“ Rapport de Projet

Analyseur Lexical et Syntaxique (TRY CATCH)en Java

Nom et Prénom : Bedjou Mohand

Groupe : A1

Filière : Informatique

année : 2025/2026

Encadrant : M.Nadia TASLOUT

Introduction

Ce rapport présente le développement d'un **analyseur lexical et syntaxique** pour un mini langage inspiré de Java (TRY CATCH). L'analyseur lexical a pour rôle de transformer le code source en une suite de *tokens*, tandis que l'analyseur syntaxique vérifie que le code respecte les règles grammaticales définies.

L'objectif principal est de détecter les erreurs lexicales et syntaxiques.

Liste des tokens

Voici les tokens reconnus par l'analyseur lexical :

- **Mots-clés :** TRY, CATCH, INT, FLOAT, DOUBLE, CHAR, WHILE, IF, ELSE, DO, PUBLIC, CLASS, BEDJOU, MOHAND, FOR, FOREACH, SWITCH, CASE
- **Identifiants et littéraux :** IDENTIFIANT, ENTIER, REEL
- **Délimiteurs :** ACOLAD_OUV '{', ACOLAD_FER '}', PAR_OUV '(', PAR_FER ')', POINT_VIRGULE ';', VIRGULE ','
- **Opérateurs arithmétiques :** PLUS '+', MOINS '-', FOIS '*', DIVISE '/'
- **Affectation / égalité :** AFFECTATION '=', EGAL '=='
- **Comparaisons :** INFERIEUR '<', INFERIEUR_EGAL '<=', SUPERIEUR '>', SUPERIEUR_EGAL '>='
- **Fin / erreurs :** EOF, ERREUR

Analyseur Lexical

L'analyseur lexical est la première étape dans le traitement d'un programme source. Son rôle principal est de transformer le code source en une suite de tokens, qui seront ensuite utilisés par l'analyseur syntaxique pour vérifier la structure du programme.

Fonctions principales

Lecture du code source : Parcourt le texte caractère par caractère.
Reconnaissance des tokens : Identifie les mots-clés, identifiants, nombres, opérateurs et délimiteurs.
Gestion des espaces et commentaires : Ignore les espaces, tabulations, retours à la ligne et commentaires (`//` ou `/* ... */`).
Détection des erreurs lexicales : Signale tout caractère ou séquence non reconnue comme un token valide.

Structure interne

L'analyseur lexical implémente:

- Une **position courante** dans le texte source.
- Un **numéro de ligne** pour faciliter la localisation des erreurs.
- Une table et une **automat à états** pour reconnaître les identifiants et nombres.
- Une **liste d'erreurs** pour stocker toutes les erreurs lexicales détectées.

Analyseur Syntaxique

L'analyseur syntaxique intervient après l'analyseur lexical. Son rôle principal est de vérifier que la suite de tokens respecte la grammaire du langage et de détecter les erreurs syntaxiques.

Fonctions principales

Lecture des tokens : Utilise les tokens fournis par l'analyseur lexical.
Vérification de la grammaire : S'assure que le code respecte la structure syntaxique définie.
Gestion des erreurs : Signale les erreurs syntaxiques
Appel récursif : Implémente souvent un algorithme récursif descendant pour analyser les structures imbriquées.

Grammaire utilisée

- **CLS** → `public class IDENT { PROG }`
- **PROG** → INSTR PROG | EOF
- **INSTR** → DECL | AFFECT | BLOC | TRY_CATCH
- **DECL** → TYPE DECL_LIST ;
- **TYPE** → int | float | double | char
- **DECL_LIST** → DECL_ITEM DECL_LIST'
- **DECL_LIST'** → , DECL_ITEM DECL_LIST' | ϵ
- **DECL_ITEM** → IDENT DECL_SUITE
- **DECL_SUITE** → AFFECT EXPR | ϵ
- **AFFECT** → IDENT AFFECT EXPR ;
- **EXPR** → TERM EXPR'
- **EXPR'** → PLUS TERM EXPR' | MOINS TERM EXPR' | ϵ
- **TERM** → FACTEUR TERM'
- **TERM'** → FOIS FACTEUR TERM' | DIVISE FACTEUR TERM' | ϵ
- **FACTEUR** → IDENT | ENTIER | REEL | (EXPR)
- **BLOC** → { LIST_INSTR }
- **LIST_INSTR** → INSTR LIST_INSTR | ϵ
- **TRY_CATCH** → TRY BLOC CATCH BLOC

Arborescence du dossier `prog_try_catch`

```
prog_trycatch/
|
|-- analyserLexical.java
|-- analyseurSyntaxique.java
|-- tokens.java
|-- tokenType.java
-- main.java
```

Exemples de programmes à tester (avec try-catch)

Tous les programmes ci-dessous contiennent un bloc `try-catch`. Certains sont corrects, d'autres contiennent volontairement des erreurs.

Programmes corrects

```
// Programme 1 : déclaration et affectation
public class Test1 {
try {
int a = 10;
float b = 5.5;
} catch {
int error = 0;
}
}
```

```
// Programme 2 : expressions arithmétiques
public class Test2 {
try {
if( int z == 0){
int r = 5;
}
int x = 3 + 4 * 2;
float y = (x + 5) / 2.0;
} catch {
int error = 1;
}
}
```

```
// Programme 3 : identifiants et types multiples
public class Test3 {
try {
double d1 = 2.5, d2 = 3.7;
} catch {
double error = 0.0;
```

```
}
}

// Programme 4 : affectations multiples
public class Test4 {
try {
int i = 0;
i = i + 5;
} catch {
int error = 1;
}
}

// Programme 5 : expression imbriquée avec parenthèses
public class Test5 {
try {
int result = (2 + 3) * (4 - 1);
} catch {
int error = 0;
}
```

Programmes contenant des erreurs

```
// Programme 6 : erreur syntaxique (point-virgule manquant)
public class Test6 {
try {
int a = 10
} catch {
int error = s1;
}
}

// Programme 7 : erreur dans try-catch (catch mal formé)
public class Test7 {
try {
int x = 5;
} catch
int y = 0;
}

// Programme 8 : identifiant invalide
public class Test8 {
try {
int 1abc = 10;
} catch {
int error = 0;
```

```
}
}

// Programme 9 : parenthèses manquantes dans expression
public class Test9 {
try {
int x = (5 + 2 * 3;
} catch {
int error = 0;
}
}

// Programme 10 : accolade fermante manquante
public class Test10 {
try {
float y = 2.5;
} catch {
float error = 0.0;
}
}
```