Project 2: CONCURRENCY

Victoria Lee

UMGC: University of Maryland Global Campus

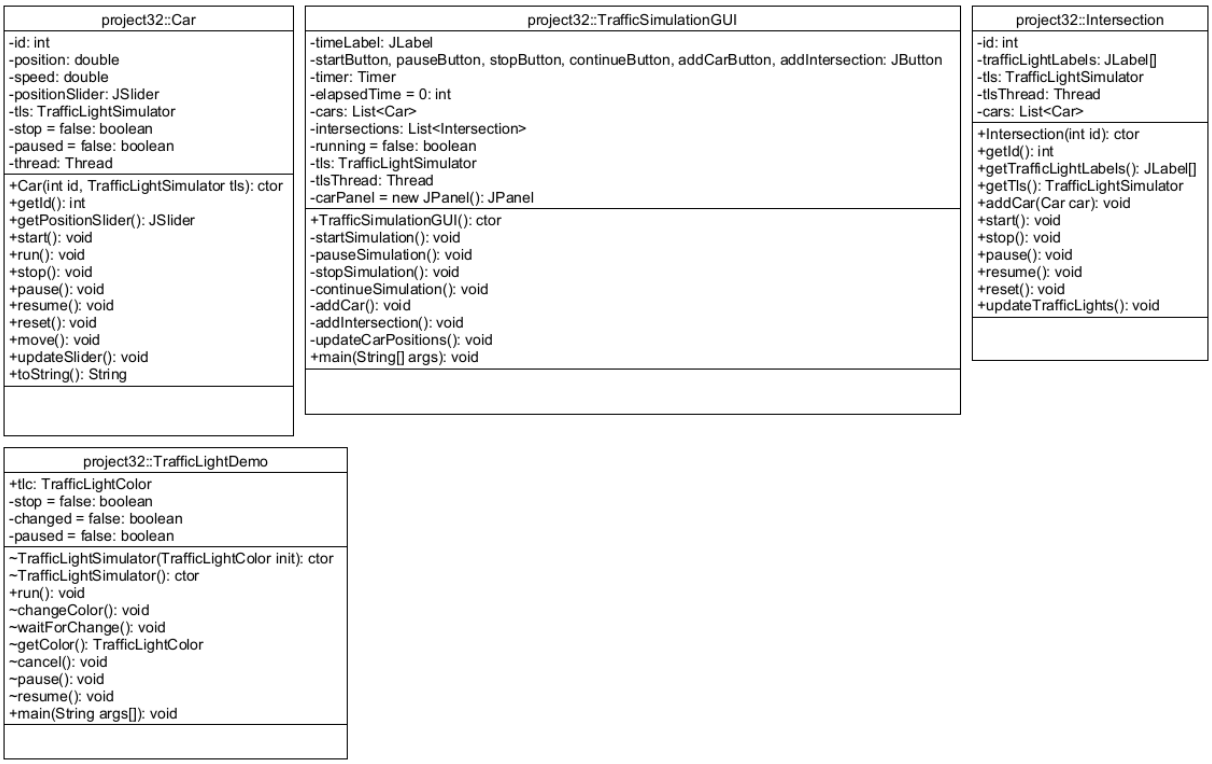
CMSC 335: Object-Oriented and Concurrent Programming

10/3/2024

UML Diagrams:

Note: The Main Method is in TrafficSimulationGUI Class. This allows it to run the program smoothly. The package is project32. The UML diagrams below include the package.

UML Class Diagrams and Package:

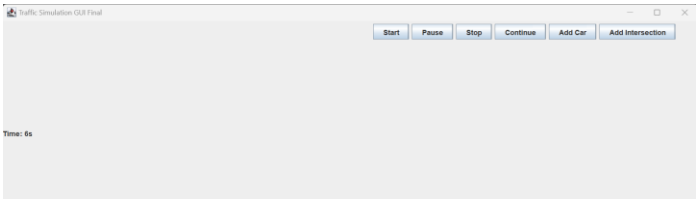


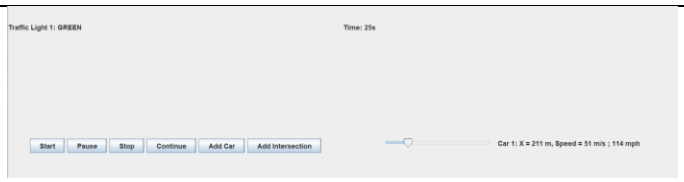
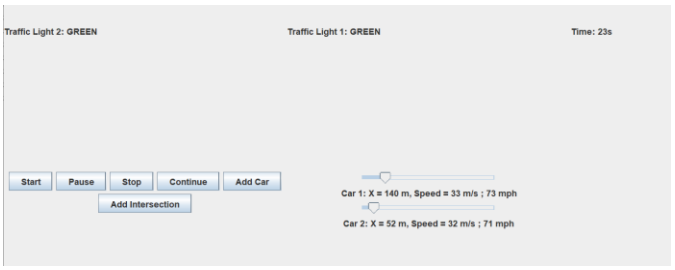
Developer's Guide, Test Cases, and Lessons Learned:

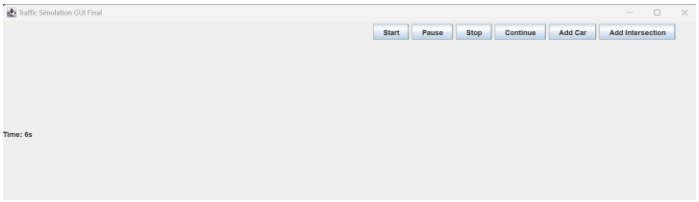
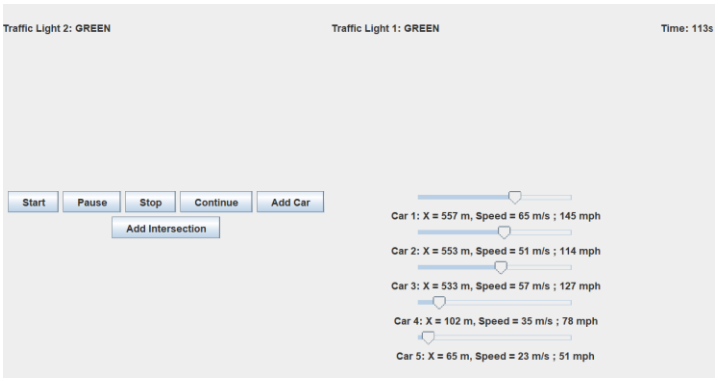
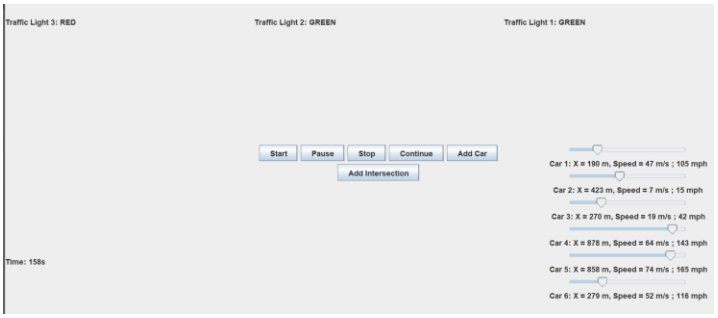
You can import the files using the new project and import all of the classes. Make sure to have a package called project32 in the "src" file. An alternative way is to create the project and import all the files including the project32 package I already included from the ".ZIP" file. You can compile the file and execute the program by going to the TrafficSimulationGUI class, right-clicking on the class, and selecting the run option. In other words, to run the program for the GUI select TrafficSimulationGUI class and run it. Make sure it is allowing the JavaSwing to run. It should pop up the Traffic Simulator in the console and then you can choose the options below.

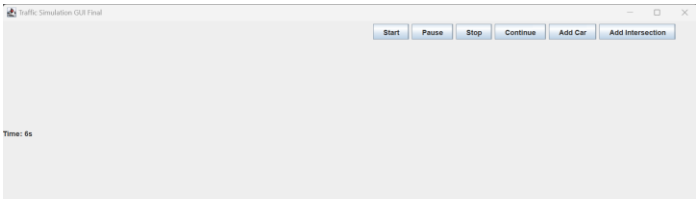
Note: Follow all directions based on the Test Cases.

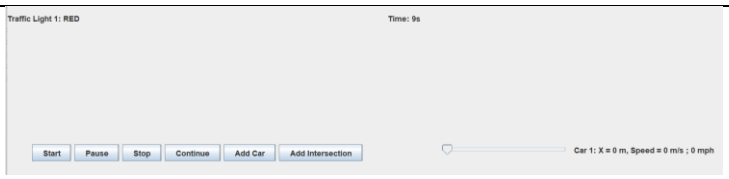
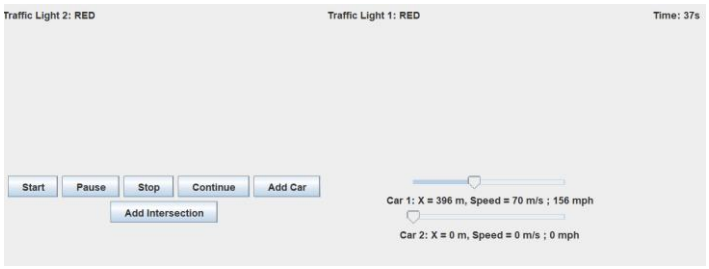
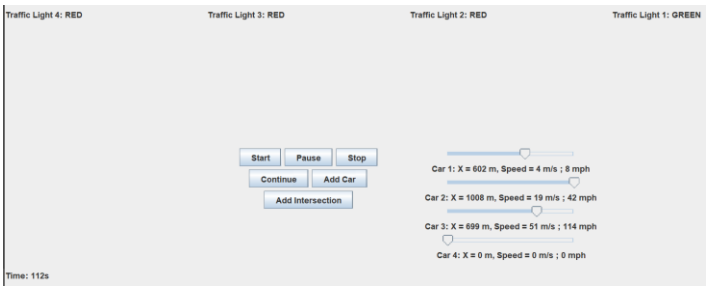
Table 1 (below): *Developer's guide describing compiling and executing the program. Documentation includes Lessons learned at the end.*

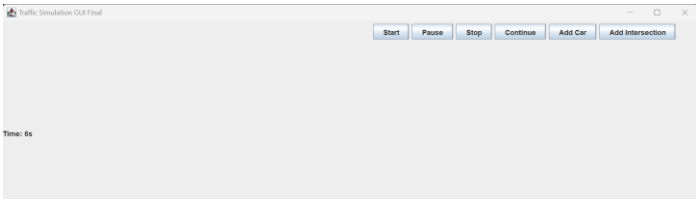
Test #	Description	Screenshot	PASS / FAIL Flag
1.	This is testing to run three separate intersections with three traffic lights and three cars in each intersection. In other words, intersection 1 has traffic light 1 with car 1. Intersection 2 has traffic light 2 with car 2. Intersection 3 has traffic light 3 with car 3. It should also test looping. It should loop and set position to 0 once 1000 m or great has been reached. It should also reset the speed to 0 once 160 mph is reached as most average cars have a max speed of 160 mph. (230 mph	  	PASS

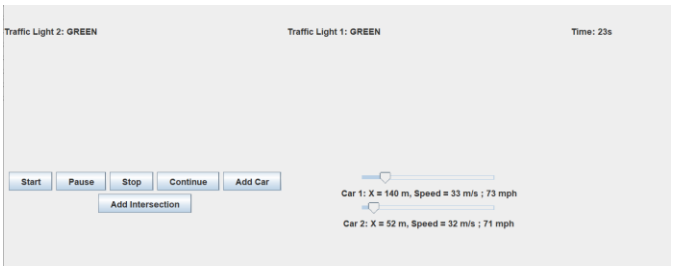
	or greater does not exist in regular cars only sports cars.) To set up this format: Run the program Press the Start Press the Add Intersection Press the Add Car. This now shows Traffic Light 1 with Car 1. Press the Add Intersection Press the Add Car. This now shows Traffic Light 2 with Car 2. Press the Add Intersection Press the Add Car. This now shows Traffic Light 3 with Car 3. It should sync the car with the traffic light where Green,	  	

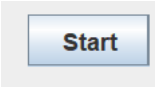
	Yellow is move, and Red is stop for the car.		
2.	This is testing if you can add more cars in each of the three intersections. For instance, Intersection 1 has three cars, Intersection 2 has 2 cars, Intersection 3 has 1 car. To set up this format: Run the program Press the Start Press the Add Intersection Press the Add Car. Press the Add Car. Press the Add Car. This now shows Traffic Light 1 with Cars 1, 2, 3. Press the Add Intersection	    	PASS

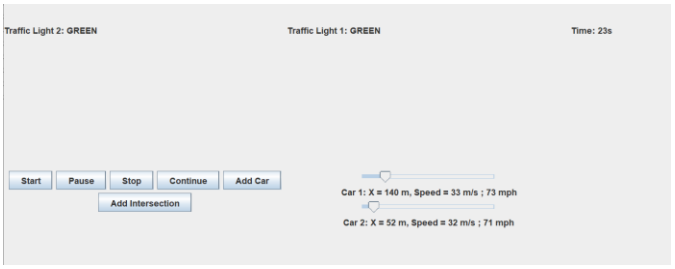
	Press the Add Car. Press the Add Car. This now shows Traffic Light 1 with Cars 4, 5. Press the Add Intersection Press the Add Car. This now shows Traffic Light 1 with Car 6. It should sync the car with the traffic light where Green, Yellow is move, and Red is stop for the car.		
3.	This is to test adding more than three intersections with 1 car per intersection. For instance, Intersection 1 has Car 1, Intersection 2 has Car 2, Intersection 3 has Car 3, Intersection 4 has Car 4.	 	PASS

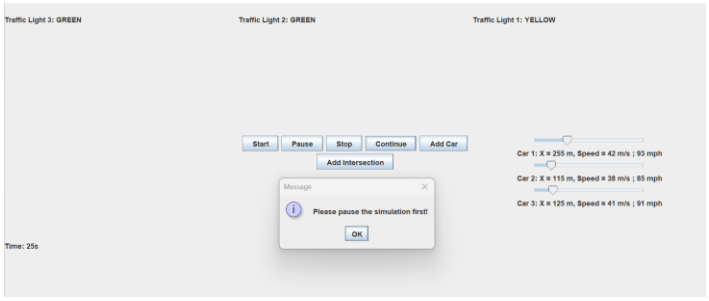
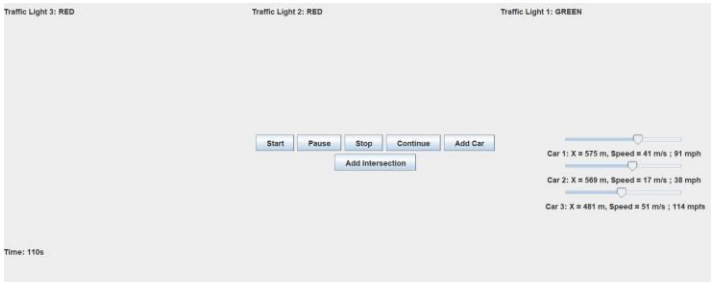
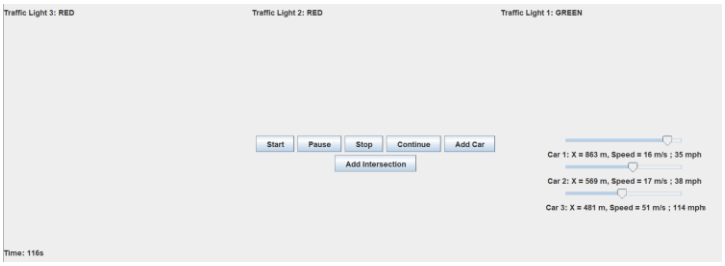
	To set up this format: Run the program Press the Start Press the Add Intersection Press the Add Car. This now shows Traffic Light 1 with Car 1. Press the Add Intersection Press the Add Car. This now shows Traffic Light 2 with Car 2. Press the Add Intersection Press the Add Car. This now shows Traffic Light 3 with Car 3. Press the Add Intersection Press the Add Car. This now shows Traffic Light 4 with Car 4.	   	

	It should sync the car with the traffic light where Green, Yellow is move, and Red is stop for the car.		
4.	This is to test the pause, continue, and stop options using one car in each intersection. It should also work in multiple intersections as well. For this case, it will use 3 intersections. Intersection 1 with car 1 and intersection 2 with car 2. And Intersection 3 with car 3. To set up this format: Run the program Press the Start Press the Add Intersection Press the Add Car.	  	PASS

	This now shows Traffic Light 1 with Car 1. Press the Add Intersection Press the Add Car. This now shows Traffic Light 2 with Car 2. Press the Add Intersection Press the Add Car. This now shows Traffic Light 3 with Car 3. It should sync the car with the traffic light where Green and Yellow is move, and Red is stop for the car. Press the pause and wait for a while. Press the continue. This should stop all cars even when it is green or yellow and stop the timer and threads.	   Now press the pause  Now Press the continue  Now press the stop Traffic Simulation GUI Final	

	The continue button will continue the simulation or resume it when the continue is pressed. To exit the program and stop all threads, Press the Stop. It should stop all threads safely and exit the system to restart the simulation. To use the simulation again, press the run and then start as before.	It stops all threads and exits the application safely. A Thread cannot be rest after it is stopped. To rest it I made it exit the application so that once run again, it will auto reset the thread.  run the program again.  start the simulation again.	
5.	Testing for continue and pause buttons and it should be pressed in the correct order. The order is pause first before continue, else pop up a message to press the pause first. This should prevent any thread		PASS

	issues if continue is pressed when it is not supposed to. This allows continuous pause and continue due to the flag that has been set up. It would not allow pause, continue and continue. It only allows intervals of pause, continue, pause, continue, etc. No continue, continue. For this case, it will use 3 intersections. Intersection 1 with car 1 and intersection 2 with car 2. And Intersection 3 with car 3. To set up this format: Run the program Press the Start Press the Add Intersection Press the Add Car.	    	

	This now shows Traffic Light 1 with Car 1. Press the Add Intersection Press the Add Car. This now shows Traffic Light 2 with Car 2. Press the Add Intersection Press the Add Car. This now shows Traffic Light 3 with Car 3. It should sync the car with the traffic light where Green and Yellow is move, and Red is stop for the car. Press continue before pause. It should pop up a message. Press the pause and wait for a while. Press the continue. Press continue again and it	Press continue before pause button  Press Pause  Press Continue  Repeat. The flags should allow repeated pauses and continue. It would not allow double continue until pause is pressed.	

	should pop up a message. Repeat if necessary.		

Lessons Learned (brief paragraphs):

To achieve my project goals, I learned how to create a project, classes, immutable classes, try/catch, throw exceptions, JavaSwing, encapsulation, inheritance, information hiding, polymorphism, threads, multithreading, methods, and functions for multithreading OOP for the Traffic Simulator GUI program. I learned that the Traffic Simulator GUI program allowed me to comprehend the significant principles of object-oriented programming and software development. A class defines all the attributes an object can have and methods that define the object's functionality. A subclass inherits the properties and behaviors of another class. Immutable classes in Java mean that once an object is created, we cannot change its content. Threading in Java Swing refers to creating separate threads of execution within a Swing application. This allows for non-blocking operations, such as long-running tasks, to be performed without freezing the GUI. Multithreading involves using multiple threads simultaneously to perform different tasks, potentially improving performance and responsiveness. However, managing thread synchronization and avoiding race conditions is crucial to ensure correct GUI updates and prevent errors. The Runnable interface in Java Swing is used to define tasks that can be executed by a thread. It requires implementing the run() method, which contains the code to be executed by the thread. To create and start a thread, you can create a Runnable object and pass it to a Thread object, then call the start() method on the Thread object. This will start the thread, which will execute the code in the run() method.

Multithreading can be achieved by creating multiple Runnable objects and passing them to separate Thread objects, allowing for concurrent execution of different tasks. The Car class creates the car threads object. The Intersection class creates the intersection object to combine the traffic light simulator into it. The TrafficLightDemo class creates the traffic light simulator for the traffic lights of the intersection to sync with the cars. And so, the main lesson learned from the TrafficSimulationGUI class is that it is responsible for creating the Traffic Simulator GUI program and menu. This menu option and the follow-up buttons for the start, pause, stop continue, add cars, and add intersection. I had to utilize the Timer and buttons to produce the cars and traffic light threads. The Labeled class is the base class for Label, Button, etc. The ButtonBase class defines the onAction property for specifying a handler for action events. When a start button is pressed, an action has occurred, etc. I also made sure to include an alert so that the user cannot press continue until pause is pressed. Moreover, these immutable classes are TrafficLightDemo, Car, and Intersection. The main lesson learned from the TrafficSimulationGUI class is that it is responsible for creating the user interface GUI. The main goals in this class's design include its GUI constructor, which initializes the object, and its methods so that the objects and threads are projected through a display slider based on user input of the car object. These lessons helped me understand good modular design for car and traffic light multithreading objects for this program. In real life, users can utilize this application for a traffic simulator driveway, construction, or creating multiplayer video games for car racing. TrafficSimulationGUI class is about comprehending Java GUI for JavaSwing, the classes, hierarchy, inheritance, encapsulation, information hiding, polymorphism, is-a, has-a, relationships, and subclasses. Overall, I learned to apply it to TrafficSimulationGUI class with the lessons about JavaSwing, threads, multithreading, try/catch, classes, subclasses, packages,

importing libraries, constructors, object-oriented programming, encapsulation, inheritance, information hiding, and polymorphism.

My design approach was to create all the required classes before implementing the TrafficSimulationGUI class. I started with a Bottom-Up Design when building the code, but then debugged the code through a Top-Down Design. I followed the instructions on what is asked for TrafficSimulationGUI and the tips. I utilized the lessons to apply them to TrafficSimulationGUI. Once it was finished, I went back into the TrafficSimulationGUI class to create the Car, Intersection, and TrafficLightDemo classes according to the rubric, and then the user input the buttons would be passed and the results back through the TrafficSimulationGUI class. TrafficSimulationGUI class creates two types of multi-threads: car and Traffic lights (intersection), which then have more threads you can make through the button selection menu program. This allows the user to input the number of cars per intersection (traffic light) to project the TrafficSimulationGUI. To debug TrafficSimulationGUI, I looked at the lessons, my old codes, and online concepts related to this chapter. I then modified the classes. Then, I checked back to see if the output was correct through the TrafficSimulationGUI class.

References

GeeksforGeeks. (2024, July 19). *Deadlock, Starvation, and Livelock*. GeeksforGeeks. https://www.geeksforgeeks.org/deadlock-starvation-and-livelock/

GeeksforGeeks. (n.d.). *Dining Philosopher Problem Using Semaphores*. GeeksforGeeks. https://www.geeksforgeeks.org/dining-philosopher-problem-using-semaphores/

GeeksforGeeks. (n.d.). *Lock Framework vs Thread Synchronization in Java*. GeeksforGeeks. https://www.geeksforgeeks.org/lock-framework-vs-thread-synchronization-in-java/

GeeksforGeeks. (2024, April 24). *Deadlock in Java Multithreading*. GeeksforGeeks. https://www.geeksforgeeks.org/deadlock-in-java-multithreading/

Goetz, B. (2006). *Java concurrency in practice*. Pearson Education. https://www.academia.edu/download/53777814/Java_Concurrency_In_Practice.pdf

Javatpoint. (n.d.). *Why Thread.stop(), Thread.suspend(), and Thread.resume() Methods are Deprecated After JDK 1.1 Version*. Javatpoint. https://www.javatpoint.com/why-thread-stop-thread-suspend-and-thread-resume-methods-are-deprecated-after-jdk-1-1-version

Laboratories, M. (n.d.). *Lock-free multithreading with atomic operations*. Internal Pointers. https://www.internalpointers.com/post/lock-free-multithreading-atomic-operations

Musib, S. (2023, May 12). *Deadlock, Livelock, Starvation*. Baeldung. https://www.baeldung.com/cs/deadlock-livelock-starvation

Oracle. (n.d.). *Thread*. Oracle. https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html

Oracle. (n.d.). *Starvation and Livelock*.Oracle. https://docs.oracle.com/javase/tutorial/essential/concurrency/starvelive.html#:~:text=Starvation%20describes%20a%20situation%20where,periods%20by%20%22greedy%22%20threads.

Oracle. (n.d.). *Deprecated Threads Methods*. Oracle. https://docs.oracle.com/cd/E19455-01/806-3461/6jck06gr5/index.html

Oracle. (n.d.). *The Java™ tutorials: Object-oriented programming concepts*. Oracle. https://docs.oracle.com/javase/tutorial/java/javaOO/index.html

Oracle. (n.d.). *Java Lesson: Subclassing*. Oracle. https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

Oracle. (n.d.). *JavaFX API Documentation*. Oracle. https://docs.oracle.com/javafx/2/api/