Gym Management System - User Documentation

Developed by: Group 10
Version: 1.0
Date: April 2025

1. Introduction

Welcome to the Gym Management System! This application allows gym administrators and staff to manage gym memberships, including adding new members, updating member details, and tracking membership status.
This user guide provides step-by-step instructions on how to set up and use the Gym Management System via the command line. It supports user registration, login, and membership management.

2. Getting Started

System Requirements:

- Java JDK 17 or above
- Command-line terminal (e.g., Command Prompt, Terminal, PowerShell)
- Maven installed
- PostgreSQL installed

3. Installation & Setup

Installing Maven:

- **Mac:** Install using Homebrew `brew install maven`
- **Windows:**
  - Download Maven from Maven Download
  - Extract the downloaded archive to a directory of your choice
  - Add the Maven bin directory to your system PATH
  - Verify installation by running `mvn -version` in your terminal

Database Setup

1. **Terminal: In terminal enter:**
   ```
   ~ psql postgres
   ```
2. **PostgreSQL: Enter Code as Following**
   ```
   CREATE DATABASE gym_management;
   CREATE USER gym_admin WITH PASSWORD 'password123';
   GRANT ALL PRIVILEGES ON DATABASE gym_management TO gym_admin;
   \c gym_management
   ```
3. **Run Scripts One at a Time**

   ```
   GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO gym_admin;
   ```

   ```
   GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO gym_admin;
   ```

   ```
   CREATE TABLE IF NOT EXISTS users (
     userId SERIAL PRIMARY KEY,
     userName VARCHAR(50) NOT NULL,
     userPassword VARCHAR(100) NOT NULL,
     userEmail VARCHAR(100) NOT NULL UNIQUE,
     userPhoneNum VARCHAR(20) NOT NULL,
     userAddress VARCHAR(255) NOT NULL,
     userEmergencyContactName VARCHAR(50),
     userEmergencyContactPhoneNum VARCHAR(20),
     userRole VARCHAR(20) NOT NULL,
     userCreationDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP
   );
   ```

```
CREATE TABLE IF NOT EXISTS memberships (
  membershipId SERIAL PRIMARY KEY,
  membershipType VARCHAR(50) NOT NULL,
  membershipDescription TEXT,
  membershipCost DECIMAL(10,2) NOT NULL,
  userId INTEGER NOT NULL,
  CONSTRAINT membershipFkey FOREIGN KEY (userId)
    REFERENCES users(userId)
    ON UPDATE NO ACTION
    ON DELETE CASCADE,
  dateRegistered DATE DEFAULT CURRENT_DATE
);


CREATE TABLE IF NOT EXISTS gymClasses (
  gymClassId SERIAL PRIMARY KEY,
  gymClassType VARCHAR(50) NOT NULL,
  gymClassDescription TEXT,
  trainerId INTEGER,
  CONSTRAINT trainerFkey FOREIGN KEY (trainerId)
    REFERENCES users(userId)
    ON UPDATE CASCADE
    ON DELETE SET NULL,
  classCreated DATE DEFAULT CURRENT_DATE
);


CREATE TABLE IF NOT EXISTS classEnrollments (
  enrollmentId SERIAL PRIMARY KEY,
  memberId INTEGER NOT NULL,
  CONSTRAINT studentFkey FOREIGN KEY (memberId)
    REFERENCES users(userId)
    ON UPDATE NO ACTION
    ON DELETE CASCADE,
  gymClassId INTEGER NOT NULL,
  CONSTRAINT gymClassFkey FOREIGN KEY (gymClassId)
    REFERENCES gymClasses(gymClassId)
    ON UPDATE NO ACTION
    ON DELETE CASCADE,
  enrollmentDate DATE DEFAULT CURRENT_DATE,
  CONSTRAINT uniqueEnrollment UNIQUE (memberId, gymClassId)
```

4. How to Run the Application

Step 1: Compile the project

```
mvn compile
```

Step 2: Run the application

```
mvn exec:java
```

Note: This command works after the `exec-maven-plugin` block was added to the `pom.xml` file.

5. Main Menu Overview

When you launch the program, you'll see a main menu with options like:

```
=== Gym Management System ===
1. Register a new user
2. Login as a user
3. Exit
Enter your choice:
```

Use the number keys (e.g., 1, 2, 3...) to navigate through the options.

6. Member Menu Options

Once logged in as a user, you'll see:

```
=== Member Menu ===
1. Browse workout classes
2. Enroll in a workout class
3. View my enrolled workout classes
4. Drop a workout class
5. View my membership
6. Purchase a membership
7. View my total membership expenses
8. Logout
Enter your choice:
```

The system also provides administrative functions including:

- Add New Member
- View Member Information
- Edit Member Details
- Delete Member
- List All Members

7. Membership Features

Add New Member:
Prompts you to enter member details such as Name, Age, Membership Type, and Start Date. After submission, the new member is saved in the system.

View Member Information:
Allows you to view details of a specific member by entering their ID. Displays the member's name, age, membership type, and current status.

Edit Member Details:
Enables updating of existing member information, including name, membership type, and status.

Delete Member:
Removes a member from the system after confirmation, permanently deleting their data.

List All Members:
Displays a complete list of all registered members in the system, including basic information.

Browse Workout Classes:
View the available fitness and workout classes offered by the gym.

Enroll in a Workout Class:
Register for a specific class based on availability. The system tracks your enrolled classes.

View My Enrolled Workout Classes:
Shows all classes that the currently logged-in user is enrolled in.

Drop a Workout Class:
Cancel your enrollment from a specific class.

Purchase a Membership:
Select a membership plan and proceed with payment to activate gym access.

View Total Membership Expenses:
Provides a breakdown of all expenses related to your active and past memberships, helping you track your spending.

8. Membership Statuses

Each member has a status:

- **Active:** Membership is current.
- **Expired:** Membership has ended.
- **Suspended:** Temporarily deactivated.

These are managed automatically or can be updated manually.

9. Error Handling & Tips

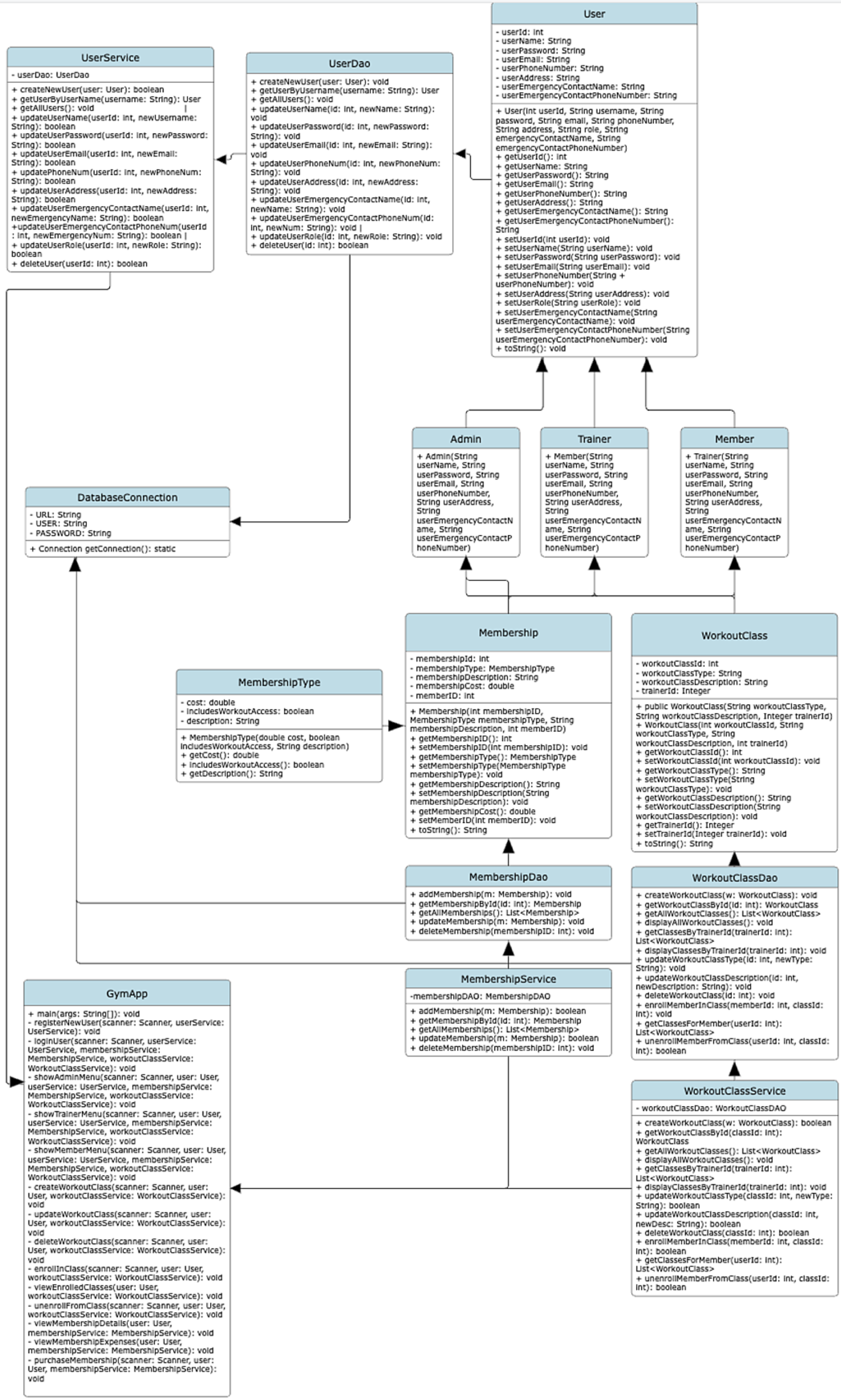
- If you enter an invalid input, the system will prompt you to try again.
- Always confirm changes when prompted to avoid accidental data loss.

10. Troubleshooting

- **Command not found:** Ensure Java and Maven are installed and properly added to your system PATH.
- **Build errors:** Make sure your project has no syntax errors and that all dependencies in `pom.xml` are correct.

11. Contact & Support

For questions or issues, please contact your development team or refer to the project README for more info.



```
UserService
- userDao: UserDao
+ createNewUser(user: User): boolean
+ getUserByUserName(username: String): User
+ getAllUsers(): void                    |
+ updateUserName(userId: int, newUsername:
String): boolean
+ updateUserPassword(userId: int, newPassword:
String): boolean
+ updateUserEmail(userId: int, newEmail:
String): boolean
+ updatePhoneNum(userId: int, newPhoneNum:
String): boolean
+ updateUserAddress(userId: int, newAddress:
String): boolean
+ updateUserEmergencyContactName(userId: int,
newEmergencyName: String): boolean
+updateUserEmergencyContactPhoneNum(userId
: int, newEmergencyNum: String): boolean |
+ updateUserRole(userId: int, newRole: String):
boolean
+ deleteUser(userId: int): boolean
```

```
UserDao
+ createNewUser(user: User): void
+ getUserByUsername(username: String): User
+ getAllUsers(): void
+ updateUserName(id: int, newName: String):
void
+ updateUserPassword(id: int, newPassword:
String): void
+ updateUserEmail(id: int, newEmail: String):
void
+ updateUserPhoneNum(id: int, newPhoneNum:
String): void
+ updateUserAddress(id: int, newAddress:
String): void
+ updateUserEmergencyContactName(id: int,
newName: String): void
+ updateUserEmergencyContactPhoneNum(id:
int, newNum: String): void |
+ updateUserRole(id: int, newRole: String): void
+ deleteUser(id: int): boolean
```

```
User
- userId: int
- userName: String
- userPassword: String
- userEmail: String
- userPhoneNumber: String
- userAddress: String
- userEmergencyContactName: String
- userEmergencyContactPhoneNumber: String
+ User(int userId, String username, String
password, String email, String phoneNumber,
String address, String role, String
emergencyContactName, String
emergencyContactPhoneNumber)
+ getUserId(): int
+ getUserName: String
+ getUserPassword(): String
+ getUserEmail(): String
+ getUserPhoneNumber(): String
+ getUserAddress(): String
+ getUserEmergencyContactName(): String
+ getUserEmergencyContactPhoneNumber():
String
+ setUserId(int userId): void
+ setUserName(String userName): void
+ setUserPassword(String userPassword): void
+ setUserEmail(String userEmail): void
+ setUserPhoneNumber(String +
userPhoneNumber): void
+ setUserAddress(String userAddress): void
+ setUserRole(String userRole): void
+ setUserEmergencyContactName(String
userEmergencyContactName): void
+ setUserEmergencyContactPhoneNumber(String
userEmergencyContactPhoneNumber): void
+ toString(): void
```

```
Admin
+ Admin(String
userName, String
userPassword, String
userEmail, String
userPhoneNumber,
String userAddress,
String
userEmergencyContactN
ame, String
userEmergencyContactP
honeNumber)
```

```
Trainer
+ Member(String
userName, String
userPassword, String
userEmail, String
userPhoneNumber,
String userAddress,
String
userEmergencyContactN
ame, String
userEmergencyContactP
honeNumber)
```

```
Member
+ Trainer(String
userName, String
userPassword, String
userEmail, String
userPhoneNumber,
String userAddress,
String
userEmergencyContactN
ame, String
userEmergencyContactP
honeNumber)
```

```
DatabaseConnection
- URL: String
- USER: String
- PASSWORD: String
+ Connection getConnection(): static
```

```
MembershipType
- cost: double
- includesWorkoutAccess: boolean
- description: String
+ MembershipType(double cost, boolean
includesWorkoutAccess, String description)
+ getCost(): double
+ includesWorkoutAccess(): boolean
+ getDescription(): String
```

```
Membership
- membershipId: int
- membershipType: MembershipType
- membershipDescription: String
- membershipCost: double
- memberID: int
+ Membership(int membershipID,
MembershipType membershipType, String
membershipDescription, int memberID)
+ getMembershipID(): int
+ setMembershipID(int membershipID): void
+ getMembershipType(): MembershipType
+ setMembershipType(MembershipType
membershipType): void
+ getMembershipDescription(): String
+ setMembershipDescription(String
membershipDescription): void
+ getMembershipCost(): double
+ setMemberID(int memberID): void
+ toString(): String
```

```
WorkoutClass
- workoutClassId: int
- workoutClassType: String
- workoutClassDescription: String
- trainerId: Integer
+ public WorkoutClass(String workoutClassType,
String workoutClassDescription, Integer trainerId)
+ WorkoutClass(int workoutClassId, String
workoutClassType, String
workoutClassDescription, int trainerId)
+ getWorkoutClassId(): int
+ setWorkoutClassId(int workoutClassId): void
+ getWorkoutClassType(): String
+ setWorkoutClassType(String
workoutClassType): void
+ getWorkoutClassDescription(): String
+ setWorkoutClassDescription(String
workoutClassDescription): void
+ getTrainerId(): Integer
+ setTrainerId(Integer trainerId): void
+ toString(): String
```

```
MembershipDao
+ addMembership(m: Membership): void
+ getMembershipById(id: int): Membership
+ getAllMemberships(): List<Membership>
+ updateMembership(m: Membership): void
+ deleteMembership(membershipID: int): void
```

```
WorkoutClassDao
+ createWorkoutClass(w: WorkoutClass): void
+ getWorkoutClassById(id: int): WorkoutClass
+ getAllWorkoutClasses(): List<WorkoutClass>
+ displayAllWorkoutClasses(): void
+ getClassesByTrainerId(trainerId: int):
List<WorkoutClass>
+ displayClassesByTrainerId(trainerId: int): void
+ updateWorkoutClassType(id: int, newType:
String): void
+ updateWorkoutClassDescription(id: int,
newDescription: String): void
+ deleteWorkoutClass(id: int): void
+ enrollMemberInClass(memberId: int, classId:
int): void
+ getClassesForMember(userId: int):
List<WorkoutClass>
+ unenrollMemberFromClass(userId: int, classId:
int): boolean
```

```
MembershipService
-membershipDAO: MembershipDAO
+ addMembership(m: Membership): boolean
+ getMembershipById(id: int): Membership
+ getAllMemberships(): List<Membership>
+ updateMembership(m: Membership): boolean
+ deleteMembership(membershipID: int): void
```

```
GymApp
+ main(args: String[]): void
- registerNewUser(scanner: Scanner, userService:
UserService): void
- loginUser(scanner: Scanner, userService:
UserService, membershipService:
MembershipService, workoutClassService:
WorkoutClassService): void
- showAdminMenu(scanner: Scanner, user: User,
userService: UserService, membershipService:
MembershipService, workoutClassService:
WorkoutClassService): void
- showTrainerMenu(scanner: Scanner, user: User,
userService: UserService, membershipService:
MembershipService, workoutClassService:
WorkoutClassService): void
- showMemberMenu(scanner: Scanner, user: User,
userService: UserService, membershipService:
MembershipService, workoutClassService:
WorkoutClassService): void
- createWorkoutClass(scanner: Scanner, user:
User, workoutClassService: WorkoutClassService):
void
- updateWorkoutClass(scanner: Scanner, user:
User, workoutClassService: WorkoutClassService):
void
- deleteWorkoutClass(scanner: Scanner, user:
User, workoutClassService: WorkoutClassService):
void
- enrollInClass(scanner: Scanner, user: User,
workoutClassService: WorkoutClassService): void
- viewEnrolledClasses(user: User,
workoutClassService: WorkoutClassService): void
- unenrollFromClass(scanner: Scanner, user: User,
workoutClassService: WorkoutClassService): void
- viewMembershipDetails(user: User,
membershipService: MembershipService): void
- viewMembershipExpenses(user: User,
membershipService: MembershipService): void
- purchaseMembership(scanner: Scanner, user:
User, membershipService: MembershipService):
void
```

```
WorkoutClassService
- workoutClassDao: WorkoutClassDAO
+ createWorkoutClass(w: WorkoutClass): boolean
+ getWorkoutClassById(classId: int):
WorkoutClass
+ getAllWorkoutClasses(): List<WorkoutClass>
+ displayAllWorkoutClasses(): void
+ getClassesByTrainerId(trainerId: int):
List<WorkoutClass>
+ displayClassesByTrainerId(trainerId: int): void
+ updateWorkoutClassType(classId: int, newType:
String): boolean
+ updateWorkoutClassDescription(classId: int,
newDesc: String): boolean
+ deleteWorkoutClass(classId: int): boolean
+ enrollMemberInClass(memberId: int, classId:
int): boolean
+ getClassesForMember(userId: int):
List<WorkoutClass>
+ unenrollMemberFromClass(userId: int, classId:
int): boolean
```

Development Documentation

JavaDocs

```
/* Author: Jack Williams
*  Dates: April 2nd - April 3rd, 2025
*  Description: The User class is the parent class for all kinds of users.
*  There are three kinds of users: Member, Trainer, and Admin */
```

```
/* Author: Jack Williams
*  Date: April 7th, 2025
*  Description: The UserDao class communicates with the database to
perform SQL queries.
*  This file includes all CRUD operations for the user */
```

```
/* Author: Jack Williams
*  Date: April 7th, 2025
*  Description: The UserService class is a middleware class used to let the app
*  communicate with the DAO. This class includes middleware for
*  all the CRUD operations */
```

```
/*
The workoutClass represents a classes that can be offered at the gym.
Trainers can create, update and delete workout classes.
```

Members can view and enroll in workout classes. */

/* The WorkoutClassDao class communicates with the database to perform SQL queries.

 * This file includes all CRUD operations for workout classes */

/*

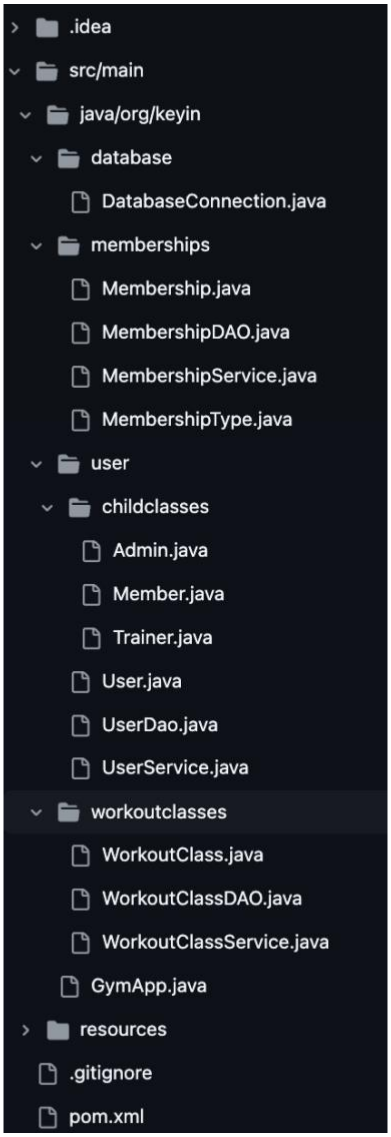
The WorkoutClassService class is a middleware class used to let the app

communicate with the DAO. This class includes middleware for

all the CRUD operations related to workout classes

*/

Directory Structure



```
> .idea
v src/main
  v java/org/keyin
    v database
        DatabaseConnection.java
    v memberships
        Membership.java
        MembershipDAO.java
        MembershipService.java
        MembershipType.java
    v user
      v childclasses
          Admin.java
          Member.java
          Trainer.java
        User.java
        UserDao.java
        UserService.java
    v workoutclasses
        WorkoutClass.java
        WorkoutClassDAO.java
        WorkoutClassService.java
      GymApp.java
  > resources
  .gitignore
  pom.xml
```

Build Process

Navigate to /src/main/java/org.keyin/GymApp and press play

Dependencies

Maven, BCrypt, Postgres

Database Setup

Create a new database in your Postgres called gym_management. Make sure to use the default localhost address of 5432. Replace the USER and PASSWORD fields in the DatabaseConnection.java file on lines 11 and 12 with your own username and password. The default username is postgres. Once all of this is completed, the program should be able to connect to the database and populate it with the needed tables when it runs.

GitHub Cloning

To clone the project from GitHub, run the following commands in your terminal:

```
git clone https://github.com/JWilliams-Keyin/Group10JavaFinalSprint

cd Group10JavaFinalSprint
```

Alternatively, you can use VSCode or IntelliJ's built-in cloning operations by following the steps they provide.

Individual Reports

Name: Jack Williams

Contributions

- Developed User class
- Developed UserDao class
- Developed UserService class
- Created development documentation
- Created Trello board
- Dealt with conflicts in GitHub repo

Challenges

- Minor naming changes
- New attributes added later, not a big deal
- Used .equals instead of == in UserDao, which always came back as true

----

Name: Zackery Strickland

Contributions

- Developed the `Membership` class to manage membership details, status, and renewals.
- Developed the Membership DAO and Service classes to work with the SQL and function.
- Implemented methods to activate, suspend, cancel, and update memberships.
- Created a professional user documentation.
- Participated in code reviews.

- Created the User Documentation

Challenges

- My tutor helped me understand how to use an Enum which was something I had not done before.
- Syncing class methods with the team's menu logic required good communication and test coordination.

Name: David Elliott

Contributions

- Created the SQL scripts in the scripts.sql file for the database structure
- Completed the DatabaseConnection.java
- Created the WorkoutClass.java, WorkoutClassDAO.java and WorkoutClassService.java files.
- Created the GymClass.java
- Integration of the classes into the GymApp main menu file.

Challenges

- Early completion of of sql structures was required to avoid bottlenecks, this demanded a quick review of SQL structures and coordination with the group to develop the scripts concurrently with the program's processes.
- Integration of the classes written by different authors into the main GymApp program took as long as the initial creation of the main menu code.
