PHP OOP

“มาเขียนเว็บแนว OOP กันเถอะ เดียวนี้พวกเฟรมเวิร์คแจกฟรีต่างๆ ก็เขียนกันในแนวนี้”

เราอาจเคยเรียนรู้การสร้างคลาส สร้างวัตถุด้วยภาษาอื่นๆ แนวคิดนั้นก็ใช้ได้กับภาษา PHP เพียงแต่ไวยากรณ์ของภาษา และข้อกำหนดอาจจะต่างกัน เรามาดูกันว่า ภาษา PHP มีการเขียนโปรแกรมเชิงวัตถุอย่างไร

การสร้างคลาส

สิ่งที่จะประกอบร่างเป็นคลาสได้ จะประกอบด้วย สมาชิกหลักๆ คือ แอททริบิว(Attribute) หรือพร็อบเพอร์ตี(Property) และเมธอด(Method) คลาสใน PHP จะต้องสร้างด้วยคีย์เวิร์ด “class” เสมอ

Code 1.

```
<?php
//class.product.php
class Product {
    private $id; //attribute
    private $name; //attribute

    public function getId(){//getter method
        return $this->id;
    }
    public function setId($id){ //setter method
        $this->id = $id;
    }
    public function getName(){ //getter method
        return $this->name;
    }
    public function setName($name){ //setter method
        $this->name = $name;
    }
}
```

มีหลายสิ่งที่เหมือนกับภาษาอื่นๆ เช่น ใช้ คำนำหน้าคลาส ว่า class (ตัวเล็กหมด) ใช้ ตัวควบคุมการเข้าถึงวัตถุ เช่น private ให้เข้าถึงได้เฉพาะคลาสตนเอง วัตถุมองเห็นไม่ได้ ส่วน public สามารถเข้าถึงได้ตลอด แต่ก็สิ่งที่สังเกตบางอย่างได้ไม่เหมือนคือว่า

- การตั้งชื่อตัวแปรจะใช้ $ นำหน้า และไม่ต้องระบุชนิดข้อมูล เพราะ Php จะเลือกให้เองเมื่อถูกกำหนดค่าให้ในครั้งแรก
- การกำหนดเมธอด ใช้ชื่อว่า function ทั้งเมธอด คืนค่า และไม่คืนค่า ไม่เมื่อกับภาษาอื่น เช่น Java จะใช้ void กับกรณีไม่คืนค่า และถ้าคืนค่าจะใช้ ชนิดข้อมูลระบุไปเลย เช่น Integer
- การอ้างอิงสมาชิก ใช้ เครื่องหมาย -> แทนการใช้ จุด กับภาษาอื่นๆ (เทียบกับจาวา)

แต่ดูดีๆ มีข้อสังเกตอีกอย่างหนึ่งคือ ตัวเปิดคำสั่ง <?php แต่ไม่มีตัวปิด ?> เพราะนี้เป็นคำสั่ง php ล้วนๆ ไม่มีภาษาอื่นๆ มาปน จึงไม่ต้องมีตัวปิด

ทดสอบสร้างวัตถุ

มาทดสอบกันเลยว่า คลาสที่สร้างก่อนหน้านี้ สร้างเป็นวัตถุได้อย่างไร ดังตัวอย่างการเขียนโปรแกรมต่อไปนี้

Code 2.

```
<?php
//index.php
include 'class.product.php';
$s1 = new Product();
$s1->setId(1);
$s1->setName("TV");
echo $s1->getId() . “:” . $s1->getName();
```

สังเกตซิว่า มีอะไรต่างกับภาษาทั่วไปอะไรบ้าง ดูเหมือนแทบหาที่ต่างกันได้น้อยมาก นอกจากไม่ต้องระบุชนิดข้อมูลของตัวแปรก่อน ซึ่งก็เท่านั้น

สร้างฟังก์ชันไว้ใช้เอง

แทนที่เราจะให้ echo จากเรียกคุณสมบัติมาดูทุกครั้ง เรามาสร้างฟังก์ชันมาใช้เองกันดีกว่า เช่น เราต้องการอ่าน id และ name ของวัตถุ เราสร้างให้อ่านเพื่อแสดงค่าได้ ดังเช่น

Code 3.

```
public function info(){
        echo $this->id . ":" .$this->name;
}
```

แล้วถ้าเราต้องการให้คืนค่า id และ name คิดว่าเราจะเขียน และเรียกใช้ได้อย่างไร

โอเวอร์โหลด (Overload)

การสร้างฟังก์ชัน ที่ใช้ชื่อเดิม แต่มีตัวแปรเข้าไม่เหมือนกัน เรียกฟังก์ชันเหล่านี้ว่า โอเวอร์โหลด ภาษาทาง OOP โดยทั่วไปก็ทำกันได้หมด แต่ PHP ยังทำโดยตรงไม่ได้ (PHP 5.6) หรือกล่าวอีกอย่างว่า เป็นการทำโอเวอร์โหลดในแบบของ PHP

ดูตัวอย่าง การทำฟังก์ชัน info( ) ก่อนหน้านี้ แต่ปรับปรุงข้อมูลภายใน เพื่อรองรับข้อมูลเข้า ในจำนวนที่ต่างๆ กัน ดังเขียนใหม่ได้ว่า

Code 4.

```
public function info(){
        $info = "";
        foreach (func_get_args() as $arg) {
            switch ($arg){
                case 'id': $info .= $this->id;  break;
                case 'name': $info .= $this->name; break;
```

```
            }
        }
        echo $info;
    }
}
```

เมธอด info( ) นี้ใช้ นับตัวแปรผ่าน func_get_args( ) ซึ่งเป็นตัวแปรในลักษณะอาร์เรย์ และใช้ตัววิ่งใน foreach เป็น $arg ของอาร์เรย์ ทำให้ฟังก์ชันนี้ รับตัวแปรที่จำนวนต่างๆ กันได้ คล้ายๆ กับการเขียนฟังก์ชัน info($array = null) ที่รับตัวแปรเข้าเป็นอาร์เรย์ แต่มีค่าปริยายเป็น null หรือหมายความว่า ถ้าต้องใช้งานไม่ใส่ตัวแปรอะไร ก็ให้ถือว่า ตัวแปร $array มีค่าเท่ากับ null ที่นี้มาดูการทดสอบใช้งานกัน

Code 5.

```
//index.php
include 'class.product.php';
$s1 = new Product();
$s1->setId(1);
$s1->setName("TV");
$s1->info("id", "name"); //print 1TV
```

การใช้ __get( ) , __set( )

ฟังก์ชัน __get( ) และ __set( ) จัดเป็นฟังก์ชันมายากล (Magic Method) และจัดให้สองฟังก์ชันนี้คือการทำโอเวอร์โหลด ในความหมายโอเวอร์โหลดของ PHP มีความหมายที่ไม่เหมือนกับภาษาอื่นๆ เพราะการทำโอเวอร์โหลดของ PHP คือสร้างคุณสมบัติแบบไดนามิก สองฟังก์ชันนี้สามารถสร้างคุณสมบัติแบบไดนามิกได้ โดยไม่ต้องสร้าง getter และ setter ให้ครบตามภาษา OOP ทั่วไป

สมมุติว่าต้องการสร้างคลาส Product ที่มีคุณสมบัติเหมือนกับคลาส Product ที่เคยสร้าง แต่ครั้งนี้ใช้ฟังก์ชันมายากลแทน

Code 6.

```
<?php
//class.product.php
class Product{
    private $data = array();
    public function __set($name, $value){
        $this->data[$name] = $value;
    }
    public function __get($name){
        return $this->data[$name];
    }
    public function info(){
        $info = "";
        foreach($this->data as $data){
            $info .= $data . " ";
        }
        return $info;
    }
}
```

ฟังก์ชันมายากลนี้ ต้องประกาศเป็น public เท่านั้น และฟังก์ชันเหล่านี้ สามารถเข้าถึงสมาชิกที่เป็น private ได้ การใช้งานทำได้ง่าย โดยไม่ต้องอ้างอิงฟังก์ชันมายากลเลย ลองดูการใช้งานกัน

Code 7.

```
<?php
//index.php
include 'class.product.php';
$product = new Product();
$product->id = 1;
$product->name = "TV";
echo $product->info();
```

คอนสทรักเตอร์ (Constructor)

สำหรับบคอนสตรักเตอร์ ตัวนี้ดูจะความแตกต่างกับภาษาอื่นๆ เพราะใช้คำสำคัญว่า __construct( ) สังเกตว่าใช้ อักษรขีดเส้นใต้สองอันเรียงติดกัน และสังเกตว่าคอนสตักเตอร์นี้ทำโอเวอร์โหลดไม่ได้ ทีนี้มาดูตัวอย่างกัน

Code 8.

```
<?php
//class.product.php
class Product {
    public function __construct( $name){
        $this->name = $name;
}
```

แล้วทดสอบกับไฟล์ index.php เดิม ที่ไม่ใช้ Magic Method ดูว่ายังทำงานได้เหมือนเดิมหรือไม่ (คิดว่ามีอะไรผิดพลาด) แล้วทดสอบอีกครั้งกับโปรแกรมต่อไปนี้

Code 9.

```
<?php
//index.php
include 'class.product.php';
$s1 = new Product("TV");
echo $s1->getName( );
```

กรณีที่ใช้ Magic Method (__get, __set) ทดสอบแสดงผลด้วย $s1->name แทน $s1->getName( ) อย่างไรก็ตาม การสร้างคอนสตรักเตอร์ ด้วยวิธีที่ผ่านมา จะไม่ใช้ชื่อคลาสแทนชื่อคอนสตรักเตอร์ เหมือนภาษาอื่นๆ แต่ PHP ก็อนุญาตให้ใช้ชื่อคลาสแทนคอนสตรักเตอร์ได้ แต่วิธีนี้ ไม่ใช่วิธีทางหลัก เพราะ PHP จะอ้างถึง __construct เป็นหลัก

สร้างดีสตรักเตอร์ (Destructor)

เมื่อสร้างคลาส จะมี คอนสตรักเตอร์ทำงานเป็นอันดับแรก เมื่อวัตถุไม่ถูกใช้งาน ดีสตรักเตอร์ก็ถูกเรียกใช้งานเป็นอันดับสุดท้าย คำสั่งเรียกนี้ เขียนด้วย __destruct( ) ลองใส่โปรแกรมข้างล่างนี้ลงในคลาสเดิมที่เคยทดสอบ และดูผลกันว่าเป็นอย่างไร

Code 10.

```
function __destruct( ) {
     echo " Object Destroyed.";
}
```

แล้วรู้ได้อย่างไรว่า วัตถุไม่ถูกใช้งานแล้ว คำถามนี้น่าสนใจ เราเองดูเองได้ว่า ไม่มีส่วนใดของโปรแกรมอ้างอิงถึงวัตถุนี้แล้ว หรือ เพื่อให้ชัดไปเลยว่าไม่ใช้งานวัตถุนี้แล้ว ก็ให้เขียน ไปเลยว่าไม่ใช้แล้ว โดยให้มีค่าเท่ากับ null

ค่าคงที่ของคลาส

แน่นอนเลยว่า เรารู้มาแล้วว่าสร้างค่าคงที่ใน PHP ด้วยการใช้คีย์เวิร์ด define เพื่อกำหนดค่าคงที่ เช่น กำหนดว่า DSN_ MYSQL ให้เป็นชุดอักษร ในการต่อเชื่อมกับฐานข้อมูล

```
define("DSN_ MYSQL", ="mysql:host=localhost;dbname=MySQLDB;charset=utf8");
```

เราก็จะใช้ DSN เป็นชื่อที่ใช้อ้างอิงทั้งระบบ ซึ่งคือตัวแปรสากล (Global Variable) นั้นเอง (ตรงข้ามกับตัวแปรท้องถิ่น) แต่การสร้างค่าคงที่ในคลาส จะต้องใช้คำสำคัญว่า const ค่าคงที่เมื่อประกาศค่าจะเปลี่ยนแปลงไม่ได้ เราสามารถสร้างค่าคงที่ และใช้งานค่านี้ได้อย่างไร

Code 11.

```
<?php
//class.student.php
class Student {
    const ID = 123456789;
    public $school = "South East";
    public function info(){
        $data = self::ID . ":" . $this->school;
        return $data;
    }
}//end class
```

Code 12.

```
<?php
//index.php
include("class.student.php");
$student = new Student();
echo  student->info();
```

Output: 123456789:South East

ทีนี้มาดูในรายละเอียดกัน ว่าใช้งานกันอย่างไร อย่างแรก

- ค่าคงที่เป็นธรรมเนียม ต้องนิยามใช้ชื่ออักษรพิมพ์ใหญ่
- ค่าคงที่นี้ไม่ต้องมีอะไรนำหน้า เช่น private หรือ public ใช้นำหน้าไม่ได้ แต่มีลักษณะเหมือน public
- ค่าคงที่ไม่ต้องมีอักษร $ นำหน้า
- การอ้างค่าคงที่ในคลาสใช้ self

- การอ้างอิงนอกคลาสใช้ ชื่อคลาสนำหน้า
- และอ้างถึงตัวแปรค่าคงที่ใดใช้ ::

ค่าสเตติก (Static)

ที่อ้างถึงค่าสเตติกมาก่อนหน้านี้ ในทางการเขียนโปรแกรมเชิงวัตถุ จะหมายถึงค่า ตัวแปรของคลาส (ไม่ใช้ตัวแปรของวัตถุ) ที่ทุกๆ วัตถุใช้ร่วมกัน แต่ค่านี้เปลี่ยนแปลงได้ (ถึงตอนนี้ คงแยกความแตกต่างได้ว่า ไม่เหมือนกับค่าคงที่ตรงไหน) มาดูกันจริงๆว่า ใช้งานตัวแปรชนิดนี้ เรียกผ่านคลาส ซึ่งตรงกันข้ามกับ ตัวแปรที่ไม่เป็นค่าสเตติก จะถือเป็นตัวแปรของวัตถุ ซึ่งจะเรียกผ่านวัตถุ

Code 13.

```
<?php
//class.product.php
class Product {
    private $id;
    private $name;
    private static  $nextID = 1;

    public function __construct($name){
        $this->name = $name;
        $this->id = self::$nextID;
        self::$nextID++;
    }
    public function getName(){
        return $this->name;
    }
    public function setName($name){
        $this->name = $name;
    }
    public function getId(){
        return $this->id;
    }
}
```

Code 14.

```
<?php
//index.php
include 'Product.php';
$products = array(new Product("TV"), new Product("TV"));
foreach ($products as $item ){
    $info = $item->getId() . ":" . $item->getName();
    echo $info . "<br>";
}
```

Output:

```
1:TV
2:TV
```

จะเห็นว่าการอ้างถึงใช้เหมือนค่าคงที่ แต่ต่างกันนิดหนึ่งคือ ยังคงมี $ นำหน้าตัวแปร เช่น ในที่นี้ใช้ self->$nextId แล้วถ้าอ้างอิงนอกคลาส จะอ้างอิงอย่างไร (ทดลองทำดู แต่ต้องแก้ จาก private เป็น public

ก่อน) และจากตัวอย่างนี้ จะเห็นว่า เรานำมาประยุกต์ใช้ตัวนับได้ เพราะค่าสเตติก จะไม่ขึ้นอยู่กับวัตถุ แต่จะคงอยู่กับตัวคลาส

การสืบทอด

คุณสมบัติยอดเยี่ยมอย่างหนึ่งใน OOP คือคุณสามารถสืบทอดคลาส และสร้างวัตถุใหม่อย่างสมบูรณ์ วัตถุใหม่สามารถรักษาทุกฟังก์ชันของวัตถุพ่อแม่ (parent) ต้นแบบได้ การสืบทอดนี้จะใช้ คำสำคัญ extends เพื่อการสืบทอด เช่น ให้คลาส Radio สืบทอดจากคลาส Product จะเขียนได้ว่า

Code 15.

```
<?php
//class.radio.php
class Radio extends Product{    }
```

เรามาลองทดสอบดูว่า เราจะสร้าง วัตถุ Radio โดยเรียกคุณบัติมาจากคลาสฐาน (Product) ได้ดังตัวอย่างต่อไปนี้ (สังเกตว่า เราต้องรวมไฟล์ Radio.php มาด้วย และต้องต่อจากไฟล์ Product.php เพราะคลาส Radio เรียกใช้คลาส Product จึงต้องมีคลาส Product มาก่อน)

Code 16.

```
<?php
//index.php
include 'Product.php';
include 'Radio.php';
$products = array(new Radio("TV"), new Radio("TV"));
foreach ($products as $item ){
    $info = $item->getId() . ":" . $item->getName();
    echo $info . "<br>";
}
```

การเรียกคอนสตรักเตอร์ที่สืบทอด

มีสิ่งที่ไม่เหมือนกับภาษาอื่นๆ ในการเรียกคอนสตรักเตอร์จากคลาสทีสืบทอด ในภาษาอื่นๆ เช่น Java คลาสที่สืบทอด จะเรียกคอนสตรักเตอร์ของคลาสที่สืบทอดก่อน หรือเกิดตามคลาสที่สืบทอด แต่สำหรับ PHP ไม่จำเป็นต้องเกิดตามคลาสทีสืบทอด ถ้าเขียนคอนสตรักเตอร์ของตนเอง

Code 17.

```
<?php
//class.radio.php
class Radio extends Product{
    public function __construct(){
       echo ("Constructor B");
}
```

ถ้าทดสอบ สร้างออปเจค Radio เช่น $radio =new Radio( ) จะได้ออปเจค $radio ที่ไม่มีค่าสเตติก ทำกำหนดค่า $nextId ติดมาด้วย วิธีการแก้คือ ต้องสร้างคอนสตรักเตอร์ตามแบบคลาส Product แล้วเรียกคอนสตรักเตอร์ของคลาส Product ดังตัวอย่างต่อไปนี้

Code 18.

```
<?php
//class.radio.php
class Radio extends Product {
    public function __construct($name) {
        echo "Constuctor Radio";
        parent::__construct($name);
    }
}
```

การเรียกในตัวอย่างที่ผ่านมา ใช้ parent:: เป็นตัวเรียกคอนสตรักเตอร์ของคลาสที่สืบทอด นอกจากนี้ถ้ามีการสืบทอดกันมาหลายๆ ทอด เราใช้ชื่อคลาสเรียกคอนสตรักเตอร์ได้ ดังตัวอย่างต่อไปนี้

Code 19.

```
<?php
//class.digitalRadio.php
class DigitalRadio extends Radio{
    public function __construct($name) {
        echo "Constuctor DigitalRadio";
        Product::__construct($name);
    }
}
```

โอเวอร์ไรด์

การสืบทอดกันของวัตถุ จากการใช้ extends เราสามารถโอเวอร์ไรด์ หรือเขียนทับเมธอดใดๆ ได้(ไม่ว่าจะประกาศเป็น protected หรือเป็น public) การเขียนทับ เมื่อเรารู้สึกว่า การทำงานของเมธอดจากคลาสฐานไม่ตรงตามความต้องการ หรือไม่อยากให้ทำงานเหมือนคลาสฐาน

Code 20.

```
<?php
//class.radio.php
class Radio extends Product{
    public function info(){
        echo "Radio:" . $this->getId() . ":" . $this->getName();
    }
```

ทดสอบเรียกใช้ จะเห็นว่าทำงานไม่เหมือนคลาสฐาน (Product) แล้ว

Code 21.

```
<?php
//index.php

include 'Product.php';
include 'Radio.php';

$radio = new Radio("Tanin");
$radio->info();

Output:
Radio:1:Tanin
```

ป้องกันการเขียนทับได้ด้วย final

ถ้าเราไม่ต้องการเปลี่ยนแปลงเมธอดในคลาสที่สืบทอด ได้อีก เราสามารถป้องกันได้ ด้วยคำสำคัญว่า final เช่น ใน info( ) ให้สิ้นสุดที่คลาส Radio จะเขียนได้ว่า

Code 22.

```
<?php
//class.radio.php
class Radio extends Product{
    public final function info(){
        echo "Radio:" . $this->getId() . ":" . $this->getName();
    }
}
```

ป้องกันสืบทอดได้ด้วย final

เมื่อป้องกันเขียนทับ ก็น่าจะป้องกันสืบทอดได้ด้วย แน่นอนทำได้ด้วยคำสำคัญเดียวกัน คือ final ดังเขียนคลาส Radio ใหม่เป็น คลาส ที่เป็นหมันแล้วได้ว่า

Code 23.

```
<?php
//class.radio.php

final class Radio extends Product{
    public final function info(){
        echo "Radio:" . $this->getId() . ":" . $this->getName();
    }
}
```

คลาสคลุมเครือ (abstract class)

ที่เขียนว่าคลุมเครือ หรือไม่ระบุให้ชัดไปเลยว่าให้ทำงานได้อย่างไร เราเรียกคลาสประเภทนี้ว่า abstract แล้วทำไมต้องเขียนคลาสประเภทนี้ไว้ด้วย โดยทั่วไปเมื่อเราตั้งใจจะทำให้คลาสไม่มีความชัดเจนในการทำงาน ก็เพื่อให้ผู้ที่นำคลาสไปใช้ต้องสร้างความชัดเจนเองภายหลัง ด้วยเหตุผลว่า ต้องการใช้ผู้ใช้ปรับแต่งได้เองตามที่ตนเองต้องการ เช่นคลาส Product เราไม่ต้องการระบุให้ชัดว่า ทำรายงานในหน้าเว็บแบบใด เราก็เขียนกำกับคลาส และเมธอด ด้วยคำสำคัญ abstract ดังเขียนคลาส Product ได้ใหม่ว่า

Code 24.

```
<?php
//class.product.php
abstract class Product {
  //เขียนเติมจากคลาสเดิมว่า
  abstract function report();
}
```

และเราเรียกใช้ ได้ว่า

Code 25.

```
<?php
```

```
//class.radio.php
class Radio extends Product{
    public function report(){
        echo "Radio:" . $this->getId() . ":" . $this->getName() . "<br>";
    }
}
```

คลาสไม่มีชื่อ

ถ้าเราต้องการสร้างออปเจค แบบปัจจุบันทันด่วน (โดยตั้งชื่อไม่ทัน) เพื่อใช้เฉพาะกิจ หรือชื่อนั้นไม่ได้สำคัญอะไร เพียงแค่ต้องการใช้งานมากกว่า จะสร้างเป็นชื่อ เป็นเรื่องเป็นราว โดยไม่ได้ตั้งใจจะไปใช้สร้างในครั้งต่อไป คลาสแบบนี้เราเรียกกันตรงๆ ว่า คลาสไม่มีชื่อหรือ anonymous class

สมมุติว่าเราต้องการสร้าง ที่เก็บรายการสินค้า และรายการร้องขอสินค้า ซึ่งเก็บคู่ๆ เช่น ออปเจค tv มีความต้องการ 2 หน่วย ออปเจค radio ต้องการ 4 หน่วย เหตุการณ์แบบนี้ เราสร้างเป็นคลาสเพื่อเก็บรายการสินค้าดังกล่าว ได้ดัง ต่อไปนี้

Code 26.

```
<?php
//index.php
include "class.product.php";
$tv = new Product();
$tv->setId(1)->setName("TV")->setStock(100)->setPrice(1000);
$radio = new Product();
$radio->setId(2)->setName("Radio")->setStock(1000)->setPrice(200);

//anonymous class
$products[ ] = (object)array('product' = > $tv, 'qty' => 2);
$products[ ] = (object)array('product' => $radio, 'qty' => 4);
```

การเรียกอ่าน ก็เหมือนกับการอ่านสมาชิกทั่วไปของคลาส ที่ยกตัวอย่าง มีสมาชิก product และ qty เรียกอ่านได้

Code 27.

```
<?php
//index.php
echo $products[0]->product->getName( ); //print TV
echo $products[0]->qty; //print 2
```

สำหรับ PHP7 สามารถ มีความสมบูรณ์มากขึ้นในด้านการสนับสนุนการเขียนโปรแกรมเชิงวัตถุ แนวทาการสร้างคลาสไม่มีชื่อ เขียนแนวทางใหม่ ดูจากตัวอย่างต่อไปนี้

Code 28.

```
<?php
$object = new class {
            public function hello($message) {
                  return "Hello $message";
            }
        };
echo $object->hello('PHP7');//print "Hello PHP7";
```

อินเตอร์เฟส (Interface)

อินเตอร์เฟสก็คือคลาสว่างเปล่า ซึ่งภายในมีได้เพียงการประกาศเมธอด ดังนั้นคลาสใดที่ใช้งานอินเตอร์เฟสจะต้องเขียนเพิ่มเติมเมธอดเอง ดังนั้นอินเตอร์เฟสไม่มีอะไรนอกจากกฎที่วางไว้ ซึ่งจะช่วยให้สืบทอดไปยังคาสใดๆ และจะต้องเขียนเติมทุกเมธอดของอินเตอร์เฟส คลาสหนึ่งสามารถใช้อินเตอร์เฟสใดๆ ด้วยการใช้คีย์เวิร์ด implements โปรดสังเกตว่าอินเตอร์เฟส คุณสามารถได้เพียงประกาศชื่อเมธอด แต่ไม่สามารถเขียนคำสั่งใดในเมธอดได้ นั้นหมายความว่าภายในตัวเมธอดใดๆ ต้องว่างเปล่า

แล้วทำไมจึงต้องมีอินเตอร์ด้วย คำถามนี้ก็คลาสกับ คลาสที่เขียนคลุมเครือ แต่สิ่งที่ต่างออกไปมากกว่านั้น ก็คือว่า อินเตอร์เฟส มีใช้สำหรับให้งานได้ทุกคลาส ที่ไม่จำเป็นต้องเกี่ยวเนื่องกับคลาสเลย เช่น เราต้องการสร้างอินเตอร์เฟสชื่อว่า DBDriver เพื่อใช้สำหรับการต่อเชื่อมกับฐานข้อมูล ให้กับคลาสต่างๆ ทุกคลาสที่ต้องการใช้งานกับฐานข้อมูล ซึ่งฐานข้อมูลอาจเป็นต่างชนิด อาจเป็น MySQL หรือเป็น SQLite แล้วแต่ว่าระบบที่ใช้งานต้องการใช้งานฐานข้อมูลประเภทใด

ตอนนี้ถ้าจะให้แต่ละคนเขียนโปรแกรมในรูปแบบของตนเองในคลาสของเขาเองได้อย่างไร นักพัฒนาที่ใช้คลาสไดเวอร์ใดก็จะตรวจสอบการนิยามเมธอดและทำตามนั้น เป็นวิธีที่เขาต้องเขียนตามโค้ดของเขา ซึ่งดูน่าเบื่อและน่ายุ่งยากในการดูแลรักษา ดังนั้นถ้าเราได้นิยามไว้ว่า ทุกคลาสไดรเวอร์ (driver) ไม่จำเป็นต้องกังวลขณะที่เปลี่ยนไดรเวอร์ เพราะเขารู้ว่าทุกคลาสเหล่านี้มีการนิยมเมธอดที่เหมือนกัน ในเหตุการณ์แบบนี้อินเตอร์เฟสช่วยได้ ลองมาสร้างอินเตอร์นี้กัน

Code 29.

```
<?php
//interface.dbdriver.php
interface DBDriver{
	public function connect();
	public function execute($sql);
}
```

สังเกตได้ไหมว่า เป็นฟังก์ชันว่างเปล่าในอินเตอร์เฟส ขณะนี้เรามาสร้างคลาส MySQLDriver ซึ่งแต่งเติมอินเตอร์เฟสนี้

Code 30.

```
<?php
//class.mysqldriver.php
include("interface.dbdriver.php");
class MySQLDriver implements DBDriver {
}
```

ตอนนี้ถ้าเราให้โค้ดนี้ทำงาน จะทำให้เกิดความผิดพลาดเพราะคลาส MySQLDriver ไม่ฟังก์ชัน connect ( ) และ execute ( ) ตามที่ได้นิยามไว้ในอินเตอร์เฟส ลองให้โค้ดทำงานแล้วอ่านความผิดพลาด

```
<b>Fatal error</b>: Class MySQLDriver contains 2 abstract methods
and must therefore be declared abstract or implement the remaining
methods (DBDriver::connect, DBDriver::execute)
```

แล้ว ตอนนี้เราต้องเพิ่มสองเมธอดเหล่านี้ในคลาส MySQLDriver ของเรา มาดูโค้ดต่อไปนี้

Code 31.

```
<?php
include("interface.dbdriver.php");
//class.mysqldriver.php
class MySQLDriver implements DBDriver {
	public function connect() {
	  //connect to database
	}
	public function execute() {
		//execute the query and output result
	}
}
```

ถ้าเราให้โค้ดทำงาน เราจะได้ข้อความผิดพลาดต่อไปนี้อีกครั้ง

```
<b>Fatal error</b>: Declaration of MySQLDriver::execute() must be
compatible with that of DBDriver::execute() in <b>
```

ข้อความที่ผิดพลาดมีแสดงว่าเมธอด execute ( ) ของเราไม่เข้ากันกับโครงสร้างเมธอด execute ( ) ที่ได้นิยามไว้ในอินเตอร์เฟส ถ้าตอนนี้เราดูในอินเตอร์เฟส คุณจะพบว่า เมธอด execute ( ) ควรมีหนึ่งตัวแปรเข้า ดังนั้นหมายความว่า เราจะอิมพลีเม้นท์ (Implement) ในคลาสของเรา ทุกโครงสร้างเมธอดต้องเหมือนกับที่ได้นยามไว้ในอินเตอร์เฟส ลองมาแก้คลาส MySQLDriver ของเราดังต่อไปนี้

Code 32.

```
<?php
//class.mysqldriver.php
include("interface.dbdriver.php");
class MySQLDriver implements DBDriver
{
	public function connect() {
	    //connect to database
	}
	public function execute($query) {
	    //execute the query and output result
	}
}
```

โพลิมอร์ฟิสึม (Polymorphism)

คุณสมบัติที่สำคัญอย่างหนึ่งที่เราเขียนโปรแกรมเชิงวัตถุคือ คุณสมบัติโพลิมอร์ฟิสึม ซึ่งก็คือ การปรับแปลงของวัตถุให้เหมาะกับการใช้งานได้เองของคลาสฐาน เช่น เราสร้างคลาส Product เป็นคลาสฐานของคลาส Radio ซึ่งไปได้ว่า เราอาจสร้างคลาสลูกของคลาส Product ได้อีกมากมาย และเมื่อเราต้องการเรียกใช้เมธอดของคลาสลูก เช่น function info( ) แต่เรียกผ่านคลาสฐาน การทำงานจะทำงานตามเมธอดของคลาสลูกแทนที่จะเป็นคลาสฐาน แล้วทำไมต้องทำอย่างนี้ ลองนึกถึง การเก็บคลาสลูก ซึ่งเป็นต่างคลาสกัน แต่มาจากคลาสฐานเดียวกัน เราจะสร้างที่เก็บ เป็นคลาสคลาสฐาน แทนที่จะคลาสลูก ซึ่งเห็นว่าสะดวกดี มาดูตัวอย่างกัน

Code 33.

```
<?php
//class.comptuer.php
include "class.product.php";
class Computer  extends Product{
      public function report() {
       echo "Computer:" . $this->getId() . ":" . $this->getName() . "<br>";
     }

}
```

Code 34.

```
<?php
//index.php

include 'class.product.php';
include 'class.radio.php';
include 'class.computer.php';

$products = array();
$products[ ] = new Radio("Tanin");
$products[ ] = new Computer("Acer");
$products[ ] = new Computer("IBM");

foreach($products as $p){
      $p->report();
}
```

Output:

```
Radio:1:Tanin
Computer:2:Acer
Computer:3:IBM
```

จากตัวอย่างนี้จะเห็นว่า เราใช้ตัวเก็บด้วยกัน และเราวนซ้ำอ่านโดยใช้ตัวแปรเดียวกันได้ โดยไม่ต้องระบุเลยว่า ต้องเป็นคลาสลูกตัวตัวใด แต่อย่างลืมจะต้องมีการทำโอเวอร์ไรด์ในคลาสลูกด้วย

เนมสเปส (Namespace)

การจัดไฟล์ให้อยู่เป็นห้อง จัดกลุ่มไฟล์พวกเดียวกันไว้ให้ที่เดียวกัน หรือเรียกว่า อยู่เป็นห้องๆ ในทางคอมพิวเตอร์เราเก็บไฟล์อยู่ในไดเร็กทอรี ดังนั้นไฟล์ที่อยู่คนละห้องกัน แต่ไฟล์ชื่อเดียวกัน ก็สามารถทำได้ ดังตัวอย่างต่อไปนี้ ที่มีชื่อไฟล์เดียวกัน และชื่อคลาสเดียวกัน แต่อยู่ต่างไดเร็กทอรี

Code 35.

```
<?php
//Radio.php
namespace tools;
class Radio {
    public function report(){
        echo "My name is a special radio." ;
    }
}
```

Code 36.

```php
<?php
//index.php
include 'Radio.php';
include 'tools/Radio.php';

$products = array();
$products[] = new Radio("Tanin");
$products[] = new tools\Radio();

foreach($products as $p){
    $p->report();
}
```

แต่สิ่งน่าสังเกตอย่างหนึ่งคือเราจะต้องระบุไฟล์ร่วมใช้งานอยู่ด้วย และต้องระบุเส้นทางของไฟล์ที่ต้องการใช้งาน ก่อนการเรียกใช้ เพียงแต่ใช้ชื่อคลาสเดียวกัน และการใช้เนมสเปสนี้ใช้ได้กับ PHP ตั้งแต่รุ่น 5.3.0 ขึ้นไป

กรณีที่สร้างเนมสเปสย่อยๆ เช่น ต้องการสร้าง products เป็น เนมสเปส ย่อยของ tools ก็เขียนได้ว่า tools\products ซึ่งใช้ \ เป็นตัวคั่น

Autoload

ที่ผ่านมา การใช้ Namespace มองแล้วไม่สะดวกเลย ที่จะต้องสร้างการ include ทุกครั้งที่ใช้งาน ใน PHP มีฟังก์ชันการโหลดไฟล์อัตโนมัติ ที่ชื่อ __autoload($class) เป็นฟังก์ชันประเภทมายากลตัวหนึ่ง จากตัวอย่างที่ผ่านมา ให้ทำการโหลดคลาสอัตโนมัติ

Code 37.

```php
<?php
//index.php
function __autoload($class){
    include $class.'.php';
}
use tools\Radio as R;

$products = array();
$products[] = new Radio();
$products[] = new R();
foreach($products as $p){
    $p->report();
}
```

ทดสอบความถูกต้อง ด้วย Assert

ในการเขียนโปรแกรม เราจะพบว่าบางครั้ง เราต้องการตรวจสอบอะไรบางอย่างก่อน เพื่อให้โปรแกรมที่อยู่บรรทัดต่อไปทำงานได้ ซึ่งดีกว่า ที่จะปล่อยให้เกิดความผิดพลาดขึ้นเอง ซึ่งอาจทำให้โปรแกรมส่วนอื่นๆ ทำงานผิดพลาดไปด้วย เช่น การทำงานกับฐานข้อมูล เราต้องการบันทึกข้อมูลลงฐานข้อมูล หลายๆ แถวๆ แต่ในระหว่างทาง เกิดความผิดพลาด ทำให้โปรแกรม ไม่สามารถทำงานต่อไปได้ แต่ก็มีบางส่วน บันทึกข้อมูลไปแล้ว

วิธีแก้ไข อย่างหนึ่งเขียนโปรแกรม ตรวจความผิดพลาดก่อนที่จะทำงานต่อไป แต่ PHP มีเครื่องมือสำหรับจัดการกับความผิดพลาดให้ใช้ ซึ่งคือ คำสั่ง assert ในภาษาอื่นๆ เขาก็มีคำสั่งนี้ใช้กัน

ยกตัวอย่างเช่น เราต้องการตรวจดูว่า 1 เท่ากับ 2 หรือไม่ เริ่มต้นเราต้องกำหนด ให้ assert ทำงานได้ก่อน ต่อมา กำหนดฟังก์ชันรองรับความผิดพลาด และเขียน ฟังก์ชันส่งความผิดพลาดไปยังฟังก์ชันรองรับความผิดพลาด

Code 38.

```
<?php
//index.php
//1.
//Active assert กำหนดให้ทำงาน
assert_options(ASSERT_ACTIVE, 1);
//Warning กำหนดให้มีการเตือนได้เมื่อมีความผิดพลาด
assert_options(ASSERT_WARNING, 0);
//กำหนด ให้ทำเฉย เมื่อมีความผิดพลาด
//assert_options(ASSERT_QUIET_EVAL, 1);
//กำหนดให้โปรแกรม หยุดทำงานต่อไป ถ้าเกิดความผิดพลาด
assert_option(ASSERT_BAIL, 1);

//2.
function assert_not_null_handler($file, $line, $code, $des=null)
{
    echo "<hr>Assertion Failed:
        File '$file'<br />
        Line '$line' '$des'<br />
        Code '$code'<br /><hr />";
}

//3. Set up the callback
assert_options(ASSERT_CALLBACK, 'assert_not_null_handler');

assert('1==2', 'two values is not equal');
```

สำหรับฟังก์ชัน รองรับการความผิดพลาด มีตัวแปรเข้า สี่ตัว คือ $file ใช้สำหรับแสดงว่า ไฟล์ใดที่ทำงานผิด $line ใช้สำหรับแจ้งว่าบรรทัดใดทำงานผิด $code ใช้สำหรับระบุ คำสั่งที่เขียนผิด และ $des ใช้สำหรับระบุข้อความเพิ่มเติมจะเขียนเพิ่มว่าผิดอะไร ซึ่งตัวแปรสุดท้ายไม่มีก็ได้

สำหรับการเรียกใช้ ใช้คำสั่ง assert ในกรณีนี้ใช้ assert('1==2', 'two values is not equal') โดยตัวแปร ต้องเป็น คำสั่งตรวจสอบความจริง หรือเท็จ และตัวแปรที่สอง ไม่ใส่ก็ได้ แต่ถ้าใส่จะแทน ตัวแปร $des ในฟังก์ชันรองรับความผิดพลาด ทั้งนี้คำสั่งนี้ตั้งใจจะทำให้ผิดพลาด ดังนั้นแล้วผลการแสดงหน้าจอ จะมีข้อความดังต่อไปนี้

```
Assertion Failed: File 'Assert\index.php'
Line '19' 'two values is not equal'
Code '1==2'
```

เขียนแบบไม่เป็น OOP

ถึงแม้ว่า แนวการเขียนแบบ OOP จะใช้เป็นข้อแนะนำให้เขียนแบบนี้ แต่ช่วงหลังๆ (PHP 5.3.0) ได้แนะนำแนวการเขียนแบบฟังก์ชันไม่มีชื่อ (Anonymous) ฟังก์ชันไม่มีชื่อนี้บางทีก็เรียกกันว่า Closure เราใช้ฟังก์ชันไม่มีชื่อนี่ในกรณีที่เราต้องการสร้างฟังก์ชันเฉพาะกิจ หรือมีความต้องการเฉพาะตอนนั้น จึงไม่จำเป็นต้องนิยมเก็บไว้ตั้งแต่ต้น ดังตัวอย่างต่อไปนี้

Code 39.

```
<?php
$closure = function($name){
	return sprint('Hello, %s', $name);
};

echo $closure("Tee");
//output : Hello, Tee
```

จากตัวอย่างนี้ มีการสร้างฟังก์ชันไม่มีชื่อ และเก็บไว้ที่ตัวแปร $closure เป็นฟังก์ชัน ทักทายชื่อ จากตัวแปรที่ใส่ และการเรียกใช้งานก็ทำได้โดย เรียกตัวแปร $closure

ฟังก์ชันไม่มีชื่อที่ยกตัวอย่าง ยังต้องใช้ชื่อตัวแปรเป็นตัวรับค่าฟังก์ชันอยู่ดี แต่นิยมใช้กันในลักษณะให้เป็นตัวเข้าฟังก์ชันเรียกกลับ (callback) หรือใช้เป็นตัวแปรเข้าเป็นฟังก์ชันไม่มีชื่อ การใช้งานจะทำงานที่หลังเรียกกลับยังฟังก์ชันที่เป็นตัวแปรเข้า ลองพิจารณาจากตัวอย่างต่อไปนี้

Code 40.

```
<?php
$increaseByOne = array_map(function ($integers){
	return $integers + 1;
}, [1, 2, 3]);

echo $increaseByOne;
//output: 2, 3, 4
```

จะเห็นว่า เขียนได้ในบรรทัดเดียว (แต่เขียนเป็นสามบรรทัดเพื่อให้อ่านง่าย) สั้น และกระชับ ฟังก์ชัน array_map รับตัวแปรสองตัว ตัวแรกเป็นฟังก์ชันไม่มีชื่อ ส่วนตัวที่สองอาร์เรย์ของจำนวนเต็ม

ถึงแม้ว่า จะมีผู้ให้ความเห็นว่า ฟังก์ชันไม่มีชื่อยังไม่ถือว่าเป็น Closure โดยแท้ เพราะความหมายที่ของ Closure คือ การใส่ตัวแปรเข้าเป็นฟังก์ชัน(ฟังก์ชันไม่มีชื่อ) หรือ encapsulating function[1] ถ้าเราใส่ฟังก์ชันไม่มีชื่อเป็นตัวแปรเข้า เหมือนดังตัวอย่างที่ผ่านมา ($increaseByone) จึงจะเรียกได้เต็มปากว่าเป็น Closure

ก็ยังมีบางความเห็น[2]ว่า Closure หมายถึงความสามารถจดจำตัวแปร จากภายนอกได้นอกจากตัวแปรภายในฟังก์ชันไม่มีชื่อนี้ นั้นคือ การส่งตัวแปรเข้าฟังก์ชันไม่มีชื่ออีกวิธีหนึ่ง คือการใช้ คีย์เวิร์ด use เพื่ออ้างอิงข้อมูลที่ใช้ในฟังก์ชันไม่มีชื่อนี้ ลองดูตัวอย่างนี้กัน

---

1 Fischer, Robert. "Java Closure and Lambda". Apress. 2015. P. 13.

2 Simpson, Kyle. "Scope & Closures". O'Reilly. 2014. P.48.

Code 41.

```
<?php
$name = "Pone.";
$closure = function($greeting) use($name){
     echo $greeting .  $name;
};
$closure("Hi,");//output : Hi, Pone.
```

การเขียนโปรแกรมเชิงฟังก์ชัน (Functional Programming)

เมื่อ OOP รวมฟังก์ชัน กับ ข้อมูล (attribute) ไว้ด้วยด้วยกัน ด้วยวิธีการที่เรียกว่า Encapsulation และเมื่องานที่ใช้ไม่ได้สนใจตัวข้อมูลมากมายอะไรนัก แต่ให้ความสำคัญกับฟังก์ชันหรือหน้าที่การทำงานมากว่ากันมาก และมากๆ แล้วเหตุใดเราต้องเขียนให้เป็น OOP ให้เสียเวลา และดูรกรุงรัง

แนวคิดนี้ก็มาจาก หัวข้อก่อนหน้านี้ การเขียนไม่เป็น OOP ซึ่งจะมาแจกแจงกันในรายละเอียดกันว่า การเขียนโปรแกรมเชิงฟังก์ชัน ควรมีลักษณะการใช้งานเพื่อการแจ้งให้ทราบว่าต้องการผลลัพธ์อะไร มากกว่าต้องแจ้งให้เครื่องคอมพิวเตอร์ทำงานอย่างไร เช่น การแจ้งความต้องการในรูป Lambda Expression พื้นฐานมากจาก ฟังก์ชันไม่มีชื่อ (anonymous function) ดูตัวอย่าง ที่ผ่านก่อนจะถูกดัดแปลงเป็น Closure

Code 42.

```
<?php
//class.Product.php
class Product{
  public function __construct($price, $cost) {
       $this->price = $price;
       $this->cost = $cost;
  }
}

$profit = function($product, $qty){
   return ($product->price - $product->cost) * $qty;
};

$margin = $profit(new Product(5,3), 2);
echo $magin;//print 2
```

ฟังก์ชันของฟังก์ชัน (Higher order function) ต่างจากฟังก์ชันธรรมดา คือแทนที่เราจะใช้งานฟังก์ชัน แบบทั่วไป ที่รับค่าตัวแปรเข้า นำตัวแปรไปทำงานบางอย่าง ฟังก์ชันของฟังก์ชัน จะรับตัวแปรเข้าเป็นฟังก์ชัน แล้วนำฟังก์ชันเข้ามาทำงานกับฟังก์ชันหลักอีกที แล้วคืนค่าเป็นฟังก์ชันอีกครั้งก็ได้ หรือคืนค่าแบบทั่วไปก็ได้ ดูตัวอย่างฟังก์ชัน array_filter( ) ที่รับค่าตัวแปรสองตัว คือ $input และ function($item)

Code 43.

```
<?php
$input = array(1, 2, 3, 4, 5);
$filter_odd = function ($item){  return ($item % 2 == 1;  };

$odd_data = array_filter($input, $filter_odd);
```

หรือ จะรับข้อมูลเข้าเป็น anonymous function เลยก็ได้

ข้อมูลในฟังก์ชันมีการเปลี่ยนแปลงได้แต่ไม่มีผลต่อข้อมูลภายนอก (Immutability) เช่น เรามีชุดข้อมูลหนึ่งใส่เข้าฟังก์ชัน ชุดข้อมูลที่ใส่เข้านี้ต้องมีสถานะคลายกับค่า final คือ ไม่เปลี่ยนแปลงอีกแล้ว เราทำได้เพียง อ่านอย่างเดียว แต่ถ้าสร้างตัวแปรภายในฟังก์ชัน ข้อมูลตัวแปรนี้เปลี่ยนแปลงได้ เรามาดัดแปลงค่า input โดยเพิ่ม เลข 7 ไปอีกหนึ่งตัว ทำให้ฟังก์ชันทำงานไม่ได้

Code 44.

```
<?php
$input = array(1, 2, 3, 4, 5);

$odd_data = array_filter($input, function($item) {
   $item[ ] = 7; //add more one element make an error
   return ($item % 2) == 1;
  }
);
foreach($odd_data as $item) echo $item . "<br>";
```

การเขียนโปรแกรมเชิงฟังก์ชันเชิงลักษณะไดนามิก พบได้ใน callback เป็นกลยุทธ์อย่างในการเรียกใช้ฟังก์ชัน ดังตัวอย่างการเรียก array_filter หรือการใช้ call_user_func( )

Code 45.

```
<?php
function increment($var)
{
    echo $var;
}

$a = "The end.";
call_user_func('increment', $a);
```

แบบฝึกหัด

ถ้ากำหนดให้ คลาส Product มีข้อมูลดังต่อไปนี้

```
<?php
//class.product.php
class Product{
   private $id;
   private $name;
   public function info(){
     return $this->id . “:” . $this->name;
   }
}
```

1. สร้างฟังก์ชัน setId และ setName เพื่อกำหนด ค่า id และ name
2. ทดสอบ สร้างวัตถุ p1 จากคลาส Product และเรียกใช้ฟังก์ชัน info
3. แก้ไขฟังก์ชัน setId และ setName ให้คืนค่า วัตถุตัวเอง
4. ทดสอบ สร้างวัตถุ p2 จากคลาส Product ให้เรียก setName ผ่านฟังก์ชัน setId แล้วเรียกใช้ฟังก์ชัน info
5. สร้างคลาส Computer ให้สืบทอดคลาส Product โดยกำหนดให้คลาส Computer มีคุณสมบัติเพิ่มเติม มีคุณสมบัติ price กำหนดเป็น private และ getter และ setter ตามลักษณะที่ทำมาก่อนหน้านี้
6. ทดสอบ สร้างวัตถุ c1 จากคลาส Computer กำหนดค่า id, name, price และเรียกใช้ฟังก์ชัน info
7. ทำการ override ฟังก์ชัน info ให้เพิ่มการคืนค่า price
8. ทดสอบสร้างวัตถุ c2 จากคลาส Computer กำหนดค่า id, name, price แล้วเรียกฟังก์ชัน info
9. สร้างคลาส Radio ให้สืบทอดมาจากคลาส Product กำหนดคุณสมบัติราคา เหมือนกับคลาส Computer
10. ทำการ override ฟังก์ชัน info ให้เพิ่มการคืนค่าราคา
11. ทดสอบสร้างวัตถุ r1 จากคลาส Radio กำหนดค่า id, name, price แล้วเรียกฟังก์ชัน info
12. สร้าง อาร์เรย์ $products เก็บ ค่า c1, c2, r1 เก็บไว้ในที่เดียวกัน
13. วนซ้ำเรียก ฟังก์ชัน info
14. สร้างตัวแปร $closure เพื่อจากฟังก์ชันไม่มีชื่อให้ทำงานได้แบบเดียวกับข้อ 13 และทดสอบการใช้งาน

อ้างอิง

https://wiki.php.net/rfc/closures