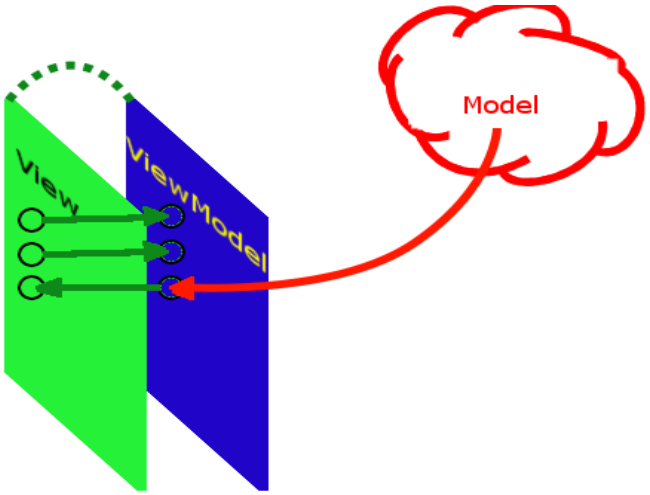
Рассмотрение паттерна MVVM на примере: Сложение двух чисел с выводом результата

В примере (**WpfMVVMdemo01**) рассматриваются следующие действия:

1. Разработка модели программы.
2. Реализация интерфейса программы.
3. Связывание интерфейса и модели прослойкой ViewModel (VM).

Задание: написать программу сложения двух чисел с выводом результата.

Согласно шаблона MVVM структура решения можно представить в следующем виде:



Предполагается, что пользователь вводит в два текстовых поля числа и сразу же в третьем текстовом поле выводится сумма.

За выполнение операции сложения будет отвечать **модель**.

За соединение интерфейса **View** (которое ничем иным, кроме как приема ввода от пользователя и предоставления ему вывода не занимается) и модели (в которой происходит вычисление) будет отвечать **ViewModel**.

Решение.

Создание проекта

1. Создайте проект WPF с именем WpfMVVMdemo01.
2. Добавьте в проект три папки: Model, ViewModel и View.
3. В папку View переместите файл главного окна MainWindow.
4. Измените в файле App.xaml путь к главному окну:

```
StartupUri="View/MainWindow.xaml"
```

Создание модели

1. Добавьте в папку Model класс **MathFuncs**.

Моделью в данной задаче будет сложение чисел с возвратом результата. Модель, в принципе, может не хранить никакого состояния, т.е. она может быть реализована статическим методом статического класса, например:

```
class MathFuncs
{
 //   public static int GetSumOf(int a, int b) => a + b; // C#6.0

    public static int GetSumOf(int a, int b)
    {
       return a + b;
    }
  }
```

Создание представления (View) – интерфейса программы

Согласно условию задачи **View** будет содержать три текстовых поля, которые можно сопроводить надписями (в данном решении это не реализовано, можете реализовать на свое усмотрение, например: число номер один, число номер два, сумма).

View реализуется в XAML.

Согласно рисунку (см. рис. выше) зеленые точки — это текстовые поля, а зеленые линии, соединяющиеся с синими будут реализованы через механизм привязки (**Binding**).

Зеленая пунктирная линия — связь всего **View** и **VM** – осуществляется, когда создаем объект **VM** и присваиванием его свойству DataContext **View**.

1. Реализуйте интерфейс приложения следующим образом (предлагается самый простой интерфейс, можно вместо StackPanel использовать другой менеджер компоновки, добавить надписи и т.п.):

```
<Window x:Class="WpfMVVMdemo01.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local = "clr-namespace:WpfMVVMdemo01.ViewModel"
        Title="MainWindow" Height="350" Width="525">
<Window.DataContext>
    <local:MainVM/>
        <!-- Создаем новый VM и соединяем его со View -->
</Window.DataContext>

<StackPanel>
        <!--Binding, соединяет текстовое поле со свойством в VM -->
        <!--UpdateSourceTrigger, в данном случае, выполняет передачу значения в VM в
момент ввода -->
   <TextBox Width="30" Text="{Binding Number1, UpdateSourceTrigger=PropertyChanged}"/>
   <TextBox Width="30" Text="{Binding Number2, UpdateSourceTrigger=PropertyChanged}"/>
      <!--Mode=OneWay необходим для привязки свойства только для чтения -->
   <TextBox Width="30" Text="{Binding Number3, Mode=OneWay}" IsReadOnly="True"/>
</StackPanel>
</Window>
```

Реализация ViewModel

ViewModel однозначно обуславливается **View** и не должна содержать в себе никакой «бизнес логики». Обусловленность от **View** означает, что если во **View** есть три текстовых поля, или три места, которые должны вводить/выводить данные, то

следовательно в **VM** (своего рода подложке) должны быть минимум три свойства, которые эти данные принимают/предоставляют.

Следовательно, в классе **ViewModel** должны быть два свойства, принимающие из **View** два числа, и третье свойство, вызывающую модель для выполнения бизнес-логики программы.

ViewModel ни в коем случае не выполняет сложение чисел самостоятельно, оно для этого действия только вызывает модель.

Чтобы **ViewModel** «автоматически» обновляла **View** требуется реализовать интерфейс **INotifyPropertyChange**. Именно посредством него **View** получает уведомления, что во **ViewModel** что-то изменилось и требуется обновить данные.

1. В папку ViewModel добавьте класс **MainVM.**
2. Реализуйте интерфейс INotifyPropertyChange:

```
using System.ComponentModel;

namespace WpfMVVMdemo01.ViewModel
{
    class MainVM : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        protected virtual void OnPropertyChanged(string propertyName)
        {
            PropertyChanged.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
    }
  }
```

3. Добавьте необходимые свойства: два свойства, принимающие из **View** два числа, и третье свойство, вызывающую модель для выполнения бизнес-логики программы (пунктирная синяя линия связи **ViewModel** и модели):

```
        private int _number1;
        public int Number1
        {
            get { return _number1; }
            set
            {
                _number1 = value;
                OnPropertyChanged("Number3"); // уведомление View о том, что
изменилась сумма
            }
        }

        private int _number2;
        public int Number2
        {
            get { return _number2; }
            set { _number2 = value; OnPropertyChanged("Number3"); }
        }
```

```
        //свойство только для чтения, оно считывается View каждый раз, когда
обновляется Number1 или Number2

       // public int Number3 { get; } => MathFuncs.GetSumOf(Number1,
Number2); // C#6.0

        public int Number3
        {
          get
          {
             return WpfMVVMdemo01.Model.MathFuncs.GetSumOf(Number1, Number2);
          }
        }
```

Таким образом, приложение с применением паттерна MVVM реализовано.

Постройте и протестируйте приложение. Проверьте, что при введении в первые два текстовых поля сразу меняется значение в третьем поле.