Parcial #2: Clases, arreglos y funciones predefinidas

Objetivo: Desarrollar un sistema dinámico de gestión de blog utilizando PHP, aplicando conceptos de programación orientada a objetos, incluyendo herencia de clases, interfaces, manejo de arreglos, y uso de funciones predefinidas.

Instrucciones:

1. Creen una interfaz llamada Detalle con el siguiente método:

   obtenerDetallesEspecificos(): string

2. Implementen las siguientes clases en el archivo clases.php:

   a. Clase abstracta Entrada que implemente la interfaz Detalle, con los siguientes atributos comunes:

      1. id (entero)
      2. fecha_creacion (cadena en formato 'YYYY-MM-DD')
      3. tipo (entero: 1, 2 o 3, representando el número de columnas)

   b. Clase EntradaUnaColumna que herede de Entrada, con los atributos adicionales:

      1. titulo (cadena)
      2. descripcion (cadena)

   c. Clase EntradaDosColumnas que herede de Entrada, con los atributos adicionales:

      1. titulo1 (cadena)
      2. descripcion1 (cadena)
      3. titulo2 (cadena)
      4. descripcion2 (cadena)

   d. Clase EntradaTresColumnas que herede de Entrada, con los atributos adicionales:

      1. titulo1 (cadena)
      2. descripcion1 (cadena)
      3. titulo2 (cadena)
      4. descripcion2 (cadena)
      5. titulo3 (cadena)

6. descripcion3 (cadena)

3. Implementen el método obtenerDetallesEspecificos() en cada clase hija de Entrada:
    1. Para EntradaUnaColumna: Retornar "Entrada de una columna: [titulo]"
    2. Para EntradaDosColumnas: Retornar "Entrada de dos columnas: [titulo1] | [titulo2]"
    3. Para EntradaTresColumnas: Retornar "Entrada de tres columnas: [titulo1] | [titulo2] | [titulo3]"
4. En la clase GestorBlog, implementen los siguientes métodos:
    1. agregarEntrada(Entrada $entrada): Agrega una nueva entrada.
    2. editarEntrada(Entrada $entrada): Actualiza una entrada existente.
    3. eliminarEntrada($id): Elimina una entrada por su ID.
    4. obtenerEntrada($id): Obtiene una entrada específica por su ID.
    5. moverEntrada($id, $direccion): Mueve una entrada hacia arriba o abajo en la lista.
5. En el archivo index.php, implementen la lógica para manejar las diferentes acciones en el switch statement:
    1. 'add': Agrega una nueva entrada al blog según el tipo seleccionado.
    2. 'edit': Modifica una entrada existente.
    3. 'delete': Elimina una entrada.
    4. 'move_up' y 'move_down': Reordena las entradas.
6. En el archivo index.php, reemplacen la impresión de la propiedad titulo, por el resultado de la ejecución del método obtenerDetallesEspecificos().
7. Implementen un manejo básico de errores para parámetros inválidos o acciones no reconocidas.

Restricciones:

1. Todo el manejo de datos persistentes debe hacerse a través del archivo JSON blog.json.
2. El código debe estar bien comentado y seguir las mejores prácticas de programación orientada a objetos.

Entregables:

1. Archivo clases.php con la implementación de todas las clases e interfaces requeridas.
2. Archivo index.php con la lógica de procesamiento de acciones en el switch statement.
3. Archivo blog.json para almacenar los datos de las entradas.

Criterios de Evaluación:

1. Correcta implementación de la interfaz, clases abstractas y concretas (30%)
2. Funcionalidad completa del gestor de blog (listar, eliminar, reordenar, ver) (35%)
3. Correcta persistencia y recuperación de datos usando JSON (20%)
4. Manejo adecuado de errores (10%)
5. Código limpio, bien comentado y que siga las mejores prácticas de POO (5%)

Tiempo máximo de desarrollo: 2 horas

Notas adicionales:

1. Se proporciona un código base en index.php que incluye la estructura HTML, manejo de POST y JavaScript necesario para el diseño Masonry. No modifiquen esta estructura, solo implementen la lógica PHP necesaria en el switch statement.
2. El archivo blog.json debe ser creado y gestionado por su implementación.
3. Asegúrense de que la aplicación maneje correctamente los diferentes tipos de entradas (1, 2 y 3 columnas) en la visualización y operaciones CRUD.