

# Analizador Léxico para Linguagem de Programação JAVA

F. Murilo F. de Oliveira Filho<sup>1</sup>, Arthur Martins M. Landim<sup>2</sup>, Luid S. da Silva<sup>3</sup>

<sup>1</sup>Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)  
Av. Parque Central, 1315 - Distrito Industrial I, Maracanaú – CE – Brazil

landarthur@gmail.com

luids.silva@gmail.com

murillovskfilho2@gmail.com

**Abstract.** *This project aimed to present a brief analysis of The History of Programming Languages, as well as its evolution, the impact on society and modern applications, addressing its origins, how they developed over the years and the changes brought in technology as we know it today.*

**Resumo.** *Este projeto teve como intuito apresentar uma breve análise sobre a A História das Linguagens de Programação, bem como sua evolução, impacto na sociedade e aplicações modernas, abordando suas origens, como se desenvolveram durante os anos e as mudanças trazidas na tecnologia como conhecemos hoje.*

## 1. Manual de Instalação

### 1.1. A Linguagem Java

Java é uma linguagem de programação orientada a objetos desenvolvida na década de 90 por uma equipe de programadores chefiada por James Gosling, na empresa Sun Microsystems. Em 2008 o Java foi adquirido pela empresa Oracle Corporation. Diferente das linguagens de programação modernas, que são compiladas para código nativo, a linguagem Java é compilada para um bytecode que é interpretado por uma máquina virtual chamada de Java Virtual Machine, mais conhecida pela sua abreviação JVM (Wikipedia, 2017).

A linguagem Java foi projetada com alguns objetivos que impactaram em sua popularidade, tais como: Orientação à Objetos, baseado no modelo de Siumlar; Portabilidade, onde carrega o lema "write once, run anywhere"; Alguns Recursos de Rede com bibliotecas que facilitavam aplicações integradas à protocolos TCP/IP, como HTTP e FTP; E também a segurança, onde possuía a capacidade de executar programas na rede com restrições de execução.

É possível utilizar frameworks para facilitar o desenvolvimento de aplicações, alguns exemplos são: Hibernate, Junit e Log4j. Assim como é possível desenvolver aplicações em Java através de diversos ambientes de desenvolvimento integrado (IDEs). Dentre as mais populares, podemos citar: JBuilder, JDeveloper (desenvolvida pela empresa Oracle, atual dona da linguagem), NetBeans (software livre muito popular entre Devs) e por último, mas não menos importante, a IDE Eclipse, um projeto aberto iniciado pela IBM que é considerada por muitos o melhor ambiente para programação em Java.

## 1.2. O Analisador Léxico

Um analisador léxico, ou scanner, é um programa que implementa um autômato finito, reconhecendo (ou não) strings como símbolos válidos de uma linguagem. (RICARTE, Ivan L. M, 2014).

A implementação de um analisador léxico requer uma descrição do autômato que reconhece as sentenças da gramática ou expressão regular de interesse. Para o desenvolvimento do projeto, foram definidos todos os operadores, palavras reservadas, identificadores e letras da gramática da linguagem, como pode ser visto na imagem 1.

```
6 public class AnalisadorLexico {
7
8     private static AnalisadorLexico instance;
9
10    //Array de numeros da gramática
11    private final String[] n = {"0","1","2","3","4","5","6","7","8","9"};
12    private final ArrayList<String> num = new ArrayList<String>(Arrays.asList(n));
13
14    //Array de letras da gramática
15    private final String[] l = {"a","b","c","d","e","f","g","h","i","j",
16        "k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z"};
17    private final ArrayList<String> letras = new ArrayList<>(Arrays.asList(l));
18
19    //Array de letras da gramática Maiuscula
20    private final String[] m = {"A","B","C","D","E","F","G","H","I","J",
21        "K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z"};
22    private final ArrayList<String> letrasm = new ArrayList<>(Arrays.asList(m));
23
24    //Array de sinal da gramática
25    private final String[] op = {"(", ")", "{", "}", "[", "]", "+",
26        "-", "*", "/", "=", "<", ">", "!", "&", "&&", "|", "||", "!", "\\", " ", ".", "."};
27    private final ArrayList<String> sinais = new ArrayList<>(Arrays.asList(op));
28
29    //Array de palavras reservadas da gramática
30    private final String[] palavras = {"int","void","float","char","return",
31        "if","while","private","boolean","protected","public","final","case","else","do",
32        "for","switch","try","catch","throws","byte","char","long","double","super",
33        "this","static","abstract","class","extends","implements","interface","native","new",
34        "strictfp","synchronized","transient","volatile","break","case",
35        "continue","instanceof","assert","finally","throw","import",
36        "package","short","this","const","goto","float","boolean","true","false","null"};
37    private final ArrayList<String> palavrasReservadas =
38        new ArrayList<>(Arrays.asList(palavras));
39
```

Figure 1. Declarações das variáveis

Para a análise dos valores de entrada, foram implementados métodos que recebem o texto, percorrem cada token e fazem uma análise pra identificar seu tipo, de forma que separe os tokens em Operadores, Palavras Reservadas, Identificadores, Letras maiúsculas ou minúsculas e Números. Á partir dessa análise, cada token é salvo como seu devido tipo e impresso no output do analisador de forma que seja informado a qual grupo pertence. A imagem 2 abaixo apresenta alguns desses métodos.

```

70 }
71
72 public boolean analisar(String texto){
73     texto = texto.replaceAll("\n", "");
74     simbolos = texto.split("");
75
76     for(int i=0; i < simbolos.length; i++){
77
78         if(isLetra(simbolos[i])){
79             token += simbolos[i];
80             analisarPalavra(i);
81         }
82
83         else if(isSinal(simbolos[i])){
84             /*
85              & tem que ser seguido de outro &
86              | tem que ser seguido de outro |
87              caso contrário, é um símbolo inválido
88             */
89             if(simbolos[i].equals("&") || simbolos[i].equals("|")){
90                 if(isOperadorDuplo(i)){
91                     token += simbolos[i] + simbolos[i+1];
92                     adicionarToken("OPERADOR");
93                     i++;
94                 }
95                 else{
96                     simboloInvalido = simbolos[i];
97                     return false;
98                 }
99             }
100
101             /*
102             Se for um símbolo válido e for o último caractere
103             a ser analisado então ele é adicionado a lista de tokens
104             */

```

Figure 2. Métodos de Análise

Para medidas de segurança, também foram implementados métodos que identificam símbolos e operadores duplos, bem como foi criado um vetor de exceções que guarda um token não identificado e informa a mensagem na tela informando que a análise não foi bem sucedida. Algumas outras funções de identificação foram implementadas para integrar os métodos de análise para fins de melhoras na performance e desempenho do projeto, como demonstrado na imagem 3.

Uma classe de construtor que instancia o objeto alexico (analisador) foi criada para estabelecer atributos e funções importantes para o projeto. E por fim, foi desenvolvido uma interface gráfica utilizando a biblioteca JFrame, nativa do Eclipse, para melhorar a visualização do Analisador e seus dados de entrada e saída.

```

80     }
81
82     public String getSimboloInvalido(){
83         return simboloInvalido;
84     }
85
86     public ArrayList<String> getTokens(){
87         return this.tokens;
88     }
89
90     public void analisarPalavra(int index){
91         if((isLetra(simbolos[index])) || (isNumero(simbolos[index]))){
92
93             //System.out.println("token - "+token);
94
95             if(isPalavraReservada(token)){
96
97                 if(index == simbolos.length-1){
98                     adicionarToken("PALAVRARESERVADA");
99                 }
100
101                 else if((isEspaco(simbolos[index+1]))
102                     || (isSinal(simbolos[index+1]))){
103                     adicionarToken("PALAVRARESERVADA");
104                 }
105             }
106
107             else if(index == simbolos.length-1){
108                 adicionarToken("ID");
109             }
110
111             else if((isEspaco(simbolos[index+1])) || (isSinal(simbolos[index+1]))){
112                 adicionarToken("ID");
113             }
114         }

```

Figure 3. Métodos que tratam possíveis erros

### 1.3. Instalação

Para facilitar a instalação e importação dos códigos, o projeto está disponibilizado no github através do link disponível na referência 3. Um arquivo executável está disponível junto a uma rotina de testes dentro da pasta, onde basta clicar no ícone que o Analisador irá ser executado, pronto para iniciar os testes de funcionamento. A imagem 4 mostra a pasta incluída dentro do repositório no git, com o .EXE, .XML e um arquivo .TXT com a rotina de testes utilizada com todos os requisitos solicitados.




Nome	Data de modificação	Tipo	Tamanho
 AnalisadorLexico	16/10/2020 16:46	Aplicativo	55 KB
 executavel	16/10/2020 16:46	Documento XML	1 KB
 Testes analisador	16/10/2020 17:01	Documento de Te...	3 KB

Figure 4. Arquivos Executáveis do Analisador Léxico

## 2. Manual de Uso

Após a instalação, o arquivo do projeto pode ser importado para a IDE Eclipse para maior verificação e a pasta contendo o executável estará contida dentro do arquivo comprimido.

Ao acessar a pasta e selecionar o arquivo .EXE, a tela do analisador será aberta, como mostrado na figura 5. Para a aplicação dos testes, está disponível o arquivo .TXT



The screenshot shows a Java Swing application window titled "Análise de Código-fonte". The window has a standard title bar with minimize, maximize, and close buttons. The main content area is divided into two vertical panels. The left panel is labeled "Digite o código para análise:" and contains a large, empty text area for input. The right panel is labeled "Resultado" and also contains a large, empty text area for output. At the bottom of the window, there are two buttons: "Léxico" and "Limpar".

[illegible]

### 2.1.5. Palavras Reservadas

ArrayList contendo todas as 52 palavras reservadas da linguagem Java: "int", "void", "float", "char", "return", "if", "while", "private", "boolean", "protected", "public", "final", "case", "else", "do", "for", "switch", "try", "catch", "throws", "byte", "char", "long", "double", "super", "this", "static", "abstract", "class", "extends", "implements", "interface", "native", "new", "strictfp", "synchronized", "transient", "volatile", "break", "case", "continue", "instanceof", "assert", "finally", "throw", "import", "package", "short", "this", "const", "goto", "float", "boolean", "true", "false", "null"

## 3. Referências

UNICAMP, 2014. "Analisadores Léxicos" Disponível em: <http://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node50.html> Acesso em: 12 de Outubro de 2020

WIKIPÉDIA, 2017. "Java (linguagem de programação)" Disponível em: [https://pt.wikipedia.org/wiki/Java\\_\(linguagem\\_de\\_programa%C3%A7%C3%A3o\)](https://pt.wikipedia.org/wiki/Java_(linguagem_de_programa%C3%A7%C3%A3o)) Acesso em: 12 de Outubro de 2020

REPOSITÓRIO DO PROJETO, 2020. Disponível em: <https://github.com/murillovskfilho/AnalizadorLexicoJava> Acesso em: 12 de Outubro de 2020