a) Description of **HungryWordMover** class

The **HungryWordMover** class is similar to the regular **WordMover** class except it takes in an extra parameter for the constructor which is a list of Falling Words. This is intended to pass the current Falling Words on screen to check if they collide with the current Hungry Word.

The run method checks if the game is not done, the word is not off-screen and the word is not waiting and then animates the word across the screen. It checks each time through the list of falling words to check if one of them collides with the Hungry Word; if it does, the Falling word is reset, and the missed counter is incremented by 1. If the game is not yet done and word slides of screen, the Hungry Word is reset and the missed counter is incremented by 1 then the **HungryWordMover** object makes the Hungry Word start waiting. When the Hungry Word is waiting the **HungryWordMover** object sleeps for a random duration ranging from 0 to 10 seconds then stops waiting and resumes.

b) Other classes added

The **HungryWord** class was also added and it is similar to **FallingWord** class except that the x and y coordinates for almost every method are reverse since it moves horizontally not vertically. The class was made to ensure that the implementer of the horizontal moving Hungry Words would not have confusion on which parameters to change when needing a word to go horizontal or vertically. The **HungryWord** class also has a *.collides(FallingWord fWord)* method to check if the Word collides with a Falling Word

c) Changes made to existing classes

The package declaration was removed from all the files and they file structure was reorganised

Class	Changes Made
CatchWord	- now an array of HungryWords as an instance variable and also has the *setHungryWords* method to set them - The run method has been changed to check for the HungryWord first in order to remove it before FallingWords. - When the run method checks for FallingWords it iterates through the list from the first match to check if there is another match which is lower then resets the lower one.
FallingWord	- Now has a *getBox()* method which gets the coordinates of the corners of the word, which is used for collision checking with the HungryWord. - The *matchWord()* method was also changed to get it not the reset a word on match since this is being handled by **CatchWord**.
GamePanel	- Now takes in an array of HungryWords to render the Hungry Word on screen - Creates borders around the screen on top of everything to make them the final layer. - Made the font monospaced because other fonts will make collision look like it is off when in fact collision calculates the length of the based on the average length of a character - The screen colour changes to green upon winning and red upon losing (cosmetic change) - Has *getValidYpos()* for the HungryWord

TypingTutorApp	- In the *createWordMoverThreads()* method Hungry Words and their mover threads were created - There are now separate word dictionaries for the HungryWord and FallingWord to avoid words appearing in both lists
WordDictionary	- Added a new word array for the HungryWord to pick from. - Created a default constructer that works as normal and another constructor that takes in the parameter to select what type of word it's creating a dictionary for. - Also changed the dictionary to be an instance variable not a static one to make them unique to each thing.

d) Race conditions identified in original code

There is a race condition in the TypingTutorApp in that when the start button is pressed once then again, the current existing threads are not killed and hence even more threads are created by the *createWordMoverThreads()* function, thereby leading to threads that can edit the Score still being alive but not represented on screen. To illustrate this someone can run the game and press the start button more than once and the game won't operate as expected.