Chapter 7

Functions, Classes, and Objects

Learning Objectives

After completing this chapter, you will be able to:

- *Understand the functions*
- *Use the file inclusion statements*
- *Understand the concept of classes*
- *Use the methods and properties*
- *Understand the concept of objects*
- *Use the constructor and destructor*
- *Understand the concept of inheritance*
- *Use the parent and final keywords*
- *Use the instanceof operator*
- *Understand the concept of interface*
- *Understand the concept of anonymous class*

INTRODUCTION

All the programs that you have studied in the earlier chapters consist of a limited number of statements. Large programs, however, consist of many statements which make the program complex and difficult to understand. You can divide a large program into small groups of statements known as function to remove the program complexity. In case of Object-Oriented Programming (OOP), the data is treated as the most critical element of the program and the primary focus is on the data and not on the procedures. Therefore, in OOP, the programs are implemented using classes, methods, and objects. In this chapter, you will learn about functions, classes, objects, methods, inheritance, constructors, destructor, and so on.

FUNCTIONS

A function is a group of statements that perform a specific task and can be used multiple times in a program. A function has a name and it returns a value. It may also have a list of arguments and it gets executed when a call is made. If a function consists of an argument then a value can be passed to that argument at the time of function calling.

There are two types of functions in PHP which are as follows:

1. User Defined Functions
2. In-built Functions

These functions are discussed next.

User Defined Functions

PHP provides you more than a thousand in-built functions which can be directly used in a program as required. Other than these in-built functions, you can create user defined functions and name them.

Creating User Defined Functions

The syntax for creating a user defined function is as follows:

```
function Func_name($arguments){
   //statements
}
```

In the given syntax, **function** is the keyword used to create or declare function. The **Func_name** represents the name of function used as a reference in the program. The Parentheses (open and close bracket **( )**) after the function name are required. Here, **$arguments** represents the arguments or input parameters of the function which are passed inside the parentheses of the function. But, these arguments are optional. The function block starts with open curly bracket ({) and ends with a closing curly bracket (}) and the function statements are written inside the block of function.

For example:

```
<?php
   function cadcim(){
```

```
            echo "Welcome to CADCIM" ;
                    }
    cadcim(); //function calling
?>
// output: Welcome to CADCIM
```

In the given example, **function** is the keyword used to declare function. **cadcim()** is the function name with no input parameters passed inside the parentheses. When the function is called by its name, it displays the statement written inside the function block in the browser.

You can also pass input parameters to the function if required in the program.

For example:

```
<?php
    function bookname($book){
          echo "$book" ;
                    }
    bookname("Introducing PHP/MySQl");
?>
//output: Introducing PHP/MySQl
```

In the given example, **function** is the keyword used to declare function in PHP, **bookname** is the function name, and **$book** is the input parameter passed to the function **bookname**. When some value is passed to the function, it will get assigned to the input parameter **$book** of the function and output will be displayed using the **echo** statement. In this case, **Introducing PHP/MySQL** is the value passed to the function **bookname()** so it will get assigned to **$book** and will be displayed as output.

Function Name

The function name is used as reference in the program. It can be used multiple times to call the same function if required in the program.

While naming a function, you must follow certain rules. The rules are as follows:

1. Only alphabetic characters, both uppercase and lowercase, digits from 0 to 9, and the underscore (_) can be used.
2. Function name can start with an alphabet or an underscore but not with a digit.
3. Function names are case insensitive. For example, case, CASE, and Case, all three will be treated as name of a single function.

The following function names are invalid in PHP:

```
9_Count() // Function name cannot start with a digit.
COUNT#() // Function name cannot contain #(hash symbol).
```

The following function names are valid in PHP:

```
COUNT_9() // Function name can start with alphabetic characters.
_ACCOUNT() // Function name can start with an underscore.
Account() // Function name is case insensitive.
```

Tip
It is recommended to name the function according to its functionality rather than random names. It will make your code more clear.

Function Arguments

The function arguments are used to pass information to the function. You can pass one or more arguments in a single function according to the requirement of the program. The arguments or input parameters of function are passed inside the parentheses just after the function name. The function arguments work like a variable inside the function.

For example:

```
<?php
   function family($fname, $age, $hobby) {
         echo "$fname is $age years old.<br>";
         echo "He loves playing $hobby.<br>";
   }
         family("John", 40, "football");//function call
?>
```

In the given example, **family($fname, $age, $hobby)** is a function created by using the **function** keyword. Here, **family** is the function name and **$fname, $age,** and **$hobby** are multiple arguments passed to the function and are separated by comma. At the time of function call, some values are passed to the function which gets assigned to the input parameters **$fname**, **$age**, and **$hobby** of the function.

Note
A function can be called multiple times by passing different values to it. If you call a function without passing any value then it takes the default value.

Example 1

The following program will illustrate the use of the function by passing multiple arguments to it. In this program, function is called multiple times and the output is displayed in the browser.

```
<!Doctype html>                                                    1
<html>                                                             2
<head>                                                             3
<title>Function with multiple arguments</title>                    4
</head>                                                            5
<body>                                                             6
<?php                                                              7
```

```
function player($fname="Smith", $age=16, $team="Not eligible") 8
{                                                              9
   if($age<18)                                                 10
    {                                                          11
      echo "$fname you are $age years old.<br>";             12
      echo "Your age is below 18. You are $team to play <br>"; 13
      echo "<br>";                                           14
    }                                                          15
   elseif($age==18)                                            16
    {                                                          17
      echo "$fname your age is $age.<br>";                   18
      echo "You will play in team $team.<br>";               19
      echo "<br>";                                           20
    }                                                          21
elseif($age==19)                                               22
   {                                                           23
     echo "$fname your age is $age.<br>";                      24
     echo "You will play in team $team.<br>";                  25
     echo "<br>";                                              26
   }                                                           27
else                                                           28
   {                                                           29
    echo "$fname players above the age 19 will play in team    30
    $team<br>";                                                31
    echo "<br>";                                               32
   }                                                           33
}                                                              34
player("John", 18, "A");                                       35
player("Justin", 19, "B");                                     36
player("All", 40, "C");                                        37
player();                                                      38
?>                                                             39
</body>                                                        40
</html>                                                        41
```

Explanation

Line 8

function player($fname="Smith", $age=16, $team="Not eligible")

In this line, **function** is the keyword used to create a function. **player** is the function name and **($fname="Smith", $age=16, $team="Not eligible")** are multiple arguments separated by comma, passed to the function inside the parentheses. Here, each argument in the function is assigned a default value.

Line 35

player("John", 18, "A");

In this line, **Player** is the function name and **John**, **18,** and **A** are the values passed to the function. Here, this line will call the function and the values passed here will get assigned to

the arguments of the function **Player**, declared in Line 8. Now, this line will display the current assigned values instead of the default values in the output.

The working of Line 36 and Line 37 is similar to Line 35.

Line 38
player();
In this line, **player()** is the function name by which function is called. In this function, no value is passed inside the parentheses. Therefore, it will call the function and display the default values assigned to the arguments of the function **player** in Line 8.

The output of Example 1 is displayed in Figure 7-1.

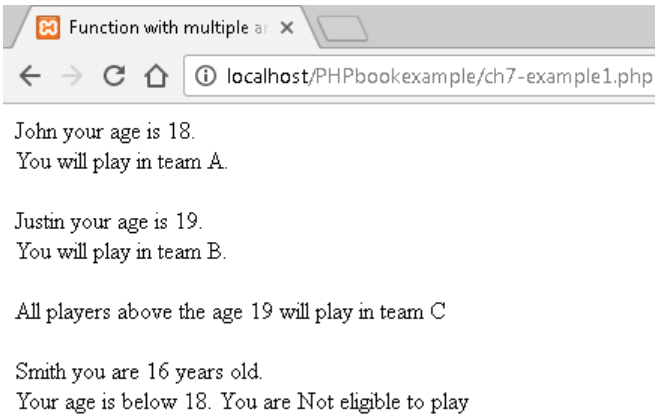

Figure 7-1 *The output of Example 1*

Function Return Value

You have already learned about **return** statement in the previous chapter. The **return** statement is used to return a function value.

For example:

```
<?php
      function mul($p, $q) {
      $r = $p * $q;
      return $r;
      }
      echo "5 * 10 = " . mul(5,10) . "<br>";
      echo "7 * 13 = " . mul(7,13) . "<br>";
?>
```

In the given example, **function** is the keyword used to create a function and **mul** is the function name. Here, **$p** and **$q** are the two arguments passed to the function. In this function, multiplication operation is performed between **$p** and **$q** variables and the resultant value gets assigned to the **$r** variable. Next, the **return** statement will return the resultant value which is assigned to the **$r** variable at the time of function call.

Dynamic Function Call

In PHP, it is possible to dynamically call a function by assigning the function name as a string to the variables. These variables will behave exactly same as the function assigned to them. Therefore, to call a function dynamically instead of the function name, you can use the variable name with parentheses.

For example:

```
<?php
      function dynamicFunc() {
         echo "It is a string <br>";
      }
      $func_var = "dynamicFunc";
      $func_var();
   ?>
```

In this example, **dynamicFunc()** is the function which consist of an **echo** statement inside its block. Here, **$func_var** is the variable which holds **dynamicFunc** (function name) as string type value. Now, **$func_var** variable will behave exactly same as the **dynamicFunc()** function. Therefore, you can use **$func_var** variable as a **$func_var()** function to dynamically call the **dynamicFunc()** function.

In-built Functions

There are many in-built functions in PHP. These in-built functions are pre-defined and can be used directly in the program. You have already learned some in-built string functions in the previous chapters. In this section, you will learn more about the in-built PHP functions which are discussed next.

date()

The PHP **date()** function is used to display current date and time. The syntax for the **date()** function is as follows:

```
date(format,timestamp);
```

In the given syntax, **date()** is the in-built function name for date. Inside the parentheses of date function, **format** represents different formats for date and **timestamp** is an optional parameter. The **date()** function displays the current time, if **timestamp** is not specified.

For example:

```
<?php
   echo "Current Date in Format 1 is" . date("Y/m/d") . "<br>";
   echo "Current Date in Format 1 is" . date("Y.m.d") . "<br>";
   echo "Current Date in Format 1 is" . date("Y-m-d") . "<br>";
   echo "Current day is" . date("l") . "<br>";
   echo "Current time is" . date("h:i:sa");
?>
```

In the given example, **date()** is the function used to display current date. Here, **y/m/d**, **y.m.d**, and **y-m-d** are the 3 different date formats defined inside the parentheses of **date()** function, where **y** represents the year (in 4 digit), **m** represents the month (from 01 to 12), **d** represents the date (from 01 to 31), and **l** represents the day of week. The **h:i:sa** is the time format to display current time, where **h** represents the 12 hour format of an hour, **i** represents minutes, **s** represents the seconds and **a** represents the ante meridiem (am) or post meridiem (pm).

var_dump()

The **var_dump()** function is used to display the variable related information including the data type of the value assigned to the variable. The syntax of **var_dump()** is as follows:

```
var_dump($var);
```

In the given syntax, **var_dump()** is the function used to get the details of a variable and **$var** can be any variable whose detail is to be displayed in the browser.

Example 2

The following program will illustrate the use of the **var_dump()** function. In this program, different values will be assigned to the variables and the output is displayed in the browser.

```
<!Doctype html>                                                    1
<html>                                                             2
<head>                                                             3
<title>Var_dump function</title>                                   4
</head>                                                            5
<body>                                                             6
<?php                                                              7
   $value1=508;                                                    8
   $value2="Hello";                                                9
   $value3=698.99;                                                 10
   $value4=true;                                                   11
   $value5="num123";                                               12
   echo var_dump($value1)."<br>";                                  13
   echo var_dump($value2)."<br>";                                  14
   echo var_dump($value3)."<br>";                                  15
   echo var_dump($value4)."<br>";                                  16
   echo var_dump($value5)."<br>";                                  17
?>                                                                 18
</body>                                                            19
</html>                                                            20
```

Explanation

Line 13

**echo var_dump($value1)."
";**

In this line, **var_dump()** is the function used to display variable related information. Here, the

var_dump() function will display the data type and value of the **$value1** variable. This line will display the following in the browser:

int(508)

The working of lines 14 to 17 is similar to Line 13.

The output of Example 2 is displayed in Figure 7-2.

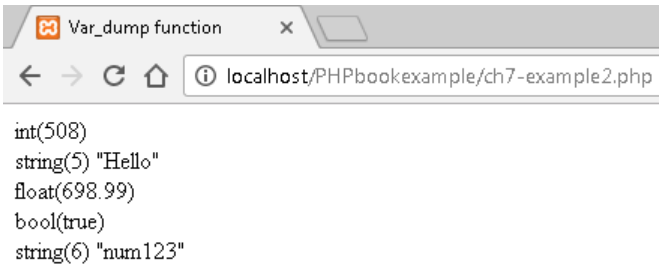

Figure 7-2 *The output of Example 2*

CSPRNG FUNCTIONS

The CSPRNG functions are introduced in PHP 7. Here, CSPRNG stands for Cryptographically Secure Pseudo-Random Number Generator. It consists of following two functions:

a. **random_bytes()**
b. **random_int()**

They are used to generate cryptographically secure integers and strings in a cross platform way. These CSPRNG functions are discussed next.

random_bytes()

The **random_bytes()** function is used to generate cryptographically secure pseudo-random bytes in PHP. It means this function is used to convert the information provided by the coder or user into randomly generated bytes to secure the information or data. The **random_bytes()** function is used to generate keys or initialization vectors. The syntax of the **random_bytes()** function is as follows:

```
string random_bytes ( int $length )
```

In the given syntax, **random_bytes()** is an in-built **string** type function used to generate random string. Inside the parentheses of this function, integer type variable **$length** is passed. This function will convert the integer value assigned to the **$length** variable into random string and return the requested number of the random string.

Example 3

The following program will illustrate the use of **random_bytes()** and **bin2hex()** functions. In this program, random string will be generated and converted into hexadecimal form and the output is displayed in the browser.

```
<!Doctype html>                                                    1
<html>                                                             2
<head>                                                             3
<title>Random bytes</title>                                        4
</head>                                                            5
<body>                                                             6
<?php                                                              7
     $string= random_bytes(11);                                    8
     print("Random string generated is ");                         9
     print ($string);                                              10
     print("<br>Binary to hex conversion value for this           11
               randomly generated string is ");
     print(bin2hex($string));                                      12
?>                                                                 13
</body>                                                            14
</html>                                                            15
```

Explanation

Line 8

$string= random_bytes(11);

In this line, **random_bytes()** is the function assigned to the **$string** variable. Inside the parentheses of function, 11 is the integer type value. This value will get converted into random string using the **random_bytes()** function.

Line 12

print(bin2hex($string));

In this line, **print** is the statement used to display the output in the browser. Inside the parentheses of **print** statement, **bin2hex()** function is passed. This function is used to convert binary data into hexadecimal data. Here, **$string** is the variable passed inside the parentheses of **bin2hex()** function. The value of **$string** variable will convert into hexadecimal form.

The output of Example 3 is displayed in Figure 7-3.

Figure 7-3 *The output of Example 3*

random_int()

The **random_int()** function is used to generate cryptographically secure pseudo-random integers in PHP. It means this function is used to convert the information provided by the coder or user into randomly generated integers to secure the information or data. The **random_int()** function is used where unbiased results are critical. The syntax of **random_int()** function is as follows:

```
int random_int ( int $min , int $max )
```

In the given syntax, **random_int()** is an in-built **int** type function used to generate random integers. Inside the parentheses of this function, integer type variables **$min** and **$max** are passed. Here, **$min** variable will hold minimum value and **$max** variable will hold maximum value. This function will return the random integer value in between the **$min** and **$max** values.

Example 4

The following program will illustrate the use of **random_int()** function. In this program, random integers will be generated and the output is displayed in the browser.

```
<!Doctype html>                                                    1
<html>                                                             2
<head>                                                             3
<title>Random integer</title>                                      4
</head>                                                            5
<body>                                                             6
<?php                                                              7
     $min_value= -1000;                                            8
     $max_value= 100;                                              9
     print("Random generated first integer is ");                  10
     print(random_int(1000 , 3000));                               11
     print("<br>Random generated second integer is ");             12
     print(random_int($min_value , $max_value));                   13
?>                                                                 14
</body>                                                            15
</html>                                                            16
```

Explanation

Line 11

print (random_int(1000 , 3000));

In this line, **random_int()** is the function. Inside the parentheses of function, 1000 and 3000 are the minimum and maximum integer type values passed, respectively. The **random_int()** function will generate the random integer value whose range will be between 1000 to 3000.

The working of Line 13 is similar to Line 11.

The output of Example 4 is displayed in Figure 7-4.

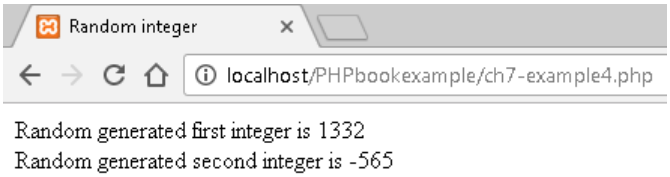

Figure 7-4 *The output of Example 4*

FILE INCLUSION STATEMENTS

File inclusion statements are used to include any functions, HTML code, or PHP code saved in another file in the current PHP file. There are two types of file inclusion statements as follows:

a. **include** statement
b. **require** statement

The **include** and **require** statements are discussed next.

The include Statement

The **include** statement is used when the current PHP file requires the content of another PHP file in it. The advantage of the **include** statement is that it loads the total content of a given file in the current file in just one line of code. There is no need to copy paste the entire code in the current file. The disadvantage of this statement is that if it does not find the included file which is essential for the specified task, it will still execute the program. The syntax of **include** statement is as follows:

```
include 'file_name.extension';
```

In the given syntax, **include** is the keyword used to include a required file. The **file_name.extension** is the required file which is to be included in the current file.

For example:

```
<?php
        include 'example.php';
        // PHP code
?>
```

In the given example, **include** is the keyword used to include **example.php** file in the current program.

The require Statement

The working of the **require** statement is same as of the **include** statement with the only difference that if the required file is missing, it will not execute the program. So if it is important to include

a file in the program, it is better to use the **require** statement in place of the **include** statement. The syntax of the **require** statement is as follows:

```
require 'file_name.extension';
```

In the given syntax, **require** is the keyword used to include a required file. The **file_name.extension** is the required file which is to be included in the current file.

Example 5

The following program will illustrate the use of the **include** and **require** statements. In this program, two files are included and the output is displayed in the browser.

The *header.php* file for inclusion.

```
<?php                                                    1
    echo "<p><b>Hello User</b></p>"                      2
?>                                                       3
```

The *footer.php* file for inclusion.

```
<?php                                                    1
    echo "<p>Copyright &copy;".date("Y")." CADCIM</p>"   2
?>                                                       3
```

The *ch7-example5.php* program file will include the two files: *header.php* and *footer.php*.

```
<!Doctype html>                                                    1
<html>                                                             2
<head>                                                             3
<title>File inclusion</title>                                      4
</head>                                                            5
<body>                                                             6
<?php                                                              7
     include 'header.php';                                         8
         echo "This is the PHP code";                            9
         echo "<br> It includes 2 external files as follows:";   10
         echo "<br>1. header.php";                               11
         echo "<br>2. footer.php";                               12
     require 'footer.php';                                        13
?>                                                                 14
</body>                                                            15
</html>                                                            16
```

Explanation (Footer.php)

Line 2

echo " Copyright ©".date("Y"). " Cadcim ";

In this line, **©** is the HTML keyword to draw the copyright symbol in the browser, **date()**

is the function used to display the current date, inside the parentheses of function, **"Y"** displays the current year.

Explanation (ch7-example5.php)

Line 8
include 'header.php';
In this line, **include** is the keyword used to load all the content of the given file in the current file. Here, **header.php** file is the required file which is included.

Line 13
require 'footer.php';
In this line, **require** is the keyword used to load all the content of required file in the current file. Here, **footer.php** file is the required file which is included.

The output of Example 5 is displayed in Figure 7-5.

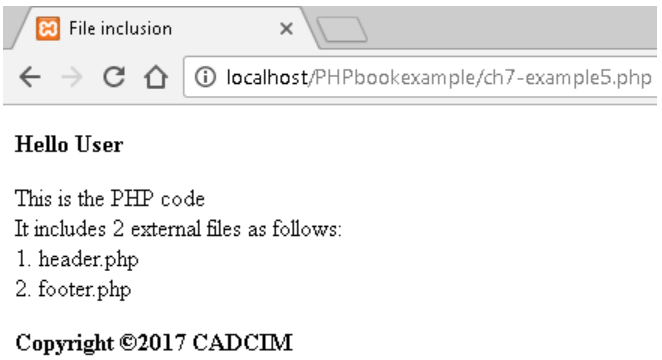

Figure 7-5 *The output of Example 5*

CLASSES

A class is an user-defined data type and is a collection of data and methods (functions). It acts as a blueprint or prototype in the creation of objects. These objects are known as the instances of a class having specific data type. The data in a class specifies the nature of a class, whereas the methods(functions) are used to operate on the data inside a class. Both the data and methods are known as the members of a class. The motive behind using a class is to encapsulate the data and methods into a single unit so that the data members can be accessed only through a well-defined interface. This process is known as data hiding.

Declaring a Class

A class is declared by using the **class** keyword with the class name. The class name should be a valid identifier. The class definition consists of data members and methods. The syntax for declaring a class is as follows:

```
class class_name
{
   // property(variable) declaration
         ----------;
         ----------;
```

```
    //method(function) declaration;
    {
          //body of method
          ----------;
          ----------;
    }
}
```

In the given syntax, the declaration of the class begins with the **class** keyword followed by a **class_name** which is an identifier given by the programmer to specify the name of the class. The body of the class is enclosed within the curly braces {}.

The functions used in a class are known as methods and the variables used in a class are known as properties.

Property and Method Scope

The property and method scope specifies the visibility of the properties or the methods of a class. There are three keywords used to specify the visibility of method and function which are:

a. **public**
b. **protected**
c. **private**

You can use any of the three keywords to specify the visibility of the properties and methods. These keywords are discussed next.

public

The **public** methods or properties of a class can be accessed by the members of the same class and by the members of the other classes. So it means the method or the property declared as **public** can be accessed from anywhere in the program.

The syntax for declaring **public** property is as follows:

```
public $variable = 'string';
```

In the given syntax, the **$variable** is the property which is declared as public by prefixing it with the **public** keyword. This specifies that the **$variable** property can be accessed from anywhere in the program.

The syntax for declaring **public** method is as follows:

```
public function MyFunc() { }
```

In the given syntax, the **MyFunc()** is the method which is declared as public by prefixing it with the **public** keyword. This specifies that the **MyFunc()** method can be accessed from anywhere in the program.

protected

The **protected** methods or the properties of a class can be accessed by members of the same class and by the inheriting classes.

The syntax for declaring the **protected** property is as follows:

```
protected $variable = 'string';
```

In the given syntax, **$variable** is the property which is declared as protected by prefixing it with the **protected** keyword. This specifies that the **$variable** property can be accessed from the same class or by the inheriting classes in the program.

The syntax for declaring the **protected** method is as follows:

```
protected function MyFunc() { }
```

In the given syntax, **MyFunc()** is the method which is declared as protected by prefixing it with the **protected** keyword. This specifies that the **MyFunc()** method can be accessed from the same class or by the inheriting classes in the program.

Note

You will learn about inheritance later in this chapter.

private

The **private** methods or the properties of a class can only be accessed by the members of the same class.

The syntax for declaring the **private** property is as follows:

```
private $variable = 'string';
```

In the given syntax, **$variable** is the property which is declared as private by prefixing it with **private** keyword. This specifies that the **$variable** property can be accessed only by the members of the same class.

The syntax for declaring **public** method is as follows:

```
public function MyFunc() { }
```

In the given syntax, **MyFunc()** is the method which is declared as private by prefixing it with the **private** keyword. This specifies that the **MyFunc()** method can be accessed only by the members of the same class.

OBJECTS

In object-oriented programming, a problem is divided into certain basic entities called objects. In this type of programming, all communication is carried out between the objects. When a

program is executed, the objects interact with each other by sending messages. The objects contain the data and the functions that can be used to manipulate data.

An object is defined as an instance or a physical instantiation of a class. It is also known as a living entity within a program. When an object is created within a class, it maintains its own copy of instance variables that are defined inside the class. A class provides certain attributes and each object can have different values for those attributes. Therefore, each object of a class is uniquely identified.

Creating an Object

After the declaration of a class, you can create the objects of that class by using the class name with the **new** keyword. The syntax of creating an object is as follows:

```
$obj = new class_name;
```

In the given syntax, **class_name** represents the name of the class. The object is created by assigning the **class_name** prefixed with the **new** keyword to the **$obj** variable. After the assignment of **new class_name**, the variable **$obj** acts as the object of the specified class. So here, **$obj** represents an object of **class_name** class.

Accessing Members Using Objects

The **objects** are used to access the members of a class using arrow operator (**->**). The syntax for accessing a member using **objects** is as follows:

```
$obj->member;
```

In the given syntax, **$obj** is representing an object. The arrow operator (**->**) is used with the object to access the member. Here, **member** can be a property or a method of the class.

The syntax to access the property of the class is as follows:

```
$obj->prop; //line 1
$obj->prop = "value"; //line 2
```

There are two syntax to access the property of a class. In the line 1, **$obj** represents the object and **prop** represents the property name without **$** (dollar) sign. This syntax is used if a value is already assigned to the property. In the other case, the second syntax is used which is line 2 where value is assigned while accessing the property.

Example 6

The following program will illustrate the use of **class** and **object**. In this program, the values of the properties are accessed using the objects of a class and the output is displayed in the browser.

```
<!Doctype html>                                                    1
<html>                                                             2
<head>                                                             3
```

```
<title>Accessing property</title>                                    4
</head>                                                              5
<body>                                                               6
<?php                                                                7
    class prop                                                       8
   {                                                                 9
     public $property = "The value of \$property is accessed        10
                        by using \$var object";
     public $property2;                                              11
   }                                                                 12
   $var = new prop;                                                  13
   $var2 = new prop;                                                 14
   echo $var->property;                                              15
   echo "<br>";                                                      16
   echo $var2->property2 = "The value of \$property2 is             17
                           by accessed using \$var2 object";
?>                                                                   18
</body>                                                              19
</html>                                                              20
```

Explanation

Line 8

class prop

In this line, **class** is the keyword and **prop** is the name of the class.

Line 9

{

This line indicates the start of the body of the **prop** class.

Line 10

public $property = "The value of \$property is accessed by using \$var object";

In this line, **$property** is the property which is declared as public by prefixing it with **public** keyword. This specifies that the **$property** property can be accessed from any part of the program. The **$property** property holds a string type value.

Line 11

public $property2;

In this line, **$property2** is the property which is declared as public by prefixing it with **public** keyword. This specifies that the **$property2** property can be accessed from any part of the program. Here, no value is assigned to this property.

Line 12

}

This line indicates the end of the body of the **prop** class.

Line 13 and Line 14

$var = new prop;

$var2 = new prop;
In these lines, the **$var** and **$var2** are the objects of the **prop** class created by using the **new** keyword.

Line 15
echo $var->property;
In this line, **property** is the property name without the **$** sign. This property is accessed by using the **$var** object with the arrow operator (**->**).

This line will display the following in the browser:

The value of $property is accessed by using $var object

Line 17
echo $var2->property2 = "The value of \$property2 is accessed by using \$var2 object";
In this line, **property2** is the property name without the **$** sign. This property is accessed by using the **$var2** object with the arrow operator (**->**). Here, the value is assigned to the **$property2** property at the time of accessing the property.

This line will display the following in the browser:

The value of $property2 is accessed by using $var2 object

The output of Example 6 is displayed in Figure 7-6.

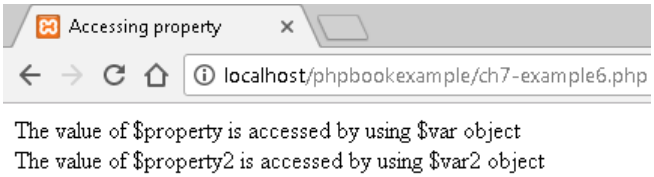

Figure 7-6 *The output of Example 6*

The syntax to access the method of the class is as follows:

```
$obj->method();
```

In the given syntax, **$obj** represents the object and **method()** represents the method. The arrow operator (**->**) is used with the object to access the method.

For example:

```
<?php
        class world
          {
            function user()
                 {
                   echo "Hello User!";
                 }
```

```
        }
    $var = new world;
    $var->user();
?>
```

In the given example, **$var** is the object and **user()** is the method. The **$var** object is used with the arrow operator (**->**) to access the **user()** method.

OBJECT CLONING

Every object is uniquely identified in PHP. But if we create a copy of an object then its properties also get completely copied by its reference. For example, if object1 is copied to object2 then they get linked. Whenever there will be change in the value of any linked object that is either object1 or object2, the same result will be displayed in the other. Therefore through copying object the same output can get generated for multiple objects. This can be avoided by cloning an object instead of copying an object.

Cloning an object creates copy of an object without referencing the main object. This means that any change in the value of one object will not affect the value of another object. The clone of an object is created by using the **clone** keyword.

The syntax of an object cloning is as follows:

```
$CopyOfObject = clone $object;
```

In the given syntax, **$object** is the main object which will get copied to **$CopyOfObject** by using **clone** keyword. Here, **$CopyOfObject** represents another object.

Example 7

The following program will illustrate object cloning in PHP. In this program, an object is copied directly and by using the **clone** keyword and the output is displayed in the browser.

```
<!Doctype html>                                                    1
<html>                                                             2
<head>                                                             3
<title>Object cloning</title>                                      4
</head>                                                            5
<body>                                                             6
<?php                                                              7
    class object_cloning                                           8
   {                                                               9
         public $gesture;                                          10
   }                                                               11
   $var = new object_cloning;                                      12
   $var->gesture = "Hello main object";                            13
   $var2 = $var; //copy of object                                  14
   $var2->gesture = "Hello copy object";                           15
```

```
    $var3 = clone $var; //clone of object                          16
    $var3->gesture = "Hello clone object";                         17
    echo "orignal value of main-object is <b>Hello main            18
          object</b>.<br>";
    echo "orignal value of copy-object is <b>Hello copy            19
          object</b>. <br>";
    echo "orignal value of clone-object is <b>Hello clone          20
          object</b>. <br>";
    echo "<b> Values:</b> <br>";                                   21
    echo "main-object = <b>" .$var->gesture."</b><br>";            22
    echo "copy-object = <b>" .$var2->gesture."</b><br>";           23
    echo "clone-object = <b>" .$var3->gesture."</b><br>";          24
?>                                                                 25
</body>                                                            26
</html>                                                            27
```

Explanation

Line 11

public $gesture;

In this line, **$gesture** is the property which is declared as public by prefixing it with **public** keyword. This specifies that the **$gesture** property can be accessed from any part of the program. Here, no value is assigned to this property.

Line 12

$var = new object_cloning;

In this line, **$var** is an object of class **object_cloning** created by using the **new** keyword.

Line 13

$var->gesture = "Hello main object";

In this line, **gesture** is the property name without the **$** sign. This property is accessed by using the **$var** object with the arrow operator (**->**). Here, the value is assigned to the **$gesture** property at the time of accessing the property.

Line 14

$var2 = $var;

In this line, **$var2** is an object which will copy **$var** object by its reference.

Line 15

$var2->gesture = "Hello copy object";

In this line, **gesture** is the property name without the **$** sign. This property is accessed by using the **$var2** object with the arrow operator (**->**). Here, the value is assigned to the **$gesture** property at the time of accessing the property.

Line 16

$var3 = clone $var;

In this line, **$var3** is an object which will copy **$var** object by using **clone** keyword.

Line 17
$var3->gesture = "Hello clone object";
In this line, **gesture** is the property name without the **$** sign. This property is accessed by using the **$var3** object with the arrow operator (**->**). Here, the value is assigned to the **$gesture** property at the time of accessing the property.

The output of Example 7 is displayed in Figure 7-7.

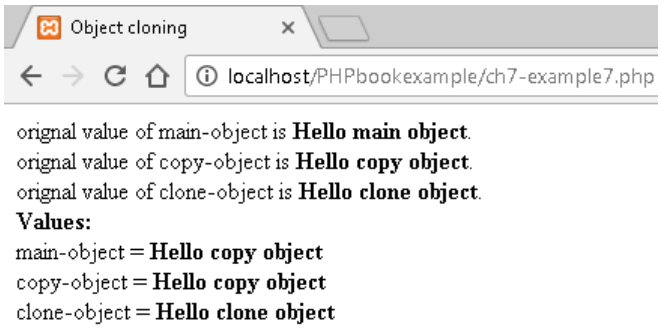

Figure 7-7 *The output of Example 7*

CONSTRUCTOR

A constructor is a special method (function) used to initialize the objects of a class. You can pass multiple parameters in the constructor while creating a new object. The **__construct()** method is the in-built constructor method of a class. The syntax of constructor is as follows:

```
class Class_name {
   function __construct() {
                            //body of constructor method
                           }
               }
```

In the given syntax, **Class_name** represents the name of class, **function** is the keyword used to create method, and **__construct()** is the constructor method of the **class_name** class.

Note
In the past, the name of constructor was the same as the name of class. This old format is deprecated in PHP 7. Now, the standard name of constructor is ***__construct()****.*

DESTRUCTOR

A destructor is also a special method used to destroy the objects that have been initialized by a constructor and are no longer required. The **__destruct()** method is the in-built destructor method of a class. The syntax of destructor is as follows:

```
class Class_name {
   function __construct()
              {
                    //body of constructor method
```

```
            }
    function __destruct()
            {
                  //body of destruct method
            }
        }
```

In the given syntax, **Class_name** represents the name of class and **function** is the keyword used to create method. Here, **__construct()** is the constructor method and **__destruct()** is the destructor method of the **class_name** class.

Example 8

The following program will illustrate the use of constructor and destructor. The program will create and destroy two objects of a class and the output is displayed in the browser.

```
<!Doctype html>                                                    1
<html>                                                             2
<head>                                                             3
<title>Constructor and Destructor</title>                          4
</head>                                                            5
<body>                                                             6
<?php                                                              7
    class objects                                                  8
         {                                                         9
               public $val = "Object";                             10
               public function __construct($val)                   11
               {                                                   12
                     echo "Object is created<br>";                13
                     $this->val = $val;                            14
               }                                                   15
                     public function __destruct()                  16
               {                                                   17
                     echo "Object is destroyed<br>";               18
               }                                                   19
         }                                                         20
   $new_val = new objects("Object");                               21
   echo $new_val->val. " is alive <br>";                           22
?>                                                                 23
</body>                                                            24
</html>                                                            25
```

Explanation

Line 11

public $val = "Object";

In this line, **Object** is a string type value assigned to the **$val** property (variable) and the property is declared as public by using the **public** keyword.

Line 12
public function __construct($val)
In this line, **function** is the keyword used to create method and **__construct($val)** is the parameterized constructor method of the **objects** class. Here, **$val** is the parameter passed inside the parentheses of constructor **__construct()**. It will initialize the object of **objects** class.

Line 14
$this->val = $val;
In this line, **$this** is the pre-defined variable used to access the current object. The **val** is the property name without the **$** sign and **$val** property is assigned to it. After the execution of this line, the control will be passed to Line 21 and Line 22.

Line 16
public function __destruct()
In this line, **function** is the keyword used to create method and **__destruct()** is the destructor method of the **objects** class which will destroy the object. It will be executed in the end when object initialized by a constructor is no longer required.

Line 21
$new_val = new objects("Object");
In this line, the **$new_val** is an object of class **objects** created by using the **new** keyword. Here, **Object** is the value passed to the object created.

Line 22
**echo $new_val->val. " is alive
" ;**
In this line, **val** is the property name without the **$** sign. The **$val** property is accessed by using the **$new_val** object with the arrow operator (**->**).

This line will display the following in the browser:

Object is alive

The output of Example 8 is displayed in Figure 7-8.

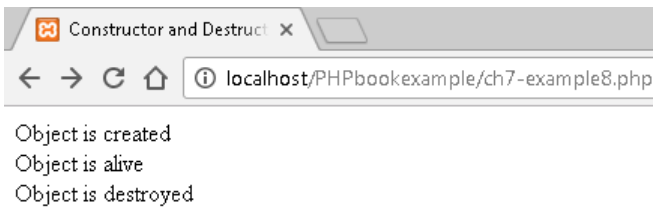

Figure 7-8 *The output of Example 8*

CLASS CONSTANT

The working of class constant is same as the constant which you learned in the previous chapter with the only difference that the constant in the class is declared using the **const** keyword instead of **define()** function. The scope of class constant is limited to the class in which it is declared.

The syntax of class constant is as follows:

```
class Class_name
{
    const NAME = 'value';
}
```

In the given syntax, **const** is the keyword used to define constant inside the class, **NAME** represents the constant name, and **value** is the value of the constant. The value of constant can be an integer, boolean, string, float, NULL, or an array type.

The name of the class followed by double colon(**::**) also known as scope resolution operator along with the name of the class constant is used to return the class constant value outside the class.

For example:

```
<?php
   class Const_class
         {
               const HELLO = 'Hello User';
         }
   echo Const_class::HELLO;
?>
```

In the given example, **Const_class** is the class name. Inside the class, a constant is defined by using the **const** keyword along with constant name **HELLO**. The value assigned to constant is **Hello User**. The **echo** statement is used to display the constant value where **Const_class::HELLO** is used to return the value of constant.

To return the value of constant inside the class, the keyword **self** is used followed by scope resolution(::) operator and class constant name. For example, **self::const_name;** where **const_name** is the class constant name.

STATIC METHODS AND PROPERTIES

Declaring a method or property of a class as static makes them accessible directly without creating or instantiating an object of that class. You can create static methods and properties by using the **static** keyword. By default, the visibility of static methods and properties is public. The static methods and properties are linked to the class in which it is declared. The value of static property can be changed multiple times in a program. But in a constant property, the value remains fixed throughout the program if once defined for a particular property.

The syntax of static property is as follows:

```
public static $static_property;
```

In the given syntax, **public** is the keyword which is used to define the visibility of the property to be public. **$static_property** is the property which is declared as static property by using the **static** keyword.

The name of the class followed by the scope resolution operator(**::**) along with the name of the property is used to call the static property outside the class.

For example:

```
class prop
{
   public static $var = 5; //Static variable or property
}
echo prop::$var;
```

In the given example, **public** is the keyword used to define the visibility of the property to be public. **$var** is the property which is declared as static property by using the **static** keyword inside the class **prop** holding integer type value 5. The **prop::$var** is used to call the static property where **prop** is the class name followed by scope resolution operator(**::**) along with **$var** static property.

To call the value of static property inside the class, the keyword **self** is used followed by scope resolution operator(**::)** and static property name. For example, **self::$var;** where **$var** is the static property.

The syntax of static method is as follows:

```
public static function static_method()
         {
                // body of method or function
         }
```

In the given syntax, **public** is the keyword used to define the visibility of the method to be public. The **static_method()** is the method which is declared as static method by using the **static** keyword.

The name of the class followed by the scope resolution operator(**::**) along with the name of the method is used to call the static method outside the class.

For example:

```
<?php
class func
{
   static function test($var1 , $var2)
   {
         echo "$var1  $var2";
   }
}
func::test("Hello" ,  "User");
?>
```

In the given example, **function** is the keyword used to create method in the class. The **test($var1, $var2)** is the parameterized method which is declared as static method by using the **static** keyword. Here, **$var1** and **$var2** are the two parameters passed inside the static method. By default, the visibility of the method is public as it is not declared in the example. The **func::test("Hello" , "User");** is used to call the static method where **func** is the class name followed by scope resolution operator(**::**) along with static method **test()**. Here, the two values **Hello** and **User** are passed to the method.

To call the value of static method inside the class, the keyword **self** is used followed by scope resolution operator(**::**) and static method name. For example, **self::test();** where **test()** is the static method.

Example 9

The following program will define class constant, static methods, and static properties. It will also call them inside as well as outside the class and the output is displayed in the browser.

```
<!Doctype html>                                                    1
<html>                                                             2
<head>                                                             3
<title>Class constant and static</title>                           4
</head>                                                            5
<body>                                                             6
<?php                                                              7
    class const_static                                             8
   {                                                               9
              const FIRST_CONST = "Hello User";                    10
              const SECOND_CONST = 10;                             11
              static $var = 5;                                     12
              static $var2 = "The output is displayed";            13
        public static function test($var1 , $var2)                14
         {                                                         15
              echo "$var1  $var2 <br>";                          16
         }                                                         17
        static function test2()                                    18
         {                                                         19
              echo "Welcome to CADCIM Technologies <br>";        20
         }                                                         21
        function showConstant()                                    22
         {                                                         23
              echo self::FIRST_CONST . "<br>";                     24
              self::test2();                                       25
              echo self::$var2 . "<br>";                           26
         }                                                         27
   }                                                               28
   $class = new const_static;                                      29
   $class->showConstant();                                         30
   const_static::test("The integer" , "values are:");              31
```

```
    echo const_static::$var . "<br>";   // returns 5            32
    echo const_static::SECOND_CONST. "<br>";  // returns 10     33
    const_static::$var = 20;   // now equals 20                34
    echo const_static::$var. "<br>";                           35
?>                                                             36
</body>                                                        37
</html>                                                        38
```

Explanation

Line 10

const FIRST_CONST = "Hello User";

In this line, **const** is the keyword used to declare the constant inside the class, **FIRST_CONST** is the class constant name which holds the string type value **Hello User**. This value will remain constant throughout the program.

Line 11

const SECOND_CONST = 10;

In this line, **const** is the keyword used to declare the constant inside the class, **SECOND_CONST** is the class constant name which holds the integer type value 10. This value will remain constant throughout the program.

Line 12

static $var = 5;

In this line, **$var** is the method declared as static method using the **static** keyword. It holds integer type value 5 which can be changed if required.

Line 13

static $var2 = "The output is displayed";

In this line, **$var2** is the method declared as static method using the **static** keyword. It holds string type value **The output is displayed** which can be changed if required.

Line 14

public static function test($var1 , $var2)

In this line, **function** is the keyword used to create method in the class. The **test($var1, $var2)** is the parameterized method which is declared as static method by using the **static** keyword. Here, **$var1** and **$var2** are the two parameters passed inside the static method. Here, **public** is the keyword used to define the visibility of the method to be public.

Line 18

static function test2()

In this line, **function** is the keyword used to create method in the class. **test2()** is the method which is declared as static method by using the **static** keyword. Here, by default the visibility of the method is public as it is not declared.

Line 22

function showConstant()

In this line, **ShowConstant()** is the method of **const_static** class which is created by using **function** keyword.

Line 24
**echo self::FIRST_CONST . "
";**
In this line, **echo** statement is used to display the value of class constant. **self::FIRST_CONST** is used to call the value of class constant defined in Line 10. Here, **self** is the keyword used to call the class constant inside the class followed by the scope resolution operator (**::**) and class constant name **FIRST_CONST**.

This line will display the following in the browser:

Hello User

Line 25
self::test2();
In this line, **self::test2();** is used to call the static method declared in Line 18. Here, **self** is the keyword used to call the static method inside the class followed by the scope resolution operator (**::**) and static method name **test2()**.

Line 26
**echo self::$var2 . "
";**
In this line, **self::$var2** is used to display the value of static **$var2** property that is defined in Line 13 inside the **const_static** class. Here, **self** is the keyword used to call the static property inside the class followed by the scope resolution operator (**::**) and static **$var2** property name.

This line will display the following in the browser:

The output is displayed

Line 29
$class = new const_static;
In this line, **$class** is the object of class **const_static** created by using the **new** keyword.

Line 30
$class->showConstant();
In this line, **$class** is the object and **showConstant()** is the method. The **$class** object is used with the arrow operator (**->**) to access the **showConstant()** method declared in Line 22.

Line 31
const_static::test("The integer" , "values are:");
In this line, **const_static** is the class name followed by **::** operator and **test()** method. It will call the static method **test()** of class **const_static** declared in Line 14. Here, two string type values are passed to the **test()** method.

Line 32
**echo const_static::$var . "
";**
In this line, **echo** statement is used to display the value of static property declared inside the class **const_static**. The **const_static::$var** is used to call the value of static property defined in Line 13. Here, **const_static** is the class name used to call the static property outside the class followed by the scope resolution operator (**::**) and static property name **$var2**.

This line will display the following in the browser:

5

Line 33
**echo const_static::SECOND_CONST . "
";**
In this line, **echo** statement is used to display the value of class constant. The **const_static::SECOND_CONST** is used to call the value of class constant defined in Line 11. Here, **const_static** is the keyword used to call the class constant outside the class followed by the scope resolution operator (**::**) and class constant name **SECOND_CONST**.

This line will display the following in the browser:

10

The output of Example 9 is displayed in Figure 7-9.

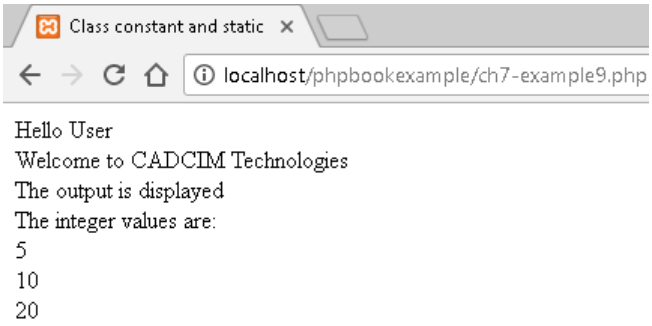

Figure 7-9 *The output of Example 9*

INHERITANCE

Different kind of objects often have certain amount of properties in common. Inheritance is a key feature of object oriented programming. It gives you the benefit of reusing methods and properties of a class and therefore, it reduces the lines of code in a PHP program. Inheritance is the process by which one object can acquire the properties of another object.

Technically, inheritance is a technique of deriving new class from an existing class by reusing the features (properties and methods) of the existing class in that new class. In addition to the inherited features, new class can also have some additional new features.

The class that inherits the property and methods of another class is known as derived class. The derived class is also known as sub class or child class and the class from which the derive class is derived is known as super class or base class or parent class.

The **extends** keyword is used to perform inheritance in PHP. In inheritance, the child class can only inherit all the public and protected features (methods and properties) of the parent class. But, it cannot inherit the private methods and properties of the parent class.

The syntax of inheritance in PHP is as follows:

```
class Parent {
  // The body of Parent class
}

class Child extends Parent {
  // The body of derived class
}
```

In the given syntax, **Parent** represents the name of the base class, **Child** represents the name of the derived class. Here, the class **Child** is derived from base class **Parent** by using the **extends** keyword. The class **Child** will inherit all the non-private methods and the properties of base class **Parent**.

Example 10

The following program will illustrate the use of inheritance. The program will calculate the area of a rectangle and the output is displayed in the browser.

```
<!Doctype html>                                                    1
<html>                                                             2
<head>                                                             3
<title>Inheritance</title>                                         4
</head>                                                            5
<body>                                                             6
<?php                                                              7
    class Rectangle {                                              8
         public $length;                                           9
         public $width;                                            10
         public function __construct($length, $width)              11
         {                                                         12
               $this->length = $length;                            13
               $this->width = $width;                              14
         }                                                         15
   }                                                               16
   class rect_area extends Rectangle {                             17
         public function GetArea()                                 18
         {                                                         19
               return $this->length * $this->width;                20
         }                                                         21
   }                                                               22
   $res = new rect_area(10, 5);                                    23
echo 'The length of rectangle is <b>'.$res->length.'</b><br>';     24
echo 'The width of rectangle is <b>'.$res->width.'</b><br>';       25
echo 'Area of rectangle is <b>'.$res->GetArea().'</b><br>';        26
?>                                                                 27
</body>                                                            28
</html>                                                            29
```

Explanation

Line 11
public function __construct($length, $width)
In this line, **__construct($val)** is the parameterized constructor method of the class **Rectangle**. Here, **$length** and **$width** are the parameters passed inside the parentheses of constructor **__construct()**. It will initialize the object of class **Rectangle**.

Line 13 and Line 14
$this->length = $length;
$this->width = $width;
In these lines, **$this** is the pre-defined variable used to access the current object. The **length** and **width** are the name of the properties without the **$** sign and in this case, **$length** and **$width** properties are assigned to these objects, respectively.

Line 17
class rect_area extends Rectangle {
In this line, **rect_area** is the derived class name which is derived from the **Rectangle** parent class by using the **extends** keyword. Here, **rect_area** class will inherit all the public methods and properties of **Rectangle** class.

Lines 18 to 21
```
public function GetArea()
{
        return $this->length * $this->width;
}
```
These lines contain the definition of the method **GetArea()** of class **rect_area**. In the body of the method, the public properties **$length** and **$width** of the base class **Rectangle** is accessed directly through **$this** keyword. The multiplication operation is performed between both the properties. After multiplication, the resultant value is returned with the help of **return** keyword.

Line 23
$res = new rect_area(10, 5);
In this line, **$res** is the instance of class **rect_area**. The class **rect_area** can access the methods and properties of class **Rectangle**. So the two integer values 10 and 5 passed inside the parentheses of class **rect_area** are assigned to **$length** and **$width** properties of class **Rectangle** at the time of instantiation.

The output of Example 10 is displayed in Figure 7-10.

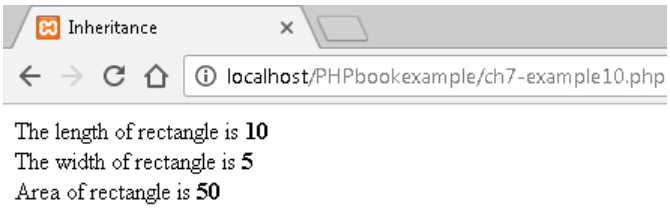

Figure 7-10 *The output of Example 10*

The parent Keyword

To call the value of static method of parent class inside the child class, the keyword **parent** is used followed by the scope resolution operator(**::**) and static method name. For example, **parent::test();** where **test()** is the static method of parent class. The working of **parent** keyword is exactly same as **self** keyword with the only difference that it call the methods and properties of base class.

The **parent** keyword can also be used to differentiate between constructors of parent and child class. To call the constructor of parent class, the **parent** keyword can be used in a similar way as discussed before. For example, **parent::__construct();** where **__construc**t is the constructor of parent class.

If a parent class and derived class both have the methods with same name then that method gets overridden. In such cases, to call the method of parent class, **parent** keyword is used which is followed by the scope resolution operator(**::**) and the name of the parent class method. Whereas **self** keyword is used to call the method of the existing class.

Example 11

The following program will illustrate the use of inheritance with **parent** keyword. The program will prevent overriding of method and constructor with the help of **parent** keyword and the output is displayed in the browser.

```
<!Doctype html>                                                    1
<html>                                                             2
<head>                                                             3
<title>parent keyword</title>                                      4
</head>                                                            5
<body>                                                             6
<?php                                                              7
class A {                                                          8
function __construct()                                             9
 {                                                                 10
  print "I am output of <b>parent class</b> constructor.<br>";     11
 }                                                                 12
function Samefunc()                                                13
 {                                                                 14
   echo "I am output of method <b>Samefunc()</b>                   15
   of parent class <b>A</b>.<br>";
 }                                                                 16
}                                                                  17
class B extends A {                                                18
function __construct()                                             19
 {                                                                 20
   parent::__construct();                                          21
   print "I am output of <b>child class</b> constructor.<br>";     22
 }                                                                 23
function Samefunc()                                                24
{                                                                  25
```

```
    echo "I am output of method <b>Samefunc()</b>                    26
    of child class <b>B</b>.<br>";
    parent::Samefunc();                                              27
   }                                                                 28
  }                                                                  29
  $b = new B;                                                        30
  $b->Samefunc();                                                    31
  ?>                                                                 32
  </body>                                                            33
  </html>                                                            34
```

Explanation

Line 18

class B extends A {

In this line, **B** is the derived class name which is derived from the **A** parent class by using the **extends** keyword. Here, class **B** will inherit all the public methods and properties of class **A**.

Lines 19 to 23

```
function __construct()
 {
        parent::__construct();
        print " I am output of <b>child class</b> constructor.<br>";
 }
```

These lines define the constructor **__construct()** of class **B**. In the body of constructor, the **parent::__construct();** is calling the constructor of parent class **A** declared in Line 9 by using the keyword **parent** followed by **::** operator and constructor name **__construct()**. Here, the use of **parent** keyword is preventing the overriding of constructors. So, this function will execute both the **print** statements of class **A** and class **B** constructors.

Lines 24 to 28

```
function Samefunc()
 {
   echo "I am output of method <b>Samefunc()</b> of child class <b>B</b>.<br>";
   parent::Samefunc();
 }
```

These lines define the method **Samefunc()** of class **B**. In the body of the method, the **parent::Samefunc();** is calling the method of parent class **A** declared in Line 13 by using the keyword **parent** followed by **::** operator and method name **Samefunc()**. Here, the use of **parent** keyword is preventing the overriding of methods. So this function will execute both the **echo** statement of class **A** and class **B** method **Samefunc()** when the object of class **B** will be called in Line 31.

Line 30

$b = new B;

In this line, the **$b** is an object of class **B** created by using the **new** keyword.

Line 31
$b->Samefunc();
In this line, object **$b** is calling the **Samefunc()** method of **B** class which in turn will call the **Samefunc()** method of **A** class because **B** class has inherited all the methods of **A** class.

The output of Example 11 is displayed in Figure 7-11.

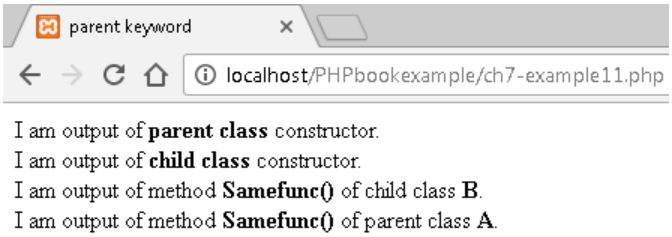

Figure 7-11 *The output of Example 11*

The final Keyword

The **final** keyword can be prefixed with the method of base class to prevent overriding of the method in derived class. It is also known as final method. The syntax of final method is as follows:

```
class Base_cl
{
   final function unique()
   {
         //Body of final method
   }
}
```

In the given syntax, **unique()** represents the method of base class **Base_cl**. The **final** keyword is used to prevent the overriding of method **unique()**.

After using the **final** keyword with the method declared in the base class if the method in derived class is declared with the same name, it will throw an error.

You can also declare a base class as final class to prevent overriding of the methods declared in the base class by prefixing the class with **final** keyword. In this case, there is no need to separately specify the method of base class to be the final method.

The syntax of final class is as follows:

```
final class Base_cl
{
    function unique()
   {
         //Body of method
   }
   function unique2()
```

```
    {
          //Body of method.
    }
}
```

In the given syntax, **unique()** and **unique2()** represent the methods of base class **Base_cl**. The **final** keyword is used to prevent the overriding of methods **unique()** and **unique2()** of class **Base_cl**.

After using the **final** keyword with the base class, if the method in the derived class is declared with the name same as one of the methods in the base class then it will throw an error.

Example 12

The following program will illustrate the use of inheritance with the **final** keyword. The program will prevent overriding of method with the help of **final** keyword and the output is displayed in the browser.

```
<!Doctype html>                                                    1
<html>                                                             2
<head>                                                             3
<title>final keyword</title>                                       4
</head>                                                            5
<body>                                                             6
<?php                                                              7
    class A {                                                      8
function __construct()                                             9
 {                                                                 10
  print "I am output of <b>parent class</b> constructor.<br>";     11
 }                                                                 12
final function Samefunc()                                         13
 {                                                                 14
   echo "I am output of method <b>Samefunc()</b>                   15
   of parent class <b>A</b>.<br>";
 }                                                                 16
}                                                                  17
class B extends A {                                                18
function __construct()                                             19
 {                                                                 20
   parent::__construct();                                          21
   print "I am output of <b>child class</b> constructor.<br>";     22
 }                                                                 23
function Samefunc()                                                24
{                                                                  25
  echo "I am output of method <b>Samefunc()</b>                    26
  of child class <b>B</b>.<br>";
  parent::Samefunc();                                              27
 }                                                                 28
}                                                                  29
```

```
$b = new B;                                                    30
$b->Samefunc();                                                31
?>                                                             32
</body>                                                        33
</html>                                                        34
```

Explanation

The working of this program is the same as the previous programming example (Example 11) except that in Line 13 the **final** keyword is used with the method **samefunc()** of parent class **A**. So this method cannot override in child class **B**. This example will output error displaying the following message in the browser:

Fatal error: Cannot override final method A::Samefunc() in C:\xampp\htdocs\PHPbookexample\ch7-example12.php on line 33

The output of Example 12 is displayed in Figure 7-12.

Figure 7-12 *The output of Example 12*

The instanceof Operator

Before using any object, the **instanceof** operator can be used to check whether an object is an instance of the specified class or subclass. If the object is of the specified class, then the **instanceof** operator evaluates to true otherwise the result is false.

The syntax for the **instanceof** operator is as follows:

```
$object_name instanceof class_name
```

In the given syntax, **$object_name** represents the object name. The **instanceof** is the operator and the **class_name** is the name of class whose instance is to be checked.

Example 13

The following program will check whether the object is an instance of the class by using the **instanceof** operator and the output is displayed in the browser.

```
<!Doctype html>                                                1
<html>                                                         2
<head>                                                         3
<title>instanceof operator</title>                             4
</head>                                                        5
<body>                                                         6
<?php                                                          7
```

```
            class firstClass                                    8
            {                                                   9
                  //Body of class firstClass                    10
            }                                                   11
            class secondClass extends firstClass                12
            {                                                   13
                  //Body of class secondClass                   14
            }                                                   15
      $a = new secondClass;                                     16
      var_dump($a instanceof secondClass);                      17
      var_dump($a instanceof firstClass);                       18
      var_dump(!($a instanceof firstClass));                    19
?>                                                              20
</body>                                                         21
</html>                                                         22
```

Explanation

Line 17

var_dump($a instanceof secondClass);

In this line, the **instanceof** operator is checking whether the **$a** object is an instance of the class **secondClass**.

This line will display the following in the browser:

bool(true)

The working of Line 18 and Line 19 is same as Line 17.

The output of Example 13 is displayed in Figure 7-13.

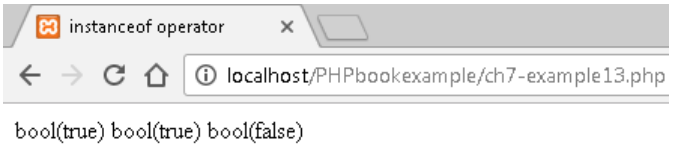

Figure 7-13 *The output of Example 13*

INTERFACE

An interface is a blueprint of a class. It is very much similar to the class but it contains only abstract method. An abstract method is a method that is declared but contains no implementation. In other words, an interface defines the task to be performed by the class but it does not specify the procedure to perform it. The interface may also contain constants but it does not contain any constructor because it cannot be instantiated. Interfaces can only be implemented by classes or can be extended by other interfaces. A single interface can be implemented by any number of classes and vice-versa. This means a class can implement multiple interfaces.

Declaring an Interface

You can declare an interface in the same way as you define a class except that you need to use the **interface** keyword in the definition statement, as shown in the following syntax:

```
interface interface_name
{
   public function();
   public function();
   -------
   -------
}
```

In this syntax, the declaration of interface begins with the keyword **interface** followed by the **interface_name** which is an identifier given by the programmer to specify the name of the interface. Here, **public** specifies the scope of the method. The scope of the method defined inside the interface body must be public. The methods inside an interface end with a semicolon and contain no implementation.

For example:

```
interface demo
{
   public function show( );
   public function sqr( );
}
```

In the given example, the **interface** is the keyword used to declare the interface and **demo** is the interface name. The interface **demo** contains two methods, **show()** and **sqr()** which are declared as **public**. The methods of the interface are not defined in the body of the interface.

Implementing an Interface

Once an interface has been declared, other classes can implement that interface by using the **implements** keyword in the class declaration statement. The syntax for implementing an interface is as follows:

```
class class_name [extends superclass] implements interface_name
{
   //Body of the class
}
```

In the given syntax, the class represented by **class_name** implements the interface that is represented by **interface_name**. If a class implements an interface, all methods declared inside the interface should be implemented by the class. Here, **extends superclass** is optional where **extends** is the keyword used to inherit the parent class and **superclass** represents the parent class.

A class can also implement more than one interface by using comma to separate the list of interfaces. The syntax for implementing more than one interfaces is as follows:

```
class class_name implements interface1, interface2, interfaceN
{
   //Body of the class
}
```

In the given syntax, **interface1**, **interface2**, and **interfaceN** represent the names of different interfaces in a program which can be implemented by a class.

Note

A class cannot perform multiple inheritance. It means that a single class cannot inherit multiple classes. In PHP, multiple inheritance is possible only by using interface. This means a class can inherit multiple interfaces but not multiple classes.

Extending an Interface

In this chapter, you have learned about inheritance. A class inherits the characteristics of another class. In the same way, the concept of inheritance can be applied on interfaces also. An interface can inherit the characteristics of another interface using the **extends** keyword. When an interface is implemented by a class and the interface inherits another interface, the class must provide the implementation for all methods of all interfaces that are inherited.

For example:

```
interface A
{
public function method1( );
}
interface B extends A
{
public function method2( );
}
class demo implements B
{
   public function method1( )
   {
         ----------;
         ----------;
   }
   public void method2( )
   {
         ----------;
         ----------;
   }
}
```

In the given example, the interface **A** contains only one method **method1()** and the interface **B** also contains only one method **method2()**. Here, the interface **B** inherits the characteristics of the interface **A** by using the **extends** keyword and the class **demo** implements it. In this way, inside the demo class, both the methods, **method1( )** and **method2( )**, of the interfaces **A** and **B** are defined.

Example 14

The following program illustrates the use of extended interface. The program will perform certain mathematical operations on the given values and the output is displayed in the browser.

```
<!Doctype html>                                                    1
<html>                                                             2
<head>                                                             3
<title>extend and implement interface</title>                      4
</head>                                                            5
<body>                                                             6
<?php                                                              7
   interface mathematical                                          8
      {                                                            9
         public function mul($num1, $num2);                        10
      }                                                            11
   interface remainder extends mathematical                        12
      {                                                            13
         public function rem($a, $b);                              14
      }                                                            15
   class Extend_interface implements remainder                     16
   {                                                               17
         public $a, $b, $num1, $num2 ;                             18
         public function mul($num1, $num2)                         19
     {                                                             20
         $this->num1 = $num1;                                      21
         $this->num2 = $num2;                                      22
     }                                                             23
  public function rem($a, $b)                                      24
    {                                                              26
         $this->a = $a;                                            27
         $this->b = $b;                                            28
    }                                                              29
  public function result()                                         30
   {                                                               31
   echo "The multiplication is: <b>".($this->num1 *                32
   $this->num2). "</b><br>";
   echo "The remainder is: <b>".($this->a % $this->b).             33
   "</b><br>";
  }                                                                34
}                                                                  35
$res_obj1 = new Extend_interface;                                  36
```

```
$res_obj1->mul(8 , 7);                                    37
$res_obj1->rem(10 , 8);                                   38
return $res_obj1->result();                               39
?>                                                        40
</body>                                                   41
</html>                                                   42
```

Explanation

Lines 8

interface mathematical

In this line, **mathematical** is defined as an interface using the keyword **interface**.

Line 10

public function mul($num1, $num2);

This line contains the declaration part of the method **mul()** where **$num1** and **$num2** are the two parameters passed inside the parentheses of the method **mul()**. This method is declared inside the **mathematical** interface. The implementation part of this method will be defined by the class that implements the **mathematical** interface.

Line 12

interface remainder extends mathematical

In this line, **remainder** is defined as an interface by using the keyword **interface**. Also, it inherits another interface **mathematical** by using keyword **extends**.

Line 14

public function rem($a, $b);

This line contains the declaration part of the **rem()** method where **$a** and **$b** are the two parameters passed inside the parentheses of the method **rem()**. This method is declared inside the **remainder** interface. Therefore, it will be implemented by the class that implements the **remainder** interface.

Line 16

class Extend_interface implements remainder

In this line, the class **Extend_interface** implements the interface **remainder** by using the keyword **implements**. This class can implement all methods of both the interfaces, **remainder** and **mathematical**.

Lines 19 to 23

```
public function mul($num1, $num2)
 {
                $this->num1 = $num1;
                $this->num2 = $num2;
 }
```

These lines contain the implementation of the method **mul()** of the interface **mathematical**. Inside the body of method, **$this** is the pre-defined variable used to access the current object. Here, **num1** and **num2** are the property names without the **$** sign, and **$num1** and **$num2** properties are assigned to them, respectively.

Lines 24 to 29

```
public function rem($a, $b)
 {
                $this->a = $a;
                $this->b = $b;
 }
```

These lines contain the implementation of the method **rem()** of the interface **remainder**. Inside the body of method, **$this** is the pre-defined variable used to access the current object. Here, **a** and **b** are the properties name without the **$** sign, and **$a** and **$b** properties are assigned to them, respectively.

Lines 30 to 34

```
public function result()
 {
        echo "The multiplication is: <b>" .($this->num1 * $this->num2)."</b><br>";
        echo "The remainder is: <b>".($this->a % $this->b). "</b><br>";
 }
```

These lines contain the declaration of the method **result()** of the class **Extend_interface**. Inside the body of method, the public properties **$num1** and **$num2** of the interface **mathematical** are accessed directly through the **$this** keyword. The multiplication operation is performed between both the properties. And, the public properties **$a** and **$b** of the interface **remainder** are accessed directly through **$this** keyword. The remainder operation is performed between both the properties.

The output of Example 14 is displayed in Figure 7-14.

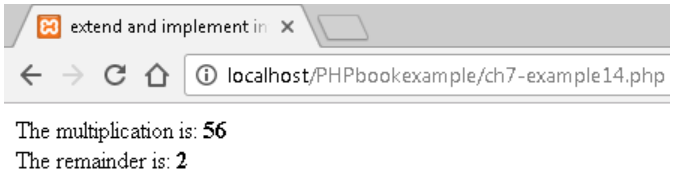

Figure 7-14 *The output of Example 14*

ANONYMOUS CLASS

When a class is declared without a class name, it is known as anonymous class. It is useful to create an object without creating a normal class. Anonymous class is introduced in PHP 7. It can be defined by using keyword **new class**. Anonymous class features are very similar to normal class. It can implement interfaces, extend classes, use constructor, and so on.

If you nest anonymous class inside the outer class (normal class) then the anonymous class cannot access any private or protected methods and properties of the outer class directly. The anonymous class can access the private properties of the outer class by using constructors. And it can also access the protected properties or methods of the outer class by extending the outer class.

Example 15

The following program illustrates the use of anonymous class. The program will perform the addition operation on 3 numbers by accessing the private and protected properties and methods of outer class inside the anonymous class and display the output in the browser.

```
<!Doctype html>                                                    1
<html>                                                             2
<head>                                                             3
<title>Anonymous class</title>                                     4
</head>                                                            5
<body>                                                             6
<?php                                                              7
class Normal                                                       8
{                                                                  9
    private $property = 20;                                        10
    protected $property2 = 30;                                     11
    protected function number()                                    12
    {                                                              13
        return 100;                                                14
    }                                                              15
    public function anoy_func()                                    16
    {                                                              17
        return new class($this->property) extends Normal {        18
            private $property3;                                    19
            public function __construct($property)                 20
            {                                                      21
                $this->property3 = $property;                      22
            }                                                      23
    public function sum()                                          24
    { echo "The sum is ";                                         25
  return $this->property2 + $this->property3 + $this->number();   26
              }                                                    27
        };                                                         28
    }                                                              29
}                                                                  30
echo (new Normal)->anoy_func()->sum();                             31
?>                                                                 32
</body>                                                            33
</html>                                                            34
```


Explanation

Line 18

return new class($this->property) extends Normal {

In this line, the function **anoy_func()** declared in Line 16 is returning an anonymous class. Here, **new class** is the keyword used to create anonymous class. The anonymous class is extending the outer class **Normal** to access the protected property **$property2** and protected method **number()** in anonymous class.

Line 20
public function __construct($property)
In this line, the private property **$property** declared in Line 10 of outer class **Normal** is accessed by using constructor **__construct()** inside the anonymous class.

Line 31
echo (new Normal)->anoy_func()->sum();
In this line, the object of class **Normal** is called directly by using the **new** keyword which is further calling the functions **anoy_funct()** and **sum()** of anonymous class to display the result. This line will display the following in the browser:

The sum is 150

The output of Example 15 is displayed in Figure 7-15.

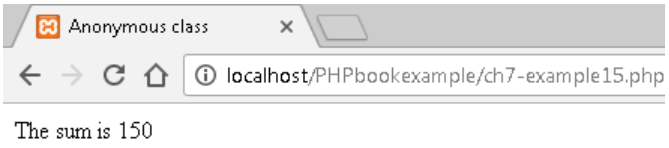

Figure 7-15 *The output of Example 15*

Self-Evaluation Test

Answer the following questions and then compare them to those given at the end of this chapter:

1. A _____ is a group of statements that perform a specific task and can be used multiple times in a program.

2. Function _____ are used to pass information to the function.

3. The _____ function is used to display the variable related information.

4. The _____ stands for Cryptographically Secure Pseudo-Random Number Generator.

5. The property and method scope specifies the _____ of the properties or the methods of a class.

6. The **random_int()** function is used to generate cryptographically secure pseudo-random bytes in PHP. (T/F)

7. A constructor is a special method which is used to initialize the objects of a class. (T/F)

8. A class can implement only one interface. (T/F)

9. An anonymous class is declared without a class name. (T/F)

Review Questions

Answer the following questions:

1. Function name can start with which of the following options?

   (a) Only an alphabet
   (b) Only an underscore
   (c) Only digits
   (d) an alphabet or an underscore

2. Which of the following is the correct function name?

   (a) _aBC12()
   (b) 2cOUNT()
   (c) Hello#()
   (d) 4hEllo_1()

3. Which of the following functions is used to display the variable related information including its data type?

   (a) **var_data()**
   (b) **var_detail()**
   (c) **var_dump()**
   (d) **var_info()**

4. Which of the following functions is used to generate cryptographically secure pseudo-random bytes in PHP?

   (a) **random_int()**
   (b) **secure_byte()**
   (c) **random_bytes()**
   (d) **crypt_dara()**

5. Which of the following is a collection of data and methods?

   (a) class
   (b) function
   (c) constructor
   (d) destructor

6. The **private** methods or the properties of a class can be accessed by the members of:

   (a) same class and other classes
   (b) same class
   (c) other classes
   (d) inherited class

7. Which of the following is known as living entity within a program?

   (a) Class
   (b) Properties
   (c) Methods
   (d) Object

8. Which of the following keywords is used to call the value of static method of parent class inside the child class?

   (a) **parent**
   (b) **self**
   (c) **friend**
   (d) **final**

EXERCISES

Exercise 1

Write a program to calculate the area of a square using inheritance.

Exercise 2

Write a program to add and subtract two numbers by implementing and extending the interface.

Answers to Self-Evaluation Test

1. function, **2.** arguments, **3. var_dump()**, **4.** CSPRNG, **5.** visibility, **6.** F, **7.** T, **8.** F, **9.** T