CSC2002S Java Threads Module
Assignment 2 (Concurrency)

Due Date: 28th August 2014, 9am

In this assignment, you will design a multithreaded Java program, ensuring both thread safety and sufficient concurrency for it to work well.

1. Problem Description

You will implement a multithreaded typing game (Fig.1).

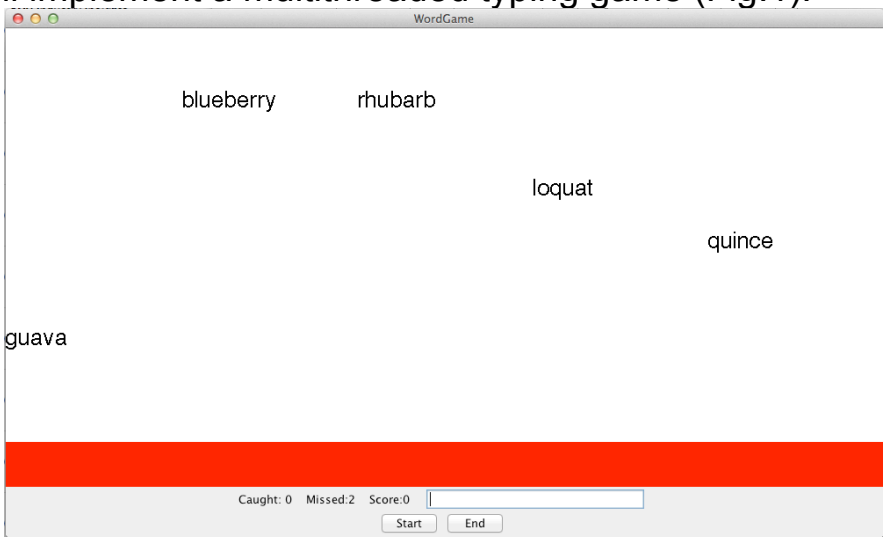


Figure 1. The main GUI window for the typing game. Note that this mockup is missing the required "Quit" button.

This game should operate as follows:

- When the start button is pressed, a specified number of words (a command line parameter) start falling at the same time from at top of the panel, but fall at different speeds – some faster, some slower.
- Words disappear when they reach the red zone, whereupon the *Missed* counter is incremented by one and a new word starts falling (with a different speed).
- The user attempts to type all the words before they hit the red zone, pressing enter after each one.
- If a user types a word correctly, that word disappears, then the *Caught* counter is incremented by one and the Score is incremented by the length of the word. A new word then starts falling (with a different speed).
- If a user types a word incorrectly, it is ignored.
- The game continues until the specified maximum number of words (a command-line parameter) is reached (whereupon a suitable message should be displayed) or the user presses the End button (which should simply stopp the current game and clear the screen). The user can then play again, beginning a new game by pressing the Start button.

- The user presses the Quit button to end the game (not shown in Fig.1).
-

You are provided with skeleton code for the assignment (package skeletonCodeAssgnmnt2). When executed, this skeleton just displays an incomplete GUI interface, showing the specified number of words, the score and some of the buttons. There is no animation and it does not do anything useful. You must build on the skeleton, improving, adding threading and ensuring thread safety when necessary. You must use appropriate synchronization and your solution should allow for maximal concurrency: operations should not be serialized unless necessary.

2. Requirements

2.1 Input

Your program should take as command line parameters:

- The number of words that will fall
- The number of words displayed at any point
- The name of a file containing a list of words (one per line) to be used as a dictionary for generating words. The first line of the file will be the total number of words in the file. An example is provided for you (example_dict.txt).

The user types words as they fall, so there should be a field to allow entry of words. (The skeleton provided actually already handles all of this.)

2.1.1 Controls

Your program needs to have a start/restart button (which will start a game from scratch) and a button to halt the current game (but not quit) and a quit button (all except the quit button are present in the skeleton).

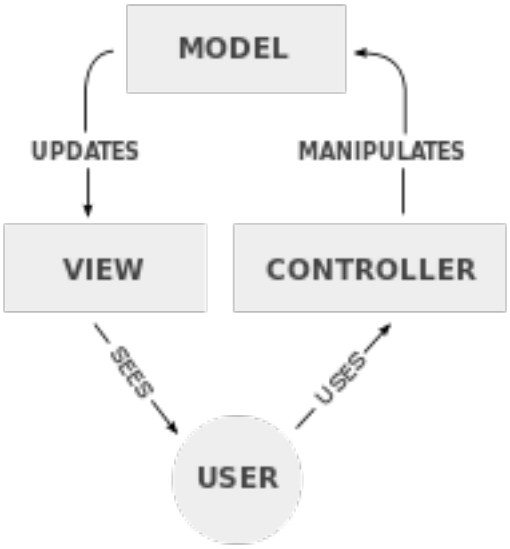
2.2 Output

The GUI needs to animate the words falling and keep a running total of words “caught” (typed correctly), “missed” and a total score. The score for typing a word correctly is equal to the length of the word. Therefore “bat” would score 3, “amazing” would score 7 (the skeleton Score class already handles this).

Once the specified word limit has been reached, an appropriate message on the user’s performance should be displayed.

2.3 Code architecture

When extending the game code, you are expected to follow the Model-View-Controller pattern (shown in Fig. 2) for user interfaces. This very common pattern for software architecture separates the internal representation of the information from the display of the information to the user.



3.

Fig. 2. The Model-View-Controller pattern has a clear separation between the display of the information (model) and its internal representation.

In this case, the model comprises the classes WordDictionary, the array of WordRecords and the Score. The view is the GUI (which must still be animated) and the controllers will be the threads that you add to alter the model and the view, moving the word positions, performing the animation, adding and removing words as necessary and updating the counters and the score.

3.1 Report

You need to write a concise report detailing and explaining the coding you have done. The report must contain:

- A description of each of the classes you added and any modifications you made to the existing classes.
- A description of all the Java concurrency features you used and why they were necessary (e.g. atomic variables, synchronized classes, synchronized collections, barriers etc.).

- You will need to explain how you wrote the code to ensure:
    1. thread safety (for both shared variables and the Swing library). You should describe when you need to protect data and when you don't – and explain why.
    2. Thread synchronization where necessary
    3. liveness
    4. no deadlock.
- An explanation of how you validated your system and checked for errors (esp. race conditions).
- An explanation of how your design conforms to the Model-View-Controller pattern.
- Any additional features/extensions to (or improvements on) the basic game that you think merit extra credit. There are many things that you can do to improve this game. However, the game must still conform to the basic operations set out above.

3.2 Handins

You are required to hand in both the entire code for your system and a concise report describing the work done.