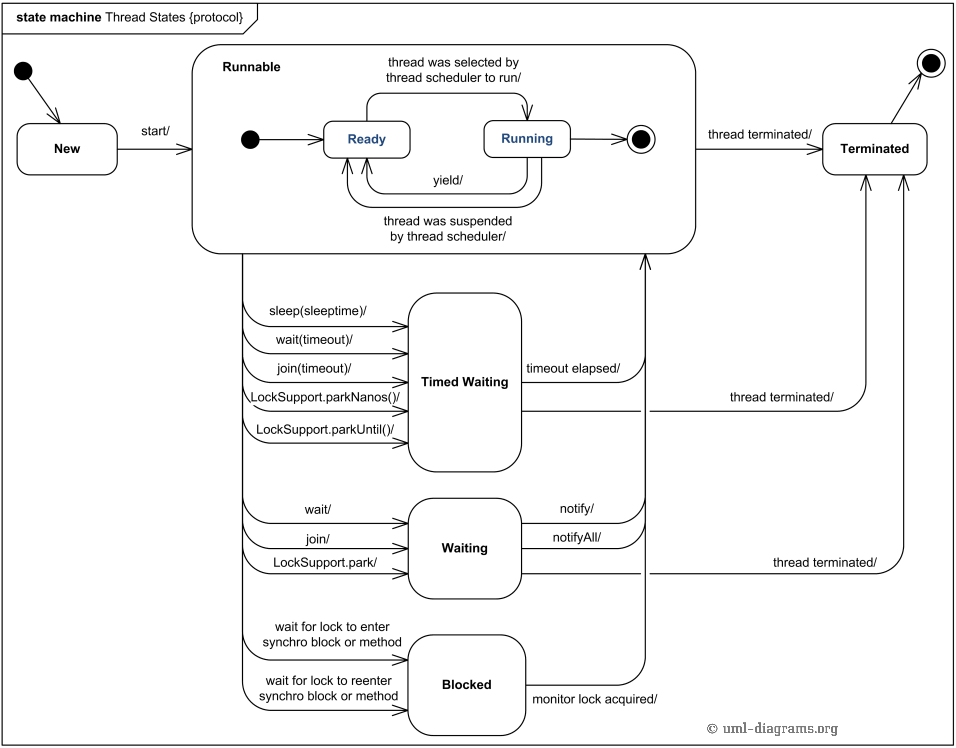
Thread life cycle:



## Sleep Method in Thread:

There will be some scenarios where in you want a thread to stop executing the code for a period of time and then start again.

Sleep is static method

* Sleep method in Thread tells the currently executing thread to sleep for specified amount of time.
* Sleep method accepts the time in MilliSeconds.
* It can throw interruptedException . So it should be embedded in the try catch block
* Remember that there is no guarantee that the thread will go to Sleep state the moment is it executed and no guarantee that the thread will sleep for specified amount of time. The thread scheduler can wake it up any time.
* Once the thread completes or out of its sleep state, it can move to Running or Runnable state.

## ****Thread Priorities:****

Understanding the Thread priorities is important to know how [yield](http://java9s.com/topic/yield) method works in Multithreading.

* Remember that all the threads carry normal priority when a priority is not specified.
* Priorities can be specified from 1 to 10. 10 being the highest, 1 being the lowest priority and 5 being the normal priority.
* Remember that the thread with highest priority will be given preference in execution. But there is no guarantee that it will be in running state the moment it starts.
* Always the currently executing thread might have the higher priority when compared to the threads in the pool who are waiting for their chance.
* It is the thread scheduler which decides what thread should be executed.
* t.setPriority() can be used to set the priorities for the threads.
* Remember that the priorities should be set before the threads start method is invoked.
* You can use the constants, MIN\_PRIORITY,MAX\_PRIORITY and NORM\_PRIORITY for setting priorities.

## ****Yield Method in Thread:****

* Yield is a Static method .
* Yield tells the currently executing thread to give a chance to the threads that have equal priority in the Thread Pool.
* There is no guarantee that Yield will make the currently executing thread to runnable state immediately.
* Remember an important point that yield method does not make the thread to go to Wait or Blocked state. It can only make a thread from Running State to Runnable State.

## ****Join Method in Thread:****

* Join method belongs to Thread object and not a static member. When join is invoked an a thread instance, this thread will tell currently executing thread to wait till the Joining thread completes.
* Join is used in the situations when a task which should be completed before the current task is going to finish.
* Just to give another example of how Join works, say we have a slow train on the track moving and another fast train should take the same track, so, we make way to the fast train and tell the slow train to join once after the fast one is left.

## ****wait Method in Thread:****

**wait( )**tells the calling thread to give up the monitor and go to sleep until some other   
thread enters the same monitor and calls **notify( )**.

## ****notify Method in Thread:****

**notify( )**wakes up the first thread that called **wait( )**on the same object.

## ****notifyAll Method in Thread:****

**notifyAll( )**wakes up all the threads that called **wait( )**on the same object. The   
highest priority thread will run first.

The Object class in Java[S](http://java.sun.com/)[W](http://en.wikipedia.org/wiki/Java_(programming_language)) has three final methods that allow threads to communicate about the locked status of a resource. These methods are wait(), notify(), and notifyAll(). A thread obtains a lock for a particular resource via a synchronized block with an instance of that resource. Suppose that a thread requires that another thread perform a certain action on the resource before it acts on the resource. That thread can synchronize on the resource and call the wait() method on resource. This says that the thread will wait until it has been notified that it can proceed to act. The wait() method can take an optional timeout value as a parameter. If this value is used, it means that the thread will either wait until it's notified or it will continue to execute once the timeout value has passed.

If a thread is required to perform a task on a resource before another thread operates on the resource (and the other thread is waiting via the wait() method on the resource), the thread needs synchronize on the resource. It can perform its actions on the resource. In order to notify the waiting thread once these actions have completed, the notify() method on the resource is called. This notifies the waiting thread that it can proceed to act. If multiple threads are waiting for the resource, there is no guarantee as to which thread will be given access to the resource. If it is desired for all waiting threads to be awoken, the notifyAll() method can be called on the resource.

Start and run method difference:

Start will create a new thread whereas run will execute as a normal method.

Thread and process difference:

Extending Thread class or implementing runnable interface:

**Race condition:**

A race condition is a situation in which two or more threads or processes are reading or writing some shared data, and the final result depends on the timing of how the threads are scheduled. Race conditions can lead to unpredictable results and subtle program bugs. A thread can prevent this from happening by locking an object. When an object is locked by one thread and another thread tries to call a synchronized method on the same object, the second thread will block until the object is unlocked.

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 readoperation 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.

## How to find Race Conditions in Java

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.

**What is deadlock** **?**"  
answer is simple , when two or more threads waiting for each other to release lock and get stuck for infinite time , situation is called deadlock . it will only happen in case of multitasking.

**Atomic operation:**

Atomic means each action take place in one step without interruption or we can say that operation is performed as a single unit of work without the possibility of interference from other operations.

An Atomic operation cannot stop in the middle, either it happened completely or doesn't happen at all.

**Java language specification guarantees that**

* Reading or writing of a variable/reference is atomic unless the variable is of type long or double.
* Read and write are atomic for all variable declared volatile including long and double variables.

Atomic action can be used without fear of thread interference.

Using simple atomic variable access is more efficient than accessing the same variable through synchronized code. But it require more  care and attention from programmer to avoid memory consistency .

Volatile Keyword:

Volatile keyword in Java is used as an indicator to Java compiler and Thread that do not cache value of this variable and always read it from [main memory](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html). So if you want to share any variable in which read and write operation is atomic by implementation e.g. read and write in int or boolean variable you can declare them as volatile variable.   
  
Java volatile keyword cannot be used with method or class and it can only be used with variable

When to use:

1) You can use Volatile variable if you want to read and write long and [double](http://javarevisited.blogspot.sg/2011/10/convert-double-to-string-example.html) variable atomically. long and double both are [64 bit](http://javarevisited.blogspot.sg/2012/01/find-jvm-is-32-or-64-bit-java-program.html) data type and by default writing of long and double is not atomic and platform dependence. Many platform perform write in long and double variable 2 step, writing 32 bit in each step, due to this its possible for a Thread to see 32 bit from two different write. You can avoid this issue by making long and double variable volatile in Java.  
  
  
2) Volatile variable can be used as an alternative way of achieving [synchronization in Java](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html) which is very expensive process in some cases, like Visibility. with volatile variable its guaranteed that all reader thread will see updated value of volatile variable once write operation completed, without volatile keyword different reader thread may see different values.  
  
  
3) volatile variable can be used to inform compiler that a particular field is subject to be accessed by multiple threads, which will prevent compiler from doing any reordering or any kind of optimization which is not desirable in multi-threaded environment. Without volatile variable compiler can re-order code, free to cache value of volatile variable instead of always reading from main memory.

Synchronization:

Synchronization in Java is an important concept since Java is a multi-threaded language where multiple threads run in parallel to complete program execution.

Synchronization in Java is possible by usingJava keywords ***"synchronized"*** and ***"volatile”***. Concurrent access of shared objects in Java introduces to kind of errors: thread interference and memory consistency errors and to avoid these errors you need to properly synchronize your Java object to allow mutual exclusive access of critical section to two threads.