explicación de implementación

Paquete lexer

- Clase Lexer
    - Responsabilidad: leer el texto de entrada y tokenizarlo según la gramática del proyecto (etiquetas tipo XML, números y tipos de operación).
    - Atributos clave: input, idx, line, col.
    - Métodos clave: next() devuelve el siguiente Token, métodos auxiliares peek(), advance(), skipWS().
    - Consideraciones: actualmente emite tokens con tipo TokenType y lexema. Maneja errores léxicos mediante TokenType.ERROR y lista de LexError.
- Clase Token
    - Estructura inmutable que guarda type, lexeme, line, column.
- Enum TokenType
    - Define todos los tipos de token reconocidos (ABRIROPERACION, CERRAROPERACION, NUMERO, etc.).

Paquete app.parser

- ParserAdapter
    - Convierte la representación de token del lexer (lexer.Token) a la representación textual que el Parser espera (clase interna Parser.Token).
    - Es un lugar correcto para centralizar transformaciones entre capas (p. ej. normalizar lexemas, convertir tags a cadenas <Operacion=SUMA>).
- Parser
    - Parser recursivo (descent) que consume la lista de tokens adaptados y construye un AST compuesto por ast.Node.
    - Gramática soportada: operaciones anidadas <Operacion, </Operacion>, números, y nodos <P>/<R> si se usan, etc.
    - Errores de parseo lanzan ParseException con información de línea/columna.

Paquete ast

- Interfaz Node
    - Contrato: evaluate() devuelve double y render() devuelve una representación textual.
- Num
    - Nodo hoja que contiene un double literal.
- Op (abstracta)
    - Nodo interior con nombre (por ejemplo SUMA) y hijos (operandos), métodos para añadir hijos y obtenerlos.
    - Subclases concretas (Suma, Resta, Division, Multiplicacion, Potencia, Raiz, Inverso, Mod) implementan evaluate() con la semántica correspondiente.
- OperacionFactory

    - Factoría que construye la subclase Op adecuada dada una cadena (por ejemplo "SUMA" -> Suma).

Paquete app.report

- HtmlReport
    - Genera un HTML con los resultados: tokens, errores, tablas de resultados, valores de operaciones y logs.
    - Contiene estilos embebidos (CSS) y una plantilla para listar resultados. Se crea en disco (ej.: out/resultados.html).
- ArbolGrafico
    - Recorre el AST y genera un archivo DOT (Graphviz). Si Graphviz está instalado, también intenta generar PNG.
- Clases auxiliares: OperacionResultado, InfixPretty ayudan a formatear salidas.

Paquete app y app.gui

- Analizar
    - Punto de integración entre lexer y parser: recibe List<lexer.Token>, adapta con ParserAdapter y devuelve List<ast.Node> o lista vacía ante errores.
- Main
    - Programa de consola: lee archivo, ejecuta lexer, guarda tokens/errores, ejecuta Analizar, genera reportes y gráficos, guarda en out/.
- AppFrame / GuiMain
    - Interfaz gráfica Swing que permite abrir archivos, ejecutar análisis, visualizar resultados y abrir PDFs/manuales.
    - Llama al pipeline de Lexer → Analizar → HtmlReport / ArbolGrafico para mostrar resultados.