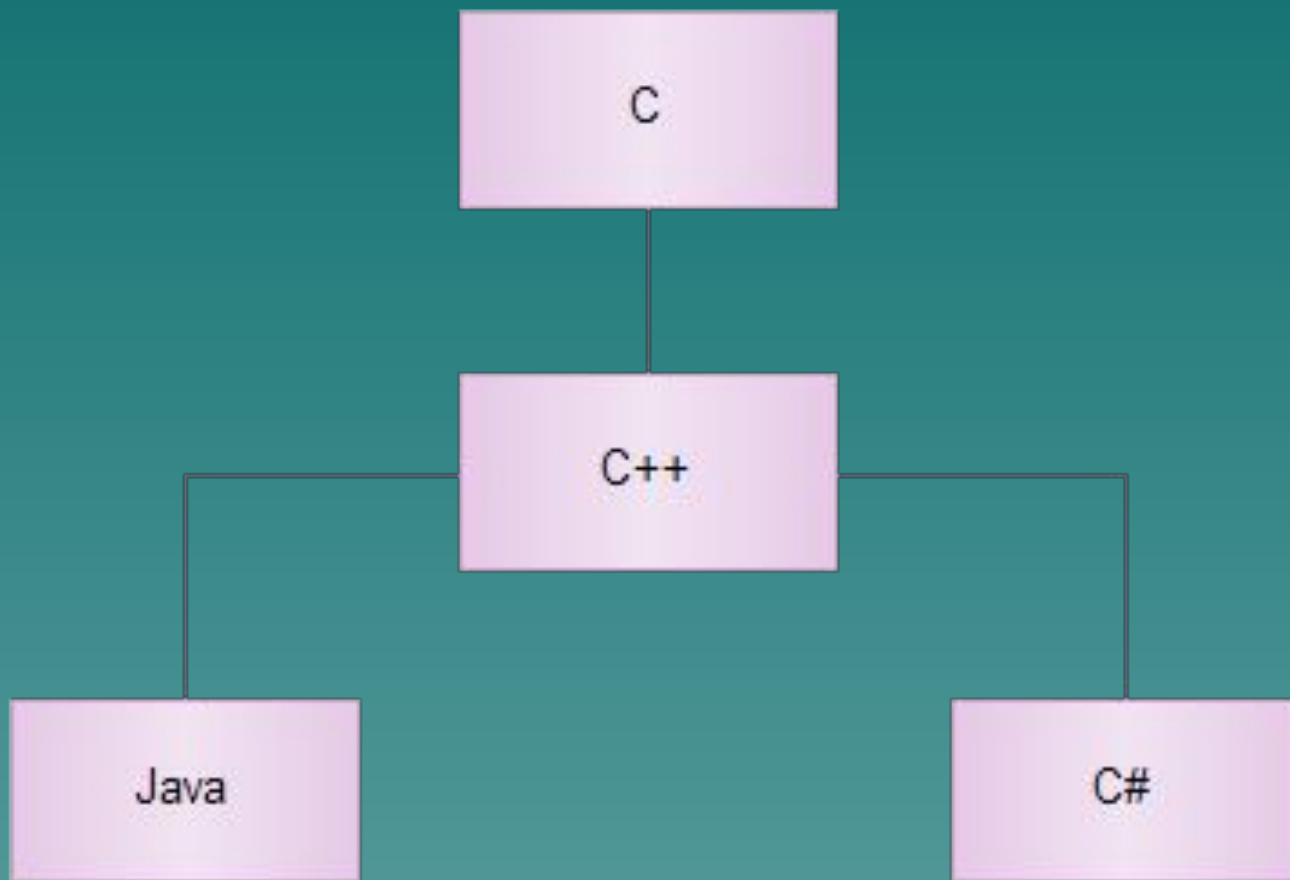


# Язык С#



# Создание C#



# Связь C# со средой .NET Framework

C# спроектирован и разработан специально для применения с .NET Framework.

**Назначение .NET Framework** — служить средой для поддержки разработки и выполнения сильно распределенных компонентных приложений. Она обеспечивает совместное использование разных языков программирования, а также безопасность, переносимость программ и общую модель программирования для платформы Windows.

# Базовые функциональные возможности платформы .NET:

- ◆ Возможность обеспечения взаимодействия с существующим программным кодом
- ◆ Поддержка для многочисленных языков программирования
- ◆ Полная интеграция языков
- ◆ Усовершенствованная поддержка для создания динамических веб-страниц
- ◆ Эффективный доступ к данным
- ◆ Установка с нулевым воздействием
- ◆ Visual Studio 2010

# Составляющие .NET

- ◆ **Common Language Runtime (CLR)**
- ◆ **Common Type System (CTS)**
- ◆ **Common Language Specification (CLS).**

# CLR

**CLR** - среда выполнения приложений  
**.NET**.

Именно она отвечает за  
использование типов, управление  
памятью, регистрацию объектов и  
многое другое.

# CTS

**CTS** - стандартная система типов, которые используют практически все **.NET** языки программирования.

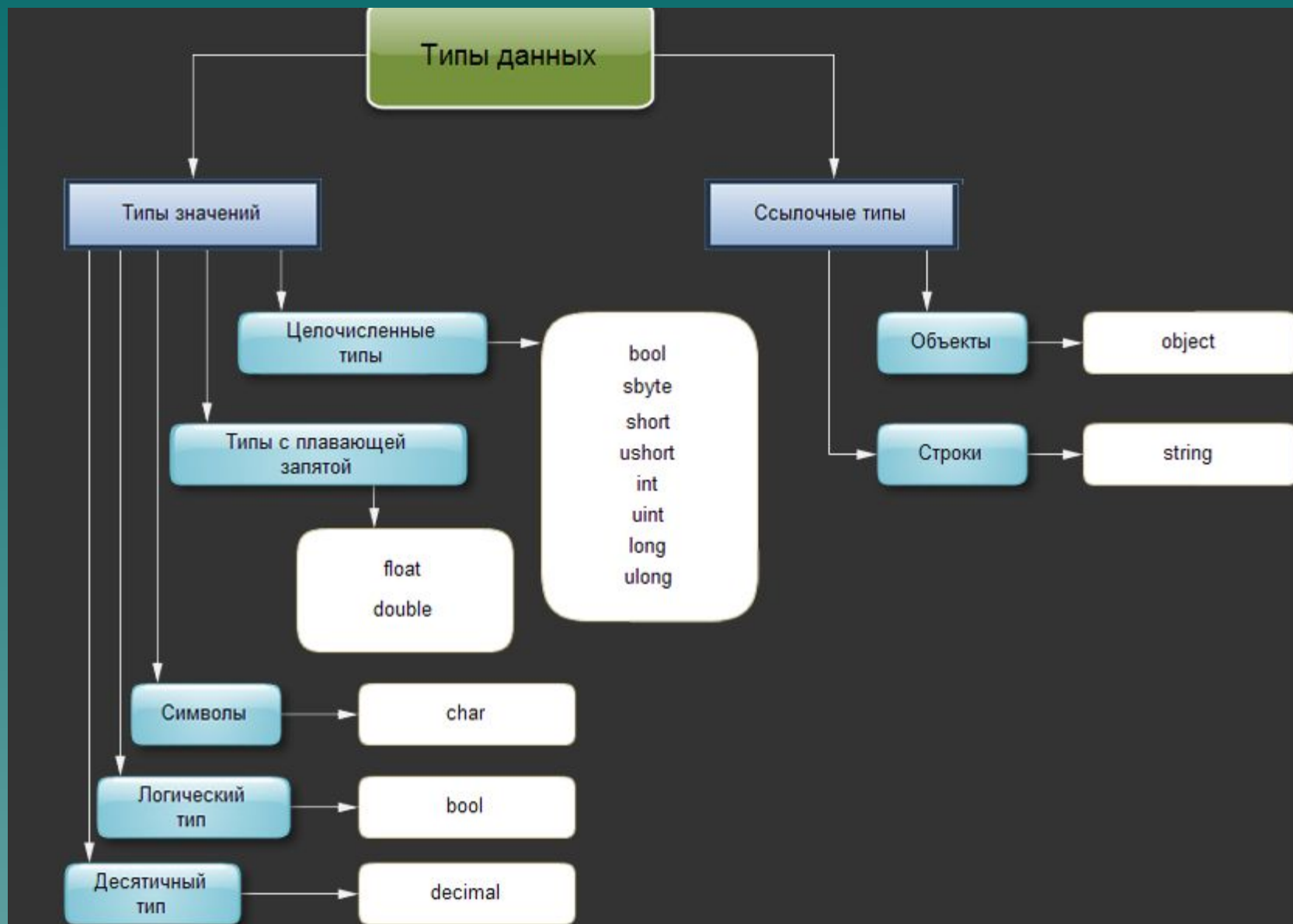
Представляет собой формальную спецификацию, в которой описано то, как должны быть определены типы для того, чтобы они могли обслуживаться в CLR-среде.

# CLS

**CLS** - набор правил, следуя которым новые типы данных будут полностью совместимы с другими **.NET** языками. Если данные правила не соблюдать, то возможно, приложение и будет работать, но с совместимостью у него будут проблемы.



# Типы данных в C#



# Целочисленные типы

Тип	Тип CTS	Разрядность в битах	Диапазон
byte	System.Byte	8	0:255
sbyte	System.SByte	8	-128:127
short	System.Int16	16	-32768 : 32767
ushort	System.UInt16	16	0 : 65535
int	System.Int32	32	-2147483648 : 2147483647
uint	System.UInt32	32	0 : 4294967295
long	System.Int64	64	-9223372036854775808 : 9223372036854775807
ulong	System.UInt64	64	0 : 18446744073709551615

# Вещественные типы

## Типы с плавающей точкой

Тип	Тип CTS	Разрядность в битах	Диапазон
float	System.Single	32	от 5E-45 до 3,4E+38
double	System.Double	64	от 5E-324 до 1,7E+308

Если нецелочисленное значение жестко кодируется в исходном тексте (например, 12.3), то обычно компилятор предполагает, что подразумевается значение типа double. Если значение необходимо специфицировать как float, потребуется добавить к нему символ F (или f)

## Десятичный тип

Тип	Тип CTS	Разрядность в битах	Диапазон
decimal	System.Decimal	128	от 1E-28 до 7,9E+28

# СИМВОЛЫ

Тип	Тип CTS	Разрядность в битах	Диапазон
char	System.Char	16	символы представлены <i>уникодом (Unicode)</i>

В уникоде набор символов представлен настолько широко, что он охватывает символы практически из всех естественных языков на свете. При этом стандартный набор символов в 8-разрядном коде ASCII является подмножеством уникода в пределах от 0 до 127.

Для того чтобы присвоить значение символьной переменной, достаточно заключить это значение (т.е. символ) в *одинарные кавычки*

# Логический тип данных

**Тип `bool`** представляет два логических значения: "истина" и "ложь". Эти логические значения обозначаются в C# зарезервированными словами *true* и *false* соответственно. Следовательно, переменная или выражение типа `bool` будет принимать одно из этих логических значений. Кроме того, в C# не определено взаимное преобразование логических и целых значений. Например, `1` не преобразуется в значение `true`, а `0` — в значение `false`.

# Преобразования типов

## Автоматическое преобразование типов

Когда данные одного типа присваиваются переменной другого типа, **неявное преобразование** типов происходит автоматически при следующих условиях:

- ♦ оба типа совместимы
- ♦ диапазон представления чисел целевого типа шире, чем у исходного типа

Если оба эти условия удовлетворяются, то происходит **расширяющее преобразование**. Например, тип `int` достаточно крупный, чтобы вмещать в себя все действительные значения типа `byte`, а кроме того, оба типа, `int` и `byte`, являются совместимыми целочисленными типами, и поэтому для них вполне возможно неявное преобразование.

Числовые типы, как целочисленные, так и с плавающей точкой, вполне совместимы друг с другом для выполнения расширяющих преобразований.

# Преобразования типов

## Приведение несовместимых типов

*Приведение* — это команда компилятору преобразовать результат вычисления выражения в указанный тип. А для этого требуется явное преобразование типов. Ниже приведена общая форма приведения типов:

*(целевой\_тип) выражение*

Здесь *целевой\_тип* обозначает тот тип, в который желательно преобразовать указанное выражение.

Если приведение типов приводит к *сужающему преобразованию*, то часть информации может быть потеряна. Например, в результате приведения типа `long` к типу `int` часть информации потеряется, если значение типа `long` окажется больше диапазона представления чисел для типа `int`, поскольку старшие разряды этого числового значения отбрасываются.

# Класс System.Convert

В пространстве имен System имеется **класс Convert**, который тоже может применяться для расширения и сужения данных:

```
byte sum = Convert.ToByte(var1 + var2);
```

Одно из преимуществ подхода с применением класса System.Convert связано с тем, что он позволяет выполнять преобразования между типами данных нейтральным к языку образом (например, синтаксис приведения типов в Visual Basic полностью отличается от предлагаемого для этой цели в C#).



# ОПЕРАЦИИ И ОПЕРАТОРЫ



# Арифметические операции

Оператор	Действие
+	Сложение
-	Вычитание, унарный минус
*	Умножение
/	Деление
%	Деление по модулю
--	Декремент
++	Инкремент

# Арифметические операции

## Операторы инкремента и декремента

Оператор инкремента ( $++$ ) увеличивает свой операнд на 1, а оператор декремента ( $--$ ) уменьшает операнд на 1.

Следовательно, операторы:

$x++$ ;  $x--$ ;

равнозначны операторам:

$x = x + 1$ ;  $x = x - 1$ ;

Следует, однако, иметь в виду, что в инкрементной или декрементной форме значение переменной  $x$  вычисляется только один, а не два раза. В некоторых случаях это позволяет повысить эффективность выполнения программы.

# Операции отношения и логические операции

Оператор	Значение
==	Равно
!=	Не равно
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно

# Операции отношения и логические операции

Оператор	Значение
&	И
	ИЛИ
^	Исключающее ИЛИ
&&	Укороченное И
	Укороченное ИЛИ
!	НЕ

# Оператор присваивания

Оператор присваивания обозначается одиночным знаком равенства (=). В C# оператор присваивания действует таким же образом, как и в других языках программирования. Ниже приведена его общая форма:

*имя\_переменной = выражение*

Здесь *имя\_переменной* должно быть совместимо с типом выражения. У оператора присваивания имеется одна интересная особенность, о которой вам будет полезно знать: он позволяет создавать цепочку операций присваивания. Рассмотрим следующий фрагмент кода:

```
int x, y, z;  
x = y = z = 10; // присвоить значение 10 переменным x, y и z
```

# Составные операторы присваивания

Оператор	Аналог (выражение из вышеуказанного примера)
<code>+=</code>	<code>x = x + 1;</code>
<code>-=</code>	<code>x = x - 1;</code>
<code>*=</code>	<code>x = x*1;</code>
<code>/=</code>	<code>x = x/1;</code>
<code>%=</code>	<code>x = x%1;</code>

# Условные операторы

## Оператор `if`

Для организации условного ветвления язык C# унаследовал от C и C++ конструкцию `if...else`. Ее синтаксис должен быть интуитивно понятен для любого, кто программировал на процедурных языках:

*if (условие)*

*оператор (операторы)*

*else*

*оператор (операторы)*

Если по каждому из условий нужно выполнить более одного оператора, эти операторы должны быть объединены в блок с помощью фигурных скобок `{...}`.



# Условные операторы

## Оператор `switch`

```
switch(выражение) {  
case константа1:  
последовательность операторов  
break;  
case константа2:  
последовательность операторов  
break;  
case константа3:  
последовательность операторов  
break;  
...  
default:  
последовательность операторов  
break;  
}
```

# Условные операторы

Заданное выражение в операторе switch должно быть целочисленного типа (char, byte, short или int), перечислимого или же строкового.

```
switch (s)
```

```
{
```

```
case "C#":
```

```
    Console.WriteLine("Вы выбрали язык C#");
```

```
break;
```

```
case "VB":
```

```
    Console.WriteLine("Вы выбрали язык Visual Basic");
```

```
break;
```

```
case "C++":
```

```
    Console.WriteLine("Вы выбрали язык C++");
```

```
break;
```

```
default:
```

```
    Console.WriteLine("Такой язык я не знаю");
```

```
break;
```

```
}
```

# Циклы for и while

## Цикл for

Цикл for в C# предоставляет механизм итерации, в котором определенное условие проверяется перед выполнением каждой итерации. Синтаксис этого оператора показан ниже:

*for (инициализатор; условие; итератор) оператор (операторы)*

Здесь:

### инициализатор

это выражение, вычисляемое перед первым выполнением тела цикла (обычно инициализация локальной переменной в качестве счетчика цикла).

Инициализация, как правило, представлена оператором присваивания, задающим первоначальное значение переменной, которая выполняет роль счетчика и управляет циклом;

### условие

это выражение, проверяемое перед каждой новой итерацией цикла (должно возвращать true, чтобы была выполнена следующая итерация);

### итератор

выражение, вычисляемое после каждой итерации (обычно приращение значения счетчика цикла).

# Циклы `for` и `while`

## Цикл `while`

Подобно `for`, `while` также является циклом с предварительной проверкой. Синтаксис его аналогичен, но циклы `while` включают только одно выражение: *while(условие) оператор (операторы);* где *оператор* — это единственный оператор или же блок операторов, а *условие* означает конкретное условие управления циклом и может быть любым логическим выражением. В этом цикле оператор выполняется до тех пор, пока условие истинно. Как только условие становится ложным, управление программой передается строке кода, следующей непосредственно после цикла. Как и в цикле `for`, в цикле `while` проверяется условное выражение, указываемое в самом начале цикла. Это означает, что код в теле цикла может вообще не выполняться, а также избавляет от необходимости выполнять отдельную проверку перед самим циклом.

# Циклы `do while`

## Цикл `do...while`

Цикл `do...while` в C# — это версия `while` с постпроверкой условия. Это значит, что условие цикла проверяется после выполнения тела цикла. Следовательно, циклы `do...while` удобны в тех ситуациях, когда блок операторов должен быть выполнен как минимум однажды. Ниже приведена общая форма оператора цикла `do-while`:

```
do { операторы; } while (условие);
```

При наличии лишь одного оператора фигурные скобки в данной форме записи необязательны. Тем не менее они зачастую используются для того, чтобы сделать конструкцию `do-while` более удобочитаемой и не путать ее с конструкцией цикла `while`. Цикл `do-while` выполняется до тех пор, пока условное выражение истинно.

# Операторы перехода

## Оператор `break`

С помощью оператора `break` можно специально организовать немедленный выход из цикла в обход любого кода, оставшегося в теле цикла, а также минуя проверку условия цикла. Когда в теле цикла встречается оператор `break`, цикл завершается, а выполнение программы возобновляется с оператора, следующего после этого цикла. Оператор `break` можно применять в любом цикле, предусмотренном в C#.

# Операторы перехода

## Оператор `continue`

С помощью оператора `continue` можно организовать преждевременное завершение шага итерации цикла в обход обычной структуры управления циклом. Оператор `continue` осуществляет принудительный переход к следующему шагу цикла, пропуская любой код, оставшийся невыполненным. Таким образом, оператор `continue` служит своего рода дополнением оператора `break`.

В циклах `while` и `do-while` оператор `continue` вызывает передачу управления непосредственно условному выражению, после чего продолжается процесс выполнения цикла. А в цикле `for` сначала вычисляется итерационное выражение, затем условное выражение, после чего цикл продолжается

# Операторы перехода

## Оператор `return`

Оператор `return` организует возврат из метода. Его можно также использовать для возврата значения. Имеются две формы оператора `return`: одна — для методов типа `void`, т.е. тех методов, которые не возвращают значения, а другая — для методов, возвращающих конкретные значения. Для немедленного завершения метода типа `void` достаточно воспользоваться следующей формой оператора `return`:

```
return;
```

Когда выполняется этот оператор, управление возвращается вызывающей части программы, а оставшийся в методе код пропускается.

Для возврата значения из метода в вызывающую часть программы служит следующая форма оператора `return`:

```
return значение;
```