PHP Classes and Objects

In the dynamic landscape of web development, PHP stands as a stalwart, powering countless websites and applications. Its versatility and extensibility have made it a top choice among developers worldwide. While PHP offers a robust foundation for crafting web solutions, its true potential emerges when combined with the principles of Object-Oriented Programming (OOP).

Welcome to our comprehensive guide on "PHP Classes and Objects." In this article, we will delve deep into the heart of OOP in PHP, where we will explore the concept of classes and objects - the cornerstone of OOP.

Objectives

1. Object Creation and Utilization: By the end of this lesson, students should be able to create instances (objects) of PHP classes, define properties, and invoke methods on these objects to demonstrate their understanding of the fundamental concept of classes and objects.
2. Encapsulation Mastery: After studying this lesson, learners should be proficient in encapsulating data within PHP classes, utilizing access modifiers like public, private, and protected, and explaining the importance of data encapsulation for code security and integrity.
3. Inheritance and Code Reusability: Students should be able to grasp the concept of inheritance, create subclasses that extend base classes, and demonstrate code reuse through inheritance. They should understand the benefits of using inheritance in PHP for building more efficient and maintainable code.
4. Polymorphism and Method Overriding: By the end of this lesson, participants should be able to implement method overriding by creating their own versions of methods defined in base classes. They should understand how polymorphism allows different classes to conform to a common interface while maintaining unique behavior.
5. These objectives are designed to ensure that students achieve specific learning outcomes related to the creation, utilization, and manipulation of PHP classes and objects, as well as the application of key Object-Oriented Programming (OOP) principles like encapsulation, inheritance, and polymorphism.

Understanding Classes and Objects

Let's start with basic definitions of classes and objects in Object-Oriented Programming (OOP):

Class: In OOP, a class is a blueprint or template for creating objects. It defines the structure, attributes (properties), and behaviors (methods) that objects of the class will possess. Think of a class as a blueprint for a specific type of object, describing how objects of that class should be created and what they can do. Classes encapsulate the characteristics and functionalities shared by multiple objects.

Object: An object is an instance of a class. It is a concrete, tangible entity created based on the blueprint provided by the class. Objects represent specific, real-world entities and encapsulate both data (values of properties) and the actions or operations (methods) that can be performed on that data. Objects are the runtime instances that interact with each other and perform tasks within a program.

Classes define the structure and behavior of objects, while objects are the actual instances or representatives of those classes in your program. Classes serve as templates, and objects are the instances that adhere to the defined blueprint.

What is a PHP Class?

In PHP, a class is a fundamental concept in Object-Oriented Programming (OOP). It serves as a blueprint or template for creating objects, which are instances of the class. A class defines the structure, attributes (properties), and behaviors (methods) that objects created from it will possess.

Key Characteristics of a PHP Class:

1. Properties (Attributes): A class can have properties, which are variables that hold data specific to the class. These properties define the characteristics or attributes of objects created from the class.
2. Methods (Functions): A class can have methods, which are functions that define the behaviors or actions that objects of the class can perform. Methods are defined within the class and can operate on the class's properties.
3. Constructor: A constructor is a special method in a class that gets executed when an object of the class is created. It is often used to initialize the object's properties.
4. Access Modifiers: PHP classes can specify access modifiers like public, private, and protected to control the visibility and accessibility of properties and methods. For example, public properties and methods can be accessed from outside the class, while private properties and methods are restricted to within the class.

Example (php_class.php):

```
<!DOCTYPE html>
<html>
<head>
    <title>PHP Class Example</title>
</head>
<body>

<?php
class Person {
    // Properties
    public $name;
    public $age;

    // Constructor
    public function __construct($name, $age) {
        $this->name = $name;
        $this->age = $age;
    }

    // Method
    public function greet() {
```

```
        return "Hello, my name is {$this->name} and I am {$this->age} years old.";
    }
}

// Create objects (instances) of the "Person" class
$person1 = new Person("Juan", 30);
$person2 = new Person("Pedro", 25);

?>

<h1>Person Information</h1>
<p><?php echo $person1->greet(); ?></p>
<p><?php echo $person2->greet(); ?></p>

</body>
</html>
```

Explanation in general: this code defines a Person class, creates two Person objects with different attributes (name and age), and displays personalized greeting messages for each person on the web page. It's a simple example of using classes and objects in PHP to model and represent real-world entities.

Let's break down what the code does step by step:

- <!DOCTYPE html>: This declaration specifies the document type and version of HTML being used, which is HTML5 in this case.
- <html>: The opening HTML tag indicates the start of an HTML document.
- <head>: The <head> section contains metadata about the document, such as the title that appears in the browser tab.
- <title>: The <title> element sets the title of the web page, which will be displayed in the browser tab as "PHP Class Example."
- </head>: This closing tag marks the end of the <head> section.
- <body>: The <body> section contains the content that will be displayed in the web page.
- PHP Code Block: Inside the <body> section, there's a PHP code block that starts with <?php and ends with ?>. This is where PHP code is executed.
- class Person { ... }: This part defines a PHP class named Person. The class has properties ($name and $age), a constructor method (__construct), and a method (greet). The constructor initializes the properties with values passed as arguments when an object of the class is created, and the greet method returns a greeting message using the properties.
- $person1 = new Person("Juan", 30); and $person2 = new Person("Pedro", 25);: These lines create two objects (instances) of the Person class, namely $person1 and $person2, with different names and ages.
- <h1>Person Information</h1>: This is an HTML heading that displays "Person Information" as a title.
- <?php echo $person1->greet(); ?> and <?php echo $person2->greet(); ?> : These lines are PHP code embedded in HTML. They call the greet method of the $person1 and

$person2 objects and display the returned greeting messages within <p> (paragraph) elements in the web page.

Class Properties in PHP

In PHP, class properties (also known as attributes or member variables) are variables that belong to a class. They define the characteristics or attributes that objects created from the class will possess. Each object created from a class has its own set of property values. Class properties are used to store and manage data related to the objects of the class.

Here's how class properties work in PHP:

1. Declaring Class Properties:

- Class properties are declared within the class using the **public**, **private**, or **protected** access modifiers, followed by the variable name.
- The **public** access modifier allows properties to be accessed and modified from outside the class.
- The **private** access modifier restricts access to properties to within the class itself.
- The **protected** access modifier allows access within the class and its subclasses (child classes).

2. Initializing Properties:

- Properties can be initialized with default values in the class constructor or directly in the property declaration.
- If properties are not explicitly initialized, they are considered to have a default value of **null**.

3. Accessing Properties:

- Class properties can be accessed using the object of the class followed by the **->** operator.
- The access level (public, private, or protected) determines where properties can be accessed.

4. Modifying Properties:

- Properties with the **public** access modifier can be modified directly.
- Properties with the **private** and **protected** access modifiers are typically modified using getter and setter methods.

Example (php_class_property.php):

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.      <title>Class Properties Example</title>
5.  </head>
6.  <body>
7.
8.  <?php
9.  // Define a class with properties
10. class MyClass {
```

```
    public $publicProperty = "Public Value";   // Public property
    private $privateProperty;                  // Private property
    protected $protectedProperty = "Protected Value"; // Protected property

    // Constructor to initialize properties
    public function __construct($privateValue) {
        $this->privateProperty = $privateValue;
    }

    // Getter method for private property
    public function getPrivateProperty() {
        return $this->privateProperty;
    }

    // Setter method for private property
    public function setPrivateProperty($newValue) {
        $this->privateProperty = $newValue;
    }

    // Method to access protected property within the class
    public function accessProtectedProperty() {
        return $this->protectedProperty;
    }
}

// Create an object of the class
$object = new MyClass("Private Initial Value");

// Access and display public property
echo "<p>Public Property: " . $object->publicProperty . "</p>";

// Access private property using getter method
echo "<p>Private Property (via Getter): " . $object->getPrivateProperty() . "</p>";

// Modify private property using setter method
$object->setPrivateProperty("New Private Value");

// Access and display modified private property using getter method
echo "<p>Private Property (Modified via Setter): " . $object->getPrivateProperty() .
"</p>";

// Access protected property (within class)
echo "<p>Protected Property (Within Class): " . $object->accessProtectedProperty() .
"</p>";
?>

</body>
</html>
```

Explanation:

- We define the **MyClass** class with three properties: a public property, a private property, and a protected property.
- The constructor method **__construct** initializes the private property.

- Getter and setter methods (**getPrivateProperty** and **setPrivateProperty**) are provided to access and modify the private property, demonstrating encapsulation.
- We create an object of the **MyClass** class and use it to access and manipulate the properties.
- The HTML part of the file is used to display the property values.

When you run this PHP file in a web server environment, it will display the values of public, private, and protected properties along with their modifications using getter and setter methods.

Class properties are essential for encapsulating data within objects and controlling their visibility and accessibility. They allow objects to have unique characteristics while sharing the same blueprint (class). Access modifiers provide control over who can read and modify the property values, enhancing data security and encapsulation.

Methods for PHP Classes

Here's how methods work in PHP classes:

1. **Declaring Methods:**

- Methods are declared within a class using the public, private, or protected access modifiers, followed by the method name and its code block.
- The public access modifier allows methods to be called from outside the class.
- The private access modifier restricts access to methods to within the class itself.
- The protected access modifier allows access within the class and its subclasses (child classes).

2. **Method Parameters:**

- Methods can accept parameters, which are variables passed to the method when it is called. Parameters are enclosed within parentheses after the method name.
- These parameters allow methods to receive input data and perform actions based on that data.

3. **Method Parameters:**

- Methods can accept parameters, which are variables passed to the method when it is called. Parameters are enclosed within parentheses after the method name.
- These parameters allow methods to receive input data and perform actions based on that data.

4. **Method Parameters:**

- Methods can accept parameters, which are variables passed to the method when it is called. Parameters are enclosed within parentheses after the method name.
- These parameters allow methods to receive input data and perform actions based on that data.

Example (php_method.php):

```php
<!DOCTYPE html>
<html>
<head>
    <title>Methods Example</title>
</head>
<body>

<?php
// Define a class with methods
class Calculator {
    // Public method to add two numbers
    public function add($a, $b) {
        return $a + $b;
    }

    // Public method to subtract two numbers
    public function subtract($a, $b) {
        return $a - $b;
    }
}

// Create an object of the class
$calculator = new Calculator();

// Call the "add" method
$sum = $calculator->add(10, 5);

// Call the "subtract" method
$difference = $calculator->subtract(10, 5);
?>

<h1>Calculator</h1>

<p>Sum of 10 and 5 is: <?php echo $sum; ?></p>

<p>Difference between 10 and 5 is: <?php echo $difference; ?></p>

</body>
</html>
```

Explanation:

- We define the Calculator class with two public methods, add and subtract. These methods perform addition and subtraction operations, respectively.
- We create an object of the Calculator class named $calculator.
- We call the add method on the $calculator object to add 10 and 5, and we store the result in the $sum variable.
- We call the subtract method on the $calculator object to subtract 5 from 10, and we store the result in the $difference variable.

- In the HTML part of the file, we display the results of the addition and subtraction operations.

When you run this PHP file in a web server environment, it will display the sum and difference of 10 and 5, demonstrating the use of methods in PHP classes to encapsulate functionality and perform actions on objects.

What Is an Object in PHP?

In PHP, an object is an instance of a class. It represents a concrete, real-world entity or concept that has been created based on the blueprint provided by the class. Objects are the building blocks of object-oriented programming (OOP) and are used to model and manipulate data and behavior in a structured and organized manner.

Here are key points to understand about objects in PHP:

1. Instances of Classes: Objects are created from classes. A class defines the structure (properties) and behavior (methods) that objects of that class will possess. When you create an object from a class, you are creating an instance that adheres to that class's blueprint.
2. Data and Behavior: Objects encapsulate both data (property values) and the actions or operations (methods) that can be performed on that data. This bundling of data and behavior makes objects self-contained and modular.
3. Unique Properties: Each object created from a class can have its own unique set of property values. For example, if you have a Person class, you can create multiple Person objects, each with its own name, age, and other characteristics.
4. Interaction: Objects can interact with each other by invoking methods on other objects. This allows for complex behaviors and relationships to be modeled within an application.
5. Encapsulation: OOP promotes the concept of encapsulation, which means that objects hide their internal details (properties and methods) from the outside world. Access to an object's properties and methods is controlled by access modifiers (e.g., public, private, protected) defined in the class.
6. Reusability: Objects and classes promote code reusability. Once you have defined a class, you can create multiple objects from it in different parts of your application, reducing the need to duplicate code.

Example (php_object.php):

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.      <title>Object Example</title>
5.  </head>
6.  <body>
7.
8.  <?php
```

```php
// Define a class
class Person {
    public $name;
    public $age;

    public function greet() {
        return "Hello, my name is {$this->name} and I am {$this->age} years old.";
    }
}

// Create an object (instance) of the class
$person1 = new Person();
$person1->name = "Juan";
$person1->age = 30;

// Call the "greet" method on the object
$greeting = $person1->greet();
?>

<h1>Person Information</h1>

<p>Name: <?php echo $person1->name; ?></p>
<p>Age: <?php echo $person1->age; ?></p>
<p>Greeting: <?php echo $greeting; ?></p>

</body>
</html>
```

Explanation:

- We define the Person class with properties $name and $age and a method greet that returns a greeting message.
- We create an object of the Person class named $person1.
- We set values for the $name and $age properties of $person1.
- We call the greet method on $person1 and store the result in the $greeting variable.
- In the HTML part of the file, we display the name, age, and greeting properties of the object, demonstrating the use of an object to encapsulate data and behavior.

When you run this PHP file in a web server environment, it will display the person's name, age, and a greeting message, all managed by the Person object.

Summary

In this lesson, we delved into the fundamental concept of objects in PHP, a cornerstone of object-oriented programming (OOP). Objects are instances of classes, serving as concrete representations of real-world entities or concepts. They encapsulate both data, in the form of properties, and behavior, through methods, within a structured blueprint defined by the class. Objects promote code organization,

reusability, and maintainability, allowing for the modeling of complex systems by breaking them into smaller, self-contained units.

Through a practical example, we demonstrated how to create and utilize objects. We defined a Person class with properties for name and age, and a method for generating a personalized greeting. By instantiating objects from this class and manipulating their properties and methods, we witnessed how objects enable the creation of dynamic and modular code. Understanding objects in PHP is pivotal for harnessing the power of OOP, as they empower developers to build scalable and organized applications with ease.

Quiz

1. **What is OOP in PHP?**

a) Object-Oriented Programming
b) Object-Oriented Process
c) Object-Oriented Protocol
d) Object-Oriented Protocol

2. **In OOP, what does "class" refer to?**

a) An instance of an object
b) A blueprint or template for creating objects
c) A variable that stores object data
d) A function that performs an action

3. **Which keyword is used to create an object in PHP?**

a) new
b) create
c) instantiate
d) object

4. **What are properties in a PHP class?**

a) Methods that define behavior
b) Functions that create objects
c) Variables that store data
d) Constants that define values

5. **Which access modifier allows a class property to be accessed from outside the class?**

a) private
b) protected
c) public
d) static

6. **What does a constructor method do in a PHP class?**

a) Initializes class methods
b) Defines class properties
c) Creates new objects
d) Initializes class properties when an object is created

7. **Which method is automatically called when an object is created?**

a) create()
b) initialize()

c) constructor()
d) destructor()

8. **What is encapsulation in OOP?**

a) The process of creating objects
b) The process of hiding the internal details of an object
c) The process of destroying objects
d) The process of defining class methods

9. **In OOP, what does "polymorphism" mean?**

a) The ability to inherit properties and methods from multiple classes
b) The ability to create objects from abstract classes
c) The ability to perform different actions based on the context
d) The ability to access private properties from outside the class

10. **Which OOP principle promotes code reusability by allowing a class to inherit properties and methods from another class?**

a) Encapsulation
b) Polymorphism
c) Inheritance
d) Abstraction