CS419 Project 2: Simulation of a Call Center

You can team up with another student or work alone. Each team can have no more than two students.

When it is due:

Sunday, April 4, at 11:59pm.

What to submit:

1. Please submit the completed **CallCenter.java** source file.
2. Please submit a **screenshot** of the outputs from running the simulation.

For a two-person team, you will just need to submit *one* copy (either member submits on Canvas). Please be sure to submit **a note** stating it was a team submission and including the names of both team members.

Objectives:

1. Practice the use of the Java Executor framework for multi-threading.
2. Reinforce the concepts of critical section and synchronization.
3. Practice the use of Java mutex locks and conditional variables.

Instructions:

1. This project requires you to implement a Java program that simulates a call center. The Java program has the following four classes:
    a. the **Agent** class that simulates a customer service agent who answers customer calls.
    b. the **Greeter** class that simulates the automated greeting service that places every customer call into a waiting queue.
    c. the **Customer** class that simulates a customer call.
    d. the **CallCenter** class that drives the simulation.

    We have implemented the *Customer* class and the *CallCenter* class. You will need to complete the *Agent* class and the *Greeter* class. You can also add additional static variables to the *CallCenter* class, but other than that you will not need to modify the *CallCenter* class. You must not modify the *Customer* class.
2. The simulation uses multiple agents, multiple customers, and a single greeter. Each of those is implemented by a thread. (This has already been implemented in the *main* method, using the Executor framework.)

3. In the *CallCenter* class You will need to create a data structure (e.g., a queue or a list) that implements the waiting queue.
    a. The greeter places <u>every</u> new customer into the waiting queue. More specifically, it places the ID of each new customer in the queue (see also *4.b.ii* below).
    b. Whenever an agent becomes available, it will remove the first customer from the queue and serve that customer (see also *5* below).
4. The member variable *admittedNewCustomer* of the *CallCenter* class is used to synchronize multiple new customers, and also to synchronize the greeter and the new customers. We have implemented the *Customer* class, and you need to complete the *Greeter* class so that the *Greeter* class works with the *Customer* class and that proper synchronization is achieved.
    a. A new customer X upon arrival checks whether *admittedNewCustomer* is equal to -1. (The -1 value means that there is no other new customer and that the greeter is waiting.)
        i. If *admittedNewCustomer* is -1, the new customer X sets the variable to its own ID, and signals the greeter.
        ii. If *admittedNewCustomer* is not -1, that means the variable has been set by another new customer and that the greeter is busy greeting that customer. So X waits for the greeter to become available and to be signaled by the greeter.
    b. The greeter keeps checking the value of the variable *admittedNewCustomer*.
        i. If *admittedNewCustomer* is -1, that means no new customer has arrived yet, and the greeter must wait for a new customer to arrive and to be signaled by that new customer.
        ii. If *admittedNewCustomer* is not -1, that means a new customer X has arrived and has set the variable to its own ID. The greeter needs to perform the following tasks:
            1) The greeter must get the customer ID, reset the variable *admittedNewCustomer* to -1, and signal another new customer who has been waiting on the shared variable *admittedNewCustomer*.
            2) The greeter then greets the customer X by calling the provided *greet()* method, places customer X's ID in the waiting queue, and signals a waiting agent that a new element has been added to the waiting queue.
            3) When calling the *greet()* method, the greeter needs to pass the expected waiting time as the argument *t*. The expected waiting time depends on the number of customers in the waiting queue:

if there are N customers in the queue, the expected waiting time *t* should be set as *t = N * 2*.

c. The greeter should exit after it has served *NUMBER_OF_CUSTOMERS* customers.

5. An agent keeps checking if the waiting queue is empty.
    a. If the waiting queue is empty, then the agent waits for a new customer to be placed in the waiting queue and to be signaled by the greeter.
    b. Otherwise, the agent will remove the first element from the queue, and serve that customer by calling the provided *serve()* method.
    c. An agent should exit after it has served *CUSTOMERS_PER_AGENT* customers.

Sample Output:

1. The output is produced by the *serve()* method of the **Agent** class and the greet() method of the **Greeter** class. We provided those two methods and you must not change them.
2. It is OK if the messages are printed in different order or the waiting times differ (There is some randomness in this simulation). Please make sure that:
    a. Each agent serves exactly *CUSTOMERS_PER_AGENT* customers.
    b. Each customer is served by a single agent.
    c. The program exits and not hangs forever. Note that the program only exits after all threads have exited; if any thread hangs then the program will hang.

```
The expected waiting time for Customer 1 is 0 minutes.
The expected waiting time for Customer 2 is 0 minutes.
The expected waiting time for Customer 3 is 2 minutes.
Agent 1 is serving customer 1
Agent 3 is serving customer 3
Agent 2 is serving customer 2
The expected waiting time for Customer 4 is 0 minutes.
The expected waiting time for Customer 5 is 2 minutes.
The expected waiting time for Customer 6 is 4 minutes.
The expected waiting time for Customer 7 is 6 minutes.
The expected waiting time for Customer 8 is 8 minutes.
Agent 3 is serving customer 4
Agent 2 is serving customer 5
Agent 1 is serving customer 6
The expected waiting time for Customer 9 is 4 minutes.
The expected waiting time for Customer 10 is 6 minutes.
The expected waiting time for Customer 11 is 8 minutes.
The expected waiting time for Customer 12 is 10 minutes.
Agent 3 is serving customer 7
The expected waiting time for Customer 13 is 10 minutes.
Agent 1 is serving customer 8
Agent 2 is serving customer 9
The expected waiting time for Customer 14 is 8 minutes.
The expected waiting time for Customer 15 is 10 minutes.
Agent 2 is serving customer 10
Agent 3 is serving customer 11
Agent 1 is serving customer 12
Agent 2 is serving customer 13
Agent 3 is serving customer 14
Agent 1 is serving customer 15
```

Grading:

This project carries 100 points. You will receive 60 points for correctly implementing the **Greeter** class. (The greeter must correctly synchronize with both the customers and the agents.) You will receive 40 points for correctly implementing the **Agent** class. (An agent must correctly synchronize with the greeter and also with other agents.)