Lab 8: Semaphore and Monitor Solutions to the Readers-and-Writers Problem

Purpose:

In this lab you will use the Semaphore class BSemaphore to implement the semaphore solution to the Readers and Writers Problem. Then you will implement a monitor solution to the Readers-and-Writers Problem using Java’s synchronized methods. This exercise illustrates the general technique for writing monitor code in Java.

Background

The Readers-and-Writers Problem was introduced last week and we discussed a solution using semaphores in class.

Recall that in the Readers-and-Writers Problem there are two classes of processes (i.e., threads): Readers and Writers that require access to a shared database. The problem is to control the access in such a manner as to allow any number of Readers to access the database at the same time but whenever a Writer accesses the database to guarantee that the Writer has exclusive access to the database.

I have provided a Reader class and a Writer class along with a driver program RWDriver. A Reader thread maintains an id, a start time, and a duration which is the length of time it reads the database. Similarly, a Writer thread maintains an id, a start time, and a duration which is the length of time it writes to the database. Both Reader and Writer threads use the sleep function to delay starting and delay during read/write access. The threads also display messages at strategic locations within the code that indicate where they are during execution. This enables one to trace the execution.

To start this lab you will need the following files:

- **RWDriver.java** - the driver program for the simulation of the Readers and Writers
- **Reader.java** - defines the behavior of a Reader thread
- **Writer.java** - defines the behavior of a Writer thread
- **BSemaphore.java** - defines a binary semaphore

Hand in:

The execution results and discussion from questions 1 - 3. A listing of the four files requested at the end of Step 2. E-mail copies of these four files to your instructor as well.

A listing of the file **RWDriver.java** requested in Step 3.

The execution results and discussion from questions 4 - 7. A listing of the four files requested at the end of Step 7. E-mail copies of these files to your instructor as well.
Instructions:

1. Create a new project with the files `RWDriver.java`, `Reader.java`, and `Writer.java`, and `BSemaphore.java`.
   Execute the program and print the results of the execution.
   In what order do the threads access the database? Is there any point at which the Writer does not have exclusive access to the database? Explain, referring to the results of the execution.

2. Design and code a Database class that controls access to the database. In addition to the constructor, the class should provide four methods: `startRead()`, `endRead()`, `startWrite()`, and `endWrite()`. The methods are to implement the semaphore solution to the Readers-and-Writers problem that was presented in class and which can be found in Figure 5.25 on p. 243 of our textbook. The Database will need to manage the semaphores needed as well as the number of readers.
   Modify the behaviors of the Reader and Writer threads to call these methods at the appropriate places. To make all of this work, have the driver program create the Database object then pass it to the Reader and Writer threads when they are created. (Which means that you will need to modify their constructors.)
   Print an execution of the modified program.
   In what order do the threads access the database? Is there any point at which the Writer does not have exclusive access to the database? Explain, referring to the results of the execution.
   Hand in appropriately documented copies of the files `RWDriver.java`, `Reader.java`, `Writer.java`, and `Database.java`. E-mail these files to your instructor as well.

3. Modify the driver program `RWDriver.java` to generate five Readers and three Writers as follows:

<table>
<thead>
<tr>
<th>Thread</th>
<th>Starts</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>R₂</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>W₁</td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>R₃</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>W₂</td>
<td>50</td>
<td>20</td>
</tr>
<tr>
<td>R₄</td>
<td>60</td>
<td>50</td>
</tr>
<tr>
<td>R₅</td>
<td>80</td>
<td>50</td>
</tr>
</tbody>
</table>

Produce two executions, one without any access control (without calls to `startRead()`, `endRead()`, `startWrite()`, `endWrite()`), and one with the access control. Discuss the correctness of the two executions referring to the results.

Hand in a final copy of the file `RWDriver.java`. 

Monitor Solution to the Readers-and-Writers Problem

Now you will write a monitor solution to the Readers-and-Writers Problem.

4. Create a new project with the files **RWDriver.java, Reader.java**, and **Writer.java**. (Note that this is the same state of the code that was found at step 1 of this lab.)

   Execute the program and print the results of the execution.

5. As we did with the semaphore solution, design and code a Database class that controls access to the database. (I suggest you name it DatabaseMon to keep from confusing it with the semaphore implementation.) In addition to a constructor, the class should provide four methods: **startRead(), endRead(), startWrite(),** and **endWrite().** The four methods are to be written as synchronized methods based on the monitor solution to the Readers-and-Writers Problem. (Note: You will encounter some implementation difficulties since Java's monitor is a bit different than the one described in class. I will give additional guidance on implementing the monitor during lab.)

   Within the implementation of the methods add statements to help trace execution as follows:

   **startRead()** – if the Reader is forced to wait, display the message:
   
   “<Reader name> is waiting to read, number of Readers = #.”
   
   (where <Reader name> is the name of the Reader and # is the private value tracking the number of Readers)
   
   When the Reader gains access to the database display the message:
   
   "<Reader name> has begun reading, number of Readers = #."

   **endRead()** – display the message:
   
   “<Reader name> is finished reading , number of Readers = #.”

   **startWrite()** – if the Writer is forced to wait, display the message:
   
   “<Writer name> is waiting to write, number of Readers = #.”
   
   When the Writer gains access to the database display the message:
   
   “<Writer name> has begun writing.”

   **endWrite()** – display the message:
   
   “<Writer name> has finished writing.”

   **Note:** Indent the Writer messages so that they are easily distinguished from Reader messages in the output.

   Look up the Thread class in the Java API to find out how you can obtain the name of the currently executing thread. You will need this to display the <Reader name> and <Writer name>.

   Modify the driver program, in **RWDriver.java**, to create a DatabaseMon object and pass the object to the Reader and Writer threads when they are created. This means that you will need to modify the Reader and Writer constructors. Further modify the Reader and Writer
classes so that calls are made to startRead(), endRead(), startWrite(), and endWrite() at the appropriate points.

Print an execution of the modified program.

In what order do the threads access the database? Is there any point at which the Writer does not have exclusive access to the database? Explain, referring to the results of the execution.

6. Modify the driver program RWDriver.java to generate five Readers and three Writers as follows:

<table>
<thead>
<tr>
<th>Thread</th>
<th>Starts</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>R₂</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>W₁</td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>R₃</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>W₂</td>
<td>50</td>
<td>20</td>
</tr>
<tr>
<td>R₄</td>
<td>60</td>
<td>50</td>
</tr>
<tr>
<td>R₅</td>
<td>80</td>
<td>50</td>
</tr>
</tbody>
</table>

Produce two executions, one without any access control (without calls to startRead, endRead, startWrite, endWrite), and one with the access control. Discuss the correctness of the two executions referring to the results.

7. Experiment with various Reader and Writer threads and their start times and durations to see if you can get an execution (using the access control) that allows several Readers access to the database, then allows a Writer access, then allows several other Readers (that arrived while the Writer had access) access to the database.

If you are able to get such an execution, print a copy of the execution. Also describe the threads and parameters you used to get this execution.

Hand in a copy of each of the files: Reader.java, Writer.java, RWDriver.java, and DatabaseMon.java. Make sure these are versions of the files that enforce correct access. E-mail copies of the files to your instructor.

Extra Credit: Modify the DatabaseMon class to implement the version of the monitor solution to the Readers and Writers Problem that does not give precedence to the Readers.