**Федеральное государственное бюджетное профессиональное образовательное учреждение**

**Торжокский политехнический колледж**

**Федерального агентства по государственным резервам**

**ОТЧЁТ**

о зимней работе

**по профессиональному модулю МДК.01.01**

«**РАЗРАБОТКА ПРОГРАМНЫХ МОДУЛЕЙ**»

наименование профессионального модуля

Специальность ***09.02.07 «Информационные системы и программирование»***

Период с «28» декабря 2024г. по «12» января 2025г.

Выполнили студенты

(ФИО, подпись)

|  |  |
| --- | --- |
| Преподаватель  Бойков А.М  (ФИО, подпись) | оценка |

2024

Оглавление

[Введение 3](#_Toc189758319)

[1. Практическая значимость программного средства 4](#_Toc189758320)

[2. Функциональные требования к программному средству 6](#_Toc189758321)

[3. Модульная структура проектируемого программного средства 8](#_Toc189758322)

[4. Описание алгоритмов 11](#_Toc189758323)

[5. Разработка проекта в Visual Studio C# 15](#_Toc189758324)

# **Введение**

Цель работы – получить практический опыт парного программирования и разработки модульной структуры программы.

Задачи:

1) выявить практическую значимость проектируемого программного средства (далее – ПС) (с точки зрения использования, с точки зрения развития продуктов большего масштаба)

2) обосновать функции разрабатываемого программного средства и варианты его использования (какие задачи позволит решить и кому);

3) спроектировать модульную структуру программного средства (структура классов: данные и методы);

4) описание нескольких алгоритмов методов отвечающие непосредственно за решение поставленной задачи;

5) разработать программное средство в среде Visual Studio.

# **1. Практическая значимость программного средства**

Для чего разрабатывается ПС?

Данная программа предназначена для учета продаж мобильных телефонов в магазине. Она позволяет вести контроль за товарами, анализировать объемы продаж за различные периоды и формировать отчеты.

Использование программы упрощает процесс ведения учета, снижает вероятность ошибок, возникающих при ручном вводе данных, и ускоряет анализ продаж, что важно для эффективного управления магазином.

Система автоматически загружает данные из файлов, вычисляет остатки товаров, формирует отчеты о проданных устройствах и стоимости продаж за указанный период.

Кто пользователи данного ПС?

1.Менеджеры магазинов – анализируют продажи, выявляют популярные модели, оценивают спрос.

2.Продавцы – проверяют наличие товара перед продажей.

3.Владельцы бизнеса – оценивают прибыльность продаж, строят стратегию закупок.

Программа предназначена для небольших и средних магазинов, торгующих мобильными устройствами. Она помогает автоматизировать процессы учета, что особенно важно при большом количестве товаров.

Если бы данного ПС не было, то…

•Приходилось бы вести учет продаж вручную, например, в таблицах Excel. Это требует больше времени и повышает риск ошибок.

•Сложнее было бы быстро узнать остатки товара, особенно при большом ассортименте.

•Анализ продаж за период (день, неделю, месяц) потребовал бы дополнительных расчетов.

# **2. Функциональные требования к программному средству**

Программа для учета продаж мобильных телефонов должна обеспечивать выполнение следующих функций:

1. Основные возможности ПС:

Функции необходимо пронумеровать

• Загрузка данных о товарах

Программа загружает список товаров из файла. Каждый товар содержит:

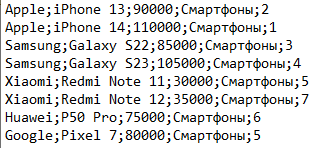
• Марку (бренд) – название производителя (например, Samsung, Apple).

• Модель – конкретная модель телефона.

• Цену за единицу – стоимость одной единицы товара.

• Категорию – тип устройства (например, “смартфон”, “кнопочный телефон”).

Рисунок 1 - Пример файла с товарами (products.txt):



Что за пятое поле с числом в этом формате? Не описано

Как пользователь может воспользоваться данной функцией:

1. Программа автоматически загружает данные о товарах при запуске.

Где должен храниться данный файл? Как называться?

2. Если файл отсутствует или содержит ошибки, программа сообщает об этом и предлагает повторить загрузку.

Описать возможные ошибочные сценарии и конкретные сообщения об ошибках в этих ситуациях

Программа загружает информацию о продажах за разные даты. Каждая запись о продаже содержит:

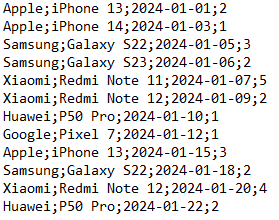
• Марку – производитель устройства.

• Модель – название модели.

• Дату продажи – день, когда товар был продан.

• Количество проданных единиц – сколько единиц продано за раз.

Рисунок 2 - Пример файла с продажами (sales.txt):



Как пользователь может воспользоваться данной функцией:

1.Программа загружает данные автоматически при запуске.

2.Если файл отсутствует, программа выдаст сообщение и предложит указать путь к файлу вручную.

3.Если формат файла неверный (например, текст вместо чисел), программа сообщит об ошибке и выделит красным цветом некорректные строки.

Такие же замечания и по этой функции. Нужно описать четкие требования к месту хранения файла, к его названию, а также описать возможные ошибочные сценарии и реакцию программы на них

Пользователь может выбрать временной интервал (от дд-мм-гггг до дд-мм-гггг), и программа выдаст список проданных товаров с указанием их количества и дат продаж.

Вывод программы в ответ на запрос пользователя:

Введите начальную дату (гггг-мм-дд): 2024-01-01

Введите конечную дату (гггг-мм-дд): 2024-01-31

Отчет о продажах за период с 2024-01-01 по 2024-01-31:

Samsung Galaxy S21 - 2 шт. Дата: 2024-01-10

Apple iPhone 13 - 1 шт. Дата: 2024-01-15

Пользователь устанавливает дату начала и конца с помощью элементов ~~DateTimePicker (dtpStart и dtpEnd). После этого нажимает кнопку “Сформировать отчет”, и данные выводятся в таблицу dgvSales.~~ (У нас консольное приложение)

Программа позволяет сохранить отчет о продажах в текстовый файл для дальнейшего использования.

Структура файла (sales\_report.txt):

Отчет о продажах за период с 2024-01-01 по 2024-01-31:

Samsung Galaxy S21 - 2 шт. Дата: 2024-01-10

Apple iPhone 13 - 1 шт. Дата: 2024-01-15

~~Сохранение осуществляется через стандартное диалоговое окно SaveFileDialog. Пользователь выбирает путь и имя файла, после чего программа сохраняет файл в формате .txt.~~ (У нас консольное приложение)

После успешного сохранения выводится уведомление:

“Отчет сохранен!” (куда?)

Если файл не удалось сохранить, программа сообщает пользователю:

Ошибка: не удалось сохранить отчет. Проверьте доступ к папке.

Пользователь может выбрать другой путь или повторить попытку.

~~2. Взаимодействие пользователя с программой:~~

~~Программа работает в графическом режиме. Основные элементы управления:~~

~~• Меню (MenuStrip): загрузка товаров и продаж, вывод остатков;~~

~~• Кнопки: btnGenerateReport, btnSaveReport, btnShowStock;~~

~~• Таблицы: dgvProducts, dgvSales, dgvStock;~~

~~• Датафильтры: dtpStart, dtpEnd.~~

Пример сценария работы:

1. Пользователь запускает программу;

2. Загружает список товаров;

3. Загружает список продаж;

4. Нажимает “Показать остатки”, чтобы увидеть текущие остатки; (функция не описана!)

5. Устанавливает диапазон дат и нажимает “Сформировать отчет”;

6. По желанию сохраняет отчет в файл.

3. ~~Ошибки и валидация данных:~~ (описать реакцию программы на ошибки в каждом отдельном требовании. Никаких ссылок на классы в требованиях быть не должно. Они должны быть понятны как разработчику, так и заказчику)

~~Для повышения надёжности загружаемых данных в проекте реализован специальный класс SalesFileValidator.~~

~~Этот класс содержит методы LoadProductFileWithValidation() и LoadSalesFileWithValidation(), которые построчно проверяют корректность содержимого файлов.~~

~~Проверяются следующие параметры:~~

~~• Количество столбцов (должно быть строго 5 для товаров, 4 — для продаж);~~

~~• Корректность типов данных (дата, цена, количество);~~

~~• Формат строки (разделитель — ;).~~

~~Каждая строка обрабатывается и превращается в объект ParsedLine, в котором указывается:~~

~~• Сама строка (RawLine);~~

~~• Валидна ли она (IsValid);~~

~~• Сообщение об ошибке, если оно есть (ErrorMessage).~~

# **3. Модульная структура проектируемого программного средства**

Рисунок 2 – Диаграмма классов

Программное средство построено на основе модульной структуры, включающей несколько классов, каждый из которых выполняет строго определенные функции. Это обеспечивает удобство расширения, поддержку кода и его повторное использование.

**Структура классов:**

Программа состоит из следующих основных классов:

• Product — представляет собой модель товара. Содержит свойства:

Brand (марка), Model (модель), Price (цена), Category (категория товара), Quantity (количество на складе).

Используется для хранения данных о товарах, загружаемых из файла.

• Sale — модель одной продажи. Содержит свойства:

Brand, Model, SaleDate (дата продажи), Quantity (количество проданных единиц).

Используется для фильтрации продаж по дате и формирования отчётов.

• Store — основной класс бизнес-логики. Содержит коллекции List<Product> и List<Sale>, а также методы:

• LoadProducts(string filePath) — загрузка товаров из файла;

• LoadSales(string filePath) — загрузка продаж из файла;

• CalculateStock() — расчёт остатков товаров;

• GetSalesReport(DateTime start, DateTime end) — формирование отчёта по продажам за выбранный период.

• ReportGenerator — отвечает за экспорт отчёта. Содержит метод:

GenerateReport(List<Sale> sales, string filePath) — сохраняет отчёт о продажах в текстовый файл.

• SalesFileValidator — осуществляет проверку корректности строк в файлах товаров и продаж. Содержит методы:

• LoadProductFileWithValidation(string filePath)

• LoadSalesFileWithValidation(string filePath)

Оба метода возвращают список объектов ParsedLine, где указано, является ли строка корректной и какое сообщение об ошибке при необходимости.

• ParsedLine — вспомогательный класс для хранения результата валидации строки. Содержит:

RawLine (исходная строка), IsValid (логическое значение корректности), ErrorMessage (сообщение об ошибке).

• ProductStock — используется для отображения остатков на складе. Содержит свойства:

Brand, Model, RemainingQuantity — рассчитанное количество оставшихся единиц товара.

• Form1 — основная форма пользовательского интерфейса (Windows Forms). Содержит:

• элементы управления: btnGenerateReport, btnSaveReport, dtpStart, dtpEnd, dgvProducts, dgvSales, MenuStrip;

• обработчики событий для загрузки файлов, формирования и сохранения отчётов.

• Form2 — дополнительная форма, используемая для отображения остатков товаров. Содержит таблицу dgvStock и меню для сохранения остатков в файл.

**Описание классов и их методов:**

Класс Product

Модель, описывающая товар. Содержит следующие свойства:

• Brand — строка, представляющая марку товара;

• Model — строка, указывающая модель телефона;

• Price — значение типа decimal, представляющее цену товара;

• Category — строка, обозначающая категорию (например, “смартфон”);

• Quantity — целое число, указывающее количество на складе.

Класс Sale

Используется для представления информации о продаже. Включает:

• Brand — марка проданного товара;

• Model — модель телефона;

• SaleDate — дата продажи (DateTime);

• Quantity — количество проданных единиц (int).

Класс Store

Центральный класс, управляющий логикой загрузки, хранения и анализа данных.

• List<Product> Products — список всех товаров;

• List<Sale> Sales — список всех продаж.

Методы:

• LoadProducts(string filePath) — загружает товары из файла;

• LoadSales(string filePath) — загружает продажи;

• CalculateStock() — рассчитывает остатки товаров по каждой модели на основе разницы между количеством и продажами. Возвращает список ProductStock;

• GetSalesReport(DateTime start, DateTime end) — фильтрует список продаж по указанному диапазону дат и возвращает соответствующий список Sale.

Класс ProductStock

Представляет собой результат анализа остатков. Свойства:

• Brand, Model — марка и модель товара;

• RemainingQuantity — рассчитанный остаток (если значение отрицательное, возвращается 0).

Класс ReportGenerator

Отвечает за формирование текстового отчёта.

Метод:

• GenerateReport(List<Sale> sales, string filePath) — записывает список продаж в указанный текстовый файл в читабельном формате.

Класс SalesFileValidator

Осуществляет расширенную валидацию данных при загрузке. Позволяет определить ошибочные строки и их причины.

Методы:

• LoadProductFileWithValidation(string filePath) — построчная загрузка и проверка файла товаров.

• LoadSalesFileWithValidation(string filePath) — построчная загрузка и проверка файла продаж.

Оба метода возвращают список ParsedLine, содержащий информацию о валидности строки.

Класс ParsedLine

Служебная модель для хранения результата проверки строки.

Свойства:

• RawLine — исходная строка из файла;

• IsValid — результат проверки (true / false);

• ErrorMessage — сообщение об ошибке (если есть).

Класс Form1

Основная форма интерфейса пользователя.

Содержит:

• Элементы управления: кнопки (btnGenerateReport, btnSaveReport), поля выбора даты (dtpStart, dtpEnd), таблицы (dgvProducts, dgvSales), меню (MenuStrip);

• Обработчики событий:

• Загрузка списка товаров и продаж через OpenFileDialog;

• Построение отчета и отображение его в таблице;

• Сохранение отчета в файл.

Класс Form2

Вспомогательная форма для отображения остатков на складе.

Содержит:

• Таблицу dgvStock для вывода рассчитанных остатков;

• Меню с пунктом сохранения остатков в файл (SaveFileDialog);

• Метод LoadStockData() — инициализирует данные в таблице.

**Взаимосвязь классов:**

Центральным элементом программы является класс Store, который отвечает за работу с товарами и продажами. Он содержит два списка: List<Product> для хранения данных о доступных товарах и List<Sale> — для учета всех продаж. Через методы LoadProducts() и LoadSales() осуществляется загрузка информации из текстовых файлов, на основе которых формируются соответствующие объекты классов Product и Sale.

Для анализа данных и вывода отчётов Store взаимодействует с другими классами. Метод CalculateStock() позволяет рассчитать остатки товаров на складе, сравнивая количество поступивших товаров и количество проданных единиц. Результат возвращается в виде списка объектов класса ProductStock, который отображается в отдельной форме Form2.

Если пользователь хочет получить отчёт о продажах за определённый период, программа запрашивает начальную и конечную дату. Затем вызывается метод GetSalesReport(), который фильтрует список продаж по заданному диапазону. Полученный список передаётся в метод GenerateReport() класса ReportGenerator, который сохраняет данные в текстовый файл. Таким образом, отчёт формируется отдельно от основной логики приложения, что повышает удобство поддержки кода.

Перед загрузкой данных может использоваться класс SalesFileValidator, который выполняет проверку структуры и содержимого файлов товаров и продаж. Он определяет наличие всех обязательных полей, проверяет правильность форматов и типов данных. Это позволяет заранее исключить некорректные строки и избежать ошибок при создании объектов.

Также в программе используются формы Form1 и Form2. Form1 является основной формой интерфейса, где размещены элементы управления — кнопки, таблицы и выпадающие меню. Через неё пользователь может загружать данные, формировать отчёт и сохранять его. Form2 используется исключительно для отображения остатков на складе.

**Логика работы классов:**

1. Создание объектов

• При запуске программы создаётся объект класса Store, который хранит список товаров (Product) и список продаж (Sale).

2. Загрузка данных

• Пользователь выбирает пункт меню «Загрузить товары».

• Вызывается Store.LoadProducts(filePath), который читает файл и добавляет товары в список.

• Пользователь выбирает пункт «Загрузить продажи».

• Вызывается Store.LoadSales(filePath), который создаёт список продаж на основе файла.

3. Валидация входных данных

• Используется SalesFileValidator, где:

• LoadProductFileWithValidation() проверяет структуру и типы в файле товаров;

• LoadSalesFileWithValidation() делает то же самое для файла продаж;

• Некорректные строки помечаются с пояснением (ParsedLine).

4. Просмотр остатков

• При нажатии «Показать остатки»:

• Вызывается Store.CalculateStock() — считает остатки товаров;

• Данные передаются на форму Form2, где отображаются в таблице dgvStock.

5. Формирование отчёта

• Пользователь выбирает диапазон дат (dtpStart, dtpEnd);

• Вызывается Store.GetSalesReport(start, end) — возвращает продажи за период;

• Результат отображается в таблице dgvSales на форме Form1.

6. Сохранение отчёта

• Нажимается кнопка «Сохранить отчёт»;

• Метод ReportGenerator.GenerateReport(sales, filePath) сохраняет отчёт в текстовый файл;

• Выводится сообщение об успешном сохранении.

7. Работа с формами

• Form1 управляет загрузкой данных, формированием и сохранением отчётов;

• Form2 — отдельная форма, отвечающая за отображение остатков.

# **4. Описание алгоритмов**

**Требуется внести к пояснениям алгоритмов блок-схемы алгоритмов**

В данном разделе описаны ключевые алгоритмы, реализованные в программном средстве. Минимальное требование – два алгоритма, непосредственно решающих поставленные задачи. В программе используются алгоритмы: алгоритм вычисления остатков товаров на складе, алгоритм формирования отчета о продажах за указанный период

Каждый алгоритм будет описан пошагово, с пояснением его работы.

**Алгоритм вычисления остатков товаров на складе:**

Назначение:

Данный алгоритм позволяет определить количество оставшихся товаров на складе, учитывая проданные единицы.

Исходные данные: список товаров, загруженный из файла (марка, модель, цена, категория). Список продаж, содержащий информацию о проданных товарах (марка, модель, дата, количество).

**Невозможно на основе представленной исходной информации произвести расчет остатка на складе!!! Алгоритм нереализуем. Требуется привести описание реализуемого алгоритма**

Описание алгоритма:

Алгоритм вычисления остатков товаров проходит по всему списку товаров, устанавливая начальное количество (как, если этой информации нет???) каждого товара. Затем он проверяет, имеются ли продажи этой модели в списке продаж, суммируя количество проданных единиц. После этого рассчитывается остаток путем вычитания проданного количества из общего запаса. Итоговые данные выводятся на экран, включая название товара, его цену и оставшееся количество.

Псевдокод алгоритма:

Для каждого товара в списке товаров:

Установить sold = 0 (количество проданных единиц)

Для каждой продажи в списке продаж:

Если модель товара совпадает с моделью в продаже:

Увеличить sold на количество проданных единиц

Вычислить остаток: stock = начальное количество - sold

Вывести марку, модель, цену и остаток

**Алгоритм формирования отчета о продажах за указанный период:**

Назначение:

Данный алгоритм фильтрует продажи по заданному диапазону дат и формирует отчет.

Исходные данные: список продаж (марка, модель, дата продажи, количество). Две даты, введенные пользователем (начальная и конечная).

Описание алгоритма:

Алгоритм формирования отчета сначала запрашивает у пользователя начальную и конечную дату, после чего проверяет корректность введенных данных. Затем он перебирает список продаж и отбирает те, что попадают в указанный диапазон. Если такие продажи есть, их список выводится в консоль или передается в модуль сохранения отчета, иначе программа уведомляет пользователя о том, что за данный период продаж не было.

Псевдокод алгоритма:

Запросить начальную и конечную дату у пользователя

Если начальная дата больше конечной:

Вывести сообщение об ошибке

Запросить ввод заново

Создать пустой список reportSales

Для каждой продажи в списке продаж:

Если дата продажи находится в заданном диапазоне:

Добавить продажу в reportSales

Если reportSales не пуст:

Вывести продажи в консоль

Иначе: вывести сообщение "Продаж за указанный период нет"

**Графическое представление алгоритмов:**

Блок-схема алгоритма вычисления остатков товаров.

Конец

Обрабатываем следующий товар

Выводим информацию

Вычисляем остаток

Ищем продажи данного товара и суммируем

Перебираем товары

Загружаем список товаров и продаж

Рисунок 1 - Блок-схема алгоритма вычисления остатков товаров

Блок-схема не соответствует стандарту. В рамках блок-схемы должны быть выделенные структурные элементы программы: начало, конец, блоки операций, блоки условий и циклов. Перебор товаров, очевидно, должен быть отображен на блок-схеме циклом

Блок схема алгоритма формирования отчета о продажах.

Вывести отчет в консоль

Вывести сообщение, что продаж нет

Нет

Да ДаДДДадапвоарворпаоравполрваопрварповалравпДа

Проверить, есть ли продажи в отчете?

Добавить продажу в отчет, если они в диапазоне дат

Вывести сообщение об ошибке

Перебрать продажи за указанный период

Нет

Да

Даты корректны?

Проверить корректность введённых данных

Запросить начальную и конечную дату

Рисунок 2 - Блок схема алгоритма формирования отчета о продажах

Блок-схема не соответствует стандарту. В рамках блок-схемы должны быть выделенные структурные элементы программы: начало, конец, блоки операций, блоки условий и циклов. Условия нарисованы неверно

# **5. Разработка проекта в Visual Studio C#**

**Буду изучать раздел после того, как внесем изменения по комментариям в разделах 2 – 3 – 4. Он должен соотноситься с тем, что описано выше**

Разрабатываемое программное средство было реализовано в среде Visual Studio C#. Оно представляет собой консольное приложение, которое позволяет загружать данные о товарах и продажах, отображать остатки товаров на складе, формировать отчеты о продажах за заданный период и сохранять их в текстовые файлы.

**Описание структуры проекта**

Проект состоит из следующих файлов:

• Program.cs – основной файл, содержащий точку входа и консольное меню для взаимодействия с пользователем.

• Product.cs – класс, описывающий товар (марка, модель, цена, категория).

• Sale.cs – класс, содержащий информацию о продаже (марка, модель, дата продажи, количество).

• Store.cs – основной класс для управления товарами и продажами, содержащий методы загрузки данных, расчета остатков и формирования отчетов.

• ReportGenerator.cs – класс, отвечающий за сохранение отчетов о продажах в файл.

**Реализация ключевых методов**

Основные функции программы реализованы в классе Store. Ниже приведены фрагменты кода, демонстрирующие работу ключевых методов.

Метод загрузки товаров из файла:

public void LoadProducts(string filePath)

{ if (!File.Exists(filePath)) { Console.WriteLine($"Ошибка: Файл {filePath} не найден!"); return; } foreach (var line in File.ReadLines(filePath)) { var parts = line.Split(';'); if (parts.Length != 4 || !decimal.TryParse(parts[2], out decimal price))

{ Console.WriteLine($"Ошибка в строке: {line}"); continue; } Products.Add(new Product { Brand = parts[0], Model = parts[1], Price = price, Category = parts[3] });

} }

Этот метод проверяет существование файла, загружает данные, обрабатывает ошибки ввода и формирует список товаров.

Метод вычисления остатков товаров:

public void ShowStock(){ Console.WriteLine("Остатки товаров:"); foreach (var product in Products) { int sold = Sales.Where(s => s.Model == product.Model).Sum(s => s.Quantity); Console.WriteLine($"{product.Brand} {product.Model} - Цена: {product.Price} руб. Остаток: {sold} шт."); } }

Метод анализирует продажи и рассчитывает, сколько единиц каждого товара осталось в наличии.

Метод формирования отчета о продажах:

public List<Sale> GetSalesReport(DateTime start, DateTime end) { return Sales.Where(s => s.SaleDate >= start && s.SaleDate <= end).ToList(); }

Метод фильтрует список продаж, выбирая только те, которые произошли в заданный период.

Метод сохранения отчета в файл:

public static void GenerateReport(List<Sale> sales, string filePath) { using (StreamWriter sw = new StreamWriter(filePath)) { sw.WriteLine("Отчет о продажах:"); foreach (var sale in sales) { sw.WriteLine($"{sale.Brand} {sale.Model} - {sale.Quantity} шт. Дата: {sale.SaleDate}"); } } Console.WriteLine($"Отчет сохранен в файл {filePath}"); }

Этот метод записывает список продаж в текстовый файл.

**Работа с консольным меню**

Пользователь взаимодействует с программой через консольное меню в Program.cs. Пример интерфейса:

while (true)

{

Console.WriteLine("\nВыберите действие:");

Console.WriteLine("1 - Показать остатки товаров");

Console.WriteLine("2 - Сформировать отчет о продажах");

Console.WriteLine("3 - Сохранить отчет в файл");

Console.WriteLine("4 - Выход");

string choice = Console.ReadLine();

if (choice == "1") { store.ShowStock(); } else if (choice == "2") {

Console.Write("Введите начальную дату (гггг-мм-дд): ");

DateTime start = DateTime.Parse(Console.ReadLine());

Console.Write("Введите конечную дату (гггг-мм-дд): ");

DateTime end = DateTime.Parse(Console.ReadLine());

var report = store.GetSalesReport(start, end);

foreach (var sale in report)

{

Console.WriteLine($"{sale.Brand} {sale.Model} - {sale.Quantity} шт. Дата: {sale.SaleDate}");

}

}

else if (choice == "3")

{

Console.Write("Введите начальную дату (гггг-мм-дд): ");

DateTime start = DateTime.Parse(Console.ReadLine());

Console.Write("Введите конечную дату (гггг-мм-дд): ");

DateTime end = DateTime.Parse(Console.ReadLine());

Console.Write("Введите имя файла: ");

string filePath = Console.ReadLine();

ReportGenerator.GenerateReport(store.GetSalesReport(start, end), filePath);

} else if (choice == "4") { break; } else { Console.WriteLine("Ошибка: неверный ввод!"); } }

Это консольное меню позволяет пользователю выбирать действия: просмотр остатков, формирование отчета или его сохранение в файл.

**Тестирование программы**

Программа была протестирована в нескольких сценариях:

1. Корректная загрузка данных – проверена обработка файлов с товарами и продажами.

2. Вывод остатков – проверено вычисление количества оставшихся товаров.

3. Фильтрация по дате – успешно сформированы отчеты за разные периоды.

4. Обработка ошибок – протестированы случаи отсутствия файлов, некорректного ввода данных и пустых продаж.

Пример работы программы в консоли:

Выберите действие:

1 - Показать остатки товаров

2 - Сформировать отчет о продажах

3 - Сохранить отчет в файл

4 - Выход

> 1

Остатки товаров:

Samsung Galaxy S21 - Цена: 49999 руб. Остаток: 5 шт.

Apple iPhone 13 - Цена: 79999 руб. Остаток: 3 шт.

Выберите действие:

> 2

Введите начальную дату (гггг-мм-дд): 2024-01-01

Введите конечную дату (гггг-мм-дд): 2024-01-31

Samsung Galaxy S21 - 2 шт. Дата: 2024-01-10

Apple iPhone 13 - 1 шт. Дата: 2024-01-15

5. Вывод

Разработанное программное средство позволяет автоматизировать учет продаж мобильных телефонов. Оно загружает данные из файлов, отображает остатки товаров, формирует отчеты по продажам за любой период и сохраняет их в текстовые файлы. Благодаря модульной структуре программа легко расширяется, а использование консольного интерфейса делает ее простой в использовании.