**Modern PHP**

**Chapter 1**

PHP used on web server like Apache or nginx. Also is used to build power command-line applications (just like bash, Ruby, Python, and so on.

Version Control software like Git helps maintain an auditable code history that can be branched, forked, and merged.

Test tool: PHPUnit

Deploy: PHP FastCGI process manager

Web Server: nginx

Original PHP engine: Zend Engine

Dynamic vs static PHP

Dynamic types are checked at runtime, wheras static types are checked at compile time.

**Chapter 2**

*Namespaces*

Symphony/httpfoundation uses common php class names like *Request, Response, and Cookie*

[*https://github.com/symfony/http-foundation*](https://github.com/symfony/http-foundation) *> Response.php*

A namespace (or subnamespace) encapsulates and organizes related PHP classes, just as a filesystem directory contains related files.

Subnamespaces are separated with a \ character. Techinically speaking, namespaces are merely a PHP language notation referenced by the PHP interpreter to apply a common name prefix to a set of classes, interfaces, functions, and constants.

Namespaces allow us to create sandboxed code that works alongside other developer’s code.

Without namespaces, a name collision causes PHP to fail!

*Declaration*

Every PHP class, interface, function, and cosntatn lives beneath a namespace (or subnamespacese). Namespaces are declared at the top of a PHP file on a new line immediately after the opening <?php tag. Namespaces are often used to establish a top-level vendor name.

For example:

*<?php*

*Namespace Oreilly;*

Subnamespaces example:

*<?php*

*Namespace Oreilly/ModernPHP;*

All classes in the same namespace or subnamespace don’t have to be declared in the same PHP file. You can specify a namespace or subnamespace at the top of any PHP file, and that file’s code becomes a part of that namespace or subnamespace. This make it possible to write multiple classes in separate file that belong to a common namespace.

*Import and Alias*

Zend\_Cloud\_DocumentServcie\_Adapter\_WindowsAzure\_Query corresponds to the PHP file   
Zend/Cloud/DocumentService/Adapter/WindowsAzure/Query.php

By alias, I mean that I tell PHP that I will reference an imported class,interface, function, or constant with a shorter name.

To Use a function

<?php

Use func Namespace\CONST\_NAME;

Echo CONST\_NAME;

To use a constant

<?php

Use constant Namespace\CONST\_NAME;

Echo CONST\_NAME;

*Multiple Imports*

<?php use path..

Path..

Path..

Instead of

<?php use path..

use path..

use path..

*Multipler Namespaces in one file*

<?php

Namespace Foo {

// Declare classes, interfaces, functions, and constants here

}

Namespace Bar {

// Declare classes, interfaces, functions, and constants here

}

*Global Namespace*

If you reference a class, interface, function, or constant without a namespace, PHP assumes the class, interface, function, or constant live in the current namespace. If you need to reference a namespaced class, interface, function, or constant inside another namespace, you must use the fully qualififed PHP class name (namespace + class name).

Some code might not have a namespace and, therefore, lives in the global namespace.

Example 2-4. Unqualified class name inside another namespace

<? php namespace My\ App;

class Foo

{

public function doSomething()

{ $ exception = new Exception(); }

}

*Code to an Interface*

So what is a PHP interface? An interface is a contract between two PHP objects that lets one object depend not on what another object is but, instead, on what another object can do. An interface decouples our code from its dependencies, and it allows our code to depend on any third-party code that implements the expected interface.

named DocumentStore that collects text from different sources: it fetches HTML from remote URLs; it reads stream resources; and it collects terminal command output. Each document stored in a DocumentStore instance has a unique ID. Example   2-6 shows the DocumentStore class.

How exactly does this work if the addDocument() method only accepts instances of the Documentable class? That’s a good observation. However, Documentable is not a class. It’s an interface, and it looks like Example   2-7.

*Why we use traits*

The classical inheritance model works well most of the time. However, what do we do if two unrelated PHP classes need to exhibit similar behavior? For example, a PHP class RetailStore and another PHP class Car are very different classes and don’t share a common parent in their inheritance hierarchies. However, both classes should be geocodable into latitude and longitude coordinates for display on a map. Traits were created for exactly this purpose. They enable modular implementations that can be injected into otherwise unrelated classes. Traits also encourage code reuse.

My first (bad) reaction is to create a common parent class Geocodable that both RetailStore and Car extend. This is a bad solution because it forces two otherwise unrelated classes to share a common ancestor that does not naturally belong in either inheritance hierarchy.

My second (better) reaction is to create a Geocodable interface that defines which methods are required to implement the geocoding behavior. The RetailStore and Car classes can both implement the Geocodable interface. This is a good solution that allows each class to retain its natural inheritance hierarchy, but it requires us to duplicate the same geocoding behavior in both classes.

My third (best) reaction is to create a Geocodable trait that defines and implements the geocodable methods. I can then mix the Geocodable trait into both the RetailStore and Car classes without polluting their natural inheritance hierarchies.

The Geocodable trait defines only the properties and methods necessary to implement the geocodable behavior. It does not do anything else. Our Geocodable trait defines three class properties: an address (string), a geocoder object (an instance of \Geocoder\ Geocoder from the excellent willdurand/ geocoder component by William Durand), and a geocoder result object (an instance of \Geocoder\ Result\ Geocoded). We also define four public methods and one protected method. The setGeocoder() method is used to inject the Geocoder object. The setAddress() method is used to set an address. The getLatitude() and getLongitude() methods return their respective coordinates. And the geocodeAddress() method passes the address string into the Geocoder instance to retrieve the geocoder result.

*How to use a trait*

Using a PHP trait is easy. Add the code use MyTrait; inside a PHP class definition. Here’s an example. Obviously, replace MyTrait with the appropriate PHP trait name:

<? php class MyClass { use MyTrait; // Class implementation goes here }

*Generators*

Unlike your standard PHP iterator, PHP generators don’t require you to implement the Iterator interface in a heavyweight class.

class. Instead, generators compute and yield iteration values on-demand. This has profound implications for application performance. Think about it. A standard PHP iterator often iterates in-memory, precomputed data sets. This is inefficient, especially with large and formulaic data sets that can be computed instead. This is why we use generators to compute and yield subsequent values on the fly without commandeering valuable memory.

*Create a Generator*

Generators are easy to create because they are just PHP functions that use the yield keyword one or more times. Unlike regular PHP functions, generators never return a value. They only yield values.

This object can be iterated with the foreach() function.

Generators do not add functionality to PHP. You can do what generators do without a generator. However, generators greatly simply certain tasks while using less memory. If you require more versatility to rewind, fast-forward, or seek through a data set, you’re better off writing a custom class that implements the Iterator interface, or using one of PHP’s prebuilt Standard PHP Library (SPL) iterators.

*Closures*

A closure is a function that encapsulates its surrounding state at the time it is created.

Closures and anonymous functions are, in theory, separate things. However, PHP considers them to be one and the same. So when I say closure, I also mean anonymous function. And vice versa.

**Create**

I typically use PHP closure objects as function and method callbacks. Many PHP functions expect callback functions, like array\_map() and preg\_replace\_callback(). This is a perfect opportunity to use PHP anonymous functions! Remember, closures can be passed into other PHP functions as arguments, just like any other value. In Example   2-20, I use a closure object as a callback argument in the array\_map() function.

*Attach State*

Let’s explore how to attach and enclose state with a PHP closure. JavaScript developers might be confused by PHP closures because they do not automatically enclose application state like true JavaScript closures. Instead, you must manually attach state to a PHP closure with the closure object’s bindTo() method or the use keyword.

It’s far more common to attach closure state with the use keyword, so let’s look at that first (Example   2-21).

***Zend OPcache***

PHP is an interpreted language. When the PHP interpreter executes a PHP script, the interpreter parses the PHP script code, compiles the PHP code into a set of existing Zend Opcodes (machine-code instructions), and executes the bytecode.

A bytecode cache stores precompiled PHP bytecode. This means the PHP interpreter does not need to read, parse, and compile PHP code on every request. Instead, the PHP interpreter can read the precompiled bytecode from memory and execute it immediately. This is a huge timesaver and can drastically improve application performance.

If you compile PHP yourself (i.e., on a VPS or dedicated server), you must include this option in your PHP ./ configure command: --enable-opcache After you compile PHP, you must also specify the path to the Zend OPcache extension in your php.ini file with this line: zend\_extension =/ path/ to/ opcache.so

Lockhart, Josh (2015-02-16). Modern PHP: New Features and Good Practices (Kindle Locations 1074-1079). O'Reilly Media. Kindle Edition.