FPT Education

FPT POLYTECHNIC

LẬP TRÌNH C#2

BÀI 1: STATIC CLASS, PARTIAL CLASS, GENERIC NAMESPACE

https://caodang.fpt.edu.vn

MỤC TIÊU

- Static class, Partial class
- Generic Namespace

NỘI DUNG

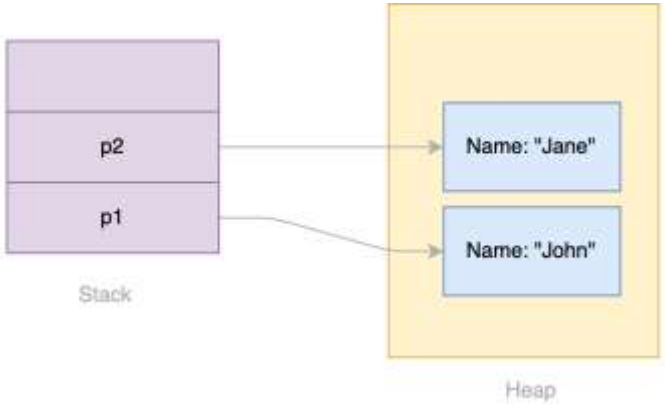
- Phần I: Static class, Partial class
  - Static members and class
  - Partial class
- Phần II: Generic Namespace
  - HashSet<T> Class
  - LinkedList<T> Class
  - List<T> Class

Thuộc tính (Attributes)

- **Thuộc tính** (hay còn gọi là **biến thành viên**) là các dữ liệu thuộc về mỗi đối tượng.
- Mỗi đối tượng có riêng một bản sao của thuộc tính, nên giá trị của thuộc tính có thể khác nhau giữa các đối tượng.
- Được truy cập thông qua tên của đối tượng.

STATIC CLASS

```
public class Person
{
    public readonly string Name;
    2 usages
    public Person(string name)
    {
        Name = name;
    }
}
public class Test
{
    public void TestMethod()
    {
        var p1 = new Person("Jane");
        var p2 = new Person("John");
        Console.WriteLine($"Name: {p1.Name}");
        Console.WriteLine($"Name: {p2.Name}");
    }
}
```



STATIC CLASS

- Đặc điểm của thành viên tĩnh:
  - Được khởi tạo 1 lần duy nhất, không phụ thuộc vào số lượng đối tượng được tạo
  - Dùng chung cho mọi loại đối tượng chứ không phải thuộc về 1 đối tượng cụ thể
  - Gọi trực tiếp thông qua tên lớp
  - Thành viên tĩnh bị hủy khi chương trình kết thúc
- Có 4 loại thành viên tĩnh chính:
  - Biến tĩnh (static variable)
  - Phương thức tĩnh (static method)
  - Phương thức khởi tạo tĩnh (static constructor)
  - Lớp tĩnh (static class)

STATIC CLASS

- Biến tĩnh(static variable):

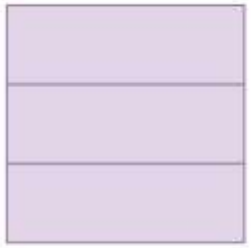
```
<phạm vi truy cập> static <kiểu dữ liệu> <tên biến>  =<giá trị khởi tạo>;
```

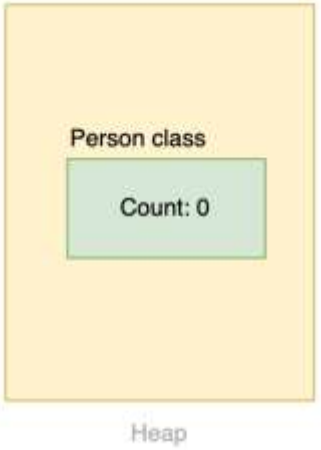
- Một số đặc điểm của **biến tĩnh**:
  - Biến tĩnh được khởi tạo một lần duy nhất ngay khi chương trình được nạp vào bộ nhớ để thực thi. Và nó Không phụ thuộc vào số lượng đối tượng được tạo.
  - Biến tĩnh được gọi trực tiếp thông qua tên của lớp.
  - Biến tĩnh thuộc về lớp chứ không phải thuộc về từng đối tượng cụ thể. Có thể truy cập mà không cần tạo đối tượng. Nên thường được sử dụng để định nghĩa cho các hằng số (constant)…
  - Biến tĩnh bị huỷ khi chương trình kết thúc.

STATIC CLASS

- Biến tĩnh(static variable):

```
public class Person
{
    public readonly string Name;
    public static int Count = 0;
    2 usages
    public Person(string name)
    {
        Name = name;
    }
}
```





STATIC CLASS

- Phương thức tĩnh(static method):

```
<phạm vi truy cập> static <kiểu dữ liệu trả về> <tên phương thức>
{
   // nội dung phương thức
}
```

- Một số đặc điểm của **phương thức tĩnh**:
  - Static method là một phương thức dùng chung của lớp. Được gọi thông qua tên lớp và không cần khởi tạo bất kỳ đối tượng nào, từ đó tránh việc lãng phí bộ nhớ..
  - Hỗ trợ trong việc viết các hàm tiện ích của thư viện để sử dụng lại.
  - Trong phương thức có sử dụng biến static thì phương thức đó cũng phải được khai báo là static.

STATIC CLASS

- Phương thức tĩnh(static method):

```
public class UnitConverter
{
    public static double KilogramsToPounds(double kilograms)
    {
        var result = kilograms * 2.20462;
        return result;
    }
    public static double PoundsToKilograms(double pounds)
    {
        var result = pounds / 2.20462;
        return result;
    }
}
```

```
// kg to pounds
public void TestMethod2()
{
    var pounds = UnitConverter.KilogramsToPounds(10);
    Console.WriteLine($"10 kg = {pounds} pounds");
}

// pounds to kg
public void TestMethod3()
{
    var kilograms = UnitConverter.PoundsToKilograms(22.0462);
    Console.WriteLine($"22.0462 pounds = {kilograms} kg");
}
```

STATIC CLASS

- Phương thức khởi tạo tĩnh(static constructor):

```
static <tên lớp>
{
   // nội dung của hàm dựng, hàm khởi tạo constructor
}
```

- Một số đặc điểm của **phương thức khởi tạo tĩnh**:
  - Phương thức khởi tạo tĩnh thường được sử dụng để khởi tạo các biến tĩnh trong lớp.
  - Nó tự động gọi khi truy cập một thành viên dữ liệu tĩnh lần đầu
  - Trong phương thức có sử dụng biến static thì phương thức đó cũng phải được khai báo là static.

STATIC CLASS

- Phương thức khởi tạo tĩnh(static constructor):

```
public class RandomNumber
{
    private static Random _random;

    static RandomNumber()
    {
        _random = new Random();
    }
    public int Get() => _random.Next();
}
```

```
public void TestMethod4()
{
    RandomNumber random = new();
    var number = random.Get();
    Console.WriteLine($"Random number: {number}");
}
```

STATIC CLASS

- Lớp tĩnh(static class):
  - static class chỉ chứa thành viên là static
  - static class không có thể hiện.
  - static class không chứa hàm constructor.
  - static class thường được dùng với mục đích khai báo một lớp tiện ích chứa các hàm tiện ích hoặc hằng số.

- Lớp tĩnh:

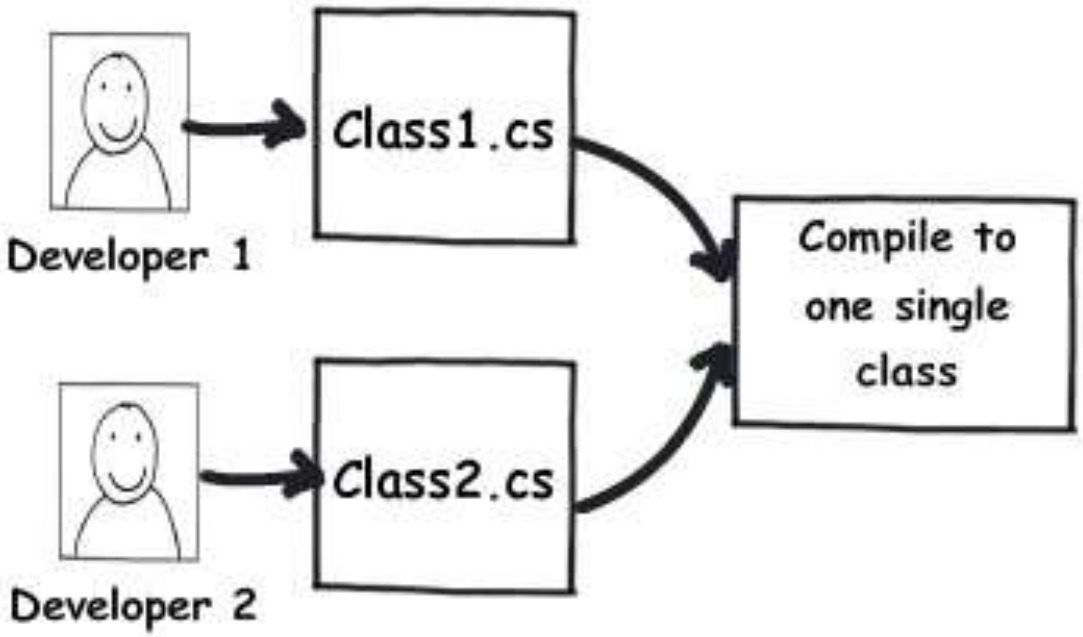
```
public static class LengthConverter
{
    1 usage
    public static double FeetToMeters(double feet)
    {
        return feet / 3.28084;
    }
    public static double MetersToFeet(double meters)
    {
        return meters * 3.28084;
    }
}
```

```
public void TestMethod5()
{
    var meters = LengthConverter.FeetToMeters(10);
    Console.WriteLine($"10 feet = {meters} meters");
}
```

STATIC CLASS

PARTIAL CLASS

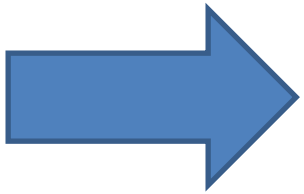
- Partial Class trong C# là một tính năng giúp chúng ta chia một class thành hai hay nhiều phần hay file khác nhau.



PARTIAL CLASS

```
public partial class ProductGaming
{
    private readonly string _name;
    public ProductGaming() { }
    1 usage
    public ProductGaming(string name)
    {
        _name = name;
    }
}
```

```
public partial class ProductGaming
{
    1 usage
    public void GetDetailProductGaming()
    {
        Debug.Log("name: " + _name);
    }
}
```



```
public class ProductGaming
{
    private readonly string _name;
    public ProductGaming() { }
    1 usage
    public ProductGaming(string name)
    {
        _name = name;
    }
    1 usage
    public void GetDetailProductGaming()
    {
        Debug.Log("name: " + _name);
    }
}
```

PARTIAL METHOD

- Partial method cho phép code sinh tự động gọi phương thức nhưng không nhất thiết phải xây dựng (implement) phương thức đó.

```
public class Partial2 : MonoBehaviour
{
    ☢ Event function
    private void Start()
    {
        var product = new ProductGaming("FPOLY");
        product.GetDetailProductGaming();
    }
}
```

STATIC CLASS

FPT Education

FPT POLYTECHNIC

LẬP TRÌNH C#2

BÀI 1: STATIC CLASS, PARTIAL CLASS, GENERIC NAMESPACE (P2)

https://caodang.fpt.edu.vn

GENERIC NAMESPACE

HashSet<T> Class

- Là một tập hợp danh sách không cho phép trùng giá trị.
- Mỗi một phần tử trong HashSet là duy nhất
- Nếu bạn thêm 1 phần tử đã tồn thì nó sẽ không được thêm vào
- HashSet không truy cập phần tử thông qua index, tức là các phần tử trong set không có thứ tự (order). Do đó 2 set {1, 2, 3} và {3, 1, 2} là như nhau.
- HashSet được tối ưu cho việc tìm kiếm và thay thế dữ liệu nhanh chóng

GENERIC NAMESPACE

- HashSet<T> Class

```
//sử dụng hash set
var games = new HashSet<string>();
games.Add("Liên minh huyền thoại");
games.Add("Genshin impact");
games.Add("CSO");
games.Add("GTA 5");
games.Add("GTA 5"); // phần tử này sẽ không được thêm vào hashset vì nó đã tồn tại

foreach (var item in games)
{
    Debug.Log("Games item: " + item);
}
Debug.Log("tổng phần tử trong Games: " + games.Count);
```

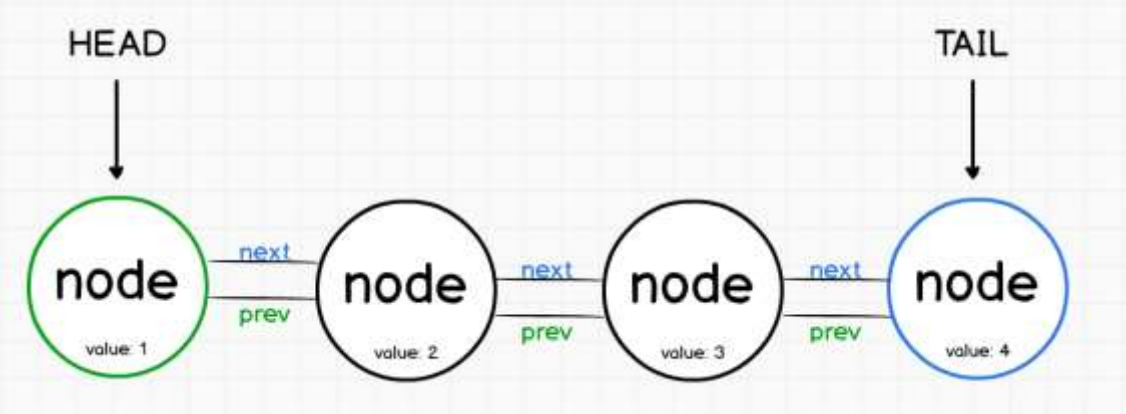
GENERIC NAMESPACE

LinkedList<T> Class

- Linked list là một tập hợp của các phần tử trong đó mỗi phần tử được liên kết (link) với phần tử trước (và sau nó)
- Các phần tử của linked list cũng được gọi là các node.
- Mỗi node bao gồm hai phần: phần dữ liệu, và phần tham chiếu
- Phần dữ liệu để lưu trữ dữ liệu (giống như phần tử của mảng).
- Phần tham chiếu chứa địa chỉ (ô nhớ) của node khác

❑ LinkedList<T> Class

Trong thư viện .NET cung cấp lớp LinkedList<T> là loại danh sách liên kết kép



Điều này có nghĩa, có có một nút - có thể lấy được nút phía trước nó - cứ thế cho đến nút đầu tiên, nút đầu, tương tự lấy được nút phía sau và dịch chuyển dần được về cuối.

GENERIC NAMESPACE

- LinkedList<T> Class

Một số Properties và hàm

Member	Ý nghĩa
Count	Số nút trong danh sách
First	Nút đầu tiên của danh sách
Last	Nút đầu tiên của danh sách
AddFirst(T)	Chèn một nút có dữ liệu T vào đầu danh sách
AddLast(T)	Chèn một nút có dữ liệu T vào cuối danh sách

GENERIC NAMESPACE

- LinkedList<T> Class

Một số Properties và hàm

Member	Ý nghĩa
AddAfter(Node, T)	Chèn nút với dữ liệu T, vào sau nút Node (kiểu LinkedListNode)
AddBefore(Node, T)	Chèn nút với dữ liệu T, vào trước nút Node (kiểu LinkedListNode)
Clear()	Xóa toàn bộ danh sách
Contains(T)	Kiểm tra xem có nút với giá trị dữ liệu bằng T

GENERIC NAMESPACE

- LinkedList<T> Class

Một số Properties và hàm

Member	Ý nghĩa
Remove(T)	Xóa nút có dữ liệu bằng T
RemoveFirst()	Xóa nút đầu tiên
RemoveLast()	Xóa nút cuối cùng
Find(T)	Tìm một nút

- LinkedList<T> Class

```
// sử dụng LinkedList
var linkedList = new LinkedList<string>();
linkedList.AddLast("Liên minh huyền thoại");
linkedList.AddLast("Genshin impact");
linkedList.AddFirst("CSO");
linkedList.AddAfter(linkedList.First, "GTA 5");

// xóa phần tử đầu tiên
linkedList.Remove(linkedList.First);

// xóa phần tử có tên là CSO
linkedList.Remove("CSO");

// xóa phần tử đầu tiên
linkedList.RemoveFirst();

// xóa toàn bộ phần tử
linkedList.Clear();
```

GENERIC NAMESPACE

- List<T> Class
  - Đại diện cho danh sách các đối tượng có thể được truy cập bởi chỉ mục
  - Có thể được sử dụng để tạo một tập các loại đối tượng có kiểu dữ liệu khác nhau: int, strings…

GENERIC NAMESPACE

- List<T> Class

Một số Properties và hàm

Member	Ý nghĩa
Add(object Value)	Thêm đối tượng Value vào cuối List.
AddRange(ICollection ListObject)	Thêm danh sách phần tử ListObject vào cuối List.
BinarySearch(object Value)	Tìm kiếm đối tượng Value trong List theo thuật toán tìm kiếm nhị phân.

GENERIC NAMESPACE

- List<T> Class

Một số Properties và hàm

Member	Ý nghĩa
Clear()	Xoá tất cả các phần tử trong Lis
Contains(T Value)	Kiểm tra đối tượng Value có tồn tại trong List hay không.
Insert(int Index, T Value)	Chèn đối tượng Value vào vị trí Index trong List.

GENERIC NAMESPACE

- List<T> Class

Một số Properties và hàm

Member	Ý nghĩa
Remove(T Value)	Xoá đối tượng Value xuất hiện đầu tiên trong List.
Reverse()	Đảo ngược tất cả phần tử trong List.
Sort()	Sắp xếp các phần tử trong List theo thứ tự tăng dần.

- List<T> Class

```
// sử dụng list
var listTxt = new List<string>();
listTxt.Add("Liên minh huyền thoại");
listTxt.Add("Genshin impact");
listTxt.Add("GTA 5");
// thêm phần tử vào vị trí 1
listTxt.Insert(1, "CSO");
// xóa phần tử GTA 5
listTxt.Remove("GTA 5");
// sắp xếp list
listTxt.Sort();
// kiểm tra phần tử có tồn tại trong list
// kết quả trả về true hoặc false
var check = listTxt.Contains("CSO");

// hiển thị list
foreach (var item in listTxt)
{
    Debug.Log("item: " + item);
}
```

GENERIC NAMESPACE

Tổng kết bài học

- 📖Phần I: Static class, Partial class
  - 📖Static class
  - 📖Partial class
- 📖Phần II: Generic Namespace
  - 📖HashSet<T> Class
  - 📖LinkedList<T> Class
  - 📖List<T> Class

Kết thúc