UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA - UESB
CAMPUS VITÓRIA DA CONQUISTA
DEPARTAMENTO DE CIÊNCIAS EXATAS E TECNOLÓGICAS - DCET
CURSO DE CIÊNCIA DA COMPUTAÇÃO

CHRISTIAN SCHETTINE PAIVA ROCHA
TIAGO SANTOS BELA

DOCUMENTAÇÃO:
ANALISADOR LÉXICO

VITÓRIA DA CONQUISTA
2025

CHRISTIAN SCHETTINE PAIVA ROCHA
TIAGO SANTOS BELA

DOCUMENTAÇÃO:
ANALISADOR LÉXICO

Documento apresentado para a disciplina de compiladores do curso Ciência da Computação da Universidade Estadual do Sudoeste da Bahia.

Orientador: José Carlos Martins Oliveira.

VITÓRIA DA CONQUISTA
2025

SUMÁRIO

1 INTRODUÇÃO

A análise léxica é a primeira fase do processo de compilação de um programa. Ela se responsabiliza por realizar os seguintes procedimentos: mapear todos o conjunto de lexemas do código apresentado e separar tudo isso em *tokens*, os quais são os lexemas categorizados em tipos, importante destacar também que durante o processo os espaços em branco e as linhas de comentário são ignoradas, conforme, afirma Ana Maria Price em seu livro Implementação de linguagens de programação: compiladores. Esse trabalho realizado pelo analisador léxico é muito útil, e o material analisado será utilizado posteriormente pelo compilador através das fases de análise sintática e semântica do algoritmo.

Aprender esses temas é muito relevante para os estudantes, lhes darão um entendimento a respeito das questões técnicas da ciência da computação. É uma base sólida para entender como uma linguagem é analisada pelo computador e este conhecimento pode servir de base para a construção de novas ferramentas, linguagens e tecnologias ligadas à programação de computadores. Diante desse cenário, o docente da disciplina José Carlos Martins Oliveira propôs aos alunos a produção de um analisador léxico em alguma linguagem de programação como composição de nota para a disciplina de compiladores visando consolidar o entendimento dessa temática e fortalecer nos alunos a prática da programação de computadores.

2 INSTALAÇÃO

Esta seção se destina ao ensino da instalação/configuração do ambiente necessário para uso do programa.

2.1 INSTALAÇÃO DA LINGUAGEM DE PROGRAMAÇÃO

Este presente trabalho documenta um projeto de software sobre a temática da análise léxica, o projeto foi feito em java 8. É recomendado utilizar a Java SE Development Kit 8u351 para a compilação e execução do programa.

O link para a instalação da linguagem de programação pode ser acessado seguindo o seguinte link disponibilizado pela Oracle, basta escolher o seu sistema operacional e baixar: https://www.oracle.com/java/technologies/javase/javase8u211-later-archive-downloads.html.

Após baixar a versão é importante também verificar se a versão do java foi baixada corretamente, para isto basta realizar o seguinte comando:

```
java -version
```

2.2 INSTALAÇÃO DO PROGRAMA

Para instalar o programa completo basta acessar o seguinte link do Google Drive disponibilizado pela equipe: https://drive.google.com/drive/folders/1DyV9rrQSkeJH5f94fzhJoUZ1_p4pWMtU?usp=sharing.

Ao instalar e verificar basta compilar e executar o programa, para isto é necessário realizar estes procedimentos.

```
javac Principal.java
```

Este *script* irá compilar todo o código. E para executar o programa com sua devida interface é necessário realizar o seguinte procedimento.

```
java Principal
```

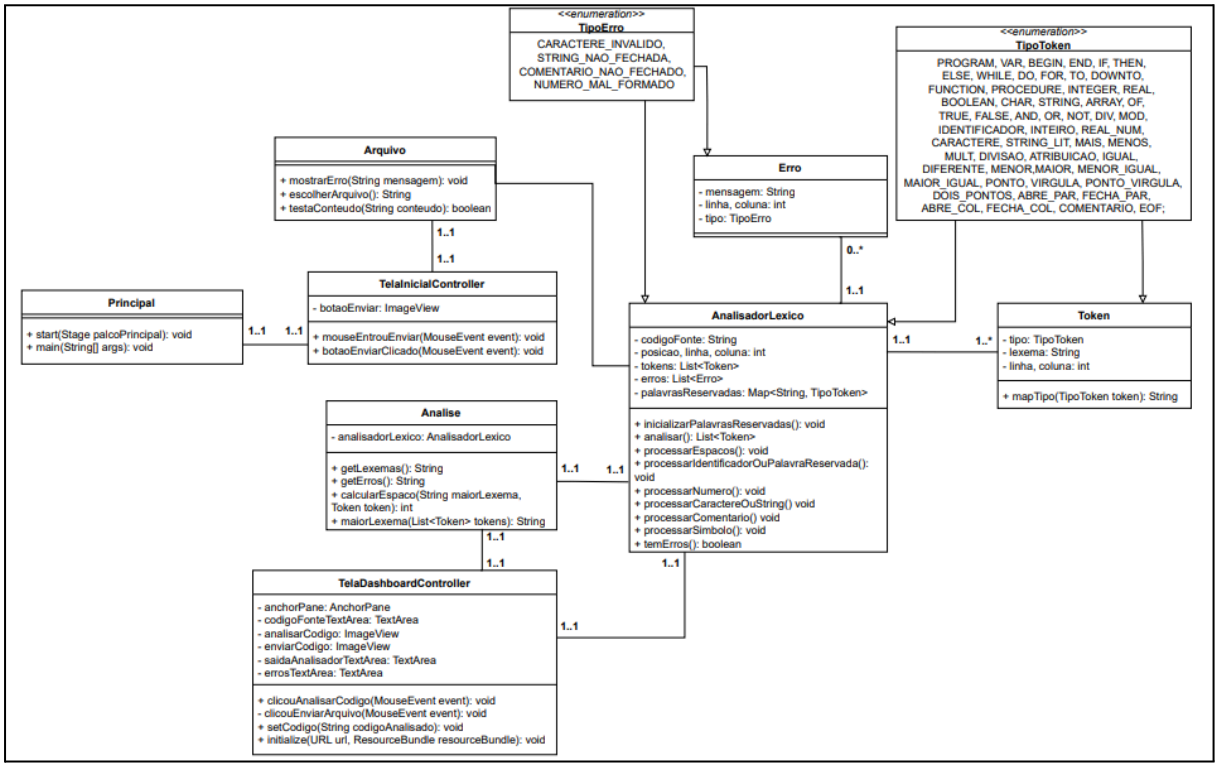
3 DIAGRAMAS

Esta seção se destina à explicação de como funciona a arquitetura do sistema e explica o funcionamento de alguns módulos presentes no código do presente artefato.

3.1 DIAGRAMA DE CLASSE

A figura 01 apresenta o diagrama de classe do sistema. O objetivo é explicar como o produto final funciona.

FIGURA 01 - DIAGRAMA DE CLASSES

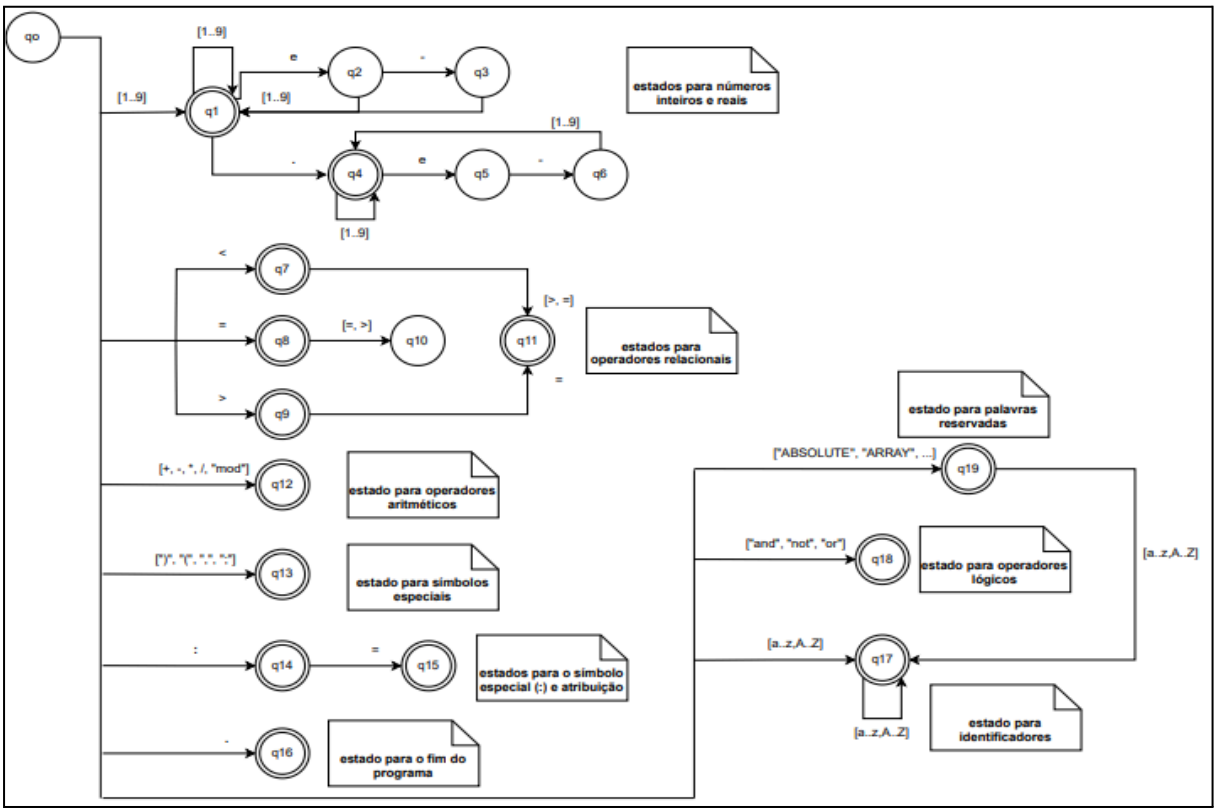


Fonte: autoria própria.

3.2 DIAGRAMA DE ESTADOS

A figura 02 apresenta o diagrama de estados do analisador léxico. O objetivo é explicar como o sistema identifica e lida com as palavras que devem ser analisadas pelo analisador.

FIGURA 02 - DIAGRAMA DE ESTADOS



Fonte: autoria própria.

Os estados q1 até o q6 do diagrama lidam com o processamento dos números do programa, no código quem lida com isso é a função *processarNumero()*, a ideia é verificar números reais e inteiros baseando-se no '.', os valores que possuem o ponto pertencem ao conjunto dos números reais e os que não possuem essa cláusula naturalmente estão contidos no conjunto dos inteiros, como o foco é identificar os lexemas e não verificar a corretude das expressões (sintaxe), esta lógica resolve o problema.

Os estados q7 até o q16 lidam com a identificação dos caracteres, símbolos e operadores, por exemplo: +, -, *, e etc. No código, quem analisa essa parte é a função *processarSimbolo()*, que possui um *switch* interno que verifica cada símbolo especificamente e adiciona uma categoria aos símbolos analisados.

Os estados q17 até o q19 lidam com as palavras que aparecem, as quais podem ser reservadas da linguagem ou não, as que não são reservadas da linguagem serão atribuídas a elas a nomenclatura de identificadores. No código quem lida com essa parte é a função

processarIdentificadorOuPalavraReservada(), todas as palavras reservadas da linguagem foram colocadas em uma estrutura do tipo *HashMap*. Com isso, ao percorrer o código-fonte, cada lexema identificado é verificado diretamente no *HashMap*. Se a palavra estiver presente neste *Hash*, ela é reconhecida como uma palavra reservada da linguagem. Caso contrário, é classificada como um identificador.

Todas as imagens podem ser visualizadas em melhor escala seguindo este link: https://drive.google.com/drive/u/0/folders/1G2BuX57G09U7pkyhcPh4xC5_gH_d-MPx

3 PROGRAMA

Esta seção se dedica a explicar como funciona a arquitetura do sistema apresentado e os elementos gráficos presentes nela.

3.1 ARQUITETURA

A arquitetura escolhida para organização do sistema foi a do *Model-View-Controller* (MVC). Essa abordagem se mostrou simples e muito eficiente, ela permite dividir a lógica o produto em *Model*, que possui os dados e a lógica de negócio, a *View* que é responsável pela interface com o usuário e o Controller, atua como intermediário, processando as requisições e coordenando as interações entre modelo e visualização, conforme afirma a Mozilla Developer Network em sua publicação a respeito do MCV.

Além dessa divisão principal da arquitetura, o projeto foi dividido em diretórios adicionais para melhor organização de arquivos auxiliares, testes, documentação e recursos gráficos, os quais serão listados nos tópicos posteriores.

3.1.1 *Model*

Nela foram implementadas as seguintes classes:

a) Analise: responsável por integrar o conteúdo de entrada com o analisador léxico, retornando os resultados processados;

b) Token: modela os lexemas reconhecidos pelo analisador, armazenando informações como tipo e valor;

c) Erro: modela os erros encontrados durante a análise léxica, com detalhes como linha, coluna e tipo de erro.

3.1.2 *View*

A camada View é responsável pela interface gráfica da aplicação. Foram utilizadas telas definidas em arquivos *Extensible Markup Language for User Interfaces* (FXML), que facilitam o desenvolvimento visual da aplicação com JavaFX:

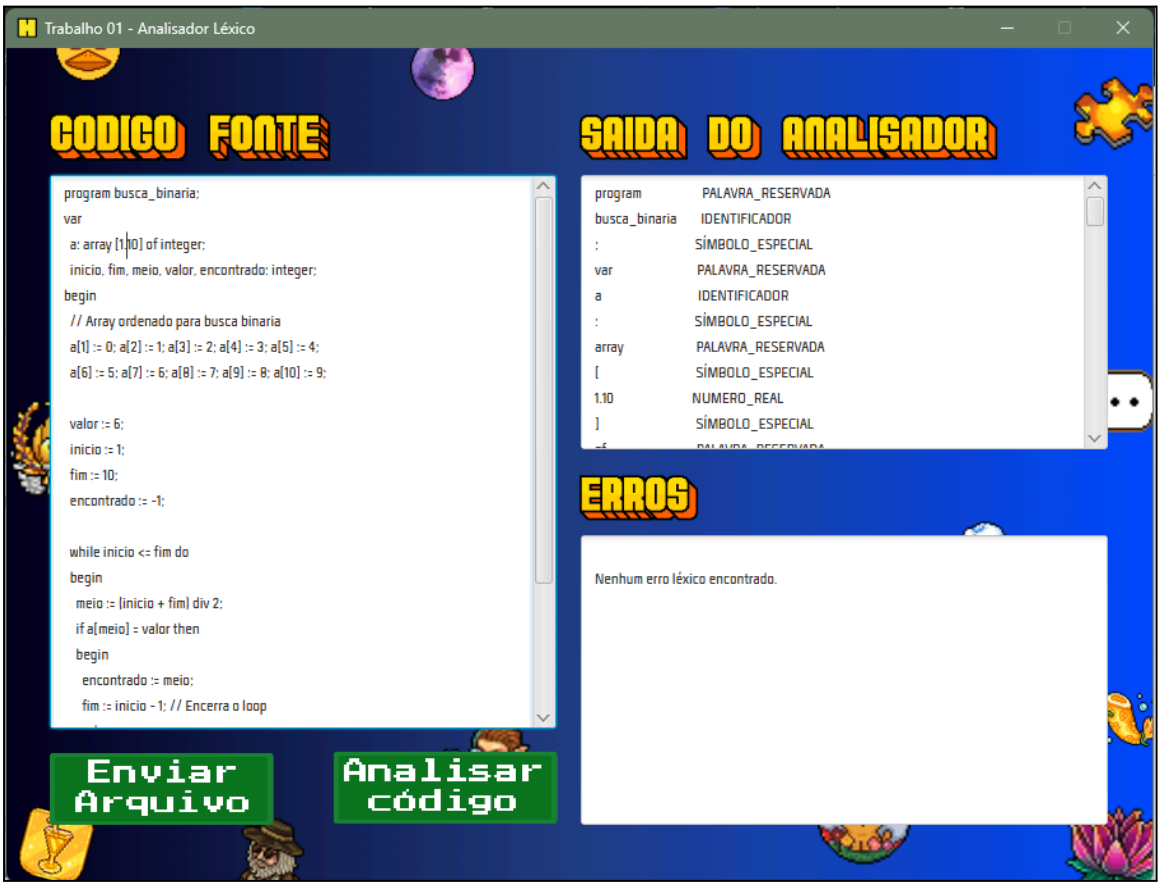
a) tela_inicial.fxml: define a interface da tela de entrada do sistema, onde a pessoa pode enviar o arquivo para análise, a figura 03 mostra essa parte.

b) tela_dashboard.fxml: representa o painel principal de análise e exibição dos resultados, a figura 04 mostra essa parte.

FIGURA 03 - TELA INICIAL

Fonte: autoria própria.

FIGURA 04 - TELA DASHBOARD



Fonte: autoria própria.

3.1.3 *Controller*

No projeto, os seguintes controladores foram implementados:

a) TelaInicialController: gerencia os eventos da tela inicial, como a abertura de arquivos ou a navegação para o painel *dashboard*;

b) TelaDashboardController: Ajuda na exibição de tokens e erros, bem como os eventos relacionados à análise.

3.1.4 *Util*

O diretório *util* contém classes auxiliares que dão suporte às funcionalidades principais do sistema. As classes são:

a) AnalisadorLexico: implementa a lógica do analisador léxico, responsável por identificar tokens, erros e categorizar lexemas.

b) Arquivo.java: gerencia a leitura de arquivos de entrada (.txt), preparando o conteúdo para análise.

c) TipoErro.java: enumeração que define os tipos possíveis de erro léxico.

d) TipoToken.java: enumeração que define os tipos de tokens válidos reconhecidos pela linguagem.

3.1.5 *Test*

O diretório *test* contém arquivos de entrada utilizados para validar o funcionamento do analisador léxico. São arquivos .txt com diferentes algoritmos, os quais usam de palavras reservadas, diferentes operadores e estruturas de controle.

3.1.6 *Doc*

O diretório *doc* armazena esta presente documentação do projeto, que possui diagramas de classe/estado e a explicação do funcionamento do programa.

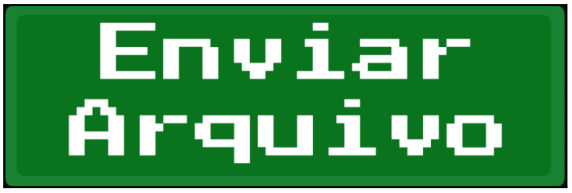
3.1.7 Img

O diretório img contém os recursos gráficos usados no projeto, como as imagens e a logotipo. As figuras posteriores mostram esses recursos usados.

FIGURA 05 - BOTÃO ANALISAR CÓDIGO

Fonte: autoria própria.

FIGURA 06 - BOTÃO ENVIAR ARQUIVO



Fonte: autoria própria.

FIGURA 07 - ÍCONE

Fonte: autoria própria.

FIGURA 08 - TELA *DASHBOARD*

Fonte: autoria própria.

FIGURA 09 - TELA INICIAL

Fonte: autoria própria.

4 CONSIDERAÇÕES FINAIS

O desenvolvimento deste projeto proporcionou um entendimento profundo a respeito dos conceitos envolvidos na construção do analisador léxico. A escolha pela arquitetura MVC contribuiu para a organização do sistema, facilitando a organização e legibilidade do aplicativo.

Durante o projeto, foi possível perceber que, apesar da análise léxica ser apenas uma das etapas iniciais de um compilador, ela possui um papel de grande importância, pois garante que a entrada seja identificada e categorizada antes de avançar para as análises sintática e semântica. Conclui-se então que este trabalho não apenas alcançou seu objetivo de construir um analisador léxico, como também serviu como base sólida para aprofundamento dos estudos em programação e contribuiu para a formação acadêmica e profissional dos participantes da equipe.

REFERÊNCIAS

BELA, Tiago. Analisador Léxico. 2025. Disponível em: https://drive.google.com/drive/u/0/folders/1G2BuX57G09U7pkyhcPh4xC5_gH_d-MPx. Acesso em: 21 set. 2025.

MOZILLA DEVELOPER NETWORK. MVC. Disponível em: https://developer.mozilla.org/en-US/docs/Glossary/MVC. Acesso em: 21 set. 2025.

ORACLE. Java SE 8 Archive Downloads (JDK 8u211 and later). Disponível em: https://www.oracle.com/java/technologies/javase/javase8u211-later-archive-downloads.html. Acesso em: 21 set. 2025.

PRICE, Ana Maria de Alencar; TOSCANI, Simão Sirineo. Implementação de linguagens de programação: compiladores. Porto Alegre: UFRGS Editora, 2000.