Managing the Java Thread Lifecycle: State Machine for Java Threads

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

• Be aware of the Java thread lifecycle

• Understand the various states in the Java thread lifecycle

![Diagram of Java thread lifecycle]

- New: new MyThread()
- Runnable: myThread.start() (wait-time elapsed)
- Blocked: attempt to access guarded resource
- Waiting: cond.notify(), cond.notifyAll(), cond.wait()
- Running: run() method returns
- Timed Waiting: myThread.sleep() wait(timeout) join(timeout)
- Terminated: run() method returns
The State Machine for Java Threads
The State Machine for the Java Thread Lifecycle

- A Java thread can be in various states (one at a time) during its lifecycle

```
Enum Thread.State
    java.lang.Object
        java.lang.Enum<Thread.State>
            java.lang.Thread.State

All Implemented Interfaces:
    Serializable, Comparable<Thread.State>

Enclosing class:
    Thread

public static enum Thread.State
    extends Enum<Thread.State>

A thread state. A thread can be in one of the following states:
- NEW
    A thread that has not yet started is in this state.
- RUNNABLE
    A thread executing in the Java virtual machine is in this state.
- BLOCKED
    A thread that is blocked waiting for a monitor lock is in this state.
- WAITING
    A thread that is waiting indefinitely for another thread to perform a particular action is in this state.
- TIMED_WAITING
    A thread that is waiting for another thread to perform an action for up to a specified waiting time is in this state.
- TERMINATED
    A thread that has exited is in this state.

A thread can be in only one state at a given point in time. These states are virtual machine states which do not reflect any operating system thread states.
```

See docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html
The State Machine for the Java Thread Lifecycle

- **New**: new MyThread()
- **Runnable**: myThread.start()
- **Blocked**: resource obtained
- **Waiting**: attempt to access guarded resource
- **Runnable**: cond.notify(), cond.notifyAll()
- **Waiting**: cond.wait()
- **Terminated**: run() method returns
- **Scheduler**: run() method returns
- **Timed Waiting**: myThread.sleep()
- **Timed Waiting**: wait(timeout)
- **Timed Waiting**: join(timeout)

See [www.uml-diagrams.org/examples/java-6-thread-state-machine-diagram-example.html](http://www.uml-diagrams.org/examples/java-6-thread-state-machine-diagram-example.html)
Begin by creating a new thread object

The State Machine for the Java Thread Lifecycle
The State Machine for the Java Thread Lifecycle

Transition to the “New” state
The State Machine for the Java Thread Lifecycle

Call start() to launch the thread
The State Machine for the Java Thread Lifecycle

Transition to the "Runnable" state
The State Machine for the Java Thread Lifecycle

The Java & Android Linux thread scheduler controls what happens next since there may be multiple threads waiting for their chance to run.
The State Machine for the Java Thread Lifecycle

When the scheduler selects a thread to execute it transitions to the "Running" state.
The Java execution environment (e.g., JVM, Dalvik, ART, etc.) then invokes the thread’s run() hook method.
A thread can call various methods that cause it to wait for a period of time, which suspends the thread.
The State Machine for the Java Thread Lifecycle

Transition to the "Timed Waiting" state

- New
- Runnable
- Blocked
- Waiting
- Running
- Timed Waiting

Transition paths:
- New → Runnable
- Runnable → Blocked
- Runnable → Waiting
- Runnable → Running
- Runnable → Timed Waiting
- Blocked → Runnable
- Waiting → Runnable
- Waiting → Timed Waiting
- Running → Timed Waiting
- Timed Waiting → Terminated

Methods:
- myThread.sleep()
- wait(timeout)
- join(timeout)
The State Machine for the Java Thread Lifecycle

- New
- Runnable
- Blocked
- Waiting
- Running
- Timed Waiting
- Terminated

The wait time elapses or the operation completes
Transition to the “Runnable” state (i.e., it doesn’t start to run immediately)
The State Machine for the Java Thread Lifecycle

When the scheduler selects a thread to execute it transitions to the “Running” state.
The Java execution environment then resumes executing the method the thread was running when it was suspended.
A thread will block (which suspends the thread) when it attempts to obtain a “guarded resource” (e.g., a monitor’s intrinsic lock) owned by another thread.
The State Machine for the Java Thread Lifecycle

Transition to the “Blocked” state

Attempt to obtain a guarded resource

Transition to the “Blocked” state
The State Machine for the Java Thread Lifecycle

The blocked thread will become unblocked when the resource is released by its current owner thread & the blocked thread acquires the resource.
Transition to the “Runnable” state (i.e., it doesn’t start to run immediately)
Ironically, the thread state for blocking I/O is “Runnable,” as discussed in stackoverflow.com/questions/19981726/java-thread-blocked-status
The State Machine for the Java Thread Lifecycle

When the scheduler selects a thread to execute it transitions to the “Running” state
The Java execution environment then resumes executing the method the thread was running when it was suspended.
A thread may call `wait()` on its monitor condition (the monitor lock must have already been acquired), which suspends the thread.
The State Machine for the Java Thread Lifecycle

Transition to the “Waiting” state
The State Machine for the Java Thread Lifecycle

When another thread calls notify() or notifyAll() waiting thread will be released (though it may need to transition to the “Blocked” state to reacquire the lock)
The State Machine for the Java Thread Lifecycle

Transition to the “Runnable” state (i.e., it doesn’t start to run immediately)
The State Machine for the Java Thread Lifecycle

Transition to the “Running” state
The State Machine for the Java Thread Lifecycle

The Java execution environment then resumes executing the method the thread was running when it was suspended.
The run() method can exit either normally (by “falling off the end” of run()) or via an unhandled exception.
The State Machine for the Java Thread Lifecycle

Transition to the “Terminated” state
The State Machine for the Java Thread Lifecycle

The Java execution environment can then reclaim the thread’s resources
End of Managing the Java Thread Lifecycle: State Machine for Java Threads