```java
package Lexer;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.nio.charset.Charset;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Created by rubenspessoa on 04/09/16.
 */
public class Lexer {

    /**
     * Internal class holding the information of a token type.
     */
    private class TokenInfo {

        public final Pattern regex;
        public final Token.TokenCategory tokenCategory;

        /**
         * Construct TokenInfo within its values.
         * @param regex Lexer.Token Regex Pattern
         * @param tokenCategory Lexer.Token Category.
         */
        public TokenInfo(Pattern regex,
                         Token.TokenCategory tokenCategory) {
            super();
            this.regex = regex;
            this.tokenCategory = tokenCategory;
        }

    }

    private LinkedList<TokenInfo> tokenInfos;
    private LinkedList<Token> tokens;
    private static Lexer lexer = null;

    private Lexer() {
        this.tokenInfos = new LinkedList<TokenInfo>();
        this.tokens = new LinkedList<Token>();
    }

    /**
     * @return the lexer for Nova programming language.
     */
    public static Lexer getLexer() throws Exception {
        if (lexer == null) {
            lexer = createLexer();
        }
        return lexer;
    }

    private static Lexer createLexer() throws Exception {
        Lexer lexer = new Lexer();
```

```
        lexer.add("readIn|printOut", Token.TokenCategory.PR_IO);
        lexer.add("void", Token.TokenCategory.PR_VOID);
        lexer.add("main", Token.TokenCategory.PR_MAIN);
        lexer.add("if", Token.TokenCategory.PR_IF);
        lexer.add("else", Token.TokenCategory.PR_ELSE);
        lexer.add("while", Token.TokenCategory.PR_WHILE);
        lexer.add("for", Token.TokenCategory.PR_FOR);
        lexer.add("shoot", Token.TokenCategory.PR_SHOOT);
        lexer.add("string|int|float|bool", Token.TokenCategory.TYPE_VALUE);
        lexer.add("True|False", Token.TokenCategory.BOOL_VALUE);
        lexer.add("=", Token.TokenCategory.OP_ATR);
        lexer.add("<|>|<=|>=", Token.TokenCategory.OP_REL1);
        lexer.add("==|!=", Token.TokenCategory.OP_REL2);
        lexer.add("\\+|-", Token.TokenCategory.OP_AD);
        lexer.add("\\*|/|%", Token.TokenCategory.OP_MULT);
        lexer.add("and", Token.TokenCategory.OP_AND);
        lexer.add("or", Token.TokenCategory.OP_OR);
        lexer.add("not", Token.TokenCategory.OP_NOT);
        lexer.add(";|,", Token.TokenCategory.SP);
        lexer.add("\\(", Token.TokenCategory.AB_PAR);
        lexer.add("\\)", Token.TokenCategory.FEC_PAR);
        lexer.add("\\[", Token.TokenCategory.AB_COL);
        lexer.add("\\]", Token.TokenCategory.FEC_COL);
        lexer.add("\\{", Token.TokenCategory.AB_CH);
        lexer.add("\\}", Token.TokenCategory.FEC_CH);
        lexer.add("::", Token.TokenCategory.VECTOR_AUX);
        lexer.add("[a-zA-Z][_a-zA-Z0-9]*\\w*", Token.TokenCategory.ID);
        lexer.add("[+|-]?([0-9]*\\.[0-9]+)", Token.TokenCategory.CTE_FLOAT);
        lexer.add("[0-9]+", Token.TokenCategory.CTE_INT);
        lexer.add("[a-zA-Z_]?\"(\\.|[^\"])*\"", Token.TokenCategory.CTE_STR)
            ;
        lexer.add("#[a-zA-Z][_a-zA-Z0-9]*", Token.TokenCategory.COMMENT);

        return lexer;
    }

    /**
     * Add a regular expression and a token id to the internal list of
         recognized tokens
     * @param regex regular expression to match against
     * @param tokenCategory tokenCategory that the regular expression is
         linked to
     */
    public void add(String regex, Token.TokenCategory tokenCategory) throws
        Exception {
        tokenInfos.add(
                new TokenInfo(
                        Pattern.compile("^(" + regex+")"),
                        tokenCategory
                )
        );
    }

    /**
     * Tokenize an input File.
     * Calls private method lex(String inputString)
     * @param file with the code to be tokenized.
     */
    public void lex(File file) throws Exception {
```

```
        tokens.clear();

        FileInputStream in;
        String line;
        InputStreamReader isr;
        BufferedReader br;

        ArrayList<String> lines = new ArrayList<String>();
        in = new FileInputStream(file);
        isr = new InputStreamReader(in, Charset.forName("UTF-8"));
        br = new BufferedReader(isr);

        while ((line = br.readLine()) != null) {
            lines.add(line);
        }

        for (int i = 0; i < lines.size(); i++) {
            lex(lines.get(i), i);
        }

    }

    /**
     * Tokenize an input string.
     * The result can be acessed via getTokens().
     * @param inputString
     */
    private void lex(String inputString, int line) throws Exception {

        String s = inputString.replace(" ", "");
        int totalLength = s.length();

        while (!s.equals("")) {
            int remaining = s.length();
            boolean match = false;

            for (TokenInfo info : tokenInfos) {
                Matcher m = info.regex.matcher(s);

                if (m.find()) {
                    match = true;
                    String token = m.group().trim();
                    s = m.replaceFirst("").trim();
                    tokens.add(
                            new Token(
                                    info.tokenCategory,
                                    token,
                                    line,
                                    totalLength - remaining
                            ));
                    break;
                }
            }

            if (!match) {
                throw new LexerException("Unexpected character in input: " +
                    s);
            }
        }
    }
```

```
    /**
     * Get the tokens generated in the last call of tokenize.
     * @return a list of tokens to be fed to Parser
     */
    public LinkedList<Token> getTokens() {
        return this.tokens;
    }

}
```