**Dear visitor,**

Thank you for your interest in our benchmark. *PDBGTGT* is the result of a novel research conducted at the [SQLab](http://sqlab.um.ac.ir/index.php?lang=en) towards benchmarking in the field of design pattern detection. To the best of our knowledge, this is the first benchmark which uses automatic generation of testbeds for the evaluation process. *PDBGTGT* has a precise, and repeatable evaluation process, and can be used to compare pattern detection methods in an objective and fair fashion. In this benchmark, a detection method is evaluated based on its ability to mine different types of patterns (and their variants) from testbeds with different levels of complexity. *PDBGTGT* meets the desiderata for benchmarks. It is fair, expandable, accessible, cost-effective, solvable, portable, and scalable.

To use our benchmark please send us a zip file containing the following files:

1. The coded algorithm of your detection method in Java (a jar file)
2. Your desired patterns for inserting into the testbeds.

* Detection methods to be evaluated in , should be able to receive inputs in the form of a Java code or a class diagram.
* The coded input algorithms should be able to receive the following arguments in the main method:
* Testbed code path (arg[0]): a path to the folder where all testbed's classes are located
* Testbed class diagram path (arg[1]): a path to testbed's class diagrams
* Save location path (arg[2]): users must write their detection results in this path
* Xamp server URL (arg[3])
* Xamp server port (arg[4])
* Database user name (arg[5])
* Database password (arg[6])
* To facilitate the comparison and ranking of methods, we have a predefined list of patterns. These patterns will be used in the generated testbeds.
* If you do not select a pattern in the following list, this means that the input method is unable to mine that pattern.
* List of the patterns: 1- Abstract Factory, 2- Adapter, 3- Composite, 4- Decorator, 5- Factory Method, 6- Observer, 7- Singleton, 8- State, 9- Strategy, 10- Template Method, 11- Visitor.

Note 1: You might want to upload some library jar files along with the detection code. Therefore, please specify for us the main source among the uploaded files.

Note 2: To define new patterns in , a specific grammar is developed which is derived from the specification of the Java and C languages and covers the object-oriented features (Click here to download the grammar file). This grammar has an XML structure and allows users to define the requirements of their patterns using a set of tags. In Table 1, these tags are listed (Click here to download the Singleton design pattern grammar as an example).

Note 3: Please test your pattern definition files by the [ANTLRWorks](http://tunnelvisionlabs.com/products/demo/antlrworks) application before sending them to us.

Table 1: Tags used for defining new patterns

|  |  |  |
| --- | --- | --- |
| Definition | Fields | Tag |
| Define the structure of a class | modifiers, name, role, opt, isMandatory\* | <class>…</class> |
| Define the structure of an interface | modifiers, name, role, opt, isMandatory | <interface>…</interface> |
| Define a variable | modifiers, type, name, isInit, value, multiplicity | <attribute>…</attribute> |
| Define a function | modifiers, name, returnType | <method>…</method> |
| Define a class constructor | modifiers | <constructor>…</ constructor > |
| Define a parameter | name, type, multiplicity | <parameter>…</parameter> |
| Define a loop | counter\_name, start\_from, until, step\_size | <loop>…</loop> |
| Define a conditional statement | condition | <if>…</if> |
| Define an assignment | left\_side, operator, right\_side | <initialize>…<initialize> |
| Call the parent constructor |  | <super>…</super> |
| Return a value from a function | return | <return>…</return> |
| Call a function | which\_object, which\_method, argument\_list | <call>…</call> |
| Inherit from a concrete class | superClass\_id | <inheritance>…</inheritance> |
| Inherit from an interface class | superClass\_id | <realization>…</realization> |
| Dependency between two classes | supplier\_id | <dependency>…</dependency> |
| Association between two classes | destination\_id, multiplicity | <association>…</association> |
| \*If the value of this field is True, it means that this role is a main role in the pattern. | | |

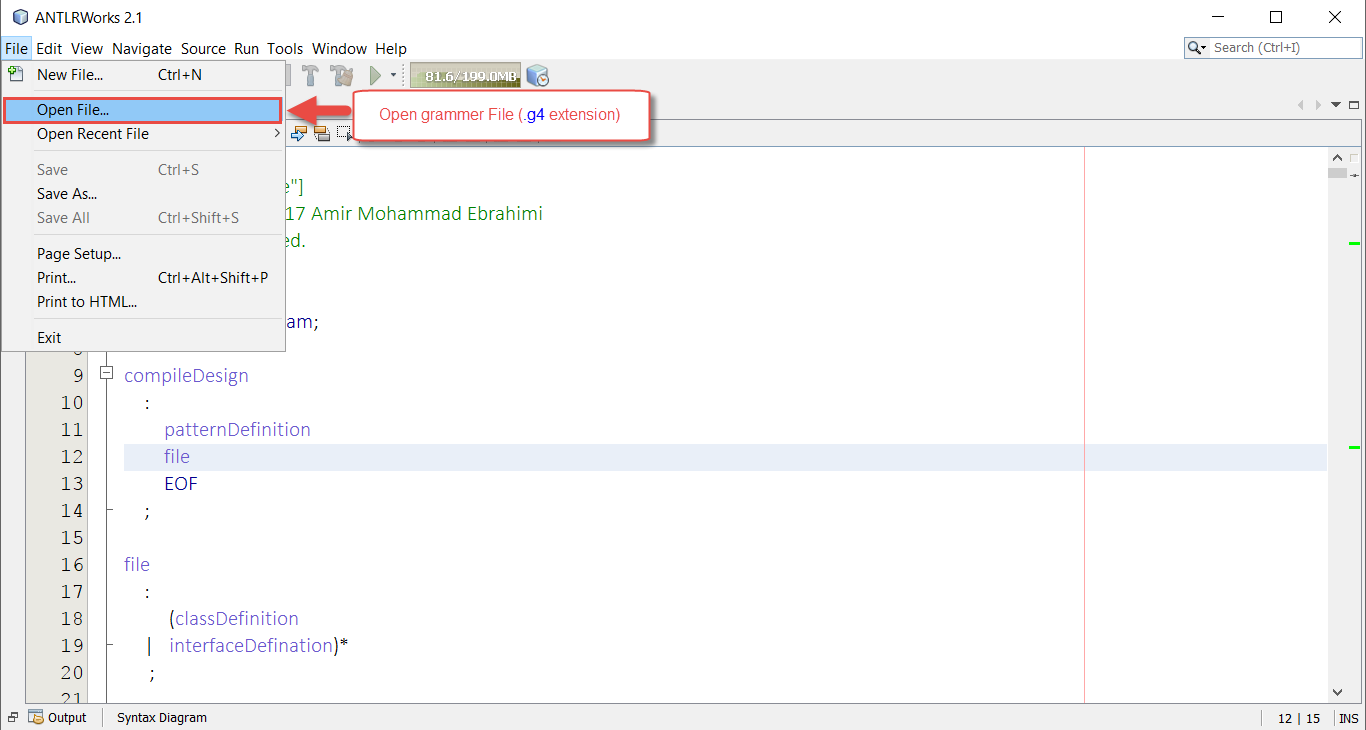
Note 4: can build precise class diagrams from the generated source codes (Click here to download the class diagram generator library and the guideline to use it).

Note 3: includes a software component (a library) which allows users to convert their algorithm’s output to the same format between all the registered methods in the (Click here to download the output generator library and the guideline to use it).

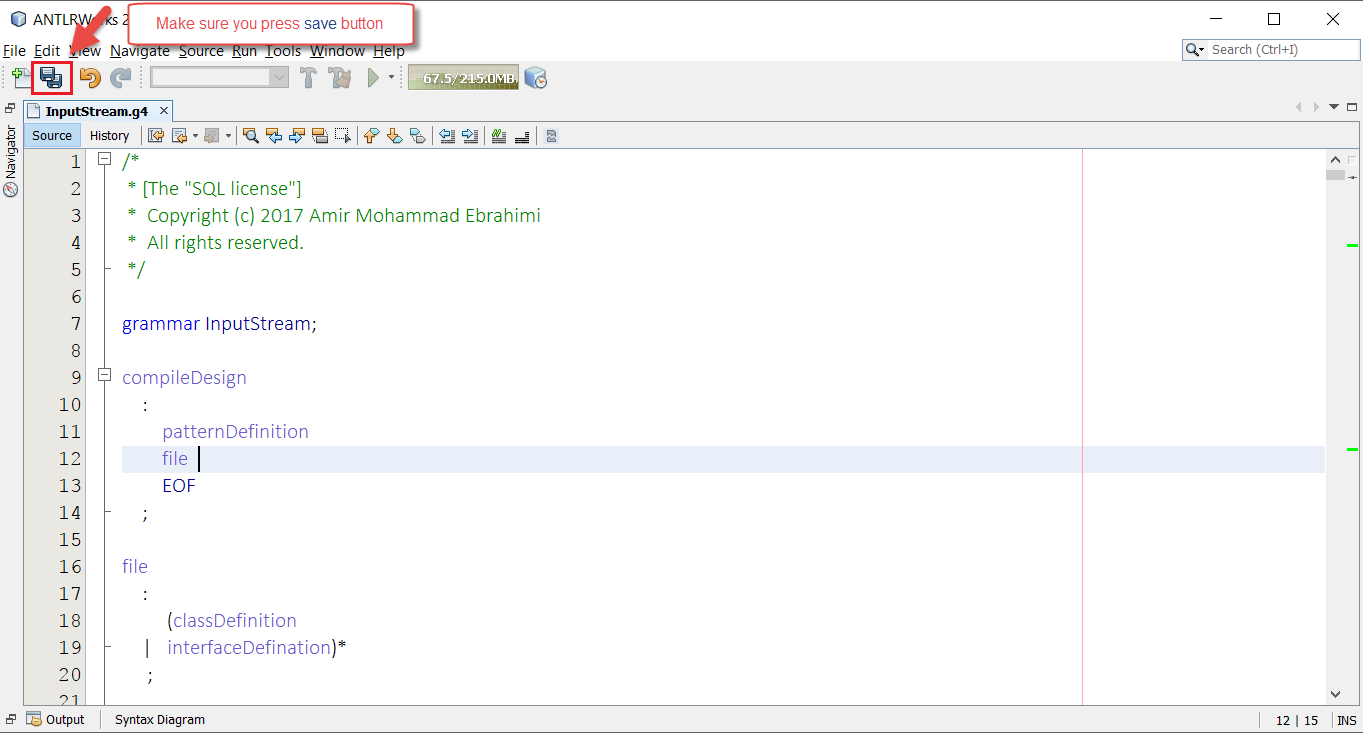
**How to Work with Antlr:**

In the following figures, we provide the step by step instructions on how to test the pattern definitions with ANTLRWorks.

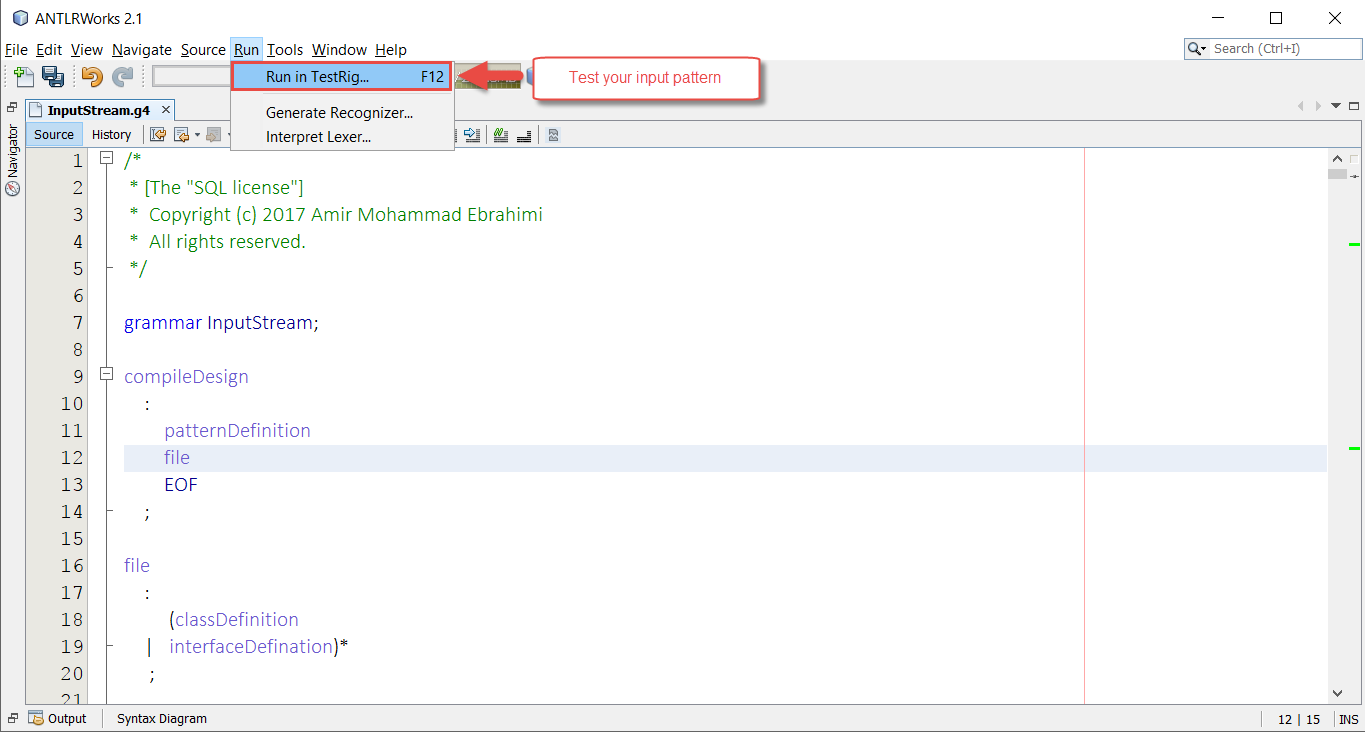
Step 1 - Open grammar file (.g4 extension):

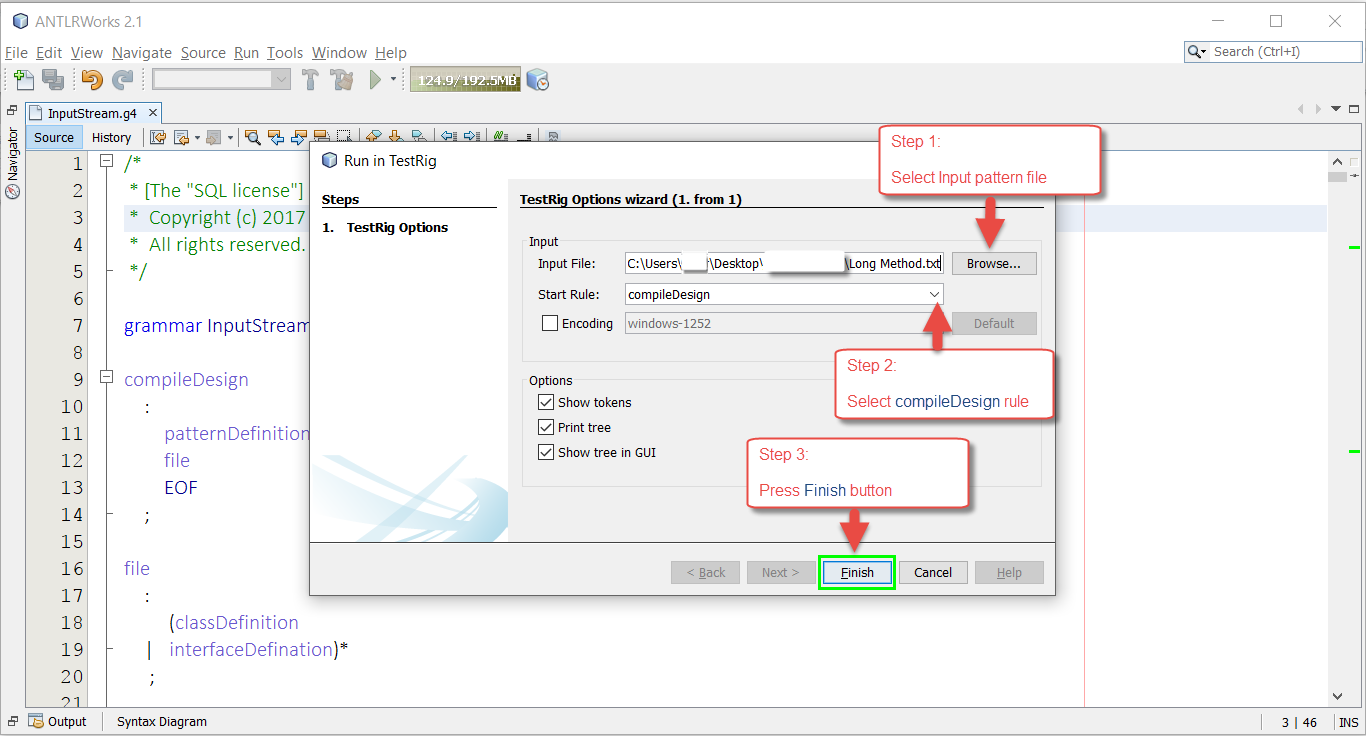


Step 2 – Make sure you press Save button:

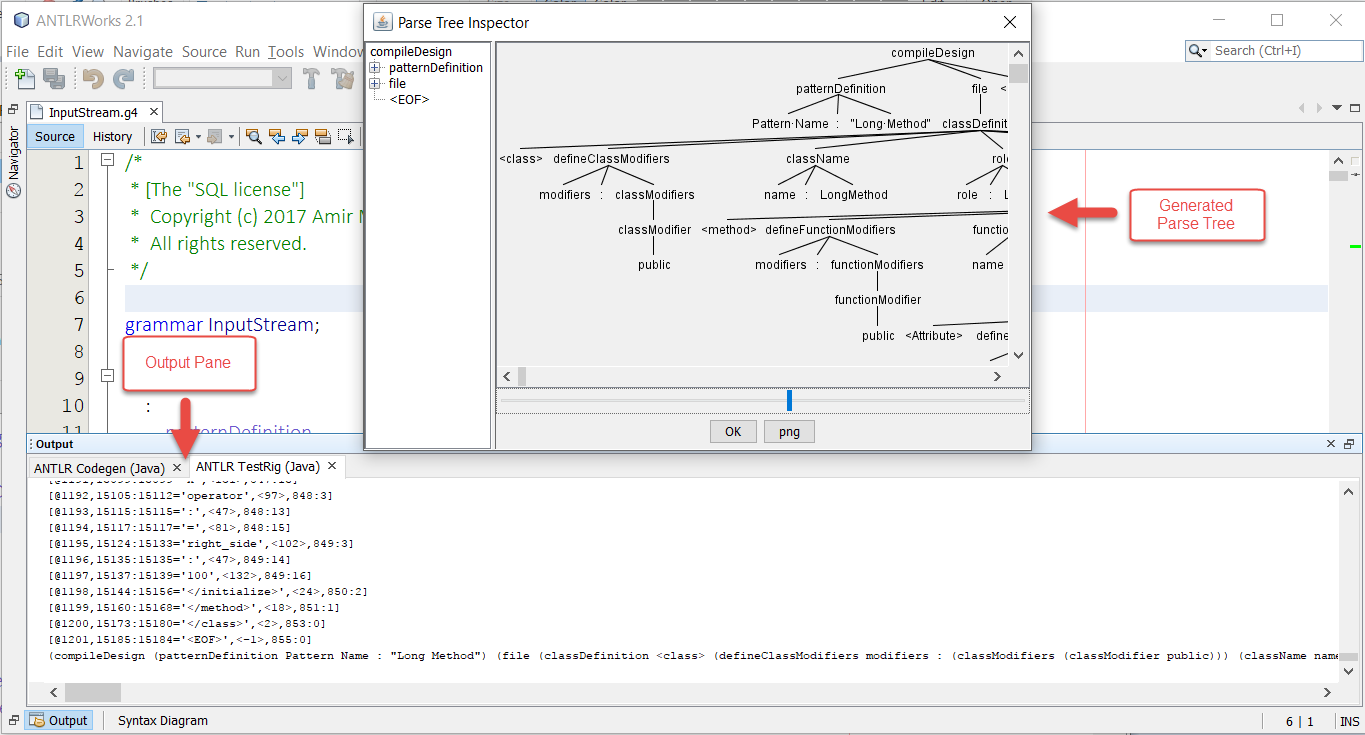


Step 3 – Test your input pattern:

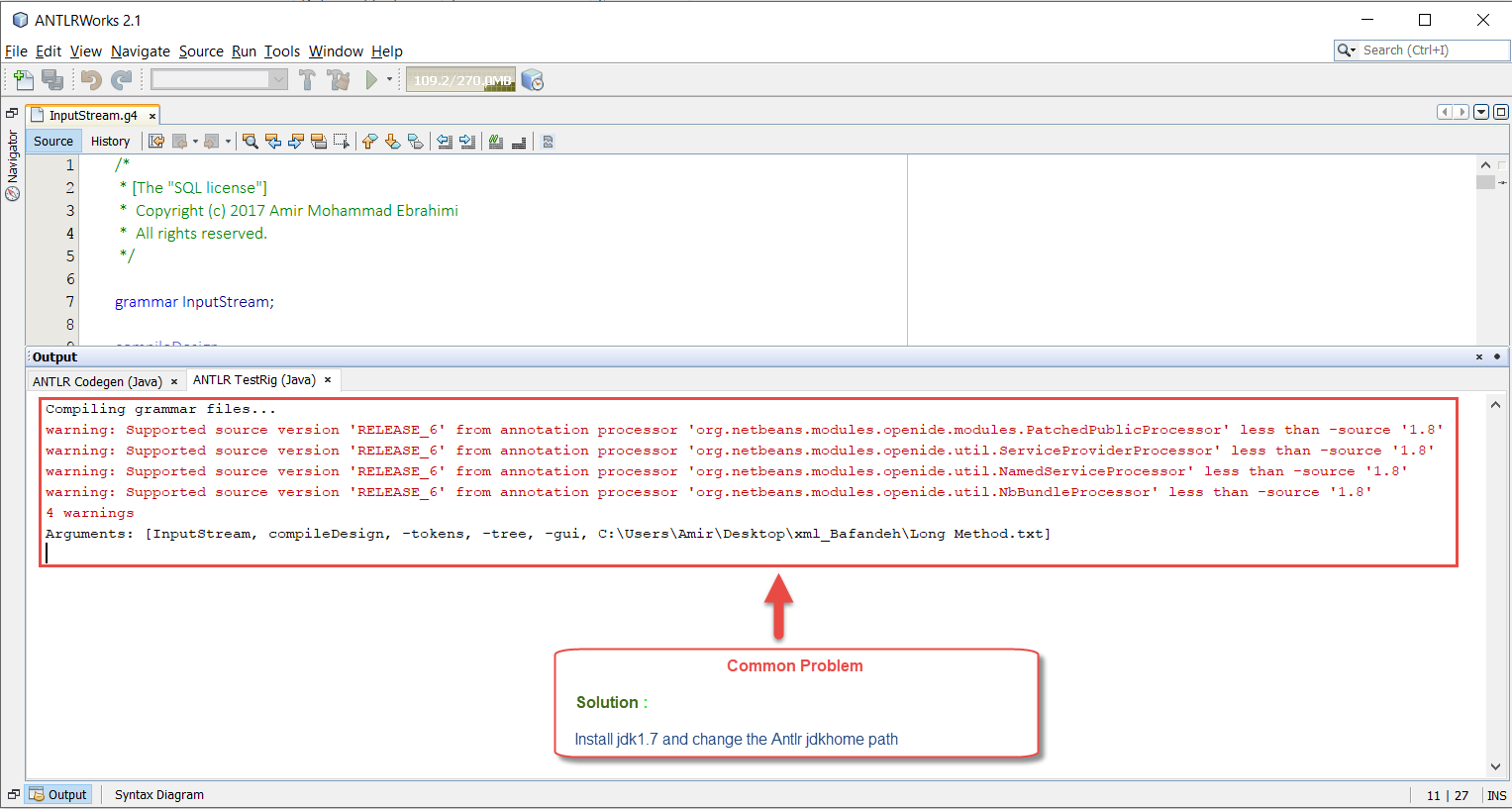


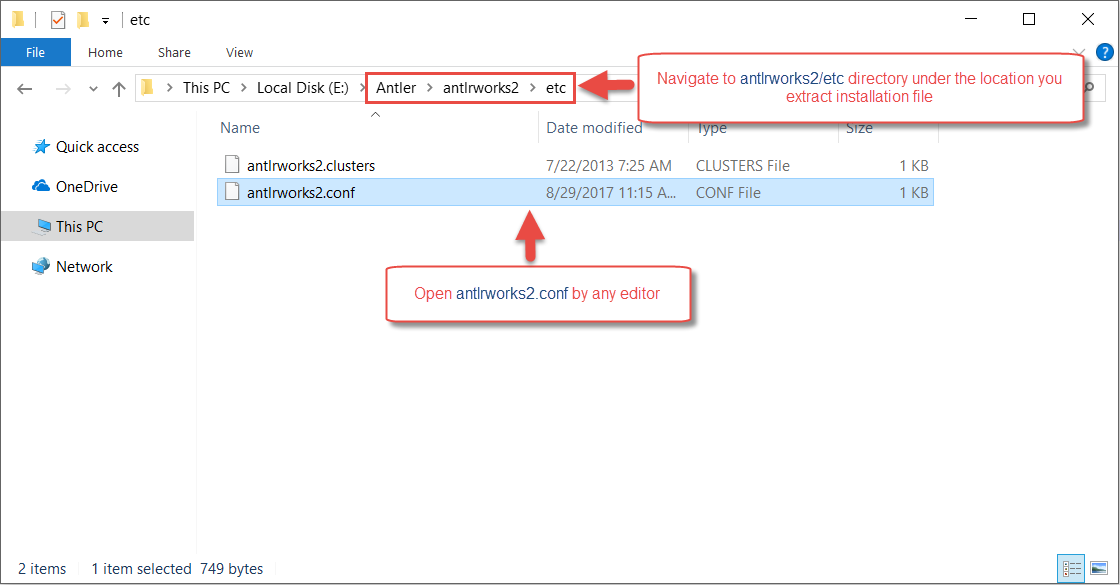


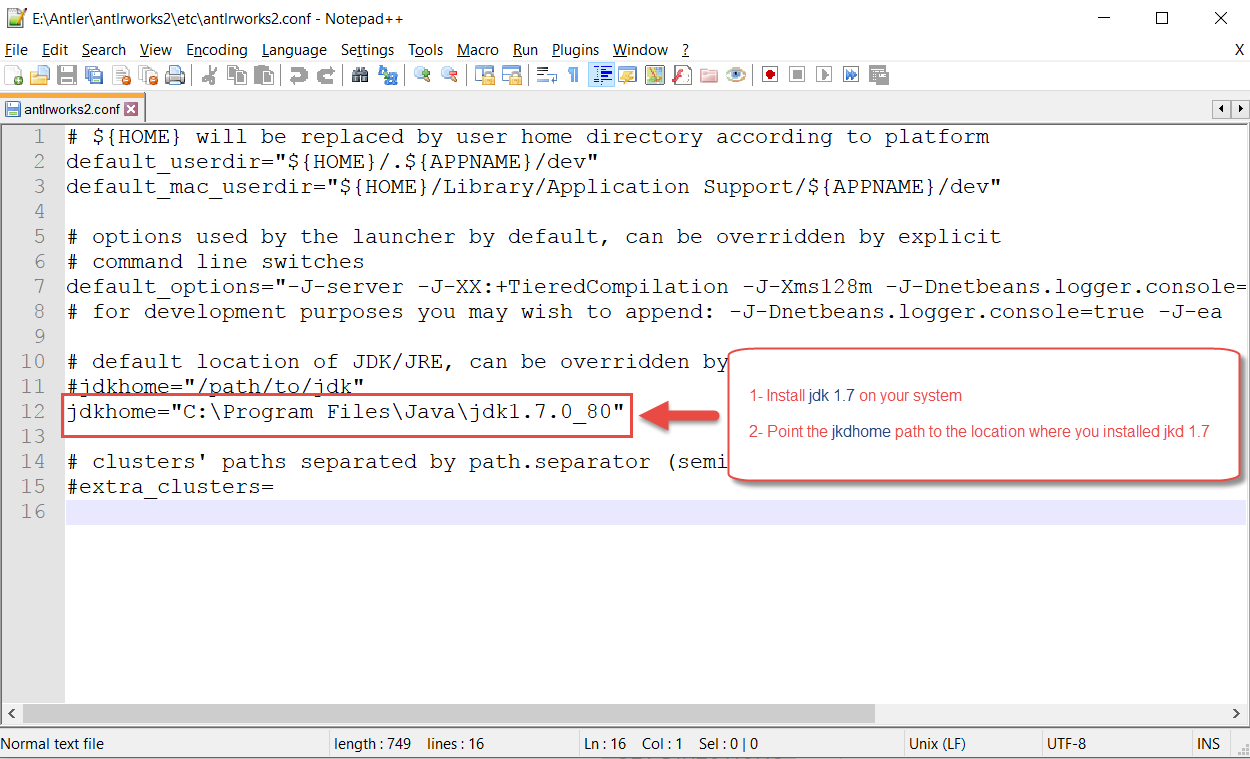
A view of the generated parse tree:

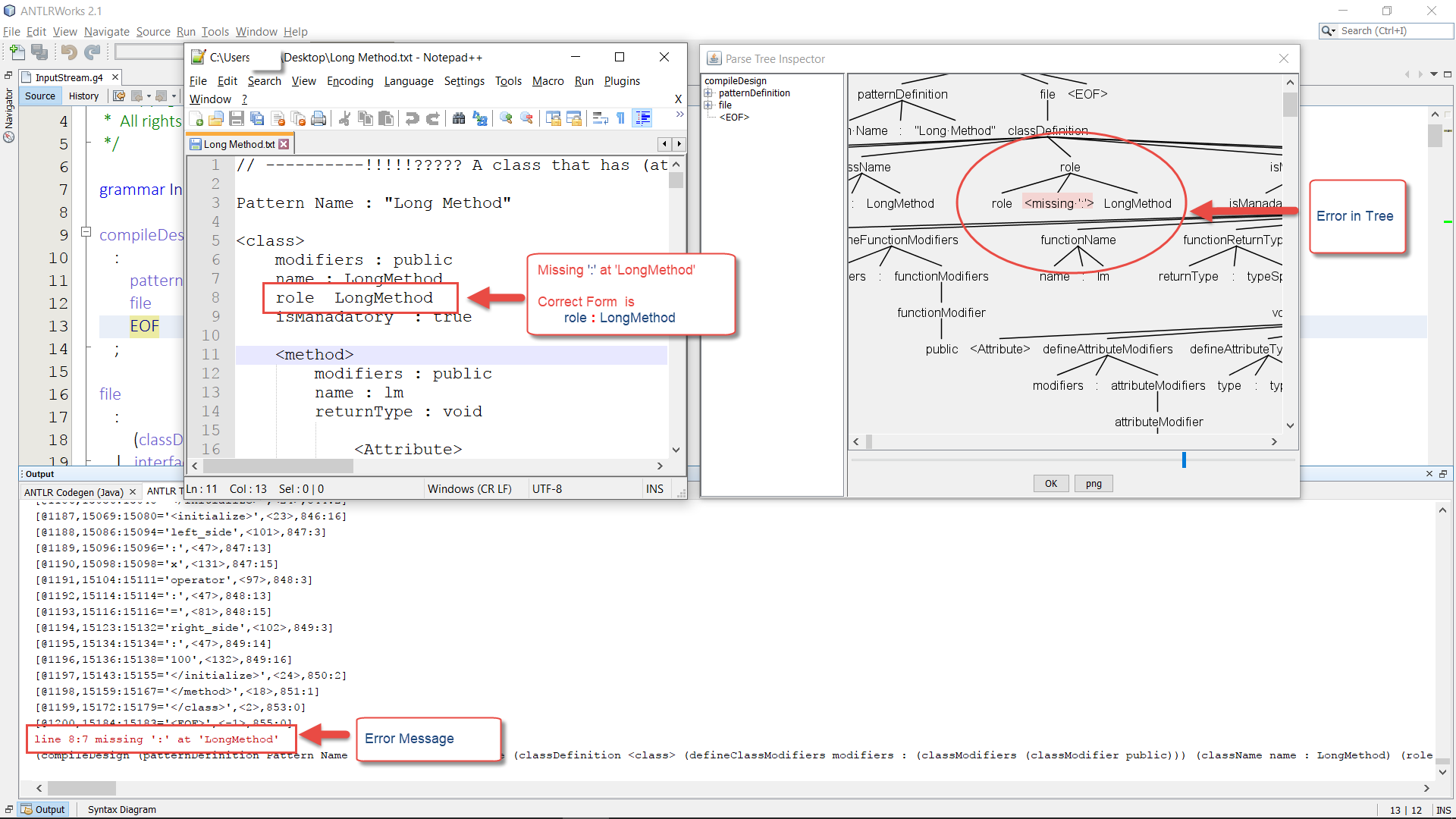


Handling common errors in the test process (1):







Handling common errors in the test process (2):

**References:**

- A New Benchmark for Evaluating Pattern Mining Methods Based on the Automatic Generation of Testbeds (journal submission)

**Contact Us:**

Have a question, suggestion, need help, or want to report an error or bug? Send us an email at [sqlab@um.ac.ir](mailto:sqlab@um.ac.ir?Subject=PDB_GTGT) and be sure to reference PDB\_GTGT in the subject line.