The Java Specialists' Newsletter
Issue 153 ➤ 2007-11-25 ➤ Category: **Tips and Tricks** ➤ Java version: 5+

Timeout on Console Input

by Dr. Heinz M. Kabutz

Abstract:
In this newsletter, we look at how we can read from the console input stream, timing out if we do not get a response by some timeout.

Welcome to the 153rd issue of **The Java(tm) Specialists' Newsletter**. Last week I presented my first Java Specialist **Master** Course (**http://www.javaspecialists.eu/courses/master.jsp**), so I would like to give some feedback on what happened. It is hard to describe the feelings that I had when I was teaching it. At first I was quite nervous about the pace, since I have never taught so much advanced information in such a short period of time. But then, after each day, instead of being drained, I felt refreshed and excited how it was panning out. I am now convinced that this is the *most comprehensive advanced Java course* you will find *anywhere*. Just the section on serialization is 70 slides, without being repetitive and boring. If you are a *good* Java programmer, and you want to go further, **have a look at the outline**.

Last week I received a rather interesting email from Maximilian Eberl, who was trying to write a simple Java console application to receive input from nurses who look after elderly and handicapped people. I must admit that my first suspicion was that this was a school project, so I quizzed him as to its application. Due to the proliferation of plagiarism on the internet, I refuse to help anybody who I suspect of either asking for my assistance with their school work (that is what tutors and teachers are supposed to be for) or when I feel that they are trying to embellish their skill, without giving credit for my help to their bosses. However, after working out a solution, and reading Maximilian's detailed explanation of why he was trying to do this, I was convinced that this is not school project material. (Besides that, he has the same name as my son, so that worked in his favour as well :-) It was thus an absolute pleasure helping him solve this interesting problem.

NEW: Please see our new "Extreme Java" course, combining concurrency, a little bit of performance and Java 8. **Extreme Java - Concurrency & Performance for Java 8**.

Timeout on Console Input

The problem that Maximilian was trying to solve was to have a simple console based application that would give users a certain time to answer the question. It would then timeout and retry a few times.

The reason this was a difficult problem to solve was that `System.in` blocks on the `readLine()` method. The thread state is `RUNNABLE` when you are blocking on IO, not `WAITING` or `TIMED_WAITING`. It therefore does not respond to interruptions. Here is some sample code that shows the thread state of the Thread:

```
import java.io.*;

public class ReadLineTest {
  public static void main(String[] args) throws IOException {
    BufferedReader in = new BufferedReader(
        new InputStreamReader(System.in)
    );
    in.readLine();
  }
}
```

The thread dump clearly shows the state (I have stripped out unnecessary lines from the stack trace):

```
"main" prio=10 tid=0x08059000 nid=0x2e8a runnable
   java.lang.Thread.State: RUNNABLE
  at java.io.BufferedReader.readLine(BufferedReader.java:362)
  at ReadLineTest.main(ReadLineTest.java:8)
```

Since blocking reads cannot be interrupted with `Thread.interrupt()`, we traditionally stop them by closing the underlying stream. In our **Java Specialist Master Course**, one of the working examples is how to write a non-blocking server using Java NIO. (Told you it was a comprehensive course :-)) However, since System.in is a traditional stream, we cannot use non-blocking techniques. Also, we cannot close it, since that would close it for all

Course Links

Extreme Java - Concurrency and Performance for Java 8

Extreme Java - Advanced Topics for Java 8

Design Patterns

In-House Courses

readers.

One little method in the `BufferedStream` will be able to help us. We can call `BufferedStream.ready()`, which will only return true if the `readLine()` method can be called without blocking. This implies that the stream not only contains data, but also a newline character.

The first problem is therefore solved. However, if we read the input in a thread, we still need to find a way to get the `String` input back to the calling thread. The `ExecutorService` in Java 5 will work well here. We can implement `Callable` and return the String that was read. Unfortunately we need to poll until something has been entered. Currently we sleep for 200 milliseconds between checks, but we could probably make that much shorter if we want instant response. Since we are sleeping, thus putting the thread in the `TIMED_WAITING` state, we can interrupt this task at any time. One last catch was that we do not want to accept an empty line as a valid input.

```
import java.io.*;
import java.util.concurrent.Callable;

public class ConsoleInputReadTask implements Callable<String> {
  public String call() throws IOException {
    BufferedReader br = new BufferedReader(
        new InputStreamReader(System.in));
    System.out.println("ConsoleInputReadTask run() called.");
    String input;
    do {
      System.out.println("Please type something: ");
      try {
        // wait until we have data to complete a readLine()
        while (!br.ready()) {
          Thread.sleep(200);
        }
        input = br.readLine();
      } catch (InterruptedException e) {
        System.out.println("ConsoleInputReadTask() cancelled");
        return null;
      }
    } while ("".equals(input));
    System.out.println("Thank You for providing input!");
    return input;
  }
}
```

The next task is to call the `ConsoleInputReadTask` and timeout after some time. We do that by calling `get()` on the `Future` that is returned by the `submit()` method on `ExecutorService`.

```
import java.util.concurrent.*;

public class ConsoleInput {
  private final int tries;
  private final int timeout;
  private final TimeUnit unit;

  public ConsoleInput(int tries, int timeout, TimeUnit unit) {
    this.tries = tries;
    this.timeout = timeout;
    this.unit = unit;
  }

  public String readLine() throws InterruptedException {
    ExecutorService ex = Executors.newSingleThreadExecutor();
    String input = null;
    try {
      // start working
      for (int i = 0; i < tries; i++) {
        System.out.println(String.valueOf(i + 1) + ". loop");
        Future<String> result = ex.submit(
            new ConsoleInputReadTask());
        try {
          input = result.get(timeout, unit);
          break;
        } catch (ExecutionException e) {
          e.getCause().printStackTrace();
        } catch (TimeoutException e) {
          System.out.println("Cancelling reading task");
          result.cancel(true);
          System.out.println("\nThread cancelled. input is null");
        }
      }
    } finally {
      ex.shutdownNow();
    }
    return input;
  }
}
```

We can put all this to the test with a little test class. It takes the number of tries and the timeout in seconds from the command line and instantiates that `ConsoleInput` class, reading from it and displaying the String:

```
import java.util.concurrent.TimeUnit;

public class ConsoleInputTest {
  public static void main(String[] args)
      throws InterruptedException {
    if (args.length != 2) {
      System.out.println(
          "Usage: java ConsoleInputTest <number of tries> " +
             "<timeout in seconds>");
```

```
      System.exit(0);
    }

    ConsoleInput con = new ConsoleInput(
        Integer.parseInt(args[0]),
        Integer.parseInt(args[1]),
        TimeUnit.SECONDS
    );

    String input = con.readLine();
    System.out.println("Done. Your input was: " + input);
  }
}
```

This seems to satisfy all the requirements that we were trying to fulfill. To be honest, when I first saw the problem, I did not think it could be done.

There is at least one way you could potentially get this program to fail. If you call the `ConsoleInput.readLine()` method from more than one thread, you run the very real risk of a data race between the `ready()` and `readLine()` methods. You would then block on the `BufferedReader.readLine()` method, thus potentially never completing.

Hopefully the rest of the program will be straighforward and soon the nurses for the disabled and elderly people will have some computer assistance, thanks to Maximilian Eberl. Good luck!

Kind regards from an airport somewhere in Europe :-)

Heinz

Tips and Tricks Articles Related Java Course

 Sitemap