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:	
Ответ:	