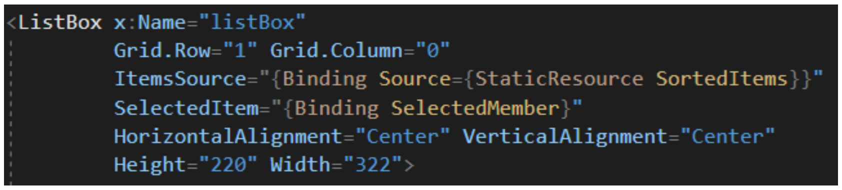
Transferring Data Between Windows

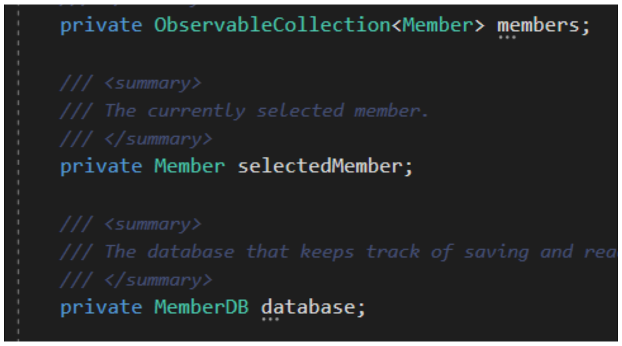
For our fifth assignment, we needed to make an update window which provides input boxes for the user to enter information, but these input boxes needed to already be pre-filled with information of the member the user selected in the main window. To solve this problem, we created a Member class to store information about a Member listed, a MessageMember class (which is like the NotificationMessage class we created in our previous lab on Stocks) to send back and forth information about a Member and any additional information through, and a Member DB class for retrieving or writing to our database of members (or text file in our case) accordingly. We make an instance of this database and a Member (called selectedMember) and store a private list of members in the MainViewModel. In the MainWindow, we bind SelectedItem to SelectedMember so that when a user selects an item from our list box, the information of the selected member will be stored in the MainWindow's selectedMember variable.

MainWindow.xaml

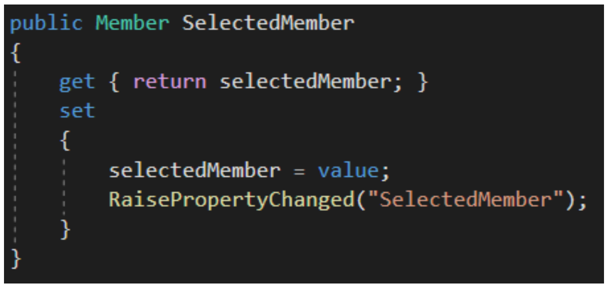


```
<ListBox x:Name="listBox"
        Grid.Row="1" Grid.Column="0"
        ItemsSource="{Binding Source={StaticResource SortedItems}}"
        SelectedItem="{Binding SelectedMember}"
        HorizontalAlignment="Center" VerticalAlignment="Center"
        Height="220" Width="322">
```

MainViewModel.cs



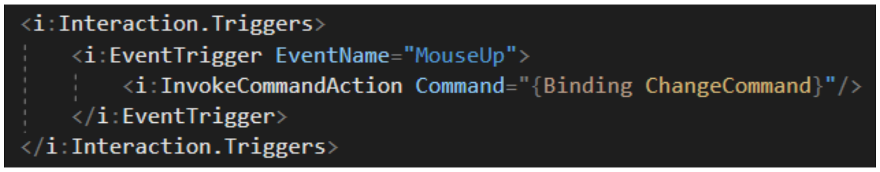
```
    private ObservableCollection<Member> members;

    /// <summary>
    /// The currently selected member.
    /// </summary>
    private Member selectedMember;

    /// <summary>
    /// The database that keeps track of saving and rea
    /// </summary>
    private MemberDB database;
```



```
public Member SelectedMember
{
    get { return selectedMember; }
    set
    {
        selectedMember = value;
        RaisePropertyChanged("SelectedMember");
    }
}
```

When the user’s mouse lifts after selecting a member in the list box in our MainWindow, the ChangeCommand of the MainWindow will be invoked.

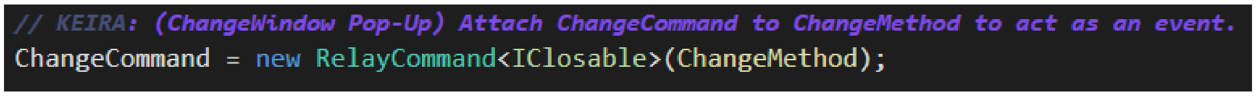
MainWindow.xaml



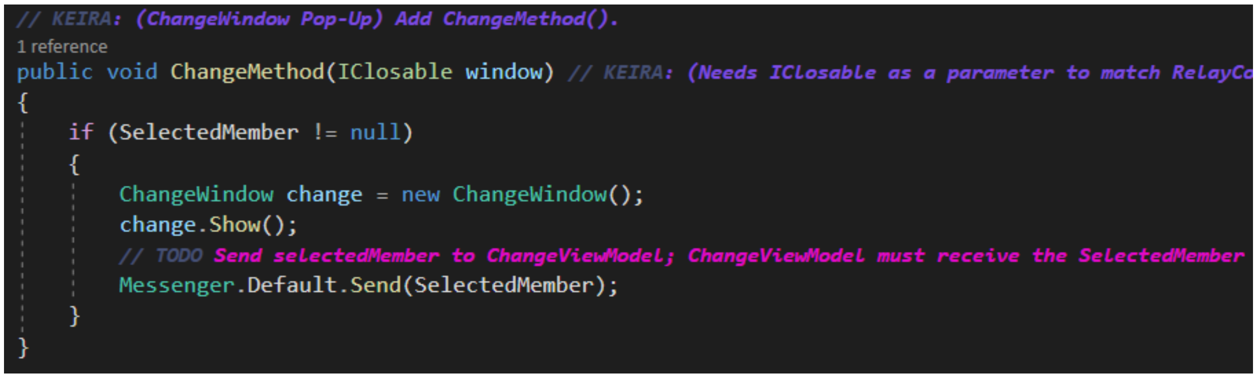
```xml
<i:Interaction.Triggers>
    <i:EventTrigger EventName="MouseUp">
        <i:InvokeCommandAction Command="{Binding ChangeCommand}"/>
    </i:EventTrigger>
</i:Interaction.Triggers>
```

Since we’ve attached our ChangeCommand to ChangeMethod() in the MainViewModel constructor, ChangeMethod() will execute as soon as ChangeCommand is invoked.

MainViewModel.cs



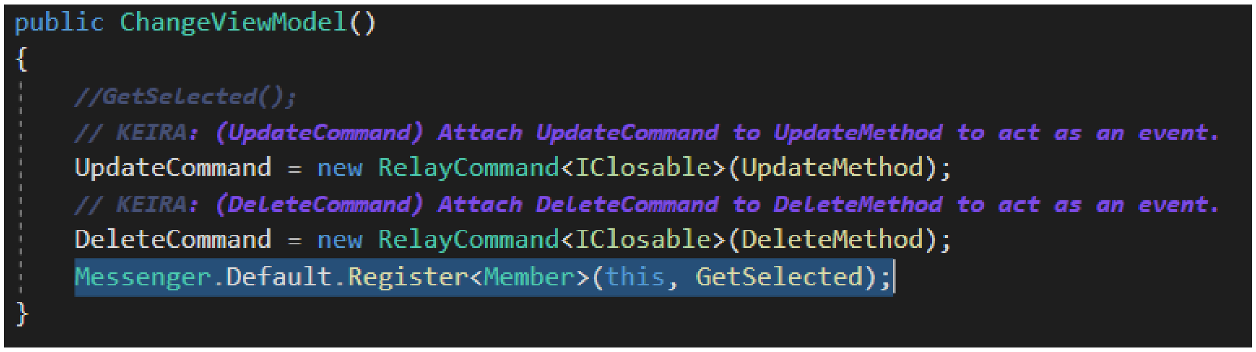
```csharp
// KEIRA: (ChangeWindow Pop-Up) Attach ChangeCommand to ChangeMethod to act as an event.
ChangeCommand = new RelayCommand<IClosable>(ChangeMethod);
```



```csharp
// KEIRA: (ChangeWindow Pop-Up) Add ChangeMethod().
1 reference
public void ChangeMethod(IClosable window) // KEIRA: (Needs IClosable as a parameter to match RelayCo
{
    if (SelectedMember != null)
    {
        ChangeWindow change = new ChangeWindow();
        change.Show();
        // TODO Send selectedMember to ChangeViewModel; ChangeViewModel must receive the SelectedMember
        Messenger.Default.Send(SelectedMember);
    }
}
```

And in our ChangeMethod(), we use the method Send() from the Messaging namespace we’ve imported to send the information of the selected member in the MainWindow that we’ve stored in MainViewModel’s own variable, SelectedMember.

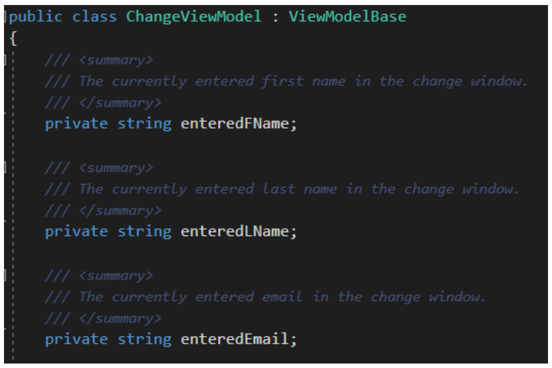
ChangeViewModel.cs



```
public ChangeViewModel()
{
    //GetSelected();
    // KEIRA: (UpdateCommand) Attach UpdateCommand to UpdateMethod to act as an event.
    UpdateCommand = new RelayCommand<IClosable>(UpdateMethod);
    // KEIRA: (DeleteCommand) Attach DeleteCommand to DeleteMethod to act as an event.
    DeleteCommand = new RelayCommand<IClosable>(DeleteMethod);
    Messenger.Default.Register<Member>(this, GetSelected);
}
```

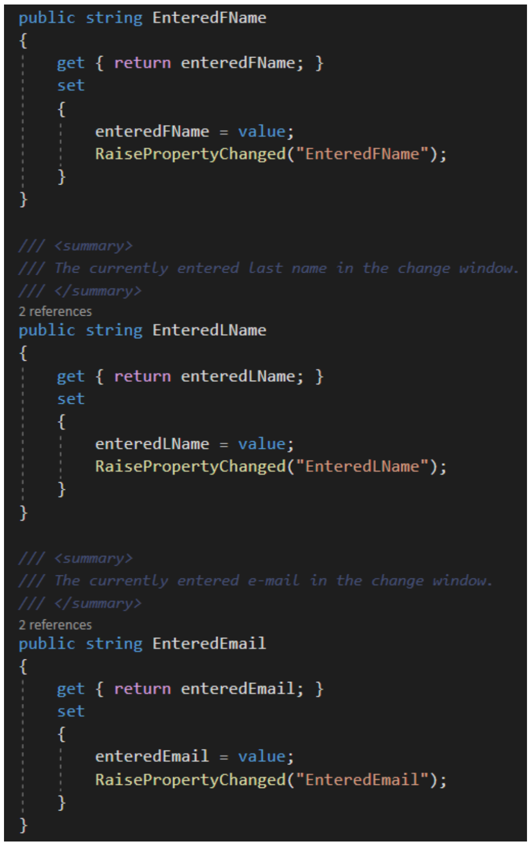
Since ChangeViewModel has been registered to this event, ChangeViewModel's method with the matching signature will receive the SelectedMember from MainWindow (GetSelected()) since SelectedMember was an object of type "Member" and here we will set ChangeViewModel's own variables, enteredFName, enteredLName, and enteredEmail, to the corresponding values of the Member that the MainViewModel sent,

ChangeViewModel.cs



```
public class ChangeViewModel : ViewModelBase
{
    /// <summary>
    /// The currently entered first name in the change window.
    /// </summary>
    private string enteredFName;

    /// <summary>
    /// The currently entered last name in the change window.
    /// </summary>
    private string enteredLName;

    /// <summary>
    /// The currently entered email in the change window.
    /// </summary>
    private string enteredEmail;
```

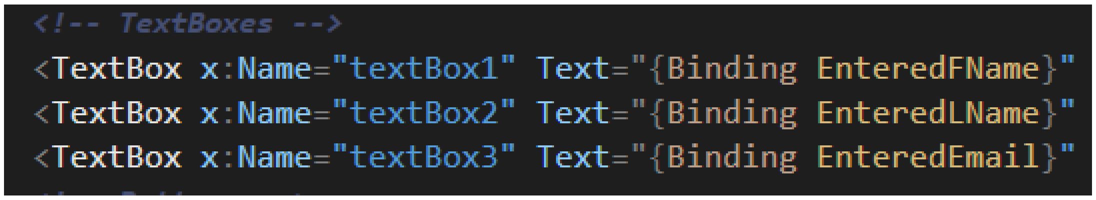
.



```
public string EnteredFName
{
    get { return enteredFName; }
    set
    {
        enteredFName = value;
        RaisePropertyChanged("EnteredFName");
    }
}

/// <summary>
/// The currently entered last name in the change window.
/// </summary>
2 references
public string EnteredLName
{
    get { return enteredLName; }
    set
    {
        enteredLName = value;
        RaisePropertyChanged("EnteredLName");
    }
}

/// <summary>
/// The currently entered e-mail in the change window.
/// </summary>
2 references
public string EnteredEmail
{
    get { return enteredEmail; }
    set
    {
        enteredEmail = value;
        RaisePropertyChanged("EnteredEmail");
    }
}
```

Which will update the text boxes in our ChangeWindow when ChangeViewModel's private variables are set to another value since the ChangeWindow's text boxes are binded to those properties.

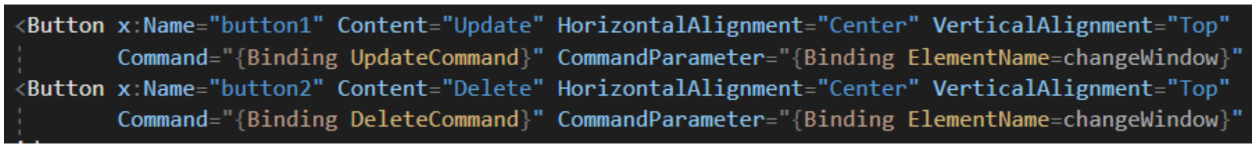
ChangeWindow.xaml



```
<!-- TextBoxes -->
<TextBox x:Name="textBox1" Text="{Binding EnteredFName}"
<TextBox x:Name="textBox2" Text="{Binding EnteredLName}"
<TextBox x:Name="textBox3" Text="{Binding EnteredEmail}"
```

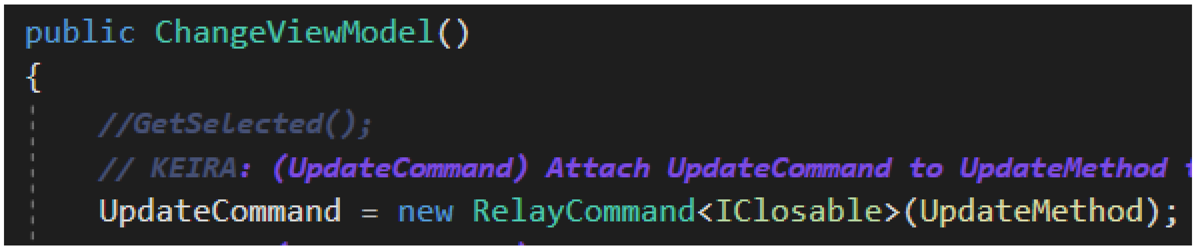
If in our ChangeWindow, the user selects the update button, our UpdateCommand will be invoked and our UpdateMethod will be called since the update button in our ChangeWindow is binded to ChangeViewModel's UpdateCommand and we have attached our UpdateCommand to our UpdateMethod in ChangeViewModel's constructor.

ChangeWindow.xaml



```
<Button x:Name="button1" Content="Update" HorizontalAlignment="Center" VerticalAlignment="Top"
        Command="{Binding UpdateCommand}" CommandParameter="{Binding ElementName=changeWindow}"
<Button x:Name="button2" Content="Delete" HorizontalAlignment="Center" VerticalAlignment="Top"
        Command="{Binding DeleteCommand}" CommandParameter="{Binding ElementName=changeWindow}"
```

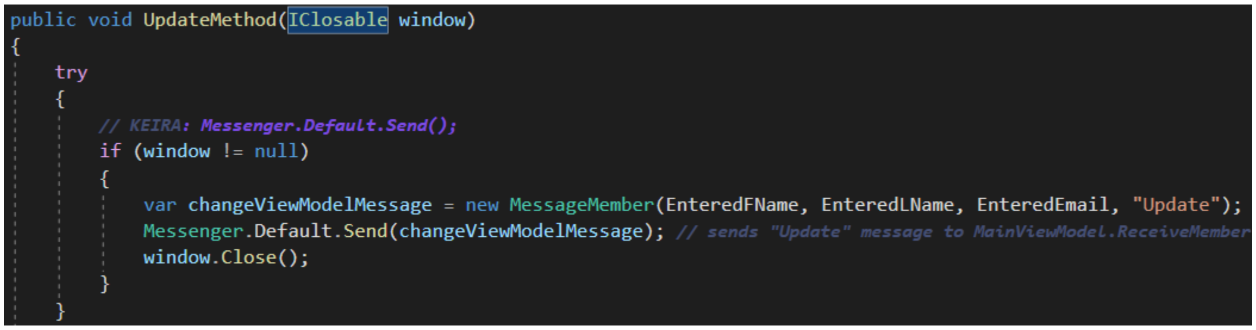
ChangeViewModel.cs



```
public ChangeViewModel()
{
    //GetSelected();
    // KEIRA: (UpdateCommand) Attach UpdateCommand to UpdateMethod
    UpdateCommand = new RelayCommand<IClosable>(UpdateMethod);
```

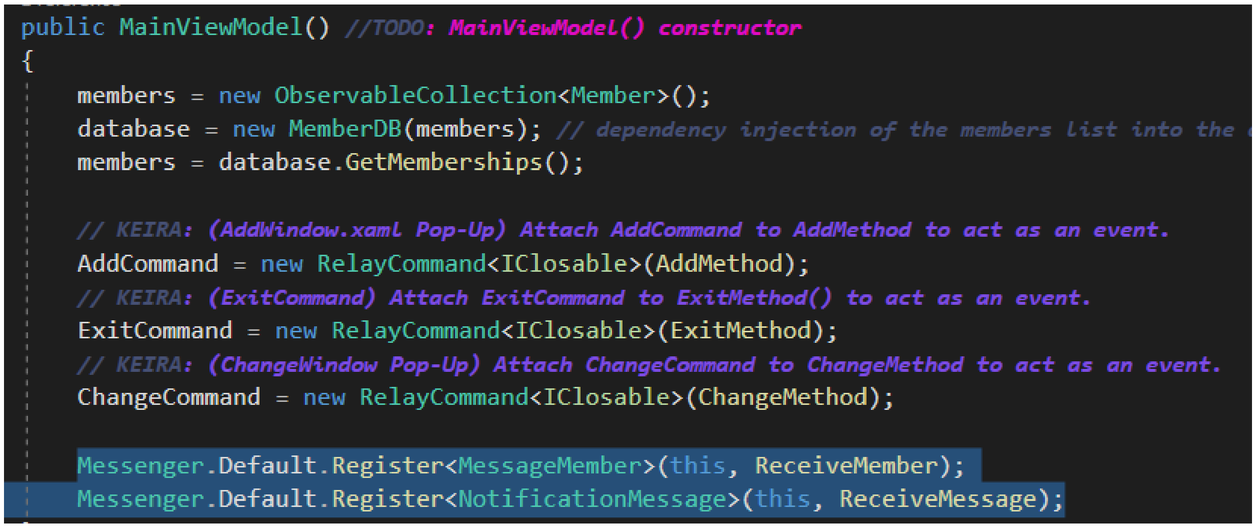
Then in ChangeViewModel's UpdateMethod(), we will create a MessageMember instance with the information that was inputted in ChangeWindow's text boxes and stored in ChangeViewModel's variables along with a message ("Update" or "Delete", but we send "Update" in this case).

ChangeViewModel.cs



```
public void UpdateMethod(IClosable window)
{
    try
    {
        // KEIRA: Messenger.Default.Send();
        if (window != null)
        {
            var changeViewModelMessage = new MessageMember(EnteredFName, EnteredLName, EnteredEmail, "Update");
            Messenger.Default.Send(changeViewModelMessage); // sends "Update" message to MainViewModel.ReceiveMember
            window.Close();
        }
    }
```

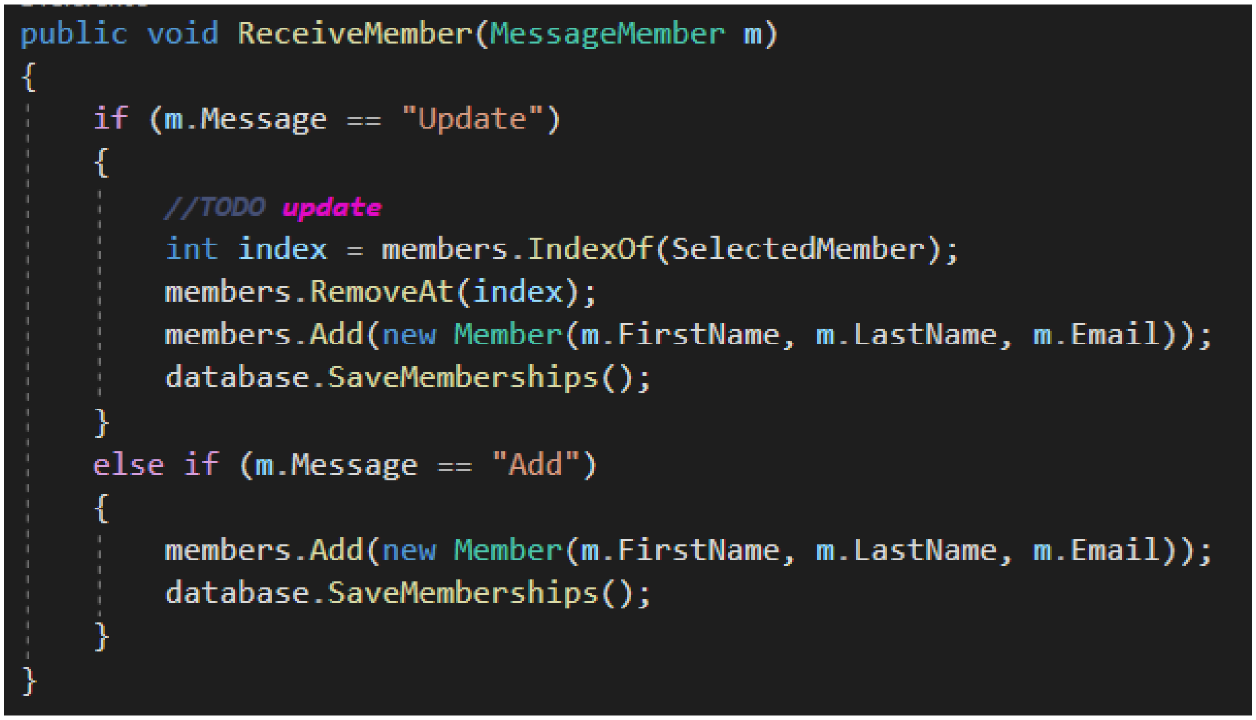
Because MainViewModel is registered to this kind of event as in MainViewModel's constructor,



```
public MainViewModel() //TODO: MainViewModel() constructor
{
    members = new ObservableCollection<Member>();
    database = new MemberDB(members); // dependency injection of the members list into the
    members = database.GetMemberships();

    // KEIRA: (AddWindow.xaml Pop-Up) Attach AddCommand to AddMethod to act as an event.
    AddCommand = new RelayCommand<IClosable>(AddMethod);
    // KEIRA: (ExitCommand) Attach ExitCommand to ExitMethod() to act as an event.
    ExitCommand = new RelayCommand<IClosable>(ExitMethod);
    // KEIRA: (ChangeWindow Pop-Up) Attach ChangeCommand to ChangeMethod to act as an event.
    ChangeCommand = new RelayCommand<IClosable>(ChangeMethod);

    Messenger.Default.Register<MessageMember>(this, ReceiveMember);
    Messenger.Default.Register<NotificationMessage>(this, ReceiveMessage);
```

And the type of object our method in ChangeViewModel sent is of type "MessageMember," MainViewModel's method, ReceiveMember(), is the method which will receive the member sent from ChangeViewModel's method, UpdateMethod().

ChangeViewModel.cs



```
public void ReceiveMember(MessageMember m)
{
    if (m.Message == "Update")
    {
        //TODO update
        int index = members.IndexOf(SelectedMember);
        members.RemoveAt(index);
        members.Add(new Member(m.FirstName, m.LastName, m.Email));
        database.SaveMemberships();
    }
    else if (m.Message == "Add")
    {
        members.Add(new Member(m.FirstName, m.LastName, m.Email));
        database.SaveMemberships();
    }
}
```

Here in ChangeViewModel's ReceiveMember() method, the first portion of our if block will execute since we have included the string "Update" along with the member's information in the MessageMember instance we sent. This will find the index of that member in MainViewModel's private list of members, replace that member, update MainViewModel's own list and finally have the database update accordingly.

TEAM MEMBER’S WORK

Jerry Belmonte

- Member Database
- Add
- Delete / Update

Keira Wong

- Delete / Update
- Cancel / Exit