Managing the Java Thread Lifecycle:
Patterns of Handling Thread Interrupts

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
- Learn the patterns of handing Java thread interrupts
Patterns of Handling Java Thread Interrupts
Patterns of Handling Java Thread Interrupts

- Recall that blocking operations in Java can return automatically & throw InterruptedException if the thread is interrupted

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

See earlier part of the lesson on “Stopping a Thread via an Interrupt”
There are patterns for dealing w/Java InterruptedException

See www.ibm.com/developerworks/java/library/j-jtp05236/index.html?ca=drs-
Patterns of Handling Java Thread Interrupts

- There are patterns for dealing w/Java InterruptedException, e.g.
  - Propagate InterruptedException to callers by not catching it

```java
public class StringBlockingQueue {
    private BlockingQueue<String> queue = new LinkedBlockingQueue<String>();

    public void put(String s) throws InterruptedException {
        queue.put(s);
    }

    public String take() throws InterruptedException {
        return queue.take();
    }
}

See docs.oracle.com/javase/8/docs/api/java/lang/InterruptedException.html
Patterns of Handling Java Thread Interrupts

- There are patterns for dealing w/Java InterruptedException, e.g.
  - Propagate InterruptedException to callers by not catching it

```java
public class StringBlockingQueue {
    private BlockingQueue<String>
        queue = new LinkedBlockingQueue<String>();

    public void put(String s)
        throws InterruptedException {
        queue.put(s);
    }

    public String take()
        throws InterruptedException {
        return queue.take();
    }
}
```

The exception is explicitly listed in each method’s “throw clause”

See docs.oracle.com/javase/tutorial/essential/exceptions/declaring.html
Patterns of Handling Java Thread Interrupts

- There are patterns for dealing with Java InterruptedException, e.g.
- Propagate InterruptedException to callers by not catching it

```java
public class StringBlockingQueue {
    private BlockingQueue<String> queue = new LinkedBlockingQueue<String>();
    
    public void put(String s) throws InterruptedException {
        queue.put(s);
    }
    
    public String take() throws InterruptedException {
        return queue.take();
    }
}

StringBlockingQueue s = new StringBlockingQueue();
...
try {
    s.take();
    ...
} catch (InterruptedException e) {
    ...
}
```

It's now the caller's responsibility to handle the exception properly.
Patterns of Handling Java Thread Interrupts

• There are patterns for dealing w/Java InterruptedException, e.g.
  • Propagate InterruptedException to callers by not catching it
  • Perform task-specific cleanup before rethrowing

```java
if (mustWait) {
    try {
        lock.wait();
    } catch (InterruptedException e) {
        synchronized (this) {
            boolean removed = mWaitQueue.remove(lock);
            if (!removed) release();
        }
        throw e;
    }
}
...
```

Avoid leaking resources or leaving resources in an inconsistent state
Patterns of Handling Java Thread Interrupts

- There are patterns for dealing w/Java InterruptedException, e.g.
  - Propagate InterruptedException to callers by not catching it
  - Perform task-specific cleanup before rethrowing
  - Restore interrupted status after catching InterruptedException

```java
class Runnable {
    public void run() {
        // Code here
    }
}
```

```java
public void doWork() {
    try {
        while (true) {
            Runnable r = queue.take(10, SECONDS);
            r.run();
        }
    } catch (InterruptedException e) {
        ... Thread.currentThread().interrupt();
    }
}
```

Preserve evidence the exception occurred for use by higher levels of the call stack

See [daniel.mitterdorfer.name/articles/2015/handling-interruptedexception](daniel.mitterdorfer.name/articles/2015/handling-interruptedexception)
Patterns of Handling Java Thread Interrupts

• There are patterns for dealing w/Java InterruptedException, e.g.
  • Propagate InterruptedException to callers by not catching it
  • Perform task-specific cleanup before rethrowing
  • Restore interrupted status after catching InterruptedException
  • Handle interrupt & “swallow” it

```java
public boolean gaze() {
    try {
        int sleepTime = 1000 +
                      mRandom.nextInt(4000;

        Thread.sleep(sleepTime);
        return true;
    }
    catch (InterruptedException e) {
        return false;
    }
}
```
Patterns of Handling Java Thread Interrupts

- There are patterns for dealing w/Java InterruptedException, e.g.
  - Propagate InterruptedException to callers by not catching it
  - Perform task-specific cleanup before rethrowing
  - Restore interrupted status after catching InterruptedException
  - Handle interrupt & “swallow” it

```java
public boolean gaze() {
    try {
        int sleepTime = 1000 + mRandom.nextInt(4000);
        Thread.sleep(sleepTime);
        return true;
    }
    catch (InterruptedException e) {
        return false;
    }
}
```

*General-purpose reusable library code should *never* swallow interrupt requests entirely (i.e., this is an “anti-pattern”)*

*e.g., often done when the thread sleep() or join() methods are called*
End of Managing the Java Thread Lifecycle: Patterns of Handling Thread Interrupts