Lexical structure

white_space ::= SP | CR | LF | HT |…

These are unicode characters for space, carriage return, new line feed, and horizontal tab. These are represented in Java as ' ', '\r', '\n', '\t' etc. White space separates tokens, but is otherwise ignored in the input (except inside string literals). You may safely use the Character.isWhiteSpace() method to recognize white space for our language.

comment ::= // NOT(CR|LF)* eol
eol ::= CR | LF | CR LF | eof

Comments separate tokens but otherwise are ignored in the input. NOT*(CR|LF) matches any character except CR and LF. See the code for additional Unicode characters that indicate eol. Note that on some systems, a line is by default terminated with two characters represented in Java by \n\r. This is irrelevant to tokenizing, but does matter for computing line numbers. Your project should handle this properly.*

token ::= ident | keyword | int_literal | string_literal | boolean_literal | separator | operator

ident ::= ident_start ident_part* (but not keyword)
ident_start ::= A .. Z | a .. z | $ | _
ident_part ::= ident_start | (0 .. 9)

keyword ::= image | int | boolean | pixel | pixels | blue | red | green | Z | shape | width | height | location | x_loc | y_loc | SCREEN_SIZE | visible | x | y | pause | while | if | else

separator ::= . | ; | , | (|) | [|] | { | } | : | ?

operators ::= = | | | & | == | != | < | > | <= | >= | + | - | * | / | % | ! | << | >>

string_literal ::= " NOT(")* "

int_literal ::= 0 | ((1..9) (0..9)*)

boolean_literal ::= true | false

The definition of an int literal considers 01 and 123a4 each to be two tokens 0,1 (two int_literal) and 123, a4 (an int_literal and an identifier) respectively. This is allowed by the lexical structure, and so your scanner should behave this way. It is not allowed by the phrase structure of the phrase structure of the language, so we won't see that in actual programs.