Дисциплина «Программирование корпоративных систем» Рабочая тетрадь 3.2

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 сlass:  **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»Ра*  *.*  *Количество мест: ------------------------------------------------------------4*  *Расписание:*  *9:00-10:00 -----------------------------------------------------*  *10:00-11:00 ---------------------------------------------------*  *11:00-12:00 ---------------------------------------------------*  *12:00-13:00 -------------------ID 3, Макс, 88005553535*  *13:00-14:00 -------------------ID 3, Макс, 88005553535*  *14:00-15:00 ------------------ ID 3, Макс, 88005553535*  *15:00-16:00 --------------------------------------------------*  *16:00-17:00 ----------------- ID 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:*** | |
|  | |  | | --- | | using System; using System.Collections.Generic; using System.Linq;  namespace RT\_3\_2\_Restoran {  class Program  {  private static List<Table> *\_tables* = new();  private static List<Reservation> *\_reservations* = new();   static void *Main*(string[] args)  {  while (true)  {  Console.*WriteLine*("\n--- Меню ---");  Console.*WriteLine*("1. Добавить стол");  Console.*WriteLine*("2. Просмотреть все столы");  Console.*WriteLine*("3. Добавить бронь");  Console.*WriteLine*("4. Просмотреть все брони");  Console.*WriteLine*("5. Отменить бронь");  Console.*WriteLine*("6. Поиск брони по имени клиента");  Console.*WriteLine*("7. Поиск брони по последним 4 цифрам телефона");  Console.*WriteLine*("8. Обновить информацию о столе");  Console.*WriteLine*("9. Показать доступные для бронирования столы по фильтру");  Console.*WriteLine*("0. Выход");  Console.*Write*("Выберите опцию: ");  var input = Console.*ReadLine*();   switch (input)  {  case "1":  *AddTable*();  break;  case "2":  *DisplayTables*();  break;  case "3":  *AddReservation*();  break;  case "4":  *DisplayReservations*();  break;  case "5":  *CancelReservation*();  break;  case "6":  *SearchReservationByName*();  break;  case "7":  *SearchReservationByPhone*();  break;  case "8":  *UpdateTableIfNoReservation*();  break;  case "9":  *DisplayAvailableTables*();  break;  case "0":  return;  default:  Console.*WriteLine*("Неверный ввод. Попробуйте снова.");  break;  }  }  }   static void *AddTable*()  {  Console.*Write*("Введите расположение стола: ");  string location = Console.*ReadLine*();  Console.*Write*("Введите количество мест: ");  int seatCount = int.*Parse*(Console.*ReadLine*() ?? "0");   var table = new Table(location, seatCount);  *\_tables*.Add(table);  Console.*WriteLine*($"Стол добавлен: {table}");  }   static void *DisplayTables*()  {  if (*\_tables*.Count == 0)  {  Console.*WriteLine*("\nСтолы не созданы");  return;  }  Console.*WriteLine*("\nСписок столов:");  foreach (var table in *\_tables*)  {  Console.*WriteLine*(*FormatTableInfo*(table));  }  }   static string *FormatTableInfo*(Table table)  {  var result = $"\nID: {table.Id:D2}.\n" +  $"Расположение:«{table.Location}».\n" +  $"Количество мест: {table.SeatCount}\n" +  $"Расписание:";   foreach (var hour in table.Availability)  {  var reservation = *\_reservations* .FirstOrDefault(r => r.AssignedTable == table &&   hour.Hour >= r.StartTime.Hour && hour.Hour < r.EndTime.Hour);   if (reservation != null)  {  result += $"\n\t{hour.Hour:00}:00-{hour.Hour + 1:00}:00 ----" +  $"ID {reservation.ClientId}, {reservation.ClientName}, {reservation.PhoneNumber}";  }  else  {  result += $"\n\t{hour.Hour:00}-{hour.Hour + 1:00} -----------------------------------------------------";  }  }   return result;  }   static void *AddReservation*()  {  Console.*Write*("Введите имя клиента: ");  string clientName = Console.*ReadLine*();  Console.*Write*("Введите телефон клиента: ");  string phoneNumber = Console.*ReadLine*();  Console.*Write*("Введите время начала брони (часы 0-23): ");  int startHour = int.*Parse*(Console.*ReadLine*() ?? "0");  Console.*Write*("Введите время окончания брони (часы 0-23): ");  int endHour = int.*Parse*(Console.*ReadLine*() ?? "0");  Console.*Write*("Введите комментарий: ");  string comment = Console.*ReadLine*();   Console.*WriteLine*("Доступные столы:");  *DisplayTables*();  Console.*Write*("Выберите ID стола: ");  int tableId = int.*Parse*(Console.*ReadLine*() ?? "0");   var table = *\_tables*.Find(t => t.Id == tableId);  if (table == null)  {  Console.*WriteLine*("Стол не найден.");  return;  }   try  {  var reservation = new Reservation(  clientName, phoneNumber,   new DateTime(1, 1, 1, startHour, 0, 0),  new DateTime(1, 1, 1, endHour, 0, 0),  comment, table  );   *\_reservations*.Add(reservation);  Console.*WriteLine*("Бронь успешно добавлена.");  }  catch (Exception ex)  {  Console.*WriteLine*($"Ошибка при создании брони: {ex.Message}");  }  }   static void *DisplayReservations*()  {  if (*\_reservations*.Count == 0)  {  Console.*WriteLine*("\nНи одной брони не создано.");  return;  }  Console.*WriteLine*("\nСписок бронирований:");  foreach (var reservation in *\_reservations*)  {  Console.*WriteLine*(reservation);  Console.*WriteLine*("========================");  }  }   static void *CancelReservation*()  {  Console.*Write*("Введите ID клиента для отмены брони: ");  Guid clientId = Guid.*Parse*(Console.*ReadLine*() ?? string.*Empty*);   var reservation = *\_reservations*.Find(r => r.ClientId == clientId);  if (reservation == null)  {  Console.*WriteLine*("Бронь не найдена.");  return;  }   reservation.CancelReservation();  *\_reservations*.Remove(reservation);  Console.*WriteLine*("Бронь успешно отменена.");  }   static void *SearchReservationByName*()  {  Console.*Write*("Введите имя клиента для поиска: ");  string name = Console.*ReadLine*();   var results = *\_reservations*.Where(r => r.ClientName.Equals(name, StringComparison.*OrdinalIgnoreCase*)).ToList();   if (results.Count == 0)  {  Console.*WriteLine*("Брони с таким именем не найдены.");  return;  }   Console.*WriteLine*("Найденные брони:");  foreach (var reservation in results)  {  Console.*WriteLine*(reservation);  Console.*WriteLine*("========================");  }  }   static void *SearchReservationByPhone*()  {  Console.*Write*("Введите последние 4 цифры номера телефона: ");  string last4Digits = Console.*ReadLine*();   var results = *\_reservations*.Where(r => r.PhoneNumber.EndsWith(last4Digits)).ToList();   if (results.Count == 0)  {  Console.*WriteLine*("Брони с такими цифрами телефона не найдены.");  return;  }   Console.*WriteLine*("Найденные брони:");  foreach (var reservation in results)  {  Console.*WriteLine*(reservation);  Console.*WriteLine*("========================");  }  }    // Метод обновления информации о столе при отсутствии активных броней  static void *UpdateTableIfNoReservation*()  {  Console.*Write*("Введите ID стола для обновления: ");  int tableId = int.*Parse*(Console.*ReadLine*() ?? "0");   var table = *\_tables*.Find(t => t.Id == tableId);  if (table == null)  {  Console.*WriteLine*("Стол не найден.");  return;  }   bool hasReservation = *\_reservations*.Any(r => r.AssignedTable == table);  if (hasReservation)  {  Console.*WriteLine*("Стол не может быть обновлен, так как на нем есть активные брони.");  return;  }   Console.*Write*("Введите новое расположение стола: ");  string newLocation = Console.*ReadLine*();  Console.*Write*("Введите новое количество мест: ");  int newSeatCount = int.*Parse*(Console.*ReadLine*() ?? "0");   table.UpdateTable(newLocation, newSeatCount);  Console.*WriteLine*("Информация о столе успешно обновлена.");  }    // Метод для вывода доступных столов по фильтру  static void *DisplayAvailableTables*()  {  Console.*Write*("Введите минимальное количество мест: ");  int minSeats = int.*Parse*(Console.*ReadLine*() ?? "0");  Console.*Write*("Введите начальный час брони (0-23): ");  int startHour = int.*Parse*(Console.*ReadLine*() ?? "0");  Console.*Write*("Введите конечный час брони (0-23): ");  int endHour = int.*Parse*(Console.*ReadLine*() ?? "0");   var availableTables = *\_tables*.Where(table =>  table.SeatCount >= minSeats &&  *IsTableAvailable*(table, startHour, endHour)).ToList();   if (availableTables.Count == 0)  {  Console.*WriteLine*("Нет доступных столов, соответствующих заданным критериям.");  }  else  {  Console.*WriteLine*("\nДоступные для бронирования столы:");  foreach (var table in availableTables)  {  Console.*WriteLine*(*FormatTableInfo*(table));  }  }  }   // Метод для проверки доступности стола в заданный диапазон времени  static bool *IsTableAvailable*(Table table, int startHour, int endHour)  {  return table.Availability  .Where(schedule => schedule.Hour >= startHour && schedule.Hour < endHour)  .All(schedule => !schedule.IsOccupied);  }  } } | | namespace RT\_3\_2\_Restoran;  public struct Schedule {  public int Hour { get; set; } // Час (0-23)  public bool IsOccupied { get; set; } // Занято ли место    public Schedule(int hour, bool isOccupied)  {  if (hour < 0 || hour > 23)  throw new ArgumentOutOfRangeException(nameof(hour), "Час должен быть в диапазоне от 0 до 23.");   Hour = hour;  IsOccupied = isOccupied;  }        public override string ToString()  {  return $"Час: {Hour}, Занято: {(IsOccupied ? "Да" : "Нет")}";  } } | | using System; using System.Collections.Generic;   namespace RT\_3\_2\_Restoran {  public class Table  {  private int \_id;  private string \_location;  private int \_seatCount;  private List<Schedule> \_availability;   private static int *\_nextId* = 1; // Статический счетчик для автоинкремента   public int Id => \_id; // Только для чтения  public string Location  {  get => \_location;  set => \_location = value;  }  public int SeatCount  {  get => \_seatCount;  set => \_seatCount = value;  }  public List<Schedule> Availability => \_availability; // Только для чтения списка    */// <summary>  /// Создание стола  /// </summary>  /// <param name="location">Расположение стола</param>  /// <param name="seatCount">Количество посадочных мест</param>* public Table(string location, int seatCount)  {  this.\_id = *\_nextId*++; // нужен ли this?  \_location = location;  \_seatCount = seatCount;   \_availability = new List<Schedule>(24);  for (int i = 0; i < 24; i++)  {  \_availability.Add(new Schedule(i, false)); // Инициализация часов как свободных  }  }   */// <summary>  /// Изменение информации стола  /// </summary>  /// <param name="location">Расположение стола</param>  /// <param name="seatCount">Количество посадочных мест</param>* public void UpdateTable(string location, int seatCount)  {  \_location = location;  \_seatCount = seatCount;  }    public override string ToString()  {  string scheduleInfo = string.*Join*(", ", \_availability);  return $"ID: {\_id}, Расположение: {\_location}, Мест: {\_seatCount}, Расписание: [{scheduleInfo}]";  }   */// <summary>  /// Добавление брони  /// </summary>  /// <param name="hour">Час, который необходимо забронировать</param>  /// <returns>true - забронирован; false - не забронирован</returns>  /// <exception cref="ArgumentOutOfRangeException">Неверное значение часа</exception>* public bool AddReservation(int hour)  {  if (hour < 0 || hour > 23)  throw new ArgumentOutOfRangeException(nameof(hour), "Час должен быть в диапазоне от 0 до 23.");   if (\_availability[hour].IsOccupied)  {  Console.*WriteLine*($"Стол уже занят в {hour}:00.");  return false;  }   \_availability[hour] = new Schedule(hour, true); // Устанавливаем занятость  return true;  }  } } | | using System;  namespace RT\_3\_2\_Restoran {  public class Reservation  {  private Guid \_clientId;  private string \_clientName;  private string \_phoneNumber;  private DateTime \_startTime;  private DateTime \_endTime;  private string \_comment;  private Table \_assignedTable;   public Guid ClientId => \_clientId; // Только для чтения  public string ClientName  {  get => \_clientName;  set => \_clientName = value;  }  public string PhoneNumber  {  get => \_phoneNumber;  set => \_phoneNumber = value;  }  public DateTime StartTime  {  get => \_startTime;  set => \_startTime = value;  }  public DateTime EndTime  {  get => \_endTime;  set => \_endTime = value;  }  public string Comment  {  get => \_comment;  set => \_comment = value;  }  public Table AssignedTable => \_assignedTable; // Только для чтения   */// <summary>  /// Создание брони  /// </summary>  /// <param name="clientName">Имя клиента</param>  /// <param name="phoneNumber">Номер телефона клиента</param>  /// <param name="startTime">Время начала брони</param>  /// <param name="endTime">Время окончания брони</param>  /// <param name="comment">Комментарий</param>  /// <param name="assignedTable">Стол</param>* public Reservation(string clientName, string phoneNumber, DateTime startTime,   DateTime endTime, string comment, Table assignedTable)  {  \_clientId = Guid.*NewGuid*();  \_clientName = clientName;  \_phoneNumber = phoneNumber;  \_startTime = startTime;  \_endTime = endTime;  \_comment = comment;  \_assignedTable = assignedTable;   AssignTable();  }   */// <summary>  /// Назначает столик  /// </summary>* private void AssignTable()  {  for (int hour = \_startTime.Hour; hour < \_endTime.Hour; hour++)  {  if (!\_assignedTable.AddReservation(hour))  throw new InvalidOperationException($"Столик занят в {hour}:00.");  }  }   */// <summary>  /// Изменение брони  /// </summary>* public void UpdateReservation(DateTime newStartTime, DateTime newEndTime, string newComment)  {  CancelReservation();  \_startTime = newStartTime;  \_endTime = newEndTime;  \_comment = newComment;  AssignTable();  }   */// <summary>  /// Отмена брони  /// </summary>* public void CancelReservation()  {  for (int hour = \_startTime.Hour; hour < \_endTime.Hour; hour++)  {  \_assignedTable.Availability[hour] = new Schedule(hour, false);  }  }   public override string ToString()  {  return $"ID клиента: {\_clientId}\nИмя клиента: {\_clientName}, Телефон: {\_phoneNumber}\nВремя: {\_startTime:HH:mm} - {\_endTime:HH:mm}\nСтолик: {\_assignedTable.Id}, Комментарий: {\_comment}";  }   } } | |
| ***Ответ:*** | |
|  | Вывод столов, если они не созданы    Вывод броней, если они не созданы    Добавление стола    Отображение информации о столе на котором нет броней  Добавление брони    Повторный вывод информации о столе    Просмотр броней    Отмена брони      Выход |