

Approach taken to create the parser:

I first began by creating an input/output class, the job of which was reading the entire input .jack file line by line, into an array of strings, where each element of the array contained a line of the input file. I passed in the path of the file into my InOut.readfile function, which I declared as static, so that I did not have to instantiate an object of this class. I used general code from StackOverFlow for the read file function, which at the end of the reading process, returned the array of strings with each line of the input being one element of the array.

I then created a Tokenizer class, which had four main entities:

1. Breakline function – which took in a string (so that I could pass in each element of the string array that I created, each containing one line of the original document). The job of this function was to divide up the line into segments, and then saving each of these segments of each line as an element into another array.

This function is where I faced the most difficulties, and it is therefore also the function which does the most work!

The function divided up each line based on whether each part of each line (each part was identified when a blank space or a symbol was reached) began with a ‘ “ ‘ (string), a symbol, or not a symbol (therefore a letter or a number). Each time a white space or a symbol was reached, the portion of the line up to that space or symbol was saved into the static array described below, and the remaining portion of the line was passed back into the breakline function recursively, where the rest of the line was parsed/broken down recursively until the base case (size of the line passed in == 0) was hit, in which case it returned.

I maintained an index for the line passed into the function, so that I could break the line according to how much of it I have parsed, and how much of it was left to pass, and hence pass back into the function recursively. Each time I hit a white space, I also incremented the index by 1 before passing in the remaining portion of the line into the function again, so that the white space didn't have to be read. Also since we were supposed to ignore comments for the output, I checked for a comment in the file, by making another nested if statement every time I hit a symbol. If the initial symbol I hit, and the one after it were both ‘ / ’, or one was ‘ / ’ and the one after was ‘ * ’, or the first one was ‘ * ’ and the next was simply a while space, I ignored that whole line by simply returning from that function, and passing the next line back into the function. This function was called in the main once for every line in my initial array of strings containing each line of the file I read.

2. Total static string array – This array contained the parsed/divided up portion of each line from the breakline function, and each portion was saved as an element of this string array. It was declared static so that when the breakline function was called recursively, the strings obtained from the functions were still added to the same array, and a new array did not have to be created for each instance of the breakline function.
3. The returndata function – Which simply returned the static array created in the breakline method, to the main, so that it can be used for the next function.
4. The gettoken function – Once the breakline function was implemented, this one came more naturally. For this function, I just called it in the main, and passed in each element of the parsed static string array that I created in the breakline function. These element were therefore already containing all the divided up chunks of each line, based on whether they were symbols, identifiers, keywords, integers or constants at this point. All I did in this function therefore was add the beginning and ending tags for each parsed entity, with the entity between the tags. If the string element passed in was a symbol for example, the function would return: `("<symbol>" + token + "</symbol>");`
Similarly, if the element was one of the keywords, it returned keyword tags. If it started with a number and contained all numbers, it returned integer tags. If it begun and ended with ' " ', it returned string tags, and finally, if it begun with a letter, and contained all letters or numbers, it returned identifier tags. This function was called in the main iteratively for each parsed element of my previous array, and the tags with the element in between that this function returned was then saved as an element into another array in the main, until tags were returned for each element of the previous parsed array. Now my final array containing all the data, with their respective tags were created and in the array called final_data in the main.

I finally passed my final_data string array into the InOut.writetofile function, which created a file named as the string passed into it from main as well, and wrote each element of the array into a separate line in that file, which saved into the project folder.

P.S: I made the Package class to accommodate two strings, in case I needed to return two strings from the breakline function, one that has been parsed and one that is yet to parse, but I never needed to return anything from that function later, since I just kept adding the parsed strings into the static array. This package object was therefore never really of any use.

Challenges faced:

Maintaining indexes – This was the biggest nightmare in the breakline function, and I had to spend hours debugging. The issue was that I have many if and else if statements in that function, and initially some strings were falling through the statements and meeting many criteria at once, therefore I had to strengthen each if statement. Then came the problem of indexes. Since my entire function depended on breaking each line of the input into parsed and unparsed strings, and passing the unparsed string back into the function, I had to maintain tight indexing to break the line effectively. This was difficult because sometimes the index would go out of scope at many of these line breaks, and threw massive errors. After many hours though, I saw a dim light at the end of the tunnel.

The beginning spaces of the file – This was actually the reason that the indexes were breaking I realized later. The file has multiple spaces/tabs, and sometimes a mixture of both for some lines. The file reader read this normally, but now each element of my initial array contained a lot of blank spaces before the actual data. This was an issue because these blank spaces were something other than just a white space somehow, and I could not describe that in an if statement (even null wouldn't work), and if I tried to eliminate the spaces by checking for them when reading the file, the file reader broke and threw an out of scope exception. Obviously I had no idea what to do, but after much googling I realized that there was a built in function in java for the string class called trim(), which automatically removes any blank spaces before or after characters! So I used this to trim each line when reading the file, and then my indexes for the breakline function stopped going out of scope as well! I am not entirely sure if we can do this, and if the .xml file I made from the output function will still work like this, but fingers crossed!

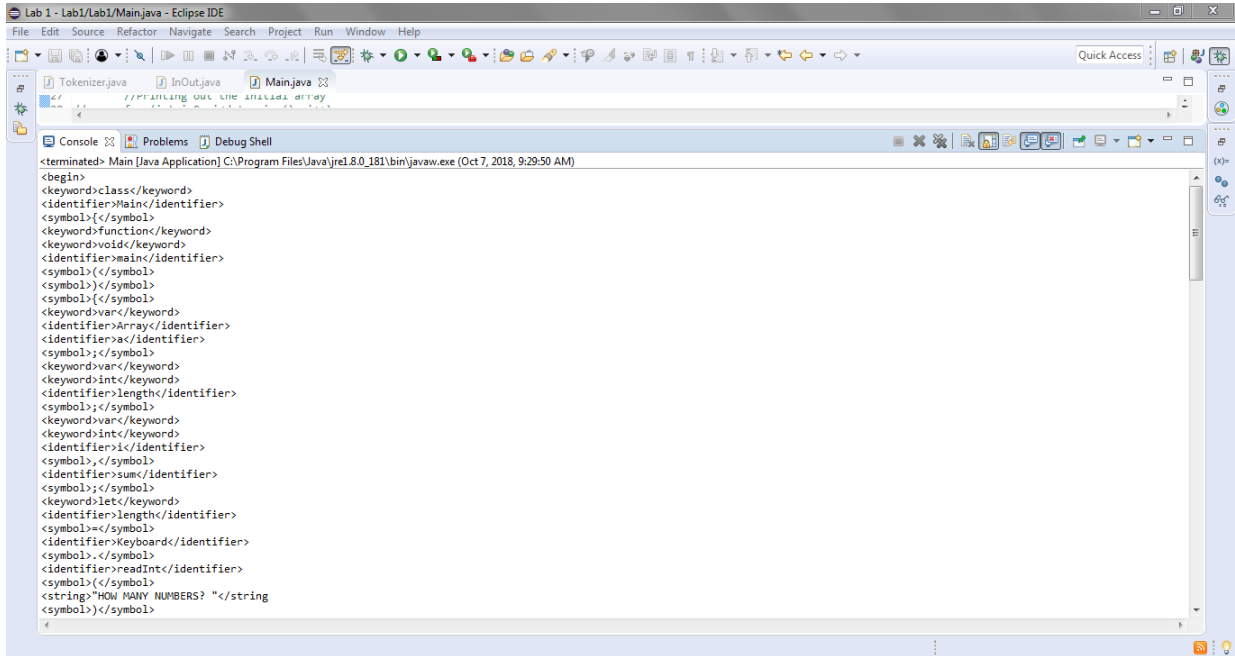
Key take-aways:

The lab definitely wasn't as easy as I expected it to be initially! Reading files from a line is far more complicated when one has to identify the spaces before after, and in between, and symbols, numbers, characters, and any combinations of all three. This is why my if statements are massive and I do apologize (later I found that JAVA also has built in functions for checking if a character is a letter or number! By this time I was done of course. .).

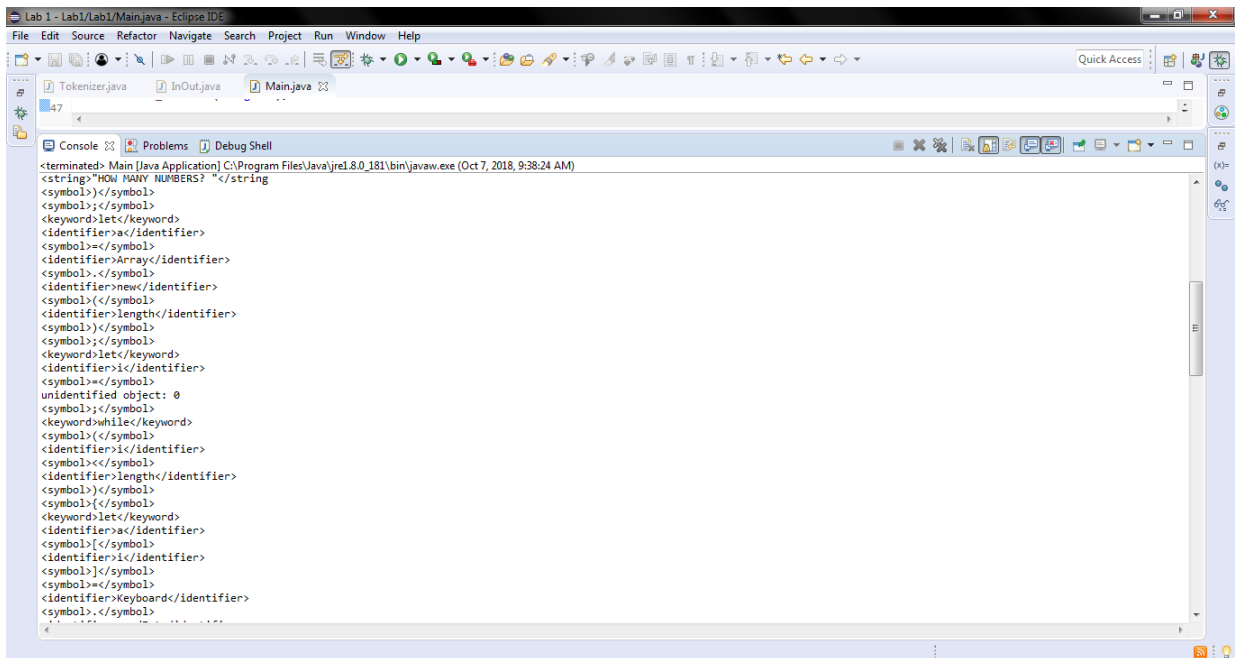
I now do however understand the tasks of a tokenizer much better, and how every line of code we type is analyzed behind the scenes for the machine to further decode and understand. I also now have a better understanding of how parse trees work, and how keyword tables and the kind that we discussed in class, would actually work. In short I learned a lot after many hours of frustrated debugging!

Screenshots:

ArrayTest - Main :



```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Oct 7, 2018, 9:29:50 AM)
<begin>
<keyword>class</keyword>
<identifier>Main</identifier>
<symbol>{</symbol>
<keyword>function</keyword>
<keyword>void</keyword>
<identifier>main</identifier>
<symbol>(</symbol>
<symbol></symbol>
<symbol>{</symbol>
<keyword>var</keyword>
<identifier>Array</identifier>
<identifier>a</identifier>
<symbol>;</symbol>
<keyword>var</keyword>
<keyword>int</keyword>
<identifier>length</identifier>
<symbol>;</symbol>
<keyword>var</keyword>
<keyword>int</keyword>
<identifier>i</identifier>
<symbol>,</symbol>
<identifier>sum</identifier>
<symbol>;</symbol>
<keyword>let</keyword>
<identifier>length</identifier>
<symbol>=</symbol>
<identifier>Keyboard</identifier>
<symbol>.</symbol>
<identifier>readInt</identifier>
<symbol>(</symbol>
<string>"HOW MANY NUMBERS? "</string>
<symbol>)</symbol>
```



```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Oct 7, 2018, 9:38:24 AM)
<string>"HOW MANY NUMBERS? "</string>
<symbol></symbol>
<symbol>;</symbol>
<keyword>let</keyword>
<identifier>a</identifier>
<symbol>=</symbol>
<identifier>Array</identifier>
<symbol>.</symbol>
<identifier>new</identifier>
<symbol>(</symbol>
<identifier>length</identifier>
<symbol>)</symbol>
<symbol>;</symbol>
<keyword>let</keyword>
<identifier>i</identifier>
<symbol>=</symbol>
<symbol>unidentified object: 0
<symbol>;</symbol>
<keyword>while</keyword>
<symbol>(</symbol>
<identifier>i</identifier>
<symbol><</symbol>
<identifier>length</identifier>
<symbol>)</symbol>
<symbol>{</symbol>
<keyword>let</keyword>
<identifier>a</identifier>
<symbol>{</symbol>
<identifier>i</identifier>
<symbol>}</symbol>
<symbol>.</symbol>
<identifier>Keyboard</identifier>
<symbol>.</symbol>
```

Lab 1 - Lab1/Main.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Tokenizer.java InOut.java Main.java

47

Console Problems Debug Shell

```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Oct 7, 2018, 9:38:24 AM)
<identifier>keyboard</identifier>
<symbol>.</symbol>
<identifier>readInt</identifier>
<symbol>(</symbol>
<string>"ENTER THE NEXT NUMBER: "</string>
<symbol>)</symbol>
<symbol>.</symbol>
<keyword>let</keyword>
<identifier>i</identifier>
<symbol>=</symbol>
<identifier>i</identifier>
<symbol>+</symbol>
<integer>i</integer>
<symbol>.</symbol>
<symbol>.</symbol>
<keyword>let</keyword>
<identifier>i</identifier>
<symbol>=</symbol>
unidentified object: 0
<symbol>.</symbol>
<keyword>let</keyword>
<identifier>sum</identifier>
<symbol>=</symbol>
unidentified object: 0
<symbol>.</symbol>
<keyword>while</keyword>
<symbol>(</symbol>
<identifier>i</identifier>
<symbol><</symbol>
<identifier>length</identifier>
<symbol>.</symbol>
<symbol>.</symbol>
<symbol>{</symbol>
<keyword>let</keyword>
```

Lab 1 - Lab1/Main.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

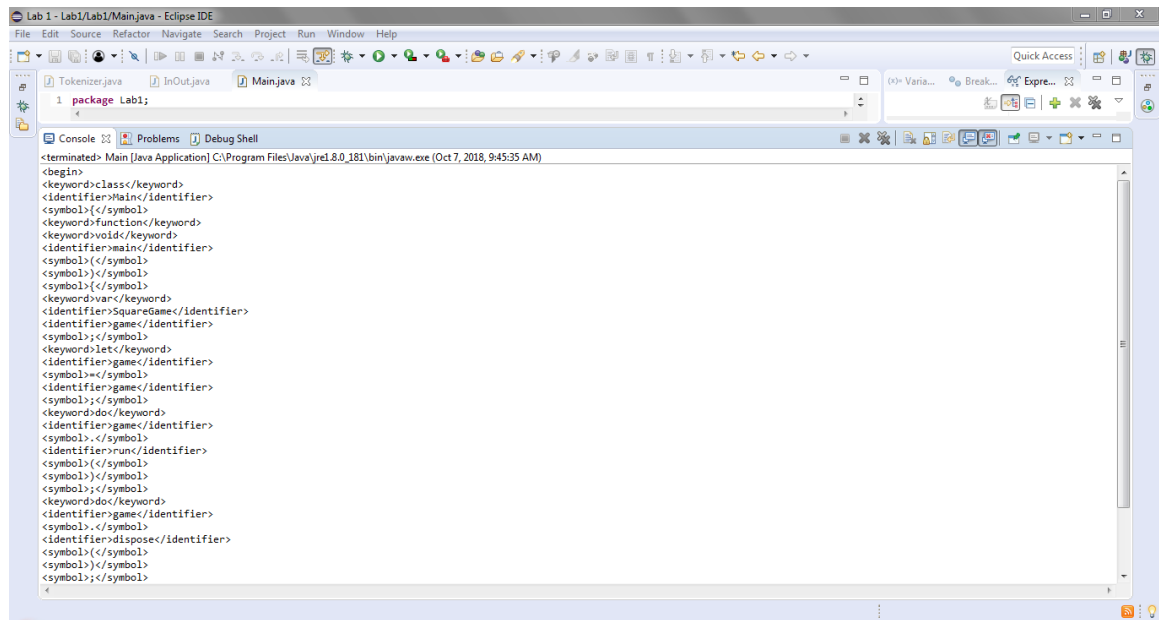
Tokenizer.java InOut.java Main.java

47

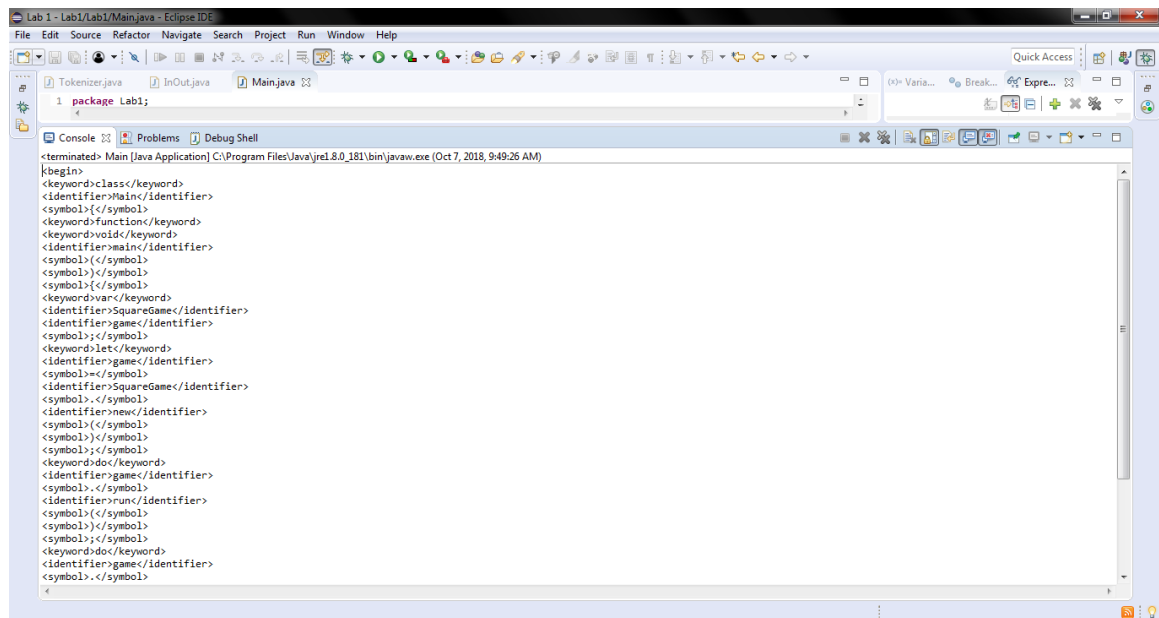
Console Problems Debug Shell

```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Oct 7, 2018, 9:38:24 AM)
<symbol>.</symbol>
<symbol>.</symbol>
<keyword>do</keyword>
<identifier>Output</identifier>
<symbol>.</symbol>
<identifier>printString</identifier>
<symbol>(</symbol>
<string>"THE AVERAGE IS: "</string>
<symbol>.</symbol>
<symbol>.</symbol>
<keyword>do</keyword>
<identifier>Output</identifier>
<symbol>.</symbol>
<identifier>printInt</identifier>
<symbol>(</symbol>
<identifier>sum</identifier>
<symbol>.</symbol>
<identifier>length</identifier>
<symbol>.</symbol>
<symbol>.</symbol>
<keyword>do</keyword>
<identifier>Output</identifier>
<symbol>.</symbol>
<identifier>println</identifier>
<symbol>(</symbol>
<symbol>.</symbol>
<symbol>.</symbol>
<keyword>return</keyword>
<symbol>.</symbol>
<symbol>.</symbol>
<symbol>.</symbol>
</end>
```

ExpressionlessSquare – Main:



Square – Main:



Actual output from output file:

ArrayTest - Main:

```
<begin>
<keyword>class</keyword>
<identifier>Main</identifier>
<symbol>{</symbol>
<keyword>function</keyword>
<keyword>void</keyword>
<identifier>main</identifier>
<symbol>(</symbol>
<symbol>)</symbol>
<symbol>{</symbol>
<keyword>var</keyword>
<identifier>Array</identifier>
<identifier>a</identifier>
<symbol>;</symbol>
<keyword>var</keyword>
<keyword>int</keyword>
<identifier>length</identifier>
<symbol>;</symbol>
<keyword>var</keyword>
<keyword>int</keyword>
<identifier>i</identifier>
<symbol>,</symbol>
<identifier>sum</identifier>
<symbol>;</symbol>
<keyword>let</keyword>
<identifier>length</identifier>
<symbol>=</symbol>
<identifier>Keyboard</identifier>
<symbol>.</symbol>
<identifier>readInt</identifier>
<symbol>(</symbol>
<string>"HOW MANY NUMBERS? "</string>
<symbol>)</symbol>
<symbol>;</symbol>
<keyword>let</keyword>
<identifier>a</identifier>
<symbol>=</symbol>
<identifier>Array</identifier>
```

<symbol>.</symbol>
<identifier>new</identifier>
<symbol>(</symbol>
<identifier>length</identifier>
<symbol>)</symbol>
<symbol>;</symbol>
<keyword>let</keyword>
<identifier>i</identifier>
<symbol>=</symbol>
unidentified object: 0
<symbol>;</symbol>
<keyword>while</keyword>
<symbol>(</symbol>
<identifier>i</identifier>
<symbol><</symbol>
<identifier>length</identifier>
<symbol>)</symbol>
<symbol>{</symbol>
<keyword>let</keyword>
<identifier>a</identifier>
<symbol>[</symbol>
<identifier>i</identifier>
<symbol>]</symbol>
<symbol>=</symbol>
<identifier>Keyboard</identifier>
<symbol>.</symbol>
<identifier>readInt</identifier>
<symbol>(</symbol>
<string>"ENTER THE NEXT NUMBER: "</string>
<symbol>)</symbol>
<symbol>;</symbol>
<keyword>let</keyword>
<identifier>i</identifier>
<symbol>=</symbol>
<identifier>i</identifier>
<symbol>+</symbol>
<integer>1</integer>
<symbol>;</symbol>
<symbol>}</symbol>
<keyword>let</keyword>
<identifier>i</identifier>
<symbol>=</symbol>
unidentified object: 0
<symbol>;</symbol>


```
<keyword>let</keyword>
<identifier>sum</identifier>
<symbol>=</symbol>
unidentified object: 0
<symbol>;</symbol>
<keyword>while</keyword>
<symbol>(</symbol>
<identifier>i</identifier>
<symbol><</symbol>
<identifier>length</identifier>
<symbol>)</symbol>
<symbol>{</symbol>
<keyword>let</keyword>
<identifier>sum</identifier>
<symbol>=</symbol>
<identifier>sum</identifier>
<symbol>+</symbol>
<identifier>a</identifier>
<symbol>[</symbol>
<identifier>i</identifier>
<symbol>]</symbol>
<symbol>;</symbol>
<keyword>let</keyword>
<identifier>i</identifier>
<symbol>=</symbol>
<identifier>i</identifier>
<symbol>+</symbol>
<integer>1</integer>
<symbol>;</symbol>
<symbol>}</symbol>
<keyword>do</keyword>
<identifier>Output</identifier>
<symbol>.</symbol>
<identifier>printString</identifier>
<symbol>(</symbol>
<string>"THE AVERAGE IS: "</string>
<symbol>)</symbol>
<symbol>;</symbol>
<keyword>do</keyword>
<identifier>Output</identifier>
<symbol>.</symbol>
<identifier>printInt</identifier>
<symbol>(</symbol>
<identifier>sum</identifier>
```

```

<symbol>/</symbol>
<identifier>length</identifier>
<symbol>)</symbol>
<symbol>;</symbol>
<keyword>do</keyword>
<identifier>Output</identifier>
<symbol>.</symbol>
<identifier>println</identifier>
<symbol>(</symbol>
<symbol>)</symbol>
<symbol>;</symbol>
<keyword>return</keyword>
<symbol>;</symbol>
<symbol>}</symbol>
<symbol>}</symbol>
</end>

```

ExpressionlessSquare – Main:

```

<begin>
<keyword>class</keyword>
<identifier>Main</identifier>
<symbol>{</symbol>
<keyword>function</keyword>
<keyword>void</keyword>
<identifier>main</identifier>
<symbol>(</symbol>
<symbol>)</symbol>
<symbol>{</symbol>
<keyword>var</keyword>
<identifier>SquareGame</identifier>
<identifier>game</identifier>
<symbol>;</symbol>
<keyword>let</keyword>
<identifier>game</identifier>
<symbol>=</symbol>
<identifier>game</identifier>
<symbol>;</symbol>
<keyword>do</keyword>
<identifier>game</identifier>
<symbol>.</symbol>
<identifier>run</identifier>
<symbol>(</symbol>
<symbol>)</symbol>

```

```
<symbol>;</symbol>
<keyword>do</keyword>
<identifier>game</identifier>
<symbol>.</symbol>
<identifier>dispose</identifier>
<symbol>(</symbol>
<symbol>)</symbol>
<symbol>;</symbol>
<keyword>return</keyword>
<symbol>;</symbol>
<symbol>}</symbol>
<symbol>}</symbol>
</end>
```

Square – Main:

```
<begin>
<keyword>class</keyword>
<identifier>Main</identifier>
<symbol>{</symbol>
<keyword>function</keyword>
<keyword>void</keyword>
<identifier>main</identifier>
<symbol>(</symbol>
<symbol>)</symbol>
<symbol>{</symbol>
<keyword>var</keyword>
<identifier>SquareGame</identifier>
<identifier>game</identifier>
<symbol>;</symbol>
<keyword>let</keyword>
<identifier>game</identifier>
<symbol>=</symbol>
<identifier>SquareGame</identifier>
<symbol>.</symbol>
<identifier>new</identifier>
<symbol>(</symbol>
<symbol>)</symbol>
<symbol>;</symbol>
<keyword>do</keyword>
<identifier>game</identifier>
<symbol>.</symbol>
<identifier>run</identifier>
<symbol>(</symbol>
```

<symbol>)</symbol>
<symbol>;</symbol>
<keyword>do</keyword>
<identifier>game</identifier>
<symbol>.</symbol>
<identifier>dispose</identifier>
<symbol>(</symbol>
<symbol>)</symbol>
<symbol>;</symbol>
<keyword>return</keyword>
<symbol>;</symbol>
<symbol>}</symbol>
<symbol>}</symbol>
</end>