**FACULDADE ENGENHEIRO SALVADOR ARENA**

Marcelly Molinari Marsura – 081210020

Vinicius Benevides da Silva - 081210030

**Easy Language**

São Bernardo do Campo, São Paulo

2023

**FACULDADE ENGENHEIRO SALVADOR ARENA**

**Marcelly Molinari Marsura – 081210020**

**Vinicius Benevides da Silva - 081210030**

**Easy Language**

Relatório de Projeto de Software desenvolvido para a disciplina de Compiladores, ministrada pelo professor Israel Florentino dos Santos.

São Bernardo do Campo, São Paulo

2023

**SUMÁRIO**

[1 Introdução 4](#_Toc151657048)

[2 ANTLR 5](#_Toc151657049)

[3 Easy Language 6](#_Toc151657050)

[3.1 MainClass.java 6](#_Toc151657051)

[3.2 EasyLanguageLexer 7](#_Toc151657052)

[3.3 EasySymbol 7](#_Toc151657053)

[3.4 EasyVariable 7](#_Toc151657054)

[3.5 AbstractCommand 7](#_Toc151657055)

[3.6 EasyProgram 8](#_Toc151657056)

[3.7 CommonTokenStream 9](#_Toc151657057)

[3.8 EasyLanguageParser 9](#_Toc151657058)

[3.9 EasyLanguage.g4 10](#_Toc151657059)

[4 Definição Comando PEC 11](#_Toc151657060)

[5 Conclusão 15](#_Toc151657061)

[6 Referências Bibliográficas 16](#_Toc151657062)

# Introdução

O objetivo central desse trabalho se deu na efetiva aplicação e desenvolvimento dos conceitos aprendidos nas disciplinas de Compiladores e Economia. Nesse sentido, focalizou-se o planejamento e desenvolvimento de um framework nativo na chamada Easy Language que permitisse o cálculo do Ponto de Equilíbrio Contábil (PEC).

A documentação que se segue detalha o processo lógico de compilação da linguagem Easy Language, utilizando o ANTLR (Another Tool for Language Recognition). O ANTLR é um poderoso gerador de analisadores sintáticos que desempenha um papel crucial na análise, processamento e tradução de textos estruturados.

Este trabalho explora a aplicação do ANTLR na criação da linguagem Easy Language, fornecendo uma visão abrangente das classes principais e do fluxo de compilação. Ao longo do documento, serão destacadas as funcionalidades-chave do ANTLR e sua contribuição na construção eficiente de linguagens de programação.

Com o uso do projeto final, será possível a declaração de uso do método nativo *PEC*. O seu cálculo é uma ferramenta crucial na análise financeira de uma empresa. Ele representa o ponto em que as receitas totais se igualam aos custos totais, resultando em lucro zero. Em outras palavras, o PEC indica o nível de vendas ou atividade em que uma empresa não está nem ganhando nem perdendo dinheiro.

Para tal objetivo, será apresentado a análise de funcionamento da ferramenta ANTLR, seu uso para esse projeto e o resultado final obtido.

# ANTLR

O ANTLR (Another Tool for Language Recognition) é um gerador de analisar sintático que usa um algoritmo LL para análise. É o sucessor do Purdue Compiler Construction Tool Set, desenvolvido pela primeira vez em 1989, e está em desenvolvimento ativo.

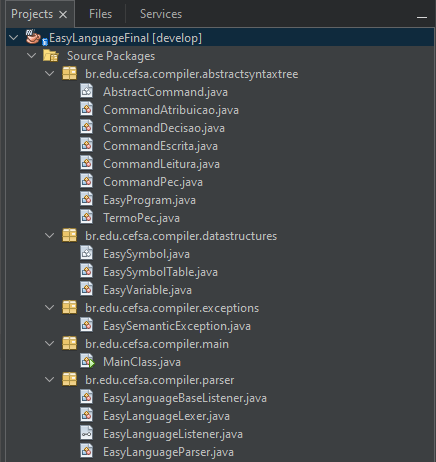
Assim, está é uma poderosa ferramenta para leitura, processamento, execução e tradução de textos estruturados ou arquivos binários. É amplamente utilizada para construir linguagens, ferramentas e estruturas.

Sua principal funcionalidade está em, através de uma gramática, gerar um analisador capaz de construir árvores de análise ou ainda uma interface ouvinte/visitante que facilita a resposta ao reconhecimento de frases de interesse.

No exemplo apresentado, o ANTLR foi usado para criar a linguagem Easy Language, com comandos básicos como leitura, escrita, atribuição e decisão, além de um comando personalizado referente ao Cálculo do PEC (Ponto de Equilíbrio Contábil).

# Easy Language

A seguir, serão apresentadas as principais classes presentes no projeto:



## MainClass.java

Nessa classe são instanciados o Lexer e o Parser. Estes são componentes essenciais em um processo de compilação, responsáveis por analisar e processar o código fonte de uma linguagem.

O Lexer (Analisador léxico) recebe o código fonte e o divide em unidades chamadas tokens. Estes são partes indivisíveis do código, como palavras-chave, identificadores, números, símbolos, etc. Ele remove também espaços em branco desnecessários, classifica os tokens e retorna a sua sequência como saída.

Por sua vez, o Parser (Analisador Sintático) consome a sequência de tokens e a analisa seguindo suas regras sintáticas, construindo a hierarquia do código (em uma estrutura, geralmente uma árvore sintática).

Para esse código em específico, são utilizadas classes customizadas para realizar essas funções. O EasyLanguageLexer herda de Lexer, enquanto o EasyLanguageParser herda de Parser (ambas classes abstratas do pacote org.antlr.v4.runtime).

## EasyLanguageLexer

Inicialmente, o código fonte contido em ./resources/input.easy é carregado como um CharStream, uma fonte de caracteres que será usada pelo Lexer. Esse é passado no construtor do **EasyLanguageLexer** (que o encaminha ao construtor da classe Lexer). A partir dele que serão obtidos os tokens.

## EasySymbol

O Lexer criado possuí um atributo EasySymbolTable. Essa é uma classe com a finalidade de armazenar os **EasySymbols** do programa atual. Ela possui um HashMap que armazena o nome da variável e sua informação em si.

## EasyVariable

A classe **EasyVariable** efetivamente representa uma variável do programa, implementando a classe abstrata EasySymbol. Uma variável, além do nome, possuí ainda o seu tipo (número ou texto) e seu valor. Também possuí um método generateJavaCode() : String. Nele, é retornado um código java referente à declaração dessa variável.

Por exemplo, ao declarar uma variável “numero a;”, será criado uma EasyVariable { name = a, type = NUMBER, value = null }, que retorna como java code a String “double a;”. Esse objeto será armazenado na EasySymbolTable com nome (a) como chave.

## AbstractCommand

O Lexer possuí uma pilha de listas de comandos abstratos (Stack<ArrayList<AbstractCommand>>), chamada stack. **AbstractCommand** é uma classe abstrata que exige a implementação do método generateJavaCode() : String. Semelhante ao gerado por EasySymbol, este irá retornar o código java correspondente a um comando.

Por exemplo, a classe *CommandEscrita* a implementa, definindo um atributo id. Assim, quando for definido o comando “escreva(a)”, ele será retornado como “System.out.println(a)”, sendo “a” a String armazenada por id.

Todos os comandos definidos possuem essa estrutura:

* **CommandAtribuicao**: Possuí os atributos id (String) e expr (String) e gera “id = expr;”
* **CommandDecisao**: Possuí uma condição (String), uma lista de comandos true e outra de comandos false (ambas ArrayList<AbstractCommand>). Para o seu código java, é concatenado “if(condição) { ” + código java dos comandos true + “}”. Caso existam comandos false, é ainda concatenado “else { ” + código em java dos comandos false + “ } ”.
* **CommandEscrita**: Possuí apenas um id (String). Retorna “System.out.println(id);”
* **CommandLeitura**: Possuí um id (String) e uma var (EasyVariable). Como ele é usado para atribuir o valor de var, é utilizada a classe java.util.Scanner para se realizar a leitura. Se o tipo da variável for NUMBER é retornado “id = \_key. nextDouble();”, enquanto para TEXT, “id = \_key. nextLine ();”. \_key é um objeto Scanner declarado na classe EasyProgram, usado para leitura.

## EasyProgram

Por fim, o lexer também possuí uma instância da classe **EasyProgram**. Essa é a classe base do programa que gera o código java no chamado target (arquivo localizado em ./resources/MainClass.java).

Por exemplo, ao final da validação do código:

programa

numero a;

a = 4;

escreva (a);

fimprog;

Será armazenado nesse arquivo:

import java.util.Scanner;

public class MainClass{

public static void main(String args[]){

Scanner \_key = new Scanner(System.in);

double a;

a = 4;

System.out.println(a);

}

}

Cada comando da EasyLanguage é convertido para sua respectiva versão em java. A classe EasyProgram possuí o método generateTarget() : void responsável por, a partir de sua lista de Símbolos (EasySymbolTable) e comandos (ArrayList<AbstractCommand>), gerar o código java na MainClass.java.

Essa classe possuí ainda:

* Direcionamento para o arquivo da gramática (“EasyLanguage.g4”),
* Variáveis estáticas e finais do tipo inteiro definindo as regras (T\_\_0=1, T\_\_1=2, T\_\_2=3, T\_\_3=4, T\_\_4=5, T\_\_5=6, T\_\_6=7, T\_\_7=8, AP=9, FP=10, SC=11, OP=12, ATTR=13, VIR=14, ACH=15, FCH=16, OPREL=17, ID=18, NUMBER=19, WS=20;)
* Um vetor de String com o nome das regras ("T\_\_0", "T\_\_1", "T\_\_2", "T\_\_3", "T\_\_4", "T\_\_5", "T\_\_6", "T\_\_7", "AP", “FP", "SC", "OP", "ATTR", "VIR", "ACH", "FCH", "OPREL", "ID", "NUMBER", "WS")
* Um vetor de String com os nomes literais (null, "'programa'", "'fimprog;'", "'numero'", "'texto'", "'leia'", "'escreva'", "'se'", "'senao'", "'('", "')'", "';'", null, "'='", "','", "'{'", "'}'")
* Um Vetor de String com os nomes simbólicos (null, null, null, null, null, null, null, null, null, "AP", "FP", "SC", "OP", "ATTR", "VIR", "ACH", "FCH", "OPREL", "ID", "NUMBER", "WS")

## CommonTokenStream

Uma vez com o Lexer configurado, é gerado um fluxo de tokens que efetivamente será encaminhado para o Parser.

## EasyLanguageParser

Efetivamente o Parser do compilador. Recebe o TokenStream contendo o fluxo de tokens do Lexer e declara de forma similar a EasySymbolTable, o EasyProgram e uma pilha de listas de AbstractCommand.

Quando configurado, é chamado o método prog() : ProgContext. Esse método é o responsável por navegar pelas regras definidas na gramática e realizar a verificação e validação dos tokens.

Por exemplo, ao ser iniciado, ele espera encontrar a palavra chave “programa”. Caso não a encontre, será reportado um erro de sua ausência. Assim, será efetivamente realizado o processo de compilação do código.

Ao final, a Main invoca os métodos exibeComandos e generateCode do Parser. O primeiro é responsável por imprimir no console um log com o fluxo de comandos e atribuições identificadas ao longo da compilação, enquanto o segundo altera o arquivo contido em “./resources/MainClass.java” para o código convertido.

## EasyLanguage.g4

Por fim, destaca-se o arquivo EasyLanguage.g4. Ele contém a gramática da qual se originará o Lexer e o Parser.

Sua estrutura inicia-se com a realização dos imports necessários ao seu funcionamento, como por exemplo os elementos da árvore de sintaxe abstrata (CommandLeitura, CommandEscrita, etc.).

Declara ainda variáveis auxiliares a leitura das entradas, como identificadores de leitura e escrita que posteriormente referenciarão as variáveis declaras, além da pilha de comando, lista da Thread atual, tabela de variáveis (EasySimbol) e lista de comandos verdadeiros e falsos a serem executados no comando de decisão.

Em seguida, estão presentes as regras que definem essa gramática, no formato:

* nomeRegra : tokens { ações } ;

Por exemplo, a regra prog é definida da seguinte forma:

prog : 'programa' decl bloco 'fimprog;'

{

program.setVarTable(symbolTable);

program.setComandos(stack.pop());

}

;

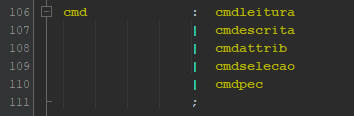
Para sua execução, deve ser antes declarado a palavra chave “programa”, seguida de uma decl (declaração das variáveis a serem utilizadas) e bloco (comandos a serem executados sobre essas variáveis), finalizado com a palavra chave “fimprog”.

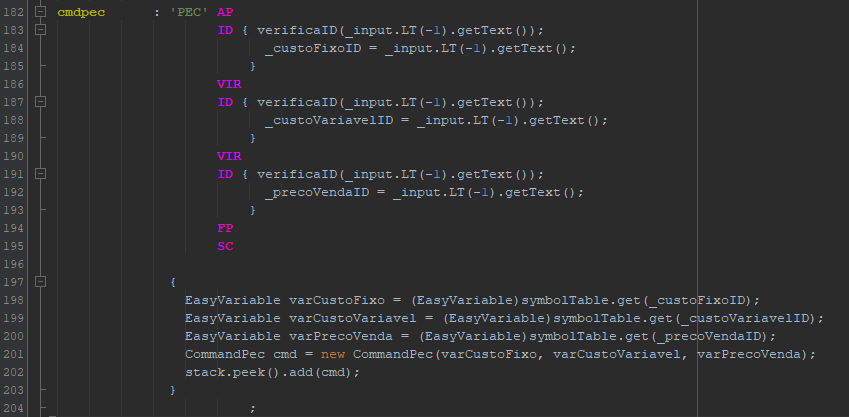
Além disso, ao final da sua execução, é atribuída a tabela de variáveis do programa em si a symbolTable e definidos os comandos como o topo da pilha de execução.

# Definição Comando PEC

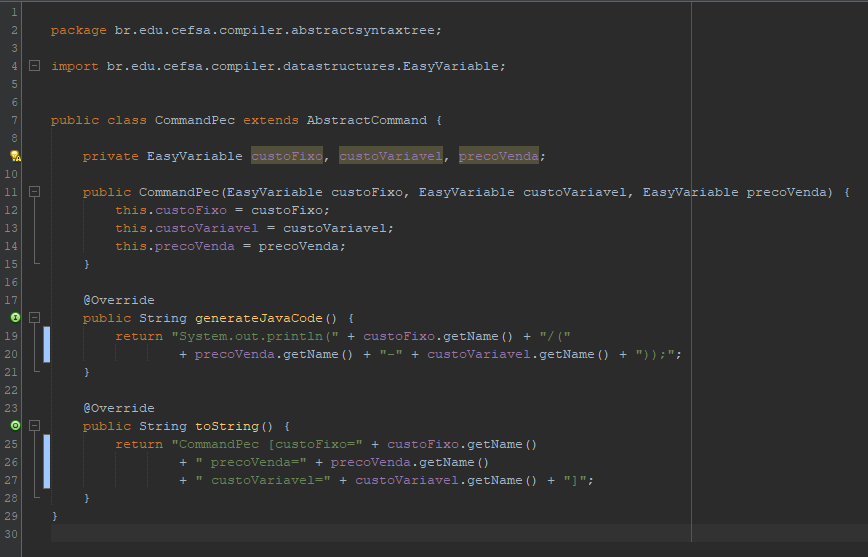
Dessa forma, diante da explicação referente à lógica de compilação empregada, é possível a definição de novas funcionalidades, fluxos de palavras chave por meio da alteração desse projeto. A seguir está apresentada a sequência de passos para criação do comando PEC (Ponto de Equilíbrio Contábil). A formula utilizada foi:

1. Criação da regra gramatical no arquivo EasyLanguage.g4 como um novo tipo de comando





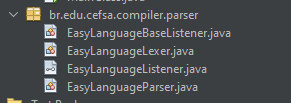
1. Criação da classe CommandPec



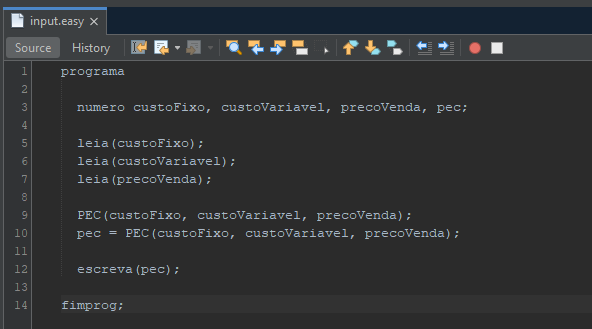
1. Execução do comando java -jar antlr-4.12.0-complete.jar EasyLanguage.g4

Esse comando deve ser executado no diretório .\Easy Language\resources, o qual possuí os arquivos antlr4-4.12.0.jar e EasyLanguage.g4. Com sua execução, serão criados os novos Parser, Lexer e Listener para o projeto

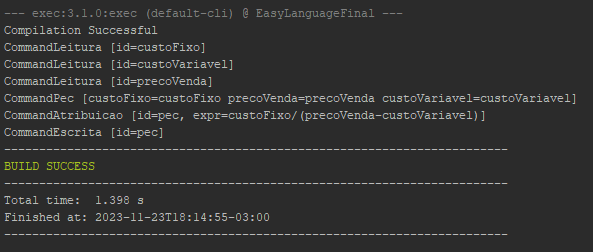
1. Substituição no package compiler.parser



1. Alteração do arquivo input.easy

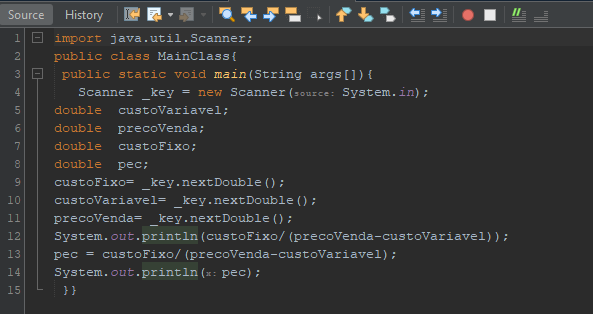


1. Execução da classe MainClass



Ao final desse projeto, foram desenvolvidas duas variações do comando PEC: Uma a ser utilizada para exibição dos valores no prompt (usando System.out.println), e outra para ser usada na atribuição de variáveis (conforme expressa na imagem assim no CommandAtribucao).

1. Conferência do Arquivo MainClass.java gerado



Pode-se reparar que, conforme esperado, o sistema converter os comandos PEC de escrita e atribuição para seus respectivos formatos em java explícitos nas suas classes respectivas.

# Conclusão

Em conclusão, este trabalho proporcionou uma oportunidade valiosa para aplicar os conhecimentos adquiridos nas disciplinas de Compiladores e Economia de forma prática e significativa. Através da análise e desenvolvimento baseados na utilização da ferramenta ANTLR para a compilação da linguagem Easy Language, foi possível criar uma aplicação contábil para cálculo do PEC.

Assim, demonstrou-se sua versatilidade e eficácia na construção de analisadores sintáticos. A análise detalhada das classes, como EasyLanguageLexer, EasyVariable, AbstractCommand, e EasyProgram, ilustra a estrutura interna do compilador e como cada componente desempenha um papel crucial na geração do código Java correspondente.

Além disso, a introdução do comando personalizado PEC (Ponto de Equilíbrio Contábil) destaca a flexibilidade do ANTLR, permitindo a expansão da linguagem com a criação de novas regras gramaticais. Este trabalho oferece uma visão abrangente do fluxo de compilação da linguagem Easy Language, fornecendo insights valiosos para desenvolvedores interessados em construir linguagens de programação utilizando o ANTLR.

# Referências Bibliográficas

PARR, Terence. ANTLR. Disponível em: https://www.antlr.org/. Acesso em: 19 nov. 2023.

OSÓRIO, Victor. Como criar uma linguagem usando ANTLR4 e Java. 2020. Disponível em: https://vepo.medium.com/como-criar-uma-linguagem-usando-antlr4-e-java-ad834fadc2c1. Acesso em: 19 nov. 2023.

BAELDUNG (org.). Java with ANTLR. 2020. Disponível em: https://www.baeldung.com/java-antlr. Acesso em: 19 nov. 2023.