

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

Принял

Барчев Николай Борисович

Москва 2018 год

Оглавление

[Задание. 3](#_Toc514615105)

[Описание программного продукта. 4](#_Toc514615106)

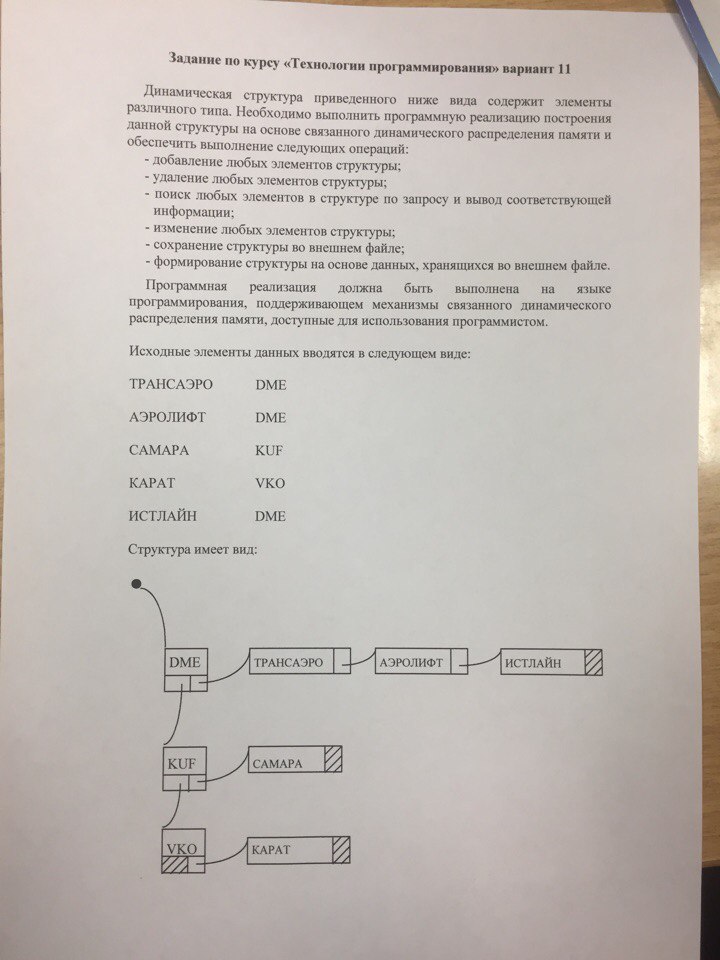
[Особенности реализации. 5](#_Toc514615107)

[Псевдокод. 7](#_Toc514615108)

[Описание программных единиц. 9](#_Toc514615109)

[Инструкция пользователя – примеры работы программы. 11](#_Toc514615110)

### 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.

### 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 **XmlSerializationService.cs**.

Класс XmlSerializationService – класс сериализатора данных в xml.

Файл **Node.cs**.

Класс Node – класс элемента двусвязного списка.

Файл **NodeDto.cs**.

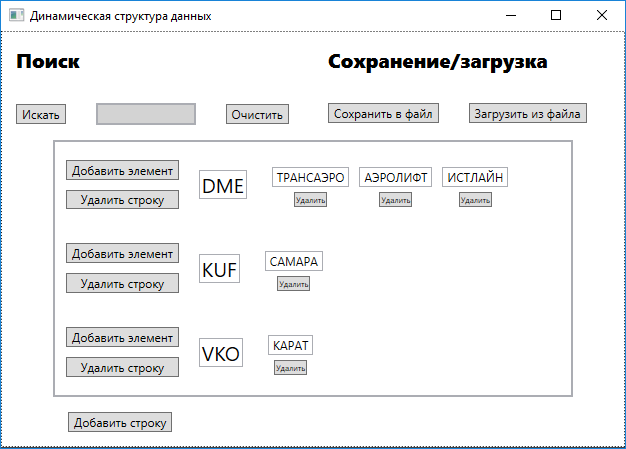
Класс NodeDto – класс для сериализации элемента двусвязного списка.

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».