UNIVERSIDAD SAN CARLOS DE GUATEMALA
CENTRO UNIVERSITARIO DE OCCIDENTE

MANUAL TÉCNICO - PARSER PY

ESTUDIANTE:

JORGE ANIBAL BRAVO RODRÍGUEZ

REGISTRO ACADÉMICO
202131782

LENGUAJES FORMALES Y DE PROGRAMACIÓN

PRÁCTICA 1

PARSER PY

1. **HERRAMIENTAS DE DESARROLLO**

SISTEMAS OPERATIVOS

- **Ubuntu:** Ubuntu es una popular distribución de Linux basada en Debian. Es conocida por su facilidad de uso y su enfoque en ofrecer una experiencia amigable para todo tipo de usuarios. Ubuntu se destaca por su estabilidad, seguridad y su compromiso con el software libre.

- **Microsoft Windows:** Es un sistema operativo desarrollado por Microsoft . Utiliza el kernel NT y tiene una arquitectura híbrida. Ofrece una interfaz gráfica intuitiva y es compatible con una amplia gama de software y hardware. Se utiliza en computadoras personales y ofrece características como multitarea, personalización y conectividad en línea.

JAVA

Java es un lenguaje de programación de propósito general , orientado a objetos y altamente versátil. Fue desarrollado por Sun Microsystems y ahora es propiedad de Oracle. Es conocido por su portabilidad, lo que significa que el código escrito en Java se puede ejecutar en diferentes plataformas sin necesidad de modificaciones. Además, Java es utilizado ampliamente en el desarrollo de aplicaciones empresariales, aplicaciones móviles y aplicaciones web.

- **Adoptium JDK:** Adoptium JDK (anteriormente conocido como "AdoptOpenJDK") es una distribución de código abierto de la plataforma Java, basada en el proyecto OpenJDK. Se enfoca en ofrecer binarios de alta calidad, estables y de confianza para múltiples plataformas, incluyendo Windows, macOS y Linux.

  Proporciona el Java Development Kit (JDK), que incluye la máquina virtual de Java (JVM), compiladores y bibliotecas necesarias para desarrollar, compilar y ejecutar aplicaciones Java. También ofrece herramientas adicionales para el desarrollo y la administración de aplicaciones, como el depurador y el administrador de paquetes, además, es compatible con los estándares y certificaciones de Java SE, y ofrece actualizaciones trimestrales y soporte a largo plazo.

NETBEANS

NetBeans es un entorno de desarrollo integrado (IDE) de código abierto que ofrece un amplio conjunto de herramientas y funcionalidades para el desarrollo de aplicaciones Java. Este IDE está escrito en Java y es compatible con múltiples plataformas, incluyendo Windows, macOS y Linux.

Proporciona un editor de código completo con resaltado de sintaxis, autocompletado y navegación inteligente. También ofrece herramientas de depuración, pruebas y perfiles para facilitar el desarrollo y la optimización del código. Además, cuenta con un sistema de gestión de proyectos que permite organizar y administrar fácilmente los archivos y recursos del proyecto.

GRAPHVIZ

Graphviz es una herramienta de visualización de gráficos que permite representar datos y relaciones complejas mediante diagramas. Utiliza un lenguaje de descripción de gráficos llamado DOT (Graph Description Language) para definir la estructura y las conexiones de los elementos del grafo.

HTML

HTML (HyperText Markup Language) es un lenguaje de marcado utilizado para crear y estructurar el contenido de las páginas web. Permite definir la estructura y presentación de los elementos de una página, como encabezados, párrafos, imágenes, enlaces y formularios. Con HTML, se pueden agregar etiquetas y atributos a los elementos para indicar su función y apariencia.

2. ¿QUÉ ES PARSER - PY?

Parser - py es un proyecto desarrollado para el curso de lenguajes formales y de programación, que tiene por objetivo conocer el funcionamiento de una de las etapas de compilación conocida como análisis léxico, centrado en reconocer componentes del lenguaje llamados lexemas para clasificarlos en diferentes tipos. Estos reciben el nombre de Tokens, por lo que parser - py es, en esta etapa del proyecto, un analizador léxico inspirado en la sintaxis del lenguaje de programación Python.

3. ANÁLISIS LÉXICO: COMPONENTES DE PARSER - PY

ANALIZADOR - BACKEND

- **Token:** La clase Token.java es la representación de un Token (componente léxico) dentro del analizador léxico. Esta almacena los atributos necesarios para clasificar este tipo de componente, los cuales son:

    - **Tipo de token:** Almacena un String correspondiente a un tipo de token. Estos tipos han sido previamente definidos en una clase Enum (TokenTypeEnum.java). Es la clasificación directa del tipo de token almacenado.
      Relacionado con: **TokenDictionary, TokenTypeEnum**

    - **Patrón del token (expresión regular)**: String con la representación de la expresión regular del token almacenado.

    - **Lexema:** String con la “palabra” identificada.

    - **Línea / columna**

  Además, posee una función que retorna un String con código HTML correspondiente al gráfico de su lexema e identificación. Dicho código se muestra a continuación:

```
public String getGraphHtml(){
    String graph ="";

    String color = switch(tokenType){
        case "ARITHMETIC", "COMPARISION","LOGICAL", "ASSIGNAMENT" → "blue";
        case "KEYWORD"              → "purple" ;
        case "CONSTANT"             → "red"    ;
        case "COMMENT"              → "grey"   ;
        case "OTHERS"               → "green"  ;
        case "IDENTIFIER"           → "yellow"  ;

        //Caso especial, lexema desconocido
        case "LEXICAL_ERROR_unknow_lexeme" → "orange";
        default → "orange";

    };

    //Estilo comun para TODOS los nodos del token
    String style = "style=\"    width: 50px; background-color:" + color + "; border: 2px solid black; margin: 2px; border-radius: 100%; display: inline-block; padding: 10px;\"";

    //Cabecera: div del token   | display: grid, flex-wrap: wrap; display: grid;
    graph += " <div style=\"   border: 2px solid black;margin:4px; text-align: center;\">"
            + "<h1>" + tokenType + ": " + lexeme + "</h1>";

    //crear div de cada parte del lexema
    for (int i = 0; i < lexeme.length(); i++) {
        //verificar si no es el final
        if(i < lexeme.length() - 1){
            graph += "<div " + style + "> " + lexeme.charAt(index: i) + "</div> <div style= \"display:inline-block;\">&rarr;</div>";
        }else{
            //display: inline-block; display: flex;
            graph += "<div style=\" display: inline-block; width: 80px; border: 2px solid black; margin: 2px; border-radius: 20px ;  padding: 10px\">";
            graph += "<div " + style + "> " + lexeme.charAt(index: i) + "</div>";
            graph += "</div>";
        }

    }

    graph += "</div> <br/> <br/>\n\r";
    //cerrar cabecera
    return graph;
}
```

- **TokenTypeEnum:** Enum con los tipos de Token, los cuales son: IDENTIFIER, ARITHMETIC, COMPARISION, LOGICAL, ASSIGNAMENT, KEYWORD, CONSTANT,COMMENT, OTHERS.

  Aparte de estos, también se almacena LEXICAL_ERROR_unknow_lexeme, pues se deben identificar los reconocidos que no encajan en ninguna de las categorías anteriores.

- **TokenDictionary:** Es un envoltorio para el diccionario de Tokens. Posee un Map con datos String y TokenTypeEnum.

- **Analizer:** Esta clase es una de las más importantes que conforman a Parser-py, ya que es aquí donde se recorre caracter a caracter todo el input obtenido desde el editor de código y se crean los tokens en base a ciertos criterios. Tiene por atributos:

  - **Diccionario de Tokens:** Previamente definido.

  - **Lista de Tokens:** Lista en donde se guarda cada token generado por el método **createToken.**

  - **Validadores numérico e identificador:** Objetos que poseen lógica para validar un posible token como numérico o identificador. Esta validación devuelve un valor booleano.

  - **Texto de reporte:** Texto que se imprimirá en un archivo .csv que contiene la tabla de todos los tokens identificados por el analizador.

  - **Texto de entrada:** Texto que se analizará, obtenido del editor de texto.

  El primer análisis se lleva a cabo dentro del método **searchMatches**, el cual recorre caracter a caracter el texto de entrada y evalúa si el caracter actual sea de un tipo que modifique la configuración de análisis. Por ejemplo, si encuentra una coincidencia con **#** configura al analizador para aceptar todo tipo de caracteres para posteriormente guardarlos en un token de tipo comentario al encontrar un **salto de línea** y **retorno de carro.**

  Al encontrar una situación favorable para la creación de un token con todos los caracteres acumulados, se procede a llamar al método **createToken**, el cual se encarga de **validar la entrada recibida desde searchMatches** en base a una serie de condiciones, como si la entrada es un texto en blanco, si el analizador estaba configurado en modo texto o de llamar a los métodos de validación de constantes o números si no se encontraron coincidencias locales.

  En caso de que el lexema no haya sido categorizado con éxito, se procede a llamar a un segundo método de reconocimiento, llamado **recognizeSeparators.** En esencia es similar al método searchMatches, pero su finalidad es distinta, ya que se encarga de separar tokens en base a **coincidencias** con el **diccionario**, para luego

llamar nuevamente al método createToken con los posibles tokens identificados y separados.

Cuando se ha creado un token de manera exitosa, el programa llama al método generateAnalysisReport, quién se encarga de recorrer toda la lista de Tokens para guardar sus características dentro de un string “reporte” con el formato adecuado para ser posteriormente escrito en un archivo **.csv**. Este tipo de archivo tiene la característica de representar una tabla similar a las tablas de las hojas de cálculo pero en texto plano, separando cada celda por una coma (,). Por esto mismo, para evitar problemas con el formato, cuando se intenta guardar un lexema “**,**” se reemplaza automáticamente por “**-comma-**”. Además, procede a llamar al método addText del componente gráfico textEditor para colorearlo.

- **FileGenerator:** Esta clase representa a un objeto que sirve de “impresora”, pues su único método -generate- se encarga de guardar un archivo y guardar contenido en él en base a un texto de entrada, la dirección que tendrá el archivo, la extensión que tendrá y el nombre con el que se guardará. En caso de que el archivo ya exista, eliminará el archivo existente para reemplazarlo con la nueva escritura.

- **FileReader:** Lee un archivo con un path especificado y luego devuelve el texto contenido en él.

- **TokenPlotter:** Manda a generar el gráfico con **Graphviz**. Crea una cadena con la estructura de código interpretable por Graphviz en base a las propiedades del token, luego la imprime con un File Generator en extensión **.dot** para pasarla al graficador por medio de un ProcessBuilder.

  Su método **plot** reconoce el tipo de token y configura el archivo para que sea coloreado correctamente con los criterios dados en el documento de la práctica. Además, establece el nombre del grafo con el tipo de token y el lexema que contiene.

INTERFAZ GRÁFICA - FRONTEND

- **Principal Frame:** Únicamente es el frame principal. Contiene un **botón analizar** que llama al método searchMatches del analizador usando como argumento input el texto contenido en uno de sus paneles, aparte de un **botón Abrir archivo** que únicamente carga el texto contenido en un archivo de texto especificado mediante un JFileChooser.

- **TextEditor:** Panel que contiene un JTextEditor, utilizado ya que este elemento de swing si permite cambiar el color de un texto de manera progresiva. Posee un método **addText** que agrega texto con un estilo previamente definido.
- **GraphViewer:** inicialmente era un panel al cual se le agregaban las imágenes generadas por Graphviz, pero debido a diversos inconvenientes con la ejecución del programa java y graphviz (java ejecuta sus tareas considerablemente más rápido que graphviz) decidí que sea un JTextEditor, ya que este es capaz de interpretar un

HTML muy básico. Por este motivo, el programa genera dos tipos de gráficos, uno con HTML (el cual puede evaluarse correctamente en un navegador) y otro con Graphviz.

- **ReportViewer:** Únicamente posee una tabla a la cual se le van añadiendo filas en base al contenido analizado. Esta tabla es -**independiente-** del archivo csv generado por el programa.

4. DIAGRAMA DE CLASES