Object Oriented Programming

3. C# Methods (Functions)

Lecturer:

Sipan M. Hameed

www.sipan.dev

2024 - 2025

Contents

3. C# Methods (Functions)

3.1. Introduction

C#, Method is a separate code block and that contains a series of statements to perform particular operations and methods must be declared either in class or struct by specifying the required parameters.

Generally, in c# Methods are useful to improve the code reusability by reducing the code duplication. Suppose if we have the same functionality to perform in multiple places, then we can create one method with the required functionality and use it wherever it is required in the application.Methods in C# are defined like this:

3.1.1 Syntax:

```
<Access Specifier> <Return Type> <Method Name>(Parameter List)
{
   Method Body
}
```

To use a method, you need to −

- Define the method
- Call the method

Following are the various elements of a 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.

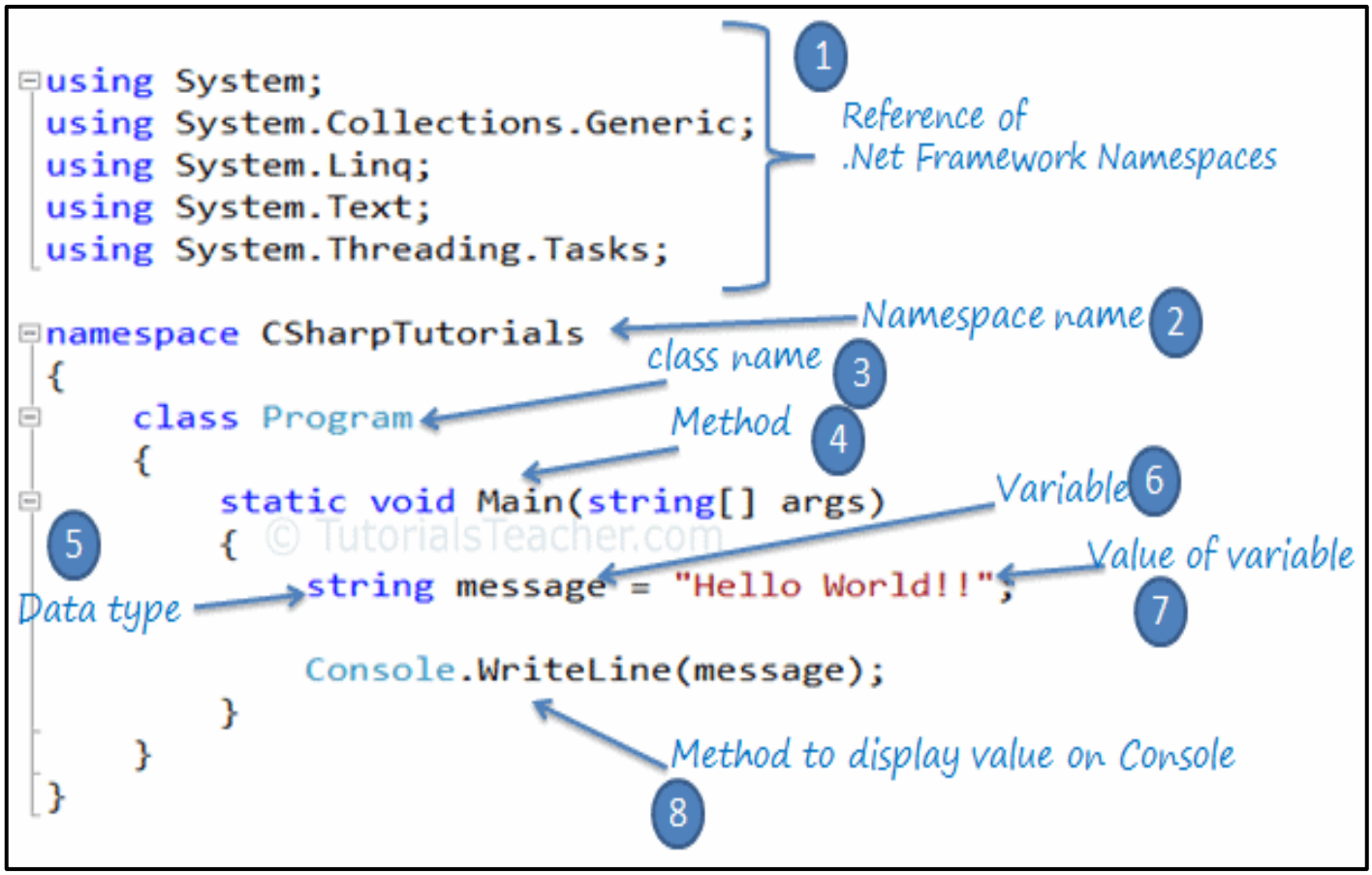
3.1.2 Example: C# Console Application illustration

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CSharpTutorials
{
    class Program
    {
        static void Main(string[] args)
        {
            string message = "Hello World!!";

            Console.WriteLine(message);
        }
    }
}
```

The following image illustrates the parts of the C# program:



Remember - C# - Data Types

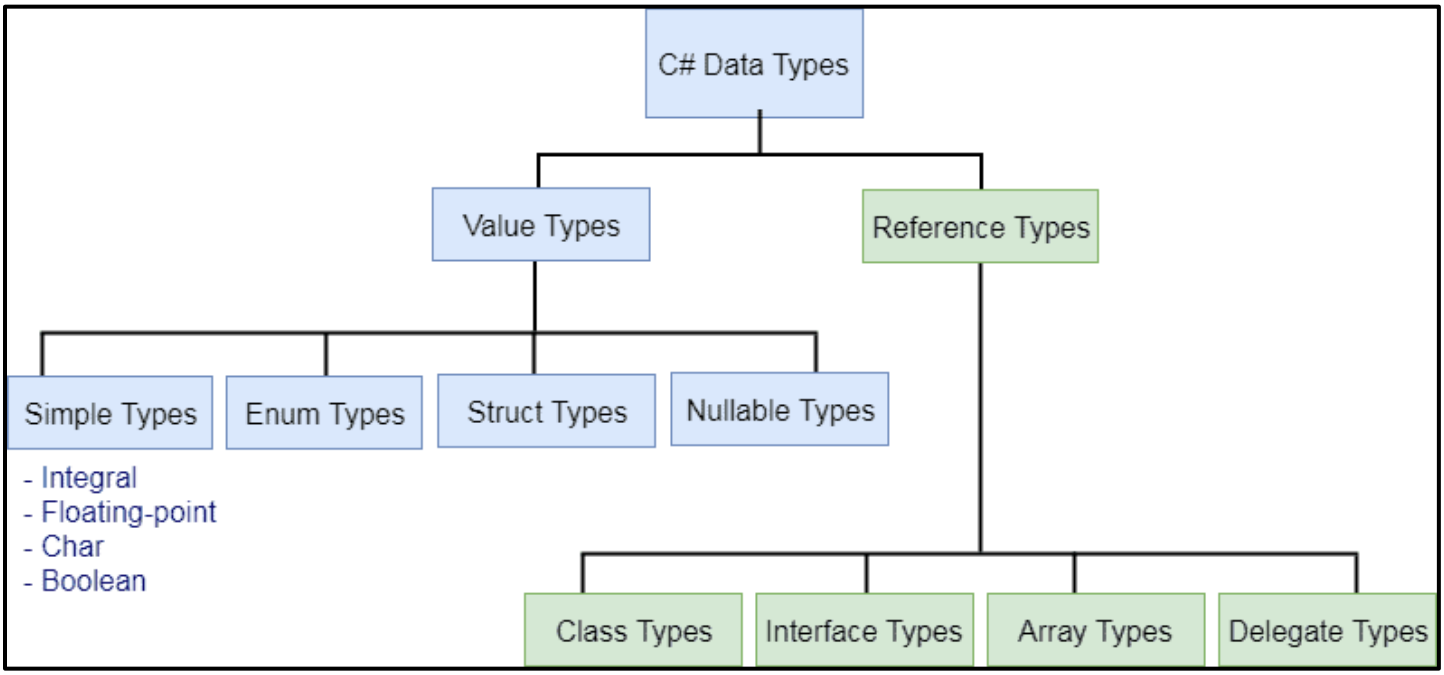
C# is a strongly-typed language. It means we must declare the type of a variable that indicates the kind of values it is going to store, such as integer, float, decimal, text, etc.

The following declares and initialized variables of different data types.

Example: Variables of Different Data Types

```
string stringVar = "Hello World!!";
int intVar = 100;
float floatVar = 10.2f;
char charVar = 'A';
bool boolVar = true;
```

C# mainly categorized data types in two types: Value types and Reference types. Value types include simple types (such as int, float, bool, and char), enum types, struct types, and Nullable value types. Reference types include class types, interface types, delegate types, and array types. Learn about value types and reference types in detail in the next chapter.



Remember - Predefined Data Types in C#

C# includes some predefined value types and reference types. The following table lists predefined data types:

Type	Description	Range	Suffix
byte	8-bit unsigned integer	0 to 255	
sbyte	8-bit signed integer	-128 to 127	
short	16-bit signed integer	-32,768 to 32,767	
ushort	16-bit unsigned integer	0 to 65,535	
int	32-bit signed integer	-2,147,483,648 to 2,147,483,647	
uint	32-bit unsigned integer	0 to 4,294,967,295	u
long	64-bit signed integer	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	l
ulong	64-bit unsigned integer	0 to 18,446,744,073,709,551,615	ul
float	32-bit Single-precision floating point type	-3.402823e38 to 3.402823e38	f
double	64-bit double-precision floating point type	-1.79769313486232e308 to 1.79769313486232e308	d
decimal	128-bit decimal type for financial and monetary calculations	(+ or -)1.0 x 10e-28 to 7.9 x 10e28	m
char	16-bit single Unicode character	Any valid character, e.g. a,*, \x0058 (hex), or\u0058 (Unicode)	
bool	8-bit logical true/false value	True or False	
object	Base type of all other types.		
string	A sequence of Unicode characters		
DateTime	Represents date and time	0:00:00am 1/1/01 to 11:59:59pm 12/31/9999	

3.1.3 Example-1:

Inside `Main()`, call the `myMethod()` method:

```
using System;

namespace MyApplication
{
    class Program
    {
        static void MyMethod()
        {
            Console.WriteLine("I just got executed!");
        }

        static void Main(string[] args)
        {
            MyMethod();
             Console.ReadKey();
        }
    }
}
```

Outputs:

```
I just got executed!
```

3.1.4 Example-2:

A method can be called multiple times:

```
using System;

namespace MyApplication
{
    class Program
    {
        static void MyMethod()
        {
            Console.WriteLine("I just got executed!");
        }

        static void Main(string[] args)
        {
            MyMethod();
            MyMethod();
            MyMethod();
            Console.ReadKey();
        }
    }
}
```

Output: -

```
I just got executed!
I just got executed!
I just got executed!
```

3.2. Types of Methods

The method is a block of code that contains a series of the statement.

Different types of methods in C# programming:

1. Parameter-less method
2. Parameterized method
3. Parameter and return method
4. Call by value method
5. Reference method
6. Out parameter method
7. Method overriding
8. Method overloading
9. Method hiding
10. Abstract method
11. Virtual method
12. Static method

3.2.1 parameter-less method

A method which does not have any parameter is called the parameter-less method. It is a block of code with the statement.

```
using System;

namespace MethodExamples
{
    class Parameter
    {
        protected void Message()
        {
            Console.WriteLine("Hello World!");
        }

        static void Main()
        {
            var p = new Parameter();
            p.Message();
            Console.ReadLine();
        }
    }
}
```

Output: -

```
Hello World!
```

3.2.2 parameterized method

A method which contains at least one or more parameters is called parameterized method.

```
using System;

namespace MethodExamples
{
    class ParameterMethod
    {
        public void PrintMessage(string message)
        {
            Console.WriteLine("Hello " + message);
        }

        static void Main()
        {
            var p = new ParameterMethod();
            p.PrintMessage("Zaxo IT Student");
            Console.ReadLine();
        }
    }
}
```

Output: -

```
Hello IT Student
```

3.2.3 return value from Method

You can call a method using the name of the method. The following example illustrates this −

```
using System;
namespace MethodExamples
{
    class Numbermax
    {
        public int FindMax(int num1, int num2)
        {
            /* local variable declaration */
            int result;

            if (num1 > num2)
                result = num1;
            else
                result = num2;
            return result;
        }

        static void Main(string[] args)
        {
            /* local variable definition */
            int a = 100;
            int b = 200;
            int ret;
            Numbermax n = new Numbermax();

            //calling the FindMax method
            ret = n.FindMax(a, b);
            Console.WriteLine("Max value is : {0}", ret);
            Console.ReadLine();
        }
    }
}
```

Output: -

```
Max value is : 200
```

3.2.4 Method parameters vs. arguments

The method definition specifies the names and types of any parameters that are required. When calling code calls the method, it provides concrete values called arguments for each parameter. The arguments must be compatible with the parameter type but the argument name (if any) used in the calling code doesn't have to be the same as the parameter named defined in the method. For example:

```
using System;
namespace MethodExamples
{
    class Number
    {
        static  int Square(int i)
        {
            // Store input argument in a local variable.
            int input = i;
            return input * input;
        }

        static void Main(string[] args)
        {
            int numA = 4;
           //int numG = Number.
            // Call with an int variable.
            int productA = Number.Square(numA);

            int numB = 32;
            // Call with another int variable.
            int productB = Square(numB);

            // Call with an integer literal.
            int productC = Square(12);

            Console.WriteLine("square of 12 = {0}", productC);


            // Call with an expression that evaulates to int.
            productC = Square(productA * 3);

            Console.ReadKey();
        }


    }
}
```

Output:-

```
square of 12 = 144
```

Another Example: -

```
using System;

namespace MyApplication
{
  class Program
  {
    static void MyMethod(string fname)
    {
      Console.WriteLine(fname + " Refsnes");
    }

    static void Main(string[] args)
    {
      MyMethod("Liam");
      MyMethod("Jenny");
      MyMethod("Anja");
    }
  }
}
```

Output: -

```
Liam Refsnes
Jenny Refsnes
Anja Refsnes
```

3.3. Practical Examples

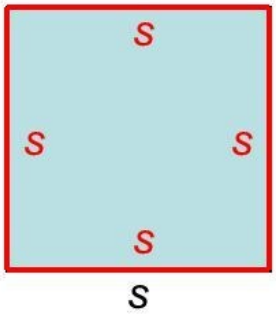
PERIMETER, CIRCUMFERENCE, AND AREA FORMULAS

SQUARE

side length s

$P = 4s$

$A = s^2$

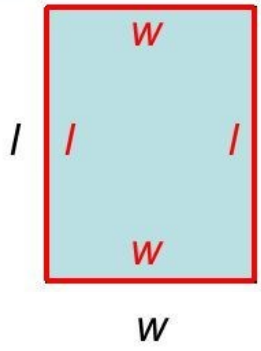


RECTANGLE

length l and width w

$P = 2l + 2w$

$A = lw$

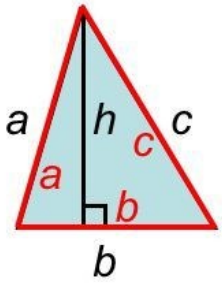


TRIANGLE

side lengths a, b, and c, base b, and height h

$P = a + b + c$

$A = \frac{1}{2}bh$



CIRCLE

radius r

$C = 2\pi r$

$A = \pi r^2$

3.3.1 Example: -

How to calculate Area and Perimeter of a circle using methods

```
using System;
namespace MethodExamples
{
    class math
    {
        static double per(double r)
        {
            return 2 * 3.14 * r;
        }

        static double area(double r)
        {
            return 3.14 * r * r;
        }
        static void Main()
        {
            double r = 2.3;

            double a = area(r);
            double p = per(r);
            Console.WriteLine("area = {0}" ,a);
            Console.WriteLine("perimeter = " + p);

            Console.ReadKey();
        }
    }
}
```

Output: -

```
area = 16.6106
perimeter = 14.444
```

3.3.2 Example-

How to write a program to calculate **square**, **cube** and power (X^n) of numbers using methods for each calculation

```
using System;
namespace MethodExamples
{
    class math
    {
        static double square(double a)
        {
            return a * a;
        }

        static double cube(double x)
        {
            return x * x;
        }

        static double powerXN(double x,double n)
        {
            double p = 1;

            for(int i=1;i<=n;i++)
            {
                p = p * x;
            }

            return p;
        }

        static void Main()
        {
            double num1 = 7;

            Console.WriteLine(square(num1));

            double num2 = cube(33);
            Console.WriteLine(num2);

            double a = 3, b = 5;
            double z = powerXN(a, b);
            Console.WriteLine(z);

            Console.ReadKey();
        }
    }
}
```

Output: -

```
49
1089
243
```