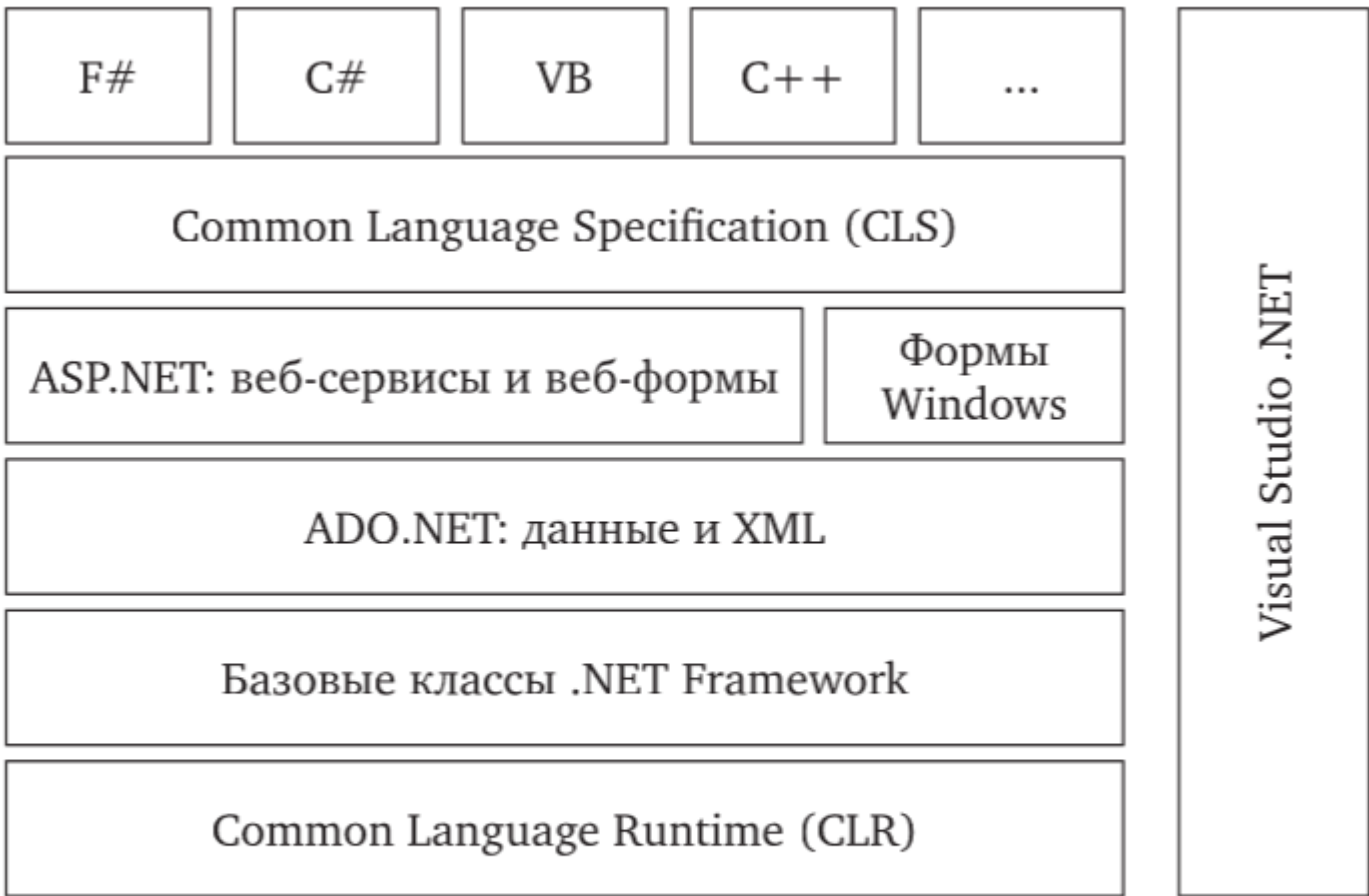


ПРОГРАММИРОВАНИЕ НА C#

Microsoft Visual Studio .NET

Текущей версией языка является версия C# 12, которая вышла 14 ноября 2023 года вместе с релизом .NET 8.

C# является языком с Си-подобным синтаксисом и близок в этом отношении к C++ и Java.



Последняя версия платформы на данный момент - .NET 8 поддерживается на большинстве современных ОС Windows, MacOS, Linux.

Используя различные технологии на платформе .NET, можно разрабатывать приложения на языке C# для самых разных платформ - Windows, MacOS, Linux, Android, iOS, Tizen.

Рис. 1.1. Архитектурная схема Microsoft .NET Framework и Visual Studio .NET

Разнообразие технологий. Общеязыковая среда исполнения CLR и базовая библиотека классов являются основой для целого стека технологий, которые разработчики могут задействовать при построении тех или иных приложений.

- Например, для работы с базами данных в этом стеке технологий предназначена технология ADO.NET и Entity Framework Core.
- Для построения графических приложений с богатым насыщенным интерфейсом - технология WPF и WinUI, для создания более простых графических приложений - Windows Forms.
- Для разработки кроссплатформенных мобильных и десктопных приложений - Xamarin/MAUI. Для создания веб-сайтов и веб-приложений - ASP.NET и т.д.
- Также еще следует отметить такую особенность языка C# и фреймворка .NET, как автоматическая сборка мусора. А это значит, что нам в большинстве случаев не придется, в отличие от C++, заботиться об освобождении памяти. Вышеупомянутая общеязыковая среда CLR сама вызовет сборщик мусора и очистит память.
- следует различать .NET Framework, который предназначен преимущественно для Windows -- устарел, и кроссплатформенный .NET 8



Рис. 1.2. Схема компиляции Common Language Runtime

Код на C# компилируется в приложения или сборки с расширениями exe или dll на языке CIL.

Далее, при запуске на выполнение подобного приложения происходит JIT-компиляция (Just-In-Time) из языка CIL (Common Intermediate Language) в машинный код, который затем выполняется.

Общая структура программы C#

Программа на языке C# состоит из одного или нескольких файлов.

Каждый файл может содержать или не содержать пространства имен.

Пространство имен может содержать типы, такие как классы, структуры, интерфейсы, перечисления и делегаты или другие пространства имен.

Метод Main

— это точка входа приложения C#.

При запуске Main приложения метод является первым вызываемым методом.

В программе на C# может существовать только одна точка входа.

Пример структуры программы на C#, содержащей все эти элементы:

```
// A skeleton of a C# program
using System;
namespace YourNamespace
{
    class YourClass    { }

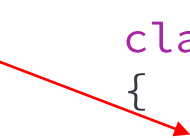
    struct YourStruct {}

    interface IYourInterface {}

    enum YourEnum {}

    namespace YourNestedNamespace
    {
        struct YourStruct { }
    }

    class Program
    {
        static void Main(string[] args)
        {
            //Your program starts here...
            Console.WriteLine("Hello world!");
        }
    }
}
```



Программы без Main методов — Операторы верхнего уровня

Не нужно явно включать Main метод в проект консольного приложения.

Ниже приведен файл Program.cs, который является *полной* программой C# в C# 10++:

```
Console.WriteLine("Hello World!");
```

Операторы верхнего уровня позволяют создавать простой программный код для небольших служебных или учебных программ.

Приложение должно иметь только одну точку входа. Проект может содержать **только один файл** с операторами верхнего уровня. Размещение операторов верхнего уровня в нескольких файлах в проекте приводит к ошибке компилятора.

Директивы using

```
using System.Text;
```

```
StringBuilder builder = new();  
builder.AppendLine("Hello");  
builder.AppendLine("World!");  
Console.WriteLine(builder.ToString());
```

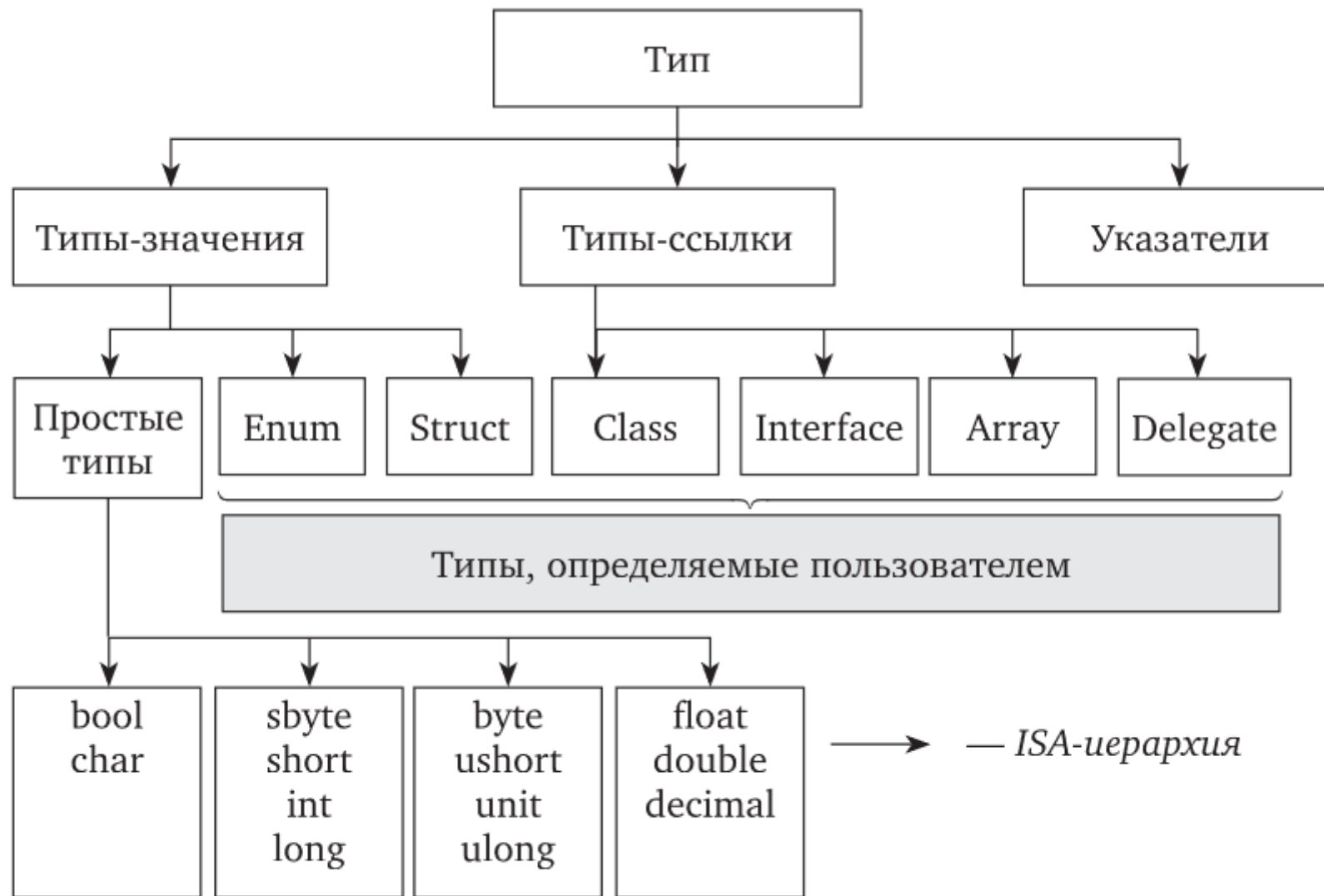


Рис. 1.4. Универсальная схема типизации (CTS)

Когда вы объявляете в программе переменную или константу, для нее нужно задать тип либо использовать ключевое слово var, чтобы компилятор определил тип самостоятельно.

```
// Declaration only:
float temperature;
string name;
MyClass myClass;

// Declaration with initializers (four examples):
char firstLetter = 'C';
var limit = 3;
int[] source = { 0, 1, 2, 3, 4, 5 };
var query = from item in source
            where item <= limit
            select item;
```

Тип данных	Описание	Пример
byte	От 0 до 255 (без знака)	35
ushort	От 0 до 65 535 (без знака)	325
uint	От 0 до 4 294 967 295 (без знака)	28 765
ulong	От 0 до 18 446 744 073 709 551 615 (1,8E+19) (без знака)	421 467
short	От -32 768 до 32 767. Это тот же тип, что и int16	-672
int	От -2 147 483 648 до 2 147 483 647. Это тот же самый тип, что и int32	12 878
long	От -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807. Это тот же тип, что и int32	-34 567 654
float	От -3,4028235e38 до 3,4028235e38	2,5F
double	От -1,79769313486231570E+308 до 1,797693486321570E + 308	-1,3476213
decimal	± 7,92281662514264337593543950335 с 28 разрядами справа от десятичной точки	-2,23451942338
char	Символ Unicode	'f'
string	От 0 до 2 миллиардов (иначе — миллиардов) символов Unicode	"top"
bool	Значение true или false	true
DateTime	От 00:00:00 1 января 0001 до 11:59:59 31 декабря 9999 года	DateTime dateValue = = new DateTime(2013, 6, 18);
Object	В переменной данного типа может храниться любой тип	—

Типы параметров и возвращаемых значений метода задаются в объявлении метода. Далее представлена сигнатура метода, который требует значение `int` в качестве входного аргумента и возвращает строку:

```
public string GetName(int ID)
{
    if (ID < names.Length)
        return names[ID];
    else
        return String.Empty;
}
private string[] names = { "Spencer", "Sally", "Doug" };
```

После объявления переменной вы не можете повторно объявить ее с новым типом и назначить ей значение, несовместимое с объявленным типом. Например, нельзя объявить переменную типа `int` и затем присвоить ей логическое значение `true`. Но значения можно преобразовать в другие типы, например при сохранении в других переменных или передаче в качестве аргументов метода.

Важно понимать две основные вещи, касающиеся системы типов, используемой в .NET:

- Она поддерживает принцип наследования.
- Типы могут быть производными от других типов, которые называются *базовыми типами*.
- Производный тип наследует все (с некоторыми ограничениями) методы, свойства и другие члены базового типа.
- Базовый тип, в свою очередь, может быть производным от какого-то другого типа, при этом производный тип наследует члены обоих базовых типов в иерархии наследования.
- Все типы, включая встроенные числовые типы, например System.Int32 (ключевое слово C#: `int`), в конечном счете являются производными от одного базового типа System.Object (ключевое слово C#: `object`).

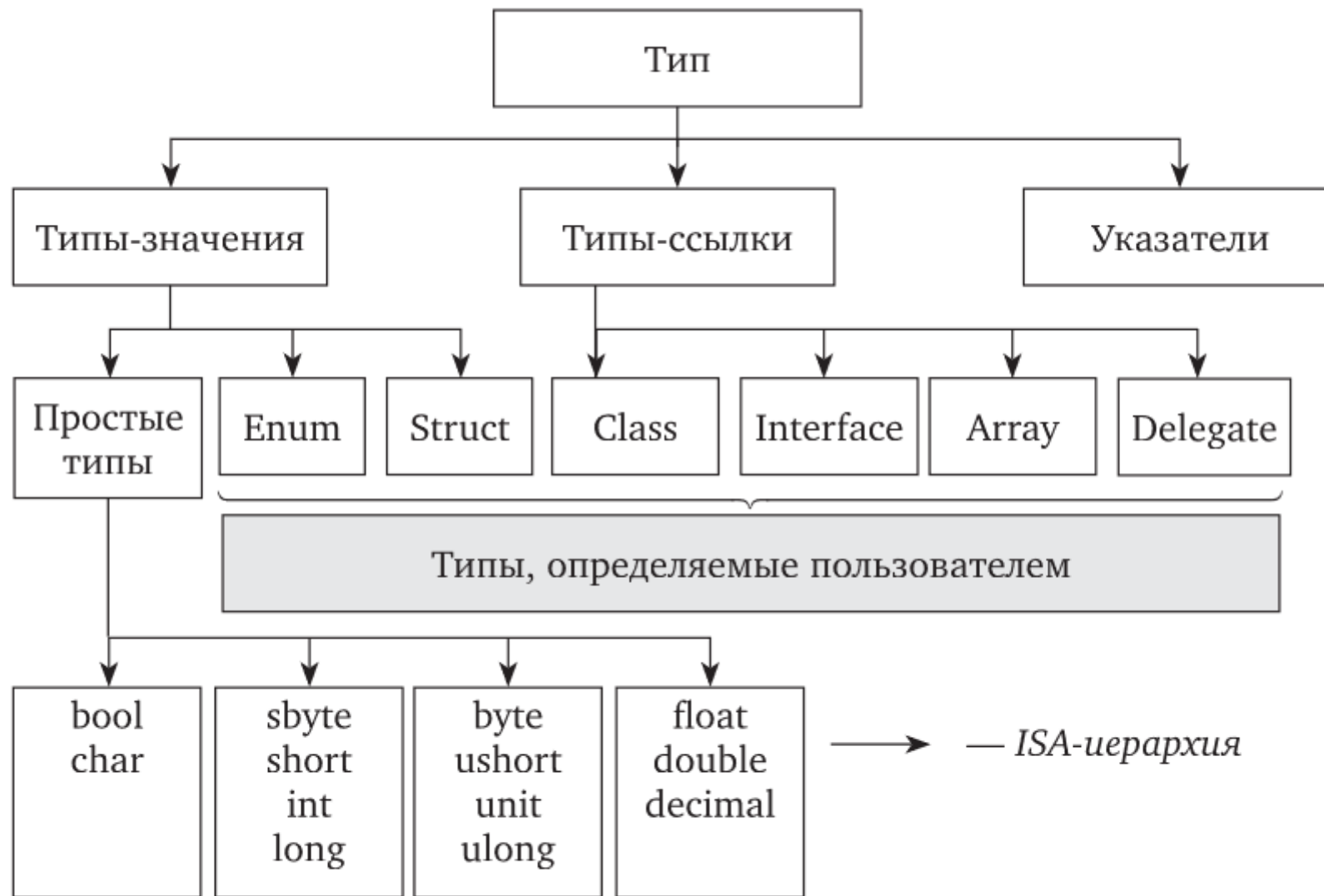


Рис. 1.4. Универсальная схема типизации (CTS)

Типы значений

Используйте ключевое слово [struct](#), чтобы создать собственные пользовательские типы значений. Как правило, структура используется как контейнер для небольшого набора связанных переменных, как показано в следующем примере:

```
public struct Coords
{
    public int x, y;

    public Coords(int p1, int p2)
    {
        x = p1;
        y = p2;
    }
}
```

Еще одна категория типов значений — это enum.

Перечисление определяет набор именованных целочисленных констант.

```
public enum FileMode
{
    CreateNew = 1,
    Create = 2,
    Open = 3,
    OpenOrCreate = 4,
    Truncate = 5,
    Append = 6,
}
```

Так как имена намного лучше воспринимаются человеком при изучении исходного кода, рекомендуется всегда использовать перечисления вместо литеральных числовых констант.

Ссылочные типы

Объявляемая переменная с типом [reference type](#) будет содержать значение [null](#) до тех пор, пока вы не назначите ей экземпляр такого типа или не создадите его с помощью оператора [new](#). Создание и назначение класса демонстрируется в следующем примере:

```
MyClass myClass = new MyClass();  
MyClass myClass2 = myClass;
```

Объявление пространств имен для упорядочивания типов

Пространства имен часто используются в программировании на C# двумя способами. Первый способ — .NET использует пространства имен для упорядочения множества ее классов следующим образом:

```
System.Console.WriteLine("Hello World!");
```

System является пространством имен, а Console — это класс в нем.

using - ключевое слово можно использовать таким образом, чтобы полное имя не требовалось, как показано в следующем примере:

```
using System;  
  
Console.WriteLine("Hello World!");
```

Второй способ — объявление собственных пространств имен поможет вам контролировать область имен классов и методов в более крупных проектах. Используйте ключевое слово [namespace](#), чтобы объявить пространство имен, как показано в следующем примере:

```
namespace SampleNamespace
{
    class SampleClass
    {
        public void SampleMethod()
        {
            System.Console.WriteLine(
                "SampleMethod inside SampleNamespace");
        }
    }
}
```


Объявление классов

Классы объявляются с помощью ключевого слова `class` и уникального идентификатора, как показано в следующем примере:

```
//[access modifier] - [class] - [identifier]
public class Customer
{
    // Fields, properties, methods and events go here...
}
```

Объекты можно создать с помощью `new` ключевое слово, за которым следует имя класса, как показано ниже.

```
Customer object1 = new Customer();
```

Консольный ввод-вывод

Для вывода информации на консоль используется встроенный метод **Console.WriteLine**.

```
string hello = "Привет мир";  
Console.WriteLine(hello);  
Console.WriteLine("Добро пожаловать в C#!");  
Console.WriteLine("Пока мир...");  
Console.WriteLine(24.5);
```

Привет мир!

Добро пожаловать в C#!

Пока мир...

24,5

```
Console.WriteLine($"Имя: {name}  Возраст: {age}  Рост: {height}м");
```

Имя: Том Возраст: 34 Рост: 1,7м

Консольный ввод

Для ввода информации от пользователя есть метод `Console.ReadLine()`. Он позволяет получить введенную строку.

```
1 Console.Write("Введите свое имя: ");  
2 string? name = Console.ReadLine();  
3 Console.WriteLine($"Привет {name}");
```

Введите свое имя: Том

Привет Том

Особенностью метода `Console.ReadLine()` является то, что он может считать информацию с консоли только в виде строки. Кроме того, возможная ситуация, когда для метода `Console.ReadLine` не окажется доступных для считывания строк, то есть когда ему нечего считывать, он возвращает значение `null`, то есть, грубо говоря, фактически отсутствие значения. И чтобы отразить эту ситуацию мы определяем переменную `name`, в которую получаем ввод с консоли, как переменную типа `string?`

Пример ввода значений:

```
1 Console.Write("Введите имя: ");
2 string? name = Console.ReadLine();
3
4 Console.Write("Введите возраст: ");
5 int age = Convert.ToInt32(Console.ReadLine());
6
7 Console.Write("Введите рост: ");
8 double height = Convert.ToDouble(Console.ReadLine());
9
10 Console.Write("Введите размер зарплаты: ");
11 decimal salary = Convert.ToDecimal(Console.ReadLine());
12
13 Console.WriteLine($"Имя: {name}  Возраст: {age}  Рост: {height}м  Зарплата: {salary}$");
```

Введите имя: Том

Введите возраст: 25

Введите рост: 1,75

Введите размер зарплаты: 300,67

Имя: Том Возраст: 25 Рост: 1,75м Зарплата: 300,67\$

Конструкция if..else и тернарная операция

простейшая форма
состоит из блока if:

```
if(условие)
{
    выполняемые инструкции
}
```

чтобы при
несоблюдении
условия также
выполнялись какие-
либо действия мы
можем добавить блок
else или else if:

```
if(num1 > num2)
{
    Console.WriteLine($"Число {num1} больше числа {num2}");
}
else if (num1 < num2)
{
    Console.WriteLine($"Число {num1} меньше числа {num2}");
}
else
{
    Console.WriteLine("Число num1 равно числу num2");
}
```

Тернарная операция

Тернарную операция также позволяет проверить некоторое условие и в зависимости от его истинности выполнить некоторые действия. Она имеет следующий синтаксис:

```
1 [первый операнд - условие] ? [второй операнд] : [третий операнд]
```

```
int x=3;  
int y=2;  
int z = x < y ? (x+y) : (x-y);  
Console.WriteLine(z); // 1
```

Циклы

Цикл for

Цикл for имеет следующее формальное определение:

```
1  for ([действия_до_выполнения_цикла]; [условие]; [действия_после_выполнения])
2  {
3      // действия
4  }
```

```
1  for (int i = 1; i < 4; i++)
2  {
3      Console.WriteLine(i);
4  }
```

```
1
2
3
```


Циклы

Цикл foreach

Цикл foreach предназначен для перебора набора или коллекции элементов. Его общее определение:

```
1 foreach(тип_данных переменная in коллекция)
2 {
3     // действия цикла
4 }
```

```
1 foreach(char c in "Tom")
2 {
3     Console.WriteLine(c);
4 }
```



T
o
m

Подпрограммы

Главное отличие подпрограммы от основной (главной) программы заключается в том, что **управление может быть передано только главной программе.**

Подпрограмма может быть откомпилирована, но не может быть запущена на исполнение.

- **Функция** - подпрограмма, выполняющая какие-либо операции и возвращающая значение.
Процедура - подпрограмма, которая только выполняет операции, без возврата значения.
Метод - это функция или процедура, которая принадлежит классу или экземпляру класса.

Подпрограммы

Главное отличие подпрограммы от основной (главной) программы заключается в том, что **управление может быть передано только главной программе.**

Подпрограмма может быть откомпилирована, но не может быть запущена на исполнение.

- **Функция** - подпрограмма, выполняющая какие-либо операции и возвращающая значение.
Процедура - подпрограмма, которая только выполняет операции, без возврата значения.
Метод - это функция или процедура, которая принадлежит классу или экземпляру класса. В C# все инструкции выполняются в контексте метода.

Сигнатуры методов

Методы объявляются в классе, структуре или интерфейсе путем указания уровня доступа, такого как

- `public` или `private`,
- необязательных модификаторов, таких как `abstract` или `sealed`,
- возвращаемого значения, имени метода и всех параметров этого метода.

Все эти части вместе представляют собой сигнатуру метода.

По сути метод - это именованный блок кода, который выполняет некоторые действия.

Общее определение методов выглядит следующим образом:

```
1 [модификаторы] тип_возвращаемого_значения название_метода ([параметры])  
2 {  
3     // тело метода  
4 }
```

Модификаторы и параметры необязательны.

Определение метода

```
abstract class Motorcycle
{
    // Anyone can call this.
    public void StartEngine() { /* Method statements here */ }

    // Only derived classes can call this.
    protected void AddGas(int gallons) { /* Method statements here */ }

    // Derived classes can override the base class implementation.
    public virtual int Drive(int miles, int speed) { /* Method statements here */ return 1; }

    // Derived classes must implement this.
    public abstract double GetTopSpeed();
}
```

после определения метод еще надо вызвать, чтобы он выполнил свою работу.

Вызов метода

Перегрузка метода

```
class TestMotorcycle : Motorcycle
{
    public override double GetTopSpeed()
    {
        return 108.4;
    }

    static void Main()
    {
        TestMotorcycle moto = new TestMotorcycle();

        moto.StartEngine();
        moto.AddGas(15);
        moto.Drive(5, 20);
        double speed = moto.GetTopSpeed();
        Console.WriteLine("My top speed is {0}", speed);
    }
}
```

Экземпляр класса

Передача параметров в подпрограмму

Передача параметров по значению (обычный способ передачи параметров)

```
void Increment(int n)
{
    n++;
    Console.WriteLine($"Число в методе Increment: {n}");
}

int number = 5;
Console.WriteLine($"Число до метода Increment: {number}");
Increment(number);
Console.WriteLine($"Число после метода Increment: {number}");
```

Число до метода Increment: 5

Число в методе Increment: 6

Число после метода Increment: 5

Передача параметров в подпрограмму

Передача параметров по ссылке и модификатор **ref**

```
void Increment(ref int n)
{
    n++;
    Console.WriteLine($"Число в методе Increment: {n}");
}

int number = 5;
Console.WriteLine($"Число до метода Increment: {number}");
Increment(ref number);
Console.WriteLine($"Число после метода Increment: {number}");
```

```
Число до метода Increment: 5
Число в методе Increment: 6
Число после метода Increment: 6
```

Передача параметров в подпрограмму

Выходные параметры. Модификатор out

Параметры могут быть как входными, так и выходными. Чтобы сделать параметр выходным, перед ним ставится **модификатор out**:

```
1 void Sum(int x, int y, out int result)
2 {
3     result = x + y;
4 }
5
6 int number;
7
8 Sum(10, 15, out number);
9
10 Console.WriteLine(number); // 25
```

Здесь результат возвращается не через оператор return, а через выходной параметр result. Причем, как и в случае с ref ключевое слово out используется как при определении метода, так и при его вызове. Плюс использования подобных параметров состоит в том, что по сути мы можем вернуть из метода не одно значение, а несколько.

Передача параметров в подпрограмму

Массив параметров и ключевое слово params

используя ключевое слово `params`, мы можем передавать неопределенное количество параметров:

```
1 void Sum(int initialValue, params int[] numbers)
2 {
3     int result = initialValue;
4     foreach (var n in numbers)
5     {
6         result += n;
7     }
8     Console.WriteLine(result);
9 }
10
11 int[] nums = { 1, 2, 3, 4, 5};
12 Sum(10, nums); // число 10 - передается параметру initialValue
13 Sum(1, 2, 3, 4);
14 Sum(1, 2, 3);
15 Sum(20);
```

При вызове метода на место параметра с модификатором `params` мы можем передать как отдельные значения, так и массив значений, либо вообще не передавать параметры. Количество передаваемых значений в метод неопределено, однако все эти значения должны соответствовать типу параметра с `params`.

Массивы

тип_переменной[] название_массива;

Здесь вначале мы объявили массив `nums`, который будет хранить данные типа `int`. Далее используя операцию `new`, мы выделили память для 4 элементов массива: `new int[4]`. Число 4 еще называется длиной массива. При таком определении все элементы получают значение по умолчанию, которое предусмотрено для их типа. Для типа `int` значение по умолчанию 0.

```
int[] nums = new int[4];
```

Также мы сразу можем указать значения для этих элементов. Все перечисленные выше способы будут равноценны:

```
int[] nums2 = new int[4] { 1, 2, 3, 5 };
```

```
int[] nums3 = new int[] { 1, 2, 3, 5 };
```

```
int[] nums4 = new[] { 1, 2, 3, 5 };
```

```
int[] nums5 = { 1, 2, 3, 5 };
```

Массивы

Примеры разных типов:

массив значений типа string:

```
string[] people = { "Tom", "Sam", "Bob" };
```

Начиная с версии C# 12 для определения массивов можно использовать выражения коллекций, которые предполагают заключение элементов массива в квадратные скобки:

```
int[] nums1 = [ 1, 2, 3, 5 ];  
int[] nums2 = [];    // пустой массив
```

Индексы и получение элементов массива

Используя индексы, мы можем получить элементы массива

```
1  int[] numbers = { 1, 2, 3, 5 };
2
3  // получение элемента массива
4  Console.WriteLine(numbers[3]);  // 5
5
6  // получение элемента массива в переменную
7  var n = numbers[1];           // 2
8  Console.WriteLine(n);         // 2
```

Также мы можем изменить элемент массива по индексу:

```
1  int[] numbers = { 1, 2, 3, 5 };
2
3  // изменим второй элемент массива
4  numbers[1] = 505;
5
6  Console.WriteLine(numbers[1]);  // 505
```

Свойство Length и длина массива

```
1 int[] numbers = { 1, 2, 3, 5 };  
2  
3 Console.WriteLine(numbers.Length); // 4
```

Получение элементов с конца массива

```
1 int[] numbers = { 1, 2, 3, 5};  
2  
3 Console.WriteLine(numbers[numbers.Length - 1]); // 5 - первый с конца или последний элемент  
4 Console.WriteLine(numbers[numbers.Length - 2]); // 3 - второй с конца или предпоследний элемент  
5 Console.WriteLine(numbers[numbers.Length - 3]); // 2 - третий элемент с конца
```

Перебор массивов

Используются foreach, цикл
for, цикл while

```
1 int[] numbers = { 1, 2, 3, 4, 5 };  
2 foreach (int i in numbers)  
3 {  
4     Console.WriteLine(i);  
5 }
```

Многомерные массивы и массивы массивов

Массивы характеризуются таким понятием как ранг или количество измерений. Массивы которые имеют два измерения (ранг равен 2) называют двухмерными

```
int[] nums1 = new int[] { 0, 1, 2, 3, 4, 5 };
```

```
int[,] nums2 = { { 0, 1, 2 }, { 3, 4, 5 } };
```

Поскольку массив nums2 двухмерный, он представляет собой простую таблицу:

Двухмерный массив nums2

0	1	2
3	4	5

Массивы могут иметь и большее количество измерений. Объявление трехмерного массива могло бы выглядеть так:

```
int[,,] nums3 = new int[2, 3, 4];
```

Все возможные способы определения двумерных массивов:

```
1 int[, ] nums1;  
2 int[, ] nums2 = new int[2, 3];  
3 int[, ] nums3 = new int[2, 3] { { 0, 1, 2 }, { 3, 4, 5 } };  
4 int[, ] nums4 = new int[, ] { { 0, 1, 2 }, { 3, 4, 5 } };  
5 int[, ] nums5 = new [, ] { { 0, 1, 2 }, { 3, 4, 5 } };  
6 int[, ] nums6 = { { 0, 1, 2 }, { 3, 4, 5 } };
```

Определенную сложность может представлять перебор многомерного массива. Прежде всего надо учитывать, что длина такого массива - это совокупное количество элементов.

```
int[, ] numbers = { { 1, 2, 3 }, { 4, 5, 6 } };  
foreach (int i in numbers)  
    Console.WriteLine($"{i} ");
```

1 2 3 4 5 6

Массивы массивов (“зубчатый массив”)

От многомерных массивов надо отличать массив массивов или так называемый "зубчатый массив"

```
int[][] nums = new int[3][];  
nums[0] = new int[2] { 1, 2 };           // выделяем память для первого подмассива  
nums[1] = new int[3] { 1, 2, 3 };        // выделяем память для второго подмассива  
nums[2] = new int[5] { 1, 2, 3, 4, 5 }; // выделяем память для третьего подмассива
```

Зубчатый массив nums

1	2			
1	2	3		
1	2	3	4	5

Основные понятия массивов:

- **Ранг (rank)**: количество измерений массива
- **Длина измерения (dimension length)**: длина отдельного измерения массива
- **Длина массива (array length)**: количество всех элементов массива

Например, возьмем массив

```
int[,] numbers = new int[3, 4];
```

Массив numbers двухмерный, то есть он имеет два измерения, поэтому его ранг равен 2. Длина первого измерения - 3, длина второго измерения - 4. Длина массива (то есть общее количество элементов) - 12.

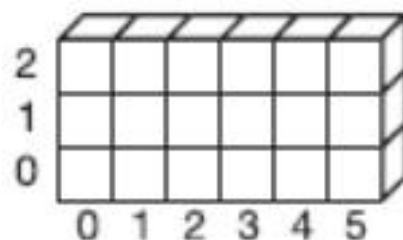
Примеры массивов:

Одномерный массив

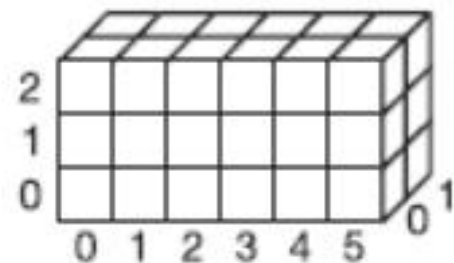


Одномерный массив
`int[5]`

Многомерные массивы

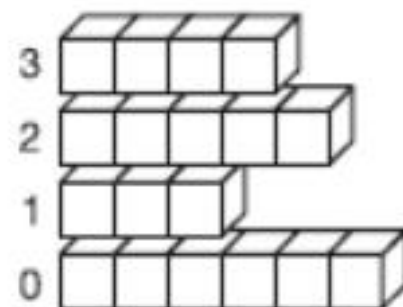


Двухмерный массив
`int[3,6]`



Трёхмерный массив
`int[3,6,2]`

Зубчатый массив



Зубчатый массив
`int[4][]`