Methods in C#

Școala Informală de IT

Cluj-Napoca

Agenda

- Using Methods
  - What is a Method? Why to Use Methods?
  - Declaring and Creating Methods
  - Calling Methods
- Methods with Parameters
  - Passing Parameters
  - Returning Values
- Best Practices

Școala informală de IT

What is a Method?

- A method is a kind of building block that solves a small problem
  - A piece of code that has a name and can be called from the other code
  - Can take parameters and return a value
- Methods allow programmers to construct large programs from simple pieces
- Methods are also known as functions or procedures

Why to Use Methods?

- More manageable programming
  - Split large problems into small pieces
  - Better organization of the program
  - Improve code readability
  - Improve code understandability
- Avoiding repeating code
  - Improve code maintainability
- Code reusability
  - Using existing methods several times

Școala
informală
de IT

Declaring and Creating Methods

Declaring and Creating Methods

```
static void PrintLogo()
{
    Console.WriteLine("Scoala Informala");
    Console.WriteLine("www.scoalainformala.ro");
}
```

Method name

Method body

- Each method has a name
  - It is used to call the method
  - Describes its purpose

Școala informală de IT

Declaring and Creating Methods

- Methods declared `static` can be called by any other method (static or not)
  - This will be discussed later in details
- The keyword `void` means that the method does not return any result
- Each method has a body
  - It contains the programming code
  - Surrounded by `{` and `}`

Declaring and Creating Methods

```
class MethodExample
{
    static void PrintLogo()
    {
        Console.WriteLine("Scoala Informala");
        Console.WriteLine("www.scoalainformala.ro");
    }
    static void Main()
    {
        // ...
    }
}
```

- Methods are always declared inside a `class`
- `Main()` is also a method like all others

Școala
informală
de IT

Calling Methods

Calling Methods

- To call a method, simply use:
  1. The method's name
  2. Parentheses (don't forget them!)
  3. A semicolon (;)

```
PrintLogo();
```

- This will execute the code in the method's body and will result in printing the following:

```
Scoala Informala
www.scoalainformala.ro
```

Calling Methods

- A method can be called from:
  - The `Main()` method

```
static void Main()
{
    // ...
    PrintLogo();
    // ...
}
```

  - Any other method
  - Itself (process known as recursion)

Școala
informală
de IT

Methods with Parameters

Passing Parameters and Returning Values

Școala informală de IT

Method Parameters

- To pass information to a method, you can use parameters (also known as arguments)
  - You can pass zero or several input values
  - You can pass values of different types
  - Each parameter has name and type
  - Parameters are assigned to particular values when the method is called
- Parameters can change the method behavior depending on the passed values

Școala informală de IT

Defining and Using Method Parameters

```
static void PrintSign(int number)
{
    if (number > 0)
        Console.WriteLine("Positive");
    else if (number < 0)
        Console.WriteLine("Negative");
    else
        Console.WriteLine("Zero");
}
```

- Method's behavior depends on its parameters
- Parameters can be of any type
  - `int`, `double`, `string`, etc.
  - Arrays (`int[]`, `double[]`, etc.)

Școala informală de IT

Defining and Using Method Parameters

- **Methods can have as many parameters as needed:**

```
static void PrintMax(float number1, float number2)
{
    float max = number1;
    if (number2 > number1)
        max = number2;
    Console.WriteLine("Maximal number: {0}", max);
}
```

- **The following syntax is not valid:**

```
static void PrintMax(float number1, number2)
```

Calling Methods with Parameters

- To call a method and pass values to its parameters:
  - Use the method's name, followed by a list of expressions for each parameter
- Examples:

```
PrintSign(-5);
PrintSign(balance);
PrintSign(2+3);

PrintMax(100, 200);
PrintMax(oldQuantity * 1.5, quantity * 2);
```

Școala informală de IT

Calling Methods with Parameters

- Expressions must be of the same type as method's parameters (or compatible)
  - If the method requires a `float` expression, you can pass `int` instead
- Use the same order like in method declaration
- For methods with no parameters do not forget the parentheses

Methods Parameters

```
static void PrintSign(int number)
{
  if (number > 0)
    Console.WriteLine("The number {0} is positive.", number);
  else if (number < 0)
    Console.WriteLine("The number {0} is negative.", number);
  else
    Console.WriteLine("The number {0} is zero.", number);
}

static void PrintMax(float number1, float number2)
{
  float max = number1;
  if (number2 > number1)
  {
    max = number2;
  }
  Console.WriteLine("Maximal number: {0}", max);
}
```

Months – Example

- Display the interval between the months, in a user-friendly way

```
class MonthsExample
{
  static void SayMonth(int month)  {
    if ((month < 1) || (month > 12)) {
      Console.WriteLine("Invalid month");
      return;
    }
    string[] monthNames = {
      "January", "February", "March", "April",
      "May", "June", "July", "August", "September",
      "October", "November", "December"};
    Console.Write(monthNames[month-1]);
  }
```

(the example continues)

Months – Example

```
  static void SayPeriod(int startMonth, int endMonth)
  {
    int period = endMonth - startMonth;
    if (period < 0)
    {
      period = period + 12;
      // From December to January the period is 1 month, not -11!
    }
    Console.Write("There are {0} + months from ", period);
    SayMonth(startMonth); Console.Write(" to ");
    SayMonth(endMonth);
  }
}
```

Optional Parameters

- C# supports optional parameters with default values assigned at their declaration:

```
static void PrintNumbers(int start = 0, int end = 100)
{
  for (int i = start; i <= end; i++) {
    Console.Write("{0} ", i);
  }
}
```

- The above method can be called in several ways:

```
PrintNumbers(5, 10);
PrintNumbers(15);
PrintNumbers();
PrintNumbers(end: 40, start: 35);
```

Școala
informală
de IT

Returning Values From Methods

Școala informală de IT

Returning Values From Methods

- A method can return a value to its caller
- Returned value:
  - Can be assigned to a variable:

```
string message = Console.ReadLine();
// Console.ReadLine() returns a string
```

  - Can be used in expressions:

```
float price = GetPrice() * quantity * 1.20;
```

  - Can be passed to another method:

```
int age = int.Parse(Console.ReadLine());
```

Școala informală de IT

Defining Methods That Return a Value

- Instead of `void`, specify the type of data to return

```
static int Multiply(int firstNum, int secondNum)
{
    return firstNum * secondNum;
}
```

- Methods can return any type of data (`int`, `string`, array, etc.)
- `void` methods do not return anything
- The combination of method's name and parameters is called method signature
- Use `return` keyword to return a result

The `return` Statement

- The `return` statement:
  - Immediately terminates method's execution
  - Returns specified expression to the caller
  - Example:

```
return -1;
```

- To terminate `void` method, use just:

```
return;
```

- Return can be used several times in a method body

Școala
informală
de IT

Overloading Methods

Multiple Methods with the Same Name

Overloading Methods

- What means "to overload a method name"?
  - Use the same method name for multiple methods with different signature (parameters)

```
Static void Print(string text) {
  Console.WriteLine(text);
}

static void Print(int number) {
  Console.WriteLine(number);
}

static void Print(string text, int number) {
  Console.WriteLine(text + ' ' + number);
}

static void Print(string text, float number) {
  Console.WriteLine(text + ' ' + number);
}
```

Școala
informală
de IT

Variable Number of Parameters

Variable Number of Parameters

- A method in C# can take variable number of parameters by specifying the `params` keyword

```
static long CalcSum(params int[] elements)
{
   long sum = 0;
   foreach (int element in elements)
      sum += element;
   return sum;
}

static void Main()
{
   Console.WriteLine(CalcSum(2, 5));
   Console.WriteLine(CalcSum(4, 0, -2, 12));
   Console.WriteLine(CalcSum());
}
```

Școala informală de IT

Methods – Best Practices

- Each method should perform a single, well-defined task
- Method's name should describe that task in a clear and unambiguous way
  - Good examples: `CalculatePrice`, `ReadName`
  - Bad examples: `f`, `g1`, `Process`
  - In C# methods should start with capital letter
- Avoid methods longer than one screen
  - Split them to several shorter methods

Summary

- Break large programs into simple methods that solve small sub-problems
- Methods consist of declaration and body
- Methods are invoked by their name
- Methods can accept parameters
  - Parameters take actual values when calling a method
- Methods can return a value or nothing