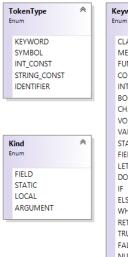
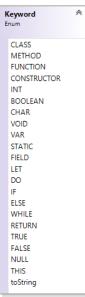
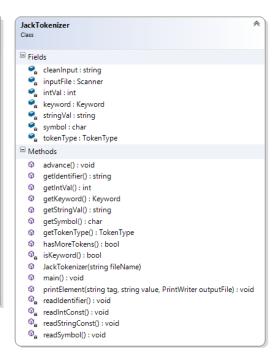
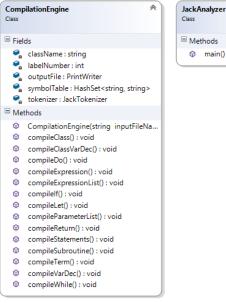
## The Jack Compiler 1: Syntax Analysis











#### Jack tokens

#### 'class'|'constructor'|'function'| 'method' | 'field' | 'static' | 'var' | 'int' | 'char' | 'boolean' | 'void' | 'true' | 'false' | 'null' | 'this' | 'let' | 'do' | 'if' | 'else' | 'while' | 'return' symbol: '/'|%'|'|'|''|''|'|' integerConstant a decimal number in the range 0 ... 32767 StringConstant: "" a sequence of Unicode characters, not including double quote or newline '"' identifier: a sequence of letters, digits, and underscore ('\_') not starting with a digit.

### Jack tokenizer

```
TestClassT.xml
TestClass.jack
                                       <tokens>
 if (x < 0) {
                                          <keyword> if </keyword>
    let sign = "negative";
                                          <symbol> ( </symbol>
 }
                                          <identifier> x </identifier>
                       tokenizer
                                          <symbol> &lt; </symbol>
                                          <integerConstant> 0 </integerConstant>
                                          <symbol> ) </symbol>
                                          <symbol> ( </symbol>
                                          <keyword> let </keyword>
string constants are outputted
                                          <identifier> sign </identifier>
without the double-quotes
                                          <symbol> = </symbol>
                                          <stringConstant> negative </stringConstant>
                                          <symbol> ; </symbol>
<, >, ", and & are outputted as
                                          <symbol> } </symbol>
<, &gt;, &quot;, and &amp;
                                       </tokens>
```

# JackTokenizer API

**JackTokenizer:** Ignores all comments and white space in the input stream, and serializes it into Jack-language tokens. The token types are specified according to the Jack grammar.

Routine	Arguments	Returns	Function
Constructor	input file / stream		Opens the input . jack file and gets ready to tokenize it.
hasMoreTokens	-	boolean	Are there more tokens in the input?
advance	_		Gets the next token from the input, and makes it the current token.  This method should be called only if hasMoreTokens is true.  Initially there is no current token.
tokenType	_	KEYWORD, SYMBOL, IDENTIFIER, INT_CONST, STRING_CONST	Returns the type of the current token as a constant.

keyWord	_	CLASS, METHOD, FUNCTION, CONSTRUCTOR, INT, BOOLEAN, CHAR, VOID, VAR, STATIC, FIELD, LET, DO, IF, ELSE, WHILE, RETURN, TRUE, FALSE, NULL, THIS	Returns the keyword which is the current token, as a constant.  This method should be called only if tokenType is KEYWORD.
symbol	-	char	Returns the character which is the current token. Should be called only if tokenType is SYMBOL.
identifier	-	string	Returns the identifier which is the current token. Should be called only if tokenType is IDENTIFIER.
intVal		int	Returns the integer value of the current token. Should be called only if tokenType is INT_CONST.
stringVal	-	string	Returns the string value of the current token, without the two enclosing double quotes. Should be called only if tokenType is STRING_CONST.

#### The Jack Language Grammar

```
Lexical elements:
                   The Jack language includes five types of terminal elements (tokens):
        keyword:
                   'class' | 'constructor' | 'function' |
                   'method' | 'field' | 'static' | 'var' |
                   'int' | 'char' | 'boolean' | 'void' | 'true' |
                   'false' | 'null' | 'this' | 'let' | 'do' |
                   'if' | 'else' | 'while' | 'return'
                  ·(' | '3' | '(' | ')' | '(' | '1' | '1' | '.' |
         symbol:
                   ',' | ';' | '+' | '-' | '*' | '/' | '&' |
                   '|' | '<' | '>' | '=' | '~'
  integerConstant: A decimal number in the range 0 .. 32767.
                   "" A sequence of Unicode characters not including double quote or
   StringConstant
                   newline '"'
        identifier:
                   A sequence of letters, digits, and underscore (' ') not starting with a
                   A Jack program is a collection of classes, each appearing in a separate file.
Program structure:
                   The compilation unit is a class. A class is a sequence of tokens structured
                   according to the following context free syntax:
            class: 'class' className '{' classVarDec* subroutineDec* '}'
     classVarDec: ('static' | 'field') type varName (', 'varName)* ';'
                  'int' | 'char' | 'boolean' | className
            type:
   subroutineDec:
                   ('constructor' | 'function' | 'method')
                   ('void' | type) subroutineName '(' parameterList ')'
                   subroutineBody
   parameterList: ((type varName) (', ' type varName)*)?
  subroutineBody:
                  '{' varDec* statements '}'
         varDec:
                  'var' type varName (',' varName)*';'
      className:
                  identifier
 subroutineName:
                   identifier
       varName:
                  identifier
```

#### The Jack Language Grammar (continued)

```
Statements:
                   statement*
       statements:
       statement: letStatement | ifStatement | whileStatement |
                   doStatement | returnStatement
     letStatement:
                   'let' varName ('[' expression ']')? '=' expression ';'
                   'if' '(' expression ')' '{' statements '}'
      ifStatement:
                   ('else' '{' statements '}')?
  whileStatement:
                   'while' '(' expression ')' '{' statements '}'
     doStatement:
                   'do' subroutineCall ';'
 ReturnStatement
                   'return' expression? ';'
Expressions:
       expression:
                   term (op term)*
            term:
                   integerConstant | stringConstant | keywordConstant |
                   varName | varName '[' expression ']' | subroutineCall |
                    '(' expression ')' | unaryOp term
   subroutineCall:
                   subroutineName '(' expressionList ')' | (className |
                   varName) '.' subroutineName '(' expressionList ')'
                   (expression (', 'expression)*)?
   expressionList:
                   '+' | '-' | '*' | '/' | '&' | '|' | '<' | '>' | '='
        unaryOp:
                   '-' | '~'
KeywordConstant: 'true' | 'false' | 'null' | 'this'
```