```
/**
 *  This is the main class for your compiler. It uses the command line interface
      (CLI) tools found in java6G6Z1110.tools.CLI.
 *  It is <em>very</em> incomplete. For the first stage, you will need to update
      the section handling the CLI.SCAN option.
 *  Study this section of code to work out what it is doing, and read the
      specifications carefully to understand what it
 *  <em>should</em> be doing, then modify it accordingly.
 *  For the second stage, some code is provided that may help you with your
      debugging, but you must change it to give appropriate
 *  correct behaviour for submission.
 *  For the final stage, you need to think carefully about what you will need to
      do here.
 *
 *  DO NOT FORGET TO CHANGE THESE COMMENTS AS WELL AS THE CODE
 */
package decaf;

import java.io.*;
import org.antlr.v4.runtime.Token;
import org.antlr.v4.runtime.tree.*;

import java6G6Z1110.tools.CLI.*;

import org.antlr.v4.runtime.ANTLRInputStream;
import org.antlr.v4.runtime.CommonTokenStream;

/**
 * @author Emma Norling (based on code from MIT OpenCourseWare http://ocw.mit.
     edu
 * for the subject 6.035 Computer Language Engineering, Spring 2010)
 *
 */
public class Main {

        /**
         * @param args - command line arguments
         */
        public static void main(String[] args) {
                // TODO Auto-generated method stub
                try {
            CLI.parse (args, new String[0]);

            InputStream inputStream = args.length == 0 ?
                    System.in : new java.io.FileInputStream(CLI.infile);
            ANTLRInputStream antlrIOS = new ANTLRInputStream(inputStream);

            if (CLI.target == CLI.SCAN || CLI.target == CLI.DEFAULT)
            {
                DecafLexer lexer = new DecafLexer(antlrIOS);
                Token token;
                boolean done = false;
                while (!done)
                {
                    try
                    {
                        for (token=lexer.nextToken();
```

```
                token.getType()!=Token.EOF; token=lexer.nextToken())
            {
                String type = "";
                String text = token.getText();


                switch (token.getType())
                {
                case DecafLexer.ID:
                    type = " IDENTIFIER";
                    break;
                case DecafLexer.STRING:
                    type = " STRINGLITERAL";
                    break;
                case DecafLexer.CHAR:
                    type = " CHARLITERAL";
                    break;
                case DecafLexer.NUMBER:
                    type = " INTLITERAL";
                    break;
                case DecafLexer.TRUE: case DecafLexer.FALSE:
                    type = " BOOLEANLITERAL";
                    break;
                }
                System.out.println (token.getLine() + type + " " +
                    text);
            }
            done = true;
        } catch(Exception e) {
            // print the error:
            System.out.println(CLI.infile+" "+e);
        }
    }
}
else if (CLI.target == CLI.PARSE)
{
    DecafLexer lexer = new DecafLexer(antlrIOS);
    CommonTokenStream tokens = new CommonTokenStream(lexer);
    DecafParser parser = new DecafParser (tokens);
    ParseTree tree = parser.program();
    if (CLI.debug) {
        TreePrinterListener listener = new TreePrinterListener(
            parser);
            ParseTreeWalker.DEFAULT.walk(listener, tree);
            String formatted = listener.toString();
            System.out.println(formatted);
    }
} else if (CLI.target == CLI.INTER)
{
    DecafLexer lexer = new DecafLexer(antlrIOS);
    CommonTokenStream tokens = new CommonTokenStream(lexer);
    DecafParser parser = new DecafParser(tokens);
    ParseTree tree = parser.program();
    ScopeListener listener = new ScopeListener();
    ParseTreeWalker.DEFAULT.walk(listener, tree);
}
```

```
        } catch(Exception e) {
            // print the error:
            System.out.println(CLI.infile+" "+e);
        }
    }
}
```