CS117 (Fall 2024) Assignment 9 (programming) 20 pts Due: Dec. 01 (Sun)

Practice for Concurrent programming with Java Threads

Task: Serially reusable resource management for concurrent processes.

Assume that there exist two types of serially reusable resources (SR1 and SR2) with 3 units and 2 units, respectively, and 6 concurrent processes in the system.
Each process acquires both SR1 and SR2 (one unit each), performs operation, and releases SR1 and SR2 units after the operation.

We can interpret the above concept with the following student activity example.
There exist 3 red pens and 2 blue pens; 6 students work for drawing a diagram each with both red and blue pens. In this situation, those 6 students compete for accessing both read and blue pens; once a student accesses both red and blue pens (one each) the student can draw the diagram; after finishing the drawing, the student releases the red and blue pens for other students who are waiting.

For the implementation, we need the following 4 classes:

Manager: This is the main class, responsible for creating multiple (6) threads for those 6 processes and making them run concurrently.
Note that the shared objects (serially reusable resources) should be created and passed as parameters to the constructor of each thread.

SR1: A monitor class for the serially reusable resource type1.
This encapsulates data (available unit counter) and related operations on that, i.e., acquire( ) and release( ), which are defined as synchronized methods for mutual exclusion. For the synchronization, you should use wait( ) and notify( ) appropriately in those synchronized methods.

SR2: A monitor class for the serially reusable resource type2.
This encapsulates data (available unit counter) and related operations on that, i.e., acquire( ) and release( ), which are defined as synchronized methods for mutual exclusion. For the synchronization, you should use wait( ) and notify( ) appropriately in those synchronized methods.

Process: Thread class for the working process, which acquires both SR1 and SR2 (1 unit each), performs operation (simply displaying a message), and releases both SR1 and SR2 (1 unit each).
For the purpose of tracing the behavior of the system, please use sleep( ) before each call for a synchronized method, i.e., acquire( ) or release( ).

For tracing the behavior of the system, the following messages should be generated at run time:
Thread for process_# created,
Process_# acquires SR#,
Process_# is waiting for SR#,
Process_# is working,
Process_# releases SR#,
(See attached sample testing output.)

Submit: Your source code(s) and run-time output – please do not edit your code/output files.
Please include documentations in your code, i.e., global, class head, function head documentations.

Sample output (one possible output)

```
===== Thread for process_1 created
===== Thread for process_2 created
===== Thread for process_3 created
===== Thread for process_4 created
===== Thread for process_5 created
===== Thread for process_6 created
process_2 acquires sr1
process_4 acquires sr1
process_1 acquires sr1
process_1 acquires sr2
---- process_1 is working
process_2 acquires sr2
---- process_2 is working
+process_6 is waiting for sr1
process_1 releases sr1
process_6 acquires sr1
+process_5 is waiting for sr1
+process_3 is waiting for sr1
+process_4 is waiting for sr2
+process_6 is waiting for sr2
process_1 releases sr2
process_6 acquires sr2
---- process_6 is working
process_2 releases sr1
process_3 acquires sr1
+process_3 is waiting for sr2
process_6 releases sr1
process_5 acquires sr1
process_2 releases sr2
process_3 acquires sr2
---- process_3 is working
+process_5 is waiting for sr2
process_3 releases sr1
process_3 releases sr2
process_5 acquires sr2
---- process_5 is working
process_6 releases sr2
process_4 acquires sr2
---- process_4 is working
process_4 releases sr1
process_5 releases sr1
process_5 releases sr2
process_4 releases sr2
```