

Japper Manual

AUTHORS

Norielle Aguila and Jasper Pillejera

WHAT IS JAPPER?

Japper is a case-insensitive language based off of the Java grammar that takes inspiration from Assembly language and modern slang. This means that you can tYPe liKe ThiS anD JAPPeR wiLL sTiLL unDeRStand YOu. All keywords in Japper is case insensitive; however, variables are still case sensitive.

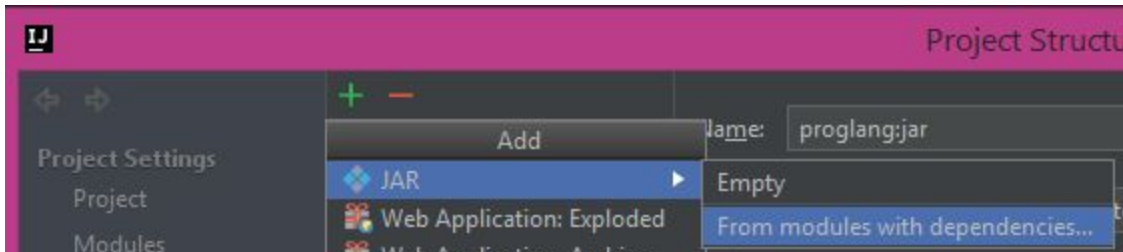
INSTALLATION

Japper comes with two projects, the interpreter (proglang folder) and the IDE (JapperParser folder).

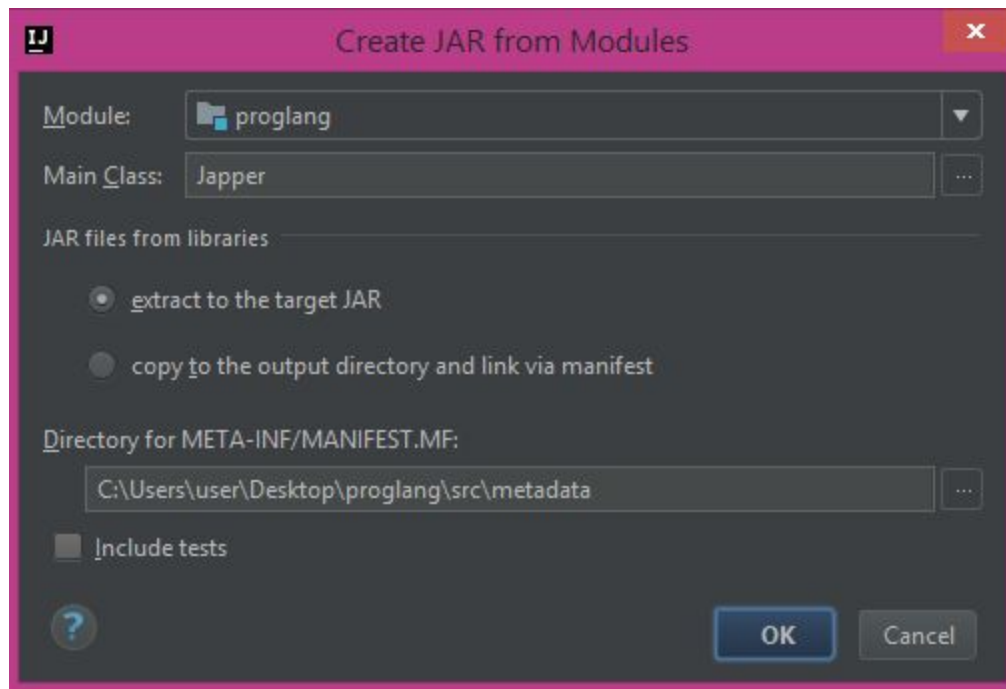
JAPPER

To install the interpreter, open the project using IntelliJ.

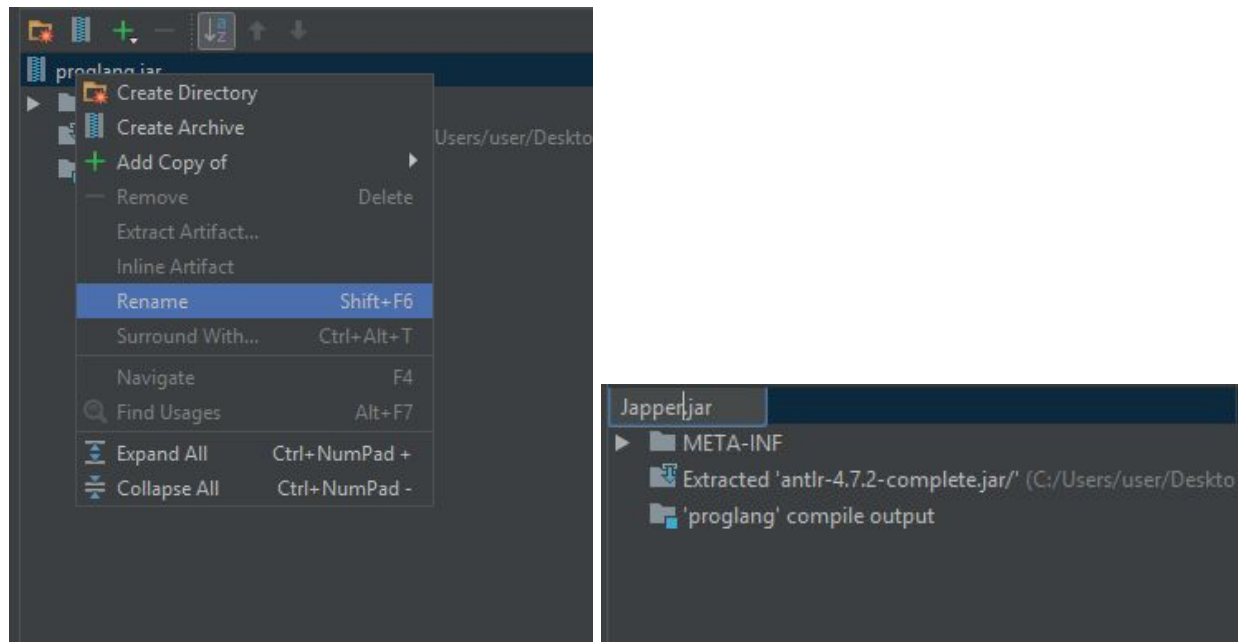
- 1 - Open project structures and add a new JAR file from modules with dependencies.



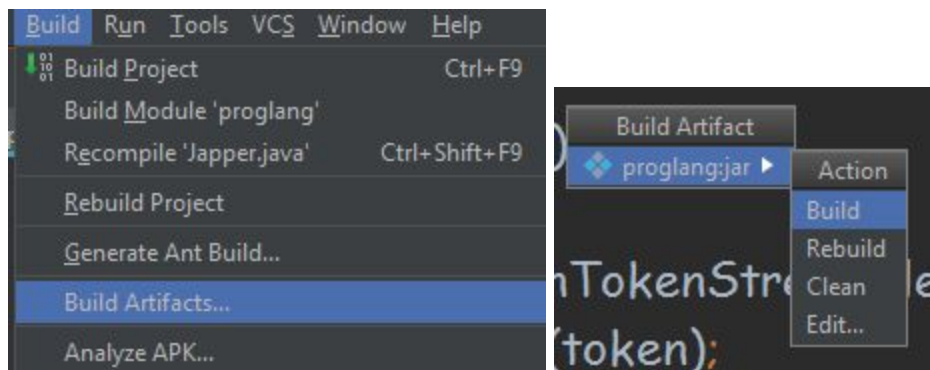
- 2 - Set the main class as Japper and set the META-INF directory.



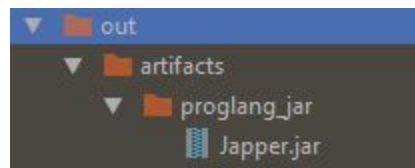
3 - Rename the Output file as Japper.jar



4 - Build the jar file.



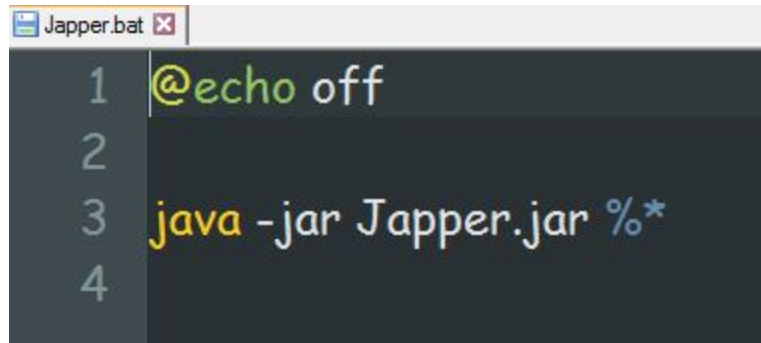
5 - This will then Output the jar file.



Using the Japper.jar file we can now use it to run japper programs using

```
>java -jar Japper.jar file1.japper
```

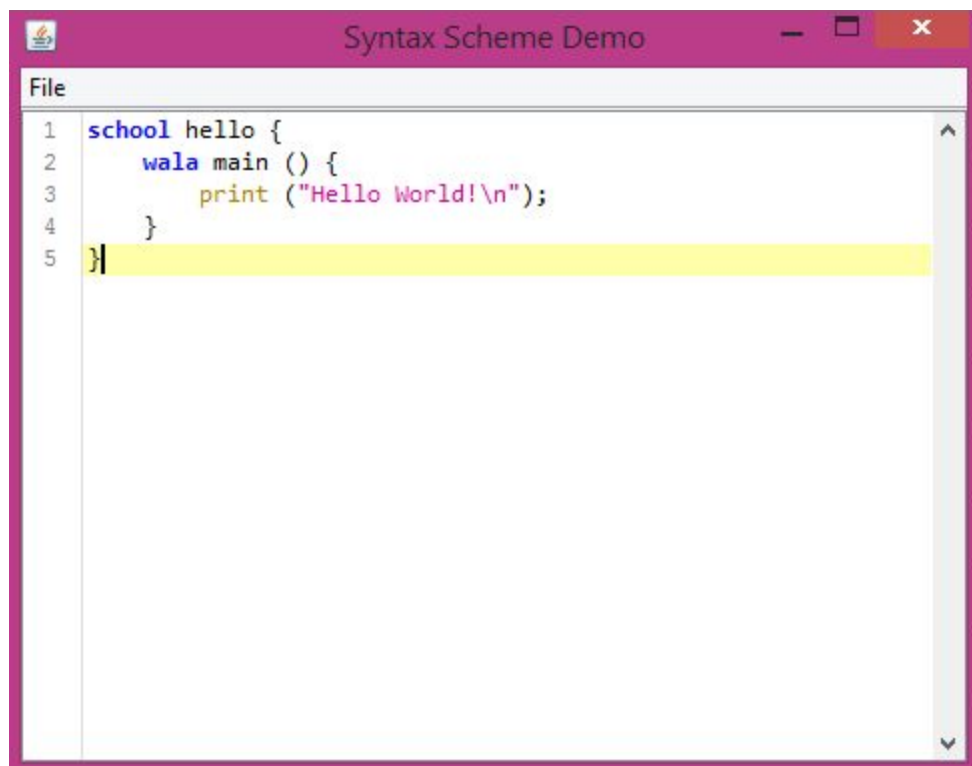
To further automate running Japper programs, we can create a batch file containing

A screenshot of a Windows batch file named 'Japper.bat'. The file contains four lines of code: 1. '@echo off', 2. (blank line), 3. 'java -jar Japper.jar %*', and 4. (blank line). The lines are numbered 1 through 4 on the left side of the editor.

```
1 @echo off
2
3 java -jar Japper.jar %*
4
```

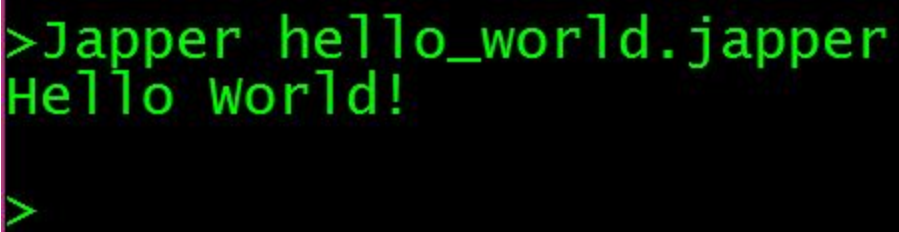
Using this batch file, we can now run programs using Japper file.japper.

Here is an example of the Hello world program written in Japper.

A screenshot of a Japper program named 'hello.japper' in a text editor titled 'Syntax Scheme Demo'. The code is written in a syntax-highlighted style and consists of five lines: 1. 'school hello {', 2. ' wala main () {', 3. ' print ("Hello World!\n");', 4. ' }', and 5. '}'. The fifth line is highlighted in yellow.

```
1 school hello {
2     wala main () {
3         print ("Hello World!\n");
4     }
5 }
```

Running the hello_world.japper file in the sample files in the input directory gives this result.

A terminal window with a black background and green text. The first line shows the command ">Japper hello_world.japper" followed by the output "Hello World!". The second line shows a new prompt ">".

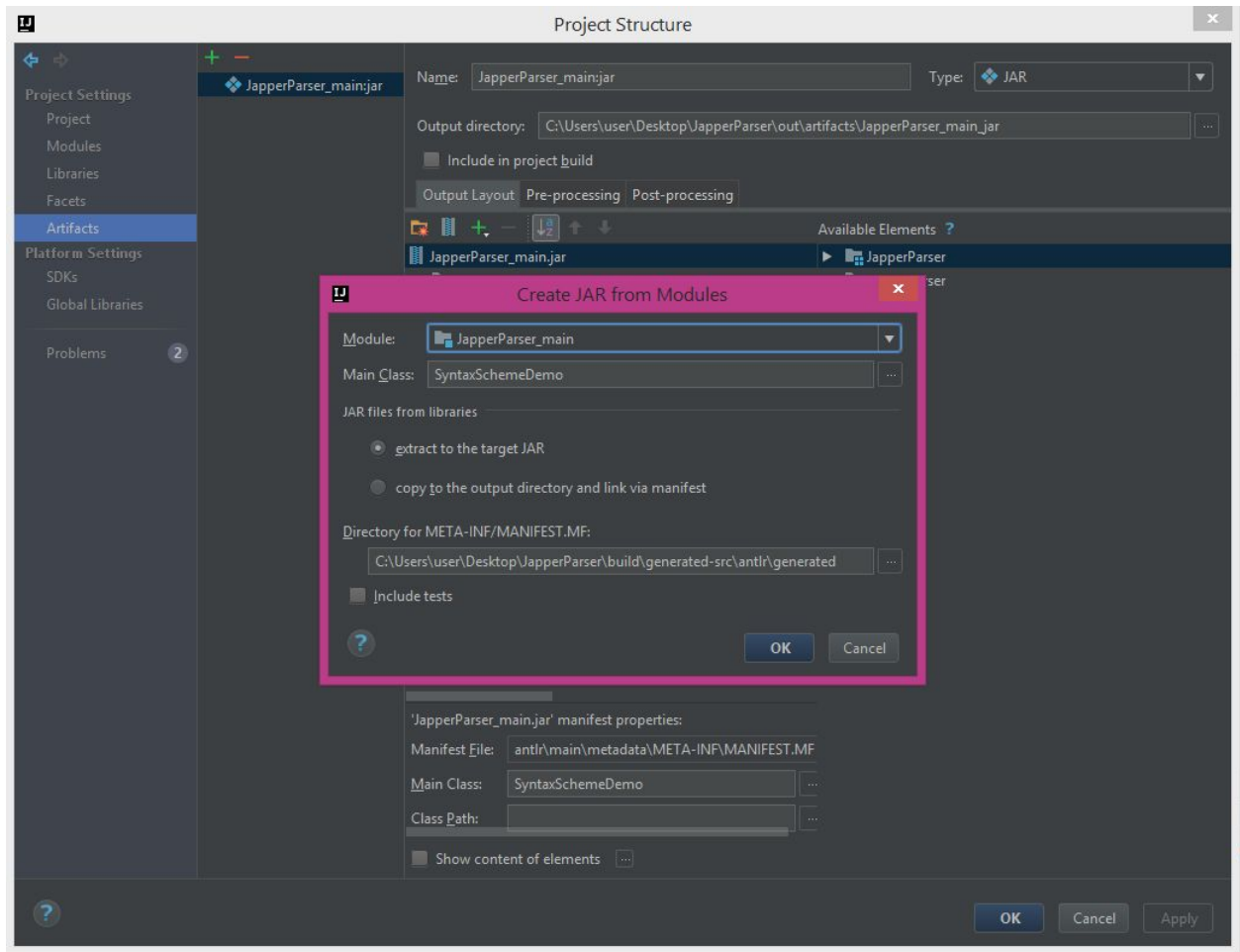
```
>Japper hello_world.japper  
Hello World!  
  
>
```

IDE

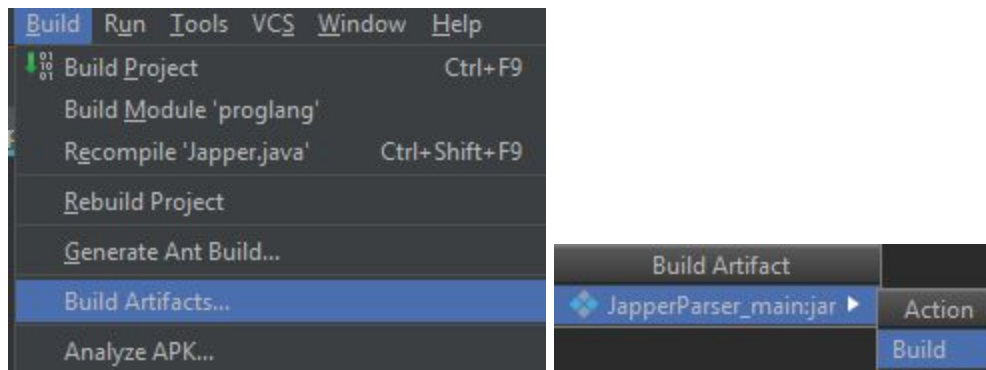
Japper comes with its own IDE with syntax highlighting and file management features.

To build the IDE, open the project using IntelliJ.

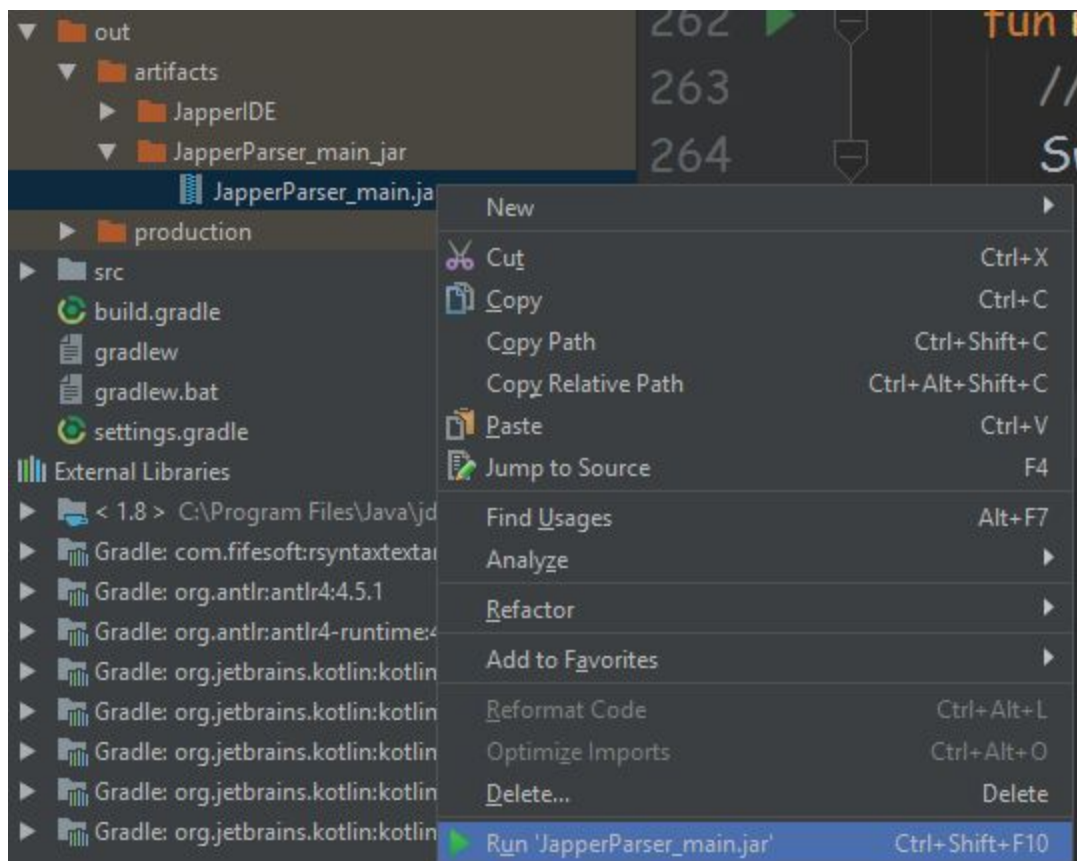
1 - Set the project structure to create a jar file with SyntaxSchemeDemo as the Main class.



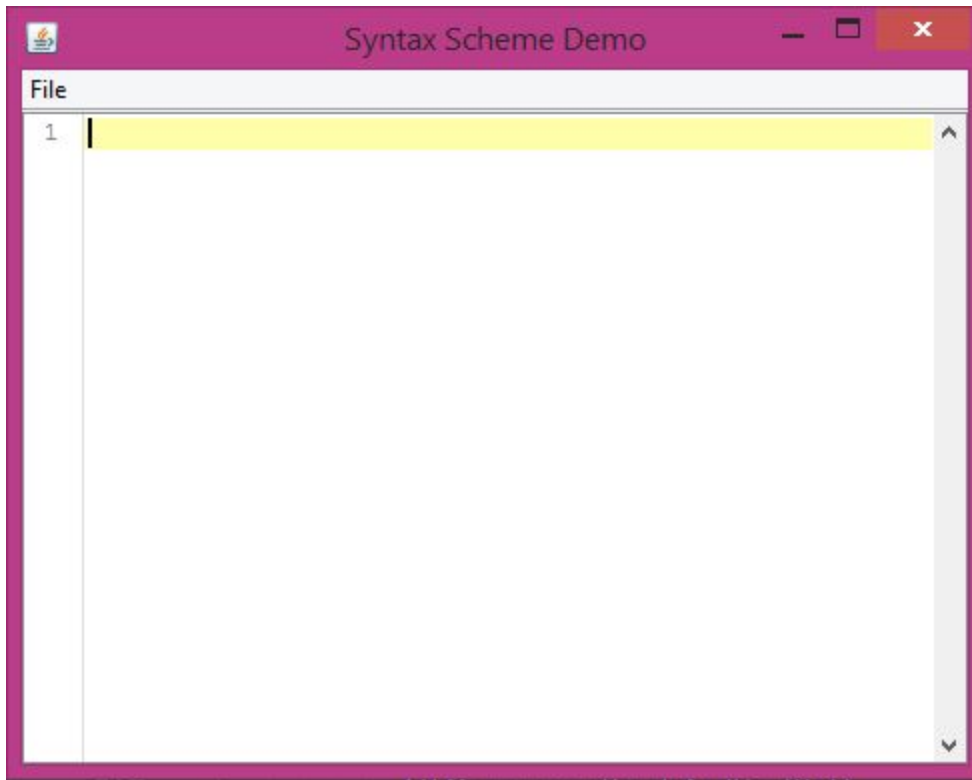
2 - Build the jar file.



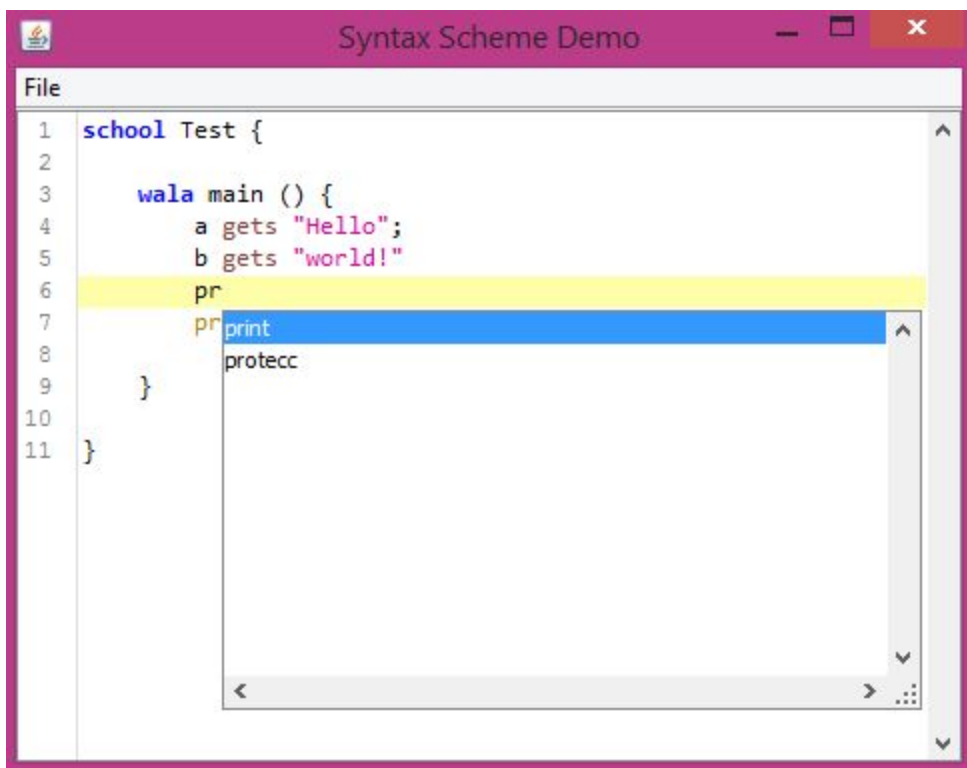
3 - Run the jar file



A new window will appear where you can write japper programs!



Syntax Scheme Demo is capable of syntax highlighting and autocompletion for Japper!



Usage

COMMENTS

Comments make use of the symbols `//` and `/* <text> */`

EXAMPLE

```
1 // This is a comment
2 /*
3 This is a multi-line comment
4 */
```

DATA TYPES

Japper makes use of the same primitive data types as Java, but with different keywords. However, Japper still recognizes Java keywords.

Japper data type	Java equivalent
f32	float
f64	double
i1	boolean
i2	
i8	byte
i16	short
i32	int
i64	long
char	char
wala	void

i2 is a fun feature to Japper data types wherein the values are in base 4. It is possible to create a variable and assign an A = 0, C = 1, G = 2, T = 3 sequence and the value will be represented in 2 bits per character.

```

1 school Test {
2
3     wala main () {
4         a gets 0qACGT;
5
6         print (a);
7
8     }
9
10 }

```

```

>Japper test.japper
27
>

```

Note that writing the data type is only required when declaring functions return types and function parameters.

CONSTANT & VARIABLE ASSIGNMENT

Japper makes use of type inference, meaning variable types are decided once values are assigned. Due to this, variables cannot be declared with no values. Assignment uses the keyword `gets`.

EXAMPLE

```

1 x gets 1;    // x is assigned the int value 1
2 y gets "Hello!";    // y is assigned the String value "Hello!"
3 last na var z gets 'c';    // z gets final char value 'c'

```

* Note that for final variables, the declaration requires the keywords 'last na var'

CLASS DECLARATION

Classes are treated as code wrappers in Japper. Each file must contain a class block, with a `wala main ()` block inside it. Classes use the keyword `school` and do not require an access modifier.

EXAMPLE

```

1 school sample {    // school can be spelled as sCho0l too
2     wala main () {
3         ...
4     }
5 }

```

MATHEMATICAL EXPRESSIONS

Expression	Description
1 plus 1	addition
1 minus 1	subtraction
1 times 1	multiplication
1 div 1	division
1 mod 1	modulo
1 eq 1	equality check
1 ne 1	non equality check
x plusEq y	x gets current value plus y
x minusEq y	x gets current value minus y
x timesEq y	x gets current value times y
x divEq y	x gets current value div y
x modEq y	x gets current value mod y
x lt y	less than
x gt y	greater than
x le y	less than equal
x ge y	greater than equal
x inc	increment
x dec	decrement

IF-ELSE STATEMENTS

If statements can use the following keywords: `if`, `kung`, `kungIf`, `thonking`, `thonkingIf`. Else still uses the keyword `else`.

EXAMPLE

```
1 kung (condition) {  
2     ...  
3 }  
4 else {  
5     ...  
6 }
```

LOOPS

Loops in Japper is simplified to just the keyword `loop`.

FOR LOOPS

EXAMPLE

```
1 loop (i gets 0; i lt 10; i inc) {  
2     print (i);  
3 }
```

WHILE LOOPS

EXAMPLE

```
1 i gets 0;  
2 loop (i lt 10) {  
3     print (i);  
4     i inc;  
5 }
```

DO-WHILE LOOPS

EXAMPLE

1	i gets 0;
2	loop {
3	print (i);
4	i inc;
5	} until (i lt 10);

BOOLEAN EXPRESSIONS

Boolean value	Japper equivalent
True	yes
False	no

INPUT STATEMENTS

Japper includes a built-in input function called `read`. It returns a whole line of String to whatever receives it. It can also be casted to other data types.

EXAMPLE

1	x gets (i32) read ();
2	y gets (f64) read ();

OUTPUT STATEMENTS

Japper includes a built-in input function called `print`. Strings and variables may be concatenated through the keyword `plus`.

EXAMPLE

1	print ("The sum is: " plus x);
---	--------------------------------

FUNCTIONS (NO PARAMETERS)

Functions in japper can have 0 to many parameters and it can return nothing or something. Code after a return statement (bbb) which will never execute is considered as dead code and will not be evaluated.

EXAMPLE

```
1 wala func (){
2     print ("in func\n");
3     bbb none;    // return null
4     print ("hello\n");    // unreachable code
5 }
```

Note that bbb can have any number of consecutive b's as long as there is a minimum of 2.

FUNCTIONS (WITH PARAMETERS)

Functions with parameters need to have their data type.

EXAMPLE

```
1 school Test {
2     i32 fib (i32 n) {
3         kung (n le 1) {
4             bbb n;
5         } else {
6             bbb fib (n minus 1) plus fib (n minus 2);
7         }
8     }
9
10    wala main () {
11        number gets (int) read ();
12        answer gets fib (number);
13        print ("The answer is " + answer);
```

14	}
15	}

TRY-CATCH BLOCKS

There are current 3 errors that can be caught by a try-catch block:

- Arithmetic Exception (AE) : zero division
- Index Out of Bounds (IOB)
- Null Pointer Exception

* currently not working on all cases

EXAMPLE

1	<code>school nanaman {</code>
2	<code> wala main () {</code>
3	<code> YeeET { // try</code>
4	<code> a gets 1 div 0;</code>
5	<code> } o00f (AE e) { // catch</code>
6	<code> print (e + "\n");</code>
7	<code> } ReeeE { // finally</code>
8	<code> print (4);</code>
9	<code> }</code>
10	<code> }</code>
11	<code>}</code>

```
>Japper.bat try_catch.japper
brodie an arithmetic exception occured. don't divide by 0 or something.
4
```