Bài 6.4: Tính đóng gói dữ liệu

- ✓ Khái niệm và đặc điểm
- ✓ Biểu hiện của tính đóng gói
- ✓ Properties & expression-bodied
- ✓ Auto implement properties
- ✓ Cú pháp khởi tạo đối tượng
- ✓ Ví dụ minh họa và bài tập

Khái niệm và đặc điểm

- ✓Lập trình hướng đối tượng có 4 đặc trưng: tính kế thừa, đóng gói, đa hình và trừu tượng.
- ✓Tính đóng gói dữ liệu: dữ liệu và hành động liên quan đến dữ liệu nên được gói gọn trong một lớp.
- ✓Tính đóng gói dữ liệu cho phép ẩn giấu các triển khai cụ thể của phương thức, thuộc tính và chỉ cung cấp các giao diện cơ bản nhất cho bên sử dụng.
- ✓Tính đóng gói dữ liệu liên quan đến quan điểm rằng, dữ liệu của đối tượng không nên được truy cập trực tiếp bởi chính đối tượng đó từ bên ngoài lớp.
- ✓Từ đó, các thành phần dữ liệu của lớp ta để là private nhằm giữ bí mật. Muốn truy cập tới các thành phần đó ta cần thông qua các phương thức public của lớp.
- ✓Từ giờ về sau dữ liệu của lớp(các trường) ta luôn để private nếu không có yêu cầu cụ thể nào.

Biểu hiện của tính đóng gói dữ liệu

- ✓Ta có thể triển khai tính đóng gói dữ liệu theo 2 cách:
  - ✓ Khai báo một trường private kèm theo cặp phương thức accessor(get) và mutator(set).
  - ✓ Sử dụng public property.
- ✓Trường private sẽ có 1 dấu gạch dưới trước tên. Ví dụ: _name, _age, _salary…
- ✓Với cách 1: phương thức get luôn có kiểu trả về cùng kiểu với kiểu của trường private. Phương thức set thường nhận vào một tham số cùng kiểu với kiểu của trường private.
- ✓Phương thức get/set bắt đầu với Get/Set + tên trường dữ liệu viết liền, viết hoa chữ cái đầu từ.
- ✓Ta có thể sử dụng triển khai đầy đủ hoặc rút gọn bằng cách sử dụng biểu thức bản thân.
- ✓Thường thì trong các setter sẽ chứa các đoạn lệnh kiểm tra, xử lý trước khi gán giá trị cho trường dữ liệu.
- ✓Các getter chỉ làm công việc đơn giản là trả về trường dữ liệu đang muốn thao tác.

Ví dụ

```
0 references
class Lesson64
{
    0 references
    static void Main()
    {
        Student student = new Student("SV1001", "Nguyen Hoang Long", 20, 3.25f);
        student.SetFullName("Nguyen Thanh Luan"); // ok
        student._age = 23; // không thể trực tiếp truy cập tới _age
    }
}
```

Truy cập từ bên ngoài lớp
vào thành phần private là không được phép!

```
3 references
class Student
{
    private string _id;
    private string _fullName;
    private int _age;
    private float _gpa;

    1 reference
    public Student(string id, string fullName, int age, float gpa)...
    // cặp get/set sử dụng biểu thức bản thân
    0 references
    public string GetId() => _id;
    0 references
    public void SetId(string id) => this._id = id;
    // hoặc triển khai đầy đủ get/set
    1 reference
    public void SetFullName(string fullName)
    {
        // kiểm tra, xử lý...
        _fullName = fullName; // gán giá trị cho _fullName
    }
```

Sử dụng properties

- ✓Bản chất property là một container gồm 2 thành phần trong thân nó: getter và setter.
- ✓Access modifier của property luôn để public. Kiểu trả về là kiểu của trường dữ liệu mà nó thao tác.
- ✓Property không có cặp ngoặc tròn của phương thức và không nhận bất kì tham số nào cả.
- ✓Phần getter được thay bằng get, phần setter thay bằng set.
- ✓Trong get/set ta có thể thực hiện các hành động kiểm tra, xử lý trước khi tới phần gán/trả về dữ liệu.
- ✓Trong khối set ta sử dụng contextual keyword value để ám chỉ giá trị được truyền vào(tham số). Nó ngầm định có kiểu cùng kiểu với kiểu của property.
- ✓Ưu điểm của property là đóng gói thao tác của accessor và mutator vào 1 khối thống nhất.

Ví dụ

```
class Student
{
    private string _id;
    private string _fullName;
    private int _age;
    private float _gpa;

    // property đi liền với _age:
    0 references
    public int Age {
        get { return _age; }
        set {
            if(value < 0)
            {
                Console.WriteLine("Tuoi khong hop le.");
                _age = 0;
            } else
            {
                _age = value;
            }
        }
    }
```

Sử dụng expression-bodied với properties

- ✓Nếu phần thân của một phương thức hoặc get/set chỉ gồm các lệnh đơn giản có thể đưa nó về 1 dòng bằng biểu thức bản thân với cú pháp =>.
- ✓Ví dụ:

```
0 references
public float Gpa {
    get => _gpa;
    set => _gpa = value;
}
```

Auto implement properties

- ✓Trong trường hợp nếu thuộc tính chỉ gồm get/set đơn giản như sau, ta nên đơn giản hóa code bằng cách sử dụng cú pháp auto property. Hỗ trợ từ C# 7 về sau.

```
0 references
public float Gpa {
    get => _gpa;
    set => _gpa = value;
}
```

- ✓Chuyển thành:

```
0 references
public float Gpa { get; set; }
```

- ✓Khi ta tạo auto property, chương trình sẽ tự sinh ra một trường private tương ứng gắn liền với property khi biên dịch chương trình.
- ✓Các lệnh gán, trả về trong get/set cũng tự động được triển khai ở đằng sau và không thể hiện ra bên ngoài để ta nhìn thấy.

Auto implement properties

- ✓Khi tạo auto property, chương trình sẽ tự động gán giá trị mặc định của kiểu property cho biến đằng sau property đó.
- ✓Ta cũng có thể chủ động gán giá trị khởi tạo mong muốn cho auto property:

```
0 references
public float Gpa { get; set; } = 1.0f; // khởi tạo auto property
```

Read-only, write-only properties

- ✓Nếu muốn 1 property chỉ cho phép ra đọc ta chỉ cần bỏ phần set khỏi property.
- ✓Nếu muốn 1 property chỉ cho phép ghi, không đọc, ta chỉ cần bỏ phần get khỏi property.
- ✓Auto property không cho phép tạo write-only property.
- ✓Những hành động trên sẽ tạo property với chỉ 1 khả năng.
- ✓Thông thường hiếm gặp write-only property.
- ✓Ví dụ:

```
0 references
class BankCard
{
    private string _cardType;
    0 references
    public int Id { get; set; }
    0 references
    public string CardNumber { get;} // read only
    0 references
    public string CardType { set => _cardType = value; } // write only
}
```

Sử dụng access modifier trong properties

- ✓Trong property có thể chỉ định mực độ truy cập cho get/set bằng cách sử dụng access modifier.
- ✓Tức là tùy theo yêu cầu thực tế và mục đích sử dụng ta có thể thiết lập các access modifier khác nhau riêng lẻ cho cụm get hoặc set.
- ✓Ví dụ: cho phép đọc dữ liệu từ bên ngoài lớp nhưng không cho phép thay đổi dữ liệu từ bên ngoài lớp mà chỉ cho phép cập nhật chúng từ bên trong, ta sẽ để public get và private set.

```
        BankCard card = new BankCard();
        card.Id = 100; // không thể truy cập
        Console.WriteLine("Ma the: " + card.Id); // đọc ra thì ok
    }
}

2 references
class BankCard
{
    private string _cardType;
    2 references
    public int Id { get; private set; }
    0 references
    public string CardNumber { get;} // read only
    0 references
    public string CardType { set => _cardType = value; } // write only
}
```

Cú pháp khởi tạo đối tượng

- ✓Để tạo đối tượng của lớp, ngoài cách phổ biến là gọi new một constructor và truyền đối số phù hợp vào đó, ta còn có thể sử dụng cú pháp khởi tạo đối tượng kiểu mới.
- ✓Cú pháp này có thể kết hợp cả gọi constructor và gán giá trị cho các thuộc tính public.

Cú pháp khởi tạo đối tượng

```
13 references
class Point
{
    4 references
    public int X { get; set; }
    5 references
    public int Y { get; set; }
    3 references
    public Point() : this(0, 0) { }
    1 reference
    public Point(int x) : this(x, 0) { }
    2 references
    public Point(int x, int y)
    {
        X = x;
        Y = y;
    }
    0 references
    public override string ToString() => $"Point({X}, {Y})";
}

0 references
static void Main()
{
    Point point = new Point(); // ok
    // cách khác:
    Point point1 = new Point { X = 3, Y = 4 };
    Point point2 = new Point() { X = 3, Y = 4 };
    Point point3 = new Point(x:5) { Y = 8 };
    Console.WriteLine(point3); // Point(5, 8)
```

Nội dung tiếp theo

Lớp partial và Records