Antlr working flow with DC-Applier\_Rules

1. What is Antlr
2. How Antlr works
3. Lexer, Parser, Listener, Visitor – What it is
4. Sample application with Maven, Java
5. Sequence Diagram
6. Custom Listener implementation to execute rule on data, to return output/error
7. DC-Applier-Rules implementation
8. Sequence Diagram with custom implementation
9. **What is Antlr:**

ANTLR (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. It's widely used to build languages, tools, and frameworks. From a grammar, ANTLR generates a parser that can build and walk parses trees.

1. **How it works:**

ANTLR is a parser generator, a tool that helps you to create parsers. **A parser takes a piece of text and transforms it in an organized structure**, a *parse tree*, also known as a *Abstract Syntax Tree (AST)*. You can think of the AST as a story describing the content of the code, or also as its logical representation, created by putting together the various pieces.

Let’s take an example to understand it:

Input : 2018-May-05 14:20:18 INFO some error occurred

Parse Tree: *<datetime> <level> <message>*

It's often used to build tools and frameworks. For example, Hibernate uses ANTLR for parsing and processing HQL queries and Elasticsearch (cloud) uses it for Painless.

And **Java is just one binding.** ANTLR also offers bindings for C#, Python, JavaScript, Go, C++ and Swift.

**What you need to do to get a parse tree**:

1. Define a **Lexer** and **Parser** grammar (\*.g4 - Antlr)

>> A lexer and a parser work in sequence: the lexer scans the input and produces the matching tokens, the parser then scans the tokens and produces the parsing result.

1. Invoke ANTLR: it will generate a lexer and a parser in your target language (e.g., Java, Python, C#, JavaScript)
2. Use the generated lexer and parser: you invoke them passing the code to recognize and they return to you a parse tree
3. Write custom Listener/ Visitor or Framework to read the parse tree and evaluate data against the nodes
4. **Lexer, Parser, Listener, Visitor – What it is**

Antlr is providing you two approaches to parse/walk the tree. They are **Listener** and **Visitor** implementations**.** Both approaches have their own advantages, and choice of preferred method depends on what you are using antlr for.

**Listener**

|  |
| --- |
| public class ArithmeticParserBaseListener implements ArithmeticParserListener {  @Override public void enterExpr(ArithmeticParser.ExprContext ctx) { }  @Override public void exitExpr(ArithmeticParser.ExprContext ctx) { }  @Override public void enterOperation(ArithmeticParser.OperationContext ctx) { }  @Override public void exitOperation(ArithmeticParser.OperationContext ctx) { }  @Override public void enterEveryRule(ParserRuleContext ctx) { }  @Override public void exitEveryRule(ParserRuleContext ctx) { }  @Override public void visitTerminal(TerminalNode node) { }  @Override public void visitErrorNode(ErrorNode node) { }  } |

**Visitor**

|  |
| --- |
| public class ArithmeticParserBaseVisitor<T> extends AbstractParseTreeVisitor<T> implements ArithmeticParserVisitor<T> {  @Override public T visitExpr(ArithmeticParser.ExprContext ctx) { return visitChildren(ctx); }  @Override public T visitOperation(ArithmeticParser.OperationContext ctx) { return visitChildren(ctx); }  } |

1. **Sample application with Maven, Java and Spring boot**

First of all, let's start by adding [antlr-runtime](https://search.maven.org/classic/#search%7Cgav%7C1%7Cg%3A%22org.antlr%22%20AND%20a%3A%22antlr4-runtime%22) to our *pom.xml*:

|  |
| --- |
| <**dependency**>  <**groupId**>org.antlr</**groupId**>  <**artifactId**>antlr4-runtime</**artifactId**>  <**version**>4.7.1</**version**>  </**dependency**> |

And also the **[antlr-maven-plugin](https://search.maven.org/classic/" \l "search%7Cga%7C1%7Corg.antlr%20antlr-maven-plugin)**:

|  |
| --- |
| <**plugin**>  <**groupId**>org.antlr</**groupId**>  <**artifactId**>antlr4-maven-plugin</**artifactId**>  <**version**>4.7.1</**version**>  <**executions**>  <**execution**>  <**goals**>  <**goal**>antlr4</**goal**>  </**goals**>  <configuration>  <sourceDirectory>${antlr.dir}</sourceDirectory>  <listener>true</listener>  <visitor>false</visitor>  <arguments>  <argument>-package</argument>  <argument>org.boazglean.dabar.parser</argument>  </arguments>  </configuration>  </**execution**>  </**executions**>  </**plugin**> |

**The sample grammer file look like below:**

The generated classes look like in below screenshot (**TODO**)

The sample parser file looks like below

The sample Lexer file looks like below

The sample Listener file looks like below

1. **Sequence Diagram:**
2. **Custom Listener implementation to execute rule on data, to return output/error**

The parse tree listener (or listener) is a class that implements [callback methods](https://datacadamia.com/code/function/callback) that are called by the parser when it creates the [parse tree](https://datacadamia.com/antlr/tree).

You can overwrite this class to get information when the parser enter or exit a [rule](https://datacadamia.com/antlr/rule) (ie as found a pattern)