Introduction:

The assignment purpose was to create a typing game through the use of parallel programming while making sure that the thread safety is insured through sufficient use of concurrency therefore the game runs error free.
This report entails the classes used, and discusses concurrency and how thread safety is ensured and race conditions are prevented. Furthermore we discuss the model used that is the Model View Controller pattern and the additional features we added to the game.

Class Description:

Word Panel class: This class was mainly used to paint the GUI and run the threads that caused the words to fall. I created an array of Thread objects of the size of noWords(words on screen) and an array of controller object of the same size. The reason I choose to have an array of threads was to allow me to create a thread for each word on the screen and hence manipulate the thread according to needs and user input. Hence for each thread to run there has to be a controller class object.

The constructor of the WordPanel object is fairly simple. It stores the score of the game, populates the word array , the length of the array, the maximum y value and a boolean flag done. The WordPanel class makes use of 3 flags, done,complete and flag. Done is used to check if all the threads have been made for the total number of words. Complete is used to check if all the threads have executed and are done. Flag is used to end the program when the user presses the end button.

The *paintComponent() method* was used for drawing the words to be displayed on the GUI canvas on the screen. It did this by placing each word on its respective x and y values. The words were painted at different positions and times which you will see when we discuss about each thread later on.

The WordPanel class also contains the run( ) method that is called the the WordApp and is used to start the game and hence the threads. In this method the thread and controller class(more on this class later on) object are initialised to the size of noWords and all the flags are set to false. After this all the threads for the number of words to be displayed are created and started. The run method also ensures once the thread has completed its execution a new thread is created for the next word to be dropped and this cycle continues until all the words have been dropped or the user presses end or quit.

Another method used is userInput() which is called when the user enter a text and checks if the text matches with the words currently dropping. If the users input matches any of the words dropping then those words are removed from the GUI .Since the question did not specify anything about repetitions I assumed that if the user enter text and 2 or more of those words are dropping then they are all removed and counted as a caught words/

The word panel have few other methods that are used to end and reset the WordPanel according to need.

Controller class: I created this class in order to create threads of this class that implements runnable which will cause each word to drop independently at their own speed. This class contains a constructor which initialised the word , WordPanel and Score.

It has a run() method which first sets the flag to false before starting. Then it continues dropping the word and repainting until the flag has been raised to stop or it has reached the bottom, Once the loop is over we check if the word has reached the bottom in order to update miss and reset the word.

Score class: I did not add any methods to this class however I changed the variables caught, missed and gameScore to AtomicInteger instead of int to implement concurrency(Further on this below).Atomic is implemented to ensure that the variables will be atomically updated when required. Also I made the get and set methods synchronized in order to ensure thread safety(which will be discussed later).

3. Java Concurrency and thread safety features:

What is concurrency?

Concurrency is the ability of different parts or units of a program, algorithm, or problem to be executed out-of-order or at the same time simultaneously partial order, without affecting the final outcome. This allows for parallel execution of the concurrent units, which can significantly improve overall speed of the execution of the program[1](from Wikipedia)

3.1 When to ensure thread safety:

We need to be worried about thread safety when several threads are editing/changing the same variable as the changes can be lost unexpectedly since the threads accessing the same memory. Furthermore if you pass in an object to the method and the method mutates the object and before the method completes you call the same method again on a different thread but with the same object, then you have a problem and thread safety can not be guaranteed.

We do not need to be worried about thread safety when a variable is thread local, it is protected using locks or when they are immutable.
If a Variable is thread local it can only be accessed by one thread which will help avoid any form of conflict between other threads which use their own local variable. Hence preventing any data races.

When we use locks it ensures that only one thread can access that variable at a time and the value is synchronized across the memory therefore preventing data races.
Immutable variables mean that their values remain fixed therefore when threads access these values since they do not change the program will not face any problem and thread safety will be kept.

To avoid problems above we use certain features to ensure no data races occur. These techniques are explained below.

3.2 Features used:

The Score class required some modifications as many threads will be accessing it and hence to ensure thread safety I used *synchronized* for the get and set methods which enabled those functions to use locks so that 1 thread can access those methods at a time. Also in this class t he caught,missed and gameScore are initialised as AtomicIntgers which minimizes synchronization and help avoid memory consistency errors. Hence, it ensures synchronization. Atomic variables allowed for incrementation of the variables without any interruptions.

Synchronization works by first obtaining the lock, in this case the Score object, and synchronizing local thread memory with shared memory. The operation is performed and written back to shared memory. The lock is then released. [1]

Both run() methods made in the Word Panel and in controller have a flag of boolean type which ensure that the loop would break when certain criteria were met. To carry this out swiftly and efficiently without causing any data races I used volatile .Using volatile variables reduces the risk of memory consistency errors, because any write to a volatile variable makes every thread actually access the value each time from memory thus getting the newest value.

3.3 Liveness :

In concurrent computing, liveness refers to a set of properties of concurrent systems, that require a system to make progress despite the fact that its concurrently executing components ("processes") may have to "take turns" in critical sections, parts of the program that cannot be simultaneously run by multiple processes(from Wikipedia page Liveness).

This means that the concurrent program must be able to execute in a timely manner. In my program I have effectively used synchronization(only for Score) and volatile variables such t hat the program works efficiently and performance of the code is excellent with no time delays.

3.4 Deadlocks :

Deadlocks occur when there is a cycle of threads waiting for a resource held by another thread. After carefully going through the code I have ensured that no two locked methods are dependent on each other hence there is no cycle hence preventing a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

4. System validation :

Ensuring that no data races occurs or thread safety is ensured completely is very difficult to prove. However I carried out both black box and white box testing. White box testing was done by making my room-mates and friends play the game multiple times while I observed and recorded the results. The game was ran a large number of times with different command line arguments and speeds. All the times the code executed accurately with no errors.
Black box testing was carried out by me where by I ran the code myself and compared the results with a truth table to see if the expected results were achieved.
Additionally I went through the entire code and checked through the concurrency to ensure thread safety and smooth running of the game.
Through the use of experimentation I can say my game works smoothly without any for of interleaving or data races.

5. The Model-View-Controller pattern :

As required by the question the game was developed using the Model-View-Controller pattern. The display of the information/game is separated from the manipulation and internal affairs of the program.

In our case the Model is made up of the classes Score, WordDictionary and WordRecord. The view is the canvas displayed which is the GUI which is set up by the WordApp class . Furthermore the text entered and button are implemented in the WordApp that is part of the view. The Controller is implemented by the controller class and WordPanel I created that is responsible for the game functionality.

The controller class makes the words fall independently until total number of words fallen have been met or end button pressed. If any word missed the model is notified which then updates the view.. From the view when the user enters text that word is sent to the WordPanel which compares with all the words falling and if met then goes to the model to update the scores and reset the words. This updates are then displayed on the view.

6. Additional Features to the game :

Firstly I have created the game such that the end button can only be pressed once the game has started and like wise the start button can be pressed once a round has ended. This prevents from any complications occurring when start is pressed twice.
Furthermore after running the game once the start buttons text changes fro “Start” to “Restart”.

Secondly I have added sound effects for when the user misses a word that is once the word reaches the end and disappears. Also I have added sound effects for when all the game is over and all the words have been dropped.

Another feature I added is once a round has been completed a pop up message appears that notifies the user the game has ended and displays the scores and words missed and caught. Also I added a small green rectangle at the top to symbolize the start of the word drop.

7. References:

[1] Volatile and Synchronized Keywords in Java. https://dzone.com/articles/difference-between-volatile-and-synchronized-keywo. Accessed: 2019-09-27.