Manual Técnico de la Aplicación de Hoja de Cálculo

1. Introducción

Este manual técnico proporciona una descripción detallada de la arquitectura, diseño y funcionamiento interno de la aplicación de hoja de cálculo desarrollada en Java. Está dirigido a desarrolladores, administradores de sistemas y cualquier persona interesada en comprender cómo funciona el software a un nivel más profundo.

1.1 Propósito

El propósito de este documento es:

- Describir la estructura del código y las responsabilidades de cada clase.
- Explicar la lógica de la interacción entre los componentes.
- Detallar la implementación de funcionalidades clave como el manejo de fórmulas y la gestión de la tabla hash.
- Servir como referencia para futuras mejoras, mantenimiento y depuración del sistema.

1.2 Arquitectura General

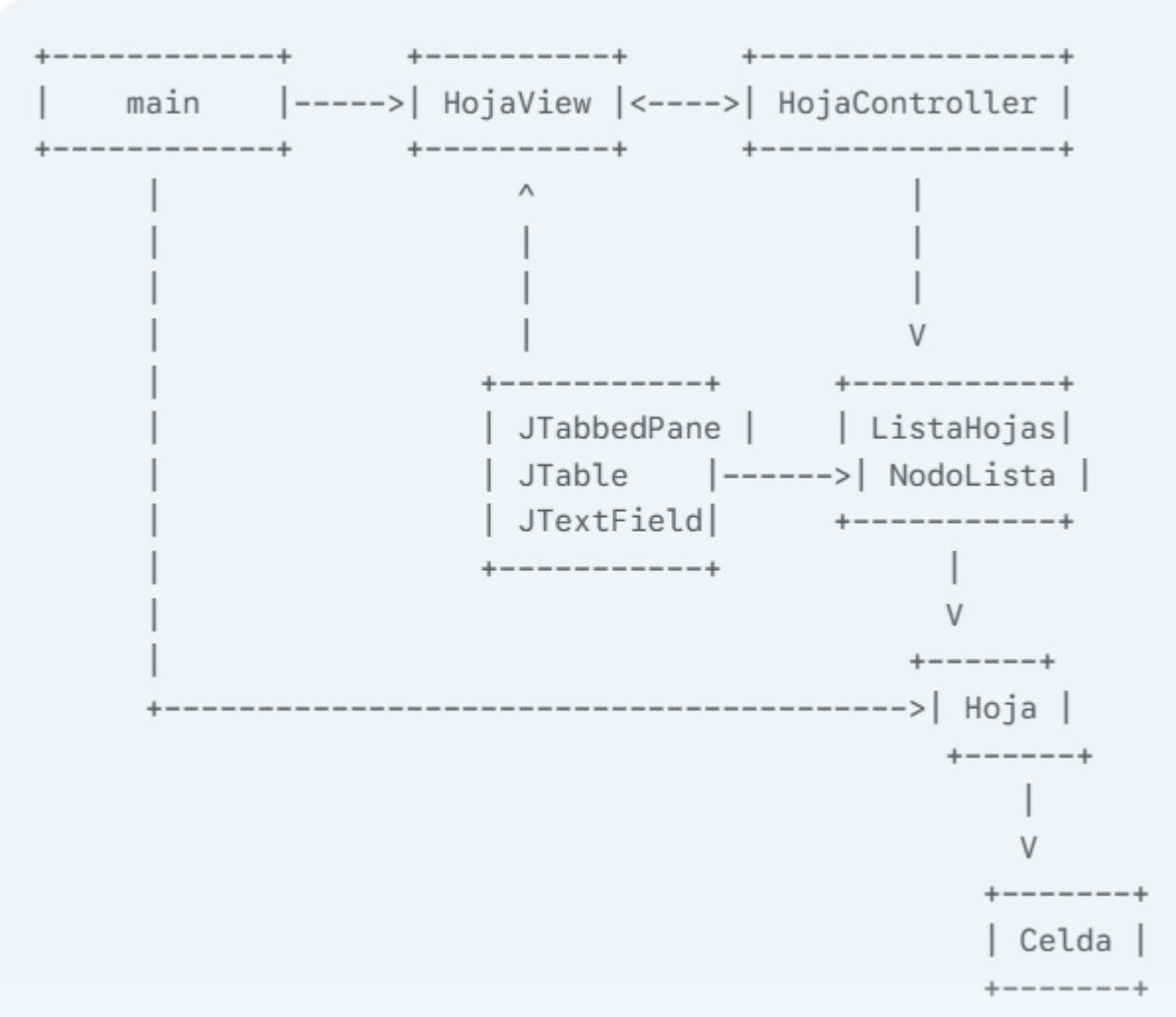
La aplicación sigue un patrón de diseño **Modelo-Vista-Controlador (MVC)**, aunque con algunas simplificaciones inherentes a una aplicación de escritorio pequeña.

- **Modelo:** Representa los datos y la lógica de negocio de la hoja de cálculo. Clases clave: Celda, Hoja, ListaHojas, NodoLista, TablaHash.
- **Vista:** Encargada de la interfaz de usuario y la presentación de los datos. Clase clave: HojaView.
- **Controlador:** Actúa como intermediario entre el Modelo y la Vista, gestionando las interacciones del usuario, actualizando el Modelo y la Vista según sea necesario. Clase clave: HojaController.

2. Estructura del Proyecto y Clases

El proyecto está organizado en varias clases, cada una con una responsabilidad específica.

2.1 Diagrama de Clases (Conceptual)



2.2 Descripción Detallada de Clases

A continuación, se describe cada clase, incluyendo sus atributos, métodos y responsabilidades.

2.2.1 Celda.java

- **Descripción:** Representa una celda individual en la hoja de cálculo. Almacena su valor actual, la fórmula asociada (si existe) y gestiona las dependencias con otras celdas para la propagación de cambios.
- **Responsabilidades:**
    - Almacenar el valor y la fórmula de la celda.
    - Mantener listas de celdas que dependen de ella (dependencias) y celdas de las que ella depende (referencias).
    - Notificar a sus celdas dependientes cuando su valor cambia.
    - Limpiar las relaciones de dependencia/referencia.
- **Atributos:**
    - private String valor: El valor visible de la celda (resultado de una fórmula o entrada directa).
    - private String formula: La fórmula original ingresada (si la celda contiene una fórmula).
    - private Set<Celda> dependencias: Conjunto de objetos Celda que dependen del valor de esta celda (se actualizan cuando esta celda cambia).
    - private Set<Celda> referencias: Conjunto de objetos Celda de los que esta celda depende (necesita sus valores para calcular su fórmula).
- **Métodos Clave:**
    - Celda(): Constructor.
    - getValor(): Obtiene el valor actual.
    - setValor(String valor): Establece el valor y notifica a las celdas dependientes.

- getFormula(): Obtiene la fórmula.
- setFormula(String formula): Establece la fórmula.
- addDependencia(Celda celda): Agrega una celda a la lista de dependientes.
- addReferencia(Celda celda): Agrega una celda a la lista de referencias.
- notificarCambio(): Itera sobre dependencias y llama a actualizarValor() en cada una.
- actualizarValor(): Marca la celda para una posible reevaluación (la lógica real de evaluación está en el controlador).
- limpiarRelaciones(): Elimina esta celda de las listas de dependencia de sus referencias y vacía su propia lista de referencias.

2.2.2 Hoja.java

- **Descripción:** Representa una hoja de cálculo individual, conteniendo una matriz bidimensional de objetos Celda. Es serializable para futuras funcionalidades de guardado/carga.
- **Responsabilidades:**
  - Almacenar la matriz de celdas.
  - Gestionar el nombre, número de filas y columnas.
  - Proporcionar acceso a celdas individuales por coordenadas.
- **Atributos:**
  - private Celda[][] celdas: Matriz que almacena todas las celdas de la hoja.
  - private String nombre: Nombre de la hoja (ej: "Hoja 1").
  - private int filas: Número de filas de la hoja.
  - private int columnas: Número de columnas de la hoja.

    - private static final long serialVersionUID = 1L: Para la serialización.

- **Métodos Clave:**
    - Hoja(String nombre, int filas, int columnas): Constructor que inicializa la matriz de celdas.
    - inicializarCeldas(): Método privado para crear instancias de Celda en la matriz.
    - getCelda(int fila, int columna): Retorna la Celda en las coordenadas dadas.
    - setCelda(int fila, int columna, String valor): Establece un valor en una celda específica.
    - getNombre(), setNombre(String nombre): Gestiona el nombre de la hoja.
    - getFilas(), getColumnas(): Obtienen las dimensiones.

2.2.3 HojaController.java

- **Descripción:** El controlador en el patrón MVC. Actúa como el puente entre HojaView (la interfaz de usuario) y ListaHojas (el modelo de datos). Contiene la lógica para la evaluación de fórmulas.
- **Responsabilidades:**
    - Capturar eventos de la HojaView (edición de celdas, clic en botones).
    - Actualizar el modelo (ListaHojas y Hoja) basándose en la interacción del usuario.
    - Actualizar la HojaView para reflejar los cambios en el modelo.
    - Evaluar fórmulas ingresadas por el usuario.
    - Gestionar las dependencias de celdas.
- **Atributos:**
    - private ListaHojas listaHojas: Referencia al modelo de datos.

    - private HojaView vista: Referencia a la vista.

- **Métodos Clave:**
    - HojaController(ListaHojas listaHojas, HojaView vista): Constructor que inicializa el controlador y configura los listeners.
    - configurarListeners(): Configura TableModelListener para la edición directa en la tabla y ListSelectionListener para actualizar el campo de fórmula cuando cambia la celda seleccionada.
    - configurarBotonAplicar(): Configura el listener para el botón "Aplicar" en la vista.
    - aplicarFormulaACeldaSeleccionada(): Lógica para aplicar el contenido del campo de fórmula a la celda seleccionada, incluyendo la limpieza de relaciones y la evaluación si es una fórmula.
    - evaluarFormula(String formula, Hoja hoja): **Lógica central para la evaluación de fórmulas.** Actualmente soporta sumas y multiplicaciones simples entre dos referencias de celda (ej: =B3+B4). Identifica las referencias de celda, obtiene sus valores y realiza la operación.
    - obtenerValorCelda(String ref, Hoja hoja): Método auxiliar para parsear una referencia de celda (ej: "B3") a coordenadas numéricas y obtener su valor. Maneja conversiones de columna (A->0, B->1) y parseo de fila.

2.2.4 HojaView.java

- **Descripción:** La Vista en el patrón MVC. Se encarga de construir y mostrar la interfaz gráfica de usuario usando componentes Swing. Permite al usuario interactuar con la hoja de cálculo.
- **Responsabilidades:**
    - Crear y organizar todos los componentes visuales (ventanas, menús, tablas, campos de texto, botones).
    - Mostrar el estado actual del modelo (Hoja, Celda).

    - Manejar la creación, eliminación y renombrado de pestañas de hojas.
    - Proporcionar métodos para que el controlador acceda a sus componentes (ej: getTabla, getModelo).
    - Implementar la visualización de la numeración de filas y la funcionalidad de la tabla hash.
- **Atributos Públicos (para acceso desde el controlador):**
    - public JMenuBar menuBar
    - public JTextField campoFormula
    - public JButton btnAplicar, public JButton btnRechazar
    - public JTabbedPane pestañas
    - public ArrayList<JTable> tablas
    - public ArrayList<DefaultTableModel> modelos
    - public JLabel lblFormula
- **Atributos Privados:**
    - private ListaHojas listaHojas: Referencia al modelo.
    - private HojaController controlador: Referencia al controlador.
    - private int contadorHojas: Para nombrar hojas nuevas por defecto.
- **Métodos Clave:**
    - HojaView(ListaHojas listaHojas): Constructor que inicializa la interfaz y carga las hojas existentes.
    - configurarInterfaz(): Configura propiedades básicas de la ventana.
    - configurarMenu(): Crea la barra de menú y sus opciones.
    - configurarPanelSuperior(): Organiza el campo de fórmula y botones.

- configurarPestañas(): Inicializa el JTabbedPane.
- cargarHojasIniciales(): Itera las hojas del modelo y las agrega a la vista.
- agregarHojaAVista(int indiceHoja): Crea un JTable y DefaultTableModel para una hoja, configura encabezados de columnas, numeración de filas y lo añade al JTabbedPane.
- agregarNumeracionFilas(JScrollPane scrollPane, JTable tabla, int numFilas): Método auxiliar para crear y añadir los números de fila.
- mostrarTablaHash(): Crea y muestra la ventana de la tabla hash.
- actualizarTablaHash(JTable tabla, TablaHash hash): Refresca los datos en la tabla hash.
- insertarNuevaHoja(), eliminarHojaActual(), renombrarHojaActual(): Métodos para la gestión de hojas, que interactúan con listaHojas y pestañas.
- setControlador(HojaController controlador): Establece el controlador.
- getTabla(int indice), getModelo(int indice), getNumeroHojas(): Métodos de acceso para el controlador.
- RowHeaderRenderer: Clase interna privada que implementa ListCellRenderer para dibujar los encabezados de fila.

2.2.5 ListaHojas.java

- **Descripción:** Una implementación de lista enlazada simple para almacenar objetos Hoja. Se utiliza para cumplir con el requisito de no usar estructuras de datos predefinidas de la API de colecciones de Java para el almacenamiento principal de hojas.
- **Responsabilidades:**
  - Mantener una colección ordenada de objetos Hoja.
  - Permitir agregar, obtener y eliminar hojas por índice.

- **Atributos:**
    - private NodoLista cabeza: El primer nodo de la lista.
    - private int tamaño: El número actual de hojas en la lista.
- **Métodos Clave:**
    - ListaHojas(): Constructor.
    - agregarHoja(Hoja hoja): Añade una hoja al final de la lista.
    - obtenerHoja(int indice): Recupera una hoja por su posición.
    - eliminarHoja(int indice): Remueve una hoja de la lista.
    - cantidad(): Retorna el número de hojas.

2.2.6 NodoLista.java

- **Descripción:** Una clase auxiliar para ListaHojas, representando un nodo individual en la lista enlazada.
- **Responsabilidades:**
    - Almacenar un objeto Hoja.
    - Mantener una referencia al siguiente NodoLista.
- **Atributos:**
    - public Hoja hoja: La hoja almacenada en este nodo.
    - public NodoLista siguiente: Referencia al siguiente nodo en la lista.
- **Métodos Clave:**
    - NodoLista(Hoja hoja): Constructor.

2.2.7 TablaHash.java

- **Descripción:** Una implementación personalizada de una tabla hash con una función hash simple y resolución de colisiones por sondeo lineal (dirección

abierta). Se utiliza como una funcionalidad de demostración independiente de la hoja de cálculo.

- **Responsabilidades:**
    - Calcular un índice hash para una clave String.
    - Insertar claves en la tabla, manejando colisiones.
    - Recuperar valores por índice.
- **Atributos:**
    - private String[] tabla: El arreglo subyacente que representa la tabla hash.
    - private int tamaño: La capacidad de la tabla.
- **Métodos Clave:**
    - TablaHash(int tamaño): Constructor.
    - hash(String clave): Función hash simple que suma los valores ASCII de los caracteres y aplica el módulo del tamaño de la tabla.
    - agregar(String clave): Inserta una clave. Si hay colisión, busca la siguiente posición disponible ((indice + 1) % tamaño).
    - obtener(int indice): Obtiene el valor en un índice específico.
    - getTabla(): Retorna el arreglo interno de la tabla.

2.2.8 main.java

- **Descripción:** La clase principal que sirve como punto de entrada de la aplicación.
- **Responsabilidades:**
    - Inicializar el modelo (ListaHojas).
    - Crear la vista (HojaView).

    - Instanciar el controlador (HojaController), pasándole el modelo y la vista para establecer las conexiones.
    - Asegurar que la interfaz de usuario se inicie en el hilo de despacho de eventos de Swing (SwingUtilities.invokeLater).
- **Métodos Clave:**
    - public static void main(String[] args): El método main estándar.

3. Flujo de Ejecución y Lógica Central

3.1 Inicio de la Aplicación

1. main.main() es llamado.
2. Un objeto ListaHojas es creado (el modelo).
3. Un objeto HojaView es instanciado, pasándole listaHojas. Si listaHojas está vacía, HojaView crea una "Hoja 1" por defecto.
4. Un objeto HojaController es instanciado, pasándole listaHojas y vista. Esto establece las referencias cruzadas y configura los listeners.
5. HojaView.setVisible(true) hace que la ventana de la aplicación sea visible.

3.2 Edición de Celdas y Evaluación de Fórmulas

Este es el flujo más importante que ilustra el patrón MVC:

1. **Interacción del Usuario (Vista):**
    - El usuario edita una celda directamente en la JTable o introduce texto en el campoFormula y hace clic en "Aplicar".
2. **Evento Capturado (Controlador):**
    - Si se edita la tabla, el TableModelListener en HojaController.configurarListeners() detecta un TableModelEvent.UPDATE.

- Si se usa el botón "Aplicar", el ActionListener de btnAplicar en HojaController.configurarBotonAplicar() es disparado.

3. **Procesamiento por el Controlador (HojaController):**
    - El controlador obtiene la celda Celda correspondiente del Hoja actual (a través de listaHojas.obtenerHoja().getCelda()).
    - **Limpieza de Relaciones:** Antes de aplicar una nueva fórmula o valor, celda.limpiarRelaciones() es llamado. Esto es crucial para evitar dependencias circulares o referencias obsoletas cuando el contenido de una celda cambia drásticamente. Elimina esta celda de las listas de dependencias de las celdas a las que previamente referenciaba.
    - **Detección de Fórmula:** Se verifica si el texto ingresado comienza con =.
        - **Si es una Fórmula:**
            - La fórmula se guarda en celda.setFormula().
            - Se llama a HojaController.evaluarFormula(textoFormula, hojaActual) para calcular el resultado.
            - Dentro de evaluarFormula():
                - La cadena de la fórmula se parsea para extraer las referencias de celda (ej: "B3", "B4").
                - Para cada referencia, se llama a obtenerValorCelda() para obtener el valor numérico de la celda referenciada.
                - Si las celdas referenciadas existen y son válidas, se debería establecer una relación de dependencia explícita: celdaReferenciada.addDependencia(celdaActual) y celdaActual.addReferencia(celdaReferenciada). **(Nota: Esta parte de la gestión de dependencias**

para la *re-evaluación automática* en la cadena de notificaciones no está explícitamente implementada en el evaluarFormula actual. Solo celda.notificarCambio() se basa en dependencias preexistentes. El limpiarRelaciones() ayuda, pero la addReferencia/addDependencia dentro de evaluarFormula actual no establece las relaciones para el grafo de dependencia.)

- Se realiza la operación (+ o *).
- El resultado se devuelve como String.

- **Si es un Valor Directo:**
  - El texto se considera el valor directo.
  - celda.setFormula("") es llamado para borrar cualquier fórmula previa.

- **Actualización del Modelo (Celda):** El resultado (o el valor directo) se establece en el modelo llamando a celda.setValor(resultado).
- **Propagación de Cambios (Modelo):** celda.setValor() llama a celda.notificarCambio().
  - notificarCambio() itera sobre todas las dependencias de esta celda y llama a dependiente.actualizarValor().
  - actualizarValor() en la celda dependiente actualmente solo verifica si tiene una fórmula. La re-evaluación de la fórmula en cascada no se maneja automáticamente a través de este método en el Celda en sí, sino que se delega al HojaController para una re-evaluación manual o impulsada por el usuario. Esto significa que si A1 contiene =B1+C1 y B1 cambia, A1 *no* se actualizará automáticamente a menos que el usuario interactúe con A1 o una función de re-cálculo global se implemente.

- **Actualización de la Vista (HojaView):** El controlador actualiza directamente el DefaultTableModel de la JTable de la vista (modelo.setValueAt(resultado, fila, columna)), reflejando el nuevo valor en la interfaz gráfica.

3.3 Gestión de Hojas

Los eventos de menú (Insertar Hoja, Eliminar Hoja, Renombrar Hoja) en HojaView son manejados por ActionListener configurados en la propia clase HojaView. Estos métodos interactúan directamente con ListaHojas para modificar el modelo y con JTabbedPane y los ArrayList de tablas y modelos para actualizar la vista.

3.4 Tabla Hash

La funcionalidad de TablaHash es independiente de la lógica de la hoja de cálculo. Cuando el usuario selecciona "Tabla hash" en el menú, HojaView.mostrarTablaHash() crea una nueva ventana y una instancia de TablaHash. La interacción con esta tabla se maneja completamente dentro de los listeners de esa ventana, llamando a TablaHash.agregar() y actualizando la JTable de esa ventana.

4. Consideraciones de Diseño y Limitaciones

- **Gestión de Dependencias:** El sistema de dependencia (Celda.dependencias, Celda.referencias) está implementado, pero la re-evaluación en cascada de fórmulas (celda.actualizarValor()) es rudimentaria. Actualmente, una celda dependiente simplemente "sabe" que necesita una actualización, pero la re-evaluación real debe ser disparada manualmente por el usuario (editando la celda o usando el botón "Aplicar").
- **Evaluación de Fórmulas:** El método evaluarFormula en HojaController es muy básico, limitado a sumas y multiplicaciones entre exactamente dos referencias de celda. No soporta:
    - Números directos en fórmulas (ej: =A1+5).
    - Múltiples operaciones (ej: =A1+B1+C1).

    - Funciones más complejas (ej: SUM(), AVERAGE()).
    - Paréntesis para alterar el orden de las operaciones.
    - Manejo de errores robusto para entradas no numéricas o referencias inválidas (actualmente retorna 0 o ERROR).
- **Estructuras de Datos Personalizadas:** Se utilizan ListaHojas (lista enlazada) y TablaHash (arreglo con sondeo lineal) en lugar de las colecciones estándar de Java (ArrayList, HashMap). Esto cumple con un requisito específico pero puede impactar el rendimiento y la facilidad de mantenimiento en aplicaciones más grandes.
- **Serialización:** La clase Hoja es Serializable, lo que sienta las bases para guardar y cargar archivos de hoja de cálculo. Sin embargo, la lógica de guardado/carga (manejo de ObjectOutputStream y ObjectInputStream) no está implementada en el HojaView ni en el HojaController.
- **Manejo de Errores en UI:** Los mensajes de error al usuario son básicos (e.g., JOptionPane).

5. Notas de Implementación

- **Numeración de Filas:** La numeración de filas se logra con un JList en el RowHeaderView del JScrollPane, usando un ListCellRenderer personalizado para alinear su apariencia con los encabezados de la tabla.
- **Independencia de Componentes:** Aunque el controlador tiene acceso a muchos componentes de la vista, la lógica de negocio principal se mantiene en el modelo.
- **SwingUtilities.invokeLater:** Es fundamental para asegurar que las operaciones de la interfaz de usuario se realicen en el Event Dispatch Thread (EDT) de Swing, garantizando la seguridad de los hilos y la capacidad de respuesta de la interfaz.