МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ

по лабораторной работе №9
по дисциплине «Объектно-ориентированное программирование»
Тема: «Использование наследования с применением интерфейсов»

Студент гр. 1302 _____ Новиков Г.В.

Преподаватель: _____ Васильев А.А.

Санкт-Петербург
2023

1. Цель работы

Изучение интерфейсов в языке C# с помощью программного продукта компании Microsoft VS 2022.

2. Анализ задачи

Необходимо:

1. Разработать структуры данных для заданной предметной области;
2. Написать программу на основе созданных в упражнении 1 структур данных.

3. Ход выполнения работы

3.1 Упражнение 1

В ходе выполнения данного упражнения написана программа, которая представляет работу склада лекарств, использует 1 абстрактный класс, 5 классов и 1 интерфейс.

3.1.1 Пошаговое описание алгоритма

1. Пользователь вводит баланс;
2. Выводится список товаров, доступных на складе для добавления в заказ;
3. Пользователь вводит выбор товара;
4. Пользователь вводит количество единиц товара;
5. Товар добавляется в заказ и удаляется со склада;
6. Выводится информация о заказе;
7. Выводится список доступных действий (добавить в заказ, удалить из заказа или отправить заказ);
8. Пользователь выбирает действие:
   8.1. Добавить лекарство в заказ – выполняются пункты 2 – 8;
   8.2. Удалить лекарство из заказа:

8.2.1. Выводится список товаров в заказе;

8.2.2. Пользователь выбирает товар;

8.2.3. Пользователь вводит количество;

8.2.4. Товар удаляется из заказа и возвращается на склад;

8.2.5. Выполняются пункты 6 – 8;

8.3. Отправить заказ:

8.3.1. Выводится информация об отправленном заказе;

8.3.2. Выводится новый баланс;

8.3.3. Программа завершается;

3.1.2 Используемые классы и методы

В программе, написанной в данном упражнении, используются следующие методы:

– System.Console.WriteLine() – служит для отображения на экране строк и значений переменных, переданных в метод в качестве параметров, с переходом на новую строку

– System.Console.ReadLine() – считывает введенную пользователем строку

– Main() – служит для запуска программы

– Drug – абстрактный класс лекарств

– Drug.Antidepressant, Drug.Barbiturate, Drug.Hormone – классы лекарств

– Order – класс заказа

– DrugStorage – класс склада лекарств

– IStorable – интерфейс для хранения товаров

Таблица 1.3.1 Описание методов класса DrugStorage

Методы	Возвращаемый тип	Модификатор доступа	Входные параметры	Назначение
Available	bool	public	string name, int quantity	Проверка наличия товаров на складе

Методы	Возвращаемый тип	Модификатор доступа	Входные параметры	Назначение
Add	void	public	Drug drug	Добавление товаров на склад
Remove	void	public	Drug drug	Удаление товаров со склада
Extract	Drug	public	string name, int quantity	Извлечение товаров со склада

Таблица 1.3.2 Описание полей класса DrugStorage

Поля	Тип	Модификатор доступа	Назначение
Drugs	List<Drug>	public	Список лекарств на складе

Таблица 1.3.3 Описание методов класса Order

Методы	Возвращаемый тип	Модификатор доступа	Входные параметры	Назначение
Send	void	Public	int balance	Отправление заказа
Add	void	Public	Drug drug	Добавление товаров в заказ
Remove	void	Public	Drug drug	Удаление товаров в заказе

Таблица 1.3.4 Описание полей класса Order

Поля	Тип	Модификатор доступа	Назначение
Drugs	List<Drug>	public	Список лекарств в заказе

Таблица 1.3.5 Описание методов интерфейса IStoring

Методы	Возвращаемый тип	Модификатор доступа	Входные параметры	Назначение
Add	void	Public	Drug drug	Добавление товаров в заказ
Remove	void	Public	Drug drug	Удаление товаров в заказе

Таблица 1.3.6 Описание полей классов Drug, Barbiturate, Antidepressant, Hormone

Поля	Тип	Модификатор доступа	Назначение
Name	string	public	Название лекарства
Quantity	int	Public	Количество
Price	int	Public	Цена
Type	string	public	Название типа

3.1.4 Диаграмма классов

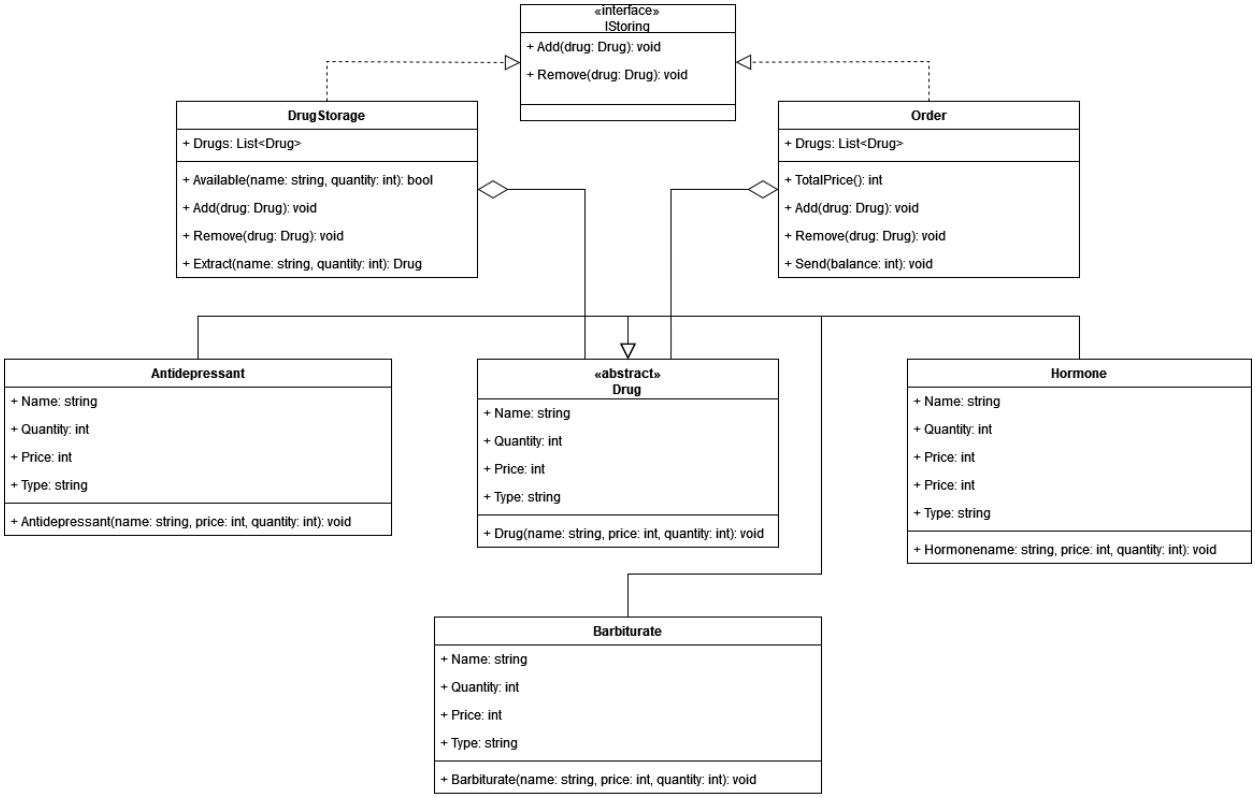

Рисунок 3.1.4. Диаграмма классов

Как мы видим по диаграмме, Order и DrugStorage реализуют интерфейс IStoring. Antidepressant, Barbiturate и Hormone реализуют абстрактный класс Drug. Между Order и Drug, а также DrugStorage и Drug существует отношение агрегации.

3.1.5 Контрольный пример

На рис.3.1.5 представлены результаты выполнения программы.

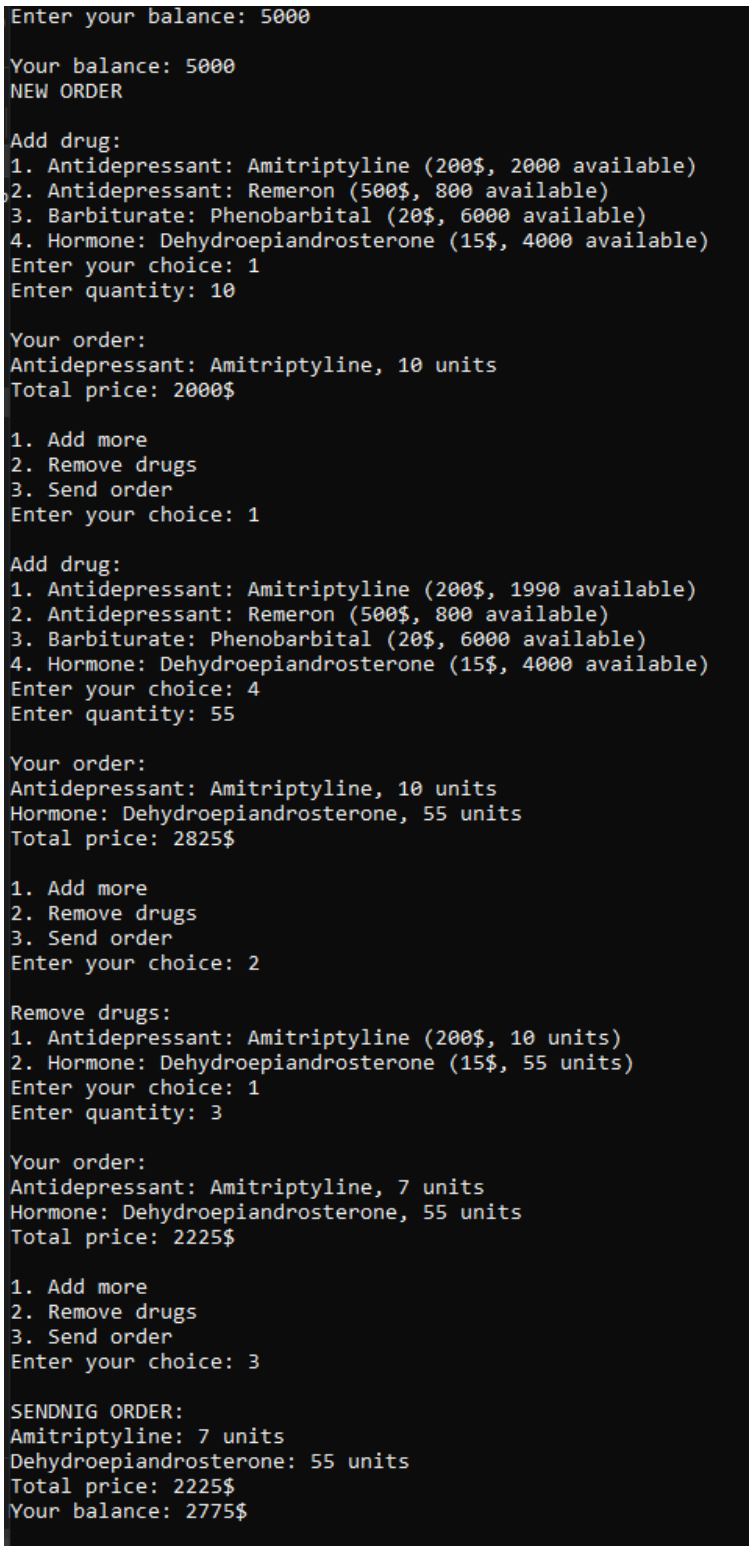

Рис.3.1.5 Контрольный пример для программы

Как видно из рисунка, пользователь вводит баланс и оформляет заказ, на экран выводится информация о заказе и товарах на складе.

4. Листинг программы

DrugStorage.cs:

```
using System;

public class DrugStorage : IStoring
{
    private List<Drug> drugs = new List<Drug>();
    public List<Drug> Drugs
    {
        get { return drugs; }
        set { drugs = value; }
    }

    public bool Available(string name, int quantity = 1)
    {
        Drug? d = Drugs.Find(d => d.Name == name);
        if (d is null) return false;
        if (d.Quantity < quantity) return false;
        return true;
    }

    public void Add(Drug drug)
    {
        Drug? d = Drugs.Find(d => d.Name == drug.Name);
        if (d is not null) d.Quantity += drug.Quantity;
        else
        {
            Drug? new_drug = Activator.CreateInstance(drug.GetType(), new object[] {
drug.Name, drug.Price, drug.Quantity }) as Drug;
            if (new_drug is not null) Drugs.Add(new_drug);
        }
    }

    public void Remove(Drug drug)
    {
        Drug? d = Drugs.Find(d => d.Name == drug.Name);
        if (d is null) throw new ArgumentException("Drug is not found");
        if (d.Quantity < drug.Quantity) throw new ArgumentException("Trying to
remove more drugs than storage has");

        if (d.Quantity == drug.Quantity) Drugs.Remove(d);
        else d.Quantity -= drug.Quantity;
    }

    public Drug Extract(string name, int quantity)
    {
        Drug? d = Drugs.Find(d => d.Name == name);
        if (d is null) throw new ArgumentException("Drug is not found");
        if (d.Quantity < quantity) throw new ArgumentException("Not enough drugs in
the storage");

        Drug? new_drug = Activator.CreateInstance(d.GetType(), new object[] {
d.Name, d.Price, quantity }) as Drug;
        if (new_drug is not null) Remove(new_drug);

        return new_drug;
    }
}
```

Order.cs:

```
using System;

public class Order : IStoring
```

```
{
    private List<Drug> drugs = new List<Drug>();
    public List<Drug> Drugs
    {
        get { return drugs; }
        set { drugs = value; }
    }

    public int TotalPrice
    {
        get
        {
            int total_price = 0;
            foreach (Drug drug in Drugs) total_price += drug.Price * drug.Quantity;
            return total_price;
        }
    }

    public void Add(Drug drug)
    {
        Drug? d = Drugs.Find(d => d.Name == drug.Name);
        if (d is not null) d.Quantity += drug.Quantity;
        else
        {
            Drug? new_drug = Activator.CreateInstance(drug.GetType(), new object[] {
drug.Name, drug.Price, drug.Quantity }) as Drug;
            if (new_drug is not null) Drugs.Add(new_drug);
        }
    }

    public void Remove(Drug drug)
    {
        Drug? d = Drugs.Find(d => d.Name == drug.Name);
        if (d is null) throw new ArgumentException("Drug is not found");
        if (d.Quantity < drug.Quantity) throw new ArgumentException("Trying to
remove more drugs than storage has");

        if (d.Quantity == drug.Quantity) Drugs.Remove(d);
        else d.Quantity -= drug.Quantity;
    }

    public void Clear()
    {
        Drugs = new List<Drug>();
    }

    public void Send(int balance)
    {
        if (Drugs.Count == 0)
        {
            Console.WriteLine("Cannot send order. Order is empty");
            return;
        }
        if (balance < TotalPrice) Console.WriteLine("Cannot send order. Balance is
too low");
        Console.WriteLine();
        Console.WriteLine("SENDNIG ORDER:");
        foreach (Drug drug in Drugs) Console.WriteLine($"{drug.Name}:
{drug.Quantity} units");
        Console.WriteLine($"Total price: {TotalPrice}$");
        Clear();
    }
}
```

IStoring.cs:

```
using System;
using System.ComponentModel.DataAnnotations;

public interface IStoring
{
    public List<Drug> Drugs { get; set; }
    void Add(Drug drug);
    void Remove(Drug drug);
}
```

Drug.cs:

```
using System;

public abstract class Drug
{
    public int Price;
    public int Quantity;
    public string Type = "";
    private string name = "";
    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    public Drug(string name, int price, int quantity)
    {
        Name = name;
        Price = price;
        Quantity = quantity;
    }


    public class Antidepressant : Drug
    {
        public Antidepressant(string name, int price, int quantity) : base(name,
price, quantity)
        {
            Type = "Antidepressant";
        }
    }


    public class Barbiturate : Drug
    {
        public Barbiturate(string name, int price, int quantity) : base(name, price,
quantity)
        {
            Type = "Barbiturate";
        }
    }


    public class Hormone : Drug
    {
        public Hormone(string name, int price, int quantity) : base(name, price,
quantity)
        {
            Type = "Hormone";
        }
    }
}
```

5. Полученные результаты

В ходе выполнения лабораторной работы была реализована иерархия классов, представляющая работу склада лекарств, был реализован ввод и вывод через консоль для создания заказа с обработкой возможных ошибок при вводе данных.

6. Выводы

В ходе выполнения данной лабораторной работы были изучены абстрактные классы, наследование и интерфейсы, была реализована иерархия классов для совершения заказа со склада лекарств.

7. Список использованной литературы

1. MSDN – сеть разработчиков Microsoft. URL: https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/abstract (дата обращения: 28.04.2023)