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.

## 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.

## 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.