**Система типов**

Язык программирования предполагает наличие правил построения корректных предложений. В свою очередь, предложения строятся из выражений. Основной характеристикой выражения является значение этого выражения. Можно утверждать, что выполнение программы состоит из вычисления значений выражений, которые образуют предложения программы. Синтаксис языка не ограничивает сложности выражения. Выражение может состоять из очень большого количества более простых (элементарных) выражений.

Значение выражения характеризуется типом (типом выражения). Выражение сложной структуры может сочетать элементарные выражения различных типов. При вычислении значений таких выражений требуются дополнительные усилия по преобразованию значений одного типа в значения другого типа. Большая часть этой работы производится автоматически, в ряде случаев от программиста требуются дополнительные усилия по разработке алгоритмов преобразования.

При разработке кода любой сложности знание правил кодирования, вычисления и преобразования значений различных типов является условием создания правильно выполняемой программы.

В разделах этой главы обсуждаются система типов и основные характеристики категорий типов, литералы которые являются основным средством кодирования значений в выражениях, проблемы преобразования значений в выражениях, операторы объявления и связанная с ними проблема области действия имени.

### Категории типов

Система типов включает несколько категорий типов:

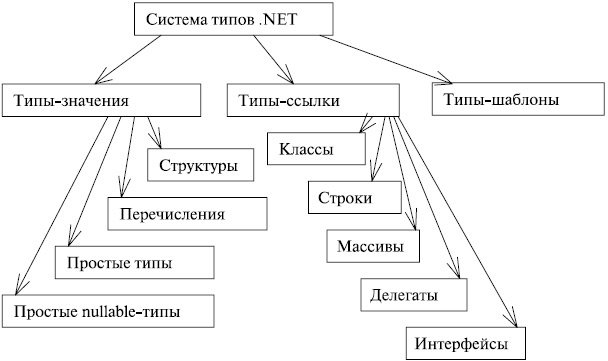
* типы значений (типы-значения),
* ссылочные типы (типы-ссылки),
* параметризованные типы (типы-шаблоны).

Схема типов представлена ниже.

Простые (элементарные) типы – это типы, имя и основные свойства которых известны компилятору. Относительно элементарных типов компилятору не требуется никакой дополнительной информации. Свойства и функциональность этих типов известны.

Среди простых типов различаются:

* ЦЕЛОЧИСЛЕННЫЕ,
* С ПЛАВАЮЩЕЙ ТОЧКОЙ,
* DECIMAL,
* БУЛЕВСКИЙ.



Некоторые характеристики простых (элементарных) типов отражены в следующей таблице. Используемые в .NET языки программирования основываются на общей системе типов. Между именами простых типов в C# и именами типов, объявленных в Framework Class Library, существует взаимно однозначное соответствие. Смысл точечной нотации в графе "Соответствует FCL-типу" состоит в явном указании пространства имен, содержащем объявление соответствующего типа:

|  |  |  |
| --- | --- | --- |
| Sbyte | System.SByte | Целый. 8-разрядное со знаком. Диапазон значений: –128 ... 127 |
| Byte | System.Byte | Целый. 8-разрядное без знака. Диапазон значений: 0 ... 255 |
| Short | System.Int16 | Целый. 16-разрядное со знаком. Диапазон значений: –32768 ... 32767 |
| Ushort | System.UInt16 | Целый. 16-разрядное без знака. Диапазон значений: 0 ... 65535 |
| Int | System.Int32 | Целый. 32-разрядное со знаком. Диапазон значений: –2147483648 ... 2147483647 |
| uint | System.UInt32 | Целый. 32-разрядное без знака. Диапазон значений: 0 ... 4294967295 |
| long | System.Int64 | Целый. 64-разрядное со знаком. Диапазон значений: –9223372036854775808 ... 9223372036854775807 |
| ulong | System.UInt64 | Целый. 64-разрядное без знака. Диапазон значений: 0 ... 18446744073709551615 |
| char | System.Char | 16 (!) разрядный символ UNICODE |
| float | System.Single | Плавающий. 32 разряда. Стандарт IEEE |
| double | System.Double | Плавающий. 64 разряда. Стандарт IEEE |
| decimal | System.Decimal | 128-разрядное значение повышенной точности с плавающей точкой |
| bool | System.Boolean | Значение true или false |

Ниже представлены основные отличия ссылочных типов и типов-значений.

|  |  |  |
| --- | --- | --- |
|  | **Типы-значения** | **Типы-ссылки** |
| Объект представлен | непосредственно значением | ссылкой в стеке или куче |
| Объект располагается | в стеке или куче | в куче |
| Значение по умолчанию | 0, false, ' \0 ', null | ссылка имеет значение null |
| При выполнении операции присваивания копируется | значение | ссылка |

В C# объявление любой структуры и класса основывается на объявлении предопределенного класса object (наследует класс object ). Следствием этого является возможность вызова от имени объектов — представителей любой структуры или класса, унаследованных от класса object методов. В частности, метода ToString. Этот метод возвращает строковое (значение типа string ) представление объекта.

Все типы (типы-значения и типы-ссылки), за исключением простых типов-значений и пары предопределенных ссылочных типов ( string и object ), должны определяться (если уже не были ранее специально определены) программистами в рамках объявлений. Подлежащие объявлению типы называются производными типами.

В разных CLS-языках типам, удовлетворяющим CLS-спецификации, будут соответствовать одни и те же элементарные типы.

Система встроенных типов C# основывается на системе типов .NET Framework Class Library. При создании IL-кода компилятор осуществляет их отображение в типы из .NET FCL.

Параметризованные типы занимают особое место в системе типов и обсуждаются позже.

### object и string: предопределенные ссылочные типы

Итак, к ссылочным типам относятся:

* классы;
* интерфейсы;
* массивы;
* делегаты.

Для каждой категории ссылочных типов существуют собственные правила объявления. Объявления классов вводятся ключевым словом class. Правила объявления классов (как и правила объявления других ссылочных типов, а также типов-значений, объявляемых с помощью ключевого слова struct ) позволяют объявлять неограниченное множество разнообразных ссылочных типов и структур.

Среди множества классов выделяют предопределенные ссылочные типы object и string, которым соответствуют FCL-типы System.Object и System.String. Свойства, функциональность и особенности применения этих типов рассматриваются ниже.

### Литералы. Представление значений

В программах на языках высокого уровня (C# в том числе) литералами называют последовательности входящих в алфавит языка программирования символов, обеспечивающих явное представление значений, которые используются для обозначения начальных значений в объявлении членов классов, переменных и констант в методах класса.

Различаются литералы арифметические (разных типов), логические, символьные (включая Escape-последовательности), строковые.

#### Арифметические литералы

Арифметические литералы кодируют значения различных (арифметических) типов. Тип арифметического литерала определяется следующими интуитивно понятными внешними признаками:

* стандартным внешним видом. Значение целочисленного типа обычно кодируется интуитивно понятной последовательностью символов ' 1 ', ..., ' 9 ', ' 0 '. Значение плавающего типа также предполагает стандартный вид (точка-разделитель между целой и дробной частью, либо научная или экспоненциальная нотация – 1.2500E+052). Шестнадцатеричное представление целочисленного значения кодируется шестнадцатеричным литералом, состоящим из символов ' 0 ', ..., ' 9 ', а также ' a ', ..., ' f ', либо ' A ', ..., ' F ' с префиксом ' 0x ';
* собственно значением. 32768 никак не может быть значением типа short;
* дополнительным суффиксом. Суффиксы l, L соответствуют типу long; ul, UL – unsigned long; f, F – float; d, D – decimal. Значения типа double кодируются без префикса.

#### Логические литералы

К логическим литералам относятся следующие последовательности символов: true и false. Больше логических литералов в C# нет.

#### Символьные литералы

Это заключенные в одинарные кавычки вводимые с клавиатуры одиночные символы: ' X ', ' p ', ' Q ', ' 7 ', а также целочисленные значения в диапазоне от 0 до 65535, перед которыми располагается конструкция вида (char) – операция явного приведения к типу char: (char)34 – ' '' ', (char)44 – ' ,', (char)7541 – какой символ будет здесь – не ясно.

#### Символьные Escape-последовательности

Следующие заключенные в одинарные кавычки последовательности символов являются Escape-последовательностями. Эта категория литералов используется для создания дополнительных эффектов (звонок), простого форматирования выводимой информации и кодирования символов при выводе и сравнении (в выражениях сравнения).

|  |  |
| --- | --- |
| \a | Предупреждение (звонок) |
| \b | Возврат на одну позицию |
| \f | Переход на новую страницу |
| \n | Переход на новую строку |
| \r | Возврат каретки |
| \t | Горизонтальная табуляция |
| \v | Вертикальная табуляция |
| \0 | Ноль |
| \' | Одинарная кавычка |
| \" | Двойная кавычка |
| \\ | Обратная косая черта |

#### Строковые литералы

Это последовательность символов и символьных Escape-последовательностей, заключенных в двойные кавычки.

Verbatim string – строковый литерал, интерпретируемый компилятором так, как он записан. Escape-последовательности воспринимаются строго как последовательности символов.

Verbatim string представляется при помощи символа @, который располагается непосредственно перед строковым литералом, заключенным в парные двойные кавычки. Представление двойных кавычек в Verbatim string обеспечивается их дублированием. Пара литералов (второй – Verbatim string )

..."c:\\My Documents\\sample.txt"...

...@"c:\My Documents\sample.txt"...

имеют одно и то же значение:

c:\My Documents\sample.txt

Представление двойных кавычек внутри Verbatim string достигается за счет их дублирования:

...@"""Focus"""

имеет значение

"Focus"

Строковые литералы являются литералами типа string.

### Переменные элементарных типов. Объявление и инициализация

Объявление – это предложение языка C#, которое используется непосредственно в теле класса для объявления членов класса (в этом случае объявлению может предшествовать спецификатор доступа) или для объявления переменных в конструкторах и методах класса.

Выполнение оператора объявления переменной типа-значения в методе класса приводит к созданию в памяти объекта соответствующего типа, возможно, проинициализированного определенным значением. Это значение может быть задано в виде литерала соответствующего типа или в виде выражения (синтаксис выражений рассматривается далее).

Предложение объявления предполагает (возможное) наличие различных спецификаторов, указание имени типа, имени объекта и (возможно) выражения инициализации. При этом имя типа может быть задано как Действительное Имя Типа (Имя FCL-типа) или как псевдоним типа (имя типа, как оно объявляется в C#). Соответствующее выражение инициализации может быть представлено литералом или выражением более сложной структуры.

Эквивалентные формы записи операторов определения переменных элементарных типов-значений:

int a;

System.Int32 a;

Эквивалентные формы записи операторов определения и инициализации переменных типа значения:

int a = 0;

int a = new int();

System.Int32 a = 0;

System.Int32 a = new System.Int32();

Здесь следует учитывать важное обстоятельство: CLR не допускает использования в выражениях неинициализированных локальных переменных. В C# к таковым относятся переменные, объявленные в теле метода. Так что при разработке алгоритма следует обращать на это особое внимание.

int a; // Объявление a.

int b; // Объявление b.

b = 10; // Инициализация b.

a=b+b; // Инициализация a.

### Константы

Объявляются с дополнительным спецификатором const. Требуют непосредственной инициализации. В данном примере инициализируется литералом 3.14.

const float Pi = 3.14;

### Операции и выражения

Для каждого определенного в C# типа существует собственный набор операций, определенных на множестве значений этого типа.

Эти операции задают диапазон возможных преобразований, которые могут быть осуществлены над элементами множеств значений типа. Несмотря на специфику разных типов, в C# существует общее основание для классификации соответствующих множеств операций. Каждая операция является членом определенного подмножества операций и имеет собственное графическое представление.

Общие характеристики используемых в C# операций представлены ниже.

|  |  |
| --- | --- |
| Arithmetic | + – \* / % |
| Логические (boolean и побитовые) | & | ^ ! ~ && || |
| Строковые (конкатенаторы) | + |
| Increment, decrement | ++ –– |
| Сдвига | >> << |
| Сравнения | == != < > <= >= |
| Присвоения | = += –= \*= /= %= &= |= ^= <<= >>= |
| Member access | . |
| Индексации | [] |
| Cast (приведение типа) | () |
| Conditional (трехоперандная) | ?: |
| Delegate concatenation and removal | + – |
| Создания объекта | new() |
| Type information | is sizeof typeof |
| Overflow exception control (управление исключениями) | checked unchecked |
| Indirection and Address (неуправляемый код) | \* –> [] & |

На основе элементарных (первичных) выражений с использованием этих самых операций и дополнительных разделителей в виде открывающихся и закрывающихся скобочек формируются выражения все более сложной структуры. Кроме того, при создании, трансляции, а главное, на стадии выполнения (определения значения выражения) учитываются следующие обстоятельства:

* приоритет операций,
* типы операндов и приведение типов.

### Приоритет операций

|  |  |
| --- | --- |
| 1 | () [] . (постфикс)++ (постфикс)–– new sizeof typeof unchecked |
| 2 | ! ~ (имя типа) +(унарный) –(унарный) ++(префикс) ––(префикс) |
| 3 | \* / % |
| 4 | + – |
| 5 | << >> |
| 6 | < > <= => is |
| 7 | == != |
| 8 | & |
| 9 | ^ |
| 10 | | |
| 11 | && |
| 12 | || |
| 13 | ?: |
| 14 | = += –= \*= /= %= &= |= ^= <<= >>= |

### Приведение типов

Приведение типов – один из аспектов безопасности языка.

Используемые в программе типы характеризуются собственными диапазонами значений, которые определяются свойствами типов – в том числе и размером области памяти, предназначенной для кодирования значений соответствующего типа. При этом области значений различных типов пересекаются. Многие значения можно выразить более чем одним типом. Например, значение 4 можно представить как значение типа sbyte, byte, short, ushort, int, uint, long, ulong. При этом в программе все должно быть устроено таким образом, чтобы логика преобразования значений одного типа к другому типу была бы понятной, а результаты этих преобразований – предсказуемы. В одном выражении могут быть сгруппированы операнды различных типов. Однако возможность подобного "смешения" при определении значения выражения приводит к необходимости применения дополнительных усилий по приведению значений операндов к "общему типу".

Иногда приведение значения к другому типу происходит автоматически. Такие преобразования называются неявными.

Но в ряде случаев преобразование требует дополнительного внимания со стороны программиста, который должен явным образом указывать необходимость преобразования, используя выражения приведения типа или обращаясь к специальным методам преобразования, определенным в классе System.Convert, которые обеспечивают преобразование значения одного типа к значению другого (в том числе значения строкового типа к значениям базовых типов).

Преобразование типа создает значение нового типа, эквивалентное значению старого типа, однако при этом не обязательно сохраняется идентичность (или точные значения) двух объектов.

Различаются:

Расширяющее преобразование – значение одного типа преобразуется к значению другого типа, которое имеет такой же или больший размер. Например, значение, представленное в виде 32-разрядного целого числа со знаком, может быть преобразовано в 64-разрядное целое число со знаком. Расширяющее преобразование считается безопасным, поскольку исходная информация при таком преобразовании не искажается.

|  |  |
| --- | --- |
| Byte | UInt16, Int16, UInt32, Int32, UInt64, Int64, Single, Double, Decimal |
| SByte | Int16, Int32, Int64, Single, Double, Decimal |
| Int16 | Int32, Int64, Single, Double, Decimal |
| UInt16 | UInt32, Int32, UInt64, Int64, Single, Double, Decimal |
| Char | UInt16, UInt32, Int32, UInt64, Int64, Single, Double, Decimal |
| Int32 | Int64, Double, Decimal |
| UInt32 | Int64, Double, Decimal |
| Int64 | Decimal |
| UInt64 | Decimal |
| Single | Double |

Некоторые расширяющие преобразования типа могут привести к потере точности. Следующая таблица описывает варианты преобразований, которые иногда приводят к потере информации.

|  |  |
| --- | --- |
| Int32 | Single |
| UInt32 | Single |
| Int64 | Single, Double |
| UInt64 | Single, Double |
| Decimal | Single, Double |

Сужающее преобразование – значение одного типа преобразуется к значению другого типа, которое имеет меньший размер (из 64-разрядного в 32-разрядное). Такое преобразование потенциально опасно потерей значения.

Сужающие преобразования могут приводить к потере информации. Если тип, к которому осуществляется преобразование, не может правильно передать значение источника, то результат преобразования оказывается равен константе PositiveInfinity или NegativeInfinity.

При этом значение PositiveInfinity интерпретируется как результат деления положительного числа на ноль, а значение NegativeInfinity — как результат деления отрицательного числа на ноль.

Если сужающее преобразование обеспечивается методами класса System.Convert, то потеря информации сопровождается генерацией исключения (об исключениях позже).

|  |  |
| --- | --- |
| Byte | Sbyte |
| SByte | Byte, UInt16, UInt32, UInt64 |
| Int16 | Byte, SByte, UInt16 |
| UInt16 | Byte, SByte, Int16 |
| Int32 | Byte, SByte, Int16, UInt16, UInt32 |
| UInt32 | Byte, SByte, Int16, UInt16, Int32 |
| Int64 | Byte, SByte, Int16, UInt16, Int32, UInt32, UInt64 |
| UInt64 | Byte, SByte, Int16, UInt16, Int32, UInt32, Int64 |
| Decimal | Byte, SByte, Int16, UInt16, Int32, UInt32, Int64, UInt64 |
| Single | Byte, SByte, Int16, UInt16, Int32, UInt32, Int64, UInt64 |
| Double | Byte, SByte, Int16, UInt16, Int32, UInt32, Int64, UInt64 |

### object. Характеристики типа

Всеобщий базовый тип. Обязательная составляющая любого типа в .NET. Функциональные характеристики типа System.Object приводятся в таблице.

Конструктор

|  |  |
| --- | --- |
| Object | Создает и инициализирует объект типа Object |

Общедоступные (public) методы

|  |  |
| --- | --- |
| Equals | Обеспечивает сравнение объектов |
| GetHashCode | Обеспечивает реализацию алгоритма хэширования для значений объектов |
| GetType | Для любого объекта создает объект типа Type, содержащий информацию о структуре типа данного объекта |
| ReferenceEquals | Проверка эквивалентности ссылок. Статический |
| ToString | Возвращает объект типа String с описанием данного объекта |

Защищенные (protected) методы

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| Finalize | Реализует процедуру уничтожения объекта. |
| MemberwiseClone | Создает копию текущего объекта. |

### Особенности выполнения арифметических операций

Особенности выполнения операций над целочисленными операндами и операндами с плавающей точкой связаны с особенностями выполнения арифметических операций и с ограниченной точностью переменных типа float и double.

Представление величин:

float – 7 значащих цифр

double – 16 значащих цифр

1000000\*1000000==1000000000000, но максимально допустимое положительное значение для типа System.Int32 составляет 2147483647. В результате переполнения получается неверный результат –727379968.

Ограниченная точность значений типа System.Single проявляется при присвоении значений переменной типа System.Double. Приводимый ниже простой программный код иллюстрирует некоторые особенности арифметики .NET:

using System;

class Class1

{

const double epsilon = 0.00001D;

static void Main(string[] args)

{

int valI = 1000000, resI;

resI = (valI\*valI)/valI;

// –727379968/1000000 == –727

Console.WriteLine

("The result of action (1000000\*1000000/1000000) is {0}", resI);

float valF00 = 0.2F, resF;

double valD00 = 0.2D, resD;

// Тест на количество значащих цифр для значений типа double и float.

resD = 12345678901234567890; Console.WriteLine(">>>>> {0:F10}",resD);

resF = (float)resD; Console.WriteLine(">>>>> {0:F10}",resF);

resD = (double)(valF00 + valF00); // 0.400000005960464

if (resD == 0.4D) Console.WriteLine("Yes! {0}",resD);

else Console.WriteLine("No! {0}",resD);

resF = valF00\*5.0F;

resD = valD00\*5.0D;

resF = (float)valD00\*5.0F;

resD = valF00\*5.0D; //1.0000000149011612

if (resD == 1.0D) Console.WriteLine("Yes! {0}",resD);

else Console.WriteLine("No! {0}",resD);

resF = valF00\*5.0F;

resD = valF00\*5.0F; //1.0000000149011612

if (resD.Equals(1.0D)) Console.WriteLine("Yes! {0}",resD);

else Console.WriteLine("No! {0}",resD);

if (Math.Abs(resD – 1.0D) < epsilon)

Console.WriteLine("Yes! {0:F7}, {1:F7}",resD – 1.0D, epsilon);

else

Console.WriteLine("No! {0:F7}, {1:F7}",resD – 1.0D, epsilon);

}

}

Листинг 2.1.

В результате выполнения программы выводится такой результат:

The result of action (1000000\*1000000/1000000) is -727

>>>>> 12345678901234600000,0000000000

>>>>> 12345680000000000000,0000000000

No! 0,400000005960464

No! 1,00000001490116

No! 1,00000001490116

Yes! 0,0000000, 0,0000100

### Особенности арифметики с плавающей точкой

* Если переменной типа float присвоить величину x из интервала –1.5E–45 < x < 1.5E–45 (x != 0), результатом операции окажется положительный ( x > 0 ) или отрицательный ( x < 0 ) нуль ( +0, –0 ).
* Если переменной типа double присвоить величину x из интервала –5E–324 < x < 5E–324 (x != 0), результатом операции окажется положительный ( x > 0 ) или отрицательный ( x < 0 ) нуль ( +0, –0 ).
* Если переменной типа float присвоить величину x, которая –3.4E+38 < x или x > 3.4E+38, результатом операции окажется положительная ( x > 0 ) или отрицательная ( x < 0 ) бесконечность ( +Infinity, –Infinity ).
* Если переменной типа double присвоить величину x, для которой –1.7E+308 > x или x < 1.7E+308, результатом операции окажется положительная ( x > 0 ) или отрицательная ( x < 0 ) бесконечность ( +Infinity, –Infinity ).
* Выполнение операции деления над значениями типов с плавающей точкой ( 0.0/0.0 ) дает NaN (Not a Number).

### checked и unchecked. Контроль за переполнением

Причиной некорректных результатов выполнения арифметических операций является особенность представления значений арифметических типов.

Арифметические типы ию. По умолчанию в C# переполнение, возникающее при выполнении операций, никак не контролируется. Возможный неверный результат вычисления остается всего лишь результатом выполнения операции, и никого не касается, КАК эта операция выполнялась.

Механизм контроля за переполнением, возникающим при выполнении арифметических операций, обеспечивается ключевыми словами checked (включить контроль за переполнением) и unchecked (отключить контроль за переполнением), которые используются в составе выражений. Конструкции управления контролем за переполнением имеют две формы:

* операторную, которая обеспечивает контроль над выполнением одного выражения:
* :::::
* short x = 32767;
* short y = 32767;
* short z = 0;
* try
* {
* z = checked(x + unchecked(x+y));
* }
* catch (System.OverflowException e)
* {
* Console.Writeline("Переполнение при выполнении сложения");
* }
* return z;

:::::

При этом контролируемое выражение может быть произвольной формы и сложности и может содержать другие вхождения как контролируемых, так и неконтролируемых выражений;

* блочную, которая обеспечивает контроль над выполнением операций в блоке операторов:
* :::::
* short x = 32767;
* short y = 32767;
* short z = 0, w = 0;
* try
* {
* unchecked
* {
* w = x+y;
* }
* checked
* {
* z = x+w;
* }
* }
* catch (System.OverflowException e)
* {
* Console.Writeline("Переполнение при выполнении сложения");
* }
* return z;

:::::

Естественно, контролируемые блоки при этом также могут быть произвольной сложности.

### Константное выражение

Константное выражение – это либо элементарное константное выражение, к которым относятся:

* символьный литерал,
* целочисленный литерал,
* символьная константа,
* целочисленная константа,
* либо выражение, построенное на основе элементарных константных выражений с использованием скобок и символов операций, определенных на множестве значений данного типа.

Отличительные черты константного выражения:

* значение константного выражения не меняется при выполнении программы;
* значение константного выражения становится известно на этапе компиляции модуля, до начала выполнения модуля.

### Перечисления

Перечисление объявляется с помощью ключевого слова enum, идентифицируется по имени и представляет собой непустой список неизменяемых именованных значений интегрального типа. Первое значение в перечислении по умолчанию инициализируется нулем. Каждое последующее значение отличается от предыдущего по крайней мере на единицу, если объявление значения не содержит явного дополнительного присвоения нового значения. Пример объявления перечисления приводится ниже:

enum Colors { Red = 1, Green = 2, Blue = 4, Yellow = 8 };

Обращение к элементу перечисления осуществляется посредством сложного выражения, состоящего из имени класса перечисления, операции доступа к элементу перечисления ' .', имени элемента перечисления:

int xVal = (int)Colors.Red; //Переменная xVal инициализируется значением перечисления.

Перечисление является классом, а это означает, что в распоряжении программиста оказываются методы сравнения значений перечисления, методы преобразования значений перечисления в строковое представление, методы перевода строкового представления значения в перечисление, а также (судя по документации) средства для создания объектов —представителей класса перечисления.

Далее приводится список членов класса перечисления.

Открытые методы

|  |  |
| --- | --- |
| CompareTo | Сравнивает этот экземпляр с заданным объектом и возвращает сведения об их относительных значениях |
| Equals | Переопределен. Возвращает значение, показывающее, равен ли данный экземпляр заданному объекту |
| Format | Статический. Преобразует указанное значение заданного перечисляемого типа в эквивалентное строчное представление в соответствии с заданным форматом |
| GetHashCode | Переопределен. Возвращает хэш-код для этого экземпляра |
| GetName | Статический. Выводит имя константы в указанном перечислении, имеющем заданное значение |
| GetNames | Статический. Выводит массив имен констант в указанном перечислении |
| GetType (унаследовано от Object ) | Возвращает Type текущего экземпляра |
| GetTypeCodeType | Возвращает базовый тип TypeCode для этого экземпляра |
| GetUnderlying | Статический. Возвращает базовый тип указанного перечисления |
| GetValues | Статический. Выводит массив значений констант в указанном перечислении |
| IsDefined | Статический. Возвращает признак наличия константы с указанным значением в заданном перечислении |
| Parse | Статический. Перегружен. Преобразует строковое представление имени или числового значения одной или нескольких перечисляемых констант в эквивалентный перечисляемый объект |
| ToObject | Статический. Перегружен. Возвращает экземпляр указанного типа перечисления, равный заданному значению |
| ToString | Перегружен. Переопределен. Преобразует значение этого экземпляра в эквивалентное ему строковое представление |

Защищенные конструкторы

|  |  |
| --- | --- |
| Enum – конструктор |  |

Защищенные методы

|  |  |
| --- | --- |
| Finalize (унаследовано от Object ) | Переопределен. Позволяет объекту Object попытаться освободить ресурсы и выполнить другие завершающие операции, перед тем как объект Object будет уничтожен в процессе сборки мусора. В языках C# и C++ для функций финализации используется синтаксис деструктора |
| MemberwiseClone (унаследовано от Object) | Создает неполную копию текущего Object |

### Объявление переменных. Область видимости и время жизни

Любая используемая в программе сущность вводится объявлением.

Объявлению подлежат:

* классы и структуры. Класс (структура) может содержать вложенные объявления других классов и структур;
* перечисления;
* объекты – переменные и константы, представляющие классы и структуры. Корректное объявление объекта предполагает, что информация, содержащая характеристики объявляемого объекта, доступна транслятору;
* элементы перечислений – объекты, представляющие перечисления;
* конструкторы классов и методы (функции) – члены классов (в том числе и специальные).

Объявляемой сущности присваивается имя. Обращение к ранее объявленным сущностям в программе обеспечивается различными вариантами имен. Имена в программе повсюду, и главная проблема заключается в том, чтобы не допустить неоднозначности при обращении из разных мест программы к классам, членам классов, перечислениям, переменным и константам.

Конфликта имен (проблемы с обращением к одноименным объектам) позволяет избежать принятая в языке дисциплина именования. В каждом языке программирования она своя.

Избежать конфликта имен в C# позволяет такая синтаксическая конструкция, как блок операторов. Блок – это множество предложений (возможно пустое), заключенное в фигурные скобки.

Различается несколько категорий блоков:

* Тело класса (структуры). Место объявления членов класса.
* Тело метода. Место расположения операторов метода.
* Блок в теле метода.

Переменные можно объявлять в любом месте блока. Точка объявления переменной в буквальном смысле соответствует месту ее создания. Обращение к объявляемой сущности (переменной или константе) "выше" точки ее объявления лишено смысла.

Новый блок – новая область видимости. Объекты, объявляемые во внутренних блоках, не видны во внешних блоках. Объекты, объявленные в методе и во внешних блоках, видны и во внутренних блоках. Одноименные объекты во вложенных областях конфликтуют.

Объекты, объявляемые в блоках одного уровня вложенности в методе, не видны друг для друга. Конфликта имен не происходит.

В теле класса не допускается объявления одноименных данных-членов. Нарушение этого правила приводит к неоднозначности.

В теле класса не допускается объявления одноименных функций — членов (методов класса) с пустыми списками параметров. Также не допускаются объявления одноименных функций — членов (методов класса) со списками параметров, у которых совпадают типы параметров.

В списках параметров имена параметров не важны. Важно, что выражение вызова метода в этом случае будет неоднозначным.

Имена данных — членов класса не конфликтуют с одноименными переменными, объявляемыми в теле методов, поскольку в теле метода обращение к членам класса обеспечивается выражениями с операцией доступа, и никакого конфликта в принципе быть не может.

Ниже приводится простой программный код, отражающий особенности принятой в C# концепции областей видимости.

using System;

class Class1

{

static int s;

static void Main(string[] args)

{

{int s;}

{int s;}

{

Class1.s = 100; // Классифицированное имя!

int s;

{

//int s;

}

}

System.Int32 z = new int();

char a = (char)41;

char b = 'X';

Console.WriteLine("{0} {1}",a,b);

}

//void QQQ(int q0)

//{

//Class1.s = 10;

//int s;

//}

//void QQQ(int q1)

//{

//Class1.s = 10;

//int s;

//}

int QQQ(int q2)

{

Class1.s = 10;

int s;

return 0;

}

int QQQ(int q2, int q3)

{

//s = 100;

Class1.s = 10;

int s;

return 0;

}

}

литералы являются литералами типа string.