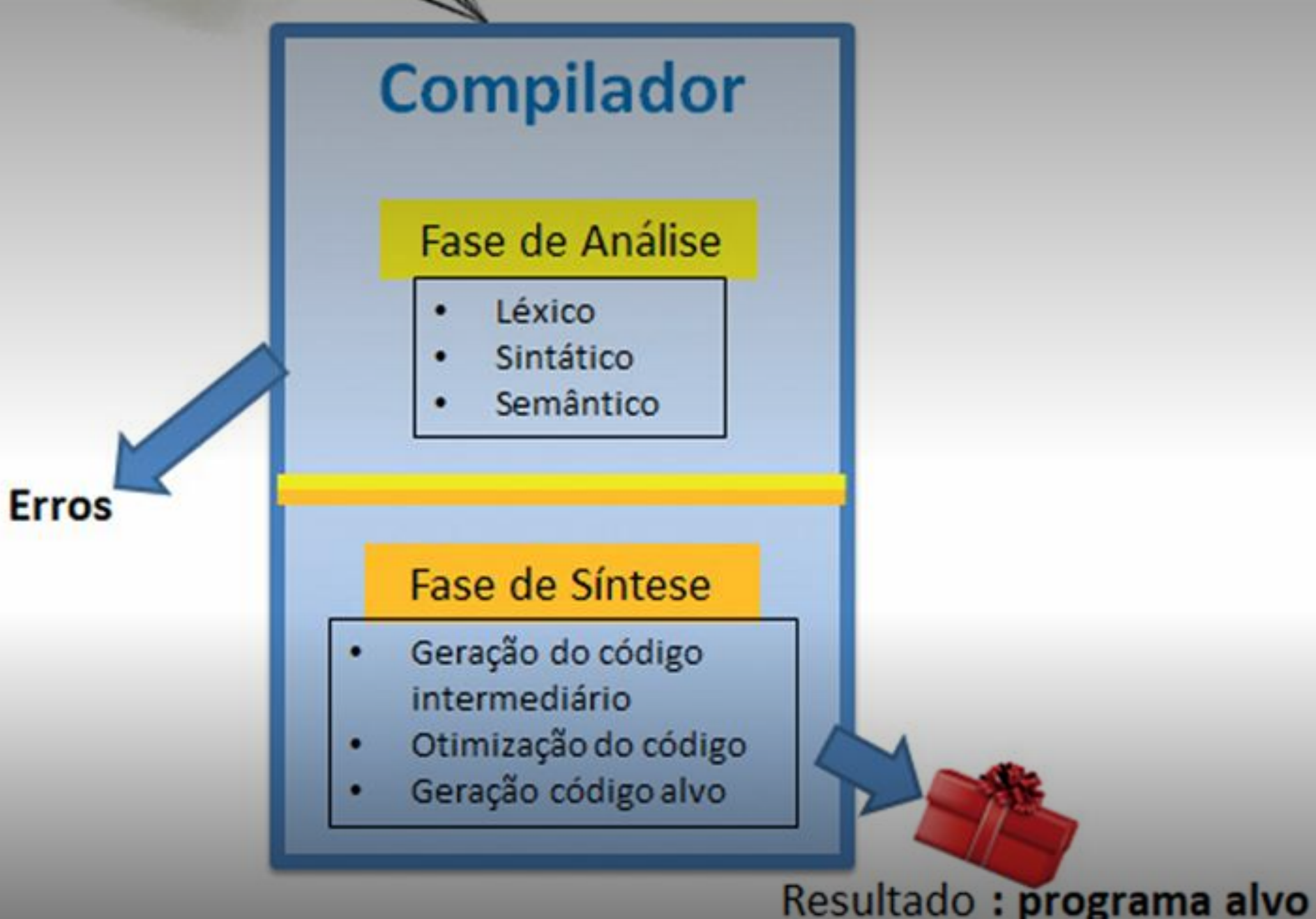
COMPILADORES

Prof. Me.Wanderlan Albuquerque

ESTUTURA DO COMPILADOR


Compilador
Fase de Análise
Léxico
Sintático
Semântico
Erros
Fase de Síntese
Geração do código intermediário
Otimização do código
Geração código alvo
Resultado : programa alvo

Tipos de tradutores

1 Montadores

Programas que traduzem um código fonte escrito em linguagem básica (assembly) em código de máquina, também denominado Assembler.

2 Compiladores

Traduzem o código fonte, programa escrito em uma linguagem de alto nível, em código alvo.

3 Interpretadores

Fazem o mesmo processo do compilador, mas cada comando (linha de programa) analisado executa todas as fases de tradução até alcançar o código alvo e já o executa.

4 Compiladores híbridos

Seguem todos os passos de um compilador, mas não geram o código executável, e sim um código intermediário que será executado por uma máquina virtual.

FASE DE ANÁLISE

A fase de análise está diretamente associada à verificação de se o programa foi escrito corretamente, isto é, de acordo com as regras da linguagem.

FASE DE ANÁLISE

A análise está subdividida em 3 etapas, segundo Aho (2007):

Léxica: em que se verifica se os nomes das entidades estão corretos.

Sintática : analisa se os comandos estão corretos e sua estrutura.

Semântica : avalia os valores envolvidos, tipos.

Cada compilador traduz conforme a linguagem

Visão Geral da Análise de Código

A análise de código é o processo de verificar se um programa foi escrito corretamente, seguindo as regras da linguagem de programação. Este processo é essencial para a correta execução de um programa. Sem uma análise adequada, erros de sintaxe, semântica ou léxicos podem impedir o funcionamento adequado do software.

Léxica

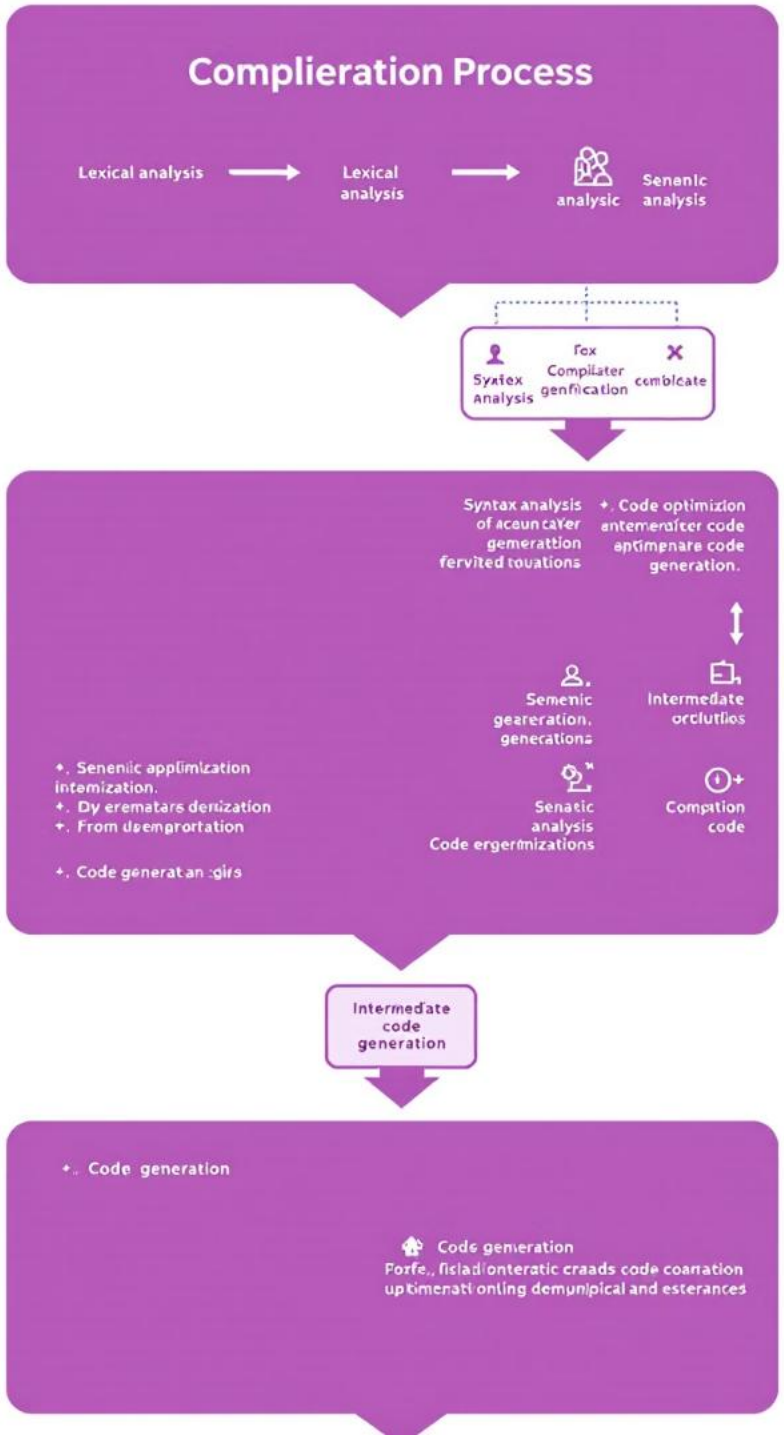
Verifica a correção dos nomes das entidades (variáveis, funções, etc.).

Sintática

Analisa a correção da estrutura dos comandos.

Semântica

Verifica a compatibilidade dos tipos de dados e o contexto geral.



Análise Léxica: Identificando Tokens

A análise léxica é a primeira etapa da análise de código, onde o código-fonte é dividido em **tokens**, que são as unidades básicas da linguagem (palavras-chave, identificadores, operadores, etc.). O analisador léxico verifica se os nomes das entidades estão corretos, seguindo as regras da linguagem.

Exemplo Teórico

Em Java, um identificador deve começar com uma letra, um underscore (_) ou um cifrão ($). Não pode começar com um número. Uma palavra-chave como **int** deve ser reconhecida como tal.

Exemplo Prático (Java)

```
int idade = 25; // Válido
int 2idade = 25; // Inválido
```

Análise Sintática: Verificando a Gramática

A análise sintática verifica se a estrutura dos comandos está correta, de acordo com a gramática da linguagem. É como verificar se uma frase está gramaticalmente correta em português. O analisador sintático constrói uma árvore sintática para representar a estrutura do código.

Exemplo Teórico

Em Java, um comando de atribuição deve seguir a forma *variável = expressão;*. Se a ordem ou a sintaxe estiver incorreta, um erro é reportado.

Exemplo Prático (Java)

```
idade = 25; // Válido
25 = idade; // Inválido
```

Análise Semântica: Avaliando o Contexto

A análise semântica verifica se o contexto do código está correto, ou seja, se os tipos de dados são compatíveis, se as variáveis foram declaradas, etc. O analisador semântico usa a árvore sintática para realizar essas verificações.

Exemplo Teórico

Se uma variável **idade** foi declarada como **int**, não se pode atribuir um valor do tipo **String** a ela. Isso geraria um erro semântico.

Exemplo Prático (Java)

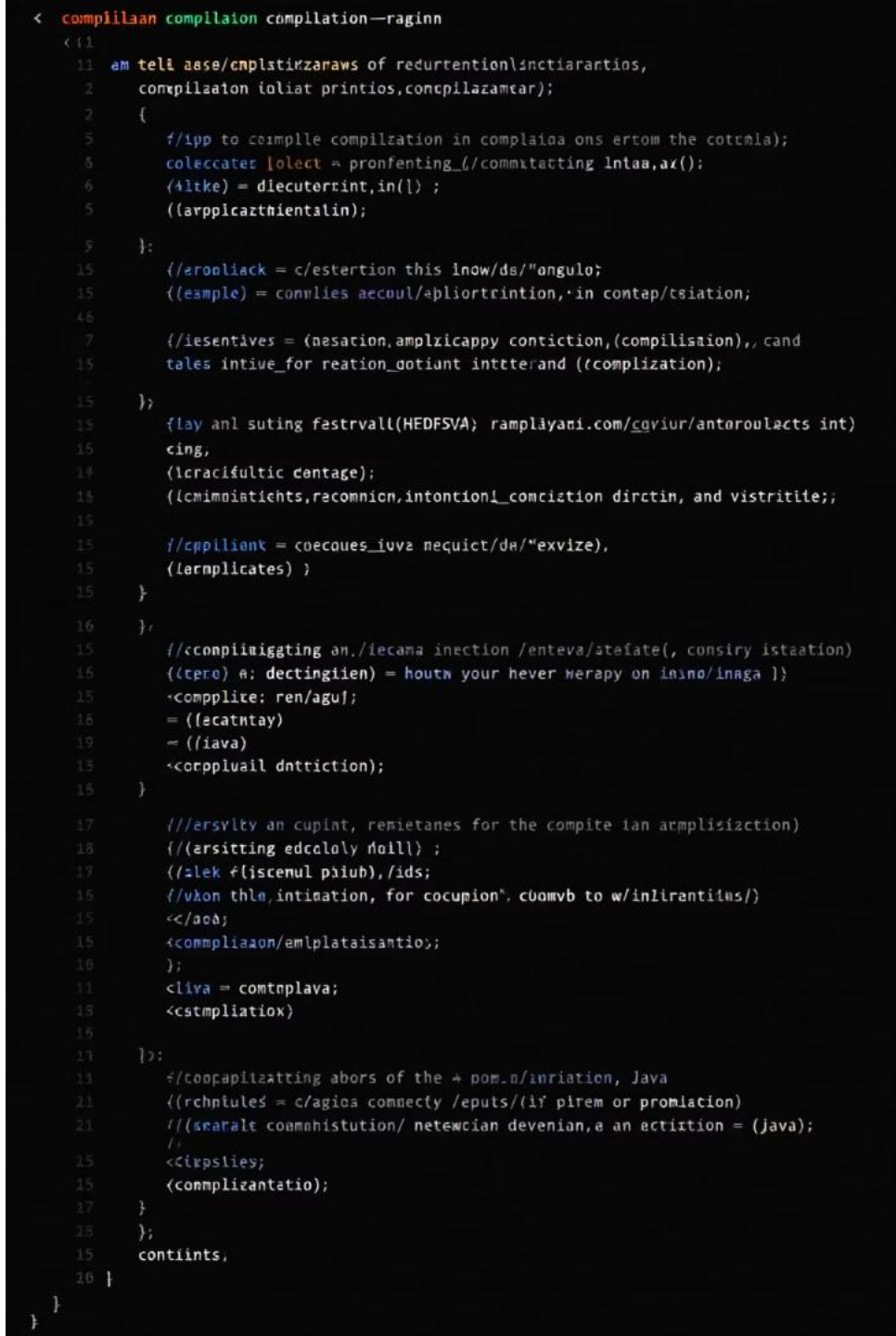
```
int idade = 25; // Válido
idade = "vinte e cinco"; // Inválido
```

Exemplo Completo em Java: Análise Integrada

Vamos analisar um exemplo completo em Java para ilustrar como as três etapas da análise (léxica, sintática e semântica) trabalham juntas para garantir a correção do código.

```
public class Exemplo {
    public static void main(String[] args) {
        int idade = 25;
        String nome = "João";
        System.out.println("Nome: " + nome + ", Idade: " +
idade);
    }
}
```

Neste exemplo, todas as etapas da análise são bem-sucedidas, e o código é considerado correto.



Análise Léxica no Exemplo

O analisador léxico divide o código em tokens como **public**, **class**, **Exemplo**, **int**, **idade**, **=**, **25**, **String**, **nome**, **"João"**, **System**, **.**, **out**, **.**, **println**, etc. Cada token é classificado e verificado quanto à sua validade.

1 Palavras-chave

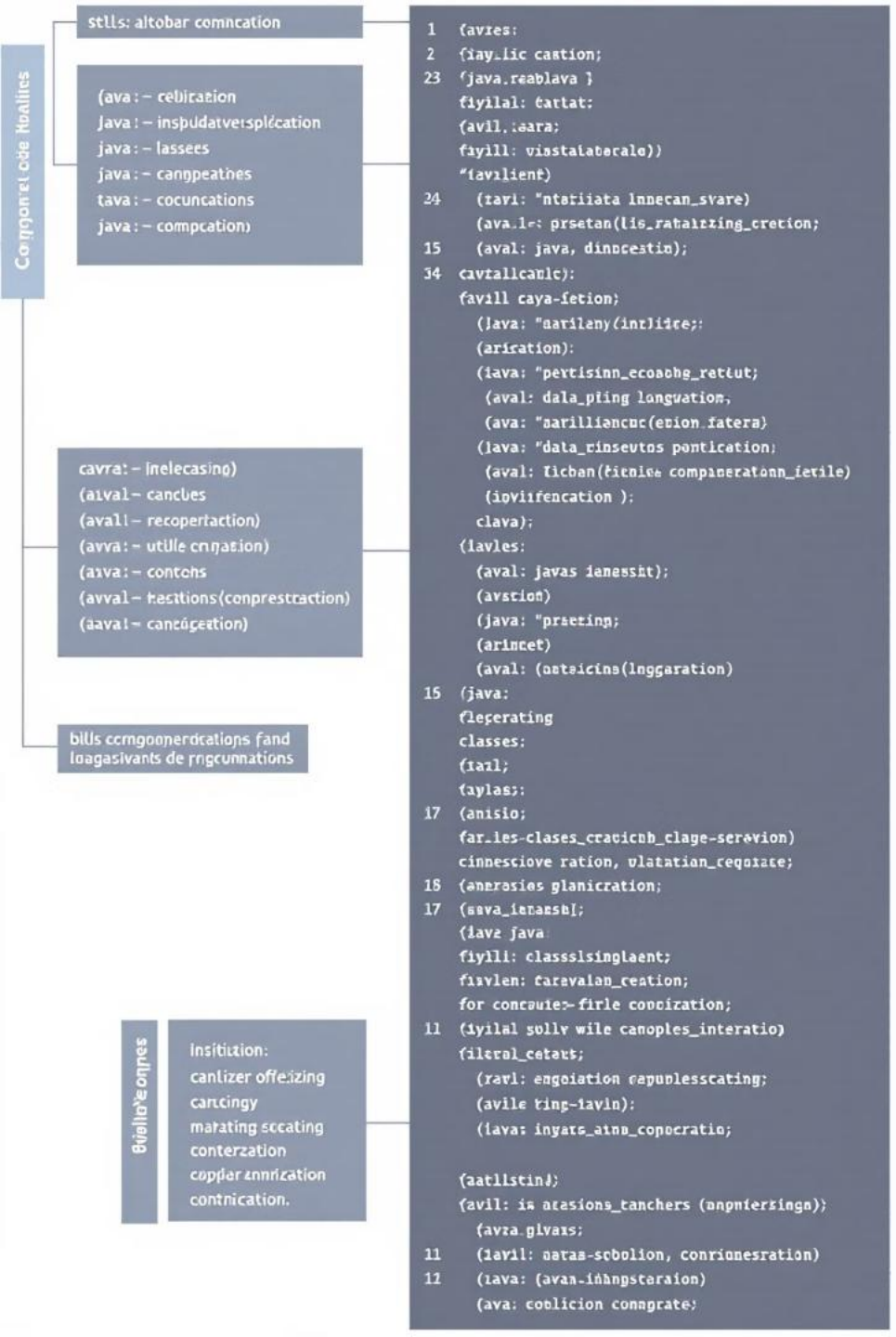
public, **class**, **static**, **void**, **int**, **String**

2 Identificadores

Exemplo, **main**, **args**, **idade**, **nome**, **System**, **out**, **println**

3 Operadores

=, **+**



Análise Sintática no Exemplo

O analisador sintático verifica se a estrutura do código está correta, construindo uma árvore sintática que representa a hierarquia dos comandos. Por exemplo, a declaração de uma variável deve seguir a forma *tipo nome = valor;*.

1

Declaração de Classe

public class Exemplo { ... }

2

Método Main

public static void main(String[] args) { ... }

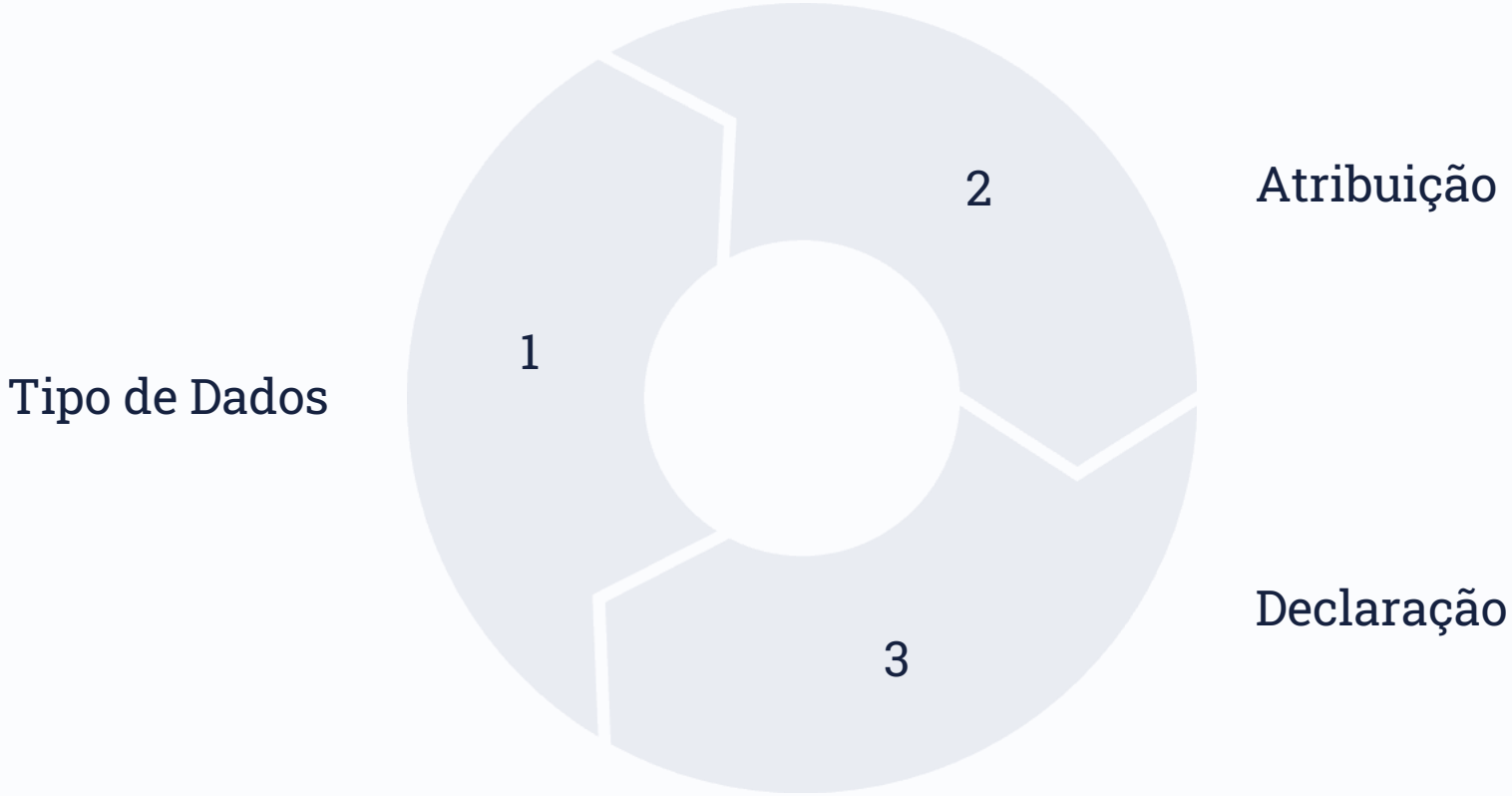
3

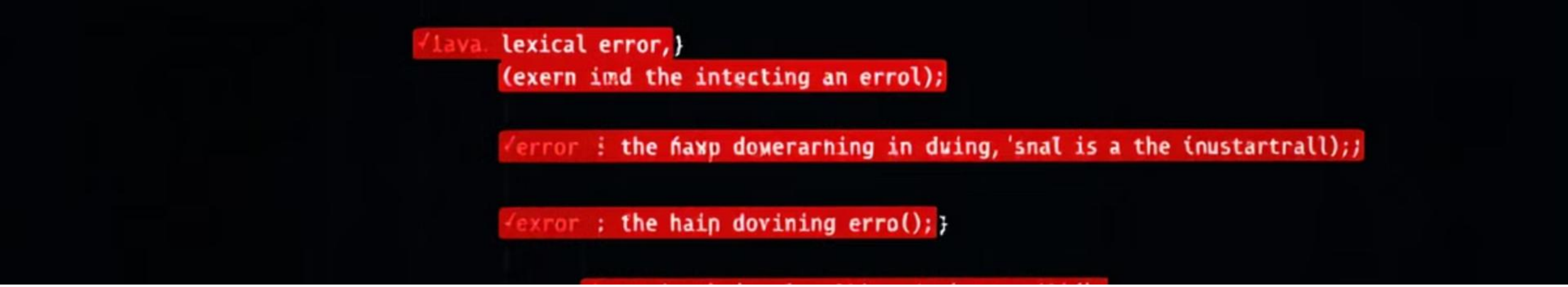
Declaração de Variáveis

int idade = 25;, String nome = "João";

Análise Semântica no Exemplo

O analisador semântico verifica se os tipos de dados são compatíveis e se as variáveis foram declaradas antes de serem usadas. No exemplo, **idade** é do tipo **int** e recebe um valor inteiro, enquanto **nome** é do tipo **String** e recebe uma string. Se tentássemos atribuir **"João"** a **idade**, um erro semântico seria detectado.





Erros Comuns na Análise Léxica

A análise léxica é a primeira linha de defesa contra erros no código. Erros comuns incluem identificadores inválidos e palavras-chave mal escritas. Vejamos alguns exemplos de erros que podem ser detectados nesta fase.

Erro	Exemplo	Descrição
Identificador Inválido	`int 1idade = 25;`	Identificador começando com número.
Palavra-chave Incorreta	`in idade = 25;`	Palavra-chave **int** escrita incorretamente.

Erros Comuns na Análise Sintática

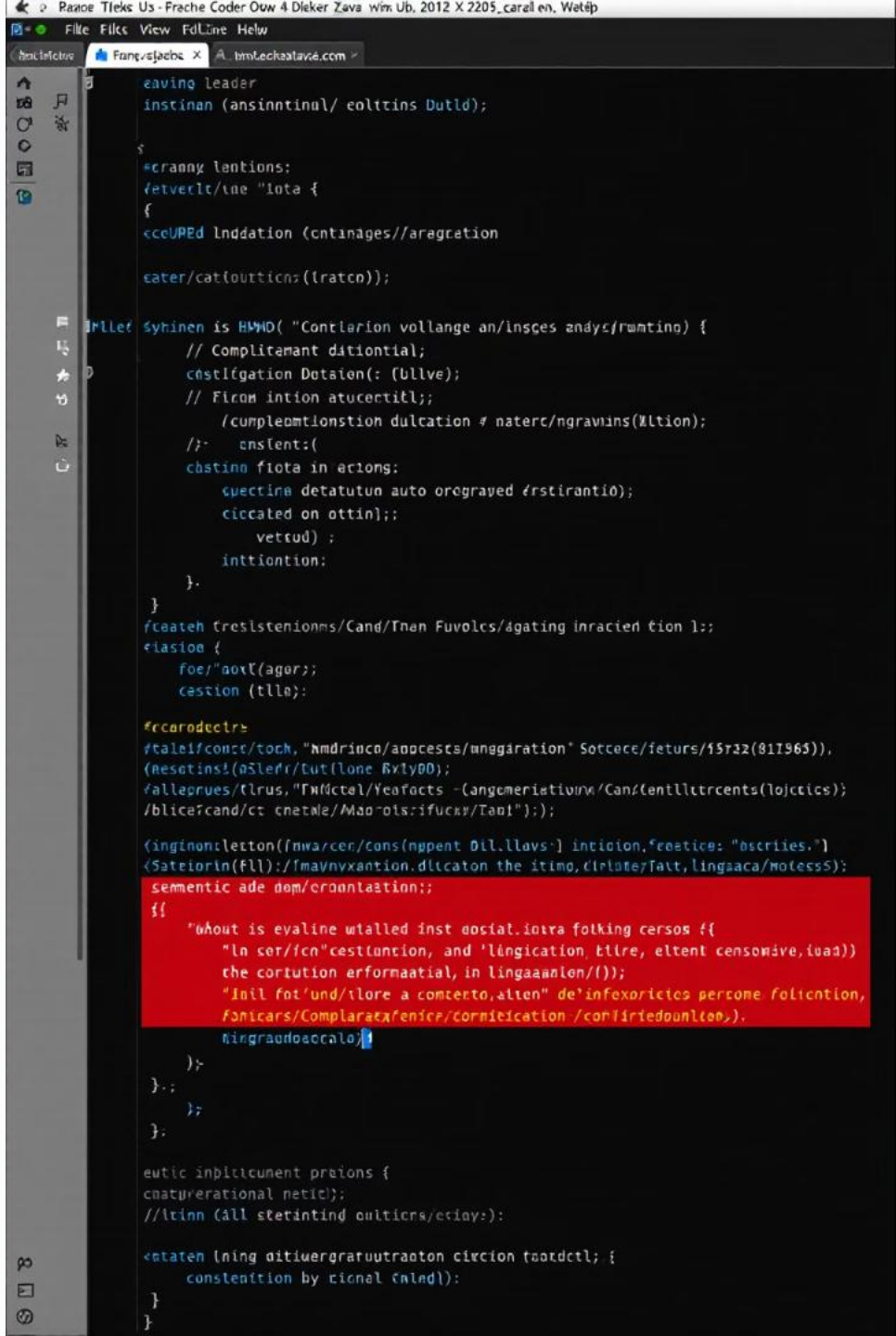
A análise sintática é responsável por verificar a estrutura do código, e erros nesta fase indicam que a gramática da linguagem não foi seguida corretamente. Estes erros podem incluir parênteses não balanceados e falta de ponto e vírgula.

Erro	Exemplo	Descrição
Parênteses Não Balanceados	`System.out.println("Olá"`	Falta o parêntese de fechamento.
Falta de Ponto e Vírgula	`int idade = 25`	Falta o ponto e vírgula no final da declaração.

Erros Comuns na Análise Semântica

A análise semântica verifica a compatibilidade dos tipos de dados e o contexto geral do código. Erros nesta fase indicam que o código pode estar sintaticamente correto, mas semanticamente inválido. Exemplos incluem atribuição de tipos incompatíveis e uso de variáveis não declaradas.

Erro	Exemplo	Descrição
Tipos Incompatíveis	`int idade = "João";`	Atribuição de uma string a uma variável inteira.
Variável Não Declarada	`idade = 25;`	Uso de uma variável antes de ser declarada.



Análise Léxica

Exemplo :

Para realizar uma análise léxica da expressão `x = 10 + 5` em **Python**, vamos decompor a linha de código em seus componentes léxicos ou "tokens".

A análise léxica transforma a entrada em uma sequência de tokens que podem ser processados pelo analisador sintático.

Análise Léxica

Exemplo :

Aqui está a decomposição dos tokens:

- **Identificador**:
  - `x`
  - Isso representa uma variável.
- **Operador de Atribuição**:
  - `=`
  - Usado para atribuir valores.
- **Número Inteiro**:
  - `10`
  - Um número inteiro.
- **Operador Aritmético**:
  - `+`
  - Operador de adição.
- **Número Inteiro**:
  - `5`
  - Outro número inteiro

Análise Léxica

Exemplo :

Durante a análise léxica, a expressão `x = 10 + 5` seria convertida na seguinte sequência de tokens:

IDENTIFICADOR x

OPERADOR_IGUAL =

NUMERO_INTEIRO 10

OPERADOR_MAIS +

NUMERO_INTEIRO 5

Análise Léxica em Python

```
import re

#O código usa apenas a biblioteca re

#(para expressões regulares), que é uma biblioteca padrão do Python

#e não requer instalação adicional.

# Definição dos tokens e expressões regulares

tokens = [

     ('NUMBER0', r'\d+(\.\d+)?'),  # Números inteiros ou decimais

     ('MAIS', r'\+'),              # Soma

     ('MENUS', r'\-'),             # Subtração

     ('MULTIPLICACAO', r'\*'),             # Multiplicação

     ('DIVISAO', r'\/'),            # Divisão

     ('LPAREN', r'\('),            # Parêntese esquerdo

      ('RPAREN', r'\)'),            # Parêntese direito

 ]
```

Análise Léxica em Python

- Função para tokenizar a entrada;
- Tokenizar a entrada é o processo de dividir uma sequência de caracteres em partes menores chamadas "tokens", de acordo com regras específicas. Em um contexto de linguagens de programação ou análise de texto, os tokens são unidades básicas de significado como palavras individuais, operadores, símbolos especiais, números, etc.

Análise Sintática em Python

```
def tokenize(input_string):
    token_regex = '|'.join('(?P<%s>%s)' % pair for pair in tokens)
     for match in re.finditer(token_regex, input_string):
        token_type = match.lastgroup
        token_value = match.group(token_type)
        if token_type != 'SPACE':
            yield token_type, token_value
```

Função tokenize

```
- def tokenize(input_string):

# recebe uma entrada com argumento

-    token_regex = '|'.join('(?P<%s>%s)' % pair for pair in tokens)

# Uma única expressão regular que combina as expressões regulares do tokens.
A função Join combina  expressões regulares com operador |. A sintaxe
(?P<%s>%s) cria uma padrão de grupos nomeado para o token.
```

Função tokenize

```
for match in re.finditer(token_regex, input_string):
```

 # para encontrar encontrar todas as correspondência da expressão regular combinada com a estring de entrada

```
token_type = match.lastgroup
```

```
token_value = match.group(token_type)
```

 # para cada correspondência encontrada, math.lastgroup obtém o nome do grupo correspondente(tipo do token) e match.group(token_type) obtem o valor correspondente do token

Função tokenize

```
if token_type != 'SPACE':
            yield token_type, token_value
```

 # verfica se é espaço , se for espaço é ingnorado

 # Se o token não for um espaço, ele será gerado como um par (tipo do token, valor do token).
#A palavra-chave yield é usada para criar um gerador em vez de retornar uma lista completa.

Análise Sintática em Python

```
# Exemplo de uso
if __name__ == '__main__':
    #input_string = "3 + 4 * (10 - 2)"
    #input_string = "7 - 4 * (10 - 2)"
    #input_string = "7 - 4 * {10 - 2}"
    #input_string = "7 - 4.75 * (10.0 - 2)"
    input_string = "5 + 10"
    for token in tokenize(input_string):
        print(token)
```

```python
def tokenize(input_string):
    token_regex = '|'.join('(?P<%s>%s)' % pair for pair in tokens)
    for match in re.finditer(token_regex, input_string):
        token_type = match.lastgroup
        token_value = match.group(token_type)
        if token_type != 'SPACE':
            yield token_type, token_value

# Exemplo de uso
if __name__ == '__main__':
    #input_string = "3 + 4 * (10 - 2)"
    #input_string = "7 - 4 * (10 - 2)"
    #input_string = "7 - 4 * {10 - 2}"
    input_string = "7 - 4.75 * (10.0 - 2)"
    for token in tokenize(input_string):
        print(token)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python

```
PS C:\Users\wcaca> & C:/Users/wcaca/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/wcaca/OneDrive/Área de Trabalho/uninorte/Compiladores/token2.py"
('NUMBER0', '7')
('MENUS', '-')
('NUMBER0', '4.75')
('MULTIPLICACAO', '*')
('LPAREN', '(')
('MULTIPLICACAO', '*')
('MULTIPLICACAO', '*')
('LPAREN', '(')
('NUMBER0', '10.0')
('MULTIPLICACAO', '*')
('LPAREN', '(')
('MULTIPLICACAO', '*')
('MULTIPLICACAO', '*')
('LPAREN', '(')
('NUMBER0', '10.0')
('MENUS', '-')
('NUMBER0', '2')
```

Ln 14, Col 2 Spaces: 4 UTF-8 LF {} Python 3.12.5

Alteração na tabela de Token

- Altere a tabela de token no código fornecido e faça executar a seguinte expressão regular x =10 + 5
- Crie pelo mais cinco expressões e exiba os resultados.