# Compilation - Programming Assignment 1

Team:

Ehud Tamir 036934644

Arbel Zinger 034666610

Inon Holdengreber 026504183

Code Structure and description of major classes

**Compiler.java** is our program’s main class. This class handles the input file and passes it through the token scanner from class Lexer. The compiler class also prints out the tokens in the required format and handles various errors. The compiler handles LexicalError which is thrown by the Lexer in case of a bad token, and also with I/O errors (such as file not found) and other errors.

The rest of the code is within the package IC.Parser.

**Lexer.java** is the lexical analysis scanner, class Lexer, is generated automatically using JFlex from the specification file IC.lex. The Lexer scans the input files and generates instances of class Token, which are returned to the caller (class Compiler in this case). In case of an invalid token the Lexer throws a LexicalError.

**IC.lex** contains our rules for tokenization. A full list of tokens and their respective regular expressions is given below.

**sym.java** defines a representation of token classes by integers. For example, the EOF token is represented by the constant sym.EOF, whose value is 0.

**Token.java** holds the representation of a token in a given input file. A token has 4 characteristics: The token ID (numeric representation of the token’s name), the line where the token appears, the token’s value (if applicable) and the token’s name. The token’s name is resolved according to its numeric value.

**LexicalError.java** implements an exception for errors in lexical analysis. Each instance of this exception has a line number where the error appeared, the string that caused the error and a custom message, sent by the Lexer.

Class Hierarchy

Package IC:

class **Compiler**

Package IC.Parser:

class **Lexer** implements **java\_cup.runtime.Scanner**

class **LexicalError** extends **Exception**

class **sym**

class **Token** extends **java\_cup.runtime.Symbol**

Testing Strategy

Our tests, consisting on both lexically correct and on lexically incorrect files, focused on verifying the following aspect of the lexical analysis:

* Correct handling of comments
* Correct handling of strings
* Handling all the language’s tokens.
* Handling an empty file.
* Identifying illegal integers (integers that have preceding zeros).
* Identifying illegal tokens.

These tests were automated using a Linux shell script. In addition, more edge cases were tested. Such as supplying the program with an illegal amount of arguments, specifying a file that does not exist and more.

List of tokens and their regular expressions

**Macros**

|  |  |
| --- | --- |
| ***Name*** | ***Regular Expression*** |
| WHITESPACE | [" "\n\t\r] |
| ALPHA | [a-zA-Z\_] |
| UPPER | [A-Z] |
| LOWER | [a-z] |
| DIGIT | [0-9] |
| NONZERO | [1-9] |
| ALPHA\_NUMERIC | {ALPHA}|{DIGIT} |
| ID | {LOWER}{ALPHA\_NUMERIC}\* |
| CLASS\_ID | {UPPER}{ALPHA\_NUMERIC}\* |
| STRING\_TEXT | ([\x20-\x21\x23-\x5b\x5d-\x7e]|\\[\\nt\"])\* |
| COMMENT\_TEXT | ([^\\*]|\\*[^/])\*\\*? |

**Tokens**

|  |  |
| --- | --- |
| ***Name*** | ***Regular Expression*** |
| EOF | N/A (detected by Lexer separately) |
| Whitespace | {WHITESPACE} |
| class | class |
| extends | extends |
| static | static |
| void | void |
| int | int |
| boolean | boolean |
| string | string |
| return | return |
| if | if |
| else | else |
| while | while |
| break | break |
| continue | continue |
| this | this |
| new | new |
| length | length |
| true | true |
| false | false |
| null | null |
| Classes ID | {CLASS\_ID} |
| ID | {ID} |
| Illegal Int | 0+{DIGIT}+ |
| Int | 0|({NONZERO}{DIGIT}\*) |
| LP | "(" |
| RP | ")" |
| LCBR | "{“ |
| RCBR | "}" |
| LB | "[" |
| RB | "]" |
| COMMA | "," |
| DOT | "." |
| SEMI | ";" |
| QUOTE | \"{STRING\_TEXT}\" |
| Single line comment | "//".\* |
| Multi Line comment | "/\*"{COMMENT\_TEXT}"\*/" |
| ASSIGN | "=" |
| EQUAL | "==" |
| GT | ">" |
| LT | "<" |
| GTE | ">=" |
| LTE | "<=" |
| NEQUAL | "!=" |
| LAND | "&&" |
| LOR | "||" |
| LNEG | "!" |
| PLUS | "+" |
| MINUS | "-" |
| MULTIPLY | "\*" |
| DIVIDE | "/" |
| MOD | "%" |