Managing the Java Thread Lifecycle

(Part 1)

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

- Understand the Java thread lifecycle & its various states
Learning Objectives in this Part of the Lesson

• Understand the Java thread lifecycle & its various states

The primarily focus of this overall lesson is on starting & stopping Java threads
Overview of the States in the Java Thread Lifecycle
Overview of the Java Thread Lifecycle

• A Java thread is a complex entity that interacts with many other entities.

See docs.oracle.com/javase/8/docs/api/java/lang/Thread.html
Overview of the Java Thread Lifecycle

- The lifecycle of a thread must therefore be managed carefully
Overview of the Java Thread Lifecycle

- The lifecycle of a thread must therefore be managed carefully

A thread transitions through various “states” based on operations performed by a program.
Overview of the Java Thread Lifecycle

- The lifecycle of a thread must therefore be managed carefully

Two of the most fundamental parts of a Java thread’s lifecycle involve starting & stopping it
Overview of the Java Thread Lifecycle

• You needn’t understand all the details of Java thread internals to program them effectively.

However, your reputation (& value) as a “full stack” developer will increase if you learn how threads work internally.
The State Machine for Java Threads
new MyThread()
myThread.start()

Blocked
attempt to access guarded resource

Waiting
cond.notify(), cond.notifyAll()
cond.wait()

Runnable
cond.wait()
run()

Running
run() method returns

Timed Waiting
myThread.sleep()
wait(timeout)
join(timeout)

Terminated

New

resource obtained

Scheduler

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

See 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

Transitions 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

Transitions to the “Runnable” state
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 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

Runnable

Blocked

Waiting

Terminated

New

Scheduler

Running

Timed Waiting

myThread.sleep()
wait(timeout)
join(timeout)
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

Runnable

Blocked

Waiting

Scheduled

Running

Timed

Waiting

Terminated
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)
The State Machine for the Java Thread Lifecycle

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

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

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()` the thread that is waiting will be released.
The State Machine for the Java Thread Lifecycle

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 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 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
End of Managing the Java Thread Lifecycle (Part 1)
Learning Objectives in this Part of the Lesson

- Understand the Java thread lifecycle & its various states
- Recognize the steps involved in starting a Java thread
Steps Involved in Starting a Java Thread
Steps Involved in Starting a Java Thread

• Starting a Java thread involves interesting design & implementation issues
Steps Involved in Starting a Java Thread

• Calling `start()` on a thread triggers the execution of its `run()` hook method.
Steps Involved in Starting a Java Thread

- Many steps occur at various layers of Java class libraries, Java execution environment, & operating system.

Different versions of Android & Java will implement these layers differently, though the key levels of abstraction are often similar.
Steps Involved in Starting a Java Thread

- Many steps occur at various layers of Java class libraries, Java execution environment, & operating system

See [github.com/douglasraigschmidt/POSA/wiki/Source-Code](github.com/douglasraigschmidt/POSA/wiki/Source-Code)
Steps Involved in Starting a Java Thread

- Many steps occur at various layers of Java class libraries, Java execution environment, & operating system.

It’s important to realize that creating & starting new threads consumes a non-trivial amount of system resources, so use them judiciously!
Steps Involved in Starting a Java Thread

- Many steps occur at various layers of Java class libraries, Java execution environment, & operating system

1. `MyThread.start()`
Steps Involved in Starting a Java Thread

• Many steps occur at various layers of Java class libraries, Java execution environment, & operating system

1. `MyThread.start()`
2. `Thread.start()` // Java method

See `libcore/luni/src/main/java/java/lang/Thread.java`
Steps Involved in Starting a Java Thread

• Many steps occur at various layers of Java class libraries, Java execution environment, & operating system

1. MyThread.start()
2. Thread.start()
3. VMThread.create() // Native method

See libcore/luni/src/main/java/java/lang/VMThread.java
Steps Involved in Starting a Java Thread

- Many steps occur at various layers of Java class libraries, Java execution environment, & operating system

1. `MyThread.start()`
2. `Thread.start()`
3. `VMThread.create()`
4. `Dalvik_java_lang_VMThread_create()`  
   // JNI method

See [dalvik/vm/native/java_lang_VMThread.cpp](dalvik/vm/native/java_lang_VMThread.cpp)
Steps Involved in Starting a Java Thread

- Many steps occur at various layers of Java class libraries, Java execution environment, & operating system

1. MyThread.start()
2. Thread.start()
3. VMThread.create()
4. Dalvik_java_lang_VMThread_create()
5. dvmCreateInterpThread() // Dalvik method

See [dalvik/vm/Thread.cpp](dalvik/vm/Thread.cpp)
Steps Involved in Starting a Java Thread

• Many steps occur at various layers of Java class libraries, Java execution environment, & operating system

1. MyThread.start()
2. Thread.start()
3. VMThread.create()
4. Dalvik_java_lang_VMThread_create()
5. dvmCreateInterpThread()
6. pthread_create(..., interpThreadStart)
   // Pthreads method

See bionic/libc/bionic/pthread.c
Steps Involved in Starting a Java Thread

- Many steps occur at various layers of Java class libraries, Java execution environment, & operating system

1. MyThread.start()
2. Thread.start()
3. VMThread.create()
4. Dalvik_java_lang_VMThread_create()
5. dvmCreateInterpThread()
6. pthread_create(..., interpThreadStart)
   // Pthreads method

This is the entry point function used to transition between C & Java code
Steps Involved in Starting a Java Thread

• Many steps occur at various layers of Java class libraries, Java execution environment, & operating system

1. MyThread.start()
2. Thread.start()
3. VMThread.create()
4. Dalvik_java_lang_VMThread_create()
5. dvmCreateInterpThread()
6. pthread_create(..., interpThreadStart)
7. Android Linux kernel...

See source.android.com/source/building-kernels.html
Steps Involved in Starting a Java Thread

- Many steps occur at various layers of Java class libraries, Java execution environment, & operating system

1. MyThread.start()
2. Thread.start()
3. VMThread.create()
4. Dalvik_java_lang_VMThread_create()
5. dvmCreateInterpThread()
6. pthread_create(..., interpThreadStart)
7. Android Linux kernel...
8. interpThreadStart(void* arg) // Adapter

See dalvik/vm/Thread.cpp
Steps Involved in Starting a Java Thread

- Many steps occur at various layers of Java class libraries, Java execution environment, & operating system

1. MyThread.start()
2. Thread.start()
3. VMThread.create()
4. Dalvik_java_lang_VMThread_create()
5. dvmCreateInterpThread()
6. pthread_create(..., interpThreadStart)
7. Android Linux kernel...
8. interpThreadStart(void* arg)
9. dvmCallMethod(self, run, self->threadObj)
   // Dalvik method

See dalvik/vm/interp/Stack.cpp
Steps Involved in Starting a Java Thread

• Many steps occur at various layers of Java class libraries, Java execution environment, & operating system

1. MyThread.start()
2. Thread.start()
3. VMThread.create()
4. Dalvik_java_lang_VMThread_create()
5. dvmCreateInterpThread()
6. pthread_create(..., interpThreadStart)
7. Android Linux kernel...
8. interpThreadStart(void* arg)
9. dvmCallMethod(self, run, self->threadObj)
10. MyThread.run() // User-defined hook method
End of Managing the Java Thread Lifecycle (Part 2)