5/18/2023

CS 622 Draft Assignment 3

Please submit a draft of Assignment 3, using the Word template supplied, with *only* the following parts drafted:

2.1

6.1

6.2: show a start on this

### 2.1 Health Information Management (old-modified from week 1)

The application must enable users to input, edit, and manage their personal health information, including BMI, blood pressure, cholesterol levels, and custom health conditions, utilizing Java generics.

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

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

I had to clarify the AI of what java material I am implementing this week and asked for some new requirements related to Java Generics for my PHI project. I modified the response by changing the “should be able to” to “must” as I remembered this is essential part of functional requirements. I realized from week 1 I had a requirement that can be combined here and so I modified it and added the java generics to it. This way I reduced one extra requirement that can be written in the existing requirement. I also simplified the requirement by getting rid of extra sentences that is not necessary to mention in the requirement line.

1. MY PROMPT SEQUENCE (ONLY)

[1]

“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 which are new vs. old.”

This is the next task. Do you think I need new functional requirements this week or use my old requirements which would be relevant in this week's iteration?

[2] Remember, this week's iteration is about Java generic.

1. CODE OR TEXT PRODUCED BY THE ABOVE PROMPT SEQUENCE

chatGPT X

Since you have already provided a set of functional requirements for the previous iterations, it would be beneficial to incorporate new requirements in order to expand the functionality of your app and demonstrate your progress. This will not only showcase your ability to introduce new features but also ensure that you are making consistent improvements to your project.

Considering the nature of your app, here are two additional requirements (features) that you can consider implementing in this release:

3.9 Health Data Visualization

The application should provide visual representations, such as graphs or charts, of the user's health data over time, allowing them to easily track their progress and identify trends.

3.10 Reminders and Notifications

The application should send reminders and notifications to users for important health-related activities, such as medication schedules, upcoming appointments, or recommended screenings, helping them stay on top of their health routines.

By incorporating these new requirements, you can enhance the functionality and user experience of your app. However, feel free to adjust or modify the requirements based on the specific goals and scope of your project.

[2] Apologies for the oversight. Given that this week's iteration focuses on implementing Java generics, it would be appropriate to incorporate requirements that specifically pertain to this feature. Here are two additional requirements related to Java generics that you can consider implementing:

3.9 Generic HealthData Class

The application should be able to implement a generic HealthData class that can store and manage health-related information of various types, allowing for flexibility in handling different health metrics.

3.10 Generic HealthData Validation

The application should be able to implement a generic validation mechanism for the HealthData class that can validate input values based on the specific data type, ensuring that only valid health data is accepted.

By incorporating these requirements, you will be able to leverage the power of Java generics to create a more flexible and reusable design for managing health data in your app.

### 6.2 Code (including test code), input (if applicable), and output showing generics

Explain why the use of *generics* is appropriate here.

Here is the Main class where I am demonstrating the PHI project:  
**package** PHI;

**import** java.util.Date;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

/\*\*

\* The main method to demonstrate the functionality of the health data tracking system.

\*

\* Pre-condition:

\* - Appropriate classes with constructors exist

\* - Custom validation exception handling method exists

\*

\* Post-condition:

\* - Prints the health data for a user, including blood pressure, cholesterol levels, blood glucose level, BMI, and custom health data.

\* - Validates each health data entry before adding it to the user's health data list.

\* - Handles and prints any validation errors that occur during the process.

\*/

**try** {

// Create a user with some health data

User<HealthData<?>> user = **new** User<>("John", "Doe", "johndoe@example.com", "Passw0rd", **new** Date(), "Male", "555-1234");

// Add common health data

HealthData<?> commonHealth1 = **new** CommonHealthData(user.getFullName(), **new** Date(), "Blood Pressure", 120, 60);

((CommonHealthData) commonHealth1).validate();

user.addHealthData(commonHealth1); // Add the health data before validation

HealthData<?> commonHealth2 = **new** CommonHealthData(user.getFullName(), **new** Date(), "Cholesterol", 120, 60, 140);

((CommonHealthData) commonHealth2).validate();

user.addHealthData(commonHealth2);

HealthData<?> commonHealth3 = **new** CommonHealthData(user.getFullName(), **new** Date(), "Blood Glucose", 100);

((CommonHealthData) commonHealth3).validate();

user.addHealthData(commonHealth3);

HealthData<?> commonHealth4 = **new** CommonHealthData(user.getFullName(), **new** Date(), "BMI", 145.0, 65.0);

((CommonHealthData) commonHealth4).validate();

user.addHealthData(commonHealth4);

// Add custom health data

HealthData<?> customHealth = **new** CustomHealthData("Ankle pain", **new** Date(), "ankle pain on 5-12-2023");

user.addHealthData(customHealth);

// Print out John Doe's health data

System.***out***.println(user.getFullName() + "'s health data:");

**for** (HealthData<?> healthData : user.getHealthDataList()) {

System.***out***.println(healthData.getMetric() + ": Recorded at: " + healthData.getDate());

System.***out***.println("Metric: " + healthData.getMetric());

**if** (healthData **instanceof** CommonHealthData) {

CommonHealthData commonHealthData = (CommonHealthData) healthData;

// Check if the health data is related to blood pressure

**if** (commonHealthData.getMetric().equals("Blood Pressure")) {

System.***out***.println("Systolic BP: " + commonHealthData.getSystolicBP());

System.***out***.println("Diastolic BP: " + commonHealthData.getDiastolicBP());

HealthDataChecker.*checkBloodPressure*(commonHealthData);

}

// Check if the health data is related to cholesterol levels

**if** (commonHealthData.getMetric().equals("Cholesterol")) {

System.***out***.println("ldl: " + commonHealthData.getLdlCholesterol());

System.***out***.println("hdl: " + commonHealthData.getHdlCholesterol());

System.***out***.println("tri: " + commonHealthData.getTriglycerideCholesterol());

HealthDataChecker.*checkCholesterol*(commonHealthData);

}

// Check if the health data is related to blood glucose level

**if** (commonHealthData.getMetric().equals("Blood Glucose")) {

System.***out***.println("blood sugar: " + commonHealthData.getGlucoseLevel());

HealthDataChecker.*checkBloodGlucose*(commonHealthData);

}

// Check if the health data is related to BMI

**if** (commonHealthData.getMetric().equals("BMI")) {

System.***out***.println("height: " + commonHealthData.getHeight());

System.***out***.println("weight: " + commonHealthData.getWeight());

System.***out***.println("bmi: " + commonHealthData.calculateBMI());

HealthDataChecker.*checkBMI*(commonHealthData);

}

} **else** **if** (healthData **instanceof** CustomHealthData) {

CustomHealthData customHealthData = (CustomHealthData) healthData;

System.***out***.println("Notes: " + customHealthData.getNotes());

}

System.***out***.println();

}

} **catch** (HealthDataException e) {

System.***out***.println("Health data validation error: " + e.getMessage());

}

}

}

Here is the HealthData class:

**package** PHI;

**import** java.util.Date;

**public** **abstract** **class** HealthData<T> {

**private** String name;

**private** Date date;

**public** HealthData(String name, Date date) {

**this**.name = name;

**this**.date = date;

}

**public** String getName() {

**return** name;

}

**public** Date getDate() {

**return** date;

}

**public** **abstract** String getMetric();

**public** **abstract** T getData();

}

And lastly, here is the User class:  
**package** PHI;

**import** java.util.ArrayList;

**import** java.util.Date;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** User<T **extends** HealthData<?>> {

**private** String firstName;

**private** String lastName;

**private** String email;

**private** String password;

**private** Date dateOfBirth;

**private** String gender;

**private** String phoneNumber;

**private** ArrayList<T> healthDataList;

**public** User(String firstName, String lastName, String email, String password, Date dateOfBirth, String gender, String phoneNumber) {

**if** (!isValidEmail(email)) {

**throw** **new** IllegalArgumentException("Invalid email address");

}

**if** (!isValidPassword(password)) {

**throw** **new** IllegalArgumentException("Invalid password");

}

**if** (dateOfBirth.after(**new** Date())) {

**throw** **new** IllegalArgumentException("Invalid date of birth");

}

**this**.firstName = firstName;

**this**.lastName = lastName;

**this**.email = email;

**this**.password = password;

**this**.dateOfBirth = dateOfBirth;

**this**.gender = gender;

**this**.phoneNumber = phoneNumber;

**this**.healthDataList = **new** ArrayList<>();

}

**public** **void** addHealthData(T healthData) {

healthDataList.add(healthData);

}

**public** **void** removeHealthData(T healthData) {

healthDataList.remove(healthData);

}

**public** ArrayList<T> getHealthDataList() {

**return** healthDataList;

}

**public** String getFullName() {

**return** firstName + " " + lastName;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**if** (!isValidEmail(email)) {

**throw** **new** IllegalArgumentException("Invalid email address");

}

**this**.email = email;

}

**public** String getPhoneNumber() {

**return** phoneNumber;

}

**public** **void** setPhoneNumber(String phoneNumber) {

**this**.phoneNumber = phoneNumber;

}

**private** **boolean** isValidEmail(String email) {

String emailRegex = "^[a-zA-Z0-9\_+&\*-]+(?:\\." +

"[a-zA-Z0-9\_+&\*-]+)\*@" +

"(?:[a-zA-Z0-9-]+\\.)+[a-z" +

"A-Z]{2,7}$";

Pattern pattern = Pattern.*compile*(emailRegex);

Matcher matcher = pattern.matcher(email);

**return** matcher.matches();

}

**private** **boolean** isValidPassword(String password) {

// Password must have at least 8 characters, one uppercase letter, one lowercase letter, and one digit

String passwordRegex = "^(?=.\*[0-9])(?=.\*[a-z])(?=.\*[A-Z]).{8,}$";

Pattern pattern = Pattern.*compile*(passwordRegex);

Matcher matcher = pattern.matcher(password);

**return** matcher.matches();

}

}

I think the use of generics increased the robustness, reusability, and flexibility of the PHI app. The User class uses an arraylist of type HealthData which is accepts generics type. Meaning, the User class can accept different kind of HealthData and doesn’t require separate classes for various type of health data. It also ensures the reusability by using the same generic type arraylist holder for each user to hold any kind of health data. With generics, the healthdatalist in the User class can accept any new type of HealthData that can be implemented in the future, i.e ExerciseData along with common and custom health data.   
The code in the Main class effectively demonstrates the usage of generics by utilizing HealthData<?>. It is used in the following code snippet,  
User<HealthData<?>> user = new User<>("John", "Doe", "johndoe@example.com", "Passw0rd", new Date(), "Male", "555-1234");  
The User can now accept any HealthData type. Also in the for loop below,  
for (HealthData<?> healthData : user.getHealthDataList()) {…  
the Usage of HealthData<?> type demonstrate the specific methods and functionalities of any type of healthdata can be accessed proving the flexibility of generics type.