Gym Management System

Software and Development Manual

Author

Kepich, Eugene

Keyin College
Software Development
Group 12

Table of Contents

1 Introduction

1.1 General Overview

GymApp is a complete management solution designed specifically for a small gym or fitness center. It provides all the essential functionality needed for managing users, memberships, workout classes, and role-based access control.

At the same time, GymApp is intentionally built as a simple, clean, and easily extensible project — making it an ideal starting point for further development and customization. The codebase follows clear logic and consistent structure, allowing future developers to quickly understand, modify, and expand the system to suit more complex needs.

Whether used as-is for a small business or as a foundation for a larger gym management platform, GymApp provides a solid and flexible starting point.

1.2 Key Features:

- User registration and login system
- Role-based access: Admin, Trainer, Member
- Membership types and subscriptions management
- Workout classes management
- PostgreSQL database backend
- Easy database initialization with built-in data or external files
- Console-based user interface
- Simple and extensible code structure for future growth

2 Installation and Compilation

2.1 Prerequisites

- Java 21 JDK installed
- Maven installed
- PostgreSQL database server running
- Git (optional, for cloning the repository)

2.2 Installation Steps

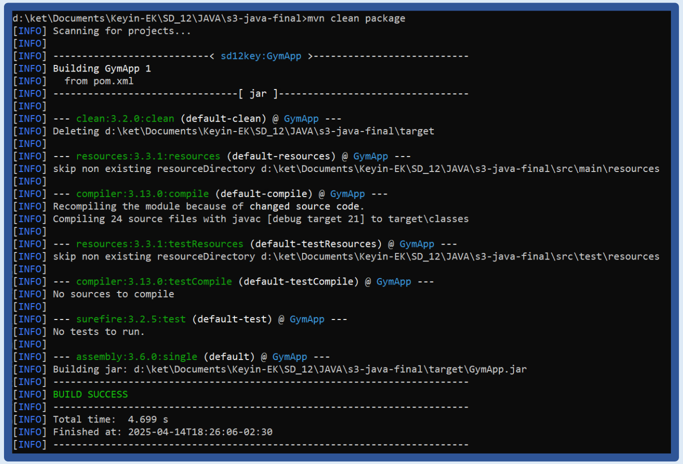
1. Clone the repository (if available) or download the source files

   ```
   git clone [repository-url]
   cd [cloned-repository-folder]
   ```

2. Set up PostgreSQL database
   - Ensure PostgreSQL server is running
   - Update the connection details in **DatabaseConnection.java** if needed
   - No need to manually create the database or tables - the application will handle this automatically (see below for a database schema diagram)

3. Compile the project using Maven

   ```
   mvn clean compile
   ```



```
d:\ket\Documents\Keyin-EK\SD_12\JAVA\s3-java-final>mvn clean package
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< sd12key:GymApp >------------------------
[INFO] Building GymApp 1
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- clean:3.2.0:clean (default-clean) @ GymApp ---
[INFO] Deleting d:\ket\Documents\Keyin-EK\SD_12\JAVA\s3-java-final\target
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ GymApp ---
[INFO] skip non existing resourceDirectory d:\ket\Documents\Keyin-EK\SD_12\JAVA\s3-java-final\src\main\resources
[INFO]
[INFO] --- compiler:3.13.0:compile (default-compile) @ GymApp ---
[INFO] Recompiling the module because of changed source code.
[INFO] Compiling 24 source files with javac [debug target 21] to target\classes
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ GymApp ---
[INFO] skip non existing resourceDirectory d:\ket\Documents\Keyin-EK\SD_12\JAVA\s3-java-final\src\test\resources
[INFO]
[INFO] --- compiler:3.13.0:testCompile (default-testCompile) @ GymApp ---
[INFO] No sources to compile
[INFO]
[INFO] --- surefire:3.2.5:test (default-test) @ GymApp ---
[INFO] No tests to run.
[INFO]
[INFO] --- assembly:3.6.0:single (default) @ GymApp ---
[INFO] Building jar: d:\ket\Documents\Keyin-EK\SD_12\JAVA\s3-java-final\target\GymApp.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.699 s
[INFO] Finished at: 2025-04-14T18:26:06-02:30
[INFO] ------------------------------------------------------------------------
```

4. Run the application

   `mvn exec:java`
   (automatically creates tables on 1st run)

   `mvn exec:java@drop-run`
   (drop tables and create new)

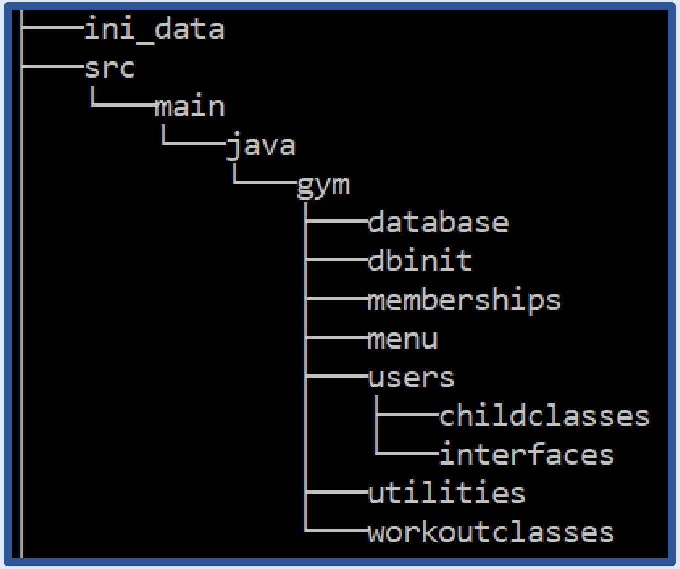
   `mvn exec:java@drop-init-run`
   (drop tables, create new and initialize from CSV files, must reside in **ini_data** folder)

5. Create a fat JAR (production build)

   `mvn clean package`
   (this will create a standalone GymApp.jar in the target directory that can be run as a standalone app using Java Runtime)

   ```
   java -jar target/GymApp.jar
   java -jar target/GymApp.jar --drop
   java -jar target/GymApp.jar --drop --init
   ```

2.3 Folder Structure



```
├───ini_data
├───src
│   └───main
│       └───java
│           └───gym
│               ├───database
│               ├───dbinit
│               ├───memberships
│               ├───menu
│               ├───users
│               │   ├───childclasses
│               │   └───interfaces
│               ├───utilities
│               └───workoutclasses
```

2.4 Database Schema

The application automatically creates the following four table

- **users** - Stores all user accounts
- **membership_types** - Contains available membership options (auto-populated with defaults)
- **memberships** - Tracks user membership purchases
- **workout_classes** - Contains class offerings by trainers

2.5 Default Membership Types

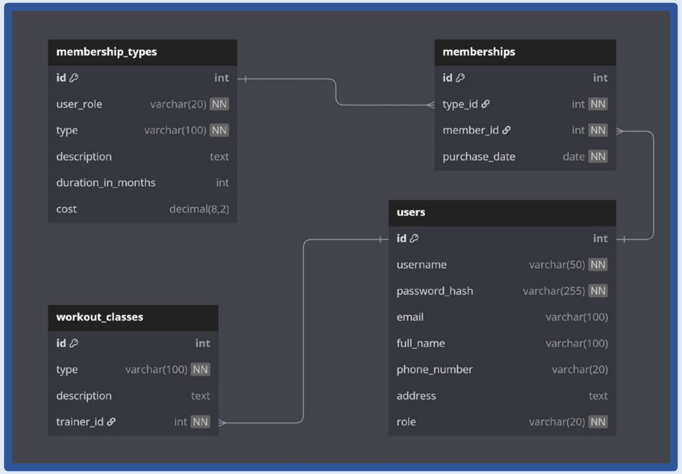
The system automatically creates these membership options if none exist.

For Members:

- Monthly ($50.00, 1 month)
- Quarterly ($135.00, 3 months)
- Semi-annual ($250.00, 6 months)
- Annual ($450.00, 12 months)

For Trainers:

- Monthly ($35.00, 1 month)
- Quarterly ($90.00, 3 months)
- Semi-annual ($170.00, 6 months)
- Annual ($300.00, 12 months)



membership_types	
id	int
user_role	varchar(20) NN
type	varchar(100) NN
description	text
duration_in_months	int
cost	decimal(8,2)

memberships	
id	int
type_id	int NN
member_id	int NN
purchase_date	date NN

users	
id	int
username	varchar(50) NN
password_hash	varchar(255) NN
email	varchar(100)
full_name	varchar(100)
phone_number	varchar(20)
address	text
role	varchar(20) NN

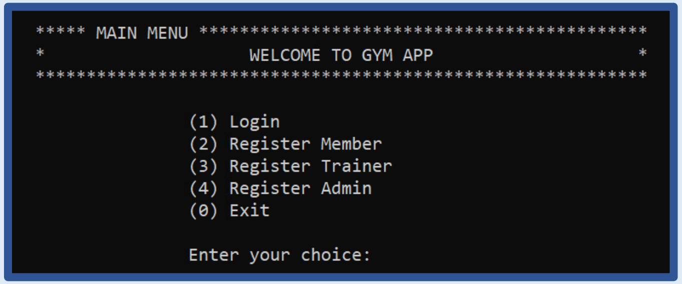
workout_classes	
id	int
type	varchar(100) NN
description	text
trainer_id	int NN

3 User Manual

3.1 Starting the Application

Run the application using one of the Maven commands mentioned above.

The main menu will appear with the following options:



```
***** MAIN MENU ***********************************************
*                  WELCOME TO GYM APP                         *
***************************************************************


               (1) Login
               (2) Register Member
               (3) Register Trainer
               (4) Register Admin
               (0) Exit

               Enter your choice:
```

3.2 User Registration

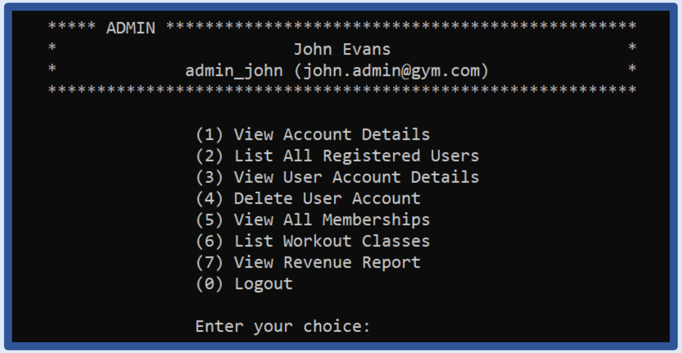
Select the appropriate registration option (**2-4**), follow the prompts to enter:

1. **Username** (must be unique)
2. **Password** (must meet complexity requirements)
3. **Full name**
4. **Address**
5. **Phone number**
6. **Email address**

3.3 Logging In

1. Select option **1** from the main menu
2. Enter your username and password
3. After successful login, you'll see a role-specific menu

3.4 Administrator Menu

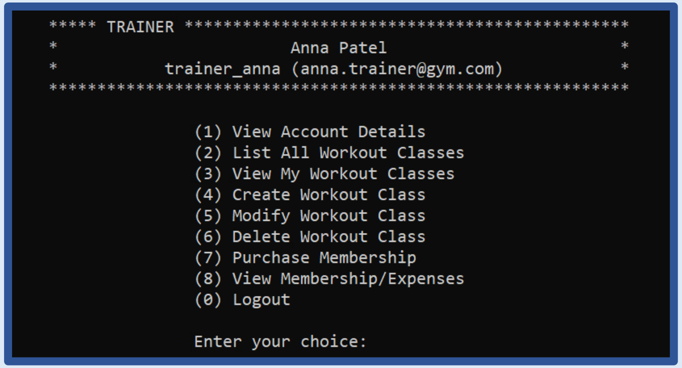


```
***** ADMIN ***************************************************
*                       John Evans                            *
*             admin_john (john.admin@gym.com)                 *
***************************************************************


               (1) View Account Details
               (2) List All Registered Users
               (3) View User Account Details
               (4) Delete User Account
               (5) View All Memberships
               (6) List Workout Classes
               (7) View Revenue Report
               (0) Logout

               Enter your choice:
```

3.5 Trainer Manu

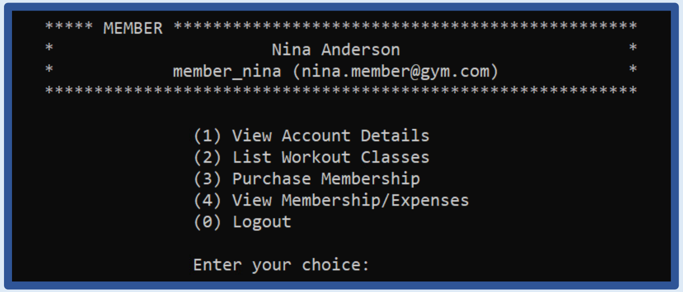


```
***** TRAINER *************************************************
*                       Anna Patel                            *
*           trainer_anna (anna.trainer@gym.com)               *
***************************************************************


               (1) View Account Details
               (2) List All Workout Classes
               (3) View My Workout Classes
               (4) Create Workout Class
               (5) Modify Workout Class
               (6) Delete Workout Class
               (7) Purchase Membership
               (8) View Membership/Expenses
               (0) Logout

               Enter your choice:
```

3.6 Member Menu



```
***** MEMBER **********************************************
*                   Nina Anderson                         *
*          member_nina (nina.member@gym.com)              *
***********************************************************

               (1) View Account Details
               (2) List Workout Classes
               (3) Purchase Membership
               (4) View Membership/Expenses
               (0) Logout

               Enter your choice:
```

3.7 Exiting the Application

1. Select option **0** from any menu to logout or exit
2. The application will clean up resources and close properly

4 Development Documentation

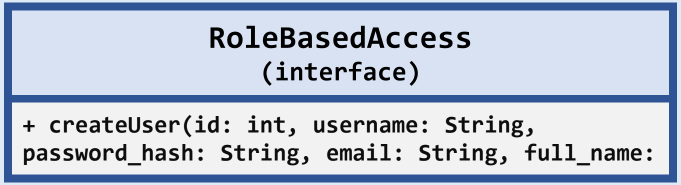
This section is designed for advanced users and developers who wish to extend the gym management system by adding new features or modifying existing behavior. Here, we provide a detailed breakdown of class relationships, including inheritance, associations, and dependencies, along with explanations of key methods and their interactions.

4.1 RoleBasedAccess Interface: Purpose and Functionality

The **RoleBasedAccess** interface serves two critical functions in the system:

- Role-Specific Behavior Enforcement: It defines abstract methods (**showUserMenu**, **handleMenuChoice**, **getMenuItems**) that each user role (**Admin**, **Trainer**, **Member**) must implement, ensuring consistent menu handling across roles while allowing customization.
- Factory Pattern for User Creation: Its static **createUser** methods centralize user instantiation, dynamically generating the correct subclass (**Admin**, **Trainer**, or **Member**) based on the provided role string. This decouples object creation from business logic, simplifying role-based scalability (e.g., adding a new role like Staff would only require a new subclass and updates to the factory method).

By combining these responsibilities, **RoleBasedAccess** acts as both a contract for role-specific workflows and a scalable factory, reducing code duplication and promoting maintainability.

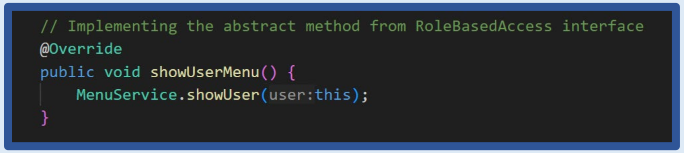

RoleBasedAccess
(interface)
+ createUser(id: int, username: String,
password_hash: String, email: String, full_name:

```
String, address: String, phone_number: String,
role: String, exit_on_error: boolean): User

(+ overloads)
```

```
+ showUserMenu(): void
+ handleMenuChoice(choice: String): void
+ getMenuItems(): String[]
```

4.2 Role-Based Access in the Menu System



```java
// Implementing the abstract method from RoleBasedAccess interface
@Override
public void showUserMenu() {
    MenuService.showUser(user:this);
}
```

The **showUser(User user)** method enables dynamic, role-specific menu navigation by leveraging polymorphism through the **RoleBasedAccess** interface. When a user logs in, calling **logged_user.showUserMenu()** triggers the following flow:

❖ **Polymorphic Dispatch**

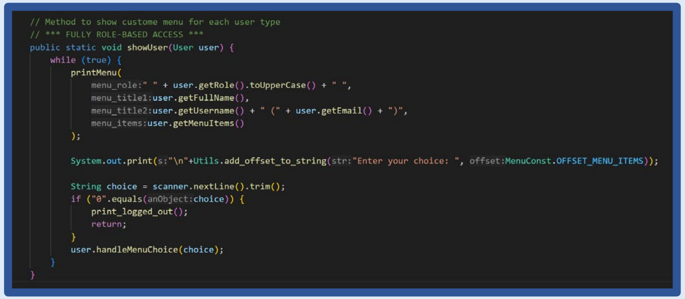
- The actual implementation of **showUserMenu()** is determined at runtime based on the user's concrete class (**Admin**, **Trainer**, or **Member**).
- For example, if **logged_user** is a Trainer, it calls Trainer's overridden **showUserMenu()**, which delegates to **MenuService.showUser(this)**.

❖ **Role-Specific Menus**

- **MenuService.showUser(user)** fetches role-specific menu items via **user.getMenuItems()** (e.g., **MenuConst.TRAINER_MENU_ITEMS** for trainers).
- User input is routed to **user.handleMenuChoice(choice)**, invoking the correct handler (e.g., **MenuService.handleTrainerMenu()**).

❖ **Key Advantage: Decoupled Design**

- Centralized entry point (**logged_user.showUserMenu()**) hides role-specific details behind the User abstraction.
- Adding a new role (e.g., **Staff**) would only require subclassing **User** and implementing **getMenuItems()/handleMenuChoice()**—no changes to **MenuService**.
- This design ensures type safety (e.g., a **Member** cannot access **Trainer** methods) and scalability while minimizing conditional checks.



```java
// Method to show custome menu for each user type
// *** FULLY ROLE-BASED ACCESS ***
public static void showUser(User user) {
    while (true) {
        printMenu(
            menu_role:" " + user.getRole().toUpperCase() + " ",
            menu_title1:user.getFullName(),
            menu_title2:user.getUsername() + " (" + user.getEmail() + ")",
            menu_items:user.getMenuItems()
        );

        System.out.print(s:"\n"+Utils.add_offset_to_string(str:"Enter your choice: ", offset:MenuConst.OFFSET_MENU_ITEMS));

        String choice = scanner.nextLine().trim();
        if ("0".equals(anObject:choice)) {
            print_logged_out();
            return;
        }
        user.handleMenuChoice(choice);
    }
}
```

❖ **Example Workflow:**

```
User logged_user = UserService.login("trainer_anna", "pass123");
// Polymorphic call: Resolves to Trainer.showUserMenu() → MenuService.showUser(this)
logged_user.showUserMenu();
```

4.3 Core Abstract User Class

```
User
(abstract, implements RoleBasesAccess)
```

```
+ ROLE_ADMIN: String = “admin”
+ ROLE_TRAINER: String = “trainer”
+ ROLE_MEMBER: String = “member”
```

```
- id: int (final)
- username: String
- password_hash: String
- email: String
- full_name: String
- address: String
- phone_number: String
- role: String (admin/trainer/member)
```

```
+ User(id: int, username: String, password_hash:
String, email: String, full_name: String,
address: String, phone_number: String, role:
String)
(+ Overload with id = 0)
```

```
+ getId(): int
+ getUsername(): String
+ getPasswordHash(): String
+ getEmail(): String
+ getFullName(): String
+ getAddress(): String
+ getPhoneNumber(): String
+ getRole(): String
+ setUsername(username: String): void
+ setPasswordHash(password_hash: String): void
+ setEmail(email: String): void
+ setFullName(full_name: String): void
+ setAddress(address: String): void
+ setPhoneNumber(phone_number: String): void
```

```
+ canHaveMembership(): boolean (abstract)
+ canTeachClass(): boolean (abstract)
```

```
+ showUserMenu(): void (from RoleBasedAccess)
+ handleMenuChoice(choice: String): void (from
RoleBasedAccess)
+ getMenuItems(): String[] (from
RoleBasedAccess)
```

```
+ toString(): String
+ toStringNoId(): String
```

4.4 User Subclasses

```
Admin
(extends User)
----------------------------------------
role: String = User.ROLE_ADMIN
----------------------------------------
+ Admin(id: int, username: String,
password_hash: String, email: String, full_name:
String, address: String, phone_number: String)

(+ Overload with id = 0)
----------------------------------------
+ canHaveMembership(): boolean
+ canTeachClass(): boolean
----------------------------------------
+ showUserMenu(): void
+ handleMenuChoice(choice: String): void
+ getMenuItems(): String[]
```

Role Specific Behaviors:
+ canHaveMembership(): boolean - Returns false (Admins cannot have memberships)
+ canTeachClass(): boolean - Returns false (Admins cannot teach classes)

Menu Handling (Delegates to MenuService):
+ showUserMenu(): void - Calls MenuService.showUser(this)
+ handleMenuChoice(choice: String): void - Calls MenuService.handleAdminMenu(this, choice)
+ getMenuItems(): String[] – Returns MenuConst.ADMIN_MENU_ITEMS

```
Trainer
(extends User)
----------------------------------------
role: String = User.ROLE_TRAINER
----------------------------------------
+ Trainer(id: int, username: String,
password_hash: String, email: String, full_name:
String, address: String, phone_number: String)

(+ Overload with id = 0)
----------------------------------------
+ canHaveMembership(): boolean
+ canTeachClass(): boolean
----------------------------------------
+ showUserMenu(): void
+ handleMenuChoice(choice: String): void
+ getMenuItems(): String[]
```

Role Specific Behaviors:
+ canHaveMembership(): boolean - Returns true (Trainers can have memberships)
+ canTeachClass(): boolean - Returns true (Trainers can teach classes)

Menu Handling (Delegates to MenuService):
+ showUserMenu(): void - Calls MenuService.showUser(this)
+ handleMenuChoice(choice: String): void - Calls MenuService.handleTrainerMenu(this, choice)
+ getMenuItems(): String[] – Returns MenuConst.TRAINER_MENU_ITEMS

```
Member
(extends User)
--------------------------------------------------
role: String = User.ROLE_MEMBER
--------------------------------------------------
+ Member(id: int, username: String,
password_hash: String, email: String, full_name:
String, address: String, phone_number: String)

(+ Overload with id = 0)
--------------------------------------------------
+ canHaveMembership(): boolean
+ canTeachClass(): boolean
--------------------------------------------------
+ showUserMenu(): void
+ handleMenuChoice(choice: String): void
+ getMenuItems(): String[]
```

Role Specific Behaviors:
+ canHaveMembership(): boolean - Returns true (Members can have memberships)
+ canTeachClass(): boolean - Returns false (Members cannot teach classes)

Menu Handling (Delegates to MenuService):
+ showUserMenu(): void - Calls MenuService.showUser(this)
+ handleMenuChoice(choice: String): void - Calls MenuService.handleMemberMenu(this, choice)
+ getMenuItems(): String[] – Returns MenuConst.MEMBER_MENU_ITEMS

4.5 MembershipType Class

```
MembershipType
--------------------------------------------------
- id: int (final)
- user_role: String
- type: String
- description: String
- duration_in_months: int
- cost: double
--------------------------------------------------
+ MembershipType(id: int, user_role: String,
type: String

(+ Overload with id = 0)
--------------------------------------------------
+ getId(): int
+ getUserRole(): String
+ getType(): String
+ getDescription(): String
+ getDurationInMonths(): int
+ getCost(): double
--------------------------------------------------
+ toString(): String
+ toStringNoId(): String
```

Note: no setters for this Class

4.6 Membership Class

```
Membership
----------------------------------------------
- id: int (final)
- type: MembershipType
- user: User
- purchase_date: LocalDate
----------------------------------------------
+ Membership(id: int, type: MembershipType,
user: User, purchase_date: LocalDate)

(+ Overload with id = 0)
----------------------------------------------
+ getId(): int
+ getType(): MembershipType
+ getUser(): User
+ getPurchaseDate(): LocalDate
----------------------------------------------
+ setType(type: MembershipType): void
+ setUser(user: User): void
+ setPurchaseDate(purchase_date: LocalDate):
void
----------------------------------------------
+ getExpirationDate(): LocalDate
+ isExpired(): boolean
----------------------------------------------
+ toString(): String
```

4.7 WorkoutClass Class

```
WorkoutClass
----------------------------------------------
- id: int (final)
- type: String (e.g., "Yoga", "HIIT")
- description: String
- trainer: Trainer (must be a Trainer
----------------------------------------------
+ WorkoutClass(id: int, type: String,
description: String, trainer: Trainer)

(+ Overload with id = 0)
----------------------------------------------
+ getId(): int
+ getType(): String
+ getDescription(): String
+ getTrainer(): Trainer
----------------------------------------------
+ setType(type: String): void
+ setDescription(description: String): void
+ setTrainer(trainer: Trainer): void
----------------------------------------------
+ toString(): String
+ toStringNoId(): String
+ toStringNoName(): String
----------------------------------------------
+ toString(): String
```

4.8 DatabaseConnection Class

The **DatabaseConnection** class handles low-level database connectivity.

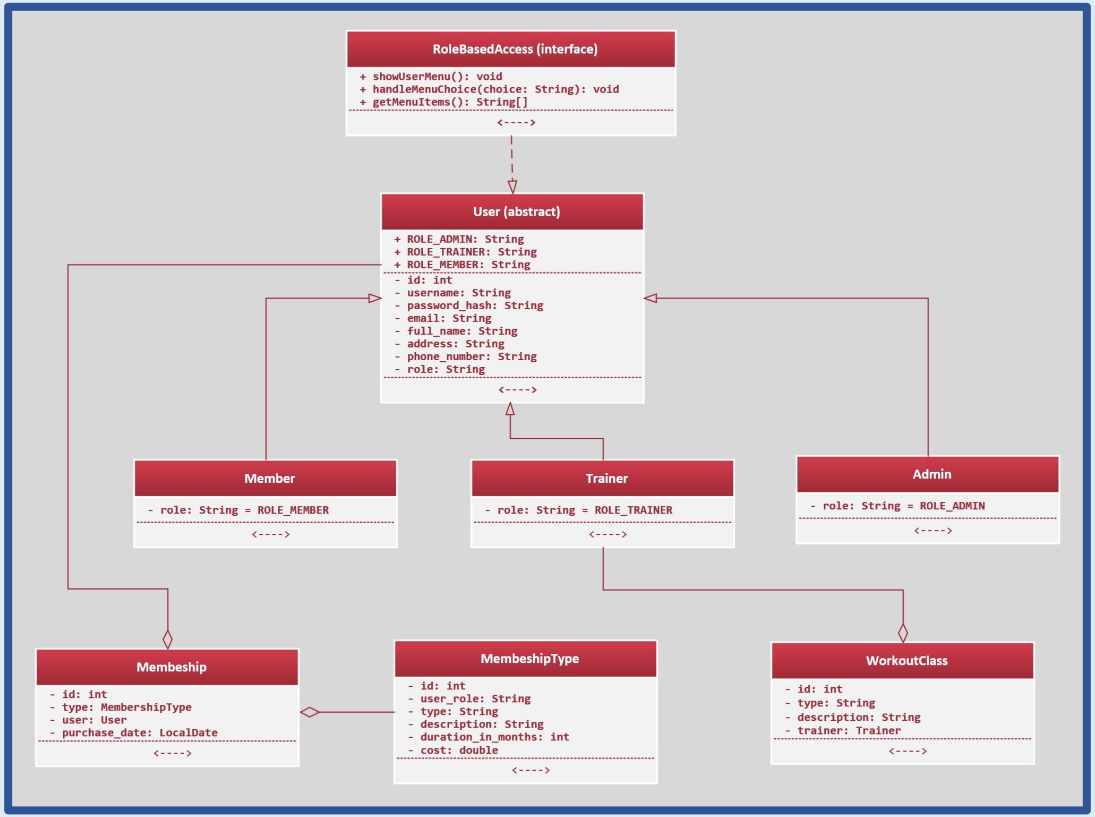
4.9 Layered Architecture

The system employs a strict separation of concerns through layered design, where each core domain class (**User**, **MembershipType**, **Membership**, **WorkoutClass**) has dedicated DAO and Service components:

- Persistence Layer DAO (Data Access Object): Handles raw database operations (CRUD) for single entities.
- Business Layer (Service): business logic, report-generation

Detailed documentation is available upon request, though the code is structured for readability.

5 Appendix: UML Diagram


RoleBasedAccess (interface)
+ showUserMenu(): void
+ handleMenuChoice(choice: String): void
+ getMenuItems(): String[]
<---->
User (abstract)
+ ROLE_ADMIN: String
+ ROLE_TRAINER: String
+ ROLE_MEMBER: String
- id: int
- username: String
- password_hash: String
- email: String
- full_name: String
- address: String
- phone_number: String
- role: String
<---->
Member
- role: String = ROLE_MEMBER
<---->
Trainer
- role: String = ROLE_TRAINER
<---->
Admin
- role: String = ROLE_ADMIN
<---->
Membeship
- id: int
- type: MembershipType
- user: User
- purchase_date: LocalDate
<---->
MembeshipType
- id: int
- user_role: String
- type: String
- description: String
- duration_in_months: int
- cost: double
<---->
WorkoutClass
- id: int
- type: String
- description: String
- trainer: Trainer
<---->