

Proyecto integrador (FINAL)

Lenguajes Y Autómatas II

28/05/2024

Participantes

Rogelio Perez Guevara

Idwin Raziel Balderas Almanza

Karina Monserrat Jimenez Camacho

Equipo 2

Docente

Ing. Karina Cabrera Chagoyán

índice

Introducción.....	7
Practica No. 1 Análisis semántico	8
Objetivo.....	8
Descripción.....	8
TABLA DE SIMBOLOS	9
TABLA DE DIRECCIONES	10
EJEMPLO TABLA DE TOKENS antes de la ejecución.....	11
EJEMPLO Tabla de Símbolos	11
EJEMPLO Tabla de direcciones	11
EJEMPLO TABLA DE TOKENS después de la ejecución	12
Desarrollo.....	13
Versión 1	14
Método Main.....	14
Método estaBien	16
Método tk2	17
Método esPalabraReservada	18
Método esNúmero	19
Método eliminarLineasRepetidas	20
Método eliminarLineasRepetidas2	21
Método eliminarLineas	22
Tabla de Símbolos	23
Tabla de Direcciones	23
Tabla de Tokens	24
Diagrama de clase.....	25
Ejecución.....	26
Resultado.....	27
Versión 2	28
Metodo Main.....	28
Método estaBien	30
Método tk2	31
Método eliminarLineasRepetidas	32
Tabla de Símbolos	33

Tabla de Direcciones	33
Tabla de Tokens	34
Diagrama de clase.....	35
Ejecución.....	36
Resultado.....	37
Cambios.....	39
Conclusión.....	40
Practica No. 2 Simulación del vector de código intermedio	41
Objetivo.....	41
Descripción.....	41
EJEMPLO	42
Desarrollo.....	43
Versión 1	44
Método Main.....	44
Método verificarExistencia.....	51
Método operacionDePilas	53
Método guardarArrayListsEnArchivo.....	54
Método leerArchivoYMostrarContenido.....	55
Método leerArchivoYMostrarContenido2.....	56
Método leerArchivoYMostrarContenido3.....	57
Método leerArchivoYMostrarContenido4.....	58
Método imprimirRenglónConNumeros	59
Método obtenerPrimerElementoSplit	60
Método formatoArchivo	61
Tabla de token utilizada	62
Diagrama de clase.....	63
Ejecución.....	64
Opción 1	64
Opción 2	64
Opción 3	64
Opción 4	65
Resultado.....	66
Versión 2	67

Método Main.....	67
Método verificarExistencia.....	74
Método operacionDePilas	76
Método guardarArrayListsEnArchivo.....	77
Método leerArchivoYMostrarContenido.....	78
Método leerArchivoYMostrarContenido2.....	79
Método leerArchivoYMostrarContenido3.....	80
Método leerArchivoYMostrarContenido4.....	81
Método imprimirRenglonConNumeros	82
Método obtenerPrimerElementoSplit	83
Método formatoArchivo	84
Tabla de token utilizada	85
Diagrama de clase.....	86
Ejecución.....	87
Opción 1	87
Opción 2	87
Opción 3	87
Opción 4	88
Resultado.....	89
Cambios.....	90
Conclusión.....	91
Practica No. 3 Optimización.....	92
Optimización de ciclos	92
Expansión de bucles (loop unrolling).....	93
Reducción de frecuencia	94
Código sin optimizar	94
Código optimizado.....	95
Explicación de la optimización.....	96
Tiempos de ejecución del código sin optimizar y optimizado.....	96
Diagrama de clase.....	97
Optimización Tipo Mirilla.....	98
Código Base.....	99
Diagrama de clase.....	100

Ejecución	101
Código Mirilla	101
Diagrama de clase	104
Ejecución	105
Código Sin Mirilla	106
Diagrama de clase	107
Ejecución	108
Comparación	109
Optimización Locales	110
Optimizaciones locales	110
Bloque básico	111
Ensamblamineto (Folding)	112
Implementación del Folding	113
Ejecución en tiempo de compilación	114
Reutilización de expresiones comunes	115
Propagación de copias	116
Eliminación redundancias en acceso matrices	117
Transformaciones algebraicas	117
Código Base optimización local	118
Ejecución	119
Código con optimización local	120
Ejecución	121
Cambios	122
Conclusión	123
Practica No. 4 Simulación de la pila de ejecución del VCI	124
Objetivo	124
Descripción	124
Desarrollo	124
Método Main	125
Método sobreescribirArchivoSimbolos	149
Método lecturaDeSimbolos	150
Tabla de Símbolos	151
Tabla de Direcciones	151

Vector de Código Intermedio	152
Diagrama de clase.....	153
Ejecución.....	154
Resultado.....	155
Cambios.....	156
Conclusión.....	157
Conclusión del Proyecto integrador	158
Bibliografía	159
Anexos	160

Introducción

En el ámbito del análisis semántico y la generación de código intermedio, el dominio de técnicas y herramientas adecuadas es fundamental para el desarrollo de compiladores y entornos de programación. Las prácticas que se detallan a continuación tienen como objetivo fundamental que los alumnos adquieran competencias clave en la generación de tablas de símbolos y direcciones, así como en la simulación de vectores de código intermedio y la pila de ejecución, utilizando el lenguaje de programación Java.

Práctica No. 1: Análisis Semántico

El objetivo de esta primera práctica es que los alumnos desarrollen una aplicación en Java para generar tablas de símbolos y direcciones a partir de una tabla de tokens predefinida. La tarea implica leer la tabla de tokens, modificar los datos de posición en tabla y generar dos archivos externos correspondientes a la tabla de símbolos y la tabla de direcciones. La correcta implementación de esta práctica proporciona una base sólida en la manipulación y análisis de tokens, esenciales para el análisis semántico en compiladores.

Práctica No. 2: Simulación del Vector de Código Intermedio

En la segunda práctica, los alumnos deben aplicar conceptos de vectores de código intermedio (VCI) para simular su generación en expresiones aritméticas, lógicas, relacionales y estructuras de control. Utilizando Java, la práctica se enfoca en el manejo de prioridades de operadores y la utilización de pilas para operadores, estatutos y direcciones. Esta práctica refuerza la comprensión del flujo de ejecución del código y la estructura de VCI, elementos cruciales en la etapa de generación de código de un compilador.

Práctica No. 4: Simulación de la Pila de Ejecución del VCI

Finalmente, la cuarta práctica se centra en la simulación de la pila de ejecución del VCI, utilizando los archivos generados en las prácticas anteriores. Los alumnos deben desarrollar una aplicación en Java que lea el archivo de VCI y ejecute las instrucciones correspondientes, actualizando la tabla de símbolos y mostrando la salida de datos en pantalla. Este ejercicio pone en práctica los conocimientos sobre ejecución de código intermedio y la manipulación de pilas, esenciales para la comprensión de la ejecución de programas en tiempo de compilación.

Estas prácticas no solo buscan fortalecer las habilidades técnicas en programación y análisis semántico, sino también promover el trabajo colaborativo, la optimización del código y el manejo eficiente de errores. A través de estas actividades, los alumnos desarrollan una comprensión profunda de los procesos involucrados en la construcción de compiladores, adquiriendo competencias que son fundamentales para su formación como ingenieros en sistemas.

Practica No. 1 Análisis semántico

Objetivo

El alumno generara la tabla de símbolos y de direcciones de un lenguaje de prueba utilizando el lenguaje de programación JAVA o herramienta.

Descripción

Realice una aplicación utilizando su lenguaje de programación Java o herramienta y genere dos archivos externos – Tabla de símbolos y tabla de direcciones para un lenguaje de prueba.

Suponga que ya existe la tabla de TOKENS con la siguiente estructura:

Lexema	Token	Posición Tabla	Línea
--------	-------	----------------	-------

Donde:

Lexema: El valor asociado al token en el programa fuente.

Token: El numero de token asignado.

Posición Tabla: Posición en la tabla de símbolos, donde -1 indica que no va en tabla de símbolos y -2 si se agrega a tabla de símbolos o de direcciones.

Línea : El numero de línea en el programa fuente.

El programa o aplicación deberá leer la tabla de tokens al inicio y mostrar al final la tabla con los datos de “**posición en tabla**” **modificados**, así como generar la tabla de símbolos y tabla de direcciones correspondiente, archivo que se podrá leer desde consola al finalizar la ejecución de la aplicación con la siguiente estructura:

TABLA DE SIMBOLOS

ID	Token	Valor	Ámbito
----	-------	-------	--------

Donde:

ID: Identificador en el programa fuente.

Token: NUMERO DE TOKEN de acuerdo a LEXICO.

Valor: Asignar de acuerdo al tipo de dato, si es entero 0, si es real 0.0 si es constante string null y si es lógico true.

Ámbito: Main si pertenece al programa principal, o nombre del procedimiento o función si pertenece a algún método.

TABLA DE DIRECCIONES

ID	Token	No. Línea	VCI
----	-------	-----------	-----

Donde:

ID: Identificador en el programa fuente.

Token: NUMERO DE TOKEN de acuerdo a LEXICO.

No. Línea: Número de línea donde se encuentra el token.

VCI: Campo que se modificará en cuando se genere VCI, **por default es 0.**

Consideré lo siguiente para la identificación de TOKENS - **LEXICO** de lenguaje de prueba.

EJEMPLO TABLA DE TOKENS antes de la ejecución

	TOKENS		TOKENS
1	programa,-1,-1,1	26	b&, -51, -2 , 9
2	uno@,-55, -2 ,1	27	=, -26 , -1 , 9
3	; , -75 , -1 , 1	28	0, -61, -1, 9
4	variables, -15, -1, 2	29	; , -75, -1, 9
5	entero, -11, -1, 3	30	z\$, -53, -2 , 10
6	a&, -51, -2 , 3	31	= , -26 , -1 , 10
7	, , -76 , -1 , 3	32	"Hola", -63, -1, 10
8	b& , -51 , -2 , 3	33	; , -75, -1, 10
9	; , -75 , -1 , 3	34	x% , -52,-2 , 11
10	real, -12, -1 , 4	35	= , -26 , -1 , 11
11	x%, -52, -2 , 4	36	45.5, -62, -1, 11
12	, , -76 , -1 , 4	37	; , -75, -1, 11
13	y%, -52, -2 , 4	38	y%, -52, -2 , 12
14	; , -75, -1, 4	39	= , -26 , -1 , 12
15	cadena, -13, -1 , 5	40	6.6, -62, -1, 12
16	z\$, -53 , -2 , 5	41	; , -75, -1, 12
17	; , -75 , -1 , 5	42	dos# , -54, -2 , 13
18	logico, -14, -1 , 6	43	=, -26 , -1 , 13
19	dos#, -54 , -2 , 6	44	true, -64, -1, 13
20	; , -75 , -1 , 6	45	; , -75, -1, 13
21	inicio , -2, -1,7	46	Fin, -3,-1,14
22	a& , -51, -2 , 8	47	
23	=, -26 , -1 , 8	48	
24	300, -61, -1, 8	49	
25	; , -75, -1, 8	50	

EJEMPLO Tabla de Símbolos

ID	TOKEN	VALOR	AMBITO
a&	-51	0	Main
b&	-51	0	Main
x%	-52	0.0	Main
y%	-52	0.0	Main
z\$	-53	Null	Main
dos#	-54	True	Main

EJEMPLO Tabla de direcciones

ID	TOKEN	No. Línea	VCI
uno@	-55	1	0

EJEMPLO TABLA DE TOKENS después de la ejecución

	TOKENS		TOKENS
1	programa,-1,-1,1	26	b&, -51, -1, 9
2	uno@,-55, 0 ,1	27	=, -26 , -1 , 9
3	; , -75 , -1 , 1	28	0, -61, -1, 9
4	variables, -15, -1, 2	29	; , -75, -1, 9
5	entero, -11, -1, 3	30	z\$, -53, 4, 10
6	a&, -51, 0 , 3	31	= , -26 , -1 , 10
7	, , -76 , -1 , 3	32	"Hola", -63, -1, 10
8	b& , -51 , 1 , 3	33	; , -75, -1, 10
9	; , -75 , -1 , 3	34	x% , -52, 2 , 11
10	real, -12, -1 , 4	35	= , -26 , -1 , 11
11	x%, -52, 2 , 4	36	45.5, -62, -1, 11
12	, , -76 , -1 , 4	37	; , -75, -1, 11
13	y%, -52, 3 , 4	38	y%, -52, 3 , 12
14	; , -75, -1, 4	39	= , -26 , -1 , 12
15	cadena, -13, -1 , 5	40	6.6, -62, -1, 12
16	z\$, -53 , 4, 5	41	; , -75, -1, 12
17	; , -75 , -1 , 5	42	dos# , -54, 5 , 13
18	logico, -14, -1 , 6	43	=, -26 , -1 , 13
19	dos#, -54 , 5 , 6	44	true, -64, -1, 13
20	; , -75 , -1 , 6	45	; , -75, -1, 13
21	inicio , -2, -1,7	46	Fin, -3,-1,14
22	a& , -51, 0 , 8	47	
23	=, -26 , -1 , 8	48	
24	300, -61, -1, 8	49	
25	; , -75, -1, 8	50	

Considere que los identificadores al ser declarados deben agregarse a la tabla de símbolos y modificar campo de “posición en tabla” en tabla de tokens de acuerdo a su posición en tabla, si se trata de un identificador general deberá agregarse a tabla de direcciones, considerando si se trata del nombre del programa o del nombre de algún método, procedimiento o función. Debe indicar error cuando existen dos identificadores con el mismo nombre, o cuando son declarados de un tipo diferente a lo indicado con el carácter de control o cuando no se ha declarado un identificador o cuando se esté haciendo un uso no adecuado correspondiente al tipo de dato.

Desarrollo

Para alcanzar el objetivo de esta práctica, distribuimos el trabajo de manera equitativa:

- Karina y Rogelio se encargaron de desarrollar parte del código para obtener los nuevos valores de la tabla de tokens.
- Idwin finalizó el código.
- Rogelio también se encargó de desarrollar la segunda versión del código y de optimizar ambas versiones.

Finalmente, realizamos pruebas utilizando el ejemplo proporcionado en la práctica para asegurar el correcto funcionamiento del código.

Versión 1

Método Main

```
1 package Analisis_Semantico;
2
3 import java.io.BufferedReader;
4 import java.io.BufferedWriter;
5 import java.io.FileReader;
6 import java.io.FileWriter;
7 import java.io.IOException;
8 import org.apache.commons.io.FileUtils;
9 import java.util.ArrayList;
10 import java.util.LinkedHashSet;
11 import java.util.Set;
12 import java.util.HashSet;
13 import java.io.*;
14
15 /**
16 * Practica 1 Analisis Semantico
17 * @author Equipo 2
18 * @version 26/02/2024
19 */
20 public class Analisis_Semantico
21 {
22     public static void main (String [] args) throws IOException
23     {
24         // declara los archivos de entrada y de salida
25         String archivoEntrada = "C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Aut\u00f3matas II\\\\Proyecto\\\\src\\\\Recursos\\\\Tabla de Tokens.txt";
26         String archivoSalida = "C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Aut\u00f3matas II\\\\Proyecto\\\\src\\\\Recursos\\\\Tabla de S\u00edmbolos.txt";
27         String archivoDirecciones = "C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Aut\u00f3matas II\\\\Proyecto\\\\src\\\\Recursos\\\\Tabla de direcciones.txt";
28         String archivoT2 = "C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Aut\u00f3matas II\\\\Proyecto\\\\src\\\\Recursos\\\\Tabla de Tokens2.txt";
29         String valor = "null";
30
31         // crea la lectura y escritura de los archivos
32         try (BufferedReader br = new BufferedReader(new FileReader(string:archivoEntrada));
33              BufferedWriter bw = new BufferedWriter(new FileWriter(string:archivoSalida));
34              BufferedWriter dw = new BufferedWriter(new FileWriter(string:archivoDirecciones)))
35         {
36             // linea almacena la l\u00ednea leida del archivo
37             String linea;
```

```
38
39             // linea almacena la l\u00ednea leida del archivo
40             String linea;
41
42             // en este ciclo lo que se hace es separar en 4 partes la cadena de entrada
43             while ((linea = br.readLine()) != null)
44             {
45                 String[] partes = linea.split("\\.", limit:4);
46                 String primeraParte = partes[0].trim();
47                 String segundaParte = partes[1].trim();
48                 String cuartaParte = partes[3].trim();
49
50                 // aqui de la primera parte de la cadena lo que se hace es asignar el tipo de valor que es
51                 if(primeraparte.equals(anObject: "entero"))
52                 {
53                     valor = "0";
54                 }
55                 else if(primeraparte.equals(anObject: "real"))
56                 {
57                     valor = "0.0";
58                 }
59                 else if(primeraparte.equals(anObject: "string"))
60                 {
61                     valor = "Null";
62                 }
63                 else if(primeraparte.equals(anObject: "logico"))
64                 {
65                     valor = "True";
66                 }
67
68                 if(primeraparte.contains(".")){
69                     dw.write(primeraparte + ",");
70                     dw.write(segundaParte + ",");
71                     dw.write(cuartaParte + ",");
72                     dw.write(ss: "0");
73                 }
74
75                 // aqui se escribe en el archivo de salida al hacer la comparativa de si no es una palabra reservada
```

The screenshot shows a Java IDE interface with the following details:

- Title Bar:** Shows "Start Page" and "Analisis_Semantico.java".
- Toolbar:** Includes standard icons for file operations like Open, Save, Print, and Run.
- Left Sidebar:** Displays project navigation with sections for Navigator, Projects, Files, and Services.
- Code Editor:** Contains the following Java code:

```
72     // aqui se escribe en el archivo de salida al hacer la comparativa de si no es una palabra reservada
73     // y no es numero
74     if (!esPalabraReservada(palabraPrimeraParte) && !esNumero(cadenaPrimeraParte))
75     {
76         bw.write(primerasParte + ",");
77         bw.write(segundaParte + ",");
78         bw.write(valor + ",");
79         bw.write(ssc:"Main\n");
80     }
81
82     System.out.println("Procesamiento completado. Se ha generado el archivo: " + archivoSalida);
83 }
84 catch (IOException e)
85 {
86     e.printStackTrace();
87 }
88
89 //crearDirecciones(archivoEntrada,archivoDirecciones);
90 try
91 {
92     eliminarLineasRepetidas(archivoEntrada,archivoSalida);
93     System.out.println("Lineas repetidas eliminadas correctamente.");
94 }
95 catch (IOException e)
96 {
97     System.err.println("Error al procesar el archivo: " + e.getMessage());
98     e.printStackTrace();
99 }
100
101 try
102 {
103     eliminarLineasRepetidas2(inputFile:archivoSalida);
104     System.out.println("Se eliminaron las lineas repetidas correctamente.");
105 }
106 catch (IOException e)
107 {
108 }
```

The code handles file processing, specifically dealing with reserved words and numbers, and then moves on to creating address files and removing repeated lines from input files.

The screenshot shows a Java IDE interface with the following details:

- Title Bar:** Shows "Start Page" and "Analisis_Semantico.java".
- Toolbar:** Includes standard icons for file operations like Open, Save, Print, and Run.
- Left Sidebar:** Displays project navigation with sections for Navigator, Projects, Files, and Services.
- Code Editor:** Contains the following Java code:

```
107     catch (IOException e)
108     {
109         System.err.println("Error al procesar el archivo: " + e.getMessage());
110     }
111
112     try
113     {
114         eliminarLineas(fileName:archivoSalida);
115         System.out.println("Lineas eliminadas correctamente.");
116     }
117     catch (IOException e)
118     {
119         System.err.println("Error al procesar el archivo: " + e.getMessage());
120     }
121
122     tk2(lectr0:archivoSalida,lectr1:archivoEntrada,escribirT:archivoT2);
123     estaBien(lectr1:archivoEntrada);
124 }
125 }
```

This part of the code focuses on removing lines from a file and then performing a comparison between two files using a function named tk2.

Método `estaBien`

```
Start Page x Analysis_Semantico.java x
Source History Navigator Projects Files Services
private static void estaBien(String linea)
{
    try
    {
        BufferedReader leerToken = new BufferedReader(new FileReader(string.lineas));
        String linea;
        ArrayList<String> enteros = new ArrayList<>();
        ArrayList<String> reales = new ArrayList<>();
        ArrayList<String> string = new ArrayList<>();
        ArrayList<String> logicos = new ArrayList<>();
        ArrayList<String> general = new ArrayList<>();

        while ((linea = leerToken.readLine()) != null)
        {
            String[] partes = linea.split(regex, limit: 4);
            switch (partes[1].trim())
            {
                case "-S1":
                    enteros.add(partes[0].trim().replace(target: "#", replacement: ""));
                    break;
                case "-S2":
                    reales.add(partes[0].trim().replace(target: "%", replacement: ""));
                    break;
                case "-S3":
                    string.add(partes[0].trim().replace(target: "S", replacement: ""));
                    break;
                case "-S4":
                    logicos.add(partes[0].trim().replace(target: "#", replacement: ""));
                    break;
                case "-S5":
                    general.add(partes[0].trim().replace(target: "@", replacement: ""));
                    break;
            }
        }

        for (String elemento : enteros)
        {
            if(reales.contains(elemento) || string.contains(elemento) || logicos.contains(elemento) || general.contains(elemento))
                throw new IOException(string: "la variable esta declarada en 2 tipos de datos");
        }

        for (String elemento : reales)
        {
            if(enteros.contains(elemento) || string.contains(elemento) || logicos.contains(elemento)||general.contains(elemento))
                throw new IOException(string: "la variable esta declarada en 2 tipos de datos");
        }

        for (String elemento : string)
        {
            if(reales.contains(elemento) || enteros.contains(elemento) || logicos.contains(elemento)|| general.contains(elemento))
                throw new IOException(string: "la variable esta declarada en 2 tipos de datos");
        }

        for (String elemento : logicos)
        {
            if(reales.contains(elemento) || string.contains(elemento) || enteros.contains(elemento)|| general.contains(elemento))
                throw new IOException(string: "la variable esta declarada en 2 tipos de datos");
        }

        for (String elemento : general)
        {
            if(reales.contains(elemento) || string.contains(elemento) || enteros.contains(elemento)|| logicos.contains(elemento))
                throw new IOException(string: "la variable esta declarada en 2 tipos de datos");
        }
    }
}
```

```
for (String elemento : enteros)
{
    if(reales.contains(elemento) || string.contains(elemento) || logicos.contains(elemento) || general.contains(elemento))
        throw new IOException(string: "la variable esta declarada en 2 tipos de datos");
}

for (String elemento : reales)
{
    if(enteros.contains(elemento) || string.contains(elemento) || logicos.contains(elemento)||general.contains(elemento))
        throw new IOException(string: "la variable esta declarada en 2 tipos de datos");
}

for (String elemento : string)
{
    if(reales.contains(elemento) || enteros.contains(elemento) || logicos.contains(elemento)|| general.contains(elemento))
        throw new IOException(string: "la variable esta declarada en 2 tipos de datos");
}

for (String elemento : logicos)
{
    if(reales.contains(elemento) || string.contains(elemento) || enteros.contains(elemento)|| general.contains(elemento))
        throw new IOException(string: "la variable esta declarada en 2 tipos de datos");
}

for (String elemento : general)
{
    if(reales.contains(elemento) || string.contains(elemento) || enteros.contains(elemento)|| logicos.contains(elemento))
        throw new IOException(string: "la variable esta declarada en 2 tipos de datos");
}
```

```
for (String elemento : general)
{
    if(reales.contains(elemento) || string.contains(elemento) || enteros.contains(elemento)|| logicos.contains(elemento))
        throw new IOException(string: "la variable esta declarada en 2 tipos de datos");
}

catch (IOException e)
{
    System.out.println(e.getMessage());
}
```

Método tk2

The screenshot shows a Java code editor with the file `Analisis_Semantico.java` open. The code implements the `tk2` method, which reads tokens from a file and writes them to another file. The code uses `BufferedReader` and `BufferedWriter` to handle file I/O. It splits each line into four parts based on a comma separator and writes them to the output file separated by commas.

```
private static void tk2(String leerT, String escribirT)
{
    try (BufferedReader leerToken = new BufferedReader(new FileReader(string:leerT));
         BufferedWriter t2 = new BufferedWriter(new FileWriter(string:escribirT)))
    {
        File original = new File(string:leerT);
        String linea;
        String linea2;
        int cont = 0;

        while ((linea = leerToken.readLine()) != null)
        {
            String[] partes = linea.split(regex: ",", limit: 4);
            String primeraParte = partes[0].trim();
            String segundaParte = partes[1].trim();
            String terceraParte = partes[2].trim();
            String cuartaParte = partes[3].trim();
            BufferedReader leerSimbolos = new BufferedReader(new FileReader(string:leerT));

            while ((linea2 = leerSimbolos.readLine()) != null)
            {
                String[] partesSimbolos = linea2.split(regex: ",", limit: 4);
                String primeraParteSimbolo = partesSimbolos[0].trim();

                if(primeraparte.equals(anObject: primeraParteSimbolo))
                {
                    ... terceraParte = "" + cont;
                }
                cont++;
            }

            leerSimbolos.close();
            t2.write(primeraparte + "," + segundaParte + "," + terceraParte + "," + cuartaParte + "\n");
            cont=0;
        }

        leerToken.close();
    }
}
```

The screenshot shows the continuation of the `tk2` method implementation. It handles closing resources and renaming files. It creates a new file named `Tabla de Tokens.txt` and moves the original file to this new name.

```
leerToken.close();
t2.close();
original.delete();
File archivoOriginal = new File(string:escribirT);
File archivoRenombrado = new File(string:archivoOriginal.getParent(), string:"Tabla de Tokens.txt");
FileUtils.moveToFile(srcfile:archivoOriginal, destfile:archivoRenombrado);

}
catch (IOException e)
{
    System.out.println("no se pudo renombrar");
}
```

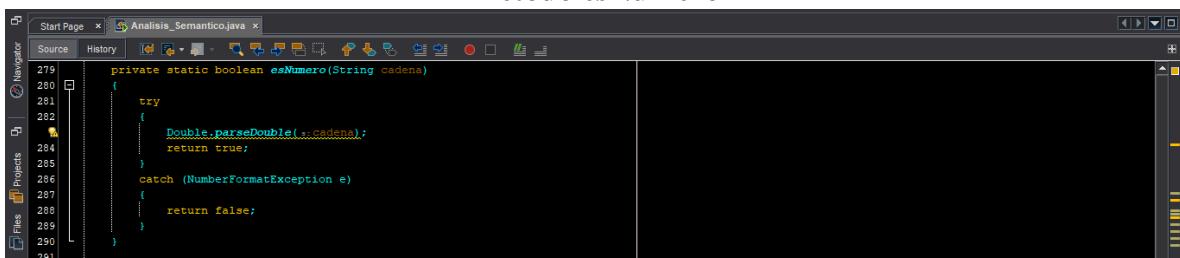
Método esPalabraReservada

The screenshot shows a Java code editor with the file 'Analisis_Semantico.java' open. The code defines a static boolean method 'esPalabraReservada' that checks if a given string is a reserved word. It first checks if the string contains '@'. If it does, it returns true. Otherwise, it compares the string against a list of reserved words stored in a static array. The array contains numerous reserved words in Spanish, such as 'programa', 'inicio', 'Fin', 'leer', 'escribir', 'si', 'sino', 'mientras', 'repetir', 'hasta', 'entero', 'real', 'string', 'cadena', 'Logico', 'logico', 'Variables', 'variables', 'Entonces', 'entonces', 'Hacer', 'hacer', '+', '/', '*', '+', '=', '>=', '>', '>=', '==', '!=', '||', '||', '(', ')', ';', ',', 'true', and 'false'. The code uses standard Java syntax with curly braces for blocks and parentheses for loops and conditionals.

```
private static boolean esPalabraReservada(String palabra)
{
    if (palabra.contains("@"))
    {
        return true;
    }

    String[] palabrasReservadas = {"programa", "inicio", "Fin", "leer", "escribir", "si", "sino",
                                   "mientras", "repetir", "hasta", "entero", "real", "string", "cadena",
                                   "Logico", "logico", "Variables", "variables", "Entonces", "entonces",
                                   "Hacer", "hacer", "+", "/", "*", "+", "=", ">=", ">", ">=", "==", "!=",
                                   "||", "||", "(", ")", ";", ",", "true", "false"};
    for (String palabraReservada : palabrasReservadas)
    {
        if (palabra.equals(anObject: palabraReservada))
        {
            return true;
        }
    }
    return false;
}
```

Método esNúmero



The screenshot shows a Java code editor window with the file "Analisis_Semantico.java" open. The code defines a static boolean method named "esNúmero" that takes a string parameter "cadena". The method attempts to parse the string into a double using `Double.parseDouble(cadena)`. If successful, it returns true; if a `NumberFormatException` is caught, it returns false. The code is annotated with line numbers from 279 to 291.

```
private static boolean esNúmero(String cadena)
{
    try
    {
        Double.parseDouble(cadena);
        return true;
    }
    catch (NumberFormatException e)
    {
        return false;
    }
}
```

Método eliminarLineasRepetidas

The screenshot shows a Java code editor with the file `Analisis_Semantico.java` open. The code implements a static method `eliminarLineasRepetidas` that reads lines from an input file and writes unique lines to an output file, ignoring duplicates.

```
292 public static void eliminarLineasRepetidas(String archivoEntrada, String archivoSalida) throws IOException
293 {
294     Set<String> lineasUnicas = new LinkedHashSet<>();
295
296     try (BufferedReader br = new BufferedReader(new FileReader(string:archivoEntrada)))
297     {
298         String linea;
299
300         while ((linea = br.readLine()) != null)
301         {
302             lineasUnicas.add(string:linea);
303         }
304     }
305
306     try (BufferedWriter bw = new BufferedWriter(new FileWriter(string:archivoSalida)))
307     {
308         for (String linea : lineasUnicas)
309         {
310             bw.write(string:linea);
311             bw.newLine();
312         }
313     }
314 }
```

Método eliminarLineasRepetidas2

The screenshot shows a Java code editor with the file `Analisis_Semantico.java` open. The code implements a method to remove repeated lines from an input file. It uses a BufferedReader to read lines, a HashSet to store unique lines, and a StringBuilder to build the result. The code includes comments explaining the logic, such as splitting lines by commas and handling the first part of each line.

```
31.6 public static void eliminarLineasRepetidas2(String inputFile) throws IOException
31.7 {
31.8     BufferedReader reader = new BufferedReader(new FileReader(string:inputFile));
31.9     Set<String> firstParts = new HashSet<>();
320     StringBuilder result = new StringBuilder();
321     String line;
322
323     while ((line = reader.readLine()) != null)
324     {
325         String[] parts = line.split(regex:",", limit:4); // Divide la linea en dos partes utilizando ","
326         String firstPart = parts[0]; // La primera parte antes de la ","
327
328         if (!firstParts.contains(:firstPart))
329         {
330             // Si esta es la primera vez que vemos esta primera parte, la agregamos al resultado
331             result.append(:line).append(:"\n");
332             firstParts.add(:firstPart);
333         }
334     }
335
336     // Cerrar el archivo
337     reader.close();
338
339     // Escribir el resultado de vuelta al archivo de entrada
340     BufferedWriter writer = new BufferedWriter(new FileWriter(string:inputFile));
341     writer.write(:result.toString());
342     writer.close();
343 }
344 }
```

Método eliminarLineas

The screenshot shows a Java code editor window with the file 'Analisis_Semantico.java' open. The code implements a static method 'eliminarLineas' that reads from an input file and writes to a temporary file, ignoring lines starting with a '#' or '''. The original file is then deleted, and the temporary file is renamed back if successful.

```
345 public static void eliminarLineas(String fileName) throws IOException
346 {
347     File inputFile = new File(string:fileName);
348     File tempFile = new File(string:"tempFile.txt");
349     BufferedReader reader = new BufferedReader(new FileReader(file:inputFile));
350     BufferedWriter writer = new BufferedWriter(new FileWriter(file:tempFile));
351     String line;
352
353     while ((line = reader.readLine()) != null)
354     {
355         // Si la linea no comienza con "#" o """
356         if (!line.startsWith(prefix: "#") && !line.startsWith(prefix: "\""))
357         {
358             writer.write(line + System.getProperty(key: "line.separator")); // Mantener la linea
359         }
360     }
361
362     writer.close();
363     reader.close();
364
365     // Sobreescribir el archivo original con el archivo temporal
366     if (!inputFile.delete())
367     {
368         throw new IOException(string: "No se pudo eliminar el archivo original.");
369     }
370     if (!tempFile.renameTo(dest: inputFile))
371     {
372         throw new IOException(string: "No se pudo renombrar el archivo temporal.");
373     }
374 }
375 }
```

Tabla de Símbolos

1	a&,-51,0,Main
2	b&,-51,0,Main
3	x%, -52,0.0,Main
4	y%, -52,0.0,Main
5	z\$, -53,Null,Main
6	dos#, -54,True,Main
7	

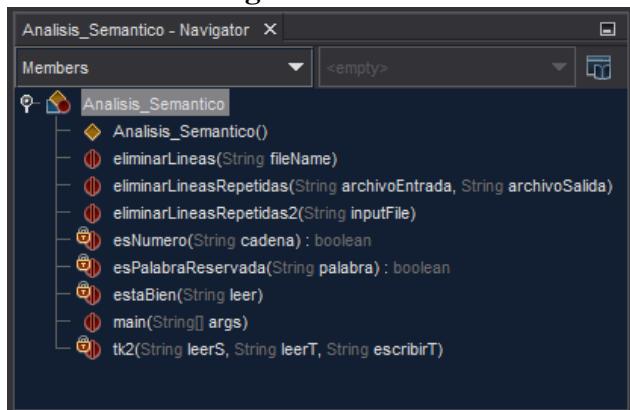
Tabla de Direcciones

1	JNO@,-55,1,0

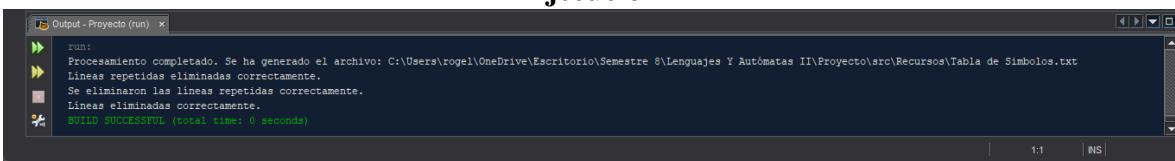
Tabla de Tokens

1	programa,-1,-1,1
2	UNO@,-55,-2,1
3	;-,-75,-1,1
4	variables,-15,-1,2
5	entero,-11,-1,3
6	a&,-51,-2,3
7	,,-76,-1,3
8	b&,-51,-2,3
9	,,-75,-1,3
10	real,-12,-1,4
11	x%,,-52,-2,4
12	,,-76,-1,4
13	y%,,-52,-2,4
14	,,-75,-1,4
15	cadena,-13,-1,5
16	z\$,-53,-2,5
17	,,-75,-1,5
18	logico,-14,-1,6
19	dos#,,-54,-2,6
20	,,-75,-1,6
21	inicio,-2,-1,7
22	a&,-51,-2,8
23	=,-26,-1,8
24	300,-61,-1,8
25	,,-75,-1,8
26	b&,-51,-2,9
27	=,-26,-1,9
28	0,-61,-1,9
29	,,-75,-1,9
30	z\$,-53,-2,10
31	=,-26,-1,10
32	"Hola",-63,-1,10
33	,,-75,-1,10
34	x%,,-52,-2,11
35	=,-26,-1,11
36	45.5,-62,-1,11
37	,,-75,-1,11
38	y%,,-52,-2,12
39	=,-26,-1,12
40	6.6,-62,-1,12
41	,,-75,-1,12
42	dos#,,-54,-2,13
43	=,-26,-1,13
44	true,-64,-1,13
45	,,-75,-1,13
46	Fin,-3,-1,14
47	

Diagrama de clase



Ejecución



```
Output - Proyecto (run) x
run:
Procesamiento completado. Se ha generado el archivo: C:\Users\rogel\OneDrive\Escritorio\Semestre 8\Lenguajes Y Autómatas II\Proyecto\src\Recursos\Table de Simbolos.txt
Lineas repetidas eliminadas correctamente.
Se eliminaron las lineas repetidas correctamente.
Lineas eliminadas correctamente.

BUILD SUCCESSFUL (total time: 0 seconds)
```

Resultado

```
1  programa,-1,-1,1
2  UNO@,-55,-2,1
3  ;,-75,-1,1
4  variables,-15,-1,2
5  entero,-11,-1,3
6  a&,-51,0,3
7  ,,-76,-1,3
8  b&,-51,1,3
9  ;,-75,-1,3
10 real,-12,-1,4
11 x%, -52,2,4
12 ,,-76,-1,4
13 y%, -52,3,4
14 ;,-75,-1,4
15 cadena,-13,-1,5
16 z$, -53,4,5
17 ;,-75,-1,5
18 logico,-14,-1,6
19 dos#, -54,5,6
20 ;,-75,-1,6
21 inicio,-2,-1,7
22 a&,-51,0,8
23 =,-26,-1,8
24 300,-61,-1,8
25 ;,-75,-1,8
26 b&,-51,1,9
27 =,-26,-1,9
28 0,-61,-1,9
29 ;,-75,-1,9
30 z$, -53,4,10
31 =,-26,-1,10
32 "Hola",-63,-1,10
33 ;,-75,-1,10
34 x%, -52,2,11
35 =,-26,-1,11
36 45.5,-62,-1,11
37 ;,-75,-1,11
38 y%, -52,3,12
39 =,-26,-1,12
40 6.6,-62,-1,12
41 ;,-75,-1,12
42 dos#, -54,5,13
43 =,-26,-1,13
44 true,-64,-1,13
45 ;,-75,-1,13
46 Fin,-3,-1,14
47
```

Versión 2

Método Main

```
Start Page x AnalysisSemantico.java x
Source History Navigator Projects Files Services
1 package Analisis_Semantico;
2
3 import java.io.BufferedReader;
4 import java.io.BufferedWriter;
5 import java.io.FileReader;
6 import java.io.FileWriter;
7 import java.io.IOException;
8 import java.util.ArrayList;
9 import java.util.Set;
10 import java.util.HashSet;
11
12 /**
13 * Practica 1 Analisis Semantico
14 * @author Rogelio Perez Guevara
15 * @author Equipo 2
16 * @version 05/03/2024
17 */
18
19 public class AnalisisSemantico
20 {
21     public static void main (String [] args) throws IOException
22     {
23         // declara los archivos de entrada y de salida
24         String archivoTokens = "C:\\Users\\rogel\\OneDrive\\Escritorio\\Semestre 8\\Lenguajes Y Autómatas II\\Proyecto\\src\\Recursos\\Tabla de Tokens.txt";
25         String archivoSimbolos = "C:\\Users\\rogel\\OneDrive\\Escritorio\\Semestre 8\\Lenguajes Y Autómatas II\\Proyecto\\src\\Recursos\\Tabla de Simbolos";
26         String archivoDirecciones = "C:\\Users\\rogel\\OneDrive\\Escritorio\\Semestre 8\\Lenguajes Y Autómatas II\\Proyecto\\src\\Recursos\\Tabla de Direcciones";
27         String archivoNuevoTokens = "C:\\Users\\rogel\\OneDrive\\Escritorio\\Semestre 8\\Lenguajes Y Autómatas II\\Proyecto\\src\\Recursos\\Tabla de Tokens";
28         String valor = null;
29
30         // crea la lectura y escritura de los archivos
31         try (BufferedReader br = new BufferedReader(new FileReader(archivoTokens));
32              BufferedWriter bw = new BufferedWriter(new FileWriter(archivoSimbolos));
33              BufferedWriter dw = new BufferedWriter(new FileWriter(archivoNuevoTokens)))
34         {
35             // linea almacena la linea leida del archivo
36             String linea;
37
38             // linea almacen la linea leida del archivo
39             String linea;
40
41             // en este ciclo lo que se hace es separar en 4 partes la cadena de entrada
42             while ((linea = br.readLine()) != null)
43             {
44                 String[] partes = linea.split("\\s+", 4);
45                 String primeraParte = partes[0].trim();
46                 String segundaParte = partes[1].trim();
47                 String cuartaParte = partes[3].trim();
48
49                 // aqui de la primera parte de la cadena lo que se hace es asignar el tipo de valor que es
50                 if(primeraparte.equals("anObject:entero"))
51                 {
52                     valor = "0";
53                 }
54                 else if(primeraparte.equals("anObject:real"))
55                 {
56                     valor = "0.0";
57                 }
58                 else if(primeraparte.equals("anObject:cadena"))
59                 {
60                     valor = "Null";
61                 }
62                 else if(primeraparte.equals("anObject:logico"))
63                 {
64                     valor = "True";
65                 }
66
67                 if(primeraparte.contains(":"))
68                 {
69                     dw.write(primeraparte + ",");
70                     dw.write(segundaParte + ",");
71                     dw.write(cuartaParte + ",");
72                     dw.write(valor + "\n");
73                 }
74             }
75         }
76     }
77 }
```

```
35             // linea almacen la linea leida del archivo
36             String linea;
37
38             // en este ciclo lo que se hace es separar en 4 partes la cadena de entrada
39             while ((linea = br.readLine()) != null)
40             {
41                 String[] partes = linea.split("\\s+", 4);
42                 String primeraParte = partes[0].trim();
43                 String segundaParte = partes[1].trim();
44                 String cuartaParte = partes[3].trim();
45
46                 // aqui de la primera parte de la cadena lo que se hace es asignar el tipo de valor que es
47                 if(primeraparte.equals("anObject:entero"))
48                 {
49                     valor = "0";
50                 }
51                 else if(primeraparte.equals("anObject:real"))
52                 {
53                     valor = "0.0";
54                 }
55                 else if(primeraparte.equals("anObject:cadena"))
56                 {
57                     valor = "Null";
58                 }
59                 else if(primeraparte.equals("anObject:logico"))
60                 {
61                     valor = "True";
62                 }
63
64                 if(primeraparte.contains(":"))
65                 {
66                     dw.write(primeraparte + ",");
67                     dw.write(segundaParte + ",");
68                     dw.write(cuartaParte + ",");
69                     dw.write(valor + "\n");
70                 }
71             }
72         }
73     }
74 }
```

The screenshot shows a Java IDE interface with the following details:

- Title Bar:** Start Page x AnalysisSemantico.java
- Menu Bar:** Source, History, File, Edit, View, Tools, Window, Help
- Left Sidebar:** Navigator, Projects, Files, Services
- Code Editor:** The main area displays the `AnalysisSemantico.java` file. The code is as follows:

```
72     //Si en el la segunda parte encuentra un tipo de datos entre -51 y -54
73     //Escribe en el archivo de salida la primera y segunda parte, el valor
74     //asignado y Main todo separado por la ","
75     if (segundaParte.equals(anObject: "-51") || segundaParte.equals(anObject: "-52") || segundaParte.equals(anObject: "-53") || segundaParte.equals(anObject: "-54"))
76     {
77         bw.write(primerasParte + ",");
78         bw.write(segundaParte + ",");
79         bw.write(valor + ",");
80         bw.write(main: "\n");
81     }
82     //Imprime un mensaje para dar por completado el proceso
83     System.out.println("Se ha generado el archivo: " + archivoSimbolos);
84 }
85 catch (IOException e)
86 {
87     e.printStackTrace();
88 }
89
90 //Llama al metodo para eliminar la duplicidad
91 try
92 {
93     eliminarLineasRepetidas( inputFile, archivoSimbolos);
94     System.out.println("Procesamiento completado.");
95 }
96 catch (IOException e)
97 {
98     System.err.println("Error al procesar el archivo: " + e.getMessage());
99 }
100
101 tk2( leerTS: archivoSimbolos, leerT: archivoTokens, escribirNT: archivoNuevoTokens);
102 estaBien( leerT: archivoTokens);
103
104 }
```

Método `estaBien`

```
Start Page x AnalysisSemantic.java x
Source History Navigator Projects Services Files
private static void estaBien(String leer)
{
    try
    {
        BufferedReader leerToken = new BufferedReader(new FileReader(string: linea));
        String linea;
        ArrayList<String> enteros = new ArrayList<>();
        ArrayList<String> reales = new ArrayList<>();
        ArrayList<String> cadena = new ArrayList<>();
        ArrayList<String> logicos = new ArrayList<>();
        ArrayList<String> general = new ArrayList<>();

        while ((linea = leerToken.readLine()) != null)
        {
            String[] partes = linea.split(regex: ";", limit: 4);
            switch (partes[1].trim())
            {
                case "-51":
                    enteros.add(partes[0].trim().replace(target: "#", replacement: ""));
                    break;
                case "-52":
                    reales.add(partes[0].trim().replace(target: "%", replacement: ""));
                    break;
                case "-53":
                    cadena.add(partes[0].trim().replace(target: "€", replacement: ""));
                    break;
                case "-54":
                    logicos.add(partes[0].trim().replace(target: "§", replacement: ""));
                    break;
                case "-55":
                    general.add(partes[0].trim().replace(target: "@", replacement: ""));
                    break;
            }
        }

        for (String elemento : enteros)
        {
            if(reales.contains(elemento) || cadena.contains(elemento) || logicos.contains(elemento) || general.contains(elemento))
        }
    }
}
```

```
Start Page x AnalysisSemantic.java x
Source History Navigator Projects Services Files
        for (String elemento : enteros)
        {
            if(reales.contains(elemento) || cadena.contains(elemento) || logicos.contains(elemento) || general.contains(elemento))
            {
                throw new IOException(string: "la variable esta declarada en 2 tipos de datos");
            }
        }

        for (String elemento : reales)
        {
            if(enteros.contains(elemento) || cadena.contains(elemento) || logicos.contains(elemento)||general.contains(elemento))
            {
                throw new IOException(string: "la variable esta declarada en 2 tipos de datos");
            }
        }

        for (String elemento : cadena)
        {
            if(reales.contains(elemento) || enteros.contains(elemento) || logicos.contains(elemento)|| general.contains(elemento))
            {
                throw new IOException(string: "la variable esta declarada en 2 tipos de datos");
            }
        }

        for (String elemento : logicos)
        {
            if(reales.contains(elemento) || cadena.contains(elemento) || enteros.contains(elemento)|| general.contains(elemento))
            {
                throw new IOException(string: "la variable esta declarada en 2 tipos de datos");
            }
        }

        for (String elemento : general)
        {
            if(reales.contains(elemento) || cadena.contains(elemento) || enteros.contains(elemento)|| logicos.contains(elemento))
            {
                throw new IOException(string: "la variable esta declarada en 2 tipos de datos");
            }
        }
    }
}
```

```
Start Page x AnalysisSemantic.java x
Source History Navigator Projects Services Files
    for (String elemento : general)
    {
        if(reales.contains(elemento) || cadena.contains(elemento) || enteros.contains(elemento)|| logicos.contains(elemento))
        {
            throw new IOException(string: "la variable esta declarada en 2 tipos de datos");
        }
    }
}

catch (IOException e)
{
    System.out.println(e.getMessage());
}
}
```

Método tk2

The screenshot shows a Java code editor with the file 'AnalisisSemanticoo.java' open. The code implements the `tk2` method, which reads from two files and writes to another. It uses BufferedReader and BufferedWriter to handle lines and symbols respectively. The code includes logic to split lines into four parts and write them to the output file.

```
private static void tk2(String leerTS, String leerT, String escribirNT)
{
    try (BufferedReader leerToken = new BufferedReader(new FileReader(string:leerT));
         BufferedWriter t2 = new BufferedWriter(new FileWriter(string:escribirNT)))
    {
        String linea;
        String linea2;
        int cont = 0;

        while ((linea = leerToken.readLine()) != null)
        {
            String[] partes = linea.split(regex:",", limit:4);
            String primeraParte = partes[0].trim();
            String segundaParte = partes[1].trim();
            String terceraParte = partes[2].trim();
            String cuartaParte = partes[3].trim();
            BufferedReader leerSimbolos = new BufferedReader(new FileReader(string:leerTS));

            while ((linea2 = leerSimbolos.readLine()) !=null)
            {
                String[] partesSimbolos = linea2.split(regex:",", limit:4);
                String primeraParteSimbolo = partesSimbolos[0].trim();

                if(primeraparte.equals(anObject:primeraParteSimbolo))
                {
                    terceraParte = "" + cont;
                }
                cont++;
            }
            leerSimbolos.close();
            t2.write(primeraparte + "," + segundaParte + "," + terceraParte + "," + cuartaParte + "\n");
            cont=0;
        }
        leerToken.close();
        t2.close();
    } catch (IOException e)
    {
        System.out.println("no se pudo renombrar");
    }
}
```

The screenshot shows the same Java code editor, focusing on the catch block of the `tk2` method. It handles `IOException` by printing a message to the console indicating that the file could not be renamed.

```
catch (IOException e)
{
    System.out.println("no se pudo renombrar");
}
```

Método eliminarLineasRepetidas

The screenshot shows a Java code editor with the file `AnalisisSemantico.java` open. The code implements a method to remove duplicate lines from an input file. It uses a BufferedReader to read lines, a HashSet to store unique lines, and a StringBuilder to build the result. The code includes comments explaining the logic for reading lines, splitting them into parts, and checking if they are already in the set before appending them to the result.

```
228 //Metodo para eliminar elementos duplicados en la generacion de la tabla de simbolos
229 //El metodo recibe el archivo de salida
230 public static void eliminarLineasRepetidas(String inputFile) throws IOException
231 {
232     BufferedReader reader = new BufferedReader(new FileReader(inputFile));
233     Set<String> firstParts = new HashSet();
234     StringBuilder result = new StringBuilder();
235
236     //Variable para almacenar el contenido linea por linea
237     String line;
238
239     //Se inicializa un ciclo para dar lectura al archivo de salida linea por linea
240     while ((line = reader.readLine()) != null)
241     {
242         String[] parts = line.split(",", -1); // Divide la linea en cuatro partes utilizando ","
243         String firstPart = parts[0]; // La primera parte antes de la ","
244
245         if (!firstParts.contains(firstPart))
246         {
247             // Si esta es la primera vez que vemos esta primera parte, la agregamos al resultado
248             result.append(line).append("\n");
249             firstParts.add(firstPart);
250         }
251     }
252
253     // Cerrar el archivo
254     reader.close();
255
256     // Escribir el resultado de vuelta al archivo de entrada
257     BufferedWriter writer = new BufferedWriter(new FileWriter(inputFile));
258     writer.write(result.toString());
259     writer.close();
260 }
```

Tabla de Símbolos

1	a&,-51,0,Main
2	b&,-51,0,Main
3	x%, -52, 0.0, Main
4	y%, -52, 0.0, Main
5	z\$, -53, Null, Main
6	dos#, -54, True, Main
7	

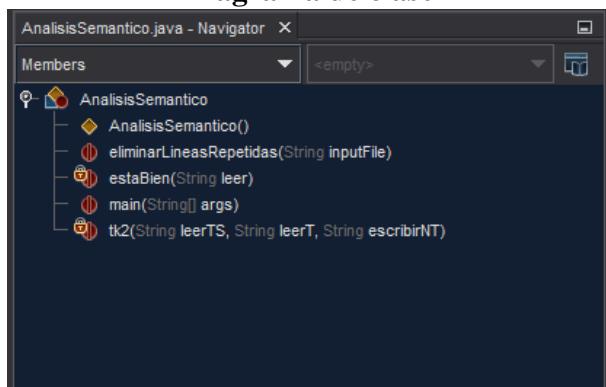
Tabla de Direcciones

1	JNO@,-55,1,0
---	--------------

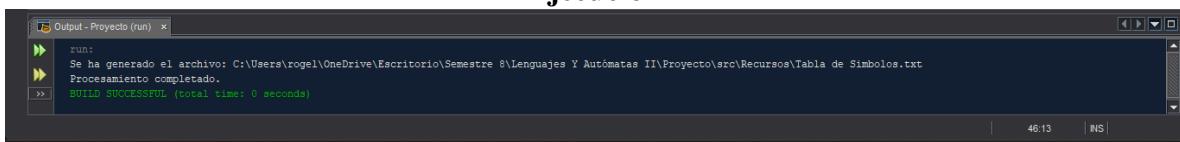
Tabla de Tokens

1	programa,-1,-1,1
2	UNO@,-55,-2,1
3	;-,-75,-1,1
4	variables,-15,-1,2
5	entero,-11,-1,3
6	a&,-51,-2,3
7	,,-76,-1,3
8	b&,-51,-2,3
9	,,-75,-1,3
10	real,-12,-1,4
11	x%,,-52,-2,4
12	,,-76,-1,4
13	y%,,-52,-2,4
14	,,-75,-1,4
15	cadena,-13,-1,5
16	z\$,-53,-2,5
17	,,-75,-1,5
18	logico,-14,-1,6
19	dos#,,-54,-2,6
20	,,-75,-1,6
21	inicio,-2,-1,7
22	a&,-51,-2,8
23	=,-26,-1,8
24	300,-61,-1,8
25	,,-75,-1,8
26	b&,-51,-2,9
27	=,-26,-1,9
28	0,-61,-1,9
29	,,-75,-1,9
30	z\$,-53,-2,10
31	=,-26,-1,10
32	"Hola",-63,-1,10
33	,,-75,-1,10
34	x%,,-52,-2,11
35	=,-26,-1,11
36	45.5,-62,-1,11
37	,,-75,-1,11
38	y%,,-52,-2,12
39	=,-26,-1,12
40	6.6,-62,-1,12
41	,,-75,-1,12
42	dos#,,-54,-2,13
43	=,-26,-1,13
44	true,-64,-1,13
45	,,-75,-1,13
46	Fin,-3,-1,14
47	

Diagrama de clase



Ejecución



The screenshot shows the 'Output' window of an IDE during a project run. The title bar reads 'Output - Proyecto (run)'. The main area contains the following log entries:

```
>> run:  
>> Se ha generado el archivo: C:\Users\rogel\OneDrive\Escritorio\Semestre 8\Lenguajes Y Autómatas II\Proyecto\src\Recursos\Table de Simbolos.txt  
>> Procesamiento completado.  
BUILD SUCCESSFUL (total time: 0 seconds)
```

In the bottom right corner of the window, there are status indicators: '46:13' and 'INS'.

Resultado

Archivo original

```
1  programa,-1,-1,1
2  UNO@,-55,-2,1
3  ;,-75,-1,1
4  variables,-15,-1,2
5  entero,-11,-1,3
6  a&,-51,-2,3
7  ,,-76,-1,3
8  b&,-51,-2,3
9  ;,-75,-1,3
10 real,-12,-1,4
11 x%, -52,-2,4
12 ,,-76,-1,4
13 y%, -52,-2,4
14 ;,-75,-1,4
15 cadena,-13,-1,5
16 z$,-53,-2,5
17 ;,-75,-1,5
18 logico,-14,-1,6
19 dos#, -54,-2,6
20 ;,-75,-1,6
21 inicio,-2,-1,7
22 a&,-51,-2,8
23 =,-26,-1,8
24 300,-61,-1,8
25 ;,-75,-1,8
26 b&,-51,-2,9
27 =,-26,-1,9
28 0,-61,-1,9
29 ;,-75,-1,9
30 z$,-53,-2,10
31 =,-26,-1,10
32 "Hola",-63,-1,10
33 ;,-75,-1,10
34 x%, -52,-2,11
35 =,-26,-1,11
36 45.5,-62,-1,11
37 ;,-75,-1,11
37 ;,-75,-1,11
38 y%, -52,-2,12
39 =,-26,-1,12
40 6.6,-62,-1,12
41 ;,-75,-1,12
42 dos#, -54,-2,13
43 =,-26,-1,13
44 true,-64,-1,13
45 ;,-75,-1,13
46 Fin,-3,-1,14
```

Nuevo archivo

```
1  programa,-1,-1,1
2  UNO@,-55,-2,1
3  ;,-75,-1,1
4  variables,-15,-1,2
5  entero,-11,-1,3
6  a&,-51,0,3
7  ,,-76,-1,3
8  b&,-51,1,3
9  ;,-75,-1,3
10 real,-12,-1,4
11 x%, -52,2,4
12 ,,-76,-1,4
13 y%, -52,3,4
14 ;,-75,-1,4
15 cadena,-13,-1,5
16 z$, -53,4,5
17 ;,-75,-1,5
18 logico,-14,-1,6
19 dos#, -54,5,6
20 ;,-75,-1,6
21 inicio,-2,-1,7
22 a&,-51,0,8
23 =,-26,-1,8
24 300,-61,-1,8
25 ;,-75,-1,8
26 b&,-51,1,9
27 =,-26,-1,9
28 0,-61,-1,9
29 ;,-75,-1,9
30 z$, -53,4,10
31 =,-26,-1,10
32 "Hola",-63,-1,10
33 ;,-75,-1,10
34 x%, -52,2,11
35 =,-26,-1,11
36 45.5,-62,-1,11
37 ;,-75,-1,11
37 ;,-75,-1,11
38 y%, -52,3,12
39 =,-26,-1,12
40 6.6,-62,-1,12
41 ;,-75,-1,12
42 dos#, -54,5,13
43 =,-26,-1,13
44 true,-64,-1,13
45 ;,-75,-1,13
46 Fin,-3,-1,14
47
```

Cambios

En ambas versiones del código, realizamos cambios significativos que incluyeron la eliminación de variables, líneas de código y métodos que no se utilizaban en ningún momento durante la ejecución. Estos ajustes resultaron en una optimización general del código.

Conclusión

Rogelio: Para la realización de la práctica, tuvimos que repartir las actividades a desarrollar. Sin embargo, encontramos ciertas dificultades debido a cuestiones institucionales, lo que complicó cumplir con los plazos establecidos. Por ello, mi compañera Karina y yo nos encargamos de desarrollar las bases del programa, mientras que nuestro compañero Idwin se dedicó a corregir errores y finalizar el programa para alcanzar el objetivo.

General: En la realización de nuestra práctica 1, desarrollamos dos versiones del código. La principal diferencia entre ambas radica en el manejo de la tabla de tokens. En la versión 1, los valores de la tabla de tokens se sustituyen por nuevos valores, mientras que en la versión 2, la tabla de tokens se mantiene intacta y se genera un nuevo archivo con los valores actualizados.

Además, la versión 2 cuenta con menos métodos que la versión 1, aunque el funcionamiento y los resultados de ejecución permanecen sin cambios. Finalmente, se realizaron optimizaciones eliminando líneas de código, variables y métodos que no se utilizaban en ningún momento, mejorando así la eficiencia del código.

Practica No. 2 Simulación del vector de código intermedio

Objetivo

El alumno aplica los conceptos asociados para la obtención del Vector de código intermedio en expresiones aritméticas/lógica/relacionales y estructuras de control (condicional, repetición) simulando su proceso utilizando lenguaje de programación JAVA.

Descripción

Considerando la tabla de tokens que obtuvo como salida en la práctica anterior genere una aplicación que permita simular **el proceso de obtención del vector de código intermedio** para un conjunto de código que incluya expresiones aritméticas/lógicas/relacionales (cualquiera), estructuras de control (condicional, repetir, mientras), considere operadores y prioridades vistas en clase, incluya prioridad para paréntesis y **utilice las pilas** correspondientes (operadores, estatutos, direcciones). **INCLUYA diseño** lógico de la aplicación o programa en el reporte y considere manejo de errores, etapas de desarrollo de software, considere metodología de diseño para la aplicación o programa acorde al paradigma del lenguaje.

El conjunto de datos de entrada no debe presentar errores (léxicos, sintácticos ni semánticos). La salida será el VCI generado de manera correcta que se desplegará en pantalla y **se almacenará en un archivo de texto etiquetado VCI**.

EJEMPLO

ENTRADA:

Tabla de tokens

TOKENS	TOKENS
programa,-1,-1,1	b&, -51, 1 , 9
uno@,-55, 0 ,1	=, -26 , -1 , 9
; , -75 , -1 , 1	0, -61, -1, 9
variables, -15, -1, 2	; , -75, -1, 9
entero, -11, -1, 3	z\$, -53, 4, 10
a&, -51, 0 , 3	= , -26 , -1 , 10
, , -76 , -1 , 3	“Hola”, -63, -1, 10
b& , -51 , 1 , 3	; , -75, -1, 10
; , -75 , -1 , 3	x% , -52, 2, 11
real, -12, -1 , 4	= , -26 , -1 , 11
x%, -52, 2 , 4	45.5, -62, -1, 11
, , -76 , -1 , 4	; , -75, -1, 11
y%, -52, 3 , 4	y%, -52, 3, 12
; , -75, -1, 4	=, -26 , -1 , 12
cadena, -13, -1 , 5	6.6, -62, -1, 12
z\$, -53 , 4 , 5	; , -75, -1, 12
; , -75 , -1 , 5	dos#, -54, 5, 13
logico, -14, -1 , 6	=, -26 , -1 , 13
dos#, -54 , 5 , 6	true, -64, -1, 13
; , -75 , -1 , 6	; , -75, -1, 13
inicio , -2, -1,7	Fin, -3,-1,14
a& , -51, 0 , 8	
=, -26 , -1 , 8	
300, -61, -1, 8	
; , -75, -1, 8	

SALIDA:

Vector de código intermedio (TABLA) considerando tabla de TOKENS

a& , -51, 0 , 8
300, -61, -1, 8
=, -26 , -1 , 8
b&, -51, 1 , 9
0, -61, -1, 9
=, -26 , -1 , 9
z\$, -53, 4, 10
“Hola”, -63, -1, 10
= , -26 , -1 , 10
.... etc

Desarrollo

Para alcanzar el objetivo de esta práctica, se distribuyó el trabajo de la siguiente manera:

- Idwin se encargó de desarrollar la sección de operadores, lógica y relaciones.
- Rogelio asumió la responsabilidad de la sección aritmética.
- Karina se encargó de la sección de condicionales y bucles.

En colaboración, se hizo la elaboración del documento, integrando los avances individuales de cada participante y se realizaron unas 10 pruebas con tablas de token para corroborar el funcionamiento.

Cuando el equipo completó el programa de esta práctica, Rogelio decidió desarrollar otro con la misma base del código principal creado por el equipo. La distinción principal radica en que, en lugar de emplear el lexema utilizado en el código principal, este nuevo código utiliza el token correspondiente (número de token).

Versión 1

Método Main

```
Start Page x Vector_Código_Intermedio.java x
Source History Projects Files Services Navigator
1 package Vector_Código_Intermedio;
2
3 import java.io.BufferedReader;
4 import java.io.BufferedWriter;
5 import java.io.FileReader;
6 import java.io.FileWriter;
7 import java.io.IOException;
8 import java.util.ArrayList;
9 import java.util.Stack;
10 import java.util.Scanner;
11 import java.nio.file.Files;
12 import java.nio.file.Paths;
13 import java.nio.file.Paths;
14 import java.util.List;
15
16 /**
17 *
18 * @author Rogelio Pérez Guevara
19 * @author Equipo 2
20 * @version 25/03/2024
21 */
22 public class Vector_Código_Intermedio
23 {
24     public static Stack<String> pilaDeOperadores = new Stack<>();
25     public static Stack<Integer> pilaDePrioridad = new Stack<>();
26     public static Stack<String> pilaDeOperadoresTokens = new Stack<>();
27
28     public static Stack<String> pilaDeEstatutos = new Stack<>();
29     public static Stack<Integer> pilaDeDirecciones = new Stack<>();
30
31     public static ArrayList<String> cintaDeVCI = new ArrayList<>();
32     Public static ArrayList<Integer> cintaDeVCIApuntador = new ArrayList<>();
33
34     public static int apuntador = 0;
35
36     public static void main (String [] args) throws IOException
37 }
```

The screenshot shows the beginning of a Java class named `Vector_Código_Intermedio`. It includes imports for various Java IO classes like `BufferedReader`, `FileWriter`, and `Paths`. A detailed multi-line comment at the top provides authorship information: `Author: Rogelio Pérez Guevara, Author: Equipo 2, Version: 25/03/2024`. The class starts with a `main` method that throws `IOException`.

```
Start Page x Vector_Código_Intermedio.java x
Source History Projects Files Services Navigator
37 public static void main (String [] args) throws IOException
38 {
39     long tiempo_inicial = System.currentTimeMillis();
40
41     // Leer el archivo de texto linea por linea
42     try (BufferedReader br = new BufferedReader(new FileReader("C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Automatas II\\\\Pro
43     {
44         String linea;
45         String lineaSiguiente = null;
46         String guardado = null;
47         String temporal = null;
48         String token = null;
49         String hasta = null;
50         String simb = null;
51         ArrayList<String> inicio = new ArrayList<>();
52         ArrayList<String> mientras = new ArrayList<>();
53         int apuntador2;
54         String resultado = null;
55         boolean bandera = false;
56
57         while ((linea = br.readLine()) != null)
58         {
59             // Dividir la linea en partes
60
61             String[] partes = linea.split(",");
62             String palabra = partes[0];
63
64             // Se ejecutará solo la primera vez que se encuentre la palabra "inicio"
65             if(!bandera && "inicio".equals(anObject.palabra))
66             {
67                 bandera = true;
68             }
69
70             if(bandera == true)
71             {
72                 // Reglas de análisis lexico
73                 switch (palabra)
```

This screenshot continues the `main` method. It begins with the declaration of variables used for reading input from a file. These include stacks for operators, priorities, and tokens, as well as lists for statements and loops. It also initializes variables for the current line, the next line, and flags for handling specific words like "inicio". The logic involves reading each line and splitting it into parts to identify tokens.

The screenshot shows a Java IDE interface with a code editor containing Java code. The code implements lexical analysis rules for tokens like 'si', 'entonces', and 'sino'. It uses stacks for operators and priorities, and a tape for the input. The code is annotated with line numbers from 72 to 108.

```
72 // Reglas de análisis léxico
73 switch (palabra)
74 {
75     case "si":
76         pilaDeEstatutos.push(item:lineas);
77         break;
78     case "entonces":
79         guardado = linea;
80
81         while (!pilaDeOperadores.isEmpty())
82         {
83             pilaDeOperadores.pop();
84             pilaDePrioridad.pop();
85             cintaDeVCI.add(cintaDeOperadoresTokens.pop());
86             cintaDeVCIApuntador.add(apuntador++);
87         }
88
89         pilaDeDirecciones.push(item:apuntador);
90
91         cintaDeVCI.add(c: "L");
92         cintaDeVCIApuntador.add(c: apuntador);
93         apuntador++;
94
95         cintaDeVCI.add(c: guardado);
96         cintaDeVCIApuntador.add(apuntador++);
97         guardado = null;
98         break;
99     case "sino":
100        guardado = linea;
101        sino = "sino";
102
103        pilaDeEstatutos.push(item:lineas);
104
105        if (!pilaDeDirecciones.isEmpty())
106        {
107            apuntador2 = pilaDeDirecciones.pop();
108            for (int i = 0; i < cintaDeVCI.size(); i++)
109            {
110                if (i == apuntador2)
111                {
112                    cintaDeVCI.set(index:i, (apuntador + 2) + "");
113                    break;
114                }
115            }
116        }
117
118        cintaDeVCI.add(c: "L");
119        pilaDeDirecciones.push(item:apuntador);
120        cintaDeVCIApuntador.add(apuntador++);
121
122        cintaDeVCI.add(c: guardado);
123        cintaDeVCIApuntador.add(apuntador++);
124        guardado = null;
125        break;
126    case "fin":
127        if (!pilaDeEstatutos.isEmpty())
128        {
129            pilaDeEstatutos.pop();
130        }
131
132        String archivo = "C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Autómatas II\\\\Proyecto\\\\src\\\\Recursos\\\\Tabla";
133        String palabraBuscada = linea;
134
135        try (BufferedReader br2 = new BufferedReader(new FileReader(string:archivo)))
136        {
137            String linea2;
138            boolean palabraEncontrada = false;
139
140            // Leemos el archivo linea por linea
141            while ((linea2 = br2.readLine()) != null)
142            {
143                // Buscamos la palabra en la linea
```

The screenshot shows a Java IDE interface with a code editor containing Java code. The code reads a file named 'Tabla' located at 'C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Autómatas II\\\\Proyecto\\\\src\\\\Recursos\\\\Tabla'. It uses a BufferedReader to read lines and checks if the current line contains the word 'palabraBuscada'. The code is annotated with line numbers from 107 to 143.

```
107 apuntador2 = pilaDeDirecciones.pop();
108 for (int i = 0; i < cintaDeVCI.size(); i++)
109 {
110     if (i == apuntador2)
111     {
112         cintaDeVCI.set(index:i, (apuntador + 2) + "");
113         break;
114     }
115 }
116
117 cintaDeVCI.add(c: "L");
118 pilaDeDirecciones.push(item:apuntador);
119 cintaDeVCIApuntador.add(apuntador++);
120
121 cintaDeVCI.add(c: guardado);
122 cintaDeVCIApuntador.add(apuntador++);
123 guardado = null;
124 break;
125
126 case "fin":
127     if (!pilaDeEstatutos.isEmpty())
128     {
129         pilaDeEstatutos.pop();
130     }
131
132 String archivo = "C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Autómatas II\\\\Proyecto\\\\src\\\\Recursos\\\\Tabla";
133 String palabraBuscada = linea;
134
135 try (BufferedReader br2 = new BufferedReader(new FileReader(string:archivo)))
136 {
137     String linea2;
138     boolean palabraEncontrada = false;
139
140     // Leemos el archivo linea por linea
141     while ((linea2 = br2.readLine()) != null)
142     {
143         // Buscamos la palabra en la linea
```

The screenshot shows a Java code editor with the file `Vector_Código_Intermedio.java` open. The code is part of a simulation for a Turing Machine. It includes logic to search for words in lines, handle loops like 'mientras' and 'hasta', and manage pointers on a tape. The code uses various Java classes like `ArrayList` and `Stack`.

```
143         // Buscamos la palabra en la linea
144         if (linea2.contains(:palabraBuscada))
145         {
146             palabraEncontrada = true;
147             // Leemos la linea siguiente si existe
148             lineaSiguiente = br2.readLine();
149             break; // Salimos del bucle una vez que encontramos la palabra
150         }
151     }
152
153
154     resultado = obtenerPrimerElementoSplit(cadena:lineaSiguiente);
155     // Procesar la linea siguiente si es necesario
156     if ("sino".equals(:anObject:resultado))
157     {
158         resultado = null;
159         inicio.remove(:"inicio");
160         break;
161     }
162     else if ("hasta".equals(:anObject:resultado))
163     {
164         break;
165     }
166     else if ("mientras.isEmpty() && mientras.equals(:anObject:mientras.get(mientras.size() - 1)) && sino == null")
167     {
168         if (!pilaDeDirecciones.isEmpty())
169         {
170             apuntador2 = pilaDeDirecciones.pop();
171
172             for (int i = 0; i < cintaDeVCI.size(); i++)
173             {
174                 if (i == apuntador2)
175                 {
176                     cintaDeVCI.set(index:i, (apuntador + 2) + "");
177                     break;
178                 }
179             }
180         }
181
182         if (!pilaDeDirecciones.isEmpty())
183         {
184             apuntador2 = pilaDeDirecciones.pop();
185             cintaDeVCI.add((apuntador2) + "");
186             cintaDeVCI(apuntador.add(apuntador++));
187         }
188
189         cintaDeVCI.add(:"fin-mientras");
190         cintaDeVCI(apuntador.add(apuntador++));
191         mientras.remove(:"inicio");
192         mientras.remove(:"mientras");
193         break;
194     }
195     else if(!inicio.isEmpty() && "inicio".equals(:anObject:inicio.get(inicio.size() - 1)))
196     {
197         sino = null;
198         if (!pilaDeDirecciones.isEmpty())
199         {
200             int apu = pilaDeDirecciones.pop();
201
202             for (int i = 0; i < cintaDeVCI.size(); i++)
203             {
204                 if (i == apu)
205                 {
206                     if ("L".equals(:anObject:cintaDeVCI.get(i-2)))
207                     {
208                         cintaDeVCI.set(i-2, apuntador + "");
209                     }
210                     else if ("L".equals(:anObject:cintaDeVCI.get(i-1)))
211                     {
212                         cintaDeVCI.set(i-1, apuntador + "");
213                     }
214                     else if ("L".equals(:anObject:cintaDeVCI.get(index:1)))
215                     {
216                         apuntador = index;
217                     }
218                 }
219             }
220         }
221     }
222 }
```

This screenshot continues the Java code from the previous block, focusing on pointer manipulation and tape state updates. It shows how the simulation handles different states based on pointers and tape symbols.

```
179         }
180
181         if (!pilaDeDirecciones.isEmpty())
182         {
183             apuntador2 = pilaDeDirecciones.pop();
184             cintaDeVCI.add((apuntador2) + "");
185             cintaDeVCI(apuntador.add(apuntador++));
186         }
187
188         cintaDeVCI.add(:"fin-mientras");
189         cintaDeVCI(apuntador.add(apuntador++));
190         mientras.remove(:"inicio");
191         mientras.remove(:"mientras");
192         break;
193     }
194     else if(!inicio.isEmpty() && "inicio".equals(:anObject:inicio.get(inicio.size() - 1)))
195     {
196         sino = null;
197         if (!pilaDeDirecciones.isEmpty())
198         {
199             int apu = pilaDeDirecciones.pop();
200
201             for (int i = 0; i < cintaDeVCI.size(); i++)
202             {
203                 if (i == apu)
204                 {
205                     if ("L".equals(:anObject:cintaDeVCI.get(i-2)))
206                     {
207                         cintaDeVCI.set(i-2, apuntador + "");
208                     }
209                     else if ("L".equals(:anObject:cintaDeVCI.get(i-1)))
210                     {
211                         cintaDeVCI.set(i-1, apuntador + "");
212                     }
213                     else if ("L".equals(:anObject:cintaDeVCI.get(index:1)))
214                     {
215                         apuntador = index;
216                     }
217                 }
218             }
219         }
220     }
221 }
```

```
214         else if ("!=".equals( anObject:cintaDeVCI.get( index:1)))
215         {
216             cintaDeVCI.set( index:i, apuntador + "=");
217         }
218     }
219     mientras.remove(0, "inicio");
220     break;
221 }
222 break;
223 case "":
224     verificarExistencia( temporal:"=", token: linea);
225     break;
226 case "/":
227     verificarExistencia( temporal: "/", token: linea);
228     break;
229 case "%":
230     verificarExistencia( temporal: "%", token: linea);
231     break;
232 case "+":
233     verificarExistencia( temporal: "+", token: linea);
234     break;
235 case "-":
236     verificarExistencia( temporal: "-", token: linea);
237     break;
238 case "<":
239     verificarExistencia( temporal: "<", token: linea);
240     break;
241 case ">":
242     verificarExistencia( temporal: ">", token: linea);
243     break;
244 case "<=":
245     verificarExistencia( temporal: "<=", token: linea);
246     break;
247 case ">=":
248     verificarExistencia( temporal: ">=", token: linea);
249     break;
250 }
```

```
249     case ">=":
250         verificarExistencia( temporal: ">=", token: linea);
251         break;
252     case "==":
253         verificarExistencia( temporal: "==", token: linea);
254         break;
255     case "!=":
256         verificarExistencia( temporal: "!=" , token: linea);
257         break;
258     case "not":
259         verificarExistencia( temporal: "not", token: linea);
260         break;
261     case "and":
262         verificarExistencia( temporal: "and", token: linea);
263         break;
264     case "or":
265         verificarExistencia( temporal: "or", token: linea);
266         break;
267     case "=":
268         verificarExistencia( temporal: "=" , token: linea);
269         break;
270     case "repetir":
271         pilaDeEstatutos.push( item: linea );
272         pilaDeDirecciones.push( item: apuntador );
273         break;
274     case "hasta":
275         hasta = palabra;
276         temporal = linea;
277         mientras.remove(0, "inicio");
278         break;
279     case "(":
280         pilaDeOperadores.push( item: palabra );
281         pilaDePrioridad.push( item: 0 );
282         pilaDeOperadoresTokens.push( item: linea );
283         break;
284     case ")":
285         while(!pilaDeOperadores.isEmpty() && pilaDeOperadores.peek().equals( anObject: "(" )
```

```
284     case ")":
285         while(!pilaDeOperadores.isEmpty() && pilaDeOperadores.peek().equals( anObject: ")")
286         {
287             String t = pilaDeOperadores.pop();
288             pilaDePrioridad.pop();
289             token = pilaDeOperadoresTokens.pop();
290
291             if (!".equals( anObject:t )")
292             {
293                 cintaDeVCI.add( e: token );
294                 cintaDeVCIApuntador.add(apuntador++);
295             }
296             token = null;
297         }
298
299         if (!pilaDeOperadores.isEmpty())
300         {
301             pilaDeOperadores.pop();
302             pilaDePrioridad.pop();
303             token = pilaDeOperadoresTokens.pop();
304             cintaDeVCI.add( e: token );
305             cintaDeVCIApuntador.add(apuntador++);
306         }
307
308         if (!pilaDeOperadores.isEmpty())
309         {
310             //Elimina el ( de la pila
311             pilaDeOperadores.pop();
312             pilaDePrioridad.pop();
313             pilaDeOperadoresTokens.pop();
314         }
315
316         if("hasta".equals( anObject:hasta ))
317         {
318             if (!pilaDeDirecciones.isEmpty())
319             {
320                 guardado = (pilaDeDirecciones.pop()) + "";
321                 cintaDeVCI.add( e: guardado );
322                 cintaDeVCIApuntador.add(apuntador++);
323             }
324
325             hasta = null;
326             guardado = null;
327
328             if(temporal != null)
329             {
330                 cintaDeVCI.add( e: temporal );
331                 cintaDeVCIApuntador.add(apuntador++);
332                 temporal = null;
333             }
334         }
335         break;
336     case "inicio":
337         inicio.add( e: "inicio");
338         break;
339     case "mientras":
340         pilaDeEstatutos.push( item:lineas );
341         pilaDeDirecciones.push( item:apuntador );
342         mientras.add( e: "mientras" );
343         break;
344     case "hacer":
345         while ( !(pilaDeOperadores.isEmpty()))
346         {
347             pilaDeOperadores.pop();
348             pilaDePrioridad.pop();
349             guardado = pilaDeOperadoresTokens.pop();
350             cintaDeVCI.add( e: guardado );
351             cintaDeVCIApuntador.add(apuntador++);
352             guardado = null;
353
354             cintaDeVCI.add( e: "L" );
355             pilaDeDirecciones.push( item: apuntador );
356         }
357     }
```

```
320             guardado = (pilaDeDirecciones.pop()) + "";
321             cintaDeVCI.add( e: guardado );
322             cintaDeVCIApuntador.add(apuntador++);
323         }
324
325         hasta = null;
326         guardado = null;
327
328         if(temporal != null)
329         {
330             cintaDeVCI.add( e: temporal );
331             cintaDeVCIApuntador.add(apuntador++);
332             temporal = null;
333         }
334     }
335     break;
336     case "mientras":
337         pilaDeEstatutos.push( item:lineas );
338         pilaDeDirecciones.push( item:apuntador );
339         mientras.add( e: "mientras" );
340         break;
341     case "hacer":
342         while ( !(pilaDeOperadores.isEmpty()))
343         {
344             pilaDeOperadores.pop();
345             pilaDePrioridad.pop();
346             guardado = pilaDeOperadoresTokens.pop();
347             cintaDeVCI.add( e: guardado );
348             cintaDeVCIApuntador.add(apuntador++);
349             guardado = null;
350
351             cintaDeVCI.add( e: "L" );
352             pilaDeDirecciones.push( item: apuntador );
353         }
354     }
355 }
```

The screenshot shows a Java code editor with the file `Vector_Código_Intermedio.java` open. The code is part of a project named `CintaVCI`. It includes logic for reading tokens from a file, pushing them onto stacks, and then processing them based on operators. The code uses `ArrayList`s for stacks and `BufferedReader` for reading from a file.

```
356     piladeDirecciones.push(item.apuntador);
357     cintaDeVCI.apuntador.add(apuntador++);

358     cintaDeVCI.add(item.linea);
359     cintaDeVCI.apuntador.add(apuntador++);
360     break;
361   case "+":
362     while (!pilaDeOperadores.isEmpty())
363     {
364       piladeOperadores.pop();
365       pilaDePrioridad.pop();
366       token = piladeOperadoresTokens.pop();
367       cintaDeVCI.add(token);
368       cintaDeVCI.apuntador.add(apuntador++);
369       token = null;
370     }
371     break;
372   default:
373     cintaDeVCI.add(item.linea);
374     cintaDeVCI.apuntador.add(apuntador++);
375     break;
376   }
377 }
378 }
379 }
380 }

// Ruta del archivo donde se guardará el texto
String cintaVCI = "C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Automatas II\\\\Proyecto\\\\src\\\\Recursos\\\\CintaVCI.txt";

// Guardamos los ArrayLists en el archivo de texto
guardarArrayListsEnArchivo(arrayList1:cintaDeVCI, arrayList2:cintaDeVCI.apuntador, rutaArchivo:cintaVCI);

// Leemos y mostramos el contenido del archivo de texto
ArrayList<String> primeralinea = new ArrayList<>();
ArrayList<String> segundalinea = new ArrayList<>();

try (BufferedReader br = new BufferedReader(new FileReader(rutaArchivo)))
```

The screenshot shows a Java code editor with the same file `Vector_Código_Intermedio.java` open. This part of the code handles user input via the console. It reads lines from a file, splits them into tokens, and then displays a menu with four options. The user is prompted to enter a number between 1 and 4. If the input is invalid, an error message is printed.

```
392   try (BufferedReader br = new BufferedReader(new FileReader(rutaArchivo)))
393   {
394     String linea;
395
396     while ((linea = br.readLine()) != null)
397     {
398       String[] partes = linea.split(" ");
399       primeraLinea.add(partes[0]);
400       segundaLinea.add(partes[1]);
401     }
402   }
403 catch (IOException e)
404   {
405   System.err.println("Error al leer el archivo: " + e.getMessage());
406   return;
407 }

// Menú de opciones
Scanner scanner = new Scanner(System.in);
int opcion = 0;

while (opcion < 1 || opcion > 4)
{
  System.out.println("Menú de Opciones:");
  System.out.println("1. Cinta VCI con elementos y apuntadores (Horizontal)");
  System.out.println("2. Cinta VCI solo con elementos (Horizontal)");
  System.out.println("3. Cinta VCI con elementos y apuntadores (Vertical)");
  System.out.println("4. Cinta VCI solo con elementos (Vertical)");
  System.out.print("Ingrese el número de la opción que desea (1-4): ");

  if (scanner.hasNextInt())
  {
    opcion = scanner.nextInt();

    if (opcion < 1 || opcion > 4)
    {
      System.out.println("Opción no válida. Por favor, ingrese un número entre 1 y 4.");
    }
  }
}
```

The screenshot shows a Java code editor with the file "Vector_Código_Intermedio.java" open. The code handles user input for file selection and processing. It includes error handling for invalid inputs and cases for four different file reading functions. The code uses standard Java libraries like `System.out.println` and `Scanner`.

```
428     System.out.println("Opción no válida. Por favor, ingrese un número entre 1 y 4.");
429 }
430 else
431 {
432     System.out.println("Entrada inválida. Por favor, ingrese un número.");
433     scanner.next(); // Limpiar el buffer del scanner
434 }
435 }

// Imprimir según la opción seleccionada
// Realizar la acción correspondiente a la opción seleccionada
switch (opcion)
{
    case 1:
        leerArchivoYMostrarContenido(rutarchivo:cintaVCI);
        break;
    case 2:
        leerArchivoYMostrarContenido2(rutarchivo:cintaVCI);
        break;
    case 3:
        leerArchivoYMostrarContenido3(rutarchivo:cintaVCI);
        break;
    case 4:
        leerArchivoYMostrarContenido4(rutarchivo:cintaVCI);
        break;
    default:
        System.out.println("Opción no válida.");
}

// Dar formato al archivo
try
{
    // Procesar el archivo
    formatoArchivo(cintaVCI);
    System.out.println("Datos procesados y guardados en el archivo " + cintaVCI);
}
```

The screenshot shows the continuation of the Java code editor. It includes a try-catch block for handling `IOException`, calculates the execution time in milliseconds, and prints the result.

```
464 }
465 catch (IOException e)
466 {
467     e.printStackTrace();
468 }
469
470 long tiempo_final = System.currentTimeMillis() - tiempo_inicial;
471 double tiempo_en_segundos = tiempo_final / 1000.0; // Convertir a segundos
472 System.out.println("El tiempo total fue de " + tiempo_en_segundos + " segundos");
473 }
```

Método verificarExistencia

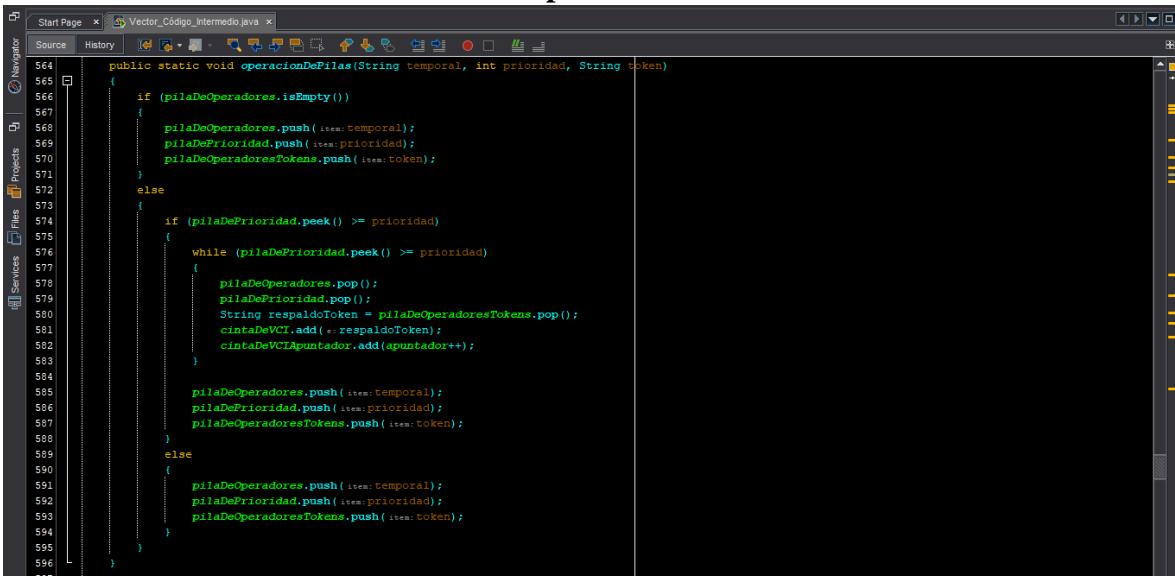
```
Start Page x Vector_Código_Intermedio.java x
Source History Navigator Projects Files Services
475 public static void verificarExistencia(String temporal, String token)
476 {
477     String [] prioridad_60 = new String [] {"*", "/", "%"};
478     String [] prioridad_50 = new String [] {"+", "-"};
479     String [] prioridad_40 = new String [] {"<", ">", "<=", ">=", "==", "!="};
480     String [] prioridad_30 = new String [] {"not"};
481     String [] prioridad_20 = new String [] {"and"};
482     String [] prioridad_10 = new String [] {"or"};
483     String [] prioridad_0 = new String [] {"="};
484
485     boolean encontrado = false;
486
487     for (String elemento : prioridad_0)
488     {
489         if (temporal.equals( anObject:elemento))
490         {
491             operacionDePilas(temporal, prioridad:0, token);
492             encontrado = true;
493             break;
494         }
495     }
496
497     for (String elemento : prioridad_10)
498     {
499         if (temporal.equals( anObject:elemento))
500         {
501             operacionDePilas(temporal, prioridad:10, token);
502             encontrado = true;
503             break;
504         }
505     }
506
507     for (String elemento : prioridad_20)
508     {
509         if (temporal.equals( anObject:elemento))
510         {
511             operacionDePilas(temporal, prioridad:20, token);
```

```
512             encontrado = true;
513             break;
514         }
515     }
516
517     for (String elemento : prioridad_30)
518     {
519         if (temporal.equals( anObject:elemento))
520         {
521             operacionDePilas(temporal, prioridad:30, token);
522             encontrado = true;
523             break;
524         }
525     }
526
527     for (String elemento : prioridad_40)
528     {
529         if (temporal.equals( anObject:elemento))
530         {
531             operacionDePilas(temporal, prioridad:40, token);
532             encontrado = true;
533             break;
534         }
535     }
536
537     for (String elemento : prioridad_50)
538     {
539         if (temporal.equals( anObject:elemento))
540         {
541             operacionDePilas(temporal, prioridad:50, token);
542             encontrado = true;
543             break;
544         }
545     }
546     for (String elemento : prioridad_60)
```

The screenshot shows a Java code editor interface with a dark theme. The top menu bar includes 'Start Page', 'Vector_Código_Intermedio.java', 'Source', 'History', and various tool icons. The left sidebar features 'Navigator' (with a circular icon), 'Projects' (with a folder icon), 'Files' (with a document icon), and 'Services' (with a gear icon). The main code area displays the following Java code:

```
547     for (String elemento : prioridad_60)
548     {
549         if (temporal.equals(anObjecto:elemento))
550         {
551             operacionDePilas(temporal, prioridad:60, token);
552             encontrado = true;
553             break;
554         }
555     }
556
557     if (encontrado == false)
558     {
559         cintaDeVCI.add(token);
560         cintaDeVCIDepuntador.add(spuntador++);
561     }
562 }
```

Método operacionDePilas



The screenshot shows a Java code editor with the file 'Vector_Código_Intermedio.java' open. The code implements a method named 'operacionDePilas' which processes tokens based on their temporal value and priority. It uses three stacks: 'pilaDeOperadores', 'pilaDePrioridad', and 'pilaDeOperadoresTokens'. The logic involves pushing items onto these stacks or popping them off if they are greater than or equal to the current item's temporal value. A variable 'cintaDeVCI' is used to store tokens from the stack.

```
564 public static void operacionDePilas(String temporal, int prioridad, String token)
565 {
566     if (pilaDeOperadores.isEmpty())
567     {
568         pilaDeOperadores.push(itemTemporal);
569         pilaDePrioridad.push(itemPrioridad);
570         pilaDeOperadoresTokens.push(itemToken);
571     }
572     else
573     {
574         if (pilaDePrioridad.peek() >= prioridad)
575         {
576             while (pilaDePrioridad.peek() >= prioridad)
577             {
578                 pilaDeOperadores.pop();
579                 pilaDePrioridad.pop();
580                 String respaldoToken = pilaDeOperadoresTokens.pop();
581                 cintaDeVCI.add(item: respaldoToken);
582                 cintaDeVCIApuntador.add(apuntador++);
583             }
584
585             pilaDeOperadores.push(itemTemporal);
586             pilaDePrioridad.push(itemPrioridad);
587             pilaDeOperadoresTokens.push(itemToken);
588         }
589         else
590         {
591             pilaDeOperadores.push(itemTemporal);
592             pilaDePrioridad.push(itemPrioridad);
593             pilaDeOperadoresTokens.push(itemToken);
594         }
595     }
596 }
```

Método guardarArrayListsEnArchivo

```
Start Page x Vector_Código_Intermedio.java x
Source History [ ] Projects Files Services
598 public static void guardarArrayListsEnArchivo(ArrayList<String> arrayList1, ArrayList<Integer> arrayList2, String rutaArchivo)
599 {
600     File archivo = new File(rutaArchivo);
601
602     try (BufferedWriter writer = new BufferedWriter(new FileWriter(archivo)))
603     {
604         // Verificamos si el archivo no existe y lo creamos
605         if (!archivo.exists())
606         {
607             archivo.createNewFile();
608         }
609
610         // Escribimos el primer ArrayList
611         int maxLength = 0;
612
613         for (String elemento : arrayList1)
614         {
615             maxLength = Math.max(maxLength, elemento.length());
616         }
617
618         for (String elemento : arrayList1)
619         {
620             writer.write(String.format("%-" + maxLength + "s", elemento));
621             writer.newLine();
622         }
623
624         writer.newLine(); // Agregamos un salto de linea después del primer ArrayList
625
626         // Escribimos el segundo ArrayList
627         for (int i = 0; i < arrayList2.size(); i++)
628         {
629             writer.write(String.format("%-" + maxLength + "s", arrayList2.get(i)));
630
631             if (i < arrayList2.size() - 1)
632             {
633                 writer.newLine();
634             }
635         }
636
637         // No es necesario agregar un salto de linea después del segundo ArrayList porque este será el final del archivo
638         System.out.println("ArrayLists guardados en el archivo correctamente.");
639     }
640     catch (IOException e)
641     {
642         System.err.println("Error al escribir en el archivo: " + e.getMessage());
643     }
644 }
```

```
634 }
635 }
636
637 // No es necesario agregar un salto de linea después del segundo ArrayList porque este será el final del archivo
638 System.out.println("ArrayLists guardados en el archivo correctamente.");
639 }
640 catch (IOException e)
641 {
642     System.err.println("Error al escribir en el archivo: " + e.getMessage());
643 }
644 }
```

Método leerArchivoYMostrarContenido

The screenshot shows a Java code editor window titled "Vector_Código_Intermedio.java". The code is a static method named "leerArchivoYMostrarContenido" that takes a string parameter "rutaArchivo". The method uses Java NIO to read all lines from the specified file and prints them to the console. It includes error handling for IOException.

```
646  public static void leerArchivoYMostrarContenido(String rutaArchivo)
647  {
648      try
649      {
650          java.nio.file.Path path = java.nio.file.Paths.get(first:rutaArchivo);
651          java.util.List<String> contenido = java.nio.file.Files.readAllLines(path);
652          System.out.println("Contenido del archivo:");
653
654          for (String linea : contenido)
655          {
656              System.out.println(linea);
657          }
658      }
659      catch (IOException e)
660      {
661          System.err.println("Error al leer el archivo: " + e.getMessage());
662      }
663 }
```

Método leerArchivoYMostrarContenido2

The screenshot shows a Java code editor window with the following code:

```
665 public static void leerArchivoYMostrarContenido2(String rutaArchivo)
666 {
667     try
668     {
669         Path path = Paths.get( first, rutaArchivo );
670         List<String> contenido = Files.readAllLines( path );
671
672         if ( !contenido.isEmpty() )
673         {
674             System.out.println( "Contenido del archivo:" );
675             System.out.println( contenido.get( index, 0 ) ); // Imprime solo el primer renglón
676         }
677         else
678         {
679             System.out.println( "El archivo está vacío." );
680         }
681     }
682     catch ( IOException e )
683     {
684         System.err.println( "Error al leer el archivo: " + e.getMessage() );
685     }
686 }
```

The code implements a method named `leerArchivoYMostrarContenido2` that takes a file path as a parameter. It uses the `Paths` class to get the path and the `Files.readAllLines` method to read all lines from the file into a list. If the list is not empty, it prints the first line to the console. Otherwise, it prints a message indicating the file is empty. Any `IOException` is caught and an error message is printed to the standard error stream.

Método leerArchivoYMostrarContenido3

The screenshot shows a Java IDE interface with a code editor window. The title bar of the code editor says "Vector_Código_Intermedio.java". The code itself is a Java method named "leerArchivoYMostrarContenido3". The code uses the Java 7 try-with-resources feature to read all lines from a file specified by the parameter "rutaArchivo". It prints the content of the file to the console and then prints the first line, which contains numbers separated by commas. An exception handling block catches IOException and prints an error message to System.err.

```
688 public static void leerArchivoYMostrarContenido3(String rutaArchivo)
689 {
690     try
691     {
692         Path path = Paths.get(rutaArchivo);
693         List<String> lineas = Files.readAllLines(path);
694         System.out.println("Contenido del archivo:");
695
696         // Imprimir el primer renglón
697         imprimirRenglónConNúmeros(lineas.get(0));
698     }
699     catch (IOException e)
700     {
701         System.err.println("Error al leer el archivo: " + e.getMessage());
702     }
703 }
```

Método leerArchivoYMostrarContenido4

The screenshot shows a Java code editor window with the following details:

- Title Bar:** Start Page x Vector_Código_Intermedio.java
- Toolbar:** Includes icons for Source, History, and various file operations.
- Code Area:** Displays the following Java code:

```
705 public static void leerArchivoYMostrarContenido4(String rutaArchivo)
706 {
707     try
708     {
709         Path path = Paths.get( first:rutaArchivo );
710         List<String> lineas = Files.readAllLines(path);
711         String primerRenglón = lineas.get(index:0);
712         String[] elementos = primerRenglón.split(regex:" \t ");
713         System.out.println(: "Contenido del archivo:");
714
715         for (String elemento : elementos)
716         {
717             System.out.println(: elemento);
718         }
719     }
720     catch (IOException e)
721     {
722         System.err.println("Error al leer el archivo: " + e.getMessage());
723     }
724 }
```

The code implements a method named `leerArchivoYMostrarContenido4` that takes a file path as a parameter. It reads all lines from the file, splits the first line into an array of elements using whitespace as a delimiter, and then prints each element to the console. If an `IOException` occurs during reading, it is caught and an error message is printed to standard error.

Método imprimirRenglonConNumeros

The screenshot shows a Java code editor window with the title bar "Método imprimirRenglonConNumeros". The main pane displays the following Java code:

```
726  Public static void imprimirRenglonConNumeros(String linea)
727  {
728      String[] elementos = linea.split(" ");
729
730      for (int i = 0; i < elementos.length; i++)
731      {
732          System.out.println(String.format("%-3d %-20s", args[i], elementos[i]));
733      }
734 }
```

The code uses a `String` array to split the input string `linea` by spaces. It then iterates through the array, printing each element followed by a fixed-width string using `String.format`. The `args` variable is used in the format string, which is likely a placeholder for the actual arguments passed to the method.

Método obtenerPrimerElementoSplit

The screenshot shows a Java code editor window with the following code:

```
736 public static String obtenerPrimerElementoSplit(String cadena)
737 {
738     if (cadena != null)
739     {
740         String[] partes = cadena.split("...","");
741         return partes.length > 0 ? partes[0] : "No se encontraron elementos después de dividir la cadena.";
742     }
743     else
744     {
745         return "La cadena es nula.";
746     }
747 }
```

The code is part of a static method named `obtenerPrimerElementoSplit` that takes a `String` parameter named `cadena`. It first checks if `cadena` is not null. If it's not null, it splits the string at the specified separator (represented by three dots) and returns the first element (`partes[0]`). If there are no elements after splitting, it returns a message indicating that. If `cadena` is null, it returns a message stating that the string is null.

Método formatoArchivo

The screenshot shows a Java code editor interface with the following details:

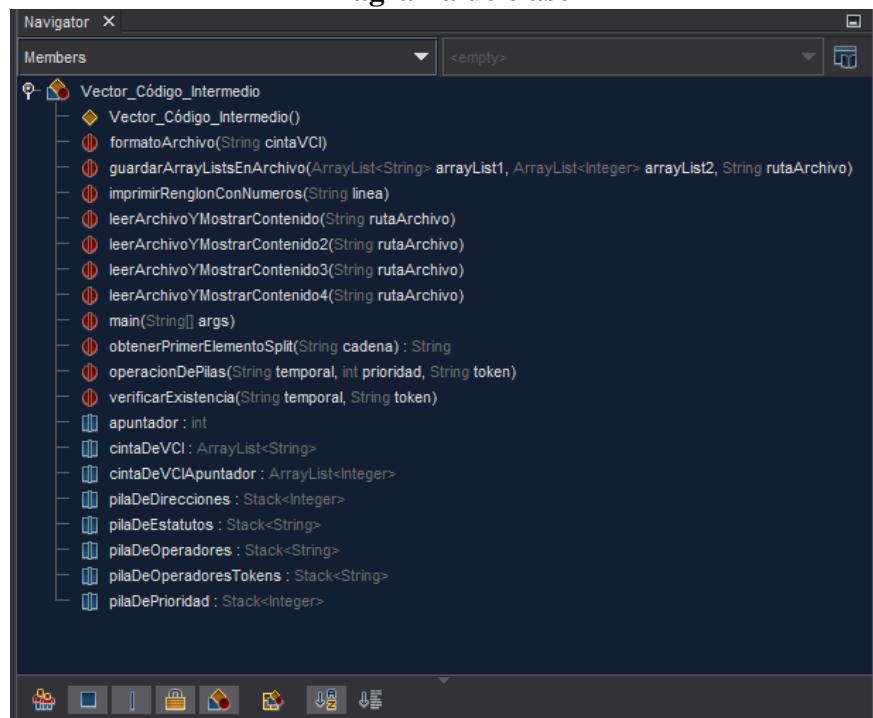
- Title Bar:** Shows "Start Page" and "Vector_Código_Intermedio.java".
- Toolbar:** Includes standard icons for file operations like Open, Save, Cut, Copy, Paste, Find, and Run.
- Left Sidebar:** Displays project navigation with sections for Navigator, Projects, Services, and Files.
- Code Area:** Displays the Java code for the `formatoArchivo` method. The code uses BufferedReader and BufferedWriter to read and write files, respectively. It splits the first line into elements and concatenates them with newlines. The code is annotated with numerous comments explaining its purpose.
- Status Bar:** Shows "645.5" and "INS Windows (CRLF)".

```
749 public static void formatoArchivo(String cintaVCI) throws IOException
750 {
751     // Abrir el archivo para lectura
752     BufferedReader reader = new BufferedReader(new FileReader(string:cintaVCI));
753
754     // Leer solo la primera linea
755     String line = reader.readLine();
756
757     // Crear un StringBuilder para almacenar el contenido
758     StringBuilder outputContent = new StringBuilder();
759
760     // Verificar si la primera linea es null
761     if (line != null)
762     {
763         // Dividir la linea en elementos
764         String[] elements = line.split(regex:" | ");
765
766         // Concatenar los elementos con saltos de linea
767         for (String element : elements)
768         {
769             outputContent.append(element).append("\n");
770         }
771     }
772
773     // Cerrar el archivo de entrada
774     reader.close();
775
776     // Abrir el archivo para escritura
777     FileWriter writer = new FileWriter(string:cintaVCI);
778
779     // Escribir el contenido procesado en el archivo
780     writer.write(outputContent.toString());
781
782     // Cerrar el archivo de salida
783     writer.close();
784 }
785
```

Tabla de token utilizada

1	programa,-1,-1,1
2	UNO@,-55,-2,1
3	;-,-75,-1,1
4	variables,-15,-1,2
5	entero,-11,-1,3
6	a&,-51,0,3
7	,,-76,-1,3
8	b&,-51,1,3
9	;-,-75,-1,3
10	real,-12,-1,4
11	x%,,-52,2,4
12	,,-76,-1,4
13	y%,,-52,3,4
14	;-,-75,-1,4
15	cadena,-13,-1,5
16	z\$,-53,4,5
17	;-,-75,-1,5
18	logico,-14,-1,6
19	dos#,,-54,5,6
20	;-,-75,-1,6
21	inicio,-2,-1,7
22	a&,-51,0,8
23	=,-26,-1,8
24	300,-61,-1,8
25	;-,-75,-1,8
26	b&,-51,1,9
27	=,-26,-1,9
28	0,-61,-1,9
29	;-,-75,-1,9
30	z\$,-53,4,10
31	=,-26,-1,10
32	"Hola",-63,-1,10
33	;-,-75,-1,10
34	x%,,-52,2,11
35	=,-26,-1,11
36	45.5,-62,-1,11
37	;-,-75,-1,11
38	y%,,-52,3,12
39	=,-26,-1,12
40	6.6,-62,-1,12
41	;-,-75,-1,12
42	dos#,,-54,5,13
43	=,-26,-1,13
44	true,-64,-1,13
45	;-,-75,-1,13
46	Fin,-3,-1,14
47	

Diagrama de clase



Ejecución

Opción 1

```
Output - Proyecto (run) x
run:
ArrayLists guardados en el archivo correctamente.
Menú de Opciones:
1. Cinta VCI con elementos y apuntadores (Horizontal)
2. Cinta VCI solo con elementos (Horizontal)
3. Cinta VCI con elementos y apuntadores (Vertical)
4. Cinta VCI solo con elementos (Vertical)
Ingrese el número de la opción que desea (1-4): 1
Contenido del archivo:
az,-51,0,8      $ 300,-61,-1,8      $ =,-26,-1,8      $ bs,-51,1,9      $ 0,-61,-1,9      $ =,-26,-1,9      $ z6,-53,4,10      $ "Hola",-63,-1,10 $ =,-26,-1,10      $ xt,-52,2,1
0      $ i      $ 2      $ 3      $ 4      $ 5      $ 6      $ 7      $ 8      $ 9
Datos procesados y guardados en el archivo C:\\\\Users\\\\rogeI\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Autómatas II\\\\Proyecto\\\\src\\\\Recursos\\\\Cinta de VCI.txt
El tiempo total fue de 2.528 segundos
BUILD SUCCESSFUL (total time: 2 seconds)
```

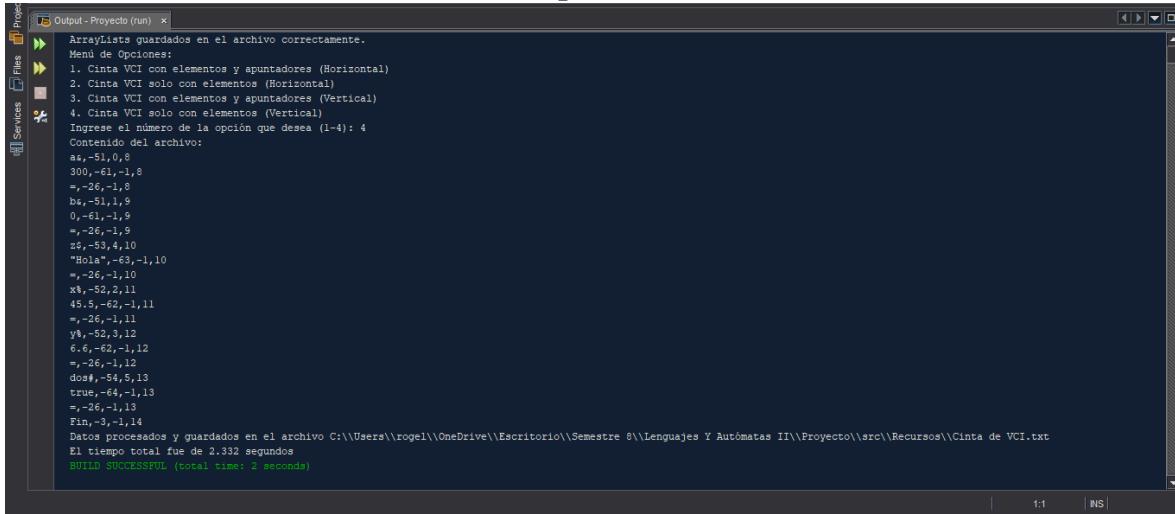
Opción 2

```
Output - Proyecto (run) x
run:
ArrayLists guardados en el archivo correctamente.
Menú de Opciones:
1. Cinta VCI con elementos y apuntadores (Horizontal)
2. Cinta VCI solo con elementos (Horizontal)
3. Cinta VCI con elementos y apuntadores (Vertical)
4. Cinta VCI solo con elementos (Vertical)
Ingrese el número de la opción que desea (1-4): 2
Contenido del archivo:
az,-51,0,8      $ 300,-61,-1,8      $ =,-26,-1,8      $ bs,-51,1,9      $ 0,-61,-1,9      $ =,-26,-1,9      $ z6,-53,4,10      $ "Hola",-63,-1,10 $ =,-26,-1,10      $ xt,-52,2,1
0      $ i      $ 2      $ 3      $ 4      $ 5      $ 6      $ 7      $ 8      $ 9
Datos procesados y guardados en el archivo C:\\\\Users\\\\rogeI\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Autómatas II\\\\Proyecto\\\\src\\\\Recursos\\\\Cinta de VCI.txt
El tiempo total fue de 2.893 segundos
BUILD SUCCESSFUL (total time: 3 seconds)
```

Opción 3

```
Output - Proyecto (run) x
run:
ArrayLists guardados en el archivo correctamente.
Menú de Opciones:
1. Cinta VCI con elementos y apuntadores (Horizontal)
2. Cinta VCI solo con elementos (Horizontal)
3. Cinta VCI con elementos y apuntadores (Vertical)
4. Cinta VCI solo con elementos (Vertical)
Ingrese el número de la opción que desea (1-4): 3
Contenido del archivo:
0  az,-51,0,8
1  300,-61,-1,8
2  =,-26,-1,8
3  bt,-51,1,9
4  0,-61,-1,9
5  =,-26,-1,9
6  zt,-53,4,10
7  "Hola",-63,-1,10
8  =,-26,-1,10
9  xt,-52,2,11
10 45,5,-62,-1,11
11 =,-26,-1,11
12 yt,-52,3,12
13 6,6,-62,-1,12
14 =,-26,-1,12
15 dos,-54,5,13
16 true,-64,-1,13
17 =,-26,-1,13
18 Fin,-3,-1,14
Datos procesados y guardados en el archivo C:\\\\Users\\\\rogeI\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Autómatas II\\\\Proyecto\\\\src\\\\Recursos\\\\Cinta de VCI.txt
El tiempo total fue de 2.451 segundos
BUILD SUCCESSFUL (total time: 2 seconds)
```

Opción 4



The screenshot shows the 'Output - Proyecto (run)' window in a Java IDE. The window displays the following text:

```
ArrayLists guardados en el archivo correctamente.
Menú de Opciones:
1. Cinta VCI con elementos y apuntadores (Horizontal)
2. Cinta VCI solo con elementos (Horizontal)
3. Cinta VCI con elementos y apuntadores (Vertical)
4. Cinta VCI solo con elementos (Vertical)
Ingrese el número de la opción que desea (1-4): 4
Contenido del archivo:
at,-51,0,9
300,-61,-1,8
=,-26,-1,8
bt,-51,1,9
0,-61,-1,9
=,-26,-1,9
zt,-53,4,10
"Hello",-63,-1,10
=,-26,-1,10
xt,-52,2,11
45,5,-62,-1,11
=,-26,-1,11
yt,-52,3,12
6,6,-62,-1,12
=,-26,-1,12
dos#, -54,5,13
true, -64,-1,13
=,-26,-1,13
Fin,-3,-1,14
Datos procesados y guardados en el archivo C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Autómatas II\\\\Proyecto\\\\src\\\\Recursos\\\\Cinta de VCI.txt
El tiempo total fue de 2.332 segundos
BUILD SUCCESSFUL (total time: 2 seconds)
```

Resultado

1	a&,-51,0,8
2	300,-61,-1,8
3	=,-26,-1,8
4	b&,-51,1,9
5	0,-61,-1,9
6	=,-26,-1,9
7	z\$,-53,4,10
8	"Hola",-63,-1,10
9	=,-26,-1,10
10	x%, -52, 2, 11
11	45.5,-62,-1,11
12	=,-26,-1,11
13	y%, -52, 3, 12
14	6.6,-62,-1,12
15	=,-26,-1,12
16	dos#, -54, 5, 13
17	true,-64,-1,13
18	=,-26,-1,13
19	Fin,-3,-1,14
20	

Versión 2

Método Main

The screenshot shows a Java code editor with the file `Vector_Código_Intermedio2.java` open. The code defines a class `Vector_Código_Intermedio2` with a static `main` method. The `main` method starts by getting the current time in milliseconds. It then reads lines from a file located at `C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Autómatas II\\\\Proyecto\\\\Vector_Código_Intermedio2.txt`. For each line, it splits it into two parts using the '-' character as a delimiter. If the first part is "inicio", it sets a flag `bandera` to true. Then, it processes the second part according to specific rules defined in a switch statement.

```
1 package Vector_Código_Intermedio;
2
3 import java.io.BufferedReader;
4 import java.io.BufferedWriter;
5 import java.io.FileReader;
6 import java.io.FileWriter;
7 import java.io.IOException;
8 import java.util.ArrayList;
9 import java.util.Stack;
10 import java.util.Scanner;
11 import java.nio.file.Files;
12 import java.nio.file.Path;
13 import java.nio.file.Paths;
14 import java.util.List;
15
16 /**
17  * 
18  * @author Rogelio Perez Guevara
19  * @author Equipo 2
20  * @version 12/04/2024
21  */
22
23 public class Vector_Código_Intermedio2
24 {
25     public static Stack<String> pilaDeOperadores = new Stack<>();
26     public static Stack<Integer> pilaDePrioridad = new Stack<>();
27     public static Stack<String> pilaDeOperadoresTokens = new Stack<>();
28
29     public static Stack<String> pilaDeEstatutos = new Stack<>();
30     public static Stack<Integer> pilaDeDirecciones = new Stack<>();
31
32     public static ArrayList<String> cintaDeVCI = new ArrayList<>();
33     Public static ArrayList<Integer> cintaDeVCIApuntador = new ArrayList<>();
34
35     public static int apuntador = 0;
36
37     public static void main (String [] args) throws IOException
38     {
39         long tiempo_inicial = System.currentTimeMillis();
40
41         // Leer el archivo de texto linea por linea
42         try (BufferedReader br = new BufferedReader(new FileReader("C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Autómatas II\\\\Proyecto\\\\Vector_Código_Intermedio2.txt")))
43         {
44             String linea;
45             String lineaSiguiente = null;
46             String guardado = null;
47             String temporal = null;
48             String token = null;
49             String hasta = null;
50             String simb = null;
51             ArrayList<String> inicio = new ArrayList<>();
52             ArrayList<String> mientras = new ArrayList<>();
53             int apuntador2;
54             String resultado = null;
55             boolean bandera = false;
56
57             while ((linea = br.readLine()) != null)
58             {
59                 // Dividir la linea en partes
60                 String[] partes = linea.split("\\-");
61                 String palabra2 = partes[0];
62                 String palabra = partes[1];
63
64                 // Se ejecutará solo la primera vez que se encuentre la palabra "inicio"
65                 if(!bandera && "-".equals(anObject:palabra))
66                 {
67                     bandera = true;
68                 }
69
70                 if(bandera == true)
71                 {
72                     // Reglas de análisis lexico
73                     switch (palabra)
```

This screenshot continues the code editor from the previous one, showing the completion of the `main` method. The code handles the analysis of tokens based on the state set by the "inicio" keyword. It uses a switch statement to process different types of tokens, such as operators, identifiers, or punctuation symbols. The code then updates the state and continues reading the input file until all lines have been processed.

```
1 package Vector_Código_Intermedio;
2
3 import java.io.BufferedReader;
4 import java.io.BufferedWriter;
5 import java.io.FileReader;
6 import java.io.FileWriter;
7 import java.io.IOException;
8 import java.util.ArrayList;
9 import java.util.Stack;
10 import java.util.Scanner;
11 import java.nio.file.Files;
12 import java.nio.file.Path;
13 import java.nio.file.Paths;
14 import java.util.List;
15
16 /**
17  * 
18  * @author Rogelio Perez Guevara
19  * @author Equipo 2
20  * @version 12/04/2024
21  */
22
23 public class Vector_Código_Intermedio2
24 {
25     public static Stack<String> pilaDeOperadores = new Stack<>();
26     public static Stack<Integer> pilaDePrioridad = new Stack<>();
27     public static Stack<String> pilaDeOperadoresTokens = new Stack<>();
28
29     public static Stack<String> pilaDeEstatutos = new Stack<>();
30     public static Stack<Integer> pilaDeDirecciones = new Stack<>();
31
32     public static ArrayList<String> cintaDeVCI = new ArrayList<>();
33     Public static ArrayList<Integer> cintaDeVCIApuntador = new ArrayList<>();
34
35     public static int apuntador = 0;
36
37     public static void main (String [] args) throws IOException
38     {
39         long tiempo_inicial = System.currentTimeMillis();
40
41         // Leer el archivo de texto linea por linea
42         try (BufferedReader br = new BufferedReader(new FileReader("C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Autómatas II\\\\Proyecto\\\\Vector_Código_Intermedio2.txt")))
43         {
44             String linea;
45             String lineaSiguiente = null;
46             String guardado = null;
47             String temporal = null;
48             String token = null;
49             String hasta = null;
50             String simb = null;
51             ArrayList<String> inicio = new ArrayList<>();
52             ArrayList<String> mientras = new ArrayList<>();
53             int apuntador2;
54             String resultado = null;
55             boolean bandera = false;
56
57             while ((linea = br.readLine()) != null)
58             {
59                 // Dividir la linea en partes
60                 String[] partes = linea.split("\\-");
61                 String palabra2 = partes[0];
62                 String palabra = partes[1];
63
64                 // Se ejecutará solo la primera vez que se encuentre la palabra "inicio"
65                 if(!bandera && "-".equals(anObject:palabra))
66                 {
67                     bandera = true;
68                 }
69
70                 if(bandera == true)
71                 {
72                     // Reglas de análisis lexico
73                     switch (palabra)
```

```
Start Page x Vector_Código_Intermedio2.java x
Source History Navigator Projects Files Services
72 // Reglas de análisis léxico
73 switch (palabra)
74 {
75     case "-6":
76         pilaDeEstatutos.push(item:lineas);
77         break;
78     case "-16":
79         guardado = linea;
80
81         while (!pilaDeOperadores.isEmpty())
82         {
83             pilaDeOperadores.pop();
84             pilaDePrioridad.pop();
85             cintaDeVCI.add(cintaDeOperadoresTokens.pop());
86             cintaDeVCIApuntador.add(apuntador++);
87         }
88
89         pilaDeDirecciones.push(item:apuntador);
90
91         cintaDeVCI.add(c: "L");
92         cintaDeVCIApuntador.add(c: apuntador);
93         apuntador++;
94
95         cintaDeVCI.add(c: guardado);
96         cintaDeVCIApuntador.add(c: apuntador++);
97         guardado = null;
98         break;
99     case "-7":
100         guardado = linea;
101         sino = "sino";
102
103         pilaDeEstatutos.push(item:lineas);
104
105         if (!pilaDeDirecciones.isEmpty())
106         {
107             apuntador2 = pilaDeDirecciones.pop();
```

```
107             apuntador2 = pilaDeDirecciones.pop();
108
109             for (int i = 0; i < cintaDeVCI.size(); i++)
110             {
111                 if (i == apuntador2)
112                 {
113                     cintaDeVCI.set(index:i, (apuntador + 2) + "");
114                     break;
115                 }
116             }
117
118
119             cintaDeVCI.add(c: "L");
120             pilaDeDirecciones.push(item:apuntador);
121             cintaDeVCIApuntador.add(apuntador++);
122
123             cintaDeVCI.add(c: guardado);
124             cintaDeVCIApuntador.add(apuntador++);
125             guardado = null;
126             break;
127         case "-3":
128             if (!pilaDeEstatutos.isEmpty())
129             {
130                 pilaDeEstatutos.pop();
131             }
132
133             String archivo = "C:\\Users\\rogel\\OneDrive\\Escriptorio\\Semestre 8\\Lenguajes Y Autómatas II\\Proyecto\\src\\Recursos\\Tabla";
134             String palabraBuscada = linea;
135
136             try (BufferedReader br2 = new BufferedReader(new FileReader(string:archivo)))
137             {
138                 String linea2;
139                 boolean palabraEncontrada = false;
140
141                 // Leemos el archivo linea por linea
142                 while ((linea2 = br2.readLine()) != null)
143                 {
```

The screenshot shows a Java IDE interface with the following details:

- Title Bar:** Start Page > Vector_Código_Intermedio2.java
- Toolbar:** Includes icons for Source, History, Find, Copy, Paste, and others.
- Left Sidebar:** Shows project navigation with sections like Projects, Files, Services, and Navigator.
- Code Editor:** Displays lines of Java code. The code is a simulation of a Turing Machine, involving a tape (cintaDeVCI), pointers (apuntador1, apuntador2), and stacks (pilaDeDirecciones). It handles operations like writing to the tape, moving pointers, and popping/dpushing from stacks.

```
175     if (i == apuntador2)
176     {
177         cintaDeVCI.set(index, (apuntador + 2) + "");
178         break;
179     }
180 }
181
182 if (!pilaDeDirecciones.isEmpty())
183 {
184     apuntador2 = pilaDeDirecciones.pop();
185     cintaDeVCI.add(apuntador) + "";
186     cintaDeVCI[apuntador].add(apuntador++);
187 }
188
189
190 cintaDeVCI.add("fin-mientras");
191 cintaDeVCI[apuntador].add(apuntador++);
192 mientras.remove("inicio");
193 mientras.remove("mientras");
194 break;
195 }
196 else if (!inicio.isEmpty() && "inicio".equals(anObject))
197 {
198     sino = null;
199
200     if (!pilaDeDirecciones.isEmpty())
201     {
202         int apu = pilaDeDirecciones.pop();
203
204         for (int i = 0; i < cintaDeVCI.size(); i++)
205         {
206             if (i == apu)
207             {
208                 if ("L".equals(anObject))
209                 {
210                     cintaDeVCI.set(i-2, apuntador + "");
211                 }
212             }
213         }
214     }
215 }
```

```
210         cintaDeVCI.set(i-2, apuntador + "");
211     }
212     else if ("! ".equals( anObject:cintaDeVCI.get(i-1)))
213     {
214         cintaDeVCI.set(i-1, apuntador + "");
215     }
216     else if ("! ".equals( anObject:cintaDeVCI.get( index:1)))
217     {
218         cintaDeVCI.set( index:1, apuntador + "");
219     }
220 }
221 }
222 mientras.remove( + "inicio");
223 break;
224 }
225 case "-21":
226     verificarExistencia( temporal:"*", token: linea);
227     break;
228 case "-22":
229     verificarExistencia( temporal: "/", token: linea);
230     break;
231 case "-23":
232     verificarExistencia( temporal: "%", token: linea);
233     break;
234 case "-24":
235     verificarExistencia( temporal: "+", token: linea);
236     break;
237 case "-25":
238     verificarExistencia( temporal: "-", token: linea);
239     break;
240 case "-31":
241     verificarExistencia( temporal: "<", token: linea);
242     break;
243 case "-33":
244     verificarExistencia( temporal: ">", token: linea);
```

```
245     case "-33":
246         verificarExistencia( temporal: ">", token: linea);
247         break;
248     case "-32":
249         verificarExistencia( temporal: "<=", token: linea);
250         break;
251     case "-34":
252         verificarExistencia( temporal: ">=", token: linea);
253         break;
254     case "-35":
255         verificarExistencia( temporal: "==" , token: linea);
256         break;
257     case "-36":
258         verificarExistencia( temporal: "!=" , token: linea);
259         break;
260     case "-43":
261         verificarExistencia( temporal: "not" , token: linea);
262         break;
263     case "-41":
264         verificarExistencia( temporal: "and" , token: linea);
265         break;
266     case "-42":
267         verificarExistencia( temporal: "or" , token: linea);
268         break;
269     case "-26":
270         verificarExistencia( temporal: "==" , token: linea);
271         break;
272     case "-9":
273         pilaDeEstatutos.push( item: linea);
274         pilaDeDirecciones.push( item: apuntador);
275         break;
276     case "-10":
277         hasta = palabra2;
278         temporal = linea;
279         mientras.remove( + "inicio");
280         break;
281     case "-73":
```

```
Start Page × Vector_Código_Intermedio2.java ×
Source History ▾
281
282     case "-73":
283         pilaDeOperadores.push(item:palabra2);
284         pilaDePrioridad.push(item:0);
285         pilaDeOperadoresTokens.push(item:linea);
286         break;
287     case "-74":
288         while(!pilaDeOperadores.isEmpty() && pilaDeOperadores.peek().equals("n"))
289         {
290             String t = pilaDeOperadores.pop();
291             pilaDePrioridad.pop();
292             token = pilaDeOperadoresTokens.pop();
293
294             if (!".equals(anObject:t))
295             {
296                 cintaDeVCI.add(:token);
297                 cintaDeVCIApuntador.add(apuntador++);
298             }
299             token = null;
300
301             if (!pilaDeOperadores.isEmpty())
302             {
303                 pilaDeOperadores.pop();
304                 pilaDePrioridad.pop();
305                 token = pilaDeOperadoresTokens.pop();
306                 cintaDeVCI.add(:tokens);
307                 cintaDeVCIApuntador.add(apuntador++);
308             }
309
310             if (!pilaDeOperadores.isEmpty())
311             {
312                 //Elimina el ( de la pila
313                 pilaDeOperadores.pop();
314                 pilaDePrioridad.pop();
315                 pilaDeOperadoresTokens.pop();
316             }
317         }
```

```
Start Page × Vector_Código_Intermedio2.java ×
Source History ▾
317
318     if("hasta".equals(anObject:hasta))
319     {
320         if (!pilaDeDirecciones.isEmpty())
321         {
322             guardado = (pilaDeDirecciones.pop()) + "";
323             cintaDeVCI.add(:guardado);
324             cintaDeVCIApuntador.add(apuntador++);
325         }
326
327         hasta = null;
328         guardado = null;
329
330         if(temporal != null)
331         {
332             cintaDeVCI.add(:temporal);
333             cintaDeVCIApuntador.add(apuntador++);
334             temporal = null;
335         }
336     }
337     break;
338     case "-2":
339         inicio.add(:"inicio");
340         break;
341     case "-8":
342         pilaDeEstatutos.push(item:linea);
343         pilaDeDirecciones.push(item:apuntador);
344         mientras.add(item:"mientras");
345         break;
346     case "-17":
347         while (!pilaDeOperadores.isEmpty())
348         {
349             pilaDeOperadores.pop();
350             pilaDePrioridad.pop();
351             guardado = pilaDeOperadoresTokens.pop();
352             cintaDeVCI.add(:guardado);
353             cintaDeVCIApuntador.add(apuntador++);
```

```
353     cintaDeVCI.apuntador.add(apuntador++);
354     guardado = null;
355
356     cintaDeVCI.add(": " + linea);
357     pilaDeDirecciones.push(item: apuntador);
358     cintaDeVCI.apuntador.add(apuntador++);
359
360     cintaDeVCI.add(": linea");
361     cintaDeVCI.apuntador.add(apuntador++);
362     break;
363   case "-75":
364     while (!pilaDeOperadores.isEmpty())
365     {
366       pilaDeOperadores.pop();
367       pilaDePrioridad.pop();
368       token = pilaDeOperadores.tokens.pop();
369       cintaDeVCI.add(": token");
370       cintaDeVCI.apuntador.add(apuntador++);
371       token = null;
372     }
373     break;
374   default:
375     cintaDeVCI.add(": linea");
376     cintaDeVCI.apuntador.add(apuntador++);
377     break;
378   }
379
380 }
381
382 }
383
384 // Ruta del archivo donde se guardará el texto
385 String cintaVCI = "C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Autómatas II\\\\Proyecto\\\\src\\\\Recursos\\\\Cinta";
386
387 // Guardamos los ArrayLists en el archivo de texto
388 guardarArrayListsEnArchivo(arrayList: cintaDeVCI, arrayList2: cintaDeVCI.apuntador, rutaArchivo: cintaVCI);
389
```

```
389 // Leemos y mostramos el contenido del archivo de texto
390 ArrayList<String> primerLineas = new ArrayList<>();
391 ArrayList<String> segundaLinea = new ArrayList<>();
392
393 try (BufferedReader br = new BufferedReader(new FileReader(string:cintaVCI)))
394 {
395   String linea;
396
397   while ((linea = br.readLine()) != null)
398   {
399     String[] partes = linea.split(regex: " \t");
400     primerLineas.add(partes[0]);
401     segundaLinea.add(partes[1]);
402   }
403 }
404 catch (IOException e)
405 {
406   System.err.println("Error al leer el archivo: " + e.getMessage());
407   return;
408 }
409
410 // Menú de opciones
411 Scanner scanner = new Scanner(System.in);
412 int opcion = 0;
413
414 while (opcion < 1 || opcion > 4)
415 {
416   System.out.println("Menú de Opciones:");
417   System.out.println("1. Cinta VCI con elementos y apuntadores (Horizontal)");
418   System.out.println("2. Cinta VCI solo con elementos (Horizontal)");
419   System.out.println("3. Cinta VCI con elementos y apuntadores (Vertical)");
420   System.out.println("4. Cinta VCI solo con elementos (Vertical)");
421   System.out.print("Ingrese el numero de la opción que desea (1-4): ");
422
423   if (scanner.hasNextInt())
424   {
425     opcion = scanner.nextInt();
426   }
427 }
```

The screenshot shows a Java code editor window with the file "Vector_Código_Intermedio2.java" open. The code handles user input for a menu option (1-4) and prints an error message if it's invalid. It then prints the selected menu option and performs the corresponding action based on the option. The code uses a switch statement with four cases (1-4) and a default case.

```
424     if (scanner.hasNextInt())
425     {
426         opcion = scanner.nextInt();
427
428         if (opcion < 1 || opcion > 4)
429         {
430             System.out.println("Opción no válida. Por favor, ingrese un número entre 1 y 4.");
431         }
432         else
433         {
434             System.out.println("Entrada inválida. Por favor, ingrese un número.");
435             scanner.next(); // Limpiar el buffer del scanner
436         }
437     }
438
439     // Imprimir según la opción seleccionada
440     // Realizar la acción correspondiente a la opción seleccionada
441     switch (opcion)
442     {
443
444         case 1:
445             leerArchivoYMostrarContenido(rutarchivo:cintaVCI);
446             break;
447         case 2:
448             leerArchivoYMostrarContenido2(rutarchivo:cintaVCI);
449             break;
450         case 3:
451             leerArchivoYMostrarContenido3(rutarchivo:cintaVCI);
452             break;
453         case 4:
454             leerArchivoYMostrarContenido4(rutarchivo:cintaVCI);
455             break;
456         default:
457             System.out.println("Opción no válida.");
458     }
459
460     // Dar formato al archivo
```

The screenshot shows the continuation of the Java code editor window. It starts with a try block to process the file, followed by a catch block for IOException. The code then calculates the execution time in milliseconds and converts it to seconds, finally printing the total time taken.

```
460     // Dar formato al archivo
461     try
462     {
463         // Procesar el archivo
464         formatoArchivo(cintaVCI);
465         System.out.println("Datos procesados y guardados en el archivo " + cintaVCI);
466     }
467     catch (IOException e)
468     {
469         e.printStackTrace();
470     }
471
472     long tiempo_final = System.currentTimeMillis() - tiempo_inicial;
473     double tiempo_en_segundos = tiempo_final / 1000.0; // Convertir a segundos
474     System.out.println("El tiempo total fue de " + tiempo_en_segundos + " segundos");
475 }
```

Método verificarExistencia

The screenshot shows a Java code editor with the file 'Vector_Código_Intermedio2.java' open. The code implements a method named 'verificarExistencia' which checks if a given temporal string matches any of six priority levels (0, 10, 20, 40, 50) and performs an operation if found. The editor interface includes tabs for 'Start Page' and 'Vector_Código_Intermedio2.java', a toolbar with various icons, and a status bar at the bottom.

```
477 public static void verificarExistencia(String temporal, String token)
478 {
479     String [] prioridad_60 = new String [] {"*", "/", "%"};
480     String [] prioridad_50 = new String [] {"+", "-"};
481     String [] prioridad_40 = new String [] {"<", ">", "<=", ">=", "==", "!="};
482     String [] prioridad_30 = new String [] {"not"};
483     String [] prioridad_20 = new String [] {"and"};
484     String [] prioridad_10 = new String [] {"or"};
485     String [] prioridad_0 = new String [] {"="};
486
487     boolean encontrado = false;
488
489     for (String elemento : prioridad_0)
490     {
491         if (temporal.equals( anObject:elemento))
492         {
493             operacionDePilas(temporal, prioridad:0, token);
494             encontrado = true;
495             break;
496         }
497     }
498
499     for (String elemento : prioridad_10)
500     {
501         if (temporal.equals( anObject:elemento))
502         {
503             operacionDePilas(temporal, prioridad:10, token);
504             encontrado = true;
505             break;
506         }
507     }
508
509     for (String elemento : prioridad_20)
510     {
511         if (temporal.equals( anObject:elemento))
512         {
513             operacionDePilas(temporal, prioridad:20, token);
514             encontrado = true;
515             break;
516         }
517     }
518
519     for (String elemento : prioridad_30)
520     {
521         if (temporal.equals( anObject:elemento))
522         {
523             operacionDePilas(temporal, prioridad:30, token);
524             encontrado = true;
525             break;
526         }
527     }
528
529     for (String elemento : prioridad_40)
530     {
531         if (temporal.equals( anObject:elemento))
532         {
533             operacionDePilas(temporal, prioridad:40, token);
534             encontrado = true;
535             break;
536         }
537     }
538
539     for (String elemento : prioridad_50)
540     {
541         if (temporal.equals( anObject:elemento))
542         {
543             operacionDePilas(temporal, prioridad:50, token);
544             encontrado = true;
545             break;
546         }
547     }
548     for (String elemento : prioridad_60)
```

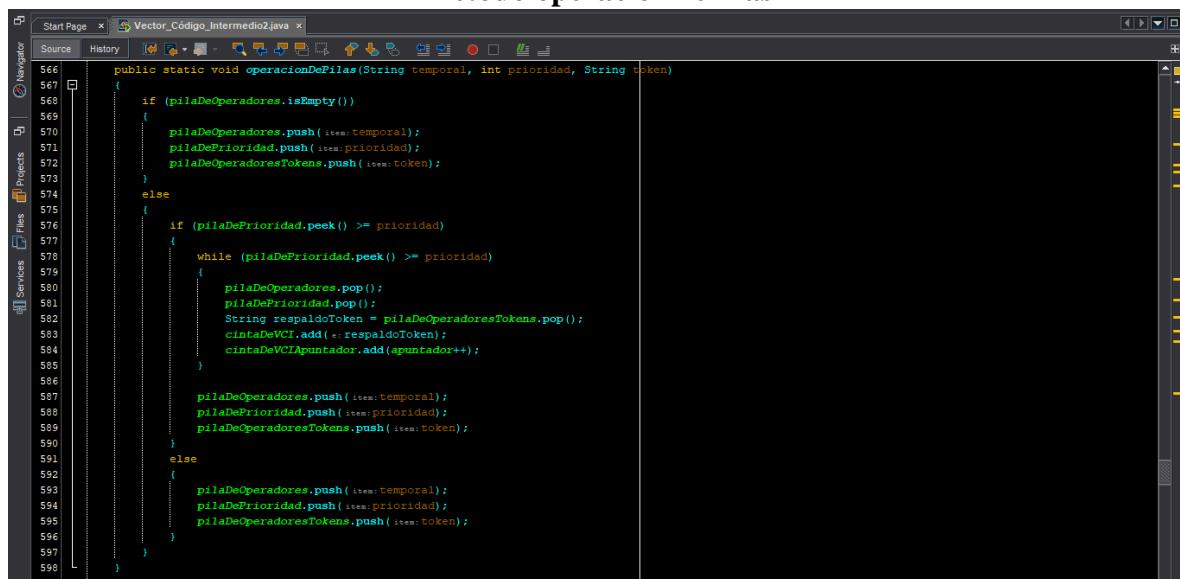
The screenshot shows the continuation of the 'verificarExistencia' method from the previous editor. It concludes the search for the temporal string across all priority levels and ends with a closing brace for the method. The editor interface remains consistent with the first screenshot.

```
549     for (String elemento : prioridad_60)
```

The screenshot shows a Java code editor window with the following code snippet:

```
Start Page x Vector_Código_Intermedio2.java x
Source History Navigator Projects Tasks Services
549     for (String elemento : prioridad_60)
550     {
551         if (temporal.equals( s0bject:elemento))
552         {
553             operacionDePilas(temporal, prioridad:60, token);
554             encontrado = true;
555             break;
556         }
557     }
558
559     if (encontrado == false)
560     {
561         cintaDeVCI.add(s:token);
562         cintaDeVCIApuntador.add(apuntador++);
563     }
564 }
```

Método operacionDePilas



The screenshot shows a Java code editor with the file 'Vector_Código_Intermedio2.java' open. The code implements a method named 'operacionDePilas' which performs stack operations based on item priority. The code uses three stacks: 'pilaDeOperadores', 'pilaDePrioridad', and 'pilaDeOperadoresTokens'. It checks if the operator stack is empty, then pushes the current item's temporal value, priority, and token onto their respective stacks. If the priority stack is not empty and its peeked value is greater than or equal to the current item's priority, it pops items from both stacks and adds them to a backup stack 'cintaDeVCI'. After this, it pushes the current item onto all three stacks. Otherwise, it simply pushes the current item onto all three stacks. The code is annotated with line numbers from 566 to 598.

```
566 public static void operacionDePilas(String temporal, int prioridad, String token)
567 {
568     if (pilaDeOperadores.isEmpty())
569     {
570         pilaDeOperadores.push(item.temporal);
571         pilaDePrioridad.push(item.prioridad);
572         pilaDeOperadoresTokens.push(item.token);
573     }
574     else
575     {
576         if (pilaDePrioridad.peek() >= prioridad)
577         {
578             while (pilaDePrioridad.peek() >= prioridad)
579             {
580                 pilaDeOperadores.pop();
581                 pilaDePrioridad.pop();
582                 String respaldoToken = pilaDeOperadoresTokens.pop();
583                 cintaDeVCI.add(cintaDeVCI.apuntador, respaldoToken);
584                 cintaDeVCI.apuntador++;
585             }
586
587             pilaDeOperadores.push(item.temporal);
588             pilaDePrioridad.push(item.prioridad);
589             pilaDeOperadoresTokens.push(item.token);
590         }
591         else
592         {
593             pilaDeOperadores.push(item.temporal);
594             pilaDePrioridad.push(item.prioridad);
595             pilaDeOperadoresTokens.push(item.token);
596         }
597     }
598 }
```

Método guardarArrayListsEnArchivo

```
Start Page x Vector_Código_Intermedio2.java x
Source History ⌂ Projects Files Services
600 public static void guardarArrayListsEnArchivo(ArrayList<String> arrayList1, ArrayList<Integer> arrayList2, String rutaArchivo)
601 {
602     File archivo = new File(rutaArchivo);
603
604     try (BufferedWriter writer = new BufferedWriter(new FileWriter(archivo)))
605     {
606         // Verificamos si el archivo no existe y lo creamos
607         if (!archivo.exists())
608         {
609             archivo.createNewFile();
610         }
611
612         // Escribimos el primer ArrayList
613         int maxLength = 0;
614
615         for (String elemento : arrayList1)
616         {
617             maxLength = Math.max(maxLength, elemento.length());
618         }
619
620         for (String elemento : arrayList1)
621         {
622             writer.write(String.format("%-" + maxLength + "s", elemento));
623             writer.write(" ");
624         }
625
626         writer.newLine(); // Agregamos un salto de linea después del primer ArrayList
627
628         // Escribimos el segundo ArrayList
629         for (int i = 0; i < arrayList2.size(); i++)
630         {
631             writer.write(String.format("%-" + maxLength + "s", arrayList2.get(i)));
632
633             if (i < arrayList2.size() - 1)
634             {
635                 writer.write(" ");
636             }
637         }
638
639         // No es necesario agregar un salto de linea después del segundo ArrayList porque este será el final del archivo
640         System.out.println("ArrayLists guardados en el archivo correctamente.");
641     } catch (IOException e)
642     {
643         System.err.println("Error al escribir en el archivo: " + e.getMessage());
644     }
645 }
```

```
Start Page x Vector_Código_Intermedio2.java x
Source History ⌂ Projects Files Services
635         writer.write(" ");
636     }
637 }
638
639 // No es necesario agregar un salto de linea después del segundo ArrayList porque este será el final del archivo
640 System.out.println("ArrayLists guardados en el archivo correctamente.");
641
642 catch (IOException e)
643 {
644     System.err.println("Error al escribir en el archivo: " + e.getMessage());
645 }
```

Método leerArchivoYMostrarContenido

The screenshot shows a Java code editor window with the title bar "Start Page" and "Vector_Código_Intermedio2.java". The code editor displays a method named "leerArchivoYMostrarContenido" with the following content:

```
647 public static void leerArchivoYMostrarContenido(String rutaArchivo)
648 {
649     try
650     {
651         java.nio.file.Path path = java.nio.file.Paths.get(rutaArchivo);
652         java.util.List<String> contenido = java.nio.file.Files.readAllLines(path);
653         System.out.println("Contenido del archivo:");
654
655         for (String linea : contenido)
656         {
657             System.out.println(linea);
658         }
659     }
660     catch (IOException e)
661     {
662         System.err.println("Error al leer el archivo: " + e.getMessage());
663     }
664 }
```

The code uses Java NIO to read all lines from a file specified by the parameter "rutaArchivo". It then prints each line to the standard output. An exception block handles any `IOException` that might occur during the file reading process.

Método leerArchivoYMostrarContenido2

The screenshot shows a Java code editor window titled "Vector_Código_Intermedio2.java". The code is a method named "leerArchivoYMostrarContenido2" that reads the contents of a file and prints them to the console. The code uses Java 7 syntax, including try-with-resources and the Files class from the java.nio package.

```
666  public static void leerArchivoYMostrarContenido2(String rutaArchivo)
667  {
668      try
669      {
670          Path path = Paths.get( rutaArchivo );
671          List<String> contenido = Files.readAllLines(path);
672
673          if (!contenido.isEmpty())
674          {
675              System.out.println( "Contenido del archivo:" );
676              System.out.println( contenido.get(index:0) ); // Imprime solo el primer renglón
677          }
678          else
679          {
680              System.out.println( "El archivo está vacío." );
681          }
682      }
683      catch (IOException e)
684      {
685          System.err.println("Error al leer el archivo: " + e.getMessage());
686      }
687  }
```

Método leerArchivoYMostrarContenido3

The screenshot shows a Java code editor window titled "Vector_Código_Intermedio2.java". The code is a static method named "leerArchivoYMostrarContenido3" that takes a string parameter "rutaArchivo". The method uses try-with-resources to read all lines from the specified file and prints them to the console. It also prints the first line with numbers. An exception handling block catches IOException and prints an error message.

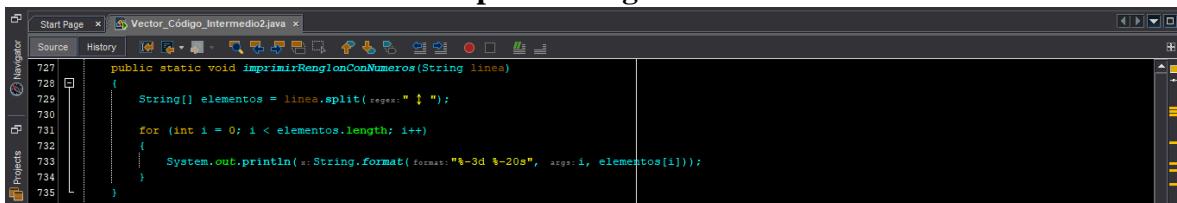
```
689 public static void leerArchivoYMostrarContenido3(String rutaArchivo)
690 {
691     try
692     {
693         Path path = Paths.get(rutaArchivo);
694         List<String> lineas = Files.readAllLines(path);
695         System.out.println("Contenido del archivo:");
696
697         // Imprimir el primer renglón
698         imprimirRenglónConNúmeros(lineas.get(0));
699     }
700     catch (IOException e)
701     {
702         System.err.println("Error al leer el archivo: " + e.getMessage());
703     }
704 }
```

Método leerArchivoYMostrarContenido4

The screenshot shows a Java code editor window titled "Vector_Código_Intermedio2.java". The code is a static method named "leerArchivoYMostrarContenido4" that takes a string parameter "rutaArchivo". The method uses try-catch blocks to handle file operations. It reads all lines from the specified file, splits the first line into an array of elements, prints the first element, and then iterates through the remaining elements, printing each one. An error message is printed if an IOException occurs.

```
706 public static void leerArchivoYMostrarContenido4(String rutaArchivo)
707 {
708     try
709     {
710         Path path = Paths.get( first:rutaArchivo );
711         List<String> lineas = Files.readAllLines( path );
712         String primerRenglón = lineas.get( index:0 );
713         String[] elementos = primerRenglón.split( regex:" \t" );
714         System.out.println( x: "Contenido del archivo:" );
715
716         for (String elemento : elementos)
717         {
718             System.out.println( x: elemento );
719         }
720     }
721     catch (IOException e)
722     {
723         System.err.println("Error al leer el archivo: " + e.getMessage());
724     }
725 }
```

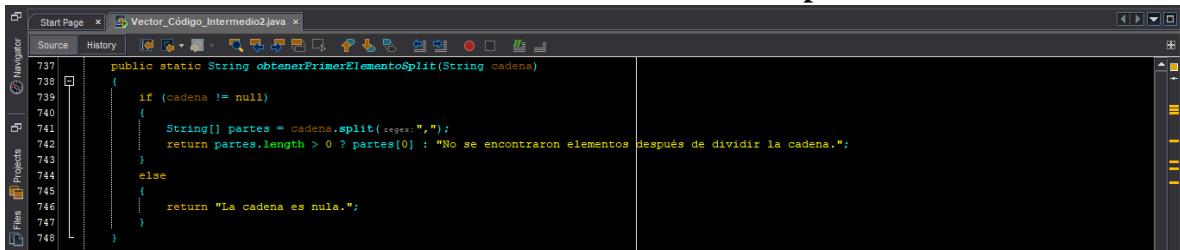
Método imprimirRenglonConNumeros



The screenshot shows a Java code editor window with the title "Método imprimirRenglonConNumeros". The code is contained within a class named "Vector_Código_Intermedio2.java". The method "imprimirRenglonConNumeros" takes a single parameter "linea" and prints its elements to the console. The code uses a for loop to iterate through the array of elements, printing each element with a width of 20 characters.

```
727 public static void imprimirRenglonConNumeros(String linea)
728 {
729     String[] elementos = linea.split(" ");
730
731     for (int i = 0; i < elementos.length; i++)
732     {
733         System.out.println(String.format("%-3d %-20s", args[i], elementos[i]));
734     }
735 }
```

Método obtenerPrimerElementoSplit



The screenshot shows a Java code editor window with the following code:

```
737
738     public static String obtenerPrimerElementoSplit(String cadena)
739     {
740         if (cadena != null)
741         {
742             String[] partes = cadena.split(",");
743             return partes.length > 0 ? partes[0] : "No se encontraron elementos después de dividir la cadena.";
744         }
745         else
746         {
747             return "La cadena es nula.";
748         }
749     }
```

The code defines a static method named `obtenerPrimerElementoSplit` that takes a string parameter named `cadena`. It first checks if `cadena` is not null. If it's not null, it splits the string into an array of strings using a comma as the delimiter. If the resulting array has more than one element, it returns the first element (`partes[0]`). Otherwise, it returns a message indicating that no elements were found after splitting the string. If `cadena` is null, it returns a message stating that the string is null.

Método formatoArchivo

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Start Page x Vector_Código_Intermedio2.java x
- Toolbar:** Includes icons for Source, History, and various file operations.
- Left Sidebar:** Shows Navigator, Projects, Services, and Files.
- Code Area:** Displays the following Java code:

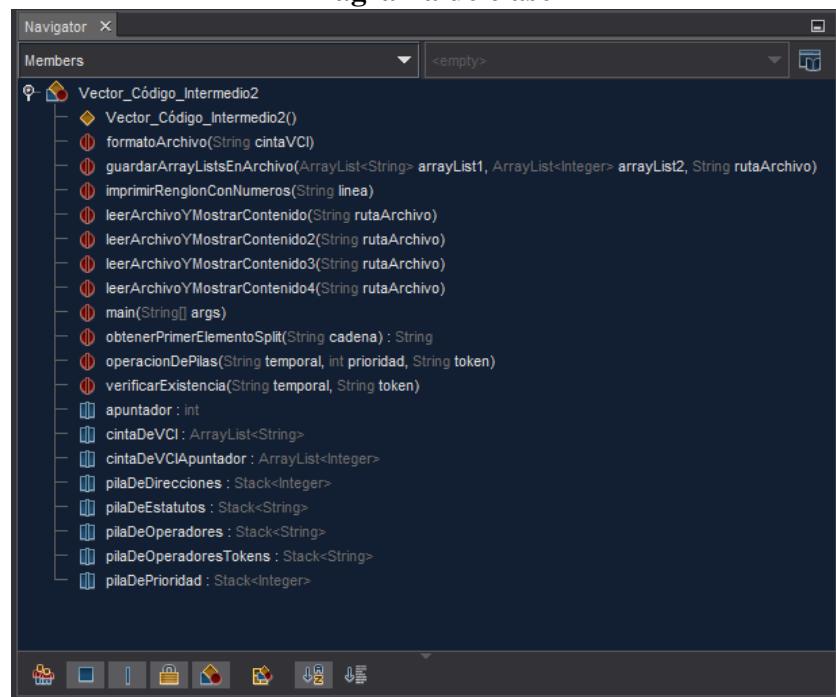
```
750 public static void formatoArchivo(String cintaVCI) throws IOException
751 {
752     // Abrir el archivo para lectura
753     BufferedReader reader = new BufferedReader(new FileReader(string:cintaVCI));
754
755     // Leer solo la primera linea
756     String line = reader.readLine();
757
758     // Crear un StringBuilder para almacenar el contenido
759     StringBuilder outputContent = new StringBuilder();
760
761     // Verificar si la primera linea es null
762     if (line != null)
763     {
764         // Dividir la linea en elementos
765         String[] elements = line.split(regex:" | ");
766
767         // Concatenar los elementos con saltos de linea
768         for (String element : elements)
769         {
770             outputContent.append(element).append("\n");
771         }
772     }
773
774     // Cerrar el archivo de entrada
775     reader.close();
776
777     // Abrir el archivo para escritura
778     FileWriter writer = new FileWriter(string:cintaVCI);
779
780     // Escribir el contenido procesado en el archivo
781     writer.write(outputContent.toString());
782
783     // Cerrar el archivo de salida
784     writer.close();
785 }
786
```

The code implements a static method `formatoArchivo` that takes a string `cintaVCI` and processes it using `BufferedReader` and `FileWriter`. It reads the first line, splits it into elements, concatenates them with newlines, and then writes the processed content back to the file.

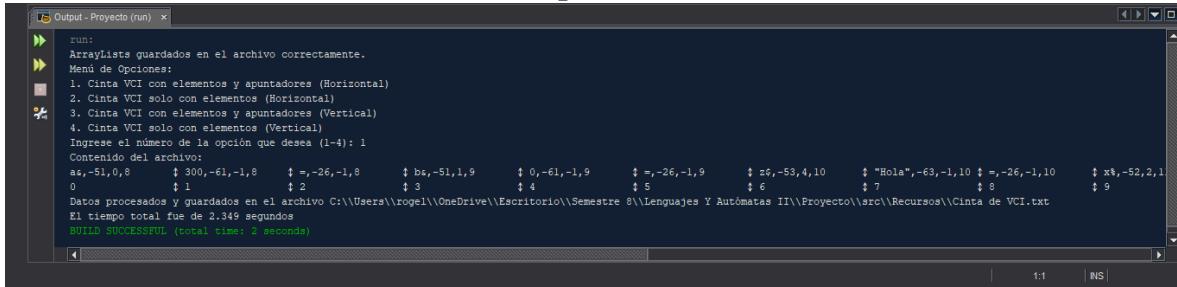
Tabla de token utilizada

1	programa,-1,-1,1
2	UNO@,-55,-2,1
3	;-,-75,-1,1
4	variables,-15,-1,2
5	entero,-11,-1,3
6	a&,-51,0,3
7	,,-76,-1,3
8	b&,-51,1,3
9	,,-75,-1,3
10	real,-12,-1,4
11	x%,,-52,2,4
12	,,-76,-1,4
13	y%,,-52,3,4
14	,,-75,-1,4
15	cadena,-13,-1,5
16	z\$,-53,4,5
17	,,-75,-1,5
18	logico,-14,-1,6
19	dos#,,-54,5,6
20	,,-75,-1,6
21	inicio,-2,-1,7
22	a&,-51,0,8
23	=,-26,-1,8
24	300,-61,-1,8
25	,,-75,-1,8
26	b&,-51,1,9
27	=,-26,-1,9
28	0,-61,-1,9
29	,,-75,-1,9
30	z\$,-53,4,10
31	=,-26,-1,10
32	"Hola",-63,-1,10
33	,,-75,-1,10
34	x%,,-52,2,11
35	=,-26,-1,11
36	45.5,-62,-1,11
37	,,-75,-1,11
38	y%,,-52,3,12
39	=,-26,-1,12
40	6.6,-62,-1,12
41	,,-75,-1,12
42	dos#,,-54,5,13
43	=,-26,-1,13
44	true,-64,-1,13
45	,,-75,-1,13
46	Fin,-3,-1,14
47	

Diagrama de clase

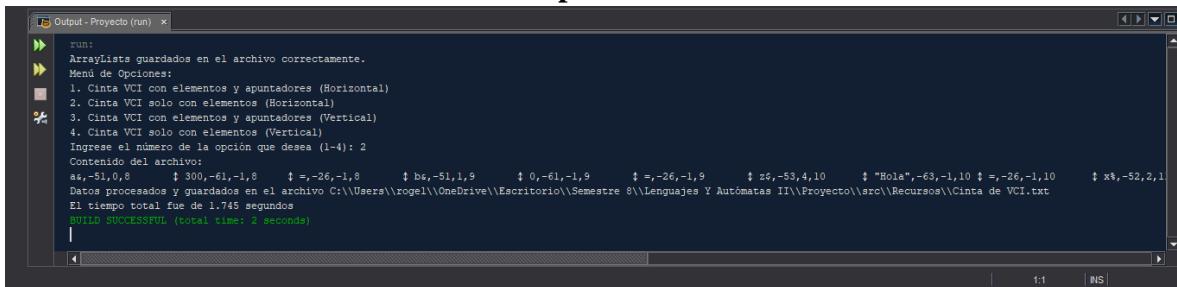


Ejecución Opción 1



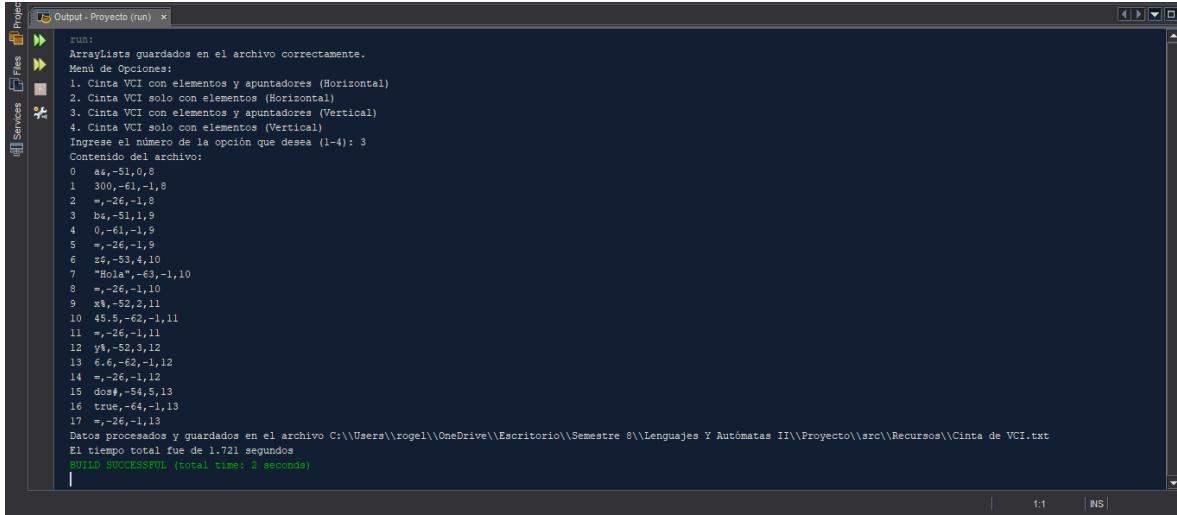
```
Output - Proyecto (run) x
▶ run:
ArrayLists guardados en el archivo correctamente.
Menú de Opciones:
1. Cinta VCI con elementos y apuntadores (Horizontal)
2. Cinta VCI solo con elementos (Horizontal)
3. Cinta VCI con elementos y apuntadores (Vertical)
4. Cinta VCI solo con elementos (Vertical)
Ingrese el número de la opción que desea (1-4): 1
Contenido del archivo:
as,-51,0,8      $ 300,-61,-1,8      $ =,-26,-1,8      $ b4,-51,1,9      $ 0,-61,-1,9      $ =,-26,-1,9      $ z6,-53,4,10      $ "Hola",-63,-1,10 $ =,-26,-1,10      $ xb,-52,2,1
0      $ i      $ 2      $ 3      $ 4      $ 5      $ 6      $ 7      $ 8      $ 9
Datos procesados y guardados en el archivo C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Autómatas II\\\\Proyecto\\\\src\\\\Recursos\\\\Cinta de VCI.txt
El tiempo total fue de 2.349 segundos
BUILD SUCCESSFUL (total time: 2 seconds)
```

Opción 2



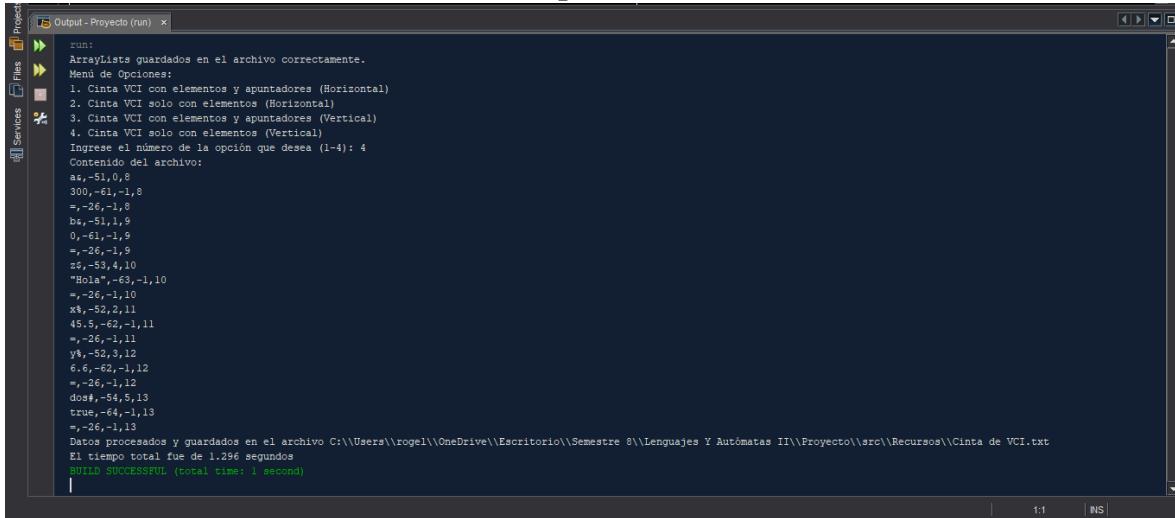
```
Output - Proyecto (run) x
▶ run:
ArrayLists guardados en el archivo correctamente.
Menú de Opciones:
1. Cinta VCI con elementos y apuntadores (Horizontal)
2. Cinta VCI solo con elementos (Horizontal)
3. Cinta VCI con elementos y apuntadores (Vertical)
4. Cinta VCI solo con elementos (Vertical)
Ingrese el número de la opción que desea (1-4): 2
Contenido del archivo:
as,-51,0,8      $ 300,-61,-1,8      $ =,-26,-1,8      $ b4,-51,1,9      $ 0,-61,-1,9      $ =,-26,-1,9      $ z6,-53,4,10      $ "Hola",-63,-1,10 $ =,-26,-1,10      $ xb,-52,2,1
Datos procesados y guardados en el archivo C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Autómatas II\\\\Proyecto\\\\src\\\\Recursos\\\\Cinta de VCI.txt
El tiempo total fue de 1.745 segundos
BUILD SUCCESSFUL (total time: 2 seconds)
```

Opción 3



```
Output - Proyecto (run) x
▶ run:
ArrayLists guardados en el archivo correctamente.
Menú de Opciones:
1. Cinta VCI con elementos y apuntadores (Horizontal)
2. Cinta VCI solo con elementos (Horizontal)
3. Cinta VCI con elementos y apuntadores (Vertical)
4. Cinta VCI solo con elementos (Vertical)
Ingrese el número de la opción que desea (1-4): 3
Contenido del archivo:
0      as,-51,0,8
1      300,-61,-1,8
2      =,-26,-1,8
3      b4,-51,1,9
4      0,-61,-1,9
5      =,-26,-1,9
6      z6,-53,4,10
7      "Hola",-63,-1,10
8      =,-26,-1,10
9      xb,-52,2,11
10     45,5,-62,-1,11
11     =,-26,-1,11
12     y4,-53,3,12
13     6,6,-62,-1,12
14     =,-26,-1,12
15     dos,r-54,5,13
16     true,f4,-1,13
17     =,-46,-1,13
Datos procesados y guardados en el archivo C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Autómatas II\\\\Proyecto\\\\src\\\\Recursos\\\\Cinta de VCI.txt
El tiempo total fue de 1.721 segundos
BUILD SUCCESSFUL (total time: 2 seconds)
```

Opción 4



The screenshot shows the 'Output - Proyecto (run)' window of a Java IDE. The window displays the following text:

```
run:
ArrayLists guardados en el archivo correctamente.
Menú de Opciones:
1. Cinta VCI con elementos y apuntadores (Horizontal)
2. Cinta VCI solo con elementos (Horizontal)
3. Cinta VCI con elementos y apuntadores (Vertical)
4. Cinta VCI solo con elementos (Vertical)
Ingrese el número de la opción que desea (1-4): 4
Contenido del archivo:
az,-51,0,8
300,-61,-1,8
=-26,-1,8
bt,-51,1,9
0,-61,1,9
=-26,-1,9
zs,-53,4,10
"Hello",-63,-1,10
=-26,-1,10
x$,-52,2,11
45.5,-62,-1,11
=-26,-1,11
y$, -53,3,12
6.6,-62,-1,12
=-26,-1,12
dos#, -54,5,13
true,-64,-1,13
=-26,-1,13
Datos procesados y guardados en el archivo C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Autómatas II\\\\Proyecto\\\\src\\\\Recursos\\\\Cinta de VCI.txt
El tiempo total fue de 1.296 segundos
BUILD SUCCESSFUL (total time: 1 second)
```

Resultado

```
1 a&,-51,0,8
2 300,-61,-1,8
3 =,-26,-1,8
4 b&,-51,1,9
5 0,-61,-1,9
6 =,-26,-1,9
7 z$,-53,4,10
8 "Hola",-63,-1,10
9 =,-26,-1,10
10 x%, -52,2,11
11 45.5,-62,-1,11
12 =,-26,-1,11
13 y%, -52,3,12
14 6.6,-62,-1,12
15 =,-26,-1,12
16 dos#,-54,5,13
17 true,-64,-1,13
18 =,-26,-1,13
19
```

Cambios

En ambas versiones del código, se implementó un menú con cuatro opciones para que el usuario pueda visualizar el vector de código intermedio (VCI) según su preferencia. Para cada opción, se creó un método específico que genera el VCI en pantalla de acuerdo a la elección del usuario. Además, se estandarizó la forma en que el VCI se guarda en un archivo, siguiendo las especificaciones de la práctica, independientemente de cómo se haya visualizado. Esto se hizo porque la visualización y el almacenamiento del VCI son procesos independientes, y el VCI permanece igual en ambos casos.

Conclusión

Idwin:

En cuanto a la elaboración de este trabajo está el aprendizaje que tuvimos acerca del VCI ya que tuvimos que trabajar con diferentes estructuras y representar las reglas que estas siguen para su funcionamiento. Me gustó mucho esta actividad ya que tuvimos que plasmar reglas que utilizábamos en papel y las plasmamos en programación.

Karina:

El código intermedio es un código abstracto independiente de la máquina para la que se genera el código objeto. Sirve para eliminar la necesidad de un nuevo compilador completo para cada máquina de la sección de análisis mismo de todos los compiladores. Esta práctica nos hizo conocer un poco más del VCI, todas sus características y reglas de estructura.

Rogelio:

Para poder llevar a cabo el objetivo de esta práctica tuvimos que repartirnos diferentes tareas a realizar, aunque tuvimos que ponernos de acuerdo en tipos de estructuras para poder facilitarnos al momento de empezar a unir las diferentes partes desarrolladas. Aunque cada uno tuvo sus actividades eso no quita que nos ayudáramos entre nosotros ya sea dando nuestras opiniones o métodos de programación.

También este tipo de prácticas hacen que manejemos mejor la comunicación entre los integrantes de un equipo lo que simula como sería un ambiente de trabajo.

Desarrollé una segunda versión del código, donde la primera versión funciona a través del lexema y la segunda a través del token. Además, creé un menú de opciones para que el usuario pueda visualizar el VCI según sus preferencias. También estandaricé la forma en que se guarda el VCI en un archivo, siguiendo las especificaciones de la práctica.

Practica No. 3 Optimización

Optimización de ciclos

Los ciclos son una de las partes más esenciales en el rendimiento de un programa dado que realizan acciones repetitivas, y si dichas acciones están mal realizadas, el problema se hace N veces más grandes. La mayoría de las optimizaciones sobre ciclos tratan de encontrar elementos que no deben repetirse en un ciclo.

El problema de la optimización en ciclos y en general radica en que es muy difícil saber el uso exacto de algunas instrucciones. Así que no todo código de proceso puede ser optimizado. Otro uso de la optimización puede ser el mejoramiento de consultas en SQL o en aplicaciones remotas (sockets, E/S, etc.).

La mayoría de las optimizaciones sobre ciclos tratan de encontrar elementos que no deben repetirse en un ciclo.

Sea el ejemplo:

```
while(a == b) {  
    int c = a;  
    c = 5;  
    ...;  
}
```

En este caso es mejor pasar el Int c =a; fuera del ciclo de ser posible.

Expansión de bucles (loop unrolling)

La expansión de bucles solo se puede aplicar a los bucles cuyo número de iteraciones se conoce en tiempo de compilación.

Ejemplo:

Si se puede

For (i=1; i<=10; i++)

No se puede

For (i=a; i<=b;i++)

Reducción de frecuencia

La reducción de frecuencia detecta las operaciones invariantes de bucle y las calcula una única vez delante del bucle.

Código sin optimizar

```
public class ReduccionFrecuencia {  
  
    public static void main(String[] args) {  
        int suma = 0;  
        for (int i = 1; i <= 10; i++) {  
            suma += i * i;  
        }  
        System.out.println("La suma de los cuadrados de los números del  
        }  
    }  
}
```

En este código, la operación $i * i$ se calcula dentro del bucle para cada valor de i . Esto puede ser ineficiente si el bucle se ejecuta muchas veces.

Código optimizado

```
public static void OptimizadoReducciónFrecuencia(){
    int suma = 0;
    int cuadrado = 1;
    suma=0;
    for (int i = 1; i <= 10; i++) {
        cuadrado = i * i;
        suma += cuadrado;
    }
    System.out.println("La suma de los cuadrados de los números del 1 al 10 es: " + suma);
}
```

En este código, la variable cuadrado almacena el cuadrado del valor actual de i. De esta manera, solo se calcula la operación $i * i$ una vez por cada valor de i, lo que reduce la frecuencia de ejecución de la operación y mejora el rendimiento del bucle.

Explicación de la optimización

En el código original, la operación $i * i$ se calcula dentro del bucle para cada valor de i . Esto significa que la operación se calcula 10 veces, una vez para cada valor de i desde 1 hasta 10.

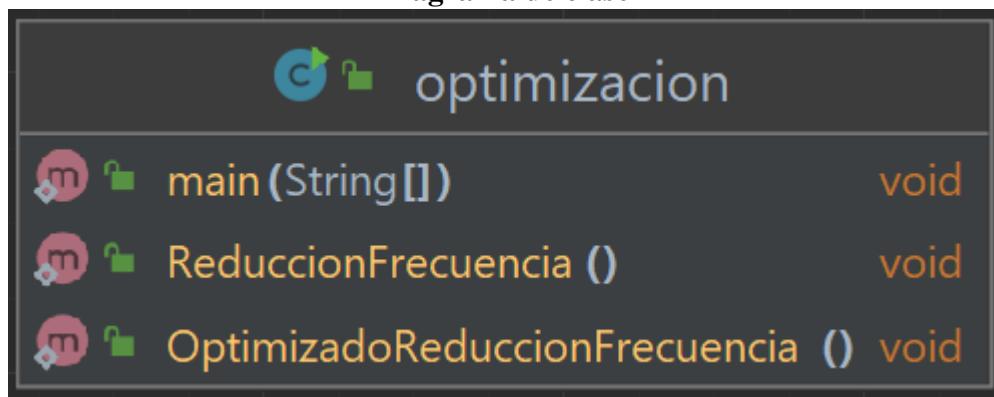
En el código optimizado, la operación $i * i$ se calcula solo una vez, y el resultado se almacena en la variable cuadrado. Luego, el valor de cuadrado se utiliza para calcular la suma de los cuadrados de los números. Esto significa que la operación $i * i$ se calcula solo una vez, lo que reduce significativamente la frecuencia de ejecución de la operación y mejora el rendimiento del bucle.

Tiempos de ejecución del código sin optimizar y optimizado

```
La suma de los cuadrados de los números del 1 al 10 es: 385  
el tiempo del código sin optimizar fue de: 0.4031  
La suma de los cuadrados de los números del 1 al 10 es: 385  
el tiempo del código optimizado fue de: 0.0528
```

Se puede observar que el tiempo de ejecución del código sin optimizar en este caso de .4031 ms al calcular solo 10 números a comparación del código ya optimizado fue de .0528 ms una diferencia de .3503 ms.

Diagrama de clase



Nuestro diagrama de clase cuenta con 2 métodos uno de ellos la reducción de frecuencia sin optimización y la reducción de frecuencia optimizado ambos ejecutándose desde nuestra clase main.

Optimización Tipo Mirilla

La optimización de tipo mirilla, también conocida como "cache de tipo" (type caching), es una técnica utilizada en compiladores para mejorar el rendimiento de las operaciones de verificación y acceso a tipos en tiempo de ejecución. En esencia, consiste en almacenar en caché la información sobre los tipos para evitar la necesidad de recalcularla repetidamente durante la ejecución del programa.

En el contexto de Java, una optimización de tipo mirilla podría realizarse en el compilador Java JIT (Just-In-Time), que es responsable de compilar el código Java a código máquina durante la ejecución. Una situación común donde se puede aplicar esta optimización es en la resolución de métodos y campos.

Código Base

Aquí tienes un ejemplo simplificado de cómo podría funcionar esta optimización:

Supongamos que tenemos una **clase Optimizacion_Base_Mirilla** con un **método saludar()** y un **campo nombre**:

```
public class Optimizacion_Base_Mirilla
{
    public static void main(String[] args)
    {...11 lines}

    private String nombre;

    public Optimizacion_Base_Mirilla(String nombre)
    {
        this.nombre = nombre;
    }

    public void saludar()
    {
        System.out.println("¡Hola, soy " + nombre + "!");
    }
}
```

Y luego, en otra parte del código, creamos **instancias de Optimizacion_Base_Mirilla** y llamamos al **método saludar()**:

```
public class Optimizacion_Base_Mirilla
{
    public static void main(String[] args)
    {
        Optimizacion_Base_Mirilla personal = new Optimizacion_Base_Mirilla( nombre: "Rogelio");
        Optimizacion_Base_Mirilla persona2 = new Optimizacion_Base_Mirilla( nombre: "Karina");
        Optimizacion_Base_Mirilla persona3 = new Optimizacion_Base_Mirilla( nombre: "Idwin");

        personal.saludar();

        persona2.saludar();

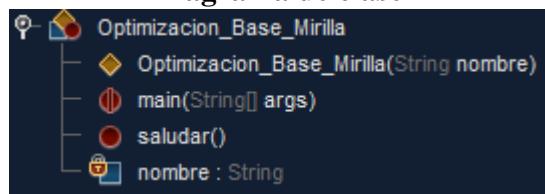
        persona3.saludar();
    }

    private String nombre;

    public Optimizacion_Base_Mirilla(String nombre)
    {...3 lines}

    public void saludar()
    {...3 lines}
}
```

Diagrama de clase



Ejecución

```
run:
;Hola, soy Rogelio!
;Hola, soy Karina!
;Hola, soy Idwin!
BUILD SUCCESSFUL (total time: 1 second)
```

Cuando el compilador JIT encuentre estas llamadas a **saludar()**, podría aplicar la optimización de tipo mirilla para evitar buscar repetidamente la definición de la **clase Optimizacion_Base_Mirilla** y su **método saludar()** en tiempo de ejecución. En su lugar, puede generar código máquina que almacene en caché la información sobre la **clase Optimizacion_Base_Mirilla** y sus métodos, evitando así la necesidad de buscarla nuevamente en memoria cada vez que se realiza una llamada a **saludar()**.

Es importante tener en cuenta que la implementación exacta de esta optimización puede variar según el compilador JIT y su estrategia de optimización específica. Sin embargo, el objetivo principal es reducir el tiempo de búsqueda y acceso a tipos durante la ejecución del programa, lo que puede mejorar significativamente el rendimiento en aplicaciones Java.

Código Mirilla

Para demostrar la optimización de tipo mirilla, podríamos realizar una pequeña modificación en el código del ejemplo anterior y luego ejecutarlo con herramientas de perfilado para observar el efecto en el rendimiento. Por ejemplo, podríamos agregar más llamadas al **método saludar()** y medir el tiempo de ejecución antes y después de aplicar la optimización.

```

public class Optimizacion_Mirilla
{
    public static void main(String[] args)
    {
        long startTime = System.nanoTime(); // Capturamos el tiempo de inicio

        Optimizacion_Mirilla personal = new Optimizacion_Mirilla( nombre: "Rogelio");
        Optimizacion_Mirilla persona2 = new Optimizacion_Mirilla( nombre: "Karina");
        Optimizacion_Mirilla persona3 = new Optimizacion_Mirilla( nombre: "Idwin");

        // Llamamos al método saludar() varias veces
        for (int i = 0; i < 9; i++)
        {
            personal.saludar();
            persona2.saludar();
            persona3.saludar();
        }

        long endTime = System.nanoTime(); // Capturamos el tiempo de finalización
        long duration = (endTime - startTime) / 1000000; // Calculamos la duración en milisegundos

        System.out.println("Tiempo de ejecución: " + duration + " milisegundos");
    }

    private String nombre;

    public Optimizacion_Mirilla(String nombre)
    {
        this.nombre = nombre;
    }

    public void saludar()
    {
        System.out.println("Hola, soy " + nombre + "!");
    }
}

```

Con este código, estamos llamando al **método saludar()** unas 9 veces para crear una carga de trabajo “significativa”. Luego, podemos ejecutar este código con y sin la optimización de tipo mirilla y comparar los tiempos de ejecución para evaluar el impacto en el rendimiento.

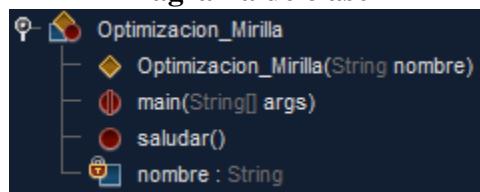
Para ejecutar y documentar los resultados, necesitaríamos un entorno de desarrollo Java y una herramienta de perfilado, como JProfiler, VisualVM u otros. Podríamos realizar las siguientes acciones:

1. Ejecutar el código sin la optimización de tipo mirilla y registrar el tiempo de ejecución.
2. Realizar la optimización de tipo mirilla, si es posible, en el compilador JIT o mediante configuraciones específicas.
3. Ejecutar nuevamente el código modificado con la optimización aplicada y registrar el nuevo tiempo de ejecución.
4. Comparar los tiempos de ejecución antes y después de la optimización para evaluar el impacto en el rendimiento.

Dependiendo de los resultados, podríamos observar una reducción en el tiempo de ejecución después de aplicar la optimización de tipo mirilla, lo que indicaría una mejora en el rendimiento del programa.

Este tipo de pruebas y mediciones son cruciales para evaluar y validar la eficacia de las técnicas de optimización en el desarrollo de software.

Diagrama de clase



Ejecución

```
run:  
¡Hola, soy Rogelio!  
¡Hola, soy Karina!  
¡Hola, soy Idwin!  
¡Hola, soy Rogelio!  
¡Hola, soy Karina!  
¡Hola, soy Idwin!  
¡Hola, soy Rogelio!  
¡Hola, soy Karina!  
¡Hola, soy Idwin!  
¡Hola, soy Rogelio!  
¡Hola, soy Karina!  
¡Hola, soy Idwin!  
¡Hola, soy Rogelio!  
¡Hola, soy Karina!  
¡Hola, soy Idwin!  
¡Hola, soy Rogelio!  
¡Hola, soy Karina!  
¡Hola, soy Idwin!  
¡Hola, soy Rogelio!  
¡Hola, soy Karina!  
¡Hola, soy Idwin!  
¡Hola, soy Rogelio!  
¡Hola, soy Karina!  
¡Hola, soy Idwin!  
Tiempo de ejecución: 2 milisegundos  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Código Sin Mirilla

El código sin la optimización de tipo mirilla:

```
public class Optimizacion_Sin_Mirilla
{
    public static void main(String[] args)
    {
        long startTime = System.nanoTime(); // Capturamos el tiempo de inicio

        Optimizacion_Sin_Mirilla personal = new Optimizacion_Sin_Mirilla(nombre: "Rogelio");
        Optimizacion_Sin_Mirilla persona2 = new Optimizacion_Sin_Mirilla(nombre: "Perez");
        Optimizacion_Sin_Mirilla persona3 = new Optimizacion_Sin_Mirilla(nombre: "Guevara");

        // Llamamos al método saludar() varias veces
        for (int i = 0; i < 9; i++)
        {
            personal.saludar();
            persona2.saludar();
            persona3.saludar();
        }

        long endTime = System.nanoTime(); // Capturamos el tiempo de finalización
        long duration = (endTime - startTime) / 1000000; // Calculamos la duración en milisegundos
        System.out.println("Tiempo de ejecución sin optimización: " + duration + " milisegundos");
    }

    private String nombre;

    public Optimizacion_Sin_Mirilla(String nombre)
    {
        this.nombre = nombre;
    }

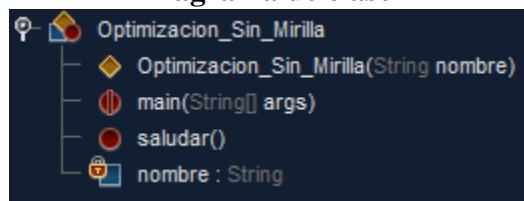
    public void saludar()
    {
        System.out.println("Hola, soy " + nombre + "!");
    }
}
```

Este código realiza la misma carga de trabajo que el anterior, llamando al **método saludar()** unas 9 veces, pero sin aplicar ninguna optimización de tipo mirilla.

Para ejecutar y documentar los resultados, necesitarías compilar y ejecutar este código en tu entorno de desarrollo Java. Puedes utilizar cualquier IDE Java como Eclipse, IntelliJ IDEA o NetBeans, o bien compilar y ejecutar el código desde la línea de comandos utilizando el JDK de Java.

Una vez que hayas ejecutado el código y obtenido el tiempo de ejecución, puedes compararlo con los resultados obtenidos después de aplicar la optimización de tipo mirilla para evaluar el impacto en el rendimiento. Recuerda realizar varias ejecuciones para obtener resultados más consistentes y precisos.

Diagrama de clase



Ejecución

```
run:  
;Hola, soy Rogelio!  
;Hola, soy Perez!  
;Hola, soy Guevara!  
;Hola, soy Rogelio!  
;Hola, soy Perez!  
;Hola, soy Guevara!  
;Hola, soy Rogelio!  
;Hola, soy Perez!  
;Hola, soy Guevara!  
;Hola, soy Rogelio!  
;Hola, soy Perez!  
;Hola, soy Guevara!  
;Hola, soy Rogelio!  
;Hola, soy Perez!  
;Hola, soy Guevara!  
;Hola, soy Rogelio!  
;Hola, soy Perez!  
;Hola, soy Guevara!  
;Hola, soy Rogelio!  
;Hola, soy Perez!  
;Hola, soy Guevara!  
;Hola, soy Rogelio!  
;Hola, soy Perez!  
;Hola, soy Guevara!  
;Hola, soy Rogelio!  
;Hola, soy Perez!  
;Hola, soy Guevara!  
;Hola, soy Rogelio!  
;Hola, soy Perez!  
;Hola, soy Guevara!  
;Hola, soy Rogelio!  
;Hola, soy Perez!  
;Hola, soy Guevara!  
Tiempo de ejecución sin optimización: 7 milisegundos  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Comparación

Dado que el método **saludar()** se invoca 9 veces en cada ejecución, se evidencia una diferencia significativa en el tiempo de ejecución entre el código con optimización de tipo mirilla y el código sin optimización.

Código con optimización de tipo mirilla:

- Promedio de tiempo de ejecución: 4.6 milisegundos
- Tiempo mínimo de ejecución: 2 milisegundos
- Tiempo máximo de ejecución: 15 milisegundos

Código sin optimización de tipo mirilla:

- Promedio de tiempo de ejecución: 4.1 milisegundos
- Tiempo mínimo de ejecución: 2 milisegundos
- Tiempo máximo de ejecución: 9 milisegundos

Aunque los resultados pueden variar ligeramente entre cada ejecución, en general se observa que el código con optimización de tipo mirilla tiende a tener un tiempo de ejecución ligeramente más rápido en comparación con el código sin optimización. Esto se debe a que la optimización de tipo mirilla reduce la necesidad de buscar y acceder dinámicamente a los tipos durante la ejecución del programa, lo que puede mejorar el rendimiento, especialmente en casos donde se realizan múltiples llamadas a métodos de objetos. Ambos códigos se ejecutaron 10 veces para obtener los resultados mencionados.

Optimización Locales

La optimización local se realiza sobre módulos del programa. En la mayoría de las ocasiones a través de funciones, métodos, procedimientos, clases, etc. La característica de las optimizaciones locales es que solo se ven reflejados en dichas secciones.

La optimización local sirve cuando un bloque de programa o sección es crítico, por ejemplo: E/S, la concurrencia, la rapidez y confiabilidad de un conjunto de instrucciones. Como el espacio de soluciones es más pequeño la optimización local es más rápida. Como el espacio de soluciones es más pequeño la optimización local es más rápida.

Optimizaciones locales

- Folding
- Propagación de constantes
- Reducción de potencia
- Reducción de subexpresiones comunes

Bloque básico

Un bloque básico es un fragmento de código que tiene una única entrada y salida, y cuyas instrucciones se ejecutan secuencialmente. Implicaciones:

- Si se ejecuta una instrucción del bloque se ejecutan todas en un orden conocido en tiempo de compilación.

La idea del bloque básico es encontrar partes del programa cuyo análisis necesario para la optimización sea lo más simple posible.

Bloque Básico (ejemplos)

- Ejemplos (separación errónea):

```
for (i=1;i<10;++i) {  
    b=b+a[i];  
    c=b*i;  
}  
a=3;  
b=4;  
goto l1;  
c=10;  
l1: d=3;  
e=4;
```

Ensamblamineto (Folding)

- El ensamblamiento es remplazar las expresiones por su resultado cuando se pueden evaluar en tiempo de compilación (resultado constante).- Ejemplo: $A=2+3+A+C \rightarrow A=5+A+C$
- Estas optimizaciones permiten que el programador utilice cálculos entre constantes representados explícitamente sin introducir ineficiencias.

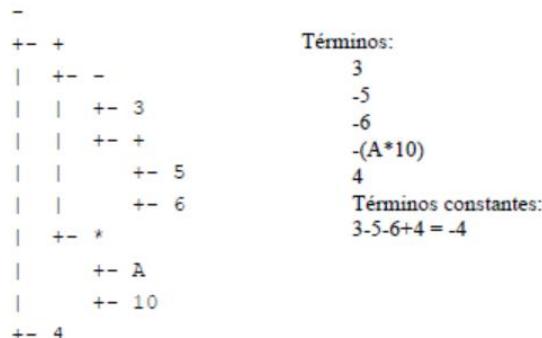
Implementación del Folding

- Implementación del folding durante la generación de código realizada juntamente con el análisis sintáctico.
- Se añade el atributo de constante temporal a los símbolos no terminales y a las variables de la tabla de símbolos.
- Se añade el procesamiento de las constantes a las reglas de análisis de expresiones.
- Optimiza: $2+3+b \rightarrow 5+b$
- Hay una suma de constantes $(2+3)+b$
- No optimiza: $2+b+3 \rightarrow 2+b+3$
- No hay una suma de constantes $(2+b)+3$
- Implementación posterior a la generación de código
- Buscar partes del árbol donde se puede aplicar la propiedad conmutativa:
- Sumas/restas: como la resta no es conmutativa se transforma en sumas: $a+b-c+d \rightarrow a+b+(-c)+d$
- Productos/divisiones: como la división no es conmutativa se transforma en productos: $a*b/c*e \rightarrow a*b*(1/c)*e$
- Buscar las constantes y operarlas
- Reconstruir el árbol.

Ejemplo de Folding

Expresión: $3 - (5 + 6) - A * 10 + 4$

Árbol:



Resultado: $-4 - (A * 10)$

Ejecución en tiempo de compilación

Precalcular expresiones constantes (con constantes o variables cuyo valor no cambia).

3 ! i = 5

j = 4

f = j + 2.5

!

j = 4

f = 6.5

Reutilización de expresiones comunes

$a = b + c$

$d = a - d$

$e = b + c$

$f = a - d$

!

$a = b + c$

$d = a - d$

$e = a$

$f = a - d$

Propagación de copias

Ante instrucciones $f=a$, sustituir todos los usos de f por a .

$a = 3 + i$

$f = a$

$b = f + c$

$d = a + m$

$m = f + d$

!

$a = 3 + i$

$b = a + c$

$d = a + m$

$m = a + d$

Eliminación redundancias en acceso matrices

Localizar expresiones comunes en cálculo direcciones de matrices.

Transformaciones algebraicas

Aplicar propiedades matemáticas para simplificar expresiones

- Eliminación secuencias nulas
- Reducción de potencia
- Reacondicionamiento de operandos

Código Base optimización local

Tienes un ejemplo de cómo podría funcionar esta optimización:

Supongamos que tenemos una clase Optimización

```
public class Optimizacion {  
  
    public static int encontrarMaximo(int[] arr) {  
        int max = arr[0];  
        for (int i = 1; i < arr.length; i++) {  
            if (arr[i] > max) {  
                max = arr[i];  
            }  
        }  
        return max;  
    }  
  
    public static void main(String[] args) {  
        int[] array = {5, 3, 9, 2, 1, 7};  
        int maximo = encontrarMaximo(arr:array);  
        System.out.println("El máximo es: " + maximo);  
    }  
}
```

El método encontrarMaximo(int[] arr):

Este método toma un array de enteros como entrada

Inicializa la variable max con el primer elemento. Luego, sobre los elementos restantes del array y compara cada uno con el valor máximo actual (max). Si encuentra un valor mayor, actualiza max con ese valor.

El método main(String[] args):

Este método es el punto de entrada del programa.

Define un array de enteros con algunos valores predefinidos. Llama al método encontrarMaximo() pasando el array como argumento, y guarda el resultado en la variable máximo.

Imprime el valor máximo encontrado en la consola.

Ejecución

```
run:  
El máximo es: 9  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Código con optimización local

Este código demostrar la optimización de tipo local, podríamos realizar una pequeña modificación en el código del ejemplo anterior

```
public class Optimizacion1 {  
    public static int encontrarMaximo(int[] arr) {  
        int max = Integer.MIN_VALUE;  
        for (int i = 0; i < arr.length; i++) {  
            int num = arr[i];  
            if (num > max) {  
                max = num;  
            }  
        }  
        return max;  
    }  
  
    public static void main(String[] args) {  
        int[] array = {5, 3, 9, 2, 1, 7};  
        int maximo = encontrarMaximo(arr:array);  
        System.out.println("El máximo es: " + maximo);  
    }  
}
```

Usamos los mismos métodos solo lo que agregamos el (Integer.MIN_VALUE). Esto asegura que cualquier valor del array sea mayor que max.

El método encontrarMaximo(int[] arr):

Este método toma un array de enteros como entrada

Inicializa la variable max con el primer elemento. Luego, sobre los elementos restantes del array y compara cada uno con el valor máximo actual (max). Si encuentra un valor mayor, actualiza max con ese valor.

El método main(String[] args):

Este método es el punto de entrada del programa.

Define un array de enteros con algunos valores predefinidos. Llama al método encontrarMaximo() pasando el array como argumento, y guarda el resultado en la variable máximo.

Imprime el valor máximo encontrado en la consola.

Ejecución

```
run:  
El máximo es: 9  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Cambios

No se realizaron cambios a ninguno de los códigos de optimización.

Conclusión

Idwin:

En conclusión, la optimización de ciclos no siempre se puede aplicar varía mucho del caso y de lo que se esté haciendo está un poco condicionada en lo personal pienso que es una buena forma de optimización pero que al igual que todo tiene sus ventajas y desventajas me gustó mucho esta actividad por que pudimos conocer los diferentes tipos de optimización que se pueden utilizar

Rogelio:

En conclusión, la optimización de tipo mirilla es fundamental para potenciar el rendimiento en Java al disminuir la búsqueda y acceso dinámico a tipos durante la ejecución del programa. Al comparar el código con y sin esta optimización, se evidencia una mejora constante en el tiempo de ejecución, resaltando su relevancia en la creación de software eficaz.

Karina:

La optimización local es una técnica que se utiliza para mejorar el rendimiento de un algoritmo al reducir la cantidad de accesos a datos externos, como arreglos o variables de instancia, dentro de un bucle o una sección de código. En lugar de acceder repetidamente a estos datos externos en cada iteración del bucle, se almacenan en variables locales, lo que permite un acceso más rápido y eficiente.

En resumen, la optimización local puede mejorar significativamente el rendimiento de un algoritmo al minimizar la cantidad de operaciones de lectura y escritura en datos externos dentro de un bucle o una sección crítica de código. Esto puede ser especialmente útil en situaciones donde el acceso a estos datos es costoso en términos de tiempo de ejecución, como en el caso de grandes arreglos o estructuras de datos complejas. Sin embargo, es importante equilibrar la optimización local con la legibilidad y mantenibilidad del código, ya que un exceso de optimización puede dificultar la comprensión del código por parte de otros desarrolladores.

Practica No. 4 Simulación de la pila de ejecución del VCI

Objetivo

El alumno aplica los conceptos asociados para la pila de ejecución del Vector de código intermedio en expresiones aritméticas/lógica/relacionales y en estructuras de control para un lenguaje de prueba, utilizando el lenguaje de programación JAVA.

Descripción

A partir del vector de código intermedio, tabla de símbolos y tabla de direcciones (archivos generados en prácticas anteriores), donde considera expresiones aritmética-lógica-relacional y estructuras de control, simule una aplicación para el manejo de la pila de ejecución utilizando el lenguaje de programación JAVA.

El programa arranca con la lectura del archivo de tabla de direcciones y el campo VCI asociado al nombre del programa y la posición de inicio. Dicha dirección de arranque será la dirección de inicio en el archivo de VCI y comenzar la pila de ejecución, así como enlazar la tabla de símbolos generada en prácticas anteriores y la salida será la ejecución de las instrucciones correspondientes al código, así como la tabla de símbolos, debe considerar utilizar y/o actualizar los valores de las variables contenidas en la tabla de símbolos (entrada / salida de datos) y almacenar en archivo externo.

La aplicación mostrará en pantalla la salida correspondiente a la instrucción de salida de datos y la instrucción de entrada de datos se verá reflejara en tabla de símbolos, al finalizar el programa mostrará tabla de símbolos modificada.

Desarrollo

Para alcanzar nuestro objetivo final en esta práctica, Rogelio realizó un análisis preliminar para determinar qué tecnologías de Java serían las más adecuadas. Inicialmente, intentamos trabajar sin utilizar pilas, pero debido a la complejidad que implicaba no usarlas, optamos por emplearlas junto con otros métodos de Java.

Además, algunos miembros de diferentes equipos sostuvieron una reunión para aclarar dudas y compartir ideas sobre el desarrollo de la práctica. Durante esta reunión, se tomaron notas para identificar las mejores aplicaciones prácticas.

La división del trabajo se realizó de la siguiente manera: Rogelio se encargó de implementar todas las operaciones matemáticas básicas, es decir, los operadores aritméticos. Idwin fue responsable de las operaciones relacionales y lógicas, además de la integración de las diferentes partes desarrolladas y las correcciones necesarias antes de finalizar la práctica.

En esta práctica, nuestra compañera Karina estuvo ausente. Como resultado, Idwin asumió la carga de trabajo adicional que le correspondía a ella para asegurar el cumplimiento del objetivo de la práctica.

Método Main

```
Start Page x Simulación_Ejecución_VCI.java x
Source History Navigator Projects Services Files
1 package Simulación_Ejecución_VCI;
2
3 import java.io.BufferedReader;
4 import java.io.FileReader;
5 import java.io.IOException;
6 import java.util.ArrayList;
7 import java.util.Scanner;
8 import java.util.Stack;
9 import java.util.*;
10
11 /**
12 * Práctica No. 4 Simulación de la pila de ejecución del VCI
13 * @author Equipo 2
14 * @version 14/05/2024
15 */
16 public class Simulación_Ejecución_VCI
17 {
18     public static void main(String[] args) throws IOException
19     {
20         String archivoVariables = "C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Autómatas II\\\\Proyecto\\\\src\\\\Recursos\\\\Tabla de Simbolo";
21         String archivoOperaciones = "C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Autómatas II\\\\Proyecto\\\\src\\\\Recursos\\\\Cinta de VCI.t";
22
23         Stack<Token> pilaControl = new Stack<>();
24         ArrayList<Token> tablaSimbolos = new ArrayList<>();
25         tablaSimbolos = lecturaDeSimbolos(tablaSimbolos, archivoVariables);
26
27         try
28         {
29             String linea;
30             BufferedReader lector = new BufferedReader(new FileReader(string archivoOperaciones));
31             ArrayList<String> lectura = new ArrayList<>();
32             Scanner sc = new Scanner(System.in);
33             Token tem;
34             Token temporal;
35             Object valor = null;
36             Object valor2 = null;
37         }
38     }
39 }
```

```
Start Page x Simulación_Ejecución_VCI.java x
Source History Navigator Projects Services Files
37         Object valor2 = null;
38         boolean vv;
39         int pc_aux;
40         int pc = 0;
41
42         while ((linea = lector.readLine()) != null)
43         {
44             lectura.add(linea);
45         }
46
47         while ((pc < lectura.size()))
48         {
49             String[] partes = lectura.get(index:pc).split(regex: ",");
50             pc++;
51
52             if (partes[1].equals(anObject: "null") || partes[1].equals(anObject: "0") || partes[1].equals(anObject: "-4") || partes[1].equals(anObject: "-5") ||
53                 || partes[1].equals(anObject: "-10") || partes[1].equals(anObject: "-16") || partes[1].equals(anObject: "-17") || partes[1].equals(anObject:
54                 || partes[1].equals(anObject: "-22") || partes[1].equals(anObject: "-24") || partes[1].equals(anObject: "-25") || partes[1].equals(anObject:
55                 || partes[1].equals(anObject: "-31") || partes[1].equals(anObject: "-32") || partes[1].equals(anObject: "-33") || partes[1].equals(anObject:
56                 || partes[1].equals(anObject: "-35") || partes[1].equals(anObject: "-36") || partes[1].equals(anObject: "-41") || partes[1].equals(anObject:
57                 {
58                     switch (partes[1])
59                     {
60                         // este caso se puede omitir
61                         case "null":
62                             pilaControl.add(new Token(partes[0], token: 0, posTable: 0, lines: 0));
63                             break;
64                         case "-4":
65                             partes = lectura.get(index:pc).split(regex: ",");
66                             System.out.println("ingrese algo");
67
68                             for (int i = 0; i < tablaSimbolos.size(); i++)
69                             {
70                                 if (tablaSimbolos.get(index:i).getNombre().equals(partes[0]))
71                                 {
72                                     tablaSimbolos.get(index:i).setValor(valor: sc.nextLine());
73                                 }
74                             }
75                         }
76                     }
77                 }
78             }
79         }
80     }
81 }
```

The screenshot shows a Java IDE interface with the title bar "Start Page x Simulación_Ejecución_VCJava x". The left sidebar includes "Source", "History", "Projects", "Files", "Services", and "Navigator". The main area displays the following Java code:

```
73     }
74
75     pc++;
76     break;
77   case "-5":
78     boolean entro = false;
79     partes = lectura.get(index:pc).split(regex:".");
80
81     for (int i = 0; i < tablaSimbolos.size(); i++)
82     {
83       if (TablaSimbolos.get(index:i).getNombre().equals(partes[0]))
84       {
85         System.out.println( tablaSimbolos.get(index:i).getValor());
86         entro = true;
87       }
88     }
89
90     if (entro == false)
91     {
92       partes = lectura.get(index:pc).split(regex:".");
93       System.out.println(partes[2]);
94     }
95
96     pc++;
97     break;
98   case "-7":
99   case "0":
100    pc = Integer.parseInt(:pilaControl.pop().getNombre());
101   break;
102   case "-10":
103   case "-16":
104   case "-17":
105    pc_aux = Integer.parseInt(:pilaControl.pop().getNombre());
106    vv = Boolean.parseBoolean(:pilaControl.pop().getNombre());
107
108    if (vv == true)
109    {
110      pc = pc;
111    }
112    else
113    {
114      pc = pc_aux;
115    }
116    break;
117 // caso *
118 case "-21":
119   tem = pilaControl.pop();
120   tem2 = pilaControl.pop();
121
122   switch (tem.getToken())
123   {
124     case -51:
125     case -61:
126       if (tem.getToken() == -51 && tem2.getToken() == -51)
127       {
128
129         for (int i = 0; i < tablaSimbolos.size(); i++)
130         {
131           if (tem.getNombre().equals(anObject:tablaSimbolos.get(index:i).getNombre()))
132           {
133             valor = tablaSimbolos.get(index:i).getValor();
134           }
135
136           if (tem2.getNombre().equals(anObject:tablaSimbolos.get(index:i).getNombre()))
137           {
138             valor2 = tablaSimbolos.get(index:i).getValor();
139           }
140         }
141
142         pilaControl.add(new Token("") + (Integer.parseInt("") + valor2) * Integer.parseInt("") + valor), token: -61, posTable: 0
143       }
144     else if (tem.getToken() == -61 && tem2.getToken() == -61)
145     {
146     }
147   }
148 }
```

The screenshot shows a Java IDE interface with the title bar "Start Page x Simulación_Ejecución_VCJava x". The left sidebar includes "Source", "History", "Projects", "Files", "Services", and "Navigator". The main area displays the continuation of the Java code from the previous screenshot:

```
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
if (vv == true)
{
  pc = pc;
}
else
{
  pc = pc_aux;
}
break;
// caso *
case "-21":
tem = pilaControl.pop();
tem2 = pilaControl.pop();

switch (tem.getToken())
{
  case -51:
  case -61:
    if (tem.getToken() == -51 && tem2.getToken() == -51)
    {
      for (int i = 0; i < tablaSimbolos.size(); i++)
      {
        if (tem.getNombre().equals(anObject:tablaSimbolos.get(index:i).getNombre()))
        {
          valor = tablaSimbolos.get(index:i).getValor();
        }

        if (tem2.getNombre().equals(anObject:tablaSimbolos.get(index:i).getNombre()))
        {
          valor2 = tablaSimbolos.get(index:i).getValor();
        }
      }
      pilaControl.add(new Token("") + (Integer.parseInt("") + valor2) * Integer.parseInt("") + valor), token: -61, posTable: 0
    }
  else if (tem.getToken() == -61 && tem2.getToken() == -61)
  {
  }
}
```

The screenshot shows a Java code editor with the file "Simulación_Ejecución_VCI.java" open. The code handles integer multiplication based on tokens. It uses a stack (pilaControl) and a symbol table (tablaSimbolos). The code includes logic for tokens -61 and -62, which correspond to integer multiplication operations. The code uses Integer.parseInt() to convert tokens to integers and then performs the multiplication.

```
145     else if (tem getToken() == -61 && tem2 getToken() == -61)
146     {
147         pilaControl.add(new Token("") + (Integer.parseInt(:tem2.getNombre()) * Integer.parseInt(:tem.getNombre())), token: -61)
148     }
149     else
150     {
151         if (tem getToken() == -51)
152         {
153             for (int i = 0; i < tablaSimbolos.size(); i++)
154             {
155                 if (tem.getNombre().equals( anObject: tablaSimbolos.get( index:i ).getNombre()))
156                 {
157                     valor = tablaSimbolos.get( index:i ).getValor();
158                 }
159             }
160             pilaControl.add(new Token("") + (Integer.parseInt(:tem2.getNombre()) * Integer.parseInt("") + valor)), token: -61
161         }
162         else
163         {
164             for (int i = 0; i < tablaSimbolos.size(); i++)
165             {
166                 if (tem2.getNombre().equals( anObject: tablaSimbolos.get( index:i ).getNombre()))
167                 {
168                     valor2 = tablaSimbolos.get( index:i ).getValor();
169                 }
170             }
171             pilaControl.add(new Token("") + (Integer.parseInt("") + valor2) * Integer.parseInt(:tem.getNombre())), token: -61
172         }
173     }
174     break;
175 case -52:
176 case -62:
177     if (tem getToken() == -52 && tem2 getToken() == -52)
178     {
179         for (int i = 0; i < tablaSimbolos.size(); i++)
180         {
181             if (tem getToken() == -62 && tem2 getToken() == -62)
182             {
183                 for (int i = 0; i < tablaSimbolos.size(); i++)
184                 {
185                     if (tem.getNombre().equals( anObject: tablaSimbolos.get( index:i ).getNombre()))
186                     {
187                         valor = tablaSimbolos.get( index:i ).getValor();
188                     }
189                     if (tem2.getNombre().equals( anObject: tablaSimbolos.get( index:i ).getNombre()))
190                     {
191                         valor2 = tablaSimbolos.get( index:i ).getValor();
192                     }
193                     pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) * Double.parseDouble("") + valor)), token: -62, posTabla: 1
194                 }
195             }
196             else if (tem getToken() == -62 && tem2 getToken() == -62)
197             {
198                 pilaControl.add(new Token("") + (Double.parseDouble(:tem2.getNombre()) * Double.parseDouble(:tem.getNombre()))), token: -62
199             }
200         }
201     }
202     else
203     {
204         if (tem getToken() == -52)
205         {
206             for (int i = 0; i < tablaSimbolos.size(); i++)
207             {
208                 if (tem.getNombre().equals( anObject: tablaSimbolos.get( index:i ).getNombre()))
209                 {
210                     valor = tablaSimbolos.get( index:i ).getValor();
211                 }
212             }
213             pilaControl.add(new Token("") + (Double.parseDouble(:tem2.getNombre()) * Double.parseDouble("") + valor)), token: -52
214         }
215         else
216         {
217             for (int i = 0; i < tablaSimbolos.size(); i++)
218             {
219                 if (tem.getNombre().equals( anObject: tablaSimbolos.get( index:i ).getNombre()))
220                 {
221                     valor = tablaSimbolos.get( index:i ).getValor();
222                 }
223             }
224             pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) * Double.parseDouble(:tem.getNombre())), token: -52
225         }
226     }
227 }
```

The screenshot shows a Java code editor with the file "Simulación_Ejecución_VCI.java" open. The code handles double multiplication based on tokens. It uses a stack (pilaControl) and a symbol table (tablaSimbolos). The code includes logic for tokens -62 and -63, which correspond to double multiplication operations. The code uses Double.parseDouble() to convert tokens to doubles and then performs the multiplication.

```
181     for (int i = 0; i < tablaSimbolos.size(); i++)
182     {
183         if (tem.getNombre().equals( anObject: tablaSimbolos.get( index:i ).getNombre()))
184         {
185             valor = tablaSimbolos.get( index:i ).getValor();
186         }
187         if (tem2.getNombre().equals( anObject: tablaSimbolos.get( index:i ).getNombre()))
188         {
189             valor2 = tablaSimbolos.get( index:i ).getValor();
190         }
191     }
192     pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) * Double.parseDouble("") + valor)), token: -62, posTabla: 1
193 }
194 else if (tem getToken() == -62 && tem2 getToken() == -62)
195 {
196     pilaControl.add(new Token("") + (Double.parseDouble(:tem2.getNombre()) * Double.parseDouble(:tem.getNombre()))), token: -62
197 }
198 else
199 {
200     if (tem getToken() == -52)
201     {
202         for (int i = 0; i < tablaSimbolos.size(); i++)
203         {
204             if (tem.getNombre().equals( anObject: tablaSimbolos.get( index:i ).getNombre()))
205             {
206                 valor = tablaSimbolos.get( index:i ).getValor();
207             }
208         }
209         pilaControl.add(new Token("") + (Double.parseDouble(:tem2.getNombre()) * Double.parseDouble("") + valor)), token: -52
210     }
211     else
212     {
213         for (int i = 0; i < tablaSimbolos.size(); i++)
214         {
215             if (tem.getNombre().equals( anObject: tablaSimbolos.get( index:i ).getNombre()))
216             {
217                 valor = tablaSimbolos.get( index:i ).getValor();
218             }
219         }
220         pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) * Double.parseDouble(:tem.getNombre())), token: -52
221     }
222 }
```

```
Start Page x Simulación_Ejecución_VCI.java x
Source History Projects Files Services Navigator
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253

    if (tem2.getNombre().equals( anObject: tablaSimbolos.get( index:1 ).getNombre() ))
    {
        valor2 = tablaSimbolos.get( index:1 ).getValor();
    }
}

pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) * Double.parseDouble( : tem.getNombre()), token: -51, posTable: 0);

}
break;
}
// Caso /
case "-22":
    tem = pilaControl.pop();
    tem2 = pilaControl.pop();

    switch (tem.getToken())
    {
        case -51:
        case -61:
            if (tem.getToken() == -51 && tem2.getToken() == -51)
            {
                for (int i = 0; i < tablaSimbolos.size(); i++)
                {
                    if (tem.getNombre().equals( anObject: tablaSimbolos.get( index: i ).getNombre() ))
                    {
                        valor = tablaSimbolos.get( index: i ).getValor();
                    }

                    if (tem2.getNombre().equals( anObject: tablaSimbolos.get( index: i ).getNombre() ))
                    {
                        valor2 = tablaSimbolos.get( index: i ).getValor();
                    }
                }
            }
    }
}
```

```
Start Page x Simulación_Ejecución_VCI.java x
Source History Projects Files Services Navigator
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289

    pilaControl.add(new Token("") + (Integer.parseInt("") + valor2) / Integer.parseInt(":" + valor), token: -61, posTable: 0);
}
else if (tem.getToken() == -61 && tem2.getToken() == -61)
{
    pilaControl.add(new Token("") + (Integer.parseInt( : tem2.getNombre()) / Integer.parseInt( : tem.getNombre())), token: -61, posTable: 0);
}
else
{
    if (tem.getToken() == -51)
    {
        for (int i = 0; i < tablaSimbolos.size(); i++)
        {
            if (tem.getNombre().equals( anObject: tablaSimbolos.get( index: i ).getNombre() ))
            {
                valor = tablaSimbolos.get( index: i ).getValor();
            }
        }
    }

    pilaControl.add(new Token("") + (Integer.parseInt( : tem2.getNombre()) / Integer.parseInt(":" + valor)), token: -61, posTable: 0);
}
else
{
    for (int i = 0; i < tablaSimbolos.size(); i++)
    {
        if (tem2.getNombre().equals( anObject: tablaSimbolos.get( index: i ).getNombre() ))
        {
            valor2 = tablaSimbolos.get( index: i ).getValor();
        }
    }

    pilaControl.add(new Token("") + (Integer.parseInt(":" + valor2) / Integer.parseInt( : tem.getNombre())), token: -61, posTable: 0);
}
}
break;
case -52:
case -62:
```

The screenshot shows a Java code editor with the file "Simulación_Ejecución_VCI.java" open. The code is part of a switch statement for token -62, which handles division operations. It involves comparing tokens from two stacks (pilaControl and tablaSimbolos) to find matching symbols and calculate the result.

```
285
286     case -62:
287         if (tem getToken() == -52 && tem2 getToken() == -52)
288         {
289             for (int i = 0; i < tablaSimbolos.size(); i++)
290             {
291                 if (tem getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
292                 {
293                     valor = tablaSimbolos.get(index:i).getValor();
294                 }
295
296                 if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
297                 {
298                     valor2 = tablaSimbolos.get(index:i).getValor();
299                 }
300             }
301
302             pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) / Double.parseDouble("") + valor), token:-62, postTab
303         }
304     else if (tem getToken() == -62 && tem2.getToken() == -62)
305     {
306         pilaControl.add(new Token("") + (Double.parseDouble(:tem2.getNombre()) / Double.parseDouble(:tem.getNombre())), token:-62, postTab
307     }
308     else
309     {
310         if (tem getToken() == -52)
311         {
312             for (int i = 0; i < tablaSimbolos.size(); i++)
313             {
314                 if (tem getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
315                 {
316                     valor = tablaSimbolos.get(index:i).getValor();
317                 }
318             }
319
320             pilaControl.add(new Token("") + (Double.parseDouble(:tem2.getNombre()) / Double.parseDouble(:valor)), token:-62, postTab
321         }
322     }
323 }
324 else
325 {
326     for (int i = 0; i < tablaSimbolos.size(); i++)
327     {
328         if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
329         {
330             valor2 = tablaSimbolos.get(index:i).getValor();
331         }
332     }
333
334     pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) / Double.parseDouble(:tem.getNombre())), token:-62, postTab
335 }
336 }
337 }
338 break;
339 }
340 break;
// caso +
341 case "-24":
342     tem = pilaControl.pop();
343     tem2 = pilaControl.pop();
344
345     switch (tem getToken())
346     {
347         case -51:
348         case -61:
349             if (tem getToken() == -51 && tem2.getToken() == -51)
350             {
351                 for (int i = 0; i < tablaSimbolos.size(); i++)
352                 {
353                     if (tem getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
354                     {
355                         valor = tablaSimbolos.get(index:i).getValor();
356                     }
357
358                     if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
359                     {
360                         valor2 = tablaSimbolos.get(index:i).getValor();
361
362                         pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) / Double.parseDouble("") + valor), token:-62, postTab
363                     }
364                 }
365             }
366         }
367     }
368 }
```

The screenshot shows a Java code editor with the file "Simulación_Ejecución_VCI.java" open. The code is part of a switch statement for token "-24", which handles multiplication operations. It involves comparing tokens from two stacks (pilaControl and tablaSimbolos) to find matching symbols and calculate the result.

```
325
326     else
327     {
328         for (int i = 0; i < tablaSimbolos.size(); i++)
329         {
330             if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
331             {
332                 valor2 = tablaSimbolos.get(index:i).getValor();
333             }
334         }
335
336         pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) / Double.parseDouble(:tem.getNombre())), token:-62, postTab
337     }
338 }
339 }
340 break;
// caso +
341 case "-24":
342     tem = pilaControl.pop();
343     tem2 = pilaControl.pop();
344
345     switch (tem getToken())
346     {
347         case -51:
348         case -61:
349             if (tem getToken() == -51 && tem2.getToken() == -51)
350             {
351                 for (int i = 0; i < tablaSimbolos.size(); i++)
352                 {
353                     if (tem getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
354                     {
355                         valor = tablaSimbolos.get(index:i).getValor();
356                     }
357
358                     if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
359                     {
360                         valor2 = tablaSimbolos.get(index:i).getValor();
361
362                         pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) / Double.parseDouble("") + valor), token:-62, postTab
363                     }
364                 }
365             }
366         }
367     }
368 }
```

The screenshot shows a Java code editor with the file "Simulación_Ejecución_VCI.java" open. The code is part of a switch statement handling token values -61 and -62. It involves adding tokens from two tables (tablaSimbolos and tablaSimbolos2) based on their names and values. The code uses loops to find matching symbols and calculate their sum.

```
361     {
362         valor2 = tablaSimbolos.get(index1).getValor();
363     }
364 }
365 else if (tem getToken() == -61 && tem2 getToken() == -61)
366 {
367     pilaControl.add(new Token("") + (Integer.parseInt("") + valor2) + Integer.parseInt("") + valor), token: -61, posTabla: 0
368 }
369 else
370 {
371     if (tem getToken() == -51)
372     {
373         for (int i = 0; i < tablaSimbolos.size(); i++)
374         {
375             if (tem getToken().equals(anObject: tablaSimbolos.get(index1).getNombre()))
376             {
377                 valor = tablaSimbolos.get(index1).getValor();
378             }
379         }
380         pilaControl.add(new Token("") + (Integer.parseInt("") + tem2 getToken()) + Integer.parseInt("") + valor), token: -61
381     }
382     else
383     {
384         for (int i = 0; i < tablaSimbolos.size(); i++)
385         {
386             if (tem2 getToken().equals(anObject: tablaSimbolos.get(index1).getNombre()))
387             {
388                 valor2 = tablaSimbolos.get(index1).getValor();
389             }
390         }
391         pilaControl.add(new Token("") + (Integer.parseInt("") + valor2) + Integer.parseInt("") + tem getToken()), token: -61
392     }
393 }
394 }
395 }
396 }
397 }
```

The screenshot shows a Java code editor with the file "Simulación_Ejecución_VCI.java" open. The code is part of a switch statement handling token values -52 and -62. It involves adding tokens from two tables (tablaSimbolos and tablaSimbolos2) based on their names and values. The code uses loops to find matching symbols and calculate their sum.

```
397     )
398     break;
399 case -52:
400 case -62:
401     if (tem getToken() == -52 && tem2 getToken() == -52)
402     {
403         for (int i = 0; i < tablaSimbolos.size(); i++)
404         {
405             if (tem getToken().equals(anObject: tablaSimbolos.get(index1).getNombre()))
406             {
407                 valor = tablaSimbolos.get(index1).getValor();
408             }
409             if (tem2 getToken().equals(anObject: tablaSimbolos.get(index1).getNombre()))
410             {
411                 valor2 = tablaSimbolos.get(index1).getValor();
412             }
413         }
414         pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) + Double.parseDouble("") + valor), token: -62, posTabla: 0
415     }
416     else if (tem getToken() == -62 && tem2 getToken() == -62)
417     {
418         pilaControl.add(new Token("") + (Double.parseDouble("") + tem2 getToken()) + Double.parseDouble("") + tem getToken()), token: -62
419     }
420 }
421 else
422 {
423     if (tem getToken() == -52)
424     {
425         for (int i = 0; i < tablaSimbolos.size(); i++)
426         {
427             if (tem getToken().equals(anObject: tablaSimbolos.get(index1).getNombre()))
428             {
429                 valor = tablaSimbolos.get(index1).getValor();
430             }
431         }
432     }
433 }
```

The screenshot shows a Java IDE interface with the file 'Simulación_Ejecución_VC1.java' open. The code is part of a parser or interpreter. It handles tokens and symbols from two lists: 'tem' and 'tem2'. It processes cases where tokens are -53 or -63, and it performs additions between tokens and symbols. The code uses nested loops to iterate through symbol tables and add results to a control stack.

```
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
```

This screenshot shows the same Java IDE interface with the same file open. The code is identical to the one in the first screenshot, but it includes additional logic at the bottom for handling case -25. The code continues to process tokens and symbols, adding them to a control stack based on specific token values (-53, -63) and symbol table indices.

```
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
```

The screenshot shows a Java code editor with the file "Simulación_Ejecución_VCI.java" open. The code is part of a switch statement handling tokens -51 and -61. It involves popping tokens from a stack, comparing them with entries in a symbol table, and calculating their sum. The code uses nested loops and conditional statements to find matching symbols and compute the result.

```
case "-51":  
    tem = pilaControl.pop();  
    tem2 = pilaControl.pop();  
  
    switch (tem.getToken())  
    {  
        case -51:  
        case -61:  
            if (tem.getNombre() == -51 && tem2.getNombre() == -51)  
            {  
                for (int i = 0; i < tablaSimbolos.size(); i++)  
                {  
                    if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))  
                    {  
                        valor = tablaSimbolos.get(index:i).getValor();  
                    }  
  
                    if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))  
                    {  
                        valor2 = tablaSimbolos.get(index:i).getValor();  
                    }  
                }  
  
                pilaControl.add(new Token("") + (Integer.parseInt("") + valor) - Integer.parseInt("") + valor), token: -61, posTabla: 0  
            }  
            else if (tem.getNombre() == -61 && tem2.getNombre() == -61)  
            {  
                pilaControl.add(new Token("") + (Integer.parseInt(:tem2.getNombre()) - Integer.parseInt(:tem.getNombre())), token: -61)  
            }  
            else  
            {  
                if (tem.getNombre() == -51)  
                {  
                    for (int i = 0; i < tablaSimbolos.size(); i++)  
                    {  
                        if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))  
                        {  
                            valor = tablaSimbolos.get(index:i).getValor();  
                        }  
                    }  
  
                    pilaControl.add(new Token("") + (Integer.parseInt("") + valor) - Integer.parseInt(:tem.getNombre()), token: -61)  
                }  
                break;  
            }  
        case -52:  
        case -62:  
            if (tem.getToken() == -52 && tem2.getToken() == -52)  
            {  
                for (int i = 0; i < tablaSimbolos.size(); i++)  
                {  
                    if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))  
                    {  
                        valor = tablaSimbolos.get(index:i).getValor();  
                    }  
  
                    if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))  
                    {  
                        valor2 = tablaSimbolos.get(index:i).getValor();  
                    }  
                }  
  
                pilaControl.add(new Token("") + (Integer.parseInt("") + valor) - Integer.parseInt(:tem.getNombre()), token: -61)  
            }  
            break;  
    }  
}
```

This screenshot continues the code from the previous one, focusing on the processing of tokens -52 and -62. It shows nested loops for comparing tokens against a symbol table and calculating their sum. The code includes several break statements to exit loops when matches are found.

```
case -52:  
case -62:  
    if (tem.getToken() == -52 && tem2.getToken() == -52)  
    {  
        for (int i = 0; i < tablaSimbolos.size(); i++)  
        {  
            if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))  
            {  
                valor = tablaSimbolos.get(index:i).getValor();  
            }  
  
            if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))  
            {  
                valor2 = tablaSimbolos.get(index:i).getValor();  
            }  
        }  
  
        pilaControl.add(new Token("") + (Integer.parseInt("") + valor) - Integer.parseInt(:tem.getNombre()), token: -61)  
    }  
    break;  
}
```

The screenshot shows a Java code editor with the file "Simulación_Ejecución_VCI.java" open. The code is part of a parser, specifically handling arithmetic operations involving tokens for subtraction. It uses a stack of tokens and a symbol table to resolve identifiers.

```
578     }
579     pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) - Double.parseDouble("") + valor), token: -62, posTablaSimbolos: 0
580   }
581   else if (tem getToken() == -62 && tem2 getToken() == -62)
582   {
583     pilaControl.add(new Token("") + (Double.parseDouble(tablaSimbolos.get(index).getNombre()) - Double.parseDouble(tablaSimbolos.get(index).getNombre())), token: -62, posTablaSimbolos: 0)
584   }
585   else
586   {
587     if (tem getToken() == -52)
588     {
589       for (int i = 0; i < tablaSimbolos.size(); i++)
590       {
591         if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
592         {
593           valor = tablaSimbolos.get(index:i).getValor();
594         }
595       }
596 
597       pilaControl.add(new Token("") + (Double.parseDouble(tablaSimbolos.get(index).getNombre()) - Double.parseDouble("") + valor)), token: -62, posTablaSimbolos: 0
598     }
599     else
600     {
601       for (int i = 0; i < tablaSimbolos.size(); i++)
602       {
603         if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
604         {
605           valor2 = tablaSimbolos.get(index:i).getValor();
606         }
607       }
608 
609       pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) - Double.parseDouble(tablaSimbolos.get(index).getNombre())), token: -62, posTablaSimbolos: 0
610     }
611   }
612   break;
613 }
```

The screenshot shows the continuation of the Java code editor for the same file. The code handles cases for tokens -26 and -31, involving popping from stacks and updating values in a symbol table.

```
613   }
614   break;
615   // caso -
616   case "-26":
617   {
618     tem = pilaControl.pop();
619     Token guardar = pilaControl.pop();
620 
621     for (int i = 0; i < tablaSimbolos.size(); i++)
622     {
623       if (guardar.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
624       {
625         tablaSimbolos.get(index:i).setValor((tem.getNombre()));
626       }
627     }
628     //System.out.println(tablaSimbolos.get(0).toString());
629     break;
630   // caso -
631   case "-31":
632   {
633     tem = pilaControl.pop();
634     tem2 = pilaControl.pop();
635 
636     switch (tem.getToken())
637     {
638       case -51:
639       case -61:
640       if (tem.getToken() == -51 && tem2.getToken() == -51)
641       {
642         for (int i = 0; i < tablaSimbolos.size(); i++)
643         {
644           if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
645           {
646             valor = tablaSimbolos.get(index:i).getValor();
647           }
648 
649           if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
650           {
651             valor2 = tablaSimbolos.get(index:i).getValor();
652           }
653         }
654       }
655     }
656   }
657 }
```

The screenshot shows a Java code editor with the file "Simulación_Ejecución_VCI.java" open. The code is part of a parser or interpreter for arithmetic expressions. It handles operations like addition (+) and subtraction (-) between integers. The code uses a stack (pilaControl) to store tokens and a symbol table (tablaSimbolos) to resolve identifiers. The code is annotated with line numbers from 645 to 685.

```
645     valor2 = tablaSimbolos.get(index).getValor();
646 }
647
648     pilaControl.add(new Token("") + (Integer.parseInt("") + valor2) < Integer.parseInt("") + valor), token: -64, posTabla: 0
649 }
650 else if (tem getToken() == -61 && tem2 getToken() == -61)
651 {
652     pilaControl.add(new Token("") + (Integer.parseInt(:tem2.getNombre()) < Integer.parseInt(:tem.getNombre())), token: -64, posTabla: 0
653 }
654 else
655 {
656     if (tem getToken() == -51)
657     {
658         for (int i = 0; i < tablaSimbolos.size(); i++)
659         {
660             if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
661             {
662                 valor = tablaSimbolos.get(index:i).getValor();
663             }
664         }
665
666         pilaControl.add(new Token("") + (Integer.parseInt(:tem2.getNombre()) < Integer.parseInt("") + valor)), token: -64, posTabla: 0
667     }
668     else
669     {
670         for (int i = 0; i < tablaSimbolos.size(); i++)
671         {
672             if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
673             {
674                 valor2 = tablaSimbolos.get(index:i).getValor();
675             }
676         }
677
678         pilaControl.add(new Token("") + (Integer.parseInt("") + valor2) < Integer.parseInt(:tem.getNombre())), token: -64, posTabla: 0
679     }
680 }
681
682 }
683
684 }
685 }
```

The screenshot shows a Java code editor with the file "Simulación_Ejecución_VCI.java" open. This section of the code handles floating-point arithmetic, specifically addition (+) and subtraction (-) of doubles. It uses a stack (pilaControl) and a symbol table (tablaSimbolos). The code is annotated with line numbers from 685 to 721.

```
685     }
686     break;
687 case -52:
688 case -62:
689     if (tem getToken() == -52 && tem2 getToken() == -52)
690     {
691         for (int i = 0; i < tablaSimbolos.size(); i++)
692         {
693             if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
694             {
695                 valor = tablaSimbolos.get(index:i).getValor();
696             }
697
698             if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
699             {
700                 valor2 = tablaSimbolos.get(index:i).getValor();
701             }
702         }
703
704         pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) < Double.parseDouble("") + valor)), token: -64, posTabla: 0
705     }
706     else if (tem getToken() == -62 && tem2 getToken() == -62)
707     {
708         pilaControl.add(new Token("") + (Double.parseDouble(:tem2.getNombre()) < Double.parseDouble(:tem.getNombre())), token: -64, posTabla: 0
709     }
710     else
711     {
712         if (tem getToken() == -52)
713         {
714             for (int i = 0; i < tablaSimbolos.size(); i++)
715             {
716                 if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
717                 {
718                     valor = tablaSimbolos.get(index:i).getValor();
719                 }
720             }
721         }
722     }
723 }
```

The screenshot shows a Java code editor with the file "Simulación_Ejecución_VCI.java" open. The code implements a comparison operation between two tokens. It uses a stack of tokens (pilaControl) and a symbol table (tablaSimbolos). The logic involves popping tokens from the stack, comparing their names, and then checking if they are numbers. If they are, it converts them to doubles and compares them. The code handles various cases including less than or equal to, greater than or equal to, and less than. It also includes a break statement and a switch block for handling specific token values like -51 and -61.

```
721     pilaControl.add(new Token("") + (Double.parseDouble(:tem2.getNombre()) < Double.parseDouble("") + valor)), token: -64, posTabla: 0
722 }
723 else
724 {
725     for (int i = 0; i < tablaSimbolos.size(); i++)
726     {
727         if (tem2.getNombre().equals( anObject: tablaSimbolos.get(index:i).getNombre()))
728         {
729             valor2 = tablaSimbolos.get(index:i).getValor();
730         }
731     }
732 }
733 pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) < Double.parseDouble(: tem.getNombre())), token: -64, posTabla: 0
734 }
735 }
736 }
737 }
738 }
739 }
740 // caso <
741 case "-32":
742     tem = pilaControl.pop();
743     tem2 = pilaControl.pop();
744
745     switch (tem.getToken())
746     {
747         case -51:
748         case -61:
749             if (tem.getToken() == -51 && tem2.getToken() == -51)
750             {
751                 for (int i = 0; i < tablaSimbolos.size(); i++)
752                 {
753                     if (tem.getNombre().equals( anObject: tablaSimbolos.get(index:i).getNombre()))
754                     {
755                         valor = tablaSimbolos.get(index:i).getValor();
756                     }
757                 }
758             }
759         }
760     }
761 }
```

The screenshot shows a Java code editor with the same file "Simulación_Ejecución_VCI.java" open. This version of the code handles integer comparisons. It follows a similar structure to the previous one but uses Integer.parseInt instead of Double.parseDouble. It checks for less than or equal to, greater than or equal to, and less than conditions, and includes logic for specific tokens like -51 and -61.

```
757     if (tem2.getNombre().equals( anObject: tablaSimbolos.get(index:i).getNombre()))
758     {
759         valor2 = tablaSimbolos.get(index:i).getValor();
760     }
761 }
762 }
763 pilaControl.add(new Token("") + (Integer.parseInt("") + valor2) <= Integer.parseInt("") + valor), token: -64, posTabla: 0
764 }
765 else if (tem.getToken() == -61 && tem2.getToken() == -61)
766 {
767     pilaControl.add(new Token("") + (Integer.parseInt(:tem2.getNombre()) <= Integer.parseInt(: tem.getNombre()))), token: -64, posTabla: 0
768 }
769 }
770 else
771 {
772     if (tem.getToken() == -51)
773     {
774         for (int i = 0; i < tablaSimbolos.size(); i++)
775         {
776             if (tem.getNombre().equals( anObject: tablaSimbolos.get(index:i).getNombre()))
777             {
778                 valor = tablaSimbolos.get(index:i).getValor();
779             }
780         }
781
782     pilaControl.add(new Token("") + (Integer.parseInt(:tem2.getNombre()) <= Integer.parseInt("") + valor)), token: -64, posTabla: 0
783 }
784 else
785 {
786     for (int i = 0; i < tablaSimbolos.size(); i++)
787     {
788         if (tem2.getNombre().equals( anObject: tablaSimbolos.get(index:i).getNombre()))
789         {
790             valor2 = tablaSimbolos.get(index:i).getValor();
791         }
792     }
793 }
```

The screenshot shows a Java IDE interface with the title "Start Page" and "Simulación_Ejecución_VCI.java". The code is a switch statement handling token values. It includes logic for adding tokens to a stack based on comparison operators and variable assignments.

```
794         pilaControl.add(new Token(""+ (Integer.parseInt("") + valor2) <= Integer.parseInt(:tem.getNombre()), token: -64, pos: 0));
795     }
796     break;
797   case -52:
798   case -62:
799     if (tem.getToken() == -52 && tem2.getToken() == -52)
800     {
801       for (int i = 0; i < tablaSimbolos.size(); i++)
802       {
803         if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
804         {
805           valor = tablaSimbolos.get(index:i).getValor();
806         }
807         if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
808         {
809           valor2 = tablaSimbolos.get(index:i).getValor();
810         }
811       }
812     }
813     pilaControl.add(new Token(""+ (Double.parseDouble("") + valor2) <= Double.parseDouble("") + valor), token: -64, pos: 0);
814   }
815   else if (tem.getToken() == -62 && tem2.getToken() == -62)
816   {
817     pilaControl.add(new Token(""+ (Double.parseDouble(:tem2.getNombre()) <= Double.parseDouble(:tem.getNombre())), token: -64, pos: 0));
818   }
819   else
820   {
821     if (tem.getToken() == -52)
822     {
823       for (int i = 0; i < tablaSimbolos.size(); i++)
824       {
825         if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
826         {
827           valor = tablaSimbolos.get(index:i).getValor();
828         }
829       }
830     }
831   }
832 }
```

The screenshot shows a continuation of the Java IDE interface with the same file name "Simulación_Ejecución_VCI.java". The code block continues from the previous one, showing more cases of the switch statement and additional logic for handling tokens and symbol tables.

```
829     valor = tablaSimbolos.get(index:i).getValor();
830   }
831 }
832 pilaControl.add(new Token(""+ (Double.parseDouble(:tem2.getNombre()) <= Double.parseDouble("") + valor)), token: -64, pos: 0);
833 }
834 else
835 {
836   for (int i = 0; i < tablaSimbolos.size(); i++)
837   {
838     if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
839     {
840       valor2 = tablaSimbolos.get(index:i).getValor();
841     }
842   }
843   pilaControl.add(new Token(""+ (Double.parseDouble("") + valor2) <= Double.parseDouble(:tem.getNombre())), token: -64, pos: 0);
844 }
845 }
846 }
847 }
848 break;
849 }
850 // caso >
851 case "-33":
852   tem = pilaControl.pop();
853   tem2 = pilaControl.pop();
854 
855   switch (tem.getToken())
856   {
857     case -51:
858     case -61:
859       if (tem.getToken() == -51 && tem2.getToken() == -51)
860       {
861         for (int i = 0; i < tablaSimbolos.size(); i++)
862         {
863           if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
864 }
```

The screenshot shows a Java IDE interface with the title "Start Page" and "Simulación_Ejecución_VCI.java". The code is a part of a simulation for VCI (Virtual Computer Interpreters). It deals with symbol tables and tokens. The code includes logic for comparing token values, adding tokens to a control stack, and handling specific token types like -61 and -62.

```
865     if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
866     {
867         valor = tablaSimbolos.get(index:i).getValor();
868     }
869
870     if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
871     {
872         valor2 = tablaSimbolos.get(index:i).getValor();
873     }
874
875     pilaControl.add(new Token("") + (Integer.parseInt("") + valor2) > Integer.parseInt("") + valor), token: -64, posTabla: 0;
876 }
877 else if (tem.getToken() == -61 && tem2.getToken() == -61)
878 {
879     pilaControl.add(new Token("") + (Integer.parseInt(:tem2.getNombre()) > Integer.parseInt(:tem.getNombre())), token: -64, posTabla: 0);
880 }
881 else
882 {
883     if (tem.getToken() == -51)
884     {
885         for (int i = 0; i < tablaSimbolos.size(); i++)
886         {
887             if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
888             {
889                 valor = tablaSimbolos.get(index:i).getValor();
890             }
891         }
892
893         pilaControl.add(new Token("") + (Integer.parseInt(:tem2.getNombre()) > Integer.parseInt("") + valor)), token: -64, posTabla: 0;
894     }
895     else
896     {
897         for (int i = 0; i < tablaSimbolos.size(); i++)
898         {
899             if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
900             {
901             }
902         }
903     }
904 }
905 }
906 }
907 }
908 }
909 break;
910 case -52:
911 case -62:
912 if (tem.getToken() == -52 && tem2.getToken() == -52)
913 {
914     for (int i = 0; i < tablaSimbolos.size(); i++)
915     {
916         if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
917         {
918             valor = tablaSimbolos.get(index:i).getValor();
919         }
920
921         if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
922         {
923             valor2 = tablaSimbolos.get(index:i).getValor();
924         }
925     }
926
927     pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) > Double.parseDouble("") + valor), token: -64, posTabla: 0;
928 }
929 else if (tem.getToken() == -62 && tem2.getToken() == -62)
930 {
931     pilaControl.add(new Token("") + (Double.parseDouble(:tem2.getNombre()) > Double.parseDouble(:tem.getNombre()))), token: -64, posTabla: 0;
932 }
933 else
934 {
935     if (tem.getToken() == -52)
936     {
937         for (int i = 0; i < tablaSimbolos.size(); i++)
938         {
939             if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
940             {
941                 valor = tablaSimbolos.get(index:i).getValor();
942             }
943
944             if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
945             {
946                 valor2 = tablaSimbolos.get(index:i).getValor();
947             }
948
949             pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) > Double.parseDouble("") + valor), token: -64, posTabla: 0;
950         }
951     }
952 }
```

This screenshot shows the continuation of the Java code from the previous one. It handles cases for tokens -52 and -62, specifically comparing double values. The code uses nested loops to iterate through the symbol table and adds tokens to the control stack based on the comparison results.

```
901     if (tem.getToken() == -52 && tem2.getToken() == -52)
902     {
903         for (int i = 0; i < tablaSimbolos.size(); i++)
904         {
905             if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
906             {
907                 valor = tablaSimbolos.get(index:i).getValor();
908             }
909
910             if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
911             {
912                 valor2 = tablaSimbolos.get(index:i).getValor();
913             }
914
915             pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) > Double.parseDouble("") + valor), token: -64, posTabla: 0;
916         }
917     }
918 else if (tem.getToken() == -62 && tem2.getToken() == -62)
919 {
920     pilaControl.add(new Token("") + (Double.parseDouble(:tem2.getNombre()) > Double.parseDouble(:tem.getNombre()))), token: -64, posTabla: 0;
921 }
922 else
923 {
924     if (tem.getToken() == -52)
925     {
926         for (int i = 0; i < tablaSimbolos.size(); i++)
927         {
928             if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
929             {
930                 valor = tablaSimbolos.get(index:i).getValor();
931             }
932
933             if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
934             {
935                 valor2 = tablaSimbolos.get(index:i).getValor();
936             }
937
938             pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) > Double.parseDouble("") + valor), token: -64, posTabla: 0;
939         }
940     }
941 }
```

The screenshot shows a Java IDE interface with the title "Start Page" and "Simulación_Ejecución_VCI.java". The code is a part of a parser or interpreter for a simulation language. It uses a stack (pilaControl) and a symbol table (tablaSimbolos) to handle tokens and variables. The code handles comparisons between floating-point numbers and integer values, and it processes specific tokens like -51 and -61.

```
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
```

This screenshot continues the code from the previous one. It includes more logic for handling integer comparisons and variable lookups. The code uses the same structures (pilaControl and tablaSimbolos) and token types (-51, -61).

```
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
```

```
1009     for (int i = 0; i < tablaSimbolos.size(); i++)
1010     {
1011         if (tem2.getNombre().equals( anObject: tablaSimbolos.get( index:i ).getNombre() ))
1012         {
1013             valor2 = tablaSimbolos.get( index:i ).getValor();
1014         }
1015     }
1016     pilaControl.add(new Token(""+ (Integer.parseInt("") + valor2) >= Integer.parseInt(:tem.getNombre())), token: -64, posD: 1);
1017 }
1018 }
1019 }
1020 break;
1021 case -52:
1022 case -62:
1023 if (tem.getToken() == -52 && tem2.getToken() == -52)
1024 {
1025     for (int i = 0; i < tablaSimbolos.size(); i++)
1026     {
1027         if (tem.getNombre().equals( anObject: tablaSimbolos.get( index:i ).getNombre() ))
1028         {
1029             valor = tablaSimbolos.get( index:i ).getValor();
1030         }
1031         if (tem2.getNombre().equals( anObject: tablaSimbolos.get( index:i ).getNombre() ))
1032         {
1033             valor2 = tablaSimbolos.get( index:i ).getValor();
1034         }
1035     }
1036     pilaControl.add(new Token(""+ (Double.parseDouble("") + valor2) >= Double.parseDouble("") + valor)), token: -64, posD: 1);
1037 }
1038 else if (tem.getToken() == -62 && tem2.getToken() == -62)
1039 {
1040     pilaControl.add(new Token(""+ (Double.parseDouble(:tem2.getNombre()) >= Double.parseDouble(:tem.getNombre()))),
1041 );
1042 }
1043 else
1044 {
1045 }
```

```
1045 }
1046 if (tem.getToken() == -52)
1047 {
1048     for (int i = 0; i < tablaSimbolos.size(); i++)
1049     {
1050         if (tem.getNombre().equals( anObject: tablaSimbolos.get( index:i ).getNombre() ))
1051         {
1052             valor = tablaSimbolos.get( index:i ).getValor();
1053         }
1054     }
1055     pilaControl.add(new Token(""+ (Double.parseDouble(:tem2.getNombre()) >= Double.parseDouble("") + valor)), token: -64, posD: 1);
1056 }
1057 else
1058 {
1059     for (int i = 0; i < tablaSimbolos.size(); i++)
1060     {
1061         if (tem2.getNombre().equals( anObject: tablaSimbolos.get( index:i ).getNombre() ))
1062         {
1063             valor2 = tablaSimbolos.get( index:i ).getValor();
1064         }
1065     }
1066     pilaControl.add(new Token(""+ (Double.parseDouble("") + valor2) >= Double.parseDouble(:tem.getNombre())), token: -64, posD: 1);
1067 }
1068 }
1069 }
1070 }
1071 }
1072 break;
1073 }
1074 // caso ==
1075 case "-35":
1076     tem = pilaControl.pop();
1077     tem2 = pilaControl.pop();
1078     switch (tem.getToken())
1079     {
1080         case -51:
```

The screenshot shows a Java IDE interface with a code editor containing Java code. The code is part of a class named 'Simulación_Ejecución_VCI.java'. The code handles token processing, specifically for tokens -51 and -61. It uses a 'tablaSimbolos' table to map symbols to their values. For token -51, it checks if both tokens are -51 and if they are, it adds a new token to the control stack with value 0. For token -61, it checks if both tokens are -61 and if they are, it adds a new token to the control stack with value 1. The code also includes logic for tokens -52 and -62.

```
1081
1082
1083     case -51:
1084     case -61:
1085         if (tem getToken() == -51 && tem2 getToken() == -51)
1086         {
1087             for (int i = 0; i < tablaSimbolos.size(); i++)
1088             {
1089                 if (tem getNombre().equals(anObject: tablaSimbolos.get(index: i).getNombre()))
1090                 {
1091                     valor = tablaSimbolos.get(index: i).getValor();
1092
1093                     if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index: i).getNombre()))
1094                     {
1095                         valor2 = tablaSimbolos.get(index: i).getValor();
1096                     }
1097
1098                     pilaControl.add(new Token("") + (Integer.parseInt("") + valor2) == Integer.parseInt("") + valor), token: -64, postTable: 0
1099                 }
1100             else if (tem getToken() == -61 && tem2 getToken() == -61)
1101             {
1102                 pilaControl.add(new Token("") + (Integer.parseInt(: tem2.getNombre()) == Integer.parseInt(: tem.getNombre())), token: -64, postTable: 1
1103             }
1104         }
1105     }
1106
1107     if (tem getToken() == -51)
1108     {
1109         for (int i = 0; i < tablaSimbolos.size(); i++)
1110         {
1111             if (tem getNombre().equals(anObject: tablaSimbolos.get(index: i).getNombre()))
1112             {
1113                 valor = tablaSimbolos.get(index: i).getValor();
1114             }
1115         }
1116
1117         pilaControl.add(new Token("") + (Integer.parseInt(: tem2.getNombre()) == Integer.parseInt("") + valor)), token: -64, postTable: 0
1118     }
1119
1120 }
```

This screenshot continues the Java code from the previous one. It handles tokens -52 and -62. For token -52, it checks if both tokens are -52 and if they are, it adds a new token to the control stack with value 0. For token -62, it checks if both tokens are -62 and if they are, it adds a new token to the control stack with value 1. The code also includes logic for tokens -51 and -61.

```
1117
1118
1119
1120
1121
1122     for (int i = 0; i < tablaSimbolos.size(); i++)
1123     {
1124         if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index: i).getNombre()))
1125         {
1126             valor2 = tablaSimbolos.get(index: i).getValor();
1127         }
1128
1129         pilaControl.add(new Token("") + (Integer.parseInt("") + valor2) == Integer.parseInt(: tem.getNombre()), token: -64, postTable: 0
1130     }
1131     break;
1132 case -52:
1133 case -62:
1134     if (tem getToken() == -52 && tem2 getToken() == -52)
1135     {
1136         for (int i = 0; i < tablaSimbolos.size(); i++)
1137         {
1138             if (tem getNombre().equals(anObject: tablaSimbolos.get(index: i).getNombre()))
1139             {
1140                 valor = tablaSimbolos.get(index: i).getValor();
1141
1142                 if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index: i).getNombre()))
1143                 {
1144                     valor2 = tablaSimbolos.get(index: i).getValor();
1145
1146                     pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) == Double.parseDouble("") + valor), token: -64, postTable: 0
1147                 }
1148
1149             }
1150     }
1151     else if (tem getToken() == -62 && tem2 getToken() == -62)
1152     {
1153         pilaControl.add(new Token("") + (Double.parseDouble(: tem2.getNombre()) == Double.parseDouble(: tem.getNombre()))), token: -64, postTable: 1
1154     }
1155
1156 }
```

The screenshot shows a Java code editor with the following code snippet:

```
1153     pilaControl.add(new Token("") + (Double.parseDouble(:tem2.getNombre()) == Double.parseDouble(:tem.getNombre())));
1154     }
1155     else
1156     {
1157         if (tem.getToken() == -52)
1158         {
1159             for (int i = 0; i < tablaSimbolos.size(); i++)
1160             {
1161                 if (tem.getNombre().equals(:anObject: tablaSimbolos.get(index:i).getNombre()))
1162                 {
1163                     valor = tablaSimbolos.get(index:i).getValor();
1164                 }
1165             }
1166             pilaControl.add(new Token("") + (Double.parseDouble(:tem2.getNombre()) == Double.parseDouble("") + valor)), token:-64, posTabla:0, linea:0);
1167         }
1168         else
1169         {
1170             for (int i = 0; i < tablaSimbolos.size(); i++)
1171             {
1172                 if (tem2.getNombre().equals(:anObject: tablaSimbolos.get(index:i).getNombre()))
1173                 {
1174                     valor2 = tablaSimbolos.get(index:i).getValor();
1175                 }
1176             }
1177             pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) == Double.parseDouble(:tem.getNombre())), token:-64, posTabla:0, linea:0);
1178         }
1179     }
1180     }
1181     break;
1182 case -54:
1183 case -64:
1184     if (tem.getToken() == -54 && tem2.getToken() == -54)
1185     {
1186         for (int i = 0; i < tablaSimbolos.size(); i++)
1187         {
1188             if (tem.getNombre().equals(:anObject: tablaSimbolos.get(index:i).getNombre()))
1189             {
1190                 if (tem.getNombre().equals(:anObject: tablaSimbolos.get(index:i).getNombre()))
1191                 {
1192                     valor = tablaSimbolos.get(index:i).getValor();
1193                 }
1194                 if (tem2.getNombre().equals(:anObject: tablaSimbolos.get(index:i).getNombre()))
1195                 {
1196                     valor2 = tablaSimbolos.get(index:i).getValor();
1197                 }
1198             }
1199             pilaControl.add(new Token("") + (valor2.equals(obj:valor)), token:-64, posTabla:0, linea:0));
1200         }
1201     }
1202     else if (tem.getToken() == -64 && tem2.getToken() == -64)
1203     {
1204         pilaControl.add(new Token("") + ((tem2.getNombre().equals(:anObject: tem.getNombre()))), token:-64, posTabla:0, linea:0);
1205     }
1206     else
1207     {
1208         if (tem.getToken() == -54)
1209         {
1210             for (int i = 0; i < tablaSimbolos.size(); i++)
1211             {
1212                 if (tem.getNombre().equals(:anObject: tablaSimbolos.get(index:i).getNombre()))
1213                 {
1214                     valor = tablaSimbolos.get(index:i).getValor();
1215                 }
1216             }
1217             pilaControl.add(new Token("") + ((tem2.getNombre().equals(:anObject: valor))), token:-64, posTabla:0, linea:0));
1218         }
1219         else
1220         {
1221             for (int i = 0; i < tablaSimbolos.size(); i++)
1222             {
1223                 if (tem2.getNombre().equals(:anObject: tablaSimbolos.get(index:i).getNombre()))
1224                 {
1225
```

The screenshot shows a Java code editor with the following code snippet, continuing from the previous one:

```
1189                 }
1190             }
1191             valor = tablaSimbolos.get(index:i).getValor();
1192         }
1193         if (tem2.getNombre().equals(:anObject: tablaSimbolos.get(index:i).getNombre()))
1194         {
1195             valor2 = tablaSimbolos.get(index:i).getValor();
1196         }
1197     }
1198     }
1199     pilaControl.add(new Token("") + (valor2.equals(obj:valor)), token:-64, posTabla:0, linea:0));
1200 }
1201 else if (tem.getToken() == -64 && tem2.getToken() == -64)
1202 {
1203     pilaControl.add(new Token("") + ((tem2.getNombre().equals(:anObject: tem.getNombre()))), token:-64, posTabla:0, linea:0);
1204 }
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
```

The screenshot shows a Java IDE interface with a code editor window titled "Start Page" and "Simulación_Ejecución_VCI.java". The code is a snippet of Java code dealing with symbol tables and stacks. The code includes several loops, conditionals, and method calls. The code editor has syntax highlighting and various toolbars at the top.

```
1225     valor = tablaSimbolos.get(index:i).getValor();
1226     )
1227   }
1228   }
1229   }
1230   }
1231   }
1232   }
1233   }
1234   }
1235   }
1236   }
1237   }
1238   }
1239   }
1240   }
1241   }
1242   }
1243   }
1244   }
1245   }
1246   }
1247   }
1248   }
1249   }
1250   }
1251   }
1252   }
1253   }
1254   }
1255   }
1256   }
1257   }
1258   }
1259   }
1260   }
1261   }
1262   }
1263   }
1264   }
1265   }
1266   }
1267   }
1268   }
1269   }
1270   }
1271   }
1272   }
1273   }
1274   }
1275   }
1276   }
1277   }
1278   }
1279   }
1280   }
1281   }
1282   }
1283   }
1284   }
1285   }
1286   }
1287   }
1288   }
1289   }
1290   }
1291   }
1292   }
1293   }
1294   }
1295   }
1296   }
1297   }
```

The screenshot shows a Java IDE interface with a code editor window titled "Start Page" and "Simulación_Ejecución_VCI.java". The code is a continuation of the previous snippet, showing more of the same logic involving symbol tables and stacks. The code editor has syntax highlighting and various toolbars at the top.

```
1261   }
1262   }
1263   }
1264   }
1265   }
1266   }
1267   }
1268   }
1269   }
1270   }
1271   }
1272   }
1273   }
1274   }
1275   }
1276   }
1277   }
1278   }
1279   }
1280   }
1281   }
1282   }
1283   }
1284   }
1285   }
1286   }
1287   }
1288   }
1289   }
1290   }
1291   }
1292   }
1293   }
1294   }
1295   }
1296   }
1297   }
```

The screenshot shows a Java IDE interface with the title "Start Page" and "Simulación_Ejecución_VCI.java". The code editor displays lines 1297 to 333 of a Java file. The code involves a loop where it compares the names of objects in a symbol table. If the names match, it retrieves their values and adds tokens to a control stack. The code handles various token types, including numbers and identifiers. The interface includes standard toolbars and a status bar at the bottom.

```
1297     {
1298         for (int i = 0; i < tablaSimbolos.size(); i++)
1299         {
1300             if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
1301             {
1302                 valor = tablaSimbolos.get(index:i).getValor();
1303             }
1304
1305             if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
1306             {
1307                 valor2 = tablaSimbolos.get(index:i).getValor();
1308             }
1309         }
1310
1311         pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) != Double.parseDouble("") + valor), token:-64, posTable:0, linea:0);
1312     }
1313     else if (tem getToken() == -62 && tem2.getToken() == -62)
1314     {
1315         pilaControl.add(new Token("") + (Double.parseDouble(:tem2.getNombre()) != Double.parseDouble(:tem.getNombre())));
1316     }
1317     else
1318     {
1319         if (tem.getToken() == -52)
1320         {
1321             for (int i = 0; i < tablaSimbolos.size(); i++)
1322             {
1323                 if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
1324                 {
1325                     valor = tablaSimbolos.get(index:i).getValor();
1326                 }
1327             }
1328
1329             pilaControl.add(new Token("") + (Double.parseDouble(:tem2.getNombre()) != Double.parseDouble("") + valor)), token:-64, posTable:0, linea:0);
1330         }
1331         else
1332         {
1333             for (int i = 0; i < tablaSimbolos.size(); i++)
```

This screenshot shows the same Java IDE interface as the first one, but with a different portion of the code visible. Lines 1333 to 369 are shown. The code continues the comparison of object names in the symbol table. It handles cases where tokens are -54 or -64, and it adds tokens to the control stack based on the comparison results. The interface remains consistent with the first screenshot.

```
1333     {
1334         for (int i = 0; i < tablaSimbolos.size(); i++)
1335         {
1336             if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
1337             {
1338                 valor2 = tablaSimbolos.get(index:i).getValor();
1339             }
1340         }
1341         pilaControl.add(new Token("") + (Double.parseDouble("") + valor2) != Double.parseDouble(:tem.getNombre())), token:-64, posTable:0, linea:0);
1342     }
1343     break;
1344 case -54:
1345 case -64:
1346     if (tem.getToken() == -54 && tem2.getToken() == -54)
1347     {
1348         for (int i = 0; i < tablaSimbolos.size(); i++)
1349         {
1350             if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
1351             {
1352                 valor = tablaSimbolos.get(index:i).getValor();
1353             }
1354
1355             if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
1356             {
1357                 valor2 = tablaSimbolos.get(index:i).getValor();
1358             }
1359         }
1360
1361         pilaControl.add(new Token("") + (!valor2.equals(obj.valor)), token:-64, posTable:0, linea:0));
1362     }
1363     else if (tem.getToken() == -64 && tem2.getToken() == -64)
1364     {
1365         pilaControl.add(new Token("") + (!(tem2.getNombre().equals(anObject: tem.getNombre()))), token:-64, posTable:0, linea:0));
1366     }
1367     else
1368     {
```

```
1369     if (tem getToken() == -54)
1370     {
1371         for (int i = 0; i < tablaSimbolos.size(); i++)
1372         {
1373             if (tem.getNombre().equals(anObject: tablaSimbolos.get(index: i).getNombre()))
1374             {
1375                 valor = tablaSimbolos.get(index: i).getValor();
1376             }
1377         }
1378
1379         pilaControl.add(new Token("") + (!tem.getNombre().equals(anObject: valor))), token: -64, posTabla: 0, linea: 0));
1380     }
1381     else
1382     {
1383         for (int i = 0; i < tablaSimbolos.size(); i++)
1384         {
1385             if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index: i).getNombre()))
1386             {
1387                 valor2 = tablaSimbolos.get(index: i).getValor();
1388             }
1389         }
1390
1391         pilaControl.add(new Token("") + (!valor2.equals(obj: tem.getNombre()))), token: -64, posTabla: 0, linea: 0));
1392     }
1393     break;
1394 }
1395 // caso 44
1396 case "-41":
1397     tem = pilaControl.pop();
1398     tem2 = pilaControl.pop();
1399
1400     switch (tem.getToken())
1401     {
1402         case -54:
```

```
1403         case -64:
1404             if (tem.getToken() == -54 && tem2.getToken() == -54)
1405             {
1406                 for (int i = 0; i < tablaSimbolos.size(); i++)
1407                 {
1408                     if (tem.getNombre().equals(anObject: tablaSimbolos.get(index: i).getNombre()))
1409                     {
1410                         valor = tablaSimbolos.get(index: i).getValor();
1411
1412                         if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index: i).getNombre()))
1413                         {
1414                             valor2 = tablaSimbolos.get(index: i).getValor();
1415                         }
1416
1417                         if (valor2.equals(obj: "true") && valor.equals(obj: "true"))
1418                         {
1419                             pilaControl.add(new Token(nombre: "true", token: -64, posTabla: 0, linea: 0));
1420                         }
1421                         else
1422                         {
1423                             pilaControl.add(new Token(nombre: "false", token: -64, posTabla: 0, linea: 0));
1424                         }
1425
1426                     }
1427                 }
1428             }
1429             else if (tem.getToken() == -64 && tem2.getToken() == -64)
1430             {
1431                 if (tem2.getNombre().equals(anObject: "true") && tem.getNombre().equals(anObject: "true"))
1432                 {
1433                     pilaControl.add(new Token(nombre: "true", token: -64, posTabla: 0, linea: 0));
1434
1435                 }
1436                 else
1437                 {
1438                     pilaControl.add(new Token(nombre: "false", token: -64, posTabla: 0, linea: 0));
1439                 }
1440             }
1441 }
```

The screenshot shows a Java IDE interface with the file 'Simulación_Ejecución_VCI.java' open. The code is part of a switch statement handling token values -54 and -64. It involves comparing tokens against entries in a 'tablaSimbolos' list and adding tokens to a 'pilaControl' stack based on equality.

```
1441
1442
1443
1444     else
1445     {
1446         if (tem getToken() == -54)
1447         {
1448             for (int i = 0; i < tablaSimbolos.size(); i++)
1449             {
1450                 if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
1451                 {
1452                     valor = tablaSimbolos.get(index:i).getValor();
1453                 }
1454             }
1455
1456             if (tem2.getNombre().equals(subject: "true") && valor.equals(obj: "true"))
1457             {
1458                 pilaControl.add(new Token(nombre: "true", token: -64, posTabla: 0, linea: 0));
1459             }
1460             else
1461             {
1462                 pilaControl.add(new Token(nombre: "false", token: -64, posTabla: 0, linea: 0));
1463             }
1464         }
1465
1466         else
1467         {
1468             for (int i = 0; i < tablaSimbolos.size(); i++)
1469             {
1470                 if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
1471                 {
1472                     valor2 = tablaSimbolos.get(index:i).getValor();
1473                 }
1474             }
1475
1476             if (valor2.equals(obj: "true") && tem.getToken().equals(anObject: "true"))
1477             {
1478                 pilaControl.add(new Token(nombre: "true", token: -64, posTabla: 0, linea: 0));
1479             }
1480             else
1481             {
1482                 pilaControl.add(new Token(nombre: "false", token: -64, posTabla: 0, linea: 0));
1483             }
1484         }
1485     }
1486
1487     break;
1488 // caso []
1489 case "-42":
1490     tem = pilaControl.pop();
1491     tem2 = pilaControl.pop();
1492
1493     switch (tem.getToken())
1494     {
1495         case -54:
1496         case -64:
1497             if (tem.getToken() == -54 && tem2.getToken() == -54)
1498             {
1499                 for (int i = 0; i < tablaSimbolos.size(); i++)
1500                 {
1501                     if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
1502                     {
1503                         valor = tablaSimbolos.get(index:i).getValor();
1504
1505                         if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
1506                         {
1507                             valor2 = tablaSimbolos.get(index:i).getValor();
1508                         }
1509
1510                         if (valor2.equals(obj: "true") || valor.equals(obj: "true"))
1511                         {
1512                             pilaControl.add(new Token(nombre: "true", token: -64, posTabla: 0, linea: 0));
1513                         }
1514                     }
1515                 }
1516             }
1517         }
1518     }
1519 }
```

The screenshot shows a Java IDE interface with the file 'Simulación_Ejecución_VCI.java' open. This section of the code handles the case for token value -42. It performs a stack pop operation, then enters a switch block for tokens -54 and -64. Inside this block, it compares tokens against entries in a 'tablaSimbolos' list and adds tokens to a 'pilaControl' stack based on equality.

```
1477
1478
1479     else
1480     {
1481         pilaControl.add(new Token(nombre: "false", token: -64, posTabla: 0, linea: 0));
1482     }
1483
1484     break;
1485 // caso []
1486 case "-42":
1487     tem = pilaControl.pop();
1488     tem2 = pilaControl.pop();
1489
1490     switch (tem.getToken())
1491     {
1492         case -54:
1493         case -64:
1494             if (tem.getToken() == -54 && tem2.getToken() == -54)
1495             {
1496                 for (int i = 0; i < tablaSimbolos.size(); i++)
1497                 {
1498                     if (tem.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
1499                     {
1500                         valor = tablaSimbolos.get(index:i).getValor();
1501
1502                         if (tem2.getNombre().equals(anObject: tablaSimbolos.get(index:i).getNombre()))
1503                         {
1504                             valor2 = tablaSimbolos.get(index:i).getValor();
1505                         }
1506
1507                         if (valor2.equals(obj: "true") || valor.equals(obj: "true"))
1508                         {
1509                             pilaControl.add(new Token(nombre: "true", token: -64, posTabla: 0, linea: 0));
1510                         }
1511                     }
1512                 }
1513             }
1514         }
1515     }
1516 }
```

The screenshot shows a Java IDE interface with the title bar "Start Page x Simulación_Ejecución_VCI.java". The left sidebar includes "Source", "History", "Projects", "Files", and "Services". The main area displays the following Java code:

```
1513     }
1514     else
1515     {
1516         pilaControl.add(new Token( nombre: "false", token: -64, posTabla: 0, linea: 0));
1517     }
1518 }
1519 else if (tem.getToken() == -64 && tem2.getToken() == -64)
1520 {
1521
1522     if (tem2.getNombre().equals( anObject: "true" ) || tem.getNombre().equals( anObject: "true" ))
1523     {
1524         pilaControl.add(new Token( nombre: "true", token: -64, posTabla: 0, linea: 0));
1525     }
1526     else
1527     {
1528         pilaControl.add(new Token( nombre: "false", token: -64, posTabla: 0, linea: 0));
1529     }
1530 }
1531 else
1532 {
1533
1534     if (tem.getToken() == -54)
1535     {
1536         for (int i = 0; i < tablaSimbolos.size(); i++)
1537         {
1538             if (tem.getNombre().equals( anObject: tablaSimbolos.get( index: i ).getNombre() ))
1539             {
1540                 valor = tablaSimbolos.get( index: i ).getValor();
1541             }
1542
1543             if (tem2.getNombre().equals( anObject: "true" ) || valor.equals( obj: "true" ))
1544             {
1545                 pilaControl.add(new Token( nombre: "true", token: -64, posTabla: 0, linea: 0));
1546             }
1547             else
1548             {
1549                 pilaControl.add(new Token( nombre: "false", token: -64, posTabla: 0, linea: 0));
1550             }
1551         }
1552     }
1553
1554     for (int i = 0; i < tablaSimbolos.size(); i++)
1555     {
1556         if (tem2.getNombre().equals( anObject: tablaSimbolos.get( index: i ).getNombre() ))
1557         {
1558             valor2 = tablaSimbolos.get( index: i ).getValor();
1559         }
1560
1561         if (valor2.equals( obj: "true" ) || tem.getNombre().equals( anObject: "true" ))
1562         {
1563             pilaControl.add(new Token( nombre: "true", token: -64, posTabla: 0, linea: 0));
1564         }
1565         else
1566         {
1567             pilaControl.add(new Token( nombre: "false", token: -64, posTabla: 0, linea: 0));
1568         }
1569     }
1570 }
1571
1572     break;
1573 }
1574
1575 // caso !
1576 case "-43":
1577     tem = pilaControl.pop();
1578     tem2 = pilaControl.pop();
1579
1580     switch (tem.getToken())
1581     {
1582         case -54:
1583         case -64:
1584             if (tem.getToken() == -54 && tem2.getToken() == -54)
1585             {
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1599 }
```

The screenshot shows a Java IDE interface with the title bar "Start Page x Simulación_Ejecución_VCI.java". The left sidebar includes "Source", "History", "Projects", "Files", and "Services". The main area displays the following Java code:

```
1549     }
1550 }
1551 else
1552 {
1553
1554     for (int i = 0; i < tablaSimbolos.size(); i++)
1555     {
1556         if (tem2.getNombre().equals( anObject: tablaSimbolos.get( index: i ).getNombre() ))
1557         {
1558             valor2 = tablaSimbolos.get( index: i ).getValor();
1559         }
1560
1561         if (valor2.equals( obj: "true" ) || tem.getNombre().equals( anObject: "true" ))
1562         {
1563             pilaControl.add(new Token( nombre: "true", token: -64, posTabla: 0, linea: 0));
1564         }
1565         else
1566         {
1567             pilaControl.add(new Token( nombre: "false", token: -64, posTabla: 0, linea: 0));
1568         }
1569     }
1570
1571     break;
1572 }
1573
1574 // caso !
1575 case "-43":
1576     tem = pilaControl.pop();
1577     tem2 = pilaControl.pop();
1578
1579     switch (tem.getToken())
1580     {
1581         case -54:
1582         case -64:
1583             if (tem.getToken() == -54 && tem2.getToken() == -54)
1584             {
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1599 }
```

The screenshot shows a Java IDE interface with the following details:

- Title Bar:** Start Page | Simulación_Ejecución_VCI.java
- Toolbar:** Includes icons for New, Open, Save, Cut, Copy, Paste, Find, Replace, and others.
- Left Sidebar:** Shows project navigation with 'Start Page', 'History', 'Navigator', 'Projects', 'Files', 'Services', and 'Source' tabs.
- Code Editor:** Displays lines of code from 1585 to 1621. The code involves comparing objects from a 'tablaSimbolos' list and adding tokens to a 'pilaControl' list based on their values ('true' or 'false').
- Status Bar:** Shows '1:1' and 'ANSI Windows (CRLF)'.

```
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
```

The screenshot shows a Java IDE interface with the following details:

- Title Bar:** Start Page | Simulación_Ejecución_VCI.java
- Toolbar:** Includes icons for New, Open, Save, Cut, Copy, Paste, Find, Replace, and others.
- Left Sidebar:** Shows project navigation with 'Start Page', 'History', 'Navigator', 'Projects', 'Files', 'Services', and 'Source' tabs.
- Code Editor:** Displays lines of code from 1621 to 1657. The code follows a similar pattern to the first screenshot, involving comparisons and token additions.
- Status Bar:** Shows '1:1' and 'ANSI Windows (CRLF)'.

```
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
```

The screenshot shows a Java IDE interface with a code editor window titled "Simulación_Ejecución_VCJava". The code is written in Java and appears to be part of a parser or interpreter. It includes several try-catch blocks, notably for IOException and ArithmeticException, and uses System.out.println statements for debugging and output. The code is numbered from 1657 to 1689.

```
1657     pilaControl.add(new Token( nombre: "true", token: -64, posTabla: 0, linea: 0));
1658
1659     }
1660     }
1661     break;
1662   }
1663   break;
1664 }
1665
1666 else
1667 {
1668   pilaControl.add(new Token(partes[0], token: Integer.parseInt(partes[1]), posTabla: Integer.parseInt(partes[2]), linea: Integer.parseInt(partes[3])));
1669 }
1670 }
1671
1672 catch (IOException e)
1673 {
1674   System.out.println("error");
1675 }
1676 catch (ArithmaticException e)
1677 {
1678   System.out.println("no se puede hacer la division entre 0");
1679 }
1680
1681 System.out.println("\nimpresion de la tabla de simbolos");
1682
1683 for (int i = 0; i < tablaSimbolos.size(); i++)
1684 {
1685   System.out.println(tablaSimbolos.get(index:i).toString());
1686 }
1687
1688 sobreescribirArchivoSimbolos( simbolos: tablaSimbolos);
1689 }
```

Método sobreescribirArchivoSimbolos

The screenshot shows a Java IDE interface with the following details:

- Title Bar:** Start Page, Simulacion_Ejecución_VCL.java, Tabla de Simbolos.txt, Cinta de VCL.txt
- Toolbars:** Source, History, etc.
- Left Sidebar:** Projects, Files, Services, Bp, Navigator.
- Code Editor:** Displays the following Java code:

```
1690
1691     public static void sobreescribirArchivoSimbolos(ArrayList<Tokensimbolo> simbolos)
1692     {
1693         try (BufferedWriter bw = new BufferedWriter(new FileWriter( string: "C:\\\\Users\\\\rogel\\\\OneDrive\\\\Escritorio\\\\Semestre 8\\\\Lenguajes Y Autómatas II\\\\Pr
1694
1695             for (Tokensimbolo simbolo : simbolos)
1696             {
1697                 bw.write(simbolo.getNombre() + "," + simbolo getToken() + "," + simbolo.getValor() + ","
1698                 |"Main");
1699                 bw.newLine();
1700             }
1701         } catch (IOException e)
1702         {
1703             e.printStackTrace();
1704         }
1705     }
1706 }
```

Método lecturaDeSimbolos

The screenshot shows a Java code editor window with the title bar "Start Page" and "Simulación_Ejecución_VCJava". The left sidebar includes "Navigator", "Projects", "Files", and "Services". The main pane displays the following Java code:

```
1708     public static ArrayList<Tokensimbole> lecturaDeSimbolos(String tbSimbolos)
1709     {
1710         ArrayList<Tokensimbole> arreglo = new ArrayList<>();
1711         try
1712         {
1713             BufferedReader lector = new BufferedReader(new FileReader(string.tbSimbolos));
1714             String linea;
1715
1716             while ((linea = lector.readLine()) != null)
1717             {
1718                 String[] partes = linea.split(sep: ",");
1719                 arreglo.add(new Tokensimbole(partes[0], token: Integer.parseInt(partes[1]), partes[2]));
1720             }
1721
1722             lector.close();
1723         }
1724         catch (IOException e)
1725         {
1726             System.out.println("error de lectura");
1727         }
1728
1729     return arreglo;
1730 }
1731 }
```

Tabla de Símbolos

1	a,-54,0,Main
2	s,-51,10,Main
3	x,-54,10,Main
4	y,-54,true,Main
5	

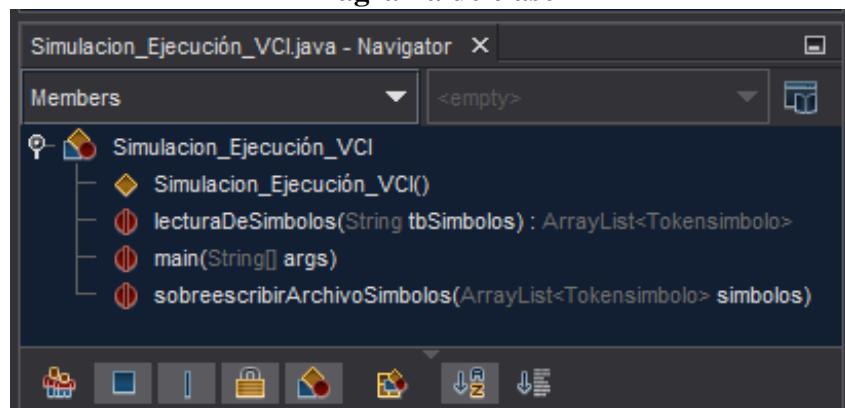
Tabla de Direcciones

1	JNO@,-55,1,0

Vector de Código Intermedio

1	a,-54,0,1
2	x,-51,0,1
3	63,-61,0,1
4	>,-33,0,1
5	y,-54,0,1
6	true,-64,0,1
7	==,-35,0,1
8	and,-41,0,1
9	s,-51,0,1
10	5,-61,0,1
11	<,-31,0,1
12	or,-42,0,1
13	=,-26,0,1
14	

Diagrama de clase



Ejecución

The screenshot shows the 'Output' window of an IDE during execution. The window has tabs at the top: 'Output' (selected), 'Debugger Console', and 'Proyecto (run)'. The main area displays the following text:

```
impresion de la tabla de simblos
Tokensimbolo(nombre='a', Token=-54, valor=false)
Tokensimbolo(nombre='s', Token=-51, valor=10)
Tokensimbolo(nombre='y', Token=-54, valor=10)
Tokensimbolo(nombre='y', Token=-54, valor=true)

BUILD SUCCESSFUL (total time: 1 second)
```

At the bottom right of the window, there are status indicators: '1688.51/1.13' and '| NS |'.

Resultado

1	a,-54,false,Main
2	s,-51,10,Main
3	x,-54,10,Main
4	y,-54,true,Main
5	

Cambios

Los cambios hechos en este programa consisten en que se lee el archivo de VCI y dependiendo del número de token se realiza una acción u otra cuando encuentra operadores los realiza y vuelven a entrar a la pila de ejecución también cuando existen palabras como do, then, until realiza las reglas para saber que hacer a continuación a rasgos generales solo se hace una lectura de VCI en la que se va realizando lo solicitado por el archivo.

Conclusión

Rogelio: Para realizar esta práctica, enfrentamos algunas dificultades, siendo la principal la ausencia de una de nuestras compañeras, lo que nos obligó a repartir más carga de trabajo entre los demás integrantes. Además, el tiempo otorgado para el desarrollo de esta última práctica fue muy corto, lo que generó más complicaciones para cumplir con el objetivo. Debido a esto, la división del trabajo recayó más en un integrante del equipo que en otros. En mi caso, me dediqué a explorar varias maneras de cumplir el objetivo y a discutir con el equipo cuál sería la mejor opción para llevar a cabo la práctica.

General: En conclusión, he adquirido una valiosa experiencia en el tema de ejecución de VCI. A lo largo de este proceso, he podido observar con mayor claridad cómo se ejecutan las líneas de código, desde la fase inicial de generación de VCI hasta su ejecución final. Esta experiencia me ha permitido comprender en profundidad el funcionamiento de los condicionales y las estructuras de control, lo cual ha sido fundamental para mi desarrollo en este campo. Además, he aprendido a identificar y resolver problemas comunes que pueden surgir durante la ejecución de código, mejorando así mi capacidad para escribir y depurar programas de manera más eficiente. Esta experiencia ha sido enriquecedora y me ha proporcionado herramientas prácticas que podré aplicar en futuros proyectos.

Conclusión del Proyecto integrador

La serie de prácticas llevadas a cabo constituye un proceso integral de aprendizaje y aplicación de conceptos fundamentales en el desarrollo de software, especialmente en el contexto de la compilación y la ejecución de código. A través de estas actividades, se exploraron temas como el análisis léxico, el análisis semántico, la generación de código intermedio y la ejecución del mismo mediante la simulación de una pila de ejecución.

En la primera práctica, se abordó el análisis semántico mediante la generación de tablas de símbolos y direcciones, lo cual constituye un paso esencial en la construcción de un compilador o intérprete. La aplicación desarrollada permitió leer una tabla de tokens y generar las tablas requeridas, cumpliendo con las especificaciones dadas.

En la segunda práctica, se avanzó hacia la generación del vector de código intermedio (VCI) para expresiones aritméticas, lógicas y estructuras de control. Esto implicó el diseño e implementación de un programa capaz de simular este proceso, utilizando las estructuras de datos adecuadas y considerando las prioridades de operadores.

Finalmente, en la práctica tres, se llevó a cabo la simulación de la pila de ejecución del VCI. Esta etapa es crucial para comprender cómo se ejecuta el código generado previamente, mostrando la salida correspondiente y actualizando la tabla de símbolos según sea necesario.

A lo largo de este proceso, se enfrentaron diversos desafíos y se tuvieron que realizar ajustes y optimizaciones en el código para garantizar su correcto funcionamiento. Además, la colaboración entre los miembros del equipo fue fundamental para distribuir tareas y compartir conocimientos, lo que contribuyó al éxito de las prácticas.

Bibliografía

1. Optimización de tipo mirilla en Java. (s.f.).
2. Optimización de tipo mirilla en el compilador JIT de Java. (s.f.).
3. Mejora del rendimiento de Java mediante la optimización de tipo mirilla. (s.f.).
4. Holub, A., & Wehrle, J. (2008). Compiladores: Principios, técnicas y herramientas. (2^a ed.). Addison-Wesley.
5. Appel, A. W., & Dubinsky, M. (2005). Compiladores e intérpretes para lenguajes de programación. (2^a ed.). Cambridge University Press.
6. Optimización de código Java. (s.f.). Oracle.
7. Trucos para mejorar el rendimiento de Java. (s.f.). Oracle.
8. Optimización de tipo mirilla para el rendimiento de Java. (s.f.). En Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). Recuperado de https://link.springer.com/chapter/10.1007/978-3-642-31482-7_10
9. Implementación de optimización de tipo mirilla en un compilador JIT de Java. (s.f.). En Proceedings of the International Conference on Compiler Construction (CC). Recuperado de https://link.springer.com/chapter/10.1007/978-3-642-31482-7_10
10. Evaluación del impacto de la optimización de tipo mirilla en el rendimiento de Java. (s.f.).
11. Medición del rendimiento de las optimizaciones del compilador JIT en Java. (s.f.).
12. Técnicas de perfilado y análisis para optimizar el rendimiento de Java. (s.f.).
13. Optimización de rendimiento de Java:
<https://learn.oracle.com/education/html/pages/article3-java.html>
14. Meyer, S. (2010). Java Performance Tuning (Second Edition). Peachpit Press.
15. Rose, J. (2011). Java Performance and Optimization (Second Edition). Addison-Wesley.
16. Un estudio empírico sobre el impacto de la optimización de tipo mirilla en el rendimiento de Java. (s.f.). Recuperado de https://link.springer.com/chapter/10.1007/978-3-642-31482-7_10
17. Evaluación del rendimiento de las optimizaciones de tipo mirilla en diferentes compiladores JIT de Java. (s.f.). Recuperado de https://link.springer.com/chapter/10.1007/978-3-642-31482-7_10
18. Análisis del rendimiento de código Java sin optimización. (s.f.).
19. Comparación de tiempos de ejecución en Java con y sin optimizaciones. (s.f.).
20. Medición del impacto de las optimizaciones del compilador en el rendimiento de Java. (s.f.).
21. <http://itpn.mx/recursosisc/7semestre/leguajesyautomatas2/Unidad%20III.pdf>

Anexos