Guess-What or Who?

The TV game show *Guess What or Who?* certainly has seen its share of contestants. There are 3 types of threads for this story:

Announcer
Host
Contestants

There is a big pool of potential Contestants. Only the best 4 contestants will be able to compete. The Contestants must first take a preliminary written exam. The Contestants form groups of **room_capacity** size. (Contestants belonging to the same group should block on the same group object). Once the group is formed, it will wait to be notified (use notifyAll) by the Announcer that they can enter a classroom and take a seat. After all contestants are seated the exam beginnings. The exam takes **exam_time**.

When the exam ends the contestants wait for the results (use a different object for each contestant, similar to rwcv.java). The Announcer will generate **num_contestants** random numbers. First number will relate to Contestant 1, second number to Contestant 2, and so on. The winners will be the contestants that have the highest 4 numbers.
The Announcer in order let know each Contestant if it is a winner or not (use a vector of objects to enforce the FCFS order). The best 4 Contestants will wait to start the game, the others will exit.

To start the show, the Announcer thread will print an opening message (something useful, it is up to you) and create the Host thread. The Host will wait until the Announcer will signal (notify) it to begin the game. The Announcer will then introduce the 4 contestants. Each contestant will then print a message that it is ready and wait for the Host to begin the game. The Announcer will then signal the Host. The Announcer thread will exit.

The Host thread will begin to ask the questions for the game. There are **numRounds** for the length of the game (use a while loop) in addition to **numQuestions**.

The following steps should be covered:

- Print a friendly message about the Host asking the question.
- Allow for a Contestant to think *(sleep for random time*). The first Contestant that wakes up will answer the question.
- The Host will decide if the answer is right or wrong (by generating a random number).
- This makes use of the **rightPercent** variable. Print an appropriate message with the **getName**() method being used for that Contestant in addition to updating his score (it will either increase or decrease with a right or wrong answer).

Once complete, the Host so he can continue asking the questions.

Once all rounds have been played, it is time for *Final Guess What or Who?* Here, the Host will signal the contestants in order. After the Host notifies the Contestants, he will wait for the contestant to notify that he is done answering the question. The Contestant will randomly choose the amount to wager (determined from 0 to the Contestant's score). If the Contestant has a negative score, have him say "good-bye" to the Host and exit. Remaining Contestants will randomly get the last question right or wrong (with a 50% chance). In either scenario, **notify** the host. The Contestants should terminate here as they are finished with the game now.

The Host will update the scores, determine the winner of the game print the scores and the winner. Make sure to print the winner after being determined.

Once complete, the Host will say good bye and exit. The program at this time is now complete. All Threads should have been terminated.

The default values for the parameters are:

numRounds = 2;
numQuestions = 5;
questionValues = 200;
rightPercent = 0.65;
room_capacity = 4
num_contestants = 13

These can be changed on the command line.

Also, use appropriate **System.out.println()** statements in the program. Also make use of the **age()** method provided to keep track of the Threads age:

```
protected static final long age() {
      return System.currentTimeMillis() - startTime;
}
```

Have fun and good luck!!!

Academic integrity must be honored at all times and no code sharing or cheating is permitted. Only submit code that you have written. Any questions you may have in regards to any assignment should be emailed to **`JLaiQC@gmail.com`** with the subject heading specifying the class – CS344/715.

Guidelines

1. Do not submit any code that does not compile and run. If there are parts of the code that contains bugs, comment it out and leave the code in. By submitting a program that does not compile nor run shows no effort and will not be graded.
2. Follow all the requirements in Project specification.
3. Main class is not a thread, threads must be manually specified; that is, either implement Runnable interface or extend from Thread class.
4. The program requires implementation of three types of threads – `Host`, `Announcer` and `Contestant`. For the `Contestant` threads, there are more than one instance of the thread. A generic class of each type of thread should be created to specify the same activity performed by all threads of the same type.
5. Design an OOP program. All thread-related tasks must be specified in its respective classes, no class body should be empty. An example demonstrating this is as follows:

```
public Contestants implements Runnable {
      public void formGroup() { … }…
      public void run() { …   formGroup();      …     }
}

public Announcer implements Runnable {
      public void alertGroup() { … }     …
      public void run() {     …     alertGroup();      … }
}

public Host implements Runnable {
      public void startGame() { … }…
      public void run() {     …     startGame();       … }
}
```

6. Interleaving refers different threads will be performing different tasks at the same time, and answering to each other. There should be printout messages indicating this interleaving. An example of this is as follows:

```
[11] Contestant-7 enters.
[11] Contestant-2 enters.
[12] Contestant-5 enters.
[18] Contestant-4 enters, last member in group 1.
[20] Announcer notifies group 1 ready to enter room.
```

7. No implementation of semaphores or busy wait implementation are allowed. All threads must wait until they are notified.
8. "`Synchronized`" is not FCFS implementation. "`synchronized`" keyword in Java allows a lock on the method or on an object, any thread that access the lock first will control that block of code; it is only used to enforce mutual exclusion on the critical section. FCFS should be implemented in queue or other data structure. (as similar to `rwcv.java`)
9. Program also must implement the following requirements:
    - Command line arguments must be implemented to allow changes to the following parameters: `numRounds`, `numQuestions`, `questionValues`, `rightPercent`, `room_capacity`, `num_contestants`.
    - Initiate Contestant and Announcer threads from the main class. Neither threads should be initiated within any other threads, as both thread type are unique. The only exception to this rule is if you explicitly implement your main class as a thread as well.
    - Javadoc is not required. Proper basic commenting explaining the flow of program, self-explanatory variable names, correct whitespace and indentations are required.
10. Archive all *.java files into single compressed file with your full name and project number as the filename, eg. JohnDoe_715P1.zip. Email the archive with the specific heading:

CS344/715 Project # Submission: Last Name, First Name

Grading Criteria

10% Programming Style

40% Program Execution

50% Program Logic