Статические члены и модификатор static

Кроме обычных полей, методов, свойств класс может иметь статические поля, методы, свойства. Статические поля, методы, свойства относятся ко всему классу и для обращения к подобным членам класса необязательно создавать экземпляр класса. Например:

```
1   class Account
2   {
3       public static decimal bonus = 100;
4       public decimal totalSum;
5       public Account(decimal sum)
6       {
7           totalSum = sum + bonus;
8       }
9   }
10  class Program
11  {
12      static void Main(string[] args)
13      {
14          Console.WriteLine(Account.bonus); // 100
15          Account.bonus += 200;
16
17          Account account1 = new Account(150);
18          Console.WriteLine(account1.totalSum); // 450
19
20
21          Account account2 = new Account(1000);
22          Console.WriteLine(account2.totalSum); // 1300
23
24          Console.ReadKey();
25      }
26  }
```

В данном случае класс Account имеет два поля: bonus и totalSum. Поле bonus является статическим, поэтому оно хранит состояние класса в целом, а не отдельного объекта. И поэтому мы можем обращаться к этому полю по имени класса:

```
1   Console.WriteLine(Account.bonus);
2   Account.bonus += 200;
```

На уровне памяти для статических полей будет создаваться участок в памяти, который будет общим для всех объектов класса.

При этом память для статических переменных выделяется даже в том случае, если не создано ни одного объекта этого класса.

Статические свойства и методы

Подобным образом мы можем создавать и использовать статические методы и свойства:

```
1   class Account
```

```
{
    public Account(decimal sum, decimal rate)
    {
        if (sum < MinSum) throw new Exception("Недопустимая
сумма!");
        Sum = sum; Rate = rate;
    }
    private static decimal minSum = 100; // минимальная допустимая
сумма для всех счетов
    public static decimal MinSum
    {
        get { return minSum; }
        set { if(value>0) minSum = value; }
    }

    public decimal Sum { get; private set; } // сумма на счете
    public decimal Rate { get; private set; } // процентная ставка

    // подсчет суммы на счете через определенный период
   //по определенной ставке
    public static decimal GetSum(decimal sum, decimal rate, int
period)
    {
        decimal result = sum;
        for (int i = 1; i <= period; i++)
            result = result + result * rate / 100;
        return result;
    }
}
```

Переменная minSum, свойство MinSum, а также метод GetSum здесь определены с ключевым словом static, то есть они являются статическими.

Переменная minSum и свойство MinSum представляют минимальную сумму, которая допустима для создания счета. Этот оказатель не относится к какому-то конкретному счету, а относится ко всем счетам в целом. Если мы изменим этот показатель для одного счета, то он также должен измениться и для другого счета. То есть в отличии от свойств Sum и Rate, которые хранят состояние объекта, переменная minSum хранит состояние для всех объектов данного класса.

То же самое с методом GetSum - он вычисляет сумму на счете через определенный период по определенной процентной ставке для определенной начальной суммы. Вызов и результат этого метода не зависит от конкретного объекта или его состояния.

Таким образом, переменные и свойства, которые хранят состояние, общее для всех объектов класса, следует определять как статические. И также методы, которые определяют общее для всех объектов поведение, также следует объявлять как статические.

Статические члены класса являются общими для всех объектов этого класса, поэтому к ним надо обращаться по имени класса:

```
Account.MinSum = 560;
decimal result = Account.GetSum(1000, 10, 5);
```

Следует учитывать, что статические методы могут обращаться только статическим членам класса. Обращаться к нестатическим методам, полям, свойствам внутри статического метода мы не можем.

Нередко статические поля применяются для хранения счетчиков. Например, пусть у нас есть класс User, и мы хотим иметь счетчик, который позволял бы узнать, сколько объектов User создано:

```
class User
{
    private static int counter = 0;
    public User()
    {
        counter++;
    }

    public static void DisplayCounter()
    {
        Console.WriteLine($"Создано {counter} объектов User");
    }
}
class Program
{
    static void Main(string[] args)
    {
        User user1 = new User();
        User user2 = new User();
        User user3 = new User();
        User user4 = new User();
        User user5 = new User();

        User.DisplayCounter(); // 5

        Console.ReadKey();
    }
}
```

Статический конструктор

Кроме обычных конструкторов у класса также могут быть статические конструкторы. Статические конструкторы имеют следующие отличительные черты:

Статические конструкторы не должны иметь модификатор доступа и не принимают параметров

Как и в статических методах, в статических конструкторах нельзя использовать ключевое слово this для ссылки на текущий объект класса и можно обращаться только к статическим членам класса

Статические конструкторы нельзя вызвать в программе вручную. Они выполняются автоматически при самом первом создании объекта данного класса или при первом обращении к его статическим членам (если таковые имеются)

Статические конструкторы обычно используются для инициализации статических данных, либо же выполняют действия, которые требуется выполнить только один раз

Определим статический конструктор:

```
class User
{
    static User()
    {
        Console.WriteLine("Создан первый пользователь");
    }
}
class Program
{
    static void Main(string[] args)
    {
        User user1 = new User(); // здесь сработает статический
                                 //конструктор
        User user2 = new User();

        Console.Read();
    }
}
```

Статические классы

Статические классы объявляются с модификатором static и могут содержать только статические поля, свойства и методы. Например, если бы класс Account имел бы только статические переменные, свойства и методы, то его можно было бы объявить как статический:

```
static class Account
{
    private static decimal minSum = 100; // минимальная допустимая
                                         //сумма для всех счетов
    public static decimal MinSum
    {
        get { return minSum; }
        set { if(value>0) minSum = value; }
    }

    // подсчет суммы на счете через определенный период по
   // определенной ставке
    public static decimal GetSum(decimal sum, decimal rate, int
```

```
14  period)
15      {
16          decimal result = sum;
17          for (int i = 1; i <= period; i++)
18              result = result + result * rate / 100;
            return result;
        }
    }
```

В C# показательным примером статического класса является класс Math, который применяется для различных математических операций.