جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

Université Abderrahmane Mira de Béjaïa
Faculté des Sciences Exactes
Département d'Informatique

Mini-Compilateur PHP

Analyseur Lexical et Syntaxique basé sur try/catch

Réalisé par :

- **Ahfir Sara**

Encadré par :

Mme Nadia TASSOULT

Section/Groupe :
A1

Année universitaire : 2025 / 2026

1 Introduction

Ce rapport présente la réalisation d'un mini-compilateur développé dans le cadre du module **Compilation**. L'objectif du projet est d'implémenter en Java un analyseur lexical et un analyseur syntaxique pour un mini-langage inspiré du langage **PHP**.

Le compilateur a été simplifié afin de se concentrer principalement sur une seule instruction structurée : **l'instruction try/catch**.

Les autres instructions (if, while, for, switch...) sont reconnues lexicalement, mais ne sont pas analysées syntaxiquement comme indiqué dans l'énoncé du projet.

2 Objectifs du projet

Les objectifs principaux sont :

— Comprendre la structure d'un langage et définir sa **grammaire**.
— Implémenter un **analyseur lexical** capable de reconnaître tous les lexèmes PHP nécessaires.
— Implémenter un **analyseur syntaxique** basé sur une descente récursive.
— Valider la syntaxe de l'instruction **try/catch/finally**.
— Gérer les erreurs lexicales et syntaxiques sans interrompre l'exécution.
— Générer un exécutable final prêt à l'emploi.

3 Présentation du langage choisi : PHP

Le langage choisi pour ce projet est **PHP**. Il s'agit d'un langage de script très utilisé pour le développement web et possédant une syntaxe relativement proche du C, Java et JavaScript.

Pour ce mini-projet, seule une partie restreinte du langage PHP a été retenue :

— déclaration et affectation de variables ;
— expressions arithmétiques et logiques ;
— comparaison ;
— incrémentation/décrémentation ;
— structure try/catch (instruction principale) ;
— reconnaissance lexicales des structures : if, while, for,switch...

4 Analyse lexicale

L'analyseur lexical est chargé de découper le programme PHP en **tokens**. Les éléments reconnus sont :

— Mots-clés : try, catch, finally, if, else, while, for, foreach, switch, return.
— Variables PHP : $identifiant
— Types : Exception, Error, mot cles personnalisés.
— Constantes : nombres, chaînes, booléens.
— Opérateurs arithmétiques : +, -, *, /, %.
— Opérateurs logiques : ==, !=, <, >, <=, >=, , ||.
— Symboles : { }, ( ), ;, =.

Les commentaires sont ignorés.

Grammaire du mini-compilateur

$$
\begin{aligned}
Z &\rightarrow S\ FIN \\
S &\rightarrow Instruction\ S \mid \epsilon \\
Signature &\rightarrow\ mot\_cle\_perso\ \text{";"} \\
\\
DeclarationClasse &\rightarrow CLASS\ IDENTIFIANT\ Bloc \\
\\
DeclarationMethode &\rightarrow FUNCTION\ IDENTIFIANT\ \text{"("}\ Parametres\ \text{")"}\ Bloc \\
Parametres &\rightarrow \epsilon \mid VARIABLE^{*} \\
DeclarationVariable &\rightarrow \text{"\$"}\ IDENTIFIANT\ [\text{" = "}\ Expression]\ \text{";"} \\
\\
Affectation &\rightarrow VARIABLE = Expression\ ; \\
&\mid\ VARIABLE{+}{+}\ ; \\
&\mid\ VARIABLE{-}{-}\ ; \\
&\mid\ VARIABLE\ ; \\
Expression &\rightarrow Terme\ Expression' \\
Expression' &\rightarrow \text{" + "}\ Terme\ Expression' \mid \text{" - "}\ Terme\ Expression' \mid \epsilon \\
Terme &\rightarrow Facteur\ Terme' \\
Terme' &\rightarrow \text{" * "}\ Facteur\ Terme' \mid \text{"/"}\ Facteur\ Terme' \mid \epsilon \\
Facteur &\rightarrow NOMBRE \\
&\mid\ VARIABLE \\
&\mid\ IDENTIFIANT \\
&\mid\ IDENTIFIANT\ \text{"("}\ Arguments\ \text{")"} \\
&\mid\ CHAINE \\
&\mid\ \text{"("}\ Expression\ \text{")"} \\
\\
Comparaison &\rightarrow Expression\ OperateurComparaison\ Expression \\
OperateurComparaison &\rightarrow \text{" == "} \mid \text{"!= "} \mid \text{" < "} \mid \text{" > "} \mid \text{" <= "} \mid \text{" >= "} \\
\\
TryCatch &\rightarrow TRY\ Bloc\ \{CatchPart\}\ [FinallyPart] \\
CatchPart &\rightarrow CATCH\ \text{"("}\ Type\ VARIABLE\ \text{")"}\ Bloc \\
FinallyPart &\rightarrow FINALLY\ Bloc \\
Bloc &\rightarrow \text{""}\ Instructions\ \text{""} \\
Type &\rightarrow IDENTIFIANT \\
Instructions &\rightarrow Instruction^{*}
\end{aligned}
$$

Explication :

- **Facteur** = élément de base (nombre, variable, ou parenthèse).
- **Terme** = suite de facteurs reliés par `*` ou `/`.
- **Expression** = suite de termes reliés par `+` ou `-`.
- **Affectation** = variable à gauche, expression à droite, terminée par `;`.
- **Type** = identifiant

5 Analyse syntaxique

L'analyse syntaxique est réalisée via une **descente récursive**. L'unique structure analysée en profondeur est :

— **try { ... } catch(type $var) { ... }** [**finally { ... }**]

Toutes les autres structures sont reconnues au niveau lexical mais ignorées par le parser.

Le parser valide :

— la présence correcte des accolades ;
— la bonne structure de try/catch ;
— la cohérence des expressions internes ;
— la présence d'au moins un bloc catch.

6 Structure du projet

L'arborescence proposée est :

```
mini_compilateur_php_Trycatch/
  AnalyseurLexical.java
  Token.java
  TokenType.java
  Interface.java
  AnalyseurSyntaxique.java
  php.txt
```

8 Exemple de test

Pour valider le fonctionnement du mini-compilateur, nous avons utilisé le fichier `php.txt` contenant différents cas de test : déclarations, affectations, structures de contrôle, mot-clé personnalisé et erreurs lexicales.

Exemple correct

```
try {
    $resultat = division(10, 0); // provoque une exception
    echo "Résultat: " . $resultat . "\n";
} catch (Exception $e) {
    echo "Erreur attrapée: " . $e->getMessage() . "\n";
} finally {
    echo "Bloc finally exécuté, nettoyage terminé.\n";
}
```

Sortie console (analyseur lexical et syntaxique)

```
===== Analyse Lexicale =====
<Type: TRY, Valeur: try, Ligne: 1>
<Type: ACCOLADE_OUVRANTE, Valeur: {, Ligne: 1>
```

```
<Type: VARIABLE, Valeur: $resultat, Ligne: 2>
<Type: AFFECTATION, Valeur: =, Ligne: 2>
<Type: IDENTIFIANT, Valeur: division, Ligne: 2>
<Type: PARENTHESE_OUVRANTE, Valeur: (, Ligne: 2>
<Type: NOMBRE, Valeur: 10, Ligne: 2>
<Type: VIRGULE, Valeur: ,, Ligne: 2>
<Type: NOMBRE, Valeur: 0, Ligne: 2>
<Type: PARENTHESE_FERMANTE, Valeur: ), Ligne: 2>
<Type: POINT_VIRGULE, Valeur: ;, Ligne: 2>
<Type: IDENTIFIANT, Valeur: echo, Ligne: 3>
<Type: CHAINE, Valeur: "Résultat: ", Ligne: 3>
<Type: POINT, Valeur: ., Ligne: 3>
<Type: VARIABLE, Valeur: $resultat, Ligne: 3>
<Type: POINT_VIRGULE, Valeur: ;, Ligne: 3>
<Type: ACCOLADE_FERMANTE, Valeur: }, Ligne: 4>
<Type: CATCH, Valeur: catch, Ligne: 5>
<Type: PARENTHESE_OUVRANTE, Valeur: (, Ligne: 5>
<Type: IDENTIFIANT, Valeur: Exception, Ligne: 5>
<Type: VARIABLE, Valeur: $e, Ligne: 5>
<Type: PARENTHESE_FERMANTE, Valeur: ), Ligne: 5>
<Type: ACCOLADE_OUVRANTE, Valeur: {, Ligne: 5>
<Type: IDENTIFIANT, Valeur: echo, Ligne: 6>
<Type: CHAINE, Valeur: "Erreur attrapée: ", Ligne: 6>
<Type: POINT, Valeur: ., Ligne: 6>
<Type: IDENTIFIANT, Valeur: getMessage, Ligne: 6>
<Type: PARENTHESE_OUVRANTE, Valeur: (, Ligne: 6>
<Type: PARENTHESE_FERMANTE, Valeur: ), Ligne: 6>
<Type: POINT_VIRGULE, Valeur: ;, Ligne: 6>
<Type: ACCOLADE_FERMANTE, Valeur: }, Ligne: 7>
<Type: FINALLY, Valeur: finally, Ligne: 8>
<Type: ACCOLADE_OUVRANTE, Valeur: {, Ligne: 8>
<Type: IDENTIFIANT, Valeur: echo, Ligne: 9>
<Type: CHAINE, Valeur: "Bloc finally exécuté, nettoyage terminé.", Ligne: 9>
<Type: POINT_VIRGULE, Valeur: ;, Ligne: 9>
<Type: ACCOLADE_FERMANTE, Valeur: }, Ligne: 10>
<Type: FIN, Valeur: , Ligne: 10>

===== Analyse Syntaxique =====
Structure try/catch/finally valide :
- Bloc try détecté
- Bloc catch détecté avec type Exception et variable $e
- Bloc finally optionnel détecté
- Accolades correctement fermées
```

Exemple erroné

```
try {
    $resultat = division(10, 0)   //  point-virgule manquant
```

```
    echo "Résultat: " . $resultat . "\n";
} catch {
    echo "Erreur attrapée sans variable !";
}
```

Sortie console (analyseur lexical et syntaxique)

```
===== Analyse Lexicale =====
<Type: TRY, Valeur: try, Ligne: 1>
<Type: ACCOLADE_OUVRANTE, Valeur: {, Ligne: 1>
<Type: VARIABLE, Valeur: $resultat, Ligne: 2>
<Type: AFFECTATION, Valeur: =, Ligne: 2>
<Type: IDENTIFIANT, Valeur: division, Ligne: 2>
<Type: PARENTHESE_OUVRANTE, Valeur: (, Ligne: 2>
<Type: NOMBRE, Valeur: 10, Ligne: 2>
<Type: VIRGULE, Valeur: ,, Ligne: 2>
<Type: NOMBRE, Valeur: 0, Ligne: 2>
<Type: PARENTHESE_FERMANTE, Valeur: ), Ligne: 2>
--  POINT_VIRGULE attendu mais absent
<Type: IDENTIFIANT, Valeur: echo, Ligne: 3>
...

===== Analyse Syntaxique =====
 ÉCHEC SYNTAXIQUE :
ERREUR SYNTAXIQUE Ligne 2
    Token trouvé : "echo"
    Attendu      : Il manque ';' à la fin de la déclaration de variable

ERREUR SYNTAXIQUE Ligne 5
    Token trouvé : "{"
    Attendu      : Paramètres attendus dans le catch (type + variable)
```

7 Conclusion

Ce mini-projet a permis de comprendre les bases d'un compilateur : analyse lexicale, analyse syntaxique, gestion des erreurs, et définition d'une grammaire.

L'objectif principal (analyse complète de l'instruction **try/catch**) a été atteint.