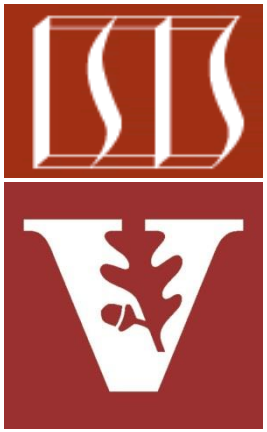
Managing the Java Thread Lifecycle: Stopping a Thread via Volatile Flag



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
  - Stopping a thread with a volatile flag

Stopping a Java Thread via a Volatile Flag

Stopping a Java Thread via a Volatile Flag

- One way to stop a Java thread is to use a "stop" flag

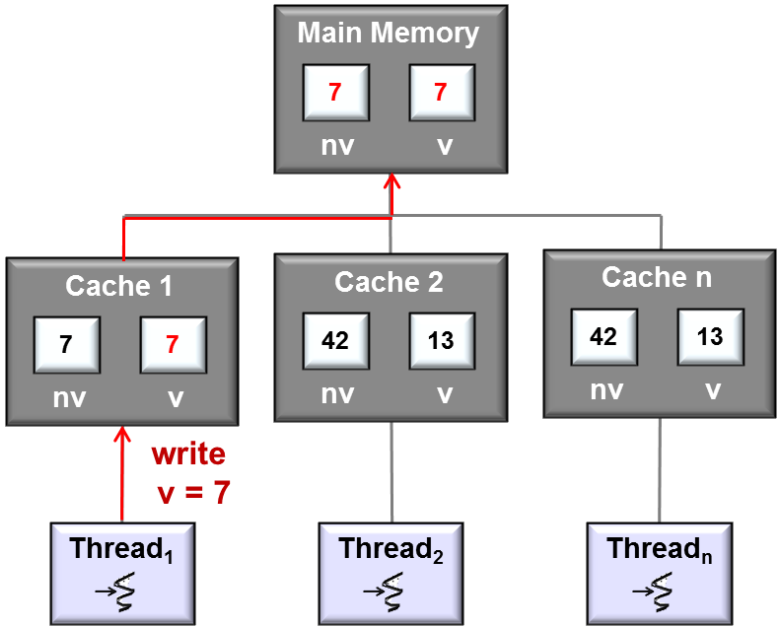
Stopping a Java Thread via a Volatile Flag

- One way to stop a Java thread is to use a "stop" flag, e.g.
  - Add a volatile boolean flag "mIsStopped" to a class

```
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
      ...
```

Stopping a Java Thread via a Volatile Flag

- One way to stop a Java thread is to use a "stop" flag, e.g.
  - Add a volatile boolean flag "mIsStopped" to a class
    - Ensures changes to a variable are consistent & visible to other threads atomically




```
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

Stopping a Java Thread via a Volatile Flag

- One way to stop a Java thread is to use a "stop" flag, e.g.
  - Add a volatile boolean flag "mIsStopped" to a class
  - Add a stopMe() method that sets "mIsStopped" to true

```
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
      ...
```

Stopping a Java Thread via a Volatile Flag

- One way to stop a Java thread is to use a "stop" flag, e.g.
  - Add a volatile boolean flag "mIsStopped" to a class
  - Add a stopMe() method that sets "mIsStopped" to true

```
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
      ...
```

volatile basically defines a "critical section" for the mIsStopped field

However, reads & writes to a volatile variable "spin" rather than "block" threads

Stopping a Java Thread via a Volatile Flag

- One way to stop a Java thread is to use a "stop" flag, e.g.
  - Add a volatile boolean flag "mIsStopped" to a class
  - Add a stopMe() method that sets "mIsStopped" to true
  - Check "mIsStopped" periodically to see if thread's been stopped

```
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
      ...
```

Stopping a Java Thread via a Volatile Flag

- One way to stop a Java thread is to use a "stop" flag, e.g.
  - Add a volatile boolean flag "mIsStopped" to a class
  - Add a stopMe() method that sets "mIsStopped" to true
  - Check "mIsStopped" periodically to see if thread's been stopped
    - Return from the run() method when the thread's been stopped

```
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
      ...
    }
    return;
    ...
```

Stopping a Java Thread via a Volatile Flag

Pros

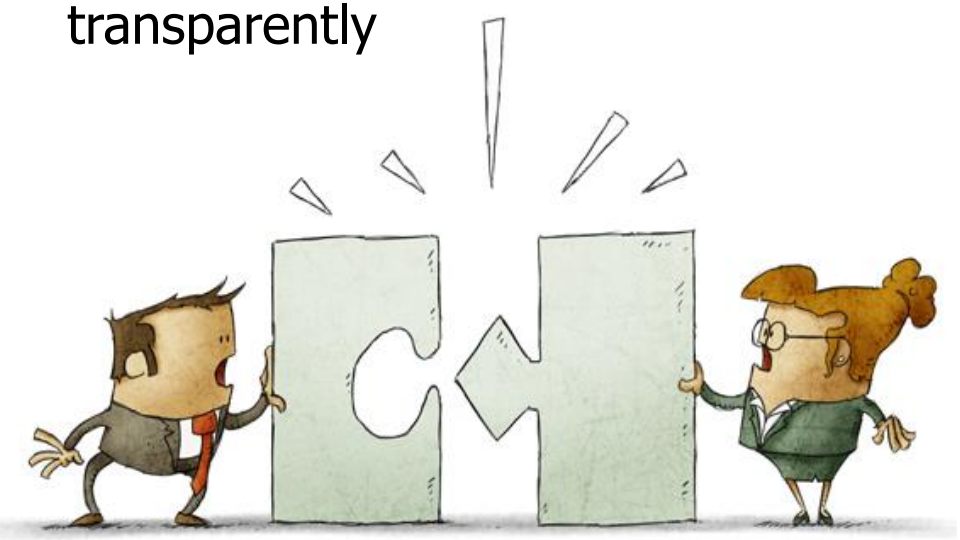
- Using a volatile flag is lightweight

```
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
      ...
    }
    return;
    ...
```

Stopping a Java Thread via a Volatile Flag

Cons

- A volatile flag isn't integrated into the Java execution environment transparently



```
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
      ...
      wait();
      ...
```

This built-in Java method knows nothing about our volatile flag!!

Stopping a Java Thread via a Volatile Flag

Cons

- A volatile flag isn't integrated into the Java execution environment transparently
  - e.g., blocking operations won't be awakened, which impedes shut down processing

```
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
      ...
      wait();
      ...
```

Managing the Java Thread Lifecycle: Stopping a Java Thread via a Volatile Flag