CSCI 340 Spring 2016

Project 2 – Due Date: May 11th

Using Java programming, synchronize the three types of threads, in the context of the problem. Closely follow the implementation requirements. The synchronization should be implemented through Java semaphores and operations on semaphores.

Any wait must be implemented using P(semaphores), any shared variable must be protected by a mutex semaphore such that Mutual Exclusion is implemented.

Document your project and explain the purpose and give the initialization of each semaphore.

DO NOT use synchronized methods (beside the operations on semaphores).

Do NOT use wait( ), notify( ) or notifyAll( ) as monitor methods. Use only the semaphore class and its methods. Whenever a synchronization issue can be resolved use semaphores and not a different type of implementation.

You should keep the concurrency of the threads as high as possible, however the access to shared structures has to be done in a Mutual Exclusive fashion, using a mutex semaphore.

Many of the activities can be simulated using the sleep(for a random time) method.

Use appropriate System.out.println( ) statements to reflect the time of each particular action done by a specific thread. This is necessary for us to observe how the synchronization is working.

Submission similar to project1.

The Adventurers and the Green Dragon

Dudley is a town, famous for its magical jewelry shop and for the green-dragon that lives on a nearby mountain. Many adventurers come to the shop and bring various items (stones, rings, necklaces, earrings) that can be combined together into more powerful objects. A precious stone can be combined with a ring to make a magical ring, or with a chain to make a magical necklace, or with an earring to make a magical earring. An adventurer may have a number of items that result in one or more magical jewels (the number of each item is randomly generated as an integer number between 0 and 3). Every adventurer has the goal of leaving the town once he has accumulated a small fortune in magical rings, earrings and necklaces of size **fortune_size**.

If an adventurer has enough items to make a complete magical ring, or necklace, or pair of earrings (at least one precious stone and ring, or one precious stone and chain, or **two** precious stones and **two** earrings), he goes directly to the jewelry shop (simulate the trip to the shop by using **sleep(random_time)**.

At the shop door the adventurer **waits** until a clerk is ready to serve him. There are **num_clerks** but one line of adventurers. The clerks randomly pick an adventurer to serve.

Any adventurer that doesn't have enough items to create a complete magical jewel, or that didn't accumulate his fortune yet, goes to the mountain to battle the dragon. Adventurers gather at the dragon's cave and **wait** for the dragon to be available.

The dragon signals one of the adventurers. The dragon loves to play dice. The dragon's game of dice is a simple one – the most points win. He will play with each adventurer **num_games**. However after each game the dragon signals an adventurer that he is available.
The adventurer takes a seat at a table. There are **num_tables** available. If there are no tables available the adventurer will **wait** (use a counting semaphore).
The dragon plays a dice game with each of the adventurers in a round-robin order based on how the adventurers are seated.

For examples: Let's say adv1 sits at a table (name it t1)
Adv5 sits at a table (name it t2)
Adv3 sits at a table (name it t3)
In fact t1, t2, and t3 are instances or the table resource.

The dragon plays the first game with adv1, second game with Adv5, 3rd with Adv3, 4th game (is the 2nd game with Adv 1) with Adv1
And so on

After the dragon and the adventurer play 3 games, if the dragon loses to an adventurer, it has to give up a random item (precious stone, or ring, or chain, or earring). If the adventurer doesn't get the item that he needs for a complete magical jewel, he will wait again for another chance to fight the dragon.

When the adventurer has the right items to create a magical jewel, he must go back to the jewelry shop.

Once the adventurer achieves his goal by collecting the desired number of magical jewels, he will be ready to go home with his treasure. However, he will wait until all the other adventurers are done. Implement a platoon policy, such that the last adventurer to be done will signal another adventurer, who will signal another one and so on. After all adventurers terminate the other threads will terminate as well.

Using Java programming, synchronize the three types of threads: adventurer, clerk, dragon in the context of the problem. Closely follow the implementation requirements.

The number of adventurers (num_adv), clerks (num_clerk), fortune size (fortune_size) and number of tables (num_table) must be entered as command line arguments.

Default values: num_adv = 8

num_clerk = 2

fortune_size = 3

num_table = 3

Choose appropriate amount of sleep time(s) that will agree with the content of the story. Note: I haven't written the code for this project yet, but from the experience of grading previous semester's projects, a project should take somewhere between 45 seconds and at most 1 minute and ½, to run and complete.

Academic integrity must be honored at all times and no code sharing or cheating is permitted. Only submit code that you have written.

Guidelines

1. Do not submit any code that does not compile and run. If there are parts of the code that contain bugs, comment it out and leave the code in. A program that does not compile nor run will not be graded.

2. Closely follow all the requirements of the Project's description.

3. Main class is run by the main thread. The other threads must be manually specified by either implementing the Runnable interface or extending the Thread

class. Separate the classes into separate files. Do not leave all the classes in one file. Create a class for each type of thread. Don't create packages.

4. The program asks you to create different types of threads. For the Clerk thread type and the Adventurer thread type, there is more than one instance of the thread.
No manual specification of each thread's activity is allowed (e.g. no Clerk1.checkFortune()).

5. Add the following lines to all the threads you make:

```
    public static long time = System.currentTimeMillis();

    public void msg(String m) {
        System.out.println("["+(System.currentTimeMillis()-time)+"] "+getName()+":
"+m);
    }
```

It is recommended to initialize time at the beginning of the main method, so that it will be unique to all threads.

6. There should be printout messages indicating the execution interleaving. Whenever you want to print something from that thread use: msg("some message about what action is simulated");

7. NAME YOUR THREADS or the above lines that were added would mean nothing. Here's how the constructors could look like (you may use any variant of this as long as each thread is unique and distinguishable):

```
// Default constructor
public RandomThread(int id) {
    setName("RandomThread-" + id);
}
```

8. Design an OOP program. All thread-related tasks must be specified in its respective classes, no class body should be empty.

9. **No** use of **wait( ), notify( ) or notifyAll( )** are allowed.
10. Waiting must be implemented thru operations on semaphores.

11. DO NOT USE System.exit(0); the threads are supposed to terminate naturally by running to the end of their run methods.

12_13. Command line arguments must be implemented to allow changes to the **num_adv, num_clerk, fortune_size and num_table.**

14. Javadoc is not required. Proper basic commenting explaining the flow of the program, self-explanatory variable names, correct whitespace and indentations are required.

Setting up project/Submission:

In Eclipse:

Name your project as follows: LASTNAME_FIRSTNAME_CSXXX_PY
where LASTNAME is your last name, FIRSTNAME is your first name, XXX is your course, and Y is the current project number.
For example: Doe_John_CS340_p2

To submit:
-Right click on your project and click export.
-Click on General (expand it)
-Select Archive File
-Select your project (make sure that .classpath and .project are also selected)
-Click Browse, select where you want to save it to and name it as LASTNAME_FIRSTNAME_CSXXX_PY
-Select Save in **zip format**, Create directory structure for files and also Compress the contents of the file should be checked.
-Press Finish

Email the archive with the specific heading: **CS340 Project # Submission: Last Name, First Name** to simina.fluture@qc.cuny.edu.
You should receive an acknowledgement within 24 hours.

The project must be done individually, not any other sources. No plagiarism, No cheating. Read the academic integrity list one more time.

Note: I found 5 clear cases of plagiarism on the first project. Obviously not all of you care to follow the requirements of the project and the academic integrity policy.

If you plagiarize, you will receive an F in the course and you will be reported to the Scholastic Committee. The committee will decide the next course of action(s).