Chương 4

Vòng đời đối tượng và sự tương tác giữa chúng

4.1 Quản lý đời sống đối tượng - Hàm Constructor

Class mô hình các đối tượng cùng loại mà phần mềm dùng. Lúc lập trình, ta chỉ đặc tả class, đối tượng chưa có. Khi ứng dụng chạy, tại thời điểm cần thiết, phần mềm sẽ phải tạo tường minh đối tượng bằng lệnh new :

```
Rectangle objRec = new Rectangle(); //tạo đối tượng
```

Trạng thái của đối tượng là tập giá trị cụ thể của các thuộc tính. Ngay sau đối tượng được tạo ra, nó cần có trạng thái ban đầu xác lập nào đó. Hàm constructor cho phép người lập trình miêu tả hoạt động xác lập trạng thái ban đầu của đối tượng.

Cũng giống như nhiều tác vụ khác, hàm constructor có thể có nhiều "overloaded" khác nhau (với số lượng tham số khác nhau hay tính chất của 1 tham số nào đó khác nhau).

Mỗi lần đối tượng được tạo ra (bởi lệnh new), máy sẽ gọi tự động constructor của class tương ứng. Tùy theo tham số của lệnh new mà constructor nào tương thích sẽ được kích hoạt chạy.

Trong nội bộ 1 class, các tác vụ chỉ có thể truy xuất các thuộc tính của mình và các thuộc tính thừa kế từ cha có tầm vực protected, public, chứ không thể truy xuất trực tiếp các thuộc tính thừa kế từ cha có thuộc tính private. Do đó nếu chỉ chạy constructor của class cần tạo đối tượng thì không thể khởi tạo hết các thuộc tính của đối tượng, cần kích hoạt hết các constructor của các class cha (gián tiếp hay trực tiếp).

Mặc định, khi cần gọi constructor của class cha chạy, máy sẽ gọi constructor không tham số. Nếu người lập trình muốn khác thì phải khai báo lại tường minh "overloaded" nào cần chạy thông qua mệnh đề base() trong lệnh định nghĩa hàm constructor.

```
//class A  có 2 hàm constructor
```

```
class A {
   A() {...}
   A(int i) {...}
   ...
};
//class B thừa kế A, có 2 hàm constructor
class B : A {
   B() : base() {...}
   B(int i) : base (i) {...}
   ...
};
//class C thừa kế B, có 2 hàm constructor
class C : B {
   C() : base () {...}
   C(int i) : base (i) {...}
   ... };
C c1 = new C();  //kích hoạt A() → B() → C()
C c2 = new C(5);   //kích hoạt A(5) → B(5) → C(5)
```

Việc xác định constructor nào được kích hoạt phải theo chiều từ dưới lên bắt đầu từ class được new, nhưng các constructor được chạy thực sự sẽ theo chiều từ trên xuống bắt đầu từ class tổ tiên đời đầu.

4.2 Quản lý đời sống đối tượng - Hàm Destructor

Đối tượng là 1 thực thể, nó có đời sống như bao thực thể khác. Như ta đã biết, khi ta gọi lệnh new, 1 đối tượng mới thuộc class tương ứng sẽ được tạo ra (trong không gian hệ thống), trạng thái ban đầu sẽ được xác lập thông qua việc kích hoạt dây chuyền các constructor của các class thừa kế. Chương trình sẽ lưu giữ tham khảo đến đối tượng trong biến tham khảo để khi cần, gởi thông điệp nhờ đối tượng thực thi dùm 1 tác vụ nào đó.

VC# không cung cấp tác vụ delete để xóa đối tượng khi không cần dùng nó nữa. Thật vậy, đánh giá 1 đối tượng nào đó có cần

dùng nữa hay không là việc không dễ dàng, dễ nhầm lẫn nếu để chương trình tự làm.

Tóm lại, trong VC#, chương trình chỉ tạo tường minh đối tượng khi cần dùng nó, chương trình không quan tâm việc xóa đối tượng và cũng không có khả năng xóa đối tượng.

Như vậy, đối tượng sẽ bị xóa lúc nào, bởi ai ? Hệ thống có 1 module đặc biệt tên là "Garbage collection" (trình dọn rác), module này sẽ theo dõi việc dùng các đối tượng, khi thấy đối tượng nào mà không còn ai dùng nữa thì nó sẽ xóa dùm tự động.

Trình dọn rác không biết trạng thái đối tượng tại thời điểm bị xóa nên nó không làm gì ngoài việc thu hồi vùng nhớ mà đối tượng chiếm. Như vậy rất nguy hiểm, thí dụ như đối tượng bị xóa đã mở, khóa file và đang truy xuất file dỡ dang.

Để giải quyết vấn đề xóa đối tượng được triệt để, trình dọn rác sẽ gọi tác vụ destructor của đối tượng sắp bị xóa, nhiệm vụ của người đặc tả class là hiện thực tác vụ này.

Tác vụ destructor không có kiểu trả về, không có tham số hình thức → không có overloaded, chỉ có 1 destructor/class mà thôi.

Mặc dù người đặc tả class sẽ hiện thực tác vụ destructor nếu thấy cần thiết, nhưng code của chương trình không được gọi trực tiếp destructor của đối tượng. Chỉ có trình dọn rác của hệ thống mới gọi destructor của đối tượng ngay trước khi xóa đối tượng đó.

Destructor của 1 class cũng chỉ xử lý trạng thái đối tượng do các thuộc tính của class đó qui định, nó cần gọi destructor của class cha để xử lý tiếp trạng thái đối tượng do các thuộc tính private của class cha qui định, và cứ thế tiếp tục.

Tóm lại trước khi xóa một đối tượng, trình dọn rác sẽ gọi các destructor theo chiều từ dưới lên, bắt đầu từ class hiện hành của đối tượng, sau đó tới class cha, ... và cuối cùng là class tổ tiên đời đầu (root).

5.3 Hiệu chỉnh thuộc tính các đối tượng giao diện

Muốn tương tác với đối tượng nào đó, ta phải có tham khảo đến đối tượng đó. Thường ta lưu giữ tham khảo đối tượng cần truy xuất trong biến đối tượng (biến tham khảo). Thông qua tham khảo đến đối tượng, ta có thể thực hiện 1 trong các hành động tương tác sau đây :

- truy xuất 1 thuộc tính vật lý của đối tượng có tầm vực cho phép (public hay internal hay protected).
- truy xuất 1 thuộc tính luận lý của đối tượng.
- gọi 1 tác vụ hay toán tử có tầm vực cho phép.
- truy xuất 1 event của đối tượng.
- truy xuất 1 phần tử trong danh sách indexer của đối tượng.

Gọi 1 tác vụ hay 1 toán tử là giống nhau và cần làm rõ chi tiết trong phần sau.

Xét đoạn lệnh sau :

```
class C1 {
        public void func1() {}         //dịch ra hàm mã máy có tên là C1_func1
        public virtual func2() {} //dịch ra hàm mã máy có tên là
C1_func2
}

class C2 : C1 {
        public override func1() {}         //dịch ra hàm mã máy có tên là C2_func1
        public override func2() {}//dịch ra hàm mã máy có tên là C2_func2
}

C1 obj = new C1();
obj.func1();  //lần 1 → gọi hàm mã máy nào ?
//đoạn code có thể làm obj chỉ về đối tượng của class C2, C3,...
obj.func1(); //lần 2 → gọi hàm mã máy nào ?
```

4.4 Liên kết tĩnh trong việc gởi thông điệp

Hai lệnh gởi thông điệp obj.func1() trong slide trước sẽ kích hoạt tác vụ func1() của class C1 hay tác vụ func1() của class C2 ?

1. Dùng kỹ thuật xác định hàm và liên kết tĩnh :

Tại thời điểm dịch, chương trình dịch chỉ biết biến obj thuộc kiểu C1 và nó dịch cả 2 lời gởi thông điệp obj.func1() thành lời gọi hàm C1_func1(). Như vậy mỗi khi máy chạy lệnh obj.func1() lần 1, hàm C1_func1() sẽ được gọi, điều này đúng theo yêu cầu của phần mềm. Nhưng khi máy chạy lệnh obj.func1() lần 2, hàm C1_func1() cũng sẽ được gọi, điều này không đúng theo yêu cầu của phần mềm vì lúc này obj đang tham khảo đối tượng của class C2.

Mặc định, VC# dùng kỹ thuật xác định hàm và liên kết tĩnh khi dịch lời gởi thông điệp, do đó tạo ra độ rủi ro cao, chương trình ứng dụng thường chạy không đúng theo yêu cầu mong muốn!!!

4.5 Liên kết động để đảm bảo tính đa xạ

Bây giờ nếu ta hiệu chỉnh 2 lệnh gởi thông điệp obj.func1() trong slide trước thành obj.func2() thì máy sẽ kích hoạt tác vụ func2() của class C1 hay tác vụ func2() của class C2 ?

2. Dùng kỹ thuật xác định hàm và liên kết động :

Lệnh gởi thông điệp obj.func2() được dịch thành đoạn lệnh máy với chức năng sau : xác định biến obj đang tham khảo đến đối tượng nào, thuộc class nào, rồi gọi hàm func2() của class đó chạy. Như vậy, lần gởi thông điệp 1, biến obj đang tham khảo đối tượng thuộc class C1 nên máy sẽ gọi hàm C1_func2(), điều này đúng theo yêu cầu của phần mềm. Khi máy chạy lệnh obj.func2() lần 2, đoạn code xác định hàm và liên kết động sẽ gọi được hàm C2_func2(), điều này cũng đúng theo yêu cầu của phần mềm. Ta nói lời gởi thông điệp obj.func2() có tính đa xạ.

Trong VC#, nếu dùng từ khóa virtual trong lệnh định nghĩa tác vụ thì 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.

Lưu ý rằng 2 tác vụ constructor và destructor của đối tượng là 2 tác vụ đặc biệt, chúng quản lý đời sống đối tượng và chỉ được gọi bởi hệ thống. Ta không được phép dùng từ khóa virtual khi định nghĩa chúng.

4.6 Xử lý sự kiện luôn có tính đa xạ

Chúng ta hãy viết 1 chương trình nhỏ gồm 1 form giao diện, trong form ta tạo 1 Button có thuộc tính Text="Làm gì đây?", thuộc tính (Name) = btnStart, định nghĩa hàm xử lý sự kiện Click cho nó rồi viết code như sau :

```
//hàm xử lý Click chuột trên button do máy tạo ra
private void btnStart_Click(object sender, EventArgs e) {
        //xuất thông báo để kiểm tra
        MessageBox.Show("Hàm btnStart_Click sẽ xử lý đây");
        //thay đổi hàm xử lý Click cho Button
        this.btnStart.Click -= new EventHandler(btnStart_Click);
        this.btnStart.Click += new EventHandler(btnStart_Click1);
}
```

Hãy viết thêm hàm btnStart_Click1() như sau :

```
//hàm xử lý Click chuột trên button tự viết thêm
private void btnStart_Click1(object sender, EventArgs e) {
        //xuất thông báo để kiểm tra
        MessageBox.Show("Hàm btnStart_Click1 sẽ xử lý đây");
        //thay đổi hàm xử lý Click cho Button
        this.btnStart.Click -= new EventHandler(btnStart_Click1);
        this.btnStart.Click += new EventHandler(btnStart_Click);
}
```

Bây giờ nếu chạy chương trình, lần đầu click chuột ta sẽ thấy hàm btnStart_Click() chạy, nhưng nếu click chuột tiếp thì hàm btnStart_Click1() chạy, cứ thế thay phiên nhau chạy (theo ý muốn người lập trình). Ta nói xử lý sự kiện người dùng luôn có tính đa xạ.

4.7 Kết chương

Chương này đã giới thiệu vòng đời của từng đối tượng trong chương trình, cách thức quản lý đời sống của đối tượng, các thời điểm quan trọng nhất như lúc tạo mới đối tượng, lúc xóa đối tượng cũng như cách miêu tả các hoạt động xảy ra tại các thời điểm này.

Chương này cũng đã giới thiệu sự tương tác giữa các đối tượng trong lúc chúng đang sống để hoàn thành nhiệm vụ của chương trình. Gởi thông điệp là sự tương tác chính yếu giữa các đối tượng, và cần phải có tính đa xạ.