<절> 5.2 객체</절>

프로그래밍 기법 중에는 절차식 프로그래밍과 객체지향 프로그래밍이 있습니다. 우리가 지금까지 작성해온 프로그램들이 절차식 프로그래밍 기법을 사용한 것인데 절차식 프로그래밍은 기능에 초점을 맞추어 순차적으로 어떤 기능 들을 수행하여야 하는지를 표현해가는 방식을 말합니다. 즉, 어떤 작업을 하는데 있어서 수행하여야 하는 기능을 함수로 만들고 그 기능들의 처리 순서를 고려하여 작성해둔 함수를 호출하는 방식입니다. 이에 반해 객체지향 프로그래밍은 기능이 아닌 데이터에 초점을 두고 객체라는 단위로 모든 처리를 표현해 가는 방법을 말합니다.
최근에는 객체지향 프로그래밍이 대세라고 불릴 만큼 각광 받고 있으며, 근래에 각광받는 언어들은 대부분 객체지향 언어입니다. 이에 발맞추어 PHP에서도 앞서 PHP4에서부터 객체 지향 프로그래밍 기법을 도입하였으나 실제로 부족한 점이 많이 있었고 PHP5가 되면서 보다 객체지향에 가까운 언어로 변모하게 되었습니다.

객체지향 프로그래밍이란 말은 말 자체에서부터 무슨 소린지 알 수 없는 포스를 갖고 있습니다. 그래서 시작도 하기 전에 겁을 잔뜩 먹게 하는 객체지향이란 말은 "Object-Oriented"라는 말을 번역한 것인데 단순히 말해서 모든 사물을 객체로 바라본다는 의미입니다. 객체는 기능(method)과 속성(property)으로 구성되어 있습니다. 그러니까 모든 사물을 기능과 속성을 가진 객체로 생각하자는 것이 바로 객체지향 프로그래밍의 핵심입니다.

인간을 객체로 생각해 보겠습니다. 인간은 어떠한 기능을 할까요? 객체지향 프로그래밍에서 말하는 기능은 객체가 할 수 있는 동작 또는 행위를 의미합니다. 따라서 인간은 걷고 달리고 말하고 음식을 먹고 잠을 자는 등의 모든 동작이 바로 기능이 됩니다. 속성은 객체가 갖는 데이터로 인간의 경우 이름이나 나이, 주소 등등이 바로 속성이 됩니다. 바로 이러한 기능과 속성을 갖는 것이 인간이다라고 인간을 정의하는 것을 인간 클래스(class)라고 합니다.
클래스는 판화와 같습니다. 사람 모양의 판화를 만들고 잉크를 발라서 똑 같은 모양을 계속해서 찍어낼 수 있듯이 인간이란 클래스를 만들어 찍어내면 철수니 영희니 돌쇠니 하는 사람들이 나오게 되는 것입니다. 이렇게 클래스를 통해 만들어진 철수를 객체라고 합니다.
클래스를 정의하기 위해서는 우선 사물을 추상화(abstraction) 해야 합니다. 인간이라는 클래스를 정의하고자 할 때 프로그래머가 인간의 모든 기능과 속성들을 정의할 수가 없습니다. 너무나 당연하지만 인간을 표현하기에는 인간이 알고 있는 지식은 터무니 없이 부족합니다. 그뿐만이 아니라 그 부족한 지식마저도 매우 방대합니다. 그렇기 때문에 인간을 정의하기가 쉽지 않습니다. 그래서 인간을 표현할 수 있게 중요하고 관계 있는 부분만 분리해서 간결하고 이해하기 쉽게 정리하는 것을 바로 추상화라고 합니다.

인간을 간단히 추상화하여 인간 클래스를 만들어 보았습니다.

<소스코드>

```
<?
class Human {  //인간 클래스를 정의합니다.

        public $Name;
        public $Age;
        public $Height;
        public $Weight;

        public function Eat ( $foods ) {
                //먹는 행위를 함수로 정의
                echo "우걱우걱~ 맛있는 " . $foods . "<BR>";
        }
        public function Walk ( $destination ) {
```

```
                //걷는 행위를 함수로 정의
                echo "뚜벅 뚜벅" . $destination . "까지 걸어요.<BR>";
        }
        public function Work ( $job ) {
                //일하는 행위를 함수로 정의
                echo "열심히 " . $job . " 일을 하자!<BR>";
        }
        public function Talk ( $words ) {
                //말하는 행위를 함수로 정의
                echo $words . "<BR>";
        }
}
?>
```
</소스코드>

간단히 이름, 나이, 키, 몸무게의 속성을 가지고 먹고 걷고 일하고 말하는 행위를 기능으로 정의하였습니다. 객체지향에 대한 개념이 머리속에 그려지시나요? 이제 좀 더 깊이 배워봅시다.

<중제목> 5.2.1 생성자 </중제목>

인간이 태어나면 일단 이름을 정해주고 나이를 한 살 먹는 등 인간의 속성값이 초기화되어야 합니다. 이를 위해서 생성자(constructor)라는 것이 존재합니다. 생성자는 객체를 초기화하기 위해서 생겨난 것으로 일반적으로 클래스의 이름과 동일한 이름의 함수가 생성자가 되는데 PHP5에서는 특별히 __construct() 라는 이름의 함수를 따로 만들어 두었습니다. 생성자는 반드시 필요한 것은 아니므로 생략하여도 상관없습니다. 인간에 대한 생성자를 정의해보면 다음과 같습니다.

<소스코드>
```
function __construct($hname)
{
        $this->Name = $hname;
        $this->Age = 1;
}
```
</소스코드>

아기가 태어나면 이름이 정해지고 나이를 한 살 먹게 합니다. 몸무게나 키도 생성자의 파라미터를 통해서 전달하여 초기화 할 수 있습니다. 여기서 $this는 현재의 클래스를 가리키는 특별한 변수입니다. 이 변수를 이용하여 클래스 내의 멤버 변수나 함수에 접근할 수 있습니다. 변수와 함수를 다루기 위해서는 위의 코드에서 볼 수 있듯이 '->' 기호를 이용하면 됩니다. 특히 변수의 경우 변수 앞의 "$" 기호를 제외하여야 제대로 변수에 접근할 수 있습니다.

PHP5에서는 생성자의 함수 오버로딩(Function Overloading)을 지원하지 않기 때문에 여러 개의 생성자를 사용할 수가 없습니다. 그렇기 때문에 파라미터 초기화를 사용하여 여러 개의 생성자가 존재하는 것과 같은 효과를 얻을 수 있습니다.

<소스코드>
```
function __construct($pName, $pAge=1, $pHeight=50, $pWeight=3.5)
{
        $this->Name = $pName;
        $this->Age = $pAge;
        $this->Height = $pHeight;
        $this->Weight = $pWeight;
```

```
}
```
</소스코드>

이 코드의 경우 첫번째 인자인 $pName은 필수 인자이지만 나머지 세가지 인자는 생략이 가능합니다. 또한 2개, 3개 그리고 4개의 인자에 가변적으로 대처할 수 있습니다. 단 인자의 순서를 변경하거나 중간 인자를 생략할 수는 없습니다.

<중제목> 5.2.2 소멸자 </중제목>

생성자와 마찬가지로 소멸자(destructor)도 존재합니다. 객체가 소멸되면서 해야 하는 일들을 정의하는 함수입니다. __destruct() 라는 이름을 가지며 인간의 경우에 죽으면서 유언을 남기는 것과 같은 일을 소멸자에 정의할 수 있습니다. 소멸자는 생성자와 달리 파라미터를 가지지 못합니다.

<소스코드>
```
function __destruct()
{
	$this->Talk( "창문을 닫아 주오!");
}
```
</소스코드>

유명한 방랑시인 김삿갓의 유언을 한번 인용해 보았습니다. 돌아가실 때 찬바람이 불었나 봅니다. 이렇게 소멸자를 만들어 두면 모든 인간들이 죽으면서 하나같이 이 말을 하게 될 것입니다. 따라서 유언이라는 속성을 만들어 두어 사람마다 다른 유언을 하도록 하는 것을 생각해 볼 수 있습니다. 생성자와 소멸자를 추가한 인간 클래스는 다음과 같습니다.

<소스코드>
```
<?
class Human {  //인간 클래스를 정의합니다.

	public $Name;
	public $Age;
	public $Height;
	public $Weight;

	function __construct($pName, $pAge = 1, $pHeight=50, $pWeight=3.5)
	{
		$this->Name = $pName;
		$this->Age = $pAge;
		$this->Height = $pHeight;
		$this->Weight = $pWeight;
	}

	function __destruct()
	{
		$this->Talk("창문을 닫아주오!");
	}

	public function Eat( $foods ) {
		//먹는 행위를 함수로 정의
		echo "우걱우걱~ 맛있는 " . $foods . "<BR>";
	}
	public function Walk( $destination ) {
		//걷는 행위를 함수로 정의
```

```
                echo "뚜벅 뚜벅" . $destination . "까지 걸어요.<BR>";
        }
        public function Work( $job ) {
                //일하는 행위를 함수로 정의
                echo "열심히 " . $job . " 일을 하자!<BR>";
        }
        public function Talk( $words ) {
                //말하는 행위를 함수로 정의
                echo $words . "<BR>";
        }
}
?>
```
</소스코드>

<중제목> 5.2.3 인스턴스 생성하기 </중제목>

인간 클래스를 정의하였으니 이제 이 클래스를 이용하여 철수, 영희와 같은 객체를 생성해 보겠습니다. 클래스를 통해 객체를 만들어내는 일을 인스턴스(Instance)를 생성한다고 말합니다. 객체는 new 키워드를 사용하여 생성할 수 있습니다.

<소스코드>
```
<?
/* Human 클래스를 여기에 삽입 */

$charles = new Human('철수');
$younghee = new Human('영희', 1, 50, 3.5);
$charles = NULL;//철수 객체 제거
$younghee = NULL;//영희 객체 제거
?>
```
</소스코드>

위의 예처럼 new 키워드를 이용하여 객체를 판화 찍어내듯이 계속해서 찍어낼 수 있습니다. 철수와 영희는 파라미터의 수가 다릅니다. 철수의 경우에는 이름만 전달되기 때문에 몸무게와 키 정보는 초기값으로 설정됩니다.

<중제목> 5.2.4 객체의 속성과 기능 사용하기 </중제목>

이제 객체를 생성하였으니 생성한 객체를 이용하는 방법에 대해 알아 봅시다. 객체의 속성과 기능에 접근하기 위해서는 앞에서와 동일한 방법인 '->' 기호를 사용하면 됩니다.

<소스코드>
```
<?
/* Human 클래스를 여기에 삽입 */

//5살짜리 철수를 생성합니다.
$charles = new Human('철수', 5);

//철수는 몇 살?
$charles->Talk($charles->Age);

//철수야! 밥 먹자~
$charles->Eat("dinner");

//밥을 먹고 난 후 철수의 키와 몸무게가 늘었다.
```

```
$charles->Height = 110; // 110 cm
$charles->Weight = 22; // 22 Kg

?>
```
</소스코드>

위와 같이 '->' 기호를 이용하여 객체의 변수와 함수들에 접근할 수 있습니다. 철수는 몇 살인지 대답을 하고 밥을 먹게 됩니다. 밥을 먹고 난 후 키와 몸무게가 증가하여 110cm와 22kg이 되었습니다. 여기서 주의할 것은 클래스의 멤버 변수를 접근할 때 "$" 기호를 붙이면 안된다는 것입니다.

<중제목> 5.2.5 클래스 상속하기 </중제목>

객체지향 프로그래밍의 장점 중 하나인 상속에 대해 배워 보겠습니다. 절차식 프로그래밍에서는 새로운 모듈을 개발하기 위해서 보통 비슷한 모듈의 소스 코드를 복사해다가 수정하여 원하는 기능을 구현하는 방법을 사용하였습니다. 그러나 객체지향 프로그래밍에서는 상속을 지원하여 클래스를 수정하는 것이 아니라 기존의 클래스의 속성과 기능을 상속받아 새로운 클래스를 정의할 수 있게 해줍니다.
아기는 인간의 한 형태이기 때문에 인간의 속성과 기능을 가집니다. 만약 아기 클래스를 새롭게 정의하고자 한다면 인간의 모든 속성과 기능을 다시 정의해주고 아기 만의 속성과 기능을 추가해 주어야 할 것입니다. 하지만 상속을 이용하면 인간의 모든 속성과 기능을 새로 정의해줄 필요가 없습니다. 즉, 아기 클래스는 인간 클래스의 속성과 기능을 모두 물려받을(상속) 수 있습니다.

<소스코드>
```
<?
/* Human 클래스를 여기에 삽입 */

class Baby Extends Human {  //인간 클래스를 상속받아 아기 클래스를 정의 합니다.

        function 모유먹기 () {
                //모유를 먹는 행위를 함수로 정의
                echo "냠냠~";
        }
        function 천사와대화 () {
                //천사와 대화하는 행위를 함수로 정의
                echo "옹알~ 옹알~";
        }
}

    //아기 클래스를 이용해 재민이 객체를 생성
$재민 = new Baby('재민');
$재민->천사와대화(); // 옹알~ 옹알~
?>
```
</소스코드>

PHP에서 상속을 받기 위해서는 Extends 라는 키워드를 사용하면 됩니다. Extends 뒤에 오는 클래스의 모든 기능과 속성을 그대로 상속 받아 사용할 수 있기 때문에 생성자와 소멸자를 비롯하여 새롭게 정의하지 않아도 모두 사용할 수 있게 됩니다.
C++와 자바와 같은 객체지향 프로그래밍 언어에서는 다중 상속을 지원합니다. 다중 상속이란 하나 이상의 클래스에 대해 상속을 받을 수 있는 것을 말합니다. 그러나 PHP에서는 다중 상속을 지원하지 않기 때문에 하나의 클래스만을 상속받을 수 있습니다.

<중제목> 5.2.6 public, private, protected </중제목>

객체지향 프로그래밍에서는 클래스를 캡슐화하여 내부를 접근할 때 제약을 둘 수 있습니다. 이는 클래스를 사용하게 될 프로그래머가 내부의 구조를 모르도록 하기 위함입니다. 왜 내부 구조를 모르도록 하는 걸까요? 클래스를 사용하는 사람의 입장에서는 클래스 내부를 자세하게 모르더라도 클래스를 만든 사람이 제공해주는 함수와 변수만을 사용하면 되기 때문입니다. 예를 들어 클래스 내부에 100개의 크고 작은 함수가 정의되어 있는데 이 모두가 work() 라는 함수의 기능을 잘게 쪼개어 표현한 것이라고 해봅시다. 정작 우리는 work() 함수만 알면 클래스의 모든 기능을 사용할 수 있는데 굳이 100개의 다른 함수들을 알 필요가 있을까요? 대부분의 경우에는 전혀 신경 쓸 필요가 없을 것입니다. 이 경우 프로그래머는 클래스를 보다 간단 명료하게 이해할 수 있습니다. 또한 클래스를 설계하는 사람의 입장에서는 자신이 설계한 클래스를 보다 안전하게 사용하기 위해서 외부로 공개된 함수와 변수로만 클래스를 접근하여 사용하도록 하고자 하는 것입니다.

이렇게 클래스 내부로의 접근 권한을 구분하기 위하여 다음과 같은 세가지 키워드를 사용하고 있습니다.

[표 5-3] public, private, protected 키워드

키워드	설명
public	모두에게 공개
private	함수와 변수가 정의된 클래스에서만 접근 가능
protected	함수와 변수가 정의된 클래스와 이를 상속한 클래스에서만 접근 가능

이들 키워드에 대한 이해를 돕기 위해서 다음과 같이 간단한 인간 클래스를 살펴봅시다.

<소스코드>
```
<?
	class Human
	{
		public $name = "김씨";
		private $secret = "임금님 귀는 당나귀귀!";
		protected $treasure = "선녀 날개옷";

		public function talk()
		{
			echo $this->name . "<BR>";
			echo $this->secret . "<BR>";
			echo $this->treasure . "<BR>";
		}
	}

	$KimC = new Human;

	$KimC->talk(); //모두 동작

	echo $KimC->name . "<BR>"; //동작
	echo $KimC->secret . "<BR>"; //에러 발생 : private 접근 불가
	echo $KimC->treasure . "<BR>"; //에러 발생 : protected 접근 불가

?>
```
</소스코드>

KimC 객체를 생성한 후 $name 변수에 대해서는 아무 문제없이 접근이 가능합니다. 그러나 $secret 변수와 $treasure 변수의 경우 각각 private와 protected로 선언되어 있기 때문에 이와 같은 방법으로는 접근이 불가능합니다. 즉, public으로 선언되어 있는 함수와 변수만 외부로 보여

지기 때문에 인간 클래스가 위와 같이 정의되어 있음에도 불구하고 우리 눈에 보이는 것은 단지 $name 변수와 talk() 함수 뿐입니다. 그래서 이 둘을 제외한 나머지 함수와 변수에 접근하면 접근을 할 수 없다는 에러가 발생합니다. 예제에는 private과 protected로 정의된 함수가 없지만 변수와 동일하게 객체로는 접근이 불가능합니다.

여기서 talk() 함수를 유심히 볼 필요가 있습니다. 객체를 통해서 직접 접근을 할 수 없었던 $secret 과 $treasure 변수가 모두 제대로 출력이 되기 때문입니다. 이는 앞서 표에서 정리하였듯이 두 변수가 정의된 클래스 내에서의 접근이기 때문에 모두 가능한 것입니다. 이 Human 클래스를 상속해보면 protected로 선언된 $treasure변수는 상속이 되지만 private로 선언된 $secret 변수는 접근이 불가능한 것을 알 수 있습니다.

<소스코드>
```
<?
	class Human
	{
		public $name = "김씨";
		private $secret = "임금님 귀는 당나귀귀!";
		protected $treasure = "선녀 날개옷";

		public function talk()
		{
			echo $this->name . "<BR>";
			echo $this->secret . "<BR>";
			echo $this->treasure . "<BR>";
		}
	}

	class Baby Extends Human
	{
		public function talk_baby()
		{
			echo $this->name . "<BR>";
			echo $this->secret . "<BR>";
			echo $this->treasure . "<BR>";
		}
	}

	$KimC = new Baby;

	$KimC->talk_baby(); //secret은 출력되지 않음
	$KimC->talk(); //모두 동작
?>
```
</소스코드>

Human 클래스를 상속한 Baby 클래스에서 talk_baby() 함수는 부모 클래스인 Human 클래스의 public과 protected 변수와 함수를 접근할 수 있습니다. 그래서 $secret 변수를 제외한 나머지가 모두 출력됩니다. $secret 변수를 출력할 때 에러가 발생하지 않는 이유는 Baby 클래스에서는 $secret이라는 변수 자체가 없는 변수로 인식되기 때문입니다. 반면에 talk() 함수를 호출해보면 모든 값이 출력되는 것을 알 수 있는데요. 이는 Baby 클래스가 부모 클래스의 talk() 함수를 그대로 상속받았기 때문입니다. talk() 함수는 부모 클래스에서 public으로 선언하여 상속이 가능하고 함수가 호출되면 부모 클래스의 권한으로 실행되기 때문에 private으로 선언된 $secret 변수에도 접근이 가능해집니다.

<중제목> 5.2.7 객체 지향 프로그래밍이 필수인가? </중제목>

최근의 프로그래밍 트렌드는 객체지향 프로그래밍입니다. 그래서 PHP에서도 서둘러 객체 지향 프로그래밍 기법을 도입했습니다. 그러나 객체 지향 프로그래밍은 대규모 프로젝트를 위해서 태어났다고 해도 과언이 아닙니다. PHP와 같이 대부분의 프로그램이 짧게 금방 작성할 수 있을 만큼 쉽고 PHP 스크립트 자체가 한 줄 한 줄 처리되는 방식이기 때문에 사실 어떻게 보면 PHP와 객체지향 프로그래밍은 어울리지 않는다고 볼 수 있습니다. 무조건적인 유행을 따라가는 것이 선두를 향해가는 것이라고 생각하는 것은 오산입니다. 필자는 개인적으로 PHP 또는 웹이라는 환경에 의한 제약 때문에 객체 지향 프로그래밍의 장점이 빛을 발하기 쉽지 않다고 생각합니다. 그래서 많은 PHP 프로그래머가 아직 객체지향 프로그래밍이 아닌 절차식 프로그래밍 방식을 사용하고 있습니다. 하지만 점점 웹 프로그램도 규모가 방대해지고 있어서 여러 명의 개발자가 참여하는 프로젝트가 많아지고 있습니다. 이에 따라 객체 지향 프로그래밍 기법도 따라서 더 많이 활용되지 않을까 생각됩니다.