Дисциплина «Программирование корпоративных систем»
Рабочая тетрадь 3
Классы.

Теоретический материал

C# является полноценным объектно-ориентированным языком. Это значит, что программу на C# можно представить в виде взаимосвязанных взаимодействующих между собой объектов.

Описанием объекта является **класс**, а объект представляет экземпляр этого класса.

По сути класс представляет новый тип, который определяется пользователем. Класс определяется с помощью ключевого слова class:

```
class название_класса
    {
        // содержимое класса
    }
```

Начиная с версии C# 12, если класс имеет пустое определение, то фигурные скобки после названия типа можно не использовать.

Класс может хранить некоторые данные. Для хранения данных в классе применяются **поля**. По сути **поля класса** - это переменные, определенные на уровне класса.

Кроме того, класс может определять некоторое поведение или выполняемые действия. Для определения поведения в классе применяются методы.

```
class Person
{
    public string name = "Undefined";  // имя
    public int age;                    // возраст
    public void Print()
    {
        Console.WriteLine($"Имя: {name}  Возраст: {age}");
    }
}
```

Создание объекта класса

После определения класса мы можем создавать его объекты. Для создания объекта применяются конструкторы.

По сути конструкторы представляют специальные методы, которые называются так же как и класс, и которые вызываются при создании нового объекта класса и выполняют инициализацию объекта.

Общий синтаксис вызова конструктора:

```
new конструктор_класса(параметры_конструктора);
```

```
Person mike = new Person();
```

Для обращения к функциональности класса - полям, методам (а также другим элементам класса) применяется точечная нотация точки - после объекта класса ставится точка, а затем элемент класса:

объект.поле_класса

объект.метод_класса(параметры_метода)

Задание 1

Задача:

Система бронирования

Разработать ПО со следующей архитектурой классов и функционалом:

Класс «Стол»:

Хранимая информация:

- ID стола;
- Расположение стола: (например: у окна, у прохода, у выхода, в глубине);
- Количество мест;
- Расписание занятости стола по часам.

Методы:

- Изменение информации стола;
- Создание стола;
- Вывод информации о столе.

Класс «Бронирование»:

Хранимая информация:

ID клиента;

Имя клиента;

Номер телефона клиента;

Время начала брони;

Время окончания брони;

Комментарий;

Назначенный столик.

Методы:

Создание брони;

Изменение брони;

Отмена брони.

Информация, вносимая в объекты класса «Бронирование», должна вносить изменения в объекты класса «Стол».

Общие требования к функционалу:

- Программный продукт должен позволять создавать набор из n (n>0) столов (каждый стол представляет собой объект класса);
- Программный продукт должен позволять создавать набор из n (n>0) бронирований (каждое бронирование представляет собой объект класса);
- Иметь возможность редактирования информации выбранного стола по его ID (если он не фигурирует в активном бронировании);
- Иметь возможность вывода полной информации о столе по его ID;

****

Пример вывода информации о столе:

ID: ----------01.

Расположение:----------«у окна».

Количество мест: ----------4

Расписание:

9:00-10:00 ----------

10:00-11:00 ----------

11:00-12:00 ----------

12:00-13:00---------- D 3, Макс, 88005553535

13:00-14:00---------- D 3, Макс, 88005553535

14:00-15:00---------- D 3, Макс, 88005553535

15:00-16:00 ----------

16:00-17:00---------- D 7, Анна, 5745552377

17:00-18:00 ----------

****

- Вывод перечня всех доступных для бронирования столов, соответствующих фильтру;
- Вывод перечня всех бронирований;
- Поиск информации о бронировании по 4 последним цифрам номера телефона и имени клиента.

Итоговый проект должен содержать 3 файла классов.

Решение:

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Globalization;

class Program
{
   static List<Table> tables = new List<Table>();
   static List<Reservation> reservations = new List<Reservation>();
   static int nextTableId = 1;
   static int nextReservationId = 1;

   static void Main(string[] args)
   {
      CultureInfo.CurrentCulture = CultureInfo.InvariantCulture;

      InitializeData();
      while (true)
      {
         ShowMenu();
         string choice = Console.ReadLine();
         Console.WriteLine();
         switch (choice)
         {
            case "1": HandleCreateTable(); break;
            case "2": HandleCreateReservation(); break;
            case "3": HandleEditTable(); break;
            case "4": HandleDisplayTableInfo(); break;
            case "5": HandleDisplayAllTables(); break;
            case "6": HandleFindAvailableTables(); break;
            case "7": HandleDisplayAllReservations(); break;
```

```
                case "8": HandleSearchReservation(); break;
                case "9": HandleCancelReservation(); break;
                case "0": return;
                default: Console.WriteLine("Неверный выбор. Пожалуйста,
введите цифру от 0 до 9."); break;
            }
        }
    }

    #region Вспомогательные методы для валидации ввода

    static int ReadInt(string prompt, string errorMessage)
    {
        while (true)
        {
            Console.Write(prompt);
            if (int.TryParse(Console.ReadLine(), out int value))
            {
                return value;
            }
            Console.WriteLine(errorMessage);
        }
    }

    static int ReadPositiveInt(string prompt, string errorMessage)
    {
        while (true)
        {
            int value = ReadInt(prompt, errorMessage);
            if (value > 0)
            {
                return value;
            }
            Console.WriteLine("Ошибка: значение должно быть
положительным числом. Попробуйте снова.");
        }
    }

    static DateTime ReadTime(string prompt, string errorMessage)
    {
        while (true)
        {
            Console.Write(prompt);
            if (DateTime.TryParse(Console.ReadLine(), out DateTime value))
            {
                return value;
            }
            Console.WriteLine(errorMessage);
        }
```

```
    }

    static string ReadNonEmptyString(string prompt, string errorMessage)
    {
        while (true)
        {
            Console.Write(prompt);
            string value = Console.ReadLine();
            if (!string.IsNullOrWhiteSpace(value))
            {
                return value;
            }
            Console.WriteLine(errorMessage);
        }
    }

    static string ReadPhoneNumber(string prompt)
    {
        while (true)
        {
            Console.Write(prompt);
            string input = Console.ReadLine();

            if (string.IsNullOrWhiteSpace(input))
            {
                Console.WriteLine("Ошибка: номер телефона не может быть
пустым.");
                continue;
            }

            var digitsOnly = new string(input.Where(char.IsDigit).ToArray());

            if (digitsOnly.Length < 7)
            {
                Console.WriteLine("Ошибка: номер телефона слишком
короткий. Введите минимум 7 цифр.");
                continue;
            }

            if (digitsOnly.Length > 15)
            {
                Console.WriteLine("Ошибка: номер телефона слишком длинный.
Введите максимум 15 цифр.");
                continue;
            }

            return input;
        }
    }
```

```
    static TableLocation ReadTableLocation(string prompt)
    {
        while (true)
        {
            Console.WriteLine(prompt);
            Console.WriteLine("1. У окна");
            Console.WriteLine("2. У прохода");
            Console.WriteLine("3. У выхода");
            Console.WriteLine("4. В глубине");
            Console.Write("Ваш выбор: ");
            string choice = Console.ReadLine();
            switch (choice)
            {
                case "1": return TableLocation.Window;
                case "2": return TableLocation.Aisle;
                case "3": return TableLocation.Exit;
                case "4": return TableLocation.DeepInside;
                default:
                    Console.WriteLine("Ошибка: неверный выбор. Пожалуйста,
введите цифру от 1 до 4.");
                    break;
            }
        }
    }

    #endregion

    #region Обработчики меню

    static void HandleCreateTable()
    {
        Console.WriteLine("--- Создание нового стола ---");
        int seats = ReadPositiveInt("Введите количество мест: ", "Ошибка:
введите целое положительное число.");
        TableLocation location = ReadTableLocation("Выберите расположение
стола:");
        var newTable = Table.CreateNew(nextTableId++, location, seats);
        tables.Add(newTable);
        Console.WriteLine($"Стол ID {newTable.Id} успешно создан.");
    }

    static void HandleCreateReservation()
    {
        Console.WriteLine("--- Создание нового бронирования ---");
        int clientId = ReadPositiveInt("Введите ID клиента: ", "Ошибка:
введите целое положительное число.");
        string name = ReadNonEmptyString("Введите имя клиента: ",
"Ошибка: имя не может быть пустым.");
```

```
        string phone = ReadPhoneNumber("Введите номер телефона: ");

        DateTime startTime = ReadTime("Введите время начала (например, 14:00): ", "Ошибка: неверный формат времени. Попробуйте снова.");

        if (startTime < DateTime.Now)
        {
            Console.WriteLine("Ошибка: нельзя создать бронирование на время, которое уже прошло.");
            return;
        }

        DateTime endTime;
        while (true)
        {
            endTime = ReadTime("Введите время окончания (например, 16:00): ", "Ошибка: неверный формат времени. Попробуйте снова.");
            if (endTime > startTime) break;
            Console.WriteLine("Ошибка: время окончания должно быть позже времени начала.");
        }

        int guests = ReadPositiveInt("Введите количество гостей: ", "Ошибка: введите целое положительное число.");
        Console.Write("Введите комментарий (необязательно): ");
        string comment = Console.ReadLine();

        var availableTables = tables.Where(t => t.Seats >= guests && t.IsAvailable(startTime, endTime)).ToList();
        if (!availableTables.Any())
        {
            Console.WriteLine("Нет доступных столов на указанное время и количество гостей.");
            return;
        }

        Console.WriteLine("\nДоступные столы:");
        foreach (var table in availableTables)
        {
            string loc = table.Location switch { TableLocation.Window => "у окна", TableLocation.Aisle => "у прохода", TableLocation.Exit => "у выхода", TableLocation.DeepInside => "в глубине", _ => "" };
            Console.WriteLine($"ID: {table.Id}, Мест: {table.Seats}, Расположение: {loc}");
        }

        int chosenTableId = ReadInt("Выберите ID стола для бронирования: ", "Ошибка: введите целое число.");
```

```
        var tableToBook = availableTables.FirstOrDefault(t => t.Id ==
chosenTableId);
        if (tableToBook == null)
        {
            Console.WriteLine("Ошибка: стол с таким ID не найден в списке
доступных.");
            return;
        }

        var newReservation = Reservation.CreateNew(nextReservationId++,
clientId, name, phone, startTime, endTime, comment, tableToBook);
        reservations.Add(newReservation);
        Console.WriteLine($"Бронирование ID {newReservation.Id} успешно
создано для стола ID {tableToBook.Id}.");
    }

    static void HandleEditTable()
    {
        Console.WriteLine("--- Редактирование стола ---");
        int tableId = ReadInt("Введите ID стола для редактирования: ",
"Ошибка: введите целое число.");
        var table = tables.FirstOrDefault(t => t.Id == tableId);
        if (table == null)
        {
            Console.WriteLine("Ошибка: стол с таким ID не найден.");
            return;
        }
        if (table.Schedule.Any(r => r.EndTime > DateTime.Now))
        {
            Console.WriteLine("Нельзя редактировать стол, так как на него
есть активные или будущие бронирования.");
            return;
        }

        int newSeats = ReadPositiveInt($"Введите новое количество мест
(текущее: {table.Seats}): ", "Ошибка: введите целое положительное
число.");
        TableLocation newLocation = ReadTableLocation("Выберите новое
расположение стола:");
        table.ChangeInfo(newLocation, newSeats);
    }

    static void HandleDisplayTableInfo()
    {
        Console.WriteLine("--- Информация о столе ---");
        int tableId = ReadInt("Введите ID стола: ", "Ошибка: введите целое
число.");
        var table = tables.FirstOrDefault(t => t.Id == tableId);
        if (table != null)
```

```
        {
            table.DisplayInfo();
        }
        else
        {
            Console.WriteLine("Ошибка: стол с таким ID не найден.");
        }
    }

    static void HandleDisplayAllTables()
    {
        Console.WriteLine("--- Список всех столов ---");
        if (!tables.Any())
        {
            Console.WriteLine("Столов пока нет.");
            return;
        }
        foreach (var table in tables)
        {
            string loc = table.Location switch { TableLocation.Window => "у
окна", TableLocation.Aisle => "у прохода", TableLocation.Exit => "у
выхода", TableLocation.DeepInside => "в глубине", _ => "" };
            Console.WriteLine($"ID: {table.Id}, Мест: {table.Seats},
Расположение: {loc}");
        }
    }

    static void HandleFindAvailableTables()
    {
        Console.WriteLine("--- Поиск доступных столов ---");
        DateTime startTime = ReadTime("Введите время начала (например,
18:00): ", "Ошибка: неверный формат времени.");

        if (startTime < DateTime.Now)
        {
            Console.WriteLine("Ошибка: нельзя искать столы на время,
которое уже прошло.");
            return;
        }

        DateTime endTime;
        while (true)
        {
            endTime = ReadTime("Введите время окончания (например, 20:00):
", "Ошибка: неверный формат времени.");
            if (endTime > startTime) break;
            Console.WriteLine("Ошибка: время окончания должно быть позже
времени начала.");
        }
```

```
        int guests = ReadPositiveInt("Введите количество гостей: ", "Ошибка: введите целое положительное число.");

        var availableTables = tables.Where(t => t.Seats >= guests && t.IsAvailable(startTime, endTime)).ToList();
        if (!availableTables.Any())
        {
            Console.WriteLine("Нет доступных столов по заданным критериям.");
            return;
        }

        Console.WriteLine("\nДоступные столы:");
        foreach (var table in availableTables)
        {
            string loc = table.Location switch { TableLocation.Window => "у окна", TableLocation.Aisle => "у прохода", TableLocation.Exit => "у выхода", TableLocation.DeepInside => "в глубине", _ => "" };
            Console.WriteLine($"ID: {table.Id}, Мест: {table.Seats}, Расположение: {loc}");
        }
    }

    static void HandleDisplayAllReservations()
    {
        Console.WriteLine("--- Список всех бронирований ---");
        if (!reservations.Any())
        {
            Console.WriteLine("Бронирований пока нет.");
            return;
        }
        foreach (var res in reservations)
        {
            Console.WriteLine($"ID Брони: {res.Id}, Клиент: {res.ClientName}, Телефон: {res.ClientPhone}, Стол: {res.AssignedTable.Id}, Время: {res.StartTime:HH:mm} - {res.EndTime:HH:mm}");
        }
    }

    static void HandleSearchReservation()
    {
        Console.WriteLine("--- Поиск бронирования ---");
        string name = ReadNonEmptyString("Введите имя клиента: ", "Ошибка: имя не может быть пустым.");

        string last4Digits;
        while (true)
        {
            Console.Write("Введите 4 последние цифры номера телефона: ");
```

```
            last4Digits = Console.ReadLine();
            if (!string.IsNullOrWhiteSpace(last4Digits) && last4Digits.Length ==
4 && int.TryParse(last4Digits, out _))
            {
                break;
            }
            Console.WriteLine("Ошибка: необходимо ввести ровно 4 цифры.");
        }

        var foundReservations = reservations
            .Where(r => r.ClientName.Equals(name,
StringComparison.OrdinalIgnoreCase) &&
r.ClientPhone.EndsWith(last4Digits))
            .ToList();

        if (!foundReservations.Any())
        {
            Console.WriteLine("Бронирования с такими данными не
найдены.");
        }
        else
        {
            Console.WriteLine("\nНайденные бронирования:");
            foreach (var res in foundReservations)
            {
                Console.WriteLine($"ID Брони: {res.Id}, Клиент:
{res.ClientName}, Телефон: {res.ClientPhone}, Стол:
{res.AssignedTable.Id}, Время: {res.StartTime:HH:mm} -
{res.EndTime:HH:mm}");
            }
        }
    }

    static void HandleCancelReservation()
    {
        Console.WriteLine("--- Отмена бронирования ---");
        if (!reservations.Any())
        {
            Console.WriteLine("Нет активных бронирований для отмены.");
            return;
        }

        HandleDisplayAllReservations();
        int reservationId = ReadInt("Введите ID бронирования для отмены: ",
"Ошибка: введите целое число.");
        var reservation = reservations.FirstOrDefault(r => r.Id == reservationId);
        if (reservation == null)
        {
            Console.WriteLine("Ошибка: бронирование с таким ID не
```

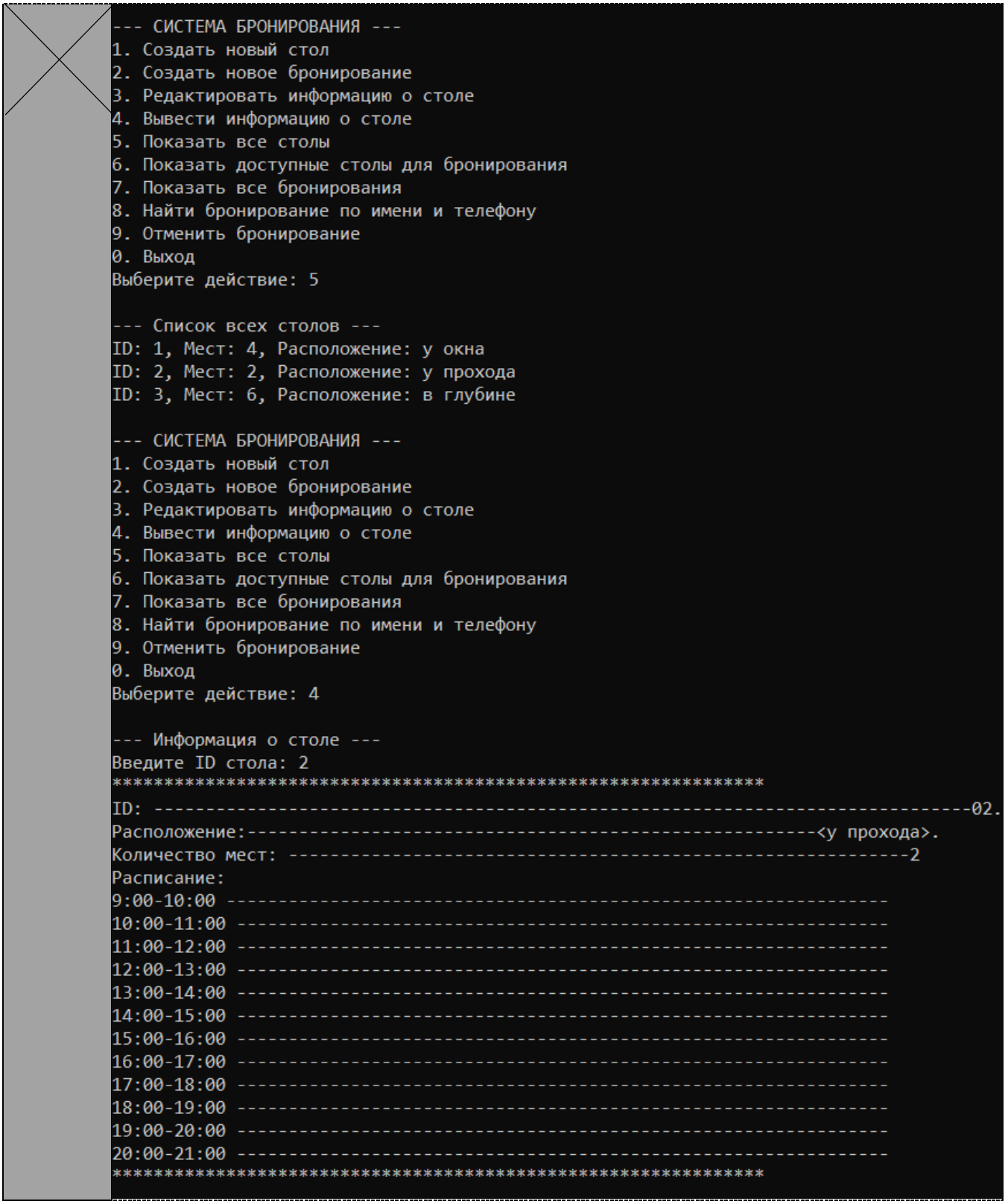
```
            найдено.");
                return;
            }

            reservation.Cancel();
            reservations.Remove(reservation);
        }

        #endregion

        #region Основные методы

        static void ShowMenu()
        {
            Console.WriteLine("\n--- СИСТЕМА БРОНИРОВАНИЯ ---");
            Console.WriteLine("1. Создать новый стол");
            Console.WriteLine("2. Создать новое бронирование");
            Console.WriteLine("3. Редактировать информацию о столе");
            Console.WriteLine("4. Вывести информацию о столе");
            Console.WriteLine("5. Показать все столы");
            Console.WriteLine("6. Показать доступные столы для
        бронирования");
            Console.WriteLine("7. Показать все бронирования");
            Console.WriteLine("8. Найти бронирование по имени и телефону");
            Console.WriteLine("9. Отменить бронирование");
            Console.WriteLine("0. Выход");
            Console.Write("Выберите действие: ");
        }

        static void InitializeData()
        {
            tables.Add(Table.CreateNew(nextTableId++, TableLocation.Window,
        4));
            tables.Add(Table.CreateNew(nextTableId++, TableLocation.Aisle, 2));
            tables.Add(Table.CreateNew(nextTableId++, TableLocation.DeepInside,
        6));
        }

        #endregion
}

Table.cs

  using System;
using System.Collections.Generic;
using System.Linq;

public enum TableLocation
{
```

```
    Window,
    Aisle,
    Exit,
    DeepInside
}

public class Table
{
    public int Id { get; private set; }
    public TableLocation Location { get; private set; }
    public int Seats { get; private set; }
    public List<Reservation> Schedule { get; private set; }

    private Table(int id, TableLocation location, int seats)
    {
        Id = id;
        Location = location;
        Seats = seats;
        Schedule = new List<Reservation>();
    }

    public static Table CreateNew(int id, TableLocation location, int seats)
    {
        return new Table(id, location, seats);
    }

    public void ChangeInfo(TableLocation newLocation, int newSeats)
    {
        Location = newLocation;
        Seats = newSeats;
        Console.WriteLine($"Информация о столе ID {Id} успешно обновлена.");
    }
    public void DisplayInfo()
    {
        string locationStr = Location switch
        {
            TableLocation.Window => "у окна",
            TableLocation.Aisle => "у прохода",
            TableLocation.Exit => "у выхода",
            TableLocation.DeepInside => "в глубине",
            _ => "неизвестно"
        };


Console.WriteLine("**********************************************************
**********");
        Console.WriteLine($"ID: ----------------------------------------------------------
----------{Id:D2}.");
        Console.WriteLine($"Расположение:-----------------------------------------------
```

```
----«{locationStr}».");
        Console.WriteLine($"Количество мест: ----------------------------------------
------------{Seats}");
        Console.WriteLine("Расписание:");

        for (int hour = 9; hour < 21; hour++)
        {
            var slotStart = new DateTime(DateTime.Today.Year,
DateTime.Today.Month, DateTime.Today.Day, hour, 0, 0);
            var slotEnd = slotStart.AddHours(1);

            var activeReservation = Schedule.FirstOrDefault(r => r.StartTime < slotEnd
&& r.EndTime > slotStart);

            string line = $"{hour}:00-{hour + 1}:00 ";
            if (activeReservation != null)
            {
                line += $"-------------------ID {activeReservation.Id},
{activeReservation.ClientName}, {activeReservation.ClientPhone}";
            }
            Console.WriteLine(line.PadRight(75, '-'));
        }

Console.WriteLine("**********************************************************
***********");
    }
    public bool IsAvailable(DateTime startTime, DateTime endTime)
    {
        return !Schedule.Any(r => startTime < r.EndTime && endTime > r.StartTime);
    }
}

Reservation.cs

            using System;

            public class Reservation
            {
                public int Id { get; private set; }
                public int ClientId { get; private set; }
                public string ClientName { get; private set; }
                public string ClientPhone { get; private set; }
                public DateTime StartTime { get; private set; }
                public DateTime EndTime { get; private set; }
                public string Comment { get; private set; }
                public Table AssignedTable { get; private set; }
                private Reservation(int id, int clientId, string clientName, string
            clientPhone, DateTime startTime, DateTime endTime, string comment, Table
            assignedTable)
```

```
        {
            Id = id;
            ClientId = clientId;
            ClientName = clientName;
            ClientPhone = clientPhone;
            StartTime = startTime;
            EndTime = endTime;
            Comment = comment;
            AssignedTable = assignedTable;
        }

        public static Reservation CreateNew(int id, int clientId, string clientName,
    string clientPhone, DateTime startTime, DateTime endTime, string comment,
    Table assignedTable)
        {
            var reservation = new Reservation(id, clientId, clientName, clientPhone,
    startTime, endTime, comment, assignedTable);
            assignedTable.Schedule.Add(reservation);
            return reservation;
        }

        public void ChangeDetails(DateTime newStartTime, DateTime
    newEndTime, string newComment)
        {
            AssignedTable.Schedule.Remove(this);
            StartTime = newStartTime;
            EndTime = newEndTime;
            Comment = newComment;

            AssignedTable.Schedule.Add(this);

            Console.WriteLine($"Бронирование ID {Id} успешно изменено.");
        }

        public void Cancel()
        {
            AssignedTable.Schedule.Remove(this);
            Console.WriteLine($"Бронирование ID {Id} для стола
    {AssignedTable.Id} отменено.");
        }
}
```

Ответ:


--- СИСТЕМА БРОНИРОВАНИЯ ---
1. Создать новый стол
2. Создать новое бронирование
3. Редактировать информацию о столе
4. Вывести информацию о столе
5. Показать все столы
6. Показать доступные столы для бронирования
7. Показать все бронирования
8. Найти бронирование по имени и телефону
9. Отменить бронирование
0. Выход
Выберите действие: 5
--- Список всех столов ---
ID: 1, Мест: 4, Расположение: у окна
ID: 2, Мест: 2, Расположение: у прохода
ID: 3, Мест: 6, Расположение: в глубине
--- СИСТЕМА БРОНИРОВАНИЯ ---
1. Создать новый стол
2. Создать новое бронирование
3. Редактировать информацию о столе
4. Вывести информацию о столе
5. Показать все столы
6. Показать доступные столы для бронирования
7. Показать все бронирования
8. Найти бронирование по имени и телефону
9. Отменить бронирование
0. Выход
Выберите действие: 4
--- Информация о столе ---
Введите ID стола: 2
****
ID: ----------02.
Расположение:----------<у прохода>.
Количество мест: ----------2
Расписание:
9:00-10:00 ----------
10:00-11:00 ----------
11:00-12:00 ----------
12:00-13:00 ----------
13:00-14:00 ----------
14:00-15:00 ----------
15:00-16:00 ----------
16:00-17:00 ----------
17:00-18:00 ----------
18:00-19:00 ----------
19:00-20:00 ----------
20:00-21:00 ----------
****