**ВВЕДЕНИЕ**

В современном мире автоматизация играет важную роль в самых разных сферах деятельности, включая фармацевтический бизнес. Использование компьютерных программ позволяет значительно упростить управление основными и вспомогательными процессами в аптеке, повысить эффективность работы и улучшить качество обслуживания клиентов.

Разработка информационной системы для управления аптекой является актуальной задачей в условиях современного фармацевтического рынка. Такая система помогает автоматизировать учет препаратов, клиентов и сотрудников, а также управлять финансовыми операциями, что позволяет сократить время на выполнение рутинных задач и минимизировать ошибки.

Информационные системы в аптечном бизнесе позволяют вести учет препаратов, контролировать их наличие, управлять данными о клиентах и сотрудниках, а также отслеживать финансовые операции, включая продажи, закупки и расчеты. Это особенно важно в условиях высокой конкуренции, когда скорость и точность работы напрямую влияют на удовлетворенность клиентов и прибыль аптеки.

Целью данной курсовой работы является разработка информационной системы для управления аптекой, которая облегчит работу администратора и повысит эффективность бизнес-процессов.

Для достижения этой цели необходимо решить следующие задачи:

* Изучить и описать предметную область фармацевтического бизнеса.
* Определить основные бизнес-процессы, которые можно автоматизировать.

Разработать информационную систему, обеспечивающую:

* учет препаратов, их стоимости и категорий;
* учет клиентов и сотрудников;
* финансовый учет, включая расчет общей стоимости покупок клиентов;
* удобный поиск и фильтрацию данных;
* безопасность и конфиденциальность информации.

Объектом исследования является фармацевтический бизнес.

Предметом исследования выступает процесс разработки информационной системы для управления аптекой.

Для решения поставленных задач использовались следующие методы: анализ (при изучении предметной области), синтез (при реализации системы), формализация (при проектировании системы).

Разрабатываемая информационная система позволит оптимизировать процессы управления аптекой, ускорить обработку информации, упростить ввод и корректировку данных, а также повысить производительность труда администратора.

В первой главе курсовой работы рассматриваются теоретические основы автоматизации бизнес-процессов в фармацевтическом бизнесе, а также ставятся задачи для реализации системы.

Во второй главе описывается практическая реализация информационной системы.

В приложении представлен листинг программы.

**ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ И ОБЩАЯ ПОСТАНОВКА ЗАДАЧ**

**1.1 История и развитие фармацевтического бизнеса**

Фармацевтический бизнес имеет долгую историю, уходящую корнями в древние времена. Еще в античных цивилизациях, таких как Древний Египет, Греция и Рим, существовали прообразы аптек — места, где готовили и продавали лекарственные средства. В те времена фармацевты, или "аптекари", использовали природные компоненты, такие как травы, минералы и животные продукты, для создания лекарств.

В Средние века фармацевтика начала развиваться как отдельная профессия. В Европе аптеки часто создавались при монастырях, где монахи занимались приготовлением лекарств для лечения больных. В это время фармацевты также начали объединяться в гильдии, которые регулировали качество препаратов и стандарты их производства.

В 18–19 веках с развитием науки и медицины фармацевтический бизнес стал более структурированным. Появились первые крупные аптечные сети, а также началось промышленное производство лекарств. Это позволило стандартизировать качество препаратов и сделать их более доступными для населения. В этот период фармацевты стали играть ключевую роль в обеспечении населения лекарствами, а аптеки превратились в важную часть системы здравоохранения.

В 20 веке фармацевтический бизнес претерпел значительные изменения благодаря развитию технологий и химической промышленности. Массовое производство лекарств, внедрение новых методов контроля качества и появление сложных медикаментов изменили роль аптек. Они стали не только местом продажи лекарств, но и центрами консультаций, где фармацевты помогают клиентам выбрать подходящие препараты и объясняют правила их применения.

На Руси первые аптеки появились в 16 веке. В 1581 году в Москве была открыта первая государственная аптека, обслуживавшая царский двор. В 17–18 веках аптеки начали распространяться по городам, а их деятельность регулировалась государством. К началу 20 века в России насчитывалось несколько тысяч аптек, большинство из которых принадлежали частным владельцам.

После Октябрьской революции аптечный бизнес в России был национализирован, и система управления аптеками подверглась серьезным изменениям. В советский период аптеки стали частью государственной системы здравоохранения, а их основная задача заключалась в обеспечении населения лекарствами по доступным ценам. В послевоенные годы началось восстановление и расширение аптечной сети, а к 1980-м годам в СССР насчитывалось более 20 тысяч аптек.

В современной России фармацевтический бизнес активно развивается. По данным Росстата, на 2023 год в стране функционировало более 65 тысяч аптек, включая крупные аптечные сети и небольшие частные аптеки. Рост конкуренции и внедрение новых технологий, таких как автоматизация учета и онлайн-продажи, способствуют дальнейшему развитию отрасли. Сегодня аптеки не только продают лекарства, но и предоставляют дополнительные услуги, такие как консультации фармацевтов и заказ редких препаратов.

История фармацевтического бизнеса показывает, что эффективное управление является ключевым фактором успеха. Современные технологии, включая информационные системы, позволяют аптекам оптимизировать свою работу, улучшать обслуживание клиентов и повышать прибыль.

**1.2 Организационная структура аптеки**

Организационная структура аптеки зависит от ее масштабов, местоположения, ассортимента товаров и услуг, а также целевой аудитории. Она определяет распределение обязанностей между сотрудниками и помогает эффективно управлять бизнес-процессами. Для небольшой аптеки структура может быть минимальной, а в крупных аптечных сетях она обычно более сложная и многоуровневая.

Основная цель организационной структуры аптеки — обеспечить качественное обслуживание клиентов, минимизировать затраты и повысить управляемость. Неправильное распределение обязанностей или недостаток персонала может привести к снижению качества обслуживания, что негативно скажется на репутации и доходах аптеки.

В любой организационной структуре аптеки можно выделить три ключевых элемента:

* Управление: Руководитель или группа руководителей, которые принимают решения и координируют работу аптеки.
* Правила работы: Явные или неявные правила, которые регулируют деятельность сотрудников.
* Распределение труда: Четкое определение обязанностей каждого сотрудника, будь то формальное или неформальное распределение.

Существуют три основные структуры управления, которые применяются в аптеках:

* Линейная структура: Используется в небольших аптеках. Все сотрудники подчиняются одному руководителю (например, заведующему аптекой), который принимает все решения. Такая структура проста, но может быть неэффективной при большом объеме задач.
* Функциональная структура: Применяется в средних и крупных аптеках. Сотрудники делятся на отделы (например, продажи, закупки, учет), каждый из которых имеет своего руководителя. Это повышает эффективность в отдельных направлениях, но может усложнить взаимодействие между отделами.
* Линейно-функциональная структура: Комбинация двух подходов, характерная для крупных аптечных сетей. Например, заведующий аптекой управляет общими процессами, а руководители отделов (закупок, продаж, склада) отвечают за свои направления.

Основные подразделения аптеки:

* **Руководство**: Заведующий аптекой, который отвечает за общее управление, контроль финансов и координацию сотрудников.
* **Фармацевты и провизоры**: Сотрудники, которые непосредственно работают с клиентами, консультируют их и продают препараты.
* **Складской персонал**: Отвечает за прием, хранение и учет препаратов.
* **Бухгалтерия**: Управляет финансовыми операциями, включая расчеты с поставщиками и учет продаж.

В аптеках также важно разделение сотрудников по степени взаимодействия с клиентами:

* **Фронт-офис**: Фармацевты, которые работают с клиентами напрямую.
* **Бэк-офис**: Сотрудники, которые не контактируют с клиентами (например, бухгалтеры или складские работники).

Организационная структура аптеки должна быть гибкой, чтобы при необходимости можно было расширить или сократить штат сотрудников

**1.3 Функции, выполняемые администратором аптеки**

Администратор аптеки (или заведующий) играет ключевую роль в управлении аптечным бизнесом. От его профессионализма зависит качество обслуживания клиентов, эффективность работы персонала и финансовая успешность аптеки.

Исторически роль администратора аптеки начала формироваться в 18–19 веках, когда аптеки стали более организованными предприятиями. В 20 веке, с развитием технологий и увеличением ассортимента препаратов, обязанности администратора значительно расширились. Сегодня администратор аптеки — это не только руководитель, но и менеджер, который использует современные технологии для управления бизнес-процессами.

Основные функции администратора аптеки:

* **Учет препаратов**: Контроль наличия препаратов, их категорий, стоимости и сроков годности.
* **Управление продажами**: Обработка заказов клиентов, консультирование и расчет стоимости покупок.
* **Учет клиентов**: Ведение базы данных клиентов, включая информацию о покупках и контактные данные (при необходимости).
* **Управление персоналом**: Координация работы сотрудников (фармацевтов, складских работников), распределение задач и контроль их выполнения.
* **Финансовые операции**: Учет доходов и расходов, расчет общей стоимости покупок клиентов, работа с поставщиками.

**Обеспечение порядка и безопасности**: Контроль за соблюдением санитарных норм, безопасностью хранения препаратов и конфиденциальностью данных клиентов.

Виды деятельности администратора аптеки:

* **Дневной администратор**: Управляет аптекой в дневную смену, отвечает за продажи, учет препаратов и работу с клиентами.
* **Ночной администратор**: Работает в ночную смену, обеспечивая бесперебойную работу аптеки и безопасность.
* **Администратор склада**: Отвечает за прием, хранение и учет препаратов, а также взаимодействие с поставщиками.
* **Главный администратор**: Координирует работу всех сотрудников, управляет финансами и разрабатывает стратегию развития аптеки.

Эффективное выполнение этих функций требует использования информационных систем, которые позволяют автоматизировать рутинные задачи и повысить точность учета.

**1.4 Постановка задачи**

Цель курсовой работы — разработка информационной системы для управления аптекой с следующим функционалом:

* Учет препаратов (номер, категория, стоимость, отдел хранения). Учет клиентов (ФИО, город, покупки, дата покупки).
* Учет сотрудников (ФИО, рабочие дни, отделы уборки).
* Финансовый учет (расчет общей стоимости покупок клиентов).

Фильтрация и удаление данных (поиск препаратов по категории, клиентов по городу, сотрудников по дню и отделу).

Задачи работы :

1. Изучить потребности рынка и определить функциональные требования к системе.

2. Выбрать подходящие технологии и язык программирования для разработки.

3. Написать программный код для реализации основных функций системы. Провести тестирование системы для устранения ошибок и обеспечения надежности.

4. Оценить эффективность системы на основе тестов и предложить рекомендации по ее улучшению.

В результате должна быть создана функциональная информационная система, которая оптимизирует работу администратора аптеки и повысит качество обслуживания клиентов.

**ГЛАВА 2. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ**

* **Требования к программе**

Для работы программы требуется минимум 120 МБ свободного места на жестком диске. Операционная система должна быть не ниже Microsoft Windows 7 (64битная версия), включая Windows 8 и Windows 10. Программа должна стабильно работать при длительном использовании. В случае сбоев оборудования необходимо обеспечить возможность восстановления данных.

Разработка программного продукта осуществлялась с использованием следующей конфигурации аппаратного и программного обеспечения:

* Процессор: AMD Ryzen 5 5500 с тактовой частотой до 3.6 ГГц.
* Оперативная память: 16 ГБ DDR4.
* Видеокарта: Nvidia Palit RTX 3060 12GB
* Операционная система: Windows 10 Pro 64-битная версия.
* Накопитель: SSD на 1ТБ.
* Дисплей: 27-дюймовый Quad HD(2560 x 1440).

Эта конфигурация обеспечила разработчикам необходимую производительность для эффективной разработки, тестирования и дебаггинга программного обеспечения.

Минимально необходимая конфигурация для использования программы:

* Операционная система: Windows 7, 8, 10 (64-битные версии);
* Процессор: Intel Core 2 Duo или аналогичный с частотой от 1.8 ГГц;
* Оперативная память: 4 ГБ;
* Свободное место на жестком диске: не менее 1 ГБ;
* Минимальное разрешение экрана: 1024 x 768 пикселей.

Такие требования к аппаратной и программной части обеспечивают комфортное и бесперебойное использование программы, а также позволяют максимально использовать её функциональные возможности.

**2.2 Руководство пользователя**

Данное руководство предназначено для пользователей информационной системы управления аптекой. Оно описывает функционал приложения и помогает освоить его использование.

Запуск приложения

Для запуска приложения дважды щелкните левой кнопкой мыши по исполняемому файлу (.exe). Данное действие показано на рисунке 1.



Рис. 1 Запуск приложения

Первое меню

После запуска открывается первое меню, где доступны следующие функции:

* Считывание данных
* Добавление клиента
* Добавление служащего
* Добавление препарата
* Перейти в главное меню



Рис. 2 Первое меню программы

Считывание данных

Пункт 1 позволяет загрузить данные из файлов "Покупатели.txt", "Сотрудники.txt" и "Препараты.txt". При успешном считывании отображается сообщение: "Считывание удалось". В случае ошибки — "Считывание не удалось". как показано на рисунке 3.



Рис. 3 Успешное считывание данных

Добавление клиента

Пункт 2 позволяет добавить нового клиента. Пользователь вводит id, ФИО, номер паспорта, город, номер препарата, тип номера, количество дней и дату покупки, данное действие показано на рисунке 4.



Рис. 4 Добавление клиента

Добавление служащего

Пункт 3 позволяет добавить нового сотрудника. Вводятся ID, ФИО, Участок и День недели. Данное действие показано на рисунке 5.



Рис. 5 Добавление служащего

Добавление препарата

Пункт 4 позволяет добавить новый препарат. Вводятся номер препарата, отдел хранения, категория и цена, что показано на рисунке 6.



Рис. 6 Добавление препарата

Переход в главное меню

Пункт 5 переводит пользователя в главное меню  


Главное меню

Главное меню включает следующие разделы:

* Покупатели
* Сотрудники
* Препараты
* Общая сумма покупок клиентов
* Обновить данные клиента
* Выход из программы
* Вернуться в первое меню

Раздел Препараты

Перейдя из главного меню на этот раздел, пользователь может увидеть несколько пунктов, которые привязаны к клавиатуре (рис. 7.1).



Рис. 7.1 Раздел номер

Вывод информации о номерах

В этом разделе доступны следующие функции:

[1] Вывод всех препаратов: отображает список всех препаратов. [2] Поиск стоимости по номеру: позволяет узнать стоимость препарата по его номеру.

* Фильтрация препаратов по категории: выводит препараты указанной категории.

Выход в главное меню(рис. 7.2).



Рис. 7.2 Вывод информации о номерах

Раздел Покупатели

* Вывод всех покупателей: показывает информацию о клиентах.
* Поиск покупателей по городу: выводит клиентов из указанного города. [3] Вывод покупателей по категории: отображает клиентов, купивших определенный тип препаратов. [4] Выход в главное меню



Раздел Сотрудники

[1] Вывод всех сотрудников: показывает информацию о сотрудниках. [2] Поиск сотрудников по дню и участку: находит сотрудника, работающего в указанном отделе в определенный день.

[3] Удаление сотрудника: позволяет уволить сотрудника. [4] Вывод сотрудников по дню недели: показывает сотрудников, работающих в указанный день.

[5] Выход в главное меню.



Общая сумма покупок клиентов

Пункт 4 главного меню отображает общую сумму, которую клиенты заплатили за препараты.



Обновление данных клиента

Пункт 5 позволяет изменить данные о клиенте (например, ФИО или город).



Выход из программы

Пункт 6 завершает работу приложения

**Руководство программиста**

Программа представляет собой консольное приложение, разработанное на языке C#. Данные хранятся в текстовых файлах: "Препараты.txt", "Покупатели.txt" и "Сотрудники.txt".

Структура кода включает в себя:

Класс **Medicine:**

Свойства**:**

* Number (int): номер препарата.
* Floor (int): отдел хранения.
* Type (string): категория препарата.
* Cost\_per\_day (int): цена препарата.

Также в классе **Medicine** представлены следующие методы:

* AddMedicine(): добавление нового препарата.
* Save(): сохранение данных в файл.
* FromFile(): чтение данных из файла.
* Print(): вывод информации о препарате.
* LoadMedicines(): загрузка данных из файла.
* ShowAll(): вывод всех препаратов.
* ShowCostByNumber(): поиск стоимости по номеру.
* FilterByType(): фильтрация по категории.

Класс **Customer**:

Свойства:

* Key (int): ключ клиента.
* FIO (string): ФИО клиента.
* Number\_pasport (int): номер паспорта.
* Where (string): город клиента.
* Number (int): номер препарата.
* Place (string): тип номера (не используется в аптеке).
* Kol\_vo\_days (int): количество дней (не используется).
* Date (DateTime): дата покупки

Также в классе **Customer** представлены следующие методы:

* AddCustomer(): добавление клиента.
* Save(): сохранение в файл.
* FromFile(): чтение из файла.
* Print(): вывод информации.
* City(): поиск клиентов по городу.
* Single(): фильтрация по категории (адаптировано для аптеки).
* CalculateTotalCost(): расчет общей стоимости покупок.
* UpdateCustomer(): обновление данных клиента

Класс **Employee:**

Свойства:

* Key (int): ключ сотрудника. FIO (string): ФИО сотрудника. floors (List): отделы уборки. days (List): дни работы.

Также в классе **Employee** представлены следующие методы:

* AddEmployee(): добавление сотрудника.
* Save(): сохранение в файл.
* FromFile(): чтение из файла.
* Print(): вывод информации.
* Cleaning(): поиск сотрудника по дню и отделу.
* Delete(): удаление сотрудника.
* ShowByDay(): вывод сотрудников по дню

Класс **General:**

Поля:

* Customers (List): список клиентов.
* Employees (List): список сотрудников.
* Medicines (List): список препаратов.

Класс **Program**:

Содержит метод Main, который управляет меню и взаимодействием с пользователем

Во время разработки были использованы следующие библиотеки объектно-ориентированного языка программирования С#: System, System.IO, System.Collections.Generic, System.Linq.

**ЗАКЛЮЧЕНИЕ**

В рамках курсовой работы была разработана информационная система для управления аптекой на языке C#. Данные хранятся в трех текстовых файлах: "Препараты.txt", "Покупатели.txt" и "Сотрудники.txt".

Пользовательский интерфейс реализован в виде консольного приложения, что обеспечивает простоту использования и понятную навигацию. Система позволяет добавлять, просматривать, фильтровать и удалять данные о препаратах, клиентах и сотрудниках, а также рассчитывать общую стоимость покупок клиентов.

Основные преимущества системы:

* Простой и интуитивно понятный интерфейс.
* Быстрый доступ к данным через текстовые файлы.
* Возможность автоматизации рутинных задач

Области для улучшения:

* Добавление функции бронирования препаратов.
* Улучшение интерфейса, например, переход на графический интерфейс.
* Интеграция с базами данных для хранения больших объемов информации.

Цель работы достигнута — разработана система, которая упрощает управление аптекой и повышает эффективность работы администратора.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

* Албахари Дж., Албахари Б.: C# 8.0. Карманный справочник - Москва: ООО «И. Д. Вильямс», 2020. – 240 с.
* Шилдт Г. C# 4.0: полное руководство. — Москва: ООО «И. Д. Вильямс», 2015. — 1056 с.
* Антон Марченко. Основы программирования на С# 2.0. Учебное пособие. — 2017. — 557 с.
* <<История фармацевтического дела. — URL[: HYPERLINK "https://pharmacyhistory.ru/"HYPERLINK "https://pharmacyhistory.ru/" HYPERLINK "https://pharmacyhistory.ru/"https://pharmacyhistory.ru](https://pharmacyhistory.ru/)>>(<https://ru.scribd.com/document/603037793/%D0%9B%D0%B5%D0%BA%D1%86%D0%B8%D1%8F-%D0%A1%D0%BB%D1%83%D0%B6%D0%B1%D1%8B-%D0%B3%D0%BE%D1%81%D1%82%D0%B8%D0%BD%D0%B8%D1%86%D1%8B>)
* «Организация работы аптеки. — URL: [https://pharmmanagement.ru](https://pharmmanagement.ru/)»

(<https://hoteladvisor.livejournal.com/32450.html>)

**ПРИЛОЖЕНИЕ А**

class Employee { ///

/// Свойства /// public int Key { set; get; } public string FIO { set; get; } public List floors = new List(7); public List days = new List(7);

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="key"></param>

/// <param name="fio"></param>

/// <param name="floors"></param>

/// <param name="days"></param>

public Employee(int key, string fio, List<int> floors, List<string> days)

{

Key = key;

FIO = fio;

this.floors = floors;

this.days = days;

}

/// <summary>

/// Метод добавления

/// </summary>

/// <returns></returns>

public static Employee AddEmployee()

{

Console.Write("Введите ключ:");

int key = Convert.ToInt32(Console.ReadLine());

Console.Write("Введите ФИО:");

string fio = Console.ReadLine();

List<int> floors = new List<int>(7);

for (int i = 0; i < floors.Capacity; i++)

{

Console.Write("Введите этаж:");

floors.Add(Convert.ToInt32(Console.ReadLine()));

if (floors[i] > 3)

{

Console.Write("Попробуйте еще раз,ошибка:");

loors[i] = Convert.ToInt32(Console.ReadLine());

}

}

List<string> days = new List<string>(7);

for (int i = 0; i < days.Capacity; i++)

{

Console.Write("Введите день недели:");

days.Add(Console.ReadLine());

}

return new Employee(key, fio, floors, days);

}

/// <summary>

/// Метод записи в файл

/// </summary>

public void SaveEmployees()

{

using (StreamWriter writer = new StreamWriter(@"C:\Users\andre\OneDrive\Рабочий стол\Курсач\Служащие.txt", true))

{

writer.WriteLine($"{Key};{FIO};{floors[0]};{floors[1]};{floors[2]};" +

$"{floors[3]};{floors[4]};{floors[5]};" +

$"{floors[6]};{days[0]};{days[1]};" +

$"{days[2]};{days[3]};{days[4]};{days[5]};{days[6]};");

}

}

/// <summary>

/// Метод чтения

/// </summary>

/// <param name="line"></param>

/// <returns></returns>

public static Employee ReaderEmployee(string line)

{

string[] parts = line.Split(';');

List<int> floors\_txt = new List<int>();

for (int i = 2; i <= 8; i++)

{

floors\_txt.Add(int.Parse(parts[i]));

}

List<string> days\_txt = new List<string>();

for (int i = 9; i <= 15; i++)

{

days\_txt.Add(parts[i]);

}

return new Employee(int.Parse(parts[0]), parts[1], floors\_txt, days\_txt);

}

/// <summary>

/// Метод вывода информации

/// </summary>

public void PrintEmployee()

{

Console.WriteLine($"Ключ:{Key},ФИО:{FIO}");

foreach (var elem in floors)

{

Console.WriteLine($"Этаж:{elem};");

}

foreach (var elem in days)

{

Console.WriteLine($"День недели:{elem}");

}

}

/// <summary>

/// Метод поиска сотрудника по дню и этажу

/// </summary>

/// <param name="day"></param>

/// <param name="floor"></param>

public void Cleaning(string day, int floor)

{

bool cleaner = false;

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

{

if (days[i] == day && floors[i] == floor)

{

PrintEmployee();

cleaner = true;

}

}

if (cleaner == false)

{

Console.WriteLine("Работник не найден");

}

}

/// <summary>

/// Метод удаления

/// </summary>

/// <param name="employees"></param>

public static void Delete(List<Employee> employees)

{

Console.Write("Введите номер служащего,которого необходимо удалить:");

int delete = Convert.ToInt32(Console.ReadLine());

employees = employees.Where(x => x.Key != delete).ToList();

string[] lines = File.ReadAllLines(@"C:\Users\andre\OneDrive\Рабочий стол\Курсач\Служащие.txt");

if (delete - 1 >= 0 && delete - 1 < lines.Length)

{

lines = lines.Where((line, index) => index != delete - 1).ToArray();

}

File.WriteAllLines(@"C:\Users\andre\OneDrive\Рабочий стол\Курсач\Служащие.txt", lines);

}

}

class Client { ///

/// Свойства /// public int Key { set; get; } public string FIO { set; get; } public int Number\_pasport { set; get; } public string Where { set; get; } public int Number { set; get; } public string Place { set; get; } public int Kol\_vo\_days { set; get; } public DateTime Date { set; get; }

/// <summary>

/// Конструктор

/// </summary>

/// <param name="key"></param>

/// <param name="fio"></param>

/// <param name="number\_pasport"></param>

/// <param name="where"></param>

/// <param name="number"></param>

/// <param name="place"></param>

/// <param name="kol\_vo\_days"></param>

public Client(int key, string fio, int number\_pasport, string where, int number, string place, int kol\_vo\_days, DateTime date)

{

Key = key;

FIO = fio;

Number\_pasport = number\_pasport;

Where = where;

Number = number;

Place = place;

Kol\_vo\_days = kol\_vo\_days;

Date = date;

}

/// <summary>

/// Добавление объекта

/// </summary>

/// <returns></returns>

public static Client AddClient()

{

Console.Write("Введите ключ:");

int key = Convert.ToInt32(Console.ReadLine());

Console.Write("Введите ФИО:");

string fio = Console.ReadLine();

Console.Write("Введите номер паспорта:");

int number\_pasport = Convert.ToInt32(Console.ReadLine());

Console.Write("Откуда прибыл:");

string where = Console.ReadLine();

Console.Write("Номер в который заселился:");

int number = Convert.ToInt32(Console.ReadLine());

Console.Write("Место номера:");

string place = Console.ReadLine();

Console.Write("Количество дней проживания:");

int kol\_vo\_days = Convert.ToInt32(Console.ReadLine());

Console.Write("Введите дату приезда:");

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

return new Client(key, fio, number\_pasport, where, number, place, kol\_vo\_days, date);

}

/// <summary>

/// Запись в файл Клиенты.txt

/// </summary>

public void SaveClients()

{

using (StreamWriter writer = new StreamWriter(@"C:\Users\andre\OneDrive\Рабочий стол\Курсач\Клиенты.txt", true))

{

writer.WriteLine($"{Key};{FIO};{Number\_pasport};{Where};{Number};{Place};{Kol\_vo\_days};{Date.ToString("yyyy-MM-dd")};");

}

}

/// <summary>

/// Считывание с файла Клиенты.txt и запись результатов считывания в коллекцию

/// </summary>

/// <param name="line"></param>

/// <returns></returns>

public static Client ReaderClient(string line)

{

string[] parts = line.Split(';');

return new Client(int.Parse(parts[0]), parts[1], int.Parse(parts[2]), parts[3], int.Parse(parts[4]), parts[5], int.Parse(parts[6]), DateTime.Parse(parts[7]));

}

/// <summary>

/// Вывод информации в консоль

/// </summary>

public void PrintClient()

{

Console.WriteLine($"Ключ:{Key},ФИО:{FIO},Номер паспорта:{Number\_pasport},Откуда:{Where}");

Console.WriteLine($"Номер:{Number},Место:{Place},Кол-во дней:{Kol\_vo\_days},Дата приезда:{Date.ToString("yyyy-MM-dd")}");

}

/// <summary>

/// Поиск клиентов по городу

/// </summary>

/// <param name="clients"></param>

public static void City(List<Client> clients)

{

Console.WriteLine("Введите название города:");

string cities = Console.ReadLine();

bool citi = false;

foreach (var elem in clients)

{

if (elem.Where == cities)

{

elem.PrintClient();

citi = true;

}

}

if (citi == false)

{

Console.WriteLine("Клиенты из указанного города не найдены");

}

}

/// <summary>

/// Фильтр по типу места

/// </summary>

public void Single()

{

if (Place == "Одноместный")

{

PrintClient();

}

}

}

class Hotel { ///

/// Свойства класса /// public int Number { set; get; } public int Floor { set; get; } public string Type { set; get; } public int Cost\_per\_day { set; get; } public List list1 = new List(); public List list2 = new List();

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="number"></param>

/// <param name="floor"></param>

/// <param name="type"></param>

/// <param name="cost\_per\_day"></param>

/// <param name="clients"></param>

/// <param name="employees"></param>

public Hotel(int number, int floor, string type, int cost\_per\_day, List<Client> clients, List<Employee> employees)

{

Number = number;

Floor = floor;

Type = type;

Cost\_per\_day = cost\_per\_day;

this.list1 = clients;

this.list2 = employees;

}

/// <summary>

/// Метод добавления номера

/// </summary>

/// <param name="clients\_to\_number"></param>

/// <param name="employees"></param>

/// <returns></returns>

public static Hotel AddNumber(List<Client> clients\_to\_number, List<Employee> employees)

{

Console.Write("Номер:");

int number = Convert.ToInt32(Console.ReadLine());

Console.Write("Этаж:");

int floor = Convert.ToInt32(Console.ReadLine());

Console.Write("Тип номера:");

string type = Console.ReadLine();

Console.Write("Стоимость одной ночи:");

int cost\_per\_day = Convert.ToInt32(Console.ReadLine());

List<Client> list1 = new List<Client>();

for (int i = 0; i < clients\_to\_number.Count; i++)

{

if (clients\_to\_number[i].Number == number && clients\_to\_number[i].Place == type)

{

list1.Add(clients\_to\_number[i]);

}

}

List<Employee> list2 = new List<Employee>();

list2 = employees;

return new Hotel(number, floor, type, cost\_per\_day, list1, list2);

}

/// <summary>

/// Метод записи информации о номерах в файл Номера.txt

/// </summary>

public void SaveHotel()

{

using (StreamWriter writer = new StreamWriter(@"C:\Users\andre\OneDrive\Рабочий стол\Курсач\Номера.txt", true))

{

writer.WriteLine($"{Number};{Floor};{Type};{Cost\_per\_day}");

}

}

/// <summary>

/// Чтение и запись в объекты строки файла

/// </summary>

/// <param name="line"></param>

/// <param name="clients"></param>

/// <param name="employees"></param>

/// <returns></returns>

public static Hotel ReaderNumbers(string line, List<Client> clients, List<Employee> employees)

{

string[] parts = line.Split(';');

List<Client> clients\_txt = new List<Client>();

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

{

if (int.Parse(parts[0]) == clients[i].Number && parts[2] == clients[i].Place)

{

clients\_txt.Add(clients[i]);

}

}

List<Employee> employees\_txt = employees;

return new Hotel(int.Parse(parts[0]), int.Parse(parts[1]), parts[2], int.Parse(parts[3]), clients\_txt, employees\_txt);

}

/// <summary>

/// Вывод информации

/// </summary>

public void Print()

{

Console.WriteLine($"Номер:{Number},Этаж:{Floor},Тип:{Type},Оплата:{Cost\_per\_day}");

}

/// <summary>

/// Расчет стоимости

/// </summary>

/// <returns></returns>

public int Cost()

{

if (Type == "Одноместный")

{

return Cost\_per\_day;

}

else if (Type == "Двухместный")

{

return Cost\_per\_day / 2;

}

else

{

return Cost\_per\_day / 3;

}

}

/// <summary>

/// Проверка свободных номеров

/// </summary>

public void Empty\_number()

{

if (list1.Count == 0)

{

Print();

}

}

}

class Program { static void Main(string[] args) { List list1 = new List(); List list2 = new List(); List list3 = new List(); bool flag = true;

while (flag)

{

Console.Clear();

Console.ForegroundColor = ConsoleColor.DarkBlue;

Console.WriteLine("ПЕРВОЕ МЕНЮ");

Console.ResetColor();

Console.ForegroundColor = ConsoleColor.DarkYellow;

Console.WriteLine("[1] Считывание данных");

Console.WriteLine("[2] Добавление клиента");

Console.WriteLine("[3] Добавление служащего");

Console.ResetColor();

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[4] Выход из первого меню");

Console.ResetColor();

switch (Console.ReadKey(true).KeyChar)

{

case '1':

try

{

Console.Clear();

using (StreamReader reader = new StreamReader(@"C:\Users\andre\OneDrive\Рабочий стол\Курсач\Служащие.txt"))

{

string line;

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

{

list3.Add(Employee.ReaderEmployee(line));

}

}

using (StreamReader reader = new StreamReader(@"C:\Users\andre\OneDrive\Рабочий стол\Курсач\Клиенты.txt"))

{

string line;

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

{

list2.Add(Client.ReaderClient(line));

}

}

using (StreamReader reader = new StreamReader(@"C:\Users\andre\OneDrive\Рабочий стол\Курсач\Номера.txt"))

{

string line;

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

{

list1.Add(Hotel.ReaderNumbers(line, list2, list3));

}

}

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Считывание удалось");

Console.ResetColor();

Console.Write("Хотите продолжить?");

string answer = Console.ReadLine();

if (answer == "Нет" || answer == "нет")

{

flag = false;

}

}

catch

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Считывание не удалось");

}

break;

case '2':

try

{

Console.Clear();

list2.Add(Client.AddClient());

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Новый клиент успешно добавлен");

Console.ResetColor();

Console.Write("Хотите продолжить выбор из первого меню? ");

string answer = Console.ReadLine();

foreach (var elem in list2)

{

elem.SaveClients();

}

if (answer == "Нет" || answer == "нет")

{

flag = false;

}

}

catch

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Произошла ошибка, попробуйте еще раз");

Console.ResetColor();

}

break;

case '3':

try

{

Console.Clear();

list3.Add(Employee.AddEmployee());

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Новый служащий успешно добавлен");

Console.ResetColor();

Console.Write("Хотите продолжить выбор из первого меню? ");

string answer = Console.ReadLine();

foreach (var elem in list3)

{

elem.SaveEmployees();

}

if (answer == "Нет" || answer == "нет")

{

flag = false;

}

}

catch

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Произошла ошибка, попробуйте еще раз");

Console.ResetColor();

}

break;

case '4':

flag = false;

break;

default:

Console.WriteLine("Ошибка,попробуйте еще раз");

break;

}

bool flag\_main = true;

while (flag\_main)

{

Console.Clear();

Console.ForegroundColor = ConsoleColor.DarkBlue;

Console.WriteLine("ГЛАВНОЕ МЕНЮ");

Console.ResetColor();

Console.ForegroundColor = ConsoleColor.DarkYellow;

Console.WriteLine("[1] Номера");

Console.WriteLine("[2] Клиенты");

Console.WriteLine("[3] Служащие");

Console.WriteLine("[4] Выход из программы");

Console.ResetColor();

switch (Console.ReadKey(true).KeyChar)

{

case '1':

bool flag\_number = true;

while (flag\_number)

{

Console.Clear();

Console.ForegroundColor = ConsoleColor.DarkYellow;

Console.WriteLine("[1] Вывод информации о номерах");

Console.WriteLine("[2] Стоимость места для определенного номера");

Console.WriteLine("[3] Кол-во свободных номеров");

Console.WriteLine("[4] Общая сумма за заселенные номера");

Console.WriteLine("[5] Выход в главное меню");

Console.ResetColor();

switch (Console.ReadKey(true).KeyChar)

{

case '1':

try

{

Console.Clear();

foreach (var elem in list1)

{

elem.Print();

}

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Информация успешно выведена");

Console.ResetColor();

Console.Write("Хотите продолжить? ");

string answer = Console.ReadLine();

if (answer == "Нет" || answer == "нет")

{

flag\_number = false;

}

}

catch

{

Console.WriteLine("Ошибка");

}

break;

case '2':

try

{

Console.Clear();

Console.WriteLine("Введите необходимый номер:");

int n = Convert.ToInt32(Console.ReadLine());

foreach (var elem in list1)

{

if (elem.Number == n)

{

Console.WriteLine($"Стоимость {elem.Number} составляет {elem.Cost()} руб. за место");

}

}

Console.Write("Хотите продолжить? ");

string answer = Console.ReadLine();

if (answer == "Нет" || answer == "нет")

{

flag\_number = false;

}

}

catch

{

Console.WriteLine("Ошибка");

}

break;

case '3':

try

{

Console.Clear();

foreach (var elem in list1)

{

elem.Empty\_number();

}

Console.Write("Хотите продолжить? ");

string answer = Console.ReadLine();

if (answer == "Нет" || answer == "нет")

{

flag\_number = false;

}

}

catch

{

Console.WriteLine("Ошибка");

}

break;

case '4':

try

{

Console.Clear();

int sum = 0;

int sum\_days = 0;

foreach (var elem in list2)

{

sum\_days += elem.Kol\_vo\_days;

}

foreach (var elem in list1)

{

if (elem.list1.Count != 0)

{

sum += elem.Cost();

}

}

Console.WriteLine($"Общая сумма выплаченная клиентами равняется {sum \* sum\_days} руб.");

Console.Write("Хотите продолжить? ");

string answer = Console.ReadLine();

if (answer == "Нет" || answer == "нет")

{

flag\_number = false;

}

}

catch

{

Console.WriteLine("Ошибка");

}

break;

case '5':

flag\_number = false;

break;

default:

Console.Clear();

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Ошибка попробуйте еще раз");

Console.ResetColor();

break;

}

}

break;

case '2':

bool flag\_client = true;

while (flag\_client)

{

Console.Clear();

Console.ForegroundColor = ConsoleColor.DarkYellow;

Console.WriteLine("[1] Вывод информации о клиентах");

Console.WriteLine("[2] Список клиентов,прибывших из указанного города");

Console.WriteLine("[3] Клиенты проживающие в одноместных номерах");

Console.WriteLine("[4] Выход в главное меню");

Console.ResetColor();

switch (Console.ReadKey(true).KeyChar)

{

case '1':

try

{

Console.Clear();

foreach (var elem in list2)

{

elem.PrintClient();

}

Console.Write("Хотите продолжить? ");

string answer = Console.ReadLine();

if (answer == "Нет" || answer == "нет")

{

flag\_client = false;

}

}

catch

{

Console.WriteLine("Ошибка,попробуйте еще раз");

}

break;

case '2':

try

{

Console.Clear();

Client.City(list2);

Console.Write("Хотите продолжить? ");

string answer = Console.ReadLine();

if (answer == "Нет" || answer == "нет")

{

flag\_client = false;

}

}

catch

{

Console.WriteLine("Ошибка,попробуйте еще раз");

}

break;

case '3':

try

{

Console.Clear();

foreach (var elem in list2)

{

elem.Single();

}

Console.Write("Хотите продолжить? ");

string answer = Console.ReadLine();

if (answer == "Нет" || answer == "нет")

{

flag\_client = false;

}

}

catch

{

Console.WriteLine("Ошибка,пробуите еще раз");

}

break;

case '4':

flag\_client = false;

break;

default:

Console.Clear();

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Ошибка попробуйте еще раз");

Console.ResetColor();

break;

}

}

foreach (var elem in list3)

{

elem.PrintEmployee();

}

break;

case '3':

bool flag\_employee = true;

while (flag\_employee)

{

Console.Clear();

Console.ForegroundColor = ConsoleColor.DarkYellow;

Console.WriteLine("[1] Вывод информации о служащих");

Console.WriteLine("[2] Информация о служащем,убиравшим этаж в определенный день");

Console.WriteLine("[3] Принятие на работу нового служащего");

Console.WriteLine("[4] Увольнение служащего");

Console.WriteLine("[5] Выход в главное меню");

Console.ResetColor();

switch (Console.ReadKey(true).KeyChar)

{

case '1':

try

{

Console.Clear();

foreach (var elem in list3)

{

elem.PrintEmployee();

}

Console.Write("Хотите продолжить? ");

string answer = Console.ReadLine();

if (answer == "Нет" || answer == "нет")

{

flag\_employee = false;

}

}

catch

{

Console.WriteLine("Ошибка");

}

break;

case '2':

try

{

Console.Clear();

Console.WriteLine("Введите день недели:");

string day = Console.ReadLine();

Console.WriteLine("Введите этаж:");

int floor = Convert.ToInt32(Console.ReadLine());

foreach (var elem in list3)

{

elem.Cleaning(day, floor);

}

Console.Write("Хотите продолжить? ");

string answer = Console.ReadLine();

if (answer == "Нет" || answer == "нет")

{

flag\_employee = false;

}

}

catch

{

Console.WriteLine("Ошибка,попробуйте еще раз");

}

break;

case '3':

try

{

Console.Clear();

list3.Add(Employee.AddEmployee());

list3.Last().SaveEmployees();

Console.Write("Хотите продолжить? ");

string answer = Console.ReadLine();

if (answer == "Нет" || answer == "нет")

{

flag\_employee = false;

}

}

catch

{

Console.WriteLine("Ошибка,попробуйте еще раз");

}

break;

case '4':

try

{

Employee.Delete(list3);

Console.Write("Хотите продолжить? ");

string answer = Console.ReadLine();

if (answer == "Нет" || answer == "нет")

{

flag\_employee = false;

}

}

catch

{

Console.WriteLine("Ошибка,попробуйте еще раз");

}

break;

case '5':

flag\_employee = false;

break;

default:

Console.WriteLine("Ошибка,попробуйте еще раз");

break;

}

}

break;

case '4':

return;

default:

Console.WriteLine("Ошибка,попробуйте еще раз");

break;

}

}

Console.ReadKey();

}

}

}