CS 235 - Week 9 Lab Exercise - 2021-10-22

Deadline

Due by the end of lab on 2021-10-22.

How to submit

Submit your `.java` files for this lab on https://canvas.humboldt.edu

Purpose

To practice a bit with Java threads.

Important notes

- IF you are attending the lab via Zoom, you are expected to pair program in a breakout room (possibly trio-program if necessary based on class members' Internet and the number of class members attending via Zoom).
  - In this case, **be sure to TYPE BOTH (all) OF YOUR NAMES** in the beginning comment of EACH of your `.java` files

  But, because of the delta variant surge, if you are attending lab in person in BSS 317, you will each work on a separate computer, although discussion amongst those attending will be encouraged!

FUN FACTS-or-REMINDERS

FUN NEW-FACT 1: A thread does have a name even if you don't explicitly give it one -- in my pre-lab playing, it looks like it automatically gets a name like `Thread-0`, `Thread-1`, etc.

FUN FACT-or-REMINDER 2: `Math`'s `random` method expects no arguments, and returns a `double` value in `[0.0, 1.0)`.

FUN FACT-or-REMINDER 3: You can cast a `double` expression to an `int` by preceding it with `(int)`

Your Task

Create a small Java application `MultiThreads.java` that either:

- expects one command line argument, expected to be a positive integer, and it tries to create that many threads, **OR**
- expects one or more command line arguments, each then treated as the name of a thread to be started

In either case, it should complain if given no command-line arguments, or (for the 1st case) if not given an positive integer command-line argument.

And, it should somehow give the threads it creates at least some time (let's say at least 20 seconds) to run before it tries to interrupt them. (You can have the `main` method do more, also, if you'd like, but make sure:

- it gives the threads at least 20 seconds to run, and
- it does attempt to interrupt each thread it creates.)

What should your threads do?

- Each thread's output should somehow include the name of the thread creating that output (so we can see that there really *are* multiple threads running at the same time.)
- Each thread should be set up such to catch an `InterruptedException` and exit gracefully -- also include some output when it exits so we can see it was interrupted and ended.

But, as long as you meet the above requirements, you then have leeway on what the threads do --

- Each thread's actions can be as simple as looping, printing a message to the screen (including the name of the running thread) and then sleeping, as in Monday's examples, until the `main` politely interrupts them, politely indicating they should exit.
- Or you can try to do something else --
  - for example, maybe each thread could loop, rolling a game die and printing its results to the screen and then sleeping; (but still include the thread's name in what it prints, so we can see that multiple threads are really running)
  - maybe each thread could pop up a `JOptionPane` when it begins and when it ends, including the thread's name in the message it displays (this *seemed* to work reasonably when I tried it, except for some mysteriously-repeated `JOptionPane` sitings...?)
  - a thread might be able to put up a `JOptionPane` (somehow including its name) when it starts, and then, perhaps, roll a die, sleep, roll a die, sleep, etc., until it is interrupted, and then it could put up a `JOptionPane` (again somehow including its name) summarizing, say, how many times that thread's die was rolled (and more or different stats if you are interested -- the average dice roll, number of times a 3 was rolled, etc.)
  - or maybe each thread starts by asking how many sides for a game die, and then silently rolls such a die, sleeps, rolls, sleeps, until it is asked to stop, when it prints a final message including the thread's name and saying how many times it was rolled (and more if you wish -- average die roll? number of times a 6 was rolled? etc.)
  - or some other task you'd like to have your threads do...

- When you are done, or before you leave lab, use Gmail to
  - MAIL a copy of your `.java` files to BOTH/ALL of you, and
  - EACH of you should SUBMIT the required files on Canvas