Week 2: 11th September 2025

INNOPOLIS UNIVERSITY

[F25] Compiler Construction

Team 806

- Timofey Ivlev
- George Selivanov

We glad to present you our project:

Javdin

Java **d**ynamic **in**terpreter with a Bison-based parser, for Dynamic academic language

Project's technology stack

- Source Language **D**
- Implementation language **Java**
- Parser development tool **Bison-based** **CUP (Construction of Useful Parsers)**
- Target platform **JVM**
- Other tools and versions:
  - Java 17
  - Maven 3.6
  - JUnit 5 for testing
  - AssertJ for assertions
  - JaCoCo for code coverage
  - CUP and JFlex for parser generation

Javdin Lexer Implementation

Overview

The lexer has been completely implemented to comply with all Project D language specification requirements:

Keywords (All Implemented)

- **Control Flow**: `if`, `then`, `else`, `end`, `while`, `for`, `in`, `loop`, `exit`

- **Functions**: `function`, `func`, `return`, `is`
- **Variables**: `var`
- **I/O**: `print`, `input`
- **Literals**: `true`, `false`, `none`
- **Logical**: `and`, `or`, `xor`, `not`
- **Legacy**: `lambda`, `break`, `continue` (for compatibility)

Type Indicators (All Implemented)

- **Primitive Types**: `int`, `real`, `bool`, `string`
- **Composite Types**: `[]` (array type), `{}` (tuple type)
- **Function Type**: `func`
- **None Type**: `none`

Operators (All Implemented)

- **Arithmetic**: `+`, `-`, `*`, `/`, `%`
- **Assignment**: `=` (comparison), `:=` (assignment)
- **Comparison**: `<`, `<=`, `>`, `>=`, `==`, `!=`, `/=`
- **Logical**: `and`, `or`, `xor`, `not`
- **Special**: `->` (arrow), `=>` (short if), `..` (range), `is` (type check)

Literals (All Implemented)

- **Integer**: `123`, `0`, `999999`
- **Real**: `3.14`, `0.5`, `123.456`
- **Boolean**: `true`, `false`
- **String**: Both single and double quotes (`'text'`, `"text"`)
- **None**: `none`

Delimiters (All Implemented)

- **Parentheses**: `(`, `)`
- **Braces**: `{`, `}`
- **Brackets**: `[`, `]`
- **Separators**: `;`, `,`, `.`, `:`

Other

- **Comment Handling**: Single-line (`//`) and multi-line (`/* */`) comments
- **Escape Sequence Support**: Full string escape sequences (`\n`, `\t`, `\r`)

Examples

First step - choosing project directory, in my case:

```
cd /home/timofey/Desktop/javdin
```

Example 1: Basic Variable Declarations

Command:

```
mvn exec:java -Dexec.mainClass="com.javdin.lexer.LexerDebug" -
Dexec.args="test-resources/project-d-variables.d"
```

Output:

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< com.javdin:javdin >---------------------
----
[INFO] Building Javdin 1.0.0
[INFO] --------------------------------[ jar ]----------------------------
----
[INFO]
[INFO] --- exec-maven-plugin:3.5.1:java (default-cli) @ javdin ---
============================================================
JAVDIN LEXER DEMONSTRATION
============================================================
File: test-resources/project-d-variables.d
Source length: 180 characters

SOURCE CODE:
----------------------------------------
  1: // Example 1: Variable declarations with Project D syntax
  2: var x := 42
  3: var y := 3.14159
  4: var name := 'Hello World!'
  5: var flag := true
  6: var empty := none
  7:
  8: print x, y, name, flag, empty
  9:

TOKENS:
----------------------------------------
#    TOKEN_TYPE           POSITION        VALUE
DESCRIPTION
---------------------------------------------------------------------------
----------
1    NEWLINE              1:58            <no value>
Line break
2    VAR                  2:1             "var"
Variable declaration
3    IDENTIFIER           2:5             "x"
Identifier/variable name
4    ASSIGN_OP            2:7             <no value>
Assignment (:=)
```

```
5    INTEGER            2:10            "42"
Integer literal
6    NEWLINE            2:12            <no value>
Line break
7    VAR                3:1             "var"
Variable declaration
8    IDENTIFIER         3:5             "y"
Identifier/variable name
9    ASSIGN_OP          3:7             <no value>
Assignment (:=)
10   REAL               3:10            "3.14159"
Real number literal
11   NEWLINE            3:17            <no value>
Line break
12   VAR                4:1             "var"
Variable declaration
13   IDENTIFIER         4:5             "name"
Identifier/variable name
14   ASSIGN_OP          4:10            <no value>
Assignment (:=)
15   STRING             4:13            "Hello World!"
String literal
16   NEWLINE            4:27            <no value>
Line break
17   VAR                5:1             "var"
Variable declaration
18   IDENTIFIER         5:5             "flag"
Identifier/variable name
19   ASSIGN_OP          5:10            <no value>
Assignment (:=)
20   TRUE               5:13            "true"
Boolean literal
21   NEWLINE            5:17            <no value>
Line break
22   VAR                6:1             "var"
Variable declaration
23   IDENTIFIER         6:5             "empty"
Identifier/variable name
24   ASSIGN_OP          6:11            <no value>
Assignment (:=)
25   NONE               6:14            "none"
None literal
26   NEWLINE            6:18            <no value>
Line break
27   NEWLINE            7:1             <no value>
Line break
28   PRINT              8:1             "print"
Print statement
29   IDENTIFIER         8:7             "x"
Identifier/variable name
30   COMMA              8:8             <no value>
Comma separator
31   IDENTIFIER         8:10            "y"
Identifier/variable name
```

```
32   COMMA                  8:11           <no value>
Comma separator
33   IDENTIFIER             8:13           "name"
Identifier/variable name
34   COMMA                  8:17           <no value>
Comma separator
35   IDENTIFIER             8:19           "flag"
Identifier/variable name
36   COMMA                  8:23           <no value>
Comma separator
37   IDENTIFIER             8:25           "empty"
Identifier/variable name
38   NEWLINE                8:30           <no value>
Line break
39   EOF                    9:1            <no value>
End of file
----------------------------------------------------------------------------
----------
Total tokens: 39
[INFO] ---------------------------------------------------------------------
----
[INFO] BUILD SUCCESS
[INFO] ---------------------------------------------------------------------
----
[INFO] Total time:  0.323 s
[INFO] Finished at: 2025-09-11T15:38:18+03:00
[INFO] ---------------------------------------------------------------------
----
```

> [!NOTE] On next examples we decided to shorten the examples output to few tokens. Because report size became too big. If want to see the full output, you can watch out demo or run these examples on your own machine.

Example 2: Control Flow Structures

Command:

```
mvn exec:java -Dexec.mainClass="com.javdin.lexer.LexerDebug" -
Dexec.args="test-resources/project-d-control-flow.d"
```

Output:

```
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------------< com.javdin:javdin >---------------------
----
[INFO] Building Javdin 1.0.0
```

```
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec-maven-plugin:3.5.1:java (default-cli) @ javdin ---
============================================================
JAVDIN LEXER DEMONSTRATION
============================================================
File: test-resources/project-d-control-flow.d
Source length: 301 characters

SOURCE CODE:
----------------------------------------
  1: // Example 2: Control flow with Project D syntax
  2: var i := 0
  3: loop
  4:     print 'Hello'
  5:     i := i + 1
  6:     if i = 100 => exit
  7: end
  8:
  9: for j in 1..3 loop
 10:     print 'Hello from loop', j
 11: end
 12:
 13: var array := [1, 2, 3, 4, 5]
 14: var sum := 0
 15: for element in array loop
 16:     sum := sum + element
 17: end
 18: print 'Sum is:', sum
 19:

TOKENS:
----------------------------------------
#    TOKEN_TYPE           POSITION        VALUE
DESCRIPTION
--------------------------------------------------------------------------------
1    NEWLINE              1:49            <no value>
Line break
2    VAR                  2:1             "var"
Variable declaration

...

82   NEWLINE              18:21           <no value>
Line break
83   EOF                  19:1            <no value>
End of file
--------------------------------------------------------------------------------
Total tokens: 83
[INFO] ------------------------------------------------------------------------
```

```
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.317 s
[INFO] Finished at: 2025-09-11T15:15:26+03:00
[INFO] ------------------------------------------------------------------------
```

Example 3: Advanced Function Syntax

Command:

```
mvn exec:java -Dexec.mainClass="com.javdin.lexer.LexerDebug" -
Dexec.args="test-resources/project-d-functions.d"
```

Output:

```
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------------< com.javdin:javdin >--------------------------
[INFO] Building Javdin 1.0.0
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec-maven-plugin:3.5.1:java (default-cli) @ javdin ---
============================================================
JAVDIN LEXER DEMONSTRATION
============================================================
File: test-resources/project-d-functions.d
Source length: 555 characters

SOURCE CODE:
----------------------------------------
  1: // Example 3: Functions and complex expressions
  2: var add := func(a, b) is
  3:     return a + b
  4: end
  5:
  6: var square := func(x) => x * x
  7:
  8: var factorial := func(n) is
  9:     if n <= 1 then
 10:         return 1
 11:     else
 12:         return n * factorial(n - 1)
 13:     end
 14: end
```

```
  15:
  16: // Array and tuple literals
  17: var numbers := [1, 2, 3, 4, 5]
  18: var person := {name := 'Alice', age := 30, 42.5}
  19:
  20: // Type checking
  21: if factorial is func then
  22:     print 'factorial is a function'
  23: end
  24:
  25: // Comparisons with both syntaxes
  26: if 10 /= 20 and 5 != 3 then
  27:     print 'Different inequality operators'
  28: end
  29:

TOKENS:
----------------------------------------
#    TOKEN_TYPE           POSITION        VALUE
DESCRIPTION
---------------------------------------------------------------------------
----------
1    NEWLINE              1:48            <no value>
Line break
2    VAR                  2:1             "var"
Variable declaration

...

129  NEWLINE              28:4            <no value>
Line break
130  EOF                  29:1            <no value>
End of file
---------------------------------------------------------------------------
----------
Total tokens: 130
[INFO] --------------------------------------------------------------------
----
[INFO] BUILD SUCCESS
[INFO] --------------------------------------------------------------------
----
[INFO] Total time:  0.395 s
[INFO] Finished at: 2025-09-11T15:23:30+03:00
[INFO] --------------------------------------------------------------------
----
```

Example 4: All operators and type indicators

Command:

```
mvn exec:java -Dexec.mainClass="com.javdin.lexer.LexerDebug" -
Dexec.args="test-resources/project-d-operators.d"
```

Here the output is shorten to first 16 tokens, which shows operators :=, +, *, -, /:

```
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------------< com.javdin:javdin >---------------------
----
[INFO] Building Javdin 1.0.0
[INFO] --------------------------------[ jar ]----------------------------
----
[INFO]
[INFO] --- exec-maven-plugin:3.5.1:java (default-cli) @ javdin ---
============================================================
JAVDIN LEXER DEMONSTRATION
============================================================
File: test-resources/project-d-operators.d
Source length: 693 characters

SOURCE CODE:
----------------------------------------
  1: // Example 4: All operators and type indicators
  2: // Arithmetic
  3: var result := (10 + 5) * 3 - 2 / 1
  4:
  5: // Comparison operators
  6: var tests := [
  7:     5 < 10,
  8:     10 > 5,
  9:     10 >= 10,
 10:     5 <= 5,
 11:     10 = 10,
 12:     10 /= 5,
 13:     10 != 5
 14: ]
 15:
 16: // Logical operators
 17: var logic := true and false or not true xor false
 18:
 19: // Type indicators
 20: var types := {
 21:     int_type := int,
 22:     real_type := real,
 23:     bool_type := bool,
 24:     string_type := string,
 25:     array_type := [],
 26:     tuple_type := {}
 27: }
 28:
 29: // Array access and assignment
 30: var arr := [10, 20, 30]
 31: arr[100] := func(x) => x + 1
 32: arr[1000] := {a := 1, b := 2.7}
 33:
 34: // Tuple access
 35: var t := {first := 'hello', second := 'world', 3.14}
 36: var x := t.first
 37: var y := t.2
 38:

TOKENS:
----------------------------------------
#    TOKEN_TYPE            POSITION        VALUE
DESCRIPTION
----------------------------------------------------------------------------
----------
1    NEWLINE               1:48            <no value>
Line break
2    NEWLINE               2:14            <no value>
Line break
3    VAR                   3:1             "var"
Variable declaration
4    IDENTIFIER            3:5             "result"
Identifier/variable name
5    ASSIGN_OP             3:12            <no value>
Assignment (:=)
6    LEFT_PAREN            3:15            <no value>
Parenthesis
7    INTEGER               3:16            "10"
Integer literal
8    PLUS                  3:19            <no value>
Arithmetic operator
9    INTEGER               3:21            "5"
Integer literal
10   RIGHT_PAREN           3:22            <no value>
Parenthesis
11   MULTIPLY              3:24            <no value>
Arithmetic operator
12   INTEGER               3:26            "3"
Integer literal
13   MINUS                 3:28            <no value>
Arithmetic operator
14   INTEGER               3:30            "2"
Integer literal
15   DIVIDE                3:32            <no value>
Arithmetic operator
16   INTEGER               3:34            "1"
Integer literal

...

----------------------------------------------------------------------------
----------
Total tokens: 186
```

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.336 s
[INFO] Finished at: 2025-09-11T15:28:45+03:00
[INFO] ------------------------------------------------------------------------
```

Testing and Quality Assurance

You can run the tests using:

```
mvn test
```

Test Coverage

- **Original Tests**: 22 tests from initial implementation
- **Enhanced Tests**: 21 additional tests for Project D compliance
- **Integration Tests**: 4 end-to-end tests
- **Parser Tests**: 7 tests for parser integration
- **Total**: **54 tests** all passing

Test Categories

1. **Basic Tokenization**: Keywords, operators, literals
2. **Project D Compliance**: All new syntax features
3. **Error Handling**: Invalid input and edge cases
4. **Complex Programs**: Real-world code examples
5. **Edge Cases**: Empty strings, nested comments, escape sequences

Package Structure

```
com.javdin.lexer/
├── Lexer.java              # Main lexer implementation
├── Token.java              # Immutable token record
├── TokenType.java          # Complete token type enumeration
├── LexicalException.java   # Error handling
└── LexerDebug.java         # Demonstration utility
```