COSC 2P05 Assignment 4

Winter 2015/16

Due: Apr. 4 @ 10:00 am (*Late Date: Apr. 7 @10:00 am*).

In preparation for this assignment, create a folder called `Assign_4` for the assignment. The objective of this assignment is to apply concurrency in a problem solution.

Concurrency
Acme Distributing (Version 3)

In this assignment we will augment the Order Entry system from Assignment 3 to support both a call center clerk entering the requests into the system and a shipping clerk handling the requests. Concurrency and a bounded buffer will be used to allow the call center clerk (producer) and the shipping clerk (consumer) to work asynchronously.

The form (`JFrame`) from Assignment 3 will be augmented to be the producer in the producer/consumer model. A new form (similar to the one from Assignment 3 but not requiring data entry or checking, i.e. it simply presents the data) will be the consumer. A bounded buffer will be implemented to replace the queue from assignment 3.

The program will load the bounded buffer from disk and then present the two forms. When the call center clerk presses `OK` on the producer form, the validated data should be entered into the bounded buffer. When the shipping clerk support presses `OK` on the consumer form, the form should be loaded from the next entry in the bounded buffer. When both the clerk and the tech support have pressed `Quit`, the system will shut down, saving the bounded buffer to disk.

The bounded buffer should be implemented as an array-based implementation of a queue. For testing of the blocking, use a small queue size (such as 5).

Since event handlers run on the user interface (GUI) thread, they should not do anything that consumes much time or the user interface will become unresponsive. In particular, they cannot block or the GUI will stop responding. A separate thread (one for producer and one for consumer) should be started to handle the posting and retrieval of the requests, blocking as appropriate. These threads should run until the `Quit` on the form has been pressed. There will thus be four threads. One for the main program (i.e. the main method) that sets things up, creates and loads the buffer, starts the producer and consumer threads, makes the forms visible and saves the buffer at shutdown. The producer thread will handle the posting of the requests to the buffer. The consumer thread will handle the retrieval of requests from the buffer. The GUI thread (in the background and not controlled by the application) will handle user interaction (calling the listeners).

Hints:

- Since the producer thread and the listener for the producer form both access the same data they can be in the same class. Similarly so for the consumer.
- The bounded buffer should be implemented as a monitor.
- `Thread.currentThread()` can be used to get the thread object of the thread executing the code.

- the `sleep(millis)` method of a thread suspends it for a while. This can be used in a loop to periodically check on the status of things.
- the `interrupt()` method of a thread causes an `InterruptedException` at the point where a thread is blocked, if it is blocked. This can be used to get a blocked thread going so it can again to check status before continuing or blocking again.
- the `join` method on a thread causes the current thread to block until the other thread finishes. This can be used to synchronize the threads before shutting down,

Much of the code can be borrowed from your solution to Assignment 3.

Submission

Write a simple program to read the order queue and write its contents to a text file verify that the data is being entered. Test your program in a variety of situations to ensure it is working properly. It should still handle appropriate exceptions as in Assignment 2 and GUI in assignment 3.

Your submission will be in two parts: electronic and paper. The paper submission should include the listings of all Java files you have written. Print the contents of the queue file before and after execution as part of the paper submission.

In addition to your paper submission, you must make an electronic submission of the assignment. You should have a folder for the assignment (`Assign_4`). This folder should include what is needed for the marker to run your program (as indicated below). Zip this folder as `Assign_4.zip`. Log on to Sakai and select the COSC 2P05 site. On the `Assignments` page select `Assignment 4`. Attach your .zip file (e.g. `Assign_4.zip`) to the assignment submission (use the `Add Attachments` button and select `Browse`. Navigate to where you stored your assignment and select the .zip file (e.g. `Assign_4.zip`). The file will be added to your submission. **Be sure to read and check the `Honor Pledge` checkbox**. Press `Submit` to submit the assignment. You should receive a confirmation email.

The complete paper submission should be placed in an envelope to which a completed coversheet (http://www.cosc.brocku.ca/coverpage) is attached. The submission should be made to the COSC 2P05 submission box outside J328 in accordance with the assignment guidelines posted on the 2P05 Sakai site and not in violation of the regulations on plagiarism (http://www.brocku.ca/webcal/2015/undergrad/areg.html#sec67, http://www.cosc.brocku.ca/about/policies/plagiarism). **Note**: Assignments not including a coversheet will **not** be marked.

DrJava

The `.zip` folder you submit should contain the project folder including all files relevant to the project—the `.drjava`, `.java` and `.class` files for the assignment.

Other Platforms

If you are using an IDE other than DrJava to prepare your assignment, you must include the `.java` source files and the `.pdf` files described above as well as a file (likely `.class` or

`.jar`) that will execute on the lab machines. It is your responsibility to ensure that the marker can execute your program.