Nome do Arquivo

workflow_controller.py

Documentação

```
#1[
#1 TITULO: WORKFLOW CONTROLLER
#1 AUTOR: EDUARDO RIBEIRO SILVA DE OLIVEIRA
#1 DATA: 25/08/2024
#1 VERSAO: 1
#1 FINALIDADE: FAZ AS CHAMADAS DE CLASSES DO GERADOR
#1 ENTRADAS: CAMINHO DO ARQUIVO E NOME DO NOVO ARQUIVO
#1 SAIDAS: ARQUIVO .PY COM CODIGO GERADO PELA LEITURA DE TDS
#1 ROTINAS CHAMADAS: READ_CODE, PROCESS_DECISION_TABLES, GENERATE_CODE,
INSERT_CODE
#1]

from src.code_reader.code_reader import CodeReader
from src.decision_table.decision_table import DecisionTable
from src.code_generator.code_generator import CodeGenerator
from src.code_inserter.code_inserter import CodeInserter

class WorkflowController:

    #1[
    #1 ROTINA: __INIT__
    #1 FINALIDADE: DEFINE ATRIBUTOS
    #1 ENTRADAS: CAMINHO DO ARQUIVO E NOME DO NOVO ARQUIVO
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: N/A
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: __init__
    def __init__(self, file_path: str, file_name: str):
    #2 DEFINE CAMINHO DO ARQUIVO A SER LIDO
        self.file_path = file_path
    #2 DEFINE NOME DO NOVO ARQUIVO A SER GERADO
        self.file_name = file_name
    #2 INICIALIZA O LEITOR DE CODIGO
        self.code_reader = CodeReader(self.file_path)
    #2 INICIALIZA LISTA DE TABELAS DE DECISAO ENCONTRADAS
        self.decision_tables_found = []
    #2 INICIALIZA A LISTA DE TABELAS DE DECISAO PROCESSADAS
        self.processed_tables = []
    #2 INICIALIZA O GERADOR DE CODIGO
        self.code_generator = CodeGenerator()
    #2 INICIALIZA A LISTA DE CODIGOS GERADOS
        self.generated_code = []
    #2 INICIALIZA O INSERIDOR DE CODIGO
        self.code_inserter = CodeInserter(self.file_path, self.file_name)
    #2]

    #1[
    #1 ROTINA: READ_CODE
    #1 FINALIDADE: LE O CODIGO E EXTRAI TABELAS DE DECISAO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: CODEREADER
    #1 CHAMADO POR: EXECUTE
    #1 CHAMA: CODEREADER.GET_EXTRACTED_DECISION_TABLES,
CODEREADER.GET_EXTRACTED_DECISION_TABLES_POSITIONS
    #1]
    #2[
    #2 PSEUDOCODIGO DE: read_code
    def read_code(self):
    #2 EXTRAI AS TABELAS DE DECISAO
        self.decision_tables_found =
self.code_reader.get_extracted_decision_tables()
    #2 EXTRAI POSICOES DAS TABELAS DE DECISAO
        self.position_of_decision_tables_found =
self.code_reader.get_extracted_decision_tables_positions()
    #2]

    #1[
    #1 ROTINA: PROCESS_DECISION_TABLES
    #1 FINALIDADE: PROCESSA AS TABELAS DE DECISAO ENCONTRADAS
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: DECISIONTABLE
    #1 CHAMADO POR: EXECUTE
    #1 CHAMA: DECISIONTABLE.__INIT__
    #1]
```

```
    #2[
    #2 PSEUDOCODIGO DE: process_decision_tables
    def process_decision_tables(self):
    #2 INICIALIZA A LISTA DE TABELAS PROCESSADAS
        self.processed_tables = []
    #2 PERCORRE O CONJUNTO DE TABELAS DE DECISAO
        for index, table in enumerate(self.decision_tables_found):
    #2 PROCESSA CADA TABELA, ARMAZENANDO A TABELA PROCESSADA E A POSICAO
ORIGINAL
            self.processed_tables.append(DecisionTable(table,
self.position_of_decision_tables_found[index]))
    #2]

    #1[
    #1 ROTINA: GENERATE_CODE
    #1 FINALIDADE: GERA CODIGO A PARTIR DAS TABELAS DE DECISAO
PROCESSADAS
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: CODEGENERATOR
    #1 CHAMADO POR: EXECUTE
    #1 CHAMA: CODEGENERATOR.GENERATE_CODE
    #1]
    #2[
    #2 PSEUDOCODIGO DE: generate_code
    def generate_code(self):
    #2 PERCORRE O CONJUNTO DE TABELAS DE DECISAO PROCESSADAS
        for table in self.processed_tables:
    #2 GERA CODIGO PARA CADA TABELA PROCESSADA E O ARMAZENA

self.generated_code.append(self.code_generator.generate_code(table))
    #2]

    #1[
    #1 ROTINA: INSERT_CODE
    #1 FINALIDADE: INSERE O CODIGO GERADO NO ARQUIVO ORIGINAL
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: CODEINSERTER
    #1 CHAMADO POR: EXECUTE
    #1 CHAMA: CODEINSERTER.INSERT
    #1]
    #2[
    #2 PSEUDOCODIGO DE: insert_code
    def insert_code(self):
    #2 CRIA UM DICIONARIO QUE MAPEIA AS POSICOES DAS TABELAS DE DECISAO E
CODIGO GERADO
        decision_table_to_code_map = {k: v for k, v in
zip([table.position_of_decision_table for table in
self.processed_tables], self.generated_code)}
    #2 INSERE O CODIGO GERADO NO ARQUIVO ORIGINAL USANDO O MAPA CRIADO
        self.code_inserter.insert(decision_table_to_code_map)
    #2]

    #1[
    #1 ROTINA: EXECUTE
    #1 FINALIDADE: EXECUTA O FLUXO COMPLETO DE LEITURA, PROCESSAMENTO,
GERACAO E INSERCAO DE CODIGO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: READ_CODE, PROCESS_DECISION_TABLES, GENERATE_CODE,
INSERT_CODE
    #1 CHAMADO POR: N/A
    #1 CHAMA: READ_CODE, PROCESS_DECISION_TABLES, GENERATE_CODE,
INSERT_CODE
    #1]
    #2[
    #2 PSEUDOCODIGO DE: execute
    def execute(self):
    #2 LE O CODIGO E EXTRAI AS TABELAS DE DECISAO
        self.read_code()
    #2 PROCESSA AS TABELAS DE DECISAO ENCONTRADAS
        self.process_decision_tables()
    #2 GERA CODIGO A PARTIR DAS TABELAS PROCESSADAS
        self.generate_code()
    #2 INSERE O CODIGO NO ARQUIVO INDICADO
        self.insert_code()
    #2]
```

Nome do Arquivo
condition.py

Documentação

```
#1[
#1 TITULO: CONDITION.PY
#1 AUTOR: EDUARDO RIBEIRO SILVA DE OLIVEIRA
#1 DATA: 25/08/2024
#1 VERSAO: 1
#1 FINALIDADE: PROCESSAR E EXTRAIR CONDICOES DE UMA TABELA DE DECISAO
FORMATADA COMO STRING
#1 ENTRADAS: EXTRACTED_DECISION_TABLE (STR) - STRING CONTENDO A TABELA DE
DECISAO EXTRAIDA
#1 SAIDAS: CONDITIONS (LIST) - LISTA DE CONDICOES EXTRAIDAS DA TABELA DE
DECISAO
#1 ROTINAS CHAMADAS: __INIT__, PARSE
#1]

import re

class ConditionParser():

    #1[
    #1 ROTINA: __init__
    #1 FINALIDADE: INICIALIZAR A INSTANCIA DA CLASSE CONDITIONPARSER
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: N/A
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: __init__
    def __init__(self) -> None:
    #2  INICIALIZA A INSTANCIA SEM CONFIGURACOES ADICIONAIS
        pass
    #2]

    #1[
    #1 ROTINA: PARSE
    #1 FINALIDADE: EXTRAIR E PROCESSAR AS CONDICOES DA TABELA DE DECISAO
FORNECIDA COMO STRING
    #1 ENTRADAS: EXTRACTED_DECISION_TABLE (STR) - STRING CONTENDO A
TABELA DE DECISAO EXTRAIDA
    #1 DEPENDENCIAS: RE (MODULO DE EXPRESOES REGULARES)
    #1 CHAMADO POR: N/A
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: parse
    def parse(self, extracted_decision_table: str) -> list:
    #2  TRATA OS TIPOS DE CONDICOES PASSADAS
        conditions = []
    #2  PROCURA A SECAO DA TABELA DE DECISAO QUE CONTEM AS CONDICOES E
ACOES
        match = re.search(r'(#TD conditions.*?#TD actions)',
extracted_decision_table, re.DOTALL)
    #2  LEVANTA UMA EXCECAO SE A DEFINICAO DE CONDICOES NAO FOR
ENCONTRADA
        if match is None:
            raise ValueError(f' [-] Erro na busca pela definicao do
condition: {match} e {extracted_decision_table}')
    #2  ITERA SOBRE AS LINHAS DE CONDICOES EXTRAIDAS E AS ADICIONA A
LISTA
        for condition_line in [line.split()[1:] for line in
match.group(0).split('')][1:-1]:
            conditions.append([condition_line[0], condition_line[1:]])
    #2  RETORNA A LISTA DE CONDICOES
        return conditions
    #2]
```

Nome do Arquivo

code_reader.py

Documentação

```
#1[
#1 TITULO: CODE_READER.PY
#1 AUTOR: EDUARDO RIBEIRO SILVA DE OLIVEIRA
#1 DATA: 25/08/2024
#1 VERSAO: 1
```

```
#1 FINALIDADE: LER UM ARQUIVO PYTHON E EXTRAIR TABELAS DE DECISAO
#1 ENTRADAS: CAMINHO DO ARQUIVO (STRING)
#1 SAIDAS: TABELAS DE DECISAO EXTRAIDAS E SUAS POSICOES
#1 ROTINAS CHAMADAS: SET_PATH_CODE, SET_CODE,
SET_EXTRACTED_DECISION_TABLES, _IS_VALID_PATH, _IS_VALID_CODE,
_FIND_DECISION_TABLES, _GET_LINE_COL_FROM_POS_END,
_GET_LINE_COL_FROM_POS_START
#1]

import os
import re

class CodeReader:

    #1[
    #1 ROTINA: __init__
    #1 FINALIDADE: INICIALIZAR A CLASSE E DEFINIR OS ATRIBUTOS INICIAIS
    #1 ENTRADAS: CAMINHO DO ARQUIVO (STR)
    #1 DEPENDENCIAS: SET_PATH_CODE, SET_CODE,
SET_EXTRACTED_DECISION_TABLES
    #1 CHAMADO POR: EXTERNO
    #1 CHAMA: SET_PATH_CODE, SET_CODE, SET_EXTRACTED_DECISION_TABLES
    #1]
    #2[
    #2 PSEUDOCODIGO DE: __init__
    def __init__(self, path_code: str) -> None:
    #2  DEFINE O CAMINHO DO ARQUIVO
        self.set_path_code(path_code)
    #2  LE E ARMAZENA O CODIGO DO ARQUIVO
        self.set_code()
    #2  EXTRAI E ARMAZENA AS TABELAS DE DECISAO
        self.set_extracted_decision_tables()
    #2]

    #1[
    #1 ROTINA: __str__
    #1 FINALIDADE: RETORNAR UMA REPRESENTACAO STRING DO OBJETO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: GET_PATH_CODE, GET_CODE,
GET_EXTRACTED_DECISION_TABLES
    #1 CHAMADO POR: EXTERNO
    #1 CHAMA: GET_PATH_CODE, GET_CODE, GET_EXTRACTED_DECISION_TABLES
    #1]
    #2[
    #2 PSEUDOCODIGO DE: __str__
    def __str__(self) -> str:
    #2  RETORNA UMA REPRESENTACAO STRING DO OBJETO CODEREADER
        return (f' [+] CodeReader:'
                f'     path_code: {self.get_path_code()}'
                f'     len(code): {len(self.get_code())}'
                f'     len(extracted_decision_tables):
{len(self.get_extracted_decision_tables())}')
    #2]

    #1[
    #1 ROTINA: SET_PATH_CODE
    #1 FINALIDADE: VALIDAR E DEFINIR O CAMINHO DO ARQUIVO
    #1 ENTRADAS: CAMINHO DO ARQUIVO (STR)
    #1 DEPENDENCIAS: _IS_VALID_PATH
    #1 CHAMADO POR: __init__
    #1 CHAMA: _IS_VALID_PATH
    #1]
    #2[
    #2 PSEUDOCODIGO DE: set_path_code
    def set_path_code(self, path_code: str) -> None:
    #2  VALIDA E ARMAZENA O CAMINHO DO ARQUIVO
        self.path_code = self._is_valid_path(path_code)
    #2]

    #1[
    #1 ROTINA: SET_CODE
    #1 FINALIDADE: LER O CODIGO DO ARQUIVO E ARMAZENAR
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: _IS_VALID_CODE
    #1 CHAMADO POR: __init__
    #1 CHAMA: _IS_VALID_CODE
    #1]
    #2[
    #2 PSEUDOCODIGO DE: set_code
    def set_code(self) -> None:
    #2  LE O CODIGO DO ARQUIVO E ARMAZENA EM SELF.CODE
        self.code = self._is_valid_code()
    #2]
```

```
    #1[
    #1 ROTINA: SET_EXTRACTED_DECISION_TABLES
    #1 FINALIDADE: EXTRAR E ARMAZENAR AS TABELAS DE DECISAO E SUAS
POSICOES
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: _FIND_DECISION_TABLES
    #1 CHAMADO POR: __init__
    #1 CHAMA: _FIND_DECISION_TABLES
    #1]
    #2[
    #2 PSEUDOCODIGO DE: set_extracted_decision_tables
    def set_extracted_decision_tables(self) -> None:
    #2  EXTRAI AS TABELAS DE DECISAO E SUAS POSICOES
        self.extracted_decision_tables, self.positions =
self._find_decision_tables()
    #2]

    #1[
    #1 ROTINA: GET_PATH_CODE
    #1 FINALIDADE: RETORNAR O CAMINHO DO ARQUIVO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: __str__
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: get_path_code
    def get_path_code(self) -> str:
    #2  RETORNA O CAMINHO DO ARQUIVO
        return self.path_code
    #2]

    #1[
    #1 ROTINA: GET_CODE
    #1 FINALIDADE: RETORNAR O CODIGO LIDO DO ARQUIVO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: __str__
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: get_code
    def get_code(self) -> str:
    #2  RETORNA O CODIGO DO ARQUIVO
        return self.code
    #2]

    #1[
    #1 ROTINA: GET_EXTRACTED_DECISION_TABLES_POSITIONS
    #1 FINALIDADE: RETORNAR AS POSICOES DAS TABELAS DE DECISAO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: EXTERNO
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: get_extracted_decision_tables_positions
    def get_extracted_decision_tables_positions(self) -> list:
    #2  RETORNA AS POSICOES DAS TABELAS DE DECISAO
        return self.positions
    #2]

    #1[
    #1 ROTINA: GET_EXTRACTED_DECISION_TABLES
    #1 FINALIDADE: RETORNAR AS TABELAS DE DECISAO EXTRAIDAS
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: EXTERNO
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: get_extracted_decision_tables
    def get_extracted_decision_tables(self) -> list:
    #2  RETORNA AS TABELAS DE DECISAO EXTRAIDAS
        return self.extracted_decision_tables
    #2]

    #1[
    #1 ROTINA: GET_POSITIONS
    #1 FINALIDADE: RETORNAR AS POSICOES DAS TABELAS DE DECISAO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
```

```
    #1 CHAMADO POR: EXTERNO
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: get_positions
    def get_positions(self) -> list:
    #2  RETORNA AS POSICOES DAS TABELAS DE DECISAO
        return self.positions
    #2]

    #1[
    #1 ROTINA: _IS_VALID_PATH
    #1 FINALIDADE: VALIDAR O CAMINHO DO ARQUIVO
    #1 ENTRADAS: CAMINHO DO ARQUIVO (STR)
    #1 DEPENDENCIAS: OS
    #1 CHAMADO POR: SET_PATH_CODE
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _is_valid_path
    def _is_valid_path(self, path_code: str) -> str:
    #2  VERIFICA SE O CAMINHO FORNECIDO E UM ARQUIVO VALIDO
        if not os.path.isfile(path_code):
            raise FileNotFoundError(f" [-] O PATH PASSADO {path_code} NAO
E UM ARQUIVO.")

    #2  VERIFICA SE O ARQUIVO TEM EXTENSAO .PY
        if not path_code.endswith('.py'):
            raise ValueError(f" [-] O PATH DO ARQUIVO PASSADO {path_code}
NAO E UM ARQUIVO PYTHON")

    #2  RETORNA O CAMINHO DO ARQUIVO SE FOR VALIDO
        return path_code
    #2]

    #1[
    #1 ROTINA: _IS_VALID_CODE
    #1 FINALIDADE: VALIDAR O CONTEUDO DO ARQUIVO DE CODIGO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: SET_CODE
    #1 CHAMA: GET_PATH_CODE
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _is_valid_code
    def _is_valid_code(self) -> str:
    #2  ABRE O ARQUIVO E LE O CONTEUDO
        with open(self.get_path_code(), 'r') as file:
            content = file.read().strip()

    #2  VERIFICA SE O CONTEUDO DO ARQUIVO NAO ESTA VAZIO
        if len(content) == 0:
            raise ValueError(f' [-] O ARQUIVO PASSADO
{self.get_path_code()} ESTA VAZIO')

    #2  RETORNA O CONTEUDO DO ARQUIVO SE NAO ESTIVER VAZIO
        return content
    #2]

    #1[
    #1 ROTINA: _FIND_DECISION_TABLES
    #1 FINALIDADE: ENCONTRAR TABELAS DE DECISAO NO CODIGO E SUAS POSICOES
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: RE
    #1 CHAMADO POR: SET_EXTRACTED_DECISION_TABLES
    #1 CHAMA: GET_CODE, _GET_LINE_COL_FROM_POS_END,
_GET_LINE_COL_FROM_POS_START
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _find_decision_tables
    def _find_decision_tables(self) -> tuple:
    #2  INICIALIZA UMA LISTA PARA ARMAZENAR AS TABELAS DE DECISAO
ENCONTRADAS
        decisions_table_found = []
    #2  INICIALIZA UMA LISTA PARA ARMAZENAR AS POSICOES DAS TABELAS DE
DECISAO
        positions = []
    #2  DEFINE UM PADRAO REGEX PARA ENCONTRAR TABELAS DE DECISAO
        pattern = re.compile(r'(#TD decision table.*?#TD end table)',
re.DOTALL)

    #2  PROCURA POR OCORRENCIAS DO PADRAO REGEX NO CODIGO
        for match in pattern.finditer(self.get_code()):
```

```
    #2      OBTEM A POSICAO DE INICIO DA TABELA DE DECISAO
            start = match.start()
    #2      OBTEM A POSICAO DE FIM DA TABELA DE DECISAO
            end = match.end()

    #2      OBTEM A LINHA E COLUNA DA POSICAO DE FIM DA TABELA
            end_line, end_column = self._get_line_col_from_pos_end(end)

    #2      OBTEM A LINHA E COLUNA DA POSICAO DE INICIO DA TABELA
            start_line, start_column =
self._get_line_col_from_pos_start(start)

    #2      ARMAZENA A TABELA DE DECISAO ENCONTRADA
            decision_table = match.group(0)
    #2      ADICIONA A TABELA ENCONTRADA A LISTA DE TABELAS
            decisions_table_found.append(decision_table)
    #2      ARMAZENA A POSICAO DA TABELA
            positions.append(((start_line, start_column), (end_line,
end_column)))

    #2  LEVANTA UMA EXCECAO SE NENHUMA TABELA DE DECISAO FOR ENCONTRADA
        if not decisions_table_found:
            raise ValueError(f' [-] O ARQUIVO NAO POSSUI NENHUMA TABELA
DE DECISAO')

    #2  RETORNA AS TABELAS DE DECISAO E SUAS POSICOES
        return decisions_table_found, positions
    #2]

    #1[
    #1 ROTINA: _GET_LINE_COL_FROM_POS_END
    #1 FINALIDADE: OBTER A LINHA E COLUNA DA POSICAO FINAL DE UMA TABELA
DE DECISAO
    #1 ENTRADAS: POSICAO (INT)
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: _FIND_DECISION_TABLES
    #1 CHAMA: GET_PATH_CODE
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _get_line_col_from_pos_end
    def _get_line_col_from_pos_end(self, pos: int) -> tuple:
    #2  ABRE O ARQUIVO E LE O CONTEUDO ATE A POSICAO FINAL
        with open(self.get_path_code(), 'r') as file:
            content = file.read()

    #2  SEPARA O CONTEUDO EM LINHAS ATE A POSICAO ESPECIFICADA
        lines = content[:pos].splitlines()

    #2  OBTEM O NUMERO DA LINHA E DA COLUNA PARA A POSICAO FINAL
        line_number = len(lines)
        column_number = len(lines[-1].replace('#TD end table','')) if
lines else 0

    #2  RETORNA O NUMERO DA LINHA E COLUNA PARA A POSICAO FINAL
        return line_number, column_number
    #2]

    #1[
    #1 ROTINA: _GET_LINE_COL_FROM_POS_START
    #1 FINALIDADE: OBTER A LINHA E COLUNA DA POSICAO INICIAL DE UMA
TABELA DE DECISAO
    #1 ENTRADAS: POSICAO (INT)
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: _FIND_DECISION_TABLES
    #1 CHAMA: GET_PATH_CODE
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _get_line_col_from_pos_start
    def _get_line_col_from_pos_start(self, pos: int) -> tuple:
    #2  ABRE O ARQUIVO E LE O CONTEUDO ATE A POSICAO INICIAL
        with open(self.get_path_code(), 'r') as file:
            content = file.read()

    #2  SEPARA O CONTEUDO EM LINHAS ATE A POSICAO ESPECIFICADA
        lines = content[:pos].splitlines()

    #2  OBTEM O NUMERO DA LINHA E DA COLUNA PARA A POSICAO INICIAL
        line_number = len(lines)
        column_number = len(lines[-1].replace('#TD decision table',''))
if lines else 0

    #2  RETORNA O NUMERO DA LINHA E COLUNA PARA A POSICAO INICIAL
        return line_number - 1, column_number
```

```
    #2]
```

Nome do Arquivo
decision_table.py

Documentação

```
#1[
#1 TITULO: DECISION_TABLE.PY
#1 AUTOR: EDUARDO RIBEIRO SILVA DE OLIVEIRA
#1 DATA: 25/08/2024
#1 VERSAO: 1
#1 FINALIDADE: GERENCIAR E PROCESSAR TABELAS DE DECISAO, INCLUINDO
PARSING E VALIDACAO DOS DADOS
#1 ENTRADAS: TABELA DE DECISAO EM FORMATO DE STRING
#1 SAIDAS: ATRIBUTOS DA TABELA DE DECISAO, INCLUINDO NOME, CONJUNTOS,
CONDICOES E ACOES
#1 ROTINAS CHAMADAS: SETPARSER, CONDITIONPARSER, ACTIONPARSER
#1]

from .set import SetParser
from .condition import ConditionParser
from .action import ActionParser #USO DE PATH RELATIVO

class DecisionTable:

    #2 IGNORE: CONSTANTE QUE REPRESENTA A IGNORANCIA DE UMA CONDICAO
    IGNORE = '!= None'

    #2 LISTA DE PALAVRAS-CHAVE QUE DEVEM ESTAR PRESENTES NA TABELA DE
DECISAO
    keywords = ['DECISION TABLE','SETS','CONDITIONS','ACTIONS', 'END
TABLE']

    #2 INSTANCIA UM OBJETO DO PARSER DE CONJUNTOS
    _set_parser = SetParser()

    #2 INSTANCIA UM OBJETO DO PARSER DE CONDICOES
    _condition_parser = ConditionParser()

    #2 INSTANCIA UM OBJETO DO PARSER DE ACOES
    _action_parser = ActionParser()

    #1[
    #1 ROTINA: __init__
    #1 FINALIDADE: INICIALIZAR A TABELA DE DECISAO COM OS ATRIBUTOS
EXTRAIDOS DA TABELA FORNECIDA
    #1 ENTRADAS: EXTRACTED_DESION_TABLE (STR), POSITION (LIST)
    #1 DEPENDENCIAS: DECISION_TABLE.SET.SETPARSER,
DECISION_TABLE.CONDITION.CONDITIONPARSER,
DECISION_TABLE.ACTION.ACTIONPARSER
    #1 CHAMADO POR: N/A
    #1 CHAMA: SET_POSITION_OF_DECISION_TABLE_DETECTED,
SET_EXTRACTED_DECISION_TABLE, SET_NAME, SET_SETS, SET_CONDITIONS,
SET_ACTIONS
    #1]
    #2[
    #2 PSEUDOCODIGO DE: __init__
    def __init__(self, extracted_desion_table: str, position: list): #AS
POSIÇÕES INDICAM A LINHA DE INICIO
    #2  DEFINE A POSICAO DETECTADA DA TABELA DE DECISAO
        self.set_position_of_decision_table_detected(position)
    #2  DEFINE O ATRIBUTO EXTRACTED_DECISION_TABLE
        self.set_extracted_decision_table(extracted_desion_table)
    #2  DEFINE O NOME DA TABELA DE DECISAO
        self.set_name()
    #2  DEFINE OS CONJUNTOS DA TABELA DE DECISAO
        self.set_sets()
    #2  DEFINE AS CONDICOES DA TABELA DE DECISAO
        self.set_conditions()
    #2  DEFINE AS ACOES DA TABELA DE DECISAO
        self.set_actions()
    #2]

    #1[
    #1 ROTINA: __str__
    #1 FINALIDADE: FORNECER UMA REPRESENTACAO EM STRING DO OBJETO
DECISIONTABLE
    #1 ENTRADAS: N/A
```

```
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: PRINT
    #1 CHAMA: GET_EXTRACTED_DECISION_TABLE, GET_NAME, GET_SETS,
GET_CONDITIONS, GET_ACTIONS
    #1]
    #2[
    #2 PSEUDOCODIGO DE: __str__
    def __str__(self) -> str:
    #2  RETORNA UMA STRING FORMATADA COM OS ATRIBUTOS DA TABELA DE
DECISAO
        return f' [+] DecisionTable:     extracted_decision_table:
{self.get_extracted_decision_table()}     name: {self.get_name()}
sets: {self.get_sets()}     conditions: {self.get_conditions()}
actions: {self.get_actions()}'
    #2]

    #1[
    #1 ROTINA: SET_POSITION_OF_DECISION_TABLE_DETECTED
    #1 FINALIDADE: DEFINIR O ATRIBUTO POSITION_OF_DECISION_TABLE_DETECTED
    #1 ENTRADAS: POSITION (LIST)
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: __INIT__
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: set_position_of_decision_table_detected
    def set_position_of_decision_table_detected(self, position: list) ->
None:
    #2  DEFINE O ATRIBUTO POSITION_OF_DECISION_TABLE_DETECTED
        self.position_of_decision_table = position
    #2]

    #1[
    #1 ROTINA: SET_EXTRACTED_DECISION_TABLE
    #1 FINALIDADE: DEFINIR O ATRIBUTO EXTRACTED_DECISION_TABLE APOS
VALIDACAO
    #1 ENTRADAS: EXTRACTED_DECISION_TABLE (STR)
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: __INIT__
    #1 CHAMA: _IS_VALID_DECISION_TABLE
    #1]
    #2[
    #2 PSEUDOCODIGO DE: set_extracted_decision_table
    def set_extracted_decision_table(self, extracted_decision_table: str)
-> None:
    #2  DEFINE O ATRIBUTO EXTRACTED_DECISION_TABLE COM BASE NA VALIDACAO
        self.extracted_decision_table =
self._is_valid_decision_table(extracted_decision_table)
    #2]

    #1[
    #1 ROTINA: SET_NAME
    #1 FINALIDADE: DEFINIR O ATRIBUTO NAME COM BASE NA PRIMEIRA LINHA DA
TABELA DE DECISAO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: __INIT__
    #1 CHAMA: _IS_VALID_NAME
    #1]
    #2[
    #2 PSEUDOCODIGO DE: set_name
    def set_name(self) -> None:
    #2  DEFINE O ATRIBUTO NAME APOS VALIDACAO
        self.name = self._is_valid_name()
    #2]

    #1[
    #1 ROTINA: SET_SETS
    #1 FINALIDADE: DEFINIR O ATRIBUTO SETS COM BASE NO PARSING DA TABELA
DE DECISAO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: DECISION_TABLE.SET.SETPARSER
    #1 CHAMADO POR: __INIT__
    #1 CHAMA: _SETPARSER.PARSE
    #1]
    #2[
    #2 PSEUDOCODIGO DE: set_sets
    def set_sets(self) -> None:
    #2  DEFINE O ATRIBUTO SETS UTILIZANDO O PARSER DE CONJUNTOS
        self.sets =
self._set_parser.parse(self.get_extracted_decision_table())
    #2]
```

```
    #1[
    #1 ROTINA: SET_CONDITIONS
    #1 FINALIDADE: DEFINIR O ATRIBUTO CONDITIONS COM BASE NO PARSING DA
TABELA DE DECISAO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: DECISION_TABLE.CONDITION.CONDITIONPARSER
    #1 CHAMADO POR: __INIT__
    #1 CHAMA: _CONDITIONPARSER.PARSE
    #1]
    #2[
    #2 PSEUDOCODIGO DE: set_conditions
    def set_conditions(self) -> None:
    #2  DEFINE O ATRIBUTO CONDITIONS UTILIZANDO O PARSER DE CONDICOES
        self.conditions =
self._condition_parser.parse(self.get_extracted_decision_table())
    #2]

    #1[
    #1 ROTINA: SET_ACTIONS
    #1 FINALIDADE: DEFINIR O ATRIBUTO ACTIONS COM BASE NO PARSING DA
TABELA DE DECISAO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: DECISION_TABLE.ACTION.ACTIONPARSER
    #1 CHAMADO POR: __INIT__
    #1 CHAMA: _ACTIONPARSER.PARSE
    #1]
    #2[
    #2 PSEUDOCODIGO DE: set_actions
    def set_actions(self) -> None:
    #2  DEFINE O ATRIBUTO ACTIONS UTILIZANDO O PARSER DE ACOES
        self.actions =
self._action_parser.parse(self.get_extracted_decision_table())
    #2]

    #1[
    #1 ROTINA: GET_EXTRACTED_DECISION_TABLE
    #1 FINALIDADE: RETORNAR O ATRIBUTO EXTRACTED_DECISION_TABLE
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: __STR__, GET_DECISION_TABLE
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: get_extracted_decision_table
    def get_extracted_decision_table(self) -> str:
    #2  RETORNA O ATRIBUTO EXTRACTED_DECISION_TABLE
        return self.extracted_decision_table
    #2]

    #1[
    #1 ROTINA: GET_KEYWORDS
    #1 FINALIDADE: RETORNAR A LISTA DE PALAVRAS-CHAVE UTILIZADAS NA
TABELA DE DECISAO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: N/A
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: get_keywords
    def get_keywords(self) -> list:
    #2  RETORNA A LISTA DE PALAVRAS-CHAVE
        return self.keywords
    #2]

    #1[
    #1 ROTINA: GET_NAME
    #1 FINALIDADE: RETORNAR O NOME DA TABELA DE DECISAO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: __STR__, GET_DECISION_TABLE
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: get_name
    def get_name(self) -> str:
    #2  RETORNA O ATRIBUTO NAME
        return self.name
    #2]

    #1[
    #1 ROTINA: GET_SETS
    #1 FINALIDADE: RETORNAR UMA COPIA DA LISTA DE CONJUNTOS DA TABELA DE
```

```
DECISAO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: __STR__, GET_DECISION_TABLE
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: get_sets
    def get_sets(self) -> list:
    #2  RETORNA UMA COPIA DO ATRIBUTO SETS
        return self.sets.copy()
    #2]

    #1[
    #1 ROTINA: GET_CONDITIONS
    #1 FINALIDADE: RETORNAR UMA COPIA DA LISTA DE CONDICOES DA TABELA DE
DECISAO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: __STR__, GET_DECISION_TABLE
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: get_conditions
    def get_conditions(self) -> list:
    #2  RETORNA UMA COPIA DO ATRIBUTO CONDITIONS
        return self.conditions.copy()
    #2]

    #1[
    #1 ROTINA: GET_ACTIONS
    #1 FINALIDADE: RETORNAR UMA COPIA DA LISTA DE ACOES DA TABELA DE
DECISAO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: __STR__, GET_DECISION_TABLE
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: get_actions
    def get_actions(self) -> list:
    #2  RETORNA UMA COPIA DO ATRIBUTO ACTIONS
        return self.actions.copy()
    #2]

    #1[
    #1 ROTINA: GET_DECISION_TABLE
    #1 FINALIDADE: RETORNAR UM DICIONARIO COM TODOS OS VALORES EXTRAIDOS
DA TABELA DE DECISAO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: N/A
    #1 CHAMA: GET_NAME, GET_SETS, GET_CONDITIONS, GET_ACTIONS
    #1]
    #2[
    #2 PSEUDOCODIGO DE: get_decision_table
    def get_decision_table(self) -> dict:
    #2  CRIA E RETORNA UM DICIONARIO COM TODOS OS ATRIBUTOS DA TABELA DE
DECISAO
        return {
            'name': self.get_name(),
            'sets': self.get_sets(),
            'conditions': self.get_conditions(),
            'actions': self.get_actions()
        }
    #2]

    #1[
    #1 ROTINA: _IS_VALID_DECISION_TABLE
    #1 FINALIDADE: VERIFICAR A VALIDADE DA TABELA DE DECISAO ATRAVES DA
PRESENCA DE PALAVRAS-CHAVE
    #1 ENTRADAS: EXTRACTED_DECISION_TABLE (STR)
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: SET_EXTRACTED_DECISION_TABLE
    #1 CHAMA: GET_KEYWORDS
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _is_valid_decision_table
    def _is_valid_decision_table(self, extracted_decision_table: str) ->
str:
    #2  VERIFICA SE A TABELA DE DECISAO CONTEM TODAS AS PALAVRAS-CHAVE
NECESSARIAS
        for keyword in self.get_keywords():
```

```
            if keyword not in extracted_decision_table.upper():
                raise ValueError(f' [-] Tabela de decisao nao possui
{keyword}: {extracted_decision_table}')
    #2  RETORNA A TABELA DE DECISAO VALIDADA
        return extracted_decision_table
    #2]

    #1[
    #1 ROTINA: _IS_VALID_NAME
    #1 FINALIDADE: VERIFICAR A VALIDADE DO NOME DA TABELA DE DECISAO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: SET_NAME
    #1 CHAMA: GET_EXTRACTED_DECISION_TABLE
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _is_valid_name
    def _is_valid_name(self) -> str:
    #2  OBTEM A PRIMEIRA LINHA DA TABELA DE DECISAO E REMOVE ESPACOS EM
BRANCO
        decision_table_lines =
self.get_extracted_decision_table().split('')[0].strip()
    #2  DIVIDE A PRIMEIRA LINHA DA TABELA DE DECISAO PELOS ESPACOS EM
BRANCO E IGNORA OS 3 PRIMEIROS ELEMENTOS
        first_line_splitted_by_empty_spaces =
decision_table_lines.split(' ')[3:]
    #2  VERIFICA SE O NOME DA TABELA E VALIDO
        if first_line_splitted_by_empty_spaces == []:
            raise ValueError(f' [-] Tabela de decisao nao possui nome
{first_line_splitted_by_empty_spaces}: {decision_table_lines}')
    #2  RETORNA O NOME VALIDADO
        return ' '.join(first_line_splitted_by_empty_spaces)
    #2]

    #1[
    #1 ROTINA: _SET_TRANSLATED_SET_BY_NAME
    #1 FINALIDADE: CRIAR UM DICIONARIO PARA TRADUZIR OS NOMES DOS
CONJUNTOS EM CODIGO
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: GET_TRANSLATED_SET_BY_NAME
    #1 CHAMA: GET_SETS
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _set_translated_set_by_name
    def _set_translated_set_by_name(self) -> None:
    #2  DEFINE AS TRADUCOES PADRAO PARA 'Y', 'N', E '-'
        self._translated_set_by_name = {
            'Y': '== True',
            'N': '== False',
            '-': self.IGNORE
            }
    #2  ADICIONA AS TRADUCOES DOS CONJUNTOS OBTIDOS DA TABELA DE DECISAO
        for td_set_name, td_set_value in self.get_sets():
            self._translated_set_by_name[td_set_name] = td_set_value
    #2]

    #1[
    #1 ROTINA: _SET_SEQUENCE_OF_ACTIONS
    #1 FINALIDADE: CRIAR UM DICIONARIO QUE MAPEIA INDICES PARA LISTAS DE
ACOES
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: GET_SEQUENCE_OF_ACTIONS_BY_ID
    #1 CHAMA: GET_ACTIONS
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _set_sequence_of_actions
    def _set_sequence_of_actions(self) -> None:
    #2  CRIA UM DICIONARIO QUE MAPEIA INDICES PARA LISTAS DE ACOES
ORDENADAS
        self._sequence_of_actions = {}
    #2  ITERA SOBRE CADA ACAO E SEU ORDENAMENTO OBTIDOS DA TABELA DE
DECISAO
        for action_name, action_ordering in self.get_actions():
    #2      ITERA SOBRE CADA INDICE E ORDEM DA ACAO
            for index, action_order in enumerate(action_ordering):
    #2          CRIA UMA ENTRADA PARA O INDICE NO DICIONARIO, CASO NAO
EXISTA
                self._sequence_of_actions[index] =
self._sequence_of_actions.get(index, [])
    #2          ADICIONA A ACAO AO INDICE SE A ORDEM NAO FOR '0'
                if action_order != '0':
```

```
self._sequence_of_actions[index].append((int(action_order), action_name))
    #2  ORDENA AS ACOES POR ORDEM NO DICIONARIO
        for index in self._sequence_of_actions.keys():
            if len(self._sequence_of_actions[index]) > 1:
                self._sequence_of_actions[index].sort(key=lambda x: x[0])
    #2]

    #1[
    #1 ROTINA: GET_SEQUENCE_OF_ACTIONS_BY_ID
    #1 FINALIDADE: RETORNAR A SEQUENCIA DE ACOES PARA UM DADO ID
    #1 ENTRADAS: ACTION_ID (INT)
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: N/A
    #1 CHAMA: _SET_SEQUENCE_OF_ACTIONS
    #1]
    #2[
    #2 PSEUDOCODIGO DE: get_sequence_of_actions_by_id
    def get_sequence_of_actions_by_id(self, action_id: int) -> list:
    #2  VERIFICA SE A SEQUENCIA DE ACOES JA FOI DEFINIDA, SE NAO, CRIA-A
        if not hasattr(self, '_sequence_of_actions'):
            self._set_sequence_of_actions()
    #2  VERIFICA SE O INDICE RECEBIDO E VALIDO
        if self._sequence_of_actions.get(action_id) is None:
            raise ValueError(f' [-] Indice da sequencia de acoes
invalido. Dicionario de sequencia de acoes: {self._sequence_of_actions}
Indice recebido: {action_id}')
    #2  RETORNA UMA LISTA VAZIA SE NAO HOUVER ACOES PARA O INDICE
        elif len(self._sequence_of_actions[action_id]) == 0:
            return []
    #2  RETORNA A SEQUENCIA DE ACOES ASSOCIADAS AO INDICE
        else:
            return [action_tuple[1] for action_tuple in
self._sequence_of_actions[action_id]]
    #2]

    #1[
    #1 ROTINA: GET_TRANSLATED_SET_BY_NAME
    #1 FINALIDADE: RETORNAR A TRADUCAO DO NOME DO CONJUNTO EM CODIGO
    #1 ENTRADAS: SET_NAME (STR)
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: N/A
    #1 CHAMA: _SET_TRANSLATED_SET_BY_NAME
    #1]
    #2[
    #2 PSEUDOCODIGO DE: get_translated_set_by_name
    def get_translated_set_by_name(self, set_name: str) -> str:
    #2  VERIFICA SE O DICIONARIO DE TRADUCOES JA FOI DEFINIDO, SE NAO,
CRIA-O
        if not hasattr(self, '_translated_set_by_name'):
            self._set_translated_set_by_name()
    #2  VERIFICA SE O NOME DO CONJUNTO RECEBIDO E VALIDO
        if self._translated_set_by_name.get(set_name) is None:
            raise ValueError(f' [-] Nome do conjunto invalido. Dicionario
de traducoes por nome do conjunto {self._translated_set_by_name} Nome
recebido: {set_name}')
    #2  RETORNA A TRADUCAO DO NOME DO CONJUNTO
        else:
            return self._translated_set_by_name[set_name]
    #2]
```

Nome do Arquivo
code_generator.py
Documentação

```
#1[
#1 TITULO: code_generator.py
#1 AUTOR: EDUARDO RIBEIRO SILVA DE OLIVEIRA
#1 DATA: 25/08/2024
#1 VERSAO: 1
#1 FINALIDADE: GERAR CODIGO A PARTIR DE TABELAS DE DECISAO UTILIZANDO
DIFERENTES METODOS DE TRADUCAO.
#1 ENTRADAS: TABELA DE DECISAO (DECISIONTABLE)
#1 SAIDAS: CODIGO PYTHON GERADO COM BASE NA TABELA DE DECISAO
#1 ROTINAS CHAMADAS: SET_METHOD, GET_METHOD,
PRODUCT_OF_ENTRIES_BY_CONDITION, LIST_ENTRIES_BY_CONDITION,
_GENERATE_DOCUMENTATION_CODE, _GENERATE_INITIALIZATION_CODE,
_GENERATE_IF_OR_ELIF_CODE, _GENERATE_ACTION_ID_CALCULATION_CODE,
```

```
_GENERATE_MATCH_CODE, _SWITCH_METHOD, _FATORACOES_SUCESSIVAS,
_BUSCA_EXAUSTIVA, _PROGRAMACAO_DINAMICA, GENERATE_CODE
#1]

from src.decision_table.decision_table import DecisionTable

class CodeGenerator():

    #1[
    #1 ROTINA: __init__
    #1 FINALIDADE: INICIALIZAR A INSTANCIA DA CLASSE CODEGENERATOR.
    #1 ENTRADAS: INITIAL_SPACING (STR), DEFAULT_SPACING (STR)
    #1 DEPENDENCIAS: DECISION_TABLE.DECISION_TABLE.DECISIONTABLE
    #1 CHAMADO POR: N/A
    #1 CHAMA: SET_METHOD
    #1]
    #2[
    #2 PSEUDOCODIGO DE: __init__
    def __init__(self, initial_spacing:str='', default_spacing:str=''):
    #2  INICIALIZA O DICIONARIO DE METODOS DE TRADUCAO
        self._decision_table_tradution_methods = {
            'switch_method' : self._switch_method,
            'fatoracoes_sucessivas': self._fatoracoes_sucessivas,
            'busca_exaustiva' : self._busca_exaustiva,
            'programacao_dinamica' : self._programacao_dinamica
        }
    #2  DEFINE O ESPACAMENTO INICIAL E O ESPACAMENTO PADRAO
        self.initial_spacing = initial_spacing
        self.default_spacing = default_spacing
    #2  DEFINE O METODO PADRAO COMO 'SWITCH_METHOD'
        self.set_method('switch_method')
    #2]

    #1[
    #1 ROTINA: set_method
    #1 FINALIDADE: DEFINIR O METODO DE TRADUCAO A SER UTILIZADO.
    #1 ENTRADAS: METHOD_NAME (STR)
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: __init__
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: set_method
    def set_method(self, method_name:str) ->None:
    #2  TENTA DEFINIR O METODO DE TRADUCAO COM BASE NO NOME FORNECIDO
        try:
    #2      DEFINE O NOME DO METODO
            self.method_name = method_name
    #2      DEFINE O METODO DE TRADUCAO
            self.method =
self._decision_table_tradution_methods[method_name]
    #2  LEVANTA UMA EXCECAO SE O NOME DO METODO FOR INVALIDO
        except Exception as e:
            raise ValueError(f' [-] Nome inválido. Nomes disponíveis:
{self._decision_table_tradution_methods.keys()} Nome recebido:
{method_name}. Exception: {e}')
    #2]

    #1[
    #1 ROTINA: get_method
    #1 FINALIDADE: RETORNAR O NOME E O METODO DE TRADUCAO ATUAL.
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: N/A
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: get_method
    def get_method(self) ->list:
    #2  RETORNA O NOME DO METODO E O METODO DE TRADUCAO ATUAL
        return self.method_name, self.method
    #2]

    #1[
    #1 ROTINA: product_of_entries_by_condition
    #1 FINALIDADE: CALCULAR O PRODUTO DA QUANTIDADE DE ENTRADAS PARA CADA
CONDICAO EM UMA TABELA DE DECISAO.
    #1 ENTRADAS: TD (DECISIONTABLE)
    #1 DEPENDENCIAS: DECISION_TABLE.DECISION_TABLE.DECISIONTABLE
    #1 CHAMADO POR: _GENERATE_ACTION_ID_CALCULATION_CODE
    #1 CHAMA: N/A
    #1]
    #2[
```

```
    #2 PSEUDOCODIGO DE: product_of_entries_by_condition
    def product_of_entries_by_condition(self, td: DecisionTable) ->int:
    #2  INICIALIZA O PRODUTO DE ENTRADAS COMO 1
        produto_de_entradas = 1
    #2  ITERA SOBRE AS CONDICOES DA TABELA DE DECISAO
        for condicao in td.get_conditions():
    #2      REMOVE O SIMBOLO '-'
            if '-' in condicao[1]:
                condicao[1] = condicao[1] +['Y','N']
    #2      MULTIPLICA O PRODUTO DE ENTRADAS PELO NUMERO DE ENTRADAS
UNICAS DA CONDICAO
            produto_de_entradas *= len(set(condicao[1]))
    #2  RETORNA O PRODUTO DE ENTRADAS
        return produto_de_entradas
    #2]

    #1[
    #1 ROTINA: list_entries_by_condition
    #1 FINALIDADE: RETORNAR UMA LISTA COM A QUANTIDADE DE ENTRADAS PARA
CADA CONDICAO.
    #1 ENTRADAS: TD (DECISIONTABLE)
    #1 DEPENDENCIAS: DECISION_TABLE.DECISION_TABLE.DECISIONTABLE
    #1 CHAMADO POR: _GENERATE_ACTION_ID_CALCULATION_CODE
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: list_entries_by_condition
    def list_entries_by_condition(self, td: DecisionTable) ->list:
    #2  INICIALIZA UMA LISTA DE ENTRADAS
        lista_de_entradas = []
    #2  ITERA SOBRE AS CONDICOES DA TABELA DE DECISAO
        for condicao in td.get_conditions():
    #2      ADICIONA O NUMERO DE ENTRADAS UNICAS DA CONDICAO A LISTA DE
ENTRADAS
            lista_de_entradas.append(len(set(condicao[1])))
    #2  RETORNA A LISTA DE ENTRADAS
        return lista_de_entradas
    #2]

    #1[
    #1 ROTINA: _generate_documentation_code
    #1 FINALIDADE: INSERIR A TABELA DE DECISAO COMO DOCUMENTACAO DE
SEGUNDO NIVEL PARA O EXTRATOR DE AUTO-DOCUMENTACOES.
    #1 ENTRADAS: TD (DECISIONTABLE)
    #1 DEPENDENCIAS: DECISION_TABLE.DECISION_TABLE.DECISIONTABLE
    #1 CHAMADO POR: _SWITCH_METHOD
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _generate_documentation_code
    def _generate_documentation_code(self, td: DecisionTable) ->None:
    #2  INICIA O BLOCO DE DOCUMENTACAO DE SEGUNDO NIVEL
        self.generated_code += f'{self.initial_spacing}#2['
    #2  EXTRAI A TABELA DE DECISAO E FORMATA COMO DOCUMENTACAO
        decision_table_str = td.get_extracted_decision_table()
    #2  ITERA SOBRE CADA LINHA DA TABELA DE DECISAO EXTRAIDA
        for linha in decision_table_str.split(''):
    #2      ADICIONA A LINHA FORMATADA AO CODIGO GERADO
            self.generated_code +=
linha.lstrip().replace('#TD',f'{self.initial_spacing}#2 #TD') + ''
    #2  FINALIZA O BLOCO DE DOCUMENTACAO DE SEGUNDO NIVEL
        self.generated_code += f'{self.initial_spacing}#2]'
    #2]

    #1[
    #1 ROTINA: _generate_initialization_code
    #1 FINALIDADE: GERAR O CODIGO DE INICIALIZACAO DAS VARIAVEIS
AUXILIARES NECESSARIAS PARA A DEFINICAO DA ACAO BASEADA NOS VALORES DAS
CONDICOES.
    #1 ENTRADAS: TD (DECISIONTABLE)
    #1 DEPENDENCIAS: DECISION_TABLE.DECISION_TABLE.DECISIONTABLE
    #1 CHAMADO POR: _SWITCH_METHOD
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _generate_initialization_code
    def _generate_initialization_code(self, td: DecisionTable) ->None:
    #2  ADICIONA A DEFINICAO DA FUNCAO DE TABELA DE DECISAO AO CODIGO
GERADO
        self.generated_code += f'{self.initial_spacing}def
decision_table_{td.get_name()}() ->None:'
    #2  ITERA SOBRE AS CONDICOES DA TABELA DE DECISAO
        for index, condicao in enumerate(td.get_conditions()):
```

```
    #2      ADICIONA A INICIALIZACAO DO AUXILIAR DA CONDICAO AO CODIGO
GERADO
            self.generated_code +=
f'{self.initial_spacing+self.default_spacing}I_{index} = 0 #Inicialização
do auxiliar da condição {condicao[0]}'
    #2  ADICIONA A INICIALIZACAO DO NUMERO DA REGRA AO CODIGO GERADO
        self.generated_code +=
f'{self.initial_spacing+self.default_spacing}I   = 0 #Inicialização do
número da regra'
    #2]

    def _generate_invoke(self, td: DecisionTable) -> None:
    #2  GERA O CODIGO PARA INVOCAR A FUNCAO DA TABELA DE DECISAO
        self.generated_code +=
f'{self.initial_spacing}decision_table_{td.get_name()}()'

    #1[
    #1 ROTINA: _generate_if_or_elif_code
    #1 FINALIDADE: GERAR O CODIGO PARA O IF/ELIF EM PYTHON, DADO UMA
CONDICAO, UM VALOR E UM INDICE DO AUXILIAR DA CONDICAO.
    #1 ENTRADAS: TD (DECISIONTABLE), CONDITION (STR), CONDITION_VALUE
(STR), INDEX (INT), AUX_VARIABLE_VALUE (INT), IF_OR_ELIF (STR)
    #1 DEPENDENCIAS: DECISION_TABLE.DECISION_TABLE.DECISIONTABLE
    #1 CHAMADO POR: _SWITCH_METHOD
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _generate_if_or_elif_code
    def _generate_if_or_elif_code(self, td: DecisionTable, condition:str,
condition_value: str, index:int, aux_variable_value:int,if_or_elif:str) -
>None:
    #2  ADICIONA A LINHA DE CODIGO IF/ELIF COM A CONDICAO E O VALOR
        self.generated_code +=
f'{self.initial_spacing+self.default_spacing}{if_or_elif} {condition}
{td.get_translated_set_by_name(condition_value)}:
{self.initial_spacing+2*self.default_spacing}I_{index} =
{aux_variable_value}'
    #2]

    #1[
    #1 ROTINA: _generate_action_id_calculation_code
    #1 FINALIDADE: GERAR O CODIGO QUE SOMA OS VALORES DAS VARIAVEIS
AUXILIARES DAS CONDICOES PARA DEFINIR O INDICE DA ACAO NO MATCH.
    #1 ENTRADAS: TD (DECISIONTABLE)
    #1 DEPENDENCIAS: DECISION_TABLE.DECISION_TABLE.DECISIONTABLE
    #1 CHAMADO POR: _SWITCH_METHOD
    #1 CHAMA: LIST_ENTRIES_BY_CONDITION
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _generate_action_id_calculation_code
    def _generate_action_id_calculation_code(self, td: DecisionTable) ->
None:
    #2  ADICIONA A INICIALIZACAO DO CALCULO DO INDICE DA ACAO AO CODIGO
GERADO
        self.generated_code += f'{self.initial_spacing +
self.default_spacing}I = '
    #2  OBTEM O NUMERO DE ENTRADAS POR CONDICAO
        entries_by_condition = self.list_entries_by_condition(td)
    #2  ITERA SOBRE AS CONDICOES PARA GERAR O CODIGO DE CALCULO DO ID
        for i in range(len(td.get_conditions())):
    #2      ADICIONA O PARENTESES DE ABERTURA AO CODIGO
            self.generated_code += '('
    #2      GERA O CODIGO DE MULTIPLICACAO PARA CALCULAR O INDICE
            for j in range(i+1, len(td.get_conditions())):
                self.generated_code += f'{entries_by_condition[j]}*'
    #2      FINALIZA O CALCULO PARA A CONDICAO ATUAL E ADICIONA AO CODIGO
GERADO
            self.generated_code += f'1)*I_{i} + '
    #2  REMOVE O ULTIMO ' + ' DA EXPRESSAO GERADA
        self.generated_code = self.generated_code[:-3]+''
    #2]

    #1[
    #1 ROTINA: _generate_match_code
    #1 FINALIDADE: GERAR O CODIGO QUE FAZ O MATCH DA INDEXACAO CALCULADA.
    #1 ENTRADAS: TD (DECISIONTABLE)
    #1 DEPENDENCIAS: DECISION_TABLE.DECISION_TABLE.DECISIONTABLE
    #1 CHAMADO POR: _SWITCH_METHOD
    #1 CHAMA: PRODUCT_OF_ENTRIES_BY_CONDITION
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _generate_match_code
    def _generate_match_code(self, td: DecisionTable) -> None:
```

```
    #2  ADICIONA O INICIO DA ESTRUTURA MATCH AO CODIGO GERADO
        self.generated_code +=
f'{self.initial_spacing+self.default_spacing}match I:'
    #2  ITERA SOBRE CADA ACAO PARA GERAR OS CASES DO MATCH
        for M in range(self.product_of_entries_by_condition(td)):
    #2      ADICIONA UM CASE AO MATCH PARA CADA POSSIVEL ACAO
            self.generated_code += f'{self.initial_spacing +
2*self.default_spacing}case {M}:'
    #2      ITERA SOBRE AS ACOES PARA O CASE ATUAL
            for action in td.get_sequence_of_actions_by_id(M):
    #2          ADICIONA A ACAO AO CODIGO GERADO
                self.generated_code += f'{self.initial_spacing +
3*self.default_spacing}{action}'
    #2      ADICIONA UMA QUEBRA DE LINHA APOS CADA CASE
            self.generated_code += ''
    #2  ADICIONA O CASO DEFAULT AO CODIGO GERADO
        self.generated_code += f'{self.initial_spacing +
2*self.default_spacing}case _:{self.initial_spacing +
3*self.default_spacing}exit()'
    #2]

    #1[
    #1 ROTINA: _switch_method
    #1 FINALIDADE: IMPLEMENTAR O METODO DE TRADUCAO DE TABELAS DE
DECISAO: SWITCH METHOD.
    #1 ENTRADAS: TD (DECISIONTABLE)
    #1 DEPENDENCIAS: DECISION_TABLE.DECISION_TABLE.DECISIONTABLE
    #1 CHAMADO POR: GENERATE_CODE
    #1 CHAMA: _GENERATE_DOCUMENTATION_CODE,
_GENERATE_INITIALIZATION_CODE, _GENERATE_IF_OR_ELIF_CODE,
_GENERATE_ACTION_ID_CALCULATION_CODE, _GENERATE_MATCH_CODE
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _switch_method
    def _switch_method(self, td: DecisionTable) ->None:
    #2  INICIALIZA A VARIAVEL DE CODIGO GERADO COMO UMA STRING VAZIA
        self.generated_code = ''
    #2  GERA O CODIGO DE DOCUMENTACAO PARA A TABELA DE DECISAO
        self._generate_documentation_code(td)
    #2  GERA O CODIGO DE INICIALIZACAO DAS VARIAVEIS NECESSARIAS
        self._generate_initialization_code(td)
    #2  ITERA SOBRE AS CONDICOES DA TABELA DE DECISAO
        for index, linha_de_condicao in enumerate(td.get_conditions()):
            C = set()
            n_i = 0
            condicao = linha_de_condicao[0]
            entradas = linha_de_condicao[1:][0]
    #2      ITERA SOBRE AS ENTRADAS DA CONDICAO PARA GERAR O CODIGO
IF/ELIF
            for C_ij in entradas:
                if C_ij not in C:
                    n_i += 1
                    c_ij = C_ij
    #2          GERA O CODIGO IF PARA A PRIMEIRA CONDICAO
                    if len(C) == 0:
                        self._generate_if_or_elif_code(td, condicao,
c_ij, index, 0, 'if')
    #2          GERA O CODIGO ELIF PARA AS CONDICOES SUBSEQUENTES
                    else:
                        self._generate_if_or_elif_code(td, condicao,
c_ij, index, n_i-1, 'elif')
    #2          ADICIONA A ENTRADA AO CONJUNTO DE ENTRADAS PROCESSADAS
                    if c_ij != '-':
                        C.add(c_ij)
    #2          ADICIONA AS ENTRADAS 'Y' E 'N' PARA O SIMBOLO '-'
                    else:
                        C.add('Y')
                        C.add('N')
    #2  GERA O CODIGO PARA CALCULO DO ID DA ACAO
        self._generate_action_id_calculation_code(td)
    #2  GERA O CODIGO MATCH PARA A INDEXACAO CALCULADA
        self._generate_match_code(td)
    #2  GERA O CODIGO PARA INVOCACAO DA FUNCAO DA TABELA DE DECISAO
        self._generate_invoke(td)
    #2]

    #1[
    #1 ROTINA: _fatoracoes_sucessivas
    #1 FINALIDADE: IMPLEMENTAR O METODO DE TRADUCAO DE TABELAS DE
DECISAO: FATORACOES SUCESSIVAS.
    #1 ENTRADAS: DECISION_TABLE (DECISIONTABLE)
    #1 DEPENDENCIAS: DECISION_TABLE.DECISION_TABLE.DECISIONTABLE
    #1 CHAMADO POR: N/A
```

```
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _fatoracoes_sucessivas
    def _fatoracoes_sucessivas(self, decision_table: DecisionTable) -
>None:
    #2  METODO AINDA NAO IMPLEMENTADO
        pass
    #2]

    #1[
    #1 ROTINA: _busca_exaustiva
    #1 FINALIDADE: IMPLEMENTAR O METODO DE TRADUCAO DE TABELAS DE
DECISAO: BUSCA EXAUSTIVA.
    #1 ENTRADAS: DECISION_TABLE (DECISIONTABLE)
    #1 DEPENDENCIAS: DECISION_TABLE.DECISION_TABLE.DECISIONTABLE
    #1 CHAMADO POR: N/A
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _busca_exaustiva
    def _busca_exaustiva(self, decision_table: DecisionTable) ->None:
    #2  METODO AINDA NAO IMPLEMENTADO
        pass
    #2]

    #1[
    #1 ROTINA: _programacao_dinamica
    #1 FINALIDADE: IMPLEMENTAR O METODO DE TRADUCAO DE TABELAS DE
DECISAO: PROGRAMACAO DINAMICA.
    #1 ENTRADAS: DECISION_TABLE (DECISIONTABLE)
    #1 DEPENDENCIAS: DECISION_TABLE.DECISION_TABLE.DECISIONTABLE
    #1 CHAMADO POR: N/A
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: _programacao_dinamica
    def _programacao_dinamica(self, decision_table: DecisionTable) -
>None:
    #2  METODO AINDA NAO IMPLEMENTADO
        pass
    #2]

    #1[
    #1 ROTINA: generate_code
    #1 FINALIDADE: GERAR CODIGO A PARTIR DE UMA TABELA DE DECISAO COM O
METODO DEFINIDO NA INSTANCIACAO.
    #1 ENTRADAS: DECISION_TABLE (DECISIONTABLE)
    #1 DEPENDENCIAS: DECISION_TABLE.DECISION_TABLE.DECISIONTABLE
    #1 CHAMADO POR: N/A
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: generate_code
    def generate_code(self, decision_table: DecisionTable) ->str:
    #2  EXECUTA O METODO DEFINIDO PARA GERAR O CODIGO
        self.method(decision_table)
    #2  RETORNA O CODIGO GERADO
        return self.generated_code
    #2]
```

Nome do Arquivo
action.py
Documentação

```
#1[
#1 TITULO: ACTION.PY
#1 AUTOR: EDUARDO RIBEIRO SILVA DE OLIVEIRA
#1 DATA: 25/08/2024
#1 VERSAO: 1
#1 FINALIDADE: ANALISAR E EXTRAIR ACOES DE UMA TABELA DE DECISAO PASSADA
COMO STRING
#1 ENTRADAS: EXTRACTED_DECISION_TABLE (STR) - STRING CONTENDO A TABELA DE
DECISAO EXTRAIDA COM DEFINICOES DE ACOES
#1 SAIDAS: LIST - LISTA DE ACOES EXTRAIDAS DA TABELA DE DECISAO
#1 ROTINAS CHAMADAS: RE.SEARCH
#1]
```

```
import re

class ActionParser():

    #1[
    #1 ROTINA: __init__
    #1 FINALIDADE: INICIALIZAR A INSTANCIA DA CLASSE ACTIONPARSER
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: N/A
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: __init__
    def __init__(self) -> None:
    #2  INICIALIZA A INSTANCIA SEM PARAMETROS
        pass
    #2]

    #1[
    #1 ROTINA: PARSE
    #1 FINALIDADE: EXTRAIR E TRATAR ACOES DE UMA TABELA DE DECISAO
PASSADA COMO STRING
    #1 ENTRADAS: EXTRACTED_DECISION_TABLE (STR) - STRING CONTENDO A
TABELA DE DECISAO COM DEFINICOES DE ACOES
    #1 DEPENDENCIAS: RE (BIBLIOTECA PADRAO DO PYTHON)
    #1 CHAMADO POR: N/A
    #1 CHAMA: RE.SEARCH
    #1]
    #2[
    #2 PSEUDOCODIGO DE: parse
    def parse(self, extracted_decision_table: str) -> list:
    #2  TRATA OS TIPOS DE ACOES PASSADAS
        actions = []
    #2  BUSCA A SECAO DE ACOES NA TABELA DE DECISAO USANDO EXPRESSAO
REGULAR
        match = re.search(r'(#TD actions.*?#TD end table)',
extracted_decision_table, re.DOTALL)
    #2  LEVANTA UM ERRO SE A BUSCA NAO ENCONTRAR UMA CORRESPONDENCIA
        if match is None:
            raise ValueError(f' [-] Erro na busca pela definicao da
action: {match} e {extracted_decision_table}')
    #2  ITERA SOBRE CADA LINHA DE ACAO EXTRAIDA E ORGANIZA OS DADOS
        for action_line in [line.split()[1:] for line in
match.group(0).split('')][1:-1]:
            actions.append([action_line[0], action_line[1:]])
    #2  RETORNA A LISTA DE ACOES EXTRAIDAS
        return actions
    #2]
```

Nome do Arquivo
set.py

Documentação

```
#1[
#1 TITULO: SET.PY
#1 AUTOR: EDUARDO RIBEIRO SILVA DE OLIVEIRA
#1 DATA: 25/08/2024
#1 VERSAO: 1
#1 FINALIDADE: DEFINIR E TRATAR OS TIPOS DE CONJUNTOS EXTRAIDOS DE UMA
TABELA DE DECISAO
#1 ENTRADAS: EXTRACTED_DECISION_TABLE (STR) - STRING CONTENDO A TABELA DE
DECISAO EXTRAIDA COM DEFINICOES DE CONJUNTOS E CONDICOES
#1 SAIDAS: LIST - LISTA DE CONJUNTOS FORMATADOS E TRATADOS COM BASE NAS
DEFINICOES ENCONTRADAS NA TABELA DE DECISAO
#1 ROTINAS CHAMADAS: RE.SEARCH
#1]

import re

class SetParser():

    #1[
    #1 ROTINA: PARSE
    #1 FINALIDADE: ANALISAR E FORMATAR CONJUNTOS ENCONTRADOS NA TABELA DE
DECISAO
    #1 ENTRADAS: EXTRACTED_DECISION_TABLE (STR) - STRING CONTENDO A
TABELA DE DECISAO EXTRAIDA
```

```
    #1 DEPENDENCIAS: RE
    #1 CHAMADO POR: N/A
    #1 CHAMA: RE.SEARCH
    #1]
    #2[
    #2 PSEUDOCODIGO DE: parse
    def parse(self, extracted_decision_table: str) -> list:
    #2  TRATA OS TIPOS DE CONJUNTOS PASSADOS
        sets = []
    #2  BUSCA PELA DEFINICAO DE CONJUNTOS E CONDICOES NA TABELA DE
DECISAO
        match = re.search(r'(#TD sets.*?#TD conditions)',
extracted_decision_table, re.DOTALL)
    #2  LEVANTA UMA EXCECAO SE NAO ENCONTRAR A DEFINICAO DO SET
        if match is None:
            raise ValueError(f' [-] Erro na busca pela definicao do set:
{match} e {extracted_decision_table}')
    #2  ITERA SOBRE AS DEFINICOES DE CONJUNTOS ENCONTRADAS
        for set_name, set_definition in [line.split()[1:] for line in
match.group(0).split('')][1:-1]:
    #2      VERIFICA SE A DEFINICAO E UM CONJUNTO DELIMITADO POR CHAVES
(EX: {a,b,c})
            if re.search(r'{.*}', set_definition):
    #2          FORMATA A DEFINICAO PARA O FORMATO DE UM CONJUNTO EM
PYTHON
                sets.append([set_name, f"in
set([{','.join(set_definition.strip('{}').split(','))}])"])
    #2      VERIFICA SE A DEFINICAO CONTEM OPERADORES RELACIONAIS (EX: >,
<, >=, <=)
            elif re.search(r'>|<|>=|<=', set_definition):
    #2          MANTEM A DEFINICAO ORIGINAL PARA OPERADORES RELACIONAIS
                sets.append([set_name, set_definition])
    #2      VERIFICA SE A DEFINICAO CONTEM UM OPERADOR DE IGUALDADE
SIMPLES (EX: =)
            elif re.search(r'=', set_definition):
    #2          FORMATA A DEFINICAO PARA UMA COMPARACAO DE IGUALDADE EM
PYTHON (==)
                sets.append([set_name, f"==
{set_definition.strip('=')}"])
    #2      VERIFICA SE A DEFINICAO E UM INTERVALO DE VALORES (EX: 1..10)
            elif re.search(r'\.\.', set_definition):
    #2          FORMATA A DEFINICAO PARA UM INTERVALO USANDO RANGE EM
PYTHON
                sets.append([set_name, f"in
range({set_definition.split('..')[0]},{set_definition.split('..')[1]})"])
    #2  RETORNA A LISTA DE CONJUNTOS TRATADOS
        return sets
    #2]
```

Nome do Arquivo
code_inserter.py
Documentação

```
#1[
#1 TITULO: CODE_INSERTER.PY
#1 AUTOR: EDUARDO RIBEIRO SILVA DE OLIVEIRA
#1 DATA: 25/08/2024
#1 VERSAO: 1
#1 FINALIDADE: INSERIR CODIGO EM UM ARQUIVO EM POSICOES ESPECIFICAS
#1 ENTRADAS: CAMINHO DO ARQUIVO, NOME DO ARQUIVO, MAPA DE POSICOES DE
CODIGO
#1 SAIDAS: ARQUIVO MODIFICADO COM O CODIGO INSERIDO
#1 ROTINAS CHAMADAS: SET_DEFAULT_SPACING, SET_FILE_PATH, SET_FILE_NAME,
SET_FILE_STRING, SET_NEW_FILE_PATH, SET_CODE_SPACING, INSERT
#1]
import os

class CodeInserter:

    #1[
    #1 ROTINA: __init__
    #1 FINALIDADE: INICIALIZA A CLASSE E DEFINE OS ATRIBUTOS INICIAIS
    #1 ENTRADAS: CAMINHO DO ARQUIVO (STR), NOME DO ARQUIVO (STR)
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: N/A
    #1 CHAMA: SET_DEFAULT_SPACING, SET_FILE_PATH, SET_FILE_NAME,
SET_FILE_STRING, SET_NEW_FILE_PATH
    #1]
```

```
    #2[
    #2 PSEUDOCODIGO DE: __init__
    def __init__(self, file_path: str, file_name: str):
    #2  DEFINE O ESPACAMENTO PADRAO COMO 4 ESPACOS
        self.set_default_spacing(" " * 4)
    #2  DEFINE O CAMINHO DO ARQUIVO
        self.set_file_path(file_path)
    #2  DEFINE O NOME DO ARQUIVO
        self.set_file_name(file_name)
    #2  LE O CONTEUDO DO ARQUIVO E ARMAZENA EM SELF.FILE_STRING
        self.set_file_string()
    #2  DEFINE O NOVO CAMINHO DO ARQUIVO
        self.set_new_file_path()
    #2]

    #1[
    #1 ROTINA: SET_DEFAULT_SPACING
    #1 FINALIDADE: DEFINIR O ESPACAMENTO PADRAO PARA O CODIGO INSERIDO
    #1 ENTRADAS: ESPACAMENTO PADRAO (STR)
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: __init__
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: set_default_spacing
    def set_default_spacing(self, default_spacing: str):
    #2  ARMAZENA O ESPACAMENTO PADRAO
        self.default_spacing = default_spacing
    #2]

    #1[
    #1 ROTINA: SET_FILE_PATH
    #1 FINALIDADE: DEFINIR O CAMINHO DO ARQUIVO A SER MODIFICADO
    #1 ENTRADAS: CAMINHO DO ARQUIVO (STR)
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: __init__
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: set_file_path
    def set_file_path(self, file_path: str):
    #2  ARMAZENA O CAMINHO DO ARQUIVO
        self.file_path = file_path
    #2]

    #1[
    #1 ROTINA: SET_FILE_NAME
    #1 FINALIDADE: DEFINIR O NOME DO ARQUIVO A SER GERADO OU MODIFICADO
    #1 ENTRADAS: NOME DO ARQUIVO (STR)
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: __init__
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: set_file_name
    def set_file_name(self, file_name: str):
    #2  ARMAZENA O NOME DO ARQUIVO
        self.file_name = file_name
    #2]

    #1[
    #1 ROTINA: SET_FILE_STRING
    #1 FINALIDADE: LER O CONTEUDO DO ARQUIVO E ARMAZENAR EM UMA LISTA DE
LINHAS
    #1 ENTRADAS: N/A
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: __init__
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: set_file_string
    def set_file_string(self):
    #2  ABRE O ARQUIVO NO MODO LEITURA
        with open(self.file_path, 'r') as file:
    #2      LE TODAS AS LINHAS DO ARQUIVO
            self.file_string = file.readlines()
    #2]

    #1[
    #1 ROTINA: SET_NEW_FILE_PATH
    #1 FINALIDADE: DEFINIR O NOVO CAMINHO DO ARQUIVO COM O NOME
ATUALIZADO
    #1 ENTRADAS: N/A
```

```
    #1 DEPENDENCIAS: OS, PATH
    #1 CHAMADO POR: __init__
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: set_new_file_path
    def set_new_file_path(self):
    #2  CONSTROI O NOVO CAMINHO DO ARQUIVO
        self.new_file_path =
os.path.join(os.path.dirname(self.file_path), self.file_name)
    #2]

    #1[
    #1 ROTINA: SET_CODE_SPACING
    #1 FINALIDADE: AJUSTAR O ESPACAMENTO DO CODIGO INSERIDO COM BASE NA
COLUNA INICIAL
    #1 ENTRADAS: CODIGO A SER INSERIDO (STR), COLUNA INICIAL (INT)
    #1 DEPENDENCIAS: N/A
    #1 CHAMADO POR: INSERT
    #1 CHAMA: N/A
    #1]
    #2[
    #2 PSEUDOCODIGO DE: set_code_spacing
    def set_code_spacing(self, code, column):
    #2  SUBSTITUI O ESPACAMENTO INICIAL PELO NUMERO DE ESPACOS
CORRESPONDENTES A COLUNA
        code = code.replace('', ' ' * column)
    #2  SUBSTITUI O ESPACAMENTO PADRAO NO CODIGO
        code = code.replace('', self.default_spacing)
    #2  RETORNA O CODIGO COM ESPACAMENTO AJUSTADO
        return code
    #2]

    #1[
    #1 ROTINA: INSERT
    #1 FINALIDADE: INSERIR O CODIGO GERADO NAS POSICOES ESPECIFICAS DO
ARQUIVO
    #1 ENTRADAS: MAPA DE POSICOES DE CODIGO (DICT)
    #1 DEPENDENCIAS: OS, SET_CODE_SPACING, SET_NEW_FILE_PATH
    #1 CHAMADO POR: EXTERNO
    #1 CHAMA: SET_CODE_SPACING
    #1]
    #2[
    #2 PSEUDOCODIGO DE: insert
    def insert(self, dt_position_to_code_map: dict):
    #2  PERCORRE OS ELEMENTOS DO CONJUNTO DE POSICOES E CODIGO GERADO
        for (start, end), generated_code in
dt_position_to_code_map.items():
    #2      DESESTRUTURA A LINHA E COLUNA DE INICIO
            start_line, start_column = start
    #2      DESESTRUTURA A LINHA E COLUNA DE FIM
            end_line, end_column = end

    #2      REMOVE O CONTEUDO ENTRE AS LINHAS DE INICIO E FIM
            for row in range(start_line, end_line):
                self.file_string[row] = ''

    #2      INSERE O CODIGO GERADO NA POSICAO CORRETA
            new_line = self.set_code_spacing(generated_code, start[1])
            self.file_string[start_line] = new_line

    #2  SALVA O CONTEUDO MODIFICADO NO NOVO ARQUIVO
        with open(self.new_file_path, 'w') as file:
            file.writelines(self.file_string)
    #2]
```