



# **LẬP TRÌNH C# 2**

## **BÀI 1: STATIC CLASS, PARTIAL CLASS - GENERIC NAMESPACE**

- ⊙ Static class, Partial class
- ⊙ Generic Namespace



## Phần I: Static class, Partial class

 Static class

 Partial class

## Phần II: Generic Namespace

 HashSet<T> Class

 LinkedList<T> Class

 List<T> Class



- ❑ Nhắc lại các thuộc tính, phương thức:
  - ❖ Chỉ có thể sử dụng sau khi khởi tạo đối tượng.
  - ❖ Dữ liệu thuộc về riêng mỗi đối tượng (xét cùng 1 thuộc tính thì các đối tượng khác nhau thì thuộc tính đó sẽ mang các giá trị khác nhau)
  - ❖ Được gọi thông qua tên của đối tượng.
- ❑ Mong muốn **1 thuộc tính nào đó được dùng chung cho mọi đối tượng** (chỉ được cấp phát 1 vùng nhớ duy nhất) → **thành viên tĩnh**

## ❑ Đặc điểm của thành viên tĩnh:

- ❖ Được khởi tạo 1 lần duy nhất ngay khi biên dịch chương trình.
- ❖ Có thể dùng chung cho mọi đối tượng.
- ❖ Được gọi thông qua tên lớp.
- ❖ Được huỷ khi kết thúc chương trình.

## ❑ 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).

❑ Biến tĩnh:      - Cú pháp khai báo:

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

❑ Một số đặc điểm của **biến tĩnh**:

- ❖ **Biến static** có thể được khởi tạo bên ngoài hàm thành viên, hoặc ngoài định nghĩa class, hoặc trong định nghĩa class.
- ❖ Là biến **dùng chung cho mọi đối tượng thuộc lớp**.
- ❖ Được khởi tạo vùng nhớ một lần duy nhất ngay khi chương trình được nạp vào bộ nhớ để thực thi. Và nó có vùng nhớ riêng, không bị thay đổi.
- ❖ Được **gọi trực tiếp thông qua tên lớp mà không cần tạo đối tượng của lớp**, chính vì điều này mà biến static thường được sử dụng để định nghĩa cho các hằng số (constant)..

## ❑ Biến tĩnh:

```
class People
{
    int id;
    string name;
    static string school = "FPoly";
    3 references
    public People(int id, string name)
    {
        this.id = id;
        this.name = name;
    }
    3 references
    public void Show()
    {
        Console.WriteLine("Id: " + id + ",\nName: " + name + ",\nschool: " + school + "\n\n");
    }
}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        People p1 = new People(1, "Nguyen Van A");
        p1.Show();
        People p2 = new People(2, "Nguyen Van B");
        p2.Show();
        People p3 = new People(3, "Nguyen Van C");
        p3.Show();
        Console.ReadKey();
    }
}
```

```
Id: 1,
Name: Nguyen Van A,
school: FPoly
```

```
Id: 2,
Name: Nguyen Van B,
school: FPoly
```

```
Id: 3,
Name: Nguyen Van C,
school: FPoly
```

❑ Phương thức tĩnh: - Cú pháp khai báo:

```
<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.



## ❑ Phương thức tĩnh:

```
public class Student
{
    // Khai báo phương thức tĩnh
    2 references
    public static void PrintName(string in_firstName, string in_lastName)
    {
        string name = string.Format("{0} {1}", in_firstName, in_lastName);
        Console.WriteLine("Student name: " + name);
    }
}
0 references
static void Main(string[] args)
{
    // Sử dụng trực tiếp tên class để truy cập vào static method.
    // mà không cần khởi tạo đối tượng student
    Student.PrintName("Fpoly", "Cntt");
    Student.PrintName("Welcome to", "Fpoly");
    Console.ReadKey();
}
```

## ❑ Phương thức khởi tạo tĩnh:

- Cú pháp khai báo:

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

## ❑ Một số đặc điểm của **phương thức khởi tạo tĩnh**:

- ❖ Không được phép khai báo phạm vi truy cập
- ❖ Static constructor sẽ được gọi một lần duy nhất khi chương trình được nạp lên.
- ❖ Trong phương thức có sử dụng biến static thì phương thức đó cũng phải được khai báo là static.

## ❑ Phương thức khởi tạo tĩnh:

```
class G1
{
    0 references
    static G1()
    {
        // The following statement produces, the first line of output,
        // and the line occurs only once.
        Console.WriteLine("Example of Static Constructor");
    }
    // Instance constructor.
    2 references
    public G1(int j)
    {
        Console.WriteLine("Instance Constructor " + j);
    }
    // Instance method.
    2 references
    public string g1_detail(string name, string branch)
    {
        return "Name: " + name + " Branch: " + branch;
    }
    0 references
    public static void Main()
    {
        // Here Both Static and instance constructors are invoked for first instance
        G1 obj = new G1(1);
        Console.WriteLine(obj.g1_detail("Sunil", "CSE"));
        // Here only instance constructor will be invoked
        G1 ob = new G1(2);
        Console.WriteLine(ob.g1_detail("Sweta", "ECE"));
    }
}
```

## ❑ Lớp tĩnh:

- ❖ static class chỉ chứa thành viên là static
- ❖ static class không có thể hiện.
- ❖ static class không chứa hàm xây dựng.
- ❖ 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 MyMath
{
    public static float PI = 3.14f;
    1 reference
    public static int squared(int n) { return n * n; }
}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        int squared;
        float pi = MyMath.PI;
        squared = MyMath.squared(10);
        Console.WriteLine("squared: " + squared);
        Console.WriteLine("\nPI: " + pi);
        Console.ReadKey();
    }
}
```

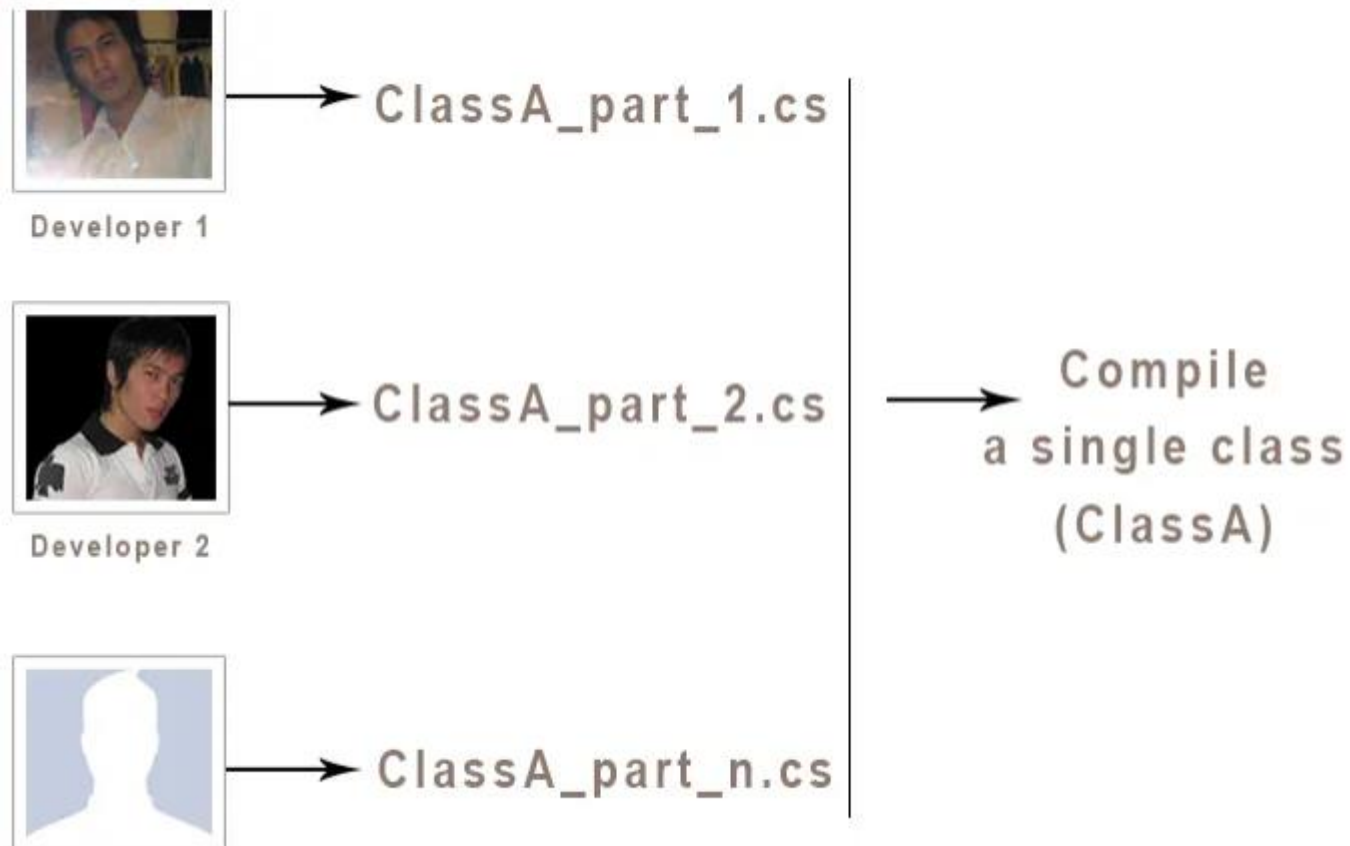


# DEMO

- Hiện thực các ví dụ



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



```
public partial class MyPartialClass
{
    partial void PartialMethod(int val);

    public MyPartialClass()
    {
    }

    public void Method2(int val)
    {
        Console.WriteLine(val);
    }
}
```

PartialClassFile1.cs

```
public partial class MyPartialClass
{
    public void Method1(int val)
    {
        Console.WriteLine(val);
    }

    partial void PartialMethod(int val)
    {
        Console.WriteLine(val);
    }
}
```

PartialClassFile2.cs

*Compiles as a single class*

```
public class MyPartialClass
{
    public MyPartialClass()
    {
    }

    public void Method1(int val)
    {
        Console.WriteLine(val);
    }

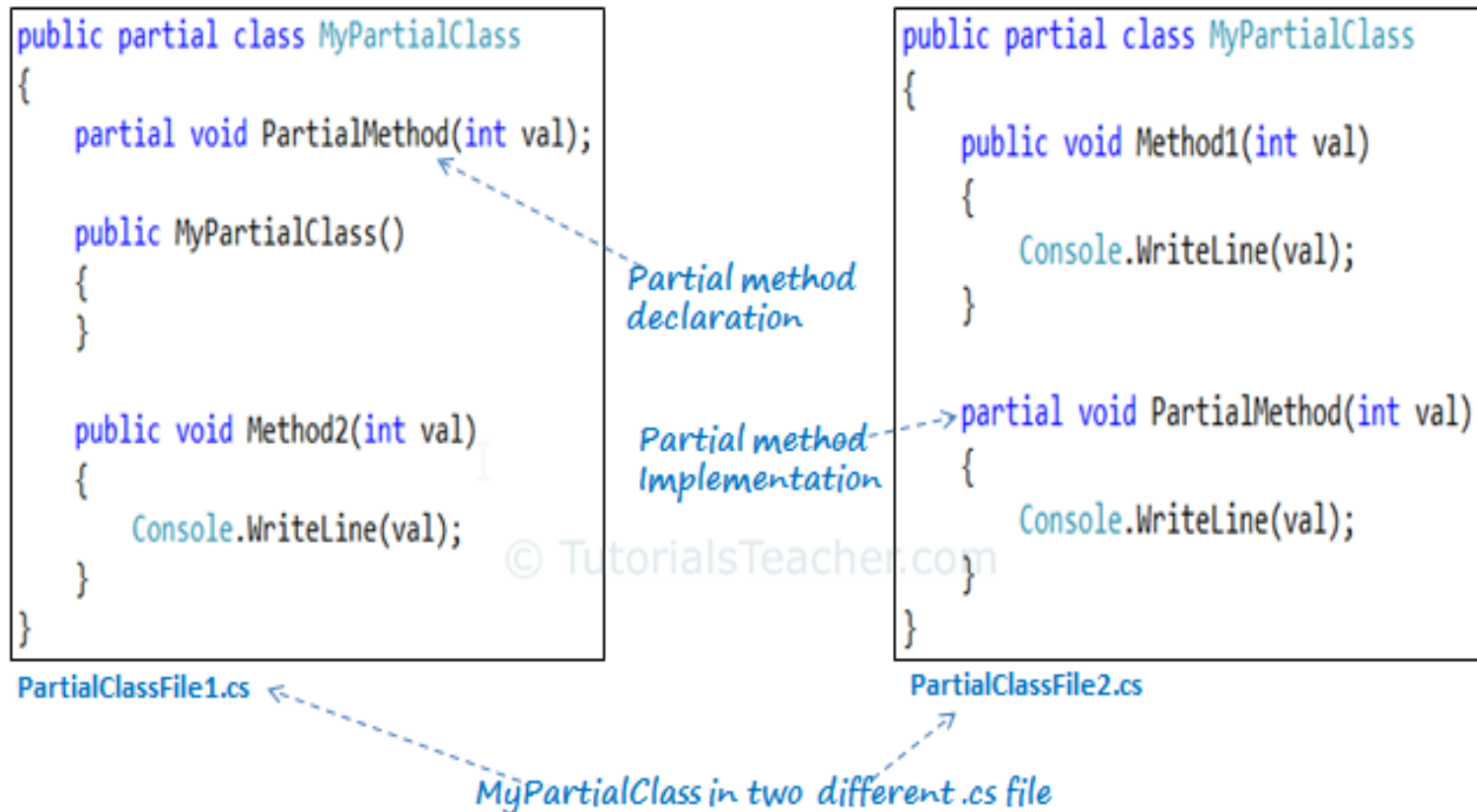
    public void Method2(int val)
    {
        Console.WriteLine(val);
    }

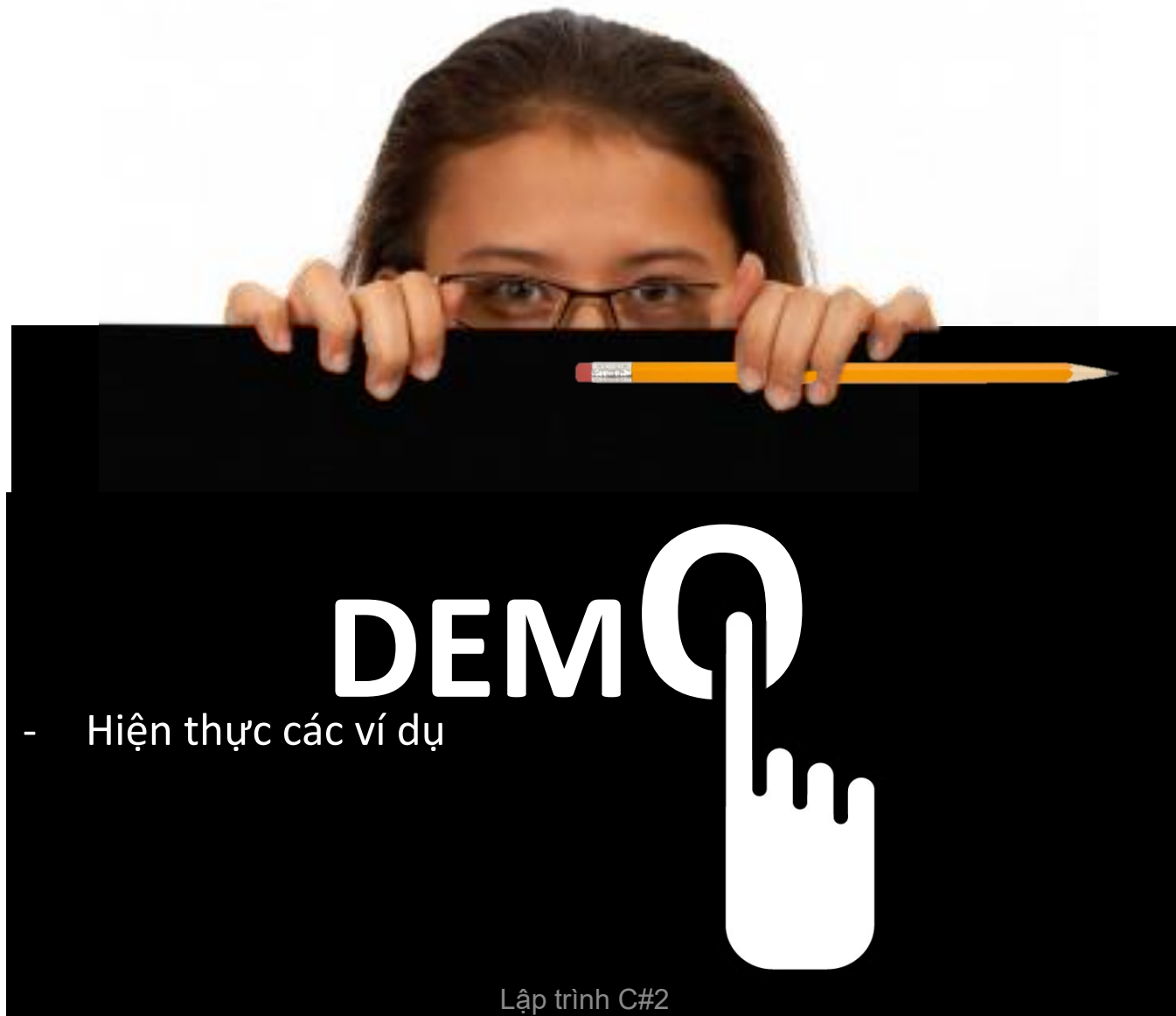
    private void PartialMethod(int val)
    {
        Console.WriteLine(val);
    }
}
```

Partial Class



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





**DEMO**

- Hiện thực các ví dụ

Lập trình C#2



# **LẬP TRÌNH C# 2**

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

## ❑ HashSet<T> Class

- ❖ Biểu diễn một tập hợp các phần tử không trùng nhau.
- ❖ 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.
- ❖ Lớp thuộc namespace **System.Collections.Generic**

## □ HashSet<T> Class

```
public static void Main(string[] args)
{
    // Create a set of strings
    var names = new HashSet<string>();
    names.Add("Sonoo");
    names.Add("Ankit");
    names.Add("Peter");
    names.Add("Irfan");
    names.Add("Ankit");//will not be added

    // Iterate HashSet elements using foreach loop
    foreach (var name in names)
    {
        Console.WriteLine(name);
    }
}
```

## ❑ HashSet<T> Class

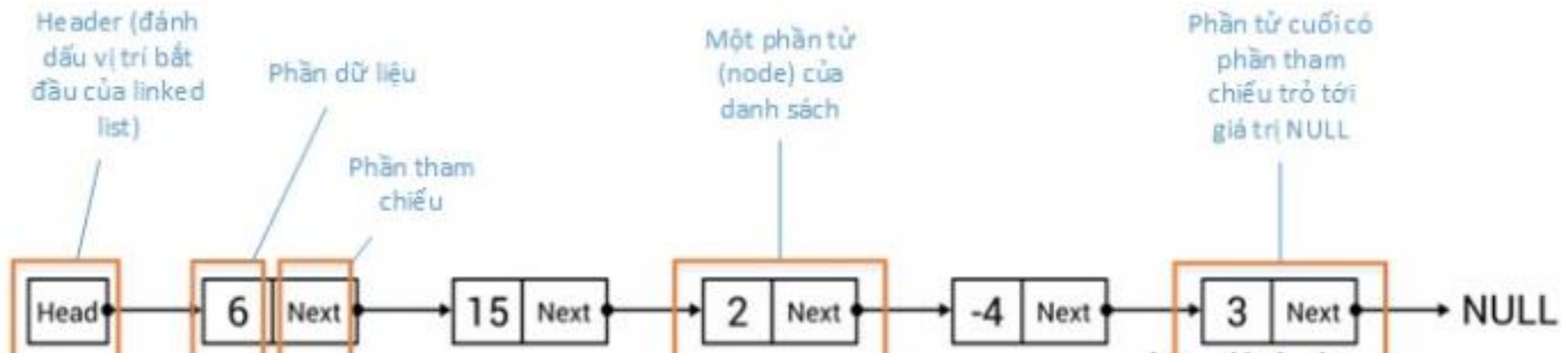
```
public static void Main(string[] args)
{
    // Create a set of strings
    var names = new HashSet<string>{"Sonoo", "Ankit", "Peter", "Irfan"};

    // Iterate HashSet elements using foreach loop
    foreach (var name in names)
    {
        Console.WriteLine(name);
    }
}
```

## ❑ 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





## ❑ LinkedList<T> Class

### ❖ Một số Properties

PROPERTY	DESCRIPTION
<b>Count</b>	Gets the number of nodes actually contained in the LinkedList.
<b>First</b>	Gets the first node of the LinkedList.
<b>Last</b>	Gets the last node of the LinkedList.

## ❑ LinkedList<T> Class

### ❖ Một số Methods

#### Methods

METHOD	DESCRIPTION
<b>AddAfter</b>	Adds a new node or value after an existing node in the LinkedList.
<b>AddBefore</b>	Adds a new node or value before an existing node in the LinkedList.
<b>AddFirst</b>	Adds a new node or value at the start of the LinkedList.
<b>AddLast</b>	Adds a new node or value at the end of the LinkedList.
<b>Clear()</b>	Removes all nodes from the LinkedList.

## ❑ LinkedList<T> Class

```
static void Main(string[] args)
{
    // Creating a LinkedList of Integers
    LinkedList<int> myList = new LinkedList<int>();
    // Adding nodes in LinkedList
    myList.AddLast(2);
    myList.AddLast(4);
    myList.AddLast(6);
    myList.AddLast(8);
    // To get the count of nodes in LinkedList
    // before removing all the nodes
    Console.WriteLine("Total nodes in myList are : " + myList.Count);
    // Displaying the nodes in LinkedList
    foreach (int i in myList)
    {
        Console.WriteLine(i);
    }
    // Removing the first node from the LinkedList
    myList.Remove(myList.First);
    // To get the count of nodes in LinkedList
    // after removing all the nodes
    Console.WriteLine("Total nodes in myList are : " + myList.Count);
    // Displaying the nodes in LinkedList
    foreach (int i in myList)
    {
        Console.WriteLine(i);
    }
}
```

## ❑ 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: integers, strings...
- ❖ Properties:

PROPERTY	DESCRIPTION
<b>Capacity</b>	Gets or sets the total number of elements the internal data structure can hold without resizing.
<b>Count</b>	Gets the number of elements contained in the List<T>.
<b>Item[Int32]</b>	Gets or sets the element at the specified index.

## □ List<T> Class

### ❖ Methods

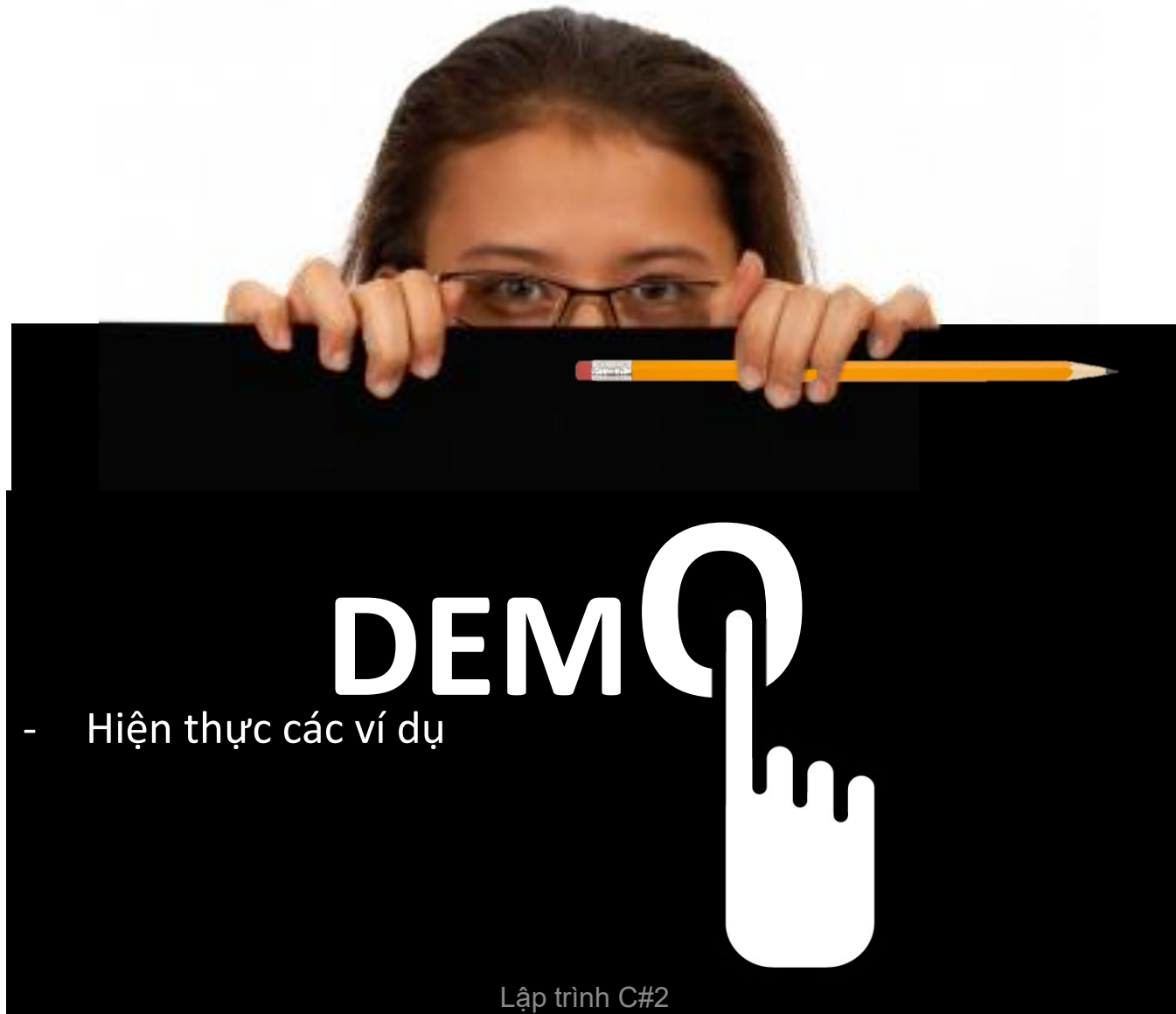
METHOD	DESCRIPTION
<b>Add(T)</b>	Adds an object to the end of the List<T>.
<b>AddRange(IEnumerable&lt;T&gt;)</b>	Adds the elements of the specified collection to the end of the List<T>.
<b>AsReadOnly()</b>	Returns a read-only ReadOnlyCollection<T> wrapper for the current collection.
<b>BinarySearch()</b>	Uses a binary search algorithm to locate a specific element in the sorted List<T> or a portion of it.
<b>Clear()</b>	Removes all elements from the List<T>.
<b>Contains(T)</b>	Determines whether an element is in the List<T>.
<b>ConvertAll(Converter)</b>	Converts the elements in the current List<T> to another type, and returns a list containing the converted elements.

## □ List<T> Class

```
static void Main(string[] args)
{
    // Creating an List<T> of Integers
    List<int> firstlist = new List<int>();

    // Adding elements to List
    firstlist.Add(1);
    firstlist.Add(2);
    firstlist.Add(3);
    firstlist.Add(4);
    firstlist.Add(5);
    firstlist.Add(6);
    firstlist.Add(7);

    // Checking whether 4 is present
    // in List or not
    Console.WriteLine(firstlist.Contains(4));
}
```



**DEMO**

- Hiện thực các ví dụ

Lập trình C#2

# 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**