EAST ASIA UNIVERSITY OF TECHNOLOGY

LẬP TRÌNH MẠNG
(Network Programming)

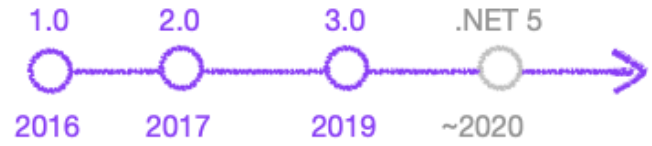
Nguyễn Anh Thơ
natho5578@gmail.com

BỔ SUNG KIẾN THỨC C#

- Giới thiệu về C# và .NET FRAMEWORK

.NET FRAMEWORK

.NET Framework



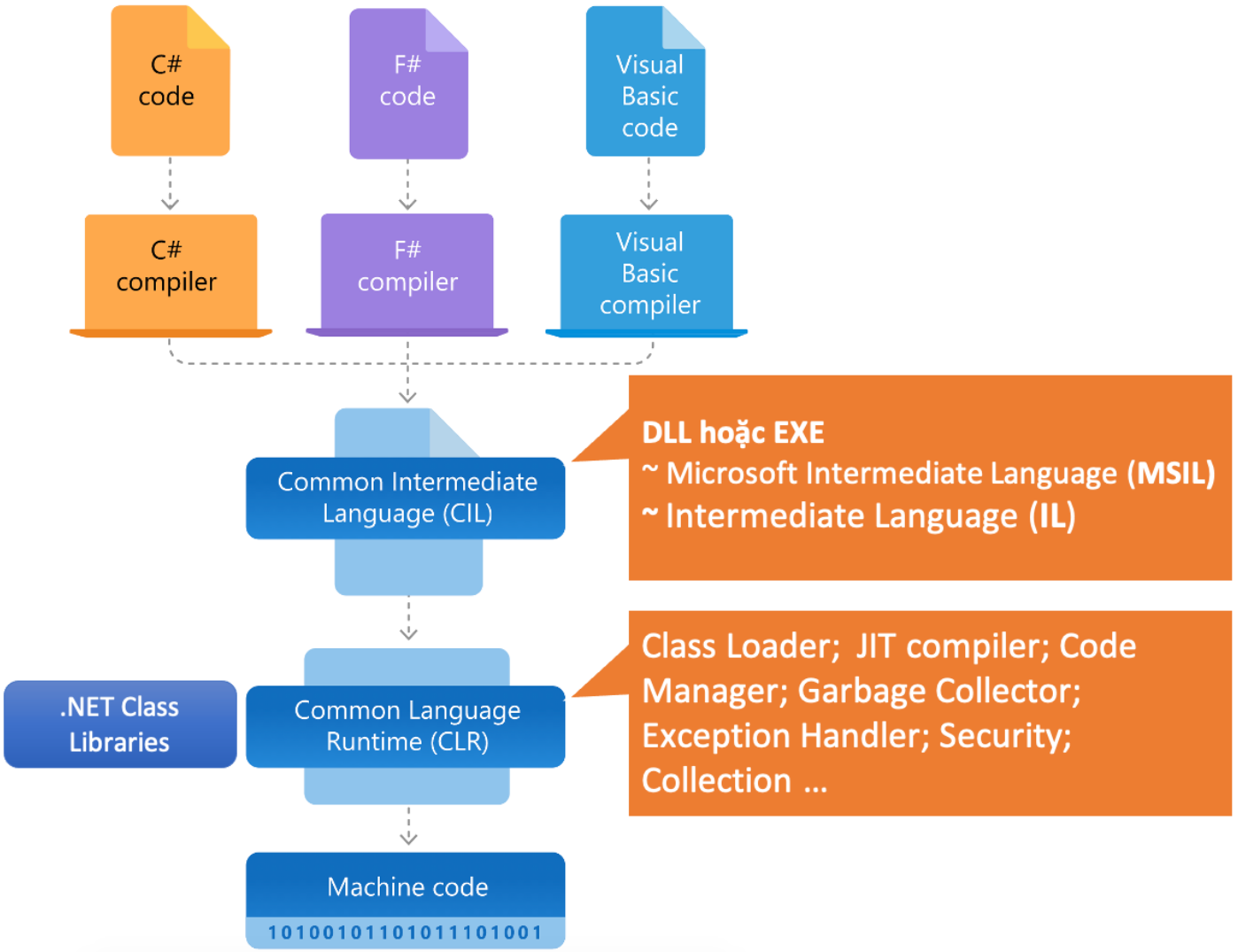
.NET Core

.NET bao gồm thành phần:

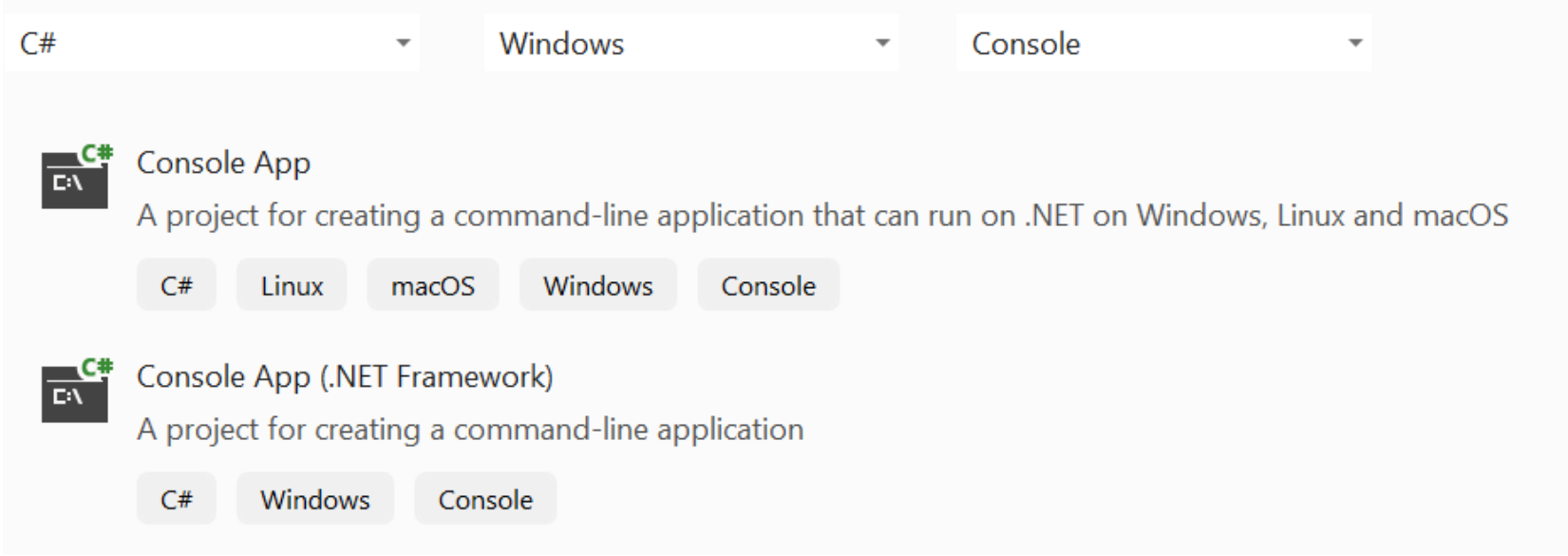
CLR (Common Language Runtime): Thực thi ứng dụng trong môi trường .Net

Framework Class Library (**FCL**): là một tập hợp các lớp (class), giao diện (interface), kiểu giá trị, chúng cho phép bạn sử dụng để thi hành rất nhiều tác vụ trong lập trình: ví dụ như xử lý dữ liệu, truy cập file, làm việc với văn bản ..

.NET FRAMEWORK



Tạo Project




```
Program
0 references
1  internal class Program
2  {
       0 references
3      private static void Main(string[] args)
4      {
5          Console.WriteLine("Hello, World!");
6      }
7  }
```

Chương trình

```
using System;

namespace CS001_HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Xin chào C# NET CORE!");
        }
    }
}
```

Chạy chương trình: Ctrl+F5

Giải thích

- Phương thức Main
  - static cho biết hàm tên Main không thuộc về đối tượng nào cả (nó thuộc về lớp)
  - void cho biết hàm Main không trả về giá trị (ngoài ra bạn có thể khai bao nếu cần hàm trả về giá trị như int là số nguyên)
  - Main tên hàm, đặt tên là Main vì nó là điểm khởi đầu của chương trình
  - args là tham số truyền vào hàm Main, nó là một mảng các chuỗi (string[]), tham số này truyền vào khi ở dòng lệnh
- Lớp - Class
- Namespace:
- using System; // Cho biết lớp sử dụng Namespace System
- WriteLine(str); // in chuỗi str

Lớp (Class)

Khai báo lớp

Cú pháp cơ bản như sau:

```
<Access Modifiers> class Class_Name {
    // khai báo các thành viên dữ  liệu (thuộc tính, biến trường dữ liệu)
     // khai báo các thành viên  hàm (phương thức)
}
```

Access Modifiers

- public : không giới hạn phạm vi truy cập
- protected : chỉ truy cập trong nội bộ lớp hay các lớp kế thừa
- private : **(mặc định)** chỉ truy cập được từ các thành viên của lớp chứa nó
- internal : chỉ truy cập được trong cùng assembly (dll, exe)
- protected internal: truy cập được khi cùng assembly hoặc lớp kế thừa
- private protected: truy cập từ lớp chứa nó, lớp kế thừa nhưng phải cùng assembly

Phương thức khởi tạo (Constructor)

```
private string name;
private decimal price;

// Khai báo phương thức khởi tạo với 2 tham số
public Product(string nameproduct, decimal priceproduct)
{
    name = nameproduct;
    price = priceproduct;
}

// Khai báo phương thức khởi tạo không tham số
public Product()
{
    name = "Không tên";
    price = 0;
}
```

Phương thức set/get

```
// Thuộc tính Name lấy hoặc thiết lập tên sản phẩm
public string Name
{
    set { name = value;}
    get { return name;}
}
```

Các hàm có thể dùng =>

```
private int so;
0 references
public Sohoc(int num) =>so = num;
0 references
public int num
{
    set=>so = value;
    get => so;
}
```

```
0 references
public int GetSo() { return so; }
0 references
public void SetSo(int num) { so = num;}
```

Sử dụng các phương thức trong lớp

```
0 references
private static void Main(string[] args)
{
    // Tạo đối tượng không truyền tham số
    Sohoc sohoc = new Sohoc();
    Console.WriteLine(sohoc.num);
    // Tạo đối tượng có truyền tham số
    Sohoc sohoc1 = new Sohoc(100);
    Console.WriteLine(sohoc1.num);

}
```

Phương thức kế thừa

```
1 reference
internal class SoNguyen : Sohoc
{
    private String Mota;
    0 references
    public SoNguyen(int num, String des):base(num) {
    Mota = des;}
}
```

Phương thức khởi tạo tĩnh(static)

```
// Phương thức khởi tạo tĩnh
 reference
class Colores
{
    private static String name_color;
    private static int index_color;
    0 references
    public Colores() {
        name_color = "Red";
        index_color = 0;
    }
}
```

Gọi phương thức Static

```
Colores colores = new Colores();
Console.WriteLine(Colores.name_color);
```

Khi khai báo từ khóa static thì khi dùng phương thức không cần khởi tạo

Tính kế thừa trong lập trình C# C Sharp

- **Lớp cơ sở** là lớp mà được lớp khác kế thừa.
- **Lớp kế thừa** là lớp kế thừa lại các thuộc tính, phương thức từ lớp cơ sở.

Khai báo lớp cơ sở

```
class Animal {
    public int Legs {get; set;}
    public float Weigh {get; set;}


    public void ShowLegs()
    {
        Console.WriteLine($"Legs: {Legs}");
    }
}
```

Khai báo lớp kế thừa

```
class Cat : Animal {
    public string food;      // thuộc tính mới thêm

    public Cat()
    {
        Legs = 4;           // Thuộc tính Legs có sẵn - vì nó kế thừa từ Animal
        food = "Mouse";
    }

    public void Eat()
    {
        Console.WriteLine(meal);
    }
}
```

Khi sử dụng Cat

```
Cat cat = new Cat();
cat.ShowLegs();             // Phương thức này kế thừa từ lớp cơ sở
cat.Eat();                  // phương thức của riêng Cat
```

Từ khóa protected trong lớp cơ sở

- Thành viên được bảo vệ (protected) của lớp cơ sở
- Ví dụ khai báo:

  protected int Legs {**get**; **set**;}

```
Cat cat = new Cat();
int l = cat.Legs;  // Lỗi - Legs không cho phép truy cập từ code bên ngoài lớp
```

Lớp niêm phong sealed

```
sealed class A {

}

class B : A {  // Chỗ này lỗi vì kế thừa lớp bị niêm phong

}
```

Namespace là gì trong C#

- Namespace là cách tổ chức nhóm code (các lớp, giao diện, cấu trúc ...) thành những nhóm, tạo ra phạm vi hoạt động của các thành phần trong nhóm.

```
namespace mynamespace {
    // Định nghĩa các lớp, cấu trúc ...
}
```

```
using mynamespace;
using mynamespace; ...
```

Namespace lồng nhau, nhiều cấp

```
namespace A {
    // Định nghĩa các lớp, cấu trúc ...
    namespace B {
        // Định nghĩa các lớp, cấu trúc ...
    }
}
```

```
using A.B;
```

Mảng - Khai báo mảng trong C#

- Khởi tạo new datatype[n] tạo ra mảng có kiểu datatype và có thể lưu n phần tử
- bienMang = **new** int[5];
- Hoặc

  int[] bienMang = new int[5];

  string[] studentNames = **new** string[10];

```
// mảng 3 phần tử chuỗi ký tự, mỗi phần tử được gán ngay giá trị chuỗi cụ thể
string[] productNames = new string[3] {"Iphone", "Samsung", "Nokia"};
// mảng 3 phần tử double, mỗi phần tử được gán giá trị luôn
double[] productPrices = new double[3] {100, 200.5, 10.1};
```

Thuộc tính và phương thức

- Thuộc tính và phương thức đối tượng mảng
- System.Array

Member	Nội dung
`Length`	Thuộc tính cho biết số lượng phần tử trong mảng
`Rank`	Thuộc tính cho biết số chiều mảng
`Clone()`	Copy (nhân bản) đối tượng mảng
`GetValue(index)`	Lấy giá trị phần tử trong mảng
`Min()`	Trả về giá trị nhỏ nhất trong mảng
`Max()`	Trả về giá trị lớn nhất trong mảng
`Sum()`	Trả về giá trị tổng cộng các phần tử

Thuộc tính và phương thức (static)

Một số phương thức tĩnh trong Array áp dụng vào mảng như:

Member	Nội dung
Array.BinarySearch(array, value)	Tìm kiếm phần tử trong mảng đã được sắp xếp, trả về chỉ số nếu tìm thấy
CopyTo(array, indexStart)	Sao chép phần tử mảng này sang mảng khác
Array.Clear(array, index, length)	Thiết lập phần tử mảng nhận giá trị mặc định
bool Exists<T> (array, Predicate<T> match);	Kiểm tra có phần tử trong mảng thỏa mãn match
Fill<T> (array, value);	Gán các phần tử của mảng bằng value
T Find<T> (array, Predicate<T> match);	Tìm phần tử mảng
int FindIndex<T> (array, Predicate<T> match);	Tìm phần tử mảng, trả về chỉ số nếu thấy
T[] FindAll<T> (array, Predicate<T> match);	Tìm tất cả phần tử mảng
int IndexOf(array, value)	Tìm chỉ số của phần tử
ForEach(array, Action<T> action)	Thi hành action trên mỗi phần tử
Sort(array)	Sắp xếp

Duyệt mảng

Ví dụ, duyệt qua các phần tử mảng kiểu int, và in ra nội dung phần tử

```
static void testForEach()
{
    int[] numbers = {9, 8, 7, 6, 5, 4, 3, 2, 1 };
    Array.ForEach<int>(numbers, (int n) => {
        Console.WriteLine(n);
    });
}
```

```
int[] myArray = {1,3,5,19, 10, 20, 40, 40};
int maxIndex = myArray.Length - 1;
for (int idx = 0; idx <= maxIndex; idx ++) {
    Console.WriteLine(myArray[idx]);
}
```

Duyệt mảng

```
int[] myArray = {1,3,5,19, 10, 20, 40, 40};
foreach (int element in myArray)
{
    Console.WriteLine(element);
}
```

```
foreach (var e in vararry) {
    // ...
}
```

Mảng nhiều chiều (rank)

- type[, , ... ,] varname = **new** type[size1, size2, ..., sizeN];
- int[,] myvar = **new** int[3,4];

```
int[,] myvar = new int[3,4] {{1,2,3,4}, {0,3,1,3}, {4,2,3,4}};

for (int i = 0; i <= 2; i ++)
{
    for(int j = 0; j <=3; j++)
    {
        Console.Write(myvar[i,j] + " ");
    }
    Console.WriteLine();
}
```

Mảng trong mảng

```
int[][] myArray = new int[][] {
                                    new int[] {1,2},
                                    new int[] {2,5,6},
                                    new int[] {2,3},
                                    new int[] {2,3,4,5,5}
                                   };


foreach (var arr in myArray) {
    foreach (var e in arr) {
        Console.Write(e + " ");
    }
    Console.WriteLine();
}
```

QUESTION ?