

Факультет №3 «Системы управления, информатика и электроэнергетика»

Кафедра 304 «Вычислительные комплексы, системы и сети»

Лабораторная работа №2 по дисциплине «Технологии программирования»

«Динамические структуры данных»

Выполнили студенты

Группы 3О-308Б

Павлов

Изотов

Приняли

Барчев Николай Борисович

Титов Юрий Павлович

Москва 2018 год

Оглавление

[Задание. 2](#_Toc515707180)

[Описание программного продукта. 4](#_Toc515707181)

[Особенности реализации. 5](#_Toc515707182)

[Реализация динамической структуры. 7](#_Toc515707183)

[Сохранение и загрузка структуры. 7](#_Toc515707184)

[Псевдокод. 8](#_Toc515707185)

[Описание программных единиц. 10](#_Toc515707186)

[Инструкция пользователя – примеры работы программы. 11](#_Toc515707187)

[Тесты. 13](#_Toc515707188)

[Содержимое тестов. 14](#_Toc515707189)

[Исходный код программы. 18](#_Toc515707190)

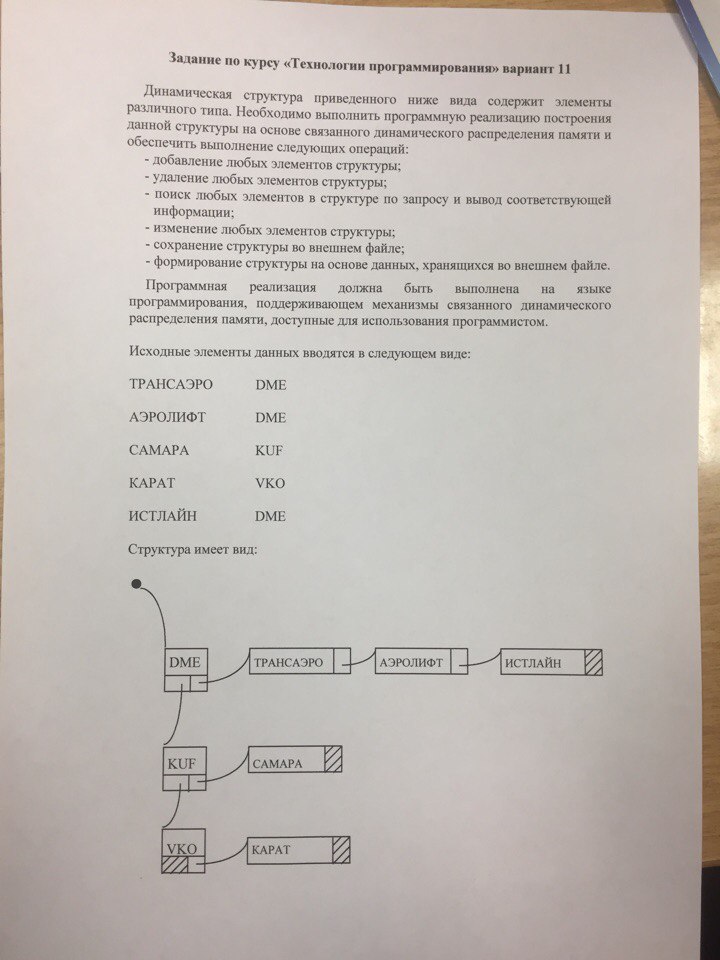
[Класс локатора/контейнера. 18](#_Toc515707191)

[Классы основной ViewModel и View. 19](#_Toc515707192)

[Классы модели – объектов предметной области (узел, список, узел для сериализации) 24](#_Toc515707193)

[Классы сервисов. 32](#_Toc515707194)

### Задание.



### Описание программного продукта.

Данное приложение реализовано на языке C# версии 6.0 в среде разработки Visual Studio 2015 и предназначено для использования под управлением ОС Windows.

Для запуска и работы приложения необходимо иметь на компьютере установленную среду .Net Framework 4.6.1.

В процессе работы приложение взаимодействует с внешними файлами в формате xml – пользователю предоставляется возможность самостоятельного выбора файлов, а также их местоположения в файловой системе ПК.

В программе предусмотрены все возможные варианты взаимодействия с пользователем, а также адекватная обработка некорректных действий работающего с ней человека (корректные действия обрабатываются также адекватно). Все это в совокупности с приятным графическим интерфейсом дает пользователю возможность легко пользоваться программой без необходимости читать объемную инструкцию.

### Особенности реализации.

Данное приложение реализовано с использованием архитектурного паттерна MVVM (**Model-View-ViewModel**) на платформе WPF .Net Framework.

Для упрощения решения задачи используются готовые библиотеки (пакеты) **MVVMLight**, **CommonServiceLocator**.

Библиотека **MVVMLight** содержит в себе реализации интерфейса ICommand, базовый класс ViewModel, базовый класс ObservableObject, реализующий интерфейс INotifyPropertyChanged и многое другое. Все это позволяет упростить реализацию паттерна MVVM, не прописывая вручную базовые классы/интерфейсы, необходимые для его реализации.

Библиотека **CommonServiceLocator** предоставляет простой в использовании IoC/DI контейнер (Inversion of Control, Dependency Injection, Service Locator).

**Inversion of Control** – принцип, согласно которому компоненты системы должны быть слабосвязанными/не должны зависеть от конкретных реализаций. Иными словами, объекты зависят от интерфейсов, а не от конкретных классов.

**Dependency Injection** – одна из реализаций предыдущего принципа. Его использование сводится к передаче объекту при его создании объектов, реализующих указанные в классе интерфейсы вместо конкретных классов. Таким образом класс может использовать любую реализацию интерфейса вместо одного конкретного класса.

Контейнер (реализация **Service Locator**) позволяет задать соответствие между интерфейсом и его конкретной реализацией, чтобы за пределами контейнера не заниматься созданием объектов нужных типов и передачей их в другие объекты для их создания.

Паттерн MVVM.

Model

Модель описывает используемые в приложении данные. Модели могут содержать логику, непосредственно связанную этими данными, например, логику валидации свойств модели. В то же время модель не должна содержать никакой логики, связанной с отображением данных и взаимодействием с визуальными элементами управления.

Нередко модель реализует интерфейсы INotifyPropertyChanged или INotifyCollectionChanged, которые позволяют уведомлять систему об изменениях свойств модели. Благодаря этому облегчается привязка к представлению, хотя опять же прямое взаимодействие между моделью и представлением отсутствует.

View

View или представление определяет визуальный интерфейс, через который пользователь взаимодействует с приложением. Применительно к WPF представление - это код в xaml, который определяет интерфейс в виде кнопок, текстовых полей и прочих визуальных элементов.

ViewModel

ViewModel или модель представления связывает модель и представление через механизм привязки данных. Если в модели изменяются значения свойств, при реализации моделью интерфейса INotifyPropertyChanged автоматически идет изменение отображаемых данных в представлении, хотя напрямую модель и представление не связаны.

ViewModel также содержит логику по получению данных из модели, которые потом передаются в представление. И также VewModel определяет логику по обновлению данных в модели.

Поскольку элементы представления, то есть визуальные компоненты типа кнопок, не используют события, то представление взаимодействует с ViewModel посредством команд.

### Реализация динамической структуры.

Для реализации в программе динамической структуры данных используются два класса – **Node** и **ObservableLinkedList**.

Класс **Node** описывает элемент двусвязного списка (узел) и его поведение. Каждый узел обладает значением (обычно имя), ссылкой на родительский элемент и списком дочерних узлов типа **ObservableLinkedList**. Также в узле описана логика добавления дочернего узла, удаления самого узла, поиска значения по текущему узлу и всем его дочерним узлам, а так же очистка поиска.

Класс **ObservableLinkedList** представляет собой двусвязный список элементов (узлов), специально модифицированный для использования в паттерне MVVM (а именно в нем добавлена реализация интерфейса **INotifyPropertyChanged** для уведомления других объектов об изменении своего состояния). В списке описаны методы добавления элемента в начало, в конце, после выбранного элемента или перед ним.

Сама динамическая структура представлена через корневой узел Root и список его дочерних элементов. Пользователь не работает напрямую с корневым элементом (кроме добавления/удаления его дочерних узлов), но видит его дочерние узлы, которые визуально будут отображаться в виде строк. Для каждой т.н. «строки» можно добавлять/удалять дочерние элементы, которые будут отображаться справа от него.

По существу, узлы первого (строки) и второго (столбцы) уровня ничем не отличаются друг от друга, что позволяет описать их одним классом.

### Сохранение и загрузка структуры.

Для сохранения в файл и загрузки структуры из файла используется механизм **XML** **сериализации**. **XML** сериализация популярна, востребована и проста в использовании – сохранение данных в файл или их загрузка из файла занимает всего несколько строк кода.

Для его реализации в программе используется класс **NodeDto**, который описывает узел списка, но имеет все необходимые для сериализации атрибуты, а также упрощает представление списка дочерних узлов, заменяя его массивом. В процессе сохранения/загрузки каждому узлу структуры ставится в соответствие экземпляр **NodeDto**, и именно экземпляры этого класса сохраняются в файл или загружаются из него. Такой подход позволяет отделить класс модели предметной области от деталей реализации сохранения/загрузки его содержимого в файл. Также такой подход (наряду с зависимостью от интерфейса **ISerializationService** и использованием контейнера в приложении) позволяет заменить механизм сериализации, не внося изменений в существующие модели предметной области. Достаточно будет создать новый сервис сериализации, зарегистрировать его в контейнере и пользоваться приложением.

### Псевдокод.

Точка входа ()

{

Инициализировать IoC-контейнер и зарегистрировать в нем типы, указанные в классе ViewModelLocator;

Создать MainViewModel для главного окна приложения;

Создать главное окно приложения;

Обрабатывать события главного окна приложения, пока оно не будет закрыто;

}

MainViewModel

{

Получить исходные данные для инициализации структуры из соответствующего сервиса;

Инициализировать команды сохранения/загрузки структуры;

}

Главное окно приложения

{

Если нажата кнопка «Поиск»

{

Выполнить поиск введенного значения среди элементов структуры;

}

Если нажата кнопка «Очистить»

{

Очистить поле поиска и подсвеченные элементы;

}

Если нажата кнопка «Сохранить в файл»

{

Открыть диалоговое окно выбора файла для сохранения структуры;

Сохранить структуру в выбранный файл формата xml;

}

Если нажата кнопка «Загрузить из файла»

{

Открыть диалоговое окно выбора файла для загрузки структуры;

Загрузить структуру из выбранного файла формата xml;

}

Если нажата кнопка «Добавить строку»

{

Добавить новый элемент «первого уровня» в структуру;

}

Если нажата кнопка «Удалить строку»

{

Удалить выбранный элемент «первого уровня» из структуры;

}

Если нажата кнопка «Удалить строку»

{

Удалить выбранный элемент «первого уровня» из структуры;

}

Если нажата кнопка «Добавить элемент»

{

Добавить новый дочерний элемент для выбранного элемента структуры;

}

Если нажата кнопка «Удалить»

{

Добавить новый дочерний элемент для выбранного элемента структуры;

}

}

### Описание программных единиц.

Файл **App.xaml и App.xaml.cs**

Точка входа приложения.

Файл **packages.config**

Конфигурационный файл сторонних пакетов (библиотек), используемых в приложении.

Файл **App.config**

Конфигурационный файл. приложения

Файл **MainView.xaml** и **MainView.xaml.cs**

Описание главного окна приложения.

Файл **MainViewModel.cs**

ViewModel для основного окна приложения.

Файл **IDataService.cs**

Интерфейс IDataService для сервиса получения исходных данных динамической структуры данных.

Файл **DataService.cs**

Класс DataService – основная реализация интерфейса **IDataService**.

Файл **DesignDataService.cs**

Класс DesignDataService – сервис получения исходных данных динамической структуры данных в дизайнере приложения.

Файл **ISerializationService.cs**

Интерфейс ISerializationService – интерфейс сервиса сериализации.

Файл **ObservableLinkedList**

Класс XmlSerializationService – класс сериализатора данных в xml.

Файл **Node.cs**

Класс Node – класс элемента двусвязного списка.

Файл **NodeDto.cs**

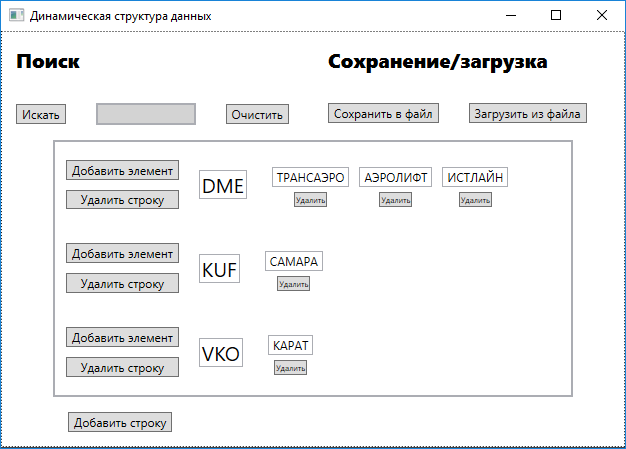
Класс NodeDto – класс для сериализации элемента двусвязного списка.

Файл **ObservableLinkedList.cs**

Класс ObservableLinkedList – класс двусвязного списка, поддерживающий.

### Инструкция пользователя – примеры работы программы.

При запуске программы открывается следующее окно:



Оно заполнено элементами “по умолчанию”.

Для добавления нового элемента (аэропорта в данном контексте данных, т.н. «первый уровень») следует нажать кнопку «Добавить строку».

Для удаления элемента «первого уровня» следует нажать кнопку «Удалить строку» для нужного элемента.

Для добавления нового дочернего элемента (т.н. «второй уровень») для выбранного элемента следует нажать кнопку «Добавить элемент» для нужного элемента.

Для удаления элемента «второго уровня» следует нажать кнопку «Удалит» под нужным элементом.

Для поиска элемента с нужным значением следует ввести его в поле ввода справа от кнопки «Искать» и нажать данную кнопку.

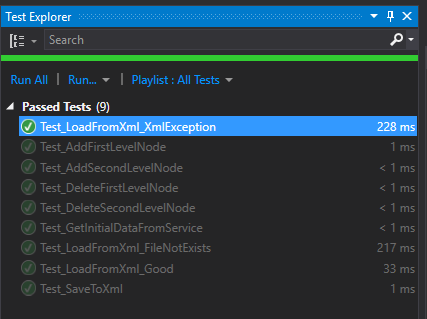
Для очистки поля поиска и подсвеченных (найденных) элементов следует нажать кнопку «Очистить».

Для сохранения текущей структуры в файл следует нажать кнопку «Сохранить в файл», выбрать желаемый файл в диалоговом окне выбора файла и нажать «Сохранить».

Для загрузки структуры из файла следует нажать кнопку «Загрузить из файла», выбрать желаемый файл в диалоговом окне выбора файла и нажать «Открыть».

### Тесты.

Для тестирования логики работы программы были написаны Unit-тесты с использованием встроенного в Visual Studio 2015 фреймворка для запуска тестов. Были написаны девять тестов, покрывающих основную логику работы программы, а именно: удаление/добавление элементов структуры, загрузка данных из файла, сохранение в файл, реакция на исключительные ситуации – отсутствие файла с данными, наличие некорректных данных в файле.



Код тестов представлен далее в отчете. Каждый тест содержит комментарий в начале, поясняя, что именно он тестирует в логике работы программы.

### Содержимое тестов.

using Microsoft.VisualStudio.TestTools.UnitTesting;

using LabaApp.Main;

using LabaApp.Model;

using LabaApp.Services;

using GalaSoft.MvvmLight.Ioc;

using System.IO;

using System;

namespace LabaUnitTests

{

[TestClass]

public class UnitTest1

{

private ViewModelLocator \_viewModelLocator;

private readonly string \_fileToSave;

private readonly string \_fileToLoad;

private readonly string \_fileToLoadException;

public UnitTest1()

{

\_viewModelLocator = new ViewModelLocator();

\_fileToSave = "save\_test.xml";

\_fileToLoad = "load\_test.xml";

\_fileToLoadException = "load\_test\_exception.xml";

}

/// <summary>

/// Тест загрузки структуры из файла - проверка отсутствия файла.

/// </summary>

[TestMethod]

public void Test\_LoadFromXml\_FileNotExists()

{

var serializationService = SimpleIoc.Default.GetInstance<ISerializationService<Node>>();

var viewModel = SimpleIoc.Default.GetInstance<MainViewModel>();

if (File.Exists(\_fileToLoadException))

File.Delete(\_fileToLoadException);

FileNotFoundException expectedException = new FileNotFoundException();

FileNotFoundException actualException = null;

Node dynamicStruct = null;

try

{

using (var stream = new FileStream(\_fileToLoadException, FileMode.Open, FileAccess.Read))

{

dynamicStruct = serializationService.Deserialize(stream);

}

}

catch (FileNotFoundException e)

{

actualException = e;

}

Assert.IsNull(dynamicStruct);

Assert.IsTrue(expectedException.GetType() == actualException.GetType());

}

/// <summary>

/// Тест загрузки структуры из файла - проверка на плохое содержимое файла.

/// </summary>

[TestMethod]

public void Test\_LoadFromXml\_XmlException()

{

var serializationService = SimpleIoc.Default.GetInstance<ISerializationService<Node>>();

var viewModel = SimpleIoc.Default.GetInstance<MainViewModel>();

File.WriteAllText(\_fileToLoadException, "not xml");

Node dynamicStruct = null;

using (var stream = new FileStream(\_fileToLoadException, FileMode.Open, FileAccess.Read))

{

dynamicStruct = serializationService.Deserialize(stream);

}

Assert.IsNull(dynamicStruct);

}

/// <summary>

/// Тест загрузки структуры из файла - проверка корректного сохранения и загрузки.

/// </summary>

[TestMethod]

public void Test\_LoadFromXml\_Good()

{

var serializationService = SimpleIoc.Default.GetInstance<ISerializationService<Node>>();

var viewModel = SimpleIoc.Default.GetInstance<MainViewModel>();

using (var stream = new FileStream(\_fileToSave, FileMode.Create, FileAccess.Write))

{

serializationService.Serialize(viewModel.Root, stream);

}

Node rootNodeLoaded;

using (var stream = new FileStream(\_fileToSave, FileMode.Open, FileAccess.Read))

{

rootNodeLoaded = serializationService.Deserialize(stream);

}

Assert.AreSame(viewModel.Root.Value, rootNodeLoaded.Value);

Assert.AreSame(viewModel.Root.Parent, rootNodeLoaded.Parent);

Assert.AreEqual(viewModel.Root.SubNodes.Count, rootNodeLoaded.SubNodes.Count);

}

/// <summary>

/// Тест загрузки структуры из файла - проверка корректного сохранения.

/// </summary>

[TestMethod]

public void Test\_SaveToXml()

{

var serializationService = SimpleIoc.Default.GetInstance<ISerializationService<Node>>();

var viewModel = SimpleIoc.Default.GetInstance<MainViewModel>();

Exception writeException = null;

using (var stream = new FileStream(\_fileToSave, FileMode.Create, FileAccess.Write))

{

try

{

serializationService.Serialize(viewModel.Root, stream);

}

catch (Exception e)

{

writeException = e;

}

}

Assert.IsNull(writeException);

}

/// <summary>

/// Тест загрузки структуры из сервиса - проверка непустоты полученных данных.

/// </summary>

[TestMethod]

public void Test\_GetInitialDataFromService()

{

var service = SimpleIoc.Default.GetInstance<IDataService>();

var data = service.GetData();

Assert.IsNotNull(data);

}

/// <summary>

/// Тест добавления узла первого уровня.

/// </summary>

[TestMethod]

public void Test\_AddFirstLevelNode()

{

var node = AddFirstLevelNode();

Assert.AreSame(node.Value, "Новый элемент");

Assert.IsTrue(node.SubNodes.Count == 0);

}

private Node AddFirstLevelNode()

{

var viewModel = SimpleIoc.Default.GetInstance<MainViewModel>();

viewModel.Root.AddNodeCommand.Execute(null);

return viewModel.Root.SubNodes.Last.Value;

}

/// <summary>

/// Тест удаления узла первого уровня.

/// </summary>

[TestMethod]

public void Test\_DeleteFirstLevelNode()

{

var viewModel = SimpleIoc.Default.GetInstance<MainViewModel>();

var root = viewModel.Root;

var rootSubNodes = root.SubNodes;

var rootSubNodesCountBefore = rootSubNodes.Count;

var addedNode = AddFirstLevelNode();

var isDeleted = root.SubNodes.Remove(addedNode);

var rootSubNodesCountAfter = rootSubNodes.Count;

Assert.AreEqual(rootSubNodesCountBefore, rootSubNodesCountAfter);

Assert.IsTrue(isDeleted);

}

/// <summary>

/// Тест добавления узла второго уровня.

/// </summary>

[TestMethod]

public void Test\_AddSecondLevelNode()

{

var node = AddSecondLevelNode();

Assert.AreSame(node.Value, "Новый элемент");

Assert.IsTrue(node.SubNodes.Count == 0);

}

private Node AddSecondLevelNode()

{

var viewModel = SimpleIoc.Default.GetInstance<MainViewModel>();

var root = viewModel.Root;

root.SubNodes.First.Value.AddNodeCommand.Execute(null);

return root.SubNodes.First.Value.SubNodes.Last.Value;

}

/// <summary>

/// Тест удаления узла второго уровня.

/// </summary>

[TestMethod]

public void Test\_DeleteSecondLevelNode()

{

var viewModel = SimpleIoc.Default.GetInstance<MainViewModel>();

var root = viewModel.Root;

var rootChildNode = root.SubNodes.First.Value;

var rootChildNodeSubNodes = rootChildNode.SubNodes;

var rootChildNodeSubNodesCountBefore = rootChildNode.SubNodes.Count;

var addedNode = AddSecondLevelNode();

var isDeleted = rootChildNode.SubNodes.Remove(addedNode);

var rootChildNodeSubNodesCountAfter = rootChildNode.SubNodes.Count;

Assert.AreEqual(rootChildNodeSubNodesCountBefore, rootChildNodeSubNodesCountAfter);

Assert.IsTrue(isDeleted);

}

}

}

### Исходный код программы.

### Класс локатора/контейнера.

using CommonServiceLocator;

using GalaSoft.MvvmLight;

using GalaSoft.MvvmLight.Ioc;

using LabaApp.Main;

using LabaApp.Model;

using LabaApp.Services;

namespace LabaApp

{

/// <summary>

/// Локатор (контейнер) в котором выполняется регистрация сервисов и декларируемых типов для дальнейшего использованиях их экземпляров.

/// </summary>

public class ViewModelLocator

{

/// <summary>

/// Конструктор <see cref="ViewModelLocator"/>.

/// </summary>

public ViewModelLocator()

{

ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);

if (ViewModelBase.IsInDesignModeStatic)

SimpleIoc.Default.Register<IDataService, DesignDataService>();

else

SimpleIoc.Default.Register<IDataService, DataService>();

SimpleIoc.Default.Register<ISerializationService<Node>, XmlSerializationService>();

SimpleIoc.Default.Register<MainViewModel>();

}

/// <summary>

/// Ссылка на ViewModel <see cref="MainViewModel"/> для использования в дизайнере форм.

/// </summary>

public MainViewModel Main

{

get { return SimpleIoc.Default.GetInstance<MainViewModel>(); }

}

}

}

### Классы основной ViewModel и View.

using GalaSoft.MvvmLight;

using GalaSoft.MvvmLight.Command;

using LabaApp.Model;

using LabaApp.Services;

using Microsoft.Win32;

using System.IO;

using System.Windows.Input;

namespace LabaApp.Main

{

/// <summary>

/// Основная ViewModel приложения для работы с динамической структурой.

/// </summary>

public class MainViewModel : ViewModelBase

{

private Node \_root;

private Node \_selectedNode;

private OpenFileDialog \_openFileDialog;

private SaveFileDialog \_saveFileDialog;

private readonly IDataService \_dataService;

private readonly ISerializationService<Node> \_serializationService;

/// <summary>

/// Конструктор <see cref="MainViewModel"/>.

/// </summary>

/// <param name="dataService">Сервис получения данных динамической структуры.</param>

/// <param name="serializationService">Сервис сериализации.</param>

public MainViewModel(IDataService dataService, ISerializationService<Node> serializationService)

{

\_dataService = dataService;

\_serializationService = serializationService;

Root = \_dataService.GetData();

\_openFileDialog = new OpenFileDialog();

\_openFileDialog.Filter = "Файл данных динамических структур|\*.xml";

\_saveFileDialog = new SaveFileDialog();

\_saveFileDialog.Filter = "Файл данных динамических структур|\*.xml";

SaveStructCommand = new RelayCommand(SaveStruct);

LoadStructCommand = new RelayCommand(LoadStruct);

}

/// <summary>

/// Корневой элемент динамической структуры.

/// </summary>

public Node Root

{

get { return \_root; }

set { Set(ref \_root, value); }

}

/// <summary>

/// Выбранный элемент структуры.

/// </summary>

public Node SelectedNode

{

get { return \_selectedNode; }

set { Set(ref \_selectedNode, value); }

}

/// <summary>

/// Сохранение структуры в файл.

/// </summary>

private void SaveStruct()

{

if (\_saveFileDialog.ShowDialog() != true)

return;

using (var stream = new FileStream(\_saveFileDialog.FileName, FileMode.Create, FileAccess.Write))

{

\_serializationService.Serialize(Root, stream);

}

}

/// <summary>

/// Загрузка структуры из файла.

/// </summary>

private void LoadStruct()

{

if (\_openFileDialog.ShowDialog() != true)

return;

using (var stream = new FileStream(\_openFileDialog.FileName, FileMode.Open, FileAccess.Read))

{

Root = \_serializationService.Deserialize(stream);

}

}

/// <summary>

/// Команда сохранения структуры.

/// </summary>

public ICommand SaveStructCommand { get; }

/// <summary>

/// Команда загрузки структуры.

/// </summary>

public ICommand LoadStructCommand { get; }

}

}

<Window x:Class="LabaApp.Main.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:LabaApp.Main"

mc:Ignorable="d"

Title="Динамическая структура данных"

DataContext="{Binding Main, Source={StaticResource ViewModelLocator}}"

d:DesignHeight="360"

d:DesignWidth="640"

Width="640"

SizeToContent="Height">

<Window.Resources>

<Style TargetType="{x:Type TextBox}" x:Key="ItemFoundStyle">

<Style.Triggers>

<DataTrigger Binding="{Binding Highlight, Mode=OneWay, UpdateSourceTrigger=PropertyChanged}" Value="true">

<Setter Property="Background" Value="Green" />

</DataTrigger>

</Style.Triggers>

</Style>

</Window.Resources>

<ScrollViewer ScrollViewer.VerticalScrollBarVisibility="Auto">

<Grid Name="MainGrid">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="\*" />

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="5\*" />

</Grid.RowDefinitions>

<!--Поиск-->

<StackPanel Orientation="Vertical"

Grid.Column="0">

<TextBlock Text="Поиск"

FontSize="20"

FontWeight="Black"

VerticalAlignment="Top"

Margin="15, 15, 15, 15" />

<StackPanel Orientation="Horizontal"

Grid.Column="1"

Grid.Row="0">

<Button Content="Искать"

Command="{Binding Root.SearchCommand}"

VerticalAlignment="Center"

CommandParameter="{Binding Text, ElementName=SearchTextBox}"

Margin="15, 15, 15, 15"

Padding="5,1" />

<TextBox x:Name="SearchTextBox"

BorderThickness="2"

Background="LightGray"

VerticalAlignment="Center"

Margin="15, 15, 15, 15"

MinWidth="100"

Padding="5,1"/>

<Button Content="Очистить"

Command="{Binding Root.ClearCommand}"

VerticalAlignment="Center"

Margin="15, 15, 15, 15"

Padding="5,1" />

</StackPanel>

</StackPanel>

<!--Сохранение/загрузка-->

<StackPanel Orientation="Vertical"

Grid.Column="1">

<TextBlock Text="Сохранение/загрузка"

FontSize="20"

FontWeight="Black"

VerticalAlignment="Top"

Margin="15, 15, 15, 15" />

<StackPanel Orientation="Horizontal"

Grid.Column="2"

Grid.Row="0">

<Button Command="{Binding SaveStructCommand}"

Content="Сохранить в файл"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Margin="15, 15, 15, 15"

Padding="5,1" />

<Button Command="{Binding LoadStructCommand}"

Content="Загрузить из файла"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Margin="15, 15, 15, 15"

Padding="5,1" />

</StackPanel>

</StackPanel>

<Grid Name="ItemsGrid"

Grid.Row="1"

Grid.ColumnSpan="3">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="0.1\*" />

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="0.1\*" />

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="0.1\*" />

</Grid.RowDefinitions>

<ListView Grid.Column="1"

Grid.Row="0"

ItemsSource="{Binding Root.SubNodes, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"

SelectedItem="{Binding SelectedNode, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"

BorderThickness="2"

VerticalAlignment="Top">

<ItemsControl.ItemsPanel>

<ItemsPanelTemplate>

<StackPanel Orientation="Vertical" />

</ItemsPanelTemplate>

</ItemsControl.ItemsPanel>

<ItemsControl.ItemTemplate>

<DataTemplate>

<DockPanel>

<StackPanel>

<Button Command="{Binding AddNodeCommand}"

Content="Добавить элемент"

Margin="5, 15, 5, 5"

Padding="5,1" />

<Button Command="{Binding DeleteNodeCommand}"

Content="Удалить строку"

Margin="5, 5, 5, 15" />

</StackPanel>

<TextBox Style="{StaticResource ItemFoundStyle}"

Text="{Binding Value, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"

Margin="15, 15, 20, 15"

TextAlignment="Center"

VerticalAlignment="Center"

FontSize="20" />

<ItemsControl ItemsSource="{Binding SubNodes, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}">

<ItemsControl.ItemsPanel>

<ItemsPanelTemplate>

<StackPanel Orientation="Horizontal" />

</ItemsPanelTemplate>

</ItemsControl.ItemsPanel>

<ItemsControl.ItemTemplate>

<DataTemplate>

<StackPanel Orientation="Vertical"

VerticalAlignment="Center">

<TextBox Style="{StaticResource ItemFoundStyle}"

Text="{Binding Value, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"

Margin="5, 5, 5, 5"

TextAlignment="Justify"

VerticalAlignment="Center"

FontSize="12"

Padding="2,1"/>

<Button Content="Удалить"

Command="{Binding DeleteNodeCommand}"

FontSize="8"

HorizontalAlignment="Center"/>

</StackPanel>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</DockPanel>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ListView>

<Button Grid.Column="1"

Grid.Row="2"

Command="{Binding Root.AddNodeCommand}"

Content="Добавить строку"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="15, 15, 15, 15"

Padding="5,1" />

</Grid>

</Grid>

</ScrollViewer>

</Window>

### Классы модели – объектов предметной области (узел, список, узел для сериализации)

using GalaSoft.MvvmLight;

using GalaSoft.MvvmLight.Command;

using System.Windows.Input;

namespace LabaApp.Model

{

/// <summary>

/// Узел/элемент списка.

/// </summary>

public class Node : ObservableObject

{

private string \_value;

private Node \_parent;

private ObservableLinkedList<Node> \_subNodes;

private bool \_highlight;

/// <summary>

/// Конструктор <see cref="Node"/>.

/// </summary>

public Node()

{

SubNodes = new ObservableLinkedList<Node>();

AddNodeCommand = new RelayCommand(AddNode);

SearchCommand = new RelayCommand<string>(Search);

ClearCommand = new RelayCommand(Clear);

DeleteNodeCommand = new RelayCommand(DeleteNode);

}

/// <summary>

/// Подсвечен ли элемент (при поиске).

/// </summary>

public bool Highlight

{

get { return \_highlight; }

set { Set(ref \_highlight, value); }

}

/// <summary>

/// Коллекция дочерних элементов.

/// </summary>

public ObservableLinkedList<Node> SubNodes

{

get { return \_subNodes; }

set { Set(ref \_subNodes, value); }

}

/// <summary>

/// Текстовое значение элемента (имя).

/// </summary>

public string Value

{

get { return \_value; }

set { Set(ref \_value, value); }

}

/// <summary>

/// Родительский элемент.

/// </summary>

public Node Parent

{

get { return \_parent; }

set { Set(ref \_parent, value); }

}

/// <summary>

/// Добавить дочерний элемент.

/// </summary>

private void AddNode()

{

SubNodes.AddLast(new Node() { Value = "Новый элемент", Parent = this });

}

/// <summary>

/// Поиск элемента с заданным текстом (текущий элемент и его дочерние элементы).

/// </summary>

/// <param name="text">Текст для поиска.</param>

private void Search(string text)

{

Highlight = Value?.Contains(text) ?? false;

foreach (Node subnode in SubNodes)

subnode.Search(text);

}

/// <summary>

/// Сброс поиска.

/// </summary>

private void Clear()

{

Highlight = false;

foreach (Node subnode in SubNodes)

subnode.Clear();

}

/// <summary>

/// Удалить данный элемент.

/// </summary>

private void DeleteNode()

{

Parent.SubNodes.Remove(this);

}

/// <summary>

/// Команда поиска значения элемента.

/// </summary>

public ICommand SearchCommand { get; }

/// <summary>

/// Команда добавления элемента.

/// </summary>

public ICommand AddNodeCommand { get; }

/// <summary>

/// Команда сброса поиска.

/// </summary>

public ICommand ClearCommand { get; }

/// <summary>

/// Команда удаления элемента.

/// </summary>

public ICommand DeleteNodeCommand { get; }

}

}

using System;

using System.Collections;

using System.Collections.Generic;

using System.Collections.Specialized;

namespace LabaApp.Model

{

/// <summary>

/// This class is a LinkedList that can be used in a WPF MVVM scenario. Composition was used instead of inheritance,

/// because inheriting from LinkedList does not allow overriding its methods.

/// </summary>

/// <typeparam name="T"></typeparam>

//see: https://stackoverflow.com/questions/6996425/observable-linkedlist

public class ObservableLinkedList<T> : INotifyCollectionChanged, IEnumerable, IEnumerable<T>

{

private LinkedList<T> underLyingLinkedList;

/// <summary>

/// Количество элементов двусвязного списка.

/// </summary>

public int Count

{

get { return underLyingLinkedList.Count; }

}

/// <summary>

/// Первый элемент двусвязного списка.

/// </summary>

public LinkedListNode<T> First

{

get { return underLyingLinkedList.First; }

}

/// <summary>

/// Последний элемент двусвязного списка.

/// </summary>

public LinkedListNode<T> Last

{

get { return underLyingLinkedList.Last; }

}

/// <summary>

/// Конструктор <see cref="ObservableLinkedList"/>.

/// </summary>

public ObservableLinkedList()

{

underLyingLinkedList = new LinkedList<T>();

}

/// <summary>

/// Конструктор <see cref="ObservableLinkedList"/>.

/// </summary>

public ObservableLinkedList(IEnumerable<T> collection)

{

underLyingLinkedList = new LinkedList<T>(collection);

}

/// <summary>

/// Добавить в список новый элемент перед заданным.

/// </summary>

/// <param name="prevNode">Элемент списка, перед котороым нужно добавить новый.</param>

/// <param name="value">Добавляемое значение.</param>

/// <returns>Ссылка на добавленный в список элемент.</returns>

public LinkedListNode<T> AddAfter(LinkedListNode<T> prevNode, T value)

{

LinkedListNode<T> ret = underLyingLinkedList.AddAfter(prevNode, value);

OnNotifyCollectionChanged();

return ret;

}

/// <summary>

/// Добавить в список новый элемент перед заданным.

/// </summary>

/// <param name="prevNode">Элемент списка, перед котороым нужно добавить новый.</param>

/// <param name="value">Добавляемое значение.</param>

public void AddAfter(LinkedListNode<T> node, LinkedListNode<T> newNode)

{

underLyingLinkedList.AddAfter(node, newNode);

OnNotifyCollectionChanged();

}

/// <summary>

/// Добавить в список новый элемент после заданного.

/// </summary>

/// <param name="prevNode">Элемент списка, после которого нужно добавить новый.</param>

/// <param name="value">Добавляемое значение.</param>

/// <returns>Ссылка на добавленный в список элемент.</returns>

public LinkedListNode<T> AddBefore(LinkedListNode<T> node, T value)

{

LinkedListNode<T> ret = underLyingLinkedList.AddBefore(node, value);

OnNotifyCollectionChanged();

return ret;

}

/// <summary>

/// Добавить в список новый элемент после заданного.

/// </summary>

/// <param name="prevNode">Элемент списка, после которого нужно добавить новый.</param>

/// <param name="value">Добавляемое значение.</param>

public void AddBefore(LinkedListNode<T> node, LinkedListNode<T> newNode)

{

underLyingLinkedList.AddBefore(node, newNode);

OnNotifyCollectionChanged();

}

/// <summary>

/// Добавить в начало списка новый элемент.

/// </summary>

/// <param name="value">Добавляемый элемент.</param>

/// <returns>Ссылка на добавленный в список элемент.</returns>

public LinkedListNode<T> AddFirst(T value)

{

LinkedListNode<T> ret = underLyingLinkedList.AddFirst(value);

OnNotifyCollectionChanged();

return ret;

}

/// <summary>

/// Добавить в начало списка новый элемент.

/// </summary>

/// <param name="value">Добавляемый элемент.</param>

public void AddFirst(LinkedListNode<T> node)

{

underLyingLinkedList.AddFirst(node);

OnNotifyCollectionChanged();

}

/// <summary>

/// Добавить в конец списка новый элемент.

/// </summary>

/// <param name="value">Добавляемый элемент.</param>

/// <returns>Ссылка на добавленный в список элемент.</returns>

public LinkedListNode<T> AddLast(T value)

{

LinkedListNode<T> ret = underLyingLinkedList.AddLast(value);

OnNotifyCollectionChanged();

return ret;

}

/// <summary>

/// Добавить в конец списка новый элемент.

/// </summary>

/// <param name="value">Добавляемый элемент.</param>

public void AddLast(LinkedListNode<T> node)

{

underLyingLinkedList.AddLast(node);

OnNotifyCollectionChanged();

}

/// <summary>

/// Очистить список.

/// </summary>

public void Clear()

{

underLyingLinkedList.Clear();

OnNotifyCollectionChanged();

}

/// <summary>

/// Содержит ли список элемент с заданным значением.

/// </summary>

/// <param name="value">Значение элемента для поиска.</param>

/// <returns>Признак наличия в списке элемента с заданным значением.</returns>

public bool Contains(T value)

{

return underLyingLinkedList.Contains(value);

}

/// <summary>

/// Копировать элементы списка с заданной позиции в массив.

/// </summary>

/// <param name="array">Целевой массив для копирования.</param>

/// <param name="index">Позиция списка, начиная с которой элементы будут скопированы в целевой массив.</param>

public void CopyTo(T[] array, int index)

{

underLyingLinkedList.CopyTo(array, index);

}

/// <summary>

/// Проверка списка на равенство с другим объектом.

/// </summary>

/// <param name="obj">Объект для сравнения.</param>

/// <returns>Равен ли список заданному элементу.</returns>

public bool LinkedListEquals(object obj)

{

return underLyingLinkedList.Equals(obj);

}

/// <summary>

/// Находит первый элемент списка с заданным значением.

/// </summary>

/// <param name="value">Значение элемента для поиска.</param>

/// <returns>Элемент списка с заданным значением.</returns>

public LinkedListNode<T> Find(T value)

{

return underLyingLinkedList.Find(value);

}

/// <summary>

/// Находит последний элемент списка с заданным значением.

/// </summary>

/// <param name="value">Значение элемента для поиска.</param>

/// <returns>Элемент списка с заданным значением.</returns>

public LinkedListNode<T> FindLast(T value)

{

return underLyingLinkedList.FindLast(value);

}

/// <summary>

/// Получить тип списка.

/// </summary>

/// <returns>Тип списка.</returns>

public Type GetLinkedListType()

{

return underLyingLinkedList.GetType();

}

/// <summary>

/// Удалить элемент с заданным значением.

/// </summary>

/// <param name="value">Значение элемента для удаления.</param>

/// <returns>Удален ли элемент.</returns>

public bool Remove(T value)

{

bool ret = underLyingLinkedList.Remove(value);

OnNotifyCollectionChanged();

return ret;

}

/// <summary>

/// Удалить выбранный элемент списка.

/// </summary>

public void Remove(LinkedListNode<T> node)

{

underLyingLinkedList.Remove(node);

OnNotifyCollectionChanged();

}

/// <summary>

/// Удалить первый элемент списка.

/// </summary>

public void RemoveFirst()

{

underLyingLinkedList.RemoveFirst();

OnNotifyCollectionChanged();

}

/// <summary>

/// Удалить последний элемент списка.

/// </summary>

public void RemoveLast()

{

underLyingLinkedList.RemoveLast();

OnNotifyCollectionChanged();

}

/// <summary>

/// Событие изменение коллекции. Реализация интерфейса <see cref="INotifyCollectionChanged"/>.

/// </summary>

public event NotifyCollectionChangedEventHandler CollectionChanged;

public void OnNotifyCollectionChanged()

{

CollectionChanged?.Invoke(this, new NotifyCollectionChangedEventArgs(NotifyCollectionChangedAction.Reset));

}

/// <summary>

/// Реализация интерфейса <see cref="IEnumerable"/>.

/// </summary>

/// <returns>Перечислитель перечисления коллекции.</returns>

IEnumerator IEnumerable.GetEnumerator()

{

return (underLyingLinkedList as IEnumerable).GetEnumerator();

}

/// <summary>

/// Реализация интерфейса <see cref="IEnumerable{T}"/>.

/// </summary>

/// <returns>Перечислитель типизированного перечисления коллекции.</returns>

IEnumerator<T> IEnumerable<T>.GetEnumerator()

{

return (underLyingLinkedList as IEnumerable<T>).GetEnumerator();

}

}

}

using System;

using System.Linq;

using System.Xml.Serialization;

namespace LabaApp.Model.Dto

{

/// <summary>

/// Обёртка для сериализации типа <see cref="Node"/> в XML.

/// </summary>

[Serializable]

[XmlRoot(ElementName = "node")]

public class NodeDto

{

/// <summary>

/// Имя элемента.

/// </summary>

[XmlElement(ElementName = "value")]

public string Value { get; set; }

/// <summary>

/// Список дочерних элементов.

/// </summary>

[XmlArray(ElementName = "subnodes")]

[XmlArrayItem(ElementName = "node", Type = typeof(NodeDto))]

public NodeDto[] SubNodes { get; set; }

/// <summary>

/// Конструктор <see cref="NodeDto"/>.

/// </summary>

/// <param name="node"></param>

public NodeDto(Node node)

{

Serialize(node);

}

/// <summary>

/// Конструктор <see cref="NodeDto"/> по умолчанию.

/// </summary>

public NodeDto()

{

}

/// <summary>

/// Инициализация объекта сериализации на основе объекта модели.

/// </summary>

/// <param name="node"></param>

public void Serialize(Node node)

{

Value = node.Value;

SubNodes = node.SubNodes.Select(subNode => new NodeDto(subNode)).ToArray();

}

/// <summary>

/// Получение объекта модели на основе объекта сериализации.

/// </summary>

/// <returns></returns>

public Node Deserialize()

{

return DeserializeInner();

}

private Node DeserializeInner(Node parent = null)

{

var result = new Node()

{

Parent = parent,

Value = Value,

SubNodes = new ObservableLinkedList<Node>()

};

foreach (var item in SubNodes)

result.SubNodes.AddLast(item.DeserializeInner(result));

return result;

}

}

}

### Классы сервисов.

using LabaApp.Model;

namespace LabaApp.Services

{

/// <summary>

/// Интерфейс получения данных приложения.

/// </summary>

public interface IDataService

{

/// <summary>

/// Получить корневой элемент динамической структуры.

/// </summary>

/// <returns>Корневой элемент структуры.</returns>

Node GetData();

}

}

using LabaApp.Model;

namespace LabaApp.Services

{

/// <inheritdoc />

public class DataService : IDataService

{

/// <inheritdoc />

public Node GetData()

{

var t1 = new ObservableLinkedList<Node>();

t1.AddLast(new Node()

{

Value = "ТРАНСАЭРО"

});

t1.AddLast(new Node()

{

Value = "АЭРОЛИФТ"

});

t1.AddLast(new Node()

{

Value = "ИСТЛАЙН"

});

var x1 = new Node

{

Value = "DME",

SubNodes = t1

};

foreach (var node in x1.SubNodes)

{

node.Parent = x1;

}

var t2 = new ObservableLinkedList<Node>();

t2.AddLast(new Node()

{

Value = "САМАРА"

});

var x2 = new Node

{

Value = "KUF",

SubNodes = t2

};

foreach (var node in x2.SubNodes)

{

node.Parent = x2;

}

var t3 = new ObservableLinkedList<Node>();

t3.AddLast(new Node()

{

Value = "КАРАТ"

});

var x3 = new Node

{

Value = "VKO",

SubNodes = t3

};

foreach (var node in x3.SubNodes)

{

node.Parent = x3;

}

var nodes = new ObservableLinkedList<Node>();

nodes.AddLast(x1);

nodes.AddLast(x2);

nodes.AddLast(x3);

var root = new Node

{

SubNodes = nodes

};

foreach (var node in nodes)

{

node.Parent = root;

}

return root;

}

}

}

using LabaApp.Model;

namespace LabaApp.Services

{

/// <inheritdoc />

public class DesignDataService : IDataService

{

/// <inheritdoc />

public Node GetData()

{

var t1 = new ObservableLinkedList<Node>();

t1.AddLast(new Node()

{

Value = "ТРАНСАЭРО"

});

t1.AddLast(new Node()

{

Value = "АЭРОЛИФТ"

});

t1.AddLast(new Node()

{

Value = "ИСТЛАЙН"

});

var x1 = new Node

{

Value = "DME",

SubNodes = t1

};

var t2 = new ObservableLinkedList<Node>();

t2.AddLast(new Node()

{

Value = "САМАРА"

});

var x2 = new Node

{

Value = "KUF",

SubNodes = t2

};

var t3 = new ObservableLinkedList<Node>();

t3.AddLast(new Node()

{

Value = "КАРАТ"

});

var x3 = new Node

{

Value = "VKO",

SubNodes = t3

};

var nodes = new ObservableLinkedList<Node>();

nodes.AddLast(x1);

nodes.AddLast(x2);

nodes.AddLast(x3);

return new Node

{

SubNodes = nodes

};

}

}

}

using System.IO;

namespace LabaApp.Services

{

/// <summary>

/// Интерфейс сервиса сериализации.

/// </summary>

public interface ISerializationService<T>

{

/// <summary>

/// Сериализация объекта в поток.

/// </summary>

/// <param name="value">Объект для сериализации.</param>

/// <param name="stream">Поток для сериализации.</param>

void Serialize(T value, Stream stream);

/// <summary>

/// Десериализация объекта из потока.

/// </summary>

/// <param name="stream">Поток для десериализации.</param>

/// <returns>Десериализованный объект.</returns>

T Deserialize(Stream stream);

}

}

using System.IO;

using LabaApp.Model;

using LabaApp.Model.Dto;

using System.Xml.Serialization;

using System;

using System.Xml;

namespace LabaApp.Services

{

/// <inheritdoc/>

public class XmlSerializationService : ISerializationService<Node>

{

/// <inheritdoc/>

private XmlSerializer \_xmlSerializer;

public XmlSerializationService()

{

\_xmlSerializer = new XmlSerializer(typeof(NodeDto));

}

/// <inheritdoc/>

public Node Deserialize(Stream stream)

{

if (stream == null)

throw new ArgumentNullException(nameof(stream));

var nodeDto = new NodeDto();

try

{

nodeDto = (NodeDto)\_xmlSerializer.Deserialize(stream);

}

catch (InvalidOperationException e1)

{

return null;

}

catch (XmlException e2)

{

return null;

}

catch (FileNotFoundException e3)

{

return null;

}

return nodeDto.Deserialize();

}

/// <inheritdoc/>

public void Serialize(Node node, Stream stream)

{

if (node == null)

throw new ArgumentNullException(nameof(node));

if (stream == null)

throw new ArgumentNullException(nameof(stream));

var nodeDto = new NodeDto();

try

{

nodeDto.Serialize(node);

\_xmlSerializer.Serialize(stream, nodeDto);

}

catch (XmlException e1)

{

}

}

}

}