2018

260CT Software Engineering

7080380
ADAM GILMOUR

Table of Contents

Task 1 Adam Gilmour 7080380

Design Model

The use case that was chosen for this assignment was the membership management. The use case diagram below shows what the actions of the user are for the member management system.

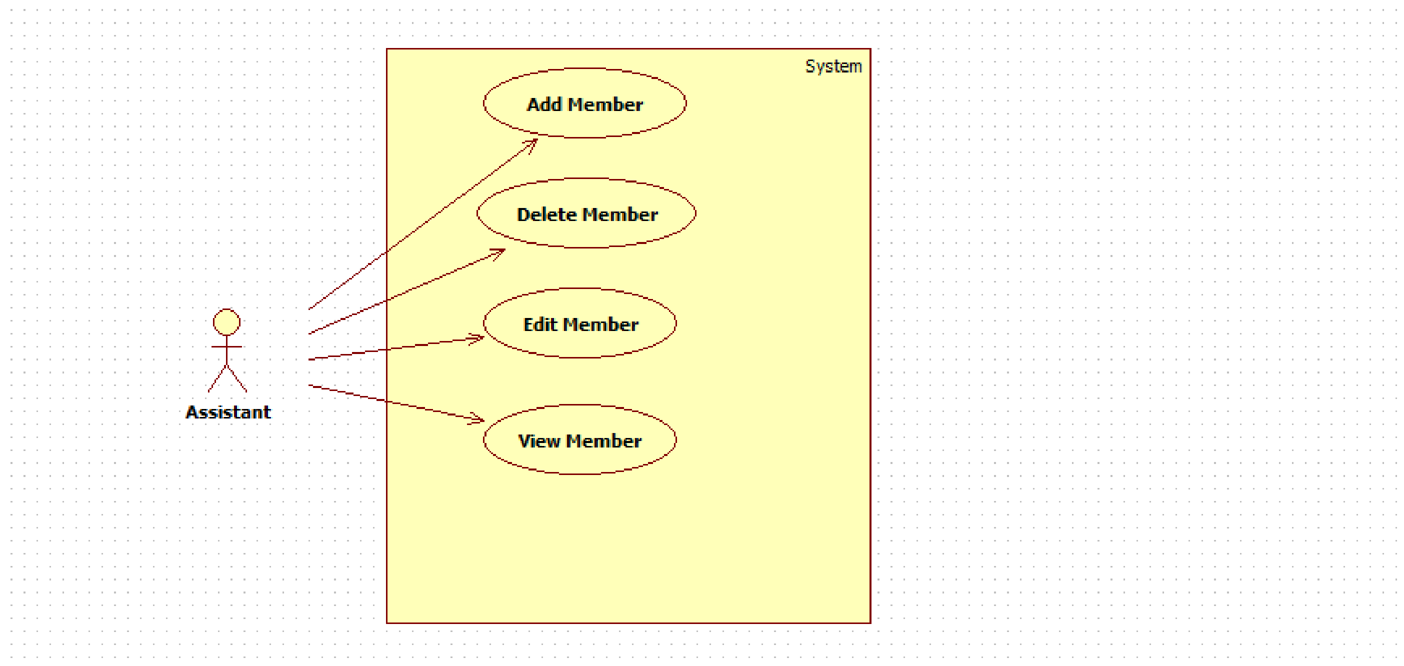


Figure 1: Use Case Diagram

The use case above shows that the Assistant (the user) can perform four different actions, that is Add Member, Delete Member, Edit Member and View Member.

From this use case diagram, you can create a class diagram. For this software a four-layered architectural approach was used, this is where the software has four different layers of classes, the layers that are used for this system was the User Interface Layer, which is what handles the output and input to external actors, the application and business logic layer which coordinates the application logic for each class, the domain data layer and data persistence layer.

UML

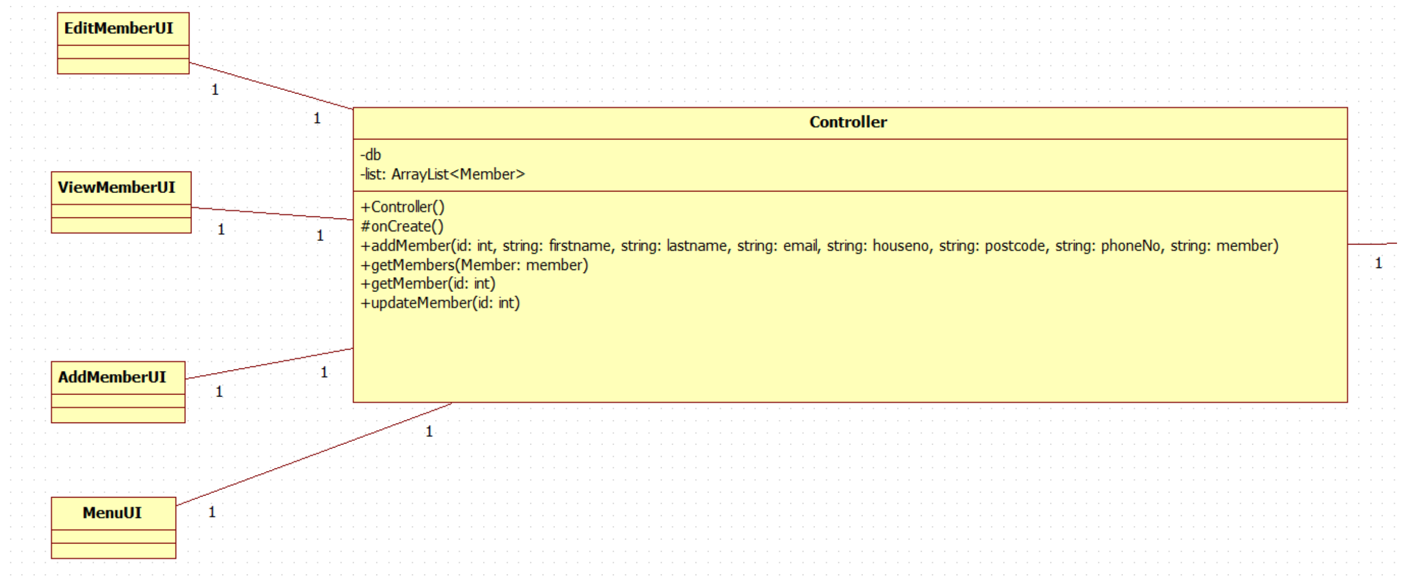

EditMemberUI
ViewMemberUI
AddMemberUI
MenuUI
Controller
-db
-list: ArrayList<Member>
+Controller()
#onCreate()
+addMember(id: int, string: firstname, string: lastname, string: email, string: houseno, string: postcode, string: phoneNo, string: member)
+getMembers(Member: member)
+getMember(id: int)
+updateMember(id: int)
1
1
1
1
1
1
1
1
1

Figure 2: Interface Layer and Application/Business Logic Layer

The picture above shows the first two layers, the user interface layer and the application /business logic layer. The picture above also shows the user of the Façade approach, which is a type of GRASP method. The design of this program has four different user interfaces because it is easier to have a different user interface for a different job, all the user interfaces are linked together, meaning you can go to one from to the other without needing to go back to the menu. The controller is there to provide control of the four different user interfaces, the user interfaces only allows for inputs and outputs, it is the controller that really does the logic of the program. Originally there was meant to be a role controller. However, I only have 1 actor, the Assistant, who has access to the four user Interfaces, if I was to have another actor, say the manager, and another class for that manager it would have been a role controller. Instead a façade controller was used; a façade controller is a class that represents the system and controls everything in that system. The controller can add a member, delete a member, get a member and edit a member. Façade controller has high coupling and low cohesion (Hedley 2017), however it does make a software library easier to use since the façade has convenient methods for common tasks (*Facade Pattern* 2018). In my UML you can see the methods for the common tasks add member, edit member, delete member and get member, it is this that makes façade controllers more desirable.

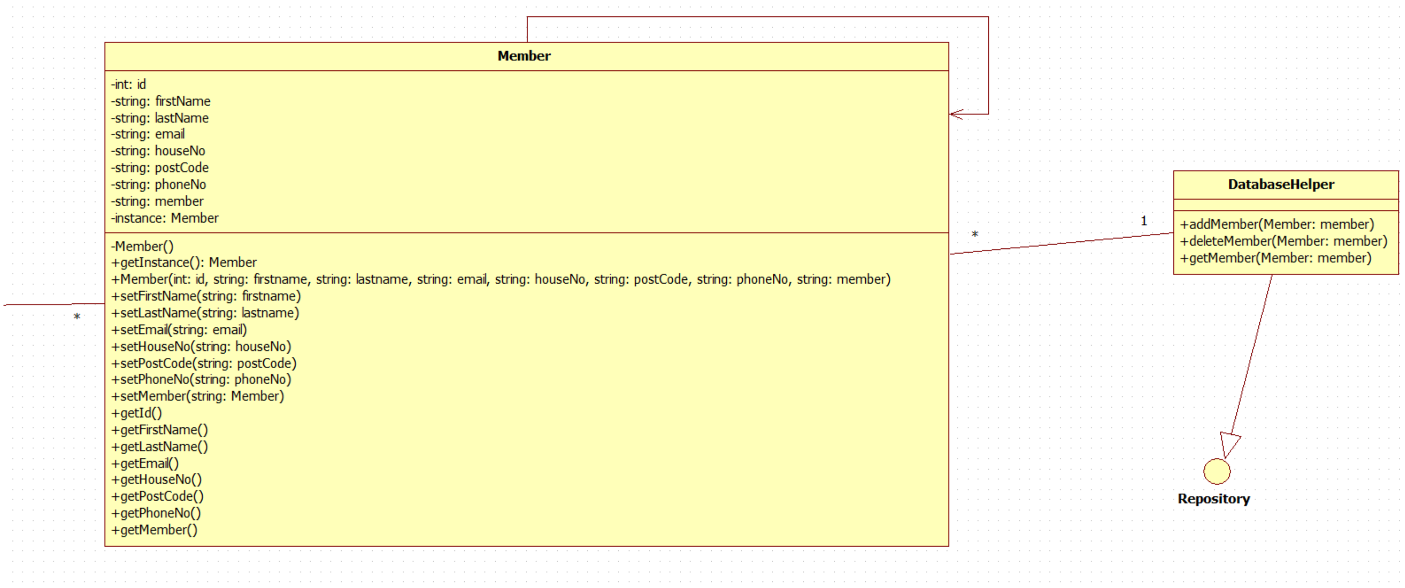

Member
-int: id
-string: firstName
-string: lastName
-string: email
-string: houseNo
-string: postCode
-string: phoneNo
-string: member
-instance: Member
-Member()
+getInstance(): Member
+Member(int: id, string: firstname, string: lastname, string: email, string: houseNo, string: postCode, string: phoneNo, string: member)
+setFirstName(string: firstname)
+setLastName(string: lastname)
+setEmail(string: email)
+setHouseNo(string: houseNo)
+setPostCode(string: postCode)
+setPhoneNo(string: phoneNo)
+setMember(string: Member)
+getId()
+getFirstName()
+getLastName()
+getEmail()
+getHouseNo()
+getPostCode()
+getPhoneNo()
+getMember()
*
*
1
DatabaseHelper
+addMember(Member: member)
+deleteMember(Member: member)
+getMember(Member: member)
Repository

Figure 3: Domain Data Layer and Data Persistence Layer

The Picture above shows the domain data layer, member, which implements the singleton gang of four, and the data persistence layer. The gang of four has 23 different design patterns that provide solutions to a variety of problems face during software development. (Hedley 2018) There are three different types of patterns for GoF; the pattern type used here was from creational patterns, which is the management of creating objects. There are several different types of creational patterns, Abstract Factory, Builder, Factory Method, Prototype and Singleton, each with their own benefits.

This software could have used the factory method type, which is used to create objects, but allows the subclasses of that class to decide what class should be instantiated, where the subclasses implement the interfaces. It is useful when the software needs to create different types of the same object. For example, the member class above could have been split in to paid members and non-paid members, something like the picture below.

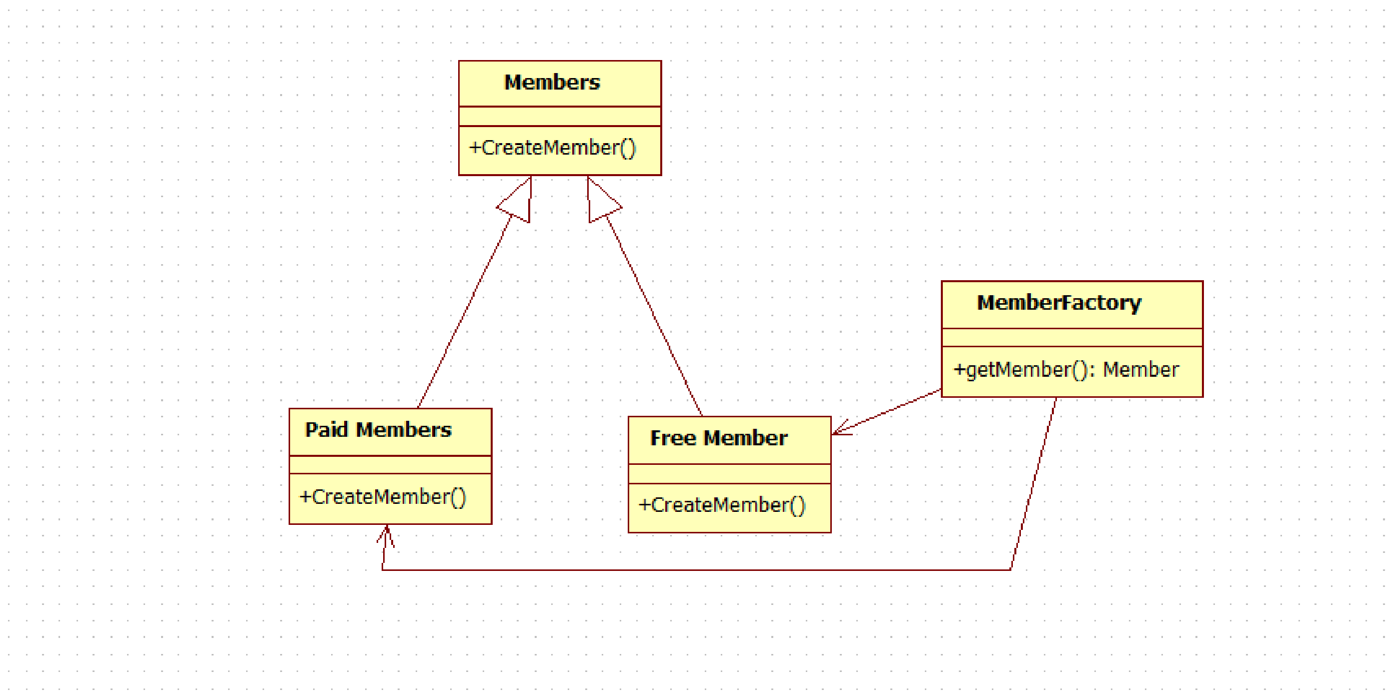


Figure 4: Simplified example of Factory Method

The reason why this Gang of Four pattern was not implemented was because the Member class had already been deeply coded into the program, to implement factory method would have required a massive restructure of the code, and perhaps the database.

The pattern that was chosen for this system instead was the singleton pattern. (Hedley 2018) The single pattern is a pattern that enables the class to have the responsibility to create its own object, this ensures that only a single object gets created at any given time. The single class provides the access to its only object directly, without the need to instantiate the object of the class, there are many reasons why this approach is useful. The first reason is because you only let one object to coordinate itself across the system, it allows the system to run a lot more quickly, this is actually shown in the unit testing, Another reason why it is useful is for system with a low amount of member, such as an android application, mobile devices have a limited amount of RAM, if you was a method that called members from a database an put them into an List in Java then that List would use more memory with more members, in a system with over a thousand members you would end up using a lot of memory. The way this document is designed you can only access one member at a time because of the singleton pattern.

The way singleton is implemented in a piece of software is to define a private static attribute in the class, then within that class you define a public static accessor method in the class and you define all the constructors in the class to private or protected, this way only the accessor method can manipulate the singleton.

Task 2 Adam Gilmour 7080380

User Interface Layer: Add Member

Design

The code used for the database and the member class are referenced from Android SQLite Database Tutorial where I learned how to create a database and a proper class to use the database. The reference itself can be found in References.

The user interface is just a standard interface that accepts user input and displays an output. For the add member user interface, their several different fields that a person can enter text into, each filed has its own input validation, both in the code and in the xml file, which is how Android Studio designs its applications. The first picture will be what the prototype looks like, the following pictures will be the second prototype listing the improvements and going through each stage describing what is happening along with the pieces of code that does various things.

The picture to the side is the first prototype, showing how a new member is added, a person must enter their details, and what type of member they are, when you click Add Member the program will add the member to the database and the display a toast message saying that the member has been added, however it does not give their ID. As part of the improvement of the application I had to add a new text view field that displays a member id when the person is added.

In the second prototype I changed the design a bit and added a TextView box, when the assistant fills in the user details it will now add the member to the database, automatically generate an id and then display that id in the Member ID bit. As part of the validation for the program a method has been created where if any of the text fields are empty, the program will not submit the data to the database, all fields must but filled. For extra validation, android allows you to directly manipulate the XML file that creates the objects you see on screen, in XML you can set the limits of what can be entered.

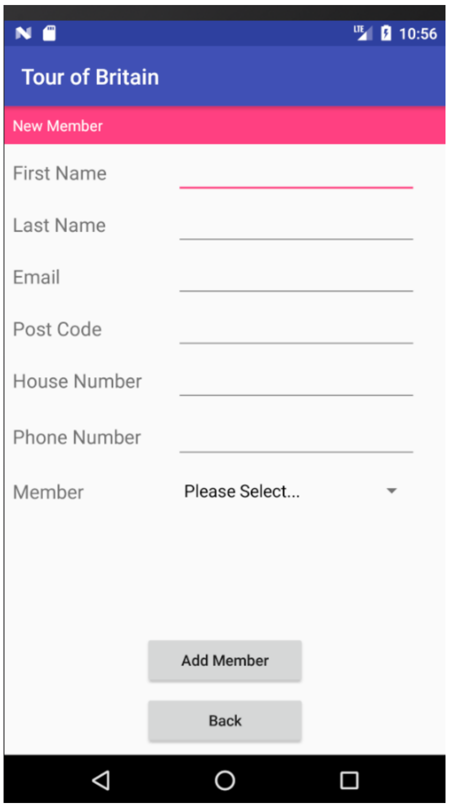


Figure 6: Prototype New Member

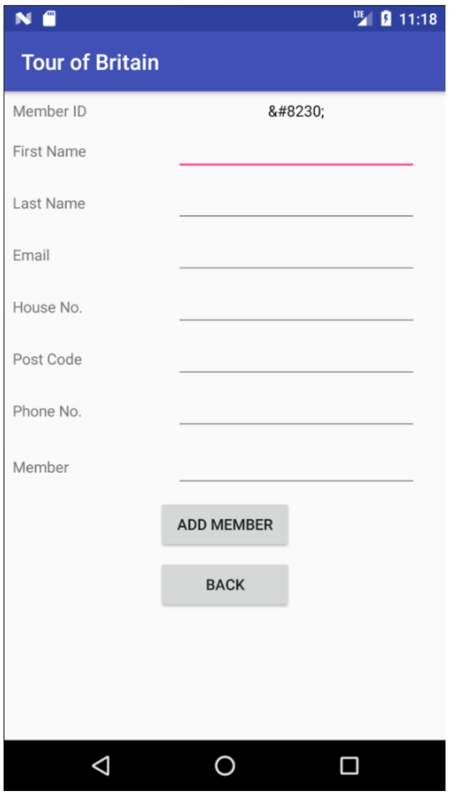
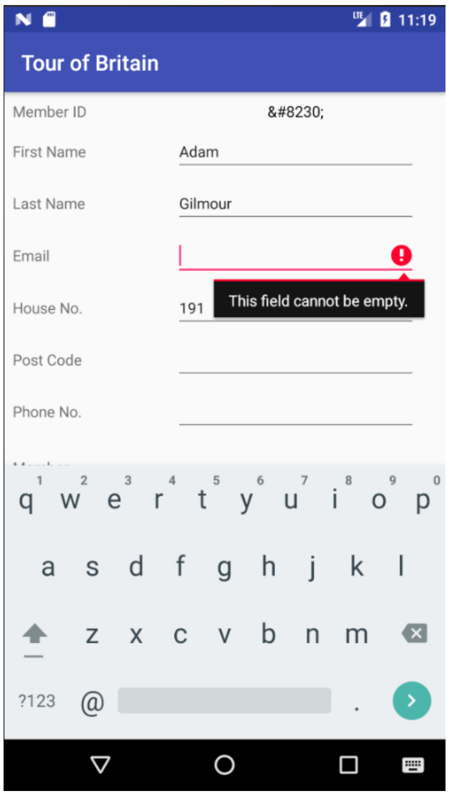
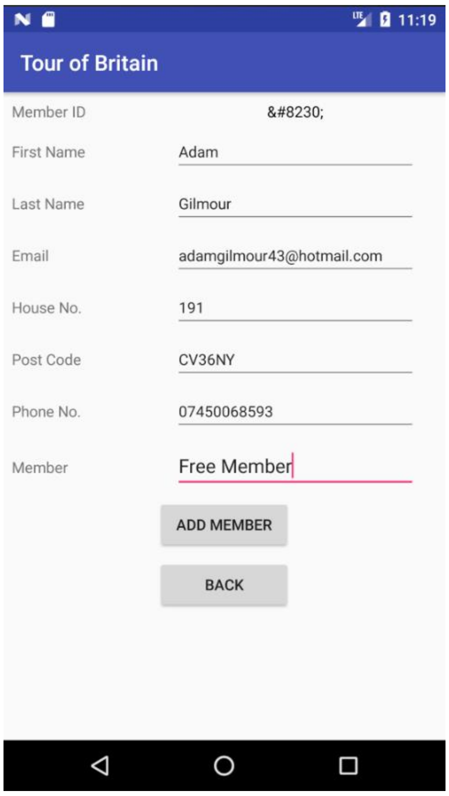


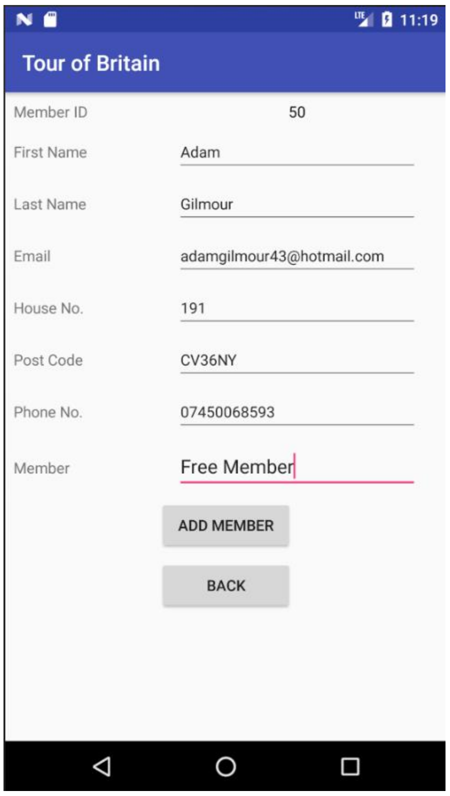
Figure 5: New Member



This field cannot be empty.


Tour of Britain

Member ID	&#8230;
First Name	Adam
Last Name	Gilmour
Email	adamgilmour43@hotmail.com
House No.	191
Post Code	CV36NY
Phone No.	07450068593
Member	Free Member


Tour of Britain

Member ID	50
First Name	Adam
Last Name	Gilmour
Email	adamgilmour43@hotmail.com
House No.	191
Post Code	CV36NY
Phone No.	07450068593
Member	Free Member

Figure 7: Member ID

Code

Java

```
package com.example.adamg.myapplication;

import android.content.Intent;
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v7.app.AppCompatActivity;
import android.telephony.PhoneNumberUtils;
import android.text.InputFilter;
import android.text.Spanned;
import android.text.TextUtils;
import android.util.Patterns;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

/**
 * Created by adamg on 27/03/2018.
 */

public class AddMember extends AppCompatActivity{

    //Database helper so that we can access the database
    DatabaseHelper db = new DatabaseHelper(this);

    EditText ETfirstName, ETlastName, ETEmail, ETHouseNo, ETPostCode, ETPhoneNo, ETMember;
    String strFirstName, strLastName, strEmail, strPostCode, strPhoneNo, strHouseNo, strMember, co
unt;
    TextView Did, TestFirst, TestSecond, TestEmail, TestHouse, TestPost, TestPhone, TestMember;

    Member member;
    Spinner spinner;

    @Override
```

```java
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.add_member);

        member = Member.getInstance();

        //Initiate the EditText fields
        //These are the inputs
        ETfirstName = (EditText)findViewById(R.id.ETFirstName);
        ETlastName = (EditText)findViewById(R.id.ETLastName);
        ETEmail = (EditText)findViewById(R.id.ETEmail);
        ETHouseNo = (EditText)findViewById(R.id.ETHouseNo);
        ETPostCode = (EditText)findViewById(R.id.ETPostCode);
        ETPhoneNo = (EditText)findViewById(R.id.ETPhoneNo);
        ETMember = (EditText)findViewById(R.id.ETMember);

        //Set text field
        Did = (TextView)findViewById(R.id.DID);
    }

    //One Button To add the member
    public void onButtonClicked(View view){
        if(view.getId() == R.id.BAddMember) {

            //Validation for all text fields
            strFirstName = ETfirstName.getText().toString();
            if (TextUtils.isEmpty(strFirstName)){
                ETfirstName.setError("This field cannot be empty.");
                return;
            }

            strLastName = ETlastName.getText().toString();
            if (TextUtils.isEmpty(strLastName)){
                ETlastName.setError("This field cannot be empty.");
                return;
            }

            strEmail = ETEmail.getText().toString();
            if (TextUtils.isEmpty(strEmail)){
                ETEmail.setError("This field cannot be empty.");
                return;
            }

            strHouseNo = ETHouseNo.getText().toString();
            if (TextUtils.isEmpty(strHouseNo)){
                ETHouseNo.setError("This field cannot be empty.");
                return;
            }

            strPostCode = ETPostCode.getText().toString();
            if (TextUtils.isEmpty(strPostCode)){
                ETPostCode.setError("This field cannot be empty.");
                return;
            }

            strPhoneNo = PhoneNumberUtils.formatNumber(ETPhoneNo.getText().toString());
            if (TextUtils.isEmpty(strPhoneNo)){
                ETPhoneNo.setError("This field cannot be empty.");
                return;
            }

            strMember = ETMember.getText().toString();
            if (TextUtils.isEmpty(strMember)){
                ETMember.setError("This field cannot be empty.");
                        return;
                    }

                    //Because these are activities we can only pass values from one class to another by using intent.
                    Intent intent = new Intent(this, Controller.class);
                    intent.putExtra("Select", 1);
                    intent.putExtra("First Name", strFirstName);
                    intent.putExtra("Last Name", strLastName);
                    intent.putExtra("Email", strEmail);
                    intent.putExtra("House No",strHouseNo);
                    intent.putExtra("Post Code", strPostCode);
                    intent.putExtra("Phone Number", strPhoneNo);
                    intent.putExtra("Member", strMember);
                    startActivity(intent);

                    //gets member id from database
                    int id = db.getID();
                    count = (Integer.toString(db.getID()));
                    member = db.getMember(id);

                    //Displays member id when you click add
                    Did.setText(count);
                }
            }
        }
```

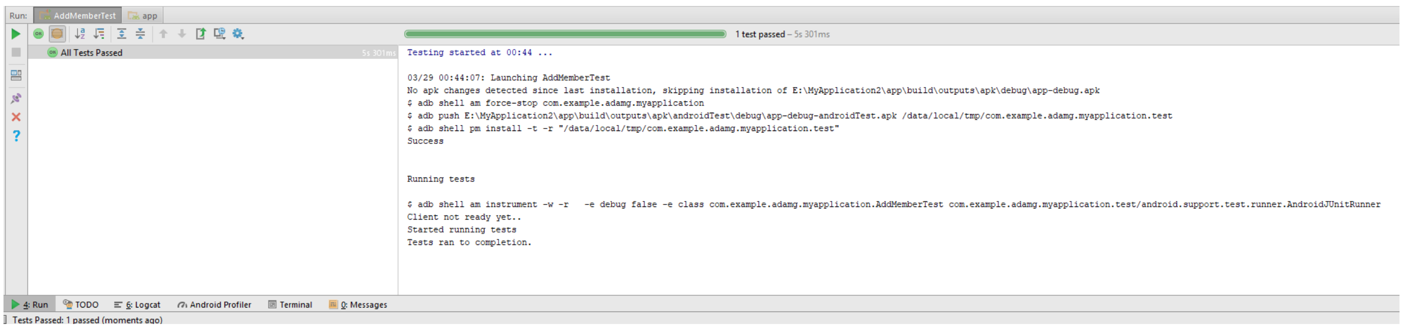
XML File

```
<EditText
    android:digits="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
    android:id="@+id/ETFirstName"
    android:layout_width="225sp"
    android:layout_height="wrap_content"
    android:layout_marginEnd="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:ems="10"
    android:inputType="textPersonName"
    android:textSize="14sp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@+id/TVFirstname"
    app:layout_constraintTop_toBottomOf="@+id/DID" />
```

The android digits is where you only allow certain letters or numbers to be entered to the text field.

Automated Testing

The automated testing is to be run each time a make a change to this specific part of the code, the idea here is to make sure that when something is added to the text fields and submitted it is returned back by using methods from the database, it checks to see if the data has been added.

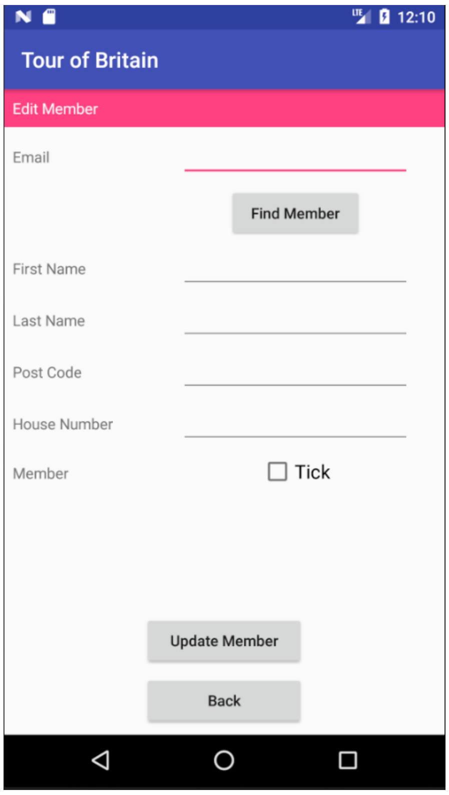


```
Testing started at 00:44 ...

03/29 00:44:07: Launching AddMemberTest
No apk changes detected since last installation, skipping installation of E:\MyApplication2\app\build\outputs\apk\debug\app-debug.apk
$ adb shell am force-stop com.example.adamg.myapplication
$ adb push E:\MyApplication2\app\build\outputs\apk\androidTest\debug\app-debug-androidTest.apk /data/local/tmp/com.example.adamg.myapplication.test
$ adb shell pm install -t -r "/data/local/tmp/com.example.adamg.myapplication.test"
Success


Running tests

$ adb shell am instrument -w -r   -e debug false -e class com.example.adamg.myapplication.AddMemberTest com.example.adamg.myapplication.test/android.support.test.runner.AndroidJUnitRunner
Client not ready yet..
Started running tests
Tests ran to completion.
```

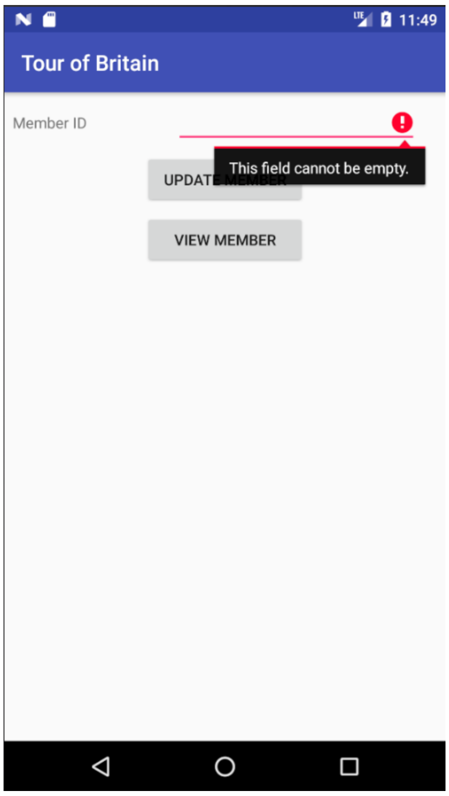
Tests Passed: 1 passed (moments ago)

User Interface View and Edit Member

View member is slightly different as you must first enter the member id, then you can select to view the member or edit, there are validations in place to make sure that you cannot select a member that does not exist. The pictures below will show the errors that occur when trying to find a member that does not exist, viewing a member, editing and member and then deleting a member and then showing that the member was deleted. The original prototype got the members from a list, but this was before I added the Singleton Pattern, when I added that I quickly found out that you can only have one member object at a time, but the members are still stored in the database, this made me redesign this part of the program, the original feedback wanted me to change all attributes of a member as well, but I needed to add a Gang of Four pattern. As a result, that is why the programs look different, the second prototype is just a massive improvement, it was the hardest part of the program to get right. The first image is the prototype.

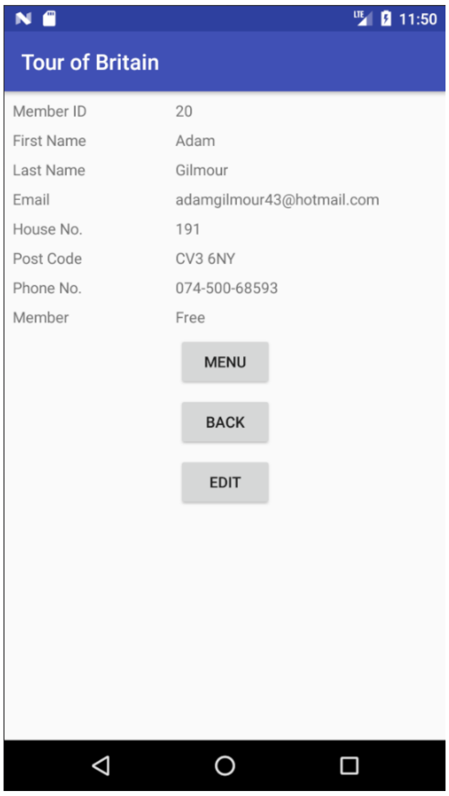


Email	
First Name	
Last Name	
Post Code	
House Number	
Member	☐ Tick

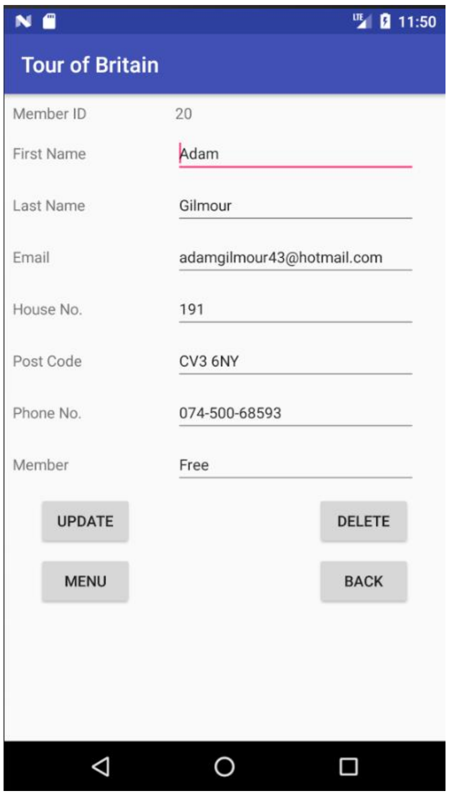


Member ID

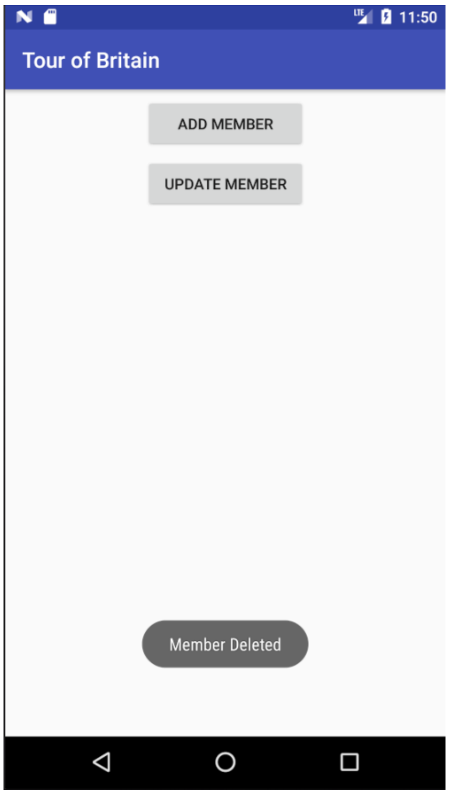
This field cannot be empty.



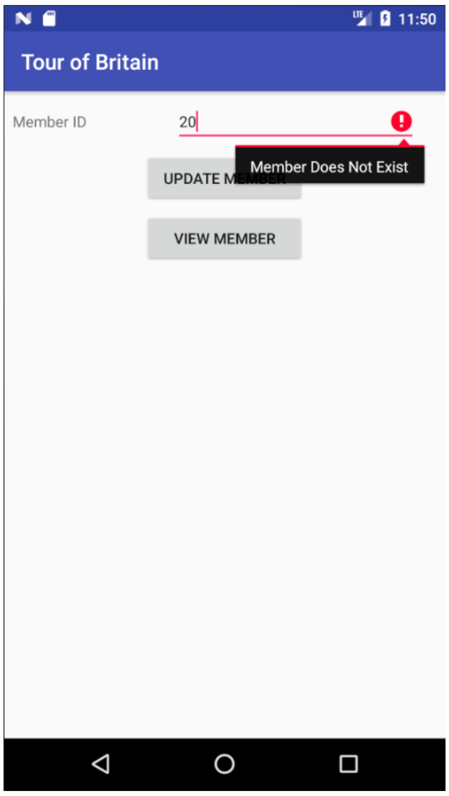
Member ID	20
First Name	Adam
Last Name	Gilmour
Email	adamgilmour43@hotmail.com
House No.	191
Post Code	CV3 6NY
Phone No.	074-500-68593
Member	Free



Member ID	20
First Name	Adam
Last Name	Gilmour
Email	adamgilmour43@hotmail.com
House No.	191
Post Code	CV3 6NY
Phone No.	074-500-68593
Member	Free



Member Deleted



Member ID 20

Member Does Not Exist

Application/Business Logic Layer

The business a logic layer was not examined during the first prototype, but as a result, the changes made from the feedback to user interfaces has made me change the logical layer. The logical layer is the part of the program that handles the methods that makes the program do stuff. Each method inside this layer will be explained here, please note that for this program work, the control must be an Activity to use the database.

In terms of the four layered design, in order for inputs to passed to the controller, I had to use intents (*Intent | Android Developers* 2018), each time the controller was called, certain values were passed to the controller, along with a select option, the select option being what type of thing the User Interface wanted to do when a button was clicked.

```java
protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Intent i = getIntent();
    Select = i.getIntExtra("Select", 0);
    count = db.memberCount();
    if (Select == 1) {
        addMembers();
    }
    else if(Select == 2){
        id = i.getIntExtra("ID", 0);
        viewMember(id);
    }
    else if(Select == 3){
        id = i.getIntExtra("ID", 0);
        getMember(id);
    }
    else  if (Select == 4){
        updateMember();
    }
    else if (Select == 5){
        deleteMember();
    }
}
```

The façade controller means that this class controls everything in this system, as shown by the following methods, you can tell that this controller adds members to a database, gets members from a database, updates members and deletes them. The methods below shows how this is done. Due to the way android works I had to use intent to get the values from the user interfaces, instead of calling the methods from the user interfaces.

Add Member Method

```java
public void addMembers(){

    Intent i = getIntent();

    firstName = i.getStringExtra("First Name");
    lastName = i.getStringExtra("Last Name");
    email = i.getStringExtra("Email");
    houseNo = i.getStringExtra("House No");
    postCode = i.getStringExtra("Post Code");
    phoneNo = i.getStringExtra("Phone Number");
    member = i.getStringExtra("Member");

    Member uMember = Member.getInstance();

    uMember.setFirstName(firstName);
    uMember.setLastName(lastName);
    uMember.setEmail(email);
    uMember.setHouseNo(houseNo);
    uMember.setPostCode(postCode);
    uMember.setPhoneNo(phoneNo);
    uMember.setMember(member);

    db.insertMember(uMember);
    finish();
}
```

Get Member Method

```java
public void getMember(int id){
    Member member = Member.getInstance();
    member = db.getMember(id);

    Intent intent = new Intent(this, FoundMember.class);
    intent.putExtra("Member", member);
    finish();
    startActivity(intent);

}
```

The button in update member uses this function to get the member from the database and then returns it to a new activity, found member, the entire process of getting a member and passing it to a new activity is done in this single function.

Data Domain Layer

This layer implements the singleton pattern. As explained above the singleton pattern is where the class has control of its own objects, as a result only one member can exist at any given time, saving resources and protecting the other objects, you cannot access another member while another one is being used.

```java
public class Member implements Serializable{

    private static Member instance = new Member();
```

This part of the program is what allows the class to provide a static attribute, called instance, to take the object via its constructor.

```java
private Member(){
}

public static Member getInstance(){
    return instance;
}

public Member(int id, String firstName, String lastName, String email, String houseNo, String postCode, String phoneNo, String member){
    this.id = id;
    this.firstName = firstName;
    this.lastName = lastName;
    this.email = email;
    this.houseNo = houseNo;
    this.postCode = postCode;
    this.phoneNo = phoneNo;
    this.member = member;
}
```

The private member is the constructor, the public Member is what allows you to set the attributes of the object, the getInstance is what allows other classes to get a Member object from this class.

When a person wants to look for a member it must first enter the member id, the program will then find the member in the database, the database will create an object by calling the instanced, from here it will add the attributes to the object and then return the object back to appropriate interface, below is the code that is used when a member is being looked for, it is using all four layers of the program, and shows how the singleton pattern works.

```java
else if(view.getId() == R.id.BUpdate){
    Intent i = new Intent(this, UpdateMember.class);
    startActivity(i);
}
```

First the person clicks on the button in the Main Activity.

The user must then enter the member id and providing it is the correct one, will go on to the next stage.

```java
if (v.getId() == R.id.BViewMember){
           strID = UMid.getText().toString();
           if (TextUtils.isEmpty(strID)){
               UMid.setError("This field cannot be empty.");
               return;
           }
           id = Integer.parseInt(strID);

           int count = db.memberCount() + 1;
           if (id > (count)){
               UMid.setError("Member Does Not Exist");
               return;
           }
           boolean checkifExists = db.exist(id);
           if (!checkifExists){
               UMid.setError("Member Does Not Exist");
               return;
           }
           else {
               Intent i = new Intent(this, Controller.class);
               i.putExtra("Select", 2);
               i.putExtra("ID", id);
               finish();
               startActivity(i);
           }
       }
```

The Boolean is there for input validation, it will not look for a member if it does not exist in the database. The function for seeing if a member exists is below:

```java
public boolean exist(int id){
    SQLiteDatabase db = this.getReadableDatabase();
    String query = "Select * FROM " + Member.TABLE_NAME + " WHERE " + Member.COLUMN_ID + " = " + i
d;
    Cursor cursor = db.rawQuery(query, null);
    if (cursor.getCount() <= 0){
        return false;
    }
    cursor.close();
    return true;
}
```

Once the button is clicked and the member exists it will then move to the controller, which is what controls the logic of the program, here you can get members, add members, edit members and delete members.

```java
public void viewMember(int id){
    Member member = Member.getInstance();
    member = db.getMember(id);

    Intent intent = new Intent(this, ViewMember.class);
    intent.putExtra("Member", member);
    finish();
    startActivity(intent);
}
```

In this method the controller will create a new instance of member first, this is so that we can store the data we are getting from the database in an object, after creating the member instance it will get the data from the table, using the code below and open a new activity to view the member.

```java
public Member getMember(long id){
     SQLiteDatabase db = this.getReadableDatabase();

     Cursor cursor = db.query(Member.TABLE_NAME,
             new String[]{Member.COLUMN_ID, Member.COLUMN_FIRST_NAME, Member.COLUMN_LAST_NAME, Member.COLUMN_EMAIL, Member.COLUMN_HOUSE_NO, Member.COLUMN_POST_CODE, Member.COLUMN_PHONE_NO, Member.COLUMN_MEMBER},
             Member.COLUMN_ID + "=?",
             new String[]{String.valueOf(id)},
             null, null, null, null);

     if(cursor!=null){
         cursor.moveToNext();
     }

     Member member = new Member(
             cursor.getInt(cursor.getColumnIndex(Member.COLUMN_ID)),
             cursor.getString(cursor.getColumnIndex(Member.COLUMN_FIRST_NAME)),
             cursor.getString(cursor.getColumnIndex(Member.COLUMN_LAST_NAME)),
             cursor.getString(cursor.getColumnIndex(Member.COLUMN_EMAIL)),
             cursor.getString(cursor.getColumnIndex(Member.COLUMN_HOUSE_NO)),
             cursor.getString(cursor.getColumnIndex(Member.COLUMN_POST_CODE)),
             cursor.getString(cursor.getColumnIndex(Member.COLUMN_PHONE_NO)),
             cursor.getString(cursor.getColumnIndex(Member.COLUMN_MEMBER)));
     cursor.close();

     return member;
 }
```

The reason why the code above works is because it is only creating one object of class Member, if it was trying to create multiple objects all it would do is only make one, and it would only be the last one as this was already tried.

After the member attributes have been fetched it is then passed all the way back to the activity that is to display them. From here the activity gets the attributes from the object and displays them in their own TextFields for the user to see.

```java
Member member = Member.getInstance();
```

This code above is needed to show that instance of member is created for this class, this allows you to get the attributes from the members and allows the class to receive the incoming object from Controller.

```java
@Override
protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.view_members);
    Intent i = getIntent();
    member = (Member)i.getSerializableExtra("Member");

    //Initialise Text Views
    DMemberID = (TextView)findViewById(R.id.DisMemberID);
    DFirstName = (TextView)findViewById(R.id.DisFirstName);
    DLastName = (TextView)findViewById(R.id.DLastName);
    DEmail = (TextView)findViewById(R.id.DEmail);
    DHouseNo = (TextView)findViewById(R.id.DHouseNo);
    DPostCode = (TextView)findViewById(R.id.DPostCode);
    DPhoneNo = (TextView)findViewById(R.id.DPhoneNo);
    DMember = (TextView)findViewById(R.id.DMember);

    //Get values
    intId = member.getId();
    strMemberID = Integer.toString(intId);
    strFirstName = member.getFirstName();
    strLastName = member.getLastName();
    strEmail = member.getEmail();
    strHouse = member.getHouseNo();
    strPostCode = member.getPostCode();
    strPhoneNo = member.getPhoneNo();
    strMember = member.getMember();

    //Set values
    DMemberID.setText(strMemberID);
    DFirstName.setText(strFirstName);
    DLastName.setText(strLastName);
    DEmail.setText(strEmail);
    DHouseNo.setText(strHouse);
    DPostCode.setText(strPostCode);
    DPhoneNo.setText(strPhoneNo);
    DMember.setText(strMember);
}
```

Data Access Layer

The data access layer is where all the database queries happens, it is what allows the rest of the program to interact with the database.

This part of system just shows that the data access layer exists, database creation and some database querying will be shown here.

```
public static final String CREATE_TABLE = "CREATE TABLE " + TABLE_NAME + "( "
        + COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
        + COLUMN_FIRST_NAME + " VARCHAR(25), "
        + COLUMN_LAST_NAME + " VARCHAR(25),"
        + COLUMN_EMAIL + " VARCHAR(225),"
        + COLUMN_HOUSE_NO + " VARCHAR(4),"
        + COLUMN_POST_CODE + " VARCHAR(12),"
        + COLUMN_PHONE_NO + " VARCHAR(15),"
        + COLUMN_MEMBER + " VARCHAR(25)"
        + ")";
```

```
private static final int DATABASE_VER = 1;

private static final String DB_NAME = "TOB.db";

public DatabaseHelper(Context context){
    super(context, DB_NAME, null, DATABASE_VER);
}

@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL(Member.CREATE_TABLE);
}

public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){
    db.execSQL("DROP TABLE IF EXISTS " + Member.TABLE_NAME);

    onCreate(db);
}
```

The code above is how the system creates a table, and below shows how data is inserted.

```
public long insertMember(Member member){
    SQLiteDatabase db = this.getWritableDatabase();

    int count = 0;
    count = memberCount();

    ContentValues values = new ContentValues();
    values.put(Member.COLUMN_FIRST_NAME, member.getFirstName());
    values.put(Member.COLUMN_LAST_NAME, member.getLastName());
    values.put(Member.COLUMN_EMAIL, member.getEmail());
    values.put(Member.COLUMN_HOUSE_NO, member.getHouseNo());
    values.put(Member.COLUMN_POST_CODE, member.getPostCode());
    values.put(Member.COLUMN_PHONE_NO, member.getPhoneNo());
    values.put(Member.COLUMN_MEMBER, member.getMember());

    long id = db.insert(Member.TABLE_NAME, null, values);
    db.close();
    return id;}
```

Program Appendix

Main Activity

```java
package com.example.adamg.myapplication;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

    }

    //Methods for button clicks
    public void onButtonClicked(View view){
        //if add is clicked, open addMember xml
        if(view.getId() == R.id.BAdd) {
            Intent i = new Intent(this, AddMember.class);
            startActivity(i);
        }
        //if update is clicked open updateMember xml
        else if(view.getId() == R.id.BUpdate){
            Intent i = new Intent(this, UpdateMember.class);
            startActivity(i);
        }
    }
}
```

Add Member

```java
package com.example.adamg.myapplication;

import android.content.Intent;
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v7.app.AppCompatActivity;
import android.telephony.PhoneNumberUtils;
import android.text.InputFilter;
import android.text.Spanned;
import android.text.TextUtils;
import android.util.Patterns;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

/**
 * Created by adamg on 27/03/2018.
 */

public class AddMember extends AppCompatActivity{

    //Database helper so that we can access the database
    DatabaseHelper db = new DatabaseHelper(this);

    EditText ETfirstName, ETlastName, ETEmail, ETHouseNo, ETPostCode, ETPhoneNo, ETMember;
    String strFirstName, strLastName, strEmail, strPostCode, strPhoneNo, strHouseNo, strMember, co
unt;
    TextView Did, TestFirst, TestSecond, TestEmail, TestHouse, TestPost, TestPhone, TestMember;
    Member member;
    Spinner spinner;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.add_member);

        member = Member.getInstance();

        //Initiate the EditText fields
        //These are the inputs
        ETfirstName = (EditText)findViewById(R.id.ETFirstName);
        ETlastName = (EditText)findViewById(R.id.ETLastName);
        ETEmail = (EditText)findViewById(R.id.ETEmail);
        ETHouseNo = (EditText)findViewById(R.id.ETHouseNo);
        ETPostCode = (EditText)findViewById(R.id.ETPostCode);
        ETPhoneNo = (EditText)findViewById(R.id.ETPhoneNo);
        ETMember = (EditText)findViewById(R.id.ETMember);

        //Set text field
        Did = (TextView)findViewById(R.id.DID);
    }

    //One Button To add the member
    public void onButtonClicked(View view){
        if(view.getId() == R.id.BAddMember) {

            //Validation for all text fields
            strFirstName = ETfirstName.getText().toString();
            if (TextUtils.isEmpty(strFirstName)){
                ETfirstName.setError("This field cannot be empty.");
                return;
            }

            strLastName = ETlastName.getText().toString();
            if (TextUtils.isEmpty(strLastName)){
                ETlastName.setError("This field cannot be empty.");
                return;
            }

            strEmail = ETEmail.getText().toString();
            if (TextUtils.isEmpty(strEmail)){
                ETEmail.setError("This field cannot be empty.");
                return;
            }

            strHouseNo = ETHouseNo.getText().toString();
            if (TextUtils.isEmpty(strHouseNo)){
                ETHouseNo.setError("This field cannot be empty.");
                return;
            }

            strPostCode = ETPostCode.getText().toString();
            if (TextUtils.isEmpty(strPostCode)){
                ETPostCode.setError("This field cannot be empty.");
                return;
            }

            strPhoneNo = PhoneNumberUtils.formatNumber(ETPhoneNo.getText().toString());
            if (TextUtils.isEmpty(strPhoneNo)){
                ETPhoneNo.setError("This field cannot be empty.");
                return;
            }

            strMember = ETMember.getText().toString();
            if (TextUtils.isEmpty(strMember)){
                ETMember.setError("This field cannot be empty.");
```

```java
                        return;
                    }

                    //Because these are activities we can only pass values from one class to another by using intent.
                    Intent intent = new Intent(this, Controller.class);
                    intent.putExtra("Select", 1);
                    intent.putExtra("First Name", strFirstName);
                    intent.putExtra("Last Name", strLastName);
                    intent.putExtra("Email", strEmail);
                    intent.putExtra("House No",strHouseNo);
                    intent.putExtra("Post Code", strPostCode);
                    intent.putExtra("Phone Number", strPhoneNo);
                    intent.putExtra("Member", strMember);
                    startActivity(intent);

                    //gets member id from database
                    int id = db.getID();
                    count = (Integer.toString(db.getID()));
                    member = db.getMember(id);

                    //Displays member id when you click add
                    Did.setText(count);
                }
            }
        }
```

Update Member

```java
package com.example.adamg.myapplication;

import android.content.Intent;
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v7.app.AppCompatActivity;
import android.text.TextUtils;
import android.view.View;
import android.widget.EditText;

import java.io.Serializable;

public class UpdateMember extends AppCompatActivity implements Serializable {

    EditText UMid;
    String strID;
    int id;

    DatabaseHelper db = new DatabaseHelper(this);

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.update_member);

        UMid = (EditText)findViewById(R.id.UMid);

    }

    public void onButtonClicked(View v){
        if (v.getId() == R.id.BFind){
            strID = UMid.getText().toString();
            //Checks if box is empty, if empty return to this point
            //Show error in EditText
            if (TextUtils.isEmpty(strID)){
                UMid.setError("This field cannot be empty.");
                return;
            }
            id = Integer.parseInt(strID);

            //Chceks if member exists
            boolean checkifExists = db.exist(id);
            if (!checkifExists){
                UMid.setError("Member Does Not Exist");
                return;
            }
            else {
                //if information valid run this activity for getMember
                Intent i = new Intent(this, Controller.class);
                i.putExtra("Select", 3);
                i.putExtra("ID", id);
                finish();
                startActivity(i);
            }
        }
        if (v.getId() == R.id.BViewMember){
            strID = UMid.getText().toString();
            //Checks if box is empty, if empty return to this point
            //Show error in EditText
            if (TextUtils.isEmpty(strID)){
                UMid.setError("This field cannot be empty.");
                return;
            }
            id = Integer.parseInt(strID);

            //Chceks if member exists
            boolean checkifExists = db.exist(id);
            if (!checkifExists){
                UMid.setError("Member Does Not Exist");
                return;
            }
            else {
                //if input is valid run this activity to update the member
                Intent i = new Intent(this, Controller.class);
                i.putExtra("Select", 2);
                i.putExtra("ID", id);
                finish();
                startActivity(i);
            }
        }
    }
}
```

View Member

```java
package com.example.adamg.myapplication;

import android.content.Intent;
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.TextView;

import java.io.Serializable;
import java.util.List;

/*
Class to get member details and display them in view_members.xml.
 */

public class ViewMember extends AppCompatActivity implements Serializable{

    Member member = Member.getInstance();
    TextView DFirstName, DLastName, DEmail, DHouseNo, DPostCode, DPhoneNo, DMember, DMemberID;
    String strFirstName, strLastName, strEmail, strHouse, strPostCode, strPhoneNo, strMember, strMemberID;
    int intId;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.view_members);
        Intent i = getIntent();
        member = (Member)i.getSerializableExtra("Member");

        //Initialise Text Views
        DMemberID = (TextView)findViewById(R.id.DisMemberID);
        DFirstName = (TextView)findViewById(R.id.DisFirstName);
        DLastName = (TextView)findViewById(R.id.DLastName);
        DEmail = (TextView)findViewById(R.id.DEmail);
        DHouseNo = (TextView)findViewById(R.id.DHouseNo);
        DPostCode = (TextView)findViewById(R.id.DPostCode);
        DPhoneNo = (TextView)findViewById(R.id.DPhoneNo);
        DMember = (TextView)findViewById(R.id.DMember);

        //Get values
        intId = member.getId();
        strMemberID = Integer.toString(intId);
        strFirstName = member.getFirstName();
        strLastName = member.getLastName();
        strEmail = member.getEmail();
        strHouse = member.getHouseNo();
        strPostCode = member.getPostCode();
        strPhoneNo = member.getPhoneNo();
        strMember = member.getMember();

        //Set values
        DMemberID.setText(strMemberID);
        DFirstName.setText(strFirstName);
        DLastName.setText(strLastName);
        DEmail.setText(strEmail);
        DHouseNo.setText(strHouse);
        DPostCode.setText(strPostCode);
        DPhoneNo.setText(strPhoneNo);
        DMember.setText(strMember);
    }

    public void onButtonClicked(final View view){
        if (view.getId() == R.id.BEdit){
            //When edit button is clicked it uses the id already got
            //and uses it to open Edit Member
            Intent i = new Intent(this, Controller.class);
            i.putExtra("ID", intId);
            i.putExtra("Select", 3);
            finish();
            startActivity(i);
        }
        if (view.getId() == R.id.BBack){
            //goes back
            Intent i = new Intent(this, UpdateMember.class);
            finish();
            startActivity(i);
        }
        else if (view.getId() == R.id.BMenu){
            //goes to menu
            Intent intent = new Intent(this, MainActivity.class);
            finish();
            startActivity(intent);
        }

    }
}
```

Found Member

```java
package com.example.adamg.myapplication;

import android.content.Intent;
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

import org.w3c.dom.Text;

import java.io.Serializable;

/**
 * Created by adamg on 27/03/2018.
 */

public class FoundMember extends AppCompatActivity implements Serializable{

    Member member = Member.getInstance();
    String strMemberId, strFirstName, strLastName, strEmail, strHouseNo, strPostCode, strPhoneNo, 
strMember;
    EditText DFirstName, DLastName, DEmail, DHouseNo, DPostcode, DPhoneNo, DMember;
    TextView DMemberID;
    int id;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.found_member);
        Intent i = getIntent();
        member = (Member)i.getSerializableExtra("Member");

        DMemberID = (TextView)findViewById(R.id.DisMemberID);

        DFirstName = (EditText)findViewById(R.id.DisFirstName);
        DLastName = (EditText)findViewById(R.id.DisLastName);
        DEmail = (EditText)findViewById(R.id.DisEmail);
        DHouseNo = (EditText)findViewById(R.id.DisHouseNo);
        DPostcode = (EditText)findViewById(R.id.DisPostCode);
        DPhoneNo = (EditText)findViewById(R.id.DisPhoneNo);
        DMember = (EditText)findViewById(R.id.DisMember);

        id = member.getId();
        strMemberId = Integer.toString(id);
        strFirstName = member.getFirstName();
        strLastName = member.getLastName();
        strEmail = member.getEmail();
        strHouseNo = member.getHouseNo();
        strPostCode = member.getPostCode();
        strPhoneNo = member.getPhoneNo();
        strMember = member.getMember();

        DMemberID.setText(strMemberId);
        DFirstName.setText(strFirstName);
        DLastName.setText(strLastName);
        DEmail.setText(strEmail);
        DHouseNo.setText(strHouseNo);
        DPostcode.setText(strPostCode);
        DPhoneNo.setText(strPhoneNo);
        DMember.setText(strMember);

    }

    public void onButtonClicked(View v){
        if (v.getId() == R.id.BUpdateMember){
            //Get new string values from text boxes
            strFirstName = DFirstName.getText().toString();
            strLastName = DLastName.getText().toString();
            strEmail = DEmail.getText().toString();
            strHouseNo = DHouseNo.getText().toString();
            strPostCode = DPostcode.getText().toString();
            strPhoneNo = DPhoneNo.getText().toString();
            strMember = DMember.getText().toString();

            //Before opening the controller create the intent
            //put string values in new intent
            //start activity.
            Intent intent = new Intent(this, Controller.class);
            intent.putExtra("FFirst", strFirstName);
            intent.putExtra("FLast", strLastName);
            intent.putExtra("FEmail", strEmail);
            intent.putExtra("FHouse", strHouseNo);
            intent.putExtra("FPost", strPostCode);
            intent.putExtra("FPhone", strPhoneNo);
            intent.putExtra("FMember", strMember);
            intent.putExtra("Select", 4);
            startActivity(intent);
        }
        else if (v.getId() == R.id.BDelete){
            //Gets id from string, created on Activity creation
            //make object with that id
            //send object to controller
            //finish when done
            Intent intent = new Intent(this, Controller.class);
            member.setId(id);
            intent.putExtra("Delete Member", member);
            intent.putExtra("Select", 5);
            finish();
                startActivity(intent);
            }
            else if (v.getId() == R.id.BMenu){
                Intent intent = new Intent(this, MainActivity.class);
                finish();
                startActivity(intent);
            }
            else if (v.getId() == R.id.BBack){
                Intent i = new Intent(this, UpdateMember.class);
                finish();
                startActivity(i);
            }
        }
    }
```

Controller

```java
package com.example.adamg.myapplication;

import android.content.Intent;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.os.Parcel;
import android.provider.ContactsContract;
import android.support.annotation.Nullable;
import android.support.v7.app.AppCompatActivity;
import android.widget.Toast;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class Controller extends AppCompatActivity implements Serializable{

    private DatabaseHelper db = new DatabaseHelper(this);
    private static List<String> memberList = new ArrayList<>();

    public static String firstName, lastName, email, houseNo, postCode, phoneNo, member;
    public static int Select, count, id;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Intent i = getIntent();
        //Gets value from intents when activity is called
        Select = i.getIntExtra("Select", 0);
        count = db.memberCount();
        if (Select == 1) {
            addMembers();
        }
        else if(Select == 2){
            id = i.getIntExtra("ID", 0);
            viewMember(id);
        }
        else if(Select == 3){
            id = i.getIntExtra("ID", 0);
            getMember(id);
        }
        else  if (Select == 4){
            updateMember();
        }
        else if (Select == 5){
            deleteMember();
        }
    }


    /*Adds a member to the database using object*/
    public void addMembers(){

        Intent i = getIntent();

        firstName = i.getStringExtra("First Name");
        lastName = i.getStringExtra("Last Name");
        email = i.getStringExtra("Email");
        houseNo = i.getStringExtra("House No");
        postCode = i.getStringExtra("Post Code");
        phoneNo = i.getStringExtra("Phone Number");
        member = i.getStringExtra("Member");

        //Get member instance
        Member uMember = Member.getInstance();

        //Set attributes
        uMember.setFirstName(firstName);
        uMember.setLastName(lastName);
        uMember.setEmail(email);
        uMember.setHouseNo(houseNo);
        uMember.setPostCode(postCode);
        uMember.setPhoneNo(phoneNo);
        uMember.setMember(member);

        //Insert Member
        db.insertMember(uMember);

        //Close activity when done
        finish();
    }


    /*Gets members from database*/
    public void getMember(int id){
        //Create new instance of member
        Member member = Member.getInstance();

        //Database creates member and passes it to function
        member = db.getMember(id);

        //Create new intent for new activity
        Intent intent = new Intent(this, FoundMember.class);
        intent.putExtra("Member", member);

        //close this activity
        finish();

        //start new activity
               startActivity(intent);

           }

           /*Does the same as above but starts different activity*/
           public void viewMember(int id){
               Member member = Member.getInstance();
               member = db.getMember(id);

               Intent intent = new Intent(this, ViewMember.class);
               intent.putExtra("Member", member);
               finish();
               startActivity(intent);
           }


           /*Find member in database and then updates it*/
           public void updateMember(){
               Intent i = getIntent();
               firstName = i.getStringExtra("FFirst");
               lastName = i.getStringExtra("FLast");
               email = i.getStringExtra("FEmail");
               houseNo = i.getStringExtra("FHouse");
               postCode = i.getStringExtra("FPost");
               phoneNo = i.getStringExtra("FPhone");
               member = i.getStringExtra("FMember");

               Member uMember = Member.getInstance();

               uMember.setId(id);
               uMember.setFirstName(firstName);
               uMember.setLastName(lastName);
               uMember.setEmail(email);
               uMember.setHouseNo(houseNo);
               uMember.setPostCode(postCode);
               uMember.setPhoneNo(phoneNo);
               uMember.setMember(member);


               db.updateMember(uMember);
               firstName = Integer.toString((uMember).getId());
               Toast pass = Toast.makeText(Controller.this, "Updated " + firstName, Toast.LENGTH_SHORT);
               pass.show();
               finish();
           }

           /*Deletes Member*/
           public void deleteMember(){
               Intent i = getIntent();

               Member uMember = Member.getInstance();

               uMember = (Member)i.getSerializableExtra("Delete Member");
               db.deleteMember(uMember);
               Toast pass = Toast.makeText(Controller.this, "Member Deleted", Toast.LENGTH_SHORT);

               pass.show();
               finish();
           }
       }
```

Member

```java
package com.example.adamg.myapplication;
```

```java

import java.io.Serializable;

/**
 * Created by adamg on 26/03/2018.
 */

public class Member implements Serializable{

    private static Member instance = new Member();

    public static final String TABLE_NAME = "Members";

    public static final String COLUMN_ID = "id";
    public static final String COLUMN_FIRST_NAME = "firstName";
    public static final String COLUMN_LAST_NAME = "lastName";
    public static final String COLUMN_EMAIL = "email";
    public static final String COLUMN_HOUSE_NO = "houseNo";
    public static final String COLUMN_POST_CODE = "postCode";
    public static final String COLUMN_PHONE_NO = "phoneNo";
    public static final String COLUMN_MEMBER = "member";

    private int id;
    private String firstName, lastName, email, houseNo, postCode, phoneNo, member;

    public static final String CREATE_TABLE = "CREATE TABLE " + TABLE_NAME + "( "
            + COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
            + COLUMN_FIRST_NAME + " VARCHAR(25), "
            + COLUMN_LAST_NAME + " VARCHAR(25),"
            + COLUMN_EMAIL + " VARCHAR(225),"
            + COLUMN_HOUSE_NO + " VARCHAR(4),"
            + COLUMN_POST_CODE + " VARCHAR(12),"
            + COLUMN_PHONE_NO + " VARCHAR(15),"
            + COLUMN_MEMBER + " VARCHAR(25)"
            + ")";

    private Member(){
    }

    public static Member getInstance(){
        return instance;
    }

    public Member(int id, String firstName, String lastName, String email, String houseNo, String
postCode, String phoneNo, String member){
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.houseNo = houseNo;
        this.postCode = postCode;
        this.phoneNo = phoneNo;
        this.member = member;
    }

    public void setId(int id) {
        this.id = id;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public void setHouseNo(String houseNo) {
        this.houseNo = houseNo;
    }

    public void setPostCode(String postCode) {
        this.postCode = postCode;
    }

    public void setPhoneNo(String phoneNo) {
        this.phoneNo = phoneNo;
    }

    public void setMember(String member) {
        this.member = member;
    }

    public int getId() {
        return id;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

          public String getEmail() {
              return email;
          }

          public String getHouseNo() {
              return houseNo;
          }

          public String getPostCode() {
              return postCode;
          }

          public String getPhoneNo() {
              return phoneNo;
          }

          public String getMember() {
              return member;
          }
      }
```

Database Handler

```java
package com.example.adamg.myapplication;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.support.v4.view.ViewPropertyAnimatorListener;

import java.util.ArrayList;
import java.util.List;

/**
 * Created by adamg on 26/03/2018.
 */

public class DatabaseHelper extends SQLiteOpenHelper{

    private static final int DATABASE_VER = 1;

    private static final String DB_NAME = "TOB.db";

    public DatabaseHelper(Context context){
        super(context, DB_NAME, null, DATABASE_VER);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(Member.CREATE_TABLE);
    }

    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){
        db.execSQL("DROP TABLE IF EXISTS " + Member.TABLE_NAME);

        onCreate(db);
    }

    public long insertMember(Member member){
        SQLiteDatabase db = this.getWritableDatabase();

        int count = 0;
        count = memberCount();

        ContentValues values = new ContentValues();
        values.put(Member.COLUMN_FIRST_NAME, member.getFirstName());
        values.put(Member.COLUMN_LAST_NAME, member.getLastName());
        values.put(Member.COLUMN_EMAIL, member.getEmail());
        values.put(Member.COLUMN_HOUSE_NO, member.getHouseNo());
        values.put(Member.COLUMN_POST_CODE, member.getPostCode());
        values.put(Member.COLUMN_PHONE_NO, member.getPhoneNo());
        values.put(Member.COLUMN_MEMBER, member.getMember());

        long id = db.insert(Member.TABLE_NAME, null, values);
        db.close();
        return id;
    }

    public Member getMember(long id){
        SQLiteDatabase db = this.getReadableDatabase();

        Cursor cursor = db.query(Member.TABLE_NAME,
                new String[]{Member.COLUMN_ID, Member.COLUMN_FIRST_NAME, Member.COLUMN_LAST_NAME, Member.COLUMN_EMAIL, Member.COLUMN_HOUSE_NO, Member.COLUMN_POST_CODE, Member.COLUMN_PHONE_NO, Member.COLUMN_MEMBER},
                Member.COLUMN_ID + "=?",
                new String[]{String.valueOf(id)},
                null, null, null, null);

        if(cursor!=null){
            cursor.moveToNext();
        }

        Member member = new Member(
                cursor.getInt(cursor.getColumnIndex(Member.COLUMN_ID)),
                cursor.getString(cursor.getColumnIndex(Member.COLUMN_FIRST_NAME)),
                cursor.getString(cursor.getColumnIndex(Member.COLUMN_LAST_NAME)),
                cursor.getString(cursor.getColumnIndex(Member.COLUMN_EMAIL)),
                cursor.getString(cursor.getColumnIndex(Member.COLUMN_HOUSE_NO)),
                cursor.getString(cursor.getColumnIndex(Member.COLUMN_POST_CODE)),
                cursor.getString(cursor.getColumnIndex(Member.COLUMN_PHONE_NO)),
                cursor.getString(cursor.getColumnIndex(Member.COLUMN_MEMBER)));
        cursor.close();

        return member;
    }

    public int memberCount(){
        String query = "SELECT * FROM " + Member.TABLE_NAME;
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(query, null);
        int count = cursor.getCount();
        cursor.close();

        return count;
    }

    public void updateMember(Member member){
        SQLiteDatabase db = this.getWritableDatabase();

        ContentValues values = new ContentValues();
        values.put(Member.COLUMN_FIRST_NAME, member.getFirstName());
        values.put(Member.COLUMN_LAST_NAME, member.getLastName());
        values.put(Member.COLUMN_EMAIL, member.getEmail());
        values.put(Member.COLUMN_HOUSE_NO, member.getHouseNo());
        values.put(Member.COLUMN_POST_CODE, member.getPostCode());
        values.put(Member.COLUMN_PHONE_NO, member.getPhoneNo());
        values.put(Member.COLUMN_MEMBER, member.getMember());

        db.update(Member.TABLE_NAME, values, Member.COLUMN_ID + " = ?", new String[]{String.valueOf(member.getId())});
        db.close();

    }

    public void deleteMember(Member member){
        SQLiteDatabase db = this.getWritableDatabase();
        db.delete(Member.TABLE_NAME, Member.COLUMN_ID + " = ?", new String[]{String.valueOf(member.getId())});
        db.close();
    }

    public int getID(){
        SQLiteDatabase db = this.getReadableDatabase();
        String query = "SELECT " + Member.COLUMN_ID + " FROM " + Member.TABLE_NAME;
        Cursor cursor = db.rawQuery(query, null);
        int i = 0;

        if(cursor.moveToLast()){
            i = cursor.getInt(0);
        }

        cursor.close();
        db.close();
        return i;
    }

    public boolean exist(int id){
        SQLiteDatabase db = this.getReadableDatabase();
        String query = "Select * FROM " + Member.TABLE_NAME + " WHERE " + Member.COLUMN_ID + " = " + id;
        Cursor cursor = db.rawQuery(query, null);
        if (cursor.getCount() <= 0){
            return false;
        }
        cursor.close();
        return true;
    }
}
```

References

Hedley, Y. (2017) *GRASP 2017*.

Hedley, Y. (2018) *Design Patterns Gof 1 2017 V1 Feedback*.

Hedley, Y. (2018) *Design Patterns Gof 2 2017 V1*.

Facade Pattern (2018) available from <https://en.wikipedia.org/wiki/Facade_pattern> [28 March 2018]

Intent | Android Developers (2018) available from <https://developer.android.com/reference/android/content/Intent.html> [29 March 2018]

Tamada, R. (2018) *Android Sqlite Database Tutorial* [online] available from <https://www.androidhive.info/2011/11/android-sqlite-database-tutorial/> [29 March 2018]