Creating SimpleLanguage Runtime

Updated: 7Jun2024 Dan Houck

This repository demonstrates the use of ANTLR, C#, and the Visitor pattern to create simple languages and their interpreter implementation with the Visitor pattern (that ANTLR generates).

Software and Tools: Everything is 64 bit (I think):

ANTLR 4.13.1

Java (for ANTLR) Version 22.0.1 (April 2024)

VS2022 C# 8 Console project

Installing the Pieces from scratch

It's easiest just to clone this repository, but here are the steps I used to create it from scratch.

To build a C# .NET 8 program that parses and runs a simple language defined in a `.g4` file using ANTLR (ANother Tool for Language Recognition), we'll proceed incrementally. Here's the first phase breakdown:

1. Define the Grammar: Create the `.g4` file to define the language syntax.

2. Set Up the Project: Create a new C# .NET 8 project and configure it to use ANTLR.

3. Generate the Parser and Lexer: Use ANTLR to generate the necessary classes.

4. Implement the Interpreter: Write the code to interpret and execute the parsed language.

Step 1: Define the Grammar

Created a file named `SimpleLang.g4` that is kept under the Grammar folder. It looks something like this early version:

~~~
```g4
grammar SimpleLang;

prog:   stat+ ;

stat:   varDecl
    |   expr
~~~

```
    |   methodDecl
    ;

varDecl: type ID ('=' expr)? ';' ;

type:    'int'
    |    'string'
    |    'int' '[' dimensions ']'
    |    'string' '[' dimensions ']'
    ;

dimensions:   INT (',' INT)* ;

methodDecl: type ID '(' argList? ')' block ;

argList: type ID (',' type ID)* ;

block: '{' stat* '}' ;

expr:    expr ('*'|'/') expr
    |    expr ('+'|'-') expr
    |    ID
    |    INT
    |    STRING
    |    ID '[' expr ']'
    ;

ID:      [a-zA-Z_][a-zA-Z_0-9]* ;
INT:     [0-9]+ ;
STRING: '"' .*? '"' ;

WS:      [ \t\r\n]+ -> skip ;
```

Step 2: Set Up the Project

1. Create a new .NET 8 Console Application:

```sh

dotnet new console -n SimpleLangInterpreter

cd SimpleLangInterpreter

```

2. Add the ANTLR NuGet package:

    ```sh
    ```

```
dotnet add package Antlr4.Runtime.Standard
```

3. Create a directory for the grammar file:

```sh
mkdir Grammar
```

4. Place the `SimpleLang.g4` file in the `Grammar` directory.

Step 3: Generate the Parser and Lexer

1. Install the ANTLR tool:

```sh
dotnet tool install -g Antlr4BuildTasks
```

2. Generate the parser and lexer classes by running antlr from the Grammar folder:

```
java -jar antlr-4.13.1-complete.jar -Dlanguage=CSharp -visitor -o Generated
SimpleLang.g4
```

This will generate the parser and lexer classes in the `Generated` directory.

Once I check them, I move the C# (*.cs) files to the top level Generated folder.

3. Include the generated classes in your project:

Edit your `.csproj` file to include the generated files:

```xml
<ItemGroup>
  <Compile Include="Generated\\*.cs" />
</ItemGroup>
```

Step 4: Implement the Interpreter with the Visitor Pattern

Under the Grammar folder is a Generated folder

1. Create an interpreter class:

  Create a new file named `SimpleLangInterpreter.cs` and implement a basic interpreter, including the very important SimpleLangCustomVisitor.cs which are mainly methods to override the visitor classes **so we never have to modify the ANTLR-generated code.**

2. Run the project:

~~~
```sh
dotnet run
```
~~~

This basic setup allows you to parse and process simple variable declarations and expressions. We'll extend the interpreter to handle arrays and methods in subsequent phases.

After a runtime error, Jarvis suggested doing this:

The error you're encountering suggests there might be an issue with the directory structure or the specific version of the tool. Let's address this step by step. Here are a few alternative steps you can follow:

Step 1: Install ANTLR via NuGet

Instead of using the `antlr4` tool directly, you can install the ANTLR NuGet package and use it within your project:

1. Install the ANTLR4.Runtime.Standard NuGet package:

```
dotnet add package Antlr4.Runtime.Standard
```

Step 2: Use the ANTLR Tool Directly

Download the ANTLR tool manually and use it to generate the parser and lexer classes:

1. Download the ANTLR tool:

   Download the ANTLR tool JAR file from the official [ANTLR website](https://www.antlr.org/download.html).

2. Generate the parser and lexer:

   Use the downloaded JAR file to generate the parser and lexer classes. Assuming you downloaded `antlr-4.13.1-complete.jar`, you can use the following command:

   java -jar antlr-4.9.3-complete.jar -Dlanguage=CSharp -o Generated Grammar/SimpleLang.g4

   ```

## Step 3: Include the Generated Files in Your Project

Make sure the generated files are included in your project:

1. Add the generated files:

   Copy the generated files to the top-level project directory “Generated”

2. Make sure your .csproj file includes the generated files:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net8.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>

  <ItemGroup>
```
```

```
    <Compile Remove="Grammar\Generated\**" />
    <EmbeddedResource Remove="Grammar\Generated\**" />
    <None Remove="Grammar\Generated\**" />
  </ItemGroup>

  <ItemGroup>
    <Compile Remove="Grammar\temp.cs" />
    <Compile Remove="SimpleLangCustomVisitor-Old05Jun2024.cs" />
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="Antlr4.Runtime.Standard" Version="4.13.1" />
  </ItemGroup>

</Project>
```

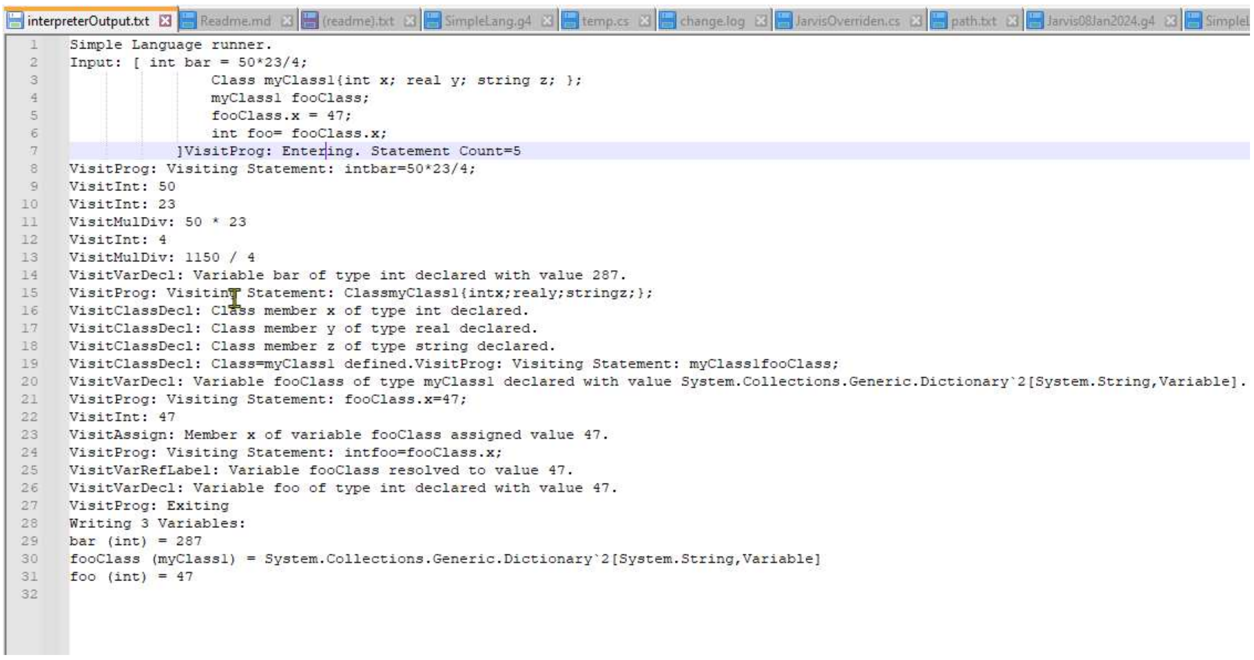
3. Implement the Interpreter

Follow the steps provided previously to implement the interpreter in C#.

Running the Project

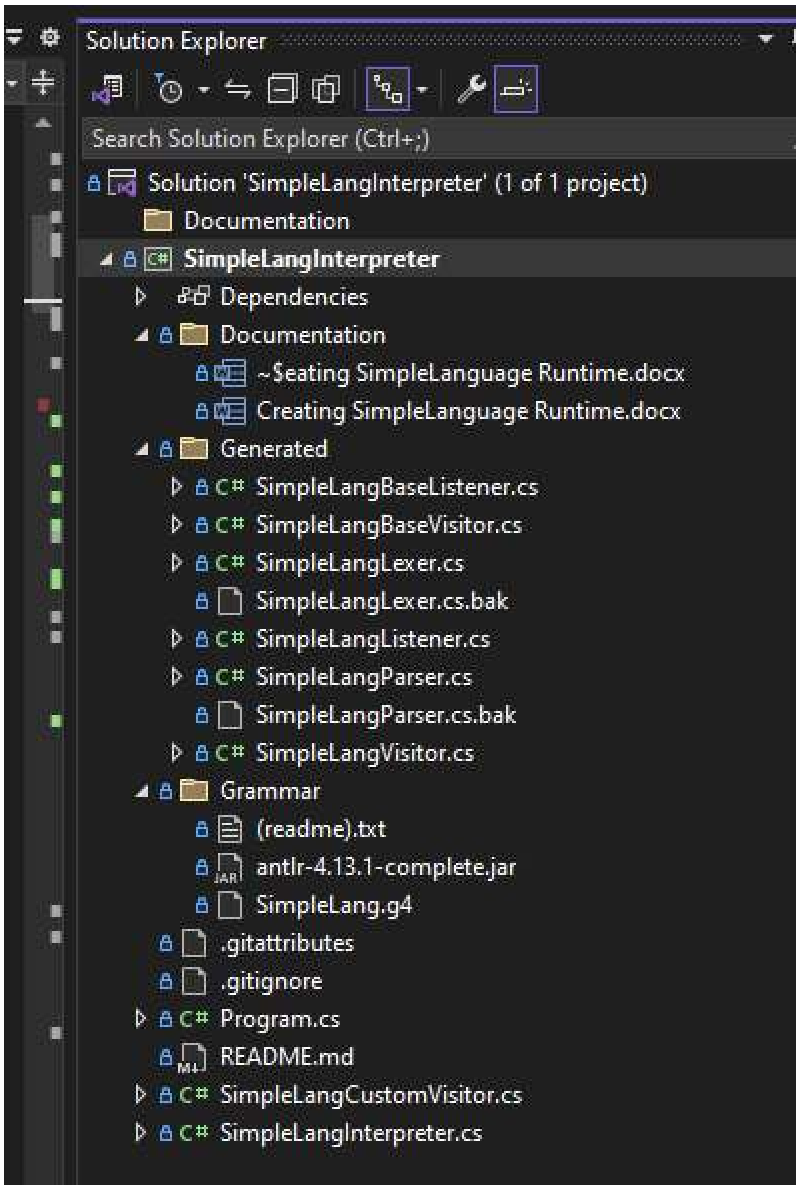
After setting up everything, you should be able to run your project without encountering the directory not found error:

The code redirects the console write to a folder at c:\Temp\interpreterOutput.txt



```
Simple Language runner.
Input: [ int bar = 50*23/4;
                Class myClass1{int x; real y; string z; };
                myClass1 fooClass;
                fooClass.x = 47;
                int foo= fooClass.x;
            ]VisitProg: Entering. Statement Count=5
VisitProg: Visiting Statement: intbar=50*23/4;
VisitInt: 50
VisitInt: 23
VisitMulDiv: 50 * 23
VisitInt: 4
VisitMulDiv: 1150 / 4
VisitVarDecl: Variable bar of type int declared with value 287.
VisitProg: Visiting Statement: ClassmyClass1{intx;realy;stringz;};
VisitClassDecl: Class member x of type int declared.
VisitClassDecl: Class member y of type real declared.
VisitClassDecl: Class member z of type string declared.
VisitClassDecl: Class=myClass1 defined.VisitProg: Visiting Statement: myClass1fooClass;
VisitVarDecl: Variable fooClass of type myClass1 declared with value System.Collections.Generic.Dictionary`2[System.String,Variable].
VisitProg: Visiting Statement: fooClass.x=47;
VisitInt: 47
VisitAssign: Member x of variable fooClass assigned value 47.
VisitProg: Visiting Statement: intfoo=fooClass.x;
VisitVarRefLabel: Variable fooClass resolved to value 47.
VisitVarDecl: Variable foo of type int declared with value 47.
VisitProg: Exiting
Writing 3 Variables:
bar (int) = 287
fooClass (myClass1) = System.Collections.Generic.Dictionary`2[System.String,Variable]
foo (int) = 47
```

Here’s a screenshot of the project:



```
Solution 'SimpleLangInterpreter' (1 of 1 project)
    Documentation
  SimpleLangInterpreter
    Dependencies
    Documentation
      ~$eating SimpleLanguage Runtime.docx
      Creating SimpleLanguage Runtime.docx
    Generated
      SimpleLangBaseListener.cs
      SimpleLangBaseVisitor.cs
      SimpleLangLexer.cs
      SimpleLangLexer.cs.bak
      SimpleLangListener.cs
      SimpleLangParser.cs
      SimpleLangParser.cs.bak
      SimpleLangVisitor.cs
    Grammar
      (readme).txt
      antlr-4.13.1-complete.jar
      SimpleLang.g4
    .gitattributes
    .gitignore
    Program.cs
    README.md
    SimpleLangCustomVisitor.cs
    SimpleLangInterpreter.cs
```

Developing the Code

The workflow is:

- Update the G4 file
- Generate the Visitor code to the Grammar > Generated folder
- If you are happy with the generated code, move it to the top-level Generated folder
- Build/Run the Interpreter

Under the Grammar folder is a (readme).txt file. Here are the contents:

```
When the grammar changes, we must modify the .g4 and regenerate all the code
located in the folder 'Generated'

This is done with a command line like this:

Move to where the Grammar folder is and run ANTLR (which was placed within
the Grammar folder)

cd c:\Users\dan_h\source\repos\SimpleLangInterpreter\Grammar

And then move the generated code up the our solutions 'Generated' folder
using File Explorer,
thus replacing the files under Generated. Of course, you can organize files
as you wish.
I just found this to be the best for me.

The change to .g4 may change what the Visitor code needs to be, so then alter
the 'Custom'
visitor code, such as 'SimpleLangCustomVisitor.cs'

*NEVER* manually alter the generated code!

C:\Users\dan_h\source\repos\SimpleLangInterpreter\Grammar>java -version
java version "22.0.1" 2024-04-16
Java(TM) SE Runtime Environment (build 22.0.1+8-16)
Java HotSpot(TM) 64-Bit Server VM (build 22.0.1+8-16, mixed mode, sharing)
```

From a command line I ran the java program as show above which placed my generated files in a "Generated" folder underneath my Grammar.

Directory of C:\Users\dan_h\source\repos\SimpleLangInterpreter\Grammar\Generated

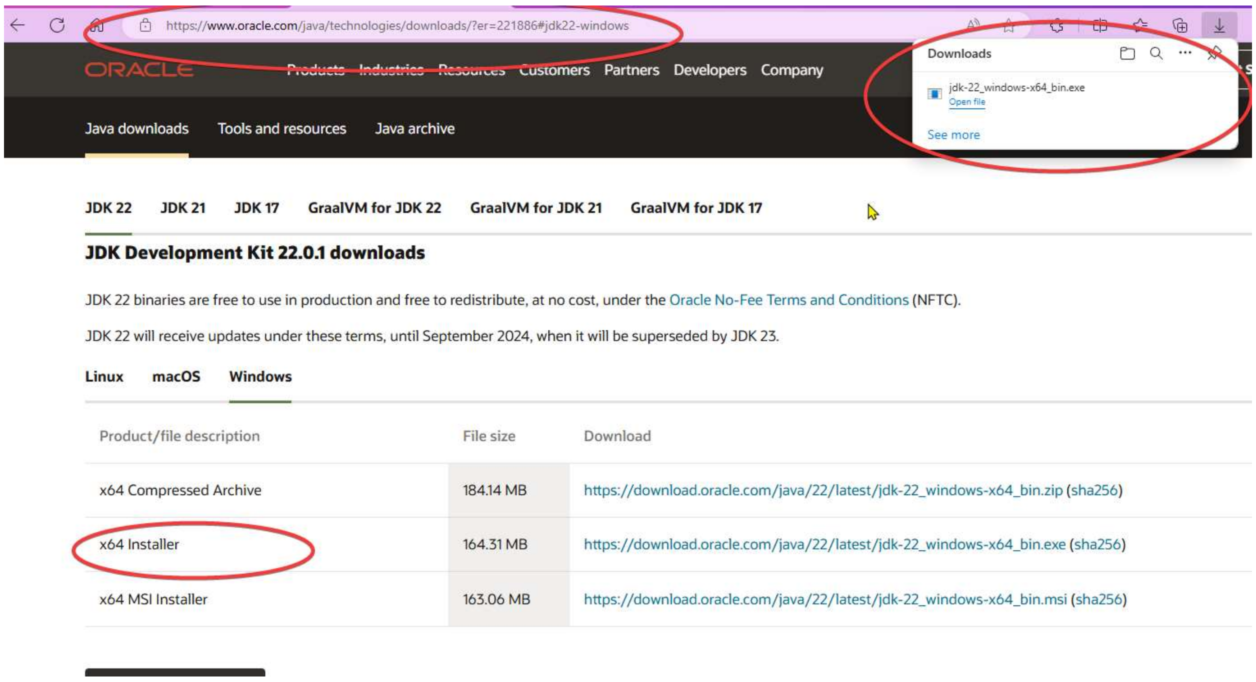
```
05/21/2024  08:44 AM    <DIR>          .
06/05/2024  05:04 AM    <DIR>          ..
06/05/2024  05:38 AM             5,709 SimpleLang.interp
06/05/2024  05:38 AM               352 SimpleLang.tokens
06/05/2024  05:38 AM            15,146 SimpleLangBaseListener.cs
06/05/2024  05:38 AM            12,350 SimpleLangBaseVisitor.cs
06/05/2024  05:38 AM             6,743 SimpleLangLexer.cs
06/05/2024  05:38 AM             4,760 SimpleLangLexer.interp
06/05/2024  05:38 AM               352 SimpleLangLexer.tokens
06/05/2024  05:38 AM            11,144 SimpleLangListener.cs
06/05/2024  05:38 AM            50,125 SimpleLangParser.cs
06/05/2024  05:38 AM             7,158 SimpleLangVisitor.cs
              10 File(s)        113,839 bytes
               2 Dir(s)  138,371,670,016 bytes free
```

This is my preferred way, as it gives me the opportunity to look at the generated files before moving them up to the Generated folder in my solution.

Errors and Resolution

Exception in thread "main" java.lang.UnsupportedClassVersionError: org/antlr/v4/Tool has been compiled by a more recent version of the Java Runtime (class file version 55.0), this version of the Java Runtime only recognizes class file versions up to 52.0

In my case, I had more than one Java installed. Doing a “java -version” revealed version 1.8.0_411


https://www.oracle.com/java/technologies/downloads/?er=221886#jdk22-windows
ORACLE
Products Industries Resources Customers Partners Developers Company
Java downloads
Tools and resources
Java archive
Downloads
jdk-22_windows-x64_bin.exe
Open file
See more
JDK 22
JDK 21
JDK 17
GraalVM for JDK 22
GraalVM for JDK 21
GraalVM for JDK 17

JDK Development Kit 22.0.1 downloads

JDK 22 binaries are free to use in production and free to redistribute, at no cost, under the Oracle No-Fee Terms and Conditions (NFTC).

JDK 22 will receive updates under these terms, until September 2024, when it will be superseded by JDK 23.

Linux macOS Windows

Product/file description	File size	Download
x64 Compressed Archive	184.14 MB	https://download.oracle.com/java/22/latest/jdk-22_windows-x64_bin.zip (sha256)
x64 Installer	164.31 MB	https://download.oracle.com/java/22/latest/jdk-22_windows-x64_bin.exe (sha256)
x64 MSI Installer	163.06 MB	https://download.oracle.com/java/22/latest/jdk-22_windows-x64_bin.msi (sha256)

Restarted the command window and now my version is “22.0.1” (16Apr2024)

Now my command line “java -jar antlr…” worked without error.