THE EXPERT'S VOICE® IN OPEN SOURCE

Pro PHP Programming

Peter MacIntyre, Brian Danchilla, and Mladen Gogala

Apress®

Pro PHP Programming

Peter MacIntyre
Brian Danchilla
Mladen Gogala

Apress®

Pro PHP Programming

ISBN 978-1-4302-3560-6

ISBN 978-1-4302-3561-3 (eBook)

President and Publisher: Paul Manning
Lead Editor: Frank Pohlmann
Technical Reviewer: Thomas Myer
Editorial Board: Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Jonathan Gennick, Jonathan Hassell, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Frank Pohlmann, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Coordinating Editor: Jessica Belanger
Copy Editor: Tracy Brown
Production Support: Patrick Cunningham
Indexer: SPi Global
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media, LLC., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `www.springeronline.com`.

For information on translations, please e-mail `rights@apress.com`, or visit `www.apress.com`.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at `www.apress.com/bulk-sales`.

The source code for this book is available to readers at `www.apress.com`. You will need to answer questions pertaining to this book in order to successfully download the code.

Having dedicated my other writings to my wife Dawn and our kids, I would like to dedicate this book to all those in the PHP community who are keeping this language fresh, robust, and ever growing. To the open source community and its ideals and concepts—may it continue ad infinitum!

–Peter

For mom and dad

–Brian

To my beloved wife and son

–Mladen

Contents at a Glance

Contents

About the Authors

Peter MacIntyre has over 20 years' experience in the information technology industry, primarily in the area of software development.

Peter is a Zend Certified Engineer (ZCE), having passed his PHP certification exam. He has contributed to many IT industry publications, including *Using Visual Objects* (Que, 1995), *Using PowerBuilder 5* (Que, 1996), *ASP.NET Bible* (Wiley, 2001), *Zend Studio for Eclipse Developer's Guide* (Sams, 2008), *Programming PHP* (Second Edition) (O'Reilly Media, 2006), and *PHP: The Good Parts* (O'Reilly Media, 2010).

Peter has been a speaker at North American and international computer conferences, including CA-World in New Orleans, USA; CA-TechniCon in Cologne, Germany; and CA-Expo in Melbourne, Australia. Peter lives in Prince Edward Island, Canada, where he is the Senior Solutions Consultant for OSSCube (`www.osscube.com`), a world leader in open source software development and consultancy. He assists OSSCube with running its Zend Center of Excellence. Peter can be reached at: `peter@osscube.com`.

Brian Danchilla is a Zend Certified PHP developer and seasoned Java programmer, and holds a BA in computer science and mathematics. Danchilla has been writing computer programs for more than half his life, including web applications, numerical analysis, graphics, and VOIP (Voice Over IP) programs. Danchilla has a strong ability to learn new technologies and APIs. He is an avid technical reader with a strong sense of the elements that make a compelling read. Through his work as a university teaching assistant, private tutor, and PHP workshop leader, Danchilla has honed the ability to transfer knowledge in an accessible way. Danchilla can also be found actively contributing to the stackoverflow community. When not programming, he likes to spend time playing guitar or being outside.

Mladen Gogala is long-term database professional who has had a long and distinguished career as an Oracle DBA, Linux, and Unix system administrator, VAX/VMS system administrator and, recently, database performance architect. He has been working with multi-terabyte databases, primarily of the Oracle variety, since the late 1990s. He knows Linux, Perl, and PHP. The latter became his favorite language in the early 2000s, and he is the author of *Easy Oracle PHP: Create Dynamic Web Pages with Oracle Data* Rampant Techpress, 2006). He has also written several articles about PHP, Oracle, and Symfony. Mladen was born in 1961 in Zagreb, Croatia.

About the Technical Reviewer

Thomas Myer is a technical author, consultant, and developer. He spends most of his time working on PHP projects (particularly CodeIgniter, ExpressionEngine, WordPress, and MojoMotor), but is also known to dabble in Python, Perl, and Objective-C projects.

Follow Thomas on twitter (if you dare) as `@myerman`. Don't forget to check out `www.tripledogs.com` for more on Triple Dog Dare Media, which he founded in 2001.

Thomas currently lives in Austin, Texas, with wife, Hope, and dogs, Kafka and Marlowe.

Foreword

Because of PHP's humble beginning as a hackers' project – an attempt to develop an easy and enjoyable way to develop web sites – nobody expected it to become nearly as popular as it is today.

Over the years we've used many different metrics to measure PHP's popularity, looking at the number of web sites that have PHP deployed on them, the number of PHP books on sale at Amazon.com, the amount of prominent companies using PHP, the number of PHP-based projects, the size of the communities that create them, and so on.

And then, there was one other, much less "scientific" metric.

Back in 2008, when I was on my honeymoon with my wife, Anya, we stayed at a small hotel called Noster Bayres in Buenos Aires. We arrived after a long flight, visiting a brand-new country full of new faces and things we'd never seen before. Imagine my surprise when, after I filled in my hotel registration form, the receptionist asked me if I was Suraski, "that PHP guy." It turned out that he was developing a social network for the San Telmo neighborhood in PHP.

Although all of the previous metrics were rock-solid proof of PHP's extreme reach, importance, and popularity, for me, this incident in a small hotel halfway across the world sealed the deal. If the receptionist at that hotel was writing PHP, we were most certainly mainstream.

Almost three years later, advanced PHP skills are essential to any power web developer, and arguably – with the explosive growth of web and HTTP-based communications – to any and all developers. *Pro PHP Programming* guides you through some of the more advanced aspects of modern PHP development, including object orientation, mobile application development, and scalable data sources that can be important for cloud-enablement. I'm sure the knowledge you'll gain will be an important part of your toolset going forward, and will help you avail of the advanced features of PHP 5.3 to their fullest. Happy PHP-ing!

Zeev Suraski, CTO, Zend

Acknowledgments

I would like to thank Frank Pohlmann and Jessica Belanger at Apress, who were instrumental in getting this book off the ground and into the hands of the PHP community. Having written a few other PHP books for various publishers, I was initially reluctant to commence this additional writing task, but Frank twisted my arm and got me to commit. I thank him for the encouragement and opportunity; we have become good friends in the process as well, so there is more value in this project than just writing more about PHP.

The technical editor, Thomas Meyer, and the copy editor, Tracy Brown, also did a bang-up job, and I tip my hat to you all as well.

To my coauthors, thanks! This has been a great journey and I have grown and learned a lot from each of you. Working with authors with different backgrounds, nationalities, and expertise is always a pleasure and a growth exercise.

Peter MacIntyre

I would like to thank my companion, Tressa, for supporting and putting up with me while I hid away to work on this book. Thanks to my mom and dad, my brother, Robert, and sister, Karen, for always believing in me and what I do – even though they do not know what I do, exactly. I would also like to thank my coauthors, Peter and Mladen, and the entire Apress team.

Brian Danchilla

I have learned a great deal from those who have worked with me over the years, and I gratefully acknowledge my debt to them, especially my colleagues from Video Monitoring Services, Vinod Mummidi and Arik Itkis, with whom I engaged in endless discussions about the language concepts. My manager, Gerry Louw, was also very supportive and helpful. I would also like to express my thanks to our coordinating editor, Jessica Belanger of Apress, for her enthusiastic and expert guidance, without which this book wouldn't have happened, and to Peter MacIntyre and Brian Danchilla, my coauthors, for tirelessly proofreading the drafts and giving me suggestions that were crucial for the book. I am also truly grateful to Tom Welsh and Tim Hawkins for all their efforts and good ideas, many of which have found their rightful place in this book. Last, but certainly not least, I have to express my eternal gratitude to my wife, Beba, and son, Marko, for their love, support, and patience during the writing of this book.

Mladen Gogala

Introducing PHP

W elcome to yet another book on the great programming language of PHP. This book is unique in that it focuses on higher-end materials and more advanced, cutting-edge topics. We have kept it as modern as possible with the fast-paced world of the Internet. We take the reader from an intermediate level to a more advanced level of this great programming language.

Origins of PHP

PHP began as a project led and designed by Mr. Rasmus Lerdorf. In June 1995, he released version 1.0 of Personal Home Page Tools (its original product name). It was a small collection of functions that helped to automate the creation and maintenance of simple home pages on the then-burgeoning Internet. Since then, PHP has grown by leaps and bounds to where it is today at version 5.3.4 (at the time of writing). PHP was one of the first web development programming languages to be open source from the outset. Lerdorf was visionary enough to see the need and the potential for a tool and language that could grow with this vein of the Internet community and expand far beyond it as well.

What Is PHP?

So then, what exactly is PHP? What does it look like and "feel" like in its current version? Well, in its simplest terms, PHP is merely an HTML markup generator. If you look at the source code of a PHP-generated web page, you will see only HTML tags; maybe some JavaScript as well, but no raw PHP code. Of course, that is an overly simplistic view of the language that has captured between 35 and 59 percent (depending on the source) of the languages in use for web development. Whatever number you settle on, PHP is the single most popular web development language on the market today.

When I use the term "on the market," you also have to appreciate that PHP is free. Yes, free! It is an open source product, so in reality there isn't an actual market for it. So it has done very well in terms of popularity and range of use for a product that is led and steered by no one entity or personality.

■ **Note** For more information on open source, be sure to read "The Cathedral and the Bazaar" by Eric S. Raymond for comparisons of open source products (Bazaar) and closed source products (Cathedral). You can find it here: `www.catb.org/~esr/writings/cathedral-bazaar/`.

Actually, Zend Corporation (zend.com) is probably the leader of the PHP world in that it has built many additional products to support and enhance PHP, and it is a key player in its guidance, since the two founding members of the company – Zeev Suraski and Andi Gutmans – have really taken up the gauntlet since version 3 of the product.

PHP is also very open and forgiving in its language structure, in that it is loosely typed (among other things). This means that variables don't have to be defined in the type of data that they will hold prior to their use in the way that some other programming languages do. Rather, it interrogates the data and tries to determine its data type based on the content the variable is holding at the time. This means, for example, that a variable called $information can have many different values during the execution of a code file. This can also be a drawback in some ways, because the data could change during the running of the code and therefore negate some code segments that may be expecting an integer but receiving a string.

PHP can also be written with an object-oriented programming (OOP) design in mind. Classes, properties, and methods; inheritance, polymorphism, and encapsulation are all part of the language. This adds a lot of robustness and re-use to code and allows for more ease of use overall. Of course, the OOP approach to programming has been around for a long time in technology-years, and PHP has been adopting and expanding on its integration for a few good years now as well.

Another valuable feature that PHP possesses is that it can be run from the command prompt (Linux or Windows), and can therefore be used in scheduled un-attended (CRON) scripts. This added level of flexibility is wonderful because you (the programmer) do not have to learn another language to accomplish different tasks while working on the server environment. You can generate web pages with the same language that you use to manage the file system (if you so choose).

PHP also has many integration points; it is a very open language, to say the least. PHP can be used for many things other than straight web development. Combine it with a database source through an appropriate connecting library, and you can have a very dynamic web presence even a web application. Combine it with an additional library (tcpdf, for example), and you can generate Adobe PDF documents on the fly. These are just two examples and we will be covering a lot of these add-on libraries throughout this book, so stay tuned!

High- Level Overview of This Book

So what do we hope to accomplish with this book for you, the reading programmer? We have made every effort to make this diatribe of current, cutting-edge value so that you will be aware of and able to use some of the most recent features and integrations of PHP. We are not spending time on the more simplistic topics of the language, such as what is a variable or how to write for / next loops.

It is our desire that you become a more advanced PHP programmer overall, and that this material may even assist you in becoming ready to take and pass the Zend Certified Engineer's exam. Following is a brief summary of what will be covered in each chapter.

Chapter 1: Object Orientation

This initial chapter is designed to get you ready for many of the concepts and code examples that will be coming in the remainder of the book. We introduce some basic concepts of OOP and how it is implemented in PHP, and then get into some of the more advanced issues right away. Be sure you really understand this chapter before going too far into the following chapters.

Chapter 2: Exceptions and References

Here we follow up on some of the OOP concepts and get into exception coding with try / catch blocks. This is a more elegant way of handling potential errors in your PHP code, and it is quite a powerful approach once it is mastered. This is followed by a discussion on reference coding and what it means in relation to the classes and functions that you may be using.

Chapter 3: PHP on the Run (Mobile PHP)

This world is getting more mobile-dependant; we see smaller and more powerful devices being released all the time. Apple, RIM, HTC, and many others are trying to capture the mindshare of this lucrative market. But there need to be applications available for these devices, and in this chapter we show you some ways that PHP is growing and adapting to also embrace this shift in mobility.

Chapter 4: Social Media and PHP

In a similar vein of technological growth, the rapid expansion of social media use is also being greatly assisted by PHP. Most of the forward-facing aspects of Facebook, for example, are written in PHP. Many other sites like Flickr, portions of Yahoo!, and even many blog applications are heavily dependent on PHP. In this chapter, we look at some of the interfaces that exist for integration with these social media sites.

Chapter 5: Cutting-Edge PHP

In its current release at the time of writing, version 5.3.4, PHP has many new features added to its actual language. Many of these features were slated for the long-awaited version 6.0, but because some of the features were ready before others, this initial collection was released as 5.3. In this chapter, we will be looking at some of the "best" of these new features and how they can be employed in your web projects.

Chapter 6: Form Design and Management

Here we take a little more time going over the features and techniques that can be implemented in designing and managing data entry forms. Controlling the data that is entered into them, responding to bad data (invalid date formats for example), and how to gracefully get that data into a web system.

Chapters 7 and 8: Database Interaction

Of course, one of the major aspects to web development these days is the ability to store and display data that is coming from a data source. In these two chapters, we look at the many different ways that data can be manipulated. From smaller footprint databases like those of the NoSQL variety to the big iron database engines like MySQLi and the techniques we can gather from using additional tools like PDO and Sphinx.

Chapter 9: Oracle

PHP and Oracle have a special connection when it comes to extra-large data sets. In this chapter, we are looking at matters that are specific to this relationship and how to make the most of their "union."

Chapter 10: PHP Libraries

As was mentioned already, PHP is very open to working with other libraries. In Chapter 10, we take a look at some of the more popular and advanced of these libraries. Being able to generate PDF forms on the fly, consume RSS feeds, generate professional e-mail, and integrate with Google maps are just a few of the library integrations that will be covered in this chapter.

Chapter 11: Basic PHP Security

Naturally it would not be a complete book if we did not cover the latest techniques in web security. Chapter 11 covers this large topic. We look at the most secure (currently) encryption algorithm called SHA-1. Other topics covered are protecting data that is being input to the web system as well as data that is going out from the web system.

Chapter 12: Team Development with Zend Studio

This chapter goes on a little tangent in that it is not purely a PHP topic. Here we look at how to use one of the more popular Integrated Development Environments (IDEs) for PHP development, Zend Studio for Eclipse. With Zend Studio, we look at how a team of developers can work together in an agile way (have you heard of Extreme Programming?) We will look at the use of SVN, Bugzilla, and MyLyn all working in concert to make the job of a team more productive on many fronts.

Chapter 13: Refactoring Unit Testing

This is actually an extension of what is covered in the previous chapter. There is more coverage here on what can be done to make PHP development more agile in how to program it. Refactoring and unit testing are the focus here, and you will learn how to make good use of them both in your day-to-day coding projects.

Chapter 14: XML and PHP

XML use has certainly become more mainstream over the years since it first became a buzzword. In this chapter, we look at how to use SimpleXML to consume XML from an outside source. We also cover the ability to generate XML data from within our own systems for use by others.

Chapter 14: JSON / Ajax

Again, we take a little step away from pure PHP with the look into the JSON library and how we can use it along with Ajax to make our web applications more responsive.

Chapter 15: Conclusion

In this last chapter, we look at additional resources for PHP that we could not fit into this book. Here we look at the many web resources available and some of the magazines and conferences that can deepen your knowledge and understanding of this great language and community.

The Future of PHP

This is a topic that I find hard to write about. With PHP being a true open source product, it is hard to really predict what direction the community will take in the near and distant future. I have implicit faith in this community however; in the years that I have been a PHP programmer, I have yet to really see a misstep taken by this collective. I know that the mobile aspect of our lives will continue to grow and expand, and PHP is already taking steps to fully embrace this truth. What else will happen in the near future? Maybe some more integration with telephony in the aspect of smart-phones and data interoperability. Possibly more expansion into voice recognition technology and web applications—who knows? I do know from my experiences so far that PHP and its supporting community will continue to have its finger on the pulse of the world of technology and they will not let us down.

Looking to the future of PHP is a comforting thing to do; it's like looking at a beautiful sunrise knowing that the coming day can only get better as it goes along.

CHAPTER 1

Object Orientation

The purpose of this chapter is to introduce the basic concepts of object orientation. What does it actually mean to say, "PHP is object oriented?" The simplest answer is that PHP allows for the definition and hierarchical organization of user data types. This book is about PHP 5.3, which introduced some new elements to PHP object apparatus. PHP underwent a fairly radical change since the version 4, which also included rudimentary object-oriented (OO) capabilities. In PHP 4, for instance, it wasn't possible to define visibility of methods and members. In PHP 5.3, namespaces were added.

In this chapter we will introduce the ideas of classes, inheritance, object creation, and interface definition. We will also introduce some less elementary stuff, like iterators. So, let's get started.

Classes

Classes are simply user defined types. In OO languages, a class serves as a template for creating objects or instances (functional copies) of that class. A class contains the description of the common characteristics of all items belonging to it. The purpose of a class (or classes) is to encapsulate object definition and behavior, and to hide its actual implementation from the end user and to enable the end user to employ the class objects in the documented and expected way. Encapsulation also makes programs smaller and more manageable, because the objects already contain the logic needed to handle them. There is also a feature called *autoloading* that helps with breaking scripts into smaller, more manageable pieces.

Before we see a simple example of a PHP class, let's introduce some more terminology:

- Class member or property: A variable, data part of the class
- Class method: A function defined within a class

Now we will define a class for a point in a 2-dimensional plane, defined with its Cartesian coordinates (see Listing 1-1). As it is designed purely for instructional purposes, this class has several serious drawbacks. We recommend that you don't use it as a code base for any code of your own.

Listing 1-1. *A 2D Plane*

```
<?php
class Point {
    public $x;
    public $y;
```

```
    function __construct($x,$y) {
        $this->x=$x;
        $this->y=$y;
    }
    function get_x() {
        return($this->x);
    }
    function get_y() {
        return($this->y);
    }
    function dist($p) {
        return(sqrt( pow($this->x-$p->get_x(),2)+
                     pow($this->y-$p->get_y(),2)));
    }
} // Class ends here
$p1=new Point(2,3);
$p2=new Point(3,4);
echo $p1->dist($p2),"\n";
$p2->x=5;
echo $p1->dist($p2),"\n";
?>
```

This class is not trivial; there are quite a few things to analyze and fix. First, as we have previously stated, this class describes a point in a plane, defined by its Cartesian coordinates, `$x` and `$y`. There is a keyword `public` to which we will return later. There is also a constructor method `__construct`, which is called when a new object (or *instance*) of the class `Point` is created in memory by invoking the operator `new`. In other words, when the line `$p1=new Point(2,3)` is executed, the method `__construct` is automatically referenced and executed, and the arguments behind the class name, in parenthesis, are passed into the `__construct` method for possible use.

The method `__construct` references the variable `$this`. The variable `$this` is the OO way of referring to the class instance itself. It always refers to the current object in focus. It is an OO equivalent of "me." A variant of this variable is present in almost all OO-based languages, although it is called "self" in some languages.

The class constructor is the method that initializes (instantiates) objects of the given class. In this particular case, it assigns coordinates. The coordinates (the variables named `$x` and `$y`) are members of this class. Several other methods are also defined, two `get` methods and a method called `dist`, which calculates the distance between two points.

The next thing to observe is the keyword `public`. Marking members as "public" allows the full access to the data members that are marked public. In our script, there is a line that reads `$p2->x=5;`. The x-coordinate of one of our points is being manipulated directly. Such access is impossible to control, and in all but the simplest cases is highly discouraged. Good practice is to write `get` and `set` methods that will read or write into class members in a controlled way. In other words, with `get` and `set` methods, it is possible to control the values of the data members. With public members, get and set functions are redundant, because it is possible to set the members directly, as in `$p2->x=5`. However, with the public members, it is not possible to control the value of the members. `Set` and `get` functions can be written directly, for each member, but PHP also provides so-called "magic methods" that can be used instead of having to write two functions for each member.

It is possible to protect the members much better, with the keywords `private` and `protected`. The exact meaning of these two keywords will be explained in the next section. It is also worth noting that `public` is the default visibility. If the visibility for a member or method is not specified, it defaults to `public`. Writing

```
class C {
$member;
       function method() {...}
….
}
```

is completely equivalent to writing :

```
 class C {
       public $member;
       pubic  function method() {…}
….
}
```

In contrast with the public class members, private class members or methods are only visible to the methods of the same class. Methods that are not connected to the class cannot access any of the private members, nor can they call any of the other private methods. If the keyword "public" is replaced with the keyword "private" for the class members `$x` and `$y`, and access is attempted, the result will be

```
PHP Fatal error:  Cannot access private property Point::$x in script2.1 on line 25
```

In other words, our neat little trick on line 25, which reads `$p2->x=5`, will no longer work. The constructor function has no problems whatsoever, and neither do functions `get_x()` and `get_y()`, as they are class members. This is a good thing, because it will no longer be possible to manipulate the class objects directly, potentially radically altering their behavior in a way that the class is not meant to do. In short, the class is more self-contained, like a controlled access highway – there are limited access and exit ramps.

Public and private members are now clear, but what are protected members and methods? Protected methods and members are accessible by the methods of the class they belong to, and by the methods of the classes that inherit from the base class they belong to. We will take a closer look at this in the next section.

Inheritance and Overloading

As stated in the beginning of this chapter, classes can be organized in a hierarchical way. The hierarchy is established through inheritance. In order to demonstrate inheritance, let us develop another class called `employee`. Some of a company's employees are managers, which will be a class that inherits from the more general `employee` class. Inheritance is also known as specialization. So, without further ado, let's see the class (see Listing 1-2).

Listing 1-2. *Example of* `employee` *Class*

```
<?php

class employee {
    protected $ename;
    protected $sal;
   function __construct($ename, $sal = 100) {
        $this->ename = $ename;
        $this->sal = $sal;
    }
```

```
    function give_raise($amount) {
        $this->sal+= $amount;
        printf("Employee %s got raise of %d dollars\n", $this->ename, $amount);
        printf("New salary is %d dollars\n", $this->sal);
    }
    function __destruct() {
        printf("Good bye, cruel world: EMPLOYEE:%s\n", $this->ename);
    }
}

class manager extends employee {
    protected $dept;
    function __construct($ename, $sal, $dept) {
        parent::__construct($ename, $sal);
        $this->dept = $dept;
    }
    function give_raise($amount) {
        parent::give_raise($amount);
        print "This employee is a manager\n";
    }
    function __destruct() {
        printf("Good bye, cruel world: MANAGER:%s\n", $this->ename);
        parent::__destruct();
    }
} // Class definition ends here.

$mgr = new manager("Smith", 300, 20);
$mgr->give_raise(50);
$emp = new employee("Johnson", 100);
$emp->give_raise(50);
?>
```

This class is just an artificial example; it is not meant to be used as a template. It is worth noticing that the `__construct` method is public in both classes. If it wasn't public, it wouldn't be possible to create new objects of either class. When executed, this script will produce the following result:

```
Employee Smith got raise of 50 dollars
New salary is 350 dollars
This employee is a manager
Employee Johnson got raise of 50 dollars
New salary is 150 dollars
Good bye, cruel world: EMPLOYEE:Johnson
Good bye, cruel world: MANAGER:Smith
Good bye, cruel world: EMPLOYEE:Smith
```

This little example is perfect for explaining the concept of inheritance. Every manager is an employee. Note the phrase "is a" characteristics for the inheritance relationship. In this case, class `employee` is the parent class for the class employee. Contrary to everyday life, a class in PHP can have only a single parent; multiple inheritance is not supported.

Furthermore, parent functions can be addressed using the `parent::` construct shown in the class `manager`. When an object of the child class is created, the constructor for the parent class is not called automatically; it is the responsibility of the programmer to call it within the constructor of the child class.

The same applies to the destructor method. The destructor method is the exact opposite of the constructor method. Constructor is called when an object is being established in memory, while the destructor is called when the object is no longer needed, or when the "unset" function is explicitly called on that object. Explicitly calling the unset function is not a common practice; it is usually used to save memory. This also means that the destructor is automatically called for all objects when the script execution is finished. Destructor methods are usually used to clean up resources, such as closing open files or disconnecting from a database. Finally, note that the destructor method of our `manager` class has complete access to the member `ename`, despite the fact that it is actually a member of the `employee` class. That is precisely the purpose of the protected members. If `ename` was a private member of the employee class, our little example would not have worked.

The method `get_raise` exists in both classes. PHP knows which method to call for which object; this is one aspect of a fundamental principle of OO: encapsulation. Object `$x` belongs to the `manager` class and the `give_raise` method generated the output, "This employee is a manager," after producing its normal output. We can rephrase this by saying that the `give_raise` function in the class `manager` overloads or supersedes the `give_raise` method in the `employee` class. Note that the meaning of the term "overload" in PHP is different from the meaning of the same term in C++ or Python, where it signifies a function (not a class method) with the same name but different argument types. Back to PHP: if the method is marked as `final`, it cannot be overloaded. If the method `give_raise` in the employee class was declared like this

```
final function give_raise($amount) {
….
}
```

overloading it in the class manager wouldn't be possible. We recommend that you try the basic OO concepts on this little script and play a bit by marking various members and methods private, protected, or public to see the results.

Finally, when speaking of inheritance, one also needs to mention abstract classes. Abstract classes cannot be instantiated; no objects belonging to them can be created. They're used primarily as templates, to force all classes that inherit from them to have the desired structure. The class is abstract if it is marked by the keyword "abstract," like this:

```
abstract class A {
….
}
```

No object of this class can be created; PHP will throw a runtime error and stop the script execution. It is also possible to declare abstract methods of an abstract class. This is done like this

```
abstract class A {
abstract protected method(...);
}
```

This is done to force classes that extend the abstract class to implement the specified method.

Abstract classes are usually used as templates for the classes that extend them. A good example of the abstract classes can be found in the Standard PHP Library (SPL). Classes for the sorted heap (`SplMinHeap`, `SplMaxHeap`) extend the abstract class `SplHeap` and implement the `compare` method differently. `SplMinHeap` will sort the elements from the smallest to the largest, while `SplMaxHeap` will sort them the other way around. Common characteristics of both classes are contained in the abstract class `SplHeap`, which is documented here:

```
http://ca2.php.net/manual/en/class.splheap.php
```

Rather than inventing an artificial example of abstract classes, let's see how they are used in SPL. Here is a brief example of how to use the `SplMinHeap` class

```
<?php
$heap = new SplMinHeap();
$heap->insert('Peter');
$heap->insert('Adam');
$heap->insert('Mladen');
foreach ($heap as $h) {
    print "$h\n";
}
?>
```

When executed, the output will be:

```
Adam
Mladen
Peter
```

Names are sorted in alphabetic order–not quite the way they were inserted into the heap. Later we will see how is it possible to use an object of the `SplMaxHeap` class in a loop, as if it was an array.

Now, let's turn our attention to more practical OO programming techniques. For example, you may have been wondering how we make classes available to PHP scripts. Classes are usually written to be re-used over and over again. The obvious answer is that we create separate files which we can then include with `require` or `include` directives, but that can soon get awkward or cumbersome as the files multiply. It turns out that PHP has a tool to help with precisely that problem–namely, the function called `__autoload`. That function takes a class name as an argument and will be called whenever PHP cannot find a class definition in the currently executing script. Essentially, the `__autoload` function is a trap handler for a "class not found" exception error. We will get back to the exceptions later. Our example in Listing 1-2 can now be rewritten in two files (see Listing 1-3).

Listing 1-3. *Listing 1-2 Rewritten in Two Files*
File script1.3.php:

```
<?php
function __autoload ($class) {
    require_once("ACME$class.php");
}

$x = new manager("Smith", 300, 20);
$x->give_raise(50);
$y = new employee("Johnson", 100);
$y->give_raise(50);
?>
```

File ACMEmanager.php:

```
<?php
class employee {
    protected $ename;
    protected $sal;
    // Note that constructor is always public. If it isn't, new objects cannot
```

```
    // be created.
    function __construct($ename, $sal = 100) {
        $this->ename = $ename;
        $this->sal = $sal;
    }
    function give_raise($amount) {
        $this->sal+= $amount;
        printf("Employee %s got raise of %d dollars\n", $this->ename, $amount);
        printf("New salary is %d dollars\n", $this->sal);
    }
    function __destruct() {
        printf("Good bye, cruel world: EMPLOYEE:%s\n", $this->ename);
    }
} // End of class "employee"

class manager extends employee {
    protected $dept;
    function __construct($ename, $sal, $dept) {
        parent::__construct($ename, $sal);
        $this->dept = $dept;
    }
    function give_raise($amount) {
        parent::give_raise($amount);
        print "This employee is a manager\n";
    }
    function __destruct() {
        printf("Good bye, cruel world: MANAGER:%s\n", $this->ename);
        parent::__destruct();
    }
} // End of class "manager"
```

This code is completely equivalent to the original script1.2.php in Listing 1-2, except it is much easier to read, because the most important part is contained in the file script1.3.php in Listing 1-3. The other file, ACMEmanager.php, only contains the class declarations. If we're not really interested in the internals of the class declarations, we don't have to read them; we only have to know how the objects of the declared classes behave. Also, note that the file is named after the first class that is being instantiated. When that file is loaded, the class `employee` will also be defined, because the definition of both classes is in the same file.

The second thing to note is that the class name was prefixed by "ACME." This is to alert the reader of the possibility of creating specialized, per-project class libraries. The function `__autoload` implemented here uses `require_once` instead of an `include` directive. The reason for that is the behavior of PHP, which will terminate the script if a file requested by the `require` directive is not available. The execution will proceed if the file simply included by an include directive is not available. Executing a script that depends on class definitions without those definitions being available doesn't make much sense.

Also, class definition files should not have a trailing `?>` defined. This is because they can often be autoloaded or included in a "header" file before the page is assembled, and any extra white space between the `?>` and EOF will be injected into the html output stream of the page at the start. PHP is quite happy not to have the trailing `?>`, omitting it is a best practice. This is probably the single biggest cause of "output already started" errors when using the `header()` function to send HTTP headers back to the browser, and for the unaware it is a death trap.