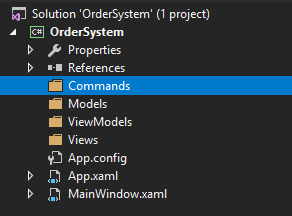
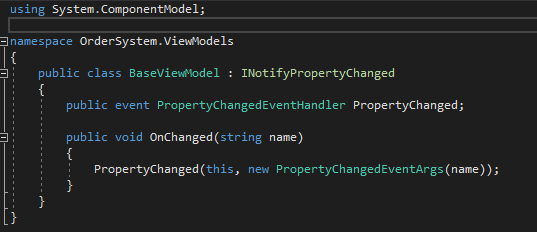
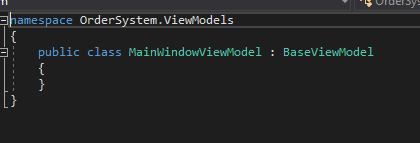
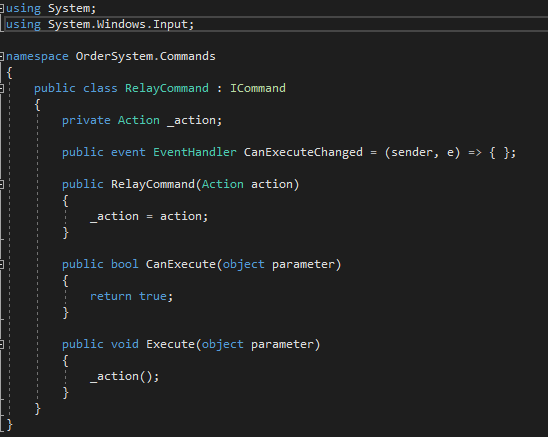
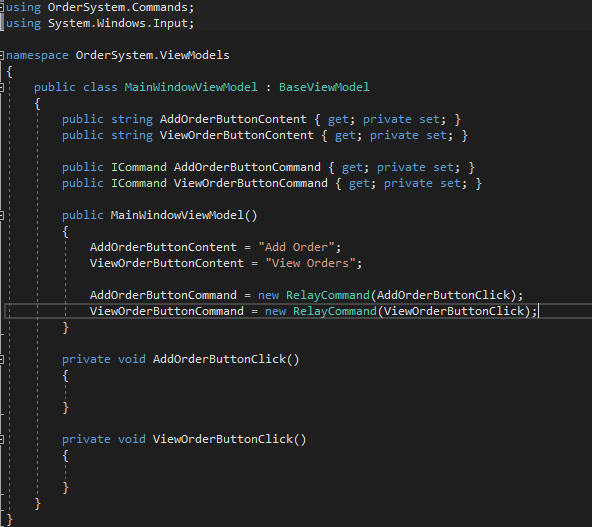
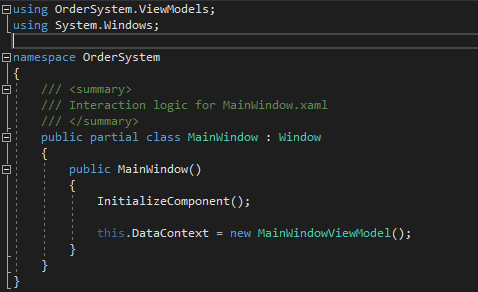
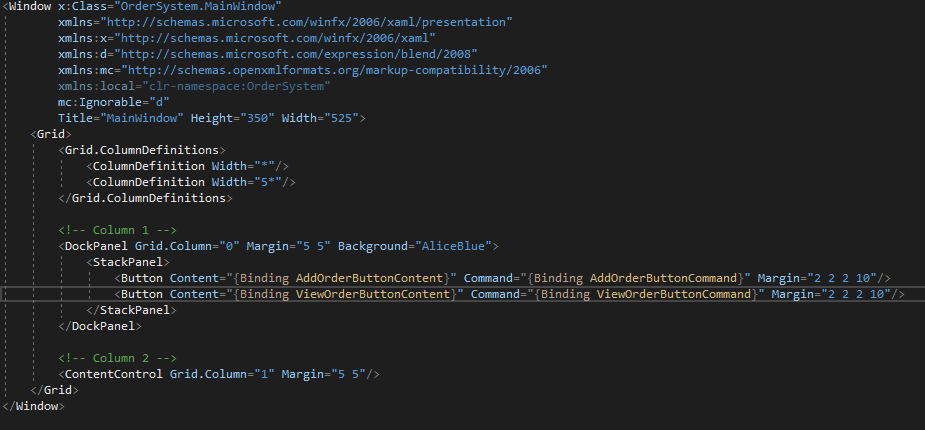
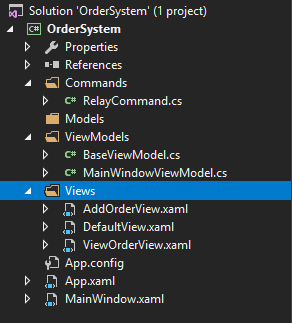
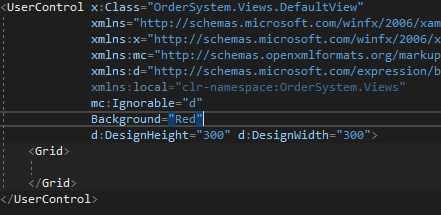
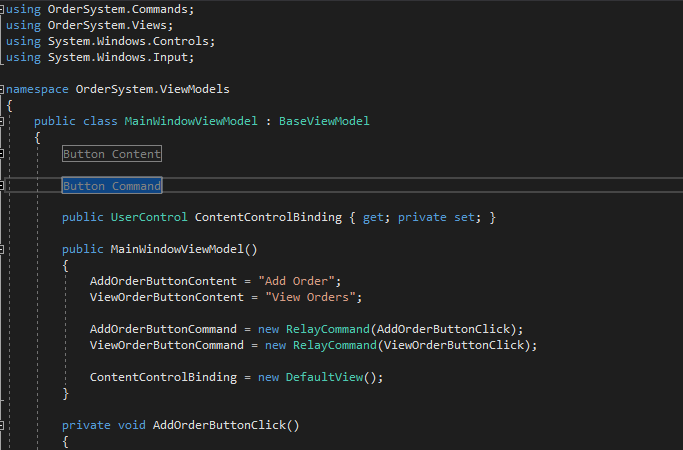
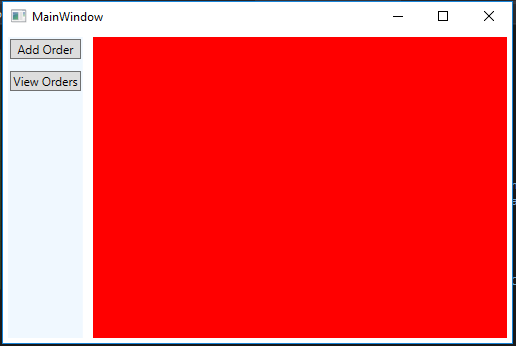
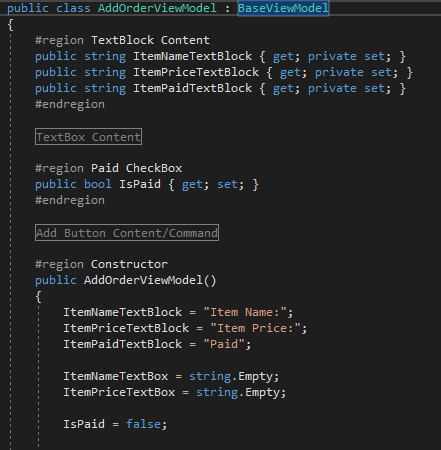
**Practical 2**

**Advanced MVVM WPF**

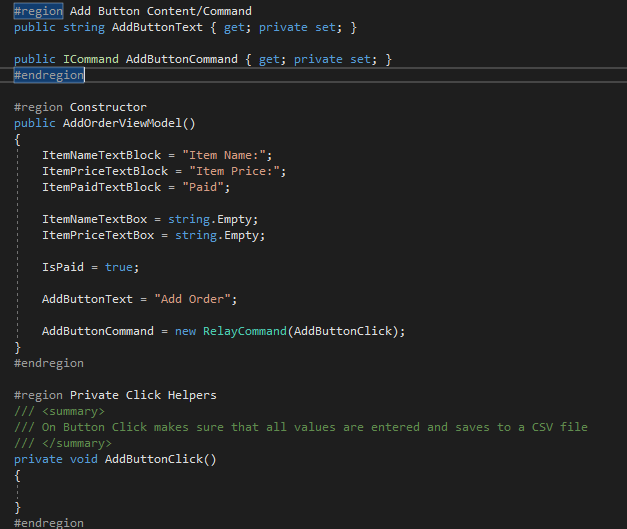
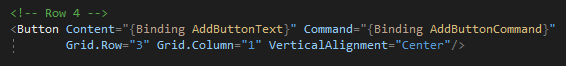
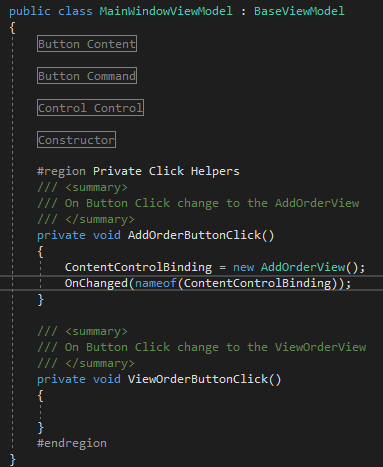
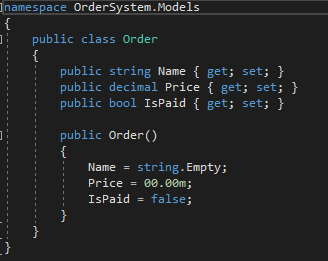
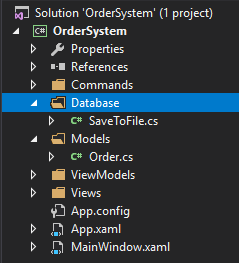
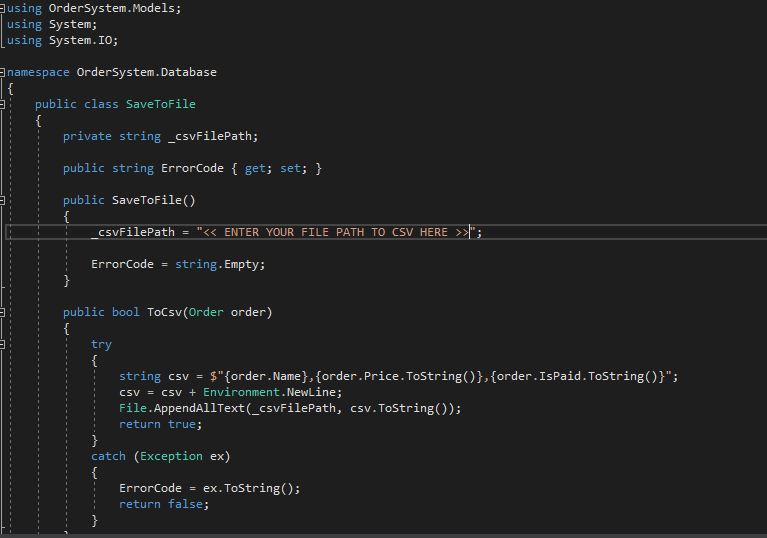
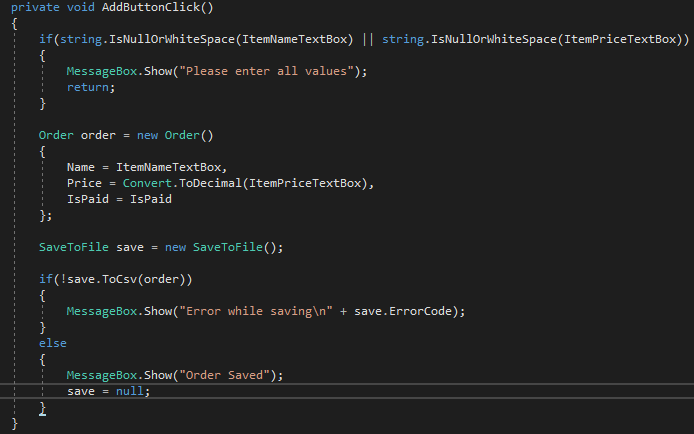
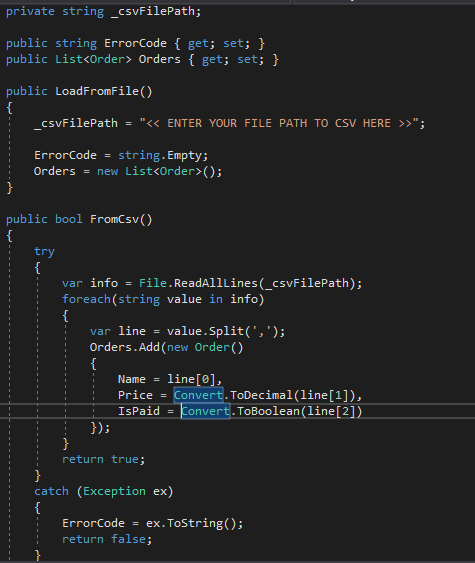
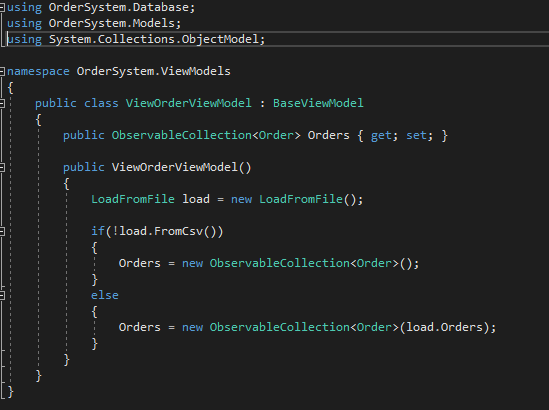
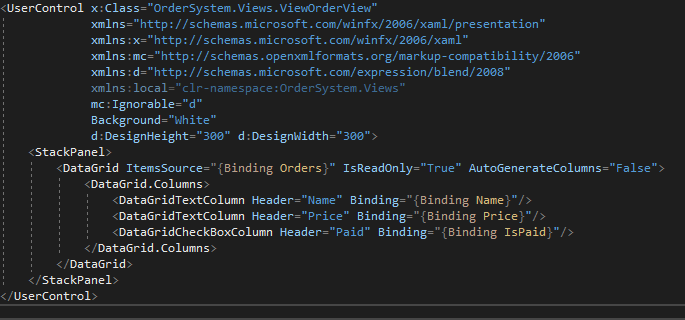
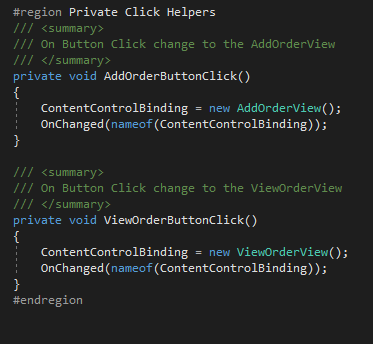
This practical will show you how to switch between screens on a WPF application following the MVVM Design Pattern.

It is to be completed as a follow on from last week’s exercise.

1. Create a new project and call it OrderSystem
2. Right click on the Project Name and go to Add and add to following folders to the Project
3. Add the *BaseViewModel* class to the *ViewModels* folder 
4. Add a blank *MainWindowViewModel* class to the *ViewModels* folder and inherit the *BaseViewModel* class
   1. 
5. Add a new class to the Commands folder called *RelayCommand* and add the following code
   1. 
6. Now go back to the *MainWindowViewModel* class and add the following code
   1. 
   2. Very similar to last weeks exercise but no textboxes this time
7. Next go to the *MainWindow.Xaml.Cs* file and add the following to set the windows *DataContext* to the *MainWindowViewModel* that we just created. Observe the *using* statements
   1. 
   2. Close the *MainWindow.Xaml.Cs* file as we will not be using it from now on
8. Now add the following code to the *MainWindow.Xaml* page
   1. 
   2. Notice in Column 2 there is a new item called *ContentControl*
   3. This is what is going to become the screen that will switch out on button press
   4. Remember that the variables in the Binding braces need to match exactly the variable names in the *MainWindowViewModel* class
9. Now in the Views folder right click, go to add and select UserControl. Add 3 UserControls to the folder with the following names
   1. 
   2. Notice how everything in the *Views* folder ends with the word *View* and in the *ViewModels* folder all classes end with *ViewModel*.
   3. This is a standard naming convention to denote what each item is.
10. Open up the DefaultView.Xaml file. Notice that there is only a border of the UserControl. This is because by default UserControls are set to take on the background colour of the WPF window it is being displayed on. For the sake of this tutorial we will change the colour of it so it is visible.
    1. 
11. Now back to the *MainWindowViewModel* class and add the following code
    1. 
    2. The line *UserControl ContentControlBinding* has been added and set to the *DefaultView* in the constructor
    3. **Ignore the fact that the Button Content and Button Command Variables have been replaced with a greyed out box**.
12. Go back to the MainWindow.Xaml file and add the following to the ContentControl item
    1. 
13. If you compile and run the program now you should see the red screen we just set on the right hand side of the screen as follows
    1. 
    2. We only used red for demo purposes by all means change the colour to whatever you want
14. Now all the work with the *MainWindow.Xaml* file is done. You can close it
15. Add a new class to the *ViewModels* folder call *AddOrderViewModel* and have it inherit the *BaseViewModel* class and add the following code



* 1. The #region tags allow you to collapse everything between the tags to help save space.

1. On the *AddOrderView.Xaml.Cs* set this to the *DataContext* just like we did with the *MainWindow*
2. Add the following code to the AddOrderView.Xaml
   1. 
   2. Now to add the button we will use to Save this order
3. Go back to the AddOrderViewModel class and add the Button Content and Button Command as we have done previously
   1. 
   2. Now add a button the AddOrderView.Xaml and bind the Content and Command to the new properties
   3. Ive left space to add a Clear button if you so wish to do one.
   4. Close the *AddOrderView.Xaml* file as now we are finished.
4. Go back to *theMainWindowViewModel* class and add the following code
   1. Now if you compile and run your code and press the Add Order button the screen should switch out to the screen we just created.
   2. If you get the error saying that the Application is in Break Mode check the get and set values of the *TextBoxes* in the *AddOrderVeiwModel* neither of them should be private
5. In the Models folder create a class called Order and enter the following properties into it
   1. 
6. Right click on the Project and add a new folder called Database and add a class to the new folder called SaveToFile
   1. 
7. Now in the SavToFile we need to create a connection string to a csv, and a method that takes in the new Order and save it to the Csv
   1. 
8. Back to the *AddOrderViewModel* class. We will now run the checks to see if the textboxes are valid and then save to the csv file
   1. 
   2. If you compile and run the application you can now save to and from the csv file
9. Now to read from the csv and display on the screen
10. Add a new class to the *Database* folder called *LoadFromFile*. This will load from the csv file and from there we will bind it to a property on screen.
    1. 
11. Now create a class called *ViewOrderViewModel* and add to the *ViewModels* class. And add have it inherit the *BaseViewModel* class again.
    1. The *ObservableCollection* will get bound to a grid on the UI that will display all the Orders that was in the csv file we have created.
12. Now on the *ViewOrderView.Xaml.Cs* set the *DataContext* as before to the *ViewOrderViewModel* we have just created.
    1. Continued below
13. In the VeiwOrderView.Xaml add the following to bind the csv file contents to the UI
    1. 
14. Now all that’s left to do is to hook up the ViewOrderView to the button click event on the MainWindowViewModel. Go to the MainWindowViewModel and add the following to the relevant method
    1. 
15. If you compile your code now and run it you should be able to Add an Order and View all the orders. Whilst switching between different views.

In this practical, you have learned how to:

* Properly organise a file structure in a WPF project.
* Separate UI and Logic completely
* Separate different concerns into their own classes
  + Having anything to do with saving to a file contained together
  + All Load functionalities are grouped together
* Bind and display data to the user.

All this seems like a lot of work but in the real world a project will be huge in comparison to these ones and you will need to learn how to start making your projects more modular in the future. In a future practical, we will start splitting everything up even more so that there is even more de-coupling of concerns. The more modular the program the more maintainable and the easier it is to debug as well.

If you are interested in Design Principles I strongly recommend that you watch videos or read about the SOLID Principles.

This does not appear as questions in your coursework or exams this is purely for those wanting to learn more outside of what is taught