5/31/2023

CS 622 Draft Assignment 5

Please submit a draft of Assignment 5, using the Word template supplied, with *only* the following parts drafted:

2.1

6—a first draft

### 2.1 Concurrent file reading (NEW)

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

### >>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.

### 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;

}

}

}