ФГАОУ ВО «КФУ им. В.И. Вернадского»
Физико-технический институт (структурное подразделение)

Кафедра компьютерной инженерии и моделирования

Скибинский Дмитрий Константинович

отчет по практической работе №5
по дисциплине **«ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ»**

Направление подготовки:
09.03.04 "Программная инженерия"

Оценка

Симферополь, 2024

Практическая работа №5
Тема: Наследование классов и использование интерфейсов

Цель работы: Научиться на практике наследовать классы и интерфейсы. Разобраться на практике с вопросами отличия интерфейсов и абстрактных классов, реализовать множественное наследование интерфейсов и явную реализация интерфейсов. Научиться строить архитектуру приложения с помощью интерфейсов, разобраться с преимуществами такого подхода при промышленном программировании.

Описание ключевых понятий:

Интерфейс - это контракт, который определяет набор методов, свойств, событий и индексаторов, которые должны быть реализованы классом или структурой, реализующей этот интерфейс. Интерфейсы не содержат реализации методов, только их сигнатуры.

Абстрактный класс - это класс, который не может быть инстанцирован напрямую, а предназначен для наследования. Он может содержать абстрактные методы (без реализации) и виртуальные методы (с реализацией по умолчанию).

Множественное наследование интерфейсов - это возможность класса или структуры реализовывать несколько интерфейсов. В отличие от множественного наследования классов, которое запрещено в C#, множественное наследование интерфейсов разрешено и широко используется.

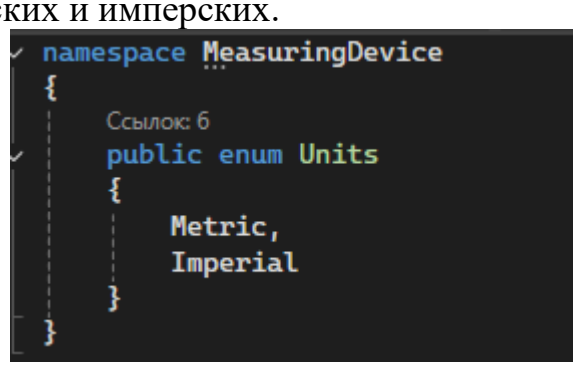
Явная реализация интерфейсов - это способ реализации методов интерфейса, при котором методы не являются публичными членами класса. Они доступны только через экземпляр интерфейса. Это позволяет избежать конфликтов имен при реализации нескольких интерфейсов с одинаковыми методами.

Перед выполнением лабораторной работы изучена следующая литература:

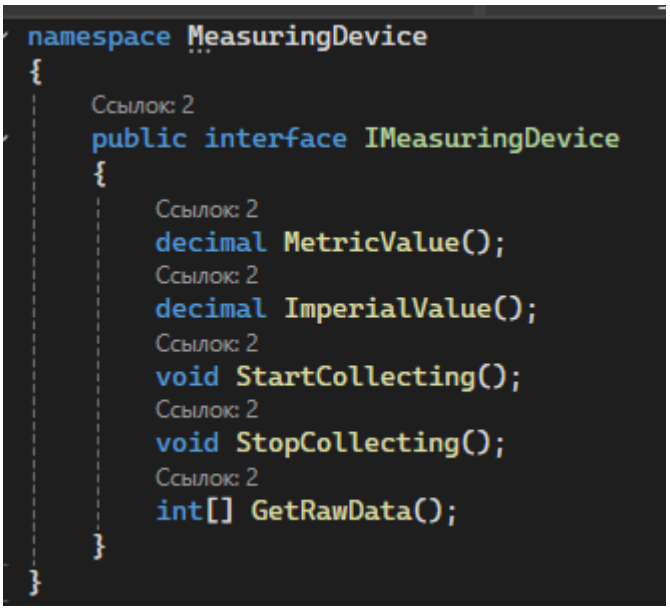
1. Изучить презентацию лектора курса: «Интерфейсы» (материалы доступны в "облаке" на Mail.ru и в Moodle КФУ).
2. Сайт Metanit.com
3. Справочник по C#. Корпорация Microsoft. http://msdn.microsoft.com/ru-ru/library/618ayhy6.aspx
4. Биллиг В.А. Основы программирования на C#. Интернет-университет информационных технологий. http://www.intuit.ru/studies/courses/2247/18/info
5. Павловская Т. Программирование на языке высокого уровня C#. http://www.intuit.ru/studies/courses/629/485/info
6. Руководство по программированию на C#. Корпорация Microsoft. http://msdn.microsoft.com/ru-ru/library/67ef8sbd.aspx
7. Корпорация Microsoft. C#. Спецификация языкаВыполнены 4 задания, описанных в методических указания к выполнению лабораторных работ.

Задание 1. Использование интерфейсов

Для выполнения сначала сделаем файл Units.cs. Перечисления используются для создания набора именованных констант, которые могут быть связаны с определенными значениями. В данном случае, перечисление Units используется для определения двух возможных единиц измерения: метрических и имперских.



Создадим файл IMeasuringDevice, который будет содержать интерфейс для приложения



MetricValue - метод возвращает текущее значение измерения в метрических единицах
ImperialValue - метод возвращает текущее значение измерения в имперских единицах
StartCollecting - метод запускает процесс сбора данных устройством
StopCollecting - метод останавливает процесс сбора данных устройством
GetRawData - метод возвращает массив сырых данных, собранных устройством
Файл MeasureLengthDevice будет реализовать все методы интерфейса

```
using DeviceController;
using System;
using System.Threading;

namespace MeasuringDevice
{
    Ссылок: 3
    public class MeasureLengthDevice : IMeasuringDevice
    {
        private Units unitsToUse;
        private int[] dataCaptured;
        private int mostRecentMeasure;
        private DeviceController.DeviceController controller;
        private const DeviceType measurementType = DeviceType.LENGTH;

        Ссылок: 2
        public MeasureLengthDevice(Units units)
        {
            unitsToUse = units;
        }

        Ссылок: 2
        public decimal MetricValue()
        {
            if (unitsToUse == Units.Metric)
                return mostRecentMeasure;
            else
                return mostRecentMeasure * 25.4m;
        }
```

л

MetricValue. Если устройство работает в метрических единицах, возвращает последнее измеренное значение. Если в имперских, конвертирует значение в метрические единицы (1 дюйм = 25.4 мм).

ImperialValue. Если устройство работает в имперских единицах, возвращает последнее измеренное значение. Если в метрических, конвертирует значение в имперские единицы (1 мм = 0.03937 дюйма).

StartCollecting. Создает экземпляр контроллера устройства и запускает процесс измерений с помощью метода GetMeasurements

StopCollecting. Если контроллер устройства существует, останавливает его и обнуляет ссылку на контроллер.

GetRawData. Возвращает массив dataCaptured

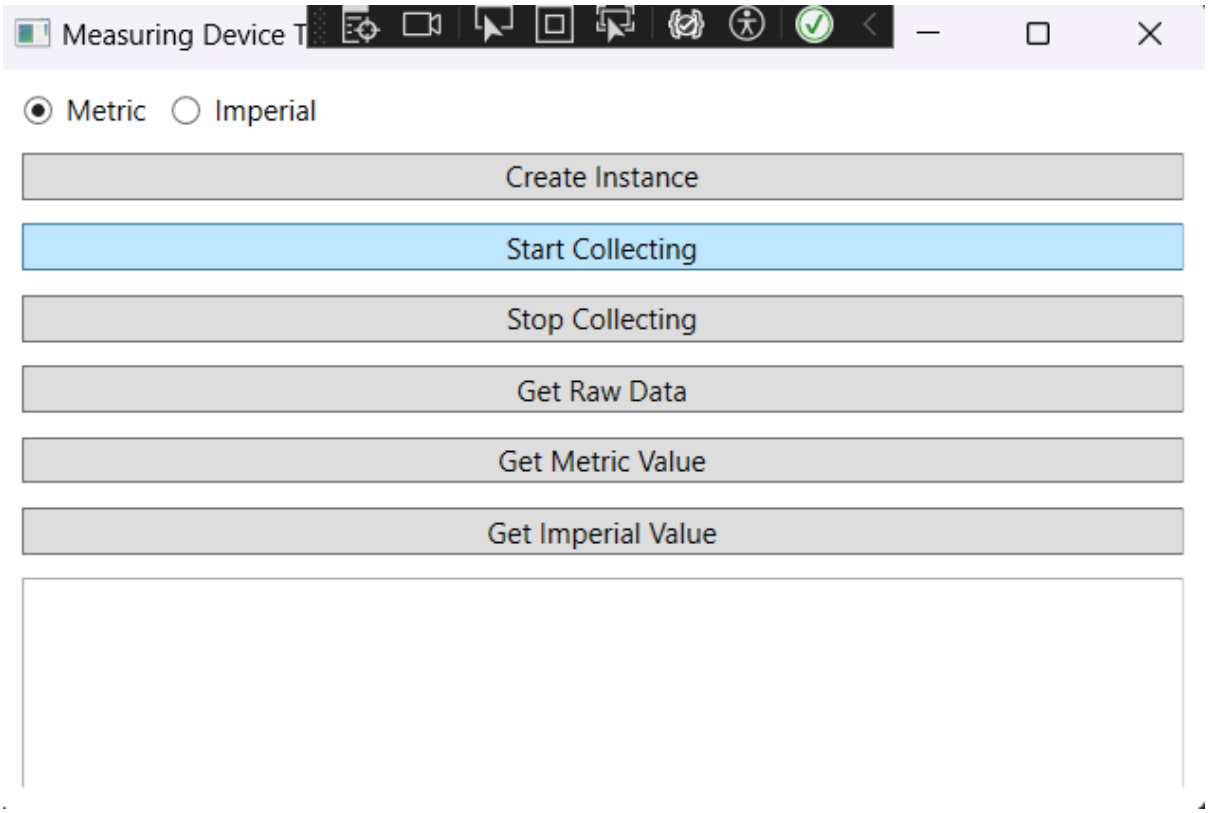
Файл DeviceController.cs имитирует контроллер реального устройства, который может запускать, останавливать и снимать измерения.

DeviceController предоставляет базовые элементы для управления и имитации работы устройств измерения.
Перечисление DeviceType определяет типы устройств, а

Класс DeviceController предоставляет методы для запуска, остановки и снятия измерений с устройства.

```
namespace DeviceController
{
    Ссылок: 3
    public enum DeviceType
    {
        LENGTH,
        MASS
    }

    Ссылок: 4
    public class DeviceController
    {
        Ссылок: 1
        public static DeviceController StartDevice(DeviceType type)
        {
            return new DeviceController();
        }

        Ссылок: 1
        public void StopDevice()
        {
        }

        Ссылок: 1
        public int TakeMeasurement()
        {
            return new Random().Next(1, 100);
        }
    }
}
```

Создадим пользовательский интерфейс WPF



Задание 2. Создание абстрактного класса

1. DeviceController.cs

Описание: Этот файл содержит класс DeviceController, который управляет различными типами устройств для измерения длины и массы. Он также определяет интерфейс IControllableDevice и два пространства имен (FabrikamDevices и ContosoDevices), которые содержат конкретные реализации устройств для измерения длины и массы.

Основные компоненты:

- DeviceController: Класс, который управляет устройством. Он содержит методы для запуска, остановки устройства, получения измерений и освобождения ресурсов.
- FabrikamDevices.LengthMeasuringDevice: Класс, представляющий устройство для измерения длины.
- ContosoDevices.MassMeasuringDevice: Класс, представляющий устройство для измерения массы.

- Функциональность:
  - StartDevice: Создает и запускает устройство определенного типа (длина или масса).
  - StopDevice: Останавливает устройство.
  - TakeMeasurement: Получает последнее измерение от устройства.
  - Dispose: Освобождает ресурсы, связанные с устройством.

```
using System;

namespace DeviceController
{
    Ссылок: 5
    public class DeviceController : IDisposable
    {
        private IControllableDevice device;

        /// <summary>
        /// A factory method to create and start a new instance of a device.
        /// </summary>
        /// <param name="MeasurementType">Specifies which type of device to start. Must be MASS or LENGTH.</param>
        /// <returns>An instance of the DeviceController class with the controlled device in the started state.</retu
        Ссылок: 1
        public static DeviceController StartDevice(DeviceType MeasurementType)
        {
            DeviceController controller = new DeviceController();
            switch (MeasurementType)
            {
                case DeviceType.LENGTH:
                    controller.device = new FabrikamDevices.LengthMeasuringDevice();
                    break;
                case DeviceType.MASS:
                    controller.device = new ContosoDevices.MassMeasuringDevice();
                    break;
            }
            if (controller.device != null)
            {
                controller.device.StartDevice();
            }
            return controller;
        }
```

2. **MeasureDataDevice.cs**

Этот файл содержит абстрактный класс MeasureDataDevice, который определяет основные методы и свойства для устройств, измеряющих данные.

Основные компоненты:
- **MeasureDataDevice**: Абстрактный класс, который содержит методы для начала и остановки сбора данных, получения необработанных данных, а также абстрактные методы для получения метрических и имперских значений.

Функциональность:
- **StartCollecting**: Начинает сбор данных с устройства.
- **StopCollecting**: Останавливает сбор данных.
- **GetRawData**: Возвращает необработанные данные, собранные с устройства.
- **MetricValue**: Абстрактный метод, который должен быть реализован в производных классах для получения метрического значения.
- **ImperialValue**: Абстрактный метод, который должен быть реализован в производных классах для получения имперского значения.

```
using DeviceController;
using System;
using System.Threading;

namespace MeasuringDevice
{
    Ссылок: 3
    public abstract class MeasureDataDevice
    {
        protected Units unitsToUse;
        protected int[] dataCaptured;
        protected int mostRecentMeasure;
        protected DeviceController.DeviceController controller;
        protected DeviceType measurementType;

        Ссылок: 4
        public abstract decimal MetricValue();
        Ссылок: 4
        public abstract decimal ImperialValue();

        Ссылок: 1
        public void StartCollecting()
        {
            controller = DeviceController.DeviceController.StartDevice(measurementType);
            GetMeasurements();
        }

        Ссылок: 1
        public void StopCollecting()
        {
            if (controller != null)
            {
                controller.StopDevice();
                controller = null;
            }
        }

        Ссылок: 1
```

3. MeasureLengthDevice.cs
- **Описание**: Этот файл содержит класс MeasureLengthDevice, который наследует от MeasureDataDevice и представляет устройство для измерения длины.
- **Основные компоненты**:
  - **MeasureLengthDevice**: Класс, который реализует методы для получения метрических и имперских значений длины.
- **Функциональность**:
  - **MetricValue**: Возвращает метрическое значение длины.

- **ImperialValue**: Возвращает имперское значение длины.

```
using DeviceController;

namespace MeasuringDevice
{
    Ссылок: 3
    public class MeasureLengthDevice : MeasureDataDevice
    {
        Ссылок: 2
        public MeasureLengthDevice(Units units)
        {
            unitsToUse = units;
            measurementType = DeviceType.LENGTH;
        }

        Ссылок: 3
        public override decimal MetricValue()
        {
            if (unitsToUse == Units.Metric)
                return mostRecentMeasure;
            else
                return mostRecentMeasure * 25.4m;
        }

        Ссылок: 3
        public override decimal ImperialValue()
        {
            if (unitsToUse == Units.Imperial)
                return mostRecentMeasure;
            else
                return mostRecentMeasure * 0.03937m;
        }
    }
}
```

4. MeasureMassDevice.cs

- **Описание**: Этот файл содержит класс MeasureMassDevice, который наследует от MeasureDataDevice и представляет устройство для измерения массы.
- **Основные компоненты**:
  - **MeasureMassDevice**: Класс, который реализует методы для получения метрических и имперских значений массы.
- **Функциональность**:
  - **MetricValue**: Возвращает метрическое значение массы.
  - **ImperialValue**: Возвращает имперское значение массы.

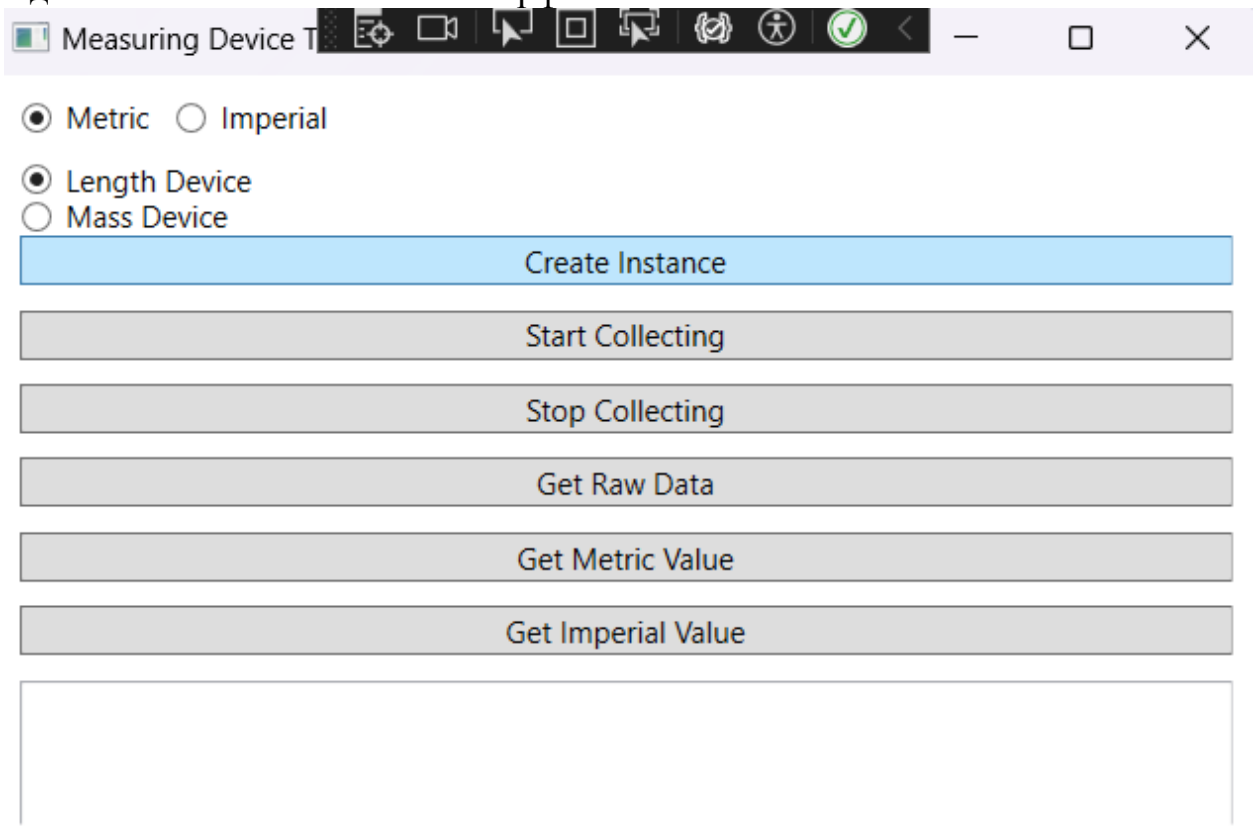
```
using DeviceController;

namespace MeasuringDevice
{
    Ссылок: 3
    public class MeasureMassDevice : MeasureDataDevice
    {
        Ссылок: 2
        public MeasureMassDevice(Units units)
        {
            unitsToUse = units;
            measurementType = DeviceType.MASS;
        }

        Ссылок: 3
        public override decimal MetricValue()
        {
            if (unitsToUse == Units.Metric)
                return mostRecentMeasure;
            else
                return mostRecentMeasure * 0.4536m;
        }

        Ссылок: 3
        public override decimal ImperialValue()
        {
            if (unitsToUse == Units.Imperial)
                return mostRecentMeasure;
            else
                return mostRecentMeasure * 2.2046m;
        }
    }
}
```

5. Units.cs

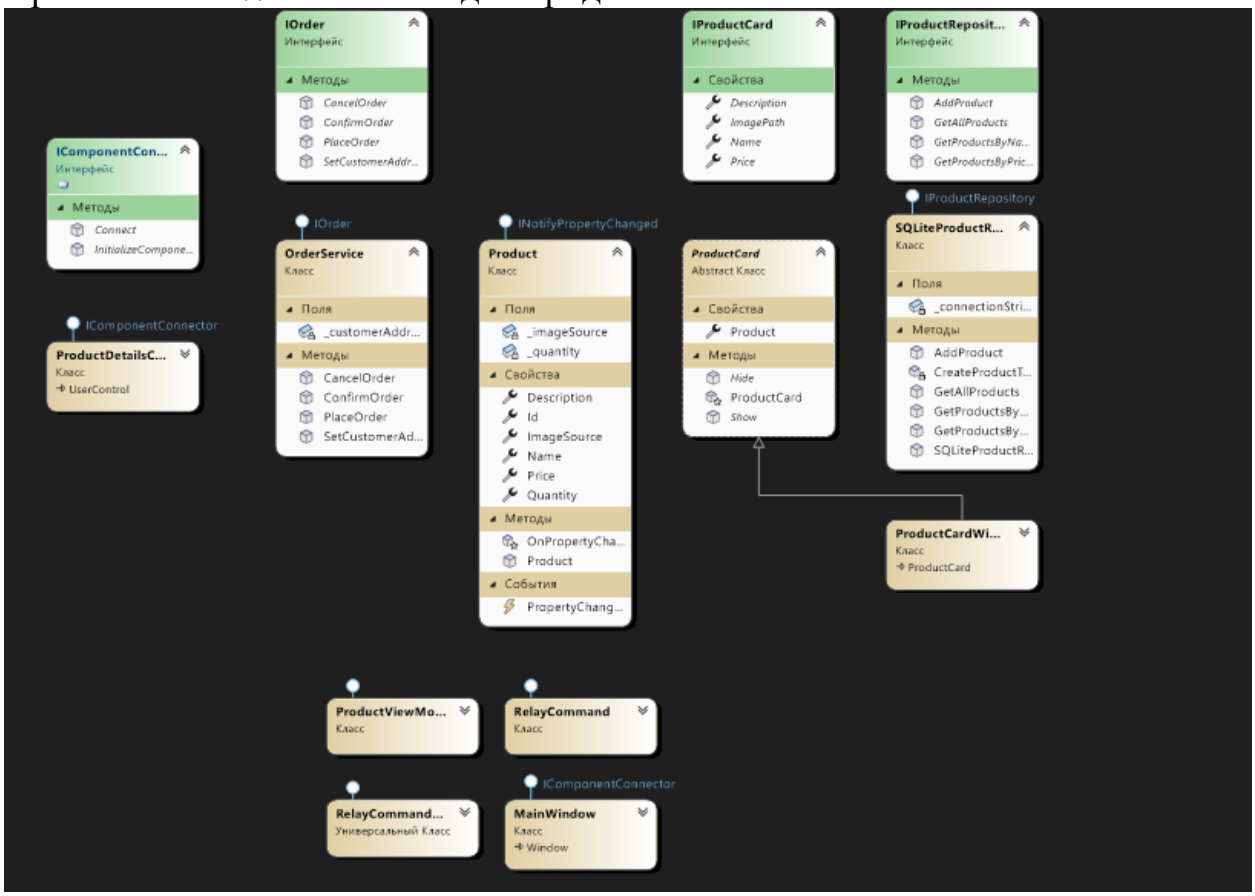
- **Описание**: Этот файл содержит перечисление Units, которое определяет возможные единицы измерения (метрические и имперские).
- **Основные компоненты**:
  - **Units**: Перечисление, которое содержит два значения: Metric и Imperial.
- **Функциональность**:
  - Определяет единицы измерения, которые могут быть использованы в устройствах для измерения данных.

```
namespace MeasuringDevice
{
    Ссылок: 11
    public enum Units
    {
        Metric,
        Imperial
    }
}
```

Сделаем пользовательский интерфейс



Задание 3. Создание архитектуры приложения с помощью интерфейсов
Приложенние WPF-магазин для продажи книг



1. Интерфейс IOrder

Описание:

Интерфейс IOrder определяет контракт для управления заказом. Он содержит методы, которые позволяют устанавливать адрес клиента, размещать заказ, отменять заказ и подтверждать заказ.

Методы:

- void SetCustomerAddress(string address): Устанавливает адрес клиента.
- void PlaceOrder(): Размещает заказ.
- void CancelOrder(): Отменяет заказ.
- void ConfirmOrder(): Подтверждает заказ.

Использование:

- OrderService реализует IOrder:
  - SetCustomerAddress(string address): Устанавливает адрес клиента в приватное поле _customerAddress.
  - PlaceOrder(): Выводит сообщение о размещении заказа с указанным адресом.
  - CancelOrder(): Выводит сообщение об отмене заказа.
  - ConfirmOrder(): Выводит сообщение о подтверждении заказа.
- В ProductViewModel:
  - Используется для управления заказом через команды:

        - SetCustomerAddressCommand: Устанавливает адрес клиента.
        - PlaceOrderCommand: Размещает заказ.
        - CancelOrderCommand: Отменяет заказ.
        - ConfirmOrderCommand: Подтверждает заказ.

2. Интерфейс IProductRepository

Описание:

Интерфейс IProductRepository определяет контракт для работы с репозиторием продуктов. Он содержит методы для получения всех продуктов и добавления нового продукта.

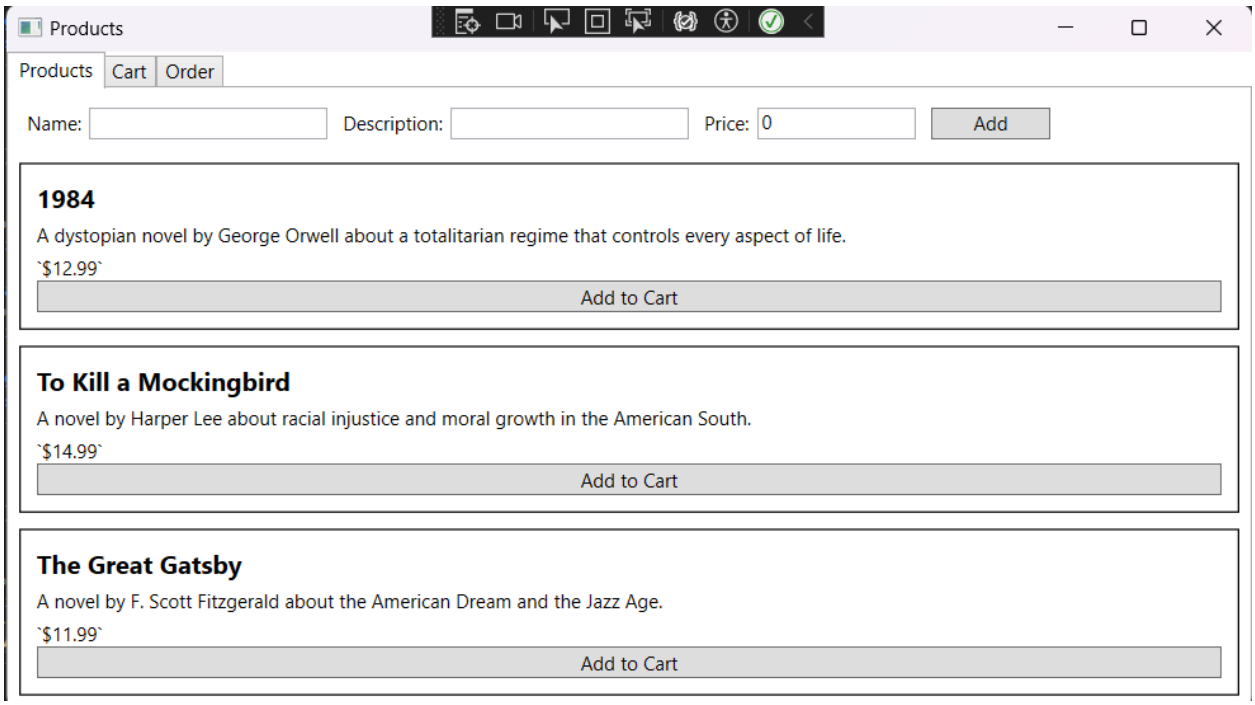
Методы:

- List<Product> GetAllProducts(): Возвращает список всех продуктов.
- void AddProduct(Product product): Добавляет новый продукт в репозиторий.

Использование:

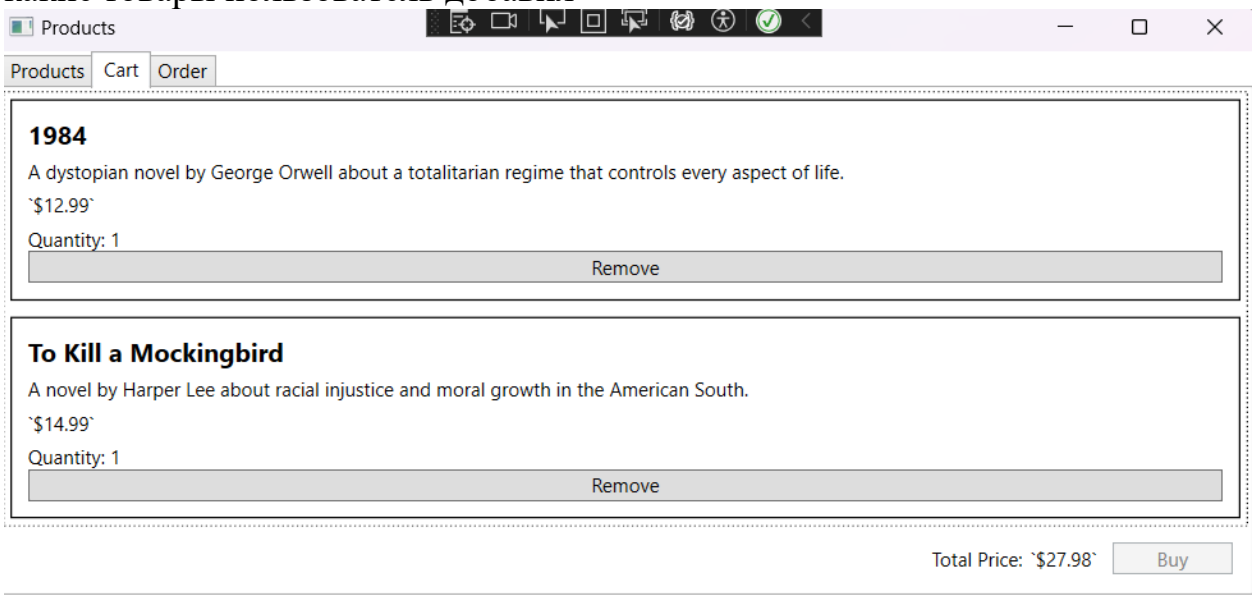
- SQLiteProductRepository реализует IProductRepository:
    - GetAllProducts(): Возвращает список всех продуктов из базы данных SQLite.
    - AddProduct(Product product): Добавляет новый продукт в базу данных SQLite.
- В ProductViewModel:
    - Используется для загрузки продуктов (LoadProducts()) и добавления новых продуктов (AddProduct()).

В проекте была использована база данных SQLite где и находились товары. В магазине присутствует лента товаров. Метод будет доставать все товары с БД и предоставлять в ленте

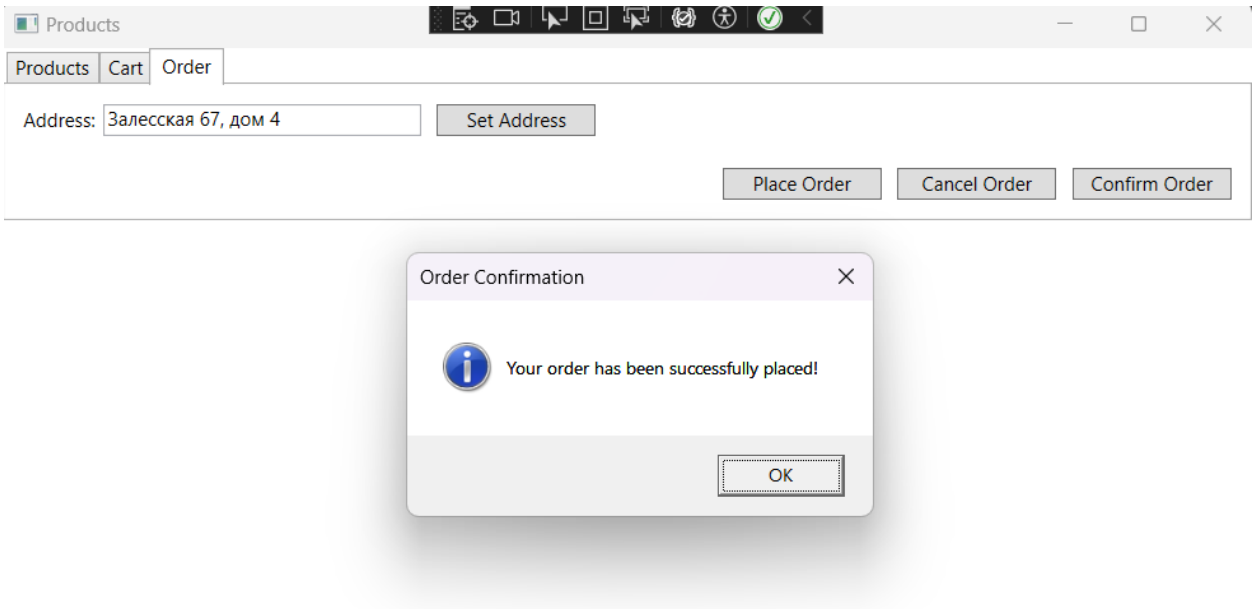


Также есть возможность выставить свой товар. Реализуется метод добавления товара в SQLite. У каждого товара есть кнопка добавить в корзину.

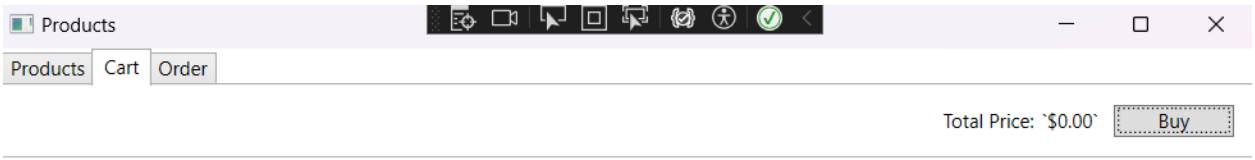
Также присутствует динамическая вкладка корзины, которая запоминает какие товары пользователь добавил



Для оформления заказа должен быть указан адрес. Адрес указывается в отдельном окне и запоминается приложением



По клавише buy происходит удаление товаров с корзины – симуляция покупки



Ответы на вопросы:

1. Что такое интерфейс?

Интерфейс — это контракт, который определяет набор методов, свойств, событий и индексаторов, которые класс или структура должны реализовать. Интерфейс сам по себе не содержит реализации этих членов, а только их сигнатуры. Классы и структуры, реализующие интерфейс, обязаны предоставить реализацию всех его членов.

2. В чем отличие интерфейсов и абстрактных классов?

Абстрактный класс	Интерфейс
Может содержать как абстрактные, так и конкретные методы.	Содержит только сигнатуры методов, свойств, событий и индексаторов.
Может иметь конструкторы.	Не может иметь конструкторы.
Может содержать поля и константы.	Не может содержать поля (за исключением констант).
Поддерживает наследование только одного класса.	Поддерживает множественное наследование.
Может предоставлять частичную реализацию.	Не может предоставлять реализацию.

3. Новые возможности интерфейсов в C# 8.

Реализация по умолчанию (Default Implementations): Интерфейсы могут содержать реализацию по умолчанию для своих методов. Это позволяет добавлять новые методы в интерфейс без нарушения существующих реализаций.

Статические члены: Интерфейсы могут содержать статические поля и методы.

Приватные методы: Интерфейсы могут содержать приватные методы, которые могут быть использованы только внутри интерфейса.

4. Возможно ли множественное наследование интерфейсов?

Да, в C# поддерживается множественное наследование интерфейсов. Класс или структура могут реализовывать несколько интерфейсов.

5. Явная реализация интерфейсов.

Явная реализация интерфейсов позволяет классу реализовывать методы интерфейса таким образом, что они не будут доступны через экземпляр класса, а только через экземпляр интерфейса. Это полезно, когда класс реализует несколько интерфейсов с одинаковыми сигнатурами методов.

6. Зачем нужны интерфейсы?

Интерфейсы играют важную роль в проектировании программного обеспечения по нескольким причинам:

- **Разделение ответственности:** Интерфейсы позволяют разделить контракты от их реализации, что упрощает проектирование и поддержку кода.

- **Гибкость и расширяемость:** Интерфейсы позволяют легко заменять реализации без изменения основного кода. Например, можно заменить одну реализацию репозитория на другую, не меняя логику приложения.
- **Тестирование:** Интерфейсы упрощают создание мок-объектов и юнит-тестирование, так как можно легко создавать заглушки для интерфейсов.
- **Множественное наследование:** Интерфейсы позволяют классам реализовывать несколько контрактов, что невозможно с абстрактными классами.
- **Инкапсуляция:** Интерфейсы позволяют скрыть детали реализации и предоставить только публичный API.