**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 3](#_Toc165050057)

[ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ И ОБЩАЯ ПОСТАНОВКА ЗАДАЧ 5](#_Toc165050058)

[1.1 История и развитие гостиничного бизнеса 5](#_Toc165050059)

[1.2 Организационная структура гостиницы 8](#_Toc165050060)

[1.3 Функции, выполняемые администратором гостиницы 12](#_Toc165050061)

[1.4 Постановка задачи 15](#_Toc165050062)

[ГЛАВА 2. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ 17](#_Toc165050063)

[2.1 Требования к программе 17](#_Toc165050064)

[2.2 Руководство пользователя 18](#_Toc165050065)

[2.3 Руководство программиста 27](#_Toc165050066)

[ЗАКЛЮЧЕНИЕ 31](#_Toc165050067)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 32](#_Toc165050068)

[ПРИЛОЖЕНИЕ А 33](#_Toc165050069)

# ВВЕДЕНИЕ

В современных условиях цифровизации и автоматизации процессов информационные системы играют ключевую роль в управлении различными видами деятельности. Организация выставок собак — одно из направлений, где возникает необходимость в систематизации большого объёма данных, связанных с регистрацией участников, распределением экспертов по рингам, оценкой результатов и формированием итогов мероприятия. Проведение таких мероприятий вручную связано с многочисленными трудностями: высокой вероятностью ошибок, затратами времени на обработку информации, сложностью хранения и поиска данных. Всё это требует внедрения специализированного программного обеспечения, способного эффективно решать подобные задачи.

Тема курсовой работы — разработка программной системы, предназначенной для организаторов выставки собак. Данная система должна обеспечивать автоматизированный учет данных о собаках, экспертах, рингах, а также управление информацией о клубах и результатах выставки. Программный продукт должен быть простым в использовании, обеспечивать высокую скорость работы с данными, возможность быстрого поиска, сортировки, фильтрации и внесения изменений. Это позволит организаторам сосредоточиться на проведении мероприятия, доверив рутинные задачи информационной системе.

Актуальность работы заключается в том, что в условиях проведения крупных и массовых выставок ручное ведение записей становится крайне неэффективным. Возникают сложности с координацией участников, отслеживанием данных об экспертах, контролем за соблюдением правил и своевременным подведением итогов. Разработка программной системы, способной автоматизировать перечисленные процессы, повысит уровень организации, снизит нагрузку на персонал, обеспечит прозрачность и точность обработки информации.

Целью настоящей курсовой работы является создание программной системы, которая обеспечит:

-хранение сведений о собаках-участниках и их владельцах;

-учет информации об экспертах, их специализации и клубной принадлежности;

-распределение пород по рингам и экспертов по рингам;

-возможность управления списками участников и экспертов;

-определение медалистов и подведение итогов выставки.

Для достижения поставленной цели необходимо решить следующие задачи:

1)определить основные сущности предметной области и связи между ними;

2)разработать структуру хранения данных (модели: собака, эксперт, ринг);

3)реализовать интерфейс пользователя для ввода, редактирования и просмотра данных;

4)реализовать функции поиска и фильтрации информации по заданным критериям;

5)обеспечить возможность изменения состава участников и экспертов;

6)сформировать отчеты по итогам выставки, включая медалистов и показатели клубов.

Объектом исследования является процесс организации и проведения выставки собак, а предметом — информационные технологии, применяемые для автоматизации данного процесса.

Методы, используемые в работе, включают анализ предметной области, моделирование данных, проектирование архитектуры программной системы и реализацию на основе выбранного языка программирования и среды разработки.

Практическая значимость разработки заключается в возможности её применения в реальных условиях при проведении выставок собак различного уровня. Система может быть использована как отдельными организаторами, так и специализированными клубами и федерациями, благодаря своей гибкости и расширяемости. Таким образом, разработка программной системы для организаторов выставки собак является важной задачей, решающей широкий спектр проблем, связанных с организацией мероприятий, и способствующей повышению качества и прозрачности судейства, а также удобству взаимодействия между участниками, экспертами и организаторами.

# ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ И ОБЩАЯ ПОСТАНОВКА ЗАДАЧ

## 1.1 История и развитие бизнеса выставки собак

Выставки собак являются одним из самых популярных и престижных мероприятий в мире кинологии. Их история насчитывает более полутора столетий и отражает развитие собаководства как науки, искусства и бизнеса. Поначалу выставки носили скорее показательный характер, а сегодня превратились в сложные организационные процессы с активным участием различных клубов, заводчиков, экспертов и коммерческих структур.

Истоки выставок собак относятся к середине XIX века. Первая официально задокументированная выставка прошла в 1859 году в Англии, в городе Ньюкасл-апон-Тайн. В то время мероприятие было узкоспециализированным и посвящалось охотничьим породам — сеттерам и пойнтерам. Основная цель заключалась в демонстрации рабочих качеств и экстерьера собак для повышения племенных стандартов. Эта выставка положила начало развитию кинологических клубов, появлению официальных стандартов и системы оценки собак.

В 1873 году в Великобритании был основан Британский кеннел-клуб (The Kennel Club), который стал первой организацией, занимающейся регулированием выставок и стандартизацией пород. Именно этот клуб начал создавать реестр племенных собак, разрабатывать стандарты пород и утверждать систему судейства. Подобные организации вскоре появились и в других странах Европы — Германии, Франции, а также в России. В Санкт-Петербурге первая крупная выставка собак прошла в 1874 году, и с тех пор кинология в России стала стремительно развиваться.

В XX веке собаководство перестало быть исключительно хобби для аристократии и превратилось в массовое явление. Появились новые породы, расширился круг участников выставок. В Советском Союзе кинологические мероприятия стали важной частью государственной системы, особенно в период послевоенного восстановления, когда активно развивалось служебное собаководство. Выставки служили не только для демонстрации пород, но и для оценки рабочих качеств, служили инструментом улучшения племенного фонда.

С конца XX века и по настоящее время выставки собак развиваются в двух взаимосвязанных направлениях — культурно-спортивном и коммерческом. Международные организации, такие как Fédération Cynologique Internationale (FCI), American Kennel Club (AKC), United Kennel Club (UKC), устанавливают единые правила проведения мероприятий, стандарты оценки собак и требования к экспертам. Крупнейшие выставки — Crufts в Великобритании, Westminster Kennel Club Dog Show в США, Евразия в России — собирают тысячи участников и зрителей, становясь не только спортивными соревнованиями, но и масштабными культурными событиями.

Современный бизнес, связанный с выставками собак, включает в себя широкий спектр услуг и направлений. Важным элементом является разведение и продажа племенных собак, что требует тщательного отбора и подготовки животных, их регистрации, оформления родословных и медицинских документов. Владельцы инвестируют значительные средства в груминг, тренировку, участие в мероприятиях, чтобы повысить престиж питомника и стоимость щенков.

Организаторы выставок работают с большим объемом информации — учет участников, распределение собак по классам и рингам, назначение экспертов, контроль результатов, публикация рейтингов и итогов. В условиях растущего числа участников и усложняющейся регламентации ручное ведение таких процессов становится неэффективным, что стимулирует развитие специализированных программных решений для автоматизации.

Эксперты — важнейший элемент выставок. Они проходят тщательный отбор, обладают высокой квалификацией и специализируются по породам. Их задача — объективно оценить экстерьер, поведение и соответствие собак установленным стандартам. Экспертные комиссии работают в тесной связке с организаторами, что требует точного расписания, распределения ролей и учета пожеланий участников.

Собаководческие клубы играют роль не только в подготовке и поддержке участников, но и в развитии бизнеса вокруг выставок. Клубы предоставляют площадки, поддерживают связь с владельцами и экспертами, участвуют в формировании правил и стандартов. Их деятельность способствует развитию кинологической культуры, повышению качества племенного материала и популяризации пород.

Коммерческая составляющая бизнеса на выставках собак включает оплату участия, спонсорство, продажу оборудования и товаров для животных, услуги ветеринаров, фотосъемку, рекламу и маркетинг. Современные технологии позволяют организовывать онлайн-трансляции, электронную регистрацию, создание электронных каталогов и баз данных, что значительно расширяет аудиторию и повышает качество обслуживания.

Помимо коммерческой выгоды, выставки собак имеют социальное значение. Они способствуют формированию у общества культуры ответственного отношения к животным, популяризируют идеи зоозащиты и гуманного содержания, способствуют развитию науки о поведении и здоровье собак.

В настоящее время рынок кинологических выставок характеризуется высокой конкуренцией, что требует постоянного совершенствования организации и внедрения инновационных решений. Информационные системы, позволяющие автоматизировать учет, управление и анализ данных, становятся необходимым инструментом для успешной деятельности организаторов. Это повышает прозрачность и объективность судейства, облегчает взаимодействие всех участников и способствует развитию отрасли в целом.

Таким образом, история и развитие бизнеса выставок собак отражают эволюцию кинологии от простого демонстрационного мероприятия до сложного и многофункционального процесса, включающего спортивные, культурные и коммерческие аспекты. Современные требования к организации выставок диктуют необходимость внедрения современных технологий, что делает актуальной задачу создания программных систем для поддержки деятельности организаторов выставок.

## 1.2 Организационная структура выставки собак

Организация выставки собак представляет собой сложный и многоуровневый процесс, который требует слаженной работы большого количества специалистов и участников. Для успешного проведения мероприятия необходимо четко выстроить организационную структуру, определить роли и обязанности каждого звена, а также обеспечить эффективное взаимодействие между всеми участниками выставочного процесса. Это позволяет создать условия для прозрачного судейства, комфортного пребывания участников и посетителей, а также гарантирует соблюдение всех стандартов и правил, принятых в кинологической сфере.

Ключевыми участниками выставки собак являются:

1. **Организаторы** — лица или группа лиц, ответственные за планирование, координацию и контроль всего процесса проведения выставки. Они разрабатывают регламент мероприятия, определяют место и время проведения, обеспечивают финансовую и юридическую поддержку, привлекают спонсоров и партнеров. Организаторы также занимаются вопросами регистрации участников, распределением собак по классам и рингам, а также обеспечением логистики и технической поддержки.
2. **Клубы и кинологические объединения** — организации, играющие важную роль в проведении выставок. Клубы выступают как соорганизаторы и предоставляют площадки для проведения мероприятий. Кроме того, клубы обеспечивают связь между заводчиками и экспертами, способствуют популяризации пород, поддерживают профессиональное сообщество и участвуют в формировании правил и стандартов.
3. **Эксперты (судьи)** — квалифицированные специалисты, обладающие глубокими знаниями стандартов пород и опытом оценки собак. Эксперты назначаются на определённые ринги, специализируясь на конкретных породах. Их задача — объективно оценить экстерьер, поведение и соответствие собак стандартам, выставить оценки и определить победителей. На одном ринге может работать несколько экспертов, что повышает точность судейства.
4. **Участники выставки** — заводчики и владельцы собак, регистрирующие питомцев для участия в соревнованиях. Они отвечают за подготовку собак, соответствие требованиям выставки и соблюдение правил поведения.
5. **Административный и технический персонал** — специалисты, обеспечивающие организационно-техническую поддержку выставки. Секретариат, регистраторы, волонтеры, ветеринары, специалисты по грумингу и обслуживанию площадок поддерживают порядок и комфорт на мероприятии.

Для эффективного управления выставкой собак создаётся иерархическая структура с распределением ролей и обязанностей:

1. **Главный организатор (руководитель выставки)** — отвечает за стратегическое планирование, ключевые решения и координацию всех этапов подготовки и проведения мероприятия. Взаимодействует с внешними организациями, спонсорами и кинологическими объединениями.
2. **Заместители и координаторы** — курируют отдельные направления деятельности: работу с участниками, взаимодействие с экспертами, техническое обеспечение, PR и маркетинг, финансовый учет. Обеспечивают оперативное управление и контроль.
3. **Комитет экспертов** — состоит из опытных и авторитетных судей. Отвечает за отбор судей, утверждение критериев оценивания, контроль качества судейства и решение спорных вопросов.
4. **Секретариат** — административное подразделение, занимающееся приемом заявок, обработкой документов, ведением реестров участников и экспертов, подготовкой итоговой документации.
5. **Технические группы** — отвечают за подготовку и оснащение площадок, монтаж оборудования, обеспечение безопасности и обслуживание инфраструктуры.

Эффективное взаимодействие между всеми участниками и службами — ключевой фактор успеха выставки:

* Организаторы обеспечивают прозрачный обмен информацией, своевременную коммуникацию и оперативное решение проблем.
* Эксперты сотрудничают с секретариатом и техническими службами для получения информации, материалов и оборудования.
* Административный персонал поддерживает порядок, помогает с регистрацией, предоставляет информацию и организует расписание.
* Медицинский и ветеринарный персонал контролируют здоровье собак и санитарные нормы, обеспечивая безопасность на выставке.

Ринги — специализированные площадки для оценки собак, каждый из которых имеет специализацию по породам. В одном мероприятии может быть несколько рингов, на каждом последовательно проходят показы различных пород. Это требует четкого расписания и координации.

Распределение экспертов по рингам учитывает их специализацию и опыт. Для бесперебойной работы выставки предусматриваются резервные эксперты, готовые заменить коллег при необходимости.

С развитием технологий все больше процессов автоматизируется:

-Используются программные системы для регистрации участников, формирования расписаний, распределения собак и экспертов, ведения протоколов и статистики.

-Автоматизация снижает человеческий фактор, уменьшает вероятность ошибок и повышает качество обслуживания.

-Электронные базы данных и цифровые протоколы обеспечивают прозрачность судейства и удобство анализа результатов.

-Современные информационные технологии расширяют возможности коммуникации, позволяя оперативно информировать участников, публиковать результаты онлайн, проводить трансляции и рекламные кампании.

Организационная структура выставки собак — сложный механизм, требующий четкой координации и взаимодействия множества специалистов. Хорошо выстроенная структура обеспечивает высокое качество проведения мероприятий, удовлетворяет потребности участников и зрителей, способствует развитию кинологической отрасли.

В современных условиях роль информационных технологий возрастает, делая автоматизацию организационных процессов необходимым условием успешной деятельности организаторов, что повышает уровень сервиса и объективность оценки.

## 1.3 Функции, выполняемые администратором выставки собак

Администратор выставки собак играет ключевую роль в организации и успешном проведении мероприятия. Его деятельность охватывает широкий спектр задач, начиная с этапа подготовки и заканчивая подведением итогов, обеспечивая эффективное взаимодействие всех участников и служб. Без слаженной работы администратора проведение выставки было бы невозможно, так как именно он отвечает за координацию процессов и контроль выполнения регламента.

История появления должности администратора связана с усложнением структуры и масштабов выставок собак. В начале организационные задачи выполнялись непосредственно владельцами клубов или главными организаторами, однако с ростом числа участников и усложнением регламентов возникла необходимость в выделении отдельной должности — администратора, который специализируется на управлении всеми организационными процессами выставки. Со временем роль администратора расширялась, и теперь она включает в себя не только организационные функции, но и взаимодействие с техническими службами, спонсорами, а также контроль за соблюдением правил и нормативов.

Сегодня обязанности администратора охватывают все этапы выставки — от планирования до подведения итогов. Ниже перечислены основные функции, которые выполняет администратор на современном этапе организации выставок собак.

1. Разрабатывает и реализует план проведения выставки, определяя основные этапы подготовки и согласовывая график работы с руководителями клубов, экспертами и техническими службами. Это позволяет обеспечить четкое и слаженное выполнение всех этапов мероприятия, минимизировать риски и оперативно реагировать на изменения.

2. Организует процесс регистрации участников — собак и экспертов, ведет реестры и проверяет документы. В его обязанности входит контроль правильности заполнения заявок, своевременной оплаты регистрационных взносов, а также внесение всей необходимой информации в электронные базы данных. Такая систематизация обеспечивает прозрачность и удобство управления информацией.

3. Координирует работу экспертов и распределяет их по рингам согласно специализации. Администратор следит за тем, чтобы каждый ринг был укомплектован квалифицированными экспертами, обладающими опытом работы с конкретными породами собак, что обеспечивает объективность и качество судейства.

4. Контролирует подготовку площадок и техническое обеспечение выставки, включая монтаж оборудования, организацию звукового сопровождения, освещения и работу электронных табло. Он взаимодействует с техническими специалистами и подрядчиками, гарантируя, что все системы работают без сбоев и соответствуют требованиям безопасности.

5. Ведет документацию по итогам мероприятия — протоколы судейства, сертификаты, дипломы и отчеты. Этот пункт является важной частью административной работы, так как обеспечивает официальное оформление результатов выставки и позволяет сохранить всю информацию для последующего анализа и отчетности.

6. Оперативно решает возникающие организационные вопросы, включая замену экспертов и отстранение участников, а также другие внештатные ситуации. Благодаря этому обеспечивается бесперебойное проведение мероприятия и соблюдение его регламента.

В современной практике различают несколько разновидностей администраторов, каждый из которых отвечает за отдельный блок задач и сферу ответственности. Такое разделение обязанностей позволяет повысить эффективность работы всей команды и обеспечить качественное проведение выставки.

1. Администратор регистрации — специализируется на приеме заявок участников, проверке документов, ведении базы данных и сопровождении процесса регистрации. Он обеспечивает, чтобы все данные были корректно оформлены и вовремя внесены в систему.

2. Технический администратор — отвечает за подготовку площадок, организацию технического оснащения выставки. Он контролирует монтаж оборудования, настройку звуковых и световых систем, работу электронных табло и других технических средств.

3. Администратор рингов — координирует работу судей и экспертов на местах проведения выставок, следит за соблюдением регламента и расписания. Его задача — обеспечить комфортные условия для проведения состязаний и своевременное выполнение всех этапов.

4. Администратор по работе с участниками — является связующим звеном между организаторами и участниками. Он решает возникающие вопросы, консультирует участников, оказывает помощь при необходимости, организует информирование и обратную связь.

5. Администратор по безопасности — контролирует порядок и соблюдение правил безопасности на территории выставки. Он отвечает за санитарные нормы, охрану, а также за профилактику и оперативное реагирование на возможные чрезвычайные ситуации.

6. Финансовый администратор — управляет финансовыми аспектами выставки, включая учет оплаты регистрационных взносов, работу со спонсорами и бухгалтерией, контроль бюджета мероприятия. Его задача — обеспечить прозрачность и своевременность финансовых операций.

Таким образом, роль администратора выставки собак комплексна и многоаспектна. Его деятельность направлена на обеспечение высокого качества организации и проведения выставок, создание комфортных условий для участников и гостей, а также поддержание репутации клуба и уровня мероприятия. Хорошо слаженная работа администрации является залогом успеха любой выставки.

## 1.4 Постановка задачи

Общая задача данной курсовой работы заключается в разработке информационной системы для администратора выставки собак, которая будет обеспечивать следующие основные функции:

1. Учёт собак-участников выставки с такими данными, как: кличка, порода, возраст, клуб, родословная, дата последней прививки, ФИО и паспортные данные владельца.
2. Учёт экспертов: ФИО, специализация по породе, номер ринга, клуб, в котором состоит эксперт.
3. Учёт рингов: название, адрес, специализация по породам, закреплённые эксперты и собаки.
4. Управление выставочными мероприятиями: регистрация новых участников и экспертов, замена судей, отстранение собак от участия.
5. Поиск, сортировка, фильтрация информации, а также удаление данных из системы при необходимости.

Задачи, решаемые в рамках работы, включают:

1. Проведение анализа потребностей организаторов выставок собак и формализация требований к программной системе с учётом специфики предметной области.
2. Выбор оптимальных технологий разработки, включая язык программирования, среду разработки и средства хранения данных.
3. Реализация программного кода, обеспечивающего основные функции системы: регистрацию собак, учёт экспертов, управление рингами, проведение операций по отстранению/замене и другие административные действия.
4. Проведение тестирования системы на корректность работы, проверка устойчивости при вводе некорректных данных и обеспечение удобного пользовательского интерфейса.
5. Оценка эффективности разработанной системы на основе сценариев использования, сбор отзывов, формирование предложений по дальнейшему расширению функционала.

Результатом выполнения курсовой работы станет полноценная информационная система для администрирования выставки собак. Система позволит автоматизировать процесс управления участниками и экспертами, упростить организацию рингов и повысить оперативность в работе организаторов, способствуя более качественному и прозрачному проведению выставочных мероприятий.

# ГЛАВА 2. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ

## Требования к программе

Для корректного функционирования программы необходимо как минимум 120 Мб свободного места на жестком диске. Операционная система должна быть не младше Microsoft Windows 7, включая Windows 8 и Windows 10, поддерживающая 64-битные версии. Программа должна стабильно работать при длительной эксплуатации компьютеров. В случае сбоев в работе аппаратной части необходимо обеспечить возможность восстановления функциональности программы без потери данных, в том числе и их автоматическое резервное копирование.

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

* Процессор: AMD 3020е с тактовой частотой до 2.6 ГГц.
* Оперативная память: 8 ГБ LPDDR4.
* Видеокарта: Интегрированная AMD Radeon Graphics Vega 3.
* Операционная система: Windows 10 Pro 64-битная версия.
* Накопитель: SSD на 256 ГБ.
* Дисплей: 14-дюймовый Full HD (1920 x 1080).

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;
* ОЗУ: не менее 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

Руководство пользователю по использованию консольного приложения информационной системы по управлению гостиницей предназначено для ознакомления пользователя с функциональными возможностями приложения, поможет научится использовать его.

Руководство пользователя обеспечивает полную информативность по структуре интерфейса приложения, описывает все его доступные функции.

Запуск приложения

Чтобы запустить приложение, нужно дважды кликнуть левой кнопкой мыши (ЛКМ) по исполняемому файлу — файл с расширением .exe. Данное действие показано на рисунке 1.

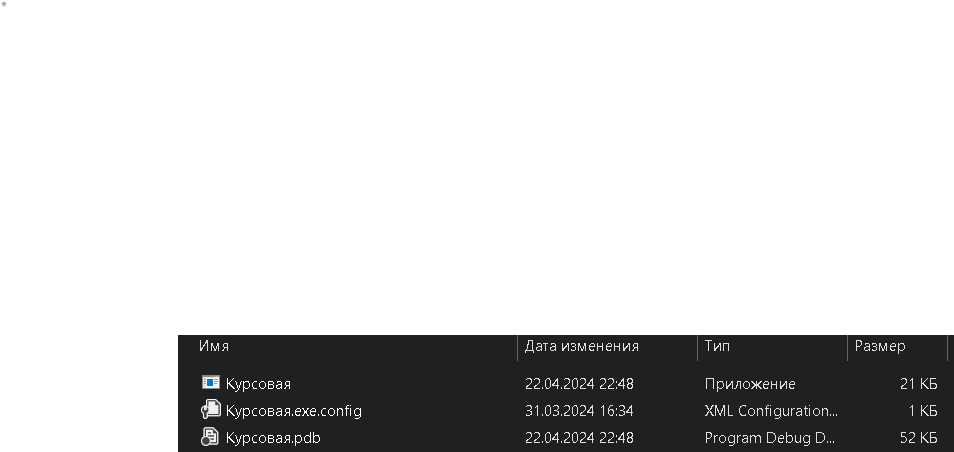


Рис. 1 Запуск приложения

Первое меню

После запуска приложения откроется первое меню, его можно увидеть на рисунке 2, на нем располагается функционал для переключения, реализованный при помощи клавиш. Нажав на клавишу на клавиатуре возможен переход на действие, которое необходимо пользователю.

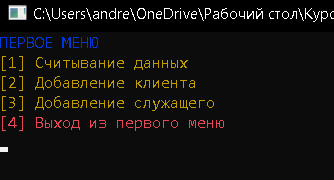


Рис. 2 Первое меню программы

Считывание данных

После перехода на первый пункт, который осуществляет считывание данных из файла в программу пользователь может увидеть надпись в случае успешного считывания данных, и в случае неудачного считывания данных как показано на рисунке 3.

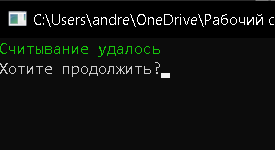


Рис. 3 Успешное считывание данных

Добавление клиента

Если пользователю необходимо добавить нового клиента в базу данных ему необходимо перейти на второй пункт и ввести все необходимые данные, данное действие показано на рисунке 4.

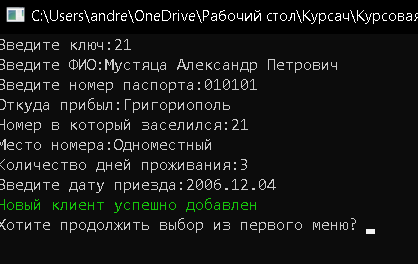


Рис. 4 Добавление клиента

Добавление служащего

После перехода на этот пункт пользователь может добавить нового служащего в файл. Данное действие показано на рисунке 5.

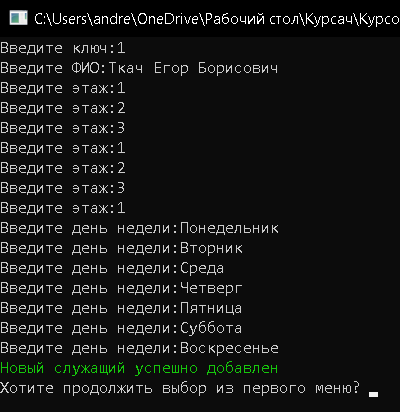


Рис. 5 Добавление служащего

Выход из первого меню

В случае если пользователь произвел все необходимые действия он может перейти в главное меню, что показано на рисунке 6.

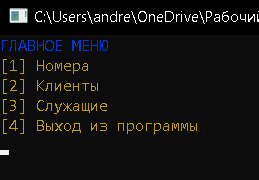


Рис. 6 Переход в главное меню

Главное меню

В главном меню пользователь может выбрать необходимые ему разделы: номера, клиенты, служащие.

Раздел Номера

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

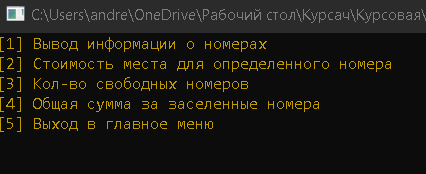


Рис. 7.1 Раздел номер

Вывод информации о номерах

Перейдя по этому пункту, пользователь может вывести на экран информацию обо всех номерах в гостинице (рис. 7.2).

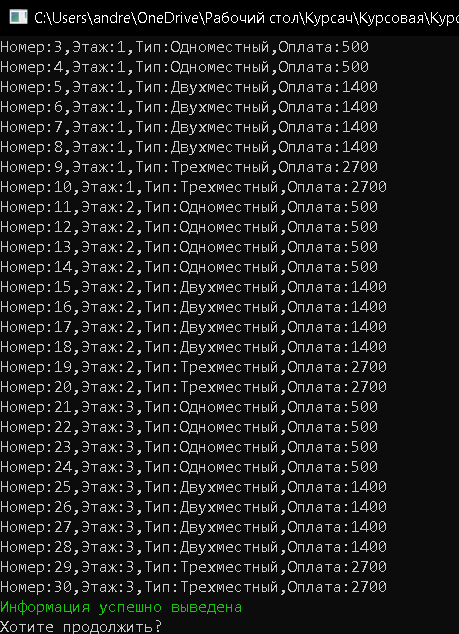


Рис. 7.2 Вывод информации о номерах

Стоимость определенного номера

Этот пункт может понадобиться пользователю в том случае, если необходимо узнать стоимость одной ночи в определенном номере для клиента, который хочет заселиться (рис 7.3).

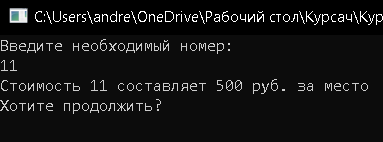


Рис. 7.3 Стоимость определенного номера

Количество свободных номеров

Нажав на клавиатуре клавишу 3, администратор сможет увидеть информацию о незаселенных номерах (рис 7.4).

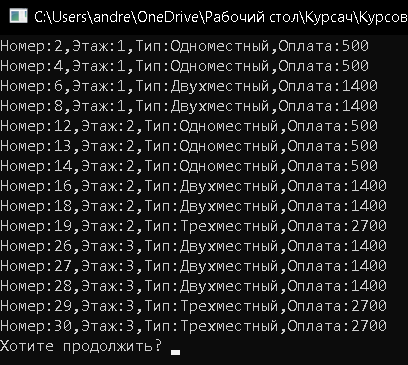


Рис. 7.4 Количество свободных номеров

Сумма, которую выплачивают за заселенные номера

Перейдя на данный пункт можно увидеть надпись, которая сообщает о сумме, которую оплатили все клиенты за оплату своих номеров (рис. 7.5).

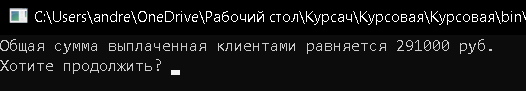


Рис. 7.5 Сумма, выплаченная клиентами

Выход в главное меню

Данный пункт реализует возвращение пользователя в главное меню для дальнейшего выбора.

Раздел Клиенты

В данном разделе пункты привязаны к клавишам нажимая на которые, пользователь переходит на различные пункты (рис. 8.1).

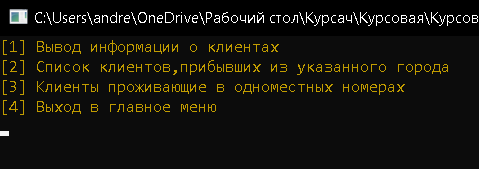


Рис. 8.1 Раздел Клиенты

Вывод информации о клиентах

Переход по этому пункту позволяет администратору увидеть полную информацию о клиентах (рис. 8.2).

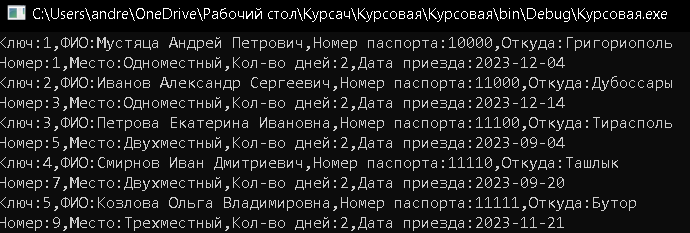


Рис. 8.2 Вывод информации о клиентах

Список клиентов, прибывших из указанного номера

Переход по данному пункту позволяет вывести информацию о клиентах, прибывших из введенного с клавиатуры города. В случае если клиентов из введенного города нет выводится соответствующая надпись (рис. 8.3).

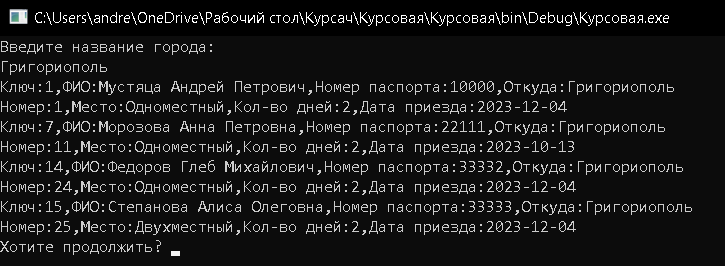


Рис. 8.3 Клиенты, прибывшие из определенного города

Клиенты, проживающие в одноместных номерах

Данный пункт необходим, если пользователю необходимо вывести список клиентов, проживающих в одноместных номерах (рис. 8.4).

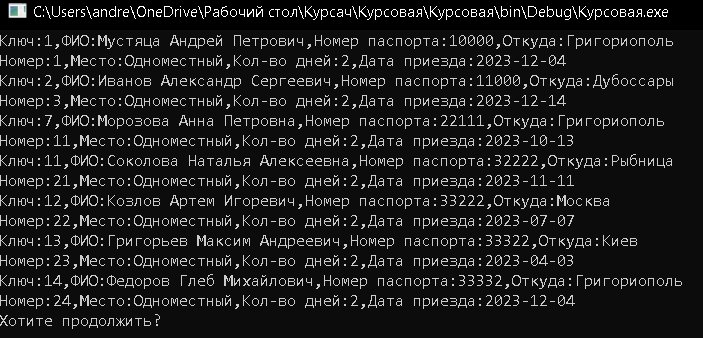


Рис. 8.4 Клиенты, проживающие в одноместных номерах

Выход в главное меню

Данный пункт реализует возвращение пользователя в главное меню для дальнейшего выбора.

Раздел Служащие

В данном разделе пользователь может выбрать действия, которые администратор может выполнять со служащими.

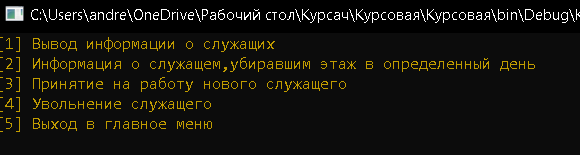


Рис. 9.1 Раздел Служащие

Вывод информации о служащих

Данный пункт позволяет вывести информацию о служащих, которые работают в гостинице (рис. 9.2).

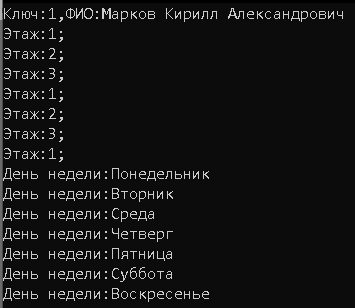


Рис. 9.2 Вывод информации о служащих

Информация о служащем, убиравшим этаж в определенный день недели

Данный пункт поможет пользователю выяснить какой именно служащий убирал определенный этаж в определенный день недели (рис. 9.3).

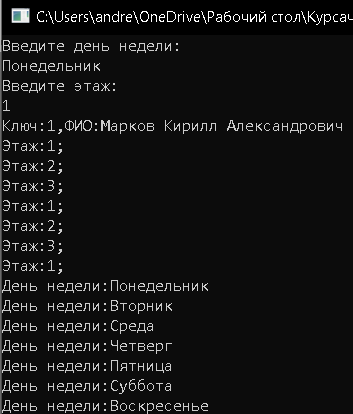


Рис. 9.3 Поиск служащего

Принятие на работу служащего

Перейдя по данному пункту администратор гостиницы может принять на работу нового служащего, введя его данные (рис 9.4).

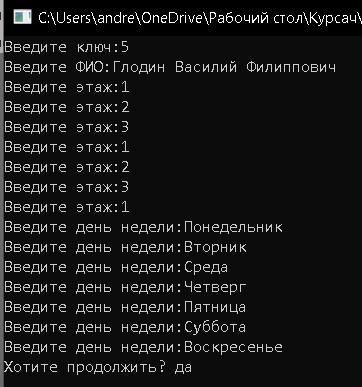


Рис. 9.4 Добавление нового служащего

Увольнение служащего

Переход по данному пункту позволяет в случае необходимости уволить служащего, что будет также отражено в базе данных (рис. 9.5).



Рис. 9.5 Увольнение служащего

Выход из программы

Данный пункт завершает программу.

Это основное руководство пользователю, которое поможет ознакомиться с функциональностью и использованием приложения

## Руководство программиста

Представленный программный продукт представляет собой консольное приложение. В разработке программного продукта реализовано множество алгоритмов.

Структура кода включает в себя:

1. Класс **Hotel** представляет собой класс, описывающий номера в гостинице. Данный класс содержит в себе следующие свойства:

* Number: свойство типа int, обозначающее номер отеля.
* Floor: свойство типа int, обозначающее этаж отеля.
* Type: свойство типа string, обозначающее тип номера (одноместный, двухместный или трехместный).
* Cost\_per\_day: свойство типа int, отвечающее за стоимость одной ночи в номере определенного типа.

Также в данном классе реализованы два поля:

* List1: поле, представляющее собой коллекцию объектов класса Client.
* List2: поле, представляющее собой коллекцию объектов класса Employee.

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

* public Hotel: конструктор, позволяющий инициализировать все свойства и поля класса Hotel.
* public void SaveHotel: метод записи информации о номере в файл.
* public static Hotel ReaderNumbers: метод позволяющий считать с файла строку и добавить ее в коллекцию в качестве объекта класса Hotel.
* public void Print: метод вывода информации о номерах гостиницы.
* public int Cost: метод, вычисляющий стоимость одной ночи в номере определенного типа.
* public void Empty\_number: метод, позволяющий выяснить какой из номеров является не заселенным.

1. Класс **Client** представляет собой класс, описывающий клиентов гостиницы. Данный класс содержит следующие свойства:

* Key: свойство типа int, обозначающее ключ клиента.
* FIO: свойство типа string, обозначающее ФИО клиента.
* Number\_pasport: свойство типа int, обозначающее номер паспорта клиента.
* Number: свойство типа int, обозначающее номер в который клиент заселен.
* Place: свойство типа string, обозначающее тип номера в который клиент заселен.
* Kol\_vo\_days: свойство типа int, обозначающее количество дней, которые клиент планирует прожить в номере.
* Date: свойство типа DateTime, обозначающее дату приезда клиента.

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

* public Client: конструктор, позволяющий инициализировать все свойства класса.
* public void SaveClients: метод для записи информации о клиенте в файл.
* public static Client ReaderClient: метод для считывания строки с файла и ее возвращения в виде объекта класса Client.
* public void PrintClient: метод для вывода информации о клиентах.
* public static void City: метод позволяющий выводить список клиентов, прибывших из определенного города.
* public void Single: метод позволяющий вывести выводить список клиентов, живущих в одноместных номерах.

1. Класс **Employee** представляет собой класс, описывающий работников гостиницы. Данный класс содержит следующие свойства и поля:

* Key: свойство типа int, представляющее собой ключ работника.
* FIO: свойство типа string, описывающее ФИО работника.
* floors: поле представляющее собой коллекцию типа int, описывающую этажи, которые убирает работник.
* days: поле представляющее собой коллекцию типа string, описывающую дни недели в которые работник работает.

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

* public Employee: конструктор, позволяющий инициализировать свойства класса.
* public void SaveEmployees: метод записи в файл.
* public static Employee ReaderEmployee: метод позволяющий считывать строку с файла и возвращать ее в качестве объекта класса **Employee**.
* public void PrintEmployee: метод, позволяющий вывести информацию о работнике.
* public void Cleaning: метод поиска работника, убирающего определенный этаж в определенный день недели.
* Public static void Delete: метод удаления (увольнения) работника.

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

# ЗАКЛЮЧЕНИЕ

В данной курсовой работе была разработана информационная система для администратора гостиницы. Программа реализована на языке программирования C# и в качестве хранилища данных используется три файла.

Пользовательский интерфейс создан с применением консольных средств ввода-вывода, что обеспечивает простоту навигации и понятные команды для управления программой. Интерфейс позволяет удобно взаимодействовать с системой, вводить, осуществлять поиск и выводить информацию.

Информация о клиентах, номерах и служащих разделена и находится отдельно в трех текстовых файлах.

Основные преимущества разработанной системы включают в себя:

1. Интуитивно понятный и легко осваиваемый интерфейс обеспечивает комфортное взаимодействие пользователя с системой.

В процессе работы были выявлены некоторые области, где можно улучшить систему:

1. Расширение функциональности системы. Например, бронирование номеров.
2. Улучшение интерфейса консольного приложения для более удобного взаимодействия пользователя с системой, или же замена интерфейса на более презентабельный вид.
3. Интеграция с внешними сервисами или базами данных для получения дополнительной информации о клиентах, служащих, такой как номер телефона, страховой полис и т.д.

Цель работы была успешно достигнута - разработана информационная система, позволяющая управлять процессом управления гостиницей.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Албахари Дж., Албахари Б.: C# 8.0. Карманный справочник - Москва: ООО «И. Д. Вильямс», 2020. – 240 с.
2. Антон Марченко: Основы программирования на С# 2.0. Учебное пособие – 2017. – 557 с.
3. Шилдт, Г. C# 4.0: полное руководство. - Москва: ООО «И. Д. Вильямс», 2015. - 1056 с.
4. Лекция «Службы гостиницы»

(<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>)

1. «История гостиничного дела»

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

# ПРИЛОЖЕНИЕ А

class Employee

{

/// <summary>

/// Свойства

/// </summary>

public int Key { set; get; }

public string FIO { set; get; }

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

public List<string> days = new List<string>(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}");

}

}

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");

}

}

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

{

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")}");

}

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");

}

}

public void Single()

{

if(Place=="Одноместный")

{

PrintClient();

}

}

}

class Hotel

{

/// <summary>

/// Свойства класса

/// </summary>

public int Number { set; get; }

public int Floor { set; get; }

public string Type { set; get; }

public int Cost\_per\_day { set; get; }

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

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

/// <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}");

}

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;

}

}

public void Empty\_number()

{

if(list1.Count==0)

{

Print();

}

}

}

class Program

{

static void Main(string[] args)

{

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

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

List<Employee> list3 = new List<Employee>();

bool flag = true;

//Добавление и запись

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,попробуйте 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();

}

}