UNIVERSITÉ DE ROUEN

Compilateur LLVM

Langage jouet Kawa

Annexe Spécification Technique de Besoin 0.1

11 décembre 2014

Auteur(s): Kheireddine Berkane, Nasser Adjibi

Version	Date	Changelog
0.1	11/12/2014	Version initiale.

Table des matières

1 Grammaire

```
// $ -> mot vide
// Les mots en minuscules sont des mots clés
// J'ai pas mis la regle des commentaire parceque ça rendait la grammaire illisible
// Les commentaires sont insérables entre deux intructions
// Les espaces, retour de lignes et commentaires sont ignorés
// [opt1|opt2....|optn] choix exclusif entre les n options


SLASH : caractère slash

NOM_CLASS : mot commençant par une lettre majiscule suivit de caractères alphanumériques ou d'underscore

ENTIER : suite de caractère numériques

DOUBLE : Nombre a virgule ayant un point en guise de virgule. Peut être suivit d'un d. Exemple: 0.152 ou

FLOAT : Nombre a virgule ayant un poin en guise de virgule et qui est suivit d'un f. Exemple : 0.152f ou

CHAR : Tout les caractères possibles, encadrés par deux apostrophes.
Les caractères apostrophe et anti-slash doivent être despécialisé. Exemple : 'a' ou '\'.

STRING : Une suite de caractères délimitée par des doubles quotes. Les caractères '"', '\' doivent être

NOM_VARIABLE_BIEN_DEFINIT : un mot ne comportant que des caractère alphanumériques ou des underscores

SLASH_ETOILE : '/' directement suivit de '*'

ETOILE_SLASH : '*' directement suivit de '/'

CHAINE_SANS_ETOILE_SLASH : Une chaine de caratère ne contenant pas le caractère * suivit d'un slash

CHAINE_SANS_RETOUR_LIGNE : Une chaine de caractère ne contenant pas le caractère '\n'


SET_PACKAGE -> package PATH; | $

IMPORTATION -> import PATH; IMPORTATION | $

PATH -> NOM_CHAMP | NOM_CHAMP.PATH | NOM_CHAMP.*

CODE_START -> SET_PACKAGE IMPORTATION DEFINITION_CLASSE_OR_INTERFACE

COMMENTAIRE -> COMMENTAIRE_LIGNE | COMMENTAIRE_BLOC

COMMENTAIRE_LIGNE -> SLASH SLASH CHAINE_SANS_RETOUR_LIGNE

COMMENTAIRE_BLOC -> SLASH_ETOILE CHAINE_SANS_ETOILE_SLASH SLASH_ETOILE

DEFINITION_CLASSE_OR_INTERFACE -> DEFINITION_CLASSE | DEFINITION_INTERFACE | DEFINITION_CLASS_ABSTRACT

DEFINITION_CLASSE ->
PORTEE [final|$]
class NOM_CLASS
EXTEND_CLASSE EXTEND_INTERFACE
```

```
BLOC_CLASS

DEFINITION_INTERFACE ->
PORTEE [final|$] interface NOM_CLASS
EXTEND_INTERFACE
BLOC_INTERFACE

DEFINITION_CLASS_ABSTRACT ->
PORTEE abstract class
EXTEND_CLASSE EXTEND_INTERFACE
BLOC_CLASS_ABSTRACT


PORTEE -> public | private

EXTEND_CLASSE -> extends NOM_CLASS | $

EXTEND_INTERFACE -> extends NOM_CLASS | extends NOM_CLASS , LIST_NOM_INTERFACE | $

LIST_NOM_INTERFACE-> NOM_CLASS | , NOM_CLASS LIST_EXTENDS_INTERFACE | $


NOM_CHAMP -> NOM_VARIABLE_BIEN_DEFINIT | super | this

TYPE -> STYPE
STYPE -> static EXTYPE | EXTYPE
EXTYPE -> final VTYPE | abstract VTYPE | VTYPE
VTYPE -> value CTYPE| CTYPE
CTYPE -> NOM_TYPE | $

BLOC_CLASS -> { DECLARATION_IN_CLASS }

BLOC_INTERFACE -> {DECLARATION_IN_INTERFACE}

BLOC_CLASS_ABSTRACT -> {DECLARATION_IN_CLASS_ABSTRACT}

DECLARATION_IN_INTERFACE -> PORTEE [static|$] [final|$] VTYPE PROTOTYPE_METHODE; DECLARATION_IN_INTERFAC

DECLARATION_IN_CLASS_ABSTRACT ->
DECLARATION_IN_CLASS DECLARATION_IN_CLASS_ABSTRACT
| PORTEE abstract VTYPE PROTOTYPE_METHODE; DECLARATION_IN_CLASS_ABSTRACT
| $


DECLARATION_IN_CLASS ->  DECLARATION_ATTRIBUT  DECLARATION_IN_CLASS
 | DECLARATION_METHODE DECLARATION_IN_CLASS
 | $

DECLARATION_ATTRIBUT -> TYPE NOM_CHAMP;

DECLARATION_METHODE -> PROTOTYPE_METHODE { BLOC_METHODE }
| PROTOTYPE_CONTRUCTEUR { BLOC_METHODE }

PROTOTYPE_METHODE -> PORTEE TYPE NOM_CHAMP (LIST_PARAM)
PROTOTYPE_CONTRUCTEUR -> PORTEE NOM_CLASS (LIST_PARAM)
```

```
PARAM -> VTYPE NOM_CHAMP
LIST_PARAM -> PARAM | PARAM, LIST_PARAM | $


INSTRUCTIONS -> INSTRUCTION  INSTRUCTIONS | $;

INSTRUCTION -> DECLARATION_VARIBALE;
   | AFFECTATION;
   | EXPRESSION_RATIONELLE;
   | EXPRESSION_UNAIRE;
   | EXPRESSION_CONDITIONNELLE
   | APPEL_METHODE;
   | OPERATEUR_UNAIRE_RACCOURCIE NOM_VARIABLE_BIEN_DEFINIT
   | RETURN EXPRESSION_RATIONELLE;
   | EXPRESSION_BOUCLAGE
   | EXPRESSION_SWITCH
   | EXPRESSION_TRY_CATCH
   | CONTINUE;
   | BREAK;
   | $


CONSTANTE -> STRING | ENTIER | DOUBLE | CHAR | FLOAT


DECLARATION_VARIBALE -> VTYPE AFFECTATION_OU_NOM_CHAMP
  | VTYPE AFFECTATION_OU_NOM_CHAMP LIST_AFFECTATION_OU_NOM_CHAMP
  | $

AFFECTATION_OU_NOM_CHAMP -> AFFECTATION | NOM_CHAMP

LIST_AFFECTATION_OU_NOM_CHAMP -> , NOM_CHAMP LIST_AFFECTATION_OU_NOM_CHAMP | $

AFFECTATION -> NOM_CHAMP = EXPRESSION | DECLARATION_VARIBALE = EXPRESSION | $;

EXPRESSION_RATIONELLE -> NOM_CHAMP | CONSTANTE | EXPRESSION_BINAIRE | EXPRESSION_UNAIRE | EXPRESSION_TER
EXPRESSION_TERNAIRE -> (EXPRESSION_RATIONELLE)? EXPRESSION_RATIONELLE : EXPRESSION_RATIONELLE
EXPRESSION_BINAIRE -> EXPRESSION_RATIONELLE OPERATEUR_BINAIRE EXPRESSION_RATIONELLE
EXPRESSION_UNAIRE  -> OPERATEUR_UNAIRE NOM_CHAMP

EXPRESSION_CONDITIONNELLE ->  if ( EXPRESSION_RATIONELLE )  INSTRUCTION;
| if ( EXPRESSION_RATIONELLE ) { INSTRUCTIONS } EXPRESSION_CONDITIONNELLE_SECONDAIRE
EXPRESSION_CONDITIONNELLE_SECONDAIRE -> else { INSTRUCTIONS } | else if (EXPRESSION) EXPRESSION_CONDITIO


APPEL_METHODE -> NOM_CHAMP.NOM_CHAMP(LIST_EXPRESSION) | NOM_CHAMP(LIST_EXPRESSION);

LIST_EXPRESSION -> EXPRESSION | EXPRESSION LIST_EXPRESSION_SUITE
LIST_EXPRESSION_SUITE -> , EXPRESSION LIST_EXPRESSION_SUITE | $

OPERATEUR_BINAIRE -> + | - | / | * | || | != | =< | => | != | -= | += | *= | /= | << | >> | <<= | >>=
OPERATEUR_UNAIRE -> ! | OPERATEUR_UNAIRE_RACCOURCIE
OPERATEUR_UNAIRE_RACCOURCIE -> ++ | --

EXPRESSION_BOUCLAGE -> EXPRESSION_FOR | EXPRESSION_WHILE | EXPRESSION_DO_WHILE
```

```
EXPRESSION_FOR -> for(DECLARATION_VARIBALE; EXPRESSION_RATIONELLE; AFFECTATION_OU_NOM_CHAMP) { INSTRUCTI

EXPRESSION_WHILE -> while(EXPRESSION_RATIONELLE) { INSTRUCTIONS }

EXPRESSION_DO_WHILE -> do {INSTRUCTIONS} while(EXPRESSION_RATIONELLE);

EXPRESSION_SWITCH -> switch(EXPRESSION_RATIONELLE) {BLOC_INSTRUCTION_SWITCH}

BLOC_INSTRUCTION_SWITCH -> SWICTH_CASE_BLOC BLOC_INSTRUCTION_SWITCH | SWITCH_DEFAULT_BLOC BLOC_INSTRUCTI

SWICTH_CASE_BLOC -> case CONSTANTE : INSTRUCTIONS

SWITCH_DEFAULT_BLOC -> default : INSTRUCTIONS

EXPRESSION_TRY -> try { INSTRUCTIONS } catch(VTYPE NOM_CHAMP) {EXPRESSION} finaly {INSTRUCTIONS}
```