Name: Fardoza Tohab Student number: THBFAR002 Date: 5 September 2021

CSC2002S Assignment 2: Java Parallelism and Concurrency

Concurrent programming: Falling words

INTRODUCTION

The purpose of this assignment is to implement /simulate a falling words game using concurrent programming. Concurrent programming is about correctly and efficiently controlling access by multiple threads to shared resources. By implementing this, we need to ensure that our game/programs run both safely and efficiently when accessed by multiple threads. To ensure this I have made use of various methods including synchronization methods and shared variables.

CLASS DESCRIPTIONS

Below are the descriptions of each class used in this assignment.

Score.java

This class keeps a running total of words "caught" (typed correctly), "missed" and a total score. It also contains getters and setters for accessing, updating and incrementing *caught*, *missed* and *total score* variables .Most of this class remained as what was given in the skeleton code except that I have added synchronization to all getters and setters. This ensures atomicity via re-entrant locks.

WordPanel.java

The WordPanel class had some changes to it while most of it was kept the same. The WordPanel class implements the Runnable interface and had an empty run method. I added some features to it so that it would continuously update and refresh the word panel so that we can visualize the words falling. I added a while loop to check if the program was running and if this was true, I called the *repaint()* method and allowed the thread to sleep for 1000 miliseconds. I also added a *start()* and *stop()* method that were called inside the WordApp class' start and stop buttons. Inside the start() method, I created a thread object and called *start()* to execute the run method. Inside the *stop()* method, I called *thread.join()* so that each thread executing should wait for another thread to finish executing.

I also added some additional code to the for loop inside the painComponent() method to drop the words at a certain speed and to check if a word was dropped and reached the red panel, the missed counter would be incremented and the labels updated on the screen.

I also added additional features to improve the user interface by adding an image to the background and changing the colour of the words to a nice white colour.

WordRecord.java

This class essentially keeps a record of each word. It contains methods that reset the position of the word, reset the word as a whole, get the speed, to check if a word has been dropped etc.Most of the existing code in the WordRecord class remained the same. I only added a method to set the text on the screen to an empty string.

WordDictionary.java

This class essentially contains the default dictionary that will be read in, in case an error occurs with the text file provided in the command line argument.

WordApp.java

This class parses in the command-line arguments, sets up and stores the dictionary, creates the Graphical User Interface (GUI), and creates and stores the WordRecord objects. I added an extra method to update the labels on the display called *UpdateLabels()*. Inside the t*extEntry.addActionListener()* method I added in additional code to check if the words falling and text entered by the user matches and if it did the caught counter was incremented and the total score updated. I also added additional code to the buttons. In the start button, I added code to check if the program was running and if not I called the *start()* method created inside the WordPanel class to invoke the run method. I also added code to ensure that if the start button was pressed it would always reset the game from the beginning. In the end button, I added additional code to ensure that the program would stop for 1 millisecond and a message, "Game over!" would be displayed on the screen. The quit button essentially stops the program from executing, ends the game and closes the screen.

CONCURRENCY FEATURES

Synchronization methods

It was necessary to use synchronization methods in the Score class to ensure thread-safety which is of great importance. This is because in the Score class, multiple classes access the same data or share the same variables and all perform multiple actions on those data items. Using atomic variables in this case would not suffice since atomic variables is ideal to make a class thread-safe only when a single variable defines the class state. Hence, the synchronized keyword is used in defining all the getter and setter methods in the Score class.

The WordRecord class and some of the other classes contain certain variables that do not have thread-safety requirements. That is because the way my programs are structured guarantees that there will be no conflicts occurring.

Synchronization was of utmost importance in my programs since I didn't make use of any extra classes. Thus, I ended up with a lot of shared variables which needed to be synchronized.

Thread Safety and No Deadlock

Thread safety is ensured through good use of synchronization and shared variables which in turn solved the issue of deadlock.

MODEL-VIEW-CONTROLLER

My code conforms to the Model-View-Controller pattern in the following way:

- **Model**

The WordPanel class acts as the model in this game. It contains the paintComponent() method which is used to display contents onto the JFrame. It also updates the missed counter and updates the labels on the screen.

- **View**

WordApp class acts as the view since it updates the JFrame when it recieves the input fields and current statistics. It contains the *setupGUI()* method which sets up our user interface using the JFrame, JPanel and JLabel classes. It is also its own thread and also contains the buttons we are using and displays it on the screen. It also updates the caught counter and score on the screen.

- **Controller**

While acting as the view, the WordPanel class also acts as the controller since it implements the runnable interface. This is where I created my threads and updated them every time *repaint()* was called.

ADDITIONAL FEATURES

I made some improvements to my graphical user interface by adding an image to the background and changing the colour of the word to a nice white colour to complement the background image. I also changed the bottom rectangle to yellow to go with the theme set out by the image.