DESARROLLO DE SOFTWARE DESDE EL PUNTO DE VISTA DEL SERVIDOR: Modelos y la POO

En diseño web, se utilizan varios modelos de diseño de sistemas para planificar y desarrollar sitios web efectivos y funcionales. Los tres modelos principales son:

1. Modelo de **Arquitectura**: Este modelo se centra en la estructura general del sistema web. Define cómo se organizarán y comunicarán los diferentes componentes del sitio web. Algunos ejemplos de arquitecturas comunes en diseño web son la arquitectura cliente-servidor y la arquitectura de tres capas.

   En resumen, el modelo de arquitectura en diseño web se trata de cómo se organizan y comunican los componentes clave, como el cliente, el servidor, la base de datos y las capas lógicas, para proporcionar una experiencia web eficiente, segura y escalable.

2. Modelo de **Componentes:** El modelo de componentes se enfoca en la descomposición del sistema web en partes más pequeñas y reutilizables. Estos componentes pueden incluir elementos de la interfaz de usuario, funcionalidades específicas o módulos de código. Los componentes pueden diseñarse de manera independiente y luego integrarse en el sistema más grande.

3. Modelo de **Interfaz:** El modelo de interfaz se concentra en la apariencia y la experiencia del usuario en el sitio web. Define cómo se presentará la información al usuario, cómo interactuará con el sitio y cómo se navegará por él.

Estos tres modelos son fundamentales en el diseño web y a menudo se interconectan. Por ejemplo, la arquitectura define cómo se comunicarán y organizarán los componentes, y la interfaz se basa en gran medida en la estructura de los componentes y la arquitectura subyacente. En conjunto, estos modelos ayudan a crear sitios web efectivos, funcionales y atractivos para los usuarios.

ESQUEMAS DE CADA UNO DE LOS MODELOS DE DISEÑO DE SISTEMAS EN DISEÑO WEB

Modelo de Arquitectura	Modelo de Componentes	Modelo de Interfaz
• Cliente • Servidor • Base de Datos • Capas Lógicas • Comunicaciones	• Componente 1 • Componente 2 • Componente 3 • ... • Componente N	• Diseño UI/UX • Elementos Visuales • Usabilidad

Estos esquemas simplificados **representan los aspectos clave de cada modelo en el diseño web**. Por supuesto, en la práctica, cada uno de estos modelos puede ser más complejo y detallado, dependiendo de los requisitos específicos del proyecto.

PROGRAMACIÓN ORIENTADA A OBJETOS

Las **características básicas** de la programación orientada a objetos (**POO**) son:

1. **Abstracción:** Permite modelar objetos del mundo real en el código, enfocándose en las características y comportamientos más relevantes y abstractos.
2. **Encapsulación:** Agrupa datos (atributos) y funciones (métodos) relacionadas en una sola entidad, ocultando los detalles internos y protegiendo los datos mediante el acceso controlado.
3. **Herencia:** Permite que una clase herede las propiedades y métodos de otra clase, lo que fomenta la reutilización de código y la creación de jerarquías de clases.
4. **Polimorfismo:** Permite que objetos de diferentes clases puedan responder de manera diferente a un mismo método, lo que facilita la interoperabilidad y la flexibilidad del código.
5. **Clases y Objetos:** Las clases son plantillas que definen la estructura y el comportamiento de los objetos. Los objetos son instancias concretas de una clase.

En resumen, la **POO** es un paradigma de programación que se basa en la creación de objetos que encapsulan datos y comportamientos relacionados.

CONCEPTOS BÁSICOS EN LA PROGRAMACIÓN ORIENTADA A OBJETOS (POO):

1. **Clase:**
   - Representa un tipo de objeto o entidad en el sistema.
   - Define las propiedades (atributos) y los comportamientos (métodos) que los objetos de esta clase tendrán.
2. **Objeto:**
   - Una instancia concreta de una clase.
   - Los objetos son entidades específicas que se crean a partir de una clase y pueden interactuar y llevar a cabo acciones definidas en la clase.
3. **Atributo (Propiedad):**
   - Representa una característica o estado de un objeto.
   - Los atributos son variables que almacenan datos específicos para cada objeto de una clase.
4. **Método:**
   - Representa un comportamiento o acción que un objeto de una clase puede llevar a cabo.
   - Los métodos son funciones que operan en los atributos y pueden interactuar con otros objetos.
5. **Herencia:**
   - Un mecanismo que permite que una clase (clase derivada o subclase) herede propiedades y métodos de otra clase (clase base o superclase).
   - Facilita la reutilización de código y la creación de jerarquías de clases.
6. **Polimorfismo:**
   - La capacidad de diferentes objetos de clases relacionadas para responder de manera diferente a un mismo método.
   - Permite que se utilice un método genérico con varios tipos de objetos sin conocer su tipo exacto.
7. **Abstracción:**
   - El proceso de simplificar conceptos complejos al enfocarse en los aspectos esenciales y omitir los detalles irrelevantes.
   - Las clases y objetos son ejemplos de abstracciones en la POO.
8. **Encapsulación:**
   - El ocultamiento de los detalles internos de una clase, permitiendo el acceso controlado a sus atributos y métodos.
   - Protege los datos y el comportamiento de una clase y evita modificaciones no autorizadas.
9. **Instanciación:**
   - El proceso de crear un objeto específico a partir de una clase.
   - Cada objeto creado se considera una instancia de esa clase particular.
10. **Clase Abstracta:**
    - Una clase que no puede ser instanciada directamente y generalmente se utiliza como una plantilla base para otras clases concretas.
    - Puede contener métodos abstractos que deben implementarse en las subclases.

Estas son algunas de las clases de objetos básicos en la programación orientada a objetos. Cada una de ellas desempeña un papel fundamental en la creación de estructuras de software más organizadas y modularizadas.

En la programación orientada a objetos (**POO**), las **CLASES y LOS OBJETOS** son conceptos fundamentales, pero tienen roles y características diferentes. Aquí te presento las similitudes y diferencias clave entre clases y objetos:

Similitudes:

1. **Abstracción:** Tanto las clases como los objetos son abstracciones de la realidad. Representan conceptos, entidades o elementos del mundo real dentro de un programa.
2. **Estructura:** Ambos tienen una estructura que define atributos (propiedades) y métodos (comportamientos).
3. **Definición en Código:** Tanto las clases como los objetos se definen en el código fuente de un programa. Las clases son plantillas para crear objetos.
4. **Herencia:** Tanto las clases como los objetos pueden estar involucrados en relaciones de herencia. Las clases pueden heredar de otras clases, y los objetos pueden ser instancias de una clase específica.

Diferencias:

1. **Clase:**
   - Es una plantilla o un plano que define la estructura y el comportamiento de los objetos.
   - No representa una entidad concreta, sino un tipo de entidad.
   - Puede contener atributos y métodos, pero no almacena datos reales ni ejecuta acciones concretas.
   - Se utiliza para crear múltiples objetos del mismo tipo.
2. **Objeto:**
   - Es una instancia concreta de una clase.
   - Representa una entidad real o un elemento específico del mundo real.
   - Almacena datos en sus atributos y puede realizar acciones a través de sus métodos.
   - Existe en tiempo de ejecución y ocupa memoria.
3. **Creación:**
   - Las clases se crean una sola vez en el código para definir el plano de los objetos.
   - Los objetos se crean en tiempo de ejecución y pueden existir en múltiples instancias.
4. **Uso:**
   - Las clases se utilizan como plantillas para crear objetos y definir la estructura y el comportamiento general.
   - Los objetos se utilizan para representar y manipular datos específicos o realizar acciones concretas en el programa.
5. **Ejemplo:**
   - Ejemplo de clase: Clase Coche que define la estructura y el comportamiento general de los automóviles.
   - Ejemplo de objeto: MiCoche que es una instancia concreta de la clase Coche, representando un automóvil en particular con sus características y estado.

En resumen, las clases son abstracciones que definen cómo deben ser los objetos, mientras que los objetos son instancias concretas de esas clases que almacenan datos y realizan acciones en tiempo de ejecución. Las clases son el plano o la plantilla, y los objetos son las entidades reales que interactúan en un programa orientado a objetos.

EJEMPLO:

Clase "Coche"	Objeto "MiCoche"
Atributos	***Atributos***
- Marca	- Marca: Toyota
- Modelo	- Modelo: Camry
- Color	- Color: Azul
- Velocidad Actual	- Velocidad Actual: 0
Métodos	**Métodos**
+ Arrancar()	+ Arrancar()
+ Acelerar()	+ Acelerar()
+ Frenar()	+ Frenar()
+ Apagar()	+ Apagar()

PROGRAMACIÓN ORIENTADA A OBJETOS DEL LADO DEL SERVIDOR

La programación orientada a objetos (POO) es un paradigma de programación que se basa en la organización de datos y funciones en unidades llamadas "objetos".

Estos objetos son instancias de clases, que definen la estructura y el comportamiento de los objetos.

1. Clases y Objetos: Creación de clases y objetos, instanciación de objetos. Las clases son plantillas o estructuras que definen la forma y el comportamiento de los objetos. Los objetos son instancias concretas de una clase.

Ejemplo: En este ejemplo, crearemos una clase llamada "Persona" con propiedades como nombre y edad, y un método para mostrar información sobre la persona:

```php
<?php
    // Definición de la clase Persona
    class Persona {
      // Propiedades (atributos) de la clase
      public $nombre;
      public $edad;

      // Método para mostrar información de la persona
      public function mostrarInformacion() {
        echo "Nombre: " . $this->nombre . "<br>";
        echo "Edad: " . $this->edad . "<br>";
      }
    }

    // Creación de objetos de la clase Persona
    $persona1 = new Persona();
    $persona1->nombre = "Juan";
    $persona1->edad = 30;

    $persona2 = new Persona();
    $persona2->nombre = "María";
    $persona2->edad = 25;

    // Llamada al método mostrarInformacion de los objetos
    echo "<h2>Persona 1:</h2>";
    $persona1->mostrarInformacion();

    echo "<h2>Persona 2:</h2>";
    $persona2->mostrarInformacion();
?>
```

En este ejemplo, los elementos básicos de la POO en PHP son:

1. **Clase**: La clase Persona define una plantilla para crear objetos que representan personas. Contiene propiedades (nombre y edad) y métodos (mostrarInformacion).
2. **Propiedades (Atributos)**: Las propiedades son variables que contienen datos asociados a un objeto. En este caso, $nombre y $edad son propiedades de la clase Persona.
3. **Métodos**: Los métodos son funciones asociadas a una clase que definen el comportamiento de los objetos. En este caso, mostrarInformacion es un método de la clase Persona que muestra el nombre y la edad de la persona.

4. **Objetos**: Los objetos son instancias de una clase. En este ejemplo, $persona1 y $persona2 son objetos de la clase Persona.

2. La encapsulación

Es el proceso de ocultar los detalles internos de una clase y proporcionar una interfaz pública para interactuar con ella. Esto se logra utilizando modificadores de acceso para controlar el acceso a las propiedades y métodos de la clase.

En este ejemplo, crearemos una clase llamada "CuentaBancaria" que tiene propiedades como saldo y métodos para depositar y retirar dinero. Utilizaremos la encapsulación para asegurarnos de que el saldo de la cuenta solo se pueda modificar a través de los métodos definidos en la clase, y no directamente desde fuera de la clase.

```php
<?php
// Definición de la clase CuentaBancaria
class CuentaBancaria {
    // Propiedades privadas para encapsulación
    private $saldo = 0;

    // Método para depositar dinero en la cuenta
    public function depositar($cantidad) {
        if ($cantidad > 0) {
            $this->saldo += $cantidad;
            echo "Se depositaron $cantidad euros.<br>";
        } else {
            echo "La cantidad a depositar debe ser mayor que 0.<br>";
        }
    }

    // Método para retirar dinero de la cuenta
    public function retirar($cantidad) {
        if ($cantidad > 0 && $cantidad <= $this->saldo) {
            $this->saldo -= $cantidad;
            echo "Se retiraron $cantidad euros.<br>";
        } else {
            echo "La cantidad a retirar debe ser mayor que 0 y menor o igual que el saldo disponible.<br>";
        }
    }

    // Método para obtener el saldo de la cuenta
    public function obtenerSaldo() {
        return $this->saldo;
    }
}

// Creación de un objeto de la clase CuentaBancaria
$cuenta = new CuentaBancaria();

// Intento de acceso directo a la propiedad privada $saldo
// Esto generará un error porque $saldo es privado y no puede ser accedido desde fuera de la clase
// echo $cuenta->saldo;

// Acceso controlado a la propiedad $saldo a través de los métodos definidos en la clase
$cuenta->depositar(100);
$cuenta->retirar(50);

// Intento de modificación directa del saldo
// Esto no tiene efecto ya que $saldo es privado y no puede ser accedido desde fuera de la clase
```

```
// $cuenta->saldo = 500;

// Obtener el saldo utilizando el método definido en la clase
echo "Saldo actual: " . $cuenta->obtenerSaldo() . " euros.<br>";
?>
```

En este ejemplo, la propiedad `$saldo` se declara como privada para encapsularla. Esto significa que no se puede acceder directamente a `$saldo` desde fuera de la clase `CuentaBancaria`. En su lugar, se proporcionan métodos públicos (`depositar`, `retirar` y `obtenerSaldo`) para interactuar con el saldo de la cuenta de manera controlada.

Esto asegura que el saldo solo se pueda modificar utilizando los métodos definidos en la clase, lo que garantiza una mayor seguridad y control sobre los datos.

3. La Herencia: Creación de clases derivadas (**subclases**) y herencia de atributos y métodos de una clase base (**superclase**).

La herencia permite crear una nueva clase basada en una clase existente, heredando sus atributos y métodos. Esto fomenta la reutilización de código y la creación de jerarquías de clases.

La herencia es un principio fundamental en la POO que permite que una clase adquiera propiedades y métodos de otra clase. En este ejemplo, crearemos una clase base llamada "FiguraGeometrica" y luego crearemos dos clases derivadas: "Rectangulo" y "Triangulo", que heredarán propiedades y métodos de la clase base.

```
<?php
    // Clase base: FiguraGeometrica
    class FiguraGeometrica {
      // Propiedades protegidas para encapsulación
      protected $base;
      protected $altura;

      // Constructor de la clase
      public function __construct($base, $altura) {
        $this->base = $base;
        $this->altura = $altura;
      }

      // Método para calcular el área de la figura
      public function calcularArea() {
        return 0; // Por defecto, el área es 0
      }
    }

    // Clase derivada: Rectangulo (hereda de FiguraGeometrica)
    class Rectangulo extends FiguraGeometrica {
      // Sobrescribe el método calcularArea para el rectángulo
      public function calcularArea() {
        return $this->base * $this->altura;
      }
    }

    // Clase derivada: Triangulo (hereda de FiguraGeometrica)
    class Triangulo extends FiguraGeometrica {
      // Sobrescribe el método calcularArea para el triángulo
      public function calcularArea() {
        return ($this->base * $this->altura) / 2;
      }
    }
```

```
    // Crear objetos de las clases derivadas y llamar a sus métodos
    $rectangulo = new Rectangulo(5, 4);
    echo "Área del rectángulo: " . $rectangulo->calcularArea() . "<br>";

    $triangulo = new Triangulo(5, 4);
    echo "Área del triángulo: " . $triangulo->calcularArea() . "<br>";
?>
```

En este ejemplo:

- Creamos una clase base llamada FiguraGeometrica que tiene propiedades base y altura, y un método calcularArea() que devuelve 0 por defecto.
- Luego, creamos dos clases derivadas: Rectangulo y Triangulo, que heredan de la clase base FiguraGeometrica.
- En las clases derivadas, sobrescribimos el método calcularArea() para calcular el área específica de cada figura geométrica: para un rectángulo, el área es base * altura, y para un triángulo, el área es (base * altura) / 2.
- Finalmente, creamos objetos de las clases derivadas (Rectangulo y Triangulo) y llamamos a sus métodos calcularArea() para obtener el área de cada figura geométrica.

La herencia permite reutilizar código y establecer relaciones entre clases, lo que facilita la creación de jerarquías de clases en la programación orientada a objetos.

4. Polimorfismo: El polimorfismo permite que objetos de diferentes clases respondan de manera diferente a la misma llamada de método. Esto se logra a través de la sobrecarga de métodos y la implementación de interfaces.

```
<?php
    // Clase base: Animal
    class Animal {
      // Método para que el animal hable
      public function hablar() {
        return "El animal emite un sonido indefinido.";
      }
    }

    // Clase derivada: Perro (hereda de Animal)
    class Perro extends Animal {
      // Sobrescribe el método hablar para que el perro ladre
      public function hablar() {
        return "El perro ladra: ¡Guau guau!";
      }
    }

    // Clase derivada: Gato (hereda de Animal)
    class Gato extends Animal {
      // Sobrescribe el método hablar para que el gato maúlle
      public function hablar() {
        return "El gato maúlla: ¡Miau miau!";
      }
    }

    // Función para hacer hablar a un animal
    function hacerHablar(Animal $animal) {
      echo $animal->hablar() . "<br>";
    }
```

```
    // Crear objetos de las clases derivadas y llamar a la función hacerHablar
    $perro = new Perro();
    hacerHablar($perro);

    $gato = new Gato();
    hacerHablar($gato);
?>
```

En este ejemplo:

- Creamos una clase base llamada Animal con un método hablar() que devuelve un sonido indefinido.
- Luego, creamos dos clases derivadas: Perro y Gato, que heredan de la clase base Animal.
- En las clases derivadas, sobrescribimos el método hablar() para que el perro ladre y el gato maúlle.
- Creamos una función llamada hacerHablar() que acepta un objeto de tipo Animal, y llamamos al método hablar() del animal pasado como argumento.
- Finalmente, creamos objetos de las clases derivadas (Perro y Gato) y los pasamos como argumentos a la función hacerHablar(). Como resultado, cada animal "habla" de acuerdo a su propia implementación del método hablar(), lo que demuestra el polimorfismo en acción.