ЛЕКЦИЯ 3А. КЛАССЫ И ОБЪЕКТЫ В ЯЗЫКЕ C#. ОПЕРАЦИИ, ПЕРЕГРУЗКА ОПЕРАЦИЙ.

В данной теме мы продолжим работу над классами, созданными на прошлом занятии.

ОСНОВНЫЕ ВОПРОСЫ, КОТОРЫЕ РАССМАТРИВАЮТСЯ В ЛЕКЦИИ:

1. Операции класса 1
2. Унарные операции 2
3. Бинарные операции 3
4. Операции преобразования типа 3
Листинги для работы на занятии 4

ПОЛНЫЙ ТЕКСТ

1. Операции класса

C# позволяет переопределить действие большинства операций так, чтобы при использовании с объектами конкретного класса они выполняли заданные функции. Это дает возможность применять экземпляры собственных типов данных в составе выражений таким же образом, как стандартных, например:

```
MyObject a, b, c;
  ...
c = a + b;   // используется операция сложения класса MyObject
```

Определение собственных операций класса часто называют ***перегрузкой операций***. Перегрузка обычно применяется для классов, описывающих математические или физические понятия, то есть таких классов, для которых семантика операций делает программу более понятной. Если назначение операции интуитивно не понятно с первого взгляда, перегружать такую операцию не рекомендуется.

Операции класса описываются с помощью методов специального вида (***функций-операций***). Перегрузка операций похожа на перегрузку обычных методов. Синтаксис операции:

```
[ атрибуты ] спецификаторы объявитель_операции тело
```

В качестве ***спецификаторов*** одновременно используются ключевые слова **public** и **static**. Кроме того, операцию можно объявить как внешнюю (**extern**).

Объявитель операции содержит ключевое слово **operator**, по которому и опознается описание операции в классе. ***Тело*** операции определяет действия,

которые выполняются при использовании операции в выражении. Тело представляет собой блок, аналогичный телу других методов.

Новые обозначения для собственных операций вводить нельзя. Для операций класса сохраняются количество аргументов, приоритеты операций и правила ассоциации (справа налево или слева направо), используемые в стандартных типах данных.

При описании операций необходимо соблюдать следующие правила:

- операция должна быть описана как открытый статический метод класса (спецификаторы **public static**);
- параметры в операцию должны передаваться по значению (то есть **не должны** предваряться ключевыми словами **ref** или **out**);
- сигнатуры всех операций класса должны различаться;
- типы, используемые в операции, должны иметь не меньшие права доступа, чем сама операция (то есть должны быть доступны при использовании операции).

В C# существуют три вида операций класса: **унарные**, **бинарные** и операции **преобразования типа**.

2. Унарные операции

Можно определять в классе следующие ***унарные операции***:

```
+     -     !     ~     ++     --     true     false
```

Синтаксис объявителя унарной операции:

```
тип operator унарная_операция ( параметр )
```

Примеры заголовков унарных операций:

```
public static int operator +( MyObject m )
public static MyObject operator --( MyObject m )
public static bool operator true( MyObject m )
```

Параметр, передаваемый в операцию, должен иметь тип класса, для которого она определяется. Операция должна возвращать:

- для операций +, -, ! и ~ величину любого типа;
- для операций ++ и -- величину типа класса, для которого она определяется;
- для операций true и false величину типа bool.

Операции не должны изменять значение передаваемого им операнда. Операция, возвращающая величину типа класса, для которого она определяется, должна создать новый объект этого класса, выполнить с ним необходимые действия и передать его в качестве результата.

Префиксный и постфиксный инкремент не различаются (для них может существовать только одна реализация, которая вызывается в обоих случаях).

3. Бинарные операции

Можно определять в классе следующие бинарные операции:

```
+ - * / % & | ^ << >> == != > < >= <=
```

Синтаксис объявителя бинарной операции:

```
тип operator бинарная_операция (параметр1, параметр2)
```

Примеры заголовков бинарных операций:

```
public static MyObject operator +  ( MyObject m1, MyObject m2 )
public static bool     operator == ( MyObject m1, MyObject m2 )
```

Хотя бы один параметр, передаваемый в операцию, должен иметь тип класса, для которого она определяется. Операция может возвращать величину любого типа.

Операции == и !=, > и <, >= и <= определяются только парами и обычно возвращают логическое значение. Чаще всего в классе определяют операции сравнения на равенство и неравенство для того, чтобы обеспечить сравнение объектов, а не их ссылок, как определено по умолчанию для ссылочных типов.

Сложные операции присваивания (например, +=) определять не требуется, да это и невозможно. При выполнении такой операции автоматически вызываются сначала операция сложения, а потом присваивания.

4. Операции преобразования типа

Операции преобразования типа обеспечивают возможность явного и неявного преобразования между пользовательскими типами данных. Синтаксис объявителя операции преобразования типа:

```
implicit operator тип ( параметр )        // неявное преобразование
explicit operator тип ( параметр )        // явное преобразование
```

Эти операции выполняют преобразование из типа параметра в тип, указанный в заголовке операции. Одним из этих типов должен быть класс, для которого определяется операция. Таким образом, операции выполняют преобразование либо типа класса к другому типу, либо наоборот. Преобразуемые типы не должны быть связаны отношениями наследования. Примеры операций преобразования типа для класса **Monster**, описанного ранее:

```
public static implicit operator int( Monster m )
{
        return m.health;
}

public static explicit operator Monster( int h )
{
        return new Monster( h, 100, "FromInt" );
```

```
}
```

Ниже приведены примеры использования этих преобразований в программе. Не надо искать в них смысл, они просто иллюстрируют синтаксис:

```
Monster Masha = new Monster( 200, 200, "Masha" );
int i = Masha;                  // неявное преобразование
Masha = (Monster) 500;          // явное преобразование
```

Неявное преобразование выполняется автоматически:

- при присваивании объекта переменной целевого типа, как в примере;
- при использовании объекта в выражении, содержащем переменные целевого типа;
- при передаче объекта в метод на место параметра целевого типа;
- при явном приведении типа.

Явное преобразование выполняется при использовании операции приведения типа.

Все операции класса должны иметь разные сигнатуры. В отличие от других видов методов, для операций преобразования тип возвращаемого значения включается в сигнатуру, иначе нельзя было бы определять варианты преобразования данного типа в несколько других. Ключевые слова **implicit** и **explicit** в сигнатуру не включаются, следовательно, для одного и того же преобразования нельзя определить одновременно явную и неявную версию.

Неявное преобразование следует определять так, чтобы при его выполнении не возникала потеря точности и не генерировались исключения. Если эти ситуации возможны, преобразование следует описать как явное.

Листинги для работы на занятии

1. **ОПЕРАЦИИ унарные**:

```
class Monster
{
   // ЗДЕСЬ НАХОДИТСЯ ВСЯ РЕАЛИЗАЦИЯ КЛАССА,
   // КОТОРАЯ ЕСТЬ НА ДАННЫЙ МОМЕНТ

   // ОПЕРАЦИИ
   // Унарные
   public static Monster operator ++(Monster m)
   {
      Monster temp = new Monster();
      temp.health = m.health + 1;
      return temp;
   }

   public static Monster operator --(Monster m)
   {
      Monster temp = new Monster();
      temp.health = m.health - 1;
      return temp;
```

```
    }
}
```

2. ПРОВЕРКА РАБОТЫ: ОПЕРАЦИИ унарные

```
class Program
{
    static void Main(string[] args)
    {
        Monster Mascha = new Monster(70,50,"Маша");
        Mascha.Passport();
        Mascha++;
        Mascha.Passport();
        Mascha--;
        Mascha.Passport();
        Console.ReadKey();
    }
}
```

3. ОПЕРАЦИИ бинарные :

```
class Monster
{
    // ЗДЕСЬ НАХОДИТСЯ ВСЯ РЕАЛИЗАЦИЯ КЛАССА,
    // КОТОРАЯ ЕСТЬ НА ДАННЫЙ МОМЕНТ!!!
    // ОПЕРАЦИИ бинарные
    public static Monster operator +(Monster m, int k)
    {
        Monster temp = new Monster();
        temp.ammo = m.ammo + k;
        return temp;
    }
    public static Monster operator +(int k, Monster m)
    {
        Monster temp = new Monster();
        temp.ammo = m.ammo + k;
        return temp;
    }

}
```

4. ПРОВЕРКА РАБОТЫ: ОПЕРАЦИИ бинарные

```
class Program
{
    static void Main(string[] args)
    {
        Monster Mascha = new Monster(70,50,"Маша");
        Mascha.Passport();
        Monster Vasia = new Monster("Вася");
        Vasia.Passport();
        Mascha = Vasia + 10;
        Mascha.Passport();
        Monster Petya = 5 + Mascha;
```

```
      Petya.Passport();
   }
}
```

5. ОПЕРАЦИИ преобразования типа

```
class Monster
{
   // ЗДЕСЬ НАХОДИТСЯ ВСЯ РЕАЛИЗАЦИЯ КЛАССА,
   // КОТОРАЯ ЕСТЬ НА ДАННЫЙ МОМЕНТ!!!

   // ОПЕРАЦИИ
   public static implicit operator int(Monster m)
   {
      return m.health;
   }

   public static explicit operator Monster(int h)
   {
      return new Monster(h, 100, "FromInt");
   }
}
```

6. ПРОВЕРКА РАБОТЫ: ОПЕРАЦИИ преобразования типа:

```
class Program
{
   static void Main(string[] args)
   {
      Monster Mascha = new Monster(70,50,"Mascha");
      Mascha.Passport();
      int i = Mascha;              // неявное преобразование
      Mascha = (Monster)500;       // явное преобразование
      Mascha.Passport();
      Console.ReadKey();
   }
}
```

7. Полная реализация класса Monster:

```
class Monster
{
   // ПОЛЯ
   private string name; // имя монстра
   private int health;  // здоровье
   private int ammo;    // оружие

   public Monster()      // конструктор
   {
      this.name = "Noname";
      this.health = 100;
      this.ammo = 100;
   }
```

```
public Monster(string name) : this()
{
   this.name = name;
}

public Monster(int health, int ammo, string name)
{
   this.name = name;
   this.health = health;
   this.ammo = ammo;
}

// МЕТОДЫ
public string GetState()
{
   // Узнать состояние
   if (health >= 90) return "Здоров как бык";
   if (health >= 70) return "Легко ранен";
   if (health >= 50) return "Ранение средней тяжести";
   if (health >= 30) return "Тяжелое ранение";
   if (health >= 10) return "Критическое состояние";
   if (health > 0) return "Смертельное ранение";
   if (health <= 0) return "Пал смертью храбрых";
   else return "Умер по непонятной причине";
}

public void Passport()
{
   // Паспортные данные
   Console.WriteLine("У монстра {0} \t Здоровье={1} Оружие={2}
         Состояние:{3}", name, health, ammo, GetState());
}

public void Passport(string m)
{
   // Перегруженная функция "Паспортные данные"
   Console.WriteLine("У {4} {0} \t Здоровье={1} Оружие={2}
         Состояние:{3}", name, health, ammo, GetState(), m);
}

public void Passport(int k, string m)
{
   // Еще одна перегруженная функция "Паспортные данные"
   for (int i=0; i < k; i++)
      Console.WriteLine("У {4} {0} \t Здоровье={1} Оружие={2}
            Состояние:{3}",  name, health, ammo, GetState(), m);
}

// СВОЙСТВА
public int Health
{
   get { return health; }
   set
   {
      if (value > 100) health = 100;
```

```
      if (value < 0) health = 0;
      if (value <= 100 && value >= 0) health = value;
   }
}

public int Ammo
{
   get { return ammo; }
   set
   {
      if (value > 100) ammo = 100;
      if (value < 0) ammo = 0;
      if (value <= 100 && value >= 0) ammo = value;
   }
}

public string Name
{
   get { return name; }
   set { name = value; }
}

// ОПЕРАЦИИ
// Унарные операции
public static Monster operator ++(Monster m)
{
   Monster temp = new Monster(m.name);
   temp.health = m.health + 1;
   return temp;
}

public static Monster operator --(Monster m)
{
   Monster temp = new Monster(m.name);
   temp.health = m.health - 1;
   return temp;
}

// бинарные операции
public static Monster operator +(Monster m, int k)
{
   Monster temp = new Monster(m.name);
   temp.ammo = m.ammo + k;
   return temp;
}

public static Monster operator +(int k, Monster m)
{
   Monster temp = new Monster(m.name);
   temp.ammo = m.ammo + k;
   return temp;
}

//Операции преобразования типа
public static implicit operator int(Monster m)
```

```
    {
        return m.health;
    }

    public static explicit operator Monster(int h)
    {
        return new Monster(h, 100, "FromInt");
    }
}
```