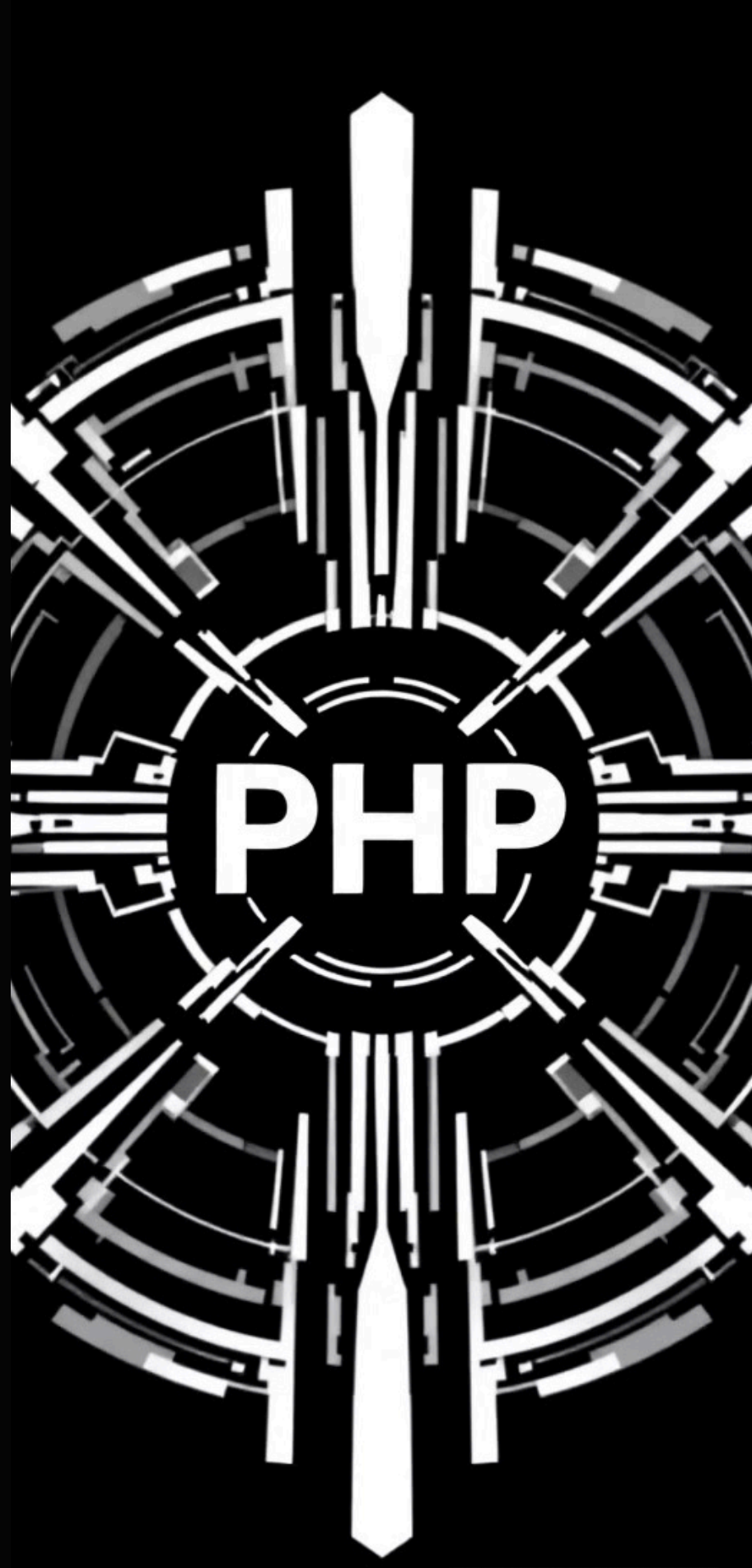
Introduction to PHP Classes and Objects

This presentation will introduce you to the fundamental concepts of classes and objects in PHP, a cornerstone of modern web development. We will delve into the core principles of Object-Oriented Programming (OOP) in PHP, exploring how classes provide blueprints for creating reusable and modular code. We will cover key concepts like class definition, object instantiation, and the powerful new feature introduced in PHP 7.0: anonymous classes. By the end of this presentation, you will be equipped with a solid understanding of how to leverage classes and objects effectively in your PHP projects.

```php
<?php
class MyClass {
    // Properties
    public $name;

    // Methods
    public function __construct($name) {
        $this->name = $name;
    }

    public function getName() {
        return $this->name;
    }
}
?>
```



by Shailesh Mishra

Defining a PHP Class

1 Class Syntax

Classes are defined using the `class` keyword, followed by the class name and curly braces. Inside, define class properties (data) and methods (functions).

```
<?php
class MyClass {
    // Public property
    public $name;

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

    // Getter method
    public function getName() {
        return $this->name;
    }
}
?>
```

2 Visibility Modifiers

Control the access level of class members. `public` members can be accessed from anywhere, `private` members only within the class, and `protected` members within the class and its subclasses. `$this` refers to the current object instance.

Creating Objects in PHP

1 Instantiating a Class

Creating an object from a class is called instantiation.

2 Using the `new` Keyword

It involves using the `new` keyword followed by the class name.

3 Accessing Properties and Methods

This creates a new object of that class, allowing you to access its properties and methods.

4 Example: Car Class

```
<?php
class Car {
  public $color;
  public $model;

  public function __construct($color, $model) {
    $this->color = $color;
    $this->model = $model;
  }

  public function startEngine() {
    echo "Engine started!";
  }
}

$myCar = new Car("red", "Sedan");
$myCar->startEngine();
?>
```

Constructor and Destructor Methods

__construct() Method

- Automatically called when an object is created.
- Used to initialize object properties.
- Sets up a new object.

```
<?php
class MyClass {
    public function __construct() {
        echo "Object created";
    }
}

$obj = new MyClass(); // Outputs: Object created
?>
```

__destruct() Method

- Called when an object is destroyed.
- Used to clean up resources.
- Ensures a tidy ending to the object's lifecycle.

```
<?php
class MyClass {
    public function __destruct() {
        echo "Object destroyed";
    }
}

$obj = new MyClass(); // Outputs: Object destroyed when the script ends
?>
```

Introduction to Anonymous Classes

1 Anonymous classes were introduced in PHP 7.0 as a powerful tool for creating classes without explicitly defining them with a name. They are useful for situations where you need a class for a specific task and don't want to create a separate named class.

2 Anonymous classes are declared within a single line of code, making them more concise and efficient for specific, one-time uses. This contrasts with traditional named classes that require a full block of code for definition.

3 Anonymous classes offer a way to create temporary and purpose-specific code structures without the overhead of defining named classes. They are particularly useful in situations where you want to create a class for a specific task and then discard it after that task is complete. This approach promotes code efficiency and avoids clutter.

Creating Anonymous Classes

Syntax for Anonymous Classes

Anonymous classes are defined using the 'new class' syntax, followed by curly braces. Inside the braces, you define the properties and methods, similar to a named class. An anonymous class is a class without a name, defined inline within your code.

```
<?php
$myAnonymousClass = new class {
    public function sayHello() {
        return "Hello from anonymous class!";
    }
};

echo $myAnonymousClass->sayHello(); // Outputs: Hello
from anonymous class!
?>
```

Example: Simple Logger

Let's create a simple anonymous class that represents a logger. This class will have a 'log' method that takes a message and prints it to the console. The class is assigned to a variable, allowing us to use its 'log' method. This illustrates the direct and efficient use of anonymous classes for specific functionalities.

```
<?php
$logger = new class {
    public function log(string $message) {
        echo date("Y-m-d H:i:s") . ": " . $message . PHP_EOL;
    }
};

$logger->log("This is a log message.");
?>
```

Features of Anonymous Classes

Extending Other Classes

Anonymous classes can extend existing named classes, inheriting their properties and methods. This makes them flexible and adaptable for specialized tasks that build upon existing structures.

Implementing Interfaces

Similar to named classes, anonymous classes can implement interfaces, providing the required methods specified by the interface. This allows you to define specific behaviors and interactions within your code.

Using Traits

Anonymous classes can also use traits, which allow you to group reusable code blocks and add them to multiple classes. This provides a mechanism to share functionality across different classes.

Accessing Outer Scope Variables

Anonymous classes have access to variables declared in the outer scope where they are defined. This enables them to leverage existing data and context for their functionalities.

Anonymous Classes vs. Named Classes

Performance Considerations

In general, anonymous classes have a slight performance advantage over named classes because they are defined at the point of use, reducing the overhead of class loading and instantiation. This can be a factor in performance-critical applications, though the difference is often minimal.

Readability and Maintainability

For larger, complex projects, named classes are often preferred for better readability and maintainability. Their clear definitions and separation from the main code flow contribute to a cleaner and easier-to-understand project structure.

Practical Examples of Anonymous Classes

Mocking Objects

1

In software testing, anonymous classes are widely used for creating mock objects that simulate the behavior of real objects. This allows you to test different scenarios without depending on external systems or data sources.

One-Time Use Implementations

2

When you need a class for a specific task and don't want to define a separate named class, anonymous classes are the perfect solution. They provide a concise and efficient way to create temporary code structures for specific functionalities.

Callback Functions with State

3

Anonymous classes can encapsulate data and methods, making them excellent for creating callback functions with state. This allows you to pass data and functionality to other parts of your code while maintaining a clear and organized structure.

Best Practices and Conclusion

While anonymous classes are a powerful tool, they should be used judiciously. Choose them for specific, one-time uses or when you need temporary functionality. For larger projects, prefer named classes for better readability and maintainability.

Always strive for clear and descriptive naming for your classes, as it enhances code readability and maintainability. Effective naming helps others understand your code, reducing confusion and errors.

Documentation is crucial for any code you write, whether it's for named classes or anonymous classes. Clearly document your code, especially when dealing with anonymous classes, to ensure understanding and clarity for future reference.

Object-Oriented Programming (OOP) continues to evolve in PHP, and anonymous classes are a testament to this. They provide a flexible and efficient way to work with classes, offering a more modern and expressive approach to coding.