TP Analyse syntaxique.

L'objectif de cette séance est de réaliser un analyseur syntaxique [1] pour une partie du langage Java[2] qui a été étudié dans l'Unité d'Enseignement de Technologies Objets (UE TOB) au second semestre de première année. Cette partie du langage a été étudiée dans une séance de TD sur les grammaires dans l'Unité d'Enseignement de Modélisation et Architecture en première année. La partie syntaxique du langage est spécifée par le document `http://docs.oracle.com/javase/specs/jls/se11/html/index.html` plus précisémant dans le chapitre `http://docs.oracle.com/javase/specs/jls/se11/html/jls-19.html`.

1 Analyseur syntaxique

Un analyseur syntaxique est la deuxième étape de l'analyse d'un langage. Celle-ci est précédée par l'analyse lexicale [3] qui a été l'objet de la séance de TP précédente et suivie par l'analyse sémantique[4] qui sera l'objet de l'UE Sémantique et Traduction des Langages de la majeure Informatique en deuxième année.

Un analyseur syntaxique est spécifié sous la forme d'une grammaire dont les règles de production peuvent contenir des actions sémantiques. Ces actions sont effectuées lorsque l'analyseur syntaxique reconnait un mot qui fait parti du langage décrit par la grammaire associée. La grammaire peut également contenir la spécification des unités lexicales pour assurer la communication avec l'analyseur lexical.

2 L'outil camlyacc

Pour réaliser cette séance, nous allons utiliser l'outil `camlyacc`[5] qui traduit la spécification d'un analyseur syntaxique en un programme `caml`[6] qui implante un analyseur syntaxique sous la forme d'un automate fini à pile. Cet outil est conçu pour collaborer avec un analyseur lexical implanté avec l'outil `camllex` étudié dans la séance de TP précédente.

3 Etude d'un exemple

L'objectif est d'étudier l'exemple suivant en s'appuyant sur la documentation des outils `camllex` et `camlyacc` : `http://caml.inria.fr/pub/docs/manual-ocaml/lexyacc.html`. Plus précisément, nous utiliserons l'outil `menhir` compatible avec le format de fichier de `camlyacc` mais qui fournit un diagnostic plus facile à interpréter si la grammaire est ambiguë (voir `http://gallium.inria.fr/~fpottier/menhir/`).

L'analyseur lexical est défini dans le fichier `lexerJava.mll` (qui est une solution de la séance de TP précédente).

L'analyseur syntaxique est défini dans le fichier `parserJava.mly`. Il utilise l'analyseur lexical généré.

Le programme principal qui utilise l'analyseur syntaxique généré est défini dans le fichier `mainJava.ml`.

Pour compiler le programme, utilisez la commande `dune build mainJava.exe` qui produit l'exécutable `mainJava.exe` dans le répertoire habituel `_build/default`. Cet exécutable prend en paramètre le nom du fichier que l'on veut analyser.

[1] `http://en.wikipedia.org/wiki/Parsing`
[2] `http://en.wikipedia.org/wiki/Java_(programming_language)`
[3] `http://en.wikipedia.org/wiki/Lexical_analysis`
[4] `http://en.wikipedia.org/wiki/Semantics`
[5] `http://caml.inria.fr/pub/docs/manual-ocaml/lexyacc.html`
[6] `http://caml.inria.fr/pub/docs/manual-ocaml/`

4 Construction d'un analyseur syntaxique

L'objectif de cet exercice est de compléter le fichier fourni en exemple `parserJava.mly` pour traiter l'ensemble de la grammaire des méthodes Java décrite sous une forme de Conway en annexe. La documentation : `http://docs.oracle.com/javase/specs/jls/se11/html/jls-19.html` décrit l'intégralité de la syntaxe de Java qui dépasse l'objectif de cette séance.

Vous vous attacherez à traiter les différentes formes d'instructions et d'expressions.

5 Construction d'un analyseur sémantique

L'objectif de cet exercice est de compléter l'analyseur syntaxique que vous venez de réaliser avec différentes actions sémantiques qui collectent et affichent des informations sur le programme Java analysé.

1. Afficher le nombre de méthodes contenues dans un fichier
2. Afficher le nombre total d'instructions dans un fichier
3. Afficher le nombre de variables locales dans un bloc, i.e. un corps (afficher le résultat pour chaque bloc)
4. Afficher la profondeur maximum d'une fonction en terme de blocs imbriqués (afficher le résultat pour chaque fonction)

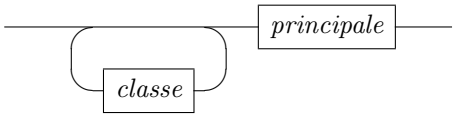
6 Extension des analyseurs lexical et syntaxique

L'objectif de cet exercice est d'étendre les fichiers `lexerJava.mll` et `parserJava.mly` pour ajouter la spécification des classes importées dans un fichier Java.

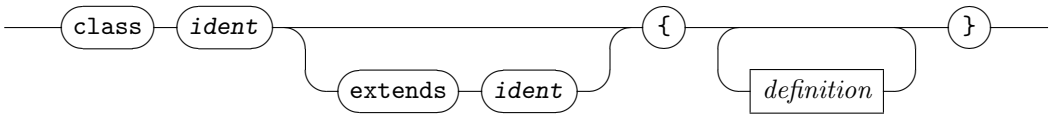
```
import Toto;
import foo.bar.Baz;
import truc.machin.*;
```

Plus précisément, il s'agit du mot-clef `import`, suivi soit d'un nom qualifié de classe (constitué d'une suite d'identificateurs séparés par des points), soit d'un nom qualifié désignant un ensemble de classes (constitué d'une suite d'identificateurs de séparés par des points et terminée par une étoile.

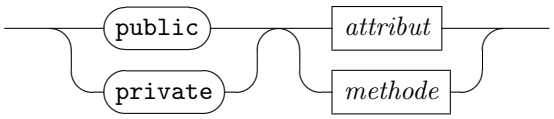
programme



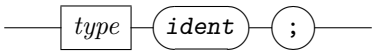
classe



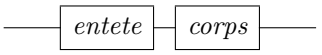
definition



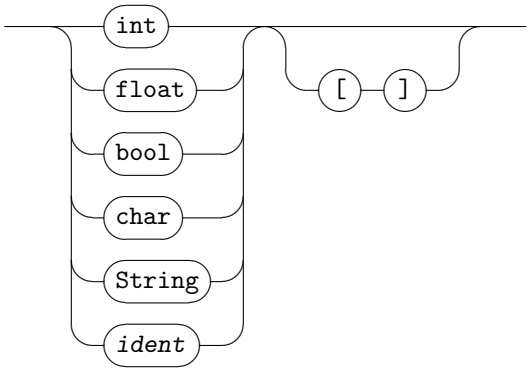
attribut



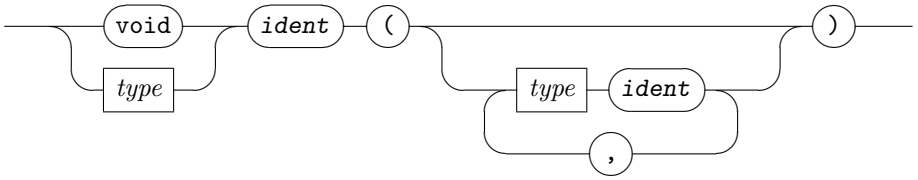
methode



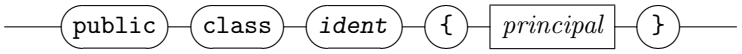
type



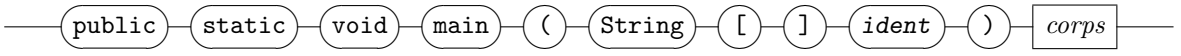
entete



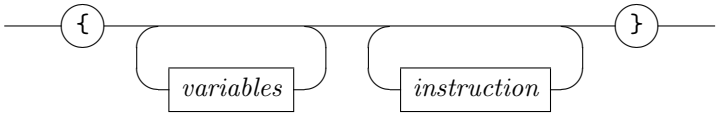
principale



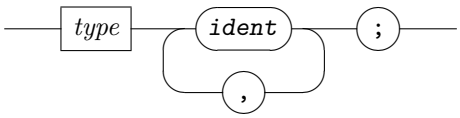
principal



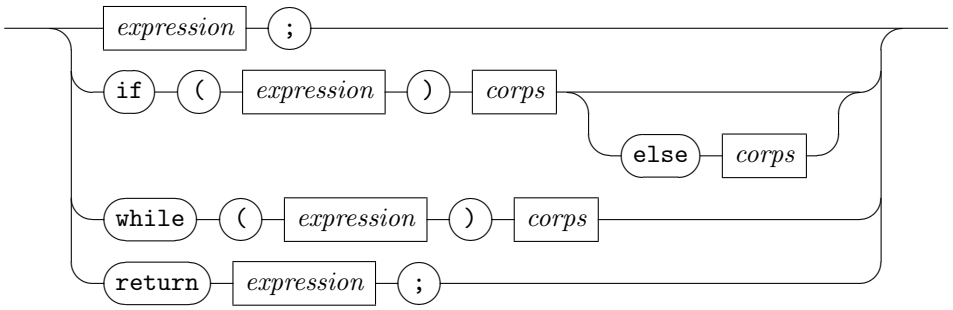
corps



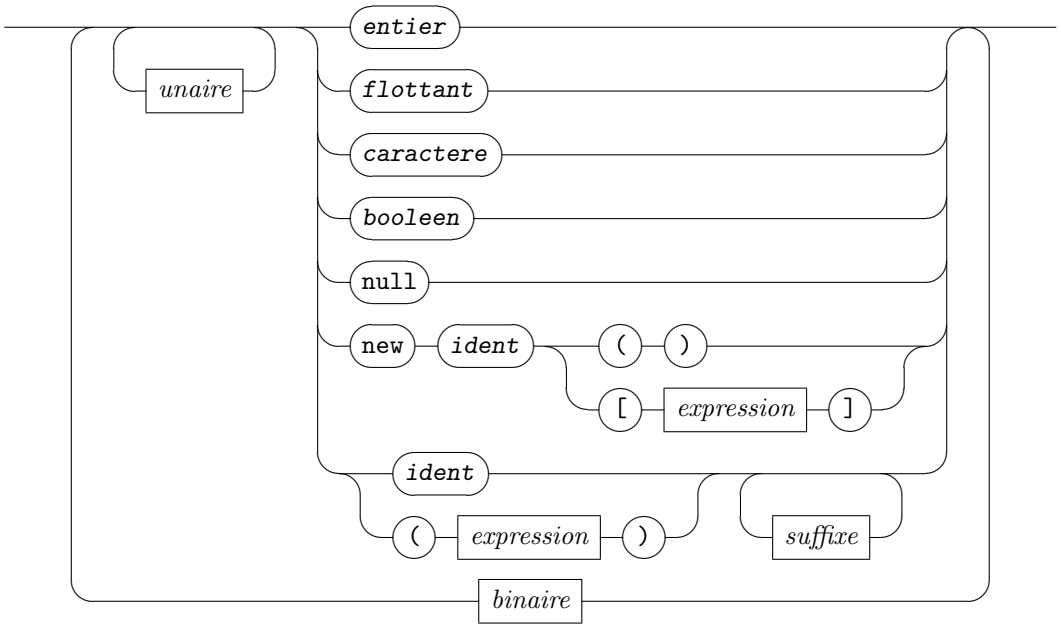
variables



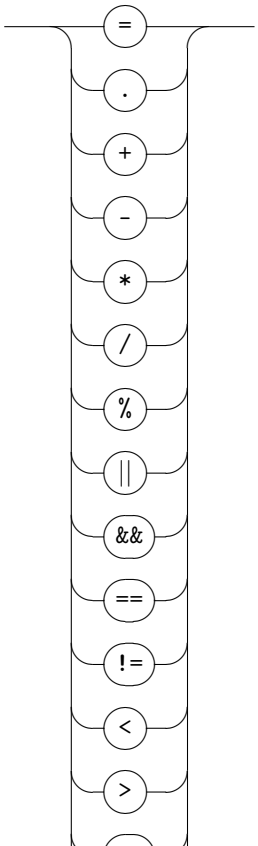
instruction



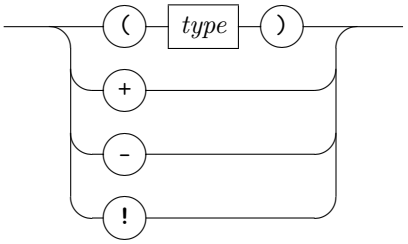
expression



binaire



unaire



suffixe