Міністерство освіти і науки, молоді та спорту України

Кременчуцький національний університет

Імені Михайла Остроградського

Кафедра ІУС

**Лабораторна робота №4**

З дисципліни

«Об’єктно-орієнтоване програмування»

Виконав: Перевірила:

Студент групи КН-15-1 ст. вкл. Кафедри ІУС

Сіянко Є.Р. Щербак І.М.

Разработка приложений с использованием технологии WPF

1.VievModelBase

using System.ComponentModel;

namespace Wpflab4.ViewModel

{

public abstract class ViewModelBase : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;// = delegate { };

protected void OnPropertyChanged(string propertyName)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

}

}

2.ThingToBuy (ViewModel)

using Wpflab4.Model;

namespace Wpflab4.ViewModel

{

public class ThingToBuyViewModel : ViewModelBase

{

ThingToBuy \_thing;

public ThingToBuyViewModel(ThingToBuy thing)

{

\_thing = thing;

}

public string Name

{

get { return \_thing.Name; }

set

{

\_thing.Name = value;

OnPropertyChanged("Name");

}

}

public int Amount

{

get { return \_thing.Amount; }

set

{

\_thing.Amount = value;

OnPropertyChanged("Amount");

}

}

}

}

3.MainWindowView (ViewModel)

using System;

using System.Collections;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Windows;

using System.Windows.Input;

using Wpflab4.Model;

using Wpflab4.View;

namespace Wpflab4.ViewModel

{

public class MainWindowViewModel : ViewModelBase

{

public MainWindowViewModel(Things things)

{

\_things = things;

ThingsList = \_things.ThingsList;

Collection = new ObservableCollection<ThingToBuy>(ThingsList);

SaveCommand = new Command(arg => Save());

AddCommand = new Command(arg => Add());

OpenCommand = new Command(arg => Open());

DeleteCommand = new Command(arg => Delete());

CloseCommand = new Command(arg => Close?.Invoke());

}

public ICommand SaveCommand { get; set; }

public ICommand OpenCommand { get; set; }

public ICommand AddCommand { get; set; }

public ICommand DeleteCommand { get; set; }

public ICommand CloseCommand { get; set; }

public ICommand AboutCommand { get; set; }

public event Action Close;

private string \_searchQuery;

public string SearchQuery

{

get { return \_searchQuery; }

set

{

\_searchQuery = value;

Search();

}

}

public ObservableCollection<ThingToBuy> \_collection;

public ObservableCollection<ThingToBuy> Collection

{

get { return \_collection; }

set { \_collection = value; OnPropertyChanged("Collection"); }

}

private string \_status;

public string Status

{

get { return \_status; }

set

{

\_status = value;

OnPropertyChanged("Status");

}

}

private Things \_things;

private List<ThingToBuy> \_thingsList;

public List<ThingToBuy> ThingsList

{

get

{

return \_thingsList;

}

set

{

\_thingsList = value;

OnPropertyChanged(nameof(ThingsList));

}

}

private ThingToBuy \_selected;

public ThingToBuy Selected

{

get

{

return \_selected;

}

set

{

\_selected = value;

OnPropertyChanged(nameof(Selected));

}

}

4. Commands

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Input;

namespace Wpflab4.ViewModel

{

class Command : ICommand

{

#region Constructor

public Command(Action<object> action)

{

ExecuteDelegate = action;

}

#endregion

#region Properties

public Predicate<object> CanExecuteDelegate { get; set; }

public Action<object> ExecuteDelegate { get; set; }

#endregion

#region ICommand members

public bool CanExecute(object parameter)

{

if (CanExecuteDelegate != null)

return CanExecuteDelegate(parameter);

return true;

}

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

public void Execute(object parameter)

{

ExecuteDelegate?.Invoke(parameter);

}

#endregion

}

}

5.AddThingsViewModel

using System;

using System.Windows;

using System.Windows.Input;

using Wpflab4.Model;

namespace Wpflab4.ViewModel

{

public class AddThingViewModel : ViewModelBase

{

public string Name { get; set; }

public string Amount { get; set; }

Things \_thingsList;

public AddThingViewModel(Things thingsList)

{

\_thingsList = thingsList;

AddCommand = new Command(arg => Add());

}

public ICommand AddCommand { get; set; }

public ICommand CloseCommand { get; set; }

public event Action<ThingToBuy> Generated;

void Add()

{

try

{

Generated?.Invoke(new ThingToBuy(this.Name, Convert.ToInt32(this.Amount)));

if (CloseCommand != null)

CloseCommand.Execute(null);

}

catch (Exception)

{

MessageBox.Show("Ошибка!");

}

}

}

6.MainWindow.xaml

<Window x:Class="Wpflab4.View.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:Wpflab4"

mc:Ignorable="d"

Title="Список покупок" Height="350" Width="525" MinWidth="525" MinHeight="350" WindowStartupLocation="CenterScreen">

<Grid Name="grid">

<Grid.RowDefinitions>

<RowDefinition Height="25"></RowDefinition>

<RowDefinition Height="25"/>

<RowDefinition Height="216\*"/>

<RowDefinition Height="53\*"></RowDefinition>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition></ColumnDefinition>

</Grid.ColumnDefinitions>

<StackPanel Grid.Row="1" Grid.Column="0" Orientation="Horizontal" Width="{Binding ElementName=grid}">

<Label VerticalAlignment="Center" HorizontalAlignment="Left" Content="\_Поиск :" Target="{Binding ElementName=SearchBox}"/>

<TextBox x:Name="SearchBox" HorizontalContentAlignment="Right" Width="150" HorizontalAlignment="Left" VerticalAlignment="Center" Text="{Binding SearchQuery, Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"></TextBox>

</StackPanel>

<DataGrid Name="dataGridMain" Grid.Column="0" Grid.Row="2" ItemsSource="{Binding Collection, Mode=TwoWay, NotifyOnSourceUpdated=True, UpdateSourceTrigger=PropertyChanged}" SelectedItem="{Binding Selected}" SelectionMode="Single" AutoGenerateColumns="False" IsReadOnly="False" CanUserAddRows="False" Margin="0,1,0,0">

<DataGrid.Columns>

<DataGridTextColumn Binding="{Binding Name, Mode=TwoWay}" MinWidth="300" Header="Что купить"/>

<DataGridTextColumn Binding="{Binding Amount, Mode=TwoWay}" MinWidth="165" Header="Количество"/>

</DataGrid.Columns>

</DataGrid>

<StackPanel Margin="0,5,5,5" MaxHeight="30" Orientation="Horizontal" HorizontalAlignment="Right" Grid.Column="0" Grid.Row="3" Width="336" >

<Button Margin="2" Height="25" Width="80" Command="{Binding AddCommand}">Добавить</Button>

<Button Margin="2" Height="25" Width="80" Command="{Binding DeleteCommand}">Удалить</Button>

<Button Margin="2" Height="25" Width="80" Command="{Binding OpenCommand}">Открыть</Button>

<Button Margin="2" Height="25" Width="80" Command="{Binding SaveCommand}">Сохранить</Button>

</StackPanel>

<Menu Height="25" VerticalAlignment="Top">

<MenuItem Header="Файл" Height="25">

<MenuItem Command="{Binding CloseCommand}" Header="Выход"/>

</MenuItem>

<Separator Width="1"/>

</Menu>

<TextBlock Grid.Column="0" Grid.Row="3" Text="{Binding Status, Mode=OneWay, UpdateSourceTrigger=PropertyChanged}" VerticalAlignment="Bottom" HorizontalAlignment="Left" Width="150" Height="25" Margin="5,0,0,5"/>

</Grid>

</Window>

7.AddThing.xaml

<Window x:Class="Wpflab4.View.AddThingView"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:Wpflab4.View"

mc:Ignorable="d"

Title="AddThingView" Height="170" Width="200" MinWidth="200" MinHeight="170" MaxWidth="200" MaxHeight="170" WindowStartupLocation="CenterScreen">

<Grid>

<StackPanel>

<Label Content="Что купить:"/>

<TextBox Margin="5,0" Text="{Binding Name, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>

<Label Content="Количество:"/>

<TextBox Margin="5,0" Text="{Binding Amount, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>

<WrapPanel>

<Button Height="30" Width="75" Margin="10,10" Content="Добавить" Command="{Binding AddCommand}" IsDefault="True"/>

<Button Height="30" Width="75" Margin="0,0" Content="Отмена" IsCancel="True" Command="{Binding CloseCommand}"/>

</WrapPanel>

</StackPanel>

</Grid>

</Window>

8. Things (Model)

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Serialization;

namespace Wpflab4.Model

{

public class Things

{

public List<ThingToBuy> ThingsList { get; set; }

public Things()

{

ThingsList = new List<ThingToBuy>();

}

public List<ThingToBuy> GetAll()

{

return ThingsList;

}

public void Save(List<ThingToBuy> things)

{

XmlSerializer formatter = new XmlSerializer(typeof(List<ThingToBuy>));

using (FileStream fs = new FileStream("things.xml", FileMode.OpenOrCreate))

{

formatter.Serialize(fs, things);

}

}

public List<ThingToBuy> Load()

{

using (FileStream fs = new FileStream("things.xml", FileMode.OpenOrCreate))

{

XmlSerializer formatter = new XmlSerializer(typeof(List<ThingToBuy>));

return (List<ThingToBuy>)formatter.Deserialize(fs);

}

}

public void Add(ThingToBuy thing)

{

ThingsList.Add(thing);

}

public void Add(List<ThingToBuy> things)

{

ThingsList.AddRange(things);

}

public void Delete(ThingToBuy thing)

{

if (ThingsList.Contains(thing))

ThingsList.Remove(thing);

else

throw new Exception("Not found.");

}

}

}

9. ThingToBuy (Model)

using System;

namespace Wpflab4.Model

{

[Serializable]

public class ThingToBuy

{

public ThingToBuy()

{

}

public ThingToBuy(string name, int amount)

{

Name = name;

Amount = amount;

}

public string Name { get; set; }

public int Amount { get; set; }

}

}

Контрольные вопросы:

1. Что такое WPF?

система для построения клиентских приложений [Windows](https://ru.wikipedia.org/wiki/Windows) с визуально привлекательными возможностями взаимодействия с пользователем, графическая (презентационная) подсистема в составе [.NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework) (начиная с версии [3.0](https://ru.wikipedia.org/wiki/.NET_Framework_3.0)), использующая язык [XAML](https://ru.wikipedia.org/wiki/XAML).

2. Что такое XAML?

расширяемый [язык разметки](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D1%80%D0%B0%D0%B7%D0%BC%D0%B5%D1%82%D0%BA%D0%B8) для приложений (произносится [замл] или [зэмл]) — основанный на [XML](https://ru.wikipedia.org/wiki/XML) язык разметки для декларативного программирования приложений, разработанный [Microsoft](https://ru.wikipedia.org/wiki/Microsoft).

3. Что такое App.xaml в WPF проекте?

С помощью атрибута x:Class элемент Application задает полное название производного класса приложения. По умолчанию класс называется App, с указанием названия проекта, то есть в данном случае LifecycleApp.App

Как правило, основная задача данного файла состоит в определении ресурсов, общих для приложения. Поэтому тут по умолчанию определен пустой элемент Application.Resources, в который, собственно, и помещаются ресурсы.

Также здесь декларативным путем можно прикрепить к событиям приложения обработчики.

4. Элементы управления WPF

**Элементы управления содержимым**, например кнопки (Button), метки (Label)

**Специальные контейнеры**, которые содержат другие элементы, но в отличие от элементов Grid или Canvas не являются контейнерами компоновки - ScrollViewer,GroupBox

**Декораторы**, чье предназначение создание определенного фона вокруг вложенных элементов, например, Border или Viewbox.

**Элементы управления списками**, например, ListBox, ComboBox.

**Текстовые элементы управления**, например, TextBox, RichTextBox.

**Элементы, основанные на диапазонах значений**, например, ProgressBar, Slider.

**Элементы для работ с датами**, например, DatePicker и Calendar.

**Остальные элементы управления**, которые не вошли в предыдущие подгруппы, например, Image.

5. Компоновка. Контейнеры компоновки

Благодаря компоновке мы можем удобным нам образом настроить элементы интерфейса, позиционировать их определенным образом. Например, элементы компоновки в WPF позволяют при ресайзе - сжатии или растяжении масштабировать элементы, что очень удобно, а визуально не создает всяких шероховатостей типа незаполненных пустот на форме.

**Контейнеры компоновки** позволяют эффективно распределить доступное пространство между элементами, найти для него наиболее предпочтительные размеры.

6. Предназначение привязки и команд

Привязка команд представляет объект CommandBinding. Его событие Executed прикрепляет обработчик, который будет выполняться при вызове команды. А свойство Command уставливает саму команду, к которой относится обработчик.

7. Виды привязки

Режимы привязки

**Свойство Mode** объекта Binding, которое представляет режим привязки, может принимать следующие значения:

**OneWay:** свойство объекта-приемника изменяется после модификации свойства объекта-источника.

**OneTime**: свойство объекта-приемника устанавливается по свойству объекта-источника только один раз. В дальнейшем изменения в источнике никак не влияют на объект-приемник.

**TwoWay:** оба объекта - применки и источник могут изменять привязанные свойства друг друга.

**OneWayToSource:** объект-приемник, в котором объявлена привязка, меняет объект-источник.

**Default:** по умолчанию (если меняется свойство TextBox.Text, то имеет значение TwoWay, в остальных случаях OneWay).

8. Предназначение интерфейса “INotifyPropertyChanged”

Cколько бы мы не нажимали на кнопку, текстовые блоки, привязанные к ресурсу, не изменятся. Чтобы объект мог полноценно реализовать механизм привязки, нам надо реализовать в его классе интерфейс INotifyPropertyChanged.

Когда объект класса изменяет значение свойства, то он через событие PropertyChanged извещает систему об изменении свойства. А система обновляет все привязанные объекты.

9. Что такое MVVM?

Паттерн MVVM (Model-View-ViewModel) позволяет отделить логику приложения от визуальной части (представления). Данный паттерн является архитектурным, то есть он задает общую архитектуру приложения.

MVVM состоит из трех компонентов: модели (Model), модели представления (ViewModel) и представления (View).

10. Преимущества MVVM

Тестируемость MVVM-приложений

Приложения, разработанные с использованием MVVM, обладают очень хорошим основанием для проведения модульного тестирования с целью проверки работы отдельных классов и методов.

Меньшее количество кода

Объем кода, необходимого для управления представлением немного снижается при использовании MVVM, а это означает, что снижается риск допустить ошибки и уменьшается код для написания модульных тестов.

Улучшенное проектирование приложений

Разработчики и дизайнеры могут самостоятельно работать над разными частями приложения. Как вы видели на примере, представление генерируется в XAML-разметке и использует базовый синтаксис привязок и команд, для взаимодействия с модель-представлением.

Легкость понимания логики представления

MVVM предусматривает хорошо организованную и легкую для понимания конструкцию построения графического интерфейса за счет использования механизмов привязок, команд и шаблонов данных.