Exercises week 7
Friday 9 October 2015

Goal of the exercises

The goal of this week's exercises is for you to show that you can write responsive user interfaces using threads and make them work correctly.

Due to the fall break, the handin deadline for these exercises is Thursday 22 October 2015. Also, due to the Algorithm Design exam on 21 October, which presumably occupy many of you, the workload is modest.

Do this first

Get and unpack this week's example code in zip file pcpp-week07.zip on the course homepage.

Exercise 7.1 File TestFetchWebGui.java contains a simple Java Swing user interface to initiate the fetching of some web pages and then report their sizes.

As implemented, the program uses a single SwingWorker subclass instance to fetch all the web pages sequentially, which is slow because each download has to complete before the next one starts. In this exercise you must change it so that it initiates multiple downloads at the same time, and prints the results as they become available.

1. Implement concurrent download. You can ignore the cancellation button and progress bar for now. There seems to be two ways to implement concurrent download of N webpages. Either (1) create N SwingWorker subclass instances that each downloads a single webpage; or (2) create a single SwingWorker subclass instance that itself uses Java's executor framework to download the N web pages concurrently. Approach (1) seems more elegant because it uses the SwingWorker executor framework only, instead of using two executor frameworks. Also, approach (2) seems dubious unless it is clear that a SwingWorker's `publish` method can be safely called on multiple threads; what does the Java class library documentation say about this? Implement and explain the correctness of your solution for concurrent download.

2. Make the cancellation button work also with concurrent download.

3. Make the progress bar work also with concurrent download. One way to do this is to create an AtomicInteger that all the download operations update as they complete, and let them all call `setProgress` with a suitable value.

Exercise 7.2 File TestLiftGui.java contains an implementation of a lift simulator, corresponding to the north end of the IT University's atrium: two lifts, both serving seven floors, from basement (floor number -1) to floor 5.

1. Explain why the whole simulation and its graphical user interface is thread-safe, in spite of the Swing GUI toolkit components not being thread-safe.

2. Apply the ThreadSafe tool to the simulation program. Does it report any potential problems?

3. Change the lift simulator and GUI to work for a hotel with four lifts, all of which serve floors -2 through 10, and still with a single lift controller.

4. In the current implementation, each lift has a thread whose `run` method uses the `Thread.sleep` method to sleep most of the time. An alternative design is to use the Java executor framework, for instance, a scheduled thread pool, to periodically update each lift's state. The `scheduleAtFixedRate` method of the ScheduledThreadPoolExecutor class in package java.util.concurrent seems relevant. In this design, each lift is represented by a Runnable whose `run` method gets called, say, 16 times a second. The main work in this rewriting probably is to introduce extra fields in the Lift object so that the lift "knows" which state it is in: going nowhere (direction `None`), going up (direction `Up`), going down (direction `Down`), opening doors, or closing doors, and so that the `run` method can act accordingly. There should be **no** calls to `sleep` left in the Lift methods.

5. Modify the user interface so that a lift's inside buttons show which floors the lift will eventually stop at. For instance, you may set the foreground (text) color of the button for a given floor to `Color.RED` when the lift will stop there, otherwise the (default) `Color.BLACK`.