```
/*
 * Skeleton code for your Lexer, provided by Emma Norling
 *
 * Please note that this code is far from complete.
 * It needs to be extended and the documentation updated to reflect your changes
    .
 *
 */
lexer grammar DecafLexer;

// This rule matches one of the keywords for Decaf - what others do you need?
BOOLEAN : 'boolean';
BREAK : 'break';
CALLOUT : 'callout';
CLASS : 'class';
CONTINUE : 'continue';
ELSE : 'else';
FALSE : 'false';
FOR : 'for';
IF : 'if';
INT : 'int';
RETURN : 'return';
TRUE : 'true';
VOID : 'void';

// These two rules deal with characters that have special meaning in Decaf -
    again, what others?
LCURLY : '{';
RCURLY : '}';
LBRACE : '(';
RBRACE : ')';
LSQUARE : '[';
RSQUARE : ']';
COMMA : ',';
ASSIGN : '=';
PLUSASSIGN : '+=';
MINUSASSIGN : '-=';
PLUS : '+';
MINUS : '-';
MULT : '*';
MOD : '%';
DIV : '/';
NOT : '!';
AND : '&&';
OR : '||';
EQ : '==';
NEQ : '!=';
LT : '<';
GT : '>';
LTE : '<=';
GTE : '>=';
END : ';';

// This says an identifier is a sequence of one or more alphabetic characters
// Decaf is a little more sophisticated than this.
ID :
   ('a'..'z' | 'A'..'Z' | '_') ('a'..'z' | 'A'..'Z' | '_' | '0'..'9')*;
```

```
// This rule simply ignores (skips) any space or newline characters
WS_ : (' ' | '\n' | '\t') -> skip;

// And this rule ignores comments (everything from a '//' to the end of the line
    )
SL_COMMENT : '//' (~'\n')* '\n' -> skip;

// These two rules incompletely describe characters and strings, and make use of
     the ESC fragment described below
// This rule says a character is contained within single quotes, and is a single
     instance of either an ESC, or any
// character other than a single quote.
CHAR : '\'' (ESC|NOTESC) '\'';

// This rule says a string is contained within double quotes, and is zero or
    more instances of either an ESC, or any
// character other than a double quote.
STRING : '"' (ESC|NOTESC)* '"';

// Rule says a number is either 0x followed by a hexidecimal combination, or
    just a series of numbers
fragment DIGIT : [0-9];
fragment HEX : [a-fA-F]|DIGIT;
NUMBER : (DIGIT+ | ('0x' (HEX)+) );

// A rule that is marked as a fragment will NOT have a token created for it. So
    anything matching the rules above
// will create a token in the output, but something matching the ESC rule below
    will only be used locally in the scope
// of this file. Any rule that should not generate an output token should be
    preceded by the fragment keyword.
// ESC matches either a pair of characters representing a newline ('\' and 'n')
    or a pair of characters representing
// a double quote ('\' and '"'). HINT: there are many other characters that
    should be escaped - think of how you need
// to write them in strings in languages like Java.
fragment
ESC :  '\\' ('n'|'"'|'t'|'\\'|'\'');

fragment
NOTESC :  ~('\n'|'"'|'\t'|'\\'|'\'');
```