EIU
TRƯỜNG ĐẠI HỌC
QUỐC TẾ
MIỀN ĐÔNG
EASTERN
INTERNATIONAL
UNIVERSITY

.NET Programming

Chapter 1: C# Programming languages

EIU

C# fundamentals

1.INTRODUCTION TO C#

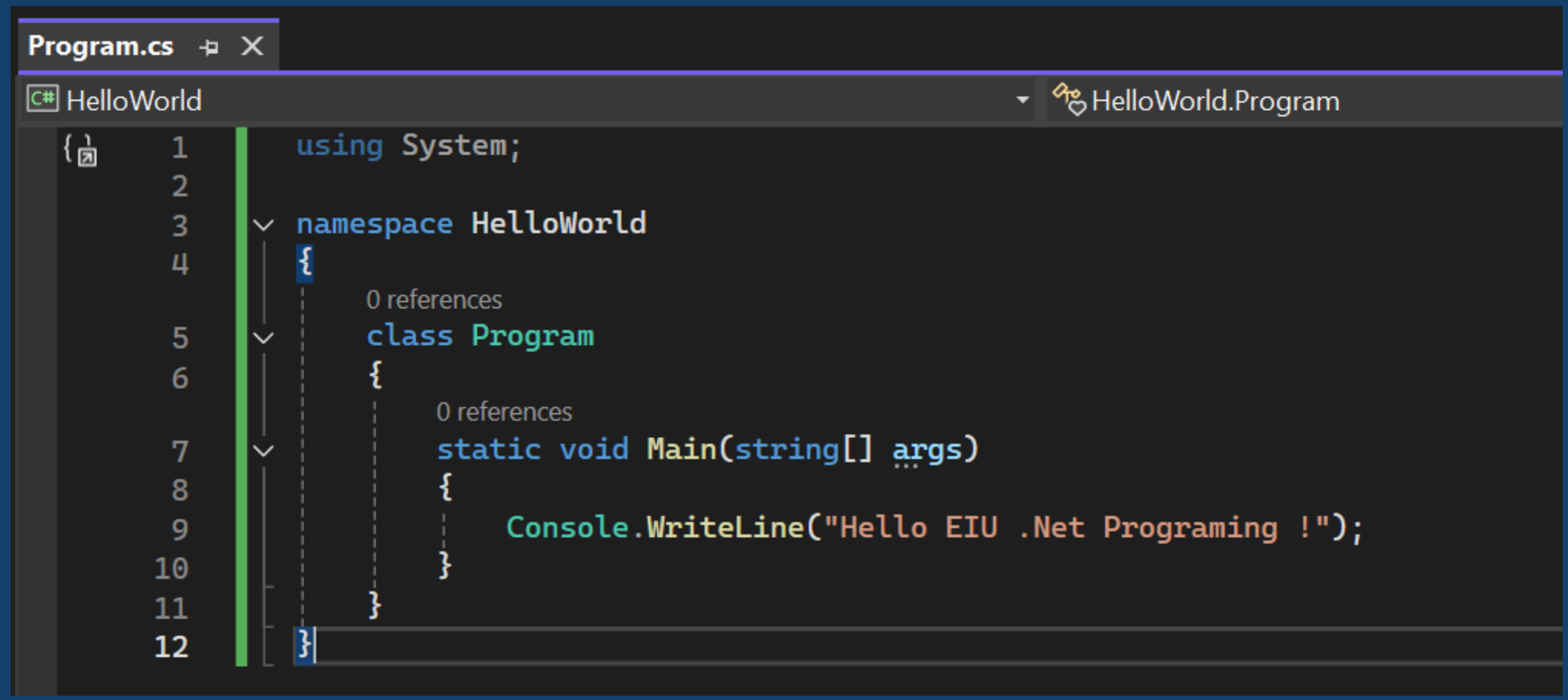
What is C# ?

- C# (C-Sharp) is a programming language developed by Microsoft that runs on the .NET Framework.
- C# has roots from the C family, and the language is close to other popular languages like C++ and Java.
- The first version was released in year 2002. The latest version, **C# 12**, was released in November 2023.
- C# is used to develop web applications, desktop applications, mobile apps, games and much more.
- A modern, object-oriented programming language developed by Microsoft.
- Part of the .NET ecosystem.
- Combines the best of C++ and Java.

2. INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

3. SYNTAX

The Basic C# file when create a console project that called “Program.cs”



3. SYNTAX

Example explained

- Line 1: using System means that we can use classes from the System namespace.
- Line 2: A blank line. C# ignores white space. However, multiple lines makes the code more readable.
- Line 3: namespace is used to organize your code, and it is a container for classes and other namespaces.
- Line 4: The curly braces {} marks the beginning and the end of a block of code.
- Line 5: class is a container for data and methods, which brings functionality to your program. Every line of code that runs in C# must be inside a class. In our example, we named the class Program.
- Line 7: Another thing that always appear in a C# program is the Main method. Any code inside its curly brackets {} will be executed. You don't have to understand the keywords before and after Main. You will get to know them bit by bit while reading this tutorial.
- Line 9: Console is a class of the System namespace, which has a WriteLine() method that is used to output/print text. In our example, it will output "Hello World!".
If you omit the using System line, you would have to write System.Console.WriteLine() to print/output text.

4. COMMENT

How can we comment code in the C# programming language ?

```
/* Comment
 * for
 multiple
line */

// Comment for single line
```

```
17
18
19          /* This is the comment for multiple line
20           *
21           * Console.WriteLine("Hello EIU .Net Programing !"); */
22
23
24          // This is the comment for single line
25          //Console.WriteLine("Hello EIU .Net Programing !");
26              }
27          }
28      }
```

5. VARIABLES

int - stores integers (whole numbers), without decimals, such as 123 or -123
double - stores floating point numbers, with decimals, such as 19.99 or -19.99
char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
string - stores text, such as "Hello World". String values are surrounded by double quotes
bool - stores values with two states: true or false

Syntax

```
type variableName = value;
```

```
string subjectName = “EIU .Net Programming”;
int myNum = 99;
```

5. VARIABLES

Constants

- Cannot declare a constant variable without assigning the value
- Eead-only
- Cannot overwrite existing values

```
21
22          const int myNum = 2024;
23          myNum = 2023; // error
24
```

5. VARIABLES

How to use variables

```
string name = "Taylor Swift";
Console.WriteLine("Hello " + name);
```

```
string firstName = ".NET ";
string lastName = "Programming";
string fullName = firstName + lastName;
Console.WriteLine(fullName);
```

```
int x = 99;
int y = 1;
Console.WriteLine(x + y); // Print the value of x + y
```

5. VARIABLES

Identifiers and clean variables

```
// Good
bool isDeleted = false;

// OK, but not so easy to understand what d actually is
bool d = false;
```

6. DATA TYPES

Data Type	Size	Description
`int`	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
`long`	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
`float`	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
`double`	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
`bool`	1 bit	Stores true or false values
`char`	2 bytes	Stores a single character/letter, surrounded by single quotes
`string`	2 bytes per character	Stores a sequence of characters, surrounded by double quotes

```
int myNum = 5;                  // Integer (whole number)
double myDoubleNum = 5.99D;     // Floating point number
char myLetter = 'D';            // Character
bool myBool = true;             // Boolean
string myText = "Hello";        // String
```

7. TYPE CASTING

Type casting is when you assign a value of one data type to another type.

Implicit Casting (automatically) - converting a smaller type to a larger type size
char -> int -> long -> float -> double

Explicit Casting (manually) - converting a larger type to a smaller size type
double -> float -> long -> int -> char

```
//Implicit Casting
int myInt = 9;
double myDouble = myInt;       // Automatic casting: int to double

Console.WriteLine(myInt);      // Outputs 9
Console.WriteLine(myDouble);   // Outputs 9
```

```
//Explicit Casting
double myDouble = 9.78;
int myInt = (int)myDouble;    // Manual casting: double to int

Console.WriteLine(myDouble);   // Outputs 9.78
Console.WriteLine(myInt);      // Outputs 9
```

8. USER INPUT

```
// Type your username and press enter
Console.WriteLine("Enter username:");

// Create a string variable and get user input from the keyboard and store it in the variable
string userName = Console.ReadLine();

// Print the value of the variable (userName), which will display the input value
Console.WriteLine("Username is: " + userName);
```

8. USER INPUT

Handle error for user Input

```
Console.WriteLine("Enter your age:");
int age = Console.ReadLine();
Console.WriteLine("Your age is: " + age);
```

	Code	Description
⊗	CS0029	Cannot implicitly convert type 'string' to 'int'

```
Console.WriteLine("Enter your age:");
int age = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("Your age is: " + age);
```

Microsoft Visual Studio Debu

```
Enter your age:
18
Your age is: 18
```

9. OPERATORS

Operators are used to perform operations on variables and values.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	x++
--	Decrement	Decreases the value of a variable by 1	x--

9. OPERATORS

Assignment Operators

- Assignment operators are used to assign values to variables.

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

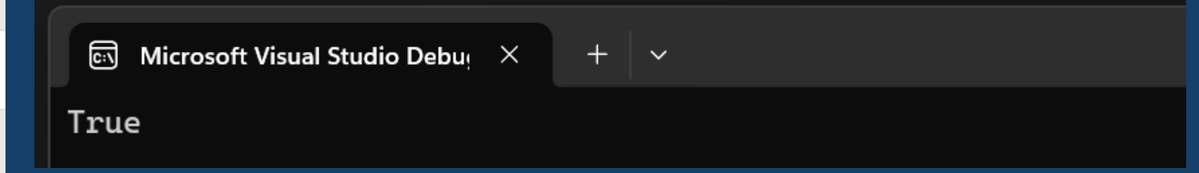
9. OPERATORS

Comparison Operators

- Comparison operators are used to compare two values (or variables). This is important in programming, because it helps us to find answers and make decisions.

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

```
int x = 5;
int y = 3;
Console.WriteLine(x > y); // returns True because 5 is greater than 3
```



9. OPERATORS

Logical Operators

- Logical operators are used to determine the logic between variables or values

Operator	Name	Description	Example
&&	Logical and	Returns True if both statements are true	x < 5 && x < 10
||	Logical or	Returns True if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns False if the result is true	!(x < 5 && x < 10)

10. STRING

A string variable contains a collection of characters surrounded by double quotes

A string in C# is actually an object, which contain properties and methods that can perform certain operations on strings.

Length:

```
string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
Console.WriteLine("The length of the txt string is: " + txt.Length);
```

```
Microsoft Visual Studio Debu
The length of the txt string is: 26
```

Uppercase or lowercase:

```
string txt = "Hello World";
Console.WriteLine(txt.ToUpper());   // Outputs "HELLO WORLD"
Console.WriteLine(txt.ToLower());   // Outputs "hello world"
```

```
Microsoft Visual Studio Debu
HELLO WORLD
hello world
```

10. STRING

String Concatenation

The “+” operator can be used between strings to combine them.
This is called concatenation:

```
string firstName = ".NET ";
string lastName = "Programming";
string name = firstName + lastName;
Console.WriteLine(name);
```

```
Microsoft Visual Studio Debu
.NET Programming
```

String Interpolation

```
string firstName = ".NET ";
string lastName = "Programming";
string name = $"My course is: {firstName} {lastName}";
Console.WriteLine(name);
```

```
Microsoft Visual Studio Debu
My course is: .NET  Programming
```

Special Characters

The strings must be written within quotes, C# will misunderstand this string, and generate an error:

```
string txt = "I'm learning the ".NET Programming" course.";
```

Escape character	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

```
string txt = "I'm learning the \".NET Programming\" course.";
Console.WriteLine(txt);
```

```
Microsoft Visual Studio Debu
I'm learning the ".NET Programming" course.
```

C# has a bool data type, which can take the values true or false

EIU

10. BOOLEANS

C# has a bool data type, which can take the values true or false.

```
bool isDotNetFun = true;
bool isFishTasty = false;
Console.WriteLine(isDotNetFun);   // Outputs True
Console.WriteLine(isFishTasty);   // Outputs False
```

```
Microsoft Visual Studio Debu
True
False
```

C# has a bool data type, which can take the values true or false

10. BOOLEANS

Boolean Expression

- A Boolean expression returns a boolean value: True or False, by comparing values/variables.
- Useful to build logic, and find answers.

```
int x = 10;
int y = 9;
Console.WriteLine(x > y); // returns True, because 10 is higher than 9
```

```
Microsoft Visual Studio Debu
True
```

10. BOOLEANS

Boolean Expression

More Example:

```
int x = 10;
Console.WriteLine(x == 10); // returns True, because the value of x is equal to 10
Console.WriteLine(10 == 15); // returns False, because 10 is not equal to 15
int myAge = 25;
int votingAge = 18;
if (myAge >= votingAge)
{
    Console.WriteLine("Old enough to vote!");
}
else
{
    Console.WriteLine("Not old enough to vote.");
}
```

11. If … Else

Conditions

C# supports the usual logical conditions from mathematics:

- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b
- Equal to a == b
- Not Equal to: a != b

C# has the following conditional statements:

- Use if to specify a block of code to be executed, if a specified condition is true
- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false
- Use switch to specify many alternative blocks of code to be executed

11. If ... Else

The If Statements

```
if (condition)
{
    // block of code to be executed if the condition is True
}

if (20 > 18)
{
    Console.WriteLine("20 is greater than 18");
}

int x = 20;
int y = 18;
if (x > y)
{
    Console.WriteLine("x is greater than y");
}
```

11. If ... Else

The Else Statement

```
if (condition)
{
    // block of code to be executed if the condition is True
}
else
{
    // block of code to be executed if the condition is False
}

int time = 20;
if (time < 18)
{
    Console.WriteLine("Good day.");
}
else
{
    Console.WriteLine("Good evening.");
}
// Outputs "Good evening."
```

11. If ... Else

The Else If Statement

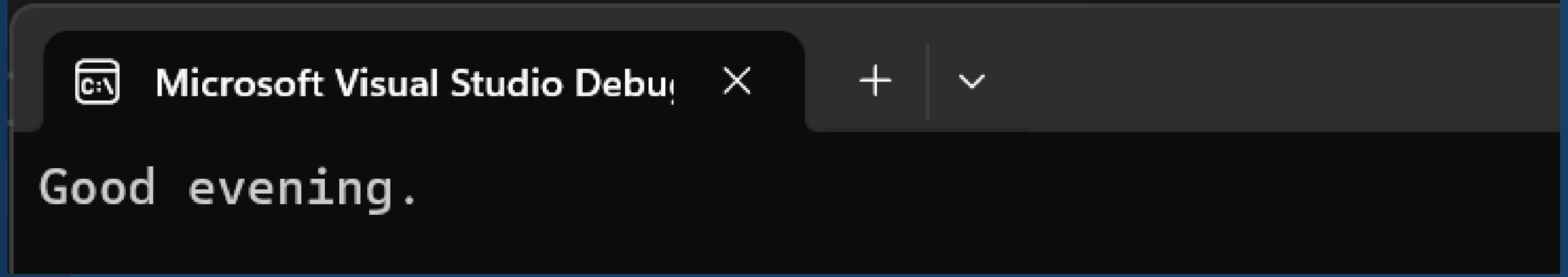
```
if (condition1)
{
    // block of code to be executed if condition1 is True
}
else if (condition1)
{
    // block of code to be executed if the condition1 is false and condition2 is True
}
else
{
    // block of code to be executed if the condition1 is false and condition2 is False
}

int time = 22;
if (time < 10)
{
    Console.WriteLine("Good morning.");
}
else if (time < 20)
{
    Console.WriteLine("Good day.");
}
else
{
    Console.WriteLine("Good evening.");
}
// Outputs "Good evening."
```

11. If ... Else

Short Hand If...Else

variable = (condition) ? expressionTrue : expressionFalse;

```
int time = 20;
string result = (time < 18) ? "Good day." : "Good evening.";
Console.WriteLine(result);
```



12. Switch

Use the switch statement to select one of many code blocks to be executed.

```
switch(expression)
{
  case 1:
    // code block
    break;
  case 2:
    // code block
    break;
  default:
    // code block
    break;
}
```

12. Switch

Example:

```
18
19          int day = 4;
20          switch (day)
21          {
22              case 1:
23                  Console.WriteLine("Monday");
24                  break;
25              case 2:
26                  Console.WriteLine("Tuesday");
27                  break;
28              case 3:
29                  Console.WriteLine("Wednesday");
30                  break;
31              case 4:
32                  Console.WriteLine("Thursday");
33                  break;
34              case 5:
35                  Console.WriteLine("Friday");
36                  break;
37              case 6:
38                  Console.WriteLine("Saturday");
39                  break;
40              case 7:
41                  Console.WriteLine("Sunday");
42                  break;
43          }
44          // Outputs "Thursday" (day 4)
45
46      }
47  }
48
```

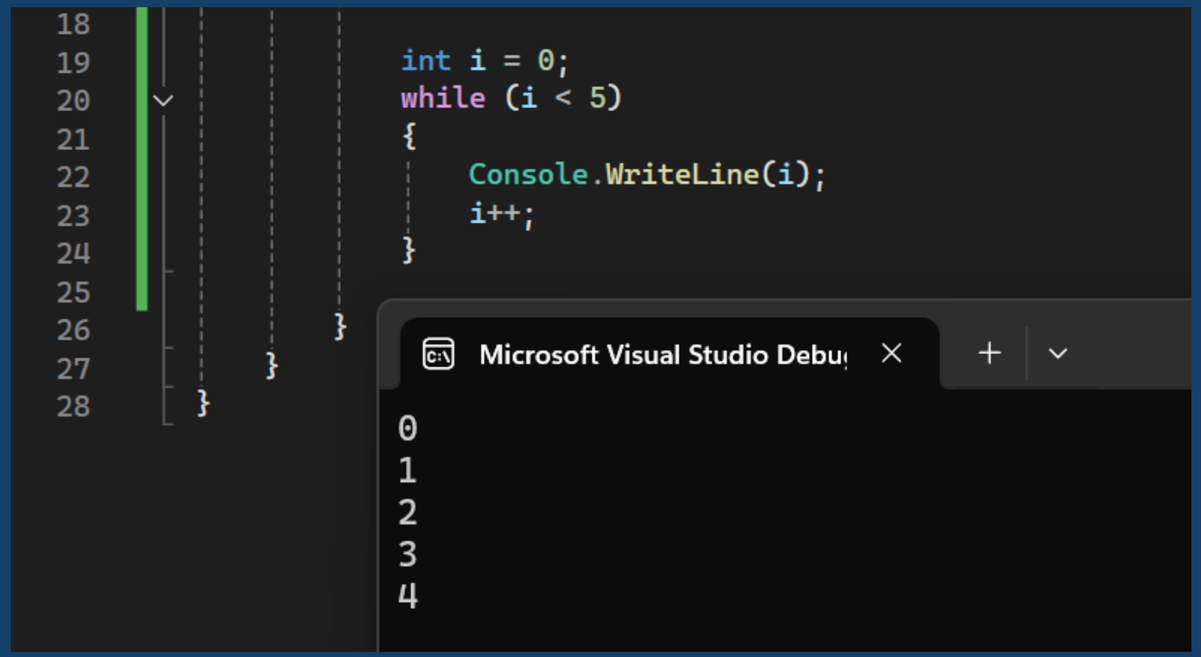
```
Microsoft Visual Studio Debu
Thursday
```

13. LOOP

While Loop

Syntax:

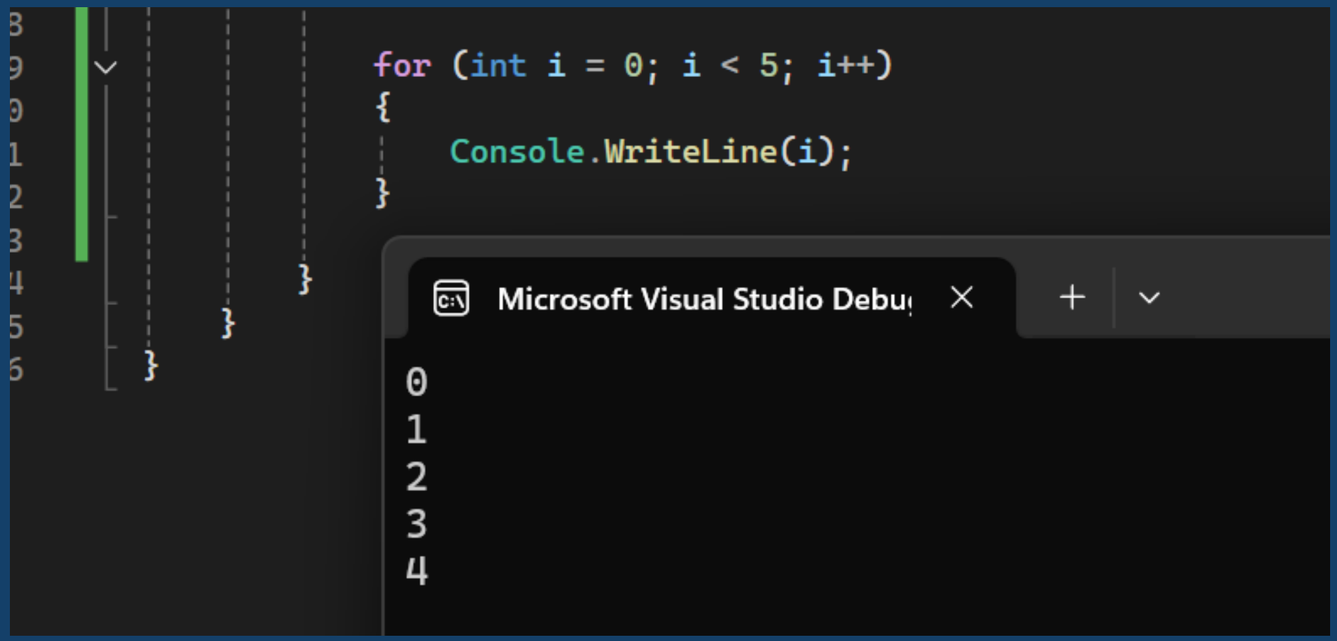
```
while (condition)
{
  // code block to be executed
}
```



13. LOOP

For Loop
Syntax:

```
for (statement 1; statement 2; statement 3)
{
  // code block to be executed
}
```

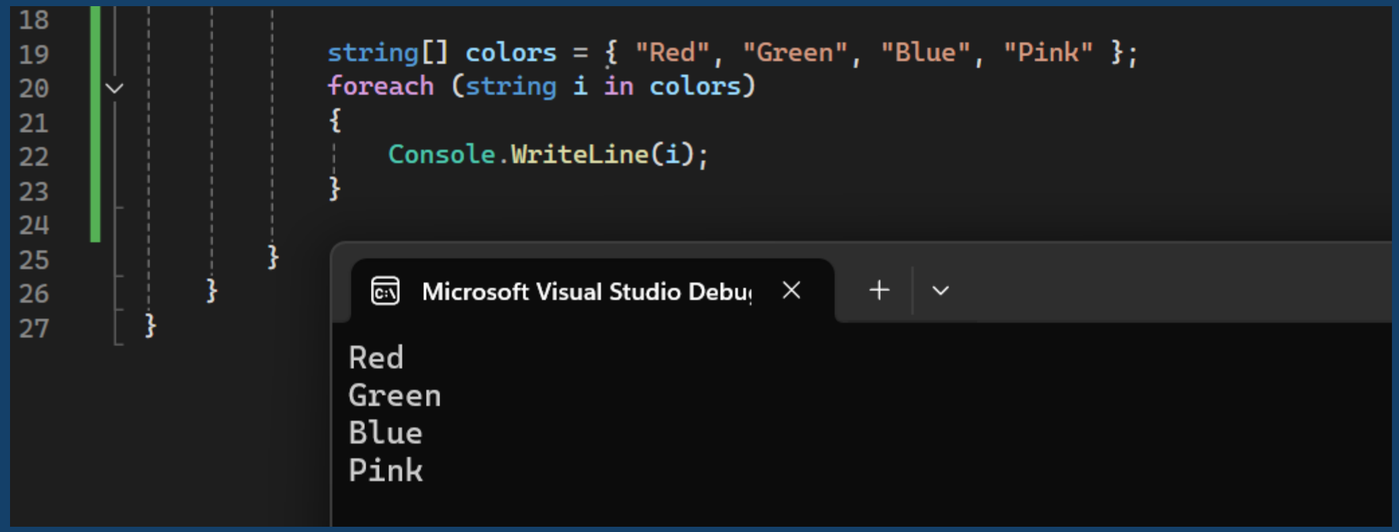


13. LOOP

Foreach Loop

Syntax:

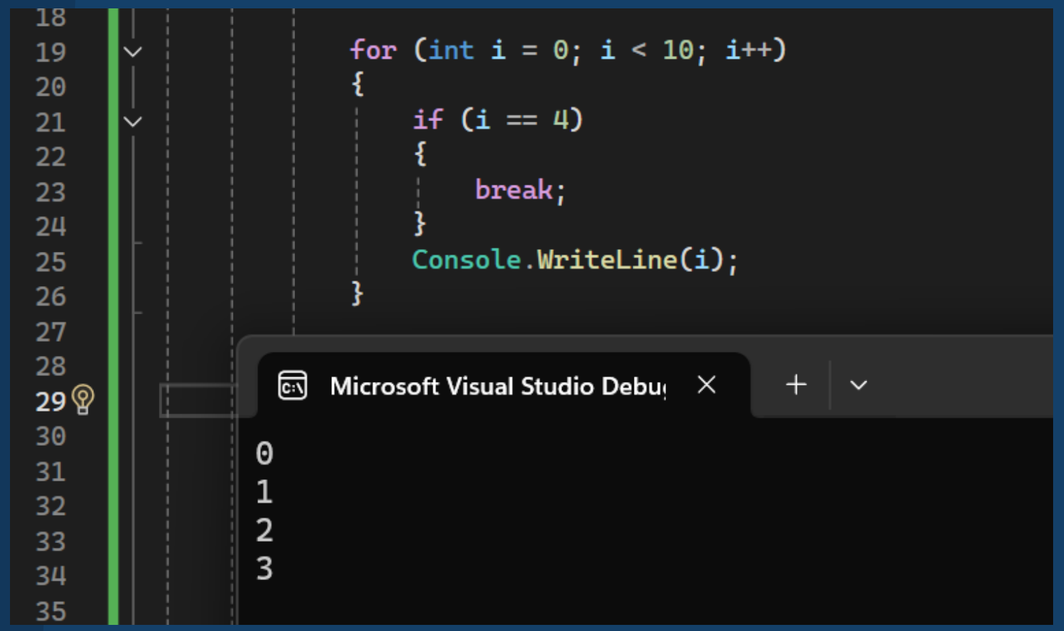
```
foreach (type variableName in arrayName)
{
  // code block to be executed
}
```

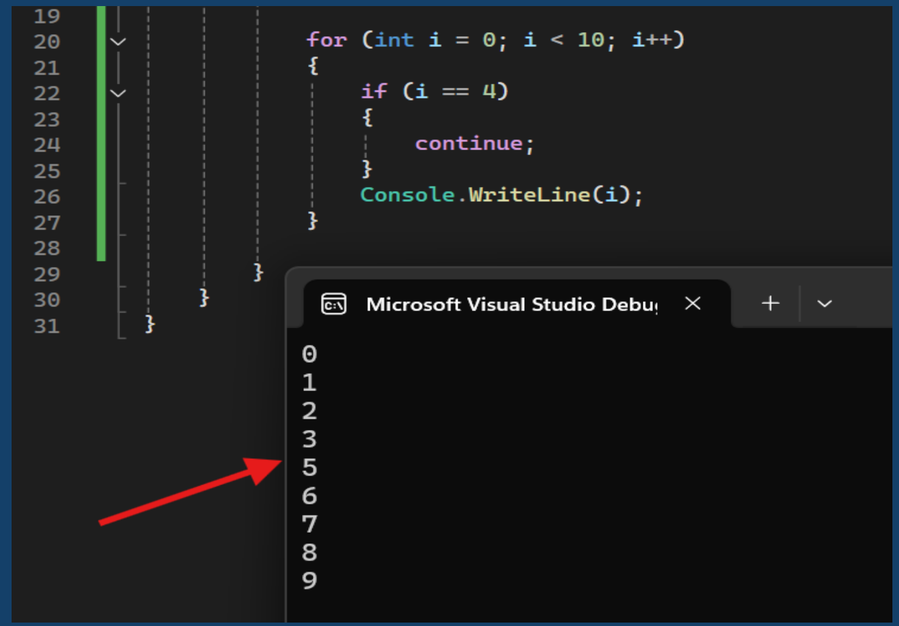


13. LOOP

Break and Continue

- **BREAK:** The break statement can also be used to jump out of a loop.
- **CONTINUE:** The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.





14. Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value

```
string[] carsList = { "Volvo", "BMW", "Ford", "Mazda" };
int[] myNum = { 10, 20, 30, 40 };
```

Access the Elements of an Array:

```
string[] cars = { "Volvo", "BMW", "Ford", "Mazda" };
Console.WriteLine(cars[0]);
// Outputs Volvo
```

Change an Array Element:

```
cars[0] = "Huyndai";
```

Array Length:

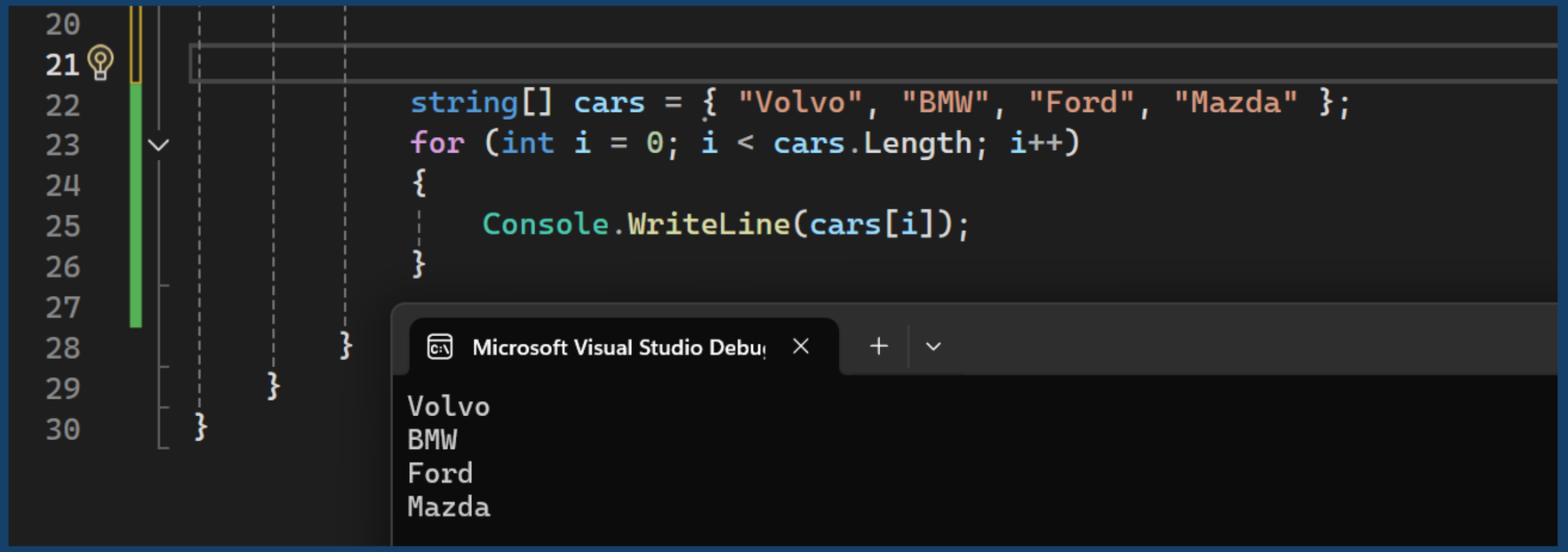
```
string[] cars = { "Volvo", "BMW", "Ford", "Mazda" };
Console.WriteLine(cars.Length);
// Outputs 4
```

Other Ways to Create an Array

```
// Create an array of four elements, and add values later
string[] carsClassA = new string[4];

// Create an array of four elements and add values right away
string[] carsClassB = new string[4] { "Volvo", "BMW", "Ford", "Mazda" };

// Create an array of four elements without specifying the size
string[] carsClassC = new string[] { "Volvo", "BMW", "Ford", "Mazda" };

// Create an array of four elements, omitting the new keyword, and without specifying the size
string[] carsClassD = { "Volvo", "BMW", "Ford", "Mazda" };
```

14. Arrays

Loop Through Arrays



15. Methods

```
class Program
{
  static void MyMethod()
  {
    // code to be executed
  }
}
```

MyMethod(): is the name of the method

static means that the method belongs to the Program class and not an object of the Program class.

void means that this method does not have a return value. You can choose another type like int, string, array, object,….

15. Methods

Call a Method

```
using System;
using HelloWorldx;

namespace HelloWorld
{
    0 references
    class Program
    {
        1 reference
        static void MyMethod()
        {
            Console.WriteLine("I just got executed!");
        }

        0 references
        static void Main(string[] args)
        {

            MyMethod();


        }
    }
}
```

15. Methods

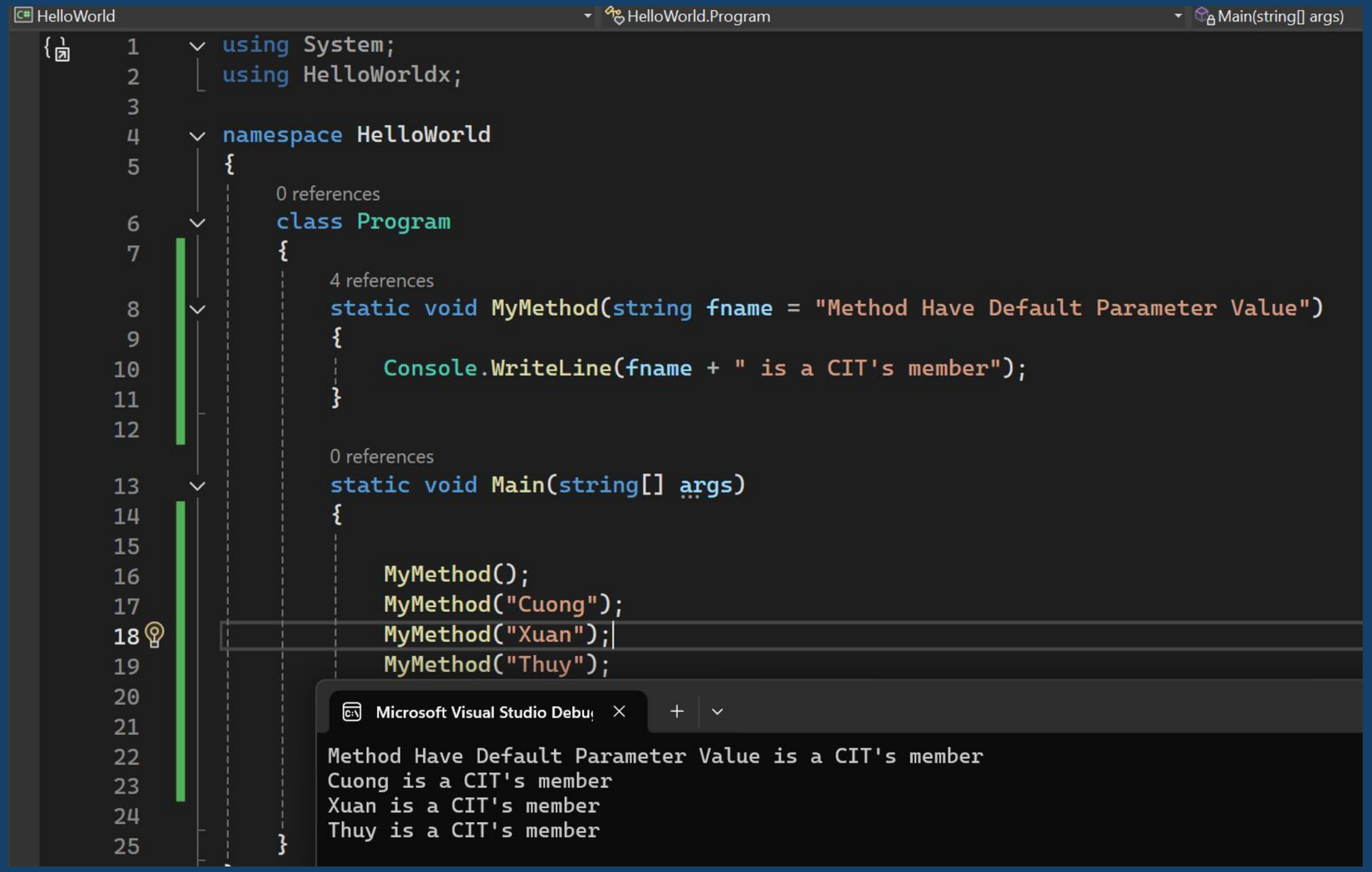
Method Parameters

```
0 references
class Program
{
    3 references
    static void MyMethod(string fname)
    {
        Console.WriteLine(fname + " is a CIT's member");
    }

    0 references
    static void Main(string[] args)
    {

        MyMethod("Cuong");
        MyMethod("Xuan");
        MyMethod("Thuy");
```

```
Microsoft Visual Studio Debu
Cuong is a CIT's member
Xuan is a CIT's member
Thuy is a CIT's member
```

15. Methods

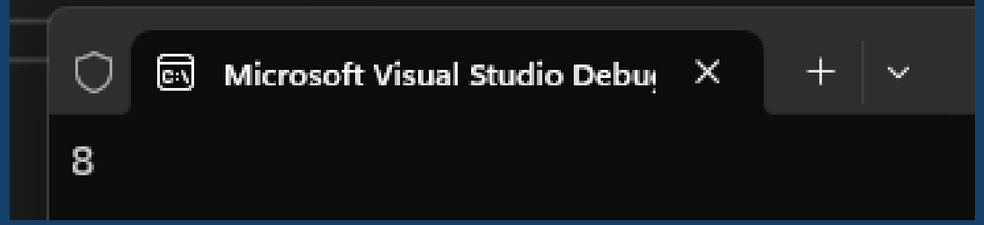
Default Parameter Value



15. Methods

Return Values

```
1 reference
static int MyMethod(int x)
{
    return 5 + x;
}

0 references
static void Main(string[] args)
{
    Console.WriteLine(MyMethod(3));
}

// Outputs 8 (5 + 3)
```



15. Methods

Named Arguments

```
1 reference
static void MyMethod(string child1, string child2, string child3)
{
    Console.WriteLine("The youngest child is: " + child3);
}

0 references
static void Main(string[] args)
{
    MyMethod(child3: "John", child1: "Liam", child2: "Liam");
}
```

```
Microsoft Visual Studio Debu
The youngest child is: John
```

EIU

Object-oriented programming with C#

1. What is OOP?

- OOP stands for Object-Oriented Programming.
- Object-oriented programming has several advantages over procedural programming:
  - OOP is faster and easier to execute
  - OOP provides a clear structure for the programs
  - OOP makes the code easier to maintain, modify and debug
  - OOP makes it possible to create full reusable applications with less code and shorter development time
- Classes and objects are the two main aspects of object-oriented programming.
- A class is a template for objects, and an object is an instance of a class

2. Classes and Objects

To create a class, use the class keyword:

```
// Create a class named "Car" with a variable color:
0 references
class Car
{
    string color = "red";
}
```

Create an Object: Create an object called "myObj" and use it to print the value of color:

```
class Car
{
    string color = "red";

    0 references
    static void Main(string[] args)
    {
        Car myObj = new Car();
        Console.WriteLine(myObj.color);
    }
}
```

```
Microsoft Visual Studio Debu
red
```

3. Class Members

Fields and methods inside classes are often referred to as "Class Members":

```
6       class MyClass
7       {
8           // Class members
9           string color = "red";          // field
10          int maxSpeed = 200;            // field
            0 references
11          public void fullThrottle()     // method
12          {
13              Console.WriteLine("The car is going as fast as it can!");
14          }
15      }
```

3. Class Members

Fields

Variables inside a class are called fields, you can access them by creating an object of the class, and by using the dot syntax (.).

The following example will create an object of the Car class, with the name myObj. Then we print the value of the fields color and maxSpeed:

```
17      class Car
18      {
19          string color = "red";
20          int maxSpeed = 200;
21
            0 references
22          static void Main(string[] args)
23          {
24              Car myObj = new Car();
25              Console.WriteLine(myObj.color);
26              Console.WriteLine(myObj.maxSpeed);
27          }
28      }
```

3. Class Members

Object Methods

Methods normally belong to a class, and they define how an object of a class behaves.

Just like with fields, you can access methods with the dot syntax. However, note that the method must be public. And remember that we use the name of the method followed by two parentheses () and a semicolon ; to call (execute) the method:

```
2 references
class Car
{
    string color;                // field
    int maxSpeed;                // field
    1 reference
    public void fullThrottle()   // method
    {
        Console.WriteLine("The car is going as fast as it can!");
    }

    0 references
    static void Main(string[] args)
    {
        Car myObj = new Car();
        myObj.fullThrottle();  // Call the method
    }
}
```

4. Constructors

A constructor is a special method that is used to initialize objects. The advantage of a constructor, is that it is called when an object of a class is created. It can be used to set initial values for fields:

```
// Create a Car class
3 references
class Car
{
    public string model;  // Create a field

    // Create a class constructor for the Car class
    1 reference
    public Car()
    {
        model = "Mustang"; // Set the initial value for model
    }

    0 references
    static void Main(string[] args)
    {
        Car Ford = new Car();  // Create an object of the Car Class (this will call the constructor)
        Console.WriteLine(Ford.model);  // Print the value of model
    }
}

// Outputs "Mustang"
```

4. Constructors

Constructor Parameters

Constructors can also take parameters, which is used to initialize fields.

The following example adds a string modelName parameter to the constructor. Inside the constructor we set model to modelName (model=modelName). When we call the constructor, we pass a parameter to the constructor ("Mustang"), which will set the value of model to "Mustang":

```
3 references
class Car
{
    public string model;

    // Create a class constructor with a parameter
    1 reference
    public Car(string modelName)
    {
        model = modelName;
    }

    0 references
    static void Main(string[] args)
    {
        Car Ford = new Car("Honda");
        Console.WriteLine(Ford.model);
    }
}

// Outputs "Honda"
```

```
Microsoft Visual Studio Debu
Honda
```

5. Access Modifiers

public string color;

The public keyword is an access modifier, which is used to set the access level/visibility for classes, fields, methods and properties.

C# has the following access modifiers:

Modifier	Description
public	The code is accessible for all classes
private	The code is only accessible within the same class
protected	The code is accessible within the same class, or in a class that is inherited from that class.
internal	The code is only accessible within its own assembly, but not from another assembly.

5. Access Modifiers

Private Modifier

If you declare a field with a private access modifier, it can only be accessed within the same class.

If you try to access it outside the class, an error will occur.

```
2 references
class Car
{
    private string model = "Mustang";

    0 references
    static void Main(string[] args)
    {
        Car myObj = new Car();
        Console.WriteLine(myObj.model);
    }
}
// output is Mustang
```

```
Microsoft Visual Studio Debu
Mustang
```

```
class Car
{
    private string model = "Mustang";
}

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Car myObj = new Car();
        Console.WriteLine(myObj.model);
    }
}
```

Code	Description
CS0122	'Car.model' is inaccessible due to its protection level

5. Access Modifiers

WHY ACCESS MODIFIERS?

To control the visibility of class members (the security level of each individual class and class member).

To achieve "Encapsulation" - which is the process of making sure that "sensitive" data is hidden from users. This is done by declaring fields as private.

6. Properties (Get and Set)

Properties and Encapsulation

Encapsulation mean is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

- Declare fields/variables as private
- Provide public get and set methods, through properties, to access and update the value of a private field

6. Properties (Get and Set)

Properties

- Private variables can only be accessed within the same class (an outside class has no access to it)
- A property is like a combination of a variable and a method, and it has two methods: a get and a set method:

```
class Program
{
    2 references
    class Course
    {
        private string nameCourse; // field

        2 references
        public string NameCourse   // property
        {
            get { return nameCourse; }   // get method
            set { nameCourse = value; }  // set method
        }
    }
    0 references
    static void Main(string[] args)
    {
        Course myObj = new Course();
        myObj.NameCourse = ".NET Programming"; // Using properties to access field
        Console.WriteLine(myObj.NameCourse);
    }
}
```

```
Microsoft Visual Studio Debu
.NET Programming
```

6. Properties (Get and Set)

Automatic Properties (Short Hand)

C# also provides a way to use short-hand / automatic properties, where you do not have to define the field for the property, and you only have to write get; and set; inside the property.

The result is the same; the only difference is less code.

```
namespace HelloWorld
{
    0 references
    class Program
    {
        2 references
        class Course
        {
            2 references
            public string NameCourse  // property
                { get; set; }  // GET SET method
        }
        0 references
        static void Main(string[] args)
        {
            Course myObj = new Course();
            myObj.NameCourse = ".NET Programming"; // Using properties to access field
            Console.WriteLine(myObj.NameCourse);
        }
    }
}
```

```
Microsoft Visual Studio Debu
.NET Programming
```

7. Inheritance

Inheritance (Derived and Base Class)
In C#, it is possible to inherit fields and methods from one class to another. We group the "inheritance concept" into two categories:

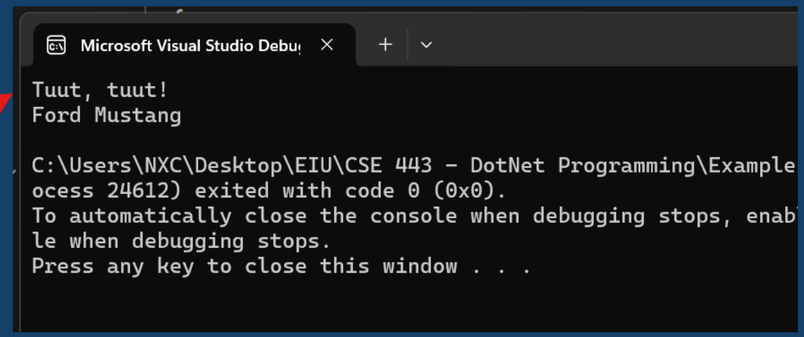
Derived Class (child) - the class that inherits from another class
Base Class (parent) - the class being inherited from

To inherit from a class, use the “ : ” symbol.

7. Inheritance

Example:

```
1 reference
class Vehicle  // base class (parent)
{
    public string brand = "Ford";  // Vehicle field
    1 reference
    public void honk()             // Vehicle method
    {
        Console.WriteLine("Tuut, tuut!");
    }
}
2 references
class Car : Vehicle  // derived class (child)
{
    public string modelName = "Mustang";  // Car field
}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        // Create a myCar object
        Car myCar = new Car();

        // Call the honk() method (From the Vehicle class) on the myCar object
        myCar.honk();

        // Display the value of the brand field (from the Vehicle class)
        // and the value of the modelName from the Car class
        Console.WriteLine(myCar.brand + " " + myCar.modelName);
    }
}
```



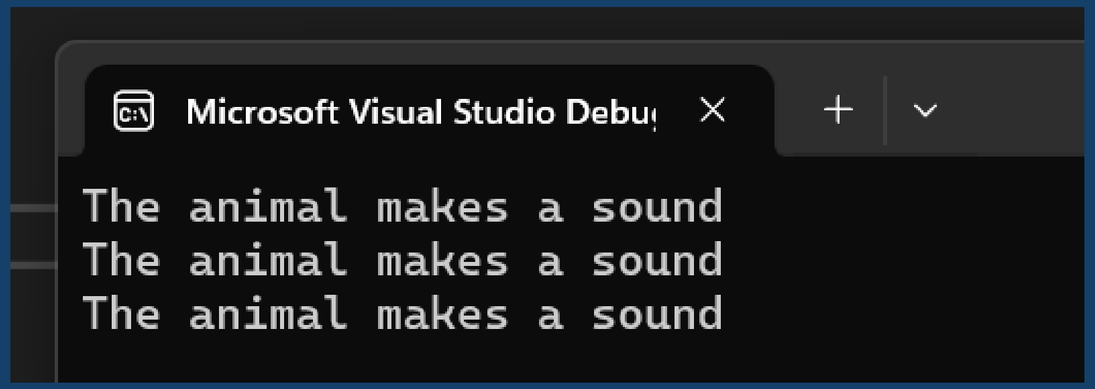
8. Polymorphism

Polymorphism and Overriding Methods

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

```
6 references
class Animal  // Base class (parent)
{
    3 references
    public void animalSound()
    {
        Console.WriteLine("The animal makes a sound");
    }
}
1 reference
class Pig : Animal  // Derived class (child)
{
    0 references
    public void animalSound()
    {
        Console.WriteLine("The pig says: wee wee");
    }
}
1 reference
class Dog : Animal  // Derived class (child)
{
    0 references
    public void animalSound()
    {
        Console.WriteLine("The dog says: bow wow");
    }
}
```

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Animal myAnimal = new Animal();  // Create a Animal object
        Animal myPig = new Pig();  // Create a Pig object
        Animal myDog = new Dog();  // Create a Dog object
        myAnimal.animalSound();
        myPig.animalSound();
        myDog.animalSound();
    }
}
```



8. Polymorphism

C# provides an option to override the base class method, by adding the virtual keyword to the method inside the base class, and by using the override keyword for each derived class methods

```
6 references
class Animal  // Base class (parent)
{
    5 references
    public virtual void animalSound()
    {
        Console.WriteLine("The animal makes a sound");
    }
}

1 reference
class Pig : Animal  // Derived class (child)
{
    4 references
    public override void animalSound()
    {
        Console.WriteLine("The pig says: wee wee");
    }
}

1 reference
class Dog : Animal  // Derived class (child)
{
    4 references
    public override void animalSound()
    {
        Console.WriteLine("The dog says: bow wow");
    }
}
```

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Animal myAnimal = new Animal();  // Create a Animal object
        Animal myPig = new Pig();  // Create a Pig object
        Animal myDog = new Dog();  // Create a Dog object

        myAnimal.animalSound();
        myPig.animalSound();
        myDog.animalSound();
    }
}
```

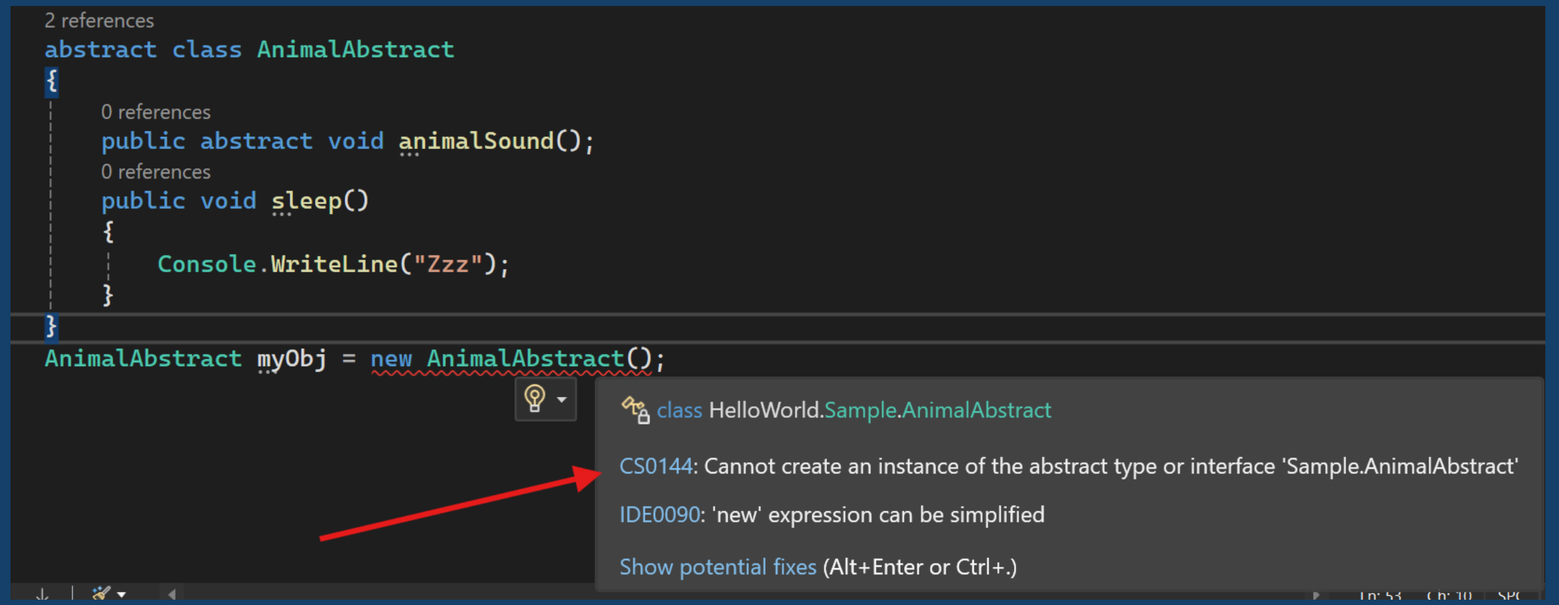
```
Microsoft Visual Studio Debu
The animal makes a sound
The pig says: wee wee
The dog says: bow wow
```

9. Abstract

The abstract keyword is used for classes and methods:

Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the derived class (inherited from).
An abstract class can have both abstract and regular methods:



9. Abstract

Example:

```
// Abstract class
1 reference
abstract class Animal
{
    // Abstract method (does not have a body)
    2 references
    public abstract void animalSound();
    // Regular method
    1 reference
    public void sleep()
    {
        Console.WriteLine("Zzz");
    }
}
// Derived class (inherit from Animal)
2 references
class Pig : Animal
{
    2 references
    public override void animalSound()
    {
        // The body of animalSound() is provided here
        Console.WriteLine("The pig says: wee wee");
    }
}
```

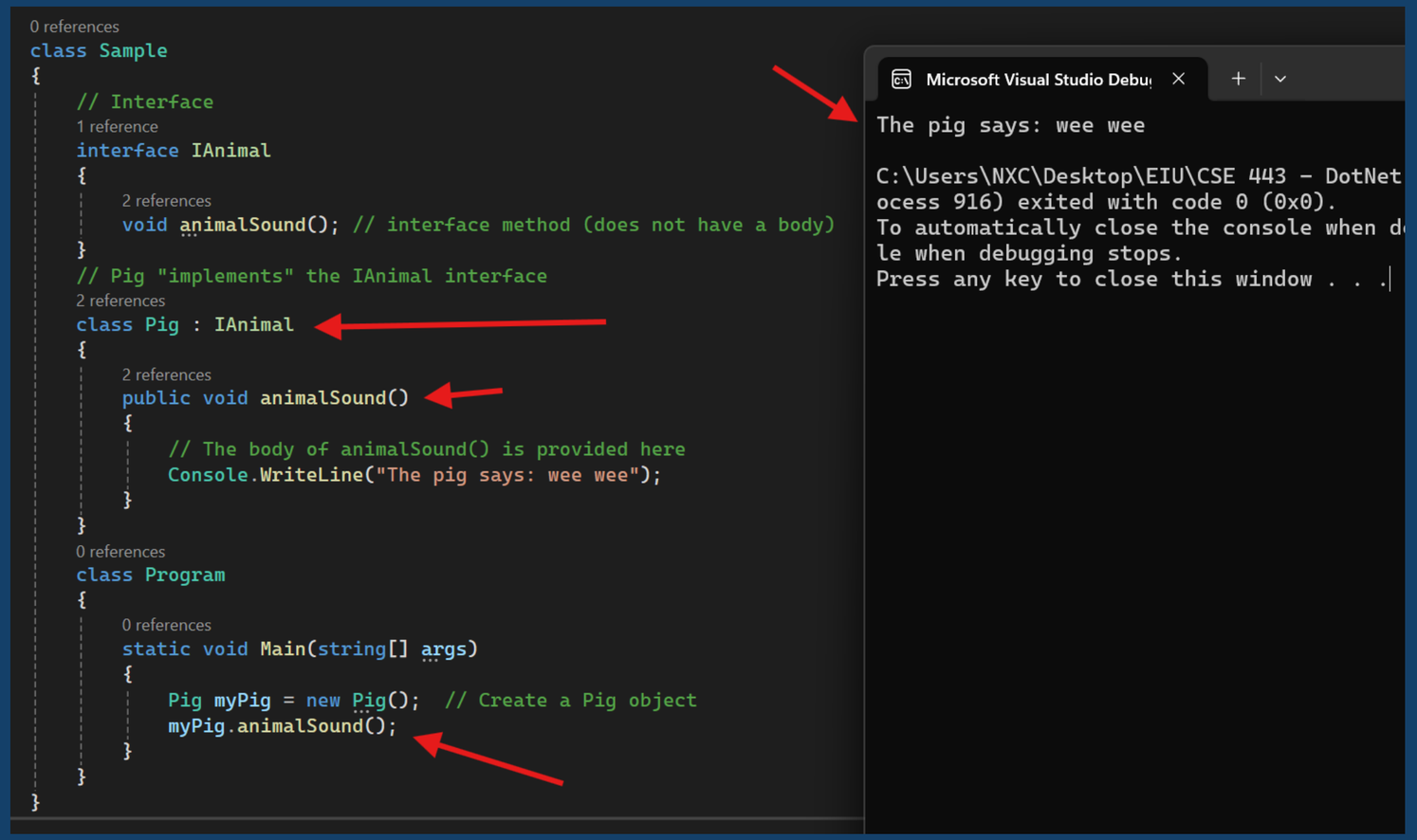
```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Pig myPig = new Pig(); // Create a Pig object
        myPig.animalSound();  // Call the abstract method
        myPig.sleep();  // Call the regular method
    }
}
```

```
Microsoft Visual Studio Debu
The pig says: wee wee
Zzz
```

9. Interface

Another way to achieve abstraction in C#, is with interfaces.

An **interface** is a completely "**abstract class**", which can only contain abstract methods and properties (with **empty bodies**):



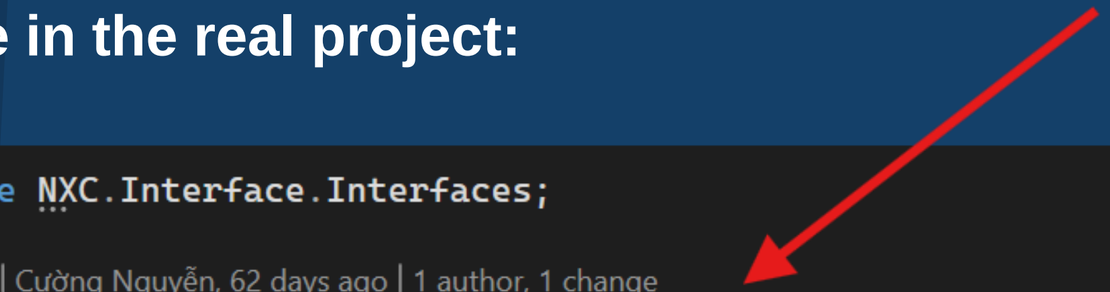
9. Interface

Example in the real project:

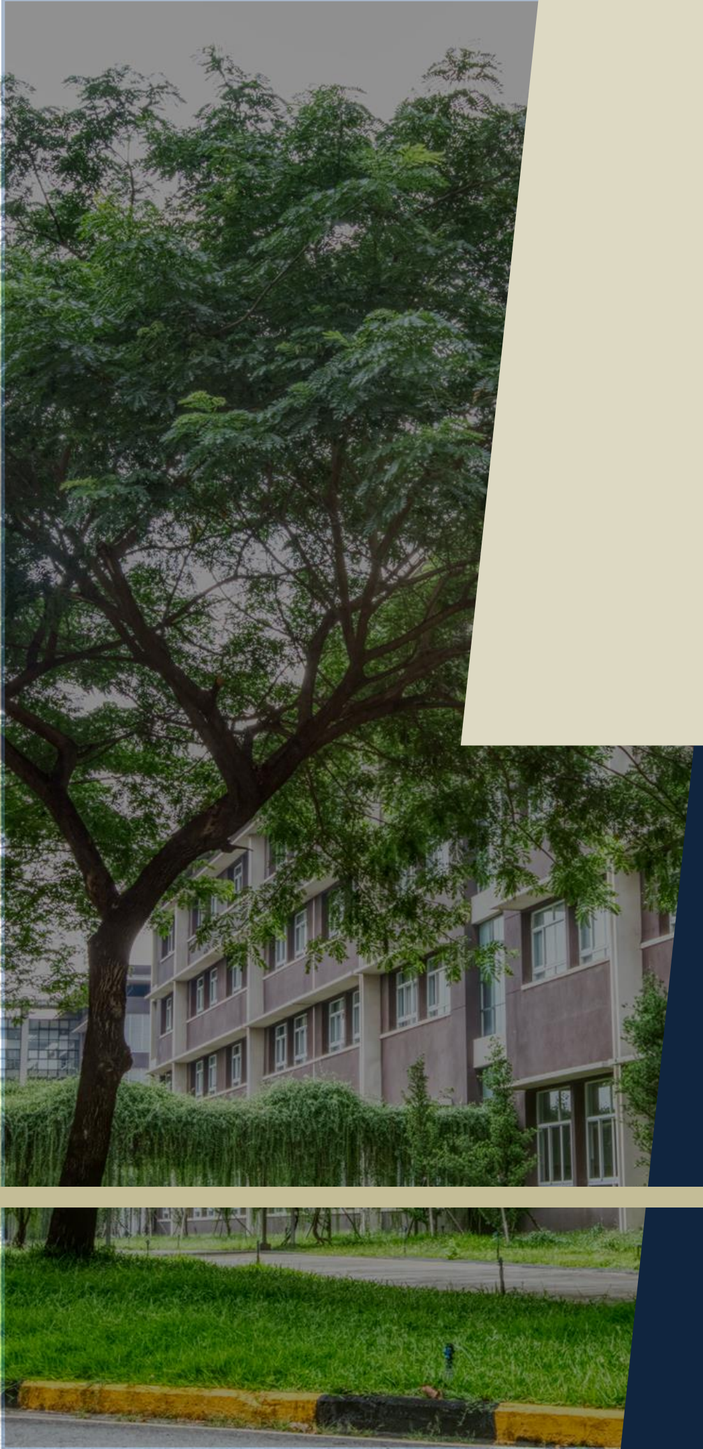
```
namespace NXC.Interface;

//this interface to define a common interface
//as a Repository design pattern we'll define a common interface and the other interface will be implement from this
//some common method like a getAll, getById, insert, update, delete ...
71 references | Cường Nguyễn, 62 days ago | 1 author, 1 change
public interface IRepository<T> where T : class
{
    99+ references | Cường Nguyễn, 62 days ago | 1 author, 1 change
    Task<TemplateApi<T>> GetAllAsync(int pageNumber, int pageSize);
    99+ references | Cường Nguyễn, 62 days ago | 1 author, 1 change
    Task<TemplateApi<T>> GetById(Guid id);
    78 references | Cường Nguyễn, 62 days ago | 1 author, 1 change
    Task<TemplateApi<T>> GetAllAvailable(int pageNumber, int pageSize);
    99+ references | Cường Nguyễn, 62 days ago | 1 author, 1 change
    Task<TemplateApi<T>> Update(T model, Guid idUserCurrent, string fullName);
    99+ references | Cường Nguyễn, 62 days ago | 1 author, 1 change
    Task<TemplateApi<T>> Insert(T model, Guid idUserCurrent, string fullName);
    99+ references | Cường Nguyễn, 62 days ago | 1 author, 1 change
    Task<TemplateApi<T>> RemoveByList(List<Guid> ids, Guid idUserCurrent, string fullName);
    78 references | Cường Nguyễn, 62 days ago | 1 author, 1 change
    Task<TemplateApi<T>> HideByList(List<Guid> ids, bool isLock, Guid idUserCurrent, string fullName);
}
```

9. Interface

Example in the real project:



```
namespace NXC.Interface.Interfaces;

5 references | Cường Nguyễn, 62 days ago | 1 author, 1 change
public interface IEmployeeRepository : IRepository<EmployeeDto>
{
    #region ===[ CRUD TABLE Employee ]==========================================================

    2 references | Cường Nguyễn, 62 days ago | 1 author, 1 change
    Task<TemplateApi<EmployeeAndBenefits>> GetEmployeeAndBenefits(Guid idEmployee);
    2 references | Cường Nguyễn, 62 days ago | 1 author, 1 change
    Task<TemplateApi<EmployeeAndAllowance>> GetEmployeeAndAllowance(Guid idEmployee);
    2 references | Cường Nguyễn, 62 days ago | 1 author, 1 change
    Task<TemplateApi<EmployeeDto>> GetEmployeeResigned(int pageNumber, int pageSize);
    2 references | Cường Nguyễn, 62 days ago | 1 author, 1 change
    Task<TemplateApi<EmployeeDto>> FilterEmployee(FilterEmployeeModel model,int pageNumber, int pageSize);

    2 references | Cường Nguyễn, 62 days ago | 1 author, 1 change
    Task<TemplateApi<EmployeeDto>> UpdateEmployeeType(Guid idEmployee, Guid typeOfEmployee,
        Guid idUserCurrent, string fullName);

    #endregion
}
```



EIU
TRƯỜNG ĐẠI HỌC
QUỐC TẾ
MIỀN ĐÔNG
EASTERN
INTERNATIONAL
UNIVERSITY

Start your future at EIU

Thank You