Частное учреждение образования

«Колледж бизнеса и права»

УТВЕРЖДАЮ

Ведущий

методист колледжа

\_\_\_\_\_\_\_\_\_\_\_ Е.В. Паскал

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2021

|  |  |
| --- | --- |
| Специальность: 2-40 01 01 «Программное обеспечение информационных технологий» | Дисциплина: «Технология разработки программного обеспечения» |

**Лабораторная работа № 26**

**Инструкционно-технологическая карта**

Тема: Создание программной системы, использующей шаблон MVVM

Цель: Научиться создавать программные системы, используя шаблон MVVM

Время выполнения: 2 часа

**Контрольныевопросы**

1. Опишите шаблон MVVM

**Теоретические сведения для выполнения работы**

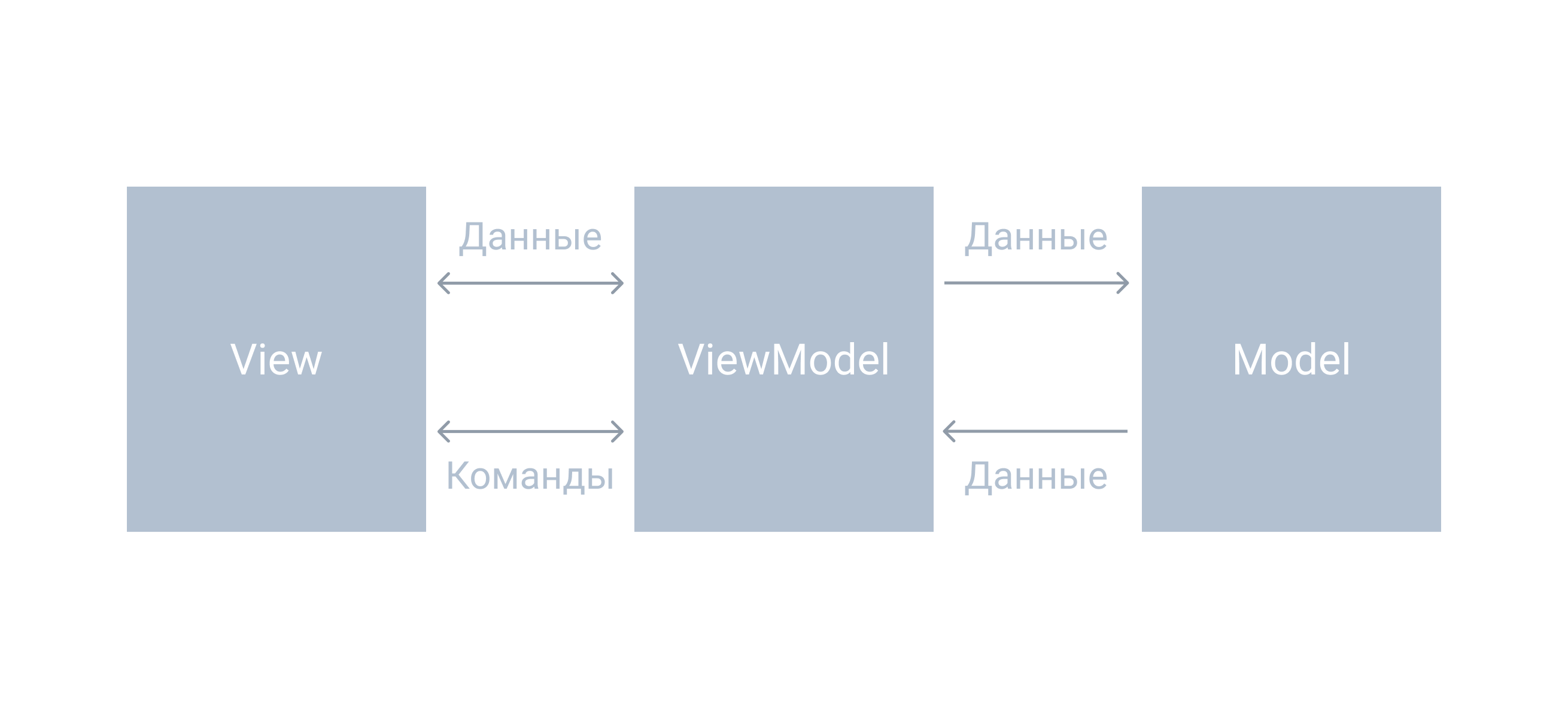
MVVM — это паттерн разработки, позволяющий разделить приложение на три функциональные части:

Model — основная логика программы (работа с данными, вычисления, запросы и так далее).

View — вид или представление (пользовательский интерфейс).

ViewModel — модель представления, которая служит прослойкой между View и Model.

Такое разделение позволяет ускорить разработку и поддерживаемость программы — можно менять один компонент, не затрагивая код другого.



Однако MVVM может быть сложно освоить, потому что он заметно отличается от более распространённых MVC и событийно-ориентированной разработки приложений.

**Как работает MVVM**

Этот паттерн можно разобрать на примере из реального мира. В главных ролях: знаменитость, PR-менеджер и пресса.

| Знаменитость (Model) | PR-менеджер (ViewModel) | Пресса (View) |
| --- | --- | --- |
| Занимается своей непосредственной работой, не отвлекаясь на продвижение. Если нужно, сообщает своему менеджеру, что произошло что-то, о чём нужно рассказать прессе. | Получает информацию от знаменитости и передаёт её прессе. Также может передать своему работодателю запрос от какой-нибудь газеты на проведение интервью или предложение сотрудничества. | Пишет публикации основываясь на данных, полученных от PR-менеджера знаменитости. |

Все компоненты работают сообща, но при этом их внутренняя кухня никак не связана. Например, пресса может поменять редакцию, изменить макет газеты, нанять новых авторов или перейти к другому владельцу. Однако на действиях PR-менеджера это никак не отразится — он будет работать так, как и работал.

Программисту этот паттерн позволяет менять отдельные части приложения, не затрагивая другие. Также он может заниматься только одним компонентом, вообще не представляя, как работают остальные. Хотя для полного понимания своей работы нужно разбираться во всех аспектах написания приложений.

Практика: пишем MVVM-приложение

Использовать MVVM можно для iOS- и Android-разработки, но лучше всего он реализован в WPF1. Что неудивительно, потому что сам паттерн был придуман компанией Microsoft для разработки приложений с графическим интерфейсом под Windows.

Концепцию этого паттерна можно разобрать на примере приложения со списком задач. Для этого создайте WPF-проект и добавьте в него следующий класс (он будет служить моделью):

public class Customer : INotifyPropertyChanged //Подключается интерфейс, который позволяет уведомлять об изменении состояния

{

//Данные о задаче

private string name;

private string task;

private int price;

private DateTime deadline;

private bool isSolved;

public Customer(string name, string task, int price, DateTime deadline) //Простой конструктор

{

this.name = name;

this.task = task;

this.price = price;

this.deadline = deadline;

this.isSolved = false;

}

//Геттеры и сеттеры

public string Name

{

get

{

return this.name;

}

}

public string Task

{

get

{

return this.task;

}

}

public int Price

{

get

{

return this.price;

}

}

public string DeadlineString

{

get

{

return $"{this.deadline.Day}.{this.deadline.Month}.{this.deadline.Year}";

}

}

public bool IsSolved

{

get

{

return this.isSolved;

}

set

{

this.isSolved = value;

OnPropertyChanged("IsSolved"); //Если свойство меняется, вызывается метод, который уведомляет об изменении модели

OnPropertyChanged("Color"); //Если изменено несколько значений, можно вызвать дополнительный метод

}

}

public string Color

{

get

{ //Если задача решена, будет возвращён синий цвет, иначе он будет зависеть от того, прошёл ли дедлайн

return this.isSolved ? "Blue" : DateTime.Now.CompareTo(this.deadline) == -1 ? "Black" : "Red";

}

}

public event PropertyChangedEventHandler PropertyChanged; //Событие, которое будет вызвано при изменении модели

public void OnPropertyChanged([CallerMemberName]string prop = "") //Метод, который скажет ViewModel, что нужно передать виду новые данные

{

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs(prop));

}

}

Теперь создайте модель представления:

public class AppViewModel : INotifyPropertyChanged

{

private Customer selectedCustomer;

private ObservableCollection<Customer> customers;

public AppViewModel()

{

customers = new ObservableCollection<Customer>() //Добавление данных для тестирования

{

new Customer("Josh", "Fix printer", 500, new DateTime(2019, 5, 11)),

new Customer("Josh", "Install fax", 350, new DateTime(2019, 6, 15)),

new Customer("Tyler", "Update soft", 100, new DateTime(2019, 6, 17)),

new Customer("Nico", "Install antivirus", 400, new DateTime(2019, 6, 19)),

new Customer("Tyler", "Fix printer", 500, new DateTime(2019, 6, 21)),

new Customer("Nico", "Update soft", 200, new DateTime(2019, 6, 27))

};

}

public Customer SelectedCustomer

{

get

{

return this.selectedCustomer;

}

set

{

this.selectedCustomer = value;

OnPropertyChanged("SelectedCustomer");

}

}

public ObservableCollection<Customer> Customers

{

get

{

return this.customers;

}

}

public event PropertyChangedEventHandler PropertyChanged;

public void OnPropertyChanged([CallerMemberName]string prop = "")

{

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs(prop));

}

}

Экземпляр этого класса и будет использоваться для обмена данными и командами между моделью и видом. Затем нужно создать View — интерфейс программы:

<Window.Resources>

<Style x:Key="Text">

<Setter Property="TextBlock.Margin" Value="5"/>

<Setter Property="TextBlock.FontSize" Value="14"/>

</Style>

<Style BasedOn="{StaticResource Text}" TargetType="TextBlock"></Style>

<Style x:Key="BoldText" BasedOn="{StaticResource Text}">

<Setter Property="TextBlock.FontWeight" Value="DemiBold"/>

</Style>

</Window.Resources>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="1\*"/>

<ColumnDefinition Width="1\*"/>

</Grid.ColumnDefinitions>

<Border Padding="5">

<ListBox ItemsSource="{Binding Customers}" SelectedItem="{Binding SelectedCustomer}" ScrollViewer.VerticalScrollBarVisibility="Auto">

<ListBox.ItemTemplate>

<DataTemplate>

<StackPanel>

<TextBlock Text="{Binding Name}" FontSize="16" FontWeight="DemiBold" Background="{x:Null}"/>

<TextBlock Text="{Binding Task}"/>

<TextBlock Text="{Binding DeadlineString}" TextAlignment="Right" Foreground="{Binding Color}"/>

</StackPanel>

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>

</Border>

<Border Grid.Column="1" Margin="5">

<StackPanel DataContext="{Binding SelectedCustomer}">

<TextBlock Text="Customer" TextAlignment="Center" FontSize="16" Style="{StaticResource BoldText}"/>

<DockPanel>

<TextBlock Text="Name: " Style="{StaticResource BoldText}"/>

<TextBlock Text="{Binding Name}"/>

</DockPanel>

<DockPanel>

<TextBlock Text="Task: " Style="{StaticResource BoldText}"/>

<TextBlock Text="{Binding Task}"/>

</DockPanel>

<DockPanel>

<TextBlock Text="Deadline: " Style="{StaticResource BoldText}"/>

<TextBlock Text="{Binding DeadlineString}"/>

</DockPanel>

<DockPanel>

<TextBlock Text="Solved: " Style="{StaticResource BoldText}"/>

<CheckBox VerticalAlignment="Center" IsChecked="{Binding IsSolved}"/>

</DockPanel>

</StackPanel>

</Border>

</Grid>

В этом коде можно заменить очень важную часть любого MVVM-приложения — привязку данных. Вместо того чтобы указывать в интерфейсе какие-то значения вручную, они добавляются автоматически благодаря подобному аргументу:

{Binding Task}

Он говорит программе, что нужно получить (привязать) данные из свойства Task. При этом у самого списка в атрибуте ItemsSource указано следующее:

{Binding Customers}

То есть он получает коллекцию Customers и выводит её элементы согласно шаблону DataTemplate. И если какой-нибудь элемент изменится, то это сразу же отобразится в приложении, благодаря вызову метода OnPropertyChanged().

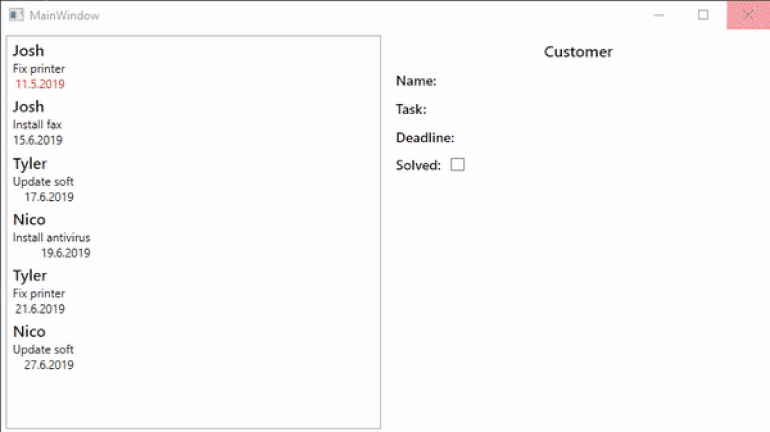
Однако чтобы использовать привязку, нужно сначала сообщить приложению, откуда брать данные. Для этого в файле MainWindow.xaml.cs добавьте следующую строку:

DataContext = new AppViewModel();

Кроме этого ничего добавлять не нужно — приложение будет функционировать без обработчиков событий.

Например, если пользователь нажмёт на CheckBox, то значение сразу отправится в модель. То есть в этом случае используется двусторонняя привязка данных: она не только передаёт в вид значение из модели, но и уведомляет модель, что что-то изменилось.

Вот как выглядит готовое приложение:



**Порядок выполнения работы**

1. Изучите теоретические сведения.

2. Реализовать программу использующую MVVM по теме индивидуального задания

Преподаватель \_\_\_\_\_\_\_\_ А.С. Комаровский

Рассмотрено на заседании цикловой

комиссии ПОИТ №10

Протокол № от «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2021

Председатель ЦК \_\_\_\_\_\_\_\_ В.Ю. Михалевич