# Assignment 2

I did not create any new classes I only modified existing ones.

Modifications to the WordApp class:

- Imported ExecutorCompletionService.
- Changed all JLabels to static so that they can be accesses and modified in the WordPanel class.
- Created a static ExecutorService and CompletionService object.
- Added a method called setupMessage that pops up when the totalWords is equal to the caught+missed values.
- Added a setter method that sets the text entered by the player to a string value.
- Added a JButton called quit in the setupGUI method that adds a quit button which when pressed exits the game.
- Created a ThreadPool and ExecutorCompletionService when the startButton is clicked.
- In the end button I shutdown the thread pool that was created and awaited its termination.

Modifications to the WordPanel class:

- In the paintComponent method I updated the caught, score and missed texts displayed on the panel
- If the game was done running, the screen is repainted as a blank rectangle.
- A loop method was added that controls the dropping of the words and checking if if the text matches, if the word is in the red zone and if the total words is equal to the caught + missed.
- The run method inludes calculating the thread number from current thread. It uses a while loop that is dependant on the volatile done to determine if the loop method should run. If done is false, the loop method tunes, the frame is then repainted. Thread.sleep was added to slow down the speed at which the threads are executed.

All the setters and getters were modified with the synchronized modifier. This ensures that there is no interleaving between threads setting and getting various varibales. In order to run the threads the ExecutorService was used to create a pool of threads that ran on the WordPanel. A completionService was then used to submit all the threads which essentially runs them. A volatile modifier was used when declaring the boolean done in both the WordApp and WordPanel class as this is used to enusre that all threads when reading this variable will read the most recent updated done. Threads were put to sleep to slow down and animate the actual moving of the words.

Thread safety was ensure by synchronizing all setters and getters in the classes. You need to protect data when threads are going access the same data simultaneously. When accessing or writing data simultaneously the result of errors from this situation is known as interleving. By syncronizing all getters and setters it interleaving could not take place as when one thread calls a getter or setter method only that thread can use it during that period of time.

No thread synchronization was used. To ensure that all threads have executed to completion (no live threads), the executorcompletionservice called the method awaiting() which waits for all threads in the thread pool to complete their execution. As a result of this there is no liveness and no race condition.

It is almost impossible to check for race conditions as there is simply too many possible outcomes for the execution of threads. Therefore I tried to use logic as mcuh as I could to ensure that there is no race condition that is created when the threads are running.

The model was the WordApp class. This class created the GUI window and all the panels and buttons on the window. The controller was the WordPanel class that was the runnable thread. This class controlled the data flow into the model object and updated the view whenever the data changed. It kept the view and model seperate.