FASTuml - Documentazione Utente

Arici Andrea, Marchesi Gabriele, Tironi Cristian  
2023/2024

Sommario

[Installazione Programma 2](#_Toc183016824)

[Esecuzione Programma 2](#_Toc183016825)

[Documentazione della Sintassi 3](#_Toc183016826)

[Creazione delle Classi 3](#_Toc183016827)

[Creazione degli Attributi 3](#_Toc183016828)

[Creazione dei Metodi 4](#_Toc183016829)

[Creazione delle Enumerazioni 5](#_Toc183016830)

[Creazione delle Relazioni 5](#_Toc183016831)

# Introduzione

FASTuml è un programma in grado di generare un diagramma delle classi ed una relativa struttura di partenza del codice implementativo utilizzando una sola sintassi. Questo evita la duplicazione nella scrittura del codice, che può portare a perdite di tempo, errori o mancata creazione di documentazione. FASTuml è in grado di generale del codice di partenza nei linguaggi java e python. Queste strutture iniziali saranno perfettamente identiche al diagramma delle classi, anch’esso generato da FASTuml sotto forma di .png.

Il programma necessita una singola riga di comando nel prompt per il funzionamento, attraverso cui si possono gestire diverse impostazioni del funzionamento

FASTuml non richiede l’uso di internet, solamente

# Installazione Programma

Requisiti:

1. Java SDK 18
2. Java deve essere presente nel path delle variabili d’ambiente (per poter eseguire i comandi sul terminale)

Scaricare l’eseguibile codeGeneration.jar e posizionarlo dentro la cartella in cui lo si vuole utilizzare.

# Esecuzione Programma

Per eseguire il programma:

1. Aprire il terminale e posizionarsi dentro nella stessa cartella in cui è stato scaricato codeGeneration.jar
2. Eseguire il comando:

**java -jar codeGeneration.jar --input-file path/to/input.file**

Di default verrà generato, nel path in cui ci si trova al momento dell’esecuzione del comando, solo l’immagine .png relativa al class diagram.

Sono inoltre possibili le seguenti opzioni:

**Genera solo il file Java**:

java -jar codeGeneration.jar --input-file path/to/input.file --language java

**Genera solo il file Python**:

java -jar codeGeneration.jar --input-file path/to/input.file --language python

**Genera sia file Java che python**:

java -jar codeGeneration.jar --input-file path/to/input.file --language both

**Specifica la cartella di output (solo per i file Java e Python)**:

java -jar codeGeneration.jar --input-file path/to/input.file --output-folder path/to/output

**Nomi file personalizzati (solo per i file Java e Python)**:

java -jar codeGeneration.jar --input-file path/to/input.file --java-file MyJavaFile.java --python-file MyPythonFile.py

# Documentazione della Sintassi

## Creazione delle Classi

* **ABSTRACT**: Facoltativo, se presente indica che la classe è astratta e non può essere istanziata.
* **CLASS**: Parola chiave che definisce una classe.
* **ID**: Nome della classe

**Esempio:**

abstract class Car {

}

## Creazione degli Attributi

* **Visibilità**: La visibilità dell'attributo può essere una delle seguenti:
  + public
  + private
  + protected
  + package
* **Tipo**: Il tipo dell'attributo può essere un tipo primitivo (es. int, float, boolean) o un tipo complesso definito precedentemente.
* **Nome**: Il nome dell'attributo.
* **Valore iniziale**: Facoltativo, l'attributo può essere inizializzato con un valore che deve essere coerente al suo Tipo.
* **READONLY**: Facoltativo, indica che l'attributo è solo in lettura.

**Esempio:**

class Car {

private String model = "Tesla";

public int speed;

readOnly int year = 2020;

}

## Creazione dei Metodi

* **Visibilità**: La visibilità del metodo (public, private, protected, package).
* **Tipo di ritorno**: Tipo di ritorno del metodo (può essere un tipo di dato o void per metodi che non restituiscono nulla).
* **Nome**: Nome del metodo
* **Parametri**: Parametri del metodo, ciascuno con un tipo e un nome. È possibile dichiarare più parametri separati da virgola.

**Esempio:**

class Car {

public void startEngine() {

}

private int calculateFuelEfficiency(int distance, int fuel) {

return distance / fuel;

}

}

## Creazione delle Enumerazioni

**Sintassi:**

* **ENUM**: Parola chiave per definire un'enumerazione.
* **Nome**: Il nome dell'enumerazione.
* **Valori Enum**: Definisce i valori che appartengono all'enumerazione.

**Esempio:**

enum Days {

Monday;

Tuesday;

Wednesday;

Thursday;

Friday;

}

## Creazione delle Relazioni

**Sintassi:**

* **nameClass1, nameClass2**: I nomi delle classi coinvolte nella relazione.
* **tipoRelazione**: Il tipo di relazione (es. Inherits, Shared, Composed).
* **molteplicità**: La molteplicità della relazione, che specifica il numero di istanze della classe coinvolta nella relazione.

**Tipi di Relazione (relationTypeRule):**

* **UNDREL**: Relazione non definita.
* **SXREL**: Relazione sinistra.
* **DXREL**: Relazione destra.
* **INHERITS**: Ereditarietà.
* **SHARED**: Condivisione.
* **COMPOSED**: Composizione.

**Esempio:**

relations {

ClasseB 1,1 inherits ClasseA 1,3;

}

# Errori e Warning

In caso di errori di sintassi o di semantica verranno evidenziati tramite un messaggio sul prompt al momento dell’esecuzione del comando iniziale. Il messaggio contiene una breve spiegazione dell’errore e anche la posizione di riga e colonna del file di input in cui si è verificato.

In caso di errore di sintassi il programma indicherà quale parola non è stata riconosciuta e cosa invece si sarebbe aspettato di trovare.

In caso di errore, non verrà generata nessuna visualizzazione e nessuna struttura codice.