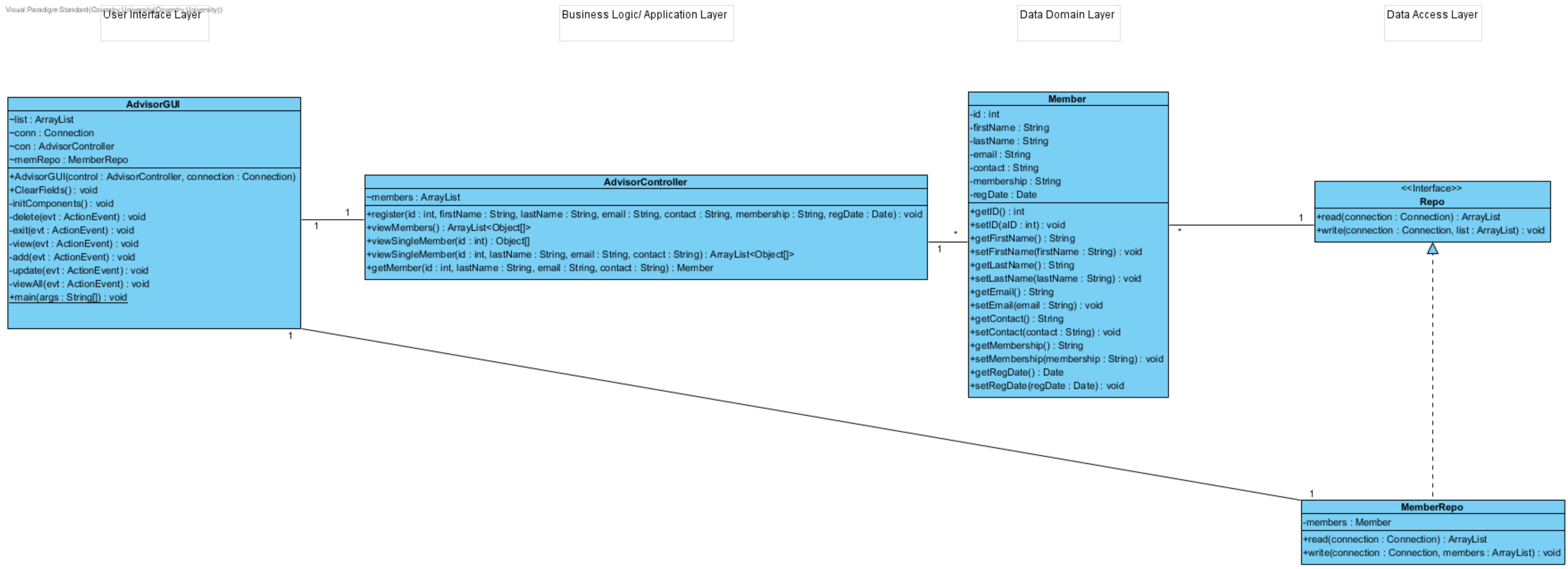
2019

260CT – Software Engineering

MEMBERSHIP MANAGEMENT SYSTEM

ROBERT GREEN - 7096325

Task 1 – UML Class Diagram


Visual Paradigm Standard(Coventry University(Coventry University))
User Interface Layer
Business Logic/ Application Layer
Data Domain Layer
Data Access Layer
AdvisorGUI
~list : ArrayList
~conn : Connection
~con : AdvisorController
~memRepo : MemberRepo
+AdvisorGUI(control : AdvisorController, connection : Connection)
+ClearFields() : void
-initComponents() : void
-delete(evt : ActionEvent) : void
-exit(evt : ActionEvent) : void
-view(evt : ActionEvent) : void
-add(evt : ActionEvent) : void
-update(evt : ActionEvent) : void
-viewAll(evt : ActionEvent) : void
+main(args : String[]) : void
AdvisorController
~members : ArrayList
+register(id : int, firstName : String, lastName : String, email : String, contact : String, membership : String, regDate : Date) : void
+viewMembers() : ArrayList<Object[]>
+viewSingleMember(id : int) : Object[]
+viewSingleMember(id : int, lastName : String, email : String, contact : String) : ArrayList<Object[]>
+getMember(id : int, lastName : String, email : String, contact : String) : Member
Member
-id : int
-firstName : String
-lastName : String
-email : String
-contact : String
-membership : String
-regDate : Date
+getID() : int
+setID(aID : int) : void
+getFirstName() : String
+setFirstName(firstName : String) : void
+getLastName() : String
+setLastName(lastName : String) : void
+getEmail() : String
+setEmail(email : String) : void
+getContact() : String
+setContact(contact : String) : void
+getMembership() : String
+setMembership(membership : String) : void
+getRegDate() : Date
+setRegDate(regDate : Date) : void
<<Interface>>
Repo
+read(connection : Connection) : ArrayList
+write(connection : Connection, list : ArrayList) : void
MemberRepo
-members : Member
+read(connection : Connection) : ArrayList
+write(connection : Connection, members : ArrayList) : void
1
1
*
1
*
1
1
1

Design - GRASP and GOF Patterns

When designing a software system, one can implement a design pattern in order streamline the development process. These patterns detail the best was to navigate common software design problems (Buschmann, F., Henney, K. and Schmidt, D.C., 2007). During the development of this project, I had to study multiple different design patterns to ensure that the code I created was reusable and reliable. I elected to implement two patterns, one from a set of patterns known as GRASP (Genera Responsibility Assignment Software Pattern), and another are the Gang of Four patterns, a collection of 23 design patterns organised into discrete groups: creational patterns, structural patterns and behavioural patterns (Gamma E., Helm R., Johnson R., Vlissides J., 1994).

There are seven different patterns descried in GRASP: Controller, Creator, High Cohesion, Information Expert, Protected Classes, Low Coupling, and Polymorphisism. I chose to use the Controller pattern, as it assigns the responsibility of handling business logic to a separate class. Specifically, I implemented a Role Controller, as the controller class is specific to the member of staff accessing the system. This allows for the user interface to be effectively separated from the other layers. However, the application I have designed is only intended for use by the advisor staff, so there exists only one variant of the controller class: AdvisorController. For this reason, it could also be considered a Façade Controller, which handles all of the business logic for a given system.

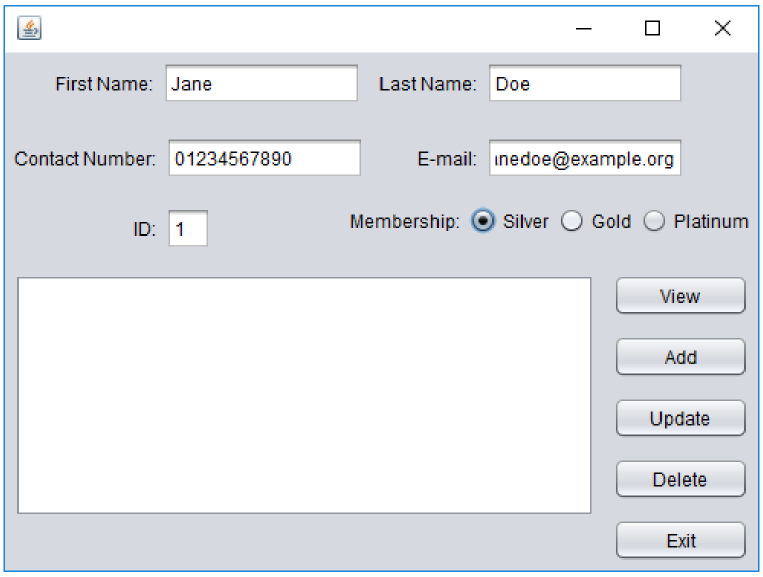
References

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (1994)
Design patterns : elements of reusable object-oriented software
Publisher: Addison Wesley (31 Oct 1994)

Task 2a – Prototype Graphical User Interface

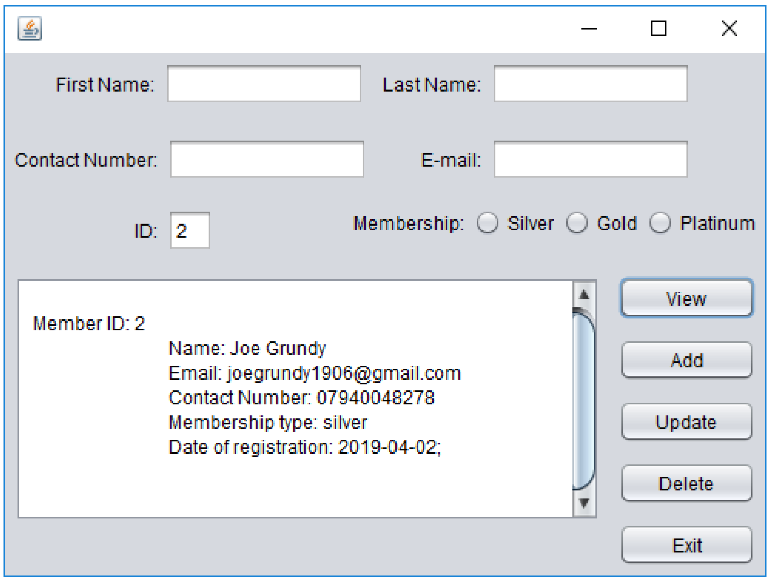
Shown below is the first prototype GUI I created for the membership management system. It is intended for use by the Sports and Leisure Village advisors to enable them to store and retrieve information about members registered to their club. The GUI affords the following functionalities:

- **Adding**
- **Viewing**
- **Deleting**
- **Updating**



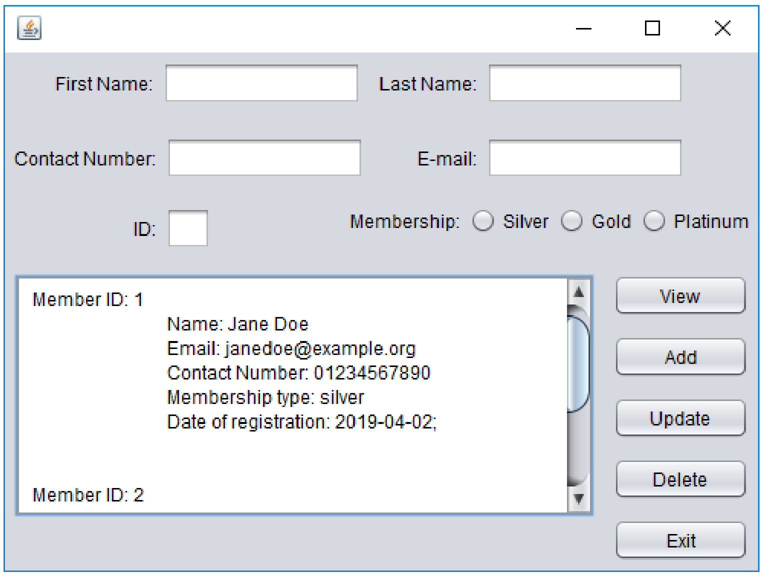
First Name: Jane
Last Name: Doe
Contact Number: 01234567890
E-mail: ınedoe@example.org
ID: 1
Membership: [✓ Silver] Gold Platinum

A new member can be added to the database by entering their details and clicking the “Add” button.



ID: 2

```
Member ID: 2
            Name: Joe Grundy
            Email: joegrundy1906@gmail.com
            Contact Number: 07940048278
            Membership type: silver
            Date of registration: 2019-04-02;
```



```
Member ID: 1
            Name: Jane Doe
            Email: janedoe@example.org
            Contact Number: 01234567890
            Membership type: silver
            Date of registration: 2019-04-02;


Member ID: 2
```

A member can be viewed by entering their Member ID and clicking the "View" button.

If no ID is specified, all members will be displayed instead.

A member can also be deleted by entering their ID and clicking "Delete".

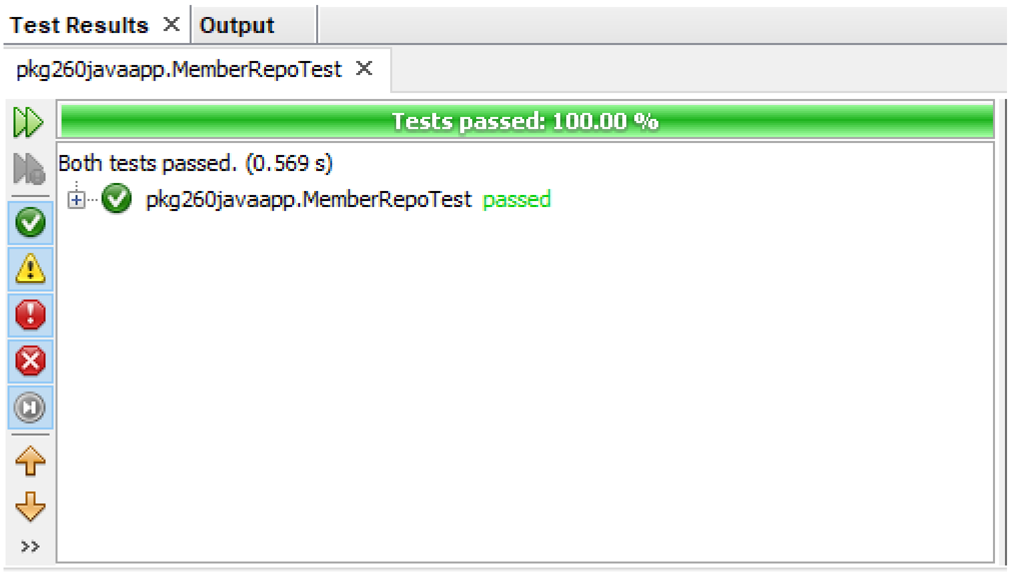
Client Feedback

- The client requested that the Members ID is automatically generated when they are first added to the system, rather than having to specify it.
- The client asked for some form of input validation to prevent erroneous data being entered
- The client would like to be able a locate a members information with alternative fields, not only ID
- The client asked that I re-design the layout of the interface, as it is not evident how to use it
- The client also asked that I use a table to display member information rather than a text area.

Task 2b – Unit Testing

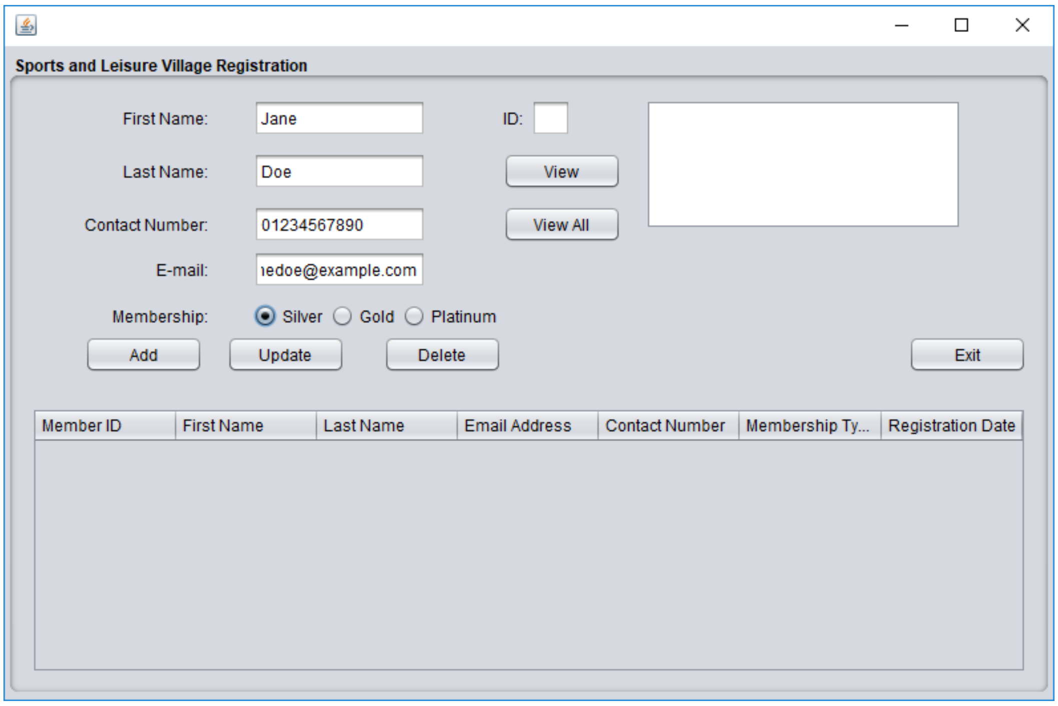
I used Junit4 to test the MemberRepo class of my application. This class is responsible for reading data from the database when the application is first opened, and writing new data to the database. I used dummy data to populate a table for the read() function, and dummy data is written to the database when testing the write() function.

```
package pkg260javaapp;

import java.sql.*;
import java.util.ArrayList;
import org.junit.Test;
import static org.junit.Assert.*;


public class MemberRepoTest {

    private ArrayList<Member> testMembers = new ArrayList();
    private String user = "test";
    private String pass = "test";
    private String url = "jdbc:derby://localhost:1527/MemberTest"; //credentials for Test
database
    private Connection connection;
    private java.util.Date jDate = new java.util.Date();
    private Date sDate = new java.sql.Date(jDate.getTime()); //for setting dummy members
regDate to todays date

    private void connect(){ //an internal method to connect to the Test database
        try{
            Connection conn = DriverManager.getConnection(url, user, pass);
            connection = conn;
        }
        catch (SQLException e){
            System.out.println("Could not connect");
            System.out.println(e);
        }
    }

    private void insertMember(){ //inserts a dummy value into the Test.Members table
        connect();
        try {
            PreparedStatement state = connection.prepareStatement(
                    "INSERT INTO TEST.MEMBERS VALUES (0, 'test', 'member',
'testmember@example.com', '04561237890', 'silver', ?)"
            );
            state.setDate(1, sDate);
            state.executeUpdate();
        }
        catch (SQLException e){
            System.out.println("Error inserting test member in test class");
            System.out.println(e);
        }
    }


    private void insertMember(){ //inserts a dummy value into the Test.Members table
        connect();
        try {
            PreparedStatement state = connection.prepareStatement(
                    "INSERT INTO TEST.MEMBERS VALUES (0, 'test', 'member',
'testmember@example.com', '04561237890', 'silver', ?)"
            );
            state.setDate(1, sDate);
        state.executeUpdate();
        }
        catch (SQLException e){
            System.out.println("Error inserting test member in test class");
            System.out.println(e);
        }
    }

    private void deleteMembers(){ //Deletes all members from Test table
        connect();
        try {
            Statement state = connection.createStatement();
            state.executeUpdate("DELETE FROM TEST.MEMBERS");
            state.close();
        }
        catch (SQLException e){
            System.out.println("Error delting members in test class");
            System.out.println(e);
        }
    }

    public void populateMembersList() { //Populates the testMembers ArrayList with
dummy data
        //dummy data
        Member dummyMember1 = new Member();
        dummyMember1.setID(1);
        dummyMember1.setFirstName("test1");
        dummyMember1.setLastName("member1");
        dummyMember1.setEmail("test1@test1.com");
        dummyMember1.setContact("01234567890");
        dummyMember1.setMembership("silver");
        java.util.Date jDate = new java.util.Date();
        java.sql.Date sDate = new java.sql.Date(jDate.getTime());
        dummyMember1.setRegDate(sDate);
        //dummy data
        Member dummyMember2 = new Member();
        dummyMember2.setID(2);
        dummyMember2.setFirstName("test2");
        dummyMember2.setLastName("member2");
        dummyMember2.setEmail("test2@test2.com");
        dummyMember2.setContact("09876543210");
        dummyMember2.setMembership("platinum");
        jDate = new java.util.Date();
        sDate = new java.sql.Date(jDate.getTime());
        dummyMember1.setRegDate(sDate);
        //dummy data
        testMembers.add(dummyMember1);
        testMembers.add(dummyMember2);


    }

    @Test
    public void testRead() {
        connect();                                          //connect to database
        insertMember();                                     //insert dummy data
        MemberRepo testRead = new MemberRepo();
        ArrayList result = testRead.read(connection);       //read from database
        assertFalse(result.isEmpty());                      //result should not be empty
        deleteMembers();                      //delete data from database for next test
    }
    @Test
    public void testWrite() {
        connect();
        populateMembersList();                              //populate testMembers
        System.out.println("write unit test");
        MemberRepo testWrite = new MemberRepo();
        try{
           testWrite.write(connection, testMembers);        //write testMembers to db
        }
        catch (Exception e){                                //catch errors
            fail("Could not write to database");
            System.out.println(e);
        }
    }

}
```



```
Test Results   Output
pkg260javaapp.MemberRepoTest
Tests passed: 100.00 %
Both tests passed. (0.569 s)
  pkg260javaapp.MemberRepoTest passed
```

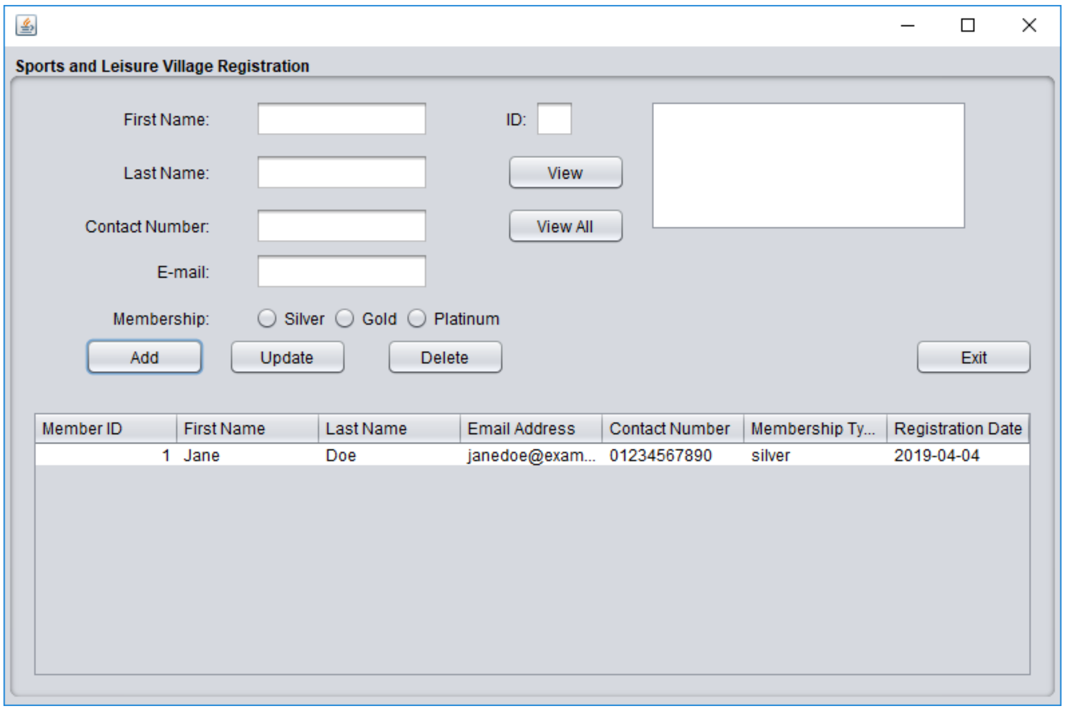
Task 2c – Final Prototype



Sports and Leisure Village Registration

First Name: Jane
Last Name: Doe
Contact Number: 01234567890
E-mail: ıedoe@example.com
Membership: [✓ Silver] Gold Platinum
ID:

Member ID	First Name	Last Name	Email Address	Contact Number	Membership Ty...	Registration Date

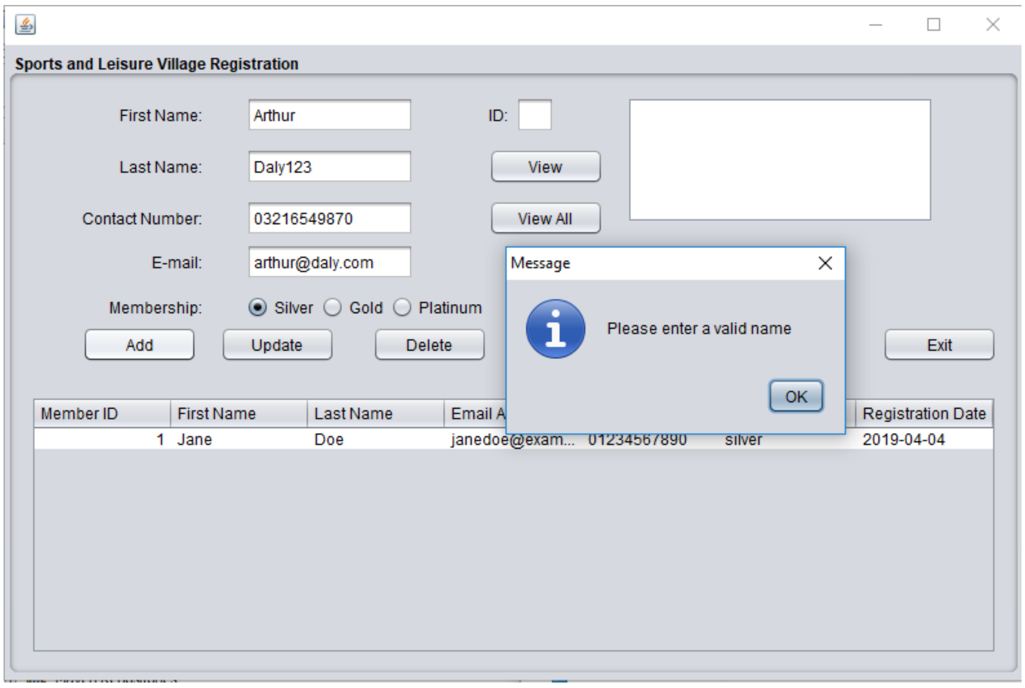


Sports and Leisure Village Registration

First Name:
Last Name:
Contact Number:
E-mail:
Membership: Silver Gold Platinum
ID:

Member ID	First Name	Last Name	Email Address	Contact Number	Membership Ty...	Registration Date
1	Jane	Doe	janedoe@exam...	01234567890	silver	2019-04-04

As requested by the client, the GUI now automatically generates an ID when a new member is added.


Sports and Leisure Village Registration

First Name: Arthur

Last Name: Daly123

Contact Number: 03216549870

E-mail: arthur@daly.com

Membership: [✓ Silver] Gold Platinum

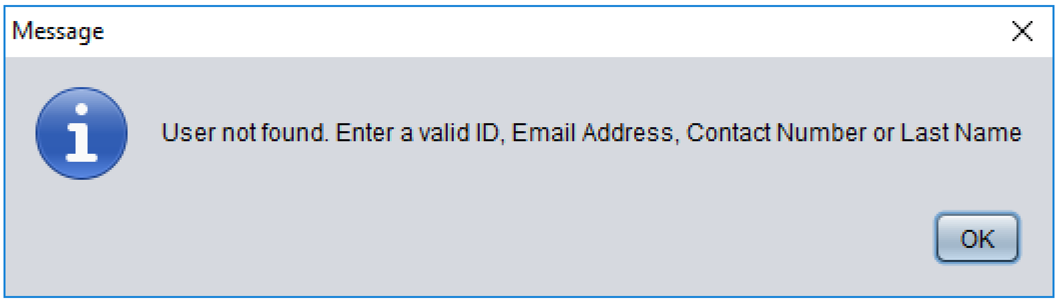
ID:

Message

Please enter a valid name

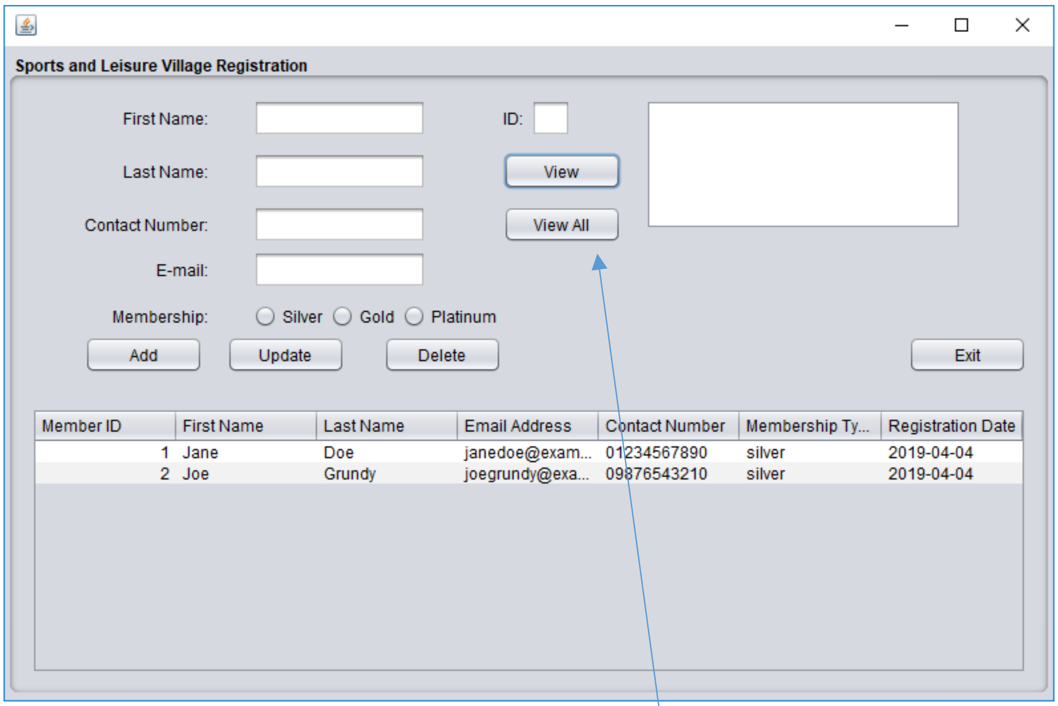
Member ID	First Name	Last Name	Email A…			Registration Date
1	Jane	Doe	janedoe@exam...	01234567890	silver	2019-04-04

There is now input validation, and the system tells the user which field contains invalid data should they attempt to add a new member.


Message

User not found. Enter a valid ID, Email Address, Contact Number or Last Name

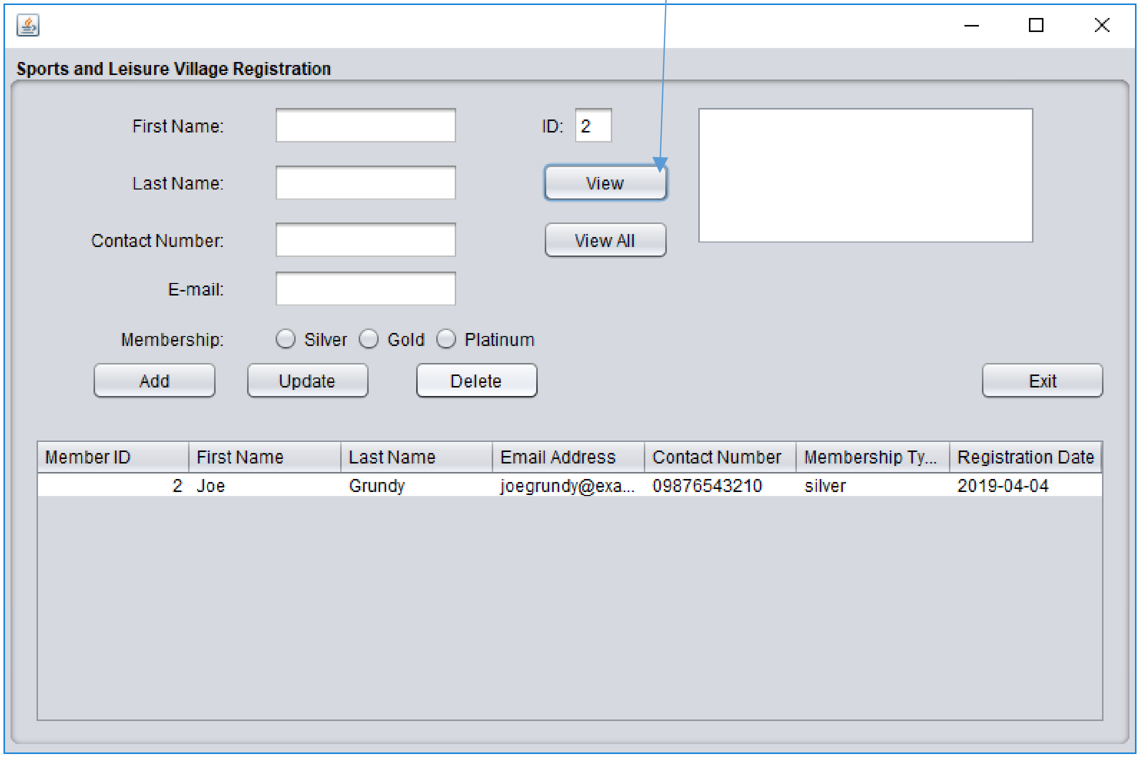
Users can now also search by ID, last name, contact number or email address


Sports and Leisure Village Registration

First Name:		ID:				
Last Name:		View				
Contact Number:		View All				
E-mail:						
Membership:	○ Silver ○ Gold ○ Platinum					

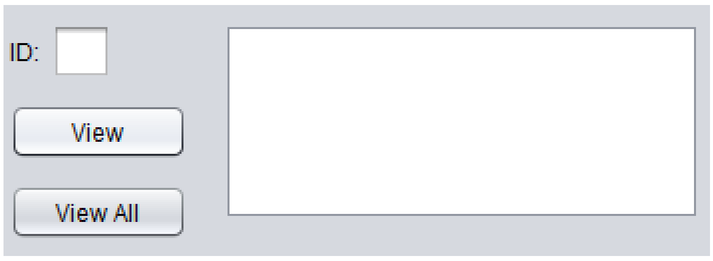
Member ID	First Name	Last Name	Email Address	Contact Number	Membership Ty...	Registration Date
1	Jane	Doe	janedoe@exam...	01234567890	silver	2019-04-04
2	Joe	Grundy	joegrundy@exa...	09876543210	silver	2019-04-04

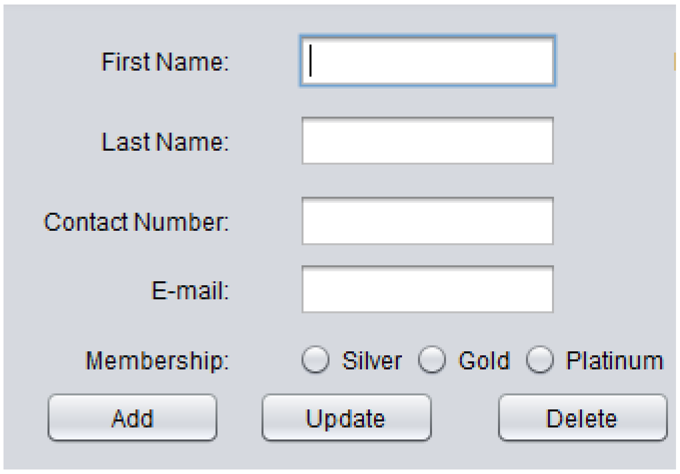
I have added a separate button for viewing every record, as to indicate that the user can view a single member by clicking the “View” button


Sports and Leisure Village Registration

First Name:		ID: 2				
Last Name:		View				
Contact Number:		View All				
E-mail:						
Membership:	○ Silver ○ Gold ○ Platinum					

Member ID	First Name	Last Name	Email Address	Contact Number	Membership Ty...	Registration Date
2	Joe	Grundy	joegrundy@exa...	09876543210	silver	2019-04-04





I have redesigned the UI, as per the clients request, so that buttons and fields are grouped better according to which buttons use which fields.