**Informe sobre Clases y Métodos y su funcionalidad**

**Funcionalidades Utilizadas en el proyecto:**

1. **Interfaz Gráfica (GUI):**
   * Construcción de la interfaz de usuario mediante componentes Swing (JFrame, **JPanel**, **JLabel**, **JButton**, etc.).
   * Distribución de los elementos en la interfaz utilizando diferentes layouts (**BorderLayout**, **FlowLayout**, **GridLayout**, **GridBagLayout**).
2. **Manipulación de Eventos:**
   * Gestión de eventos de ratón y acciones del usuario (**MouseEvent**, **MouseAdapter**, **ActionListener**).
   * Ejecución de acciones específicas en respuesta a eventos del usuario (clics de botón, interacciones con la interfaz).
3. **Listas y Modelos:**
   * Utilización de **JList**, **DefaultListModel**, **DefaultListCellRenderer** para mostrar y gestionar listas de elementos.
   * Personalización de la representación de elementos en la lista (**DefaultListCellRenderer**).
4. **Manipulación de Imágenes:**
   * Carga, lectura y manipulación de imágenes utilizando **ImageIO**, **BufferedImage**.
   * Posible visualización o manipulación de imágenes en la interfaz gráfica.
5. **Temporización y Tareas Repetitivas:**
   * Utilización del **Timer** para ejecutar tareas repetitivas o temporizadas.
6. **Operaciones de Entrada/Salida (I/O):**
   * Manipulación de archivos y operaciones de entrada/salida (**java.io.\*)** para lectura o escritura de datos.

**Bibliotecas y Clases Importantes:**

1. **Bibliotecas Estándar:**
   * **java.awt.\*, javax.swing.\*:** Para la construcción de la interfaz gráfica y gestión de eventos.
   * **java.util.\*:** Para utilidades como colecciones, temporizadores y mapas.
   * **java.io.\*, java.net.\*:** Para operaciones de entrada/salida y manipulación de recursos de red.
   * **javax.imageio.ImageIO**: Para lectura/escritura de imágenes.
2. **Clases y Componentes Importantes:**
   * Componentes Swing (**JFrame**, **JPanel**, **JLabel**, **JButton**, **JList**, etc.): Para la construcción de la interfaz.
   * Clases de gestión de eventos (**MouseEvent**, **ActionListener**): Para manejar la interacción del usuario.
   * Utilidades para la manipulación de imágenes (**BufferedImage**, **ImageIO**): Para trabajar con imágenes.

**Clase ChessBoardGUI:**

Esta clase representa la interfaz gráfica del tablero de ajedrez y contiene métodos para gestionar las interacciones del usuario y las piezas en el tablero.

* **Atributos:**
  + **marco**: Representa el marco principal de la aplicación.
  + **panelTablero**: Panel que contiene las casillas del tablero de ajedrez.
  + **listaMovimientosModelo**: Modelo para la lista de movimientos en la interfaz.
  + **listaMovimientos**: Lista que muestra el historial de movimientos.
  + **turnoBlancas**: Variable que indica el turno actual (blancas o negras).
  + **TAMANIO\_TABLERO**: Tamaño del tablero (8x8).
  + **TAMANIO\_CASILLA**: Tamaño de cada casilla del tablero.
  + **posicionesPiezas**: Mapa que almacena las posiciones de las piezas en el tablero.
  + **casillaSeleccionada**: Casilla seleccionada por el usuario para realizar un movimiento.
  + **etiquetaTurno**: Etiqueta que muestra el turno actual (blancas o negras).
  + **grupoColores**, **blancoButton**, **grisButton**: Componentes relacionados con la selección del color del tablero.
* **Métodos Principales:**
  + **cambiarColorTablero(**Color **colorCasillasBlancas,** Color **colorCasillasNegras)**: Cambia el color del tablero según los colores especificados.
  + **configurarInterfaz()**: Configura la interfaz gráfica del tablero de ajedrez.
  + **inicializarTablero()**: Inicializa el tablero y coloca las piezas en sus posiciones iniciales.
  + **actualizarLabelTurno()**: Actualiza la etiqueta del turno de las piezas en la interfaz gráfica.
  + **colocarPiezasIniciales()**: Coloca las piezas iniciales en el tablero de ajedrez.
  + **moverPieza(JPanel desdeCasilla, JPanel aCasilla)**: Mueve una pieza desde una casilla de origen hasta una casilla de destino.
  + **cambiarTurno()**: Cambia el turno del jugador después de un movimiento válido.
  + **obtenerNombreCasilla(JPanel casilla)**: Obtiene el nombre de una casilla en formato de notación algebraica (por ejemplo, "a1", "e5").
  + **esMovimientoValido(JPanel desdeCasilla, JPanel aCasilla)**: Verifica la validez de un movimiento desde una casilla hasta otra.

**Clase EscuchaClicPieza (Clase interna en ChessBoardGUI):**

Esta clase implementa la interfaz **MouseAdapter** para manejar los clics del mouse en las casillas del tablero.

* **Métodos:**
  + **mouseClicked(MouseEvent e)**: Método que se ejecuta cuando se hace clic en una casilla del tablero. Gestiona las interacciones del usuario con las piezas y los movimientos.

En resumen, la clase **ChessBoardGUI** representa la interfaz gráfica del tablero de ajedrez y proporciona métodos para realizar movimientos válidos, cambiar el turno de los jugadores y gestionar la visualización de la partida. La clase **EscuchaClicPieza** se encarga de manejar los clics del mouse en las casillas del tablero para permitir que el usuario interactúe con las piezas.

Los métodos **esCaminoVerticalLibre**, **esCaminoHorizontalLibre**, **esCaminoDiagonalLibre** y **obtenerCasilla**: son esenciales para el funcionamiento del juego de ajedrez, ya que se utilizan para verificar la legalidad de los movimientos de las piezas en el tablero.

**esCaminoVerticalLibre(int filaDesde, int columnaDesde, int filaHasta, int columnaHasta):**

Este método verifica si el camino vertical entre dos casillas está libre de piezas. Toma las coordenadas de fila y columna de las casillas de origen (**filaDesde**, **columnaDesde**) y destino (**filaHasta**, **columnaHasta**). Determina el rango vertical entre estas filas y luego itera a través del rango, verificando si hay alguna pieza en las casillas intermedias. Si encuentra alguna pieza en el camino vertical, devuelve false; de lo contrario, devuelve true, indicando que el camino está libre de piezas.

**esCaminoHorizontalLibre(int filaDesde, int columnaDesde, int filaHasta, int columnaHasta):**

Similar al método **esCaminoVerticalLibre**, este método verifica si el camino horizontal entre dos casillas está libre de piezas. Toma las coordenadas de fila y columna de las casillas de origen y destino, determina el rango horizontal entre estas columnas y luego itera a través del rango, verificando si hay alguna pieza en las casillas intermedias. Devuelve false si encuentra alguna pieza en el camino horizontal; de lo contrario, devuelve true.

**esCaminoDiagonalLibre(int filaDesde, int columnaDesde, int filaHasta, int columnaHasta):**

Este método verifica si el camino diagonal entre dos casillas está libre de piezas. Toma las coordenadas de fila y columna de las casillas de origen y destino. Calcula la diferencia en las filas entre las casillas de origen y destino, determina el incremento para las filas y columnas en la dirección diagonal, y luego itera a través del camino diagonal. Durante la iteración, verifica si

hay alguna pieza en las casillas intermedias. Si encuentra alguna pieza en el camino diagonal, devuelve false; de lo contrario, devuelve true.

**obtenerCasilla(int fila, int columna):**

Este método toma las coordenadas de fila y columna y devuelve el panel (casilla) correspondiente en la posición especificada por esas coordenadas en el tablero. Utiliza la fórmula para calcular el índice del array que representa el tablero y devuelve el JPanel en esa posición.

En este proyecto también se está importando las clases **EditorTexto** y **Ventana** desde otros paquetes (**editorTexto** y **videoReproductor**, respectivamente) con intención de trabajar con esta faceta de la codificación en Java.

**Sobre la importación de clases externas dentro de otros package en un proyecto**

Esta práctica es beneficiosa por varias razones:

1. **Reutilización de código:** Al importar clases externas, puedes reutilizar funcionalidades ya implementadas en esas clases. Por ejemplo, si **EditorTexto** tiene métodos para manejar texto y **Ventana** ofrece funcionalidades de visualización de video, puedes utilizar esas capacidades sin tener que volver a escribir el código.
2. **Organización:** Las importaciones claras y bien organizadas facilitan la comprensión del código. Los desarrolladores podemos ver rápidamente qué clases externas se están utilizando, lo que mejora la legibilidad y la comprensión del proyecto.
3. **Modularidad:** Al separar la lógica en diferentes clases o paquetes, pues promueve la modularidad y la división del código en componentes más manejables. Esto ayuda a mantener un código más limpio y estructurado.

Es importante tener en cuenta:

* **Gestión de dependencias:** Si las clases importadas dependen de otras clases o bibliotecas externas, es esencial gestionar esas dependencias para asegurarte de que el proyecto pueda acceder a ellas correctamente.
* **Nomenclatura clara y consistente:** Es buena práctica nombrar las importaciones de manera clara y mantener una convención consistente en todo el código para mejorar la comprensión.

En definitiva, este proyecto se centra en la construcción de una interfaz gráfica interactiva utilizando Java Swing y AWT, con características como manipulación de imágenes, gestión de eventos y manipulación de texto. También parece incluir funcionalidades específicas o extensiones a través de bibliotecas propias o externas (**editorTexto** y **videoReproductor**) para manejar tareas específicas de edición de texto y reproducción de video, respectivamente.