Projeto de PLP

DataScript

IN1007 - Paradigmas de Linguagens de Programação - 2025.2

Prof. Dr. Augusto Cezar Alves Sampaio acas@cin.ufpe.br

Equipe 5:
Andrezza de Melo Bonfim – amb8@cin.ufpe.br
Athos Pugliese – amps3@cin.ufpe.br
Jordan Kalliure Souza Carvalho - jksc@cin.ufpe.br

Programa de Pós-Graduação Acadêmico em Ciência da Computação

Centro de Informática UFPE

Universidade Federal de Pernambuco

Motivação

Motivação

- Evoluir a linguagem de programação imperativa 2 para uma ferramenta com capacidade de **análise de dados**, introduzindo um conjunto de comandos para manipulação de dados, transformando a linguagem em uma ferramenta prática para análises simples.
- Implementar funcionalidades essenciais como:
  - Carregamento de Dados: load("arquivo.csv")
  - Filtragem: filter(condicao)
  - Cálculos Estatísticos: mean, max, std, variance, etc.

Exemplos

Exemplos

A ideia é permitir que o usuário escreva, por exemplo:

```
// 1. Carregar dados
LOAD "Testes/csvs/funcionarios_completo.csv" AS func;
LOAD "Testes/csvs/vendas.csv" AS vendas;

// 2. Análise estatística com atribuição
MEAN func.salario AS media_salarial;
MEDIAN func.salario AS mediana_salarial;
MODE func.anos_experiencia AS experiencia_mais_comum;
STD func.idade AS desvio_idade;
VARIANCE func.salario AS variancia_salario;
MIN func.salario AS menor_salario;
MAX func.salario AS maior_salario;
RANGE func.idade AS amplitude_idades;
QUARTILES func.salario AS quartis_salario;

// 3. Contagem de registros (Sintaxe: COUNT Id AS Id)
COUNT func AS total_funcionarios;
COUNT vendas AS total_vendas;
```

```
// 4. Filtragem de dados para criar novos subconjuntos
FILTER func INTO funcionarios_seniores WHERE idade > 30;
FILTER func INTO func_ti WHERE departamento == "TI";
FILTER vendas INTO vendas_grandes WHERE valor > 1000.5;

// 5. Análise nos dados filtrados
MEAN funcionarios_seniores.salario AS media_seniores;
COUNT funcionarios_seniores AS total_seniores;

// 6. Visualização e salvamento
SHOW func;

// SHOW estatísticas diretas
SHOW MEAN func.salario;
SHOW MIN func.idade;

// SAVE Expressao AS Expressao
SAVE funcionarios_seniores AS
"Testes/csvs/resultados/seniores.csv"
```

Exemplos

```
//“{“ “proc” Id “(“ [ ListaDeclaracaoParametro ] “)” “{“ Comando “}” “;” “call” Id “(“ [ ListaExpressao ] “)” “}”

{
   // DECLARANDO O PROCEDIMENTO
   PROC analisarFuncionarios (STRING arquivo_csv, STRING nome_dataframe) {
      LOAD arquivo_csv AS temp_df;
      MEAN temp_df.salario AS media_salarial;
      MEDIAN temp_df.salario AS mediana_salarial;
      STD temp_df.idade AS desvio_idade;
      FILTER temp_df INTO seniores WHERE idade > 30;
      MEAN seniores.salario AS media_seniores;
      SHOW media_salarial;
      SHOW media_seniores
   };

   // CHAMANDO NO MESMO BLOCO (2 vezes)
   CALL analisarFuncionarios("Testes/csvs/funcionarios_cin.csv", "func");
   CALL analisarFuncionarios("Testes/csvs/funcionarios_ctg.csv", "func2")
}
```

BNF

Mudanças na BNF da LI2

```
Comando ::= Atribuicao
        | ComandoDeclaracao
        | While
        | IfThenElse
        | IO
        | Comando ";" Comando
        | Skip
        | ChamadaProcedimento
        | ComandoEstatistico

ValorConcreto ::= ValorInteiro
        | ValorBooleano
        | ValorString
        | ValorDouble
        | ValorDataFrame

Tipo ::= "string" | "int" |
"boolean" | "double"
| TipoDataFrame

StringLiteral ::= """ [^\"]* """ |
"'" [^']* "'"

Id ::= [a-zA-Z_][a-zA-Z0-9_]*
```

```
ComandoEstatistico ::= ComandoLoad
| ComandoFiltro
| ComandoCalculo
| ComandoShow
| ComandoSave

ComandoLoad ::= "LOAD" StringLiteral "AS" Id

ComandoFiltro ::= "FILTER" Id INTO Id "WHERE" Expressao

ComandoCalculo ::= AnaliseColuna | ContagemTabela

AnaliseColuna ::= OpEstatistica ReferenciaColuna "AS" Id

ContagemTabela ::= "COUNT" Id "AS" Id | "COUNT" Id

Comando Show ::= "SHOW" Id | "SHOW" OpEstatistica
ReferenciaColuna

OpEstatistica ::= "MEAN" | "MEDIAN" | "MODE" | "STD" |
"VARIANCE" | "MIN" | "MAX" | "RANGE" | "QUARTILES"

ReferenciaColuna ::= Expressao "." Id

ComandoSave ::= "SAVE" Id "AS" StringLiteral
```

BNF da LI2 extendida

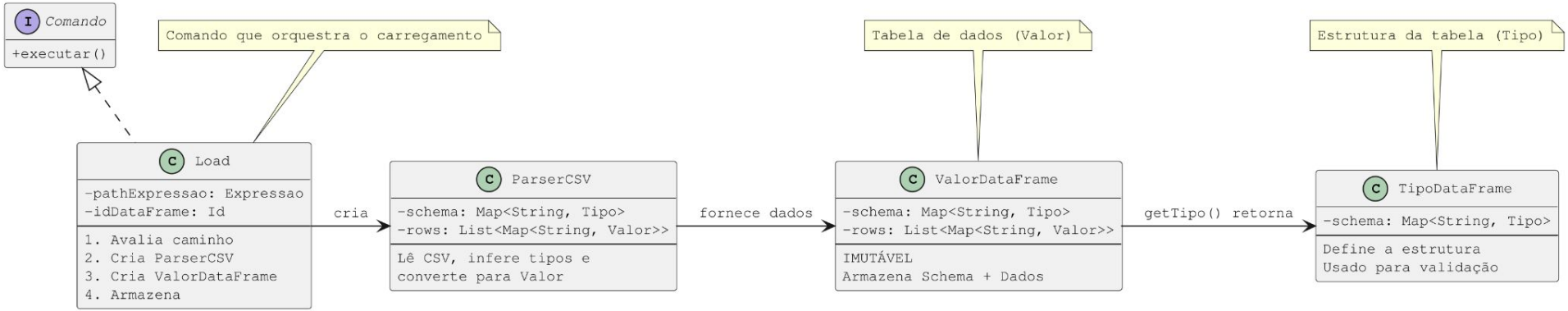
```
Programa ::= Comando

Comando ::= Atribuicao
          | ComandoDeclaracao
          | While
          | IfThenElse
          | IO
          | Comando ";" Comando
          | Skip
          | ChamadaProcedimento
          | ComandoEstatistico    // ---> ADICIONADO

Skip ::=

Atribuicao ::= Id ":=" Expressao

Expressao ::= Valor
            | ExpUnaria
            | ExpBinaria
            | Id

Valor ::= ValorConcreto

ValorConcreto ::= ValorInteiro
                | ValorBooleano
                | ValorString
                | ValorDouble          // ---> ADICIONADO
                | ValorDataFrame       // ---> ADICIONADO

ExpUnaria ::= "-" Expressao
            | "not" Expressao
            | "length" Expressao

ExpBinaria ::= Expressao "+" Expressao
             | Expressao "-" Expressao
             | Expressao "and" Expressao
             | Expressao "or" Expressao
             | Expressao "==" Expressao
             | Expressao "++" Expressao
```

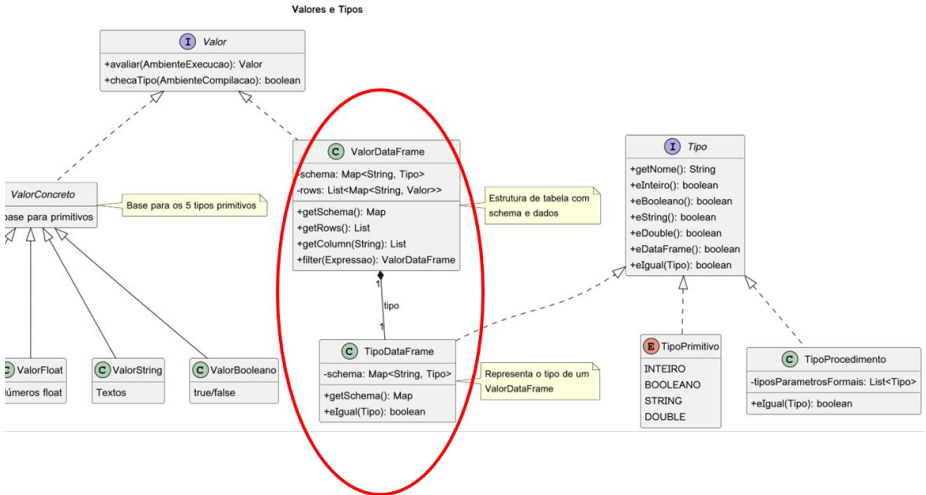
```
ComandoDeclaracao ::= "{" Declaracao ";" Comando "}"

Declaracao ::= DeclaracaoVariavel
             | DeclaracaoProcedimento
             | DeclaracaoComposta

DeclaracaoVariavel ::= "var" Id "=" Expressao

DeclaracaoComposta ::= Declaracao "," Declaracao

DeclaracaoProcedimento ::= "proc" Id "(" [ ListaDeclaracaoParametro ] ")" "{" Comando "}"

ListaDeclaracaoParametro ::= Tipo Id
                           | Tipo Id "," ListaDeclaracaoParametro

Tipo ::= "string" | "int" | "boolean" | "double" | TipoDataFrame

While ::= "while" Expressao "do" Comando

IfThenElse ::= "if" Expressao "then" Comando "else" Comando

IO ::= "write" "(" Expressao ")"
     | "read" "(" Id ")"

ChamadaProcedimento ::= "call" Id "(" [ ListaExpressao ] ")"

ListaExpressao ::= Expressao | Expressao, ListaExpressao
```

```
// --- SEÇÃO DA DSL DE DADOS ---

ComandoEstatistico ::= ComandoLoad
| ComandoFiltro
| ComandoCalculo
| ComandoShow
| ComandoSave

ComandoLoad ::= "LOAD" StringLiteral "AS" Id

ComandoFiltro ::= "FILTER" Id INTO Id "WHERE" Expressao

ComandoCalculo ::= AnaliseColuna | ContagemTabela

Comando Show ::= "SHOW" Expressao    | "SHOW" OpEstatistica ReferenciaColuna

ComandoSave ::= "SAVE" Expressao "AS" StringLiteral

AnaliseColuna ::= OpEstatistica ReferenciaColuna "AS" Id

ContagemTabela ::= "COUNT" Expressao "AS" Id

ReferenciaColuna ::= Expressao "." Id

OpEstatistica ::= "MEAN" | "MEDIAN" | "MODE" | "STD" | "VARIANCE"
| "MIN" | "MAX" | "RANGE" | "QUARTILES"

// --- Definições Auxiliares ---

StringLiteral ::= "\"" [^\"]* "\""
                | "'" [^']* "'"

Id ::= [a-zA-Z_][a-zA-Z0-9_]*
```

Implementação

Implementação - Classe LOAD





Implementação - Classe LOAD

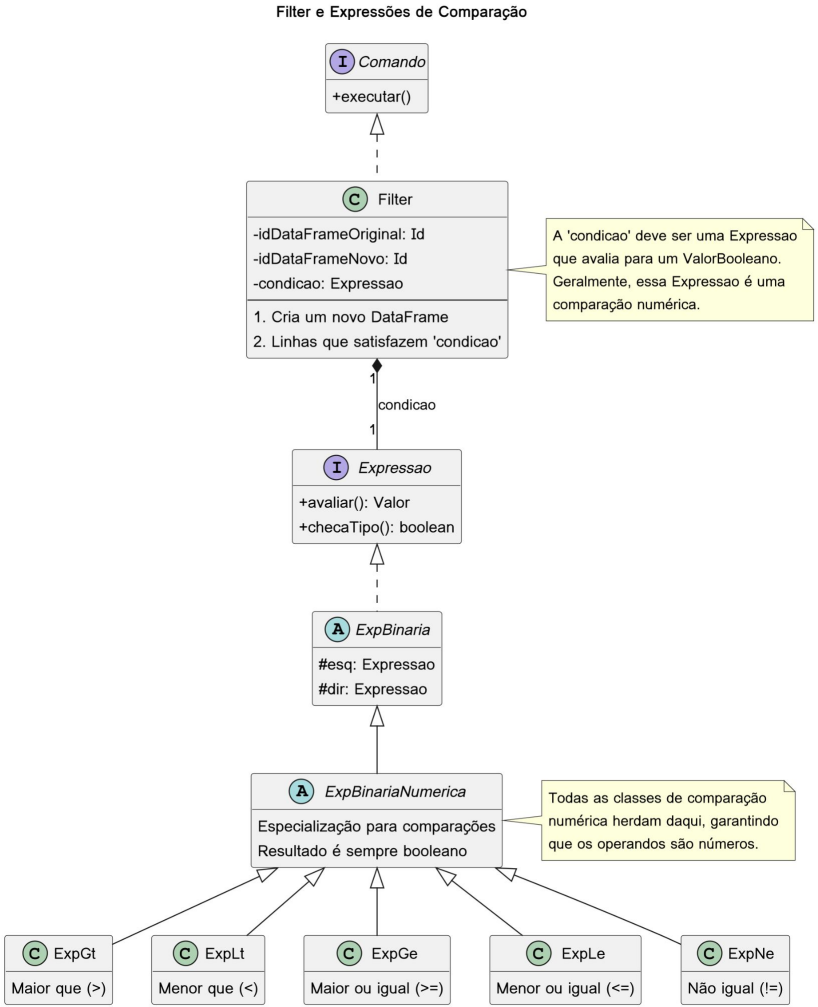
```
public boolean checaTipo(AmbienteCompilacaoImperativa amb) {
        // Verifica se a expressão do caminho é válida
        if (!pathExpressao.checaTipo(amb)) return false;
        if (!pathExpressao.getTipo(amb).eString()) return false;

        // Se tiver ID para salvar
        if (idDataFrame != null) {
            Map<String, Tipo> schema = null;

            // Caso 1: String literal -> Conseguimos inferir Schema
            if (pathExpressao instanceof ValorString) {
                String path = ((ValorString) pathExpressao).valor();
                try {
                    ParserCSV parser = new ParserCSV(path);
                    schema = parser.getSchema(); // Descobrimos as colunas
                } catch (Exception e) {
                    System.out.println("Aviso de Compilação: " +
e.toString());
                    // Mesmo com erro de leitura, vamos tentar registrar
para não travar o parser
                }
            }
```

```
// Caso 2: Caminho Dinâmico (Variável) -> Schema desconhecido (null)
            else {
                // schema continua null
            }
            // REGISTRA NO AMBIENTE (Com Schema ou Null)
            // Isso impede o erro "VariavelNaoDeclarada" nos comandos
seguintes
            try {
                // Cria o "Crachá" do DataFrame
                TipoDataFrame tipoDf = new TipoDataFrame(schema);
                // Avisa ao compilador: "Existe uma variável 'func' que é
um DataFrame"
                amb.map(idDataFrame, tipoDf);
            } catch (Exception e) {
                System.out.println("Erro ao registrar variável: " +
e.toString());
                return false;
            }
        }
        return true;
    }
```

Implementação - Classe LOAD

```
public AmbienteExecucaoImperativa executar(AmbienteExecucaoImperativa amb)
{
        try {
            // 1. Descobre o nome do arquivo e o interpretador busca o
valor dela na memória (ex: "dados.csv").
            Valor val = this.pathExpressao.avaliar(amb);
            if (!(val instanceof ValorString)) throw new
RuntimeException("Caminho inválido");
            String path = ((ValorString) val).valor();//Garante que o
caminho resultante é um Texto.

            // valida o formato do arquivo
            if (!path.toLowerCase().endsWith(".csv")) throw new
RuntimeException("Arquivo deve ser .csv");
            File arquivo = new File(path);
            if (!arquivo.exists()) throw new RuntimeException("Arquivo não
encontrado: " + path);
```

```
//cria o parsercsv que abre o arquivo, lê o cabeçalho, lê todas as linhas,
infere se é número ou texto e guarda tudo na memória dela.
            ParserCSV parser = new ParserCSV(path);

            //Cria o ValorDataFrame (A Planilha na Memória). Pegamos o
resultado do parser (Schema + Linhas) e colocamos dentro do objeto
ValorDataFrame
            ValorDataFrame dataFrame = new
ValorDataFrame(parser.getSchema(), parser.getRows());

            // Salva na Variável
            if (this.idDataFrame != null) {
                // é onde a mágica acontece. A variável func (o
idDataFrame) passa a apontar para essa planilha carregada
                amb.map(this.idDataFrame, dataFrame);
                System.out.println(">> Dataset carregado em '" +
this.idDataFrame.getIdName() + "' (" + parser.getRows().size() + "
linhas)");
            }

        } catch (RuntimeException re) { throw re;
        } catch (Exception e) { throw new RuntimeException("Erro ao
carregar CSV: " + e.toString()); }
        return amb;
    }
```



Implementação - Classe Filter

Classe Filter

- Dados tornam-se variáveis no amb

```
public AmbienteExecucaoImperativa executar(AmbienteExecucaoImperativa amb) {
       // ... (Implementação do executar permanece igual) ...
       Valor val = amb.get(idDataFrameOriginal);
       if (!(val instanceof ValorDataFrame)) throw new RuntimeException("Não é
DataFrame.");
       ValorDataFrame dfOriginal = (ValorDataFrame) val;

       List<Map<String, Valor>> linhasFiltradas = new ArrayList<>();
       for (Map<String, Valor> linha : dfOriginal.getRows()) {
           amb.incrementa();
           for (Map.Entry<String, Valor> entry : linha.entrySet()) {
               amb.map(new Id(entry.getKey()), entry.getValue());
           }
           try {
               Valor resultado = condicao.avaliar(amb);
               if (!(resultado instanceof ValorBooleano)) throw new
RuntimeException("Filtro deve ser booleano");
               if (((ValorBooleano) resultado).valor())
linhasFiltradas.add(linha);
           } finally {
               amb.restaura();
           }
       }
       ValorDataFrame dfNovo = new ValorDataFrame(dfOriginal.getSchema(),
linhasFiltradas);
       amb.map(idDataFrameNovo, dfNovo);
       return amb;
   }
```

- Verificação do dataframe e dados no amb

```
public boolean checaTipo(AmbienteCompilacaoImperativa amb) {
       Tipo tipoOrigem;
       try {
           tipoOrigem = amb.get(idDataFrameOriginal);
       } catch (Exception e) { return false; }

       if (tipoOrigem == null || !tipoOrigem.eDataFrame()) return false;

       TipoDataFrame tdf = (TipoDataFrame) tipoOrigem;
       amb.incrementa();
           if (tdf.getSchema() != null) {
           for (Map.Entry<String, Tipo> entry : tdf.getSchema().entrySet()) {
               amb.map(new Id(entry.getKey()), entry.getValue());
           }
           if (!condicao.checaTipo(amb)) {
               amb.restaura();
               return false;
           }
           if (!condicao.getTipo(amb).eBooleano()) {
               amb.restaura();
               return false;
           }
       } else {
       }
       amb.restaura();
```

Classe ExpBinariaNumerica

```
  protected boolean
checaTipoElementoTerminal(AmbienteCompilacao amb)
           throws VariavelNaoDeclaradaException {

       Tipo tipoEsq = getEsq().getTipo(amb);
       Tipo tipoDir = getDir().getTipo(amb);

       boolean esqNumerico =
tipoEsq.eIgual(TipoPrimitivo.INTEIRO) ||
tipoEsq.eIgual(TipoPrimitivo.DOUBLE);
       boolean dirNumerico =
tipoDir.eIgual(TipoPrimitivo.INTEIRO) ||
tipoDir.eIgual(TipoPrimitivo.DOUBLE);

       return esqNumerico && dirNumerico;
   }
```

```
protected double getValorNumerico(Valor v) {
       if (v instanceof ValorInteiro) {
           return (double) ((ValorInteiro)
v).valor();
       } else if (v instanceof ValorDouble) {
           return ((ValorDouble) v).valor();
       }
       // Isso não deve acontecer se checaTipo()
foi chamado
       throw new RuntimeException("Erro de tipo:
esperado valor numérico.");
   }
```

Subclasses adicionadas de Filter

ExpGt

```
   public Valor avaliar(AmbienteExecucao amb) throws
VariavelNaoDeclaradaException {
       Valor valEsq = esq.avaliar(amb);
       Valor valDir = dir.avaliar(amb);

       double numEsq = getValorNumerico(valEsq);
       double numDir = getValorNumerico(valDir);

       return new ValorBooleano(numEsq > numDir);
```

ExpNe

```
   public Valor avaliar(AmbienteExecucao amb) throws
VariavelNaoDeclaradaException {
       Valor valEsq = esq.avaliar(amb);
       Valor valDir = dir.avaliar(amb);

       double numEsq = getValorNumerico(valEsq);
       double numDir = getValorNumerico(valDir);

       // Comparação direta de != para floats pode ser
imprecisa
       // mas para este projeto, deve ser suficiente.
       return new ValorBooleano(numEsq != numDir);
   }
```

ExpGe

```
 public Valor avaliar(AmbienteExecucao amb) throws
VariavelNaoDeclaradaException {
       Valor valEsq = esq.avaliar(amb);
       Valor valDir = dir.avaliar(amb);

       double numEsq = getValorNumerico(valEsq);
       double numDir = getValorNumerico(valDir);

       return new ValorBooleano(numEsq >= numDir);
   }
```

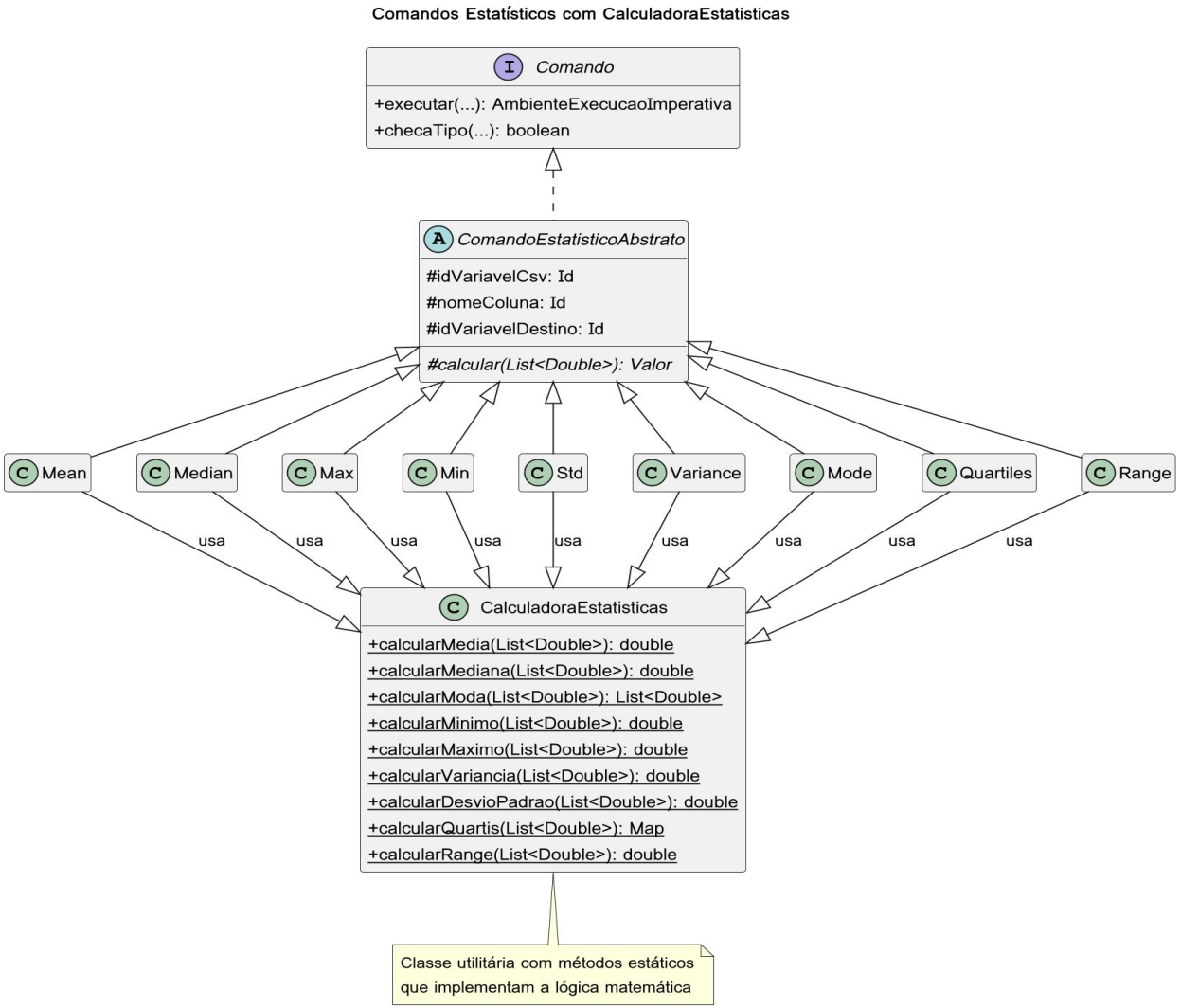
ExpLe

```
   public Valor avaliar(AmbienteExecucao amb) throws
VariavelNaoDeclaradaException {
       Valor valEsq = esq.avaliar(amb);
       Valor valDir = dir.avaliar(amb);

       double numEsq = getValorNumerico(valEsq);
       double numDir = getValorNumerico(valDir);

       return new ValorBooleano(numEsq <= numDir);
```

ExpLt

```
   public Valor avaliar(AmbienteExecucao amb) throws
VariavelNaoDeclaradaException {
       Valor valEsq = esq.avaliar(amb);
       Valor valDir = dir.avaliar(amb);

       double numEsq = getValorNumerico(valEsq);
       double numDir = getValorNumerico(valDir);

       return new ValorBooleano(numEsq < numDir);
   }
```

Implementação - Operações Estatísticas



Comandos Estatísticos Abstratos

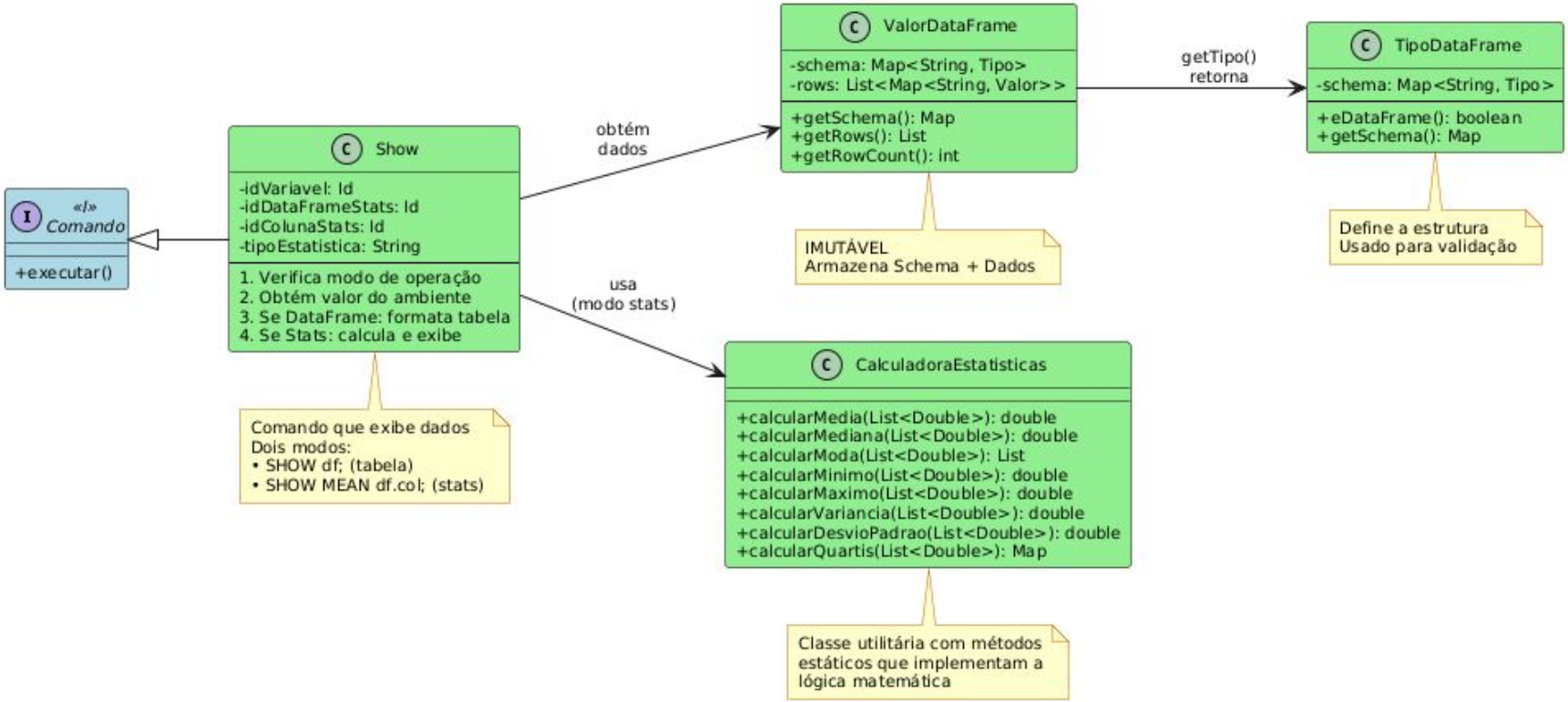
```
public abstract class ComandoEstatisticoAbstrato implements
Comando {
    protected Id idVariavelCsv;
    protected Id nomeColuna;
    protected Id idVariavelDestino;

    protected abstract Valor calcular(List<Double> numeros);}
```

```
public AmbienteExecucaoImperativa
executar(AmbienteExecucaoImperativa amb) {
    ValorDataFrame df = (ValorDataFrame) amb.get(idVariavelCsv);
    List<Double> numeros = extrairNumeros(df, nomeColuna);
    Valor resultado = calcular(numeros);  // Polimorfismo
    amb.map(idVariavelDestino, resultado);
    return amb;
}
```

```
public class Mean extends ComandoEstatisticoAbstrato {
    @Override
    protected Valor calcular(List<Double> numeros) {
        return new
ValorDouble(CalculadoraEstatisticas.calcularMedia(numeros));
    }
}
```

```
public static double calcularMedia(List<Double> numeros) {
    double soma = 0.0;
    for (double num : numeros) soma += num;
    return soma / numeros.size();
}
```

Implementação - Classe Show



Implementação - Classe Show

```
// 1. Recupera o valor do ambiente
Valor valor = amb.get(idVariavel);

// 2. Se for DataFrame, exibe tabela formatada
if (valor instanceof ValorDataFrame) {
    ValorDataFrame df = (ValorDataFrame) valor;
    List<String> colunas = new
ArrayList<>(df.getSchema().keySet());

    // Cabeçalho
    System.out.println(String.join("\t|\t", colunas));

    // Linhas
    for (Map<String, Valor> linha : df.getRows()) {
        List<String> celulas = colunas.stream()
            .map(col -> linha.get(col).toString())
            .collect(Collectors.toList());
        System.out.println(String.join("\t|\t", celulas));
    }
}
```

```
// 1. Recupera DataFrame e extrai números da coluna
ValorDataFrame df = (ValorDataFrame) amb.get(idDataFrameStats);
List<Double> numeros = new ArrayList<>();
for (Map<String, Valor> linha : df.getRows()) {
    Valor v = linha.get(colName);
    if (v instanceof ValorInteiro)
        numeros.add((double)((ValorInteiro)v).valor());
    else if (v instanceof ValorDouble)
        numeros.add(((ValorDouble)v).valor());
}

// 2. Calcula estatística usando CalculadoraEstatisticas
switch (tipoEstatistica) {
    case "MEAN": output =
CalculadoraEstatisticas.calcularMedia(numeros); break;
    case "MEDIAN": output =
CalculadoraEstatisticas.calcularMediana(numeros); break;
    case "STD": output =
CalculadoraEstatisticas.calcularDesvioPadrao(numeros); break;
    // ... demais operações
}
System.out.println(">> " + tipoEstatistica + ": " + output);
```

Implementação - Classe Save

```
// 1. Avalia o caminho
       Valor valPath = pathExpressao.avaliar(amb);
       if (!(valPath instanceof ValorString)) {
            throw new RuntimeException("Erro: O caminho
para salvar o arquivo deve ser uma String.");
       }
       String path = ((ValorString) valPath).valor();

       // 2. Recupera o DataFrame
       Valor valDf = amb.get(idDataFrame);
       if (!(valDf instanceof ValorDataFrame)) {
           throw new RuntimeException("Erro: Variável
'" + idDataFrame.getIdName() + "' não é um
DataFrame.");
       }
       ValorDataFrame df = (ValorDataFrame) valDf;
```

```
      // 3. Escreve no disco
       try (BufferedWriter writer = new BufferedWriter(new
FileWriter(path))) {
           // Cabeçalho
           String[] colunas = df.getSchema().keySet().toArray(new
String[0]);
           writer.write(String.join(",", colunas));
           writer.newLine();

           // Linhas
           for (Map<String, Valor> linha : df.getRows()) {
               String[] celulas = new String[colunas.length];
               for (int i = 0; i < colunas.length; i++) {
                   Valor v = linha.get(colunas[i]);
                   // Se for string, pega o valor cru. Se for outro,
usa toString.
                   celulas[i] = (v instanceof ValorString) ?
((ValorString)v).valor() : v.toString();
               }
               writer.write(String.join(",", celulas));
               writer.newLine();
           }
       } catch (IOException e) {
           throw new RuntimeException("Erro ao salvar arquivo '" + path
+ "': " + e.getMessage());
```

Implementação - Classe Count

```
public boolean checaTipo(AmbienteCompilacaoImperativa amb) throws
RuntimeException {
        // 1. Verifica se a variável alvo existe e é um DataFrame
        Tipo tipoDf = amb.get(idDataFrame);

        if (tipoDf == null || !tipoDf.eDataFrame()) {
            // Se o tipo for null, a variável não foi declarada.
            // Se eDataFrame() for false, é um Inteiro/String/Etc, não dá
pra contar linhas.
            return false;
        }

        // 2. Se houver variável de destino (AS ...), registra ela como
INTEIRO
        if (idVariavelDestino != null) {
            // O resultado de um COUNT é sempre um número inteiro
            amb.map(idVariavelDestino, TipoPrimitivo.INTEIRO);
        }

        return true;
    }

public AmbienteExecucaoImperativa executar(AmbienteExecucaoImperativa amb)
throws RuntimeException {
        // 1. Pega o DataFrame original do amb
        Valor val = amb.get(idDataFrame);
        ValorDataFrame df;
        try {
            df = (ValorDataFrame) val;
        } catch (ClassCastException e) {
            throw new RuntimeException("Erro de Execução: '" +
idDataFrame.getIdName() + "' não é um DataFrame válido.");
        }
        // 2. Obtém a contagem
        int contagem = df.getRows().size();
        ValorInteiro resultado = new ValorInteiro(contagem);

        // 3. Salva o resultado ou imprime
        if (idVariavelDestino != null) {
            amb.map(idVariavelDestino, resultado);
            System.out.println(">> Contagem de '" +
idDataFrame.getIdName() + "': " + contagem + " (salvo em '" +
idVariavelDestino.getIdName() + "')");
        } else {
             System.out.println(">> Contagem de '" +
idDataFrame.getIdName() + "': " + contagem);
        }

        return amb;
    }
```

Testes

```
// COMANDO CORRETO
LOAD "Testes/csvs/funcionarios_completo.csv" AS df;
MEAN df.salario AS media;
SHOW media;
FILTER df INTO ricos WHERE salario > 5000.00;
SHOW ricos;
COUNT ricos;
FILTER df INTO pessoal_ti WHERE departamento == "TI";
SHOW pessoal_ti;
COUNT pessoal_ti

// COMANDO ERRADO
LOAD 12345 AS df_erro
LOAD "12345" AS df_erro

// COMANDO CORRETO NA TABELA COM DADO ERRADO
LOAD "Testes/csvs/funcionarios_completo_valorerrado.csv" AS df;
MEAN df.salario AS media;
SHOW media

// COMANDO CORRETO MAS TABELA COM DADO ERRADO
LOAD "Testes/csvs/funcionarios_completo_vazio.csv" AS df;
```

Obrigado!

Programa de Pós-Graduação Acadêmico
em Ciência da Computação

Centro de Informática
UFPE

Universidade
Federal
de Pernambuco