CSC2002S

Assignment 4

JFFMAH001

Mahmoodah Jaffer

9-28-2019

Contents

Introduction

The aim of this assignment was to design a multithreaded Java program while ensuring thread safety and sufficient concurrency. When extending the skeleton code provided for the game, it was required that the Model-View-Controller pattern needed to be used for user interfaces. In this report I will discuss the classes that I added to the skeleton code in order to conform to the MVC pattern as well as any changes that I made to the code that was provided. All concurrency features that were used in the classes and why these features were necessary will be discussed as well as how thread safety was ensured. System validation will be discussed as well.

Classes

Existing classes

Score.java

In order to prevent having to synchronize the get and set methods in the Score class the variables in the Score class were made Atomic Integers as opposed to having the variables be normal integers. The methods **getTotal(), caughtWord()** and **resetScore()** were changed to be synchronized methods since these methods were not atomic.

WordPanel.java

In order to animate the game code was added to the game, code was added to the **run()** method provided to us. **repaint()** was used to update the screen and repaint the panel while the game was running.

WordRecord.java

Created an **equals()** method that overrides the actual equals() method to check that the words typed in by the user is the same as the word falling down.

WordApp.java

Code was added to create a pause and quit button. Code was added to create functionality for all buttons. Code was added in order to bring up a panel that shows the user their score once the game ends.

WordDictionary.java

No new code was added to this class.

New Classes

WordThread.java

This class was created to deal with all the words that were falling in the game. It deals with the functionality of the game in that it checks the state of the game continuously until the game ends.

WordControl.java

This class is meant to serve as the controller in the MVC pattern in that through this class many actions are performed on the model through the controller based on the state that the game is in.

Concurrency Features

Synchronized method

The reason why it was necessary to use the synchronized keyword for certain methods was because it allowed one thread at a time into a certain section of code thereby protecting variables/data (such as the array containing the words falling) from being corrupted by simultaneous modifications from different threads. [1]

AtomicInteger Variables

The **java.util.concurrent.atomic** package was used since it supports lock-free and thread-safe programming. Variables that would be accessed with different threads were made **AtomicInteger**'s instead of normal int's since this protected these variables from being accessed simultaneously by different threads (e.g. **missingwords,caughtwords** variables).

Volatile Variables

The Boolean variables used in the program were made volatile since it allows access to the variable as though it were in a synchronized block and therefore allowed the variables to be accessed by multiple thread objects.

Concurrency Code

Thread Safety

Data that needs to be protected is **shared data** – meaning data that can be accessed by multiple threads. It is necessary that shared data must be protected because if threads are trying to access the data simultaneously it can lead to race conditions – which can cause errors to occur and then the program would not function as it should. In my code I ensured thread safety by making use of atomic variables and synchronized methods.

Thread synchronization when necessary

Thread synchronization was only necessary for the array that contained the words that were falling as it needed to be accessed by more than one thread. This was accounted for in code by making use of the **synchronized** keyword for methods.

Liveness

The liveness of this concurrent application was ensured by using synchronized methods as ensured the timely running of the application and there were no obvious delays.

No deadlock

Since synchronized methods are being used in the code where necessary deadlock has been avoided as there are no threads that will be waiting for each other to finish.

System Validation

The game was played multiple times and no obvious errors occurred. In order to test thread safety (and in turn race conditions) the falling speed of all words were set to the game speed so that the words reached the bottom of the panel at the same time and the game still worked properly.

Conformation to MVC Pattern

The MVC pattern separates internal representation of the information from the display of the information for the user. This pattern is represented visually in the image below:

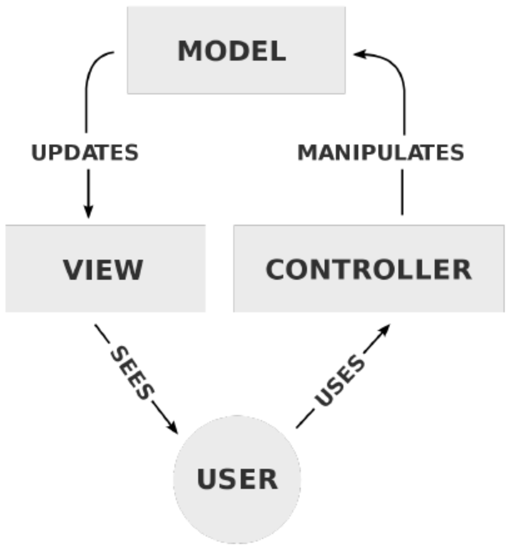

Figure 1: Visual representation of Model-View-Controller pattern

The model in the MVC pattern represents an object that carries data and it can also contain logic that updates the controller if it's data changes. In the case of the program for this assignment the model comprises of the classes **WordDictionary,** the array in **WordRecord** and **Score.**

The view represents the visualization of the data that the model contains and for this assignment the view is represented by the **WordPanel** class as it repaints the view seen by the user every time the model changes.

The controller acts on both the model and view and it controls the data flow into the model object, and it updates the view whenever data changes. In the case of this assignment the controller is comprised on the **WordThread** and **WordControl** classes. [2]

Additional Features

- A pause button was added so that the user has the option to take a break and not lose their progress in the game.
- Created a counter that counts how many incorrect attempts a user makes when typing in the falling words. This is displayed on the screen.

Conclusion

This assignment demonstrated the usefulness of thread safety and the importance of concurrency in terms of allowing a program the run smoothly (with liveness) and how to prevent race conditions from occurring.

References

[1] https://www.javamex.com/tutorials/synchronization_concurrency_synchronized1.shtml

[2] https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm