```java
package test;

import static org.junit.Assert.*;

import java.io.IOException;
import java.io.StringReader;

import lexer.Lexer;

import org.junit.Test;

import frontend.Token;
import frontend.Token.Type;
import static frontend.Token.Type.*;

/**
 * This class contains unit tests for your lexer. Currently, there is only one test, but you
 * are strongly encouraged to write your own tests.
 */
public class LexerTests {
    // helper method to run tests; no need to change this
    private final void runtest(String input, Token... output) {
        Lexer lexer = new Lexer(new StringReader(input));
        int i=0;
        Token actual, expected;
        try {
            do {
                assertTrue(i < output.length);
                expected = output[i++];
                try {
                    actual = lexer.nextToken();
                    assertEquals(expected, actual);
                } catch(Error e) {
                    if(expected != null)
                        fail(e.getMessage());
                    return;
                }
            } while(!actual.isEOF());
        } catch (IOException e) {
            e.printStackTrace();
            fail(e.getMessage());
        }
    }

    /** Example unit test. */
    @Test
    public void testKWs() {
        // first argument to runtest is the string to lex; the remaining arguments
        // are the expected tokens
        runtest("module false return while boolean break else if import int public true type void",
                new Token(MODULE, 0, 0, "module"),
                new Token(FALSE, 0, 7, "false"),
                new Token(RETURN, 0, 13, "return"),
                new Token(WHILE, 0, 20, "while"),
                new Token(BOOLEAN, 0, 26, "boolean"),
                new Token(BREAK, 0, 34, "break"),
                new Token(ELSE, 0, 40, "else"),
                new Token(IF, 0, 45, "if"),
                new Token(IMPORT, 0, 48, "import"),
                new Token(INT, 0, 55, "int"),
                new Token(PUBLIC, 0, 59, "public"),
                new Token(TRUE, 0, 66, "true"),
                new Token(TYPE, 0, 71, "type"),
                new Token(VOID, 0, 76, "void"),
                new Token(EOF, 0, 80, ""));
    }

    public void testOperators() {
        runtest("/ == = >= > <= < - != + *",
                new Token(DIV, 0, 0, "/"),
                new Token(EQEQ, 0, 2, "=="),
                new Token(EQL, 0, 5, "="),
                new Token(GEQ, 0, 8, ">="),
                new Token(GT, 0, 11, ">"),
                new Token(LEQ, 0, 14, "<="),
                new Token(LT, 0, 17, "<"),
                new Token(MINUS, 0, 19, "-"),
                new Token(NEQ, 0, 21, "!="),
                new Token(PLUS, 0, 24, "+"),
                new Token(TIMES, 0, 26, "*"),
                new Token(EOF, 0, 28, ""));
    }

    @Test
    public void testIDs() {
        runtest("aSdf923k_dsf2145",
                new Token(ID, 0, 0, "aSdf923k_dsf2145"),
                new Token(EOF, 0, 16, ""));
        runtest("___aSdf923k_dsf2n45",
                new Token(ID, 0, 0, "___aSdf923k_dsf2n45"),
                new Token(EOF, 0, 19, ""));
        runtest("_",
                new Token(ID, 0, 0, "_"),
                new Token(EOF, 0, 1, ""));
    }

    @Test
    public void testStringLiteralWithDoubleQuote() {
        runtest("\"\"\"",
                new Token(STRING_LITERAL, 0, 0, ""),
                (Token)null);
    }

    @Test
    public void testStringLiteral() {
        runtest("\"\\n\"",
                new Token(STRING_LITERAL, 0, 0, "\\n"),
                new Token(EOF, 0, 4, ""));

    }

    @Test
    public void testStringLiteralWhitespace() {
        runtest("\"  foo_  % ^&_trump says module\"",
                new Token(STRING_LITERAL, 0, 0, "  foo_  % ^&_trump says module"),
                new Token(EOF, 0, 32, ""));

    }

    @Test
    public void testIntLiteral() {
        runtest("0000123000",
                new Token(INT_LITERAL, 0, 0, "0000123000"),
                new Token(EOF, 0, 10, ""));

    }

    @Test
    public void testPunctuation() {
        runtest("if (foo[0] == bar[1]) {method(1, \"trump\")};",
                new Token(IF, 0, 0, "if"),
                new Token(LPAREN, 0, 3, "("),
                new Token(ID, 0, 4, "foo"),
                new Token(LBRACKET, 0, 7, "["),
                new Token(INT_LITERAL, 0, 8, "0"),
                new Token(RBRACKET, 0, 9, "]"),
                new Token(EQEQ, 0, 11, "=="),
                new Token(ID, 0, 14 , "bar"),
                new Token(LBRACKET, 0, 17, "["),
                new Token(INT_LITERAL, 0, 18, "1"),
                new Token(RBRACKET, 0, 19, "]"),
                new Token(RPAREN, 0, 20, ")"),
                new Token(LCURLY, 0, 22, "{"),
                new Token(ID, 0, 23, "method"),
                new Token(LPAREN, 0, 29, "("),
                new Token(INT_LITERAL, 0, 30, "1"),
                new Token(COMMA, 0, 31, ","),
                new Token(STRING_LITERAL, 0, 33, "trump"),
                new Token(RPAREN, 0, 40, ")"),
                new Token(RCURLY, 0, 41, "}"),
                new Token(SEMICOLON, 0, 42, ";"),
                new Token(EOF, 0, 43, ""));

    }


}

```