Introduction to ANTLR

CSEN 1002

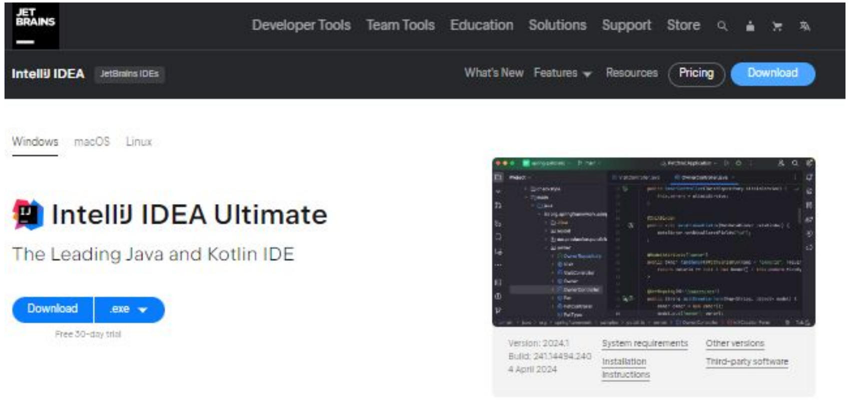
About ANTLR

- 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 parse trees.
- Twitter search uses ANTLR for query parsing, with over 2 billion queries a day. The languages for Hive and Pig, the data warehouse and analysis systems for Hadoop, both use ANTLR. Lex Machina uses ANTLR for information extraction from legal texts. Oracle uses ANTLR within SQL Developer IDE and their migration tools. NetBeans IDE parses C++ with ANTLR. The HQL language in the Hibernate object-relational mapping framework is built with ANTLR.
- From a formal language description called a grammar, ANTLR generates a parser for that language that can automatically build parse trees, which are data structures representing how a grammar matches the input. ANTLR also automatically generates tree walkers that you can use to visit the nodes of those trees to execute application-specific code.
- Aside from these big-name, high-profile projects, you can build all sorts of useful tools like configuration file readers, legacy code converters, wiki markup renderers, and JSON parsers. You can easily build little tools for object-relational database mappings, describing 3D visualizations, injecting profiling code into Java source code, or even simple DNA pattern matching.

Setting up IntelliJ & ANTLR

Download & Install

- Download IntelliJ IDEA from https://www.jetbrains.com/idea/download/
  - Both Community Edition or Ultimate would suffice
  - Ultimate requires an account for activation, you can get a free education license from https://www.jetbrains.com/shop/eform/students
- Install the appropriate version for your OS.
- Older versions might work, but we highly recommend using the latest version (2023.3.6).



IntelliJ IDEA Ultimate

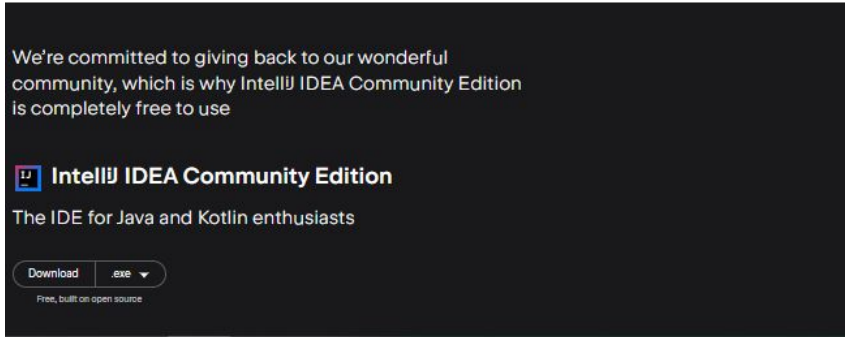
The Leading Java and Kotlin IDE

Download .exe

Free 30-day trial

Version: 2024.1
Build: 241.14494.240
4 April 2024

System requirements
Installation Instructions
Other versions
Third-party software



We're committed to giving back to our wonderful community, which is why IntelliJ IDEA Community Edition is completely free to use

IntelliJ IDEA Community Edition

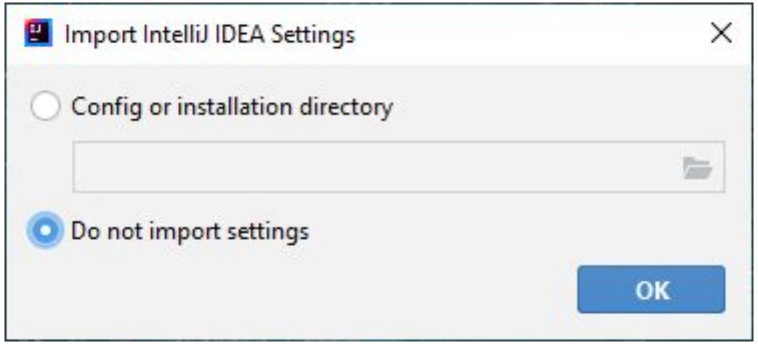
The IDE for Java and Kotlin enthusiasts

Download .exe

Free, built on open source

First Launch

- If you have an older IntelliJ installation, a “Import IntelliJ IDEA Settings” window will appear.
- If you have specific settings you need to preserve, choose “Config or installation directory” and navigate to your configuration.
- Otherwise, simply choose “Do no import settings”.

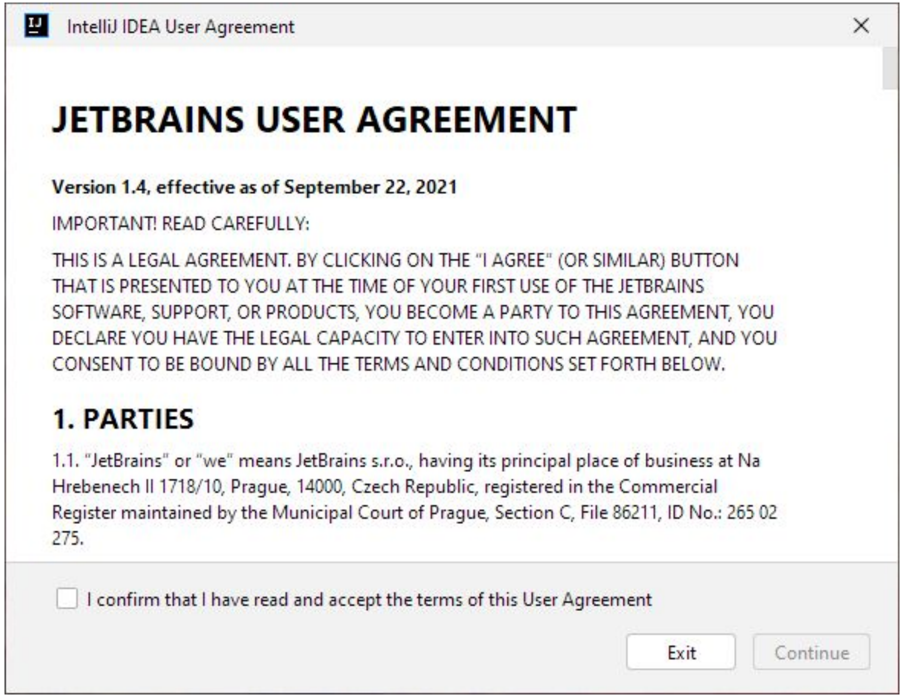

Import IntelliJ IDEA Settings

( ) Config or installation directory

(•) Do not import settings

First Launch

- In the “IntelliJ IDEA User Agreement” window, read the user agreement, and if you accept, check the”I confirm that I have read and accept the terms of this User Agreement”.
- Click “Continue”



JETBRAINS USER AGREEMENT

Version 1.4, effective as of September 22, 2021

IMPORTANT! READ CAREFULLY:

THIS IS A LEGAL AGREEMENT. BY CLICKING ON THE "I AGREE" (OR SIMILAR) BUTTON THAT IS PRESENTED TO YOU AT THE TIME OF YOUR FIRST USE OF THE JETBRAINS SOFTWARE, SUPPORT, OR PRODUCTS, YOU BECOME A PARTY TO THIS AGREEMENT, YOU DECLARE YOU HAVE THE LEGAL CAPACITY TO ENTER INTO SUCH AGREEMENT, AND YOU CONSENT TO BE BOUND BY ALL THE TERMS AND CONDITIONS SET FORTH BELOW.

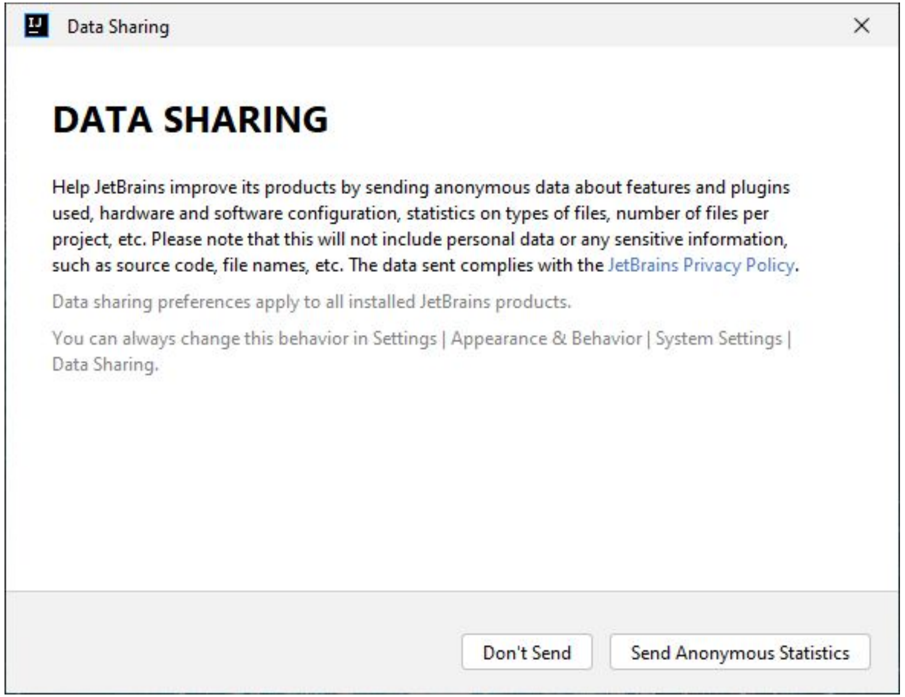
1. PARTIES

1.1. "JetBrains" or "we" means JetBrains s.r.o., having its principal place of business at Na Hrebenech II 1718/10, Prague, 14000, Czech Republic, registered in the Commercial Register maintained by the Municipal Court of Prague, Section C, File 86211, ID No.: 265 02 275.

☐ I confirm that I have read and accept the terms of this User Agreement

First Launch

- In the “Data Sharing” window, choose either option based on your preference.


Data Sharing

DATA SHARING

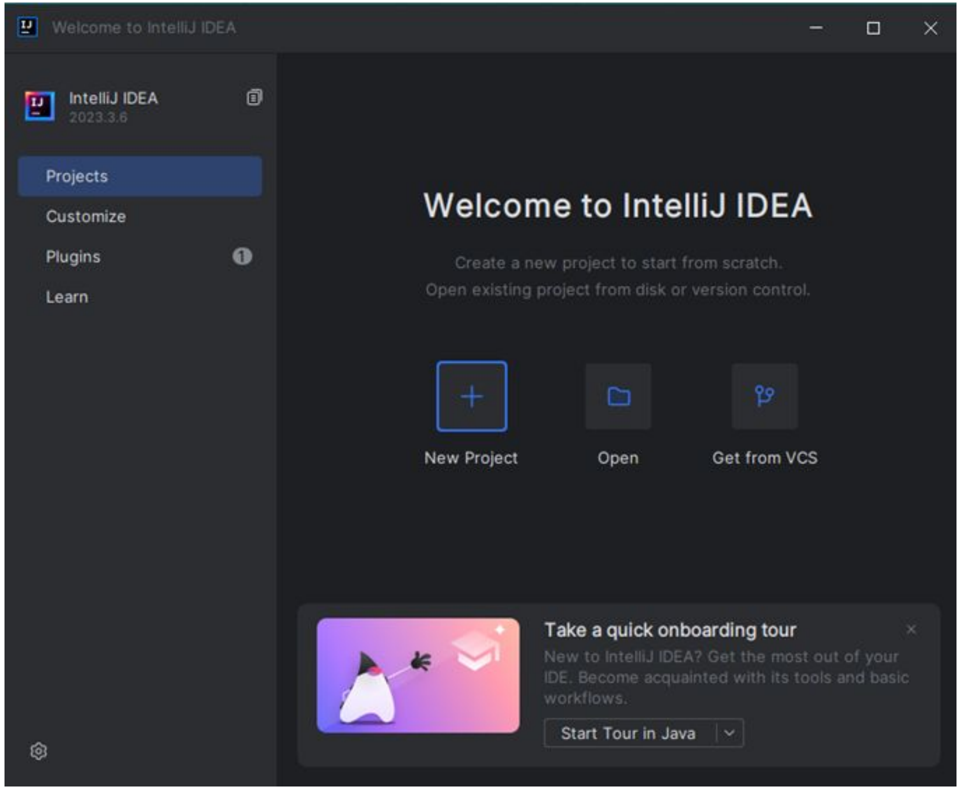
Help JetBrains improve its products by sending anonymous data about features and plugins used, hardware and software configuration, statistics on types of files, number of files per project, etc. Please note that this will not include personal data or any sensitive information, such as source code, file names, etc. The data sent complies with the JetBrains Privacy Policy.

Data sharing preferences apply to all installed JetBrains products.

You can always change this behavior in Settings | Appearance & Behavior | System Settings | Data Sharing.

First Launch

- Congratulations! You have reached the Welcome Page

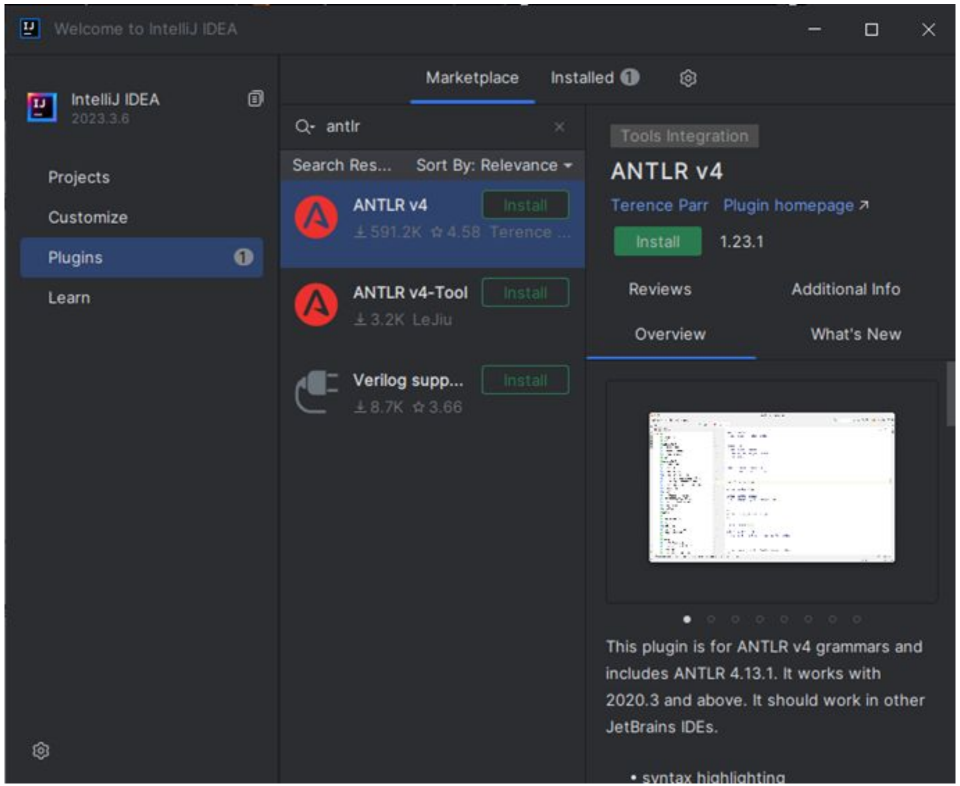


Create a new project to start from scratch.
Open existing project from disk or version control.

New to IntelliJ IDEA? Get the most out of your IDE. Become acquainted with its tools and basic workflows.

Install ANTLR Plugin

- Select “Plugins” from the sidebar
- Type “ANTLR v4” into the search bar
- Click “Install”
- A “Third-Party Plugins Notice” will pop up. Click “Accept”
- Click “Restart IDE”

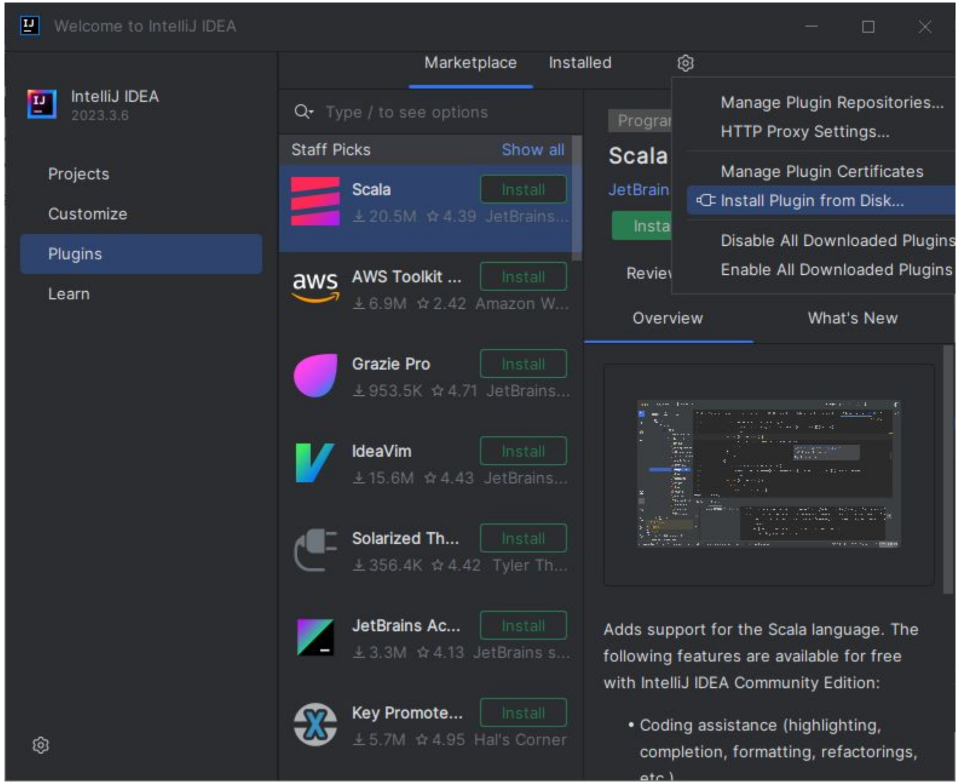


This plugin is for ANTLR v4 grammars and includes ANTLR 4.13.1. It works with 2020.3 and above. It should work in other JetBrains IDEs.

• syntax highlighting

Install ANTLR Plugin (Offline Method)

- If the online plugin installation fails, you can attempt an offline installation.
- Download the appropriate plugin version (1.23.1) from https://plugins.jetbrains.com/plugin/7358-antlr-v4/versions
  - Do **not** unzip the downloaded file
- Select "Plugins" from the sidebar
- Click the "Cog"
- Select "Install Plugin from Disk…"
- Navigate to downloaded zip file and select it
- Click "Restart IDE"



Adds support for the Scala language. The following features are available for free with IntelliJ IDEA Community Edition:

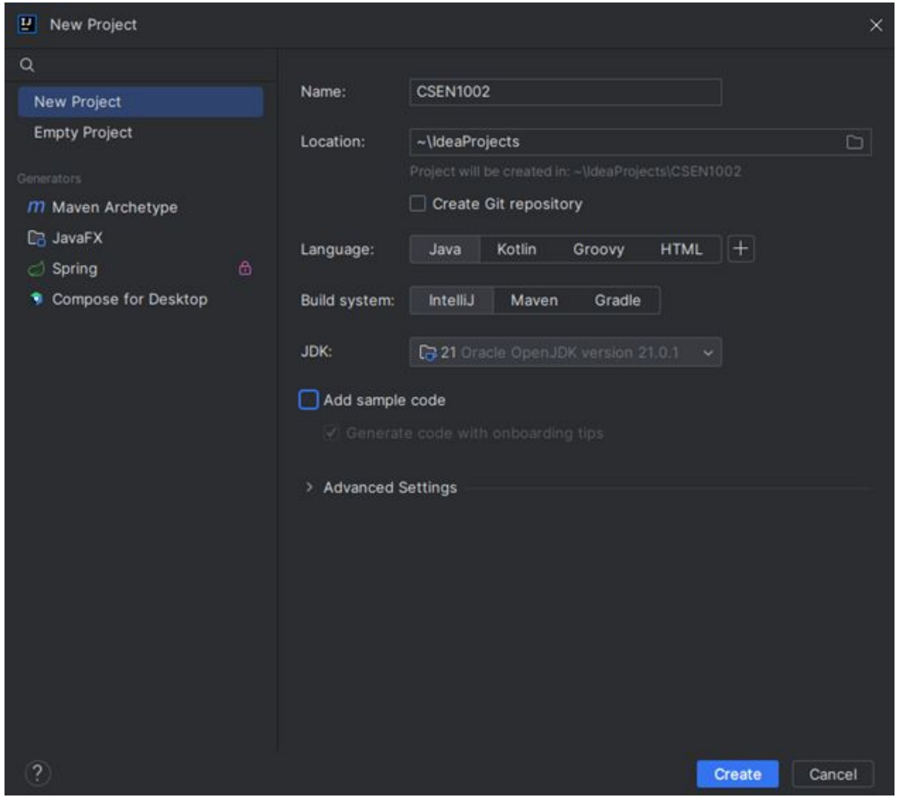
- Coding assistance (highlighting, completion, formatting, refactorings, etc.)

Creating a New Java Project

Please note that the following steps must be repeated for **every** java project which uses ANTLR and JUnit.

New Project

- By default the “Projects” sidebar item should be selected
- Click “New Project”
- Make sure that the select sidebar item is “New Project”
- For the “Name”, type your preferred project name.
- For the “Location”, select your preferred path. This is where your project will be created.
- Leave “Language” as “Java” and “Build System” as IntelliJ.
- If your JDK configured correctly it should be set in the “JDK” field. If not, you can either
  - Download JDK from https://www.oracle.com/eg/java/technologies/downloads/#java21. Click “Add JDK..” and select the directory.
  - Select “Download JDK..” and choose “21” as the version and any vendor.
- Click “Create”



Name: CSEN1002

Location: ~\IdeaProjects

Project will be created in: ~\IdeaProjects\CSEN1002

Create Git repository

Language: Java

Build system: IntelliJ

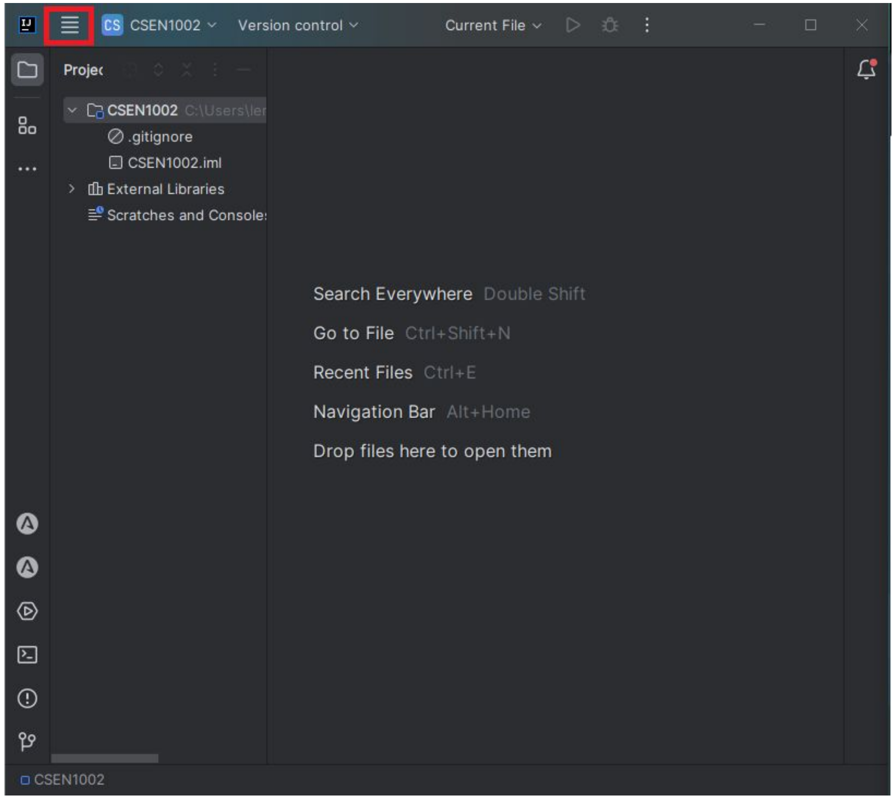
JDK: 21 Oracle OpenJDK version 21.0.1

Add sample code

Generate code with onboarding tips

Add ANTLR as a Dependency

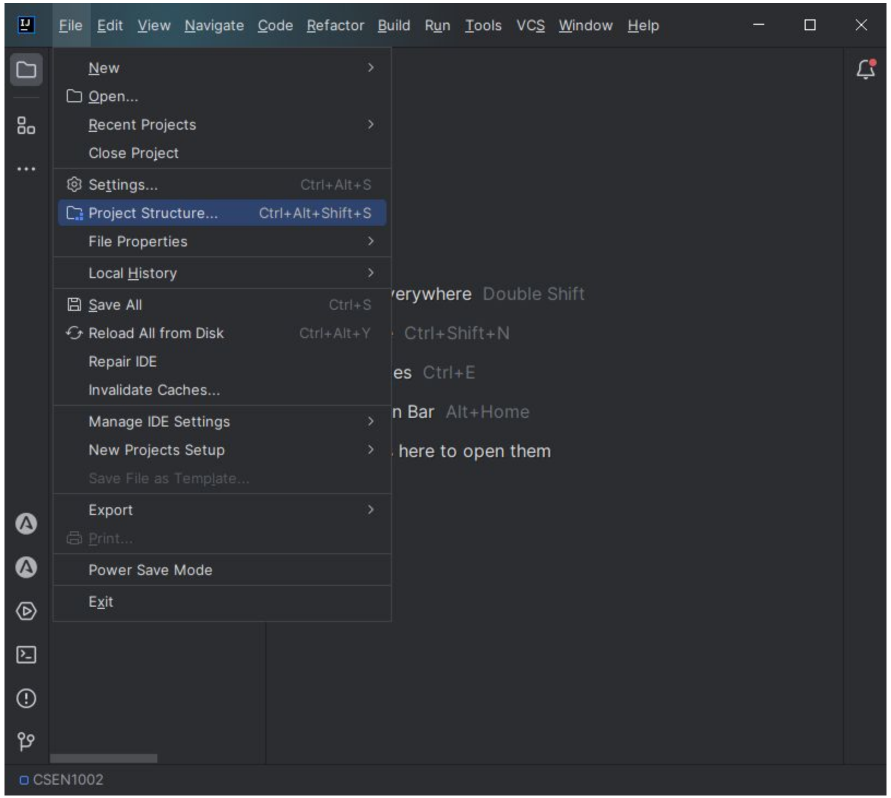
- Please note that the following steps must be repeated for **every** java project which uses ANTLR.
- Click “File” then “Project Structure…”



Search Everywhere Double Shift
Go to File Ctrl+Shift+N
Recent Files Ctrl+E
Navigation Bar Alt+Home
Drop files here to open them

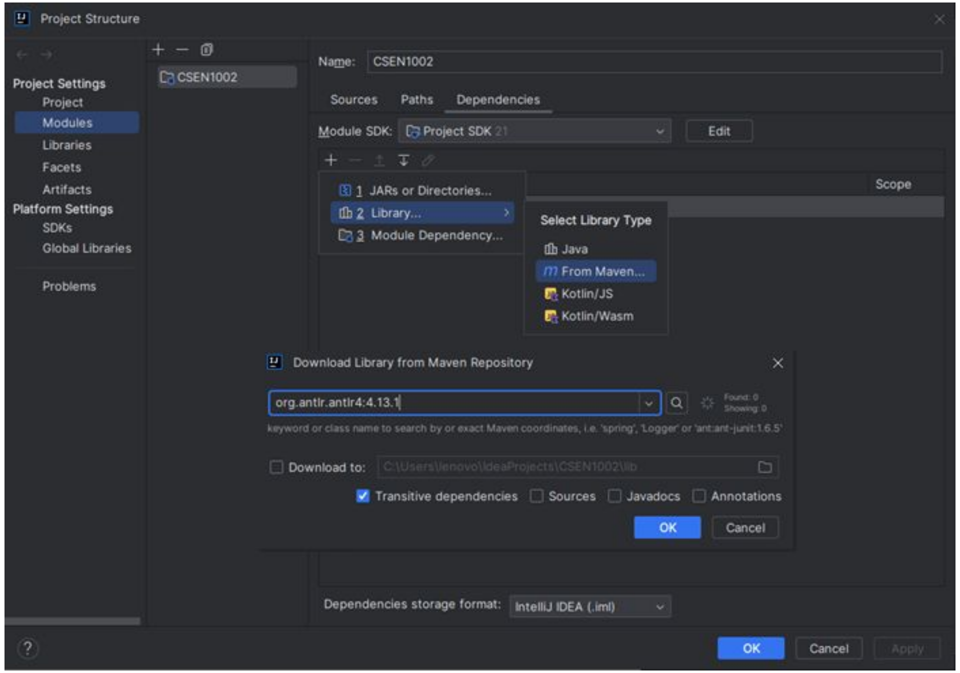
Add ANTLR as a Dependency

- Please note that the following steps must be repeated for **every** java project which uses ANTLR.
- Click “File” then “Project Structure…”



Add ANTLR as a Dependency (Maven)

- From the sidebar, select “Modules”
- On the right panel, select the “Dependencies” tab
- Click the “+” symbol in the **right** panel
- Click “Library..” then “From Maven…”
- A dialog box should appear
- Type “org.antlr:antlr4:4.13.1” into the search box.
- Click “OK”



Name: CSEN1002

Module SDK: Project SDK 21

org.antlr.antlr4:4.13.1

keyword or class name to search by or exact Maven coordinates, i.e. 'spring', 'Logger' or 'ant:ant-junit:1.6.5'

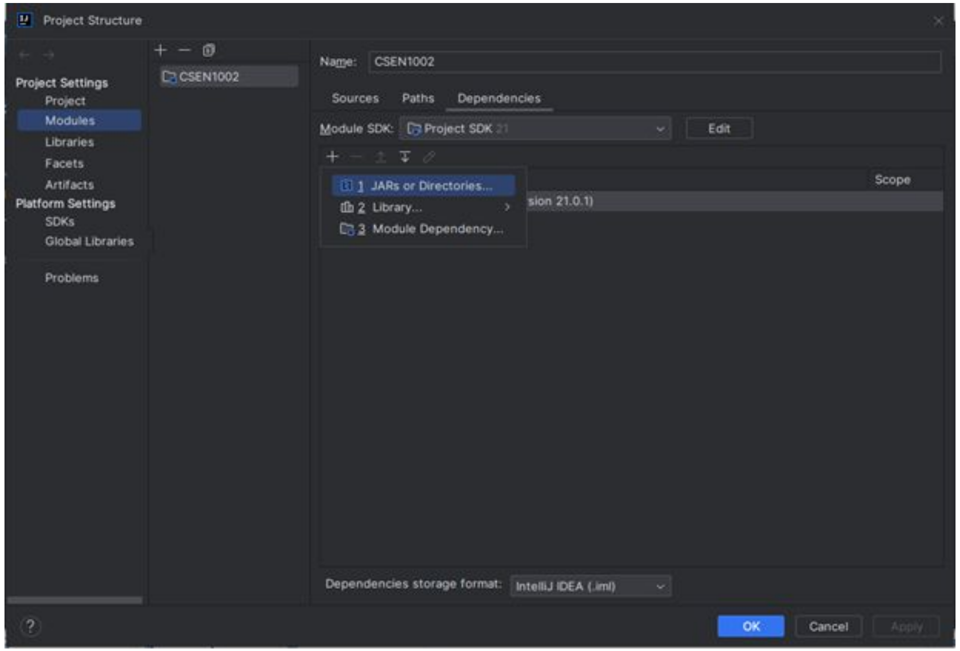
☐ Download to: C:\Users\lenovo\IdeaProjects\CSEN1002\lib

☑ Transitive dependencies ☐ Sources ☐ Javadocs ☐ Annotations

Dependencies storage format: IntelliJ IDEA (.iml)

Add ANTLR as a Dependency (Local)

- Download the complete ANTLR binaries from https://www.antlr.org/download/antlr-4.13.1-complete.jar
- From the sidebar, select “Modules”
- On the right panel, select the “Dependencies” tab
- Click the “+” symbol in the **right** panel
- Click “JARs or Directories…”
- Navigate to downloaded jar file and select it
- Click “OK”



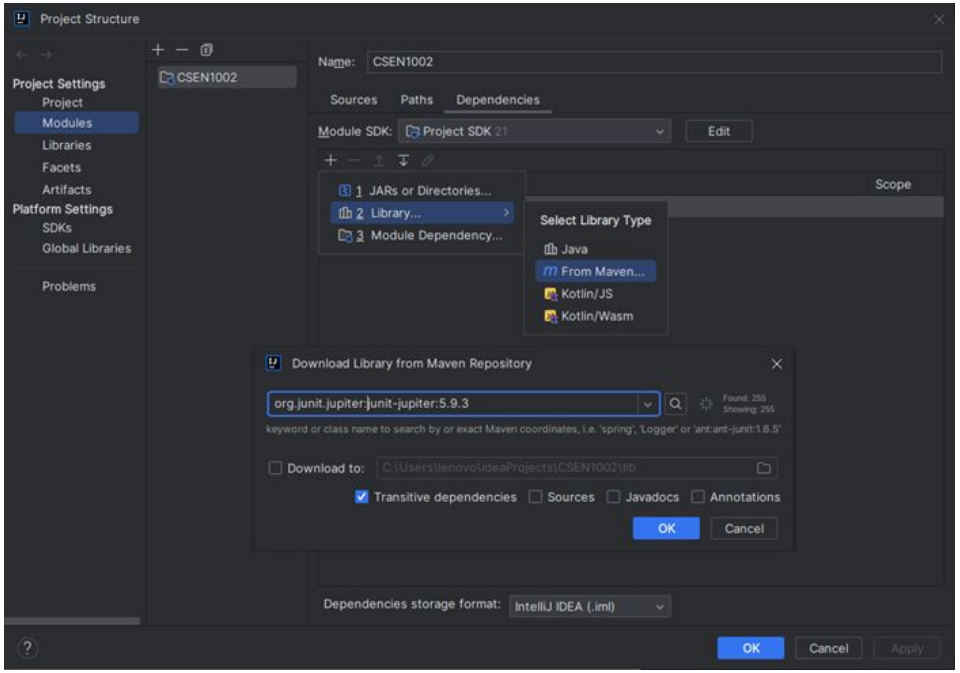
Name: CSEN1002

Module SDK: Project SDK 21

Dependencies storage format: IntelliJ IDEA (.iml)

Add JUnit as a Dependency

- From the sidebar, select “Modules”
- On the right panel, select the “Dependencies” tab
- Click the “+” symbol in the **right** panel
- Click “Library..” then “From Maven…”
- A dialog box should appear
- Type “org.junit.jupiter:junit-jupiter:5.9.3” into the search box.
- Click “OK”



Name: CSEN1002

Module SDK: Project SDK 21

org.junit.jupiter:junit-jupiter:5.9.3

keyword or class name to search by or exact Maven coordinates, i.e. 'spring', 'Logger' or 'ant:ant-junit:1.6.5'

☐ Download to: C:\Users\lenovo\IdeaProjects\CSEN1002\lib

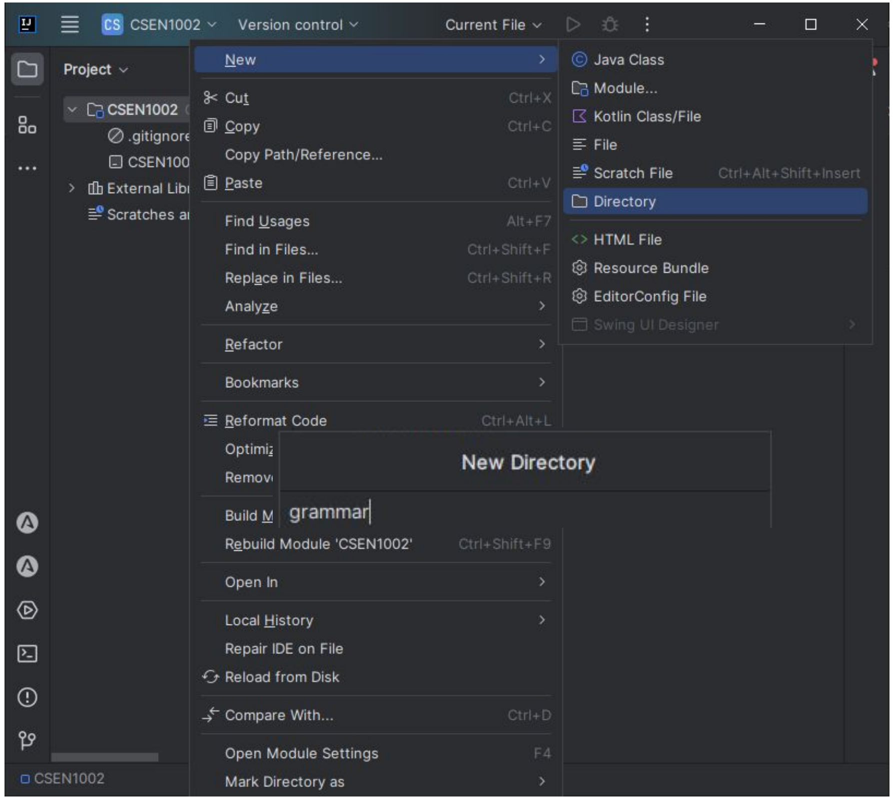
☑ Transitive dependencies ☐ Sources ☐ Javadocs ☐ Annotations

Dependencies storage format: IntelliJ IDEA (.iml)

Your First ANTLR Grammar

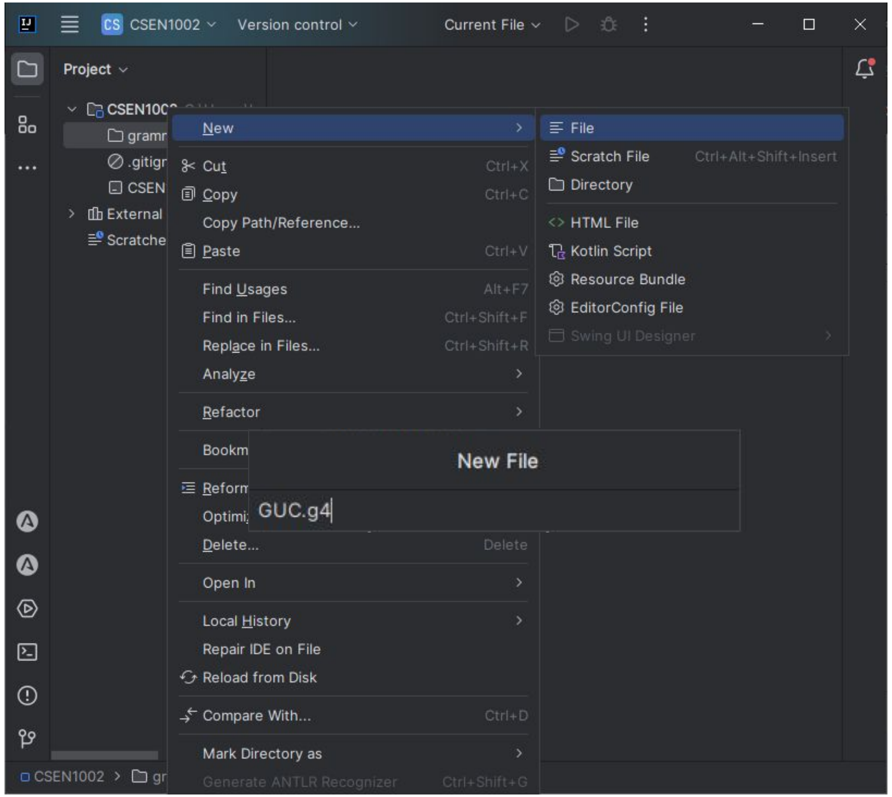
Creating the Grammars Directory

- Right click on the root project directory
- Select "New"
- Select "Directory"
- Type the name of the directory in which the grammar files will be created (e.g. grammars)



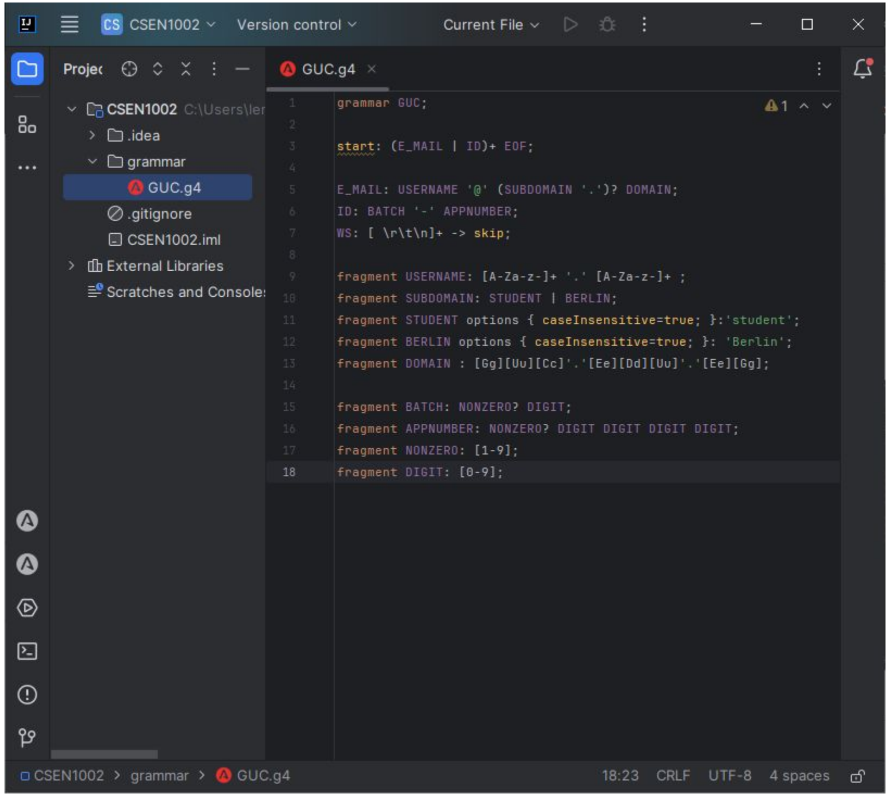
Creating the GUC.g4 File

- Right click on the ‘grammars’ directory
- Select ‘New’
- Select ‘File’
- Type the grammar name (case sensitive) followed by ‘.g4’ (i.e. GUC.g4)



Example Grammar

```
grammar GUC;

start: (E_MAIL | ID)+ EOF;

E_MAIL: USERNAME '@' (SUBDOMAIN '.')? DOMAIN;
ID: BATCH '-' APPNUMBER;
WS: [ \r\t\n]+ -> skip;

fragment USERNAME: [A-Za-z-]+ '.' [A-Za-z-]+ ;
fragment SUBDOMAIN: STUDENT | BERLIN;
fragment STUDENT options { caseInsensitive=true; }:'student';
fragment BERLIN options { caseInsensitive=true; }: 'Berlin';
fragment DOMAIN : [Gg][Uu][Cc]'.'[Ee][Dd][Uu]'.'[Ee][Gg];

fragment BATCH: NONZERO? DIGIT;
fragment APPNUMBER: NONZERO? DIGIT DIGIT DIGIT DIGIT;
fragment NONZERO: [1-9];
fragment DIGIT: [0-9];
```



```
grammar GUC;

start: (E_MAIL | ID)+ EOF;

E_MAIL: USERNAME '@' (SUBDOMAIN '.')? DOMAIN;
ID: BATCH '-' APPNUMBER;
WS: [ \r\t\n]+ -> skip;

fragment USERNAME: [A-Za-z-]+ '.' [A-Za-z-]+ ;
fragment SUBDOMAIN: STUDENT | BERLIN;
fragment STUDENT options { caseInsensitive=true; }:'student';
fragment BERLIN options { caseInsensitive=true; }: 'Berlin';
fragment DOMAIN : [Gg][Uu][Cc]'.'[Ee][Dd][Uu]'.'[Ee][Gg];

fragment BATCH: NONZERO? DIGIT;
fragment APPNUMBER: NONZERO? DIGIT DIGIT DIGIT DIGIT;
fragment NONZERO: [1-9];
fragment DIGIT: [0-9];
```

Grammar Structure

- All statements in ANTLR have to end with a ‘`;`’.
- An ANTLR file starts with a grammar declaration.
- A grammar declaration is in the form of ‘`grammar Name;`’
- The file name containing grammar `X` must be called `X.g4`.
- There are single-line, multiline, and Javadoc-style comments.
- Following that, the grammar consists of rules in the form of:
    - `ruleName : alternative1 | ... | alternativeN ;`

Parser Rules

- Parsers consist of a set of parser rules either in a parser or a combined grammar. A Java application launches a parser by invoking the rule function, generated by ANTLR, associated with the desired start rule.
- ANTLR uses parser rules to generate a parse tree, which can be used for a multitude of purposes.
- Parser rules are the nodes in the parse tree.
- We will focus more on parser rules in the upcoming labs.

Lexer Rules

- Lexer rules specify token definitions.
- ANTLR splits the input string into tokens based on these rules.
- These rules represent the leaves of the parse tree.
- Token names must **begin with an uppercase letter**, which distinguishes them from parser rule names.
    - `TokenName : alternative1 | ... | alternativeN ;`
- Rules can also have alternatives separated by the “|”

Syntax	Description
'literal'	Match that character or sequence of characters. E.g., 'while' or '='. ANTLR does not distinguish between character and string literals as most languages do.
[char set]	Match one of the characters specified in the character set. Interpret x-y as the set of characters between range x and y, inclusively.
'x'..'y'	Match any single character between range x and y, inclusively. E.g., 'a'..'z'. 'a'..'z' is identical to [a-z].
T	Invoke lexer rule T; recursion is allowed in general, but not left recursion. T can be a regular token or fragment rule.
.	The dot is a single-character wildcard that matches any single character.
~x	Match any single character not in the set described by x. Set x can be a single character literal, a range, or a subrule set like ~('x'|'y'|'z') or ~[xyz].

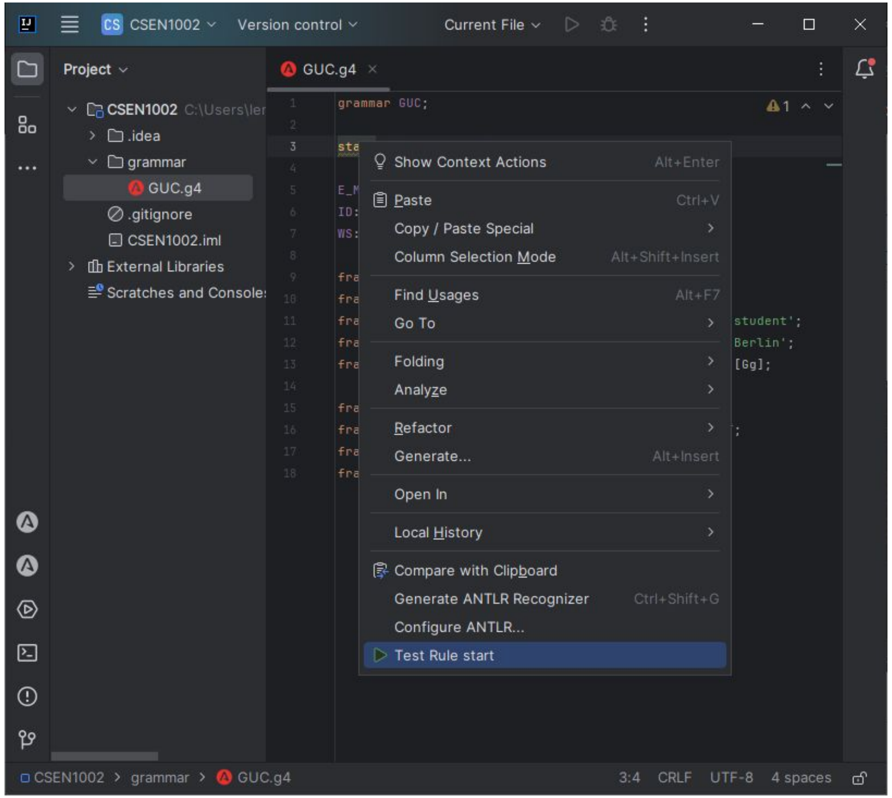
Lexer Rules

- You can also define rules that are not tokens but rather aid in the recognition of tokens. These `fragment` rules do not result in tokens visible to the parser
    - `fragment HelperTokenRule : alternative1 | ... | alternativeN ;`
    - For example, `DIGIT` is a pretty common fragment rule:

        -
          ```
          INT : DIGIT+ ; // references the DIGIT helper rule
          fragment DIGIT : [0-9] ; // not a token by itself
          ```
- `caseInsensitive` defines if the current lexer rule is case-insensitive. The argument can be `true` or `false`.
    - `GUC options { caseInsensitive=true; } : 'GUC'; // Matches ‘guc’, ‘Guc’, …, ‘GUC’`
- A `skip` command tells the lexer to get another token and throw out the current text.
    - `WS : [ \t]+ -> skip ; // toss out whitespace`

ANTLR Preview

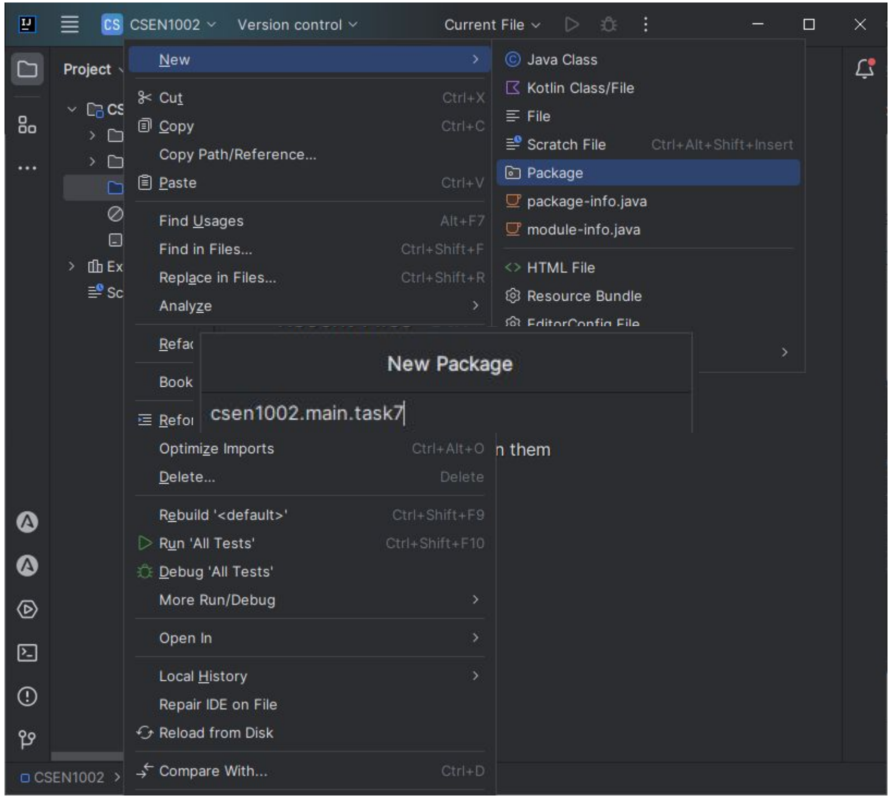
- Right Click on the rule you want to test (e.g. start)
- Select “Test Rule {ruleName}”
- The “ANTLR Preview” pane will appear.
    - This window can be shown/hidden by clicking the “ANTLR Preview” button in the bottom bar.
- Select the “Input” radio button to test your rule with input from the textbox below
- Type your test input and observe the output parse tree.
- To check tokens, hold “Ctrl” and hover over the input text to view the lexeme and the token type.
- To check the parse, hold “Alt” and hover over the input text to view the substring and the parser rule.



Preparing your Project for Task 7

Creating the Main Package

- Right click on the “src” directory in your project
- Select “New”
- Select package
- Type the package name “csen1002.main.task7”

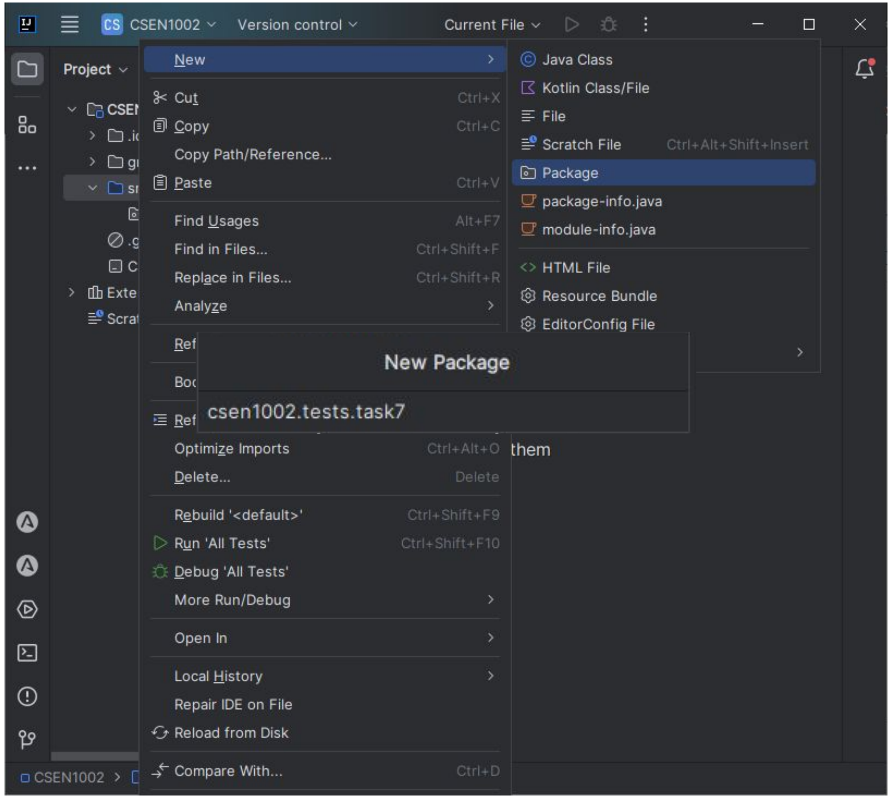


New Package

csen1002.main.task7

Creating the Tests Package

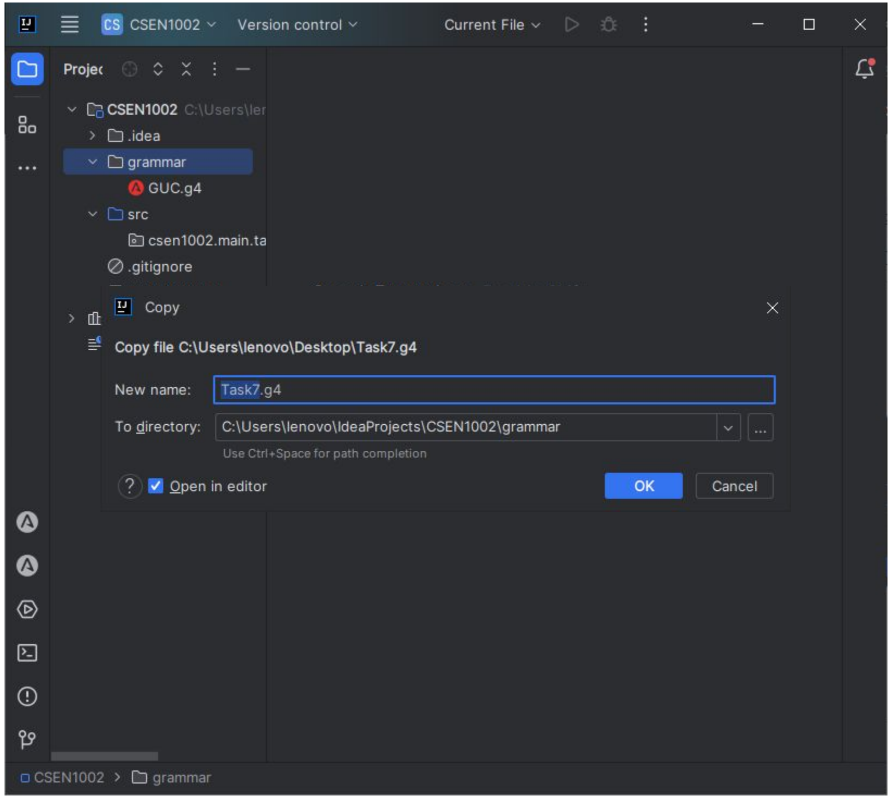
- Right click on the “src” directory in your project
- Select “New”
- Select package
- Type the package name “csen1002.tests.task7”



New Package: csen1002.tests.task7

Importing the Grammar File

- Download “Task 7: Grammar Template” form the CMS website.
- Extract the zip file
- Copy the extracted “Task7.g4” file using the context menu or “Ctrl”+ “C”
- Select the “grammars” directory from the project explorer
- Right click and select “Paste” or press “Ctrl” + “V”
- In the “Copy” dialog box that appears, press “OK”
- This is the file to be submitted via google form



Copy file C:\Users\lenovo\Desktop\Task7.g4

New name: Task7.g4

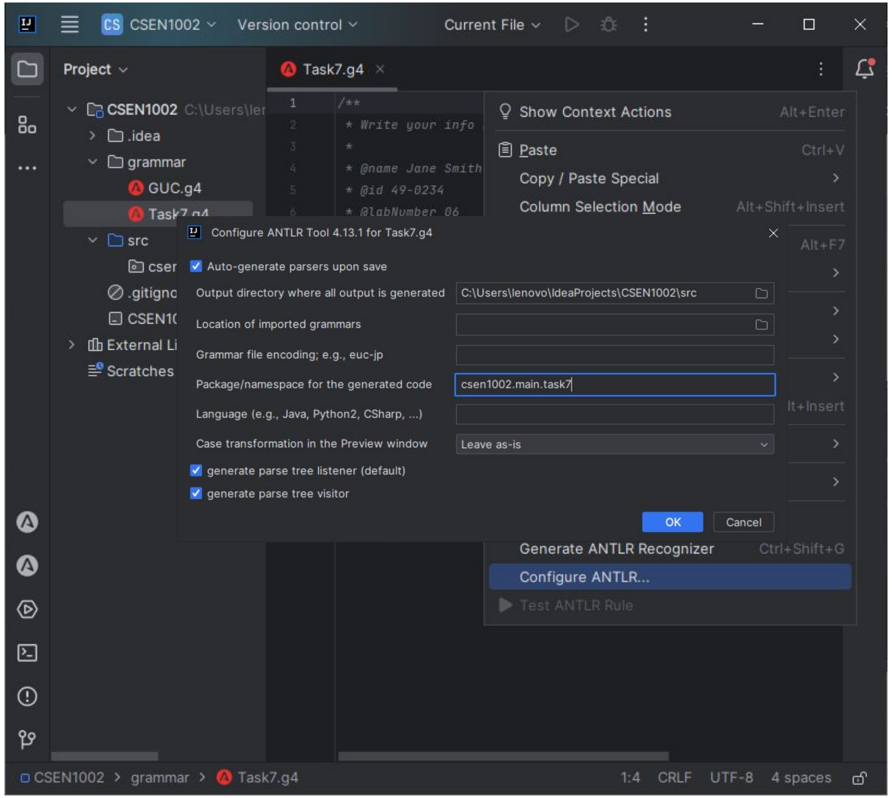
To directory: C:\Users\lenovo\IdeaProjects\CSEN1002\grammar

Use Ctrl+Space for path completion

☑ Open in editor

Configuring the Grammar File

- Right Click anywhere inside your file
- Select “Configure ANTLR”
- Select “src” subdirectory of your project as the “Output directory”
    - Do **not** choose the package subdirectory
- Type “csen1002.main.task7” as the “Package name for the generated code”
- Click “OK”



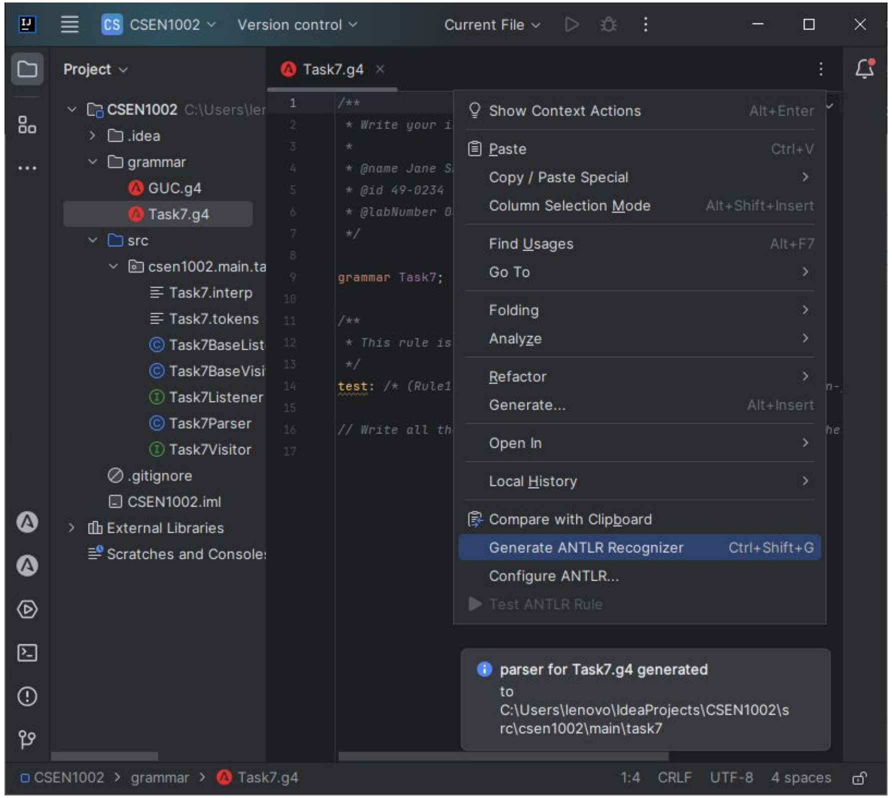
```
/**
 * Write your info
 *
 * @name Jane Smith
 * @id 49-0234
 * @labNumber 06
```

Configure ANTLR Tool 4.13.1 for Task7.g4

- [✓] Auto-generate parsers upon save
- Output directory where all output is generated: C:\Users\lenovo\IdeaProjects\CSEN1002\src
- Location of imported grammars:
- Grammar file encoding; e.g., euc-jp:
- Package/namespace for the generated code: csen1002.main.task7
- Language (e.g., Java, Python2, CSharp, ...):
- Case transformation in the Preview window: Leave as-is
- [✓] generate parse tree listener (default)
- [✓] generate parse tree visitor

Generating the ANTLR Recognizer

- Right Click anywhere inside your file
- Select “Generate ANTLR Recognizer”
- Alternatively, you can use the keyboard shortcut “Ctrl” + “Shift” + “G”
- A **pop up** should appear in the bottom right informing you of the successful generation of the recognizer
  - If you had unsaved changes, the pop up would not appear and you will need to generate the grammar one more time
- New files should appear inside “src/csen1002/main/task7”
- **This step MUST be repeated after any grammar modification for the changes to be reflected in the Java code**

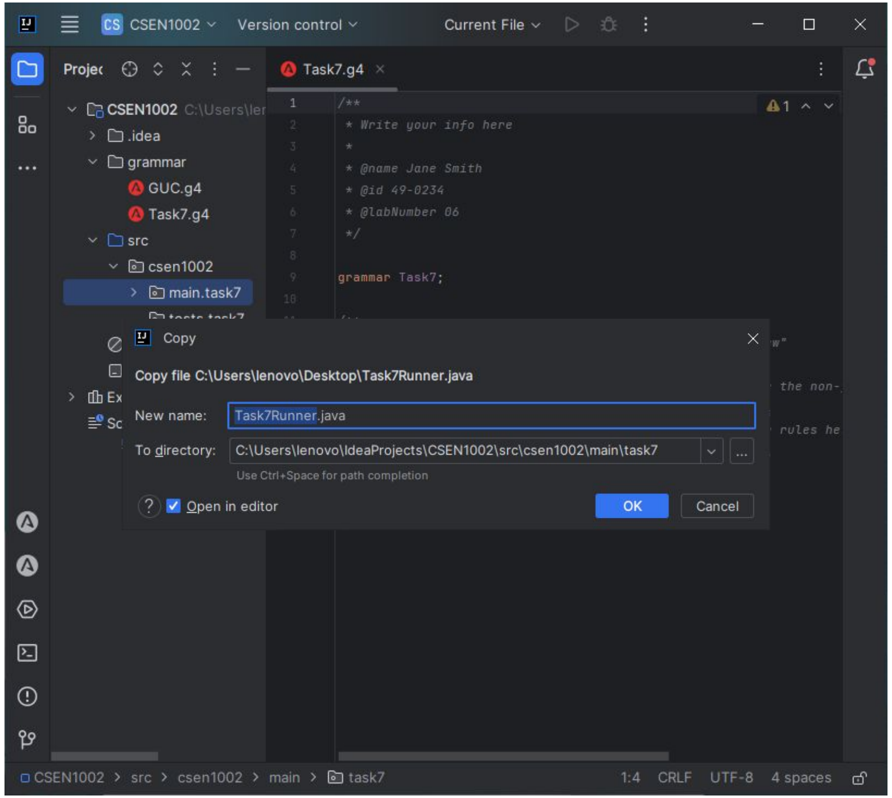


```
/**
 * Write your i
 *
 * @name Jane S
 * @id 49-0234
 * @labNumber 0
 */

grammar Task7;

/**
 * This rule is
 */
test: /* (Rule1

// Write all th
```

parser for Task7.g4 generated to C:\Users\lenovo\IdeaProjects\CSEN1002\src\csen1002\main\task7

Importing the Runner File

- Download “Task 7: Grammar Runner” form the CMS website.
- Extract the zip file
- Copy the extracted “Task7Runner.java” file using the context menu or “Ctrl”+ “C”
- Select the “src/csen1002/main/task7” directory from the project explorer
- Right click and select “Paste” or press “Ctrl” + “V”
- In the “Copy” dialog box that appears, press “OK”
- This file is for testing purposes and should **not** be submitted



```
/**
 * Write your info here
 *
 * @name Jane Smith
 * @id 49-0234
 * @labNumber 06
 */

grammar Task7;
```

Copy

Copy file C:\Users\lenovo\Desktop\Task7Runner.java

New name: Task7Runner.java

To directory: C:\Users\lenovo\IdeaProjects\CSEN1002\src\csen1002\main\task7

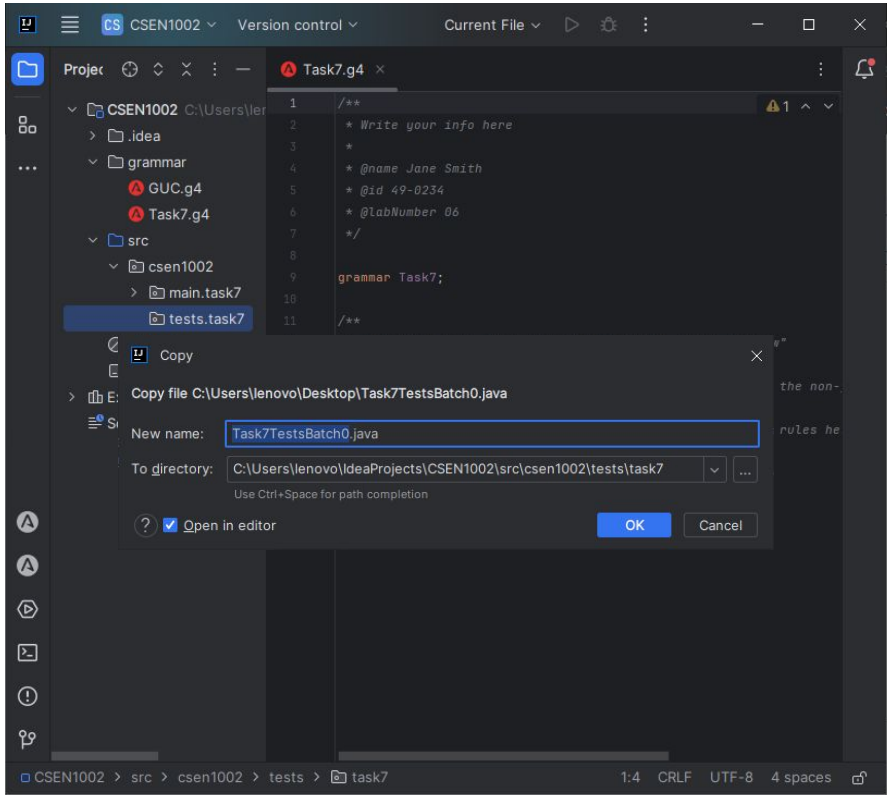
Use Ctrl+Space for path completion

[✓] Open in editor

OK Cancel

Importing Test Files

- Download “Task 7: Public Test Cases” form the CMS website.
- Extract the zip file
- Copy the extracted “Task7TestsBatch0.java” file using the context menu or “Ctrl”+ “C”
- Select the “src/csen1002/tests/task7” directory from the project explorer
- Right click and select “Paste” or press “Ctrl” + “V”
- In the “Copy” dialog box that appears, press “OK”



```
/**
 * Write your info here
 *
 * @name Jane Smith
 * @id 49-0234
 * @labNumber 06
 */

grammar Task7;

/**
```

Copy

Copy file C:\Users\lenovo\Desktop\Task7TestsBatch0.java

New name: Task7TestsBatch0.java

To directory: C:\Users\lenovo\IdeaProjects\CSEN1002\src\csen1002\tests\task7

Use Ctrl+Space for path completion

☑ Open in editor

OK Cancel

Troubleshooting

Common Issues

- The imports for ANTLR packages (org.antlr.v4…) are not working in the test/runner files.
  - Redo the ANTLR import steps
  - If you used the local method, do not move/delete the .jar file from its location
- The imports for JUnit are not working in the test files.
  - Redo the JUnit import steps
  - Make sure that you are using a version >=5.9.1
- The import for Task7Lexer is not working the test/runner files.
  - Check that the configuration for Task7.g4 is set correctly
  - Regenerate the ANTLR recognizer
- There is a “csen1002.main.task7” folder inside “src/main/task7”
  - The “Output directory” is incorrectly set, make sure it points to “src” and not a subdirectory of it

Common Issues

- Changes to Task7.g4 are not reflected in the test/runner files.
  - Delete the existing Task7*.java (e.g. Task7Lexer.java, Task7Listener.java) files in “csen1002.main.task7”
  - Check that the configuration for Task7.g4 is set correctly
  - Regenerate the ANTLR recognizer
  - Make sure that the pop up appears
  - Make sure that the grammar does not contain errors
- My grammar has errors and I cannot figure out what they are
  - Open the “Tool Output” panel from the bottom bar
  - Right click and select “Clear All” to clear outdated errors
  - Make a change to the grammar and save
  - The panel should show the exact errors in the grammar
- There is a “Non-fragment lexer rule matches the empty string” warning/ IntelliJ crashes when testing rules
  - Rewrite this rule so there is no variation of it that can match the empty string.
  - Tokens that can match the empty string are dangerous, as they can be infinitely generated from a string.

Common Issues

- The option to configure/generate my grammar is greyed out
    - Make sure that the file extension is “.g4” **not** “.G4”
- I dragged and dropped a file into IntelliJ, now it’s gone
    - Unlike eclipse, dragging and dropping files into IntelliJ moves them instead of copying them
    - Copy then paste files into IntelliJ instead of dragging and dropping them
- I moved/copied a file into IntelliJ, now it has no/incorrect package declaration
    - Unlike eclipse, IntelliJ automatically refactors imported files, and changes the package declaration based on the destination directory
    - Manually fix the package declaration or make sure that the correct package already exists before copying the file directly into it