

# Object-Oriented PHP

Writing Resilient & Reusable Code in PHP 7

Junade Ali

# **Object-Oriented PHP**

Writing Resilient & Reusable Code in PHP 7

Junade Ali

ISBN 978-0-244-90350-3

Associated source code files for this book are available over Git and can be found by visiting <https://ju.je/object-orientedphp-book-code> online.

© 2017 Northern Optic Limited

# Contents

<b>Introduction</b> . . . . .	<b>1</b>
Conventions . . . . .	2
Code Style . . . . .	2
Errata and Comments . . . . .	3
<b>Back to Basics</b> . . . . .	<b>5</b>
Composer Package Manager . . . . .	5
Classes . . . . .	10
Scalar Types Hinting . . . . .	12
Interfaces . . . . .	15
Abstract Classes . . . . .	18
Inheritance . . . . .	19
Coding Standards . . . . .	26
Conclusion . . . . .	28
<b>Advanced Concepts</b> . . . . .	<b>29</b>
Magic Methods . . . . .	29
Polymorphism . . . . .	31
Dependency Injection . . . . .	35
Generators . . . . .	41
Composition vs Inheritance . . . . .	43
Traits . . . . .	54
Conclusion . . . . .	58
<b>SOLID Design Principles</b> . . . . .	<b>59</b>
Single Responsibility Principle . . . . .	59
Open/Closed Principle . . . . .	67
Liscov Substitution Principle . . . . .	69
Interface Segregation Principle . . . . .	72
Dependency-Inversion Principle . . . . .	74
Conclusion . . . . .	78
<b>An Introduction to Design Patterns</b> . . . . .	<b>79</b>
Design Pattern Principles . . . . .	80

## CONTENTS

Creational Design Patterns . . . . .	80
Structural Design Patterns . . . . .	87
Behavioural Design Patterns . . . . .	93
Conclusion . . . . .	99
<b>Testing . . . . .</b>	<b>101</b>
Unit Testing . . . . .	101
Conclusion (Testing Matters!) . . . . .	112
<b>How to Write Better code . . . . .</b>	<b>113</b>
Refactoring . . . . .	113
PHP-MD . . . . .	118
Conclusion . . . . .	120

# Introduction

After finishing my previous book “Mastering PHP Design Patterns”, I suddenly felt as if I had lost a major activity for transatlantic flights. I have never been a fan of on-board Wi-Fi, there is undoubtedly a huge amount of writing that can be achieved in the 12 hours of disconnect between London Heathrow and San Francisco International, especially if you’re an insomniac.

There is however, a far more important reason for writing this book. I recently published a blog post regarding containing some PHP Interview Questions, containing a list of interview questions/answers that senior PHP Developers who work on Enterprise grade software should be able to answer. A number of people then reached out to me asking good material where they could learn the fundamentals of Object Oriented Programming whilst working with PHP. I searched, but failed to find a book on PHP Object Orientation that I could confidently recommend, so I decided to write my own.

This book will seek to cover the fundamentals of Object Oriented Programming strictly within the context of PHP. This book will be written in PHP 7, due to the fact that this version of PHP introduced a number of features which are incredibly useful when writing Object Oriented code.

Going beyond the basics, we’ll cover more detailed topics such as Dependency Injection, SOLID Principles and Design Patterns. We’ll also cover how you can use some open-source tools to improve the reliability of your software and using detect bad practice within your codebase.

I am a great believer in teaching my demonstration, this book will work on the same principle. Each chapter will try to be as self-contained on the topic as possible and heavily supported with code examples, if there is a concept or language feature you struggle with, please run the code example and have a play around. Try and break the code and see what’s gone wrong, and then try and fix it.

## Purpose

Sometimes it is easier to write badly written code than well-written code, indeed it may even be faster to do so at the start of a project. However, code which is badly written gets harder to change as time goes on and the design deteriorates; well-designed code, however, remains relatively easy to adjust and add new features to. Fundamentally, bad code stops a business being able to react to forces of change.

Recently we’ve heard Agile being used as a Project Management buzzword a lot, but behind the buzz there is a fundamentally sound principle. Agile software development is rooted in the ability for a project to react to changing requirements and ultimately deliver value on a more reliable basis. However, in order to successfully implement Agile, your code must be resilient to the forces of change, you must be able to re-design your code mid-flow such that a change in feature can be

elegantly implemented into your codebase. This is fundamentally the business benefit of having well-designed code; resiliency and increased speed of delivering new functionality.

If you're an experienced developer, chances are you've seen teams rewrite bad code for what eventually becomes equally bad code. You've likely seen the staff-churn from developers quitting over having to work long hours in stressed environments whilst struggling to deliver code. Whilst not a book on refactoring, this book seeks to describe the best practise approaches for Object-Oriented software development, allowing you to ensure that your code is resilient to the forces of change and your codebase can grow as easily today as it did yesterday.

The key to writing well-written code in modern PHP is a sound understanding of Object-Orientation - this book seeks to marry the theory behind Object-Orientation and how you can practically implement this in PHP 7. This book does not seek to be a PHP 101 book, but seeks to build on your existing experience to introduce new patterns and practises into your workflow.

## Conventions

This book is written to version 7 and above of the PHP language, the reason for this is that there are some significant changes and improvements in PHP 7 as to how Object Orientation is implemented.

In many cases this book will be written in the format of a tutorial from chapter to chapter, the source code is available on GitHub and can be accessed by visiting <https://ju.je/object-orientedphp-book-code>

## Where's the UML?

Traditionally, books on Object-Oriented Programming will be filled with UML diagrams. This was a convention I broke previously in my last book and it is indeed a convention I intend to continue to break throughout this book.

As Agile software development practices become more common, large upfront design is becoming a relic of the past. Whilst deployment diagrams are still in use, it's rare to see a team of developers pondering a UML class diagram.

This is a book about Object-Oriented Programming *in PHP*, therefore we have already established a common language for us to share ideas through. I can write PHP which you can read, take apart and play with. There is no need to add additional complexity by communicating ideas through yet another language.

## Code Style

PHP is presented as follows:

```
1 <?php
2 echo "hello";
```

Shell scripts will be indicated as follows:

```
1 echo "192.168.99.100 project.local" >> /etc/hosts
```

Command line inputs and outputs will be marked as follows:

```
1 # dig A google.com +short
2 216.58.204.78
```

Please note that # marks a privileged shell and \$ marks an unprivileged shell

## Errata and Comments

Errata will be handled promptly, and can be emailed to me directly at [mjsa@junade.com](mailto:mjsa@junade.com).





# Back to Basics

This first chapter aims to be a self-contained introduction to the language features that allow us to use Object-Oriented in modern PHP. We'll set-up a basic PHP project and walk through some OOP (Object-Oriented Programming) principles. Along the way, we'll stumble on some other bits of non-OOP knowledge that will benefit you through the course of this book.

We'll be working inline with PHP 7 in this book, so if you want to follow along, make sure your PHP installation is at a minimum of 7.0.

## Composer Package Manager

Let's get started with our first project. A key tool to grasp the use of is Composer, a Dependency Manager for PHP. Using Composer we can include third-party libraries into our codebase and also perform Autoloading, which we'll discuss later.

Using third-party code in your codebase a very good idea in a lot of cases. There are developers who insist on building their own everything, they fundamentally adopt a world view that everything is better if they build it themselves; this is referred to "Not Invented Here" Syndrome. This outlook can prove to be problematic for a number of reasons; external implementations have often be scrutinised and refined to an extent your own never will be. Additionally building your own implementations of things that are commonly used by other developers can lead to a maintenance burden. It is often to extend a library that does 95% of what you want it to do, rather than build your own. Composer allows us to effectively manage external dependencies and the manage which versions of those dependencies we pull in.

Instead of us having to copy and paste classes into our codebase, or use PEAR, Composer offers us a fairly elegant way of managing project dependencies and their versions. If you come from a JavaScript background you'll find it similar to NPM.

## Installing Composer

In order to get started, we need to have Composer installed on our system. You can install Composer on Mac or Linux by running the following commands on your system. For installing the current version of Composer on my Mac, I ran the following commands:

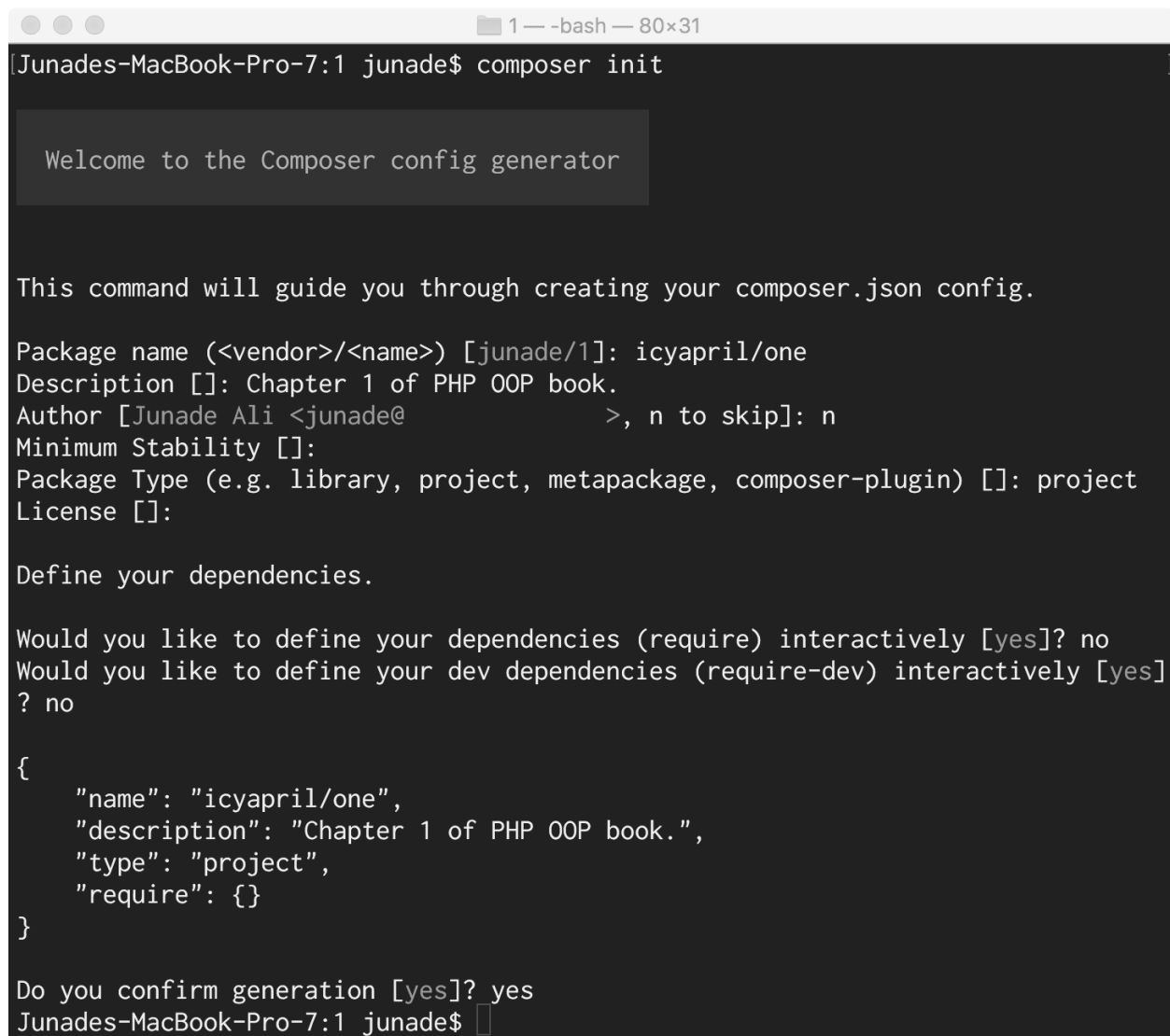
```
1 curl -sS https://getcomposer.org/installer | php
2 sudo mv composer.phar /usr/local/bin/
```

Further information about Composer alongside how you can install and use it on Windows, is available at [getcomposer.org](http://getcomposer.org).

After you have installed Composer, you should be able to run `composer` within the terminal or Command Prompt.

## Creating our Project

We can interactively set-up our `composer.json` file running `composer init` in our project directory. This will then open an interactive prompt from where we can configure our composer dependencies. At the very bottom we'll see a preview of the JSON that will be written to our `composer.json` file and then we can confirm it's creation.



```
1 — -bash — 80x31
[Junades-MacBook-Pro-7:1 junade$ composer init]

Welcome to the Composer config generator

This command will guide you through creating your composer.json config.

Package name (<vendor>/<name>) [junade/1]: icyapril/one
Description []: Chapter 1 of PHP OOP book.
Author [Junade Ali <junade@>, n to skip]: n
Minimum Stability []:
Package Type (e.g. library, project, metapackage, composer-plugin) []: project
License []:

Define your dependencies.

Would you like to define your dependencies (require) interactively [yes]? no
Would you like to define your dev dependencies (require-dev) interactively [yes]
? no

{
  "name": "icyapril/one",
  "description": "Chapter 1 of PHP OOP book.",
  "type": "project",
  "require": {}
}

Do you confirm generation [yes]? yes
Junades-MacBook-Pro-7:1 junade$
```