Session 12

Object-Oriented Concepts in PHP

Session Overview

In this session, learners would be able to:

- Explain Object-Oriented Programming Features in PHP
- Explain about PHP Constructors and Destructors
- Describe PHP Traits
- Elaborate on PHP Namespaces and Iterables

Classes and Objects in PHP [1-3]

Class

Variables (also known as properties), constants, and functions (often also known as methods) can all be found in a class and are used to accomplish one or more tasks.

Objects

Individual copies or instances of a class are referred to as objects. A single class can have an unlimited number of objects.

Classes and Objects in PHP [2-3]

Class	Objects
Animals	Elephant Tiger Leopard
Motorbikes	Harley-Davidson BMW Ducati

Table: Examples of Classes and Objects

Classes and Objects in PHP [3-3]

Terminology	Description
Member Properties	Variables declared inside a class are called member properties or member variables. These are also called data members. Member variables can be accessed only through member functions and not outside the class directly. Inside objects, these variables are called attributes.
Member Methods	Functions or methods declared inside the class are called member functions or methods. These methods can access data members of the class.
Inheritance	Through inheritance, a class can reuse the properties of an already defined class. The class which reuses the properties is called a sub class or derived class or child class. The class from which the properties are reused is called a super class or base class or parent class. A derived class can reuse either all or only the required properties from the parent class.
Constructor	This is a special type of function, which is invoked automatically at the time of object creation.
Destructors	This is a special type of function, which is used to delete the memory occupied by an object. It is invoked automatically when an object is deleted or goes out of scope.

Table: Important Terms Used in OOP in PHP

Defining a Class in PHP

Code Snippet:

```
<html>
<body>
<?php
class City {
  // Properties
  public $name;
  public $country;
  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function set_country($country)
{
   $this->country=$country;
}
function get_country() {
    return $this->country;
  }

  function get_name() {
    return $this->name;
  }
}
// creating object of class

$NewYork = new City();
$America = new City();
$NewYork->set_name('New York');
$America->set_country('America');
echo $NewYork ->get_name();
echo "<br>";
echo $America->get_country();
?>
</body>
</html>
```

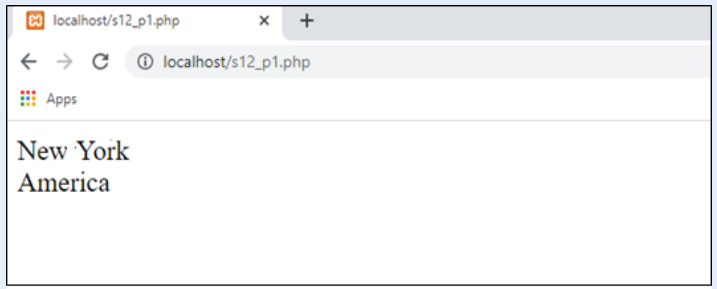

Figure: Output for Code Snippet

Objects Creation in PHP

Following are some examples showing the usage of `new` operator to create the objects:

- `$physics = new Books;`
- `$maths = new Books;`
- `$chemistry = new Books;`

Two of the keywords which deal with object operations are:

PHP – `$this` Keyword

PHP – `instanceof` keyword

PHP – $this Keyword [1-2]

Code Snippet:

```
<html>
<body>
<?php
class Flower {
  // Properties
  public $name;
  public $color;
  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
  function set_color($color) {
    $this->color = $color;
  }
  function get_color() {
    return $this->color;
  }
}
$Rose = new Flower();
$Red=new Flower();
$Rose->set_name('Rose');
$Red->set_color('Red');
echo "Name: " . $eRose->get_name();
echo "<br>";
echo "Color: " .  $Red->get_color();
?>
</body>
</html>
```

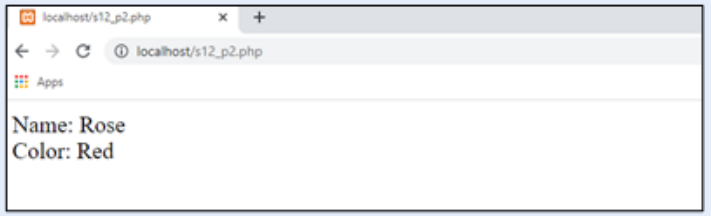

Figure: Output for Code Snippet

PHP – $this Keyword [2-2]

Following are the two ways of using `$this` keyword:

- Outside the class
- Inside the class

Using $this keyword outside the class

Code Snippet:

```
<html>
<body>
<?php
class House {
  public $color;
}
$white = new House();
$white->color = "WHITE";
echo $white->color;
?>
 </body>
</html>
```

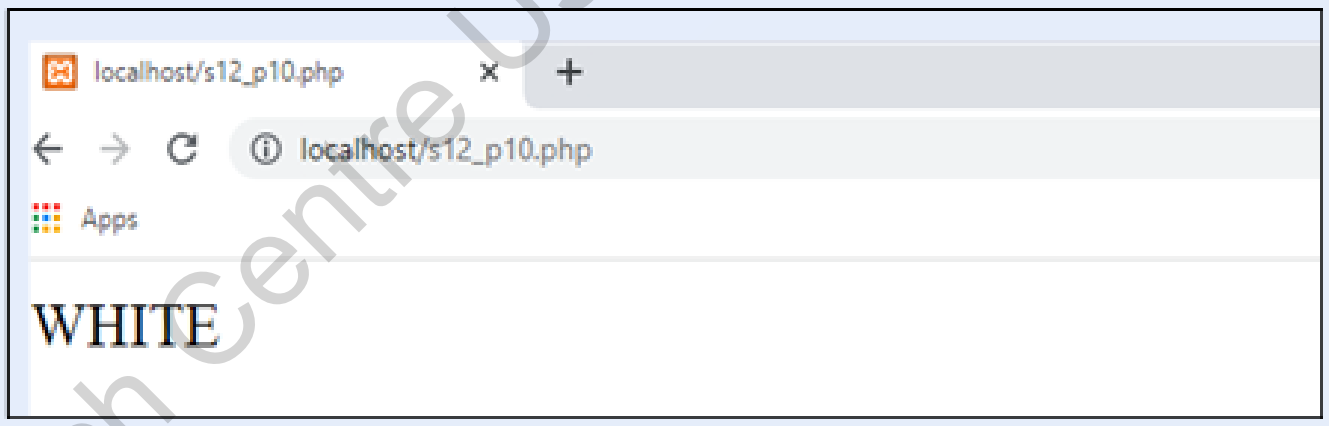

Figure: Output for Code Snippet

Using $this keyword inside the class

Code Snippet:

```
<html>
<body>
<?php
class House {
  public $color;
  function set_color($color) {
    $this->color = $color;
  }
}
$Black = new House();
$Black->set_color("Black");
echo $Black->color;
?>
</body>
</html>
```

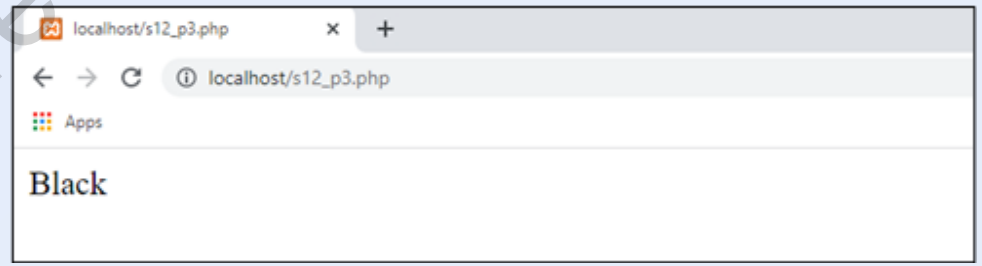

Figure: Output for Code Snippet

PHP – instanceof keyword

Code Snippet:

```
<html>
<body>
<?php
class City {
  // Properties
  public $name;
  public $country;
 // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
$London = new City();
var_dump($London instanceof City);
?>
</body>
</html>
```

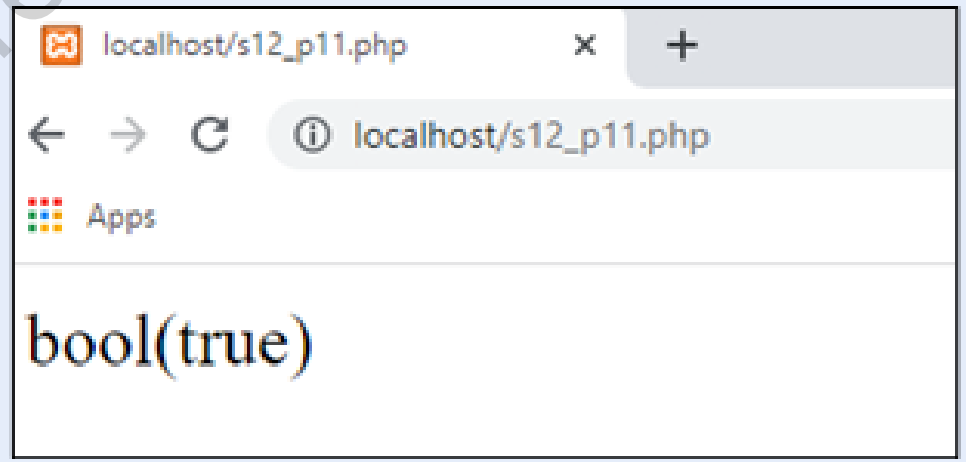

Figure: Output for Code Snippet

PHP OOP – Constructor and Destructor

Constructors	• Constructors are special number methods invoked automatically when an object is created.
Destructors	• Destructors are special member methods invoked automatically when an object is destroyed or an object goes out of scope. Usually, it is invoked at the end of the script.

Constructor Method [1-2]

Code Snippet:

```
<html>
<body>
<?php
class Example {
  public $name;
  public $color;
  function __construct($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
$alpha = new Example("Alpha");
echo $alpha->get_name();
?>
 </body>
</html>
```

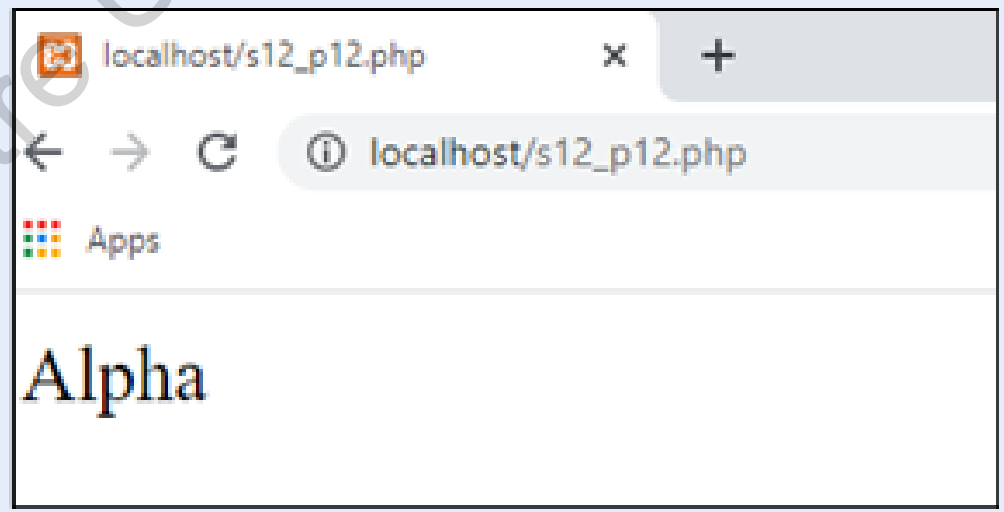

Figure: Output for Code Snippet

Constructor Method [2-2]

Code Snippet:

```
<html>
<body>
<?php
class City {
  public $name;
  public $country;
 function __construct($name, $country) {
    $this->name = $name;
    $this->country = $country;
  }
  function get_name() {
    return $this->name;
  }
  function get_country() {
    return $this->country;
  }
}
$london = new City("London", "England");
echo $london->get_name();
echo "<br>";
echo $london->get_country();
?>
</body>
</html>
```

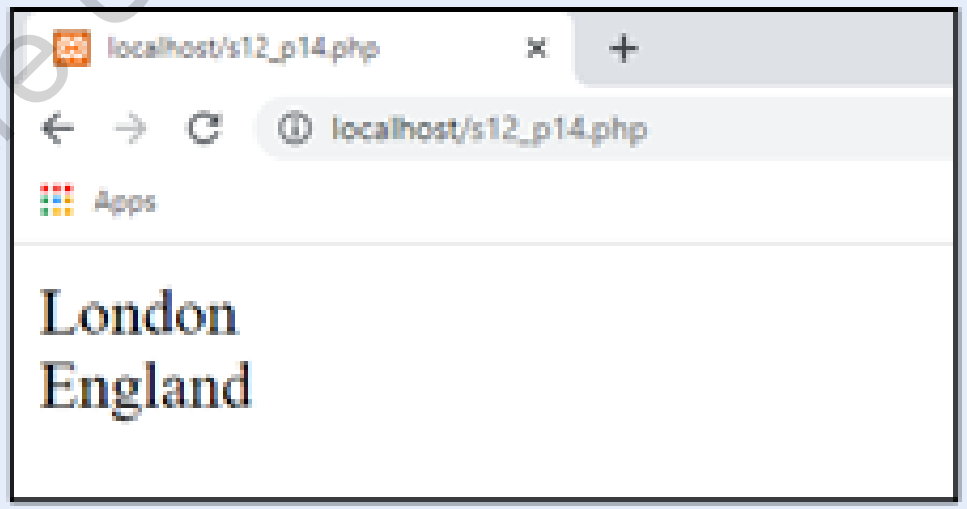

Figure: Output for Code Snippet

Destructor Method

Code Snippet:

```
<html>
<body>
<?php
class Flower {
  public $name;
  public $color;
  function __construct($name) {
    $this->name = $name;
  }
  function __destruct() {
    echo "The flower is {$this->name}.";
  }
}
$Lotus = new Flower("Pink");
?>
 </body>
</html>
```

Figure: Output for Code Snippet

PHP OOP – Traits [1-2]

Code Snippet:

```
<html>
<body>
<?php
trait T1 {
  public function msg1() {
    echo "This is an example of a trait!";
  }
}
class Message {
  use T1;
}
$obj = new Message();
$obj->msg1();
?>
 </body>
</html>
```

Output: This is an example of a trait!

PHP OOP – Traits [2-2]

With traits, users can declare methods that can be used in more than one class.

By using traits, code redundancy can be reduced.

A user cannot instantiate a trait on its own. Objects cannot be created.

Traits are allowed to have methods and abstract methods in any visibility mode (public, private or protected).

Traits are declared same as that of method, but with the `trait` keyword as prefix.

PHP – Using Multiple Traits

Code Snippet:

```
<html>
<body>
<?php
trait T1 {
  public function msg1() {
    echo " This is an example of a trait! <br>";
  }
}
trait T2 {
  public function msg2() {
    echo "Code reusability can be achieved through OOP feature
of traits!";
  }
}
class Message1 {
  use T1;
}
class Message2 {
  use T1, T2;
}
$obj1 = new Message1();
$obj1->msg1();
echo "<br>";
$obj2 = new Message2();
$obj2->msg1();
$obj2->msg2();
?>
 </body>
</html>
```

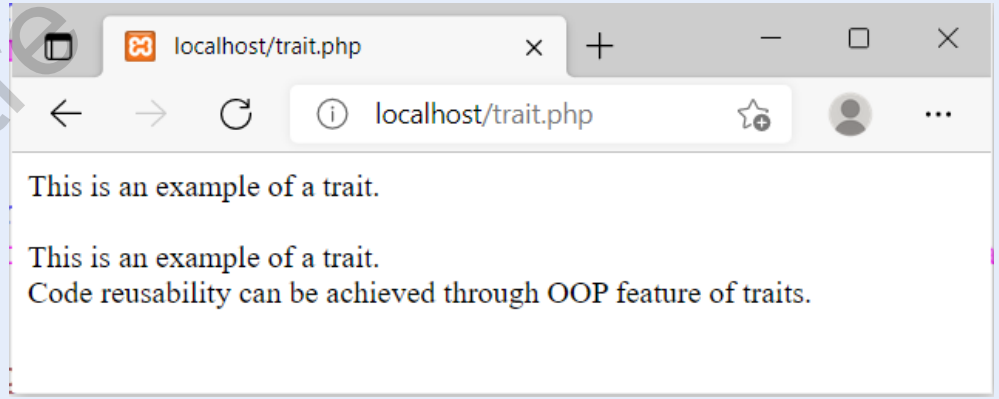

Figure: Output for Code Snippet

PHP Namespaces [1-2]

Explanation 1

Assume that a program has a global variable named `$title`, inside a method. This program has another variable with the same name `$title`. If user assigns and changes values of `$title` inside of the method, the global variable is not affected. It will remain unchanged. This is called scope of a variable. Following the same method, two classes can be declared with same name to give it scope.

Explanation 2

Assume that a user is creating an open-source PHP library to send mails and the user shares that with the developer community. The library of the user has a class named `Email`. If a developer who already has a class called `Email` downloads this library, name collision/conflict will occur. The user can now either rename those classes or use Namespacing.

PHP Namespaces [2-2]

Explanation 3

Assume that there are two people with the name 'John'. To separate them, their surname can be used. In the same way, two classes having the same name can be separated by Namespacing.

By using namespaces two different problems are solved:

- They group the classes that work together to perform a task. This allows the user to organize code in a better manner.
- The same name can be used for more than one class, therefore, avoiding dilemma among names.

Requirement of Namespaces in PHP and Declaring a Namespace

Code Snippet:

```
<?php
namespace PHP;
class Table1 {
 public $title = "";
 public $numRows = 0;
 public function msg() {
  echo "<p>Table {$this->title} has {$this->numRows}
rows.</p>";
 }
}
$table1 = new Table1();
$table1->title = "<br>My table ";
$table1->numRows = 10;
?>
<!DOCTYPE html>
<html>
<body>
<?php
$table1->msg();
?>
</body>
</html>
```

Figure: Output for Code Snippet

Using namespaces

Code Snippet:

index.php

```
<?php
include "HTML.php";
$table = new Html1\Table1();
$table->title = "<br>My table";
$table->numRows = 7;
$row = new Html1\Row();
$row->numCells = 4;
?>
<html>
<body>
<?php $table->msg(); ?>
<?php $row->msg(); ?>
</body>
</html>
```

HTML.php

```
<?php
namespace Html1;
class Table1 {
  public $title = "";
  public $numRows = 0;
  public function msg() {
    echo "<p>Table {$this-
>title} has {$this->numRows}
rows.</p>";
  }
}
class Row {
  public $numCells = 0;
  public function msg() {
    echo "<p>The row has
{$this->numCells} cells.</p>";
  }
}
?>
```

Figure: Output for Code Snippet

PHP Iterables [1-2]

Code Snippet:

```
<html>
<body>
<?php
function displayIterable(iterable
$testIterable) {
  foreach($testIterable as $item) {
    echo $item;
  }
}
$arr1 = ["1", "2", "3"];
displayIterable($arr1);
?>
</body>
</html>
```

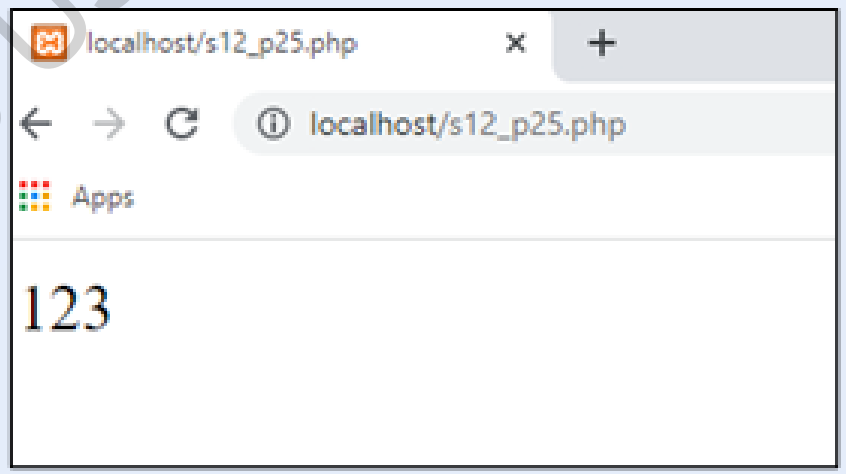

Figure: Output for Code Snippet

PHP Iterables [2-2]

Code Snippet:

```
<html>
<body>
<?php
function getIterable():iterable {
  return ["x", "y", "z"];
}
$testIterable = getIterable();
foreach($testIterable as $alpha) {
  echo $alpha;
}
?>
</body>
</html>
```

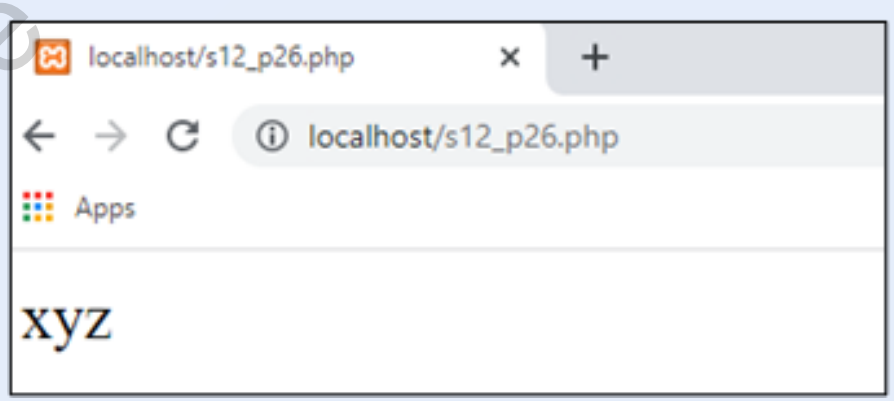

Figure: Output for Code Snippet

Arrays and Iterators

Arrays

- All arrays are iterables. Therefore, any array can be used as an argument of a method. However, it requires an iterable.

Iterators

- PHP contains an interface called `Iterator`, which can be used as an argument of a function that requires an `iterable`.

Iterator Methods [1-2]

An iterator should have following methods:

`current()`	• It returns the element that the pointer is currently pointing to. It can be of any data type.
`key()`	• It returns the key associated with the current element in the list. The data type of the key can only be an integer, float, Boolean, or string.
`next()`	• It moves the pointer to the next element in the list.
`rewind()`	• It moves the pointer to the beginning (first element) of the list.
`valid()`	• If the internal pointer is operated to point to an invalid element of the list (for example, if `next()` was called at the end of the list), this function should return false. In any other case, it returns true.

Iterator Methods [2-2]

Code Snippet:

```
<html>
<body>
<?php
// Create an Iterator
class TestIterator implements Iterator {
  private $alpha = [];
  private $pointer = 0;
  public function __construct($alpha) {
     $this->alpha = array_values($alpha);
  }
  public function current() {
    return $this->alpha[$this->pointer];
  }
  public function key() {
    return $this->pointer;
  }
  public function next() {
    $this->pointer++;
  }
  public function rewind() {
    $this->pointer = 0;
  }
  public function valid() {
     return $this->pointer < count($this->alpha);
  }
}
// A function that uses iterables
function printIterable(iterable $testIterable) {
  foreach($testIterable as $alpha) {
    echo $alpha;
  }
}

// Use the iterator as an iterable
$iterator = new TestIterator(["m", "n", "o"]);
printIterable($iterator);
?>
</body>
</html>
```

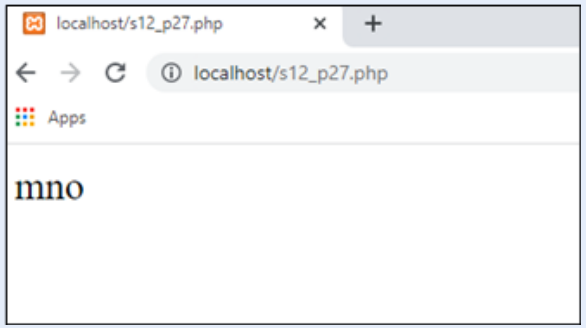

Figure: Output for Code Snippet

Summary

- Object-Oriented Programming (OOP) is a programming paradigm that is mainly based on the use of classes and objects.
- A class is a data type that can contain constants, variables (properties'), and functions (methods).
- An object is an instance of a class and is created using new operator.
- A class is created by using the keyword class, followed by the class name, and a pair of curly brackets ({}).
- $this is a reserved keyword used to access the current object.
- PHP makes use of instanceof keyword to check if an object belongs to a particular class.
- Constructors and destructors are special member functions.
- A constructor is invoked automatically when an object is created.
- A destructor is invoked automatically when an object is destroyed or goes out of scope.
- In PHP, the traits concept is used to inherit multiple behaviors.
- Namespaces are qualifiers that help to organize code better and use the same name for multiple classes.