

# Install and Configure ANTLR 4 for Ubuntu and MacOS X

Ronald Mak

Department of Computer Engineering

Department of Computer Science

Department of Applied Data Science

January 7, 2020

## Introduction

This tutorial describes how to install the ANTLR 4 and the ANTLR 4 plugin for Eclipse running on either Ubuntu or MacOS X.

ANTLR 4 (“Another Tool for Language Recognition”) is a “compiler-compiler”, a tool that generates components of a compiler for a programming language. See <https://www.antlr.org>. We will use this tool in our compiler design class. Given a grammar file for a programming language, it will generate a parser, lexer (scanner), and parse tree classes written in Java or C++. Other included tools create graphical syntax diagrams and parse tree diagrams.

If you want to use Ubuntu and have not already installed it, read “Install Ubuntu on Windows 10 and on VirtualBox” (<http://www.cs.sjsu.edu/~mak/tutorials/InstallUbuntu.pdf>) and “Configure Ubuntu for Software Development” (<http://www.cs.sjsu.edu/~mak/tutorials/ConfigureUbuntu.pdf>).

If you have not already installed Eclipse, read the tutorial, “Install Eclipse for Java and C++ Development” (<http://www.cs.sjsu.edu/~mak/tutorials/InstallEclipse.pdf>).

**If you only want ANTLR to generate Java code**, then this tutorial is all you need to read. If you want ANTLR to generate C++ code, then after reading this tutorial, read “Install and Configure ANTLR 4 for C++” (<http://www.cs.sjsu.edu/~mak/tutorials/InstallANTLR4Cpp.pdf>).

## Download ANTLR 4 jar file

Go to <https://www.antlr.org/download.html>. Download file `antlr-4.7.2-complete.jar` which contains the ANTLR tool itself: <https://www.antlr.org/download/antlr-4.7.2-complete.jar>. **Create a directory named ANTLR-4.7.2** and move the jar file into it.

## Edit your startup script

In an Ubuntu terminal window, **change to your home directory**. On Ubuntu, edit the startup script `.bashrc`. On MacOS X, edit `.bash_profile`. Add the following lines at the end of the script:

```
# ANTLR 4
export ANTLR_HOME="$HOME/ANTLR-4.7.2"
export ANTLR_JAR="$ANTLR_HOME/antlr-4.7.2-complete.jar"
export CLASSPATH=".:$ANTLR_JAR:$CLASSPATH"
alias antlr4="java -jar $ANTLR_JAR"
alias grun="java org.antlr.v4.gui.TestRig"
```

To make your changes take effect immediately in your terminal, type the command

```
source .bashrc
```

on Ubuntu, or the command

```
source .bash_profile
```

on MacOS X.

Any new terminal window that you open in the future will have the new `CLASSPATH` and the new aliases.

## A simple expression grammar

In your home directory, **create a directory named `example` and `cd` to it**. Create a grammar file `Expr.g4` which is a text file containing:

```
grammar Expr;

prog: (expr NEWLINE)* ;
expr: expr ('*' | '/' ) expr
     | expr ('+' | '-' ) expr
     | INT
     | '(' expr ')'
     ;

NEWLINE : [\r\n]+ ;
INT     : [0-9]+ ;
```

This is a grammar for simple arithmetic expressions. Use your `antlr4` alias to invoke ANTLR 4 to process the grammar file:

```
antlr4 Expr.g4
```

You should see that ANTLR generated Java source files for the parser and lexer and other files in the directory. Compile the Java source files with the command:

```
javac Expr*.java
```

## R. Mak, Install and Configure ANTLR 4 for Ubuntu and MacOS X

Now you should also see `.class` files. Use your `grun` alias to run the parser and lexer on a Java test rig. Give an arithmetic expression as input and generate a graphical parse tree. Terminate the input with control-D (^D):

```
grun Expr prog -gui
100+2*34
^D
```

A parse tree should display (Figure 1).

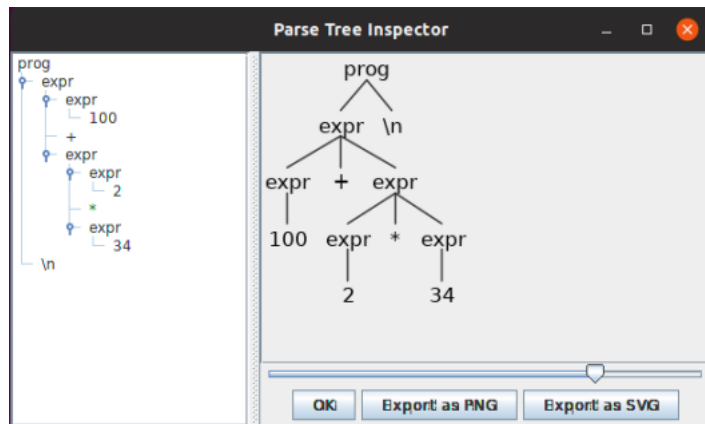


Figure 1. An expression parse tree.

## Install the ANTLR plugin for Eclipse

Go to <https://github.com/jknack/antlr4ide>, scroll down the page, and carefully follow the *Installation* instructions under *Eclipse Installation*. Modifications to the instructions:

- Ignore the instructions' version numbers of the components, but instead install the latest version of each component. In particular, install the latest version of Eclipse (currently 2019-12).
- In steps 2 and 3, wait until the installer finishes and you get the prompt to restart Eclipse. This can take several minutes.
- In step 3, select *2019-12 - http://download.eclipse.org/releases/2019-12* (or whatever the latest version of Eclipse is) from the *Work with* dropdown menu.
- In step 4, install ANTLR 4 IDE 0.3.6.

After all the plugin installations, select *Help | Install New Software* again. Click the *already installed* link at the bottom right. You should now see:

- ANTLR 4 SDK
- Eclipse Faceted Project Framework
- Eclipse Faceted Project Framework JDT Enablement
- Xtext Complete SDK

## Create the Java Hello ANTLR project

Go to <https://github.com/jknack/antlr4ide>, scroll down the page and carefully follow the instructions for *Creating a project in Eclipse*:

- First close the *Welcome* tab if it's still open.
- In step 12, you must first open a `.g4` grammar file in the editor window before you will see ANTLR 4 in the list of properties.
- In step 15, after you're done with the project specific settings, uncheck the *Enable project specific setting*. Click the *Configure Workspace Settings* link. Then add and select your `antlr-4.7.2-complete.jar` file.

The default Java-based ANTLR project will include a sample `Hello.g4` grammar file and generate several `.java` and `.token` files in the project under `target/generated-sources/antlr4`. You can use these files to verify that the ANTLR plugin works properly.

From the *Window* dropdown menu at the top, select *Show View | Other*. Under ANTLR 4, select *Syntax Diagram* and click the *Open* button. Repeat and select *Parse Tree*. Open the `Hello.g4` grammar file in the editor window. Select the *Syntax Diagram* tab. You should see a syntax diagram that ANTLR generated from the grammar file (Figure 2). If you do not see a syntax diagram, try closing the project and then reopening it.

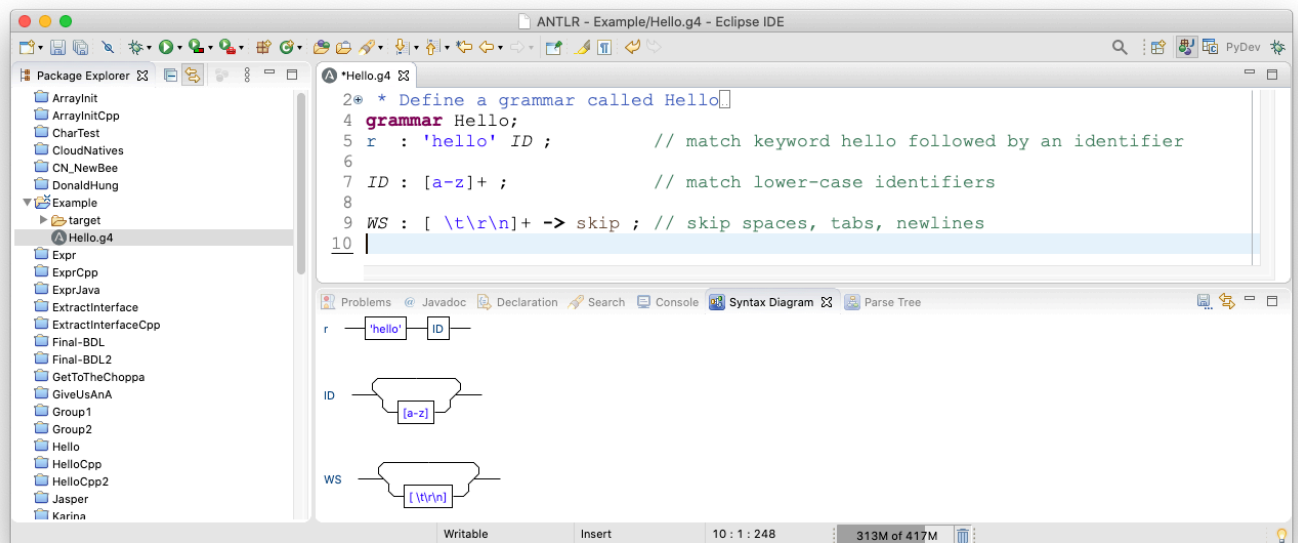


Figure 2. The sample `Hello.g4` grammar file, generated code, and syntax diagram.

## Create the Java Expr ANTLR project

Create a new Java project named `Expr`. Open the project properties and select *Java Compiler*. Set the *Compiler compliance level* to a level below 9, such as 1.8. Uncheck *Use '—release' option*. See Figure 3. Click the *Apply and Close* button. (ANTLR 4 has compatibility problems with Java levels 9 and above.)

## R. Mak, Install and Configure ANTLR 4 for Ubuntu and MacOS X

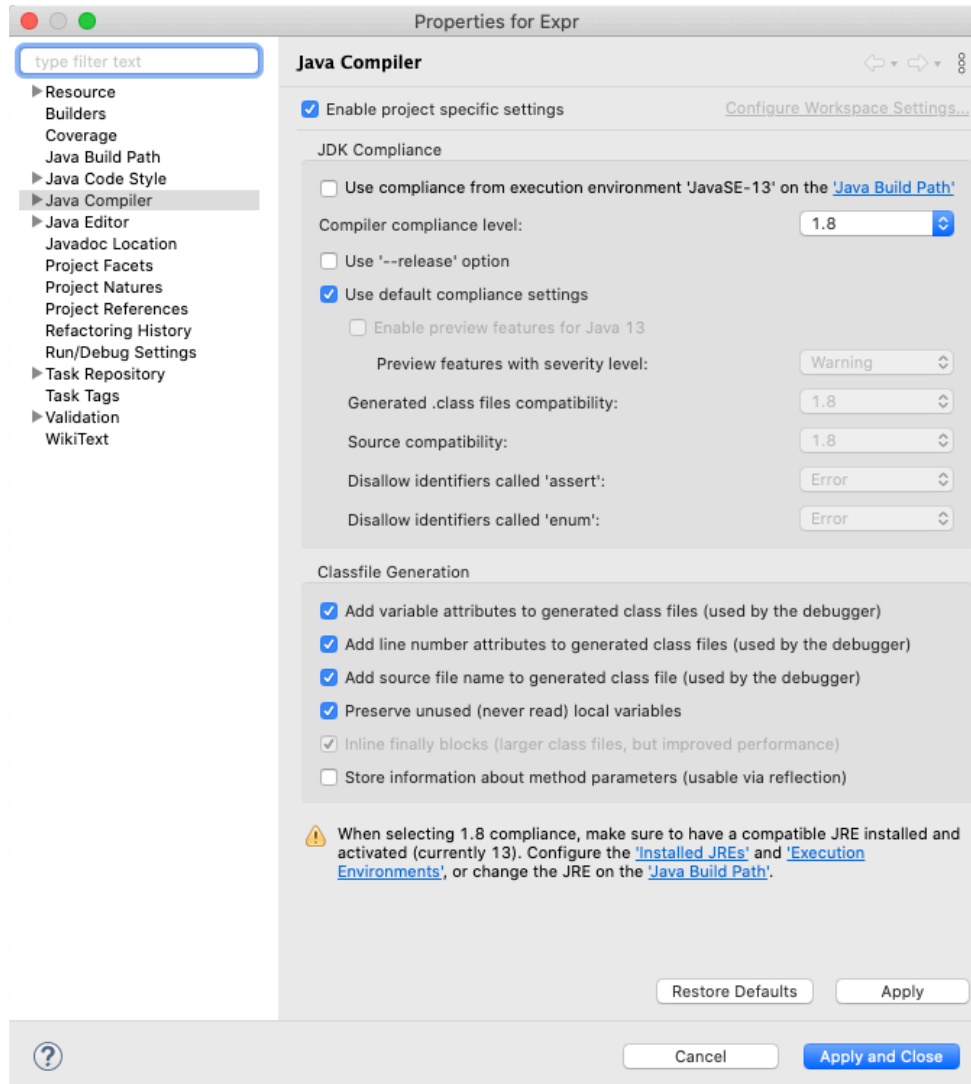


Figure 3: *Java Compiler* settings.

If it exists, delete the source file `module-info.java` from the project.

Copy the `Expr.g4` grammar file to the top level of your project, not under `src`. Open `Expr.g4` in the editor window. If ANTLR did not automatically generate compiler files, select `Expr.g4` in the project's file list, right click, and select `Run As | Generate ANTLR Recognizer` from the context menu. Then you should be able open the syntax diagram view (Figure 4).

## R. Mak, Install and Configure ANTLR 4 for Ubuntu and MacOS X

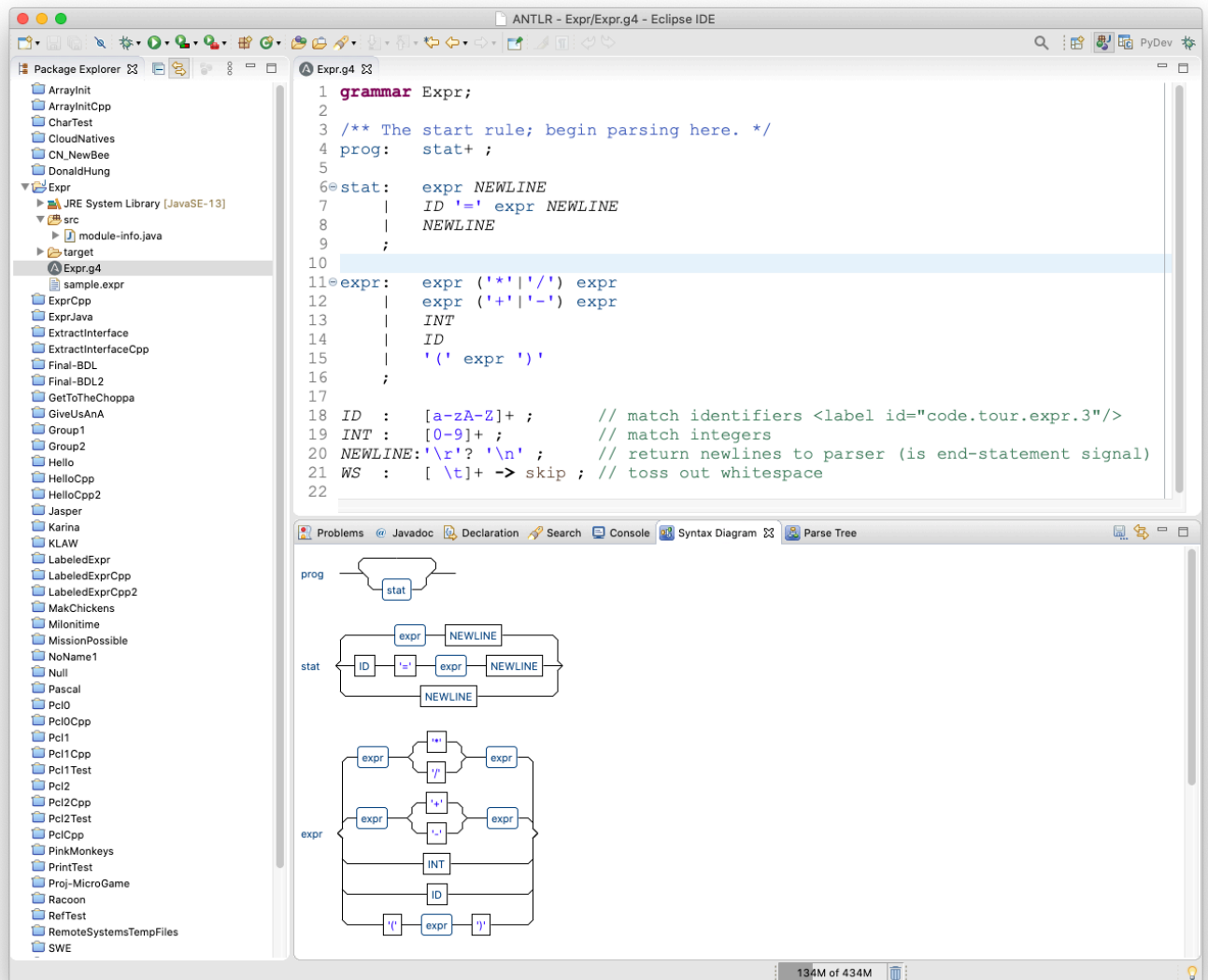


Figure 4. Syntax diagrams generated from `Expr.g4`.

The generated compiler files will regenerate automatically each time you modify the grammar file. You can manually cause a regeneration by right-clicking the name of the grammar file and selecting *Run As | Generate ANTLR Recognizer* from the context menu.

Right-click the project name and select *New | File* to create an input source file `sample.expr` at the top level. Enter the arithmetic expression `100+2*34` which we used before into the file and then save the file. (Figure 5).

## R. Mak, Install and Configure ANTLR 4 for Ubuntu and MacOS X

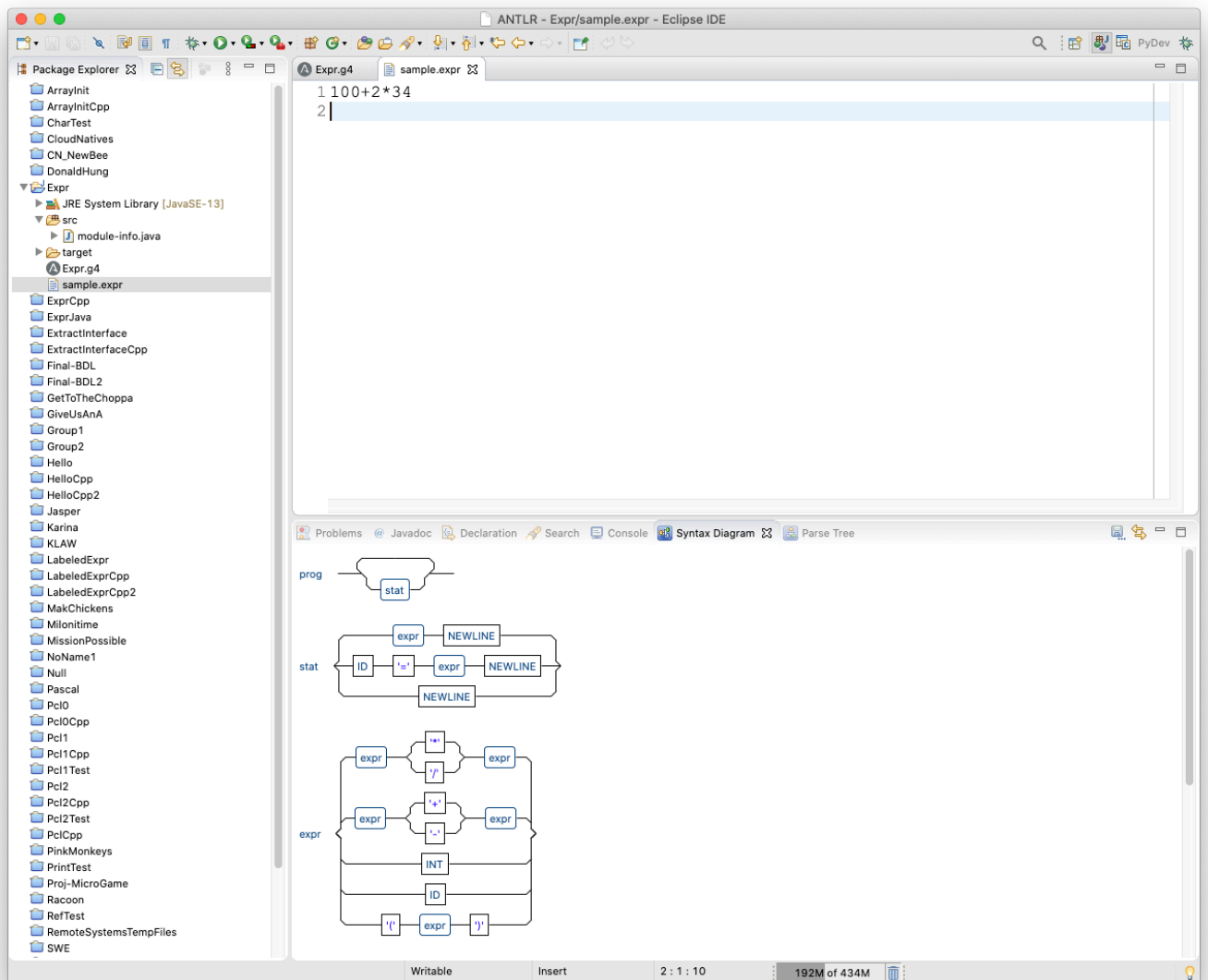


Figure 5. Input source file **sample.expr**.

Copy the contents of **sample.expr** onto the clipboard. Select the *Parse Tree* tab. Select the **Expr.g4** tab and click on **prog**, the start rule. Paste the contents of the clipboard into the **Expr : : prog** panel, and you should see a parse tree that the plugin generated from the source file contents (Figure 6).

## R. Mak, Install and Configure ANTLR 4 for Ubuntu and MacOS X

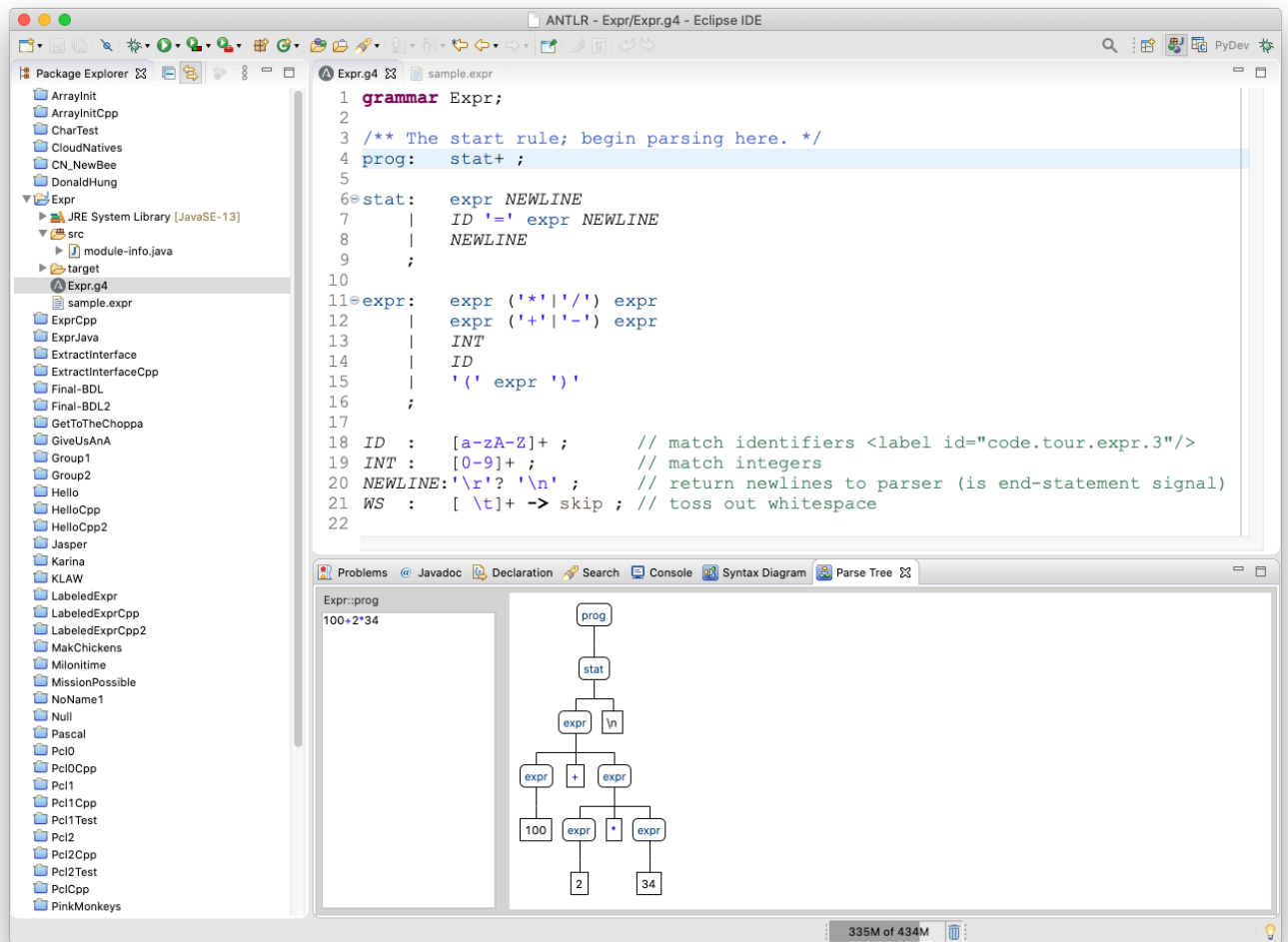


Figure 6. A parse tree generated from the input source file `sample.expr`.

### Run a Java-based ANTLR project

Open the project properties and select *Java Build Path*.

In the *Source* tab, click the *Add Folder* button and select *target | generated-sources | antlr4* (Figure 7).

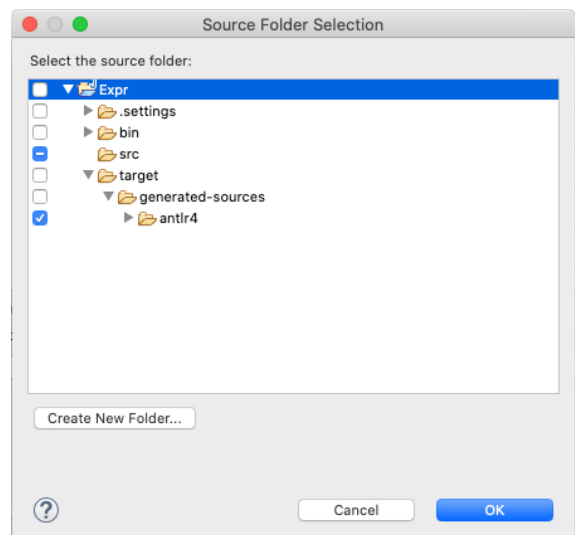


Figure 7. Adding the generated sources.



## R. Mak, Install and Configure ANTLR 4 for Ubuntu and MacOS X

In the *Libraries* tab, click the *Add External JARs* button, navigate to where you stored **antlr-4.7.2-complete.jar**, and add the jar file to the list of libraries (Figure 8). Click the *Apply and Close* button.

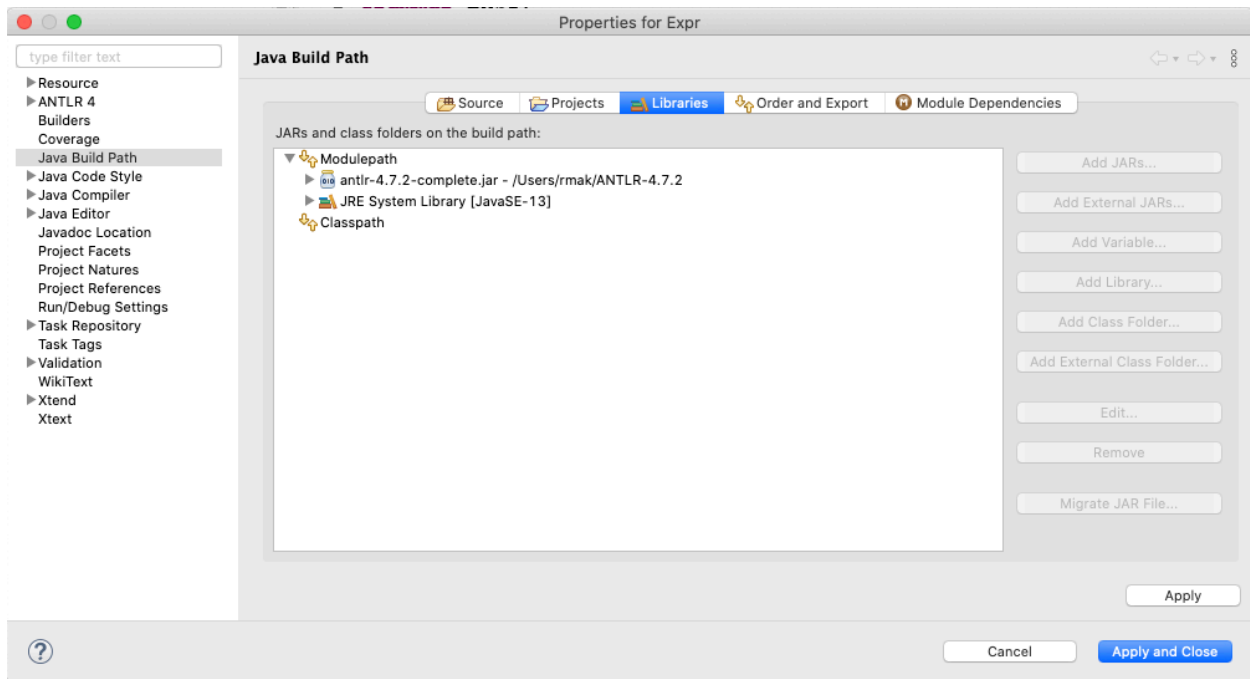


Figure 8. Adding **antlr-4.7.2-complete.jar** to the list of libraries.

To run in Eclipse, our sample project needs a main. Right-click the project name and select *New | Class* from the context menu. Enter **ExprMain** as the name of the class (Figure 9). Click the *Finish* button.

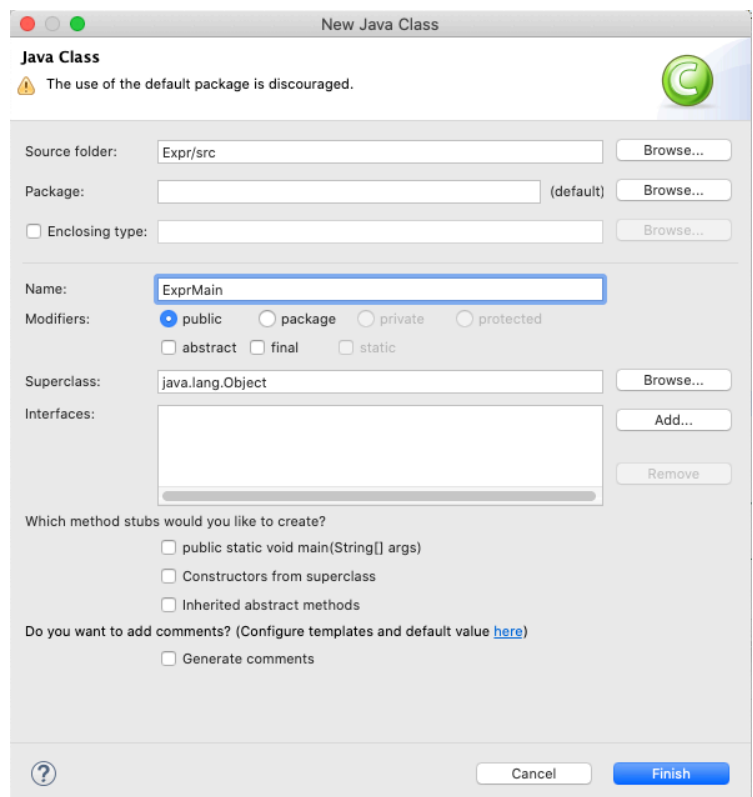


Figure 9. Creating the main class **ExprMain**.

Enter the following code for class **ExprMain**:

```
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.tree.*;
import java.io.FileInputStream;
import java.io.InputStream;

public class ExprMain
{
    public static void main(String[] args) throws Exception
    {
        String inputFile = null;

        // Create the input stream.
        if (args.length > 0) inputFile = args[0];
        InputStream is = System.in;
        if (inputFile != null) is = new FileInputStream(inputFile);

        // Create the character stream from the input stream.
        CharStream cs = CharStreams.fromStream(is);

        // Create a lexer which scans the character stream
        // to create a token stream.
        ExprLexer lexer = new ExprLexer(cs);
        CommonTokenStream tokens = new CommonTokenStream(lexer);

        // Print the token stream.
        System.out.println("Tokens:");
        tokens.fill();
        for (Token token : tokens.getTokens())
        {
            System.out.println(token.toString());
        }

        // Create a parser which parses the token stream
        // to create a parse tree.
        ExprParser parser = new ExprParser(tokens);
        ParseTree tree = parser.prog();

        // Print the parse tree in Lisp format.
        System.out.println("\nParse tree (Lisp format):");
        System.out.println(tree.toStringTree(parser));
    }
}
```

Now create a run configuration for this project. Select the project name. From the context menu, select *Run As | Run Configurations*. In the **Run Configurations** dialog box, select *Java Application* in the left panel and click the *New Launch Configuration* button above it (the document icon with the yellow plus sign).

In the right panel, type **Expr** for the configuration name (Figure 10).

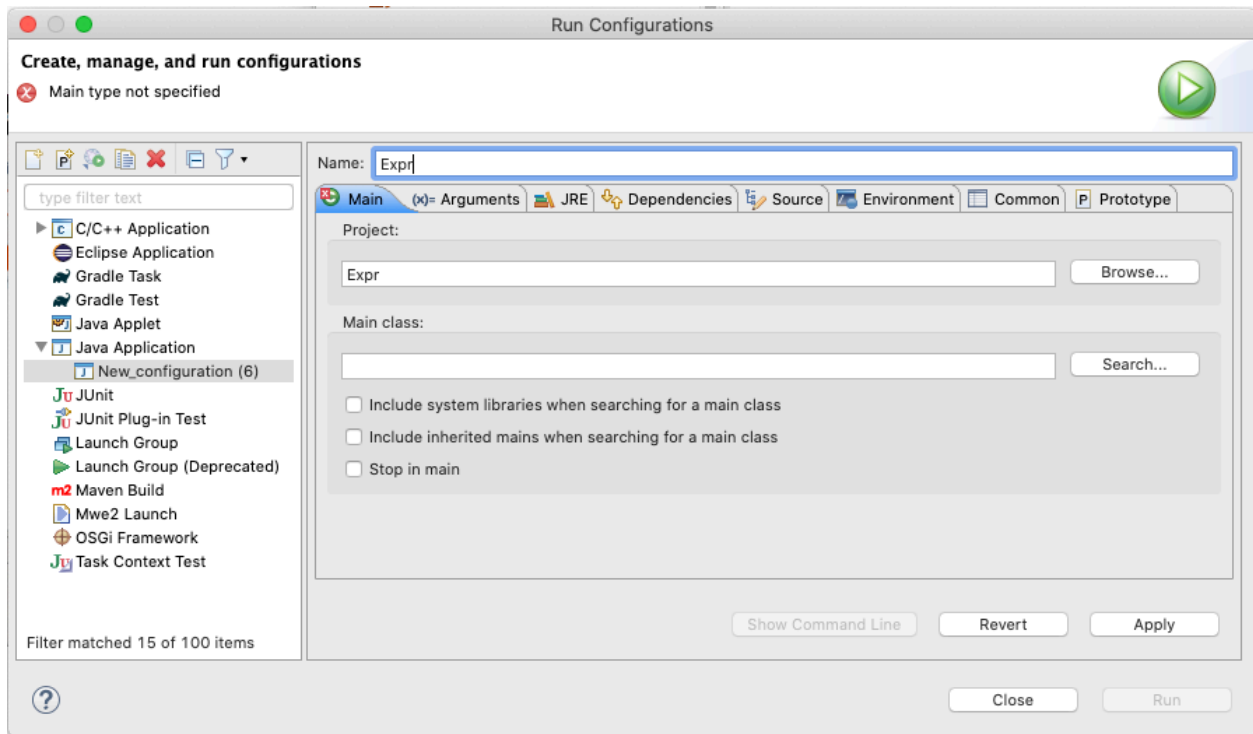


Figure 10. Creating a run configuration for the project.

Under *Main class*, click the *Search* button. In the **Select Main Type** dialog box, select **ExprMain** and click the OK button (Figure 11).

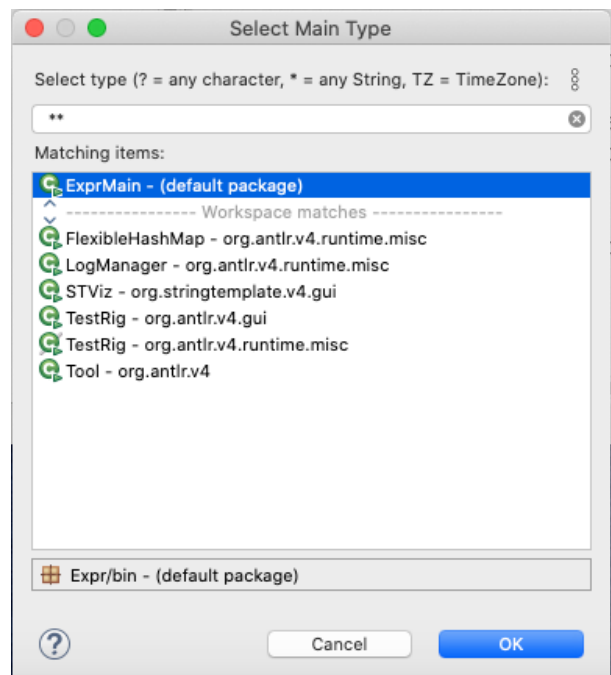


Figure 11. Selecting the main class **ExprMain**.

## R. Mak, Install and Configure ANTLR 4 for Ubuntu and MacOS X

Back in the **Run Configurations** dialog box, click the *Arguments* tab and enter **sample.expr** as the program's command-line argument, the source file to compile. (Figure 12).

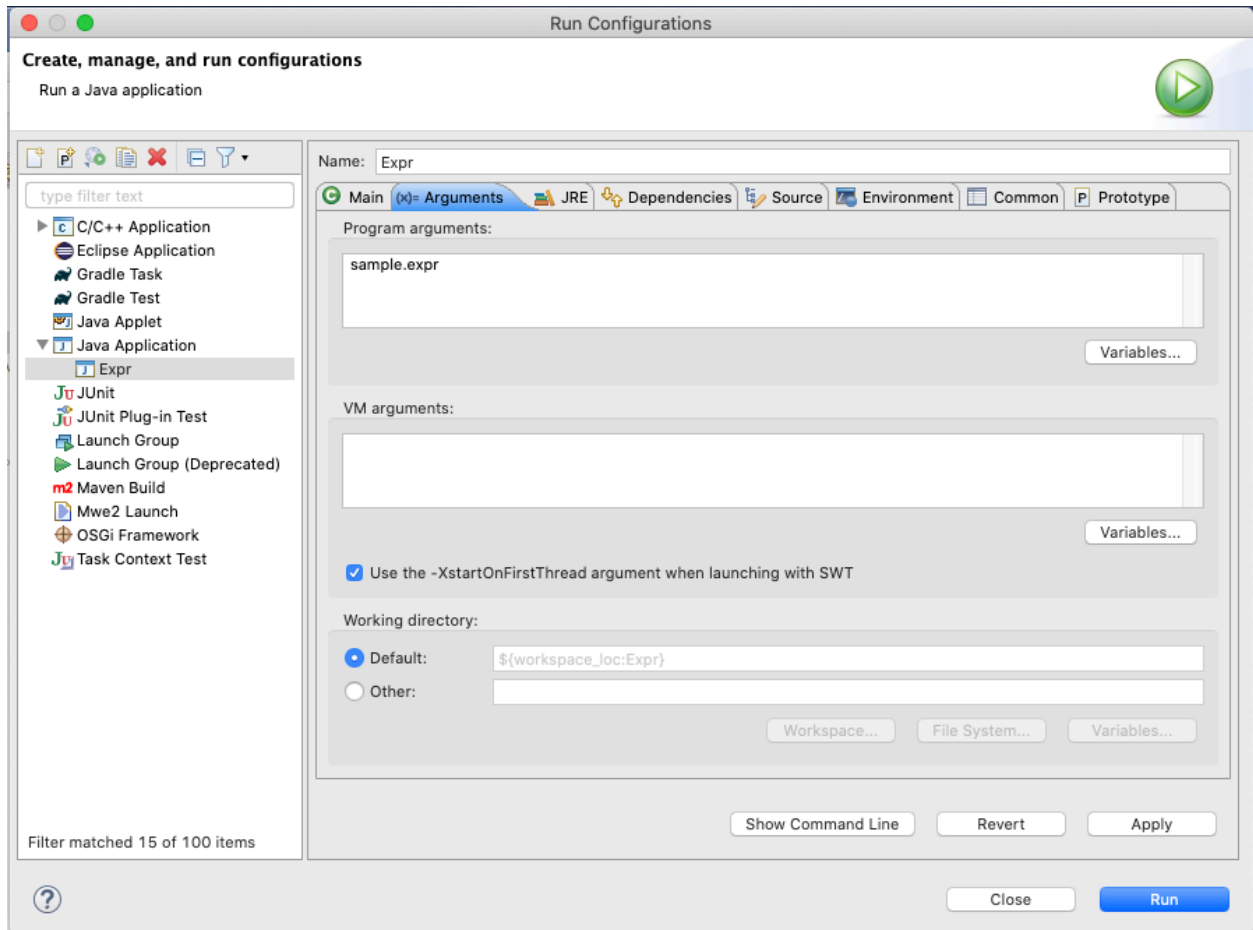
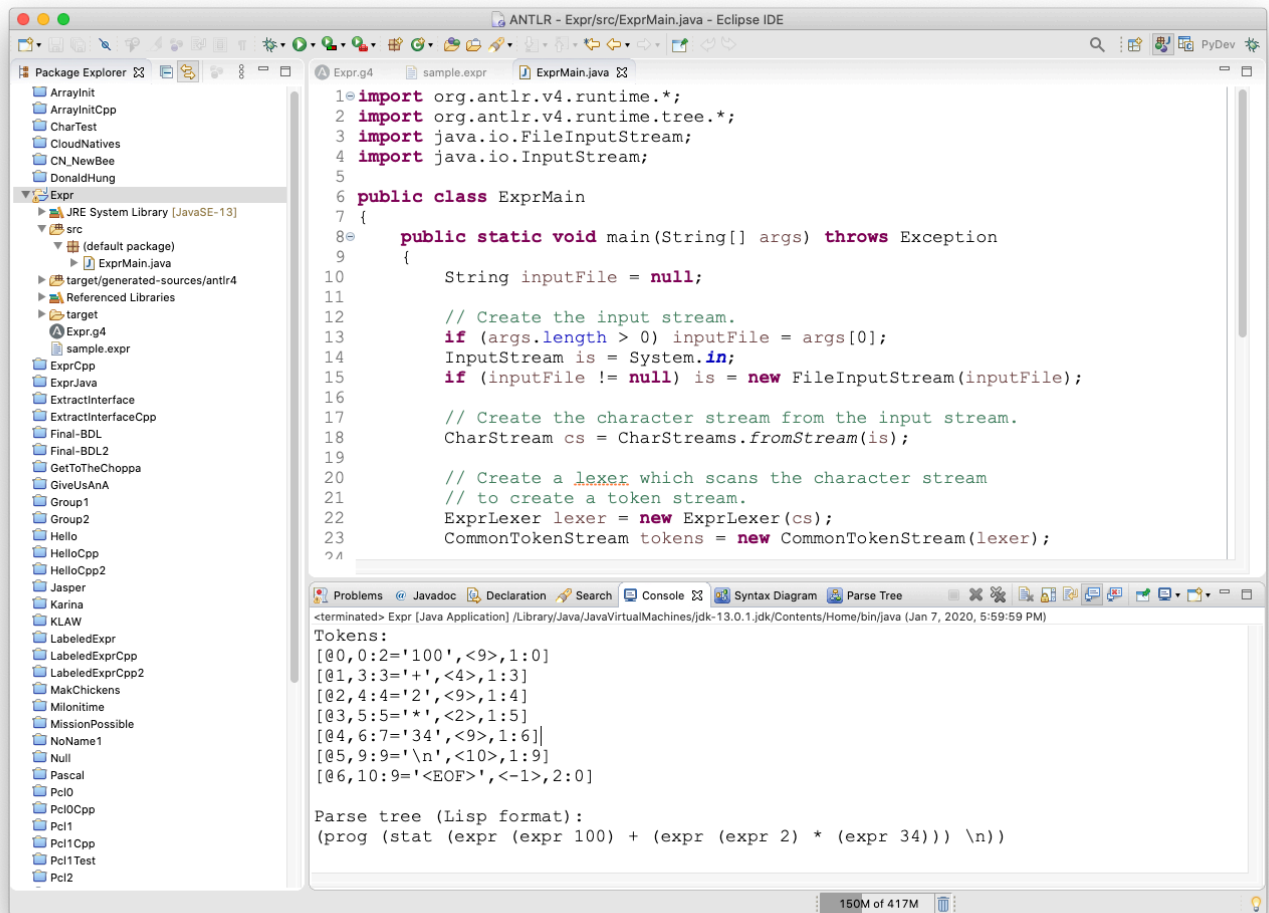


Figure 12. Specifying the command-line argument **sample.expr**, the source file to compile.

Click the *Apply* button and then the *Run* button. You should see output from running the generated compiler in the *Console* tab of the main Eclipse window. First is a list of tokens from the token stream generated by the lexer, and then the parse tree generated by the parser in Lisp format (Figure 13).

## R. Mak, Install and Configure ANTLR 4 for Ubuntu and MacOS X



```
1 import org.antlr.v4.runtime.*;
2 import org.antlr.v4.runtime.tree.*;
3 import java.io.FileInputStream;
4 import java.io.InputStream;
5
6 public class ExprMain
7 {
8     public static void main(String[] args) throws Exception
9     {
10         String inputFile = null;
11
12         // Create the input stream.
13         if (args.length > 0) inputFile = args[0];
14         InputStream is = System.in;
15         if (inputFile != null) is = new FileInputStream(inputFile);
16
17         // Create the character stream from the input stream.
18         CharStream cs = CharStreams.fromStream(is);
19
20         // Create a lexer which scans the character stream
21         // to create a token stream.
22         ExprLexer lexer = new ExprLexer(cs);
23         CommonTokenStream tokens = new CommonTokenStream(lexer);
24     }
25 }
```

```
<terminated> Expr [Java Application] /Library/Java/JavaVirtualMachines/jdk-13.0.1.jdk/Contents/Home/bin/java (Jan 7, 2020, 5:59:59 PM)
Tokens:
[[@0,0:2='100',<9>,1:0}
[[@1,3:3='+',<4>,1:3}
[[@2,4:4='2',<9>,1:4}
[[@3,5:5='*',<2>,1:5}
[[@4,6:7='34',<9>,1:6}
[[@5,9:9='\n',<10>,1:9}
[[@6,10:9='<EOF>',<-1>,2:0}

Parse tree (Lisp format):
(prog (stat (expr (expr 100) + (expr (expr 2) * (expr 34)))) \n))
```

Figure 13. Output from running the generated compiler to compile `sample.expr`.

If you plan to do only Java development, then you have successfully installed ANTL4 and the ANTLR 4 Eclipse plugin.

If you plan to do C++ development, then next read the tutorial “Install and Configure ANTLR 4 for C++” (<http://www.cs.sjsu.edu/~mak/tutorials/InstallANTLR4Cpp.pdf>).