected clone() method from java.lang.Object.  
  
A call to clone() method will result in CloneNotSupportedException if that class doesn't implement Cloneable interface.

2) No constructor is called during cloning of Object in Java.

3) Default implementation of clone() method in Java provides "*shallow copy"* of object, because it creates copy of Object by creating new instance and then copying content by assignment, which means if your class contains a mutable field, then both original object and clone will refer to same internal object. This can be dangerous because any change made on that mutable field will reflect in both original and copy object. In order to avoid this, override clone() method to provide the [deep copy of an object](http://javarevisited.blogspot.sg/2014/03/how-to-clone-collection-in-java-deep-copy-vs-shallow.html).

4) By convention, clone of an instance should be obtained by calling super.clone() method, this will help to preserve invariant of object created by clone() method i.e. **clone != original** and **clone.getClass() == original.getClass()**. Though these are not absolute requirement as mentioned in Javadoc.

5) A shallow copy of an instance is fine, until it only contains primitives and Immutable objects, otherwise, you need to modify one or more mutable fields of object returned by super.clone(), before returning it to caller.

That's all on **How clone method works in Java**. Now we know, what is the clone and what is Cloneable interface, a couple of things about clone method and what does default implementation of clone method do in Java. This information is enough to move ahead and read [second part](http://javarevisited.blogspot.sg/2015/01/java-clone-tutorial-part-2-overriding-with-mutable-field-example.html) of this Java cloning tutorial, on which we will learn, *how to override clone() method in Java*, for classes composed with primitives, mutable and immutable objects in Java.