CSC2002S 2022
PCP Part 2: Concurrency
VERSION 2

Assignment: Extending the Multithreaded *Typing Tutor Game* with a Hungry Word

Introduction

This assignment aims to increase your understanding of thread synchronisation and thread safety. In this assignment, you will **extend** a multithreaded educational "Typing Tutor" game coded in Java, so that it has **additional features**.

The Typing Tutor game challenges the player to type in the words displayed before they hit the floor.

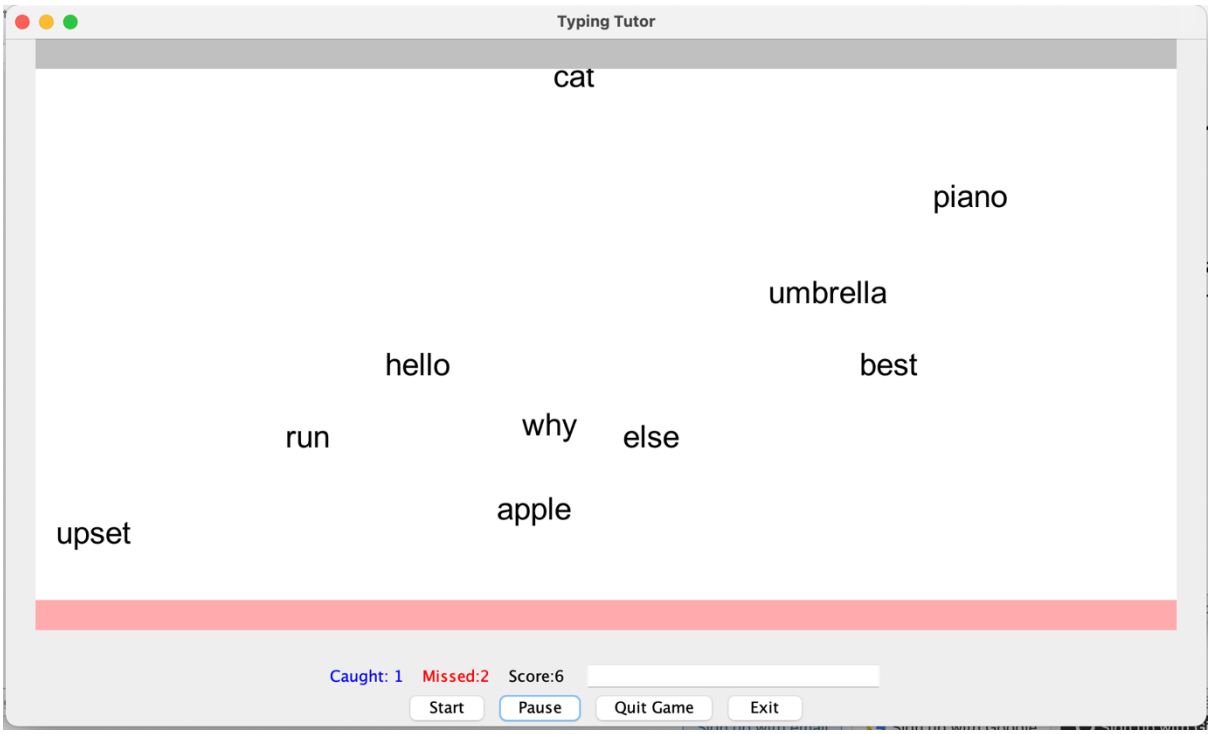

Figure 1: The Typing Tutor App, in the middle of a paused game.

This game operates 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 pink 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 pink 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** counter 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 either of the following situations.
    1. The specified maximum number of words (also a command-line parameter) have been typed in correctly. When this occurs, the user has won and the screen displays "Well done!"
    2. The user misses three words. When this occurs, the user has lost and the screen displays "Game Over!"
    3. The user presses the ***Quit Game* button**, stops the current game and display "Game Over!".
- The user can start again at any time, beginning a new game by pressing the ***Start* button**.
- The user presses the ***Exit* button** to exit the app completely.
- At any point, the user can press the ***Pause* button** to halt the game. The game then restarts from the same point when the ***Start* button** is pressed again. However, they user should not be able to type in words (i.e. cheat) when game is paused.

The TTworking.jar program provided with the assignment illustrates a working game.

Your task in this assignment will be to **add two new features** to the program:

1. Ensure that, when duplicate words appear on the screen, that the **lowest word disappears first** (this is not currently supported by the game).
2. A green word – Hungry Word - that is selected randomly from the dictionary, appears randomly and moves horizontally across the middle of the screen. Any words that Hungry Word bumps into disappear (and their total is added the *Missed* counter). If the user types the Hungry Word then it disappears and the total is added to the *Caught* counter. If the Hungry Word exits before being typed, the total is added the *Missed* counter.

These additions should be made *without breaking the existing code*: it should still operate as specified above.

Assignment Specifications

You are given the Typing Tutor (TT) program (which may be downloaded from the assignment page). This program takes the following command line parameters (in order):

- The number of words that will fall in total (`totalWords`);
- The number of words falling at any point in the game (`noWords`, less than `totalWords`);

- 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 is the total number of words in the file. An example is provided for you (*words.*txt in the code main folder), but the program will use a default list if you do not provide one.

The TT program then creates a number of threads and has a number of shared objects. The game has all the code completed (and all the necessary threads). You need to examine the code, work out what it does and how to ensure that the **lowest word disappears first** and **how to add the Hungry Word.** For the latter, you need to add an additional class to the game: `HungryWordMover.java`. Adding this class will necessitate changes to the other files. When these changes are made, you need to ensure that the game still behaves as specified above and that the buttons still work.

You must then write a short expnalation of the changes you made (to help the tutor), including

a) A description of the HungryWordMover class.
b) A description of any other classes you added (and a justification of why they were necessary).
c) A broad description of modifications you made to the existing classes and how you ensured thread safety.
d) Any race conditions that you identified in the original code (for **extra credit)**.

NB: You can of course add extra methods, objects etc. to the code. You may add extra classed to the code, but they must be justified (and kept to a minimum).

Marking and submission

You will need to submit an **assignment archive**, named with your **student number** and the **assignment number e.g. KTTMIC004_CSC2002S_PCP2.** Your submission archive must contain

- **your Typing Tutor program with the changes and the additional** `HungryWordMover.java` class
- a **Makefile** for compilation
- a **short list of changes (in pdf or plain text format)**
- **a GIT usage log (**as a .txt file, use git log –all to display all commits and save).

Upload the archive file and **then check that it is uploaded.** It is your responsibility to check that the uploaded file is correct, as mistakes cannot be corrected after the due date.

The deadline for marking **queries** on your assignment is **one week after the return of your mark.** After this time, you may not query your mark.