```java
package dragon;

import org.antlr.v4.runtime.CharStream;
import org.antlr.v4.runtime.CharStreams;
import org.antlr.v4.runtime.Token;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.Path;

public class DragonLexerGrammarTest {
  InputStream is = System.in;

  @BeforeMethod
  public void setUp() throws IOException {
    is = new FileInputStream(Path.of("src/test/java/dragon/dragon0.txt
").toFile());
  }

  @AfterMethod
  public void tearDown() {
  }

  @Test
  public void testGetAllTokens() throws IOException {
    CharStream input = CharStreams.fromStream(is);
    DragonLexerGrammar lexer = new DragonLexerGrammar(input);

    for (Token token : lexer.getAllTokens()) {
      System.out.println(token);
    }
  }
}

```

```java
package dragon;

import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;

public class DragonLexerTest {
  private String input;
  private DragonLexer lexer;

  @BeforeMethod
  public void setUp() throws IOException {
    input = Files.readString(Path.of("src/test/java/dragon/dragon0.txt
"));
    lexer = new DragonLexer(input);
  }

  @AfterMethod
  public void tearDown() {
  }

  @Test
  public void testNextToken() {
    Token token = lexer.nextToken();

    while (token != Token.EOF && token != Token.WS) {
      System.out.println(token);
      token = lexer.nextToken();
    }
  }
}
```

```java
package dragon;

public class DragonLexer extends Lexer {
  private final KeywordTable kwTable = new KeywordTable();

  public DragonLexer(String input) {
    super(input);
  }

  @Override
  public Token nextToken() {
    if (peek == EOF) {
      return Token.EOF;
    }

    if (Character.isWhitespace(peek)) {
      WS();
    }

    if (Character.isLetter(peek)) {
      return ID();
    }

    if (Character.isDigit(peek)) {
      return NUMBER();
    }

    if (peek == '=') {
      consume();
      return Token.EQ;
    }

    if (peek == '<') {
      consume();
      if (peek == '=') {
        consume();
        return Token.LE;
      }

      if (peek == '>') {
        consume();
        return Token.NE;
      }

      return Token.LT;
    }

    if (peek == '>') {
      consume();

      if (peek == '=') {
        consume();
        return Token.GE;
      }

      return Token.GT;
    }

    Token unknown = new Token(TokenType.UNKNOWN, Character.toString(
  peek));
    consume();
    return unknown;
  }

  private Token WS() {
    while (Character.isWhitespace(this.peek)) {
      consume();
    }

    return Token.WS;
  }

  private Token ID() {
    StringBuilder sb = new StringBuilder();

    do {
      sb.append(peek);
      consume();
    } while (Character.isLetterOrDigit(peek));

    Token token = this.kwTable.getKeyword(sb.toString());
    if (token == null) {
      return new Token(TokenType.ID, sb.toString());
    }

    return token;
  }

  private Token INT() {
    StringBuilder sb = new StringBuilder();

    do {
      sb.append(peek);
      consume();
    } while (Character.isDigit(peek));

    return new Token(TokenType.INT, sb.toString());
  }

  private Token NUMBER() {
    StringBuilder intStr = new StringBuilder();
    intStr.append(peek);
    consume();

    int intPos = -1;
    int realPos = -1;
```

```java

    StringBuilder realStr = new StringBuilder();
    StringBuilder sciStr = new StringBuilder();

    int state = 13;
    while (true) {
      switch (state) {
        case 13:
          intPos = pos;
          if (Character.isDigit(peek)) {
            intStr.append(peek);
            consume();
            state = 13;
            break;
          } else if (peek == '.') {
            realStr.append(peek);
            consume();
            state = 14;
          } else if (peek == 'E' || peek == 'e') {
            sciStr.append(peek);
            consume();
            state = 16;
            break;
          } else {
            return new Token(TokenType.INT, intStr.toString());
          }
        case 14:
          if (Character.isDigit(peek)) {
            realStr.append(peek);
            consume();
            state = 15;
            break;
          } else {
            this.reset(intPos);
            return new Token(TokenType.INT, intStr.toString());
          }
        case 15:
          realPos = pos;
          if (Character.isDigit(peek)) {
            realStr.append(peek);
            consume();
            state = 15;
            break;
          } else if (peek == 'E') {
            sciStr.append(peek);
            consume();
            state = 16;
            break;
          } else {
            return new Token(TokenType.REAL, intStr.append(realStr).
  toString());
          }
        case 16:
          if (peek == '+' || peek == '-') {
            sciStr.append(peek);
            consume();
            state = 17;
            break;
          } else if (Character.isDigit(peek)) {
            sciStr.append(peek);
            consume();
            state = 18;
            break;
          } else {
            this.reset(realPos);
            return new Token(TokenType.REAL, intStr.append(realStr).toString());
          }
        case 17:
          if (Character.isDigit(peek)) {
            sciStr.append(peek);
            consume();
            state = 18;
            break;
          } else {
            this.reset(realPos);
            return new Token(TokenType.REAL, intStr.append(realStr).toString());
          }
        case 18:
          if (Character.isDigit(peek)) {
            sciStr.append(peek);
            consume();
            state = 18;
            break;
          } else {
            return new Token(TokenType.SCI, intStr.append(realStr).append(sciStr).toString());
          }
        default:
          System.err.println("Unreachable");
      }
    }
  }
}

```

```java
package dragon;

import java.util.HashMap;
import java.util.Map;

public class KeywordTable {
  private final Map<String, Token> keywords = new HashMap<>();

  public KeywordTable() {
    this.reserve(Token.IF);
    this.reserve(Token.ELSE);
  }

  public Token getKeyword(String text) {
    return this.keywords.get(text);
  }

  private void reserve(Token token) {
    keywords.put(token.getText(), token);
  }
}
```

```java
package dragon;

public abstract class Lexer {
  public static final char EOF = (char) -1;

  private final String input;
  char peek;
  int pos;

  public Lexer(String input) {
    this.input = input;
    this.pos = 0;
    this.peek = input.charAt(pos);
  }

  public abstract Token nextToken();

  public void reset(int pos) {
    this.pos = pos;
    this.peek = input.charAt(pos);
  }

  public void consume() {
    this.pos++;
    if (this.pos >= this.input.length()) {
      this.peek = EOF;
    } else {
      this.peek = input.charAt(this.pos);
    }
  }
}
```

```java
package dragon;

public class Token {
  public static final Token EOF = new Token(TokenType.EOF, "EOF");
  public static final Token WS = new Token(TokenType.WS, " ");

  public static final Token IF = new Token(TokenType.IF, "if");
  public static final Token ELSE = new Token(TokenType.ELSE, "else");

  public static final Token EQ = new Token(TokenType.EQ, "=");
  public static final Token NE = new Token(TokenType.NE, "<>");
  public static final Token LT = new Token(TokenType.LT, "<");
  public static final Token LE = new Token(TokenType.LE, "<=");
  public static final Token GT = new Token(TokenType.GT, ">");
  public static final Token GE = new Token(TokenType.GE, ">=");

  private final TokenType type;
  private final String text;

  public Token(TokenType type, String text) {
    this.type = type;
    this.text = text;
  }

  public TokenType getType() {
    return type;
  }

  public String getText() {
    return this.text;
  }

  @Override
  public String toString() {
    return String.format("token {type : %s, text : %s}",
        this.type, this.text);
  }
}
```

```java
package dragon;

/**
 * Types of tokens
 */
public enum TokenType {
  // Group 0
  EOF,
  WS,
  UNKNOWN,

  // Group 1
  IF, ELSE,
  ID,
  INT,

  // Group 2
  // =, <>, <, <=, >, >=
  EQ, NE, LT, LE, GT, GE,

  // Group 3
  REAL,
  SCI,
}
```