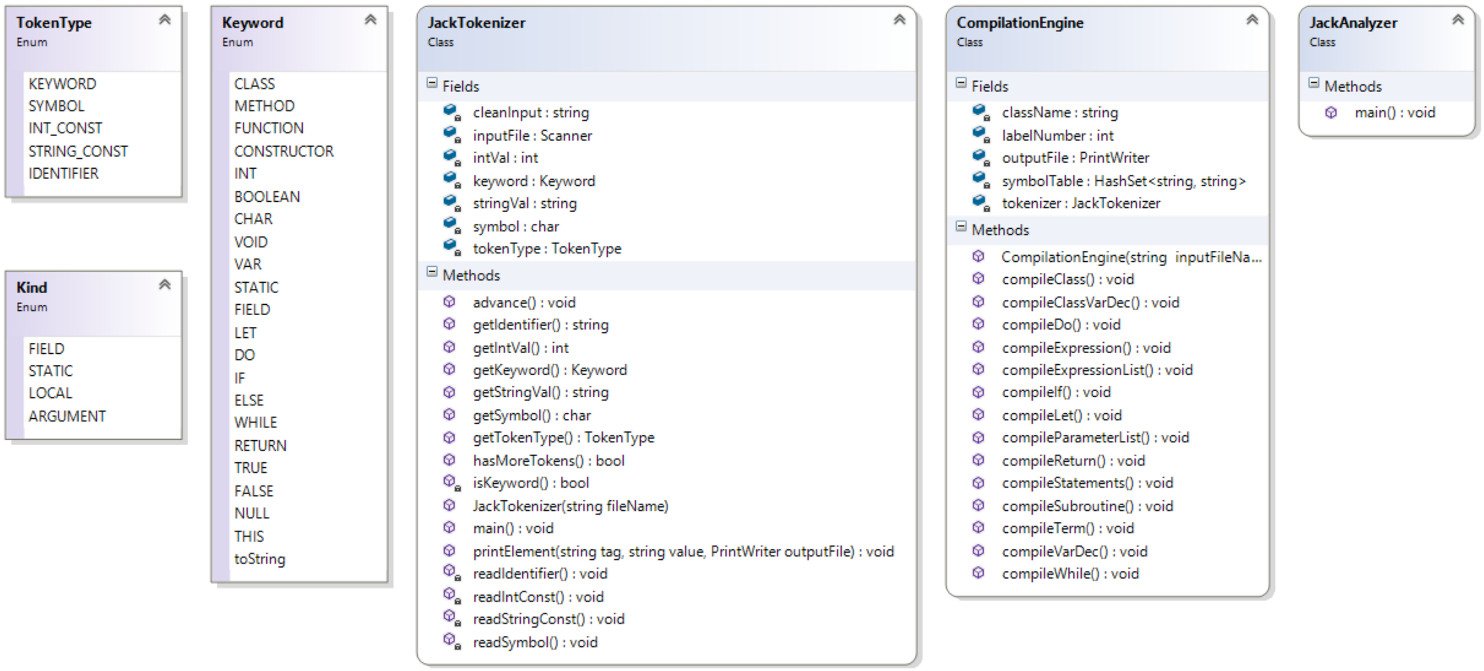
Name: Raul Aguilar
Date: ____________

The Jack Compiler 1: Syntax Analysis



TokenType Enum
- KEYWORD
- SYMBOL
- INT_CONST
- STRING_CONST
- IDENTIFIER

Kind Enum
- FIELD
- STATIC
- LOCAL
- ARGUMENT

Keyword Enum
- CLASS
- METHOD
- FUNCTION
- CONSTRUCTOR
- INT
- BOOLEAN
- CHAR
- VOID
- VAR
- STATIC
- FIELD
- LET
- DO
- IF
- ELSE
- WHILE
- RETURN
- TRUE
- FALSE
- NULL
- THIS
- toString

JackTokenizer Class

Fields
- cleanInput : string
- inputFile : Scanner
- intVal : int
- keyword : Keyword
- stringVal : string
- symbol : char
- tokenType : TokenType

Methods
- advance() : void
- getIdentifier() : string
- getIntVal() : int
- getKeyword() : Keyword
- getStringVal() : string
- getSymbol() : char
- getTokenType() : TokenType
- hasMoreTokens() : bool
- isKeyword() : bool
- JackTokenizer(string fileName)
- main() : void
- printElement(string tag, string value, PrintWriter outputFile) : void
- readIdentifier() : void
- readIntConst() : void
- readStringConst() : void
- readSymbol() : void

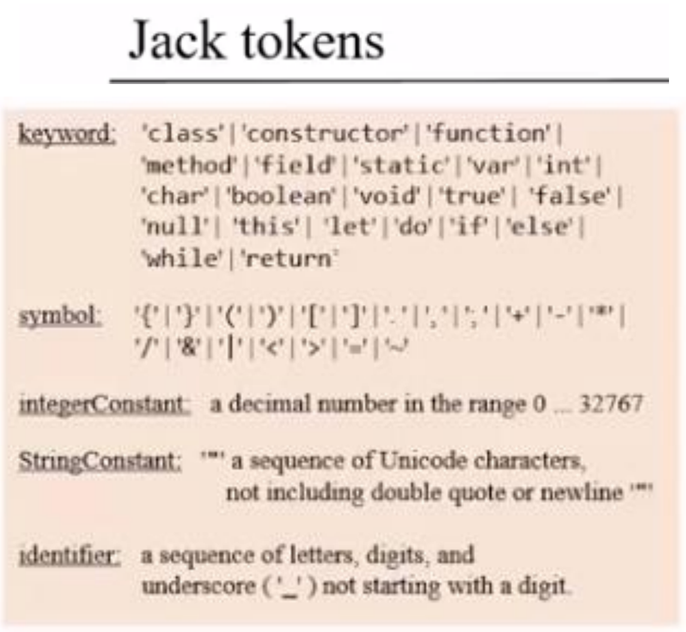
CompilationEngine Class

Fields
- className : string
- labelNumber : int
- outputFile : PrintWriter
- symbolTable : HashSet<string, string>
- tokenizer : JackTokenizer

Methods
- CompilationEngine(string inputFileNa...
- compileClass() : void
- compileClassVarDec() : void
- compileDo() : void
- compileExpression() : void
- compileExpressionList() : void
- compileIf() : void
- compileLet() : void
- compileParameterList() : void
- compileReturn() : void
- compileStatements() : void
- compileSubroutine() : void
- compileTerm() : void
- compileVarDec() : void
- compileWhile() : void

JackAnalyzer Class

Methods
- main() : void



Jack tokens

keyword:	'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.

Tokenizer Questions:

1. Translate the following Jack code into its tokenized XML code.

Jack Code	Tokenized XML
How many tokens are generated? 9	

`var int x;`
`let x = 3;`
	`<keyword> var </keyword>`
	`<keyword> int </keyword>`
	`<identifier> x </identifier>`
	`<symbol> ; </symbol>`
	`<keyword> let </keyword>`
	`<identifier> x </identifier>`
	`<symbol> = </symbol>`
	`<integerConstant> 3`
	`</integerConstant>`
	`<symbol> ; </symbol>`

2. Translate the following Jack code into its tokenized XML code.

Jack Code	**Tokenized XML**
How many tokens are generated? 19	

```
var int value;
if (y > x)
{
      let value = x & y;
}
```

```
<tokens>
 <keyword> var </keyword>
 <keyword> int </keyword>
 <identifier> value </identifier>
 <symbol> ; </symbol>
 <keyword> if </keyword>
 <symbol> ( </symbol>
 <identifier> y </identifier>
 <symbol> &gt; </symbol>
 <identifier> x </identifier>
 <symbol> ) </symbol>
 <symbol> { </symbol>
 <keyword> let </keyword>
 <identifier> value </identifier>
 <symbol> = </symbol>
 <identifier> x </identifier>
 <symbol> &amp; </symbol>
 <identifier> y </identifier>
 <symbol> ; </symbol>
 <symbol> } </symbol>
</tokens>
```

3. How many tokens will be produced by the following Jack code? Do not translate into XML, just count tokens.

Jack Code **How many tokens are generated? 54**	**Answer =**
(code below)	

```
var int x, y, greatest;
let x = 3;
let y = 5;
if (x > y)
{
      let greatest = x;
}
else
{
      let greatest = y;
}
do Output.printInt(greatest);
do Output.println();
```

Syntax Analyzer Questions:

1. Translate the following Jack code into its XML parse tree (using the Jack grammar).

Jack Code	**XML Parse Tree**

Jack Code:

```
let greatest = “x is greater”;
```

XML Parse Tree:

```
<tokens>
      <letStatement>
            <keyword> let </keyword>
            <varName>
                  <identifier> greatest
</identifier>
            </varName>
            <symbol> = </symbol>
            <expression>
                  <term>

      <stringConstant> x is greater
</stringConstant>
                  </term>
            </expression>
            <symbol> ; </symbol>
      </letStatement>
</tokens>
```

2. Translate the following Jack code into its XML parse tree (using the Jack grammar).

Jack Code	XML Parse Tree

`do Output.printInt(greatest);`	`<tokens>`
	`<doStatement>`
	`<keyword> do </keyword>`
	`<subroutineCall>`
	`<className>`
	`<identifier> Output </identifier>`
	`</className>`
	`<symbol> . </symbol>`
	`<subroutineName>`
	`<identifier> printInt </identifier>`
	`</subroutineName>`
	`<symbol> ( </symbol>`
	`<expressionList>`
	`<expression>`
	`<term>`
	`<varName>`
	`<identifier> greatest </identifier>`
	`</varName>`
	`</term>`
	`</expression>`
	`</expressionList>`
	`<symbol> ) </symbol>`
	`</subroutineCall>`
	`<symbol> ; </symbol>`
	`</doStatement>`
	`</tokens>`

3. Translate the following Jack code into its XML parse tree (using the Jack grammar).

Jack Code

```
class Point
{
   field int x, y;

   constructor Point new()
   {
      let x = 0;
      let y = 0;
   }
}
```

XML Parse Tree

```
<tokens>
      <class>
            <className>
                  <identifier> Point
</identifier>
            </className>
            <symbol> { </symbol>
            <classVarDec>
                  <keyword> field
</keyword>
                        <type>

      <keyword> int </keyword>
                        </type>
                        <varName>

      <identifier> x </identifier>
                        </varName>
                        <symbol> ,
</symbol>
                        <varName>

      <identifier> y </identifier>
                        </varName>
                        <symbol> ;
</symbol>
            </classVarDec>
            <subroutineDec>
                  <keyword> constructor
</keyword>
                  <type>
                        <className>

      <identifier> Point </identifier>
                        </className>
                  </type>
                  <subroutineName>
                        <identifier>
new </identifier>
                  </subroutineName>
                  <symbol> ( </symbol>
                  <parameterList>

                  </parameterList>
                  <symbol> ) </symbol>
                  <subroutineBody>
                        <symbol> {
</symbol>
                        <statements>

      <letStatement>

      <keyword> let </keyword>

      <varName>

      <identifier> x </identifier>
```

```
       </varName>

       <symbol> = </symbol>

       <expression>

       <term>

              <integerConstant> 0
</integerConstant>

       </term>

       </expression>

       <symbol> ; </symbol>

       </letStatement>

       <letStatement>

       <keyword> let </keyword>

       <varName>

       <identifier> y </identifier>

       </varName>

       <symbol> = </symbol>

       <expression>

       <term>

              <integerConstant> 0
</integerConstant>

       </term>

       </expression>

       <symbol> ; </symbol>

       </letStatement>
                            </statements>
                            <symbol> }
</symbol>
                     </subroutineBody>
              </subroutineDec>
              <symbol> } </symbol>
       </class>
</tokens>
```

Summative Questions:

1) Consider the following statements, taken from three different Jack programs. In each one of these programs, the identifier foo represents a different thing:

```
let x = 5 + foo - a      // Here foo represents a simple variable.
let y = foo[12] - 3      // Here foo represents an array.
let z = 2 * foo.val()    // Here foo represents an object.
```

Suppose that we are parsing any one of these statements (we don't know which), and that the current token is foo. How many more tokens do we have to read until we can tell what we have is a simple variable, an array reference, or a method call?

a) Zero
b) One
[✓ c) Two]
d) Three
e) All answers are wrong

2) Consider the following rule, taken from the Jack grammar:

```
parameterList: ( ( type varName ) ( ',' type varName ) * ) ?
```

Select the correct statement(s):

[✓ a)] The rule will parse successfully an empty parameter list
b) The symbol * (in this context) is part of the Jack language
c) The symbol * means 0 or 1
[✓ d)] The symbol * means 0 or more
[✓ e)] The symbol ? means 0 or 1
f) The symbol ? means 0 or more
g) The rule will parse successfully the parameter list (int x int y)
[✓ h)] The rule will parse successfully the parameter list (int x, int y)

3) XML code is structured as a:

a) Stack

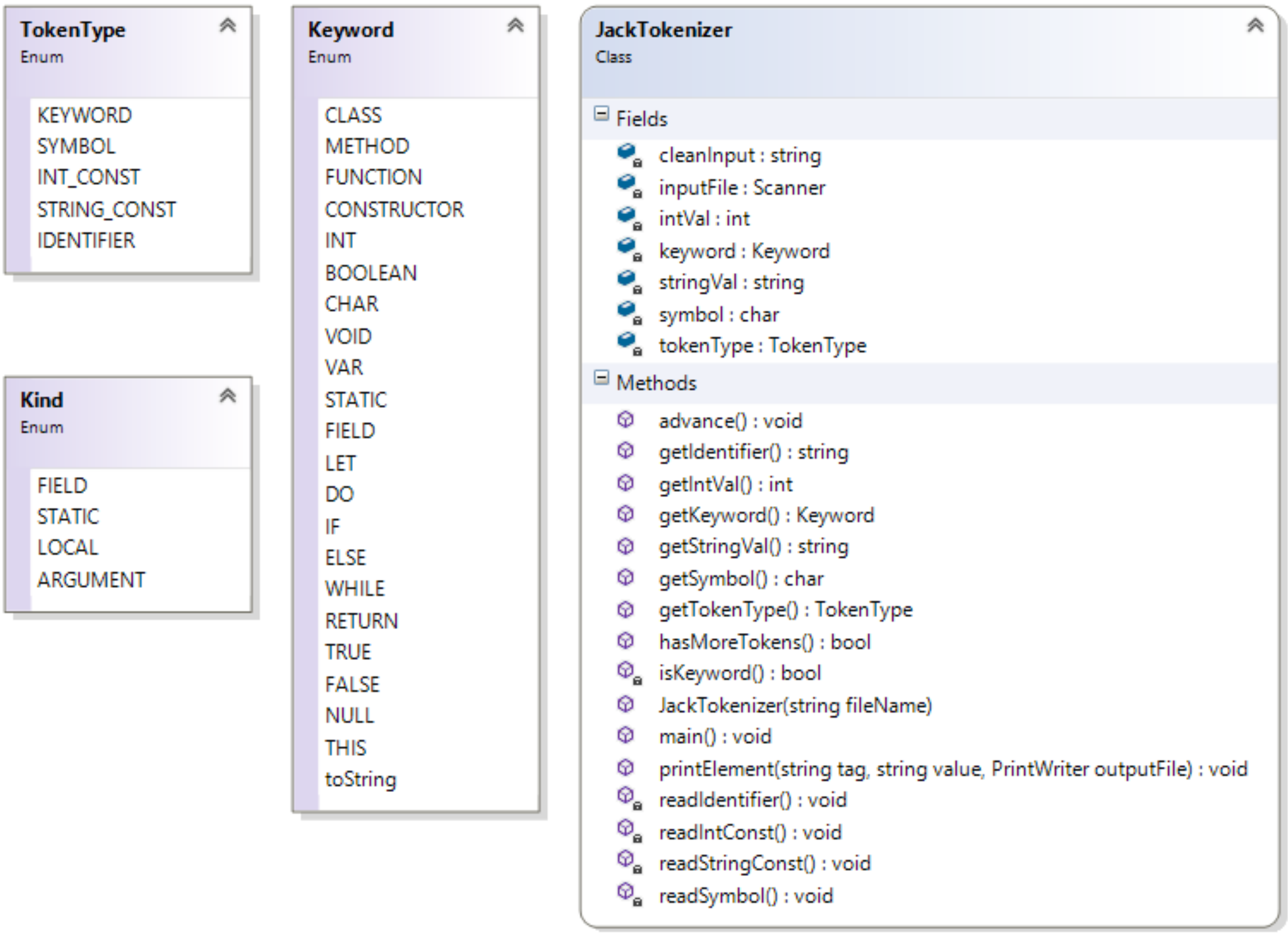
b) Graph

[✓ c) Tree]

d) Multi-dimensional array

Programming Exercise:

Using Java, complete the JackTokenizer class (JackTokenizer.java, part of the overall compiler) and implement the following enums (TokenType, Kind and Keyword). You may use the following API design to help guide your work, or choose to implement your own design.

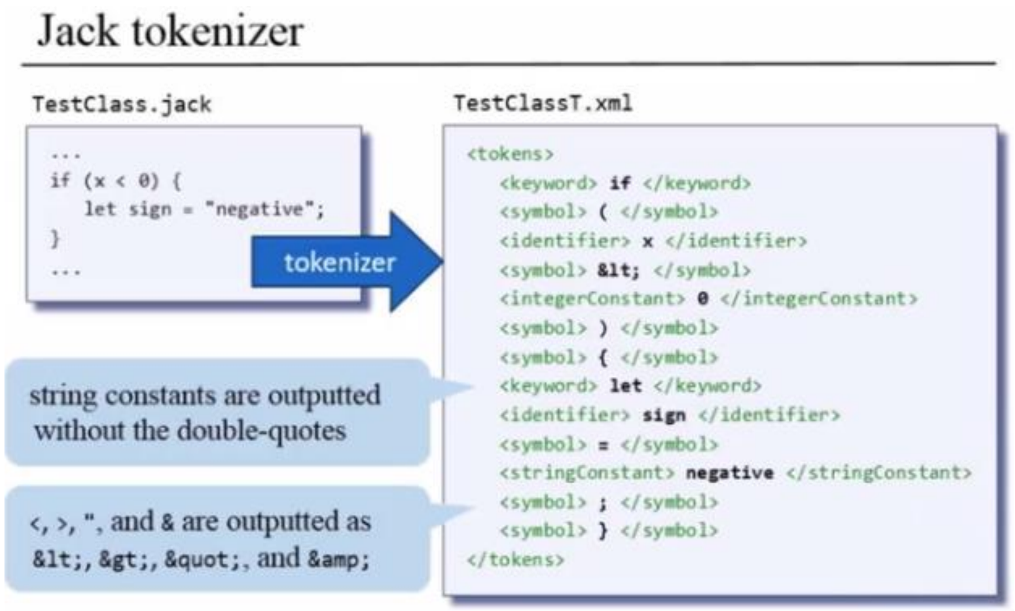


Following is an explanation of each of the methods in the JackTokenizer class. Please note the methods **keyword(), intVal(), stringVal(), symbol(), tokenType()** are really accessor (getter) methods, thus they have been renamed to **getKeyword(), getIntVal(), getStringVal(), getSymbol() and getTokenType()** accordingly in the class diagram above.

The main method of the JackTokenizer should prompt the user to enter the name of a Jack file (e.g. SquareDance.jack), then the program will produce an XML output file of all the tokens in the file with the extension xxxT.xml (e.g. SquareDanceT.xml).

You can use the TextComparer.bat (or .sh) supplied with the nand2tetris software to test your output against the solution (attached to this lab assignment on Canvas).

Here's a sample input/output of the Jack Tokenizer:



Jack tokenizer

TestClass.jack

```
...
if (x < 0) {
   let sign = "negative";
}
...
```

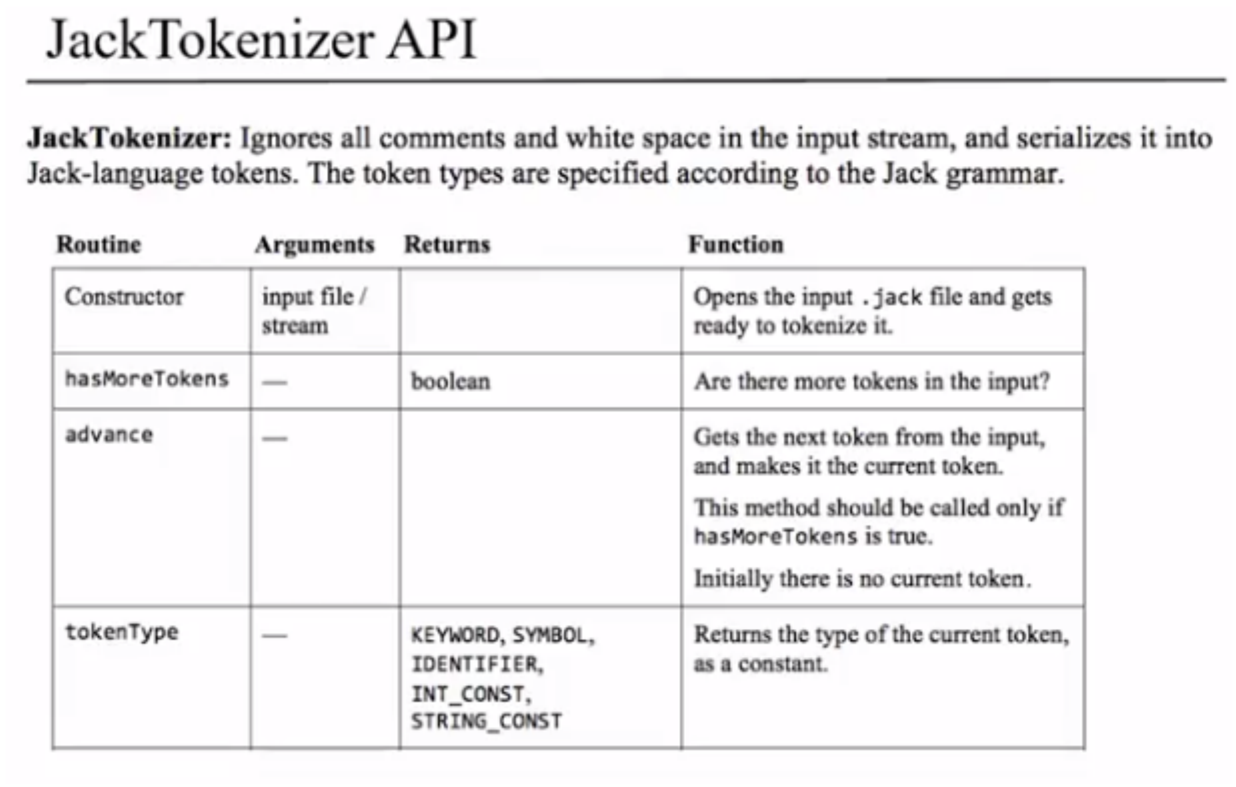
string constants are outputted without the double-quotes

<, >, ", and & are outputted as &lt;, &gt;, &quot;, and &amp;

TestClassT.xml

```
<tokens>
   <keyword> if </keyword>
   <symbol> ( </symbol>
   <identifier> x </identifier>
   <symbol> &lt; </symbol>
   <integerConstant> 0 </integerConstant>
   <symbol> ) </symbol>
   <symbol> { </symbol>
   <keyword> let </keyword>
   <identifier> sign </identifier>
   <symbol> = </symbol>
   <stringConstant> negative </stringConstant>
   <symbol> ; </symbol>
   <symbol> } </symbol>
</tokens>
```

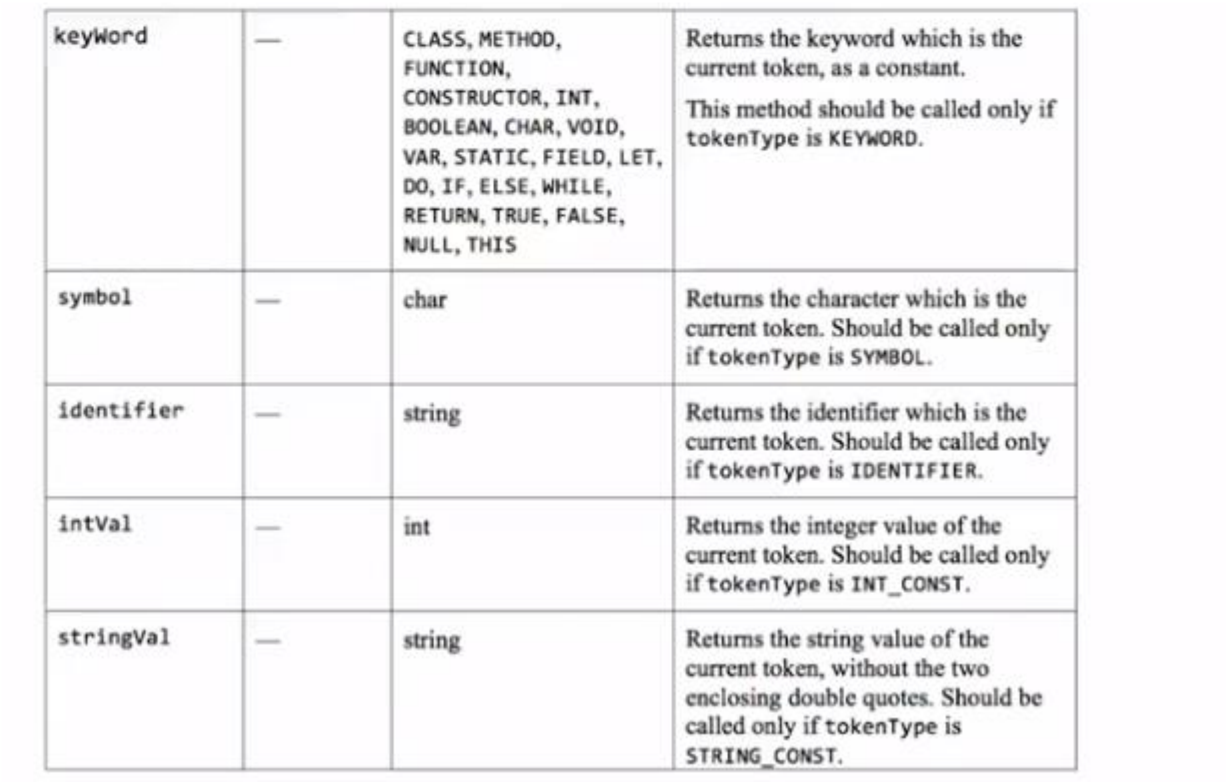
Here's the provided API for the Jack Tokenizer (also on Helper Sheet):



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`.

When you're finished, please upload a single Word document (or PDF) containing the following in order on Canvas:

1. The .xml files from your 2 sample programs (SquareDanceT.xml and ArrayMain.xml)
2. The source code of your JackTokenizer
3. The source code of each of your enums (TokenType, Kind and Keyword)