Interface là gì? Tại sao lại sử dụng interface

Interface (nhiều tài liệu gọi là giao diện hoặc lớp giao tiếp) là 1 tập các thành phần chỉ có khai báo mà không có phần định nghĩa.

Một interface được hiểu như là 1 khuôn mẫu mà mọi lớp thực thi nó đều phải tuân theo. Interface sẽ định nghĩa phần "**làm gì**" (khai báo) và những lớp thực thi interface này sẽ định nghĩa phần "**làm như thế nào**" (định nghĩa nội dung) tương ứng.

Đặc điểm của interface

- Chỉ chứa khai báo không chứa phần định nghĩa (giống phương thức trừu tượng). Mặc dù giống phương thức trừu tượng nhưng bạn không cần phải khai báo từ khoá abstract.
- Việc ghi đè 1 thành phần trong interface cũng không cần từ khoá override.
- Không thể khai báo phạm vi truy cập cho các thành phần bên trong interface. Các thành phần này sẽ mặc định là public.
- Interface không chứa các thuộc tính (các biến) dù là hằng số hay biến tĩnh vẫn không được.
- Interface không có constructor cũng không có destructor.

- Các lớp có thể thực thi nhiều interface cùng lúc (ở 1 góc độ nào đó có thể nó là phương án thay thế đa kế thừa).
- Một interface có thể kế thừa nhiều interface khác nhưng không thể kế thừa bất kỳ lớp nào.

Mục đích sử dụng interface

- Vì C# không hỗ trợ đa kế thừa nên interface ra đời như là 1 giải pháp cho việc đa kế thừa này. một lớp chỉ có thể kế thừa từ một lớp cơ sở). Tuy nhiên, nó có thể đạt được với nhiều interface.
- Trong 1 hệ thống việc trao đổi thông tin giữa các thành phần cần được đồng bộ và có những thống nhất chung. Vì thế dùng interface sẽ giúp đưa ra những quy tắc chung mà bắt buộc các thành phần trong hệ thống này phải làm theo mới có thể trao đổi với nhau được.

Khai báo và sử dụng interface

Cú pháp:

```
interface <tên interface>
  {
     // Khai báo các thành phần bên trong interface
  }
```

Trong đó:

- **Interface** là từ khoá dùng để khai báo 1 interface.
- **<tên interface>** là tên do người dùng đặt và tuân theo các quy tắc đặt tên trong lập trình.(để tránh nhầm lẫn với lớp kế thừa thì khi đặt tên interface người ta thường thêm tiền tố "**I**" để nhận dạng).

Việc thực thi 1 interface hoàn toàn giống kế thừa từ 1 lớp (đã trình bày trong bài lớp kế thừa)

Ví dụ:

```
interface ISpeak
    {
        /*
            Khai báo phương thức nhưng không định nghĩa nội
dung
         */
        void Speak();
    }

    class Animal : ISpeak // lớp Animal thực thi interface
ISpeak
    {
        /*
            Định nghĩa nội dung cho phương thức trong
interface
      Phương thức Speak() phải có phạm vi là public vì phương
thức Speak() trong interface mặc định là public rồi.
         */
        public void Speak()
        {
            Console.WriteLine("Animal is speaking. . .");
        }
    }
```

Trong hàm main ta thử phương thức **Speak()** xem có chạy được không:

```
Animal animal = new Animal();

animal.Speak();
```

Kết quả khi chạy chương trình: `Animal is speaking. . .`

Vì việc thực thi interface rất giống với kế thừa nên ta hoàn toàn có thể sử dụng câu lệnh sau:

```
ISpeak animal = new Animal();
1
```

Khi đó chạy lại chương trình vẫn ra kết quả như ban đầu. (`Animal is speaking. . .`)

Việc thiết kế, sử dụng interface và abstract class chính là cách thể hiện tính trừu tượng trong lập trình hướng đối tượng.

Lưu ý: các lớp thừa kế interface phải định nghĩa nội dung cho tất cả thành phần trong interface.

So sánh giữa interface và lớp trừu tượng

Những điểm giống nhau giữa interface và abstract class:

- Đều có thể chứa phương thức trừu tượng.
- Đều không thể khởi tạo đối tượng.

Những điểm khác nhau:

Interface	Abstract class
Chỉ có thể kế thừa nhiều interface khác.	Có thể kế thừa từ 1 lớp và nhiều interface.
Chỉ chứa các khai báo và không có phần nội dung. Không thể chứa biến.	Có thể chứa các thuộc tính (biến) và các phương thức bình thường bên trong.
Không cần dùng từ khoá override để ghi đè.	Sử dụng từ khoá override để ghi đè.
Không có constructor và cũng không có destructor.	Có constructor và destructor.
Không có phạm vi truy cập. Mặc định luôn là public.	Có thể khai báo phạm vi truy cập.
Dùng để định nghĩa 1 khuôn mẫu hoặc quy tắc chung.	Dùng để định nghĩa cốt lõi của lớp, thành phần chung của lớp và sử dụng cho nhiều đối tượng cùng kiểu.
Cần thời gian để tìm phương thức thực tế tương ứng với lớp dẫn đến thời gian chậm hơn 1 chút.	Nhanh hơn so với interface.
Khi ta thêm mới 1 khai báo. Ta phải tìm hết tất cả những lớp có thực thi interface này để định nghĩa nội dung cho phương thức mới.	Đối với abstract class, khi đĩnh nghĩa 1 phương thức mới ta hoàn toàn có thể định nghĩa nội dung phương thức là rỗng hoặc những thực thi mặc định nào đó. Vì thế toàn bộ hệ thống vẫn chạy bình thường.

Viết câu lệnh nayf trước khi bạn muốn in ra màng hình bằng tiếng việt.

```
Console.OutputEncoding = System.Text.Encoding.UTF8;
```