A **thread is a single sequential flow of control** and takes an independent path of execution within a program. Threads are basically **used to do background processing and parallel processing when the order of execution doesn't matter**.

There is **exactly one way to create a new thread in Java** and that is to **instantiate** [**java.lang.Thread**](http://download.oracle.com/javase/7/docs/api/java/lang/Thread.html) (to actually **run that thread** we also need to call **start()**). Everything else that creates threads in Java falls back to this one way behind the cover (e.g. a [ThreadFactory](http://download.oracle.com/javase/7/docs/api/java/util/concurrent/ThreadFactory.html) implementation will instantiate Thread objects at some point, …).

There are **two different ways** **to specify which code to run in that Thread**:

1. **Implement the interface** [**java.lang.Runnable**](http://download.oracle.com/javase/7/docs/api/java/lang/Runnable.html) and **pass an instance** **of the class implementing it** **to** [**the Thread constructor**](http://download.oracle.com/javase/7/docs/api/java/lang/Thread.html#Thread%28java.lang.Runnable%29).
2. **Extend Thread itself** and **override its** [**run()**](http://download.oracle.com/javase/7/docs/api/java/lang/Thread.html#run%28%29) **method**.

The **first approach (implementing Runnable)** is usually **considered the more correct approach** because you don't usually create a new "kind" of Thread, but simply want to run some code (i.e. a Runnable) in a dedicated thread.

**wait()**: It **tells the calling thread to give up the lock and go to sleep until some other thread enters the same monitor and calls notify()**. The wait() method releases the lock prior to waiting and re-acquires the lock prior to returning from the wait() method.

**synchronized**(**lockObject**)

{

while(!condition)

{

**lockObject.wait()**;

}

// take the action here;

}

**notify()**: It **wakes up single thread which called wait() on the same object**. It should be noted that **calling notify() does not actually give up a lock on a resource**. It **tells a waiting thread that that thread can wake up**. However, **the lock is not actually given up until the notifier’s synchronized block has been completed**. So, if a notifier calls notify() on a resource but the notifier still needs to perform 10 seconds of actions on the resource within its synchronized block, the thread that had been waiting will need to wait at least another additional 10 seconds for the notifier to release the lock on the object, even though notify() had been called.

**synchronized**(**lockObject**)

{

// establish\_the\_condition;

**lockObject.notify()**;

// any additional code if needed

}

**notifyAll()**: It **wakes up all the threads that are called wait() on the same object**. The **highest priority thread will run first in most of the situations, though not guaranteed**. Other things are the same as notify() method above.

**synchronized**(**lockObject**)

{

// establish\_the\_condition;

**lockObject.notifyAll()**;

}

**wait(timeout)**: It **wakes up the calling thread if it is still waiting after the timeout has elapsed or a notify() is called**. wait(timeout) with a timeout > 0 allows to potentially save from locking up the application "forever" if a call to notify() never occurs.

**sleep(timeout)**: **won't wake up until the timeout has elapsed**;