# C# 9 – Thuộc tính và Chỉ mục

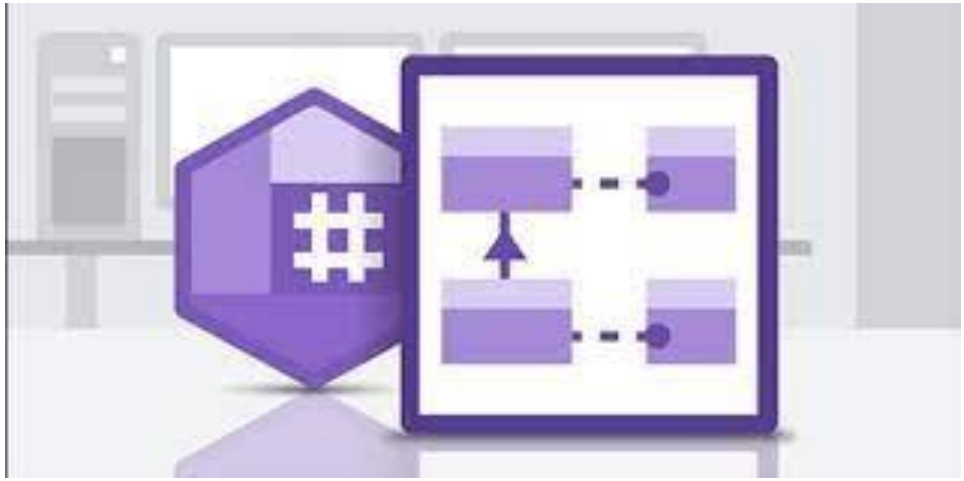

Giảng viên: **ThS. Lê Thiện Nhật Quang**

Email: quangltn.dotnet.vn@gmail.com

Website: http://dotnet.edu.vn

Điện thoại: **0868.917.786**





# MỤC TIÊU

- Tìm hiểu thuộc tính – Properties trong C#
- Giải thích thuộc tính, trường, phương thức
- Giải thích Chỉ mục – indexers

# 1.1. THUỘC TÍNH - PROPERTIES

- C# sử dụng một tính năng gọi là Property  để cho phép bạn có thể thiết lập và lấy các giá trị của các trường của lớp mà không quan tâm tới bổ từ truy cập của những trường đó là gì.
- Property cho phép bạn kiểm tra giá trị trước khi gán nó cho trường của lớp.
- Property cho phép bạn bảo vệ một trường của lớp bằng cách đọc hoặc ghi dữ liệu vào trường thông qua property đó.
- Property có thể kiểm tra dữ liệu trước khi cho phép bạn thay đổi chúng và thực hiện những công việc cần thiết thông qua những thay đổi đó.
- Property hỗ trợ đặc tính trừu tượng và đóng gói.

# 1.1. THUỘC TÍNH – PROPERTIES (2)

Cú pháp:

```
Bổ_từ_truy_cập Kiểu_trả_về Tên_property
{
   //Thân của property
}
```

```
public string Name
{
    set
    {
        ......
    }
    get
    {
        ......
    }
}
```

# 1.2. ACCESSOR TRONG C#

Accessor là một thuộc tính chứa các lệnh có thể thực thi, mà giúp đỡ trong việc *lấy* (đọc hoặc tính toán) hoặc *thiết lập* (ghi) thuộc tính. Các khai báo accessor có thể thu được một get accessor, một set accessor, hoặc cả hai. Ví dụ:

```
// khai báo một thuộc tính Code có kiểu dữ liệu string:
public string Code
{
   get
   {
      return code;
   }
   set
   {
      code = value;
   }
}
```

```
// khai báo một thuộc tính Name có kiểu dữ liệu String:
public string Name
{
   get
   {
      return name;
   }
   set
   {
      name = value;
   }
}
```

```
// khai báo một thuộc tính Age có kiểu dữ liệu int:
public int Age
{
    get
    {
        return age;
    }
    set
    {
        age = value;
    }
}
```

Ví dụ dưới đây minh họa cách sử dụng của các thuộc tính: tạo 2 lớp có tên lần lượt là Student, TestCsharp như sau:

```
class Student
{
    private string code = "N/A";
    private string name = "unknown";
    private int age = 0;

    // khai bao thuoc tinh Code co kieu string:
    public string Code
    {
        get
        {
            return code;
        }
        set
        {
            code = value;
        }
    }
public override string ToString()
{
    return "MSSV = " + Code + ", Ho Ten = " + Name + ", Tuoi = " + Age;
}
```

```
// khai bao thuoc tinh Name co kieu string:
public string Name
{
    get
    {
        return name;
    }
    set
    {
        name = value;
    }
}
// khai bao thuoc tinh Age co kieu int:
public int Age
{
    get
    {
        return age;
    }
    set
    {
        age = value;
    }
}
```

```
class TestCsharp
{
    static void Main(string[] args)
    {
        Console.WriteLine("Thuoc tinh (Property) trong C#");
        Console.WriteLine("-----------------------------------");

        // tao mot doi tuong Student
        Student s = new Student();

        // thiet lap cac thuoc tinh code, name va age cho Student
        s.Code = "001";
        s.Name = "Minh Chinh";
        s.Age = 21;
        Console.WriteLine("Thong tin sinh vien: {0}", s);

        //bay gio tang age them 1
        s.Age += 1;
        Console.WriteLine("Thong tin sinh vien: {0}", s);

        Console.ReadLine();
        Console.ReadKey();
    }
}
```

```
Thuoc tinh (Property) trong C#
-----------------------------------
Thong tin sinh vien: MSSV = 001, Ho Ten = Minh Chinh, Tuoi = 21
Thong tin sinh vien: MSSV = 001, Ho Ten = Minh Chinh, Tuoi = 22
```

# 2.1. PHÂN LOẠI PROPERTY – CHỈ ĐỌC

```
string name;
public string Name
{
    get
    {
        return name;
    }
}
```

# 2.1. PHÂN LOẠI PROPERTY – CHỈ GHI

```
string name;
public string Name
{
    set
    {
        name = value;
    }
}
```

# 2.1. PHÂN LOẠI PROPERTY – ĐỌC ĐƯỢC GHI ĐƯỢC

```
string name;
public string Name
{
    get
    {
        return name;
    }
    set
    {
        name = value;
    }
}
```

# 2.2. SO SÁNH PROPERTY VÀ FIELD

| Property | Field |
| --- | --- |
| Property là thành phần dữ liệu có thể gán và truy xuất giá trị. | Field là thành phần dữ liệu dùng để lưu trữ giá trị. |
| Property không phải là biến và như vậy nó không sử dụng được các từ khóa ref và out | Field có thể coi là biến của lớp nên nó có thể sử dụng được các từ khóa ref và out. |
| Property được định nghĩa như là một tập lệnh có thể thực thi. | Field được định nghĩa bằng một câu lệnh đơn. |
| Property được định nghĩa với các bộ truy cập get và set. | Field không được định nghĩa với các bộ truy cập. |
| Property có thể thực hiện các hành động tùy chỉnh trong quá trình thay đổi giá trị của field. | Field không có khả năng thực hiện các hành động tùy chỉnh. |

# 2.3. SO SÁNH PROPERTY VÀ METHOD

| Property | Method |
|---|---|
| Property đại diện cho các đặc điểm của một đối tượng. | Method đại diện cho các hành vi của một đối tượng. |
| Property chứa hai phương thức mà sẽ tự động được gọi mà không cần có tên. | Method được gọi bằng cách xác định tên các phương thức và phải thông qua đối tượng của lớp. |
| Property không thể có tham số. | Method có thể không có hoặc có nhiều tham số. |
| Property có thể được ghi đè nhưng không được tải chồng. | Method có thể được ghi đè và cũng có thể được tải chồng. |
| Ví dụ: tạo một Property tên Amount public int Amount { get{}; set{}; } | Ví dụ: Tạo một Method tên Amount có hai đối số public int Amount (int numOne,int numTwo) { } |

# 2.4. THUỘC TÍNH VIRTUAL VÀ ABSTRACT

C# cho phép bạn tạo một thuộc tính virtual hay abstract. Để khai báo một thuộc tính virtual, overriden hay abstract bạn chỉ cần thêm từ khoá đó trong lúc định nghĩa thuộc tính. Ví dụ để tạo một thuộc tính abstract thì cú pháp như sau:

```
public abstract string ForeName
{
   get;
   set;
}
```

# 3.1. INDEXER

Trong C#, indexer cho phép các thể hiện của lớp hay struct được đánh chỉ mục giống như mảng. Indexer có cú pháp khá tương tự với property, nhưng nó không phải là property, bộ truy cập của indexer cho phép có một hoặc nhiều tham số.

# 3.2. MỤC ĐÍCH CỦA INDEXER

Giả sử bạn là một giảng viên và bạn muốn thông qua hồ sơ của từng sinh viên để nắm được sự tiến bộ của các sinh viên của bạn. Có hai cách, cách thứ nhất là bạn sử dụng một số thao tác thích hợp để thiết lập (set) và lấy (get) một bản ghi cụ thể bạn muốn, điều này có thể làm bạn mất thời gian và đôi chút buồn tẻ; cách thứ hai là bạn tạo ra một indexer cho các mã sinh viên, điều này sẽ làm cho việc tiếp cận các hồ sơ dễ dàng hơn nhiều. Có được điều này là do indexer sử dụng vị trí chỉ mục của mã sinh viên để xác định vị trí của hồ sơ tương ứng.

# 3.3. ĐỊNH NGHĨA INDEXER

Indexer là thành phần dữ liệu cho phép bạn truy cập dữ liệu bên trong các đối tượng theo cách tương tự như truy cập mảng. Indexer cung cấp khả năng truy cập nhanh hơn tới dữ liệu trong đối tượng bởi vì nó thực hiện việc đánh chỉ mục cho những dữ liệu đó. Cách để indexer có thể truy cập được vào bên trong từng đối tượng đó chính là thông qua chỉ số của từng đối tượng đó.

Việc thực thi indexer cũng tương tự như thực thi property, chỉ khác ở một điểm đó là việc khai báo một indexer có thể có chứa tham số. Trong C#, indexer còn được biết đến như là mảng thông minh (**smart array**).

# 3.4. KHAI BÁO INDEXER

Có thể áp dụng indexer cho lớp, struct hoặc interface, và để khai báo một indexer bạn cần:

- Bổ từ truy cập, nó sẽ quyết định tầm vực của indexer.
- Kiểu trả về của indexer.
- Từ khoá this, dùng để tham chiếu tới đối tượng (thể hiện) hiện thời của lớp hiện thời.
- Cặp ngoặc vuông ([]), trong đó có kiểu dữ liệu và bộ định danh cho chỉ số.
- Cặp ngoặc xoắn ({}), nơi chứa các bộ truy cập set và get.

# 3.4. KHAI BÁO INDEXER (2)

Cú pháp:

```
Bổ_từ_truy_cập Kiểu_trả_về this[Tham_số]
{
    get => Tên_mảng[Tham_số];

    set => Tên_mảng[Tham_số] = Giá_trị;
}
```

# 3.4. KHAI BÁO INDEXER (3)

Ví dụ:

```
class Indexer
{
  public string[] empName = new string[2];
  public string this[int index]
  {
    get => empName[index];
    set => empName[index] = value;
  }
  static void Main(string[] args)
  {
    Indexer objEmp = new Indexer();
    objEmp[0] = "Nguyen Thi Phuong Thuy";
    objEmp[1] = "Dao Thi Soi";
    Console.WriteLine("Ten nhan vien:");
    for (int i = 0; i < 2; i++)
    {
      Console.Write(objEmp[i] + "\n");
    }
  }
}
```

# 3.4. THAM SỐ CỦA INDEXER

Indexer phải có ít nhất một tham số. Tham số dùng để biểu thị chỉ số của đối tượng, tham số sẽ lưu giá trị là vị trí chỉ số nơi đối tượng được gán hoặc được truy cập; điều này tương tự với việc gán hoặc truy cập giá trị trong mảng một chiều. Ngoài ra, nếu indexer có nhiều tham số thì việc truy cập cũng sẽ tương tự như đối với mảng nhiều chiều.

Về vấn đề truy cập indexer, indexer được truy cập thông qua thể hiện của lớp hiện thời kèm với chỉ số đằng sau thể hiện đó.

# 3.5. THỪA KẾ INDEXER

```
class Numbers
{
  private int[] num = new int[3];
  public int this[int index]
  {
    get => num[index];
    set => num[index] = value;
  }
}

class EvenNumbers : Numbers
{
  public static void Main()
  {
    EvenNumbers objEven = new EvenNumbers();
    objEven[0] = 0;
    objEven[1] = 2;
    objEven[2] = 4;
    for (int i = 0; i < 3; i++)
    {
      Console.WriteLine(objEven[i]);
    }
  }
}
```

- Indexer có thể được thừa kế giống như những thành phần khác của lớp, tức là những indexer của lớp cơ sở có thể được thừa kế bởi các lớp dẫn xuất. Ví dụ:
- Trong ví dụ trên, lớp Numbers tạo một indexer với một đối số kiểu int, lớp EvenNumbers thừa kế lớp Numbers, trong hàm Main() tạo một thể hiện của lớp dẫn xuất EvenNumbers, khi thể hiện này được gán các giá trị tại mỗi vị trí chỉ số thì bộ truy cập set của indexer trong lớp cơ sở Numbers được gọi với mỗi vị trí chỉ số. Vòng lặp for lặp 3 lần và hiển thị các giá trị được gán cho mỗi vị trí chỉ số sử dụng bộ truy cập get.
- Như vậy thông qua biện pháp thừa kế thì indexer trong lớp cơ sở đã được tái sự dụng trong lớp thừa kế.

# 3.6. ĐA HÌNH TRONG INDEXER

- Indexer còn có thể thực hiện tính đa hình bằng cách ghi đè indexer của lớp cơ sở hoặc bằng cách nạp chồng indexer.
- Bằng cách thực thi tính đa hình, lập trình viên cho phép các indexer của lớp dẫn xuất ghi đè indexer của lớp cơ sở. Ngoài ra, trong cùng một lớp ta có thể tạo nhiều indexer, miễn sao những indexer này phải có những dấu hiệu khác nhau; điều này có nghĩa là ta có thể nạp chồng indexer.
- Như vậy, tính đa hình cho phép indexer có thể hoạt động với các kiểu dữ liệu khác nhau của C# và tạo đầu ra (output) một cách tuỳ chỉnh.
- Ví dụ, đoạn mã dưới đây sẽ cho thấy tính đa hình của indexer với việc ghi đè indexer của lớp cơ sở:

```
class Result : Student
{
  string[] result = new string[2];
  public override string this[int index]
  {
    get => base[index];
    set =>base[index] = value;
  }
  static void Main(string[] args)
  {
    Result objResult = new Result();
    objResult[0] = "First";
    objResult[1] = "Pass";
    Student objStudent = new Student();
    objStudent[0] = "Luong";
    objStudent[1] = "Hoang";
    for (int i = 0; i < 2; i++)
    {
      Console.WriteLine(objStudent[i] + "\t\t" + objResult[i] + " class");
    }
  }
}
```

```
class Student
{
  string[] studName = new string[2];
  public virtual string this[int index]
  {
    get => studName[index];
    set => studName[index] = value;
  }
}
```

# 3.7. INDEXER NHIỀU THAM SỐ

- Indexer bắt buộc phải có tối thiểu một tham số. Ngoài ra, ta có thể tạo indexer với nhiều tham số, khi đó indexer sẽ được truy cập giống như mảng nhiều chiều; những indexer này có thể được dùng để tổ chức một tập các giá trị liên quan, chẳng hạn như nó có thể được dùng để lưu trữ cũng như thay đổi các giá trị đó.
- Dưới đây là ví dụ về indexer có hai tham số:

```
class Account
{
  string[,] accountDetails = new string[4, 2];
  public string this[int pos, int column]
  {
    get => accountDetails[pos, column];
    set => accountDetails[pos, column] = value;
  }
  static void Main(string[] args)
  {
    Account objAccount = new Account();
    string[] id = new string[3] { "1001", "1002", "1003" };
    string[] name = new string[3] { "Long", "Thanh", "Binh" };
    int counter = 0;
    for (int i = 0; i < 3; i++)
    {
      for (int j = 0; j < 1; j++)
      {
        objAccount[i, j] = id[counter];
        objAccount[i, j + 1] = name[counter++];
      }
    }
```

```
        Console.WriteLine("ID    Ten");
        Console.WriteLine();
        for (int i = 0; i < 4; i++)
        {
          for (int j = 0; j < 2; j++)
          {
            Console.Write(objAccount[i, j] + " ");
          }
          Console.WriteLine();
        }
    }
}
```

Trong ví dụ trên, lớp Account tạo một biến mảng accountDetails gồm 4 hàng và hai cột. Indexer sẽ nhập các giá trị các phần tử của mảng accountDetails. Indexer gồm hai tham số và chúng sẽ định nghĩa các vị trí của các giá trị mà sẽ được lưu trong mảng. Phương thức Main() tạo một thể hiện của lớp Account, thể hiện này được dùng để nhập các giá trị cho mảng accountDetails thông qua vòng lặp for, điều này sẽ gọi đến bộ truy cập set của indexer và set sẽ gán các giá trị cho các phần tử mảng. Vòng lặp for lồng phía dưới sẽ hiển thị ID được lưu trong mảng bằng cách gọi bộ truy cập get.

# 3.8. SỬ DỤNG INDEXER TRONG INTERFACE

Trong interface đương nhiên bạn cũng có quyền khai báo indexer, chỉ có lưu ý rằng bạn chỉ được khai báo indexer, trong đó các bộ truy cập của indexer cũng chỉ được là khai báo, không có phần định nghĩa. Mỗi indexer khai báo trong interface cũng mặc định có bổ từ truy cập là public và phải được thực thi  trong lớp thực thi interface đó. Điều này sẽ làm tăng khả năng tái sử dụng và cung cấp một cách linh hoạt các interface tùy chỉnh. Ví dụ:

```
class Students : IDetails
{
  string[] studentName = new string[3];
  int[] studentID = new int[3];
  public string this[int index]
  {
    get => studentName[index];
    set => studentName[index] = value;
  }
  static void Main(string[] args)
  {
    Students objStudent = new Students();
    objStudent[0] = "Phuong";
    objStudent[1] = "Hung";
    objStudent[2] = "Trinh";
    Console.WriteLine("Ten sinh vien:");
    for (int i = 0; i < 3; i++)
    {
      Console.WriteLine(objStudent[i]);
    }
  }
}
```

Trong ví dụ trên, interface IDetails khai báo một indexer read-write. Lớp Student thực thi interface IDetails và thực thi indexer được định nghĩa trong interface. Phương thức Main() tạo một thể hiện của lớp Students và thông qua bộ truy cập set của indexer nó gán các giá trị cho mảng studentName. Thông qua bộ truy cập get của indexer, vòng lặp sẽ in ra màn hình giá trị của các phần tử mảng studentName.

```
public interface IDetails
{
  string this[int index]
  {
    get;
    set;
  }
}
```

# 3.9. SỰ KHÁC NHAU GIỮA PROPERTY VÀ INDEXER

- Property được gán một tên duy nhất trong quá trình khai báo, còn Indexer không thể có tên, từ khóa this sẽ thay thế khi khai báo.
- Property được gọi thông qua tên của nó, còn Indexer được gọi thông qua chỉ số của thể hiện được tạo.
- Property có thể được khai báo là tĩnh (dùng từ khóa static), Indexer không thể được khai báo là tĩnh.
- Property luôn luôn được khai báo mà không có tham số, Indexer phải được khai báo với ít nhất một tham số.
- Property không thể được nạp chồng, nhưng Indexer có thể được nạp chồng.
- Việc ghi đè Property được thực hiện bằng cách sử dụng cú pháp: base.Tên_property. Còn việc ghi đè Indexer được thực hiện bằng cách sử dụng cú pháp: base[Danh_sách_tham_số]