

Managing the Java Thread Lifecycle: Stopping a Thread via an Interrupt



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 thread with an interrupt request



Stopping Java Threads with an Interrupt Request

Stopping Java Threads with an Interrupt Request

- A thread can be stopped voluntarily by calling its `interrupt()` method



See docs.oracle.com/javase/8/docs/api/java/lang/Thread.html#interrupt

Stopping Java Threads with an Interrupt Request

- A thread can be stopped voluntarily by calling its `interrupt()` method
- Posts an *interrupt request* to a thread

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

Stopping Java Threads with an Interrupt Request

- A thread can be stopped voluntarily by calling its `interrupt()` method
 - Posts an *interrupt request* to a thread
- Interrupts are implemented via an internal *interrupt status* flag



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.

Stopping Java Threads with an Interrupt Request

- A thread can be stopped voluntarily by calling its `interrupt()` method
 - Posts an *interrupt request* to a thread
- Interrupts are implemented via an internal *interrupt status* flag
 - Invoking `Thread.interrupt()` sets this flag

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.

Stopping Java Threads with an Interrupt Request

- A thread can be stopped voluntarily by calling its `interrupt()` method
 - Posts an *interrupt request* to a thread
- `Interrupts` is implemented via an internal *interrupt status* flag
 - Invoking `Thread.interrupt()` sets this flag
 - Programs can check this flag via two thread accessor methods

static boolean	<u>interrupted()</u> – Tests whether the current thread has been interrupted
boolean	<u>isInterrupted()</u> – Tests whether this thread has been interrupted

Each method has different side-effects on interrupt status, as discussed shortly

Stopping Java Threads with an Interrupt Request

- Here's a simple Java program that starts, runs, & interrupts a background thread

```
static int main(String args[]) {  
    Thread t1 =  
        new Thread(() -> {  
            for (int i = 0;  
                i < args.length; i++) {  
                processBlocking(args[i]);  
                processNonBlocking(args[i]);  
            }  
        });  
  
    t1.start();  
    ... // Run concurrently for a while  
    t1.interrupt();  
    ...  
}
```

Stopping Java Threads with an Interrupt Request

- Here's a simple Java program that starts, runs, & interrupts a background thread

Create a new thread



```
static int main(String args[]) {  
    Thread t1 =  
        new Thread(() -> {  
            for (int i = 0;  
                i < args.length; i++) {  
                processBlocking(args[i]);  
                processNonBlocking(args[i]);  
            }  
        });  
  
    t1.start();  
    ... // Run concurrently for a while  
    t1.interrupt();  
    ...  
}
```

Stopping Java Threads with an Interrupt Request

- Here's a simple Java program that starts, runs, & interrupts a background thread

Start the new thread



```
static int main(String args[]) {  
    Thread t1 =  
        new Thread(() -> {  
            for (int i = 0;  
                i < args.length; i++) {  
                processBlocking(args[i]);  
                processNonBlocking(args[i]);  
            }  
        });  
  
    t1.start();  
    ... // Run concurrently for a while  
    t1.interrupt();  
    ...  
}
```

Stopping Java Threads with an Interrupt Request

- Here's a simple Java program that starts, runs, & interrupts a background thread

```
static int main(String args[]) {  
    Thread t1 =  
        new Thread(() -> {  
            for (int i = 0;  
                i < args.length; i++) {  
                processBlocking(args[i]);  
                processNonBlocking(args[i]);  
            }  
        });  
  
    t1.start();  
    ... // Run concurrently for a while  
    t1.interrupt();  
    ...  
}
```

The main thread continues running

Stopping Java Threads with an Interrupt Request

- Here's a simple Java program that starts, runs, & interrupts a background thread

After the thread starts, it runs this lambda expression, whose methods perform blocking & non-blocking computations

```
static int main(String args[]) {  
    Thread t1 =  
        new Thread(() -> {  
            for (int i = 0;  
                i < args.length; i++) {  
                processBlocking(args[i]);  
                processNonBlocking(args[i]);  
            }  
        });  
  
    t1.start();  
    ... // Run concurrently for a while  
    t1.interrupt();  
    ...  
}
```

Stopping Java Threads with an Interrupt Request

- Here's a simple Java program that starts, runs, & interrupts a background thread

```
static int main(String args[]) {  
    Thread t1 =  
        new Thread(() -> {  
            for (int i = 0;  
                i < args.length; i++) {  
                processBlocking(args[i]);  
                processNonBlocking(args[i]);  
            }  
        });  
  
    t1.start();  
    ... // Run concurrently for a while  
    t1.interrupt();  
    ...  
}
```


After the main thread performs some computations it interrupts thread t1

Stopping Java Threads with an Interrupt Request

- Here's a simple Java program that starts, runs, & interrupts a background thread

```
static int main(String args[]) {  
    Thread t1 =  
        new Thread(() -> {  
            for (int i = 0;  
                i < args.length; i++) {  
                processBlocking(args[i]);  
                processNonBlocking(args[i]);  
            }  
        });  
  
    t1.start();  
    ... // Run concurrently for a while  
    t1.interrupt();  
    ...  
}
```

*Methods running in thread t1
check periodically to see if the
thread's been stopped yet*



Stopping Java Threads with an Interrupt Request

- Certain blocking operations in the Java language & class libraries return automatically & throw `InterruptedException` if the thread is interrupted

```
void processBlocking(String args) {  
    ...  
    while (true) {  
        try {  
            Thread.currentThread().  
                sleep(interval);  
            synchronized(this) {  
                while (someConditionFalse)  
                    wait();  
            }  
        }  
        catch (InterruptedException e)  
        { ... }  
        ...  
    }  
}
```

e.g., wait(), join(), sleep() & blocking I/O calls on "interruptable channels"

Stopping Java Threads with an Interrupt Request

- Methods whose operations do not block must periodically check if `Thread.interrupt()` has been called

```
void processNonBlocking(String args) {  
    ...  
    while (true) {  
        ... // Long-running computation  
        if (Thread.interrupted())  
            throw  
                new InterruptedException();  
        ...  
    }  
}
```

interrupted() is a static method that returns true if the calling thread has its interrupt status flag set

`interrupted()` clears the current thread's interrupt status the first time it's called

Stopping Java Threads with an Interrupt Request

- Methods whose operations do not block must periodically check if `Thread.interrupt()` has been called

```
void processNonBlocking(String args) {  
    ...  
    while (true) {  
        ... // Long-running computation  
        if (Thread.interrupted())  
            throw  
                new InterruptedException();  
        ...  
    }  
}
```

This example explicitly throws an `InterruptedException`, which is created/treated like a normal object

Stopping Java Threads with an Interrupt Request

- Methods whose operations do not block must periodically check if `Thread.interrupt()` has been called

```
void processNonBlocking(String args) {  
    ...  
    final myThread =  
        Thread.currentThread();  
  
    while (true) {  
        ... // Long-running computation  
        if (myThread.isInterrupted())  
            throw  
                new InterruptedException();  
        ...  
    }  
}
```

isInterrupted() is a non-static method that returns true if the designated thread has its interrupt status flag set

`isInterrupted()` can be called multiple times w/out affecting its *interrupt status*

Stopping Java Threads with an Interrupt Request

- Programs can override thread interrupt methods since they are virtual
- e.g., `interrupt()`, `interrupted()`, & `isInterrupted()`

```
public class BeingThread
    extends Thread {
    volatile boolean mInterrupted;

    BeingThread(Runnable runnable) {
        super(runnable);
        mInterrupted = false;
    }

    public void interrupt() {
        mInterrupted = true;
        super.interrupt();
    }

    public boolean isInterrupted() {
        return mInterrupted
            || super.isInterrupted()
    }
}
```

See stackoverflow.com/questions/23369891/overriding-interrupt-isinterrupted-method-in-thread-class

Stopping Java Threads with an Interrupt Request

- Programs can override thread interrupt methods since they are virtual
- e.g., `interrupt()`, `interrupted()`, & `isInterrupted()`

```
public class BeingThread
    extends Thread {
    volatile boolean mInterrupted;

    BeingThread(Runnable runnable) {
        super(runnable);
        mInterrupted = false;
    }

    public void interrupt() {
        mInterrupted = true;
        super.interrupt();
    }

    public boolean isInterrupted() {
        return mInterrupted
            || super.isInterrupted()
    }
}
```

But make sure you know what you're doing...

End of Managing the Java Thread Lifecycle: Stopping a Thread via an Interrupt