ASSIGNMENT 4 REPORT: CONCURRENT PROGRAMMING

OCTOBER 29, 2019

CLNSIH001
Sihle Ca

ASSIGNMENT 4 REPORT: CONCURRENT PROGRAMMING

Table of Contents:

1. Description of classes

I modified the WordApp class so that the start button would actually initiate the game causing the words to fall. I also added a function to the text input box so that it would be able to compare the word inserted to the words on screen. The End button now serves a purpose as it can now end the game. As a result of this it also gives the start button an additional feature. The start button can now also serve as a restart button after the end button has been pressed. I also included a third button, Quit, that allows the user to exit the game. I added a method that sets the values of each of the counters. There is also a confirmation method that I have added that I will speak more about in section 6. I have also created a feature that displays a message on completion of the game.

I also added code to the run method of the WordPanel object to aid with the registering scores and the animating of the words

2. & 3.Description of concurrency features & Thread Safety / Thread Synchronization

In the actionListener for the start button I created multiple threads a and started them. The threads and store them in an array. This is crucial later for when we want to kill these threads when we press the end button. I then started the thread which implemented the run method of the word panel class. Each thread becomes a column of the word displayed on screen. I use the the volatile Boolean (called done) to control the threads in the while loop of the run. While done was false the word in the particular column for each thread would continue to fall and regenerating and falling. I used a lock on the drop function of the word (which is of the type WordRecord) as drop is a mutable method in the wordrecord class that was also contained in a lock. So re-entry locks are used by each thread to ensure thread synchronization. More than one thread will be using the drop function but never for the exact same word in the exact same position as another thread. Other functions such as the resetWord and WordApp Counters method were also put in re-entry lock to ensure no two or more threads are setting the word or the score values at a time to avoid race conditions, more specifically bad interleavings. For the score class I made the caught, missed and gameScore atomic Integers to ensure that if certain process (like incrementing values) are either completely done or not done at all to avoid other threads from interrupting said processes. Everytime a

the getTotal of the score class reached the total words the volatile Boolean called done was made true. Since it is volatile, it retains the value set by the lastest thread. The condition for the while loop in the run method of the WordPanel class only applies as long as done is false. Since we exit the while loop the run method is excuted and the current thread is terminated.

For the end button I interrupted the threads stored in the array from earlier. Since the run() method in WordPanel catches the thread interrupted exception I made the particular thread reset the word before breaking from the while loop to end the thread.

4. Debugging techniques

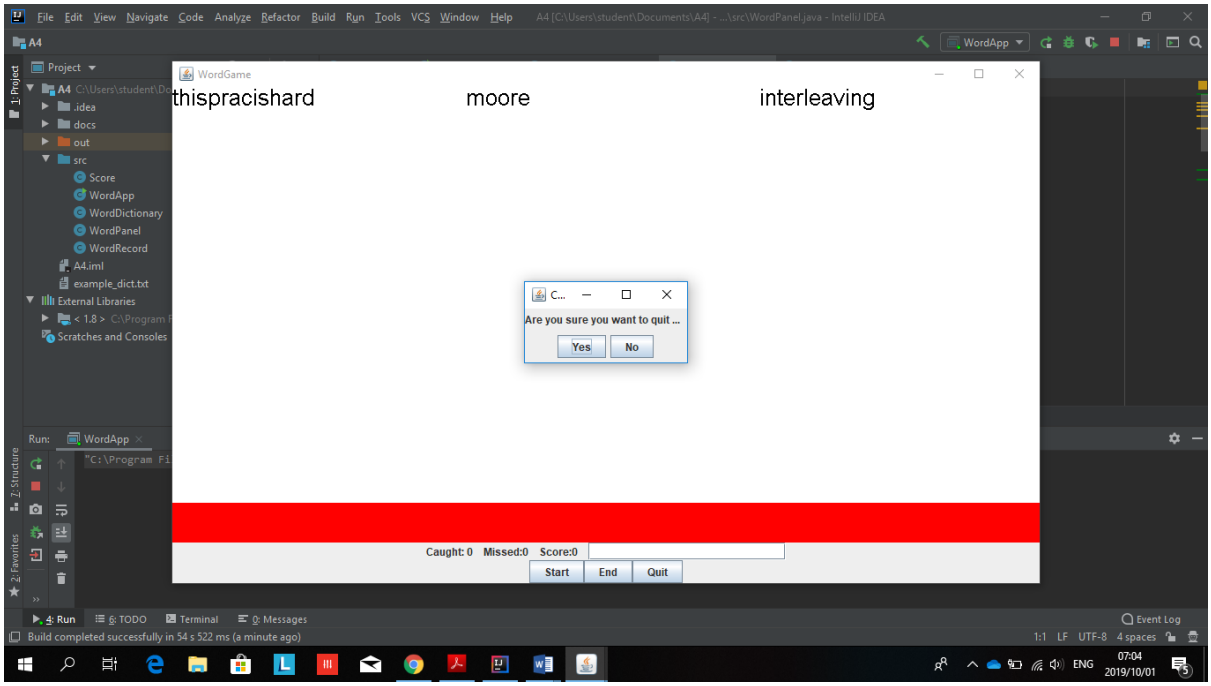
I used print statements to check the states of certain threads and specific points. At one point in my run method one particular thread was taking long to process and would get interrupted causing the word to stay at the top of the screen. After using synchronization the problem was averted. I also but print statements after the end button in WordApp class the ensure the threads stored in the array were terminated

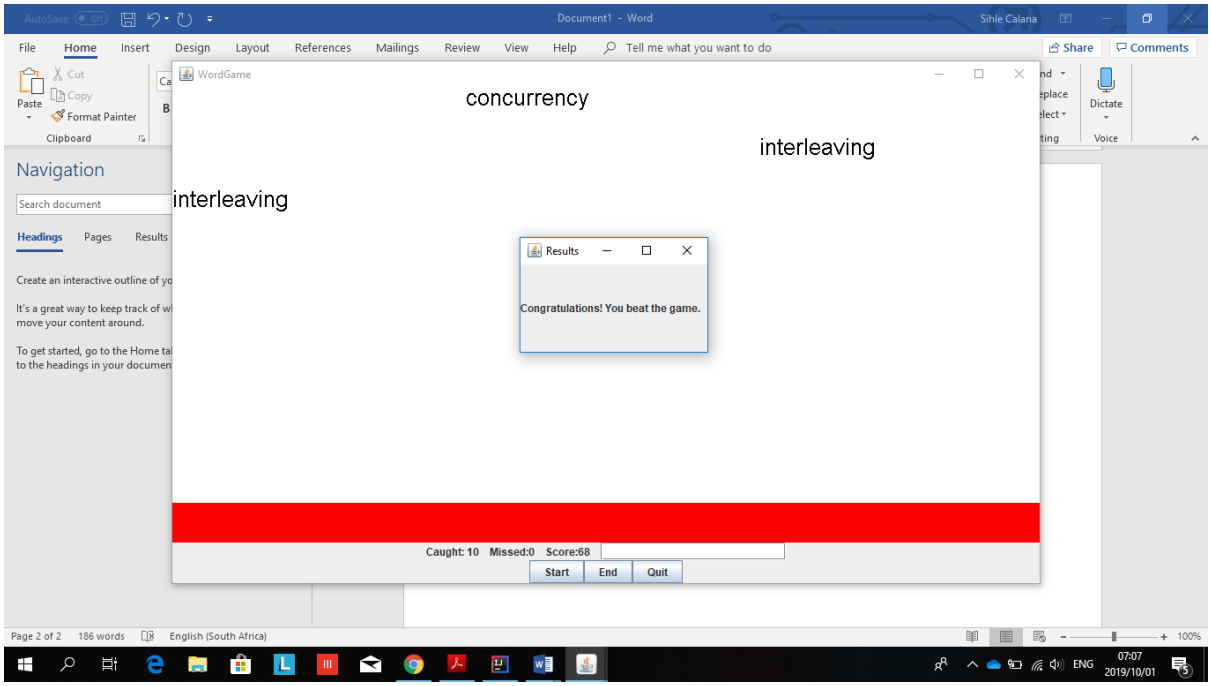
5. How the design conforms to the MVC design

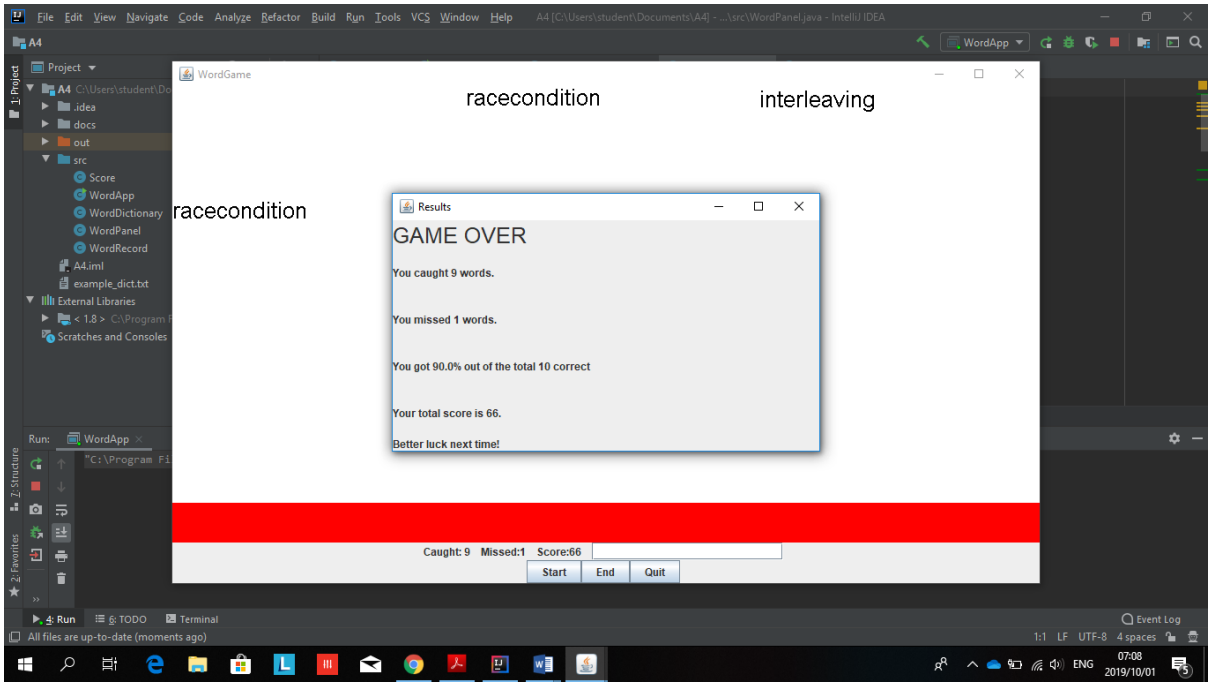
The model comprised of the WordDictionary class, the array of WorldRecords and the Score class. View was Mostly the WordApp class and the controllers were the threads that ran the WordPanel class

6. Update to the original idea of the game

I created a gui to confirm with the usere whether or not they were ready to exit the game. Prompting the user may help in the case where the user accidentally taps the quit button. I also have different guis the are displayed depending on the players performance as seem by the images below.


WordGame
thispracishard
moore
interleaving
Are you sure you want to quit ...
Yes
No
Caught: 0 Missed:0 Score:0
Start
End
Quit
Build completed successfully in 54 s 522 ms (a minute ago)


Document1 - Word
WordGame
concurrency
interleaving
interleaving
Results
Congratulations! You beat the game.
Caught: 10 Missed:0 Score:68
Start
End
Quit
Navigation


racecondition
interleaving
racecondition
WordGame
Results
GAME OVER
You caught 9 words.
You missed 1 words.
You got 90.0% out of the total 10 correct
Your total score is 66.
Better luck next time!
Caught: 9 Missed:1 Score:66
Start
End
Quit