**26.** **Язык разметки XAML**

Задание №1

В каждом задании создать XML файл состоящий минимум из 7 элементов. WPF по примеру(см. Пример Задания 1.docx).

Дополнительно реализовать Форму добавления записи в XML Файл. Добавить прокрутку элемента TextBlock для вывода информации. Проверку доступности кнопок если XML файл не выбран.

Создать XML файл Автостоянка состоящий из автомобилей (марка, год выпуска, срок аренды стоянки(дни) ).

Листинг программы:

MainWindow.xaml

<Window x:Class="z1.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:z1"

mc:Ignorable="d"

Title="MainWindow" Height="450" Width="800">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="3\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="auto"/>

<RowDefinition Height="auto"/>

<RowDefinition />

</Grid.RowDefinitions>

<StackPanel Grid.Column="0" Grid.Row="0">

<Button x:Name="buttonOpenFile"

Content="Открыть файл"

Command="{Binding Open}"/>

</StackPanel>

<StackPanel Grid.Column="0"

Grid.Row="1">

<StackPanel Margin="10,5">

<Label Content="Марка:"/>

<TextBox Text="{Binding Stamp, UpdateSourceTrigger=PropertyChanged}"/>

</StackPanel>

<StackPanel Margin="10,5">

<Label Content="Год выпуска:"/>

<TextBox Text="{Binding YearRelease, UpdateSourceTrigger=PropertyChanged}"/>

</StackPanel>

<StackPanel Margin="10,5">

<Label Content="Срок аренды стоянки(дни):"/>

<TextBox Text="{Binding ParkingRentalPeriod, UpdateSourceTrigger=PropertyChanged}"

MinWidth="150"/>

</StackPanel>

<Button Content="Добавить"

Command="{Binding Add}"/>

</StackPanel>

<StackPanel Grid.Column="0"

Grid.Row="2">

<StackPanel Margin="10,5">

<Label Content="Id:"/>

<TextBox x:Name="textBoxId" MinWidth="150"/>

</StackPanel>

<Button Content="Удалить"

Command="{Binding Delete}"

CommandParameter="{Binding ElementName=textBoxId, Path=Text}"/>

</StackPanel>

<TextBox Grid.Column="1"

Grid.Row="0"

Grid.RowSpan="3"

AcceptsReturn="True"

Text="{Binding ReadData}"/>

</Grid>

</Window>

MainWindow.xaml.cs

using System.Windows;

using z1.ViewModels;

namespace z1

{

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

DataContext = new MainWindowsViewModel();

}

}

}

XmlDocumentWorker

using System.Collections.Generic;

using z1.Interfaces;

using z1.Models;

using System.Xml.Linq;

using System.Linq;

using System;

namespace z1.Share

{

internal class XmlDocumentWorker : IXmlWorker<Auto>

{

private int \_maxId;

private XDocument \_doc;

private string \_pathDoc;

public XDocument Doc

{

get => \_doc;

}

public void Load(string path)

{

\_pathDoc = path;

\_doc = XDocument.Load(\_pathDoc);

}

public void Add(Auto obj)

{

XElement? root = \_doc.Element("autos");

if (root != null)

{

root.Add(new XElement("auto",

new XAttribute("id", $"{++\_maxId}"),

new XAttribute("stamp", $"{obj.Stamp}"),

new XAttribute("yearRelease", $"{obj.YearRelease}"),

new XAttribute("parkingRentalPeriod", $"{obj.ParkingRentalPeriod}")));

}

\_doc.Save(\_pathDoc);

}

public void Delete(int id)

{

XElement? root = \_doc.Element("autos");

if (root != null)

{

var auto = root.Elements("auto")

.FirstOrDefault(a => a.Attribute("id")?.Value == id.ToString());

if(auto != null)

{

auto.Remove();

\_doc.Save(\_pathDoc);

}

}

}

public Auto DindBy(string id)

{

XElement? root = \_doc.Element("autos");

if (root != null)

{

var auto = root.Elements("auto")

.FirstOrDefault(a => a.Attribute("id")?.Value == id.ToString());

if (auto != null)

{

return GetAuto(auto);

}

}

return null;

}

public List<Auto> GetAll()

{

var elems = \_doc.Element("autos")?

.Elements("auto").ToList();

List<Auto> autos = new List<Auto>();

foreach(var e in elems)

{

Auto a = GetAuto(e);

if(\_maxId < a.ID)

{

\_maxId = a.ID;

}

autos.Add(a);

}

return autos;

}

private Auto GetAuto(XElement xAuto)

{

try

{

int id = Convert.ToInt32(xAuto.Attribute("id").Value);

string stamp = xAuto.Attribute("stamp").Value;

int yearRelease = Convert.ToInt32(xAuto.Attribute("yearRelease").Value);

int parkingRentalPeriod = Convert.ToInt32(xAuto.Attribute("parkingRentalPeriod").Value);

return new Auto(id, stamp, yearRelease, parkingRentalPeriod);

}

catch(Exception ex) when (ex is FormatException ||

ex is NullReferenceException)

{

throw new Exception(ex.Message);

}

}

}

}

IXmlWorker

using System.Collections.Generic;

namespace z1.Interfaces

{

internal interface IXmlWorker<T>

{

public void Load(string path);

public void Add(T obj);

public void Delete(int id);

public T DindBy(string id);

public List<T> GetAll();

}

}

Auto

using System.ComponentModel;

using System.Runtime.CompilerServices;

namespace z1.Models

{

internal class Auto : INotifyPropertyChanged

{

private int \_id;

private string \_stamp;

private int \_yearRelease;

private int \_parkingRentalPeriod;

public event PropertyChangedEventHandler? PropertyChanged;

public int ID

{

get => \_id;

set

{

\_id = value;

OnPropertyChanged("ID");

}

}

public string Stamp

{

get => \_stamp;

set

{

\_stamp = value;

OnPropertyChanged("Stamp");

}

}

public int YearRelease

{

get => \_yearRelease;

set

{

\_yearRelease = value;

OnPropertyChanged("YearRelease");

}

}

public int ParkingRentalPeriod

{

get => \_parkingRentalPeriod;

set

{

\_parkingRentalPeriod = value;

OnPropertyChanged("ParkingRentalPeriod");

}

}

public Auto(int id, string stamp,

int yearRelease, int parkingRentalPeriod)

{

\_id = id;

\_stamp = stamp;

\_yearRelease = yearRelease;

\_parkingRentalPeriod = parkingRentalPeriod;

}

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

{

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs(prop));

}

public override string ToString()

{

return $"===============================================================\n" +

$"ID: {\_id}\n"+

$"Марка: {\_stamp}\n" +

$"Год выпуска: {\_yearRelease}\n" +

$"Срок аренды стоянки(дни): {\_parkingRentalPeriod}\n" +

$"===============================================================\n" +

$"\n";

}

}

}

ParkingLot

using System.Collections.Generic;

namespace z1.Models

{

internal class ParkingLot

{

List<Auto> \_autos = new List<Auto>();

public List<Auto> Autos

{

get => \_autos;

set => \_autos = value;

}

public Auto this[int i]

{

get => \_autos[i];

set => \_autos[i] = value;

}

}

}

using Microsoft.Win32;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Runtime.CompilerServices;

using System.Windows;

using z1.Commands;

using z1.Models;

using z1.Share;

MainWindowsViewModel

namespace z1.ViewModels

{

internal class MainWindowsViewModel : INotifyPropertyChanged

{

private string \_path;

private XmlDocumentWorker \_xmlDocWorker;

private List<Auto> \_autos;

private string \_stamp;

public string Stamp

{

get { return \_stamp; }

set

{

\_stamp = value;

OnPropertyChanged("Stamp");

}

}

private int \_yearRelease;

public string YearRelease

{

get { return \_yearRelease.ToString(); }

set

{

try

{

\_yearRelease = Convert.ToInt32(value);

OnPropertyChanged("YearRelease");

}

catch (FormatException)

{

}

}

}

private int \_parkingRentalPeriod;

public string ParkingRentalPeriod

{

get { return \_parkingRentalPeriod.ToString(); }

set

{

try

{

\_parkingRentalPeriod = Convert.ToInt32(value);

OnPropertyChanged("ParkingRentalPeriod");

}

catch (FormatException)

{

}

}

}

private Command \_open;

public Command Open

{

get

{

return \_open ??

(\_open = new Command(obj => OpenFile()));

}

}

private Command \_add;

public Command Add

{

get

{

return \_add ??

(\_add = new Command(obj =>

{

AddAuto(\_stamp, \_yearRelease, \_parkingRentalPeriod);

}));

}

}

private Command \_delete;

public Command Delete

{

get

{

return \_delete ??

(\_delete = new Command(obj =>

{

string id = obj as string;

if (id != null)

{

DeleteAuto(id);

}

}));

}

}

private string \_readData;

public string ReadData

{

get

{

return \_readData;

}

set

{

\_readData = value;

OnPropertyChanged("ReadData");

}

}

public MainWindowsViewModel()

{

\_xmlDocWorker = new XmlDocumentWorker();

}

private void OpenFile()

{

OpenFileDialog openFileDialog = new OpenFileDialog();

if (openFileDialog.ShowDialog() == true)

{

ReadData = string.Empty;

\_path = openFileDialog.FileName;

\_xmlDocWorker.Load(\_path);

\_autos = \_xmlDocWorker.GetAll();

ReadData = GetText();

}

}

private void AddAuto(string stamp,

int yearRelease, int parkingRentalPeriod)

{

if (\_xmlDocWorker.Doc != null)

{

Auto auto = new Auto(0, stamp, yearRelease,

parkingRentalPeriod);

\_xmlDocWorker.Add(auto);

\_autos = \_xmlDocWorker.GetAll();

ReadData = GetText();

}

else

{

MessageBox.Show(

"Для начало откройте файл",

"Файл",

MessageBoxButton.OK);

}

}

private void DeleteAuto(string id)

{

if (\_xmlDocWorker.Doc != null)

{

try

{

int idInt = Convert.ToInt32(id);

\_xmlDocWorker.Delete(idInt);

\_autos = \_xmlDocWorker.GetAll();

ReadData = GetText();

}

catch (FormatException)

{

MessageBox.Show(

"Неправельный формат поля",

"Формат",

MessageBoxButton.OK);

}

}

else

{

MessageBox.Show(

"Для начало откройте файл",

"Файл",

MessageBoxButton.OK);

}

}

private string GetText()

{

if (\_autos != null)

{

if (\_autos.Count != 0)

{

\_readData = string.Empty;

foreach (Auto a in \_autos)

{

\_readData += a.ToString();

}

return \_readData;

}

}

return string.Empty;

}

public event PropertyChangedEventHandler? PropertyChanged;

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

{

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs(prop));

}

}

}

Command

using System;

using System.Windows.Input;

namespace z1.Commands

{

internal class Command : ICommand

{

private Action<object> execute;

private Func<object, bool> canExecute;

public event EventHandler? CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

public Command(Action<object> execute, Func<object, bool> canExecute = null)

{

this.execute = execute;

this.canExecute = canExecute;

}

public bool CanExecute(object? parameter)

{

return this.canExecute == null || this.canExecute(parameter);

}

public void Execute(object? parameter)

{

this.execute(parameter);

}

}

}

Анализ результатов:

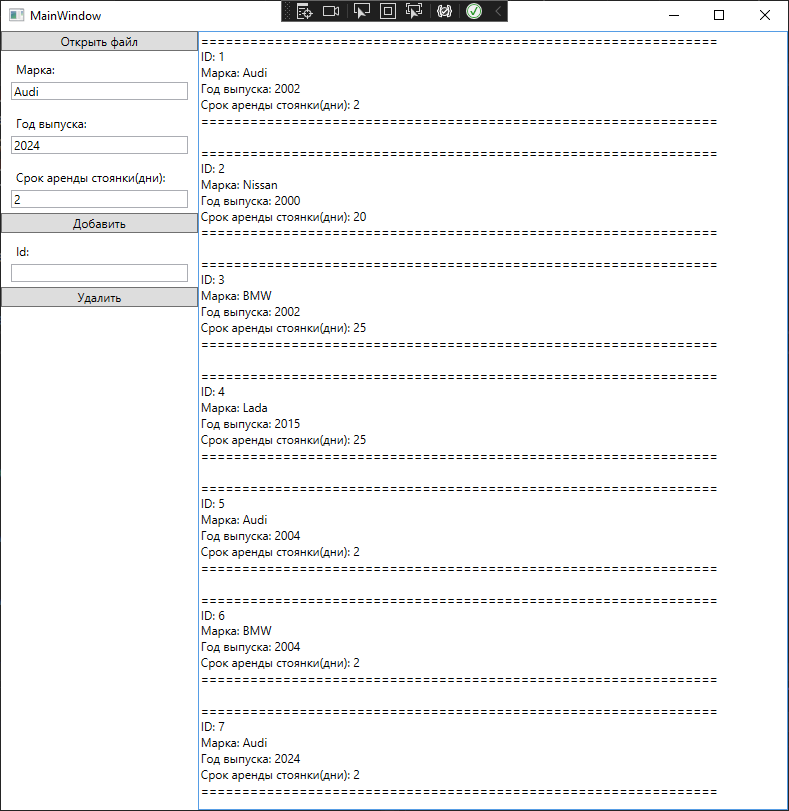


Рисунок 26.1 – Результат выполнения программы задание №1