El patrón de arquitectura MVC (Modelo-Vista-Controlador). Este es un concepto fundamental en el desarrollo de software, especialmente en aplicaciones web.

**Patrón de Arquitectura MVC (Modelo-Vista-Controlador)**

El MVC es un patrón de diseño de software que separa la lógica de la aplicación en tres componentes interconectados. Esto se hace para separar las representaciones internas de la información de las formas en que se presenta y acepta la información del usuario.

**Componentes del MVC**

1. **Modelo**:
   * Representa los datos y la lógica de negocio de la aplicación.
   * Gestiona el acceso a la información, generalmente interactuando con la base de datos.
   * Define las reglas para manipular los datos.
   * Es independiente de la interfaz de usuario.
2. **Vista**:
   * Es la representación visual de los datos.
   * Se encarga de cómo se muestran los datos al usuario.
   * Puede ser cualquier tipo de salida de representación de los datos, como una tabla o un diagrama.
   * Recibe los datos del modelo y los muestra al usuario.
3. **Controlador**:
   * Actúa como un intermediario entre el Modelo y la Vista.
   * Recibe la entrada del usuario a través de la Vista.
   * Procesa las solicitudes del usuario (como hacer clic en un botón).
   * Recupera datos del Modelo y los pasa a la Vista para su visualización.

**Flujo de trabajo del MVC**

1. El usuario interactúa con la interfaz de usuario (Vista) de alguna manera (por ejemplo, haciendo clic en un botón).
2. El Controlador recibe la notificación de la acción solicitada por el usuario.
3. El Controlador accede al Modelo, posiblemente actualizándolo de acuerdo con la acción del usuario.
4. El Controlador delega a los objetos de la Vista la tarea de desplegar la interfaz de usuario.
5. La Vista obtiene los datos del Modelo para generar la interfaz apropiada para el usuario.
6. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

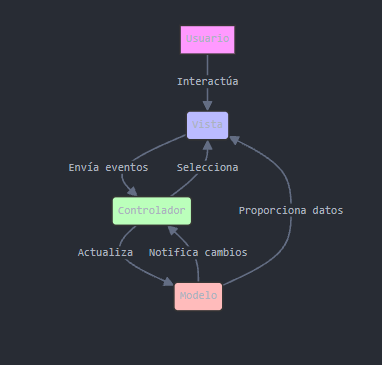
**Ventajas del MVC**

* **Separación de Conceptos**: Divide la lógica de negocio, la lógica de control y la presentación.
* **Reutilización de Código**: Permite reutilizar el código y desarrollar componentes en paralelo.
* **Fácil de Modificar**: Cambios en una parte no afectan necesariamente a las otras.
* **Mantenibilidad**: Facilita el mantenimiento y la escalabilidad del software.
* **Pruebas Unitarias**: Facilita la realización de pruebas unitarias de cada componente.

**Desventajas del MVC**

* **Complejidad**: Puede ser excesivo para aplicaciones pequeñas y simples.
* **Curva de Aprendizaje**: Requiere tiempo para comprender y aplicar correctamente.
* **Exceso de Archivos**: Puede resultar en un mayor número de archivos en comparación con otros enfoques.

El MVC es ampliamente utilizado en frameworks web como Laravel (PHP), Ruby on Rails (Ruby), ASP.NET MVC (C#), y muchos otros.



Este diagrama muestra el flujo típico en una aplicación MVC:

1. El usuario interactúa con la Vista.
2. La Vista envía los eventos de usuario al Controlador.
3. El Controlador actualiza el Modelo según sea necesario.
4. El Modelo notifica al Controlador sobre los cambios.
5. El Controlador selecciona la Vista apropiada.
6. El Modelo proporciona datos a la Vista para su representación.

En el contexto de nuestras aplicaciones de lista de tareas:

* **Modelo**: En la versión OOP, sería la clase TodoItem. En la versión estructurada, serían las funciones en functions.php que interactúan con la base de datos.
* **Vista**: Los archivos PHP en la carpeta views en ambas versiones.
* **Controlador**: En la versión OOP, sería la clase TodoController. En la versión estructurada, gran parte de la lógica del controlador está en index.php.

El MVC ayuda a organizar el código de manera que sea más fácil de entender, mantener y escalar. Es especialmente útil en aplicaciones web complejas donde la separación de la lógica de negocio, la presentación y el control de flujo es crucial para un desarrollo y mantenimiento eficientes.

**Programación Orientada a Objetos (POO)**

La Programación Orientada a Objetos es un paradigma de programación basado en el concepto de "objetos", que pueden contener datos en forma de campos (a menudo conocidos como atributos o propiedades) y código en forma de procedimientos (a menudo conocidos como métodos).

**Conceptos Fundamentales de la POO**

1. **Clase**:
   * Es una plantilla o blueprint para crear objetos.
   * Define las propiedades y métodos que tendrán los objetos de esa clase.
2. **Objeto**:
   * Es una instancia de una clase.
   * Representa una entidad específica con sus propias propiedades y métodos.
3. **Atributos (Propiedades)**:
   * Son las características o estados de un objeto.
   * Definen qué información contiene el objeto.
4. **Métodos**:
   * Son las acciones que un objeto puede realizar.
   * Definen el comportamiento de los objetos de una clase.
5. **Encapsulación**:
   * Oculta los detalles internos de cómo funciona un objeto.
   * Proporciona una interfaz pública para interactuar con el objeto.
6. **Herencia**:
   * Permite que una clase (subclase) herede atributos y métodos de otra clase (superclase).
   * Facilita la reutilización de código y la creación de jerarquías de clases.
7. **Polimorfismo**:
   * Permite que objetos de diferentes clases respondan al mismo método de diferentes maneras.
   * Proporciona una interfaz única para entidades de diferentes tipos.
8. **Abstracción**:
   * Simplifica sistemas complejos ocultando detalles innecesarios.
   * Se centra en las características esenciales de un objeto.

**Ventajas de la POO**

* **Modularidad**: El código se organiza en unidades lógicas (objetos), facilitando el desarrollo y mantenimiento.
* **Reutilización**: A través de la herencia y la composición, se puede reutilizar código eficientemente.
* **Flexibilidad y Escalabilidad**: Es más fácil modificar y extender el código existente.
* **Intuitivo**: Modela el mundo real de una manera más natural, lo que facilita el diseño de software.

**Desventajas de la POO**

* **Curva de Aprendizaje**: Puede ser más difícil de aprender inicialmente que la programación procedural.
* **Tamaño del Programa**: Los programas OOP pueden ser más grandes que los equivalentes procedurales.
* **Velocidad**: En algunos casos, puede ser ligeramente más lento que el código procedural puro.

La POO es ampliamente utilizada en muchos lenguajes de programación populares como Java, C++, Python, PHP, y C#, entre otros.

<?php// Definición de una clase

class Coche {

    // Atributos

    private $marca;

    private $modelo;

    private $color;

    // Constructor

    public function \_\_construct($marca, $modelo, $color) {

        $this->marca = $marca;

        $this->modelo = $modelo;

        $this->color = $color;

    }

    // Métodos

    public function arrancar() {

        return "El {$this->marca} {$this->modelo} está arrancando.";

    }

    public function frenar() {

        return "El {$this->marca} {$this->modelo} está frenando.";

    }

    // Getter

    public function getColor() {

        return $this->color;

    }

    // Setter

    public function setColor($nuevoColor) {

        $this->color = $nuevoColor;

    }

}

// Creación de un objeto (instancia de la clase)

$miCoche = new Coche("Toyota", "Corolla", "Rojo");

// Uso de métodos

echo $miCoche->arrancar();  // Salida: El Toyota Corolla está arrancando.

echo $miCoche->frenar();    // Salida: El Toyota Corolla está frenando.

// Uso de getter y setter

echo $miCoche->getColor();  // Salida: Rojo

$miCoche->setColor("Azul");

echo $miCoche->getColor();  // Salida: Azul

// Ejemplo de herencia

class CocheElectrico extends Coche {

    private $capacidadBateria;

    public function \_\_construct($marca, $modelo, $color, $capacidadBateria) {

        parent::\_\_construct($marca, $modelo, $color);

        $this->capacidadBateria = $capacidadBateria;

    }

    public function cargar() {

        return "El {$this->marca} {$this->modelo} está cargando su batería de {$this->capacidadBateria} kWh.";

    }

}

$miCocheElectrico = new CocheElectrico("Tesla", "Model 3", "Blanco", 75);

echo $miCocheElectrico->arrancar();  // Hereda el método de Coche

echo $miCocheElectrico->cargar();    // Método específico de CocheElectrico

En estos ejemplos:

1. Definimos una clase Coche con atributos y métodos.
2. Usamos encapsulación al hacer los atributos privados y proporcionar métodos públicos para acceder a ellos.
3. Creamos un objeto $miCoche como instancia de la clase Coche.
4. Demostramos el uso de getters y setters para acceder y modificar atributos privados.
5. Implementamos herencia con la clase CocheElectrico que extiende de Coche.
6. Mostramos polimorfismo, ya que CocheElectrico puede usar métodos de Coche pero también tiene sus propios métodos específicos.

La POO permite modelar sistemas complejos de una manera más intuitiva y modular, lo que facilita el desarrollo, mantenimiento y escalabilidad de aplicaciones de software grandes y complejas.

**Programación Estructurada**

La Programación Estructurada es un paradigma de programación que tiene como objetivo mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora, utilizando únicamente subrutinas, estructuras de bloque y estructuras de control.

**Conceptos Fundamentales de la Programación Estructurada**

1. **Secuencia**:
   * Ejecución de instrucciones una tras otra.
2. **Selección (Condicional)**:
   * Ejecución de un bloque de código u otro dependiendo de una condición.
   * Incluye estructuras como if-else, switch-case.
3. **Iteración (Bucles)**:
   * Repetición de un bloque de código mientras se cumpla una condición.
   * Incluye estructuras como for, while, do-while.
4. **Subrutinas (Funciones o Procedimientos)**:
   * Bloques de código que realizan tareas específicas y pueden ser llamados desde diferentes partes del programa.

**Principios de la Programación Estructurada**

1. **Diseño Top-Down**:
   * Descomponer un problema grande en problemas más pequeños y manejables.
2. **Modularidad**:
   * Dividir el programa en módulos o funciones independientes.
3. **Estructuras de Control Limitadas**:
   * Uso de solo tres tipos de estructuras de control: secuencia, selección e iteración.
4. **Alcance Limitado**:
   * Limitar el alcance de las variables para reducir errores y efectos secundarios no deseados.

**Ventajas de la Programación Estructurada**

* **Claridad**: El código es más fácil de leer y entender.
* **Mantenibilidad**: Facilita la detección y corrección de errores.
* **Reutilización de Código**: Las funciones pueden ser reutilizadas en diferentes partes del programa.
* **Eficiencia en el Desarrollo**: Permite desarrollar programas de manera más rápida y sistemática.

**Desventajas de la Programación Estructurada**

* **Limitaciones en Modelado**: Puede ser menos intuitiva para modelar ciertos problemas del mundo real en comparación con la POO.
* **Dificultad con Programas Muy Grandes**: En programas extremadamente grandes, puede volverse difícil de manejar sin conceptos adicionales como los que ofrece la POO.
* **Menor Encapsulación**: No ofrece el mismo nivel de encapsulación que la POO.

La Programación Estructurada sentó las bases para muchos conceptos modernos de programación y sigue siendo fundamental en la enseñanza de la programación y en el desarrollo de software.

<?php

// Ejemplo de un programa estructurado para calcular el factorial de un número

// Función para calcular el factorial

function factorial($n) {

    if ($n <= 1) {

        return 1;

    } else {

        return $n \* factorial($n - 1);

    }

}

// Función principal del programa

function main() {

    echo "Calculadora de Factorial\n";

    // Bucle para permitir múltiples cálculos

    while (true) {

        // Entrada del usuario

        echo "Ingrese un número (o 'q' para salir): ";

        $input = trim(fgets(STDIN));

        // Estructura de control para salir del programa

        if ($input === 'q') {

            echo "Gracias por usar la calculadora. ¡Hasta luego!\n";

            break;

        }

        // Convertir la entrada a un número entero

        $number = intval($input);

        // Estructura de control para validar la entrada

        if ($number < 0) {

            echo "Por favor, ingrese un número no negativo.\n";

            continue;

        }

        // Cálculo y salida del resultado

        $result = factorial($number);

        echo "El factorial de $number es $result\n";

    }

}

// Llamada a la función principal

main();

?>

Ejemplo de Programación Estructurada:

1. **Secuencia**: El código se ejecuta línea por línea en un orden específico.
2. **Selección**: Usamos estructuras if-else para manejar diferentes casos (salir del programa, validar entrada).
3. **Iteración**: Utilizamos un bucle while para permitir múltiples cálculos.
4. **Subrutinas**: Definimos dos funciones: factorial() para el cálculo y main() como punto de entrada del programa.
5. **Diseño Top-Down**: El problema se descompone en funciones más pequeñas y manejables.
6. **Modularidad**: Las funciones factorial() y main() son módulos independientes que realizan tareas específicas.

Este programa demuestra los principios clave de la Programación Estructurada:

* Usa solo las estructuras de control básicas (secuencia, selección, iteración).
* Divide el problema en funciones más pequeñas y manejables.
* Tiene un flujo de control claro y fácil de seguir.
* Utiliza variables con alcance limitado dentro de las funciones.

La Programación Estructurada proporciona una base sólida para escribir código limpio y bien organizado, y muchos de sus principios siguen siendo relevantes incluso en paradigmas más modernos como la Programación Orientada a Objetos.

Ejercicios propuestos

**1. Sistema de Gestión de Inventario**

**Descripción:** Un sistema simple para gestionar el inventario de una tienda pequeña.

**Funcionalidades:**

* Agregar nuevos productos al inventario
* Actualizar la cantidad de productos existentes
* Buscar productos por nombre o categoría
* Mostrar una lista de productos con bajo stock
* Generar un informe simple de inventario

**Conceptos a aplicar:**

* POO: Clases para Producto, Inventario
* Estructurada: Funciones para buscar, actualizar y generar informes

**2. Gestor de Tareas Personal**

**Descripción:** Una aplicación para gestionar tareas personales o de un equipo pequeño.

**Funcionalidades:**

* Crear nuevas tareas con título, descripción y fecha límite
* Marcar tareas como completadas
* Categorizar tareas (trabajo, personal, etc.)
* Listar tareas por estado o categoría
* Establecer recordatorios para tareas próximas a vencer

**Conceptos a aplicar:**

* POO: Clases para Tarea, Lista de Tareas
* Estructurada: Funciones para filtrar y mostrar tareas

**3. Sistema de Reservas para un Restaurante**

**Descripción:** Un sistema básico para gestionar reservas en un restaurante pequeño.

**Funcionalidades:**

* Crear una nueva reserva (fecha, hora, número de personas)
* Verificar disponibilidad de mesas
* Modificar o cancelar reservas existentes
* Ver lista de reservas para un día específico
* Gestionar mesas disponibles

**Conceptos a aplicar:**

* POO: Clases para Reserva, Mesa, Restaurante
* Estructurada: Funciones para verificar disponibilidad y generar listas

**4. Juego de Preguntas y Respuestas**

**Descripción:** Un juego simple de preguntas y respuestas con diferentes categorías.

**Funcionalidades:**

* Agregar nuevas preguntas y respuestas
* Seleccionar categoría de preguntas
* Jugar una ronda de preguntas
* Llevar un registro de puntuación
* Mostrar estadísticas del jugador

**Conceptos a aplicar:**

* POO: Clases para Pregunta, Categoría, Jugador
* Estructurada: Funciones para seleccionar preguntas aleatorias y calcular puntuaciones

**5. Conversor de Unidades**

**Descripción:** Una aplicación que permite convertir entre diferentes unidades de medida.

**Funcionalidades:**

* Convertir entre diferentes unidades de longitud (metros, pies, pulgadas, etc.)
* Convertir entre diferentes unidades de peso (kilogramos, libras, onzas, etc.)
* Convertir entre diferentes unidades de temperatura (Celsius, Fahrenheit, Kelvin)
* Agregar nuevas unidades de conversión
* Guardar conversiones frecuentes

**Conceptos a aplicar:**

* POO: Clases para diferentes tipos de unidades (Longitud, Peso, Temperatura)
* Estructurada: Funciones para realizar cálculos de conversión y mostrar resultados