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 use the Character.isWhiteSpace method to recognize white space for our language. In order to perform useful error checking, you will want to keep track of the current line number in the source. Depending on the system, lines can be terminated with LF (Line feed, '\n', 0x0A, 10 in decimal), or CR (Carriage return, '\r', 0x0D, 13 in decimal) individually, or CR followed by LF (CR+LF, '\r\n', 0x0D0A).

comment ::= /* NOT(*/)* */

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

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

You can use Character.isJavaIdentifierStart and Character.isJavaIdentifierPart

keyword ::= int | string | boolean | import | class | def | while | if | else | return | print

boolean_literal ::= true | false

null_literal ::= null

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

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

string_literal ::= " string_element* "
string_element ::= esc_sequence | NOT(")
esc_sequence ::= \(n | r | ")

The Scanner should consider the entire string, including the quotation marks as part of the token marked by beg and end. Escape characters will be handled in the getText method in the (provided) TokenStream class.

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

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.