IMPORTANT METHODS OF THE THREAD CLASS

Thread class has many important methods like init, start, stop, run, getName(), getPriority(), isAlive() and join(), and also few static methods like sleep(), yield() etc. Usually we override only the run() method and use the inherited versions of init, start etc

start() and run()

Invoking the **start()** method on a thread object executes the **run()** method as a new thread of execution. When you call the start method for a thread object, it will call a native code method that causes the OS to initiate another thread from which the run() method executes. We can call start method only once on a thread or it will throw IllegalStateException.
You can also call the run method directly, but if you call the run() method directly, it would simply operate like any other method and will run as part of the same thread that called it.

final isAlive()

The final **isAlive()** method returns true if the thread is still running or the Thread has not terminated.

final join()

The final **join()** method waits until thread on which it is called is terminated. For example, thread1.join() suspends the current thread until thread1 dies.
You can also pass a long value to join method to specify the number of milliseconds you are prepared to wait for the death of a thread. For example, thread1.join(1000) wait upto 1 second for thread1 to die, and then continue execution.

The join() method can throw an InterruptedException if the current thread is interrupted by another thread.

yield()

Calling **yield()** will move the current thread from running to runnable, to give other threads a chance to execute. However the scheduler may still bring the same thread back to running. A better alternative according to Effective Java by Joshua Bloch is to use Thread.sleep(1) instead of Thread.yield().

Wait, notify and notifyAll

Java.lang.Object provides three methods – notify(), notifyAll() and wait () – to improve the efficiency communication between threads. You will need to understand the synchronization process in Java to understand the communication using wait, notify and notifyAll.

- **obj.wait()** makes a thread wait on an object (obj) until it receives a notification from a notify() or notifyAll() on the same object.
- **obj.notify()** sends a notification to any one waiting thread on ab object (obj) that the object lock is available.
- **obj.notifyAll()** sends notification to all waiting threads on an object (obj) that the object lock is available.

You should call the above methods inside a synchronized block, or you will get IllegalMonitorStateException. The IllegalMonitorStateException is thrown to indicate that a thread has attempted to wait on an object's monitor or to notify other threads waiting on an object's monitor without owning the specified monitor.

The wait, notify and notifyAll methods should be used with caution for thread communication; if not used properly, it may result in deadlock.

interrupt(), isInterrupted() and interrupted()

Thread's stop() method is deprecated and should not be used. Instead, you should use the interrupt mechanism.

A thread can signal another thread that it should stop executing by calling the **interrupt() method** for that thread. This doesn't stop the thread, but sets a flag in the thread. This flag must be checked in the run() method to have any effect and the thread should then terminate itself.
The **isInterrupted() method** returns true if the interrupted flag has been set. This method does not reset the flag, but another **static method interrupted()** tests the flag for the currently executing thread and if it has been interrupted, it clears the interrupted flag in the current thread object and returns true.
When an **InterruptedException** is thrown, the flag that registers the interrupt is cleared, so a subsequent call to isInterrupted() or interrupted() returns false.

static currentThread()

The static **currentThread()** method returns a reference to the thread in which it is called.

static sleep()

This static **sleep()** method causes the thread to suspend execution for a given time. The sleep method has two overloaded versions:

- static void sleep (long milliseconds) throws InterruptedException
- static void sleep (long milliseconds, int nanoseconds) throws InterruptedException

Sleep vs Wait

Both sleep() and wait() methods can also be used to suspend a thread of execution for a specified time. The difference is that, when sleep() is executed inside a synchronized block, the object still holds the lock, but when wait() method is executed, it releases the lock and breaks the synchronization block, so other threads can acquire the lock.

The wait() method is used in connection with notify() or notifyAll() methods in thread communication. A waiting thread comes out of waiting when some other thread notify it using notify() or notifyAll() on the same object on which it is waiting. There is an overloaded version of the wait that takes a time as input and comes out of wait when that time is over even though no one has notified using notify() or notifyAll().

Source : http://javajee.com/important-methods-of-the-thread-class