Object-Oriented Programming (OOP) with PHP

Object-Oriented Programming is a programming paradigm that organizes software design around data, or objects, rather than functions and logic. PHP provides full support for OOP features.

Key Concepts of OOP:

1. **Classes** and **Objects**
2. **Methods**
3. **Inheritance**
4. **Constructors and Destructors**
5. **Self and Parent Keywords**
6. **Object Cloning**
7. **OOP with MySQL Database**

---

1. Creating Classes and Objects in PHP

A **class** is a blueprint for creating objects. An **object** is an instance of a class.

Example 1: Creating a Simple Class and Object

```
php
Copy code
// Define the class
class Car {
    // Properties
    public $color;
    public $brand;

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

    // Method
    public function displayDetails() {
        echo "This car is a " . $this->color . " " . $this->brand . ".";
    }
}

// Create an object (instance of the class)
$car = new Car("red", "Toyota");
$car->displayDetails();  // Output: This car is a red Toyota.
```

Explanation:

- **Class**: The `Car` class defines properties (`$color`, `$brand`) and a method (`displayDetails()`).
- **Object**: `$car` is an object of the class `Car`, created using the `new` keyword.
- **Constructor**: The `__construct()` method is a special method used to initialize objects with values.

---

2. Working with Methods

Methods are functions defined inside a class. They define the behaviors of objects.

Example 2: Methods in Classes

```
php
Copy code
class Circle {
    public $radius;

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

    // Method to calculate area
    public function calculateArea() {
        return pi() * pow($this->radius, 2);  // Area = π * r^2
    }
}

// Create an object and use the method
$circle = new Circle(5);
echo "Area of the circle: " . $circle->calculateArea();  // Output: Area of
the circle: 78.539816339745
```

Explanation:

- **Methods** like `calculateArea()` define behaviors for the object.
- You call a method using the `->` operator on an object, such as `$circle->calculateArea()`.

---

3. Inheritance

Inheritance allows a class to inherit properties and methods from another class. This helps in reusing code.

Example 3: Inheritance in PHP

```
php
Copy code
// Parent class
class Animal {
    public $name;

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

    public function speak() {
        return "Animal makes a sound.";
    }
}

// Child class inheriting from Animal
class Dog extends Animal {

    // Overriding the parent method
    public function speak() {
        return $this->name . " barks.";
    }
}

// Creating an object of the child class
$dog = new Dog("Buddy");
echo $dog->speak();  // Output: Buddy barks.
```

Explanation:

- **Parent class**: `Animal` has a property `$name` and a method `speak()`.
- **Child class**: `Dog` extends `Animal`, inheriting its properties and methods. The method `speak()` is overridden to provide a specific behavior for dogs.
- **Inheritance**: The `Dog` class inherits from the `Animal` class, allowing access to its properties and methods.

---

4. Constructors and Destructors

- **Constructors** are used to initialize objects when they are created.
- **Destructors** are used to clean up when an object is destroyed.

Example 4: Constructor and Destructor

```
php
Copy code
class Book {
    public $title;
```

```
    // Constructor
    public function __construct($title) {
        $this->title = $title;
        echo "Book '$this->title' has been created.<br>";
    }

    // Destructor
    public function __destruct() {
        echo "Book '$this->title' has been destroyed.<br>";
    }
}

// Creating and destroying an object
$book1 = new Book("OOP with PHP");
unset($book1);  // This calls the destructor
```

Explanation:

- **Constructor**: The `__construct()` method runs when the object is created.
- **Destructor**: The `__destruct()` method runs when the object is destroyed (e.g., using `unset()` or when the script ends).

---

5. Self and Parent Keywords

- **`self`**: Refers to the current class.
- **`parent`**: Refers to the parent class.

Example 5: Using `self` and `parent`

```
php
Copy code
class Animal {
    protected $name;

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

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

class Dog extends Animal {

    // Using the parent constructor
    public function __construct($name) {
        parent::__construct($name);  // Calls the constructor of the parent
class
    }
```

```
    // Using the self keyword to access static method
    public function printName() {
        echo self::getName();  // Calls getName() from this class or parent
    }
}

// Create a dog object
$dog = new Dog("Rex");
$dog->printName();  // Output: Rex
```

Explanation:

- **`parent::`**: Used to call methods or constructors from the parent class.
- **`self::`**: Used to access static methods and properties from the current class.

---

6. Object Cloning

In PHP, you can clone an object using the `clone` keyword. The cloned object gets its own copy of the properties, but not the references.

Example 6: Object Cloning

```
php
Copy code
class Person {
    public $name;

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

// Create an object
$person1 = new Person("Alice");

// Clone the object
$person2 = clone $person1;
$person2->name = "Bob";

echo $person1->name;  // Output: Alice
echo $person2->name;  // Output: Bob
```

Explanation:

- The `clone` keyword creates a copy of the object. The cloned object is independent of the original.

---

7. Object-Oriented Programming with MySQL

OOP and MySQL go hand-in-hand in modern PHP applications. By combining the two, you can interact with databases in an object-oriented way.

Example 7: OOP with MySQL (Basic CRUD Operations)

```
php
Copy code
class Database {
    private $host = 'localhost';
    private $username = 'root';
    private $password = '';
    private $dbname = 'test_db';
    private $conn;

    // Constructor to initialize database connection
    public function __construct() {
        $this->conn = new mysqli($this->host, $this->username, $this-
>password, $this->dbname);
        if ($this->conn->connect_error) {
            die("Connection failed: " . $this->conn->connect_error);
        }
    }

    // Method to execute a query
    public function query($sql) {
        return $this->conn->query($sql);
    }

    // Method to close the connection
    public function close() {
        $this->conn->close();
    }
}

class User {
    private $db;

    public function __construct() {
        $this->db = new Database();  // Create an instance of Database class
    }

    public function addUser($name, $email) {
        $sql = "INSERT INTO users (name, email) VALUES ('$name', '$email')";
        return $this->db->query($sql);
    }

    public function getUsers() {
        $sql = "SELECT * FROM users";
        $result = $this->db->query($sql);
        while ($row = $result->fetch_assoc()) {
            echo $row['name'] . " - " . $row['email'] . "<br>";
        }
    }
```

```
}

// Usage
$user = new User();
$user->addUser("John", "john@example.com");
$user->getUsers();
```

Explanation:

- **Database Class**: Manages the database connection.
- **User Class**: Handles user-related functionality (e.g., adding and retrieving users).
- **CRUD Operations**: In this case, we perform a simple **Create** operation to insert a user into the database.