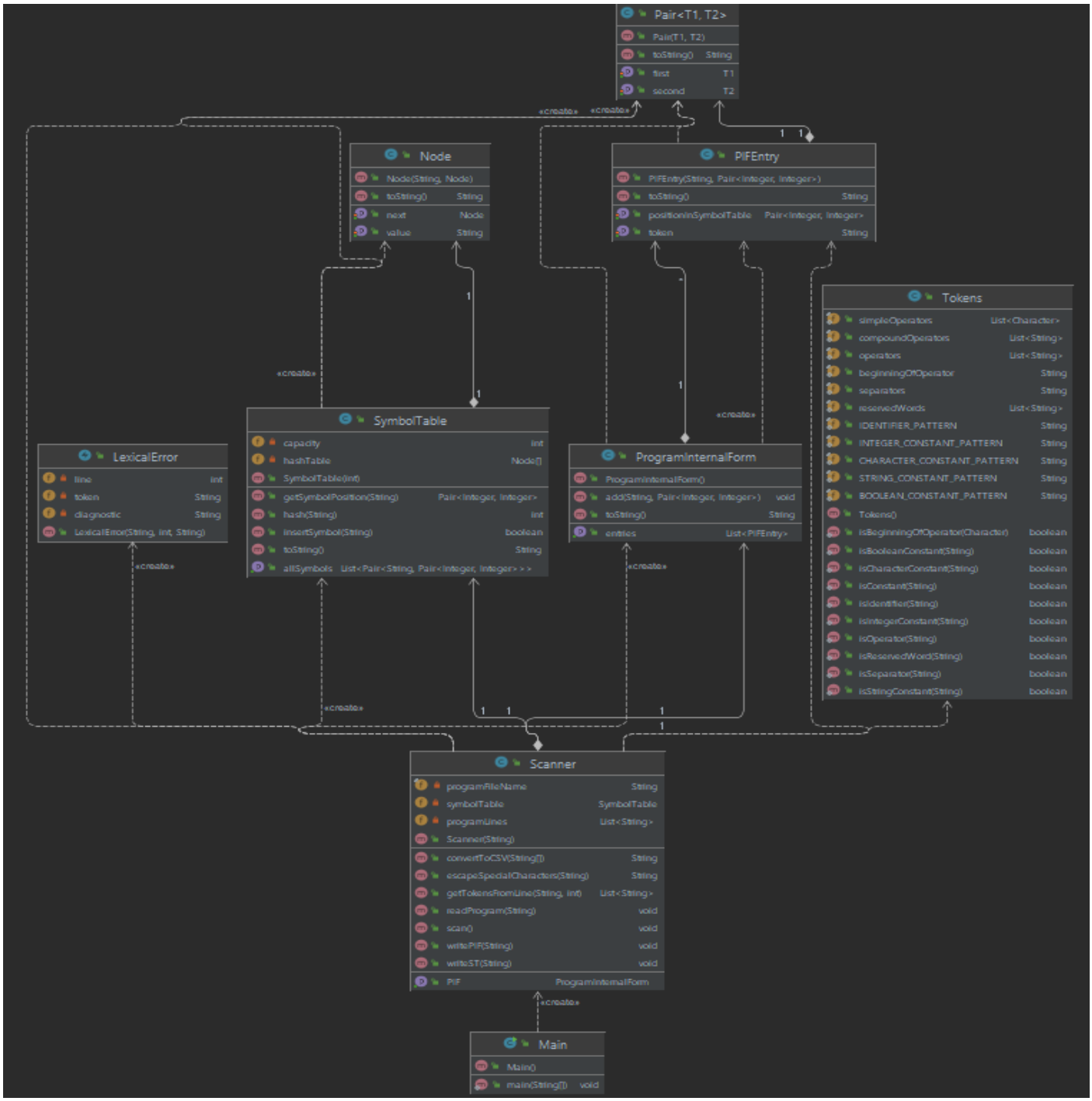
Scanner implementation

Github link:

https://github.com/CimpeanAndreea/FLCD/tree/master/Lab3

Implementation: To show the method chosen the class diagram is representative



The method chosen to implement the Scanner is to read the program, split it in lines, and take each line character by character and determine if it's a space, separator, operator, possible part of an identifier/constant/reserverd word. Following this method will result in lines split in tokens. After obtaining the tokens from a line, they are classified in separators, operators, reserved words, identifiers, tokens, and if the token does not follow the pattern of neither of these a lexical error is thrown. If there is no lexical error, each token is put in the Program Internal Form as a string together with its position in the Symbol Table ((-1,1) if the token is an operator/ a separator/ a reserved word).

The classes used are those needed for the Symbol Table, explained in the Symbol Table representation and beside them there are 5 more classes:

1. Lexical Error:

   Extends Java RuntimeException class and is used when a lexical error is detected to indicate the line and token

2. Tokens:

   Contains the tokens that can appear in a program (separators, operators, reserved words) and the patterns of identifiers and constants, as well as methods to determine if a token is of a certain category

3. PIFEntry:

   Represents one Program Internal Form entry as a string(for the token) and a pair of 2 integers (for the position in the symbol table)

4. ProgramInternalForm:

   Represents the Program Internal Form as a list of PIFEntries and contains a methos to add a PIFEntry in the list

5. Scanner:

   Contains the logic of scanning the program.

   It has the following properties:

   programFileName: string

   symbolTable: SymbolTable

   PIF: ProgramInternalForm

   programLines: string[]

   It has the following methods:

   function readProgram(fileName)

description: read an input program from a specified file, and add the lines of it in the programLines

pre: fileName is a valid file

prost: programLines contains the lines of the program or if invalid fileName throws an error

function writePIF(fileName) & function writeST(fileName)

description: write PIF(ST) in a csv formar in the specified file

function convertToCsv(data) & function escapeSpecialCharacters(data)

description: helper functions to write data in a csv file

function getTokensFromLine(line, indexLine)

description: split a line in tokens

pre: line is a string(a line from the program), indexLine is an integer corresponding to the number of the line in the program

post: return a list of strings (tokens) or throw an error if a lexical error could be detected while taking the line character by character(no closing ",' for constants)

function scan()

description: for each line of the program obtain the tokens with the getTokensFromLine() function and after tokens are obtained classify them, add them in PIF and ST(if the case) or throw an LexicalError if a token could not be identified

post: complete PIF & ST and print a "Lexically correct" message or throw an LexicalError

Tests:

1. **p1.txt**

```
PROGRAM
   MAIN -> {
      DECLARATIONS
         INTEGER: nr_1, nr_2, nr_3, min;
         STRING: output_message <- "The minimum of the 3 numbers is- ";
      STATEMENTS
      {

         in>>nr_1, nr_2, nr_3;
         if (nr_1 < nr_2)
            min<-nr_1;
         else
            min <- nr_2;
```

```
        if (nr_3 < min)
           min <- nr_3;
        out<<output_message, min;
     }
  }
```

st.csv

	Token	Position
2	nr_1	(8,0)
3	nr_2	(9,0)
4	nr_3	(10,0)
5	"The minimum of the 3 numbers is- "	(19,0)
6	min	(24,0)
7	output_message	(25,0)

pif.csv

	Token	Position in ST
2	PROGRAM	(-1,-1)
3	MAIN	(-1,-1)
4	->	(-1,-1)
5	{	(-1,-1)
6	DECLARATIONS	(-1,-1)
7	INTEGER	(-1,-1)
8	:	(-1,-1)
9	identifier	(8,0)
10	,	(-1,-1)
11	identifier	(9,0)
12	,	(-1,-1)
13	identifier	(10,0)
14	,	(-1,-1)
15	identifier	(24,0)
16	;	(-1,-1)
17	STRING	(-1,-1)
18	:	(-1,-1)
19	identifier	(25,0)
20	<-	(-1,-1)
21	constant	(19,0)
22	;	(-1,-1)
23	STATEMENTS	(-1,-1)
24	{	(-1,-1)
25	in	(-1,-1)
26	>>	(-1,-1)
27	identifier	(8,0)
28	,	(-1,-1)

29	identifier	(9,0)
30	,	(-1,-1)
31	identifier	(10,0)
32	;	(-1,-1)
33	if	(-1,-1)
34	(	(-1,-1)
35	identifier	(8,0)
36	<	(-1,-1)
37	identifier	(9,0)
38	)	(-1,-1)
39	identifier	(24,0)
40	<-	(-1,-1)
41	identifier	(8,0)
42	;	(-1,-1)
43	else	(-1,-1)
44	identifier	(24,0)
45	<-	(-1,-1)
46	identifier	(9,0)
47	;	(-1,-1)
48	if	(-1,-1)
49	(	(-1,-1)
50	identifier	(10,0)
51	<	(-1,-1)
52	identifier	(24,0)
53	)	(-1,-1)
54	identifier	(24,0)
55	<-	(-1,-1)
56	identifier	(10,0)
57	;	(-1,-1)
58	out	(-1,-1)
59	<<	(-1,-1)
60	identifier	(25,0)
61	,	(-1,-1)
62	identifier	(24,0)
63	;	(-1,-1)
64	}	(-1,-1)
65	}	(-1,-1)

2. **p1err.txt**

```
PROGRAM
   MAIN -> {
      DECLARATIONS
         INTEGER: 1_nr, nr_2, nr_3, min;
         STRING: output_message <- "The minimum of the 3 numbers is: ;
      STATEMENTS
      {
         in>>nr_1, nr_2, nr_3;
         if (nr_1 < nr_2)
            min <- nr_1;
         else
            min <- nr_2;
         if (nr_3 < min)
            min <- nr_3;
         out<<output_message, min;
      }
   }
```

Lexical error at line: 4 at token: 1_nr Unidentified token

3. **p2.txt**

```
PROGRAM
   MAIN -> {
      DECLARATIONS
         INTEGER: input_number, i;
         BOOLEAN: prime <- true;
      STATEMENTS
      {
         in("Give a number: ")>>input_number;
         for(i <- +2; i <= input_number / 2; i <- i+1)
            if (input_number % i == 0)
               prime <- false;
         if (prime == true)
            out<<"The number ", input_number, " is prime";
         else
            out<<"The number ", input_number, " is not prime";
      }
   }
```

st.csv

	Token	Position
2	prime	(1,0)
3	" is not prime"	(2,0)
4	+2	(3,0)
5	"Give a number: "	(13,0)
6	false	(13,1)
7	input_number	(14,0)
8	i	(15,0)
9	0	(18,0)
10	1	(19,0)
11	2	(20,0)
12	"The number "	(20,1)
13	" is prime"	(23,0)
14	true	(28,0)

pif.csv

	Token	Position in ST
2	PROGRAM	(-1,-1)
3	MAIN	(-1,-1)
4	->	(-1,-1)
5	{	(-1,-1)
6	DECLARATIONS	(-1,-1)
7	INTEGER	(-1,-1)
8	:	(-1,-1)
9	identifier	(14,0)
10	,	(-1,-1)
11	identifier	(15,0)
12	;	(-1,-1)
13	BOOLEAN	(-1,-1)
14	:	(-1,-1)
15	identifier	(1,0)
16	<-	(-1,-1)
17	constant	(28,0)
18	;	(-1,-1)
19	STATEMENTS	(-1,-1)
20	{	(-1,-1)
21	in	(-1,-1)
22	(	(-1,-1)
23	constant	(13,0)
24	)	(-1,-1)
25	>>	(-1,-1)
26	identifier	(14,0)
27	;	(-1,-1)
28	for	(-1,-1)

28	for	(-1,-1)
29	(	(-1,-1)
30	identifier	(15,0)
31	<-	(-1,-1)
32	constant	(3,0)
33	;	(-1,-1)
34	identifier	(15,0)
35	<=	(-1,-1)
36	identifier	(14,0)
37	/	(-1,-1)
38	constant	(20,0)
39	;	(-1,-1)
40	identifier	(15,0)
41	<-	(-1,-1)
42	identifier	(15,0)
43	+	(-1,-1)
44	constant	(19,0)
45	)	(-1,-1)
46	if	(-1,-1)
47	(	(-1,-1)
48	identifier	(14,0)
49	%	(-1,-1)
50	identifier	(15,0)
51	==	(-1,-1)
52	constant	(18,0)

53	)	(-1,-1)
54	identifier	(1,0)
55	<-	(-1,-1)
56	constant	(13,1)
57	;	(-1,-1)
58	if	(-1,-1)
59	(	(-1,-1)
60	identifier	(1,0)
61	==	(-1,-1)
62	constant	(28,0)
63	)	(-1,-1)
64	out	(-1,-1)
65	<<	(-1,-1)
66	constant	(20,1)
67	,	(-1,-1)
68	identifier	(14,0)
69	,	(-1,-1)
70	constant	(23,0)
71	;	(-1,-1)
72	else	(-1,-1)
73	out	(-1,-1)
74	<<	(-1,-1)
75	constant	(20,1)
76	,	(-1,-1)
77	identifier	(14,0)
78	,	(-1,-1)
79	constant	(2,0)
80	;	(-1,-1)
81	}	(-1,-1)
82	}	(-1,-1)