CSCI 340 Spring 2016

Project 1 – Due Date: April 21st

Through its implementation, this project will familiarize you with the creation and execution of threads, and with the use of the Thread class methods. In order to synchronize the threads you will have to use (when necessary), run( ), start( ), currentThread( ), getName( ), join( ), yield( ), sleep(time), isAlive( ), getPriority( ), setPriority( ), interrupt( ), isInterrupted( ), synchronized methods

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 sets his variable **need_assistance(i)** to true and **busy waits** until a clerk is ready to serve him. There are **num_clerks** but one line of adventurers. A clerk should pick the next waiting adventurer in a mutual exclusion fashion (this can be done from inside of a **synchronized method**) and in a FCFS order.

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** (simulated by a **sleep of a very long time**) for the dragon to be available.

The dragon randomly interrupts one of the adventurers (use **interrupt()** and **isInterrupted()**). The dragon loves to play dice. The dragon's game of dice is a simple one – the most points win.
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.

If the adventurer loses, the dragon, being very happy about his victory, will allow the adventurer to increase his priority for a very short time and play one more game. (use **getPriority( ), setPriority( )** and **currentThread( )** methods). After the game the adventurer will immediately reset his priority back to the default value and will allow another adventurer to fight the dragon (use **yield( )**).

Once the adventurer achieves his goal by collecting the desired number of magical jewels, he is ready to go home with his treasure. However each adventurer must join the next adventurer in sequence. (use **isAlive( )** and **join( )**.
For example if they are 5 adventurers: A0, A1, A2, A3, A4
A0 waits for A1 to terminate, A1 waits for A2 to terminate, ……A4 just terminates.
The last adventurer to leave for home (adventurer(0)) will let the dragon know that it is time for him to terminate. The clerks 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) and fortune size (fortune_size) must be entered as command line arguments.

Default values: num_adv = 6

fortune_size = 5

num_clerk = 2

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** implementation of semaphores or use of **wait( ), notify( ) or notifyAll( )** are allowed.
10. thread.sleep() is not busy wait. while (expr) {..} is busy wait.

11. "Synchronized" is not a FCFS implementation. The “Synchronized” keyword in Java allows a lock on the method, any thread that accesses the lock first will control that block of code; it is used to enforce mutual exclusion on the critical section. **FCFS should be implemented in a queue or other data structure**.

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

13. Command line arguments must be implemented to allow changes to the **num_clerk, num_adv, fortune_size variables.**

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

Tips:
-If you run into some synchronization issues, and don't know which thread or threads are causing it, press F11 which will run the program in debug mode. You will clearly see the thread names in the debug perspective.

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_p1

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.