SCJP Tips for Threads
www.techfaq360.com
Define, instantiate, and start new threads using both java.lang.Thread and java.lang.Runnable.

Two ways to create a new thread:
1. Have a class implement the Runnable interface. Ex:

```
class X implements Runnable
{
  public void run()  //must implement this method.
  {
    ...
  }
}
```

Now, create a Thread object :

```
 X obj = new X(); //Not a new thread yet.
 Thread t = new Thread( obj );  //This creates a new Thread. It's not started yet.
```

2. Have a class extend from Thread class. Ex:

```
class X extends Thread
{
  public void run(){ ... } //should implement this method to do something useful.
}
```

Now, create a Thread object :

```
Thread t = new X();
```

---

Points to remember:
1. A thread is started only when you call start() on a Thread object. In above example, t.start() will start the newly created thread.
2. Calling t.run() DOES NOT start a new thread. It will execute in the same thread just like any other method.
3. Method run() of thread class is not abstract, so not implementing it in a subclass it not a problem. But the Thread class's run method doesn't do anything.
4. Thread is created only by instantiating Thread or a subclass of Thread
5. Instantiating objects of classes that implement Runnable does not create new thread.

---

Recognize conditions that might prevent a thread from executing.

Methods that will definitely stop/pause a running thread:
sleep() : Does not release the locks (if any).

wait() : should have the lock before calling this method. It releases the lock and waits till somebody calls a notify/notifyAll.
stop() : releases all the locks. Deprecated.
suspend() : DOES NOT release any locks. Deprecated.

Methods that MAY or MAY NOT stop/pause a running thread:
yield() : If there are no threads of the same priority, this call is ignored
setPriority() : even if you lower the priority, the OS may not preempt this thread.
notify/notifyAll() : These methods simply release the locks and other thread which are waiting on them become "read to run". But CPU may or may not schedule them.

Points to note :
join() : It will pause the current thread till the thread on which it has called join, dies.

---

Write code using synchronized, wait, notify, or notifyAll, to protect against concurrent access problems and to communicate between threads. Define the interaction between threads and between threads and object locks when executing synchronized, wait, notify, or notifyAll.
Important Facts:

1. A "lock" is a part of any object. One object has only one lock but it may be acquired multiple times (but only by the same thread which already has got it for the first time). If a thread acquires the lock twice then it should release it twice.
2. For static blocks (where there is no instance), there is a class object for that class which has a lock.
3. It's the thread (not a Thread object but the flow of control) that 'acquires' lock. Understand the distinction between a Thread object and a thread. Thread object is just another object. A thread is the flow of control that executes the code. You need a Thread object to create a thread.
4. As there is only one lock for one object, only one thread can get the lock for an object at any given time.

Points to remember:

1. The thread that is calling wait/notify/notifyall on an object MUST have the lock of that object otherwise an IllegalMonitorState exception will be thrown. In other words, acquiring lock of one object and calling notify() on another DOES NOT WORK.
2. When a thread tries to enter a synchronized method/block, it waits till it acquires the lock for the object whose method it is trying to enter. For static methods, it waits for the class object's lock.
3. A thread dies when it's run method ends (or if the stop method, which is deprecated) is called. It cannot be restarted.
4. Methods of a Thread object can be called anytime as if it were just another normal object.

Except start() which can be called only once. Calling start() creates a new thread of execution.
5. A thread spawned by a daemon thread is a daemon thread but you can change it by calling setDaemon(false).
6. A thread can be made a daemon thread by calling setDaemon(true) method. This method must be called before the thread is started, otherwise an IllegalThreadStateException will be thrown.
7. Threads have priorities. Thread class defines the int constants MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY. Their values are 10, 0 and 5 but you should use the constant names instead of the values.
8. A newly created thread has the same priority as the thread which created it. You can change it by calling setPriority().
9. Which thread is scheduled when depends on the JVM and platform. So, you can NEVER say for sure about which thread would be running at at what time. Ie. If you start 2 threads you can't say anything about their execution schedule. And your code should not depend on any such assumptions.
10. wait() and sleep() must be enclosed in a try/catch block as they throw InterruptedException.
11. A thread can obtain multiple locks on the same object or multiple objects. If a thread acquires a lock for the same object twice, it should release it twice otherwise the object will remain locked.
12. A thread owning the lock of an object can call other synchronous methods on the same object. In a sense, it is acquiring the same lock more than once.
13. Synchronized methods can be overridden to be non-synchronized. But it does not change the behavior for the super class's method.
14. Beware of deadlock: Consider this situation: Thread1 gets the lock for object1 and tries to acquire the lock for object2. Just before it could get the lock for obj2, the OS preempts this thread and runs another thread t2. Now t2 gets the lock for obj2 (which was available as T1 was stopped just before it could acquire the lock) and then tries to get the lock for Object1 (which was already acquired by T1). Here, you can see that T1 is waiting for obj2's lock which is acquired by T2, and T2 is waiting for obj1's lock which is accuired by T1. Neither of the threads is able to proceed. This is a Deadlock.
15. Java does not provide any mechanism to detect, avoid or solve a deadlock. You must program so that deadlocks don't happen at runtime.