Министерство просвещения ПМР

ГОУ СПО «Тираспольский техникум информатики и права»

**Курсовая работа**

**на тему «Разработка информационной системы регистрации клиентов гостиницы с применением концепций объектно-ориентированного программирования на языке С#»**

**Специальность 2.09.02.07 «Информационные системы и программирование»**

**Исполнитель** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И.А. Тизул

(подпись)

**Руководитель**

**курсовой работы** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю.В. Балашова

(подпись)

**Курсовая работа допущена к защите «**\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

**Заместитель директора**

**по учебной работе** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Л.И. Гончарук

(подпись)

Тирасполь 2024

**Оглавление**

[**Введение** 3](#_Toc169262751)

[**Глава 1. Теоретический обзор** 5](#_Toc169262752)

[**1.1. Основные принципы объектно-ориентированного программирования** 5](#_Toc169262753)

[**1.2. Преимущества применения ООП в разработке информационных систем** 10](#_Toc169262754)

[**1.3. Сравнение ООП с другими парадигмами программирования** 12](#_Toc169262755)

[**Глава 2. Разработка информационной системы регистрации клиентов гостиницы** 15](#_Toc169262756)

[**2.1. Описание структуры и функционала программы** 15](#_Toc169262757)

[**2.2. План создания и тестирования программы** 16](#_Toc169262758)

[**2.3 Анализ эффективности программы** 18](#_Toc169262759)

[**2.4. Перспективы дальнейшей разработки программы** 20](#_Toc169262760)

[**2.5. Руководство для программиста** 21](#_Toc169262761)

[**Заключение** 24](#_Toc169262762)

[**СПИСОК ЛИТЕРАТУРЫ** 26](#_Toc169262763)

# **Введение**

В современном мире информационные технологии играют ключевую роль в различных отраслях, включая гостиничный бизнес. Эффективное управление данными о клиентах и автоматизация бизнес-процессов позволяет повысить качество обслуживания и оптимизировать операционные затраты. Информационная система регистрации клиентов является важным инструментом для любой гостиницы, так как она позволяет автоматизировать процессы управления данными о клиентах, улучшить точность и скорость обслуживания, а также обеспечить надежное хранение информации.

Целью данной курсовой работы является разработка информационной системы регистрации клиентов гостиницы с применением концепций объектно-ориентированного программирования на языке C#. В рамках работы будут рассмотрены основные принципы ООП, преимущества их применения в разработке информационных систем, проведен анализ требований к системе, спроектировано хранение информации о клиентах, разработан пользовательский интерфейс и реализована бизнес-логика системы.

Объектно-ориентированное программирование (ООП) является одним из наиболее популярных и эффективных подходов к разработке программного обеспечения. Его основные принципы, такие как инкапсуляция, наследование и полиморфизм, позволяют создавать гибкие, модульные и легко поддерживаемые системы. Применение этих принципов в разработке информационной системы для гостиницы позволяет создать функциональное и надежное приложение, которое удовлетворяет потребности пользователей и соответствует современным стандартам разработки ПО.

Актуальность данной темы обусловлена тем, что современные гостиницы стремятся к максимальной автоматизации своих процессов, что требует создания надежных и эффективных информационных систем. Это позволяет не только улучшить качество обслуживания клиентов, но и существенно снизить операционные затраты. В условиях высокой конкуренции на гостиничном рынке информационные системы играют решающую роль в повышении конкурентоспособности предприятия.

Для достижения цели данной курсовой работы были поставлены следующие задачи:

Рассмотреть теоретические аспекты объектно-ориентированного программирования.

1.Проанализировать требования к информационной системе регистрации клиентов гостиницы.

2.Спроектировать структуру хранения информации о клиентах.

3.Разработать пользовательский интерфейс системы.

4.Реализовать бизнес-логику системы с применением принципов ООП на языке C#.

5.Провести тестирование и отладку разработанной системы.

Объектом исследования является информационная система регистрации клиентов гостиницы. Предметом исследования является процесс разработки данной системы с применением концепций объектно-ориентированного программирования.

Методы исследования включают анализ и синтез теоретических знаний, проектирование системы, программирование, а также тестирование и отладку программного обеспечения. Используемый методический подход позволит получить комплексное представление о процессе разработки информационной системы, а также выявить и устранить возможные проблемы на всех этапах создания системы.

Данная курсовая работа структурирована следующим образом. В первой главе рассматриваются теоретические аспекты ООП и его преимущества при разработке информационных систем. Во второй главе описывается процесс разработки информационной системы регистрации клиентов гостиницы, включая анализ требований, проектирование структуры хранения данных, разработку пользовательского интерфейса и реализацию бизнес-логики. И тестирование откладки системы.

# **Глава 1. Теоретический обзор**

## **1.1. Основные принципы объектно-ориентированного программирования**

Основные принципы объектно-ориентированного программирования (ООП) определяют способ организации кода и работы с данными в программах, основанных на объектах.

1.Инкапсуляция: Этот принцип позволяет скрыть детали реализации объекта от внешнего мира и предоставить доступ к ним только через определенный интерфейс. Объекты могут содержать в себе данные (поля) и методы, которые могут использоваться для манипуляции этими данными. При инкапсуляции данные объекта доступны только через методы, что обеспечивает безопасность и целостность данных.

Например, в примере выше класс Person имеет поле age, которое хранит возраст. При прямом доступе к полю age ему можно присвоить любое значение, в том числе и некорректно значение возраста, например:

Person tom = new Person("Tom", 39);

tom.Age = 12345;

2.Наследование: Этот принцип позволяет создавать новые классы на основе существующих (родительских) классов. Наследование позволяет расширять функциональность существующего класса, добавляя новые поля и методы или изменяя существующие. Это способствует повторному использованию кода и созданию иерархии классов, что упрощает структуру программы.

Наследование в какой-то степени похоже с биологическим наследованием. Вы получаете какие-то черты от своих родителей, но, в то же время, отличаетесь от них. Или представьте это как базовую модель гаджета, к которой затем добавляются улучшенные версии с дополнительными функциями. Давайте рассмотрим несколько примеров, чтобы лучше понять это.

Пример: Версии смартфона

Рассмотрим смартфон, у которого есть базовая модель и есть версия Pro, которая наследует все базовые функции, плюс, добавляет некоторые продвинутые.

public class BasicSmartphone

{

public void Call()

{

Console.WriteLine("Совершаем звонок...");

}

}

public class ProSmartphone : BasicSmartphone

{

public void VideoCall()

{

Console.WriteLine("Совершаем видеозвонок...");

}

}

ProSmartphone может звонить так же, как и BasicSmartphone, но также имеет дополнительную функцию видеозвонок.

3.Полиморфизм: Этот принцип позволяет объектам одного и того же базового класса обнаруживать различное поведение при вызове одного и того же метода. Полиморфизм может быть реализован через перегрузку методов (когда у метода одного имени есть несколько реализаций) или через наследование (когда классы-потомки могут переопределять методы классов-родителей).

Полиморфизм немного напоминает универсальный пульт дистанционного управления, который может адаптироваться для управления различными устройствами. В программировании это означает, что один интерфейс может использоваться для управления разными методами, давая разные результаты в зависимости от контекста.

Пример: Виртуальный ассистент

Подумайте о виртуальном ассистенте, который работает на смартфоне, смарт-часах и смарт-колонке. Вы можете попросить все эти устройства "Включить свет", но ответ может быть адаптирован в зависимости от устройства.

|  |
| --- |
| public class VirtualAssistant  {  public virtual void ExecuteCommand(string command)  {  Show($"Выполняю команду {command}...");  }  }  public class SmartwatchAssistant : VirtualAssistant  { |

public override void ExecuteCommand(string command)

{

ShowOnSmallScreen($"Выполняю команду {command}...");

}

}

public class SmartSpeakerAssistant : VirtualAssistant

{

public override void ExecuteCommand(string command)

{

Say($"Выполняю команду {command}...");

}

}

Команда одинакова, но ее выполнение адаптируется в зависимости от контекста устройства. В базовом случае мы просто выводим сообщение о том, что команда выполняется, на экран (Show). У умных часов экран маленький, поэтому нам нужен особый способ вывода сообщения на экран (ShowOnSmallScreen), а у умной колонки вообще может не быть экрана, поэтому сообщение лучше озвучить голосом (Say).

4.Абстракция: Этот принцип позволяет отделить сущности программы от их реализации и представления, сосредоточив внимание на их существенных характеристиках. Абстракция позволяет создавать абстрактные классы и интерфейсы, описывающие общие свойства и методы объектов, что упрощает процесс проектирования и понимание кода.

Применение этих принципов в разработке программного обеспечения обеспечивает более чистый, модульный и легко поддерживаемый код, что является ключевым преимуществом объектно-ориентированного подхода к программированию.Начало формы

объектно-ориентированном программировании основу или асбстракцию для объектов определяют классы. Классы содержат общее состояние и поведение объектов. Например, нам надо представить в программе человека. В большинстве языков программирования для определения классов применяется ключевое слово class. Так, в С# мы могли бы определить следующий класс Person, который представляет человека:

public class Person{}

В процессе определения абстрации объектов - класса мы абстрагируемся от конкретных признаков объектов и выделяем общие для них характеристики и поведение. Набор общих признаков объектов образует состояние класса. Так, мы можем выделить у человека такие признаки как имя и возраст. Эти характеристики будут представлять состояние. Для определения состояния в классах обычно используются поля или переменные класса (в некоторых языках их называют свойствами, в других языках свойства и поля класса разделяются). Например, в С# мы могли бы определить состояние следующим образом:

public class Person

{

// Поля класса для хранения состояния объекта

public string Name; // Имя человека

public int Age; // Возраст человека

}

## **1.2. Преимущества применения ООП в разработке информационных систем**

Плюсы, минусы и перспективы ООП

Программисты знакомы с объектно-ориентированным программированием. В таких языках создается виртуальный объект, а из объектов создается приложение. Понять ООП (объектно-ориентированное программирование) можно на примере компьютерной игры. Допустим, в ней создан объект «персонаж», который имеет свойства (оружие, уровень здоровья, силы). У объектов есть методы, персонаж ходит, сражается, собирает предметы. Популярные языки ООП: C#, Java, PHP, Python.

Преимущества ООП

Технология создания виртуальных объектов имеет много преимуществ. Программный код легко разобрать. К другим плюсам относятся:

При написании больших проектов без объектно-ориентированного программирования не обойтись. На написание не потребуется много времени.

Удобство. Программист создает сущность, а она работает с программой.

Повторять одно и то же не потребуется. Достаточно задать одну сущность и вставлять ее на протяжении всего кода в нужные места.

Виртуальный объект создается для выполнения определенных задач. Программы надежно защищены от атак извне. Это происходит за счет скрытия переменных внутри классов. Также объекты используют в других приложениях.

Минусы ООП

Объектно-ориентированное программирование не лишено недостатков. Среди недостатков выделяют:

Часть программ требует больших ресурсов оборудования.

На сложные объекты затрачивается больше оперативной памяти.

Изучение объектно-ориентированного программирования сложнее, чем функционального программирования. На него потребуется больше временных ресурсов.

Повторное использование объектов не представляется возможным. Это объясняется их привязкой к определенным классам, что затрудняет использование с другими классами. Сложность языка вызывает затруднение у начинающих программистов.

Перспективы ООП

С помощью объектно-ориентированного программирования программа делится на самостоятельные части. Доступ к содержимому предоставляется посредством сообщений. Изначально технология не предполагала наследование. Это когда программист создает подкласс со свойствами, как у родителя. Полиморфизм также не входил в первоначальные планы. Он означает, что метод или объект может стать шаблоном для других методом и объектов.

До появления ООП существовало другое программирование. Разработчик писал машинный код. Для этого нужны были знания внутреннего устройства компьютера. Другой стороной технологии является то, что разработчик объектно-ориентированного программирования воспринимает только такой подход и пропускает через него все задачи.

У ООП есть несколько проблем:

Зависимость подклассов от родителей, что делает необходимым написание лишнего кода.

При изменении класса код может перестать работать корректно.

Увеличение свойств класса приводит к усложнению иерархии.

Хотя у объектно-ориентированного программирования есть проблемы, оно будет популярно еще не менее 20-30 лет и больше. Но параллельно с этим будут появляться и другие языки программирования, на которые стоит обратить внимание. Чем богаче арсенал программиста, тем больше он будет востребован. Появление такой технологии перенесло внимание с машины на человека. А программистов стали сравнивать с проектировщиками. Сейчас ООП широко используется для написания больших и сложных проектов и программ.

## **1.3. Сравнение ООП с другими парадигмами программирования**

Объектно-ориентированное программирование (ООП) – одна из наиболее популярных парадигм разработки программного обеспечения, однако существуют и другие парадигмы, каждая из которых имеет свои особенности и области применения. В данном разделе мы рассмотрим основные отличия ООП от процедурного, функционального и логического программирования.

Процедурное программирование

Процедурное программирование – это парадигма, в которой программа представляет собой последовательность инструкций, разделенных на процедуры или функции. Основные характеристики процедурного программирования:

Структура программы: Программа делится на процедуры и функции, каждая из которых выполняет определенную задачу.

Использование переменных: Переменные используются для хранения данных и изменяются по мере выполнения программы.

Поток управления: Контроль выполнения программы осуществляется через управляющие структуры, такие как циклы и условные операторы.

Отличия от ООП:

Инкапсуляция: В ООП данные и методы, работающие с этими данными, объединяются в одном классе. В процедурном программировании данные и функции существуют отдельно.

Наследование и полиморфизм: В ООП классы могут наследовать свойства и методы других классов, что упрощает повторное использование кода и его расширение. Процедурное программирование не поддерживает эти концепции.

Модульность: ООП позволяет создавать более модульные и легко расширяемые программы, в то время как процедурное программирование может привести к созданию монолитного и трудноподдерживаемого кода.

Функциональное программирование

Функциональное программирование – это парадигма, в которой основное внимание уделяется вычислению значений через вызов функций. Основные характеристики функционального программирования:

Иммутабельность данных: Данные не изменяются после создания. Вместо изменения данных создаются новые.

Функции высшего порядка: Функции могут принимать другие функции в качестве аргументов и возвращать их в качестве результатов.

Чистые функции: Функции не имеют побочных эффектов и всегда возвращают один и тот же результат для одних и тех же входных данных.

Отличия от ООП:

Состояние и изменение данных: В ООП объекты могут изменять свое состояние, в то время как в функциональном программировании данные остаются неизменными.

Модель вычислений: В ООП основной единицей является объект, в функциональном программировании – функция.

Параллелизм: Функциональное программирование, благодаря иммутабельности данных и чистым функциям, лучше подходит для реализации параллельных вычислений.

Логическое программирование

Логическое программирование – это парадигма, в которой программа представляет собой набор логических утверждений и правил. Основные характеристики логического программирования:

Декларативный подход: Программист описывает, что должно быть достигнуто, а не как это должно быть сделано.

Использование фактов и правил: Программа состоит из фактов и правил, которые используются для вывода новых фактов.

Решение задач: Программа решает задачи путем поиска доказательств для логических утверждений.

Отличия от ООП:

Способ описания программ: В ООП программа описывается через взаимодействие объектов, в логическом программировании – через логические утверждения и правила.

Контроль выполнения: В ООП программист управляет потоком выполнения программы, в логическом программировании управление осуществляется автоматически через систему вывода.

Применимость: Логическое программирование часто используется в системах искусственного интеллекта и для задач, требующих сложного логического вывода, тогда как ООП более универсально и применяется в широком спектре областей.

# **Глава 2. Разработка информационной системы регистрации клиентов гостиницы**

## **2.1. Описание структуры и функционала программы**

Программа разрабатывается на языке C# с использованием объектно-ориентированного подхода для управления данными о клиентах гостиницы через Windows Forms. Основные компоненты программы включают:

-RegisterClient – этот метод будет выполняться при нажатии на кнопку регистрации клиента на форме Form1. Он создает новый объект класса Client на основе введенных данных, добавляет его в список клиентов и сохраняет информацию о клиенте в файл Base.txt.

-Класс Client содержит основную информацию о клиентах, необходимую для работы с системой управления клиентами. Включает свойства для хранения имени, номера телефона и времени регистрации клиента, а также метод ToString(), который возвращает строку с форматированной информацией о клиенте.

-zareganble(ref List<Client> clients): Конструктор класса, который принимает список клиентов и инициализирует форму для отображения этих клиентов.

-zareganble\_Load(object sender, EventArgs e): Метод, вызываемый при загрузке формы, который добавляет каждого клиента из списка clients в элемент управления listBoxone на форме. Каждый клиент представлен в виде строки, сгенерированной методом ToString() класса Client.

-button1\_Click(object sender, EventArgs e): Метод, вызываемый при клике на кнопку, который закрывает текущую форму.

-registerButton\_Click(object sender, EventArgs e): Метод, вызываемый при клике на кнопку "Зарегистрировать".

- registerClientButton\_Click(object sender, EventArgs e):этот метод нужен для того чтобы при нажатии кнопки (зарегистрироваться) перекидывало на окно регистрации.

- showRegisteredClientsButton\_Click\_1(object sender, EventArgs e): этот метод нужен для того чтобы при нажатии кнопки (проверить зарегистрированных клиентов) перекидывало на окно зарегистрированных клиентов.

Основной функционал программы будет включать:

- Регистрация клиентов для бронирования номера.

- Просмотр зарегистрированных клиентов.

- Данные о клиентах будет сохраняться после закрытие программы.

## **2.2. План создания и тестирования программы**

Шаги разработки программы:

1.Определение структуры класса Client:

- Создание класса Client для хранения информации о клиентах, включая поля для имени, номера телефона, даты въезда и даты выезда.

2.Написание формы главной страницы (Form2):

- Создана страница для регистрации и просмотра зарегистрированных клиентов.

3.Написание формы регистрации клиента (Form1):

- Создание пользовательского интерфейса для ввода данных о новом клиенте.

Обработка события нажатия кнопки "Зарегистрировать", которая добавляет нового клиента в список clients.

4.Написание формы просмотра зарегистрированных клиентов (zareganble):

- Создание пользовательского интерфейса для отображения списка зарегистрированных клиентов.

- Инициализация списка клиентов clients при загрузке формы zareganble.

Тестирование программы:

1.Тест регистрации клиента:

- Создание нового объекта Form1.

- Регистрация клиента.

- Проверка, что клиент добавлен в список.

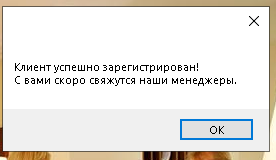
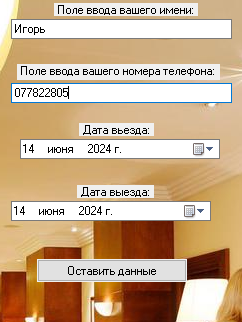


Рис. 1. Регистрация клиента. Рис.2. Добавлен в список.

2. Тест проверка зарегистрированных клиентов:

- Добавление клиента в список приложение.

- Добавление клиента в базу данных.

- Проверка, что клиент есть в списке.

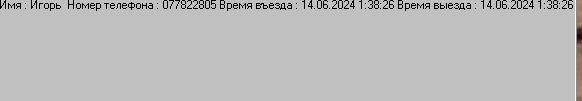


Рис. 3. Список клиентов в приложении.

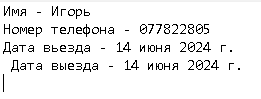


Рис. 4. Список клиентов в base.

Возможное тестирование и оптимизация

Тестирование пользовательского интерфейса является важной частью процесса разработки, позволяя выявить ошибки, недочеты и проблемы в функциональности интерфейса до его внедрения.

Виды тестирования:

Функциональное тестирование - проверка работы всех функций интерфейса.

Usability тестирование - оценка удобства использования интерфейса конечными пользователями.

Тестирование совместимости - проверка корректной работы интерфейса на различных устройствах и в различных браузерах.

Нагрузочное тестирование - проверка стабильности интерфейса при больших нагрузках.

Инструменты для тестирования:

Для тестирования пользовательского интерфейса используются различные инструменты, такие как Selenium, TestComplete, BrowserStack и другие.

Обратная связь пользователей:

Сбор обратной связи от реальных пользователей после тестирования поможет выявить проблемные моменты и предложить улучшения для оптимизации интерфейса.

Оптимизация интерфейса:

На основе результатов тестирования и обратной связи пользователей проводится оптимизация интерфейса для улучшения его удобства, эффективности и привлекательности.

Примеры тестирования и оптимизации:

A/B тестирование - сравнение двух версий интерфейса для определения наиболее эффективной.

Тестирование на различных устройствах - проверка корректной отображаемости интерфейса на мобильных устройствах, планшетах и компьютерах.

Тестирование производительности - оценка скорости работы интерфейса и его отзывчивости на действия пользователя.

## **2.3 Анализ эффективности программы**

Разработка программы для управления информацией о клиентах гостиницы на языке C# может быть эффективным способом хранения и обработки данных. Преимущества такой реализации заключаются в следующем:

Быстрый доступ к данным:

С использованием списка клиентов, программа обеспечивает быстрый доступ к информации о каждом клиенте.

Итерация по данным:

Программа позволяет удобно и быстро просматривать список клиентов, что упрощает управление данными и их анализ.

Гибкость структуры данных:

Программа предоставляет гибкость в изменении и обновлении информации о клиентах. Это полезно для поддержания актуальности данных и быстрого внесения изменений.

Удобство работы с объектами:

Использование объектно-ориентированного подхода в языке C# позволяет эффективно работать с данными о клиентах. Классы и методы четко структурированы, что упрощает их использование и поддержку.

Масштабируемость:

Программа позволяет легко управлять большим объемом данных о клиентах. Она может масштабироваться в зависимости от потребностей, обеспечивая эффективное управление данными даже при увеличении их объема.

Оптимизация и тестирование:

Программа предусматривает проведение различных видов тестирования, включая функциональное, юзабилити, совместимость и нагрузочное тестирование. Это позволяет выявить и устранить возможные ошибки до развертывания программы.

Удобство использования:

Программа имеет дружественный интерфейс, что облегчает работу для конечных пользователей. Сбор обратной связи и проведение A/B тестирования позволяют улучшить интерфейс, делая его еще более удобным и интуитивно понятным.

Обратная связь пользователей:

Сбор и анализ обратной связи от пользователей позволяет выявить слабые места и предложить улучшения, что способствует созданию более эффективного и удобного продукта.

Примеры тестирования и оптимизации:

Тестирование на различных устройствах, проверка производительности и отзывчивости интерфейса помогают убедиться в его корректной работе и эффективности.

## **2.4. Перспективы дальнейшей разработки программы**

Программа для регистрации пользователей в отеле, представленная выше, является базовой версией и предоставляет основные функции для сбора и отображения информации о регистрации. Однако, для повышения её функциональности и удобства использования, можно рассмотреть следующие перспективы дальнейшей разработки:

1. Валидация данных пользователя

Валидация полей ввода: Добавление проверки корректности введённых данных (например, правильный формат номера телефона, обязательное заполнение всех полей).

Проверка дат: Убедиться, что дата заезда не позже даты выезда, а также что даты находятся в будущем.

2. Улучшение интерфейса пользователя (UI/UX)

Использование таблиц и других элементов управления: Замена ListBox на DataGridView для более удобного отображения и редактирования списка регистраций.

Дизайн и удобство использования: Улучшение визуального оформления приложения, добавление всплывающих подсказок, иконок и других элементов для повышения удобства использования.

3. Сохранение данных

База данных: Подключение программы к базе данных (например, SQLite, SQL Server) для хранения информации о пользователях и их регистрациях, что позволит обрабатывать большие объёмы данных и обеспечит быстрый доступ к информации.

4. Обработка и анализ данных

Фильтрация и поиск: Добавление возможности фильтрации и поиска по списку регистраций (например, по имени, дате заезда/выезда).

Статистика: Введение функций для генерации статистики (например, количество регистраций за определённый период, процент заполненности отеля).

Интеграция с внешними системами: Возможность интеграции с системами управления отелем, календарями и другими внешними сервисами.

6. Безопасность

Шифрование данных: Обеспечение безопасности данных пользователей, например, шифрование персональных данных перед сохранением.

Авторизация и аутентификация: Введение системы регистрации и входа для администратора и пользователей с разными уровнями доступа.

7. Многоязычность

Локализация: Поддержка нескольких языков для интерфейса приложения, что сделает программу доступной для пользователей из разных стран.

8. Мобильная версия

Разработка мобильного приложения: Создание мобильной версии программы для платформ Android и iOS, что позволит пользователям регистрироваться в отеле с помощью своих мобильных устройств.

## **2.5. Руководство для программиста**

using System;

namespace WindowsFormsApp1

{

public class Client

{

public string Name { get; set; }

public string Phone { get; set; }

public DateTime CheckInDate { get; set; }

public DateTime CheckOutDate { get; set; } = DateTime.Now;

public string ToString()

{

return $"Имя : {Name}\n Номер телефона : {Phone} \nВремя въезда : {CheckInDate.ToString()}\n Время выезда : {CheckOutDate.ToString()} ";

}

}

}

Выше представлен class Client, а также его:

Поля:

- Name (строка) - имя клиента.

- Phone (строка) – номер клиента.

- DateTimeCheckInDate (строка) – время въезда.

- DateTimeCheckOutDate (строка) – время выезда.

Методы:

- public string ToString() – этот метод конвертирует все данные в строку.

|  |
| --- |
| public void registerClientButton\_Click(object sender, EventArgs e)  {  new Form1(ref clients).ShowDialog();  }  private void showRegisteredClientsButton\_Click\_1(object sender, EventArgs e)  {  new zareganble(ref clients).ShowDialog();  } |

Метод registerClientButton\_Click(object sender, EventArgs e) – этот метод выполняется при нажатии кнопки зарегистрироваться и перекидывает на новую Form1.

Метод showRegisteredClientButton\_Click\_1(object sender, EventArgs e) – этот метод выполняется при нажатии кнопки показать зарегистрированных пользователей и перекидывает на новую форму zareganble.

|  |
| --- |
| using (StreamWriter wr = new StreamWriter(@"D:\приложение курсовой\database\Base.txt",true))  {  wr.WriteLine($"Имя - {nameTextBox.Text}\nНомер телефона - {phoneTextBox.Text}\nДата въезда - {checkInDateTimePicker.Text}\n Дата выезда - {checkOutDateTimePicker.Text}");  } |

Пример использование в коде StreamWriter – записывает данные   
зарегистрированных клиентов и хранит их в себе после закрытие программы.

|  |
| --- |
| if (string.IsNullOrWhiteSpace(name) || string.IsNullOrWhiteSpace(phone))  {  MessageBox.Show("Пожалуйста, введите все данные.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);  return; // Прекращаем выполнение метода, так как не все данные введены  } |

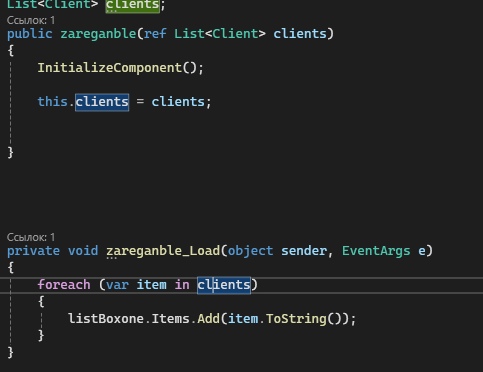
Пример написанного условного оператора if для проверки заполненных всех полей регистрации.

|  |
| --- |
| nameTextBox.Text = "";  phoneTextBox.Text = "";  MessageBox.Show("Клиент успешно зарегистрирован!\nС вами скоро свяжутся наши менеджеры.");  this.Close(); |

Пример очистки полей после регистрации клиента.

|  |
| --- |
| public partial class Form1 : Form  {  List<Client> clients;  public Form1(ref List<Client> clients)  {  InitializeComponent();  this.clients = clients;  }  private void registerButton\_Click(object sender, EventArgs e)  {  using (StreamWriter wr = new StreamWriter(@"D:\приложение курсовой\database\Base.txt",true))  {  wr.WriteLine($"Имя - {nameTextBox.Text}\nНомер телефона - {phoneTextBox.Text}\nДата въезда - {checkInDateTimePicker.Text}\n Дата выезда - {checkOutDateTimePicker.Text}");  }  // Получение данных о клиенте из элементов управления  string name = nameTextBox.Text;  string phone = phoneTextBox.Text;  DateTime checkInDate = checkInDateTimePicker.Value;  DateTime checkOutDate = checkOutDateTimePicker.Value;  // Проверка наличия данных о клиенте  if (string.IsNullOrWhiteSpace(name) || string.IsNullOrWhiteSpace(phone))  {  MessageBox.Show("Пожалуйста, введите все данные.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);  return; // Прекращаем выполнение метода, так как не все данные введены  }  // Создание нового объекта клиента  Client newClient = new Client  {  Name = name,  Phone = phone,  CheckInDate = checkInDate,  CheckOutDate = checkOutDate  };  clients.Add(newClient);  // Очистка полей после регистрации  nameTextBox.Text = "";  phoneTextBox.Text = "";  MessageBox.Show("Клиент успешно зарегистрирован!\nС вами скоро свяжутся наши менеджеры.");  this.Close();  } |

Весь код:  
Рис.5. Пример добавление клиентов в список из List.



## **Заключение**

В ходе выполнения курсового проекта была разработана базовая версия программы для регистрации пользователей в отеле с использованием технологии WinForms на языке программирования C#. Программа позволяет пользователям вводить свои данные и даты заезда/выезда, после чего информация сохраняется и отображается в интерфейсе приложения.

1.Основные задачи, поставленные перед началом разработки, были успешно выполнены:

- Создан удобный и понятный графический интерфейс для ввода данных пользователя.

- Реализована функциональность для сбора и хранения информации о регистрациях.

- Обеспечена возможность просмотра списка зарегистрированных пользователей.

2.Перспективы дальнейшего развития

Разработанная программа является лишь основой для дальнейших улучшений и расширений. Для повышения её функциональности и удобства использования были предложены следующие перспективы развития:

- Валидация данных пользователя: Добавление проверок корректности ввода данных, что позволит избежать ошибок и повысить качество собираемой информации.

- Улучшение интерфейса пользователя (UI/UX): Применение более сложных элементов управления и улучшение визуального оформления для повышения удобства использования программы.

- Сохранение данных: Реализация механизмов для сохранения данных в файлы или базы данных, что обеспечит сохранность информации при закрытии программы.

- Обработка и анализ данных: Введение возможностей фильтрации, поиска и генерации статистики для эффективного управления информацией о регистрациях.

-Уведомления и интеграции: Добавление функций уведомлений пользователей и интеграции с внешними системами для автоматизации и повышения удобства работы.

-Безопасность: Обеспечение безопасности данных пользователей через шифрование и внедрение систем аутентификации и авторизации.

- Многоязычность: Поддержка нескольких языков для расширения аудитории пользователей.

-Мобильная версия: Разработка мобильного приложения, что позволит пользователям регистрироваться в отеле с помощью мобильных устройств.

Реализация данных перспектив позволит значительно расширить функциональные возможности программы, сделав её более мощной и востребованной среди пользователей. Внедрение новых функций и улучшений повысит удобство и эффективность работы с программой, обеспечит безопасность данных пользователей и позволит применять её в реальных условиях гостиничного бизнеса.

## **СПИСОК ЛИТЕРАТУРЫ**

1. Общая информация о объектно-ориентированномпрограммирование на языке С# - <https://metanit.com/sharp/>
2. Примеры, реализации кода, связанных с программированием на C# и использованием различных структур данных - <https://learn.microsoft.com/ru-ru/dotnet/framework/>

3. Общая информация о объектно-ориентированномпрограммирование на языке С# - [habr.com](tyt.lnk)