**Université A. Mira de Béjaia**
Département d'Informatique - L3 Académique
Module : Compilation

# Rapport de Projet TP

## Réalisation d'un Mini-Compilateur PHP

**Réalisé par :**
SALMANI Fakhreddine

**Encadré par :**
Mrs Tassoult.N

08 Décembre 2025

# Table des matières

**1 Introduction** **2**

**2 Architecture du Projet** **3**

**3 Analyse Lexicale** **4**
3.1 Rôle . . . . . 4
3.2 Tokens Reconnus . . . . . 4
3.3 Gestion des Spécificités . . . . . 4

**4 Analyse Syntaxique** **5**
4.1 La Grammaire . . . . . 5
4.2 Gestion des Structures Ignorées . . . . . 5
4.3 Gestion des Erreurs . . . . . 5

**5 Tests et Résultats** **6**
5.1 Test Valide (If/Else imbriqué) . . . . . 6
5.2 Test avec Boucle Ignorée (For/While) . . . . . 6
5.3 Test d'Erreur . . . . . 6

**6 Conclusion** **7**

# Chapitre 1

# Introduction

Ce projet a pour objectif la réalisation d'un mini-compilateur pour un sous-ensemble du langage **PHP**. Conformément au cahier des charges, nous avons implémenté un analyseur lexical et un analyseur syntaxique en langage **Java**.

Le langage cible a été attribué par tirage au sort : **PHP**. La structure de contrôle principale analysée est l'instruction `if / else`. Les autres structures comme `while`, `for` ou `switch` sont reconnues lexicalement mais ignorées syntaxiquement, permettant ainsi de se concentrer sur la logique conditionnelle.

# Chapitre 2

# Architecture du Projet

L'application suit une architecture modulaire classique pour un compilateur, structurée en trois fichiers principaux :

- **LexicalePHP.java** : Contient l'analyseur lexical (Scanner). Il transforme le code source brut en une suite de *tokens*.
- **syntaxique.java** : Contient l'analyseur syntaxique (Parser). Il vérifie la structure grammaticale du code selon une méthode de descente récursive.
- **SimpleInterface.java** : Une interface graphique (GUI) réalisée avec Swing pour faciliter l'édition du code et la visualisation des erreurs.

# Chapitre 3

# Analyse Lexicale

## 3.1 Rôle

L'analyseur lexical parcourt le code source caractère par caractère pour générer des unités lexicales appelées `lexèmes` ou `tokens`.

## 3.2 Tokens Reconnus

Notre analyseur identifie les catégories suivantes :
— **Balises PHP :** `<?php`, `?>`.
— **Mots-clés :** `if`, `else`, `while`, `for`, `echo`, etc.
— **Types :** `int`, `double`, `void` (ajoutés pour gérer les déclarations typées).
— **Variables :** Toute chaîne commençant par `$` (ex : `$score`).
— **Nombres :** Entiers et flottants (ex : `10`, `3.14`).
— **Opérateurs :** `+`, `-`, `*`, `/`, `=`, `>`, `<`, etc.
— **Délimiteurs :** `{`, `}`, `(`, `)`, `;`.

## 3.3 Gestion des Spécificités

— **Commentaires :** Les commentaires commençant par `//` sont détectés et ignorés jusqu'à la fin de la ligne, afin de ne pas perturber l'analyse syntaxique.
— **Mots-clés Personnalisés :** Conformément à la consigne, le nom `salmani` et le prenom `fakhreddine` sont reconnus comme des tokens spéciaux `NOM_ETUDIANT`.

# Chapitre 4

# Analyse Syntaxique

## 4.1 La Grammaire

Nous avons défini une grammaire simplifiée inspirée de PHP. L'analyseur utilise une méthode de **descente récursive** ($LL(1)$).

**Règles de production simplifiées :**

```
Z -> S (Axiome)
S -> <?php C ?>
C -> Instruction C | epsilon
Instruction ->
    | Declaration (int $x = 10;)
    | Affectation ($x = 5;)
    | Echo (echo $x;)
    | IfElse
    | BlocIgnore (while, for, switch)
IfElse -> if ( Condition ) { C } ElsePart
ElsePart -> else { C } | epsilon
```

## 4.2 Gestion des Structures Ignorées

Pour les structures `while`, `for` et `switch`, l'analyseur adopte une stratégie robuste : il consomme les tokens sans les analyser strictement, mais utilise des compteurs de parenthèses et d'accolades pour s'assurer de sauter l'intégralité du bloc, même s'il contient des imbrications complexes ou des points-virgules.

## 4.3 Gestion des Erreurs

Le compilateur ne s'arrête pas à la première erreur. Lorsqu'une erreur est détectée (ex : point-virgule manquant), un message explicite est ajouté aux logs (`>> ERREUR (Pos X) : Attendu...`), et l'analyseur tente de continuer la lecture (mode "Panic Recovery" simplifié).

# Chapitre 5

# Tests et Résultats

Le projet est livré sous forme exécutable. Voici des exemples de tests effectués.

## 5.1 Test Valide (If/Else imbriqué)

```php
<?php
    int $note = 15;
    if ($note > 10) {
        echo $note;
    } else {
        $note = 0;
    }
    // Test des noms
    salmani;
?>
```

**Résultat :** [SUCCES] Analyse PHP terminée.

## 5.2 Test avec Boucle Ignorée (For/While)

```php
<?php
    // Ce bloc est syntaxiquement ignoré
    for ($i = 0; $i < 10; $i++) {
        $res = $i + 1;
    }
    echo $res;
?>
```

**Résultat :** [SUCCES] Analyse PHP terminée. (Le parser ne plante pas sur les points-virgules du `for`).

## 5.3 Test d'Erreur

```php
<?php
    $x = 10  // Manque le point-virgule
    if ($x > 5) { echo $x; }
?>
```

**Résultat :** »> ERREUR (Pos 12) : Attendu 'POINTVERGULE', mais trouvé 'IF'.

# Chapitre 6

# Conclusion

Ce projet nous a permis de comprendre en profondeur les mécanismes internes d'un compilateur. Nous avons réussi à implémenter un analyseur lexical capable de tokeniser du code PHP et un analyseur syntaxique capable de valider des structures conditionnelles tout en gérant intelligemment les blocs de code complexes à ignorer. L'ajout d'une interface graphique rend l'outil convivial et prêt à l'emploi.