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
The State Machine for Java Threads
A Java thread can be in various states (one at a time) during its lifecycle.

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.
The State Machine for the Java Thread Lifecycle

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)
The State Machine for the Java Thread Lifecycle

Begin by creating a new thread object

```java
new MyThread()
```
The State Machine for the Java Thread Lifecycle

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

Runtime thread stack

New

Runnable

Blocked

Waiting

Passed

Timed Waiting

Scheduled

Runnable

Runnable

Running

Terminated

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

Transitions 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.
When the scheduler selects a thread to execute it transitions to the “Running” state.
The State Machine for the Java Thread Lifecycle

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

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

The wait time elapses or the operation completes
Transitions 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 tries to access a “guarded resource” (e.g., a monitor lock) in use by another thread.
The State Machine for the Java Thread Lifecycle

Transitions to the “Blocked” state

- Attempt to access guarded resource
- New
- Runnable
- Scheduler
- Running
- Timed Waiting
- Waiting
- Terminated
The State Machine for the Java Thread Lifecycle

When the resource is released by the other thread the blocked thread will acquire it & become unblocked
Transitions 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](https://stackoverflow.com/questions/19981726/java-thread-blocked-status)
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

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

Transitions to the “Running” state
The Java execution environment then resumes executing the method the thread was running when it was suspended.
The State Machine for the Java Thread Lifecycle

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

Transitions to the “Terminated” state
The Java execution environment can then reclaim the thread’s resources