# Using Methods in C#

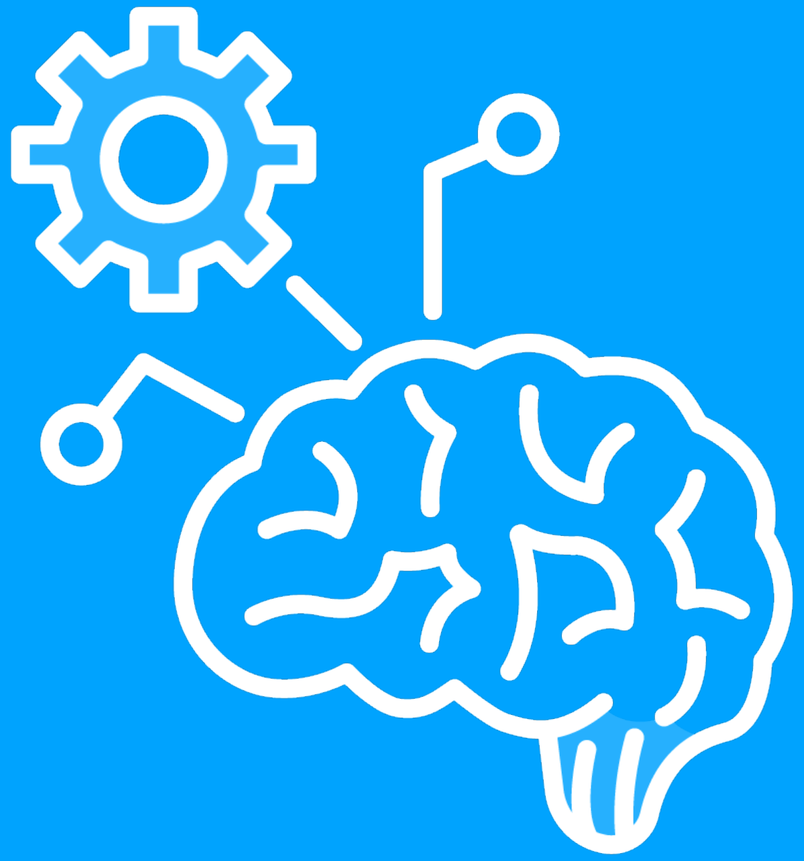
**Gill Cleeren**
CTO Xebia Microsoft Services Belgium

@gillcleeren

# Overview

- Understanding methods
- Adding a helper file
- Finding the correct method
- Understanding variable scope
- More options with methods
- Introducing the Main method

# Understanding Methods



# Our current code...

Is one large block

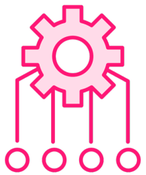
Some code can be reused multiple times though

Using methods will help

Similar to functions or subroutines

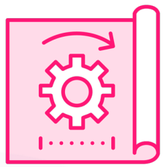
# Methods in C#

Code block



Receives parameters and (optionally) returns value

Readable code and code reuse



Declared within a class or struct

# C# Method Syntax

```
<access modifier> <return type> Method_Name (Parameters)
{
  //method statements
}
```

```
public int AddTwoNumbers()
{

}
```

# Looking at a First Method

```
public int AddTwoNumbers(int a, int b)
{

}
```

# Adding Method Parameters

```
public int AddTwoNumbers(int a, int b)
{
    return a + b;
}
```

# Returning a Value

## Return type must be specified

```
public int AddTwoNumbers(int a, int b)
{
    if (a > b)
    {
        return a + b;
    }
    //no value returned if we get here → compile time error
}
```

## Returning a Value

Value must be returned for all possible execution paths

```
public void DisplaySum(int a, int b)
{
    int sum = a + b;
    Console.WriteLine("The sum is " + sum);
}
```

# A Method without Return Value

```
...
DisplaySum(3, 52);
...
```

# Invoking a Method

We can pass arguments: values for the parameter(s)

# Flow of Execution

```
Console.WriteLine(“Before DisplaySum");

DisplaySum(3, 5);

Console.WriteLine("After DisplaySum");
```

```
public void DisplaySum(int a, int b)
{
    int sum = a + b;
    Console.WriteLine("The sum is " + sum);
}
```

```
DisplaySum(3, 52);
int result = AddTwoNumbers(55, 44);
```

## Capturing a Return Value

Only possible if method isn’t returning void

```
int p1 = 3;
int p2 = 52;

DisplaySum(p1, p2);
int result = AddTwoNumbers(55, 44);
```

## Capturing a Return Value

Only possible if method isn’t returning void

# Demo

Creating a method

Adding parameters

Returning a value

Invoking the method

# Adding a Helper File

# Demo

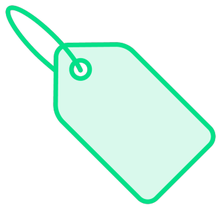
Adding a helper file

Moving the method

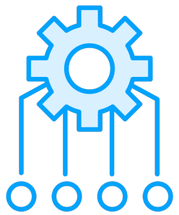
Invoking the method again

# Finding the Correct Method

# Calling the Correct Method



Method name



Parameter types and arguments


Number of parameters

# Method Overloading

## Matching the Parameters

**Program.cs**

```
DisplaySum(3, 52);
```

**Utilities.cs**

```
public static void DisplaySum
    (int a, int b)
{ ... }

public static void DisplaySum
    (int a, int b, int c)
{ ... }
```

# Demo

## Using method overloading

# Understanding Variable Scope

# Understanding Local Scope

```
public static double SomeMethod()
{
    double value = 0.04;

    ...

    return value;
}
```

# Understanding Local Scope

```
public static double SomeMethod()
{
    double value = 0.04;
    ...
    return value;
}

public static double AnotherMethod()
{
     ...

    return value;
}
```

# Class Scope

```
public static class Utilities
{
    double value = 0.04;

    public static double SomeMethod()
    {
        return value * 3;
    }

    public static double AnotherMethod()
    {
        return value / 2;
    }
}
```

# Demo

## Using variable scope

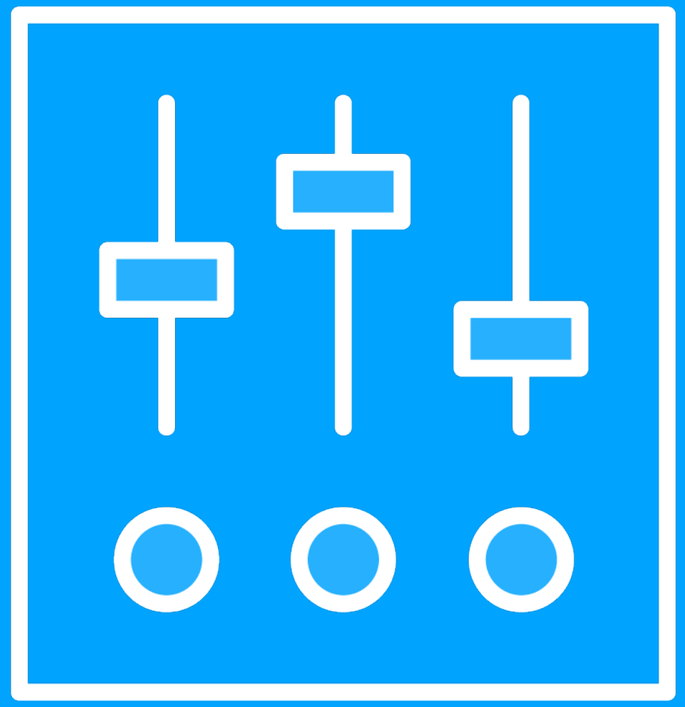
# More Options with Methods

# Doing More with Methods

Optional parameters

Named arguments

Expression-bodied syntax



# Optional Parameters

Specify default value for one or more parameters

Caller can omit the optional ones

# Working with Optional Parameters

**Method with optional parameters**

```
public int AddNumbers
    (int a,  int b, int c = 100)
{
    int sum = a + b + c;
    return sum;
}
```

**Calling the method**

```
//no third parameter
AddNumbers(10, 20);
AddNumbers(10, 20, 30);
```


# Named arguments

Not required to follow order of parameters

One or more parameters can have a name defined when invoking the method

# Working with Named Arguments

## Method with parameters

```
public static int AddNumbers
    (int a,  int b)
{
    ...
}
```

## Using named arguments

```
AddNumbers(b: 10, a: 20);
```

# Demo

Using optional parameters

Working with named arguments

# Demo

Using expression-bodied syntax

# Introducing the Main Method

# Comparing Our 2 Files

## Understanding Top-level Statements

**Program.cs (current)**

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

**Utilities.cs**

```
using System;

namespace ConsoleApp1
{
  internal class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Hello World!");
    }
  }
}
```

# Different Ways to Start the Application

**Program.cs (current)**

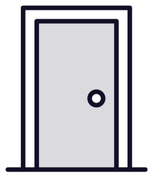
```
Console.WriteLine("Hello, World!");
```

**Program.cs (pre-C# 10)**

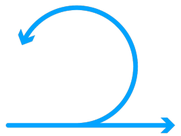
```
using System;

namespace ConsoleApp1
{
  internal class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Hello World!");
    }
  }
}
```

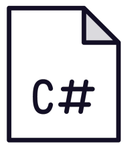
# The Main Method



Entry method which gets called upon start of the app



Gets created implicitly now




Top-level statements is default way

# Demo

## Exploring the Main method

# Going Forward

We will continue using Top-Level statements for Program.cs

Both options work though

# Summary

Methods are used to bring in reuse of code

Parameters are declared to accept incoming values

Main method is implicit now

Up Next:

# Working with strings