DIE SCHRIFTART SMALL CAPS (KAPITÄLCHEN). Die Schriftart sans serif (serifenlos).

Inhaltsverzeichnis

1 Parser

In our recursive-descent Parser we implemented all EBNF production rules as methods. The production rules of our EBNF allows alternatives, repetitions and optional parts. Figure (1) shows a dependence diagram of the parser methods.

To ensure that our EBNF is LL(1) conform, we created the a list with the first sets. The parser compares the actual token with the first-sets symbol and takes decisions between alternatives. To guarantee that the decision between two alternatives is correct, the first set of the alternatives must be disjunctive. We checked our EBNF with the Parser tool: coco, to ensure we are are LL(1) conform.

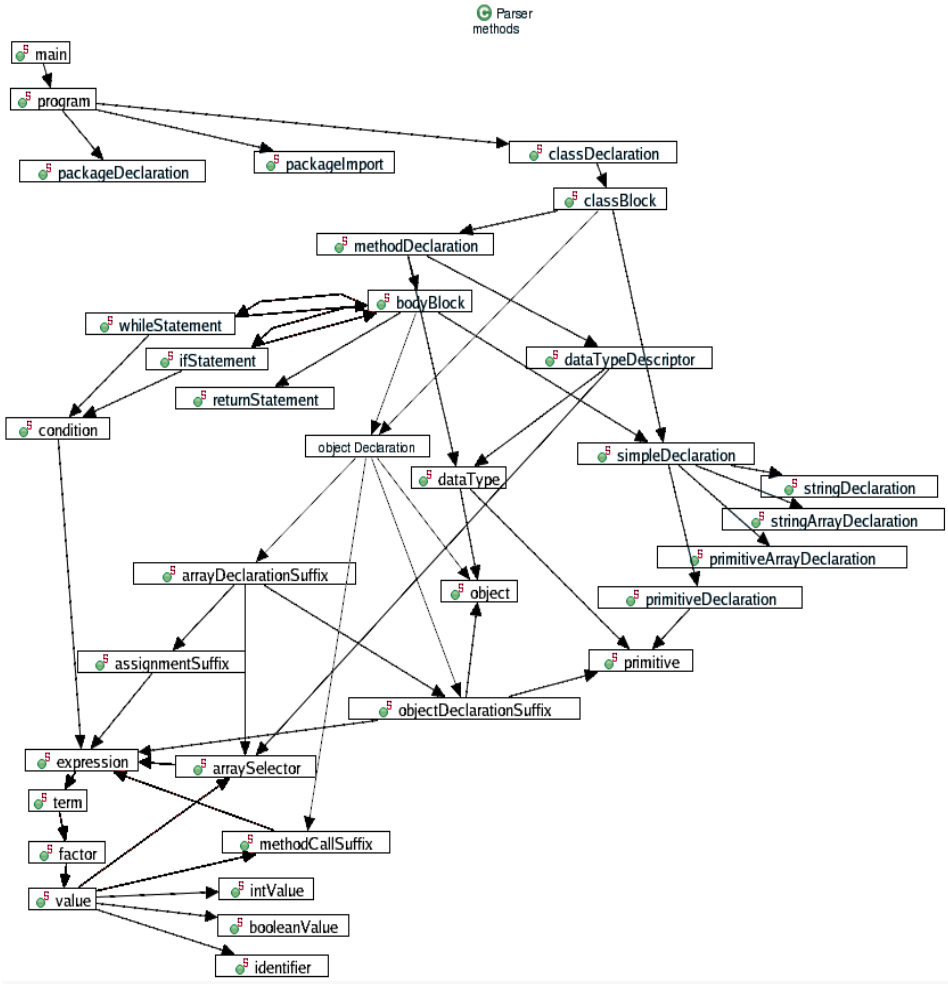

Abbildung 1: parser methods dependencies

1.1 First Symbols(first sets)

program	“package” “import” “public”
packageDeclaration	“package”
packageImport	“import”
classDeclaration	“public”
identifier	simpleIdentifier
classBlock	simpleIdentifier “public” “String” “String[]” “int” “boolean” “char” “int[]” “boolean[]” “char[]”
objectDeclaration	simpleIdentifier
simpleDeclaration	“String” “String[]” “int” “boolean” “char” “int[]” “boolean[]” “char[]”
methodDeclaration	“public”
object	simpleIdentifier
objectDeclarationSuffix	simpleIdentifier
primitiveDeclaration	“int” “boolean” “char”
primitiveArrayDeclaration	“int[]” “boolean[]” “char[]”
stringDeclaration	“String”
stringArrayDeclaration	SString[]"
datatype	simpleIdentifier SStringintbooleanchar"
datatypeDescriptor	simpleIdentifier SStringintbooleanchar"
bodyBlock	simpleIdentifier SStringString[]whileif returnintbooleancharint[]boolean[]char[]"
primitive	ïntbooleanchar"
assignmentSuffix	equal
primitiveArray	ïnt[]boolean[]char[]"
objectDeclarationAssignmentMethodcall	simpleIdentifier
arrayDeclarationSuffix	equal simpleIdentifier "["
methodCallSuffix	"("
expression	not number simpleIdentifier StringValue charValue "NULL-truefalse"
arraySelector	"["
whileStatement	"while"
ifStatement	ïf"
returnStatement	"return"
condition	not number simpleIdentifier StringValue charValue "NULL-truefalse"
value	not number simpleIdentifier StringValue charValue "NULL-truefalse"
intValue	number "
booleanValue	"truefalse"
factor	not number simpleIdentifier StringValue charValue "NULL-truefalse"
term	not number simpleIdentifier StringValue charValue "NULL-truefalse"

2 Syntax Error Handling

The Compiler is able to detect a certain amount of Syntax Errors. We distinguish between two error levels : week and strong errors.

- **Week Errors** will be fixed by the parser.
- **Strong Errors** can not be fixed.

2.1 Week Errors (Warnings)

Week errors are missing tokens which will be inserted by the parser. The code generator never get notice about those errors. When a week error occurs a warning will be displaced, the token will be inserted, parsing and code generation continues. Compiler is able to detect and correct following week errors:

- any missing ";"
- any missing "){}]"
- near all missing "(ëxcept in an object declaration, because we have to distinguish between "[änd "("
- in class declarations a missing “class” will be fixed.
- in method declarations when return type is missing (TODO !!!)
- in method declarations when “static” is missing
- in parameterlists we will detect and correct missing ","

2.2 Strong Errors

Strong Errors can be syntax errors as well as invalid symbols or invalid words. When a strong error occurs the parser displaces a error message and the code generation will be stopped. The parser goes to the next strong symbol and continue parsing. In our parser implementation we have sync operation implemented in three methods. See Figure (1), marked methods (TODO) serves as synchronisation points. Synchronization works this way:

- one of the synchronization methods recognices a strong error in one of its called method
- the error will be printed and code generation will stop
- the synchronization methods fetches new tokens until the next “first set” token of the appropiate method is fetched

Strong Errors can be:

- misplaced tokens. When the current token does not correspond to EBNF rules, except of week errors.
- any missing identifiers.

- any illegal terminal symbols, like identifier starting with a digit. This kind of error detects the scanner, and a error token will be delivered to the parser.

  For example, if we had a production

  ```
  A = a b c.
  ```

  for which the input was

  ```
  a x c
  ```

  the parser reports

  ```
  TODO Message
  ```