**Editor de codigo – Proyecto Final de Estructuras de Datos**

# Resumen Ejecutivo

**iKode** es un **editor de código** con interfaz gráfica diseñado para gestionar proyectos de programación con eficiencia y orden. El sistema permite abrir, editar, cerrar y visualizar archivos en pestañas, registrando acciones para poder revertirlas, organizar archivos abiertos, y gestionar historial reciente. Aplica estructuras de datos complejas como listas enlazadas, pilas, colas con prioridad, árboles binarios y AVL, además de un TAD personalizado. Su diseño modular, orientado a objetos y con validación de entradas, garantiza robustez y escalabilidad.

# Introducción

## Contexto o problema abordado

En la actualidad, los desarrolladores requieren herramientas que les permitan gestionar múltiples archivos de código de manera ordenada y eficiente, especialmente al trabajar en proyectos grandes.

## Justificación del proyecto

El desarrollo de **iKode** responde a la necesidad de contar con un sistema educativo y funcional que no solo permita editar código, sino que también aplique estructuras de datos avanzadas para resolver tareas comunes como manejo de historial, ordenamiento, prioridades y revertir acciones.

# Objetivos

## **General:**

Desarrollar un editor de código funcional que implemente estructuras de datos aprendidas en clase para gestionar acciones y archivos.

## Específicos:

* Aplicar arreglos, pilas, colas, listas, árboles y TADs.
* Implementar interfaz gráfica con Swing.
* Garantizar modularidad y reutilización.
* Validar entradas y manejar errores básicos.

# Descripción del Sistema

## Funcionalidades principales:

* Abrir y cerrar archivos en pestañas.
* Mostrar historial de archivos recientes.
* Registrar y revertir acciones (pila).
* Gestionar archivos prioritarios (cola con prioridad).
* Ordenar archivos y buscar con arreglo.
* Navegar archivos con árbol binario.
* Balanceo automático con AVL.

Público objetivo: Estudiantes y desarrolladores que deseen aprender estructuras de datos aplicadas.

Casos de uso generales: Abrir carpeta de proyecto, navegar archivos, editar código, deshacer acciones, ver historial, buscar archivos.

# Diagrama de Clases

## PanelEditor: interfaz principal con menús, pestañas y JTree.

## ArchivoManager: maneja archivos abiertos (arreglo).

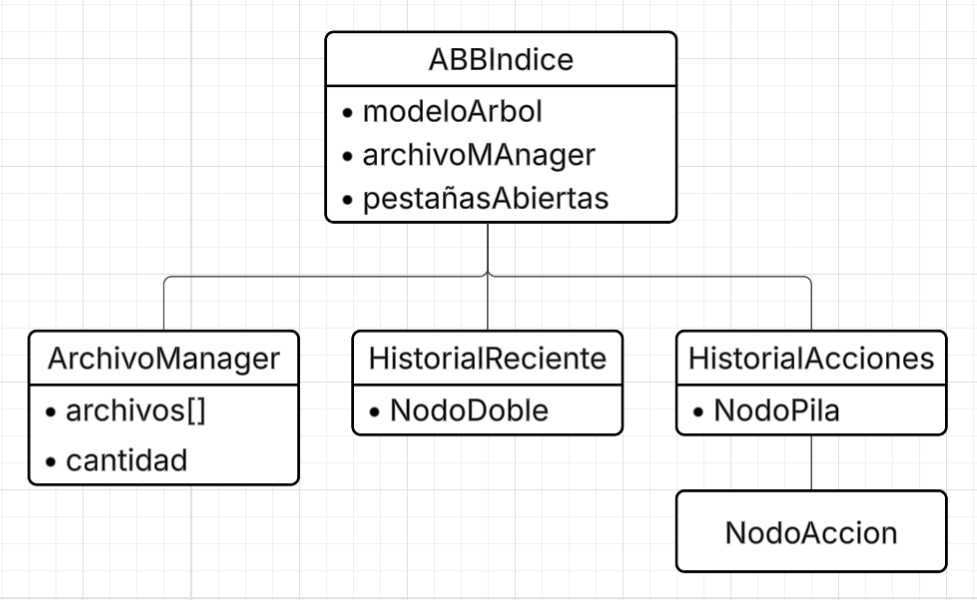
## HistorialRecientes: lista doble para historial de archivos.

## HistorialAcciones: pila para revertir acciones.

## ColaConPrioridad: gestión de archivos según prioridad.

## ABBIndice: búsqueda ordenada de archivos.

## Tarea: TAD personalizado para tareas programables.



# Uso de Estructuras de Datos

## Arreglo unidimensional

Usado en **ArchivoManager** para almacenar archivos abiertos y ordenarlos alfabéticamente. Aplica inserción, eliminación lógica, ordenamiento y búsqueda binaria.

## Lista enlazada (doble)

Implementada en **HistorialRecientes**, para manejar navegación cronológica de archivos. Se recorren hacia adelante o atrás, y se pueden eliminar nodos en cualquier posición.

## Pila dinámica

En **HistorialAcciones**, para registrar acciones (como abrir, cerrar archivo o ejecutar), permitiendo deshacer las 5 últimas con facilidad.

## Cola con prioridad

Implementada en **ColaConPrioridad**, permite encolar archivos según su prioridad de atención (normal o urgente) y procesarlos en orden.

## Árbol binario de búsqueda (ABB)

Clase **ABBIndice**, para organizar los nombres de archivo en un árbol que permite recorrer en preorden, inorden y postorden.

## Árbol AVL

Extensión del ABB con balanceo automático tras inserciones y eliminaciones. Se garantiza eficiencia logarítmica.

## TAD personalizado

Clase **Tarea**: TAD con campos como descripción, prioridad y estado. Se puede recorrer y modificar. Incluye especificación informal (funciones tipo **agregarTarea**, **completarTarea**) y formal (definición de campos y operaciones).

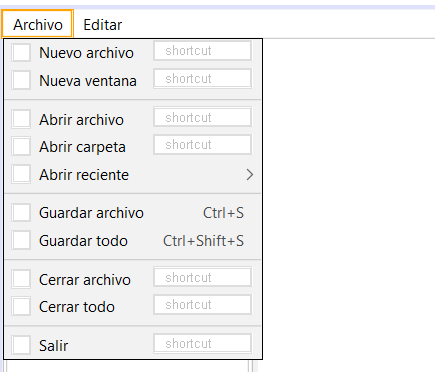
## Lectura/escritura de archivos

Se realiza persistencia básica para respaldar archivos recientes, tareas y configuraciones con objetos serializables.

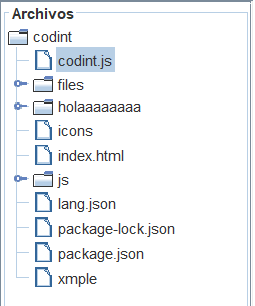
# Interfaz de Usuario y Experiencia

Se usó Java Swing. Interfaz amigable con:

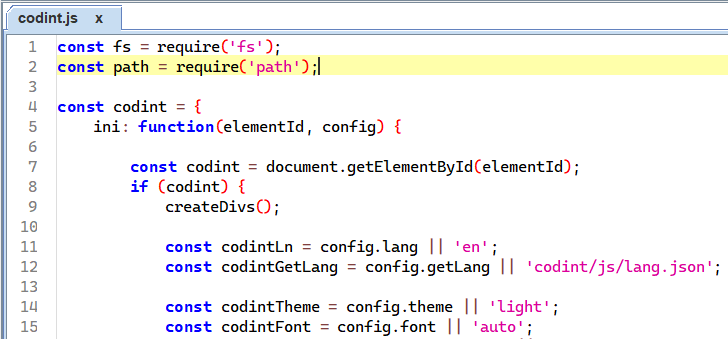
## Menú superior (abrir carpeta, salir).



## Panel de navegación (JTree).

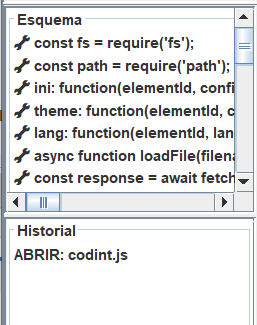


## Pestañas con código (RSyntaxTextArea).



## Terminal.

## Esquema y Historial.



# Retos Técnicos y Soluciones

* **Integrar estructuras en GUI**: Solucionado separando lógica en clases independientes.
* **Balanceo AVL**: Desarrollado con prueba unitaria previa.
* **Evitar pestañas duplicadas**: Se usó HashMap para manejar pestañas abiertas.
* **Serialización**: Se manejó con try-catch y objetos simples.

# Conclusiones

Se logro implementar correctamente todos los requisitos solicitados, cumpliendo con los objetivos establecidos para el desarrollo del proyecto. La aplicación de una arquitectura modular facilitó la realización de pruebas unitarias y permitió una mayor flexibilidad para futuras mejoras del sistema. Asimismo, la incorporación de una interfaz gráfica (GUI) aporta una experiencia de usuario más intuitiva y amigable. Como línea de desarrollo futura, se plantea la posibilidad de integrar funcionalidades colaborativas en red, lo que permitiría potenciar su uso en entornos educativos o de trabajo en equipo.

# Referencias

Oracle. Java Platform, Standard Edition Doc. <https://docs.oracle.com/javase/>

Oracle. The Java™ Tutorials. <https://docs.oracle.com/javase/tutorial/>

GeeksForGeeks. Java Programming Language. Disponible en: <https://www.geeksforgeeks.org/java/>

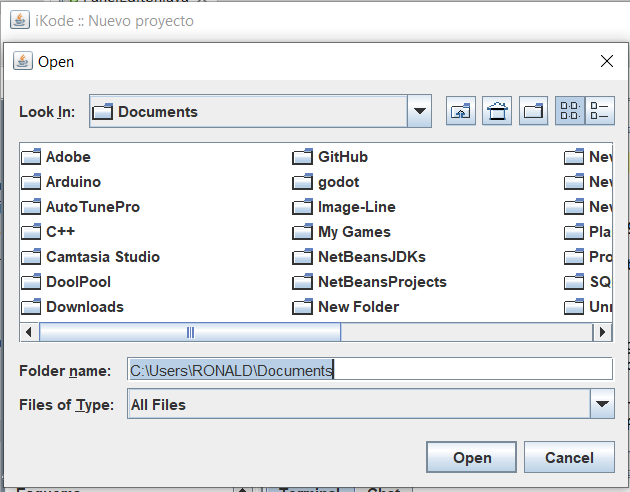
Apuntes y material proporcionado durante el curso de Programación Orientada a Objetos.

# Anexos

Enlace GitHub: <https://github.com/ronaldbit/iKode-Java-Swing>

Capturas de pantalla:

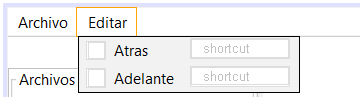
Abrir carpeta



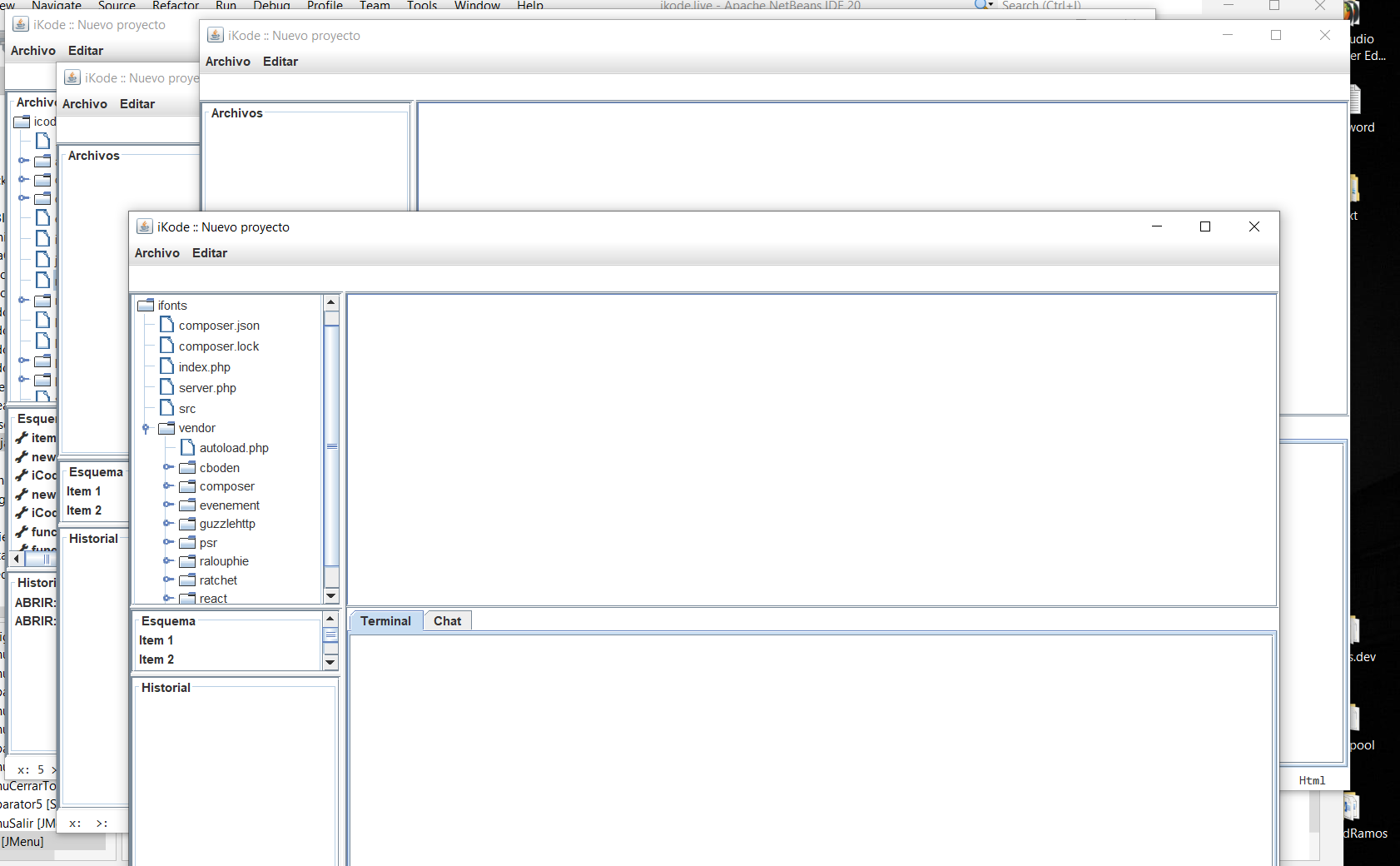
Barra de mensajes, linea y columna, lenguaje, advertencias y errores.



Acciones reversibles



Multiples proyectos



Estructura ABB / AVL (ABBIndice)

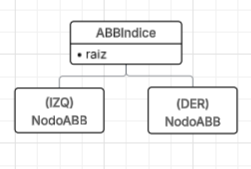


Diagrama General del proyecto:

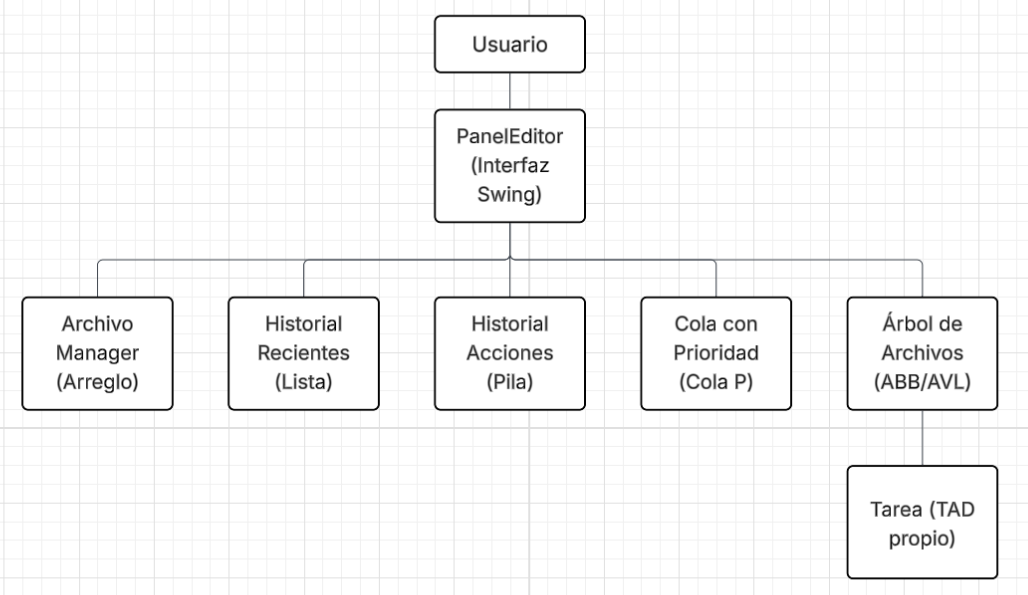


Diagrama de Flujo del Sistema (uso típico del usuario)

