**)  What is Thread in Java?** The thread is an independent path of execution. It's way to take advantage of multiple CPU available in a machine. By employing multiple threads you can speed up CPU bound task. For example, if one thread takes 100 milliseconds to do a job, you can use 10 thread to reduce that task into 10 milliseconds. Java provides excellent support for multithreading at the language level, and it's also one of the strong selling points.  
  
  
**2)  What is the difference between Thread and Process in Java?**   
The thread is a subset of Process, in other words, one process can contain multiple threads. Two process runs on different memory space, but all threads share same memory space. Don't confuse this with stack memory, which is different for the different thread and used to store local data to that thread.   
  
**3)  How do you implement Thread in Java?**   
At the language level, there are two ways to implement Thread in Java. An instance of java.lang.Thread represent a thread but it needs a task to execute, which is an instance of interface java.lang.Runnable. Since Thread class itself implement Runnable, you can override run() method either by extending Thread class or just implementing Runnable interface.

**4)  When to use Runnable vs Thread in Java?**   
This is a follow-up of previous multi-threading interview question. As we know we can implement thread either by extending Thread class or implementing Runnable interface, the question arise, which one is better and when to use one? This question will be easy to answer if you know that Java programming language doesn't support multiple inheritances of class, but it allows you to implement multiple interfaces. Which means, it's better to implement Runnable then extends Threadif you also want to extend another class e.g. Canvas or CommandListener.

**6)  What is the difference between start() and run() method of Thread class?**   
One of trick Java question from early days, but still good enough to differentiate between shallow understanding of Java threading model start() method is used to start newly created thread, while start() internally calls run() method, there is difference calling run() method directly. When you invoke run() as normal method, its called in the same thread, no new thread is started, which is the case when you call start() method.

**7)  What is the difference between Runnable and Callable in Java?**   
Both Runnable and Callable represent task which is intended to be executed in a separate thread. Runnable is there from JDK 1.0 while Callable was added on JDK 1.5. Main difference between these two is that Callable's call() method can return value and throw Exception, which was not possible with Runnable's run() method. Callable return Future object, which can hold the result of computation.

**8)  What is the difference between CyclicBarrier and CountDownLatch in Java?**   
Though both CyclicBarrier and CountDownLatch wait for number of threads on one or more events, the main difference between them is that you can not re-use CountDownLatch once count reaches to zero, but you can reuse same CyclicBarrier even after barrier is broken.     
  
**9)  What is Java Memory model?**   
Java Memory model is set of rules and guidelines which allows Java programs to behave deterministically across multiple memory architecture, CPU, and operating system. It's particularly important in case of multi-threading. Java Memory Model provides some guarantee on which changes made by one thread should be visible to others, one of them is happens-before relationship. This relationship defines several rules which allows programmers to anticipate and reason behaviour of concurrent Java programs. For example, happens-before relationship guarantees :

* Each action in a thread happens-before every action in that thread that comes later in the program order, this is known as program order rule.
* An unlock on a monitor lock happens-before every subsequent lock on that same monitor lock, also known as Monitor lock rule.
* A write to a volatile field happens-before every subsequent read of that same field, known as Volatile variable rule.
* A call to Thread.start on a thread happens-before any other thread detects that thread has terminated, either by successfully return from Thread.join() or by Thread.isAlive() returning false, also known as Thread start rule.
* A thread calling interrupt on another thread happens-before the interrupted thread detects the interrupt( either by having InterruptedException thrown, or invoking isInterrupted or interrupted), popularly known as Thread Interruption rule.
* The end of a constructor for an object happens-before the start of the finalizer for that object, known as Finalizer rule.
* If A happens-before B, and B happens-before C, then A happens-before C, which means happens-before guarantees Transitivity.

**10) What is volatile variable in Java?**

volatile is a special modifier, which can only be used with instance variables. In concurrent Java programs, changes made by multiple threads on instance variables is not visible to other in absence of any synchronizers e.g. synchronized keyword or locks.

**11) What is thread-safety? is Vector a thread-safe class?**    
Thread-safety is a property of an object or code which guarantees that if executed or used by multiple threads in any manner e.g. read vs write it will behave as expected. For example, a thread-safe counter object will not miss any count if same instance of that counter is shared among multiple threads. Apparently, you can also divide collection classes in two category, thread-safe and non-thread-safe. Vector is indeed a thread-safe class and it achieves thread-safety by synchronizing methods which modify state of Vector, on the other hand, its counterpart ArrayList is not thread-safe.  
  
  
**12) What is race condition in Java? Given one example?**  
Race condition are cause of some subtle programming bugs when Java programs are exposed to concurrent execution environment. As the name suggests, a race condition occurs due to race between multiple threads, if a thread which is supposed to execute first lost the race and executed second, behaviour of code changes, which surface as non-deterministic bugs. This is one of the hardest bugs to find and re-produce because of random nature of racing between threads.   
  
**13) How to stop a thread in Java?**   
Java doesn't provide a sure shot way of stopping thread. There was some control methods in JDK 1.0 e.g. stop(), suspend() and resume() which was deprecated in later releases due to potential deadlock threats, from then Java API designers has not made any effort to provide a consistent, thread-safe and elegant way to stop threads. Programmers mainly rely on the fact that thread stops automatically as soon as they finish execution of run() or call() method. To manually stop, programmers either take advantage of volatile boolean variable and check in every iteration if run method has loops or interrupt threads to abruptly cancel tasks.   
  
  
  
**14) What happens when an Exception occurs in a thread?**  
In simple words, If not caught thread will die, if an uncaught exception handler is registered then it will get a call back. Thread.UncaughtExceptionHandler is an interface, defined as nested interface for handlers invoked when a Thread abruptly terminates due to an uncaught exception. When a thread is about to terminate due to an uncaught exception the Java Virtual Machine will query the thread for itsUncaughtExceptionHandler using Thread.getUncaughtExceptionHandler() and will invoke the handler's uncaughtException() method, passing the thread and the exception as arguments.  
  
  
**15) How do you share data between two thread in Java?**   
You can share data between threads by using shared object, or concurrent data structure like BlockingQueue. See this tutorial to learn [inter-thread communication in Java](http://javarevisited.blogspot.sg/2013/12/inter-thread-communication-in-java-wait-notify-example.html). It implements Producer consumer pattern using wait and notify methods, which involves sharing objects between two threads.

16) **16) What is the difference between notify and notifyAll in Java?**  
This is another tricky questions from core Java interviews, since multiple threads can wait on single monitor lock, Java API designer provides method to inform only one of them or all of them, once waiting condition changes, but they provide half implementation. There notify() method doesn't provide any way to choose a particular thread, that's why its only useful when you know that there is only one thread is waiting. On the other hand, notifyAll() sends notification to all threads and allows them to compete for locks, which ensures that at-least one thread will proceed further.  
  
**17) Why wait, notify and notifyAll are not inside thread class?**   
This is a design related question, which checks what candidate thinks about existing system or does he ever thought of something which is so common but looks in-appropriate at first. In order to answer this question, you have to give some reasons why it make sense for these three method to be in Object class, and why not on Thread class. One reason which is obvious is that Java provides lock at object level not at thread level. Every object has lock, which is acquired by thread. Now if thread needs to wait for certain lock it make sense to call wait() on that object rather than on that thread. Had wait() method declared on Thread class, it was not clear that for which lock thread was waiting. In short, since wait, notify and notifyAll operate at lock level, it make sense to defined it on object class because lock belongs to object.

**18) What is ThreadLocal variable in Java?**   
ThreadLocal variables are special kind of variable available to Java programmer. Just like instance variable is per instance, ThreadLocal variable is per thread. It's a nice way to achieve thread-safety of expensive-to-create objects, for example you can make SimpleDateFormat thread-safe using ThreadLocal. Since that class is expensive, its not good to use it in local scope, which requires separate instance on each invocation. By providing each thread their own copy, you shoot two birds with one arrow. First, you reduce number of instance of expensive object by reusing fixed number of instances, and Second, you achieve thread-safety without paying cost of synchronization or immutability. Another good example of thread local variable is ThreadLocalRandom class, which reduces number of instances of expensive-to-create Random object in multi-threading environment.

**19) What is FutureTask in Java?**  
FutureTask represents a cancellable asynchronous computation in concurrent Java application. This class provides a base implementation of Future, with methods to start and cancel a computation, query to see if the computation is complete, and retrieve the result of the computation. The result can only be retrieved when the computation has completed; the get methods will block if the computation has not yet completed. A FutureTask object can be used to wrap a Callable or Runnable object. Since FutureTask also implements Runnable, it can be submitted to an Executor for execution.  
  
**20) What is the difference between the interrupted() and isInterrupted() method in Java?**  
Main difference between interrupted() and isInterrupted() is that former clears the interrupt status while later does not. The interrupt mechanism in Java multi-threading is implemented using an internal flag known as the interrupt status. Interrupting a thread by calling Thread.interrupt() sets this flag. When interrupted thread checks for an interrupt by invoking the [static method](http://java67.blogspot.com/2012/11/what-is-static-class-variable-method.html) Thread.interrupted(), interrupt status is cleared. The non-static isInterrupted()method, which is used by one thread to query the interrupt status of another, does not change the interrupt status flag. By convention, any method that exits by throwing an InterruptedException clears interrupt status when it does so. However, it's always possible that interrupt status will immediately be set again, by another thread invoking interrupt.

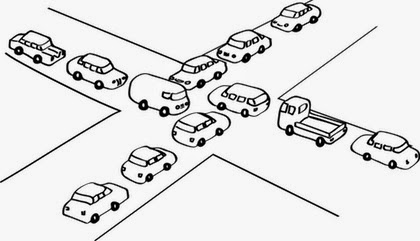
**21) Why wait and notify method are called from synchronized block?**   
Main reason for calling wait and notify method from either synchronized block or method is that it made mandatory by Java API. If you don't call them from synchronized context, your code will throw IllegalMonitorStateException. A more subtle reason is to avoid the race condition between wait and notify calls.

**22) Why should you check condition for waiting in a loop?**   
Its possible for a waiting thread to receive false alerts and spurious wake up calls, if it doesn't check the waiting condition in loop, it will simply exit even if condition is not met. As such, when a waiting thread wakes up, it cannot assume that the state it was waiting for is still valid. It may have been valid in the past, but the state may have been changed after the notify() method was called and before the waiting thread woke up. That's why it always better to call wait() method from loop, you can even create template for calling wait and notify in Eclipse.

**23) What is the difference between synchronized and concurrent collection in Java?**  
Though both synchronized and concurrent collection provides thread-safe collection suitable for multi-threaded and concurrent access, later is more scalable than former. Before Java 1.5, Java programmers only had synchronized collection which becomes source of contention if multiple thread access them concurrently, which hampers scalability of system. Java 5 introduced concurrent collections like ConcurrentHashMap, which not only provides thread-safety but also improves scalability by using modern techniques like lock stripping and partitioning internal table.

**24) What is thread pool? Why should you thread pool in Java?**  
Creating thread is expensive in terms of time and resource. If you create thread at time of request processing it will slow down your response time, also there is only a limited number of threads a process can create. To avoid both of these issues, a pool of thread is created when application starts-up and threads are reused for request processing. This pool of thread is known as "thread pool" and threads are known as worker thread. From JDK 1.5 release, Java API provides Executor framework, which allows you to create different types of thread pools e.g. single thread pool, which process one task at a time, fixed thread pool (a pool of fixed number of threads) or cached thread pool (an expandable thread pool suitable for applications with many short lived tasks).

25) **How do you avoid deadlock in Java? Write Code?**

[](http://4.bp.blogspot.com/-m2IldPcxiJI/U6-Zwvkdd1I/AAAAAAAABns/-zHIHjzM3nM/s1600/deadlock+in+Java.jpg)

Deadlock is a condition in which two threads wait for each other to take action which allows them to move further. It's a serious issue because when it happen your program hangs and doesn't do the task it is intended for. In order for deadlock to happen, following four conditions must be true:

* **Mutual Exclusion:** At least one resource must be held in a non-shareable mode. Only one process can use the resource at any given instant of time.
* **Hold and Wait:** A process is currently holding, at least, one resource and requesting additional resources which are being held by other processes.
* **No Pre-emption:** The operating system must not de-allocate resources once they have been allocated; they must be released by the holding process voluntarily.
* **Circular Wait:**A process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource.

The easiest way to avoid deadlock is to prevent *Circular wai*t, and this can be done by acquiring locks in a particular order and releasing them in reverse order so that a thread can only proceed to acquire a lock if it held the other one. 

26) **What is the difference between livelock and deadlock in Java?**  
A livelock is similar to a deadlock, except that the states of the threads or processes involved in the livelock constantly change with regard to one another, without any one progressing further. Livelock is a special case of resource starvation. A real-world example of livelock occurs when two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass, but they end up swaying from side to side without making any progress because they both repeatedly move the same way at the same time. In short, the main difference between livelock and deadlock is that in former state of process change but no progress is made.

27)**How do you check if a Thread holds a lock or not?**   
I didn't even know that you can check if a Thread already holds lock before this question hits me in a telephonic round of Java interview. There is a method called holdsLock() on java.lang.Thread, it returns true if and only if the current thread holds the monitor lock on the specified object.

28) **Which JVM parameter is used to control stack size of a thread?**   
This is the simple one, -Xss parameter is used to control stack size of Thread in Java.

29) **32) What is the difference between synchronized and ReentrantLock in Java?**   
There were days when the only way to provide mutual exclusion in Java was via synchronized keyword, but it has several shortcomings e.g. you can not extend lock beyond a method or block boundary, you can not give up trying for a lock etc. Java 5 solves this problem by providing more sophisticated control via Lock interface. ReentrantLock is a common implementation of Lock interface and provides re-entrant mutual exclusion Lock with the same basic behavior and semantics as the implicit monitor lock accessed using synchronized methods and statements, but with extended capabilities.

30) **34) What does yield method of Thread class do?**   
Yield method is one way to request current thread to relinquish CPU so that other thread can get a chance to execute. Yield is a static method and only guarantees that current thread will relinquish the CPU but doesn't say anything about which other thread will get CPU. Its possible for the same thread to get CPU back and start its execution again.

31) **What is the concurrency level of ConcurrentHashMap in Java?**   
ConcurrentHashMap achieves it's scalability and thread-safety by partitioning actual map into a number of sections. This partitioning is achieved using concurrency level. Its optional parameter of ConcurrentHashMap constructor and it's default value is 16. The table is internally partitioned to try to permit the indicated number of concurrent updates without contention. 

32) **What is Semaphore in Java?**(answer)  
Semaphore in Java is a new kind of synchronizer. It's a counting semaphore. Conceptually, a semaphore maintains a set of permits. Each acquire() blocks if necessary until a permit is available, and then takes it. Each release() adds a permit, potentially releasing a blocking acquirer. However, no actual permit objects are used; the Semaphore just keeps a count of the number available and acts accordingly. Semaphore is used to protect an expensive resource which is available in fixed number e.g. database connection in the pool.

33) **What is the difference between the volatile and atomic variable in Java?**   
This is an interesting question for Java programmer, at first, volatile and atomic variable look very similar, but they are different. Volatile variable provides you happens-before guarantee that a write will happen before any subsequent write, it doesn't guarantee atomicity. For example count++ operation will not become atomic just by declaring count variable as volatile. On the other hand AtomicInteger class provides atomic method to perform such compound operation atomically e.g.getAndIncrement() is atomic replacement of increment operator. It can be used to atomically increment current value by one. Similarly you have atomic version for other data type and reference variable as well.  
  
34) **How do you force to start a Thread in Java?**   
This question is like how do you force garbage collection in Java, there is no way though you can make a request using System.gc() but it's not guaranteed. On Java multi-threading there is absolute no way to force start a thread, this is controlled by thread scheduler and Java exposes no API to control thread schedule. This is still a random bit in Java.

35) **What is the difference between calling wait() and sleep() method in Java multi-threading?**   
1) wait is called from synchronized context only while sleep can be called without synchronized block  
  
2) waiting thread can be awake by calling notify and notifyAll while sleeping thread can not be awakened by calling notify method.  
  
3) wait is normally done on condition, Thread wait until a condition is true while sleep is just to put your thread on sleep.

4) wait for release lock on an object while waiting while sleep doesn’t release lock while waiting.  
  
5) The wait() method  is called on an Object on which the synchronized block is locked, while sleep is called on the Thread.

### 36) Why thread communication methods wait(), notify() and notifyAll() are in Object class?

In Java every Object has a monitor and wait, notify methods are used to wait for the Object monitor or to notify other threads that Object monitor is free now. There is no monitor on threads in java and synchronization can be used with any Object, that’s why it’s part of Object class so that every class in java has these essential methods for inter thread communication.

37)**When threads are not lightweight process in java?**

Threads are lightweight process only if threads of same process are executing concurrently. But if threads of different processes are executing concurrently then threads are heavy weight process.

**38)Are you aware of preemptive scheduling and time slicing?**

In preemptive scheduling, the highest priority thread executes until it enters into the waiting state.

In time slicing, a thread executes for a certain predefined time and then enters runnable pool. Than thread can enter running state when selected by thread scheduler.

**39)What are deamon threads?**

Deamon threads are low priority threads which runs intermittently in background for doing garbage collection.

* Thread scheduler schedules these threads only when CPU is idle.
* Deamon threads are service oriented threads, they serves all other threads.
* These threads are created before user threads are created and die after all other user threads dies.
* Priority of daemon threads is always 1 (i.e. MIN\_PRIORITY).
* User created threads are non daemon threads.
* JVM can exit when only daemon threads exist in system.
* we can use isDaemon() method to check whether thread is daemon thread or not.
* we can use setDaemon(boolean on) method to make any user method a daemon thread.
* If setDaemon(boolean on) is called on thread after calling start() method than IllegalThreadStateException is thrown.
* You may like to see how daemon threads work, for that you can use VisualVM or jStack. I have provided Thread dumps over there which shows daemon threads which were intermittently running in background.

Some of the daemon threads which intermittently run in background are garbage collector.

40) **Difference between object lock and class lock?**

|  |  |
| --- | --- |
| [Object lock](http://www.javamadesoeasy.com/2015/03/acquiring-lock-on-class-2-ways-to.html)  Thread can acquire [object lock](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) by-   1. Entering synchronized block or 2. by entering synchronized methods. | [Class lock](http://www.javamadesoeasy.com/2015/03/acquiring-lock-on-class-2-ways-to.html)  Thread can acquire lock on [class’s class object](http://www.javamadesoeasy.com/2015/03/acquiring-lock-on-class-2-ways-to.html) by-   1. Entering synchronized block or 2. by entering static synchronized methods. |
| [Multiple threads may exist on same object but only one thread of that object can enter synchronized method at a time.](http://www.javamadesoeasy.com/2015/03/suppose-you-have-2-threads-thread-1-and_5.html)    [Threads on different object can enter same method at same time.](http://www.javamadesoeasy.com/2015/03/suppose-you-have-2-threads-thread-1-on.html) | Multiple objects of class may exist but there is always one class’s class object lock available. |

**41)What do you mean by thread starvation?**

When thread does not enough CPU for its execution Thread starvation happens.

Thread starvation may happen in following scenarios >

* Low priority threads get less CPU (time for execution) as compared to high priority threads. Lower priority thread may starve away waiting to get enough CPU to perform calculations.
* In deadlock two threads waits for each other to release lock holded by them on resources. There both Threads starves away to get CPU.
* Thread might be waiting indefinitely for lock on object’s monitor (by calling wait method), because no other thread is calling notify/notifyAll() method on object. In that case, Thread starves away to get CPU.
* Thread might be waiting indefinitely for lock on object’s monitor (by calling wait () method), but notify () may be repeatedly awakening some other threads. In that case also Thread starves away to get CPU.

42) **What is addShutdownHook method in java?**

* addShutdownHook method registers a new virtual-machine shutdown hook.
* A shutdown hook is a initialized but unstarted thread.
* When JVM starts its shutdown it will start all registered shutdown hooks in some unspecified order and let them run concurrently.

When JVM (Java virtual machine)  shuts down >

* When the last non-deamon thread finishes, or
* when the System.exit is called.

Once JVM’s shutdown has begunnew shutdown hook cannot be registered neither previously-registered hook can be de-registered. Any attempt made to do any of these operations causes an IllegalStateException.

43) **What are thread priorities?**

Thread priority range is from 1 to 10.

Where 1 is minimum priority and 10 is maximum priority.

Thread class provides variables of final static int type for setting thread priority.

44) **Can you find whether thread holds lock on object or not?**

holdsLock(object) method can be used to find out whether current thread holds the lock on monitor of specified object.

holdsLock(object) method returns true if the current thread holds the lock on monitor of specified object.

45) **Does thread leaves object lock when sleep() method is called?**

Answer. When **sleep()** method is called Thread does not leaves object lock and goes from running to waiting state. Thread waits for sleep time to over and once sleep time is up it goes from waiting to runnable state.

46)**Does thread leaves object lock when wait() method is called?**

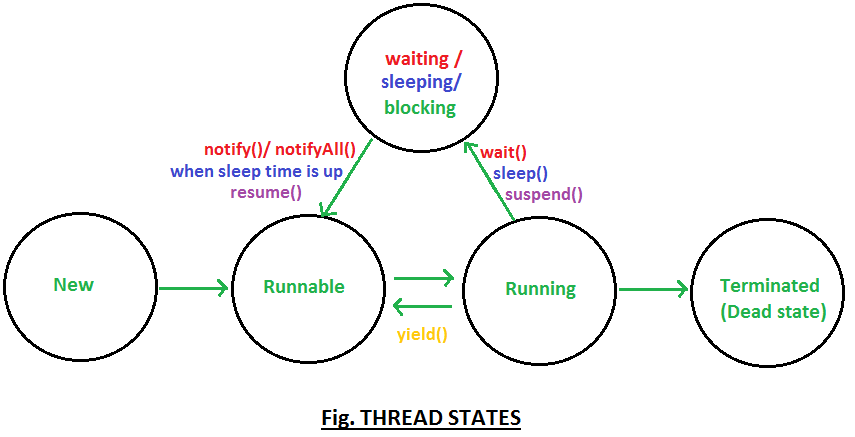
Answer. When wait() method is called Thread leaves the object lock and goes from running to waiting state. Thread waits for other threads on same object to call notify() or notifyAll() and once any of notify() or notifyAll() is called it goes from waiting to runnable state and again acquires object lock.

47) **What is life cycle of Thread, explain thread states? (Important)**

Thread have following states

* New
* Runnable
* Running
* Waiting/blocked/sleeping
* Terminated (Dead)

Thread states/ Thread life cycle in diagram >



Thread states in detail

New : When instance of thread is created using new operator it is in new state, but the start() method has not been invoked on the thread yet, thread is not eligible to run yet.

Runnable : When start() method is called on thread it enters runnable state.

Running : Thread scheduler selects thread to go fromrunnable to running state. In running state Thread starts executing by entering run() method.

Waiting/blocked/sleeping : In this state a thread is not eligible to run.

>Thread is still alive, but currently it’s not eligible to run. In other words.

> How can Thread go from running to waiting state?

 By calling wait()method thread go from running to waiting state. In waiting state it will wait for other threads to release object monitor/lock.

> How can Thread go from running to sleeping state?

 By calling sleep() method thread go from running to sleeping state. In sleeping state it will wait for sleep time to get over.

Terminated (Dead) : A thread is considered dead when its run() method completes.

48) How threads communicate between each other?

Answer. This is very must know question for all the interviewees, you will most probably face this question in almost every time you go for interview.

Threads can communicate with each other by using wait(), notify() and notifyAll()methods.

49) **Can you again start Thread?**

Answer.No, we cannot start Thread again, doing so will throw runtimeException java.lang.IllegalThreadStateException. The reason is once run() method is executed by Thread, it goes into dead state.

Let’s take an example-

Thinking of starting thread again and calling start() method on it (which internally is going to call run() method) for us is some what like asking dead man to wake up and run. As, after completing his life person goes to dead state.