TABLA DE CONTENIDO

DESCRIPCIÓN DE LA SOLUCIÓN

OBJETIVOS

GENERALES

- Ofrecer una manual con las especificaciones técnicas de la aplicación proveída al usuario.

ESPECÍFICOS

- Explicar de forma breve y clara la funcionalidad interna de la aplicación a través de diagramas.
- Especificar el funcionamiento del autómata finito determinista usado en la aplicación.

ALCANCES

Este manual pretende dar las especificaciones y características técnicas de la aplicación, así como mostrar el funcionamiento de la aplicación a través de diagramas de flujo, clases y secuencia. Además, ya que la aplicación es, en sí, un analizador léxico, también se explica el funcionamiento a través de un autómata finito determinístico.

FUNCIONALIDAD

La aplicación es un analizador léxico, es decir, a través de procesos internos reconoce comandos preestablecidos, y si estos poseen la sintaxis correcta, devolverá al usuario la salida de los comandos ingresados.

En general, la aplicación posee la función de graficar árboles binarios y calcular expresiones aritméticas con operadores básicos (a través de líneas de comandos). Estos poseen como salida dos archivos HTML, si estuviesen bien escritos; y un archivo HTML, si no lo estuvieren.

Para graficar los árboles binarios la aplicación utiliza librerías externas, proveídas por Graphviz.

DESCRIPCIÓN DEL DISEÑO

REQUISITOS TÉCNICOS

Los requisitos técnicos describen los que el sistema en dónde se ejecutará la aplicación, necesita para que la ejecución de la aplicación se la óptima.

Los detallados a continuación son los requisitos que la aplicación proveída:

- Windows 8x32 o Windows 10x32.
- .NET Framework 4.6.1
- Espacio libre en el disco duro mayor a 10GB.
- Memoria RAM 2GB.

CARACTERISTICAS TÉCNICAS

Las características técnicas describen a los atributos de la aplicación resaltando la más importantes, en cuanto a la forma en la que está funciona y como lo hace.

- La aplicación utiliza el .NET Framework 4.6.1
- Escrita con el lenguaje orientado a objetos: C#
- Utiliza una serie de librerías externas proveídas por Grapvhiz, que se utilizan en la generación de árboles binarios.

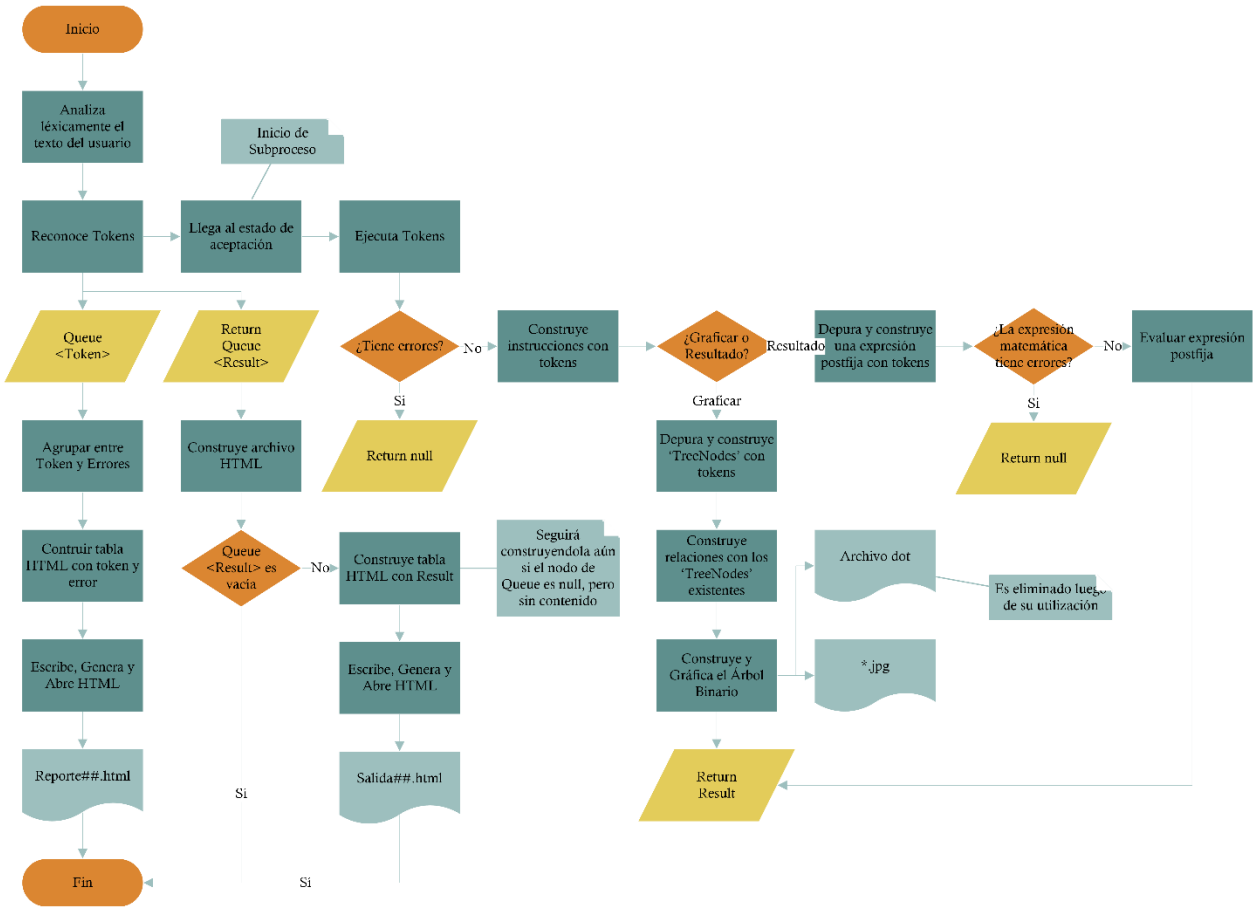
Es de notar que las librerías externas se proveen junto con la aplicación y su ubicación no debe ser modificada ni su contenido eliminado, ya que podría causar fallos durante la ejecución de la aplicación.

Los archivos de salida se almacenan junto con las librerías externas. Los archivos de salida comprenden:

- Archivos HTML.
    - Reporte de tokens
    - Resultado de funciones utilizadas.
- Imágenes .jpg
    - Generada a partir de la función «Graficar». Son las imágenes de los árboles binarios que se muestra en los archivos HTML.

DISEÑO DE LA APLICACIÓN

DIAGRAMA DE FLUJO

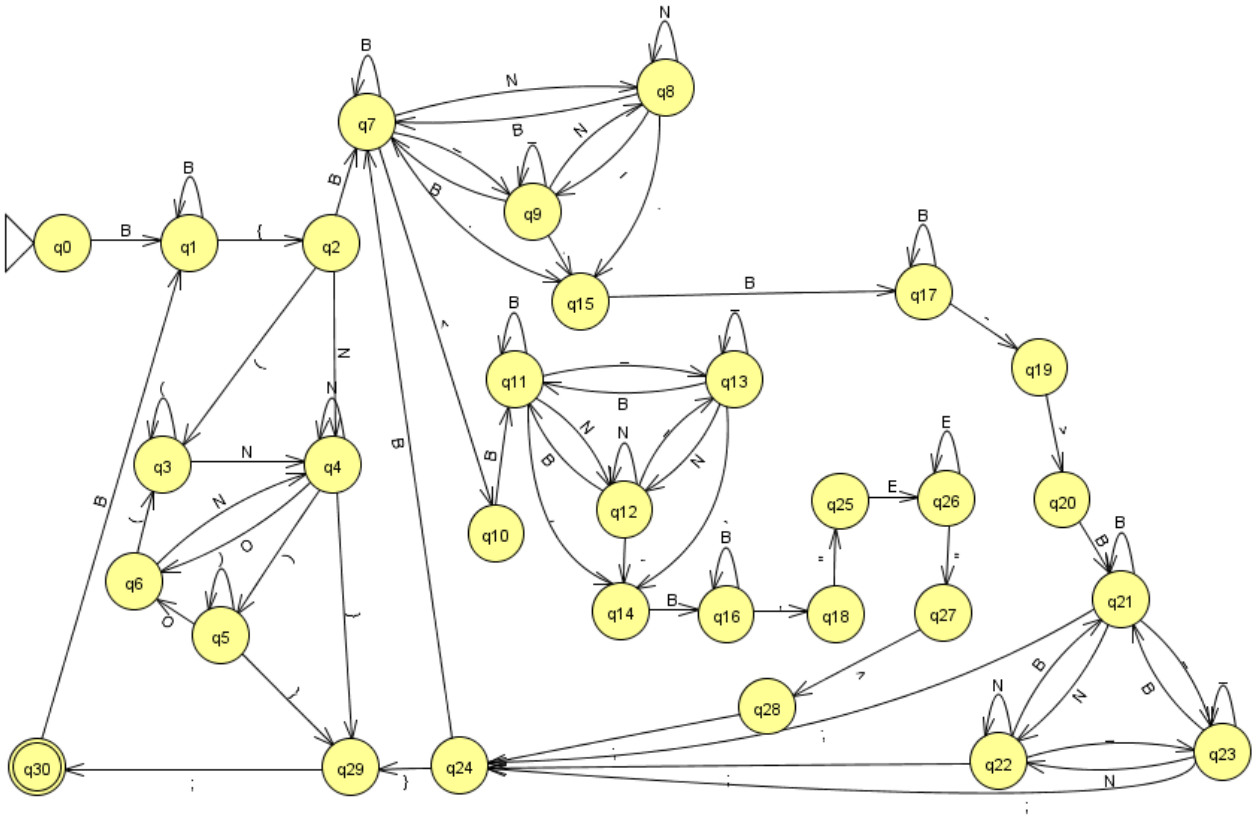


Este diagrama de flujo representa el funcionamiento de la aplicación cuando a esta se le indica que ejecute un análisis léxico, construya árboles binarios u opere una expresión matemática.

En el proceso "**Llega al estado de aceptación"**, es de resaltar, que cuando se está ejecutando el análisis léxico, el Autómata Finito Determinístico puede llegar en más de una ocasión a un estado de aceptación. Es por eso que este proceso *retorna* en "**Return Queue<Result>**" una cola con *resultados* de las ejecuciones de 'graficar' y 'resultado'.

Cuando se intenta construir una HTML con una cola (Queue) con *nodos* vacíos o nulos, el HTML no se genera, de ahí la condición que se encuentra después de "**Construye archivo HTML**" tenga propósito al evitar gastar recursos de memoria.

AUTÓMATA FINITO DETERMINÍSTICO

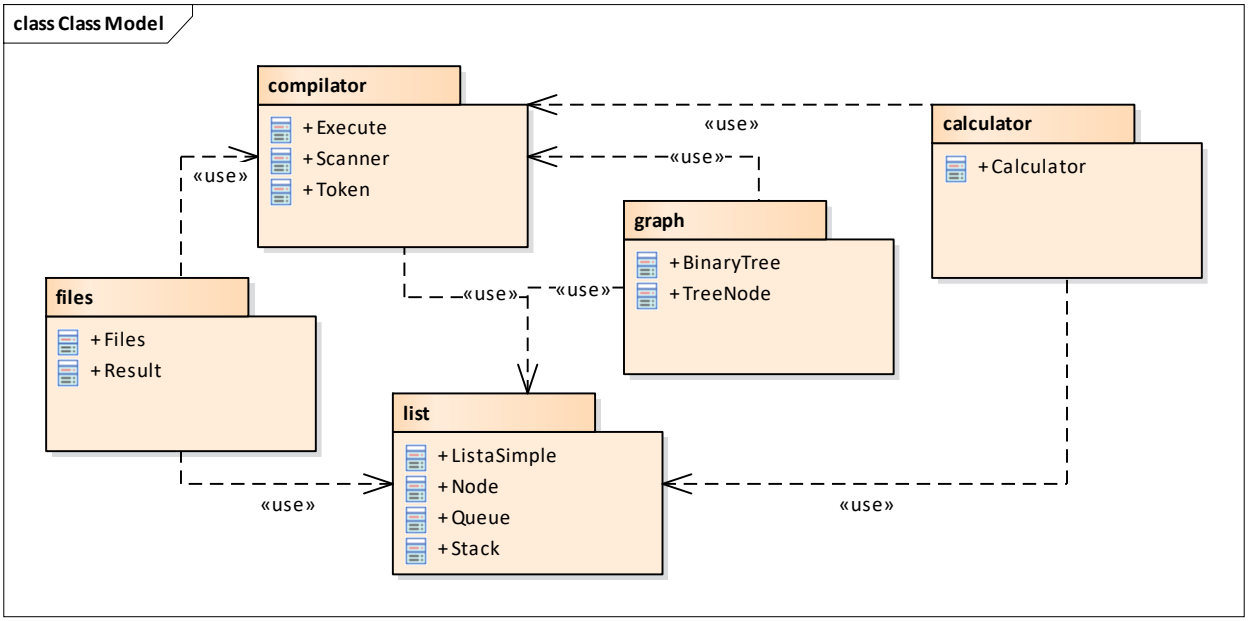


El AFD presente ilustra el funcionamiento del analizador léxico que posee la aplicación. En la implementación, se agregaron a cada estado un *estado trampa*, sin embargo y debido a que se procuró mantener la integridad del diagrama, aquí no se muestran.

A continuación, se describe el alfabeto que utiliza el AFD para construir los tokens:

- **Alfabeto:**
  - $B = \{alfabeto\ inglés\}$
  - $N = \{numero\ natural\ 0-9\}$
  - $E = \{cualquier\ caracter\ que\ se\ encuentre\ disponible\}$
  - $O = \{operador\ \ aritmético: + - * /\}$
- **Símbolos**
  - Llaves → {} | Paréntesis → () | Menor que y mayor que → < >
  - Punto → . | Punto y coma → ; | Comillas → “” | Guion bajo → _

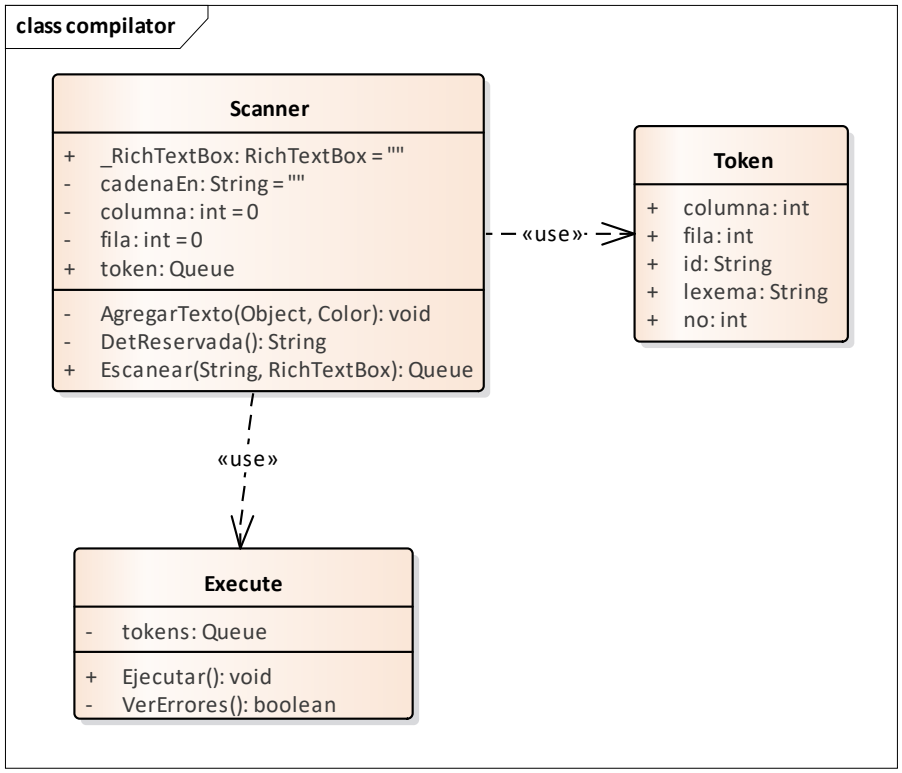
DIAGRAMA DE CLASES



El diagrama muestra los paquetes utilizados en la aplicación, así como las clases que cada paquete contiene. Además, se muestra como los paquetes se relacionan entre sí a través del uso que se dan entre sí.

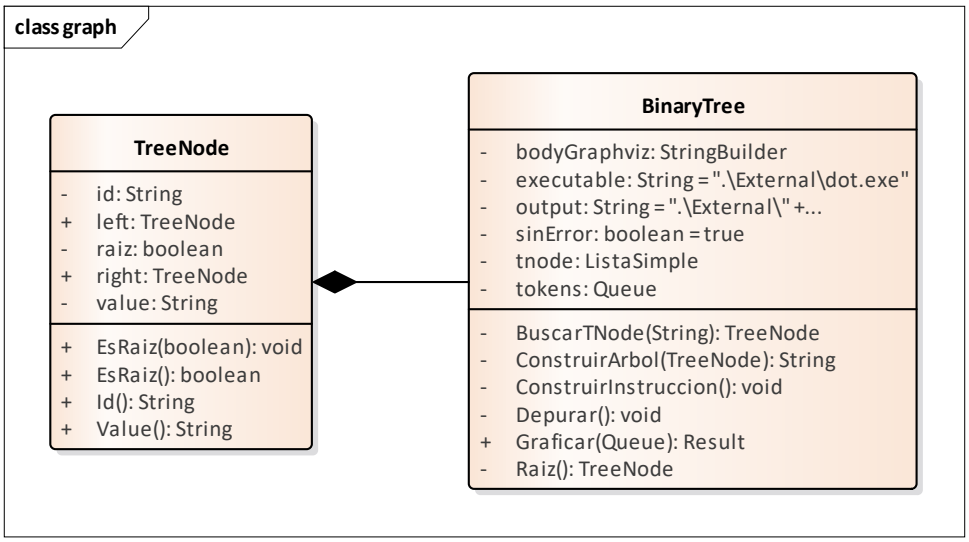
- **Compilator.** Este paquete contiene la implementación del AFD, así como las clases necesarias para la construcción de Tokens.
- **Graph.** Este paquete contiene las clases que se utilizan para construir y generar un árbol binario a un archivo JPG.
- **Calculator.** Este paquete contiene una única clase que se encarga de evaluar una operación matemática a través del Algoritmo Shunting-yard
- **Files.** Este paquete contiene las clases que se encargan de la generación de archivo HTML y archivos TXT.
- **List.** Contiene las listas dinámicas que se utilizan durante toda la ejecución de la aplicación.

COMPILATOR



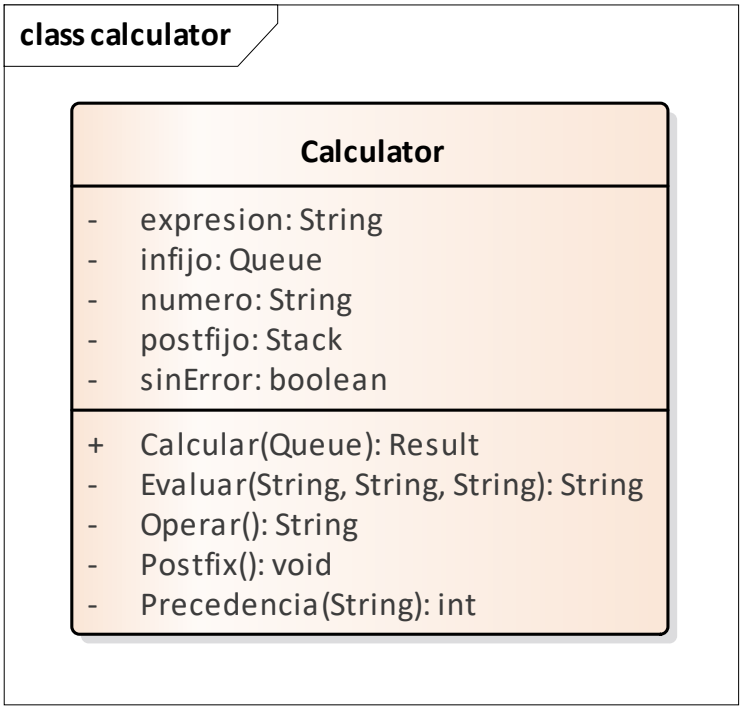
- **Scanner** contiene la implementación del AFD mostrado en diagramas anteriores.
    - **AgregarTexto**, agrega texto a **«_RichTextBox»** con un formato específico.
    - **DetReservada**, determina si la cadena en «**cadenaEn**» es una palabra reservada.
    - **Escanear**, es la implementación del AFD. Aquí se agrega el *estado trampa* a cada estado que se mostró en el diagrama del AFD.
- **Token** es la clase que se utiliza para construir los tokens generados por **Scanner**, posee un constructor con todos sus atributos como argumentos.
- **Execute** se utiliza para determinar la función a ejecutar cuando la operación **Escanear** de **Scanner** llega a un estado de aceptación y, dependiendo del resultado de **VerErrores**, la cola «**tokens**» se manda para su posterior ejecución a la clase correspondiente, según la función determinada por Ejecutar. Es de resaltar que la cola que se manda a ejecución, es depurada previamente enviando únicamente los tokens más significativos.

GRAPH



- **TreeNode** ayuda en la construcción de los nodos que conforman el Árbol Binario.
    - **EsRaiz** ayuda a saber (get) si el **TreeNode** en cuestión es una raíz, es decir, si desde este nodo se empieza a generar el Árbol Binario. Al mismo tiempo determina (set) si un **TreeNode** puede ser o no raíz.
    - **Id** devuelve el id de un **TreeNode** en particular. **Value** posee la misma función.
- **BinaryTree** construye y genera el Árbol Binario a través de una cola de tokens.
    - Desde **Graficar** se desencadena todas las demás operaciones privadas de la clase. Acepta una cola que hace referencia en «**tokens**»
    - **Depurar**, crea los **TreeNodes** con «**tokens**» y los relaciona entre sí en «**tnode**» ya sea en «**left**» o «**right**».
    - **BuscarTNode**, busca un **TreeNode** en una lista dinámica dado el id del **TreeNode**.
    - **ConstruirArbol**, es una operación recursiva que construye las relaciones binarias desde un **TreeNode.** El resultado se guarda en «**bodyGraphviz»**.
    - **ConstruirInstrucción,** construye una instrucción utilizando «**output**» de tal forma que la librería externa de Graphviz, dot, pueda reconocerla y ejecutarla correctamente.
    - **EsRaiz** utiliza «**tnode**» para determinar cuál **TreeNode** es la raíz del Árbol Binario.

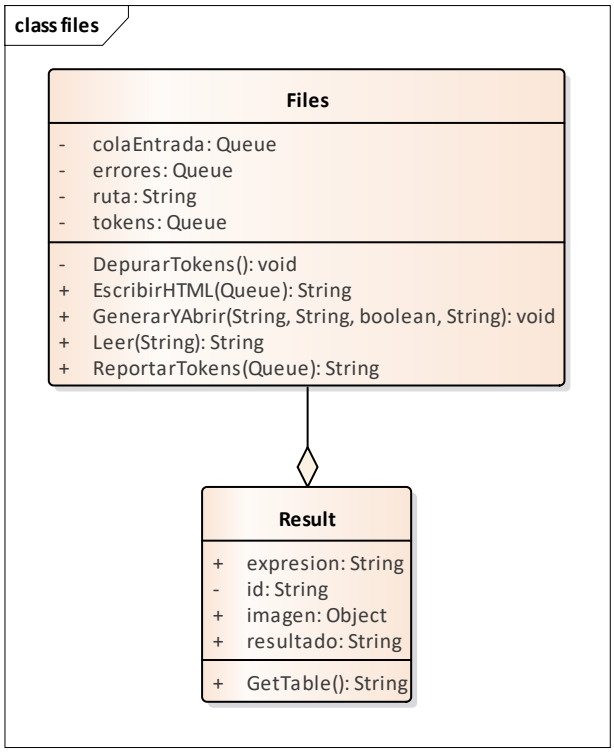
CALCULATOR



Calculator es la única clase que contiene el paquete con el mismo nombre, es utilizada para calcular expresiones aritméticas con operadores básicos (suma, resta, multiplicación y división)

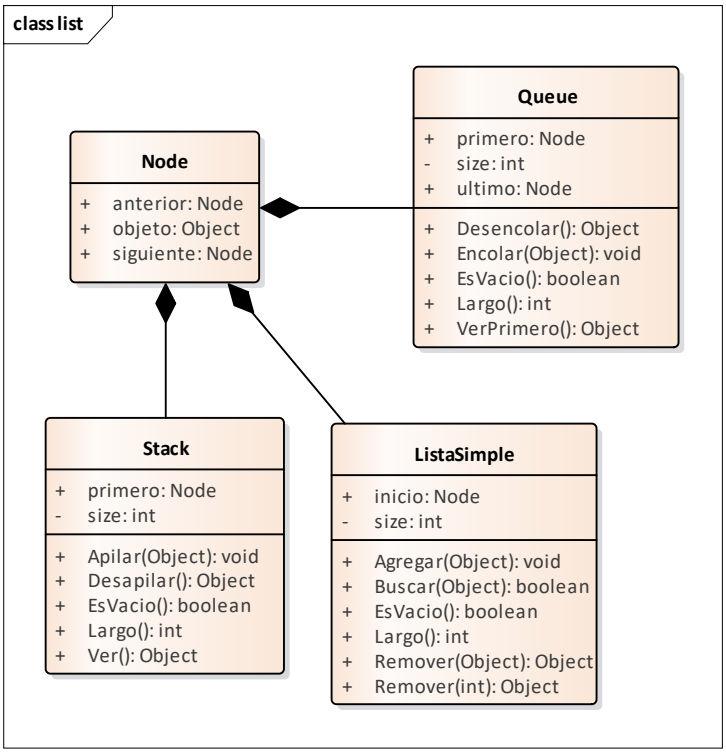
- Desde **Calcular** se desencadena todas las demás operaciones privadas de la clase. Acepta una cola que hace referencia en «**infijo**».
- **Postfix**, se encarga de transformar la los tokens contenidos en «**infijo**», en una expresión postfija utilizando el Algoritmo Shunting-yard. El resultado lo guarda en «**postfijio**».
    - **Precedencia**, es utilizado para determinar la jerarquía de símbolo en la construcción de «**postfijo**».
- **Operar**, a través de un algoritmo, opera los tokens de «**postfijo**».
    - **Evaluar**, es utilizado para evaluar aritméticamente las operaciones alojadas en «**postfijo**».

FILES



- **Files**, se utiliza en la creación y lectura de archivos.
    - **EscribirHTML**, se utiliza para escribir un archivo HTML.
    - **GenerarYAbrir**, se utiliza para crear físicamente un archivo en particular dentro del disco duro, con un contenido, nombre y una extensión dada.
    - **ReportarTokens**, acepta como entrada una cola de tokens, y la utiliza para referenciar a la cola «**colaEntrada**». Utiliza «**errores**» y «**tokens**» para escribir un archivo HTML.
        - **DepurarTokens**, utiliza «**colaEntrada**» y la separa en «**errores**» y «**tokens**».
    - **Leer**, utiliza una URL para ubicar un archivo dentro del disco y devuelve el contenido de dicho archivo.
- **Result**, esta clase se utiliza para almacenar los resultados de las funciones de Resultado (**Calculator**) y Graficar (**Graph**), respectivamente utilizan los atributos de expresión y resultado, e imagen (guardando la URL de la ubicación de la imagen en el disco). A través de **GetTable** se obtiene una tabla HTML con los atributos alojados en un **Result** particular.

LIST



Este paquete, contiene las tres listas dinámicas que se utilizan durante la ejecución de la aplicación. Cada una de ellas posee funciones que agrega o eliminan Nodos (**Nodes**), que determinan la cantidad de nodos que la conforma y que determinan si está vacía.

- **Node**, aloja un objeto en particular. Estos nodos son partes de las diferentes listas.
- **Stack**, manejada como una pila, utiliza la idea: "primero en entrar, último en salir".
- **ListaSimple**, esta lista funciona como una lista enlazada simple, se agregan al final y pueden ser removidas, por una referencia dada o por un índice dado.
- **Queue**, manejada como una cola, utiliza la idea: "primero en entrar, primero en salir".

GLOSARIO

1. **Algoritmo Shunting-yard**. Método para analizar ecuaciones matemáticas especificadas en una notación infijo. Está basado en una pila.
2. **Analizador Léxico**. Primera fase de un compilador que consiste en un programa que recibe como entrada una secuencia de caracteres y produce una salida compuesta de *tokens*.
3. **Árbol binario**. Es una estructura de datos en la cual, cada nodo puede tener un hijo izquierdo y un hijo derecho. No puede tener más de dos hijos.
4. **Autómata finito determinista/determinístico**. Es un autómata finito en el que cada estado en el que se encuentre dicho autómata, y con cualquier símbolo del alfabeto leído, existe siempre no más de una transición posible desde ese estado y con ese símbolo.
5. **Cola**. Es una estructura de datos caracterizada por ser una secuencia FIFO (del inglés *First In First Out*), debido a que el primer elemento en entrar será también el primero en salir.
6. **Diagrama de Clases**. En ingeniería de software, un diagrama de clases en Lenguaje Unificado de Modelado (UML) es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones y las relaciones entre los objetos.
7. **Diagrama de Flujo**. Es un diagrama que representa paso a paso los flujos de trabajo y operaciones de los componentes en un sistema.
8. **Estado trampa**. Es un estado de aceptación de un autómata, al cual se llega cuando una transición no está definida en el conjunto de transiciones de dicho autómata, produciendo así que la cadena analizada sea inválida.
9. **Graphviz**. Es un conjunto de herramientas de software para el diseño de diagramas definido en el lenguaje descriptivo DOT.
10. **Lista Dinámica.** Es una forma particular de organizar datos en una computadora para que puedan ser utilizados de manera eficiente.
11. **Nodo.** Es un registro que contiene un dato de interés y al menos un puntero para referenciar (apuntar) a otro nodo.
12. **Notación Infijo.** Es la notación común de fórmulas aritmética y lógicas, en la cual se escriben los operadores entre los operandos.

13. **Notación Postfijo.** Es un método algebraico de introducción de datos en la cual los operandos se encuentra de primero y después viene el operador que va a realizar los cálculos sobre ellos.
14. **Pila.** Es una estructura de datos caracterizada por ser una secuencia LIFO (del inglés *Last In First Out*), debido a que el último elemento en entrar será también el primero en salir.
15. **Sintaxis.** Grupo de normas que marcan las secuencias correctas de los elementos propios de un lenguaje de programación.
16. **Token.** Es una cadena de caracteres que tiene un significado coherente en cierto lenguaje de programación.