บทที่ 11
การเขียนโปรแกรมเชิงวัตถุ

ในบทนี้จะกล่าวถึงการเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming) ซึ่งแม้ว่าในความเป็นจริงแล้ว ไม่จำเป็นต้องมีความรู้เกี่ยวกับเชิงวัตถุเลย ก็สามารถสร้างเว็บด้วยภาษา PHP ได้ แต่ในปัจจุบันมักจะมีผู้สร้างส่วนที่มาขยายขีดความสามารถของภาษา PHP (หรือ Extension) ใหม่ๆ เพิ่มขึ้นมาเรื่อยๆ โดย Extension เหล่านี้ มักถูกสร้างในรูปแบบของคลาส ดังนั้นจึงควรมีพื้นฐานความรู้เกี่ยวกับการเขียนโปรแกรมเชิงวัตถุเพื่อจะนำไปใช้กับการเขียนโปรแกรมในบางเรื่องได้

11.1 คลาส

โดยปกติแล้วภาษา PHP จะเป็นภาษาแบบ Procedural โดยไม่ได้อยู่ภายใต้ข้อกำหนด หรือโครงสร้างแบบเชิงวัตถุ จะเห็นได้ว่าภาษา PHP นั้นประกอบด้วยฟังก์ชันต่างๆ เป็นจำนวนมาก และก็สามารถใช้ฟังก์ชันเหล่านี้ได้โดยตรง และตั้งแต่ PHP4 เป็นต้นมา ได้พัฒนาให้ภาษา PHP รองรับการเขียนโปรแกรมทางด้านเชิงวัตถุเพิ่มเข้ามา โดยรูปแบบของการเขียนโปรแกรมเชิงวัตถุนั้นจะช่วยให้สามารถรวบรวมการกระทำที่เกี่ยวข้องกับเรื่องใดเรื่องหนึ่งให้มารวมเป็นหน่วยเดียวกัน หรือที่เรียกว่า "ออบเจ็กต์" นั่นเอง โดยภายในตัวออบเจ็กต์นั้นก็จะมีองค์ประกอบหลายอย่าง เช่น เมธอด (Method) ฟิลด์ (Field) พร็อปเพอร์ตี้ (Property) เป็นต้น มักจะเรียกสิ่งที่อยู่ในออบเจ็กต์ว่า "สมาชิกของออบเจ็กต์" และการเรียกใช้งานสมาชิกเหล่านี้ก็จะต้องทำผ่านออบเจ็กต์เท่านั้น ดังนั้นการเขียนโปรแกรมเชิงวัตถุเป็นวิธีการเรียกใช้การทำงานของออบเจ็ก จะเรียกว่า "คลาส (Class)" ดังนั้นจากที่ได้กล่าวมาทั้งหมดก็พอสรุปได้ว่าคลาส คือ โครงร่างที่ใช้ในการกำหนดองค์ประกอบ และวิธีการทำงานของออบเจ็กต์

ก่อนที่จะเรียนรู้เกี่ยวกับออบเจ็กต์ และองค์ประกอบภายในออบเจ็กต์ ต้องเริ่มศึกษาจากรูปแบบของคลาสก่อน โดยรูปแบบทั่วไปของคลาส มีดังนี้

รูปแบบ

```
Class ชื่อคลาส {
        ....
        ....
}
```

โดยทั่วไปนิยมตั้งชื่อคลาสขึ้นต้นด้วยอักษรตัวพิมพ์ใหญ่ (ไว้เป็นเพียงจุดสังเกตุเท่านั้น เพราะจริงๆ แล้วไม่จำเป็น) ส่วนข้อกำหนดอื่นๆ ในการตั้งชื่อจะคล้ายกับการตั้งชื่อตัวแปร เช่น

```
Class MyClass {
        ....
        ....
}
```

11.2 การนำคลาสมาใช้ในสคริปต์ PHP

เนื่องจากภาษา PHP ไม่ได้มีโครงสร้างแบบเชิงวัตถุอย่างแท้จริง การนำคลาสที่สร้างขึ้นมาใช้ก็จะใช้รูปแบบเดียวกับการนำเข้าไฟล์โดยทั่วๆ ไป มีแนวทางดังนี้

11.2.1 สร้างเอกสารเว็บเพจ ชนิด PHP

11.2.2 เขียนคลาสที่จะสร้างลงไปเหมือนการเขียนสคริปต์ PHP ปกติ เช่น

```
<?php
        Class PHPMySQL {
                ....
                ....
        }
?>
```

11.2.3 การบันทึกให้กำหนดชื่อเหมือนกับไฟล์เว็บเพจทั่วไป แต่ให้มีส่วนขยายเป็น .php เช่น MyClass.php หรือบางคนอาจต้องการให้ต่างจากไฟล์เว็บเพจทั่วไป อาจกำหนดชื่อในรูปแบบ ชื่อไฟล์.class.php เช่น สมมติคลาสชื่อ "PHPMySQL" ก็บันทึกไฟล์นี้ด้วยชื่อ "PHPMySQL.class.php" เป็นต้น

เมื่อจะนำคลาสนี้ไปใช้ในเว็บเพจใด ก็ใช้ฟังก์ชัน include ( ) หรือ require ( ) เพื่อนำคลาสเข้าไปใช้งานร่วมกับสคริปต์อื่นๆ

```
<?php
        include ("PHPMySQL.class.php");
        ....
        ....
?>
```

11.3 ออบเจ็กต์ และอินสแตนซ์

คลาสที่สร้างขึ้นเป็นเพียงข้อกำหนดหรือวิธีในการประมวลผลข้อมูลเท่านั้น แต่ยังไม่สามารถนำคลาสไปใช้งานได้ จนกว่าคลาสนั้นจะถูกสร้างเป็นออบเจ็กต์เสียก่อน ดังนั้นคลาสกับออบเจ็กต์ก็คือ สิ่งเดียวกันเพียงแต่อยู่คนละสถานภาพ ด้วยเหตุนี้จึงเรียกออบเจ็กต์ว่าเป็นอินสแตนซ์ (Instance) ของคลาส หรือสิ่งที่ใช้แทนคลาสนั่นเอง การสร้างออบเจ็กต์มีรูปแบบดังนี้

รูปแบบ

```
$ชื่อของอินสแตนซ์ = new ชื่อคลาส ( );
```

ตัวอย่างที่ 11.1 การสร้างออบเจ็กต์

```
$mycls = new MyClass ( );
```

ขั้นตอนในการสร้างออบเจ็กต์ของคลาสนี้เรียกว่า Instantiate หลังจากออบเจ็กต์ถูกสร้างขึ้นแล้วก็จะมีความสามารถตามที่ระบุไว้ในคลาสทุกประการโดยคลาสแต่ละคลาสสามารถนำไปสร้างเป็นออบเจ็กต์ได้มากกว่า 1 ออบเจ็กต์ มีตัวอย่างดังนี้

ตัวอย่างที่ 11.2 การสร้างเป็นออบเจ็กต์ได้มากกว่า 1 ออบเจ็กต์

```
<?php
        $myclass1 = new MyClass ( );
        $myclass2 = new MyClass ( );
?>
```

11.4 เมธอด

เมธอด (Method) เป็นองค์ประกอบของคลาสสำหรับการกระทำอย่างใดอย่างหนึ่ง โดยทั่วไปแล้วคลาสจะต้องประกอบด้วยเมธอดอย่างน้อย 1 เมธอดเสมอ รายละเอียดเกี่ยวเมธอดมีดังนี้

11.4.1 การสร้างเมธอด

ความจริงแล้วเมธอดก็คือ ฟังก์ชันของคลาสนั่นเอง ดังนั้นรูปแบบ และวิธีการสร้างเมธอดก็เหมือนกันกับการสร้างฟังก์ชันธรรมดานั่นเอง และภายในคลาสหนึ่งๆ จะมีกี่เมธอดก็ได้ ตัวอย่างลักษณะของการสร้างเมธอด ตัวอย่างดังนี้

ตัวอย่างที่ 11.3 ลักษณะของการสร้างเมธอด

```
1   <?php
2       class Circle {
3           function area ($radius) return pi ( ) * $radius * $radius;
4           function perimeter ($radius) return 2 * pi ( ) * $radius;
5       }
6   ?>
```

11.4.2 การเรียกใช้เมธอด

การเรียกใช้เมธอดจะใช้โอเปอเรเตอร์ -> ในการระบุเมธอดที่ต้องการเรียกใช้งาน โดยหากเป็นการเรียกใช้ภายในคลาสเดียวกัน (เรียกใช้เมธอดหนึ่งจากอีกเมธอดหนึ่ง) ก็ใช้รูปแบบดังนี้

```
$this -> เมธอด (อาร์กิวเมนต์)
```

เช่น สมมติว่าต้องการเรียกใช้เมธอด area ( ) ภายในคลาส Circle ก็ทำดังนี้

ตัวอย่างที่ 11.4 ตัวอย่างการเรียกใช้เมธอด

```
$a = $this -> area (10);
```

หากเป็นการเรียกจากนอกคลาส ก็ต้องเรียกผ่านอินสแตนซ์ในรูปแบบดังนี้

```
$อินสแตนซ์ -> เมธอด (อาร์กิวเมนต์)
```

เช่น สมมติว่าต้องการเรียกใช้เมธอด area ( ) ของคลาส Circle

ตัวอย่างที่ 11.5 ตัวอย่างการเรียกใช้เมธอด area ( ) ของคลาส Circle

```
1   <?php
2       $cir = new Circle ( );
3       echo $cir -> area(10);
4   ?>
```

จากตัวอย่างที่ 11.5 ตัวอย่างการเรียกใช้เมธอด area ( ) ของคลาส Circle เป็นการทดสอบการสร้าง และการใช้คลาส โดยเป็นคลาสง่ายๆ เกี่ยวกับการคำนวณเลข ทั้งนี้ต้องสร้างขึ้นมา 2 เว็บเพจ โดยเว็บเพจแรกเป็นคลาส ส่วนอีกเว็บเพจสำหรับทดสอบคลาส ดังนี้

ตัวอย่างที่ 11.6 ตัวอย่าง เป็นการทดสอบการสร้าง และการใช้คลาส โดยเป็นคลาสง่ายๆ

ชื่อไฟล์ calculator.class.php

```
1<?php
2   class Calculator {
3       function showResult ($num1, $num2, $op) {
4           if (!is_numeric ($num1) || !is_numeric ($num2)) {
5               echo "ตัวแปรที่รับมาไม่ใช่ตัวเลขไม่สามารถคำนวณได้ <br>";
6           } else if ( !in_array ($op, array ("+", "-" , "*", "/"))) {
7               echo "เครื่องหมายดำเนินการไม่ถูกต้อง <br>";
8           } else {
9               echo "$num1 $op $num2 = ";
10              eval ("echo $num1 $op $num2;");
11              // ฟังก์ชัน eval ( ) ใช้ประมวลผล คำสั่ง PHP ที่เขียนในแบบสตริง
12          }
13      }
```

```
14      }
15?>
```

ชื่อไฟล์ที่จะเรียกใช้งานคลาส index.php

```
1       <?php
2               include ("calculator.class.php");
3               $cal = new Calculator ( );
4               $cal -> showResult (10, 20, "*");
5       ?>
```

ผลลัพธ์

```
        10 * 20 = 200
```

11.4.3 การกำหนดค่าพารามิเตอร์เริ่มต้นโดยปริยาย (Default Parameter) ให้กับเมธอด

เนื่องจากเมธอดนั้นก็คือฟังก์ชัน และฟังก์ชันใน PHP สามารถกำหนดค่าเริ่มต้นโดยปริยายให้ได้ สามารถนำรูปแบบตรงนี้มาใช้เพื่อเป็นทางเลือกให้ผู้ใช้สามารถที่จะกำหนดพารามิเตอร์บางตัวหรือไม่ก็ได้ ทั้งนี้วิธีการต่างๆ ก็เหมือนกับฟังก์ชันที่ได้ศึกษามาแล้ว มีตัวอย่างดังนี้

ตัวอย่างที่ 11.7 การกำหนดค่าพารามิเตอร์เริ่มต้นโดยปริยายให้กับเมธอด

```
1       <?php
2               class Payment {
3                       function total ($quantity, $price, $vat=7) {
4                               return $quantity * $price * (1+$vat/100);
5                       }
6               }
7       ?>
```

จากคลาส Payment เมื่อเรียกใช้เมธอด total ( ) จะกำหนด %VAT หรือไม่ก็ได้ โดยหากไม่กำหนดก็จะใช้ค่าเริ่มต้นโดยปริยาย คือ 7% แต่หากต้องการใช้ค่าอื่นก็กำหนดลงไปโดยตรง มีตัวอย่างดังนี้

ตัวอย่างที่ 11.8 การเรียกใช้คลาส Payment

```
1       <?php
2               $p = new Payment();
3               $t1 = $p -> total (10, 20);                     //VAT 7%
```

```
4               $t2 = $p -> total (30, 40, 10);                 //VAT 10%
5       ?>
```

11.5 โมดิฟายเออร์ แบบ public และ private

โมดิฟายเออร์ (Modifier) เป็นการควบคุมการเข้าใช้งานสมาชิกของคลาส เช่น เมธอด และพร็อปเพอร์ตี้ทั้งนี้เนื่องจากตามข้อกำหนดของการเขียนโปรแกรมเชิงวัตถุนั้น สามารถที่จะเรียกใช้สมาชิกของคลาสหนึ่งจากภายนอกคลาสได้ แต่ภายในคลาสอาจมีข้อมูลบางอย่างที่มีความสำคัญ จึงจำเป็นต้องมีวิธีการที่จะจำกัดขอบเขตในการเข้าใช้งานข้อมูลเหล่านี้ โดยใช้คีย์เวิร์ดกลุ่มหนึ่งเรียกว่า โมดิฟายเออร์ สำหรับภาษา PHP มีโมดิฟายเออร์หลายตัว ในบทนี้จะกล่าวถึงเฉพาะโมดิฟายเออร์บางชนิดที่ควรรู้จักในเบื้องต้น ดังนี้

public ใช้ public เมื่อไม่ต้องการปกป้องสิทธิการเข้าใช้สมาชิกนั้น

private ใช้ private เมื่อไม่ต้องการให้เข้าถึงสมาชิกนั้นจากภายนอกคลาสได้ จึงมักใช้ private กับสมาชิกที่มีความสำคัญ และต้องการสงวนไว้ใช้เฉพาะภายในคลาสเองเท่านั้น

ลักษณะการนำไปใช้กับเมธอด มีตัวอย่างดังนี้

ตัวอย่างที่ 11.9 ตัวอย่างการกำหนดใช้โมดิฟายเออร์ให้กับเมธอด

```
1       <?php
2           class Cylinder {
3               private function CircleArea ($radius) {
4                       return pi ( ) * $radius * $radius;
5               }
6               public function volume ($radius, $height) {
7                       $base_area = $this -> circleArea ($radius);
8                       $volume = $base_area * $height;
9                       return $volume;
10              }
11          }
12      ?>
```

จากตัวอย่างที่ 11.9 ตัวอย่างการกำหนดใช้โมดิฟายเออร์ให้กับเมธอด อธิบายดังนี้

บรรทัดที่ 3 กำหนดโมดิฟายเออร์ ชนิด private คือ ไม่ต้องการให้เข้าถึงสมาชิกนั้นจากภายนอกคลาส และต้องการสงวนไว้ใช้เฉพาะภายในคลาสเท่านั้น

บรรทัดที่ 6 กำหนดโมดิฟายเออร์ ชนิด public คือ อนุญาตให้เข้าถึงสมาชิกจากทุกๆ ส่วนของสคริปต์

ตัวอย่างที่ 11.10 ลักษณะผลการเรียกใช้ใช้เมธอดแบบ private และ public

```
1    <?php
2    include ("Cylinder.class.php");
3    $cyl = new Cylinder ( );
4    echo $cyl  -> circleArea (10);
5    echo $cyl -> volume (10, 20);
6    ?>
```

จากตัวอย่างที่ 11.10 แสดงลักษณะผลการเรียกใช้ใช้เมธอดแบบ private และ public อธิบายเพิ่มเติม ดังนี้

บรรทัดที่ 4 ไม่สามารถเรียกใช้งานได้ เนื่องจาก circleArea( ) เป็น private method

บรรทัดที่ 5 สามารถใช้งานได้เพราะ volume ( ) เป็น public method

11.6 พร็อปเพอร์ตี้ และฟิลด์

พร็อปเพอร์ตี้ (Property) คือ ข้อมูลประจำตัวของคลาส ส่วนใหญ่แล้วมักเป็นข้อมูลที่ต้องใช้ร่วมกันในหลายๆ เมธอด หรือไม่ก็เป็นข้อมูลเบื้องต้นที่จำเป็นต่อการทำงานของคลาส เช่น หากเป็นวงกลมข้อมูลที่จำเป็นต้องใช้เสมอ คือ รัศมี เป็นต้น ถึงแม้ว่าข้อมูลนี้จะรับผ่านพารามิเตอร์ของเมธอดได้ แต่ก็จะต้องระบุข้อมูลนี้ทุกๆ ครั้ง ทั้งนี้การสร้างพร็อปเพอร์ตี้มีรูปแบบดังนี้

```
public $ชื่อพร็อปเพอร์ตี้
```

สำหรับโมดิฟายเออร์ โดยทั่วไปจะกำหนดเป็น public เนื่องจากในกรณีของพร็อปเพอร์ตี้นั้นจะต้องสามารถเข้าถึงจากภายนอกคลาสได้ มีตัวอย่างดังนี้

```
class Cylinder {
        public $radius;
        public $height;
        ....
        ....
}
```

หมายเหตุ

การเขียนโปรแกรมเชิงวัตถุสำหรับภาษา PHP ในยุคแรกการกำหนดพร็อปเพอร์ตี้จะใช้คีย์เวิร์ด "var" ตามด้วยชื่อพร็อปเพอร์ตี้ เช่น var $radius; แต่ในปัจจุบันนิยมกำหนดโดยโมดิฟายเออร์ "public" แทน แต่หากกำหนดโมดิฟายเออร์เป็น "private" มักเรียกว่า "ฟิลด์"

เนื่องจากพร็อปเพอร์ตี้ คือ ข้อมูลของคลาส ดังนั้นจึงอาจถูกเรียกใช้จากทั้งภายใน และภายนอกคลาส หากเป็นการเรียกภายในคลาสจะต้องระบุผ่านคีย์เวิร์ดพิเศษ $this และหากเป็นการอ้างอิงภายนอกคลาสจะต้องระบุผ่านอินสแตนซ์ ตามรูปแบบดังนี้

เรียกใช้ภายในคลาส: $this -> ชื่อพร็อปเพอร์ตี้
เรียกใช้ภายนอกคลาส: $อินสแตนซ์ -> ชื่อพร็อปเพอร์ตี้

ตัวอย่างที่ 11.11 การเรียกใช้ภายในคลาสผ่านคีย์เวิร์ดพิเศษ $this

```
1   <?php
2       class Cylinder {
3           public $radius = 0;
4           public $hight = 0;
5           private function CircleArea ( ) {
6               return pi ( ) * $this -> circleArea ($this -> radius);
7           }
8           public function volume ( ) {
9               $base_area = $this -> circleArea($this -> radius);
10              $volime = $base_area * $this -> hight;
11              return $volume;
12          }
13      }
14  ?>
```

ตัวอย่างที่ 11.12 การใช้พร็อปเพอร์ตี้จากภายนอกคลาสผ่านอินสแตนซ์

```
1   <?php
2       include ("Cylinder.class.php");
3       $cy1 = new Cylinder ( );
4       $cy1 -> redius = 10;
5       $cy2 -> redius = 20;
```

```
6           echo $cy1 -> volume();
7       ?>
```

ฟิลด์ (Field) ตามรูปแบบการเขียนโปรแกรมเชิงวัตถุของภาษา PHP จะมีลักษณะคล้ายกับพร็อปเพอร์ตี้ คือ เป็นข้อมูลของคลาสแต่ละข้อมูลแบบฟิลด์นี้มักจะใช้เฉพาะภายในคลาสเท่านั้น โดยไม่อนุญาตให้เข้าถึงจากนอกคลาสได้ ดังนั้นจึงต้องกำหนดโมดิฟายเออร์เป็น "private" ขณะที่พร็อปเพอร์ตี้นั้นสามารถเข้าถึงจากนอกคลาสได้ โดยทั่วไปแล้วมักใช้ฟิลด์ร่วมกับคอนสตรักเตอร์เป็นหลัก เพื่อรับข้อมูลที่จำเป็นเบื้องต้นจากนอกคลาสเข้ามา ตัวอย่างลักษณะการกำหนดฟิลด์ดังนี้

```
class Cylinder {
    private $redius = 0;
    private $hight = 0;
    ....
}
```

11.7 ค่าคงที่

ค่าคงที่ (Constant) คือ ค่าที่กำหนดไว้อย่างตายตัว เหมือนกับค่าคงที่ที่ใช้ในการเขียนสคริปต์ปกตินั่นเอง โดยรูปแบบการกำหนดดังนี้

```
const ชื่อค่าคงที่ = ค่าที่กำหนด;
```

ค่าคงที่นี้ ไม่ต้องกำหนดโมดิฟายเออร์ใดๆ และชื่อค่าคงที่ก็ไม่มีเครื่องหมาย $ นำหน้าและวิธีการเรียกใช้ค่าคงที่ก็จะต่างไปจากการเรียกใช้พร็อปเพอร์ตี้ โดยสามารถเรียกผ่านชื่อคลาสได้โดยตรงโดยไม่ต้องเรียกผ่านอินสแตนซ์ของคลาส มีรูปแบบดังนี้

```
ชื่อคลาส :: ชื่อค่าคงที่
```

ตัวอย่างที่ 11.13 การกำหนดค่าคงที่ภายในคลาส

```
1   <?php
2       class Circle {
3           const PI = 3.14;
4           public function area($radius) {
5               return Circle :: PI * $radius * $radius;
6           }
7       }
8   ?>
```

11.8 คอนสตรักเตอร์

คลาสบางคลาสก่อนที่จะทำการประมวลผลใดๆ ได้ จำเป็นต้องอาศัยข้อมูลพื้นฐานบางอย่างเสียก่อน หากขาดข้อมูลเหล่านี้ไปแล้ว คลาสจะทำงานไม่ได้เลย เช่น คลาสวงกลม หากขาดข้อมูลที่เป็นค่ารัศมี ก็ไม่อาจคำนวณหาค่าพื้นที่ และเส้นรอบวงได้ และผู้ใช้คลาสก็อาจไม่ทราบว่าต้องกำหนดข้อมูลพื้นฐานอะไรให้แก่คลาสนั้นบ้าง ด้วยเหตุนี้คลาสจึงต้องมีเมธอดพิเศษที่เรียกว่าคอนสตรักเตอร์ (Constructor) เพื่อใช้ในการกำหนดข้อมูลที่จำเป็นเริ่มแรกให้แก่คลาส โดยลักษณะที่สำคัญของคอนสตรักเตอร์ ดังนี้

11.8.1 คอนสตรักเตอร์เป็นเมธอดพิเศษที่ใช้ในการรับข้อมูลเริ่มต้นเข้ามายังคลาสในรูปแบบพารามิเตอร์ แต่คอนสตรักเตอร์ไม่จำเป็นต้องมีพารามิเตอร์เสมอไป

11.8.2 เมธอดที่เป็นคอนสตรักเตอร์จะถูกเรียกขึ้นมาทำงานโดยอัตโนมัติในขั้นตอนการสร้างออบเจ็กต์

11.8.3 สำหรับ PHP คอนสตรักเตอร์จะถูกกำหนดตายตัวไปเลยว่าต้องใช้ชื่อ __construct ( ) เท่านั้น (หน้า construct มีเครื่องหมาย underscore 2 ขีด) ส่วนโมดิฟายเออร์ไม่จำเป็นต้องกำหนด หรือกำหนดเป็น public เท่านั้น

11.8.4 พารามิเตอร์ที่คอนสตรักเตอร์รับเข้ามาจะต้องนำไปเก็บไว้ในตัวแปรฟิลด์ เพื่อรอการนำไปใช้งานเมธอดอื่นๆ ต่อไป

ตัวอย่างที่ 11.14 การสร้างคอนสตรักเตอร์ของคลาส

```
1   <?php
2       class Circle{
3           private $redius = 0;
4           function __construct($radius) {
5               $this -> radius = $radius;      //นำค่าพารามิเตอร์ไปเก็บไว้ในตัวแปรฟิลด์
6           }
7       }
8   ?>
```

11.8.5 การเรียกใช้คอนสตรักเตอร์ เมธอดที่เป็นคอนสตรักเตอร์จะถูกเรียกขึ้นมาทำงานโดยอัตโนมัติในขั้นตอนการสร้างออบเจ็กต์ ทั้งนี้หากคอนสตรักนั้นมีพารามิเตอร์ ต้องระบุลงไปด้วย เช่น

```
$cir = new Circle(10);          //กำหนดรัศมีเท่ากับ 10
```

11.9 โมดิฟายเออร์ Static

โมดิฟายเออร์แบบ static ใช้เมื่อต้องการให้เกิดการอ้างอิงข้อมูลเดียวกันของทุกออบเจ็กต์ที่สร้างมาจากคลาสเดียวกัน ทำให้เมื่อเปลี่ยนแปลงข้อมูลที่ออบเจ็กต์ตัวใด จะส่งผลให้ข้อมูลอันเดียวกันในออบเจ็กต์อื่นๆ เปลี่ยนตามไปด้วย เพราะเป็นการอ้างอิงค่าเดียวกัน ทั้งนี้ static ใช้ได้กับทั้งพร็อปเพอร์ตี้และเมธอด โดยแต่ละคลาสจะมีสมาชิกที่เป็นแบบ static มีดังต่อไปนี้

```
โมดิฟายเออร์ static $พร็อปเพอร์ตี้;
โมดิฟายเออร์ static function เมธอด ( ) {
        .
        .
}
```

ตัวอย่างที่ 11.15 โมดิฟายเออร์ Static

```
private static $field;
public static $property;

public static function method ( ) {
        .
        .
}
```

11.9.1 การเรียกใช้งานสมาชิกแบบ static ภายในคลาส

ภายในคลาสเมื่อต้องการเรียกใช้สมาชิกแบบ static จะต้องระบุด้วยคำว่า self ตามด้วยโอเปอเรเตอร์ :: และชื่อพร็อปเพอร์ตี้หรือเมธอด ตามรูปแบบคือ

```
        self :: $พร็อปเพอร์ตี้
หรือ    self :: เมธอด ( )
```

ตัวอย่างที่ 11.16 การเรียกใช้งานสมาชิกแบบ static ภายในคลาส

```
self :: $x          //พร็อปเพอร์ตี้
self :: f1 ( );     //เมธอด
```

11.9.2 การเรียกใช้งานสมาชิกแบบ static จากนอกคลาส

ส่วนภายนอกคลาสการระบุถึงสมาชิกที่เป็น static จะระบุผ่านชื่อคลาสโดยตรง หรือระบุผ่านอินสแตนซ์ก็ได้ ในรูปแบบดังนี้

```
คลาส :: เมธอด
คลาส :: พร็อปเพอร์ตี้
```

หรือเรียกใช้ผ่านอินสแตนซ์

```
อินสแตนซ์ -> เมธอด
อินสแตนซ์ -> พร็อปเพอร์ตี้
```

นอกจากนี้ในการเรียกใช้สมาชิกแบบ static มีข้อกำหนดที่สำคัญดังนี้

1) สามารถเรียกใช้สมาชิกแบบ static ได้โดยตรง โดยไม่จำเป็นต้องเรียกผ่านอินสแตนซ์

2) เฉพาะเมธอดแบบ static เท่านั้นที่สามารถเรียกพร็อปเพอร์ตี้แบบ static ได้

ตัวอย่างการทดสอบการกำหนดสมาชิกแบบ static โดยประกอบด้วย 2 เว็บเพจ สำหรับคลาสและทดสอบการใช้คลาส

ตัวอย่างที่ 11.17 การกำหนดคลาสภายในไฟล์ Visitor.class.php

```
1   <?php
2       class Visitor {
3           private static $num_visitor = 0;
4           function _construct ( ) {
5               self :: $num_visitor++;
6           }
7           public static function getVisitor ( ) {
8               return self :: $num_visitor;
9           }
10      }
11  ?>
```

ตัวอย่างที่ 11.18 การเรียกใช้คลาสในไฟล์ index.php

```
1   <?php
2       include ("Visitor.class.php");
3       $v1 = new Visitor();
4       echo "Visitors :" . $v1 -> getVisitor ( ) . "<br>";
5       $v2 = new Visitor ( );
6       echo "Visitors :" . $v2 -> getVisitor ( ) . "<br>";
7       $v3 = new Visitor ( );
8       echo "Visitors :" . $v3 -> getVisitor ( ) . "<br>";
```

```
9       ?>
```

ตัวอย่างนี้สร้างออบเจ็กต์ของคลาส Visitor จำนวน 3 ครั้ง โดยที่คอนสตรักเตอร์ได้เพิ่มค่าให้แก่พร็อปเพอร์ตี้ $num_visitor เป็นแบบ static ไปอีก 1 และเมื่อเรียกเมธอด getVisitor ( ) เมธอดนี้จะไปอ่านข้อมูลจาก $num_visitor จะพบว่าค่าจะเพิ่มขึ้นเรื่อยๆ แสดงว่าพร็อปเพอร์ตี้ $num_visitor มีการอ้างอิงข้อมูลเดียวกัน จึงทำให้ค่าเพิ่มขึ้นจากค่าเดิมได้ไม่เช่นนั้นแล้วค่าจะต้องเป็น 1 ทั้งหมด

11.10 การสืบทอด

การสืบทอด (Inheritance) หมายถึง การที่คลาสหนึ่งถูกสร้างขึ้นโดยการสืบทอดมาจากอีกคลาสหนึ่ง ผลจากการสืบทอดจะทำให้คลาสที่เป็นผู้สืบทอดมีองค์ประกอบทั้งหมดเหมือนกับคลาสต้นแบบ เรียกคลาสที่เป็นคลาสต้นแบบว่า Parent Class ส่วนคลาสที่เป็นผู้ทำการสืบทอดมาเรียกว่า Sub Class ทั้งนี้การสืบทอดจะใช้คีย์เวิร์ด extends เป็นตัวกำหนด เช่น

```
class Shape {
        ....
        ....
} class Rectangle extends Shape {          //Rectangle สืบทอดมาจาก Shape
        ....
        ....
}
```

การสืบทอดมักทำเพื่อความสามารถบางอย่างเข้าไปใน Sub Class โดยนำความสามารถเดิมบางอย่างของ Parent Class มาใช้ เนื่องจากภายหลังการสืบทอดจะทำให้สามารถเรียกสมาชิกต่างๆ ของ Parent Class จาก Sub Class ได้เหมือนกับคลาสเดียวกัน ยกเว้นสมาชิกที่มีโมดิฟายเออร์เป็น private มีตัวอย่างดังนี้

ตัวอย่างที่ 11.19 การสืบทอดคลาส

```
1       <?php
2           class circle {
3               function circle_area ($radius) {
4                       return pi ( ) * pow ($radius, 2);
5               }
6           }
7           class Cylinder extends Circle {
8               function volume ($radius, $height) {
```

```
9                       $base_area = $this -> circle_area ($radius);   //ปริมาตร=พื้นที่ฐาน*สูง
10                      return $base_area * $height;
11              }
12          }
13          $cyl = new Cylinder ( );
14          echo $cyl -> volume (10, 20);
15          echo $cyl -> circle_area (30);
16      ?>
```

จากโค้ดตัวอย่างนี้จะพบว่าคลาส Cylinder นั้นสืบทอดมาจาก Circle และได้เพิ่มเมธอด volume ( ) เข้ามาในคลาส Cylinder จึงสามารถเข้าถึงเมธอด circle_area ( ) ของคลาส Circle จากคลาส Cylinder ได้โดยใช้ $this -> เหมือนกับอยู่ในคลาสเดียวกัน รวมถึงการเข้าถึงผ่านทางอินสแตนซ์ได้เช่นกัน

แต่ทั่วไปแล้วคลาสมักต้องมีคอนสตรักเตอร์ และกรณีการสืบทอด Sub Class สามารถมีคอนสตรักเตอร์เป็นของตนเองที่ต่างไปจาก Parent Class ได้ และสามารถเข้าถึงคอนสตรักเตอร์ของ Parent Class ด้วยการอ้างอิงในรูปแบบดังนี้

```
parent :: _construct (อาร์กิวเมนต์)
```

จากการสืบทอดระหว่างคลาส Circle/Cylinder ที่ผ่านมา สามารถแก้ไขให้ถูกต้องตามแนวทางการเขียนโปรแกรมเชิงวัตถุมากยิ่งขึ้นได้ดังนี้

ตัวอย่างที่ 11.20 การสืบทอดระหว่างคลาส

```
<?php
        class Circle {
                private $radius = 0;
                function _construct($radius) {
                        $this -> radius = $radius;
                }
                function area ( ) {
                        return pi ( ) * pow ($this -> radius, 2);
                }
        }
        class Cylinder extends Circle {
                private $height = 0;
```

```
            function _construct ($radius, $height) {
                $this -> height = $height;
                private :: _construct($radius); //เรียกคอนสตรักเตอร์ของคลาส Circle
            }
            function volume ( ) {
                $base_area = $this -> area ( );
                return $base_area * $this -> height;
            }
        }
//-------------------------------------------------------------------------------------------
        $cy = new Cylinder (10, 20);
        echo $cy -> volume();
?>
```

สรุป

หลักการของการเขียนโปรแกรมเชิงวัตถุ เป็นการสร้างความสัมพันธ์ระหว่างฟังก์ชันการทำงานที่อยู่ภายในโปรแกรมให้อยู่ในรูปของวัตถุ โดยที่แต่ละวัตถุ จะประกอบไปด้วยการกระทำที่เรียกว่าเมธอด และคุณสมบัติของวัตถุเรียกว่าพร็อปเพอร์ตี้ ที่มีลักษณะคล้ายกันจะถูกจัดเป็นกลุ่มที่เรียกว่าคลาส การเขียนโปรแกรมเชิงวัตถุ อาจไม่เหมาะสำหรับผู้ที่เริ่มศึกษาเนื่องจากมีความรายละเอียดของแต่ละส่วนค่อนข้างซับซ้อน แต่หากมีความเข้าใจแล้วจะเห็นได้ว่าการเขียนโปรแกรมเชิงวัตถุจะทำให้การพัฒนาระบบทำได้รวดเร็ว ปรับแก้ไขสคริปต์ได้ง่าย เพราะใช้เทคนิควิธีการสืบทอดคุณสมบัติ

คำถามท้ายบท

1. จงอธิบายองค์ประกอบของคลาส พร้อมยกตัวอย่างประกอบ
2. จงอธิบายถึงออบเจ็กต์ และอินสแตนซ์ รูปแบบการสร้าง พร้อมยกตัวอย่างประกอบ
3. จงอธิบายถึงเมธอด และความเกี่ยวข้องกับคลาส
4. จงอธิบายถึงโมดิฟายเออร์ แบบ public และ private มีความเหมือนหรือแตกต่างกันอย่างไร
5. จงอธิบายถึงลักษณะที่สำคัญของคอนสตรักเตอร์