Дисциплина «Программирование корпоративных систем» Рабочая тетрадь 4

ПО "Система заказов"

|  |
| --- |
| **Теоретический материал** |
| Метод может возвращать значение, какой-либо результат для этого применяется оператор **return**, после которого идет возвращаемое значение:  **return возвращаемое значение;**  Методы с типом **void** не возвращают никакого значения. Они просто выполняют некоторые действия.  Наиболее простой способ передачи параметров представляет передача по значению, по сути это обычный способ передачи параметров:  **void Increment(int n)**  **{**  **n++;**  **Console.WriteLine($"Число в методе Increment: {n}");**  **}**  При передаче аргументов параметрам по значению параметр метода получает не саму переменную, а ее копию и далее работает с этой копией независимо от самой переменной.  **Increment(number);**  При передаче параметров по ссылке перед параметрами используется модификатор **ref**:  **void Increment(ref int n)**  **{**  **n++;**  **Console.WriteLine($"Число в методе Increment: {n}");**  **}**  При передаче значений параметрам по ссылке метод получает адрес переменной в памяти. И, таким образом, если в методе изменяется значение параметра, передаваемого по ссылке, то также изменяется и значение переменной.  Обратите внимание, что модификатор ref указывается как перед параметром при объявлении метода, так и при вызове метода перед аргументом, который передается параметру.  **Increment(ref number);**  Параметры могут быть также выходными. Чтобы сделать параметр выходным, перед ним ставится модификатор **out**:  **void Sum(int x, int y, out int result)**  **{**  **result = x + y;**  **}**  Причем, как и в случае с **ref** ключевое слово **out** используется как при определении метода, так и при его вызове.  **Sum(10, 15, out number);**  Кроме выходных параметров с модификатором **out** метод может использовать входные параметры с модификатором **in**.  Модификатор **in** указывает, что данный параметр будет передаваться в метод по ссылке, однако внутри метода его значение параметра нельзя будет изменить. Например, возьмем следующий метод:  **void GetRectangleData(in int width, in int height, out int rectArea, out int rectPerimetr)**  **{**  **//width = 25; // нельзя изменить, так как width - входной параметр**  **rectArea = width \* height;**  **rectPerimetr = (width + height) \* 2;**  **}**  Вызов:  **GetRectangleData(w, h, out var area, out var perimetr); // in w, in h**  В примерах выше можно было изменять значение **ref**-параметра. Однако иногда это может быть нежелательно. И чтобы гарантировать, что **ref**-параметр не изменит своего значения, начиная с версии C# 12 можно применять **ref**-параметры только для чтения. Такие параметры предваряются ключевым словом **readonly**:  **void Increment(ref readonly int n)**  **{**  **// n++; // нельзя, иначе будет ошибка компиляции**  **Console.WriteLine($"Число в методе Increment: {n}");**  **}**  Вызов:  **Increment(ref number);**  Во всех предыдущих примерах мы использовали постоянное число параметров. Но, используя ключевое слово **params**, мы можем передавать неопределенное количество параметров:  **void Sum(params int[]  numbers)**  **{**  **int result = 0;**  **foreach (var n in numbers)**  **{**  **result += n;**  **}**  **Console.WriteLine(result);**  **}**  Сам параметр с ключевым словом **params** при определении метода должен представлять одномерный массив того типа, данные которого мы собираемся использовать.  При вызове метода на место параметра с модификатором params мы можем передать как отдельные значения, так и массив значений, либо вообще не передавать параметры. Количество передаваемых значений в метод неопределённо, однако все эти значения должны соответствовать типу параметра с **params**.  Вызов:  **int[] nums = { 1, 2, 3, 4, 5};**  **Sum(nums);**  **Sum(1, 2, 3, 4);**  **Sum(1, 2, 3);**  **Sum();** |

|  |  |
| --- | --- |
|  | |
| **Задание 1** | |
| ***Задача:*** | |
|  | **Система заказов**  Разработать ПО со следующей архитектурой классов и функционалом:  **Класс «Блюдо»:**  Хранимая информация:   * + - Id блюда (целое число);     - Название (тип строка);     - Состав (тип строка);     - Вес (строка формата (100/20/50));     - Цена (вещественный тип);     - Категория (Тип перечисление (напитки, салаты, холодные закуски, горячие закуски, супы, горячие блюда, десерт и т.д ));     - Время готовки (целое число);     - Тип (массив строк (острое, веганское, халяль, кошерное и т. д.).   Методы:   * + - Создание блюда;     - Редактирование блюда;     - Вывод информации о блюде;     - Удаление блюда.   **Класс «Заказ»:**  Хранимая информация:   * + - Id заказа (целое число);     - Id стола (целое число);     - Массив блюд (объекты класса, учесть возможность дублирования блюда);     - Комментарий (строка);     - Время принятия заказа (тип время или строка);     - Официант (целое число);     - Время закрытия заказа (тип время или строка);     - Итоговая стоимость (Вещественное число).   Методы:   * + - Создание заказа;     - Изменение заказа;     - Вывод информации о заказе;     - Закрытие заказа;     - Вывод чека (только для закрытых заказов).   \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Столик:  Официант:  Период обслуживания: с …. по….  Категория блюда\_1:  Название блюда\_1 кол-во\*цена=итог цена  Название блюда\_2 кол-во\*цена=итог цена  Название блюда\_3 кол-во\*цена=итог цена  Под\_итог категории  …  Категория блюда\_n:  Название блюда\_1 кол-во\*цена=итог цена  Название блюда\_2 кол-во\*цена=итог цена  Название блюда\_3 кол-во\*цена=итог цена  Под\_итог категории  Итог счета: итог цена  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **Общие требования к функционалу:**   * Программный продукт должен позволять создавать набор из n (n>0) блюд (каждое блюдо представляет собой объект класса); * Программный продукт должен позволять создавать набор из n (n>0) заказов (каждый заказ представляет собой объект класса); * Метод вывода меню (блюда должны быть распределены по категориям и результат работы метода должен содержать значимую для клиента информацию, в дальнейшем планируется печать меню); * Метод подсчета стоимости всех закрытых заказов на текущий момент; * Метод подсчета закрытых заказов конкретного официанта на текущий момент; * Метод сбора статистики по количеству заказанных блюд.   Итоговый проект должен содержать 3 файла классов.  Использовать возможности: in, out, ref, params. |
| ***Решение:*** | |
|  | Program.cs  using System;  using System.Collections.Generic;  using System.Linq;  class Program  {  private static List<Dish> \_dishes = new List<Dish>();  private static List<Order> \_orders = new List<Order>();  static void Main(string[] args)  {  bool isRunning = true;  while (isRunning)  {  Console.WriteLine("Добро пожаловать в меню программы!");  Console.WriteLine("Выберите что хотите сделать:");  Console.WriteLine("1. Действия с блюдами");  Console.WriteLine("2. Действия с заказами");  Console.WriteLine("3. Меню ресторана");  Console.WriteLine("4. Полная стоимость закрытых заказов");  Console.WriteLine("5. Стоимость закрытых заказов определенного официанта ");  Console.WriteLine("6. Статистика заказанных блюд");  Console.WriteLine("7. Выход");  int choice = GetUserChoice();  switch (choice)  {  case 1:  ManageDishes();  break;  case 2:  ManageOrders();  break;  case 3:  PrintMenu();  break;  case 4:  Console.WriteLine($"Полная стоимость всех закрытых заказов: {CalculateTotalClosedOrders()}");  break;  case 5:  Console.Write("Введите ID официанта: ");  int waiterId = int.Parse(Console.ReadLine());  PrintWaiterOrdersInfo(waiterId);  break;  case 6:  PrintDishOrderStatistics();  break;  case 7:  isRunning = false;  Console.WriteLine("Выход из приложения...");  break;  default:  Console.WriteLine("Неправильный ввод, попробуйте еще раз.");  break;  }  Console.WriteLine();  }  }  static void ManageDishes()  {  bool isDishManagementRunning = true;  while (isDishManagementRunning)  {  Console.WriteLine("Меню выбора действия с блюдами:");  Console.WriteLine("1. Создать блюдо");  Console.WriteLine("2. Изменить блюдо");  Console.WriteLine("3. Удалить блюдо");  Console.WriteLine("4. Вернуться в главное меню");  int choice = GetUserChoice();  switch (choice)  {  case 1:  CreateDish();  break;  case 2:  EditDish();  break;  case 3:  DeleteDish();  break;  case 4:  isDishManagementRunning = false;  break;  default:  Console.WriteLine("Неправильный ввод, попробуйте еще раз.");  break;  }  Console.WriteLine();  }  }  static void ManageOrders()  {  bool isOrderManagementRunning = true;  while (isOrderManagementRunning)  {  Console.WriteLine("Меню выбора действий с заказами:");  Console.WriteLine("1. Создание заказа");  Console.WriteLine("2. Изменение заказа");  Console.WriteLine("3. Закрытие заказа");  Console.WriteLine("4. Вывод чека");  Console.WriteLine("5. Вернуться в главное меню.");  int choice = GetUserChoice();  switch (choice)  {  case 1:  CreateOrder();  break;  case 2:  EditOrder();  break;  case 3:  CloseOrder();  break;  case 4:  PrintReceipt();  break;  case 5:  isOrderManagementRunning = false;  break;  default:  Console.WriteLine("Неправильный ввод, попробуйте еще раз.");  break;  }  Console.WriteLine();  }  }  static void PrintMenu()  {  Console.WriteLine("Меню:");  var dishByCategory = \_dishes.GroupBy(d => d.Category)  .ToDictionary(g => g.Key, g => g.ToList());  foreach (var kvp in dishByCategory)  {  Console.WriteLine($"{kvp.Key}:");  foreach (var dish in kvp.Value)  {  dish.PrintDishInfo();  Console.WriteLine();  }  }  }  static double CalculateTotalClosedOrders()  {  return \_orders.Where(o => o.CloseTime != DateTime.MinValue)  .Sum(o => o.TotalCost);  }  static void PrintWaiterOrdersInfo(int waiterId)  {  var waiterOrders = \_orders  .Where(o => o.CloseTime != DateTime.MinValue && o.WaiterId == waiterId)  .OrderBy(o => o.Id)  .ToList();  if (!waiterOrders.Any())  {  Console.WriteLine($"Официант с ID {waiterId} не имеет закрытых заказов.");  return;  }  Console.WriteLine($"\nЗакрытые заказы официанта с ID {waiterId}:");  Console.WriteLine("==========================================");  foreach (var order in waiterOrders)  {  Console.WriteLine($"Заказ с ID {order.Id} на сумму {order.TotalCost:F2}");  }  double totalSum = waiterOrders.Sum(o => o.TotalCost);  Console.WriteLine("------------------------------------------");  Console.WriteLine($"Общая сумма заказов: {totalSum:F2}");  Console.WriteLine($"Количество закрытых заказов: {waiterOrders.Count}");  Console.WriteLine("==========================================\n");  }  static void PrintDishOrderStatistics()  {  var dishOrders = CollectDishOrderStatistics();  Console.WriteLine("\n=== Статистика заказанных блюд ===");  Console.WriteLine("----------------------------------");  var groupedStats = dishOrders  .GroupBy(kvp => kvp.Key.Category)  .OrderBy(g => g.Key);  foreach (var categoryGroup in groupedStats)  {  Console.WriteLine($"\nКатегория: {categoryGroup.Key}");  Console.WriteLine("----------------------------------");  foreach (var dishStat in categoryGroup.OrderByDescending(d => d.Value))  {  Console.WriteLine($"{dishStat.Key.Name,-30} | {dishStat.Value,5} заказов | На сумму: {dishStat.Value \* dishStat.Key.Price:C}");  }  }  int totalOrders = dishOrders.Sum(x => x.Value);  double totalRevenue = dishOrders.Sum(x => x.Value \* x.Key.Price);  Console.WriteLine("\n=== Общая статистика ===");  Console.WriteLine($"Всего заказано блюд: {totalOrders}");  Console.WriteLine($"Общая сумма: {totalRevenue:C}");  Console.WriteLine("=======================\n");  }  static Dictionary<Dish, int> CollectDishOrderStatistics()  {  var dishOrders = new Dictionary<Dish, int>();  foreach (var order in \_orders)  {  foreach (var dish in order.Dishes)  {  if (dishOrders.ContainsKey(dish))  {  dishOrders[dish]++;  }  else  {  dishOrders[dish] = 1;  }  }  }  return dishOrders;  }  static void PrintDishCategories()  {  Console.WriteLine("Существующие категории блюд:");  foreach (DishCategory category in Enum.GetValues(typeof(DishCategory)))  {  Console.WriteLine($"- {category}");  }  }  static void CreateDish()  {  Console.Write("Введите название блюда: ");  string name = Console.ReadLine();  Console.Write("Введите состав блюда: ");  string ingredients = Console.ReadLine();  Console.Write("Введите вес блюда: ");  string weight = Console.ReadLine();  Console.Write("Введите цену блюда: ");  double price = double.Parse(Console.ReadLine());  PrintDishCategories();  Console.Write("Введите категорию блюда: ");  DishCategory category = (DishCategory)Enum.Parse(typeof(DishCategory), Console.ReadLine());  Console.Write("Введите время приготовления блюда: ");  int prepTime = int.Parse(Console.ReadLine());  Console.Write("Введите типы блюда (через запятую): ");  string[] types = Console.ReadLine().Split(',');  var newDish = new Dish();  int nextId = GetNextAvailableId(\_dishes);  newDish.Id = nextId;  newDish.CreateDish(name, ingredients, weight, price, category, prepTime, types);  \_dishes.Add(newDish);  Console.WriteLine($"Блюдо '{name}' успешно создано с ID: {nextId}");  }  static void EditDish()  {  Console.Write("Введит ID блюда для изменения: ");  int id = int.Parse(Console.ReadLine());  var dish = \_dishes.Find(d => d.Id == id);  if (dish == null)  {  Console.WriteLine($"Блюдо с ID '{id}' не найдено.");  return;  }  if (IsDishUsedInOpenOrders(id))  {  Console.WriteLine($"Невозможно изменить блюдо '{dish.Name}'. Оно используется в открытых заказах.");  return;  }  Console.Write("Введите новое название блюда: ");  string name = Console.ReadLine();  Console.Write("Введите новые ингридиенты для блюда: ");  string ingredients = Console.ReadLine();  Console.Write("Введите новый вес блюда: ");  string weight = Console.ReadLine();  Console.Write("Введите новую цену блюда: ");  double price = double.Parse(Console.ReadLine());  PrintDishCategories();  Console.Write("Введите новую категорию блюда: ");  DishCategory category = (DishCategory)Enum.Parse(typeof(DishCategory), Console.ReadLine());  Console.Write("Введите новое время готовки блюда: ");  int prepTime = int.Parse(Console.ReadLine());  Console.Write("Введите новые типы блюда (через запятую): ");  string[] types = Console.ReadLine().Split(',');  dish.EditDish(name, ingredients, weight, price, category, prepTime, types);  Console.WriteLine($"Блюдо '{dish.Name}' успешно обновлено.");  }  static void DeleteDish()  {  Console.Write("Введите ID блюда для удаления: ");  int id = int.Parse(Console.ReadLine());  var dish = \_dishes.Find(d => d.Id == id);  if (dish == null)  {  Console.WriteLine($"Блюда с ID '{id}' не найдено.");  return;  }  if (IsDishUsedInOpenOrders(id))  {  Console.WriteLine($"Невозможно удалить блюдо '{dish.Name}'. Оно используется в открытых заказах.");  return;  }  \_dishes.Remove(dish);  Console.WriteLine($"Блюдо'{dish.Name}' успешно удалено.");  }  static void CreateOrder()  {  Console.Write("Введите номер стола: ");  int tableId = int.Parse(Console.ReadLine());  Console.Write("Введите количество блюд в заказе: ");  int numDishes = int.Parse(Console.ReadLine());  List<Dish> dishes = new List<Dish>();  for (int i = 0; i < numDishes; i++)  {  Console.Write($"Введите {i + 1} ID блюда: ");  int dishId = int.Parse(Console.ReadLine());  var dish = \_dishes.Find(d => d.Id == dishId);  if (dish != null)  {  dishes.Add(dish);  }  else  {  Console.WriteLine($"Блюда с ID '{dishId}' не найдено.");  }  }  Console.Write("Введите комментарий к заказу: ");  string comment = Console.ReadLine();  Console.Write("Введите ID официанта: ");  int waiterId = int.Parse(Console.ReadLine());  var newOrder = new Order();  int nextId = GetNextAvailableOrderId(\_orders);  newOrder.Id = nextId;  newOrder.CreateOrder(in tableId, dishes, comment, DateTime.Now, in waiterId);  \_orders.Add(newOrder);  Console.WriteLine($"Заказ с ID {nextId} создан успешно");  }  static void EditOrder()  {  Console.Write("Введите ID заказа для редактирования: ");  int id = int.Parse(Console.ReadLine());  var order = \_orders.Find(o => o.Id == id);  if (order == null)  {  Console.WriteLine($"Заказа с ID '{id}' не найдено.");  return;  }  // Проверка, не закрыт ли заказ  if (order.CloseTime != DateTime.MinValue)  {  Console.WriteLine($"Невозможно отредактирвать заказ с ID {id} так как он уже закрыт.");  return;  }  Console.Write("Введите новые блюда (через запятую): ");  string[] dishIds = Console.ReadLine().Split(',');  List<Dish> newDishes = new List<Dish>();  foreach (var dishId in dishIds)  {  var dish = \_dishes.Find(d => d.Id == int.Parse(dishId));  if (dish != null)  {  newDishes.Add(dish);  }  else  {  Console.WriteLine($"Блюда с ID '{dishId}' не найдено.");  }  }  Console.Write("Введите новый комментарий к заказу: ");  string newComment = Console.ReadLine();  order.EditOrder(ref newDishes, newComment);  Console.WriteLine($"Заказ {id} успешно обновлен.");  }  static void CloseOrder()  {  Console.Write("Введите ID заказа для закрытия: ");  int id = int.Parse(Console.ReadLine());  var order = \_orders.Find(o => o.Id == id);  if (order == null)  {  Console.WriteLine($"Заказа с ID '{id}' не найдено.");  return;  }  if (order.CloseTime != DateTime.MinValue)  {  Console.WriteLine($"Заказ с ID {id} уже был закрыт в {order.CloseTime}.");  return;  }  double finalCost;  order.CloseOrder(out finalCost);  Console.WriteLine($"Заказ {id} успешно закрыт. Финальная стоимость: {finalCost:C}");  }  static void PrintReceipt()  {  Console.Write("Введите ID заказа для печати чека: ");  int id = int.Parse(Console.ReadLine());  var order = \_orders.Find(o => o.Id == id);  if (order == null)  {  Console.WriteLine($"Заказа с ID '{id}' не найдено.");  return;  }  // Проверка, закрыт ли заказ  if (order.CloseTime == DateTime.MinValue)  {  Console.WriteLine($"Невозможно вывести чек для заказа с ID {id}. Заказ еще не закрыт.");  return;  }  order.PrintReceipt();  }  static int GetUserChoice()  {  Console.WriteLine();  Console.Write("Введите свой выбор: ");  Console.WriteLine();  return int.Parse(Console.ReadLine());  }  private static int GetNextAvailableId(List<Dish> dishes)  {  if (!dishes.Any()) return 1;  var sortedIds = dishes.Select(d => d.Id).OrderBy(id => id).ToList();  for (int i = 0; i < sortedIds.Count; i++)  {  if (sortedIds[i] != i + 1)  {  return i + 1;  }  }  return sortedIds.Count + 1;  }  private static int GetNextAvailableOrderId(List<Order> orders)  {  if (!orders.Any()) return 1;  var sortedIds = orders.Select(o => o.Id).OrderBy(id => id).ToList();  for (int i = 0; i < sortedIds.Count; i++)  {  if (sortedIds[i] != i + 1)  {  return i + 1;  }  }  return sortedIds.Count + 1;  }  static bool IsDishUsedInOpenOrders(int dishId)  {  return \_orders.Any(o =>  o.CloseTime == DateTime.MinValue &&  o.Dishes.Any(d => d.Id == dishId)  );  }  }  Dish.cs  using System;  using System.Collections.Generic;  public enum DishCategory  {  DrinksDon,  SaladsDon,  ColdAppetizersDon,  HotAppetizersDon,  SoupsDon,  MainDishesDon,  DessertsDon  }  public class Dish  {  public int Id { get; set; }  public string Name { get; set; }  public string Ingredients { get; set; }  public string Weight { get; set; }  public double Price { get; set; }  public DishCategory Category { get; set; }  public int PrepTime { get; set; }  public string[] Type { get; set; }  public void CreateDish(string name, string ingredients, string weight, double price, DishCategory category, int prepTime, params string[] type)  {  Name = name;  Ingredients = ingredients;  Weight = weight;  Price = price;  Category = category;  PrepTime = prepTime;  Type = type;  }  public void EditDish(string name, string ingredients, string weight, double price, DishCategory category, int prepTime, params string[] type)  {  Name = name;  Ingredients = ingredients;  Weight = weight;  Price = price;  Category = category;  PrepTime = prepTime;  Type = type;  }  public void PrintDishInfo()  {  Console.WriteLine($"ID: {Id}");  Console.WriteLine($"НазваниеДон: {Name}");  Console.WriteLine($"СоставДон: {Ingredients}");  Console.WriteLine($"ВесДон: {Weight}");  Console.WriteLine($"ЦенаДон: {Price}");  Console.WriteLine($"КатегорияДон: {Category}");  Console.WriteLine($"Время приготовления Дон: {PrepTime} минут");  Console.WriteLine("ТипДон: ");  foreach (var dishType in Type)  {  Console.WriteLine($"- {dishType}");  }  }  public void DeleteDish(List<Dish> dishes)  {  // Поиск блюда по ID  var dish = dishes.Find(d => d.Id == Id);  if (dish != null)  {  // Удаление блюда из списка  dishes.Remove(dish);  Console.WriteLine($"Блюдо '{Name}' было удалено Дон.");  }  else  {  Console.WriteLine($"Блюдо с ID '{Id}' не найдено Дон.");  }  }  }  Order.cs  using System;  using System.Collections.Generic;  public class Order  {  public int Id { get; set; }  public int TableId { get; set; }  public List<Dish> Dishes { get; set; }  public string Comment { get; set; }  public DateTime OrderTime { get; set; }  public int WaiterId { get; set; }  public DateTime CloseTime { get; set; }  public double TotalCost { get; set; }  public void CreateOrder(in int tableId, List<Dish> dishes, string comment, DateTime orderTime, in int waiterId)  {  TableId = tableId;  Dishes = dishes;  Comment = comment;  OrderTime = orderTime;  WaiterId = waiterId;  }  public void EditOrder(ref List<Dish> dishes, string comment)  {  Dishes = dishes;  Comment = comment;  }  public void PrintOrderInfo()  {  Console.WriteLine($"Номер заказа Дон: {Id}");  Console.WriteLine($"Номер стола Дон: {TableId}");  Console.WriteLine("Заказанные блюда Дон:");  foreach (var dish in Dishes)  {  Console.WriteLine($"- {dish.Name} x {dish.Price}");  }  Console.WriteLine($"Комментарий Дон: {Comment}");  Console.WriteLine($"Время заказа Дон: {OrderTime}");  Console.WriteLine($"Номер официанта Дон: {WaiterId}");  Console.WriteLine($"Время закрытия заказа Дон: {CloseTime}");  Console.WriteLine($"Полная стоимость дон: {TotalCost}");  }  public void CloseOrder(out double finalCost)  {  CloseTime = DateTime.Now;  CalculateTotalCost();  finalCost = TotalCost;  }  public void PrintReceipt()  {  Console.WriteLine("==========================================");  Console.WriteLine($"Стол: {TableId}".PadRight(20) + $"Официант: {WaiterId}".PadLeft(30));  Console.WriteLine($"Период: {OrderTime:yyyy-MM-dd HH:mm} - {CloseTime:yyyy-MM-dd HH:mm}");  Console.WriteLine("==========================================");  Console.WriteLine();  // Группируем блюда по категориям и подсчитываем количество каждого блюда  var groupedDishes = Dishes  .GroupBy(d => d.Category)  .ToDictionary(  g => g.Key,  g => g.GroupBy(d => d.Name)  .Select(dg => new {  Dish = dg.First(),  Count = dg.Count(),  SubTotal = dg.Count() \* dg.First().Price  })  .ToList()  );  foreach (var category in groupedDishes)  {  Console.WriteLine($"{category.Key}:");  Console.WriteLine("------------------------------------------");  double categoryTotal = 0;  foreach (var dishGroup in category.Value)  {  // Форматируем строку: название блюда слева, количество и цена справа  string dishInfo = $"{dishGroup.Dish.Name}";  string priceInfo = $"{dishGroup.Count} x {dishGroup.Dish.Price:F2} = {dishGroup.SubTotal:F2}";  // Padding для выравнивания  Console.WriteLine($" {dishInfo.PadRight(40)} {priceInfo.PadLeft(20)}");  categoryTotal += dishGroup.SubTotal;  }  Console.WriteLine("------------------------------------------");  Console.WriteLine($"{"Подитог категории:".PadRight(40)} {categoryTotal:F2}".PadLeft(60));  Console.WriteLine();  }  Console.WriteLine("==========================================");  Console.WriteLine($"{"ИТОГО:".PadRight(40)} {TotalCost:F2}".PadLeft(60));  Console.WriteLine("==========================================");  }  private void CalculateTotalCost()  {  TotalCost = Dishes.Sum(d => d.Price);  }  } |
| ***Ответ:*** | |
|  |  |