– Université de Carthage –
Institut National des Sciences Appliquées et de Technologie

PROJET

PROGRAMMATION ORIENTÉE OBJET

INTERPRÉTEUR D'ALGORITHME

Realisé Par:

Ayoub Ben Yedder

Enseignant: prof. Amira Dhkil

Année Universitaire: 2024–2025

1 Problématique

Les élèves au lycée étudiant la matière "Algorithme et Programmation" écrivent très souvent des solutions aux problèmes en syntaxe algorithmique. Cependant, ils ne peuvent tester leurs algorithmes qu'après les avoir traduits dans un autre langage de programmation (Pascal, Python, C, etc.). Cette traduction peut introduire des erreurs si elle n'est pas réalisée correctement.

Pour résoudre ce problème, j'ai proposé ce projet, qui permettra aux étudiants d'exécuter directement leurs algorithmes sans avoir besoin de les traduire.

2 Description du projet

2.1 Fonctionnement

L'interpréteur d'algorithme est un programme qui prend en entrée un code source écrit en syntaxe algorithmique et le traduit en un code exécutable. Le programme est composé de deux parties principales : l'analyse lexicale et l'analyse syntaxique. L'analyse lexicale consiste à analyser le code source et à le découper en tokens. L'analyse syntaxique consiste à vérifier la syntaxe du code source et à l'exécuter. Le programme prend en entrée un fichier contenant le code source et affiche le résultat de sa exécution.

2.2 Le syntaxe implémentée

- **les mots clés** : Algorithme, TDO, Debut, Fin, ecrire, si, sinon, finsi, tantque, fintantque.
- **les opérateurs** : $<-, =, !=, <, >, <=, >=, +, -, *, /$, mod, div, et, ou, non.
- **les types de données** : entier, reel, chaine, bool, char.

2.3 Exemple de syntaxe algorithmique

```
Algorithme Premier
TDO
x : entier;
d : entier;

Debut
    x <- 5;
    d <- 2;
    tantque x mod d != 0 et d < x div 2 faire
        d <- d + 1;
    fintantque
    si x mod d != 0 ou x = 2 alors
        ecrire("le nombre",x,"est premier!");
    sinon
        ecrire("le nombre",x,"n'est pas premier!");
    finsi
Fin
```

2.4 Description des classes

- **TokenType** : Énumération des types de tokens.
- **Token** : Représente un élément du code source (un token).
- **Lexer** : Gère l'analyse lexicale du code source.
- **Interpreter** : Effectue l'analyse syntaxique et l'interprétation du code source.
- **Variable** : Représente une variable.
- **Algo** : Classe principale qui contient la fonction `main`.
- **Logger** : Interface pour les logs et les erreurs.
- **ErrorThrower** : Classe abstraite pour la gestion des erreurs et des logs.

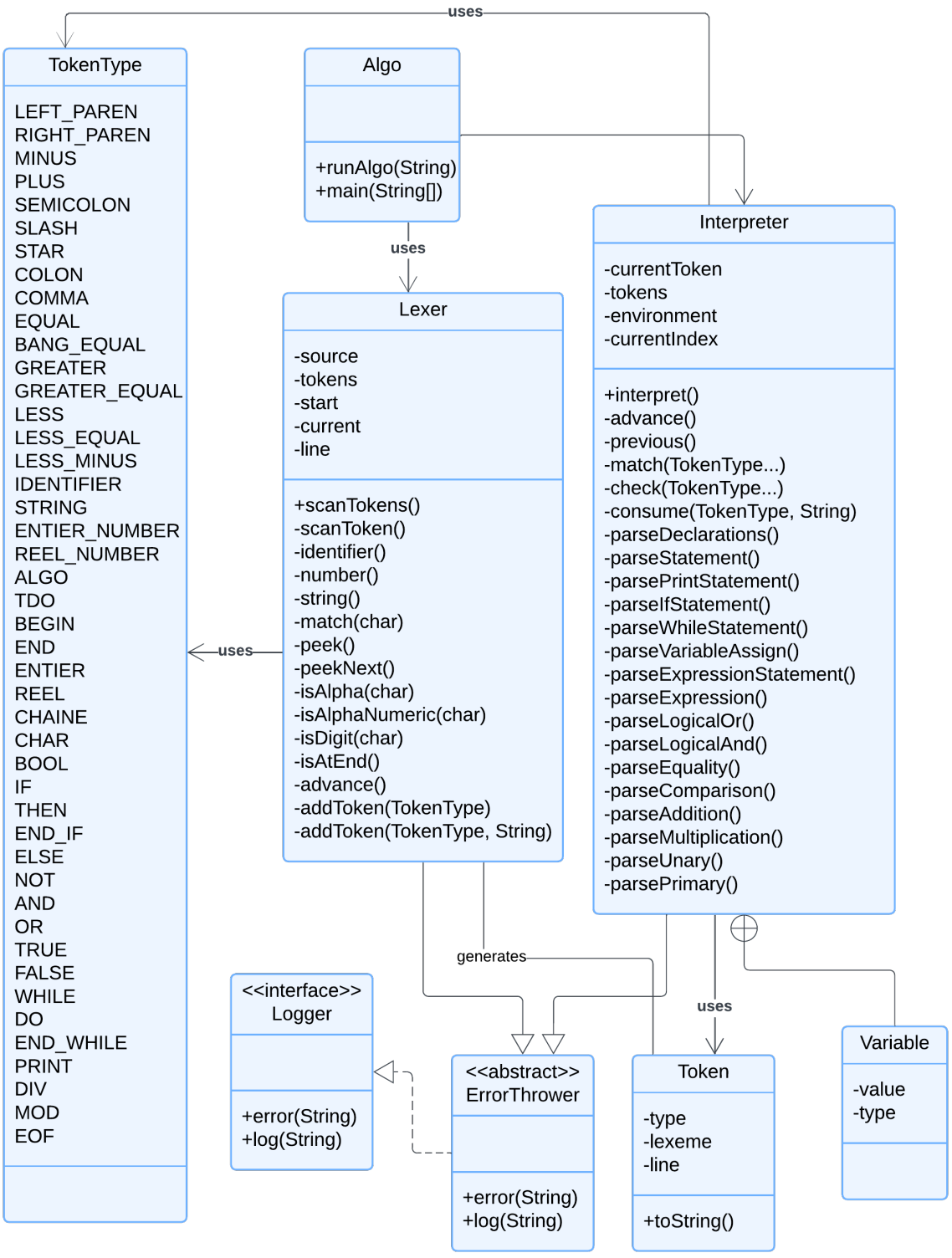

Figure 1: Diagramme de classe UML