CHƯƠNG 4

Lập trình hướng đối tượng trong PHP

Các chủ đề chính

Mục tiêu

Sau khi hoàn thành chương này, chúng ta sẽ có thể:

- Trình bày được bản chất của lập trình hướng đối tượng.
- Biết cách khai báo và xây dựng các lớp, phương thức,thuộc tính và đối tượng.
- Trình bày được cơ chế đóng kín (ý nghĩa của public, private, protected) trong lập trình hướng đối tượng.
- Biết cách khai báo kế thừa lớp cũng như ghi đè (nạp chồng) hàm.

Câu hỏi kiểm tra mở đầu

Trả lời các câu hỏi sau:

1. Hàm trong lập trình hướng đối tượng được gọi là?
   a. Thuộc tính
   b. Phương thức
   c. Đối tượng
2. Tính kế thừa có ở?
   a. Lập trình hướng cấu trúc
   b. Lập trình hướng đối tượng
   c. Cả (a) và (b)
3. Pascal, VB là ngôn ngữ lập trình?
   a. Hướng đối tượng
   b. Hướng cấu trúc
4. Lớp chỉ có duy nhất trong lập trình hướng đối tượng?
   a. Đúng

b. Sai

1.Lập trình hướng đối tượng là gì?

Lập trình hướng đối tượng (gọi tắt là OOP, từ chữ Anh ngữ `object-oriented programming`), hay còn gọi là lập trình định hướng đối tượng, là kĩ thuật lập trình hỗ trợ công nghệ đối tượng. OOP được xem là giúp tăng năng suất, đơn giản hóa độ phức tạp khi bảo trì cũng như mở rộng phần mềm bằng cách cho phép lập trình viên tập trung vào các đối tượng phần mềm ở bậc cao hơn. Ngoài ra, nhiều người còn cho rằng OOP dễ tiếp thu hơn cho những người mới học về lập trình hơn là các phương pháp trước đó.

Một cách giản lược, đây là khái niệm và là một nỗ lực nhằm giảm nhẹ các thao tác viết mã cho người lập trình, cho phép họ tạo ra các ứng dụng mà các yếu tố bên ngoài có thể tương tác với các chương trình đó giống như là tương tác với các đối tượng vật lý.

Những đối tượng trong một ngôn ngữ OOP là các kết hợp giữa mã và dữ liệu mà chúng được nhìn nhận như là một đơn vị duy nhất. Mỗi đối tượng có một tên riêng biệt và tất cả các tham chiếu đến đối tượng đó được tiến hành qua tên của nó. Như vậy, mỗi đối tượng có khả năng nhận vào các thông báo, xử lý dữ liệu (bên trong của nó), và gửi ra hay trả lời đến các đối tượng khác hay đến môi trường.

2. Các tính chất cơ bản của lập trình OOP

Lập trình hướng đối tượng là một phương pháp lập trình có các tính chất chính sau:

2.1 Tính trừu tượng (abstraction)

Đây là khả năng của chương trình bỏ qua hay không chú ý đến một số khía cạnh của thông tin mà nó đang trực tiếp làm việc lên, nghĩa là nó có khả năng tập trung vào những cốt lõi cần thiết. Mỗi đối tượng phục vụ như là một "động tử" có thể hoàn tất các công việc một cách nội bộ, báo cáo, thay đổi trạng thái của nó và liên lạc với các đối tượng khác mà không cần cho biết làm cách nào đối tượng tiến hành được các thao tác. Tính chất này thường được gọi là sự trừu tượng của dữ liệu.

Tính trừu tượng còn thể hiện qua việc một đối tượng ban đầu có thể có một số đặc điểm chung cho nhiều đối tượng khác như là sự mở rộng của nó nhưng bản thân đối tượng ban đầu này có thể không có các biện pháp thi hành. Tính trừu tượng này thường được xác định trong khái niệm gọi là lớp trừu tượng hay hay lớp cơ sở trừu tượng.

2.2 Tính đóng gói (encapsulation) và che dấu thông tin (information hiding):

Tính chất này không cho phép người sử dụng các đối tượng thay đổi trạng thái nội tại của một đối tượng. Chỉ có các phương thức nội tại của đối tượng cho phép thay đổi trạng thái của nó. Việc cho phép môi trường bên ngoài tác động lên các dữ liệu nội tại của một đối tượng theo cách nào là hoàn toàn tùy thuộc vào người viết mã. Đây là tính chất đảm bảo sự toàn vẹn của đối tượng.

2.3 Tính đa hình (polymorphism)

Thể hiện thông qua việc gửi các thông điệp (message). Việc gửi các thông điệp này có thể so sánh như việc gọi các hàm bên trong của một đối tượng. Các phương thức dùng trả lời cho một thông điệp sẽ tùy theo đối tượng mà thông điệp đó được gửi tới sẽ có phản ứng khác nhau. Người lập trình có thể định nghĩa một đặc tính (chẳng hạn thông qua tên của các phương thức) cho một loạt các đối tượng gần nhau nhưng khi thi hành thì dùng cùng một tên gọi mà sự thi hành của mỗi đối tượng sẽ tự động xảy ra tương ứng theo đặc tính của từng đối tượng mà không bị nhầm lẫn.

Thí dụ khi định nghĩa hai đối tượng "hinh_vuong" và "hinh_tron" thì có một phương thức chung là `"chu_vi".` Khi gọi phương thức này thì nếu đối tượng là `"hinh_vuong"` nó sẽ tính theo công thức khác với khi đối tượng là `"hinh_tron".`

2.4 Tính kế thừa (inheritance)

Đặc tính này cho phép một đối tượng có thể có sẵn các đặc tính mà đối tượng khác đã có thông qua kế thừa. Điều này cho phép các đối tượng chia sẻ hay mở rộng các đặc tính sẵn có mà không phải tiến hành định nghĩa lại. Tuy nhiên, không phải ngôn ngữ định hướng đối tượng nào cũng có tính chất này.

3. Các khái niệm liên quan đến các ngôn ngữ lập trình OOP hiện đại

3.1 Lớp (class)

Một lớp được hiểu là một kiểu dữ liệu bao gồm các thuộc tính và các phương thức được định nghĩa từ trước. Đây là sự trừu tượng hóa của đối tượng. Một đối tượng sẽ được xác lập khi nó được thực thể hóa từ một lớp. Khác với kiểu dữ liệu thông thường, một lớp là một đơn vị (trừu tượng) bao gồm sự kết hợp giữa các phương thức và các thuộc tính. Để có một đối tượng (mà có thể được xem như là một biến) hoạt động được thì việc thực thể hóa sẽ có thể bao gồm việc cài đặt các giá trị ban đầu của các thuộc tính cũng như việc đăng kí bộ nhớ, mà công việc này thường được giao cho các phương thức gọi là "máy kết cấu" (constructor) hay hàm dựng. Ngược lại khi một đối tượng thuộc về một lớp không còn sử dụng nữa thì cũng có thể có một phương thức để xử lý gọi là "máy hủy diệt" (destructor) hay hàm hủy.

Như vậy, để có được các đối tượng thì người lập trình OOP cần phải thiết kế lớp của các đối tượng đó bằng cách xây dựng các thuộc tính và các phương thức có các đặc tính riêng.

Mỗi một phương thức hay một thuộc tính đầy đủ của một lớp còn được gọi tên là một thành viên (member) của lớp đó.

3.2 Lớp con (subclass)

Lớp con là một lớp thông thường nhưng có thêm tính chất kế thừa một phần hay toàn bộ các đặc tính của một lớp khác. Lớp mà chia sẽ sự kế thừa gọi là lớp phụ mẫu (parent class).

3.3 Lớp trừu tượng hay lớp cơ sở trừu tượng (abstract class)

Lớp trừu tượng là một lớp mà nó không thể thực thể hóa thành một đối tượng thực dụng được. Lớp này được thiết kế nhằm tạo ra một lớp có các đặc tính tổng quát nhưng bản thân lớp đó chưa có ý nghĩa (hay không đủ ý nghĩa) để có thể tiến hành viết mã cho việc thực thể hóa. (xem thí dụ)

Ví dụ: Lớp `"hinh_phang"` được định nghĩa không có dữ liệu nội tại và chỉ có các phương thức (hàm nội tại) `"tinh_chu_vi", "tinh_dien_tich".` Nhưng vì lớp `hình_phẳng` này chưa xác định được đầy đủ các đặc tính của nó (cụ thể các biến nội tại là tọa độ các đỉnh nếu là đa giác, là đường bán kính và toạ độ tâm nếu là hình tròn, ...) nên nó chỉ có thể được viết thành một lớp trừu tượng. Sau đó, người lập trình có thể tạo ra các lớp con chẳng hạn như là lớp `"tam_giac"`, lớp `"hinh_tron"`, lớp `"tu_giac",....` Và trong các lớp con này người viết mã sẽ cung cấp các dữ liệu nội tại (như là biến nội tại r làm bán kính và hằng số nội tại Pi cho lớp `"hinh_tron"` và sau đó viết mã cụ thể cho các phương thức `"tinh_chu_vi" và "tinh_dien_tich"`).

3.4 Phương Thức (method)

Là hàm nội tại của một lớp (hay một đối tượng). Tùy theo đặc tính mà người lập trình gán cho, một phương pháp có thể chỉ được gọi bên trong các hàm khác của lớp đó, có thể cho phép các câu lệnh bên ngoài lớp gọi tới nó, hay chỉ cho phép các lớp có quan hệ đặc biệt như là quan hệ lớp con, và quan hệ bạn bè (friend) được phép gọi tới nó. Mỗi phương pháp đều có thể có kiểu trả về, chúng có thể trả các kiểu dữ liệu cổ điển

hay trả về một kiểu là một lớp đã được định nghĩa từ trước. Một tên gọi khác của phương pháp của một lớp là hàm thành viên.

Người ta còn định nghĩa thêm vài loại phương pháp đặc biệt:

Hàm dựng (constructor) là hàm được dùng để cài đặt các giá tri ban đầu cho các biến nội tại và đôi khi còn dùng để khai báo về việc xử dụng bộ nhớ.

Hàm hủy (destructor) là hàm dùng vào việc làm sạch bộ nhớ và hủy bỏ tên của một đối tượng sau khi đã dùng xong, trong đó có thể bao gồm cả việc xóa các con trỏ nội tại và trả về các phần bộ nhớ mà đối tượng đã dùng.

Trong một số trường hợp thì hàm hủy hay hàm dựng có thể được tự động hóa bởi ngôn ngữ OOP như là trường hợp của Visual C++, C#.

Tiện ích (utility) là các hàm chỉ hoạt động bên trong của một lớp mà không cho phép môi trường bên ngoài gọi tới. Các hàm này có thể là những tính toán trung gian nội bộ của một đối tượng mà xét thấy không cần thiết phải cho thế giới bên ngoài của đối tượng biết là gì.

3.5 Thuộc tính (attribute)

Thuộc tính của một lớp bao gồm các biến, các hằng, hay tham số nội tại của lớp đó. Ở đây, vai trò quan trọng nhất của các thuộc tính là các biến vì chúng sẽ có thể bị thay đổi trong suốt quá trình hoạt động của một đối tượng. Các thuộc tính có thể được xác định kiểu và kiểu của chúng có thể là các kiểu dữ liệu cổ điển hay đó là một lớp đã định nghĩa từ trước. Như đã ghi, khi một lớp đã được thực thể hoá thành đối tượng cụ thể thì tập hợp các giá trị của các biến nội tại làm thành trạng thái của đối

tượng. Giống như trường hợp của phương pháp, tùy theo người viết mã, biến nội tại có thể chỉ được dùng bên trong các phương pháp của chính lớp đó, có thể cho phép các câu lệnh bên ngoài lớp, hay chỉ cho phép các lớp có quan hệ đặc biệt như là quan hệ lớp con, (và quan hệ bạn bè (friend) trong C++) được phép dùng tới nó (hay thay đổi giá trị của nó). Mỗi thuộc tính của một lớp còn được gọi là thành viên dữ liệu của lớp đó.

3.6 Đối tượng (object)

Các dữ liệu và chỉ thị được kết hợp vào một đơn vị đầy đủ tạo nên một đối tượng. Đơn vị này tương đương với một chương trình con và vì thế các đối tượng sẽ được chia thành hai bộ phận chính: phần các phương thức (method) và phần các thuộc tính (property). Trong thực tế, các phương thức của đối tượng là các hàm và các thuộc tính của nó là các biến, các tham số hay hằng nội tại của một đối tượng (hay nói cách khác tập hợp các dữ liệu nội tại tạo thành thuộc tính của đối tượng). Các phương thức là phương tiện để sử dụng một đối tượng trong khi các thuộc tính sẽ mô tả đối tượng có những tính chất gì.

Các phương thức và các thuộc tính thường gắn chặt với thực tế các đặc tính và sử dụng của một đối tượng.

Trong thực tế, các đối tượng thường được trừu tượng hóa qua việc định nghĩa của các lớp (class).

Tập hợp các giá trị hiện có của các thuộc tính tạo nên trạng thái của một đối tượng.

Mỗi phương thức hay mỗi dữ liệu nội tại cùng với các tính chất được định nghĩa (bởi người lập trình) được xem là một đặc tính riêng của

đối tượng. Nếu không có gì lầm lẫn thì tập hợp các đặc tính này gọi chung là đặc tính của đối tượng.

3.7 Thực thể (instance)

Thực thể hóa (instantiate) là quá trình khai báo để có một tên (có thể được xem như là một biến) trở thành một đối tượng từ một lớp nào đó.

Một lớp sau khi được tiến hành thực thể hóa để có một đối tượng cụ thể gọi là một thực thể. Hay nói ngược lại một thực thể là một đối tượng riêng lẽ của một lớp đã định trước. Như các biến thông thường, hai thực thể của cùng một lớp có thể có trạng thái nội tại khác nhau (xác định bởi các giá trị hiện có của các biến nội tại) và do đó hoàn toàn độc lập nhau nếu không có yêu cầu gì đặc biệt từ người lập trình.

3.8 Công cộng (public)

Công cộng là một tính chất được dùng để gán cho các phương pháp, các biến nội tại, hay các lớp mà khi khai báo thì người lập trình đã cho phép các câu lệnh bên ngoài cũng như các đối tượng khác được phép dùng đến nó.

Thí dụ: Trong C++ khai báo `public: int my_var;` thì biến `my_var` có hai tính chất là tính công cộng và là một `integer` cả hai tính chất này hợp thành đặc tính của biến `my_var` khiến nó có thể được xử dụng hay thay đổi giá trị của nó (bởi các câu lệnh) ở mọi nơi bên ngoài lẫn bên trong của lớp.

3.9 Riêng tư (private)

Riêng tư là sự thể hiện tính chất đóng mạnh nhất (của một đặc tính hay một lớp). Khi dùng tính chất này gán cho một biến, một phương pháp thì biến hay phương pháp đó chỉ có thể được sử dụng bên trong của lớp mà chúng được định nghĩa. Mọi nỗ lực dùng đến chúng từ bên ngoài qua các câu lệnh hay từ các lớp con sẽ bị phủ nhận hay bị lỗi.

3.10 Bảo tồn (protected)

Tùy theo ngôn ngữ, sẽ có vài điểm nhỏ khác nhau về cách hiểu tính chất này. Nhìn chung đây là tính chất mà mà khi dùng để áp dụng cho các phương pháp, các biến nội tại, hay các lớp thì chỉ có trong nội bộ của lớp đó hay các lớp con của nó (hay trong nội bộ một gói như trong Java) được phép gọi đến hay dùng đến các phương pháp, biến hay lớp đó.

So với tính chất riêng tư thì tính bảo tồn rộng rãi hơn về nghĩa chia sẻ dữ liệu hay chức năng. Nó cho phép một số trường hợp được dùng tới các đặc tính của một lớp (từ một lớp con chẳng hạn).

Lưu ý: Các tính chất cộng cộng, riêng tư và bảo tồn đôi khi còn được dùng để chỉ thị cho một lớp con cách thức kế thừa một lớp cha mẹ như trong C++.

3.11 Đa kế thừa (muliple inheritance)

Đây là một tính chất cho phép một lớp con có khả năng kế thừa trực tiếp cùng lúc nhiều lớp khác.

Vài điểm cần lưu ý khi viết mã dùng tính chất đa kế thừa:

Khi muốn có một sự kế thừa từ nhiều lớp phụ mẫu thì các lớp này cần phải độc lập và đặc biệt tên của các dữ liệu hay hàm cho phép kế thừa phải có tên khác nhau để tránh lỗi "ambiguilty". Bởi vì lúc đó phần

mềm chuyển dịch sẽ không thể xác định được là lớp con sẽ thừa kế tên nào của các lớp phụ mẫu.

Không phải ngôn ngữ OOP loại phân lớp nào cũng hỗ trợ cho tính chất này.

Ngoài các khái niệm trên, tùy theo ngôn ngữ, có thể sẽ có các chức năng OOP riêng biệt được cấp thêm vào.

4. Lập trình hướng đối tượng trong PHP

4.1 Khai báo lớp và thể hiện của lớp trong PHP

Như chúng ta đã biết, một lớp bao gồm các kiểu thuộc tính và phương thức. Trong PHP, chúng ta khai báo một lớp với cú pháp như sau:

```
class tên_lớp
{
// Danh sách các biến, hằng, lớp... (thuộc tính)
// Danh sách các hàm (phương thức)
}
```

Trong đó, các lớp được khai báo thông qua từ khoá class, các thuộc tính được khai báo dưới dạng các biến, còn các phương thức được xây dựng dưới dạng các hàm.

Các thuộc tính và các phương thức trong lập trình hướng đối tượng có thể được thiết lập những tính chất đặc biệt như : riêng tư (private), công cộng (public)... Các tính chất này thường được đặt trước các khai báo thuộc tính và phương thức.

Ví dụ đơn giản dưới đây thể hiện một lớp có tên là `ho_so` với các thuộc tính công cộng bao gồm `ho_ten, ngay_sinh`:

```
class hoso
{
    public $ho_ten;
    public $ngay_sinh;
}
```

Đoạn mã trên mới chỉ khai báo một lớp thực thể với hai biến là $ho_ten và $ngay_sinh. Bây giờ chúng ta sẽ khai báo một thể hiện của lớp trên.

Để khai báo một thể hiện của một lớp, ta dùng từ khoá new, tiếp đó là tên lớp:

```
$ten_thuc_the = new ten_lop;
```

Để truy cập vào từng thuộc tính hay phương thức của lớp, ta dùng toán tử → với cú pháp như sau:

```
$ten_thuc_the → ten_thuoc_tinh;
```

Ví dụ sau đây sẽ tạo ra một thực thể của một cá nhân có tên là Hoàng:

```
<HTML>
<BODY>
<?phpphp
class hoso
{
    public $ho_ten;
    public $ngay_sinh;
}
$hoang=new hoso;
$hoang -> ho_ten="Nguyễn Huy Hoàng";
$hoang -> ngay_sinh = "25/7/2003";
echo "Họ tên: " . $hoang -> ho_ten . ". Ngày sinh: " .
$hoang -> ngay_sinh;
?>
</BODY>
</HTML>
```

Bây giờ, chúng ta sẽ trang bị thêm một phương thức để thay đổi giá trị của các thuộc tính ho_ten và ngay_sinh. Để tham chiếu tới các phần tử trong chính bản thân lớp đối tượng, chúng ta sử dụng biến $this và toán tử tham chiếu →, theo sau đó là tên của phương thức hoặc thuộc tính:

```
<HTML>
<HEAD>
     <meta http-equiv="Content-Type"
                  content="text/html; charset=UTF-8" />
</HEAD>
<BODY>
<?php
class hoso
{
     public $ho_ten;
     public $ngay_sinh;
     public function lap_gia_tri($hoten,$ngaysinh)
     {
          $this->ho_ten = $hoten;
          $this->ngay_sinh=$ngaysinh;
     }
}
$hoang=new hoso;
$hoang->lap_gia_tri("Nguyễn Huy Hoàng","25/7/2003");
echo "Họ tên: " . $hoang->ho_ten . ". Ngày sinh: " .
$hoang->ngay_sinh;
?>
```

```
</BODY>
</HTML>
```

Chú ý rằng khi khai báo một lớp, chúng ta cũng có thể thiết lập những giá trị mặc định ban đầu cho tất cả các thành viên được tạo ra từ lớp đó. Ví dụ:

```
<HTML>
<HEAD>
     <meta http-equiv="Content-Type"
               content="text/html; charset=UTF-8" />
</HEAD>
<BODY>
<?php
class hoso
{
     public $ho_ten = "Nguyễn Huy Hoàng";
     public $ngay_sinh = "25/7/2003";
     public function lap_gia_tri($hoten,$ngaysinh)
     {
          $this->ho_ten = $hoten;
          $this->ngay_sinh=$ngaysinh;
     }
}
$hoang=new hoso;
echo "Họ tên: " . $hoang->ho_ten . ". Ngày sinh: " .
$hoang->ngay_sinh;
```

4.2 Cơ chế đóng kín và tính rõ ràng của các phần tử trong lớp

Như chúng ta đã biết, lập trình hướng đối tượng tập trung vào việc "đóng gói" các phương thức và thuộc tính của một đối tượng nào đó.

Trong lập trình hướng đối tượng, các thành viên trong một lớp cần phải được xác định xem chúng có thể được truy xuất từ đâu (tính rõ ràng). Có ba khả năng xảy ra:

- Chế độ truy xuất công cộng (`public`): Các thành viên nếu được thiết lập ở chế độ này sẽ được nhìn thấy và truy xuất ở mọi nơi trong chương trình.

- Chế độ truy xuất riêng tư (`private`): Các thành viên nếu được thiết lập ở chế độ này sẽ chỉ được nhìn thấy và truy xuất được ở bản thân lớp định nghĩa thành viên đó.

- Chế độ bảo vệ (`protected`): Chế độ này sẽ được dùng để giới hạn truy cập tới các lớp được thừa kế và bản thân lớp định nghĩa thành viên đó.

Trong PHP, tất cả các thành viên của một lớp đều phải được khai báo tính rõ ràng với các từ khoá tương ứng là `public, protected` và `private`.

Ví dụ:

```
<?phpphp
class MyClass
{
```

```
   public $public = 'Public';
   protected $protected = 'Protected';
   private $private = 'Private';

   public function printHello()
   {
       echo $this->public;
       echo $this->protected;
       echo $this->private;
   }
}
$obj = new MyClass();
echo $obj->public; // Làm việc tốt
echo $obj->protected;
//Gây lỗi do thuộc tính $protected đã bị đặt ở chế độ bảo vệ
echo $obj->private;
// Gây lỗi do thuộc tính $protected đã bị đặt ở chế độ riêng tư
$obj->printHello();
// Hoạt động bình thường, do các thuộc tính được triệu gọi
bên trong một phương thức nằm trong lớp.
class MyClass2 extends MyClass
{
   protected $protected = 'Protected2';
   function printHello()
   {
       echo $this->public;
       echo $this->protected;
       echo $this->private;
   }
}
$obj2 = new MyClass2();
echo $obj->public; // Chạy tốt
echo $obj2->private; // Chưa được định nghĩa
echo $obj2->protected; // Gây lỗi
$obj2-
>printHello(); // Hiển thị Public, Protected2, không Pr
```

```
ivate
?>
```

Nếu như chúng ta không đặt các từ khoá xác định tính rõ ràng của các thành viên, theo mặc định chúng sẽ ở chế độ public.

4.3 Kế thừa lớp trong PHP

Ở phần trên, các bạn đã biết đến khái niệm kế thừa giữa các lớp, đó là khả năng một lớp có thể được kế thừa các thành phần dữ liệu cũng như phương thức của một lớp nào đó. Lớp thừa kế những phương thức và thuộc tính của lớp khác được gọi là lớp con, còn lớp được kế thừa được gọi là lớp cha.

Trong PHP, một lớp có thể thừa kế các phương thức cũng như các thuộc tính của một lớp khác bằng cách sử dụng từ khoá `extends` trong khi khai báo tên lớp.

Ví dụ dưới đây thực hiện việc mở rộng lớp `hoso` ở trên thêm một số thuộc tính mới bằng cách kế thừa từ lớp `hoso`:

```
<HTML>
<HEAD>
     <meta http-equiv="Content-Type"
                  content="text/html; charset=UTF-8" />
</HEAD>
<BODY>
<?php
class hoso
{
     public $ho_ten = "Nguyễn Huy Hoàng";
     public $ngay_sinh = "25/7/2003";
     public function lap_gia_tri($hoten,$ngaysinh)
     {
          $this->ho_ten = $hoten;
          $this->ngay_sinh=$ngaysinh;
     }
```

```
}
class hoso2 extends hoso
{
     public $noi_sinh="Thanh Hoá";
     public function in_hoso()
     {
          echo "Họ tên: " . $this->ho_ten . ". Ngày
           sinh: " . $this->ngay_sinh . " . Nơi sinh: "
            . $this->noi_sinh;
     }
}
$hoang=new hoso2;
$hoang->in_hoso();
?>
</BODY>
</HTML>
```

Các phương thức và thành viên thừa kế từ lớp cha sang lớp con có thể bị ghi đè nếu như lớp cha không định nghĩa chúng dưới dạng các phương thức cuối cùng với từ khoá `final`.

Kỹ thuật ***ghi đè*** cho phép chúng ta có thể định nghĩa lại các hàm trong lớp cha bằng các hàm cùng tên trong lớp con nhưng hai hàm này có hai chức năng hoàn toàn khác nhau. Nhiều người gọi kỹ thuật này là "đa hình" do chúng tạo ra nhiều hình thái khác nhau ở các lớp thừa kế.

Chúng ta có thể truy cập đến các phương thức hay thành viên đã bị ghi đè bằng cách tham chiếu chúng với từ khoá parent và toán tử tham chiếu `::`, tiếp theo là phương thức hay thành viên cần tham chiếu (`parent::ten_phuong_thuc`).

Ví dụ dưới đây mô tả cách thức ghi đè:

```
<BODY>
<?php
class hoso
{
     public $ho_ten = "Nguyễn Huy Hoàng";
     public $ngay_sinh = "25/7/2003";
     public function in_hoso()
     {
          echo "Họ tên:" . $this->ho_ten . ". Ngày sinh:
                                   " . $this->ngay_sinh;

     }
}
class hoso2 extends hoso
{
     public $noi_sinh="Thanh Hoá";
     public function in_hoso()
     {
          parent::in_hoso();
          echo ". Nơi sinh: " . $this->noi_sinh;
     }
}
$hoang=new hoso2;
$hoang->in_hoso();
?>
</BODY>
</HTML>
```

Như chúng ta đã thấy, đối tượng `$hoang` thuộc lớp `hoso2` đã được kế thừa mọi thứ ở lớp `hoso`. Trong lớp `hoso2`, chúng ta đã định nghĩa một hàm trùng tên với một hàm đã có sẵn trong lớp `hoso` (hàm `in_hoso()`). Hàm `in_hoso()` của lớp `hoso2` đã ghi đè lên hàm

`in_hoso()` của lớp `hoso`, nhưng trong bản thân nó lại có thể triệu gọi trực tiếp đến hàm `in_hoso()` trong lớp cha (`hoso`).

5. Tổng kết

Lập trình hướng đối tượng là kỹ thuật thường sử dụng trong công nghệ phần mềm hiện đại. lập trình hướng đối tượng (Object Oriented Programming OOP) là sử dụng lớp (classes), quan hệ (Relationships), thuộc tính (Properties) và phương thức (method) của đối tượng (Object) trong hệ thống để phát triển phần mềm theo cấu trúc kế thừa và sử dụng lại mã hiệu quả.

Khai báo lớp là một trong những chức năng để đóng gói các chức năng ứng dụng của chúng ta. Ngoài ra, chúng ta có thể xây dựng các thuộc tính cũng như các phương thức nhằm sử dụng chung cho toàn ứng dụng hay những module có thể sử dụng lại nhiều lần.

Trong chương này, chúng ta đã học cách định nghĩa một lớp, lớp kế thừa lớp, cách gọi thuộc tính và phương thức của lớp trong PHP.

Câu hỏi trắc nghiệm kết chương

Trả lời các câu hỏi sau:

1. Để khai báo một lớp trong PHP chúng ta sử dụng câu lệnh?
   a. Classes
   b. Creater class
   c. Class
   d. Creater classes
2. Câu lệnh chính xác để tạo ra một thuộc tính "a" trong lớp?
   a. $a;
   b. var a;
   c. int a;
3. Câu lệnh chính xác để khai báo lớp B kế thừa từ lớp A?
   a. Class A extend B
   b. Class A extends class B
   c. Class A extends B
   d. A extend B
4. Các thành viên nếu được thiết lập ở chế độ ..... sẽ chỉ được nhìn thấy và truy xuất được ở bản thân lớp định nghĩa thành viên đó.
5. Chế độ ..... sẽ được dùng để giới hạn truy cập tới các lớp được thừa kế và bản thân lớp định nghĩa thành viên đó.