CSC2002S CONCURRENT PROGRAMMING

ASSIGNMENT 4

SEPTEMBER 30, 2019

TAWANDA MUZANENHAMO
MZNTAW004

Contents

1. Introduction

The aim of this assignment is to design a multithread falling words game to ensure both thread safety and sufficient concurrency for the game to work well. Concurrency helps ensure that programs execute different tasks efficiently whilst managing data resources.

2. Method

In order to complete the assignment, classes provided in the skeleton code were modified and new classes were made.

2.1 Modification on the Skeleton and Additional Classes

i. Score.java

The score class is made up of getters and setters which are used to set values for caught, game score. I also added a method to count the number of words entered correctly.

ii. WordApp.java

In the WordApp, actions performed by the start, pause, end and quit buttons were added. The start button is meant to start the game and words start falling at different speeds when the button is pressed, it can also be used to start the game when paused. The pause button is responsible for stopping the words from falling whilst maintaining the current score. The end button is meant to stop the current game, clear the screen and reset the scores. The quit button is meant to stop the game and close the window.

iii. WordRecord.java

In the WordRecord, a new equals method which enabled me to match words more efficiently whilst also matching with the word closest to the redline if the same word appears more than once amongst the falling words.

iv. WordPanel.java

In the WordPanel, a run method was added to animate the game. The method calls on the start method from the controller class in order to begin the game, if there are changes in the panel i.e. a word was caught it repaints the panel to show the new changes.

v. Controller.java

The controller class is responsible for controlling the game depending on the current state of the game i.e., if there are any changes on the panel, the controller detects them and updates the panel accordingly. The Controller is also responsible to handle when the player has either won or lost the game and showing a message with information on the player score, caught words, missed words and incorrectly typed words.

vi. Threads.java

This is the class that's responsible for the dropping of the words on the game panel. The Threads instantiated by this class runs as long as the game has not been stopped.

2.2 Concurrency Features

i. Synchronization

Synchronization allows the use of re-entrant locks during the modification of data. In order to prevent bad interleaving, other sub methods were also synchronized in order to ensure that threads do not get access to the same resources until current thread with access to the resource is done executing.

ii. Volatile

The volatile key word on variables is used to indicate that values of the variable are not thread-local hence dealing with variables accessed by different threads i.e. ended, running or paused.

iii. Atomic Integers

The atomic integer class is used because it protects the underlying by providing methods which perform atomic operations on the value.

2.3 Handling Different Threading Problems

i. Thread Safety

Thread safety refers to a programming concept which is applicable when implementing multithreaded programs. A program is deemed thread safe if and only if it functions correctly when multiple threads are executing for example when multiple threads access shared resources.

The score class is accessed by many threads hence to ensure safety, the getters and setters in the class are made mutually exclusive by making them synchronized.

ii. Liveness

Liveness means that something good eventually happens. Liveness failures like contention for access to data can be avoided by protecting the data through reducing the scope of the lock, reducing the number of times a lock is acquired.

In the controller class only parts of the program like the updateScore were locked hence allowing other threads to access the parts of the class.

iii. Deadlocks

Deadlocks occur when a thread is waiting for an object lock that is acquired by another thread and the second thread is waiting for an object lock that is acquired by the first thread leading to both threads waiting for each other to release the lock.

Deadlocks are prevented by ensuring that there are no nested locks i.e. no nested synchronization or calling of one synchronized method from another. The score class and the has the synchronized getters and setters.

2.4 Validation and Error Checking

In order to validate the system, the features of the system are checked to see if the conform to the requirements of the system. The game was started with different number of words from the file as well as words displayed in the game panel and the words conformed to having different speeds as required. Different words were captured as well as missed and the labels were updated accordingly. The buttons were pressed, and they all conformed to their required functionality.

Possible race conditions were avoided by using synchronization blocks on the getters and setters.

3. Model View Controller Pattern

In order to implement the model view controller pattern, the classed were made in such a way that the controller is used to coordinate events between the classes.

3.1 Threads.java and Controller.java

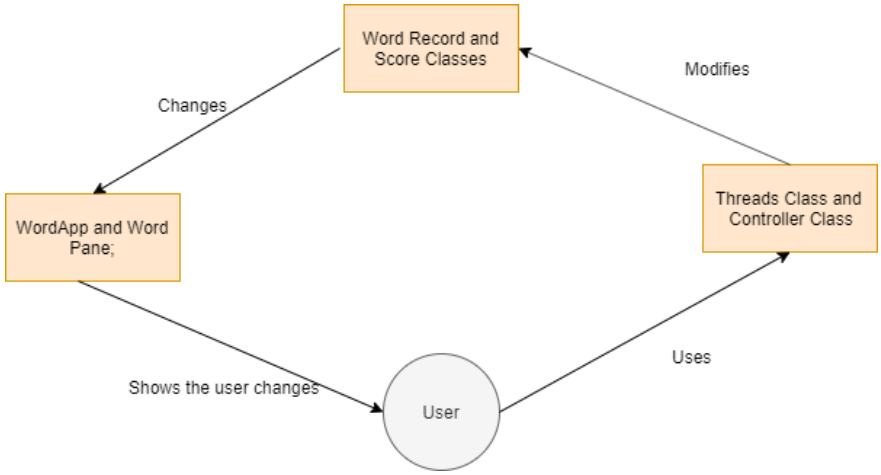
The threads and controller classes save as the controller in the MVC pattern. They aid in the changing of the model for example when the game has ended, the controller class is responsible for updating the view to show the player's results whilst the threads class includes the threads which are responsible for updating when words are missed and updating the scores.

3.2 WordPanel.java and WordApp.java

The wordpanel is the view of the system as it is updates whenever there are any changes in the scores. The WordApp is responsible for setting up the GUI i.e. creating dimensions, buttons, JPanels and frames.

3.3 WordRecord.java and Score.java

The WordRecord and Score classes are the modes of the system as they record values that are sent to the view from the controller. They are also responsible for managing the general logic of the entire system.



4. Additional Features

In order to make sure that the game is close to a finished product, the following additional features were added.

4.1 Pause

The pause button was implemented in order to allow the player to take a break from the game without losing their scores

4.2 Number of incorrectly typed words

The number of words typed incorrectly were implemented to help the player get a rough idea of how they performed.