|  |
| --- |
| THNGEO002 |
| Concurrent Programming: Falling words |
| A multithreaded Java program, ensuring both thread safety and sufficient concurrency for it to work well. |

|  |
| --- |
| Georgeo Thanathara  9/29/2019 |

Table of Contents

[Classes and Modifications 3](#_Toc20654303)

[wordThreads.java 3](#_Toc20654304)

[wordEntry.java 3](#_Toc20654305)

[WordApp.java 3](#_Toc20654306)

[WordPanel.java 4](#_Toc20654307)

# Classes and Modifications

Please note these are just descriptions of what the classes do, The handling of the concurrency safety aspects are talked about later.

## wordThreads.java

I created a wordThreads class that implements the Runnable such that the instance of this class can be executed by a thread. Runnable objects are much more flexible that extending the Thread class, this is because Runnable class’ allows extending from any other class.

The run method of this class contains the code that needs to be executed by the running Thread. This run method handles the dropping of the words on the screen by getting the getSpeed method in the WordRecord class, this allows each word to drop by a random amount. The gui is then repainted and the thread is put to sleep for 20 milliseconds.

The wordThread also handles the missed counter, by first checking if the words have reached the red bar on the bottom of the screen, If it has, the words are rest and brought to the top while the missed counter increments. But during this check the run method also checks whether the maximum number of words has been reached. In the case where it has been reached, a message window pops up on the screen showing the user their results of the game.

## wordEntry.java

This is an inner class that I created In the WordApp class that also implements the Runnable interface. The run method in this class ensures that when a word is typed, it is compared against all the current words on the screen and depending on whether the correct word was typed, it will increment the caught and the score.

Like the wordThread class , this class also does a checking to see whether the maximum number of words has been reached by using the getTotal method within the Score class and similarly a message window pops up giving the user the results.

## WordApp.java

The WordApp class was modified such that GUI is responsive to its respective buttons.

* **Start :** The start button on the screen first sets the volatile Boolean “check” in wordPanel to true (More about this later) and creates a thread while passing in the Runnable wordPanel object. The thread then begins executing by using the start Method. The visibility of the start button is then set to false such that it cannot be clicked again.
* **Pause :** The pause button only sets the check variable to false such that the items on the screen stop moving. It then sets the visibility of the start button to true to be able to continue the game from where is was stopped.
* **End** : The end button stops the game by setting the Boolean to false, A message dialogue then pops up showing the users results for the current game. When “ok” on the message dialogue is clicked, the screen is reset and the visibility of the start button is set to true.
* **Quit :** The quit button is to exit the application and end the program. To do this the dispose method, followed by System.exit is used to ensure safe quitting.

## WordPanel.java