

<http://csharpfundamentals.telerik.com>



# Methods

---

## Subroutines in Computer Programming

Telerik Software Academy  
Learning & Development Team  
<http://academy.telerik.com>



## 1. Using Methods

- What is a Method? Why to Use Methods?
- Declaring and Creating Methods
- Calling Methods

## 2. Methods with Parameters

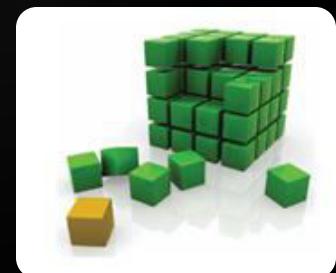
- Passing Parameters
- Returning Values

## 3. Best Practices



# 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, procedures, and subroutines



# 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





# Declaring and Creating Methods

# Declaring and Creating Methods

```
static void PrintLogo()  
{  
    Console.WriteLine("Telerik Corp.");  
    Console.WriteLine("www.telerik.com");  
}
```

Method  
name

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



# Declaring and Creating Methods (2)

```
static void PrintLogo()  
{  
    Console.WriteLine("Telerik Corp.");  
    Console.WriteLine("www.telerik.com");  
}
```

- ◆ 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

# Declaring and Creating Methods (3)

```
static void PrintLogo()  
{  
    Console.WriteLine("Telerik Corp.");  
    Console.WriteLine("www.telerik.com");  
}
```

Method  
body

- ◆ Each method has a body
  - ◆ It contains the programming code
  - ◆ Surrounded by { and }



# Declaring and Creating Methods (4)

```
using System;

class MethodExample
{
    static void PrintLogo()
    {
        Console.WriteLine("Telerik Corp.");
        Console.WriteLine("www.telerik.com");
    }

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



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

# 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:

Telerik Corp.  
[www.telerik.com](http://www.telerik.com)

# Calling Methods (2)

- ◆ A method can be called from:

- ◆ The Main() method

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



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

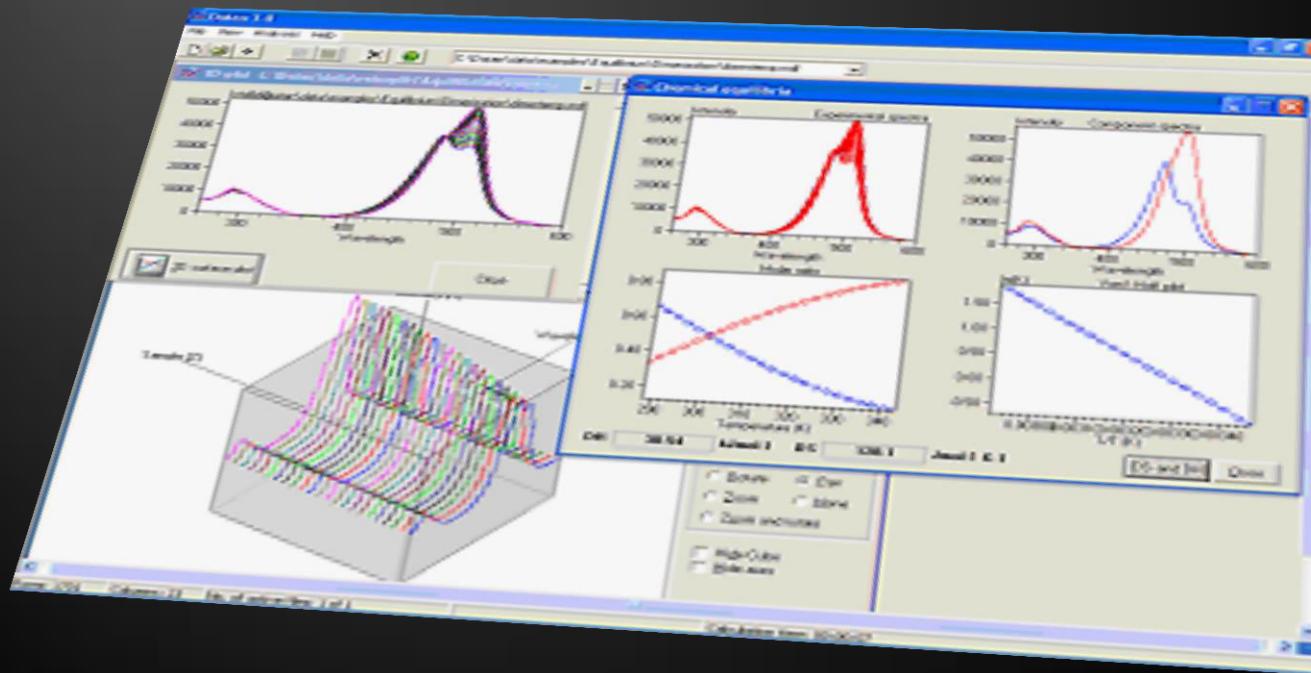


# Declaring and Calling Methods

Live Demo

# Methods with Parameters

## Passing Parameters and Returning Values



# 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

# 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.)

# Defining and Using Method Parameters (2)

- ◆ 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);
```

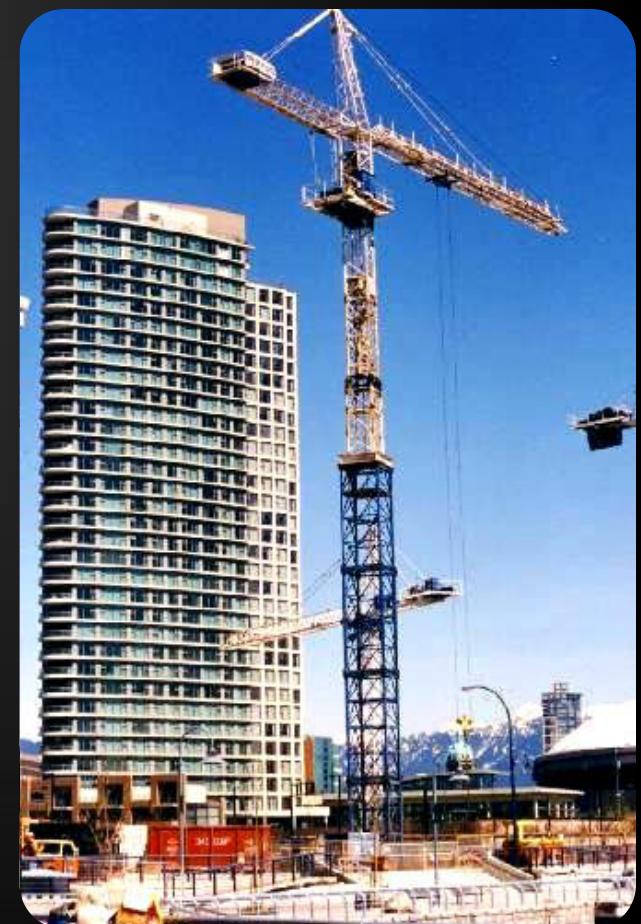


# Calling Methods with Parameters (2)

- ◆ 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

# Using Methods With Parameters

Examples



# Methods Parameters – Example

```
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);
}
```

# Method Parameters

Live Demo



# Months – Example

- ◆ Display the period between two months in a user-friendly way

```
using System;

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

*(the example continues)*

# Months – Example (2)

```
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);
}
```

# Months

Live Demo



# Printing Triangle – Example

- ◆ Creating a program for printing triangles as shown below:

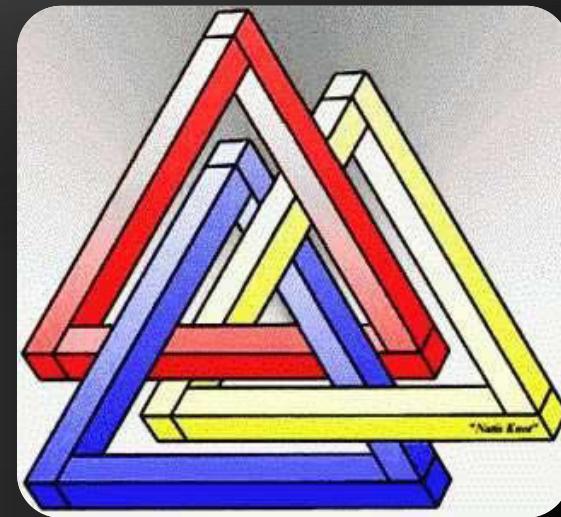
						1
					1	2
				1	2	3
			1	2	3	4
		1	2	3	4	5
n=5 →	1	2	3	4	5	
	1	2	3	4		
	1	2	3			
	1	2				
	1					

# Printing Triangle – Example

```
static void Main()
{
    int n = int.Parse(Console.ReadLine());

    for (int line = 1; line <= n; line++)
        PrintLine(1, line);
    for (int line = n-1; line >= 1; line--)
        PrintLine(1, line);
}

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



# Printing Triangle

Live Demo

# Optional Parameters

- ◆ C# 4.0 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.WriteLine("{0} ", i);
    }
}
```

- ◆ The above method can be called in several ways:

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



# Optional Parameters

Live Demo

# Returning Values From Methods



# 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());
```

# 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

# Returning Values From Methods

## Examples



# Returning Values From Methods

## Examples



# Temperature Conversion – Example

- ◆ Convert temperature from Fahrenheit to Celsius:

```
static double FahrenheitToCelsius(double degrees)
{
    double celsius = (degrees - 32) * 5 / 9;
    return celsius;
}

static void Main()
{
    Console.Write("Temperature in Fahrenheit: ");
    double t = Double.Parse(Console.ReadLine());
    t = FahrenheitToCelsius(t);
    Console.WriteLine("Temperature in Celsius: {0}", t);
}
```

# Temperature Conversion

Live Demo



# Positive Numbers – Example

- ◆ Check if all numbers in a sequence are positive:

```
static bool ArePositive(int[] sequence)
{
    foreach (int number in sequence)
    {
        if (number <= 0)
        {
            return false;
        }
    }
    return true;
}
```



# Positive Numbers

Live Demo



# Data Validation – Example

## ◆ Validating input data:

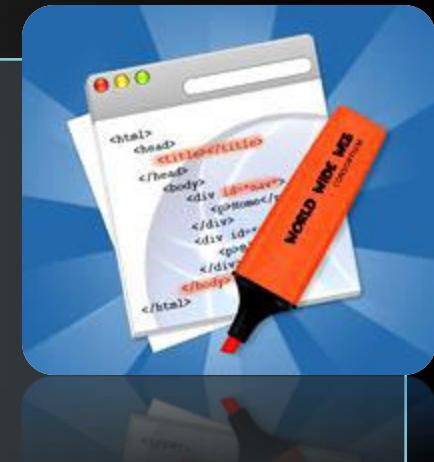
```
using System;

class ValidatingDemo
{
    static void Main()
    {
        Console.WriteLine("What time is it?");

        Console.Write("Hours: ");
        int hours = int.Parse(Console.ReadLine());

        Console.Write("Minutes: ");
        int minutes = int.Parse(Console.ReadLine());

        // (The example continues on the next slide)
```



# Data Validation – Example

```
bool isValidTime =  
    ValidateHours(hours) &&  
    ValidateMinutes(minutes);  
if (isValidTime)  
    Console.WriteLine("It is {0}:{1}",  
        hours, minutes);  
else  
    Console.WriteLine("Incorrect time!");  
}  
  
static bool ValidateMinutes(int minutes)  
{  
    bool result = (minutes>=0) && (minutes<=59);  
    return result;  
}  
  
static bool ValidateHours(int hours) { ... }  
}
```





# Data Validation

## Live Demo



# 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);
}
```

# 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());
}
```

# Methods – Best Practices

- ◆ Each method should perform a single, well-defined task
- ◆ Method's name should describe that task in a clear and non-ambiguous 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

- ◆ 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

# Questions?

1. Write a method that asks the user for his name and prints “Hello, <name>” (for example, “Hello, Peter!”). Write a program to test this method.
2. Write a method **GetMax()** with two parameters that returns the bigger of two integers. Write a program that reads 3 integers from the console and prints the biggest of them using the method **GetMax()**.
3. Write a method that returns the last digit of given integer as an English word. Examples: **512** → "two", **1024** → "four", **12309** → "nine".

4. Write a method that counts how many times given number appears in given array. Write a test program to check if the method is working correctly.
5. Write a method that checks if the element at given position in given array of integers is bigger than its two neighbors (when such exist).
6. Write a method that returns the index of the first element in array that is bigger than its neighbors, or -1, if there's no such element.
  - Use the method from the previous exercise.

7. Write a method that reverses the digits of given decimal number. Example: 256 → 652
8. Write a method that adds two positive integer numbers represented as arrays of digits (each array element `arr[i]` contains a digit; the last digit is kept in `arr[0]`). Each of the numbers that will be added could have up to 10 000 digits.
9. Write a method that return the maximal element in a portion of array of integers starting at given index. Using it write another method that sorts an array in ascending / descending order.

10. Write a program to calculate  $n!$  for each  $n$  in the range  $[1..100]$ . Hint: Implement first a method that multiplies a number represented as array of digits by given integer number.
11. Write a method that adds two polynomials.  
Represent them as arrays of their coefficients as in the example below:

$$x^2 + 5 = 1x^2 + 0x + 5 \rightarrow$$

5	0	1
---	---	---

12. Extend the program to support also subtraction and multiplication of polynomials.

13. Write a program that can solve these tasks:

- Reverses the digits of a number
- Calculates the average of a sequence of integers
- Solves a linear equation  $a * x + b = 0$

Create appropriate methods.

Provide a simple text-based menu for the user to choose which task to solve.

Validate the input data:

- The decimal number should be non-negative
- The sequence should not be empty
- $a$  should not be equal to 0

14. Write methods to calculate minimum, maximum, average, sum and product of given set of integer numbers. Use variable number of arguments.
15. \* Modify your last program and try to make it work for any number type, not just integer (e.g. decimal, float, byte, etc.). Use generic method (read in Internet about generic methods in C#).

# Free Trainings @ Telerik Academy

- ◆ “C# Programming @ Telerik Academy

- ◆ [csharpfundamentals.telerik.com](http://csharpfundamentals.telerik.com)



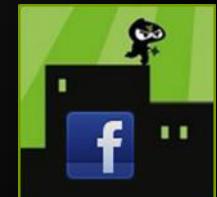
- ◆ Telerik Software Academy

- ◆ [academy.telerik.com](http://academy.telerik.com)



- ◆ Telerik Academy @ Facebook

- ◆ [facebook.com/TelerikAcademy](https://facebook.com/TelerikAcademy)



- ◆ Telerik Software Academy Forums

- ◆ [forums.academy.telerik.com](http://forums.academy.telerik.com)

