Managing the Java Thread Lifecycle: Overview of Stopping a Java Thread

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

• Know various ways to stop Java threads
Overview of Stopping a Java Thread
Overview of Stopping a Java Thread

• It may be necessary to stop a Java thread for various reasons
Overview of Stopping a Java Thread

• It may be necessary to stop a Java thread for various reasons, e.g.
  • Users may want to cancel a long-running operation
  • e.g., they get bored or tired of waiting for it to complete
Overview of Stopping a Java Thread

• It may be necessary to stop a Java thread for various reasons, e.g.
  • Users may want to cancel a long-running operation
  • Other “speculative computations” should be cancelled after first result is found or a timeout elapses
    • e.g., The ExecutorService invokeAny() method cancels other threads after a result is found or time expires

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#invokeAny
Overview of Stopping a Java Thread

- It may be necessary to stop a Java thread for various reasons, e.g.
  - Users may want to cancel a long-running operation
  - Other “speculative computations” should be cancelled after first result is found or a timeout elapses
  - In response to errors encountered during processing that require an app to shutdown
    - e.g., if a disk fills up during a web crawl
Overview of Stopping a Java Thread

- It may be necessary to stop a Java thread for various reasons, e.g.
  - Users may want to cancel a long-running operation
  - Other “speculative computations” should be cancelled after first result is found or a timeout elapses
  - In response to errors encountered during processing that require an app to shutdown
  - An app or activity is destroyed, stopped, or paused
    - e.g., due to runtime configuration changes or pressing the “back” button

The GCD Concurrent app contains an (intentional) design flaw where it “leaks” threads when an orientation change occurs

See github.com/douglasraigschmidt/POSA/tree/master/ex/M3/GCD/Concurrent
Overview of Stopping a Java Thread

• Stopping Java threads is surprisingly hard
Overview of Stopping a Java Thread

• Stopping Java threads is surprisingly hard
• i.e., the “Sorcerer’s Apprentice” problem

See www.youtube.com/watch?v=5rzyuY8-Ao8
Overview of Stopping a Java Thread

• There’s no safe way to stop a Java thread involuntarily

See docs.oracle.com/javase/8/docs/technotes/guides/concurrency/threadPrimitiveDeprecation.html
Overview of Stopping a Java Thread

• There’s no safe way to stop a Java thread involuntarily
  • The stop() method is deprecated since it’s inherently unsafe

See geekexplains.blogspot.com/2008/07/why-stop-suspend-resume-of-thread-are.html
Overview of Stopping a Java Thread

- There’s no safe way to stop a Java thread involuntarily
- The stop() method is deprecated since it’s inherently unsafe, e.g.
  - All locked monitors are unlocked as the exception propagates up the stack
Overview of Stopping a Java Thread

• There’s no safe way to stop a Java thread involuntarily
  • The stop() method is deprecated since it’s inherently unsafe, e.g.
    • All locked monitors are unlocked as the exception propagates up the stack
    • Any objects protected by these monitors are thus left in an inconsistent state

Call Stack

method1 ()

method2 () throws IOException {…}

calls

method3 () throws IOException {…}

calls

method4 () throws IOException {…}

calls
Overview of Stopping a Java Thread

• There’s no safe way to stop a Java thread involuntarily
  • The stop() method is deprecated since it’s inherently unsafe, e.g.
    • All locked monitors are unlocked as the exception propagates up the stack
    • Any objects protected by these monitors are thus left in an inconsistent state
  • There is no way for an object’s methods to control when stop() takes effect..

Call Stack

```
method1()

  ↓ calls

method2() throws IOException {...}

  ↓ calls

method3() throws IOException {...}

  ↓ calls

method4() throws IOException {...}
```
Overview of Stopping a Java Thread

- Long running operations in a thread must be coded to stop *voluntarily*!

```java
public void run(){
    while (true) {
        // Check if thread
        // should stop
    }
}
```
Overview of Stopping a Java Thread

• There are two ways to stop a Java thread voluntarily
Overview of Stopping a Java Thread

• There are two ways to stop a Java thread voluntarily
• Use a volatile flag

```
public class MyRunnable implements Runnable {
    private volatile boolean mIsStopped = false;

    public void stopMe() {
        mIsStopped = true;
    }

    public void run() {
        while(mIsStopped != true) {
            // a long-running operation
        }
    }

    // ...

See en.wikipedia.org/wiki/Volatile_variable#In_Java
```
Overview of Stopping a Java Thread

- There are two ways to stop a Java thread voluntarily
  - Use a volatile flag
  - Use Java thread interrupt requests

Interrupts

An *interrupt* is an indication to a thread that it should stop what it is doing and do something else. It's up to the programmer to decide exactly how a thread responds to an interrupt, but it is very common for the thread to terminate. This is the usage emphasized in this lesson.

A thread sends an interrupt by invoking `interrupt` on the `Thread` object for the thread to be interrupted. For the interrupt mechanism to work correctly, the interrupted thread must support its own interruption.

See [docs.oracle.com/javase/tutorial/essential/concurrency/interrupt.html](docs.oracle.com/javase/tutorial/essential/concurrency/interrupt.html)
Overview of Stopping a Java Thread

- Stopping a Java thread voluntarily requires cooperation between threads
Overview of Stopping a Java Thread

- Stopping a Java thread voluntarily requires cooperation between threads
- A thread should rarely be stopped immediately since shared data could be left in an inconsistent state
Overview of Stopping a Java Thread

- Stopping a Java thread voluntarily requires cooperation between threads
  - A thread should rarely be stopped immediately since shared data could be left in an inconsistent state
  - A thread must there check periodically to see if it has been told to stop
Overview of Stopping a Java Thread

- Stopping a Java thread voluntarily requires cooperation between threads
  - A thread should rarely be stopped immediately since shared data could be left in an inconsistent state
  - A thread must there check periodically to see if it has been told to stop
  - Thread interrupts are fragile since they require all parts of a program follow consistent usage patterns

See weblogs.java.net/blog/2009/03/02/cancelling-tasks-threadinterrupt-fragility
Overview of Stopping a Java Thread

• Stopping a Java thread voluntarily requires cooperation between threads
  • A thread should rarely be stopped immediately since shared data could be left in an inconsistent state
  • A thread must there check periodically to see if it has been told to stop
  • Thread interrupts are fragile since they require all parts of a program follow consistent usage patterns
• Voluntary checking is tedious & error-prone, but it’s the only way to halt Java threads reliably

See stackoverflow.com/questions/8505707/android-best-and-safe-way-to-stop-thread
Managing the Java Thread Lifecycle: Overview of Stopping a Java Thread