PHP Eğitim Kitabı

Mehmet Ali Uysal

Mehmet Ali Uysal

phpegitimkitabi.com

linkedin.com/in/mehmet-ali-uysal-23774246

instagram.com/mehmetali.eses

mehmetali@gelistir.org

destek@phpegitimkitabi.com

phpegitimkitabi@gmail.com

Kitaba dair örnek uygulama ve kaynak kodları aşağıdaki linkten indirebilirsiniz.

https://github.com/phpegitim/kitap

İçindekiler

Bölüm 1: Giriş 1

PHP Nedir? 2

PHP'nin Tarihi 2

PHP ile Neler Yapılabilir? 3

PHP'nin Kullanım Alanları ve Avantajları Nelerdir? 5

Neler Öğrendik? 5

Bölüm 2: Ayarlamalar ve Kurulum 7

Geliştirme Ortamının Hazırlanması 8

IDE Seçimi ve Kurulumu 8

- Notepad++ 9
- NetBeans IDE 9
- Eclipse 9
- Aptana Studio 9
- Aptana Studio Kurulumu 9

Apache, PHP ve MySQL Kurulumu 18

- Apache HTTP Server 18
- MySQL 18
- PHP 18

Yerel Sunucu Ayarlarının Yapılması 26

- Yerel Saat Tanımlaması 31
- PHP Kısa Açılış Etiketinin Aktif Edilmesi 32
- Hata Gösterimin Açılması ve Hata Raporlama Seviyesinin Tanımlanması 33
- E_WARNING 34
- E_DEPRECATED 35
- E_STRICT 35
- İhtiyaç Duyulan Dinamik Eklentilerin Aktif Edilmesi 37

Neler Öğrendik? 39

Bölüm 3: PHP'nin Temelleri 41

Giriş 42

Syntax 46

PHP Etiketleri 48

PHP Yorum Satıları 50

PHP Büyük Küçük Harf Duyarlılığı 50

Neler Öğrendik? 59

Bölüm 4: PHP'de Değişkenler ve Sabitler 61

Değişkenler 62

Sabitler 65

PHP'de Kaçış Sekansları 69

Neler Öğrendik? 74

Bölüm 5: Veri Tipleri 75

PHP'de Veri Tipleri 76

Primitive Data Types 76

Non-Primitive Data Types 76

Boolean Veri Tipi 77

integer Veri Tipi 78

float Veri Tipi 80

string Veri Tipi 82

null Veri Tipi 85

array Veri Tipi 87

object Veri Tipi 89

PHP'de Tür Dönüşümleri 90

Type Casting (Tür Dönüşümü) 90

Type Juggling (Öz Devinimli Tür Dönüşümü) 102

Neler Öğrendik? 105

Bölüm 6: PHP'de Operatörler 107

Aritmetik Operatörler 108

Toplama Operatörü 108

Çıkarma Operatörü 108

Çarpma Operatörü 109

Bölme Operatörü 109

Mod (Modulo) Operatörü 110

Üs Alma (Exponentiation) Operatörü 111

Atama Operatörleri 113

Artırarak Değer Atama Operatörü 113

Bölerek Değer Atama Operatörü 114

Azaltarak Değer Atama Operatörü 114

Çarparak Değer Atama Operatörü 115

Mod Değer Atama Operatörü 116

Birleştirme Operatörü 116

Artırma ve Azaltma Operatörleri 118

Sonradan Artırma Operatörü 119

Önceden Artırma Operatörü 120

Sonradan Azaltma Operatörü 121

Önceden Azaltma Operatörü 121

Karşılaştırma Operatörleri 122

İki Değerin Denkliği (Eşitliği) Durumu 123

İki Değerin Aynı Olma Durumu 126

İki Değerin Denk Olmama Durumu 128

İki Değerin Aynı Olmama Durumu 129

Küçüktür Operatörü 130

Büyüktür Operatörü 130

Küçük veya Eşittir Operatörü 131

Büyük veya Eşittir Operatörü 131
Spaceship Operatörü 131
Mantıksal Operatörler 133
ve Operatörü (&&, AND) 133
ya da operatörü (|| , OR) 137
Farklılık, Yahut operatörü (xor) 140
Değilse (not) Operatörü (!) 140
Neler Öğrendik? 141

Bölüm 7: PHP'de Koşullu İfadeler 143

if İfadesi 144
else İfadesi 148
elseif İfadesi 153
switch İfadesi 157
Üçlü Koşul Operatörü 165
Neler Öğrendik? 168

Bölüm 8: PHP'de Diziler 171

PHP'de Dizilerin Tanımlanması 172
PHP'de Dizi Elemanlarına Erişim 179
PHP'de Diziye Eleman Ekleme 180
PHP'de Diziden Eleman(ları) Silme 184
unset Fonksiyonu 184
array_splice Fonksiyonu 186
array_shift Fonksiyonu 188
array_pop Fonksiyonu 189
array_diff Fonksiyonu 190
array_unique Fonksiyonu 191
Neler öğrendik? 191

Bölüm 9: PHP'de Dizilerde Sıralama 193

sort Fonksiyonu 194

rsort Fonksiyonu 195

asort Fonksiyonu 195

arsort Fonksiyonu 196

ksort Fonksiyonu 196

krsort Fonksiyonu 197

natsort Fonksiyonu 197

array_reverse Fonksiyonu 200

shuffle Fonksiyonu 200

Neler Öğrendik? 201

Bölüm 10: PHP'de Dizilerde Dahili İşaretçiler 203

key ve current Fonksiyonları 204

next Fonksiyonu 205

prev Fonksiyonu 206

end Fonksiyonu 206

reset Fonksiyonu 207

Neler Öğrendik? 208

Bölüm 11: PHP'de Çok Boyutlu Diziler 209

Çok Boyutlu Dizilerin Tanımlanması 210

Çok Boyutlu Dizi Elemanlarına Erişim 219

Neler Öğrendik? 223

Bölüm 12: PHP'de Dizi Fonksiyonları 225

is_array Fonksiyonu 226

in_array Fonksiyonu 227

array_key_exist ve isset Fonksiyonları 229

array_unique Fonksiyonu 231
array_search Fonksiyonu 232
count Fonksiyonu 233
max ve min Fonksiyonları 235
array_rand Fonksiyonu 236
array_keys Fonksiyonu 238
array_count_values Fonksiyonu 238
array_combine Fonksiyonu 240
explode ve implode Fonksiyonları 242
Neler Öğrendik? 247

Bölüm 13: PHP'de Döngüler 249

Giriş 250
PHP'de for Döngüsü 250
PHP'de foreach Döngüsü 262
PHP'de while Döngüsü 278
PHP'de do…while Döngüsü 281
Neler Öğrendik? 285

Bölüm 14: PHP'de Ön Tanımlı Değişkenler 287

$GLOBALS 288
$_SERVER 290
HTTP_HOST 291
HTTP_CONNECTION 292
HTTP_UPGRADE_INSECURE_REQUESTS 292
HTTP_USER_AGENT 293
HTTP_ACCEPT 293
HTTP_REFERER 294
HTTP_ACCEPT_ENCODING 294

HTTP_ACCEPT_LANGUAGE 294
PATH 295
SERVER_SOFTWARE 295
SERVER_ADDR 296
SERVER_PORT 296
REMOTE_ADDR 296
DOCUMENT_ROOT 296
SERVER_ADMIN 297
REMOTE_PORT 297
SERVER_PROTOCOL 297
REQUEST_METHOD 297
QUERY_STRING 298
REQUEST_URI 298
SCRIPT_NAME, PHP_SELF 298
REQUEST_TIME 298
Neler Öğrendik? 298

Bölüm 15: PHP'de Fonksiyonlar 301

PHP'de Fonksiyon Tanımlanması 303
PHP'de Fonksiyon Argümanları 305
Fonksiyon Argümanlarının Referans ile Kullanılması 306
İstenilen Uzunlukta Argüman ile Fonksiyon Çağırma 307
PHP'de Değişkenlerde Etki Alanı (Variable Scopes) 311
Yerel Etki Alanı 311
Küresel Etki Alanı 312
PHP'de Fonksiyonların Varlığının Kontrol Edilmesi 314
PHP'de Tanımlı Tüm Fonksiyonların Görüntülenmesi 316
PHP'de Bir Değişkenin Fonksiyon Gibi Çağırılabilmesi 318

PHP'de Anonim Fonksiyonlar 319
PHP'de İç İçe Fonksiyonlar 326
PHP'de "Static" Anahtar Sözcüğü ve Fonksiyonlarda Kullanımı 328
PHP'de Özyineli Fonksiyonlar 331
PHP'de Üreteç Yapısı ve Üreteç Fonksiyonlar 337
PHP 7'De Tip Beyanı ve Dayatması 342
Neler Öğrendik? 345

Bölüm 16: Nesne Yönelimli Programlama 347

Nesne Nedir? 348
Nesne Yönelimi Programlama Nedir? 350
Sınıf Nedir? Nesne ile Farkı Nedir? 351
Nesnelerde Görünürlük 356
Statik Kavramı ve Sınıf Sabitleri 357
Sihirli Metotlar 361
Kurucu ve Yıkıcı Metotlar 361
Get, Set, Isset, Unset, Call, CallStatic Metotları ve Aşırı Yükleme 363
Sleep, Wakeup Metotları ve Serialization 367
Nesne Kopyalama ve Clone Sihirli Metodu 375
Try, Catch, Finally Yapıları ve İstisnalar ile Hata Yönetimi 380
Neler Öğrendik? 383

Bölüm 17: Nesnelerde Kapsülleme ve Bilgi Gizleme 385

Koruma Maddeleri (Guard Clauses) 386
Nesnelerde Kapsülleme ve Bilgi Gizleme 389
PHP'de Dosya Dâhil Etme 398
Neler Öğrendik? 399

Bölüm 18: Nesnelerde Kalıtım ve Çok biçimlilik 401

Kalıtım (Inheritance) 402

Çok biçimlilik (Polymorphism) 403

Kalıtım, Çok Biçimlilik ve Kapsülleme Örneği 404

Neler Öğrendik? 412

Bölüm 19: Nesnelerde Soyutlama 415

Nesnelerde Soyutlama 416

Nesnelerde Singleton Tasarım Deseni 432

Neler Öğrendik? 437

Bölüm 20: Nesne Arayüzleri ve Ortak Nitelik Sınıfları 439

Arayüzler 440

Ortak Kullanılabilen Nitelikler 446

DRY, KISS ve YAGNI ile Prensip Sahibi Yazılımlar 447

Metot Zincirleme 458

Neler Öğrendik? 461

Bölüm 21: PHP'de İsim Alanı Kullanımı ve Otomatik Yükleme 463

PHP'de İsim Alanı (Namespace) Kullanımı 464

Otomatik Yükleme 468

Neler Öğrendik? 477

Bölüm 22: Nesne Yönelimli Programlamada SOLID Tasarım Prensipleri479

Tek Sorumluluk Prensibi 480

Açık/Kapalı Prensibi 487

Yerine Geçme Prensibi 491

Arayüz Ayrıştırma Prensibi 493

Bağımlılığın Ters Çevrilmesi Prensibi 497

Neler Öğrendik? 499

Bölüm 23: PHP'de Tarih ve Zaman İşlemleri 501

date() Fonksiyonu 502

unix Zaman Damgası ve time() Fonksiyonu 504

getDate() Fonksiyonu 506

mktime() Fonksiyonu 506

localTime() Fonksiyonu 507

strftime() Fonksiyonu 509

Karşılaştırma 513

Mevcut Zamanı Manipüle Etme 515

Neler Öğrendik? 516

Bölüm 24: Çerez ve Oturum İşlemleri 519

Çerezler 520

setcookie Fonksiyonu 520

$_COOKIE Değişkeni 524

PHP'de Oturum İşlemleri 526

PHP'de Oturum Güvenliği 529

Neler Öğrendik? 533

Bölüm 25: PHP'de Form İşlemleri 535

HTML Formlar 536

$_GET ve $_POST Dizi Değişkenleri ile Form İşleme 538

Neler Öğrendik? 549

Bölüm 26: Veritabanı 551

Veritabanı İşlemleri 552

LAMP 552

MySQL 553

MySQL'in Temel Özellikleri 554

MySQL Yönetim Araçlar (GUI) 555

Veritabanı tasarımı 555

PHP ile Veritabanı Bağlantısı 564
PDO ile CRUD işlemleri (CREATE/READ/UPDATE/DELETE) 566
CREATE 566
READ 569
UPDATE 571
DELETE 574
Neler Öğrendik? 575

Bölüm 27: MVC Yazılım Mimari Deseni 577

Model Katmanı 579
Controller Katmanı 579
View Katmanı 580
Neler Öğrendik? 582

Bölüm 28: to-do List Projesi 585

İhtiyaç ve Proje Analizi 586
Tasarım 587
Programlama 589
Uygulama Çatısı 590
TO-DO List Uygulaması 613
Neler öğrendik? 665

Bölüm 29: PHP'de Web Servis ve REST, SOAP, CURL Yapıları 667

REST API 668
CURL 669
SOAP 671
Neler Öğrendik? 673
Son Söz 674
Dizin 680

1

Giriş

Bu Bölümde

PHP Nedir? 2
PHP'nin Tarihi 2
PHP ile Neler Yapılabilir? 3
PHP'nin Kullanım Alanları ve Avantajları Nelerdir? 5
Neler Öğrendik? 5

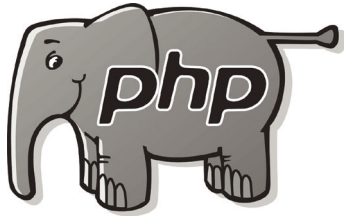
Bu bölümde, PHP programlama dilinin yapısını, tarihini ve yetkinliğini detaylı olarak öğrenecek, PHP'nin, neden en çok tercih edilen sunucu taraflı programlama dilleri arasında olduğunu irdeleyeceğiz.

Ayrıca PHP Web Programlamada doğru bir tercih mi? sorusunun cevabını arayacak, betik programlama dili nedir? bu konu hakkında bilgi sahibi olacağız. Bunlarla birlikte PHP ile neler yapılabilir, PHP geliştiricisi olmanın avantajları nelerdir? açıklamaya çalışacağız.

PHP NEDİR?

PHP, sunucu tarafında çalışan açık kaynaklı betik bir programlama dilidir. PHP kısaltmasının açılımı, **Hypertext Preprocessor** olarak ifade edilmektedir. PHP, yazım standardı açısından esnek bir yapıya sahip olduğundan öğrenim süreci oldukça hızlı ve basit bir dildir. Sunucu taraflı programlama dilleri arasında en çok tercih edilen dilin PHP olduğunu rahatlıkla söyleyebiliriz.

NOT	Bir betik dili genel anlamda, diğer programlama dilleriyle bütünleşmek ve iletişim kurmak için tasarlanmış bir programlama dilidir. Betikler doğrudan kaynak kodundan çalıştırılır. Diğer programlama dilleri ile yazılan kodlar hazır hale getirilmeden önce, makine koduna çevrilip, sonuç olarak üretilen program çalıştırılırken, betik kodları yorumlayıcı (interpreter) tarafından doğrudan okunur ve yorumlanır. Örneğin; JavaScript, PHP, Python, Ruby, VBScript



PHP'NİN TARİHİ

Rasmus Lerdorf tarafından 1994 yılında, C dili ile CGI (Common Gateway Interface) dizisi olarak PHP'nin ilk görünümü oluşturuldu. Lerdorf'un amacı web sayfasının çevrimiçi trafiğini ölçmekti. Bu amaçla yola çıkan Lerdorf zamanla dile veritabanı yönetimi ve basit web uygulamaları yapılabilecek özellikleri eklemeye başladı. Dile ilk ismini veren tanım bu aşamada ortaya çıktı.

Lerdorf, Andi Gutmans ve Zeev Suraski isimli geliştiriciler ile 1997 yılında işbirliği yaparak, PHP'yi tamamen bağımsız ve kararlı bir dil olarak tekrar yazmak için çalışmaya başladı ve dilin bugünkü yapısına benzer ilk versiyonu ortaya çıktı. Bu gelişimi dilin adında da değişiklik yaparak duyurdular. Artık PHP (Hypertext Preprocessor) olarak bilinecekti. Bu aşamada dilin versiyonu **PHP 3** olarak adlandırıldı. Bu sürecte sunucu tarafında işlem yapan ve HTML içine yerleştirilip çalıştırılabilen betik dili mantığı geliştiricilerinin büyük ilgisini çekti.

3. Sürümün yayınlamasınından kısa bir süre sonra, 1998 yılı kış aylarında **Andi Gutmans** ve **Zeev Suraski** dili geliştirmek için dilin çekirdeğini tekrar yazmaya başladılar. Amaçları komplex işlemlerde dilin performansını artırmak ve dilin kod tabanının (code base) modulerliğini geliştirmekti. Bu sayede üçüncü taraf bir çok veritabanı ve API desteklenmiş olacaktı ancak dilin hali hazırdaki yapısı (PHP 3) bu yapılara tam olarak uyumlu değildi.

1999 yılında PHP'nin yeni yorumlayıcı motoru olan **Zend Engine** tanıtıldı ve **PHP4** ortaya çıkmış oldu. **Zend Engine** ismi geliştiricilerin ilk isimlerinden oluşacak şekilde seçildi. (Zeev ve Andi). Bu sürümün oldukça gelişmiş performansına ek olarak, PHP 4, birçok web sunucusu desteği, HTTP oturumları, çıktı arabellekleme, kullanıcı girişini ele almada daha güvenli yollar ve birkaç yeni dil yapısı gibi diğer önemli özellikleri içeriyordu.

Günümüzde PHP versiyonları arasında %83.9 oranda en çok kullanan sürüm olan PHP 5, (2018 Mayıs) uzun geliştirme sürecinin sonunda Zend Engine 2.0 ile birlikte temmuz 2004'te piyasaya sürüldü. Çekirdeği, yeni nesne modeli ve yeni bir çok özellikle birlikte geliştirildi.

2015 yılında ise **PHP 7** ya da farklı bir isimle PHPNG (PHP Next Generation) yeni özellikler karşımıza çıktı. Bu sürümün en önemli özelliği ise daha az kaynak ile 2 kata kadar daha yüksek performans. Buna ek olarak PHP 5'in günümüz de en stabil ve en çok kullanılan alt versiyonu olan PHP 5.6 ile yazılan tüm kodların PHP 7'de desteklenmesi, geçiş ve öğrenme sürecinin en büyük artısı diyebiliriz.

PHP ile Neler Yapılabilir?

PHP ile her şey yapılabilir!

PHP esas olarak **sunucu taraflı betik programlamaya odaklanmış bir dildir**. Bu sayede diğer CGI programlarının, form verilerini toplaması, dinamik sayfa içeriği oluşturması veya çerezleri gönderip alma gibi yapabilecekleri her şeyi yapabilirsiniz. Ama PHP daha fazlasını yapabilir.

Sunucu Taraflı Programlama; Sunucu taraflı betik kodlama yapmak için ihtiyacınız olan şeyler, bir PHP yorumlayıcısı, bir web sunucusu ve bir web browser hepsi bu kadar.

Komut Satırı Programlama; Herhangi bir sunucu veya tarayıcı olmadan çalıştırmak için bir PHP betiği oluşturabilirsiniz. Bu şekilde kullanmak için sadece PHP ayrıştırıcısına ihtiyacınız var. Bu tarz kullanım genelde zamanlanmış görevlerde tercih edilir.

Masaüstü Programlama; PHP GTK grafik kullanıcı arabirimi ile masaüstü programlar yazabilirsiniz. Ancak sunucu taraflı programalaya odaklanmış bir dil olduğundan, bu alanda PHP'in iddialı bir dil olduğunu söylemeyiz.

PHP, Linux, bir çok Unix varyantı (HP-UX, Solaris ve OpenBSD dahil), Microsoft Windows, Mac OS X, RISC İşletim Sistemi ve muhtemelen diğerleri gibi tüm ana işletim sistemlerinde kullanılabilir.

PHP ile, işletim sistemi ve web sunucusu seçme özgürlüğüne sahibiz. Ayrıca, prosedürel programlama veya Nesne Yönelimli Programlama - Object Oriented Programming (OOP) veya bunların her ikisinin bir karışımını kullanma seçeneğine de sahibiz.

PHP ile sadece HTML çıktısı ile sınırlı değiliz. PHP'nin yetenekleri arasında, PDF dosyalarının ve hatta videoların anında üretilmesi gibi şeylerde var. Ayrıca XHTML ve diğer XML dosyaları gibi herhangi bir metni kolayca çıktı alabiliriz. PHP, bu dosyaları otomatik olarak düzenleyebilir ve bunları yazdırmak yerine, sunucu tarafındaki bir önbellek oluşturarak dosya sistemine kaydedebilir.

PHP'nin en güçlü ve en önemli özelliklerinden biri de çok çeşitli veritabanlarına destek vermesidir. Veritabanı özellikli bir web sayfası yazarken, veritabanına özgü uzantılar (Örneğin; MySQL için MySQLi) kullanılarak veya PDO gibi bir soyutlama katmanı kullanılarak veya ODBC uzantısı aracılığıyla bir veritabanına bağlanmak inanılmaz derecede basittir.

PHP, farklı hizmetlerle LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (Windows için) ve daha sayısız protokol aracılığıyla iletişim kurabilecek bir altyapıya da sahiptir. Hazır modüllerin haricinde ham ağ soketleri açıp bu soketler üzerinden istediğiniz bütün protokollerle çalışabilirsiniz. PHP, WDDX üzerinden sanal olarak sanal doku üzerinde, hangi dilde yazılmış olursa olsun tüm uygulamalarla haberleşebilir.

Ayrıca Java nesnelerinin oluşturulabilmesi ve şeffaf biçimde PHP nesneleri olarak kullanılabilmeleri önemli bir diğer özelliktir.

PHP'nin akılda kalan bazı özellikleri ve faydalarını sıraladık. Hepsini tek tek açıklamak yerine ***php.net*** sayfasında **Yapılandırma ve Kurulum** bölümünde ve **İşlev başvuru kılavuzunda** listelenen eklentilerin başlangıç bölümlerindeki açıklamalardan her bir eklenti için daha fazla bilgiye ulaşabilirsiniz.

3

PHP'nin Temelleri

Bu Bölümde

Giriş 42
Syntax 46
Neler Öğrendik? 59

PHP öğrenim sürecimizin yol haritasını belirleyecek temel yaklaşımları, bağlı kalacağımız küresel standartları ve neden gerekli olduklarını anlayacağız.

PHP'de, Uygulama çatılarından (framework), Nesne yönelimli (OOP), Fonksiyonel ya da yordamsal programlama kavramlarından bahsedeceğiz.

PHP'yi öğrenirken benimsenmesi gereken, ihtiyaç analizi ve uygulama geliştirme disiplinin neden önemli olduğunu anlayacak, bu bağlamda sadece PHP dilini kullanmayı öğrenmek yerine, doğru kararlar veren, temiz kod yazan ve etkili uygulamalar üretebilen başarılı bir programcı olmayı hedeflemiş olacağız. :)

Bunların yanı sıra PHP dilinin temel söz dizimi (syntax) kurallarını öğrenecek ve ilk örnek uygulamamızı söz dizimi kuralları ile yapacağız. Söz dizimi hatalarını örneklendirerek, hata okumanın nasıl yapılması gerektiğini öğreneceğiz.

Ayrıca seçilen IDE'ye kodlama sürecini hızlandırmak adına yardımcı eklentilerin (code assist) nasıl kurulduğunu öğreneceğiz.

Giriş

Bölüm başlığında da ifade edildiği üzere. PHP'nin öğrenim sürecinin temellerini atacağız. Bu anlamda sağlam adımlarla ilerlemek için bazı küresel standartlara bağlı kalacağız. PHP öğrenim sürecimizde bağlı kalacağımız PSR (PHP Standards Recommendations) standartlarının neden gerekli olduğundan kısaca bahsedelim.

PHP yayınlandığı günden bugüne tam anlamıyla merkezileşmiş bir standarta maalesef ki sahip olmadı. PHP'nin yazım standartları konusunda ilk günden beri esnek bir yaklaşım sergilemesi **Nasıl olsa çalışıyor...** mantığı ile çok farklı ve anamoli içeren yazım şekillerine sahip geliştiricilerin ortaya çıkmasına yol açtı. İlk başlarda bazı büyük yazılım geliştirme ekipleri barındıran kurumlar ve birden fazla kod tabanı (code base) ile geliştirme yapmak zorunda kalan geliştiriciler kendi projeleri için çeşitli isimlendirme ve kodlama tasarımları hazırladılar. Bazıları; **PEAR** veya **Zend Framework** gibi iyi dökümante edilmiş standartları benimser iken, diğerleri daha farklı yapılar üretmeye çalıştılar.

Günümüzde PHP için hazırlanmış birçok başarılı uygulama çatısı (framework) mevcut. Bunlara birkaç örnek vermek gerekir ise; Laravel, Phalcon, Symfony2, CodeIgniter ve Slim çatılarını örnek gösterebiliriz.

PHP ile framework kullanımı konusunda uzun zamandır birçok farklı bakış açısı var. Bu bakış açılarını biraz irdelemeden önce üst seviye programlama dillerinde uygulama geliştirme yetkinliği başlığı altında 1 ile 10 arasında bir seviye cetveli belirleyelim ve gruplandıralım.

Bu cetveli PHP geliştiricilerinin bakış açılarını, daha iyi anlamak için kullanacağız.

a. İhtiyacım olan her türlü geliştirmeyi istediğim gibi framework olmadan yapıyorum.

Bu grubu cetvelimizde ilk 3 seviye içinde değerlendirebiliriz.

Yukarıda bahsettiğimiz **Nasıl olsa çalışıyor...** standardına sahip geliştiricilerin bu grubun içinde olduğunu rahatlıkla söyleyebiliriz.

PHP daha önce de bahsettiğimiz gibi çok esnek bir dil. Bir betiği birçok şekilde yazabiliriz. Bu bağlamda yordamsal, fonksiyonel, nesne yönelimli ya da tip güvenli (type safety) olmadan kodlayarak aynı sonucu elde edebiliriz.

PHP için bu cetvelin ilk 3 aşamasındaki geliştiricilerin özelliklerini genel olarak ifade etmemiz gerekirse;

Kendi standartları vardır.

» Yordamsal ya da fonsiyonel kodlama yaparlar.

» Test ve hata ayıklama yapılarını kullanmazlar.

» Girintileme ve anlaşılırlık (Indentation and Comprehensibility) stillerine bağlı kalmazlar.

» Bilinen bir framework kullanmazlar.

Yüzdelik dilimde PHP geliştiricilerinin maalesef büyük parçası bu ilk üç aşamada yer almaktadır. Sektörel tecrübelere dayanarak bunu rahatlıkla söyleyebiliriz. PHP dilinin yeterliliği konusunda yapılan tartışmaların ana sebeplerinden birisi, bu yaklaşım ile geliştirilen uygulamalardır.

b. Tüm uygulamalarımda **X** framework kullanıyorum mükemmel içinde bütün araçlar var. Her şey çok kolay...

Puan cetvelimizde bu grubu konumlandırmak gerekirse 4. seviye ve sonrası diyebiliriz. Yüzdelik dilimde PHP geliştiricilerin ikinci büyük kısmı bu seviyelerde yer almaktadır.

Bilindik PHP framework'lerin ortaya çıkış amacı ilk 3 aşamada bulunan geliştiricileri kolay yönetilebilir, kodlama standartlarına sadık, gelişmiş ve kolay bir çatı altında toplayarak bir standart oluşturmaktır.

» Uygulama çatısının küresel standartlara sadık kuralları vardır.

» Büyük oranda Nesne Yönelimli programlama yapılması zorunludur.

» Test ve hata ayıklama yapılarını çatı içerisinde mevcuttur.

PHP dilini diğer web programlama dilleri ile kıyaslamak istenirse 4. seviye ve sonrası için kıyaslamak daha doğru olur.

Bu bağlamda web programlamada JAVA ve ASP.Net ile kıyas yapılacak ise bu yapılarda kesinlikle ilk 3 aşamanın olmadığını söyleyebiliriz. Web programlama özelinde PHP'yi JAVA ve ASP.Net ile kıyaslayacak isek 4. seviye ve sonrası prensipler ile geliştirilen uygulamalar temel alınarak kıyaslanmalıdır.

Java, ASP.Net gibi dillerde ilk 3 aşama olarak belirttiğimiz serbestliğin kesinlikle olmadığını söyleyebiliriz.

» Framework kullanımı neredeyse olmazsa olmazdır.

» Nesne Yönelimli Programlama yapılması zorunludur.

» Uygulama geliştirme ortamlarının güçlü olması ve framework mimarisi sebebiyle küresel standartlara bağlıdırlar.

Bu bilgilerden sonra PHP için bu yaklaşım tarzının tam olması gerektiği gibi bir geliştirme ortamı olduğunu düşünebiliriz. Ancak PHP'de farklı bir durum söz konusu. PHP Framework'lerinin sayısının çok fazla olması ile birlikte **Zend Framework** harici diğerlerinin gönüllü geliştirici grupları ya da kurumları tarafından geliştirildiğini görüyoruz. Bu sebeple muhtemelen karşılaşabileceğimiz ilk sorun platforma bağımlılık olacaktır. Framework geliştiricileri konuyu bir proje olarak ele aldıkları için projeye desteklerini herhangi bir sebeple çekmeleri durumunda zaman içerisinde yeni teknolojilere uyum sağlayamayacağınız bir yapı içinde sıkışıp kalmamız olasıdır.

Örneğin; CodeIgniter'ın çok popüler bir framework olmasına rağmen geliştirme sürecinin sonlandırıldığı duyurulmuştu. Bir süre sonra bir üniversitenin bu framework'u tekrar ele alarak geliştirme sürecini devralacağını duyurduğunu biliyoruz.

Bunların yanında, tek bir framework ile uygulama geliştiriliyor ise çok basit bir uygulama için bile çatıya bağlı kalmamız gerekebiliyor. Frameworklerde de performans sorunları bug'lar olasıdır. Bunun giderilmesi için yeni sürümün çıkması beklemek zorunda kalınması yine bir sorun olarak nitelendirilebilir.

c. Projenin gereksinimlerine göre bir framework seçiyorum ya da küresel standartlara bağlı kalarak bir çatı oluşturuyorum...

Olması gereken bakış açısıdır. Yüzdelik dilimde en az bu durumda geliştirici vardır diyebiliriz. Bu modeli benimseyen PHP geliştiricisi puan cetvelinde 1. seviyeden 10. seviyeye kadar tüm durumlarda optimum çözümü üretebilir.

» Dilin tüm yeteneklerine teknik olarak hakimdir.

» Global standartlara sıkı bir şekilde bağlıdır.

» Nesne Yönelimli Programlamaya hakimdir ve tercih eder. İhtiyaca yönelik kolaylıkla yordamsal ya da fonksiyonel programlama yapabilir.

» Dile hakım olmanın sonucu olarak ihtiyaca yönelik bilindik bir framework seçebilir. İlgili framework'u kısa sürede kolayca öğrenebilir.

» MVC, MVP, MVVM gibi yazılım mimarilerini bilir. Bu mimarilerde kendi uygulama çatısını oluşturabilir.

Bu bağlamda kitabımızda da PSR standartlarına bağlı kalarak bu model ile geliştirme yapmaya yönelik çalışacağız.

6

PHP'de Operatörler

Bu Bölümde

Aritmetik Operatörler 108
Atama Operatörleri 113
Artırma ve Azaltma Operatörleri 118
Karşılaştırma Operatörleri 122
Mantıksal Operatörler 133
Neler Öğrendik? 141

Operatörler aldıkları değerlerin sayısına göre gruplandırılabilirler. Bu bölümde öğrenme önceliğimize göre gruplayarak sırayla inceleyeceğiz.

ARITMETIK OPERATÖRLER

Okulda öğrendiğimiz temel aritmetik kavramlarını PHP sözdizimi ile öğreneceğiz. Önce matematikte 4 işlem olan Toplama, Çıkarma, Çarpma, Bölme işlemlerini öğrenecek. Sonrasında ise mod alma ve üslü sayı işlemlerini yapacağız. Hadi başlayalım.

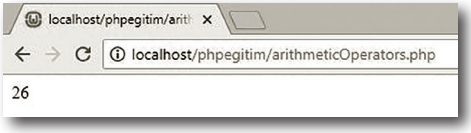
Projemiz içerisinde **arithmeticOperators.php** isimli bir dosya oluşturalım.

TOPLAMA OPERATÖRÜ

PHP'de toplama operatörü matematikte de olduğu gibi **artı** + işaretidir

```
<?php

$a = 20;
$b = 6;

echo $a + $b;
```

Üstteki örnekte a değişkenine integer 20 değerini, b değişkenine integer 6 değerini atadık. Son olarak bu iki ifadeyi toplama operatörü olan **artı** + ile toplayarak echo ile sayfaya yazdırmasını istedik. Sayfamızı çalıştıralım.



Sonuç tahmin ettiğimiz gibi **26** tam sayı (integer) değeri oldu.

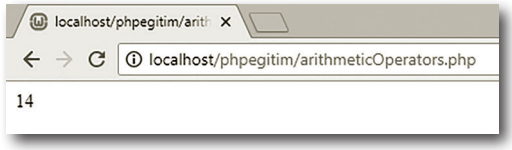
ÇIKARMA OPERATÖRÜ

PHP'de çıkarma operatörü matematikte de olduğu gibi **eksi** - işaretidir. Örneğimizi yazalım ve çalıştıralım.

```
<?php

$a = 20;
$b = 6;

echo $a - $b;
```

Üstteki örnekte a değişkenine 20 tamsayı değerini, b değişkenine ise 6 tamsayı değerini atadık. Sonraki satırda a değişkeni değerinden, b değişkeni değerini

çıkarmak için **eksi - çıkarma** operatörünü kullandık. Son olarak `echo` ile yazdırma komutunu verdik.



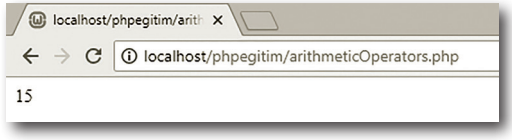
Sonuç olarak a değişkeni değeri 20 sayısından, b değişkeni değeri 6 çıkarıldı ve 14 sayısı elde edildi.

ÇARPMA OPERATÖRÜ

PHP'de çarpma operatörü *** asterisk** (yıldız) işaretidir. Örneğimizi yazalım ve çalıştıralım.

```
<?php

$a = 3;
$b = 5;

echo $a * $b;
```

Üstteki örnekte a değişkenine 3 tamsayı değerini, b değişkenine 5 tam sayı değerini verdik. Sonraki satırda a değişkeni değeri ile b değişkeni değerini çarpmak için **yıldız * çarpma** operatörünü kullandık. Son olarak `echo` ile yazdırma komutunu verdik.



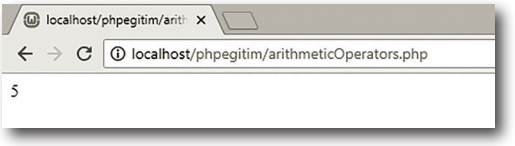
Beklenildiği gibi çıkan sonuç 3 ve 5 tam sayı değerlerinin çarpımının sonucu 15 oldu.

BÖLME OPERATÖRÜ

PHP'de bölme operatörü sağa **eğik çizgi /** işaretidir.

```
<?php

$a = 20;
$b = 4;

echo $a / $b;
```

Üstteki örnekte a değişkenine 20 tamsayı değerini, b değişkenine 4 tam sayı değerini verdik. Sonraki satırda a değişkeni değerini, b değişkeni değerine bölmek için / **eğik çizgi** bölme operatörünü kullandık. Son olarak echo ile yazdırma komutunu verdik.



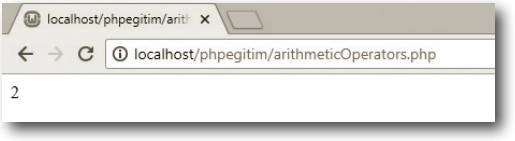
Sonuç olarak a değişkeninin değeri olan 20 tam sayısının 4 sayısına bölümünden elde edilen 5 sonucunu sayfaya yazdırmış olduk.

MOD (MODULO) OPERATÖRÜ

Modüler aritmetikten hatırladığımız modulo işlemidir. Kısaca **mod** olarak ifade edilmektedir. Yani bir sayının bir sayıya bölümünden arta kalan değerdir. PHP'de modulo operatörü karakteri ise % **yüzde** işaretidir.

```
<?php

$a = 20;
$b = 6;

echo $a % $b;
```

Üstteki örnekte a değişkenine 20 tamsayı değerini, b değişkenine 6 tam sayı değerini verdik. Sonraki satırda a değişkeni değerinin, b değişkeni değerine bölümünden kalan değeri alabilmek için % **yüzde mod** operatörünü kullandık. Son olarak echo ile yazdırma komutunu verdik.



Üstteki örnekte 20 sayısı 6 sayısına bölündü ve kalan tam sayı değeri sonuç olarak ekrana yazdırıldı. Modulo işlemini tam olarak anlayamadı isek, kısa bir Google aratması yaparak **Matematikte Mod İşlemi Alma** konusunu pekiştirebilirsiniz. Bu aşamada bir konunun üstünden geçebiliriz. Değişkenlerimi oluştururken tam sayı değerleri seçtiğimiz için veri tipleri integer olarak tanımlandı. Örneğin bir bölme işleminde sonuç ondalıklı bir sayı çıkttığında, çıkan sonucun veri tipinin otomatik olarak float (double) olarak tanımlandığını bilmeliyiz.

Üs Alma (Exponentiation) Operatörü

PHP'de iki sayı taban ve üs olacak şekilde hesaplanarak sonuç alınmak istendiğinde bu işlem kısa yoldan sadece `pow` fonksiyonu ile yapılmakta idi. PHP 5.6 sürümü ve sonrasında bu işlem aritmetik operatörlere eklendi. Bu operatörün söz dizimi iki tane **yan yana asterisk (yıldız)** işareti `**` olarak belirlenmiştir.

```
<?php

$a = 5;
$b = 3;

echo $a ** $b;
```

Üstteki örnekte a değişkenine 5 tam sayı değeri ve b değişkenine 3 tamsayı değeri verildi. Üs alma operatörü ile yapılan işlemde a değeri taban ve b değeri üs olarak belirlendi. Örneğimizi çalıştıralım.

NOT Aptana sözdizimi desteği şuan için PHP versiyonu 5.4'e kadar olduğundan, üs alma söz dizimini tanımayarak hata gösterebilir. Bu hatayı görmezden geliniz.

Yapılan işlemde 5 taban, 3 ise üs olarak belirlenerek 5^3 üslü ifadesi oluşturuldu. Sonuç olarak 125 değeri elde edildi.

Operatör Önceliği (Operator Precedence)

Aritmetik operatörleri genel olarak gördükten sonra bu konu içerisinde hayatı önemi olan Operatör Önceliği (Operator Precedence) konusunu irdeleyelim.

Basit bir örnek ile;

1 + 5 * 3 ifadesinin sonucu işlem önceliği olmadığını varsayar isek 18 olacaktır. Ancak çarpma işleminin toplama işlemine göre önceliği olduğunu biliyoruz. Bu sebeple 18 sonucu yanlıştır.

```
<?php

$a = 1;
$b = 5;
$c = 3;

echo $a+$b*$c;
```

13

PHP'de Döngüler

Bu Bölümde

Giriş	250
PHP'de for Döngüsü	250
PHP'de foreach Döngüsü	262
PHP'de while Döngüsü	278
PHP'de do...while Döngüsü	281
Neler Öğrendik?	285

Bu bölümde programlama dillerinde en önemli yapılardan biri olan döngüleri öğreneceğiz.

PHP'de hangi döngü metodu nerelerde kullanılmalı? sorusuna cevaplar arayacağız.

Giriş

Bir döngü, belirli bir duruma ulaşılana kadar sürekli olarak tekrarlanan bir komut dizisidir. Döngülerle tipik olarak, bir veriyi almak ve değiştirmek gibi belirli bir işlem yapılır ve daha sonra bir sayacın belirlenmiş bir sayıya ulaşıp ulaşmadığı gibi bir durum kontrol edilir. Uygulama içinde belirli satırların birden fazla tekrar edilmesi istenebilir. Böyle durumlarda döngü yapıları kullanılır. Döngü yapılarında döngünün kaç kere tekrar edeceği dinamik olarak belirlenebilir. Hatta döngünün tekrarlaması bir koşula bağlanabilir.

Döngüde belirli sınır durum belirtilmedi ise ya da döngüden bir çıkış kuralı tanımlanmadı ise bu döngü sonsuz bir döngü olacaktır. Bu durumda işletim sistemi bunu algılayana ya da sürekli olarak bir hata ile veya başka bir olay gerçekleşene kadar (belirli bir süre sonra programın otomatik olarak sonlandırılması gibi) döngü sürekli olarak tekrarlanır.

Döngünün her bir tekrarı **iteration** (iterasyon-tekrarlama, yineleme ya da tur) gibi kelimeler ile ifade edilebilmektedir. PHP'de döngülerin en çok ilişkili olduğu kavram dizilerdir. Bu sebeple dizileri bu konudan önce öğrendik ve döngüler ile pekiştireceğiz. Şimdi PHP'de ki döngüleri görelim.

PHP'DE FOR DÖNGÜSÜ

for, kodun tekrar tekrar uygulanmasına izin veren yinelemeyi belirtmek için bir kontrol akışı ifadesidir. for döngüsü hemen hemen tüm programlama dillerinde vardır. C dili ile aynıdır.

`for` döngüsü syntax olarak en karmaşık gözüken döngü türüdür.

```
for (ifade1; ifade2; ifade3)
    komut
```

Üstte `for` döngüsünü anlamak için 4 farklı parçaya ayırdık. Bunları `ifade1`, `ifade2`, `ifade3` ve `komut` olarak belirledik.

`ifade1` döngünün en başında bir kereliğine koşulsuz olarak çalıştırılır.

`ifade2` her yinelemede yeniden değerlendirilir. Bu ifadeden **true** sonucu elde edilirse döngüye devam edilir ve döngünün etki alanındaki komutlar çalıştırılır. Bu ifadeden **false** sonucu elde edildiğinde etki alanındaki komutlar çalıştırılmaksızın döngü sonlandırılır.

İfade3 ise her yinelemenin sonunda çalıştırılır.

Her ifade bir birinden **noktalı virgül ;** ile ayrılır.

Çalışacak komut bloğu şayet bir cümlelik bir komuttan oluşuyor ise kod bloğunu belirtir süslü parantezlerin kullanılmasına gerek yoktur. Bu ifadelerin her birini boş bırakmak da mümkündür. ifade2'nin boş olması döngünün sonsuza kadar çalıştırılacağı anlamına gelir. Bu tip kullanımları çok tercih etmemenizi öneriyoruz. Bir for döngüsünden çıkmak için break komutu kullanılır. break komutu iç içe döngülerde bir üstteki döngüden çıkmak için seviye parametresi alabilir. Örneğin; break 2 komutu verildiğinde bir üstteki döngüyle beraber döngüden çıkılır. Bir for döngüsünün bir sonraki tura atlamasını sağlamak için continue komutu kullanılır. continue komutu iç içe döngülerde bir üstteki döngüyü atlatmak için seviye parametresi alabilir. Örneğin; continue 2 komutu verildiğinde bir üstteki döngü sonraki tura atlatılır. Proje dosyamızda **loopFor.php** isimli bir dosya oluşturalım ve aşağıdaki örneği yazalım.

```
<?php

for ($i = 1; $i < 10; $i++) {
    echo $i;
}
```

Örnekte for ifadesini yazdıktan sonra for döngüsünün çalışma kriterlerini belirlemek için parantezlerimizi açıyoruz.

3 adet ifade yazacağız. Sırayla ilk ifadeden başlayalım.

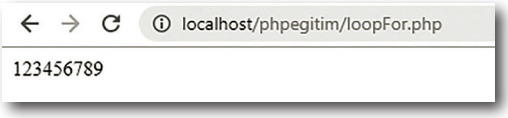
İlk ifademiz $i = 1 ifadesi. Bu ifade ile sadece döngü içerisinde geçerli olacak $i değişkeninin döngü başlarken bir sefere mahsus 1 değerine eşitlenmesini istiyoruz. **Noktalı virgül ;** koyarak ilk ifademizi bitiriyoruz.

İkinci ifadede, $i < 10 tanımını görüyoruz. Büyüktür küçüktür operatörlerini öğrenmiştik. İkinci ifade aslında bir koşullu ifadedir. Yani i değişkeni, 10 tamsayı değerinden küçük olduğu sürece döngünün çalışmaya devam etmesi gerektiğini söylüyoruz. **Noktalı virgül ;** koyarak ilk ifademizi bitiriyoruz.

Son ve üçüncü ifade de ise $i++ tanımını gördük. Artırma azaltma operatörlerinden ++ sözdizimine aşınayız. Bu tanımlama ile döngünün her turunun sonunda, i değişkeni değerinin bir artırılması gerektiğini söylüyoruz.

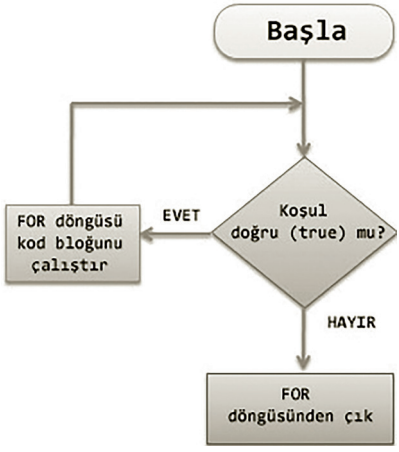
Son olarak döngünün her turunda çalışacak olan kodları yazmak için süslü parantezleri açıyoruz. `echo` komutu ile `i` değişkenini yazdırıyoruz.

İlk döngü örneğimizi çalıştıralım.



Çıktıyı incelediğimizde döngünün 9 kez çalıştığını gördük. İlk ifadede `i` değişkenine 1 değerini atamıştık. Bu sebepten dolayı döngümüz `i` değişkeni değeri 1 ile başladı ve ekrana 1 yazdırıldı ve `i` değeri 1 artırıldı. Döngünün her seferinde ikinci ifade yani `i` değerinin 10 değerinden küçük olup olmadığı kontrol edildi.

Bu durum `i` değerinin her seferinde 1 artırılarak 9 değerine gelmesine kadar devam edildi. Son olarak `i` değeri 10 olduğunda ikinci ifadedeki koşul sağlanmadı ve döngü sonlandırıldı. Sonuç olarak sayfaya, 1'den 9'a kadar sayılar yazdırılmış oldu.



Şimdi aynı örneği **süslü parantez** olmadan ve her seferinde yeni bir satıra geçecek şekilde değiştirelim.

```
<?php

for ($i = 1; $i < 10; $i++)
      echo $i . '<br>';
```

`for` ifadesinde bir önceki örnekten farklı olarak sadece kendinden sonraki ilk komutu çalıştıracağımız için **süslü parantezleri** kullanmadık.

18

Nesnelerde Kalıtım ve Çok biçimlilik

Bu Bölümde

Kalıtım (Inheritance) 402
Çok biçimlilik (Polymorphism) 403
Kalıtım, Çok Biçimlilik ve Kapsülleme Örneği 404
Neler Öğrendik? 412

Bu bölümde; Kalıtım, miras alma, çok biçimlilik ve aidiyet (IS-A) kavramları hakkında bilgi sahibi olacağız.

KALITIM (INHERITANCE)

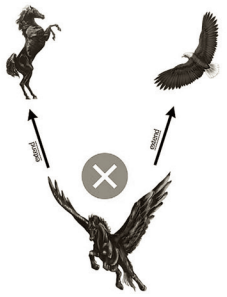
Kalıtım kelimesi ilk olarak, gerçek dünyadaki genetik kalıtımı aklımıza getirmektedir. Bir çocuğun babasının bazı özelliklerini genetik olarak miras almasını buna örnek gösterebiliriz. OOP'de bilgisayar programları her şeyin birbiriyle etkileşime girdiği bir nesne olacak şekilde tasarlanmaktadır. OOP'de kalıtım ise bir sınıfın özelliklerinin diğeri tarafından miras alınabileceği kavramdır. Kodun yeniden kullanılmasına ve farklı sınıflar arasında bir ilişki kurulmasına yardımcı olur.

Üstteki görsel de, yavru aslan babadan bazı özelliklerini miras alıyor. Ancak gerçek dünyadan bir farklılık var diye düşünebilirsiniz. Anne'den hiç miras almıyor mu? Gerçek dünyada tabi ki alıyor pek bilgi sahibi değilim ama belki anneden daha fazla alıyor. Çocuğun iki ebeveynden miras alması durumu OOP'de **Çoklu Kalıtım** (Multi Inheritance) olarak ifade edilir. Modern dillerin bazılarında desteklenmekte bazılarında ise karışıklığa sebep vermemek için çoklu kalıtım desteklenmemektedir. (Bu işlevsel olarak eksiklik değildir.) Çoklu kalıtım desteklendiği dillerden bazıları C++, C#, Python, Lisb'dır.

Çoklu kalıtım PHP'de desteklenmez. Çoklu kalıtım nedeniyle oluşan karmaşıklığı kontrol etmek çok zordur. Tip dönüşümü, zincirleme metotları (öğreneceğiz) vb. gibi çeşitli işlemler sırasında sorun yaratır ve her şeyden önce birden fazla mirasa ihtiyaç duyduğumuz durum yok denecek kadar azdır. Bu yüzden işleri basit ve açık tutmak en iyisidir. Çoklu kalıtımla elde edilmek istenen düzen PHP, JAVA vb. dillerde arayüzler (interface) vasıtasıyla yapılır. Ara yüzleri sonraki başlıklarda öğreneceğiz.

PHP'nin çoklu kalıtımı desteklememesi bir eksiklik değil, tercihtir.

Örneğin Yunan MIT'lerinden Pegasus'u düşünün. Birden fazla kalıtımla Pegasus'un at ve kuştan miras aldığını söyleyebilirsiniz. Çünkü kanatlı bir hayvanı kuş olarak sınıflandırıyoruz.



Kuşların yapabildiği ancak Pegasus'un yapmadığı başka özellikler vardır. Örneğin, kuşlar yumurta bırakırlar, Pegasus canlı doğum yapar. Kalıtım, tüm özellikleri aktardığından bu durum kuş türünden miras alan pegasusu çoğalma yöntemi konusunda ikileme sokacak. Sonuç olarak yumurtlama ya da doğum yapma seçenekleri için karar verilmesi gerekir.

Bahsettiğimiz bazı diller çoklu kalıtımı açık halde bırakmayı tercih etmişlerdir. Ama bu diller yine de çoklu kalıtıma alternatif çözümleri interface, trait gibi yapıları barındırmaktadırlar. Kişisel tecrübelerime dayanarak web uygulamalarında **bunu doğru şekilde yapmak için çoklu kalıtıma gerçekten ihtiyacım var** diye düşündüğüm tek bir durum yok diyebilirim. Çok spesifik bir örnekte ihtiyaç duyulsa bile alternatif yöntemler ile çözmenin daha akıllıca olacağı (örn; trait) kanaatindeyim.

Şimdi ise çok biçimliliği (polymorphism) açıklamaya çalışalım.

Çok biçimlilik (Polymorphism)

Şayet kalıtım zincirleme olarak devam ediyorsa yani en alt sınıf birden fazla üst sınıftan miras alıyorsa **çok biçimli** durumdadır diyebiliriz. Polimorfizm sergileyen bir programlama dilinde aynı hiyerarşik ağacın (ortak bir temel sınıftan miras alınan) sınıflarına ait nesneler aynı adı taşıyan özellik ve metotlara sahip olabilir ancak temelde her biri farklı özellik ve metotlara sahiptir.

Bunu canlılar üzerinden örneklendirebiliriz. Örneğin; bir köpek ilk olarak bir memelidir. Bir memeli de, bir hayvandır. Bu bağlamda düşünecek olursak, **hayvan**>**memeli**>**köpek** gibi hiyerarşik durum ortaya çıkar. Son aşamada bulunan köpek çok biçimlidir. Biraz daha açık ifade ile hem memeli hem de hayvandır. Bunun sonucu olarak köpek kendi özelliklerinin haricinde memelilerin özelliklerini miras alarak oluşmuştur. (canlı doğum vb.) Aynı zamanda da hayvanların genel özelliklerini miras almıştır.

KALITIM, ÇOK BİÇİMLİLİK VE KAPSÜLLEME ÖRNEĞİ

Şimdi, hem Inheritance (kalıtım) hem Polymorphism (Çok biçimlilik) hem de Encapsulation (kapsülleme) içeren geniş kapsamlı bir örnek yapalım. Örnekte hayvanların kalıtımını anlatmaya çalışacağız. Proje dosyamızdaki OOP klasorunun içine **inheritance** isimli bir klasör oluşturalım.

İlk olarak en üst sınıfımız olan `Animal` sınıfını oluşturacağız. Inheritance klasörü içinde **AnimalsClass.php** isimli bir dosya oluşturalım. Aynı klasör içinde bir de **init.php** isimli bir dosya daha oluşturalım.

AnimalClass.php dosyasını açarak sınıfı yazmaya başlayalım.

```
<?php

class AnimalsClass {

	private $family;

	const familyOptions = ['memeli','memeli olmayan'];

	function __construct(string $family) {
		$this->setFamily($family);
	}

	public function getFamily() {
		return ucwords($this -> family);
	}

	private function setFamily(string $family) {

		if(!in_array($family, self::familyOptions))
			throw new InvalidArgumentException("Familya geçersiz");

		$this -> family = $family;
	}

	public function makeNoise(){
		echo 'Ses: ....<br>';
	}

	public function sleep(){
		echo 'Uyku: ZzZzZ...<br>';
	}

}
```

21

PHP'de İsim Alanı Kullanımı ve Otomatik Yükleme

İlk olarak namespace kavramının ne olduğunu sonrasında sınıf öğelerinin otomatik yükleme işleminin nasıl yapılabileceğini öğreneceğiz.

Bu Bölümde

PHP'de İsim Alanı (Namespace) Kullanımı 464
Otomatik Yükleme 468
Neler Öğrendik? 477

PHP'de İsim Alanı (Namespace) Kullanımı

Namespace'i Türkçe ifade etmek gerekirsek **Ad alanı** olarak çevirebiliriz ama bu şekilde pek bir şey ifade etmiyor gibi ilk bakışta.

Namespace, uygulamanın öğelerini gruplamamanın en geniş şeklidir. Sadece nesneler için düşünmeyin. Değişkenler sabitler fonksiyonlarda namespace ile gruplanabilir. Ama biz genelde OOP'de tercih edeceğiz. Namespace Soyut bir kavramdır. **information hiding** (Bilgi gizleme) yapmak için güzel bir araçtır.

Namespace, sanal bir hiyerarşide dosyalarınızın organizasyonunu sağlayabildiğiniz düzendir. Genellikle verilen en klişe örnek ise namespace kullanmadan aynı isme sahip iki farklı class oluşturup kullanmayacağımız yönündedir. Namespace ile bu kolayca yapılabilir. Bu kavramı örnek ile uygularsak daha iyi anlayabiliriz. OOP klasöründe namespace isimli klasör oluşturalım ve **library1.php** isimli bir dosya oluşturalım.

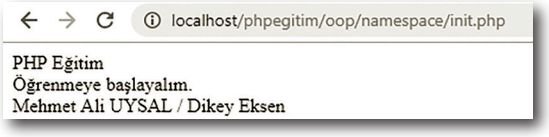
```
<?php

namespace App;

const NAME = 'PHP Eğitim';

function learning() {
	return 'Öğrenmeye başlayalım.';
}

class Organization {
	static function getAuthor() {
		return 'Mehmet Ali UYSAL';
	}

	static function getBookStore() {
		return 'Dikey Eksen';
	}

}
```

Namespace söz dizimini ilk kez görüyoruz. Bu tanımlama betik içinde bir kez daha tanımlanıncaya kadar kendinden sonra tanımlanan değişken, sabit, fonksiyon ve nesne yapıları **App namespace** grubu içinde kümelecek.

Library1.php dosyasında `NAME` adında bir sabit `learning` adında bir fonksiyon Organizaton adında bir sınıf oluşturuyoruz. Şimdi aynı dizinde **init.php** isimli bir dosya oluşturalım.

```
<?php

require 'library1.php';

echo App\NAME,'<br>';
echo App\learning(),'<br>';
echo App\Organization::getAuthor(),' / ', App\Organization::getBookStore();
```

require ile **library1.php** dosyasını betiğe dahil ediyoruz. Sonrasın **library1.php** dosyasında tanımladığımız öğeleri NameSpaceAdı\Oge olacak şekilde çalıştırıyoruz.



Aranılan öğeler belirlediğimiz namespace altından getirildi. Namespace ile grupladıktan sonra eskisi gibi direk öğe ye erişmek istersek artık ulaşamayız.

Namespace'ler, istediğimiz dizin uzunluğunda olabilir.

Uygulamanın bilgilerini direk App namespace ile tutmak çok doğru olmayabilir.

Örneğin şöyle bir namespace olabilir. **App\Data\Info**

Library1.php dosyasında namespace'i **App\Data\Info** olarak değiştirelim.

Bu durumda her öğeye erişmek için başına **App\Data\Info** namespace tanımını yapmamız gerekecek. Bunun için as sözdizimi alias özelliğini görelim.

```
<?php

require 'library1.php';

use App\Data\Info as AppInfo;

echo AppInfo\NAME, '<br>';
echo AppInfo\learning(), '<br>';
echo AppInfo\Organization::getAuthor(), ' / ', AppInfo\Organization::getBookStore();
```

Örnekte, namespace'in genel olarak nasıl çalıştığını gördük. Şimdi ise OOP'de nasıl kullanacağız bir örnek ile uygulayalım.

Namespace klasörü altında sırayla **App**>**Tools**>**DataTypes** klasörlerini oluşturalım. (Tools App klasörü altında olacak şekilde, DataTypes da Tools klasörü

altında olacak şekilde). **DataTypes** klasörü içine `str` isimli bir Sınıf oluşturalım. Bu sınıf string türünde yapılan işlemlerde bize kolaylık sağlayabilecek araçları (metot) barındıracak.

```
<?php

namespace App\Tools\DataTypes;

Class Str {

       public static function rand($length = 8) {

              if ($length === null)

                     throw new BadMethodCallException(__METHOD__.':uzunluk
değeri gönderilmelidir');

              if (!is_numeric($length))

                    throw new InvalidArgumentException(__METHOD__.':uzunluk
değeri numerik olmalıdır');

              // maximum ve minimum uzunluk kontrol cümlesi yazılmalıdır.

              $length = intval($length);

static $characters = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';

              $charactersLength = strlen($characters);
              $randomString = '';
              for ($i = 0; $i < $length; $i++)
                 $randomString .= $characters[rand(0, $charactersLength - 1)];

              return $randomString;
       }

}
```

Bu sınıf sonunda dikkat edersiniz Class kullanmadık. Sınıfın görevini ifade eder ön ekleri namespace tanımlamasında yaptık. `NameSpace` kullanmasa idik bu sınıf ismi şu şekilde olabilirdi. `AppToolsDataTypesStrClass` bu şekilde bir hiyerarşik isimlendirmek yapmak istediğimizde standart dışı çok uzun bir sınıf adı olacaktı. Bunun yerine `namespace` ile `App\Tools\DataTypes\Str` şeklinde daha anlaşılır şekilde grupluyoruz. Bu araç içine örnek bir metot tanımladık. Bu metot verilen uzunlukta rasgele string değerler üretiyor. 2 adet koruma cümlesi var maximum ve minimum uzulunluk kontrollerini de yapılırsa daha iyi olur.

```
$randomString .= $characters[rand(0, $charactersLength - 1)];
```

Cümlesi kafanızı karıştırmasın işlevi aslında çok basit. Bir string değerin aslında bir karakter dizisi olduğunu biliyoruz. Bir dizi gibi yorumlanabilir. Bu bağlamda characters değeri bir dizi olduğundan dizinin içinde rand ile rasgele anahtarlar seçiyoruz. rand fonksiyonu bir başlangıç ve bitiş değeri istediğinden characters dizisinin 0. elemanından sonuncu elemanına (charactersLength-1) (0 dan başladığı için 1 eksilttik) kadar rasgele seç diyoruz. Her seferinde rasgele getirilen eleman değeri randomString değişkenine eklenerek biriktiriliyor. Döngüde istediğimiz uzunluğa kadar döneceği için her turda rasgele bir karakter randomString değişkenine ekleniyor.

Metot static türde değişmesine gerek yok. Bir ustanın sürekli kullandığı cebinde taşıdığı bir el aleti olarak düşünebilirsiniz. Bizimde örneğin rasgele bir şifre bir token vb. bir string üretmek istediğimizde kullanacağımız araç olabilir.

Şimdi **init.php** dosyasına gelelim.

```
<?php

require_once 'App/Tools/DataTypes/Str.php';

use App\Tools\DataTypes\Str;

echo Str::rand(10),'<br>';

echo App\Tools\DataTypes\Str::rand(10);
```

Üstteki kullanımda önce gerekli olan dosyayı yüklüyoruz. Sonrasında kullanacak olduğumuz aracı use sözdizimi ile belirtip aracı kullanabiliyoruz. use kullanmadan da en son satırdaki gibi metoda erişebiliriz. PHP 7 ile birlikte aynı grupta olan yapıları bir use cümlesinde belirtebiliyoruz. Bu alternatif söz dizimi çok kullanışlıdır.

```
use App\Tools\DataTypes\{Str,Number};
```

ya da takma isim kullanacak isek,

```
use App\Tools\DataTypes\{Str as StrTool,Number as NumTool};
```

Başka bir namespace içinde yine Str isimli bir class olsa idi namespace ile artık grupladığımız için aynı isimli iki sınıfı sorunsuz şekilde kullanabilecektik.

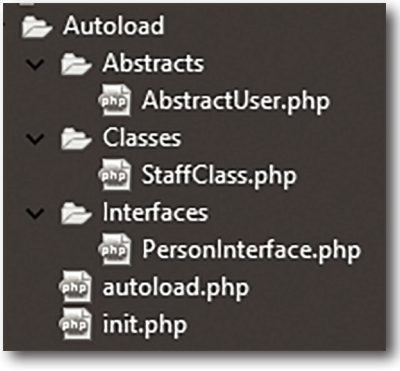
Namespace hiyerarşisi uygulama dizini ile aynı olmalıdır. Örneğin; `App/Tools/DataTypes` klasörlerimiz mutlaka olmalıdır. Bu gerekli bir PSR standardıdır.

OOP'de namespace'leri sürekli kullanacağız. OOP konularının sonlarına doğru gelirken, gerekli sınıfları artık manuel olarak betiğe eklemeyelim. Hadi PHP'de **AutoLoading** (Otomatik Yükleme) özelliğini öğrenelim.

OTOMATİK YÜKLEME

Otomatik yükleme yapısı adından da tahmin edebileceğimiz gibi sınıflarla ilgili işlem yapmadan önce o sınıfın kaynak dosyasını sayfamıza otomatik dâhil eder. Sınıf diyoruz ama aslında tüm OOP öğelerini yükleyebiliriz. Şimdiye kadar bir kaç yazılım geliştirme prensibinden bahsettik. **SRP**, **DRY** vb. bu prensiplerde yazılım geliştirmek için farklı sorumluluğu olan sınıfların farklı dizinlerde oluşturulması gerektiğini anlamıştık. Örneğin; bir önceki konuda olduğu gibi araçların bulunduğu sınıfları farklı bir dizinde veritabanı işlemlerini yapacak olan sınıfları farklı bir dizinde gibi.

Ya da interface, abstract veya class'ların ayrı dizinlerde olması gibi.



Öğelerin otomatik yüklenmesini örneklendirmek için **OOP** klasörü altında görseldeki gibi bir dizin oluşturacağız.

Şimdi klasör ve dosyaları görseldeki gibi oluşturalım.

PHP'de otomatik yükleme iki şekilde yapılabilir.

İlki artık güncel olmayan ve önerilmeyen `__autoload` fonksiyonudur.

İkinci ve benimsememiz gereken yöntem ise `spl_autoload_register` fonksiyonudur. İlk olarak **PersonInterface.php** isimli dosyayı açalım.

```
<?php

interface PersonInterface {
       public function getName();
       public function setName($name);
}
```

PersonInterface kendini uygulayacak sınıflara name özelliğinin getter ve setter metotlarının oluşturulması gerektiğini söylesin. Her kişinin bir adı vardır mantığıyla kişi arayüzüne sadece isim ile ilgili metotların tanımlamasını yaptık.

Şimdi AbstactUser sınıfını açalım. (Abstract sınıfların sonuna Class değil, başına Abstract ön eki konulması standartlar gereğidir.)

```
<?php

abstract class AbstractUser implements PersonInterface {

	private $name;
	private $userName;
	private $pass;

	public function getName() {
		return $this -> name;
	}

	public function setName($name) {
		$this -> name = $name;
	}

	public function getUserName() {
		return $this -> userName;
	}

	public function setUserName($userName) {
		$this -> userName = $userName;
	}

	public function getPass() {
		return $this -> pass;
	}
	public function setPass($pass) {
		$this -> pass = $pass;
	}

}
```

Soyut kullanıcı sınıfı PersonInterface arayüzünü uyguluyor. Bir kullanıcıda mutlaka olması gereken 3 özellik taşıyor. Bunlar isim kullanıcı adı ve parola. Bu özelliklerin getter ve setter metotlarını yazıyoruz.

Şimdi StaffClass sınıfını yazalım.

```
<?php

class StaffClass extends AbstractUser {

	private $position;

	public function getPosition() {
		return $this -> position;
	}

	public function setPosition($position) {
		$this -> position = $position;
	}

}
```

Bu sınıf ise soyut soyut kullanıcı sınıfı olan AbstractUser sınıfından miras alıyor. Bir personelin çalıştığı pozisyonun bulunması ve tanımlanması sorumluluğu ise StaffClass sınıfında olsun. PHP 5 ile gelen __autoload fonksiyonunu tanımlamak için **autoload.php** dosyasını açalım.

```
<?php

function __autoload($name) {

	$file = __DIR__ . '/%s/%s.php';

	if (strpos($name, 'Class') !== false) {
		$file = sprintf($file, 'Classes', $name);
		if (is_readable($file)){
			require $file;
		   echo 'Gerekli olan, '.$name.' sınıfının dosyası yüklendi<br>';
		}

	}

	if (strpos($name, 'Interface') !== false) {
		$file = sprintf($file, 'Interfaces', $name);
		if (is_readable($file)){
			require $file;
		    echo 'Gerekli olan, '.$name.' arayüzünün dosyası yüklendi<br>';
		}
```

24

Çerez ve Oturum İşlemleri

Bu Bölümde

Çerezler	520
PHP'de Oturum İşlemleri	526
PHP'de Oturum Güvenliği	529
Neler Öğrendik?	533

Çerez ve oturum kavramları bir birleriyle ilişkili kavramlardır. Sırayla öğrenmeye başlayalım.

ÇEREZLER

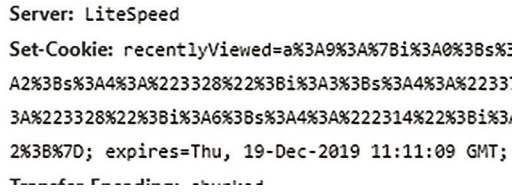
Çerezler veya bir başka deyişle tarayıcı çerezleri, web sunucusundan istemcinin web tarayıcısında saklamasını istediği küçük veri parçalarıdır. Web sunucusuna geri gönderilen her istek bu veri parçalarını içerecektir. Veriler anahtar/değer (key/value) çiftleri olarak saklanmaktadır.

Çerezler sadece küçük ve anonim veriler için tercih edilmelidir. Kişiye özel şifre, kredi kartı numarası gibi veriler çerezlerde kesinlikle saklanmamalıdır. Veyahut saklanması zorunlu ise çok iyi bir algoritma ile şifrelenmelidir.

Örneğin kullanıcı tekrar sayfanıza geldiğinde, tekrar hoş geldin demek için bir çerez bırakılabilir. Ya da bir E-Ticaret sayfasında kullanıcının incelediği ürün kimlikleri çerezlerde saklanabilir.

SETCOOKİE FONKSİYONU

setcookie, sunucu tarafından HTTP başlıklarıyla (Header) gönderilecek bir çerez tanımlar. Diğer header tanımlamaları gibi çerezler betik henüz bir HTML çıktı vermeden önce tanımlanmalıdır. Biraz daha yalın anlatmak gerekirse sunucu tarafından gönderilecek olan sayfa çerez tanımlaması içeriyor ise sayfa gönderilmeye başlanmadan önce tanımlanmalıdır. Aksi halde hem uyarı alırız hem de çerez gönderilemez.



Bir sayfanın çerez kullanıp kullanmadığını tarayıcı geliştirici konsolundan cevap başlık (response header) bilgilerinde görebilirsiniz. Sunucu tarafı haricinde de istemci tarafında JavaScript ile çerez tanımlanabilir. Setcookie fonksiyonu 1 zorunlu 6 opsiyonel argüman alır. 6 aralık 2018'de yayımlanan PHP 7.3 ile birlikte array `options` argümanı da eklendi ve toplam 8 oldu.

```
setcookie(anahtar, değer, süre, dizin, domain, güvenli, httpOnly, options);
```

Sıklıkla ilk üç anahtarı kullanılır ancak diğer argümanları da mutlaka bilmeliyiz.

Parametre	Açıklama
Anahtar	Çerezin adı (anahtarı) (string)
Değer	Çerezin değeri (string)
Süre	Çerezin geçerli olduğu süre. Saniye cinsinden tanımlanır. Bunun için çerezin sona ereceği saniyeyi Unix zaman damgası ile tanımlarız. (integer)
Dizin	Çerezin hangi dizinde geçerli olacağının belirtildiği parametredir. Ön tanımlı olarak, çerezin tanımlandığı betiğin dizini alınır. "/" tanımlanırsa tüm alan adı içinde geçerli olur. "/phpegitim/cookies/" gibi bir dizin belirtirsek çerez sadece bu dizinde kullanılabilir.
Domain	Çerezin kullanılabileceği alan adını belirtir. Örneğin 3. Taraf bir platformun sayfanızda çerez işlemleri yapabilmesi bu özellik sayesindedir. (Google Analytcs, Facebook, Criteo vb.) (string)
Güvenli	Çerezin istemciye HTTPS güvenli protokolü üzerinden mi aktarılması gerektiği bilgisidir. Boolean türünde dir. True ise çerez sadece https bağlantı durumunda gönderilecektir. (boolean)
httpOnly	Çerezler biliyoruz ki istemci tarafı programlama dili JavaScript tarafından da okunabiliyor. Bu değer true olarak gönderilirse, js ile okunamaz denilmektedir. Ancak bazı tarayıcılar tarafından desteklenmez. (boolean)
options	PHP 7.3 ile dile eklenen ve faydalı bir güvenlik önlemidir. Samesite tanımlaması yapmamızı sağlar. SameSite, tarayıcının çerezlerinin farklı siteler arası isteklerle göndermesini önler. Ana amaç, bilgi sızıntısı riskini azaltmaktır. Ayrıca, CSRF saldırısı için başlangıç seviyesinde güvenlik sağlar. Olası değerler Lax veya Strict'dir. (array)

`Setcookie` fonksiyonu boolean değer döner. Proje klasöründe `cookies` isimli bir dizin oluşturalım ve **setCookies.php** isimli bir dosya açalım.

Dosya dizini: phpegitim/cookies/setCookies.php

```
<?php

session_start();

// süre tanımlamadan atanan bir çerez
setcookie('test', 'içerik');

// süre ile atanan bir çerez
setcookie('test2', 'içerik2', time()+3600);

// özel bir süre ile ve güvenlik önlemleri ile atanan bir çerez
$expiryTimeObj = new DateTime();
$expiryTime = $expiryTimeObj
```

```
                    -> add(new DateInterval('P1M'))
                    -> add(new DateInterval('P1D'))
                    -> add(new DateInterval('PT1H'))
                    -> getTimestamp();

setcookie('test3', 'içerik3', $expiryTime, '/phpegitim/', 'localhost',
false, true /* ,['samesite' => 'strict']*/);

// çerezlere bir dizi değerin tanımlanması
setcookie('product[1]', 'ürün 1');
setcookie('product[2]', 'ürün 2');
setcookie('product[3]', 'ürün 3');

$productOff = ['name'=>'X mobile phone','price'=>'4999','offer_time'=>time()];
setcookie('productOffer', serialize($productOff), strtotime(+30 days'));
```

Bu örnekte birçok farklı senaryoda çerez atadık. İlk olarak `test` isminde ve `içerik` değerinde süre tanımlaması olmayan ve tarayıcı kapandığında sonlanacak bir çerez tanımladık. Bir sonraki senaryoda `test2` ve `içerik2` isim ve değerlerini verdikten sonra süre tanımlaması yaptık. Bu süre şuandan itibaren 3600 saniye sonrasını yani 1 saat sonrasını ifade etmektedir. Bu sebeple şu anki Unix zaman değerini `time()` fonsiyonu ile aldık ve 3600 ekleyerek ilerde olan bir saniyeyi bitiş zamanı olarak işaretlemiş olduk.

Sonraki senaryoda daha spesifik bir zaman tanımlaması yapmak istedik. Bu çerez 1 ay 1 gün ve 1 saat sonra sona ersin istiyoruz. Bunun için `DateTime` sınıfını kullandık. Metot zincirleme yöntemi ile zaman nesnesine önce bir ay sonra 1 gün ve son olarak 1 saat ekledik. Zincirin sonunda `getTimeStamp()` metodu ile nesnenin son halinin Unix zamanını elde ettik. Bu değeri `expiryTime` değişkenine atadık. Çerezi tanımlarken ise bu değişkeni süre sonu olarak belirttik. Bunun haricinde çerezin sadece `phpegitim/cookies/` dizininde değil bir üst seviye dizinde de geçerli olması için `'/phpegitim/'` dizin tanımlaması yaptık. Bu çerezin sadece **localhost** alan adında geçerli olmasını istedik.

Şayet tanımlamasa idik ön tanımlı olarak betiğin çalıştığı alan adı tanımlanacaktı. Bir sonraki parametre olan güvenli bağlantı seçeneğini false olarak tercih ettik(daha sonra true değere çevirerek test edeceğiz). Son parametreyi de true olarak değiştirerek sadece http isteklerinde kullanılmasını istedik.

Pekala bir diziyi çerez olarak saklamak istersek?. Bunun yöntemi de, dizi köşeli parantezlerini kullanmaktır. Bir sonraki örnekte product isimli bir diziyi farklı anahtarlar ile atama yaptık.

27

MVC Yazılım Mimari Deseni

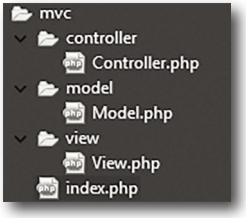
Bu Bölümde

Model Katmanı 579
Controller Katmanı 579
View Katmanı 580
Neler Öğrendik? 582

Bu Bölümde, web projelerinin yapısına oldukça uygun olan ve çok tercih edilen MVC yazılım deseninin mantığını kavramaya çalışacağız.

olursanız, ilk iki tercihim Laravel ve Phalcon. Ancak genel olarak hepsi başarılı çatılardır. Geliştirilmesi devam eden dilediğiniz çatıyı seçebilirsiniz.

Proje analiz aşamasında, ihtiyaçlara yönelik bir MVC framework seçildiği takdirde geliştirme süresini büyük oranda kısaltabiliriz. Ya da MVC deseninde ve SOLID prensipleriyle kendi çatınızı oluşturabilirsiniz. Size tavsiyem ilk olarak kendiniz bir çatı oluşturarak framework'lerin nasıl işlediğini öğrenin. Sonrasında isterseniz bir framework ile örnek; Phalcon ya da Laravel ile devam edebilirsiniz. Genel prensiplere uyduğunuz sürece hiçbir sorun yoktur. Bu başlıkta MVC katmanlarının iletişim mantığını anlamak için basit bir örnek yapacağız.



Proje dizini; **phpegitim/mvc**

mvc klasörüne model, view ve controller isimlerinde 3 klasör ve bu klasörlerin içine aynı isimlerde birer dosya oluşturalım.

Model Katmanı

İlk olarak uygulamanın kalıcı verileri ile yapılacak işlem taleplerini alan uygulayan ve ileten, model katmanı görelim. İlk olarak **Model.php** dosyasını açalım.

```
<?php

class Model {
        public $string;

        public function __construct() {
              $this -> string = 'MVC PHP için en iyi mimarilerden biridir!';
        }

}
```

`Model` isimli bir sınıf oluşturduk ve `public string` isminde bir özellik atadık. Bu özellik değerinin veritabanından getirildiğini düşünebiliriz. Kurucuda veritabanından değer getiriliyormuş gibi bir özelliği tanımladık.

Controller Katmanı

Arayüz ve Veri katmanı arasında kontrolörlük görevini üstlenen, işlem taleplerini ara yüzden alıp gerektiğinde model katmanından bilgi talep eden controller katmanıdır.

Model ve View arasında bir nevi iletişim görevini üstlenir.

MVC yapısında ana mantık Model ve View yapısının ayrılmasıdır. Bu iki yapı arasındaki haberleşmeyi sağlayan ve denetim görevi olan köprüye Controller diyoruz.

Dosya: **Controller.php**

```
<?php

class Controller {
        private $model;

        public function __construct($model) {
                $this -> model = $model;
        }

        public function clicked() {
              $this -> model -> string = 'Veri güncellendi, teşekkurler :)';
        }

}
```

`Controller` isimli bir sınıf oluşturduk. Sınıfın model adında bir özelliği var ve kurucu metotta bu özelliğe değer atanıyor. Sonrasında ise kontrolörlük görevi gereği kullanıcının `click` olayını dinleyen bir metot yazıyoruz. Bu metot çalıştığında kendisine özellik olarak atanan model'in `string` özelliği değerinin değiştirilmesi talep ediliyor.

VIEW KATMANI

Basitçe, uygulamanızın kullanıcılarınızın gözüyle gördüğü kısmıdır, ara yüz ya da ara yüz yöneticisidir.

Dosya: **View.php**

```
<?php
class View {
        private $model;
        private $controller;

        public function __construct($controller, $model) {
                $this -> controller = $controller;
```

```
                $this -> model = $model;
        }

        public function output() {

                return '<p><a href="index.php?action=clicked">' . $this ->
model -> string . '</a></p>';
        }

}
```

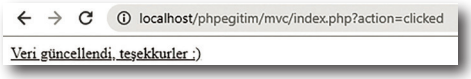
View isimli bir sınıf oluşturuyoruz. Bu sınıf ise, hem model hem de controller isimli özellikler taşıyor. Ve bu özellikler kurucu metotta tanımlanıyor. Son olarak output (çıktı) isimli bir metot oluşturarak HTML kodları çıkarıyoruz.

Dosya: **index.php**

```
<?php

require_once 'model/Model.php';
require_once 'controller/Controller.php';
require_once 'view/View.php';

$model = new Model();
$controller = new Controller($model);
$view = new View($controller, $model);

if (isset($_GET['action']) && !empty($_GET['action'])) {
    $controller->{$_GET['action']}();
}
echo $view->output();
```

İlk olarak MVC katmanlarını ifade eder dosyaları betiğe dahil ediyoruz. İlk olarak Model sınıfını türetiyoruz. Sonrasında Controller sınıfını az önce türettiğimiz model nesnesini argüman olarak göndererek türetiyoruz. Son olarak View sınıfını türetiyoruz. View sınıfına argüman olarak controller ve model nesnelerini argüman olarak gönderiyoruz. Bir koşullu ifade ile action parametresinin URL'de tanımlı olması gerektiğine ve boş olmaması gerektiğine bakıyoruz. Şayet şart sağlanırsa {} **süslü parantezler** dinamik metot ismi söz dizimi ile controller nesnesinin metodunu çalıştırıyoruz. Son olarak view nesnesinin output (çıktı) metodunu kullanıyoruz. Örneği çalıştıralım.

Çıkan sonuç model nesnesinin sağladığı veridir. Ancak veriyi `view` nesnesinin output metodu ile elde ettik. Çıkan metne tıklayarak durumu yeniden gözlemleyelim.



`Controller`, `action` parametresini dinleyerek model katmanından veriyi değiştirmesini istedi. View katmanı model katmanından veriyi değişmiş halde aldı.

Bu örnekte model aktif durumdadır. Yani view katmanı veriyi kontrolörü aradan çıkararak alabilmektedir. Diğer durum için `Controller` sınıfına `getStringData` isimli bir metot yazarak model katmanından veriyi tekrar isteyebilir, `view` katmanında da model sınıfına erişmek yerine `controller`, `getStringData` yöntemi ile aynı sonuca ulaşabilirdik. Uyguladığımız, MVC katmanlarını kavramamız için soyut bir örnek. Tamamlanmış bir MVC çatısında her katman birer bir soyut katman sınıfından türemelidir.

Örneğin: Abstract Controller (Controller) sınıfından türeyen bir gerçek controller (UserController)

Soyut katman sınıfları bir arayüz uygularlarsa daha sağlıklı olur. (örn: ControllerInterface)

Umarım MVC mimarisi hakkında biraz fikir edinebilmişizdir. Bir sonraki konuda veri tabanı tasarımını yaptığımız `todo_list` veri tabanının uygulama katmanını, şimdiye kadar öğrendiğimiz kavramlarla bir MVC çatısı oluşturarak kodlayacak ve `todoList` projesini kullanıma hazır hale getireceğiz.

NELER ÖĞRENDİK?

Kodun farklı amaçlara hizmet eden yapılarını birbirinden ayırarak, geliştirilebilir ve test edilebilir bir yapıya getirmek amacıyla tercih edeceğimiz, MVC mimari deseninin ne olduğunu anlatmaya çalıştık.