MVC Timer Stopwatch FXML S21

<u>Description:</u>
Create a Java 8 SE application in NetBeans using JavaFX FXML, an associated Controller, and an associated Model for each the analog and digital stopwatch. These elements combined will implement a stopwatch with an analog and a digital display. The stopwatch has two buttons. The text on the two buttons should be displayed and changed dynamically. The record board will be used to record the most recent lap time and average lap time. The code should be separated into the proper Model-View-Controller (MVC) architecture that was discussed in class.

<u>Purpose:</u>
This challenge develops skills in creating and manipulating JavaFX interfaces, using FXML and Scene Builder, using a Controller associated with an FXML interface, generating and handling events, and organizing code in the proper Model-View-Controller Architecture.

<u>Requirements:</u>
- Project Name: <Pawprint>MVCTimerStopwatchFXMLS21
- For the Project Name follow the same naming scheme used in the first challenge. The Project Name is to be comprised of your pawprint with the first letter capitalized followed by MVCTimerStopwatchFXMLS21. For example, if the pawprint is **abcxyz9** the project is to be named **Abcxyz9MVCTimerStopwatchFXMLS21.**
- This challenge allows you to apply your creativity while meeting a set of requirements. Not every aspect of the application is described in the requirements. The requirements establish what functionality must exist, but it is up to you to apply your creativity in implementing the functionality and the user interface to support that functionality.
- You can use the Stopwatch design you created for the previous Stopwatch challenge or, if you prefer, you can implement a new design. Just make sure you use FXML, Scene Builder, a Controller that is associated with the FXML, and two separated Models that are associated with the Controller.
- There will be one Model for the analog stopwatch and one Model for the digital stopwatch. The Model-View-Controller (MVC) Architecture should be applied and you should use the correct MVC communication between layers.
- Analog and Digital Stopwatches should be in separate classes, two Models, and will <u>not</u> communicate to the View directly
- The FXML Controller Class will be separate from the Analog and Digital Models and will communicate between the Models and the View
- If done correctly, you will, minimally, have a main method class that runs the FXML application, a model class that contains all domain objects for the analog stopwatch, another model class that contains all the domain objects for the digital stopwatch, a view which will be the FXML file consisting of the layout elements, and a controller class that communicates to the view and model class (4 separate classes and the FXML file, 5 in total, minimally)
- Must have a record board that will be used to record the lap time between the differences of the most recent two button clicks (i.e. Difference of current time with last recorded time) and the average lap time.

- Have a line chart that will be used to plot all lap times and an area chart that will be used to plot all average lap times.
- Have a timer text that displays the time remaining.
- Platform: Java 8 SE, JavaFX, FXML with Controller and 2 Model Classes
- Project IDE: NetBeans

Create a stopwatch application that contains an analog display, a digital display, a timer display, a record board that contains the most recent lap time and the average lap time, and two record charts one being a line chart and the other being an area chart.

- Before the stopwatch appears on the display, there should be a Text Input Dialog asking for a start time for the timer. The dialog should receive an integer number, minimally, and your timer will start from the input number. **NOTE**: Make sure you have proper error checking for the input fields, it should be a positive integer. *See Figure 1: Timer set up dialog for an example.*
- The analog display contains a circular dial with 60 tick marks where each tick mark represents 1 second of elapsed time and a sweeping hand that increments **smoothly** to the next tick mark every elapsed second. See *Figure 2: Analog Interface* for an example. It takes 60 seconds for the hand to sweep one time around the dial.
- The digital display, minimally, shows minutes, seconds, and hundredths of a second in a mm:ss.SS format. For each elapsed centisecond (i.e. hundredth of a second), the digital display is to increment 1 centisecond (i.e. hundredth of a second). The digital display is to increment 1 second after the hundredths of a second is equal to 100. (“Minimally” means you can do more if you like, which is usually rewarded when going above and beyond the rest of the class. You could display milliseconds or nanoseconds, hours, etc.)
- The timer text object starts from the time you input in the Input dialog (e.g. 60) and displays the time left after the stopwatch started, for example, if 5 seconds has passed after the stopwatch has started, the timer text will show “Timer: 55.00s”. This means the timer text, minimally, will start from the start number passed from the Input Dialog and show the remaining time left as it is counting down. Using our example of 60 seconds, when the passed time arrives at 60 seconds, the timer text will show “Time's Up!” and the “Record” button will stop working, when the button is pressed, and an alert will appear with the message “Time is up… No more records…”. In addition, the stopwatch will not be paused and will continue operating as normal no matter what happens with the timer, because the stopwatch will only be paused when you click the “Stop” button. For an example, see *Figure 3: Digital display Example.*
- The record board will be used to record the lap time and average lap time. The lap time is the time between the most recent button presses, which would be the difference between the current time and the last lap/recorded time. The average lap time is the average amount of time for each lap, so it would be the total lap time divided by the number of laps at that particular time when the button was pressed.

MVC Timer Stopwatch FXML S21

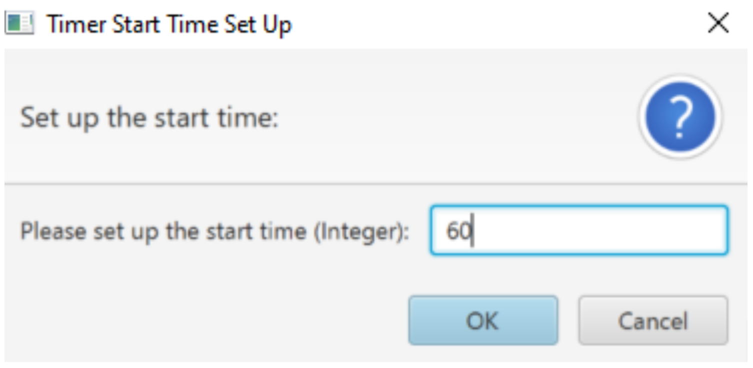


Figure 1: Timer set up dialog Example

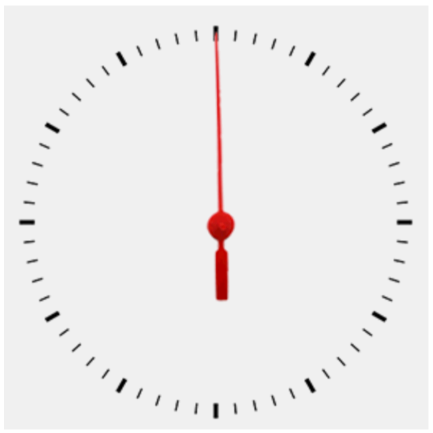


Figure 2: Analog Interface Example

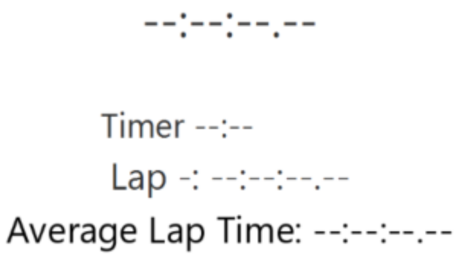


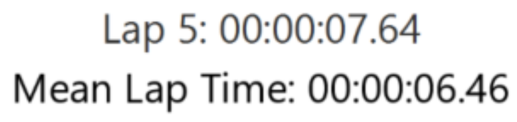
Figure 3: Digital Display Example

The stopwatch has **<u>two</u>** buttons. When the stopwatch is initialized the two buttons will be “Record” and “Start”. When the stopwatch is in motion and running, the two buttons will be “Record” and “Stop”. If the user presses the “Stop” button and the stopwatch is no longer in motion or running, then the two buttons will be “Reset” and “Start”. If the user presses the reset button, the two buttons will return to “Record” and “Start”. Therefore, the buttons will change dynamically depending on the current state of the stopwatch.

The start button will start the stopwatch and timer. The stop button will halt the stopwatch and timer. The reset button resets the displays, elapsed time, lap count, places the displays in a halt state, reset the time left in the timer, and clears the record board and record charts. The record button will take the current time displayed at the moment the button was pressed, subtract it from the previous lap time, and record the current lap time on the board in the same format as the digital display, mm:ss.SS. Therefore, pressing the record button, will result in the difference between the two button clicks or the difference between the current time and the last lap time. If you were able to press the record button, then immediately press the stop button, without letting any time lapse, the sum of all the individual lap times should equal the total time shown on the digital display. At the same time, when you click the record button, the average lap time will be calculated and update the average lap time text object. The average lap time will be the sum of all the lap times divided by the total lap count.

For the record chart, there minimally needs to be a LineChart and an AreaChart, which are both from the JavaFX FXML library, where both will have the x-axis as a CategoryAxis and the y-axis as a NumberAxis. The y-axis for the line chart and the area chart should both have the “setAutoRanging” property set to “false”, the “setLowerBound” set to 0, and the “setUpperBound” set to 5. The line chart will be used to display the line for the lap times by having the x-axis as the lap count and the y-axis as the lap time. The area chart will be used to display the area between the line for the average lap times by having the x-axis as the lap count and the y-axis as the average lap time.

For both carts, the newest record should always be placed and displayed on the right. When the “Record” button is pressed, the current lap time and current average lap time will be added to the line chart and area chart, respectively. The upper bound of the y-axis for these two charts will change dynamically depending on the largest value in these charts. In order to do this, you need to make sure the lap time and average lap time are always smaller than the upper bound of the y-axis. If the value is larger, then you need to increase the y-axis upper bound to support the new value. Use the previous challenge as an example and there is example code in the previous challenge as well. *See Figure 4: Record Board and Figure 5: Record Charts* for examples.



Lap 5: 00:00:07.64

Mean Lap Time: 00:00:06.46

Figure 4: Record Board Example

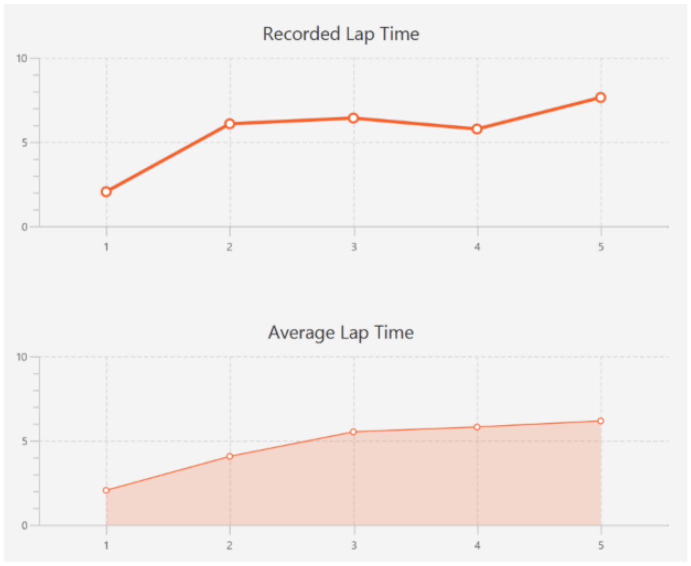


Figure 5: Record Charts Example

Starting the stopwatch causes it to increment from the last elapsed time value. If the stopwatch is at 0, the stopwatch increments from 0 when the start button is pressed. If the stopwatch is stopped at 00:07.11 and then the start button is selected, the stopwatch will increment from 00:07.11. The reset button is used to set the stopwatch to 0 and place it in a halted state. If the elapsed time of the stopwatch is 00:07.11 and the record button is pressed for the first time, then the current time of 00:07.11 will be displayed to the record board while keeping track of which lap it is. However, if the monitor has not started yet, meaning the start button has not been pressed for the first time, then the record button has no action and nothing will happen if pressed.

The digital display consists of three parts, a digital time text, a timer text, and a record area.

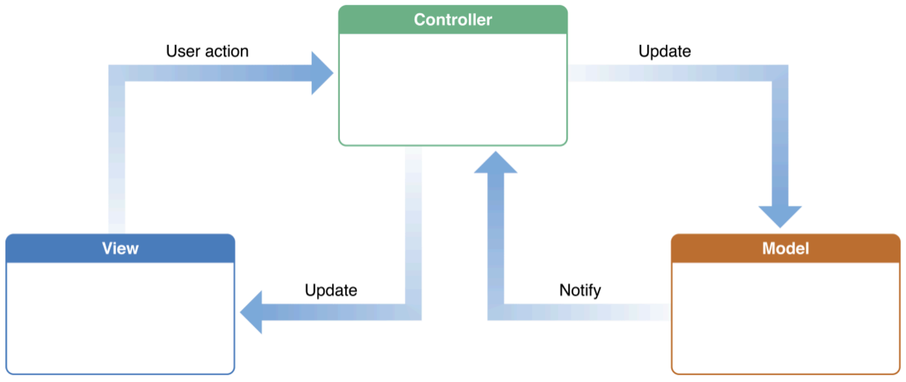
- The timer text will be initialized as the number the user input in the dialog. If the start button is pressed and as time passes, the time left will be the start seconds minus the time elapsed, assuming the timer default value is 60 seconds. If the time passed is larger than 60 seconds, the text will always show “Time’s Up!”. When the reset button is pressed, the timer text will be reset to its initial state.
- The record area consists of a record board and two record charts.
- When the “Record” button is pressed for the first time, the time on the digital display will be displayed on the record board and the record charts. If the “Record” button is pressed for the second time, the difference between the first click and the second click will be calculated (i.e. the difference between the current time and the last lap time), the average lap time value will be calculated (i.e. the average lap time is the average value of all the lap times, which is the sum or total of all lap times divided by the current lap number), then updates the records on the record board, and the calculated values will be displayed on the record charts at the same time. The newer record will always replace the older one on the record board and the newest record will always be placed on the right on the record charts.
- After the “Reset” button is pressed, the timer text and the record board will be set back to the initial state, the lap count, lap time and average lap time will be set back to zero, the record charts will be cleaned and the y-axis will be set to its initial value, and the record board and the record charts should look the same as it was in the beginning (i.e. initial state).

Note: In order to implement the average lap time below are two proposed ways, the second way is an opportunity for bonus points:

1. You have two variables, one is for the seconds elapsed and another is for the total lap time. Every time the user presses the record button, you calculate the current lap time by taking the difference of the current time and the last lap time, then you add that to the total lap time variable. Once you have the total lap time, you can divide that number by the number of laps to get the average.
2. Another approach would be to use a collection. We have not discussed collections yet in lecture, so that is why this method is a way to get bonus points. You could use an ArrayList or a LinkedList, which are built in libraries in the JDK under the “java.util” package. Every time a lap record is calculated you could store it to the collection. Then you loop through all the elements (e.g. use an iterator) in the collection, calculate the sum, and divide by the total number of laps (e.g. could even be the total number of elements now). A benefit

of keeping each individual lap time is it could be used to calculate more things like what was the best lap time, what was the worst lap time, what was the average lap time, what were all the lap times, maybe the user wants to write all the lap times to a file and store them for later, if you look at your phone's stopwatch sometimes it tells you the difference between each lap (e.g. Am I getting slower or faster and by how much), if you look at an iOS stopwatch they have a record board that can go "forever" which would be a good "real-world" implementation where if you are slower or faster it changes colors, and so on. Therefore, by using a data structure (i.e. In Java they are called collections), it opens the door for even more bonus opportunities, some of which I just stated.

The code should be separated into the proper Model-View-Controller (MVC) architecture that was discussed in class. This means the model is completely independent from the controller and the view, the controller is completely independent from the model and view, and the view is completely independent from the model and controller. In short, the model should know nothing about the view, the controller handles both managing the model and updating the view, and the view tells the controller about the user actions. Lastly, the communication between the MVC architecture should exist both ways in each layer as shown in this diagram (i.e. Two-Way Communication between each Component in MVC):



<u>Note:</u> Diagram was received from this website: [Apple Developer MVC]

This means the Controller should communicate to the View and the View should communicate back to the controller. The Controller should also communicate to the Model and the Model should communicate back to the Controller**<u>. The communication is both ways.</u>** However, the Model will never communicate with the View directly. The code in the MVC lecture demonstrates this communication technique, you can find the link on the Canvas page for this challenge. **<u>Make sure analog and digital stopwatches are in separate model classes.</u>** This means there will be an analog model and a digital model, both independent from each other.

Note: Even though you need to make two separate model classes, I don't recommend you start with two. I would get this challenge to work without any models, just an FXML file and a controller. Then when that is working, separate the code into 1 model for both analog and digital. Then when you get that working, separate the code into 2 models, step by step, just like we did in the lectures. If you start with 2 models from the beginning, you will have a rough time completing this challenge.

Two image files are provided with this challenge that you can use: clockface.png and hand.png. You are not required to use the provided images. If you want to use alternatives, that is okay. The clockface.png image is the dial of the stopwatch. The hand.png image is the sweeping hand of the stopwatch.

The layout and design of the user interface is up to you. You get to decide the locations and designs of the analog display, digital display, record board, two record charts, and the two dynamically changing buttons. Whatever you choose should be a well-organized, thoughtful, aesthetically pleasing, and a useable interface or an interface that logically makes sense. The layout should look like you made intentional choices and are in control of their placement. This means the layout should not be a disorganized mess that is a result of not knowing how to implement the user interface, layout, and/or code in a meaningful way.

Note: One rule to follow is Labels should never be used for dynamic text, labels are for static text, whereas Text Objects are used for dynamic text (i.e. text that will be changing) … read the oracle docs to make sure you are using proper programming techniques.

You may expand the functionality beyond the basic requirements provided above if you choose. Usually expanding beyond the basic requirements gets rewarded if you go above and beyond the rest of the class. You could do the following for example:

- Add an additional sweeping hand that displays the minutes elapsed on the analog stopwatch
- Add an additional sweeping hand that displays the hours elapsed on the analog stopwatch
- Add an additional smaller hand, that is underneath the seconds hand, that displays hundredths of a second on the analog stopwatch
- Add hours to the digital display
- Add milliseconds to the digital display
- Add nanoseconds to the digital display
- Add additional functionality to the stopwatch (Could add additional buttons, etc.)
    - Maybe have it where the user can modify the timer stopwatch without restarting the program
    - Maybe create additional functionality that would be useful
    - Sometimes I get ideas from the Stopwatch or Timer on your cell phone or various other sources, just make sure you reference those sources.
- Change the colors of the buttons (For example, when the “Start” button is showing the button’s color could be green and when the “Stop” button is showing the button’s color could be red)
- Add additional record board text objects
- Add a hover over feature on the line graph
- Etc.

For inspiration look at the stopwatch or timer on your phone or check out this link:
https://en.wikipedia.org/wiki/Stopwatch

Run your application and make sure everything works as expected. Export the project directory located in your NetBeans project folder. You will use NetBeans to export a ZIP file and submit the ZIP on Canvas.

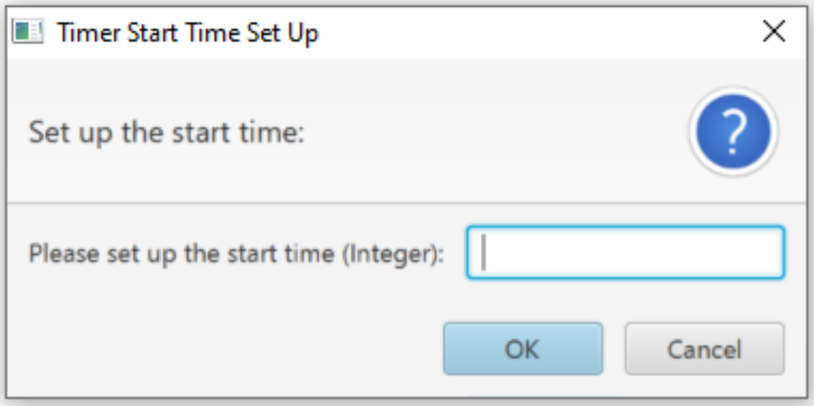
Things to Submit on Canvas:

1. You will use NetBeans to export a ZIP file and submit the ZIP on Canvas.
   - Run your application and make sure everything works as expected
   - Make sure you include all files associated with your challenge, if you use images, make sure you use relative paths and include the images in your project directory
   - Export the project directory located in your NetBeans project folder
   - Make sure to add the zip extension, otherwise it will not save properly
   - Then submit that zip file to Canvas
2. Submit screenshots of your application running for proof with the system clock and pawprint.
   - The system clock must include the current date and time in order to be valid
   - It is recommended you take screenshots of the output from your code and screenshots of **ALL** of code as well
   - Take screenshots like the examples below to show all the functionality of the app
   - The more screenshots the better

Examples Screenshots:

Note: Your UI does not need to look the same as the examples below, as long as you have the same functionality, your UI can look completely different, and I hope that you improve the design compared to the ones in the screenshots (e.g. possibility for bonus here). In addition, if there are any differences between the examples below and the requirements, then follow the requirements above.

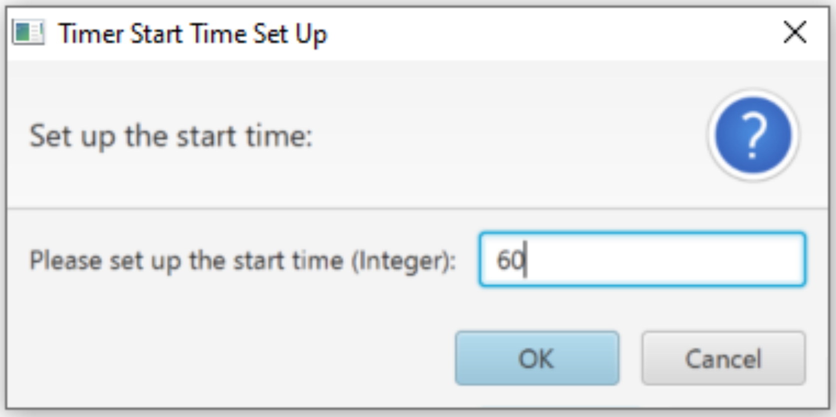
- Initial Application after Loading:


Timer Start Time Set Up

Set up the start time:

Please set up the start time (Integer):

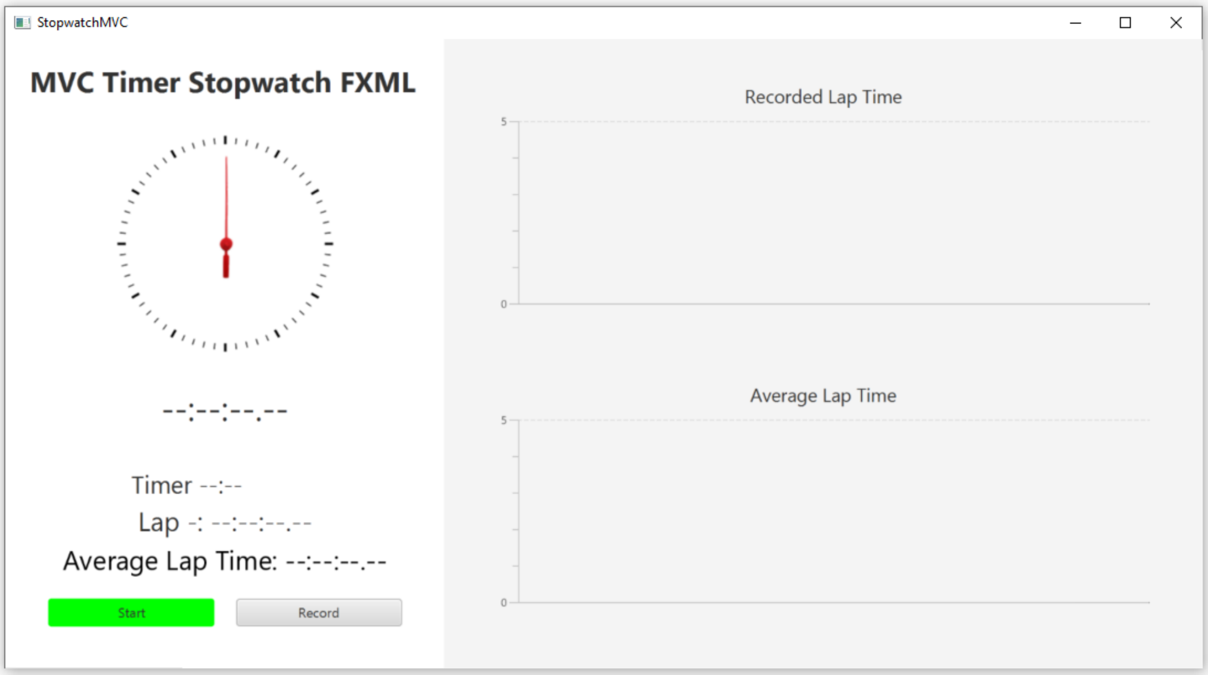
- User inputs correct value:


Timer Start Time Set Up

Set up the start time:

Please set up the start time (Integer): 60

- Default home screen after user inputs value correctly:


StopwatchMVC

MVC Timer Stopwatch FXML

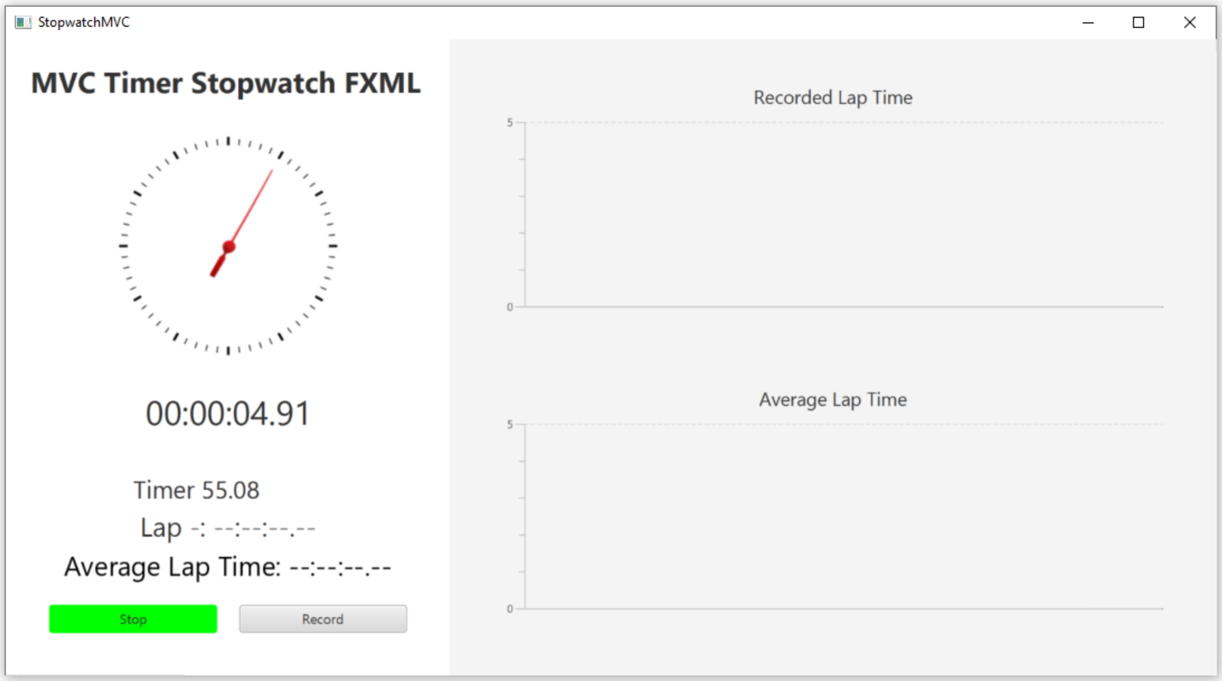
--:--:--.--

Timer --:--

Lap -: --:--:--.--

Average Lap Time: --:--:--.--

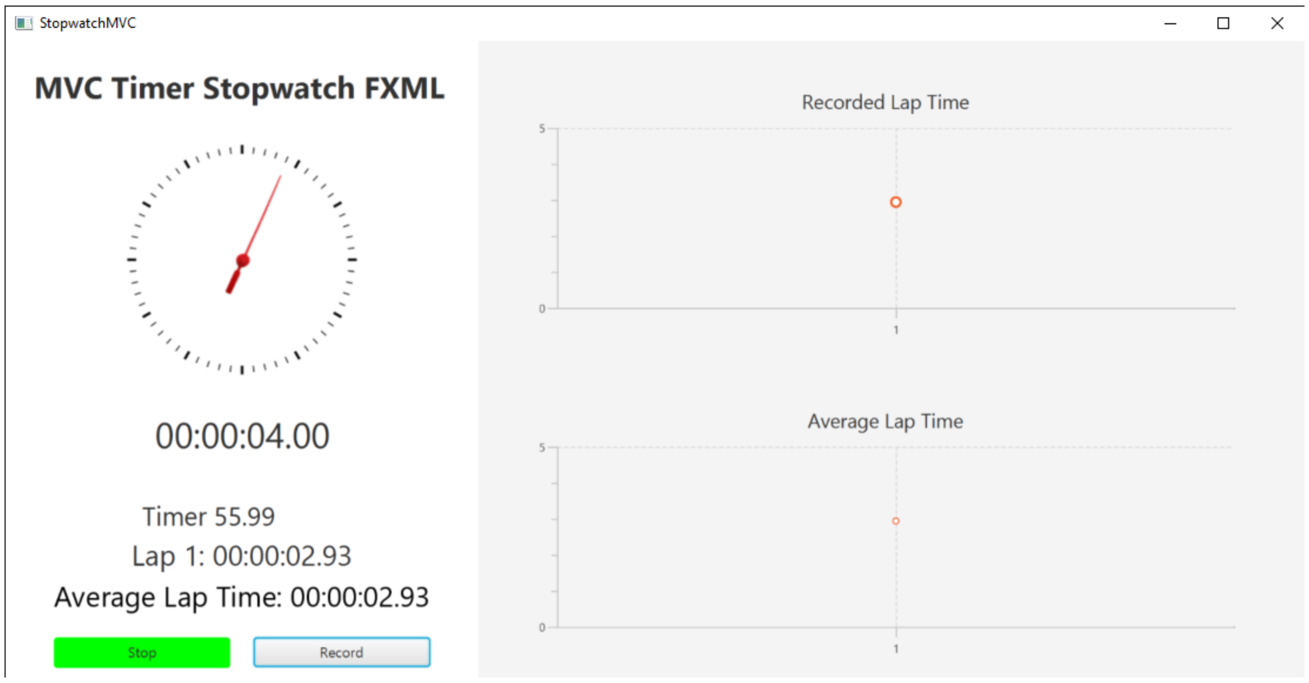
- After start button is pressed



00:00:04.91

Timer 55.08
Lap -: --:--:--.--
Average Lap Time: --:--:--.--

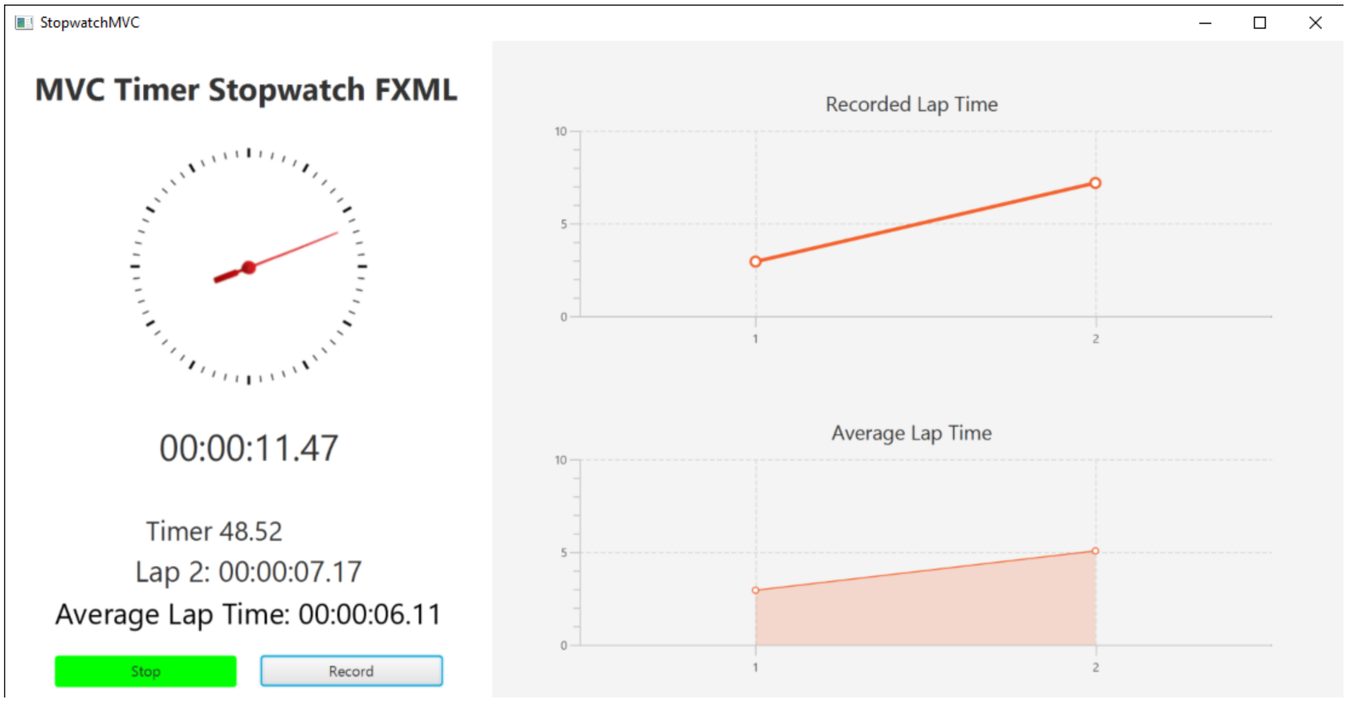
- After record button is pressed for first time



00:00:04.00

Timer 55.99
Lap 1: 00:00:02.93
Average Lap Time: 00:00:02.93

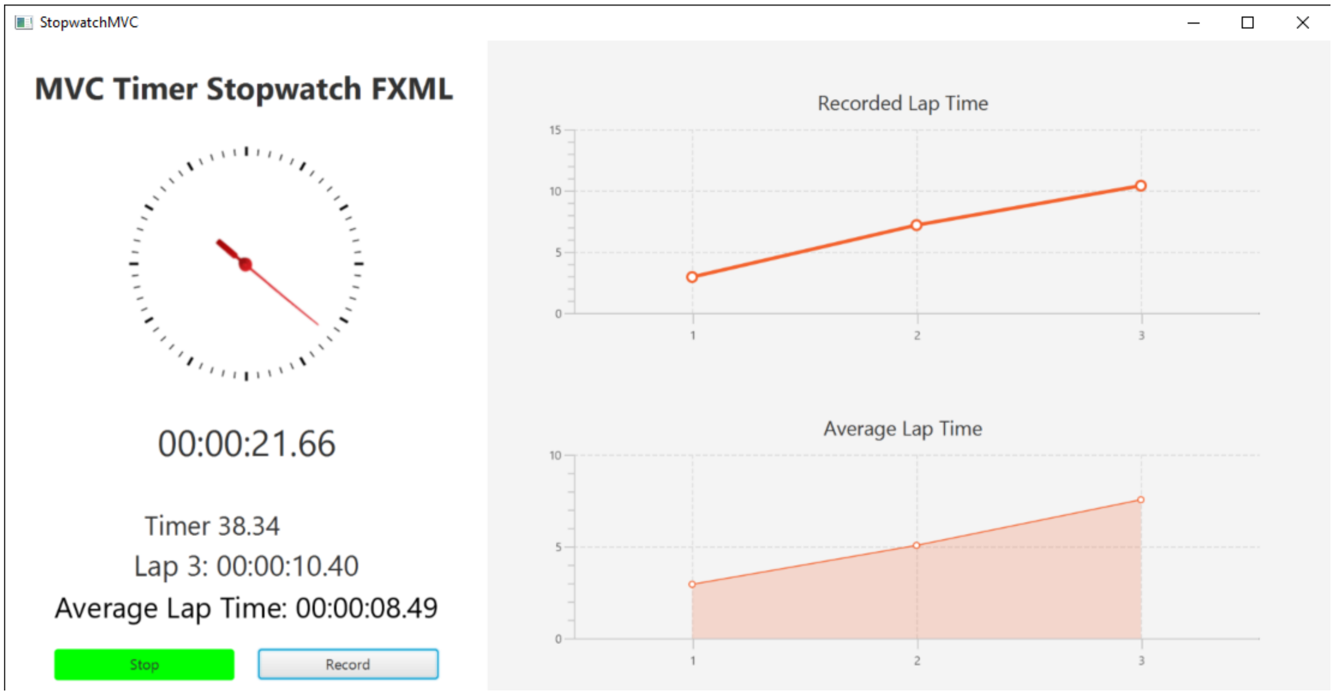
- After the record button is pressed the second time, notice how the Y-Axis upper bounds on the line chart and area chart increased because the record lap time and average lap time were greater than the maximum Y-Axis value of each chart.



00:00:11.47

Timer 48.52
Lap 2: 00:00:07.17
Average Lap Time: 00:00:06.11

- Record button pressed and the Y-Axis upper bound of the line chart increased again.

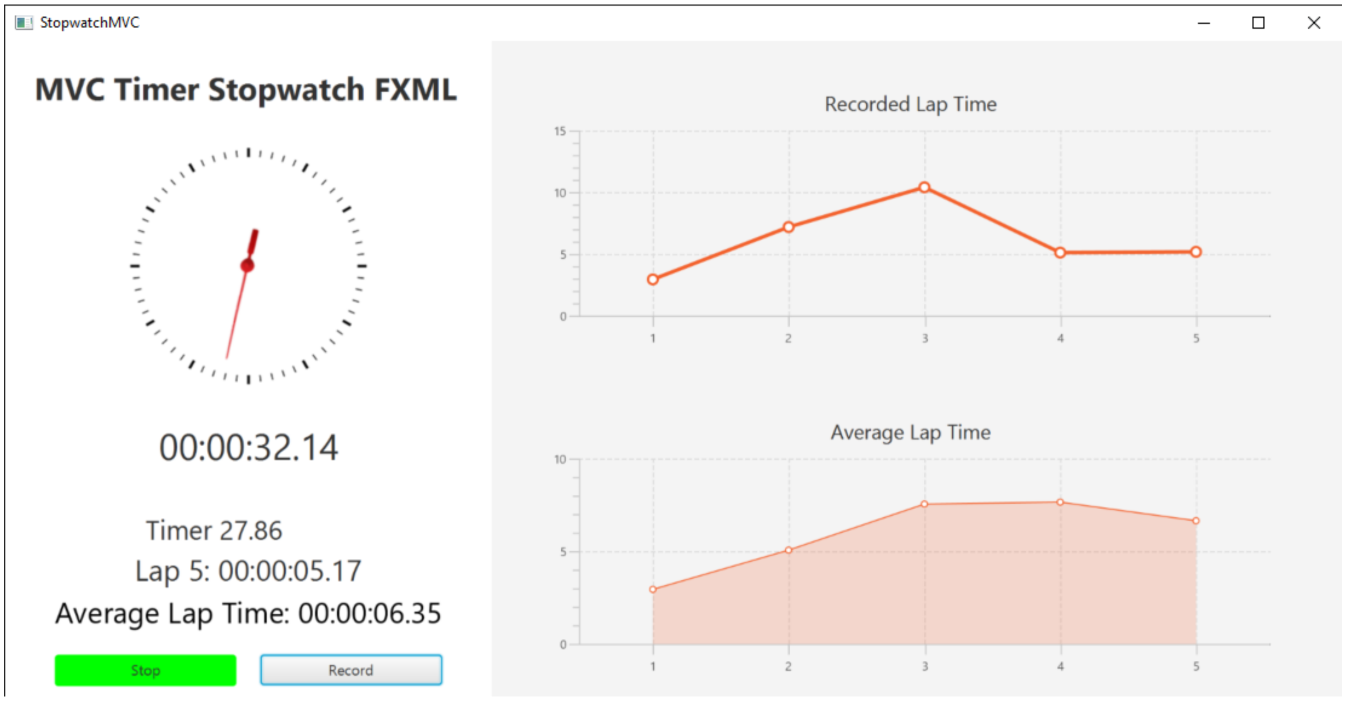


00:00:21.66

Timer 38.34
Lap 3: 00:00:10.40
Average Lap Time: 00:00:08.49

MVC Timer Stopwatch FXML S21

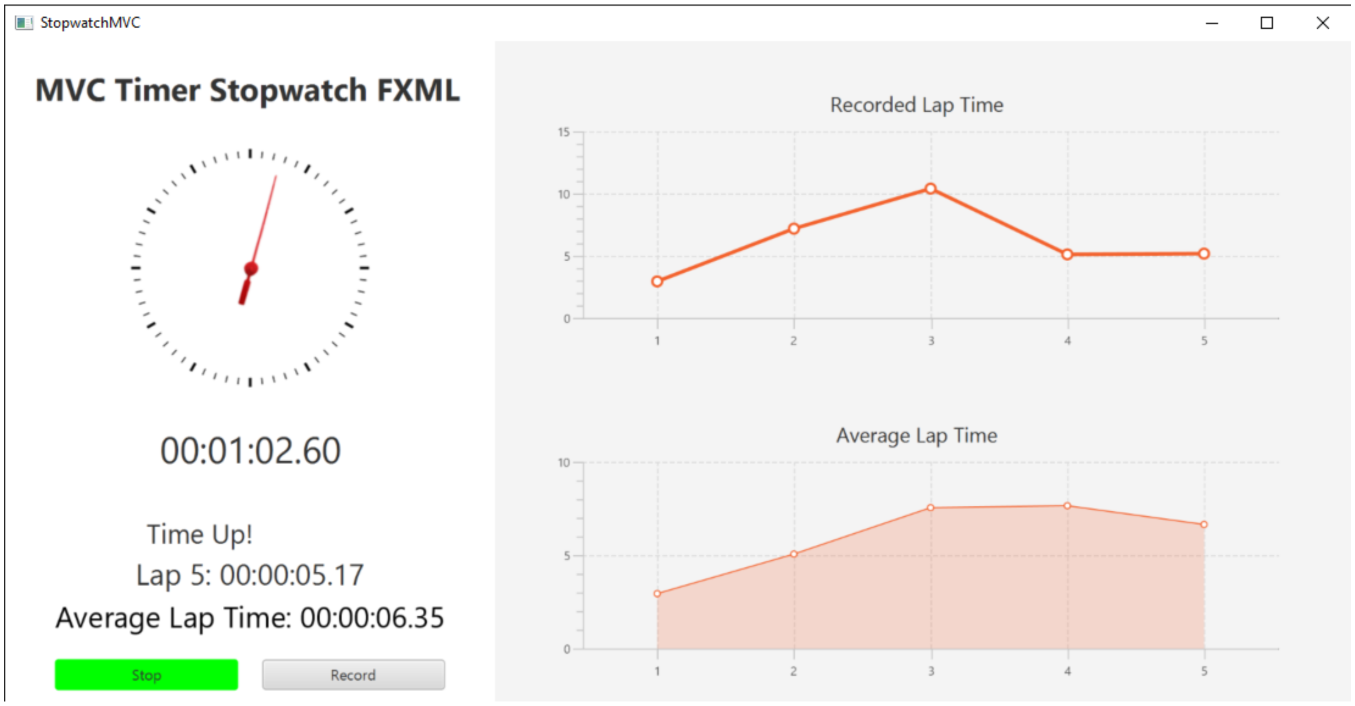
- Record button pressed a few more times:



00:00:32.14

Timer 27.86
Lap 5: 00:00:05.17
Average Lap Time: 00:00:06.35

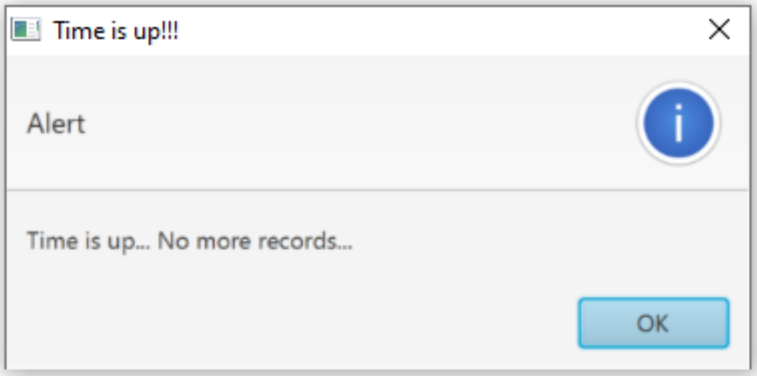
- Once 60 seconds has passed, the timer display will change to “Time Up!” but stopwatch continues



00:01:02.60

Time Up!
Lap 5: 00:00:05.17
Average Lap Time: 00:00:06.35

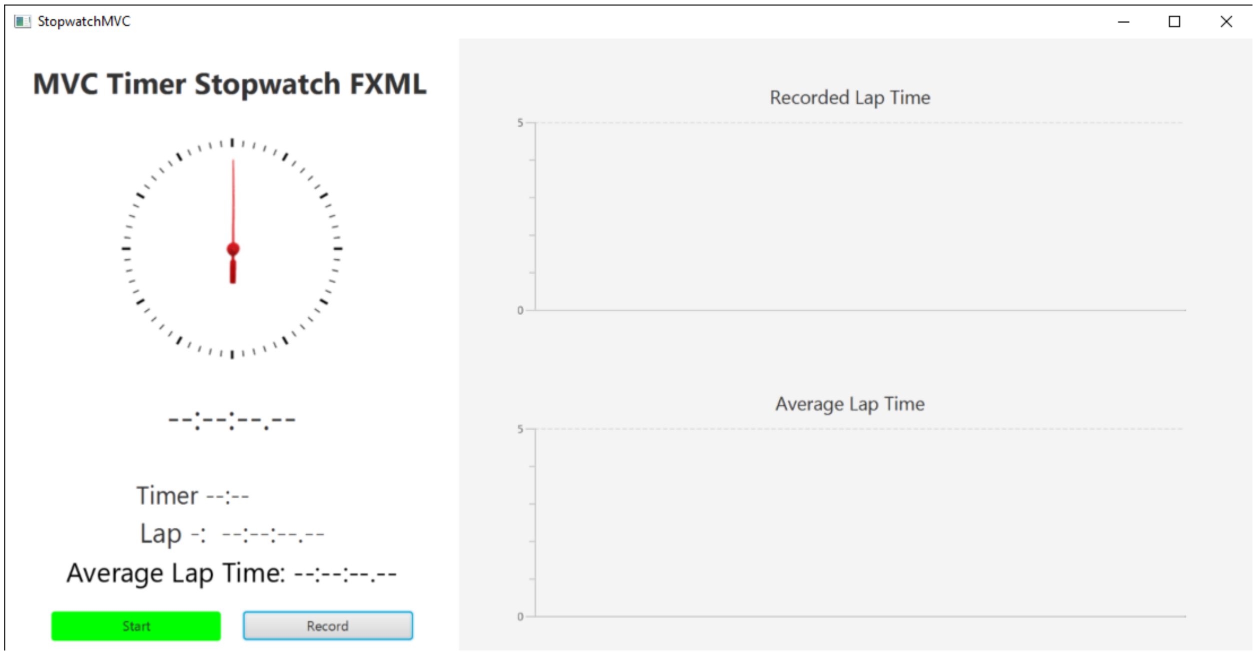
- If the user clicks on the record button again after the timer is up, they will get an error message from an alert:


Time is up!!!

Alert

Time is up... No more records...

- After the reset button is pressed the UI will go back to its initial state


StopwatchMVC

MVC Timer Stopwatch FXML

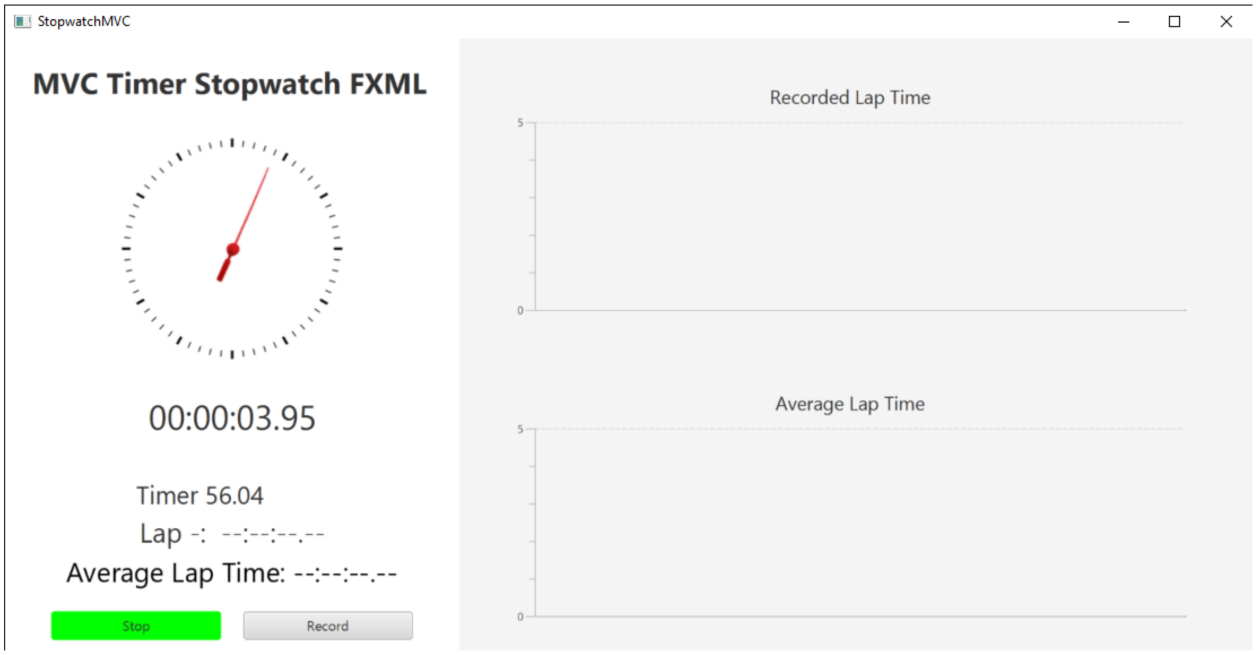
--:--:--.--

Timer --:--

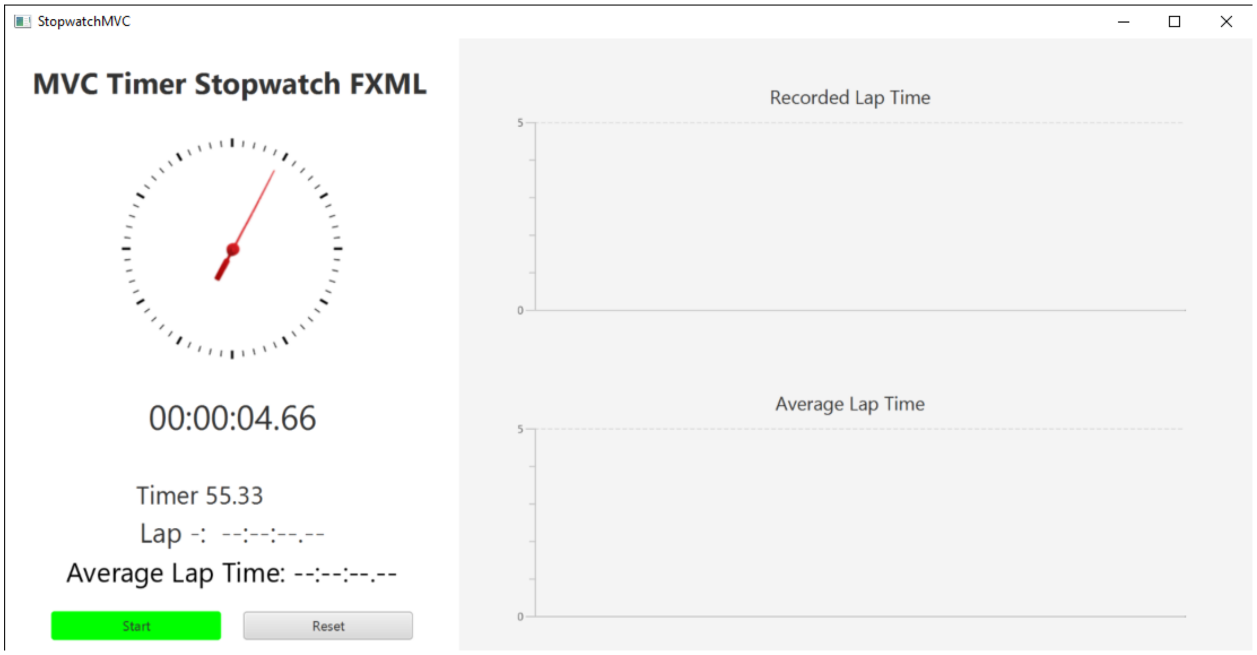
Lap -: --:--:--.--

Average Lap Time: --:--:--.--

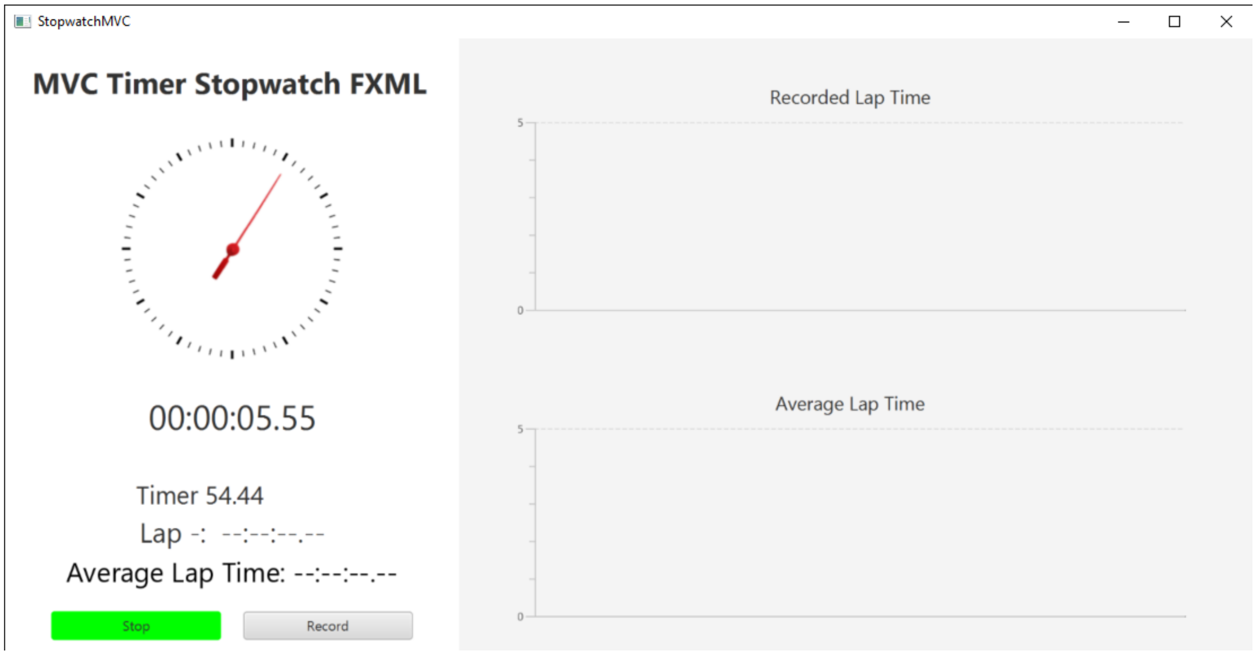
- Start the timer and stopwatch back up again, it will use the 60 second timer again … however, you could have the app ask the user each time before they start for the timer input … you could have an additional button to change the countdown timer initial time, by calling the setup count down timer method when the user clicks on the button … many different implementations can be applied here for bonus.



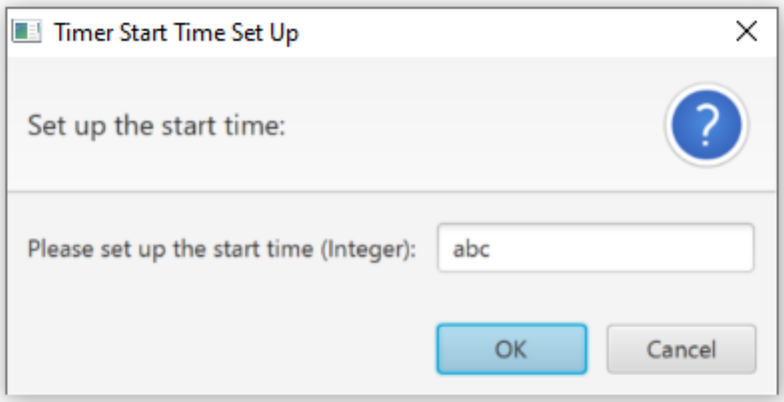
- After the stop button is pressed …



- Then the start button is pressed again, to start everything back where it left off (Note: The time displayed is a little off because it took me some time to click the start button and then snap the screenshot).



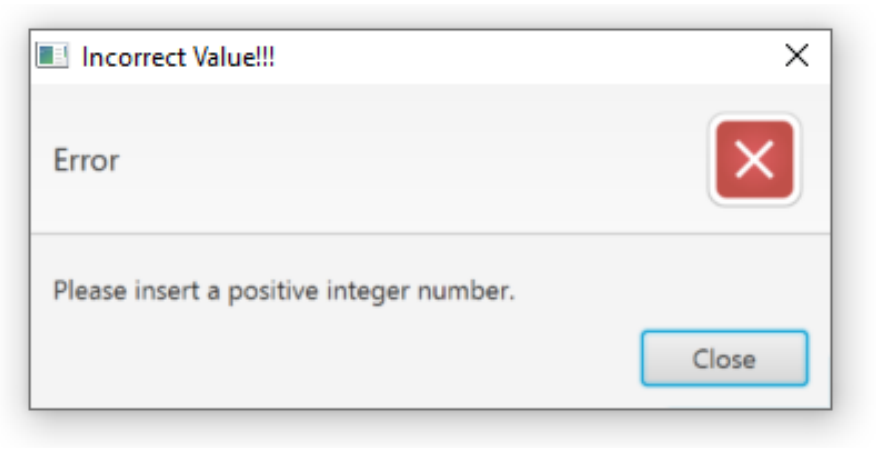
- Testing user inputs, if the user enters an incorrect number on the input dialog:



Set up the start time:

Please set up the start time (Integer): abc

- Then they should get some error message and then be allowed to try again until they get it correct

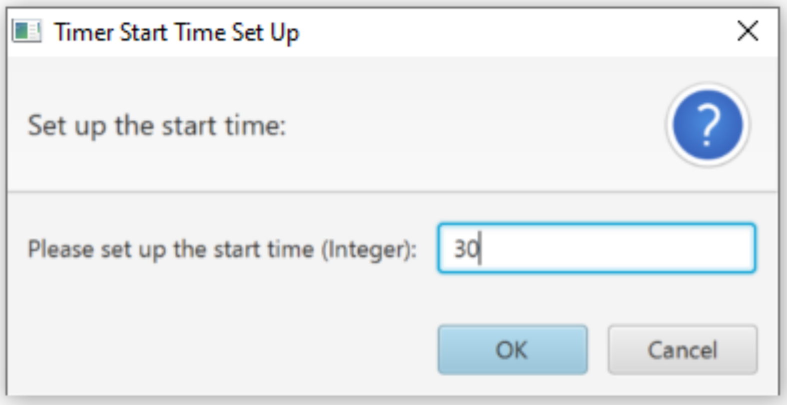


Incorrect Value!!!

Error

Please insert a positive integer number.

- Once correct …

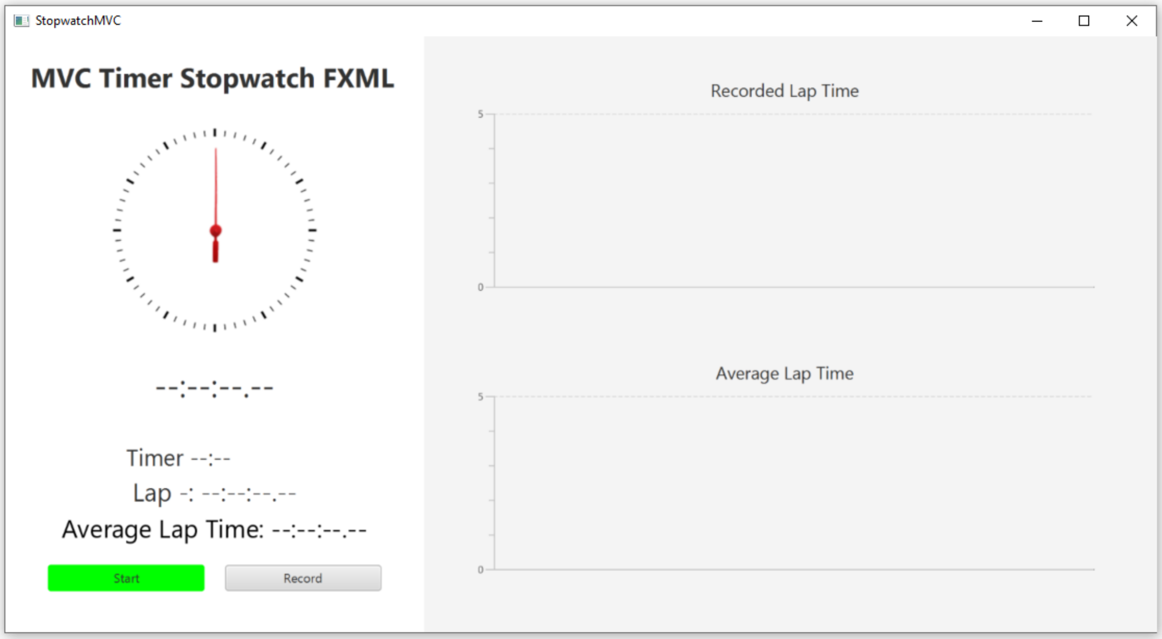


Timer Start Time Set Up

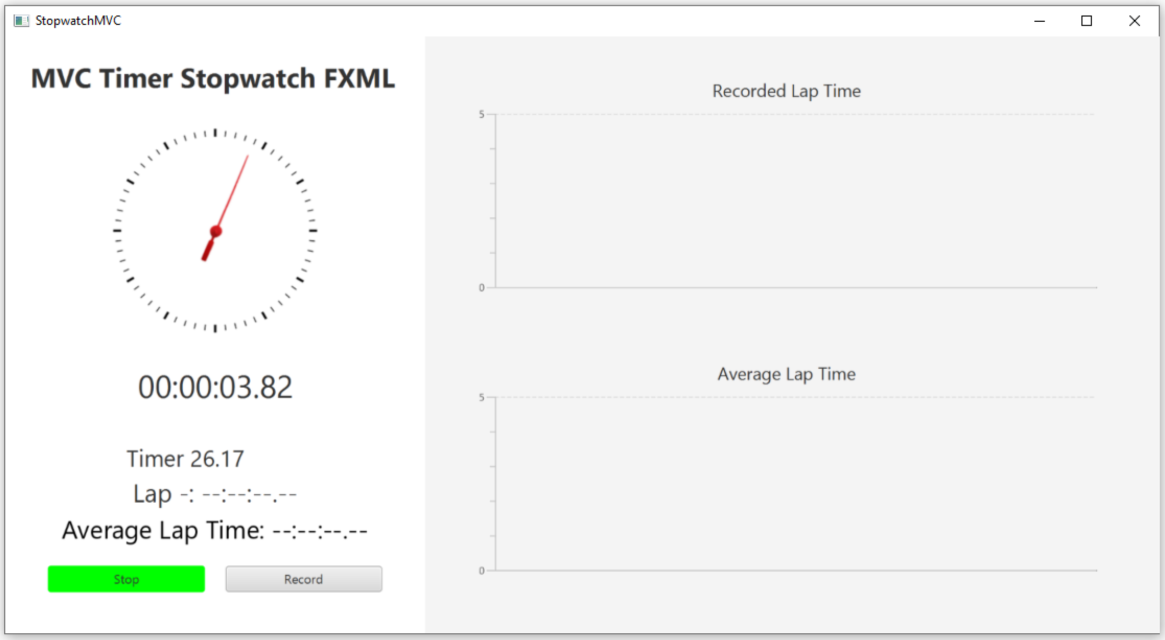
Set up the start time:

Please set up the start time (Integer): 30

- They should be directed to the home page



- Where they can start the timer and stopwatch


StopwatchMVC
MVC Timer Stopwatch FXML
00:00:03.82
Timer 26.17
Lap -: --:--:--.--
Average Lap Time: --:--:--.--
Stop
Record
Recorded Lap Time
5
0
Average Lap Time
5
0