Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Organización de lenguajes y compiladores 2

Carlos Andree Avalos Soto
201408580

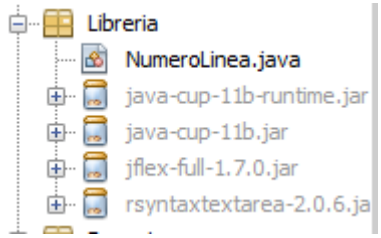
Manual técnico

IDE RMB

Guatemala, 20 de diciembre de 2019

Librerías

Para la realización de este proyecto se utilizaron librerías que nos ayudan al análisis léxico, tanto como al análisis sintáctico, así también se utilizó la librería Rsyntaxis para el editor de código.



Método de implementación

Para el método de ejecución se utilizo por medio de la interfaz, a esto nos referimos a que las clases heredan del padre, en este caso la clase instrucción emplea 5 métodos, los cuales deben venir en todos los hijos que hereden de él, esto da a paso que se puedan juntar varias instrucciones de diferente tipo, pero con los mismos métodos, como el de ejecutar y recolectar.

```
public interface Instruccion {
    public int getLine();
    public int getColumn();
    public Object Ejecutar(TablaDeSimbolos ts);
    public void Recolectar(TablaDeSimbolos ts);
    public Tipo getType();
}
```

Se hicieron dos corridas para este compilador, la primera que es la recolección de todas las variables globales, métodos, funciones, importes y demás cosas que se declaren globalmente.

Así también se implementó el método ejecutar, el cual ejecuta las operaciones locales, como instrucciones dentro de un evento al dar clic, o instrucciones adentro de un método.

Tabla de Símbolos

Para la implementación de la tabla de símbolos se utilizo una clase símbolo, la cual es la que contiene diferentes atributos los cuales nos ayudaran a la fácil detección a la hora de ejecución y recolección de identificadores, métodos, etc.

Símbolo

```java
public class Simbolo {

    private TipoSimbolo tipo;//Aqui puede venir (numero,int),(numero,double) etc
    private String id;//el id de la variable
    private Object valor;//el valor de la variable
    private Instruccion contenido;
    private boolean declarado;//si tiene valor o no.
    private Tipo tipo_instruccion;
    private ArrayList<String> referencias = new ArrayList<>(); // id a los cual esta referida
    private ArrayList<String> referidos; // id a los cual esta referida
    private boolean PorReferencia; // si esta referenciada
    private String clase;

    public Simbolo copy(Simbolo sim) {
        this.tipo = sim.getTipo();
        this.id = sim.getId();
        this.valor = sim.getValor();
        this.contenido = sim.getContenido();
        this.declarado = false;
        this.tipo_instruccion = sim.getTipo_instruccion();
        this.referencias = sim.getReferencias();
        this.PorReferencia = sim.PorReferencia;
        this.clase = sim.clase;
        return this;
    }
```

El método copy es para copiar todos los valores de un símbolo a otro sin copiar el espacio de memoria.

Tabla de símbolos

```java
public void setPadre(TablaDeSimbolos tablapadre) {
    this.padre = tablapadre;
}
// -------------------------------METODO PARA DAR VALOR SIMPLE-----------------------------

public void setValor(String id, Object valor) {

    setValor(id, valor, this);
}

private void setValor(String id, Object valor, TablaDeSimbolos padre) {
    for (Simbolo item : padre) {
        if (item.getTipo().getTipo() != null) {
            if (item.getId().equals(id)) {
                try {
                    switch (item.getTipo_instruccion()) {
                        case VARIABLE:
                            switch (item.getTipo().getTipo()) {
                                case Entero:
                                    item.setValor((int) Double.parseDouble(valor = null | ));
                                    return;
                                case Decimal:
                                    item.setValor(Double.parseDouble(valor.toString()));
                                    return;
                                case Char:
                                    item.setValor((char) valor.toString().charAt(0));
                                    return;
                                case Cadena:
                                    item.setValor((String) valor.toString());
                                    return;
                                case Bool:
                                    item.setValor(Boolean.valueOf(valor.toString()));
                                    return;
                                case String:
                                    item.setValor(valor.toString());
                                    return;
                                default:
                                    //deberia tirar error ya que no existe el ID
                                    break;
                            }
                        case CONSTANTE:
                            //error porque no se puede cambiar el valor
                            System.out.println("No se puede cambiar valor a la variable: \'" + id + "\' es una constante.");
                            break;
                        case COMPONENTE:
                            item.setValor(valor);
                        case FUNCION:
```

Ejemplos de asignación de un valor a una variable, el cual verifica su tipo de símbolo y lo castea a su valor.

Tambien tenemos el método setPadre, lo cual anexa un ámbito anterior.

```java
//-----------------------------------------------------------------------------------------------
public void recorrerPadres(TablaDeSimbolos padre, int nivel) {
    System.out.println("-----SU NIVEL:" + nivel + "-----");
    if (padre.getPadre() != null) {
        recorrerPadres(padre.getPadre(), nivel + 1);
    }
}

//-----------------------------------------------------------------------------------------------
public void setValorByIndex(int index, Object valor) {
    setValorByIndex(index, valor, this);
}

private void setValorByIndex(int index, Object valor, TablaDeSimbolos tsPadre) {

    Simbolo item = tsPadre.get(index);
    if (item.getTipo_instruccion() == Tipo.ARREGLO) {
        item.setValor(valor);
        return;
    }
    switch (item.getTipo().getTipo()) {
        case Entero:
            item.setValor((int) Double.parseDouble(valor.toString()));
            break;
        case Decimal:
            item.setValor(Double.parseDouble(valor.toString()));
            break;
        case Char:
            item.setValor((char) valor.toString().charAt(0));
            break;
        case Cadena:
            item.setValor((String) valor.toString());
            break;
        case Bool:
            item.setValor(Boolean.valueOf(valor.toString()));
            break;
        case Struct:
            item.setValor(valor);
        default:
            //deberia tirar error ya que no existe el ID
            break;
    }
}

//-----------------------------------------------------------------------------------------------
```

Metodo que sirve para poder asignar un valor a los parámetros por medio de su index.

```java
//-----------------------------------------------------------------------------------------
public Simbolo getSimbolo(String id) {
    return getSimbolo(id, this);
}

private Simbolo getSimbolo(String id, TablaDeSimbolos padre) {
    for (Simbolo item : padre) {
        if (item.getId().equals(id)) {
            return item;
        }
    }
    if (padre.getPadre() != null) {
        return getSimbolo(id, padre.getPadre());
    } else {
        return null;
    }
}
```

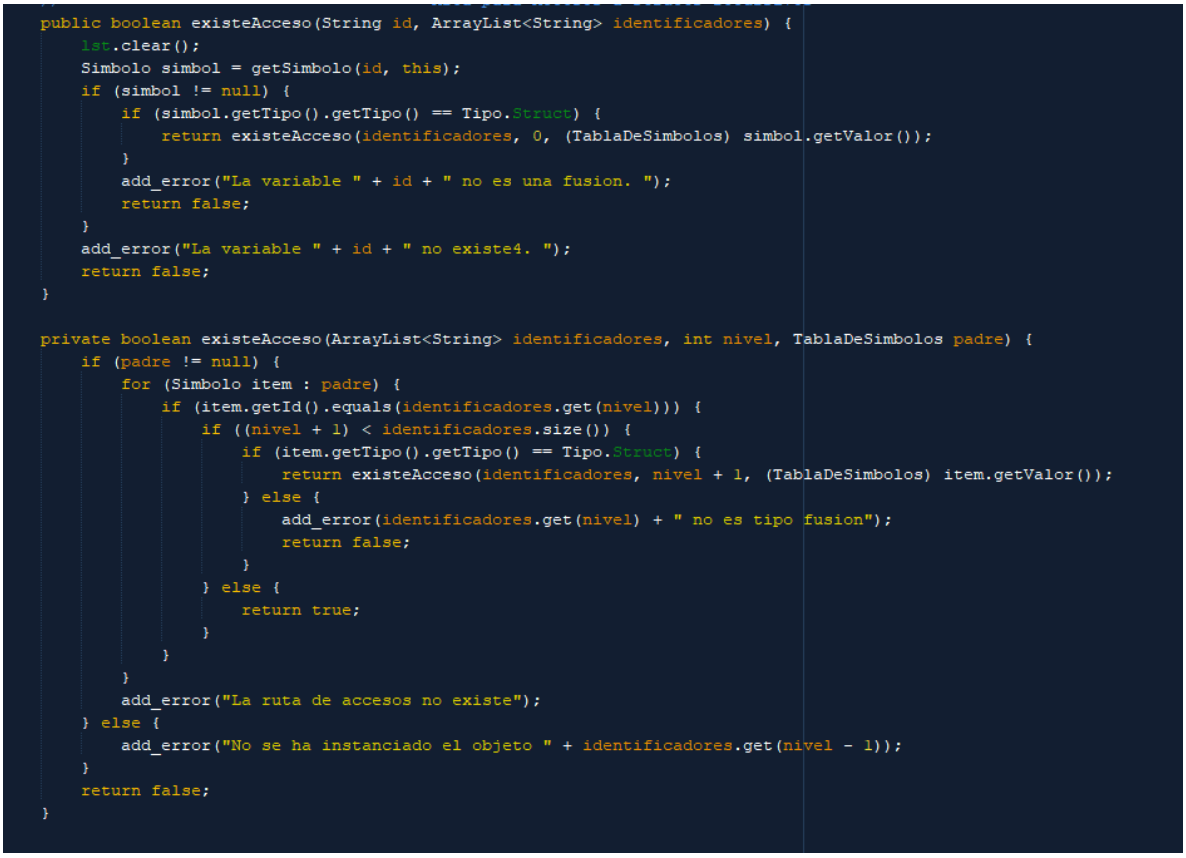
Método que sirve para obtener un símbolo por medio de su id, es muy útil para cuando solo se conoce el id de una variable.

```java
public Object getValor(String id) {
    return getValor(id, this);
}

private Object getValor(String id, TablaDeSimbolos tsPadre) {
    for (Simbolo item : tsPadre) {
        if (item.getId().equals(id)) {
            return item.getValor();
        }
    }
    if (tsPadre.getPadre() != null) {
        return getValor(id, tsPadre.getPadre());
    }
    return null;
}
```

Método que sirve para obtener el valor asignado a una variable.

```java
public Instruccion getContenido(String id) {
    return getContenido(id, this);
}

private Instruccion getContenido(String id, TablaDeSimbolos tsPadre) {
    for (Simbolo item : tsPadre) {
        if (item.getId().equals(id)) {
            return item.getContenido();
        }
    }
    if (tsPadre.getPadre() != null) {
        return getContenido(id, tsPadre.getPadre());
    }
    return null;
}
```

Método realizado para poder obtener las instrucciones dentro de un método, función, evento, etc.



```
public boolean existeAcceso(String id, ArrayList<String> identificadores) {
    lst.clear();
    Simbolo simbol = getSimbolo(id, this);
    if (simbol != null) {
        if (simbol.getTipo().getTipo() == Tipo.Struct) {
            return existeAcceso(identificadores, 0, (TablaDeSimbolos) simbol.getValor());
        }
        add_error("La variable " + id + " no es una fusion. ");
        return false;
    }
    add_error("La variable " + id + " no existe4. ");
    return false;
}

private boolean existeAcceso(ArrayList<String> identificadores, int nivel, TablaDeSimbolos padre) {
    if (padre != null) {
        for (Simbolo item : padre) {
            if (item.getId().equals(identificadores.get(nivel))) {
                if ((nivel + 1) < identificadores.size()) {
                    if (item.getTipo().getTipo() == Tipo.Struct) {
                        return existeAcceso(identificadores, nivel + 1, (TablaDeSimbolos) item.getValor());
                    } else {
                        add_error(identificadores.get(nivel) + " no es tipo fusion");
                        return false;
                    }
                } else {
                    return true;
                }
            }
        }
        add_error("La ruta de accesos no existe");
    } else {
        add_error("No se ha instanciado el objeto " + identificadores.get(nivel - 1));
    }
    return false;
}
```

Método utilizado para acceder a las propiedades un objeto, tales como **nodo.siguiente.**

```java
public boolean existeSimboloAmbienteActual(String id) {
    return existeSimboloAmbienteActual(id, this);
}

private boolean existeSimboloAmbienteActual(String id, TablaDeSimbolos padre) {
    for (Simbolo item : padre) {
        if (item.getId().equals(id)) {
            return true;
        }
    }
    return false;
}

public void removeReferencias(String id) {
    Simbolo sim = getSimbolo(id);
    for (String item : sim.getReferencias()) {
        Simbolo aux = getSimbolo(item);
        aux.removeReferencia(id);
    }
    sim.setReferencias(new ArrayList<>());
}
```

Método empleado para verificar si el símbolo existe en el ambiente local.

Tipos de retorno

Tipo Break

Tipo que da como resultado la ejecución de “romper”

Tipo Seguir

Tipo que da como resultado la ejecución de “seguir”

Tipo Return

Tipo que da como resultado la ejecución de “return id”

Ejemplos de instrucciones

Declaración de variables

```java
/**
 * Metodo implementado de instruccion que sirve para recolectar variables
 * del ambito global
 *
 * @param ts tabla de simbolos global
 */
@Override
public void Recolectar(TablaDeSimbolos ts) {

    if (ts.getPadre() == null) {
        if (!ts.existeSimbolo(id)) {
            ts.add(new Simbolo(new TipoSimbolo(tipo_simbolo, ""), id,0, Tipo.VARIABLE));

        } else {
            Principal.add_error("La varaible \'" + id + "\' ya esta declarada","Semantico",line,column);
            //aqui va el mensaje de error que ya esta declarada la variable en el ambito
            return;
        }
    }

    if (asignacion != null) {
        boolean pass = (boolean) asignacion.Ejecutar(ts);
        if (pass) {
            // si se asigno el valor;
        } else {
            ts.eliminarSimbolo(id);
        }
    }

}
```

Asignación de Variables

```java
@Override
public Object Ejecutar(TablaDeSimbolos ts) {
    Object resultado = valor.Ejecutar(ts);
    if (ts.existeSimbolo(id)) {
        Object aux = resultado;
        if (ts.asignValor(id, resultado)) {
            ts.setValor(id, aux);
            local = ts.getEntorno(id);
            if (ts.equals(ts)) {
                local = ts;
            }
            if (local.existReferencia(id)) {
                ArrayList<String> lst = local.getListaReferencia(id);
                if (lst.size() > 1) {
                    lst.forEach((item) -> {
                        local.setValor(item, resultado);
                    });
                }
            }

            return true;
        } else {
            //error porque no se puede hacer el casteo explicito
            Principal.add_error("No es posible hacer el casteo ", "Semantico", line, column);
            return false;
        }
    } else {
        Principal.add_error("No existe la variable: " + id, "Semantico", line, column);
        return false;
    }
}
```

Declaracion de Fusion

```
@Override
public void Recolectar(TablaDeSimbolos ts) {
    if (ts.getPadre() == null) {
        if (valor == null) {
            if (!tipo.equals(tipo2)) {
                Principal.add_error("El tipo: " + tipo2 + " con la fusion: " + tipo2, "Semantico", line, column);
                return;
            }
            if (ts.getPadre() == null) {
                if (ts.existeSimbolo(tipo)) {
                    if (!ts.existeSimbolo(id)) {
                        TablaDeSimbolos local = ts.get_struct(tipo);

                        TablaDeSimbolos nueva = new TablaDeSimbolos();

                        for (Simbolo item : local) {
                            Simbolo new_simbol = new Simbolo("", ")");
                            nueva.add(new_simbol.copy(item));
                        }
                        nueva.nombre = id;
                        ts.add(new Simbolo(new TipoSimbolo(Tipo.Struct, tipo), id, Tipo.FUSION));
                        ts.setValor(id, nueva);

                    } else {
                        Principal.add_error("La variable \'" + id + "\' ya esta declarada", "Semantico", line, column);

                    }
                } else {
                    Principal.add_error("No existe el tipo: " + tipo, "Semantico", line, column);

                }
            }
        } else {
```

Declaración de arreglo

```
@Override
public void Recolectar(TablaDeSimbolos ts) {
    if (ts.getPadre() == null) {
        if (!ts.existeSimbolo(id)) {
            ts.add(new Simbolo(new TipoSimbolo(tipo_dato, "arreglo"), id, Tipo.ARREGLO));
            Arbol arbol_declaracion = new Arbol();
            arbol_declaracion.crearArbol(num_dimensiones);
            ts.setValor(id, arbol_declaracion);
        } else {
            Principal.add_error("La variable \'" + id + "\' ya esta declarada", "Semantico", line, column);
            //aqui va el mensaje de error que ya esta declarada la variable en el ambito
        }
    }
}
```

Declaración de Componente

```
@Override
public void Recolectar(TablaDeSimbolos ts) {
    if (ts.getPadre() == null) {
        if (!ts.existeSimbolo(id)) {
            ts.add(new Simbolo(new TipoSimbolo(tipo_simbolo, ""), id, 0, Tipo.COMPONENTE));

        } else {
            Principal.add_error("La varaible \'" + id + "\' ya esta declarada", "Semantico", line, column);
            //aqui va el mensaje de error que ya esta declarada la variable en el ambito
            return;
        }
    }

}
```

Declaración de Objeto

```
@Override
public void Recolectar(TablaDeSimbolos ts) {
    if (ts.getPadre() == null) {
        if (!ts.existeSimbolo(id)) {
            ts.add(new Simbolo(new TipoSimbolo(Tipo.Struct, tipo), id, Tipo.FUSION));
            ts.setValor(id, null);
        } else {
            Principal.add_error("La variable \'" + id + "\' ya esta declarada", "Semantico", line, column);
            //aqui va el mensaje de error que ya esta declarada la variable en el ambito
        }
    }
}
```

Declaración de Método

```
  @Override
  public void Recolectar(TablaDeSimbolos ts) {

      if (ts.getPadre() == null) {
          if (!ts.existeSimbolo(id)) {
              ts.add(new Simbolo(new TipoSimbolo(Tipo.METODO, ""), id, this, Tipo.METODO));
          } else {
              Principal.add_error("La funcion \'" + id + "\' ya esta declarada", "Semantico", line, column);
              return;
              //aqui va el mensaje de error que ya esta declarada la variable en el ambito
          }
          parametros.forEach((item) -> {
              item.Recolectar(local);
          });
      }
  }
```

Declaración de Función

```
@Override
public void Recolectar(TablaDeSimbolos ts) {

    if (ts.getPadre() == null) {
        if (!ts.existeSimbolo(id)) {
            if (tipo_funcion == TipoFuncion.NORMAL) {
                ts.add(new Simbolo(new TipoSimbolo(tipo_simbolo, tipo_struct), id, this, Tipo.FUNCION));
            } else if (tipo_funcion == TipoFuncion.STRUCT) {
                if (ts.existeSimbolo(tipo_struct)) {
                    Simbolo sim = ts.getSimbolo(tipo_struct);
                    if (sim.getTipo_instruccion() == Tipo.OBJETO) {
                        ts.add(new Simbolo(new TipoSimbolo(tipo_simbolo, tipo_struct), id, this, Tipo.FUNCION));
                    } else {
                        Principal.add_error("El tipo: " + tipo_struct + " no es una fusion.", "Semantico", line, column);
                        return;
                    }
                } else {
                    Principal.add_error("El tipo: \'" + tipo_struct + "\' no esta declarada", "Semantico", line, column);
                    return;
                }
            } else if (tipo_funcion == TipoFuncion.ARREGLO) {
                arreglo = new Arbol();
                arreglo.setNivel(num_niveles.size());
                ts.add(new Simbolo(new TipoSimbolo(tipo_simbolo, "arreglo"), id, arreglo, this, Tipo.FUNCION));
            }
        } else {
            Principal.add_error("La funcion \'" + id + "\' ya esta declarada", "Semantico", line, column);
            //aqui va el mensaje de error que ya esta declarada la variable en el ambito
            return;
        }
        parametros.forEach((item) -> {
            item.Recolectar(local2);
        });
    }
}
```

Sentencia If

```java
@Override
public Object Ejecutar(TablaDeSimbolos ts) {
    TablaDeSimbolos tabla_local = new TablaDeSimbolos();
    tabla_local.setPadre(ts);
    Operacion expresion = instruccion_if.getExpresion();
    LinkedList<Sentencia_IF> else_if = instruccion_if.getElse_if();
    LinkedList<Instruccion> sentencias_if = instruccion_if.getSentencias_if();
    Sentencia_IF sentencia_else = new Sentencia_IF();
    for (Instruccion item : else_if) {
        Sentencia_IF objeto = (Sentencia_IF) item;
        if (objeto.tipo_sentencia == Sentencia_IF.TipoIf.TIPO_ELSE) {
            sentencia_else = objeto;
            break;
        }
    }

    try {
        if ((boolean) expresion.Ejecutar(ts)) {
            for (Instruccion item : sentencias_if) {
                switch (item.getType()) {
                    case BREAK:
                        return new Tipo_Retorno(Tipo.ETIQUETA_RETURN, null);
                    case SEGUIR:
                        return new Tipo_Retorno(Tipo.ETIQUETA_SIGUE, null);
                    case RETURN:
                        return item.Ejecutar(tabla_local);
                    default:
                        Object result = item.Ejecutar(tabla_local);
                        if (result != null) {
                            try {
                                Tipo_Retorno etiqueta = (Tipo_Retorno) result;
                                return etiqueta;
                            } catch (Exception e) {
                            }
                        }
                        break;
                }
            }
```

Sentencia For

```
@Override
public Object Ejecutar(TablaDeSimbolos ts) {
    TablaDeSimbolos tabla_local = new TablaDeSimbolos();
    tabla_local.setPadre(ts);

    declaracion.Ejecutar(tabla_local);
    boolean siguiente = false;

    while ((boolean) expresion.Ejecutar(tabla_local)) {

        for (Instruccion item : contenido) {

            switch (item.getType()) {
                case BREAK:
                    return null;
                case SEGUIR:
                    return new Tipo_Retorno(Tipo.ETIQUETA_SIGUE, null);
                case RETURN:
                    return item.Ejecutar(tabla_local);
                default:
                    Object result = item.Ejecutar(tabla_local);
                    if (result != null) {
                        try {
                            Tipo_Retorno etiqueta = (Tipo_Retorno) result;
                            if (etiqueta.getEtiqueta() == Tipo.ETIQUETA_RETURN) {
                                return etiqueta;
                            }
                            if (etiqueta.getEtiqueta() == Tipo.ETIQUETA_SIGUE) {
                                siguiente = true;
                                break;
                            } else {
                                return null;
                            }
                        } catch (Exception e) {
                        }
                    }
                    break;
            }
            if (siguiente) {
                return null;
            }
        }
        operador.Ejecutar(tabla_local);
    }

    return null;
}
```

Sentencia While

```
@Override
public Object Ejecutar(TablaDeSimbolos ts) {

    int maximo_iteraciones = 0;
    boolean siguiente = false;
    while ((boolean) expresion.Ejecutar(ts)) {
        if (maximo_iteraciones == 5000) {
            Principal.add_error("Stack over flow", "Semantico", line, column);
            return null;
        }
        TablaDeSimbolos tabla_local = new TablaDeSimbolos();
        tabla_local.setPadre(ts);
        for (Instruccion item : contenido) {

            switch (item.getType()) {
                case BREAK:
                    return null;
                case SEGUIR:
                    return new Tipo_Retorno(Tipo.ETIQUETA_SIGUE, null);
                case RETURN:
                    return item.Ejecutar(tabla_local);
                default:
                    Object result = item.Ejecutar(tabla_local);
                    if (result != null) {
                        try {
                            Tipo_Retorno etiqueta = (Tipo_Retorno) result;
                            if (etiqueta.getEtiqueta() == Tipo.ETIQUETA_RETURN) {
                                return etiqueta;
                            }
                            if (etiqueta.getEtiqueta() == Tipo.ETIQUETA_SIGUE) {
                                siguiente = true;
                                break;
                            } else {
                                return null;
                            }
                        } catch (Exception e) {
                        }
                    }
                    break;
            }
            if(siguiente){
                return null;
            }
        }
        maximo_iteraciones++;
    }
```

Operación Logicas, Relacionales y Numericas

Tipos de Operación

```
public enum TipoOperacion {
    SUMA,
    RESTA,
    MULTIPLICACION,
    DIVISION,
    NEGATIVO,
    NUMERO,
    MODULAR,
    POTENCIA,
    IDENTIFICADOR,
    CADENA,
    MAYOR_QUE,
    MENOR_QUE,
    MAYOR_IGUAL,
    MENOR_IGUAL,
    IGUAL_IGUAL,
    DIFERENTE,
    CONCATENACION,
    CARACTER,
    RSTRING,
    BOOL,
    NOT,
    AND,
    OR,
    FUNCION,
    PESODE,
    ACCESO_ARREGLO,
    ACCESO_STRUCT,
    NULO,
    MASMAS,
    MENOSMENOS,
    EQUALS,
    ATXT,
    AENT,
    ADEC,
    ENTERO,
    DECIMAL,
    GETTEXTO,
    GETANCHO,
    GETALTO,
    GETPOS
}
```

Constructores para la clase operación

```
public Operacion(Operacion operadorIzq, Operacion operadorDer, TipoOperacion tipo, int line, int column) {
    this.operadorDer = operadorDer;
    this.operadorIzq = operadorIzq;
    this.tipo = tipo;
    this.line = line;
    this.column = column;
}

public Operacion(Operacion operadorIzq, TipoOperacion tipo, int line, int column) {
    this.operadorIzq = operadorIzq;
    this.tipo = tipo;
    this.line = line;
    this.column = column;
}

public Operacion(Object valor, TipoOperacion tipo, int line, int column) {
    this.tipo = tipo;
    this.valor = valor;
    this.line = line;
    this.column = column;
}

public Operacion(String valor, int line, int column) {
    this.valor = valor;
    this.line = line;
    this.column = column;
    this.tipo = TipoOperacion.NUMERO;
}

public Operacion(String id_objeto, LinkedList<Operacion> accesos, TipoOperacion tipo, int line, int column) {
    this.id_objeto = id_objeto;
    this.accesos = accesos;
    this.tipo = tipo;
    this.line = line;
    this.column = column;
}

public Operacion(String id_objeto, ArrayList<String> identificadores, TipoOperacion tipo, int line, int column) {
    this.id_objeto = id_objeto;
    this.identificadores = identificadores;
    this.tipo = tipo;
    this.line = line;
    this.column = column;
}
```

Ejecucion de Operaciones

```
switch (tipo) {
    case NULO:
        return null;
    case DIVISION:
        return (Double) Double.parseDouble(operadorIzq.Ejecutar(ts).toString()) / (Double) Double.parseDouble(operadorDer.Ejecutar(ts).toString());
    case MULTIPLICACION:
        return (Double) Double.parseDouble(operadorIzq.Ejecutar(ts).toString()) * (Double) Double.parseDouble(operadorDer.Ejecutar(ts).toString());
    case RESTA:
        return (Double) Double.parseDouble(operadorIzq.Ejecutar(ts).toString()) - (Double) Double.parseDouble(operadorDer.Ejecutar(ts).toString());
    case SUMA:
        return (Double) Double.parseDouble(operadorIzq.Ejecutar(ts).toString()) + (Double) Double.parseDouble(operadorDer.Ejecutar(ts).toString());
    case NEGATIVO:
        return (Double) Double.parseDouble(operadorIzq.Ejecutar(ts).toString()) * -1;
    case DECIMAL:
        return Double.parseDouble(valor.toString());
    case ENTERO:
        return (int) Double.parseDouble(valor.toString());
    case IDENTIFICADOR:
        Object val = ts.getValor(valor.toString());
        Simbolo sim = ts.getSimbolo(valor.toString());
        if (sim != null) {
            if (sim.getTipo_instruccion() == Tipo.ESTRUCT) {
                Arbol aux = (Arbol) val;
                return aux;
            } else if (sim.getTipo_instruccion() == Tipo.FUSION) {
                String tipo_struct = sim.getTipo().getAsignado();
                if (ts.existeSimbolo(tipo_struct)) {
                    return val;
                }
            } else {
                return val;
            }
        }
        return null;
    case CADENA:
        Arbol arbol_aux2 = new Arbol();
        arbol_aux2.convertirString(valor.toString());
        return arbol_aux2;
    case MAYOR_QUE:
        return ((Double) Double.parseDouble(operadorIzq.Ejecutar(ts).toString())) > ((Double) Double.parseDouble(operadorDer.Ejecutar(ts).toString()));
    case MENOR_QUE:
        return ((Double) Double.parseDouble(operadorIzq.Ejecutar(ts).toString())) < ((Double) Double.parseDouble(operadorDer.Ejecutar(ts).toString()));
    case MAYOR_IGUAL:
        return ((Double) Double.parseDouble(operadorIzq.Ejecutar(ts).toString())) >= ((Double) Double.parseDouble(operadorDer.Ejecutar(ts).toString()));
    case MENOR_IGUAL:
        return ((Double) Double.parseDouble(operadorIzq.Ejecutar(ts).toString())) <= ((Double) Double.parseDouble(operadorDer.Ejecutar(ts).toString()));
    case IGUAL_IGUAL:
```

Método de Impresión

```
String salida = cadena[0];
Object val = null;
for (int i = 1; i < cadena.length; i++) {
    String trozo = cadena[i];
    letra = trozo.charAt(0);
    switch (letra) {
        case 's':
            try {
                Arbol aux = (Arbol) valores.get(i - 1);
                aux.print();
                salida += aux.getSalida();
                break;
            } catch (Exception ex) {
                Principal.add_error("Tipo " + valores.get(i - 1) + " con chr[]", "Semantico", line, this.column);
                return null;
            }
        case 'e':
            try {
                val = (int) Integer.parseInt(valores.get(i - 1).toString());
                salida += val.toString();
                break;
            } catch (NumberFormatException ex) {
                Principal.add_error("Tipo " + valores.get(i - 1) + " con entero", "Semantico", line, column);
                return null;
            }
        case 'd':
            try {
                val = (double) Double.parseDouble(valores.get(i - 1).toString());
                salida += val.toString();
            } catch (NumberFormatException ex) {
                Principal.add_error("Tipo " + valores.get(i - 1) + " con decimal", "Semantico", line, column);
                return null;
            }
            break;
        case 'c':
            try {
                val = (char) valores.get(i - 1).toString().charAt(0);
                salida += val.toString();
            } catch (NumberFormatException ex) {
                Principal.add_error("Tipo " + valores.get(i - 1) + " con chr", "Semantico", line, column);
                return null;
            }
            break;
        case 'b':
            try {
                val = (boolean) Boolean.parseBoolean(valores.get(i - 1).toString());
                salida += val.toString();
            } catch (NumberFormatException ex) {
```

Definir

```
@Override
public void Recolectar(TablaDeSimbolos ts) {
    boolean declarada = false;
    Tipo tipo_simbolo = ts.getTipo(valor);

    if (ts.getPadre() == null) {
        if (!ts.existeSimbolo(id)) {
            ts.add(new Simbolo(new TipoSimbolo(tipo_simbolo, ""), id, valor.Ejecutar(ts), Tipo.CONSTANTE));
        } else {
            Principal.add_error("La variable \'" + id + "\' ya esta declarada", "Semantico",line,column);
            //aqui va el mensaje de error que ya esta declarada la variable en el ambito
        }
    }
    String resultado = valor.Ejecutar(ts).toString();
}
```