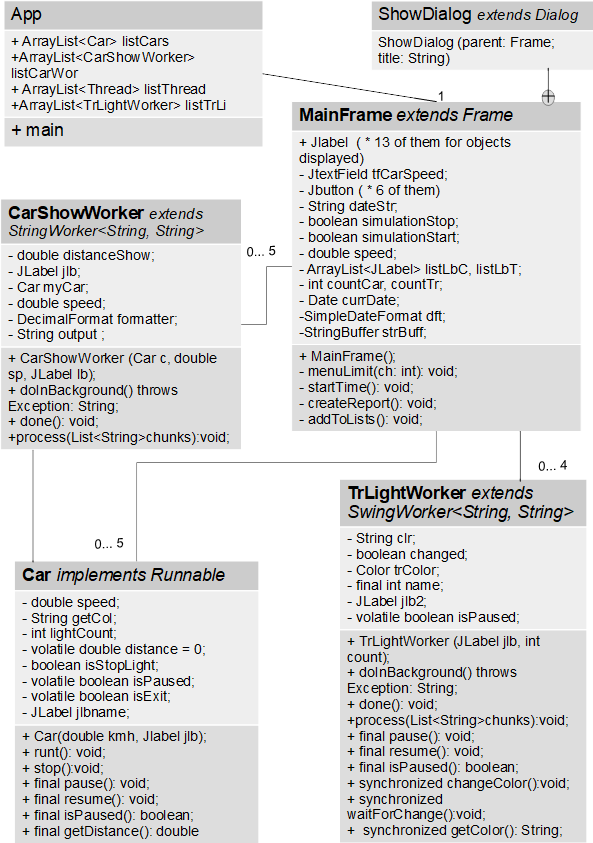
**CMSC 335, Project Final -- December, 4, 2021**

**Karina D’Abbraccio**

**UML Diagram**

****

**Description**

At first, it was working interaction of threads, but with wrong logic of cars moving: the cars need to move little by little, to imitate driving, but updates need to be made only once in a second. For easier testing and catching cars on intersections, I increased time of the yellow and red light.

Min speed is 30 km/h, max speed is 170 km/h; per second it is min 8.3 m/s to max 47.2 m/s. If in run() of a Car thread sleeps for 100 ms, in distance about 5m or less (2.5 heights of a tall man) before light it should be able to stop. Max distance, after which the thread is not running, is 5 km: the Label for the car will say ‘Car Arrived’. Maximum number of cars is 5, max number of traffic lights is 4. They would be added in cells of GridLayout with already added labels. Labels are available to use by new objects that are created at runtime. All new cars and intersections are added to the list of them, to loop through all on Pause/Continue buttons. Methods wait() and notify() were not possible to apply, because the error that the thread is not the owner appeared. So, custom methods to pause() and resume() were added, changing a volatile boolean isPaused to proper value. All labels are added to list also, to retrieve the next available display spot. This is the final UI of a project:

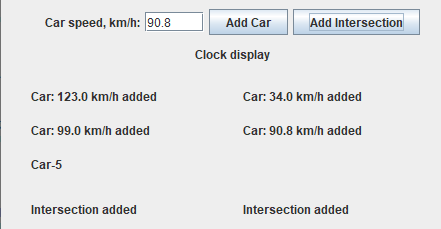
Изображение выглядит как текст

Автоматически созданное описание Изображение выглядит как текст

Автоматически созданное описание

Menu has two menu items with dialog windows, showing limitations (number of cars and intersections, speed), and user manual, with explanations on buttons.

If the cars or intersections are added before “Start” button, they don’t start moving:



1. Button "Start": starts timer, added cars and intersections. The Start time will appear at the bottom of the screen. More cars and intersections can be added after ‘Start’ button.

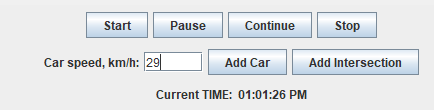
2. Button "Pause": will pause cars and traffic lights only. Does not pause current time.

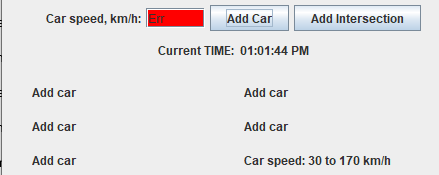
3. Button "Continue": will resume cars and traffic lights: they will continue moving/switching lights.

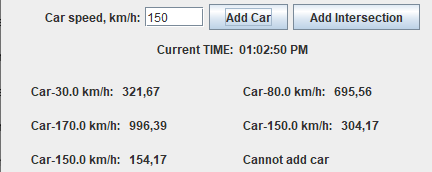
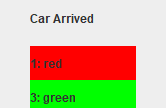
4. Button "Stop": will show the last state of cars and intersections, labels will be reset and stop time appear at the bottom. Intersections are a little slow, they start, pause and stop with 1-2 seconds delay, so the light might change after “Pause”/”Stop” once again.

5. Button "Add car": will get the speed and create the new car, that will have name based on its speed. If the speed is not correct, the Error will be shown as red text field and with the explanation of speed limits. If the limit on number of cars is exceeded, message appears.

Cars added before “Start” move after pressing “Start”. Cars added after “Start” will start immediately.





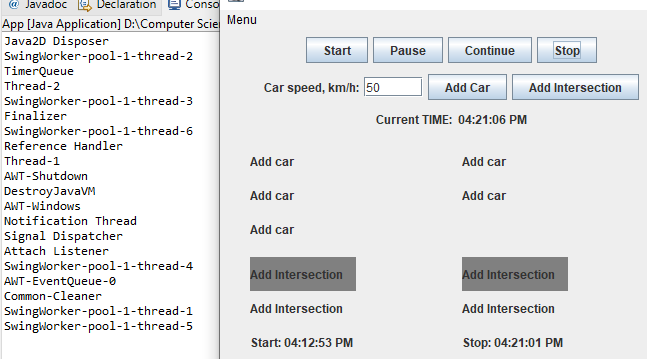
The car route is 5000 m maximum, then it arrives.

6. Button "Add Intersection": will add one. Lights time is: red -7s, yellow -2s, green -5s.

Изображение выглядит как текст

Автоматически созданное описание

At first, my idea was that “Stop” button resets the simulation, stops threads and SwingWorkers. But I did not find the way to assign new tasks to SwingWorkers correctly. The new cars were mowing with the SwigWorker attached to it updating the distance. But at the first intersection they would only ‘see’ the red light of a traffic light and never proceed moving. I’ve added menuItem to print active threads, SwingWorkers were still active after cancellation:



So, the function of a “Stop” button was changed into stop the simulation and finishing the work with the app, **no simulation can be started after pressing “Stop”**, buttons will show:

Изображение выглядит как текст

Автоматически созданное описание

Also, the “Stop” button generates simple report in the default folder:

Изображение выглядит как текст

Автоматически созданное описание

**Testing:**

As shown above on screen captures in Description section, the functions tested:

1. Design and correct actions on each element: menu items, buttons. Timer object.

2. Error handling on car speed out of limits, non-numerical values.

3. Adding Car object and starting CarShowWorker (extends SwingWorker) for that car.

4. The Cars ‘move’ and CarShowWorkers updating the labels for them.

5. Adding intersection as object of TrLightWorker, the change in light as it displayed in labels: there are synchronized methods in it and adding an intersection has 1-2 seconds delay.

6. The Cars stop as they come closer than 5 m when the light is red, continue moving when the light goes to green. Stop on each intersection and proceed.

7. “Car Arrived” appears when the distance is maximum close to 5000 m and doesn’t exceed it.

8. Work of “Pause” and “Continue” buttons for cars, traffic lights: traffic lights are slow, they might do one more change after pressing a pause.

9. Adding cars and intersections after “Start” button.

10. Error handling on adding more than 5 cars or more than 4 intersections.

11. “Stop” button: at first design option revealed bug that I could not fix. After changing the app, all previous tests were repeated and for “Stop”: no other objects can be added and message in a Dialog window appears; the “Start” button doesn’t work: the message appears; report with information on time and objects added, car distances appears in a file.

**Lessons learned:**

The threads that interact with GUI differ from other threads.

Custom methods for pause and stop the thread may be designed.

Differences between volatile, atomic, and synchronized.

The fact that there is a list of bugs in Java language standard libraries and SwingWorker has a lot of them.

Applied different GUI elements and combined them with multithreading, included file writing in the program.

Ambiguity in specifications leads to freedom in implementation. May be implemented very differently by different people. Not being restricted in tools gives a possibility to choose the best working solution, but also a chance of an error to apply the wrong ones.

Looked the description of StringBuilder and StringBuffer classes, I’ve never used before.

Testing of threads is longer, since need to ‘catch’ certain events, like car must be close to the intersection when the light is not green. Need to test static functions, and in scenarios of sequences of actions. For testing, some elements may be added (menu item to print active Threads).