WEB PROGRAMLAMA II

Öğr. Gör. M. Mutlu YAPICI

Ankara Üniversitesi
Elmadağ Meslek Yüksekokulu

Öğr. Gör. M. Mutlu YAPICI

Ders İzlencesi

Hafta	Modüller/İçerik/Konular
1. Hafta	Oturum yönetimi
2. Hafta	Cookies kullanımı ve oturum yönetimi
3. Hafta	Session kullanımı ve oturum yönetimi
4. Hafta	Sayfalama ve Arama İşlemleri
5. Hafta	JavaScript, JQuery ve PHP
6. Hafta	AJAX ve PHP
7. Hafta	AJAX ve PHP
8. Hafta	ARA SINAV
9. Hafta	PHP'de Nesne Yönelimli Programlamaya Giriş(347)
10. Hafta	PHP de PDO
11. Hafta	
12. Hafta	
13. Hafta	
14. Hafta	

Bu Ünitede Ele Alınan Konular

- Nesne tabanlı programlama ve tanımı
- Class, Method, Fonction, KeyWord kavramları
- Class ve Method oluşturma
- Access Modifiers (Erişim Düzenleyiciler)
- Kurucu ve yıkıcı metodlar

Encapsulation (Kapsülleme)

- Bilindiği gibi PHP nesneye dayalı bir programlama dilidir. Kapsülleme (Encapsulation) kavramı bir nesnenin bazı özellik ve işlevlerini başka sınıflardan ve nesnelerden saklamaktır. Private erişim belirteci sayesinde bu şekilde tanımlanan bir alanı başka sınıflardan gizlemiş oluruz. Ayrıca bu alan başka sınıflarda kullanılamaz.
- Kapsülleme (Encapsulation) sayesinde nesneler bilinçsiz kullanımdan korunmuş olur. Fakat bazı durumlarda private alanlara erişmemiz ve özelliklerini kullanmamız gerekebilir. Bu durumda **Property** kavramı devreye girer. Property bir alanın değerini geri döndürmeye (**Get**) ve değerini ayarlamaya (**Set**) olanak sağlar.

Encapsulation (Kapsülleme)

Neden Kullanılır?

- Kontrolsüz veri girişi yapılmasını önler. Bunu da "property" nesneleri, "**get**" ve "**set**" metodlarıyla sağlar. Ayrıca bu get ve set metodları sayesinde, nesnelerimi sadece okunabilir ya da sadece yazılabilir tanımlayabilirim.
- Set metodu içerisinde gelen value değerini değişkene yüklemeden önce benim kriterlerime uygun olup olmadığını kontrol edebilirim. Uygun değilse yüklemeden mesajla kullanıcıyı uyarabilirim. Örneğin: yaş değişkenine rakam yerine metinsel ifade yüklenmesini engelleyebilirim.

Methods (Metodlar)

Erişim Belirleyiciler (Access Modifiers)

- **Erişim Şekli :** Programın diğer bölümlerinden (sınıflardan) metoda erişilirken bu erişim şeklinin nasıl olacağı bu bölümde belirlenir.
- Erişim Şekli **private, public, protected, static** gibi değerler alabilir.
- **private :** Bu erişim şekli belirlenen bir metot **yalnızca tanımlandığı sınıf içerisinde erişilebilir** olacaktır. Sınıf dışından erişim şekli private olan bir metoda erişilemez.
- **public :** Program içerisinde herhangi bir alanda metot çağrılabilmesi için erişim şeklinin public olarak tanımlanması gerekir.
- **protected**, bu erişim belirteci sadece sınıfın içinden erişilebileceği fakat alt sınıflara aktarılabileceği anlamına gelmektedir (**extends** ile)
- **static:** Metotlar dahil oldukları sınıf adları ile birlikte çağrılabilirler. static olarak tanımlanan bir metot ana programdan (Main()) çağrılırken sınıf adını yazmaya gerek yoktur.

Metod (Method) Ekleme

Şimdi bir örnekle erişim belirleyicilerini (access modifiers) anlatmaya çalışalım. Aslında erişim belirleyiciler miras alma (inheritance) ile birlikte daha anlamlı olmaktadır. Miras almayı ileride işleyecek olsakta burada bir örnekle ele alalım. Eğer sınıfımız başka bir sınıfın alt sınıfı (yani o sınıftan türetilmiş, o sınıfın özelliklerini miras almış) olacaksa **extends** anahtar sözcüğü ile oluşturulur.

```
class anaSinifim {
    public $ad = "anaSinifim";
    protected $id = 6124;
    private $tanimlimi = "evet";
}
class turemisSinifim extends anaSinifim {
    public function bilgiVer() {
        echo "Bu Sınıf Ana Sınıftan Türetildi";
        echo "Bu Sınıfta Tanımlanan Özellikler: \n";
        echo $this->ad . "\n";
        echo $this->id . "\n";
        echo $this->tanimlimi . "\n";
    }
}
$turet = new turemisSinifim();
$turet->bilgiVer();
```

Burada, koda baktığımızda çoğu şeyi biliyoruz tek bilmediğimiz **extends** bölümü olabilir. Burada **turemisSinifim** sınıfı **anaSinifim** sınıfından türetiliyor. Bunun anlamı **anaSinifim** sınıfı içerisindeki değişkenler ve var olsaydı metodları **turemisSinifim** sınıfına aktarıyor manasına gelmekte. Yukarıda erişim belirteclerini hatırlarsak ve bu kodu çalıştırdığımızda **anaSinifim** sınıfındaki değişkenler içinde sadece **private** $tanimlimi= “evet”; bölümünün **turemisSinifim** içerisine aktarılmadığını görüyoruz.

Kısaca, **public** heryerden ulaşılır ve aktarılır extends ile… **private**, sınıf içinden ulaşılır sadece ve extends ile **aktarılamaz**… **protected** ise sadece sınıf içinden ulaşılır ancak private ten farklı olarak extends ile de aktarılabilir.

Öğr. Gör. M. Mutlu YAPICI

Metod (Method) Ekleme

Protected erişim belirteçine sahip sınıf/nesne özelliklerine, sınıf/nesne dışından erişmek mümkün değildir. Erişim ihtiyacı için sarmalama yöntemine başvurmamız gerekmektedir. **Protected** olarak declare edilmiş sınıf/nesne özellikleri, kalıtım yolu ile alt sınıf/nesnelere aktarılabilir. Protected için bir örnek yapalım;

```
class Personel {
        public $Firma = "Kodcu A.Ş";
        public $Adres = "İstanbul/Türkiye";
        protected $TicSicilNO = "123456";
        protected $NaceKod = "855903";
}
class Isci extends Personel {
        public $Adi;
        function __construct($Adi){
                $this -> Adi = $Adi;
                }
        function getTicSicilNO(){
                return $this->TicSicilNO;
                }
        function getNaceKod(){
                return $this->NaceKod;
                }
}
```

NESNE VE ÇIKTILARI

```
$isci = new Isci("Hüseyin Akdoğan");

echo $isci->Firma."\n"; //Kodcu A.Ş çıktılanacaktır
echo $isci->Adres."\n"; //İstanbul/Türkiye çıktılanacaktır
echo $isci->TicSicilNO."\n"; //Fatal error: Cannot access protected property
echo $isci->NaceKod."\n"; //Fatal error: Cannot access protected property
echo $isci->getTicSicilNO()."\n"; //TicSicilNO bilgisi çıktılanacaktır
echo $isci->getNaceKod()."\n"; //NaceKod bilgisi çıktılanacaktır
```

Personel sınıfı/nesnesinde **protected** olarak deklare edilen **TicSicilNO** ve **NaceKod** özelliklerine, kalıtım yolu ile **Personel** sınıfından türemiş, **Isci** sınıfının bir yeni/new örneğini barındıran **isci** değişkenini kullanarak direk ulaşmak istediğimizde "Fatal error: Cannot access protected property" hatası alıyoruz. Hata mesajı ulaşmak istediğimiz özellik/property'nin **korumalı/protected** olduğunu ve erişim iznimiz bulunmadığını söylüyor. Ancak bu iki özelliğe, kendilerini sarmaladığımız (encapsulation) iki metod ile ulaştığımızda, hem kalıtım yolu ile bu iki **protected** özelliğin alt sınıf/nesmemizce miras alındığını görüyor, hem değerlerine ulaşabiliyoruz.

Metod (Method) Ekleme

Private erişim belirtecine sahip sınıf/nesne özelliklerine, yalnızca sınıf/nesne içinden erişmek mümkündür ve bu özellikler kalıtım yolu ile aktarılmazlar. **Private** erişim belirtecine sahip sınıf/nesne özelliklerine erişim ihtiyacı için sarmalama yöntemine başvurmamız gerekmektedir.. Private için bir örnek yapalım;

```
class Asker {
                private $Birligi="İstanbul";
                function getirBirligi() {
                                return $this->Birligi;
                                }
}
class Er extends Asker {
                public $Kunye;
                function kunyeOku ($Kunye){
                                $this -> Kunye = $Kunye;
                                }
}
```

NESNE VE ÇIKTILARI

```
$asker = new Asker;
echo $asker->getirBirligi(); //İstanbul bilgisi çıktılanacaktır
$er = new Er;
$er->kunyeOku("Hüseyin Akdoğan, İstanbul, 1975/4, M");
echo $er->Kunye."\n"; //Kunye bilgisi çıktılanacaktır
echo $er->Birligi."\n"; //Undefined property hatası alınacaktır
```

Metod (Method) Ekleme

Erişim belirleyicileri ile ilgili diğer bir örnek.

```
class Sınıfım
{
   public $genel = 'Genel';
   protected $korumalı = 'Korumalı';
   private $özel = 'Özel';

   function selamVer()
   {
      echo $this->genel;
      echo $this->korumalı;
      echo $this->özel;
   }
}

$nesne = new Sınıfım();
echo $nesne->genel;    // Çalışır
echo $nesne->korumalı; // Ölümcül Hata
echo $nesne->özel;     // Ölümcül Hata
$nesne->selamVer();    // Genel, Korumalı ve Özel görüntüler
```

```
/**
 * ÖbürSınıfım tanımı
 */
class ÖbürSınıfım extends Sınıfım
{
   // public ve protected yöntemleri yeniden tanımlayabiliriz,
   // ancak private olanlar tanımlanamaz
   protected $korumalı = 'Öbür korumalı';

   function selamVer()
   {
      echo $this->genel;
      echo $this->korumalı;
      echo $this->özel;
   }
}

$nesne2 = new ÖbürSınıfım();
echo $nesne2->genel;    // Çalışır
echo $nesne2->özel;     // Tanımsız
echo $nesne2->korumalı; // Ölümcül Hata
$nesne2->selamVer();    // Genel, Öbür korumalı
ve Undefined görüntülenir
```

Metod (Method) Ekleme

Metodlarda da Erişim belirleyicileri kullanılabilir.

```
/** * Sınıfım tanımı */
class Sınıfım
{
   // public kurucu bildirimi
   public function __construct() { }
   // public yöntem bildirimi
   public function Genel() { }
   // protected yöntem bildirimi
   protected function Korumalı() { }
   // private yöntem bildirimi
   private function Özel() { }
   // Bu da public bir yöntem
   function Falanca()
   {
      $this->Genel();
      $this->Korumalı();
      $this->Özel();
   }
}
```

NESNE VE ÇIKTILARI

```
$sınıfım = new Sınıfım;
$sınıfım->Genel();    // Çalışır
$sınıfım->Korumalı(); // Ölümcül Hata
$sınıfım->Özel();     // Ölümcül Hata
$sınıfım->Falanca();  // Public, Protected ve Private çalisir
```

Metod (Method) Ekleme

Bir önceki metoddan türetilmiş metod ile sonuçları tekrar görelim. Bu kodları bir önceki sınıfın altına ekliyoruz.

```
**
 * ÖbürSınıfım tanımı
 */
class ÖbürSınıfım extends Sınıfım
{
   // Bu public bir yöntemdir
   function Filanca()
   {
      $this->Genel();
      $this->Korumalı();
      $this->Özel();  // Ölümcül Hata
   }
}

$öbürSınıfım = new Sınıfım;
$öbürSınıfım->Genel();   // Çalışır
$öbürSınıfım->Filanca(); // Genel ve Korumalı çalışır, Özel çalışmaz
```

Static Metod Ekleme

Static anahtarı ile deklare edilen sınıf / nesne özellik ve metotlarına, o sınıf/nesnenin bir yeni / new örneğini oluşturmadan sınıf/nesne dışından erişmek mümkündür. Bu erişim, yine static bir metod yada etki alanı çözünürlük işleci(:: anahtarı) ile sağlanır.

```
class Dersler {
        static $Konu = "Fen Bilimleri";
         public $Not;
        function  notEkle ($Not){
                $this -> Not = $Not;
               }
 }
 echo Dersler::$Konu."\n";
echo Dersler->Konu."\n"; ///hata oluşur
```

Kodumuzda **static** olarak deklare edilmiş "Konu" adındaki sınıf/nesne özelliğine, Dersler sınıfı/nesnesinin bir yeni/new örneğini oluşturmadan, etki alanı çözünürlük **işleci(::)** ile ve **"$" sembolünü kullanarak** eriştiğimizi görüyorsunuz. **Static** olarak deklare edilmiş sınıf/nesne özelliklerine, sınıf/nesnenin bir yeni/new örneği oluşturulmadan erişebiliyor oluşumuz sebebi ile, bu özelliklere **"$this" ve "->" anahtarları ile ulaşamayız.** Aynı sebeple**"$this"** anahtarını, **static** bir **metod** içinde de kullanamayız.

Static Metod Ekleme

Static anahtarı ile deklare edilen sınıf/nesne özellik ve metotlarına, o sınıf/nesnenin bir yeni/new örneğini oluşturmadan sınıf/nesne dışından erişmek mümkündür. Bu erişim, yine static bir metod yada etki alanı çözünürlük işleci(:: anahtarı) ile sağlanır.

```
class Dersler {
        static $Konu = "Fen Bilimleri";
         public $Not;
        function  notEkle ($Not){
                $this -> Not = $Not;
               }
        public static function okuKonu(){
                return self::$Konu;
                                }
}
```

NESNE VE ÇIKTILARI

```
echo Dersler::okuKonu()."\n";
 $ders = new Dersler();
 $ders->notEkle("8")
 echo $ders->Konu; //Undefined property hatası alınacaktır
```

Kodumuzda yer alan **okuKonu** metoduna dikkat ederseniz, onun da **static** olarak deklare edildiğini ve bu sebeple Dersler sınıf/nesnesinin bir **yeni/new** örneği oluşturulmadan çağrılabildiğini görürsünüz. Metod içinde "**self**" anahtarına da dikkat edin. **Self** anahtarı isimlendirmesinden de anlayabileceğiniz gibi , sınıf/nesnenin statik özellik ve metodlarına erişim için kullanılmaktadır. Son olarak, static bir özellik yada metodu aynı zamanda **protected** yada **private** olarak da deklare edebileceğinizi ve bu static özellik ve metodların kalıtım yolu ile alt sınıf/nesnelerce miras alınacağını da belirtelim.

Yapılandırıcı (Kurucu) Metodlar

PHP 5 geliştiricilerin sınıflar için kurucu yöntemler bildirmesine imkan verir. Bir kurucu yöntemi olan sınıflar her yeni nesne oluşturuluşunda bu yöntemi çağırırlar, bu bakımdan nesne kullanılmadan önce yapılması gereken için kurucular çok uygundur. Bir nesne oluşturulduğunda **otomatik olarak** kurucu bir metod çağrılır. Kurucu metod oluşturmak için sınıf adıyla aynı adda bir metod oluşturabilir ya da metodun ismine __construct ismi verilerek kurucu metod tanımlanabilir. Yapılandırıcılar, sınıfların miras alındıkları ancak **otomatik** tetiklenecek metoddur.

```
class Urun {
                public $isim = "Masa";
                public $yapim = "X Firması";
                public $fiyat = 77;
                 public function __construct($isim,$yapim,$fiyat) {
                                $this->isim=$isim;
                                $this->yapim=$yapim;
                                $this->fiyat=$fiyat;
                                                }
                public function yazdir() {
                                echo $this->isim;
                                echo $this->yapim;
                                echo $this->fiyat;
                                  }
}
  $nesne1= new Urun("Buzdolabı","Arçelik",1250);
 $nesne1->yazdir();
```

Yukarıda kurucu metodumuzu tanımladık ve nesne oluştururken parametrelerimizi yazdık.Nesnemizin yazdir metodunu kullandıktan sonra Ekran çıktısı : **Buzdolabı Arçelik 1250**

Yapılandırıcı (Kurucu) Metodlar

Çocuk sınıflardan kurucusu olanlar için örtük olarak ebeveyn sınıfın kurucusu çağrılmaz. Ebeveyn sınıfın kurucusunu çocuk sınıftan çağırmak için, çocuk sınıf içinden **parent::__construct()** çağrısı yapılması gerekir

```
class AnaSınıf {
  function __construct() {
     print "AnaSınıf kurucusundayız\n";
  }
}
class AltSınıf extends AnaSınıf {
  function __construct() {
     parent::__construct();
     print "AltSınıf kurucusundayız\n";
  }
}
$obj = new AnaSınıf();
$obj = new AltSınıf();
```

PHP 5.3.3 itibariyle, bir **isim alanlı** sınıf isminin son elemanı ile aynı isimdeki yöntemler artık kurucu olarak ele alınmayacaktır. Bu değişiklik isim alanlı olmayan sınıfları etkilemez.

```
class Bar
{
   public function Bar()
   {
      // PHP 5.3.0-5.3.2'de kurucu olarak ele alınır
      // PHP 5.3.3 itibariyle sıradan bir yöntem olarak ele alınır
   }
```

Yıkıcı Metodlar

PHP 5, C++ gibi nesne yönelimli dillerdekine benzer bir yıkıcı tasarımına sahiptir. Yıkıcı yöntem, belli bir nesneye başka bir gönderim yoksa veya nesne kapanma sırasında açıkça yok edildiği anda çalıştırılacaktır.

```
class YıkıcıSınıf
{
  function __construct()
  {
     print "Kurucu çalıştı\n";
     $this->name = "YıkıcıSınıf";
  }
 function normalFonk ()
  {
     print " Normal Fonksiyorn (Metod) Çalışıyor\n";
  }

  function __destruct()
  {
     print "" . $this->name . " yok ediliyor\n";
  }
}
$obj = new YıkıcıSınıf();
```

Kuruculardа olduğu gibi ebeveyn yıkıcılar çocuk sınıflar için dolaylı olarak çağrılmayacaktır. Ebeveyn sınıfın yıkıcısının çalışması için çocuk sınıfın yıkıcısından **parent::__destruct()** çağrısının yapılması gerekir.

KAYNAKLAR

- İnternet ortamı
- PHP ve AJAX Haydar TUNA
- A'dan Z'ye PHP Rıza ÇELİK