Rapport d'activité 2

TELECOM Nancy
Ingénieurs du numérique • Inspiring your digital future

Rapport d'activité 2
PCL1

Professeurs référents : S. COLLIN et S. DA SILVA

TELECOM NANCY
Projet de Compilation
Deuxième année (S7)

Semaine 49
Dernière mise à jour : le 5 décembre 2022

Membres du groupe :

Nicolas FRACHE
Théo GOUREAU
Cyrielle LACRAMPE–DITER
Rida MOUSSAOUI

TELECOM Nancy
193, Avenue Paul Muller
54600 Villers-lès-Nancy
+33 (0)3 72 74 59 00

Table des matières

1 Travail réalisé et difficultés rencontrées

Cette période s'étend du vendredi 18 novembre (semaine 46) au lundi 5 décembre (semaine 49). Le manque de temps est notre principale difficulté.

Semaine 46 Le checkpoint de PCL étant le vendredi et étant suivi d'un week-end (19 et 20 novembre) durant lequel s'organisait une manifestation majeure d'une association de l'école dans laquelle nous sommes tous les quatre engagés, nous n'avons pas pu profiter de cette fin de semaine travailler.

Semaine 47 Notre réunion hebdomadaire était le mercredi : nous avons mis au propre la grammaire en y ajoutant des *labels* à chaque règle. Nous nous sommes quittés en nous partageant la conception de l'AST.

Semaine 48 Notre réunion hebdomadaire était le mercredi : nous avons mis en commun la conception de l'AST. Nous nous sommes quittés en nous partageant l'implémentation de l'AST (deadline : le dimanche 4 décembre à 8:00).

Semaine 49 Le checkpoint de PCL étant le lundi à 8:00, nous n'avons pas le temps de travailler cette semaine là. L'inter-checkpoint était donc en réalité de seulement deux semaines (nous en avions prévues trois).

2 La conception de l'AST

2.1 Implémentation des visiteurs

Nous nous sommes servis du pattern *Visiteur* pour le parcours de notre structure arborescente. En effet :

- Nous avons labellisé toutes les règles de la grammaire suivant leurPl sémantique (cf. 1 : Semaine 47) ;
- Nous avons généré des des classes correspondantes au parser et au classes des visiteurs avec Maven ;
- Nous avons créé les classes de l'AST pour la création des noeuds de notre arbre abstrait ;
- Nous avons implémenté les classes de notre structure d'AST (boucles, expressions conditionnelles, déclarations, records, etc.).

2.2 Méthode de conception

Lors de la construction de l'AST, nous avons constaté la présence du motif visible sur la figure 2.2 qui se répète dans notre arbre syntaxique lors de la construction de listes. Notre intention dans ces cas est d'*aplatir* la liste afin de créer une liste de noeud sur un seul niveau.

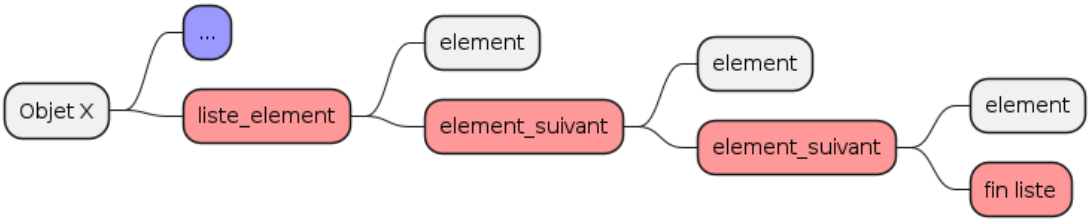


FIGURE 1 – Diagramme de classe d'un morceau de l'AST

Pour cela nous avons crée un objet *ListAccumulator<T>* visible sur le diagramme de classe de la figure 2.2. L'idée est de donner la responsabilité au noeud le plus profond (le plus à droite) de créer une instance de *ListAccumulator<T>* avec pour type générique celui de l'élément qu'on veut lister. Puis l'objet sera transmis successivement au noeud parent qui ajoute un élément en tête de l'objet. La méthode suivante nous permet d'encapsuler ce comportement les différents noeuds qui reprennent ce principe — la déclaration de record ou de fonction notamment.

```
public <T extends Ast> ListAccumulator<T> getListNode(
                            Ast gauche, Ast droite) {
if (droite == null) { // Si tout à droite, on crée un accumulateur
    return new ListAccumulator<>((T) gauche);
}
// Sinon on modifie celui existant
return ((ListAccumulator<T>) droite).addInHead((T) gauche);
```

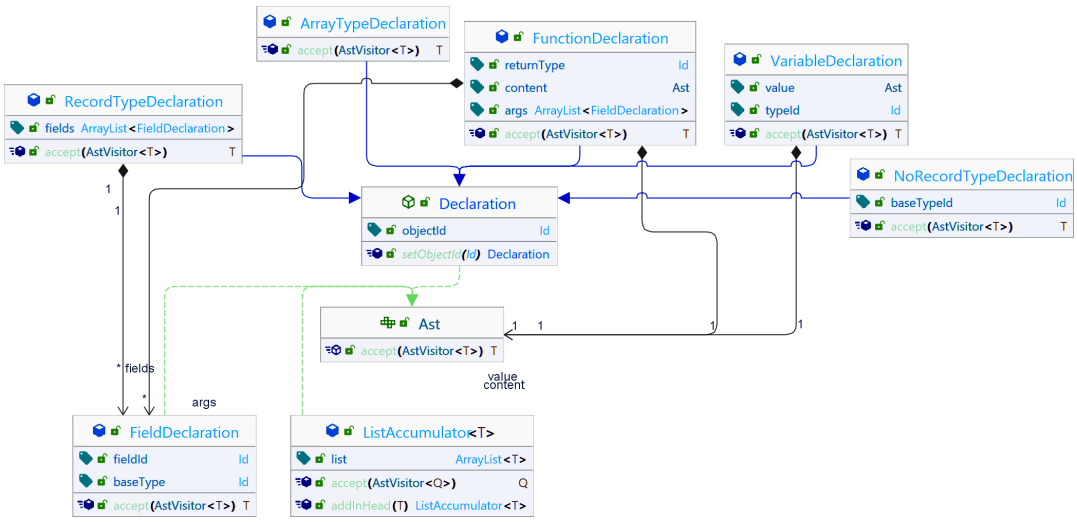
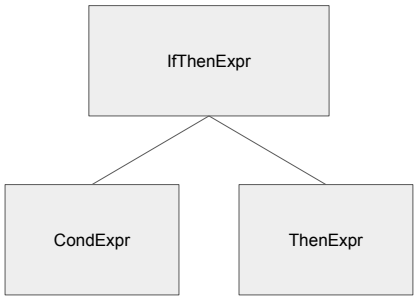


FIGURE 2 – Diagramme de classe d'un morceau de l'AST

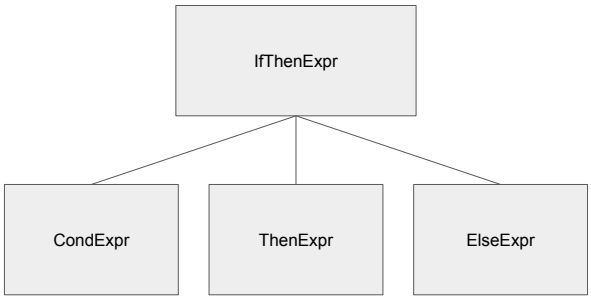
2.3 Nos morceaux d'AST

Expressions conditionnelles

IfThenExpr

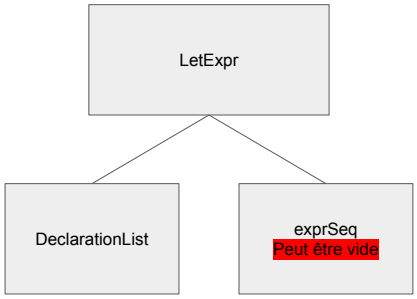


IfThenElseExpr

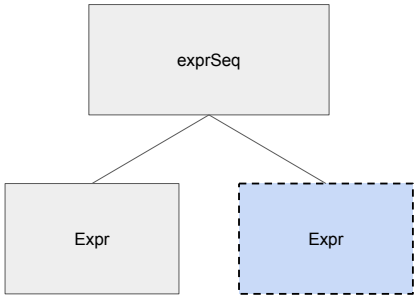


Let

LetExpr

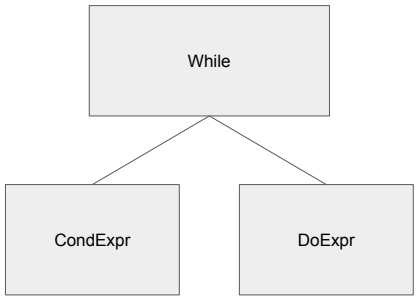


exprSeq

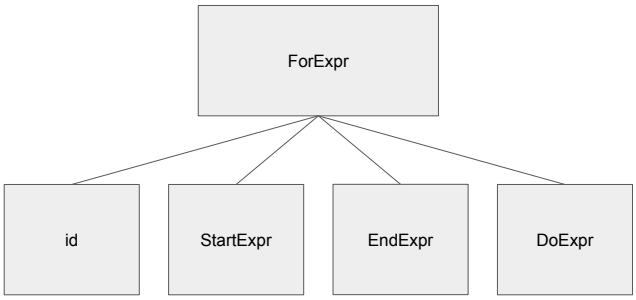


Boucles

WhileExpr

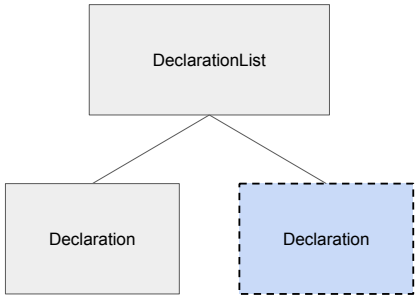


ForExpr

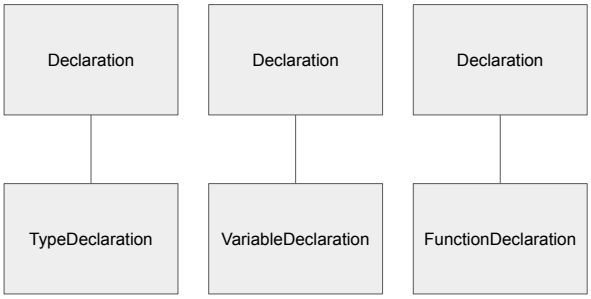


DeclarationList

DeclarationList

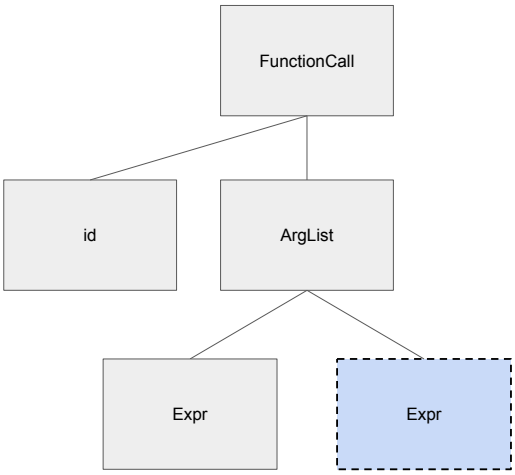


Declaration (Interface)



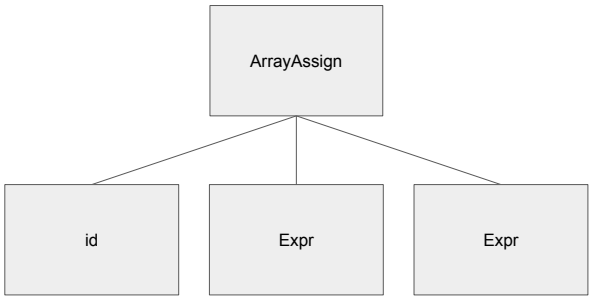
Function call

FunctionCall

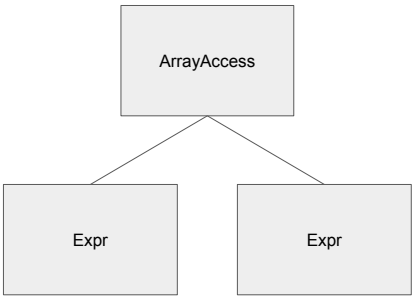


Array

ArrayAssign

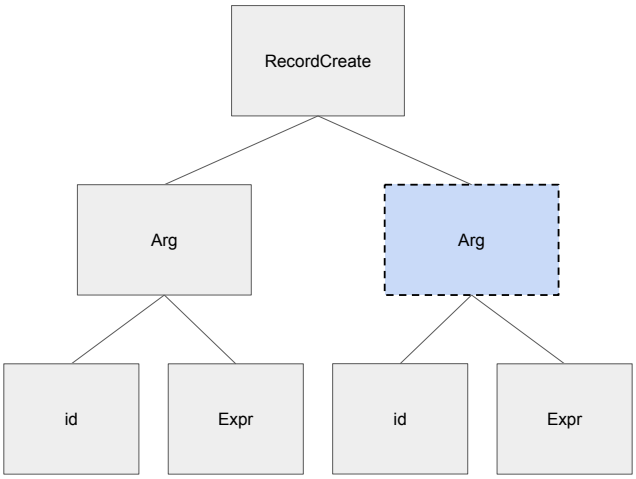


ArrayAccess

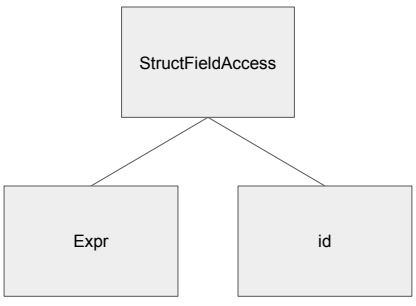


Record

RecordCreate

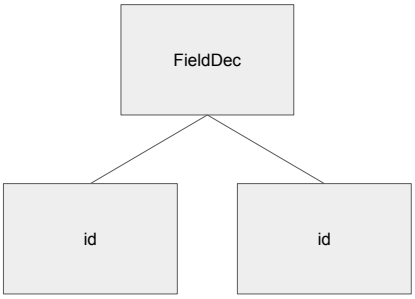


ForExpr

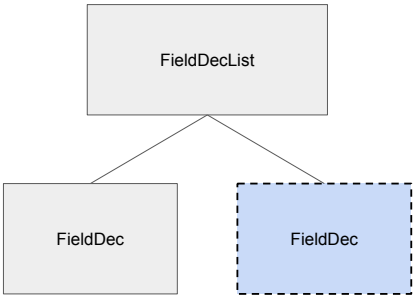


Field dec

FieldDec

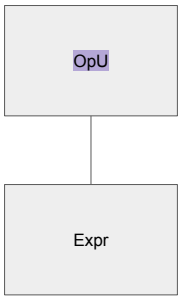


FieldDecList

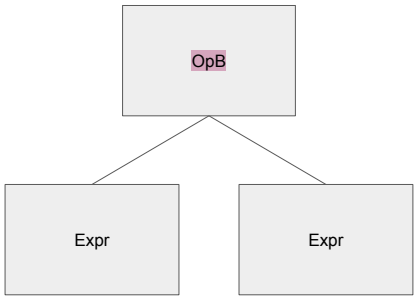


Opérateurs

Unaires (- de la négation, les parenthèses n'apparaissent pas dans l'AST)

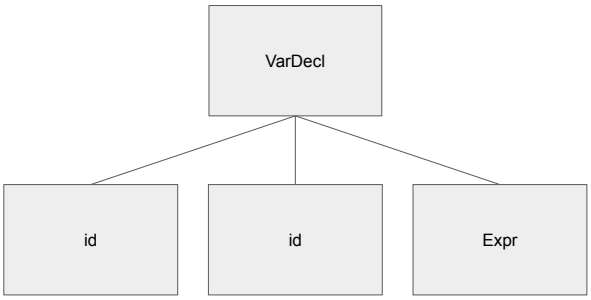


Binaires (| & * / + - > < <> <= >= = :=)

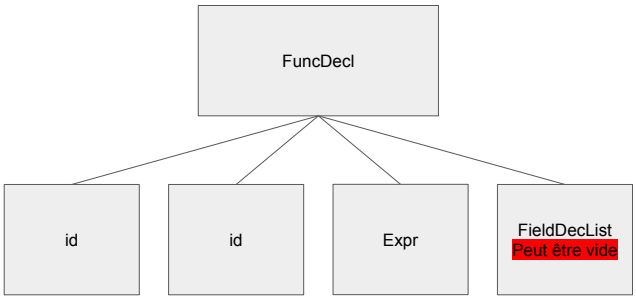


Déclaration (1/2)

VarDecl

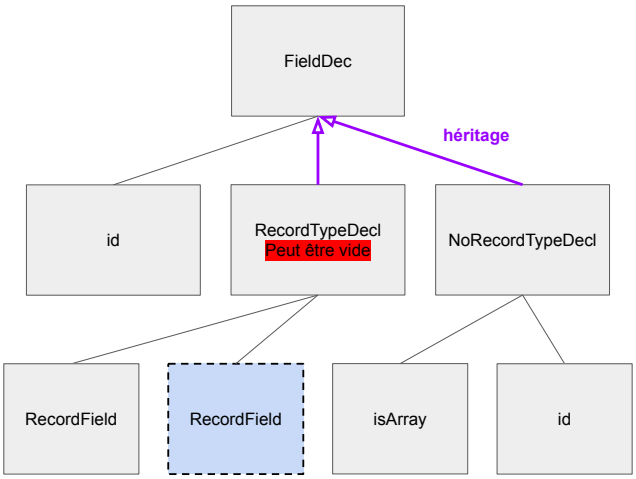


FuncDecl



Déclaration (2/2)

TypeDecl



RecordField