Java Revisited

2010

**How do you detect deadlock in Java ?**

* Take a thread dump and analyse it.
* Use **jconsole**, it will show you exactly which threads are get locked and on which object.

[**Difference between Hashtable and ConcurrentHashMap**](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html)

* Hashtable is belongs to the Collection framework; ConcurrentHashMap belongs to the Executor framework.
* ConcurrentHashMap locking is applied only for updates.
* ConcurrentHashMap returns Iterator, which fails-safe
* ConcurrentHashMap uses a database shards logic (Segment<K, V>[] segments) is known as **Concurrency-Level**, i.e. divides the data into shards(segments) then puts locks on each shard (segment) instead of putting a single lock for whole data (Map). The default value is 16.
* ConcurrentHashMap uses multiple buckets to store data. This avoids read locks and greatly improves performance over a HashTable
* When you read from a ConcurrentHashMap using get(), there are no locks, contrary to the HashTable for which all operations are simply synchronized.

**Comparing Integer object with == in Java**

Integer i1 = **100**;

Integer i2 = **100**;

**if** (i1 == i2) {

System.out.println("i1 and i2 is equal"); 🡸 This will be printed

} **else** {

System.out.println("i1 and i2 is not equal ");

}  
Because int values from -127 to 127 are in a range which most JVM will like to cache so the VM actually uses the same object instance (and therefore memory address) for both i1 and i2. As a result, == returns a true result. 

[**Why String is Immutable or Final in Java**](http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html)

* String is Immutable because String objects are cached in String pool.
* Since cached String literal is shared between multiple client there is always a risk, where one client's action would affect all other client.
* Since caching of String objects was important from performance reason this risk was avoided by making String class Immutable.

*String was made final* so that no one can compromise invariant of String class e.g. Immutability, Caching, hascode calculationetc by extending and overriding behaviors. Another reason of *why String class is immutable* could be due to HashMap. Since Strings are very popular as HashMap key, it's important for them to be immutable so that they can retrieve the value object which was stored in HashMap.

**Why String is Final in Java**

* String has been widely used as parameter for many Java classes e.g. for opening network connection, you can pass hostname and port number as string , you can pass database URL as string for opening database connection. In case, if String is not immutable, this would lead serious security threat.
* Since String is immutable it can safely shared between many threads
* Another reason of Why String is immutable in Java is to allow String to cache its hashcode , being immutable String in Java caches its hashcode, and do not calculate every time we call hashcode method of String, which makes it very fast as hashmap key to be used in hashmap in Java.  This one is also suggested by  Jaroslav Sedlacek in comments below. In short because String is immutable, no one can change its contents once created which guarantees hashCode of String to be same on multiple invocation.
* Another good reason of Why String is immutable in Java suggested by Dan Bergh Johnsson on comments is: The absolutely most important reason that String is immutable is that it is used by the [class loading mechanism](http://javarevisited.blogspot.sg/2012/07/when-class-loading-initialization-java-example.html), and thus have profound and fundamentalsecurity aspects. Had String been mutable, a request to load "java.io.Writer" could have been changed to load "mil.vogoon.DiskErasingWriter"
* **Security and String pool being primary reason of making String immutable**.

[**Why wait notify and notifyAll called from synchronized block or method in Java**](http://javarevisited.blogspot.com/2011/05/wait-notify-and-notifyall-in-java.html)

* **wait() ,notify() and notifyAll() method of object class** must have to be called inside **synchronized** method or synchronized block in Java.
* we will receive IllegalMonitorStateException in java.
* We use **wait () and notify () or notifyAll () method mostly for inter-thread communication**. One thread is waiting after checking a condition e.g. In Producer Consumer example Producer Thread is waiting if buffer is full and Consumer thread notify Producer thread after he creates a space in buffer by consuming an element.
* calling notify() or notifyAll() issues a notification to a single or multiple thread that a condition has changed and once notification thread leaves synchronized block , all the threads which are waiting, fight for object lock on which they are waiting and lucky **thread returns from wait() method after reacquiring the lock** and proceed further

**Difference between java.lang.NoClassDefFoundError and ClassNotFoundException in Java**

* [ClassNotFoundException](http://javarevisited.blogspot.sg/2011/08/classnotfoundexception-in-java-example.html) which comes while trying to load a class at run-time only and name was provided during runtime not on compile time.
* **NoClassDefFoundError** will come if a class was present during compile time but not available in java classpath during runtime.

**Why Java doesn't support multiple inheritance**

* First reason is **ambiguity around Diamond problem**.
* **multiple inheritances does complicate the design and creates problem during casting, constructor chaining etc by creating ambiguities.**

**What is the advantage of new Lock interface over synchronized block in Java?**

The major advantage of lock interfaces on multi-threaded and concurrent programming is they provide two separate lock for reading and writing which enables you to write high performance data structure like [ConcurrentHashMap](http://javarevisited.posterous.com/difference-between-concurrenthashmap-and-coll) and [conditional blocking](http://javarevisited.blogspot.sg/2012/02/what-is-blocking-methods-in-java-and.html).

**Why we call start() method which in turns calls run() method, why not we directly call run() method ?**

You call start() method it creates new Thread and execute code declared in run() while directly calling run() method doesn’t create any new thread and execute code on same calling thread.

**Difference between start and run in Java Thread**

* When program calls start() method a **new Thread**is created and code inside run() method is executed in new Thread while if you call run() method directly **no new Thread is created** and code inside run() will execute on **current Thread**.
* Another**difference between start vs run in Java thread** is that you **can not call start()method twice**on thread object. once started, second call of start() will throw IllegalStateException in Java while you can call run() method twice.

**Benefits of Enums in Java**

* Enum is type-safe you can not assign anything else other than predefined Enum constants to an Enum variable.
* Best feature of Enum is you can use Enum in Java inside Switch statement
* Adding new constants on Enum in Java is easy and you can add new constants without breaking existing code.

[**Why Java does not support Operator overloading**](http://javarevisited.blogspot.com/2011/08/why-java-does-not-support-operator.html)  
1) Simplicity and Cleanliness

2) Avoid Programming Errors

3) JVM Complexity

4) Easy Development of Tools

**What is difference between poll() and remove() method of Queue interface?**.

* If Queue is empty() then a call to remove() method will throw Exception, while a call to poll() method returns null.

**Why ListIterator has add() method but Iterator doesn't**

* ListIterator has add() method because of its ability to traverse or iterate in both direction of collection. it maintains two pointers in terms of previous and next call and in position to add new element without affecting current iteration.

[**Why main method is public static in Java**](http://javarevisited.blogspot.com/2011/12/main-public-static-java-void-method-why.html)

* Main method in Java is entry point for any core Java program.
* Execution starts from main method when you type java main-class-name, JVM search for **public static void main(String args[])** method in that class and if it doesn't find that method it throws error **NoSuchMethodError:main** and terminates.

**Why main method is static in Java**

* Since main method is static Java virtual Machine can call it without creating any instance of class which contains main method.
* If main method were not declared static than JVM has to create instance of main Class and since constructor can be overloaded and can have arguments there would not be any certain and consistent way for **JVM to find main method in Java**.

2012

**Nonstatic variable can not be called from static method**

Since code in static context can be run even without creating any instance of class, it does not make sense asking value for an specific instance which is not yet created.

# [Why character array is better than String for Storing password in Java](http://javarevisited.blogspot.com/2012/03/why-character-array-is-better-than.html)

1. Since **Strings are immutable in Java** if you store password as plain text it will be available in memory until Garbage collector clears it and since String are used in String pool for reusability there is pretty high chance that it will be remain in memory for long duration, which pose a security threat.
2. Since any one who has access to memory dump can findthe password in clear text and that's another reason you should always used an encrypted password than plain text. Since Strings are immutable there is no way contents of Strings can be changed because [any change will produce new String](http://javarevisited.blogspot.com/2011/07/string-vs-stringbuffer-vs-stringbuilder.html), while if you char[] you can still set all his element as blank or zero. So **Storing password in character array clearly mitigates security risk of stealing password**.

**What is Encapsulation in Java**

**Encapsulation is nothing but protecting anything which is prone to change. rational behind encapsulation is that if any functionality which is well encapsulated in code i.e maintained in just one place and not scattered around code is easy to change**

[What is bounded and unbounded wildcards in Generics Java?](http://javarevisited.blogspot.com/2012/04/what-is-bounded-and-unbounded-wildcards.html)

* In short <? extends T> and <? super T> represent bounded wildcards
* while <?> represent an unbounded wildcard in generics .

[What is -XX:+UseCompressedOops in 64 bit JVM](http://javarevisited.blogspot.com/2012/06/what-is-xxusecompressedoops-in-64-bit.html)

* **XX:+UseCompressedOops** enables use of c*ompressed 32 bit OOPS in 64 bit JVM* which effectively compensate performance penalty imposed by 64 bit JVM without scarifying heap size advantage offered by them.
* Though its important to note that use of **Compressed Oops option limits your heap size up to 32Gigs** which is still more than handy but yes  a limitation if you looking for seriously gigantic heap.

# [Difference between EnumMap and HashMap in Java](http://javarevisited.blogspot.com/2012/09/difference-between-enummap-and-hashmap-in-java-vs.html)

**HashMap vs EnumMap in Java**

What is difference between EnumMap and HashMap in Java is the [latest Java collection interview question](http://javarevisited.blogspot.sg/2011/11/collection-interview-questions-answers.html) which has been asked to couple of my friends. This is one of the [tricky Java question](http://java67.blogspot.sg/2012/09/top-10-tricky-java-interview-questions-answers.html), specially if you are not very much familiar with [EnumMap in Java](http://javarevisited.blogspot.sg/2012/09/what-is-enummap-in-java-example-tutorial.html" \t "_blank" \o "Click to open in a new window), which is not uncommon, given you can use it with only [Enum keys](http://javarevisited.blogspot.sg/2011/08/enum-in-java-example-tutorial.html). Main *difference between EnumMap and HashMap* is that EnumMap is a specialized Map implementation exclusively for Enum as key. Using Enum as key, allows to do some implementation level optimization for high performance which is generally not possible with other object's as key. We have seen lot of [interview questions on HashMap](http://javarevisited.blogspot.in/2011/02/how-hashmap-works-in-java.html) in our article [How HashMap works in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html) but what we missed there is this question which is recently asked to some of my friend. Unlike HashMap, EnumMap is not applicable for every case but its best suited when you have Enum as key. We have already covered basics of EnumMap and some EnumMap example in my last article [What is EnumMap in Java](http://javarevisited.blogspot.sg/2012/09/what-is-enummap-in-java-example-tutorial.html) and In this post we will focus on keydifferences between HashMap and EnumMap in Java.

## Interview questions on Java Garbage collection

**Question 3 - What is difference between ParNew and DefNew Young Generation Garbage collector?**

By the way ParNew and DefNew is two young generation garbage collector. ParNew is a multi-threaded GC used along with concurrent Mark Sweep while DefNew is single threaded GC used along with Serial Garbage Collector.

**Question 6 – When does an Object becomes eligible for Garbage collection in Java ?**

Answer : An object becomes [eligible for garbage collection](http://javarevisited.blogspot.sg/2011/04/garbage-collection-in-java.html) when there is no live reference for that object or it can not be reached by any live thread.

**Question 7 - What is finalize method in Java ? When does Garbage collector calls finalize method in Java ?**

Answer : Finalize method in Java also called finalizer is a method defined in java.lang.Object and called by Garbage collector before collecting any object which is eligible for GC. Finalize() method provides last chance to object to do cleanup and free any remaining resource, to learn more about finalizers, read [What is finalize method in Java](http://javarevisited.blogspot.sg/2012/03/finalize-method-in-java-tutorial.html).

**Question 10 - Does Garbage collection occur in permanent generation space in JVM?**

By the way Garbage Collection does occur in PermGen space and if PermGen space is full or cross a threshold, it can trigger Full GC. If you look at output of GC you will find that PermGen space is also garbage collected. This is why correct sizing of PermGen space is important to avoid frequent full GC. You can control size of PermGen space by [JVM options](http://javarevisited.blogspot.sg/2011/11/hotspot-jvm-options-java-examples.html) -XX:PermGenSize and -XX:MaxPermGenSize.

## Difference between java.sql.Time, java.sql.Timestamp and java.sql.Date:

JDBC TIME or java.sql.Time represent only time information e.g. hours, minutes and seconds **without any date information**.

JDBC DATE or java.sql.Date represent only date information e.g. year, month and day **without any time information.**

JDBC TIMESTAMP or java.sql.Timestamp  **represent both date and time information** including nanosecond details.

2013 Java Revisited

**Difference between Stack vs Heap in Java**

* Stack memory is used to store [local variables](http://javarevisited.blogspot.com/2012/02/difference-between-instance-class-and.html) and function call,
* 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.
* 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.
* 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.
* 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.

**When to use Static Class in place of Singleton in Java**

* If your Singleton is not maintaining any state, and just providing global access to methods, than consider using static class

**Benefits of Immutable Classes in Java**

* 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.

**How to Compare Two Enum in Java - Equals vs == vs CompareTo**

* Both are technically correct. If you look at the source code for .equals(), it simply defers to ==.
* I use ==, however, as that will be null safe.

**2014 Java Revisted**

**Law of Demeter in Java - Principle of least Knowledge - Real life Example**Law of Demeter also known as*principle of least knowledge* is a coding principle, which says that a module should not know about the inner details of the objects it manipulates. If a code depends upon internal details of a particular object, there is good chance that it will break as soon as internal of that object changes. Since [Encapsulation](http://javarevisited.blogspot.com/2012/03/what-is-encapsulation-in-java-and-oops.html) is all about hiding internal details of object and exposing only operations, it also assert *Law of  Demeter*.

**How String in Switch works in Java 7  
Using hashCode of the string**.

## What is Absolute, Relative and Canonical Path

This is an example of the difference between an absolute path and a canonical path:

absolute path: C:\abc\..\abc\file.txt  
canonical path: C:\abc\file.txt

**2015 Revisited**  
**When to use intern() method of String in Java?**  
The method intern() creates an exact copy of a String object in the heap memory and stores it in the String constant pool.

Note that, if another String with the same contents exists in the String constant pool, then a new object won't be created and the new reference will point to the other String.

Java automatically interns String literals. This means that in many cases, the == operator appears to work for Strings in the same way that it does for ints or other primitive values.

Since interning is automatic for String literals, the intern() method is to be used on Strings constructed with new String()

String s1 = "Rakesh";

String s2 = "Rakesh";

String s3 = "Rakesh".intern();

String s4 = new String("Rakesh");

String s5 = new String("Rakesh").intern();

if ( s1 == s2 ){

System.out.println("s1 and s2 are same"); // 1.

}

if ( s1 == s3 ){

System.out.println("s1 and s3 are same" ); // 2.

}

if ( s1 == s4 ){

System.out.println("s1 and s4 are same" ); // 3.

}

if ( s1 == s5 ){

System.out.println("s1 and s5 are same" ); // 4.

}

will return:

s1 and s2 are same

s1 and s3 are same

s1 and s5 are same