CS4001NI Programming

Final Submission

2024 Autumn

Student Name: Bikram Tamang

London Met ID: 24046576

College ID: NP01AI4A240095

Group: AI5

Assignment Due Date: 16th May, 2025

Assignment Submission Date: 16th May, 2025

I confirm that I understand my coursework needs to be submitted online via MySecondTeacher under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.

10% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

- **90** Not Cited or Quoted 10%
  Matches with neither in-text citation nor quotation marks
- **0** Missing Quotations 0%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

1% Internet sources
0% Publications
10% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Submitted works	
	Asia Pacific Instutute of Information Technology on 2021-06-01	1%
2	Submitted works	
	iacademy on 2025-05-14	1%
3	Submitted works	
	Colorado Technical University Online on 2015-01-28	<1%
4	Submitted works	
	National University of Ireland, Galway on 2024-11-23	<1%
5	Submitted works	
	Central Queensland University on 2023-04-25	<1%
6	Submitted works	
	London School of Science & Technology on 2018-01-12	<1%
7	Submitted works	
	National University of Ireland, Galway on 2024-10-19	<1%
8	Submitted works	
	Asia Pacific University College of Technology and Innovation (UCTI) on 2025-02-04	<1%
9	Internet	
	theee.ai	<1%
10	Submitted works	
	Southern New Hampshire University - Continuing Education on 2020-04-26	<1%

11	Internet	
www.driversinsurance.com.au		<1%
12	Submitted works	
University of Mauritius on 2016-04-07		<1%
13	Submitted works	
Australian Catholic University on 2023-11-08		<1%
14	Internet	
www.bartleby.com		<1%
15	Internet	
www.saddleback.edu		<1%
16	Submitted works	
AlHussein Technical University on 2023-09-06		<1%
17	Submitted works	
NCC Education on 2022-05-21		<1%
18	Submitted works	
University College of Bahrain on 2025-04-28		<1%
19	Submitted works	
Global Banking Training on 2023-07-09		<1%
20	Internet	
www.coursehero.com		<1%

Fig: Turnitin Report

Contents

1. Introduction

The problem this project is trying to solve is to make it easier to store and track the members of the Gym. Old methods like spreadsheet and handwritten registers are not so effective for and are prone to errors. For the easier access and more automated results of the action performed, this project is designed in this manner to mitigate those errors and help in automating the process of registration and other things.

The objective of this project named **Gym Management System** is to allow the owner of the gym to easily register member in the system, manage gym membership, their payments and track records of their attendance and membership plans. This project also allows the owner to easily fetch the data of the gym members, their plans, membership track records and calculate discount in an effective manner. Also, this project features the clean and easy to use GUI for the convenience of the user using this. This project is built upon

Java Language and **BlueJ** was used as an Integrated Development Environment. Concept of OOPs i.e. Object-Oriented Programming like classes, inheritance, and polymorphism etc. were used for structural and management of the data efficiently.

This project aims to provide user-friendly member administration, enhance overall experience and automate various things in a proper manner.

1.1. Aim and Objective

The main goal of the **Gym Management System** project is to make it easier to store and track gym members, replacing old and error-prone methods like spreadsheets and handwritten registers. The system is designed to automate registration and other tasks, reducing errors and simplifying the process.

The project's objectives are to allow gym owners to easily register members, manage memberships and payments, and track

attendance and membership plans. It also enables gym owners to quickly access member data, plans, and records, and to calculate discounts efficiently. The system features a clean and user-friendly interface for ease of use.

The project is built using the Java programming language and the BlueJ integrated development environment (IDE). It incorporates Object-Oriented Programming (OOP) concepts like classes, inheritance, and polymorphism to efficiently structure and manage data.

1.2. Tools Used

1.2.1. Java Language

Figure 1: Java Logo

Java is a versatile and widely-used programming language known for its platform independence, meaning code written in Java can run on any device with a Java Virtual Machine (JVM). It's an object-oriented language, making it easy to organize and manage complex software projects. Java boasts a robust standard library, strong security features, and a large community, making it a reliable choice for building enterprise-level applications, mobile apps, and webbased systems.

1.2.2. BlueJ IDE

Figure 2: BlueJ Logo

BlueJ is an integrated development environment (IDE) designed for beginners learning **Java**. It offers a simple interface, object visualization, interactive testing, basic code editing, debugging tools, and automatic class diagrams to help users understand object-oriented programming concepts easily and effectively.

1.2.3. Balsamiq

Figure 3: Balsamiq Logo

Balsamiq is a popular wireframing tool designed to help teams quickly create low-fidelity mock-ups of user interfaces. It allows users to sketch out their ideas and get everyone on the same page without getting bogged down in design details. Balsamiq's drag-and-drop interface makes it easy to create wireframes using pre-built UI components, icons, and templates. This tool is particularly useful for product managers, UX designers, developers, and consultants who need to communicate their ideas clearly and efficiently. Balsamiq wireframes are

designed to focus on functionality and user experience, helping teams iterate and refine their designs quickly.

2. Wireframe

This project also utilized **Balsamiq** for wireframing and designing the user interface. Balsamiq was used to create low-fidelity mock-ups of the Gym Management System, helping in visualizing the layout, structure, and navigation before implementing the actual GUI in **Java Swing**. It allowed for better planning of user interactions and improved the overall design process.

2.1. Wireframe of Home Page

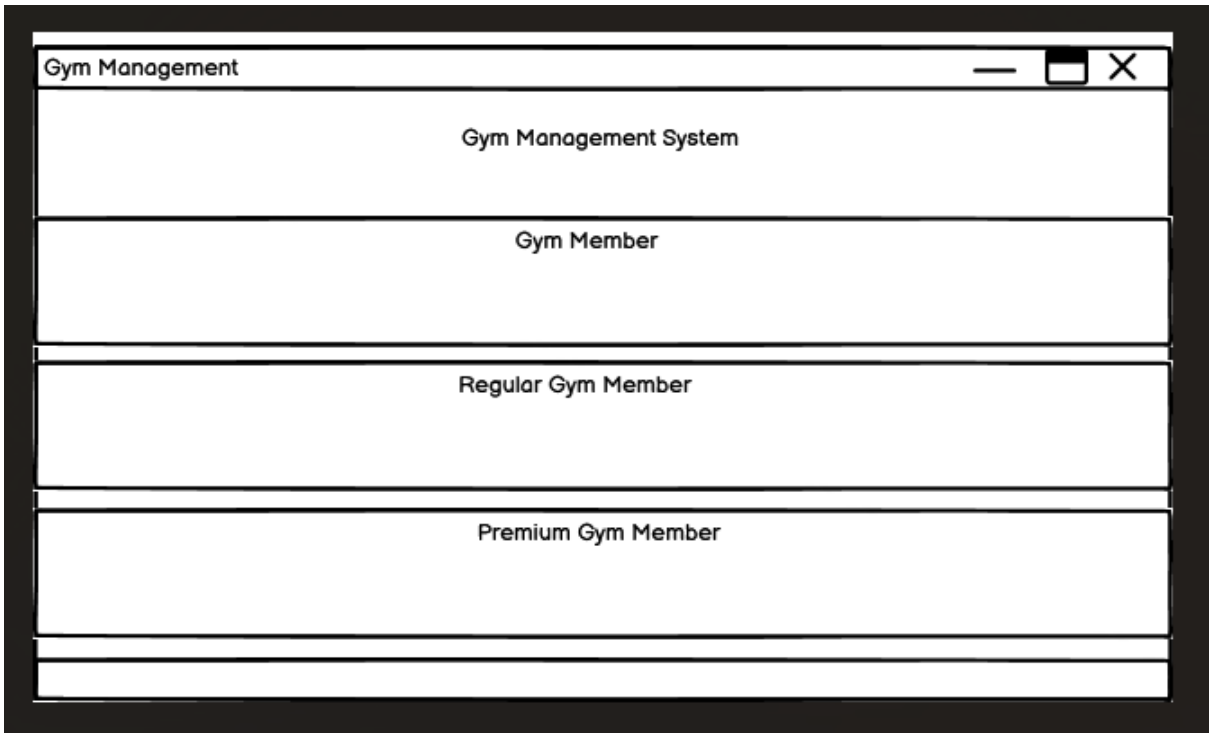


Figure 4: WIREFRAME OF HOMEPAGE

This wire represents the button arranged vertically which opens new GUI everytime they are pressed. They all correspond to different GUI like Gym Member GUI, Premium Member GUI and Regular Member GUI. It also has a title named as Gym Management System.

2.2. Wireframe of Registration of Regular Member

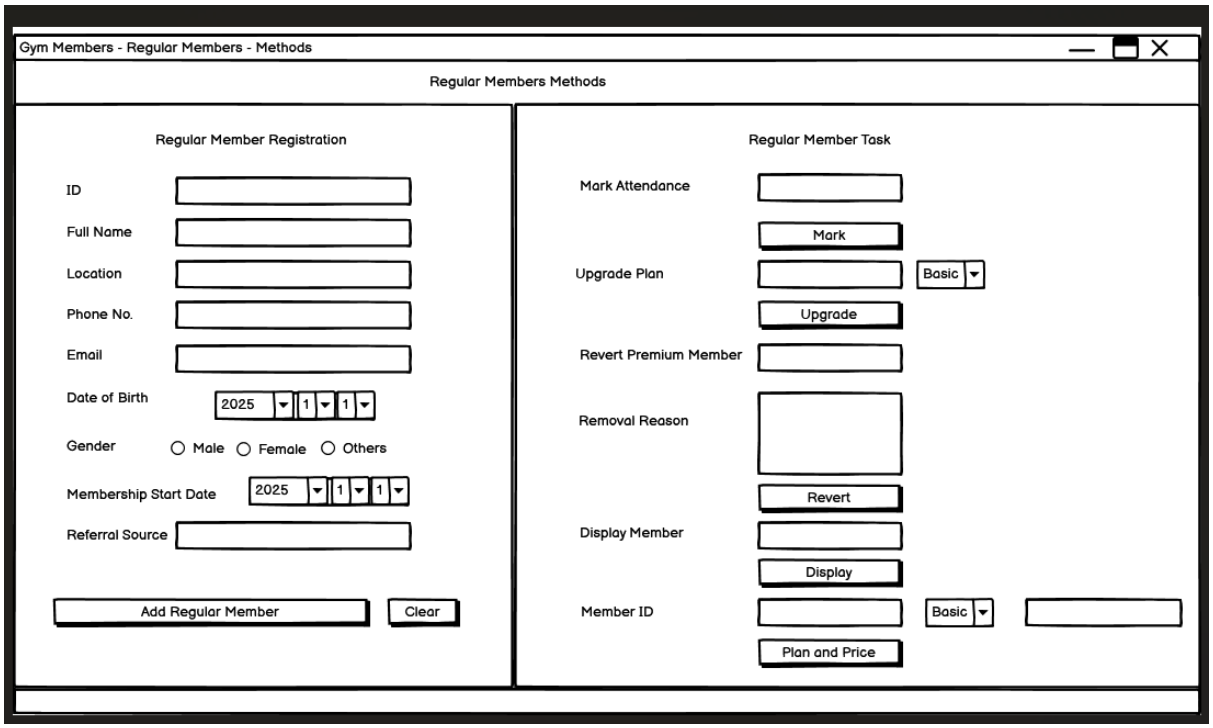


Figure 5: WIREFRAME OF REGULAR MEMBER REGISTRATION

The Regular Members Methods GUI is designed to streamline the management of regular gym members through a structured and user-friendly interface. It primarily focuses on three core areas: member registration, operational actions, and information display.

For **member registration**, the interface collects essential details such as the member's ID, full name, location, phone number, and email. It includes dropdown menus for selecting the year of birth and membership start date, with "2025" likely acting as a placeholder for dynamic year selection. Gender is captured using radio buttons (Male, Female, Others), ensuring straightforward input. A dedicated field for the referral source helps track how members discover the gym, which can be useful for marketing analysis.

The **actions section** allows staff to perform critical tasks. Users can add new members or clear the form to reset input fields, ensuring error-free data entry. Attendance marking is integrated to log gym visits, which may tie into loyalty programs or usage statistics. The "Upgrade Plan" feature enables switching membership tiers (Basic, Standard, Deluxe), reflecting flexibility in service

offerings. To handle member cancellations or downgrades, the "Revert Member" option includes a mandatory "Removal Reason" field, ensuring accountability and record-keeping for administrative purposes.

For **display and reporting**, the interface provides tools to view member details, including their current plan and associated pricing. This helps staff quickly access information during interactions or audits. The "Plan and Price" feature displays membership costs based on the selected tier, aiding transparency and decision-making. Overall, the GUI combines registration, activity tracking, and administrative functions into a cohesive system, simplifying daily operations for gym staff while maintaining organized records of member interactions and preferences.

2.3. Wireframe of Registration of Premium Member

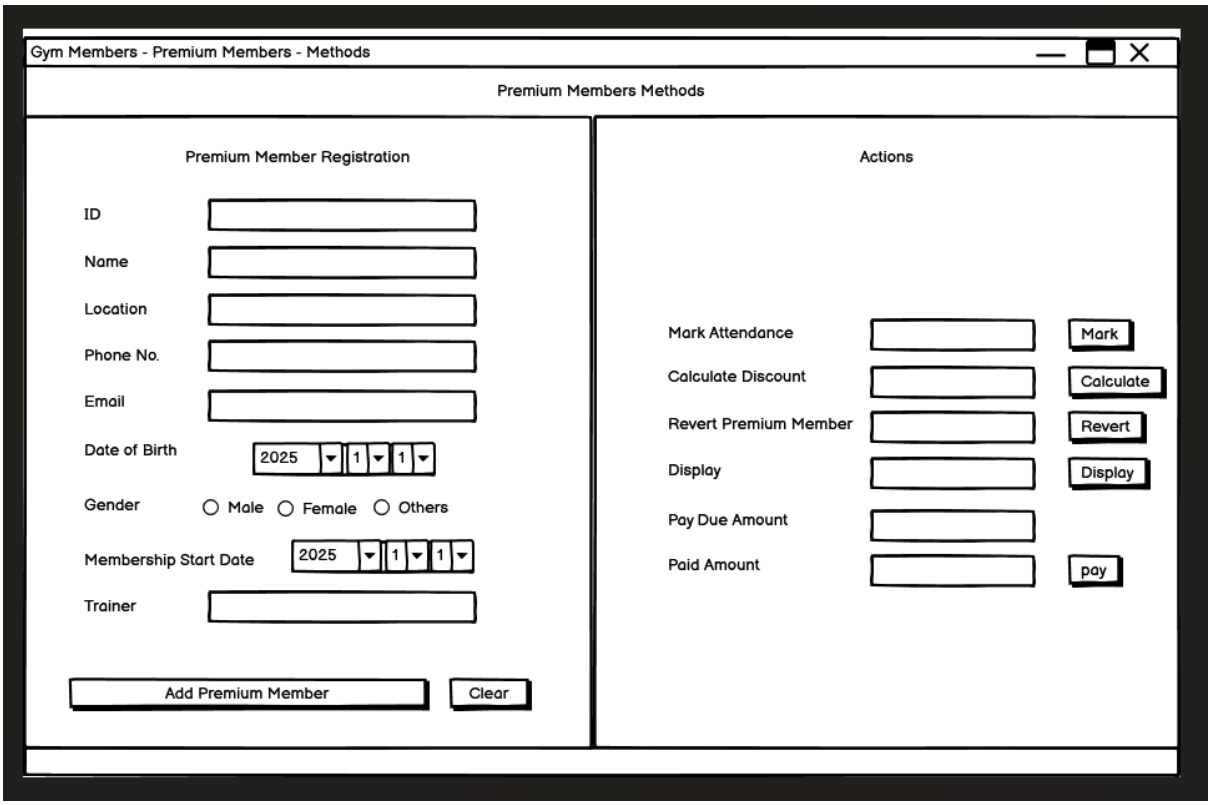


Figure 6: WIREFRAME OF REGISTRATION OF PREMIUM MEMBER REGISTRATION

The Premium Members Methods GUI is tailored to manage high-tier gym members with specialized features and financial tracking. The interface begins with a registration section that captures essential details like ID, name, location, phone number, and email, along with date of birth and gender selection. Unique to premium members, it includes a field to assign a personal trainer, emphasizing personalized service. The membership start date uses a structured input (year, month, day), likely with "2025" as a placeholder for dynamic selection. Staff can add new premium members or clear forms using dedicated buttons.

The actions section focuses on premium-specific operations: attendance tracking, discount calculations (likely for full payments), and membership reversion (downgrading or removal with associated financial adjustments). A prominent payment module allows staff to process due amounts and log paid balances, ensuring transparent financial management. The "Display" feature provides quick access to member profiles, including trainer assignments and payment histories. This GUI integrates fitness management with financial oversight, catering to the advanced needs of premium members while streamlining administrative tasks like payment processing and membership customization.

2.4. WireFrame of GymMember Methods

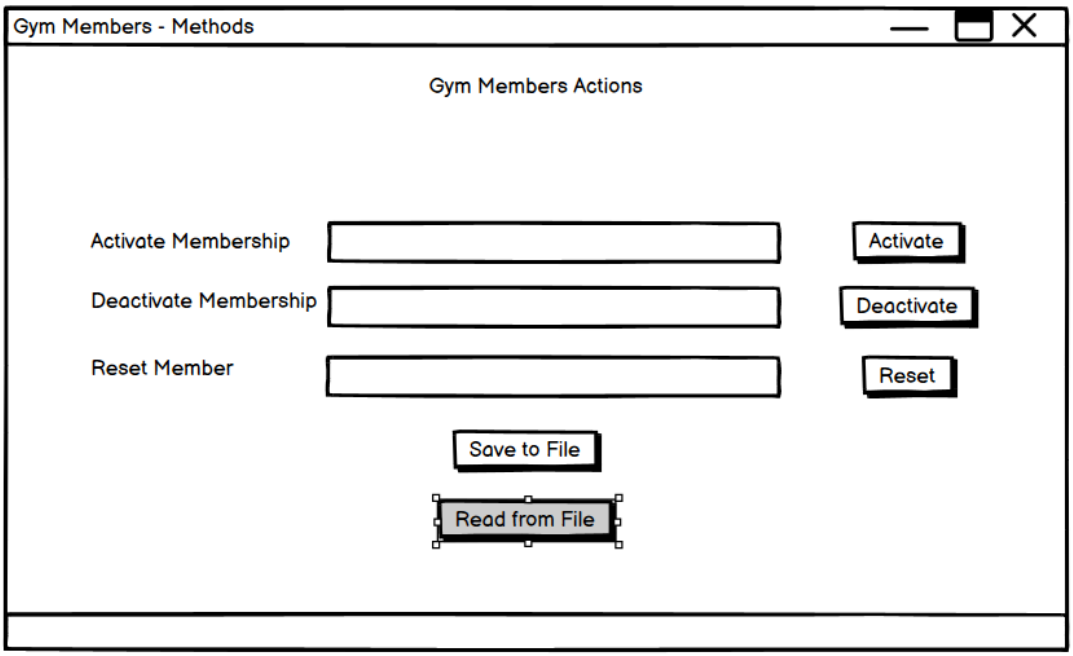


Figure 7: WIREFRAME OF GYM MEMBER METHODS

The Gym Members - Methods GUI is designed to centralize critical administrative tasks for managing member accounts and ensuring data integrity. This interface focuses on two primary functions: membership status management and data handling.

The membership status controls allow administrators to dynamically adjust a member's access to gym facilities. The "Active Membership" feature enables staff to activate new or lapsed accounts, granting members full access to gym services. Conversely, "Deactivate Membership" temporarily or permanently revokes access, useful for handling cancellations, frozen accounts, or disciplinary actions. The "Reset Member" option serves as a restorative tool, clearing attendance records, loyalty points, and other activity-based data while retaining core profile information. This is particularly helpful for members rejoining after a hiatus or for correcting account discrepancies without deleting historical data entirely.

For data management, the interface includes robust tools for preserving and retrieving member information. The "Save to File" function exports comprehensive member details—such as profiles, attendance logs, payment histories, and membership tiers—into a structured file format. This ensures backups for disaster recovery or compliance purposes. The "Read from File" feature allows administrators to import archived data, facilitating quick restoration of records during system updates or transitions. Together, these tools safeguard against data loss and streamline audits or reporting processes.

By consolidating these functions into a single panel, the GUI simplifies routine administrative workflows. Staff can swiftly toggle member statuses, reset accounts, or manage data persistence without navigating complex menus, reducing the risk of errors. This design prioritizes efficiency and clarity, ensuring gym operations remain organized and adaptable to changing member needs.

2.5. Wireframe of Regular Member Display Method

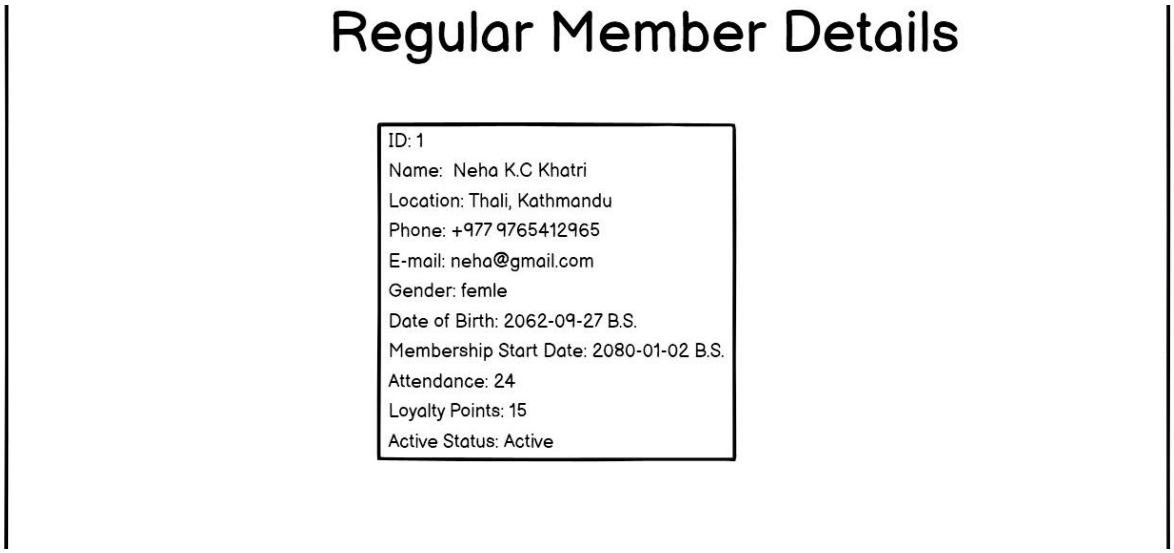


FIG: WIREFRAME OF REGULAR MEMBER DISPLAY METHOD

This wireframe represents the "Regular Member Display" page in a Gym Member Management system. The layout is designed to show the details of a regular gym member. The main section is titled "Regular Member Details" and contains a box displaying the member's information. The member's details include their ID, name, location, phone number, email address, gender, date of birth, membership start date, attendance count, loyalty points, and active status. This wireframe provides a clear and concise layout for displaying a regular gym member's profile, ensuring that all necessary details are easily accessible.

2.6. Wireframe of Premium Member Display Method

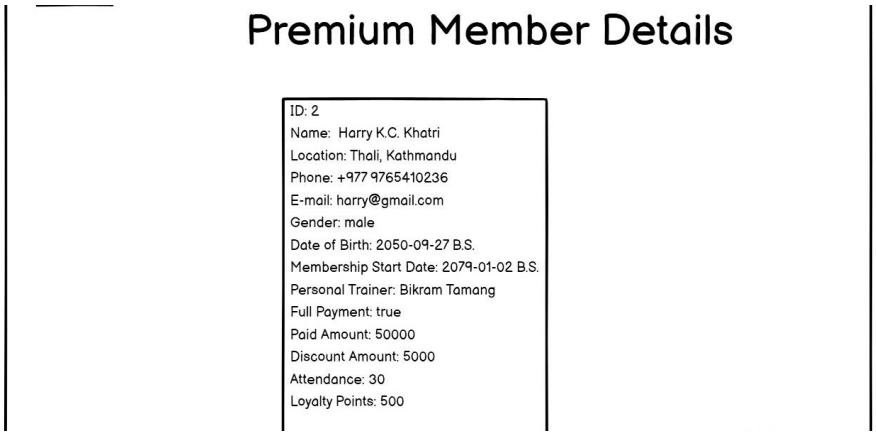


FIG: WIREFRAME OF PREMIUM MEMBER DETAILS

This wireframe represents the "Premium Member Display" page within a gym management system. The layout is designed to showcase detailed information about a premium member.

The main section of the page centrally displays the member's details in an organized manner. These details include the member's ID (2), full name (Harry K.C. Khatri), location (Thali, Kathmandu), phone number (+977 9765410236), email address (harry@gmail.com), gender (male), date of birth (2050-09-27 B.S.), and membership start date (2079-01-02 B.S.).

Additional information displayed includes the personal trainer's name (Bikram Tamang), confirmation of full payment, the paid amount (50000), the discount amount (5000), the attendance count (30), and the loyalty points accrued (500). This wireframe ensures that all necessary details of a premium gym member are easily accessible and well-organized.

3. Class Diagram - Overview

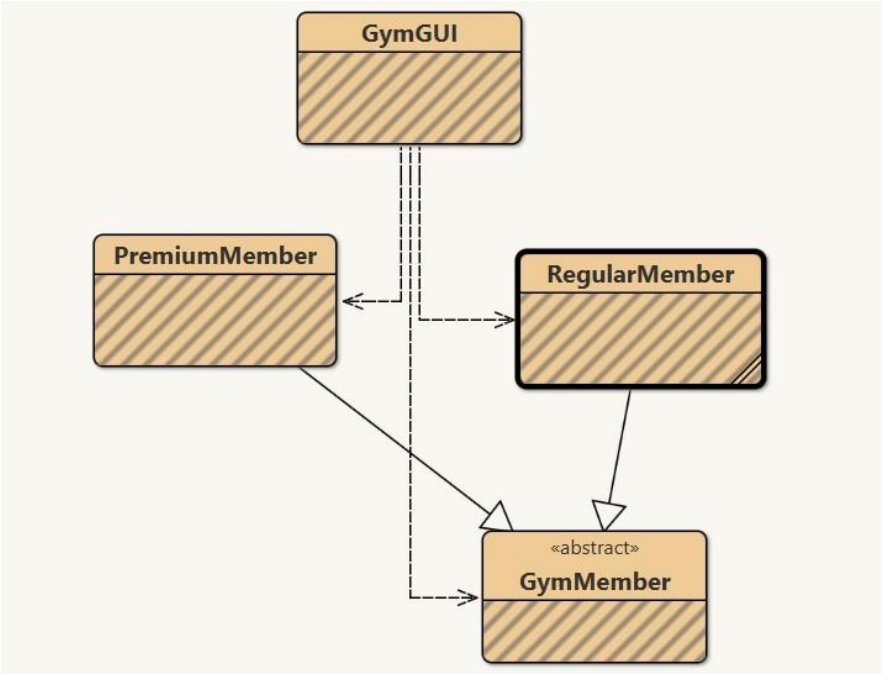


Figure 8: Class Diagram

3.1. GymMember Class (Abstract Class)

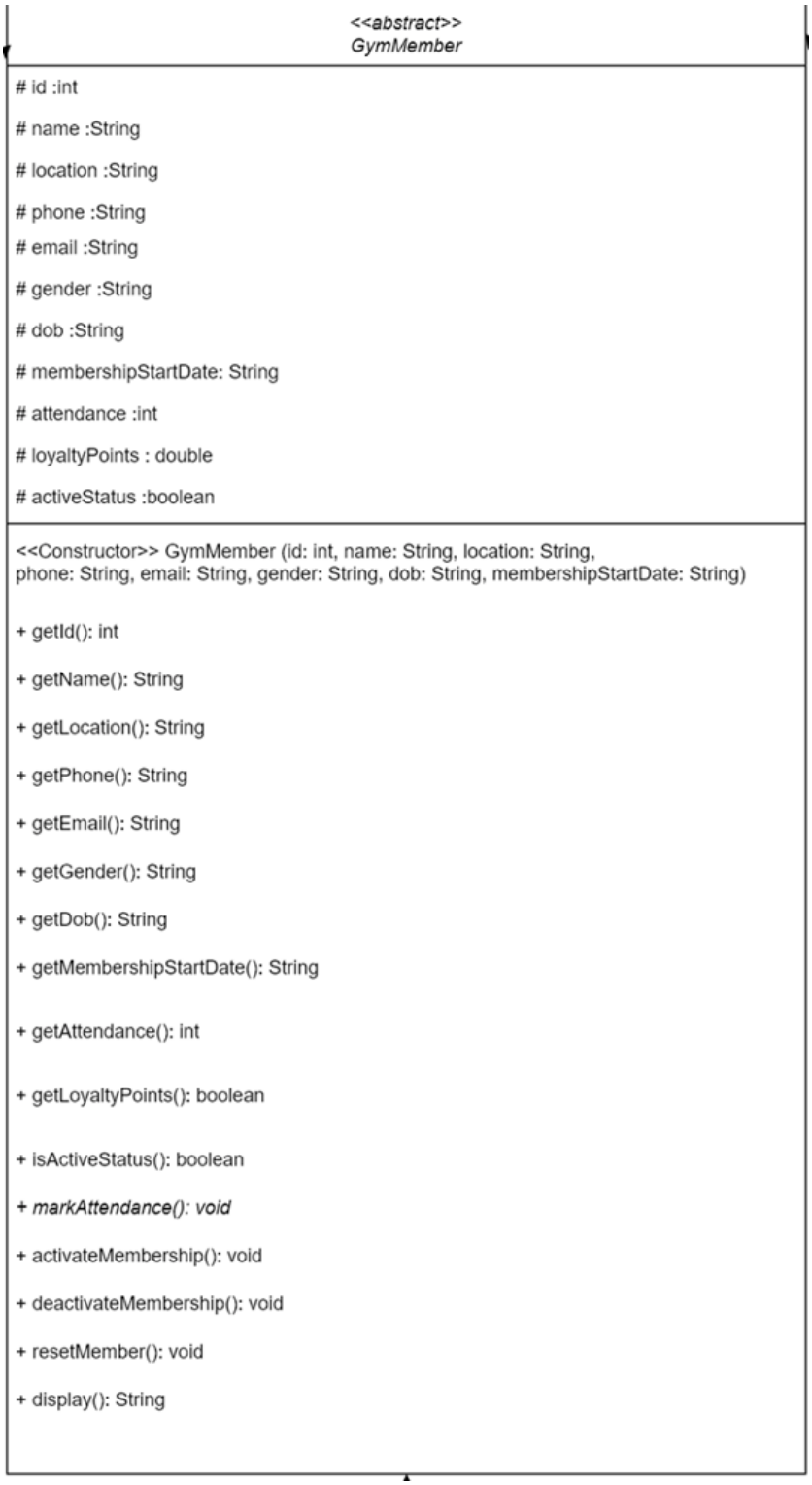


Fig: Class Diagram of GymMember class(abstract)

The GymMember class is an abstract class used in the Gym Management System to represent a general gym member. Being abstract, this class cannot be instantiated directly; instead, it must be extended by specific member types such as the RegularMember and PremiumMember classes. Since abstract classes cannot create objects on their own, objects must be created using one of its child classes. All attributes in the GymMember class are declared as protected, indicated by the “#” symbol in the class diagram, meaning they are accessible within the class and its subclasses. The constructor of the GymMember class initializes all member-related data at the time an object is created. To reduce code repetition in subclasses, accessor (getter) methods are defined for each attribute, enabling direct access to the data by child classes.

3.2. RegularMember (Subclass)

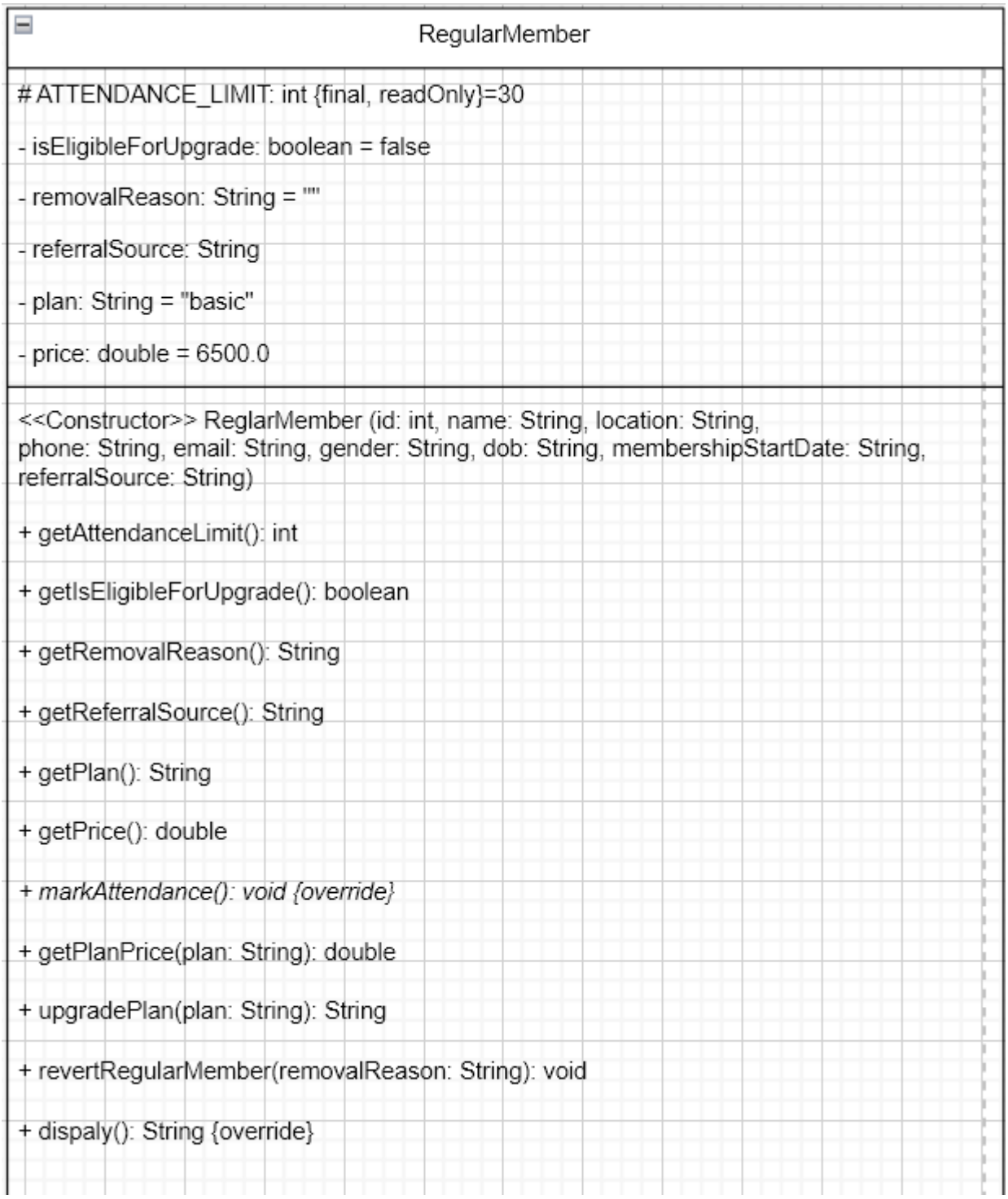


Fig: Class Diagram of RegularMember (Subclass)

The class diagram of RegularMember represents a concrete class which inherits from the GymMember class. It is the child class of GymMember. All the attributes are declared private, so they are denoted by “-”.

Attributes include an attendance limit of 30, which is also a final attribute, eligibility status for upgrade plans, removal reasons for cancellation of membership, referral source, membership plan type, and price of the plan. The constructor initializes all member details, also including attributes from GymMember, as the RegularMember class inherits the GymMember class while adding extra information for the referral source.

Since the class attributes are private, they are accessed through accessor or getter methods. These methods provide access to the private fields, including ways to retrieve attendance records, upgrade eligibility, removal reasons, referral sources, plan details, and price details.

In this class, important business logic is implemented through the markAttendance() method, which helps in tracking attendance of the member and is an overridden method, upgradePlan() for upgrading membership, revertRegularMember() to handle downgrades with specific removal reasons, and finally, display() to show the complete information of the member.

3.3. PremiumMember (Subclass)

Fig: Class Diagram of PremiumMember (Subclass)

The PremiumMember class diagram represents a concrete class that extends the GymMember class, making it a child class of GymMember. All attributes in this class are declared as private, indicated by the "-"

symbol. The attributes include a fixed premium charge of 50,000.0, along with personal trainer, payment status, paid amount, and discount amount. The constructor is responsible for initializing all member details, including the attributes inherited from the GymMember class, since PremiumMember is a subclass. Additionally, it initializes specific attributes related to the premium membership, such as the personal trainer information. Because the class attributes are private, they are accessed through accessor or getter methods. This class also includes several business rule methods, in payDueAmount() it handles the payment process, calculateDiscount() it calculates the discount, revertPremiumMember() it allows downgrading the membership from premium. This class override certain methods from GymMember as it is the child class of GymMember class like markAttendance() for tracking attendance, display() for displaying member information.

3.4. GymGUI (Main Class)

GymGUI

- pDisplayFrame, frame, gFrame, rFrame, pFrame: JFrame

- rPlanPriceButton, pDueCalculate, pMarkButton, pPayButton, pCalculateButton, pRevertButton, pDisplayButton, pAddButton, gButton, rButton, pButton, acButton, deacButton, resButton, rAddButton, rClearButton, rMarkButton, rUpgradeButton, rRevertButton, rDisplayButton, saveButton, readButton: JButton

- pMSDPanel, rGenderPanel, rDPanel, pDatePanel, fTPanel, fBPanel, gTPanel, gFPanel, rTPanel, rMPanel, rFPanel, rFTPanel, rFFPanel, rAPanel, rATPanel, rAFPanel: JPanel

- rPlanLabel, pDueLabel, pPaidAmtLabel, pMALabel, pPDALabel, pCDLabel, pRPMLabel, pDisplayLabel, pRRLabel, pTrainerLabel, pMSDLabel, pDobLabel, fTLabel, sTLabel, acLabel, deacLabel, resLabel, rTLabel, rFTLabel, rATLabel, rIdLabel, rFullNameLabel, rLocationLabel, rPhoneNumberLabel, rEmailLabel, rDobLabel, rGenderLabel, rMembershipStartDateLabel, rReferralLabel, rAMALabel, rAUPLabel, rARRMLabel, rRemovalLabel, rADeactivateLabel, rDisplayLabel: JLabel

- rPlanId, rPriceField, pDueIDField, pMAField, pPDAField, pPaidAmtField, pCDField, pRPMField, pDisplayField, pTrainerField, acField, deacField, resField, rIdField, rFullNameField, rLocationField, rPhoneNumberField, rEmailField, rReferralField, rAMAField, rAUPField, rARRMField, rDeactivateField, rDisplayField: JTextField

- rPlanPrice, rDYear, rDMonth, rDDay, rMembershipStartYear, rMembershipStartMonth, rMembershipStartDay, rPlan, pDYear, pDMonth, pDDay, pMSDYear, pMSDMonth, pMSDDay: JComboBox

- rMale, rFemale, rOthers pMaleRadio, pFemaleRadio, rOthersRadio : JRadioButton

- pDisplayArea, pRRArea, rRemovalArea; JTextArea

- rMSDPanel, pTPanel, pMPanel, pFPanel, pAPanel, pFTPanel, pFFPanel, pATPanel, pAFPanel, pGenderPanel: JPanel

- pTLabel, pFTLabel, pIdLabel, pNameLabel, pGenderLabel, pEmailLabel, pAddressLabel, pPhoneLabel, pATLabel: JLabel

- pIdField, pNameField, pAddressField, pEmailField, pPhoneField: JTextField

- pGenderGroup: ButtonGroup

- pRegisterButton, pClearButton, pAttendanceButton: JButton

<<Constructor>> GymGUI()

+ gMGui(): void

+ rMGui(): void

+ pMGui(): void

+ writeMembersToFile()ArrayList<GymMember> memberList: void

<u>+ main(String[] args): void</u>

Fig: Class Diagram of GymGUI (Main Class)

The GymGUI.java file is a Java Swing-based application designed to manage gym members through a graphical interface. It offers separate sections for general member management, regular members, and premium members, each with tailored features. Regular members can register with details like ID, contact information, and referral sources, while premium members include additional fields such as personal trainers. The interface uses panels, buttons, text fields, and drop-down menus to collect and display data, with color-coded layouts for visual clarity. Core functionalities include activating/deactivating memberships, marking attendance, upgrading plans, calculating discounts, and processing payments. Input validation ensures IDs are numeric and unique, with error messages displayed via pop-up dialogs for invalid inputs or missing fields.

The application integrates file operations to persist member data. It writes member details to MemberDetails.txt, formatting entries differently for regular and premium members. Regular member entries include referral sources and plan prices, while premium entries track discounts, paid amounts, and trainers. Data is loaded and displayed using a text area within a scrollable pane. The code assumes underlying classes like GymMember, RegularMember, and PremiumMember handle member-specific logic, leveraging inheritance and polymorphism. For example, methods like markAttendance() or calculateDiscount() behave differently based on whether the member is regular or premium, demonstrating object-oriented principles.

While functional, the system has limitations. It relies on a flat text file for storage, which may become inefficient for large datasets, suggesting a future shift to a database. Input validation could be expanded to enforce formats for emails, phone numbers, or dates. The

UI, though organized, lacks responsiveness for varying screen sizes. Additionally, the absence of the GymMember class definitions in the provided code limits insight into how member states or loyalty points are managed internally. Despite these areas for improvement, the application effectively combines Swing components, event-driven programming, and file I/O to create a modular and extensible gym management system.

3.5. Class Diagram of the System and their Relationship

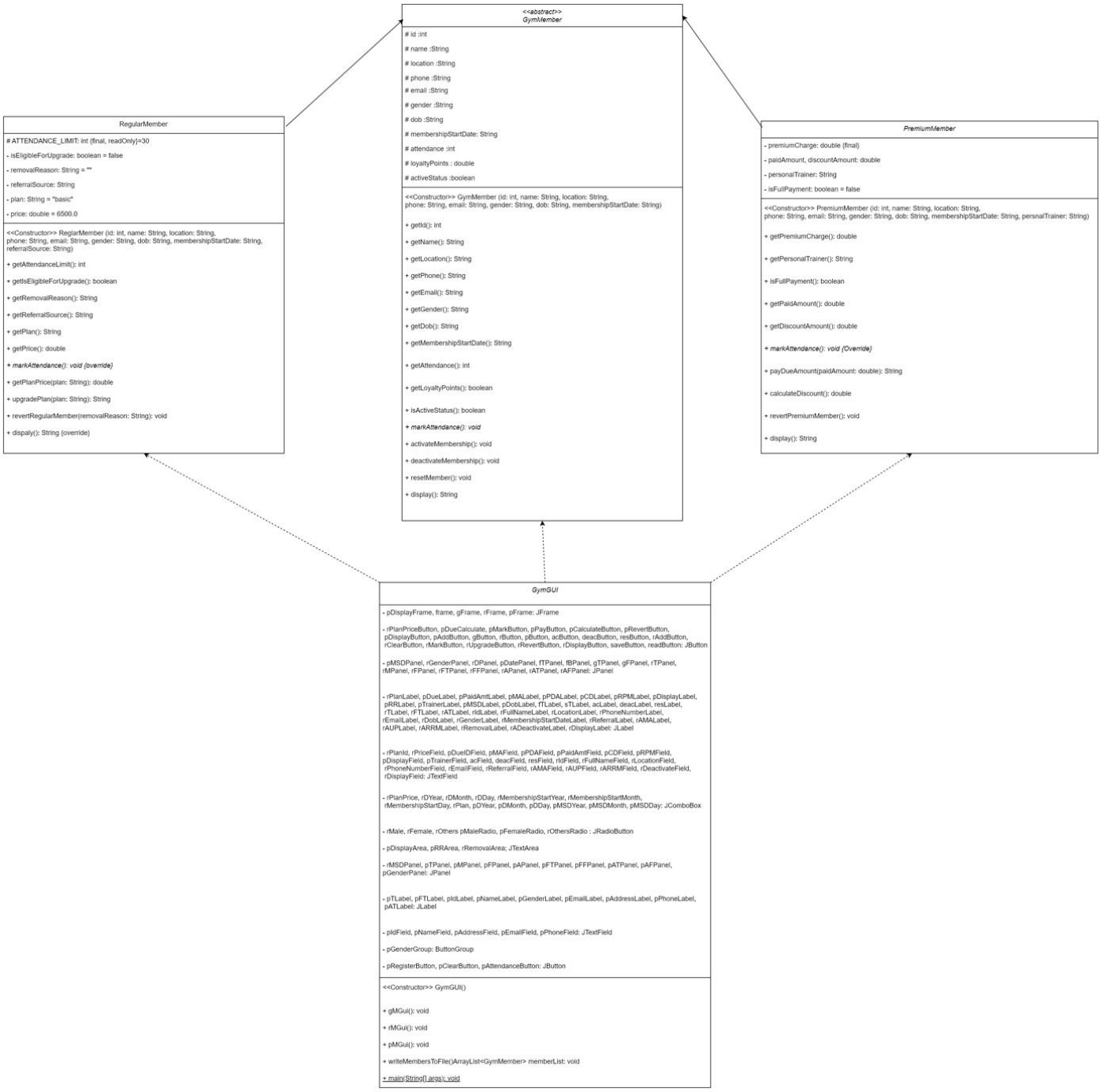


Fig: Class diagram of the System and their Relationship

The class diagram for the gym management system demonstrates a clear use of object-oriented principles such as abstraction, inheritance, and method overriding. At the core of the system is the abstract class GymMember, which acts as the parent class and defines all the common attributes and behaviours that apply to

both regular and premium gym members. It includes basic details like ID, name, contact information, gender, date of birth, and membership start date. This class also defines shared functionalities such as marking attendance, displaying member details, and resetting member data, although the markAttendance() method is abstract and must be implemented by child classes.

From this base class, two specific member types extend it: RegularMember and PremiumMember. These subclasses inherit the common properties from GymMember but add their own features as well. RegularMember introduces attributes like attendance limits, upgrade eligibility, plan types, prices, and referral sources. It includes logic to upgrade the membership plan if certain conditions are met, such as attendance reaching the required threshold. On the other hand, PremiumMember includes functionalities related to full payments, personal trainers, and discounts. It tracks how much the member has paid, whether they've completed full payment, and calculates a discount accordingly.

The GymGUI class plays a central role as the graphical interface controller. It is responsible for creating instances of RegularMember and PremiumMember and allowing the user to interact with them through buttons, forms, and other GUI components. This class accesses methods from all the other classes to register new members, display their details, mark attendance, manage payments, and perform upgrades or resets. In essence, the GUI acts as the bridge between the user and the underlying logic.

The relationships between the classes form a hierarchical inheritance structure, with GymMember at the top and the other two as its subclasses. This promotes code reusability and organization, allowing shared logic to exist in one place while subclass-specific behavior can be customized. Overall, the class design is well-structured, separating responsibilities cleanly between data handling and user interaction.

3.6. Features Implementation

- Member Registration: GUI forms for Regular/Premium members.
- Attendance Tracking: Mark attendance for members and track their gym visits.

- Plan Management: Upgrade/downgrade membership plans (Basic, Standard, Deluxe).
- Payment System: Calculate discounts (Premium) and process payments with due amount tracking.
- Membership Status: Activate, deactivate, or reset memberships.

3.7. Advanced Features

- Polymorphism: Overridden methods (markAttendance(), display()) for Regular/Premium members.
- Input Validation: Checks for unique IDs, numeric inputs, and invalid dates.
- GUI Navigation: Separate windows for Regular/Premium member operations.

4. GUI of the Gym Management Developed

4.1. Home GUI

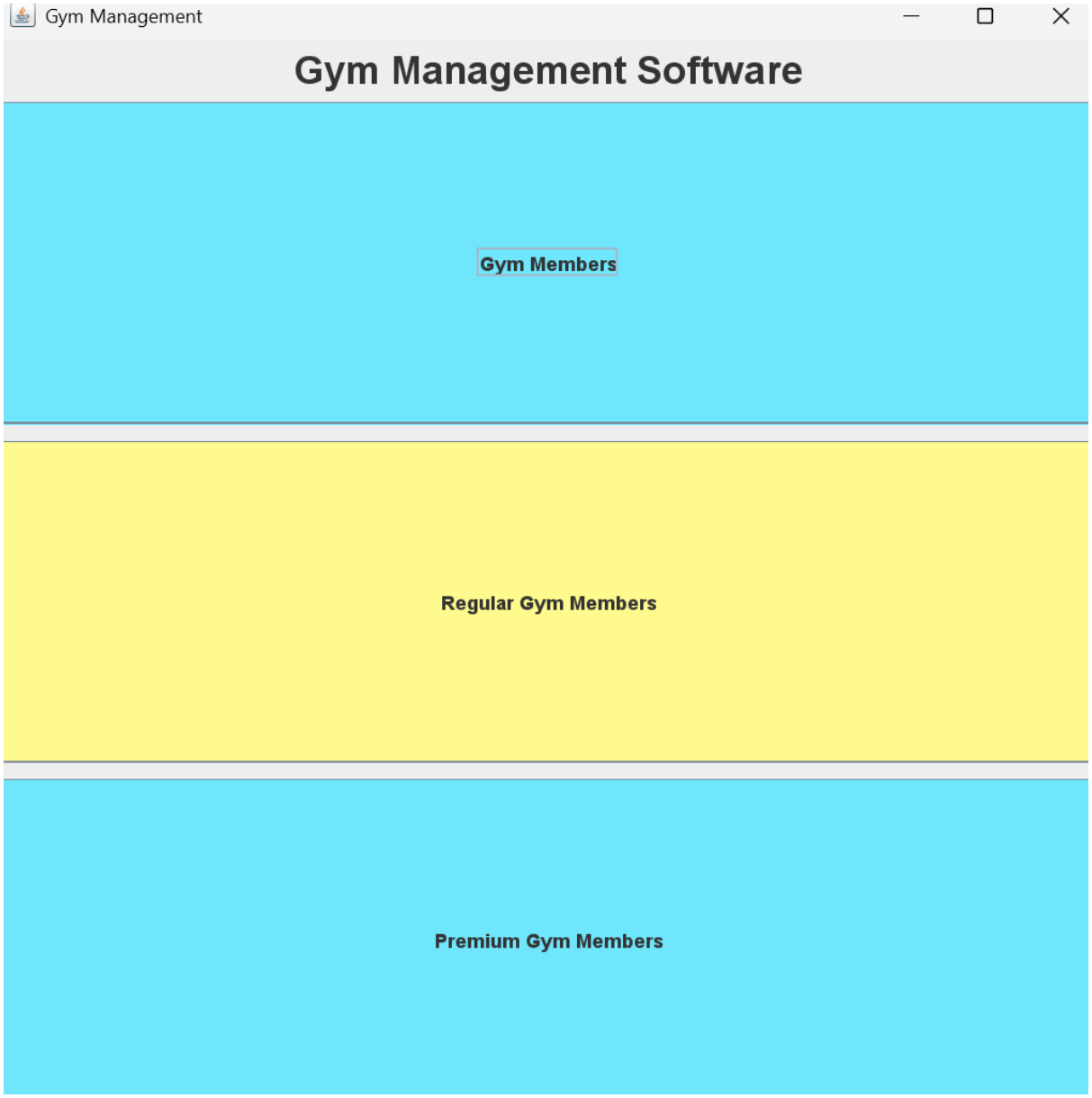


Figure 9: Gym Management Home Section

4.2. Gym Member GUI

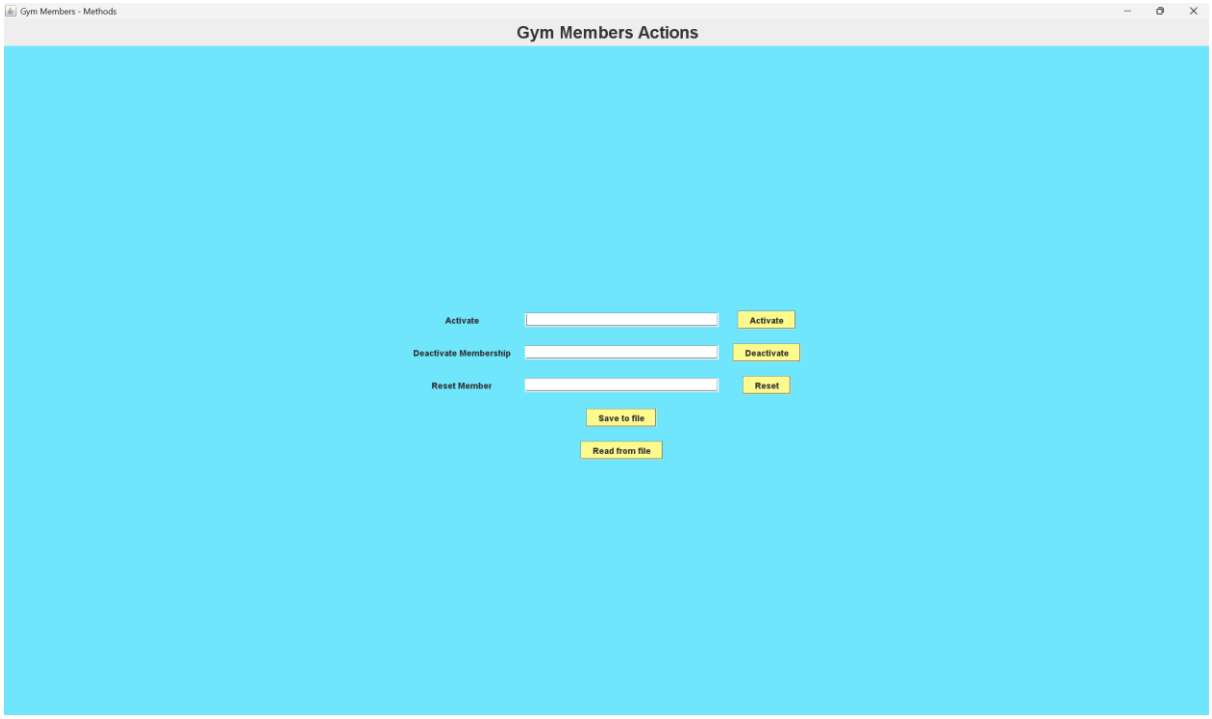


Figure 10: Gym Member GUI

4.3. Regular Member GUI

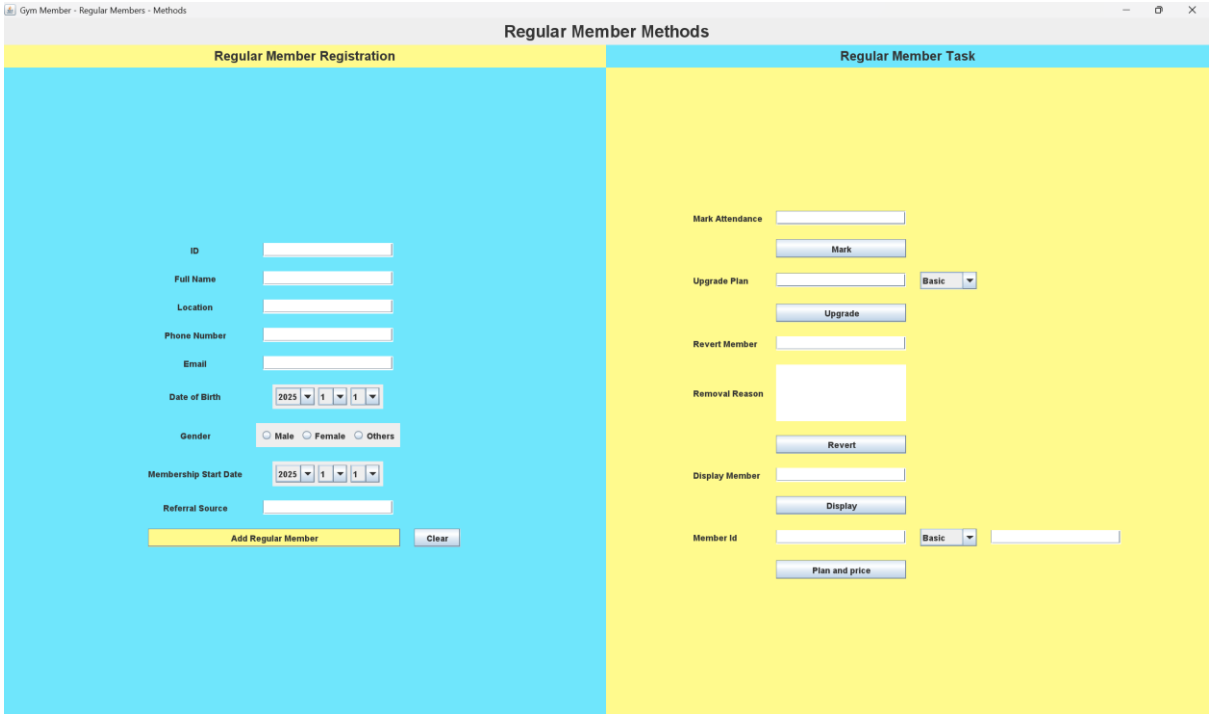


Figure 11: Regular Member GUI

4.4. Premium Member GUI

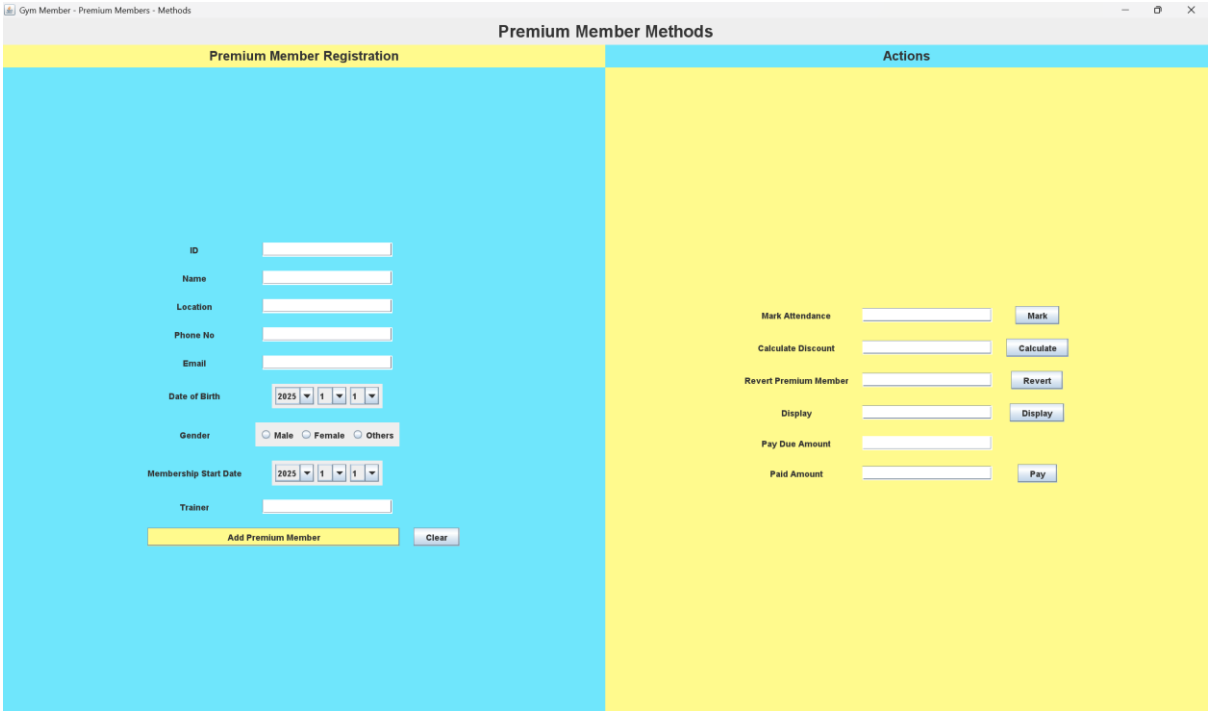


Figure 12: Premium Member GUI

5. Pseudocode

5.1. Pseudocode - GymMember.java

GymMember.java
CREATE an abstract parent Class GymMember DO DECLARE instance variable id as int using protected access modifier DECLARE instance variable name as String using protected access modifier DECLARE instance variable location as String using protected access modifier DECLARE instance variable phone as String using protected access modifier DECLARE instance variable email as String using protected access modifier DECLARE instance variable gender as String using protected access modifier DECLARE instance variable DOB as String using protected access modifier

```
DECLARE instance variable membershipStartDate as String using protected access modifier
DECLARE instance variable attendance as int using protected access modifier
DECLARE instance variable loyaltyPoints as double using protected access modifier
DECLARE instance variable activeStatus as boolean using protected access modifier

CREATE constructor GymMember(id, name, location, phone, email, gender, DOB, membershipStartDate)
DO
    SET instance variable id as id
    SET instance variable name as name
    SET instance variable location as location
    SET instance variable phone as phone
    SET instance variable email as email
    SET instance variable gender as gender
    SET instance variable DOB as DOB
    SET instance variable membershipStartDate as membershipStartDate
    SET instance variable attendance as 0
    SET instance variable loyaltyPoints as 0.0
    SET instance variable activeStatus as false
END DO

CREATE abstract method markAttendance() with return type void

CREATE instance method activateMembership() with return type void
DO
    SET activeStatus as true
END DO

CREATE instance method deactivateMembership() with return type void
DO
    IF activeStatus is true
        SET activeStatus as false
    ELSE
```

```
            DISPLAY "Membership has already been deactivated."
        END IF
    END DO

    CREATE instance method resetMember() with return type void
    DO
        SET activeStatus as false
        SET attendance as 0
        SET loyaltyPoints as 0.0
    END DO

    CREATE instance method display() with return type string
    DO
        SET result as ""
        APPEND "ID: " + id + "\n" to result
        APPEND "Name: " + name + "\n" to result
        APPEND "Location: " + location + "\n" to result
        APPEND "Phone: " + phone + "\n" to result
        APPEND "Email: " + email + "\n" to result
        APPEND "Gender: " + gender + "\n" to result
        APPEND "DOB: " + DOB + "\n" to result
        APPEND "Membership Start Date: " + membershipStartDate + "\n"
to result
        APPEND "Attendance: " + attendance + "\n" to result
        APPEND "Loyalty Points: " + loyaltyPoints + "\n" to result
        APPEND "Active Status: " + activeStatus + "\n" to result
        RETURN result
    END DO
END DO
```

5.2. Pseudocode - RegularMember.java

RegularMember.java
CREATE Class RegularMember EXTENDS GymMember DO DECLARE constant ATTENDANCE_LIMIT as int and assign 30

```
DECLARE instance variable isEligibleForUpgrade as boolean using private access modifier
DECLARE instance variable removalReason as String using private access modifier
DECLARE instance variable referralSource as String using private access modifier
DECLARE instance variable plan as String using private access modifier
DECLARE instance variable price as double using private access modifier

CREATE constructor RegularMember(id, name, location, phone, email, gender, dob, membershipStartDate, referralSource)
DO
    CALL parent constructor GymMember(id, name, location, phone, email, gender, dob, membershipStartDate)
    SET ATTENDANCE_LIMIT as 30
    SET isEligibleForUpgrade as false
    SET removalReason as ""
    SET referralSource as referralSource
    SET plan as "basic"
    SET price as 6500.0
END DO

CREATE method getAttendanceLimit() with return type int
DO
    RETURN ATTENDANCE_LIMIT
END DO

CREATE method getIsEligibleForUpgrade() with return type boolean
DO
    RETURN isEligibleForUpgrade
END DO

CREATE method getRemovalReason() with return type String
DO
    RETURN removalReason
END DO

CREATE method getReferralSource() with return type String
```

```
DO
   RETURN referralSource
END DO

CREATE method getPlan() with return type String
DO
   RETURN plan
END DO

CREATE method getPrice() with return type double
DO
   RETURN price
END DO

OVERRIDE method markAttendance() with return type void
DO
   IF activeStatus is true
      INCREMENT attendance by 1
      INCREMENT loyaltyPoints by 5
      IF attendance is greater than or equal to ATTENDANCE_LIMIT
         SET isEligibleForUpgrade as true
      END IF
   END IF
END DO

CREATE method getPlanPrice(plan) with return type double
DO
   CONVERT plan to lowercase
   SWITCH plan
      CASE "basic":
         SET price as 6500
         BREAK
      CASE "standard":
         SET price as 12500
         BREAK
      CASE "deluxe":
         SET price as 18500
```

```
            BREAK
        DEFAULT:
            DISPLAY "Invalid Plan Name:"
            DISPLAY "Choose among these: Basic, Standard, Deluxe"
            RETURN -1
    END SWITCH
    RETURN price
END DO

CREATE method upgradePlan(plan) with return type String
DO
    IF isEligibleForUpgrade is true
        IF plan equals current plan
            RETURN "You are already Subscribed to " + plan
        END IF
        SWITCH plan
            CASE "Basic":
                SET price as 6500
                SET plan as "Basic"
                BREAK
            CASE "Standard":
                SET price as 12500
                SET plan as "Standard"
                BREAK
            CASE "Deluxe":
                SET price as 18000
                SET plan as "Deluxe"
                BREAK
            DEFAULT:
                DISPLAY "Invalid plan selected."
                RETURN "Invalid plan selected"
        END SWITCH
        RETURN "Plan upgraded successfully to " + plan + " Price: " + price
    ELSE
        RETURN "You are not eligible for an upgrade"
    END IF
END DO
```

```
    CREATE method revertRegularMember(removalReason) with return type void
    DO
        CALL resetMember() from super class
        SET isEligibleForUpgrade as false
        SET plan as "basic"
        SET price as 6500
        SET this.removalReason as removalReason
    END DO

    OVERRIDE method display() with return type String
    DO
        SET memberInfo as CALL super.display()
        APPEND "\nPlan: " + plan + "\n" to memberInfo
        APPEND "Price: " + price to memberInfo
        IF removalReason is not null AND removalReason is not blank
            APPEND "\nRemoval Reason: " + removalReason to memberInfo
        END IF
        RETURN memberInfo
    END DO
END DO
```

5.3. Pseudocode - PremiumMember.java

PremiumMember.java
CREATE Class PremiumMember EXTENDS GymMember DO DECLARE constant premiumCharge as double and assign 50000 DECLARE personalTrainer as String using private access modifier DECLARE isFullPayment as boolean using private access modifier DECLARE paidAmount as double using private access modifier DECLARE discountAmount as double using private access modifier CREATE constructor PremiumMember(id, name, location, phone, email, gender, dob, membershipStartDate, personalTrainer) DO

```
        CALL parent constructor GymMember(id, name, location, phone, email,
gender, dob, membershipStartDate)
        SET premiumCharge = 50000
        SET this.personalTrainer = personalTrainer
        SET isFullPayment = false
        SET paidAmount = 0
        SET discountAmount = 0
    END DO

    CREATE method getPremiumCharge() with return type double
    DO
        RETURN premiumCharge
    END DO

    CREATE method getPersonalTrainer() with return type String
    DO
        RETURN personalTrainer
    END DO

    CREATE method isFullPayment() with return type boolean
    DO
        RETURN isFullPayment
    END DO

    CREATE method getPaidAmount() with return type double
    DO
        RETURN paidAmount
    END DO

    CREATE method getDiscountAmount() with return type double
    DO
        RETURN discountAmount
    END DO

    OVERRIDE method markAttendance() with return type void
    DO
        INCREMENT attendance by 1
```

```
    INCREMENT loyaltyPoints by 10
END DO

CREATE method payDueAmount(paidAmount) with return type String
DO
    IF isFullPayment is true
        RETURN "Full Amount is already paid."
    END IF

    IF this.paidAmount + paidAmount > premiumCharge
        RETURN "More balance paid than the premium charge"
    END IF

    INCREMENT this.paidAmount by paidAmount

    SET remainingAmount = premiumCharge - this.paidAmount

    IF remainingAmount == 0
        SET isFullPayment = true
        RETURN "Successfully Paid the Remaining Balance."
    END IF

    RETURN "Payment successful! Remaining Balance: " + remainingAmount
END DO

CREATE method calculateDiscount() with return type double
DO
    IF isFullPayment is true
        SET discountAmount = premiumCharge * 0.10
    ELSE
        SET discountAmount = 0
    END IF
    RETURN discountAmount
END DO

CREATE method revertPremiumMember() with return type void
DO
```

```
        CALL resetMember() from parent class
        SET personalTrainer = ""
        SET isFullPayment = false
        SET paidAmount = 0
        SET discountAmount = 0
    END DO

    OVERRIDE method display() with return type String
    DO
        RETURN result of super.display()
        + "\nPersonal Trainer: " + personalTrainer
        + "\nPaid Amount: " + paidAmount
        + "\nFull Payment: " + isFullPayment
        + "\nDiscount Amount: " + discountAmount
        + "\nRemaining Charge: " + (premiumCharge - getPaidAmount())
    END DO
END DO
```

5.4. Pseudocode - GymGUI.java

```
GymmGUI.java
Program GymManagementSystem:
    Initialize GymGUI:
        Create main frame with title "Gym Management"
        Set frame size to 700x700
        Set layout to BorderLayout

        Do Create Title Panel:
            Add label "Gym Management Software" with large font
        End Do

        Do Create Button Panel:
            Add "Gym Members" button (light blue) which triggers gMGui()
            Add "Regular Members" button (yellow) which triggers rMGui()
            Add "Premium Members" button (light blue) which triggers pMGui()
        End Do
```

```
    Display main frame
End Initialize

Method gMGui (General Member Actions):
    Create general members frame
    Set layout to BorderLayout

    Do Create Activation Section:
        Text field for member ID
        "Activate" button which validate ID and activate member
        "Deactivate" button which validate ID and deactivate member
        "Reset" button which validate ID and reset member
    End Do

    Do Create File Operations:
        "Save to file" button which writeMembersToFile()
        "Read from file" button which display memberDetails.txt
    End Do
End Method

Method rMGui (Regular Members):
    Create regular members frame
    Set layout to GridLayout

    Do Create Registration Panel:
        Input fields: ID, Name, Location, Phone, Email
        Date pickers: DOB, Membership Start Date
        Radio buttons for gender
        Referral source field
        "Add" button which validate inputs and create RegularMember
        "Clear" button which reset all fields
    End Do

    Do Create Actions Panel:
        Attendance marking section with ID input
        Plan upgrade selector with price display
        Revert member section with reason text area
```

```
        Member display feature with ID input
    End Do
End Method

Method pMGui (Premium Members):
    Create premium members frame
    Set layout to GridLayout

    Do Create Registration Panel:
        Input fields: ID, Name, Location, Phone, Email, Trainer
        Date pickers: DOB, Membership Start Date
        Radio buttons for gender
        "Add" button which validate inputs and create PremiumMember
        "Clear" button which reset all fields
    End Do

    Do Create Actions Panel:
        Attendance marking section
        Discount calculation with ID input
        Payment processing section with amount input
        Revert to regular member feature
        Member display feature
    End Do
End Method

Function writeMembersToFile(memberList):
    Open/Create MemberDetails.txt
    Write formatted header
    For each member in memberList:
        If RegularMember:
            Write regular member data with plan details
        Else If PremiumMember:
            Write premium data with payment info
    End For
    Display success message
End Function
```

```
Main Program:
   Instantiate GymGUI
End Program

Event Handling Pseudocode (Example):
   On Button Click "Add Regular Member":
      Do:
         Validate all fields filled
         Check ID uniqueness
         Create RegularMember object
         Add to members list
         Show success message
      End Do

   On Button Click "Mark Attendance":
      Do:
         Validate numeric ID input
         Find member in list
         Check membership type
         Update attendance if active
         Show appropriate feedback
      End Do

   On Button Click "Save to File":
      Do:
         Call writeMembersToFile()
         Handle file exceptions
      End Do
```

6. Method Description

6.1. GymMember.java

- getId() : It retrieve the member's unique identifier. It's return type is int. It returns id of the GymMember instance.

- getName() : It helps to retrieve the member's full name. It's return type is String. It returns name stored in the object.

- getLocation() : It helps to retrieve the member location or address. It's return type is String. It returns the location attribute.

- getPhone() : It helps to retrieve member's contact number. It's return type is String. It returns phone value.

- getEmail() : It retrieve the member's email address. It's return type is String. It returns the email attribute.

- getGender() : It retrieve the member's gender. It's return type is String. It returns gender value as male or female or others.

- getDOB() : It retrieve the member's Date of Birth. It's return type is String. It returns the Date of Birth value of the object stored in the file.

- getMembershipStartDate() : It retrieve the member's date when the member started the membership. It's return type is String. It returns the membershipStartDate attribute.

- getAttendance() : It retrieve the member's attendance count. It's return type is int. It returns the attendance value.

- getLoyalityPoints() : It helps to retrieve the member's loyaltyPoints. It's return type is double. It returns the loyalyPoints value.

- getActiveStatus() : It helps to retrieve the active status of membership to chek if the membership is active or not. It's return type is Boolean. It returns true if the membership is active and false if not.
- GymMember():

  This method serves as the class initializer which will automatically called while creating object using new keyword. It has no return type not even void. In this method default values for attendance is set as 0, loyaltyPoints is set as 0.0, and activeStatus is set as false.
- GetId():

  This method helps to

- markAttendance():

  This method is declared as abstract method to enforce implementation in subclasses. It's return type is void which means it doesn not return any value. It allows different attendance logic according to the members type.

- activateMembership():

  This method modifies activeStatus from false to true. It is typically triggered by activate button in GUI. It doesnot have any condition except valid id else membership can directly be activated.

- deactivateMembership():

  This method modifies activeStatus from true to false. It checks for one condition for using deactivate method activeStatus should be true at first if the activeStatus is already false it will display" Membership has been already deactivated". When clicked "deactivate" button in GUI this method will trigger.

- resetMember():

  This method helps to resets all progress reset values like attendance, loyaltyPoints, activeStatus. When reset button is clicked in GUI this

resetMember() method will be triggred and will reset all progress to default values. It is used for membership cancellation.

- display():
  It prints all the member details in structured format. It provides a human-readable member summary. Its return type is String which means it returns String value.

6.2. RegularMember.java

- getAttendanceLimit() : It helps to retrieve the attendance limit. It's retuen type is int. It returns the attendance of the member.

- getIsELigibleForUpgrade() : It helps to check if the member is eligible for upgrade or not. It's return type is Boolean.

- getRemovalReason() : It helps to retrieve the reason for the cancellation or downgrading of member's plan. It's return type is String.

- getReferralSource() : It helps to retrieve the source from where members has know or listened about our gym. It's return type is String.

- getPlan() : It helps to retrieve the current plan subscription of the member. It's return type is String.

- getPrice() : It helps to retrieve the cost of the current plan subscription of member. It's return type is double.
- RegularMember():
  Initializes objects when created with 'new' keyword. Inherits parent class

variables via super(). Sets default values: isEligibleForUpgrade=false, plan="basic", price=65000.0, removalReason="".

- getPlanPrice(String Plan):
  Returns plan price as double. Uses switch-case to determine price based on selected plan.

- markAttendance():
  Overrides parent's abstract method. Increments attendance by 1 and loyaltyPoints by 5. Returns void.

- UpgradePlan(String newPlan):
  Returns String message about upgrade status. Handles plan upgrades between basic/standard/deluxe. Triggered by GUI upgrade button.

- revertRegularMember():
  Resets all values to defaults: calls parent's resetMember(), sets isEligibleForUpgrade=false, plan="basic", price=6500.0, removalReason="". Triggered by "revert" button.

- display():
  Returns String. Extends parent's display() using super. Adds plan/price details. Handles empty removalReason with special message.

6.3. PremiumMember.java

- getPremiumCharge() : It helps to retrieve the fixed premium membership amount. It’s return type is double.

- getPersonalTrainer() : It helps to retrieve the name of the personal trainer assigned to the member.

- getIsFullPayment() : It helps to retrieve the fee if the member has cleared all payment or not. It’s return type is Boolean.

- getPaidAmount() : It helps to retrieve the total amount the member has paid from the final premium charge. It’s return type is double.

- getDiscountAmount() : It helps to retrieve the discount applied after member has fully paid the fee. It’s return type is double.

- PremiumMember():
  Initializes objects with super() inheritance. Sets defaults: personalTrainer="", paidAmount=0.0, isFullPayment=false, discountAmount=0.0.

- markAttendance():
  Overrides parent method. Increments attendance by 1 and loyaltyPoints by 10. Returns void.

- payDueAmount(double paidAmount):
  Returns String. Calculates payment status (full/partial) and remaining balance.

- calculateDiscount():
  Returns String. Computes discount amount when full payment is made.

- revertPremiumMember():
  Resets all values: calls resetMember(), sets personalTrainer="", isFullPayment=false, paidAmount=0.0, discountAmount=0.0. Triggered by "revert" button.

- display():
  Returns String. Extends parent's display() via super. Shows trainer info, payment status, and amounts. Includes discount calculation for full payments.

7. Error Detection and Correction

1. Syntax Error

A syntax error occurs when the code violates the grammatical rules of the programming language. Since the compiler or interpreter checks for proper structure before execution, this type of error is detected at compile-time. Common examples include missing semicolons, unmatched parentheses, or incorrect keyword usage. Until syntax errors are fixed, the program cannot run.

Syntax Error Detection	`public GymGUI() {`
	`  frame = new JFrame("Gym Management");`
	`  frame.setBounds(0, 0, 700, 700);`
	`  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);`
	`  frame.setBackground(Color.lightGray);`
	`  frame.setLayout(new BorderLayout())`

Error Handling	• Add semicolon after frame.setLayout(new BorderLayout());
	`public GymGUI() {`
	`  frame = new JFrame("Gym Management");`
	`  frame.setBounds(0, 0, 700, 700);`
	`  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);`
	`  frame.setBackground(Color.lightGray);`
	`  frame.setLayout(new BorderLayout());`

2. Runtime Error

A runtime error arises during the execution of a program, after it has been compiled successfully. These errors occur due to unexpected conditions, such as division by zero, invalid memory access, or file not found exceptions. Unlike syntax errors, runtime errors do not prevent compilation but cause the program to crash or behave unexpectedly while running.

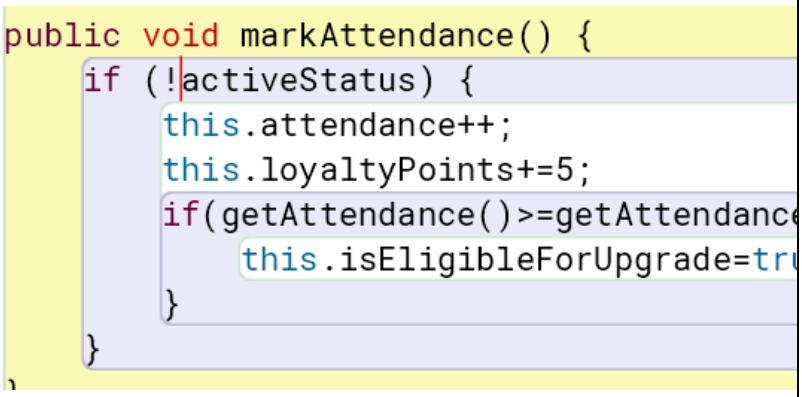
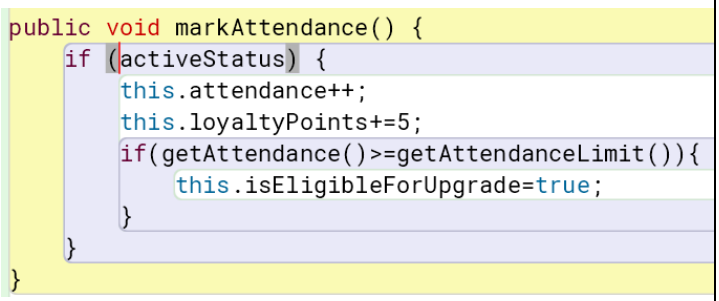
Runtime Error Detection	• If the user enters something like "101" then it parses. • If the user enters "abc" or "10a" while filling id field then throws NumberFormatException and is caught and handled with a message (no crash).
Error Handelling	• By adding NumberFormatException the runtime error is successfully handled.

```java
acButton.setBackground(yellow);
acButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            int memberId = Integer.parseInt(acField.getText().trim());
            boolean idisThere = false;

            for (GymMember member : members) {
                if (member.getId() == memberId) {
                    member.activateMembership();
                    idisThere = true;
                    JOptionPane.showMessageDialog(null, "Member Activated Successfully", "Success", JOptionPane.PLAIN
                    break;
                }
            }
            if (!idisThere) {
                JOptionPane.showMessageDialog(null, "Member Id not not Registered!!!", "Error", JOptionPane.ERROR_MES
            }
            // You can add your logic here, like searching in the ArrayList and activating a member
        } catch (NumberFormatException exception) {
            JOptionPane.showMessageDialog(null, "Enter a valid ID", "Error", JOptionPane.ERROR_MESSAGE);
        }
    }
});
```

3. Logical Error

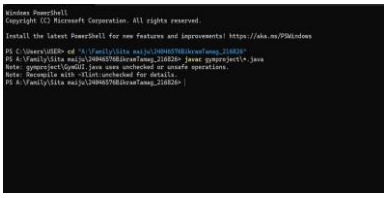
A logical error happens when the program runs without crashing but produces incorrect results due to flawed reasoning in the code. These errors are often the hardest to detect because the program executes normally, but the output does not match expectations. Identifying logical errors requires thorough testing, debugging, and careful analysis of the program's logic.

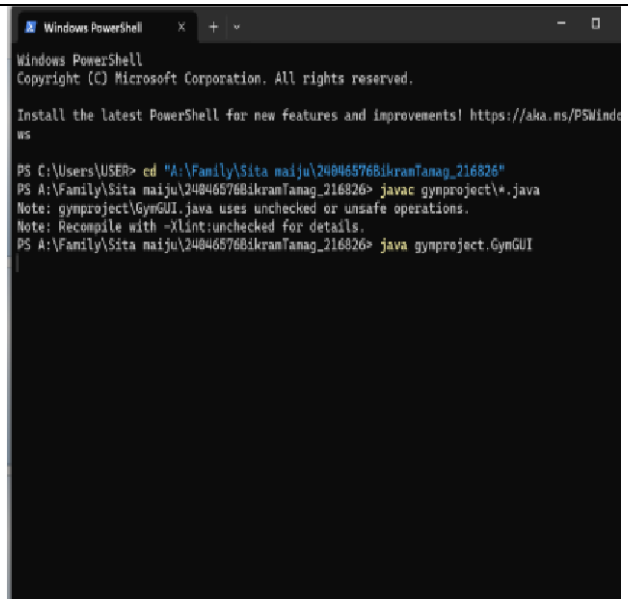
Logical Error Detection	
Error Handelling	• Remove exclamation mark before activeStatus to correct the logic.  •

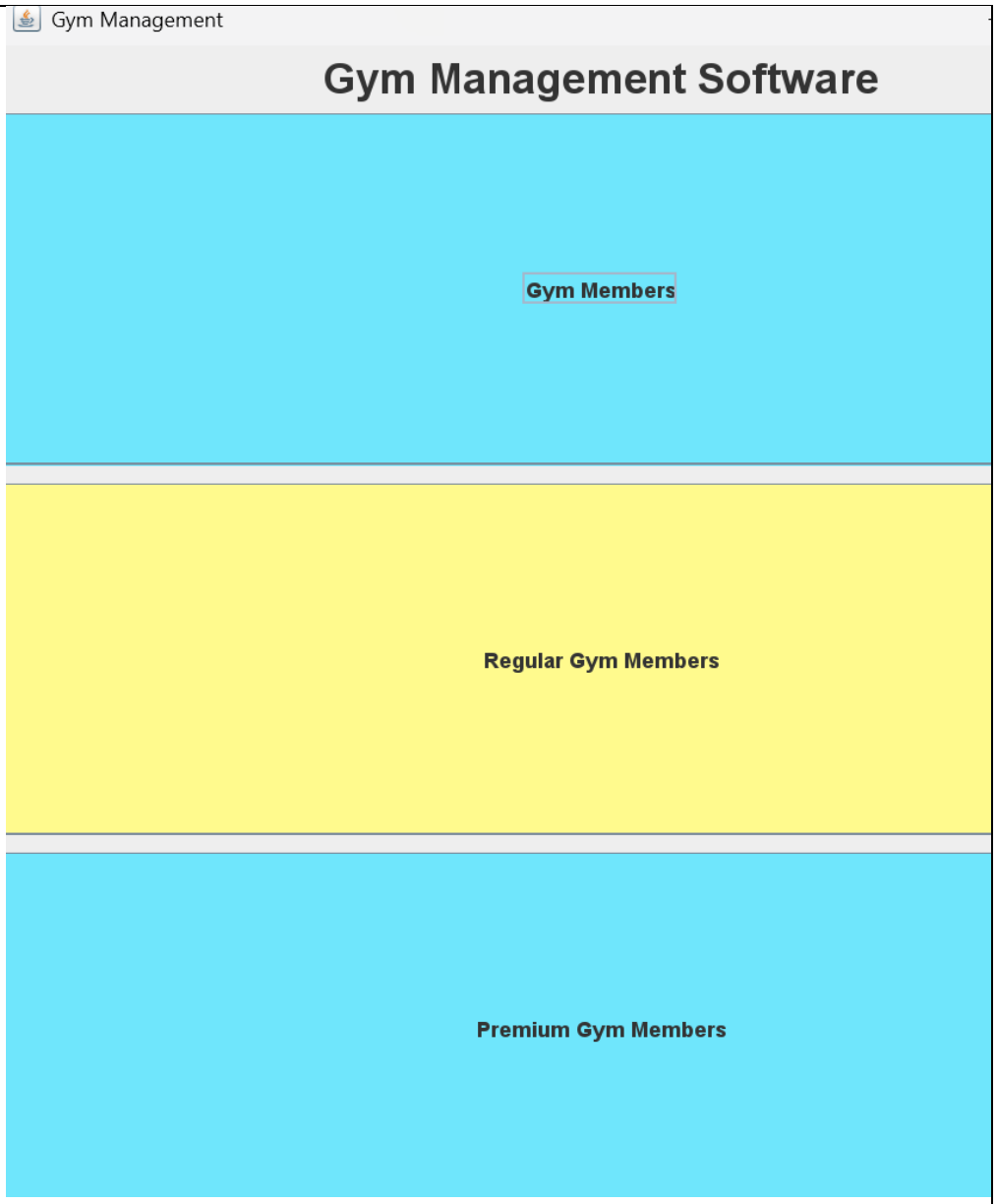
8. Testing

8.1. Test 1

Test	1
Objective	To test File Compilation.
Objective	To compile all the files using command prompt / terminal.
Activity	• Opened terminal and write javac *. Java. the as a whole i.e. GymGUI.java, GymMember.Java,

	• RegularMember.java and PremiumMember.java or say package as a whole.
Expected Output	Package should be compiled and the GymGUI.java's GUI should be displayed.
Actual Output	Leading to the correct directory  Whole package compiled  GymGUI.java class ran successfully

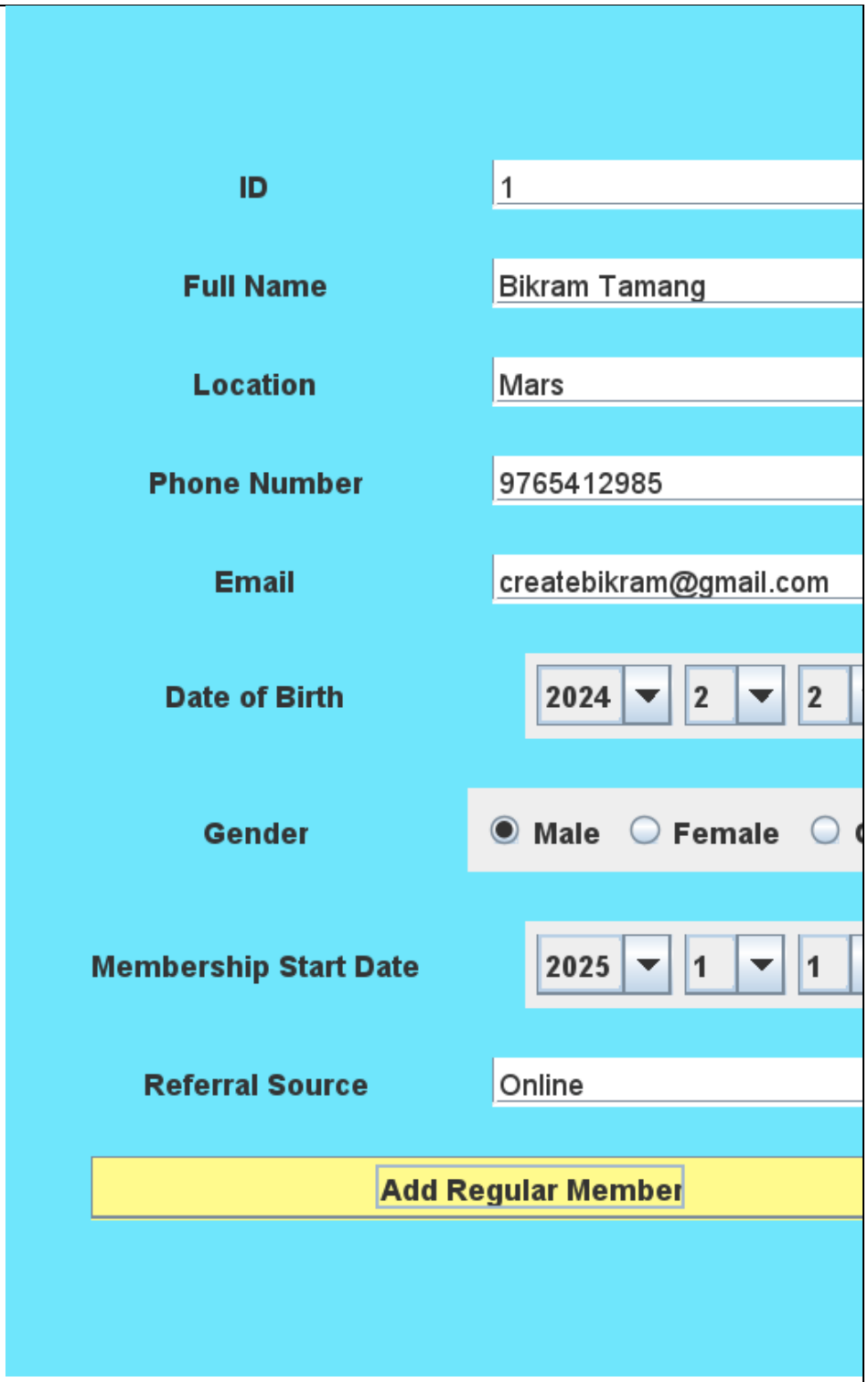
	 GymGUI.java's GUI was opened. Package was compiled successfully and GymGUI.java'sGUI was displayed.

	
Conclusion	Test was successful.

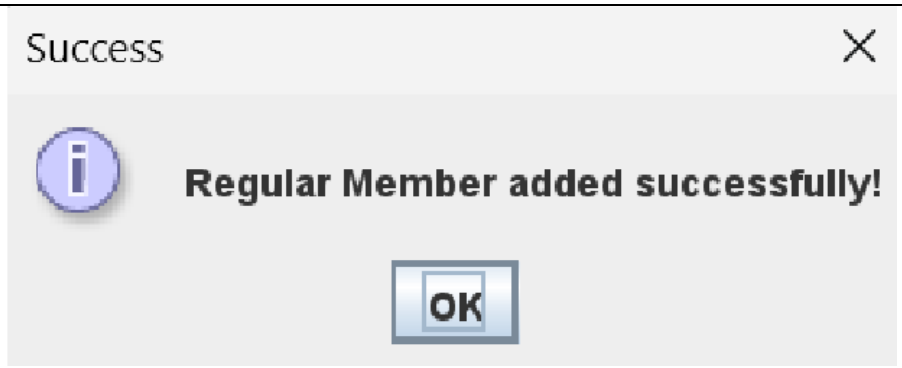
8.2. Test 2 For Regular Member

Test	2
Objective	To test the functionality of “Add Regular Member” Button

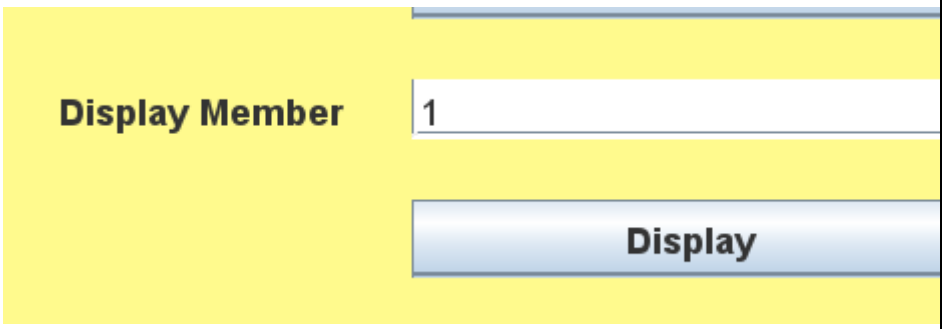
Activity	• If any field is left empty, the system displays an error message. • If all fields are filled with valid input and the “Add Member” button is clicked, the member is added.
Expected Output	• If any field is left empty, an error message was displayed. • If all fields are filled but contain invalid input, an error message is displayed. • If all fields are filled with valid input a confirmation message “Regular Member was added successfully” is displayed. Member information appears in the member list.
Actual Output	• Error message displayed when not all fields were filled. • Error message displayed when all fields were filled but the ID was invalid. • When all fields were filled with valid input Confirmation message “Regular Member added successfully” was displayed. • Valid input was filled in all fields.

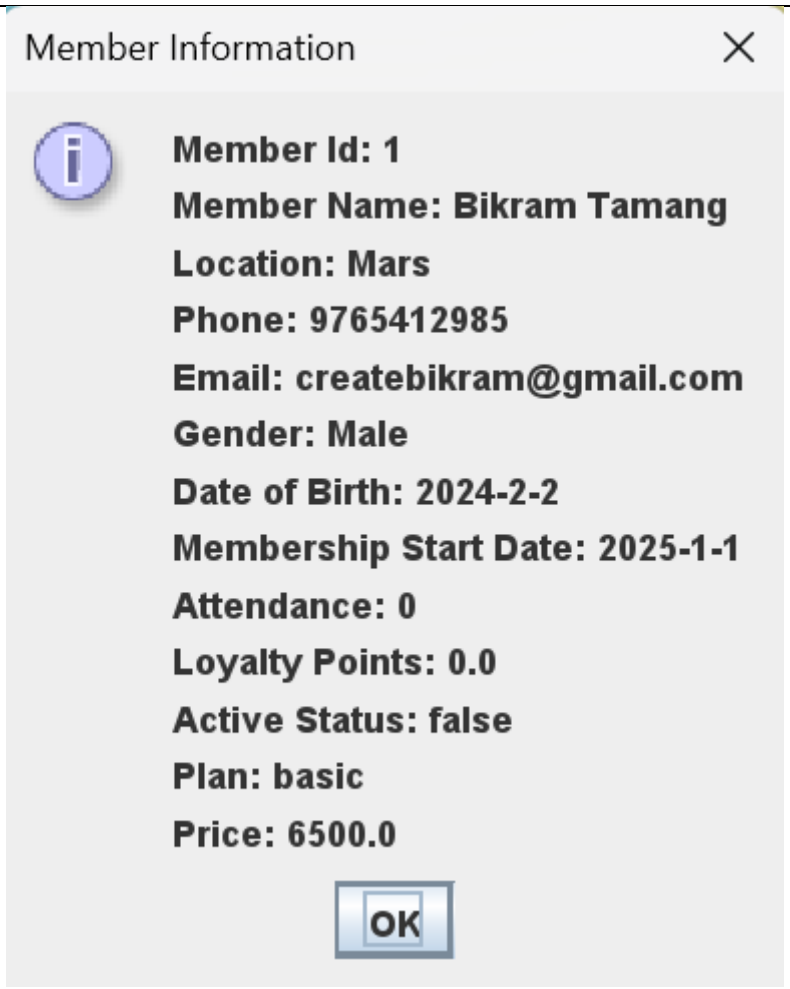


- Confirmation message



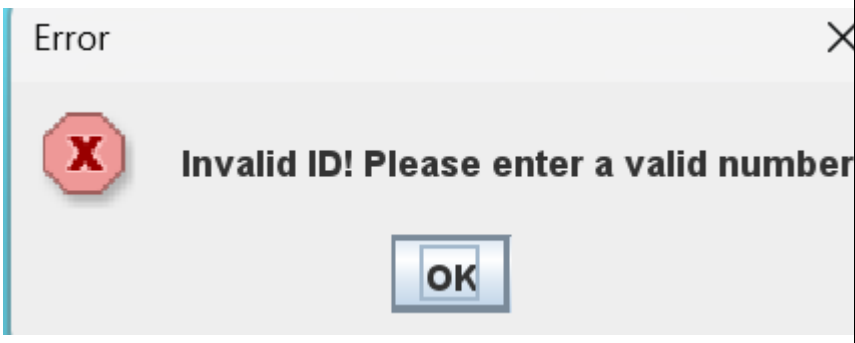
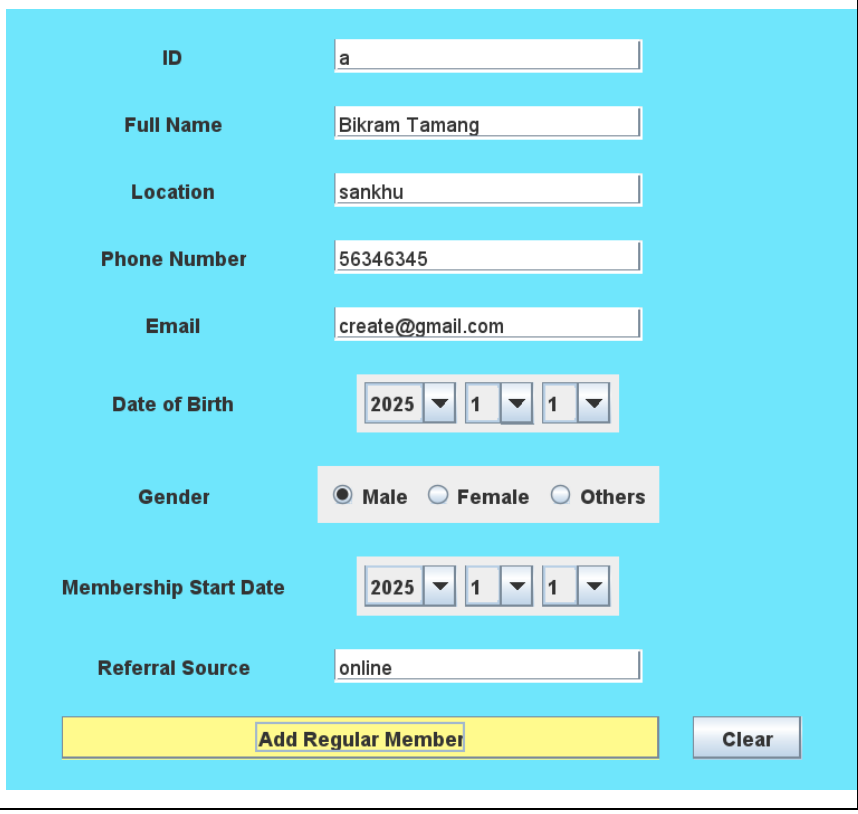
- For confirmation Members were displayed in screen when display button is pressed along with filling member id field in JOptionPane.

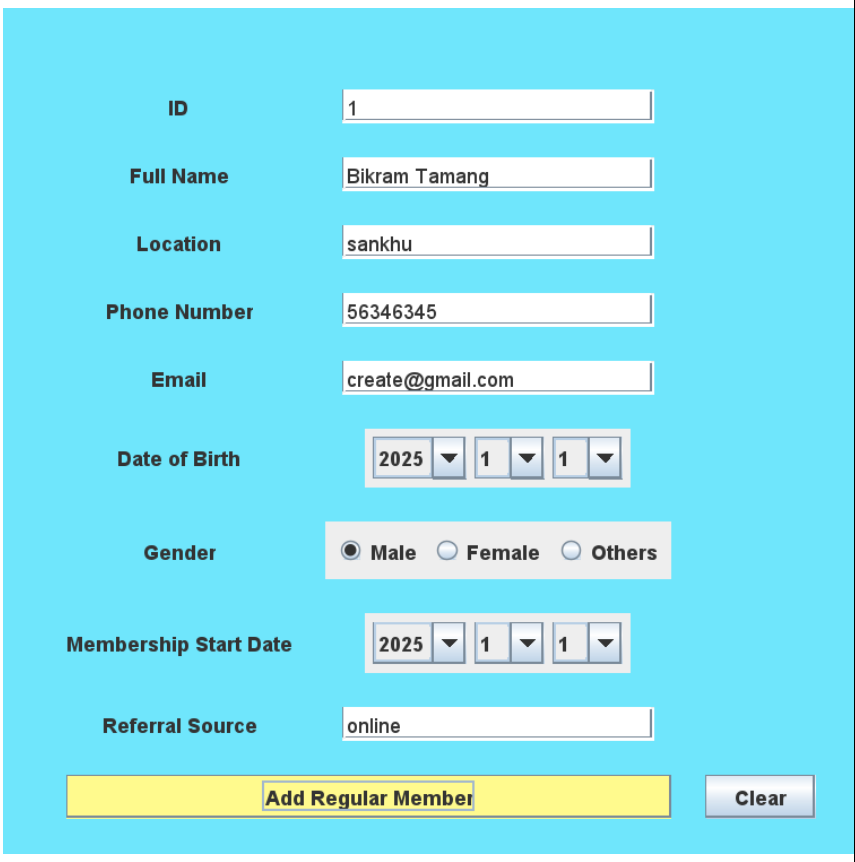
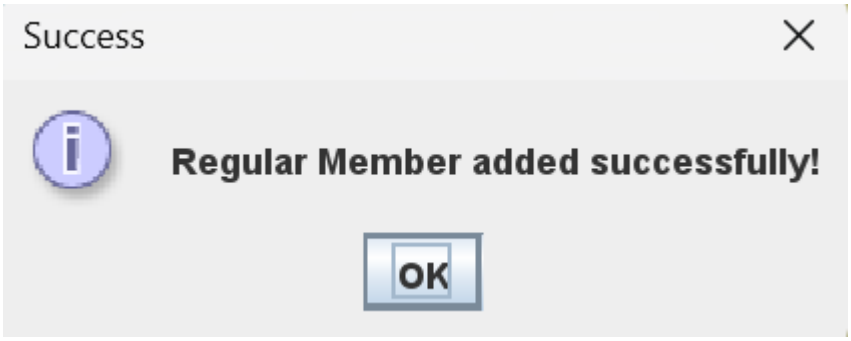


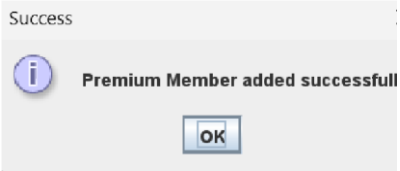
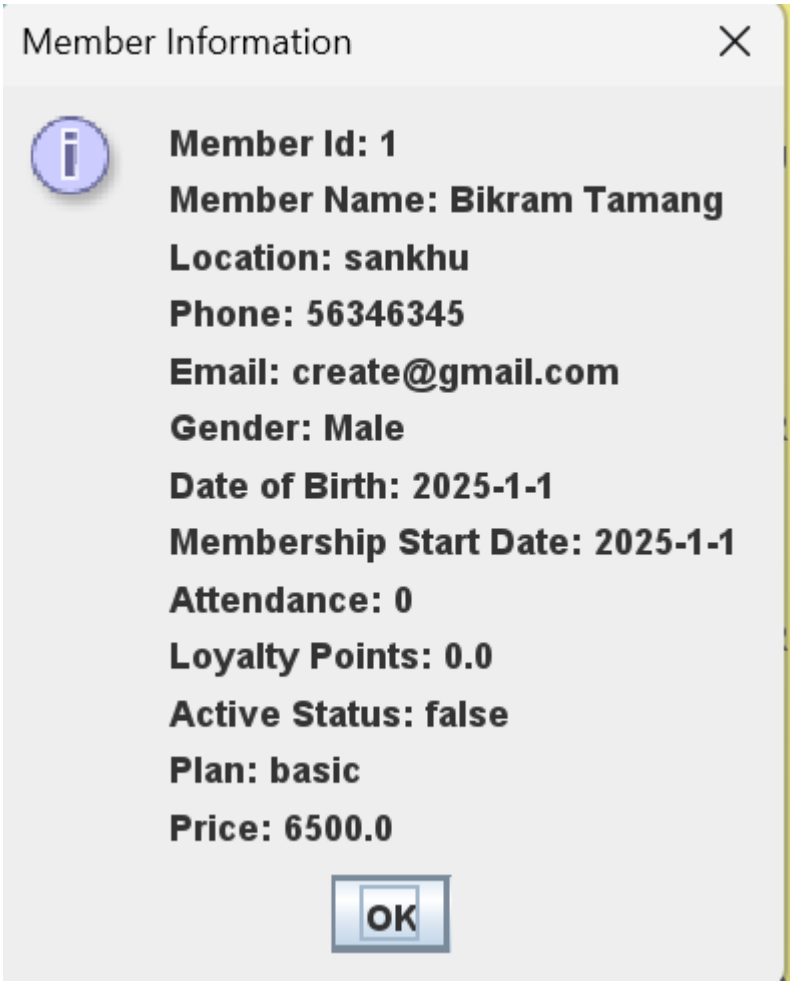
	
Conclusion	The Test was Successful.

8.3. Test 2 For Premium Member

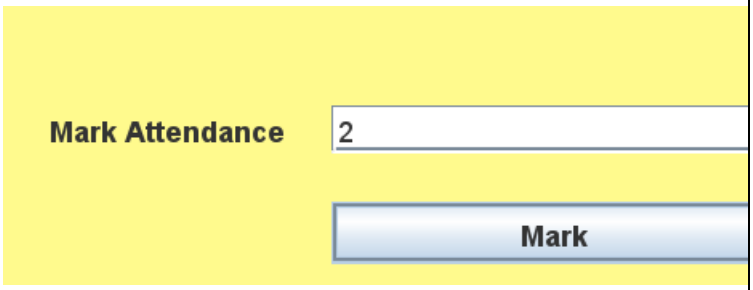
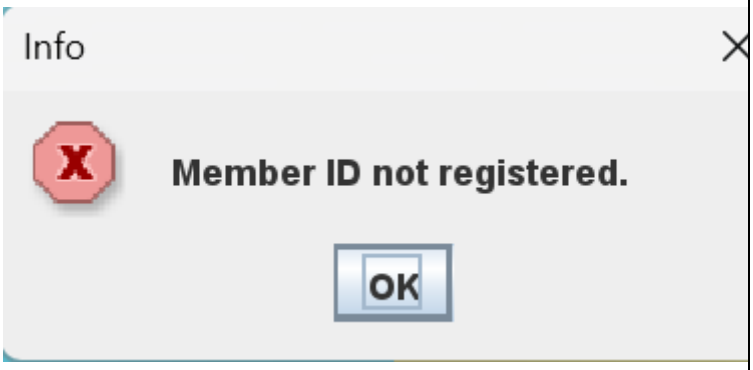
Test	2
Objective	To test the functionality of “Add Premium Member” Button.
Activity	• If any field is left empty, the system displays an error message. • If all fields are filled with valid input and the “Add Member” button is clicked, the member is added.

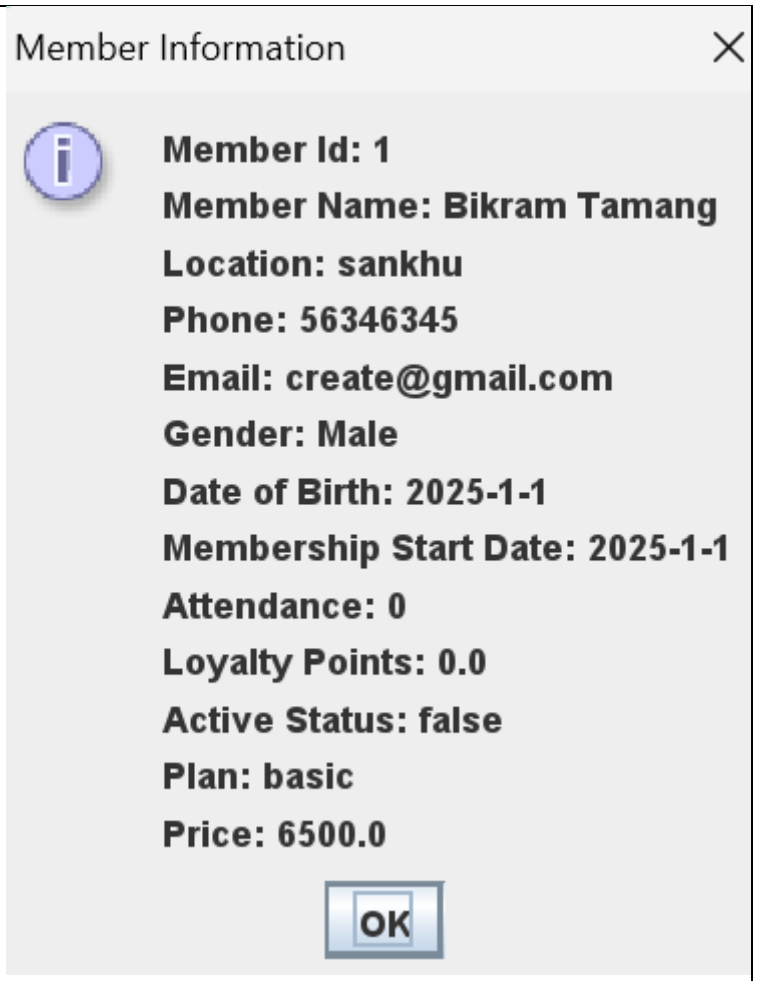
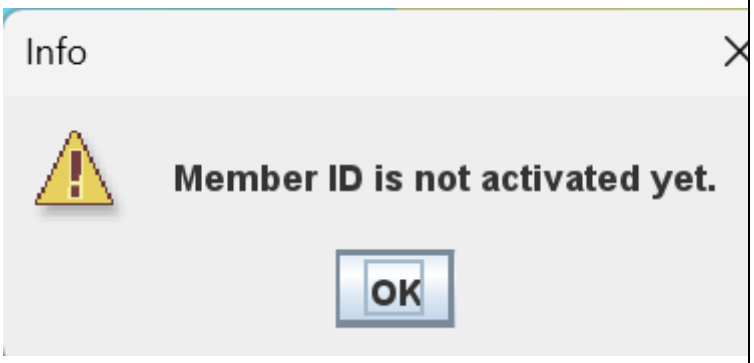
Expected Output	• If any field is left empty, an error message is displayed. • If all fields are filled but contain invalid input, an error message is displayed. • If all fields are filled with valid input a confirmation message “Regular Member was added successfully” is displayed. Member information appears in the member list.
Actual Output	• Error message displayed when not all fields were filled.  • • Error message displayed when all fields were filled but the ID was invalid.  •

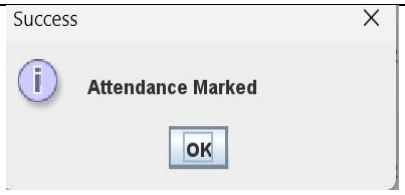
	•  • When all fields were filled with valid input Confirmation message “Regular Member added successfully” was displayed. •  •  • Valid input was filled in all fields.

	• Confirmation message •  • Members were displayed in screen when display button is pressed along with filling member id field. • 
Conclusion	The Test was Successful.

8.4. Test 3 For Mark Attendance

Test	3
Objective	To test the functionality of “Mark Attendance” button for both regular and premium member.
Activity	• Entered an ID in the input field. • Clicked the “Mark Attendance” button for regular member and premium member.
Expected Output	• If the ID is invalid, an error message is displayed. • If the ID is valid but activeStatus is false, an error message is displayed. • If the ID is valid and activeStatus is true, attendance is incremented by 1.
Actual Output	• Error message displayed when the ID was invalid. •  •  • Error message displayed when the ID was valid but activeStatus was false.

	•  •  • Display suitable message and Attendance increased by 1 when the ID was valid and activeStatus was true and also increase loyaltypoints by 5 for regular member and by 10 for premium member

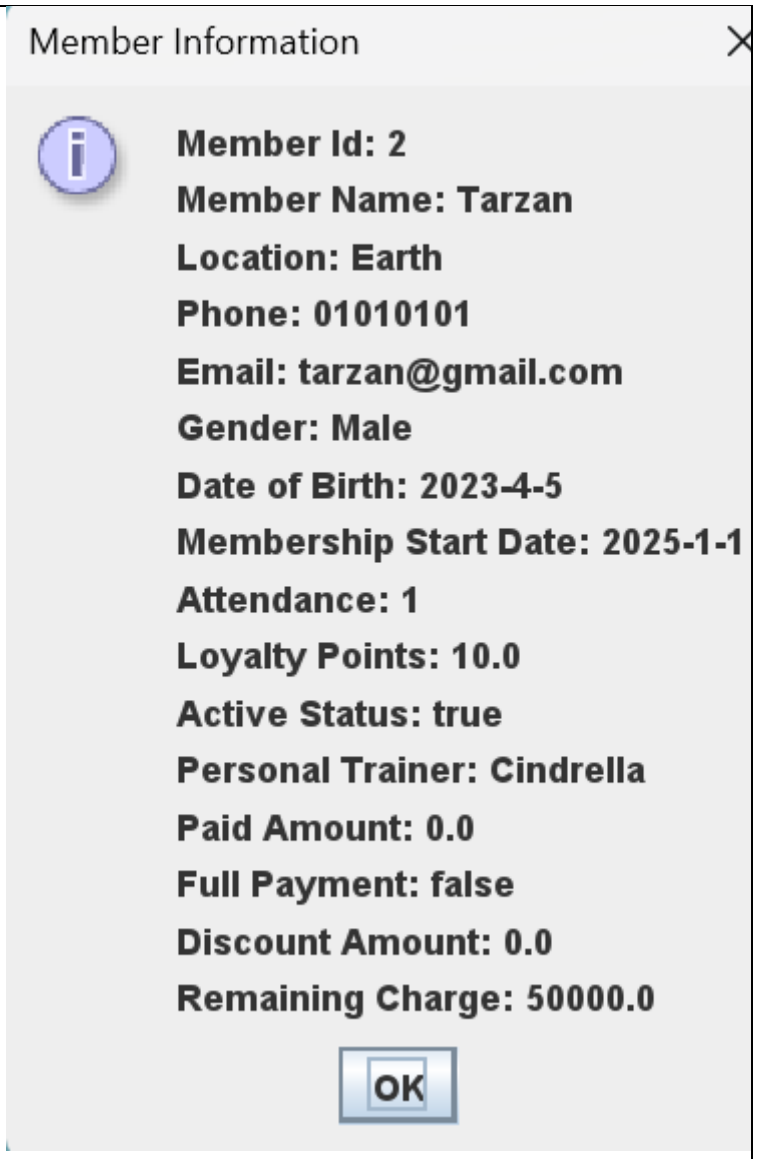
- 

- When id is valid.

- For regular member: loyalty points was increased by 5.

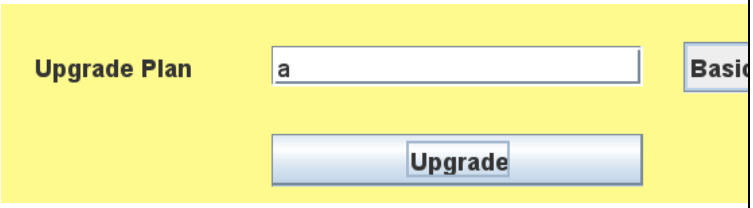
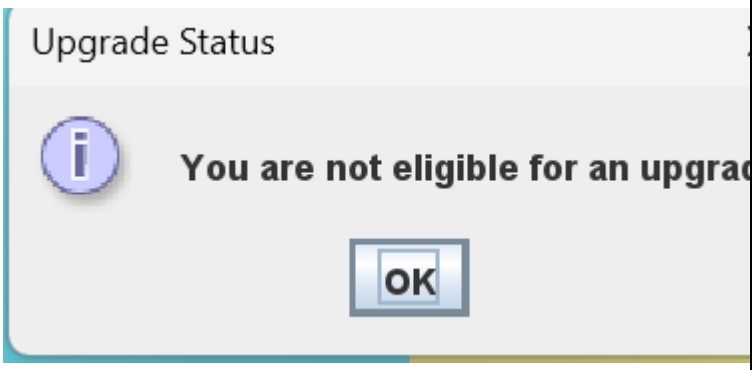
- 

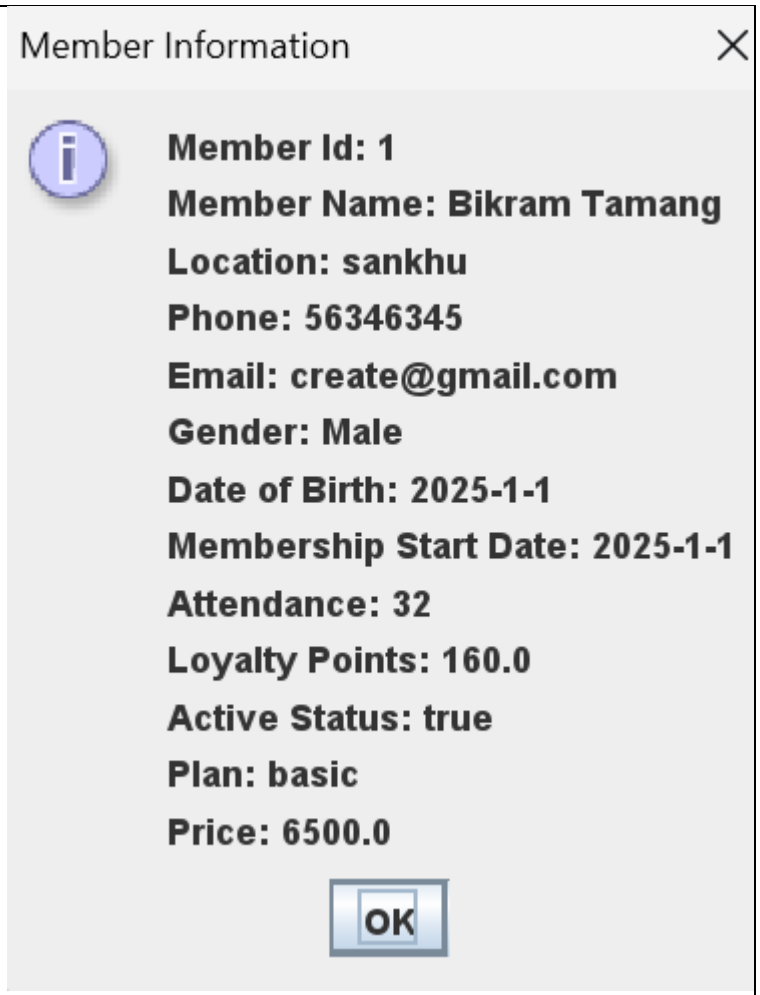
- For Premium Member: loyalty points was increased by 10.

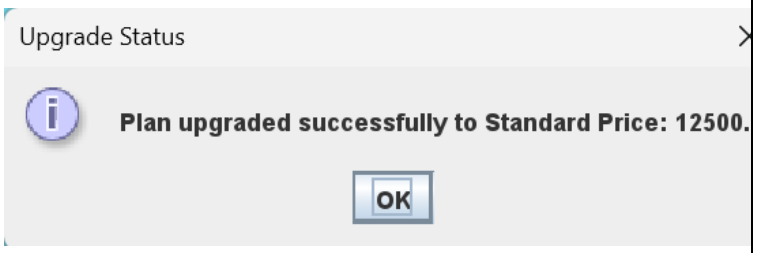
	• 
Conclusion	The Test was Successful.

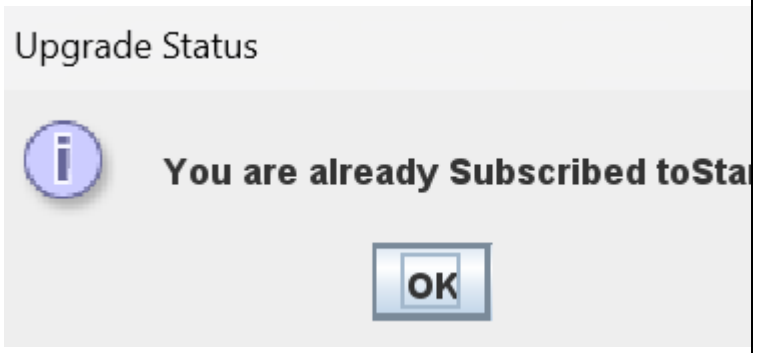
8.5. Test 4 For Upgrade Plan

Test	3
Objective	To test the functionality of “Upgrade” button.
Action	Entered an ID in the input field.

	Clicked the “Upgrade” button.
Expected Output	• If the ID is invalid, an error message is displayed. • If the ID is valid but activeStatus is false, an error message is displayed. • If the ID is valid and activeStatus is true, the member is upgraded successfully.
Actual Output	• Error message displayed when the ID was invalid. •  •  • Error message displayed when the ID was valid attendance was less than 30 i.e. isEligibleForUpgrade was false. •  • Member was successfully upgraded when the ID was valid and isEligibleForUpgrade was true i.e. attendance was greater or equal to 30.

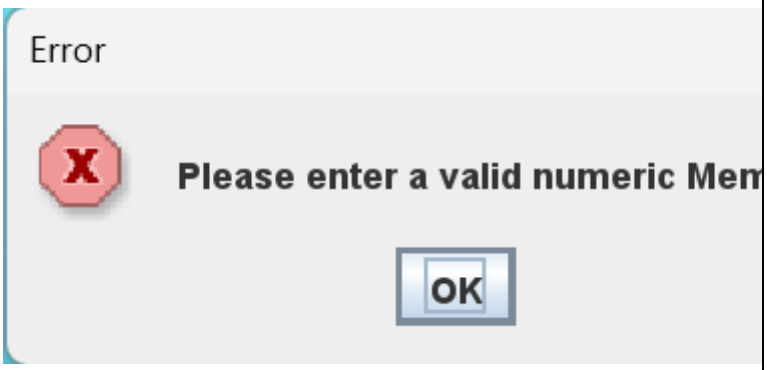
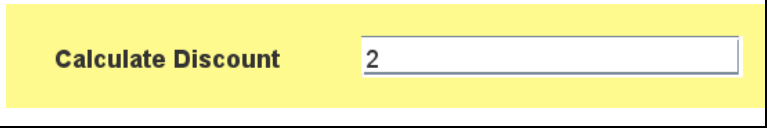
- 

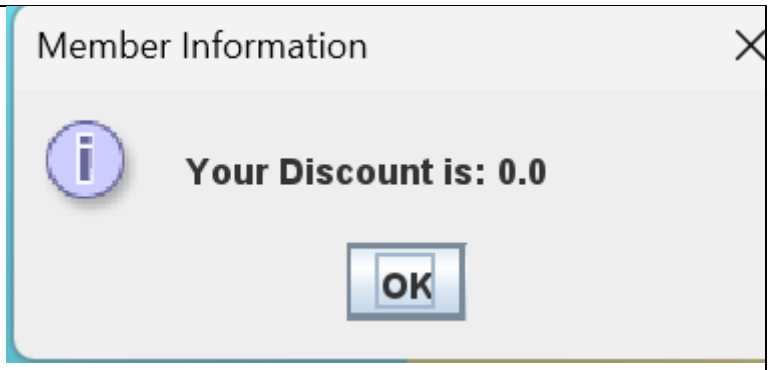
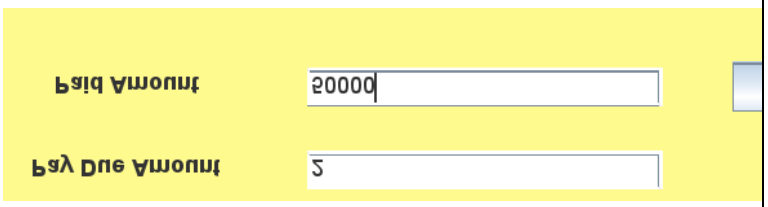
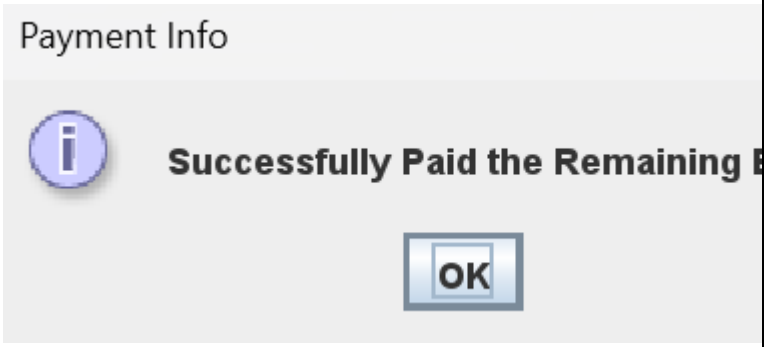
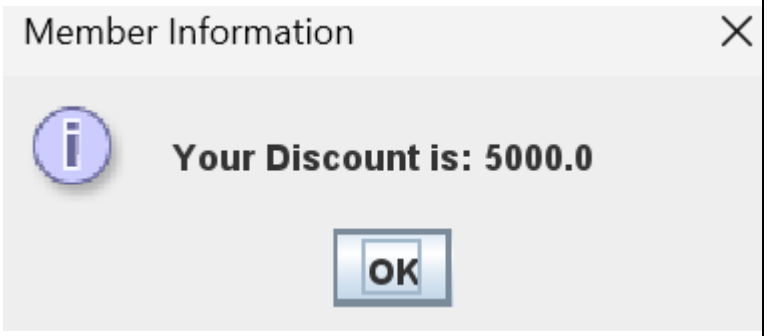
- 

- If current object was already subscribed to certain plan, it showcased a message.
- 

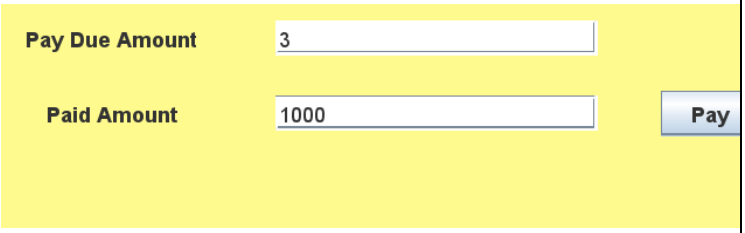
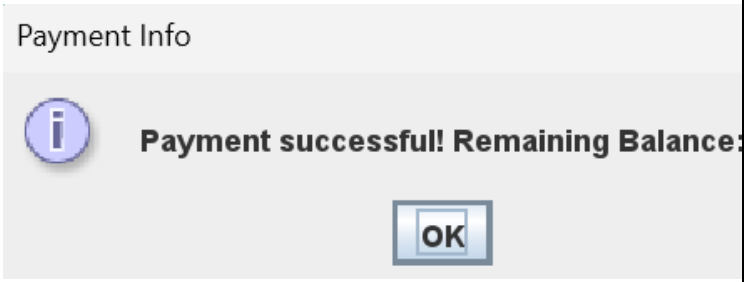
Conclusion	The Test was Successful.

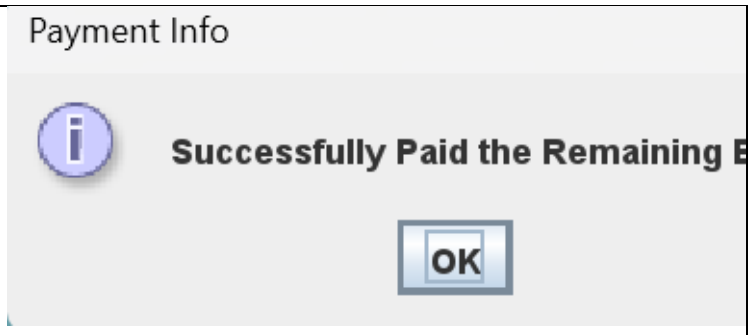
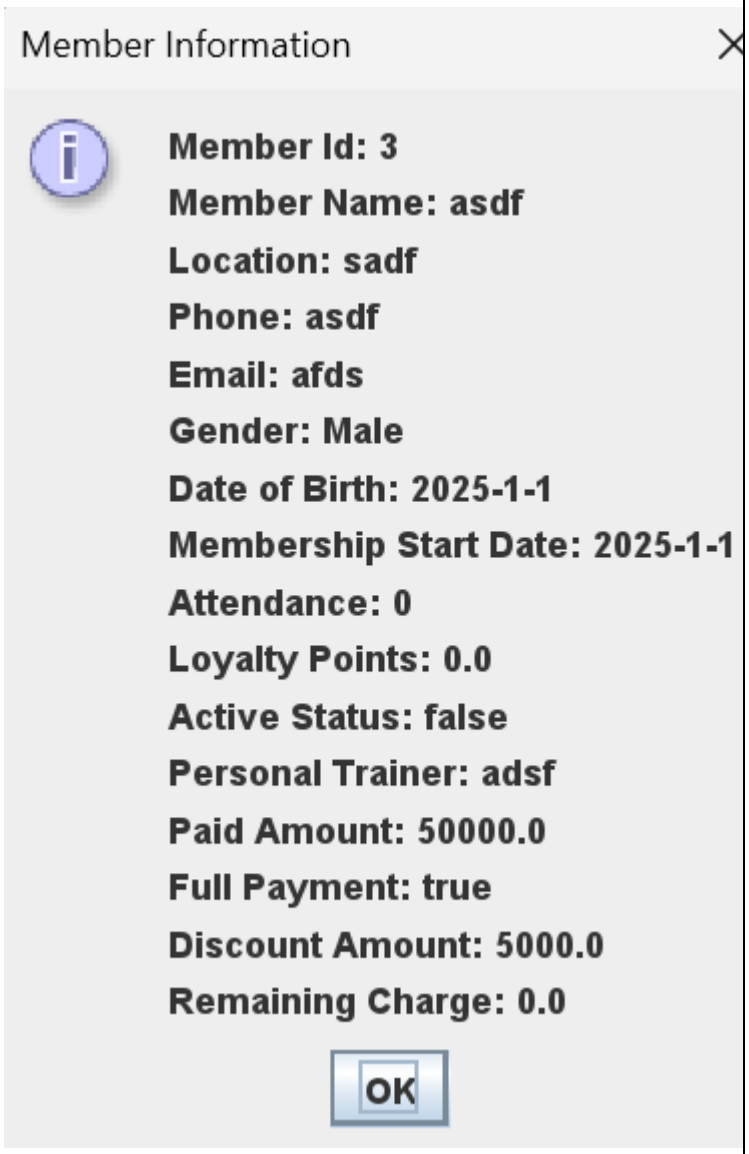
8.6. Test 5 For Calculate Discount

Test	4
Objective	To test the functionality of “calculate” button.
Activity	Entered an ID in the input field. Clicked the “Calculate” button.
Expected Output	• If the ID is invalid, an error message is displayed. • If the ID is valid but the payment is incomplete, a message like “Full Payment needed” is displayed. • If the ID is valid and payment is complete, the discount is calculated.
Actual Output	• Error message displayed when the ID was invalid. •  •  • Discount was calculated 0 when full payment was not done. • 

	•  • Discount was calculated when the ID was valid and full payment was complete. •  •  • 
Conclusion	The test was successful.

8.8. Test 6 For Pay Due Amount

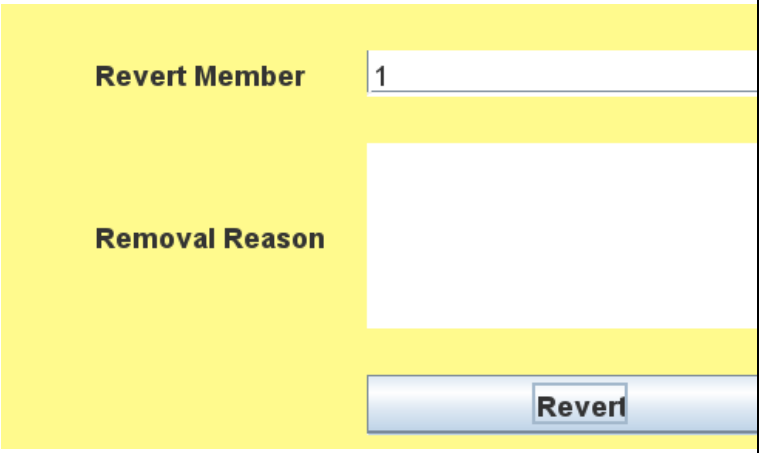
Test	4
Objective	To test the functionality of “Pay” button.
Activity	Entered an ID in the input field. Clicked the “Pay” button.
Expected Output	• If the ID is invalid, an error message is displayed. • If the ID is valid but there is no due amount, a message like “No due amount” is displayed. • If the ID is valid and there is a due amount, payment is processed and due is cleared.
Actual Output	• Error message displayed when the ID was invalid.  • Remaining amount was shown when the paid amount was less than the full price.  • “Successfully Paid the Remaining Balance” was shown when the paid was full price i.e. 50000 in this case.

	•  • Due amount was successfully cleared when the ID was valid and a payment was due. • 
Conclusion	The test was successful.

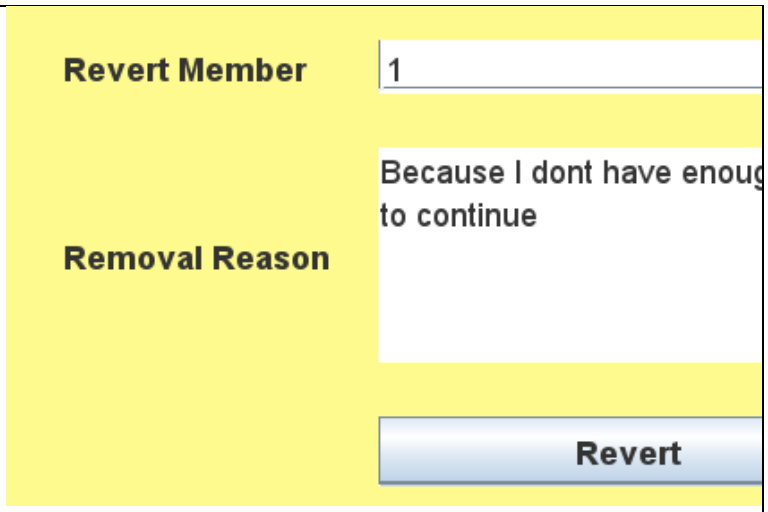
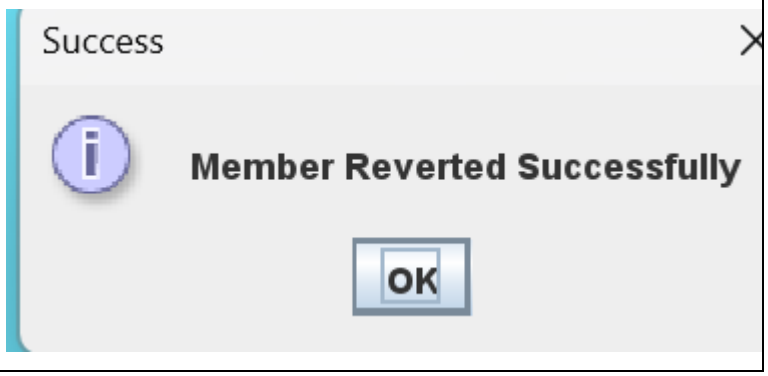
8.9. Test 7 For Revert Member

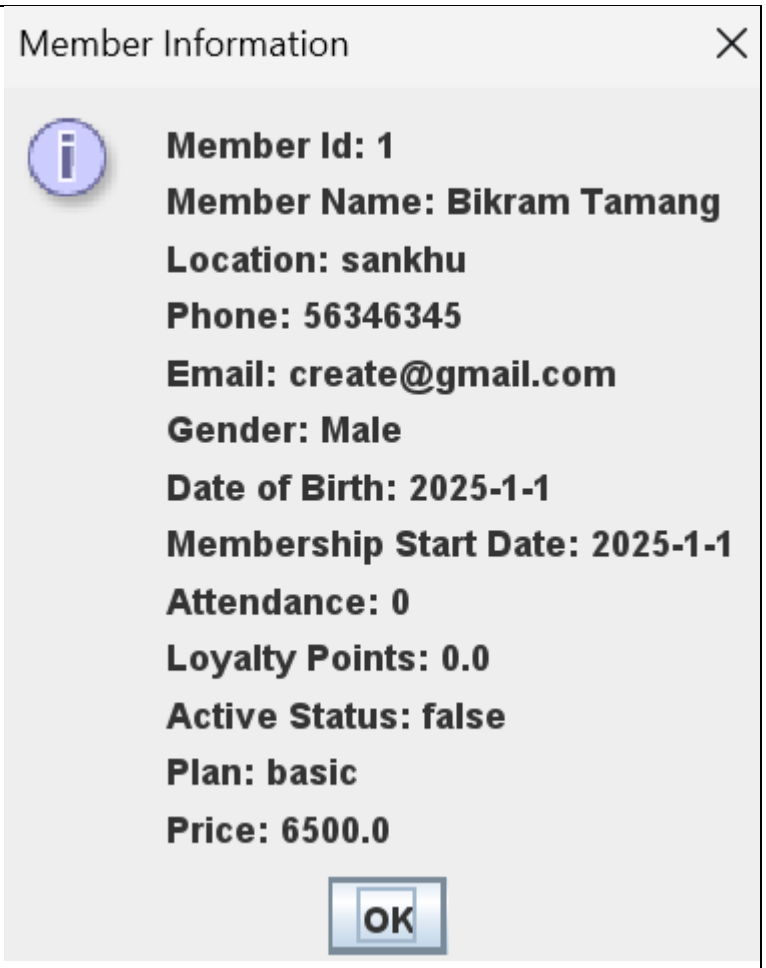
Test	5
Objective	To test the functionality of “revert” button.
Activity	Filled the id input And clicked the “revert” button.
Expected Output	• If the ID is invalid, an error message is displayed. • **For a Regular Member:** • An error message was shown when both the fields were not filed. • If the ID is valid, clicking the **“Revert Regular”** button sets activeStatus to false, attendance to 0, loyaltyPoints to 0.0, isEligibleForUpgrade to false, plan to basic, price to 6500, removalReason to removalReason. • • **For a Premium Member:** • If the ID is valid, clicking the **“Revert Premium”** button sets activeStatus to false, attendance to 0, loyaltyPoints to 0.0, personalTrainer is set to empty string, isFullPayment set to false, paidAmount set to 0, discountAmount set to 0.
Actual Output	1. Error message displayed both the fields were not filled.

2. **For a Regular Member and for this current object,** valid ID resulted in:

- activeStatus set to false,
- attendance set to 0,
- loyaltyPoints set to 0.0,
- isEligibleForUpgrade set to false,
- plan set to basic,
- price set to 6500,
- removalReason set to removalreason after clicking **"Revert Regular"**.
- 

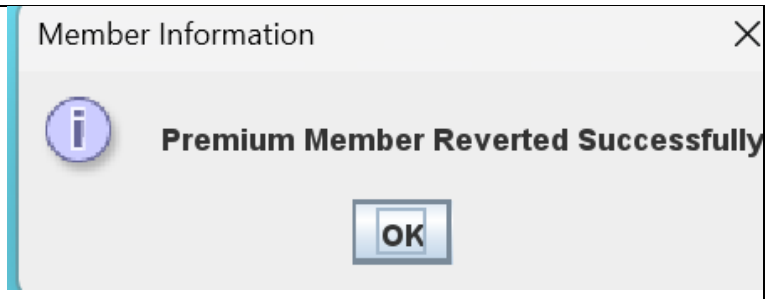
- 

	•  Revert Member 1 Removal Reason Because I dont have enoug to continue Revert •  Success Member Reverted Successfully OK

- 

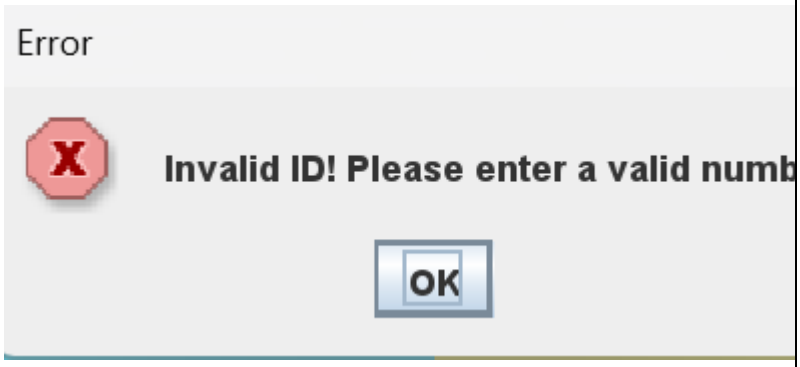
3. **For a Premium Member and for this current object ,** valid ID resulted in:

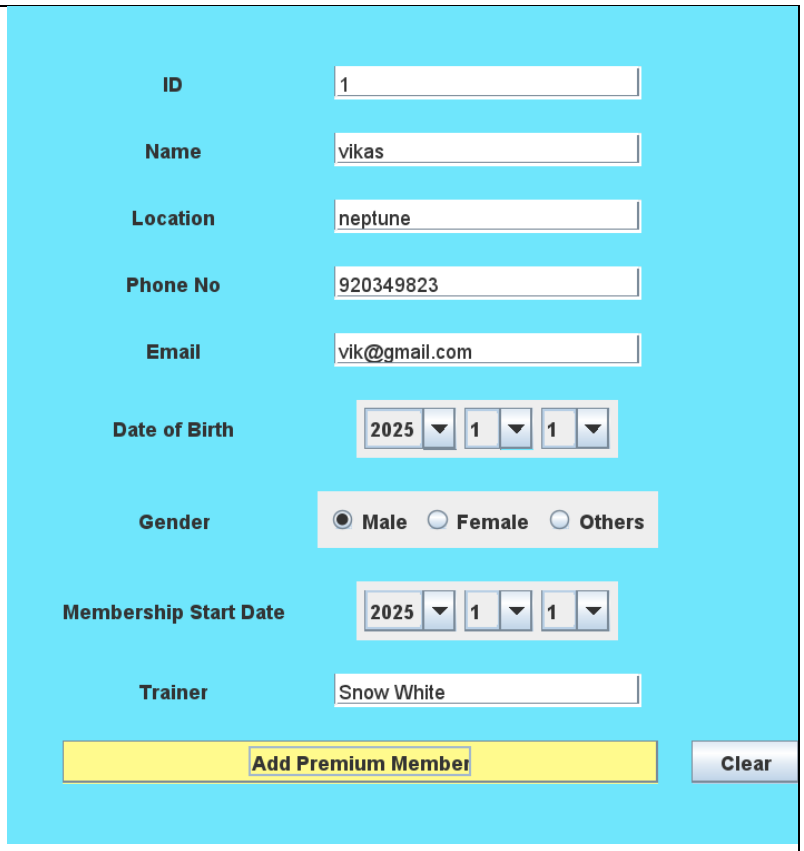
- activeStatus set to false,
- attendance set to 0,
- loyaltyPoints set to 0.0,
- personalTrainer is set to empty string,
- isFullPayment set to false,
- paidAmount set to 0,
- discountAmount set to 0 after clicking **"Revert Premium"**.
- 

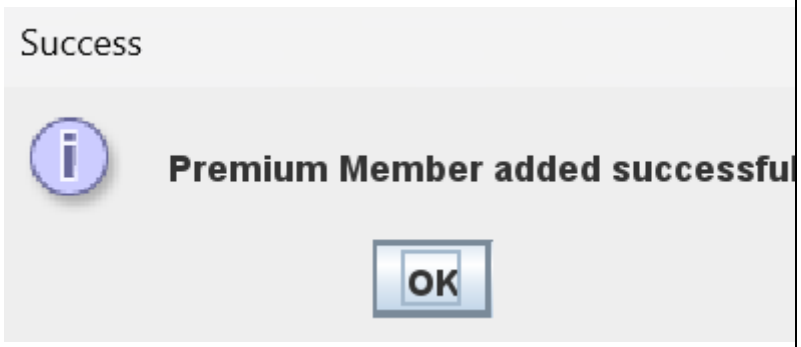
	•  • 
Conclusion	The Test was successful.

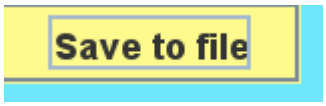
8.10. Test 8 For save and read file

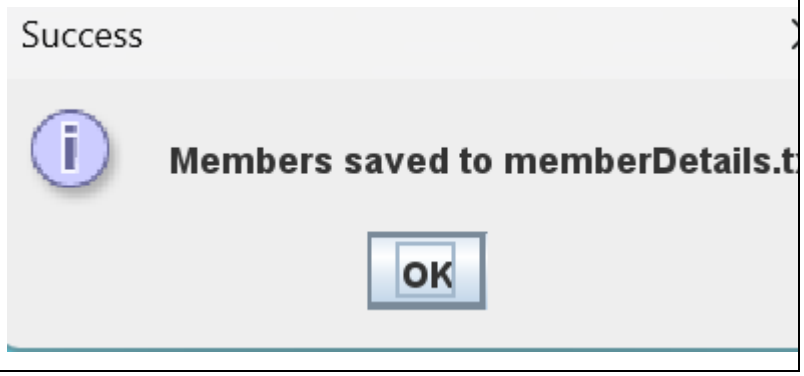
Test	5
Objective	To test the functionality of “save” and “Read All Members” button.
Activity	• Filled all required fields in the registration form. • For **Regular Member**, clicked **“Add Regular Member”**. • For **Premium Member**, clicked **“Add Premium Member”**. • Then clicked **“Save”** to store member data. • Clicked **“Read All Members”** to retrieve member details.
Expected Output	• If the ID is invalid, an error message is displayed. • For **Regular Member**: If the ID is valid and all fields are filled: • Clicking **“Add Regular Member”** adds the member to the arraylist. • Clicking **“Save”** stores data in MemberDetails.txt. • Clicking **“Read All Members”** displays the member’s details by ID. • For **Premium Member**: If the ID is valid and all fields are filled: • Clicking **“Add Premium Member”** adds the member. • Clicking **“Save”** stores data in MemberDetails.txt. • Clicking **“Read All Members”** displays the member’s details by ID.

Actual Output	• Error message was shown when the **ID was invalid**.  •  • • For **Premium Member**: Valid ID and all fields filled:

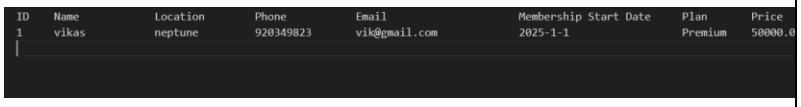
- 

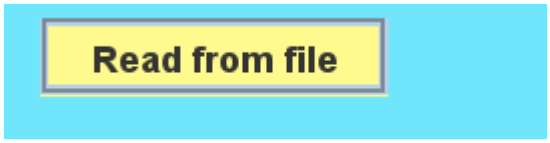
- 

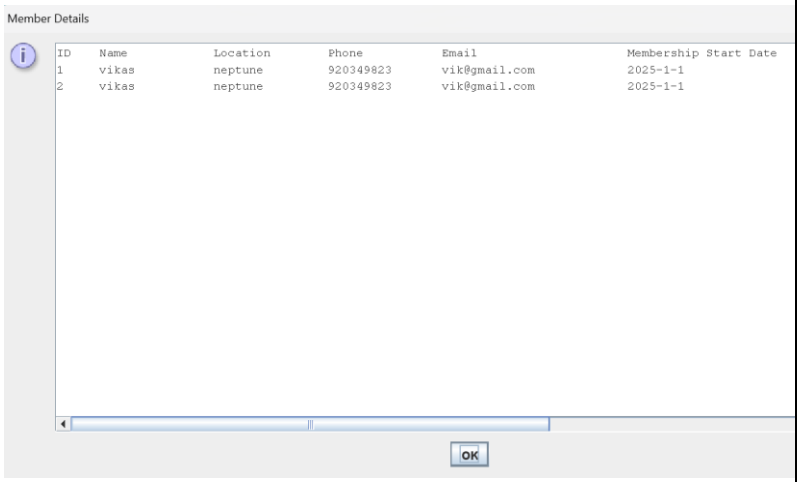
- “Add Premium Member” added the member.
- “Save” saved the data to MemberDetails.txt file.
- 

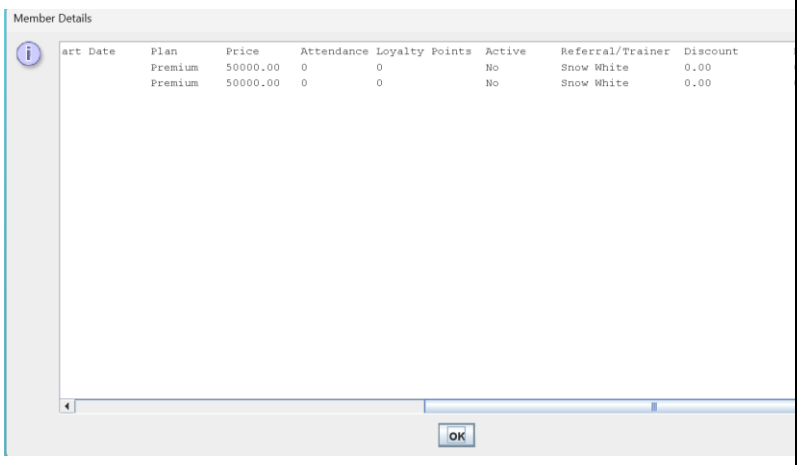
- 

- 

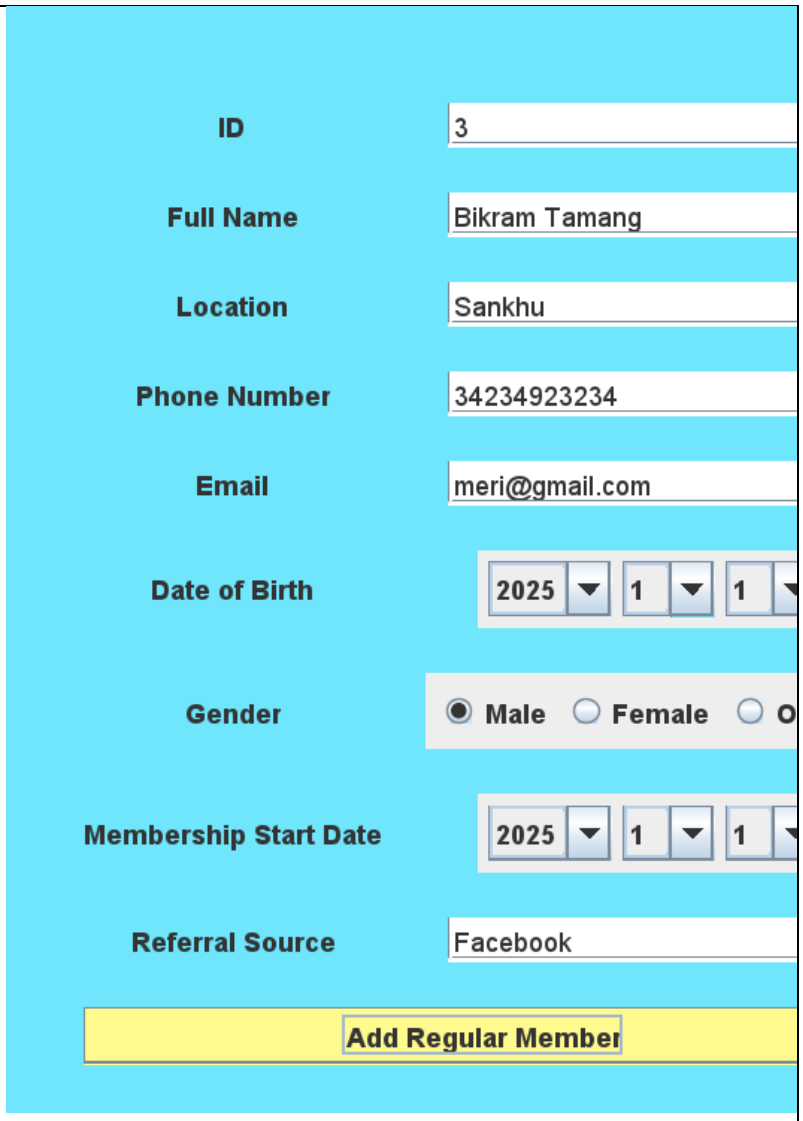
- 

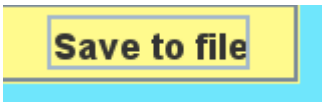
- "Read All Members" displayed correct details of all the arraylist.
- 

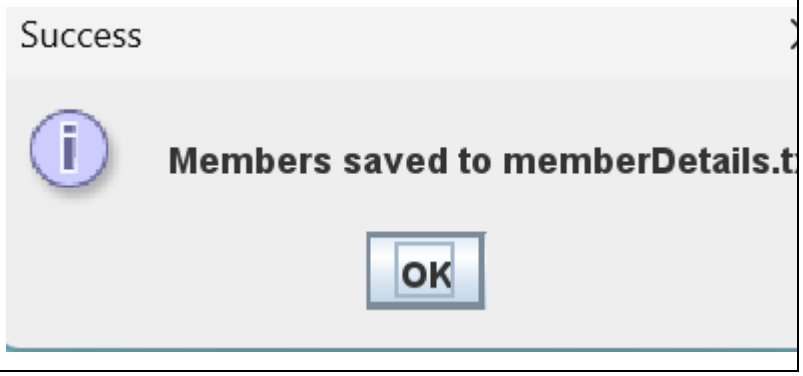
- 

- 

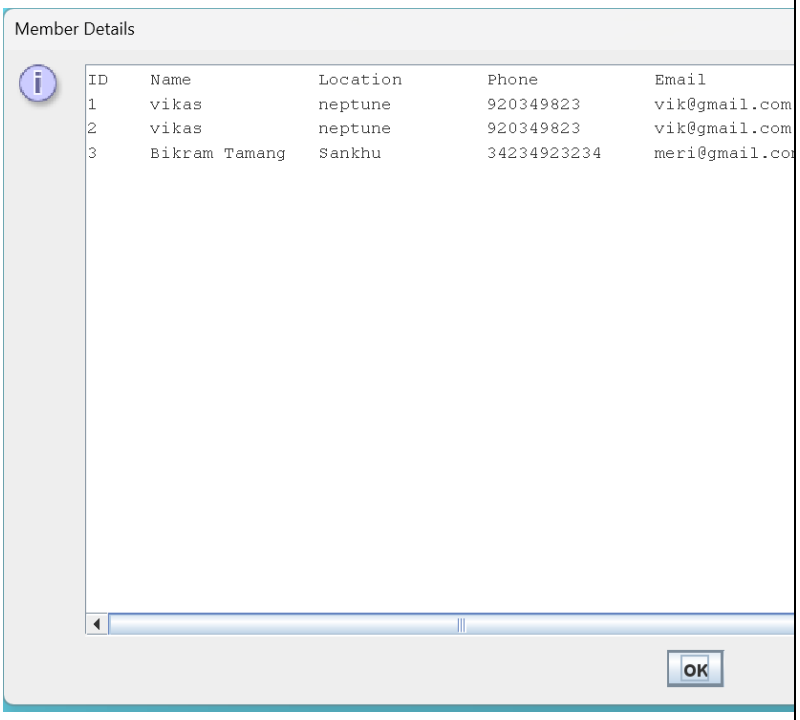
- For **Regular Member**: Valid ID and all fields filled:

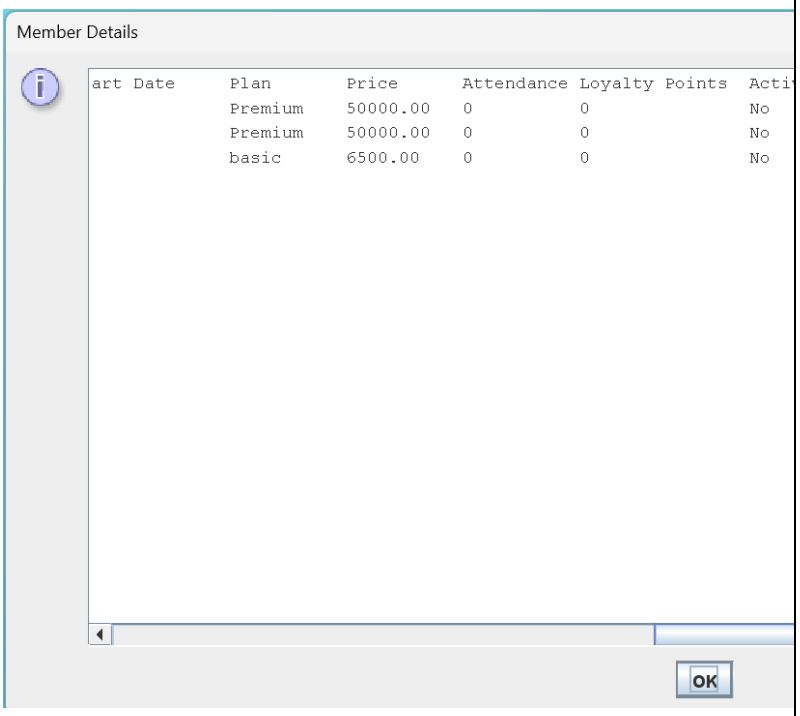
- 

- “Add Regular Member” added the member.
- “Save” saved the data to .txt file.
- 

- 

- 

- “Read All Members” displayed correct details of all the regular member in arraylist.
- 

- 

Con clusi on	The test was Successful.

9. Conclusion

Creating the GymMember class was an insightful experience in applying key object-oriented programming principles, including abstraction, encapsulation, and inheritance. Designed as an abstract base class, it allowed different gym member types to inherit common attributes and methods while defining their own unique behaviors. Essential features such as activating/deactivating memberships, tracking attendance, and displaying member details were implemented systematically using utility and accessor methods. This approach ensured the code remained modular, reusable, and easy to maintain.

During development, I strengthened my understanding of constructors for initializing object properties, using protected access modifiers to facilitate inheritance, and implementing getter methods to securely retrieve private data. I also gained clarity on abstract methods, learning how they enforce child classes to provide their own implementations. Structuring the class in this manner not only streamlined member data management but also highlighted the significance of writing clean, adaptable code for future enhancements.

Despite these benefits, the project presented challenges. Deciding which methods should be abstract and correctly overriding them in subclasses required careful consideration. Additionally, handling default values like attendance records and membership status in the constructor led to initial confusion. Overcoming these obstacles involved consulting Java documentation, studying comparable examples, and conducting method-by-method testing to ensure proper functionality. This process not only refined my problem-solving skills but also deepened my grasp of effective Java class design.

10. Appendix

GymMember.java

```
package gymproject; //this is the package named gymproject


public abstract class GymMember {


    protected int id; //protected access modifier so that it is accessible within the same package and by subclasses (even if they are in different package)
    protected String name;
    protected String location;
    protected String phone;
    protected String email;
    protected String gender;
    protected String dob;
    protected String membershipStartDate;
    protected int attendance;
    protected double loyaltyPoints;
    protected boolean activeStatus;


    /**
 * Abstract class representing a Gym Member.
 * Holds common attributes and behaviors for all gym members.
 */
    public GymMember(int id, String name, String location, String phone, String email, String dob, String gender, String membershipStartDate) {
        this.id = id;
```

```
        this.name = name;
        this.location = location;
        this.phone = phone;
        this.email = email;
        this.gender = gender;
        this.dob = dob;
        this.membershipStartDate = membershipStartDate;
        this.attendance = 0;
        this.loyaltyPoints = 0;
        this.activeStatus = false;
    }

    // Getter method for id
        public int getId() {
            return id;
        }

    // Getter method for name
        public String getName() {
            return name;
        }

    // Getter method for location
        public String getLocation() {
            return location;
        }
    // Getter method for phone
```

```
    public String getPhone() {
        return phone;
    }

// Getter method for email
    public String getEmail() {
        return email;
    }

// Getter method for gender
    public String getGender() {
        return gender;
    }

// Getter method for dob
    public String getDob() {
        return dob;
    }

// Getter method for membershipStartDate
    public String getMembershipStartDate() {
        return membershipStartDate;
    }
// Getter method for attendance

    public int getAttendance() {
```

```
        return attendance;
    }

// Getter method for loyaltyPoints
    public double getLoyaltyPoints() {
        return loyaltyPoints;
    }

// Getter method for activeStatus
    public boolean isActiveStatus() {
        return activeStatus;
    }

    /**
     * Abstract method to mark attendance.
     * Must be implemented by subclass
     */
    public abstract void markAttendance();

    /**
 * Activates the membership by setting activeStatus to true.
 */
    public void activateMembership() {
        this.activeStatus = true;
    }

    /**
```

```
    * Deactivates the membership if currently active.
    */
      public void deactivateMembership() {
          if (activeStatus) {
              this.activeStatus = false; //sets the activeStatus to false
          } else {
              System.out.println("Already Deactivated or Not activated till now");
          }
      }

      /**
     * Resets the member's status: deactivates, resets attendance and loyalty points.
    */
    public void resetMember() {
      this.activeStatus = false; //resets activeStatus to false
      this.attendance = 0; //resets attendance to 0
      this.loyaltyPoints = 0; //resets loyaltyPoints to 0
    }

    /**
     * Displays the member's information as a string.
     * @return Formatted member details.
     */    public String display() {
      return "Member Id: " + this.id + "\n"
              + "Member Name: " + this.name + "\n"
              + "Location: " + this.location + "\n"
```

```
            + "Phone: " + this.phone + "\n"
             + "Email: "+this.email+"\n"
            + "Gender: " + this.gender + "\n"
            + "Date of Birth: " + this.dob + "\n"
            + "Membership Start Date: " + this.membershipStartDate + "\n"
            + "Attendance: " + this.attendance + "\n"
            + "Loyalty Points: " + this.loyaltyPoints + "\n"
            + "Active Status: " + this.activeStatus;
    }

}
```

RegularMember.java

```
package gymproject;

import javax.swing.JOptionPane;

/**
 * Represents a Regular Gym Member with plan-based pricing and upgrade eligibility.
 * Inherits common member properties from GymMember.
 */
public class RegularMember extends GymMember {

    /**

     * */
    private final int ATTENDANCE_LIMIT;
    private boolean isEligibleForUpgrade;
    private String removalReason;
    private String referralSource;
    private String plan;
    private double price;
     // GYMMEMBER CLASS KAI CONSTRUCTOR MAA QUESTION ANUSAR CHANGES GARNA XA

    /**
     * Constructor to initialize a RegularMember with provided details.
```

```
    *
    * @param id                Member ID
    * @param name              Name of the member
    * @param location          Location
    * @param phone             Phone number
    * @param email             Email address
    * @param gender            Gender
    * @param dob               Date of birth
    * @param membershipStartDate Date of membership start
    * @param referralSource    Source of referral
    */
   public RegularMember(int id, String name, String location, String phone, String email, String gender, String dob, String membershipStartDate, String referralSource) {
       // Calling the parent constructor of GymMember Class using super keyword
       super(id, name, location, phone, email, gender, dob, membershipStartDate);

       // Setting the default values of the attributes of child class
       this.ATTENDANCE_LIMIT=30;
       this.isEligibleForUpgrade = false; // Use the passed parameter
       this.removalReason = ""; // Use the passed parameter
       this.referralSource = referralSource; // Use the passed parameter
       this.plan = "basic"; // Use the passed parameter
       this.price = 6500.0; // Use the passed parameter
   }

   // Getters for member-specific fields
```

```
    public int getAttendanceLimit() {
        return ATTENDANCE_LIMIT;
    }

    public boolean getIsEligibleForUpgrade() {
        return isEligibleForUpgrade;
    }

    public String getRemovalReason() {
        return removalReason;
    }

    public String getReferralSource() {
        return referralSource;
    }

    public String getPlan() {
        return plan;
    }

    public double getPrice() {
        return price;
    }

     /**
     * Overrides the abstract method markAttendance.
     * Increments attendance and loyalty points.
```

```
     * If attendance reaches the limit, member becomes eligible for upgrade.
     */
    @Override
    public void markAttendance() {
        if (activeStatus) {
            this.attendance++;
            this.loyaltyPoints+=5;
            if(getAttendance()>=getAttendanceLimit()){
                this.isEligibleForUpgrade=true;
            }
        }
    }

    /**
     * Returns the price for a given plan.
     *
     * @param plan Name of the plan (case-insensitive)
     * @return Price of the plan or -1 if invalid
     */
    public double getPlanPrice(String plan) {
        switch (plan.toLowerCase()) //plan is converted to lowercase using String
method .toLowerCase() method for flexibility and eradication of unwanted errors
due to capital and small letters
        {
            case "basic":
                price = 6500;
                break;
```

```
        case "standard":
            price = 12500;
            break;

        case "deluxe":
            price = 18500;

            break;

        default:
            System.out.println("Invalid Plan Name: ");
            System.out.println("Choose among these: Basic, Standard, Deluxe");
            return -1;
    }
    return price;
}

/**
 * Upgrades the member's plan if eligible.
 *
 * @param plan Plan to upgrade to
 * @return Result message after attempting upgrade
 */
public String upgradePlan(String plan) {
    // Check eligibility first
    if (this.isEligibleForUpgrade) {
```

```
        // Handle the plan selection
        if (plan.equals(this.plan)) {
            return "You are already Subscribed to" +this.plan;
        }
        switch (plan) {
            case "Basic":
                this.price = 6500;
                this.plan="Basic";
                break;
            case "Standard":
                this.price = 12500;
                this.plan="Standard";
                break;
            case "Deluxe":
                this.price = 18000;
                this.plan="Deluxe";
                break;
            default:
                JOptionPane.showMessageDialog(null, "Invalid plan selected.");
                return "Invalid plan selected";
        }
        // Successfully upgraded
        return "Plan upgraded successfully to " + plan+ " Price: " +price;
    } else {
        // If not eligible for upgrade
        return "You are not eligible for an upgrade";
    }
```

```
    }



    /**
     * Resets the member data and sets removal reason.
     *
     * @param removalReason Reason for removing or reverting the member
     */
    public void revertRegularMember(String removalReason) {
        super.resetMember();
        this.isEligibleForUpgrade = false;
        this.plan = "basic";
        this.price = 6500;
        this.removalReason = removalReason;
    }

    /**
     * Displays member details including inherited and RegularMember-specific
data.
     *
     * @return Formatted string containing member details
     */
    @Override
    public String display() {
        // Call the base class display and get member info
        String memberInfo = super.display(); // Assuming the super class display()
returns a String
```

```
        // Append additional info specific to RegularMember
        memberInfo += "\nPlan: " + this.plan + "\n" +
                    "Price: " + this.price;

        // Append removal reason if available
        if (this.removalReason != null && !this.removalReason.isBlank()) {
            memberInfo += "\nRemoval Reason: " + this.removalReason;
        }

        return memberInfo;
    }
}
```

PremiumMember.java

```
package gymproject;

/**
 * Represents a Premium Member of the gym. Inherits from the abstract class
 * GymMember and adds additional premium features like personal trainer, full
 * payment tracking, and discount management.
 */
public class PremiumMember extends GymMember {

    // Final premium charge fixed at 50000
    private final double premiumCharge;
    private String personalTrainer;// Name of the personal trainer assigned to the member
    private boolean isFullPayment;// Indicates whether full payment has been made
    private double paidAmount;// Amount that has been paid so far
    private double discountAmount;// Discount amount calculated on full payment

    /**
     * Constructor to initialize a PremiumMember object with given details.
     *
     * @param id Member ID
     * @param name Member's name
     * @param location Member's location
     * @param phone Contact number
```

```
     * @param email Email address
     * @param gender Gender of the member
     * @param dob Date of birth
     * @param membershipStartDate Date when the membership started
     * @param personalTrainer Assigned personal trainer
     */
     public PremiumMember(int id, String name, String location, String phone, String email, String gender, String dob, String membershipStartDate, String personalTrainer) {
        super(id, name, location, phone, email, gender, dob, membershipStartDate);
        this.premiumCharge = 50000;
        this.personalTrainer = personalTrainer;
        this.isFullPayment = false;
        this.paidAmount = 0;
        this.discountAmount = 0;
    }

    // Getter method for premiumCharge
    public double getPremiumCharge() {
        return premiumCharge;
    }

    // Getter method for personalTrainer
    public String getPersonalTrainer() {
        return personalTrainer;
    }
```

```
    // Getter method for isFullPayment
    public boolean isFullPayment() {
        return isFullPayment;
    }

    // Getter method for paidAmount
    public double getPaidAmount() {
        return paidAmount;
    }

    // Getter method for discountAmount
    public double getDiscountAmount() {
        return discountAmount;
    }

    /**
     * Overrides the abstract method from GymMember to mark attendance.
     * Increases attendance count and loyalty points accordingly.
     */
    @Override
    public void markAttendance() {
        attendance++;
        loyaltyPoints += 10;
    }

    /**
     * Handles payment by the member towards their premium charge.
```

```
     *
     * @param paidAmount Amount to be paid
     * @return Message indicating the payment status
     */
    public String payDueAmount(double paidAmount) {
        if (isFullPayment) {
            return "Full Amount is already paid.";
        }

        if (this.paidAmount + paidAmount > premiumCharge) {
            return "More balance paid than the premium charge";
        }

        this.paidAmount += paidAmount;

        double remainingAmount = premiumCharge - this.paidAmount;

        if (remainingAmount == 0) {
            isFullPayment = true;
            return "Successfully Paid the Remaining Balance.";
        }

        return "Payment successful! Remaining Balance: " + remainingAmount;
    }

    /**
     * Calculates discount only if the member has fully paid the premium charge.
```

```
     *
     * @return Calculated discount amount
     */
    public double calculateDiscount() {
        if (isFullPayment) {
            discountAmount = premiumCharge * 0.10; //10% discount on premium
charge
        } else {
            discountAmount = 0;
        }
        return discountAmount;
    }

    /**
     * Resets all premium-specific fields and also calls the superclass method
     * to reset general member data.
     */
    public void revertPremiumMember() {
        // call parent class
        super.resetMember();
        this.personalTrainer = "";
        this.isFullPayment = false;
        this.paidAmount = 0;
        this.discountAmount = 0;
    }

    /**
```

```
     * Displays all member information, including inherited and premium-specific
     * fields.
     *
     * @return Formatted string containing member details
     */
    @Override
    public String display() {
        return super.display() + "\n"
                + "Personal Trainer: " + personalTrainer + "\n"
                + "Paid Amount: " + paidAmount + "\n"
                + "Full Payment: " + isFullPayment + "\n"
                + "Discount Amount: " + discountAmount + "\n"
                + "Remaining Charge: " + (premiumCharge - getPaidAmount());
    }

}
```

GymGUI.java

```
package gymproject;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
```

```
import java.util.Scanner;

import javax.swing.*;

public class GymGUI {

    // GUI Components Declaration
    private JFrame pDisplayFrame, frame, gFrame, rFrame, pFrame;
    private JButton rPlanPriceButton, pDueCalculate, pMarkButton, pPayButton, pCalculateButton, pRevertButton, pDisplayButton, pAddButton, gButton, rButton, pButton, acButton, deacButton, resButton, rAddButton, rClearButton, rMarkButton, rUpgradeButton, rRevertButton, rDisplayButton, saveButton,readButton;
    private JPanel pMSDPanel, rGenderPanel, rDPanel, pDatePanel, fTPanel, fBPanel, gTPanel, gFPanel, rTPanel, rMPanel, rFPanel, rFTPanel, rFFPanel, rAPanel, rATPanel, rAFPanel;
    private JLabel rPlanLabel, pDueLabel, pPaidAmtLabel, pMALabel, pPDALabel, pCDLabel, pRPMLabel, pDisplayLabel, pRRLabel, pTrainerLabel, pMSDLabel, pDobLabel, fTLabel, sTLabel, acLabel, deacLabel, resLabel, rTLabel, rFTLabel, rATLabel, rIdLabel, rFullNameLabel, rLocationLabel, rPhoneNumberLabel, rEmailLabel, rDobLabel, rGenderLabel, rMembershipStartDateLabel, rReferralLabel, rAMALabel, rAUPLabel, rARRMLabel, rRemovalLabel, rADeactivateLabel, rDisplayLabel;
    private JTextField rPlanId, rPriceField, pDueIDField, pMAField, pPDAField, pPaidAmtField, pCDField, pRPMField, pDisplayField, pTrainerField, acField, deacField, resField, rIdField, rFullNameField, rLocationField, rPhoneNumberField, rEmailField, rReferralField, rAMAField, rAUPField, rARRMField, rDeactivateField, rDisplayField;
```

```
        private JComboBox rPlanPrice, rDYear, rDMonth, rDDay, rMembershipStartYear, rMembershipStartMonth, rMembershipStartDay, rPlan, pDYear, pDMonth, pDDay, pMSDYear, pMSDMonth, pMSDDay;
    private JRadioButton rMale, rFemale, rOthers;
    private JTextArea pDisplayArea, pRRArea, rRemovalArea;

    //Color Declaration
    Color lightBlue=new Color(111,230,252);
    Color yellow=new Color(255,250,141);

    // Frame and Panels
      JPanel rMSDPanel, pTPanel, pMPanel, pFPanel, pAPanel, pFTPanel, pFFPanel, pATPanel, pAFPanel, pGenderPanel;
       JLabel pTLabel, pFTLabel, pIdLabel, pNameLabel, pGenderLabel, pEmailLabel, pAddressLabel, pPhoneLabel, pATLabel;
    JTextField pIdField, pNameField, pAddressField, pEmailField, pPhoneField;
    JRadioButton pMaleRadio, pFemaleRadio, pOthersRadio;
    ButtonGroup pGenderGroup;
    JButton pRegisterButton, pClearButton, pAttendanceButton;

    /**
 * Gym Management System GUI interface handling member operations.
 * Provides functionality for managing both Regular and Premium gym members,
 * including registration, attendance tracking, membership upgrades, and file I/O operations.
 */
    /** Main list storing all gym members */
    private ArrayList<GymMember> members = new ArrayList<>();
```

```
public ArrayList<GymMember> getMembers() {
    return members;
}

  /**
 * Constructor initializes the main application window
 * and sets up core UI components.
 */
public GymGUI() {
    frame = new JFrame("Gym Management");
    frame.setBounds(0, 0, 700, 700);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setBackground(Color.lightGray);
    frame.setLayout(new BorderLayout());

    // Title Panel
    fTPanel = new JPanel(new FlowLayout());
    fTLabel = new JLabel("Gym Management Software");
    fTLabel.setFont(new Font("Arial", Font.BOLD, 24));
    fTPanel.add(fTLabel);
    frame.add(fTPanel, BorderLayout.NORTH);

    // Button Panel
    fBPanel = new JPanel(new GridLayout(3, 1, 10, 10)); // 3 buttons vertically
    gButton = new JButton("Gym Members");
    gButton.setBackground(lightBlue);
```

```
        gButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                gMGui();
            }
        });

        rButton = new JButton("Regular Gym Members");
        rButton.setBackground(yellow);
        rButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                rMGui();
            }
        });
        pButton = new JButton("Premium Gym Members");
        pButton.setBackground(lightBlue);
        pButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                pMGui();
            }
        });
        fBPanel.add(gButton);
        fBPanel.add(rButton);
        fBPanel.add(pButton);

        frame.add(fBPanel, BorderLayout.CENTER);
        frame.setVisible(true);
    }
```

```
/**
     * Creates the general member management window with activation/deactivation features
 */
public void gMGui() {
    gFrame = new JFrame("Gym Members - Methods");
    gFrame.setExtendedState(JFrame.MAXIMIZED_BOTH);
    gFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    // gFrame.setBounds(0, 0, 700, 700);
    gFrame.setLayout(new BorderLayout());

    // Title Panel
    gTPanel = new JPanel(new FlowLayout());
    sTLabel = new JLabel("Gym Members Actions");
    sTLabel.setFont(new Font("Arial", Font.BOLD, 24));
    gTPanel.add(sTLabel);
    gFrame.add(gTPanel, BorderLayout.NORTH);

    // form Panel with gridbaglayout
    gFPanel = new JPanel(new GridBagLayout());
    gFPanel.setBackground(lightBlue);
    GridBagConstraints sFGbc = new GridBagConstraints();
    sFGbc.insets = new Insets(10, 10, 10, 10);

    acLabel = new JLabel("Activate");
    sFGbc.gridx = 0;
```

```
        sFGbc.gridy = 0;
        gFPanel.add(acLabel, sFGbc);

        acField = new JTextField(30);
        sFGbc.gridx = 1;
        sFGbc.gridy = 0;
        gFPanel.add(acField, sFGbc);

        acButton = new JButton("Activate");
        sFGbc.gridx = 2;
        sFGbc.gridy = 0;
        acButton.setBackground(yellow);
        acButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    int memberId = Integer.parseInt(acField.getText().trim());
                    boolean idisThere = false;

                    for (GymMember member : members) {
                        if (member.getId() == memberId) {
                            member.activateMembership();
                            idisThere = true;
                                JOptionPane.showMessageDialog(null, "Member Activated
Successfully", "Success", JOptionPane.PLAIN_MESSAGE);
                            break;
                        }
                    }
```

```
                if (!idisThere) {
                        JOptionPane.showMessageDialog(null, "Member Id not not Registered!!!", "Error", JOptionPane.ERROR_MESSAGE);
                }
                 // You can add your logic here, like searching in the ArrayList and activating a member
            } catch (NumberFormatException exception) {
                 JOptionPane.showMessageDialog(null, "Enter a valid ID", "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    });

    acButton.setSize(new Dimension(10, 20));
    gFPanel.add(acButton, sFGbc);

    deacLabel = new JLabel("Deactivate Membership");
    sFGbc.gridx = 0;
    sFGbc.gridy = 1;
    gFPanel.add(deacLabel, sFGbc);
    gFrame.add(gFPanel, BorderLayout.CENTER);

    deacField = new JTextField(30);

    sFGbc.gridx = 1;
    sFGbc.gridy = 1;
    gFPanel.add(deacField, sFGbc);
```

```
deacButton = new JButton("Deactivate");
deacButton.setSize(new Dimension(10, 20));
deacButton.setBackground(yellow);
sFGbc.gridx = 2;
sFGbc.gridy = 1;
deacButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            int memberId = Integer.parseInt(deacField.getText().trim());
            boolean idisThere = false;

            for (GymMember member : members) {
                if (member.getId() == memberId) {
                    member.deactivateMembership();
                    idisThere = true;
                    JOptionPane.showMessageDialog(null, "Member De Activated
Successfully", "Success", JOptionPane.PLAIN_MESSAGE);
                    break;
                }
            }
            if (!idisThere) {
                    JOptionPane.showMessageDialog(null, "Member Id not not
Registered!!!", "Error", JOptionPane.ERROR_MESSAGE);
            }
        } catch (NumberFormatException exception) {
```

```
                    JOptionPane.showMessageDialog(null, "Enter a valid ID", "Error",
JOptionPane.ERROR_MESSAGE);
                }
            }
        });
        gFPanel.add(deacButton, sFGbc);

        resLabel = new JLabel("Reset Member");
        sFGbc.gridx = 0;
        sFGbc.gridy = 2;
        gFPanel.add(resLabel, sFGbc);

        resField = new JTextField(30);
        sFGbc.gridx = 1;
        sFGbc.gridy = 2;
        gFPanel.add(resField, sFGbc);

        resButton = new JButton("Reset");
        resButton.setSize(new Dimension(10, 20));
        resButton.setBackground(yellow);
        sFGbc.gridx = 2;
        sFGbc.gridy = 2;
        resButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    int memberId = Integer.parseInt(resField.getText().trim());
                    boolean idisThere = false;
```

```
                for (GymMember member : members) {
                    if (member.getId() == memberId) {
                        member.resetMember();
                        idisThere = true;
                            JOptionPane.showMessageDialog(null, "Member Reset Successfully", "Success", JOptionPane.PLAIN_MESSAGE);
                        break;
                    }
                }
                if (!idisThere) {
                        JOptionPane.showMessageDialog(null, "Member Id not Registered!!!", "Error", JOptionPane.ERROR_MESSAGE);
                }
                 // You can add your logic here, like searching in the ArrayList and activating a member
            } catch (NumberFormatException exception) {
                 JOptionPane.showMessageDialog(null, "Enter a valid ID", "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    });
    gFPanel.add(resButton,sFGbc);

    saveButton=new JButton("Save to file");
    saveButton.setBackground(yellow);
    saveButton.setSize(new Dimension(10, 20));
```

```
        sFGbc.gridx = 1;
        sFGbc.gridy = 3;
        saveButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        writeMembersToFile(members);
    }
});

        gFPanel.add(saveButton, sFGbc);

        readButton=new JButton("Read from file");
        readButton.setBackground(yellow);
        readButton.setSize(new Dimension(10, 20));
        sFGbc.gridx = 1;
        sFGbc.gridy = 4;
        readButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
       File file = new File("MemberDetails.txt");

    if (!file.exists()) {
            JOptionPane.showMessageDialog(null,  "MemberDetails.txt  not  found!",
"Error", JOptionPane.ERROR_MESSAGE);
        return;
    }

    try {
        Scanner scanner = new Scanner(file);
```

```
        String content = "";

        while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            content = content + line + "\n";
        }

        scanner.close();

        JTextArea textArea = new JTextArea(content);
        textArea.setFont(new Font("Monospaced", Font.PLAIN, 12));
        textArea.setEditable(false);

        JScrollPane scrollPane = new JScrollPane(textArea);
        scrollPane.setPreferredSize(new Dimension(900, 400));

            JOptionPane.showMessageDialog(null,  scrollPane,  "Member  Details",
JOptionPane.INFORMATION_MESSAGE);

    } catch (Exception ap) {
        JOptionPane.showMessageDialog(null, "Error reading MemberDetails.txt!",
"Error", JOptionPane.ERROR_MESSAGE);
    }
}

});
```

```
        gFPanel.add(readButton, sFGbc);

        gFrame.setVisible(true);

    }

    /**
     * Creates the Regular Member management interface with registration and
task features
     */
    public void rMGui() {
        rFrame = new JFrame("Gym Member - Regular Members - Methods");
        rFrame.setExtendedState(JFrame.MAXIMIZED_BOTH);
        // rFrame.setBounds(700, 0, 700, 700);
        rFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        rFrame.setBackground(Color.lightGray);
        rFrame.setLayout(new BorderLayout());

        rTPanel = new JPanel();
        rTLabel = new JLabel("Regular Member Methods");
        rTLabel.setFont(new Font("Arial", Font.BOLD, 24));
        rTPanel.add(rTLabel);

        rMPanel = new JPanel();
        rMPanel.setLayout(new GridLayout(1, 2));
```

```
rFPanel = new JPanel();
rFPanel.setLayout(new BorderLayout());
// rFPanel.setBackground(Color.gray);
rMPanel.add(rFPanel);

rAPanel = new JPanel();
rAPanel.setLayout(new BorderLayout());
rAPanel.setBackground(lightBlue);
rMPanel.add(rAPanel);

rFTPanel = new JPanel();
rFTPanel.setBackground(yellow);
rFTPanel.setLayout(new FlowLayout());
rFTLabel = new JLabel("Regular Member Registration");
rFTLabel.setFont(new Font("Arial", Font.BOLD, 18));
rFTPanel.add(rFTLabel);
rFPanel.add(rFTPanel, BorderLayout.NORTH);

rFFPanel = new JPanel();
rFFPanel.setBackground(lightBlue);
rFFPanel.setLayout(new GridBagLayout());
GridBagConstraints rFGbc = new GridBagConstraints();
rFGbc.insets = new Insets(10, 10, 10, 10);

rIdLabel = new JLabel("ID");
rFGbc.gridx = 0;
```

```
        rFGbc.gridy = 0;
        rFFPanel.add(rIdLabel, rFGbc);

        rIdField = new JTextField(20);
        rFGbc.gridx = 1;
        rFGbc.gridy = 0;
        rFFPanel.add(rIdField, rFGbc);

        rFullNameLabel = new JLabel("Full Name");
        rFGbc.gridx = 0;
        rFGbc.gridy = 1;
        rFFPanel.add(rFullNameLabel, rFGbc);

        rFullNameField = new JTextField(20);
        rFGbc.gridx = 1;
        rFGbc.gridy = 1;
        rFFPanel.add(rFullNameField, rFGbc);

        // Location
        rLocationLabel = new JLabel("Location");
        rFGbc.gridx = 0;
        rFGbc.gridy = 2;
        rFFPanel.add(rLocationLabel, rFGbc);

        rLocationField = new JTextField(20);
        rFGbc.gridx = 1;
        rFGbc.gridy = 2;
```

```
        rFFPanel.add(rLocationField, rFGbc);

// Phone Number
        rPhoneNumberLabel = new JLabel("Phone Number");
        rFGbc.gridx = 0;
        rFGbc.gridy = 3;
        rFFPanel.add(rPhoneNumberLabel, rFGbc);

        rPhoneNumberField = new JTextField(20);
        rFGbc.gridx = 1;
        rFGbc.gridy = 3;
        rFFPanel.add(rPhoneNumberField, rFGbc);

        // Email
        rEmailLabel = new JLabel("Email");
        rFGbc.gridx = 0;
        rFGbc.gridy = 4;
        rFFPanel.add(rEmailLabel, rFGbc);

        rEmailField = new JTextField(20);
        rFGbc.gridx = 1;
        rFGbc.gridy = 4;
        rFFPanel.add(rEmailField, rFGbc);

        rDobLabel = new JLabel("Date of Birth");
        rFGbc.gridx = 0;
        rFGbc.gridy = 5;
```

```
        rFFPanel.add(rDobLabel, rFGbc);

        rDPanel = new JPanel();
        rFGbc.gridx = 1;
        rFGbc.gridy = 5;

        rDYear = new JComboBox<>();
        for (int i = 2025; i >= 1875; i--) {
            rDYear.addItem(String.valueOf(i));
        }

        rDMonth = new JComboBox<>();
        for (int i = 1; i <= 12; i++) {
            rDMonth.addItem(String.valueOf(i));
        }

        rDDay = new JComboBox<>();
        for (int i = 1; i <= 31; i++) {
            rDDay.addItem(String.valueOf(i));
        }
        rDPanel.add(rDYear);
        rDPanel.add(rDMonth);
        rDPanel.add(rDDay);
        rFFPanel.add(rDPanel, rFGbc);

        rGenderLabel = new JLabel("Gender");
        rFGbc.gridx = 0;
```

```
        rFGbc.gridy = 6;
        rFFPanel.add(rGenderLabel, rFGbc);

        rGenderPanel = new JPanel();
        rFGbc.gridx = 1;
        rFGbc.gridy = 6;
        rMale = new JRadioButton("Male");
        rFemale = new JRadioButton("Female");
        rOthers = new JRadioButton("Others");

        ButtonGroup genderGroup = new ButtonGroup();
        genderGroup.add(rMale);
        genderGroup.add(rFemale);
        genderGroup.add(rFemale);
        genderGroup.add(rOthers);

        rGenderPanel.add(rMale);
        rGenderPanel.add(rFemale);
        rGenderPanel.add(rOthers);
        rFFPanel.add(rGenderPanel, rFGbc);

        rMembershipStartDateLabel = new JLabel("Membership Start Date");
        rFGbc.gridx = 0;
        rFGbc.gridy = 7;
        rFFPanel.add(rMembershipStartDateLabel, rFGbc);

// Membership Start Year
```

```
        rMSDPanel = new JPanel();
        rFGbc.gridx = 1;
        rFGbc.gridy = 7;
        rMembershipStartYear = new JComboBox<>();
        for (int i = 2025; i >= 2000; i--) {
            rMembershipStartYear.addItem(String.valueOf(i));
        }
        rMSDPanel.add(rMembershipStartYear);

// Membership Start Month
        rMembershipStartMonth = new JComboBox<>();
        for (int i = 1; i <= 12; i++) {
            rMembershipStartMonth.addItem(String.valueOf(i));
        }
        rMSDPanel.add(rMembershipStartMonth);

// Membership Start Day
        rMembershipStartDay = new JComboBox<>();
        for (int i = 1; i <= 31; i++) {
            rMembershipStartDay.addItem(String.valueOf(i));
        }
        rMSDPanel.add(rMembershipStartDay);
        rFFPanel.add(rMSDPanel, rFGbc);

        rReferralLabel = new JLabel("Referral Source");
        rFGbc.gridx = 0;
        rFGbc.gridy = 8;
```

```
        rFFPanel.add(rReferralLabel, rFGbc);

        rReferralField = new JTextField(20);
        rFGbc.gridx = 1;
        rFGbc.gridy = 8;
        rFFPanel.add(rReferralField, rFGbc);

        rAddButton = new JButton("Add Regular Member");
        rAddButton.setBackground(yellow);
        rFGbc.gridx = 0;
        rFGbc.gridy = 9;
        rFGbc.gridwidth = 2;
        rFGbc.fill = GridBagConstraints.HORIZONTAL;
        rFFPanel.add(rAddButton, rFGbc);

        rAddButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    // Retrieve input values
                    int id = Integer.parseInt(rIdField.getText().trim());
                    String fullName = rFullNameField.getText().trim();
                    String location = rLocationField.getText().trim();
                    String phone = rPhoneNumberField.getText().trim();
                    String email = rEmailField.getText().trim();
                                String dob = rDYear.getSelectedItem() + "-" +
rDMonth.getSelectedItem() + "-" + rDDay.getSelectedItem();
                    String gender = "";
```

```
            if (rMale.isSelected()) {
                gender = "Male";
            } else if (rFemale.isSelected()) {
                gender = "Female";
            } else if (rOthers.isSelected()) {
                gender = "Others";
            }

             String startDate = rMembershipStartYear.getSelectedItem() + "-" +
rMembershipStartMonth.getSelectedItem() + "-" +
rMembershipStartDay.getSelectedItem();
            String referral = rReferralField.getText().trim();

            // Check if all fields are filled
                    if (rIdField.getText().isEmpty() || fullName.isEmpty() ||
location.isEmpty() || phone.isEmpty() || email.isEmpty() || dob.isEmpty() ||
gender.isEmpty() || startDate.isEmpty() || referral.isEmpty()) {
                JOptionPane.showMessageDialog(null, "Please fill all the fields",
"Error", JOptionPane.ERROR_MESSAGE);
                return; // Return early if any field is empty
            }

            // Check if ID already exists in members list
            for (GymMember member : members) {
                if (member.getId() == id) {
                    JOptionPane.showMessageDialog(null, "ID already exists.
Please use a unique ID.", "Error", JOptionPane.ERROR_MESSAGE);
```

```
                    return; // Return early if ID is not unique
                }
            }

            // Create and add new RegularMember to list
              RegularMember rm = new RegularMember(id, fullName, location,
phone, email, dob, gender, startDate, referral);
            members.add(rm);

              JOptionPane.showMessageDialog(null, "Regular Member added
successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
        } catch (NumberFormatException ex) {
             JOptionPane.showMessageDialog(null, "Invalid ID! Please enter a
valid number.", "Error", JOptionPane.ERROR_MESSAGE);
        }
      }
    });

    rClearButton = new JButton("Clear");
    rFGbc.gridx = 2;
    rFGbc.gridy = 9;
    rFGbc.gridwidth = 2;
    rFGbc.fill = GridBagConstraints.HORIZONTAL;
    rClearButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        rIdField.setText("");
        rFullNameField.setText("");
```

```
            rLocationField.setText("");
            rPhoneNumberField.setText("");
            rEmailField.setText("");
            rReferralField.setText("");
            genderGroup.clearSelection();
            rDYear.setSelectedIndex(0); // index 0 selects the value which is in the
0th index of the list
            rDMonth.setSelectedIndex(0);
            rDDay.setSelectedIndex(0);
            rMembershipStartYear.setSelectedIndex(0);
            rMembershipStartMonth.setSelectedIndex(0);
            rMembershipStartDay.setSelectedIndex(0);
        }
    });

    rFFPanel.add(rClearButton, rFGbc);
    rFPanel.add(rFFPanel, BorderLayout.CENTER);

    rATPanel = new JPanel();
    rATPanel.setLayout(new FlowLayout());
    rATPanel.setBackground(lightBlue);
    rATLabel = new JLabel("Regular Member Task");
    rATLabel.setFont(new Font("Arial", Font.BOLD, 18));
    rATPanel.add(rATLabel);
    rAPanel.add(rATPanel, BorderLayout.NORTH);

    rAFPanel = new JPanel();
```

```
        rAFPanel.setLayout(new GridBagLayout());
        rAFPanel.setBackground(yellow);
        rAPanel.add(rAFPanel, BorderLayout.CENTER);
        GridBagConstraints rAGbc = new GridBagConstraints();
        rAGbc.insets = new Insets(10, 10, 10, 10);

        rAMALabel = new JLabel("Mark Attendance");
        rAGbc.gridx = 0;
        rAGbc.gridy = 0;
        rAFPanel.add(rAMALabel, rAGbc);

        rAMAField = new JTextField(20);
        rAGbc.gridx = 1;
        rAGbc.gridy = 0;
        rAFPanel.add(rAMAField, rAGbc);

        rMarkButton = new JButton("Mark");
        rAGbc.gridx = 1;
        rAGbc.gridy = 1;
        rAGbc.gridwidth = 2;
        rAGbc.fill = GridBagConstraints.HORIZONTAL;
        rMarkButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Validate field first
                if (rAMAField.getText().trim().isEmpty()) {
                        JOptionPane.showMessageDialog(null, "Member ID cannot be
empty!", "Input Error", JOptionPane.WARNING_MESSAGE);
```

```
            return;
        }

        try {
            int id = Integer.parseInt(rAMAField.getText().trim());
            boolean found = false;

            for (GymMember member : members) {
                if (member.getId() == id) {
                    found = true;

                    if (member instanceof PremiumMember) {
                        JOptionPane.showMessageDialog(null, "This ID belongs to
a Premium Member.", "Error", JOptionPane.ERROR_MESSAGE);
                    }
                    else if (member instanceof RegularMember) {
                        if (member.isActiveStatus()) {
                            member.markAttendance();
                                JOptionPane.showMessageDialog(null, "Attendance
Marked", "Success", JOptionPane.INFORMATION_MESSAGE);
                        } else {
                            JOptionPane.showMessageDialog(null, "Member ID is not
activated yet.", "Info", JOptionPane.WARNING_MESSAGE);
                        }
                    }
                    return; // Stop after matching member is found
                }
```

```
                }

                if (!found) {
                        JOptionPane.showMessageDialog(null, "Member ID not
registered.", "Info", JOptionPane.ERROR_MESSAGE);
                }

            } catch (NumberFormatException ex) {
              JOptionPane.showMessageDialog(null, "Please enter a valid numeric
Member ID.", "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    });

    rAFPanel.add(rMarkButton, rAGbc);

    rAUPLabel = new JLabel("Upgrade Plan");
    rAGbc.gridx = 0;
    rAGbc.gridy = 2;
    rAFPanel.add(rAUPLabel, rAGbc);

    rAUPField = new JTextField(20);
    rAGbc.gridx = 1;
    rAGbc.gridy = 2;
    rAFPanel.add(rAUPField, rAGbc);

    rUpgradeButton = new JButton("Upgrade");
```

```
        rAGbc.gridx = 1;
        rAGbc.gridy = 3;
        rUpgradeButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    int id = Integer.parseInt(rAUPField.getText().trim());
                    boolean isThere = false;

                    for (GymMember member : members) {
                        if (member.getId() == id) {
                            isThere = true; // Move this up to prevent false "not found" alert

                            if (member instanceof RegularMember) {
                             RegularMember regularMember = (RegularMember) member;
                                String selectedPlan = (String) rPlan.getSelectedItem();

                                // Get the message from the upgradePlan method
                             String message = regularMember.upgradePlan(selectedPlan);

                                // Show the message returned by upgradePlan
                                JOptionPane.showMessageDialog(null, message, "Upgrade
Status", JOptionPane.INFORMATION_MESSAGE);
                            } else {
                                 JOptionPane.showMessageDialog(null, "This id belongs to
Premium Member", "Error", JOptionPane.ERROR_MESSAGE);
                            }
                            break;
```

```
                }
            }

            if (!isThere) {
                    JOptionPane.showMessageDialog(null, "Member ID not
Registered.", "Info", JOptionPane.ERROR_MESSAGE);
            }

        } catch (Exception exception) {
          JOptionPane.showMessageDialog(null, "Please input a valid Member
ID", "Error", JOptionPane.ERROR_MESSAGE);
        }
      }
    });
    rAFPanel.add(rUpgradeButton, rAGbc);
    rPlan = new JComboBox<>();
    rPlan.addItem("Basic");
    rPlan.addItem("Standard");
    rPlan.addItem("Deluxe");
    rAGbc.gridx = 2;
    rAGbc.gridy = 2;
    rAFPanel.add(rPlan, rAGbc);

    rARRMLabel = new JLabel("Revert Member");
    rAGbc.gridx = 0;
    rAGbc.gridy = 4;
    rAFPanel.add(rARRMLabel, rAGbc);
```

```
        rARRMField = new JTextField(20);
        rAGbc.gridx = 1;
        rAGbc.gridy = 4;
        rAGbc.gridwidth = 2;
        rAGbc.fill = GridBagConstraints.HORIZONTAL;
        rAFPanel.add(rARRMField, rAGbc);

        rRemovalArea = new JTextArea(5, 20);
        rAGbc.gridx = 1;
        rAGbc.gridy = 5;
        rAGbc.fill = GridBagConstraints.HORIZONTAL;
        String removalReason = rRemovalArea.getText();
        rAFPanel.add(rRemovalArea, rAGbc);

        rRevertButton = new JButton("Revert");
        rAGbc.gridx = 1;
        rAGbc.gridy = 6;
        rAGbc.gridwidth = 2;
        rAGbc.fill = GridBagConstraints.HORIZONTAL;
        rRevertButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                                        if     (!rRemovalArea.getText().isEmpty()     &&
!rARRMField.getText().trim().isEmpty()) {
                try {
                    int id = Integer.parseInt(rARRMField.getText().trim());
                    boolean isThere = false;
```

```
                for (GymMember member : members) {
                    if (member.getId() == id) {
                        isThere = true; // Move this up to prevent false "not found"
alert

                        if (member instanceof RegularMember) {
                            RegularMember regularMember = (RegularMember)
member;
                            regularMember.revertRegularMember(removalReason);
                            JOptionPane.showMessageDialog(null, "Member
Reverted Successfully", "Success", JOptionPane.INFORMATION_MESSAGE);
                        }
                        else if (member instanceof PremiumMember) {
                            PremiumMember premium = (PremiumMember) member;
                            premium.revertPremiumMember();
                        JOptionPane.showMessageDialog(null, "Member Reverted
Successfully", "Success", JOptionPane.INFORMATION_MESSAGE);
                        }
                        break;
                    }
                }

                if (!isThere) {
                    JOptionPane.showMessageDialog(null, "Member ID not
Registered.", "Info", JOptionPane.ERROR_MESSAGE);
                }
```

```
                } catch (NumberFormatException exception) {
                        JOptionPane.showMessageDialog(null, "Please input a valid
Member ID", "Error", JOptionPane.ERROR_MESSAGE);
                }
            } else {
                JOptionPane.showMessageDialog(null, "Please input both the field
to continue", "Warning", JOptionPane.WARNING_MESSAGE);
            }
        }
    });
    rAFPanel.add(rRevertButton, rAGbc);

    rRemovalLabel = new JLabel("Removal Reason");
    rAGbc.gridx = 0;
    rAGbc.gridy = 5;
    rAFPanel.add(rRemovalLabel, rAGbc);

    rDisplayLabel = new JLabel("Display Member");
    rAGbc.gridx = 0;
    rAGbc.gridy = 7;
    rAFPanel.add(rDisplayLabel, rAGbc);

    rDisplayField = new JTextField(20); // 20 columns wide
    rAGbc.gridx = 1;                    // next to the label
    rAGbc.gridy = 7;
    rAFPanel.add(rDisplayField, rAGbc);
```

```
        rDisplayButton = new JButton("Display");
        rAGbc.gridx = 1;                    // next to the text field
        rAGbc.gridy = 8;
        rDisplayButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String input = rDisplayField.getText().trim();

                if (!input.isEmpty()) {
                    try {
                        int id = Integer.parseInt(input);
                        boolean found = false;

                        for (GymMember member : members) {
                            if (member.getId() == id) {
                                found = true;

                                if (member instanceof RegularMember) {
                                        RegularMember regularMember = (RegularMember)
member;
                                    String message = regularMember.display();
                                 JOptionPane.showMessageDialog(null, message, "Member
Information", JOptionPane.INFORMATION_MESSAGE);
                                }
                                else{
                                    JOptionPane.showMessageDialog(null, "This id belongs
to a Premium Member", "NO!!", JOptionPane.ERROR_MESSAGE);
                                }
```

```
                    break;
                }
            }

            if (!found) {
                JOptionPane.showMessageDialog(null, "Member ID not registered.", "Info", JOptionPane.ERROR_MESSAGE);
            }

        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(null, "Please enter a valid numeric Member ID", "Error", JOptionPane.ERROR_MESSAGE);
        }
    } else {
        JOptionPane.showMessageDialog(null, "Please enter a Member ID to continue", "Warning", JOptionPane.WARNING_MESSAGE);
    }
  }
});

rAFPanel.add(rDisplayButton, rAGbc);

rPlanLabel = new JLabel("Member Id");
rAGbc.gridx = 0;
rAGbc.gridy = 9;
rAFPanel.add(rPlanLabel, rAGbc);
```

```
        rPlanId = new JTextField(20);
        rAGbc.gridx = 1;
        rAGbc.gridy = 9;
        rAFPanel.add(rPlanId, rAGbc);

        rPlanPrice = new JComboBox<>();
        rAGbc.gridx = 2;
        rAGbc.gridy = 9;
        rPlanPrice.addItem("Basic");
        rPlanPrice.addItem("Standard");
        rPlanPrice.addItem("Deluxe");
        rAFPanel.add(rPlanPrice, rAGbc);

        rPriceField = new JTextField(20);
        rAGbc.gridx = 4;
        rAGbc.gridy = 9;
        rAFPanel.add(rPriceField, rAGbc);

        rPlanPriceButton = new JButton("Plan and price");
        rAGbc.gridx = 1;
        rAGbc.gridy = 10;
        rPlanPriceButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    // Parse the member ID from text field
                    int id = Integer.parseInt(rPlanId.getText().trim());
                    boolean found = false;
```

```
            for (GymMember member : members) {
                if (member.getId() == id) {
                    found = true;

                    if (member instanceof RegularMember) {
                     RegularMember regularMember = (RegularMember) member;

                       // Get member's current plan
                       String currentPlan = regularMember.getPlan();

                       // Set the selected item in the JComboBox
                       rPlanPrice.setSelectedItem(currentPlan);
                       rPlanPrice.setEnabled(false);  // make it non-editable

                       // Get price for the plan and show it
                       double price = regularMember.getPlanPrice(currentPlan);
                       rPriceField.setText(String.valueOf(price));
                       rPriceField.setEditable(false);

                       JOptionPane.showMessageDialog(null,
                              "Plan and Price loaded for Member ID: " + id,
                              "Info", JOptionPane.INFORMATION_MESSAGE);
                    } else {
                        JOptionPane.showMessageDialog(null, "This ID belongs to
a Premium Member.", "Info", JOptionPane.ERROR_MESSAGE);
                    }
```

```
                    break;
                }
            }

            if (!found) {
                JOptionPane.showMessageDialog(null,
                        "Member ID not registered.",
                        "Info", JOptionPane.ERROR_MESSAGE);
            }

        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(null,
                    "Please enter a valid numeric Member ID.",
                    "Error", JOptionPane.WARNING_MESSAGE);
        }
    }
});

rAFPanel.add(rPlanPriceButton, rAGbc);

rMPanel.add(rAPanel);
rFrame.add(rTPanel, BorderLayout.NORTH);
rFrame.add(rMPanel, BorderLayout.CENTER);

rFrame.setVisible(true);
```

```
    }

    /**
     * Creates the Premium Member management interface with payment and
trainer features
     */
    public void pMGui() {
        pFrame = new JFrame("Gym Member - Premium Members - Methods");
        pFrame.setExtendedState(JFrame.MAXIMIZED_BOTH);
        pFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        pFrame.setBackground(Color.lightGray);
        pFrame.setLayout(new BorderLayout());

        pTPanel = new JPanel();
        pTLabel = new JLabel("Premium Member Methods");
        pTLabel.setFont(new Font("Arial", Font.BOLD, 24));
        pTPanel.add(pTLabel);

        pMPanel = new JPanel();
        pMPanel.setLayout(new GridLayout(1, 2));

        pFPanel = new JPanel();
        pFPanel.setLayout(new BorderLayout());
        pMPanel.add(pFPanel);

        pAPanel = new JPanel();
        pAPanel.setLayout(new BorderLayout());
```

```
pAPanel.setBackground(lightBlue);
pMPanel.add(pAPanel);

pFTPanel = new JPanel();
pFTPanel.setBackground(yellow);
pFTPanel.setLayout(new FlowLayout());
pFTLabel = new JLabel("Premium Member Registration");
pFTLabel.setFont(new Font("Arial", Font.BOLD, 18));
pFTPanel.add(pFTLabel);
pFPanel.add(pFTPanel, BorderLayout.NORTH);

pFFPanel = new JPanel();
pFFPanel.setBackground(lightBlue);
pFFPanel.setLayout(new GridBagLayout());
GridBagConstraints pFGbc = new GridBagConstraints();
pFGbc.insets = new Insets(10, 10, 10, 10);

pIdLabel = new JLabel("ID");
pFGbc.gridx = 0;
pFGbc.gridy = 0;
pFFPanel.add(pIdLabel, pFGbc);

pIdField = new JTextField(20);
pFGbc.gridx = 1;
pFGbc.gridy = 0;
pFFPanel.add(pIdField, pFGbc);
```

```
pNameLabel = new JLabel("Name");
pFGbc.gridx = 0;
pFGbc.gridy = 1;
pFFPanel.add(pNameLabel, pFGbc);

pNameField = new JTextField(20);
pFGbc.gridx = 1;
pFGbc.gridy = 1;
pFFPanel.add(pNameField, pFGbc);

pAddressLabel = new JLabel("Location");
pFGbc.gridx = 0;
pFGbc.gridy = 2;
pFFPanel.add(pAddressLabel, pFGbc);

pAddressField = new JTextField(20);
pFGbc.gridx = 1;
pFGbc.gridy = 2;
pFFPanel.add(pAddressField, pFGbc);

pEmailLabel = new JLabel("Email");
pFGbc.gridx = 0;
pFGbc.gridy = 4;
pFFPanel.add(pEmailLabel, pFGbc);

pEmailField = new JTextField(20);
pFGbc.gridx = 1;
```

```
pFGbc.gridy = 4;
pFFPanel.add(pEmailField, pFGbc);

pPhoneLabel = new JLabel("Phone No");
pFGbc.gridx = 0;
pFGbc.gridy = 3;
pFFPanel.add(pPhoneLabel, pFGbc);

pPhoneField = new JTextField(20);
pFGbc.gridx = 1;
pFGbc.gridy = 3;
pFFPanel.add(pPhoneField, pFGbc);

pGenderLabel = new JLabel("Gender");
pFGbc.gridx = 0;
pFGbc.gridy = 7;
pFFPanel.add(pGenderLabel, pFGbc);

pGenderPanel = new JPanel();
pGenderGroup = new ButtonGroup();
pMaleRadio = new JRadioButton("Male");
pFemaleRadio = new JRadioButton("Female");
pOthersRadio = new JRadioButton("Others");
pGenderGroup.add(pMaleRadio);
pGenderGroup.add(pFemaleRadio);
pGenderGroup.add(pOthersRadio);
pGenderPanel.add(pMaleRadio);
```

```
        pGenderPanel.add(pFemaleRadio);
        pGenderPanel.add(pOthersRadio);
        pFGbc.gridx = 1;
        pFGbc.gridy = 7;
        pFFPanel.add(pGenderPanel, pFGbc);

        pDobLabel = new JLabel("Date of Birth");
        pFGbc.gridx = 0;
        pFGbc.gridy = 6;
        pFFPanel.add(pDobLabel, pFGbc);

        pDatePanel = new JPanel();
        pFGbc.gridx = 1;
        pFGbc.gridy = 6;
        pDYear = new JComboBox<>();
        for (int i = 2025; i >= 1875; i--) {
            pDYear.addItem(String.valueOf(i));
        }

        pDMonth = new JComboBox<>();
        for (int i = 1; i <= 12; i++) {
            pDMonth.addItem(String.valueOf(i));
        }

        pDDay = new JComboBox<>();
        for (int i = 1; i <= 30; i++) {
            pDDay.addItem(String.valueOf(i));
```

```
        }

        pDatePanel.add(pDYear);
        pDatePanel.add(pDMonth);
        pDatePanel.add(pDDay);
        pFFPanel.add(pDatePanel, pFGbc);

        pMSDLabel = new JLabel("Membership Start Date");
        pFGbc.gridx = 0;
        pFGbc.gridy = 8;
        pFFPanel.add(pMSDLabel, pFGbc);

        pMSDPanel = new JPanel();
        pFGbc.gridx = 1;
        pFGbc.gridy = 8;
        pMSDYear = new JComboBox<>();
        for (int i = 2025; i >= 1875; i--) {
            pMSDYear.addItem(String.valueOf(i));
        }

        pMSDMonth = new JComboBox<>();
        for (int i = 1; i <= 12; i++) {
            pMSDMonth.addItem(String.valueOf(i));
        }

        pMSDDay = new JComboBox<>();
        for (int i = 1; i <= 30; i++) {
```

```
        pMSDDay.addItem(String.valueOf(i));
    }

    pMSDPanel.add(pMSDYear);
    pMSDPanel.add(pMSDMonth);
    pMSDPanel.add(pMSDDay);
    pFFPanel.add(pMSDPanel, pFGbc);

    pTrainerLabel = new JLabel("Trainer");
    pFGbc.gridx = 0;
    pFGbc.gridy = 9;
    pFFPanel.add(pTrainerLabel, pFGbc);
    pFPanel.add(pFFPanel, BorderLayout.CENTER);

    pTrainerField = new JTextField(20);
    pFGbc.gridx = 1;
    pFGbc.gridy = 9;
    pFFPanel.add(pTrainerField, pFGbc);

    pAddButton = new JButton("Add Premium Member");
    pAddButton.setBackground(yellow);
    pFGbc.gridx = 0;
    pFGbc.gridy = 10;
    pFGbc.gridwidth = 2;
    pFGbc.fill = GridBagConstraints.HORIZONTAL;
    pAddButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
```

```
            try {
                // Retrieve input values
                int id = Integer.parseInt(pIdField.getText().trim());
                String fullName = pNameField.getText().trim();
                String location = pAddressField.getText().trim();
                String phone = pPhoneField.getText().trim();
                String email = pEmailField.getText().trim();
                        String dob = pDYear.getSelectedItem() + "-" +
pDMonth.getSelectedItem() + "-" + pDDay.getSelectedItem();
                String gender = "";

                if (pMaleRadio.isSelected()) {
                    gender = "Male";
                } else if (pFemaleRadio.isSelected()) {
                    gender = "Female";
                } else if (pOthersRadio.isSelected()) {
                    gender = "Others";
                }

                    String startDate = pMSDYear.getSelectedItem() + "-" +
pMSDMonth.getSelectedItem() + "-" + pMSDDay.getSelectedItem();
                String trainer = pTrainerField.getText().trim();

                // Check if all fields are filled
                    if (pIdField.getText().isEmpty() || fullName.isEmpty() ||
location.isEmpty() || phone.isEmpty() || email.isEmpty() || dob.isEmpty() ||
gender.isEmpty() || startDate.isEmpty() || trainer.isEmpty()) {
```

```
                    JOptionPane.showMessageDialog(null, "Please fill all the fields",
"Error", JOptionPane.ERROR_MESSAGE);
                    return; // Return early if any field is empty
                }

                // Check if ID already exists in members list
                for (GymMember member : members) {
                    if (member.getId() == id) {
                        JOptionPane.showMessageDialog(null, "ID already exists.
Please use a unique ID.", "Error", JOptionPane.ERROR_MESSAGE);
                        return; // Return early if ID is not unique
                    }
                }

                // Create and add new RegularMember to list
                PremiumMember pm = new PremiumMember(id, fullName, location,
phone, email, dob, gender, startDate, trainer);
                members.add(pm);

                JOptionPane.showMessageDialog(null, "Premium Member added
successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(null, "Invalid ID! Please enter a
valid number.", "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    });
    pFFPanel.add(pAddButton, pFGbc);
```

```
pClearButton = new JButton("Clear");
pFGbc.gridx = 2;
pFGbc.gridy = 10;
pClearButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Clear all text fields
        pIdField.setText("");
        pNameField.setText("");
        pAddressField.setText("");
        pEmailField.setText("");
        pPhoneField.setText("");
        pTrainerField.setText("");

        // Reset JComboBox selections
        pDYear.setSelectedIndex(0);  // Set to the first year
        pDMonth.setSelectedIndex(0); // Set to the first month
        pDDay.setSelectedIndex(0);   // Set to the first day
        pMSDYear.setSelectedIndex(0); // Set to the first year (or reset it)
        pMSDMonth.setSelectedIndex(0); // Set to the first month
        pMSDDay.setSelectedIndex(0);   // Set to the first day

        // Deselect gender radio buttons
        pGenderGroup.clearSelection();
    }
});
pFFPanel.add(pClearButton, pFGbc);
```

```
pFPanel.add(pFFPanel, BorderLayout.CENTER);

pATPanel = new JPanel();
pATPanel.setLayout(new FlowLayout());
pATPanel.setBackground(lightBlue);
pATLabel = new JLabel("Actions");
pATLabel.setFont(new Font("Arial", Font.BOLD, 18));
pATPanel.add(pATLabel);
pAPanel.add(pATPanel, BorderLayout.NORTH);

pAFPanel = new JPanel();
pAFPanel.setLayout(new GridBagLayout());  // Using GridBagLayout
GridBagConstraints pAFGbc = new GridBagConstraints();
pAFGbc.insets = new Insets(10, 10, 10, 10);
pAFPanel.setBackground(yellow);

pMALabel = new JLabel("Mark Attendance");
pAFGbc.gridx = 0;  // Setting the grid position
pAFGbc.gridy = 0;
pAFPanel.add(pMALabel, pAFGbc);  // Add label with GridBagConstraints

pMAField = new JTextField(20);
pAFGbc.gridx = 1;  // Setting the grid position
pAFGbc.gridy = 0;
pAFPanel.add(pMAField, pAFGbc);  // Add label with GridBagConstraints
```

```
        pMarkButton = new JButton("Mark");
        pAFGbc.gridx = 2;
        pAFGbc.gridy = 0;
        pMarkButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e){
                if (pMAField.getText().trim().isEmpty()) {
                    JOptionPane.showMessageDialog(null, "Please input Member ID to
continue","Error",JOptionPane.WARNING_MESSAGE);
                    return;
                }

                try {
                    int id =Integer.parseInt(pMAField.getText().trim());
                    boolean found=false;



                    for (GymMember member : members) {
                        if (member.getId()==id) {
                            found=true;

                            if (member instanceof RegularMember) {
                                JOptionPane.showMessageDialog(null, "This ID belongs to
Regular Member", "Information",JOptionPane.INFORMATION_MESSAGE);
                            }
                            else if (member instanceof PremiumMember) {
                                if (member.isActiveStatus()) {
                                    member.markAttendance();
```

```
                                JOptionPane.showMessageDialog(null, "Attendance
Marked Successfully", "Success",JOptionPane.INFORMATION_MESSAGE);
                        }
                        else{
                            JOptionPane.showMessageDialog(null, "This is is not yet
to be Activated!!!", "Information", JOptionPane.WARNING_MESSAGE);
                        }
                        }
                    return;
                }
            }
                if (!found) {
                 JOptionPane.showMessageDialog(null, "This ID is not registered
till now", "Warning", JOptionPane.ERROR_MESSAGE);
                }

            } catch (NumberFormatException ex) {
              JOptionPane.showMessageDialog(null, "Please enter a valid numeric
id","Error", JOptionPane.WARNING_MESSAGE);
            }
        }
    });



    pAFPanel.add(pMarkButton, pAFGbc);


    pCDLabel = new JLabel("Calculate Discount");
    pAFGbc.gridx = 0;
```

```
        pAFGbc.gridy = 1;
        pAFPanel.add(pCDLabel, pAFGbc);

        pCDField = new JTextField(20);
        pAFGbc.gridx = 1;
        pAFGbc.gridy = 1;
        pAFPanel.add(pCDField, pAFGbc);

        pCalculateButton = new JButton("Calculate");
        pAFGbc.gridx = 2;
        pAFGbc.gridy = 1;
        pCalculateButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String input = pCDField.getText().trim();

                if (!input.isEmpty()) {
                    try {
                        int id = Integer.parseInt(input);
                        boolean found = false;

                        for (GymMember member : members) {
                            if (member.getId() == id) {
                                found = true;

                                if (member instanceof PremiumMember) {
                                    PremiumMember premiumMember = (PremiumMember)
member;
```

```
                    double message = premiumMember.calculateDiscount();
                    JOptionPane.showMessageDialog(null, "Your Discount is: " + message, "Member Information", JOptionPane.INFORMATION_MESSAGE);
                }
                else{
                    JOptionPane.showMessageDialog(null, "This id belongs to Regular Member, Thus No discount!!" , "Member Information", JOptionPane.ERROR_MESSAGE);
                }
                break;
            }
        }

        if (!found) {
                JOptionPane.showMessageDialog(null, "Member ID not registered.", "Info", JOptionPane.ERROR_MESSAGE);
        }

    } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(null, "Please enter a valid numeric Member ID", "Error", JOptionPane.ERROR_MESSAGE);
    }
} else {
    JOptionPane.showMessageDialog(null, "Please enter a Member ID to continue", "Warning", JOptionPane.WARNING_MESSAGE);
}
}
});
```

```
        pAFPanel.add(pCalculateButton, pAFGbc);

        pRPMLabel = new JLabel("Revert Premium Member");
        pAFGbc.gridx = 0;
        pAFGbc.gridy = 2;
        pAFPanel.add(pRPMLabel, pAFGbc);

        pRPMField = new JTextField(20);
        pAFGbc.gridx = 1;
        pAFGbc.gridy = 2;
        pAFPanel.add(pRPMField, pAFGbc);

        pRevertButton = new JButton("Revert");
        pAFGbc.gridx = 2;
        pAFGbc.gridy = 2;
        pRevertButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String input = pRPMField.getText().trim();

                if (!input.isEmpty()) {
                    try {
                        int id = Integer.parseInt(input);
                        boolean found = false;

                        for (GymMember member : members) {
                            if (member.getId() == id) {
                                found = true;
```

```
                    if (member instanceof PremiumMember) {
                        PremiumMember premiumMember = (PremiumMember) member;
                        premiumMember.revertPremiumMember();
                        JOptionPane.showMessageDialog(null, "Premium Member Reverted Successfully", "Member Information", JOptionPane.INFORMATION_MESSAGE);
                    }
                    else{
                        JOptionPane.showMessageDialog(null, "This id belongs to Regular Member", "Information", JOptionPane.ERROR_MESSAGE);
                    }
                    break;
                }
            }

            if (!found) {
                JOptionPane.showMessageDialog(null, "Member ID not registered.", "Info", JOptionPane.ERROR_MESSAGE);
            }

        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(null, "Please enter a valid numeric Member ID", "Error", JOptionPane.WARNING_MESSAGE);
        }
    } else {
```

```
                JOptionPane.showMessageDialog(null, "Please enter a Member ID
to continue", "Warning", JOptionPane.WARNING_MESSAGE);
            }
        }
    });
    pAFPanel.add(pRevertButton, pAFGbc);

    pDisplayLabel = new JLabel("Display");
    pAFGbc.gridx = 0;
    pAFGbc.gridy = 3;
    pAFPanel.add(pDisplayLabel, pAFGbc);

    pDisplayField = new JTextField(20);
    pAFGbc.gridx = 1;
    pAFGbc.gridy = 3;
    pAFPanel.add(pDisplayField, pAFGbc);

    pDisplayButton = new JButton("Display");
    pAFGbc.gridx = 2;
    pAFGbc.gridy = 3;
    pDisplayButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            String input = pDisplayField.getText().trim();

            if (!input.isEmpty()) {
                try {
                    int id = Integer.parseInt(input);
```

```
                boolean found = false;

                for (GymMember member : members) {
                    if (member.getId() == id) {
                        found = true;

                        if (member instanceof PremiumMember) {
                            PremiumMember premiumMember = (PremiumMember)
member;
                            String message = premiumMember.display();
                        JOptionPane.showMessageDialog(null, message, "Member
Information", JOptionPane.INFORMATION_MESSAGE);

                        } else {
                            JOptionPane.showMessageDialog(null, "This ID belongs
to a Regular Member.", "Info", JOptionPane.ERROR_MESSAGE);
                        }
                        break;
                    }
                }

                if (!found) {
                            JOptionPane.showMessageDialog(null, "Member ID not
registered.", "Info", JOptionPane.ERROR_MESSAGE);
                }

            } catch (NumberFormatException ex) {
```

```
                    JOptionPane.showMessageDialog(null, "Please enter a valid
numeric Member ID", "Error", JOptionPane.WARNING_MESSAGE);
                }
            } else {
                JOptionPane.showMessageDialog(null, "Please enter a Member ID
to continue", "Warning", JOptionPane.WARNING_MESSAGE);
            }
        }
    });
    pAFPanel.add(pDisplayButton, pAFGbc);

    pPDALabel = new JLabel("Pay Due Amount");
    pAFGbc.gridx = 0;
    pAFGbc.gridy = 4;
    pAFPanel.add(pPDALabel, pAFGbc);

    pPDAField = new JTextField(20);
    pAFGbc.gridx = 1;
    pAFGbc.gridy = 4;
    pAFPanel.add(pPDAField, pAFGbc);

    pPaidAmtLabel = new JLabel("Paid Amount");
    pAFGbc.gridx = 0;
    pAFGbc.gridy = 5;
    pAFPanel.add(pPaidAmtLabel, pAFGbc);

    pPaidAmtField = new JTextField(20);
```

```
        pAFGbc.gridx = 1;
        pAFGbc.gridy = 5;
        pAFPanel.add(pPaidAmtField, pAFGbc);

        pPayButton = new JButton("Pay");
        pAFGbc.gridx = 2;
        pAFGbc.gridy = 5;
        pPayButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String idInput = pPDAField.getText().trim();
                  String amountInput = pPaidAmtField.getText().trim(); // assume this
JTextField exists

                if (idInput.isEmpty() || amountInput.isEmpty()) {
                    JOptionPane.showMessageDialog(null, "Please enter both Member
ID and Amount to proceed.", "Warning", JOptionPane.WARNING_MESSAGE);
                    return;
                }

                try {
                    int id = Integer.parseInt(idInput);
                    double paidAmount = Double.parseDouble(amountInput);
                    boolean found = false;

                    for (GymMember member : members) {
                        if (member.getId() == id) {
                            found = true;
```

```
                    if (member instanceof PremiumMember) {
                        PremiumMember premiumMember = (PremiumMember) member;
                                                    String message = premiumMember.payDueAmount(paidAmount);
                        JOptionPane.showMessageDialog(null, message, "Payment Info", JOptionPane.INFORMATION_MESSAGE);
                    }
                    else{
                        JOptionPane.showMessageDialog(null, "This id belongs to Regular Member!!! " , "Member Information", JOptionPane.ERROR_MESSAGE);
                    }
                    break;
                }
            }

            if (!found) {
                JOptionPane.showMessageDialog(null, "Member ID not found.", "Error", JOptionPane.ERROR_MESSAGE);
            }

        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(null, "Invalid ID or Amount. Please enter numeric values.", "Error", JOptionPane.ERROR_MESSAGE);
        }
    }
});
```

```
        pAFPanel.add(pPayButton, pAFGbc);

        pAPanel.add(pAFPanel, BorderLayout.CENTER);

        pFrame.add(pTPanel, BorderLayout.NORTH);
        pFrame.add(pMPanel, BorderLayout.CENTER);
        pFrame.setVisible(true);
    }

     /**
     * Writes member data to a formatted text file
     * @param memberList ArrayList containing all gym members
     */
    public void writeMembersToFile(ArrayList<GymMember> memberList) {
    File file = new File("MemberDetails.txt");
    try {
        FileWriter writer = new FileWriter(file); // no 'true' = overwrite mode

        // Header row
         writer.write(String.format("%-5s %-15s %-15s %-15s %-25s %-25s %-10s
%-10s %-10s %-15s %-10s %-17s %-15s %-15s\n",
              "ID", "Name", "Location", "Phone", "Email", "Membership Start Date",
"Plan", "Price",
              "Attendance", "Loyalty Points", "Active", "Referral/Trainer", "Discount",
"Net Paid"));
```

```
        for (GymMember member : memberList) {
            if (member instanceof RegularMember) {
                RegularMember rm = (RegularMember) member;

                writer.write(String.format("%-5d %-15s %-15s %-15s %-25s %-25s %-
10s %-10.2f %-10d %-15.0f %-10s %-17s %-15s %-15s\n",
                    rm.getId(),
                    rm.getName(),
                    rm.getLocation(),
                    rm.getPhone(),
                    rm.getEmail(),
                    rm.getMembershipStartDate(),
                    rm.getPlan(),
                    rm.getPrice(),
                    rm.getAttendance(),
                    rm.getLoyaltyPoints(),
                    rm.isActiveStatus() ? "Yes" : "No",
                    rm.getReferralSource(),
                    "-",   // Discount
                    "-"    // Paid
                ));
            } else if (member instanceof PremiumMember) {
                PremiumMember pm = (PremiumMember) member;

                writer.write(String.format("%-5d %-15s %-15s %-15s %-25s %-25s %-
10s %-10.2f %-10d %-15.0f %-10s %-17s %-15.2f %-15.2f\n",
                    pm.getId(),
```

```
                pm.getName(),
                pm.getLocation(),
                pm.getPhone(),
                pm.getEmail(),
                pm.getMembershipStartDate(),
                "Premium",
                pm.getPremiumCharge(),
                pm.getAttendance(),
                pm.getLoyaltyPoints(),
                pm.isActiveStatus() ? "Yes" : "No",
                pm.getPersonalTrainer(),
                pm.getDiscountAmount(),
                pm.getPaidAmount()
            ));
        }
    }

    writer.close();
                JOptionPane.showMessageDialog(null, "Members saved to
memberDetails.txt", "Success", JOptionPane.INFORMATION_MESSAGE);

  } catch (IOException e) {
        JOptionPane.showMessageDialog(null, "Error writing to file!", "Error",
JOptionPane.ERROR_MESSAGE);
  }
}
  /**
```

```
    * Entry point for the application
     * @param args Command-line arguments (not used)
     */
    public static void main(String[] args) {
        new GymGUI();
    }
}
```