Interface trong C#, loosely coupling

Interface trong C# là một bản "hợp đồng" mô tả những gì cần phải làm mà các class thực thi interface đó phải tuân thủ theo. Interface trong C# là một công cụ đặc biệt mạnh giúp tạo ra mối quan hệ lỏng giữa các class, qua đó giúp phát triển và test các thành phần (class) một cách độc lập.

Tuy vậy, đây là một kiểu dữ liệu rất khó hiểu với nhiều bạn. Đặc biệt, rất nhiều bạn không biết cách vận dụng interface trong lập trình.

Bài học này sẽ cố gắng giúp bạn hiểu interface là gì, vai trò của interface trong C#, cũng như các kỹ thuật làm việc với interface trong C#.

NỘI DUNG CỦA BÀI [Ẩn]

1. Interface trong C# là gì?
   1.1. Quan hệ phụ thuộc giữa các class
   1.2. Khái niệm interface trong C#
   1.3. Ví dụ minh họa
2. Kỹ thuật lập trình với Interface trong C#
   2.1. Khai báo kiểu interface
   2.2. Thực thi interface
3. Sử dụng interface trong C#
   3.1. Kiểu interface
   3.2. Khởi tạo object
4. Kết luận

Interface trong C# là gì?

Quan hệ phụ thuộc giữa các class

Class B được coi là phụ thuộc chặt vào class A nếu class A được sử dụng trong code của B (như tham chiếu tới A, nhận tham số kiểu A, khởi tạo object của A, khai báo biến của A, v.v.).

Quan hệ phụ thuộc chặt này đơn giản khi sử dụng nhưng có thể gây ra nhiều hậu quả. Quan hệ phụ thuộc chặt yêu cầu các lớp phụ thuộc phải xây dựng sau, dẫn tới không thể phát triển song song các class. Quan hệ chặt cũng có thể gây khó khăn cho việc test các class độc lập (vì chúng phụ thuộc vào nhau).

Để có thể phát triển song song hoặc dễ dàng thay thế class này bằng class khác, người ta cần làm giảm sự phụ thuộc giữa các class, thay phụ thuộc chặt bằng *phụ thuộc lỏng* (loosely-coupling).

Một công cụ rất mạnh thường được sử dụng để làm giảm sự phụ thuộc này là *giao diện* (interface).

Khái niệm interface trong C#

Interface là một kiểu dữ liệu tương tự như class nhưng chỉ đưa ra mô tả (specification / declaration) của các thành viên mà không đưa ra phần thực thi (phần thân, body / implementation). Phần thân của các phương thức sẽ phải được xây dựng trong các class thực thi giao diện này.

Một cách gần đúng, interface gần giống như một abstract class trong đó tất cả các phương thức của class đều được đánh dấu là abstract.

Cũng giống như abstract class, interface không thể dùng để khởi tạo object mà chỉ để các lớp cụ thể "kế thừa". Khi một class "kế thừa" từ một interface, nó bắt buộc phải cung cấp phần thực thi cho tất cả các thành viên của interface (tương tự như phải thực thi tất cả các thành viên abstract).

Interface tạo ra một bản "hợp đồng" mô tả những gì cần phải làm mà các class thực thi interface đó phải tuân thủ theo. Khi đó, các class phối hợp với nhau thông qua bản hợp đồng này mà không cần biết đến nhau nữa (làm mất quan hệ chặt).

Vì đặc điểm đó, interface trở thành một công cụ đặc biệt mạnh giúp tạo ra mối quan hệ lỏng giữa các class, qua đó giúp phát triển và test các thành phần (class) một cách độc lập.

Khi sử dụng interface vẫn phải thực hiện khởi tạo object của một class cụ thể thực thi interface này. Thao tác khởi tạo này thực hiện ở một class trung gian.

Ví dụ minh họa

Hãy cùng thực hiện ví dụ sau để hiểu kỹ hơn về cách sử dụng interface

```
1.  using System;
2.  namespace ConsoleApp
3.  {
4.      internal interface IPet // khai báo interface với hai phương thức
5.      {
6.          void Feed(); // mô tả phương thức (không có thân)
7.          void Sound();
8.      }
9.      internal interface IBird // khai báo interface với ba phương thức
10.     {
11.         void Fly();
12.         void Sound();
13.         void Feed();
14.     }
15.     internal class Cat : IPet // Cat thực thi IPet
16.     {
17.         public Cat() => Console.WriteLine("I'm a cat. ");
18.         // thực thi cho phương thức Feed và Sound
19.         // hai phương thức này thực thi theo kiểu implicit
20.         public void Feed() => Console.WriteLine("Fish, please!");
21.         public void Sound() => Console.WriteLine("Meow meow!");
22.     }
23.     internal class Dog : IPet // Dog thực thi IPet
24.     {
25.         public Dog() => Console.WriteLine("I'm a dog. ");
26.         // cả hai phương thức Feed và Sound thực thi kiểu explicit.
27.         // Object của Dog không thể gọi hai phương thức này.
28.         // Hai phương thức này chỉ có thể gọi qua giao diện IPet
29.         void IPet.Feed() => Console.WriteLine("Bone, please!");
30.         void IPet.Sound() => Console.WriteLine("Woof woof!");
31.     }
32.     internal class Parrot : IPet, IBird // Parrot thực thi cả hai giao diện
33.     {
```

```
        public Parrot() => Console.WriteLine("I'm a parrot. ");
        // hai phương thức này thực thi kiểu implicit, do đó
        // có thể gọi từ object của Parrot
        public void Feed() => Console.WriteLine("Nut, please!");
        public void Fly() => Console.WriteLine("Yeah, I can fly!");
        // hai phương thức này thực thi kiểu explicit, do đó
        // không thể gọi từ object của Parrot
        // mà chỉ có thể gọi qua giao diện IPet hoặc IBird
        void IPet.Sound() => Console.WriteLine("I can speak!");
        void IBird.Sound() => Console.WriteLine("I can sing, too!");
    }
    internal class BirdLover
    {
        private IBird _bird;
        public BirdLover(IBird bird) => _bird = bird;
        public void Play()
        {
            // _bird có thể gọi đủ các phương thức của IBird
            Console.Write("Fly ...");
            _bird.Fly();
            Console.Write("Say something ...");
            _bird.Sound();
            Console.Write("What do you like to eat? ");
            _bird.Feed();
        }
    }
    internal class PetLover
    {
        private IPet _pet;
        public PetLover(IPet pet) => _pet = pet;
        public PetLover() { }
        public void Play()
        {
            // _pet có thể gọi đủ các phương thức của IPet
            Console.Write("What do you like to eat? ");
            _pet.Feed();
            Console.Write("Now say something ... ");
            _pet.Sound();
        }
    }
    internal class _18_interface
    {
        private static void Main()
        {
            IPet pet = new Dog();
            PetLover petLover = new PetLover(pet);
            petLover.Play();
            petLover = new PetLover(new Parrot());
            petLover.Play();
            BirdLover birdLover = new BirdLover(new Parrot());
            birdLover.Play();
            Cat cat = new Cat();
            // cat có thể gọi được Feed và Sound
            cat.Feed(); cat.Sound();
            IPet cat2 = new Cat();
            // cat2 có thể gọi Feed và Sound
            cat2.Feed(); cat2.Sound();
            Parrot parrot = new Parrot();
            // (gọi qua object) parrot chỉ gọi được Feed và Fly, không gọi được Sound
            parrot.Feed(); parrot.Fly();
            IBird parrot2 = new Parrot();
            // (gọi qua giao diện) parrot2 gọi được đủ 3 phương thức của IBird
            parrot2.Feed(); parrot2.Fly(); parrot2.Sound();
            // dog không gọi được phương thức nào (gọi qua object) do
            // cả hai phương thức của Dog đều thực hiện kiểu explicit
            Dog dog = new Dog();
            IPet dog2 = new Dog();
            // gọi qua giao diện: dog2 gọi được cả Feed và Sound
            dog2.Feed(); dog2.Sound();
            Console.ReadKey();
        }
    }
}
```

Kỹ thuật lập trình với Interface trong C#

Khai báo kiểu interface

Trong ví dụ trên chúng ta xây dựng hai interface: IPet và IBird

- IPet
- IBird

```
1.  internal interface IPet // khai báo interface với hai phương thức
2.  {
3.      void Feed(); // mô tả phương thức (không có thân)
4.      void Sound();
5.  }
```

Interface được khai báo với từ khóa *interface* và danh sách mô tả các phương thức, đặc tính hoặc biến thành viên.

Một interface có thể được sử dụng nội bộ trong project, hoặc được sử dụng bởi các project khác. Trong tình huống thứ nhất (mặc định), interface sử dụng từ khóa điều khiển truy cập internal (tương tự class), và do đó có thể không cần viết từ khóa internal. Trong tình huống thứ hai sử dụng từ khóa public.

Interface là một kiểu dữ liệu cùng cấp độ với class, do đó có thể được khai báo trực tiếp trong không gian tên hoặc trong phạm vi của class khác. Tên của interface được đặt giống quy ước tên class nhưng có thêm chữ "I" đứng trước. Như ví dụ trên, tên hai interface lần lượt là IPet, IBird.

Trong interface chỉ có các mô tả, không có thân phương thức. Mô tả phương thức không có từ khóa điều khiển truy cập (tức là không có public, private, protected trước các mô tả).

Thực thi interface

Mặc dù interface là một kiểu dữ liệu nhưng tự bản thân nó không có khả năng sinh ra object mà chỉ có thể tạo ra biến tham chiếu đến object của các class khác tuân thủ theo quy định của interface.

Interface được sử dụng làm khuôn mẫu để sinh ra các class khác (gần giống như lớp abstract). Việc tạo ra một class trên cơ sở khuôn mẫu của interface gọi là *thực thi interface*.

Cấu trúc cú pháp để một class thực thi một interface như sau:

```
internal class Cat : IPet // Cat thực thi IPet
internal class Dog : IPet // Dog thực thi IPet
```

Một class cũng có thể thực thi nhiều interface:

```
internal class Parrot : IPet, IBird // Parrot thực thi cả hai giao diện
```

Khi một class thực thi một hoặc nhiều interface, nó có nghĩa vụ phải xây dựng tất cả các thành viên được mô tả trong interface. Visual Studio hỗ trợ bằng cách đánh dấu lỗi cú pháp (gạch chân đỏ) nếu class chưa xây dựng đủ các thành viên của interface theo yêu cầu.

Có hai cách thức thực thi các thành viên của interface: implicit và explicit.

Trong cách thực thi implicit không chỉ rõ là phương thức được thực thi thuộc về interface nào; ngược lại, cách thực thi explicit phải chỉ rõ phương thức đang thực thi thuộc về interface nào.

Lớp Cat ở đây hoàn toàn áp dụng cách thực thư implicit.

```
internal class Cat : IPet // Cat thực thi IPet
{
    public Cat() => Console.WriteLine("I'm a cat. ");
    // thực thi cho phương thức Feed và Sound
    // hai phương thức này thực thi theo kiểu implicit
    public void Feed() => Console.WriteLine("Fish, please!");
    public void Sound() => Console.WriteLine("Meow meow!");
}
```

Lớp Dog lại hoàn toàn thực thi kiểu explicit. Mỗi phương thức khi thực thi phải chỉ rõ nó thuộc interface nào.

```
internal class Dog : IPet // Dog thực thi IPet
{
    public Dog() => Console.WriteLine("I'm a dog. ");
    // cả hai phương thức Feed và Sound thực thi kiểu explicit.
    // Object của Dog không thể gọi hai phương thức này.
    // Hai phương thức này chỉ có thể gọi qua giao diện IPet
    void IPet.Feed() => Console.WriteLine("Bone, please!");
    void IPet.Sound() => Console.WriteLine("Woof woof!");
}
```


```
internal class Dog : IPet // Dog thực thi IPet
{
    public Dog() => Console.WriteLine("I'm a dog. ");
    // cả hai phương thức Feed và Sound thực thi kiểu explicit.
    // Object của Dog không thể gọi hai phương thức này.
    // Hai phương thức này chỉ có thể gọi qua giao diện IPet
    void IPet.Feed() => Console.WriteLine("Bone, please!");
    void IPet.Sound() => Console.WriteLine("Woof woof!");
}
```


Lớp Parrot áp dụng cả implicit và explicit

```
internal class Parrot : IPet, IBird // Parrot thực thi cả hai giao diện
{
    public Parrot() => Console.WriteLine("I'm a parrot. ");
    // hai phương thức này thực thi kiểu implicit, do đó
    // có thể gọi từ object của Parrot
    public void Feed() => Console.WriteLine("Nut, please!");
    public void Fly() => Console.WriteLine("Yeah, I can fly!");

    // hai phương thức này thực thi kiểu explicit, do đó
    // không thể gọi từ object của Parrot
    // mà chỉ có thể gọi qua giao diện IPet hoặc IBird
    void IPet.Sound() => Console.WriteLine("I can speak!");
    void IBird.Sound() => Console.WriteLine("I can sing, too!");
}
```

Nếu phương thức được thực thi theo kiểu explicit thì không được phép sử dụng từ khóa điều khiển truy cập.

Sự khác biệt lớn nhất giữa implicit và explicit thể hiện ở việc sử dụng object của class.

Sử dụng interface trong C#

Kiểu interface

Interface có thể sử dụng như một kiểu dữ liệu để khai báo biến. Biến của interface cho phép gọi các thành viên của interface giống như một object bình thường của class.

- BirdLover
- PetLover

```
internal class BirdLover
{
    private IBird _bird;
    public BirdLover(IBird bird) => _bird = bird;
    public void Play()
    {
        // _bird có thể gọi đủ các phương thức của IBird
        Console.Write("Fly ...");
        _bird.Fly();
        Console.Write("Say something ...");
        _bird.Sound();
        Console.Write("What do you like to eat? ");
        _bird.Feed();
    }
}
```

Trong hai class BirdLover và PetLover chúng ta sử dụng hai biến _bird và _pet giống như một object bình thường.

Tuy nhiên, biến của interface bắt buộc phải tham chiếu tới một object thực sự. Như trong hai lớp trên, object của class được truyền qua tham số của hàm tạo. Nếu không cho biến của interface tham chiếu tới một object thực sự, khi chạy chương trình sẽ gặp lỗi 'Object reference not set to an instance of an object' ở các lời gọi hàm hoặc truy xuất thành viên.

Ví dụ, lệnh sau sẽ báo lỗi khi chạy:

```
PetLover petLover2 = new PetLover();
petLover2.Play();
```

Ở đây chúng ta sử dụng constructor không tham số của lớp PetLover (nghĩa là không truyền object nào để gán cho biến _pet. Chương trình sẽ báo lỗi ở lời gọi _pet.Feed() vì _pet không hề tham chiếu tới một object nào.

Khởi tạo object

Interface có thể dùng để khai báo biến (như ở trên) nhưng không thể tự khởi tạo object. Biến kiểu interface chỉ có thể tham chiếu tới object của class thực thi interface đó.

```
IPet pet = new Dog();
PetLover petLover = new PetLover(pet);
petLover.Play();

petLover = new PetLover(new Parrot());
petLover.Play();

BirdLover birdLover = new BirdLover(new Parrot());
birdLover.Play();
```

Nói một cách khác, chúng ta cần sử dụng một class cụ thể thực thi interface để khởi tạo object rồi gán object đó cho biến interface.

Đối với các class thực thi interface phụ thuộc vào cách thực thi (explicit hay implicit), có sự khác biệt khi sử dụng object của các class này:

```
Cat cat = new Cat();
// cat có thể gọi được Feed và Sound
cat.Feed(); cat.Sound();

IPet cat2 = new Cat();
// cat2 có thể gọi Feed và Sound
cat2.Feed(); cat2.Sound();

Parrot parrot = new Parrot();
// (gọi qua object) parrot chỉ gọi được Feed và Fly, không gọi được Sound
parrot.Feed(); parrot.Fly();

IBird parrot2 = new Parrot();
// (gọi qua giao diện) parrot2 gọi được đủ 3 phương thức của IBird
parrot2.Feed(); parrot2.Fly(); parrot2.Sound();

// dog không gọi được phương thức nào (gọi qua object) do
// cả hai phương thức của Dog đều thực hiện kiểu explicit
Dog dog = new Dog();

IPet dog2 = new Dog();
// gọi qua giao diện: dog2 gọi được cả Feed và Sound
dog2.Feed(); dog2.Sound();
```

Việc thực thi implicit tạo cho object của class khả năng sử dụng các phương thức như class bình thường:

```
Cat cat = new Cat();
// cat có thể gọi được Feed và Sound
cat.Feed(); cat.Sound();
```

Những phương thức nào được thực thi kiểu explicit thì không thể gọi được trên object:

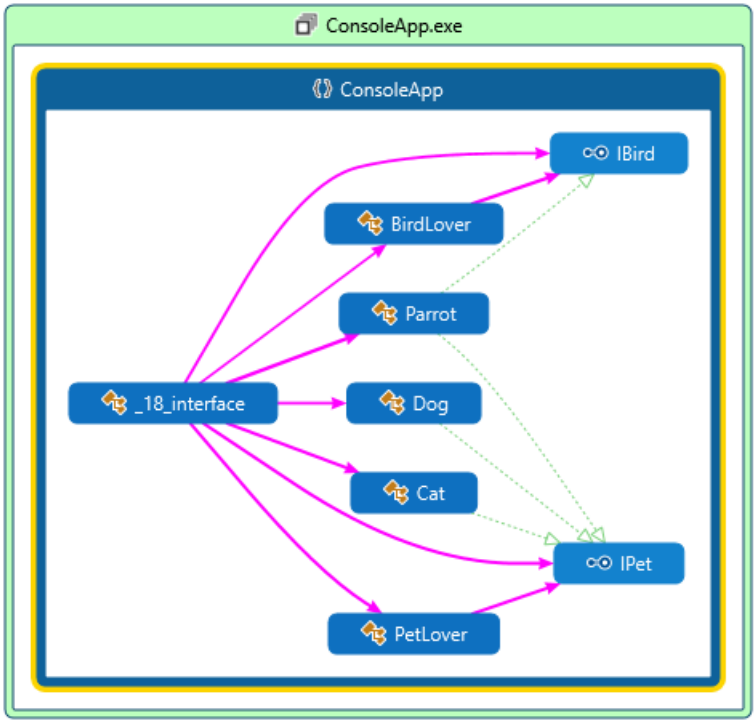
```
Parrot parrot = new Parrot();
// (gọi qua object) parrot chỉ gọi được Feed và Fly, không gọi được Sound
parrot.Feed(); parrot.Fly();
// dog không gọi được phương thức nào (gọi qua object) do
// cả hai phương thức của Dog đều thực hiện kiểu explicit
Dog dog = new Dog();
```

Vì có sự khác biệt giữa hai cách sử dụng object (parrot và parrot2)

```
Parrot parrot = new Parrot(); parrot.Feed(); parrot.Fly()
IBird parrot2 = new Parrot(); parrot2.Feed(); parrot2.Fly(); parrot2.Sound();
```

Chúng ta gọi cách sử dụng thứ nhất là “gọi qua object”, cách sử dụng thứ hai gọi là “gọi qua interface”.

Trong ví dụ trên, sự phụ thuộc giữa các class được thể hiện qua sơ đồ code sau.



Sơ đồ code của ví dụ

Kết luận

Trong bài này chúng ta đã xem xét một kỹ thuật đặc biệt quan trọng trong phát triển ứng dụng: sử dụng interface. Chúng ta cũng thấy, interface cho phép giảm bớt sự phụ thuộc giữa các class và tạo ra quan hệ lỏng giữa chúng. Quan hệ này cho phép các class có thể được xây dựng và test độc lập.

> + Nếu bạn thấy site hữu ích, trước khi rời đi hãy **giúp đỡ** site bằng một hành động nhỏ để site có thể phát triển và phục vụ bạn tốt hơn.
> + Nếu bạn thấy bài viết hữu ích, hãy giúp **chia sẻ** tới mọi người.
> + Nếu có thắc mắc hoặc cần trao đổi thêm, mời bạn viết trong phần **thảo luận** cuối trang.
> Cảm ơn bạn!