## Differences Between wait() And sleep() Methods In Java :

* Both wait() and sleep() methods are used to pause the execution of current thread for some period of time. Whenever a thread calls wait() method, it goes into **WAITING** state after releasing the lock it holds. Whenever a thread calls sleep() method, it goes into **TIMED\_WAITING** state without releasing the lock it holds.
* A thread which is in **WAITING** state (state after calling wait() method) can be woken up by other threads by calling **notify()** or **notifyAll()** methods on the same lock. But, a thread which is in **TIMED\_WAITING** state (state after calling sleep() method) can not be woken up. If any threads interrupt sleeping thread, InterruptedException will be raised.
* wait() method along with notify() and notifyAll() are used for **inter thread communication** where as sleep() method is used to **pause the execution of current thread** for specific period of time.
* wait() method is an instance method of **java.lang.Object** class. That means, this method is available in all objects you create in java. Where as sleep() method is a static method of **java.lang.Thread** class. That means, it is available only in threads.
* wait() method is called on **objects**. Whenever it is called by a thread on a particular object, thread releases the lock of that object and waits until other threads call either notify() or notifyAll() methods on the same object. Where as sleep() method is called on **threads**.
* Whenever sleep() method is called, only **current thread** is going for sleep. For example, if **main thread** calls sleep() method on a **thread t**, i.e **t.sleep()**, main thread itself is going to sleep not thread t.
* To call wait() method, calling thread must hold the lock of the object on which it is calling wait() method. That means, wait() method must be called **within the** **synchronized block**. Where as to call sleep() method, thread need not to hold the object lock. That means, sleep() method can be called **outside the synchronized block** also.

[**What is the benefit of ThreadGroup in java over creating separate threads?**](http://stackoverflow.com/questions/1649133/what-is-the-benefit-of-threadgroup-in-java-over-creating-separate-threads)

Using ThreadGroup can be a useful diagnostic technique in big application servers with thousands of threads. If your threads are logically grouped together, then when you get a stack trace you can see which group the offending thread was part of (e.g. "Tomcat threads", "MDB threads", "thread pool X", etc), which can be a big help in tracking down and fixing the problem.

Each thread in Java is assigned to a thread group upon the thread's creation. These groups are implemented by the java.lang.ThreadGroup class. When the thread group name is not specified explicitly, the main default group is assigned by the Java Virtual Machine (JVM)

Even though thread groups are useful for keeping threads organized, programmers seldom benefit from their use because many of the methods of the ThreadGroup class are deprecated

final class HandleRequest implements Runnable {

  public void run() {

    // Do something

  }

}

public final class NetworkHandler implements Runnable {

  private static ThreadGroup tg = new ThreadGroup("Chief");

  @Override public void run() {

    new Thread(tg, new HandleRequest(), "thread1").start();

    new Thread(tg, new HandleRequest(), "thread2").start();

    new Thread(tg, new HandleRequest(), "thread3").start();

  }

  public static void printActiveCount(int point) {

    System.out.println("Active Threads in Thread Group " + tg.getName() +

        " at point(" + point + "):" + " " + tg.activeCount());

  }

  public static void printEnumeratedThreads(Thread[] ta, int len) {

    System.out.println("Enumerating all threads...");

    for (int i = 0; i < len; i++) {

      System.out.println("Thread " + i + " = " + ta[i].getName());

    }

  }

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

    // Start thread controller

    Thread thread = new Thread(tg, new NetworkHandler(), "controller");

    thread.start();

    // Gets the active count (insecure)

    Thread[] ta = new Thread[tg.activeCount()];

    printActiveCount(1); // P1

    // Delay to demonstrate TOCTOU condition (race window)

    Thread.sleep(1000);

    // P2: the thread count changes as new threads are initiated

    printActiveCount(2);

    // Incorrectly uses the (now stale) thread count obtained at P1

    int n = tg.enumerate(ta);

    // Silently ignores newly initiated threads

    printEnumeratedThreads(ta, n);

                                   // (between P1 and P2)

    // This code destroys the thread group if it does

    // not have any live threads

    for (Thread thr : ta) {

      thr.interrupt();

      while(thr.isAlive());

    }

    tg.destroy();

  }

}

Logs: (KronosSiebelDataThread) --- is thread name

2013-02-18 09:12:19,306 [-KronosSiebelDataThread-0] com.ebay.kronos.runtime.receiver.KronosSiebelRequestCompiler The year used in the query to get Siebel Data Event is : 2013-02-18

2013-02-18 09:12:19,306 [-KronosSiebelDataThread-0] com.ebay.kronos.runtime.receiver.KronosSiebelRequestCompiler The hour used in the query to get Siebel Data Event is : 16:12

2013-02-18 09:12:19,306 [-KronosSiebelDataThread-0] com.ebay.kronos.runtime.receiver.KronosSiebelRequestCompiler The status string in the query to get Siebel Data Event is : S

# [Top 15 Java Multithreading, Concurrency Interview Questions Answers asked in Investment banks](http://javarevisited.blogspot.sg/2011/07/java-multi-threading-interview.html)

**Thread interview questions Java**

Multi-threading and concurrency questions are essential part of any Java interview. If you are going for any Java interview on any Investment bank for equities front office position expect lots of **muti-threading interview questions** on your way. Multi-threading and concurrency is a favorite topics on Investment banking specially on electronic trading development and they grill candidate on many [confusing java thread interview questions](http://javarevisited.blogspot.sg/2011/04/top-20-core-java-interview-questions.html). They just want to ensure that the guy has solid knowledge of multi-threading and concurrent programming in Java  because most of them are in business of performance. High volume and low latency Electronic trading System which is used for Direct to Market (DMA) trading is usually concurrent in nature. These are my favorite thread interview questions on Java  asked on different on different time. I am not providing answer of these *thread interview questions* but I will give you hint whenever possible, some time hint is enough to answer. I will update the post further with detailed answers just like I did for [10 Singleton interview questions in Java](http://javarevisited.blogspot.sg/2011/03/10-interview-questions-on-singleton.html) recently.  With introduction of concurrency package in Java 5 questions on concurrent utility and concurrent collections are on rise as well. [ThreadLocal](http://javarevisited.blogspot.sg/2012/05/how-to-use-threadlocal-in-java-benefits.html), [BlockingQueue](http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html), [Counting Semaphore](http://javarevisited.blogspot.sg/2012/05/counting-semaphore-example-in-java-5.html) and [ConcurrentHashMap](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html) are popular among those.

## 15 Java Thread Interview Questions and answers

**1) You have thread T1, T2 and T3, how will you ensure that thread T2 run after T1 and thread T3 run after T2?**

This thread interview questions is mostly asked in first round or phone screening round of interview and purpose of this multi-threading question is to check whether candidate is familiar with concept of *"join"* method or not. Answer of this multi-threading questions is simple it can be achieved by using **join** method of Thread class.

**2) What is the advantage of new Lock interface over synchronized block in Java? You need to implement a high performance cache which allows multiple reader but single writer to keep the integrity how will you implement it?**

The major advantage of lock interfaces on multi-threaded and concurrent programming is they provide two separate lock for reading and writing which enables you to write high performance data structure like [ConcurrentHashMap](http://javarevisited.posterous.com/difference-between-concurrenthashmap-and-coll) and [conditional blocking](http://javarevisited.blogspot.sg/2012/02/what-is-blocking-methods-in-java-and.html). This java threads interview question is getting increasingly popular and more and more follow-up questions come based upon answer of interviewee. I would strongly suggest reading **Locks** before appearing for any *java multi-threading interview* because now days Its  heavily used to build cache for electronic trading system on client and exchange connectivity space.

**3) What are differences between wait and sleep method in java?**

Another frequently asked thread interview question in Java mostly appear in phone interview. Only major difference is wait release the lock or monitor while sleep doesn't release any lock or monitor while waiting. Wait is used for inter-thread communication while sleep is used to introduce pause on execution. See my post [wait vs sleep in Java](http://javarevisited.blogspot.sg/2011/12/difference-between-wait-sleep-yield.html) for more differences

**4) Write code to implement blocking queue in Java?**

This is relatively tough java multi-threading interview question which servers many purpose, it checks whether candidate can actually write Java code using [thread](http://javarevisited.blogspot.sg/2011/02/how-to-implement-thread-in-java.html) or not, it sees how good candidate is on understanding concurrent scenarios and you can ask lot of follow-up question based upon his code. If he uses [wait() and notify() method](http://javarevisited.blogspot.sg/2011/05/wait-notify-and-notifyall-in-java.html) to implement blocking queue, Once interviewee successfully writes it  you can ask him to write it again using new java 5 concurrent classes etc.

**5) Write code to solve the Produce consumer problem in Java?**

Similar to above questions on thread but more classic in nature, some time interviewer ask follow up questions How do you solve producer consumer problem in Java, well it can be solved in multiple way, I have shared one way to solve [producer consumer problem using BlockingQueue in Java](http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html) , so be prepare for surprises. Some time they even ask to implement solution of dining philosopher problem as well.

**6) Write a program which will result in deadlock? How will you fix deadlock in Java?**

This is my favorite java thread interview question because even though deadlock is quite common while writing multi-threaded concurrent program many candidates not able to write deadlock free code and they simply struggle. Just ask them you have n resources and n thread and to complete an operation you require all resources. Here n can be replace with 2 for simplest case and higher number to make question more intimidating. see  [How to avoid deadlock in java](http://javarevisited.blogspot.com/2010/10/what-is-deadlock-in-java-how-to-fix-it.html)  for more information on deadlock in Java.

**7) What is atomic operation? What are atomic operations in Java?**

Simple java thread interview questions, another follow-up is do you need to synchronized an atomic operation? :) You can read more about [java synchronization](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html) here.

**8) What is volatile keyword in Java? How to use it? How is it different from synchronized method in Java?**

Thread questions based on [volatile keyword in Java](http://javarevisited.blogspot.com/2011/06/volatile-keyword-java-example-tutorial.html) has become more popular after changes made on it on Java 5 and Java memory model. It’s good to prepare well about how volatile variables ensures visibility, ordering and consistency in concurrent environment.

**9) What is race condition? How will you find and solve race condition?**

Another multi-threading question in Java which appear mostly on senior level interviews. Most interviewer grill on recent race condition you have faced and how did you solve it and some time they will write sample code and ask you detect race condition. See my post on [Race condition in Java](http://javarevisited.blogspot.sg/2012/02/what-is-race-condition-in.html) for more information. In my opinion this is one of the best java thread interview question and can really test the candidate's experience on solving race condition or writing code which is free of data race or any other race condition. Best book to get mastery of this topic is "Concurrency practices in Java'".

**10) How will you take thread dump in Java? How will you analyze Thread dump?**

In UNIX you can use **kill -3** and then thread dump will print on log on windows you can use **"CTRL+Break".** Rather simple and focus thread interview question but can get tricky if he ask how you analyze it. Thread dump can be useful to analyze deadlock situations as well.

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

Another classic java multi-threading interview question This was my original doubt when I started programming in thread. Now days mostly asked in phone interview or first round of interview at mid and junior level java interviews. Answer to this question is that, when you call start() method it creates new Thread and execute code declared in run() while directly calling run() method doesn’t create any new thread and execute code on same calling thread. Read my post [Difference between start and run method in Thread](http://javarevisited.blogspot.sg/2012/03/difference-between-start-and-run-method.html) for more details.

**12) How will you awake a blocked thread in java?**

This is tricky question on threading, blocking can result on many ways, if thread is blocked on IO then I don't think there is a way to interrupt the thread, let me know if there is any, on the other hand if thread is blocked due to result of calling wait(), sleep() or join() method you can interrupt the thread and it will awake by throwing InterruptedException. See my post [How to deal with blocking methods in Java](http://javarevisited.blogspot.sg/2012/02/what-is-blocking-methods-in-java-and.html) for more information on handling blocked thread.

**13) What is difference between CyclicBarriar and CountdownLatch in Java ?**

New java thread interview questions mostly to check familiarity with JDK 5 concurrent packages. One difference is that you can reuse CyclicBarrier once barrier is broken but you can not reuse ContdownLatch.

**14) What is immutable object? How does it help on writing concurrent application?**

Another classic interview questions on multi-threading, not directly related to thread but indirectly helps a lot. This java interview question can become more tricky if ask you to write an immutable class or ask you [Why String is immutable in Java](http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html) as follow-up.

**15) What are some common problems you have faced in multi-threading environment? How did you resolve it?**

Memory-interference, race conditions, [deadlock](http://javarevisited.blogspot.sg/2010/10/what-is-deadlock-in-java-how-to-fix-it.html), live lock and starvation are example of some problems comes in multi-threading and concurrent programming. There is no end of problem if you get it wrong and they will be hard to detect and debug. This is mostly experienced based interview question on java thread instead of fact based.

These were my favorite Java thread interview questions and mostly asked on Investment banks. This list is by no means complete so please contribute some of interesting java thread questions you have faced during interview. Purpose of this article is to collect and share great interview questions on multi-threading concept which not only helps on interview but opens door for learning new threading concept.

**Update:**

One of Javarevisited reader, Hemant has contributed some more thread interview questions in Java, though he hasn’t provide answer and left that job for me, I will certainly do when time allows, just like I have recently updated 10 Singleton interview question in Java with answers. If you guys know answers of this java concurrency questions than please post as comment:

Here is his comment “Good questions on multi-threading though you may need to prepare more in order to clear any multi-threading interview, you need to be familiar with concept of [immutability](http://avdheshsemwal.blogspot.sg/2012/02/why-string-is-immutable-or-final-in.html), [thread-safety](http://javarevisited.blogspot.sg/2011/07/java-multi-threading-interview.html), [race condition](http://javarevisited.blogspot.sg/2012/02/what-is-race-condition-in.html) and many more. 10 or 15 question is good for quick recap but you at-least need to prepare more than 50 questions on threading and concurrency to perform better on Java interview. You can find some interesting thread question below which is no doubt highly popular –

1)  Difference between green thread and native thread in Java?

2)  Difference between thread and process?

3)  What is context switching in multi-threading?

4)  Difference between deadlock and livelock, deadlock and starvation?

5)  What thread-scheduling algorithm is used in Java?

6)  What is thread-scheduler in Java?

7)  How do you handle un-handled exception in thread?

8)  What is thread-group, why its advised not to use thread-group in Java?

9)  Why Executor framework is better than creating and managing thread by application ?

10) Difference between Executor and Executors in Java?

10) How to find which thread is taking maximum cpu in windows and Linux server?

Read more: <http://javarevisited.blogspot.com/2011/07/java-multi-threading-interview.html#ixzz2gmFXSvlF>

# [Why wait, notify and notifyAll is defined in Object Class and not on Thread class in Java](http://javarevisited.blogspot.sg/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.

[Why Wait notify method is declared in Object Class and not in Thread in Java](http://javarevisited.blogspot.com/2012/01/google-interview-questions-answers-top.html)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*.

Update:  
@Lipido has made an insightful comment , which is worth adding here. read his comment for full text

"Java is based on Hoare's monitors idea (http://en.wikipedia.org/wiki/Monitor\_%28synchronization%29). In Java all object has a monitor. Threads waits on monitors so, to perform a wait, we need 2 parameters:  
- a Thread  
- a monitor (any object)  
  
In the Java design, the thread can not be specified, it is always the current thread running the code. However, we can specify the monitor (which is the object we call wait on). This is a good design, because if we could make any other thread to wait on a desired monitor, this would lead to an "intrusion", posing difficulties on designing/programming concurrent programs. Remember that in Java all operations that are intrusive in another thread's execution are deprecated (e.g. stop())."

Read more: <http://javarevisited.blogspot.com/2012/02/why-wait-notify-and-notifyall-is.html#ixzz2gmFgDX7p>

# [Difference between Thread vs Runnable interface in Java](http://javarevisited.blogspot.sg/2012/01/difference-thread-vs-runnable-interface.html)

**Thread vs Runnable in Java** is always been a confusing decision for beginners  in java. [Thread in Java](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) seems easy in comparison of Runnable because you just deal with one class **java.lang.Thread** while in case of using Runnable to implement Thread you need to deal with both Thread and Runnable two classes. though decision of using Runnable or Thread should be taken considering **differences between Runnable and Thread** and pros and cons of both approaches. This is also a very [popular thread interview questions](http://javarevisited.blogspot.com/2011/07/java-multi-threading-interview.html) and most of interviewer are really interested to know what is your point of view while choosing *Thread vs Runnable or opposite*. In this java article we will try to point out some *differences between Thread and Runnable in Java* which will help you to take an informed decision.

## Difference between Thread and Runnable interface in Java

### Thread vs Runnable in Java

[Difference between Thread vs Runnable in Java](http://javarevisited.blogspot.com/2011/09/spring-interview-questions-answers-j2ee.html)Here are some of my thoughts on whether I should use **Thread or Runnable** for implementing task in Java, though you have another choice as "Callable" for implementing thread which we will discuss later.

1) [Java doesn't support multiple inheritance](http://javarevisited.blogspot.com/2011/07/why-multiple-inheritances-are-not.html), which means you can only extend one class in Java so once you extended Thread class you lost your chance and can not extend or inherit another [class in Java](http://javarevisited.blogspot.com/2011/10/class-in-java-programming-general.html).

2) In Object oriented programming extending a class generally means adding new functionality, modifying or improving behaviors. If we are not making any modification on Thread than use Runnable interface instead.

3) **Runnable** interface represent a Task which can be executed by either plain Thread or Executors or any other means. so logical separation of Task as Runnable than Thread is good design decision.

4) Separating task as **Runnable** means we can reuse the task and also has liberty to execute it from different means. since you can not restart a Thread once it completes. again **Runnable vs Thread** for task, Runnable is winner.

5) Java designer recognizes this and that's why Executors accept Runnable as Task and they have worker thread which executes those task.

6) Inheriting all Thread methods are additional overhead just for representing a Task which can can be done easily with Runnable.

These were some of notable **difference between Thread and Runnable in Java**, if you know any other differences on Thread vs Runnable than please share it via comments. I personally use Runnable over Thread for this scenario and recommends to use Runnable or Callable interface based on your requirement.

Read more: <http://javarevisited.blogspot.com/2012/01/difference-thread-vs-runnable-interface.html#ixzz2gmFiOi6j>

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

### Why wait (), notify () and notifyAll () must be called from synchronized block or method in Java

[Why wait(), notify() and notifyAll() method must be called from synchronized block or method in Java](http://2.bp.blogspot.com/-wrzDeQGAe1I/TWu8pLuLr4I/AAAAAAAAADE/V017G-6Q61w/s1600/java_logo_50_50.jpg)Most of Java developer knows that **wait() ,notify() and notifyAll() method of object class** must have to be called inside **synchronized** method or synchronized block in Java but how many times we thought why ? Recently this questions was asked to in Java interview to one of my friend, he pondered for a moment and replied that if we don't call wait () or notify () method from synchronized context we will receive IllegalMonitorStateException in java. He was right in terms of behavior of language but as per him interviewer was not completely satisfied with the answer and wanted to explain more about it. After the interview he discussed the same questions with me and I thought he might have told about race condition between **wait () and notify () in Java** that could exists if we don't call them inside synchronized method or block. Let’s see how it could happen:

We use **wait () and notify () or notifyAll () method mostly for inter-thread communication**. One thread is waiting after checking a condition e.g. In Producer Consumer example Producer Thread is waiting if buffer is full and Consumer thread notify Producer thread after he creates a space in buffer by consuming an element. calling notify() or notifyAll() issues a notification to a single or multiple thread that a condition has changed and once notification thread leaves synchronized block , all the threads which are waiting fight for object lock on which they are waiting and lucky **thread returns from wait() method after reacquiring the lock** and proceed further. Let’s divide this whole operation in steps to see a possibility of *race condition between wait () and notify () method in Java*, we will use **Produce Consumer thread example** to understand the scenario better:

   1. The Producer thread tests the condition (buffer is full or not) and confirms that it must wait (after finding buffer is full).

   2. The Consumer thread sets the condition after consuming an element from buffer.

   3. The Consumer thread calls the notify () method; this goes unheard since the Producer thread is not yet waiting.

   4. The Producer thread calls the wait () method and goes into waiting state.

So due to race condition here we potential lost a notification and if we use buffer or just one element Produce thread will be waiting forever and your program will hang.

Now let's think how does this potential race condition get resolved? This race condition is resolved by using [synchronized keyword and locking provided by java](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html). In order to call the **wait (), notify () or notifyAll () methods in Java**, we must have obtained the lock for the object on which we're calling the method. Since the **wait () method in Java also releases the lock prior to waiting and reacquires the lock prior to returning from the wait () method**, we must use this lock to ensure that checking the condition (buffer is full or not) and setting the condition (taking element from buffer) is atomic which can be achieved by using synchronized method or block in Java.

[](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html)

I am not sure if this is what interviewer was actually expecting but this what I thought would at least make sense, please correct me If I wrong and let us know if there is any other convincing reason of calling wait(), notify() or notifyAll method in Java.

Just to summarize we call **wait** (), **notify** () or **notifyAll** method in Java from [synchronized method or synchronized block in Java](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html) to avoid:

1) **IllegalMonitorStateException in Java** which will occur if we don't call wait (), notify () or notifyAll () method from synchronized context.

2) **Any potential race condition between wait and notify method in Java**.

Read more: <http://javarevisited.blogspot.com/2011/05/wait-notify-and-notifyall-in-java.html#ixzz2gmFlCnWE>

# [How to Stop Thread in Java Code Example](http://javarevisited.blogspot.sg/2011/10/how-to-stop-thread-java-example.html)

Thread is one of important Class in Java and multi-threading is most widely used feature,but there is no clear way to stop Thread in Java. Earlier there was a stop method exists in Thread Class but Java deprecated that method citing some safety reason. By default a Thread stops when execution of run() method finish either normally or due to any Exception.In this article we will How to Stop Thread in Java by using a boolean State variable or flag. Using flag to stop Thread is very popular way  of stopping thread and its also safe, because it doesn't do anything special rather than helping run() method to finish it self.

## How to Stop Thread in Java

[](http://javarevisited.blogspot.com/2011/09/generics-java-example-tutorial.html)

As I said earlier Thread in Java will stop once run() method finished. Another important point is that you can not restart a Thread which run() method has finished already , you will get an IllegalStateExceptio, here is a Sample Code for Stopping Thread in Java:

### Sample Code to Stop Thread in Java

    private class Runner extends Thread{

    boolean bExit = false;

    public void exit(boolean bExit){

        this.bExit = bExit;

    }

    @Override

    public void run(){

        while(!bExit){

            System.out.println("Thread is running");

                try {

                    Thread.sleep(500);

                } catch (InterruptedException ex) {

                    Logger.getLogger(ThreadTester.class.getName()).log(Level.SEVERE, null, ex);

                }

        }

    }

}

**Should we make bExit Volatile**

Since every Thread has its own local memory in Java its good practice to make bExit volatile because we may alter value of bExit from any thread and making it volatile guarantees that Runner will also see any update done before making bExit.

That’s all on how to stop thread in Java , let me know if you find any other way of stopping threads in Java without using deprecated stop() method.

Read more: <http://javarevisited.blogspot.com/2011/10/how-to-stop-thread-java-example.html#ixzz2gmFpBbcx>

# [How to implement Thread in Java ?Example of Runnable interface](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html)

**How to implement Thread in Java**  
In my opinion Thread is the most wonderful feature of Java programming language and I remember when I started learning Java in one of programming class in India how important Thread was portrait and how much emphasis given on clear understanding of [multi threading](http://javarevisited.blogspot.sg/2011/07/java-multi-threading-interview.html). It’s indeed still popular and one of most sought after skill in Java because writing concurrent and multi-threaded application in Java is challenging, despite Java providing excellent support at language level using [synchronized](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html) and [volatile](http://javarevisited.blogspot.sg/2011/06/volatile-keyword-java-example-tutorial.html) keyword. Main problem with using multiple threads and writing multi-threaded code is issues related to concurrency e.g. [deadlock](http://javarevisited.blogspot.sg/2010/10/what-is-deadlock-in-java-how-to-fix-it.html), livelock,  [race conditions](http://javarevisited.blogspot.sg/2012/02/what-is-race-condition-in.html) etc, It takes lot of effort to implement multi-threading correctly in Java application.  In this core java tutorial I will share my experience on different way of implementing Thread in Java; [Difference between Thread and Runnable in Java](http://javarevisited.blogspot.sg/2012/01/difference-thread-vs-runnable-interface.html) is also a very [common core java interview question](http://javarevisited.blogspot.com/search/label/core%20java%20interview%20question) and asked mostly during junior level java interview.  
  
There are two ways of implementing threading in Java   
1) By extending java.lang.Thread class, or  
2) By implementing java.lang.Runnable interface.  
  
Before we go into implementation details I just like to cover when we use Thread in Java ? so we use thread if we want some part of code is executed parallel and we put that code inside run() method of either [Thread class or Runnable interface](http://javarevisited.blogspot.sg/2012/01/difference-thread-vs-runnable-interface.html).  
  
Actually public void run () method is defined in Runnable interface and since java.lang.Thread class implements Runnable [interface](http://javarevisited.blogspot.sg/2012/04/10-points-on-interface-in-java-with.html) it gets this method automatically. I remember by first Java multi threading example which was an animation program where multiple threads were used in Applet to create animation of words falling from top left, middle and top right of the page. That was pretty exciting at that time because till then I only know program which takes input from command prompt and [print output on command prompt](http://javarevisited.blogspot.sg/2011/11/run-java-program-from-command-prompt.html).

## Java Thread Tutorial and Example

[How to create Thread in Java - Thread and Runnable class](http://3.bp.blogspot.com/-K6q0DQ1v-tw/TWu8owBtc2I/AAAAAAAAADA/oBoHDBiJ8ag/s1600/17.jpg)So now the interview question "*Which way of implementing Thread is better? Extending Thread class or implementing Runnable method*?  
In my opinion implementing Runnable is better because in Java we can only extend one class so if we extend Thread class we can not extend any other class while by implementing Runnable interface we still have that option open with us.   
  
Second reason which make sense to me is more on [OOPS concept](http://javarevisited.blogspot.sg/2012/03/10-object-oriented-design-principles.html) according to OOPS if we [extend a class](http://java67.blogspot.sg/2012/08/what-is-inheritance-in-java-oops-programming-example.html) we provide some new feature or functionality , So if the purpose is just to use the run() method to define code its better to use Runnable interface.  
  
till then we have just created a thread , Thread will not start until you call the start() method of java.lang.Thread class. When we call start () method Java Virtual machine execute run () method of that Thread class into separate Thread other than calling thread. Anybody guess what will happen if we call the [run() method directly instead of calling start() method](http://javarevisited.blogspot.sg/2012/03/difference-between-start-and-run-method.html) ?   
  
That [another popular multi-threading interview question](http://javarevisited.blogspot.sg/2011/07/java-multi-threading-interview.html) and answer is simple there would be no Error or Exception run() method will simply be executed in the same Thread and new Thread will not be created. Another follow up question would be what will happen if you call start() method twice in same Thread object e.g.

mythread.start();   
mythread.start(); *//this line will throw IllegalThreadStateException*  
  
  
*//implementing Thread by extending Thread class*  
 **public** **class** MyThread **extends** **Thread**{         
  
   **public** **void** run(){  
      **System**.out.println(" Thread Running " + **Thread**.currentThread().getName());  
   }  
 }  
  
  
*//implementing Thread by implementing Runnable interface*  
  
**public** **class** MyRunnable **implements** **Runnable**{           
  
    **public** **void** run(){  
       **System**.out.println(" Create Thread " + **Thread**.currentThread().getName());  
    }  
  
 }  
  
  
*//starting Thread in Java*  
**Thread** mythread = **new** MyThread(); *//Thread created not started*  
mythread.setName("T1");  
**Thread** myrunnable = **new** **Thread**(**new** MyRunnable(),"T2"); *//Thread created*         
  
mythread.start(); //Thread started now but not running   
myrunnable.start();

TIP1: It’s not guaranteed that mythread will start before myrunnable it depends upon Thread scheduler.  
  
TIP2: Thread will be said to go on dead state once execution of run() method finished and you can not start that thread again.

Read more: <http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html#ixzz2gmFuuq4x>