Mini-Projet de Compilation

Mini-Compilateur en Java

Lounis Arezki

Année universitaire 2024–2025

Introduction

Ce mini-projet porte sur la réalisation d'un **mini-compilateur en Java** capable d'effectuer une analyse lexicale et une analyse syntaxique d'un sous-ensemble simplifié du langage Java, centré sur la structure `switch/case/default`.

Le compilateur lit un fichier source, découpe le texte en tokens, puis vérifie la validité syntaxique du code selon une grammaire définie. En cas d'erreurs, le système les signale avec le numéro de ligne et le lexème fautif.

1 Grammaire utilisée

La grammaire suivante décrit le sous-ensemble du langage Java pris en charge par le mini-compilateur. Elle est exprimée en notation EBNF.

Programme général

```
PROGRAMME      = { DECLARATION | INSTRUCTION } ;
```

Déclarations

```
DECLARATION    = DECL_CLASSE
               | DECL_METHODE
               | DECL_VAR
               | AFFECTATION ;

DECL_CLASSE    = [ MODIFICATEUR ] "class" IDENT
                 "{" { DECLARATION | INSTRUCTION } "}" ;

MODIFICATEUR   = "public" | "private" | "protected" ;

DECL_METHODE   = [ MODIFICATEUR ] [ "static" ] ( TYPE | "void" )
                 IDENT "(" PARAMS ")"
                 "{" { DECLARATION | INSTRUCTION } "}" ;

PARAMS         = { ANY_TOKEN } ;   (* Paramètres ignorés *)

DECL_VAR       = TYPE IDENT [ "=" EXPRESSION ] ";" ;

AFFECTATION    = IDENT "=" EXPRESSION ";" ;
```

Types

```
TYPE            = "int" | "char" | "float"
                | "double" | "boolean" ;
```

Instructions

```
INSTRUCTION     = SWITCH
                | AUTRE_INSTR ;

AUTRE_INSTR     = { TOKEN_SANS(";", "}") } ";" ;
```

Structure switch

```
SWITCH          = "switch" "(" EXPRESSION ")"
                  "{" { CASE } [ DEFAULT ] "}" ;

CASE            = "case" ( NOMBRE | CARACTERE ) ":"
                  { INSTRUCTION } ;

DEFAULT         = "default" ":" { INSTRUCTION } ;
```

Expressions

```
EXPRESSION      = OPERANDE { OPERATEUR OPERANDE } ;

OPERANDE        = IDENT
                | NOMBRE
                | CARACTERE
                | "true"
                | "false" ;

OPERATEUR       = "+" | "-" | "*" | "/" | "%"
                | "=" | "==" | "!="
                | "<" | "<=" | ">" | ">="
                | "++" | "--" ;
```

Atoms lexicaux

```
IDENT           = identificateur valide ;
NOMBRE          = suite de chiffres ;
CARACTERE       = caractère 'x' ;
```

2 Analyseur lexical

L'analyseur lexical lit le fichier source caractère par caractère et produit une liste de tokens. Chaque token contient :

— le lexème ;
— son type (mot-clé, identificateur, opérateur, etc.) ;
— son numéro de ligne.

Les catégories reconnues sont :

— mots-clés ;
— identificateurs ;

— constantes numériques ;
— constantes caractères ;
— chaînes ;
— opérateurs ;
— séparateurs ;
— erreur lexicale.

L'analyseur gère correctement les commentaires `//` et `/* */`, les chaînes de caractères et l'incrémentation des numéros de ligne.

3 Analyseur syntaxique

L'analyseur syntaxique utilise une **descente récursive**. Il vérifie dans l'ordre :
— les déclarations de classe ;
— les déclarations de méthode ;
— les déclarations de variables ;
— les affectations ;
— les instructions simples ;
— la structure `switch`.

Pour la structure `switch`, il exige la forme suivante :

```
switch (expression) {
    case valeur:
        instructions...
        break;
    ...
    default:
        instructions...
        break;
}
```

Chaque erreur est signalée sans interrompre l'analyse.

4 Cas de test

4.1 Programme correct

```
public class TestSwitch {

    public static void main(String[] args) {
        int x = 2;
        boolean t = true;

        switch (x) {
            case 1:
                System.out.println("un");
                break;
            case 2:
                System.out.println("deux");
                t = false;
                break;
            default:
                System.out.println("autre");
                break;
        }
    }
}
```

Résultat attendu :

```
ANALYSE SYNTAXIQUE SWITCH/CASE BIEN FAITE
```

4.2 Programme erroné

```
switch x {
    case :
        break
    default
        println("ok")
}
```

Erreurs détectées :
— parenthèse `(` manquante ;
— valeur absente dans `case` ;
— `break` sans point-virgule ;
— `default` sans deux-points ;
— accolade fermante manquante.

Conclusion

Ce mini-projet a permis d'implémenter un **analyseur lexical** et un **analyseur syntaxique** pour un sous-ensemble de Java. La structure `switch/case` est entièrement vérifiée selon une grammaire formelle, et les erreurs syntaxiques sont détectées avec précision.

Ce travail constitue une base solide pour comprendre les mécanismes fondamentaux des compilateurs modernes.