Lab 6: Producer/Consumer Problem with a Bounded Buffer

Purpose:

In this lab you will replace the CubbyHole class from Lab 5 (the Producer/Consumer Problem) with a BoundedBuffer class. You will also incorporate Java's synchronization primitives into the BoundedBuffer to prevent race conditions. Then you will use two versions of a Semaphore class to implement the semaphore solution that prevents race conditions.

Background:

The Java program you will be working with in this lab consists of the four classes encountered in Lab 5. The four classes are defined in separate files (I assume you still have these files from Lab 5):

- `Producer` class - a thread class that defines the behavior of a producer
- `Consumer` class - a thread class that defines the behavior of a consumer
- `CubbyHole` class - a one-item buffer that is shared by the producer and consumer threads (in the course of the lab you will replace this with the `BoundedBuffer`)
- `PCTest` class - the main (program) class that creates the buffer and creates and starts the threads

Later in the lab you will need to make use of the following classes:

- `BSemaphore` class - a class that defines a *binary semaphore*
- `CSemaphore` class - a class that defines a *counting semaphore*

Instructions:

1. Replace the `CubbyHole` class from Lab 5 with a `BoundedBuffer` class.

   A `BoundedBuffer` will have the same methods as a `CubbyHole`, however it will be capable of holding up to *n* integers instead of just a single integer. (The capacity *n* should be a parameter of the constructor.) Implement the `BoundedBuffer` as a circular array.

   Note: There is a discussion in section 5.4 of the textbook about the Producer/Consumer Problem. Near the end of the discussion the author talks about a "Finite Circular Buffer". This is the same as what I call a "circular array".

   Note: Remove the synchronization features that you added at the end of Lab 5. That is, *synchronized*, the *if* (or *while*) with the *wait()*, and the *notify()*. We will re-introduce these in the next step of this lab.

Implementation notes:
- the `BoundedBuffer` should store integer values
- create the array in the constructor
- the buffer needs to maintain both a *count* and a *capacity*
- to manage the circular array the buffer needs to maintain two indices *in* and *out*

Modify the program `PCTest` to create a `BoundedBuffer` and pass the `BoundedBuffer` to both the `Producer` and `Consumer` threads.

Test the execution of the program with a `BoundedBuffer` of capacity 4 and with the `Producer` and `Consumer` threads each processing 10 items using the `BoundedBuffer`.

If everything worked correctly one should have the `Producer` depositing the values 0 through 9 into the shared buffer and the `Consumer` retrieving the values 0 through 9 in that order. Print an execution of the program.

Reverse the order in which the Producer and Consumer threads are started and run the program again. Print an execution of the program.

Does it work as it should? (As with Lab 5 a correct execution is identified by the correct sequence of values produced/consumed, not by the order the statements appear in the console.)

2. Modify your `BoundedBuffer` class so that it provides synchronized access to the buffer.

   Under what conditions should the `Consumer` wait? Define a boolean variable that represents this condition. What is the initial value of this variable? Where and when should this variable be changed?

   Under what conditions should the `Producer` wait? Define a second boolean variable that represents this condition. What is the initial value of this variable? Where and when should this variable be changed?

   Note: Rather than use an *if* statement to guard the wait() you should use a *while* statement. The reason is that the thread may be notify'd by a thread of the same class (Producer or Consumer). You will also want to use *notifyAll* to notify all the threads waiting on a condition.

   Compile and execute the program. Does the program behave as it should? Reverse the order in which the Producer and Consumer threads are started and run the program again.

   Before proceeding with the other steps of this lab, have your lab instructor review your `BoundedBuffer` for correctness.

   Print an execution of the program. Print out and hand in a copy of your (modified)

`BoundedBuffer` class.

3. Modify the Producer and Consumer thread classes as follows.

   Modify the constructors for both classes to take an additional parameter specifying the number of items to process. Further modify the run() method of the Producer class to generate random integers in the range 1 to 100 to deposit in the buffer. Here is Java code to generate a random integer in that range:

   ```
   int val = (int) (100*Math.random()) + 1;
   ```

   Test the modifications made before proceeding.

   Modify the main method in PCTest to take four command-line arguments:
   - buffer size
   - total number of items to process
   - number of Producer threads to create
   - number of Consumer threads to create

   (Note: The number of items to process should be divisible by both the number of producers and the number of consumers. Terminate the program with a message if these conditions aren't satisfied.)

   To convert a String to an int use the following technique:

   ```
   int val = Integer.parseInt(str);
   ```

   For example, command-line arguments `5 20 2 5`
   should create a buffer of size 5, two Producer threads each generating 10 random integers, and five Consumer threads each retrieving 4 items from the buffer.

   Test the program on the example data. Print an execution of the program and turn it in with your lab write-up. Print a copy of `PCTest.java` and turn it in with your lab.

Hand in:

Hand in the answers to the questions, the discussion, and the results of execution requested in steps 1 - 3. Also hand in a copy of the file `BoundedBuffer.java` from Step 2. Also hand in a copy of the file `PCTest.java` from Step 3. Finally e-mail the final copies of all the files `Producer.java`, `Consumer.java`, `BoundedBuffer.java` and `PCTest.java` to your instructor.