Instructor: Dr. Simina Fluture

CSCI 340 Summer 2012

Project 1 – **Due Date: Fri, August 3rd**

Through its implementation, this project will familiarize you with the creation and execution of threads, and the use of the Thread class methods. In order to synchronize the threads you will have to use the following methods when necessary: run( ), start( ), currentThread( ), getName( ), join( ), yield( ), sleep(time), isAlive( ), getPriority( ), setPriority( ), interrupt( ), and synchronized methods for mutual exclusion. You CANNOT use wait(), notify, or notifyAll( )

Note: the projects must be done individually. No exceptions. Collaboration will be treated as cheating.

Online Office Hours

Students come to school and wait (*use busy waiting*) in front of the computer lab until it is time for the lab to open. Once the lab is open, students enter the lab to the **capacity** of the lab. If the capacity is reached no additional students can enter the lab. Students beyond the lab capacity will terminate their execution.

There are two types of questions that the student might have. The type and number of questions that each student will be allowed to ask is determined randomly.
Type A: the student can email the teacher up to **#questions_A**. These questions do not need an immediate answer from the teacher.
Type B: the student will have a chat session with the teacher. During a chat session, the student is allowed to ask at most **#questions_B.**

Once the student enters the lab, he will take some time to think about his questionA. *(use **yield( )** and later on **sleep(random time)**)*. Each question should contain a timestamp (*use **age()***) and the name of the student (*use **getName()***).

Type A questions: As soon as the student determines his questionA, he will be in a rush to send it to the teacher. *(implement this by having the student increase his priority, use **getPriority()**, and **setPriority()**. Simulate sending the question by **sleep(random time)** and immediately after that reset the thread's priority to the default value)*. All of the questions sent by students will be answered by the teacher **in the order in which they have been sent.**

After the student is done sending his typeA questions, if he wants to have a chat with the teacher, he needs to wait until the chat session starts and it is his turn to chat. Once it is his turn to chat, he will send a question to the teacher and wait for an answer. If he has fewer than **#questions_B** he needs to let the teacher know when it is his very last question (when his chat session ends).

After his chatting session, the student will browse the Internet waiting for the online office hour to end (*use sleep(time), where time is long enough such that when the online office hours end, the student will be awakened by an **interrupt()***)

Next the students will leave the lab in descending order of their name. (*Implement this using **isAlive()** and **join()***). For example if there are 4 student threads named from T0 to T3. T0 will join T1, T1 will join T2, T2 will join T3 and T3 after some sleep of random time will terminate.

The teacher arrives at his office before the online office hour begins. He spends some time answering typeA questions, but might not be able to answer all of them. Once the time for the online chatting session arrives, he will inform the first student waiting to chat that he is online and can take questions. He chats with students in the order in which they queue for the online chat session. When done chatting, if there is any remaining time until office hours ends, the teacher checks if there are any new or unanswered typeA questions and answers them.
Once the office hour ends, the teacher will terminate as well. The timer will keep track of the times and, if necessary, signal the correct threads.

Synchronize the *student* threads, *teacher* thread and *timer* thread following the conditions of the story. Instead of creating a timer thread, you can chose to have the main thread taking track of the specific times. It is your choice.

For example a possible time scale can be (you might consider additional time intervals):

Teacher's arrival time
Start of online office hour
start online chatting session
end online chatting session
end online office hour

Use appropriate System.out.println() statements in the program. Also make use of the **age()** method provided to keep track of the Threads age and action time.
There are no real messages exchanged; it is just a simulation.

Instructor: Dr. Simina Fluture

capacity, ***#Students***, **#questions_A, and #questions_B should be read as command line arguments.**

Defaults values:	**capacity**	**15**
	#Students	**11**
	#questions_A	**4**
	#questions_B	**3**

If you have any questions related to the project please email the TA with a cc to me.

The submission requirements will be posted shortly in a separate file.

Good Luck !!!!!!!!!!!!!!!!!!!