TRAININGSAPP

Sophia Ladwig sl153
Franziska Link fl056
Yuting Xia yx006

Projektname: trainingsapp2

https://gitlab.mi.hdm-stuttgart.de/fl056/trainingsapp2.git

Stuttgart - 05.02.2023

TRAININGSAPP

1. Kurzbeschreibung

Unser Projekt handelt von einer Trainingsapp für das zweite Semester der Studiengänge Medieninformatik und Mobile Medien an der Hochschule der Medien.
Wenn man das Programm startet gelangt man zur Startseite, welche den Übungsplan für den jeweiligen Tag anzeigt. Im Hintergrund läuft Musik, welche den Nutzer zum Training motivieren soll. Man kann per Buttonclick in der Menüleiste einen anderen Tag auswählen und der Trainingsplan für den jeweiligen Tag wird geladen. Wenn man eine Übung hinzufügen möchte kann man auf das Plus drücken und kommt dann auf die Übersicht der Übungskategorien. Dort kann der User eine Kategorie auswählen und gelangt danach auf die Seite mit einer Liste an Übungen der jeweiligen Kategorie. Möchte er eine eigene Übung erstellen kann er auf das Plus drücken und kann dort theoretisch eine neue Übung anlegen.

In der Menüleiste kann der User auswählen, ob er sich den Trainingsplan anzeigen lassen möchte oder sein Profil ansehen möchte. Wenn man rechts auf das Profilicon drückt kommt man auf die Übersicht der Profildaten. Anfangs sind dort Defaultwerte eingepeichert. Beim Drücken der drei Punkte kann man das Profil bearbeiten und der Save-Button speichert die Daten und man gelangt automatisch auf die Profil-Übersicht mit den neuen Daten.

2. Startklasse

Die Main-Methode befindet sich in der Klasse HelloApplication.

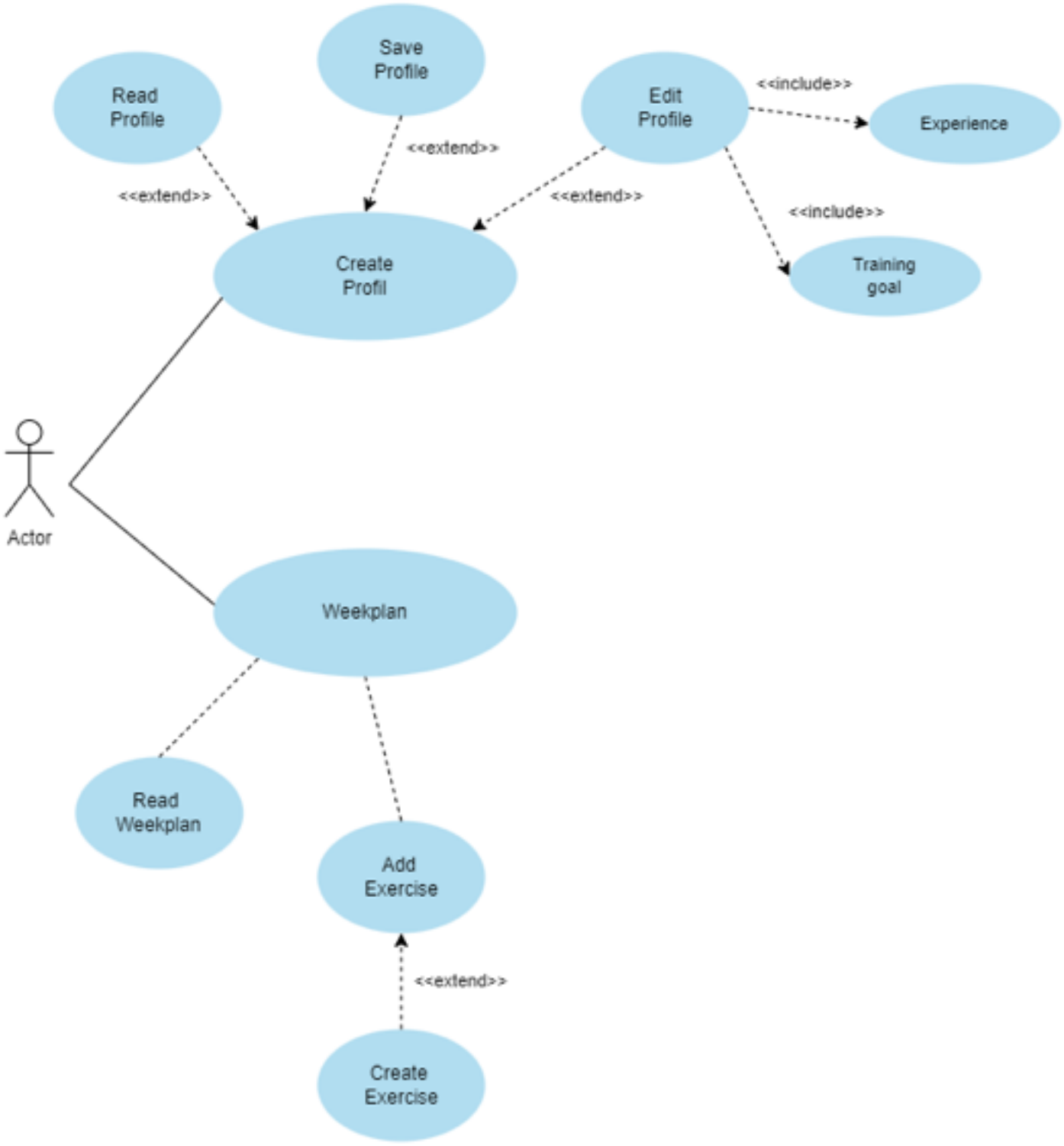
3. Besonderheiten

Da dieses Projekt für uns alle eine neue Erfahrung war, da wir noch keine Vorkenntnisse, außer SE1 hatten, wussten wir anfangs zunächst nicht wie wir an die Sache ran gehen sollten. Wir würden jetzt im Nachhinein die Dinge etwas anders machen. Beispielsweise würde wir im nächsten Projekt zunächst mit der GUI anfangen und daraufhin die Logik und die Controller anpassen.

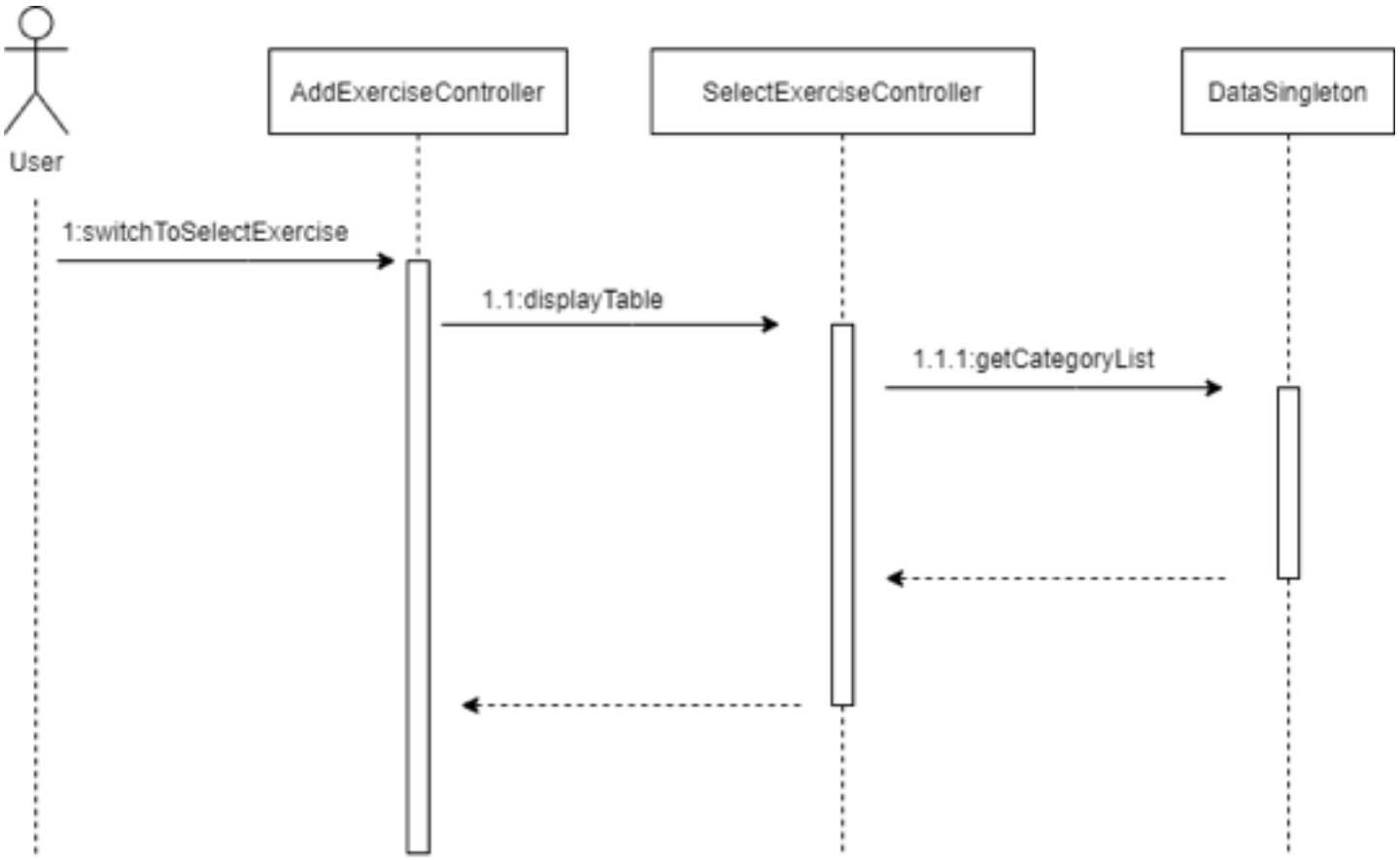
Im Moment funktioniert es noch nicht, dass man das Alter im Profil verändern kann. Außerdem kann man leider noch keine eigenen Übungen zu den Kategorielisten und auch nicht zu den Trainingsplänen hinzufügen. Sobald man von der Seite, auf welcher man die eigene Übung theoretisch erstellen kann wieder zurück geht, werden keine Übungen mehr angezeigt. Außerdem kann man im Profil noch kein Foto hinzufügen. Grund dafür ist, dass wir aus zeittechnischen Gründen noch nicht dazukamen die Controller und die Logik zu erweitern.

4. UML Klassendiagramm

Use Case Diagramm



Sequenz Diagramm



5.Stellungnahme

Architektur
1. Wir haben drei Packages erstellt:
- **Model**: beinhaltet alle Logikklassen
- **View**: beinhaltet die Startklasse
- **Controller**: beinhaltet alle Controller

Im Ordner Resources befinden sich die Ordner
- **fxml**: mit den FXML-Files
- **images**: mit den png-Files (Icons)
- **music**: mit dem wav-File

2. Die abstrakte Klasse **Exercise** erbt an die konkreten Klassen **Cardio**, **Strength** und **Stretching**. In Exercises sind die Felder **exName**, **exSet** und **exMuscle**, welche jede Übung hat. Strength-Übungen haben zusätzlich noch **exReps** (Anzahl an Wiederholungen) und Cardio- und Stretching-Übungen haben zusätzlich noch eine **exDuration** (Dauer der Übung). Wir konnten aus den beiden Klassen nicht eine Klasse machen, da die Klassen später in Kategorielisten eingeteilt werden.

Der abstrakte **BaseController** erbt an alle anderen Controller die **switchTo()**-Methode.

3. Wir benutzen viele Konstanten und haben deshalb Enums erstellt, die diese speichern:
- **Day**
- **ExerciseCategory**
- **Fitnesslevel**
- **Goal**
- **Muscle**
- **ProfileAttributes**
- **Sex**

4. Wir haben die Daten über die **DataSingleton**-Klasse gespeichert, damit alle alles Klassen auf die Daten zugreifen können und wir dafür nur das Objekt der DataSingleton-Klasse benötigen. Der Konstruktor ist private, weshalb andere Klassen keine neuen Instanzen davon bilden können. Ein Vorteil davon ist, dass man gegenüber einfach global sichtbaren Variablen eine gewisse Zugriffskontrolle sicherstellen kann und über das DataSingleton-Objekt data immer die gleichen Daten referenziert werden. Die Erzeugung des Objekts nach Bedarf kann außerdem Performance einsparen. Es ist somit eine einfache Möglichkeit einen globalen state (die Instanzvariablen) zu verwalten. Ein Problem bei der Singletonklasse ist, dass es fast nicht testbar ist.

Clean Code
1. Es werden nirgends public member verwendet. Außerdem werden die Objekte von einem Getter() Aufruf erst in ein neues Objekt kopiert, und dieses wird dann ausgegeben.

2. Dadurch, dass wir eine Singleton-Klasse implementiert haben wird das SOLID-Prinzip (konkret das Single Responsibility Principle) verletzt, da sich die Singleton-Klasse um alles selbst kümmert und nicht nur für eine Aufgabe zuständig ist.

Tests
Wir testen die Klassen Profile, DataSingleton und AddExerciseController.

GUI
Wir haben die FXML-Dateien mit dem SceneBuilder gestaltet, was es uns erleichtert hat die Elemente gut anzuordnen. Wir haben einen **BaseController** geschrieben, der eine switchTo()-Methode enthält und auf die id des jeweiligen UI-Elements zugreift. Das hat für uns in dem Moment Sinn gemacht, da wir sonst nicht wirklich wussten, wie wir eine neue Szene laden sollten ohne eine neue Methode für das jeweilige Element zu schreiben. Im Laufe der Entwicklung haben wir dann darauf aufgebaut und haben teilweise noch neue SwitchTo()-Methoden implementiert (Bsp.: im **AddExerciseController)**. Der Grund war, dass wir mit drei Buttons das gleiche FXML-File laden wollten, das hätte mit der ursprünglichen switchTo()-Methode nicht funktioniert, denn jeder Button braucht eine eigene Id. Das hatten wir am Anfang nicht auf dem Schirm.

Logging/Exceptions:
Exceptions werden mit dem Level Error geloggt. Die Loggings werden in das File LogTrainingsapp.log gespeichert.

Threads:
Ein zweiter Thread **BGM** lässt im Hintergund Musik laufen. Der Thread wird in der start()-Methode in HelloApplication gestartet.

Streams/ Lambda-Funktionen:
Wir haben in den Enums Fitnesslevel, Goal und Sex eine Methode getValueArr() implementiert, welche in einem Stream einen Array mit den Enumkonstanten beinhaltet. Die Enumkonstanten werden in einer Map auf ihre String-Werte abgebildet, welche wiederum in einem String-Array gespeichert werden.

Außerdem haben wir beispielsweise in der Klasse AddExerciseController in der Methode buttonClicked() Lambda-Expressions verwendet. Dies haben wir getan, da wir in dem Fall je nach Button der geclickt wird eine andere Category abspeichern. die

UML
Wir haben ein Klassendiagramm, ein Use-Case-Diagramm und Sequenzdiagramm.

VIEL SPASS BEIM AUSPROBIEREN!