Assignment 2

By RBLCAM001 – Cameron Rebelo

Introduction

The aim of this assignment is to understand the concepts of concurrent programming by creating a typing game of falling words. Thread safety is an important factor to consider when doing multi-thread computing as many concurrency problems can arise from parallel programming. The approach that I took was to create threads of the WordRecord class that are each responsible for updating the y position of that instance after waiting the designating time by using thread.sleep(fallingSpeed). The WordPanel class then constantly repaints to update the position of the thread classes on the main frame, thereby making them fall down the screen. The score is kept track of by instantiating a variable in the WordApp class that is then parsed to each of the thread classes. Therefore, the Score class has to implement concurrency so that the threads don't cause any race conditions.

Classes

Score

This class serves to keep track of the users score and the number of words they've caught and missed. As this class will be accessed by multiple threads at the same time, concurrency had to be implemented here to ensure variable correctness. Because each thread has its own chance to update missed, caught and score – locks needed to be implemented on the get and set methods. This was done using the *synchronized* keyword which only allows a single thread to access a method at any given time. In addition to this, atomic variables were used on the instance variables to ensure that at no point during class creation or modification could interleaving cause the variables to be updated incorrectly. These atomic variables are added security, but the locks would've worked just fine.

WordDictionary

This class serves the purpose of creating a dictionary of words from an array of strings that is parsed to it. This dictionary is where the words that fall from the screen are chosen from. The class also has the functionality to get a new word to replace an old one. This method is synchronised so that there is no unexpected behaviour of two threads choosing the same newly random generated word.

This class was not modified from the skeleton code.

WordRecord

This class is responsible for creating threads of each of the words and updating their position to fall down the screen. When an instance of this class is created in a new thread, it is assigned a word and a randomly generated falling speed. Each time the word is reset, these two things will change. Here we have concurrency by means of thread local variables; each thread of this class has its own word, falling speed and coordinate attributes. This ensures that no threads will interfere with each other and cause deadlock.

This class was modified from the skeleton code to implement Runnable and make this the thread class of the program. The idea is that each falling word is its own independent thread that continuously updates its y position while that it is less than the maxY value (i.e. the bottom of the screen) and while the flag is not triggered. The flag is a Boolean that tells the thread to end which is used when the end button is pressed.

WordPanel

This class serves the purpose of creating a panel on which the words can be animated to fall. It sets up a new JPanel that is added to the JFrame form the WordApp class.

The class was modified to include a startGame and endGame function that start and stop the game loop. The startGame methos creates the appropriate number of threads and then starts them, and the stopGame method triggers the flag in the thread to stop it. The class itself is designed to be the view for the user. It implements an actionListener that is triggered every millisecond by a Timer. This actionlistner triggers the repaint method which redraws the words on the screen, the animation happens because each word has a different y value each time due to the threads updating it in the background; the falling speed that was randomly generated determines how quickly those y variables are updated.

WordApp

This is the main class of the program that is responsible for handling user input and initialising the frame and rest of the classes. It accepts command arguments that determine the number of words to win, the number of words on the screen at once and name of the file to read words from. The class then sets up a new JFrame and creates instances of the Score and WordPanel classes. Buttons are also added so that the user can interact with the frame and a text field is added so that the user has somewhere to type. When the start button is pressed the game begins and it is stopped when the end button is pressed or when the win condition is met.

The code was modified from the skeleton code to include a quit button, that exits the program. A flag was also added to check if the game is running to ensure the start and stop buttons can't be pressed twice in a row. A Timer was also added that, once the start button has been pressed, will constantly refreshes the Score, Caught and Missed Labels to keep them up to date and will also checks if the win condition has been met.

Concurrency

Score

Synchronized keywords were used on all methods. This is because many threads access one instance of this class and many updates and reads are done so the locks are necessary to prevent race conditions from happening.

Atomic variables are used to add extra concurrency measures when creating, accessing, and mutating the score, caught and missed variables.

WordDictionary

The synchronized keyword is used on the getNewWord method so that no threads can cause race conditions when accessing a new word.

WordRecord

This class is the class that becomes the threads that are used to update all the words and so all the variables used here are thread local to ensure that they don't interfere with each other and cause deadlock. The only references to other classes here are when the score variable is used to increment a missed word and when a new word is chosen from the word dictionary. Both of the accessed methods are synchronised.

Thread Safety

It is more important to protect the shared variables between the threads like the Score variables. These are most important to protect when incrementing the variables because if locks were not used then many threads may update the value at the same time and race conditions, more specifically data races might happen where missed might be read from and updated to at the same time. Since the WordApp is always refreshing to check the total words, the integrity of those variables is crucial to keeping the program running. It is less important to protect the data when displaying the variables to the screen as it isn't going to make the program crash or cause any error in the logic of the program.

Thread synchronization

The score class implements synchronisation to ensure that threads don't cause race conditions.

Liveness

No threads are dependent on other threads or on shared variables in order to run. The while loop of the thread only accesses and mutates variables that are thread local. When accessing outside the class for dict and updating missed, those methods have locks so no thread can be placed in deadlock.

Deadlock

The measures mentioned above prevent the program from entering deadlock.

System Information

Validation

The program was run 50 times to test that on average there were no concurrency errors. The expected output was always correctly displayed and there were no crashes therefore showing that no race conditions or deadlock occurred during these many trials.

Model-View-Controller

The WordApp class is the view and controller as it displays the graphics to the user and also takes the input from them. The WordPanel class is the model as it repaints graphics and takes the updates done in WordRecord and adds them to a panel to be displayed back on WordApp which is the view.

Additional Features

The program also contains checks to make sure that the start and end buttons can’t be pressed if the game is already started or ended. This is to ensure no 2nd instances of the program could be running in the back. The program also features a help button that display the information about the game and how to play.