МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Южный федеральный университет»

ИНСТИТУТ КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

Кафедра физико-математических основ инженерного образования



**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5**

**«C# приложение»**

по курсу «Объектно-ориентированное программирование»

Выполнил:

студент гр. КТбо2-1

Фирсов А. А.

Проверила:

доцент каф. ФМОИО

Бугаева И.А.

Таганрог 2023

Содержание

[Задание 3](#_Toc154614432)

[Цель работы 3](#_Toc154614433)

[Описание алгоритма: 3](#_Toc154614434)

[Класс FileWorker 3](#_Toc154614435)

[Класс Product 4](#_Toc154614436)

[Класс AcceptanceActs.Headers 6](#_Toc154614437)

[Класс AcceptanceActs.Lines 7](#_Toc154614438)

[Класс Documents.Headers 7](#_Toc154614439)

[Класс Catalog.Suppliers 8](#_Toc154614440)

[Класс Catalog.Deliverers 9](#_Toc154614441)

[Программа 9](#_Toc154614442)

[FileWorker.cs 9](#_Toc154614443)

[Products.cs 11](#_Toc154614444)

[AcceptanceActs.Headers.cs 14](#_Toc154614445)

[AcceptanceLines.Lines.cs 17](#_Toc154614446)

[Documents.Headers.cs 18](#_Toc154614447)

[Documents.Lines.cs 22](#_Toc154614448)

[Catalog.Suppliers.cs 22](#_Toc154614449)

[Catalog.Deliverers.cs 24](#_Toc154614450)

[Program.cs 26](#_Toc154614451)

[Тестирование 27](#_Toc154614452)

Вариант 4

# **Задание**

Склад торговой организации.

# **Цель работы**

Спроектировать и разработать законченное программное приложение по одному из приведенных ниже вариантов.

# Описание

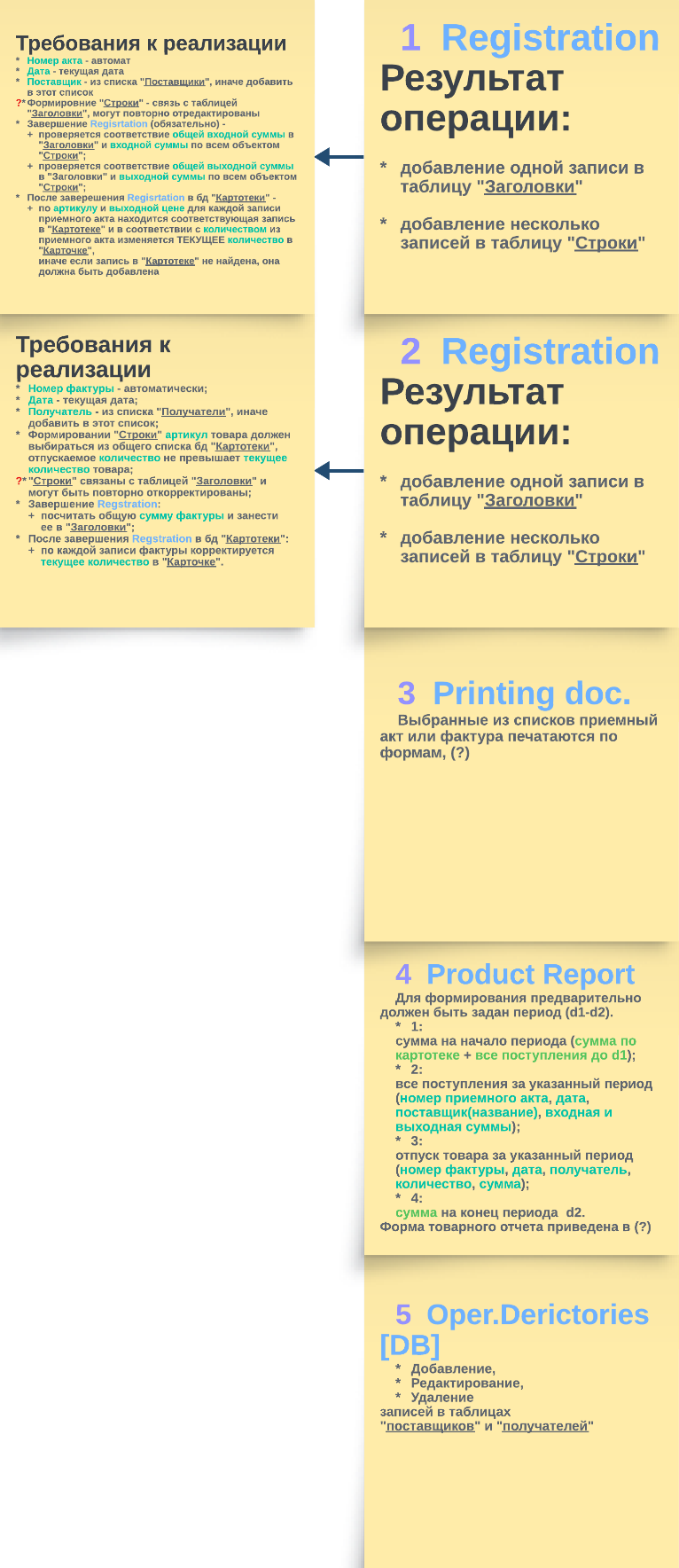


Рисунок 1 – Основной план

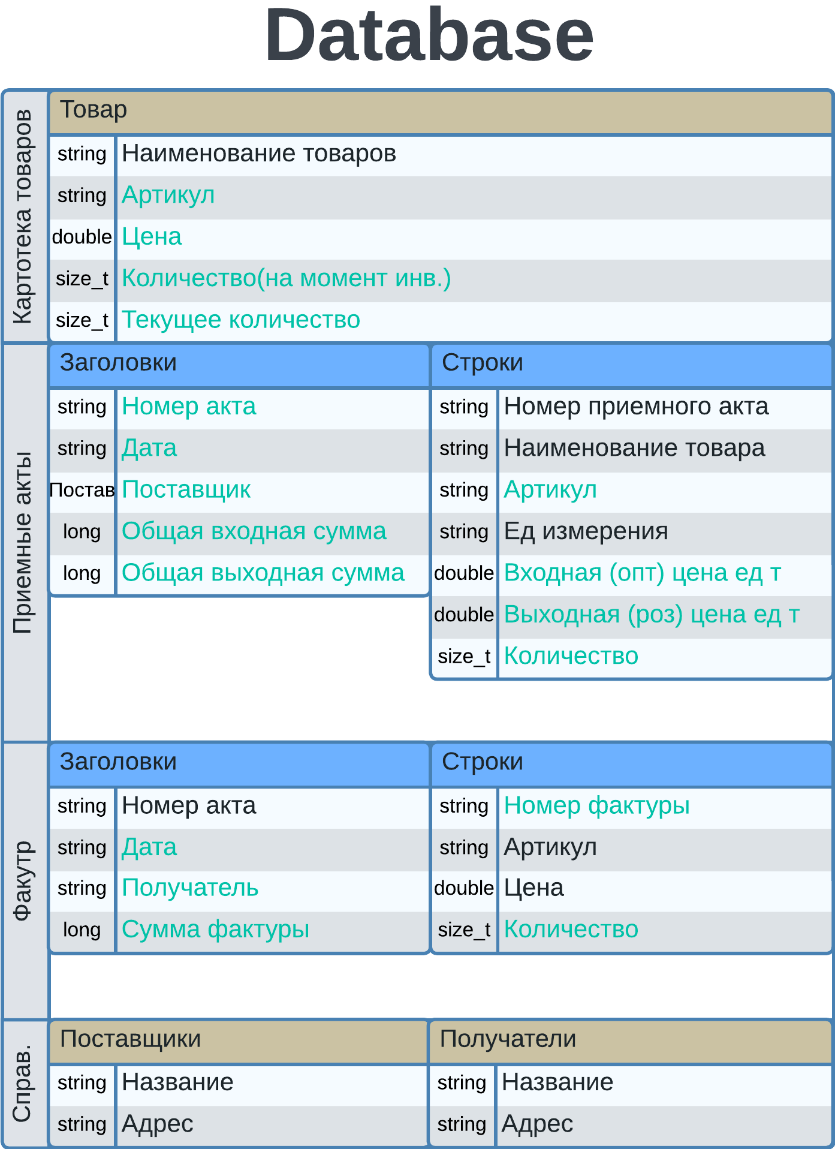


Рисунок 2 – Иерархия классов(начальный результат)

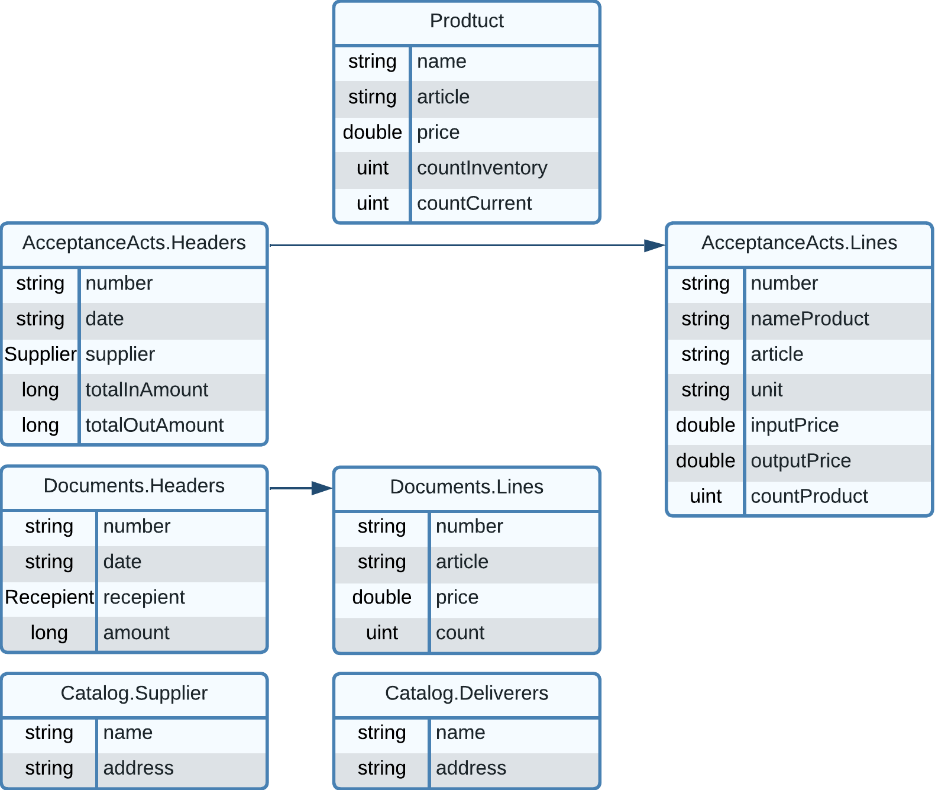


Рисунок 3 – Конечный результат

## Класс FileWorker

Представляет собой абстрактный базовый класс, который реализует общие методы для работы с файлами. Этот класс предоставляет функциональность для чтения данных из файлов и записи объектов или списков обратно в файлы. Он содержит несколько открытых методов и постоянных констант, представляющих пути к различным файлам в рамках проекта:

*ReadFileToList<T>(string path, List<T> items)* – Шаблонная функция, которая читает из указанного файла и записывает все корректные данные(объекты) в список. Items – список, в который будут добавлены корректные данные. В случае некорректного path(файла) будет вызвано исключение(не найден путь). После обработки функции вызывается сборщик муссора.

*WriteObjectToFile<T>(string path, bool append, T obj)* – Записывает или перезаписывает объект в файл. Флаг, указывающий на добавление в конец файла или перезапись – append. Объект, который будет записан в файл – obj. Вызывает исключение, если файл не найден или объект равен null.

*WriteListToFile<T>(string path, bool append, List<T> items)* – Тоже самое что и *WriteObjectToFile*, только со списокм.

Класс имеет закрытый метод:

*IsNewLine(string path*) – Возвращает значение, указывающее, является ли последний символ в файле новой строкой. Метод используется внутри методов выше, чтобы понять на какую строку записывать объект(чтобы не возникало проблем с повторным чтением файла(ошибка количества аргументов)).

Абстрактный класс также имеет две абстрактные(вспомогательная) функции для методов чтения и записи:

*FillFromLine(string[] parts)* – Если части(массив строк) корректно создаст объект экземпляра класса T, то вернет 1, иначе 0(Некорректные данные строки). В методе чтение в список, если объект(строка) подходит(строка подходит если количество аргументов корректное и правильные преобразования), то объект добавляется в список.

*StringBuild()* – Возвращает строку(поле+символ пробел+…) из объекта T.

У всех дочерних классов имеется свойство *GetLengthArgs* – Статическое свойство, представляющее ожидаемое количество аргументов при разборе строки.

## Класс Product

Класс содержит в себе название, артикул, цену, количество на момент инвентаризации, текущее количество. Наследует FileWorker и реализует интерфейс IComparable<Product> - для использования *Sort* (сортировка по полю article).

**Свойства:**

Name, Article, Price, CountInventory, CountCurrent – Свойства для доступа и модификации соответствующих полей.

**Методы:**

*Report* – Статический метод, который создает отчет о товарах на основе указанных диапазонов дат. Он читает данные из разных файлов, вычисляет суммы начала периода и конца периода, а также записывает все поступления и отпущенные товары в отчет в указанный файл.

*Product* (конструктор): Инициализирует продукт заданными значениями или значениями по умолчанию.

*~Product* (деструктор): Выводит сообщение, когда экземпляр класса уничтожается сборщиком мусора. Можно отслеживать, когда освободится память. Таким образом можно понять, что объект не уничтожается и следует намеренно вызвать сборщик мусора. Пробовал реализовывать метод Dispose от интерфейса IDisposable, однако попытка была бессмысленной.

*CompareTo* – Реализует интерфейс IComparable<Product> для сравнения продуктов на основе их артикула.

*FillFromLine* – Переопределяет метод из базового класса (FileWorker) для заполнения свойств объекта из массива строк.

*StringBuild*: Переопределяет метод из базового класса (FileWorker) для создания строки, представляющей объект.

**Использование других классов:**

Класс взаимодействует с классами AcceptanceActs и Documents для чтения данных из их соответствующих файлов.

Класс выводит информацию на консоль во время процесса создания отчета.

**Обработка Ulid:**

Тип Ulid используется для генерации и работы с уникальными идентификаторами. Использовал качестве артикула. Необходимо подключить библиотеку Ulid, чтобы работать с ней.

Для использования библиотеки Ulid подключаю через терминал(в дериктории проекта) командой: "dotnet add package Ulid --version 1.3.3".

Пришлось изменить файл .csproj чтобы сработала библиотека, добавил такую запись:

<ItemGroup>

<PackageReference Include="Ulid" Version="1.3.3" />

</ItemGroup>

Можно было воспользоваться GUID для уникального номера, но различие их в том, что Ulid сортируемый.

Сделал проверку метода *FillFromLine* (в классах Product, AcceptanceActs.Lines, Documents.Lines) на преобразования артикула в Ulid формат.

Минус этой библиотеки, что длинна номера, должна быть равной 26 символов(цифры и английский алфавит).

**Кодировка:**

Используется Encoding.GetEncoding(1251) для записи в файл, указывая использование кодировки Windows-1251.

## Класс AcceptanceActs.Headers

**Поля:** Номер приемного акта, дата, поставщик товаров, общая входная сумма, общая выходная сумма.

**Методы:**

*Register* – Статический метод для регистрации нового приемного акта.

[Номер] - номер последнего элемента списка Headers + 1;

[Дата] - сегодня;

[Поставщик] - из списка поставщиков, в противном случае поставщик из header добавляется в базу поставщиков;

Затем для каждого номера из списка listLines присвается [номер] header;

Создается список lines из базы AcceptanceActs.Lines, добвляются все элементы из списка listLines;

Нахожу общую [выходную] и [входную] сумму с помощью статической функции *Lines.TotalInputAmount(header.number, lines)* и *Lines.TotalOutputAmount(header.number, lines)*, затем присваиваю их header;

Добавляю Заголовок[header] и несколько Строк[listLines] в свои таблицы;

Под конец метода создается список listProducts из базы Products, двойной цикл, один для lines, другой для listProducts: Если артикул и цена совпали, то текущее количество у продукта увеличивается на количество товаров строки.

После цикла listProducts проверяется: Если артикулы совпали, но не совпала цена, то объект[header] перезапишется в базе, иначе если ничего не совпало, то объект[header] добавится в конец базы (использовал флаг и также сохранял индекс, если объект нужно перезаписать).

*PrintHeadersLinesToFile* – Статический метод для вывода информации о приемных актах и соответствующих строках в указанный файл.

## Класс AcceptanceActs.Lines

*TotalInputAmount* – Статический метод для расчета общей входной суммы товаров по указанному номеру приемного акта и списку строк.

*TotalOutputAmount* – Статический метод для расчета общей выходной суммы товаров по указанному номеру приемного акта и списку строк.

Реализация метода *CompareTo* для сравнения объектов по номеру приемного акта (Удобно вывести на печать сразу отсортированные строки).

**Обработка исключений:**

Обработка исключения в методе *CompareTo*, если переданный объект равен null.

Обработка исключения при разборе строки для заполнения объекта.

## Класс Documents.Headers

**Поля:**

Номер фактуры, дата фактуры, поставщик товаров, общая сумма фактуры.

**Статический метод** *Register***:**

Регистрация новой фактуры. Метод заполняет поля фактуры, сохраняет поставщика в соответствующий файл, обновляет информацию о товарах в файле продуктов, а также сохраняет информацию о заголовке и строках в соответствующих файлах.

[Номер] - номер последнего элемента списка Headers + 1;

[Дата] - текущая;

[Получатель] - из списка получателей, иначе получатель из header добавляется в базу получателей(Delivereres);

Список "Строки" – артикул из базы Продуктов и [количество](Строки) не должно превышать [текущему количеству](Продукта). Для каждого элемента в списке присваивается его номер и находим общую сумму фактуры(сумма произведения количества на цену). По каждой записи фактуры корректируется(уменьшается) текущее количество в "Карточке". Перезапись базы Продуктов.

**Вспомогательные методы:**

*SetArticulesForLines* – Метод для установки артикулов в строках фактуры. Пользователю предлагается выбрать артикулы из списка продуктов.

*PrintHeadersLinesToFile* – Статический метод для вывода информации о фактурах и соответствующих строках в указанный файл.

## Класс Catalog.Suppliers

**Поля:** поставщик, адрес

**Методы:**

*Add(Suppliers supplier)* – Добавление поставщика в базу данных.

*Replace(Suppliers supplier, Suppliers other)* – Замена информации о поставщике в базе данных.

*Delete(Suppliers supplier)* – Удаление поставщика из базы данных.

**Операторы:**

**==** и **!=** Перегрузка операторов сравнения для объектов типа Suppliers.

**Методы сравнения:**

*Equals(object obj)* – Переопределение метода *Equals* для сравнения объектов типа Suppliers.

*GetHashCode()* – Переопределение метода *GetHashCode* для получения хэш-кода.

## Класс Catalog.Deliverers

**Поля:** доставщик, адрес

**Методы:**

*Add(Deliverers deliverer)* – Добавление доставщика в базу данных.

*Replace(Deliverers deliverer, Deliverers other)* – Замена информации о доставщике в базе данных.

*Delete(Suppliers supplier)* – Удаление поставщика из базы данных.

**Операторы:**

**==** и **!=** Перегрузка операторов сравнения для объектов типа Suppliers.

**Методы сравнения:**

*Equals(object obj)* – Переопределение метода *Equals* для сравнения объектов типа Suppliers.

*GetHashCode()* – Переопределение метода *GetHashCode* для получения хэш-кода.

# Программа

## FileWorker.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

namespace TradeWarehouse

{

/// <summary>Работа с файлом</summary>

public abstract class FileWorker

{

#region Paths

public static readonly string pProduct = @"D:\GitHub\TradeWarehouse\TradeWarehouse\Products.txt";

public static readonly string pAcceptanceActsHeaders = @"D:\GitHub\TradeWarehouse\TradeWarehouse\AcceptanceActs\Headers.txt";

public static readonly string pAcceptanceActsLines = @"D:\GitHub\TradeWarehouse\TradeWarehouse\AcceptanceActs\Lines.txt";

public static readonly string pDocumentsHeaders = @"D:\GitHub\TradeWarehouse\TradeWarehouse\Documents\Headers.txt";

public static readonly string pDocumentsLines = @"D:\GitHub\TradeWarehouse\TradeWarehouse\Documents\Lines.txt";

public static readonly string pCatalogSuppliers = @"D:\GitHub\TradeWarehouse\TradeWarehouse\Catalog\Suppliers.txt";

public static readonly string pCatalogDeliverers = @"D:\GitHub\TradeWarehouse\TradeWarehouse\Catalog\Deliverers.txt";

#endregion

/// <summary>Записывает все корректные строки(объекты) из файла в спикок.</summary>

/// <exception cref="Exception"></exception>

public static void ReadFileToList<T>(string path, List<T> items) where T : FileWorker, new()

{

if (File.Exists(path))

using (StreamReader file = new StreamReader(path, Encoding.GetEncoding(1251)))

{

string line;

while ((line = file.ReadLine()) != null)

{

string[] parts = line.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

T newItem = new T();

if (newItem.FillFromLine(parts))

items.Add(newItem);

else

Console.WriteLine($"Некорректные данные в строке: <{line}>, <{path}>. Неверное количество аргументов или неверный формат строки.");

}

}

else

throw new Exception("Не найден путь " + path);

GC.Collect();

}

/// <summary>Записывает/Перезаписывает объект в файл</summary>

/// <param name="append"> true добавляние в конец файла, false - перезаписывание файла</param>

/// <exception cref="Exception"></exception>

public static void WriteObjectToFile<T>(string path, bool append, T obj) where T : FileWorker

{

if (File.Exists(path) && obj != null)

{

bool isNewLine = IsNewLine(path);

using (StreamWriter file = new StreamWriter(path, append, Encoding.GetEncoding(1251)))

{

if (!isNewLine && append)

file.WriteLine();

file.WriteLine(obj.StringBuild().ToString());

}

if (append)

Console.WriteLine($"+ Объект был добавлен в <{path}>");

else

Console.WriteLine($"+ Объект был перезаписан в <{path}>");

}

else

throw new Exception($"Не найден файл <{path}>. Или объект null.");

}

/// <summary>Записывает/Перезаписывает список в файл</summary>

/// <param name="append"> true добавляние в конец файла, false - перезаписывание файла</param>

/// <exception cref="Exception"></exception>

public static void WriteListToFile<T>(string path, bool append, List<T> items) where T: FileWorker

{

if (File.Exists(path))

{

bool isNewLine = IsNewLine(path);

using (StreamWriter file = new StreamWriter(path, append, Encoding.GetEncoding(1251)))

{

if (!isNewLine && append)

file.WriteLine();

for (int i = 0; i < items.Count; ++i)

file.WriteLine(items[i].StringBuild().ToString());

}

if (append)

Console.WriteLine($"+ Список был добавлен в <{path}>");

else

Console.WriteLine($"+ Список был перезаписан в <{path}>");

}

else

throw new Exception("Не найден путь " + path);

}

private static bool IsNewLine(string path)

{

using (StreamReader lastChar = new StreamReader(path, Encoding.GetEncoding(1251)))

{

string content = lastChar.ReadToEnd();

return string.IsNullOrEmpty(content) || content.Last() == '\n';

}

}

protected abstract bool FillFromLine(string[] parts);

public abstract StringBuilder StringBuild();

}

}

## Products.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Text;

namespace TradeWarehouse

{

internal class Product : FileWorker, IComparable<Product>

{

string name;

Ulid article;

double price;

uint countInventory;

uint countCurrent;

public static ushort GetLengthArgs { get => 5; }

/// <summary>Товарный отчет вывод в файл</summary>

/// <param name="d1">Начало периода</param>

/// <param name="d2">Конец периода</param>

public static void Report(string path, DateTime d1, DateTime d2)

{

List<Product> fileListProducts = new List<Product>();

ReadFileToList(pProduct, fileListProducts);

List<AcceptanceActs.Headers> fileListActsHeaders = new List<AcceptanceActs.Headers>();

ReadFileToList(pAcceptanceActsHeaders, fileListActsHeaders);

List<AcceptanceActs.Lines> fileListActsLines = new List<AcceptanceActs.Lines>();

ReadFileToList(pAcceptanceActsLines, fileListActsLines);

double summStart = 0;

List<AcceptanceActs.Headers> listActsHeaders = new List<AcceptanceActs.Headers>();

foreach (AcceptanceActs.Headers header in fileListActsHeaders)

{

if (header.Date < d1)

foreach (AcceptanceActs.Lines line in fileListActsLines)

if (header.Number == line.Number)

foreach (Product product in fileListProducts)

if (line.Article == product.Article)

summStart += product.Price \* (product.CountInventory - product.CountCurrent);

if (header.Date >= d1 && header.Date <= d2)

listActsHeaders.Add(header);

}

List<Documents.Headers> fileListDocsHeaders = new List<Documents.Headers>();

ReadFileToList(pDocumentsHeaders, fileListDocsHeaders);

List<Documents.Lines> fileListDocsLines = new List<Documents.Lines>();

ReadFileToList(pDocumentsLines, fileListDocsLines);

double summEnd = 0;

List<Documents.Headers> listDocsHeaders = new List<Documents.Headers>();

foreach (Documents.Headers header in fileListDocsHeaders)

{

if (header.Date < d2 && header.Date > d1)

foreach (Documents.Lines line in fileListDocsLines)

if (header.Number == line.Number)

foreach (Product product in fileListProducts)

if (line.Article == product.Article)

summEnd += product.Price \* (product.CountInventory - product.CountCurrent);

if (header.Date >= d1 && header.Date <= d2)

listDocsHeaders.Add(header);

}

using (StreamWriter fileWriter = new StreamWriter(path, false, Encoding.GetEncoding(1251)))

{

fileWriter.WriteLine("--==--==--");

fileWriter.WriteLine($"{d1.ToShortDateString()} - {d2.ToShortDateString()}");

//1. сумма на начало периода("Заработанная сумма", По всем товарам +=цену\*(кол-во - текущее кол-во))

fileWriter.WriteLine($"Сумма на начало периода: {summStart}");

//2. все поступления за указанный период

if (listActsHeaders.Count > 0)

foreach (AcceptanceActs.Headers header in listActsHeaders)

fileWriter.WriteLine(header.StringBuild());

else

fileWriter.WriteLine("Не было поступающих товаров");

//3. отпуск товара за указанный период

fileWriter.WriteLine();

if (listDocsHeaders.Count != 0)

foreach (Documents.Headers header in listDocsHeaders)

fileWriter.WriteLine(header.StringBuild());

else

fileWriter.WriteLine("Не было отпущенных товаров.");

//4. сумма на конец период

fileWriter.WriteLine($"Сумма на конец периода: {summEnd}");

fileWriter.WriteLine("--==--==--");

}

Console.WriteLine("Товарный отчет был напечатан");

fileListActsHeaders.Clear();

fileListActsLines.Clear();

fileListDocsHeaders.Clear();

fileListDocsLines.Clear();

fileListProducts.Clear();

}

public Product(string name, Ulid article, double price, uint countInventory, uint countCurrent)

{

this.name = name;

this.article = article == null ? Ulid.NewUlid() : article;

this.price = price;

this.countInventory = countInventory;

this.countCurrent = countCurrent;

}

public Product()

{

name = "-";

article = Ulid.NewUlid();

price = 0;

countInventory = 0;

countCurrent = 0;

}

~Product()

{

Console.WriteLine($"~ Product: {name}");

}

public string Name { get => name; set => name = value; }

public Ulid Article { get => article; set { article = value == null ? Ulid.NewUlid() : value; } }

public double Price { get => price; set => price = value; }

public uint CountInventory { get => countInventory; set => countInventory = value; }

public uint CountCurrent { get => countCurrent; set => countCurrent = value; }

int IComparable<Product>.CompareTo(Product other)

{

if (other is null) throw new ArgumentException("Некорректное значение параметра");

return article.CompareTo(other.article);

}

protected override bool FillFromLine(string[] parts)

{

if (parts.Length == GetLengthArgs && Ulid.TryParse(parts[1], out article)) {

name = parts[0];

price = double.Parse(parts[2]);

countInventory = uint.Parse(parts[3]);

countCurrent = uint.Parse(parts[4]);

return true;

}

else

return false;

}

public override StringBuilder StringBuild()

{

return new StringBuilder().Append(name).Append(' ').Append(article.ToString()).Append(' ').Append(price).Append(' ').Append(countInventory).Append(' ').Append(countCurrent);

}

}

}

## AcceptanceActs.Headers.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Text;

using TradeWarehouse.Catalog;

namespace TradeWarehouse.AcceptanceActs

{

internal class Headers : FileWorker, IComparable<Headers>

{

uint number;

DateTime date;

Suppliers supplier;

double totalInputAmount;

double totalOutputAmount;

//public DateTime Date { get => date; }

public static ushort GetLengthArgs { get => 6; }

/// <summary>Добавление в таблицу один Заголовок и несколько записей Строк</summary>

public static void Register(Headers header, List<Lines> listLines)

{

if (header == null || listLines == null) return;

#region Auto number

List<Headers> fileListHeader = new List<Headers>();

ReadFileToList(pAcceptanceActsHeaders, fileListHeader);

fileListHeader?.Sort();

header.number = fileListHeader.Count == 0 ? 0 : ++fileListHeader[fileListHeader.Count - 1].number;

#endregion

header.date = DateTime.Now;

#region Supplier - from fileListSuppliers, else add header.supplier in this list

List<Suppliers> fileListSuppliers = new List<Suppliers>();

ReadFileToList(pCatalogSuppliers, fileListSuppliers);

bool foundMatch = false;

foreach (var item in fileListSuppliers)

if (item.Name == header.supplier.Name && item.Address == header.supplier.Address)

{

foundMatch = true;

break;

}

if (!foundMatch) WriteObjectToFile(pCatalogSuppliers, true, header.supplier);

#endregion

//Для каждого элемента в списке присваивает его номер

for (int i = 0; i < listLines.Count; ++i)

listLines[i].Number = header.number;

List<Lines> fileListLines = new List<Lines>();

ReadFileToList(pAcceptanceActsLines, fileListLines);

//Добавление новых элементов в список

fileListLines.AddRange(listLines);

header.totalInputAmount = Lines.TotalInputAmount(header.number, fileListLines);

header.totalOutputAmount = Lines.TotalOutputAmount(header.number, fileListLines);

#region Product article and price are compared with lines, else add to Product. Overwrites Product

List<Product> fileListProducts = new List<Product>();

ReadFileToList(pProduct, fileListProducts);

foreach (var item in listLines)

{

int index = -1;

foundMatch = false;

for (int j = 0; j < fileListProducts.Count; ++j)

if (fileListProducts[j].Article == item.Article)

{

if (fileListProducts[j].Price == item.OutputPrice)

{

fileListProducts[j].CountCurrent += item.CountProduct;

foundMatch = true;

}

index = j;

}

Product itemProduct = new Product(item.NameProduct, item.Article, item.OutputPrice, item.CountProduct, item.CountProduct);

if (!foundMatch && index != -1)

{

uint countInventory = fileListProducts[index].CountInventory;

itemProduct.CountCurrent += fileListProducts[index].CountCurrent;

fileListProducts[index] = itemProduct;

fileListProducts[index].CountInventory = countInventory;

} else

if (!foundMatch && index == -1)

fileListProducts.Add(itemProduct);

}

WriteListToFile(pProduct, false, fileListProducts);

#endregion

WriteObjectToFile(pAcceptanceActsHeaders, true, header);

WriteListToFile(pAcceptanceActsLines, true, listLines);

fileListHeader.Clear();

fileListLines.Clear();

fileListProducts.Clear();

fileListSuppliers.Clear();

}

/// <summary>Печать Приемного акта в указанный файл из баз данных Headers и Lines</summary>

/// <param name="path">Перезапись файла. Если файл остутсвует, он будет создан</param>

public static void PrintHeadersLinesToFile(string path)

{

List<Headers> fileListHeaders = new List<Headers>();

ReadFileToList(pAcceptanceActsHeaders, fileListHeaders);

List<Lines> fileListLines = new List<Lines>();

ReadFileToList(pAcceptanceActsLines, fileListLines);

fileListHeaders.Sort();

fileListLines.Sort();

using (StreamWriter fileWriter = new StreamWriter(path, false, Encoding.GetEncoding(1251)))

foreach (Headers header in fileListHeaders)

{

fileWriter.WriteLine(header.StringBuild());

for (int i = 0; i < fileListLines.Count; ++i)

if (header.number == fileListLines[i].Number)

{

fileWriter.WriteLine(fileListLines[i].StringBuild().Insert(0, '\t'));

if (i != fileListLines.Count - 1 && fileListLines[i].Number != fileListLines[i + 1].Number)

break;

}

fileWriter.WriteLine();

}

Console.WriteLine("Приемный акт был напечатан");

}

public uint Number { get => number; private set => number = value; }

public DateTime Date { get => date; private set => date = value; }

public Headers(string supplier)

{

string[] parts = supplier.Replace("[", string.Empty).Replace("]", string.Empty).Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

if (parts.Length == Suppliers.GetLengthArgs)

this.supplier = new Suppliers(parts[0], parts[1]);

else

throw new Exception("Некорректное создание объекта. Возможно аргументов должно быть " + GetLengthArgs.ToString());

}

public Headers() {}

~Headers()

{

Console.WriteLine("~ AcceptanceActs.Headers: "+number.ToString());

}

int IComparable<Headers>.CompareTo(Headers other)

{

if (other is null) throw new ArgumentException("Некорректное значение параметра");

return number.CompareTo(other.number);

}

protected override bool FillFromLine(string[] parts)

{

if (parts.Length == GetLengthArgs

&& uint.TryParse(parts[0], out number)

&& DateTime.TryParse(parts[1], out date)

&& double.TryParse(parts[4], out totalInputAmount)

&& double.TryParse(parts[5], out totalOutputAmount))

{

supplier = new Suppliers(parts[2].Replace("[", string.Empty), parts[3].Replace("]", string.Empty));

return true;

}

else

return false;

}

public override StringBuilder StringBuild()

{

return new StringBuilder().Append(number).Append(" ").Append(date.ToShortDateString()).Append(" [").Append(supplier.Name).Append(" ").Append(supplier.Address).Append("] ").Append(totalInputAmount).Append(" ").Append(totalOutputAmount);

}

}

}

## AcceptanceLines.Lines.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace TradeWarehouse.AcceptanceActs

{

internal class Lines : FileWorker, IComparable<Lines>

{

uint number;

string nameProduct;

Ulid article;

string unit;

double inputPrice;

double outputPrice;

uint countProduct;

public static ushort GetLengthArgs { get => 7; }

public static double TotalInputAmount(uint number, List<Lines> lines)

{

double amount = 0;

bool flag = false;

for (int i = 0; i < lines.Count; ++i)

if (lines[i].number == number)

{

amount += lines[i].inputPrice \* lines[i].countProduct;

flag = true;

}

return flag ? amount : -1;

}

public static double TotalOutputAmount(uint number, List<Lines> lines)

{

double amount = 0;

bool flag = false;

for (int i = 0; i < lines.Count; ++i)

if (lines[i].number == number)

{

amount += lines[i].outputPrice \* lines[i].countProduct;

flag = true;

}

return flag ? amount : -1;

}

public Lines() {}

public Lines(string nameProduct, Ulid article, string unit, double inputPrice, double outputPrice, uint countProduct)

{

this.nameProduct = nameProduct;

this.article = article;

this.unit = unit;

this.inputPrice = inputPrice;

this.outputPrice = outputPrice;

this.countProduct = countProduct;

}

public string NameProduct { get => nameProduct; set => nameProduct = value; }

public uint Number { get => number; set => number = value; }

public Ulid Article { get => article; }

public double InputPrice { get => inputPrice; set => inputPrice = value; }

public double OutputPrice { get => outputPrice; set => outputPrice = value; }

public uint CountProduct { get => countProduct; set => countProduct = value; }

int IComparable<Lines>.CompareTo(Lines other)

{

if (other is null) throw new ArgumentException("Некорректное значение параметра");

return number.CompareTo(other.number);

}

protected override bool FillFromLine(string[] parts)

{

if (parts.Length == GetLengthArgs

&& uint.TryParse(parts[0], out number)

&& Ulid.TryParse(parts[2], out article)

&& double.TryParse(parts[4], out inputPrice)

&& double.TryParse(parts[5], out outputPrice)

&& uint.TryParse(parts[6], out countProduct))

{

nameProduct = parts[1];

unit = parts[3];

return true;

}

else

return false;

}

public override StringBuilder StringBuild()

{

return new StringBuilder().Append(number).Append(" ").Append(nameProduct).Append(" ").Append(article.ToString()).Append(" ").Append(unit).Append(" ").Append(inputPrice).Append(" ").Append(outputPrice).Append(" ").Append(countProduct);

}

}

}

## Documents.Headers.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Text;

using TradeWarehouse.Catalog;

namespace TradeWarehouse.Documents

{

internal class Headers : FileWorker, IComparable<Headers>

{

uint number;

DateTime date;

Deliverers deliverer;

double amount;

public static ushort GetLengthArgs { get => 5; }

/// <summary> Добавление в таблицу один Заголовок и несколько записей Строк </summary>

public static void Register(Headers header, List<Lines> listLines)

{

if (header == null || listLines == null) return;

#region Auto number

List<Headers> fileListHeader = new List<Headers>();

ReadFileToList(pDocumentsHeaders, fileListHeader);

fileListHeader?.Sort();

header.number = fileListHeader.Count == 0 ? 0 : ++fileListHeader[fileListHeader.Count - 1].number;

#endregion

header.date = DateTime.Now;

#region Deliverers - from fileListDeliverers, else add header.deliverer in this list

List<Deliverers> fileListDeliverers = new List<Deliverers>();

ReadFileToList(pCatalogDeliverers, fileListDeliverers);

bool foundMatch = false;

foreach (Deliverers deliverer in fileListDeliverers)

if (deliverer.Name == header.deliverer.Name && deliverer.Address == header.deliverer.Address)

{

foundMatch = true;

break;

}

if (!foundMatch)

WriteObjectToFile(pCatalogDeliverers, true, header.deliverer);

#endregion

List<Product> fileListProducts = new List<Product>();

ReadFileToList(pProduct, fileListProducts);

SetArticulesForLines(listLines, fileListProducts);

//Для каждого элемента в списке присваивает его номер

//+ общ сумму фактуры += цене Строки

foreach (Lines lines in listLines)

{

lines.Number = header.number;

header.amount += lines.Price \* lines.Count;

}

//По каждой записи фактуры корректируется текущее количество в "Карточке"

foundMatch = false;

foreach (Lines lines in listLines)

for (int i = 0; i < fileListProducts.Count; ++i)

if (lines.Article == fileListProducts[i].Article)

{

fileListProducts[i].CountCurrent -= lines.Count;

foundMatch = true;

break;

}

if (foundMatch)

WriteListToFile(pProduct, false, fileListProducts);

WriteObjectToFile(pDocumentsHeaders, true, header);

WriteListToFile(pDocumentsLines, true, listLines);

fileListHeader.Clear();

fileListDeliverers.Clear();

fileListProducts.Clear();

}

private static void SetArticulesForLines(List<Lines> listLines, List<Product> fileListProducts)

{

int listCount = listLines.Count;

Console.WriteLine($"Выберете {listCount.ToString()} артикул/а/ов из списка продуктов:");

foreach (Product product in fileListProducts)

{

Console.WriteLine($"\t{product.Name} {product.Article.ToString()} текущее количество: {product.CountCurrent.ToString()}");

}

Console.WriteLine("(Значение не должно превышать текущее количество товара с отпускаемым количеством)");

int countDeleted = 0;

for (int i = 0; i < listCount; ++i)

{

Console.Write($"Отпускаемое количество = {listLines[i - countDeleted].Count.ToString()}, Артикул: ");

string articule = Console.ReadLine();

Ulid ulid;

int countCurrentProduct = -1;

while (!Ulid.TryParse(articule, out ulid) ||

(countCurrentProduct = Contains(ulid, fileListProducts)) == -1)

{

Console.Write("Не верный формат или уже есть в списке продуктов, попробуйте снова: ");

articule = Console.ReadLine();

}

if (listLines[i - countDeleted].Count <= countCurrentProduct)

{

listLines[i - countDeleted].Article = ulid;

Console.WriteLine($"\t{(i + 1).ToString()} : {articule} добавлен");

}

else

{

listLines.RemoveAt(i - countDeleted);

++countDeleted;

Console.WriteLine($"\t{(i + 1).ToString()} : {articule} не добавлен");

}

}

}

/// <summary>Печать Фактуры в указанный файл из баз данных Headers и Lines</summary>

/// <param name="path">Перезапись файла. Если файл остутсвует, он будет создан</param>

public static void PrintHeadersLinesToFile(string path)

{

List<Headers> fileListHeaders = new List<Headers>();

ReadFileToList(pDocumentsHeaders, fileListHeaders);

List<Lines> fileListLines = new List<Lines>();

ReadFileToList(pDocumentsLines, fileListLines);

fileListHeaders.Sort();

fileListLines.Sort();

using (StreamWriter fileWriter = new StreamWriter(path, false, Encoding.GetEncoding(1251)))

foreach (Headers header in fileListHeaders)

{

fileWriter.WriteLine(header.StringBuild());

for (int i = 0; i < fileListLines.Count; ++i)

if (header.number == fileListLines[i].Number)

{

fileWriter.WriteLine(fileListLines[i].StringBuild().Insert(0, '\t'));

if (i != fileListLines.Count - 1 && fileListLines[i].Number != fileListLines[i + 1].Number)

break;

}

fileWriter.WriteLine();

}

Console.WriteLine("Фактура была напечатана");

}

private static int Contains(Ulid ulid, List<Product> fileListProducts)

{

foreach (var product in fileListProducts)

if (product.Article == ulid)

return (int)product.CountCurrent;

return -1;

}

public Headers(string deliverer)

{

string[] parts = deliverer.Replace("[", string.Empty).Replace("]", string.Empty).Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

if (parts.Length == Deliverers.GetLengthArgs)

this.deliverer = new Deliverers(parts[0], parts[1]);

else

throw new Exception("Некорректное создание объекта. Возможно аргументов должно быть " + GetLengthArgs.ToString());

}

public Headers() { }

~Headers()

{

Console.WriteLine("~ Documents.Headers: " + number.ToString());

}

public DateTime Date { get => date; }

public uint Number { get => number; }

int IComparable<Headers>.CompareTo(Headers other)

{

if (other is null) throw new ArgumentException("Некорректное значение параметра");

return number.CompareTo(other.number);

}

protected override bool FillFromLine(string[] parts)

{

if (parts.Length == GetLengthArgs

&& uint.TryParse(parts[0], out number)

&& DateTime.TryParse(parts[1], out date)

&& double.TryParse(parts[4], out amount))

{

deliverer = new Deliverers(parts[2].Replace("[", string.Empty), parts[3].Replace("]", string.Empty));

return true;

}

else

return false;

}

public override StringBuilder StringBuild()

{

return new StringBuilder().Append(number).Append(" ").Append(date.ToShortDateString()).Append(" [").Append(deliverer.Name).Append(" ").Append(deliverer.Address).Append("] ").Append(amount);

}

}

}

## Documents.Lines.cs

using System;

using System.Text;

namespace TradeWarehouse.Documents

{

internal class Lines : FileWorker, IComparable<Lines>

{

uint number;

Ulid article;

double price;

uint count;

public static ushort GetLengthArgs { get => 4; }

public Lines() { }

public Lines(double price, uint count)

{

this.price = price;

this.count = count;

}

public uint Number { get => number; set => number = value; }

public Ulid Article { get => article; set => article = value; }

public double Price { get => price; set => price = value; }

public uint Count { get => count; set => count = value; }

int IComparable<Lines>.CompareTo(Lines other)

{

if (other is null) throw new ArgumentException("Некорректное значение параметра");

return number.CompareTo(other.number);

}

protected override bool FillFromLine(string[] parts)

{

if (parts.Length == GetLengthArgs

&& uint.TryParse(parts[0], out number)

&& Ulid.TryParse(parts[1], out article)

&& double.TryParse(parts[2], out price)

&& uint.TryParse(parts[3], out count))

{

return true;

}

else

return false;

}

public override StringBuilder StringBuild()

{

return new StringBuilder().Append(number).Append(" ").Append(article.ToString()).Append(" ").Append(price).Append(" ").Append(count);

}

}

}

## Catalog.Suppliers.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace TradeWarehouse.Catalog

{

internal class Suppliers : FileWorker

{

string name;

string address;

public static ushort GetLengthArgs { get => 2; }

public Suppliers() { }

public Suppliers(string name, string address)

{

this.name = name;

this.address = address;

}

public string Name { get => name; set => name = value; }

public string Address { get => address; set => name = value; }

public void Add(Suppliers supplier)

{

WriteObjectToFile(pCatalogSuppliers, true, supplier);

Console.WriteLine($"Поставщик [{supplier.StringBuild()}] добавлен в базу.");

}

public void Replace(Suppliers supplier, Suppliers other)

{

List<Suppliers> fileList = new List<Suppliers>();

ReadFileToList(pCatalogSuppliers, fileList);

bool notFoundMatch = true;

for (int i = 0; i < fileList.Count && notFoundMatch; ++i)

{

if (fileList[i] == supplier)

{

fileList[i].name = other.name;

fileList[i].address = other.address;

notFoundMatch = false;

}

}

if (notFoundMatch)

Console.WriteLine($"Не удалось найти поставщика [{other.StringBuild()}] в базе.");

else

{

WriteListToFile(pCatalogSuppliers, false, fileList);

Console.WriteLine($"Поставщик [{supplier.StringBuild()}] изменен на [{other.StringBuild()}].");

}

}

public void Delete(Suppliers supplier)

{

List<Suppliers> fileList = new List<Suppliers>();

ReadFileToList(pCatalogSuppliers, fileList);

bool notFoundMatch = true;

for (int i = 0; i < fileList.Count && notFoundMatch; ++i)

{

if (fileList[i] == supplier)

{

fileList.RemoveAt(i);

notFoundMatch = false;

}

}

if (notFoundMatch)

Console.WriteLine($"Не удалось найти поставщика [{supplier.StringBuild()}] в базе.");

else

{

WriteListToFile(pCatalogSuppliers, false, fileList);

Console.WriteLine($"Поставщик [{supplier.StringBuild()}] удален из базы.");

}

}

public static bool operator ==(Suppliers sup1, Suppliers sup2)

{

return sup1.name == sup2.name && sup1.address == sup2.address;

}

public static bool operator !=(Suppliers sup1, Suppliers sup2)

{

return !(sup1 == sup2);

}

public override bool Equals(object obj)

{

return obj is Suppliers suppliers &&

name == suppliers.name &&

address == suppliers.address;

}

public override int GetHashCode()

{

int hashCode = -99900638;

hashCode = (hashCode \* -1521134295) + EqualityComparer<string>.Default.GetHashCode(name);

hashCode = (hashCode \* -1521134295) + EqualityComparer<string>.Default.GetHashCode(address);

return hashCode;

}

protected override bool FillFromLine(string[] parts)

{

if (parts.Length == GetLengthArgs) {

name = parts[0];

address = parts[1];

return true;

}

else

return false;

}

public override StringBuilder StringBuild()

{

return new StringBuilder().Append(name).Append(" ").Append(address);

}

}

}

## Catalog.Deliverers.cs

using System.Collections.Generic;

using System;

using System.Text;

namespace TradeWarehouse.Catalog

{

internal class Deliverers : FileWorker

{

string name;

string address;

public static ushort GetLengthArgs { get => 2; }

public Deliverers(string name, string address)

{

this.name = name;

this.address = address;

}

public Deliverers() { }

public string Name { get => name; set => name = value; }

public string Address { get => address; set => address = value; }

public void Add(Deliverers deliverer)

{

WriteObjectToFile(pCatalogDeliverers, true, deliverer);

Console.WriteLine($"Получатель [{deliverer.StringBuild()}] добавлен в базу.");

}

public void Replace(Deliverers deliverer, Deliverers other)

{

List<Deliverers> fileList = new List<Deliverers>();

ReadFileToList(pCatalogDeliverers, fileList);

bool notFoundMatch = true;

for (int i = 0; i < fileList.Count && notFoundMatch; ++i)

{

if (fileList[i] == deliverer)

{

fileList[i].name = other.name;

fileList[i].address = other.address;

notFoundMatch = false;

}

}

if (notFoundMatch)

Console.WriteLine($"Не удалось найти получателя [{other.StringBuild()}] в базе.");

else

{

WriteListToFile(pCatalogDeliverers, false, fileList);

Console.WriteLine($"Получатель [{deliverer.StringBuild()}] изменен на [{other.StringBuild()}].");

}

}

public void Delete(Deliverers deliverer)

{

List<Deliverers> fileList = new List<Deliverers>();

ReadFileToList(pCatalogDeliverers, fileList);

bool notFoundMatch = true;

for (int i = 0; i < fileList.Count && notFoundMatch; ++i)

{

if (fileList[i] == deliverer)

{

fileList.RemoveAt(i);

notFoundMatch = false;

}

}

if (notFoundMatch)

Console.WriteLine($"Не удалось найти получателя [{deliverer.StringBuild()}] в базе.");

else

{

WriteListToFile(pCatalogSuppliers, false, fileList);

Console.WriteLine($"Получатель [{deliverer.StringBuild()}] удален из базы.");

}

}

public static bool operator ==(Deliverers dev1, Deliverers dev2)

{

return dev1.name == dev2.name && dev1.address == dev2.address;

}

public static bool operator !=(Deliverers dev1, Deliverers dev2)

{

return !(dev1 == dev2);

}

public override bool Equals(object obj)

{

return obj is Deliverers deliverer&&

name == deliverer.name &&

address == deliverer.address;

}

public override int GetHashCode()

{

int hashCode = -99900638;

hashCode = (hashCode \* -1521134295) + EqualityComparer<string>.Default.GetHashCode(name);

hashCode = (hashCode \* -1521134295) + EqualityComparer<string>.Default.GetHashCode(address);

return hashCode;

}

protected override bool FillFromLine(string[] parts)

{

if (parts.Length == GetLengthArgs) {

name = parts[0];

address = parts[1];

return true;

}

else

return false;

}

public override StringBuilder StringBuild()

{

return new StringBuilder().Append(name).Append(" ").Append(address);

}

}

}

## Program.cs

using System;

using System.Collections.Generic;

namespace TradeWarehouse

{

internal class Program

{

static void Main()

{

try

{

List<Product> listProduct = new List<Product>();

FileWorker.ReadFileToList(FileWorker.pProduct, listProduct);

foreach (Product product in listProduct)

Console.WriteLine(product.Article.ToString());

Console.WriteLine("List<Product> = " + listProduct.Count.ToString());

List<AcceptanceActs.Lines> listLines = new List<AcceptanceActs.Lines>

{

new AcceptanceActs.Lines("Товар3", Ulid.Parse("01HJ4FEMAMEVXXC0GAWXSKJA5J"), "шт", 9.5, 24.1, 100),

new AcceptanceActs.Lines("Товар4", Ulid.Parse("01HJ4FEMAMC9P4SPWKAYP2ZR4N"), "шт", 90.49, 100.59, 1902)

};

AcceptanceActs.Headers.Register(header: new AcceptanceActs.Headers(supplier: "[Александр Черкесск]"), listLines: listLines);

Console.WriteLine("========");

List<Documents.Lines> listDocLines = new List<Documents.Lines>

{

new Documents.Lines(price: 10.1, count: 15),

new Documents.Lines(price: 20.2, count: 20),

new Documents.Lines(price: 30.3, count: 25),

new Documents.Lines(price: 40.4, count: 30)

};

Documents.Headers.Register(new Documents.Headers(deliverer: "[Артем Чекрекесск,Ленина,1]"), listDocLines);

AcceptanceActs.Headers.PrintHeadersLinesToFile(@"C:\Users\sant6\Desktop\AcceptanceActs.txt");

Documents.Headers.PrintHeadersLinesToFile(@"C:\Users\sant6\Desktop\Documents.txt");

Product.Report(@"C:\Users\sant6\Desktop\Report.txt", new DateTime(2023, 12, 19), new DateTime(2023, 12, 23));

}

catch (Exception ex)

{

Console.WriteLine("<!>-" + ex.Message);

}

}

}

}

# База данных Склад торговой организации

В качестве базы данных воспользовался блокнотом.

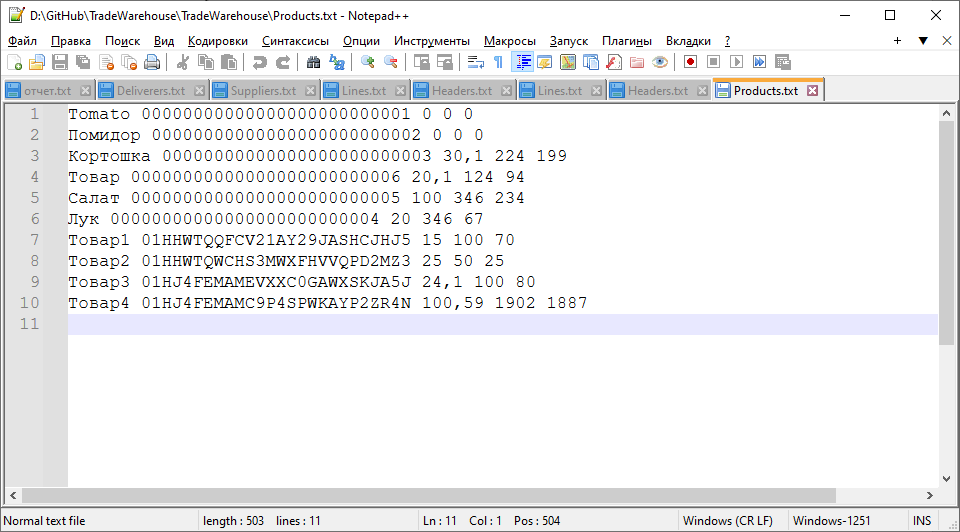


Рисунок 4 – Product.txt

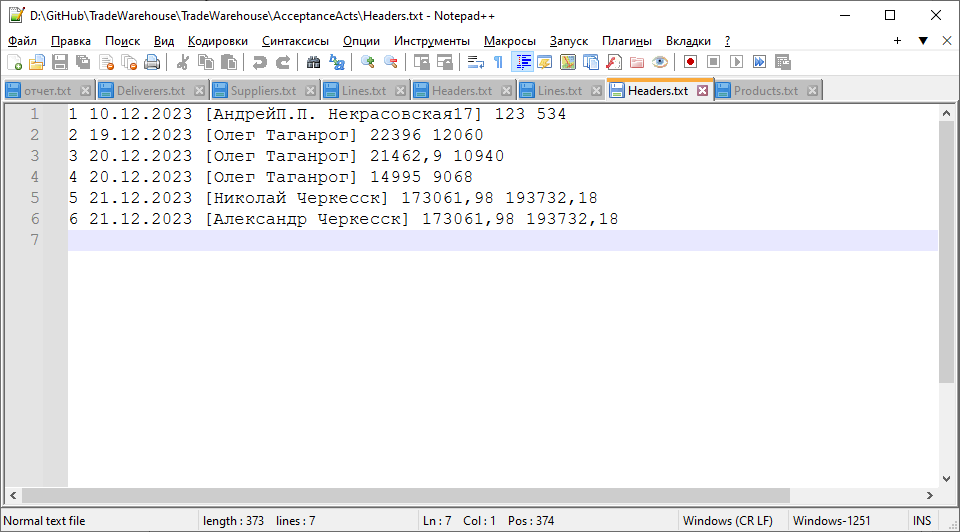


Рисунок 5 – AcceptanceActs\Headers.txt

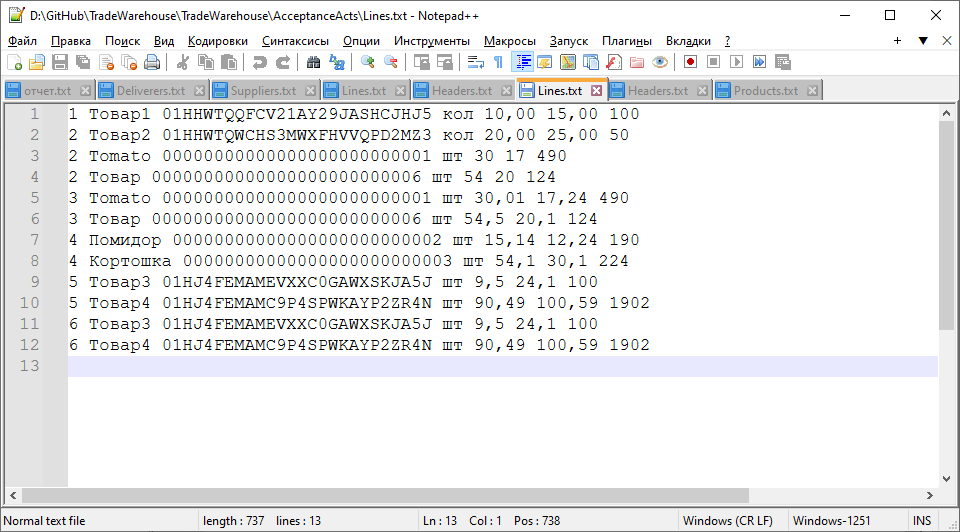


Рисунок 6 – AcceptanceActs\Lines.txt

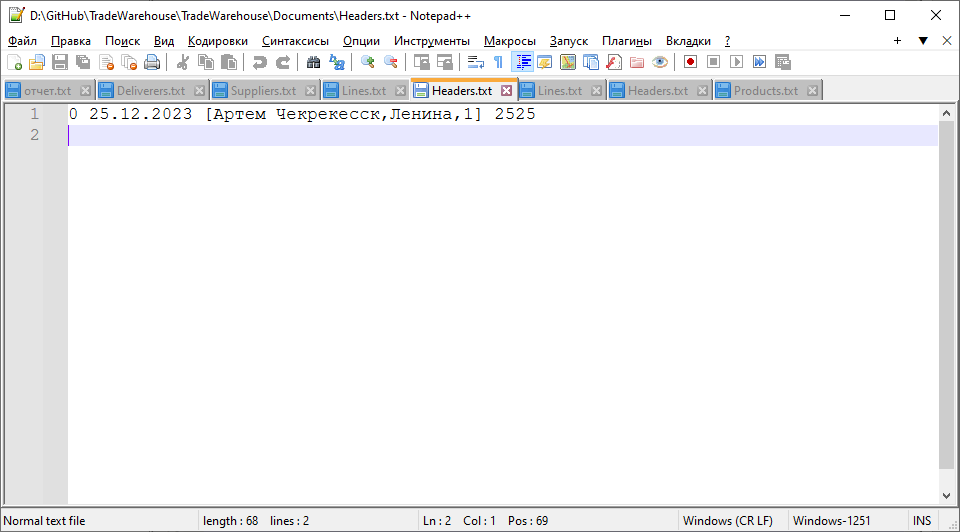


Рисунок 7 – Documents\Headers.txt

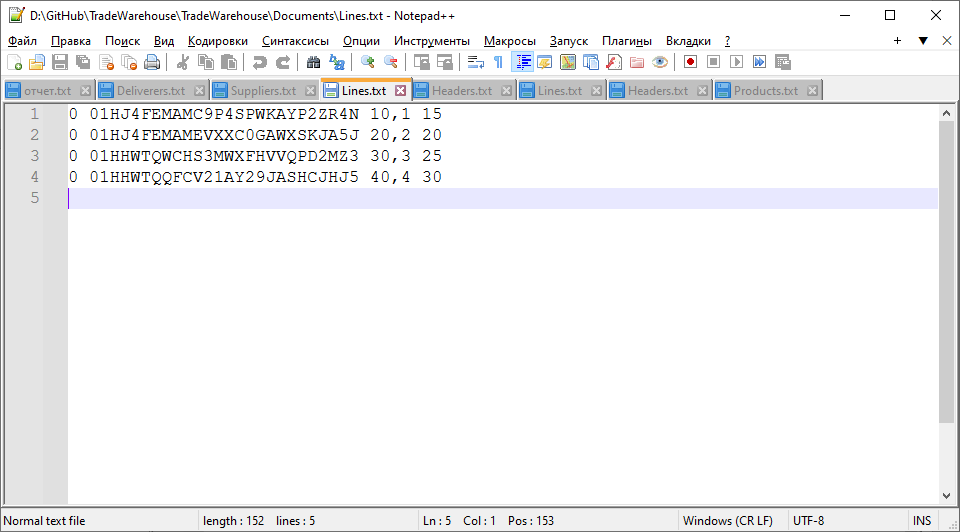


Рисунок 8 – Documents\Lines.txt

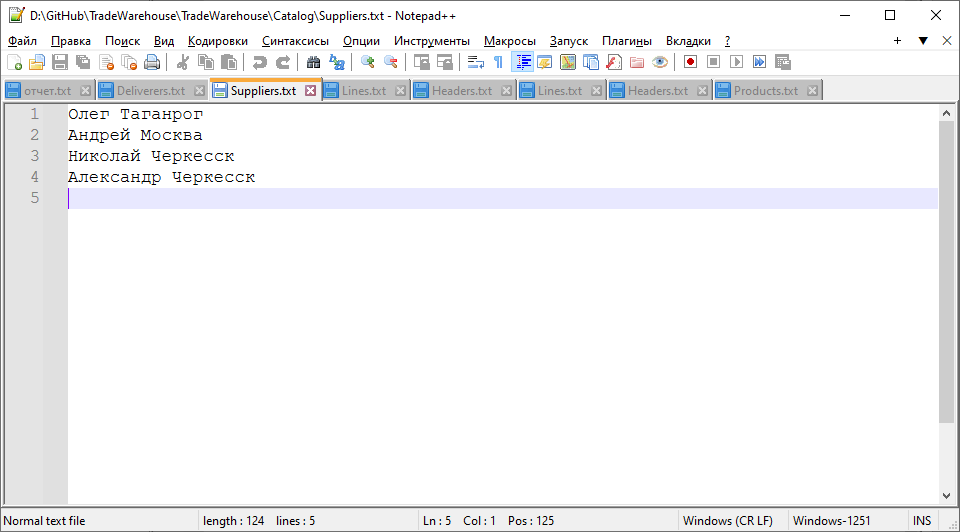


Рисунок 9 – Catalog\Suppliers.txt

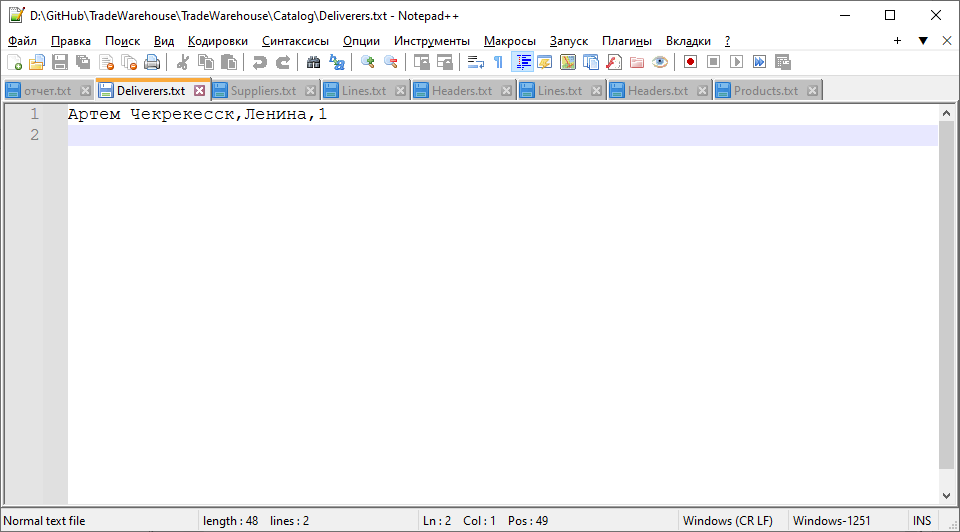


Рисунок 10 – Catalog\Deliverers.txt

# Тестирование

### *AcceptanceActs.Headers.Register:*

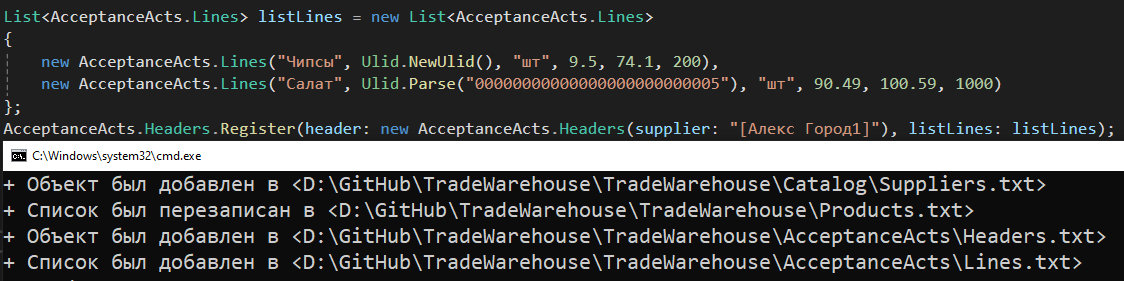


Рисунок 11 – Сработал метод *AcceptanceActs.Headers.Register*

Соблюдая план (рисунок 1), на рисунке 11 видно вывод выполнения программы после функции.

**Результат:**

Поставщик [Алекс Город1] был добавлен в список поставщиков(рисунок 15), потому что в базе (рисунок 9) такого нет.

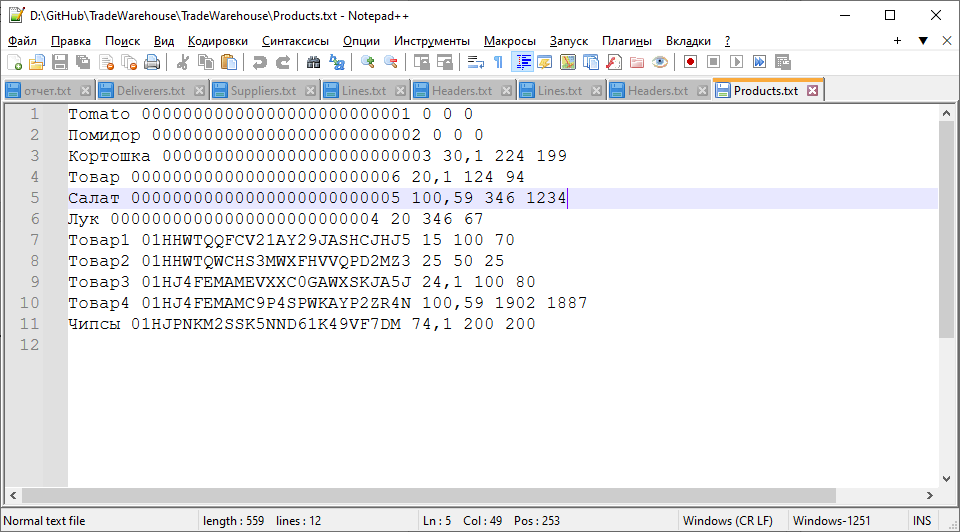


Рисунок 12 – “Список был перезаписан”

На рисунке 12 видно, как обновился список, изменилась 5 и 11 строка в сравнении с рисунком 4. “Салат” имел текущее количество 234, однако после регистрации приемного акта, где количество этого же продукта имело 1000, теперь стало быть равным 1234. Также добавился новый продукт “Чипсы”

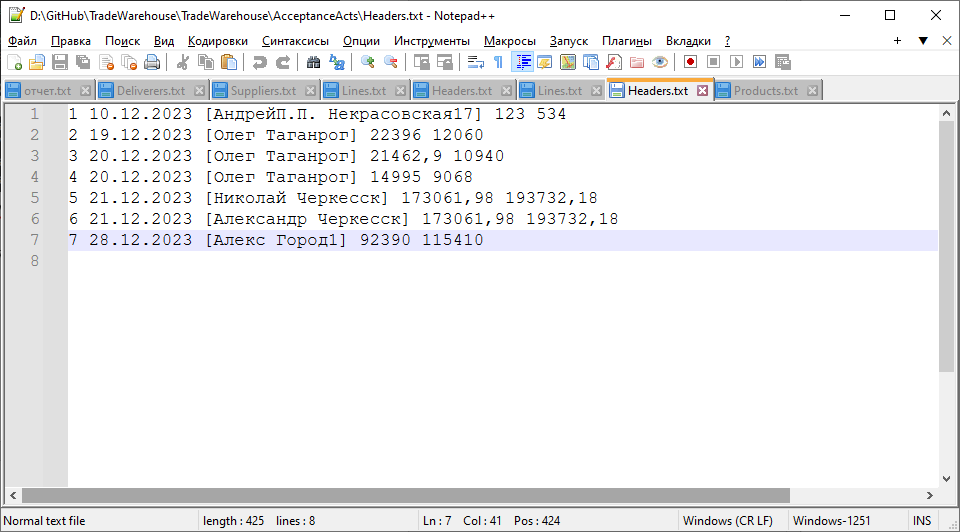


Рисунок 13 – “Объект был добавлен”

Седьмой строкой добавилась новая запись(рисунок 13).

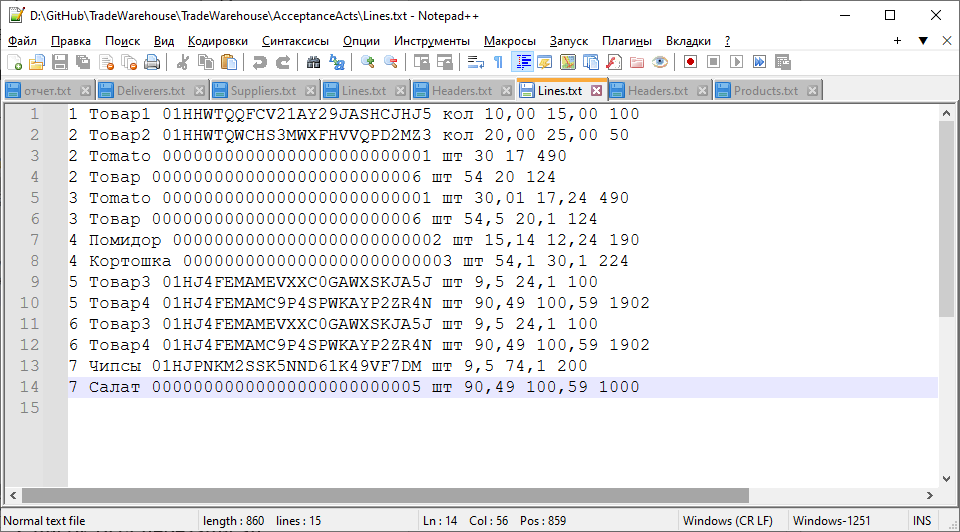


Рисунок 14 – “Список был добавлен”

На рисунке 14 добавился список номеров 7, как у заголовка(рисунок 13).

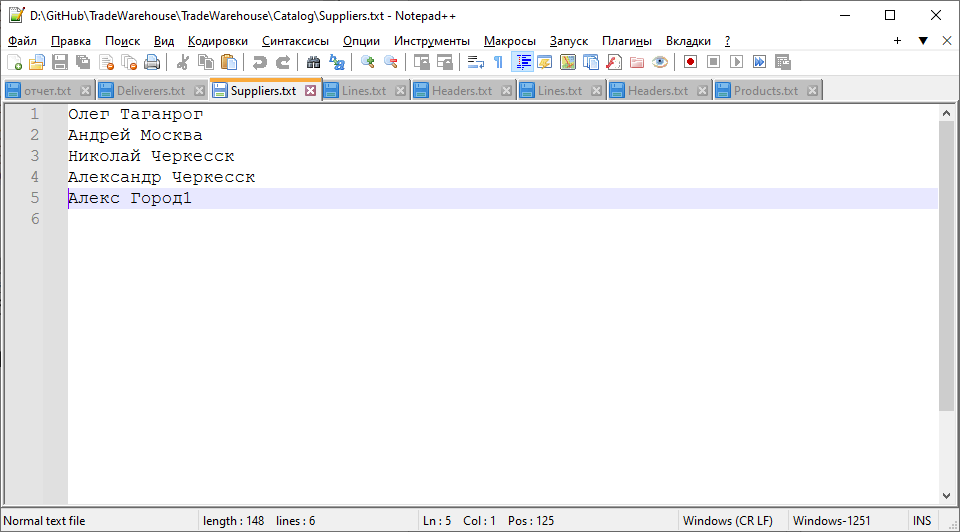


Рисунок 15 – Поставщики