CSC Assigntment 4: Concurrency

This assignment is based on creating or designing a multithreaded Java program, that ensures both thread safety and sufficient concurrency for it to function properly and ignore any possible errors. In this assignment we calculate the average sunlight in the cells covered by a certain tree, we are provided with co-ordinates of the tree trunk and must do calculations to go to the initial point where the tree starts the corner of the tree, this makes it easy to calculate the average of the tree. We then are required to reduce the sunlight in the cells to 10% of their original value. Thus, trees in the later layers will receive less sunlight. This simulates the filtering of sunlight through the canopy in a forest. A tree then grows in proportion to the average sunlight divided by a factor of 1000. This assignment builds from the previous one and we still use parallelism, we add safety rules to ignore deadlocks, bad interleaving's and data races.

Methods

SunData.java

This class is one of the classes provided for successfully completing the assignment. This class makes use of *Land.java* and *Trees.java*. In this class we have two functions which are explained in detail below.

Functions of Class:

- ***readData(String filename) :***

This function accepts a String parameter, which is the name of the file that has the data to be read. In this function we firstly read the file provided in the parameter and initialise our 2D array by calling the constructor in Land.java, the constructor accepts two integer parameters which are the x and y size of the terrain, by doing this we initialise the size of the array. We also populate the 2d array in this class by reading the values in the file, we populate by calling methods in *Land.java* and *Tree.java*.

- ***writeData(String filename):***

This function receives a String parameter which is the file name to be written into, it is responsible for writing the data into being processed into files.

Land.java

This class has to do with specifying the details in the terrain we are working with. In the constructor method we accept two integer parameters, these integers initialise the size of the terrain, the x and y (horizontal size and vertical size).

Functions:

- ***getDimX()***

This function returns the length of the x-value (the size of the horizontal), it is a getter method, it returns a value..

- ***getDimY()***

This function returns the length of the y-value (the size of the vertical), it is a getter method, it returns a value.

- ***resetShade()***

This method resets the shaded landscape (terrain) to the same as the initial sun exposed landscape, this method is called after each growth pass of the simulator.

- ***Synchronized float getFull(int x, int y):***

This function returns the sun exposure of the initial unshaded terrain or landscape at the point or location (x,y). The integer parameters x and y are used as the position in this case.

CSC Assigntment 4: Concurrency

- ***synchronized void setFull(int x, int y, float val)***

This function sets the exposure of the initial unshaded landscape at the given position of x and y provided and the value that is inserted in the position is the **val** which is inserted as the parameter. The word synchronized as to why it is used it will be explained later in the report.

- ***synchronized float getShade(int x, int y)***

This function returns the current sun exposure of the shaded landscape at the position x and y

- **synchronized void setShade(int x, int y, float value)**

This function set the exposure of the shaded landscape at position x and y to "value", these co-ordinates and the value is received from the parameters.

- **synchronized void shadow (Tree tree)**

This function receives as a parameter an object of type Tree, this parameter is made to access methods in the class ***Tree.java***. This function basically is meant to reduce the sun exposure of the shaded landscape to 10% of the original within the extent of tree.

Tree.java

This class is also used by the *SunData.java* class as well *Land.java*. It is used to define a canopy which covers a square area of the landscape.

Functions:

- **synchronized float sunexposure(Land land)**

This function receives the average sunlight for the cells covered by the tree. This function gets as a parameter an object of type Land, this is because this function access methods in the Land class, which are getDimY, getDimX, getShade and other methods or functions as well. This method firstly calculates the sum covered by the cells of the tree and then divide by number of cells to get the average of the tree, which is then returned.

- **synchronized boolean inrange(float minr, float maxr)**

This method receives as parameter min and max, which is used to check whether a certain value is between the minimum and maximum supplied in the parameters. This method returns true or false. If true something will happen else nothing will happen. Basically, it checks if the tree extent is within the provided range.

- **synchronized void sungrow(Land land)**

This method receives an object of type land as a parameter to make use of the methods or functions in the Land class. This method grows a tree according to its sun exposure. It updates the sun exposure as it is called by the threads.

ForestPanel.java

This class is responsible for the frame that displays the trees. It also changes the colour of the trees as they grow. This class makes use of repaint () which changes the look of the component, but not the size (it changes the colour, animating and more). *ForestPanel.java* is used in the *TreeGrow.java* class.

thread.java

This class is where the processing of this assignment happens. A separate thread as required in the assignment is created, this class runs the processing of updating the extend of the trees and this is automatically reflected in the graphical user interface. This class *extends* ***Thread,*** and then it overrides the function called **run**. Functions of this class are explained in detail below.

CSC Assigntment 4: Concurrency

Functions:

- **thread(Sundata sundata):**

This is the contractor of this class, it accepts a parameter of type Sundata, which will be inserted when this class is called by a different class.

- **Pausing_the_threads():**

This forces the infinite while loop to stop and pause from growing trees, this method is called when the pause button is pressed.

- **run():**

When extending Thread, we override the method called **run**. In this method we have an infinite while loop that is made grow the trees repeatedly until this process of growing trees is paused. In this function **run**, before growing a tree we use another method from another class that checks if the extent of the tree is in a certain range.
To make use of *thread.java*, we create the object of this class first in ThreeGrow.java and the call start. Example shown: this is how you would create an object,
thread t=new thread(); → This creates new object of the thread.java class
t.start(); → This makes the thread to start.

Required Concurrency Features

In this assignment I used the following concurrency features, volatile variable modifier, synchronization and the AtomicInteger class.

- I made use of *synchronization* in several functions or methods which have a possibility to be accessed by multiple threads at once, I locked these methods to prevent bad interleaving. When one thread is making use of a locked method or function, all other threads wanting to access the method block, until the thread that was working with the method first finishes, only one thread uses the method at a time.
- I made use of *AtomicInteger* class to be able to make use of atomic actions and to reduce the number of methods being synchronized in the program.
- I made use of *volatile* for integer (the counter to increase the number of years), this counter is accessed by more than one thread object.

Further Explanations on how the following has been ensured or implemented:

1. I ensured thread safety in this assignment by making use of atomic data classes like AtomicIntegers and I also made use of synchronization in necessary methods, this reduces the race conditions, it allows data to be manipulated without the risk of race conditions.
2. I wrote code to synchronize in the methods in Land.java and Tree.java classes. Data in these classes is accessed by different threads which increase the chances of data races or bad interleaving's, for example sun grow is accessed by multiple threads which also access other methods, and this might cause bad interleaving's if it is not synchronized, methods accessed by multiple threads are synchronized/ locked to allow only one thread to access the method at a time.
3. I ensured liveness by making use of synchronization functions to prevent any deadlocks and I ensured that threads which access data that is shared are not greedy or making use of a function by themselves all the time and not giving other threads a chance to make use of the function, this will prevent starvation as well, meaning that the threads will not be unable to make use or access shared resources, all threads will be able to make progress and not be deprived to make use of the functions.

CSC Assigntment 4: Concurrency

4. As explained above I ensured that there are no deadlocks by making use of synchronization or synchronized methods.

Explanation of how the system was validated and checked for errors:

In this assignment accessor and setter methods are being synchronized or the variables used in them are made to be atomic instead of making use of the normal variables.
Some integers and boolean variables are made volatile, so that the updated values which are made volatile are available all the time when checking on other methods or the while loops being used, this ensures the threads to pause or stop when the user wants them to do so.
Updating by time has also been used, meaning that the threads are made to sleep for a certain time, this has an effect or advantage of preventing or decreasing the chances of starvation by updating the data to where it must be a time. Even when it sometimes takes too much time to update the animation will tend to appear in a consistent and smooth manner as to when threads are not made to sleep for a certain time. This does have its disadvantage for example if it takes too much time then there will be jumps instead of the preferred slow smooth performance, but this does not happen all the time, it is unlikely to occur.

Model-View-Controller

In this assignment as expected the changes in data are made by the controller class, the view will update the animation being showed which are in this case the growth of trees, from the model.

Model:

The model in this assignment is the *Land.java* and *Tree.java* classes. These two classes have the data that is being used in this assignment, for example the sun exposure data of the trees, growing factors and more.

View:

The view is the graphical user interface that appears, which in this case will be the *ForestPanel.java* class also the *TreeGrow.java* class add buttons to the view. ForestPanel.java draws the trees in a form of a square and changes colours of the trees, without making any updated on them, or the data.

Controller:

Thread.java is the controller class in this assignment. This class manipulates the data, it is responsible for updating the extents of the trees, so that how they grow appears in the graphical interface.
TreeGrow.java is not in any of these categories, the TreeGrow.java class is responsible for setting up the interface and adding buttons and their actions. When a certain button is pressed, there will be a certain reaction as well, for example when paused is pressed, the growing of trees will pause until play is pressed again.

Additional Features or Extensions:

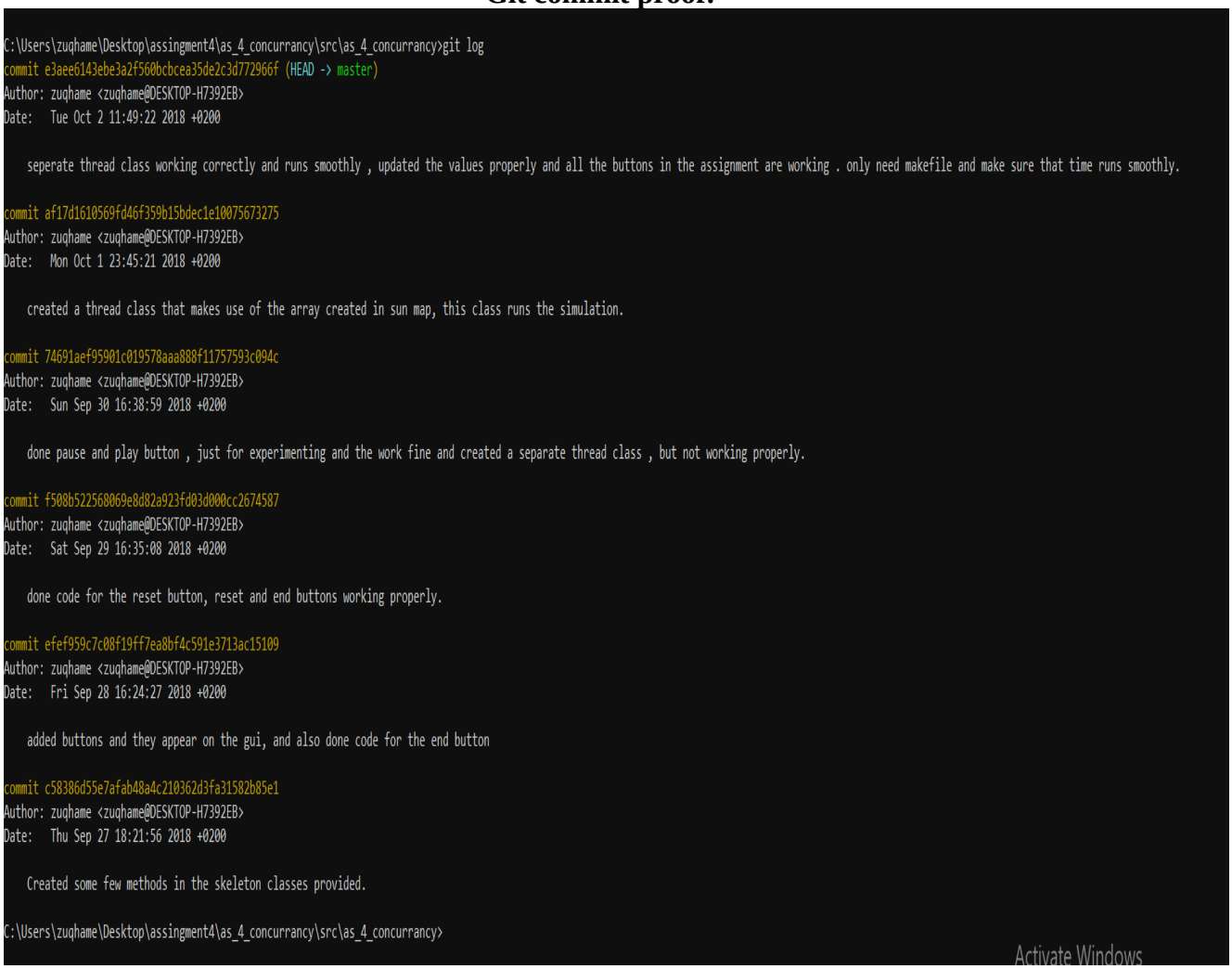
1. In this assignment I generated javadocs
2. I made the buttons easily interactive with the user, meaning that, A user cannot press pause while the start button has not been pressed, so initially I disable the pause button from preventing the user from making a mistake of pressing pause before play, because this causes an error if it done.
3. Confirmation pops up when user exits, the user is asked whether they really want to end the game, in cases where they press end button by mistake.

CSC Assigntment 4: Concurrency

Conclusion

The trees grow smoothly, and years are also shown, all the buttons are working, but the buttons take few seconds to respond or do the action. But In a faster machine they are better. The trees start appearing at an average of 60 years, not that they are not growing but they start very small and take time to appear.

Git commit proof:



```
C:\Users\zuqhame\Desktop\assingment4\as_4_concurrancy\src\as_4_concurrancy>git log
commit e3aee6143ebe3a2f560bcbcea35de2c3d772966f (HEAD -> master)
Author: zuqhame <zuqhame@DESKTOP-H7392EB>
Date:   Tue Oct 2 11:49:22 2018 +0200

    seperate thread class working correctly and runs smoothly , updated the values properly and all the buttons in the assignment are working . only need makefile and make sure that time runs smoothly.

commit af17d1610569fd46f359b15bdec1e10075673275
Author: zuqhame <zuqhame@DESKTOP-H7392EB>
Date:   Mon Oct 1 23:45:21 2018 +0200

    created a thread class that makes use of the array created in sun map, this class runs the simulation.

commit 74691aef95901c019578aaa888f11757593c094c
Author: zuqhame <zuqhame@DESKTOP-H7392EB>
Date:   Sun Sep 30 16:38:59 2018 +0200

    done pause and play button , just for experimenting and the work fine and created a separate thread class , but not working properly.

commit f508b522568069e8d82a923fd03d000cc2674587
Author: zuqhame <zuqhame@DESKTOP-H7392EB>
Date:   Sat Sep 29 16:35:08 2018 +0200

    done code for the reset button, reset and end buttons working properly.

commit efef959c7c08f19ff7ea8bf4c591e3713ac15109
Author: zuqhame <zuqhame@DESKTOP-H7392EB>
Date:   Fri Sep 28 16:24:27 2018 +0200

    added buttons and they appear on the gui, and also done code for the end button

commit c58386d55e7afab48a4c210362d3fa31582b85e1
Author: zuqhame <zuqhame@DESKTOP-H7392EB>
Date:   Thu Sep 27 18:21:56 2018 +0200

    Created some few methods in the skeleton classes provided.

C:\Users\zuqhame\Desktop\assingment4\as_4_concurrancy\src\as_4_concurrancy>
```


The picture is also in the folder of the assignment