

# Курс «С#. Программирование на языке высокого уровня»

---

Павловская Т.А.

# Лекция 3. Переменные, операции, выражения

---

**Правила описания переменных и именованных констант, основные операции языка и их приоритеты, правила записи выражений, введение в обработку исключительных ситуаций.**

# Структура простейшей программы на C#

```
using System;
namespace A
{
    class Class1
    {
        static void Main()
        {
            // описания и операторы, например:
            Console.WriteLine("Превед медвед ");
        }

        // описания
    }
}
```

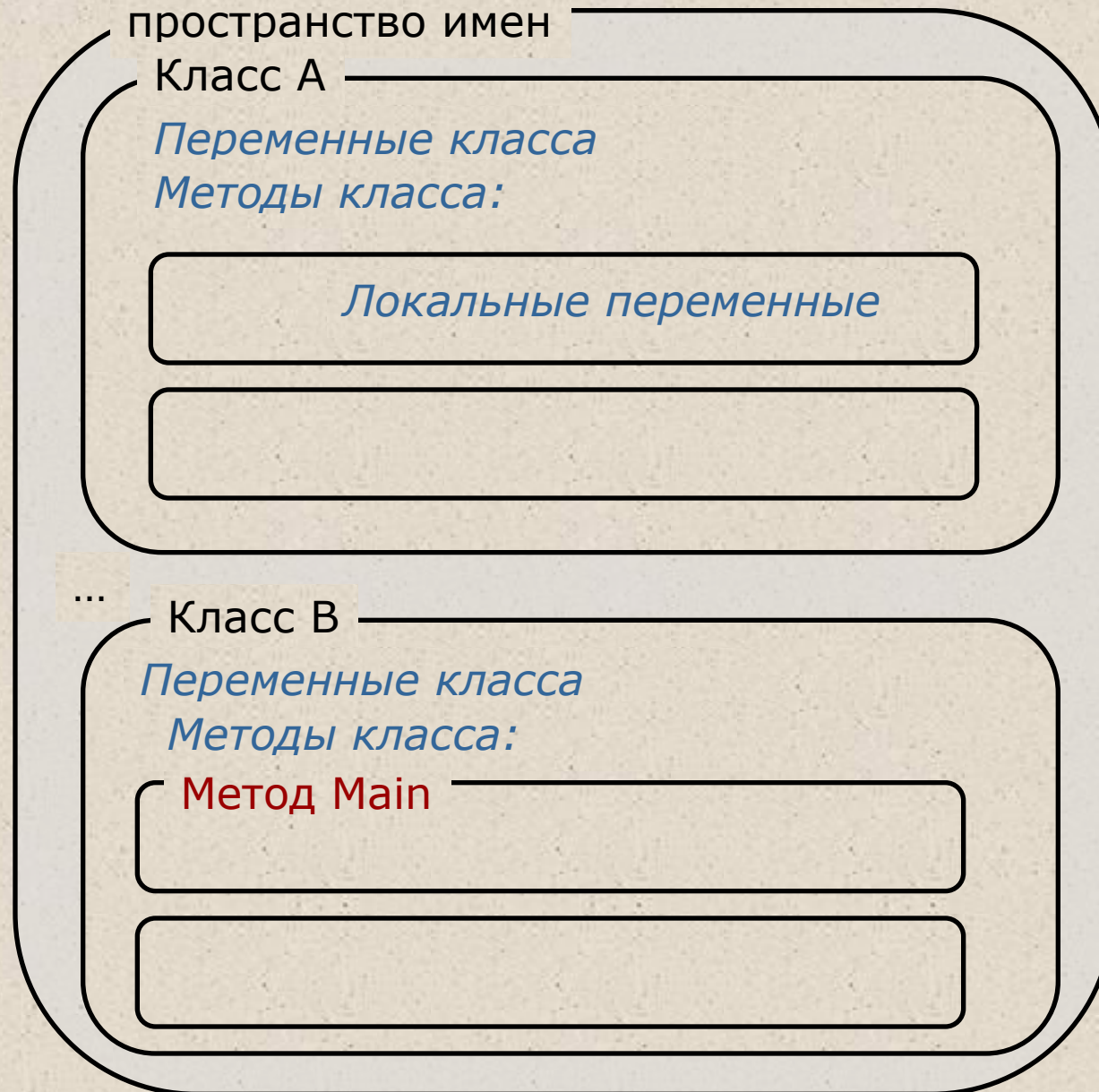
# Переменные

- *Переменная* — это величина, которая во время работы программы может изменять свое значение.
- Все переменные, используемые в программе, должны быть описаны.
- Для каждой переменной задается ее *имя и тип*:

```
int    number;  
float  x, y;  
char   option;
```

Тип переменной выбирается исходя из диапазона и требуемой точности представления данных.

# Общая структура программы на С#



# Область действия и время жизни переменных

- Переменные описываются внутри какого-л. блока (класса, метода или блока внутри метода)
  - **Блок** — это код, заключенный в фигурные скобки. Основное назначение блока — группировка операторов.
  - Переменные, описанные непосредственно внутри класса, называются **полями класса**.
  - Переменные, описанные внутри метода класса, называются **локальными переменными**.
- **Область действия переменной** - область программы, где можно использовать переменную.
- Область действия переменной начинается в точке ее описания и длится до конца блока, внутри которого она описана.
- **Время жизни**: переменные создаются при входе в их область действия (блок) и уничтожаются при выходе.

# Инициализация переменных

- При объявлении можно присвоить переменной начальное значение (инициализировать).

```
int number = 100;  
float x = 0.02;  
char option = 'ю';
```

При инициализации можно использовать не только константы, но и выражения — главное, чтобы на момент описания они были вычислимыми, например:

```
int b = 1, a = 100;  
int x = b * a + 25;
```

- Поля класса инициализируются «значением по умолчанию» (0 соответствующего типа).
- Инициализация локальных переменных возлагается на программиста. Рекомендуется всегда инициализировать переменные при описании.

# Пример описания переменных

```
using System;
namespace CA1
{
    class Class1
    {
        static void Main()
        {
            int    i = 3;
            double y = 4.12;
            decimal d = 600m;
            string  s = "Вася";

        }
    }
}
```



# Именованные константы

Вместо значений констант можно (и нужно!) использовать в программе их имена.

Это облегчает читабельность программы и внесение в нее изменений:

```
const float weight = 61.5;  
const int   n       = 10;  
const float g       = 9.8;
```

# Выражения

- *Выражение* — правило вычисления значения.
- В выражении участвуют *операнды*, объединенные знаками операций.
- Операндами выражения могут быть константы, переменные и вызовы функций.
- Операции выполняются в соответствии с *приоритетами*.
- Для изменения порядка выполнения операций используются *круглые скобки*.
- Результатом выражения всегда является значение определенного типа, который определяется типами операндов.
- Величины, участвующие в выражении, должны быть *совместимых типов*.

■ `t + Math.Sin(x)/2 * x`

результат имеет  
вещественный тип

■ `a <= b + 2`

результат имеет  
логический тип

■ `x > 0 && y < 0`

результат имеет  
логический тип

# Ассоциативность выражений

- Слева направо

- $a + b - c + d$

- $(( ( a + b ) - c ) + d )$

- $a * b / c * d$

- $(( ( a * b ) / c ) * d )$

$$\frac{a * b}{c} * d$$

- Справа налево

- $a = b = c = d$

- $( a = ( b = ( c = d ) ) )$

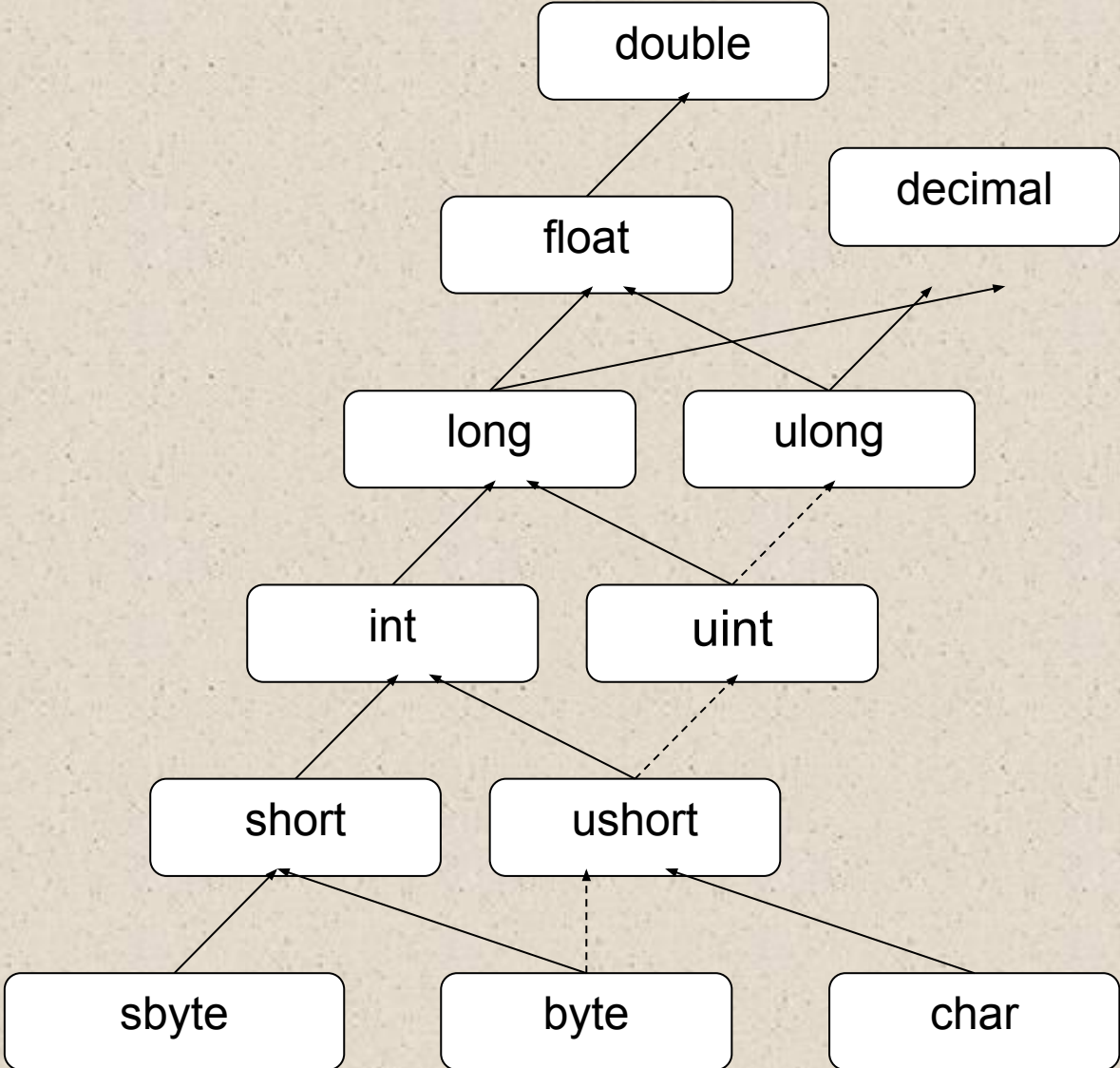
# Приоритеты операций C#

1. Первичные `()`, `[]`, `++`, `--`, `new`, ...
2. Унарные `~`, `!`, `++`, `--`, `-`, ...
3. Типа умножения (мультипликативные) `*`, `/`, `%`
4. Типа сложения (аддитивные) `+`, `-`
5. Сдвига `<<`, `>>`
6. Отношения и проверки типа `<`, `>`, `is`, ...
7. Проверки на равенство `==`, `!=`
8. Поразрядные логические `&`, `^`, `|`
9. Условные логические `&&`, `||`
10. Условная `?:`
11. Присваивания `=`, `*=`, `/=`, ...

# Тип результата выражения

- Если операнды, входящие в выражение, одного типа, и операция для этого типа определена, то результат выражения будет иметь тот же тип.
- Если операнды разного типа и (или) операция для этого типа не определена, перед вычислениями автоматически выполняется **преобразование типа** по правилам, обеспечивающим приведение более коротких типов к более длинным для сохранения значимости и точности.
- Автоматическое (**неявное**) преобразование возможно не всегда, а только если при этом не может случиться потеря значимости.
- Если неявного преобразования из одного типа в другой не существует, программист может задать **явное** преобразование типа с помощью операции **(тип)х**.

# Неявные арифметические преобразования типов в C#



# Введение в исключения

- При вычислении выражений могут возникнуть ошибки (переполнение, деление на ноль).
- В C# есть механизм *обработки исключительных ситуаций (исключений)*, который позволяет избегать аварийного завершения программы.
- Если в процессе вычислений возникла ошибка, система сигнализирует об этом с помощью *выбрасывания (генерирования) исключения*.
- Каждому типу ошибки соответствует свое исключение. Исключения являются классами, которые имеют общего предка — класс Exception, определенный в пространстве имен System.
- Например, при делении на ноль будет выброшено исключение DivideByZeroException, при переполнении — исключение OverflowException.

# Инкремент и декремент

```
using System;
namespace CA1
{
    class C1
    {
        static void Main()
        {
            int x = 3, y = 3;
            Console.Write( "Значение префиксного выражения: " );
            Console.WriteLine( ++x );
            Console.Write( "Значение x после приращения: " );
            Console.WriteLine( x );

            Console.Write( "Значение постфиксного выражения: " );
            Console.WriteLine( y++ );
            Console.Write( "Значение y после приращения: " );
            Console.WriteLine( y );
        }
    }
}
```

Результат работы программы:

```
Значение префиксного
выражения: 4
Значение x после приращения:
4
Значение постфиксного
выражения: 3
```

Значение y после приращения:



# Операция new

Операция new служит для создания нового объекта. Формат операции:

**new тип ( [ аргументы ] )**

С помощью этой операции можно создавать объекты как ссылочных, так и значимых типов, например:

```
object z = new object();
```

```
int i = new int();           // то же самое, что int i = 0;
```

# Операции отрицания

```
using System;
namespace ConsoleApplication1
{
    class Class1
    {
        static void Main()
        {
            sbyte a = 3, b = -63, c = 126;
            bool d = true;
            Console.WriteLine( -a ); // Результат -3
            Console.WriteLine( -c ); // Результат -126
            Console.WriteLine( !d ); // Результат false
            Console.WriteLine( ~a ); // Результат -4
            Console.WriteLine( ~b ); // Результат 62
            Console.WriteLine( ~c ); // Результат -127
        }
    }
}
```

# Явное преобразование типа

- `long b = 300;`
- `int a = (int) b; // данные не теряются`
- `byte d = (byte) a; // данные теряются`

# Умножение

- *Операция умножения* (\*) возвращает результат перемножения двух операндов.
- Стандартная операция умножения определена для типов `int`, `uint`, `long`, `ulong`, `float`, `double` и `decimal`.
- К величинам других типов ее можно применять, если для них возможно неявное преобразование к этим типам. Тип результата операции равен «наибольшему» из типов операндов, но не менее `int`.
- Если оба операнда *целочисленные* или типа `decimal` и результат операции слишком велик для представления с помощью заданного типа, генерируется исключение `System.OverflowException`

# Результаты вещественного умножения

*	$+y$	$-y$	$+0$	$-0$	$+\infty$	$-\infty$	NaN
$+x$	$+z$	$-z$	$+0$	$-0$	$+\infty$	$-\infty$	NaN
$-x$	$-z$	$+z$	$-0$	$+0$	$-\infty$	$+\infty$	NaN
$+0$	$+0$	$-0$	$+0$	$-0$	NaN	NaN	NaN
$-0$	$-0$	$+0$	$-0$	$+0$	NaN	NaN	NaN
$+\infty$	$+\infty$	$-\infty$	NaN	NaN	$+\infty$	$-\infty$	NaN
$-\infty$	$-\infty$	$+\infty$	NaN	NaN	$-\infty$	$+\infty$	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

# Пример

```
using System;
namespace ConsoleApplication1
{
    class Class1
    {
        static void Main()
        {
            int x = 11, y = 4;
            float z = 4;
            Console.WriteLine( z * y );           // Результат 16
            Console.WriteLine( z * 1e308 );      // Рез. "бесконечность"
            Console.WriteLine( x / y );          // Результат 2
            Console.WriteLine( x / z );          // Результат 2,75
            Console.WriteLine( x % y );          // Результат 3
            Console.WriteLine( 1e-324 / 1e-324 ); // Результат NaN
        }
    }
}
```

# Операции сдвига

- *Операции сдвига* (<< и >>) применяются к целочисленным операндам. Они сдвигают двоичное представление первого операнда влево или вправо на количество двоичных разрядов, заданное вторым операндом.
- При *сдвиге влево* (<<) освободившиеся разряды обнуляются. При *сдвиге вправо* (>>) освободившиеся биты заполняются нулями, если первый операнд беззнакового типа, и знаковым разрядом в противном случае.
- Стандартные операции сдвига определены для типов `int`, `uint`, `long` и `ulong`.

# Пример

```
using System;
namespace ConsoleApplication1
{
    class Class1
    {
        static void Main()
        {
            byte a = 3, b = 9;
            sbyte c = 9, d = -9;
            Console.WriteLine( a << 1 );           // Результат 6
            Console.WriteLine( a << 2 );           // Результат 12
            Console.WriteLine( b >> 1 );           // Результат 4
            Console.WriteLine( c >> 1 );           // Результат 4
            Console.WriteLine( d >> 1 );           // Результат -5
        }
    }
}
```

00001100



# Операции отношения и проверки на равенство

- *Операции отношения* ( $<$ ,  $<=$ ,  $>$ ,  $>=$ ,  $==$ ,  $!=$ ) сравнивают первый операнд со вторым.
- Операнды должны быть арифметического типа.
- Результат операции — логического типа, равен true или false.

$x == y$  -- true, если  $x$  равно  $y$ , иначе false

$x != y$  -- true, если  $x$  не равно  $y$ , иначе false

$x < y$  -- true, если  $x$  меньше  $y$ , иначе false

$x > y$  -- true, если  $x$  больше  $y$ , иначе false

$x <= y$  -- true, если  $x$  меньше или равно  $y$ , иначе false

$x >= y$  -- true, если  $x$  больше или равно  $y$ , иначе false

# Условные логические операции

```
using System;
namespace ConsoleApplication1
{
    class Class1
    {
        static void Main()
        {
            Console.WriteLine( true && true );    // Результат true
            Console.WriteLine( true && false );    // Результат false
            Console.WriteLine( true || true );     // Результат true
            Console.WriteLine( true || false );    // Результат true
        }
    }
}
```

# Условная операция

## ■ **операнд\_1 ? операнд\_2 : операнд\_3**

Первый операнд — выражение, для которого существует неявное преобразование к логическому типу.

Если результат вычисления первого операнда равен true, то результатом будет значение второго операнда, иначе — третьего операнда.

```
using System;
namespace ConsoleApplication1
{
    class Class1
    {
        static void Main()
        {
            int a = 11, b = 4;
            int max = b > a ? b : a;
            Console.WriteLine( max );    // Результат 11
        }
    }
}
```

# Операция присваивания

Присваивание – это замена старого значения переменной на новое. Старое значение стирается бесследно.

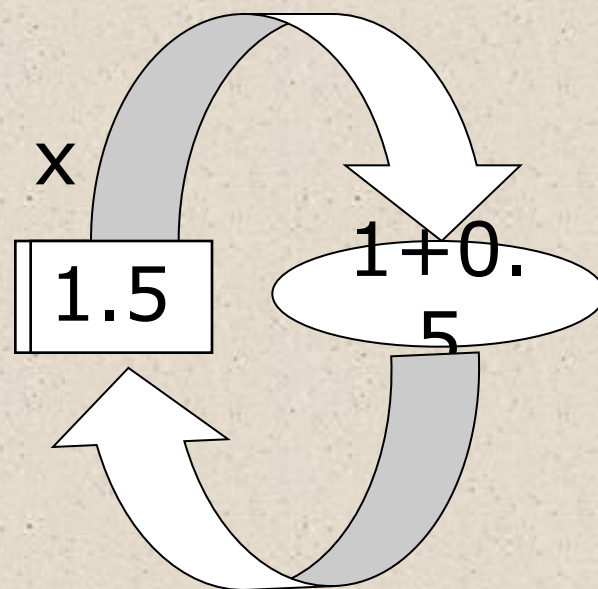
Операция может использоваться в программе как законченный оператор.

переменная = выражение

$a = b + c;$

$x = 1;$

$x = x + 0.5;$



Правый операнд операции присваивания должен иметь **неявное преобразование** к типу левого операнда, например:

**вещественная переменная = целое выражение;**

# Сложное присваивание в C#

- $x += 0.5;$       соответствует       $x = x + 0.5;$
- $x *= 0.5;$       соответствует       $x = x * 0.5;$
  
- $a \% = 3;$       соответствует       $a = a \% 3;$
- $a << = 2;$       соответствует       $a = a << 2;$

и т.п.