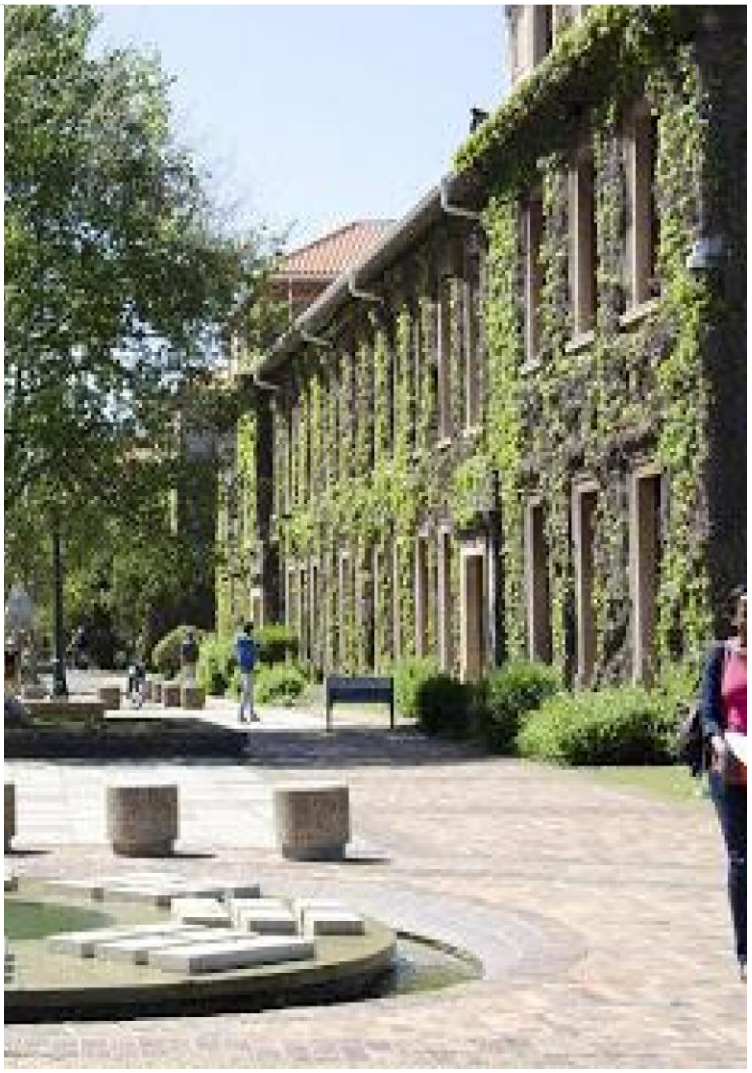
CSC2002S: Assignment 2 report 2021



Sep 06

Authored by: Kevin Chiloane CHLKEV001

The focus of was to design multithreaded java program while ensuring both thread safety and sufficient concurrency for it to function.
Provided a skeleton package consisting of classes to extend in order to support the implementation of multithreaded coding game.
This report explains the edit and classes introduced to enable the Model View Controller and concurrency functions and their necessity for thread Safety.

Classes:

1. **WordDictionary:**

   From the skeleton package, no edits were made on the class

2. **WordPanel:**

   From the package skeleton, Run method was edited to enable the animation.
   Repaint method added to paint the panel whilst Playing.

3. **WordRecord:**

   From the package skeleton, auto generated equals method to check if the words Inserted by the player are the same as the one displayed

4. **Score.java:**

   from the skeleton package, the variable are changed to "AtomicInteger" to avoid synchronizing the getters and setters methods and also three methods namely getTotal, caughtWord and resetWord were changed to synchronized

6. **WordThread:**

Added to the existing skeleton package, handles falling Words and maintain the state of the program.

7.WordControl:

added to the existing skeleton package, called control as it controls the Model View Controller pattern because the class is responsible for the action performed given the the state of the program when running.

5.WordApp:

From the package skeleton, under comment "snip" the pause and quit functions were created and enabled all button use, and panel display of results.

Concurrency:

AtomicInteger variables:

To protect the variable from being exposed to Different threads at the same time, we used type AtomicInteger from the concurrent package instead of int type.

Synchronized classes:

The use of synchronized method protected Variables and info from being accessed by multiple Thread at the same time and avoid corruption of data

Volatile Variable:

like synchronized method volatile allows different threads object to access variable of type volatile

Thread Safety:

Tackling the race conditions by using atomic and synchronized methods to avoid thread from accessing certain information simultaneously, which avoid possible errors that might compromise program functionality

Thread synchronization when necessary:

The falling word were contained in an array therefore the use of thread synchronization was necessary to allow multiple thread to access the array.

Liveness:

The use of synchronized methods to ensure good running time of the program and avoiding delays thus ensuring the liveness of the program.

No deadlock:

The synchronized method avoids the deadlock Because all threads are running and no threads are waiting on other threads.

Validation:

I played the game multiple times testing all the Buttons and for a different perspective I invited 5 of my friends to try catch every falling words, they got bored and ended up adding more word to the dictionary but no bugs reported.

Conforming to the MVC:

The architecture separates the internal representation of the information from the display of the information to the user.

the model comprises the classes WordDictionary, the array of WordRecords, and the Score.

the MVC controls data flow into the model object and updates the panel when data changes and the controller is made of the classes WordThread and WordControl.

Additional Features:

a counter to increment all incorrect attempts.

A pause button to freeze the game and continue anytime with no data loss.

Conclusion:

the use of concurrency and thread safety proves the effectiveness for a program to run smoothly and to prevent a race conditions.