# Fitness Application - Design Manual

## 1. Introduction

This fitness application provides a user friendly interface that encompasses all the essential functions of the fitness app. Moreover, to allow you to tailor your experience according to your needs, the fitness application allows you to use this app either as a registered user (through login or signup) or a guest. If you are just interested in using a calorie calculator and chatbot, you can simply use this application as a guest. However, if you are also interested in using a calendar to keep track of all your workout information and visualize your progress using a chart, you will need to use this application as a registered user by logging in or signing up if you do not already have your account. For a registered user, all your workout information will be saved using serialization so that you will still have access to it when you use this application in the future. Thus, with a robust combination of exercise options, data tracking, and interactive features, our FITNESS application provides a comprehensive solution for individuals looking to achieve their fitness goals in a motivating and sustainable manner.

## 2. User Stories

Among our clients is Amelia Nicolas, a dedicated gym trainer seeking a fitness app to help her clients accurately track progress remotely. We've implemented a calendar feature to record daily workouts, allowing users like Amelia to monitor their consistency and view past sessions.

Chloe Patel, a professional marathon runner, aims to track her daily morning runs and observe her progression over time. To meet diverse needs, we've introduced a tracker with a graph displaying monthly and yearly consistency, enabling users to visualize fluctuations within the week.

Elli Jutila, a fitness influencer, values consistency and a healthy lifestyle. Our app, equipped with a calorie calculator, chatbox for suggestions, and a journaling calendar, aligns perfectly with her goals of showcasing how consistency leads to a fabulous physique and a healthy life.

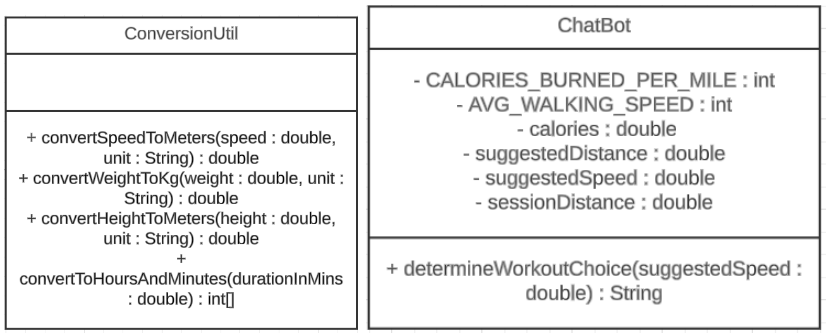
Finally, Heidelinde Gerke, a busy student at Bucknell University, desires an app that provides quick access without lengthy registration processes. We've created a guest user section allowing access to specific features, such as the calorie calculator and chatbox. Although guest users can't access the calendar or past workout charts due to not being registered, they still gain valuable insights into their workout and calorie data.

## 3. Object Oriented Design

Our Fitness Application is based on a FitnessApp MVC. The Fitness App controller mainly contains event handlers for the several features in the application. The model contains key methods that create functionality in the app while the view contains java FX features that aid in the visualization of the app features.

In our program, the Fitness App Controller handles the calculate button using both the calculate calories class and conversion Util class. As demonstrated in the UML diagram in Figure 1, the conversion util class converts the user's input to the required units of measurements and calculates calories according to the figures given by the conversion.

Additionally, the workout class is used by the controller to display information on the user's workouts in the calendar while the workout type enumeration class is used to specify the type of workouts displayed in the view. The ChatBot class is also used in the Controller when handling buttons for the Chat Box. The gender enumeration class is used in the controller by handling the creation of a new user account by specifying if the new user is a Male or Female.


ConversionUtil
+ convertSpeedToMeters(speed : double, unit : String) : double
+ convertWeightToKg(weight : double, unit : String) : double
+ convertHeightToMeters(height : double, unit : String) : double
+ convertToHoursAndMinutes(durationInMins : double) : int[]
ChatBot
- CALORIES_BURNED_PER_MILE : int
- AVG_WALKING_SPEED : int
- calories : double
- suggestedDistance : double
- suggestedSpeed : double
- sessionDistance : double
+ determineWorkoutChoice(suggestedSpeed : double) : String

The Fitness App model also uses the various classes to implement methods. The fitness model has an aggregation relationship with the user information class as the fitness model creates an instance of user information in the form of an array list . When a user logs in the model calls a verify method is called that uses the serialization class to get contents of the user information array list to verify if there is a valid username and password. When a user signs up, the model calls a serialize method that uses the serialization class to add the new user's username and password into the array list .

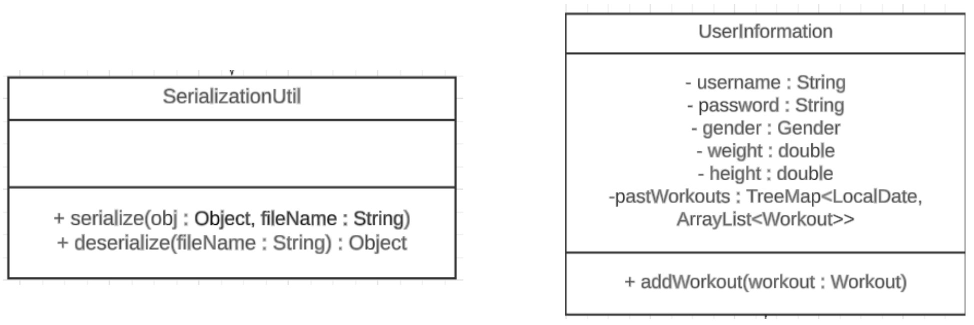
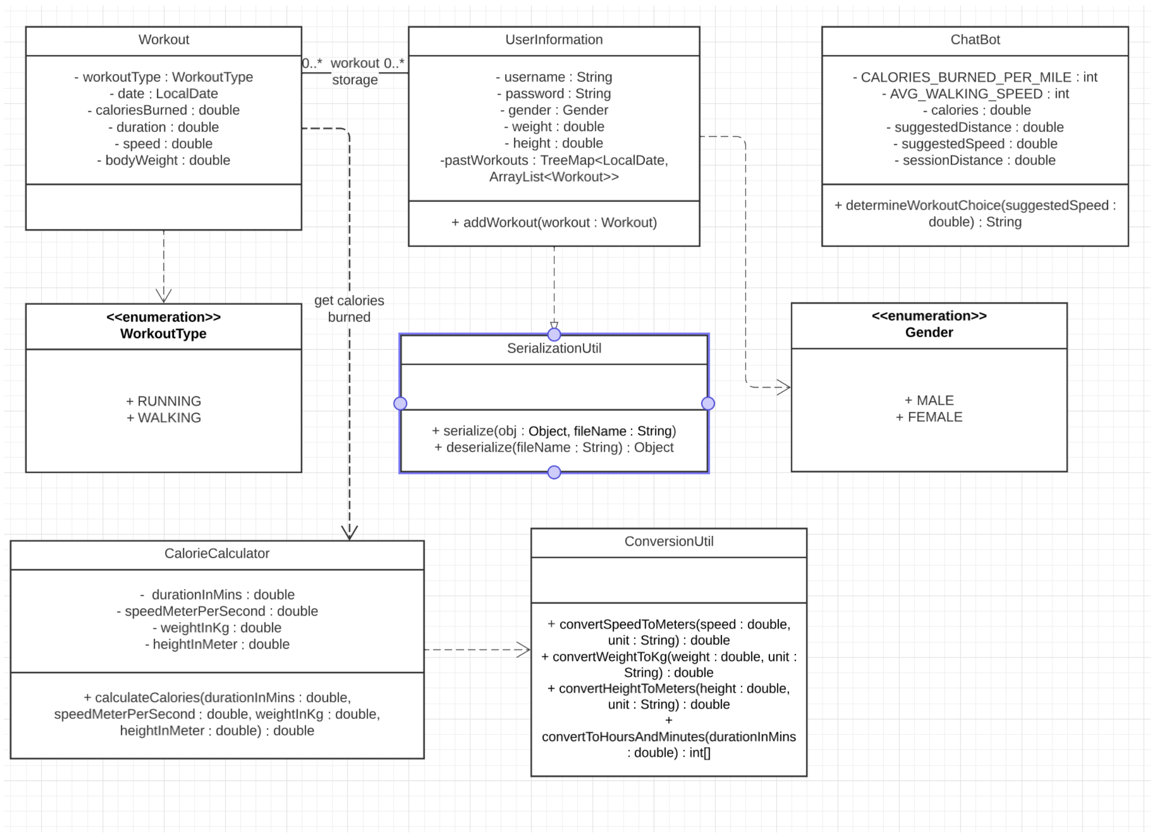
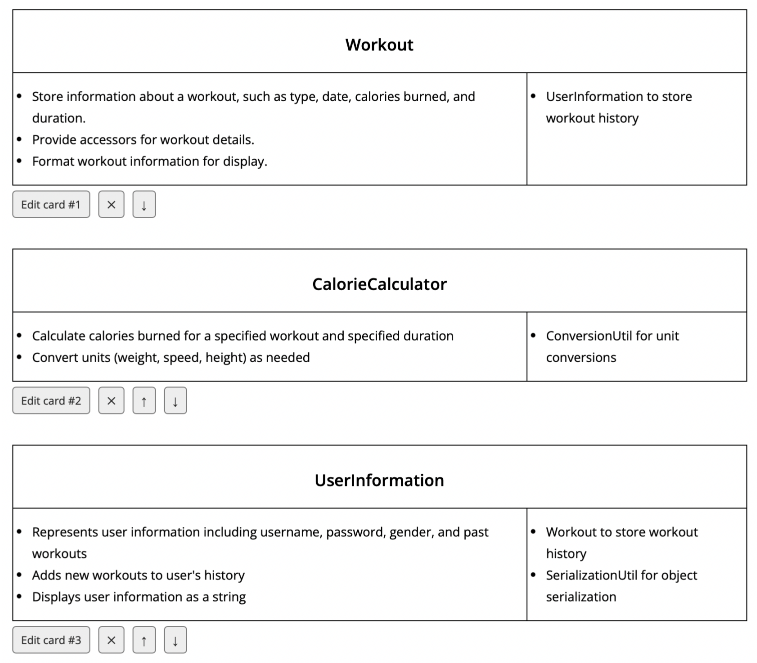

SerializationUtil
+ serialize(obj : Object, fileName : String)
+ deserialize(fileName : String) : Object
UserInformation
- username : String
- password : String
- gender : Gender
- weight : double
- height : double
-pastWorkouts : TreeMap<LocalDate, ArrayList<Workout>>
+ addWorkout(workout : Workout)

**Figure 1: UML diagram**

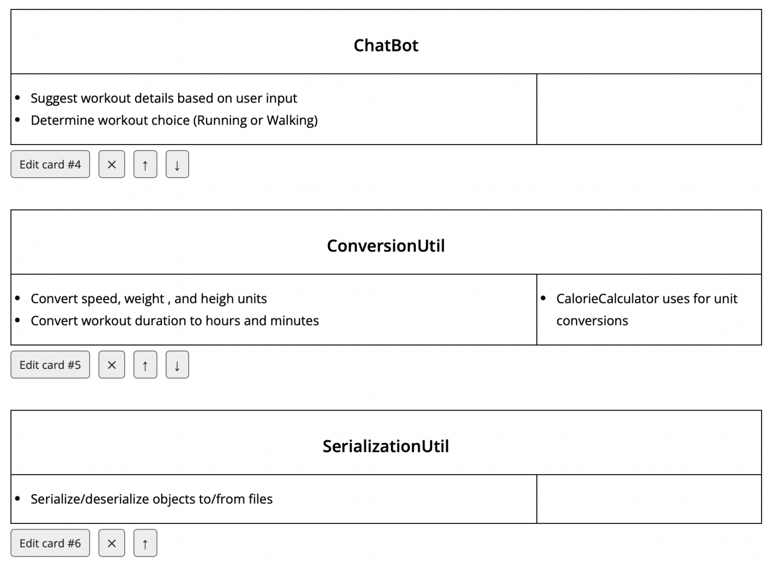

Workout
- workoutType : WorkoutType
- date : LocalDate
- caloriesBurned : double
- duration : double
- speed : double
- bodyWeight : double
0..*
workout storage
0..*
UserInformation
- username : String
- password : String
- gender : Gender
- weight : double
- height : double
-pastWorkouts : TreeMap<LocalDate, ArrayList<Workout>>
+ addWorkout(workout : Workout)
ChatBot
- CALORIES_BURNED_PER_MILE : int
- AVG_WALKING_SPEED : int
- calories : double
- suggestedDistance : double
- suggestedSpeed : double
- sessionDistance : double
+ determineWorkoutChoice(suggestedSpeed : double) : String
<<enumeration>>
WorkoutType
+ RUNNING
+ WALKING
get calories burned
SerializationUtil
+ serialize(obj : Object, fileName : String)
+ deserialize(fileName : String) : Object
<<enumeration>>
Gender
+ MALE
+ FEMALE
CalorieCalculator
- durationInMins : double
- speedMeterPerSecond : double
- weightInKg : double
- heightInMeter : double
+ calculateCalories(durationInMins : double, speedMeterPerSecond : double, weightInKg : double, heightInMeter : double) : double
ConversionUtil
+ convertSpeedToMeters(speed : double, unit : String) : double
+ convertWeightToKg(weight : double, unit : String) : double
+ convertHeightToMeters(height : double, unit : String) : double
+ convertToHoursAndMinutes(durationInMins : double) : int[]



| Workout | |
|---|---|
| • Store information about a workout, such as type, date, calories burned, and duration.<br>• Provide accessors for workout details.<br>• Format workout information for display. | • UserInformation to store workout history |

| CalorieCalculator | |
|---|---|
| • Calculate calories burned for a specified workout and specified duration<br>• Convert units (weight, speed, height) as needed | • ConversionUtil for unit conversions |

| UserInformation | |
|---|---|
| • Represents user information including username, password, gender, and past workouts<br>• Adds new workouts to user's history<br>• Displays user information as a string | • Workout to store workout history<br>• SerializationUtil for object serialization |



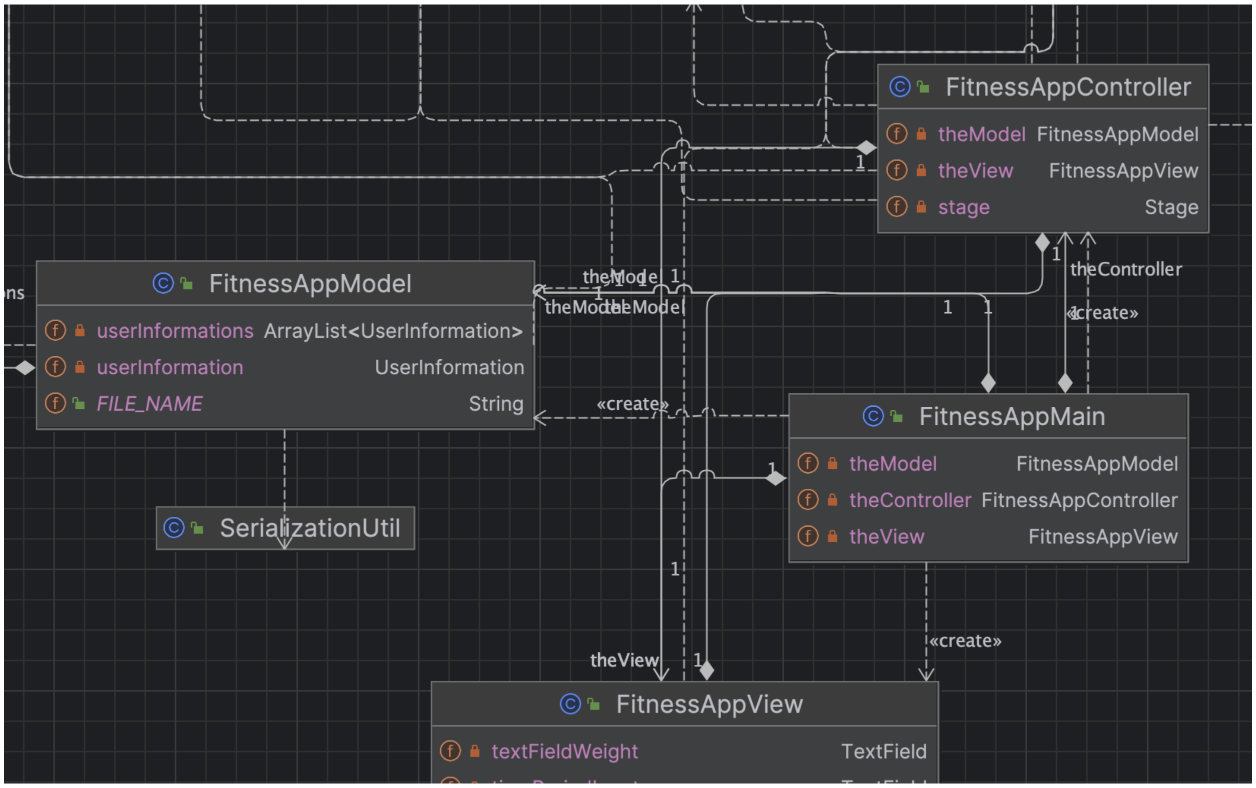
| ChatBot | |
|---|---|
| • Suggest workout details based on user input<br>• Determine workout choice (Running or Walking) | |

| ConversionUtil | |
|---|---|
| • Convert speed, weight , and heigh units<br>• Convert workout duration to hours and minutes | • CalorieCalculator uses for unit conversions |

| SerializationUtil | |
|---|---|
| • Serialize/deserialize objects to/from files | |

The program is organized into several classes, each responsible for specific aspects of the system like calorie calculation, user information, workout details, and unit conversions. The CRC cards help to outline these responsibilities and the relationships between classes. The object-oriented design of our program promotes code organization and flexibility, focusing on clear class responsibilities.

## IntelliJ UML



The above is a uml generated by intelliJ that illustrates relationships between the main, controller, model, and the view. The MVC follows an aggregate relationship: the view creates an object of the model and the controller creates both view and model objects. This is an aggregation relationship because these classes are shared between each other.