CMC UNIVERSITY

Chapter 2: Basic Declaration in C#

Nguyen Tien Dong

Content

CMC UNIVERSITY

2.1 Class

CMC UNIVERSITY

2.1.1 Definition

2.1.2 Declaration

2.1 Class

CMC UNIVERSITY

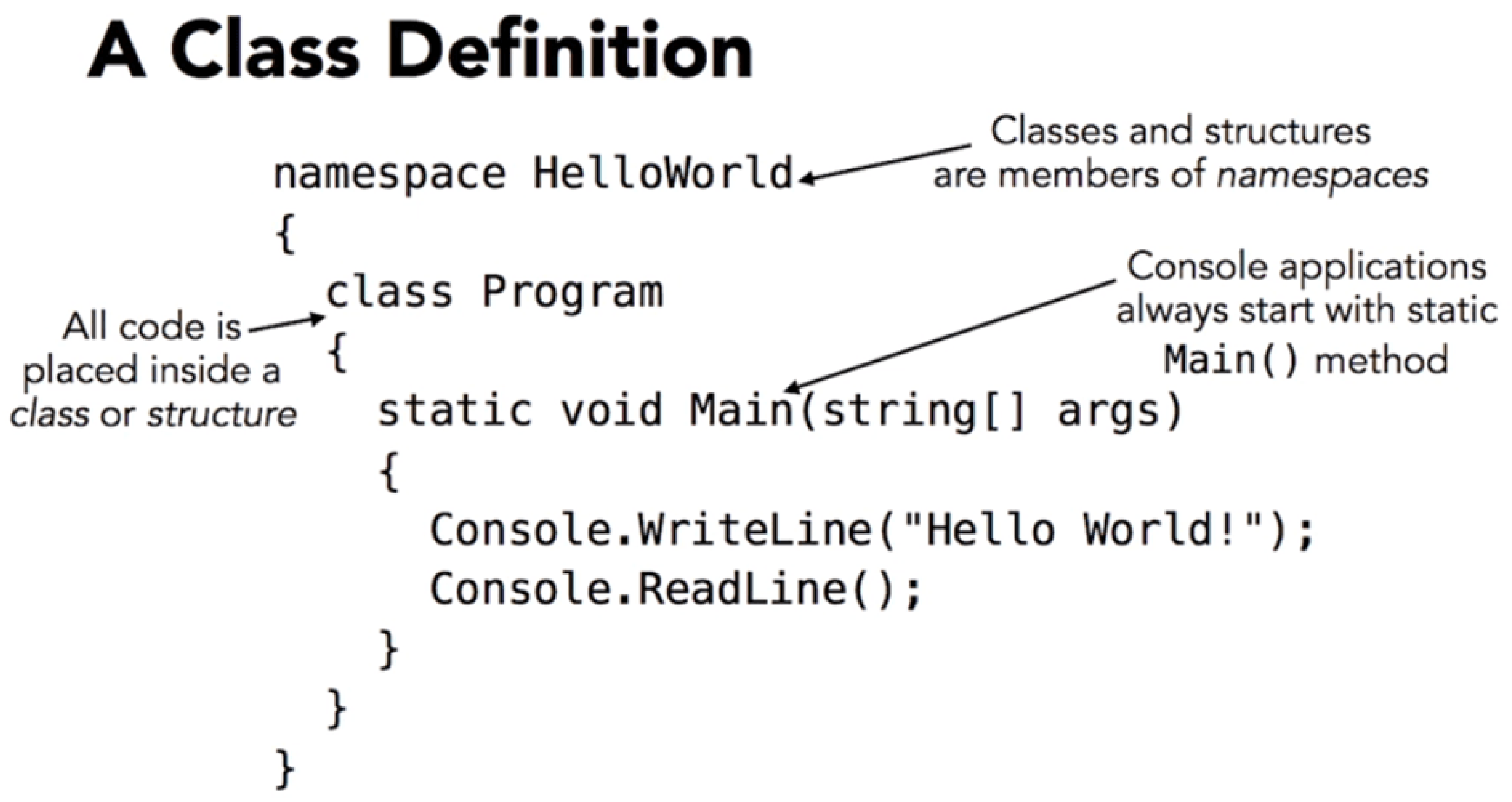
Definition

- **A Class** is an abstract description of an object
- For example, A customer class would describe the typical attributes and actions of a customer. A customer would have a name, address, phone, etc.
- An object is a particular instance of the class in your program, for instance of the class in your program, for instance the customer Robert Jones who lives at 121 Dexter bld. With the phone number 206.55.3212

2.1 Class

Declaration

- In C#, Classes can be contained in a Namespace, but everything else must be contained in a class


A Class Definition
Classes and structures are members of *namespaces*
All code is placed inside a *class or structure*
Console applications always start with static Main() method

```
namespace HelloWorld
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Hello World!");
      Console.ReadLine();
    }
  }
}
```

2.2 Variable and Constant

2.2.1 Variable

- Definition
- Declaration
- Initialization

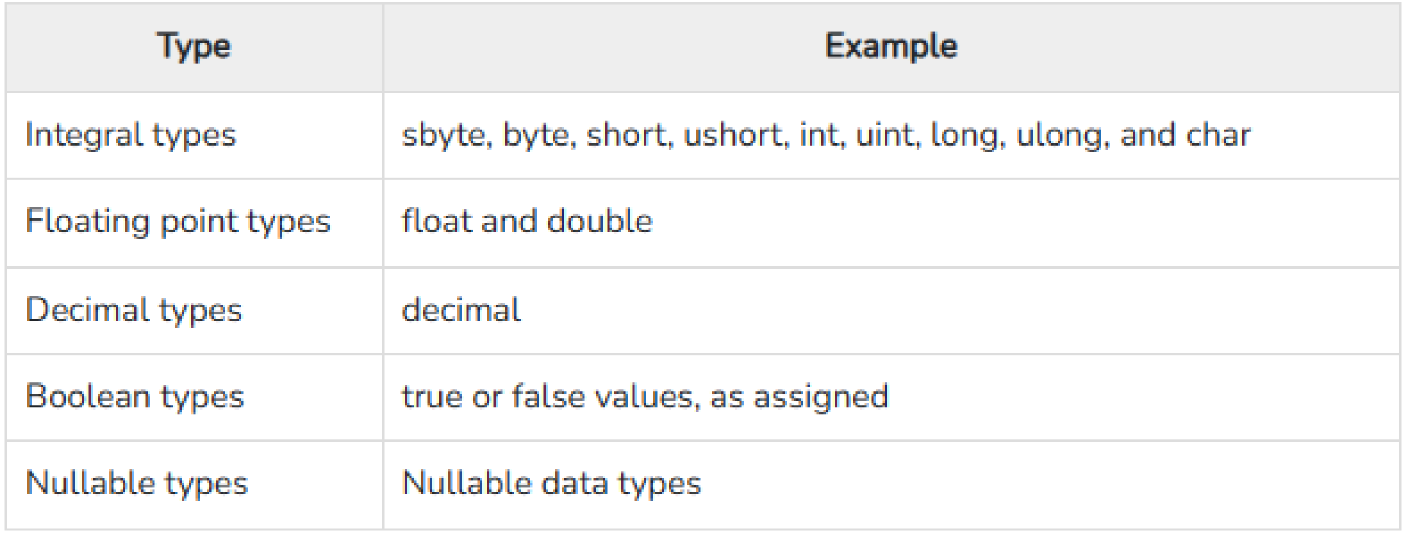
2.1.2 Constant

- Definition
- Constant Types
- Declaration

2.2 Variable and Constant

Definition

- **A variable** is nothing but a name given to a storage area that our programs can manipulate. Each variable in C# has a specific type, which determines the size and layout of the variable's memory the range of values that can be stored within that memory and the set of operations that can be applied to the variable.



Type	Example
Integral types	sbyte, byte, short, ushort, int, uint, long, ulong, and char
Floating point types	float and double
Decimal types	decimal
Boolean types	true or false values, as assigned
Nullable types	Nullable data types

2.2 Variable and Constant

Declaration

- Syntax for variable definition in C#

```
<data_type> <variable_list>;
```

- Data_type must be a valid C# data type including char, int, float, double, or any user-defined data type, and variable_list may consist of one or more identifier names separated by commas.
- Example

```
int i, j, k;
char c,ch;
float f,salary;
double d;
int i = 100;
```

2.2 Variable and Constant

Initailization

- **Intitializing Variables**:Variables are initialized (assigned a value) with an equal sign followed by a constant expression. The general form of initialization is

```
variable_name = value;
```

- Variables can be initialized in their declaration. The initializer consists of an equal sign followed by a constant expression as

```
<data_type> <variable_name> = value;
```

- Example

```
int d = 3, f = 5;    /* initializing d and f. */
byte z = 22;         /* initializes z. */
double pi = 3.14159; /* declares an approximation of pi. */
char x = 'x';        /* the variable x has the value 'x'. */
```

2.2 Variable and Constant

Definition

- **The constants** refer to fixed values that the program may not alter during its execution. These fixed values are also called literals. Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal. There are also enumeration constants as well.
- The constants are treated just like regular variables except that their values cannot be modified after their definition.

2.2 Variable and Constant

Constant Types:

- **Integer literal** can be a decimal, or hexadecimal constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, and there is no prefix id for decimal.
- An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order
- Example:

```
85        /* decimal */
0x4b       /* hexadecimal */
30        /* int */
30u        /* unsigned int */
30l       /* long */
30ul       /* unsigned long */
```

2.2 Variable and Constant

CMC UNIVERSITY

Constant Types:

- A **floating-point literal** has an integer part, a decimal point, a fractional part, and an exponent part

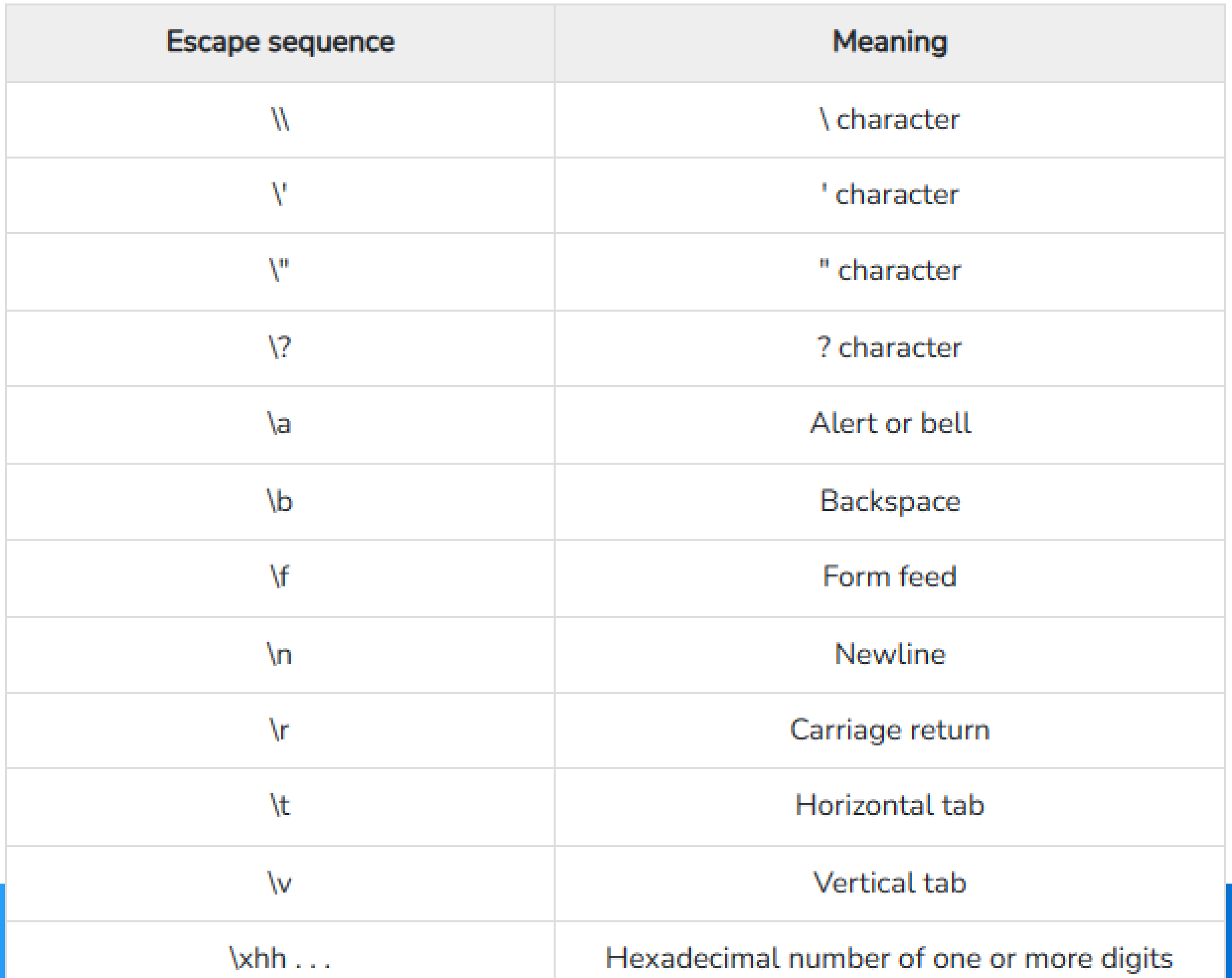
Example:

```
3.14159       /* Legal */
314159E-5F    /* Legal */
510E          /* Illegal: incomplete exponent */
210f          /* Illegal: no decimal or exponent */
.e55          /* Illegal: missing integer or fraction */
```

2.2 Variable and Constant

Constant Types:

- **Character literals** are enclosed in single quotes. For example, 'x' and can be stored in a simple variable of char type. A character literal can be a plain character (such as 'x'), an escape sequence (such as '\t'), or a universal character (such as '\u02C0').



Escape sequence	Meaning
\\	\ character
\'	' character
\"	" character
\?	? character
\a	Alert or bell
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\xhh . . .	Hexadecimal number of one or more digits

Constant Types

- **String literals** or constants are enclosed in double quotes "" or with @"". A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters.
- Example

```
"hello, dear"
"hello, \
dear"
"hello, " "d" "ear"
@"hello dear"
```

2.2 Variable and Constant

CMC UNIVERSITY

Declaration

- Constants are defined using the **const** keyword. Syntax for defining a constant"hello, dear"

```
const <data_type> <constant_name> = value;
```

- Example:

```
const int MaxValue = 100;
const double Pi = 3.14159;
const string Greeting = "Hello, World!";
const bool IsEnabled = true;
const char FirstLetter = 'A';
```

2.3 Array

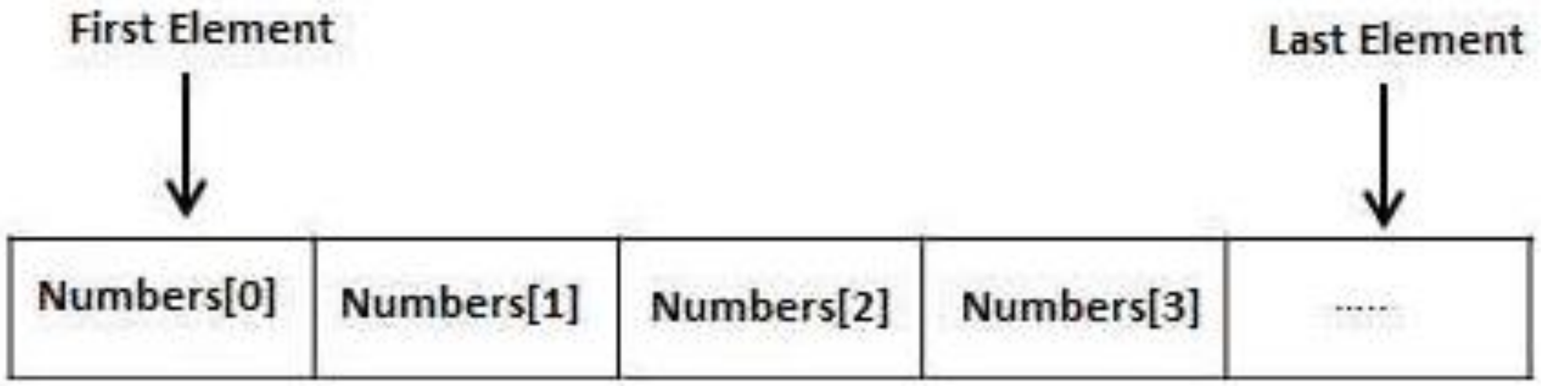
2.2.1 Array

- Definition
- Declaration
- Assigning
- Accessing
- Iteration

CMC UNIVERSITY

Definition

- **An array** stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type stored at contiguous memory locations
- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



2.3 Array

Declaration

- Syntax of an array:

```
datatype[] arrayName;
```

Where:

- datatype is used to specify the type of elements in the array.
- [ ] specifies the rank of the array. The rank specifies the size of the array.
- arrayName specifies the name of the array.

2.3 Array

CMC UNIVERSITY

Declaration

- Declaring an array does NOT initialize the array in the memory. When the array variable is initialized, you can assign values to the array.
- Array is a reference type, use the **new** keyword to create an instance of the array.
- Example

```
        int[] numbers = new int[5];//array of integers
        double[] balance = new double[10];//array of double
        string[] fruits = { "Apple", "Banana", "Orange", "Mango" }; //array of
strings
        int[,] matrix = new int[3, 4];//Multidimensional array
```

2.3 Array

Assign

- Can assign values to individual array elements, by using the index number, like:

```
[] balance = new double[10];
balance[0] = 4500.0;
```

- Assign values to the array at the time of declaration

```
double[] balance = { 2340.0, 4523.69, 3421.0};
```

- Create and Initialize an array

```
int [] marks = new int[5]  { 99,  98, 92, 97, 95};
```

Accessing

- An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array.
- For example:

```
double salary = balance[9];
```

CMC UNIVERSITY

Interatation

- using For loop for accessing each array element. You can also use a **foreach** statement to iterate through an array.

```
using System;
namespace ArrayApplication {
   class MyArray {
      static void Main(string[] args) {
         int []  n = new int[10]; /* n is an array of 10 integers */
         /* initialize elements of array n */
         for ( int i = 0; i < 10; i++ ) {
            n[i] = i + 100;
         }
         /* output each array element's value */
         foreach (int j in n ) {
            int i = j-100;
            Console.WriteLine("Element[{0}] = {1}", i, j);
         }
         Console.ReadKey();
      }
   }
}
```

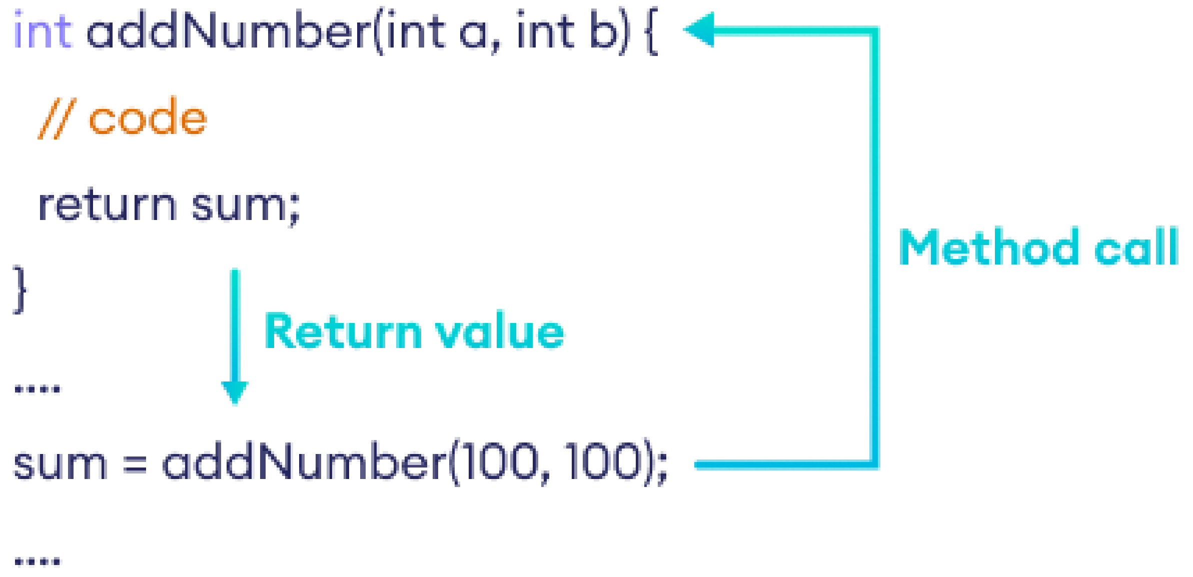
2.4 Method

- Definition
- Declaration

2.4 Method

Definition

- A method is a group of statements that together perform a task. Every C# program has at least one class with a method named Main.
- To use a method:
  - Define the method
  - Call the method



```
int addNumber(int a, int b) {
  // code
  return sum;
}
....
sum = addNumber(100, 100);
....
```

2.4 Method

CMC UNIVERSITY

Declaration

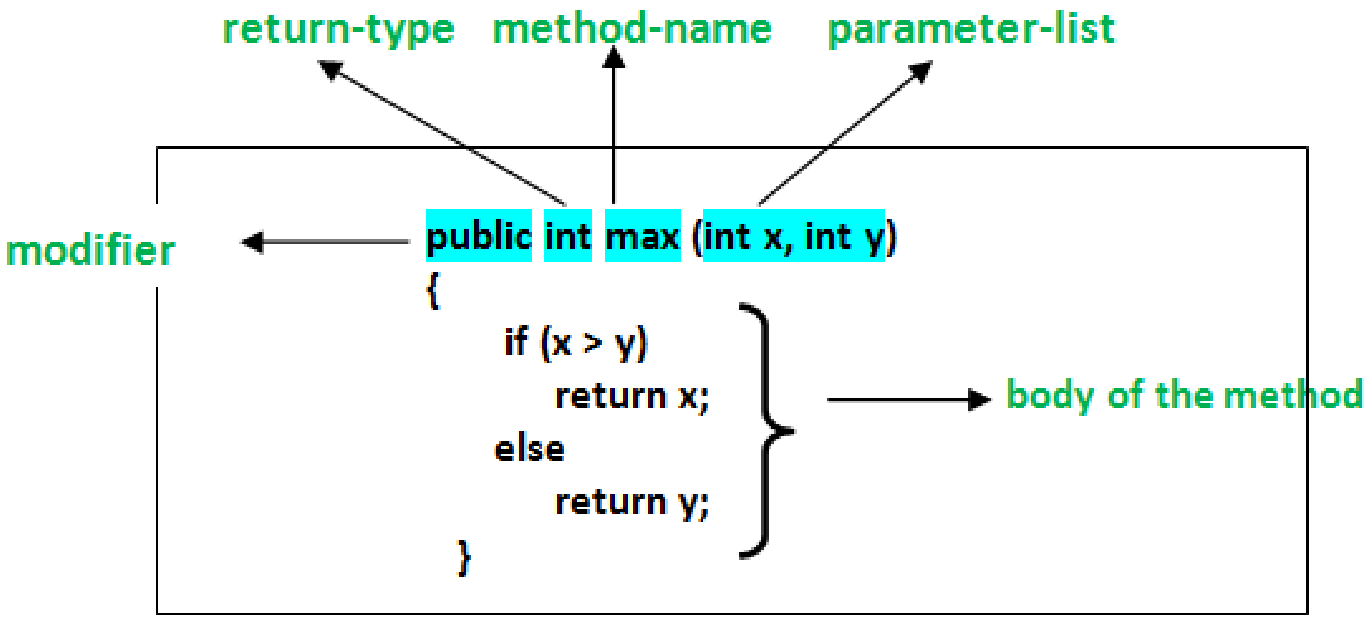
- Syntax of Method:

<Access Specifier> <Return Type> <Method Name>(Parameter List)

{

Method Body

}



```
public int max (int x, int y)
{
    if (x > y)
        return x;
    else
        return y;
}
```

2.4 Method

Access Specifier – This determines the visibility of a variable or a method from another class.

Return type – A method may return a value. The return type is the data type of the value the method returns. If the method is not returning any values, then the return type is void.

Method name – Method name is a unique identifier and it is case sensitive. It cannot be same as any other identifier declared in the class.

Parameter list – Enclosed between parentheses, the parameters are used to pass and receive data from a method. The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.

Method body – This contains the set of instructions needed to complete the required activity.

2.4 Method

Example:**AddNumbers**

```
using System;
class Program
{
    static void Main()
    {
        Console.WriteLine("Enter two numbers to find their sum:");
        int num1 = Convert.ToInt32(Console.ReadLine());
        int num2 = Convert.ToInt32(Console.ReadLine());

        int sum = AddNumbers(num1, num2);

        Console.WriteLine("The sum of {0} and {1} is {2}.", num1, num2, sum);
    }

    static int AddNumbers(int a, int b)
    {
        int result = a + b;
        return result;
    }
}
```

2.4 Method

Example:**CountWords, VowelCount**

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("Enter a sentence:");
        string sentence = Console.ReadLine();

        int wordCount = CountWords(sentence);
        int vowelCount = CountVowels(sentence);

        Console.WriteLine("Word Count: {0}", wordCount);
        Console.WriteLine("Vowel Count: {0}", vowelCount);
    }

    static int CountWords(string text)
    {
        string[] words = text.Split(new char[] { ' ', '.', ',', ';', '!', '?' },
StringSplitOptions.RemoveEmptyEntries);
        return words.Length;
    }
```

```
    static int CountVowls(string text)
    {
        int count = 0;
        string vowels = "aeiouAEIOU";

        foreach (char c in text)
        {
            if (vowels.Contains(c))
            {
                count++;
            }
        }

        return count;
    }
}
}
```

2.4 Exercises

Exercise 1:

- Declare a variable named "celsius" and assign a value of 25 to it.
- Declare a constant named "FAHRENHEIT_RATIO" and assign a value of 9/5.
- Declare a variable named "fahrenheit" and calculate its value by multiplying "celsius" with "FAHRENHEIT_RATIO" and adding 32.
- Display the converted temperature in Fahrenheit in the console.

Example:

Temperature Conversion:

Celsius: 25

Fahrenheit: 77

2.4 Exercises

Exercise 2:

- Declare an integer array named "numbers" with the following elements: 7, 2, 9, 1, 5.
- Declare a variable named "maxNumber" and initialize it with the first element of the "numbers" array.
- Iterate over the elements of the "numbers" array and update the value of "maxNumber" if a larger element is found.
- Display the maximum number in the console.

Example:

Temperature Conversion:

Finding the Maximum Number:

Numbers: 7, 2, 9, 1, 5

Maximum Number: 9

2.4 Exercises

Exercise 3:

- Declare a method named "CalculateRectangleArea" that takes two parameters: width (double) and height (double). This method should calculate and return the area of a rectangle using the formula: area = width * height.Declare a variable named "maxNumber" and initialize it with the first element of the "numbers" array.
- Declare a method named "CalculateCircleArea" that takes one parameter: radius (double). This method should calculate and return the area of a circle using the formula: area = π * radius * radius. You can use the Math.PI constant to represent πDisplay the maximum number in the console.
- In the Main method, prompt the user to enter the dimensions of a rectangle (width and height) and store the input values in variables.
- Call the "CalculateRectangleArea" method with the provided dimensions and store the returned area in a variable.
- Display the calculated rectangle area in the console.
- Prompt the user to enter the radius of a circle and store the input value in a variable.
- Call the "CalculateCircleArea" method with the provided radius and store the returned area in a variable.
- Display the calculated circle area in the console.

2.4 Exercises

Exercise 3:

- Example:

```
Area Calculation:
Enter the dimensions of a rectangle:
Width: 4
Height: 6
Rectangle Area: 24
Enter the radius of a circle:
Radius: 3
Circle Area: 28.2743338823081
```

THANK YOU