Статические члены и методы класса

```
{
  public static int Val = 100;

  public static int ValDiv2()
  {
    return Val/2;
  }
}
```

```
class SDemo
{
  static void Main()
  {
Console.WriteLine("Исходное значение переменной"  + StaticDemo.Val);

StaticDemo.Val = 8;
```

Ошибка: Непосредственный доступ к нестатической переменной из статического метода недопустим.

```
class StaticError
{

  public int Denom = 3; // обычная переменная экземпляра
  public static int Val = 1024; // статическая переменная



static int ValDivDenom()
  {
    return Val/Denom; // не подлежит компиляции!
  }
}
```

Ошибка! Непосредственный вызов нестатического метода из статического метода недопустим.

```
using System;

class AnotherStaticError
{
  // Нестатический метод
  void NonStaticMeth()
  {
    Console.WriteLine("В методе NonStaticMeth().");
  }

static void staticMeth()
  {
    NonStaticMeth();   // не подлежит компиляции!
  }
}
```

Верно. Нестатический метод может быть вызван из статического метода по ссылке на объект.

```
class MyClass
{
  // Нестатический метод.
  void NonStaticMeth()
  {
    Console.WriteLine("В методе NonStaticMeth().");
  }

public static void staticMeth(MyClass ob)
  {
    ob.NonStaticMeth(); // все верно!
  }
}
```

Пример использования поля типа static для подсчета количества экземпляров существующих объектов.

```
  // Инкременировать подсчет при создании объекта.
  public CountInst()
  {
    count++;
  }

  // Декременировать подсчет при уничтожении объекта.
  ~CountInst()
  {
    count--;
  }


 public static int GetCount()
{     return count;   }
}

class CountDemo
{
```

Применение статического конструктора

```
class Cons
{
  public static int alpha;
  public int beta;

  // Статический конструктор
  static Cons()
  {
    alpha = 99;
    Console.WriteLine("Внутри статического конструктора.");
  }

  // Конструктор экземпляра
  public Cons()
  {
    beta = 100;
    Console.WriteLine("Внутри конструктора экземпляра.");
  }
}

class ConsDemo
{
```

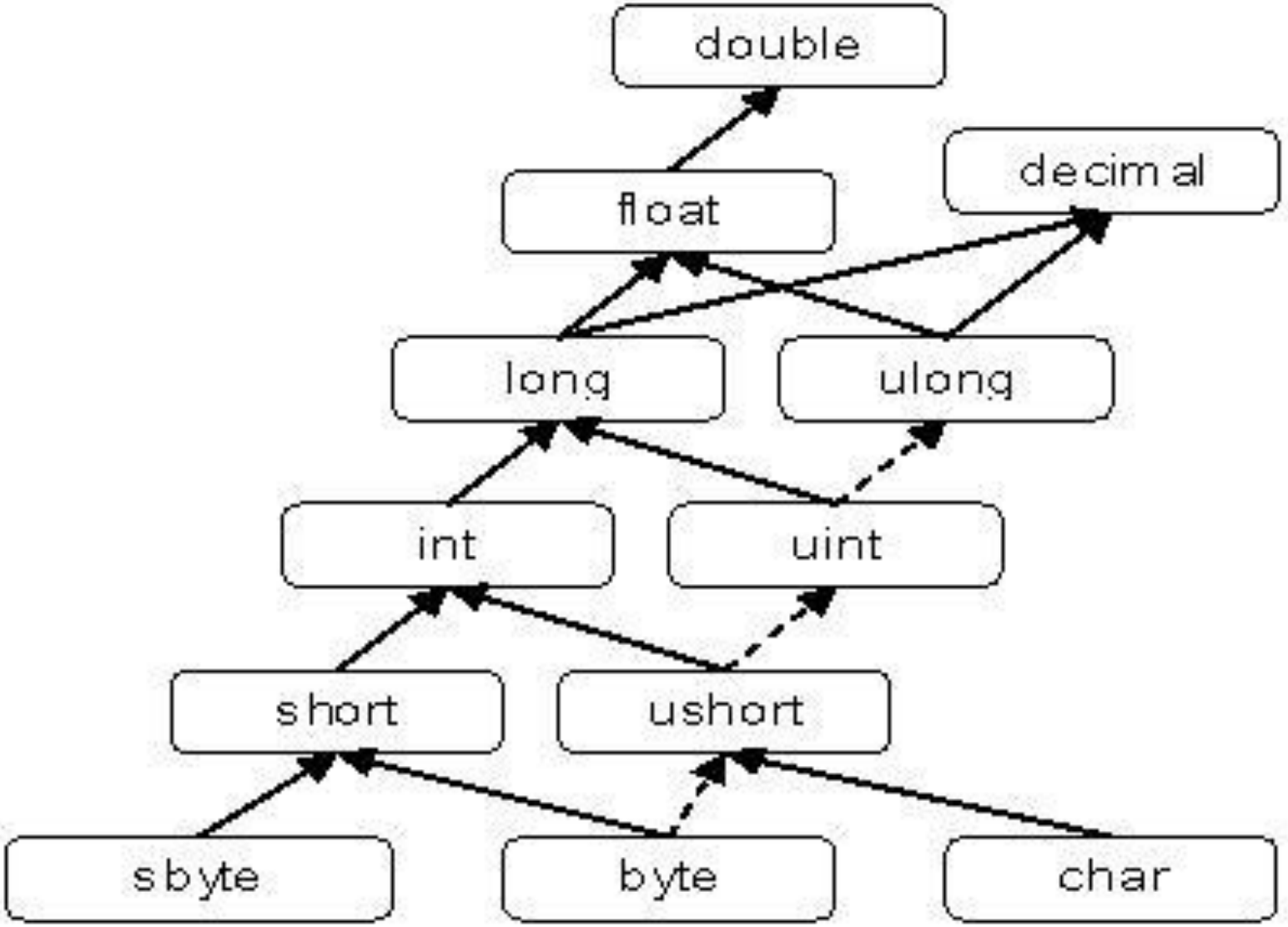
Перегрузка методов

```
 // Перегрузка метода OvlDemo с одним целочисленным параметром.
 public void OvlDemo(int a)
 { Console.WriteLine("Один параметр: " + a); }

 // Перегрузка метода OvlDemo с двумя целочисленными параметрами.
 public int OvlDemo(int a, int b)
 {
   Console.WriteLine("Два параметра: " + a + " " + b);
   return a + b; }

 // Перегрузка метода OvlDemo с двумя параметрами типа double.
 public double OvlDemo(double a, double b)
 {
   Console.WriteLine("Два параметра типа double: " +a + " "+ b);
   return a + b;
 }  }
 class OverloadDemo
{
 static void Main()
 {
   Overload ob = new Overload();
   int resI;
   double resD;

   // Вызвать все варианты метода OvlDemo().
   ob.OvlDemo();
   Console.WriteLine();
```

Ошибка!

```
public void OvlDemo(int a)
{
  Console.WriteLine("Один параметр: " + a);
}

public int OvlDemo(int a)
{
  Console.WriteLine("Один параметр: " + a);
  return a * a;
}
```

Автоматические преобразования типов параметров перегружаемых методов



Автоматические преобразования типов параметров перегружаемых методов

```
  public void MyMeth(double x)
  {
    Console.WriteLine("В методе MyMeth(double): " + x);
  }
}
```

```
class TypeConv
{
  static void Main()
  {
    Overload2 ob = new Overload2();

    int i = 10;
    double d = 10.1;

    byte b = 99;
```

Модификаторы ref и out при перегрузке методов

```
public void MyMeth(int x)
{
  Console.WriteLine("В методе MyMeth(int):  " + x);
}

public void MyMeth(ref int x)
{
  Console.WriteLine("В методе MyMeth(ref int):  " + x);
}
ob.MyMeth(i);
ob.MyMeth(ref i);
```

Неверно!

```
public void MyMeth(out int x) { //...
public void MyMeth(ref int x) { //...
```

Пример перегрузки конструктора

```
stck = new char[ob.stck.Length];
for(int i=0; i < ob.tos; i++)
    stck[i] = ob.stck[i];
tos = ob.tos;
}

// Поместить символы в стек.
public void Push(char ch
{  stck[tos] = ch;
  tos++;  }
//Извлечь символы из стека
public char Pop()
  {
   tos--;
   return stck[tos];
  }}
```

```
class StackDemo
{
  static void Main()
  {
    Stack stk1 = new Stack(10);
    char ch;
```

Вызов перегружаемого конструктора с помощью ключевого слова this

имя_конструктора
(*список_параметров1*):
this(*список_параметров2*)
{
// ... Тело конструктора, которое может быть пустым.
}

Перегрузка различных категорий символов операций

+, –, !, ~, ++, —, true, false	Унарные символы операций, допускающие перегрузку. true и false также являются операциями
+, –, *, /, %, &, |, ^, <<, >>	Бинарные символы операций, допускающие перегрузку
==, !=, <, >, <=, >=	Операции сравнения перегружаются парами
&&, ||	Условные логические операции моделируются с использованием ранее переопределенных операций & и |
[]	Операции доступа к элементам массивов моделируются за счет индексаторов
()	Операции преобразования реализуются с использованием ключевых слов implicit и explicit
+=, –=, *=, /=, %=, &=, =, ^=, <<=, >>=	Операции не перегружаются, по причине невозможности перегрузки операции присвоения, однако вы получаете их автоматически, перегружая соответствующую бинарную операцию
=, ., ?:, –>, new, is, sizeof, typeof	Операции, не подлежащие перегрузке

Операторный метод

// Общая форма перегрузки унарного оператора

```
public static возвращаемый_тип operator op (тип_параметра операнд)
{
    // операции
}
```

// Общая форма перегрузки бинарного оператора

```
public static возвращаемый_тип operator op (тип_параметра1 операнд1,
                                             тип_параметра2 операнд2)
{
    // операции
}
```

Перегрузка бинарных операторов

```
    public MyArr(int x, int y, int z)
    { this.x = x; this.y = y; this.z = z; }
// Перегружаем бинарный оператор +
    public static MyArr operator +(MyArr obj1, MyArr obj2)
    {
        MyArr arr = new MyArr();
        arr.x = obj1.x + obj2.x;
        arr.y = obj1.y + obj2.y;
        arr.z = obj1.z + obj2.z;
        return arr;
    }
  // Перегружаем бинарный оператор -
    public static MyArr operator -(MyArr obj1, MyArr obj2)
    {
        MyArr arr = new MyArr();
        arr.x = obj1.x - obj2.x;
        arr.y = obj1.y - obj2.y;
        arr.z = obj1.z - obj2.z;
        return arr;
    }
}
class Program
{
    static void Main(string[] args)
    {
        MyArr Point1 = new MyArr(1, 12, -4);
        MyArr Point2 = new MyArr(0, -3, 18);
```

Перегрузка унарных операторов

```
// Перегружаем унарный оператор
    public static MyArr operator -(MyArr obj1)
    {
      MyArr arr = new MyArr();
    arr.x = -obj1.x; arr.y = -obj1.y; arr.z = -obj1.z;
      return arr;
    }

 // Перегружаем унарный оператор ++
    public static MyArr operator ++(MyArr obj1)
    {
      obj1.x += 1; obj1.y += 1;  obj1.z +=1;
      return obj1;  }

// Перегружаем унарный оператор --
    public static MyArr operator --(MyArr obj1)
    {
      obj1.x -= 1;   obj1.y -= 1;  obj1.z -= 1;
      return obj1;  }   }
  class Program
  {
    static void Main(string[] args)
    {
      MyArr Point1 = new MyArr(1, 12, -4);
      MyArr Point2 = new MyArr(0, -3, 18);

      Point3 = -Point1;
      Console.WriteLine("-Point1 = " + Point3.x          + " " + Point3.y + " " + Point3.z);
```

Перегрузка операторного метода

```
public static string operator +(MyArr obj1, string s)
{
   return s + " " + obj1.x + " " + obj1.y + " " + obj1.z;
}
public static MyArr operator +(MyArr obj1, int i)
{
    MyArr arr = new MyArr();
    arr.x = obj1.x + i;
    arr.y = obj1.y + i;
    arr.z = obj1.z + i;
    return arr;
 }
...
string s = Point2 + "Координаты точки Point2:";
Console.WriteLine(s);
```

Перегрузка операторов отношения

```
    public int x, y, z;
    public MyArr(int x, int y, int z)
    {   this.x = x;  this.y = y;  this.z = z;  }

// Перегружаем логический оператор ==
    public static bool operator ==(MyArr obj1, MyArr obj2)
    {
        if ((obj1.x == obj2.x) && (obj1.y == obj2.y) && (obj1.z == obj2.z))
            return true;
        return false;
    }
  // Теперь обязательно нужно перегрузить логический оператор !=
    public static bool operator !=(MyArr obj1, MyArr obj2)
    {
        if ((obj1.x != obj2.x) || (obj1.y != obj2.y) || (obj1.z != obj2.z))
            return true;
        return false;
    }
  }
class Program
{
    static void Main(string[] args)
    {
        MyArr myObject1 = new MyArr
            (4, 5, 12);
        MyArr myObject2 = new MyArr
```

Перегрузка операторов true и false

Общая форма перегрузки

```
public static bool operator true(тип_параметра операнд)
{
// Возврат логического значения true или false.
}

public static bool operator false(тип_параметра операнд)
{
// Возврат логического значения true или false.
}
```

Пример перегрузки false и true

```
// Перегружаем оператор false
      public static bool operator false(MyArr obj)
      {
         if ((obj.x <= 0) || (obj.y <= 0) || (obj.z <= 0))
            return true;
         return false;
      }

      // Обязательно перегружаем оператор true
      public static bool operator true(MyArr obj)
      {
         if ((obj.x > 0) && (obj.y > 0) && (obj.z > 0))
            return true;
         return false;
      }
if (myObject1)
            Console.WriteLine("Все координаты объекта myObject1
    положительны");
```

Перегрузка логических операторов

```
// Перегружаем логический оператор &
public static bool operator &(MyArr obj1, MyArr obj2)
{
  if (((obj1.x > 0) && (obj1.y > 0) &&
(obj1.z > 0))
        & ((obj2.x > 0) && (obj2.y > 0) &&
(obj2.z > 0)))
    return true;
    return false;
}

// Перегружаем логический оператор !
public static bool operator !(MyArr obj1)
{
   if ((obj1.x > 0) && (obj1.y > 0) &&
(obj1.z > 0))
    return false;
return true;
}
```

```
}
// Перегружаем логический оператор |
public static MyArr operator |(MyArr obj1, MyArr obj2)
{
    if (((obj1.x > 0) || (obj1.y > 0) || (obj1.z > 0))
        | ((obj2.x > 0) || (obj2.y > 0) || (obj2.z > 0)))
        return obj1;
    return new MyArr(0, 0, 0);
}
// Перегружаем оператор true
public static bool operator true(MyArr obj)
{
    if ((obj.x > 0) || (obj.y > 0) || (obj.z > 0))
        return true;
    return false;
}
// Перегружаем оператор false
public static bool operator false(MyArr obj)
{
    if ((obj.x > 0) && (obj.y > 0) && (obj.z > 0))
        return false;
    return true;
}
// Вспомогательный метод
public static bool And(MyArr obj1, MyArr obj2)
{
    if (obj1 && obj2)
        return true;
    return false;
}
}
```

Операторы преобразования

public static explicit operator *целевой_тип* (*исходный_тип* v)

{return *значение*;}

public static implicit operator *целевой_тип* (*исходный_тип* v)

{return *значение*;}

Неявное преобразование

```
public static implicit operator int(MyArr op1)
{
  return op1.x * op1.y * op1.z;
}
…

MyArr a = new MyArr (1, 2, 3);
int i;
i = a; // преобразовать в тип int
```

Явное преобразование

```
public static explicit operator int(MyArr  op1)
  {
    return op1.x * op1.y * op1.z;
  }


…
MyArr a = new MyArr (1, 2, 3);
int i;
i = (int)a * 2 - (int)b; // явно требуется приведение типов
```

Явное преобразование

```
   {
      public string Name, Family;
      public byte Age;
      public UserInfo(string Name,
string Family, byte Age)
      {
         this.Name = Name;
         this.Family = Family;
         this.Age = Age;
      }
     // Явное преобразование типа UserInfo к string
      public static explicit operator string(UserInfo obj)
      {
        return "Информация о пользователе: " + obj.Name+" "+obj.Family+"("+obj.Age+"
лет)";
      }
   }


class Program
   {
      static void Main(string[] args)
      {
         UserInfo ui1 = new UserInfo
             ("Alexandr", "Erohin", 26);
```