TPS02-J. Ensure that tasks submitted to a thread pool are interruptible

Programs may submit only tasks that support interruption using `Thread.interrupt()` to thread pools that require the ability to shut down the thread pool or to cancel individual tasks within the pool. Programs must not submit tasks that lack interruption support to such thread pools. According to the Java API [API 2014], the `java.util.concurrent.ExecutorService.shutdownNow()` method attempts to stop all actively executing tasks, halts the processing of waiting tasks, and returns a list of the tasks that were awaiting execution....

There are no guarantees beyond best-effort attempts to stop processing actively executing tasks. For example, typical implementations will cancel via `Thread.interrupt()`, so any task that fails to respond to interrupts may never terminate.

Noncompliant Code Example (Shutting Down Thread Pools)

This noncompliant code example submits the `SocketReader` class as a task to the thread pool declared in `PoolService`:
public final class SocketReader implements Runnable { // Thread-safe class
private final Socket socket;
private final BufferedReader in;
private final Object lock = new Object();

public SocketReader(String host, int port) throws IOException {
    this.socket = new Socket(host, port);
    this.in = new BufferedReader(
            new InputStreamReader(this.socket.getInputStream()));
}

// Only one thread can use the socket at a particular time
@Override public void run() {
    try {
        synchronized (lock) {
            readData();
        }
    } catch (IOException ie) {
        // Forward to handler
    }
}

public void readData() throws IOException {
    String string;
    try {
        while ((string = in.readLine()) != null) {
            // Blocks until end of stream (null)
        }
    } finally {
        shutdown();
    }
}

public void shutdown() throws IOException {
    socket.close();
}
}

public final class PoolService {
private final ExecutorService pool;

public PoolService(int poolSize) {
    pool = Executors.newFixedThreadPool(poolSize);
}

public void doSomething() throws InterruptedException, IOException {
    pool.submit(new SocketReader("somehost", 8080));
    // ...
    List<Runnable> awaitingTasks = pool.shutdownNow();
}

public static void main(String[] args)
    throws InterruptedException, IOException {
    PoolService service = new PoolService(5);
    service.doSomething();
}
}

The shutdownNow() method may fail to shut down the thread pool because the task lacks support for interruption using the Thread.interrupt() method and because the shutdown() method must wait until all executing tasks have finished.

Similarly, tasks that use some mechanism other than Thread.interrupted() to determine when to shut down will be unresponsive to shutdown() and shutdownNow(). For instance, tasks that check a volatile flag to determine whether it is safe to shutdown are unresponsive to these methods. THIO5-J. Do not use Thread.stop() to terminate threads provides more information on using a flag to terminate threads.

Compliant Solution (Submit Interruptible Tasks)
This compliant solution defines an interruptible version of the `SocketReader` class, which is instantiated and submitted to the thread pool:

```java
public final class SocketReader implements Runnable {
    private final SocketChannel sc;
    private final Object lock = new Object();

    public SocketReader(String host, int port) throws IOException {
        sc = SocketChannel.open(new InetSocketAddress(host, port));
    }

    @Override public void run() {
        ByteBuffer buf = ByteBuffer.allocate(1024);
        try {
            synchronized (lock) {
                while (!Thread.interrupted()) {
                    sc.read(buf);
                    // ...
                }
            }
            catch (IOException ie) {
                // Forward to handler
            }
        }
    }
}
```

### Exceptions

**TPS02-J-EX0**: Short-running tasks that execute without blocking are exempt from this rule.

### Risk Assessment

Submitting tasks that are uninterruptible may prevent a thread pool from shutting down and consequently may cause **DoS**.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Severity</th>
<th>Likelihood</th>
<th>Remediation Cost</th>
<th>Priority</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPS02-J</td>
<td>Low</td>
<td>Probable</td>
<td>Medium</td>
<td>P4</td>
<td>L3</td>
</tr>
</tbody>
</table>

### Bibliography

- [API 2014](#) Interface `ExecutorService`
- [Goetz 2006a](#) Chapter 7, “Cancellation and Shutdown”