CS4001NI Programming

60% Individual Coursework

2025 Spring

Student Name: Suyog Subedi

London Met ID: 24046680

College ID: NP01AI4A240057

Assignment Due Date: Friday, May 16, 2025

Assignment Submission Date: Friday, May 16, 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.

turn it in.docx

Islington College,Nepal

Document Details

Submission ID
trn:oid:::3618:96185463

Submission Date
May 16, 2025, 11:33 AM GMT+5:45

Download Date
May 16, 2025, 11:35 AM GMT+5:45

File Name
turn it in.docx

File Size
18.9 KB

31 Pages

2,410 Words

14,172 Characters

19% Overall Similarity

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

Match Groups

45 Not Cited or Quoted 19%
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

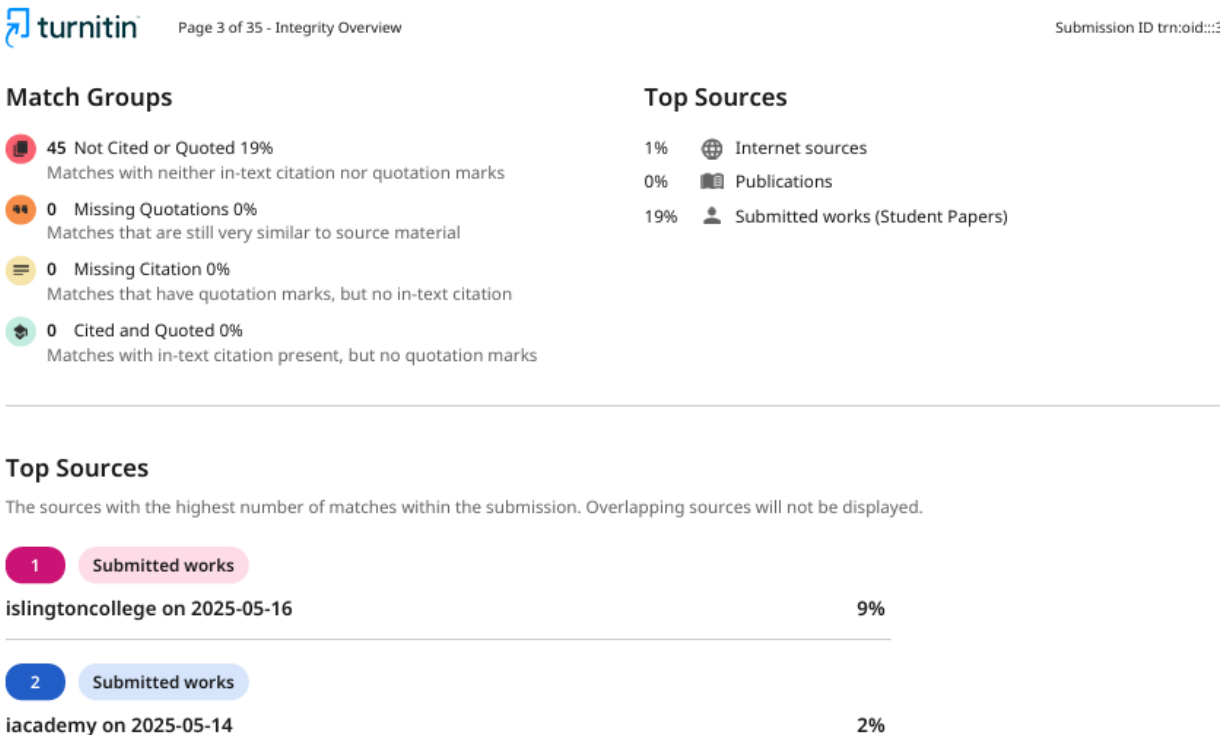
1% Internet sources
0% Publications
19% Submitted works (Student Papers)

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.



turnitin Page 3 of 35 - Integrity Overview Submission ID trn:oid:::

Match Groups

45 Not Cited or Quoted 19%
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
19% Submitted works (Student Papers)

Top Sources

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

1 Submitted works
islingtoncollege on 2025-05-16 9%

2 Submitted works
iacademy on 2025-05-14 2%

Contents

List of Figure

Table of Tables

1 Introduction

This coursework is a programming task that allows us to practice OOP (Object-Oriented Programming) and its features in Java.

In this project, we must create a system to manage gym memberships by making interconnected classes where Gym Member is the parent class (Super class) and Regular Member, Premium Member, and Gymgui are the child classes (Sub class), where we learn to use OOP features like polymorphism, Inheritance, Abstraction and encapsulation.

This system will record attendance, loyalty points, member status, and other real-life gym scenarios.

As interest in gyms rises, there is more demand for such systems. These systems will help gym management record, track, and manage the members, making it easier and more convenient.

1.1 Aims

The aim of this project is to create a simple gym management system that uses Java features to keep records and manage gym members.

1.2 Objectives

- To track Attendance
- To handle payments and upgrades
- To enable user interaction
- To display member records

2 Wireframe

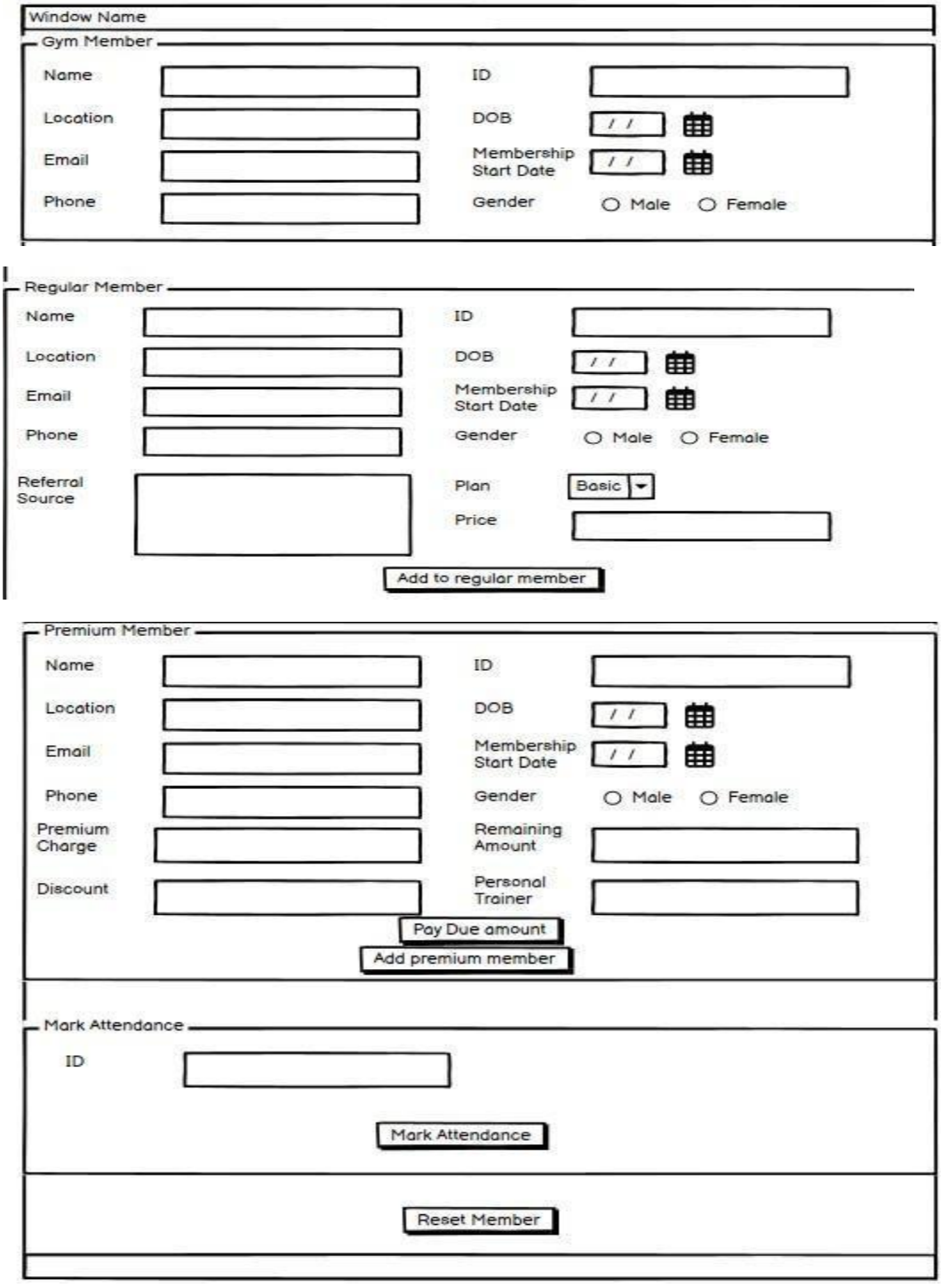


Figure 1: GUI WireFrame

3 Class Diagram

Figure 2: Full class Diagram

Figure 3: GymGUI class diagram

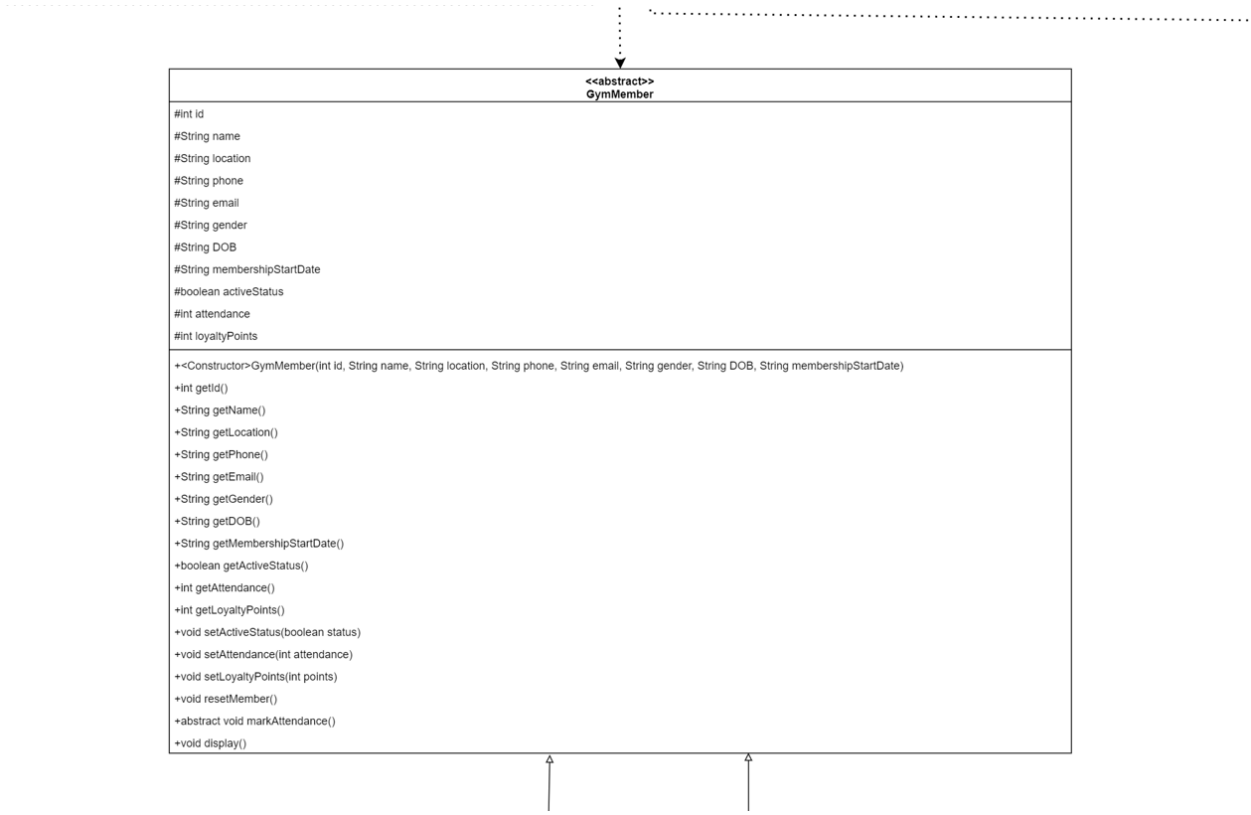


Figure 4: GymMember class diagram

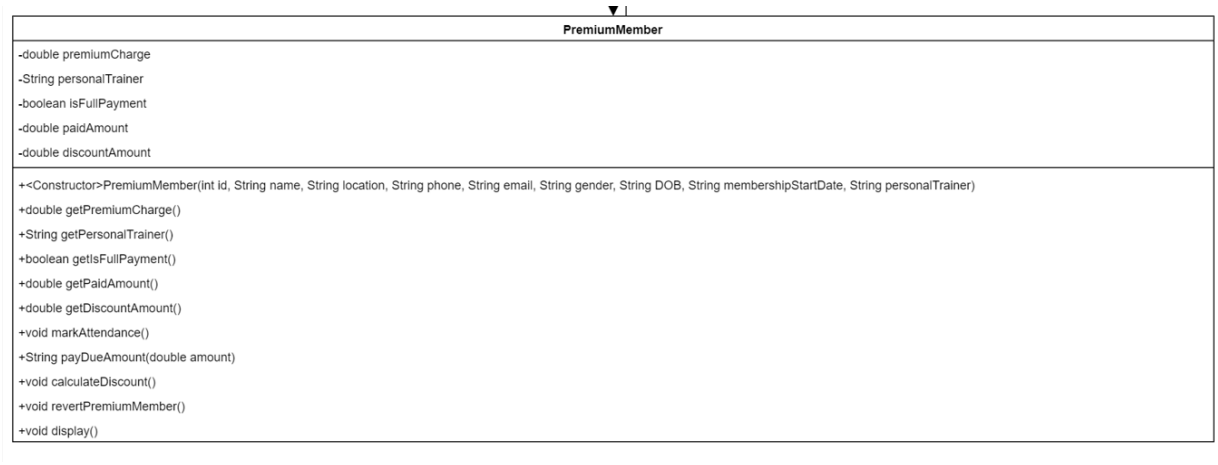


Figure 5: Premium Member class diagram

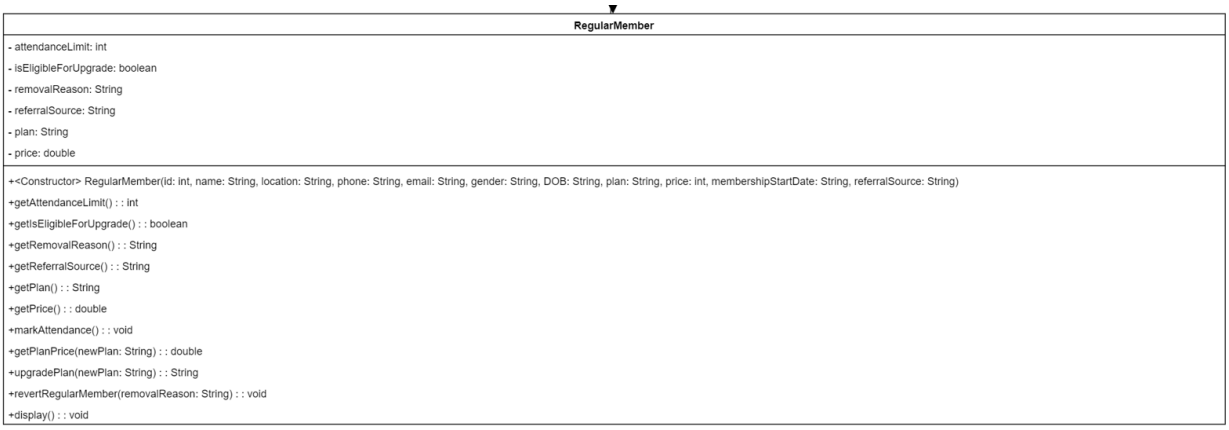


Figure 6: Regular Member class diagram

4 Pseudocode

GymMember

```
CREATE CLASS GymMember AS ABSTRACT

DO

    DECLARE id, attendance AS INTEGER

    DECLARE name, location, phone, email, gender, DOB, membershipStartDate AS STRING

    DECLARE loyaltyPoints AS DOUBLE

    DECLARE activeStatus AS BOOLEAN


<<CONSTRUCTOR>> GymMember(id, name, location, phone, email, gender, DOB, membershipStartDate)

DO

    SET id TO id

    SET name TO name

    SET location TO location

    SET phone TO phone

    SET email TO email

    SET gender TO gender

    SET DOB TO DOB

    SET membershipStartDate TO membershipStartDate

    SET attendance TO 0

    SET loyaltyPoints TO 0

    SET activeStatus TO FALSE


<<METHOD>> activateMembership()

DO

    SET activeStatus TO TRUE
```

<<METHOD>> deactivateMembership()

DO

IF activeStatus IS TRUE

SET activeStatus TO FALSE

<<METHOD>> resetMember()

DO

SET activeStatus TO FALSE

SET attendance TO 0

SET loyaltyPoints TO 0

<<METHOD>> display()

DO

DISPLAY all values of GymMember

RegularMember

CREATE CLASS RegularMember EXTENDS GymMember

DO

DECLARE attendanceLimit AS INTEGER

DECLARE isEligibleForUpgrade AS BOOLEAN

DECLARE removalReason, referralSource, plan AS STRING

DECLARE price AS DOUBLE

<<CONSTRUCTOR>> RegularMember(id, name, location, phone, email, gender, DOB, membershipStartDate, price, plan, referralSource)

DO

CALL super constructor

```
    SET price TO price
    SET plan TO plan
    SET referralSource TO referralSource
    SET removalReason TO ""
    SET isEligibleForUpgrade TO FALSE
    SET attendanceLimit TO 30

<<METHOD>> markAttendance()
DO
    INCREMENT attendance BY 1
    INCREMENT loyaltyPoints BY 5

<<METHOD>> getPlanPrice(plan)
DO
    IF plan IS "basic" THEN RETURN 6500
    ELSE IF plan IS "standard" THEN RETURN 12500
    ELSE IF plan IS "deluxe" THEN RETURN 18500
    ELSE RETURN -1

<<METHOD>> upgradePlan(newPlan)
DO
    IF activeStatus IS FALSE
        RETURN "Cannot upgrade. Membership is inactive."
    IF newPlan IS EQUAL TO current plan
        RETURN "Already on this plan."
    IF attendance < attendanceLimit
        RETURN "Not eligible for upgrade."
    SET plan TO newPlan
```

SET price TO getPlanPrice(newPlan)

SET isEligibleForUpgrade TO TRUE

RETURN "Plan upgraded"

<<METHOD>> revertRegularMember(reason)

DO

CALL resetMember()

SET plan TO "basic"

SET price TO 6500

SET isEligibleForUpgrade TO FALSE

SET removalReason TO reason

<<METHOD>> display()

DO

CALL super.display()

DISPLAY plan, price, referralSource, removalReason

PremiumMember

CREATE CLASS PremiumMember EXTENDS GymMember

DO

DECLARE premiumCharge AS DOUBLE

DECLARE personalTrainer AS STRING

DECLARE isFullPayment AS BOOLEAN

DECLARE paidAmount, discountAmount AS DOUBLE

<<CONSTRUCTOR>> PremiumMember(id, name, location, phone, email, gender, DOB, membershipStartDate, personalTrainer)

DO

```
    CALL super constructor
    SET personalTrainer TO personalTrainer
    SET paidAmount TO 0
    SET discountAmount TO 0
    SET isFullPayment TO FALSE
    SET premiumCharge TO 50000

<<METHOD>> markAttendance()
DO
    INCREMENT attendance BY 1
    INCREMENT loyaltyPoints BY 10

<<METHOD>> payDueAmount(amount)
DO
    IF isFullPayment IS TRUE
        RETURN "Already paid in full"
    IF paidAmount + amount > premiumCharge
        RETURN "Overpayment not allowed"
    INCREMENT paidAmount BY amount
    IF paidAmount IS EQUAL TO premiumCharge
        SET isFullPayment TO TRUE
    RETURN "Payment accepted with remaining amount"

<<METHOD>> calculateDiscount()
DO
    IF isFullPayment IS TRUE
        SET discountAmount TO 10% of premiumCharge
        RETURN "Discount applied"
```

ELSE

RETURN "Discount not applicable"

<<METHOD>> revertPremiumMember()

DO

CALL resetMember()

SET personalTrainer TO ""

SET paidAmount TO 0

SET discountAmount TO 0

SET isFullPayment TO FALSE

<<METHOD>> display()

DO

CALL super.display()

DISPLAY personalTrainer, paidAmount, discountAmount, isFullPayment

GymGui

CREATE CLASS GymGUI

DO

DECLARE frame AS JFrame

DECLARE members AS ArrayList OF GymMember

DECLARE GUI components: labels, text fields, combo boxes, radio buttons, buttons, panels

<<CONSTRUCTOR>> GymGUI()

DO

INITIALIZE frame and set title, size, and close operation

INITIALIZE members AS new ArrayList

SET UP main panel with title "Home Page"

ADD buttons: Regular Member, Premium Member, Display Members, Read from File

WHEN Regular Member button is clicked

CREATE a new frame titled "Add Regular Member"

ADD fields: ID, Name, Location, Phone, Email, Gender, DOB, Plan, Start Date, Referral

WHEN plan is changed

UPDATE plan price

WHEN Add Regular Member is clicked

VALIDATE inputs

CREATE RegularMember object

ADD to members

SHOW confirmation

WHEN Upgrade Plan is clicked

ASK for ID and new plan

IF member is eligible

CALL upgradePlan() and show result

WHEN Mark Attendance is clicked

ASK for ID

IF active

CALL markAttendance()

WHEN Revert or Deactivate is clicked

ASK for ID

REMOVE or deactivate member

WHEN Save to File is clicked

CALL saveToFile()

WHEN Clear Form is clicked

RESET all fields

WHEN Premium Member button is clicked

CREATE a new frame titled "Add Premium Member"

ADD fields: ID, Name, Location, Phone, Email, Gender, DOB, Start Date, Personal Trainer

SET plan TO "Premium" and price TO 50000

WHEN Add Premium Member is clicked

VALIDATE inputs

CREATE PremiumMember object

ADD to members

SHOW confirmation

WHEN Mark Attendance is clicked

ASK for ID

CALL markAttendance()

WHEN Pay Due Amount is clicked

ASK for ID and amount

CALL payDueAmount() and show message

WHEN Calculate Discount is clicked

ASK for ID

IF full payment is done

CALCULATE and APPLY 10% discount

WHEN Revert or Deactivate is clicked

ASK for ID

REMOVE or deactivate member

WHEN Save to File is clicked

CALL saveToFile()

WHEN Clear Form is clicked

RESET all fields

WHEN Display Members button is clicked

```
        IF members list is empty
            SHOW "No members to display"
        ELSE
            LOOP through members
                APPEND details to JTextArea
            DISPLAY in scrollable window

    WHEN Read from File button is clicked
        IF file does not exist
            SHOW warning
        ELSE
            READ content from file
            DISPLAY in JTextArea inside scroll pane

<<METHOD>> saveToFile()
DO
    OPEN MemberDetails.txt using FileWriter
    FOR each member in members
        IF member is RegularMember
            WRITE regular details: ID, name, plan, price, etc.
        ELSE IF member is PremiumMember
            WRITE premium details: trainer, price, paid amount, etc.
    CLOSE file
    SHOW "Saved successfully"
```

5 Method Description

Method Name: activateMembership()
Description: Activates a member by setting their activeStatus to true.
Return Type: void

Method Name: deactivateMembership()
Description: Deactivates the member only if already active. Sets activeStatus to false.
Return Type: void

Method Name: resetMember()
Description: Resets member's status by clearing attendance and loyalty points and deactivating them.
Return Type: void

Method Name: markAttendance()
Description: Abstract method for recording attendance and updating loyalty points, implemented by subclasses.
Return Type: String

Method Name: display()
Description: Displays all the personal and status information of a member.
Return Type: void

Method Name: markAttendance()
Description: Increases attendance by 1 and loyalty points by 5.
Return Type: String

Method Name: getPlanPrice(plan)
Description: Returns the numerical price of a plan based on its name. Returns -1 if the plan is invalid.
Return Type: int

Method Name: upgradePlan(newPlan)
Description: Upgrades a member's plan if they meet all criteria (active and attended enough).
Return Type: String

Method Name: revertRegularMember(reason)
Description: Resets the member, reverts plan and price to basic, and stores the reason.
Return Type: void

Method Name: display()
Description: Displays regular member's extended information like plan, price, and removal reason.
Return Type: void

Method Name: markAttendance()
Description: Increases attendance by 1 and loyalty points by 10.
Return Type: String

Method Name: payDueAmount(amount)
Description: Accepts partial payment and tracks the total paid. Sets full payment status when complete.
Return Type: String

Method Name: calculateDiscount()
Description: Calculates and stores 10% discount only if the full payment is completed.
Return Type: String

Method Name: revertPremiumMember()
Description: Resets all premium-specific fields including trainer name and payment details.
Return Type: void

Method Name: display()
Description: Displays extended premium details including trainer, payment, and discount info.
Return Type: void

Method Name: getId()
Description: Returns the ID of the member.
Return Type: int

Method Name: getName()
Description: Returns the name of the member.
Return Type: String

Method Name: getLocation()
Description: Returns the location of the member.
Return Type: String

Method Name: getPhone()
Description: Returns the phone number of the member.
Return Type: String

Method Name: getEmail()
Description: Returns the email address of the member.
Return Type: String

Method Name: getGender()
Description: Returns the gender of the member.
Return Type: String

Method Name: getDOB()
Description: Returns the date of birth of the member.
Return Type: String

Method Name: getAttendance()
Description: Returns the attendance count of the member.
Return Type: int

Method Name: getLoyaltyPoints()
Description: Returns the loyalty points of the member.
Return Type: double

Method Name: getActiveStatus()
Description: Returns true if the member is currently active.
Return Type: boolean

Method Name: setAttendance(value)
Description: Sets the attendance count to the specified value.
Return Type: void

Method Name: setActiveStatus(status)
Description: Sets the member's active status to true or false.
Return Type: void

Method Name: getPlan()
Description: Returns the current membership plan.
Return Type: String

Method Name: getPrice()
Description: Returns the price of the current membership plan.
Return Type: double

Method Name: getReferralSource()
Description: Returns the referral source information.
Return Type: String

Method Name: getRemovalReason()
Description: Returns the reason why the member was reverted.
Return Type: String

Method Name: getPersonalTrainer()
Description: Returns the name of the personal trainer.
Return Type: String

Method Name: getPaidAmount()
Description: Returns the total amount paid by the premium member.
Return Type: double

Method Name: getIsFullPayment()
Description: Returns true if the member has paid the full premium amount.
Return Type: boolean

Method Name: getDiscountAmount()
Description: Returns the discount amount applied.
Return Type: double

Method Name: setDiscountAmount(amount)
Description: Sets the discount amount to the specified value.
Return Type: void

6 Testing

Test 1

Objective	To compile and run the program using command prompt
Action	Right click on the folder with the file open in terminal compile all the files using javac *.java and then running using java GymGUI
Expected Output	The program should run
Actual Output	The program runs
Result	The test was successful

Table 1: Test 1

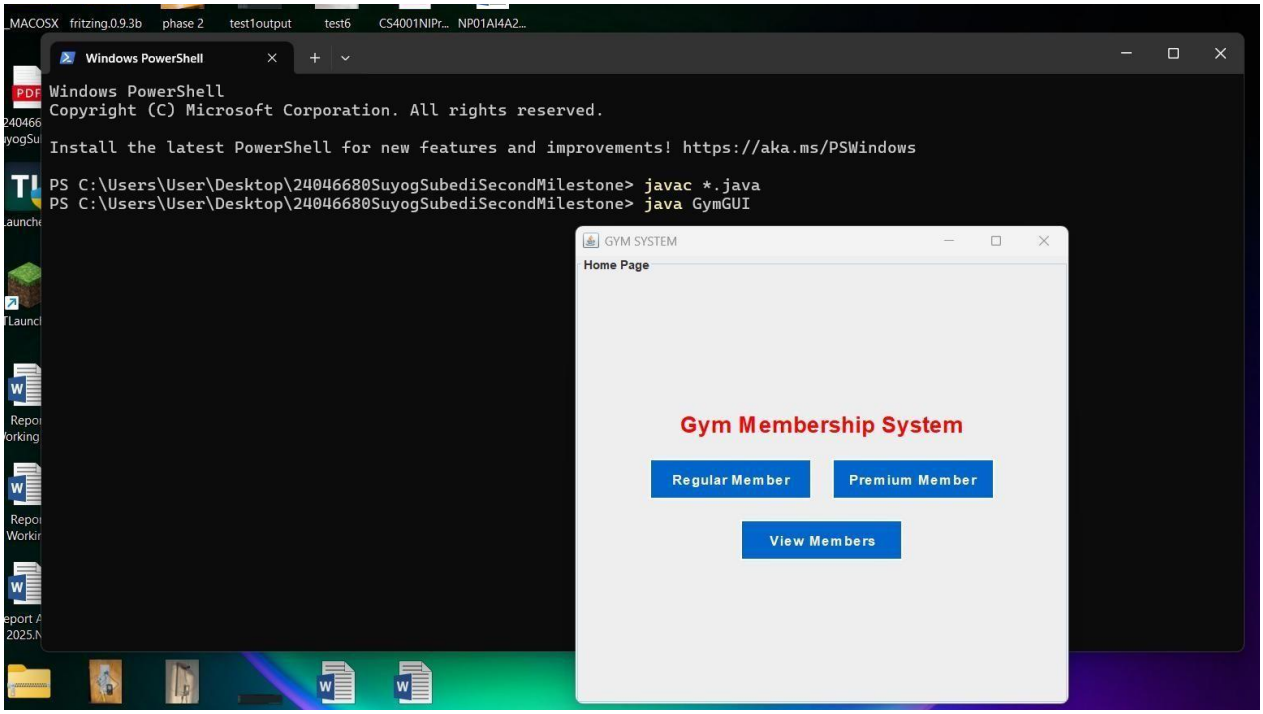


Figure 7: Test 1

Test 2

Objective	To add regular and premium members
Action	Fill all the field in premium and regular members and press add button
Expected Output	The data input should be successful and a dialogue box should appear
Actual Output	The data input is successful and dialogue box appears
Result	The test was successful

Table 2: Test 2

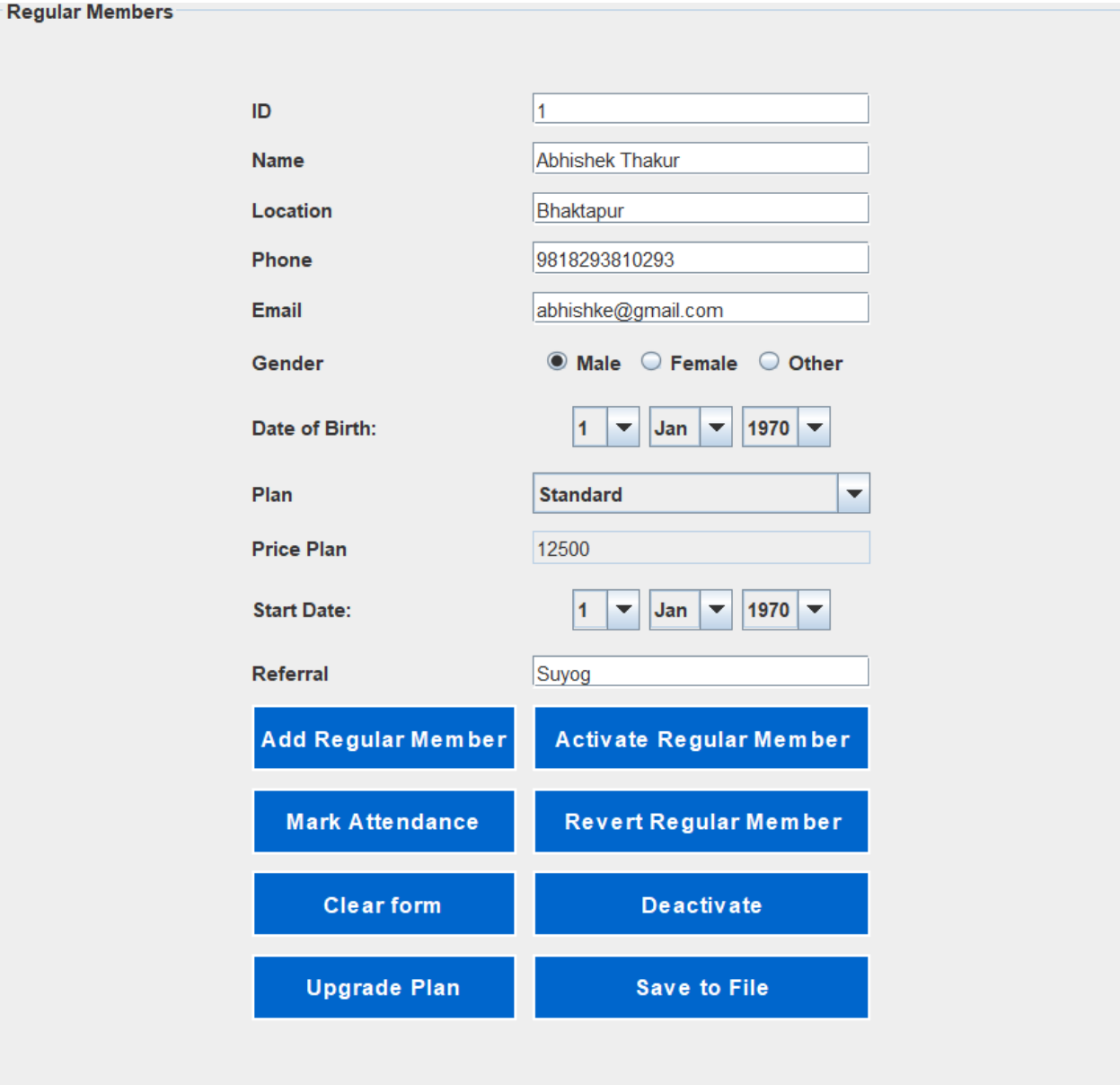


Figure 8: Test 2.1

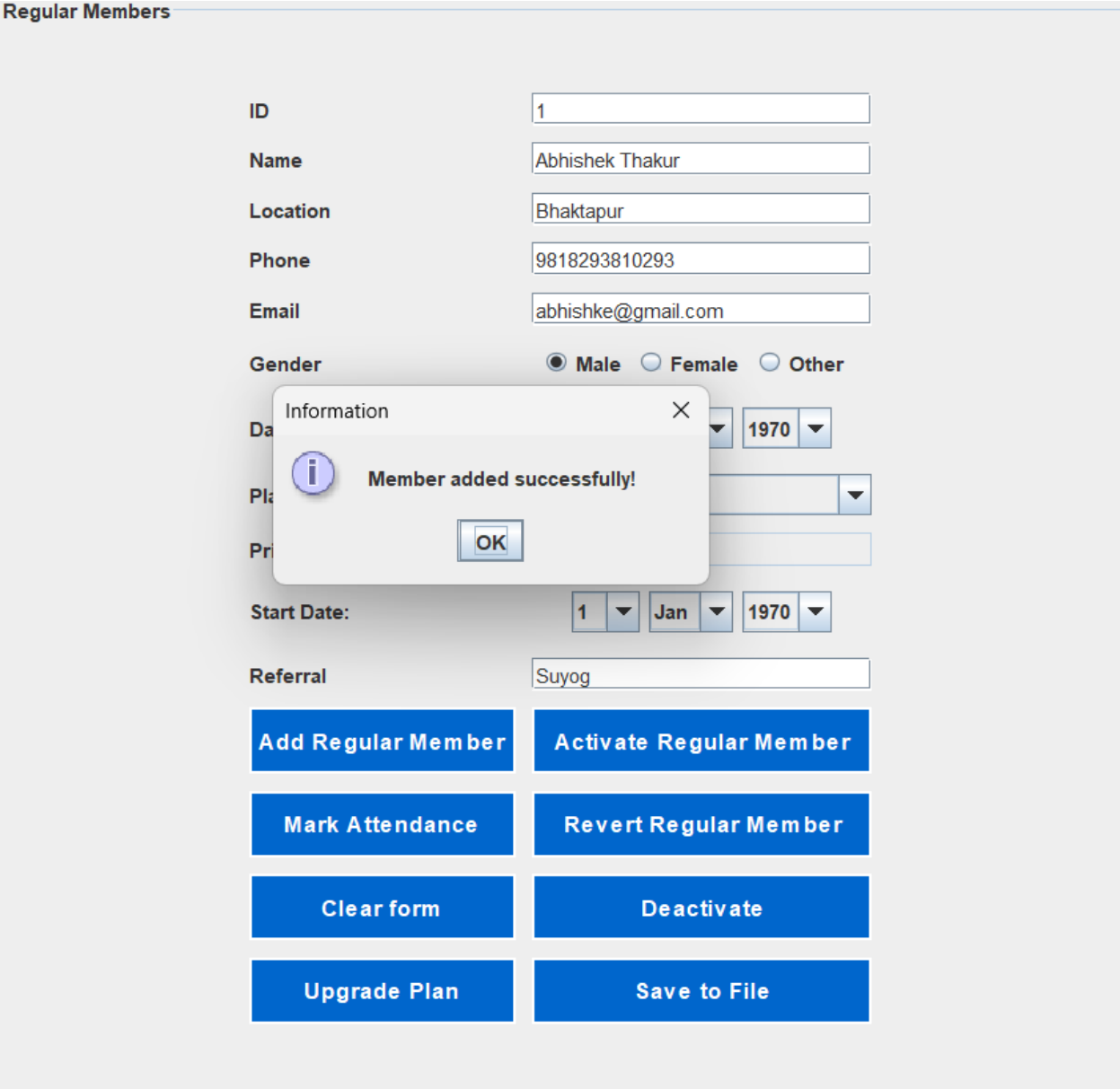


Figure 9:Test 2.2

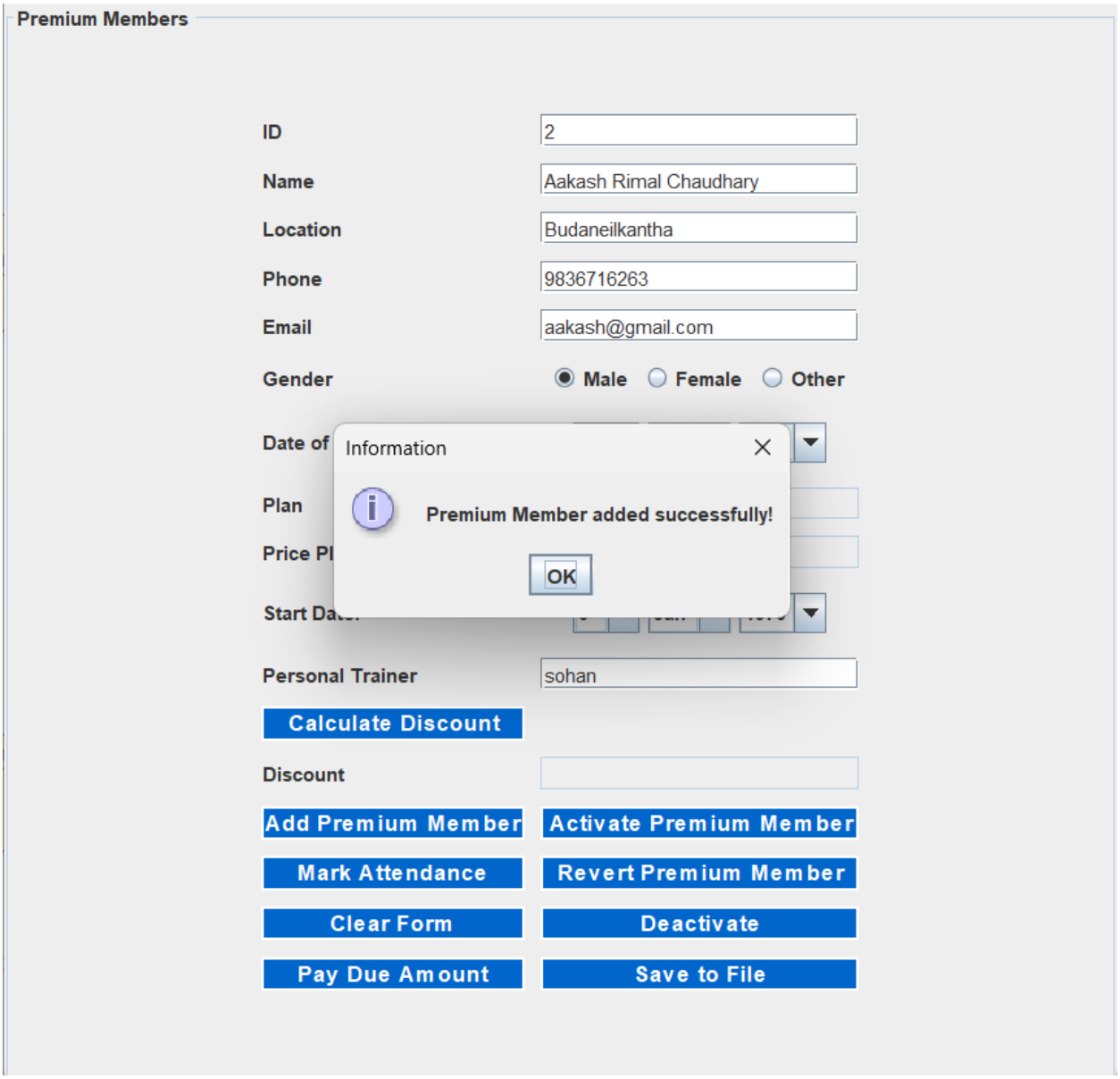


Figure 10: Test 2.3

Figure 11: Test 2.4

Test 3.1

Objective	To check if mark attendance works
Action	Click the mark attendance button and input the Id we want to mark as present
Expected Output	A dialogue box should appear where we can input id and when we press ok it should display appropriate message along with attendance count
Actual Output	A dialogue box appears where we input 12 and press ok it displays an appropriate message with attendance count
Result	The test was successful

Table 3: Test 3.1

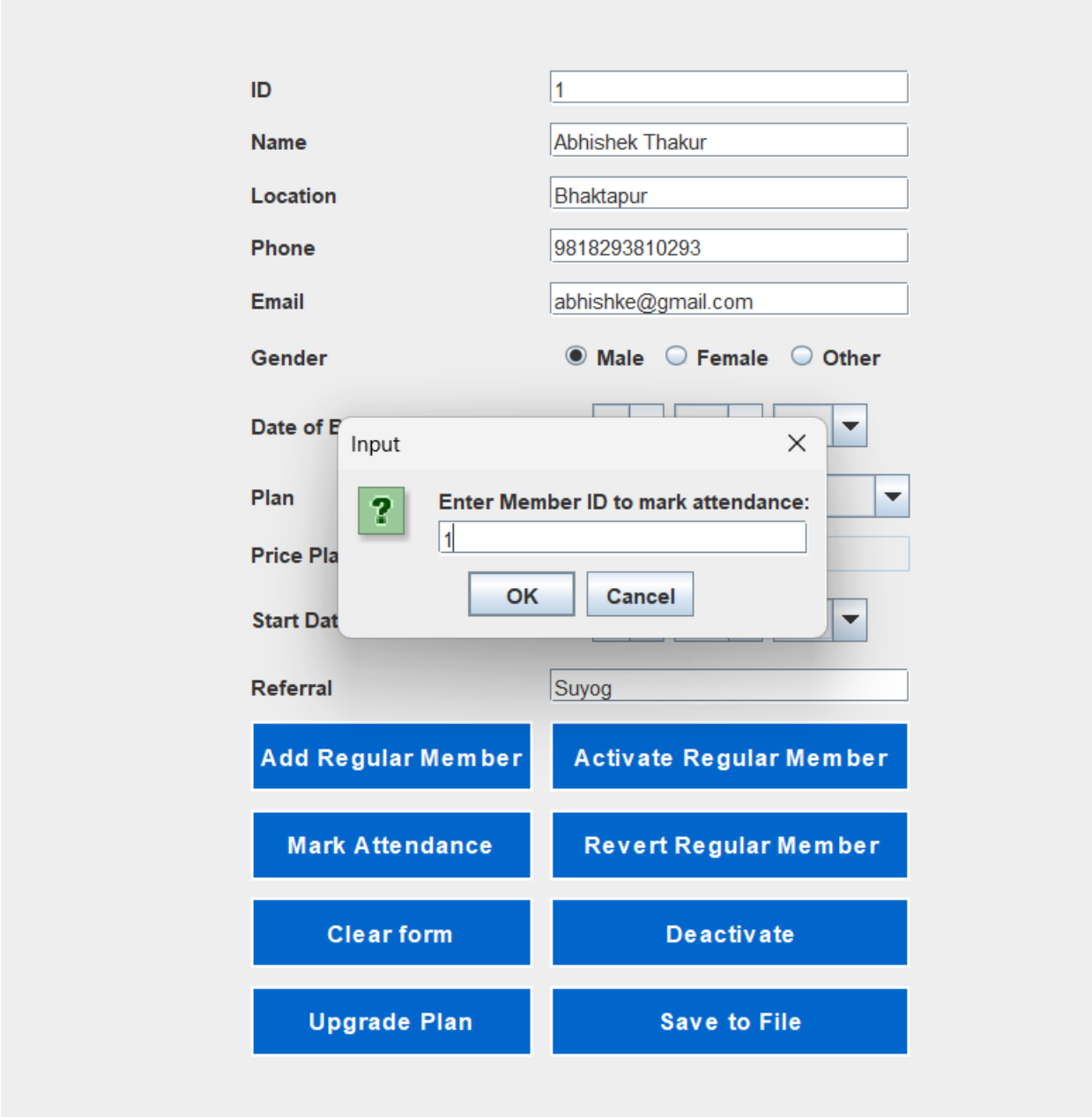


Figure 12:Test 3.1

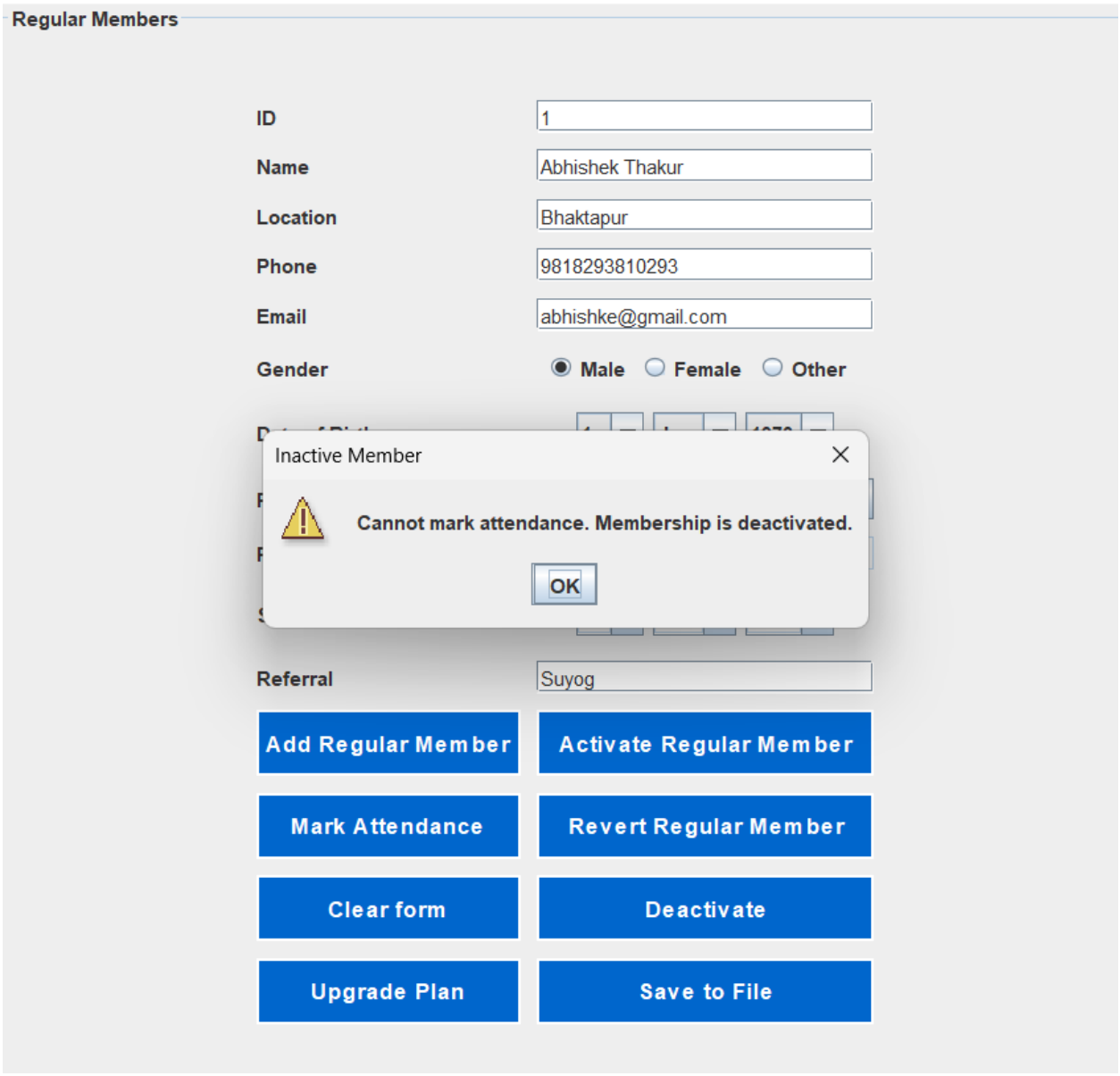


Figure 13:Test 3.2

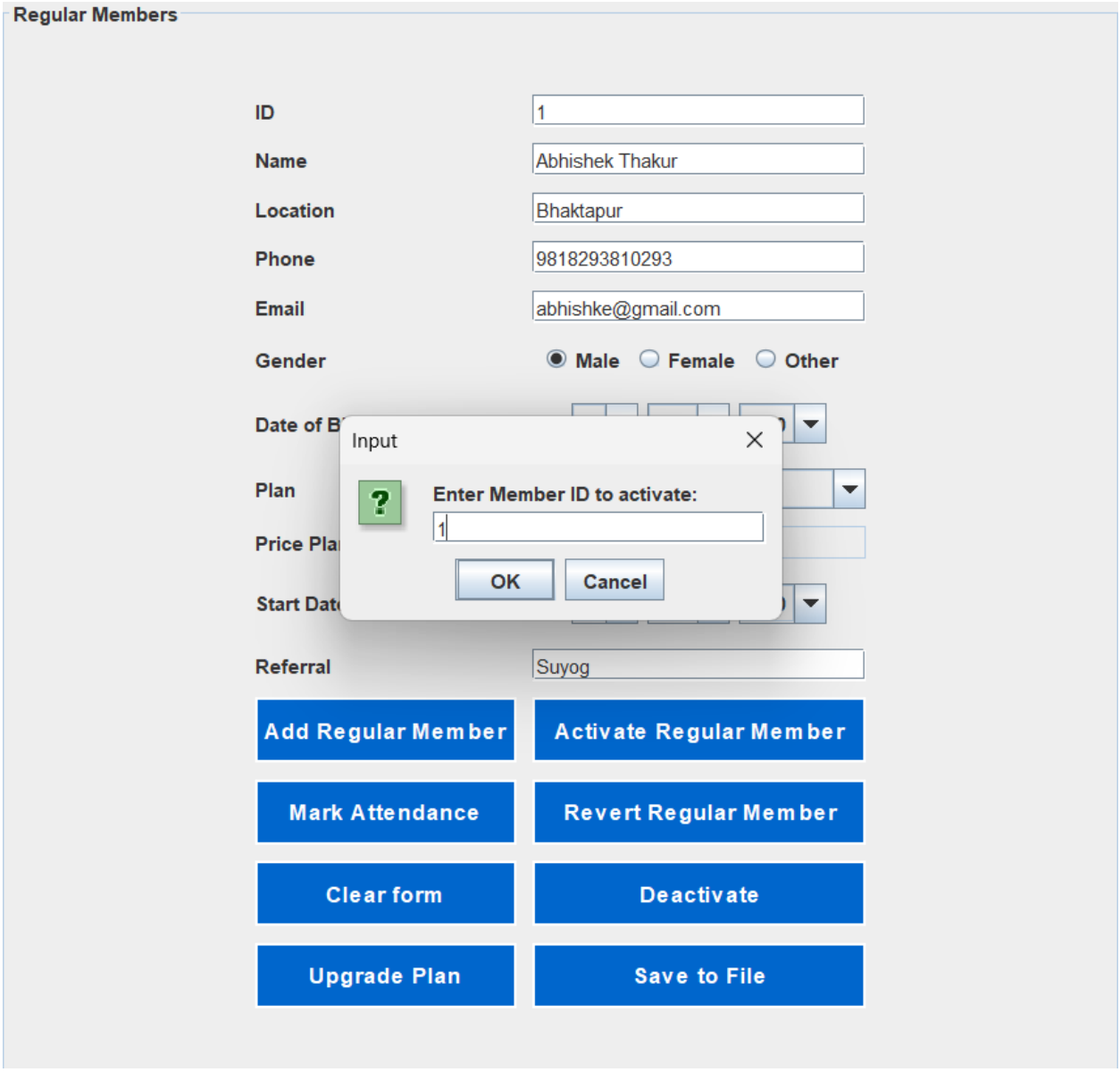


Figure 14:Test 3.3

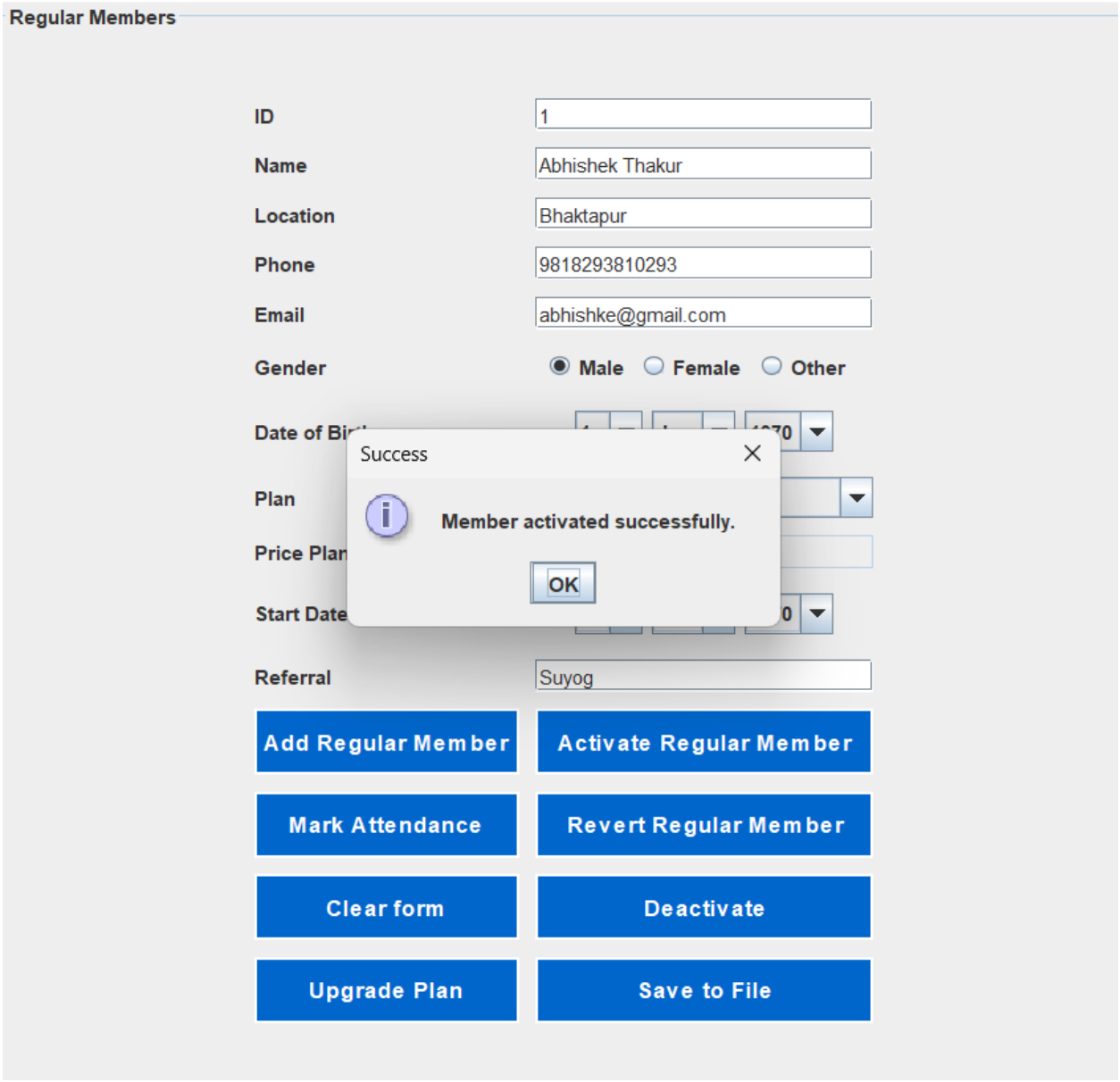


Figure 15: Test 3.4

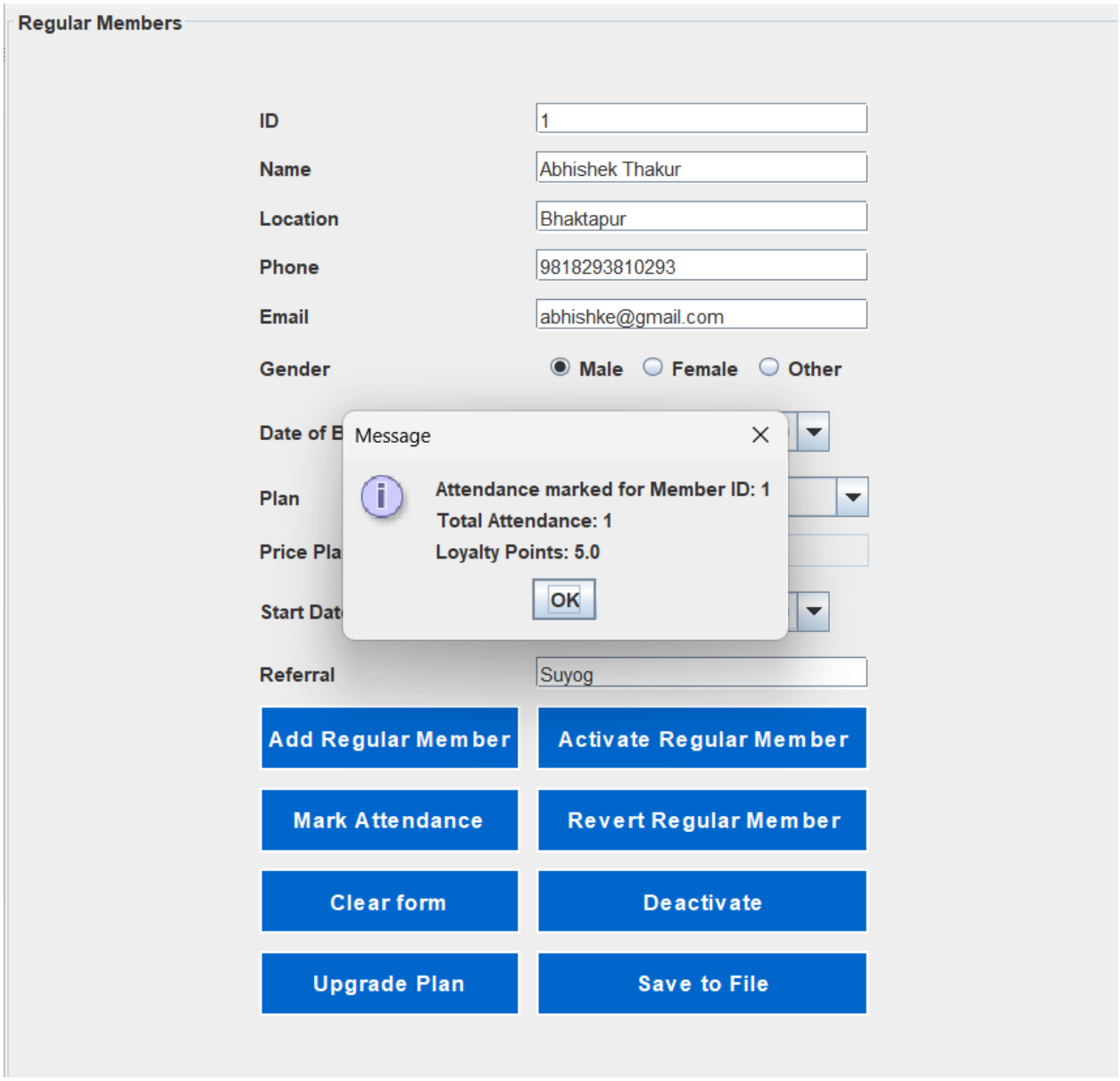


Figure 16:Test 3.5

Test 3.2

Objective	To check if upgrade button works
Action	Click the upgrade button and input the Id we want to upgrade as well as selecting the plan we want to upgrade
Expected Output	A dialogue box should appear where we can input id and select plan when we press ok it should display appropriate message
Actual Output	A dialogue box appears where we input 1 and press ok it displays an appropriate message
Result	The test was successful

Table 4: Test 3.2

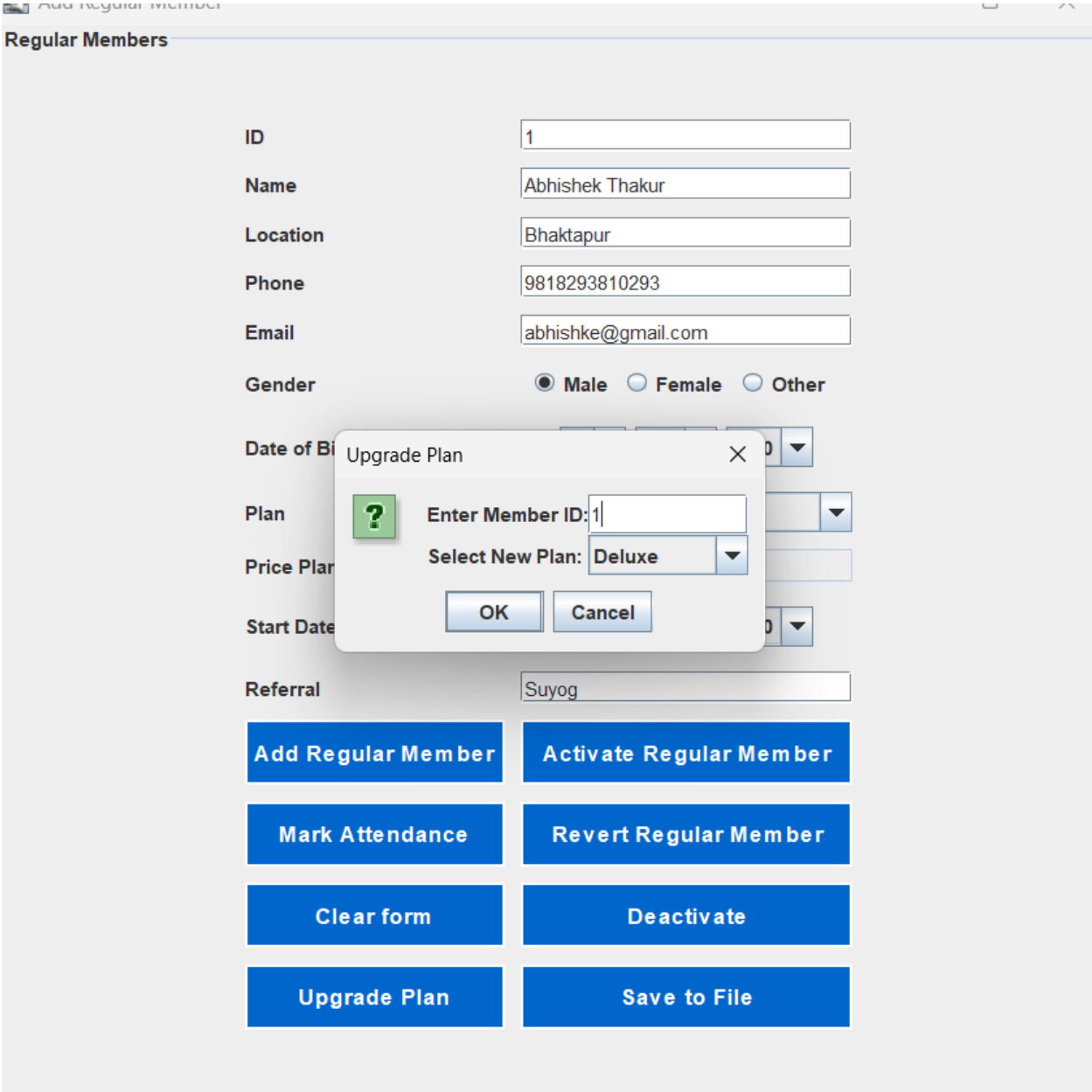


Figure 17:Test 3.6

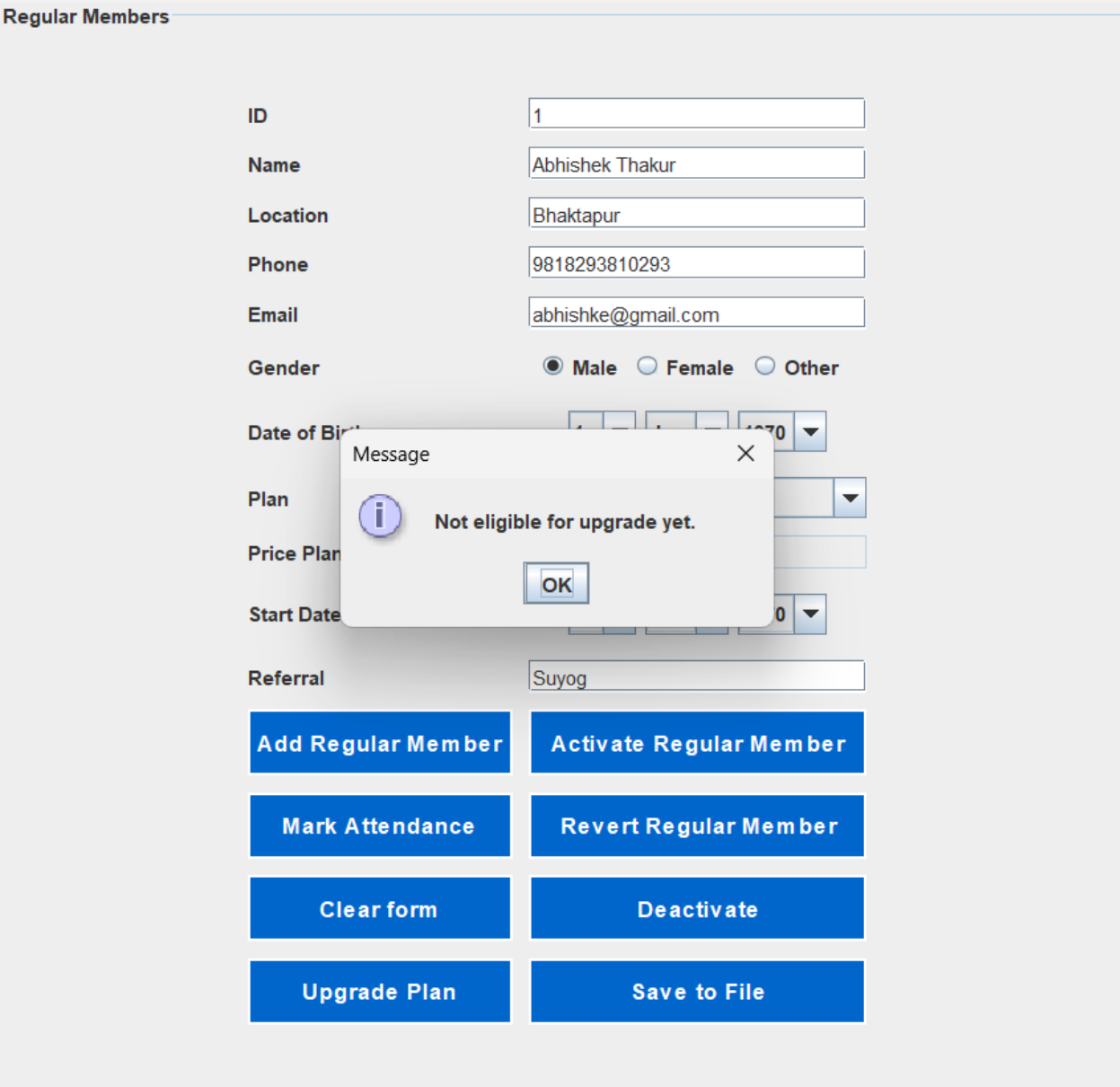


Figure 18:Test 3.7

4.1

Objective	To check if pay due, calculate discount works and revert member
Action	Click the pay due amount and then the calculate discount then remove the member
Expected Output	A dialogue box should appear where we can input id and the amount when we press ok it should display appropriate message. The if the amount is paid in full then discount is applied. Then when we press the revert member it asks for removal reason and then removes the member
Actual Output	A dialogue box appears where we input 2 and input the amount press ok it displays an appropriate message and when we click the discount it applies and also the member is reverted
Result	The test was successful

Table 5: Test 4

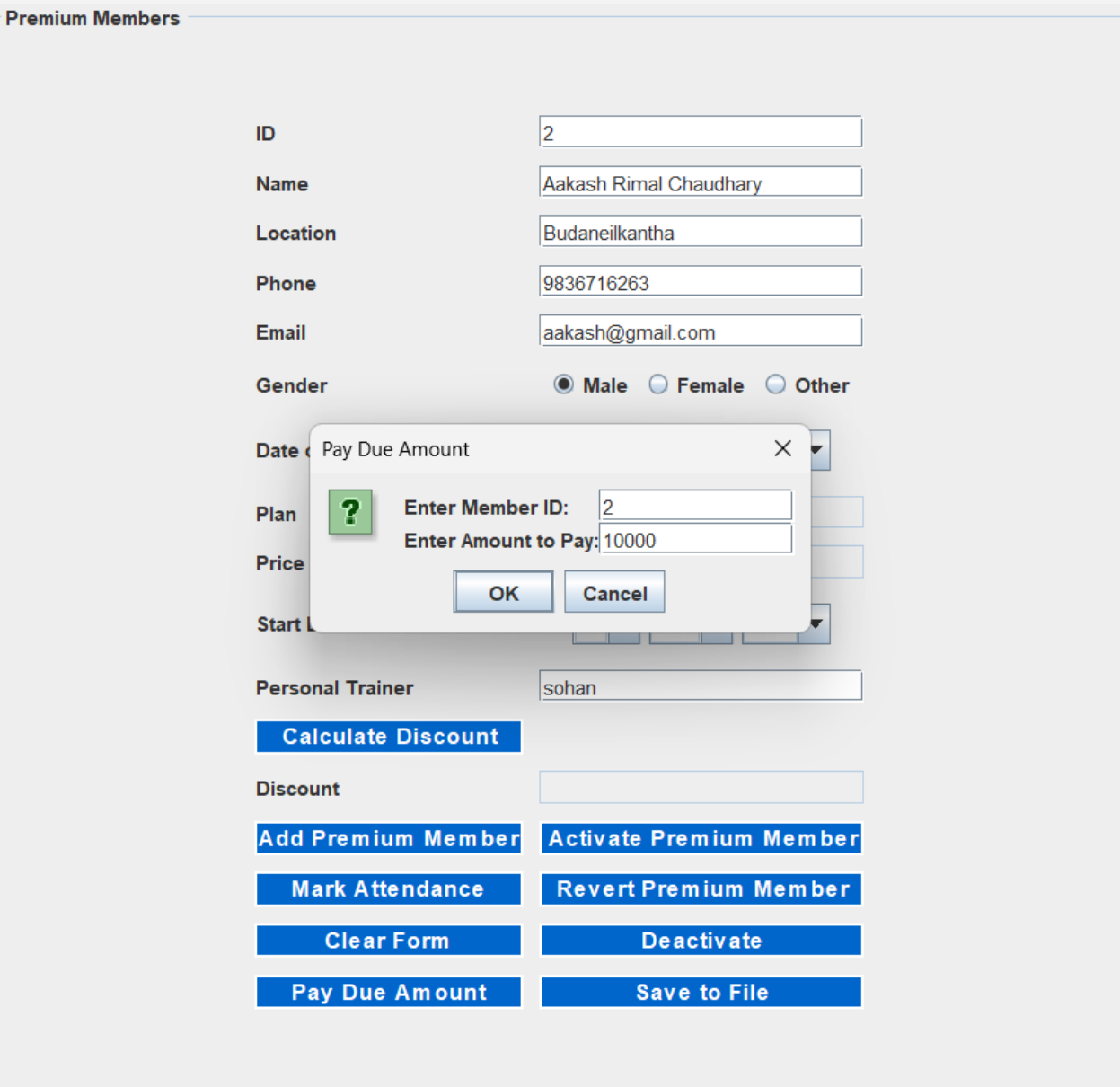


Figure 19:Test 4.1

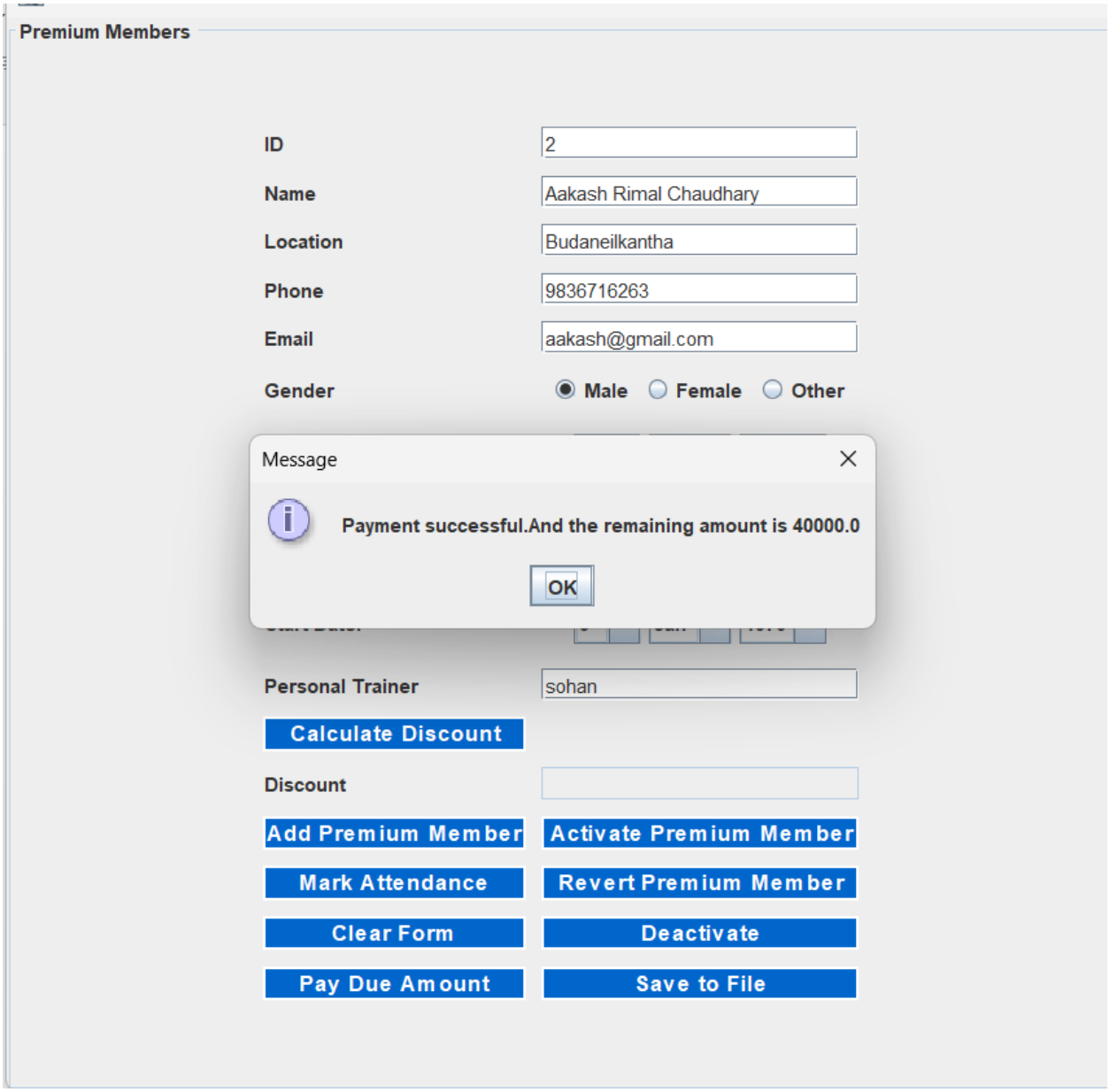


Figure 20: Test 4.2

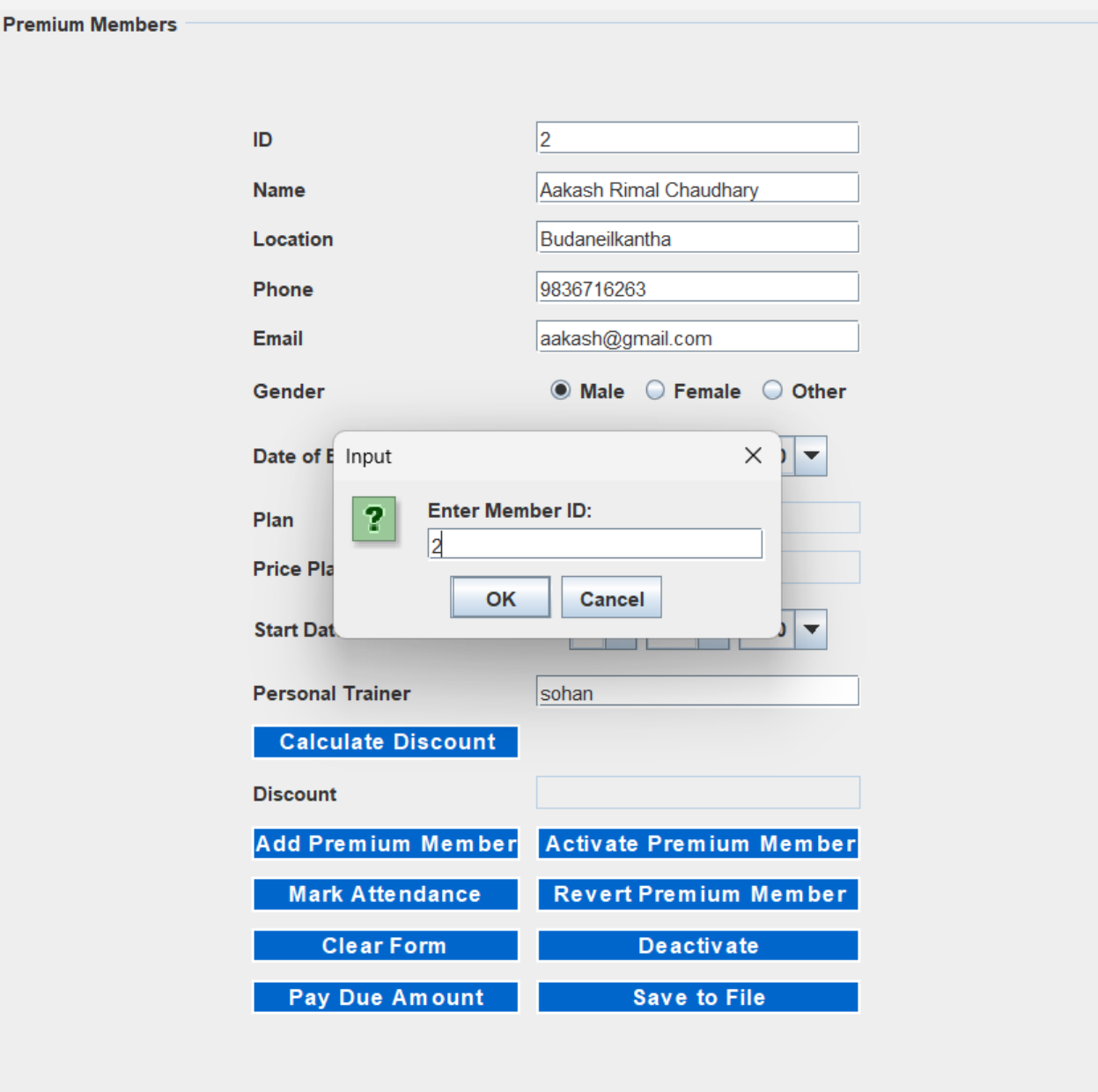


Figure 21:Test 4.3

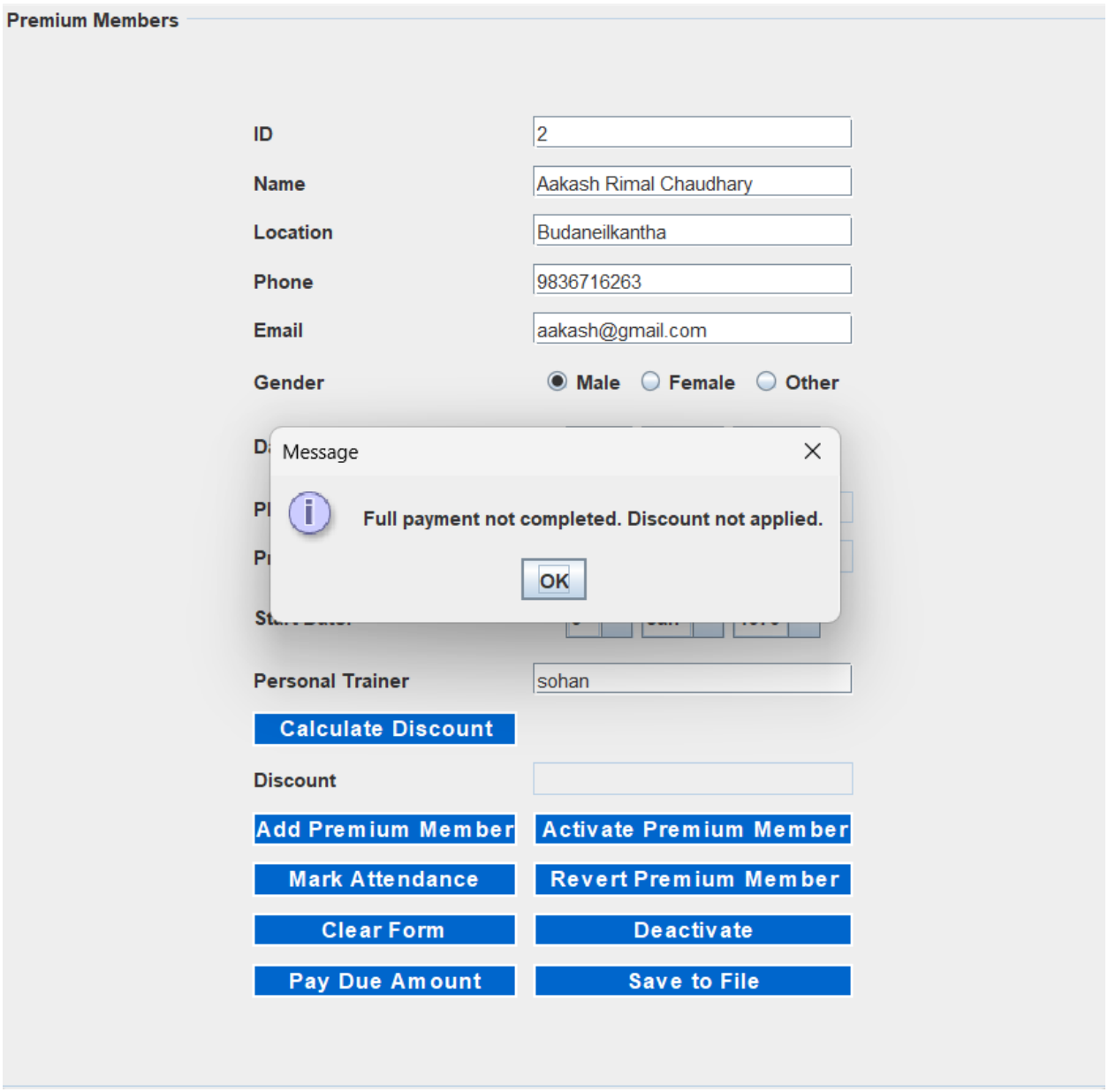


Figure 22:Test 4.4

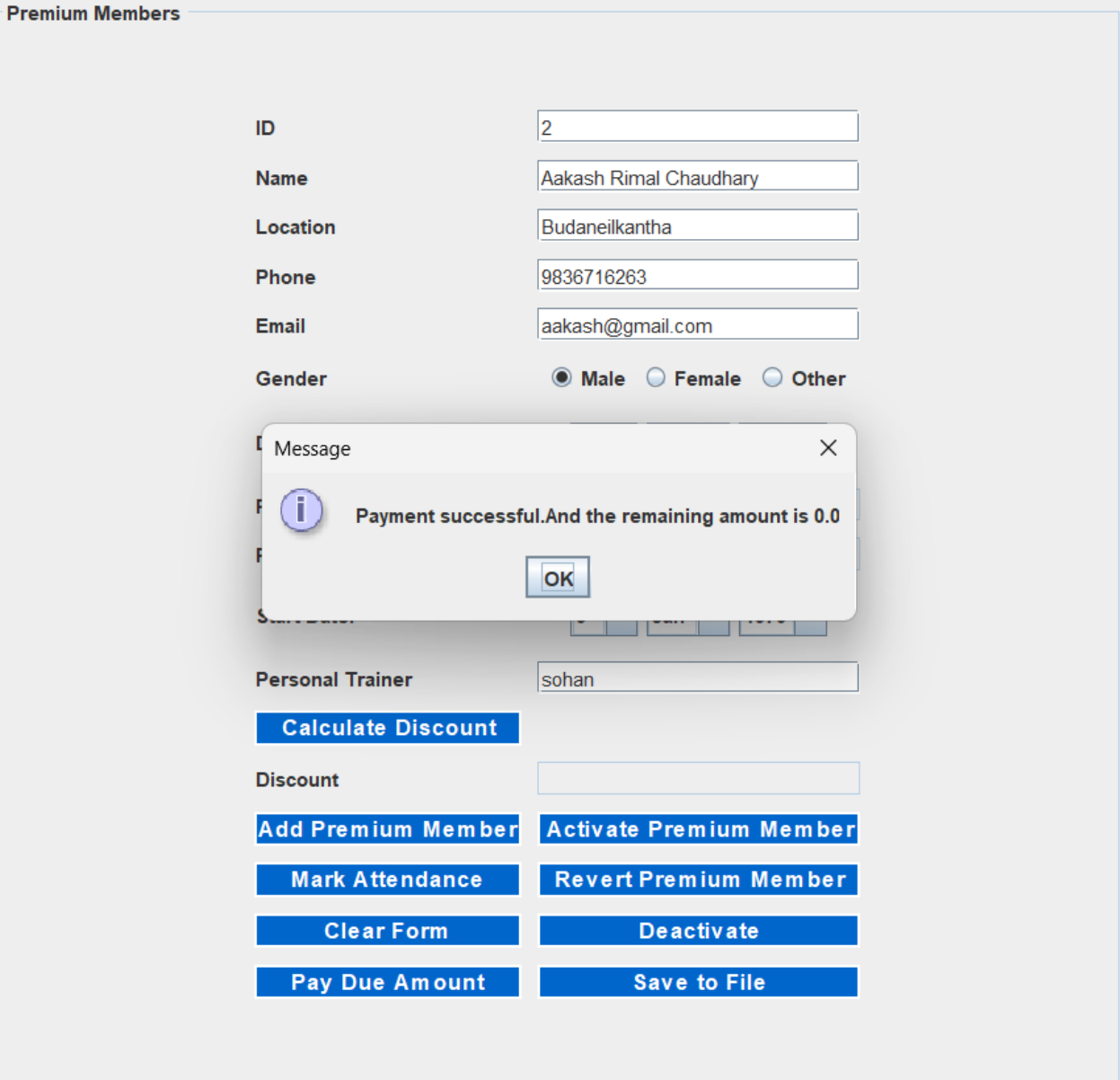


Figure 23:Test 4.5

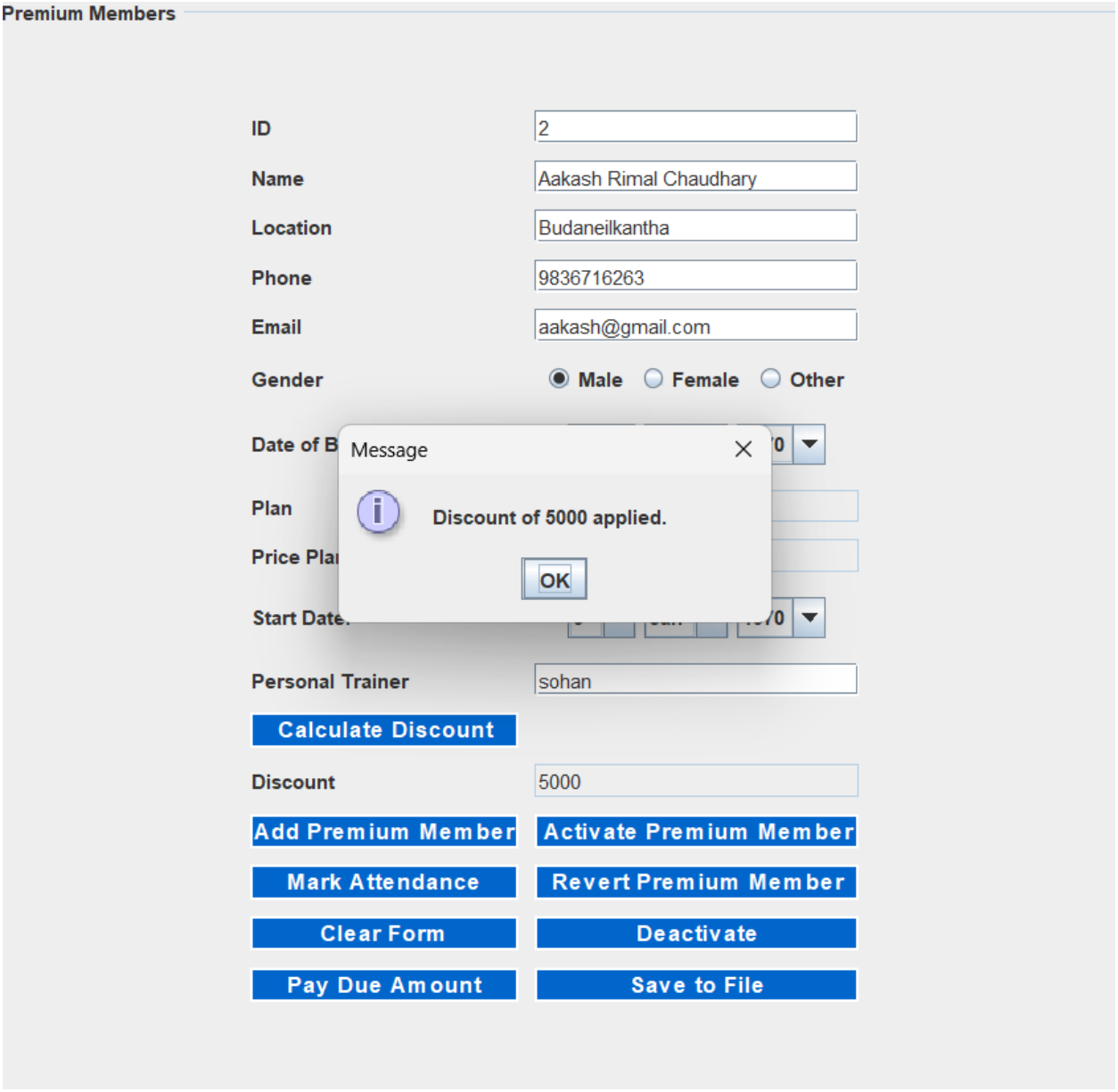


Figure 24:Test 4.6

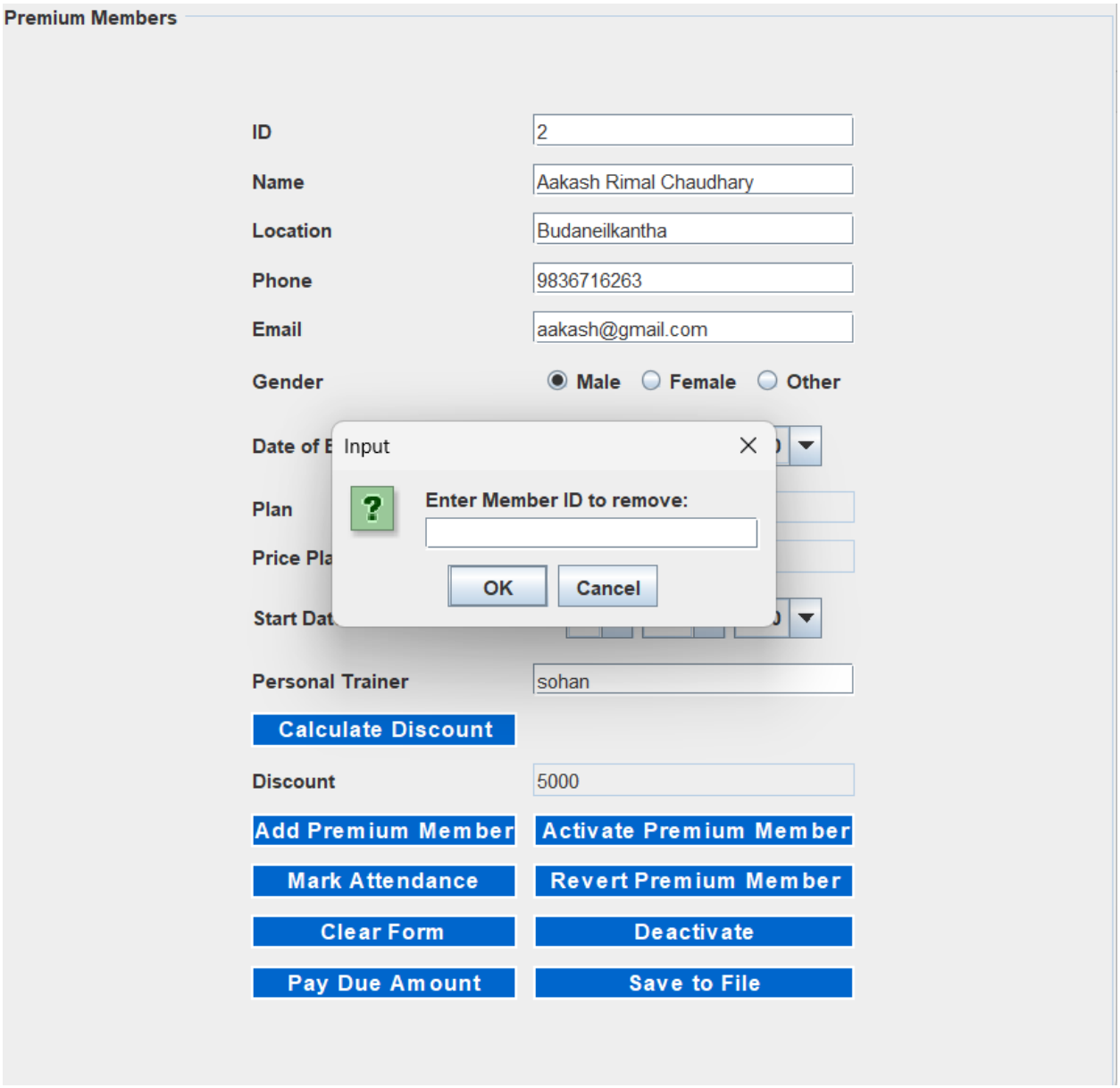


Figure 25:Test 4.7

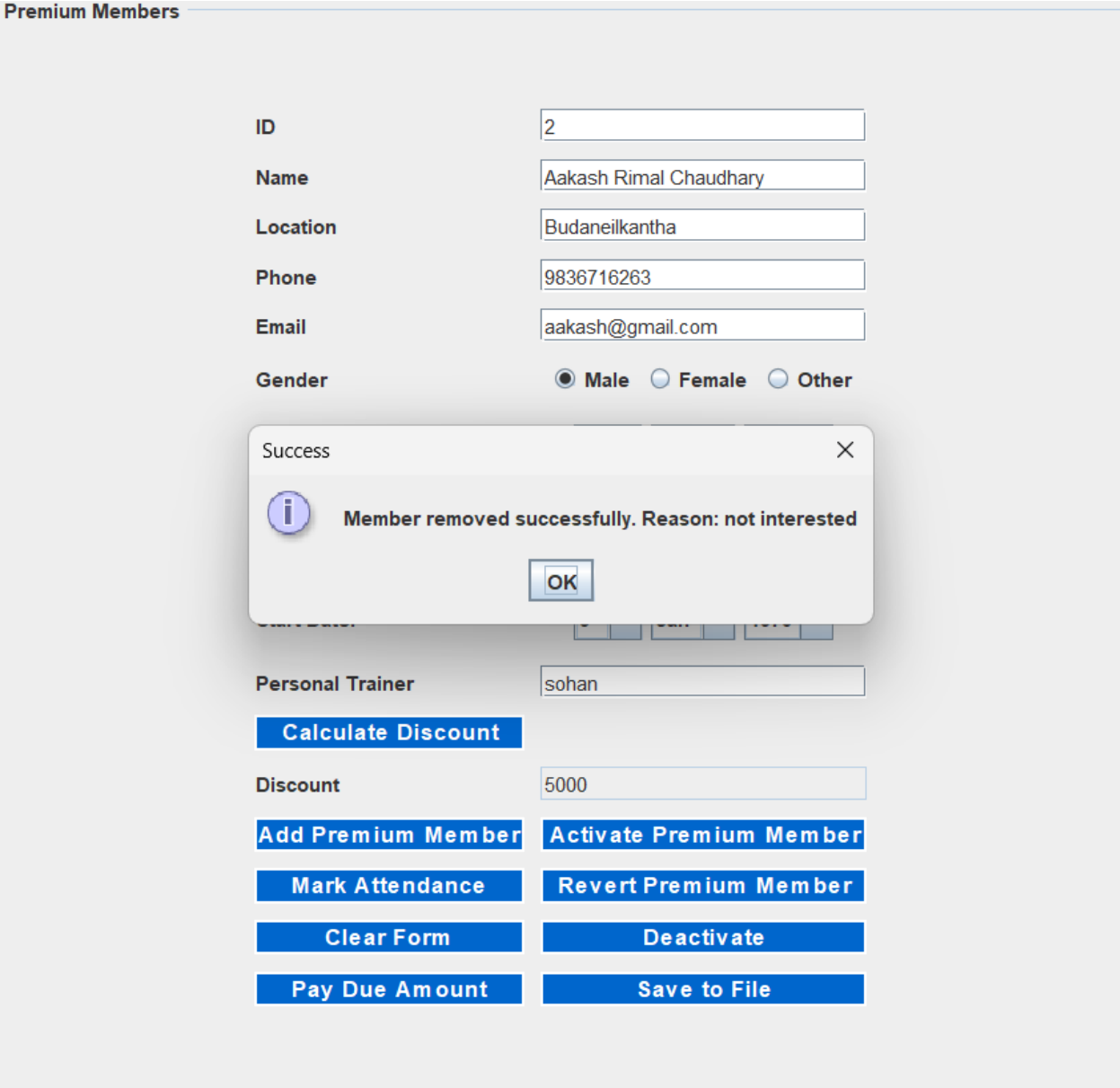


Figure 26:Test 4.8

Test 5

Objective	To check if save to file and read from file works
Action	Click the save to file first after entering details of the member then clicking on the read from file
Expected Output	The information should be stored in file and can read from the file
Actual Output	The information is stored in file and also can read from the file
Result	The test was successful

Table 6: Test 5

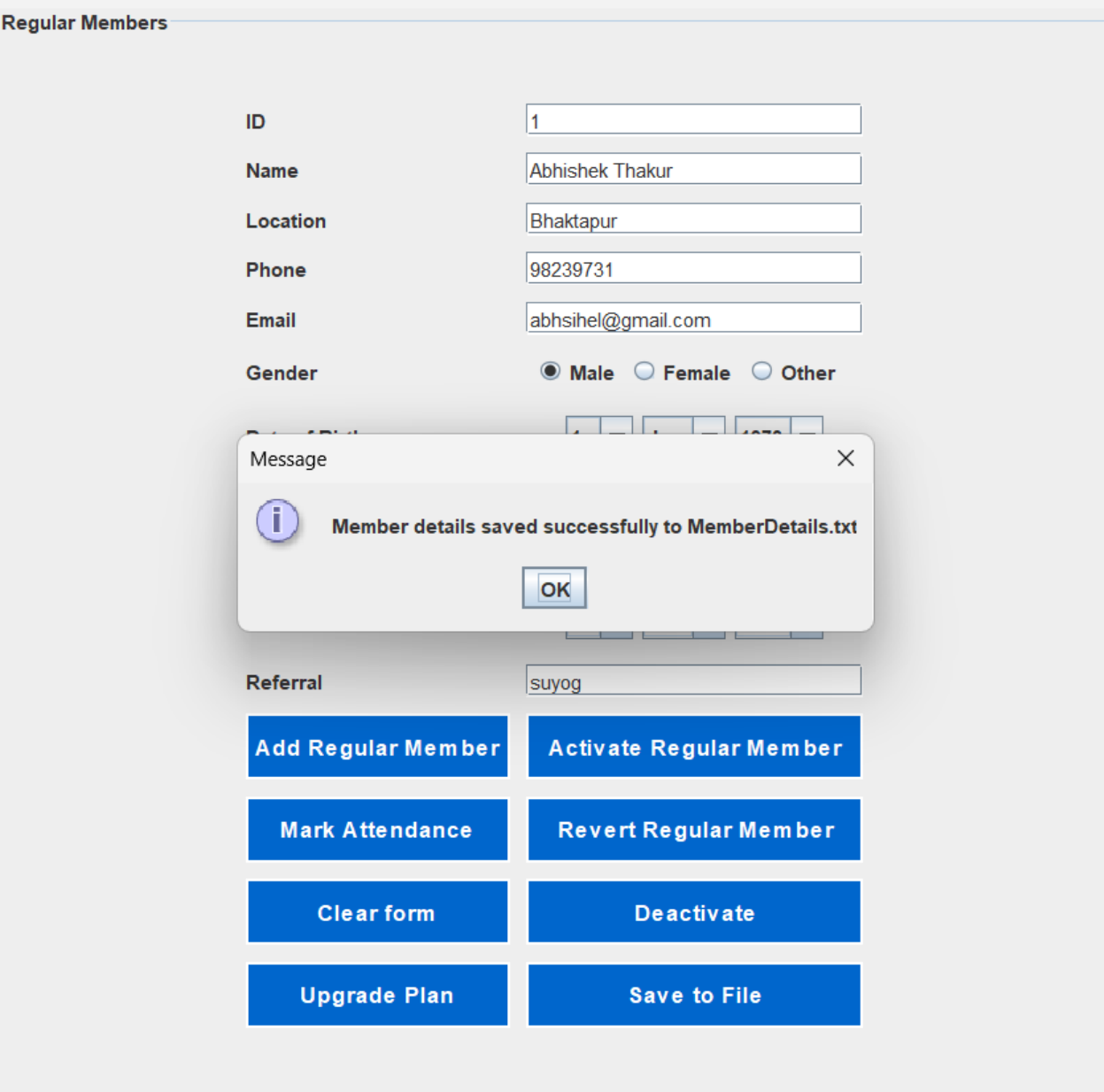


Figure 27:Test 5.1

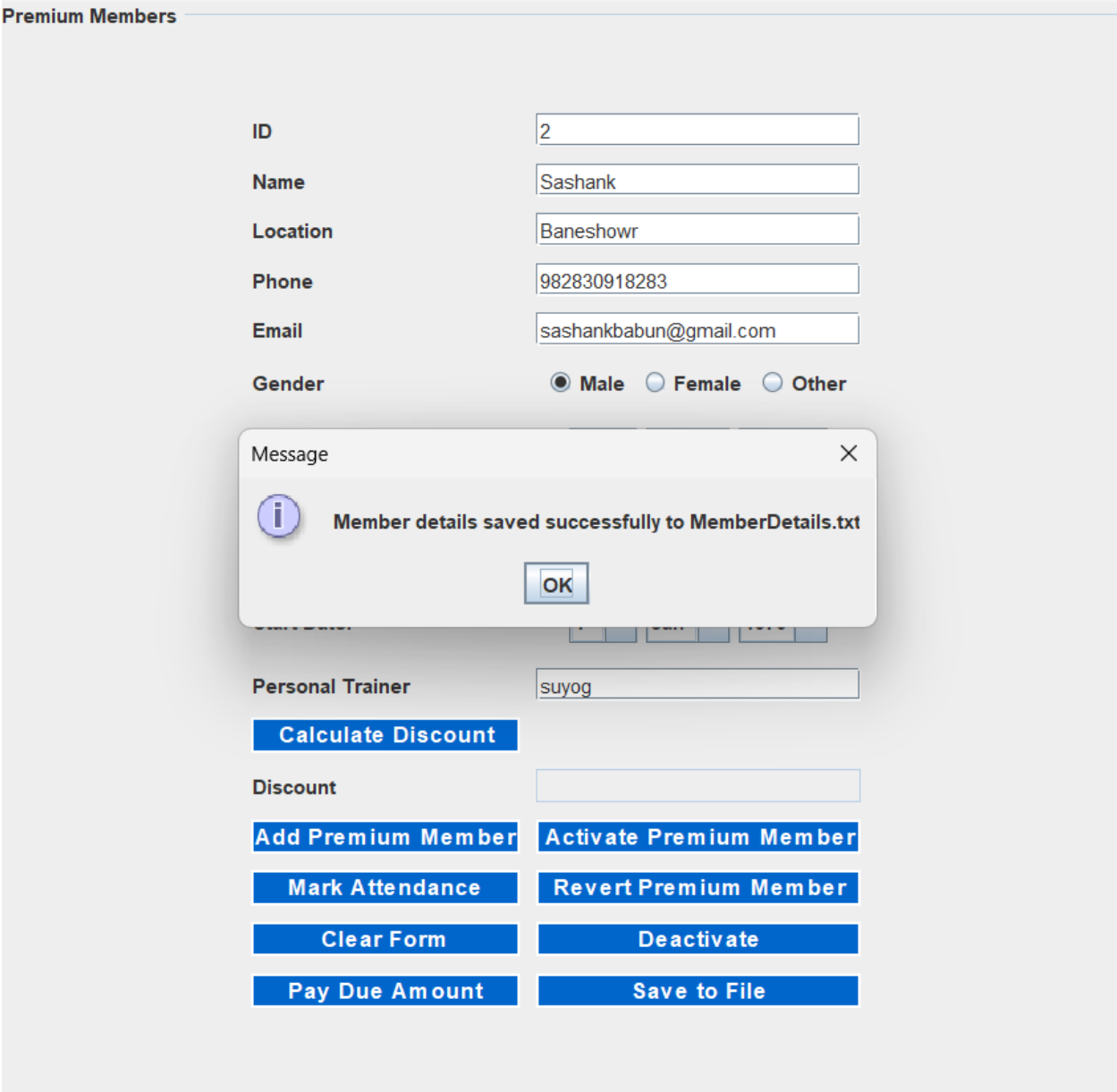


Figure 28:Test 5.2

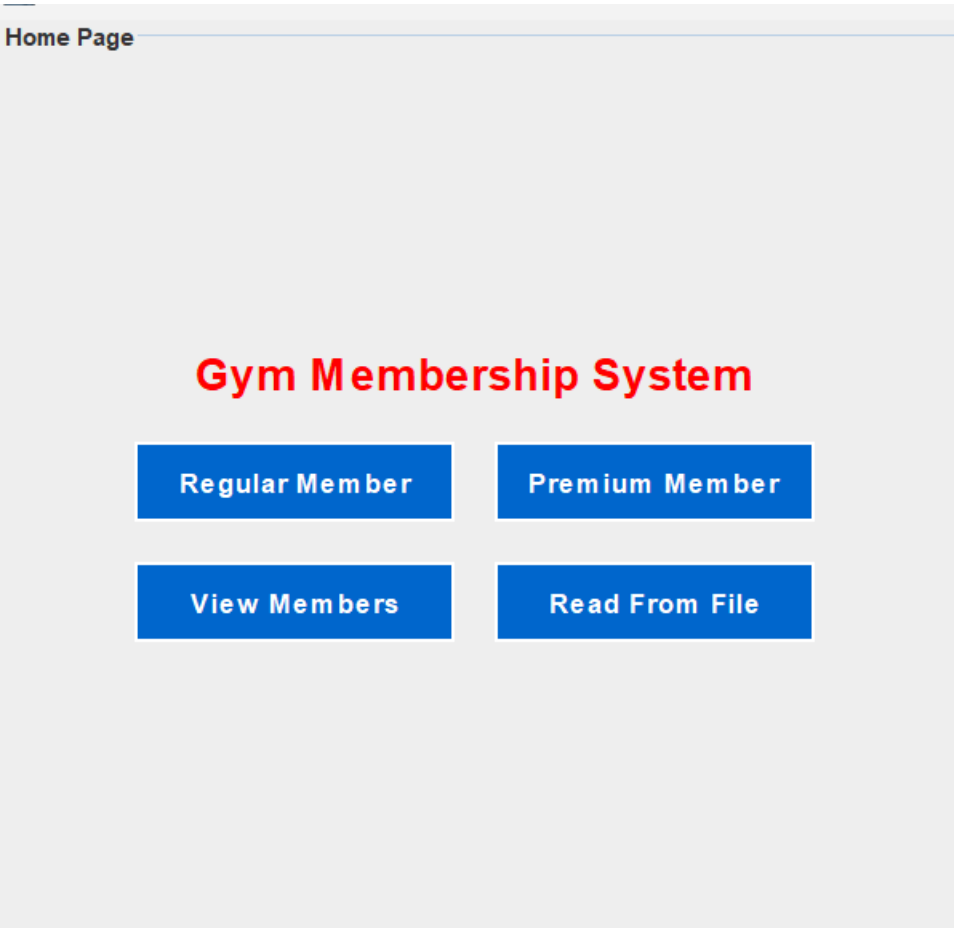


Figure 29:Test 5.3

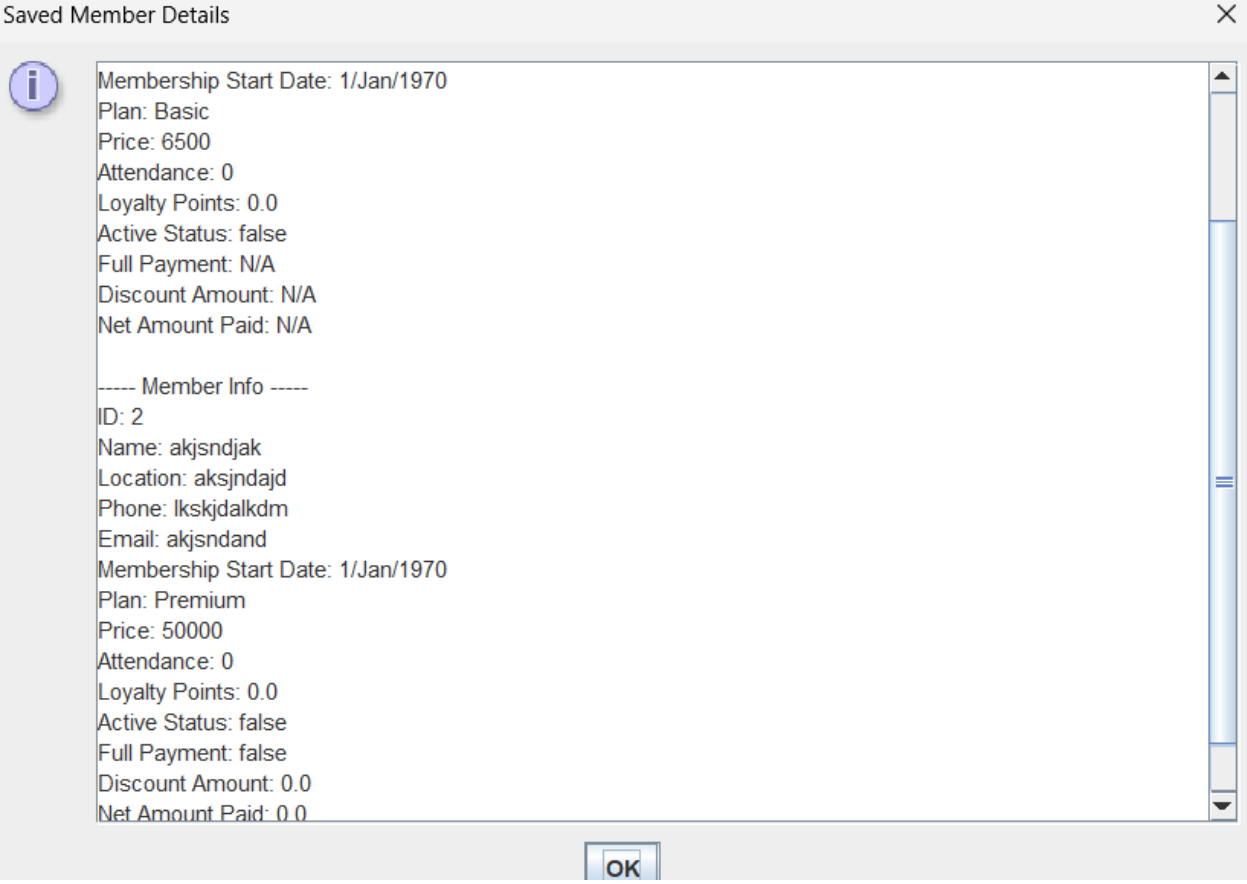


Figure 30:Test 5.3

7 Error Detection and Correction

Error Type: Logical Error

Description: Premium members were not being saved to the file.
Cause: The saveToFile() method did not include a block to handle PremiumMember using instanceof.
Correction: Added an else if (member instanceof PremiumMember) block to correctly write all premium member details.

```
for (GymMember member : members) {
    writer.write("----- Member Info -----\n");
    writer.write("ID: " + member.getId() + "\n");
    writer.write("Name: " + member.getName() + "\n");
    writer.write("Location: " + member.getLocation() + "\n");
    writer.write("Phone: " + member.getPhone() + "\n");
    writer.write("Email: " + member.getEmail() + "\n");
    writer.write("Membership Start Date: " + member.membershipStartDate + "\n");

    if (member instanceof RegularMember) {
        RegularMember rm = (RegularMember) member;
        writer.write("Plan: " + rm.getPlan() + "\n");
        writer.write("Price: " + (int) rm.getPrice() + "\n");
        writer.write("Attendance: " + rm.getAttendance() + "\n");
        writer.write("Loyalty Points: " + rm.getLoyaltyPoints() + "\n");
        writer.write("Active Status: " + rm.getActiveStatus() + "\n");
        writer.write("Full Payment: N/A\n");
        writer.write("Discount Amount: N/A\n");
        writer.write("Net Amount Paid: N/A\n");


    writer.write("\n");
}
```

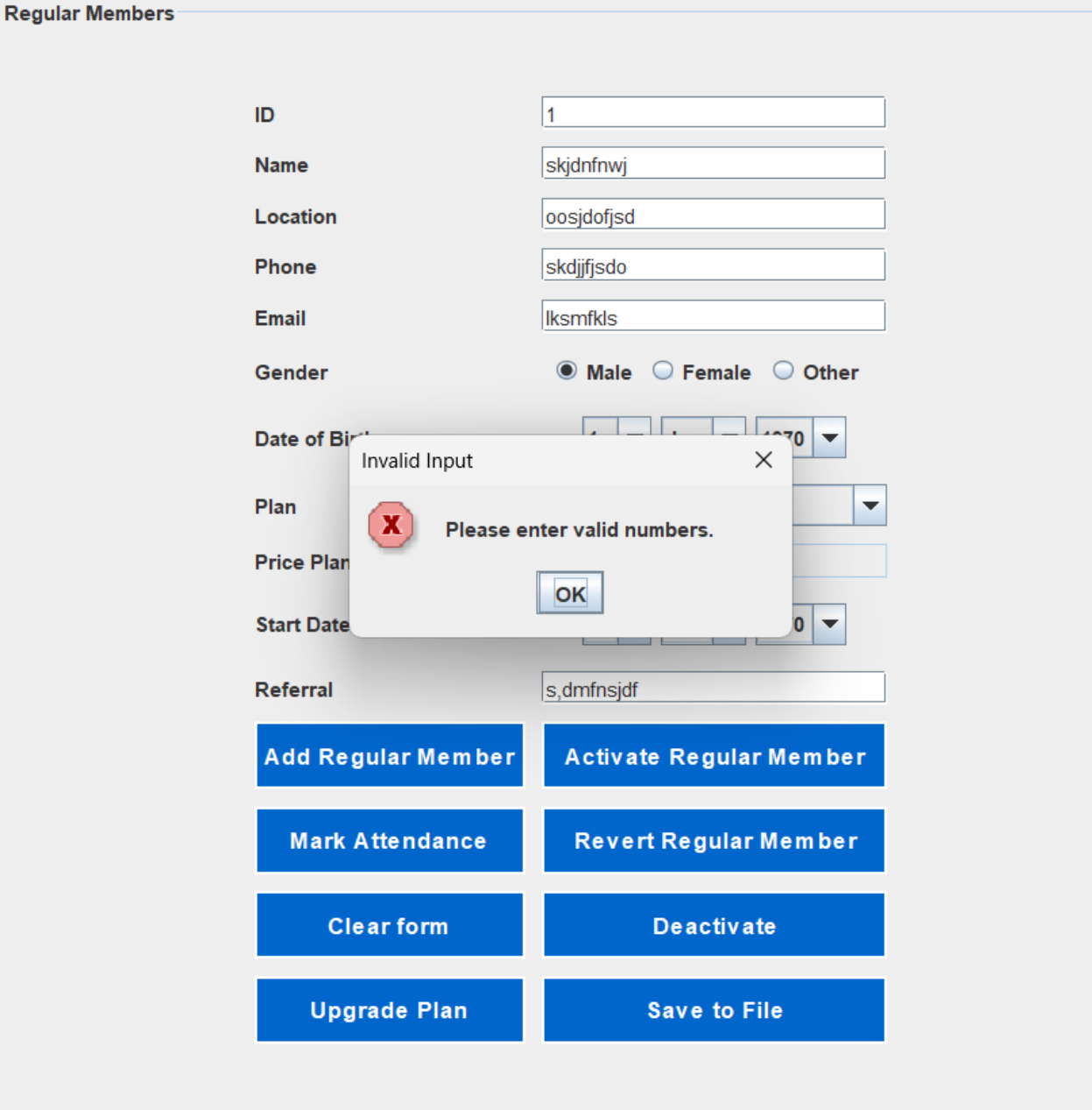
Figure 31:Errro 1

```
for (GymMember member : members) {
    writer.write("----- Member Info -----\n");
    writer.write("ID: " + member.getId() + "\n");
    writer.write("Name: " + member.getName() + "\n");
    writer.write("Location: " + member.getLocation() + "\n");
    writer.write("Phone: " + member.getPhone() + "\n");
    writer.write("Email: " + member.getEmail() + "\n");
    writer.write("Membership Start Date: " + member.membershipStartDate + "\n");

    if (member instanceof RegularMember) {
        RegularMember rm = (RegularMember) member;
        writer.write("Plan: " + rm.getPlan() + "\n");
        writer.write("Price: " + (int) rm.getPrice() + "\n");
        writer.write("Attendance: " + rm.getAttendance() + "\n");
        writer.write("Loyalty Points: " + rm.getLoyaltyPoints() + "\n");
        writer.write("Active Status: " + rm.getActiveStatus() + "\n");
        writer.write("Full Payment: N/A\n");
        writer.write("Discount Amount: N/A\n");
        writer.write("Net Amount Paid: N/A\n");

    } else if (member instanceof PremiumMember) {
        PremiumMember pm = (PremiumMember) member;
        writer.write("Plan: Premium\n");
        writer.write("Price: 50000\n");
        writer.write("Attendance: " + pm.getAttendance() + "\n");
        writer.write("Loyalty Points: " + pm.getLoyaltyPoints() + "\n");
        writer.write("Active Status: " + pm.getActiveStatus() + "\n");
        writer.write("Full Payment: " + pm.getIsFullPayment() + "\n");
        writer.write("Discount Amount: " + pm.getDiscountAmount() + "\n");
        writer.write("Net Amount Paid: " + pm.getPaidAmount() + "\n");
    }
```

Figure 32:Error 1 fix

Error Type: Runtime Error (NumberFormatException)

Description: Application crashed when trying to convert a non-numeric string to an integer.
Cause: Integer.parseInt() was used on plan names like "Standard" and "Deluxe" which are non-numeric.
Correction: Replaced this with getPlanPrice(plan) method to safely retrieve the correct numeric value.




```
}

// changing to parseInt
int id = Integer.parseInt(null);
String stringPlan = planPriceTxt.getText();
int planPrice=Integer.parseInt(stringPlan);

//Check if ID already exists
```

```
// changing to parseInt
int id = Integer.parseInt(idStr);
String stringPlan = planPriceTxt.getText();
int planPrice=Integer.parseInt(stringPlan);
```

Error Type: Syntax Error

Description: The program did not compile because a semicolon was missing.
Cause: While taking the value from the input and setting it in a variable, one line did not end with a semicolon.
Correction: Added the missing semicolon at the end of the line so the code could compile successfully.

```
try {
    // storing input
    String idStr = idTxt.getText()
    String name = nameTxt.getText();
    String location = locationTxt
    String phone = phoneTxt.getText();
    String email = emailTxt.getText();
    String referral = referalTxt.getText();
    String planStr = planPriceTxt.getText();
    String selectedPlan = (String) planBox.getSelectedItem();

    String gender = null;
    if (male.isSelected()) gender = "Male";
    else if (female.isSelected()) gender = "Female";
    else if (other.isSelected()) gender = "Other";
```

';' expected

Figure 33: error 2

```
        }});
addReg.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            try {
                // storing input
                String idStr = idTxt.getText();
                String name = nameTxt.getText();
                String location = locationTxt.getText();
                String phone = phoneTxt.getText();
                String email = emailTxt.getText();
                String referral = referalTxt.getText();
                String planStr = planPriceTxt.getText();
                String selectedPlan = (String) planBox.getSelectedItem();

                String gender = null;
                if (male.isSelected()) gender = "Male";
                else if (female.isSelected()) gender = "Female";
                else if (other.isSelected()) gender = "Other";

                String day = (String) dobDay.getSelectedItem();
                String month = (String) dobMonth.getSelectedItem();
                String year = (String) dobYear.getSelectedItem();
                String dob = day + "/" + month + "/" + year;

                String startday = (String) startDay.getSelectedItem();
```

Figure 34: error 3 fix

Conclusion

This project is a Gym Membership Management System created using Java and Swing. It allows users to add, manage, and track both regular and premium members. Key features include attendance tracking, loyalty point calculation, plan upgrades, discount application, and file saving and reading.

During the development process, I applied object-oriented programming concepts such as inheritance, abstraction, and method overriding. I also learned how to handle user input, events, and errors. Some errors occurred, such as trying to convert non-numeric input and forgetting to handle premium members while saving. These were identified and corrected through testing.

The system works as expected and meets all the required functions. This project helped me improve my understanding of Java programming, GUI design, and error handling in real applications.

Appendix

```
public  abstract class GymMember{
    //declaring variables with protected access modifier according to the question
    protected int id;
    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;
    //creating a constructor to intitialize values according to the question
    public GymMember(int id,String name,String location,String phone,String email,String
gender,
    String DOB,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;
```

```
    }
    //creating accessor method for each attribute(variable)
    public int getId(){
        return id;
    }

    public String getDOB(){
        return DOB;
    }

    public String getName(){
        return name;
    }

    public String getPhone(){
        return phone;
    }

    public String getEmail(){
        return email;
    }

    public String getGender(){
        return gender;
    }

    public String getLocation(){
```

```
        return location;
    }
    public String getDob(){
        return DOB;

    }

    public String getMembershipStartDate(){
        return membershipStartDate;
    }

    public int getAttendance(){
        return attendance;
    }

    public double getLoyaltyPoints(){
        return loyaltyPoints;
    }

    public boolean getActiveStatus(){
        return activeStatus;
    }

    //creating abstract method markAttendance for the sub class
    public abstract void markAttendance();

    //creating method activate membership which when called will set the active status to true as active status is set to false by default
    public void activateMembership(){
```

```
        activeStatus=true;
    }
    //creating method deavtivateMembership which when called will set the active status to false if it is already false it displays a message


    public void deactivateMembership(){
        if(activeStatus==true){


            activeStatus=false;
        }else{
            System.out.println("You don't have an active membership");
        }
    }
    //reset membet method to reset the values of attendance and loyalty points to zero and active status to false
    public void resetMember(){
        activeStatus=false;
        attendance=0;
        loyaltyPoints=0;
    }
    // display method displays the details of the members
    public void display(){
        System.out.println("ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("Location: " + location);
        System.out.println("Phone: " + phone);
        System.out.println("Email: " + email);
        System.out.println("Gender: " + gender);
```

```
        System.out.println("Date of Birth: " + DOB);
        System.out.println("Membership Start Date: " + membershipStartDate);
        System.out.println("Attendance: " + attendance);
        System.out.println("Loyalty Points: " + loyaltyPoints);
        System.out.println("Active Status: " + activeStatus);
    }



public void setAttendance(int attendance) {
    this.attendance = attendance;
}
public void setActiveStatus(boolean activeStatus){
    this.activeStatus=activeStatus;


}
}
public class RegularMember extends GymMember{
    // declaring variables with access modifier private
    private final int attendanceLimit;
    private boolean isEligibleForUpgrade;
    private String removalReason;
    private String referralSource;
    private String plan;
    private double price;
    //creating a constructor to intialize the values
    public RegularMember(int id,String name,String location,String phone, String
email,String gender,String DOB,String plan,int price,String membershipStartDate,String
referralSource){
        //using super keyword to call the parent class constructor
```

```
        super(id, name, location, phone, email, gender, DOB, membershipStartDate);
        this.isEligibleForUpgrade = false;
        this.removalReason = "";
        this.referralSource = referralSource;
        this.plan = plan;
        this.price = price;
        this.attendanceLimit=30;

    }
    //accessor method for each attribute
    public int getAttendanceLimit(){
        return attendanceLimit;
    }

    public boolean getIsEligibleForUpgrade(){
        return isEligibleForUpgrade;
    }

    public String getRemovalReason(){
        return removalReason;
    }

    public String getReferralSource(){
        return referralSource;
    }

    public String getPlan(){
        return plan;
```

```
    }


    public double getPrice(){
      return this.price;
    }
    // overriding the markattendance as it is an abstract method  which increases the attendance and the loyalty points and checks if the user is eligible for an upgrade
    @Override
    public void markAttendance(){
        attendance++;
        loyaltyPoints+=5;
        if(attendance>=attendanceLimit){
            isEligibleForUpgrade=true;
        }
    }
    //method to check the price for membership plans
    public double getPlanPrice(String newPlan){
        switch(newPlan.toLowerCase()){
                case"basic" : return 6500;
            case "standard":return 12500;
            case "deluxe":return 18500;
            default: return -1;


        }
    }
    //checking if the user is eligible for upgrade
    public String upgradePlan(String newPlan) {
    if (!activeStatus) {
        return "Membership is not active. Cannot upgrade plan.";
```

```
    }

    if (this.plan.equalsIgnoreCase(newPlan)) {
        return "Plan is already active.";
    }

    double newPrice = getPlanPrice(newPlan);
    if (newPrice == -1) {
        return "Invalid plan name.";
    }

    if (!isEligibleForUpgrade) {
        return "Not eligible for upgrade yet.";
    }

    this.plan = newPlan.toLowerCase();
    this.price = newPrice;
    isEligibleForUpgrade = true;

    return "Plan upgraded successfully to " + newPlan;
}

    //method which resets the plan to "basic" price to 6500 asks for removal reason and
sets the isEligibleForUpgrade to false
    public void revertRegularMember(String removalReason) {
        super.resetMember();
        this.isEligibleForUpgrade = false;
        this.plan = "basic";
        this.price = 6500;
```

```
        this.removalReason = removalReason;
    }

    // Display details of the user
    @Override
    public void display() {
        super.display();
        System.out.println("Plan: " + plan);
        System.out.println("Price: " + price);
        if (removalReason !="") {
            System.out.println("Removal Reason: " + removalReason);
        }

    }
}

class PremiumMember extends GymMember {
    //declaring variables with private acccess modifier accourding to the question
    private final double premiumCharge ;
    private String personalTrainer;
    private boolean isFullPayment;
    private double paidAmount, discountAmount;
    //creating a constructor to pass the values
    public PremiumMember(int id, String name, String location, String phone, String
email, String gender, String DOB, String membershipStartDate, String personalTrainer)
{
        //using super keyword to call the parent class constructor
        super(id, name, location, phone, email, gender, DOB, membershipStartDate);
        this.premiumCharge=50000;
```

```
        this.personalTrainer = personalTrainer;
        this.isFullPayment = false;
        this.paidAmount = 0;
        this.discountAmount = 0;
    }
    //accessor for all the attributes
    public double getPremiumCharge() {
        return premiumCharge;
    }

    public String getPersonalTrainer() {
        return personalTrainer;
    }

    public boolean getIsFullPayment() {
        return isFullPayment;
    }

    public double getPaidAmount() {
        return paidAmount;
    }

    public double getDiscountAmount() {
        return discountAmount;
    }
    //overiding as it is an abstract method and inceasing attendance and loyalty points
    @Override
    public void markAttendance() {
```

```
        this.attendance++;
        this.loyaltyPoints += 10;
    }
    // method to check if the full payment is done or not
    public String payDueAmount(double amount) {
        if (isFullPayment==true) return "Payment is already completed.";
        if (paidAmount + amount > premiumCharge) return "Total payment exceeds
premium charge.";


        this.paidAmount += amount;
        if (this.paidAmount == premiumCharge) {
            this.isFullPayment = true;
        }
        double remainingAmount = premiumCharge - this.paidAmount;
        return "Payment successful.And the remaining amount is " + remainingAmount;
    }
    //calulating discount for preminum members
    public void calculateDiscount() {
        if (isFullPayment==true) {
            this.discountAmount = premiumCharge * 0.10;
            System.out.println("Discount is" + discountAmount);
        }
    }


    public void setDiscountAmount(double discount) {
        this.discountAmount = discount;
    }
    //method to reset premium members
    public void revertPremiumMember() {
```

```
        resetMember();
        this.personalTrainer = "";
        this.isFullPayment = false;
        this.paidAmount = 0;
        this.discountAmount = 0;
    }
    //method to diplay the details
    @Override
    public void display() {
        super.display();
        System.out.println("Personal Trainer: " + personalTrainer);
        System.out.println("Paid Amount: " + paidAmount);
        System.out.println("Full Payment: " + isFullPayment);
        System.out.println("Remaining Amount: " + (premiumCharge - paidAmount));
        if (isFullPayment) {
            System.out.println("Discount Amount: " + discountAmount);
        }
    }

}

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import java.io.IOException;
import java.io.FileWriter;
import java.io.BufferedReader;
```

```
import java.io.FileReader;
import java.io.File;


public class GymGUI {
    //creating instance variables
    private ArrayList<GymMember> members;


    private JFrame frame;
    private JLabel id, name, location, phone, email, gender, dob, plan, planPrice, startDate, referral, paidAmount,personalTrainer, removalReason, trainerName, primiumPrice, regularPrice, premiumPlan, regularPlan,pricePlan;
    private JTextField idTxt, nameTxt, locationTxt, planPriceTxt, phoneTxt, emailTxt, referalTxt, paidTxt, trainerTxt,personalTrainerTxt;
    private JButton regular, premium, display, Read,addReg, activateReg, markReg, revertReg, clearReg,  deactivateReg, addPrem, activatePrem, markPrem, revertPrem,discountPrem, clearPrem, payDue, deactivatePrem,upgradePlan,saveButtonReg, saveButtonPrem;
    private JTextArea removalTxt;
    private JComboBox<String> dobBox, startBox, planBox;
    private JRadioButton male, female, other;
    private JTextArea memberList;
    //creating constructors to intialize
    public GymGUI() {
        //creating frame
        frame = new JFrame("GYM SYSTEM");
        members= new ArrayList<>();
        members=new ArrayList<>();


        frame.setSize(500, 500);
        frame.getContentPane().setBackground(Color.CYAN);
```

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setLocationRelativeTo(null);
//cree

JPanel mainPanel = new JPanel(new GridBagLayout());
mainPanel.setBorder(BorderFactory.createTitledBorder("Home Page"));
GridBagConstraints gbc = new GridBagConstraints();
gbc.insets = new Insets(10, 10, 10, 10);

JLabel title = new JLabel("Gym Membership System", SwingConstants.CENTER);
title.setFont(new Font("Arial", Font.BOLD, 22));
title.setForeground(Color.RED);
gbc.gridwidth = 2;
gbc.gridx = 0;
gbc.gridy = 0;
gbc.anchor = GridBagConstraints.CENTER;
mainPanel.add(title, gbc);

regular = new JButton("Regular Member");
premium = new JButton("Premium Member");
display = new JButton("View Members");
Read =new JButton("Read From File");

buttonStyle(regular);
buttonStyle(premium);
buttonStyle(display);
buttonStyle(Read);
```

```
        gbc.gridy = 1;
        gbc.gridwidth = 1;
        gbc.gridx = 0;
        mainPanel.add(regular, gbc);

        gbc.gridx = 1;
        mainPanel.add(premium, gbc);

        gbc.gridy = 2;
        gbc.gridx = 0;
        mainPanel.add(display, gbc);

        gbc.gridy=2;
        gbc.gridx=1;
        mainPanel.add(Read, gbc);

        regular.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                JFrame regularFrame = new JFrame("Add Regular Member");
                regularFrame.setSize(700, 700);
                regularFrame.setLocationRelativeTo(null);
                regularFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

                JPanel regularPanel = new JPanel(new GridBagLayout());
                regularPanel.setBorder(BorderFactory.createTitledBorder("Regular
Members"));
                GridBagConstraints gbc = new GridBagConstraints();
                gbc.insets = new Insets(5, 5, 5, 5);
```

```
gbc.fill = GridBagConstraints.HORIZONTAL;

JLabel id = new JLabel("ID");
gbc.gridx = 0;
gbc.gridy = 0;
regularPanel.add(id, gbc);
JTextField idTxt = new JTextField(20);
gbc.gridx = 1;
regularPanel.add(idTxt, gbc);

JLabel name = new JLabel("Name");
gbc.gridy = 1;
gbc.gridx = 0;
regularPanel.add(name, gbc);
JTextField nameTxt = new JTextField(20);
gbc.gridx = 1;
regularPanel.add(nameTxt, gbc);

JLabel location = new JLabel("Location");
gbc.gridy = 2;
gbc.gridx = 0;
regularPanel.add(location, gbc);
JTextField locationTxt = new JTextField(20);
gbc.gridx = 1;
regularPanel.add(locationTxt, gbc);

JLabel phone = new JLabel("Phone");
gbc.gridy = 3;
```

```
gbc.gridx = 0;
regularPanel.add(phone, gbc);
JTextField phoneTxt = new JTextField(20);
gbc.gridx = 1;
regularPanel.add(phoneTxt, gbc);

JLabel email = new JLabel("Email");
gbc.gridy = 4;
gbc.gridx = 0;
regularPanel.add(email, gbc);
JTextField emailTxt = new JTextField(20);
gbc.gridx = 1;
regularPanel.add(emailTxt, gbc);

JLabel gender = new JLabel("Gender");
gbc.gridx = 0;
gbc.gridy = 5;
regularPanel.add(gender, gbc);

JRadioButton male = new JRadioButton("Male");
JRadioButton female = new JRadioButton("Female");
JRadioButton other = new JRadioButton("Other");
ButtonGroup genderGroup = new ButtonGroup();
genderGroup.add(male);
genderGroup.add(female);
genderGroup.add(other);

JPanel genderPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5,
0));
```

```
        genderPanel.add(male);
        genderPanel.add(female);
        genderPanel.add(other);
        gbc.gridx = 1;
        regularPanel.add(genderPanel, gbc);

        // Arrays for year, month, day for DOB
        String[] yearsArray = new String[61];  // 1970 to 2030
        for (int i = 0; i < 61; i++) {
            yearsArray[i] = String.valueOf(1970 + i);
        }

        String[] monthArray = { "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
"Sep", "Oct", "Nov", "Dec" };

        String[] dayArray = new String[31];
        for (int i = 0; i < 31; i++) {
            dayArray[i] = String.valueOf(i + 1);
        }

        // Date of Birth
        JLabel dob = new JLabel("Date of Birth:");
        gbc.gridx = 0;
        gbc.gridy = 6;
        regularPanel.add(dob, gbc);

        JComboBox<String> dobDay = new JComboBox<>(dayArray);
        JComboBox<String> dobMonth = new JComboBox<>(monthArray);
        JComboBox<String> dobYear = new JComboBox<>(yearsArray);
```

```
JPanel dobPanel = new JPanel(new FlowLayout());
dobPanel.add(dobDay);
dobPanel.add(dobMonth);
dobPanel.add(dobYear);
gbc.gridx = 1;
regularPanel.add(dobPanel, gbc);

plan= new JLabel("Plan");
gbc.gridx=0;
gbc.gridy=7;
regularPanel.add(plan,gbc);

String[] planArray={"Basic","Standard","Delux"};
JComboBox<String> planBox = new JComboBox<>(planArray);
gbc.gridx=1;
gbc.gridy=7;
regularPanel.add(planBox,gbc);

pricePlan=new JLabel("Price Plan");
gbc.gridx=0;
gbc.gridy=8;
regularPanel.add(pricePlan,gbc);
planPriceTxt=new JTextField("6500");
gbc.gridx=1;
gbc.gridy=8;
planPriceTxt.setEditable(false);
```

```
regularPanel.add(planPriceTxt,gbc);

//start date label and combo box
startDate = new JLabel("Start Date:");
gbc.gridy = 9;
gbc.gridx=0;
regularPanel.add(startDate, gbc);
JComboBox <String> startDay=new JComboBox<>(dayArray);
JComboBox<String> startMonth=new JComboBox<>(monthArray);
JComboBox<String> startYear=new JComboBox<>(yearsArray);
JPanel startPanel=new JPanel(new FlowLayout());
startPanel.add(startDay);
startPanel.add(startMonth);
startPanel.add(startYear);

gbc.gridy=9;
gbc.gridx=1;
gbc.fill = GridBagConstraints.HORIZONTAL;
regularPanel.add(startPanel, gbc);
//referral label and text field
referral = new JLabel("Referral");
gbc.gridy = 10;
gbc.gridx=0;
regularPanel.add(referral, gbc);
referalTxt=new JTextField(20);
gbc.gridx=1;
regularPanel.add(referalTxt,gbc);
//removal label and text area
```

```
regularFrame.add(regularPanel);
regularFrame.setVisible(true);

addReg= new JButton("Add Regular Member");
gbc.gridx=0;
gbc.gridy=12;
regularPanel.add(addReg,gbc);

activateReg= new JButton("Activate Regular Member");
gbc.gridx=1;
gbc.gridy=12;
regularPanel.add(activateReg,gbc);

markReg= new JButton("Mark Attendance");
gbc.gridx=0;
gbc.gridy=13;
regularPanel.add(markReg,gbc);

revertReg= new JButton("Revert Regular Member");
gbc.gridx=1;
gbc.gridy=13;
regularPanel.add(revertReg,gbc);

clearReg=new JButton("Clear form");
gbc.gridx=0;
gbc.gridy=14;
regularPanel.add(clearReg,gbc);
```

```
deactivateReg=new JButton("Deactivate");
gbc.gridx=1;
gbc.gridy=14;
gbc.gridwidth=2;
regularPanel.add(deactivateReg,gbc);

upgradePlan=new JButton("Upgrade Plan");
gbc.gridx=0;
gbc.gridy=15;
gbc.gridwidth=1;
regularPanel.add(upgradePlan,gbc);

saveButtonReg = new JButton("Save to File");
gbc.gridx = 1;
gbc.gridy = 15; // or next available row
regularPanel.add(saveButtonReg, gbc);

buttonStyle(upgradePlan);
buttonStyle(addReg);
buttonStyle(activateReg);
buttonStyle(markReg);
buttonStyle(revertReg);
buttonStyle(clearReg);
buttonStyle(deactivateReg);
buttonStyle(saveButtonReg);

planBox.addActionListener(new ActionListener() {
```

```
        public void actionPerformed(ActionEvent e) {
            String selectedPlan = (String) planBox.getSelectedItem();
            if ("Basic".equals(selectedPlan)) {
                planPriceTxt.setText("6500");

            }else if("Standard".equals(selectedPlan)){
                planPriceTxt.setText("12500");
            }else if("Delux".equals(selectedPlan)){
                planPriceTxt.setText("18000");
            }

        }});
    addReg.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            try {
                // storing input
                String idStr = idTxt.getText();
                String name = nameTxt.getText();
                String location = locationTxt.getText();
                String phone = phoneTxt.getText();
                String email = emailTxt.getText();
                String referral = referalTxt.getText();
                String planStr = planPriceTxt.getText();
                String selectedPlan = (String) planBox.getSelectedItem();

                String gender = null;
                if (male.isSelected()) gender = "Male";
                else if (female.isSelected()) gender = "Female";
```

```
                else if (other.isSelected()) gender = "Other";

                String day = (String) dobDay.getSelectedItem();
                String month = (String) dobMonth.getSelectedItem();
                String year = (String) dobYear.getSelectedItem();
                String dob = day + "/" + month + "/" + year;

                String startday = (String) startDay.getSelectedItem();
                String startmonth = (String) startMonth.getSelectedItem();
                String startyear = (String) startYear.getSelectedItem();
                String startDate = startday + "/" + startmonth + "/" + startyear;

                //  Check for empty fields
                if (idStr.isEmpty() || name.isEmpty() || location.isEmpty() ||
phone.isEmpty() ||
                email.isEmpty() || referral.isEmpty() || planStr.isEmpty() ||
                gender == null || day == null || month == null || year == null ||
                startday == null || startmonth == null || startyear == null ||
selectedPlan == null) {

                    JOptionPane.showMessageDialog(null, "Please fill in all the
fields.", "Incomplete Form", JOptionPane.WARNING_MESSAGE);
                    return;
                }

                // changing to parseInt
                int id = Integer.parseInt(idStr);
                String stringPlan = planPriceTxt.getText();
                int planPrice=Integer.parseInt(stringPlan);
```

```
                    //Check if ID already exists
                    boolean exists = false;
                    for (GymMember member : members) {
                        if (member.getId() == id) {
                            exists = true;
                            break;
                        }
                    }

                    // Step 5: Handle based on existence
                    if (exists) {
                        JOptionPane.showMessageDialog(null, "Member ID already exists. Please use a unique ID.", "Duplicate ID", JOptionPane.ERROR_MESSAGE);
                    } else {
                        RegularMember newMember = new RegularMember(id, name, location, phone, email, gender, dob,selectedPlan,  planPrice,startDate, referral);
                        members.add(newMember);
                        JOptionPane.showMessageDialog(null, "Member added successfully!", "Information", JOptionPane.INFORMATION_MESSAGE);

                    }
                } catch (NumberFormatException nfe) {
                    JOptionPane.showMessageDialog(null, "Please enter valid numbers.", "Invalid Input", JOptionPane.ERROR_MESSAGE);
                } catch (Exception ex) {
                    JOptionPane.showMessageDialog(null, "An unexpected error occurred: " + ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
                }
            }
```

```
        });
        clearReg.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                //clearing all the textfields radio button and combo box
                idTxt.setText("");
                nameTxt.setText("");
                locationTxt.setText("");
                phoneTxt.setText("");
                emailTxt.setText("");
                referalTxt.setText("");
                planBox.setSelectedIndex(0);
                genderGroup.clearSelection();
                dobDay.setSelectedIndex(0);
                dobMonth.setSelectedIndex(0);
                dobYear.setSelectedIndex(0);
                startDay.setSelectedIndex(0);
                startMonth.setSelectedIndex(0);
                startYear.setSelectedIndex(0);

            }
        });
        markReg.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String inputId = JOptionPane.showInputDialog(null, "Enter Member
ID to mark attendance:");

                try {
```

```
                int memberId = Integer.parseInt(inputId);
                boolean found = false;

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

                        if (member.getActiveStatus()) {
                            member.markAttendance(); // ☑ Correct way to handle it
                            JOptionPane.showMessageDialog(null,
                                "Attendance marked for Member ID: " + memberId +
                                "\nTotal Attendance: " + member.getAttendance() +
                                "\nLoyalty Points: " + member.getLoyaltyPoints());
                        } else {
                            JOptionPane.showMessageDialog(null,
                                "Cannot mark attendance. Membership is
deactivated.",
                                "Inactive Member",
JOptionPane.WARNING_MESSAGE);
                        }
                        break;
                    }
                }

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

```
                } catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(null,
                        "Invalid ID. Please enter a numeric value.",
                        "Input Error", JOptionPane.WARNING_MESSAGE);
                }
            }
        });

        revertReg.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String input = JOptionPane.showInputDialog(null, "Enter Member ID to remove:");

                if (input != null) {  // Only proceed if user didn't cancel
                    try {
                        int id = Integer.parseInt(input);
                        boolean found = false;

                        // Ask for removal reason
                        String removalReason = JOptionPane.showInputDialog(null, "Enter removal reason:");

                        if (removalReason == null || removalReason.isEmpty()) {
                            JOptionPane.showMessageDialog(null, "Removal reason is required.", "Error", JOptionPane.ERROR_MESSAGE);
                            return;  // Exit if no reason is provided
                        }
```

```
                    // Iterate over members to find the one with the given ID
                    for (GymMember member : members) {
                        if (member.getId() == id) {
                            found = true;

                            // Remove the member from the list
                            members.remove(member);
                            JOptionPane.showMessageDialog(null, "Member removed successfully. Reason: " + removalReason, "Success", JOptionPane.INFORMATION_MESSAGE);
                            break;
                        }
                    }

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

                } catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(null, "Invalid ID format. Please enter a number.", "Input Error", JOptionPane.ERROR_MESSAGE);
                }
            }
        }
    });
    activateReg.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
```

```
                String input = JOptionPane.showInputDialog(null, "Enter Member ID to activate:");

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

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

                            if (member.getActiveStatus()) {
                                JOptionPane.showMessageDialog(null, "Member is already active.", "Info", JOptionPane.INFORMATION_MESSAGE);
                            } else {
                                member.setActiveStatus(true);
                                JOptionPane.showMessageDialog(null, "Member activated successfully.", "Success", JOptionPane.INFORMATION_MESSAGE);
                            }
                            break;
                        }
                    }

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

                } catch (NumberFormatException ex) {
```

```
                    JOptionPane.showMessageDialog(null, "Invalid ID format. Please
enter a number.", "Input Error", JOptionPane.ERROR_MESSAGE);
                }


            }
        });
        deactivateReg.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String input = JOptionPane.showInputDialog(null, "Enter Member ID
to deactivate:");

                if (input != null) {  // Only proceed if user didn't cancel
                    try {
                        int id = Integer.parseInt(input);
                        boolean found = false;

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

                                if (!member.getActiveStatus()) {
                                    JOptionPane.showMessageDialog(null, "Member is
already inactive.", "Info", JOptionPane.INFORMATION_MESSAGE);
                                } else {
                                    member.setActiveStatus(false);
                                    JOptionPane.showMessageDialog(null, "Member
deactivated successfully.", "Success", JOptionPane.INFORMATION_MESSAGE);
                                }
                                break;
```

```
                    }
                }

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

            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(null, "Invalid ID format. Please enter a number.", "Input Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    }
});
upgradePlan.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        JTextField idField = new JTextField();
        String[] plans = {"Basic", "Standard", "Deluxe"};
        JComboBox<String> planCombo = new JComboBox<String>(plans);

        JPanel panel = new JPanel(new GridLayout(2, 2));
        panel.add(new JLabel("Enter Member ID:"));
        panel.add(idField);
        panel.add(new JLabel("Select New Plan:"));
        panel.add(planCombo);
```

```
                int result = JOptionPane.showConfirmDialog(null, panel, "Upgrade
Plan", JOptionPane.OK_CANCEL_OPTION);

                if (result == JOptionPane.OK_OPTION) {
                    try {
                        int id = Integer.parseInt(idField.getText());
                        String newPlan = (String) planCombo.getSelectedItem();
                        boolean found = false;

                        for (int i = 0; i < members.size(); i++) {
                            GymMember member = members.get(i);
                            if (member instanceof RegularMember && member.getId()
== id) {
                                if (!member.getActiveStatus()) {
                                    JOptionPane.showMessageDialog(null, "Membership
must be active to upgrade.");
                                    return;
                                }
                                String message = ((RegularMember)
member).upgradePlan(newPlan);
                                JOptionPane.showMessageDialog(null, message);
                                found = true;
                                break;
                            }
                        }

                        if (!found) {
                            JOptionPane.showMessageDialog(null, "No Regular
Member found with ID: " + id);
                        }
```

```
                        } catch (NumberFormatException ex) {
                            JOptionPane.showMessageDialog(null, "Invalid ID. Please
enter a number.");
                        }
                    }
                }
            });
            saveButtonReg.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    saveToFile();
                }
            });
            regularFrame.add(regularPanel);
            regularFrame.setVisible(true);

        }
    });
    display.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if (members.isEmpty()) {
                JOptionPane.showMessageDialog(frame, "No members to display.");
                return;
            }

            JTextArea displayArea = new JTextArea(20, 60);
            displayArea.setEditable(false);

            String output = "";
```

```
            for (GymMember member : members) {
                output += "----------- Member Info -----------\n";
                if (member instanceof RegularMember) {
                    RegularMember rm = (RegularMember) member;
                    output += "Member Type: Regular Member\n";
                    output += "ID: " + rm.getId() + "\n";
                    output += "Name: " + rm.getName() + "\n";
                    output += "Location: " + rm.getLocation() + "\n";
                    output += "Phone: " + rm.getPhone() + "\n";
                    output += "Email: " + rm.getEmail() + "\n";
                    output += "Gender: " + rm.getGender() + "\n";
                    output += "DOB: " + rm.getDOB() + "\n";
                    output += "Membership Start Date: " + rm.membershipStartDate +
"\n";
                    output += "Attendance: " + rm.attendance + "\n";
                    output += "Loyalty Points: " + rm.loyaltyPoints + "\n";
                    output += "Active Status: " + rm.activeStatus + "\n";
                    output += "Plan: " + rm.getPlan() + "\n";
                    output += "Price: " + rm.getPrice() + "\n";
                    output += "Referral Source: " + rm.getReferralSource() + "\n";
                    if (!rm.getRemovalReason().isEmpty()) {
                        output += "Removal Reason: " + rm.getRemovalReason() + "\n";
                    }
                } else if (member instanceof PremiumMember) {
                    PremiumMember pm = (PremiumMember) member;
                    output += "Member Type: Premium Member\n";
                    output += "ID: " + pm.getId() + "\n";
                    output += "Name: " + pm.getName() + "\n";
                    output += "Location: " + pm.getLocation() + "\n";
```

```
                output += "Phone: " + pm.getPhone() + "\n";
                output += "Email: " + pm.getEmail() + "\n";
                output += "Gender: " + pm.getGender() + "\n";
                output += "DOB: " + pm.getDOB() + "\n";
                output += "Membership Start Date: " + pm.membershipStartDate +
"\n";
                output += "Attendance: " + pm.attendance + "\n";
                output += "Loyalty Points: " + pm.loyaltyPoints + "\n";
                output += "Active Status: " + pm.activeStatus + "\n";
                output += "Personal Trainer: " + pm.getPersonalTrainer() + "\n";
                output += "Paid Amount: " + pm.getPaidAmount() + "\n";
                output += "Is Full Payment: " + pm.getIsFullPayment() + "\n";
                double remaining = pm.getPremiumCharge() - pm.getPaidAmount();
                output += "Remaining Amount: " + remaining + "\n";
                if (pm.getIsFullPayment()) {
                    output += "Discount Amount: " + pm.getDiscountAmount() + "\n";
                }
            }
            output += "\n";
        }

        displayArea.setText(output);
        JScrollPane scrollPane = new JScrollPane(displayArea);
        JFrame displayFrame = new JFrame("Member Details");
        displayFrame.add(scrollPane);
        displayFrame.pack();
        displayFrame.setLocationRelativeTo(null);
        displayFrame.setVisible(true);
    }
```

```
        });

    premium.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            JFrame premiumFrame = new JFrame("Add Premium Member");
            premiumFrame.setSize(700, 700);
            premiumFrame.setLocationRelativeTo(null);

premiumFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

            JPanel premiumPanel = new JPanel(new GridBagLayout());
            premiumPanel.setBorder(BorderFactory.createTitledBorder("Premium
Members"));
            GridBagConstraints gbc = new GridBagConstraints();
            gbc.insets = new Insets(5, 5, 5, 5);
            gbc.fill = GridBagConstraints.HORIZONTAL;

            // ID Label and TextField
            JLabel id = new JLabel("ID");
            gbc.gridx = 0;
            gbc.gridy = 0;
            premiumPanel.add(id, gbc);
            JTextField idTxt = new JTextField(20);
            gbc.gridx = 1;
            premiumPanel.add(idTxt, gbc);

            // Name Label and TextField
            JLabel name = new JLabel("Name");
```

```
gbc.gridy = 1;
gbc.gridx = 0;
premiumPanel.add(name, gbc);
JTextField nameTxt = new JTextField(20);
gbc.gridx = 1;
premiumPanel.add(nameTxt, gbc);

// Location Label and TextField
JLabel location = new JLabel("Location");
gbc.gridy = 2;
gbc.gridx = 0;
premiumPanel.add(location, gbc);
JTextField locationTxt = new JTextField(20);
gbc.gridx = 1;
premiumPanel.add(locationTxt, gbc);

// Phone Label and TextField
JLabel phone = new JLabel("Phone");
gbc.gridy = 3;
gbc.gridx = 0;
premiumPanel.add(phone, gbc);
JTextField phoneTxt = new JTextField(20);
gbc.gridx = 1;
premiumPanel.add(phoneTxt, gbc);

// Email Label and TextField
JLabel email = new JLabel("Email");
gbc.gridy = 4;
```

```
        gbc.gridx = 0;
        premiumPanel.add(email, gbc);
        JTextField emailTxt = new JTextField(20);
        gbc.gridx = 1;
        premiumPanel.add(emailTxt, gbc);

        // Gender Label and Radio Buttons
        JLabel gender = new JLabel("Gender");
        gbc.gridx = 0;
        gbc.gridy = 5;
        premiumPanel.add(gender, gbc);

        JRadioButton male = new JRadioButton("Male");
        JRadioButton female = new JRadioButton("Female");
        JRadioButton other = new JRadioButton("Other");
        ButtonGroup genderGroup = new ButtonGroup();
        genderGroup.add(male);
        genderGroup.add(female);
        genderGroup.add(other);

        JPanel genderPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5,
0));
        genderPanel.add(male);
        genderPanel.add(female);
        genderPanel.add(other);
        gbc.gridx = 1;
        premiumPanel.add(genderPanel, gbc);

        // Arrays for year, month, day for DOB
```

```
        String[] yearsArray = new String[61];  // 1970 to 2030
        for (int i = 0; i < 61; i++) {
            yearsArray[i] = String.valueOf(1970 + i);
        }

        String[] monthArray = { "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
"Sep", "Oct", "Nov", "Dec" };

        String[] dayArray = new String[31];
        for (int i = 0; i < 31; i++) {
            dayArray[i] = String.valueOf(i + 1);
        }

        // Date of Birth
        JLabel dob = new JLabel("Date of Birth:");
        gbc.gridx = 0;
        gbc.gridy = 6;
        premiumPanel.add(dob, gbc);

        JComboBox<String> dobDay = new JComboBox<>(dayArray);
        JComboBox<String> dobMonth = new JComboBox<>(monthArray);
        JComboBox<String> dobYear = new JComboBox<>(yearsArray);

        JPanel dobPanel = new JPanel(new FlowLayout());
        dobPanel.add(dobDay);
        dobPanel.add(dobMonth);
        dobPanel.add(dobYear);
        gbc.gridx = 1;
        premiumPanel.add(dobPanel, gbc);
```

```
// Plan (Uneditable and Always Premium)
JLabel plan = new JLabel("Plan");
gbc.gridx = 0;
gbc.gridy = 7;
premiumPanel.add(plan, gbc);

JTextField planTxt = new JTextField("Premium");
planTxt.setEditable(false); // Make it uneditable
gbc.gridx = 1;
premiumPanel.add(planTxt, gbc);

JLabel pricePlan = new JLabel("Price Plan");
gbc.gridx = 0;
gbc.gridy = 8;
premiumPanel.add(pricePlan, gbc);

JTextField planPriceTxt = new JTextField("50000");
planPriceTxt.setEditable(false); // Make it uneditable
gbc.gridx = 1;
premiumPanel.add(planPriceTxt, gbc);

// Start Date
JLabel startDate = new JLabel("Start Date:");
gbc.gridy = 9;
gbc.gridx = 0;
premiumPanel.add(startDate, gbc);
```

```
JComboBox<String> startDay = new JComboBox<>(dayArray);
JComboBox<String> startMonth = new JComboBox<>(monthArray);
JComboBox<String> startYear = new JComboBox<>(yearsArray);

JPanel startPanel = new JPanel(new FlowLayout());
startPanel.add(startDay);
startPanel.add(startMonth);
startPanel.add(startYear);
gbc.gridx = 1;
premiumPanel.add(startPanel, gbc);

// Referral
JLabel personalTrainer = new JLabel("Personal Trainer");
gbc.gridy = 10;
gbc.gridx = 0;
premiumPanel.add(personalTrainer, gbc);

JTextField personalTrainerTxt = new JTextField(20);
gbc.gridx = 1;
premiumPanel.add(personalTrainerTxt, gbc);

// Discount Calculation Button and Display Field
JButton calculateDiscount = new JButton("Calculate Discount");
gbc.gridx = 0;
gbc.gridy = 11;
premiumPanel.add(calculateDiscount,gbc);

JLabel discountLabel = new JLabel("Discount");
```

```
gbc.gridx = 0;
gbc.gridy = 12;
premiumPanel.add(discountLabel, gbc);

JTextField discountTxt = new JTextField();
discountTxt.setEditable(false);  // Make the discount text field uneditable
gbc.gridx = 1;
premiumPanel.add(discountTxt, gbc);

// Adding the panel to the frame
premiumFrame.add(premiumPanel);
premiumFrame.setVisible(true);

// Buttons
JButton addPrem = new JButton("Add Premium Member");
gbc.gridx = 0;
gbc.gridy = 13;
premiumPanel.add(addPrem, gbc);

JButton activatePrem = new JButton("Activate Premium Member");
gbc.gridx = 1;
gbc.gridy = 13;
premiumPanel.add(activatePrem, gbc);

JButton markPrem = new JButton("Mark Attendance");
gbc.gridx = 0;
gbc.gridy = 14;
premiumPanel.add(markPrem, gbc);
```

```
JButton revertPrem = new JButton("Revert Premium Member");
gbc.gridx = 1;
gbc.gridy = 14;
premiumPanel.add(revertPrem, gbc);
//clear button
JButton clearPrem = new JButton("Clear Form");
gbc.gridx = 0;
gbc.gridy = 15;
premiumPanel.add(clearPrem, gbc);
//deactivate button
JButton deactivatePrem = new JButton("Deactivate");
gbc.gridx = 1;
gbc.gridy = 15;
premiumPanel.add(deactivatePrem, gbc);

JButton payDueButton = new JButton("Pay Due Amount");
gbc.gridx = 0;
gbc.gridy = 16;
premiumPanel.add(payDueButton, gbc);
buttonStyle(payDueButton);

saveButtonPrem = new JButton("Save to File");
gbc.gridx = 1;
gbc.gridy = 16; // adjust row index
premiumPanel.add(saveButtonPrem, gbc);
buttonStyle(saveButtonPrem);
```

```
        saveButtonPrem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                saveToFile();
            }
        });

        // Button Styling
        buttonStyle(addPrem);
        buttonStyle(activatePrem);
        buttonStyle(markPrem);
        buttonStyle(revertPrem);
        buttonStyle(clearPrem);
        buttonStyle(deactivatePrem);
        buttonStyle(calculateDiscount);

        addPrem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    // store the input as strings
                    String idStr = idTxt.getText();
                    String name = nameTxt.getText();
                    String location = locationTxt.getText();
                    String phone = phoneTxt.getText();
                    String email = emailTxt.getText();

                    String personalTrainer = personalTrainerTxt.getText();

                    // For Premium Member the Plan is always Premium and Price is
always 50000
```

```
                String selectedPlan = "Premium";
                String planStr = "50000";  // Price for Premium plan is always
50000

                String gender = null;
                if (male.isSelected()) gender = "Male";
                else if (female.isSelected()) gender = "Female";
                else if (other.isSelected()) gender = "Other";

                String day = (String) dobDay.getSelectedItem();
                String month = (String) dobMonth.getSelectedItem();
                String year = (String) dobYear.getSelectedItem();
                String dob = day + "/" + month + "/" + year;

                String startday = (String) startDay.getSelectedItem();
                String startmonth = (String) startMonth.getSelectedItem();
                String startyear = (String) startYear.getSelectedItem();
                String startDate = startday + "/" + startmonth + "/" + startyear;

                //checking if the fields are empty
                if (idStr.isEmpty() || name.isEmpty() || location.isEmpty() ||
phone.isEmpty() ||
                email.isEmpty()  || planStr.isEmpty() ||
                gender == null || day == null || month == null || year == null ||
                startday == null || startmonth == null || startyear == null ||
selectedPlan == null) {

                    JOptionPane.showMessageDialog(null, "Please fill in all the
fields.", "Incomplete Form", JOptionPane.WARNING_MESSAGE);
                    return;
```

```
                    }

                    //converting to parse
                    int id = Integer.parseInt(idStr);
                    int planPrice = Integer.parseInt(planStr);

                    //Check if ID already exists
                    boolean exists = false;
                    for (GymMember member : members) {
                        if (member.getId() == id) {
                            exists = true;
                            break;
                        }
                    }

                    //Handle based on existence
                    if (exists) {
                        JOptionPane.showMessageDialog(null, "Member ID already exists. Please use a unique ID.", "Duplicate ID", JOptionPane.ERROR_MESSAGE);
                    } else {
                        PremiumMember newPremiumMember = new PremiumMember( id, name, location, phone, email, gender,dob, startDate, personalTrainer);
                        members.add(newPremiumMember);
                        JOptionPane.showMessageDialog(null, "Premium Member added successfully!", "Information", JOptionPane.INFORMATION_MESSAGE);

                    }
                } catch (NumberFormatException nfe) {
```

```
                    JOptionPane.showMessageDialog(null, "Please enter valid
numbers for ID and Plan Price.", "Invalid Input", JOptionPane.ERROR_MESSAGE);
                } catch (Exception ex) {
                    JOptionPane.showMessageDialog(null, "An unexpected error
occurred: " + ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
                }
            }
        });

        markPrem.addActionListener(new ActionListener(){
            @Override
            public void actionPerformed(ActionEvent e){
                String inputId = JOptionPane.showInputDialog(null, "Enter Member
ID to mark attendance:");

                try {
                    int memberId = Integer.parseInt(inputId);
                    boolean found = false;

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

                            if (member.getActiveStatus()) {
                                member.setAttendance(member.getAttendance() + 1);
                                JOptionPane.showMessageDialog(null, "Attendance
marked for Member ID: " + memberId +
                                        "\nTotal Attendance: " + member.getAttendance());
                            } else {
```

```
                                    JOptionPane.showMessageDialog(null, "Cannot mark
attendance. Membership is deactivated.",
                                        "Inactive Member",
JOptionPane.WARNING_MESSAGE);
                                }
                                break;
                            }
                        }

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

                    } catch (NumberFormatException ex) {
                        JOptionPane.showMessageDialog(null, "Invalid ID. Please enter
a numeric value.",
                            "Input Error", JOptionPane.WARNING_MESSAGE);
                    }
                }
            });
        calculateDiscount.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String idStr = JOptionPane.showInputDialog(null, "Enter Member
ID:");

                try {
                    int id = Integer.parseInt(idStr);
                    boolean found = false;
```

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

                        if (pm.getIsFullPayment()) {
                            double discount = 0.10 * 50000; // 10% of
premiumCharge
                            pm.setDiscountAmount(discount);
                            discountTxt.setText(String.valueOf((int) discount));
                            JOptionPane.showMessageDialog(null, "Discount of " +
(int) discount + " applied.");
                        } else {
                            discountTxt.setText("");
                            JOptionPane.showMessageDialog(null, "Full payment not
completed. Discount not applied.");
                        }

                        break;
                    }
                }

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

            } catch (Exception ex) {
                JOptionPane.showMessageDialog(null, "Invalid ID.");
```

```
                }
            }
        });

        revertPrem.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String input = JOptionPane.showInputDialog(null, "Enter Member ID to remove:");

                if (input != null) {  // Only proceed if user didn't cancel
                    try {
                        int id = Integer.parseInt(input);
                        boolean found = false;

                        // Ask for removal reason
                        String removalReason = JOptionPane.showInputDialog(null, "Enter removal reason:");

                        if (removalReason == null || removalReason.isEmpty()) {
                            JOptionPane.showMessageDialog(null, "Removal reason is required.", "Error", JOptionPane.ERROR_MESSAGE);
                            return;  // Exit if no reason is provided
                        }

                        // Iterate over members to find the one with the given ID
                        for (GymMember member : members) {
                            if (member.getId() == id) {
                                found = true;
```

```
                        // Remove the member from the list
                        members.remove(member);
                        JOptionPane.showMessageDialog(null, "Member removed successfully. Reason: " + removalReason, "Success", JOptionPane.INFORMATION_MESSAGE);
                        break;
                    }
                }

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

            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(null, "Invalid ID format. Please enter a number.", "Input Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    }
});
activatePrem.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        String input = JOptionPane.showInputDialog(null, "Enter Member ID to activate:");

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

```
                    boolean found = false;

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

                            if (member.getActiveStatus()) {
                                JOptionPane.showMessageDialog(null, "Member is already active.", "Info", JOptionPane.INFORMATION_MESSAGE);
                            } else {
                                member.setActiveStatus(true);
                                JOptionPane.showMessageDialog(null, "Member activated successfully.", "Success", JOptionPane.INFORMATION_MESSAGE);
                            }
                            break;
                        }
                    }

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

                } catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(null, "Invalid ID format. Please enter a number.", "Input Error", JOptionPane.ERROR_MESSAGE);
                }

            }
        });
```

```
        deactivatePrem.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String input = JOptionPane.showInputDialog(null, "Enter Member ID to deactivate:");

                if (input != null) {  // Only proceed if user didn't cancel
                    try {
                        int id = Integer.parseInt(input);
                        boolean found = false;

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

                                if (!member.getActiveStatus()) {
                                    JOptionPane.showMessageDialog(null, "Member is already inactive.", "Info", JOptionPane.INFORMATION_MESSAGE);
                                } else {
                                    member.setActiveStatus(false);
                                    JOptionPane.showMessageDialog(null, "Member deactivated successfully.", "Success", JOptionPane.INFORMATION_MESSAGE);
                                }
                                break;
                            }
                        }

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

```
                    }


                } catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(null, "Invalid ID format.
Please enter a number.", "Input Error", JOptionPane.ERROR_MESSAGE);
                }
            }
        }
    });
    clearPrem.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            idTxt.setText("");
            nameTxt.setText("");
            locationTxt.setText("");
            phoneTxt.setText("");
            emailTxt.setText("");
            personalTrainerTxt.setText("");
            genderGroup.clearSelection();
            dobDay.setSelectedIndex(0);
            dobMonth.setSelectedIndex(0);
            dobYear.setSelectedIndex(0);
            startDay.setSelectedIndex(0);
            startMonth.setSelectedIndex(0);
            startYear.setSelectedIndex(0);
            discountTxt.setText("");

        }
    });
```

```
            payDueButton.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    JTextField idField = new JTextField();
                    JTextField amountField = new JTextField();

                    JPanel panel = new JPanel(new GridLayout(2, 2));
                    panel.add(new JLabel("Enter Member ID:"));
                    panel.add(idField);
                    panel.add(new JLabel("Enter Amount to Pay:"));
                    panel.add(amountField);

                    int result = JOptionPane.showConfirmDialog(null, panel, "Pay Due
Amount", JOptionPane.OK_CANCEL_OPTION);

                    if (result == JOptionPane.OK_OPTION) {
                        try {
                            int id = Integer.parseInt(idField.getText());
                            double amount = Double.parseDouble(amountField.getText());
                            boolean found = false;

                            for (GymMember member : members) {
                                if (member instanceof PremiumMember && member.getId()
== id) {
                                    found = true;
                                    PremiumMember pm = (PremiumMember) member;
                                    String message = pm.payDueAmount(amount);
                                    JOptionPane.showMessageDialog(null, message);
                                    break;
```

```
                    }
                }

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

            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(null, "Invalid input. Please enter numeric values only.", "Input Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    }
});

    }
});
    Read.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    try {
        File file = new File("MemberDetails.txt");

        if (!file.exists()) {
            JOptionPane.showMessageDialog(null, "No saved member file found.", "Info", JOptionPane.INFORMATION_MESSAGE);
            return;
        }
```

```
            BufferedReader reader = new BufferedReader(new FileReader(file));
            String content = "";
            String line;

            while ((line = reader.readLine()) != null) {
                content += line + "\n";
            }

            reader.close();

            JTextArea textArea = new JTextArea(content);
            textArea.setEditable(false);
            JScrollPane scrollPane = new JScrollPane(textArea);
            scrollPane.setPreferredSize(new Dimension(600, 400));

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

        } catch (IOException ex) {
            JOptionPane.showMessageDialog(null, "Error reading file: " + ex.getMessage());
        }
    }
});

        frame.add(mainPanel);
        frame.setVisible(true);
    }

    public void buttonStyle(JButton button) {
```

```
        button.setFont(new Font("Arial", Font.BOLD, 14));
        button.setForeground(Color.WHITE);
        button.setBackground(new Color(0, 102, 204));
        button.setOpaque(true);
        button.setBorder(BorderFactory.createLineBorder(Color.WHITE, 2));
        button.setFocusPainted(false);
        button.setPreferredSize(new Dimension(160, 40));
        button.setCursor(new Cursor(Cursor.HAND_CURSOR));
    }

    public void saveToFile() {
    try {
        FileWriter writer = new FileWriter("MemberDetails.txt");

        for (GymMember member : members) {
            writer.write("----- Member Info -----\n");
            writer.write("ID: " + member.getId() + "\n");
            writer.write("Name: " + member.getName() + "\n");
            writer.write("Location: " + member.getLocation() + "\n");
            writer.write("Phone: " + member.getPhone() + "\n");
            writer.write("Email: " + member.getEmail() + "\n");
            writer.write("Membership Start Date: " + member.membershipStartDate + "\n");

            if (member instanceof RegularMember) {
                RegularMember rm = (RegularMember) member;
                writer.write("Plan: " + rm.getPlan() + "\n");
                writer.write("Price: " + (int) rm.getPrice() + "\n");
                writer.write("Attendance: " + rm.getAttendance() + "\n");
```

```
            writer.write("Loyalty Points: " + rm.getLoyaltyPoints() + "\n");
            writer.write("Active Status: " + rm.getActiveStatus() + "\n");
            writer.write("Full Payment: N/A\n");
            writer.write("Discount Amount: N/A\n");
            writer.write("Net Amount Paid: N/A\n");

        } else if (member instanceof PremiumMember) {
            PremiumMember pm = (PremiumMember) member;
            writer.write("Plan: Premium\n");
            writer.write("Price: 50000\n");
            writer.write("Attendance: " + pm.getAttendance() + "\n");
            writer.write("Loyalty Points: " + pm.getLoyaltyPoints() + "\n");
            writer.write("Active Status: " + pm.getActiveStatus() + "\n");
            writer.write("Full Payment: " + pm.getIsFullPayment() + "\n");
            writer.write("Discount Amount: " + pm.getDiscountAmount() + "\n");
            writer.write("Net Amount Paid: " + pm.getPaidAmount() + "\n");
        }

        writer.write("\n");
    }

    writer.close();
    JOptionPane.showMessageDialog(null, "Member details saved successfully to
MemberDetails.txt");

  } catch (IOException e) {
    JOptionPane.showMessageDialog(null, "Error saving to file: " + e.getMessage(),
"File Error", JOptionPane.ERROR_MESSAGE);
  }
```

```
}


    public static void main(String[] args) {
        new GymGUI();
    }
}
```