FASTuml - Documentazione Tecnica

Arici Andrea, Marchesi Gabriele, Tironi Cristian

2023/2024

Sommario

Funzionamento

FASTuml semplifica la scrittura della documentazione permettendo, scrivendo una sola volta il codice, di generare i diagrammi uml delle proprie classi e allo stesso tempo generare i file (java o python) con classi metodi e attributi pronti per essere implementati.

Il funzionamento è immediato:

1. Scaricare l'eseguibile codeGeneration.jar
2. Aprire il terminale e posizionarsi dentro nella stessa cartella in cui è stato scaricato codeGeneration.jar
3. Eseguire il comando:
   java -jar codeGeneration.jar --input-file path/to/input.file

Il software FASTuml viene distribuito formato eseguibile jar.

Controlli Semantic Handler

Segue una lista, divisa per categorie, dei controlli eseguiti dal semantic handler.

Controlli e Gestione delle Classi

Dichiarazione delle Classi

- Metodo: manageClassName(Token className)
- Controllo:
    - Verifica se una classe è già stata dichiarata (isClassDeclared).
    - Se duplicata, genera l'errore ALREADY_DEF_ERROR.
    - Se valida, aggiunge la classe alla classTable.

Inizializzazione di una Classe

- Metodo: setClass(Token className)
- Inizializza il contesto semantico per la classe corrente:
    - Pulisce le tabelle degli attributi, delle operazioni e dei costruttori.
    - Aggiorna le relazioni associate alla classe.

Controllo del Tipo

- Metodo: isType(String type)
- Verifica se un tipo è primitivo (es. int, float, string) o una classe definita.

Controlli e Gestione degli Attributi

Dichiarazione degli Attributi

- Metodo: attDeclaration(String visibility, String arrayType, String type, Token attName, Token defaultValue)
- Controlli:
    1. **Dichiarazione duplicata**: Controlla se l'attributo è già dichiarato nella classe corrente (isAttDeclared).
        - In caso affermativo, genera l'errore ALREADY_DEF_ERROR.
    2. **Validità del valore di default**: Se un valore di default è fornito, verifica che sia compatibile con il tipo (isDefaultValueCorrect).
        - In caso di incompatibilità, genera l'errore INCORRECT_VALUE.
    3. **Aggiornamento delle relazioni**: Se il tipo è una classe, aggiorna la relazione nella tabella classRelTable.

Controlli e Gestione delle Operazioni

Dichiarazione delle Operazioni

- Metodo: opDeclaration(String visibility, String returnType, Token opName, List<TypeRuleContext> paramsType, List<Token> paramsName)
- Controlli:
    1. **Costruttore non valido**: Verifica se il nome dell'operazione coincide con il nome della classe, segnalando un errore (INVALID_CONSTRUCTOR_IN_OP_ERROR).
    2. **Parametri duplicati**: Controlla che i nomi dei parametri siano univoci, generando l'errore ALREADY_DEF_ERROR per eventuali duplicati.
    3. **Operazione duplicata**: Costruisce una chiave univoca (getOpKey) per identificare l'operazione e verifica che non sia già dichiarata.
        - In caso contrario, genera l'errore ALREADY_DEF_OP_ERROR.
    4. **Aggiornamento delle relazioni**: Se i tipi dei parametri o il tipo di ritorno sono classi, aggiorna le relazioni.

Controlli e Gestione dei Costruttori

Dichiarazione dei Costruttori

- Metodo: constrDeclaration(Token opName, List<TypeRuleContext> paramsType, List<Token> paramsName)
- Controlli:
    1. **Nome del costruttore**: Verifica che coincida con il nome della classe corrente.
        - In caso contrario, genera l'errore INVALID_CONSTRUCTOR_ERROR.
    2. **Parametri duplicati**: Identifica eventuali duplicati nei nomi dei parametri.
    3. **Costruttore duplicato**: Controlla l'unicità del costruttore mediante una chiave univoca (getConstrKey).

Controlli e Gestione delle Relazioni

Dichiarazione delle Relazioni

- Metodo: relDeclaration(Token nameClass1, String relationType, Token nameClass2)
- Controlli:
    1. **Dichiarazione delle classi**: Verifica che entrambe le classi coinvolte siano dichiarate.
        - In caso contrario, genera l'errore NO_DECLARATION_ERROR.

2. **Relazione duplicata**: Controlla che la relazione non sia già dichiarata, generando l'errore ALREADY_DEF_REL_ERROR se duplicata.

Coerenza delle Relazioni

- Metodo: relationsCoherent()
- Verifica che tutte le classi utilizzate come tipi di attributi o ritorni di operazioni abbiano una relazione valida definita nel blocco delle relazioni.
    - Genera errori semantici per le classi mancanti.

Controllo degli Enums

Dichiarazione delle Enums

- Metodo: manageEnum(Token enumName)
- Controlli:
    1. **Nome duplicato**: Verifica che l'enum non abbia lo stesso nome di una classe o di un'altra enum.
        - In caso contrario, genera l'errore ALREADY_DEF_ERROR.
    2. **Aggiunta ai contesti semantici**: L'enum viene aggiunta sia alla classTable che alla enumTable.

Dichiarazione dei Valori Enum

- Metodo: enumDeclaration(List<Token> tEnums)
- Controlli:
    - Identifica duplicati nei valori dell'enum, generando l'errore ALREADY_DEF_ERROR per eventuali conflitti.

Controllo della Molteplicità

- Metodo: manageMultiplicity(Token min, Token max)
- Verifica che la molteplicità minima non ecceda quella massima. In caso di violazione, genera l'errore MULTIPLICITY_ERROR.

Tecnologie

Per lo sviluppo di FASTuml sono state utilizzate le seguenti tecnologie:

- **Java** + Eclipse IDE
- **ANTLR 4.13.2** (una delle versioni più recenti di ANTLR)
- **Apache Maven**: framework per la gestione del progetto Java e la generazione dell'eseguibile in formato Jar.
- **JGraphX** per creare e configurare diagrammi graficamente.
- **Swing e AWT** per la visualizzazione opzionale del diagramma in una finestra.
- **ImageIO** per esportare il diagramma come immagine PNG.

Struttura

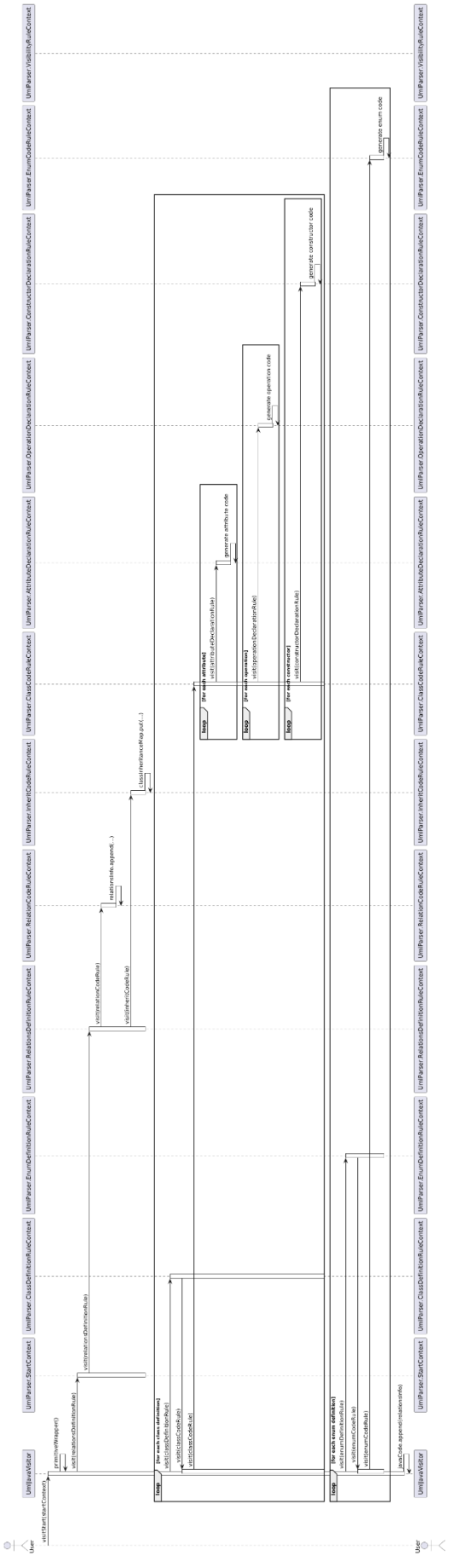
L'intera struttura si basa sulla sintassi contenuta del file "Uml.g4" e sui controlli presenti nel file "SemanticHandler".

Tramite il comando "java -jar antlr-4.13.2-complete.jar Uml.g4 -visitor" sono stati generati i file contenenti il Lexer ed il Parser. Inoltre, grazie alla versione 4+ di ANTLR, è possibile generare i file "UmlBaseVisitor" ed "UmlBaseListener" che permettono di visitare l'albero sintattico del linguaggio e di eseguire operazioni su di esso. Tramite il visitor sono state implementate le funzioni di generazione degli scheletri nel linguaggio Java e Python ed anche la generazione visiva del diagramma delle classi.

Lista dei package:

- **Compiler Package:** contiene tutti i componenti del compilatore ed i visitors usati per la generazione del codice
- **Converter Package:** Contiene il visitor per la generazione dei png con i class diagrams
- **Uml Package:** Qua è presente il file contenente il main
- **Resources Package:** Contiene il file di input per quando si esegue il programma
- **Generated Package:** Contiene i file con il codice generato in java e python.

Descrizione dei metodi del file UmlJavaVisitor:



visitStart

- **Descrizione**: Inizializza la generazione del codice Java per un modello UML completo. Visita classi, enum e relazioni, aggregando il codice generato e aggiungendo informazioni sulle relazioni.

primitiveWrapper

- **Descrizione**: Popola la mappa primitiveToWrapper con le associazioni tra tipi primitivi Java e i loro wrapper corrispondenti.

visitClassDefinitionRule

- **Descrizione**: Genera la dichiarazione e il corpo di una classe Java, aggiungendo il modificatore abstract e la clausola extends in caso di ereditarietà.

visitClassCodeRule

- **Descrizione**: Elabora gli attributi, i metodi e i costruttori definiti all'interno di una classe Java.

visitEnumDefinitionRule

- **Descrizione**: Genera la dichiarazione di un enum Java e ne popola i valori.

visitRelationsDefinitionRule

- **Descrizione**: Analizza le relazioni tra le classi definite nel modello UML, aggiornando le informazioni interne senza produrre direttamente codice.

visitRelationCodeRule

- **Descrizione**: Gestisce relazioni specifiche tra classi (inherits, shared, composed) aggiornando la mappa di ereditarietà o le informazioni sulle relazioni.

visitAttributeDeclarationRule

- **Descrizione**: Genera la dichiarazione di un attributo Java, specificando visibilità, tipo e nome, e ne gestisce l'inizializzazione in base al contesto.

visitOperationDeclarationRule

- **Descrizione**: Genera la dichiarazione di un metodo Java, includendo visibilità, tipo di ritorno, parametri e un corpo base per l'implementazione.

visitConstructorDeclarationRule

- **Descrizione**: Genera un costruttore Java per una classe, con i relativi parametri e una struttura di implementazione iniziale.

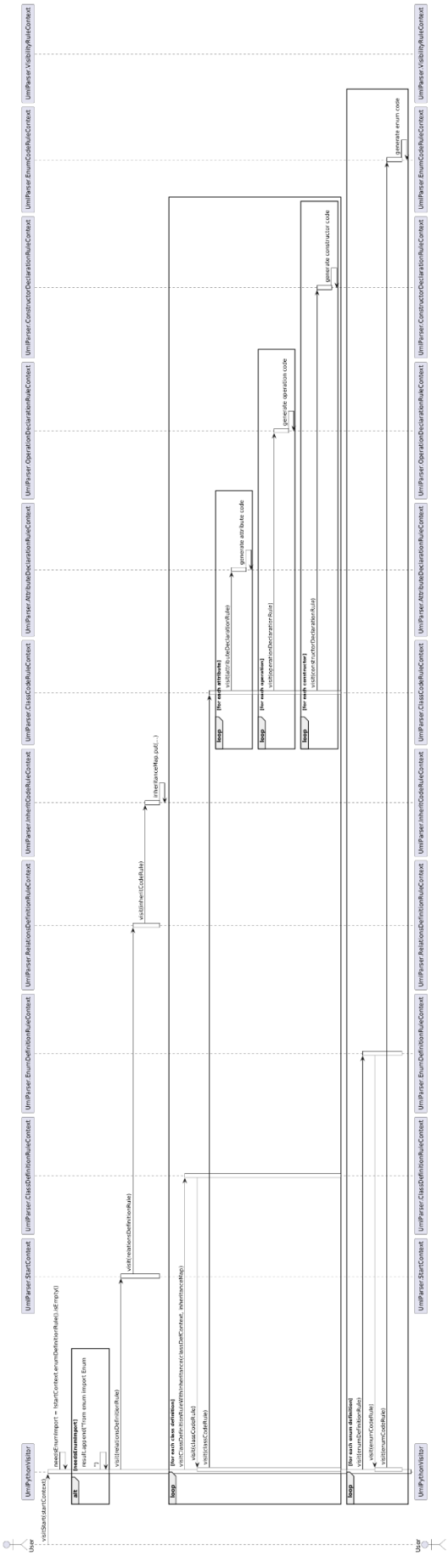
visitEnumCodeRule

- **Descrizione**: Compone e restituisce i valori di un enum come elenco separato da virgole.

visitVisibilityRule

- **Descrizione**: Traduce i modificatori di visibilità UML (public, protected, ecc.) nei corrispondenti modificatori di visibilità Java.

Descrizione dei metodi del file UmlPythonVisitor:



visitStart

- **Descrizione**: Inizia la generazione del codice Python da un modello UML, gestendo importazioni, ereditarietà e relazioni tra classi, oltre alla definizione di enum e classi.

visitClassDefinitionRuleWithInheritance

- **Descrizione**: Genera il codice Python per una classe, includendo l'ereditarietà basata sulle relazioni "inherits". Aggiunge il supporto per classi astratte se necessario.

visitEnumDefinitionRule

- **Descrizione**: Genera la definizione di un enum in Python, includendo i suoi valori come attributi.

visitClassDefinitionRule

- **Descrizione**: Genera la definizione di una classe Python, identificando le classi astratte e aggiungendo il loro corpo.

visitClassCodeRule

- **Descrizione**: Genera il corpo di una classe Python, includendo attributi, metodi e costruttori. Aggiunge il segnaposto pass se il corpo è vuoto.

visitAttributeDeclarationRule

- **Descrizione**: Crea la dichiarazione di un attributo in Python, specificando visibilità, tipo e un valore predefinito None.

visitOperationDeclarationRule

- **Descrizione**: Genera la definizione di un metodo in Python, includendo visibilità, tipo di ritorno, parametri e un corpo predefinito con pass.

visitConstructorDeclarationRule

- **Descrizione**: Genera la definizione di un costruttore Python (__init__), includendo i parametri e un corpo predefinito con pass.

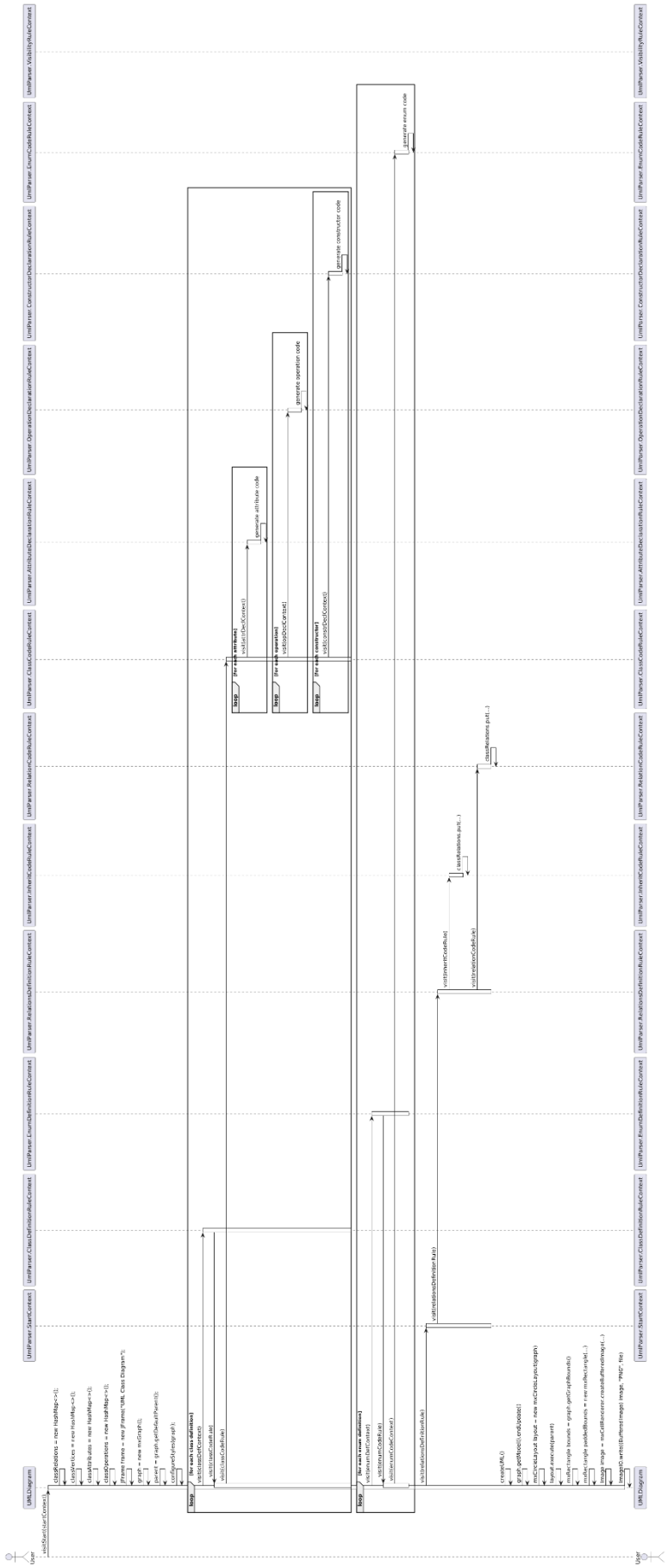
visitVisibilityRule

- **Descrizione**: Converte la visibilità UML (public, private, protected) nei prefissi Python corrispondenti (` `, _, __).

visitRelationsDefinitionRule

- **Descrizione**: Genera una rappresentazione testuale delle relazioni UML nel modello, principalmente come commenti Python.

Descrizione dei metodi del file UMLDiagram:



visitStart(UmlParser.StartContext ctx)

- **Descrizione:** Punto di ingresso per la visita dell'albero di parsing. Configura il grafico UML, visita le definizioni di classi, enum e relazioni, e genera l'immagine del diagramma UML.
- **Output:** Salva l'immagine del diagramma in formato PNG.

visitClassDefinitionRule(UmlParser.ClassDefinitionRuleContext ctx)

- **Descrizione:** Gestisce la definizione di una classe UML. Inserisce un vertice per rappresentare la classe e visita il corpo della classe per elaborare attributi e operazioni.
- **Output:** Crea un nodo per la classe e memorizza informazioni sugli attributi e operazioni.

visitClassCodeRule(UmlParser.ClassCodeRuleContext ctx)

- **Descrizione:** Elabora il corpo della classe, visitando gli attributi e le operazioni definiti all'interno.
- **Output:** Popola le strutture dati con informazioni sugli attributi e operazioni della classe corrente.

visitAttributeDeclarationRule(UmlParser.AttributeDeclarationRuleContext ctx)

- **Descrizione:** Processa una dichiarazione di attributo, includendo visibilità, tipo e valore di default.
- **Output:** Aggiunge l'attributo alla lista degli attributi della classe corrente.

visitOperationDeclarationRule(UmlParser.OperationDeclarationRuleContext ctx)

- **Descrizione:** Elabora una dichiarazione di metodo, includendo visibilità, nome, parametri e tipo di ritorno.
- **Output:** Aggiunge il metodo alla lista delle operazioni della classe corrente.**visitConstructorDeclarationRule(UmlParser.ConstructorDeclarationRuleContext ctx)**
- **Descrizione:** Gestisce la definizione di un costruttore, elaborando i parametri del metodo.
- **Output:** Aggiunge il costruttore alla lista delle operazioni della classe corrente.

visitRelationsDefinitionRule(UmlParser.RelationsDefinitionRuleContext ctx)

- **Descrizione:** Visita la sezione del parser dedicata alla definizione delle relazioni tra le classi.
- **Output:** Memorizza le informazioni relative alle relazioni.

visitRelationCodeRule(UmlParser.RelationCodeRuleContext ctx)

- **Descrizione:** Processa una relazione tra due classi, analizzandone i tipi (associazione, ereditarietà, composizione) e le molteplicità.
- **Output:** Aggiunge la relazione alla lista delle relazioni.

visitEnumDefinitionRule(UmlParser.EnumDefinitionRuleContext ctx)

- **Descrizione:** Gestisce la definizione di un tipo enum, creando un nodo per rappresentare l'enumerazione e memorizzandone i valori.

- **Output:** Crea un vertice per l'enumerazione e aggiunge i valori associati.

visitEnumCodeRule(UmlParser.EnumCodeRuleContext ctx)

- **Descrizione:** Processa i valori definiti in un enum.
- **Output:** Memorizza i valori nella lista degli attributi dell'enumerazione corrente.

visitVisibilityRule(UmlParser.VisibilityRuleContext ctx)

- **Descrizione:** Determina la visibilità (public, private, protected o package-private) di un attributo o metodo.
- **Output:** Restituisce il simbolo di visibilità corrispondente.

createUML()

- **Descrizione:** Costruisce il diagramma UML finale. Inserisce gli attributi e le operazioni nei nodi delle classi e disegna le relazioni tra di essi.
- **Output:** Configura graficamente gli elementi UML nel diagramma.

configureStyles(mxGraph graph)

- **Descrizione:** Configura gli stili grafici per classi, attributi, enum, frecce di relazione, ecc.
- **Output:** Applica gli stili al grafo mxGraph.

Il seguente diagramma delle classi mostra i metodi, attributi e le relazioni tra tutte le classi del progetto.

Specifica Antlr

Di seguito è riportato l'intero codice del file .g4 contente la sintassi del linguaggio FASTuml e anche le chiamate alle funzioni del semantic handler.

```
grammar Uml;

@header {
   package compiler_package;
}

@members {
   SemanticHandler h = new SemanticHandler();

   public SemanticHandler getHandler() {
      return h;
   }
}
```

```
/* ***********************************************
        Syntactic Rule definition starts here
********************************************** */

start
   : classDefinitionRule* enumDefinitionRule*? relationsDefinitionRule?
   ;

classDefinitionRule
   : ABSTRACT? CLASS c=ID { h.manageClassName($c); h.setClass($c); }
     classCodeRule
   ;

enumDefinitionRule
   : ENUM n=ID enumCodeRule { h.manageEnum($n); h.setEnum($n); }
   ;

relationsDefinitionRule
   : RELATIONS LBR relationCodeRule* { h.relationsCoherent(); } RBR
   ;

classCodeRule
   : LBR (
                  (CONSTRUCTOR LBR constructorDeclarationRule* RBR)?
                  (ATTRIBUTE LBR attributeDeclarationRule* RBR)?
        (OPERATION LBR operationDeclarationRule* RBR)? ) RBR
   ;

enumCodeRule
   : LBR (eName+=ID SC)* RBR { h.enumDeclaration($eName); }
   ;
```

```
relationCodeRule
   : nameClass1=ID multiplicityRule relationTypeRule
     nameClass2=ID multiplicityRule SC
    {
      h.relDeclaration($nameClass1, $relationTypeRule.text, $nameClass2);
    }
   ;

attributeDeclarationRule
   : v=visibilityRule ar=arrayTypeRule? t=typeRule a=ID (EQ  d=(STRING | INT | FLOAT))? READONLY? SC
    { h.attDeclaration($v.text, $ar.text != null ? $ar.text : null, $t.text, $a, $d != null ? $d : null); }
   ;

visibilityRule
   : ( PUBLIC | PROTECTED | PRIVATE | PACKAGE )
   ;

arrayTypeRule
   : ( SET | MULTISET | LIST | ORDEREDSET )
   ;

typeRule
   : ( INT_TYPE | FLOAT_TYPE | LONG_TYPE | DOUBLE_TYPE | BOOLEAN_TYPE | CHAR_TYPE |
STRING_TYPE | VOID_TYPE | ID )
   ;

relationTypeRule
   : UNDREL | SXREL | DXREL | INHERITS | SHARED | COMPOSED
   ;

multiplicityRule
   : (n=INT COMMA m=INT)
```

```
  ;

operationDeclarationRule
  : v=visibilityRule t=typeRule? a=ID LP (pType+=typeRule pName+=ID (COMMA pType+=typeRule pName+=ID)*)? RP SC
    {h.opDeclaration($v.text, $t.text != null ? $t.text : null, $a, $pType, $pName);
    }
  ;

constructorDeclarationRule
  : a=ID LP (pType+=typeRule pName+=ID (COMMA pType+=typeRule pName+=ID)*)? RP SC
    { h.constrDeclaration($a, $pType, $pName); }
  ;
/* ***********************************************
      Tokens definition part starts here
*********************************************** */

EQ : '=';
COMP : '==';
NEQ : '!=';
SXREL : '<';
DXREL : '>';
LTE : '<=';
GTE : '>=';
MOD : '%';
ADD : '+';
UNDREL : '-';
MUL : '*';
DIV : '/';
AADD : '++';
SSUB : '--';
```

DP : ':';

SC : ';';

DOT : '.';

COMMA : ',';

LP : '(';

RP : ')';

LBR : '{';

RBR : '}';

LB : '[';

RB : ']';

ABSTRACT : 'abstract';

BOOLEAN_TYPE : 'boolean';

BYTE : 'byte';

CHAR_TYPE : 'char';

CLASS : 'class';

CONSTRUCTOR: 'constructor';

CONST : 'const';

DOUBLE_TYPE : 'double';

ENUM : 'enum';

EXTENDS : 'extends';

FALSE : 'false';

FINAL : 'final';

FLOAT_TYPE : 'float';

IMPLEMENTS : 'implements';

INHERITS : 'inherits';

INT_TYPE : 'int';

INTERFACE : 'interface';

LONG_TYPE : 'long';

NONUNIQUE : 'non-unique';

NULL : 'null';

ORDER : 'ordered';

```
PRIVATE : 'private';

PROTECTED : 'protected';

PUBLIC : 'public';

PACKAGE : 'package';

READONLY : 'readOnly';

SET : 'Set';

MULTISET : 'Multi-set';

ORDEREDSET : 'Ordered-set';

LIST : 'List';

SHORT : 'short';

STATIC : 'static';

THROWS : 'throws';

STRING_TYPE : 'String';

TRUE : 'true';

UNIQUE : 'unique';

UNORDERED : 'unordered';

VOID_TYPE : 'void';

SHARED : 'shared';

COMPOSED : 'composed';

ATTRIBUTE : 'attribute';

RELATIONS : 'relations';

OPERATION : 'operation';

MIN : 'min';

MAX : 'max';


ID : [a-zA-Z_] [a-zA-Z0-9_]*;

INT : [0-9]+;

FLOAT : [0-9]+ '.' [0-9]* EXPONENT? | '.' [0-9]+ EXPONENT? | [0-9]+ EXPONENT;


COMMENT
   :   ( '//' ~[\r\n]* '\r'? '\n'
       | '/*' .*? '*/'
```

```
      ) -> channel(HIDDEN)
   ;
WS : [ \t\r\n]+ -> skip;

STRING : '"' ( ESC_SEQ | ~('\\'|'"') )* '"';
CHAR : '\'' ( ESC_SEQ | ~('\''|'\\') ) '\'';

fragment EXPONENT : [eE] [+-]? [0-9]+;
fragment HEX_DIGIT : [0-9a-fA-F];
fragment ESC_SEQ
   : '\\' [btnfr"'\\]
   | UNICODE_ESC
   | OCTAL_ESC
   ;

fragment OCTAL_ESC
   : '\\' [0-3] [0-7] [0-7]
   | '\\' [0-7] [0-7]?
   ;

fragment UNICODE_ESC
   : '\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
   ;



ERROR_TOKEN
          :
                    . // {     $channel = HIDDEN;    }
          ;
```