

Introduction to Compiler Design

Lesson 6:

Scanners, JLex Scanner Generator

JLex Scanner Generator

Declarative specification

tell what you want to scan, it will generate the rest

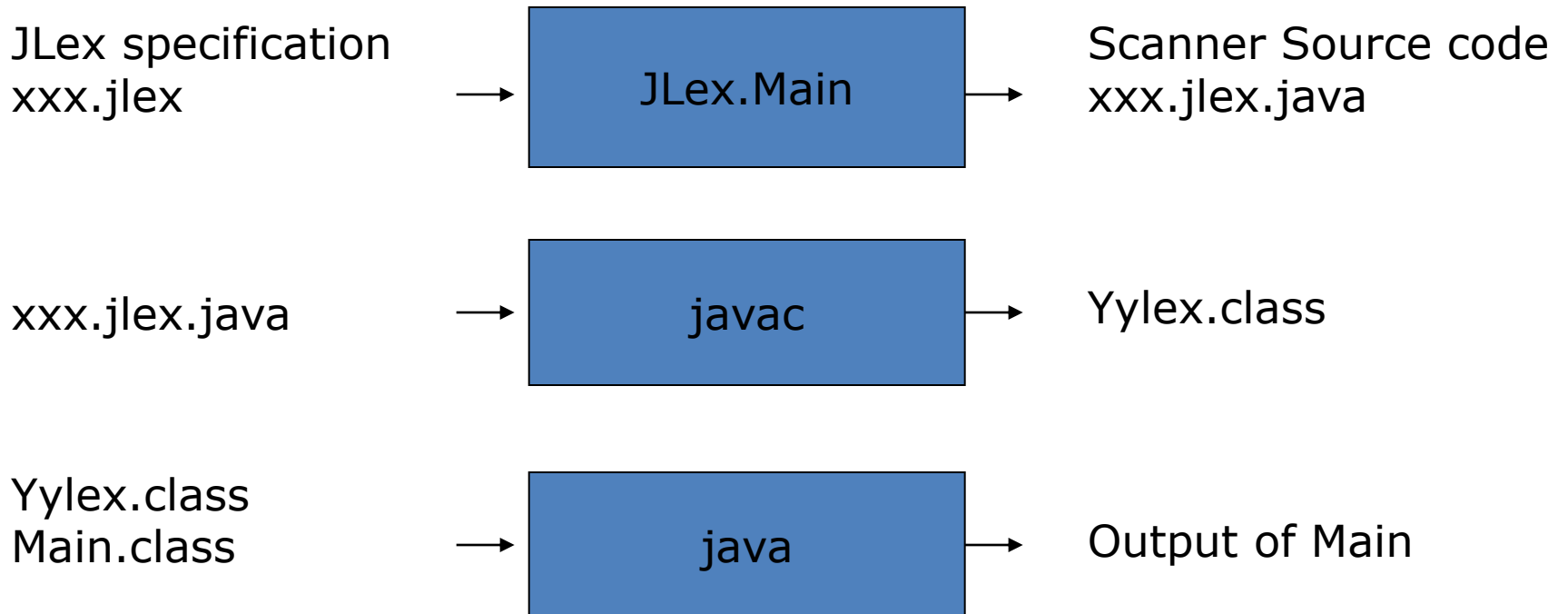
Input: set of regexps + associated actions

Inside a xyz.jlex file

Output: Java source code for a scanner

xyz.jlex.java source code of scanner

JLex Scanner Generator



Yylex Class

```
class Yylex {  
  
    ...  
    public Yylex(java.io.Reader i) ...  
    public Yylex(java.io.InputStream i) ...  
  
    ...  
    public <Type> next_token() ...  
  
    ...  
}
```

Regular Expression Rules

Regular expression



Pattern to be matched

{ action }



Java code to be executed
When pattern is matched

When the scanner's `next_token()` method is called, it repeats:

- 1/ Find the longest sequence of characters in the input (starting with current character) that matches a pattern
- 2/ Perform the associated action.

next_token method

When the scanner's `next_token` method is called, it repeats:

1. Find the longest sequence of characters in the input (starting with current character) that matches a pattern
2. Perform the associated action.

until an action causes the `next_token` method to return.

If there are several patterns that match the same (longest) sequence of characters, then the first such pattern is considered to be matched (so the order of the patterns can be important).

If an input character is not matched in any pattern, the scanner throws an exception (so it is important to make sure that there can be no such unmatched characters, since it is not good to have a scanner that can "crash" on bad input).

Regular Expressions

- Most Characters match themselves

abc

==

while

- Characters in “” match themselves
(even special characters)

“abc”

“==”

“while”

“a|b”

Regular Expressions

- Special Characters

- | means "or"

- * means zero or more instances of

- + means one or more instances of

- ? means zero or one instance of

- () are used for grouping

- . means any character except newline

- \ means special escape character follows (\n, \t, \", \\, etc.)

- ^ means only match at beginning of a line

- \$ means only match at end of a line

Character Classes

- Characters in square brackets [] create a character class
- [a-z] matches any single lower-case character
- Special Characters in Character Classes:
 - - means range
 - ^ means negate the class (if at beginning)
 - \ escape character (same as before)
 - "" means match actual character (similar to \)
- Example Character Classes:
 - [a b] matches "a", " ", or "b"
 - [a"-z] matches "a", "-", or "z"
 - [a\n] matches "a" or newline (\n)
 - [^a-z] matches anything except lowercase characters
- Whitespace outside character classes terminates the regular expression

.jlex file

User Code

Copied verbatim into
.java file



```
graph TD; UC[User Code] --- P1[%%]; P1 --- JD[JLex directives]; JD --- P2[%%]; P2 --- RER[Regular Expression rules];
```

%%

JLex directives

Includes macros and parameters
For changing how JLex runs

%%

Regular Expression rules

Explains how to divide up user
Input into tokens

JLex Directives

- Directives include
 - specifying the value that should be returned on end-of-file
 - specifying that line counting should be turned on
 - specifying that the scanner will be used with the Java parser generator java cup.
- The directives part also includes **macro definitions**.

Macro Definitions

The form of a macro definition is:

- name = regular-expression
- where name is any valid Java identifier, and regular-expression is any regular expression.

DIGIT= [0-9]

LETTER= [a-zA-Z]

WHITESPACE= [\t\n]

- Using Macros
 - in a regular expression
 - In other macros (only use macros previous defined)
- Example Usage – Use curly braces {}
 {LETTER}({LETTER}|{DIGIT})*

| Symbol | Meaning in RE | Meaning in Char Class |
|--------|---|--|
| (| Matches with) to group sub-expressions. | Represents itself. |
|) | Matches with (to group sub-expressions. | Represents itself. |
| [| Begins a character class. | Represents itself. |
|] | Is illegal. | Ends a character class. |
| { | Matches with } to delimit a macro name. | Matches with } to delimit a macro name. |
| } | Matches with { to delimit a macro name. | Represents itself or matches with { to delimit a macro name. |
| " | Matches with " to delimit strings (only \ is special within strings). | Matches with " for a string of chars that belong to the char class. Only \ is special within the string. |
| \ | Escapes special characters (n, t, etc). Also used unicode/hex/octal. | Escapes characters that are special inside a character class. |
| . | Matches any character except newline | Matches itself |
| | Or | Matches itself |
| * | Kleene Closure | Matches itself |
| + | One or more matches | Matches itself |
| ? | Zero or One matches | Matches itself |
| ^ | Matches only at beginning of line | Complements chars in class |
| \$ | Matches only at end of line | Matches itself |
| - | Matches ifself | Range of characters |

//User Code Section (uninterpreted java code)

%%

//Directives Section

```
DIGIT = [0-9]
LETTER = [a-zA-Z]
WHITESPACE = [\040\t\n]
```

} Macro definitions

%state SPECIALINTSTATE — State declaration

//Configure for use with java CUP (Parser generator)

%implements java_cup.runtime.Scanner

%function next_token

%type java_cup.runtime.Symbol

//End of file behavior

%eofval{

System.out.println("All done");

return null;

%eofval}

//Turn on line counting

%line

%%

//Regular Expression rules

Rules Section

Format is <regex>{code} where regex is a regular expression for a single token

can use macros from the directive sections in regex, surround with curly braces

Conventions

chars represent themselves (except special characters)

chars inside “” represent themselves (except \)

Regex operators

| * + ? () .

Character class operators

- range

^ not

\ escape

Rules Section

```
"="      { System.out.println(yyline + 1 + ": ASSIGN"); }
"+"      { System.out.println(yyline + 1 + ": PLUS"); }
"^"      { System.out.println(yyline + 1 + ": EXP"); }
"<"      { System.out.println(yyline + 1 + ": LT"); }
"+="     { System.out.println(yyline + 1 + ": INC"); }
"<="     { System.out.println(yyline + 1 + ": LEQ"); }
{WHITESPACE} { }
({LETTER}|"_")({DIGIT}|{LETTER}|"_")* {
    System.out.println(yyline+1 + ": ID " + yytext()));}
.      { System.out.println(yyline + 1 + ": badchar"); }
```


More Info On JLex

For a complete description:

<http://www.cs.princeton.edu/~appel/modern/java/JLex/>