2012

URL-rewriting vs URL-encoding in Servlet JSP

Main **difference between URL-rewriting and URL-encoding** is that URL-rewriting is a technique to maintain user session if cookies are not enabled on client browser or browser doesn't support cookie while **URL-encoding** is a way to pass string to server containing special characters  by converting special characters like space into some other characters like + . people often confuse between URL encoding and URL rewriting because of there names which sounds quite similar for new guys but functionality wise they are totally different to each other, Also servlet encodeURL() method adds more confusion because its sounds like its used for **URL Encoding** but indeed used for **URL Rewriting**. This is also a very popular servlet and JSP interview questions , I have also shared some more questions on my posts [10 Servlet Interview questions answers](http://javarevisited.blogspot.com/2011/09/servlet-interview-questions-answers.html) and [10 JSP interview questions and answers for Java programmer](http://javarevisited.blogspot.com/2011/10/jsp-interview-questions-answers-for.html).

Difference between URL Rewriting and URL Encoding in JSP Servlet

You often need to encode URL before sending it to server and need to rewrite URL with session id in order to maintain session where cookie is not present. here are some more *differences between URL-rewriting and URL encoding in Servlet* JSP

1) java.servlet.http.HttpServletResponse methods encodeURL(String url) and encodeRedirectURL(String URL) is used to **encode SesssionID on URL** to support URL-rewriting. don't confuse with name encodeURL() because it doesn't do URL encoding instead it **embeds sessionID in url** if necessary. logic to include sessionID is in method itself and it doesn't embed sessionID if browser supports cookies or session maintenance is not required. In order to implement a robust **session tracking** all URL from Servlet and JSP should have session id embedded on it.

In order to implement **URL-rewriting in JSP** you can use use JSTL core tag all URL passed to it will automatically be URL-rewriting if browser doesn't support cookies.

While java.net.URLEncoder.encode() and java.net.URLDecoder.decode()is used to perform **URL Encoding and decoding** which replace special character from String to another character. This method uses default encoding of system and also deprecated instead of this you can use java.net.URLEncoder.encode(String URL, String encoding) which allows you to specify encoding. as per W3C UTF-8 encoding should be used to **encode URL in web application**.

2) In **URL rewriting** session id is appended to URL and in case of URL-encoding special character replaced by another character.

**Using float and double for monetary or financial calculation**Java is considered very safe programming language compared to C and C++ as it doesn't have free() and malloc() to directly do memory allocation and deallocation, You don't need to worry of array overrun in Java as they are bounded and there is pointer arithmetic in Java. Still there are some sharp edges in [Java programming language](http://javarevisited.blogspot.com/2011/11/run-java-program-from-command-prompt.html) which you need to be aware of while writing enterprise application. Many of us make subtle mistake in Java which looks correct in first place but turn out to be buggy when looked carefully. In this series of java articles I will be sharing some of common Java mistake programmers make while programming application in Java. As I have said earlier every day we learn new things but we forget something equally important. This again highlight importance of [code review](http://javarevisited.blogspot.com/2011/09/code-review-checklist-best-practice.html) and following [best practices in Java](http://javarevisited.blogspot.com/2011/08/code-comments-java-best-practices.html). In this part we will discuss why double and float should not be used in monetary or financial calculation where exact result of calculation is expected.

**Using double and float for exact calculation**

This is one of common mistake Java programmer make until they are familiar with BigDecimal class. When we learn Java programming we have been told that use float and double to represent decimal numbers its not been told that result of floating point number is not exact, which makes them unsuitable for any financial calculation which requires exact result and not approximation. float anddouble are designed for engineering and scientific calculation and many times doesn’t produce exact result also result of floating point calculation may vary from [JVM](http://javarevisited.blogspot.com/2011/12/jre-jvm-jdk-jit-in-java-programming.html) to JVM. Look at below example of BigDecimal and double primitive which is used to represent money value, its quite clear that floating point calculation may not be exact and one should use BigDecimal for financial calculations.

public class BigDecimalExample {

     public static void main(String args[]) throws IOException {

      //floating point calculation

      double amount1 = 2.15;

      double amount2 = 1.10;

      System.out.println("difference between 2.15 and 1.0 using double is: " + (amount1 - amount2));

      //Use BigDecimal for financial calculation

      BigDecimal amount3 = new BigDecimal("2.15");

      BigDecimal amount4 = new BigDecimal("1.10") ;

      System.out.println("difference between 2.15 and 1.0 using BigDecimal is: " + (amount3.subtract(amount4)));

    }

}

Output:

difference between 2.15 and 1.0 using double is: 1.0499999999999998

difference between 2.15 and 1.0 using BigDecmial is: 1.05

From above example of floating point calculation is pretty clear that result of floating point calculation may not be exact at all time and it should not be used in places where exact result is expected.

Using Incorrect BigDecimal constructor

Another mistake Java Programmers make is using wrong constructor of BigDecmial. BigDecimal has [overloaded constructor](http://javarevisited.blogspot.com/2012/01/what-is-constructor-overloading-in-java.html) and if you use the one which accept double as argument you will get same result as you do while operating with double. So always use BigDecimal with String constructor. here is an example of using BigDecmial constructed with double values:

//Creating BigDecimal from double values

BigDecimal amount3 = new BigDecimal(2.15);

BigDecimal amount4 = new BigDecimal(1.10) ;

System.out.println("difference between 2.15 and 1.0 using BigDecmial is: " + (amount3.subtract(amount4)));

Output:

difference between 2.15 and 1.0 using double is: 1.0499999999999998

difference between 2.15 and 1.0 using BigDecmial is: 1.049999999999999822364316059974953532218933105468750

I agree there is not much difference between these two constructor but you got to remember this.

Using result of floating point calculation in loop condition

One more mistake from Java programmer can be using result of floating point calculation for determining conditions on loop. Though this may work some time it may result  in infinite loop another time. See below example where your Java program will get locked inside infinite while loop:

double amount1 = 2.15;

double amount2 = 1.10;

while((amount1 - amount2) != 1.05){

  System.out.println("We are stuck in infinite loop due to comparing with floating point numbers");

}

Output:

We are stuck in infinite loop due to comparing with floating point numbers

We are stuck in infinite loop due to comparing with floating point numbers

This code will result in infinite loop because result of subtraction of amount1 and amount 2 will not be 1.5 instead it would be "1.0499999999999998" which make boolean condition true.

That’s all on this part of learning from mistakes in Java, bottom line is :

* Don’t use float and double on monetary calculation.
* Use BigDecimal, long or int for monetary calculation.
* Use BigDecimal with String constructor and avoid double one.
* Don’t use floating point result for comparing loop conditions.

[**What is Race Condition in multithreading – 2 Examples in Java**](http://javarevisited.blogspot.com/2012/02/what-is-race-condition-in.html)

**Race condition in Java** is a type of concurrency bug or issue which is introduced in your program because  parallel execution of your program by multiple threads at same time, Since Java is a multi-threaded programming language hence risk of Race condition is higher in Java which demands clear understanding of what causes a race condition and how to avoid that. Anyway Race conditions are just one of hazards or risk presented by  use of multi-threading in Java just like [deadlock in Java](http://javarevisited.blogspot.com/2010/10/what-is-deadlock-in-java-how-to-fix-it.html). **Race conditions** occurs when two thread operate on same object without proper synchronization and there operation interleaves on each other. Classical **example of Race condition** is incrementing a counter since increment is not an atomic operation and can be further divided into three steps like read, update and write. if two [threads](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) tries to increment count at same time and if they read same value because of interleaving of read operation of one thread to update operation of another thread, one count will be lost when one thread overwrite increment done by other thread. atomic operations are not subject to race conditions because those operation cannot be interleaved. This is also [a popular multi-threading interview questions](http://javarevisited.blogspot.com/2011/07/java-multi-threading-interview.html) during core java interviews. In this article we will see **how to find race condition in Java** and  two sample code patterns which often causes race conditions in Java.

How to find Race Conditions in Java

**Finding Race conditions** in any language is most difficult job and Java is no different, though since readability of Java code is very good and synchronized constructs are well defined heaps to find race conditions by code review. finding race conditions by unit testing is not reliable due to random nature of race conditions. since race conditions surfaces only some time your unit test may passed without facing any race condition. only sure shot way to find race condition is reviewing code manually or using code review tools which can alert you on potential race conditions based on code pattern and use of synchronization in Java. I solely rely on [code review](http://javarevisited.blogspot.com/2011/09/code-review-checklist-best-practice.html) and yet to find a suitable *tool for exposing race condition in java*.

Code Example of Race Condition in Java

Based on my experience in Java synchronization and where we use synchronized keyword I found that two code patterns namely "**check and act**" and "**read modify write**" can suffer race condition if not synchronized properly. both cases rely on natural assumption that a single line of code will be atomic and execute in one shot which is wrong e.g. ++ is not atomic.

**"Check and Act" race condition pattern**

classical example of "check and act" race condition in Java is getInstance() method of Singleton Class, remember that was one questions which we have discussed on 10 Interview questions on Singleton pattern in Java as "[How to write thread-safe Singleton in Java](http://javarevisited.blogspot.com/2011/03/10-interview-questions-on-singleton.html)". getInstace() method first check for whether instance is null and than initialized the instance and return to caller. Whole purpose of Singleton is that getInstance should always return same instance of Singleton. if you call getInstance() method from two thread simultaneously its possible that while one thread is initializing singleton after null check, another thread sees value of \_instance reference variable as null (quite possible in java) especially if your object takes longer time to initialize and enters into critical section which eventually results in getInstance() returning two separate instance of Singleton. This may not happen always because a fraction of delay may result in value of \_instance updated in main memory. here is a code example

public Singleton getInstance(){

if(\_instance == null){   //**race condition if two threads sees \_instance= null**

\_instance = new Singleton();

}

}

an easy way to fix "**check and ac**t" race conditions is to synchronized keyword and enforce locking which will make this operation atomic and guarantees that block or method will only be executed by one thread and result of operation will be visible to all threads once [synchronized blocks](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html) completed or thread exited form synchronized block.

**read-modify-update race conditions**

This is another code pattern in Java which cause race condition, classical example is the non thread safe counter we discussed in [how to write thread safe class in Java](http://javarevisited.blogspot.com/2012/01/how-to-write-thread-safe-code-in-java.html). this is also a very popular multi-threading question where they ask you to find bugs on concurrent code. read-modify-update pattern also comes due to improper synchronization of **non-atomic operations** or combination of two individual atomic operations which is not atomic together e.g. put if absent scenario. consider below code

if(!hashtable.contains(key)){

hashtable.put(key,value);

}

here we only insert object into hashtable if its not already there. point is both contains() and put() are atomic but still this code can result in race condition since both operation together is not atomic. consider thread T1 checks for conditions and goes inside if block now CPU is switched from T1 to thread T2 which also checks condition and goes inside if block. now we have two thread inside if block which result in either T1 overwriting T2 value or vice-versa based on which thread has CPU for execution. In order to **fix this race condition in Java** you need to wrap this code inside synchronized block which makes them atomic together because no thread can go inside synchronized block if one thread is already there.

These are just some of *examples of race conditions in Java*, there will be numerous based on your business logic and code. best approach to find Race conditions is code review but its hard because thinking concurrently is not natural and we still assume code to run sequentially. Problem can become worse if JVM reorders code in absent of proper synchronization to gain performance benefit and this usually happens on production under heavily load, which is worst. I also suggest doing **load testing** in production like environment which many time helps to expose race conditions in java. Please share if you have faced any race condition in java projects.

[**Why non-static variable cannot be referenced from a static context?**](http://javarevisited.blogspot.com/2012/02/why-non-static-variable-cannot-be.html)

"**non-static variable cannot be referenced from a static context**" is biggest nemesis of some one who has just

started programming and that too in Java. Since [main method in java](http://javarevisited.blogspot.com/2011/12/main-public-static-java-void-method-why.html) is most popular method among all beginners and

they try to put program code there they face "*non-static variable cannot be referenced from a static context*" **compiler error**when they  try to access a non static member variable inside main in Java which is static. if you want to know

why main is declared static in Java see the link.

public class **StaticTest** {

    private int count=0;

    public static void main(String args[]) throws IOException {

        count++; //**compiler error: non-static variable count cannot be referenced from a static context**

    }

}

Why non static variable can not be called from static method

[non-static variable cannot be referenced from a static context](http://javarevisited.blogspot.com/2011/10/java-iterator-tutorial-example-list.html)Now before finding answer of compiler error "non-static variable cannot be referenced from a static context", let's have a quick revision of static. [Static variable in Java](http://javarevisited.blogspot.com/2011/11/static-keyword-method-variable-java.html) belongs to Class and its **value remains same for all instance**. static variable initialized when class is loaded into [JVM](http://javarevisited.blogspot.com/2011/12/jre-jvm-jdk-jit-in-java-programming.html) on the other hand instance variable has different value for each instances and they get created when instance of an object is created either by using new() operator or using reflection like Class.newInstance(). So if you try to access a non static variable without any instance compiler will complain because **those variables are not yet created** and they don't have any existence until an instance is created and they are associated with any instance. So in my opinion only reason which make sense to disallow [non static or instance variable](http://javarevisited.blogspot.com/2012/02/difference-between-instance-class-and.html) inside static context is non existence of instance.

In summary 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.

How to access non static variable inside static method or block

You can still access any non static variable inside any static method or block by creating an instance of [class in Java](http://javarevisited.blogspot.com/2011/10/class-in-java-programming-general.html)

and using that instance to reference instance variable. This is the only legitimate way to access non static variable

on static context. here is a code **example of accessing non static variable inside static context**:

public class **StaticTest** {

    private int count=0;

    public static void main(String args[]) throws IOException {

        StaticTest test = new StaticTest(); *//accessing static variable by creating an instance of class*

        test.count++;

    }

}

So next time if you get compiler error “**non-static variable cannot be referenced from a static context”**access static member by creating an instance of Class. Let me know if you find any other reason on why non-static variable cannot be referenced from a static context.

# [**fail-safe vs fail-fast Iterator in Java**](http://javarevisited.blogspot.com/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html)

**Difference between fail-safe and fail-fast Iterator** is becoming [favorite core java interview questions](http://javarevisited.blogspot.com/2011/04/top-20-core-java-interview-questions.html) day by day, reason

it touches concurrency a bit and interviewee can go deep on it to ask *how fail-safe or fail-fast behavior is implemented*.

In this article article we will see **what is fail-safe and fail fast iterators in java** and differences between fail-fast and fail-safe iterators . Concept of fail-safe iterator are relatively new in Java and first introduced with [Concurrent Collections in Java](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) 5 like ConcurrentHashMap and CopyOnWriteArrayList.

**Difference between fail-fast Iterator vs fail-safe Iterator in Java**

fail-fast Iterators in Java

As name suggest **fail-fast Iterators** fail as soon as they realized that *structure of Collection has been changed since iteration has begun*. Structural changes means adding, removing or updating any element from collection while one thread is Iterating over that collection. fail-fast behavior is implemented by keeping a modification count and if iteration thread realizes the change in modification count it throws ConcurrentModificationException.

Java doc says this is not a guaranteed behavior instead its done of "best effort basis", So application programming can not  rely on this behavior. Also since multiple threads are involved while updating and checking modification count and this check  is done without synchronization, there is a chance that Iteration thread still sees a stale value and might not be able to detect any change done by parallel threads. Iterators returned by most of JDK1.4 collection are fail-fast including Vector, [ArrayList](http://javarevisited.blogspot.com/2011/05/example-of-arraylist-in-java-tutorial.html), HashSet etc. to read more about Iterator see my post [What is Iterator in Java](http://javarevisited.blogspot.com/2011/10/java-iterator-tutorial-example-list.html).

fail-safe Iterator in java

Contrary to fail-fast Iterator, **fail-safe iterator** doesn't throw any Exception if Collection is modified structurally

while one thread is Iterating over it because they work on clone of Collection instead of original collection and that’s why they are called as fail-safe iterator. Iterator of CopyOnWriteArrayList is an example of fail-safe Iterator also iterator written by ConcurrentHashMap keySet is also fail-safe iterator and never throw ConcurrentModificationException in Java.

# That’s all on **difference between fail-safe vs fail-fast Iterator in Java**, As I said due to there confusing or not to easy differentiation they are quickly becoming [popular java collection questions](http://javarevisited.blogspot.com/2011/11/collection-interview-questions-answers.html) asked on various level of java interview. Let me know if you are aware of any other difference between fail-fast and fail-safe iterator. [**Why wait, notify and notifyAll is defined in Object Class and not on Thread class in Java**](http://javarevisited.blogspot.com/2012/02/why-wait-notify-and-notifyall-is.html)

**Why wait, notify and notifyAll is declared in Object Class instead of Thread** is famous core java interview question which is asked during all levels of Java interview ranging from 2 years, 4years to quite senior level position on java development. Beauty of this question is that it reflect what does interviewee knows about wait notify mechanism, how does it sees whole wait and notify feature and whether his understanding is not shallow on this topic. Like [Why Multiple inheritance is not supported in Java](http://javarevisited.blogspot.com/2011/07/why-multiple-inheritances-are-not.html) or [why String is final in java](http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html) there could be multiple answers of *why wait and notify is defined in Object class* and every one could justify there reason.

In my all interview experience I found that wait and notify still remains most confusing for most of Java programmer specially up-to 2 to 3 years and if they asked to write code using wait and notify they often struggle. So if you are going for any Java interview make sure you have sound knowledge of wait and notify mechanism as well as you are comfortable writing code using wait and notify like Produce Consumer problem or implementing Blocking queue etc. by the way This article is in continuation of  my earlier article related to wait and notify e.g. [Why Wait and notify requires to be called from Synchronized block or method](http://javarevisited.blogspot.com/2011/05/wait-notify-and-notifyall-in-java.html) and  [Difference between wait, sleep and yield method in Java](http://javarevisited.blogspot.com/2011/12/difference-between-wait-sleep-yield.html), if you haven’t read you may found interesting.

Reason Why Wait , Notify and NotifyAll are in Object Class.

Here are some thoughts on why they should not be in Thread class which make sense to me :

1) Wait and notify is not just normal methods or synchronization utility, more than that they are **communication mechanism between two threads in Java**. And Object class is correct place to make them available for every object if this mechanism is not available via any java keyword like synchronized. Remember synchronized and wait notify are two different area and don’t confuse that they are same or related. Synchronized is to provide mutual exclusion and ensuring [thread safety of Java class](http://javarevisited.blogspot.com/2012/01/how-to-write-thread-safe-code-in-java.html) like race condition while wait and notify are communication mechanism between two thread.

2 )**Locks are made available on per Object basis**, which is another reason wait and notify is declared in Object class rather then Thread class.

3) In Java in order to enter critical section of code, Threads needs lock and they wait for lock, they don't know which threads holds lock instead they just know the lock is hold by some thread and they should wait for lock instead of knowing which thread is inside the synchronized block and asking them to release lock. this analogy fits with wait and notify being on object class rather than thread in Java.

These are just my thoughts on **why wait and notify method is declared in Object class rather than Thread in Java** and you have different version than me. In reality its another design decision made by Java designer like [not supporting Operator overloading in Java](http://javarevisited.blogspot.com/2011/08/why-java-does-not-support-operator.html). Anyway please post if you have any other convincing reason *why wait and notify method should be in Object class and not on Thread*.

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

So what is difference between start and run method? Main difference is that 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**. Most of the time calling run() is bug or [programming mistake](http://javarevisited.blogspot.com/2012/02/java-mistake-1-using-float-and-double.html) because caller has intention of calling start() to create new thread and this error can be detect by many static code coverage tools like findbugs. If you want to perform time consuming task than always call start() method otherwise your [main thread](http://javarevisited.blogspot.com/2011/12/main-public-static-java-void-method-why.html) will stuck while performing time consuming task if you call run() method directly. 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.

## **Spring Security Example: Limit Number of User Session**

As I said it’s simple and easy when you use spring security framework or library. In fact is all declarative and no code is require to enable **concurrent session disable functionality**. You will need to include following xml snippet in your *Spring Security Configuration file* mostly named as applicaContext-security.xml. Here is sample **spring security Example** of limiting user session in Java web application:

**<session-management** invalid-session-url="/logout.html"**>**  
    **<concurrency-control** max-sessions="1" error-if-maximum-exceeded="true" **/>**  
**</session-management>**

As you see you can **specify how many concurrent session per user is allowed**, most secure system like online banking portals allow just one authenticate session per user. You can even specify a URL where user will be taken if they submit an invalid session identifier can be used to detect session timeout. Session-management element is used to capture session related stuff. Max-session specify how many concurrent authenticated session is allowed and if error-if-maximum-exceeded set to true it will flag error if user tries to login into another session.

[**What is Daemon thread in Java and Difference to Non daemon thread - Tutorial Example**](http://javarevisited.blogspot.com/2012/03/what-is-daemon-thread-in-java-and.html)

**Daemon thread in Java** are those thread which runs in background and mostly created by [JVM](http://javarevisited.blogspot.com/2011/12/jre-jvm-jdk-jit-in-java-programming.html) for performing background task like Garbage collection and other house keeping tasks. **Difference between Daemon and Non Daemon(User Threads)** is also an [interesting multi-threading interview question](http://javarevisited.blogspot.com/2011/07/java-multi-threading-interview.html), which asked mostly on fresher level java interviews. In one line main difference between daemon thread and user thread is that as soon as all user thread finish execution java program or JVM terminates itself, JVM doesn't wait for daemon thread to finish there execution. As soon as *last non daemon thread finished JVM terminates* no matter how many Daemon thread exists or running inside JVM. In this java thread tutorial we will see **example of Daemon thread in Java** and some more differences between Daemon and [non daemon threads](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html).

Important points about Daemon threads in Java

1. Any thread created by main thread, which runs [main method in Java](http://javarevisited.blogspot.com/2011/12/main-public-static-java-void-method-why.html) is by default non daemon because Thread inherits its daemon nature from the Thread which creates it i.e. parent Thread and since main thread is a non daemon thread, any other thread created from it will remain non-daemon until explicitly made daemon by callingsetDaemon(true).

2. Thread.setDaemon(true) makes a Thread daemon but it can only be called before starting Thread in Java. It will throw IllegalThreadStateException if corresponding Thread is already started and running.

3. Daemon Threads are suitable for doing background jobs like housekeeping, Though I have yet to use it for any practical purpose in application code. let us know if you have used daemon thread in your java application for any practical purpose.

Difference between Daemon and Non Daemon thread in Java

here are couple of *differences between daemon and user thread in Java*:

1) JVM doesn't [wait](http://javarevisited.blogspot.com/2011/05/wait-notify-and-notifyall-in-java.html) for any daemon thread to finish before existing.

2) Daemon Thread are treated differently than User Thread when JVM terminates, finally blocks are not called, Stacks are not unwounded and JVM just exits.

Daemon Thread Example in Java

Here is a **code example of daemon thread in java**. we make a user thread daemon by calling setDaemon(true) and every time you run you will see variable number of print statement related to "**daemon thread is running**" you will never see print statement written in finally block because finally will not be called.

**public** **class** DaemonThreadExample {  
  
    **public** **static** **void** main(**String** args[]){  
      
       **Thread** daemonThread = **new** [**Thread**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Thread.html)(**new** [**Runnable**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Runnable.html)(){  
            @**Override**  
           **public** **void** run(){  
               **try**{  
               while(**true**){  
                   **System**.out.println("Daemon thread is running");  
               }  
                    
               }**catch**(**Exception** e){  
                    
               }**finally**{  
                   **System**.out.println("Daemon Thread exiting"); *//never called*  
               }  
           }  
       }, "Daemon-Thread");  
        
       daemonThread.setDaemon(**true**); *//making this thread daemon*  
       daemonThread.start();        
}

Output:

Daemon thread is running

Daemon thread is running

Daemon thread is running

Daemon thread is running

Daemon thread is running

Daemon thread is running

Daemon thread is running

Daemon thread is running

That’s all on What is Daemon Thread in Java and **difference between Daemon and non daemon thread in Java** with code example of Daemon thread in Java. JVM also uses daemon thread for Garbage collection.

# [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)

**Why character array is better than String for storing password in Java** was recent question asked to one of my friend in a java interview. he was interviewing for a Technical lead position and has over 6 years of experience.Both [Character array and String](http://javarevisited.blogspot.com/2012/02/how-to-convert-char-to-string-in-java.html) can be used to store text data but choosing one over other is difficult question if you haven't faced the situation already. But as my friend said any question related to String must have a clue on special property of Strings like immutability and he used that to convince interviewer. here we will explore some reasons on why should you used char[] for storing password than String.  
  
This article is in continuation of my earlier interview question post on String e.g. [Why String is immutable in Java](http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html) or [How Substring can cause memory leak in Java](http://javarevisited.blogspot.com/2011/10/how-substring-in-java-works.html), if you haven't read those you may find them interesting.Here are few reasons which makes sense to believe that character array is better choice for storing password in Java than String:

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. 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**.

2) **Java itself recommends**using getPassword() method of JPasswordField which returns a char[] and deprecated getText() method which returns password in clear text stating security reason. Its good to follow advice from Java team and adhering to standard rather than going against it.

3) With String there is always a risk of printing plain text in [log file or console](http://javarevisited.blogspot.com/2011/05/top-10-tips-on-logging-in-java.html) but if use [Array](http://javarevisited.blogspot.com/2012/01/anonymous-array-example-java-create.html) you won't print contents of array instead its memory location get printed. though not a real reason but still make sense.

[**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) strPassword="Unknown";  
**char**[] charPassword= **new** **char**[]{'U','n','k','w','o','n'};  
**System**.out.println("String password: " + strPassword);  
**System**.out.println("Character password: " + charPassword);  
  
**String** password: Unknown  
**Character** password: [C@110b053

That's all on *why character array is better choice than String for storing passwords in Java*.  Though using char[] is not just enough you need to erase content to be more secure. I also suggest working with hash'd or [encrypted password](http://javarevisited.blogspot.com/2012/02/how-to-encode-decode-string-in-java.html) instead of plain text and clearing it from memory as soon as authentication is completed.

# [What is Encapsulation in Java and OOPS with Example](http://javarevisited.blogspot.com/2012/03/what-is-encapsulation-in-java-and-oops.html)

**Encapsulation in Java** or object oriented programming language is a concept which enforce protecting variables, functions from outside of class, in order to better manage that piece of code and having least impact or no impact on other parts of program duec to change in protected code. *Encapsulation in Java* is visible at different places and Java language itself provide many construct to encapsulate members. You can completely encapsulate a member be it a variable or method in Java by using private keyword and you can even achieve a lesser degree of encapsulation in Java by using other access modifier like protected or public. true value of encapsulation is realized in an environmentwhich is prone to change a lot and we know that in software requirements changes every day at that time if you have your code well encapsulated you can better manage risk with change in requirement. Along with [abstaction in java](http://javarevisited.blogspot.com/2010/10/abstraction-in-java.html) and [polymorphism in Java](http://javarevisited.blogspot.com/2011/08/what-is-polymorphism-in-java-example.html), Encapsulation is a must know concept. In this java tutorial  we will see **How to use encapsulation in Java**, advantage and disadvantage of Encapsulation and various design patterns and real life problems which makes use of Encapsulation object oriented concept. If you are looking for a quick guide on both OOPS and SOLID design principle in Java than you may find [**10 Object Oriented Design principles Java programmer should know**](http://javarevisited.blogspot.com/2012/03/10-object-oriented-design-principles.html)  interesting.

## **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. this can be better explained with a simple example of encapsulation in Java. we all know that constructor is used to create object in Java and constructor can accept argument. Suppose we have a class Loan has a constructor and than in various classes you have created instance of loan by using this [constructor](http://javarevisited.blogspot.com/2012/01/what-is-constructor-overloading-in-java.html). now requirements change and you need to include age of borrower as well while taking loan. Since this code is not well encapsulated i.e. not confined in one place you need to change everywhere you are calling this constructor i.e. for one change you need to modify several file instead of just one file which is more error prone and tedious, though it can be done with refactoring feature of advanced IDE wouldn't it be better if you only need to make change at one place ? Yes that is possible if we encapsulate Loan creation logic in one method say createLoan() and client code call this method and this method internally crate Loan object. in this case you only need to modify this method instead of all client code.

**Example of Encapsulation in Java**

**class** Loan{  
    **private** **int** duration;  //private variables examples of encapsulation  
    **private** [**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) loan;  
    **private** **String** borrower;  
    **private** **String** salary;

    //public constructor can break encapsulation instead use factory method  
    **private**Loan(**int** duration, **String** loan, **String** borrower, **String** salary){  
        **this**.duration = duration;  
        **this**.loan = loan;  
        **this**.borrower = borrower;  
        **this**.salary = salary;  
    }

    //no argument consustructor omitted here

   // create loan can encapsulate loan creation logic

**public** Loan createLoan(**String** loanType){  
    
     //processing based on loan type and than returning loan object

**return** loan;

    }    
}

In this same example of *Encapsulation in Java* you see all member variables are made private so they are well encapsulated you can only change or access this variable directly inside this class. if you want to allow outside world to access these variables is better creating a getter and setter e.g. getLoan() that allows you to do any kind of validation, security check before return loan so it gives you complete control of whatever you want to do and single channel of access for client which is controlled and managed.

## **Advantage of Encapsulation in Java and OOPS**

Here are few advantages of using **Encapsulation** while writing code in Java or any Object oriented programming language:

1. Encapsulated Code is more flexible and easy to change with new requirements.

2. Encapsulation in Java makes unit testing easy.

3. Encapsulation in Java allows you to control who can access what.

4. Encapsulation also helps to write immutable class in Java which are a good choice in multi-threading

environment.

5. Encapsulation reduce coupling of modules and increase cohesion inside a module because all piece of one thing

are encapsulated in one place.

6. Encapsulation allows you to change one part of code without affecting other part of code.

**What should you encapsulate in code**

Anything which can be change and more likely to change in near future is candidate of Encapsulation. This also helps to write more specific and cohesive code. Example of this is object creation code, code which can be improved in future like sorting and searching logic.

## **Design Pattern based on Encapsulation in Java**

Many design pattern in Java uses encapsulation concept, one of them is [Factory pattern](http://javarevisited.blogspot.com/2011/12/factory-design-pattern-java-example.html) which is used to create objects. Factory pattern is better choice than new operator for creating object of those classes whose creation logic can vary and also for creating different implementation of same interface. BorderFactory class of JDK is a good example of encapsulation in Java which creates different types ofBorder and encapsulate creation logic of Border.[Singleton pattern in Java](http://javarevisited.blogspot.com/2011/03/10-interview-questions-on-singleton.html) also encapsulate how you create instance by providing getInstance() method. since object

is created inside one class and not from any other place in code you can easily change how you create object without

affect other part of code.

### **Important points aboue encapsulation in Java.**

1. "Whatever changes encapsulate it" is a famous design principle.

2. Encapsulation helps in loose coupling and high cohesion of code.

3. Encapsulation in Java is achieved using access modifier private, protected and public.

4. Factory pattern , Singleton pattern in Java makes good use of Encapsulation.

## **java vs javaw command**

Both java.exe and javaw.exe can execute java programs including [jar file](http://javarevisited.blogspot.com/2012/03/how-to-create-and-execute-jar-file-in.html), only difference is that with *java.exe* you **have a console executed** and you need to wait until your java program finishes to run any other command on the other hand in case of javaw.exe no console or window is associated with execution.

you can [run your java program](http://javarevisited.blogspot.com/2011/11/run-java-program-from-command-prompt.html) either by using java.exe or javaw.exe. Its suggested to use ***javaw.exe*** when you don't need command line window to appear. javaw launcher will display in error in a dialog box if execution fails.

In summary **main difference between java and javaw command** is that with java.exe you will get a command prompt window but with javaw.exe there is no command prompt windows associated.

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

**Bounded and un-bounded wildcards in Generics** are two types of wildcard available on Java. Any Type can be bounded either upper or lower of class hierarchy in Generics by using **bounded wildcards**. In short <? extends T> and <? super T> represent bounded wildcards while <?> represent an unbounded wildcard in generics . In our last article [How Generics works in Java](http://javarevisited.blogspot.com/2011/09/generics-java-example-tutorial.html) , we have seen some basic details about *bounded and unbounded wildcards in generics* and In this Java tutorial we will see bounded and unbounded generics wildcards in details. We will start from basics like *what is bounded wild card in generics* and *what is unbounded wildcard*  and than will some [popular java interview questions](http://javarevisited.blogspot.com/2011/04/top-20-core-java-interview-questions.html) on generics like **Difference between ArrayList<? extends T>  and ArrayList<T super ?>.**

## **What is bounded and unbounded wildcards in Generics**

*bounded and unbounded wildcards in generics* are used to bound any Type. Type can be upper bounded by using <? extends T> where **all Types must be sub-class of T** or lower bounded using <? super T> where **all Types required to be super class of T**, here T represent lower bound. Single <?> is called an unbounded wildcard in generic and it can represent any type, similar to Object in Java. For example  **List<?>** can represent any List e.g. List<String> or List<Integer> its provides highest level of flexibility on passing method argument.

On the other hand **bounded wildcards provides limited flexibility within bound**. Any Type with bounded wildcards can only be instantiated within bound and any instantiation outside bound will result in compiler error.One of the important benefit of using bounded wildcard is that it not only **restrict number of Type** can be passed to any method as argument it also provides access to methods declared  by bound. for example TreeMap(Comparator<? super K> comparator) allows access to compare() method of [Comparator in Java](http://javarevisited.blogspot.com/2011/06/comparator-and-comparable-in-java.html).

## **Example of Bounded and Unbounded wildcards in Java Generics:**

Java Collection frameworks has several *examples of using bounded and unbounded wildcards in generics*. Utility method provided in Collections class accepts parametrized arguments. Collections.unmodifiableSet(Set<? extends T> s) and Collections.unmodifiableMap(Map<? extends K,? extends V> m) are written using  bounded wildcards which allows them to operate on either Collection of T or Collection of sub class or super class of T. just look at Java API for 1.5 and you will find lot of example of bounded and unbounded generic wildcards within JDK itself. If you are learning Java 1.5  you can also check [my post on Java Enum](http://javarevisited.blogspot.com/2011/08/enum-in-java-example-tutorial.html) and  [Variable arguments in Java](http://javarevisited.blogspot.com/2011/09/variable-argument-in-java5-varargs.html).

### When to use super and extends wildcards in Generics Java

Since there are two kinds of bounded wildcards in generics, **super and extends**, When should you use super wildcard and when should you extends wildcards. Joshua Bloch in Effective Java book has suggested **Producer extends, Consumer super** mnemonic regarding use of bounded wildcards. This book also has some good advice regarding how to use generics in Java and if you haven’t read it already, its worth reading book for Java programmer. Any way if type T is used as producer than use <? extends T>  and if type T represent consumer than use <? super T> bounded wildcards. Bounded wildcards in generics also increase flexibility of any API. To me its question of requirement, if a method also needs to accept any implementation of T then use extends wildcards.

### Difference between ArrayList<? extends T>  and ArrayList<? super T>

This is one of popular **generics interview question** , which is asked to check whether you are familiar to **bounded wildcards in generics**. both <? extends T> and <? super T> represent bounded wildcards, one will accept only T or sub class while other will accept T or super class. *bounded wildcards* gives more flexibility to methods which can operate on collection of T or its sub class. If you look at java.util.Collections class you will find several example of bounded wildcards in generics method. e.g. Collections.unmodifiableSet(Set<? extends T> s) will accept Set of type T or Set of sub class of T.

That's all on **what is bounded wildcards in generics**. both bounded and unbounded wild cards provides lot of flexibility on API design specially because Generics is not co-variant and List<String> can not be used in place of List<Object>. Bounded wildcards allows you to write methods which can operate on Collection of Type as well as Collection of Type subclasses.

# [Different types of JDBC drivers in Java - Quick overview](http://javarevisited.blogspot.com/2012/05/different-types-of-jdbc-drivers-in-java.html)

**How many types of JDBC drivers in Java** is a classical **JDBC interview question**, though I have not see this question recently but it was very popular during 2006 - 2008 period and still asked mostly on Junior programmer level interviews. There are mainly 4 types of JDBC drivers in Java, those are referred as type 1 to type 4 jdbc drivers. I agree its easy to remember them by type rather than with there actual name, Which I have yet to get in memory except plain old JDBC-ODBC bridge driver. By the way here are there full names :

Type 1 JDBC Driver is called JDBC-ODBC Bridge driver (bridge driver)

Type 2 JDBC Driver is referred as Native-API/partly Java driver (native driver)

Type 3 JDBC Driver is called AllJava/Net-protocol driver (middleware driver)

Type 4 JDBC Driver is called All Java/Native-protocol driver (Pure java driver)

This JDBC tutorial is in continuation of my earlier tutorials in JDBC like [How to connect to Oracle database using JDBC](http://javarevisited.blogspot.com/2012/04/java-program-to-connect-oracle-database.html)  
and  [4 tips to improve performance of JDBC applications](http://javarevisited.blogspot.com/2012/01/improve-performance-java-database.html). If you are new here and haven't read them already, Its worth looking.  Anyway out of all those 4 types, **JDBC-ODBC Bridge driver** is most common for connecting **SQL Server**, **MS Access** and mostly on training and development. here are quick review of all these four types of JDBC drivers. Also there has been some speculation of type 5 JDBC driver, I have to yet to see it.

## **JDBC ODBC Bridge Driver or Type 1 JDBC driver**

In case of JDBC ODBC bridge driver all JDBC calls doesn't directly goes to database instead they go via ODBC driver. JDBC-ODBC driver translates JDBC calls into ODBC callas and send them to ODBC driver for passing to database. Since type 1 driver act as bridge between JDBC and ODBC and that's why its called JDBC-ODBC bridge driver. This driver is not fast and good for production use mainly because of **several layer of translation** on back and fourth database traffic but it has  advantage in terms of of availability and can be your last choice.

## **Native-API/partly Java driver or Type 2 JDBC driver**

This is also called **type 2 driver** and its slightly better than type 1 JDBC driver. type 2 JDBC driver convert JDBC calls into database calls by using native API provided by database. This driver is database specific so once you switch from one database to another you need to change type 2 JDBC driver. performance is better than *JDBC-ODBC bridge driver* since communication layer is reduced. type 2 JDBC driver requires database native library to be available on client but it poses several version and compatibility issue. This was liked by Database vendors though because they can reuse there existing native libraries.

## **AllJava/Net-protocol driver or Type 3 JDBC driver**

both type 1 and type 2 JDBC drivers were not written in Java so there was need for pure Java JDBC driver to resolve portability issue. type 3 JDBC driver comes with pure java implementation (that's why All Java word ) but it uses 3 tier architecture where you have a Java client and Java Server which talk with Net protocol and Server speaking to database. type 3 JDBC driver never get popular among databasevendors as it was costly for them to rewrite there existing native database library which was mainly on C and C++.

## **All Java/Native-protocol driver or Type 4 JDBC driver**

type 4 JDBC driver is most popular among all *four types of JDBC driver*. it has not only implemented in Java but also incorporates all database call in single driver. It was pretty easy to use and deploy as well just include driver's [jar](http://javarevisited.blogspot.com/2012/03/how-to-create-and-execute-jar-file-in.html) in [classpath](http://javarevisited.blogspot.com/2011/01/how-classpath-work-in-java.html) and you are ready. It also removes 3 tier architecture of type 3 JDBC driver which makes it faster than type 3.  Major development happens on type 4 JDBC driver whendatabase upgrade themselves, though some of them still upgrade native database library or type 2 driver.

That's all on **quick overview of different types of JDBC drivers in Java**. JDBC drivers has evolved from JDBC ODBC bridge driver to type 4 JDBC driver, which is clean and portable. There has been some buzz around **JDBC driver 5 on Java** community which may include some advanced functionality. let us know if you come across some news on JDBC 5 driver.

# [Counting Semaphore Example in Java 5 – Concurrency Tutorial](http://javarevisited.blogspot.com/2012/05/counting-semaphore-example-in-java-5.html)

**Counting Semaphore in Java** is a synchronizer which allows to impose a bound on resource is added in Java 5 along with other popular concurrent utilities like CountDownLatch, CyclicBarrier and Exchanger etc. **Counting Semaphore** in Java maintains specified number of pass or permits, In order to access a shared resource, [Current Thread](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) must acquire a permit. If permit is already exhausted by other thread than it can wait until a permit is available due to release of permit from different thread. This concurrency utility can be very useful to implement [producer consumer design pattern](http://javarevisited.blogspot.com/2012/02/producer-consumer-design-pattern-with.html) or implement bounded pool or resources like **Thread Pool**, **DB Connection pool** etc. java.util.Semaphore class represent a Counting semaphore which is initialized with number of permits. Semaphore provides two main method acquire() and release() for getting permits and releasing permits. acquire() method blocks until permit is available. Semaphore provides both [blocking method](http://javarevisited.blogspot.com/2012/02/what-is-blocking-methods-in-java-and.html) as well as unblocking method to acquire permits. This Javaconcurrency tutorial focus on a very simple example of **Binary Semaphore** and demonstrate how mutual exclusion can be achieved using Semaphore in Java.

## **Counting Semaphore Example in Java (Binary Semaphore)**

a *Counting semaphore with one permit is known as****binary semaphore*** because it has only two state permit available or permit unavailable. Binary semaphore can be used to implement mutual exclusion or critical section where only one thread is allowed to execute. Thread will wait on acquire() until Thread inside critical section release permit by calling release() on semaphore.

here is a simple **example of counting semaphore in Java** where we are using binary semaphore to provide mutual exclusive access on critical section of code in java:

**import** java.util.concurrent.Semaphore;  
  
**public** **class** SemaphoreTest {  
  
    [**Semaphore**](http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/Semaphore.html) binary = **new** **Semaphore**(1);  
    
    **public** **static** **void** main(**String** args[]) {  
        **final** SemaphoreTest test = **new** SemaphoreTest();  
        **new** **Thread**(){  
            @**Override**  
            **public** **void** run(){  
              test.mutualExclusion();   
            }  
        }.start();  
        
        **new** **Thread**(){  
            @**Override**  
            **public** **void** run(){  
              test.mutualExclusion();   
            }  
        }.start();  
        
    }  
    
    **private** **void** mutualExclusion() {  
        **try** {  
            binary.acquire();  
  
            *//mutual exclusive region*  
            **System**.out.println(**Thread**.currentThread().getName() + " inside mutual exclusive region");  
            **Thread**.sleep(1000);  
  
        } **catch** (**InterruptedException** i.e.) {  
            ie.printStackTrace();  
        } **finally** {  
            binary.release();  
            **System**.out.println(**Thread**.currentThread().getName() + " outside of mutual exclusive region");  
        }  
    }   
    
}  
  
Output:  
Thread-0 inside mutual exclusive region  
Thread-0 outside of mutual exclusive region  
Thread-1 inside mutual exclusive region  
Thread-1 outside of mutual exclusive region

**Some Scenario where Semaphore can be used:**

1) To implement better Database connection pool which will block if no more connection is available instead of failing and handover Connection as soon as its available.

2) To put a bound on collection classes. by using semaphore you can implement bounded collection whose bound is specified by counting semaphore.

That's all on **Counting semaphore example in Java**. Semaphore is real nice concurrent utility which can greatly simply design and implementation of bounded resource pool. Java 5 has added several useful  concurrent utility and deserve a better attention than casual look.

### **Important points of Counting Semaphore in Java**

1. Semaphore class supports various overloaded version of tryAquire() method which acquires permit from semaphore only if its available during time of call.

2. Another worth noting method from Semaphore is acquireUninterruptibly() which is a blocking call and wait until a permit is available.

# [5 Difference between Application Server and Web Server in Java](http://javarevisited.blogspot.com/2012/05/5-difference-between-application-server.html)

**Application server and web server in Java** both are used to host Java web application. Though both application server and web server are generic terms, difference between application server and web server is a [famous J2EE interview question](http://javarevisited.blogspot.com/2011/09/servlet-interview-questions-answers.html). On  Java J2EE perspective **main difference between web server and application server is support of EJB**. In order to run EJB or host enterprise Java application (.ear) file you need an application server like JBoss, WebLogic, WebSphere or Glassfish, while you can still run your servlet and JSP or java web application (.war) file inside any web server like Tomcat or Jetty.

This Java interview question are in continuation of my previous post on interviews like [Top 10 Spring interview questions](http://javarevisited.blogspot.com/2011/09/spring-interview-questions-answers-j2ee.html) and  [Top 10 Struts interview question](http://javarevisited.blogspot.com/2011/11/struts-interview-questions-answer-j2ee.html).  Here we will see some difference between application server and web server in point format which will help you to answer this question in any javainterview.

## **Application Server vs Web Server**

1**. Application Server**supports **distributed transaction and EJB**. While Web Server only supports Servlets and JSP.

2. Application Server can contain web server in them. most of App server e.g. JBoss or WAS has Servlet and JSP container.

3. Though its not limited to Application Server but they used to provide services like **Connection pooling**, **Transaction management**, messaging, clustering, load balancing and persistence. Now Apache tomcat also provides connection pooling.

4. In terms of l*ogical difference between web server and application server*. web server is supposed to provide http protocol level service while application server provides support to web service and expose business level service e.g. EJB.

5. Application server are more heavy than web server in terms of resource utilization.

Personally I don't like to ask questions like ***Difference between Application Server and Web Server***. But since its been asked in many companies, you got to be familiar with some differences. Some times different interviewer expect different answer but I guess on Java's perspective until you are sure when do you need an application server and when you need a web server, you are good to go.

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

-XX:+UseCompressedOops JVM command line option is one of the most talked option of [64 bit JVM](http://javarevisited.blogspot.sg/2012/01/find-jvm-is-32-or-64-bit-java-program.html). Though 64 bit JVM allows you to specify larger Java heap sizes it comes with a performance penalty by using ***64 bit OOPS***. **Ordinary object pointers** also known as OOPS which is used to represent Java objects in Virtual Machine has an increased width of 64 bit than smaller 32 bit from earlier 32 bit JVM. because of increased size of OOPS, fewer OOPS can be stored in CPU cache registers which effectively reduced CPU cache efficiency. **-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. You should use -XX:+UseCompressedOops if [maximum heap size](http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html) specified by -Xmx is less than 32G. This is my 3rd article on JVM after [10 JVM option Java programmer should know](http://javarevisited.blogspot.com/2011/11/hotspot-jvm-options-java-examples.html) and  [how to find 32 bit JVM or 64 bit JVM](http://javarevisited.blogspot.sg/2012/01/find-jvm-is-32-or-64-bit-java-program.html" \o "Click to open in a new window" \t "_blank), I suggest reading those if you want to learn more about JVM.

## **Why should you use -XX:+UseCompressedOops JVM option**

-XX:+UseCompressedOops JVM option neutralize penalty imposed by 64 bit JVM. By using -XX:+UseCompressedOops you can avail benefit of both 64 bit JVM in terms of [larger Java heap size](http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html) and 32 bit JVM in terms of compressed size of OOPS which results in better performance by utilizing CPU cache better than larger, space inefficient64 bit OOPS pointers. Since better application performance is directly proportional to better CPU cache utilization, **-XX:+UseCompressedOops** allows you to get most of your available CPU registers along with additional CPU registers provided by some platforms like AMD x64. Some people may argue that further expansion of 32 bit compressed OOPS into 64 bit pointers may slow down things but that shouldn't be problem with modern high end processors.

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.

That's all on What is UseCompressedOops [JVM](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html)option and Why should you use -XX:+UseCompressedOops in 64 bit JVM. Importance of this option can also be realized by fact that from Java 6 update 18 Oracle by default enable -XX:+UseCompressedOops in HotSpot JVM based upon maximum Java heap size.

# [Video Example of depth first search (DFS) and breadth first search (BFS) graph algorithm](http://javarevisited.blogspot.com/2012/06/video-example-of-depth-first-search-dfs.html)

**Breadth first Search (BFS)** and **Depth first search (**DFS) algorithm are two most important graph and tree algorithm used for traversal. you can traverse all nodes of tree or graph by using BFS or DFS. Even though most of us learn about **Breadth first search**and depth first search in college its not easy to understand and it takes time to grasp the concept and I believe that once you understand how BFS or DFS works its easy to implement logic in Java or C++ but trying to implement or copy code without first understanding the algorithm is not going to work. I have read about BFS and DFS in text books, Wikipedia and several other places but never find an explanation as shown in this video. My friend got thisvideo on YouTube few years back while revising concept of BFS and DFS while preparing for [programming interview](http://javarevisited.blogspot.com/2011/06/top-programming-interview-questions.html?showComment=1322371935231) and data structure and shared with me that this is very simple, clear and concise. So I thought to share this BFS and DFS search video with you guys. This video explains How *depth first search algorithm* works with detailed example and stack data structure along with How *breadth first search algorithm* works with queue in simple words and live example of Breadth first search. if you are looking for code example than check out our last post [Breadth first search in java code example](http://javarevisited.blogspot.sg/2012/06/how-breadth-first-search-algorithm.html)

## **Depth First Search (DFS) traversal in Graph or Tree**

Here are the steps to follow for traversing graph in depth first search algorithm, remember we will use Stack data structure to perform Depth first search in Graph or Tree:

1) In Depth First Search we **pick a node** and call it a root for starting of traversal.

2) Put the node in Stack and mark it visited

3) Look for unvisited adjacent node, If there are more than one unvisited node than pick the one which comes

first in alphabetical or numerical order, put it on stack and mark it visited and treat it as new root.

4) repeat the procedure and look for unvisited adjacent node and if more than one choose which comes first alphabetically.

5) If there is no unvisited adjacent vertex than pop that element or node from stack, which takes you back to previous node.

6) now repeat the process of looking for unvisited adjacent vertex and visiting it.

7) DFS search will finish when you visited all the nodes and there is no element in Stack.

## **Breadth First Search (BFS) traversal in Graph**

Here are the steps to follow for traversing graph in breadth first search algorithm, [Queue](http://javarevisited.blogspot.com/2012/02/producer-consumer-design-pattern-with.html) datastrucutre will be used to perform BFS in Graph and tree:

1) In Breadth First Search we pick a node and call it a root for starting of traversal.

2) Put the node in Queue and mark it visited but we don't move to next node, we are still on current node

3) Look for another unvisited adjacent node, If there are more than one unvisited node than pick the one which comes

first in alphabetical or numerical order, put it on queue and mark it visited.

4) repeat the procedure until all adjacent node from current node is visited.

5) If there is no unvisited adjacent vertex than we are done with current node and time to move on to next node which is the first element from Queue.

6) now repeat the process and look for unvisited adjacent nodes and put them on Queue and mark visited but remain in current node.

7) BFS search will finish when you visited all the nodes and there is no more nodes remaining in Queue.

Watch the attached **video for complete example of Breadth first search algorithm**.

## **Difference between BFS and DFS algorithm**

One of the important *difference between Depth first search and Breadth first search*  which comes in my mind after watch above example is that, in DFS we use stack to put the visited nodes while in BFS we use Queue to store visited nodes. For those who are not familiar with Stack and Queue datastrucutre, Stack follow LIFO (Last In First Out) order and Queue follows FIFO (First in First Out) order, another difference between DFS and BFS is that on DFS we move to adjacent node as soon as we visit it, while in BFS we don't move and remain on current node even after visiting adjacent vertices. Once all the adjacent nodes visited we move current node or root to the first element in the Queue.

# [What is CountDownLatch in Java - Concurrency Example Tutorial](http://javarevisited.blogspot.com/2012/07/countdownlatch-example-in-java.html)

**What is CountDownLatch in Java**

CountDownLatch in Java is a kind of synchronizer which allows one Thread  to wait for one or more Threads before starts processing. This is very crucial requirement and often needed in server side core Java application and having this functionality built-in as CountDownLatch greatly simplifies the development. CountDownLatch in Java is introduced on Java 5 along with other concurrent utilities like [CyclicBarrier](http://javarevisited.blogspot.sg/2012/07/cyclicbarrier-example-java-5-concurrency-tutorial.html), [Semaphore](http://javarevisited.blogspot.sg/2012/05/counting-semaphore-example-in-java-5.html), [ConcurrentHashMap](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html) and [BlockingQueue](http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html) in java.util.concurrent package. In this Java concurrency tutorial we will  what is CountDownLatch in Java, How CountDownLatch works in Java, an example of **CountDownLatch in Java** and finally some worth noting points about this concurrent utility. You can also implement same functionality using  [wait and notify mechanism](http://javarevisited.blogspot.sg/2011/05/wait-notify-and-notifyall-in-java.html" \o "Click to open in a new window" \t "_blank) in Java but it requires lot of code and getting it write in first attempt is tricky,  With CountDownLatch it can  be done in just few lines. CountDownLatch also allows flexibility on number of thread for which [main thread](http://javarevisited.blogspot.sg/2011/12/main-public-static-java-void-method-why.html)should wait, It can wait for one thread or n number of thread, there is not much change on code.  Key point is that you need to figure out where to use CountDownLatch in Java application which is not difficult if you understand *What is CountDownLatch in Java*, What does CountDownLatch do and How CountDownLatch works in Java.

**How CountDownLatch works in Java**

Now we know What is CountDownLatch in Java, its time to find out How CountDownLatch works in Java. CountDownLatch works in latch principle,  main thread will wait until Gate is open. One [thread waits](http://javarevisited.blogspot.sg/2012/02/why-wait-notify-and-notifyall-is.html) for n number of threads specified while creating CountDownLatch in Java. Any thread, usually main thread of application,  which callsCountDownLatch.await() will wait until count reaches zero or its interrupted by another Thread. All other thread are required to do count down by calling CountDownLatch.countDown() once they are completed or ready to the job. as soon as count reaches zero, [Thread](http://javarevisited.blogspot.sg/2011/02/how-to-implement-thread-in-java.html) awaiting starts running. One of the disadvantage ofCountDownLatch is that **its not reusable once count reaches to zero** you can not use CountDownLatch any more, but don't worry Java concurrency API has another concurrent utility called [CyclicBarrier](http://javarevisited.blogspot.sg/2012/07/cyclicbarrier-example-java-5-concurrency-tutorial.html) for such requirements.

## **CountDownLatch Exmaple in Java**

In this section we will see a full featured real world example of using *CountDownLatch in Java*. In following **CountDownLatch example**, Java program requires 3 services namely CacheService, AlertService  and ValidationService  to be started and ready before application can handle any [request](http://javarevisited.blogspot.sg/2011/09/servlet-interview-questions-answers.html) and this is achieved by using CountDownLatch in Java.

**import** java.util.Date;  
**import** java.util.concurrent.CountDownLatch;  
**import** java.util.logging.Level;  
**import** java.util.logging.Logger;  
  
/\*\*  
 \* Java program to demonstrate How to use CountDownLatch in Java. CountDownLatch is

 \* useful if you want to start main processing thread once its dependency is completed

 \* as illustrated in this CountDownLatch Example  
 \*   
 \* @author Javin Paul  
 \*/  
**public** **class** CountDownLatchDemo {  
  
    **public** **static** **void** main(**String** args[]) {  
       **final** **CountDownLatch** latch = **new** **CountDownLatch**(3);  
       **Thread** cacheService = **new** **Thread**(**new** Service("CacheService", 1000, latch));  
       **Thread** alertService = **new** **Thread**(**new** Service("AlertService", 1000, latch));  
       **Thread** validationService = **new** **Thread**(**new** Service("ValidationService", 1000, latch));  
        
       cacheService.start(); //separate thread will initialize CacheService  
       alertService.start(); //another thread for AlertService initialization  
       validationService.start();  
        
       *// application should not start processing any thread until all service is up*

*// and ready to do there job.*  
       *// Countdown latch is idle choice here, main thread will start with count 3*

*// and wait until count reaches zero. each thread once up and read will do*

*// a count down. this will ensure that main thread is not started processing*

*// until all services is up.*  
        
       *//count is 3 since we have 3 Threads (Services)*  
        
       **try**{  
            latch.await();  //main thread is waiting on CountDownLatch to finish  
            **System**.out.println("All services are up, Application is starting now");  
       }**catch**(**InterruptedException** ie){  
           ie.printStackTrace();  
       }  
        
    }  
}  
  
/\*\*  
 \* Service class which will be executed by Thread using CountDownLatch synchronizer.  
 \*/  
**class** Service **implements** **Runnable**{  
    **private** **final** **String** name;  
    **private** **final** **int** timeToStart;  
    **private** **final** **CountDownLatch** latch;  
    
    **public** Service(**String** name, **int** timeToStart, **CountDownLatch** latch){  
        **this**.name = name;  
        **this**.timeToStart = timeToStart;  
        **this**.latch = latch;  
    }  
    
    @**Override**  
    **public** **void** run() {  
        **try** {  
            **Thread**.sleep(timeToStart);  
        } **catch** (**InterruptedException** ex) {  
            **Logger**.getLogger(Service.**class**.getName()).log(**Level**.SEVERE, **null**, ex);  
        }  
        **System**.out.println( name + " is Up");  
        latch.countDown(); //reduce count of CountDownLatch by 1  
    }  
    
}  
  
**Output:**  
ValidationService is Up  
AlertService is Up  
CacheService is Up  
All services are up, Application is starting now

By looking at output of this CountDownLatch example in Java, you can see that Application is not started until all services started by individual Threads are completed.

### When should we use CountDownLatch in Java :

Use CountDownLatch when one of Thread like [main thread](http://javarevisited.blogspot.sg/2011/12/main-public-static-java-void-method-why.html), require to wait for one or more thread to complete, before its start doing processing. Classical example of using CountDownLatch in Java  is any server side core Java application which uses services architecture,  where multiple services is provided by multiple threads and application can not start processing  until all services have started successfully as shown in our CountDownLatch example.

**CountDownLatch in Java – Things to remember**

Few points about Java CountDownLatch which is worth remembering:

1) You can not reuse CountDownLatch once count is reaches to zero, this is the main [difference between CountDownLatch and CyclicBarrier](http://javarevisited.blogspot.sg/2012/07/cyclicbarrier-example-java-5-concurrency-tutorial.html), which is frequently asked in [core Java interviews](http://javarevisited.blogspot.sg/2011/04/top-20-core-java-interview-questions.html) and [multi-threading  interviews](http://javarevisited.blogspot.sg/2011/07/java-multi-threading-interview.html).

2) Main Thread wait on Latch by calling CountDownLatch.await() method while other thread calls CountDownLatch.countDown() to inform that they have completed.

That’s all on **What is CountDownLatch in Java**, What does CountDownLatch do in Java, How CountDownLatch works in Java along with a real life CountDownLatch example in Java. This is a very useful concurrency utility and if you master *when to use CountDownLatch* and how to use CountDownLatch you will be able to reduce good amount of complex concurrency control code written using wait and notify in Java.

# [What is CyclicBarrier Example in Java 5 – Concurrency Tutorial](http://javarevisited.blogspot.com/2012/07/cyclicbarrier-example-java-5-concurrency-tutorial.html)

**What is CyclicBarrier in Java**

CyclicBarrier in Java is a synchronizer introduced in JDK 5 on java.util.Concurrent package along with other concurrent utility like [Counting Semaphore](http://javarevisited.blogspot.sg/2012/05/counting-semaphore-example-in-java-5.html), [BlockingQueue](http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html), [ConcurrentHashMap](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html) etc. CyclicBarrier is similar to CountDownLatch which we have seen in last article  [What is CountDownLatch in Java](http://javarevisited.blogspot.sg/2012/07/countdownlatch-example-in-java.html" \o "Click to open in a new window" \t "_blank) and allows multiple threads to wait for each other (barrier) before proceeding. Difference between CountDownLatch and CyclicBarrier is a also very [popular multi-threading interview question](http://javarevisited.blogspot.sg/2011/07/java-multi-threading-interview.html) in Java. CyclicBarrier is a natural requirement for concurrent program because it can be used to perform final part of task once individual tasks  are completed. All threads which [wait](http://javarevisited.blogspot.sg/2011/05/wait-notify-and-notifyall-in-java.html) for each other to reach barrier are called parties, CyclicBarrier is initialized with number of parties to be wait and threads wait for each other by calling CyclicBarrier.await() method which is a [blocking method in Java](http://javarevisited.blogspot.sg/2012/02/what-is-blocking-methods-in-java-and.html) and  blocks until all Thread or parties call await(). In general calling await() is shout out that Thread is waiting on barrier. await() is a blocking call but can be timed out or Interrupted by other thread. In this Java concurrency tutorial we will see *What is CyclicBarrier in Java*  and  an example of CyclicBarrier on which three Threads will wait for each other before proceeding further.

**Difference between CountDownLatch and CyclicBarrier in Java**

In our [last article](http://javarevisited.blogspot.sg/2012/07/countdownlatch-example-in-java.html) we have see how CountDownLatch can be used to implement multiple [threads](http://javarevisited.blogspot.sg/2011/02/how-to-implement-thread-in-java.html) waiting for each other. If you look at CyclicBarrier it also the does the same thing but there is a different you **can not reuse CountDownLatch** once count reaches zero while you can reuse CyclicBarrier by calling reset() method which resets Barrier to its initial State. What it implies that CountDownLatch is good for one time event like application start-up time and CyclicBarrier can be used to in case of recurrent event e.g. concurrently calculating solution of big problem etc. If you like to learn more about threading and concurrency in Java you can also check my post on [When to use Volatile variable in Java](http://javarevisited.blogspot.sg/2011/06/volatile-keyword-java-example-tutorial.html) and [How Synchronization works in Java](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html).

## **CyclicBarrier in Java – Example**

Now we know what is CyclicBarrier in Java and it's time to see example of CyclicBarrier in Java. Here is a simple example of CyclicBarrier in Java on which we initialize CyclicBarrier with 3 parties, means in order to cross barrier, 3 thread needs to call await() method. each thread calls await method in short duration but they don't proceed until all 3 threads reached barrier, once all thread reach barrier, barrier gets broker and each [thread](http://javarevisited.blogspot.sg/2011/10/how-to-stop-thread-java-example.html) started there execution from that point. Its much clear with the output of following example of CyclicBarrier in Java:

**import** java.util.concurrent.BrokenBarrierException;  
**import** java.util.concurrent.CyclicBarrier;  
**import** java.util.logging.Level;  
**import** java.util.logging.Logger;  
  
/\*\*  
 \* Java program to demonstrate how to use CyclicBarrier in Java. CyclicBarrier is a

 \* new Concurrency Utility added in Java 5 Concurrent package.

 \*  
 \* @author Javin Paul  
 \*/  
**public** **class** CyclicBarrierExample {  
  
    *//Runnable task for each thread*  
    **private** **static** **class** Task **implements** [**Runnable**](http://javarevisited.blogspot.sg/2012/01/difference-thread-vs-runnable-interface.html) {  
  
        **private** **CyclicBarrier** barrier;  
  
        **public** Task(**CyclicBarrier** barrier) {  
            **this**.barrier = barrier;  
        }  
  
        @**Override**  
        **public** **void** run() {  
            **try** {  
                **System**.out.println(**Thread**.currentThread().getName() + " is waiting on barrier");  
                barrier.await();  
                **System**.out.println(**Thread**.currentThread().getName() + " has crossed the barrier");  
            } **catch** (**InterruptedException** ex) {  
                **Logger**.getLogger(CyclicBarrierExample.**class**.getName()).log(**Level**.SEVERE, **null**, ex);  
            } **catch** (**BrokenBarrierException** ex) {  
                **Logger**.getLogger(CyclicBarrierExample.**class**.getName()).log(**Level**.SEVERE, **null**, ex);  
            }  
        }  
    }  
  
    **public** **static** **void** main(**String** args[]) {  
  
        *//creating CyclicBarrier with 3 parties i.e. 3 Threads needs to call await()*  
        **final** **CyclicBarrier** cb = **new** **CyclicBarrier**(3, **new** **Runnable**(){  
            @**Override**  
            **public** **void** run(){  
                *//This task will be executed once all thread reaches barrier*  
                **System**.out.println("All parties are arrived at barrier, lets play");  
            }  
        });  
  
        *//starting each of thread*  
        **Thread** t1 = **new** **Thread**(**new** Task(cb), "Thread 1");  
        **Thread** t2 = **new** **Thread**(**new** Task(cb), "Thread 2");  
        **Thread** t3 = **new** **Thread**(**new** Task(cb), "Thread 3");  
  
        t1.start();  
        t2.start();  
        t3.start();  
        
    }  
}  
  
**Output:**  
**Thread** 1 is waiting on barrier  
**Thread** 3 is waiting on barrier  
**Thread** 2 is waiting on barrier  
All parties are arrived at barrier, lets play  
**Thread** 3 has crossed the barrier  
**Thread** 1 has crossed the barrier  
**Thread** 2 has crossed the barrier

**When to use CyclicBarrier in Java**

Given the nature of CyclicBarrier it can be very handy to implement map reduce kind of task similar to [fork-join framework of Java 7](http://javarevisited.blogspot.sg/2011/09/fork-join-task-java7-tutorial.html), where a big task is broker down into smaller pieces and to complete the task you need output from individual small task e.g. to count population of India you can have 4 threads which counts population from North, South, East and West and once complete they can wait for each other, When last thread completed there task, Main thread or any other thread can add result from each zone and print total population. You can use CyclicBarrier in Java :

1) To implement multi player game which can not begin until all player has joined.

2) Perform lengthy calculation by breaking it into smaller individual tasks, In general to implement Map reduce technique.

**Important point of CyclicBarrier in Java**

1. CyclicBarrier can perform a completion task once all thread reaches to barrier, This can be provided while creating CyclicBarrier.

2. If CyclicBarrier is initialized with 3 parties means 3 thread needs to call await method to break the barrier.

3. [Thread will block](http://javarevisited.blogspot.sg/2012/02/what-is-blocking-methods-in-java-and.html) on await() until all parties reaches to barrier, another thread interrupt or await timed out.

4. If another thread interrupt the thread which is waiting on barrier it will throw BrokernBarrierException as shown below:

java.util.concurrent.**BrokenBarrierException**  
        at java.util.concurrent.**CyclicBarrier**.dowait(**CyclicBarrier**.java:172)  
        at java.util.concurrent.**CyclicBarrier**.await(**CyclicBarrier**.java:327)

5.CyclicBarrier.reset() put Barrier on its initial state, other thread which is waiting or not yet reached barrier will terminate with java.util.concurrent.BrokenBarrierException.

That's all on  What is CyclicBarrier in Java , When to use CyclicBarrier in Java and a Simple Example of How to use CyclicBarrier in Java . We have also seen difference between CountDownLatch and CyclicBarrier in Java and got some idea where we can use CyclicBarrier in Java Concurrent code.

# [Difference between get and load in Hibernate](http://javarevisited.blogspot.com/2012/07/hibernate-get-and-load-difference-interview-question.html)

**get vs load in Hibernate**

Difference between get and load method in Hibernate is a one of the most popular question asked in Hibernate and [spring interviews](http://javarevisited.blogspot.fr/2011/09/spring-interview-questions-answers-j2ee.html). Hibernate Session  class provides two method to access object e.g. session.get() and session.load() both looked quite similar to each other but there are subtle difference between load and get method which canaffect performance of application. Main **difference between get() vs load method** is that get() involves database hit if object doesn't exists in Session Cache and returns a fully initialized object which may involve several [database](http://javarevisited.blogspot.sg/2011/11/database-transaction-tutorial-example.html) call while load method can return proxy in place and only initialize the object or hit the database if any method other thangetId() is called on persistent or entity object. This [lazy initialization](http://javarevisited.blogspot.sg/2011/03/10-interview-questions-on-singleton.html) can save couple of database round-trip which result in better performance. By the way there are many articles on interview questions in Java, you can use search button on top left to find them. Some of them like [20 design pattern interview questions](http://javarevisited.blogspot.sg/2012/06/20-design-pattern-and-software-design.html) and [10 Singleton pattern questions](http://javarevisited.blogspot.sg/2011/03/10-interview-questions-on-singleton.html)are my favorites, you may also like. Coming back to article, you can find more difference between load and get in rest of this article in point format but this is the one which really makes difference while comparing both of them. If you look at how get and load gets called its pretty identical.

## **Difference between get and load method**

Here are few differences between get and load method in Hibernate.

**1. Behavior when Object is not found in Session Cache**

Apart from performance this is another difference between get and load which is worth remembering. get method of Hibernate Session class returns null if object is not found in cache as well as on database while load() method [throws](http://javarevisited.blogspot.sg/2012/02/difference-between-throw-and-throws-in.html) ObjectNotFoundException if object is not found on cache as well as on database but never return null.

**2. Database hit**

Get method always hit [database](http://javarevisited.blogspot.sg/2011/10/how-to-use-truncate-and-delete-command.html) while load() method may not always hit the database, depending upon which method is called.

**3. Proxy**

Get method never returns a proxy, it either returns null or fully initialized Object, while load() method may return proxy, which is the object with ID but without initializing other properties, which is lazily initialized. If you are just using returned object for creating relationship and only need Id then load() is the way to go.

**4. Performance**

By far most important difference between get and load in my opinion. get method will return a completely initialized object if  Object is not on the cache but exists on [Database](http://javarevisited.blogspot.sg/2011/10/selct-command-sql-query-example.html), which may involve multiple round-trips to database based upon object relational mappings while load() method of Hibernate can return a **proxy** which can be initialized on demand (lazy initialization) when a non identifier method is accessed. Due to above reason use of load method will result in slightly **better performance**, but there is a caveat that proxy object will throw ObjectNotFoundException later if corresponding row doesn’t exists in database, instead of failing immediately so not a [fail fast](http://javarevisited.blogspot.sg/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) behavior.

5. load method exists prior to get method which is added on user request.

**When to use Session get() and load() in Hibernate**

So far we have discussed how get and load are different to each other and how they can affect performance of your web application, after having this information in our kitty we can see some best practices to get most of load and get together. This section suggest some scenario which help you when to use get and load in Hibernate.

1. Use get method to determine if an [instance](http://javarevisited.blogspot.sg/2012/02/difference-between-instance-class-and.html) exists or not because it can return null if instance doesn’t exists in cache and database and use load method to retrieve instance only if you think that instance should exists and non availability is an error condition.

2.  As stated in difference number 2 between get and load in Hibernate. get() method could suffer performance penalty if only identifier method like getId()  is accessed. So consider using load method  if  your [code](http://javarevisited.blogspot.sg/2011/09/code-review-checklist-best-practice.html) doesn't access any method other than identifier or you are OK with lazy initialization of object, if persistent object is not in Session Cache because load() can return proxy.

**How to call get records in Hibernate using get and load method**

If you look at below code , there is not much difference on calling get() and load() method, though both are [overloaded](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html) now and can accept few more parameters but the primary methods looks exactly identical. It’s there behavior which makes them different.

*//Example of calling get method of Hiberante Session class*  
**Session** session = SessionFactory.getCurrentSession();  
**Employee** Employee = (Employee) session.get(Employee.**class**, EmployeeID);  
  
*//Example of calling load method of Hiberante Session*  
**Session** session = SessionFactory.getCurrentSession();  
**Employee** Employee = (Employee) session.load(Employee.**class**, EmployeeID);

That’s all on **difference between get and load in Hibernate**. No doubt Hibernate is a great tool for Object relational mapping but knowing this subtle differences can greatly help to improver performance of your J2EE application, apart from practical reason get vs load method is also frequently asked questions in Hibernate interview, so familiarity with differences between load and get certainly helps.

# [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), 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.

## **EnumMap vs HashMap**

Before looking differences between EnumMap and HashMap, few words about What is common between them. Both of them implements [Map interface](http://javarevisited.blogspot.sg/2012/07/create-read-only-list-map-set-example-java.html) so they can be used in all methods which accept Map and data can be accessed using common Map methods e.g. get() and put(). Internally EnumMap is represented using Array and provides constant time performancefor common methods e.g. get() or put(). Now let's see few differences between EnumMap vs HashMap :

1) As said earlier, first and foremost difference between EnumMap and HashMap is that, EnumMap is **optimized for enum** keys while HashMap is a general purpose Map implementation similar to [Hashtable](http://javarevisited.blogspot.sg/2012/01/java-hashtable-example-tutorial-code.html). you can not use any type other than [Enum](http://javarevisited.blogspot.gr/2012/07/why-enum-singleton-are-better-in-java.html) as key in EnumMap but you can use both Enum and any other Object as key in HashMap.

2) Another difference between EnumMap and HashMap is **performance**. as discussed in previous point, due to specialized optimization done for Enum keys, EnumMap is likely to perform better than HashMap when using enum as key object.

3) One more thing which can be considered as difference between HashMap and EnumMap is **probability of Collision**. Since Enum is internally maintain as array and they are stored in there natural order using ordinal(), as shown in following code which is taken from put() method of EnumMap

    int index = ((Enum)key).ordinal();

    Object oldValue = vals[index];

    vals[index] = maskNull(value);

Since EnumMap doesn't call [hashCode method on keys](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html), there is no chance of collision.

These were some notable **difference between EnumMap and HashMap in Java**. In short EnumMap is best suited for enum keys, for which it has optimized and perform better than HashMap in Java. Use EnumMap whenever you can use enum as keys.

# [Difference between trustStore and keyStore in Java - SSL](http://javarevisited.blogspot.com/2012/09/difference-between-truststore-vs-keyStore-Java-SSL.html)

**trustStore vs keyStore in Java**

trustStore and keyStore are used in context of setting up SSL connection in Java application between client and server. TrustStore and keyStore are very much similar in terms of construct and structure as both are managed by [keytoolcommand](http://java67.blogspot.sg/2012/09/keytool-command-examples-java-add-view-certificate-ssl.html) and represented by KeyStore programatically but they often confused Java programmer both beginners and intermediate alike. Only *difference between trustStore and keyStore* is what they store and there purpose. In SSL handshake purpose of **trustStore is to verify credentials** and purpose of**keyStore is to provide credential**. keyStore in Java stores private key and certificates corresponding to there public keys and require if you are SSL Server or SSL requires client authentication. TrustStore stores certificates from third party, your Java application communicate or certificates signed by CA(certificate authorities like Verisign, Thawte, Geotrust or GoDaddy) which can be used to identify third party. This is second article on setting up SSL on Java program, In last post we have seen [How to import SSL certificates into trustStore and keyStore](http://javarevisited.blogspot.com/2012/03/add-list-certficates-java-keystore.html) and In this Java article we will some differences between keystore and truststore in Java, which will help to understand this concept better.

## **Difference between trustStore and keyStore in Java**

Here is the list of most *common difference between keyStore and trustStore*. I have already mentioned key difference in first paragraph which is related to purpose of keyStore and trustStore, which we will see here is little more detail.

1)First and major difference between trustStore and keyStore is that trustStore is used by TrustManager and keyStore is used by KeyManager [class in Java](http://javarevisited.blogspot.sg/2011/10/class-in-java-programming-general.html). KeyManager and TrustManager performs different job in Java, TrustManager determines whether remote connection should be trusted or not i.e. whether remote party is who it claims to and KeyManager decides which [authentication credentials](http://javarevisited.blogspot.sg/2012/03/why-character-array-is-better-than.html) should be sent to the remote host for authentication during SSL handshake. if you are an SSL Server you will use private key during key exchange algorithm and send certificates corresponding to your public keys to client, this certificate is acquired from keyStore. On SSL client side, if its written in Java, it will use certificates stored in trustStore to verify identity of Server. SSL certificates are most commonly comes as .cer file which is added into keyStore or trustStore by using any key management utility e.g. [keytool](http://java67.blogspot.sg/2012/09/keytool-command-examples-java-add-view-certificate-ssl.html). See my post [How to add certificates into trustStore](http://javarevisited.blogspot.sg/2012/03/add-list-certficates-java-keystore.html) for step by step guide on adding certificates into keyStore or trustStore in Java.

2) Another difference between trustStore and keyStore in rather simple terms is that keyStore contains private keys and required only if you are running a Server in SSL connection or you have enabled [client authentication](http://javarevisited.blogspot.sg/2011/11/ldap-authentication-active-directory.html) on server side. On the other hand trustStore stores public key or certificates from CA (Certificate Authorities) which is used to trust remote party or SSL connection.

3)One more difference between trustStore vs KeyStore is that we use -Djavax.net.ssl.keyStore to specify [path](http://javarevisited.blogspot.sg/2011/10/how-to-set-path-for-java-unix-linux-and.html) for keyStore and -Djavax.net.ssl.trustStore to specify path for trustStore in Java.

4) Another difference between trustStore and keyStore is that, If you store your personal certificate along with signer certificate in trustStore,  you can use same file as both trustStore and keyStore. By the way its good idea to separate personal certificate and signer certificates in keyStore and trustStore for better management.

5) One more API level difference between keyStore and trustStore is that  password of keyStore is provided using -Djavax.net.ssl.keyStorePassword and password of trustStore is provided using -Djavax.net.ssl.trustStorePassword.

That’s all on difference between trustStore and keyStore in Java. You can still use same file as trustStore and keyStore in Java to avoid maintaining two separate files, but its good idea to segregate public keys and private keys in two different files, its more verbose and self explanatory that which one holds CA certificates to trust server and which contains client's private keys.

# [Difference between save vs persist and saveOrUpdate in Hibernate](http://javarevisited.blogspot.com/2012/09/difference-hibernate-save-vs-persist-and-saveOrUpdate.html)

**Save vs saveOrUpdate vs persist in Hibernate**

What is difference between save and saveOrUpdate or Difference between save and persist are common interview question in any Hibernate interview, much like [Difference between get and load method in Hibernate](http://javarevisited.blogspot.sg/2012/07/hibernate-get-and-load-difference-interview-question.html). Hibernate Session class provides couple of ways to save object into database by methods like save, saveOrUpdate and persist. You can use either save(),  saveOrUpdate() or persist() based upon your requirement for persisting object into [Database](http://javarevisited.blogspot.sg/2011/11/database-transaction-tutorial-example.html). Along with [Spring framework Interview questions](http://javarevisited.blogspot.sg/2011/09/spring-interview-questions-answers-j2ee.html), Hibernate questions are also quite popular on J2EE interviews because of its status as leading ORM. It’s good to prepare some questions from Hibernate before appearing in any[J2EE interviews](http://javarevisited.blogspot.sg/2011/09/servlet-interview-questions-answers.html). One of them is *Difference between save , saveOrUpdate and persist*, which we will see in this Hibernate article.

**Difference between save and saveOrUpdate in Hibernate**

Main *difference between save and saveOrUpdate method* is that save() generates a **new identifier** and INSERT record into [database](http://javarevisited.blogspot.sg/2011/10/selct-command-sql-query-example.html) while saveOrUpdate can either INSERT or UPDATE based upon existence of record. Clearly saveOrUpdate is more flexible in terms of use but it involves an extra processing to find out whether recordalready exists in table or not. In summary  save() method saves records into database by INSERT SQL query, Generates a new identifier and return the [Serializable](http://javarevisited.blogspot.sg/2011/04/top-10-java-serialization-interview.html) identifier back. On the other hand  saveOrUpdate() method either INSERT or UPDATE based upon existence of object in database. If persistence object already exists indatabase then UPDATE SQL will [execute](http://javarevisited.blogspot.sg/2012/03/how-to-create-and-execute-jar-file-in.html) and if there is no corresponding object in database than INSERT will run.

## **Difference between save and persist method in Hibernate**

In last section we saw What are difference between save and saveOrUpdate and now we will see Difference on save vs persist method.

1)First difference between save and persist is there return type. Similar to save method persist also INSERT records into database but **return type of persist is void** while return type of save is [Serializable](http://javarevisited.blogspot.sg/2012/01/serializable-externalizable-in-java.html) object.

2) Another difference between persist and save is that both methods make a [transient](http://javarevisited.blogspot.sg/2012/03/difference-between-transient-and.html) instance persistent. However, persist() method doesn't guarantee that the identifier value will be assigned to the persistent instance immediately, the assignment might happen at flush time.

3) One more thing which differentiate persist and save method in Hibernate is that is there behavior on outside of transaction boundaries. persist() method guarantees that it will not execute an INSERT statement if it is called outside of [transaction boundaries](http://javarevisited.blogspot.sg/2011/11/database-transaction-tutorial-example.html). save() method does not guarantee the same, it returns an identifier, and if an INSERT has to be executed to get the identifier (e.g. "identity"generator), this INSERT happens immediately, no matter if you are inside or outside of a transaction.

4) Fourth difference between save and persist method in Hibernate is related to previous difference on save vs persist. Because of its above behavior of persist method outside transaction boundary, its useful in long-running conversations with an extended Session context. On the other hand save method is not good in a long-running conversation with an extended Session context.

These were some **differences between save, saveOrUpdate and persist method of Hibernate**. All three method are related to saving Object into database but there behavior are quite different. Knowledge of save, persist and saveOrUpdate not only helps to decide better use of Hibernate API but also help you to do well in Hibernate interviews.

## Interview questions on Java Garbage collection

[Java Garbage collection Interview Question Answer for 4+ experience](http://3.bp.blogspot.com/-K6q0DQ1v-tw/TWu8owBtc2I/AAAAAAAAADA/oBoHDBiJ8ag/s1600/17.jpg)Here is some Garbage collection Interview questions from my personal collection, which I have created from my experience and with the help of various friends and colleagues which has shared *GC interview questions* with me. Actually there are lot many questions than What I am sharing here but to keep this post small I thought to only share some questions, I can think of second part of GC interview question if you guys find this useful.

**Question 1 - What is structure of Java Heap ? What is Perm Gen space in Heap ?**

Answer : In order to better perform in Garbage collection questions in any Java interview, It’s important to have basic understanding of  Java Heap space. To learn more about heap, see my post [10 points on Java heap space](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html). By the way Heap is divided into different generation e.g. new generation, old generation and PermGen space.PermGen space is used to store class’s metadata and filling of PermGen space can cause [java.lang.OutOfMemory:PermGen space](http://javarevisited.blogspot.sg/2012/01/tomcat-javalangoutofmemoryerror-permgen.html). Its also worth noting to remember [JVM option to configure PermGen](http://javarevisited.blogspot.sg/2011/09/javalangoutofmemoryerror-permgen-space.html) space in Java.

**Question 2 - How do you identify minor and major garbage collection in Java?**

Answer: Minor collection prints “GC” if garbage collection [logging](http://javarevisited.blogspot.sg/2011/05/top-10-tips-on-logging-in-java.html) is enable using –verbose:gc or -XX:PrintGCDetails, while Major collection prints “Full GC”. This Garbage collection interview question is based on understanding of Garbage collection output. As more and more Interviewer are asking question to check candidate’s ability to understand GC output, this topic become even more important.

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

Answer : This *Garbage Collection interview questions* is recently asked to one of my friend. It require more than average knowledge on GC to answer this question. 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 4 - How do you find GC resulted due to calling System.gc()?**

Answer : Another GC interview question which is based on GC output. Similar to major and minor collection, there will be a word “System” included in Garbage collection output.

**Question 5 - What is difference between Serial and Throughput Garbage collector?**

Answer : Serial Garbage collector is a stop the world GC which stops application thread from running during both [minor and major collection](http://javarevisited.blogspot.sg/2011/04/garbage-collection-in-java.html). Serial Garbage collector can be enabled using JVM option -XX:UseSerialGC and it's designed for Java application which doesn't have pause time requirement and have client configuration. **Serial Garbage collector** was also default GC in JDK 1.4 before ergonomics was introduced in JDK 1.5. Serial GC is most suited for small application with less number of [thread](http://javarevisited.blogspot.sg/2011/02/how-to-implement-thread-in-java.html) while throughput GG is more suited for large applications. On the other hand Throughput garbage collector is parallel collector where minor and major collection happens in parallel taking full advantage of all the system resources available like multiple processor. Though both major and minor collection runs on stop-the-world fashion and introduced pause in application. Throughput Garbage collector can be enable using -XX:UseParallelGC or -XX:UseOldParallelGC. It increases overall throughput of application my minimizing time spent in Garbage collection but still has long pauses during full GC.This is a kind of *Garbage collection interview questions* which gives you an opportunity to show your knowledge in detail while answering. I always suggest to answer these kind of questions in detail.

**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. Cyclic reference doesn’t count as live reference and if two objects are pointing to each other and there is no live reference for any of them, than both are eligible for GC. Also Garbage collection thread is a [daemon thread](http://javarevisited.blogspot.sg/2012/03/what-is-daemon-thread-in-java-and.html) which will run by JVM based upon GC algorithmand when runs it collects all objects which are eligible for GC.

**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 8 - If Object A has reference to Object B and Object B refer to Object A, apart from that there is no live reference to either object A or B, Does they are eligible to Garbage collection ?**

This Garbage collection interview questions is related question 5 “When object become eligible for Garbage collection”. An object becomes eligible for Garbage collection if there is no live reference for it. It can not be accessible from any Thread and cyclic dependency doesn’t prevent Object from being Garbage collected. Which means in this case both Object A and Object B are eligible of Garbage collection. See [How Garbage collection works in Java](http://javarevisited.blogspot.com/2011/04/garbage-collection-in-java.html) for more details.

**Question 9 -Can we force Garbage collector to run at any time ?**

Answer : No, you can not force Garbage collection in Java. Though you can request it by calling Sytem.gc() or its cousin Runtime.getRuntime().gc(). It’s not guaranteed that GC will run immediately as result of calling these method.

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

Answer : This  is a tricky Garbage collection interview question as many programmers are not sure whether PermGen space is part of [Java heap space](http://javarevisited.blogspot.sg/2011/08/increase-heap-size-maven-ant.html) or not and since it maintains class Meta data and String pool, whether its eligible for garbage collection or not. 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.

**Question 11 : How to you monitor garbage collection activities?**

Answer : One of my favorite interview questions on [Garbage collection](http://www.blogger.com/javarevisited.blogspot.in/2011/04/garbage-collection-in-java.html), just to check whether candidate has ever monitored GC activities or not. You can monitor garbage collection activities either offline or real-time. You can use tools like **JConsole** and **VisualVM** VM with its Visual GC plug-in to monitor real time garbage collection activities and memory status of JVM or you can redirect Garbage collection output to a log file for offline analysis by using -XlogGC=&lt;PATH&gt; JVM parameter. Anyway you should always enable GC options like -XX:PrintGCDetails -X:verboseGC and -XX:PrintGCTimeStamps as it doesn't impact [application performance](http://javarevisited.blogspot.sg/2012/01/improve-performance-java-database.html) much but provide useful states for performance monitoring.

**Question 12: Look at below Garbage collection output and answer following question :**

[GC

       [ParNew: 1512K->64K(1512K), 0.0635032 secs]

       15604K->13569K(600345K), 0.0636056 secs]

       [Times: user=0.03 sys=0.00, real=0.06 secs]

 1. Is this output of Major Collection or Minor Collection ?

 2. Which young Generation Garbage collector is used ?

 3. What is size of Young Generation, Old Generation and total Heap Size?

 4. How much memory is freed from Garbage collection ?

 5. How much time is taken for Garbage collection ?

 6. What is current Occupancy of Young Generation ?

This Garbage collection Interview questions is completely based on GC output. Following are answers of above GC questions which will not only help you to answer these question but also help you to understand and interpret GC output.

**Answer 1**:  It's Minor collection because of "GC" word, In case of Major collection, you would see "Full GC".

**Answer 2**: This output is of multi-threaded Young Generation Garbage collector "ParNew", which is used along with CMS concurrent Garbage collector.

**Answer 3**: [1512K] which is written in bracket is total size of Young Generation, which include Eden and two survivor space. 1512K on left of arrow is occupancy of Yong Generation before GC and 64K is occupancy after GC. On the next line value if bracket is total heap size which is (600345K). If we subtract size of young generation to total heap size we can calculate size of Old Generation. This line also shows occupancy of heap before and after Garbage collection.

**Answer 4**: As answered in previous garbage collection interview question, second line shows heap occupancy before and after Garbage collection. If we subtract value of right side 13569K, to value on left side 15604K, we can get total memory freed by GC.

**Answer 5**: 0.0636056 secs on second line denotes total time it took to collect dead objects during Garbage collection. It also include time taken to GC young generation which is shown in first line (0635032 secs).

**Answer 6**: 64K

Here are few more interesting *Garbage collection Interview question* for your practice, I haven’t provided answers of all garbage collection interview questions. If you know the answer than you can  post via comments.

Question -  What is difference between -XX:ParallelGC and -XX:ParallelOldGC?

Question - When do you ConcurrentMarkSweep Garbage collector and Throughput GC?

Question -  What is difference between ConcurrentMarkSweep and G1 garbage collector?

Question -  Have you done any garbage collection tuning? What was your approach**?**

These were some Garbage collection interview questions and answers, may help on your Java Interview preparation. If you have got any interesting interview questions related to GC than don’t forget to share with us.

# [Difference between notify and notifyAll in Java - When and How to use](http://javarevisited.blogspot.com/2012/10/difference-between-notify-and-notifyall-java-example.html)

**notify vs notifyAll in Java**

What is difference between notify and notifyAll method is one of the [tricky Java question](http://java67.blogspot.sg/2012/09/top-10-tricky-java-interview-questions-answers.html), which is easy to answer but once Interviewer ask followup questions, you either got confused or not able to provide clear cut and to the point answers. Main difference between notify and notifyAll is that notify method will only notify one [Thread](http://javarevisited.blogspot.sg/2011/02/how-to-implement-thread-in-java.html) and notifyAllmethod will notify all Threads  which are waiting on that monitor or lock. By the way this is something you have been reading in all over places and to be frank,  this statement despite being correct is not complete and its very difficult to understand *difference between notify vs notifyAll* by just reading this statement. Lot of questions comes in mind like

Which thread will be notified if I use notify()?

How do I know how many threads are waiting, so that I can use notifyAll() ?

How to call notify()?

What are these thread waiting for being notified etc.

Actually discussion of notify and notifyAll is incomplete without discussing wait method in Java and I had touched based on this on my earlier article [why wait and notify must be called from synchronized context](http://javarevisited.blogspot.sg/2011/05/wait-notify-and-notifyall-in-java.html). In order to get answer of those questions and understand difference between notify and notifyAll we will use a simple Java Thread example using wait and notify code :

**Difference between notify and notifyAll in Java**

Java provides two methods notify and notifyAll for waking up threads waiting on some condition and you can use any of them but there is subtle difference between notify and notifyAll in Java which makes it one of the [popular multi-threading interview question in Java](http://javarevisited.blogspot.sg/2011/07/java-multi-threading-interview.html). When you call notify only one of waiting thread will be woken and its not guaranteed which thread will be woken, it depends upon Thread scheduler. While if you call notifyAll method, all threads waiting on that lock will be woken up, but again all woken thread will fight for lock before executing remaining code and that's why wait is called on loop because if multiple threads are woken up, the thread which will get lock will first execute and it may reset waiting condition, which will force subsequent threads to [wait](http://javarevisited.blogspot.sg/2011/12/difference-between-wait-sleep-yield.html).So key difference between notify and notifyAll is that notify() will cause only one thread to wake up while notifyAll method will make all thread to wake up.

**When to use notify and notifyAll in Java**

This is the follow-up question if you get pass the earlier one *Difference between notifyAll and notify in Java*. If you understand notify vs notifyAll then you can answer this by applying little common sense. You can use notify over notifyAll if all thread are waiting for same condition and only one Thread at a time can benefit from condition becoming true. In this case notify is optimized call over notifyAll because waking up all of them because we know that only one thread will benefit and all other will wait again, so calling notifyAll method is just waste of cpu cycles. Though this looks quite reasonable there is still a caveat that unintended recipient swallowing critical notification. by using notifyAll we ensure that all recipient will get notify. Josh bloach has explained this in good detail in his book Effective Java , I highly recommend this book if you haven't read them already. Another one you can try is Concurrency Practice in Java and Java Thread, which discusses [wait and notify methods](http://javarevisited.blogspot.sg/2012/02/why-wait-notify-and-notifyall-is.html) in good details.

## **Example of notify and notifyAll method in Java**

I have put together an example to show how all threads gets notified when we call notifyAll method in Java and just one Thread will wake up when we call notify method in Java. In this example three threads will wait if boolean variable go is false, remember boolean go is a [volatile variable](http://javarevisited.blogspot.sg/2011/06/volatile-keyword-java-example-tutorial.html), so that all threads will see its updated value. Initially three threads WT1, WT2, WT3 will wait because variable go isfalse than one thread NT1 will make go true and notify all threads by calling notifyAll method or notify just one thread by calling notify() method. In case of notify() call there is no guarantee which thread will woke up and you can see it by running this Java program multiple times. In case of notifyAll all thread will woke up but they will compete for monitor or lock and the Thread which will get the lock first will finish its execution and resetting go to false which will force other two threads still waiting. At the end of this program you will have two threads waiting and two threads including notification thread finished. Program will not terminate because other two threads are still waiting and they are not [daemon threads](http://javarevisited.blogspot.sg/2012/03/what-is-daemon-thread-in-java-and.html). Purpose of this notify and notifyAll example is to show you How to use them and How notify and notifyAll method works in Java.

**Code Example of notify and notifyAll**

Here is complete code example of How to use notify and notifyAll method in Java. We have already explained when to use notify vs notifyAll method and this example will clarify effect of calling notify and notifyAll method  in Java.

**import** java.util.logging.Level;  
**import** java.util.logging.Logger;  
  
/\*\*  
 \* **Java program to demonstrate How to use notify and notifyAll method in Java** and

 \* How notify and notifyAll method notifies thread, which thread gets woke up etc.  
 \*/  
**public** **class** NotificationTest {  
  
    **private** **volatile** **boolean** go = **false**;  
  
    **public** **static** **void** main(**String** args[]) **throws** **InterruptedException** {  
        **final** NotificationTest test = **new** NotificationTest();  
        
        **Runnable** waitTask = **new** **Runnable**(){  
        
            @**Override**  
            **public** **void** run(){  
                **try** {  
                    test.shouldGo();  
                } **catch** (**InterruptedException** ex) {  
                    **Logger**.getLogger(NotificationTest.**class**.getName()).

                           log(**Level**.SEVERE, **null**, ex);  
                }  
                **System**.out.println(**Thread**.currentThread() + " finished Execution");  
            }  
        };  
        
        **Runnable** notifyTask = **new** **Runnable**(){  
        
            @**Override**  
            **public** **void** run(){  
                test.go();  
                **System**.out.println(**Thread**.currentThread() + " finished Execution");  
            }  
        };  
        
        **Thread** t1 = **new** **Thread**(waitTask, "WT1"); *//will wait*  
        **Thread** t2 = **new** **Thread**(waitTask, "WT2"); *//will wait*  
        **Thread** t3 = **new** **Thread**(waitTask, "WT3"); *//will wait*  
        **Thread** t4 = **new** **Thread**(notifyTask,"NT1"); *//will notify*  
        
        *//starting all waiting thread*  
        t1.start();  
        t2.start();  
        t3.start();  
        
        *//pause to ensure all waiting thread started successfully*  
        **Thread**.sleep(200);  
        
        *//starting notifying thread*  
        t4.start();  
        
    }  
    */\*  
     \* wait and notify can only be called from synchronized method or bock  
     \*/*  
    **private** **synchronized** **void** shouldGo() **throws** **InterruptedException** {  
        while(go != **true**){  
            **System**.out.println(**Thread**.currentThread()

                         + " is going to wait on this object");  
            wait(); *//release lock and reacquires on wakeup*  
            **System**.out.println(**Thread**.currentThread() + " is woken up");  
        }  
        go = **false**; *//resetting condition*  
    }  
    
    */\*  
     \* both shouldGo() and go() are locked on current object referenced by "this" keyword  
     \*/*  
    **private** **synchronized** **void** go() {  
        while (go == **false**){  
            **System**.out.println(**Thread**.currentThread()

            + " is going to notify all or one thread waiting on this object");  
  
            go = **true**; *//making condition true for waiting thread*  
            *//notify(); // only one out of three waiting thread WT1, WT2,WT3 will woke up*  
            notifyAll(); *// all waiting thread  WT1, WT2,WT3 will woke up*  
        }  
        
    }  
  }  
  
**Output in case of using notify**  
**Thread**[WT1,5,main] is going to wait on **this** object  
**Thread**[WT3,5,main] is going to wait on **this** object  
**Thread**[WT2,5,main] is going to wait on **this** object  
**Thread**[NT1,5,main] is going to notify all or one thread waiting on **this** object  
**Thread**[WT1,5,main] is woken up  
**Thread**[NT1,5,main] finished Execution  
**Thread**[WT1,5,main] finished Execution  
  
**Output in case of calling notifyAll**  
**Thread**[WT1,5,main] is going to wait on **this** object  
**Thread**[WT3,5,main] is going to wait on **this** object  
**Thread**[WT2,5,main] is going to wait on **this** object  
**Thread**[NT1,5,main] is going to notify all or one thread waiting on **this** object  
**Thread**[WT2,5,main] is woken up  
**Thread**[NT1,5,main] finished Execution  
**Thread**[WT3,5,main] is woken up  
**Thread**[WT3,5,main] is going to wait on **this** object  
**Thread**[WT2,5,main] finished Execution  
**Thread**[WT1,5,main] is woken up  
**Thread**[WT1,5,main] is going to wait on **this** object

I strongly recommend to run this Java program and understand output produce by it and try to understand it. Theory should complement practical example and if you see any inconsistency than let us know or try to rectify it. Along with [deadlock](http://javarevisited.blogspot.sg/2010/10/what-is-deadlock-in-java-how-to-fix-it.html), [race condition](http://javarevisited.blogspot.sg/2012/02/what-is-race-condition-in.html) and [thread-safety](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html),  inter thread communication is one of the fundamental of concurrent programming in Java.

**Summary**

In short here are answers of questions on notify and notifyAll we have raised at the start of this tutorial:

**Which thread will be notified if I use notify()?**

No guaranteed,  ThreadScheduler will pick a random thread from pool of waiting thread on that monitor. What is guaranteed is that only one Thread will be notified.

**How do I know how many threads are waiting, so that I can use notifyAll() ?**

Its depend upon your application logic, while coding you need to think whether a piece of code can be run by multiple thread or not. A good example to understand inter-thread communication is implementing [producer consumer pattern in Java](http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html).

**How to call notify()?**

Wait() and notify() method can only be called from [synchronized method or block](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html), you need to call notify method on object on which other threads are waiting.

**What are these thread waiting for being notified etc.**

Thread wait on some condition e.g. in producer consumer problem, producer thread wait if shared queue is full and consumer thread wait if shared queue is empty. Since multiple thread are working with a shared resource they communicate with each other using wait and notify method.

That’s all on *What is difference between notify and notifyAll method in Java*  and when to use notify vs notifyAll in Java. Now you should be able to understand and use notify and notifyAll method for inter-thread communication in your Java program.

# [What is difference between java.sql.Time, java.sql.Timestamp and java.sql.Date - JDBC interview Question](http://javarevisited.blogspot.com/2012/10/difference-between-javasqltime-date-timestamp-jdbc-interview-question.html)

Difference between java.sql.Time, java.sql.Timestamp and java.sql.Date  is most common JDBC question appearing on many [core Java interviews](http://javarevisited.blogspot.sg/2011/04/top-20-core-java-interview-questions.html). As JDBC provides three classes java.sql.Date, java.sql.Time and java.sql.Timestamp to represent date and time and you already have java.util.Date which can represent both date and time, this question poses lot of confusion among Java programmer and that’s why this is one of those [tricky Java questions](http://java67.blogspot.in/2012/09/top-10-tricky-java-interview-questions-answers.html) which is tough to answer. It becomes really tough if differences between them is not understood correctly. We have already seen some frequently asked or common JDBC questions like [why JDBC has java.sql.Date despite java.util.Date](http://javarevisited.blogspot.sg/2012/04/difference-between-javautildate-and.html) and [Why use PreparedStatement in Java](http://javarevisited.blogspot.sg/2012/03/why-use-preparedstatement-in-java-jdbc.html) in our last tutorials and we will see difference between java.sql.Date, java.sql.Time and java.sql.Timestamp in this article. By the way apart from these JDBC interview questions, if you are looking to get most from JDBC you can also see [4 JDBC performance tips](http://javarevisited.blogspot.in/2012/01/improve-performance-java-database.html) and [10 JDBC best practices to follow](http://javarevisited.blogspot.in/2012/08/top-10-jdbc-best-practices-for-java.html). Those article not only help you to understand and use JDBC better but also help on interviews. Let’s come back to difference sql time, timestamp and sql date.

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

JDBC in Java has three date/time types corresponding to DATE, TIME and TIMESTAMP type of ANSI SQL. These types are used to convert SQL types into Java types.

1) First difference on java.sql.Time vs java.sql.Timestamp vs java.sql.Date is about information they represent :

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.

2) java.sql.Time and java.sql.Timestamp extends [java.util.Date](http://javarevisited.blogspot.in/2011/09/convert-date-to-string-simpledateformat.html) class but java.sql.Date is independent.

3) Time information from java.sql.Date and Date information from java.sql.Time is normalized and may set to zero in order to confirm ANSI SQL DATE and TIME types.

So difference between Time, Timestamp and Date of SQL package is clear in terms of what they represent. On contrary java.util.Date also represent Date and time information but **without nanosecond details** and that's why many people prefer to store date as long value (millisecond passed from epoch January 1, 1970 00:00:00.000 GMT). If you compare to java.sql.Timestamp with [equals() method](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html) itwill return false as value of nanosecond is unknown.

That's all on difference between java.sql.Date, java.sql.Time and java.sql.Timestamp. All differences lies on what exactly the represent. This kinds of questions are worth looking before going to any JDBC interview as time and date are integral part of any JDBC interview.

# [What is difference between BeanFactory and ApplicationContext in Spring framework](http://javarevisited.blogspot.com/2012/11/difference-between-beanfactory-vs-applicationcontext-spring-framework.html)

**BeanFactory vs ApplicationContext**

Difference between BeanFactory and ApplicationContext in Spring framework is a another frequently asked [Spring interview question](http://javarevisited.blogspot.in/2011/09/spring-interview-questions-answers-j2ee.html) mostly asked to Java programmers with 2 to 4 years experience in Java and Spring. Both BeanFactory and ApplicationContext provides way to get bean from Spring IOC container by calling getBean("bean name"), but there are some difference in there working and features provided by them. One difference between bean factory and application context is that former only instantiate bean when you call getBean() method while ApplicationContext instantiate Singleton bean when container is started,  It doesn't wait for getBean to be called. Thisinterview questions is third in my list of frequently asked spring questions e.g. Setter vs Constructor Injection and  [What is default scope of Spring bean](http://javarevisited.blogspot.in/2012/05/what-is-bean-scope-in-spring-mvc.html). If you are preparing for Java interview and expecting some Spring framework question, It’s worth preparing those questions. If you are new in Spring framework and exploring Spring API and classes than you would like check my post on some Spring utility functions e.g. [calculating time difference with StopWatch](http://javarevisited.blogspot.sg/2012/04/how-to-measure-elapsed-execution-time.html) and  [escaping XML special characters using Spring HtmlUtils](http://javarevisited.blogspot.in/2012/09/how-to-replace-escape-xml-special-characters-java-string.html). Coming back to BeanFactory vs ApplicationContext, let’s see some more difference between them in next section.

## **BeanFactory vs ApplicationContext in Spring**

Before seeing difference between ApplicationContext and BeanFactory, let see some similarity between both of them. Spring provides two kinds of IOC container, one is BeanFactory and other is ApplicationContext. Syntactically BeanFactory and ApplicationContext both are [Java interfaces](http://javarevisited.blogspot.in/2012/04/10-points-on-interface-in-java-with.html) and ApplicationContext extends BeanFactory. Both of them are configuration using [XML configuration file](http://javarevisited.blogspot.in/2012/03/how-to-read-properties-file-in-java-xml.html). In short BeanFactory provides basic IOC and DI features while ApplicationContext provides advanced features. Apart from these, Here are few more difference between BeanFactory and ApplicationContext which is mostly based upon features supported by them.

1) BeanFactory doesn't provide support for internationalization i.e. i18n but ApplicationContext provides support for it.

2) Another difference between BeanFactory vs ApplicationContext is ability to publish event to beans that are registered as listener.

3) One of the popular implementation of BeanFactory interface is XMLBeanFactory while one of the popular implementation of ApplicationContext interface is ClassPathXmlApplicationContext. On [Java web application](http://javarevisited.blogspot.sg/2012/08/what-is-jsessionid-in-j2ee-web.html) we use WebApplicationContext  which extends ApplicationContext interface and adds getServletContext method.

4) If you are using auto wiring and using BeanFactory than you need to register AutoWiredBeanPostProcessor using API which you can configure in XML if you are using  ApplicationContext. In summary BeanFactory is OK for testing and non [production](http://javarevisited.blogspot.in/2011/09/how-to-write-production-quality-code.html) use but ApplicationContext is more feature rich container implementation and should be favored over BeanFactory

These were some worth noting difference between BeanFactory and ApplicationContext in Spring framework. In most practical cases you will be using ApplicationContext but knowing about BeanFactory is important to understand fundamental concept of spring framework. I mostly use XML configuration file and ClassPathXmlApplicationContext to quickly run any Spring based Java program from [Eclipse](http://javarevisited.blogspot.sg/2012/10/eclipse-shortcut-to-remove-all-unused-imports-java.html) by using following snippet of code :

**public** **static** **void** main(**String** args[]){  
    ApplicationContext ctx = **new** ClassPathXmlApplicationContext("beans.xml");  
    Hello hello = (Hello) ctx.getBean("hello");  
    hello.sayHello("John");  
}

here beans.xml is your spring configuration file and “hello” is a bean defined in that spring configuration file. Here we have used ClassPathXmlApplicationContext  which is an implementation of ApplicationContext interface in Spring.

# [Difference between Setter vs Constructor Injection in Spring](http://javarevisited.blogspot.com/2012/11/difference-between-setter-injection-vs-constructor-injection-spring-framework.html)

**Spring Setter vs Constructor Injection**

Spring supports two types of dependency Injection, using setter method e.g. setXXX() where XXX is dependency or via constructor argument. First way of dependency injection is known as **setter injection** while later is known as **constructor injection**. Both approaches of Injecting dependency on Spring bean has there pros and cons, which we will see in this Spring framework article. *Difference between Setter Injection and Constructor Injection in Spring* is also a popular [Spring framework interview question](http://javarevisited.blogspot.sg/2011/09/spring-interview-questions-answers-j2ee.html).Some time interviewer also ask as When do you use Setter Injection over Constructor injection in Spring or simply benefits of using setter vs constructor injection in Spring framework. Points discussed in this article not only help you to understand Setter vs Constructor Injection but also Spring's dependency Injection process. By the way if you are new in Spring framework and learning it, you may want to take a look at my list of [5 good books to learn Spring framework](http://javarevisited.blogspot.sg/2013/03/5-good-books-to-learn-spring-framework-mvc-java-programmer.html). That will certainly help on  your learning process. Since Spring is now a must have skill for Java programmers, it worth putting time and effort to learn this powerful framework

## **Difference between Setter and Constructor Injection in Spring framework**

As I said earlier Spring supports both setter and constructor Injection which are two standard way of injecting dependency on beans managed by IOC constructor. Spring framework doesn't support Interface Injection on which dependency is injected by implementing a particular interface. In this section we will see couple of difference between setter and constructor Injection, which will help you decide when to use setter Injection over constructor Injection in Spring and vice-versa.

1) Fundamental difference between setter and constructor injection, as there name implies is How dependency is injected.  Setter injection in Spring uses setter methods like setDependency() to inject dependency on any bean managed by Spring's IOC container. On the other hand constructor injection uses [constructor](http://javarevisited.blogspot.sg/2012/01/what-is-constructor-overloading-in-java.html) to inject dependency on any Spring managed bean.

2) Because of using setter method, setter Injection in more readable than constructor injection in Spring configuration file usually applicationContext.xml . Since setter method has name e.g. setReporotService() by reading Spring XML config file you know which dependency you are setting. While in constructor injection, since it uses index to inject dependency, its not as readable as setter injection and you need to refer either Java documentation or code to find which index corresponds to which property.

3) Another difference between setter vs constructor injection in Spring and one of the drawback of  setter injection is that it does not ensures [dependency Injection](http://javarevisited.blogspot.sg/2012/03/10-object-oriented-design-principles.html). You can not guarantee that certain dependency is injected or not, which means you may have an object with incomplete dependency. On other hand constructor Injection does not allow you to construct object, until your dependencies are ready.

4) One more drawback of setter Injection is Security. By using setter injection, you can [override](http://javarevisited.blogspot.in/2011/12/method-overloading-vs-method-overriding.html) certain dependency which is not possible which is not possible with constructor injection because every time you call constructor, a new object is gets created.  
  
5) One of our reader Murali Mohan Reddy, pointed out one more difference between Setter and Constructor injection in Spring, where later can help, if there is a circular dependency between two object A and B.

If Object A and B are dependent each other i.e A is depends ob B and vice-versa. Spring throws ObjectCurrentlyInCreationException while creating objects of A and B bcz A object cannot be created until B is created and vice-versa. So spring can resolve circular dependencies through setter-injection. Objects constructed before setter methods invoked.

See comment section for more inputs from other readers.

## When to use Setter Injection over Constructor Injection in Spring

Setter Injection has upper hand over Constructor Injection in terms of readability. Since for configuring Spring we use [XML files](http://javarevisited.blogspot.in/2011/12/parse-xml-file-in-java-example-tutorial.html), readability is much bigger concern. Also drawback of setter Injection around ensuring mandatory dependency injected or not can be handled by configuring Spring to check dependency using "dependency-check" attribute of  tag or tag. Another worth noting point to remember while comparing Setter Injection vs Constructor Injection is that, once number of dependency crossed a threshold e.g. 5 or 6 its handy manageable to passing dependency via constructor. Setter Injection is preferred choice when number of dependency to be injected is lot more than normal, if some of those arguments is optional than using [Builder design pattern](http://javarevisited.blogspot.in/2012/06/builder-design-pattern-in-java-example.html) is also a good option.

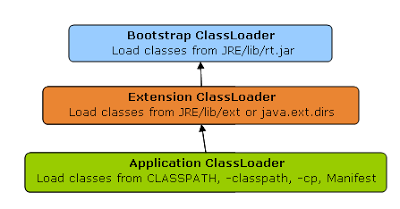
In Summary both Setter Injection and Constructor Injection has there own advantage and disadvantage. Good thing about Spring is that it doesn't restrict you to use either Setter Injection or Constructor Injection and you are free to use both of them in one Spring configuration file. Use Setter injection when number of dependency is more or you need readability. Use Constructor Injection when Object must be created with all of its dependency.

# [How ClassLoader Works in Java](http://javarevisited.blogspot.com/2012/12/how-classloader-works-in-java.html)

Java class loaders are used to load classes at runtime. ClassLoader in Java works on three principle: delegation, visibility and uniqueness. Delegation principle forward request of class loading to parent class loader and only loads the class, if parent is not able to find or load class. Visibility principle allows child class loader to see all the classes loaded by parent ClassLoader, but parent class loader can not see classes loaded by child. Uniqueness principle allows to load a class exactly once, which is basically achieved by delegation and ensures that child ClassLoader doesn't reload the class already loaded by parent. Correct understanding of class loader is must to resolve issues like[NoClassDefFoundError in Java](http://javarevisited.blogspot.sg/2011/06/noclassdeffounderror-exception-in.html) and [java.lang.ClassNotFoundException](http://javarevisited.blogspot.sg/2011/08/classnotfoundexception-in-java-example.html), which are related to class loading. ClassLoader is also an important topic in advanced Java Interviews, where good knowledge of working of Java ClassLoader and [How classpath works in Java](http://javarevisited.blogspot.ca/2011/01/how-classpath-work-in-java.html) is expected from Java programmer. I have always seen questions like, **Can one class be loaded by two different ClassLoader in Java** on various [Java Interviews](http://javarevisited.blogspot.sg/2011/04/top-20-core-java-interview-questions.html).  In this Java programming tutorial, we will learn what is ClassLoader in Java, How ClassLoader works in Java and some specifics about Java ClassLoader.

**What is ClassLoader in Java**

ClassLoader in Java is a class which is used to load [class files in Java](http://javarevisited.blogspot.ca/2012/05/10-points-about-class-file-in-java.html). Java code is compiled into class file by [javac](http://javarevisited.blogspot.sg/2012/12/javac-is-not-recognized-as-internal-or-external-command.html)compiler and [JVM](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html)executes Java program, by executing byte codes written in class file. ClassLoader is responsible for loading class files from file system, network or any other source. There are three default class loader used in Java,**Bootstrap** , **Extension** and **System or Application class loader**. Every class loader has a predefined location, from where they loads class files. Bootstrap ClassLoader is responsible for loading standard JDK class files from rt.jar and it is parent of all class loaders in Java. Bootstrap class loader don't have any parents, if you callString.class.getClassLoader() it will return null and any code based on that may throw [NullPointerException in Java](http://javarevisited.blogspot.com/2012/06/common-cause-of-javalangnullpointerexce.html). Bootstrap class loader is also known as**Primordial ClassLoader** in Java.  Extension ClassLoader delegates class loading request to its parent, Bootstrap and if unsuccessful, loads class form jre/lib/ext directory or any other directory pointed by java.ext.dirs system property. Extension ClassLoader in JVM is implemented by  sun.misc.Launcher$ExtClassLoader. Third default class loader used by JVM to load Java classes is called System or Application class loader and it is responsible for loading application specific classes from [CLASSPATH](http://javarevisited.blogspot.sg/2011/01/how-classpath-work-in-java.html) environmentvariable, -classpath or -cp command line option, Class-Path attribute of Manifest file inside JAR. Application class loader is a child of Extension ClassLoader and its implemented by sun.misc.Launcher$AppClassLoader class. Also, except Bootstrap class loader, which is implemented in native language mostly in C,  all  Java class loaders are implemented using java.lang.ClassLoader.

[](http://2.bp.blogspot.com/-HCTsr-j_ojw/USTOh1f8JwI/AAAAAAAAAjg/YegPspR5K48/s1600/java_classloader_hierarchy.PNG)

In short here is the location from which Bootstrap, Extension and Application ClassLoader load Class files.

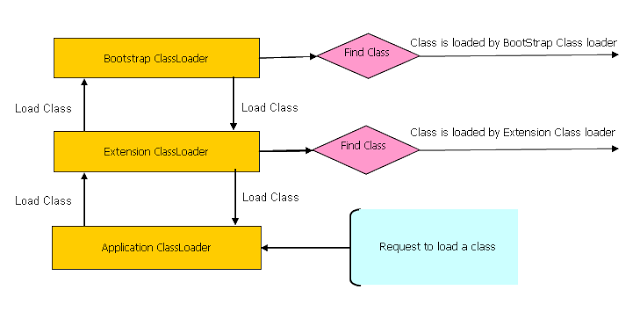
1) Bootstrap ClassLoader - JRE/lib/rt.jar

2) Extension ClassLoader - JRE/lib/ext or any directory denoted by java.ext.dirs

3) Application ClassLoader - CLASSPATH environment variable, -classpath or -cp option, Class-Path attribute of Manifest inside [JAR file](http://javarevisited.blogspot.sg/2012/03/how-to-create-and-execute-jar-file-in.html).

## **How ClassLoader works in Java**

As I explained earlier Java ClassLoader works in three principles : delegation, visibility and uniqueness. In this section we will see those rules in detail and understand working of Java ClassLoader with example. By the way here is a diagram which explains How ClassLoader load class in Java using delegation.

[](http://1.bp.blogspot.com/-0gOWex7Pb2E/USTOh2K7zpI/AAAAAAAAAjc/_viQADzxrsk/s1600/Java+classloader+working.PNG)

**Delegation principles**

As discussed on [when a class is loaded and initialized in Java](http://javarevisited.blogspot.sg/2012/07/when-class-loading-initialization-java-example.html), a class is loaded in Java, when its needed. Suppose you have an application specific class called Abc.class, first request of loading this class will come to Application ClassLoader which will delegate to its parent Extension ClassLoader which further delegates to Primordial or Bootstrap class loader. Primordial will look for that class inrt.jar and since that class is not there, request comes to Extension class loader which looks on jre/lib/ext directory and tries to locate this class there, if class is found there than Extension class loader will load that class and Application class loader will never load that class but if its not loaded by extension class-loader than Application class loader loads it from [Classpath in Java](http://java67.blogspot.sg/2012/08/what-is-path-and-classpath-in-java-difference.html). Remember Classpathis used to load class files while [PATH](http://javarevisited.blogspot.ca/2011/10/how-to-set-path-for-java-unix-linux-and.html) is used to locate executable like javac or java command.

**Visibility Principle**

According to visibility principle, Child ClassLoader can see class loaded by Parent ClassLoader but vice-versa is not true. Which mean if class Abc is loaded by Application class loader than trying to load class ABC explicitly using extension ClassLoader will throw either [java.lang.ClassNotFoundException](http://javarevisited.blogspot.ca/2011/08/classnotfoundexception-in-java-example.html). as shown in below Example

**package** test;  
  
**import** java.util.logging.Level;  
**import** java.util.logging.Logger;  
  
/\*\*  
 \* Java program to demonstrate How ClassLoader works in Java,

 \* in particular about visibility principle of ClassLoader.

 \*  
 \* @author Javin Paul  
 \*/  
  
**public** **class** ClassLoaderTest {  
    
    **public** **static** **void** main(**String** args[]) {  
        **try** {            
            *//printing ClassLoader of this class*  
            **System**.out.println("ClassLoaderTest.getClass().getClassLoader() : "  
                                 + ClassLoaderTest.**class**.getClassLoader());  
  
            
            *//trying to explicitly load this class again using Extension class loader*  
            **Class**.forName("test.ClassLoaderTest", **true**   
                            ,  ClassLoaderTest.**class**.getClassLoader().getParent());  
        } **catch** (**ClassNotFoundException** ex) {  
            **Logger**.getLogger(ClassLoaderTest.**class**.getName()).log(**Level**.SEVERE, **null**, ex);  
        }  
    }  
  
}  
  
**Output:**  
ClassLoaderTest.getClass().getClassLoader() : sun.misc.Launcher$AppClassLoader@601bb1  
16/08/2012 2:43:48 AM test.ClassLoaderTest main  
SEVERE: **null**  
java.lang.**ClassNotFoundException**: test.ClassLoaderTest  
        at java.net.**URLClassLoader**$1.run(**URLClassLoader**.java:202)  
        at java.security.**AccessController**.doPrivileged(Native **Method**)  
        at java.net.**URLClassLoader**.findClass(**URLClassLoader**.java:190)  
        at sun.misc.Launcher$ExtClassLoader.findClass(Launcher.java:229)  
        at java.lang.**ClassLoader**.loadClass(**ClassLoader**.java:306)  
        at java.lang.**ClassLoader**.loadClass(**ClassLoader**.java:247)  
        at java.lang.**Class**.forName0(Native **Method**)  
        at java.lang.**Class**.forName(**Class**.java:247)  
        at test.ClassLoaderTest.main(ClassLoaderTest.java:29)

**Uniqueness Principle**

According to this principle a class loaded by Parent should not be loaded by Child ClassLoader again. Though its completely possible to write class loader which violates Delegation and Uniqueness principles and loads class by itself, its not something which is beneficial. You should follow all  class loader principle while writing your own ClassLoader.

## **How to load class explicitly in Java**

Java provides API to explicitly load a class by Class.forName(classname) and Class.forName(classname, initialized, classloader), remember JDBC code which is used to load JDBC drives we have seen in [Java program to Connect Oracle database](http://javarevisited.blogspot.ca/2012/04/java-program-to-connect-oracle-database.html). As shown in above example you can pass name of ClassLoader which should be used to load that particular class along with binary name of class. Class is loaded by calling loadClass() method of java.lang.ClassLoader class which calls findClass() method to locate bytecodes for corresponding class. In this example Extension ClassLoader uses java.net.URLClassLoader which search for class files and resources in [JAR](http://javarevisited.blogspot.ca/2012/10/5-ways-to-add-multiple-jar-to-classpath-java.html) and directories. any search path which is ended using "/" is considered directory. If findClass() does not found the class than it throws [java.lang.ClassNotFoundException](http://javarevisited.blogspot.de/2012/03/jdbc-javalangclassnotfoundexception.html) and if it finds it calls defineClass() to convert bytecodes into a .class instance which is returned to the caller.

**Where to use ClassLoader in Java**

ClassLoader in Java is a powerful concept and used at many places. One of the *popular example of ClassLoader* is AppletClassLoader which is used to load class by Applet, since Applets are mostly loaded from internet rather than local file system, By using separate ClassLoader you can also loads same class from multiple sources and they will be treated as different class in [JVM](http://javarevisited.blogspot.ca/2011/12/jre-jvm-jdk-jit-in-java-programming.html). J2EE uses multiple class loaders to load class from different location like classes from WAR file will be loaded by Web-app ClassLoader while classes bundled in EJB-JAR is loaded by another class loader. Some web server also supports hot deploy functionality which is implemented using ClassLoader. You can also use ClassLoader to load classes from database or any other persistent store.

That's all about **What is ClassLoader in Java** and **How ClassLoader works in Java**. We have seen delegation, visibility and uniqueness principles which is quite important to debug or troubleshoot any ClassLoader related issues in Java. In summary knowledge of How ClassLoader works in Java is must for any Java developer or architect to design Java application and packaging.

# [BlockingQueue in Java – ArrayBlockingQueue vs LinkedBlockingQueue Example program Tutorial](http://javarevisited.blogspot.com/2012/12/blocking-queue-in-java-example-ArrayBlockingQueue-LinkedBlockingQueue.html)

BlockingQueue in Java is added in Java 1.5 along with various other concurrent Utility classes like [ConcurrentHashMap](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html), [Counting Semaphore](http://javarevisited.blogspot.sg/2012/05/counting-semaphore-example-in-java-5.html), [CopyOnWriteArrrayList](http://java67.blogspot.sg/2012/09/what-is-copyonwritearraylist-in-java-example-vs-arraylist.html) etc. BlockingQueue is a unique collection type which not only store elements but also supports flow control by introducing blocking if either BlockingQueue is full or empty. take()method of BlockingQueue will block if Queue is empty and put() method of BlockingQueue will block if Queue is full. This property makes BlockingQueue an ideal choice for implementing [Producer consumer design pattern](http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html) where one thread insert element into BlockingQueue and other thread consumes it. In this Java tutorial we will learn about What is BlockingQueue in Java, How to use BlockingQueue, ArrayBlockingQueue and LinkedBlockingQueue and some important properties of it.

## **Important properties of BlockingQueue in Java**

Before using any new Collection class e.g. BlockingQueue, I always read there API documentation to know more about it. There are always some important points which is worth remembering and avoids potential programming errors while using new Collection class. Following list of points about BlockingQueue in Java will help to learn and understand more about it.

1) BlockingQueue in Java doesn't allow null elements, various implementation of BlockingQueue like ArrayBlockingQueue, LinkedBlockingQueue throws [NullPointerException](http://javarevisited.blogspot.sg/2012/06/common-cause-of-javalangnullpointerexce.html) when you try to add null on queue.

**BlockingQueue**<**String**> bQueue = **new** **ArrayBlockingQueue**<**String**>(10);  
*//bQueue.put(null); //NullPointerException - BlockingQueue in Java doesn't allow null*  
        
bQueue = **new** **LinkedBlockingQueue**<**String**>();  
bQueue.put(**null**);  
  
**Exception** in thread "main" java.lang.**NullPointerException**  
        at java.util.concurrent.**LinkedBlockingQueue**.put(**LinkedBlockingQueue**.java:288)

2) BlockingQueue can be bounded or unbounded. A bounded BlockingQueue is one which is initialized with initial capacity and call to put() will be blocked if BlockingQueue is full and size is equal to capacity. This bounding nature makes it ideal to use a shared queue between multiple threads like in most common [Producer consumer solutions in Java](http://javarevisited.blogspot.de/2012/02/producer-consumer-design-pattern-with.html). An unbounded Queue is one which is initialized without capacity, actually by default it initialized with Integer.MAX\_VALUE. most common example of BlockingQueue uses **bounded BlockingQueue** as shown in below example.

**BlockingQueue**<**String**> bQueue = **new** **ArrayBlockingQueue**<**String**>(2);  
bQueue.put("Java");  
**System**.out.println("Item 1 inserted into BlockingQueue");  
bQueue.put("JDK");  
**System**.out.println("Item 2 is inserted on BlockingQueue");  
bQueue.put("J2SE");  
**System**.out.println("Done");  
  
Output:  
Item 1 inserted into **BlockingQueue**  
Item 2 is inserted on **BlockingQueue**

This code will only insert Java and JDK into BlockingQueue and then it will block while inserting 3rd element J2SE because size of BlockingQueue is 2 here.

3)BlockingQueue implementations like ArrayBlockingQueue, LinkedBlockingQueue and PriorityBlockingQueue are [thread-safe](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html). All queuing method uses concurrency control and internal locks to perform operation atomically. Since BlockingQueue also extend Collection, bulk Collection operations like addAll(), containsAll() are not performed atomically until any BlockingQueueimplementation specifically supports it. So call to addAll() may fail after inserting couple of elements.

4) Common methods of BlockingQueue is are put() and take() which are [blocking methods in Java](http://javarevisited.blogspot.sg/2012/02/what-is-blocking-methods-in-java-and.html) and used to insert and retrive elements from BlockingQueue in Java. put() will block if BlockingQueue is full and take() will block if BlockingQueue is empty, call to take() removes element from head of Queue as shown in following example:

**BlockingQueue**<**String**> bQueue = **new** **ArrayBlockingQueue**<**String**>(2);  
bQueue.put("Java"); *//insert object into BlockingQueue*  
**System**.out.println("BlockingQueue after put: " + bQueue);  
bQueue.take(); *//retrieve object from BlockingQueue in Java*  
**System**.out.println("BlockingQueue after take: " + bQueue);  
  
Output:  
**BlockingQueue** after put: [Java]  
**BlockingQueue** after take: []

5) BlockingQueue interface extends Collection, Queue and Iterable interface which provides it all Collection and Queue related methods like poll(), and peak(), unlike take(), peek() method returns head of the queue without removing it, poll() also retrieves and removes elements from head but can wait till specified time if Queue is empty.

**BlockingQueue**<**String**> linkedBQueue = **new** **LinkedBlockingQueue**<**String**>(2);  
linkedBQueue.put("Java"); *//puts object into BlockingQueue*  
**System**.out.println("size of BlockingQueue before peek : " + linkedBQueue.size());         
**System**.out.println("example of peek() in BlockingQueue: " + linkedBQueue.peek());  
**System**.out.println("size of BlockingQueue after peek : " + linkedBQueue.size());  
**System**.out.println("calling poll() on BlockingQueue: " + linkedBQueue.poll());  
**System**.out.println("size of BlockingQueue after poll : " + linkedBQueue.size());  
  
Output:  
size of **BlockingQueue** before peek : 1  
example of peek() in **BlockingQueue**: Java  
size of **BlockingQueue** after peek : 1  
calling poll() on **BlockingQueue**: Java  
size of **BlockingQueue** after poll : 0

6)Other important methods from BlockingQueue in Java is remainingCapacity() and offer(), former returns number remaining space in BlockingQueue, which can be filled without blocking while later insert object into queue if possible and return true if success and false if fail unlike add() method which [throws](http://javarevisited.blogspot.sg/2012/02/difference-between-throw-and-throws-in.html) IllegalStateException if it fails to insert object into BlockingQueue. Use offer() overadd() wherever possible.

**Usage of BlockingQueue in Java**

There can be many creative usage of BlockingQueue in Java given its flow control ability. Two of the most common ways I see programmer uses BlockingQueue is to implement Producer Consumer design pattern and implementing Bounded buffer in Java. It surprisingly made coding and inter thread communication over a shared object very easy.

**ArrayBlockingQueue and LinkedBlockingQueue in Java**

ArrayBlockingQueue and LinkedBlockingQueue are common implementation of BlockingQueue<E> interface. ArrayBlockingQueue is backed by array  and Queue impose orders as FIFO. head of the queue is the oldest element in terms of time and tail of the queue is youngest element. ArrayBlockingQueue is also fixed size bounded buffer on the other hand LinkedBlockingQueue is an optionally bounded queue built on top of Linked nodes. In terms of throughput LinkedBlockingQueue provides higher throughput than ArrayBlockingQueue in Java.