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:*** | |
|  | |  | | --- | | namespace RT\_4; using System; using System.Collections.Generic;  class Program {  private static List<Dish> *menu* = *InitializeMenu*(); // Инициализация начального набора блюд   static void *Main*()  {  List<Order> orders = new List<Order>();  bool running = true;   while (running)  {  Console.*WriteLine*("\nСписок действий:");  Console.*WriteLine*("1. Создать заказ");  Console.*WriteLine*("2. Создать несколько заказов");  Console.*WriteLine*("3. Создать блюдо");  Console.*WriteLine*("4. Вывести меню");  Console.*WriteLine*("5. Закрыть заказ");  Console.*WriteLine*("6. Показать чек");  Console.*WriteLine*("7. Подсчитать стоимость всех закрытых заказов");  Console.*WriteLine*("8. Подсчитать закрытые заказы конкретного официанта");  Console.*WriteLine*("9. Собрать статистику по количеству заказанных блюд");  Console.*WriteLine*("10. Показать все заказы (открытые и закрытые)");  Console.*WriteLine*("11. Изменить заказ");  Console.*WriteLine*("12. Выйти");  Console.*Write*("Выберите действие: ");  string choice = Console.*ReadLine*();   switch (choice)  {  case "1":  *CreateOrder*(orders);  break;  case "2":  *CreateMultipleOrders*(orders);  break;  case "3":  *CreateDish*();  break;  case "4":  *DisplayMenu*();  break;  case "5":  *CloseOrder*(orders);  break;  case "6":  *PrintReceipt*(orders);  break;  case "7":  *CalculateAndDisplayTotalClosedOrders*(orders);  break;  case "8":  *CalculateAndDisplayClosedOrdersByWaiter*(orders);  break;  case "9":  *DisplayDishStatistics*(orders);  break;  case "10":  *DisplayAllOrders*(orders);  break;  case "11":  *ModifyOrder*(orders);  break;  case "12":  running = false;  break;  default:  Console.*WriteLine*("Неверный выбор, попробуйте еще раз.");  break;  }  }  }   */// <summary>  /// Инициализация меню с набором блюд по умолчанию  /// </summary>* static List<Dish> *InitializeMenu*()  {  return new List<Dish>  {  new Dish("Суп1", "Вода, Соль", "200г", 599.99f, Category.*Soups*, 10, "Вегетарианское"),  new Dish("Суп2", "Вода, Курица", "250г", 799.99f, Category.*Soups*, 15, "Не вегетарианское"),  new Dish("Напиток1", "Вода, Сахар", "150мл", 299.99f, Category.*Drinks*, 2, "Холодный"),  new Dish("Десерт1", "Мука, Сахар", "100г", 499.99f, Category.*Desserts*, 5, "Сладкое")  };  }   */// <summary>  /// Создание нового блюда через ввод данных с консоли  /// </summary>* static void *CreateDish*()  {  Console.*Write*("Сколько блюд вы хотите создать? ");  int numberOfDishes = int.*Parse*(Console.*ReadLine*());   for (int i = 0; i < numberOfDishes; i++)  {  Console.*WriteLine*($"Создание блюда {i + 1}:");  Console.*Write*("Введите название блюда: ");  string name = Console.*ReadLine*();   Console.*Write*("Введите состав блюда: ");  string ingredients = Console.*ReadLine*();   Console.*Write*("Введите вес блюда: ");  string weight = Console.*ReadLine*();   Console.*Write*("Введите цену блюда: ");  float price = float.*Parse*(Console.*ReadLine*());   Console.*Write*("Введите категорию блюда (0 - Drinks, 1 - Salads, 2 - ColdAppetizers, 3 - HotAppetizers, 4 - Soups, 5 - MainCourses, 6 - Desserts): ");  Category category = (Category)int.*Parse*(Console.*ReadLine*());   Console.*Write*("Введите время готовки блюда (в минутах): ");  int cookingTime = int.*Parse*(Console.*ReadLine*());   Console.*Write*("Введите характеристики блюда (через запятую): ");  string[] types = Console.*ReadLine*().Split(',');   Dish newDish = new Dish( name, ingredients, weight, price, category, cookingTime, types);  *menu*.Add(newDish);   Console.*WriteLine*($"Блюдо '{name}' создано.");  }  }   */// <summary>  /// Создание одного заказа с автоматическим присвоением ID  /// </summary>* static void *CreateOrder*(List<Order> orders)  {  int tableId, waiterId;  *GetOrderDetails*(out tableId, out waiterId);   Order order = new Order(tableId, waiterId);  *AddDishesToOrder*(order);  orders.Add(order);   Console.*WriteLine*("Заказ создан.");  }   */// <summary>  /// Создание нескольких заказов  /// </summary>* static void *CreateMultipleOrders*(List<Order> orders)  {  Console.*Write*("Сколько заказов вы хотите создать? ");  int numberOfOrders = int.*Parse*(Console.*ReadLine*());   for (int i = 0; i < numberOfOrders; i++)  {  Console.*WriteLine*($"Создание заказа {i + 1}:");  *CreateOrder*(orders);  }  }   */// <summary>  /// Метод для ввода общих данных заказа (ID столика и официанта)  /// </summary>* static void *GetOrderDetails*(out int tableId, out int waiterId)  {  Console.*Write*("Введите ID столика: ");  tableId = int.*Parse*(Console.*ReadLine*());   Console.*Write*("Введите ID официанта: ");  waiterId = int.*Parse*(Console.*ReadLine*());  }   */// <summary>  /// Метод для добавления блюд в заказ  /// </summary>* static void *AddDishesToOrder*(Order order)  {  Console.*WriteLine*("Меню:");  *DisplayMenu*();   while (true)  {  Console.*Write*("Введите ID блюда для добавления в заказ (или 0 для завершения): ");  int dishId = int.*Parse*(Console.*ReadLine*());  if (dishId == 0) break;   Dish dish = *menu*.Find(d => d.Id == dishId);  if (dish != null)  {  order.AddDish(dish);  Console.*WriteLine*($"Добавлено блюдо: {dish.Name}");  }  else  {  Console.*WriteLine*("Блюдо не найдено.");  }  }  }   */// <summary>  /// Вывод меню, отсортированного по категориям  /// </summary>* static void *DisplayMenu*()  {  var groupedMenu = new Dictionary<Category, List<Dish>>();   foreach (var dish in *menu*)  {  if (!groupedMenu.ContainsKey(dish.DishCategory))  groupedMenu[dish.DishCategory] = new List<Dish>();  groupedMenu[dish.DishCategory].Add(dish);  }   foreach (var category in groupedMenu.Keys)  {  Console.*WriteLine*($"\n{category}:");  foreach (var dish in groupedMenu[category])  {  Console.*WriteLine*($"{dish.Id} - {dish.Name} ({dish.Weight}), Цена: {dish.Price} руб.");  }  }  }   */// <summary>  /// Закрытие заказа по ID  /// </summary>* static void *CloseOrder*(List<Order> orders)  {  Console.*Write*("Введите ID заказа для закрытия: ");  int id = int.*Parse*(Console.*ReadLine*());  Order order = orders.Find(o => o.Id == id);   if (order != null)  {  order.CloseOrder();  Console.*WriteLine*("Заказ закрыт.");  }  else  {  Console.*WriteLine*("Заказ не найден.");  }  }   */// <summary>  /// Печать чека для конкретного заказа  /// </summary>* static void *PrintReceipt*(List<Order> orders)  {  Console.*Write*("Введите ID заказа для печати чека: ");  int id = int.*Parse*(Console.*ReadLine*());  Order order = orders.Find(o => o.Id == id);   if (order != null)  {  order.PrintReceipt();  }  else  {  Console.*WriteLine*("Заказ не найден.");  }  }    */// <summary>  /// Подсчет общей стоимости всех закрытых заказов  /// </summary>* static float *CalculateTotalClosedOrders*(List<Order> orders)  {  float total = 0;  foreach (var order in orders)  {  if (order.CloseTime != null)  {  total += order.TotalCost;  }  }  return total;  }    */// <summary>  /// Подсчет общей стоимости всех закрытых заказов и вывод результата  /// </summary>* static void *CalculateAndDisplayTotalClosedOrders*(List<Order> orders)  {  float total = *CalculateTotalClosedOrders*(orders);  Console.*WriteLine*($"Общая стоимость всех закрытых заказов: {total} руб.");  }   */// <summary>  /// Подсчет количества закрытых заказов конкретного официанта и вывод результата  /// </summary>* static void *CalculateAndDisplayClosedOrdersByWaiter*(List<Order> orders)  {  Console.*Write*("Введите ID официанта для подсчета закрытых заказов: ");  int waiterId = int.*Parse*(Console.*ReadLine*());  int count = *CountClosedOrdersByWaiter*(orders, waiterId);  Console.*WriteLine*($"Количество закрытых заказов у официанта {waiterId}: {count}");  }    */// <summary>  /// Подсчет количества закрытых заказов конкретного официанта  /// </summary>* static int *CountClosedOrdersByWaiter*(List<Order> orders, int waiterId)  {  int count = 0;  foreach (var order in orders)  {  if (order.CloseTime != null && order.WaiterId == waiterId)  {  count++;  }  }  return count;  }   */// <summary>  /// Сбор и отображение статистики по заказанным блюдам  /// </summary>* static void *DisplayDishStatistics*(List<Order> orders)  {  var dishStats = *CollectDishStatistics*(orders);  if (dishStats.Count == 0)  {  Console.*WriteLine*("Ни один заказ не закрыт.");  return;  }  Console.*WriteLine*("Статистика по заказанным блюдам:");  foreach (var entry in dishStats)  {  Console.*WriteLine*($"{entry.Key}: {entry.Value} раз");  }  }    */// <summary>  /// Метод для отображения всех заказов (открытых и закрытых)  /// </summary>  /// <param name="orders">Список заказов</param>* static void *DisplayAllOrders*(List<Order> orders)  {  Console.*WriteLine*("Список всех заказов:");   // Фильтрация и вывод открытых заказов  Console.*WriteLine*("\nОткрытые заказы:");  var openOrders = orders.Where(o => o.CloseTime == null).ToList();  if (openOrders.Any())  {  foreach (var order in openOrders)  {  Console.*WriteLine*($"ID заказа: {order.Id}, Столик: {order.TableId}, Официант: {order.WaiterId}, Время принятия: {order.OrderTime}");  if (order.Dishes.Any())  {  foreach (var dish in order.Dishes.GroupBy(d => d.Name))  {  int count = dish.Count();  float cost = count \* dish.First().Price;  Console.*WriteLine*($"- {dish.Key}: {count} шт. \* {dish.First().Price:F2} = {cost:F2}");  }  }  else  {  Console.*WriteLine*("Заказ пуст.");  }  }  }  else  {  Console.*WriteLine*("Нет открытых заказов.");  }   // Фильтрация и вывод закрытых заказов  Console.*WriteLine*("\nЗакрытые заказы:");  var closedOrders = orders.Where(o => o.CloseTime != null).ToList();  if (closedOrders.Any())  {  foreach (var order in closedOrders)  {  Console.*WriteLine*($"\nID заказа: {order.Id}, Столик: {order.TableId}, Официант: {order.WaiterId}, Время закрытия: {order.CloseTime}");  Console.*WriteLine*("Состав заказа:");  if (order.Dishes.Any())  {  foreach (var dish in order.Dishes.GroupBy(d => d.Name))  {  int count = dish.Count();  float cost = count \* dish.First().Price;  Console.*WriteLine*($"- {dish.Key}: {count} шт. \* {dish.First().Price:F2} = {cost:F2}");  }  }  else  {  Console.*WriteLine*("Заказ пуст.");  }  }  }  else  {  Console.*WriteLine*("Нет закрытых заказов.");  }  }    */// <summary>  /// Метод для изменения заказа (добавление и удаление блюд)  /// </summary>  /// <param name="orders">Список заказов</param>* static void *ModifyOrder*(List<Order> orders)  {  Console.*Write*("Введите ID заказа для редактирования: ");  int orderId = int.*Parse*(Console.*ReadLine*());  Order order = orders.FirstOrDefault(o => o.Id == orderId);   if (order == null)  {  Console.*WriteLine*("Заказ с указанным ID не найден.");  return;  }   if (order.CloseTime != null)  {  Console.*WriteLine*("Заказ уже закрыт. Изменение невозможно.");  return;  }   bool editing = true;   while (editing)  {  Console.*WriteLine*("\n1. Добавить блюдо");  Console.*WriteLine*("2. Удалить блюдо");  Console.*WriteLine*("3. Просмотреть состав заказа");  Console.*WriteLine*("4. Завершить редактирование");  Console.*Write*("Выберите действие: ");  string choice = Console.*ReadLine*();   switch (choice)  {  case "1":  *AddDishToOrder*(order);  break;  case "2":  *RemoveDishFromOrder*(order);  break;  case "3":  *DisplayOrderContents*(order);  break;  case "4":  editing = false;  break;  default:  Console.*WriteLine*("Неверный выбор. Попробуйте снова.");  break;  }  }  }    */// <summary>  /// Метод для добавления блюда в заказ  /// </summary>  /// <param name="order">Редактируемый заказ</param>* static void *AddDishToOrder*(Order order)  {  Console.*WriteLine*("Меню:");  *DisplayMenu*();   Console.*Write*("Введите ID блюда для добавления: ");  int dishId = int.*Parse*(Console.*ReadLine*());  Dish dish = *menu*.FirstOrDefault(d => d.Id == dishId);   if (dish != null)  {  order.AddDish(dish);  Console.*WriteLine*($"Блюдо '{dish.Name}' добавлено в заказ.");  }  else  {  Console.*WriteLine*("Блюдо с указанным ID не найдено.");  }  }    */// <summary>  /// Метод для удаления блюда из заказа  /// </summary>  /// <param name="order">Редактируемый заказ</param>* static void *RemoveDishFromOrder*(Order order)  {  if (!order.Dishes.Any())  {  Console.*WriteLine*("Заказ пуст. Нечего удалять.");  return;  }   Console.*WriteLine*("Состав заказа:");  *DisplayOrderContents*(order);   Console.*Write*("Введите название блюда для удаления: ");  string dishName = Console.*ReadLine*();   Dish dishToRemove = order.Dishes.FirstOrDefault(d => d.Name.Equals(dishName, StringComparison.*OrdinalIgnoreCase*));   if (dishToRemove != null)  {  order.RemoveDish(dishToRemove);  Console.*WriteLine*($"Блюдо '{dishToRemove.Name}' удалено из заказа.");  }  else  {  Console.*WriteLine*("Блюдо с указанным названием не найдено в заказе.");  }  }    */// <summary>  /// Метод для отображения состава заказа  /// </summary>  /// <param name="order">Редактируемый заказ</param>* static void *DisplayOrderContents*(Order order)  {  if (!order.Dishes.Any())  {  Console.*WriteLine*("Заказ пуст.");  return;  }   Console.*WriteLine*("Состав заказа:");  foreach (var dish in order.Dishes.GroupBy(d => d.Name))  {  int count = dish.Count();  float cost = count \* dish.First().Price;  Console.*WriteLine*($"- {dish.Key}: {count} шт. \* {dish.First().Price:F2} = {cost:F2}");  }  }    */// <summary>  /// Сбор статистики по количеству каждого заказанного блюда  /// </summary>* static Dictionary<string, int> *CollectDishStatistics*(List<Order> orders)  {  var dishStats = new Dictionary<string, int>();   foreach (var order in orders)  {  if (order.CloseTime != null)  {  foreach (var dish in order.Dishes)  {  if (dishStats.ContainsKey(dish.Name))  {  dishStats[dish.Name]++;  }  else  {  dishStats[dish.Name] = 1;  }  }  }  }   return dishStats;  }   */// <summary>  /// Показ статистики по заказам и блюдам  /// </summary>* static void ShowStatistics(List<Order> orders)  {  Console.*WriteLine*($"Общая стоимость всех закрытых заказов: {Statistics.*CalculateTotalClosedOrders*(orders)} руб.");  Console.*Write*("Введите ID официанта для подсчета закрытых заказов: ");  int waiterId = int.*Parse*(Console.*ReadLine*());  Console.*WriteLine*($"Количество закрытых заказов у официанта {waiterId}: {Statistics.*CountClosedOrdersByWaiter*(orders, waiterId)}");   var dishStats = Statistics.*CollectDishStatistics*(orders);  Console.*WriteLine*("Статистика по заказанным блюдам:");  foreach (var entry in dishStats)  {  Console.*WriteLine*($"{entry.Key}: {entry.Value} раз");  }  } } | | namespace RT\_4; using System;  */// <summary> /// Перечисление категорий блюд /// </summary>* public enum Category {  *Drinks*,  Salads,  ColdAppetizers,  HotAppetizers,  *Soups*,  MainCourses,  *Desserts* }  */// <summary> /// Класс "Блюдо" для хранения информации о блюде и выполнения операций с ним /// </summary>* public class Dish {  private static int *idCounter* = 1; // Статическое поле-счетчик для генерации ID  public int Id { get; set; }  public string Name { get; set; }  public string Ingredients { get; set; }  public string Weight { get; set; }  public float Price { get; set; }  public Category DishCategory { get; set; }  public int CookingTime { get; set; }  public string[] Types { get; set; }   */// <summary>  /// Конструктор для создания нового блюда  /// </summary>* public Dish(string name, string ingredients, string weight, float price, Category category, int cookingTime, params string[] types)  {  Id = *idCounter*++;  Name = name;  Ingredients = ingredients;  Weight = weight;  Price = price;  DishCategory = category;  CookingTime = cookingTime;  Types = types;  }   */// <summary>  /// Метод для создания нового блюда  /// </summary>* public static Dish CreateDish(string name, string ingredients, string weight, float price, Category category, int cookingTime, params string[] types)  {  return new Dish(name, ingredients, weight, price, category, cookingTime, types);  }   */// <summary>  /// Метод для редактирования информации о блюде  /// </summary>* public void EditDish(string newName, string newIngredients, string newWeight, float newPrice, Category newCategory, int newCookingTime, params string[] newTypes)  {  Name = newName;  Ingredients = newIngredients;  Weight = newWeight;  Price = newPrice;  DishCategory = newCategory;  CookingTime = newCookingTime;  Types = newTypes;  }   */// <summary>  /// Метод для вывода информации о блюде  /// </summary>* public void DisplayInfo()  {  Console.*WriteLine*($"{Name} - Состав: {Ingredients}, Вес: {Weight}, Цена: {Price}");  }   */// <summary>  /// Метод для удаления блюда (вывод сообщения об удалении)  /// </summary>* public void DeleteDish()  {  Console.*WriteLine*($"Блюдо '{Name}' удалено.");  } } | | namespace RT\_4; using System; using System.Collections.Generic; using System.Linq;  */// <summary> /// Класс "Заказ" для хранения информации о заказе и выполнения операций с ним /// </summary>* public class Order {  private static int *idCounter* = 1; // Статическое поле-счетчик для автоматической генерации ID  public int Id { get; set; }  public int TableId { get; set; }  public List<Dish> Dishes { get; set; }  public string Comment { get; set; }  public DateTime OrderTime { get; private set; }  public int WaiterId { get; set; }  public DateTime? CloseTime { get; private set; }  public float TotalCost { get; private set; }   public Order(int tableId, int waiterId, string comment = "")  {  Id = *idCounter*++;  TableId = tableId;  WaiterId = waiterId;  Comment = comment;  OrderTime = DateTime.Now;  Dishes = new List<Dish>();  }   */// <summary>  /// Метод для добавления блюда в заказ  /// </summary>* public void AddDish(Dish dish)  {  Dishes.Add(dish);  }   */// <summary>  /// Метод для удаления блюда из заказа  /// </summary>* public void RemoveDish(Dish dish)  {  Dishes.Remove(dish);  }   */// <summary>  /// Метод для закрытия заказа и подсчета итоговой стоимости  /// </summary>* public void CloseOrder()  {  CloseTime = DateTime.Now;  TotalCost = Dishes.Sum(dish => dish.Price);  }   */// <summary>  /// Метод для изменения комментария к заказу  /// </summary>* public void ModifyOrder(string newComment)  {  if (CloseTime == null)  {  Comment = newComment;  Console.*WriteLine*("Комментарий к заказу изменен.");  }  else  {  Console.*WriteLine*("Заказ уже закрыт. Изменение невозможно.");  }  }   */// <summary>  /// Метод для вывода информации о заказе  /// </summary>* public void DisplayOrderInfo()  {  Console.*WriteLine*($"Заказ ID: {Id}, Столик: {TableId}, Официант: {WaiterId}");  Console.*WriteLine*($"Время заказа: {OrderTime}");  Console.*WriteLine*("Блюда:");  foreach (var dish in Dishes)  {  Console.*WriteLine*($"- {dish.Name}: {dish.Price} руб.");  }  Console.*WriteLine*($"Итоговая стоимость: {TotalCost} руб.");  }   */// <summary>  /// Метод для вывода чека  /// </summary>* public void PrintReceipt()  {  if (CloseTime != null)  {  Console.*WriteLine*("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Чек \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  Console.*WriteLine*($"Столик: {TableId}");  Console.*WriteLine*($"Официант: {WaiterId}");  Console.*WriteLine*($"Период обслуживания: с {OrderTime} по {CloseTime}");  Console.*WriteLine*();   // Группировка блюд по категориям  var categories = Dishes.GroupBy(d => d.DishCategory);  foreach (var categoryGroup in categories)  {  Console.*WriteLine*($"{categoryGroup.Key}:");    // Группировка блюд по названию для подсчета количества  var groupedDishes = categoryGroup  .GroupBy(d => d.Name)  .Select(g => new  {  Name = g.Key,  Count = g.Count(),  Price = g.First().Price // Цена одинакова для одинаковых блюд  });   float categoryTotal = 0;   foreach (var dishGroup in groupedDishes)  {  float cost = dishGroup.Count \* dishGroup.Price;  categoryTotal += cost;  Console.*WriteLine*($"{dishGroup.Name} \t {dishGroup.Count} \* {dishGroup.Price:F2} = {cost:F2}");  }   Console.*WriteLine*($"Подитог для {categoryGroup.Key}: {categoryTotal:F2}");  Console.*WriteLine*();  }   Console.*WriteLine*($"Итоговая стоимость: {TotalCost:F2}");  Console.*WriteLine*("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  }  else  {  Console.*WriteLine*("Заказ не закрыт.");  }  } } | | namespace RT\_4; using System; using System.Collections.Generic; using System.Linq;  */// <summary> /// Класс для сбора статистики по заказам и блюдам /// </summary>* public class Statistics {  */// <summary>  /// Подсчет суммы всех закрытых заказов  /// </summary>* public static float *CalculateTotalClosedOrders*(List<Order> orders)  {  return orders.Where(order => order.CloseTime != null).Sum(order => order.TotalCost);  }   */// <summary>  /// Подсчет количества закрытых заказов у конкретного официанта  /// </summary>* public static int *CountClosedOrdersByWaiter*(List<Order> orders, int waiterId)  {  return orders.Count(order => order.CloseTime != null && order.WaiterId == waiterId);  }   */// <summary>  /// Сбор статистики по количеству заказов каждого блюда  /// </summary>* public static Dictionary<string, int> *CollectDishStatistics*(List<Order> orders)  {  var dishCounts = new Dictionary<string, int>();  foreach (var order in orders.Where(o => o.CloseTime != null))  {  foreach (var dish in order.Dishes)  {  if (dishCounts.ContainsKey(dish.Name))  {  dishCounts[dish.Name]++;  }  else  {  dishCounts[dish.Name] = 1;  }  }  }  return dishCounts;  } } | |
| ***Ответ:*** | |
|  | 1. Создать заказ    2. Создать несколько заказов      3. Создать блюдо    4. Вывести меню    5. Закрыть заказ        6. Показать чек        7. Подсчитать стоимость всех закрытых заказов    8. Подсчитать закрытые заказы конкретного официанта    9. Собрать статистику по количеству заказанных блюд    10. Показать все заказы (открытые и закрытые)    11. Изменить заказ              12. Выйти |