CSC2002 Assignment 4
Concurrency - Falling Words
RHNSUV001 – Suvana Rohanlal

Introduction

In this assignment the concepts of concurrent data access, synchronization and multithreading were tested in the form of a typing game.

The program uses the Model-View-Controller (MVC) Pattern in which specified words fall from the top of the screen at varying speeds. The user must type the words correctly before they reach the red "Missed" zone. If a player types a word correctly, then a "Caught" counter is incremented and the players score is updated by the length of the word.

Current classes

The classes modified was the score class, wordApp, WordRecord and WordPanel.
In the score class, the variables were associated with the atomic class.

In the wordApp class a new button called "quit" was added and all buttons were set to perform an event.

In wordPanel the run method was completed.

New classes

There are two new classes added namely, Threads and HandleWords. The classes are associated with the controller part of the MVC pattern.

Threads – This is used to handle the words and dropping them. It is also to ensure that the game continues. It performs the basic function of running the game.

HandleWords – this is in charge of updating the scores and states of the game.

Concurrency Features

- Atomic Integer Class: I used this class in order to allow for atomic actions and to reduce the number of synchronized methods used.
- Volatile Variable Modifier: I used this modifier for Booleans that needed to be accessed by more than one Thread object
- Synchronized Variable Modifier: This modifier was used on variables to poll or update them.

Code Methodology
Atomic calls cannot be interrupted and can be done by one thread at a time. In order to ensure thread safety, I used atomic data classes and synchronized methods where necessary. This set up allowed for data manipulation without risking race conditions. The score class was set up to use the atomic integer class. The variables needed to store the information of the game that is related to the score was set up using this class.

The synchronized method was also used. This keyword prevents reordering of code and involves locking and unlocking. This also prevented deadlock which is a situation where two or more threads are blocked forever because they are waiting for each other.

Validation
The only method used to test the program was to play the game numerous times and look for issues with each play. There were initially incrementing and score issues that where then solved. More than one person played the game in order to receive more feedback.

MVC Pattern:
The model-view-controller pattern is used to separate the application's concerns.

Model – This represents an object carrying data that gets updated. In terms of my program, the model classes are the Score class, Dictionary class and word record.
View – This represents the GUI of the program. The Panel and WordApp class complied with this part of the MVC pattern.
Controller – This updates any data that is in the model and view. The two controller classes used was the HandleWords class and the Thread class.

Additional features
The only additional feature I added was a high score updater. One the player has won the game, their score is checked against other scores they had received. If they have a new high score, it will be displayed on a JOptionPane.

A message will also be displayed to inform the user when the game is over and if they have won or lost and the score they received. The game is won when you "catch" ten words and is lost when you "miss" ten words.

GIT log: