



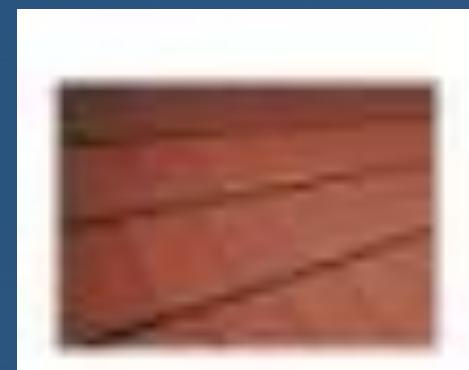
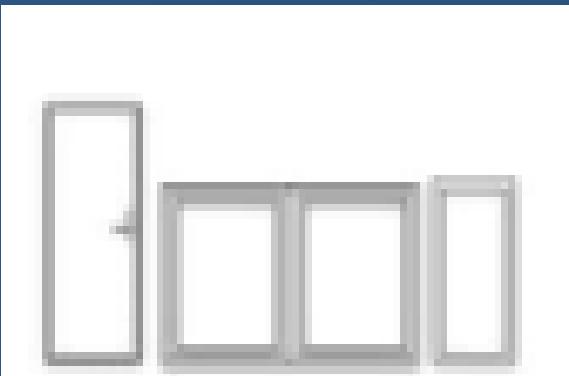
OBJECT ORIENTED PROGRAMMING

OOP



OOP NEDİR?

- Nesne Yönelimli Programlama



Gerçek Hayat Mantığı

- Bir bina yapılacak:
 - **Tuğla** ayrı
 - **Pencere** ayrı
 - **Beton** ayrı
 - ➔ **Hepsini tek dosyada karışık yazmak yerine,**
➔ **Her parçayı ayrı sınıf (class) olarak tanımlarız.**



Tugla Sınıfı ve Üretilmesi

```
class Tugla {  
    public function uret() {  
        echo "1 adet tuğla üretildi<br>";  
    }  
}
```

```
$tugla = new Tugla();  
  
for ($i = 1; $i <= 100; $i++) {  
    $tugla->uret();  
}
```



Beton Sınıfı ve Üretilmesi

```
class Beton {  
    public function dok() {  
        echo "Beton döküldü<br>";  
    }  
}
```

```
$beton = new Beton();  
$beton->dok();
```

Pencere Sınıfı ve Üretilmesi

```
class Pencere {  
    public function tak() {  
        echo "Pencere takıldı<br>";  
    }  
}
```

```
$pencere = new Pencere();  
$pencere->tak();
```

```
class Bina {  
    public function insaEt() {  
  
        $beton = new Beton();  
        $beton->dok();  
  
        $tuğla = new Tuğla();  
        for ($i = 1; $i <= 100; $i++) {  
            $tuğla->uret();  
        }  
  
        $pencere = new Pencere();  
        $pencere->tak();  
  
        echo "Bina tamamlandı ✅";  
    }  
}
```

```
$bina = new Bina();  
$bina->insaEt();
```

```
<?php  
  
echo "İnşaat başladı<br>";  
  
// Beton  
echo "Çimento alındı<br>";  
echo "Kum alındı<br>";  
echo "Su eklendi<br>";  
echo "Beton karıştırıldı<br>";  
echo "Beton döküldü<br><br>";  
  
// 100 tane tuğla  
echo "1. tuğla üretildi<br>";  
echo "2. tuğla üretildi<br>";  
echo "3. tuğla üretildi<br>";  
echo "4. tuğla üretildi<br>";  
echo "5. tuğla üretildi<br>";  
// ...  
// (bunu 100 kere yazdığını düşün)  
echo "100. tuğla üretildi<br><br>";  
  
// Pencere  
echo "Cam kesildi<br>";  
echo "Çerçeve hazırlandı<br>";  
echo "Pencere takıldı<br><br>";  
  
// İkinci kat yapılacak  
echo "Çimento alındı<br>";  
echo "Kum alındı<br>";  
echo "Su eklendi<br>";  
echo "Beton karıştırıldı<br>";  
echo "Beton döküldü<br><br>";  
  
echo "1. tuğla üretildi<br>";  
echo "2. tuğla üretildi<br>";  
// tekrar tekrar...  
echo "100. tuğla üretildi<br><br>";  
  
echo "Cam kesildi<br>";  
echo "Çerçeve hazırlandı<br>";  
echo "Pencere takıldı<br><br>";  
  
echo "İnşaat bitti";
```

OOP

◆ KONU	✓ OOP VARKEN (Avantajları)	✗ OOP YOKKEN (Zararları)
Kod Düzeni	Kodlar sınıflara ayrıılır, temiz ve düzenlidir	Kodlar tek dosyada karışır
Tekrar Kullanım	Aynı sınıf defalarca kullanılır	Aynı kod tekrar tekrar yazılır
Bakım	Değişiklik tek yerden yapılır	Her yerde tek tek düzeltme gereklidir
Okunabilirlik	Kod ne yaptığı anlaşılır	Kod karmaşık ve anlaşılmaz olur
Hata Riski	Hata yapma ihtimali düşüktür	Tekrar yüzünden hata riski artar
Genişletilebilirlik	Yeni özellik kolay eklenir	Yeni özellik eklemek zordur
Takım Çalışması	Herkes ayrı sınıf üzerinde çalışabilir	Kod çakışmaları olur
Profesyonellik	Framework'lerle uyumludur	Framework kullanımı zor
Performans Yönetimi	Yapı kontrollüdür	Kontrolsüz büyür
Gerçek Hayat Modelleme	Gerçek dünyaya uygundur	Gerçek hayatla bağı zayıf

SINIF TANIMLAMA

Tuğla Sınıfı

en: 10
boy: 15
yükseklik:8

Özellikler; en- boy-yükseklik
İşlevler; hacim

en: 20
boy: 30
yükseklik:10

d₁
d₂
d₃

en: 5
boy: 7
yükseklik:3

Tugla Sınıfını Oluşturalım

```
<?php
class Tugla {
    //özellikler
    public $en=10;
    public $boy=7;
    public $yükseklik=5;
    // Hacim hesaplayan metot
    public function hacim() {
        return $this->en * $this->boy * $this->yükseklik;
    }
}
$tugla1 = new Tugla(); // en, boy, yükseklik (cm)
echo "Tuğlanın hacmi: " . $tugla1->hacim() . " cm3";
?>
```

Aynı Safya İçinde Nesne Üretmek Mantıklı Değil!!

```
/*
proje/
└── classes/
    ├── Tugla.php
    ├── Kapi.php
    └── Pencere.php

    └── ana.php //index.php
*/
```

- Bir proje yapısı oluşturulmalı
- Proje altında ilgili dosyalar ilgili yerlerde toplanmalıdır.
- Sınıflar için classes adında bir klasör açılmalı
- Üretilen tüm sınıflar bunun içinde toplanabileceği gibi kullanacağınız proje mimarisine göre de klasörleme yapabilirsiniz. (MVC)

Tüm Sınıfları Classes Klasörü Altında Oluşturalım

```
<?php  
class Kapı {  
    public $en=90;  
    public $boy=200;  
    public $renk="kahverengi";  
    public $yon="sağ";  
  
    public function kapıAc() {  
        if($this->yon=="sol")  
            echo "kapı sola açılır";  
        else  
            echo "kapı sağa açılır";  
    }  
?>
```

Tüm Sınıfları Classes Klasörü Altında Oluşturalım

```
<?php  
class Pencere {  
    public $en=200;  
    public $boy=100;  
    public $tur="ahşap";  
    public function saglikliMi() {  
        if($this->tur=="ahşap")  
            echo "sağlıklı";  
        else  
            echo "sağlıksız";  
    }  
?>
```

Tüm Sınıfları Classes Klasörü Altında Oluşturralım

```
<?php
class Tugla {
    //özellikler
    public $en=10;
    public $boy=7;
    public $yukseklik=5;
    // Hacim hesaplayan metot
    public function hacim() {
        return $this->en * $this->boy * $this->yukseklik;
    }
}

?>
```



Class larımız oluşturulduktan sonra bunları istediğimiz yerde kullanalım

```
/*
proje/
└── classes/
    ├── Tugla.php
    ├── Kapi.php
    └── Pencere.php

    └── ana.php //index.php
*/
```

- Bir kez yazıp sürekli kullanalım.
- ana.php sayfasında bu sınıflardan nesne üretelim

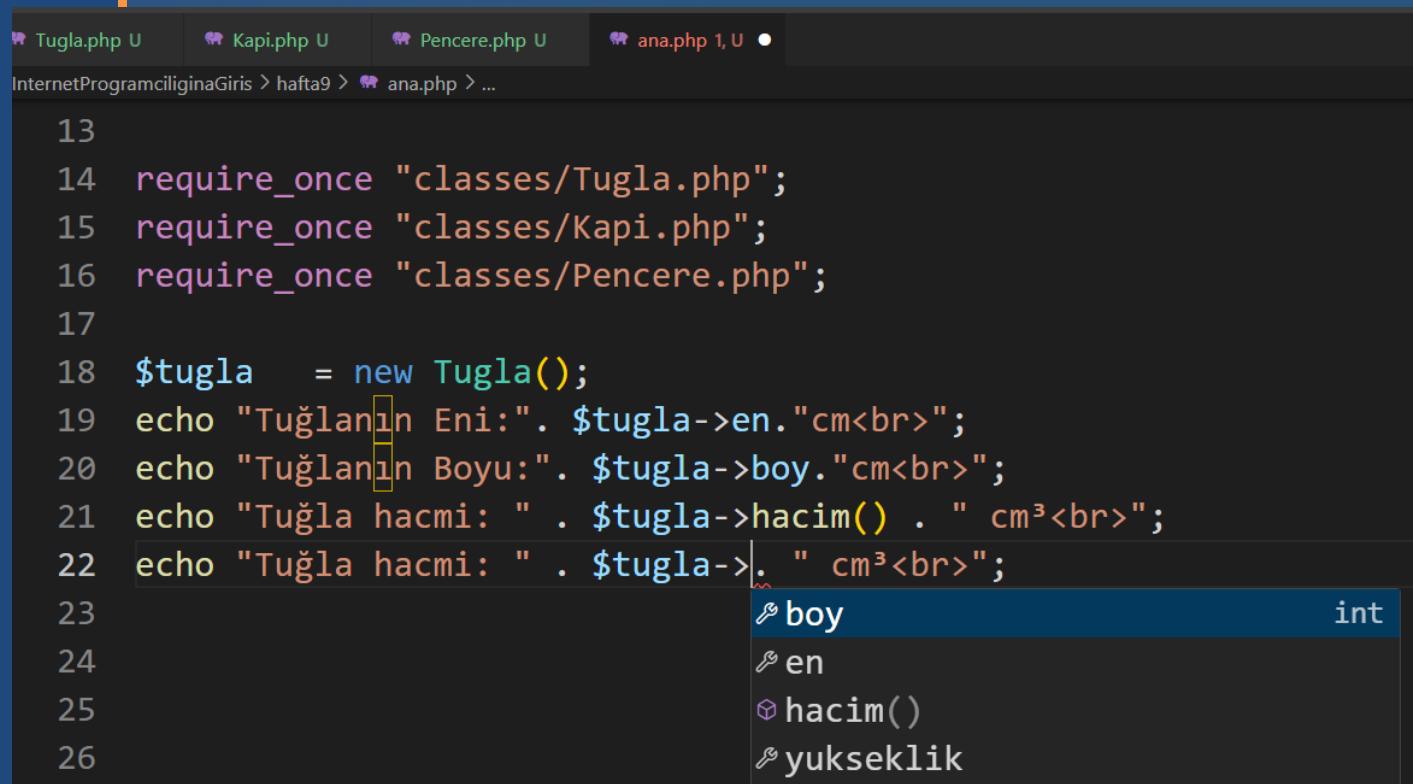


Class larımız oluşturulduktan sonra bunları istediğimiz yerde kullanalım

```
require_once "classes/Tugla.php";  
require_once "classes/Kapi.php";  
require_once "classes/Pencere.php";
```

- Ana.php sayfasına önce bu sınıflara ait php dosyalarını dahil etmeliyiz.

Class larımız oluşturulduktan sonra bunları istediğimiz yerde kullanalım



The screenshot shows a code editor with several tabs at the top: Tugla.php U, Kapi.php U, Pencere.php U, and ana.php 1, U. Below the tabs, the file content is displayed:

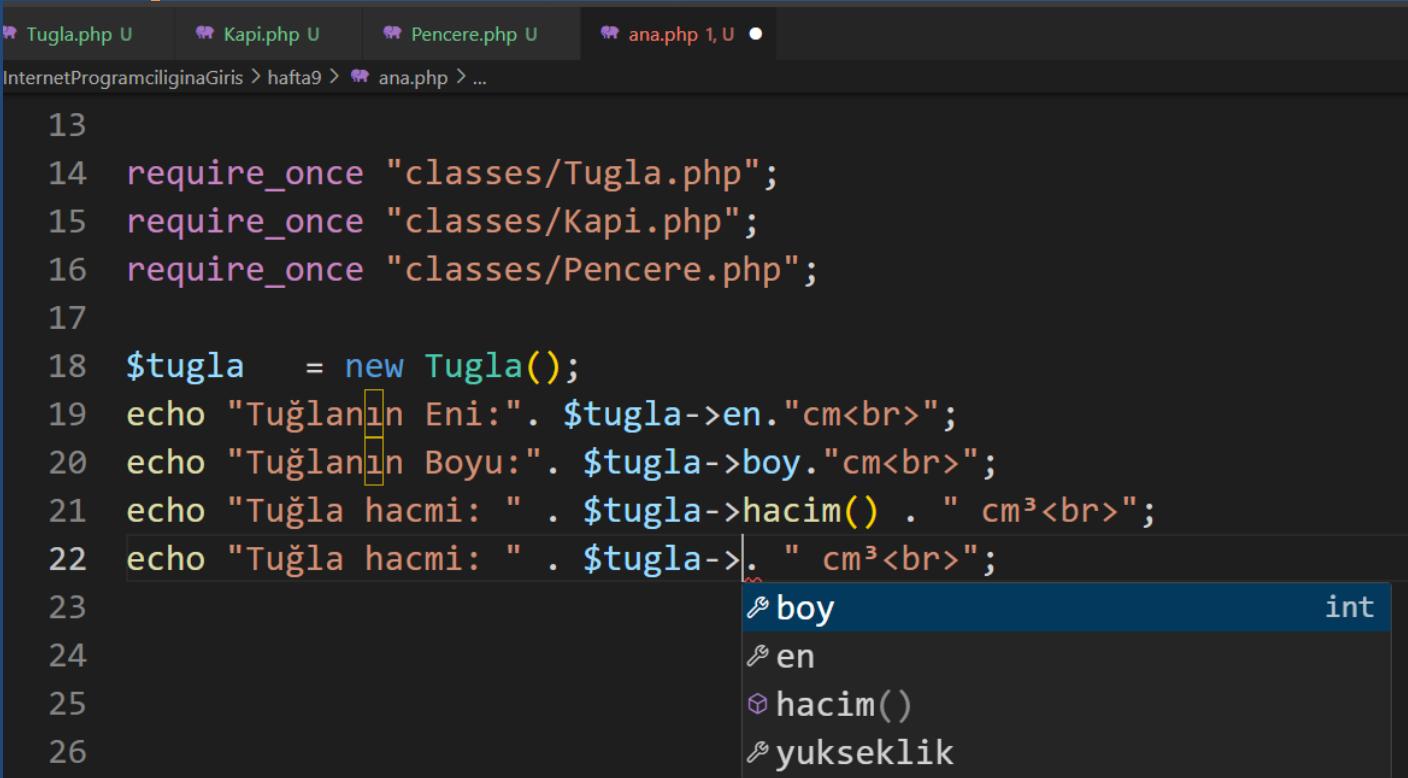
```
13
14 require_once "classes/Tugla.php";
15 require_once "classes/Kapi.php";
16 require_once "classes/Pencere.php";
17
18 $tugla = new Tugla();
19 echo "Tuğlanın Eni:". $tugla->en."cm<br>";
20 echo "Tuğlanın Boyu:". $tugla->boy."cm<br>";
21 echo "Tuğla hacmi: " . $tugla->hacim() . " cm³<br>";
22 echo "Tuğla hacmi: " . $tugla->.yükseklik." cm³<br>";
23
24
25
26
```

A tooltip is visible over the word `yükseklik`, showing suggestions: `boy`, `en`, `hacim()`, and `yükseklik`. The `boy` suggestion is highlighted.

```
$tugla = new Tugla();
```

- Kodu yeni bir tuğla nesnesi oluşturulmasını sağlar.

Class larımız oluşturulduktan sonra bunları istediğimiz yerde kullanalım



```
13
14 require_once "classes/Tugla.php";
15 require_once "classes/Kapi.php";
16 require_once "classes/Pencere.php";
17
18 $tugla = new Tugla();
19 echo "Tuğlanın Eni:". $tugla->en."cm<br>";
20 echo "Tuğlanın Boyu:". $tugla->boy."cm<br>";
21 echo "Tuğla hacmi: " . $tugla->hacim() . " cm³<br>";
22 echo "Tuğla hacmi: " . $tugla->. " cm³<br>";
23
24
25
26
```

int
boy
en
hacim()
yukseklik

```
$tugla = new Tugla();
```

```
$tugla->en
```

- Kodu ise \$tuğla adındaki nesnenin en değişkenine ulaşmanızı sağlar.
- Görüldüğü üzere en boy yükseklik değerlerinin tamamına ulaşabilirsiniz.
- Bununla berber hacim fonksiyonunu da çağırabilirsiniz .

Artık istediğimiz classlar dan istediğimiz kadar nesne üretip kullanabilir.

```
require_once "classes/Tugla.php";
require_once "classes/Kapi.php";
require_once "classes/Pencere.php";

$tugla = new Tugla();
echo "Tuğlanın Eni:". $tugla->en."cm<br>";
echo "Tuğlanın Boyu:". $tugla->boy."cm<br>";
echo "Tuğla hacmi: " . $tugla->hacim() . " cm3<br>";
echo "Tuğla hacmi: " . $tugla->yukseklik. " cm3<br>";
$kapi = new Kapi();
echo "Kapı Ne Yöne Açılıyor: " . $kapi->kapiAc(). " <br>";
echo "Kapı Rengi: " . $kapi->renk. " <br>";

$pencere = new Pencere();
echo "Pencere Malzemesi Sağlıklı Mı?: " . $pencere->saglikliMi() . " <br>";
echo "Pencere Boy: " . $pencere->boy . " <br>";
```

Tuğlanın Eni:10cm
Tuğlanın Boyu:7cm
Tuğla hacmi: 350 cm³
Tuğla hacmi: 5 cm³
kapı sağa açılırKapı Ne Yöne Açılıyor:
Kapı Rengi: kahverengi
sağlıklıPencere Malzemesi Sağlıklı Mı?:
Pencere Boy: 100

CONSTRUCTOR

Kurucu Metot

⚙️ Nesne Oluşturulduğunda Otomatik Çalışır

📝 İlk Değerleri Ayarlar

⚙️ Nesneyi Kuilanıma Hazırlar

```
public function __construct() {  
    // Kurulum başlatılır  
}
```

Nesneyi Başlatan İlk Metot



Constructor Nedir?

- **Constructor**, bir sınıfın nesne oluşturulduğu anda otomatik çalışan özel metottur.
 - PHP'de adı **`__construct()`** olmak zorundadır.

Constructor Ne İşe Yarar?

- Nesne oluşturulurken **ilk değerleri atamak**
- Nesneyi **kullanıma hazır hale getirmek**
- Zorunlu bilgileri baştan almak

Constructor kullanılmazsa:

- Özellikler boş (null) kalabilir
- Hatalı veya eksik nesneler oluşabilir

Parametreli Constructor Tanımlama

```
<?php
class Tugla1 {
    //özellikler
    public $en=10;
    public $boy=7;
    public $yukseklik=5;
    //constructor tanımlama
    public function __construct($en, $boy, $yukseklik) {
        $this->en = $en;
        $this->boy = $boy;
        $this->yukseklik = $yukseklik;
    }
    // Hacim hesaplayan metot
    public function hacim() {
        return $this->en * $this->boy * $this->yukseklik;
    }
}
?>
```

Sınıf İçinde Constructorı ÇAĞIRMA

```
<?php
require_once "Tugla1.php";
$tugla = new Tugla1(20, 30, 10);
/*→□ new anahtar kelimesi kullanılır
→□ __construct() otomatik çağrıılır
→□ Değerler nesneye atanır*/
echo "Tuğlanın Eni:". $tugla->en."cm<br>";
echo "Tuğlanın Boyu:". $tugla->boy."cm<br>";
echo "Tuğla hacmi: " . $tugla->hacim() . " cm3<br>";
echo "Tuğla hacmi: " . $tugla->yukseklik. " cm3<br>";
?>
```

Tuğlanın Eni:20cm
Tuğlanın Boyu:30cm
Tuğla hacmi: 6000 cm³
Tuğla hacmi: 10 cm³

CONSTRUCTOR İLE METOD ARASINDAKİ FARK

- Constructor
- Otomatik çalışır
- Nesne oluşurken
- `__construct`

- Normal Metot
- Manuel çağrıılır
- İstenince
- İstenilen isim

Constructor, bir sınıfın nesne oluşturulduğunda otomatik çalışan ve nesneyi başlatan metottur.

Parametresiz Constructor Tanımlama

```
<?php  
class Tugla2 {  
    //özellikler  
    public $en=10;  
    public $boy=7;  
    public $yukseklik=5;  
    //constructor tanımlama -> Parametresiz Constructor  
    public function __construct() {  
        echo "tugla oluşturuldu";  
    }  
    // Hacim hesaplayan metot  
    public function hacim() {  
        return $this->en * $this->boy * $this->yukseklik;  
    }  
}  
?>
```

Parametresiz Constructor Tanımlama

```
<?php
require_once "Tugla2.php";
$tugla = new Tugla2();
/*→□ new anahtar kelimesi kullanılır
 →□ __construct() otomatik çağrıılır
 →□ Değer olarak başlangıçta sınıfıta belirlenen değerler ne ise onlar
nesneye atanır*/
echo "Tuğlanın Eni:". $tugla->en."cm<br>";
echo "Tuğlanın Boyu:". $tugla->boy."cm<br>";
echo "Tuğla hacmi: " . $tugla->hacim() . " cm3<br>";
echo "Tuğla hacmi: " . $tugla->yukseklik. " cm3<br>";
?>
```

```
tugla oluşturulduTuğlanın Eni:10cm
Tuğlanın Boyu:7cm
Tuğla hacmi: 350 cm3
Tuğla hacmi: 5 cm3
```

‘ Bir Sınıfta Kaç Constructor Olur?

PHP’de:

X Sadece 1 tane

// **X** Bu mümkün değil

```
public function __construct() {}  
public function __construct($a) {}
```

★ Java/C# ile fark burada çıkar.

PHP'de Çoklu Constructor Yerine Ne Yapılır?



Varsayılan parametreli constructor

```
<?php  
class Tugla3 {  
    //özellikler  
    public $en;  
    public $boy;  
    public $yukseklik;  
    //constructor tanımlama ->  
Parametresiz Constructor  
    public function __construct($en,  
$boy=null,$yukseklik=null) {  
        $this->en=$en;  
        $this->boy=$boy;  
        $this->yukseklik=$yukseklik;  
    }  
    // Hacim hesaplayan metod  
    public function hacim() {  
        return $this->en * $this->boy *  
$this->yukseklik;  
    }  
}  
?>
```

```
<?php  
require_once "Tugla3.php";  
$t1 = new Tugla3(10);  
echo "Tuğlanın Eni:". $t1->en."cm<br>";  
echo "Tuğlanın Boyu:". $t1->boy."cm<br>";  
echo "Tuğla yükseklik: " . $t1->yukseklik. " cm3<br>";  
echo "Tuğla hacmi: " . $t1->hacim() . " cm3<br>";  
  
echo "<hr>ikinci Tuğla bilgileri<br>";  
$t2 = new Tugla3(10, 20);  
echo "Tuğlanın Eni:". $t2->en."cm<br>";  
echo "Tuğlanın Boyu:". $t2->boy."cm<br>";  
echo "Tuğla yükseklik: " . $t2->yukseklik. " cm<br>";  
echo "Tuğla hacmi: " . $t2->hacim() . " cm3<br>";  
  
echo "<hr>üçüncü Tuğla bilgileri<br>";  
$t3 = new Tugla3(10, 20,5);  
echo "Tuğlanın Eni:". $t3->en."cm<br>";  
echo "Tuğlanın Boyu:". $t3->boy."cm<br>";  
echo "Tuğla yükseklik: " . $t3->yukseklik. " cm<br>";  
echo "Tuğla hacmi: " . $t3->hacim() . " cm3<br>";
```

Array parametreli constructor

```
<?php
class Tugla4 {
    //özellikler
    public $en;
    public $boy;
    public $yukseklik;
    //constructor tanimlama -> Parametresiz
    Constructor
    public function __construct(array $data=[]) {
        $this->en=$data['en'];
        $this->boy=$data['boy'];
        $this->yukseklik=$data['yukseklik'];
    }
    // Hacim hesaplayan metot
    public function hacim() {
        return $this->en * $this->boy * $this-
>yukseklik;
    }
}
?>
```

```
<?php
require_once "Tugla4.php";
$dizi=['en'=>5,'boy'=>6,'yukseklik'=>11];
$t1 = new Tugla4($dizi);
echo "Tuğlanın Eni:". $t1->en."cm<br>";
echo "Tuğlanın Boyu:". $t1->boy."cm<br>";
echo "Tuğla yükselik: " . $t1->yukseklik. "cm3<br>";
echo "Tuğla hacmi: " . $t1->hacim() . "cm3<br>";
?>
```

PHP 8+ Named Arguments Constructor

```
<?php
class Tugla5 {
    //constructor tanımlama ->PHP 8+
Named Arguments Constructor
    public function __construct(
        public int $en,
        public ?int $boy=null,
        public ?int $yukseklik=null
    ) { }
    // Hacim hesaplayan metot
    public function hacim() {
        return $this->en * $this->boy
* $this->yukseklik;
    }
}
?>
```

```
<?php
require_once "Tugla5.php";
$t1 = new Tugla5(en:56,yukseklik:23);
echo "Tuğlanın Eni:". $t1->en."cm<br>";
echo "Tuğlanın Boyu:". $t1->boy."cm<br>";
echo "Tuğla yükseltik: " . $t1->yukseklik. "
cm3<br>";
echo "Tuğla hacmi: " . $t1->hacim() . " cm3<br>";

?>
```

SORU

/* ? SORU:

Aşağıda özellikleri verilen Ogrenci adlı bir sınıfı PHP nesne yönelimli programlama (OOP) kurallarına uygun olarak oluşturunuz.

◆ Ogrenci Sınıfı

Özellikler (Properties):

adSoyad → string

sinif → int

tcNo → string

ders → string

dersNotu → float

◆ Constructor

PHP'de constructor overload olmadığı için tek bir constructor tanımlanacaktır.

Constructor, tüm özellikleri parametre olarak almalıdır.

ders ve dersNotu parametreleri varsayılan değerlere sahip olmalıdır:

ders → "Henüz atanmadı"

dersNotu → 0.0

◆ Metotlar

notDurumu()

Ders notu 50 ve üzerindeyse "Geçti"

Aksi halde "Kaldı" mesajını döndürmelidir.

harfNotu() Ders notuna göre aşağıdaki harf notunu döndürmelidir:

Not Aralığı Harf Notu

90 - 100 AA

80 - 89 BA

70 - 79 BB

60 - 69 CB

50 - 59 CC

0 - 49 FF

bilgileriGoster() Öğrenciye ait tüm bilgileri ekrana yazdırmalıdır.

Çıktı içinde not durumu ve harf notu da yer almalıdır.

◆ Main (ana.php)

En az iki adet Ogrenci nesnesi oluşturunuz.

Nesnelerden biri için tüm parametreleri,

diğeri için yalnızca zorunlu parametreleri gönderiniz.

Oluşturulan nesneler için:

bilgileriGoster()

notDurumu()

metotlarını çağrıarak sonuçları ekrana yazdırınız. */

PASS BY VALUE

Bir değişkeni **fonksiyona normal şekilde gönderirsek**,
PHP varsayılan olarak pass by value kullanır.

Yani:

Fonksiyon, değişkenin **kopyası** ile çalışır
Orijinal değişken **değişmez**

PASS BY VALUE

```
<?php
function artir($sayi) {
    $sayi++;
}
$x = 5;
artir($x);
/*
➤
$x değişmedi, çünkü:
$sayı, $x'in kopyasıdır
*/
echo $x; // 5
?>
```

PASS BY REFERANCE

```
<?php
function artirRef(&$sayi) {
    $sayi++;
}
$x = 5;
artirRef($x);
/*📌 Bu sefer:
Fonksiyon gerçek değişkeni değiştirdi
çünkü & işaretini adresi=referansı iletti*/
echo $x; // 6
?>
```

NESNELERDE PASS BY REFERANCE

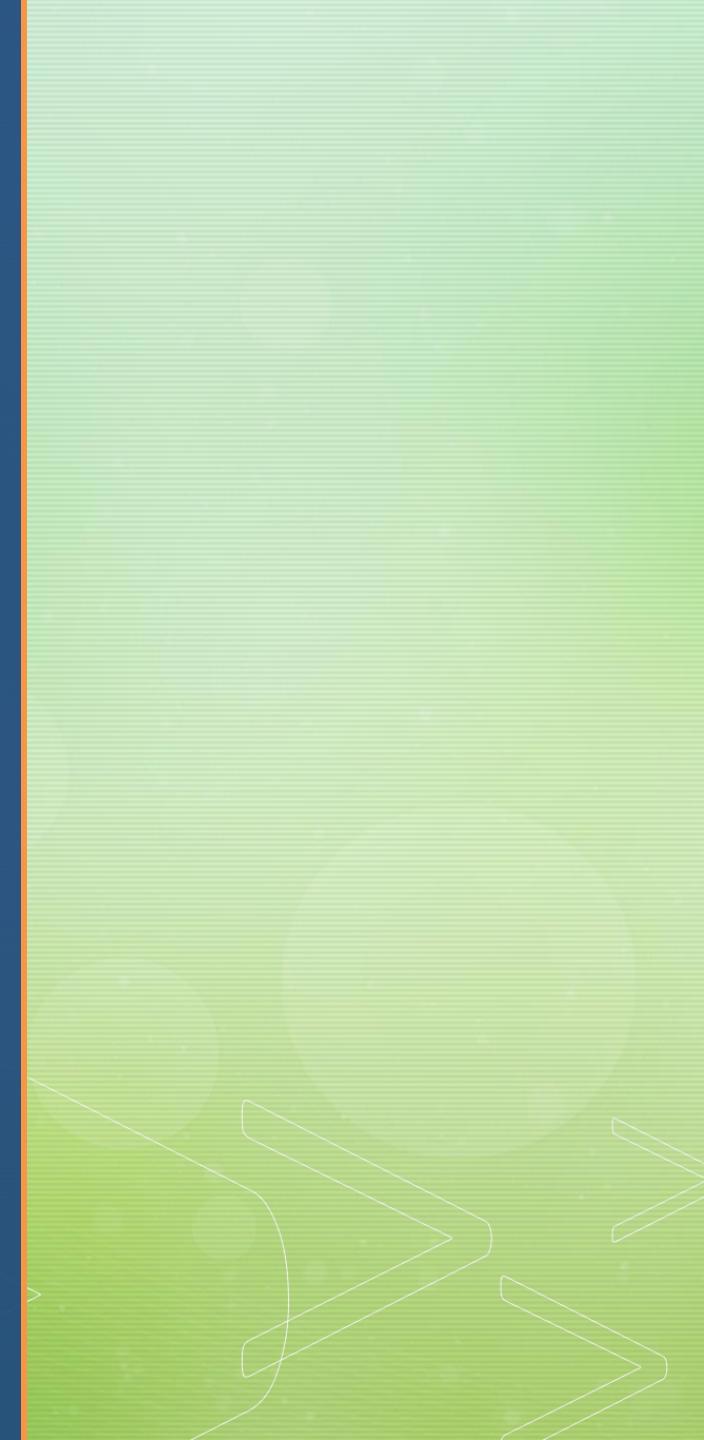
```
<?php
class Test {
    public $deger = 10;
}
/*◆ Nesnelerde Durum Farklıdır ▲ (Önemli)
PHP'de object'ler referans benzeri davranıştır.
➤ Neden?
Nesneler handle (referans) gibi aktarılır
& yazmasan bile değişiklik olur*/
function degistir($nesne) {
    $nesne->deger = 20;
}
$t = new Test();
degistir($t);
echo $t->deger; // 20

?>
```



Erişim Belirleyiciler

ACCESS MODIFIERS



ACCESS MODIFIERS

Access modifier'lar,

**Sınıf içindeki property ve metotlara nereden
erişilebileceğini** belirler.

PHP'de **3 tane** erişim belirleyici vardır:

- 1.public
- 2.protected
- 3.private

PUBLIC

- ✓ Her yerden erişilebilir
- Sınıf içinden
- Sınıf dışından
- Alt sınıflardan

PROTECTED

- ✓ Sadece sınıf içinden ve alt sınıflardan erişilir
- ✗ Sınıf dışından erişilemez

PRIVATE

- ✓ Sadece tanımlandığı sınıf içinden erişilir
- ✗ Alt sınıflardan bile erişilemez
- ✗ Sınıf dışından erişilemez

Modifier	Sınıf İçi	Alt Sınıf	Sınıf Dışı
public	✓	✓	✓
protected	✓	✓	✗
private	✓	✗	✗

```
<?php  
class AnaSinif  
{  
    public string $publicDegisken = "Kütüphane";  
    protected string $protectedDegisken = "Aile Albümü";  
    private string $privateDegisken = "Kasa";  
  
    public function sinifIcindenErisim()  
    {  
        echo $this->publicDegisken . "<br>"; // ✓ ÇALIŞIR  
        echo $this->protectedDegisken . "<br>"; // ✓ ÇALIŞIR  
        echo $this->privateDegisken . "<br>"; // ✓ ÇALIŞIR  
    }  
}  
  
$nesne = new AnaSinif();  
$nesne->sinifIcindenErisim();  
?>
```

```
<?php
class FarkliSinif
{
    public function erisimiDene()
    {
        $nesne = new AnaSinif();

        echo $nesne->publicDegisken . "<br>";           // ✓ ÇALIŞIR

        // echo $nesne->protectedDegisken;                 // ✗ HATA
        // echo $nesne->privateDegisken;                   // ✗ HATA
    }
}

$f = new FarkliSinif();
$f->erisimiDene();
?>
```

```
<?php
class TuretilmisSinif extends AnaSinif
{
    public function erisimiDene()
    {
        echo $this->publicDegisken . "<br>";    // ✓
        echo $this->protectedDegisken . "<br>";   // ✓
        // echo $this->privateDegisken;           // ✗
    }
}

$t = new TuretilmisSinif();
$t->erisimiDene();
?>
```

Durum

Her yerden erişim

Kalıtımda kullanılacak

Gizli veri

Modifier

public

protected

private

STATIC

◆ PHP'de static Nedir?

static, sınıfı ait olan özellik ve metotları ifade eder.

! static üyeleri nesneye değil, sınıfı aittir.

📌 Önemli:

- \$this kullanılamaz
- SınıfAdı::\$degisken şeklinde erişilir

STATIC

“Ev – Buzdolabı – Diş Fırçası”
benzetmesiyle **adım adım** anlatıyorum.

🎯 Amaç:

- **static = ortak (herkese ait)**
- **normal property = bireysel (kişkiye ait)**

STATIC

Bir **ev** düşünelim:

?

Buzdolabı

Evde **tektir**

Herkes aynı **buzdolabını** kullanır

→ **static**

?

Diş fırçası

Her bireyin **kendi diş fırçası** vardır

Kimse paylaşmaz

→ **normal (non-static)**

STATIC

```
<?php
/* ➔ Açıklama:static $buzdolabi → evde 1 tane
$disFirca → her nesneye özel */
class Ev
{
    // Herkes için ORTAK (static)
    public static string $buzdolabi = "Bos";
    // Kişiye ÖZEL (non-static)
    public string $disFirca;
    public static function buzdolabiniGoster()
    {
        echo "Buzdolabında: " . self::$buzdolabi;
    }
}
```

```
//Eve İlk Kişiyi Ekliyoruz
$ali = new Ev();
$ali->disFirca = "Ali'nin dis fircasi";
/*Buzdolabına Bir Şey Koyalım (Static) Önemli:Buzdolabı Ali'ye
değil, eve ait Nesne ile değil, sınıf adıyla erişilir */
Ev::$buzdolabi = "Sut ve Yumurta";
```

```
//Eve İkinci Kişi Giriyor
$ayse = new Ev();
$ayse->disFirca = "Ayse'nin dis fircasi";
//Durumu Kontrol Edelim
echo $ali->disFirca . "<br>";
echo $ayse->disFirca . "<br>";

echo Ev::$buzdolabi;
/*Ali'nin dis fircasi Ayse'nin dis fircasi Sut ve Yumurta
```

Diş fırçaları farklı Buzdolabı aynı */

```
//Biri Buzdolabını Değiştirirse Ne Olur?  
$ayse::$buzdolabi = "Peynir ve Zeytin";  
echo Ev::$buzdolabi; //Peynir ve Zeytin  
/*✖ Çünkü:  
Static = herkes için ortak */
```

```
//static metod kullanımı  
Ev::buzdolabiniGoster();
```

```
//Static Metotta $this Neden Yok?  
/**✖ Çünkü:  
* $this → nesne  
* static → sınıf */  
?>
```