Name: Uyen-Dung Vo
UIN: 424008044
Acknowledgments: N/A

Question 4: Describe your implementation in a detailed design document.

Overview

This *GameShow.java* is a multi-threaded program with three tasks: displaying a new question, reading in the input, and timing the 10 seconds for the current question. These questions may be randomized or not, based on my understanding of the problem, and the GameShow should be knowledgeable of the answers to these questions. The answer to a particular question is then compared to the answer in which has two possible events: the user is right (in which their points should increment) or if the user is wrong (in which the correct answer will be displayed and no points rewarded). There is also the event that there is no answer and the time runs out, in which the correct answer will display and no points rewarded. The next question is displayed at all times. The correct answer only displays when the user is wrong/lack of submitting an answer.

Analysis and Implementation

Three tasks means that the *PrintQ*, *Read*, and *TimeTen* tasks must have their own classes in which implement the *Runnable* interface and override the *run()* function to dictate how the task is supposed to be carried out. The *lock* should exist to separate the “print question” and “reading” tasks” because the timing task should be running in the background as both of these tasks run. Three tasks means three threads that we must call *Thread.start()* on to begin.

There should be an array or *ArrayList* of String type that contain the questions and answers. These may be of global scope to allow all the classes to have access to this content so we can allow for displaying the Correct answer in the various events. Doing this means we must load the questions/answers into the *ArrayList* before conception of the threads. Since the *GameShow.java* seems to be a one-player game, I assume a global variable to keep track of *points* should be implemented.

Things to consider using in this assignment are semaphores, locks (specifically the *ReentrantLock*), and the various *Thread* functions such as *sleep(*x*), interrupt(),* and *join().* Sleeping puts the thread to sleep for *x* milliseconds, interrupting puts an interrupted flag on the thread, and joining causes the current thread to wait for the thread that called it to finish before continuing the contents in its *run().* Conditions created by calling *Lock.newCondition()* are also a tool.

Actual Implementation Comments

I implemented this by instantiating and defining the *questions* list in the *PrintQ* class. Putting these in the global scope of the *GameShow.java* caused problems, but this worked and in a way I find it more elegant because the *answers* list is instantiated and defined in the *Read* class, which keeps the questions and answer separate. I put locks on both the *PrintQ* and *Read* as I

had said and controlled the *TimeTen* class via interruption. I don't know if this is proper, but I believe using *Thread.sleep* to dictate the timing was unorthodox to begin with.

The *GameShow.java* operates by traversing through a previously made list of questions and answers; more questions and answers can be added easily by manipulating the constructors of the respective classes described above. User-input event-handling is managed primarily by the *Read* class in which through the use of global flags (*timeFlag* and *answered* to help make sure each function is aware of the current state of the program) and *Conditions,* we output the correct line of text and point control depending on the user input which is always a String. I used the *String.compareTo(String* x*)* but I see I could also have used *String.equals(String* x*).* The *Condition* was used in the *PrintQ* and *Read* class as I ran into problems of the next question advancing too early. Therefore I put a *Condition.await()* in *PrintQ* and a *Condition.signalAll()* in *Read* to let the question printing wait for when the *Read* thread fully completes. Further event handling occurs in the *PrintQ* task since I used the *TimeTen* exclusively for just simple timing. I had issues getting the *TimeTen* task to restart for every question since they began overlapping as well if the time did not run out yet for the previous question. I solved this issue via calling *Thread.interrupt()* and thus prompting the *catch* block which will print out nothing and move on smoothly to the next question. In the case of the time running out, I simply use *continue* since my traversal format through the list is a *for loop* in which skips the remaining code of the for loop and moves onto the next question. The *Read* thread is able to output the correct text line in this event as well. Things I can do to make this program better is to implement a restart as well as support for multiplayer.