RELATÓRIO DE TEORIA DOS COMPILADORES

Felippe Rocha Lôbo de Abreu - 201765185AC

Nathan Toschi Reis - 201865064C

Passo a passo)

O primeiro passo para criar o interpretador foi, com base na linguagem definida, separar os tipos de tokens que vamos utilizar, e para isso criamos uma classe enumerada no arquivo TOKEN_TYPE.java contendo os 11 tipos de tokens:

```java
/**
 *
 * @author felip
 */
public enum TOKEN_TYPE {
    ID, INT, TYPE, FLOAT, CHAR, BOOL, NULL, OP, SEP, PRINT, DEFOP;

}

```

Com isso em mãos, definimos o que é um objeto totem. Criamos assim uma classe em que nela podemos armazenar as informações de um dado token. Lá guardamos a linha, coluna, o lexeme (o carácter), o tipo e o objeto caso seja preciso guardar algo.

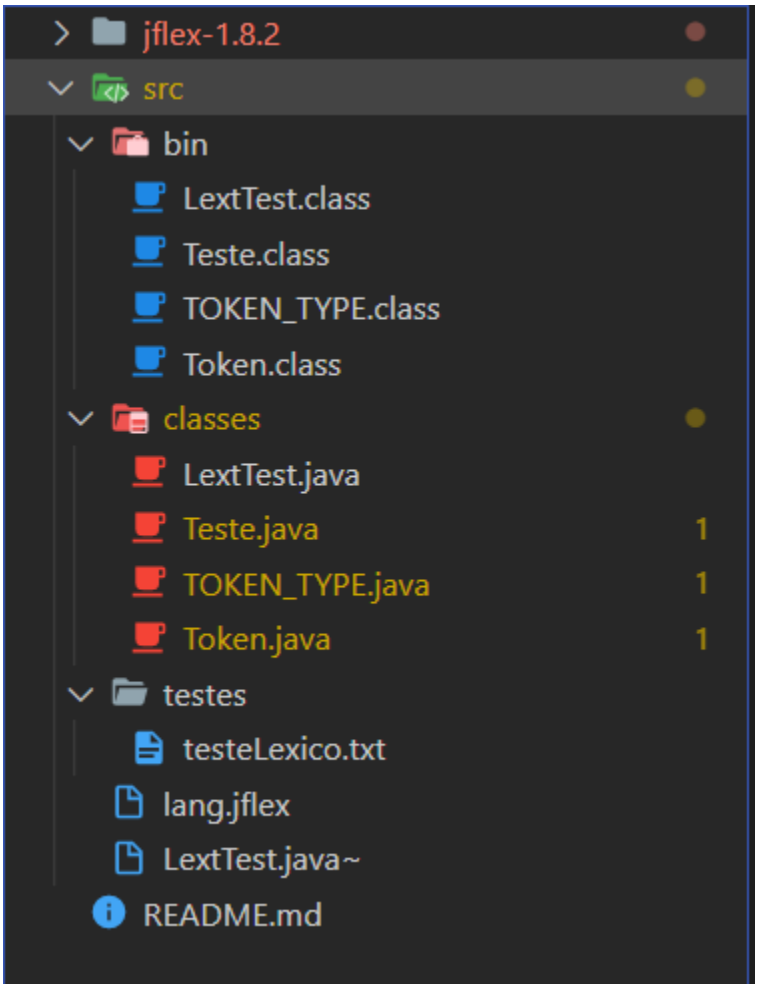
```java
/**
 * @author felip
 */
public class Token {
    public int l, c;
    public TOKEN_TYPE t;
    public String lexeme;
    public Object info;

    public Token(TOKEN_TYPE t, String lex, Object o, int l, int c) {
        this.t = t;
        lexeme = lex;
        info = o;
        this.l = l;
        this.c = c;

    }

    public Token(TOKEN_TYPE t, String lex, int l, int c) {
        this.t = t;
        lexeme = lex;
        info = null;
        this.l = l;
        this.c = c;

    }

    public Token(TOKEN_TYPE t, Object o, int l, int c) {
        this.t = t;
        lexeme = "";
        info = o;
        this.l = l;
        this.c = c;

    }

    @Override
    public String toString() {
        return this.t + (((this.lexeme == "") || (this.lexeme == null)) ? "" : ":" + this.lexeme)
                + (((this.info == "") || (this.info == null)) ? "" : ":" + this.info.toString());
    }
}

```

Com os tokens, criamos então o java.jflex e definimos eles lá dentro. Definimos o que caracteriza os nossos tokens e o que deve ser feito quando nos encontramos com eles durante a análise:

```
/* Agora vamos definir algumas macros */
FimDeLinha  = \r|\n|\r\n
Brancos     = {FimDeLinha} | [ \t\f]
type = [:uppercase:] [:uppercase:]*
float      = [:digit:] [:digit:]* "." [:digit:] [:digit:]*
inteiro = [:digit:] [:digit:]*
char = [:lowercase:] | [:uppercase:]
boolean = true | false
op = "="| "+"| "*" | "/" | "-"| "%"| "<"| ">"| "=="| "≠"| "&&"| "||"| "!"
defOp = "::" | ":"
sep = ";"| "("| ")"| "["| "]"| "{"| "}"| "."| ","
identificador = {char} {char}*
LineComment = "//" (.)* {FimDeLinha}
nulo = NULL


%state COMMENT

%%

<YYINITIAL>{
    "print"         { return symbol(TOKEN_TYPE.PRINT);}
    {nulo}          { return symbol(TOKEN_TYPE.NULL, null);}
    {type}          { return symbol(TOKEN_TYPE.TYPE, yytext()); }
    {boolean}       { return symbol(TOKEN_TYPE.BOOL, Boolean.parseBoolean(yytext()));}
    {identificador} { return symbol(TOKEN_TYPE.ID);   }
    {inteiro}       { return symbol(TOKEN_TYPE.INT, Integer.parseInt(yytext()) );  }
    {float}         { return symbol(TOKEN_TYPE.FLOAT, Float.parseFloat(yytext()) );  }
    {defOp}         { return symbol(TOKEN_TYPE.DEFOP);  }
    {op}            { return symbol(TOKEN_TYPE.OP, yytext()); }
    {sep}           {return symbol(TOKEN_TYPE.SEP, yytext());}
    "/*"            { yybegin(COMMENT);              }
    {Brancos}       { /* Não faz nada  */            }
    {LineComment}   {                       }
}
```

De resto mantivemos a execução própria do jflex. Executamos então o .jar, passando o nosso arquivo teste que será lido, e então criamos uma classe de Teste e passamos o arquivo para poder ser executado em cima deste LextText que foi criado.

Por fim, ficou dessa forma:



Como rodar)

1) Descompactar o arquivo.zip e abrir o terminal dentro do mesmo.
2) Digitar o comando no terminal: “java -cp src\bin\ Teste src\testes\testeLexico.txt”

Problemas enfrentados)

Tivemos um problema pois decidimos testar primeiro no NetBeams, e ao tentar rodar por cmd era dito que não encontrava o arquivo “Teste”. Ao levarmos os arquivos para outro repositório o programa rodou sem problemas.