Manual Técnico

Brandon Josué Pinto Méndez

201930236

Analizador Léxico:

El analizador Léxico de manera resumida se encarga de generar tokens según las reglas que le instruyamos, a continuación, veremos los métodos que se encargan de esto en el programa.

analisisLexico(String texto):

Este método es el que se encarga del análisis léxico el cual genera los tokens que se utilizan para el análisis sintáctico más adelante, se puede seguir el funcionamiento del método por los comentarios en este.

```
texto += " ";
        char temporal;
        tokenRecopilado = new ArrayList<>();
        tokenErrores = new ArrayList<>();
        while (posicion < texto.length()-1) {
            temporal = texto.charAt(posicion);
            String apoyoCadena = "";
            //if para validar si es comentario:
            if(EsComentario(temporal)){
                inicio = columna;
                while(temporal!='\n' && posicion < texto.length()-1){
                    apoyoCadena += temporal;
                    posicion++;
                    columna++;
                    temporal = texto.charAt(posicion);
                }
                reporteComentario(apoyoCadena);
                fila++;
                columna=0;
                posicion++;
                //if para validar una cadena de comilla doble
            }else if(esComillasDobles(temporal)){
                boolean seCierra = true;
                inicio = columna;
                posicion++;
                columna++;
                temporal = texto.charAt(posicion);
                while(temporal!=""&& posicion < texto.length()-1){
                    //aqui se maneja si no se cerro la doble comilla
                    if(temporal=='\n'){
                        reporteErrorCadena(apoyoCadena);
                        fila++;
                        columna = 0;
                        posicion++;
                        temporal = "";
                        seCierra = false;
                    }else{
```

```
                apoyoCadena += temporal;
                posicion++;
                columna++;
                temporal = texto.charAt(posicion);
            }
        }
        if(seCierra){
            reporteCadena(apoyoCadena);
            posicion++;
        }
    //if para validar una cadena de comilla simple
    }else if(esComillaSimple(temporal)){
        inicio = columna;
        posicion++;
        columna++;
        temporal = texto.charAt(posicion);
        boolean seCierra = true;
        while(temporal!='\'' && posicion < texto.length()-1){
            //aqui se maneja si no se cerro la comilla simple
            if(temporal=='\n'){
                reporteErrorCadena(apoyoCadena);
                fila++;
                columna = 0;
                posicion++;
                temporal = "";
                seCierra = false;
            }else{
                apoyoCadena += temporal;
                posicion++;
                columna++;
                temporal = texto.charAt(posicion);
            }
        }
        if(seCierra){
            reporteCadena(apoyoCadena);
        }
        //if encargado de manejar identificadores xd
    }else if ((temporal >= 'a' && temporal <= 'z')||(temporal >= 'A' && temporal <= 'Z')||temporal=='_'){
        inicio = columna;
        boolean sinError = true; //variable que define que el identificador es legal
        //ciclo el cual se encarga de acabar todo cuando encuentre un espacio
        while(temporal!=' ' && posicion < texto.length()-1 && temporal!='\n' && clasificadorCaracter(temporal)==-1){
            //condicion la cual revisa que sea un caracter valido para identificador
            if((temporal >= 'a' && temporal <= 'z')||(temporal >= 'A' && temporal <= 'Z')||temporal=='_'||(temporal >= '0' && temporal <= '9')){
```

```
                apoyoCadena += temporal;
                posicion++;
                columna++;
                temporal = texto.charAt(posicion);
              //aqui se maneja si se encuentra un caracter invalido para un identificador:
              }else if(temporal=='\r'||temporal=='\t'||temporal=='\f'||temporal=='\b'){
                posicion++;
                temporal = texto.charAt(posicion);
              }else if(temporal=='.'){
                break;
              }else{
                reporteErrorIdentificadores(apoyoCadena);
                //este ciclo while se encarga de llevar la lectura hasta el siguiente espacio o
fin.
                while(temporal!=' ' && posicion < texto.length()-1){
                  if(temporal=='\n'){
                    fila++;
                    columna=0;
                    temporal = ' ';
                  }else{
                    columna++;
                    posicion++;
                    temporal = texto.charAt(posicion);
                  }
                }
                sinError = false;
              }
            }//si no hay errores se crea el reporte correctamente, de lo contrario no
            if(sinError){
              //aqui verificamos que el identificador no sea un operador logico:
              if(esLogico(apoyoCadena)!=-1){
                reporteOperadorLogico(apoyoCadena);
                //aqui verificamos que la cadena no sea un boolean:
              }else if(esBolean(apoyoCadena)){
                reporteBooleans(apoyoCadena);
                //aqui verificamos que la cadena no sea una palabra reservada:
              }else if(esPalabraReservada(apoyoCadena)){
                reportePalabraReservada(apoyoCadena);
                //aqui se crea el reporte como identificador:
              }else{
                reporteIdentificadores(apoyoCadena);
              }
            }
            //condición para los enteros y decimales:
          }else if(temporal >= '0' && temporal <= '9'){
            inicio = columna;
            boolean sinError = true; //variable que define que el entero o decimal es legal
```

```
            boolean sinErrorDec = true;//variable para manejar error de no poner algo después
del .
            boolean esDecimal = false; //variable para definir si es decimal o no
            //ciclo el cual se encarga de acabar todo cuando encuentre un espacio
            while(temporal!=' '&& posicion < texto.length()-1 && temporal!='\n'){
               //aqui se mete si es un numero entero
               if((temporal >= '0' && temporal <= '9')){
                  apoyoCadena += temporal;
                  if(esDecimal){
                     sinErrorDec = true;
                  }
                  posicion++;
                  columna++;
                  temporal = texto.charAt(posicion);
               //aqui se mete si es un . lo cual lo vuelve un decimal, solo se puede una vez
               }else if(temporal == '.' && !(esDecimal)){
                  esDecimal = true;
                  sinErrorDec = false;
                  apoyoCadena += temporal;
                  posicion++;
                  columna++;
                  temporal = texto.charAt(posicion);
               }else if(!(clasificadorCaracter(temporal)==-1)||temporal=='\n'||temporal=='\r'){
                  break;
               }else{
                  apoyoCadena += temporal;
                  //aqui va si hay un error en la cadena, lo cual genera un error dependiendo si
es decimal o no y acaba el ciclo
                  sinError = false;
                 //este ciclo while se encarga de llevar la lectura hasta el siguiente espacio o fin.
                  while(temporal!=' ' && posicion < texto.length()-1){
                     if(temporal=='\n'){
                        fila++;
                        columna=0;
                        temporal = ' ';
                     }else{
                        columna++;
                        posicion++;
                        temporal = texto.charAt(posicion);
                        apoyoCadena += temporal;
                     }

               }
               if(esDecimal){
                     reporteErrorDecimales(apoyoCadena);
                  }else{
                     reporteErrorEntero(apoyoCadena);
```

```
                }
            }//si no hay errores se crea el reporte correctamente, de lo contrario n
            }
            if(sinError && sinErrorDec){

if(esDecimal){reporteDecimales(apoyoCadena);}else{reporteEnteros(apoyoCadena);}
                }else if(!sinErrorDec){
                   reporteErrorDecimales(apoyoCadena);
                }
          }else if(!(clasificadorCaracter(temporal)==-1)){
             inicio = columna;
             //if para validar si es caracter especial
             String Ayudador = "";
             boolean llave = false;
             try{
                Ayudador                                                                    =
Character.toString(temporal)+Character.toString(texto.charAt(posicion + 1));
                llave = EsCaracterDoble(Ayudador);
             }catch(StringIndexOutOfBoundsException e){
                System.out.println("No es doble");
             }
             //Validación si es caracter de 2 o 1 caracteres, dependiendo de esto incluye el
caracter como una u otra cosa
             if(llave){//entra aqui cuando el caracter es doble
                reporteCaracterDoble(Ayudador);
                columna++;
                columna++;
                posicion++;
                posicion++;
             }else{//entra aqui si el caracter es solo uno
                reporteCaracter(temporal);
                posicion++;
                columna++;
             }
          //if para si hay salto de linea
          }else if(temporal == '\n'){
             columna = 0;
             fila++;
             posicion++;
          }else if(temporal == ' '){
             columna++;
             posicion++;
          }else if(temporal=='\r'||temporal=='\t'||temporal=='\f'||temporal=='\b'){
             posicion++;
          }else{
             inicio = columna;
             reporteErrorNoIdentificado(temporal);
```

```
            columna++;
            posicion++;
        }
    }
```

clasificadorCaracter(char carac):

Este método se encarga de regresar un entero según el carácter que le llegue, con este se consigue clasificar los chars y luego crear los reportes.

```
if(carac == '+'){
        return 1;
    }else if(carac == '-'){
        return 2;
    }else if(carac == '/'){
        return 3;
    }else if(carac == '%'){
        return 4;
    }else if(carac == '*'){
        return 5;
    }else if(carac == '>'){
        return 6;
    }else if(carac == '<'){
        return 7;
    }else if(carac == '='){
        return 8;
    }else if(carac == '('){
        return 9;
    }else if(carac == ')'){
        return 10;
    }else if(carac == '{'){
        return 11;
    }else if(carac == '}'){
        return 12;
    }else if(carac == '['){
        return 13;
    }else if(carac == ']'){
        return 14;
    }else if(carac == ','){
        return 15;
    }else if(carac == ';'){
        return 16;
    }else if(carac == ':'){
        return 17;
    }else if(carac == '.'){
        return 18;
    }else if(carac == '!'){
```

```
            return 19;
        }else{
            return -1;
        }
```

reporteCaracter(char Carac):

Método que se encarga de crear el reporte por cada carácter, utiliza un switch para clasificar según lo que llegue, este método depende del método anterior.

```
Simbolos token;
        switch(clasificadorCaracter(Carac)){
            case 1:
                token = new
Simbolos("Aritméticos","Suma",Character.toString(Carac),fila,columna,inicio);
                tokenRecopilado.add(token);
                break;
            case 2:
                token = new
Simbolos("Aritméticos","Resta",Character.toString(Carac),fila,columna,inicio);
                tokenRecopilado.add(token);
                break;
            case 3:
                token = new
Simbolos("Aritméticos","División",Character.toString(Carac),fila,columna,inicio);
                tokenRecopilado.add(token);
                break;
            case 4:
                token = new
Simbolos("Aritméticos","Módulo",Character.toString(Carac),fila,columna,inicio);
                tokenRecopilado.add(token);
                break;
            case 5:
                token = new
Simbolos("Aritméticos","Multiplicación",Character.toString(Carac),fila,columna,inicio);
                tokenRecopilado.add(token);
                break;
            case 6:
                token = new Simbolos("Comparación","Mayor que
",Character.toString(Carac),fila,columna,inicio);
                tokenRecopilado.add(token);
                break;
            case 7:
                token = new Simbolos("Comparación","Menor que
",Character.toString(Carac),fila,columna,inicio);
                tokenRecopilado.add(token);
                break;
```

```
            case 8:
                token = new
Simbolos("Asignación","Asignación",Character.toString(Carac),fila,columna,inicio);
                tokenRecopilado.add(token);
                break;
            case 9:
                token = new
Simbolos("Otros","Paréntesis",Character.toString(Carac),fila,columna,inicio);
                tokenRecopilado.add(token);
                break;
            case 10:
                token = new
Simbolos("Otros","Paréntesis",Character.toString(Carac),fila,columna,inicio);
                tokenRecopilado.add(token);
                break;
            case 11:
                token = new
Simbolos("Otros","Llaves",Character.toString(Carac),fila,columna,inicio);
                tokenRecopilado.add(token);
                break;
            case 12:
                token = new
Simbolos("Otros","Llaves",Character.toString(Carac),fila,columna,inicio);
                tokenRecopilado.add(token);
                break;
            case 13:
                token = new
Simbolos("Otros","Corchetes",Character.toString(Carac),fila,columna,inicio);
                tokenRecopilado.add(token);
                break;
            case 14:
                token = new
Simbolos("Otros","Corchetes",Character.toString(Carac),fila,columna,inicio);
                tokenRecopilado.add(token);
                break;
            case 15:
                token = new
Simbolos("Otros","Coma",Character.toString(Carac),fila,columna,inicio);
                tokenRecopilado.add(token);
                break;
            case 16:
                token = new Simbolos("Otros","Punto y
coma",Character.toString(Carac),fila,columna,inicio);
                tokenRecopilado.add(token);
                break;
            case 17:
```

```
            token = new Simbolos("Otros","Dos
puntos",Character.toString(Carac),fila,columna,inicio);
            tokenRecopilado.add(token);
            break;
          case 18:
            token = new
Simbolos("Otros","Punto",Character.toString(Carac),fila,columna,inicio);
            tokenRecopilado.add(token);
            break;
          default:
            System.out.println("wtf xd");
            break;
      }
```

Estos son algunos de los métodos utilizados para realizar el análisis Léxico, en general se utiliza la misma dinámica en más métodos los cuales no se tocarán en este Manual.

Símbolos

Esta clase se encargaba de guardar la información de cada token, como pudimos ver era muy importante a lo largo del Analizador Léxico, en general la estructura de esta clase era la siguiente:

```
 public Simbolos(String tipoToken, String patron ,String lexema, int fila, int columna, int
inicio) {
      this.TipoDeToken = tipoToken;
      this.patron = patron;
      this.Lexema = lexema;
      this.fila = fila;
      this.columna = columna;
      this.inicio = inicio;
   }
```

Analizador Sintáctico:

El analizador sintáctico se encarga de validar que los tokens estén cumplan con el patrón que se requiere, como veremos a continuación con los métodos los cuales se encargan de este, antes de esto miraremos la manera en que se organizaron algunas expresiones hecha como guía para organizar los diferentes patrones.

Expresión para Declaración:

Declaración |ID ASIGNACION EXPRESION

|ID OPERADOR/ ASIGNACION EXPRESION

EXPRESION -> |Constante(Operador Y Constante)*;

| ID

|Otros(ID|Constantes|Otros)*

Expresión para Métodos:

Métodos |PALABRA_CLAVE ID OTROS ((ID|Constantes)(Otros)?)* OTROS OTROS

Expresión para invocación de método:

Métodos |ID OTROS ((ID|Constantes)(Otros)?)* OTROS

Expresión para Condicionales

Condicional -> | if expresión [condicional expresión]*

EXPRESION -> | boolean

| ID

| (Entero|Double|ID) Comparación (Entero|Double|ID);

Expresión para For

For -> | For Expresión

Expresion - > | ID in (ID|Range ((ID|Constante)(Otros)?)*)OTROS

Expresión para While

While -> | While Expresión

Expresión -> | (ID|Constante) Condicional (ID|Constante)

Métodos del analizador sintáctico:

bloques():

Este método se encargaba de recibir los tokens creados por el analizador Léxico y clasificarlos a través de un sistema de bloques el cual indicaba si el código era parte de un mismo conjunto o no, esto a través del funcionamiento de Python el cual maneja su separación a través de tabulaciones o espacios, con esto se separaron los bloques según la sangría de las instrucciones.

```
boolean AumentarBloque = false;
      int Fila;
      while (posicion < Tokens.size()) {
         Simbolos elemento = Tokens.get(posicion);
         Fila = elemento.getFila();
         if(elemento.getInicio()==0){
            ArrayList<TablaSintactica> ListaDeReportes = Reportes.getreporteRecopilado();
            ArrayList<TablaSintactica> ElementosEnElBloque = new ArrayList<>();
            for (TablaSintactica objeto : ListaDeReportes) {
               if (objeto.getbloque() == bloque) {
                  ElementosEnElBloque.add(objeto);
               }
            //condición si el bloque tiene contenido adentro aqui se hace la distinción para saber
que hay que hacer según su calsificación
            }
            //Es nuevo:
            if(ElementosEnElBloque.isEmpty()){
               analisisSintactico(Fila);
            //si ya existen elementos en ese bloque comprueba si un nuevo elemento pertenece
y si no crea un nuevo bloque
            }else{
               //comprobar si es if
               if(ElementosEnElBloque.get(0).getSimbolo().equals("if")){

if(elemento.getLexema().equals("elif")||elemento.getLexema().equals("else")){
                     AumentarBloque = false;
                  }else{
                     AumentarBloque = true;
                  }
               }else if(ElementosEnElBloque.get(0).getSimbolo().equals("for")){
                  if(elemento.getLexema().equals("else")){
                     AumentarBloque = false;
                  }else{
                     AumentarBloque = true;
                  }
               }else{
                  AumentarBloque = true;
               }
               //if encargado de cerrar el bloque
               if(AumentarBloque){
                  bloque++;
```

```
            }
            analisisSintactico(Fila);
          }
          //aqui va el comportamiento si es contenido de un bloque:
        }else{
          analisisSintactico(Fila);
        }
      }
```

analisisSintactico(int Fila):

El anterior método depende del presente método, este se encarga de la clasificación de cada token y arreglar el comportamiento de este, también de generar los reportes de error y funcionamiento de este.

```
Simbolos elemento = Tokens.get(posicion);
        String Simbolo = elemento.getLexema();
        //condición si el elemento es ID: Declaración |ID ASIGNACION
EXPRESION | ID ARIMETICO/ASIGNACION EXPRESION
        if(elemento.getTipoToken().equals("ID")){
          posicion++;
          elemento = Tokens.get(posicion);
          //Si es asignación de una
          if(elemento.getPatron().equals("Asignación")){
            posicion++;
            String Cadena = "";
            elemento = Tokens.get(posicion);
            //If donde se define el comportamiento de la expresión
            //EXPRESION ->  |Constante(Operador Y Constante)*;
            if(elemento.getTipoToken().equals("Constantes")){
              boolean ContieneID = false;
              boolean Asignacion = false;
              boolean ExpresionCorrecta = false;
              boolean esCadena = false;
              boolean esOtro = false;
              boolean huboError = false;
              boolean seCerroLaCadena = false;
              boolean esTernario = false;
              boolean multiplesValores = false;
              boolean CerrarCiclo = true;
              String Tipo = elemento.getPatron();
              String Cad = "";
              if(elemento.getPatron().equals("booleanas")){
                huboError = true;
                Reportes.reporteAsignacion(Simbolo,Tipo, elemento.getLexema(),
elemento.getFila(),elemento.getColumna(),bloque,"Asignación");
                posicion++;
```

```
                }else
if(elemento.getPatron().equals("Entero")||elemento.getPatron().equals("Decimal")||element
o.getTipoToken().equals("ID")){
                    Cad += elemento.getLexema();
                    if(elemento.getTipoToken().equals("ID")){
                       ContieneID = true;
                    }
                       posicion++;
                       elemento = Tokens.get(posicion);

if((elemento.getTipoToken().equals("Aritméticos")||elemento.getTipoToken().equals("Com
paración"))&& Fila == elemento.getFila()){
                    while (posicion < Tokens.size() && Fila == elemento.getFila() &&
CerrarCiclo) {
                       elemento = Tokens.get(posicion);
                       if(Fila != elemento.getFila()){}else{
                          posicion++;

if(elemento.getPatron().equals("Entero")||elemento.getPatron().equals("Decimal")||element
o.getTipoToken().equals("ID")){
                             if(elemento.getTipoToken().equals("ID")){ContieneID = true;}
                             Cad += " "+elemento.getLexema();
                             ExpresionCorrecta = true;
                          }else if(elemento.getTipoToken().equals("Aritméticos")){
                             Cad += " "+elemento.getLexema();
                             ExpresionCorrecta = false;
                          }else if(elemento.getTipoToken().equals("Comparación")){
                             Asignacion = true;
                             ExpresionCorrecta = false;
                          }else if(elemento.getTipoToken().equals("Comentario")){
                             posicion++;
                          }else{
                             huboError = true;
                             Reporte s.reporteErrorAsignacion(2,elemento.getFila()          ,
elemento.getColumna(),bloque,"Asignación");
                             posicion++;
                             CerrarCiclo = false;
                          }
                       }
                    }
                    }else{
                       huboError = true;
                       if(ContieneID){Reportes.reporteAsignacion(Simbolo,Tipo,
"Undefined",                Tokens.get(posicion-1).getFila(),Tokens.get(posicion-
1).getColumna(),bloque,"Asignación");
```

```
                    }else{Reportes.reporteAsignacion(Simbolo,Tipo,Tokens.get(posicion-
1).getLexema(), Tokens.get(posicion-1).getFila(),Tokens.get(posicion-
1).getColumna(),bloque,"Asignación");}
                }
                if(huboError){}else{
                    if(ExpresionCorrecta){
                        if(ContieneID){
                            Reportes.reporteAsignacion(Simbolo,Tipo, "Undefined",
elemento.getFila(),elemento.getColumna(),bloque,"Asignación");
                        }else{
                            if(Asignacion){
                                Reportes.reporteAsignacion(Simbolo,Tipo, "Undefined",
elemento.getFila(),elemento.getColumna(),bloque,"Asignación");
                            }else{
                                Reportes.reporteAsignacion(Simbolo,Tipo,
Double.toString(calc.evaluarExpresion(Cad)),
elemento.getFila(),elemento.getColumna(),bloque,"Asignación");
                            }
                        }

}else{Reportes.reporteErrorAsignacion(3,elemento.getFila(),elemento.getColumna(),bloqu
e,"Asignación");}
                }
            }else{
                while (posicion < Tokens.size() && Fila == elemento.getFila()) {
                elemento = Tokens.get(posicion);
                if(Fila != elemento.getFila()){}else{
                    multiplesValores=true;
                    if(elemento.getPatron().equals("Cadena")){
                        esCadena = true;
                    }else
if(elemento.getPatron().equals("Entero")||elemento.getPatron().equals("Decimal")){
                        esOtro = true;
                    }
                    if(esTernario){
                        if(elemento.getLexema().equals("**")){
                            Cadena += " ^";
                        }else{
                            Cadena += " "+elemento.getLexema();
                        }
                        if(elemento.getTipoToken().equals("Aritméticos")){
                            seCerroLaCadena = false;
                        }else if(elemento.getTipoToken().equals("Constantes")){
                            seCerroLaCadena = true;
                        }else if(elemento.getTipoToken().equals("Palabras
clave")||elemento.getTipoToken().equals("Comparación")){
                            esTernario = true;
```

```
                        seCerroLaCadena = false;
                    }
                    //se entra a esta condición si se combinan tipos cadena con cualquier
otro, lo cual haría un error de sintaxis
                }else if((esCadena && esOtro)||(!esCadena && !esOtro)){
                        Reportes.reporteErrorAsignacion(1,elemento.getFila()          ,
elemento.getColumna(),bloque,"Asignación");
                        huboError = true;
                        //bucle para seguir en la siguiente linea
                        CambiarDeLinea(Fila);
                        break;
                }else{
                    if(elemento.getLexema().equals("**")){
                        Cadena += " ^";
                    }else{
                        Cadena += " "+elemento.getLexema();
                    }
                    if(elemento.getTipoToken().equals("Aritméticos")){
                        seCerroLaCadena = false;
                    }else if(elemento.getTipoToken().equals("Constantes")){
                        seCerroLaCadena = true;
                    }else                    if(elemento.getTipoToken().equals("Palabras
clave")||elemento.getTipoToken().equals("Comparación")){
                        esTernario = true;
                        seCerroLaCadena = false;
                    }else{
                        if(!esTernario){
                            Reportes.reporteErrorAsignacion(2,elemento.getFila()          ,
elemento.getColumna(),bloque,"Asignación");
                            huboError = true;
                        //bucle para seguir en la siguiente linea
                            CambiarDeLinea(Fila);
                            break;
                        }else{

                        }
                    }
                }
                    posicion++;
                }
                }
            }
            //condicion final, si hubo un error anterior no hace nada, si es error de tipo 3
y si no crea el reporte
            if(huboError){

            }else if(seCerroLaCadena){
```

```
                    elemento = Tokens.get(posicion-1);
                    if(esTernario){
                        Reportes.reporteAsignacion(Simbolo,"Ternario",          "Undefined",
elemento.getFila(),elemento.getColumna(),bloque,"Ternario");
                    }else if(esCadena){
                        Reportes.reporteAsignacion(Simbolo,Tipo,                Cadena,
elemento.getFila(),elemento.getColumna(),bloque,"Asignación");
                    }else if(multiplesValores){
                        Reportes.reporteAsignacion(Simbolo,Tipo,
Double.toString(calc.evaluarExpresion(Cadena)),
elemento.getFila(),elemento.getColumna(),bloque,"Asignación");
                    }else{
                        Reportes.reporteAsignacion(Simbolo,Tipo,                Cadena,
elemento.getFila(),elemento.getColumna(),bloque,"Asignación");
                    }

                }else{

Reportes.reporteErrorAsignacion(3,elemento.getFila(),elemento.getColumna(),bloque,"Asi
gnación");
                }
                //| ID realizar esto si sobre tiempo
            }else if(false){

                //|Otros(ID|Constantes|Otros)*
            }else if(elemento.getTipoToken().equals("Otros")){
                String Abrio = elemento.getPatron();
                posicion++;
                boolean cerro = false;
                while (posicion < Tokens.size() && Fila == elemento.getFila()) {
                    elemento = Tokens.get(posicion);
                    if(Fila != elemento.getFila()){
                        break;
                    }
                    Cadena += elemento.getLexema();
                    posicion++;
                    if(elemento.getPatron().equals(Abrio)){
                        cerro = true;
                    }
                }
                if(cerro){
                    elemento = Tokens.get(posicion-1);
                    Reportes.reporteAsignacion(Simbolo,"Arreglo",           "Undefined",
elemento.getFila(),elemento.getColumna(),bloque,"Asignación");
                }else{
```

```
Reportes.reporteErrorAsignacion(4,elemento.getFila(),elemento.getColumna(),bloque,"Asi
gnación");
                    }
                    //Error con el formato de la asignación
                }else{

Reportes.reporteErrorAsignacion(5,elemento.getFila(),elemento.getColumna(),bloque,"Asi
gnación");
                    CambiarDeLinea(Fila);
                }
            //si tiene un token arimetico antes:
            }else if(elemento.getTipoToken().equals("Aritméticos")){
                boolean cerrarCiclo = true;
                posicion++;
                elemento = Tokens.get(posicion);
                if(elemento.getTipoToken().equals("Asignación")){
                    //bucle para seguir en la siguiente linea
                    posicion++;
                    while (posicion < Tokens.size() && cerrarCiclo
&&!(elemento.getTipoToken().equals("Comentario"))) {
                        elemento = Tokens.get(posicion);
                        if(Fila != elemento.getFila()){cerrarCiclo = false;}else{
                        posicion++;
                        }
                    }
                    elemento = Tokens.get(posicion-1);
                    Reportes.reporteAsignacion(Simbolo,"Asignación",  "Undefined",
elemento.getFila(),elemento.getColumna(),bloque,"operadores");
                }else{

Reportes.reporteErrorAsignacion(6,elemento.getFila(),elemento.getColumna(),bloque,"Asi
gnación");
                    CambiarDeLinea(Fila);
                }
                //Condición para ver las invocaciones de metodos
            }else if(elemento.getLexema().equals("(")){
                String Cadena = "";
                boolean cerro = false;
                int PosCerrado = 0;
                posicion++;
                    //bucle para seguir en la siguiente linea
                    while (posicion < Tokens.size() && Fila == elemento.getFila() && !cerro) {
                        //condicional para evitar que se vaya una fila de más
                        elemento = Tokens.get(posicion);
                        if(Fila != elemento.getFila()){
                        }else{
```

```
                        if(elemento.getLexema().equals(")")){
                            cerro = true;
                            PosCerrado = posicion;
                        }else{
                            Cadena += elemento.getLexema();
                        }
                        posicion++;
                    }
                }
                elemento = Tokens.get(PosCerrado);
                if(cerro){
                    Reportes.reporteFuncion(Simbolo,"Funciones",                Cadena,
elemento.getFila(),elemento.getColumna(),bloque,"Invocaciones");
                }else{
                    elemento = Tokens.get(posicion-1);

Reportes.reporteErrorFuncion(1,elemento.getFila(),elemento.getColumna(),bloque,"Funcio
nes");
                }
            //condición para invocaciones multiples:
            }else if(elemento.getLexema().equals(",")){
                posicion++;
                boolean coma = false;
                while (posicion < Tokens.size() && Fila == elemento.getFila()) {
                    elemento = Tokens.get(posicion);
                    posicion++;
                        if(elemento.getLexema().equals(",")){
                            coma = false;
                        }else{
                            coma = true;
                        }
                }
                elemento = Tokens.get(posicion-1);
                if(coma){
                    Reportes.reporteAsignacion(Simbolo,"Asignación",            "Undefined",
elemento.getFila(),elemento.getColumna(),bloque,"Asignación");
                }
                //si no cumple con ninguno de los requisitos genera error sintactico:
            }else{

Reportes.reporteErrorAsignacion(7,elemento.getFila(),elemento.getColumna(),bloque,"Asi
gnación");
            }
            //para palabras clave:
          }else if(elemento.getTipoToken().equals("Palabras clave")){
            String palabra = elemento.getLexema();
            //Switch para el manejo de palabras claves:
```

```
            switch (palabra) {
               //para bloques IF y elif
               case "if":
               case "elif":
                  String Tipo = "boolean";
                  posicion++;
                  elemento = Tokens.get(posicion);
                  //EXPRESION ->      | boolean
                  if(elemento.getPatron().equals("booleanas")){
                     String Valor = elemento.getLexema();
                     posicion++;
                     elemento = Tokens.get(posicion);
                     if(elemento.getLexema().equals(":")){
                        posicion++;
                        elemento = Tokens.get(posicion);
                        if(Fila == elemento.getFila()){
                           //si existe un elemento en la misma linea ve si es un comentario si lo
es no pasa nada
                           if(elemento.getTipoToken().equals("Comentario")){
                              posicion++;
                           }else{

Reportes.reporteErrorCondicional(1,elemento.getFila(),elemento.getColumna(),bloque,"Co
ndicional");
                              CambiarDeLinea(Fila);
                           }
                        }else{
                           Reportes.reporteCondicional(Simbolo,Tipo,                Valor,
elemento.getFila(),elemento.getColumna(),bloque,"Condicional");
                        }
                     }else{
                        //error si no tiene : acabando la condicional

Reportes.reporteErrorCondicional(2,elemento.getFila(),elemento.getColumna(),bloque,"Co
ndicional");
                        CambiarDeLinea(Fila);
                     }
                  //| (Entero|Double|ID) Comparación (Entero|Double|ID) || ID
                  }else
if(elemento.getTipoToken().equals("ID")||elemento.getTipoToken().equals("Constantes")){
                     boolean HayId = false;
                     //condición para saber si entro con un id
                     if(elemento.getTipoToken().equals("ID")){
                        HayId = true;
                     }
                     posicion++;
                     elemento = Tokens.get(posicion);
```

```
                    //condición para saber si es por medio de comparación el boolean

if(elemento.getTipoToken().equals("Comparación")||elemento.getTipoToken().equals("Pala
bras clave")){
                        posicion++;
                        elemento = Tokens.get(posicion);
                        //aqui se recibe el siguiente valor ya sea constante o ID

if(elemento.getTipoToken().equals("ID")||elemento.getTipoToken().equals("Constantes")){
                            posicion++;
                            elemento = Tokens.get(posicion);
                            if(elemento.getTipoToken().equals("ID")){
                                HayId = true;
                            }
                            //se verifica que se cierre la sentencia con :
                            if(elemento.getLexema().equals(":")){
                               if(HayId){
                                    Reportes.reporteCondicional(Simbolo,Tipo,"Undefined",
elemento.getFila(),elemento.getColumna(),bloque,"Condicional");
                                }else{
                                    Reportes.reporteCondicional(Simbolo,Tipo,"Undefined",
elemento.getFila(),elemento.getColumna(),bloque,"Condicional");
                                }
                               posicion++;
                               elemento = Tokens.get(posicion);
                               //se verifica que el siguiente elemento no esté en la misma linea
                               if(Fila == elemento.getFila()){
                                  //si existe un elemento en la misma linea ve si es un comentario
si lo es no pasa nada
                                  if(elemento.getTipoToken().equals("Comentario")){
                                     posicion++;
                                  }else{
                                     //si el elemento no era un comentario tira un error

Reportes.reporteErrorCondicional(1,elemento.getFila(),elemento.getColumna(),bloque,"Co
ndicional");
                                     CambiarDeLinea(Fila);
                                  }
                               }
                               //error si no tiene : acabando la condicional
                            }else{

Reportes.reporteErrorCondicional(2,elemento.getFila(),elemento.getColumna(),bloque,"Co
ndicional");
                               CambiarDeLinea(Fila);
                            }
                            //error si recibe un elemento que no sea Id o constante
```

```
                    }else{

Reportes.reporteErrorCondicional(3,elemento.getFila(),elemento.getColumna(),bloque,"Co
ndicional");
                        CambiarDeLinea(Fila);
                    }
                    // si no recibe un signo de comparación prueba con la otra forma que
sería recibir : si no lo recibe tira error
                }else if(elemento.getLexema().equals(":")){
                    Reportes.reporteCondicional(Simbolo,Tipo,"Undefined",
elemento.getFila(),elemento.getColumna(),bloque,"Condicional");
                    CambiarDeLinea(Fila);
                }else{

Reportes.reporteErrorCondicional(4,elemento.getFila(),elemento.getColumna(),bloque,"Co
ndicional");
                    CambiarDeLinea(Fila);
                }
            }else if(elemento.getTipoToken().equals("Lógicos")){
                posicion++;
                elemento = Tokens.get(posicion);
                    //aqui se recibe el siguiente valor ya sea constante o ID

if(elemento.getTipoToken().equals("ID")||elemento.getTipoToken().equals("Constantes")){
                        posicion++;
                        elemento = Tokens.get(posicion);
                        //se verifica que se cierre la sentencia con :
                        if(elemento.getLexema().equals(":")){
                            Reportes.reporteCondicional(Simbolo,Tipo,"Undefined",
elemento.getFila(),elemento.getColumna(),bloque,"Condicional");
                            posicion++;
                        }else{

Reportes.reporteErrorCondicional(2,elemento.getFila(),elemento.getColumna(),bloque,"Co
ndicional");
                            CambiarDeLinea(Fila);
                        }
                    }else{

Reportes.reporteErrorCondicional(3,elemento.getFila(),elemento.getColumna(),bloque,"Co
ndicional");
                        CambiarDeLinea(Fila);
                    }
                }else{

Reportes.reporteErrorCondicional(5,elemento.getFila(),elemento.getColumna(),bloque,"Co
ndicional");
```

```
                    CambiarDeLinea(Fila);
                }
                break;
            case "def":
            //Métodos          |PALABRA_CLAVE ID OTROS ((ID|Constantes)(Otros)?)*
OTROS OTROS def 4325(1):
                boolean EsComa = false;
                boolean huboError = false;
                boolean Finalizador = true;
                String Cadena = "";
                posicion++;
                elemento = Tokens.get(posicion);
                if(elemento.getTipoToken().equals("ID")){
                    Simbolo = elemento.getLexema();
                    posicion++;
                    elemento = Tokens.get(posicion);
                    if(elemento.getLexema().equals("(")){
                        posicion++;
                        while (posicion < Tokens.size() && Fila ==
Tokens.get(posicion).getFila() && Finalizador) {
                            elemento = Tokens.get(posicion);

if(elemento.getTipoToken().equals("ID")||elemento.getTipoToken().equals("Constantes")){
                                EsComa = false;
                                Cadena += elemento.getLexema();
                                posicion++;
                            }else if(elemento.getLexema().equals(",")){
                                EsComa = true;
                                Cadena += elemento.getLexema();
                                posicion++;
                            }else if(elemento.getTipoToken().equals("Aritméticos")){
                                EsComa = true;
                                Cadena += elemento.getLexema();
                                posicion++;
                            }else if(elemento.getLexema().equals(")")){
                                posicion++;
                            }else if (elemento.getLexema().equals(":")){
                                Finalizador = false;
                                posicion++;
                            }else{
                                huboError = true;
                                //si viene algún tipo de token que no se esperaba:

Reportes.reporteErrorFuncion(4,elemento.getFila(),elemento.getColumna(),bloque,"Funcio
nes");
                                //bucle para seguir en la siguiente linea
                                CambiarDeLinea(Fila);
```

```
                        }
                    }
                    if(EsComa){
                        //No se armo el parametro como se esperaba:

Reportes.reporteErrorFuncion(5,elemento.getFila(),elemento.getColumna(),bloque,"Funcio
nes");
                    }else if(!huboError){
                        Reportes.reporteFuncion(Simbolo,"Funciones",  Cadena,
Tokens.get(posicion-1).getFila(),Tokens.get(posicion-
1).getColumna(),bloque,"Funciones");
                    }
                }else{
                    //error si no viene un parentesis lo cual define la función:

Reportes.reporteErrorFuncion(3,elemento.getFila(),elemento.getColumna(),bloque,"Funcio
nes");
                    //bucle para seguir en la siguiente linea
                    CambiarDeLinea(Fila);
                }
            }else{

Reportes.reporteErrorFuncion(2,elemento.getFila(),elemento.getColumna(),bloque,"Funcio
nes");
                //bucle para seguir en la siguiente linea
                CambiarDeLinea(Fila);
            }
            break;
            //para Bloques For
        case "for":
        //For ->   | For Expresión
            //Expresion - >          | ID in (ID|Range ((ID|Constante)(Otros)?)*)OTROS
            posicion++;
            elemento = Tokens.get(posicion);
                if(elemento.getTipoToken().equals("ID")){
                    posicion++;
                    elemento = Tokens.get(posicion);
                    if(elemento.getLexema().equals("in")){
                        posicion++;
                        elemento = Tokens.get(posicion);
                        if(elemento.getLexema().equals("range")){
                            boolean cerroFor = false;
                            boolean errorParametro = false;
                            boolean DoblePuntos = false;
                            Cadena = "";
                            posicion++;
                            //aqui va el funcionamiento si se usa metodo range
```

```
                        while (posicion < Tokens.size()&& Fila == Tokens.get(posicion).getFila()) {
                            elemento = Tokens.get(posicion);
                            if(elemento.getLexema().equals(",")){
                                Cadena += elemento.getLexema();
                                errorParametro = true;
                            }else if(elemento.getTipoToken().equals("ID")||elemento.getTipoToken().equals("Constantes")){
                                Cadena += elemento.getLexema();
                                errorParametro = false;
                            }else if(elemento.getLexema().equals(")")){
                                cerroFor = true;
                            }else if(elemento.getLexema().equals(":")){
                                DoblePuntos = true;
                            }else{

                            }
                            posicion++;
                        }
                        //ciclos que validan q no haya ocurrido ningún error
                        if(!errorParametro){
                            if(cerroFor){
                                if(DoblePuntos){
                                    Reportes.reporteCiclo(Simbolo,"Ciclos", Cadena, elemento.getFila(),elemento.getColumna(),bloque,"For");
                                }else{
                                    //entra aqui si no viene :

                                    Reportes.reporteErrorCiclos(4,elemento.getFila(),elemento.getColumna(),bloque,"For");
                                    CambiarDeLinea(Fila);
                                    }
                            }else{
                                //entra aqui si no se cierra )

                                Reportes.reporteErrorCiclos(6,elemento.getFila(),elemento.getColumna(),bloque,"For");
                                CambiarDeLinea(Fila);
                            }
                        }else{
                            //entra aqui si no se genero correctamente el parametro

                            Reportes.reporteErrorCiclos(5,elemento.getFila(),elemento.getColumna(),bloque,"For");
                        }
                    }else if(elemento.getTipoToken().equals("ID")){
                    //aqui va el funcionamiento si es sobre otro ID
                        posicion++;
                        elemento = Tokens.get(posicion);
                        if(elemento.getLexema().equals(":")){
```

```
                                Reportes.reporteCiclo(Simbolo,"Ciclos",          "Undefined",
elemento.getFila(),elemento.getColumna(),bloque,"For");
                                CambiarDeLinea(Fila);
                            }else{
                                //entra aqui si no se cierra la expresión correctamente

Reportes.reporteErrorCiclos(4,elemento.getFila(),elemento.getColumna(),bloque,"For");
                                //bucle para seguir en la siguiente linea
                                CambiarDeLinea(Fila);
                            }
                        }else{
                            //entra aqui si no recibe el Id o range esperado

Reportes.reporteErrorCiclos(3,elemento.getFila(),elemento.getColumna(),bloque,"For");
                                //bucle para seguir en la siguiente linea
                                CambiarDeLinea(Fila);
                            }
                        }else{
                            //entra aqui si no recibe el in esperado

Reportes.reporteErrorCiclos(2,elemento.getFila(),elemento.getColumna(),bloque,"For");
                                //bucle para seguir en la siguiente linea
                                CambiarDeLinea(Fila);
                            }
                        }else{
                        //error si no recibe un ID en el cual guardar ciclo for

Reportes.reporteErrorCiclos(1,elemento.getFila(),elemento.getColumna(),bloque,"For");
                        //bucle para seguir en la siguiente linea
                        CambiarDeLinea(Fila);
                        }
                    break;
                case "while":
                    //Expresión ->| (ID|Constante) Condicional (ID|Constante) OTROS
                    posicion++;
                    elemento = Tokens.get(posicion);
                    //condicional al que entra si es un ID O Constante

if(elemento.getTipoToken().equals("ID")||elemento.getTipoToken().equals("Constantes")){
                            posicion++;
                            elemento = Tokens.get(posicion);
                            if(elemento.getTipoToken().equals("Comparación")){
                                posicion++;
                                elemento = Tokens.get(posicion);

if(elemento.getTipoToken().equals("ID")||elemento.getPatron().equals("Entero")||elemento.
getPatron().equals("Decimal")){
```

```
                        posicion++;
                        elemento = Tokens.get(posicion);
                        if(elemento.getLexema().equals(":")){
                            Reportes.reporteCiclo(Simbolo,"Ciclos",          "Undefined",
elemento.getFila(),elemento.getColumna(),bloque,"While");
                            posicion++;
                        }else{
                            //error si no recibe un :

Reportes.reporteErrorCiclos(4,elemento.getFila(),elemento.getColumna(),bloque,"While");
                            //bucle para seguir en la siguiente linea
                            CambiarDeLinea(Fila);
                        }
                    }else{
                        //error si no recibe un id o constante

Reportes.reporteErrorCiclos(7,elemento.getFila(),elemento.getColumna(),bloque,"While");
                        //bucle para seguir en la siguiente linea
                        CambiarDeLinea(Fila);
                    }
                }else{
                    //error si no recibe un id o constante

Reportes.reporteErrorCiclos(8,elemento.getFila(),elemento.getColumna(),bloque,"While");
                    //bucle para seguir en la siguiente linea
                    CambiarDeLinea(Fila);
                }
            }else if(elemento.getPatron().equals("booleanas")){
                posicion++;
                elemento = Tokens.get(posicion);
                    if(elemento.getLexema().equals(":")){
                        Reportes.reporteCiclo(Simbolo,"Ciclos",          "Undefined",
elemento.getFila(),elemento.getColumna(),bloque,"While");
                        posicion++;
                    }else{
                        //error si no recibe :

Reportes.reporteErrorCiclos(4,elemento.getFila(),elemento.getColumna(),bloque,"While");
                        //bucle para seguir en la siguiente linea
                        CambiarDeLinea(Fila);
                    }
            }else{
                //error si no recibe un id o constante

Reportes.reporteErrorCiclos(7,elemento.getFila(),elemento.getColumna(),bloque,"While");
                //bucle para seguir en la siguiente linea
                CambiarDeLinea(Fila);
```

```
                    }
                break;
            case "else":
                posicion++;
                elemento = Tokens.get(posicion);
                    if(elemento.getLexema().equals(":")){
                        Reportes.reporteCiclo("else","else",                    "--",
elemento.getFila(),elemento.getColumna(),bloque,"else");
                        posicion++;
                    }else{
                        //error si no recibe :

Reportes.reporteErrorCiclos(4,elemento.getFila(),elemento.getColumna(),bloque,"else");
                        //bucle para seguir en la siguiente linea
                        CambiarDeLinea(Fila);
                    }
                break;
            case "return":
                //Return ->     | Return Expresión
                //Expresión -> |                (Id|Constante)[{((Arimetico)||(Comparación))(
Id|Constante)}*]?
                Cadena = "";
                boolean TieneID = false;
                boolean CerroParentesis = false;
                boolean FinalConstante = false;
                boolean TokenRaro = false;
                posicion++;
                elemento = Tokens.get(posicion);

if(elemento.getTipoToken().equals("ID")||elemento.getTipoToken().equals("Constantes")){
                    if(elemento.getTipoToken().equals("ID")){
                        TieneID = true;
                    }
                    Cadena += " "+elemento.getLexema();
                    posicion++;
                    elemento = Tokens.get(posicion);
                    if(elemento.getTipoToken().equals("Comentario")){
                        posicion++;
                        elemento = Tokens.get(posicion);
                    }
                    FinalConstante = true;
                    //Si el siguiente elemento de la lista sigue en la misma fila después de la
constante se hace analisis
                    if(Fila == Tokens.get(posicion).getFila()){
                        switch (elemento.getTipoToken()) {
                            case "Aritméticos":
                            case "Comparación":
```

```
                        FinalConstante = false;
                        Cadena += " "+elemento.getLexema();
                        posicion++;
                           while (posicion < Tokens.size() && Fila ==
Tokens.get(posicion).getFila()) {
                              elemento = Tokens.get(posicion);

                              if(Fila != elemento.getFila()){
                              }else{
                                 switch (elemento.getTipoToken()) {
                                 case "Aritméticos":
                                 case "Comparación":
                                    FinalConstante = false;
                                    Cadena += " "+elemento.getLexema();
                                    posicion++;
                                    break;
                                 case "ID":
                                 case "Constantes":
                                    FinalConstante = true;
                                    if(elemento.getTipoToken().equals("ID")){
                                       TieneID = true;
                                    }
                                    Cadena += " "+elemento.getLexema();
                                    posicion++;
                                    break;
                                 case "Comentario":
                                    posicion++;
                                    break;
                                 default:
                                    TokenRaro = true;

Reportes.reporteErrorReturn(4,elemento.getFila(),elemento.getColumna(),bloque,"Return")
;
                                    Fila--;
                                    break;
                                 }
                              }
                           }
                        //condiciones para los reportes:
                  //si hubo algún error no hace nada
                  if(TokenRaro){
                  //si no hay error entra al siguiente:
                  }else{
                     //si construyó bien el parametro entra, si no reporte de error
                     if(FinalConstante){
                        if(TieneID){
```

```
                        Reportes.reporteCiclo("return","Palabra reservada","Undefined",
elemento.getFila(),elemento.getColumna(),bloque,"return");
                    }else{
                        Reportes.reporteCiclo("return","Palabra
reservada",Double.toString(Calculadora.evaluarExpresion(Cadena)),
elemento.getFila(),elemento.getColumna(),bloque,"return");
                    }
                }else{

Reportes.reporteErrorReturn(6,elemento.getFila(),elemento.getColumna(),bloque,"Return")
;
                }
            }
                    break;
                case "Otros":
                    Cadena += " "+elemento.getLexema();
                        while (posicion < Tokens.size()-1 && Fila ==
Tokens.get(posicion).getFila()) {
                            elemento = Tokens.get(posicion);
                            switch (elemento.getTipoToken()) {
                                case "Otros":
                                    if(elemento.getLexema().equals(")")){
                                        CerroParentesis = true;
                                        FinalConstante = false;
                                        Fila--;
                                    }else{
                                        FinalConstante = false;
                                    }
                                    Cadena += " "+elemento.getLexema();
                                    break;
                                case "ID":
                                case "Constantes":
                                    FinalConstante = true;
                                    Cadena += " "+elemento.getLexema();
                                    if(elemento.getTipoToken().equals("ID")){
                                        TieneID = true;
                                    }
                                    break;
                                default:
                                    TokenRaro = true;

Reportes.reporteErrorReturn(3,elemento.getFila(),elemento.getColumna(),bloque,"Return")
;
                                    Fila--;
                                    break;
                            }
                            posicion++;
```

```
                        }
                         //condiciones para los reportes:
                //si hubo algún error no hace nada
                if(TokenRaro){
                //si no hay error entra al siguiente:
                }else{
                   //si se cerro el parentesis entra si no crea el reporte de error
                   if(CerroParentesis){
                      //si construyó bien el parametro entra, si no reporte de error
                      if(FinalConstante){
                         if(TieneID){
                            Reporte
```

s.reporteCiclo("return","Palabra reservada","Undefined",
elemento.getFila(),elemento.getColumna(),bloque,"return");

```
                         }else{
                            Reporte
```

s.reporteCiclo("return","Palabra
reservada",Double.toString(Calculadora.evaluarExpresion(Cadena)),
elemento.getFila(),elemento.getColumna(),bloque,"return");

```
                         }
                      }else{
```

Reportes.reporteErrorReturn(6,elemento.getFila(),elemento.getColumna(),bloque,"Return")
;

```
                      }
                   }else{
```

Reportes.reporteErrorReturn(5,elemento.getFila(),elemento.getColumna(),bloque,"Return")
;

```
                   }
                }
                         break;
                      default:
```

Reportes.reporteErrorReturn(2,elemento.getFila(),elemento.getColumna(),bloque,"Return")
;

```
                         break;
                   }
                }else{
                   //si el siguiente elemento no está en la misma fila se crea el reporte
                   Reporte
```

s.reporteCiclo("return","Palabra reservada",Cadena,
elemento.getFila(),elemento.getColumna(),bloque,"return");

```
                }
             }else{
                //error si se recibe un token que no se esperaba
```

Reportes.reporteErrorReturn(1,elemento.getFila(),elemento.getColumna(),bloque,"Return")
;

```
             }
```

```
                    break;
                case "break":
                    CambiarDeLinea(Fila);
                    Reportes.reporteCiclo("break","Palabra reservada", "break",
elemento.getFila(),elemento.getColumna(),bloque,"return");
                    break;
                default:
                    // Código por defecto si la opción no coincide con ningún caso
                    posicion++;
                    System.out.println("Opción no válida");
                    break;
            }
        }else if(elemento.getTipoToken().equals("Comentario")){
            posicion++;
        }else{

Reportes.reporteErrorGeneral(4,elemento.getFila(),elemento.getColumna(),bloque,"Genera
l");
            CambiarDeLinea(Fila);
}
```

ResetBloques():

Método utilizado para reiniciar todos los valores y poder realizar un nuevo análisis del texto ingresado.

```
bloque = 1;
    Reportes = new ReportesSintacticos();
    Reportes.ResetArreglos();
    calc = new Calculadora();
    bloque = 1;
    posicion = 0;
    Tokens = recopiladorLexico();
```

Reportes sintácticos

A diferencia de la parte léxica se destino una clase distinta para poder guardar el trabajo hecho por el analizador Léxico, para eso esta la clase Reportes sintácticos, los métodos como tal sirven para guardar los errores y valores creados por el análisis sintáctico.

reporteErrorAsignacion(int caso, int fila, int columna, int bloque, String nivel):

En general se usa un mismo formato para las distintas utilidades que se requerían, este método sirve para la creación de reportes de error en las asignaciones.

```
TablaSintactica token;
    switch(caso){
        //asignación de enteros
```

```
            case 1:
                token = new TablaSintactica("","declaracion","","No se puede operar una cadena con otro valor",fila,columna,bloque,nivel);
                errorRecopilado.add(token);
                break;
            case 2:
                 token = new TablaSintactica("","declaracion","","Se recibió un token invalido al construir la operación",fila,columna,bloque,nivel);
                errorRecopilado.add(token);
                break;
            case 3:
                token = new TablaSintactica("","declaracion","","No se cerro la cadena correctamente",fila,columna,bloque,nivel);
                errorRecopilado.add(token);
                break;
            //asignación de otros
            case 4:
                token = new TablaSintactica("","declaracion","","No se cerro el arreglo",fila,columna,bloque,nivel);
                errorRecopilado.add(token);
                break;
                //Error en el formato más general
            case 5:
                token = new TablaSintactica("","declaracion","","hubo un error en el formato:",fila,columna,bloque,nivel);
                errorRecopilado.add(token);
                break;
            case 6:
                token = new TablaSintactica("","declaracion","","Se esperaba un signo =",fila,columna,bloque,nivel);
                errorRecopilado.add(token);
                break;
            case 7:
                token = new TablaSintactica("","declaracion","","Ocurrió un error con el formato al intentar la declaracion",fila,columna,bloque,nivel);
                errorRecopilado.add(token);
                break;
            default:
                System.out.println("wtf xd");
                break;
        }
```

public void reporteAsignacion(String Simbolo, String tipo, String valor, int fila, int columna,int bloque, String nivel):

Este método se encargaba de la creación de un objeto Tabla Sintactica y guardarla en el arrayList correspondiente.

```
        TablaSintactica token = new
TablaSintactica(Simbolo,tipo,valor,"",fila,columna,bloque,nivel);
        reporteRecopilado.add(token);
    }
```

Tabla sintáctica

Esta clase se encargaba de guardar los objetos creados por el analizador sintáctico, era una clase objeto como tal y sigue la siguiente estructura:

```
public TablaSintactica(String Simbolo, String Tipo ,String Valor, String Descripcion, int fila,
int columna, int bloque, String nivel) {
        this.Simbolo = Simbolo;
        this.Tipo = Tipo;
        this.Valor = Valor;
        this.Descripcion = Descripcion;
        this.fila = fila;
        this.columna = columna;
        this.bloque = bloque;
        this.nivel = nivel;
    }
```

Menú Inicial

Las clase de menú inicial es una clase Jframe la cual se encarga de la interfaz gráfica del programa, esta fue creada utilizando el editor de netbeans por lo cual no se indagará en detalle sobre los métodos de esta