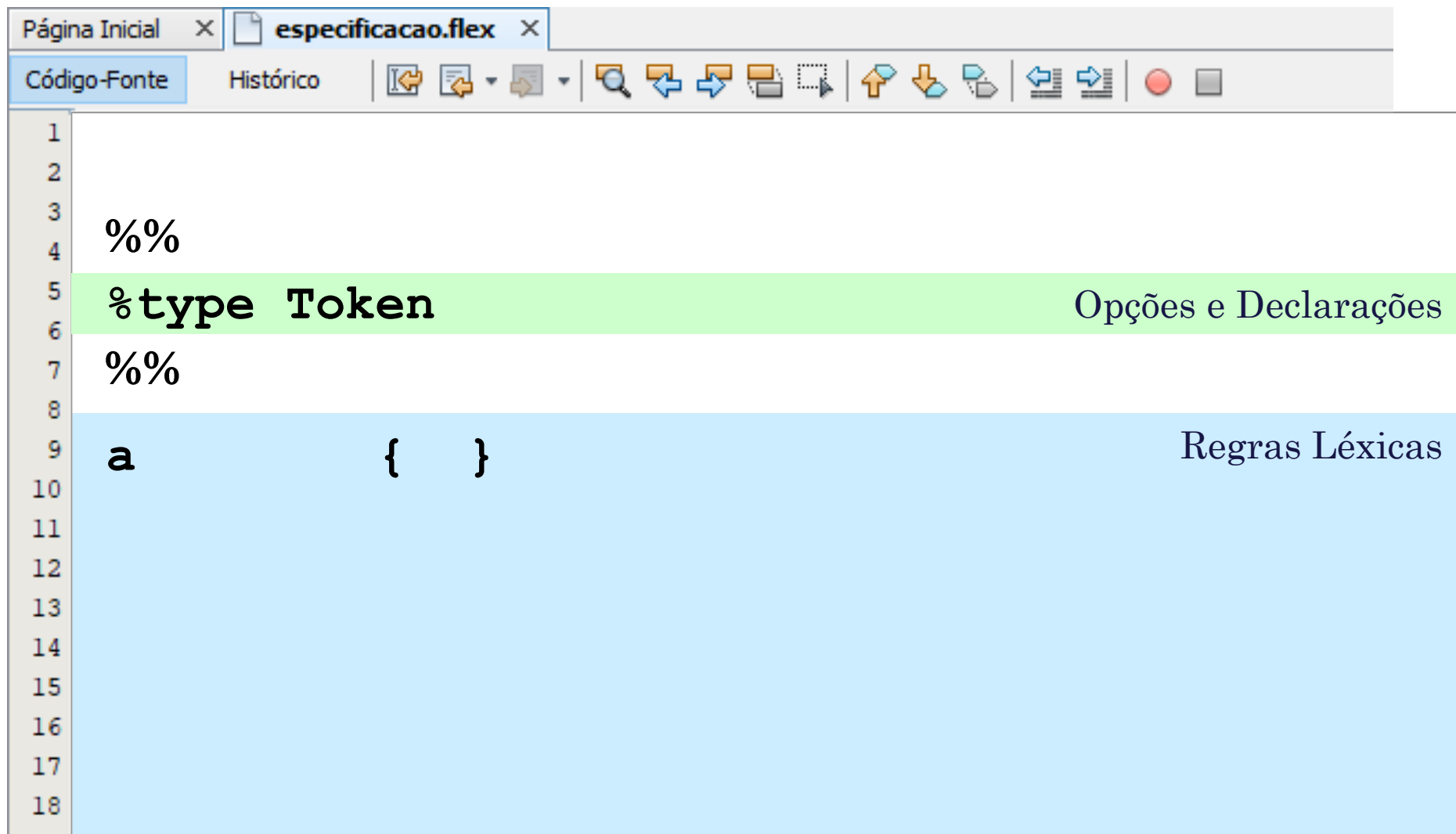


Associando a Classe de Tipos na especificação do JFLEX

É obrigatório possuir Regras léxicas



```
1
2
3  %%
4
5  %type Token
6
7  %%
8
9  a      {      }
10
11
12
13
14
15
16
17
18
```

Opções e Declarações

Regras Léxicas

Após executar o projeto

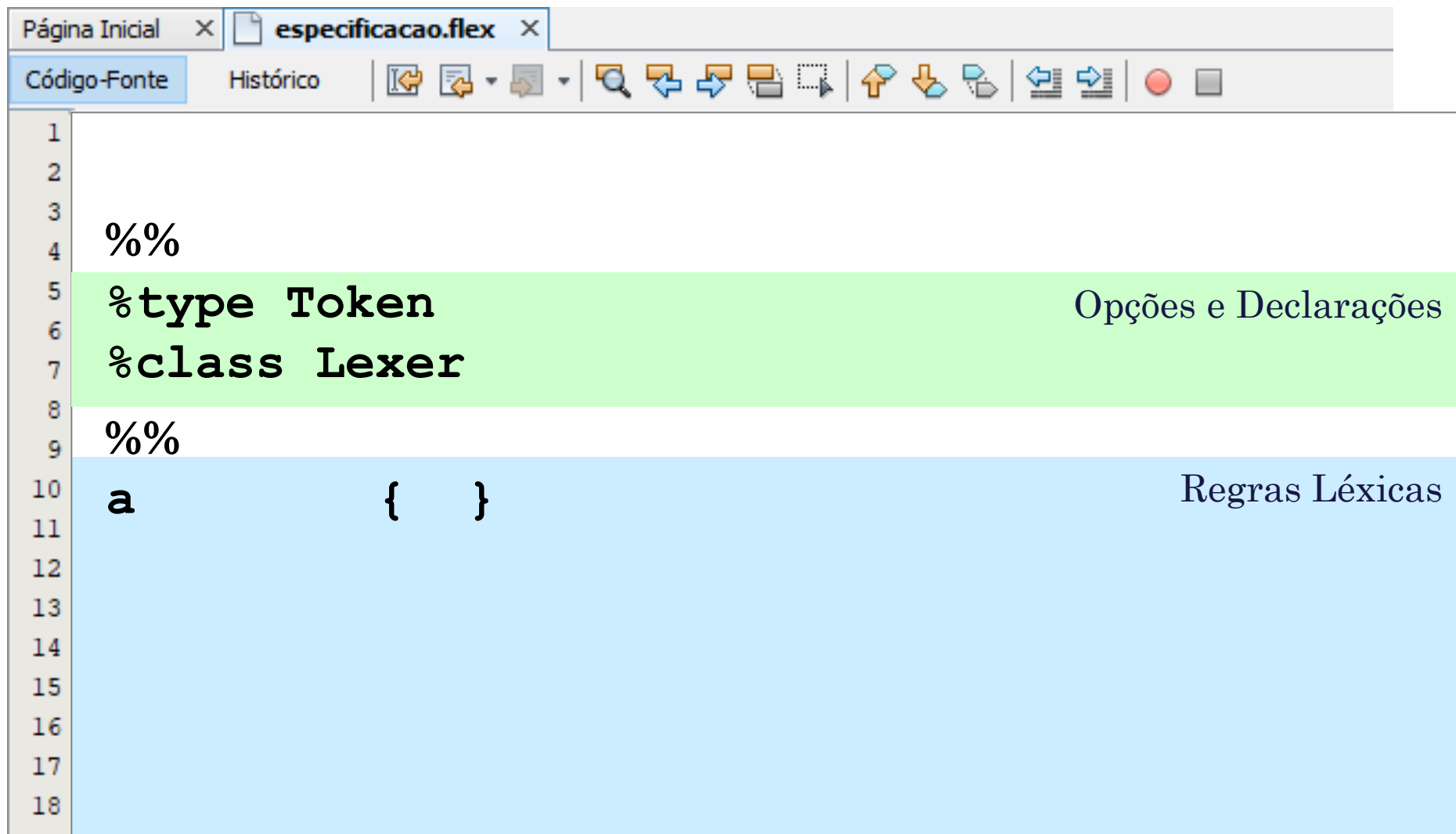
- Verifique que agora no método YYLEX() dentro da classe YYLEX o tipo da função a ser retornado será TOKEN.

```
451 | L      */
452 | [-]    public Token yylex() throws java.io.IOException {
453 | |      int zzInput;
```

Definindo novo nome para classe do Analizador Léxico

- O analisador gerado poderá ter um novo nome ao invés de `Yylex.java`.
- Para padronizarmos nosso projeto o chamaremos de `Lexer.java`

Gerando a classe com um nome definido



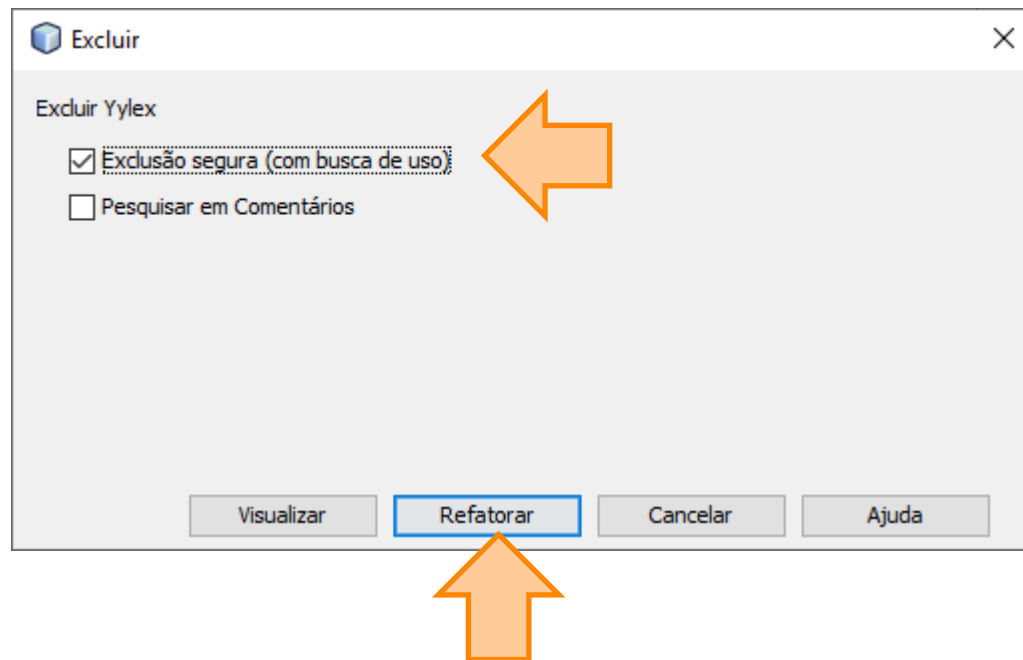
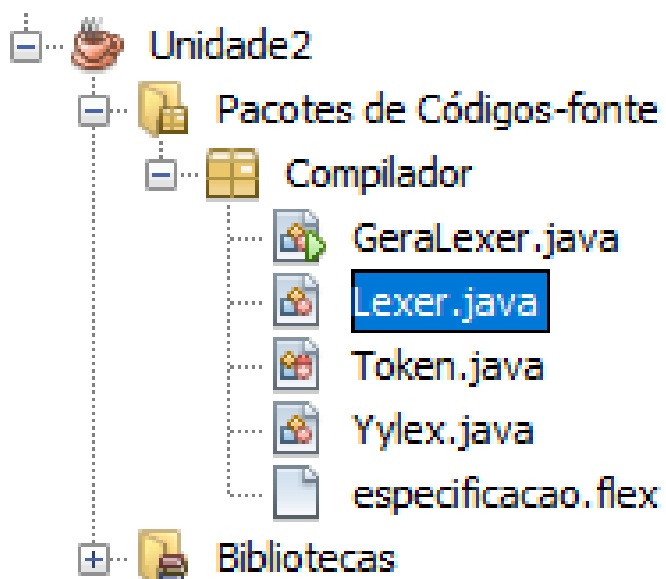
```
1
2
3 %%
4
5 %type Token
6 %class Lexer
7
8 %%
9
10 a { }
11
12
13
14
15
16
17
18
```

Opções e Declarações

Regras Léxicas

- Note que será gerada a classe chamada Lexer.

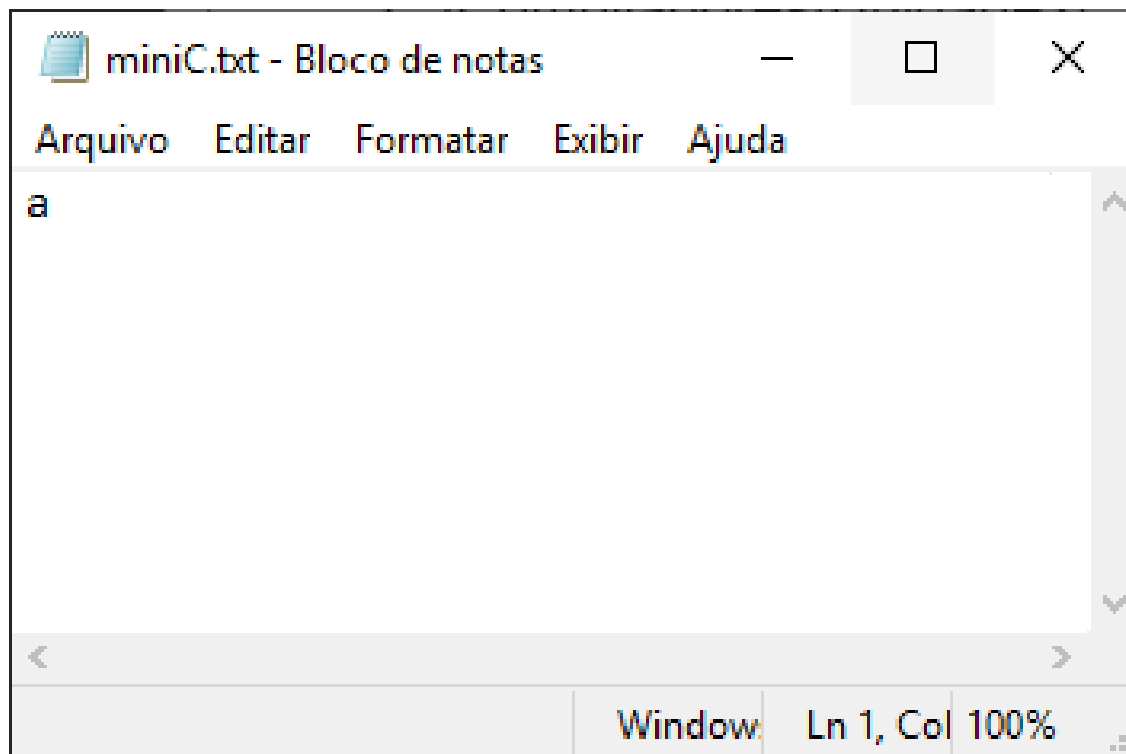
- Podemos excluir a classe Yylex.java
 - Botão direito > Excluir



Testar o Analisador

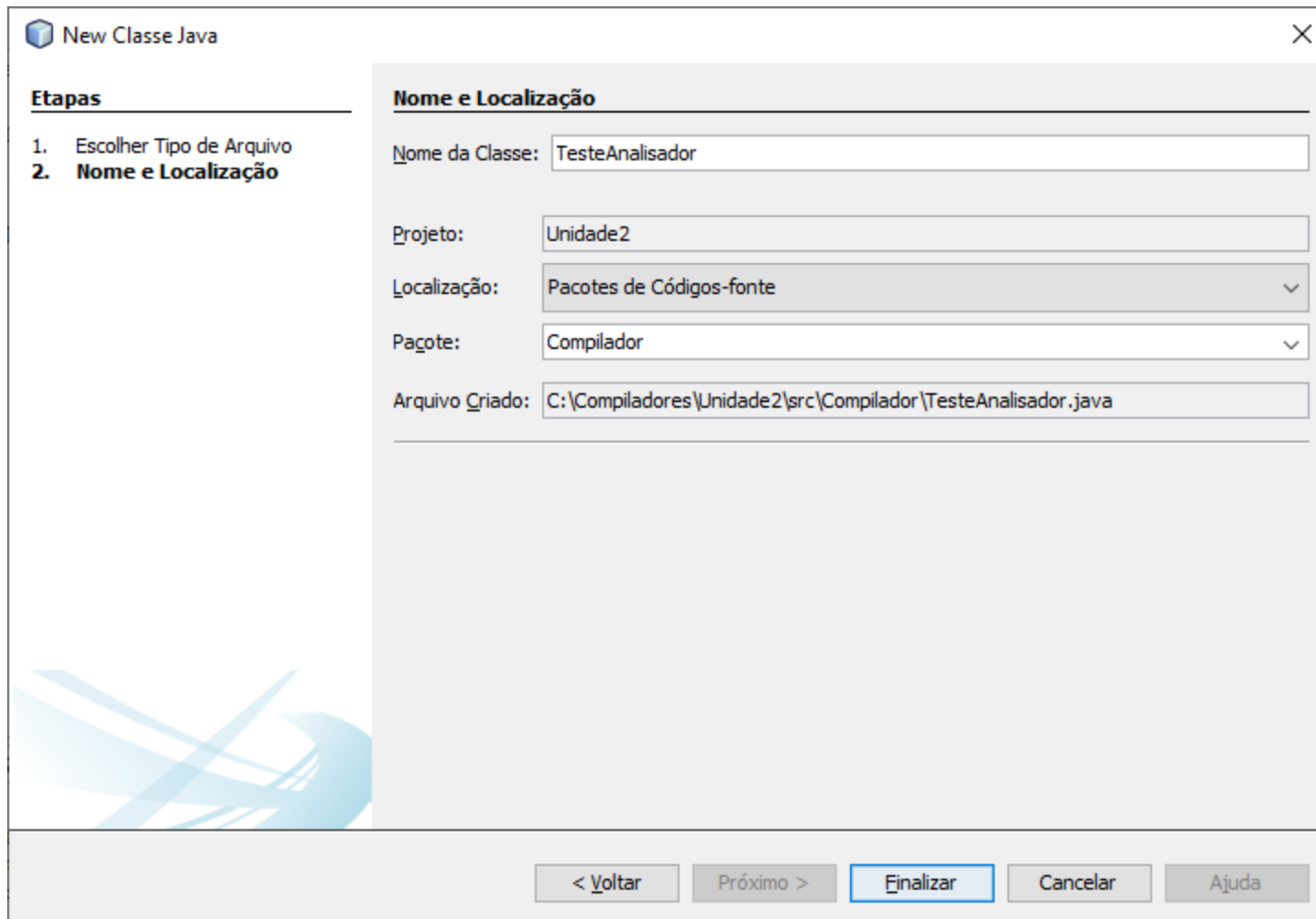
Criar um arquivo MiniC.TXT

- Crie um arquivo chamado miniC.TXT em:
 - C:/Compiladores/Unidade2/
 - No corpo do arquivo coloque apenas uma letra “a”



Criar a classe TesteAnalizador.java

- Botão direito sobre o pacote “Compilador”, Novo > Classe Java



New Classe Java

Etapas

- Escolher Tipo de Arquivo
- Nome e Localização**

Nome e Localização

Nome da Classe:

Projeto:

Localização:

Pacote:

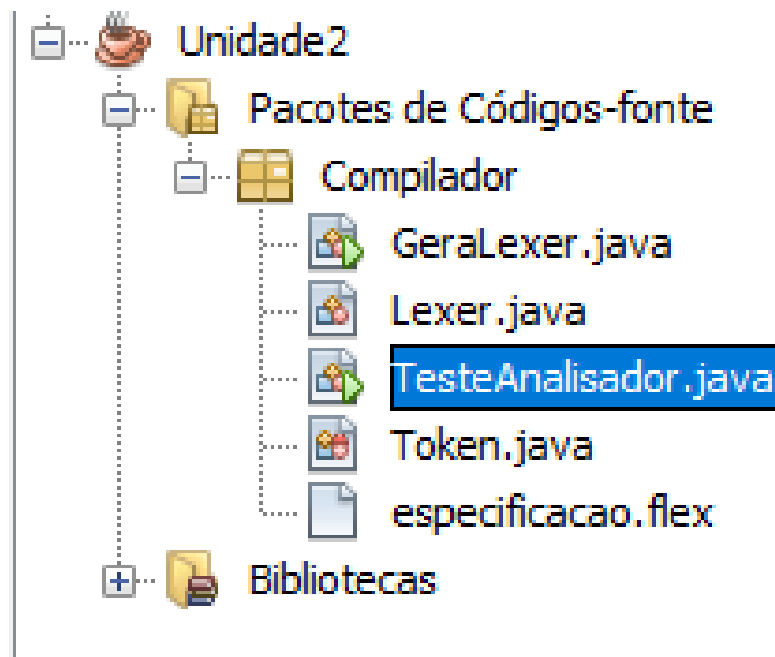
Arquivo Criado:

< Voltar Próximo > **Finalizar** Cancelar Ajuda

```
package Compilador;
import java.io.*;
public class TesteAnalizador {
    public static void main(String[] args) throws IOException {
        String arq = "C:/Compiladores/Unidade2/MiniC.TXT";
        BufferedReader in = new BufferedReader(new FileReader(arq));
        Lexer analyse = new Lexer(in);
        while (true){
            Token token = analyse.yylex();
            if (token==null) break; // EOF
        }
    }
}
```

Executar TesteAnalizador.Java

- Clique com botão ESQUERDO para selecionar o TesteAnalizador;
- Pressione <SHIFT>+F6



Padrões e Ações

- O analisador léxico gerado funciona com base em três regras fundamentais:
 1. apenas uma regra é aplicada a cada entrada do analisador, caractere ou string;
 2. a ação executada corresponde à regra para a qual foi conseguida a coincidência mais longa com a entrada corrente;
 3. no ponto anterior, caso exista duas ou mais regras com igual resultado para uma entrada, tem precedência a regra que aparece em primeiro lugar na especificação;

Padrões: Expressão regular

c	Caractere c
.	Qualquer caractere
[abc]	Um caractere da classe de caracteres, no caso um 'a' ou 'b' ou 'c'.
[a-z]	Qualquer caractere entre 'a' e 'z'.
[abj-oz]	Um caractere da classe de caracteres, no caso um 'a' ou 'b' ou qualquer caractere entre 'j' e 'o' ou 'z'.
[^a-g]	Um caractere que não esteja entre 'a' e 'g'.
c*	Zero ou mais caracteres 'c'.
c+	Um ou mais caracteres 'c'.
c?	Zero ou um caractere 'c'.
c{2,5}	Dois até cinco caracteres 'c'.
c{2,}	Dois ou mais caracteres 'c'.
c{4}	Exatamente 4 caracteres 'c'.
a b	Caractere 'a' ou 'b'
(a)	Parênteses são usados para modificar precedência.
\X	Utilizado para significado especial, como o ponto, aspas, etc.

Padrões: Expressão regular

<code>^r</code>	O carácter “r” apenas se no início da linha
<code>r\$</code>	O carácter “r” apenas se no final da linha (não consome o <code>\n</code>)
<code>[-+*/]</code>	Qualquer um dos operadores “-”, “+”, “*” ou “/”, sendo que o símbolo “-” tem de aparecer em primeiro lugar dada a possibilidade de ambiguidade com a definição de intervalo.
<code>[a-zA-Z]</code>	Um dos caracteres no intervalo de “a” a “z” ou de “A” a “Z”
<code>[abj-oz]</code>	Um caractere da classe de caracteres, no caso um ‘a’ ou ‘b’ ou qualquer caractere entre ‘j’ e ‘o’ ou ‘z’.
<code>xyz*</code>	A sequência “xy” seguida de zero ou mais “z”s
<code>(xyz)*</code>	A sequência “xyz” repetida zero ou mais vezes
<code>\n</code>	Mudança de linha
<code>\t</code>	Tabulação
<code>\r</code>	<i>Carrige return</i>
<code>r/s</code>	Encontra r apenas se for seguido por s (s não fará parte de <code>yytext</code>); nem todos as expressões regulares podem ser usadas em s – ver manual do Jflex

- O método **yytext()** retorna o lexema como uma *string*;
- O método **yylength()** retorna o número de caracteres de um lexema,

Exercício

Exercício

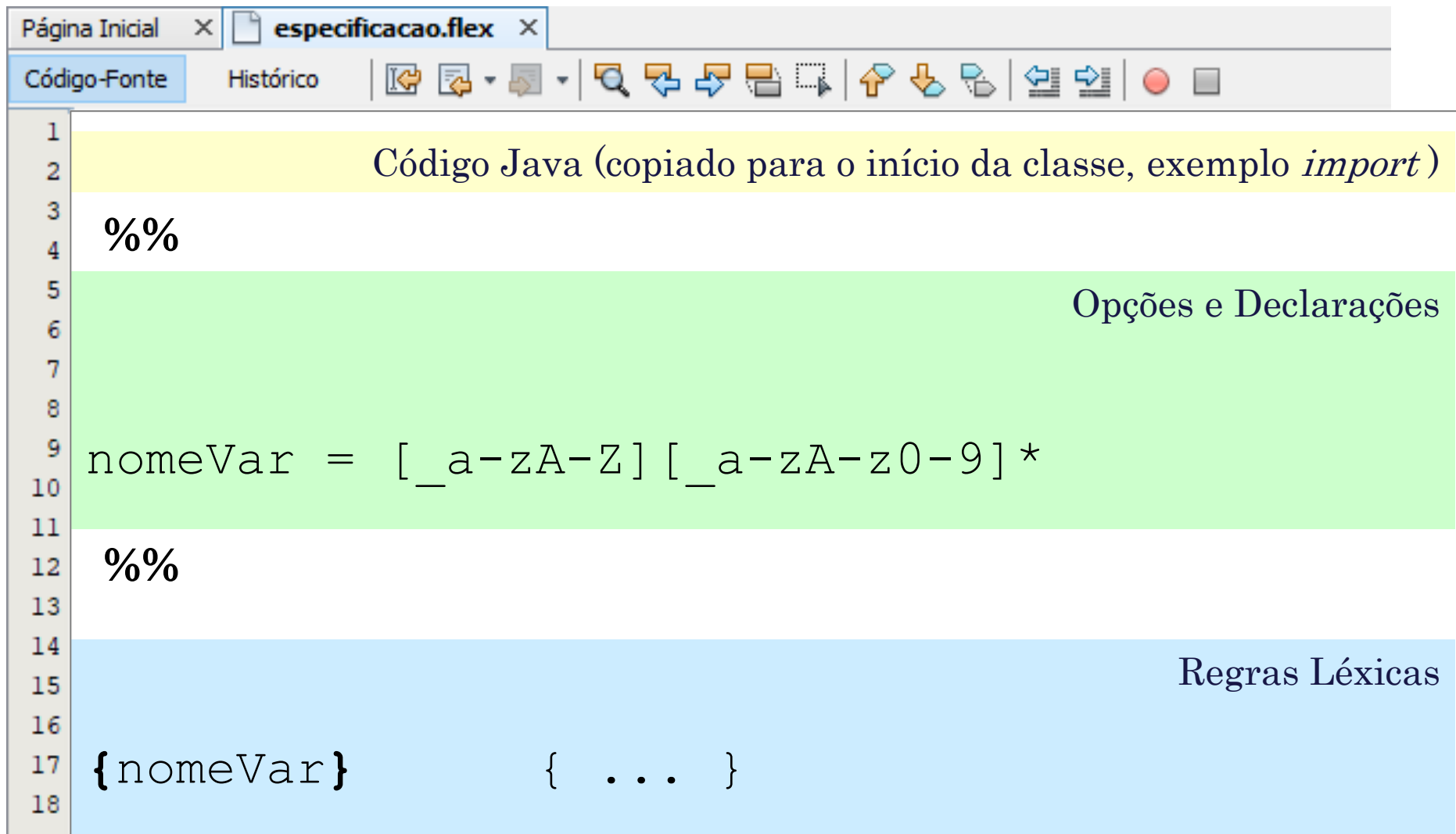
1. Crie uma especificação no JFLEX que possibilite reconhecer nomes válidos de variáveis da linguagem C.
 - **Restrições:** O único caractere especial permitido no nome da variável é o Underscore (_).
- Exemplos de entrada:

Entrada	Saída
<code>total</code>	Var: total ==> reconhecido 5 caracteres
<code>1total</code>	Outro: 1 ==> Não reconhecido Var: total ==> reconhecido 5 caracteres
<code>total1</code>	Var: total1 ==> reconhecido 6 caracteres
<code>_total</code>	Var: _total ==> reconhecido 6 caracteres
<code>1_total</code>	Outro: 1 ==> Não reconhecido Var: _total ==> reconhecido 6 caracteres

Especificação Léxica

Especificação Léxica

- A expressões regulares podem ser agrupadas em uma definição.



```
1  Código Java (copiado para o início da classe, exemplo import)
2
3  %%
4
5  Opções e Declarações
6
7
8
9  nomeVar = [_a-zA-Z] [_a-zA-z0-9] *
10
11
12  %%
13
14  Regras Léxicas
15
16
17  {nomeVar}      { ... }
18
```

Fim do arquivo

Fim do arquivo

- É uma especificação léxica que identifica o fim do arquivo:

<<EOF>> { ação }

- Quando é encontrado o fim do arquivo é executado uma ação especificada pelo programador.
 - **Atenção:** Ele executará ação infinitas vezes até que o retorno do método seja igual a NULL.

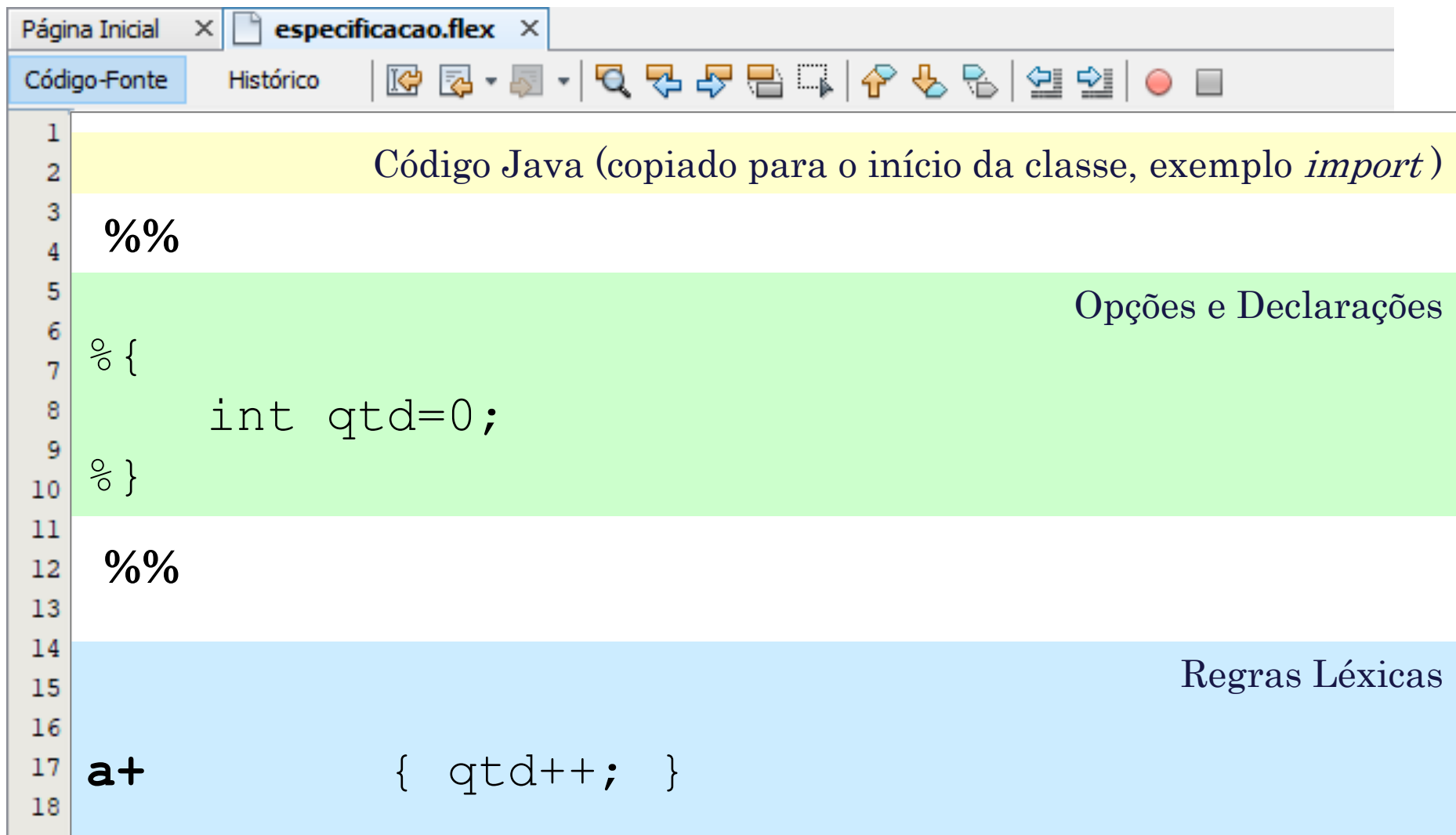
Fim do arquivo

- Também é possível especificar uma expressão regular para identificar o fim de um arquivo:
- Por exemplo:
 - Exigir que um código fonte seja encerrado com a palavra “**fim**”.
- **Importante:** Tomar cuidado com a ordem das regras especificadas para que a palavra “fim” não seja reconhecida por outra regra, por exemplo, como nome de uma variável.

Declaração de variáveis

Declaração de variáveis

- A declaração de variáveis ou métodos dentro da especificação Jflex deverá ocorrer dentro da área de “Opções e Declarações”.



```
1 Código Java (copiado para o início da classe, exemplo import)
2
3 %%
4
5 Opções e Declarações
6 % {
7     int qtd=0;
8 % }
9
10 %%
11
12 Regras Léxicas
13
14
15
16
17 a+      { qtd++; }
18
```

Exercício

Desenvolver no JFlex

- 2) Dada uma entrada apresentar quantas palavras são octais, decimais, hexadecimais e reais;

Exemplo:

Entrada: 0717 17A5 1,3 198 122 569 76B 975 45,980 A10 879

Saída:

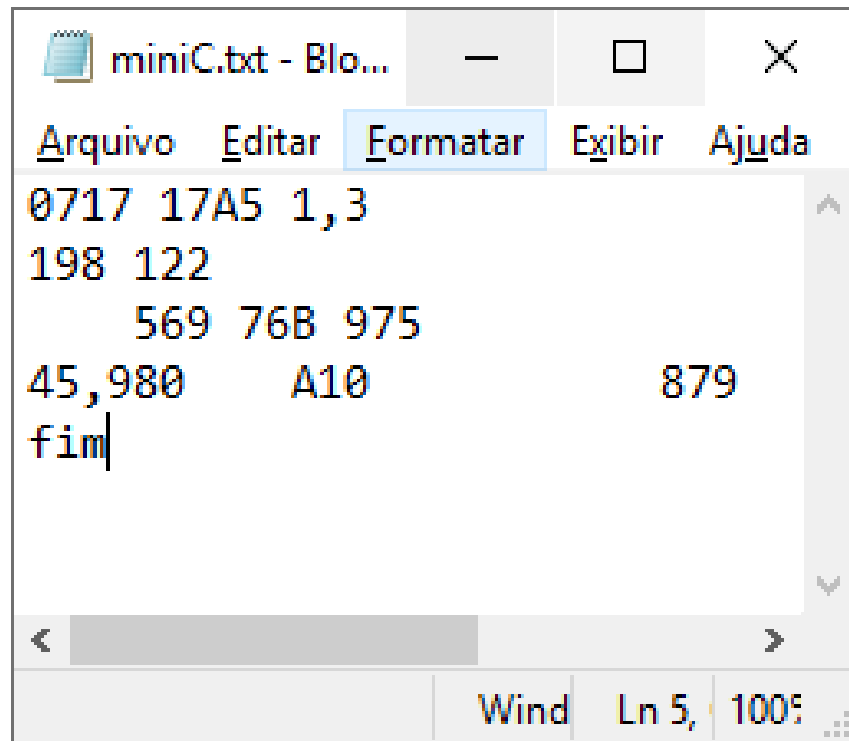
4 valores decimais

2 valores octais

3 valores hexadecimais

2 valores reais

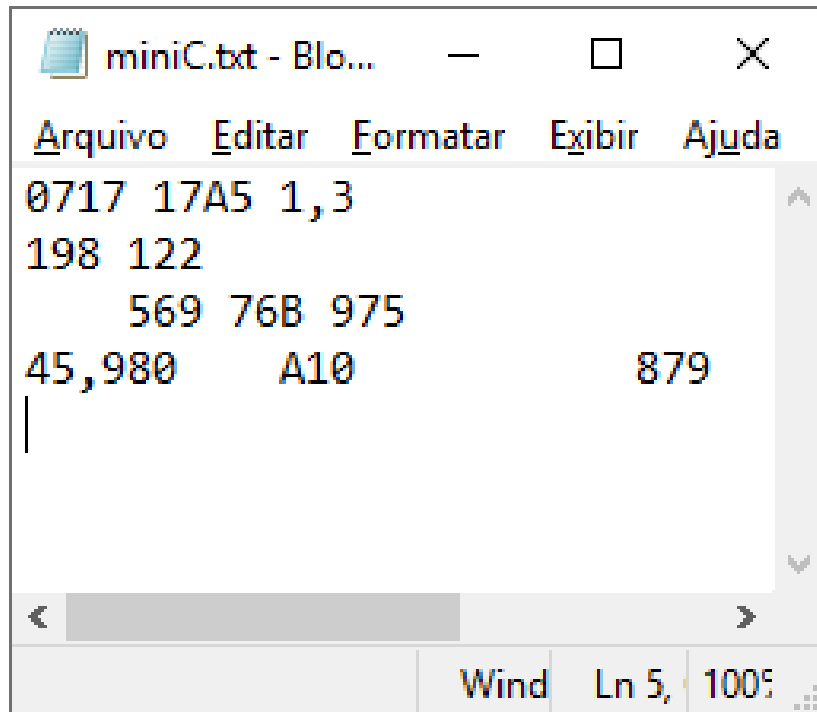
Observação: Escrever o resultado apenas quando o fim do arquivo for encontrado. Se preferir poderá definir a palavra “fim” no arquivo para encerrar a leitura.



```
0717 17A5 1,3
198 122
      569 76B 975
45,980      A10      879
fim
```

Saída:

```
run:
Octal=2 Decimal=4 Hexa=3 Real=2
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```



```
0717 17A5 1,3  
198 122  
569 76B 975  
45,980 A10 879  
|  
Wind Ln 5, 100%
```

Saída:

```
run:  
Fim inesperado do arquivo  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

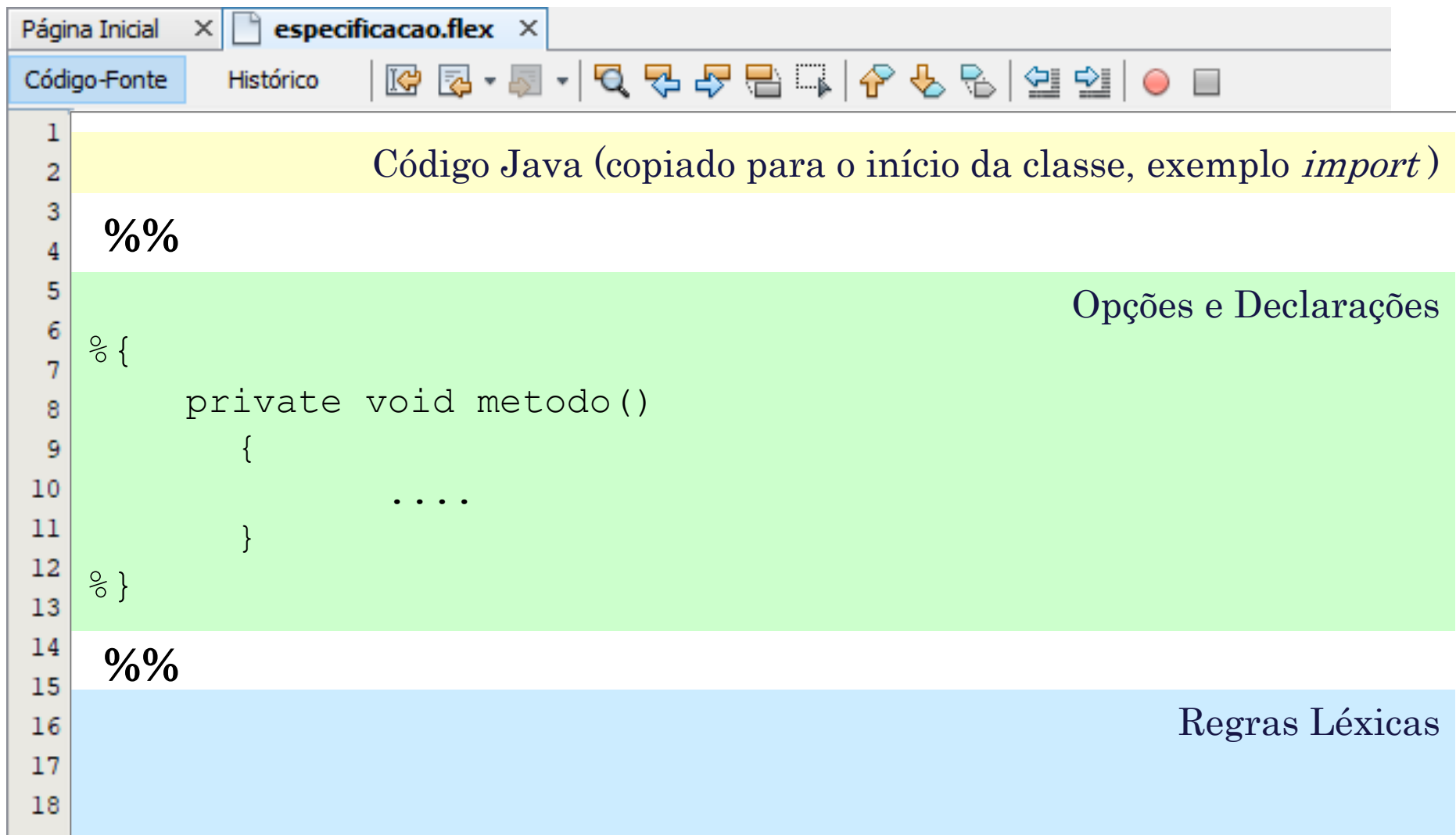
Inclusão de métodos na especificação JFlex

- No arquivo de especificação é possível inserir métodos que serão incluídos na classe que será criada pelo Jflex;
- Para isso, é necessário que o código a ser inserido esteja dentro do bloco:

```
%{  
    ... Código do método  
%}
```


Inclusão de métodos na especificação JFlex

- O bloco com o novo método deverá ser inserido na área de “Opções e Declarações”.



```
1  
2      Código Java (copiado para o início da classe, exemplo import)  
3  
4  %%  
5  
6  %{  
7  
8      private void metodo()  
9      {  
10  
11          . . . .  
12      }  
13  %}  
14  %%  
15  
16      Regras Léxicas  
17  
18
```

Exercício

3) Desenvolva 1 (um) arquivo de especificação no Jflex que reconheça os dois tipos de comentários:

- I. Separados por /* e */
- II. Iniciados por //
- III. O que não for comentário reconheça como “INVÁLIDO”.

Exemplo:

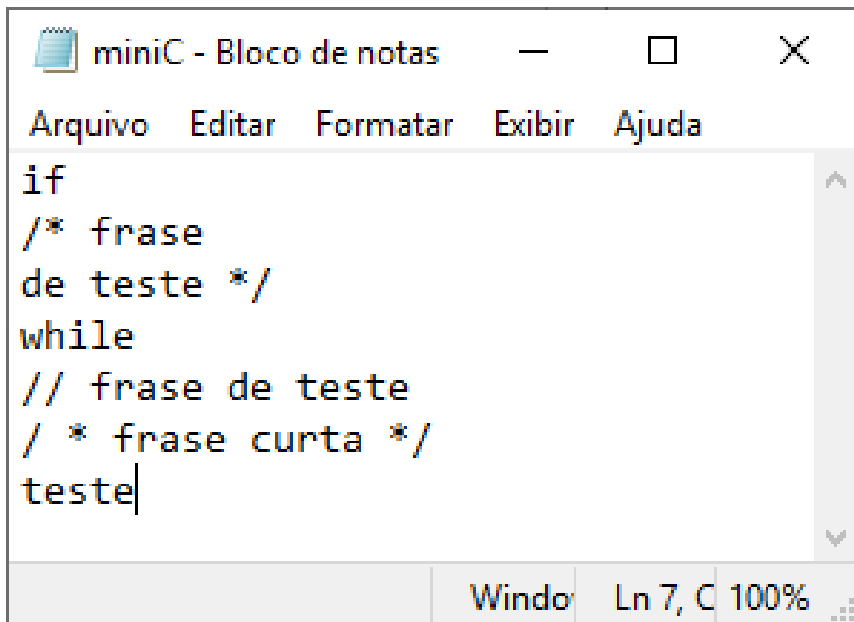
```
/* primeiro comentario */
```

```
// segundo comentário
```

```
/* terceiro comentário  
mais longo */
```

Exercício

- Exemplo de entrada e saída



```
if
/* frase
de teste */
while
// frase de teste
/ * frase curta */
teste
```

```
run:
if ==>> INVALIDO :
/* frase
de teste */ ==>> COMENTARIO :
while ==>> INVALIDO :

// frase de teste ==>> COMENTARIO :
/ * frase curta */ ==>> INVALIDO :
teste ==>> INVALIDO :
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```