Chương 3

Hiện thực mô hình hướng đối tượng trên C#

3.0 Dẫn nhập

- Chương này giới thiệu cú pháp của phát biểu class C# được dùng để đặc tả chi tiết hiện thực 1 loại đối tượng được dùng trong chương trình.
- Chương này cũng giới thiệu cú pháp các phát biểu để định nghĩa các thành phần cấu thành đối tượng như thuộc tính vật lý, thuộc tính giao tiếp, tác vụ chức năng, toán tử, delegate, event, indexer.
- Chương này cũng phân biệt 2 loại thành phần được đặc tả trong 1 class : thành phần dùng chung (static) và thành phần nhân bản theo từng đối tượng.

3.1 Tổng quát về phát biểu class của C#

Ngôn ngữ C# (hay bất kỳ ngôn ngữ lập trình nào khác) cung cấp cho người lập trình nhiều phát biểu (statement) khác nhau, trong đó phát biểu class để đặc tả chi tiết hiện thực từng loại đối tượng cấu thành phần mềm là phát biểu quan trọng nhất. Sau đây là 1 template của 1 class C# :

```
class MyClass : BaseClass, I1, I2, I3 {
  //định nghĩa các thuộc tính vật lý của đối tượng
  //định nghĩa các tác vụ chức năng, các toán tử
  //định nghĩa các thuộc tính giao tiếp (luận lý)
  //định nghĩa các đại diện hàm chức năng (delegate)
  //định nghĩa các sự kiện (event)
  //định nghĩa indexer của class
  //định nghĩa các tác vụ quản lý đời sống đối tượng
}
```


3.1 Tổng quát về phát biểu class của C#

- Khi định nghĩa 1 class mới, ta có thể thừa kế tối đa 1 class đã có (đơn thừa kế), tên class này nếu có, phải nằm ở vị trí đầu tiên ngay sau dấu ngăn ":".
- Khi định nghĩa 1 class, ta có thể hiện thực nhiều interface khác nhau (đa hiện thực), danh sách này nếu có, phải nằm sau tên class cha. Trong trường hợp nhiều interface có cùng 1 tác vụ (phân biệt bằng chữ ký) và nếu class muốn hiện thực chúng khác nhau thì ta dùng tên dạng phân cấp :

```
class MyClass : BaseClass, I1, I2, I3 {
  //hiện thực các tác vụ cùng chữ ký trong các interface khác nhau
  void I1.func1() {}
  void I2.func1() {}
  void I3.func1() {}
  ...
}
```

3.2 Định nghĩa thuộc tính vật lý

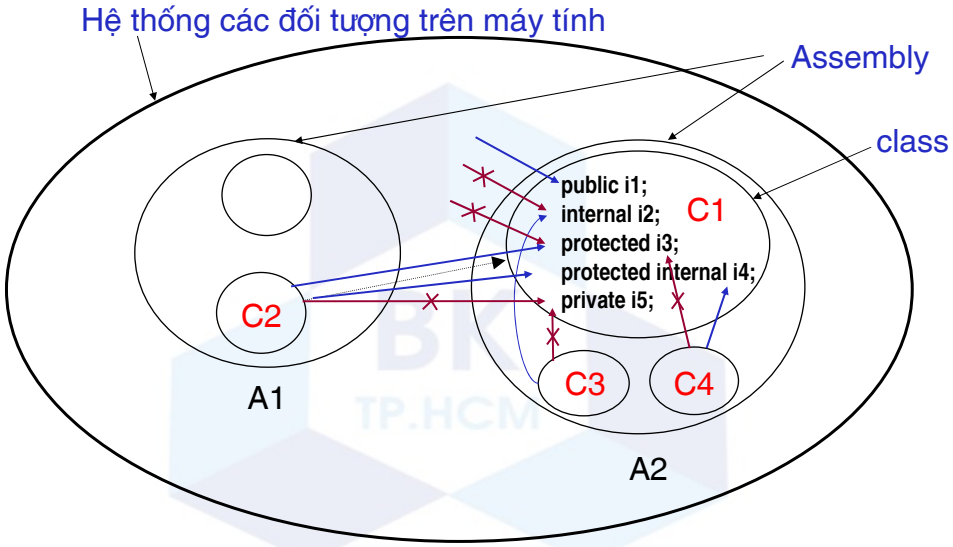
- Mỗi thuộc tính vật lý của đối tượng là 1 biến dữ liệu cụ thể. Phát biểu định nghĩa 1 thuộc tính vật lý sẽ đặc tả các thông tin sau về thuộc tính tương ứng :
  - Tên nhận dạng.
  - Kiểu dữ liệu.
  - Giá trị ban đầu.
  - Tầm vực truy xuất
- Cú pháp đơn giản để định nghĩa 1 thuộc tính vật lý như sau :

  [scope] *type name* [= *value*];

3.2 Định nghĩa thuộc tính vật lý

- thành phần scope miêu tả tầm vực truy xuất của thuộc tính, có thể chọn 1 trong 5 khả năng sau :
  - public : thuộc tính có thể được truy xuất bất kỳ đâu.
  - internal : thuộc tính có thể được truy xuất bất kỳ đâu trong cùng assembly chứa class.
  - protected : thuộc tính có thể được truy xuất bởi class hiện hành và các class con, cháu.
  - protected internal : thuộc tính có thể được truy xuất bất kỳ đâu trong cùng assembly chứa class hay các class con, cháu.
  - private : thuộc tính chỉ có thể được truy xuất nội bộ trong class hiện hành.
  - nếu thành phần scope không được miêu tả tường minh, thuộc tính sẽ có tầm vực internal.

3.2 Định nghĩa thuộc tính vật lý



3.2 Định nghĩa thuộc tính vật lý

- thành phần *type* thường là tên kiểu dữ liệu của thuộc tính tương ứng, nó có thể là tên kiểu giá trị hay tên kiểu tham khảo.
- thành phần *name* là tên nhận dạng thuộc tính.
- thành phần [= *value*] miêu tả biểu thức xác định trị ban đầu của thuộc tính.
- thành phần nào nằm trong [] là nhiệm ý (optional), có thể có hoặc không. Các thành phần khác bắt buộc phải có.
- Thí dụ :

```
private int dorong = 10;
private int docao = 10;
```

3.3 Định nghĩa tác vụ chức năng

- Mỗi tác vụ (operation) thực hiện 1 chức năng xác định, rõ ràng nào đó mà bên ngoài đối tượng (client) cần dùng. Định nghĩa tác vụ gồm 2 phần : định nghĩa interface sử dụng và định nghĩa thuật giải chi tiết mà tác vụ thực hiện (method).
- Lệnh định nghĩa 1 tác vụ thường gồm 5 phần sau :
  [scope | attribute] *return_type name* **(**arg_list**)** *body*
  - scope miêu tả tầm vực truy xuất của tác vụ : public, protected, internal, protected internal, private.
  - attribute miêu tả tính chất hoạt động của tác vụ : static, virtual, sealed, override, abstract, extern.
  - return_type là tên kiểu của giá trị mà tác vụ sẽ trả về.
  - name là tên tác vụ, arg_list là danh sách từ 0 tới n tham số hình thức cách nhau bởi dấu ',', định nghĩa mỗi tham số hình thức gần giống như định nghĩa thuộc tính vật lý.

3.4 Định nghĩa toán tử chức năng

- Mỗi toán tử (operator) thực hiện 1 phép toán xác định. Toán tử là trường hợp đặc biệt của tác vụ. Định nghĩa toán tử gồm 2 phần : định nghĩa interface sử dụng và định nghĩa thuật giải chi tiết mà toán tử thực hiện (method).
- Lệnh định nghĩa 1 toán tử thường gồm 6 phần sau :
  [scope] *return_type* **operator** *name* **(**arg_list**)** *body*
  - scope miêu tả tầm vực truy xuất của toán tử : public, static, extern.
  - return_type là tên kiểu của giá trị mà toán tử sẽ trả về.
  - name là tên toán tử : +,-,*,/,...
  - arg_list là danh sách từ 0 tới 2 tham số hình thức cách nhau bởi dấu ',', định nghĩa mỗi tham số hình thức gần giống như định nghĩa thuộc tính vật lý.

3.5 Định nghĩa thuộc tính giao tiếp (luận lý)

- Mỗi thuộc tính giao tiếp (luận lý) chẳng qua là 1 hay 2 tác vụ get/set (tham khảo/thiết lập) nội dung thuộc tính tương ứng. Định nghĩa thuộc tính giao tiếp là định nghĩa 1 hay 2 tác vụ get/set.
- Lệnh định nghĩa 1 thuộc tính thường có dạng sau :

  [scope | attribute] *type name* **{[**getdef] [setdef]**};**
  - scope, attirbute, type, name có ý nghĩa giống như lệnh định nghĩa tác vụ.
  - getdef và setdef là lệnh định nghĩa tác vụ get và set thuộc tính tương ứng.

```
class Rectangle {
        private int m_cao;  //thuộc tính vật lý
        public int Cao {  //thuộc tính luận lý
           get { return m_cao; }
           set { if (value>0 && value <1024) m_cao = value; }
        }
}
```


3.6 Định nghĩa đối tượng đại diện hàm (delegate)

- Nhiều khi chúng ta cần viết lệnh gọi hàm mà chưa biết tên cụ thể, tên của hàm chỉ có thể xác định tại thời điểm run-time. Delegate của C# cho phép ta giải quyết được yêu cầu này.
- Delegate là 1 class đối tượng đặc biệt, đối tượng delegate chỉ chứa 1 field thông tin, field này là địa chỉ của 1 hàm chức năng nào đó.
- Delegate đặc biệt hữu dụng khi kết hợp với sự kiện (Event) mà ta sẽ trình bày sau.
- Lệnh định nghĩa delegate thường có dạng :

  [scope] **delegate** return_*type name* **(***arg_list***);**
  - scope, return_type, name, arg_list có ý nghĩa giống như lệnh định nghĩa tác vụ.

3.7 Định nghĩa sự kiện (Event)

- Tác vụ chỉ có thể được (gọi) kích hoạt bởi người lập trình, trong khi nhiều lúc ta muốn người dùng có thể kích hoạt trực tiếp chức năng nào đó của đối tượng (thí dụ đối tượng giao diện). Event là phương tiện giải quyết yêu cầu này.
- Event là 1 đối tượng thuộc class delegate, sau khi được khởi động, nó có thể miêu tả từ 1 tới n tác vụ chức năng mà sẽ được tự kích hoạt mỗi khi event xảy ra.
- Lệnh định nghĩa Event giống như lệnh định nghĩa thuộc tính dữ liệu :

  [scope] **event** *delegate_type name*;
  - scope, name có ý nghĩa giống như lệnh định nghĩa thuộc tính.
  - delegate_type là tên của 1 delegate đã định nghĩa trước.

3.8 Định nghĩa phần tử quản lý danh sách (indexer)

- Để truy xuất 1 đối tượng thuộc 1 class, ta dùng biến đối tượng. Thông qua biến đối tượng (tham khảo), ta truy xuất từng thành phần được phép (thuộc tính, tác vụ, toán tử,...) thông qua cú pháp gởi thông điệp : objVar.tên thành phần.
- Ngoài khả năng thông thường trên, C# còn cho phép kết hợp với đối tượng 1 danh sách các phần tử dữ liệu thuộc 1 kiểu nào đó. Indexer chính là khả năng này.
- Nếu thuộc tính giao tiếp cho phép ta miêu tả 1 giá trị luận lý duy nhất thì Indexer cho phép ta miêu tả 1 danh sách nhiều giá trị luận lý. Lệnh định nghĩa Indexer giống như lệnh định nghĩa thuộc tính luận lý :

  [scope | attribute] *type* **this [int i] {**[getdef] [setdef]**};**

3.8 Định nghĩa phần tử quản lý danh sách (indexer)

- scope, attirbute, type có ý nghĩa giống như lệnh định nghĩa thuộc tính.
- getdef và setdef là lệnh định nghĩa tác vụ get và set phần tử thứ i trong danh sách.

```
class Rectangle {
    private int[] arr = new int[100];
    public int this[int index]  { //định nghĩa Indexer
        get {
            if (index < 0 || index >= 100) {  return 0; }
            else { return arr[index]; }
        }
        set {
            if (!(index < 0 || index >= 100)) { arr[index] = value;  }
        }
    }
}
```


3.8 Định nghĩa phần tử quản lý danh sách (indexer)

- Để truy xuất phần tử thứ i trong danh sách, ta dùng cú pháp giống như truy xuất biến array.

```
Rectangle objRec = new Rectangle();
objRec[0] = 0;
int ret = objRec[10];
```

3.9 Thành phần static và thành phần không static

- Phát biểu class được dùng để đặc tả các đối tượng cùng loại mà phần mềm dùng. Về nguyên tắc, khi đối tượng được tạo ra (bằng lệnh new), nó sẽ chứa tất cả các thành phần được đặc tả trong class tương ứng. Tuy nhiên, nếu xét chi li thì VC# cho phép đặc tả 2 loại thành phần trong 1 class như sau :
  1. Thành phần static : là thành phần có từ khóa static trong lệnh định nghĩa nó. Đây là thành phần kết hợp với class, nó không được nhân bản cho từng đối tượng và như thế đối tượng không thể truy xuất nó. Cách duy nhất để truy xuất thành phần static là thông qua tên class.

```
//Console là tên class chứa các hàm truy xuất
//các thiết bị nhập/xuất chuẩn
Console.Writeln("Nội dung cần hiển thị");
```


3.9 Thành phần static và thành phần không static

  2. Thành phần không static : là thành phần không dùng từ khóa static trong lệnh định nghĩa nó. Đây là thành phần kết hợp với từng đối tượng, nó sẽ được nhân bản cho từng đối tượng. Ta truy xuất thành phần không static thông qua tham khảo đối tượng.

```
class Rectangle {
   private int m_cao;  //thuộc tính vật lý
   public int Cao {  //thuộc tính luận lý
      get { return m_cao; }
      set { if (value>0 && value <1024) m_cao = value; }
   }
}
Rectangle r = new Rectangle();
r.Cao = 10;
```

3.9 Các thuộc tính miêu tả hành vi

- Từ khóa virtual trong lệnh định nghĩa tác vụ miêu tả tác vụ này sẽ được xử lý theo cơ chế liên kết động và sẽ đảm bảo được tính đa xạ, tức đảm bảo tính đúng đắn trong lời gởi thông điệp. Biết được điều này, từ đây về sau, mỗi lần định nghĩa 1 tác vụ hay 1 toán tử, ta hãy luôn dùng từ khóa virtual kết hợp với nó.
- Từ khóa sealed trong lệnh định nghĩa tác vụ miêu tả tác vụ này sẽ được khóa lại, class con cháu không còn cơ may override được.
- Từ khóa abstract trong lệnh định nghĩa tác vụ miêu tả tác vụ này chỉ có interface sử dụng, class con cháu sẽ phải override để hiện thực theo yêu cầu riêng.
- Từ khóa override trong lệnh định nghĩa tác vụ miêu tả tác vụ này đã có trong class cha, class hiện hành chỉ muốn override nó chứ không phải định nghĩa mới.

3.9 Các thuộc tính miêu tả hành vi

```
abstract class C1 {
      public void f1() {}  //không cho con override
      public virtual void f2() {}  //cho phép con override
      public abstract void f3();  //để con override
}
class C2 : C1 {
      public void f1() {}  //không được phép
      public sealed void f2() {}  //được phép override và khóa lại
      public override void f3() {}  //phải override
}
class C3 : C2 {
      public void f1() {}  //không được phép
      public void f2() {}  //không được phép
      public void f3() {}  //được phép override
}
```

3.9 Các thuộc tính miêu tả hành vi

- Từ khóa extern trong lệnh định nghĩa tác vụ miêu tả tác vụ này đã được hiện thực ở bên ngoài class hiện hành (thường là trong thư viện DLL). Đây là phương pháp chuyển 1 hàm chức năng trong thư viện DLL truyền thống thành 1 tác vụ của class C# để ứng dung C# có thể gọi được (mặc định thì không gọi được).

```
//chuyển hàm API Windows thành tác vụ C#.
[DllImport("Kernel32.dll", CharSet = CharSet.Auto)]
  public static extern IntPtr FindFirstFile(String fileName, [In, Out]
  FindData findFileData);
[DllImport("Kernel32.dll", CharSet = CharSet.Auto)]
  public static extern Boolean FindNextFile(IntPtr handle, [In, Out]
  FindData findFileData);
```


3.10 Lệnh định nghĩa 1 class C# điển hình

```
class MyClass {
    //1. định nghĩa các thuộc tính vật lý
    private int m_x;
    private int[] arr = new int[100];
    //2. định nghĩa các tác vụ & toán tử chức năng
    public void button1_Click(object sender, System.EventArgs e) {}
    ...
    //3. định nghĩa đối tượng đại diện hàm chức năng
    public delegate void EventHandler (Object sender, EventArgs e);
    //4. định nghĩa sự kiện Click được xử lý bởi delegate EventHandler.
    public event EventHandler Click;
```

3.10 Lệnh định nghĩa 1 class C# điển hình

```
//5. định nghĩa thuộc tính luận lý x
public int x {
    get { return m_x; }
    set { m_x = value; }
}
//6. định nghĩa các tác vụ quản lý đối tượng
public MyClass() { this.Click += new
EventHandler(button1_Click); }
~MyClass() {...}    //hàm destructor
//còn tiếp ở slide kế sau
```


3.10 Lệnh định nghĩa 1 class C# điển hình

```
//7. định nghĩa indexer
public int this[int index]  {
    get {
      //kiểm tra giới hạn để quyết định
      if (index < 0 || index >= 100) {  return 0; }
      else { return arr[index]; }
    }
    set {
      if (!(index < 0 || index >= 100)) { arr[index] = value;  }
    }
}
};
```

Lệnh định nghĩa 1 inreface C# điển hình

```
interface IMyInterface {
    //2. định nghĩa các tác vụ & toán tử chức năng
    void button1_Click(object sender, System.EventArgs e) {}
    //4. định nghĩa sự kiện Click được xử lý bởi delegate EventHandler.
    event EventHandler Click;
    //5. định nghĩa thuộc tính luận lý x
    int x {get; set;}
    //7. định nghĩa indexer
    int this[int index] {get; set;}
}
```


3.11 Kết chương

- Chương này đã giới thiệu cú pháp của phát biểu class C# được dùng để đặc tả chi tiết hiện thực 1 loại đối tượng được dùng trong chương trình.
- Chương này cũng đã giới thiệu cú pháp các phát biểu để định nghĩa các thành phần cấu thành đối tượng như thuộc tính vật lý, thuộc tính giao tiếp, tác vụ chức năng, toán tử, delegate, event, indexer.
- Chương này cũng đã phân biệt 2 loại thành phần được đặc tả trong 1 class : thành phần dùng chung (static) và thành phần nhân bản theo từng đối tượng.