ЛАБОРАТОРНАЯ РАБОТА №2.
LINQ

СОДЕРЖАНИЕ

Цель

Ознакомиться работать с LINQ-методами.

Задание

1. Создать ветку от ветки первой лабораторной.
2. *Требуется сделать приложение по учету изготовления изделий на заказ. Необходимо реализовать следующие требования:*
    a. *Ввод списка компонент, используемых при изготовлении изделий. Каждый компонент имеет уникальное имя, отличающее его от других компонент и цену за единицу компонента.*
    б. *Ввод списка производимых изделий. Для каждого изделия предусмотреть возможность указания компонент, из которых оно изготавливается и в каком количестве каждый компонент требуется при изготовлении изделия. Каждое изделие имеет стоимость, а также уникальное имя, отличающее его от других изделий. Стоимость изделия рассчитывается из суммы компонент, которые используются при его создании, и добавления определенного процента к этой сумме.*
    в. *Создание заказов. В заказ добавляется только одно изделие в любом количестве.*
    г. *Необходимо фиксировать дату создания заказа и дату выполнения.*
    д. *Заказ должен проходить ряд стадий: создание, изготовление, готов к выдачи, выдача заказа.*

    Приложение должно быть оформлено в виде desktop-приложения. Требуется реализовать способ хранения данных в файлах (простое хранение, для каждой сущности свой файл со списком, без учета связанности данных).

3. Вылить полученный результат в созданную ветку. Убедится, что там нет лишних файлов (типа .exe или .bin). Создать pull request.

Проектирование

Для реализации хранения данных в файлах создадим новый проект **AbstractShopFileImplement**. Этот проект будет относится к слою хранения данных и иметь те же зависимости, что и **AbstractShopListImplement**. По структуре он также будет идентичен проекту **AbstractShopListImplement**, 3 модели, 3 реализации и класс-Singleton для работы с файлами.

Реализация

Модели возьмем из того же проекта, но расширим их методы. Так как будет необходимо выгружать данные в файл и загружать данные из файла, то добавим метод создания объект класса-модели из данных файла и свойство формирования записи на основе объекта класса-модели для записи в файл. Работать будет через технологию LINQ to XML, у которого есть класс XElement, отвечающий за взаимодействия с записями в xml-файлах. В метод создания объекта класса-модели будет передаваться объект от класса XElement, а свойство формирования записи будет возвращать объект XElement. Рассмотрим на примере класса-модели сущности «Компонент» (листинг 2.1).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.ViewModels;
using AbstractShopDataModels.Models;
using System.Xml.Linq;

namespace AbstractShopFileImplement.Models
{
    public class Component : IComponentModel
    {
        public int Id { get; private set; }

        public string ComponentName { get; private set; } = string.Empty;

        public double Cost { get; set; }

        public static Component? Create(ComponentBindingModel model)
        {
            if (model == null)
            {
```

```
                return null;
            }
            return new Component()
            {
                Id = model.Id,
                ComponentName = model.ComponentName,
                Cost = model.Cost
            };
        }

        public static Component? Create(XElement element)
        {
            if (element == null)
            {
                return null;
            }
            return new Component()
            {
                Id = Convert.ToInt32(element.Attribute("Id")!.Value),
                ComponentName = element.Element("ComponentName")!.Value,
                Cost = Convert.ToDouble(element.Element("Cost")!.Value)
            };
        }

        public void Update(ComponentBindingModel model)
        {
            if (model == null)
            {
                return;
            }
            ComponentName = model.ComponentName;
            Cost = model.Cost;
        }

        public ComponentViewModel GetViewModel => new()
        {
            Id = Id,
            ComponentName = ComponentName,
            Cost = Cost
        };

        public XElement GetXElement => new("Component",
                        new XAttribute("Id", Id),
                        new XElement("ComponentName", ComponentName),
                        new XElement("Cost", Cost.ToString()));
    }
}
```

Листинг 2.1 – Класс Component

Идентификатор будем хранить в виде атрибута записи, а все остальные данные, как вложенные элементы.

Класс-изделие будет отличаться от аналогичного в проекте **AbstractShopListImplement**. Чтобы не хранить избыточную информацию с названиями компонент в файле, сделаем новое свойство в классе-изделии, которое будет хранить данные в виде «идентификатор компонента – количество компонента». А при получении записи через свойство

GetViewModel формировать данные с подстановкой названий компонент (листинг 2.2).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.ViewModels;
using AbstractShopDataModels.Models;
using System.Xml.Linq;

namespace AbstractShopFileImplement.Models
{
    public class Product : IProductModel
    {
        public int Id { get; private set; }

        public string ProductName { get; private set; } = string.Empty;

        public double Price { get; private set; }

        public Dictionary<int, int> Components { get; private set; } = new();

        private Dictionary<int, (IComponentModel, int)>? _productComponents =
null;

        public Dictionary<int, (IComponentModel, int)> ProductComponents
        {
            get
            {
                if (_productComponents == null)
                {
                    var source = DataFileSingleton.GetInstance();
                    _productComponents = Components.ToDictionary(x => x.Key, y =>
((source.Components.FirstOrDefault(z => z.Id == y.Key) as IComponentModel)!,
y.Value));
                }
                return _productComponents;
            }
        }

        public static Product? Create(ProductBindingModel model)
        {
            if (model == null)
            {
                return null;
            }
            return new Product()
            {
                Id = model.Id,
                ProductName = model.ProductName,
                Price = model.Price,
                Components = model.ProductComponents.ToDictionary(x => x.Key, x
=> x.Value.Item2)
            };
        }

        public static Product? Create(XElement element)
        {
            if (element == null)
            {
                return null;
            }
            return new Product()
            {
                Id = Convert.ToInt32(element.Attribute("Id")!.Value),
                ProductName = element.Element("ProductName")!.Value,
```

```
                Price = Convert.ToDouble(element.Element("Price")!.Value),
                Components =
element.Element("ProductComponents")!.Elements("ProductComponent")
                                .ToDictionary(x =>
Convert.ToInt32(x.Element("Key")?.Value), x =>
Convert.ToInt32(x.Element("Value")?.Value))
            };
        }

        public void Update(ProductBindingModel model)
        {
            if (model == null)
            {
                return;
            }
            ProductName = model.ProductName;
            Price = model.Price;
            Components = model.ProductComponents.ToDictionary(x => x.Key, x =>
x.Value.Item2);
            _productComponents = null;
        }

        public ProductViewModel GetViewModel => new()
        {
            Id = Id,
            ProductName = ProductName,
            Price = Price,
            ProductComponents = ProductComponents
        };

        public XElement GetXElement => new("Product",
                        new XAttribute("Id", Id),
                        new XElement("ProductName", ProductName),
                        new XElement("Price", Price.ToString()),
                        new XElement("ProductComponents", Components.Select(x =>
new XElement("ProductComponent",

new XElement("Key", x.Key),

new XElement("Value", x.Value)))

.ToArray()));
    }
}
```

Листинг 2.2 – Класс Product

Так как модель требует наличие свойства ProductComponents, то новое свойство назовем просто Components. При создании/редактировании записи от binding-модели будем преобразовывать свойство-словарь ProductComponents binding-модели в свойство-словарь Components модели. При формировании view-модели будем выполнять обратную операцию, для чего в модели определим метод get свойства ProductComponents. Для того, чтобы избежать лишних вычислительных операций при вызове get-метода введем поле _productComponents. Если оно пустое, то будем формировать из свойства

Components нужный набор данных, если оно уже есть, то просто его возвращаем.

Класс-модель по сущности «Заказ» будем иметь аналогичные модификации, что и класс-модель сущности «Компонент». Класс реализовать самостоятельно.

Класс со списками также будет реализован на основе паттерна Singleton. Однако появятся ряд новых методов. При работе с файлами возникает проблема, когда сохранять? Если на каждое действие пользователя будут производится манипуляции с файлами (чтение/запись), то это создаст дополнительную нагрузку на жесткий диск и каналы связи, а также замедлит (для современных компьютеров незначительно) работу программы. Самое логичное – при старте программы загружать из файлов данные, а при завершении программы – выгружать их. Вопрос: как это сделать, не изменяя логики приложения? Для загрузки все просто, у нас есть класс Singleton, в котором объявлены все списки. Объект от класса создается при первой обращении к этому классу и при создании можно загружать разом данные из всех файлов. А когда выгружать данные? Тут тоже можно воспользоваться этим классом. У любого класса есть деструктор, который вызывается сборщиком мусора при отчистке объекта класса из памяти (когда на него нет ни одной ссылки). Для нашего класса-Singleton есть только один экземпляр, который будет существовать все время работы программы и отчистится (вызовется деструктор) только при завершении программы. Это в идеальном мире… а на деле, начиная с .Net Core при закрытии программы нет гарантии вызова деструктора класса, поэтому придется идти по первому варианту и сохранять данные после вставки, изменения или удаления записи сущности.

Загрузка и сохранение данных для каждой сущности будет одинаковым по логике, так что сделаем универсальные методы для загрузки и для сохранения и для каждой сущности будем вызывать такой метод (листинг 2.3).

```
using AbstractShopFileImplement.Models;
using System.Xml.Linq;

namespace AbstractShopFileImplement
```

```
{
    internal class DataFileSingleton
    {
        private static DataFileSingleton? instance;

        private readonly string ComponentFileName = "Component.xml";

        private readonly string OrderFileName = "Order.xml";

        private readonly string ProductFileName = "Product.xml";

        public List<Component> Components { get; private set; }

        public List<Order> Orders { get; private set; }

        public List<Product> Products { get; private set; }

        public static DataFileSingleton GetInstance()
        {
            if (instance == null)
            {
                instance = new DataFileSingleton();
            }
            return instance;
        }

        public void SaveComponents() => SaveData(Components, ComponentFileName,
"Components", x => x.GetXElement);

        public void SaveProducts() => SaveData(Products, ProductFileName,
"Products", x => x.GetXElement);

        public void SaveOrders() { }

        private DataFileSingleton()
        {
            Components = LoadData(ComponentFileName, "Component", x =>
Component.Create(x)!)!;
            Products = LoadData(ProductFileName, "Product", x =>
Product.Create(x)!)!;
            Orders = new List<Order>();
        }

        private static List<T>? LoadData<T>(string filename, string xmlNodeName,
Func<XElement, T> selectFunction)
        {
            if (File.Exists(filename))
            {
                return
XDocument.Load(filename)?.Root?.Elements(xmlNodeName)?.Select(selectFunction)?.To
List();
            }
            return new List<T>();
        }

        private static void SaveData<T>(List<T> data, string filename, string
xmlNodeName, Func<T, XElement> selectFunction)
        {
            if (data != null)
            {
                new XDocument(new XElement(xmlNodeName,
data.Select(selectFunction).ToArray())).Save(filename);
            }
        }
    }
```

```
}
```

Листинг 2.3 – Класс DataFileSingleton

Остается реализовать 3 интерфейса с проекта **AbstractShopContracts**.

Реализация интерфейса IComponentStorage представлена в листинге 2.4.

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.StoragesContracts;
using AbstractShopContracts.ViewModels;
using AbstractShopFileImplement.Models;

namespace AbstractShopFileImplement.Implements
{
    public class ComponentStorage : IComponentStorage
    {
        private readonly DataFileSingleton source;

        public ComponentStorage()
        {
            source = DataFileSingleton.GetInstance();
        }

        public List<ComponentViewModel> GetFullList()
        {
            return source.Components
                    .Select(x => x.GetViewModel)
                    .ToList();
        }

        public List<ComponentViewModel> GetFilteredList(ComponentSearchModel
model)
        {
            if (string.IsNullOrEmpty(model.ComponentName))
            {
                return new();
            }
            return source.Components
                    .Where(x => x.ComponentName.Contains(model.ComponentName))
                    .Select(x => x.GetViewModel)
                    .ToList();
        }

        public ComponentViewModel? GetElement(ComponentSearchModel model)
        {
            if (string.IsNullOrEmpty(model.ComponentName) && !model.Id.HasValue)
            {
                return null;
            }
            return source.Components
                    .FirstOrDefault(x =>
(!string.IsNullOrEmpty(model.ComponentName) && x.ComponentName ==
model.ComponentName) ||
                                            (model.Id.HasValue && x.Id == model.Id))
                    ?.GetViewModel;
        }

        public ComponentViewModel? Insert(ComponentBindingModel model)
        {
            model.Id = source.Components.Count > 0 ? source.Components.Max(x =>
x.Id) + 1 : 1;
            var newComponent = Component.Create(model);
            if (newComponent == null)
```

```
            {
                return null;
            }
            source.Components.Add(newComponent);
            source.SaveComponents();
            return newComponent.GetViewModel;
        }

        public ComponentViewModel? Update(ComponentBindingModel model)
        {
            var component = source.Components.FirstOrDefault(x => x.Id ==
model.Id);
            if (component == null)
            {
                return null;
            }
            component.Update(model);
            source.SaveComponents();
            return component.GetViewModel;
        }

        public ComponentViewModel? Delete(ComponentBindingModel model)
        {
            var element = source.Components.FirstOrDefault(x => x.Id ==
model.Id);
            if (element != null)
            {
                source.Components.Remove(element);
                source.SaveComponents();
                return element.GetViewModel;
            }
            return null;
        }
    }
}
```

Листинг 2.4 – Класс ComponentStorage

Методы добавления, редактирования и удаления также преобразуются. Там для получения максимального значения идентификатора, для присвоения новой записи используется метод Max. А перед вызовом оператора return будет выполняться сохранение списка компонентов в файл.

Потребуется также реализовать интерфейсы IProductStorage и IOrderStorage.

Последний штрих. В проект **AbstractShopView** добавить ссылку на **AbstractShopFileImplement**. И в классе Program поменять всего одну строку в using и, к сожалению, добавить строку перед завершением метода main для сохранения списков. В результате приложение будет работать с другой реализацией и хранить данные в файлах (листинг 2.5).

```
using AbstractShopBusinessLogic.BusinessLogics;
using AbstractShopContracts.BusinessLogicsContracts;
using AbstractShopContracts.StoragesContracts;
using AbstractShopFileImplement.Implements;
```

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using NLog.Extensions.Logging;

namespace AbstractShopView
{
    internal static class Program
    {
             private static ServiceProvider? _serviceProvider;
             public static ServiceProvider? ServiceProvider => _serviceProvider;
             /// <summary>
             ///  The main entry point for the application.
             /// </summary>
             [STAThread]
        static void Main()
        {
            // To customize application configuration such as set high DPI
settings or default font,
            // see https://aka.ms/applicationconfiguration.
            ApplicationConfiguration.Initialize();
            var services = new ServiceCollection();
            ConfigureServices(services);
                   _serviceProvider = services.BuildServiceProvider();

      Application.Run(_serviceProvider.GetRequiredService<FormMain>());
            }

             private static void ConfigureServices(ServiceCollection services)
             {
                   services.AddLogging(option =>
                                {

      option.SetMinimumLevel(LogLevel.Information);
                                      option.AddNLog("nlog.config");
                                });
                   services.AddTransient<IComponentStorage, ComponentStorage>();
                   services.AddTransient<IOrderStorage, OrderStorage>();
                   services.AddTransient<IProductStorage, ProductStorage>();

                   services.AddTransient<IComponentLogic, ComponentLogic>();
                   services.AddTransient<IOrderLogic, OrderLogic>();
                   services.AddTransient<IProductLogic, ProductLogic>();

                   services.AddTransient<FormMain>();
                   services.AddTransient<FormComponent>();
                   services.AddTransient<FormComponents>();
                   services.AddTransient<FormCreateOrder>();
                   services.AddTransient<FormProduct>();
                   services.AddTransient<FormProductComponent>();
                   services.AddTransient<FormProducts>();
             }
      }
}
```

Листинг 2.5 – Класс Program

Контрольный пример

После первого запуска приложения появятся 3 файла с сохранёнными данными. При повторных запусках приложения данные будут сразу доступны пользователям.

Требования

1. Название проектов должны ОТЛИЧАТЬСЯ от названия проектов, приведенных в примере и должны соответствовать логике вашего задания по варианту.
2. Название форм, классов, свойств классов должно соответствовать логике вашего задания по варианту.
3. НЕ ИСПОЛЬЗОВАТЬ в названии класса, связанного с изделием слово «Product» (во вариантах в скобках указано название класса для изделия)!!!
4. Все элементы форм (заголовки форм, текст в label и т.д.) должны иметь подписи на одном языке (или все русским, или все английским).
5. Создать класс-модель для сущности «Заказ».
6. В классе DataFileSingleton прописать логику сохранения и загрузки данных для «Заказа».
7. Реализовать логику для классов-реализаций IProductStorage и IOrderStorage с использованием LINQ-запросов.

Порядок сдачи базовой части

1. Запустить приложение (должны отображаться созданные до этого заказы).
2. Открыть справочник по компонентам (должны отобразиться имеющиеся компоненты).
3. Открыть справочник по изделиям (должны отобразиться имеющиеся изделия).
4. Ответить на вопрос преподавателя.

Контрольные вопросы к базовой части

1. В коде показать пример LINQ-метода, позволяющего из набора данных получить поднабор данных по условию выборки (выполнить фильтрацию данных).
2. В коде показать пример LINQ-метода, позволяющего из набора данных получить первый элемент, удовлетворяющий условию выборки.
3. В коде показать пример LINQ-метода, позволяющего набор данных одного типа преобразовать в набор данных другого типа.

Усложненная лабораторная (необязательно)

1. Для сущности «Магазин» добавить новое поле – максимальное количество изделий.
2. Сохранять в файлы информацию по магазинам и их наполненность изделиями.
3. Сделать реализацию интерфейса логики хранения данных для сущности «Магазин» с сохранением в файл с использованием LINQ-запросов.
4. В логике для хранения данных «Магазин» прописать метод проверки наличия в магазинах изделий в нужном количестве (нужно считать суммарное количество каждого изделия в магазинах и сравнивать с требуемым количеством) и продажу изделий из магазинов в требуемом количестве (возможно, потребуется продать сразу с нескольких магазинов изделия, если на одном нет изделий в нужном количестве). Метод должен возвращать булевское значение по результату продажи изделий.
5. В реализации логики «Магазинов» с хранением данных в оперативной памяти метод проверки и продажи оставить нереализованным.

6. В бизнес-логике работы с заказами при переводе заказа в статус «Готов к выдаче» вызывать метод проверки и пополнения магазинов изделиями (можно пополнять сразу несколько магазинов, если в один все не помещается), и не переводить в статус «Выдан», если все магазины заполнены.

Порядок сдачи усложненной части

1. Показать магазины (сколько изделий в них расположено).
2. Создать заказ (количество изделий указывается преподавателем).
3. Попытаться перевести заказ в выдан (должна выйти ошибка о переполненности магазинов).
4. Продать с 2-х магазинов необходимое количество изделий.
5. Перевести заказ в выдан.
6. Показать магазины (сколько изделий в них расположено).
7. Ответить на вопрос преподавателя.

Контрольные вопросы к усложненной части

1. Какова логика проверки наполненности магазинов?
2. Какова логика продажи изделий с магазинов?
3. Как изменилась логика создания заказа?

Варианты

1. Кондитерская. В качестве компонентов выступают различные виды шоколада и наполнители, типа орехов, изюма и т.п. Изделие – кондитерское изделие (pastry).
2. Автомастерская. В качестве компонентов выступают различные масла, смазки и т.п. Изделия – ремонт автомобиля (repair).
3. Моторный завод. В качестве компонентов выступают различные детали для производства двигателей. Изделия – двигатели (engine).

4. Суши-бар. В качестве компонентов выступают различные продукты для суши (рыба, водоросли, соусы). Изделия – суши (sushi).
5. Продажа компьютеров. В качестве компонентов выступают различные части для компьютеров (планки памяти, жесткие диски и т.п.). Изделия – компьютеры (computer).
6. Сборка мебели. В качестве компонентов выступают различные заготовки (ножки, спинки и т.п.). Изделия – мебель (furniture).
7. Рыбный завод. В качестве компонентов выступают различные виды рыб + дополнения к ним, типа соусов и т.п. Изделия – консервы (canned).
8. Установка ПО. В качестве компонентов выступают различное ПО. Изделия – пакеты установки, например, пакет установки офисных приложений, пакет разработчика и т.п. (package).
9. Ремонтные работы в помещении. В качестве компонентов выступают различные расходные материалы (клей, обои, краска, плитка, цемент и т.п.). Изделия – ремонтные работы в различных помещениях (repair).
10. Кузнечная мастерская. В качестве компонентов выступают различные болванки (заготовки), из которых изготавливаются подковы, кочерги и т.п. Изделия – кузнечные изделия (manufacture).
11. Пиццерия. В качестве компонентов выступают различные ингредиенты для пицц (тесто, соусы, паста и т.д.). Изделия – пиццы (pizza).
12. Завод ЖБИ. В качестве компонентов выступают различные виды бетона и металлоконструкций. Изделия – железобетонные изделия (reinforced).
13. Закусочная. В качестве компонентов выступают различные продукты для закусок (колбаса, сыр, хлеб и т.п.). Изделия – различные закуски (snack).

14. Пошив платьев. В качестве компонентов выступают различные ткани, нитки и т.п. Изделия – платья (dress).
15. Типография. В качестве компонентов выступают различные типы бумаг, тонер или чернила и т.п. Изделия – печатная продукция (листовки, брошюры, книги) (printed).
16. Автомобильный завод. В качестве компонентов выступают различные части для сборки автомобилей (кузов, двигатель, стекла и т.п.). Изделия – автомобили (car).
17. Юридическая фирма. В качестве компонентов выступают различные бланки для документов. Изделия – пакеты документов, например, для страховки или завещания (document).
18. Туристическая фирма. В качестве компонентов выступают различные условия поездки (отель проживания, туры в рамках поездок). Изделия – туристические путевки (travel).
19. Цветочная лавка. В качестве компонентов выступают различные цветы и украшения к ним. Изделия – цветочные композиции (flower).
20. Ювелирная лавка. В качестве компонентов выступают различные драгоценные камни и металлы. Изделия – драгоценности (jewel).
21. Авиастроительный завод. В качестве компонентов выступают различные части для сборки самолета (двигатели, крылья, фюзеляж и т.п.). Изделия – самолеты (plane).
22. Магазин подарков. В качестве компонентов выступают различные упаковочные материалы, ленты и подарки. Изделия – подарочные наборы (gift).
23. Система безопасности. В качестве компонентов выступают различные камеры, датчики и т.п. Изделия – базовые комплектации охраны, продвинутые, для предприятий, для частных и т.п. (secure).
24. Заказы еды. В качестве компонентов выступают различные блюда. Изделия – это наборы блюд (типа обеденный набор, или утренний набор, или набор для пикника) (dish).

25. Ремонт сантехники. В качестве компонентов выступают различные трубы, прокладки, смесители т.п. Изделия – замены смесителей, труб и т.п. (work).
26. Лавка с мороженым. В качестве компонентов выступают различные виды мороженного и добавки (орехи, шоколад и т.п.). Изделия – мороженное (icecream).
27. Судостроительный завод. В качестве компонентов выступают различные части для сборки судов (корпуса, двигатели и т.п.). Изделия – суда (ship).
28. Столярная мастерская. В качестве компонентов выступают различные деревянные заготовки. Изделия – деревянные игрушки, утварь и т.п. (wood).
29. Бар. В качестве компонентов выступают различные ингредиенты для коктейлей. Изделия – коктейли (cocktail).
30. Швейная фабрика. В качестве компонентов выступают различные заготовки для штор, покрывал и т.п. (textile).