BÖLÜM 2

PHP NESNEYE YÖNELİK PROGRAMLAMA

ANAHTAR KAVRAMLAR

Sınıf, Nesne, Kurucu/Yıkıcı, Soyut, Kalıtım, Arayüz, Yineleyiciler

ÖĞRENME HEDEFLERİ

1. Sınıfların ve nesneleri kullanmayı ve etkileşimlerini öğrenmek
2. Erişim belirleyicileri kullanarak sınıf özellik ve yöntemlerine erişimi kontrol etmek
3. Soyut sınıfların tanımlanmasını ve kullanılmasını öğrenmek
4. Statik yöntemlerin ve özelliklerin tanımlanmasını ve kullanılmasını öğrenmek
5. Yineleyicileri kullanarak veri koleksiyonları yönetmek

GİRİŞ

Nesne Yönelimli Programlama (Object-Oriented Programming) (OOP), yazılım geliştirme sürecinde ihtiyaçlar, nesneler ve bu nesneler arasındaki ilişkiler üzerine odaklanır. Bu yaklaşım, gerçek dünyadaki nesnelerin (objelerin) modellenmesi ve bu objeler arasındaki etkileşimlerin programlama dili içinde temsil edilmesi üzerine kuruludur. Nesne yönelimli programlamada, veri ve işlevsellik bir araya getirilerek bir nesne oluşturulur. Bu nesneler, belirli özellikleri (nitelikler veya öznitelikler) ve davranışları (metodlar veya işlevler) içerebilir. Bu sayede, programlar daha modüler, anlaşılabilir ve bakımı kolay hale gelir. Nesne yönelimli programlama, programların yeniden kullanılabilirliğini artırır ve büyük ölçekli yazılım projelerinin yönetimini kolaylaştırır.

Nesne tabanlı programlamada, nesneler özellikleri (Properties), işlevleri (Methods) ve olayları (Events) ile tanımlanır. Bu programlama paradigmalarına örnek olarak C++, C#, Java, Objective-C, PHP, Smalltalk, Python, Ruby, ABAP/4 ve Visual Basic .NET örnek olarak verilebilir.

Prosedürel programlama, işlemleri ve fonksiyonları veriler üzerinde uygulamakla ilgiliyken, nesne yönelimli programlama verileri ve fonksiyonları içeren nesneler oluşturmayı hedefler.

Nesne yönelimli programlamanın prosedürel programlamaya göre birkaç avantajı bulunmaktadır:

- NYP, daha hızlı ve daha kolay bir şekilde yürütülür.
- NYP, programlar için net bir yapı sağlar.
- NYP, PHP kodunun "Kendini Tekrar Etmeme" (DRY) ilkesine uymasına yardımcı olur ve kodun bakımını, değiştirilmesini ve hata ayıklamasını kolaylaştırır.
- NYP, daha az kod yazma ve daha kısa geliştirme süresi ile tamamen yeniden kullanılabilir uygulamaların oluşturulmasını mümkün kılar.

İpucu: "Kendini Tekrar Etmeme" (DRY) prensibi, kod tekrarını azaltmayı hedefler. Ortak kodları uygulamadan ayıklamalı, tek bir yerde tutmalı ve yeniden kullanmalısınız, böylece tekrarı önlemiş olursunuz.

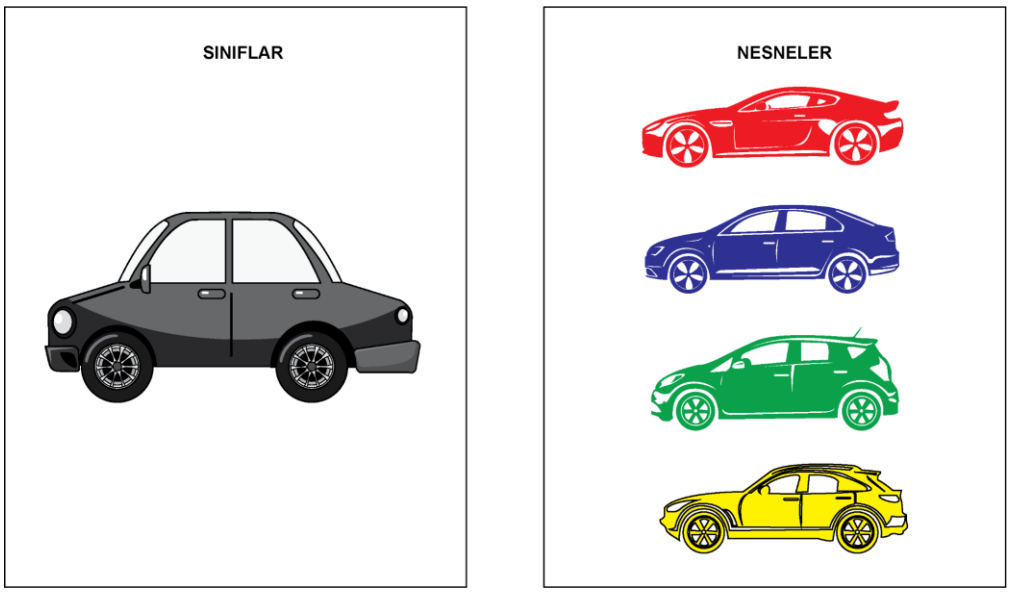
Nesne tabanlı programlama, üç temel prensibe dayanır:

- ***Kapsülleme/Paketleme*** (*Encapsulation*): Kullanıcıya gösterilmeyen kod parçalarının gizlenmesini ifade eder. Kullanıcı, bir aracın istediği sonucu alırken aracın iç işleyişiyle ilgilenmez.
- ***Kalıtım*** (*Inheritance*): Bir sınıftan yeni bir sınıf oluşturmayı ifade eder. Yeni sınıf, özelliklerini ve davranışlarını temel aldığı eski sınıftan miras alır.

- ***Çok Biçimlilik (Polymorphism):*** Aynı işlemin farklı nesneler veya ortamlar üzerinde farklı sonuçlar üretebilme yeteneğidir. Örneğin, konuşma işlemi yüksek sesle, normal ses tonuyla veya kısık sesle gerçekleştirilebilir.

SINIFLAR VE NESNELER

PHP'de, modern programlama dillerinde olduğu gibi sınıf tanımlamaları yapılabilir. PHP'nin 5. versiyonuyla birlikte sınıf tanımlama konusunda önemli değişiklikler yapılmış ve tamamen yeni bir Nesne Modeli oluşturulmuştur. Programlama dillerinde sınıflar, kendi içinde tutarlılığı olan özel işleri modellemek için kullanılır. Bu işlerin yürütülmesinden tamamen tanımlanan bu sınıf sorumlu olacaktır. Böylece, programlar gerçek hayatta olduğu gibi sınıflar aracılığıyla modellenir ve daha anlaşılır hale gelir. Sınıf tanımlamalarıyla kodlar düzenli bir şekilde yazılır ve tekrar tekrar kullanılabilir hale gelir. Örneğin, veritabanı işlemleri için bir sınıf tanımlaması yapılabilir ve veritabanıyla ilgili tüm işlemler bu sınıf üzerinden gerçekleştirilebilir. Benzer şekilde, web sayfasında kullanıcı işlemleriyle ilgili bir sınıf tanımlaması yapılabilir ve kullanıcı girişi, çıkışı, yeni kullanıcı tanımlama gibi birçok işlem tamamen bu sınıf üzerinden gerçekleştirilebilir. Aşağıdaki Resim 1'de Araba sınıfından türetilmiş fakat kendine özgü farklı özellikleri bulunan arabalar sınıf-nesne ilişkisine örnek gösterilebilir.



Resim 1: Sınıf- Nesne ilişkisi

Sınıflar içerisinde sabitler, değişkenler ve fonksiyonlar bulunabilir. Bunlara sınıfın üyeleri denilmektedir. Sınıf tanımlanırken class kelimesi kullanılır. Sınıftan türetilen her bir örnek ise nesne olarak tanımlanmaktadır. Program gereği yapılan işlemler genelde sınıfın bir örneği olan nesneler üzerinden yürütülmektedir.

Sınıf tanımı, **class** anahtar kelimesiyle başlamalıdır. Sınıf adı, **class** kelimesinden sonra gelir ve ardından iki süslü parantez "{}" içerisinde sınıfa ait özelliklerin ve yöntemlerin tanımları bulunur.

Sınıf adı olarak herhangi bir etiket kullanılabilir, ancak bu PHP için özel olarak ayrılmış bir kelime olmamalıdır. Geçerli bir sınıf adı, bir harf veya alt çizgi ile başlar ve ardından sayılar,

harfler veya alt çizgiler içerebilir. Türkçe karakterlerin kullanımında bir kısıtlama olmasa da programlamada mümkün olduğunca kullanmamak doğru olacaktır. Örnek: `Araba, _Araba, _Araba2`

PHP'de sınıf tanımlamalarında geçerli kabul edilmeyen tanımlamalar şunlardır.

- Rakamla Başlayan Sınıf İsmi: class 1Sınıf { } // Hatalı
- Özel Karakterler İçeren Sınıf İsmi: class #Sınıf { } // Hatalı
- Boşluk İçeren Sınıf İsmi: class Uzun Sınıf { } // Hatalı
- Ayrılmış PHP Anahtar Kelimeleri Kullanan Sınıf İsmi: class function { } // Hatalı

Aşağıdaki örnekte bir sınıf ve metotlar yardımıyla bir nesnenin özellikleri ekranda gösteriliyor.

```php
<?php
class Araba
{
    public $marka;
    public $model;
    public $renk;

    //kurucu metot oluşturuluyor
    public function __construct($marka, $model, $renk)
    {
        $this->marka = $marka;
        $this->model = $model;
        $this->renk = $renk;
    }
    //bir metot yardımıyla gösterim sağlanıyor
    function ozellikleriGoster()
    {
        echo "Araba Özellikleri: Marka -" . $this->marka . "Model - " . $this-
>model . "Renk -" . $this->renk;
    }
}
$araba1 = new Araba("Toyota", "Corolla", "Beyaz");
$araba1->ozellikleriGoster();
?>
```

$this Anahtar Kelimesi

$this anahtar sözcüğü geçerli nesneyi ifade eder ve yalnızca yöntemlerin içinde kullanılabilir.

Örnek:

```php
<?php
    class Meyve{
        public $meyveAdi;
    }
    $elma = new Meyve();
?>
```

Örnekte yer alan $meyveAdi özelliğinin değerini iki yöntemle değiştirebiliriz.

Yöntem 1: Sınıfın içinde (set_meyveAdi() yöntemini ekleyerek ve $this'i kullanarak):

```php
<?php
    class Meyve {
       public $meyveAdi;
       function set_meyveAdi($meyveAdi){
         $this -> meyveAdi = $meyveAdi;
       }
}
$elma = new Meyve();
$elma->set_meyveAdi("Elma");

echo $elma->meyveAdi;
?>
```

Yöntem 2: Sınıfın dışında (özellik değerini doğrudan değiştirerek):

```php
<?php
    class Meyve {
       public $meyveAdi;
}
$elma = new Meyve();
$elma->meyveAdi="Elma";

echo $elma->meyveAdi;
?>
```

Bir nesnenin belirli bir sınıfa ait olup olmadığını kontrol etmek için ***instanceof*** anahtar sözcüğü kullanılır.

```php
<?php
    class Meyve {
       public $meyveAdi;
}
$elma = new Meyve();
$elma->meyveAdi="Elma";

//echo $elma->meyveAdi;

var_dump($elma instanceof Meyve);
?>
```

new Anahtar Sözcüğü

Bir sınıfın örneğini oluşturmak için new anahtar kelimesi kullanılmalıdır. Örneğin: $ornek = new Sinif();

Nesnenin oluşturulması sırasında bir hata oluşturmaması için yapıcı metot, istisna (exception) oluşturmayacak şekilde yapılandırılmış olması gerekir. Ayrıca sınıflardan örnek

oluşturulmadan önce tanımlanmış olması gereklidir. new kelimesi ile tanımlanmış olan nesnenin özelliklerinde bellekte yeni bir nesne oluşturulur.

PHP - Kurucular (Constructor)

Sınıflar (nesneler) oluşturulduğunda otomatik çalışan metotlar yapıcı veya kurucu metot (constructor method) olarak isimlendirilirler. Her yeni örnek oluşturulduğunda kurucu metot çalıştırılır ve ilklendirme işlemleri bu yöntemle gerçekleştirilir.

PHP'de, bir sınıfın kurucusunu tanımlamak için **__construct()** metodunu kullanılır. Bu metod, sınıf adından önce çift alt çizgi ile belirtilir ve sınıfın özelliklerini başlatmak veya diğer başlatma işlemlerini gerçekleştirmek için kullanılır.

```
    //kurucu metodumuz (bir kere çalışacak)
public function __construct()
{
    echo "Lastik sayısı: " . $this->lastik . ", Vites sayısı: " . $this->vites
. "<br>";
}
```

İki alt çizgi ile başlayan kurucu metotların tanımlanmasında, zorunlu olmayan fakat ihtiyaç duyulduğu kadar parametre kullanılabilir.

```
<?php
    class Bisiklet {
        public $lastik = 2;
        public $vites = 6;
    //kurucu metodumuz (bir kere çalışacak)
    public function __construct()
    {
        echo "Lastik sayısı: " . $this->lastik . ", Vites sayısı: " . $this-
>vites . "<br>";
    }
    public function bisiklet_sur(){
        if($this->lastik >= 2 && $this->vites >= 1){
           //En az iki lastik ve bir vites varsa
            $this->lastik -=2;
            $this->vites -= 1;
            echo   "Bisiklet Sürüldü";
        }else{
            echo "Bisikletin lastik ve/veya vitesini kontrol ediniz";
        }
    }
}
$bisiklet = new Bisiklet(); // Nesne olusturduk
$bisiklet->bisiklet_sur(); // Hesne ile Slnlf icerisindeki bir metoda
ilerledik
?>
```

PHP - Yıkıcılar (Destructors)

Yıkıcılar (destructors) bir sınıftan örnek alınan nesnenin kullanımı sona erdiğinde, örneğin bellekten kaldırıldığında çalıştırılan özel metodlardır. Yıkıcı metotlar, PHP'de __destruct() olarak adlandırılır ve sınıf içinde tanımlanırlar.

Yıkıcı metotlar, örneğin sonlandırılması gereken bazı kaynakları serbest bırakmak, dosyaları kapatmak, veritabanı bağlantılarını sonlandırmak gibi işlemleri gerçekleştirmek için kullanılır. Bu, nesnenin artık ihtiyaç duyulmadığı ve programın bellek kullanımını optimize etmek için bellekten kaldırıldığı durumlarda önemli olabilir.

Bir sınıf içinde tanımlanan bir yıkıcı metot, nesnenin kullanımı sona erdiğinde otomatik olarak çağrılır ve her örneklemede yalnızca bir kez çalışır. Yıkıcı metotlar genellikle başlatıcı metotlar (constructors) ile çift olarak kullanılır. Başlatıcı metod nesnenin oluşturulduğu zaman, yıkıcı metot ise nesnenin bellekten kaldırıldığı zaman çalışır.

```php
<?php
    class Araba {
        public function __construct(){ //kurucu metot tanımı
            echo "Araba nesnesi oluşturuldu.";
        }
        public function araba_kullan(){
            echo "Araba kullanılıyor.";
        }
        public function __destruct(){//yıkıcı metot tanımı
            echo "Araba nesnesi bellekten kaldırıldı.";
        }
}
$araba = new Araba(); // Araba nesnesi oluşturuldu.
$araba->araba_kullan(); // Araba kullanılıyor.
unset($araba); // Araba nesnesi bellekten kaldırıldı.
?>
```

İpucu: Yapıcılar ve yıkıcılar kod miktarını azaltmaya yardımcı olduğundan çok faydalıdırlar!

PHP - Erişim Belirleyiciler (Access Modifiers)

Erişim belirleyiciler, nesne tabanlı programlamada kapsülleme prensibini uygulamak için kullanılan anahtar kelimelerdir. Bir sınıfın üyelerine (özellikler ve yöntemler) nerelerden erişilebileceğini belirlerler.

PHP'de 3 temel erişim belirleyici vardır:

Public: Public olarak tanımlanan bir üyeye, sınıfın kendisinden, oluşturulan nesnelerden ve bu sınıftan miras alan alt sınıflardan erişilebilir.

Private: Private olarak tanımlanan bir üyeye sadece sınıfın kendisinden erişilebilir. Dışarıdan, nesnelerden ve alt sınıflardan erişilemez.

Protected: Protected olarak tanımlanan bir üyeye, sınıfın kendisinden ve bu sınıftan miras alan alt sınıflardan erişilebilir. Dışarıdan ve nesnelerden erişilemez.

Erişim belirleyicilerde bilinmesi gereken bazı önemli noktalar ise şunlardır.

- **Varsayılan erişim belirleyici:** Bir üyenin erişim belirleyicisi belirtilmezse, varsayılan olarak **public** olur.
- **Erişim belirleyiciler ve kalıtım:** Bir sınıftan miras alan alt sınıf, üst sınıftaki **protected** üyelere erişebilir.

Erişim belirleyicileri kullanmanın faydaları:

- **Kod güvenliği:** Kapsülleme prensibini sağlayarak kodun daha güvenli ve sağlam olmasını sağlar.
- **Değişiklik yönetimi:** Kodda değişiklik yapmayı kolaylaştırır.
- **Bakım kolaylığı:** Kodun daha kolay anlaşılmasını ve bakımı yapılmasını sağlar.

```php
<?php
    class Kişi {
        public $ad; // Her yerden erişilebilir
        private $yaş; // Sadece Kişi sınıfından erişilebilir
        protected $adres; // Kişi sınıfından ve alt sınıflardan erişilebilir

        // Ad özelliğine erişim sağlayan metot
        public function getAd()
        {
            return $this->ad;
        }

        // Yaş özelliğine erişim sağlayan metot
        private function getYaş()
        {
            return $this->yaş;
        }

        // Adres özelliğine erişim sağlayan metot
        protected function getAdres()
        {
            return $this->adres;
        }
    }

    $kişi = new Kişi();
    echo $kişi->ad; // Çalışır
    echo $kişi->yaş; // Hata: 'yaş' özelliği erişilebilir değil
    echo $kişi->adres; // Hata: 'adres' özelliği erişilebilir değil
?>
```

PHP'de Kalıtım Nedir?

PHP'de kalıtım, Nesneye Dayalı Programlama'da (OOP) yeni sınıflar (alt sınıflar) oluşturmanıza ve mevcut sınıfların (üst sınıflar) özelliklerini ve yöntemlerini yeniden kullanmanıza olanak tanıyan temel bir kavramdır. Bu, kodun yeniden kullanılabilirliğini teşvik eder, yedekliliği azaltır ve kodunuzu hiyerarşik bir şekilde düzenlemenize yardımcı olur.

Kalıtımın Temel Noktaları:

- Bir alt sınıf, üst sınıfından tüm halka açık ve korunan üyeleri (özellikler ve yöntemler) devralır.
- Alt sınıf, devralınan üyelere ek olarak kendi özel üyelerine de sahip olabilir.
- Bir alt sınıfı, bir üst sınıftan devralmak için **extends** anahtar kelimesi kullanılır.

Kalıtımın Faydaları:

- ***Kod Yeniden Kullanılabilirliği***: Bir üst sınıftan ortak işlevleri devralarak aynı kodu tekrar tekrar yazmaktan kaçınabilirsiniz.
- ***Azaltılmış Yedeklilik:*** Kalıtım, kod kopyalamayı önler ve kod tabanınızı temiz tutar.
- ***Hiyerarşik İlişkiler:*** Sınıf hiyerarşileri oluşturarak nesneler arasındaki gerçek dünya ilişkilerini modellemenize olanak tanır.
- ***Polimorfizm:*** Kalıtım, farklı sınıflardaki nesneleri benzer şekilde ele almanıza olanak tanıyan polimorfizm için bir temel oluşturur.

```php
<?php
    class Hayvan
    {
        public $isim;
        public function sesCikar()
        {
            echo "Genel hayvan sesi";
        }
    }

    class Kopek extends Hayvan
    {
        public function havla()
        {
            echo "Hav!";
        }
    }

    $kopek = new Kopek();
    $kopek->sesCikar(); // Hayvan'dan devralındı
    $kopek->havla(); // Kopek'e özgü yöntem
?>
```

EK BİLGİ

PHP, **tek kalıtımı** destekler, yani bir alt sınıf doğrudan yalnızca bir üst sınıftan miras alabilir.

Miras alma zinciri oluşturarak (örneğin, torun sınıfın alt sınıftan miras alması) çok seviyeli bir miras alma yapısı oluşturabilirsiniz.

Kalıtım, alt sınıfta devralınan yöntemlerin **geçersiz kılınmasına** izin verir. Bu, devralınan bir yöntemin davranışını, alt sınıfın özel gereksinimlerine uyacak şekilde yeniden tanımlayabileceğiniz anlamına gelir.

PHP'de Kalıtım ve Korumalı Erişim Değiştirici

Korumalı Erişim Değiştirici: protected erişim değiştirici, bir sınıfın üyelerine (özellikler ve yöntemler) **sadece sınıfın kendisinden ve bu sınıftan miras alan alt sınıflardan** erişilebilmesini sağlar.

Kalıtım ile İlişkisi:

- Bir alt sınıf, bir üst sınıftan devralınan korumalı üyelere **doğrudan erişebilir**.
- Dışarıdan ve nesnelerden korumalı üyelere erişilemez.

Faydaları:

- **Kapsülleme:** Sınıfın iç işleyişini gizleyerek kodun daha güvenli ve sağlam olmasını sağlar.
- **Değişiklik Yönetimi:** Kodda değişiklik yapmayı kolaylaştırır.
- **Bakım Kolaylığı:** Kodun daha kolay anlaşılmasını ve bakımı yapılmasını sağlar.

```php
<?php
    class UstSinif
    {
        protected $isim;
        protected function getIsim()
        {
            return $this->isim;
        }
    }

    class AltSinif extends UstSinif
    {
```

```
        public function getUstSinifIsmi()
        {
            return $this->isim; // Çalışır, isim devralındı
        }
    }

    $ustSinif = new UstSinif();
    echo $ustSinif->isim; // Hata: 'isim' özelliği erişilebilir değil
    $altSinif = new AltSinif();
    echo $altSinif->isim; // Hata: 'isim' özelliği erişilebilir değil
    echo $altSinif->getUstSinifIsmi(); // Çalışır, getIsim metodu aracılığıyla
erişim
?>
```

Dikkat Edilmesi Gerekenler:

- Korumalı üyeler, alt sınıflarda **değiştirilebilir**.
- Korumalı üyeler, **sadece miras yoluyla erişilebilir**, doğrudan erişim mümkün değildir.

PHP'de Miras Yöntemleri Geçersiz Kılma

PHP'de, bir alt sınıfta **üst sınıftan devralınan bir yöntemin davranışını yeniden tanımlamak** için **geçersiz kılma** (override) işlemini kullanabilirsiniz.

Nasıl Yapılır:

- Alt sınıfta, aynı ada ve parametre listesine sahip bir yöntem tanımlanır.
- Bu yöntem, üst sınıftan devralınan yöntemi **geçersiz kılar** ve onun yerine geçer.

Aşağıda yer alan kod bir "Meyve" sınıfı oluşturur ve ondan türetilmiş bir "Çilek" sınıfı içerir. "Meyve" sınıfı meyve özelliklerini temsil ederken, "Çilek" sınıfı, meyve özelliklerine ek olarak ağırlığı da temsil eder.

```
<?php
// Meyve sınıfı tanımlanıyor
class Meyve
{
    // Meyve özellikleri
    public $isim;
    public $renk;

    // Meyve sınıfının yapıcı metodunu tanımlıyoruz
    public function __construct($isim, $renk)
    {
        // Kurucu metotta meyve özelliklerini atıyoruz
        $this->isim = $isim;
        $this->renk = $renk;
    }

    // Meyveyi tanıtan metot
```

```php
    public function tanitim()
    {
        // Meyvenin adını ve rengini ekrana yazdırıyoruz
        echo "Meyvenin adı {$this->isim} ve rengi {$this->renk}.";
    }
}

// Çilek sınıfı Meyve sınıfından türetiliyor
class Cilek extends Meyve
{
    // Çileğin ek özelliği: ağırlık
    public $agirlik;

    // Çilek sınıfının yapıcı metodunu tanımlıyoruz
    public function __construct($isim, $renk, $agirlik)
    {
        // Çilek sınıfının yapıcı metodu, Meyve sınıfının yapıcı metodunu
çağırır
        parent::__construct($isim, $renk);
        // Çileğin ağırlığını atıyoruz
        $this->agirlik = $agirlik;
    }

    // Çileği tanıtan metot
    public function tanitim()
    {
        // Çileğin adını, rengini ve ağırlığını ekrana yazdırıyoruz
        echo "Meyvenin adı {$this->isim}, rengi {$this->renk} ve ağırlığı
{$this->agirlik} gram.";
    }
}

// Çilek nesnesi oluşturuluyor
$cilek = new Cilek("Çilek", "kırmızı", 50);
// Çileği tanıtan metot çağrılıyor
$cilek->tanitim();
?>
```

Dikkat Edilmesi Gerekenler:

- Geçersiz kılma işleminde, alt sınıftaki yöntemin **imzası** (adı ve parametre listesi) **üst sınıftaki yöntemin imzasıyla tam olarak eşleşmelidir**.
- Geçersiz kılınan yöntem, **override anahtar kelimesiyle** açıkça belirtilebilir. Bu, kodun daha okunabilir ve anlaşılır olmasını sağlar.

Geçersiz Kılmanın Faydaları:

- **Kodun Esnekliği:** Farklı alt sınıflar için farklı davranışlar sağlayarak kodun daha esnek olmasını sağlar.
- **Kod Yeniden Kullanılabilirliği:** Genel işlevselliği üst sınıfta tanımlayarak ve alt sınıflarda özelleştirerek kodun yeniden kullanılabilirliğini artırır.

PHP'de final Anahtar Kelimesi

PHP'de final anahtar kelimesi, kalıtımı ve yöntem geçersiz kılmayı kısıtlayarak kodun netliğini ve istenmeyen davranışları önler. İşlevleri şunlardır:

1. final Sınıflar:

- Bir sınıf final olarak bildirildiğinde, **başka sınıflar tarafından türetilemez**.
- Bu, temel bir kavramı temsil eden veya miras yoluyla değiştirilmemesi gereken iyi tanımlanmış bir uygulaması olan sınıflar için kullanışlıdır.

```php
<?php
    final class Matematik {
        public static function toplama($a, $b)
        {
            return $a + $b;
        }
    }

    /* final olarak tanımlanmış bir sınıftan yeni bir
    sınıf türetme işlemi gerçekleştirilemez. Bu kod hata verir.
    */
    class Geometri extends Matematik {
    }
?>
```

PHP - Sınıf Sabitleri

Sınıf sabitleri, bir sınıfın tüm nesneleri tarafından paylaşılan ve değiştirilemeyen değerlerdir. Sabitler, **const** anahtar kelimesi kullanılarak tanımlanır.

- Sınıf adı ve :: operatörü kullanılarak sabite erişebilirsiniz.
- Bir nesne örneği aracılığıyla da sabite erişebilirsiniz.

Sınıf Sabitlerinin Özellikleri:

- Sabitler, bir kez tanımlandıktan sonra değiştirilemez.
- Sabitler tüm nesneler tarafından paylaşılır.
- Sabitler, statik (static) olarak da bilinir.
- Sınıf sabitleri büyük/küçük harfe duyarlıdır. Ancak sabitlerin tamamının büyük harflerle adlandırılması önerilir.
- Sabitler, basit veri türleri (string, integer, float) veya null değer alabilir.

Sınıf Sabitlerini Kullanmanın Avantajları:

- Kodun daha okunabilir ve anlaşılır olmasını sağlar.
- Tekrarlayan değerleri tek bir yerde tanımlayarak kodun daha az yer kaplamasını sağlar.
- Değişiklikleri tek bir noktadan yaparak kodun daha kolay bakımı yapılmasını sağlar.

self veya, aşağıdaki gibi , anahtar kelimeyi, ardından kapsam çözümleme operatörünü (::) ve ardından sabit adını kullanarak sınıf içinden bir sabite erişebiliriz :

```php
<?php
    class Veritabani
    {
        const HOST = "localhost";
        const KULLANICI_ADI = "admin'";
        const SIFRE = "1234";
        const VERITABANI_ADI = "phpDers";

        public function baglan()
        {
            echo "HOST : " . self::HOST . "\n";
            echo "KULLANICI_ADI: " . self::KULLANICI_ADI . "\n";
            echo "SIFRE : " . self::SIFRE . "\n";
            echo "VERITABANI_ADI: " . self::VERITABANI_ADI . "\n ";
        }
    }
    $baglanti = new Veritabani();
    $baglanti->baglan( ) ;
?>
```

PHP - Soyut Sınıflar ve Yöntemler

Soyut sınıflar ve yöntemler, PHP'de nesne yönelimli programlamada kullanılan bir kavramdır. Bir sınıf soyut olarak işaretlenebilir, yani soyut bir sınıf, kendisi örneklenemez ancak diğer sınıfların temelini oluşturabilir. Soyut sınıflar, genellikle benzer özelliklere ve davranışlara sahip nesneler için bir şablon veya arayüz sağlamak için kullanılır. Bir sınıfı soyut yapmak için **abstract** anahtar kelimesi kullanılır. Soyut sınıflar içinde soyut yöntemler tanımlanabilir. Soyut yöntemler, sınıfların bu yöntemi uygulamak zorunda olduğunu belirtir, ancak soyut sınıf içinde bu yöntemlerin gerçek bir uygulaması bulunmaz. Alt sınıflar, soyut sınıftan kalıtım alırken soyut yöntemleri uygulamak zorundadır. Bu sayede, soyut sınıflar ve yöntemler, kodun daha modüler ve esnek olmasını sağlar, çünkü alt sınıflar aynı temel davranışı paylaşabilir ve bu davranışı özelleştirebilir.

Soyut Sınıfları Tanımlama:

- **abstract** anahtar kelimesi kullanılarak tanımlanır.

- En az bir soyut yöntem içerir.
- Soyut yöntemler, gövdesiz (içeriği olmayan) olarak tanımlanır.

Soyut Yöntemler:

- Soyut yöntemler, alt sınıflarda **mutlaka** geçersiz kılınmalıdır (override).
- Geçersiz kılınan yöntemlerde, soyut yöntemin imzası (ad ve parametre listesi) eşleştirilmelidir

Aşağıdaki örnekte soyut bir sınıf Hayvan ve içerisinde soyut bir yöntem sesCikar() oluşturulmuştur. Hayvan sınıfından türetilen Kopek sınıfında, üst sınıfta yer alan soyut yöntem sesCikar() geçersiz kılınarak (override) kullanılmıştır.

```php
<?php
    abstract class Hayvan {
      abstract public function sesCikar();
      public function hareketEt () {
        echo "Genel hayvan hareketi";
      }
    }

    class Kopek extends Hayvan
    {
        public function sesCikar()
        {
            echo "Hav!";
        }
    }
    $kopek = new Kopek();
    $kopek->sesCikar() ; // "Hav!" yazdırır
?>
```

Soyut Sınıfların Avantajları:	Soyut Sınıfların Dezavantajları:
• Kodun yeniden kullanılabilirliğini artırır. • Ortak işlevselliği tek bir yerde tanımlayarak kodun daha az yer kaplamasını sağlar. • Kodun daha organize ve tutarlı olmasını sağlar.	• Somut nesneler oluşturamazlar. • Daha karmaşık bir kod yapısı oluşturabilirler.

Soyut Sınıfların Kullanım Alanları:

- Ortak bir arayüze sahip nesneler oluşturmak için.
- Kalıtım yoluyla kodun yeniden kullanılabilirliğini artırmak için.

- Kodun daha organize ve tutarlı olmasını sağlamak için.

Örnek Kullanım:

- Hayvan, Bitki gibi kategorileri soyut sınıflar ile tanımlamak.
- Veritabanı bağlantısı, dosya işlemleri gibi genel işlevleri soyut sınıflar ile implemente etmek

PHP Arayüzler

Arayüzler (interfaces), bir sınıfın belirli bir davranışı tanımlayan bir sözleşme veya şablon sağlar. Arayüzler, bir sınıfın belirli bir davranış kümesine sahip olmasını garantileyen yöntem bildirimlerini içerir. Bir sınıf, birden fazla arayüzü uygulayabilir.

Arayüzler, bir sınıfın belirli bir davranışı nasıl gerçekleştireceğini tanımlamaz, sadece hangi yöntemleri uygulaması gerektiğini belirtir. Arayüzler, kodun modülerliğini artırır ve farklı sınıfların belirli bir davranışı nasıl gerçekleştireceğini önceden belirlemesine olanak tanır.

Arayüzler, **interface** anahtar kelimesiyle tanımlanır ve sınıflar tarafından **implements** anahtar kelimesiyle uygulanır. Bir sınıf, bir veya daha fazla arayüzü uygulayabilir. Arayüzlerdeki tüm yöntemler, uygulayan sınıflar tarafından aynı isimde ve imzada (parametrelerle birlikte) uygulanmalıdır.

Arayüzleri Tanımlama:

- **interface** anahtar kelimesi kullanılarak tanımlanır.
- Yalnızca **soyut yöntemler** içerir.
- Yöntem gövdeleri tanımlanamaz.

Aşağıdaki örnekte, SesCikarabilir adında bir arayüz tanımlanıyor. Bu arayüz, sesCikar() adında bir soyut yöntem içeriyor. Hayvan adında bir sınıf SesCikarabilir arayüzünü uyguluyor. Bu sınıf, sesCikar() yöntemini "Genel hayvan sesi" yazdıracak şekilde özelleştiriyor. Köpek adında bir sınıf Hayvan sınıfından miras alıyor. Bu sınıf, sesCikar() yöntemini "Hav!" yazdıracak şekilde yeniden tanımlıyor. $kopek adında bir Köpek nesnesi oluşturuluyor. $kopek->sesCikar() fonksiyonu çağrıldığında, Köpek sınıfındaki sesCikar() yöntemi çalıştırılıyor ve "Hav!" yazdırılıyor.

```php
<?php
    interface SesCikarabilir
    {
        public function sesCikar();
    }

    class Hayvan implements SesCikarabilir
    {
        public function sesCikar()
```

```
        {
            echo "Genel hayvan sesi";
        }
    }

    class Köpek extends Hayvan
    {
        public function sesCikar()
        {
            echo "Hav!";
        }
    }
    $kopek = new Köpek();
    $kopek->sesCikar(); // "Hav!" yazdırır
?>
```

Arayüzlerin Avantajları	Arayüzlerin Dezavantajları
• Kodun yeniden kullanılabilirliğini artırır. • Sınıflar arasında tutarlı bir davranış sağlar. • Kodun daha organize ve tutarlı olmasını sağlar.	• Somut nesneler oluşturamazlar. • Daha karmaşık bir kod yapısı oluşturabilirler.

Arayüzlerin Kullanım Alanları:

- Farklı tiplerde nesneler arasında ortak bir davranış tanımlamak için.
- Kalıtım yoluyla kodun yeniden kullanılabilirliğini artırmak için.
- Kodun daha organize ve tutarlı olmasını sağlamak için.

Örnek Kullanım Alanı:

- Ödeme, kimlik doğrulama gibi genel işlevleri arayüzler ile implemente etmek.
- Farklı veri kaynaklarına erişmek için arayüzler ile soyutlama katmanı oluşturmak.

Arayüzler ve Soyut Sınıflar Arasındaki Farklar:

- Arayüzler yalnızca soyut yöntemler içerirken, soyut sınıflar hem soyut hem de somut yöntemler içerebilir.
- Bir sınıf birden fazla arayüzü uygulayabilirken, tek bir soyut sınıftan miras alabilir.

PHP - Nitelikler (Traits)

PHP'de nitelikler, birden fazla sınıf tarafından **paylaşılabilen** bir dizi **yöntem ve özelliği** tanımlayan bir yapıdır. Özellikler, kodun **yeniden kullanılabilirliğini** ve **tutarlılığını** artırmak için kullanılır.

Nitelikler, **trait** anahtar kelimesiyle tanımlanır ve **use** anahtar kelimesiyle sınıflara eklenir. Bir sınıf, bir veya daha fazla trait kullanabilir.

Aşağıdaki örnekte, Yuzme ve Kosma adında iki trait tanımlanmıştır. Her biri birer metod içerir. Insan sınıfı, bu iki traiti kullanır ve böylece yuz() ve kos() metodlarına erişebilir. Traitler, kodun parçalanmasını ve yeniden kullanılabilirliğini artırır, böylece kod tekrarından kaçınmamıza yardımcı olur.

```php
<?php
    trait Yuzme {
        public function yuz() {
            echo "Yüzüyorum...";
        }
    }

    trait Kosma {
        public function kos() {
            echo "Koşuyorum...";
        }
    }

    class Insan {
        use Yuzme, Kosma;
    }

    $insan = new Insan();
    $insan->yuz(); // Yüzüyorum...
    $insan->kos(); // Koşuyorum...
?>
```

Niteliklerin Avantajları	Niteliklerin Dezavantajları
• Kodun yeniden kullanılabilirliğini artırır. • Sınıflar arasında tutarlı bir davranış sağlar. • Kodun daha organize ve tutarlı olmasını sağlar.	• Karmaşıklığı artırabilir. • İsimlendirme çakışmalarına yol açabilir.

Niteliklerin Kullanım Alanları:

- Ortak işlevselliği birden fazla sınıf arasında paylaşmak için.

- Kodun daha modüler ve esnek olmasını sağlamak için.
- Sınıflar arasında kod tekrarını önlemek için.

Nitelikler ve Sınıflar Arasındaki Farklar:

- Nitelikler, doğrudan nesne oluşturamazken, sınıflar nesne oluşturabilir.
- Nitelikler, yalnızca yöntem ve özellikler içerirken, sınıflar bunlara ek olarak sabitler ve yapıcılar da içerebilir.
- Nitelikler, birden fazla sınıf tarafından kullanılabilirken, bir sınıf tek bir sınıftan miras alabilir.

PHP Statik Yöntemler

Statik yöntemler, **sınıfın bir örneği oluşturulmadan** doğrudan **sınıf adı üzerinden** çağrılabilen yöntemlerdir. Statik yöntemler, genellikle **nesneye ait olmayan** ve **sınıfın genel işlevselliğini** temsil eden işlemler için kullanılır.

Statik yöntemler, **static** anahtar kelimesiyle tanımlanır ve normal sınıf yöntemlerinden farklı olarak **$this** anahtar kelimesine erişemezler. Bunun yerine, sınıf adını (**self** anahtar kelimesiyle) veya mevcut sınıfın adını (**static** anahtar kelimesiyle) kullanırlar.

Bir sınıf hem statik hem de statik olmayan yöntemlere sahip olabilir. Statik bir yönteme aynı sınıftaki bir yöntemden **self** anahtar sözcüğü ve iki nokta üst üste (**self::**) kullanılarak erişilebilir. Bir alt sınıftan statik bir yöntem çağırmak için, alt sınıfın içinde **parent** anahtar sözcüğü kullanılır (**parent::**).

Aşağıda verilen örnekte; Islemci::toplama() ve Islemci::daireAlani() statik yöntemleri, nesne örneği oluşturmadan doğrudan sınıf adı üzerinden çağrılır. Islemci::$pi statik özelliği, self anahtar kelimesi kullanılarak statik yöntemler içerisinde erişilebilir. $x ve $y değişkenleri, statik yöntemlere parametre olarak gönderilir. Örnekteki statik yöntemler ve statik özellikler public olarak tanımlanmıştır. Private veya protected olarak da tanımlanabilirler.

```php
<?php
    class Islem {
    static public $pi = 3.14;
    static public function toplama($a, $b) {
      return $a + $b;
    }

    static public function cikarma($a, $b) {
      return $a - $b;
    }

    static public function daireAlani($r) {
      return self : :$pi * $r * $r;
    }

    echo Islem::toplama (5, 10); // 15 yazdırır
    echo "\n" ; // alt satıra geçirir
    echo Islem::cikarma (10, 5); // 5 yazdırır
```

```php
    echo "\n" ; // alt satıra geçirir
    $x = 10;
    $y = 20 ;
    echo Islem::toplama ($x, $y); // 30 yazdırır
?>
```

Statik Yöntemlerin Avantajları	Statik Yöntemlerin Dezavantajları
• Nesne oluşturma ihtiyacını ortadan kaldırarak kodun daha az yer kaplamasını sağlar. • Sınıfın genel işlevselliğini daha organize bir şekilde sunar. • Bellek kullanımını optimize etmeye yardımcı olabilir.	• Nesneye ait olmayan işlemler için uygun değildir. • Kalıtım ile aktarılamaz. • $this değişkenine erişemedikleri için daha az esnektir.

Statik Yöntemlerin Kullanım Alanları:

- Yardımcı fonksiyonlar ve matematiksel işlemler için.
- Sınıfın genel ayarlarını ve konfigürasyonlarını yönetmek için.
- Fabrika desenleri gibi tasarım örüntülerinde.

Statik Özellikler:

- Statik yöntemler gibi, statik özellikler de nesne örneği oluşturulmadan kullanılabilir.
- Sınıfın tüm nesneleri tarafından paylaşılan bir veri saklamak için kullanılır.

PHP - Statik Özellikler

Statik özellikler, **sınıfın tüm nesneleri tarafından paylaşılan** ve **nesne örneği oluşturmadan** erişilebilen veri değerleridir. Statik özellikler, **sınıfın genel durumunu** veya **ayarlarını** temsil eden verileri saklamak için kullanılır.

Statik bir özelliğe, aynı sınıftaki bir yöntemden **self** anahtar sözcüğü ve iki nokta üst üste (**self::**) kullanılarak erişilebilir. Bir alt sınıftan statik bir özellik çağırmak için, alt sınıfın içinde **parent** anahtar sözcüğü kullanılır (**parent::**).

Aşağıdaki örnekte; Islemci sınıfında $pi adında bir **statik özellik** tanımlanmıştır. Bu özellik, public olarak belirlenmiş ve 3.14 değerine atanmıştır. daireAlani() adında bir **statik yöntem** tanımlanmıştır. Bu yöntem, $r parametresini kullanarak bir dairenin alanını hesaplar. Islemci::$pi ifadesi, Islemci sınıfının $pi **statik özelliğine** doğrudan erişmek için kullanılır. Islemci::daireAlani(5) ifadesi, Islemci sınıfının daireAlani() **statik yöntemini** çağırmak için kullanılır. Yöntem, 5 değerini parametre olarak alır. $daireAlani değişkeni, daireAlani() yönteminin **dönüş değerini** saklamak için kullanılır.

```php
<?php
    class Islem
    {
        static public $pi = 3.14;
        static public function daireAlani($r)
        {
            return self::$pi * $r * $r;
        }
    }
    echo Islem::$pi; // 3.14 yazdırır
    echo "\n" ; // alt satıra geç
    $daireAlani = Islem::daireAlani(5) ;
    echo $daireAlani; // 78.5 yazdırır
?>
```

Statik Özelliklerin Avantajları	Statik Özelliklerin Dezavantajları
• Sınıfın tüm nesneleri arasında veri paylaşımı sağlar. • Kodun daha az yer kaplamasını sağlar. • Sınıfın genel durumunu ve ayarlarını yönetmek için daha kolay bir yol sunar.	• Nesneye ait olmayan veriler için uygun değildir. • Kalıtım ile aktarılamaz.

Statik Özelliklerin Kullanım Alanları:

- Sınıfın genel ayarlarını ve konfigürasyonlarını saklamak için.
- Sabit değerleri saklamak için.
- Sayaç ve istatistik gibi sınıf genelinde paylaşılan verileri saklamak için.

Statik Yöntemlerle İlişkisi:

- Statik özellikler, statik yöntemler tarafından **self** anahtar kelimesi kullanılarak erişilebilir.
- Statik yöntemler ve statik özellikler birlikte kullanılarak, sınıfın genel işlevselliği daha organize bir şekilde sunulabilir.

PHP Ad Alanları

PHP'de **ad alanları (namespaces)**, bir kodun içindeki sınıf, fonksiyon ve sabit isimlerini organize etmek için kullanılan bir mekanizmadır. Ad alanları, farklı kütüphanelerin veya bileşenlerin bir arada kullanılmasını kolaylaştırır ve çakışmaları önler. Ad alanları, kodu sınıflar, arayüzler, işlevler ve sabitleri kategorilere ayırır.

Neden Ad Alanı Kullanmalıyız?

- **Kod Organizasyonu:** Kodunuzu mantıksal birimlere ayırarak daha okunabilir ve bakımı kolay hale getirebilirsiniz.
- **İsim Çakışmalarını Önleme:** Farklı kütüphaneler veya framework'ler aynı sınıf, işlev veya sabit isimlerini kullanabilir. Ad alanları ile kendi kodunuzun isimlerini, bu kütüphanelerin isimlerinden ayırt edebilirsiniz.
- **Kod Yeniden Kullanımı:** Ad alanları ile kodunuzu hiyerarşik bir şekilde organize ederek, kod parçalarını farklı projelerde daha kolay yeniden kullanabilirsiniz.

Ad Alanı Tanımlama:

- namespace anahtar kelimesi kullanılarak tanımlanır.
- Ad alanları birbirlerine nokta (.) ile bağlanarak hiyerarşik bir yapı oluşturabilirsiniz.

Aşağıdaki örnekte; geometri isimli bir ad alanı oluşturulmuştur. Kare adında bir sınıf ve alanHesapla adında bir metot kullanarak bir karenin alanını hesaplar. pi özelliği ise tüm nesneler tarafından paylaşılan sabit bir değerdir. use anahtar kelimesi ile tanımlanan ad alanından bir sınıf çağrılmıştır.

```php
<?php
    namespace geometri;

    class Kare
    {
        static public $pi = 3.14;

        public function alanHesapla($kenar)
        {
            return self::$pi * $kenar * $kenar;
        }
    }
    use geometri\Kare;

    $kare = new Kare();
    $alan = $kare->alanHesapla(S);

    echo "Alan : " . $alan . " (pi : " . Kare::$pi . ")"; //78.5 (pi:
3.14)yazdırır
?>
```

Ad Alanı Takma Adı

Yazmayı kolaylaştırmak için bir ad alanına veya sınıfa takma ad vermek yararlı olabilir. Bu **use** anahtar sözcüğü ile yapılır. Alan adı yazıldıktan sonra **as** anahtar kelimesi ile takma ad verilir.

```php
<?php
    use Html as H;
    $table = new H\Table();
?>
```

PHP Yineleyiciler (Iterable)

PHP 7.1'den itibaren, **iterable** (yineleyiciler) pseudo-tipi, **foreach** döngüsü ile döngüye girebileceğiniz herhangi bir değeri temsil eder. Bu, diziler, nesneler, generatorler ve Traversable (dizilere benzer şekilde yinelemeye izin veren nesneler) arayüzünü uygulayan herhangi bir şey dâhil olmak üzere birçok veri türünü kapsar.

Iterable Oluşturma:

- **Diziler:** Herhangi bir dizi iterable'dır.
- **Nesneler:** Bir nesnenin iterable olması için Traversable arayüzünü uygulaması veya __iterator yöntemini tanımlaması gerekir.
- **Generatorler:** yield anahtar sözcüğünü kullanarak generator fonksiyonları iterable oluşturur.

Iterable Kullanımı:

- **foreach** döngüsü ile iterable'ları kolayca döngüye alabilirsiniz.
- **iterable** pseudo-tipini fonksiyon parametresi veya dönüş değeri olarak kullanabilirsiniz.
- **is_iterable()** fonksiyonu, bir değerin iterable olup olmadığını kontrol etmek için kullanılabilir.
- **count()** fonksiyonu, iterable'daki öğelerin sayısını döndürmek için kullanılabilir.
- **foreach** döngüsü, iterable'daki öğeleri key-value çiftleri olarak da döndürebilir.

Iterables, Iterator arayüzünü uygulayan veya __iterator yöntemini tanımlayan nesnelerdir. Bu arayüz, iterasyon (döngü) için gerekli olan bazı yöntemleri tanımlar.

Iterator Yöntemleri:

Her iterable, aşağıdaki temel Iterator arayüzü yöntemlerini uygulayarak verilere erişebilir:

- **current()**: Döngünün şu anki öğesini döndürür.
- **key()**: Döngünün şu anki öğesinin anahtarını döndürür (diziler için indeks).
- **next()**: Döngüyü bir sonraki öğeye taşır.
- **valid()**: Döngünün geçerli bir öğeye işaret edip etmediğini kontrol eder (true veya false döndürür).
- **rewind()**: Başlangıca geri döndürür.

Iterable'ların Avantajları	Iterables'ın Dezavantajları
• Daha genel bir kod yazımı sağlar. • Farklı veri türleri üzerinde aynı döngü yapısını kullanabilirsiniz. • Kodunuzu daha okunabilir ve anlaşılır hale getirir.	• Performans: Bazı durumlarda, iterable'lar doğrudan döngüye alınan veri türlerinden daha az performanslı olabilir. • Geriye dönük uyumluluk: PHP 7.1'den önce yazılmış kodlar iterable'ları desteklemez. • Hata ayıklama: Iterable'larda hata ayıklamak, doğrudan döngüye alınan veri türlerine göre daha zor olabilir.

Aşağıdaki örnekte: Sinif adında bir sınıf, Iterator arayüzünü kullanarak iterable hale getirilir. degerler dizisi, pozisyon ile takip edilir. __construct metodu, degerleri başlatır. current, key, next ve valid metotları, foreach döngüsü ile degerler üzerinde döngüye girilmesini sağlar. rewind metodu opsiyoneldir. foreach döngüsü, her öğenin anahtarını ve değerini ekrana yazdırır.

```php
<?php
    class Sinif implements Iterator {
        private $degerler = [];
        private $pozisyon = 0;

        public function _construct () {
           $this->degerler = [1, 2, 3, 4, 5];
        }

        public function current() {
```

```
            return $this->degerler [$this->pozisyon];
        }

        public function key() {
            return $this->pozisyon;
        }

        public function next() {
            $this->pozisyon++ ;
        }

        public function valid () {
            return isset($this->degerler[$this->pozisyon]) ;
        }

        public function rewind() {
            $this->pozisyon = 0;
        }
    }
  $sinif = new Sinif ();
  foreach ($sinif as $key => $deger) {
    echo "Değer ($key): " . $deger , "<br>";

?>
```

BÖLÜM ÖZETİ

Nesne tabanlı programlama yani NYP, nesnelerin özellikleri, işlevleri ve olayları ile tanımlanır ve si plus plus, Java gibi dillerde kullanılır. Prosedürel programlamadan farklı olarak, veri ve fonksiyonları bir arada tutarak daha düzenli ve yeniden kullanılabilir kod yapısı sağlar. NYP'nin avantajları arasında hızlı yürütme, net yapı, "Kendini Tekrar Etmeme" ilkesine uyum ve geliştirme süresinin kısalığı yer alır. Temel prensipleri kapsülleme, kalıtım ve çok biçimliliktir. bunlar sırasıyla kodun gizlenmesi, sınıflar arası özellik aktarımı ve aynı işlemin farklı nesnelerde farklı sonuçlar vermesini sağlar. PHP'de sınıflar, belirli işlevleri yerine getiren ve kodun düzenli ve tekrar kullanılabilir olmasını sağlayan yapılar olarak tanımlanır. PHP'5 ile geliştirilen yeni Nesne Modeli sayesinde, veritabanı işlemleri gibi spesifik görevler için sınıflar oluşturulabilir. Sınıflar, sabitler, değişkenler ve fonksiyonlar içerir ve class anahtar kelimesiyle tanımlanır. Her sınıf örneği bir nesne olarak adlandırılır ve new anahtar kelimesiyle oluşturulur. Sınıf adları, PHP için ayrılmış kelimeler olmamalı ve belirli kurallara uygun olmalıdır. dolar this anahtar sözcüğü, yöntemler içinde geçerli nesneyi temsil eder. Sınıfların oluşturulmasıyla otomatik olarak çalışan yapıcı metotlar çoklu parantez içinde construct ve nesnelerin bellekten kaldırılmasıyla çalışan yıkıcı metotlar çoklu parantez içinde destruct bulunur. Yapıcı metotlar, sınıf örneklerinin başlatılması için kullanılırken, yıkıcı metotlar kaynakların serbest bırakılması ve belleğin temizlenmesi işlemleri için devreye girer. Her ikisi de, sınıf tanımlamalarının verimliliğini ve kodun yönetilebilirliğini artırır. Erişim belirleyiciler yani public, private, protected, sınıf üyelerinin erişilebilirlik düzeyini tanımlar. Public üyelere her yerden, private üyelere sadece tanımlandıkları sınıf içinden, protected üyelere ise tanımlandıkları sınıf ve miras alan sınıflardan erişilebilir. Varsayılan olarak, belirleyici belirtilmezse üyeler public olur. Erişim belirleyicileri, kod güvenliğini artırır, değişiklik yönetimini ve bakımını kolaylaştırır.

PHP'de kalıtım, yeni sınıfların mevcut sınıfların özelliklerini ve yöntemlerini devralmasını sağlayan bir OOP kavramıdır. Kalıtım, kodun yeniden kullanılabilirliğini artırır, yedekliliği azaltır ve hiyerarşik düzenlemeyi kolaylaştırır. PHP, tek kalıtımı destekler ve extends anahtar kelimesiyle alt sınıflar üst sınıflardan türetilir. Kalıtım, kod güvenliği ve bakım kolaylığı sağlar, değişiklik yönetimini kolaylaştırır ve polimorfizm için temel oluşturur. Korumalı erişim değiştiricisi yani protected, sınıf üyelerinin sadece sınıfın kendisinden ve miras alan alt sınıflardan erişilebilir olmasını sağlar. Miras yöntemleri geçersiz kılma, alt sınıfların üst sınıftan devralınan yöntemlerin davranışını yeniden tanımlamasına olanak tanır. final anahtar kelimesi, bir sınıfın veya yöntemin başka sınıflar tarafından türetilmesini veya geçersiz kılınmasını engeller. Sınıf sabitleri, const anahtar kelimesi ile tanımlanan ve sınıfın tüm nesneleri tarafından paylaşılan değiştirilemez değerlerdir. Sabitlere, sınıf adı ve çift iki nokta üst üste operatörü kullanılarak veya bir nesne örneği üzerinden erişilebilir. Sabitler, değiştirilemez, tüm nesneler tarafından paylaşılan ve genellikle büyük harflerle adlandırılan statik değerlerdir. Sabitlerin kullanımı, kodun okunabilirliğini ve anlaşılırlığını artırır, tekrarlayan değerlerin merkezi bir noktada yönetilmesini sağlar ve kod bakımını kolaylaştırır. Sabitlere sınıf içinden self anahtar kelimesi ve kapsam çözümleme operatörü parantez içinde çift iki nokta üst üste ile erişilir. Soyut sınıflar ve yöntemler, nesne yönelimli programlamada kullanılır.

Soyut sınıflar, abstract anahtar kelimesiyle tanımlanır ve kendileri örneklenemezler ancak diğer sınıflar için bir temel oluştururlar. Soyut yöntemler, alt sınıfların uygulaması gereken, gövdesiz yöntemlerdir. Alt sınıflar, soyut sınıflardan miras alırken bu soyut yöntemleri gerçekleştirmek zorundadır, böylece kod daha modüler ve esnek hâle gelir. Arayüzler, sınıfların uygulaması gereken yöntemleri tanımlayan sözleşmelerdir. Inter face anahtar kelimesiyle tanımlanır ve sınıflar tarafından implements ile uygulanır. Arayüzler, bir sınıfın belirli davranışları nasıl gerçekleştireceğini değil, hangi yöntemleri içermesi gerektiğini belirtir, böylece kodun modülerliğini ve esnekliğini artırır. Arayüzler yalnızca soyut yöntemler içerir ve bu yöntemlerin gövdeleri tanımlanamaz; uygulayan sınıflar bu yöntemleri kendi içerikleriyle doldurmalıdır.

PHP'de, nitelikler yani traits, birden fazla sınıf arasında paylaşılan yöntem ve özellikleri tanımlayan yapılar olarak kullanılır. trait anahtar kelimesiyle tanımlanır ve use ile sınıflara dâhil edilir. Nitelikler, kodun modülerliğini ve esnekliğini artırırken, sınıflar arasında işlevsellik paylaşımını sağlar ve kod tekrarını azaltır. Nitelikler doğrudan nesne oluşturamaz ve yalnızca yöntem ve özellikler içerir, sınıflar ise sabitler ve yapıcılar da dâhil olmak üzere daha geniş bir yelpazede özelliklere sahip olabilir. Bir sınıf, birden fazla niteliği kullanabilirken, yalnızca bir sınıftan miras alabilir. Statik yöntemler ve özellikler, sınıf örnekleri olmadan kullanılabilen sınıf bileşenleridir. Statik yöntemler, dolar this yerine "self" veya "static" anahtar kelimeleri ile sınıfın kendisine erişirken, statik özellikler sınıfın tüm örnekleri arasında paylaşılan verileri saklar. Her ikisi de self çift iki nokta üst üste veya parent çift iki nokta üst üste kullanılarak çağrılabilir ve sınıfın genel işlevselliği için kullanılır. PHP'de ad alanları, kodun düzenli ve çakışmasız bir şekilde organize edilmesini sağlayan bir yapıdır. Ad alanları sayesinde, farklı kütüphanelerden gelen benzer isimlerdeki sınıf, fonksiyon ve sabitler arasında ayrım yapılabilir. Kodun okunabilirliğini ve bakımını kolaylaştırır, aynı zamanda kodun farklı projelerde yeniden kullanılmasına olanak tanır. Ad alanları namespace anahtar kelimesiyle tanımlanır ve use ile takma ad verilebilir, bu da kod yazımını daha da basitleştirir. PHP'de yinelebilirler yani iterable, PHP'7.1 ile tanıtılmış olup, foreach döngüsünde kullanılabilen her türlü değeri ifade eder.

GÖZDEN GEÇİRELİM

1. Aşağıdaki kod parçasıyla ilgili olarak hangi ifade doğrudur?

```php
<?php
class Hayvan {
	public $name;
	public function __construct($name) {
	$this->name = $name;
	}
	public function sound() {
	return "hav hav!";
	}
}

class Kopek extends Hayvan {
	public function sound() {
	return "hav!";
	}
}
$dog = new Dog("Karabaş");
echo $dog->name . $dog->sound()." dedi." ;
```

A) "Karabaş hav hav! dedi." çıktısı verir.

B) "Karabaş hav! dedi." çıktısı verir.

C) Kod hata verir.

D) "dedi. hav! Karabaş" çıktısı verir.

E) "hav! dedi. Karabaş" çıktısı verir.

2. PHP'de bir sınıfın constructor metodu tanımlanırken hangi isim kullanılır?

A) setup()

B) __init()

C) constructor()

D) __construct()

E) create()

3. Aşağıdakilerden hangisi PHP’de kalıtımı engelleyen anahtar kelimedir?

A) final

B) static

C) private

D) protected

E) public

4. PHP’de bir sınıfın başka bir sınıftan türemesi için hangi anahtar kelime kullanılır?

A) extends

B) implements

C) inherits

D) uses

E) requires

5. Bir abstract sınıf içindeki abstract metodlar ne tür erişim belirleyicilere sahip olabilir?

A) Yalnızca private

B) Yalnızca public veya protected

C) Yalnızca public

D) Her türlü erişim belirleyiciye sahip olabilir

E) Yalnızca protected

6. PHP’de bir arayüz tanımlamak için hangi anahtar kelime kullanılır?

A) class

B) struct

C) interface

D) function

E) implement

7. Aşağıdaki kod parçasında, $obj nesnesi hangi sınıfın trait'ini kullanır?

```php
<?php
trait MyTrait {
    public function sayHello() {
    echo "Hello!";
    }
}

class MyClass {
use MyTrait;
}
$obj = new MyClass();
$obj->sayHello();
?>
```

A) MyTrait

B) MyClass

C) MyClassTrait

D) sayHello

E) HelloTrait

8. Bir namespace içindeki sınıf veya fonksiyonlara erişmek için hangi operatör kullanılır?

A) ::

B) .

C) ->

D) =>

E) ::

CEVAP ANAHTARI

1	2	3	4	5	6	7	8
B	D	A	A	B	C	A	A

KAYNAKÇA

[1] Learn to code. W3Schools Online Web Tutorials. (n.d.). https://www.w3schools.com/

FAYDALI KAYNAKLAR VE INTERNET KAYNAKLARI

Dimes, T., & L., C. (2017). PHP. Babelcube Inc.

PHP and JSON. (n.d.). https://www.w3schools.com/php/php_json.asp

PHP Basics. (n.d.). Beginning PHP and MySQL, 55–112. https://doi.org/10.1007/978-1-4302-0299-8_3

PHP - OOP (n.d.). https://www.w3schools.com/php/php_oop_what_is.asp

PHP - Class/Object Information (n.d.). https://www.php.net/manual/en/book.classobj

PHP - Construct Functions (n.d.). https://www.w3schools.com/php/php_oop_constructor.asp

PHP - Destruct Functions (n.d.). https://www.w3schools.com/php/php_oop_destructor.asp

PHP - Access Modifiers (n.d.). https://www.w3schools.com/php/php_oop_access_modifiers.asp

PHP - Inheritance (n.d.). https://www.w3schools.com/php/php_oop_inheritance.asp

PHP - Traits? (n.d.). https://www.w3schools.com/php/php_oop_traits.asp

PHP - Static Methods (n.d.). https://www.w3schools.com/php/php_oop_static_methods.asp

PHP - Static Properties (n.d.). https://www.w3schools.com/php/php_oop_static_properties.asp

PHP - Namespaces (n.d.). https://www.w3schools.com/php/php_namespaces.asp

PHP - Iterables (n.d.). https://www.w3schools.com/php/php_iterables.asp