Design Pattern in MVVM architecture

Background:

Larman chap. 17 + 26 (singleton + observer).

Windows GUI design guides

- Overview: https://msdn.microsoft.com/en-us/library/windows/apps/dn894631.aspx
- guidelines: https://msdn.microsoft.com/en-us/library/windows/apps/hh465424.aspx
- guidelines for controls etc.: https://msdn.microsoft.com/en-us/library/windows/apps/dn611856.aspx

CSharpNote (Per Laursen) p. 131-180

Assignment 1: Start GUI

Create two folders (i.e. two layers) 'model' and 'viewmodel'.

Model

In model – layer / folder create a class 'Person' with properties 'Name' and 'PhoneNo', remember to create a constructor and the 'ToString'-method.

Viewmodel

In viewmodel – layer / folder create a class 'MainViewModel'

In MainViewModel:

- Implement the 'INotifyPropertyChanged'
- Make a Property of persons as ObservableCollection <Person>
- Make a Property SelectedPerson of type Person, remember to call OnPropertyChanged() in the set-method
- Create a constructor, add at least two Persons to the List (ObservableCollection) of persons.

Questions

Why do we split the app up into View, ViewModel and Model?

What do we have in each layer i.e. which functions do we have in each layer?

Which one of the layers fulfil the controller pattern?

Explain the link between low coupling and layers.

Why is INotifyPropertyChanged necessary? (Who are using it?)

Explain the link between low coupling and INotifyPropertyChanged.

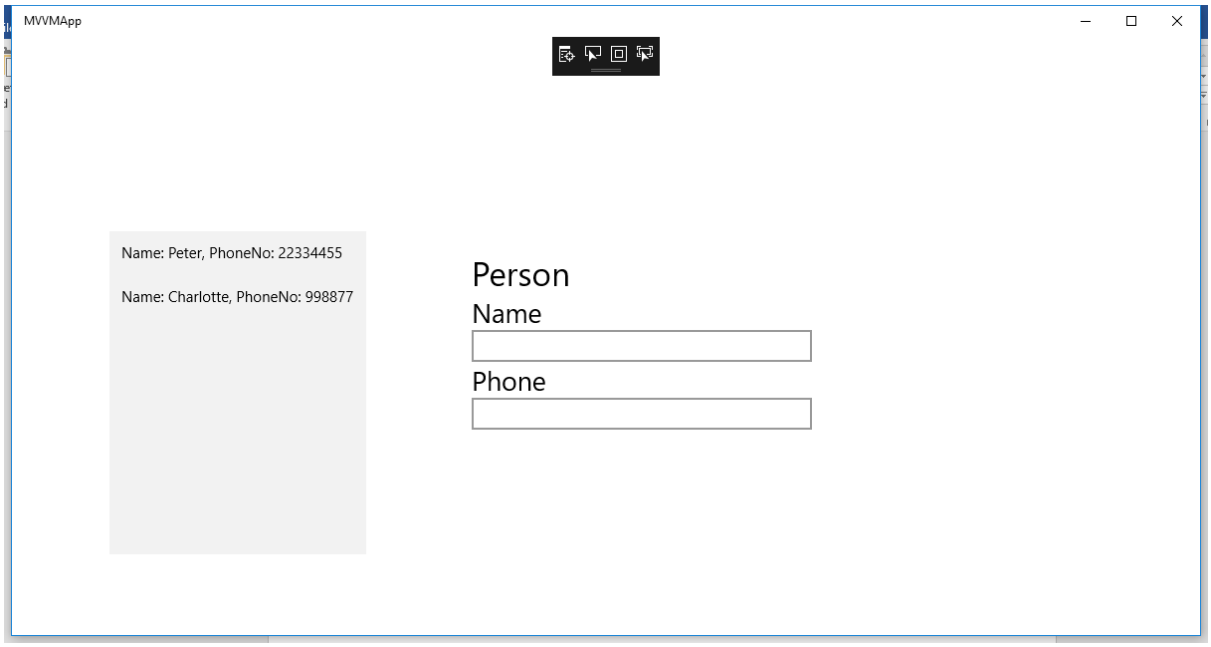
Explain High Cohesion in relation to the three layers.

Explain the INotifyPropertyChanged in terms of the Observer Pattern

View

Build / create a GUI to show and select persons.

It could look like this:



Having a listbox for the person-collection. And a stackpanel with 3 textblocks and 2 textboxes.

In the GUI (opened in blend), add the MainViewModel as a DataContext to the Page.

Bind the listbox (ItemsSource property) to the persons (ObservableCollection) and the textbox for the name to selectedPerson.Name and the Phone textbox to SelectedPerson.PhoneNo.

Then bind the listbox (SelectedItem) to the SelectedPerson – remember to do it TwoWay

Now you should be able to run the app and select a person which then will be shown beside.

Assignment 2: Add Create Functionality

ViewModel

Make a new class 'RelayCommand'.

Implement the ICommand interface.

It should look like this:

```
class RelayCommand:ICommand
{
    private Action _function;

    public RelayCommand(Action function)
    {
      _function = function;
    }

    public bool CanExecute(object parameter)
    {
      return true;
    }

    public void Execute(object parameter)
    {
      _function();

    }

    public event EventHandler CanExecuteChanged;
  }
```

Modify the 'MainViewModel', to support the 'Add person' functionality.

- Create a method with the signature and body
  public void AddPersonMethod(){
    Person person = new Person(SelectedPerson.Name, SelectedPerson.PhoneNo);
    persons.Add(person);
  }
- Add a property of RelayCommand type give it the name 'AddPerson'
- In the constructor assign to the AddPerson property an object of RelayCommand where the method AddPersonMethod are passed as parameter to the constructor.

View

To the view add a button and bind it to the AddPerson command in the MainViewModel.

Modify the textbox – Name binding to be TwoWay. Do the same for the textbox – PhoneNo.

Now you would be able to add persons to the list of persons.

Assignment 3: Make Shared class to all ViewModels (singleton)

Next step is to create a class which is a singleton, whereby all instances of viewModels will only have one single object.

Create a class 'SharedInformation'

Make this class a singleton i.e.

1. Make the constructor private
2. Make a static instance field of the class it self + make an object.
3. Make a static method Get, which return the object of the class it self.

Now this class is a singleton and wherever you are in the system, you will only get the same object.

For this case then the shared info is the selected Person, so move the SelectedPerson property into the SharedInformation – class including OnPropertyChenged etc.

Modify you MainViewModel, so it have the sharedInfoemation as a property (at least the get-method).

Change the bindings in the app to be binded to the selectedPerson now placed inside the sharedInformation.

Extra Assignment A: A second GUI

Modify the model class person with a property Address.

Now create a new view e.g. 'DetailedView', which show all details of a person i.e. Name,Phone and Address.

In the DetailedView add the DataContext to be the MainViewModel and some textboxes to show each of the properties in a person.

In the MainPage make a button to navigate to this new detailed view, i.e. in blend add a button + add behaviour 'Navigate to'.

Check to see this is working – otherwise you can have problem with your Singleton.