Assignment 5 5/31/2023

Implement the next release of your term project, employing *concurrency* if concurrency can fit; otherwise create a different project with concurrency. The same instructions apply as before to this completed Word document, the gray text, the 5 page limit, appendices, JUnit tests, and a ReadMe file. As usual, copying someone else’s application without clear attribution is plagiarism, and will be subjected to the College’s academic conduct process.

## 1 SUMMARY DESCRIPTION

One- or two-paragraph overall description of your proposed term project. Color red the parts changed from Assignment 2, if any (all in red if this is a separate application).

The Personal Health Information (PHI) app is a comprehensive solution designed to assist users in monitoring and managing their personal health. It allows users to create an account, securely log in, and track various health metrics such as BMI, blood pressure, and cholesterol levels. The app enables users to input and store their health data, providing a user-friendly interface for seamless data entry and updates. The app provides a user-friendly interface for adding and managing health data, making it easy for users to enter and update their information. It also offers the option to create custom health entries, where users can record specific health issues or concerns and add notes to track their progress.

With a strong emphasis on privacy and data security, the app ensures that each user has full control over their health information. Users can choose to keep their data private or share it with healthcare professionals or family members as needed. The app has the potential to not only prevent unwanted future health complications, but also act as a motivator to keep the health in track. Overall, the Health Tracker app provides a convenient and centralized platform for users to monitor and maintain their personal health records, empowering them to make informed decisions about their well-being.

### >>AI generation OR check: *I did not use AI generation here* \_NO\_(Collapse section before submitting.)

PARAGRAPH DESCRIBING YOUR VALUE ADDED TO THE AI-GENERATED MATERIAL

MY PROMPT SEQUENCE (ONLY)

[1]

CODE OR TEXT PRODUCED BY THE ABOVE PROMPT SEQUENCE

chatGPT X

## 2 ADDITIONAL REQUIREMENTS (FEATURES) IMPLEMENTED IN THIS RELEASE

Title and one or two sentences per requirement. Don’t repeat requirements implemented for prior assignments unless they are necessary to provide context—in which case, make it clear they are old.

### 2.1 Concurrent file reading (NEW)

### The app should be able to read the "HealthMetrics.txt" file concurrently

### 2.2 HealthDataEntry Page should run efficiently (NEW)

### The app should be handle button press in the HealthData Entry page simultaneously

### >>AI generation OR check: *I did not use AI generation here* \_\_(Collapse section before submitting.)

PARAGRAPH DESCRIBING YOUR VALUE ADDED TO THE AI-GENERATED MATERIAL

Chat gpt answers were not exactly what I wanted so I clarified it wit more description. The answers were slightly better second time however, I made changes and edit them. I simplified the response by getting rid of extra details and made them such that my app can achieve and with less complications.

MY PROMPT SEQUENCE (ONLY)

[1] I need to create two or more functional requirements for this week's iteration about concurrency in my app. Give me three functional requirements each no more than a sentence long.

[2] I need to create two or more functional requirements for this week's iteration about concurrency in my app. Give me three functional requirements each no more than a sentence long. Remember, functional requirements means what the app shall be able to do, not how it would do it, in other words, don't include implementations.

CODE OR TEXT PRODUCED BY THE ABOVE PROMPT SEQUENCE

chatGPT X

[1] Sure! Here are three functional requirements related to introducing concurrency in your app:

Asynchronous data retrieval: Implement a background thread or task to fetch health metric data from external sources, allowing the user interface to remain responsive during the retrieval process.

Concurrent filtering of health metrics: Enable concurrent execution of filtering operations on health metric data to ensure the user interface remains responsive, even when working with large datasets.

Parallel computation of average values: Utilize parallel processing techniques to calculate average values of health metrics, reducing the computation time and providing real-time updates to the user interface.

These requirements focus on utilizing concurrency to enhance the performance and responsiveness of your app's data retrieval, filtering, and computation processes.

[2] Apologies for the confusion. Here are three functional requirements related to concurrency in your app, focusing on what the app should be able to do:

Concurrent data retrieval: The app should be able to retrieve health metric data from external sources concurrently, allowing multiple data retrieval operations to occur simultaneously.

Parallel filtering of health metrics: The app should support parallel processing of health metric data, enabling concurrent filtering operations to improve performance and responsiveness when filtering large datasets.

Real-time updating of average metrics: The app should provide real-time updates of average metric values, utilizing concurrency to calculate averages in parallel and immediately reflect any changes in the displayed values.

## 3 I/O SUPPORTING THE NEW REQUIREMENTS LISTED ABOVE

Provide an example of input / output showing the new features of your application.

### Input / Output

Javafx gui concurrency in the healthdataentry page:

A screenshot of a computer

Description automatically generated with medium confidence  
Here you can see how multiple states opens up smottly and this process of opening and closing these scenes are handled separately by JavaFX Platform concurrently for smooth operation of the gui.   
  
After I submit them, following is the proof of how the filereader concurrency being put to work by simultanelsuly read the file and compare the input values against the idealvalues.  
A screenshot of a computer

Description automatically generated

## 4 YOUR DIRECTORY

Show a screenshot of your directory. Include your “.dat” files (where objects are written). This should include JUnit tests—class-by-class, and method-by-method, except for trivial and inappropriate ones.

Your response replaces this.

## 5 DESIGN

Supply a main use case, the class model, and the sequence diagram corresponding to the use case. These should be consistent. Indicate in red your class model where you applied object read, object write, streams and lambdas. Excellent assignments will typically include the use of Java FX (speak to your facilitator first if you wish to use alternative API’s) and event-driven programming.

Your response replaces this.

### >>AI generation OR check: *I did not use AI generation here* \_\_(Collapse section before submitting.)

PARAGRAPH DESCRIBING YOUR VALUE ADDED TO THE AI-GENERATED MATERIAL

MY PROMPT SEQUENCE (ONLY)

[1]

CODE OR TEXT PRODUCED BY THE ABOVE PROMPT SEQUENCE

chatGPT X

## 6 CODE SNIPPETS

### 6.1 Code showing where concurrency is *defined*

(1) In the HealthMetricsReader class:  
A picture containing text, font, screenshot

Description automatically generated  
The definition of concurrency is seen in the creation of the ExecutorService which allocates 4 threadpools to perform the filereading task concurrently.

(2) In the healthdataentry class here is one area where concurrency is applied   
A picture containing text, screenshot, font, line

Description automatically generated  
All the showXXXXX() functions have this javafx specific concurrency in the submit button actions.

### 6.2 Code showing where concurrency is *used*

(1) In the HealthMetricsReader Class:  
A screenshot of a computer program

Description automatically generated with medium confidence

### For (1) usage of concurrency is seen in the submission of the Runnable task to the ExecutorService for concurrent execution. A FileReadertask from Runnable interface is created when running the filereader. It gets submitted to the executor, future.get makes it wait until the current task it over and lastly updates the idealvalues array.

(2) In the HealthdataEntry class, one example is given from one of the showXXXXX() functions:  
A screenshot of a computer code

Description automatically generated with medium confidence

The code snippets from healthdataentry class demonstrate where concurrency is applied (using Platform.runLater()) and where it is being used (enclosing UI operations that need to run on the JavaFX Application Thread).

### >>AI generation OR check: *I did not use AI generation here* \_\_(Collapse section before submitting.)

PARAGRAPH DESCRIBING YOUR VALUE ADDED TO THE AI-GENERATED MATERIAL

When I saw the chatgpt using callable interface, I thought it should be used when I need something to be returned. However, the filereading task in my case don’t need to return anything, it just needs to fill and update the idealvalues array, so I changed the callable to Runnable interface for concurrency which I thought in my case, is more fitting. And for the second requirement it gave me very complex concurrency related to java fx consisting of TASK, executor, etc. I then did research and found out there is an easier way which is by using Platform.runlater. I utilized chatgpt’s help in this, but I gave it the idea which made the process and code easy to read and understand.

MY PROMPT SEQUENCE (ONLY)

[1] For the first requirement, show me how and where in the code I can implement concurrency.

CODE OR TEXT PRODUCED BY THE ABOVE PROMPT SEQUENCE

chatGPT X

[1] import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.concurrent.Callable;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Future;

public class HealthMetricsReader {

private static final ExecutorService executor = Executors.newFixedThreadPool(4);

public static double[] readIdealValue(String metric) {

double[] idealValues = new double[2];

try (BufferedReader reader = new BufferedReader(new FileReader("C:\\Users\\mahim\\Documents\\GitHub\\CS622\\Assignments\\Hw3\\HealthMetrics.txt"))) {

Future<double[]> future = executor.submit(new FileReaderTask(reader, metric));

idealValues = future.get();

} catch (IOException e) {

System.out.println("An error occurred while reading the HealthMetrics file: " + e.getMessage());

} catch (Exception e) {

System.out.println("An error occurred while executing the file reading task: " + e.getMessage());

}

return idealValues;

}

private static class FileReaderTask implements Callable<double[]> {

private final BufferedReader reader;

private final String metric;

public FileReaderTask(BufferedReader reader, String metric) {

this.reader = reader;

this.metric = metric;

}

@Override

public double[] call() throws Exception {

double[] idealValues = new double[2];

String line;

while ((line = reader.readLine()) != null) {

String[] values = line.split(",");

if (values.length >= 3 && values[0].trim().equals(metric.trim())) {

idealValues[0] = Double.parseDouble(values[1].trim());

idealValues[1] = Double.parseDouble(values[2].trim());

break;

}

}

return idealValues;

}

}

}

## 7 YOUR CODE

Unless your facilitator arranges another method, copy your Eclipse project to your file system, zip it, and attach it to your Blackboard response. Please contact your facilitator in advance if you require an exception.

Your response replaces this.

### >>AI generation OR check: *I did not use AI generation here* \_\_(Collapse section before submitting.)

PARAGRAPH DESCRIBING YOUR VALUE ADDED TO THE AI-GENERATED MATERIAL

MY PROMPT SEQUENCE (ONLY)

[1] I have a javafx gui with buttons to ask user for specific healthmetrics and save it so it can be viewed in the history screen. There. it can be filtered by date, also the average values of the health metrics are shown. Where do you think concurrency will be beneficial and easy to implement?

CODE OR TEXT PRODUCED BY THE ABOVE PROMPT SEQUENCE

chatGPT X

[1] In your scenario, there are a few areas where introducing concurrency can be beneficial and relatively easy to implement:

Fetching data from external sources: If your application needs to retrieve data from external sources, such as an API or a database, you can perform these operations in a background thread. This allows the UI to remain responsive while the data is being fetched. For example, when loading historical health metrics from a remote server, you can use a separate thread to fetch the data, and once it's retrieved, update the UI with the results.

Filtering data by date: When filtering health metrics by date, if the dataset is large, the filtering process can be time-consuming. By performing the filtering operation in a background thread, you ensure that the UI remains responsive during the process. Users can continue interacting with the application while the filtering is in progress. Once the filtering is complete, you can update the UI with the filtered results.

Calculating average values: If calculating average values of health metrics involves a large dataset or complex calculations, it can be a good candidate for concurrency. By performing the calculations in a separate thread, you prevent the UI from freezing while the averages are being computed. Once the calculations are finished, you can update the UI with the average values.

To implement concurrency in these areas, you can utilize Java's ExecutorService or the JavaFX concurrency utilities. Here's a general outline of how you can approach it:

When fetching data from external sources, use a separate thread or an ExecutorService to perform the network request and retrieve the data. Once the data is obtained, update the UI accordingly.

When filtering data by date, initiate the filtering operation in a background thread using an ExecutorService. As the filtering progresses, you can update the UI with the partial results or provide a progress indicator. Once the filtering is complete, update the UI with the final filtered data.

When calculating average values, use a background thread or an ExecutorService to perform the calculations. Update the UI with the partial results or provide a progress indicator if the calculations take a significant amount of time. Once the calculations are finished, update the UI with the final average values.

Remember to handle any necessary thread synchronization if multiple threads access shared resources or modify the same data. Also, ensure that the UI updates are performed on the JavaFX Application Thread using the Platform.runLater() method.

By introducing concurrency in these areas, you can enhance the responsiveness of your application and provide a smoother user experience, especially when dealing with large datasets or computationally intensive operations.

## 8 Evaluation

