5/10/2023

CS 682 Draft Assignment 2

Please submit a draft of Assignment 2, using the Word template supplied, with *only* the following parts drafted:

3.1, 6.1 showing exception class, and 6.2

## 3 REQUIREMENTS IMPLEMENTED IN THIS RELEASE NOT IMPLEMENTED BEFORE *EVALUATION CRITERION (i) APPLIES*

### 3.1 Input exception

The application shall be able to throw exception if some of the user health input for health data is not positive

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

1. PARAGRAPH DESCRIBING YOUR VALUE ADDED TO THE AI-GENERATED MATERIAL

I simplified the AI response. I edited the response so that it only includes simple title and a simple sentence of what it shall be able to achieve.I also added the exception requirement which is 3.1. I believe it is one of the most important requirements for this iteration of the project.

1. MY PROMPT SEQUENCE (ONLY)

[1] I want some functional requirements for this weeks iteration of my project PHI. I want the requirements to be consisted of blood pressure, cholesterol, diabetis, and bmi. Keep in mind that this week I will checking and compare the values against the ideal values listed in a file.

1. CODE OR TEXT PRODUCED BY THE ABOVE PROMPT SEQUENCE

chatGPT  
3.1 Be able to tell if the user blood pressure status:

The application shall be able to compare the user's systolic and diastolic blood pressure inputs against the ideal range for blood pressure. It shall provide feedback to the user indicating whether their blood pressure is within the ideal range or not.

3.2 Be able to tell if the user cholesterol status:

The application shall be able to compare the user's LDL and HDL cholesterol inputs against the ideal range for cholesterol levels. It shall provide feedback to the user indicating whether their cholesterol levels are within the ideal range or not.

3.3 Be able to tell if the user blood sugar status:

The application shall be able to compare the user's blood glucose level input against the ideal range for blood sugar. It shall determine if the user is diabetic or not based on their blood glucose level and provide appropriate feedback.

3.4 Be able to tell if the user bmi status:s

The application shall be able to calculate the user's Body Mass Index (BMI) using their age, height, and weight inputs. It shall compare the calculated BMI against the ideal range for BMI and provide feedback to the user indicating whether their BMI is within the acceptable range or not.

### 6.1 Class model and Sequence Diagram

Indicate clearly in your class model where you applied file IO and exception handling, including a user-defined exception if possible. “Enforce what you intend.” For example, make classes and members *static* or not as per their intended usage. To do this use tools, PowerPoint, or combine models as in [this RUML example](https://docs.google.com/spreadsheets/d/1vBmDVtWWh3EX0oehFFLRU0P6eR-fn4d0qVg1-XOUooM/edit?usp=sharing) (which you are free to copy, cut and paste from). Insert indications in red (as in the example) to show where the three features below apply.

Class Diagram:

A picture containing text, diagram, plan, screenshot

Description automatically generated

Sequence Diagram:

A screenshot of a computer

Description automatically generated with medium confidence

I have included the IO exception and user defined exception class in the uml diagram. The validate() in CommonHealthData class throws HealthDataException class.

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

1. PARAGRAPH DESCRIBING YOUR VALUE ADDED TO THE AI-GENERATED MATERIAL

ChatGPT did a very good job of identifying function calls and their purposes. It made my job simpler. However, it was too perfect to be true. So, I removed the for loop of the healthDAta and emitted that from the sequence diagram as I thought it was too much and too repetitive. I simplified it. I also removed the print out statements as it was too much information too. Instead, I edited the diagram to only include the functions that were called and returned and the sequence diagram is much simpler and easy to read now. Additionally, I added aggregation, multiplicities, and dependencies to the uml class diagram that was generated by the chatgpt.

1. MY PROMPT SEQUENCE (ONLY)

[1]

List the function calls, including the functions initiating each call, when this program runs with the following input: I gave it the Main Class code.

[2] generate a sequence diagram in the Plantuml text format

[3] generate a class uml (from the classes I provided earlier)diagram in the Plantuml text format

1. CODE OR TEXT PRODUCED BY THE ABOVE PROMPT SEQUENCE

chatGPT X

chatGPT X

[1] Based on the provided code, here are the function calls that occur when the program runs with the given input:

User constructor:

Parameters: "John", "Doe", "johndoe@example.com", "Passw0rd", new Date(), "Male", "555-1234"

CommonHealthData constructor:

Parameters: user.getFullName(), new Date(), "Blood Pressure", 120, 60

CommonHealthData validate method:

Called on commonHealthData1

User addHealthData method:

Parameter: commonHealth1

CommonHealthData constructor:

Parameters: user.getFullName(), new Date(), "Cholesterol", 120, 60, 140

CommonHealthData validate method:

Called on commonHealthData2

User addHealthData method:

Parameter: commonHealth2

CommonHealthData constructor:

Parameters: user.getFullName(), new Date(), "Blood Glucose", 100

CommonHealthData validate method:

Called on commonHealthData3

User addHealthData method:

Parameter: commonHealth3

CommonHealthData constructor:

Parameters: user.getFullName(), new Date(), "BMI", 145.0, 65.0

CommonHealthData validate method:

Called on commonHealthData4

User addHealthData method:

Parameter: commonHealth4

CustomHealthData constructor:

Parameters: "Ankle pain", new Date(), "ankle pain on 5-12-2023"

User addHealthData method:

Parameter: customHealth

Printing user's health data:

Prints the user's full name and the string "'s health data:".

Iterating over health data:

Loops through each HealthData object in user.getHealthDataList().

Printing health data details:

Prints the metric and recorded date of each health data object.

Prints the metric of each health data object.

For CommonHealthData objects:

Checks if the metric is "Blood Pressure".

Prints the systolic and diastolic blood pressure values.

Calls HealthDataChecker.checkBloodPressure with commonHealthData.

Checks if the metric is "Cholesterol".

Prints the LDL, HDL, and triglyceride cholesterol values.

Calls HealthDataChecker.checkCholesterol with commonHealthData.

Checks if the metric is "Blood Glucose".

Prints the blood sugar level.

Calls HealthDataChecker.checkBloodGlucose with commonHealthData.

Checks if the metric is "BMI".

Prints the height, weight, and BMI.

Calls HealthDataChecker.checkBMI with commonHealthData.

For CustomHealthData objects:

Prints the notes.

Exception handling:

Catches HealthDataException and prints the error message.

[2] @startuml

class Main {

+main(args: String[]) : void

}

class User {

+User(firstName: String, lastName: String, email: String, password: String, birthDate: Date, gender: String, phone: String)

+getFullName() : String

+addHealthData(healthData: HealthData) : void

+getHealthDataList() : List<HealthData>

}

abstract class HealthData {

+HealthData(metric: String, date: Date)

+getMetric() : String

+getDate() : Date

}

class CommonHealthData {

+CommonHealthData(name: String, date: Date, metric: String, systolicBP: int, diastolicBP: int)

+CommonHealthData(name: String, date: Date, metric: String, ldlCholesterol: int, hdlCholesterol: int, triglycerideCholesterol: int)

+CommonHealthData(name: String, date: Date, metric: String, glucoseLevel: double)

+CommonHealthData(name: String, date: Date, metric: String, height: double, weight: double)

+validate() : void

+getSystolicBP() : int

+getDiastolicBP() : int

+getLdlCholesterol() : int

+getHdlCholesterol() : int

+getTriglycerideCholesterol() : int

+getGlucoseLevel() : double

+getHeight() : double

+getWeight() : double

+calculateBMI() : double

}

class CustomHealthData {

+CustomHealthData(name: String, date: Date, notes: String)

+getNotes() : String

}

class HealthDataChecker {

+checkBloodPressure(common: CommonHealthData) : void

+checkCholesterol(common: CommonHealthData) : void

+checkBloodGlucose(common: CommonHealthData) : void

+checkBMI(common: CommonHealthData) : void

}

class HealthDataException {

+HealthDataException(message: String)

+getMessage() : String

}

Main --> User

Main --> CommonHealthData

Main --> CustomHealthData

Main --> HealthDataChecker

Main --> HealthDataException

User --> HealthData

User "1" \*-- "\*" HealthData : contains

HealthData <|-- CommonHealthData

HealthData <|-- CustomHealthData

CommonHealthData --> HealthDataChecker

CommonHealthData "1" o-- "1" User

@enduml

[3] @startuml

class User {

- firstName: String

- lastName: String

- email: String

- password: String

- dateOfBirth: Date

- gender: String

- phoneNumber: String

- healthDataList: ArrayList<HealthData>

+ User(firstName: String, lastName: String, email: String, password: String, dateOfBirth: Date, gender: String, phoneNumber: String)

+ addHealthData(healthData: HealthData): void

+ removeHealthData(healthData: HealthData): void

+ getHealthDataList(): ArrayList<HealthData>

+ getFullName(): String

+ getEmail(): String

+ setEmail(email: String): void

+ getPhoneNumber(): String

+ setPhoneNumber(phoneNumber: String): void

}

abstract class HealthData {

- name: String

- date: Date

+ HealthData(name: String, date: Date)

+ getName(): String

+ getDate(): Date

+ abstract getMetric(): String

+ abstract getData(): String

}

class CustomHealthData {

- notes: ArrayList<String>

+ CustomHealthData(name: String, date: Date, note: String)

+ addNote(note: String): void

+ getNotes(): ArrayList<String>

+ getMetric(): String

+ getData(): String

}

class CommonHealthData {

- metric: String

- systolicBP: int

- diastolicBP: int

- age: int

- weight: double

- height: double

- ldlCholesterol: int

- hdlCholesterol: int

- triglycerideCholesterol: int

- glucoseLevel: double

+ CommonHealthData(name: String, date: Date, metric: String, systolicBP: int, diastolicBP: int)

+ CommonHealthData(name: String, date: Date, metric: String, ldlCholesterol: int, hdlCholesterol: int, triglycerideCholesterol: int)

+ CommonHealthData(name: String, date: Date, metric: String, glucoseLevel: double)

+ CommonHealthData(name: String, date: Date, metric: String, weight: double, height: double)

+ getMetric(): String

+ getSystolicBP(): int

+ getDiastolicBP(): int

+ getAge(): int

+ setAge(age: int): void

+ getWeight(): double

+ setWeight(weight: double): void

+ getHeight(): double

+ setHeight(height: double): void

+ getLdlCholesterol(): int

+ setLdlCholesterol(ldlCholesterol: int): void

+ getHdlCholesterol(): int

+ setHdlCholesterol(hdlCholesterol: int): void

+ getTriglycerideCholesterol(): int

+ setTriglycerideCholesterol(triglycerideCholesterol: int): void

+ getGlucoseLevel(): double

+ setGlucoseLevel(glucoseLevel: double): void

+ calculateBMI(): double

+ validate(): void

+ getData(): String

}

class HealthDataChecker {

+ checkBloodPressure(common: CommonHealthData): void

+ checkCholesterol(common: CommonHealthData): void

+ checkBloodGlucose(common: CommonHealthData): void

}

User --> HealthData

HealthData <|-down- CustomHealthData

HealthData <|-down- CommonHealthData

CommonHealthData --> HealthDataChecker

@enduml

### 6.2 Code showing *file I/O*

Show the relevant code (only). It should be clear where the code is located (class and method). Specify nontrivial methods with pre- and postconditions (and examples if this clarifies).

In HealthMetricsReader class:

**package** PHI;

**import** java.io.BufferedReader;

**import** java.io.FileReader;

**import** java.io.IOException;

**public** **class** HealthMetricsReader {

/\*\*

\* Reads the ideal values for a given metric from the HealthMetrics file.

\*

\* **@param** metric the metric for which to retrieve the ideal values

\* **@return** an array containing the ideal values [lowerBound, upperBound]

\* **@precondition** the HealthMetrics file exists and is accessible

\* **@postcondition** the returned array contains the ideal values for the given metric,

\* or [0.0, 0.0] if the metric is not found in the file

\*/

**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\\Hw2\\HealthMetrics.txt"))) {

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**;

}

}

} **catch** (IOException e) {

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

}

**return** idealValues;

}

}

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

1. PARAGRAPH DESCRIBING YOUR VALUE ADDED TO THE AI-GENERATED MATERIAL

ChatGPT response saved me a lot of time but I had to verify the ideal metrics and found out that the Systolic Blood Pressure range is ideal if it is 80-120 instead of 90-120. Also, I made chatgpt to find out the metric Blood sugar for me to determine diabetes. I also found out the normal range instead of adding separate rates depending on if the user is fasting or not. I removed the comments in the text file to make sure the app is not reading wrong or unnecessary information from the file. I removed age, weight, and height as I don’t think I need to put them in a file to have ideal values of these metrics so I changed the file.  
Lastly, I moved the file reading to a class to make it more readable. I also had to correct the file path as it wasn’t correct. Idealvalue needed to have two values, min and max so I made it an array of double and made changes later.

1. MY PROMPT SEQUENCE (ONLY)

[1] Here are the metrics I have in mind,

private int systolicBP;

private int diastolicBP;

private double bmi;

private int age;

private double weight;

private int ldlCholesterol;

private int hdlCholesterol;

private int triglycerideCholesterol;

private boolean diabetes;

So I need ideal values for them in a file. Also, for diabetes, how and what values do I need to compare against?

[2] Instead of diabetes Boolean, I need another metric that I can use to determine diabetes.

[3] Write me java code that reads this HealthMetrics file and extract three pieces of information: name, min, and max values.

1. CODE OR TEXT PRODUCED BY THE ABOVE PROMPT SEQUENCE

chatGPT:

[1] To determine the ideal values for the health metrics you mentioned, it's important to consult reputable sources such as medical guidelines or healthcare professionals. Ideal values may vary depending on factors such as age, gender, and individual health conditions. Here is a general guideline for the ideal values of the metrics you mentioned:

Systolic Blood Pressure (systolicBP): The ideal range is typically considered to be around 90-120 mmHg.

Diastolic Blood Pressure (diastolicBP): The ideal range is typically considered to be around 60-80 mmHg.

BMI (bmi): The ideal range is generally between 18.5 and 24.9.

Age (age): There is no specific ideal range for age as it varies depending on the individual's context.

Weight (weight): The ideal weight can vary based on factors such as height, body composition, and overall health. It is best to consult a healthcare professional for personalized guidance.

LDL Cholesterol (ldlCholesterol): The ideal range is generally below 100 mg/dL.

HDL Cholesterol (hdlCholesterol): The ideal range is generally above 40 mg/dL for men and above 50 mg/dL for women.

Triglyceride Cholesterol (triglycerideCholesterol): The ideal range is typically below 150 mg/dL.

Diabetes (diabetes): For diabetes, you can compare the user's status (true or false) against the ideal value of "false" since having diabetes is considered outside the ideal range.

Please note that these values are general guidelines, and it is crucial to consult with healthcare professionals or reputable sources to determine specific ideal ranges based on individual circumstances and medical guidelines.

Once you have obtained the ideal values for these metrics, you can store them in a file following a similar format as mentioned earlier:

bash

Copy code

# Ideal Health Metrics

# Format: metric\_name,ideal\_min,ideal\_max

Systolic Blood Pressure,90,120

Diastolic Blood Pressure,60,80

BMI,18.5,24.9

Weight,0,0 # Add specific values based on individual circumstances

LDL Cholesterol,0,100

HDL Cholesterol,40,0 # Add separate values for men and women

Triglyceride Cholesterol,0,150

Diabetes,false,false

Remember to update the values based on reliable sources and healthcare recommendations.

[2] Blood Glucose Level (glucoseLevel): This metric represents the user's blood glucose level. You can compare the user's input for this metric against specific thresholds (e.g., fasting blood glucose level of 126 mg/dL or higher) to determine if the user is diabetic. If the user's blood glucose level exceeds the threshold, you would set the diabetes metric to true, indicating the presence of diabetes.

[3]

private static double readIdealValue(String metric) {

double idealValue = 0.0;

try (BufferedReader reader = new BufferedReader(new FileReader("HealthMetrics.txt"))) {

String line;

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

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

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

idealValue = Double.parseDouble(values[1]);

break;

}

}

} catch (IOException e) {

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

}

return idealValue;

}