Methods

Defining and Using Methods, Overloads

greenwich.edu.vn

UNIVERSITY of GREENWICH

Alliance with FPT Education

Table of Contents

WHAT IS A METHOD

Void Method

Simple Methods

- Named block of code, that can be invoked later
- Sample method definition:

```
static void PrintHelloWorld()
{
  Console.WriteLine("Hello World");
}
```

Method named `PrintHelloWorld`

Method body always surrounded by { }

- Invoking (calling) the method several times:

```
PrintHeader();
PrintHeader();
```

Why Use Methods?

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

Void Type Method

- Executes the code between the brackets
- Does not return result

```
static void PrintHello()
{
    Console.WriteLine("Hello");
}
```

Prints "Hello" on the console

```
static void Main()
{
    Console.WriteLine("Hello");
}
```

Main() is also a method

DECLARING AND INVOKING METHODS

Declaring Methods

Type

Method Name

Parameters

```
static void PrintText(string text)
{
  Console.WriteLine(text);
}
```

Method Body

- Methods are declared inside a class
- Variables inside a method are local

Invoking a Method

- Methods are first declared, then invoked (many times)

```
static void PrintHeader()
{
  Console.WriteLine("----------");
}
```

Method Declaration

- Methods can be invoked (called) by their name + ():

```
static void Main()
{
  PrintHeader();
}
```

Method Invocation

Invoking a Method (2)

- A method can be invoked from:
  - The main method – Main()

```
static void Main()
{
  PrintHeader();
}
```

  - Its own body – recursion

```
static void Crash()
{ Crash(); }
```

  - Some **other method**

```
static void PrintHeader()
{
  PrintHeaderTop();
  PrintHeaderBottom();
}
```

UNIVERSITY of GREENWICH

Alliance with FPT Education

METHODS WITH PARAMETERS

Method Parameters

- Method parameters can be of any data type

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

Multiple parameters separated by comma

- Call the method with certain values (arguments)

```
static void Main()
{
  PrintNumbers(5, 10);
}
```

Passing arguments at invocation

Method Parameters (2)

- You can pass zero or several parameters
- You can pass parameters of different types
- Each parameter has name and type

Multiple parameters of different types

Parameter type

Parameter name

```
static void PrintStudent(string name, int age, double grade)
{
    Console.WriteLine("Student: {0}; Age: {1}, Grade: {2}",
        name, age, grade);
}
```

Problem: Sign of Integer Number

- Create a method that prints the sign of an integer number n:

`2`	→	`The number 2 is positive.`
`-5`	→	`The number -5 is negative.`
`0`	→	`The number 0 is zero.`

Solution: Sign of Integer Number

```
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 Main()
{ PrintSign(int.Parse(Console.ReadLine())); }
```

Problem: Grades

- Write a method that receives a grade between 2.00 and 6.00
  and prints the corresponding grade in words
  - 2.00 - 2.99 - "Fail"
  - 3.00 - 3.49 - "Poor"
  - 3.50 - 4.49 - "Good"
  - 4.50 - 5.49 - "Very good"
  - 5.50 - 6.00 - "Excellent"

`3.33`	→	`Poor`
`4.50`	→	`Very good`
`2.99`	→	`Fail`

Optional Parameters

- Parameters can accept **default values**:

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

Default values

- The above method can be called in several ways:

```
PrintNumbers(5, 10);
```

```
PrintNumbers(end: 40, start: 35);
```

```
PrintNumbers(15);
```

```
PrintNumbers();
```

Can be skipped at method invocation

Problem: Printing Triangle

- Create a method for printing triangles as shown below:

```
3
```

```
1
1 2
1 2 3
1 2
1
```

```
4
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3
1 2
1
```

Solution: Printing Triangle

- Create a method that **prints a single line**, consisting of numbers from a **given start** to a **given end**:

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

Solution: Printing Triangle (2)

- Create a method that prints the **first half (1..n)** and then the **second half (n-1…1)** of the triangle:

```
static void PrintTriangle(int n)
{
  for (int line = 1; line <= n; line++)
    PrintLine(1, line);

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

Method with parameter n

Lines 1…n

Lines n-1…1

UNIVERSITY of GREENWICH

Alliance with FPT Education

RETURNING VALUES FROM METHODS

The Return Statement

- The **return** keyword immediately stops the method's execution
- Returns the specified value

```
static string ReadFullName()
{
  string firstName = Console.ReadLine();
  string lastName = Console.ReadLine();
  return firstName + " " + lastName;
}
```

Returns a string

- Void methods can be **terminated** by just using **return**

Using the Return Values

- Return value can be:
  - **Assigned** to a variable:

```
int max = GetMax(5, 10);
```

  - **Used** in expression:

```
decimal total = GetPrice() * quantity * 1.20m;
```

  - **Passed** to another method:

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

Problem: Calculate Rectangle Area

- Create a method which returns rectangle area with given width and height

3 4	→	12	6 8	→	48
5 10	→	50	7 8	→	56

Solution: Calculate Rectangle Area

```
static void Main()
{
  double width = double.Parse(Console.ReadLine());
  double height = double.Parse(Console.ReadLine());
  double area = CalcRectangleArea(width, height);
  Console.WriteLine(area);
}
```

```
static double CalcRectangleArea(double width,double height)
{
  return width * height;
}
```

Problem: Repeat String

- Write a method that receives a string and a repeat count n. The method should return a new string.

```
abc
3
```

→

```
abcabcabc
```

```
String
2
```

→

```
StringString
```

Problem: Math Power

- Create a method that calculates and returns the value of a **number raised to a given power**

2^8 → 256

3^4 → 81

```
static double MathPower(double number, int power)
{
  double result = 1;
  for (int i = 0; i < power; i++)
    result *= number;
  return result;
}
```

UNIVERSITY of GREENWICH

Alliance with FPT Education

OVERLOADING METHODS

Method Signature

- The combination of method's **name** and **parameters** is called **signature**

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

Method's signature

- Signature **differentiates** between methods with same names
- When methods with the **same name** have **different signature**, this is called method "**overloading**"

Overloading Methods

- Using same name for multiple methods with different signatures (method name and 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);
}
```

Different method signatures

Signature and Return Type

- Method's return type is not part of its signature

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

Compile-time error!

- How would the compiler know which method to call?

Problem: Greater of Two Values

Create a method **`GetMax()`** that **returns the greater** of two values (the values can be of type **`int`**, **`char`** or **`string`**)

```
int
2
16
```

→

```
16
```

```
char
a
z
```

→

```
z
```

```
string
aaa
bbb
```

→

```
bbb
```

UNIVERSITY of GREENWICH

Alliance with FPT Education

PROGRAM EXECUTION FLOW

Program Execution

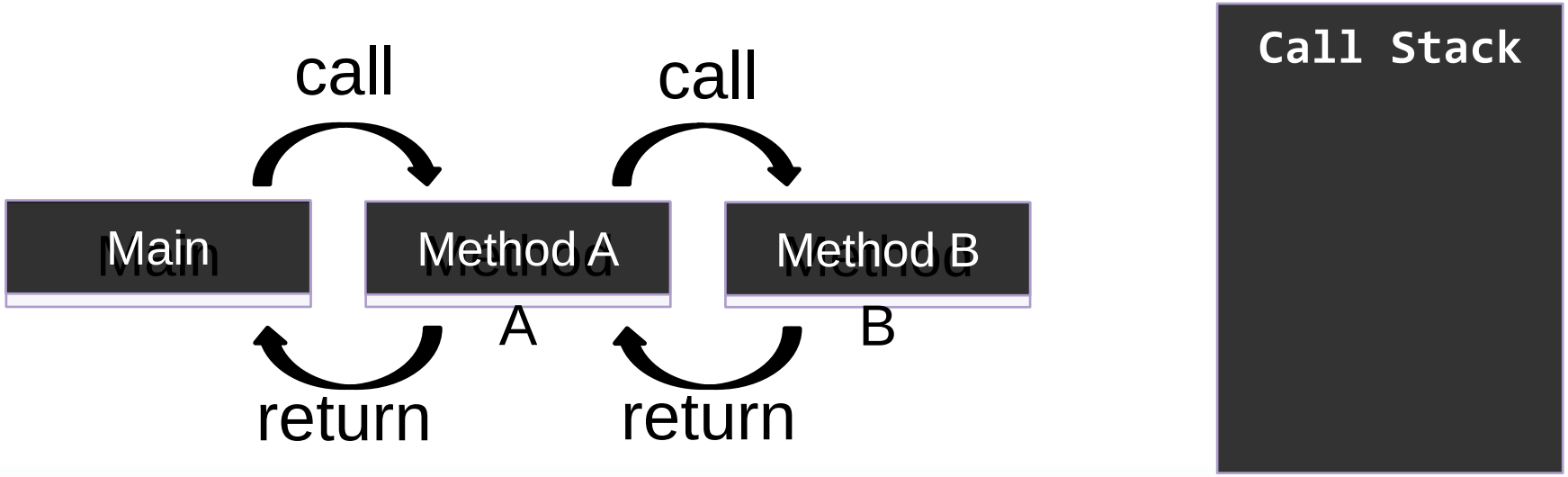
- The program continues, after a method execution completes:

```
static void Main()
{
    Console.WriteLine("before method executes");
    PrintLogo();
    Console.WriteLine("after method executes");
}
```

```
static void PrintLogo()
{
   Console.WriteLine("Company Logo");
   Console.WriteLine("http://www.companywebsite.com");
}
```

Program Execution – Call Stack

- "The stack" stores information about the active subroutines (methods) of a computer program
- Keeps track of the point to which each active subroutine should return control when it finishes executing



Problem: Multiply Evens by Odds

- Create a program that **multiplies the sum** of **all even digits** of a number **by the sum of all odd digits** of the same number:
  - Create a method called `GetMultipleOfEvensAndOdds()`
  - Create a method `GetSumOfEvenDigits()`
  - Create `GetSumOfOddDigits()`
  - You may need to use `Math.Abs()` for negative numbers

```
-12345
```

→

```
Evens: 2 4
Odds: 1 3 5
```

→

```
Even sum: 6
Odd sum: 9
```

→

```
54
```

UNIVERSITY of GREENWICH

Alliance with FPT Education

NAMING AND BEST PRACTICES

Naming Methods

- Methods naming guidelines
  - Use meaningful method names
  - Method names should answer the question:
    - What does this method do?

```
FindStudent, LoadReport, Sine
```

  - If you cannot find a good name for a method, think about whether it has a clear intent

```
Method1, DoSomething, HandleStuff, SampleMethod, DirtyHack
```

Naming Method Parameters

- Method parameters names
  - Preferred form: [**Noun**] or [**Adjective**] + [**Noun**]
  - Should be in `camelCase`
  - Should be **meaningful**

```
firstName, report, speedKmH,
usersList, fontSizeInPixels, font
```

  - **Unit of measure should be obvious**

```
p, p1, p2, populate, LastName, last_name, convertImage
```

Methods – Best Practices

- Each method should perform a single, well-defined task
  - A Method's name should describe that task in a clear and non-ambiguous way
- Avoid methods longer than one screen
  - Split them to several shorter methods

```
private static void PrintReceipt()
{
    PrintHeader();
    PrintBody();
    PrintFooter();
}
```

Self documenting and easy to test

Code Structure and Code Formatting

- Make sure to use correct indentation

```
static void Main()
{
    // some code…
    // some more code…
}
```

```
static void Main()
    {
        // some code…
// some more code…
}
```

- Leave a blank line between methods, after loops and after if statements
- Always use curly brackets for loops and if statements bodies
- Avoid long lines and complex expressions

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
- Methods can return a value or nothing (void)