Mutual Exclusion

DESCRIPTION

The purpose of this assignment is to implement and examine a Java solution to synchronize the threads to access the critical section. (1) Lamport's Bakery Algorithm: Integrate this algorithm into the provided Race.java program from the previous assignment to schedule an arbitrary number of processes, ensuring mutual exclusion when accessing the critical section. (2) Semaphore: use semaphore to ensure mutual exclusion.

GOALS

1. Learn Java solutions to ensure Mutual Exclusion in a multithread problem.

TASKS

Step1: Modify race.java

a. For the Racer class:
   1) Have the Racer class extend **Thread** and NOT implement **Runnable** using **extends** keyword.
   2) Add an integer **id** data member and an **Arbitrator** data member.
   3) Add above two parameters to the existing **constructor** to be able to initialize these two new data members.
   4) Make **sum** static.
   5) Put appropriate **calls** to the public members of Arbitrator to enforce Mutual Exclusion of the **ENTIRE** for loop in the run method.

b. For RaceTwoThreads:
   1) Rename the class RaceManyThreads instead of RaceTwoThreads.
   2) Remove final from numRacers.
   3) Add a static Arbitrator object declaration.
   4) Add an option letter N that expects an argument to designate the number of racers. Change the usage String variable to be: *"Usage: -M m -N numracers"*
   5) Remove the declaration and any reference to racer Object, create an Arbitrator **object**, make racer **an array of** Racers NOT Threads. When creating individual **racer object** make sure to pass the appropriate constructor parameters.
   6) After printing the value of M, **print** the number of racers.

Step2: Do NOT make any changes to Arbitrator.java.

Step3: Compile & Run

To compile, run the command:

```
javac *.java
```

To run:

java RaceManyThreads -M20 -N4

Step4: See sample.out **for sample output.**

Step5: Document all your code in Arbitrator.java and race.java according to the programming ground rules (of 2430).

Step6: Run a script as follows:

Script: part1

javac *.java

java RaceManyThreads -U

java RaceManyThreads -M 20 -N 4

java RaceManyThreads -M 30 -N 5

exit

Step7: Write a script to run your program according to the table:

Write a script: part2, to run your program using the given data in the table below for M and number of racers so as to complete the following table:

Run #	Value of M	Number of racers	Last value of sum
1	10	2	
2	10	3	
3	20	4	
4	20	5	
5	50	5	

Step8: Modify race.java

Use semaphore to ensure mutual exclusion.

SUBMISSION

1. Zipped source code with Arbitrator.java and race.java documented according to the programming ground rules (of 2430).
2. A completed table from step 7.
3. A programming report with running result of script files, part1 and part2.

Rubric

Criteria	Ratings			Pts
Execute: Compile and execute correctly with no errors	**10 pts** Correct	**0 pts** Not correct or no submission		10 pts
Script part1: The result is correct.	**10 pts** Correct	**0 pts** Not correct or no submission		10 pts
Script part2 and table in step 7: The result is correct.	**10 pts** Correct	**0 pts** Not correct or no submission		10 pts
Lamport's algorithm: Race class revised to control CS entering.	**20 pts** Correct	**15 pts** Some errors	**0 pts** Not correct or no submission	20 pts
Semaphore: Race class revised to control CS entering.	**10 pts** Correct	**0 pts** Not correct or no submission		10 pts
Document code: Arbitrator.java and race.java documented according to the programming ground rules (of 2430).	**20 pts** Correct	**10 pts** Some errors	**0 pts** Not correct or no submission	20 pts
Report: Fill up the programming report with reasonable details	**20 pts** Report with reasonable details.	**10 pts** Tried without enough details.	**0 pts** No submission	20 pts

Criteria	Ratings	Pts
Total Points: 100		

CONGRATULATIONS, YOU'VE COMPLETED PROGRAM 5!