

# Bài: Các loại phạm vi truy cập trong Lập trình hướng đối tượng

Xem bài học trên website để ủng hộ Kteam: [Các loại phạm vi truy cập trong Lập trình hướng đối tượng.](#)

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage [How Kteam](#) nhé!

## Dẫn nhập

Ở các bài học trước, chúng ta đã cùng nhau tìm hiểu về [CLASS TRONG C#](#). Hôm nay chúng ta sẽ cùng tìm hiểu về **Các loại phạm vi truy cập trong C#**.

## Nội dung

Để đọc hiểu bài này tốt nhất các bạn nên có kiến thức cơ bản về các phần:

- [BIẾN](#) và [KIỂU DỮ LIỆU](#) trong C#
- [TOÁN TỬ TRONG C#](#)
- [CÂU ĐIỀU KIỆN TRONG C#](#)
- [CẤU TRÚC CƠ BẢN CỦA VÒNG LẶP TRONG C#](#)
- [CẤU TRÚC HÀM CƠ BẢN TRONG C#](#)
- [MẢNG MỘT CHIỀU TRONG C#](#)


Trong bài học này, chúng ta sẽ cùng tìm hiểu các vấn đề:

- Các loại phạm vi truy cập và ý nghĩa.
- Phương thức truy vấn, phương thức cập nhật.
- Từ khoá [get](#) và [set](#).

## Các loại phạm vi truy cập và ý nghĩa

**Phạm vi truy cập** là cách mà người lập trình quy định về quyền được truy xuất đến các thành phần của lớp.

Trong C# có 5 loại phạm vi truy cập:

Phạm vi truy cập	Ý nghĩa
<a href="#">public</a>	Không hạn chế. Thành phần mang thuộc tính này có thể được truy cập ở bất kỳ nơi nào.
<a href="#">private</a>	Thành phần mang thuộc tính này là thành phần riêng tư <b>chỉ</b> có nội bộ bên trong lớp chứa nó mới có quyền truy cập.
<a href="#">protected</a>	Tương tự như <a href="#">private</a> ngoài ra còn có thể truy cập từ lớp dẫn xuất lớp chứa nó. (sẽ được trình bày trong bài <a href="#">KẾ THỪA TRONG C#</a> )
<a href="#">internal</a>	Chỉ được truy cập trong cùng 1 Assembly (nói cách khác là cùng project). Thuộc tính này thường được dùng cho <a href="#">class</a> .
 <a href="#">protected internal</a>	Tương tự như <a href="#">internal</a> ngoài ra còn có thể truy cập từ lớp dẫn xuất lớp chứa nó (lớp dẫn xuất sẽ được trình bày trong bài <a href="#">KẾ THỪA TRONG C#</a> )

Lưu ý:

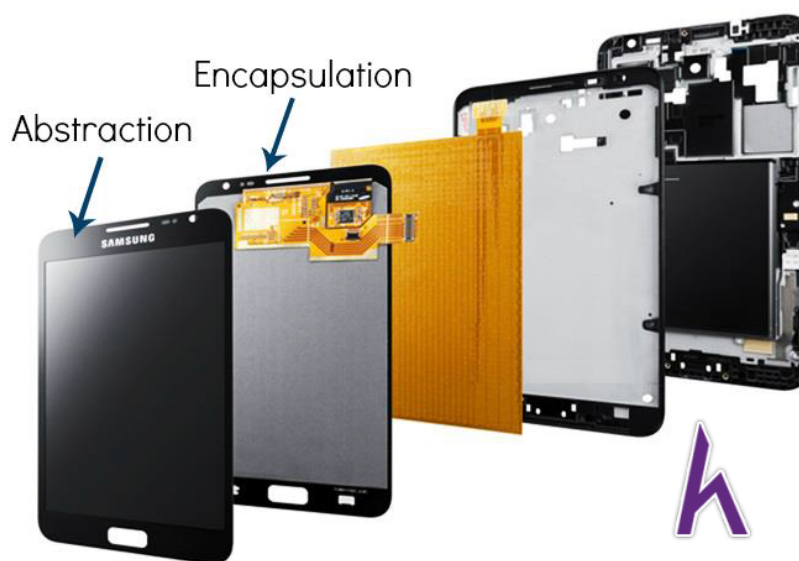
- Nếu khai báo lớp mà không chỉ ra phạm vi cụ thể thì phạm vi mặc định là **internal**.
- Nếu khai báo thành phần bên trong lớp mà không chỉ ra phạm vi cụ thể thì phạm vi mặc định là **private**.

Tính đóng gói là 1 trong 4 đặc điểm của lập trình hướng đối tượng (các đặc điểm của lập trình hướng đối tượng đã được trình bày trong bài [TỔNG QUAN LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG](#)).

### Trong tính đóng gói có 2 ý chính:

- Các dữ liệu và phương thức có liên quan với nhau được đóng gói thành các lớp để tiện cho việc quản lý và sử dụng. Điều này được thể hiện qua cách ta xây dựng 1 **class**.
- Đóng gói còn để che giấu một số thông tin và chi tiết cài đặt nội bộ để bên ngoài không thể nhìn thấy. Điều này được thể hiện qua các phạm vi truy cập đã trình bày ở trên. Cụ thể:
  - **Các thuộc tính** thường sẽ có phạm vi là **private**. Vì đây chính là các thông tin nội bộ của lớp không thể để truy cập 1 cách tùy tiện được (che giấu thông tin).
  - **Các phương thức** thường sẽ có phạm vi là **public**. Vì đây chính là các hành vi (thao tác) mà lớp hỗ trợ cho chúng ta thực hiện những công việc nhất định nên cần phải cho phép mọi người có thể sử dụng được.

Một ví dụ thực tế về tính đóng gói. Nhà sản xuất chế tạo ra 1 smartphone họ phải đóng gói che giấu đi những mạch điện tử bên trong và chỉ để lộ ra màn hình, khe sạc, camera để giao tiếp.



Ví dụ:

C#:

```

/*
    Vì không chỉ định từ khoá cụ thể nên class SinhVien sẽ có phạm vi là internal.
*/

class SinhVien
{
    /*
        Các thuộc tính đều mang phạm vi là private. Vì thế chỉ được sử dụng nội bộ trong class
        Ra bên ngoài sẽ không truy cập được.
    */

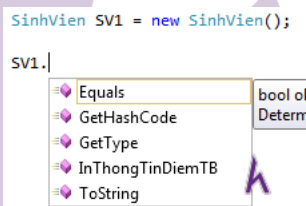
    private string MASV;
    private string HoTen;
    private double DiemToan;
    private double DiemVan;

    /*
        Phương thức này có phạm vi là public nên có sử dụng từ bên ngoài.
        Và vì phương thức này nằm trong lớp nên có thể sử dụng các thuộc tính private ở trên.
    */

    public void InThongTinDiemTB()
    {
        double DTB = (DiemToan + DiemVan) / 2;
        Console.WriteLine(" Sinh vien " + HoTen + " co diem TB la: " + DTB);
    }
}

```

Trong hàm main ta khởi tạo đối tượng thuộc lớp **SinhVien** xem thế nào.



```

SinhVien SV1 = new SinhVien();
SV1.

```

Rõ ràng các thuộc tính đều đã ẩn đi và không thể truy cập chỉ còn lại các thành phần **public**.

## Phương thức truy vấn, phương thức cập nhật

Một vấn đề đặt ra là nếu như mọi thuộc tính bên trong lớp đều là **private** thì khi người dùng bên ngoài muốn xem hoặc thay đổi giá trị thì phải làm sao?

Đến đây đòi hỏi người lập trình phải viết ra **1 cách nào đó** để hỗ trợ người dùng làm điều này. Và cách để người dùng có thể xem hoặc thay đổi giá trị cho các thuộc tính bên trong lớp đó chính là thông qua các **phương thức truy vấn, phương thức cập nhật**.

**Phương thức truy vấn** là phương thức giúp người dùng có thể xem được dữ liệu của 1 thuộc tính nào đó. Cụ thể, phương thức truy vấn chỉ cần trả về giá trị của thuộc tính tương ứng là đủ.

**Phương thức cập nhật** là phương thức giúp người dùng có thể thay đổi giá trị cho 1 thuộc tính nào đó. Cụ thể, phương thức cập nhật chỉ cần thực hiện cập nhật giá trị mới cho thuộc tính tương ứng (có thể kiểm tra tính đúng đắn của dữ liệu trước khi truyền vào).

Thực ra, phương thức truy vấn hay phương thức cập nhật chỉ là một phương thức bình thường. Về khai báo và sử dụng hoàn toàn như phương thức bình thường.

Một số quy ước nhỏ về cách đặt tên các phương thức này:

- Những phương thức truy vấn nên bắt đầu bằng từ khoá **get** và kèm theo sau là tên thuộc tính tương ứng. Ví dụ: getHoTen(), getDiemToan(), ...
- Những phương thức cập nhật nên bắt đầu bằng từ khoá **set** và kèm theo sau là tên thuộc tính tương ứng. Ví dụ: setDiemToan(), setHoTen(), ...
- Nếu thuộc tính kiểu luận lý (**bool**) thì tên phương thức truy vấn nên bắt đầu bằng từ khoá **is** và kèm theo sau là tên thuộc tính tương ứng.
- Phương thức truy vấn sẽ có kiểu trả về trùng với kiểu dữ liệu của thuộc tính tương ứng và không có tham số truyền vào.
- Phương thức cập nhật sẽ có kiểu trả về là **void** và có 1 tham số truyền vào có kiểu dữ liệu trùng với kiểu dữ liệu của thuộc tính tương ứng.

**Ví dụ:**

**C#:**

```
class SinhVien
{
    private string MASV;
    private string HoTen;
    private double DiemToan;
    private double DiemVan;

    /*
        Đây là phương thức truy vấn giá trị của thuộc tính MASV
        Vì thế phương thức sẽ trả về string (trùng với kiểu dữ liệu của thuộc tính MASV)
    */
    public string getMASV()
    {
        return MASV;
    }

    /*
        Đây là phương thức cập nhật giá trị cho thuộc tính DiemToan
        Vì thế phương thức có 1 tham số truyền vào kiểu double trùng với kiểu của DiemToan.
    */
    public void setDiemToan(int diemtoan)
    {
        DiemToan = diemtoan;
    }
}
```

## Từ khoá get và set

Trong C#, **phương thức truy xuất** và **phương thức cập nhật** đã được nâng cấp lên thành 1 cấu trúc mới ngắn gọn hơn và tiện dụng hơn đó là **property**.

Sử dụng property giúp ta có thể thao tác dữ liệu tự nhiên hơn nhưng vẫn đảm bảo tính đóng gói của lập trình hướng đối tượng.

## Cú pháp:

```
<kiểu dữ liệu> <tên property>

{

    get { return <tên thuộc tính>; }

    set { <tên thuộc tính> = value; }

}
```

Trong đó:

- **<kiểu dữ liệu>** là kiểu dữ liệu của property. Thường sẽ trùng với kiểu dữ liệu của thuộc tính **private** tương ứng bên trong lớp.
- **<tên property>** là tên do người dùng đặt và tuân theo quy tắc đặt tên đã trình bày trong bài [BIẾN TRONG C#](#).
- **get, set, value** là từ khoá có ý nghĩa:
  - Từ khoá **get** tương đương với phương thức truy vấn.
  - Từ khoá **set** tương đương với phương thức cập nhật.
  - Từ khoá **value** đại diện cho giá trị mà người gán vào property (tương đương với tham số truyền vào của phương thức cập nhật).
- **<tên thuộc tính>** là tên thuộc tính thực sự bên trong lớp.

**Ví dụ:** ta có 1 thuộc tính **diemLy** cần được đóng gói. Khi đó ta sẽ có 1 property tương ứng để thực hiện điều này:

**C#:**

```
private double diemLy;
public double DiemLy
{
    get { return diemLy; }
    set { diemLy = value; }
}
```

Rõ ràng là mọi thứ trở nên ngắn gọn hơn nhiều.

Khi sử dụng thì ta xem **property** như một thuộc tính đã được **public**.

**C#:**

```
SinhVien SV1 = new SinhVien();

SV1.DiemLy = 8; // khi gán giá trị cho property thì các câu lệnh bên trong set sẽ được thực hiện

Console.WriteLine(" Diem ly: " + SV1.DiemLy); // khi lấy giá trị của property thì các câu lệnh bên trong get sẽ được thực hiện.
```

### Lưu ý:

- Người ta dùng **Property** thay cho phương thức truy vấn, phương thức cập nhật vì thế tên **property** thường phải làm gợi nhớ đến tên thuộc tính **private** bên trong lớp.
- Tùy theo nhu cầu và tính bảo mật mà người lập trình có thể ngăn không cho gán giá trị hoặc ngăn không cho lấy dữ liệu bằng cách bỏ đi từ khoá tương ứng.

Bạn hoàn toàn có thể viết những kiểm tra nếu muốn vào bên trong **get, set**.

**C#:**

```
private double diemLy;
public double DiemLy
{
    get { return diemLy; }
    set
    {
        /*
         * Kiểm tra điểm lý có thoả mãn hay không
         * Nếu có sẽ thực hiện gán vào thuộc tính private.
         * Nếu không sẽ không làm gì cả.
         * Điều này nhằm đảm bảo tính đúng đắn của dữ liệu.
         */

        if (value <= 10 || value >= 0)
        {
            diemLy = value;
        }
    }
}
```

## Kết luận

Nội dung bài này giúp các bạn nắm được:

- Các loại phạm vi truy cập và ý nghĩa.
- Phương thức truy vấn, phương thức cập nhật.
- Từ khoá `get` và `set`.

Bài sau chúng ta sẽ tìm hiểu về [TỪ KHÓA STATIC TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG](#).

Cảm ơn các bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên “**Luyện tập – Thử thách – Không ngại khó**”.

