CS 346 – Lab 5 – Synchronization

In chapter 5 we see some approaches for synchronizing threads or processes.
Why is this such an important issue, one that cannot be ignored? ______________________

____

____

We will program in Java. Create a new directory called lab05 and do today's work in there.

Today we will examine a problem similar to producer/consumer or readers/writers. Write a Java program that implements the following simulation. We have a chef that prepares various dishes. And there are two picky eaters, one that only eats meat, and the other that prefers vegetarian food. We can model the situation as follows: the chef is a producer or writer thread that generates a random integer and places it in a buffer. You may assume that the buffer has no size limit. The two other threads will be the picky eaters: one that only consumes even numbers, and the other that only consumes odd numbers. By consuming food, we actually remove the number from the buffer.

In your program, you must enforce mutual exclusion: only 1 thread at a time may access the shared buffer. Also, please don't let an eater starve. Both eating threads should have a fair opportunity to eat.

There are several ways to tackle this problem. For example, you may want to start out with a simplified version of the problem in which both eaters are omnivores, and then add numerical constraints later. I am keeping the constraints on this problem minimal in order to give you some flexibility on how you go about finding a solution. However, note that your program needs to gracefully handle the following situations:

- An eater is ready to eat something but there is no food (number) in the buffer.
- The next dish of food is not what this eater wants to eat.

Output: Your program should produce some diagnostic output to show the user what is happening as the threads execute. It should be clear in the output that both threads are in fact eating. Periodically you should print out what "food" is still uneaten, and what is in the "stomachs" of the 2 eaters.

A suggested solution outline is given on the next page.

- Table.java (i.e. the database accessible by the threads)
    - Just 1 attribute, a collection (queue or linked list) of integer "food"
    - cookFood(integer) – Here, we add or "offer" the specified integer to the collection of food. Then, we "notify all" threads of this action.
    - eatFood(boolean) – Based on whether the eater calling this method desires an odd or even number, call wait() if there is no food available or if the next item of food available is not palatable to the eater.
      After you are done waiting, you can remove the number from the collection and return it to the eater so he can put it in its stomach.

- Chef.java, implements Runnable
    - Two attributes: a Table object, and a String containing the chef's name.
    - Initial-value constructor
    - run() – All the code is enclosed in an infinite loop. Sleep for a random amount of time, say up to 2 seconds. Then, generate a random integer value in the range 1-100. Print out this number. Then call the table's cookFood() method.

- Eater.java, implements Runnable
    - 4 attributes: a table object, the eater's name, the number of "meals" i.e. numbers eaten, whether this eater prefers odd or even food
    - Initial-value constructor – but the number of meals eaten does not need to be passed as a parameter because this needs to start at 0.
    - run() – All the code is enclosed in an infinite loop. First, call the table's eatFood() method. When it returns, announce your name and what you just ate. Increment the number of meals eaten, and print this value out as well. Then, sleep a random amount of time, slightly more time than you gave the chef. For example up to 3 seconds.

- Restaurant.java – this is the main program
    - Create a table object
    - Create a chef thread. For example:
      Thread cook = new Thread(new Chef("Francois", table));
    - Create two eater threads. Give them different names and different food preferences (odd/even).
    - Start each of the threads.

One important detail: some of the above methods need to be declared as "public synchronized" in order to make use of Java's synchronization software. Which ones?

____

Run your program. What about the output tells you that it is behaving correctly?

____

____

When you turn in your program, please include about one page of output from the beginning.