# Assignment 3 5/18/2023-1

Implement the next release of your term project. You will incorporate Java generics, as specified below.

Implement the next release of your term project (preferably—start a new one if you have to). The same instructions as in Assignment 2 apply to this completed Word document, the gray text, the AI-generation, the 5 page limit, appendices, JUnit tests, and a ReadMe file.

## 1 SUMMARY DESCRIPTION, UPDATED AS NECESSARY

One- or two-paragraph overall description of your proposed term project. Color in red the parts different from Assignment 2.

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.

## 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 which are new vs. old.

### 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.

### 2.2 Generic HealthData Validation (New)

The application must implement a generic validation mechanism for the HealthData or it’s subclasses that can validate input values based on the specific data type.

### >>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.

## 3 I/O EVIDENCE OF ACCOMPLISHING THE REQUIREMENTS LISTED ABOVE

Provide an example of actual input / output corresponding to the requirements above

Input:

In the Main Class, the following code snippet demonstrate one example of editing capability.

A picture containing text, screenshot, font

Description automatically generated

### Output:

A screenshot of a computer

Description automatically generated with medium confidence

Input:  
The User class snippet:

A screenshot of a computer code

Description automatically generated with low confidence  
The HealthData Class:

A screenshot of a computer code

Description automatically generated with low confidence  
Main class snipptets:  
A picture containing text, font, line, screenshot

Description automatically generated  
A picture containing text, screenshot, line, font

Description automatically generated  
  
Output:  
A screenshot of a computer

Description automatically generated

Input:   
In the CommonHealthData, a subclass of HealthData generic class, here is the validate() method  
A screenshot of a computer code

Description automatically generated with medium confidence  
Output:  
A screenshot of a computer

Description automatically generated with medium confidence  
  
As you can see there is one example of the app showing it’s ability to let the user change some healthData. Next the example show the usage of generic type in the healtDAta and User class. The Main class shows the advantage of using generics. It shows the User can take any healthData type denoted by HealthData<?> and store it. It also shows the generics are working properly by printing out various type of healthData information. Regardless of type, because of generics, any type of healthData information was accessible from the healthDataarraylist which takes healthData<?> data type. Lastly, If I put a negative value for any of the commonhealthdata metric, the validate method will catch it. It shows even though the healthdata was generic, the commonhealthdata class which extends HealthData, was able to utilize its validated method to catch any invalid input.

## 4 YOUR DIRECTORY

Show a screenshot of your directory. This should include a parallel directory of JUnit tests where applicable—package-by-package, class-by-class, and method-by-method, except for trivial and inapplicable ones.

A screenshot of a computer

Description automatically generated with medium confidence

## 5 YOUR UPDATED CLASS MODEL AND CLARIFICATION OF HOW THE EXECUTION WORKS

Supply a main use case, the class model, and the sequence diagram corresponding to the use case. These should be consistent and clear. Indicate clearly in your class model where you applied generics. To do this use tools, PowerPoint, or a combine models as in [this example](https://docs.google.com/spreadsheets/d/1vBmDVtWWh3EX0oehFFLRU0P6eR-fn4d0qVg1-XOUooM/edit?usp=sharing) (which you are free to cut and paste from). Insert indications in red to show where generics apply.

Class diagram:  
A screenshot of a computer

Description automatically generated with medium confidence  
The User is a concrete generic class with <T extends HeatlhData<?>> has a relationship with HealthData which is shown using the multiplicity and composition. HealthData is an abstract class thus written in italic and is a generic class. The inheritance is also shown in the diagram using the open arrow.   
  
Sequence diagram:  
A screenshot of a computer

Description automatically generated with medium confidence  
  
I have kept the sequence diagram more simple than the last week’s version and it is now much more easier to follow.

## 6 WHERE GENERICS ARE IMPLEMENTED

### 6.1 Class model fragment showing generic class

Explain where and how you applied *generic classes* in your class model.

Generics are applied in two classes of the class diagram: HealthData<T> and User<T extends HealthData<?>>.

HealthData<T> class:

The HealthData<T> class is a generic class that represents health data. The type parameter T is used to indicate the type of data associated with the health metric.

The class has a constructor HealthData(name: String, date: Date) that takes the name and date of the health data.

It provides methods getName(): String and getDate(): Date to retrieve the name and date of the health data.

The class also declares abstract methods getMetric(): String and getData(): T that must be implemented by its subclasses.

By using generics, the HealthData<T> class can be extended to handle different types of health data while ensuring type safety.

User<T extends HealthData<?>> class:

The User<T extends HealthData<?>> class represents a user who has associated health data. It is a generic class where the type parameter T is bounded by HealthData<?>, indicating that it can accept any subclass of HealthData.

The class has attributes to store user information such as firstName, lastName, email, password, dateOfBirth, gender, and phoneNumber.

It also includes an ArrayList<T> called healthDataList to store the user's health data.

The class provides methods addHealthData(healthData: T), removeHealthData(healthData: T), editHealthData(index: int, newHealthData: T), and getHealthDataList(): ArrayList<T>, which all operate on the generic type parameter T.

By utilizing generics, the User class allows the association of different types of health data with a user, maintaining type safety and enabling the user to work with various health metrics.

### >>>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

In general, I liked the format chatGPT used to answer my prompt. However, I edited the response and added more essential detail to clarify where and how the generics were used in the PHI app. You can read my answer above and compare it against the AI generated response.

1. MY PROMPT SEQUENCE (ONLY)

[1]

6.1 Class model fragment showing generic class

Explain where and how you applied generic classes in your class model.

Here is my current UML:

@startuml

' Abstract class

abstract class HealthData<T> {

- name: String

- date: Date

+ HealthData(name: String, date: Date)

+ getName(): String

+ getDate(): Date

{abstract} +getMetric(): String

{abstract} +getData(): T

}

' Concrete class implementing HealthData

class CommonHealthData {

+systolicBP: int

+diastolicBP: int

+ldlCholesterol: int

+hdlCholesterol: int

+triglycerideCholesterol: int

+glucoseLevel: int

+height: double

+weight: double

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

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

+CommonHealthData(date: Date, metric: String, glucoseLevel: int)

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

+validate(): void

+getSystolicBP(): int

+getDiastolicBP(): int

+getLdlCholesterol(): int

+getHdlCholesterol(): int

+getTriglycerideCholesterol(): int

+getGlucoseLevel(): int

+getHeight(): double

+getWeight(): double

+calculateBMI(): double

}

' Concrete class extending HealthData

class CustomHealthData {

+notes: List<String>

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

+addNote(note: String): void

+getNotes(): List<String>

+getMetric(): String

+getData(): String

}

' Class representing HealthMetricsReader

class HealthMetricsReader {

{static} + readIdealValue(metric: String): double[]

}

' Class representing User

class User<T extends HealthData<?>> {

- firstName: String

- lastName: String

- email: String

- password: String

- dateOfBirth: Date

- gender: String

- phoneNumber: String

- healthDataList: ArrayList<T>

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

+ addHealthData(healthData: T): void

+ removeHealthData(healthData: T): void

+ editHealthData(index: int, newHealthData: T): void

+ getHealthDataList(): ArrayList<T>

+ getFullName(): String

+ getEmail(): String

+ setEmail(email: String): void

+ getPhoneNumber(): String

+ setPhoneNumber(phoneNumber: String): void

}

' Exception class

class HealthDataException {

- message: String

+ HealthDataException(message: String)

+ getMessage(): String

}

' Association between User and HealthData (aggregation relationship)

User "1" \*-- "0..\*" HealthData: healthDataList

' Inheritance between HealthData and CommonHealthData

HealthData <|-- CommonHealthData

' Inheritance between HealthData and CustomHealthData

HealthData <|-- CustomHealthData

' Association between CommonHealthData and HealthDataException

CommonHealthData "1" -- "0..\*" HealthDataException: throws

' Association between HealthMetricsReader and HealthDataException

HealthMetricsReader --> HealthDataException

' Class representing HealthDataChecker

class HealthDataChecker {

{static} +checkBloodPressure(common: CommonHealthData): void

{static} +checkCholesterol(common: CommonHealthData): void

{static} +checkBloodGlucose(common: CommonHealthData): void

{static} +checkBMI(common: CommonHealthData): void

}

' Association between HealthDataChecker and HealthMetricsReader

HealthDataChecker --> HealthMetricsReader

@enduml

1. CODE OR TEXT PRODUCED BY THE ABOVE PROMPT SEQUENCE

chatGPT X

In your current UML diagram, you have applied generic classes in the HealthData and User classes. Here's how they are applied:

HealthData<T> class:

The HealthData<T> class is a generic class with a type parameter T. This allows the HealthData class to be flexible and work with different types of data.

The HealthData<T> class has a generic type parameter T which represents the type of data associated with the health metric.

The getData() method is defined as T getData() to return the data of type T.

User<T extends HealthData<?>> class:

The User<T extends HealthData<?>> class is also a generic class with a type parameter T that extends the HealthData class.

The type parameter T restricts the type of health data that can be associated with a user.

The healthDataList attribute is defined as ArrayList<T> to store a list of health data of type T.

The methods addHealthData, removeHealthData, editHealthData, and getHealthDataList all use the generic type parameter T to ensure type safety when working with the user's health data.

By using generic classes, you provide flexibility and type safety in working with different types of health data for the HealthData and User classes.

### 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.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**import** java.util.Scanner;

**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

// Create a Scanner object to read input from the terminal

Scanner scanner = **new** Scanner(System.***in***);

// Prompt the user to enter their information

System.***out***.print("Enter your first name: ");

String firstName = scanner.nextLine();

System.***out***.print("Enter your last name: ");

String lastName = scanner.nextLine();

System.***out***.print("Enter your email: ");

String email = scanner.nextLine();

System.***out***.print("Enter your password: ");

String password = scanner.nextLine();

System.***out***.print("Enter your date of birth (yyyy-mm-dd): ");

String dobString = scanner.nextLine();

Date dob = **new** SimpleDateFormat("yyyy-MM-dd").parse(dobString);

System.***out***.print("Enter your gender: ");

String gender = scanner.nextLine();

System.***out***.print("Enter your phone number: ");

String phoneNumber = scanner.nextLine();

// Create a user with the provided information

User<HealthData<?>> user = **new** User<>(firstName, lastName, email, password, dob, gender, phoneNumber);

// 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 (original)

System.***out***.println("Original health data:");

*printHealthData*(user);

// Edit health data

**int** indexToEdit = 1; // Index of the health data entry to edit

HealthData<?> newHealthData = **new** CommonHealthData(user.getFullName(), **new** Date(), "Cholesterol", 130, 70, 300);

((CommonHealthData) newHealthData).validate();

user.editHealthData(indexToEdit, newHealthData);

System.***out***.println("Updated health data:");

// Print out John Doe's health data (updated)

*printHealthData*(user);

} **catch** (HealthDataException e) {

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

} **catch** (ParseException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**private** **static** **void** printHealthData(User<HealthData<?>> user) {

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();

}

}

}

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** **void** editHealthData(**int** index, T newHealthData) {

**if** (index < 0 || index >= healthDataList.size()) {

**throw** **new** IllegalArgumentException("Invalid index for editing health data");

}

HealthData<?> existingData = healthDataList.get(index);

String existingMetric = existingData.getMetric();

String newMetric = newHealthData.getMetric();

**if** (!existingMetric.equals(newMetric)) {

**throw** **new** IllegalArgumentException("Cannot edit health data at index " + index + " with a different metric type");

}

System.***out***.println("Editing health data at index " + index + " for " + existingMetric);

healthDataList.set(index, newHealthData);

}

**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.

## 7 YOUR CODE

Unless your facilitator arranges another method, copy your Eclipse project to your file system, zip it, and attach it. Please contact your facilitator in advance if you want to request another transmission process (e.g., github).

  
Github: https://github.com/mahimcbu/CS622/tree/main/Assignments/Hw3/phi

### >>>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 utilized the response in the appropriate places and made sure of changing the type of Health Data to T everywhere the AI forgot to mention. In the Main class, I needed to change the code so it can work with generics. The AI forgot to add the validate() method before adding healthDAta to the arraylist. I added that portion of validation in the provided Main class by the chatGPT. It ensured the try catch block can catch the HealhtDataException user defined exception.

1. MY PROMPT SEQUENCE (ONLY)

[1] I would like to incorporate Java generics on User and HelathData classes of my PHI project. How should I approach this?

[2] As you can see the main class is using User, HealthData class here. After I changed them by introducing generics, change the main class so it doesn't error out such as type mismatch, etc.

1. CODE OR TEXT PRODUCED BY THE ABOVE PROMPT SEQUENCE

chatGPT X

[1] To incorporate generics into the User class and the HealthDataList ArrayList, you can make the following modifications:

User Class: Make the User class generic by specifying the type parameter for the HealthData stored in the HealthDataList ArrayList. Here's an example:

java

Copy code

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;

// Constructor, getters, setters, and other methods

}

HealthData Class: Update the HealthData class to also use generics for the metric and data types. Here's an example:

java

Copy code

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();

}

Update references: Update the references to the HealthData class throughout your code to use the new generic type parameter. For example, in the User class:

java

Copy code

private ArrayList<T> healthDataList;

With these modifications, you can use generics to specify the type of HealthData stored in the HealthDataList ArrayList of the User class, providing type safety and flexibility in handling different types of health data.

[2] 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);

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

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

user.addHealthData(commonHealth2);

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

user.addHealthData(commonHealth3);

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

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());

}

}

}

## 8 INSTRUCTOR’S EVALUATION

