FXML CPU Monitor S21

Description:

Create a Java 8 SE application in NetBeans using JavaFX FXML and an associated controller that implements a CPU monitor having an analog display, two buttons, a digital display, a peak usage text object, a mean usage text object, a record chart, and a load chart. The analog display is comprised of a dial with tick marks and a sweeping hand that displays real time CPU usage. The digital display shows real time CPU usage as a percentage in decimal format from 0.00%-100.00% depending on the system's activity during the period being observed.

The CPU monitor has **two** buttons. When the monitor is first initialized the two buttons will be "Record" and "Start". When the monitor is running the two buttons will be "Record" and "Stop". When the monitor is stopped, or paused, the two buttons will be "Reset" and "Start". The text on the two buttons should be displayed dynamically and change dynamically.

The peak usage text object will always show the peak CPU usage value once the monitor has started running. The peak CPU's usage value will be 0.00% before the monitor has started. After reset button is pressed, the peak CPU usage value will be reset back to 0.00%.

The mean usage text object will always show the mean of the CPU usage once the monitor has started running. The default mean CPU usage value will be 0.00% before the monitor has started. After reset button is pressed, the mean CPU usage value will be reset back to 0.00%.

When the "Start" button is pressed, the load chart will record each mean CPU value on the AreaChart automatically as the values are being displayed on the mean usage text object.

The record chart will be used to record the CPU usage. When the "Record" button is clicked, the number on the digital display and the record count (number of records) will be recorded, a new point will be added to the line chart. If the monitor has not started, then the record button should not have any action when clicked.

NOTE: Labels should be used for text that will not frequently change whereas Text Objects should be use for text that will changed frequently. If you use a Label and change the text frequently, then points will be deducted. The first sentence on the Oracle Docs (our textbook) says a "A Label is a non-editable text control," so make sure you use it as such.

FXML CPU Monitor S21

Purpose:

This challenge develops skills in creating and manipulating JavaFX interfaces, using FXML and Scene Builder, using a controller associated with an FXML interface, learning and implementing the Model-View-Controller architecture, generating and handling events, and organizing code.

Essential Requirements:

- Project Name: <Pawprint>FXMLCPUMonitorS21
- 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 FXMLCPUMonitorS21. For example, if the pawprint is **abcxyz9,** the project is to be named **Abcxyz9FXMLCPUMonitorS21**.
- This challenge allows you to apply your creativity while meeting a set of minimum 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 it.
- The design of UI depends on you, an example will be given in case you don't know how or where to start, however, you can design your own cool CPU monitor for possible bonus!
- Platform: Java 8 SE, JavaFX, FXML with Controller
- Project IDE: NetBeans

Requirements:

- Create a CPU monitor application that contains an analog display, two buttons, a digital display, a peak usage text object, a mean usage text object, a record chart, and a load chart. The analog display contains a gauge with 11 tick marks, starting from 0, where each tick mark represents 10% of CPU usage and a sweeping hand that will change between ticks based off the current CPU load. See *Figure 1: Analog and Digital Interfaces* for an example.
- The digital display, minimally, shows the CPU usage as a percentage to the nearest 100^{th} decimal place in a --.--% format ("Minimally" means you can do more if you'd like which is usually rewarded).
- The record chart will be used to display a line chart with x-axis as the record number and y-axis as the CPU load. The newest record should always be placed and displayed on the right. See *Figure 2: Record Chart* for an example.
- The peak usage text object will always show the peak (i.e. maximum) CPU usage value since the monitor has started running. The mean usage text object will always show the mean CPU

usage value since the monitor has started running. See *Figure 3: Peak Usage Label & Mean Usage Label* for an example.

- The mean CPU load chart will record continually what the digital display is recording. See *Figure 4: Mean CPU Load chart* for an example.

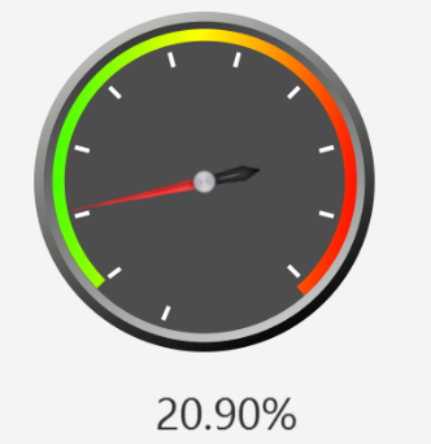


Figure 1:
Analog and Digital Interfaces Example

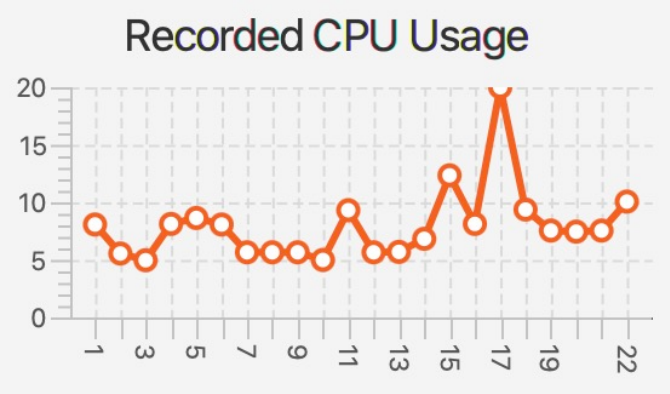


Figure 2:
Record Chart Example

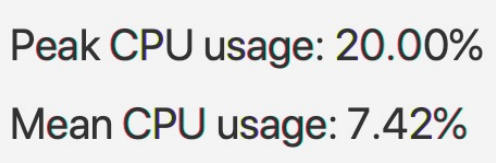


Figure 4:
Peak & Mean Usage Label Example

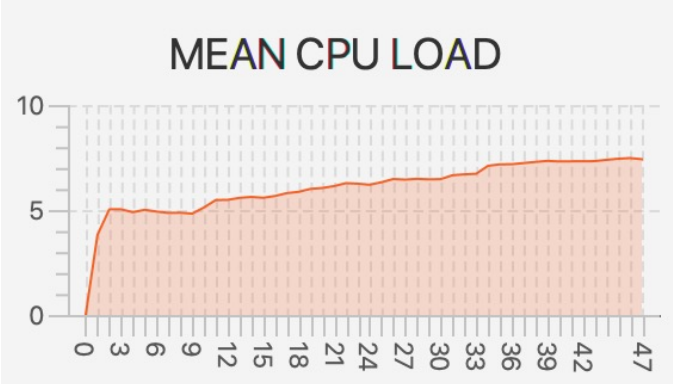


Figure 3:
Mean CPU Load Chart Example

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

FXML CPU Monitor S21

The start button will start the monitor. The stop button will halt the monitor. The reset button resets the analog, digital displays, two usage text objects, places the displays in a halt state, and clears both charts. The record button will take the current CPU usage displayed at the moment in time the button was pressed and record the percentage on the line chart. Starting the monitor causes the app to get the real time CPU usage from the entire system based on a 0.00-100.00% value depending on whether the system is idle or active during the recent period being observed. If the monitor is at 0 and the start button is pressed, the monitor starts from 0. If the monitor is stopped at 15.00% and then the start button is pressed again, the monitor will start from 15.00%. The reset button is used to place the monitor in a halted state **and** set it to 0. For the record chart, the newest record should be placed on the right of the previous one, with each point in ascending order. 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 peak usage text object will always show the peak (i.e. the Maximum) CPU usage value since the monitor has started. If the monitor has not started yet, then the peak CPU usage value is 0%. After the reset button is pressed, the peak CPU usage value will reset back to 0%. For example, let's say the first update to the CPU usage value you obtained a value of 8%, then the peak CPU usage value will be 8%. Then the second update to the CPU usage value occurs and you obtain a value of 10%, then peak CPU usage value will be 10%, since 10% > 8%. Lastly, the third update to the CPU usage value occurs and you obtain a value of 6.00%, the peak CPU usage value will remain 10.00%, as the peak value doesn't change because 6.00% < 10.00%. Then this method will continue as the monitor continues to run.

The mean usage text object will always show the mean (i.e. the Average) CPU usage value since the monitor has started. If the monitor has not started yet, the mean CPU usage value will be 0%. After reset button is pressed, the mean CPU usage value will be reset back to 0%. For example, let's say the first update to the CPU usage value you obtained a value of 8%, then the mean CPU usage value will be 8%, since 8/1 = 8. Then the second update to the CPU usage value occurs and you obtain a value of 10%, then the mean CPU usage value will be 9%, since 8+10=18 and 18/2 = 9. Lastly, the third update to the CPU usage value occurs and you obtain a value of 6%, the mean CPU usage value would be 8%, since 18+6=24 and 24/3 = 8. Then this method will continue as the monitor continues to run, try to come up with a method to save as much space as possible. You will also want to let your monitor run for awhile to see if the values becomes too large to store.

FXML CPU Monitor S21

The record chart is a LineChart from the JavaFX FXML library, with the x-axis as a CategoryAxis and y-axis as a NumberAxis (don't forget to assign the fx:id for the two axes in the Scene Builder). The y-axis should have the "setLowerBound" to 0. When the "Record" button is clicked, the current CPU usage percentage will be converted to a float number between 0 and 100 and will be added to the line chart. After the "Reset" button is clicked, the line chart should be cleared.

The mean CPU load chart is an AreaChart from the JavaFX FXML library, with the x-axis as a CategoryAxisArea and y-axis as a NumberAxisArea (don't forget to assign the fx:id for the two axes in the SceneBuilder). The y-axis should have the have the "setAutoRanging" property to "false", the "setLowerBound" to 0, and the "setUpperBound" to 100. The AreaChart will be updated at the same time the mean usage label is updated. The x-axis will be the duration of one keyframe and the y-axis will be the mean usage value at that time. If the keyframe duration is 1 second, then every second the mean CPU load chart will be updated with the value of the digital display. If the CPU value returned from the getCPUUsage() method is null, then the value will not be recorded on the load chart.

Below is an example of the code to help you understand how the LineChart works:

```
private LineChart<String, Number> lineChart;
private CategoryAxis xAxis;
private NumberAxis yAxis;
private XYChart.Series<String, Number> series;

series = new XYChart.Series();
yAxis.setAutoRanging(false);
yAxis.setLowerBound(0);
yAxis.setUpperBound(100);
lineChart.getData().add(series);
lineChart.setAnimated(false);

// To add points to the chart, for example, add point (1, 100) and (2, 80)
series.getData().add(new XYChart.Data(Integer.toString(1), 100));
series.getData().add(new XYChart.Data(Integer.toString(2), 80));
```

The following is the code you should use to get the real time CPU usage in a double format:

```
public double getCPUUsage() {
   OperatingSystemMXBean operatingSystemMXBean =
                              ManagementFactory.getOperatingSystemMXBean();

    double value = 0;
    for(Method method : operatingSystemMXBean.getClass().getDeclaredMethods()) {
        method.setAccessible(true);
        if (method.getName().startsWith("getSystemCpuLoad")
                && Modifier.isPublic(method.getModifiers())) {
            try {
                value = (double) method.invoke(operatingSystemMXBean);
            } catch (Exception e) {
                value = 0;
            }
            return value;
        }
    }
    return value;
}
```

`double getSystemCpuLoad()` returns the "recent cpu usage" for the entire system. This value is a double in the [0.0, 1.0] interval. A value of 0.0 means that all CPUs were idle during the recent time period observed, while a value of 1.0 means that all CPUs were actively running 100% of the time during the recent time period observed. All values between 0.0 and 1.0 are possible depending of the activities happening in the system at any given time. If the system's recent CPU usage is not available, the method returns a negative or NULL value, which should be handled on your display correctly, without changing the original getCPUUsage() method. In addition, if the code above goes into the catch statement, meaning there was an error when trying to get the current CPU value, then you should not modify the display with a null value. To do this, check for null values before you do any updates to the display.

NOTE: A null value is common, however you need to handle this case to make sure there are no invalid values on the digital display or chart, otherwise your program will either crash or look like it is not working correctly. There are multiple ways to fix this problem, here are two basic ideas:

- Skip this update and do not apply any update
- Replace this invalid value with the latest valid value and continue to update

FXML CPU Monitor S21

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

NOTE: Make sure you use relative paths for the images in scene builder. Otherwise, your images will not load when grading your assignment and points will be deducted, you have been warned.

Gauge.png

Hand.png

How to design the user interface layout is up to you. You get to decide the locations of the analog display, digital display, record chart, mean CPU load chart, two usage text objects, 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.

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

- Add additional functionality to the monitor or additional buttons
- 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 more information

    - Get the available memory
    - Get the current memory used
    - Which processes/programs are running
    - For each process, how much CPU usage is it taking
    - Get the CPU time for each process
    - Etc.
- Make the clock more precise for each record
- Add additional sweeping hands to the analog display gauge
- Have a hover over feature where it will show the system time for each point
- You can look at the memory usage program on your local machine for ideas, what are cool things that were implemented, and what could you try to implement after you meet the minimum requirements?
- Etc.

Run your application and make sure everything works as expected. Export the project directory to a ZIP using NetBeans and submit it on Canvas.

Things to Submit on Canvas:

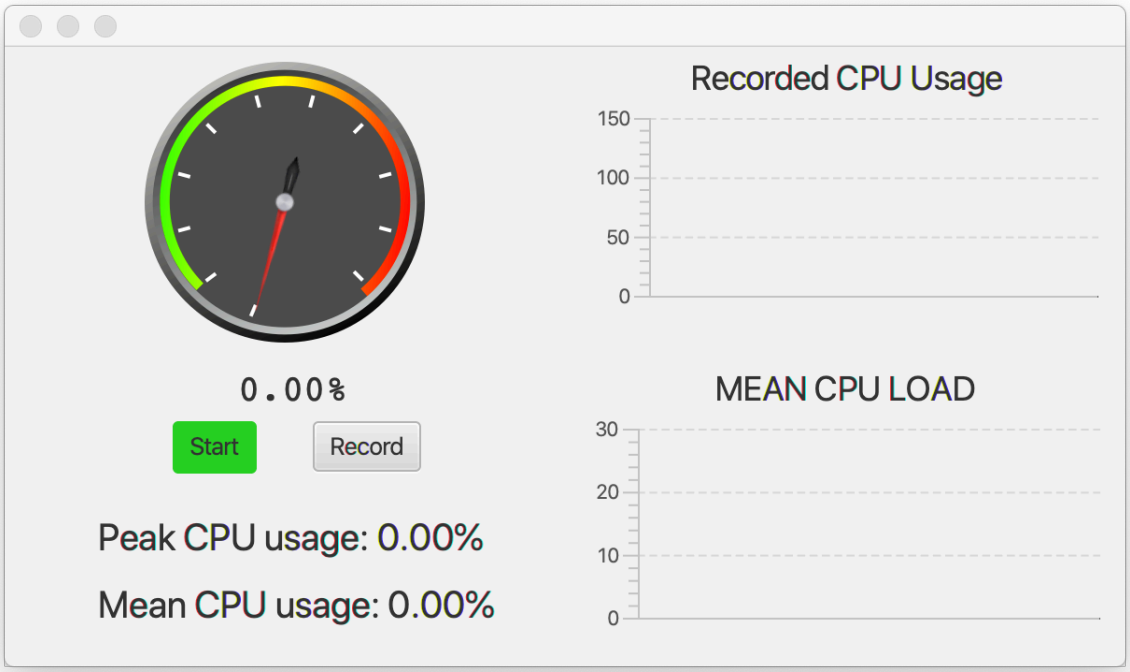
1. You will use NetBeans to export a ZIP file and submit the ZIP on Canvas.
    - Run your application and make sure everything works as expected
    - Make sure you include all files associated with your challenge, if you use images, make sure you use relative paths and include the images in your project directory
        - Use a similar naming convention as the project name for ALL assets for your projects
        - For example if you use your own dial image or hand image then name it
            - <Pawprint>FXMLCPUMonitorS21DailImage.png
            - or <Pawprint>FXMLCPUMonitorS21HandImage.jpg … etc.
    - 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. Screenshots can be submitted as extra files on the Canvas assignment submission (preferred method) along with your zip file (All in one submission)
    - The system clock must include the current **date and time** in order to be valid
    - You must take screenshots of the output from your code
    - Show the application running and the functionality you built

- Similar to the ones shown above as examples
- You must also take **screenshots of ALL of the code**
- The more screenshots the better

Example Screenshots:

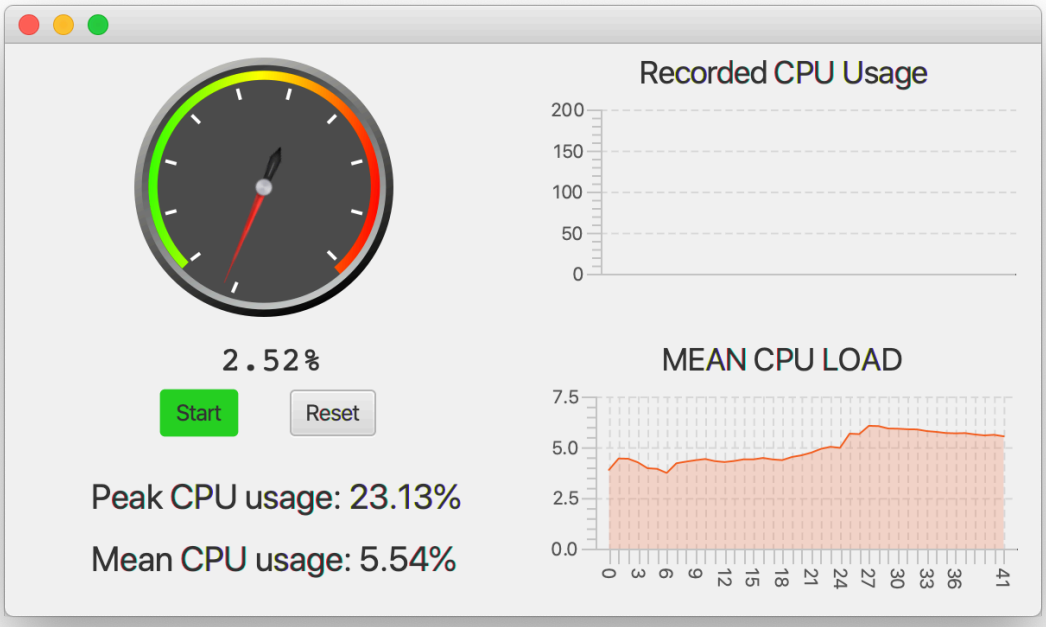
Note: Your UI does not need to look the same as the examples below, as long as you have the same functionality, your UI can look completely different. In addition, if there are any differences between the examples below and the requirements, then follow the requirements above.

Initial Application after Loading:

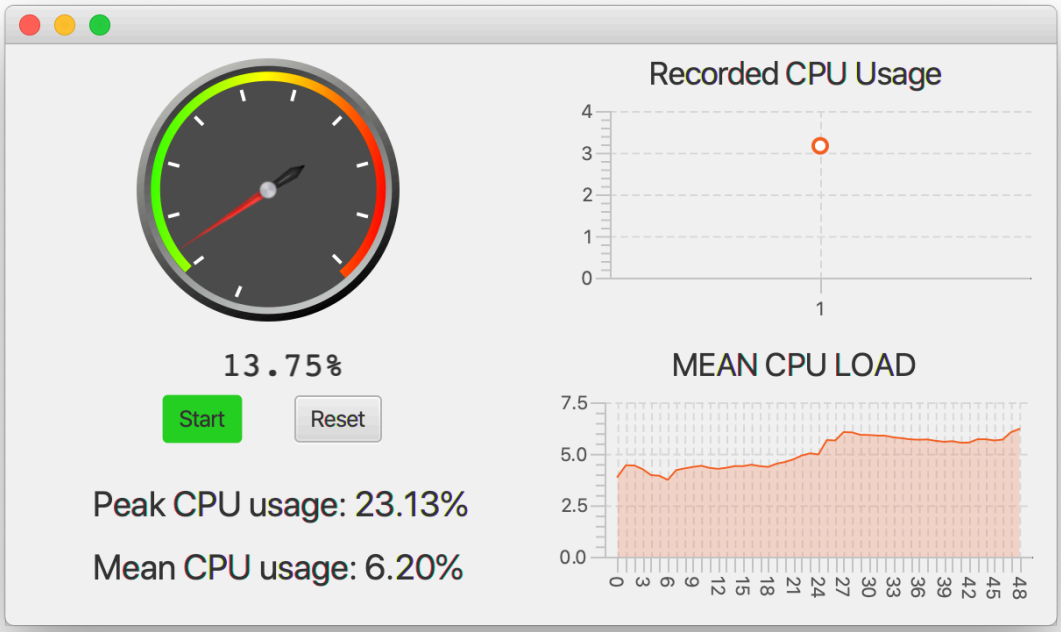


FXML CPU Monitor S21

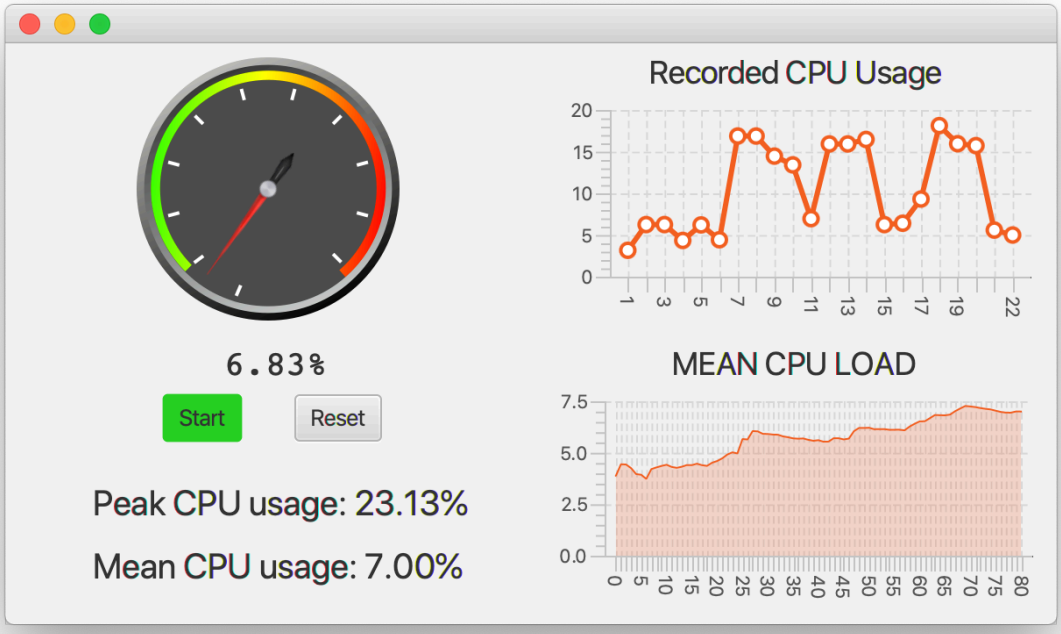
Start button pressed and monitor running for ~10 iterations with valid (i.e. not null) CPU values:



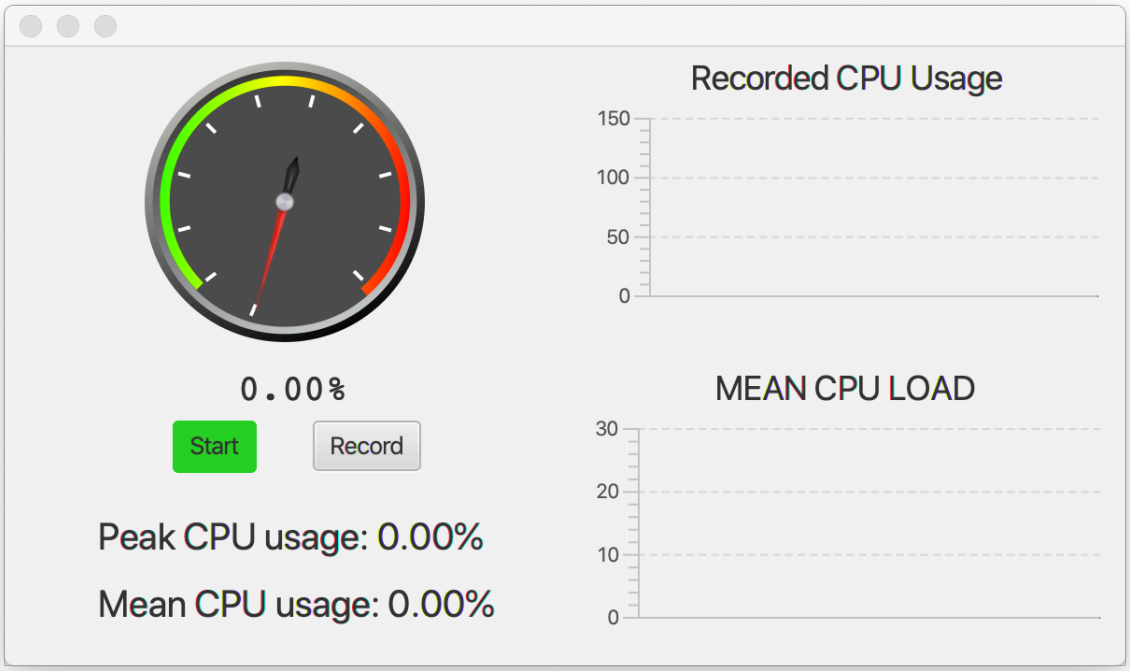
Pressing the Record button for the first time, then immediately stopping the monitor afterwards:



Pressing the Record button several times and then stopping the monitor:

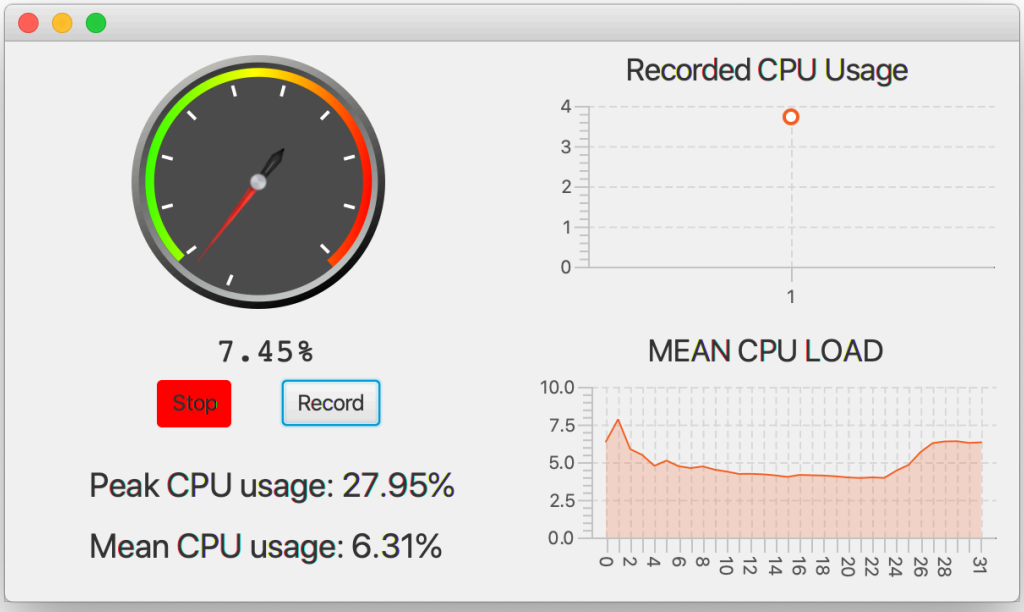


After the Reset button was clicked:



FXML CPU Monitor S21

Starting the Monitor back up and pressing the Record button while letting the Monitor run:



Pressing the Record button several times while letting the Monitor Run: