

Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Clase : Organización de Lenguajes y
Compiladores 1

Sección N

Auxiliar: Jose Puac



Manual de Tecnico Proyecto JPR

Mynor Alison Isai Saban Che

201800516

Division del proyecto

La división del proyecto fue en cuatro partes la cual fueron las siguientes:



Expresiones:

aquí se tienen todos los tipos de datos, también las operaciones aritméticas, relacionales y lógicas.

Instrucciones:

aquí se encuentra la magia del proyecto, ya que se utilizó el método interprete, también en esta carpeta se encuentra todas las clases que se usó para el analizador

Tabla Árbol:

aquí están todas las clases importantes para el analizador, los

cuales son el árbol, tabla de símbolos, nodo para el ast .

Nativas: aquí se encuentra todas las funciones nativas del lenguaje

Editor

Proyecto 1 - Fase 1

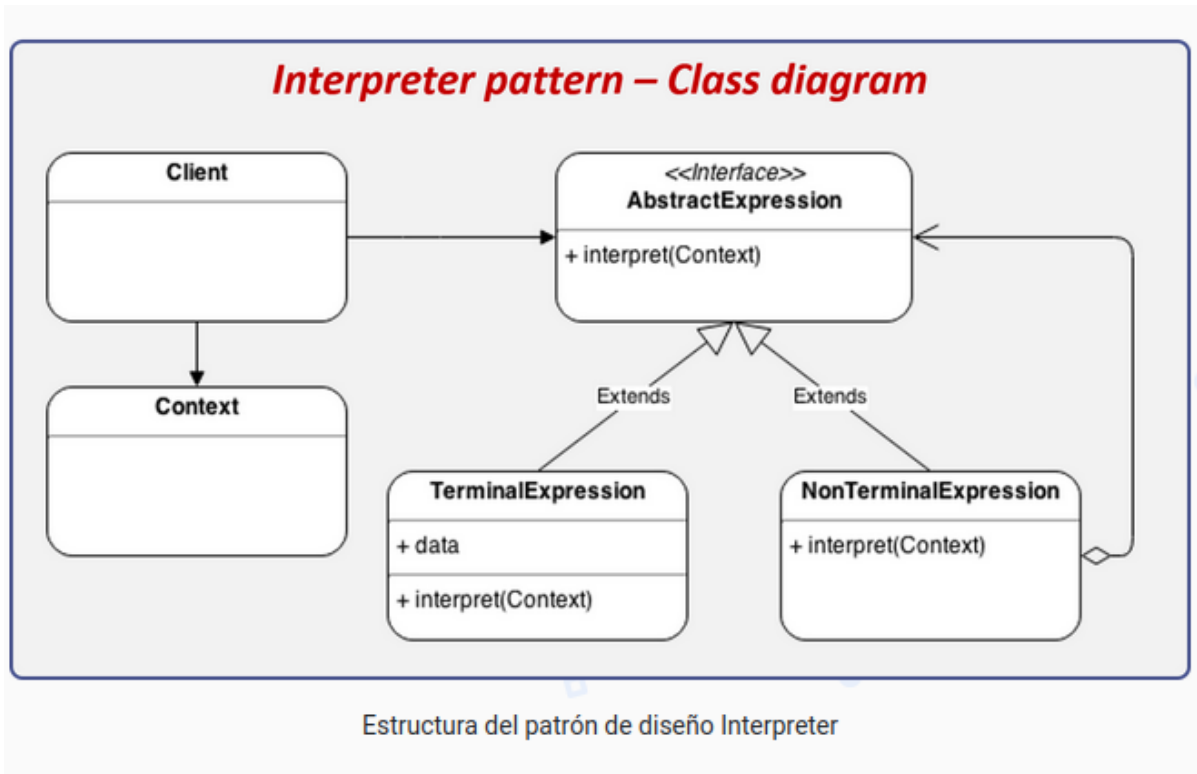
```
92 97")
93
94 # IMPRIMIENDO CLASES
95
96 for(var j = 0; j < 4; j++){
97     print("El nombre del alumno es: " + Clases[j][0]) # imprimiendo el no
98     print("El carnet del alumno es: " + Clases[j][2]) # imprimiendo el ca
99     print("La edad del alumno es: " + Clases[j][3]) # imprimiendo la ed
100     print("La nota del alumno es: " + Clases[j][4]) # imprimiendo la no
101     print("La nota del alumno es: " + Clases[j][4]) # imprimiendo la no
102 }
103
104 }
105 }
106
107 func CasteosYMas(){
108     print("Validando Continue")
109     var cont = 1
110     while(cont < 11){
```

```
INICIO DEL PROGRAMA
Ingrese su nombre:
Bienvenido mynor
Validando Continue
El valor del ciclo con continue es: 1
El valor del ciclo con continue es: 2
El valor del ciclo con continue es: 3
El valor del ciclo con continue es: 4
El valor del ciclo con continue es: 5
Se salta esta iteracion
El valor del ciclo con continue es: 7
El valor del ciclo con continue es: 8
El valor del ciclo con continue es: 9
El valor del ciclo con continue es: 10
CASTEOS
String
String
String
String
2
String
50
String
String
```

Identificador	Tipo	Tipo	Entorno	Valor	Línea	Columna
nombre	ARREGLO	CADENA	MAIN	mynor	14	9
punteorecursividad	ARREGLO	ENTERO	MAIN	3	31	9
discos	ARREGLO	ENTERO	MAIN	3	48	9
origen	ARREGLO	ENTERO	MAIN	1	49	9
auxiliar	ARREGLO	ENTERO	MAIN	2	50	9
destino	ARREGLO	ENTERO	MAIN	3	51	9
vectornumeros	ARREGLO	ENTERO	MAIN	0 1 2 7 9 9 15 26 44 51 73	65	11
clases	ARREGLO	CADENA	GLOBAL	{{GABRIEL ORLANDO AJSI'	1	12

Patron interprete

el patron interprete es el que hace la magia en este proyecto porque el es el que llama todas las funciones, pero como se usa este patrón, a continuación se muestra una imagen de cómo se estructura y una breve explicación



- **Cliente:** Actor que dispara la ejecución del intérprete.
- **Context:** Objeto con información global que será utilizada por el intérprete para leer y almacenar información global entre todas las clases que conforman el patrón, este es enviado al interpretar el cual lo replica por toda la estructura.
- **Abstract Expression:** Interface que define la estructura mínima de una expresión.
- **Terminal Expresión:** Se refiere a expresiones que no tienen más continuidad y al ser evaluadas o interpretadas terminan la ejecución de esa rama. Estas expresiones marcan el final de la ejecución de un sub-árbol de la expresión.
- **NonTerminal Expression:** Son expresiones compuestas y dentro de ellas existen más expresiones que deben ser evaluadas. Estas estructuras son interpretadas utilizando recursividad hasta llegar a una expresión Terminal.

- Fuente :

<https://reactiveprogramming.io/blog/es/patrones-de-diseno/interpreter>

Para nuestro caso el patrón intérprete nos soluciono la vida. En la imagen siguiente se coloca la clase que sirvió como intérprete, la cual poseía los métodos abstractos necesarios para obtener dicho patron

```
Instrucciones > Instruccion.py
1  from TablaArbol.Excepcion import Excepcion
2  from TablaArbol.Tipo import OperadorAritmetico, TIPO
3  from abc import ABC, abstractmethod
4  from TablaArbol.Simbolo import Simbolo
5  from TablaArbol.ts import TablaSimbolos
6
7
8  class Instruccion(ABC):
9      def __init__(self, fila, columna):
10         self.fila = fila
11         self.columna = columna
12         self.arreglo=False
13         super().__init__()
14
15         @abstractmethod
16         def interpretar(self, tree, table):
17             pass
18
19         @abstractmethod
20         def getNodo(self):
21             pass
```

un ejemplo claro de como se utilizo el patron interprete se muestra en la siguiente trozo de codigo

```
def interpretar(self, tree, table):
    value = self.expression.interpretar(tree, table) # Valor a asignar a la variable
    if isinstance(value, Excepcion): return value

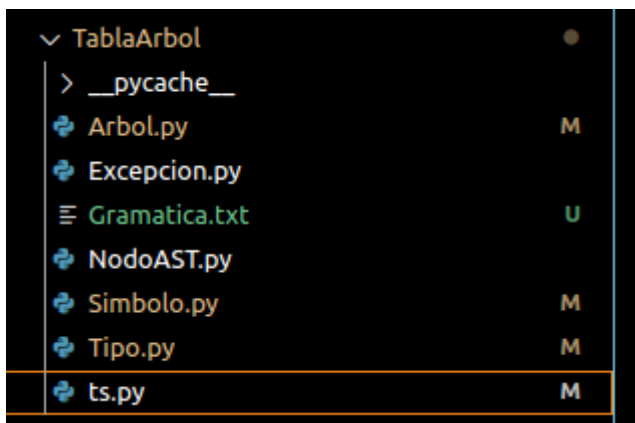
    simbolo = Simbolo(self.identificador.lower(), self.expression.tipo, self.arreglo, self)
    result = table.actualizarTabla(simbolo)

    if isinstance(result, Excepcion): return result

    return None
```

aquí se aprecia todo lo que puede abarcar el método, como se puede apreciar este trozo de código depende completamente de la instrucción (nuestro intérprete “clase maestra”).

También existen las siguientes clases para poder ejecutar el intérprete.



- la clase arbol es algo esencial ya que por medio de esta se guardan las instrucciones, excepciones que tiene el código, funciones, la tabla de símbolos “algo muy importante para nuestro analizador”.
- la clase excepcion lo único que hace es un objeto de tipo excepción y poder guardarlo en el árbol, para cuando se termine la ejecución de todo solo se llama para mostrar las excepciones que ocurrieron dentro de todo el programa
- la clase NodoAST, este su trabajo es ser nodo como su nombre lo indica, este grafica los nodos del árbol ast, este es objeto de tipo NODOAST, para cuando se grafique es más flexible modelar el árbol.

- la clase simbolo, su tabajo como su nombre lo indica es guardar todos los simbolos que tiene nuestro editor al momento de ser analizado, este es flexible, ya que acepta todo tipo de dato
- la clase tipo, solamente es una clase numerada con la capacidad de ordenar de manera facil y flexible los tipos de datos que maneja nuestro analizador.

Grafico de Arbol AST

```
def recorrerAST(self, idPadre, nodoPadre):
    for hijo in nodoPadre.getNodos_Hijos():
        nombreHijo = "n" + str(self.contador)

        print(hijo.getValor())
        try:
            self.dot += nombreHijo + "[label=\"\" + hijo.valor.replace(\"\\\", \"\\\\\") + "\"];\\n"
        except:
            self.dot += nombreHijo + "[label=\"\" + str(hijo.valor)+ "\"];\\n"

        print(hijo.valor)
        self.dot += nombreHijo + "[label=\"\" + hijo.getValor().replace(\"\\\", \"\\\\\") + "\"];\\n"

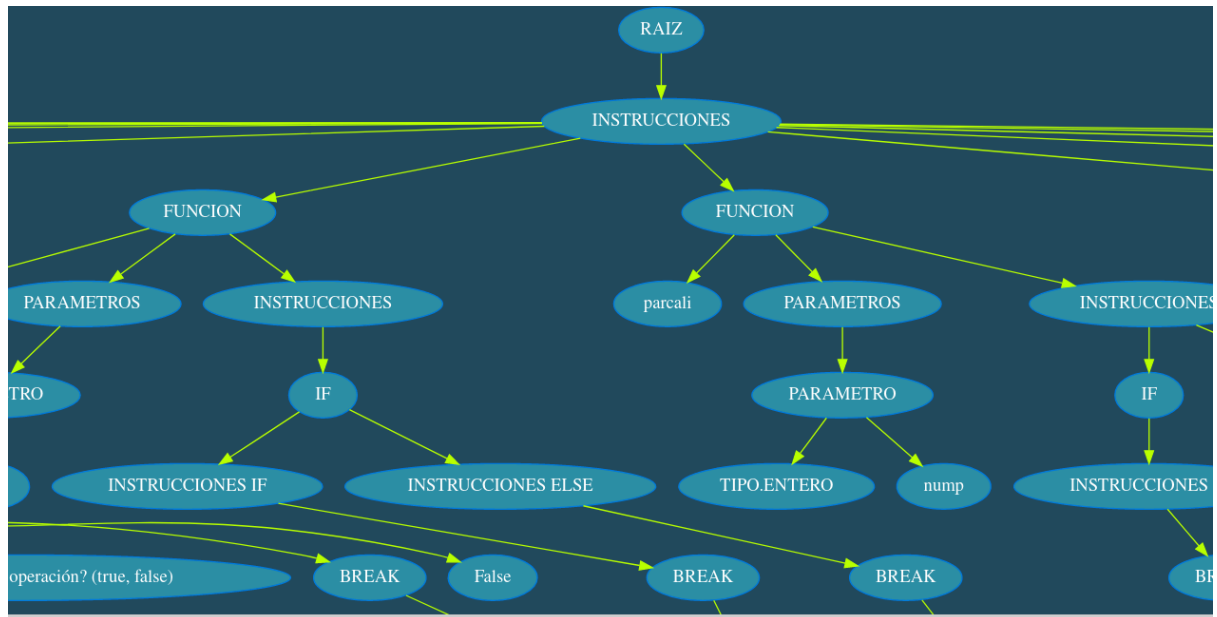
def recorrerAST(self, idPadre, nodoPadre):
    for hijo in nodoPadre.getNodos_Hijos():
        nombreHijo = "n" + str(self.contador)

        print(hijo.getValor())
        try:
            self.dot += nombreHijo + "[label=\"\" + hijo.valor.replace(\"\\\", \"\\\\\") + "\"];\\n"
        except:
            self.dot += nombreHijo + "[label=\"\" + str(hijo.valor)+ "\"];\\n"

        print(hijo.valor)
        self.dot += nombreHijo + "[label=\"\" + hijo.getValor().replace(\"\\\", \"\\\\\") + "\"];\\n"

        self.dot += idPadre + "->" + nombreHijo + ";\\n"
        self.contador += 1
        self.recorrerAST(nombreHijo, hijo)
```

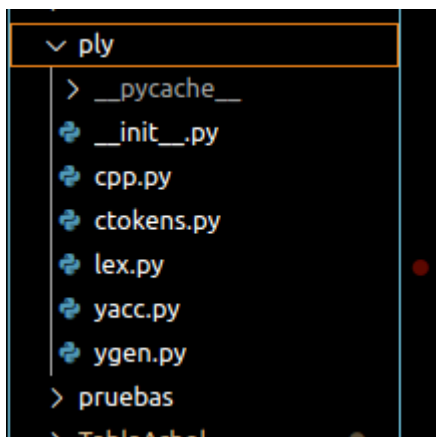
Se muestra el trozo de código que tiene la funcionalidad de poder graficar nuestro árbol, esto es posible gracias a la recursividad y la forma de trabajar con el patrón interprete, este ha sido muy útil para realizar el árbol.



esta es una pequeña representación del árbol generado por la recursividad y el patron interprete

Analizador PLY de python

para realizar este programa o editor de texto se utilizó el analizador ascendente PLY de python. Dentro de nuestra proyecto existe esta carpeta que comprueba con certeza que se utilizo este analizador



equipo donde se tabajo el proyecto

Gráficos	Intel® HD Graphics 405 (BSW)
Capacidad del disco	500.1 GB
Nombre del SO	Ubuntu 20.04.2 LTS
Tipo de SO	64 bits
Versión de GNOME	3.36.8
Sistema de ventanas	X11
Actualizaciones de software	>

