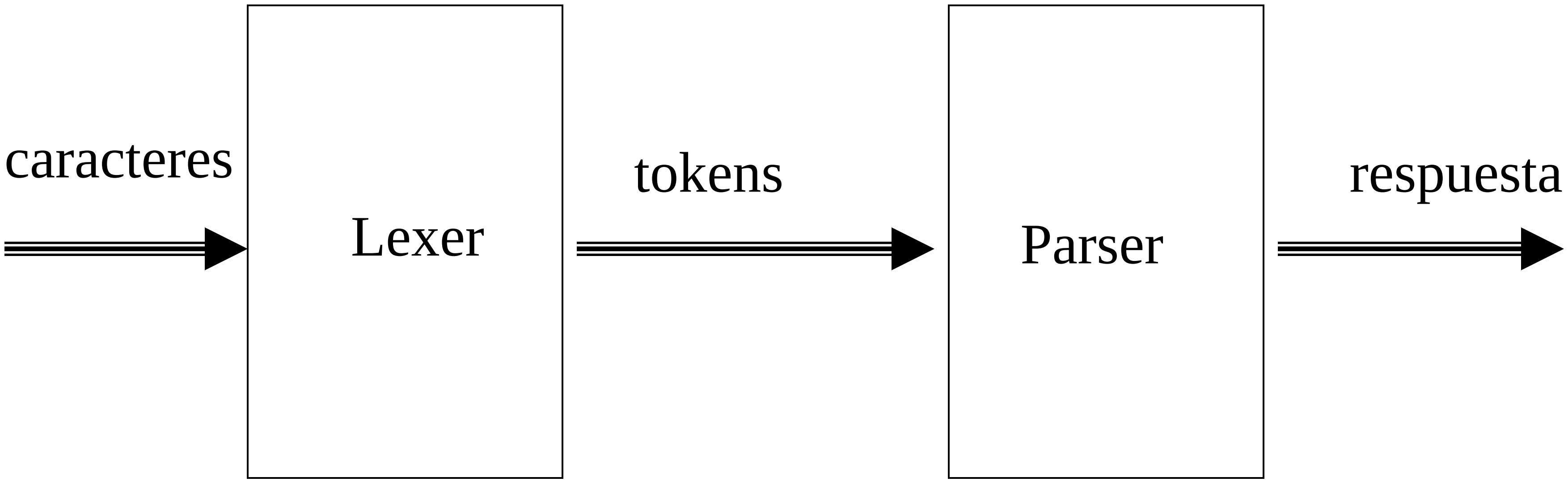
Generadores de Compiladores

JavaCC

Fases en el proceso de análisis de lenguajes



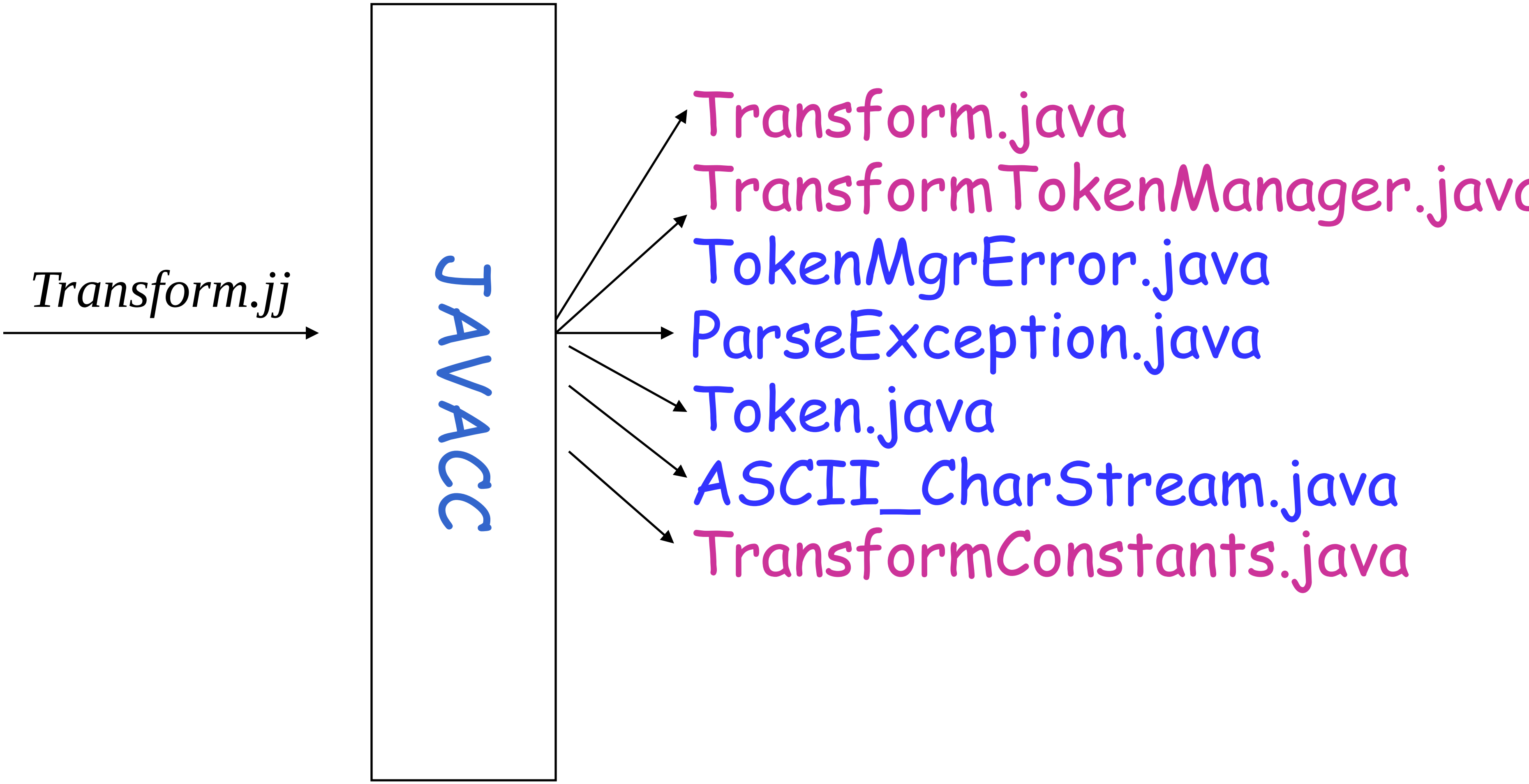
Generadores de Compiladores

- Generadores de analizadores léxicos
- Generadores de analizadores sintácticos

JavaCC

- Genera el lexer
- Genera el parser
- Permite construir una herramienta de análisis de lenguajes fácilmente

¿ Qué hace javaCC ?



Transform.jj

JAVACC

Transform.java

TransformTokenManager.java

TokenMgrError.java

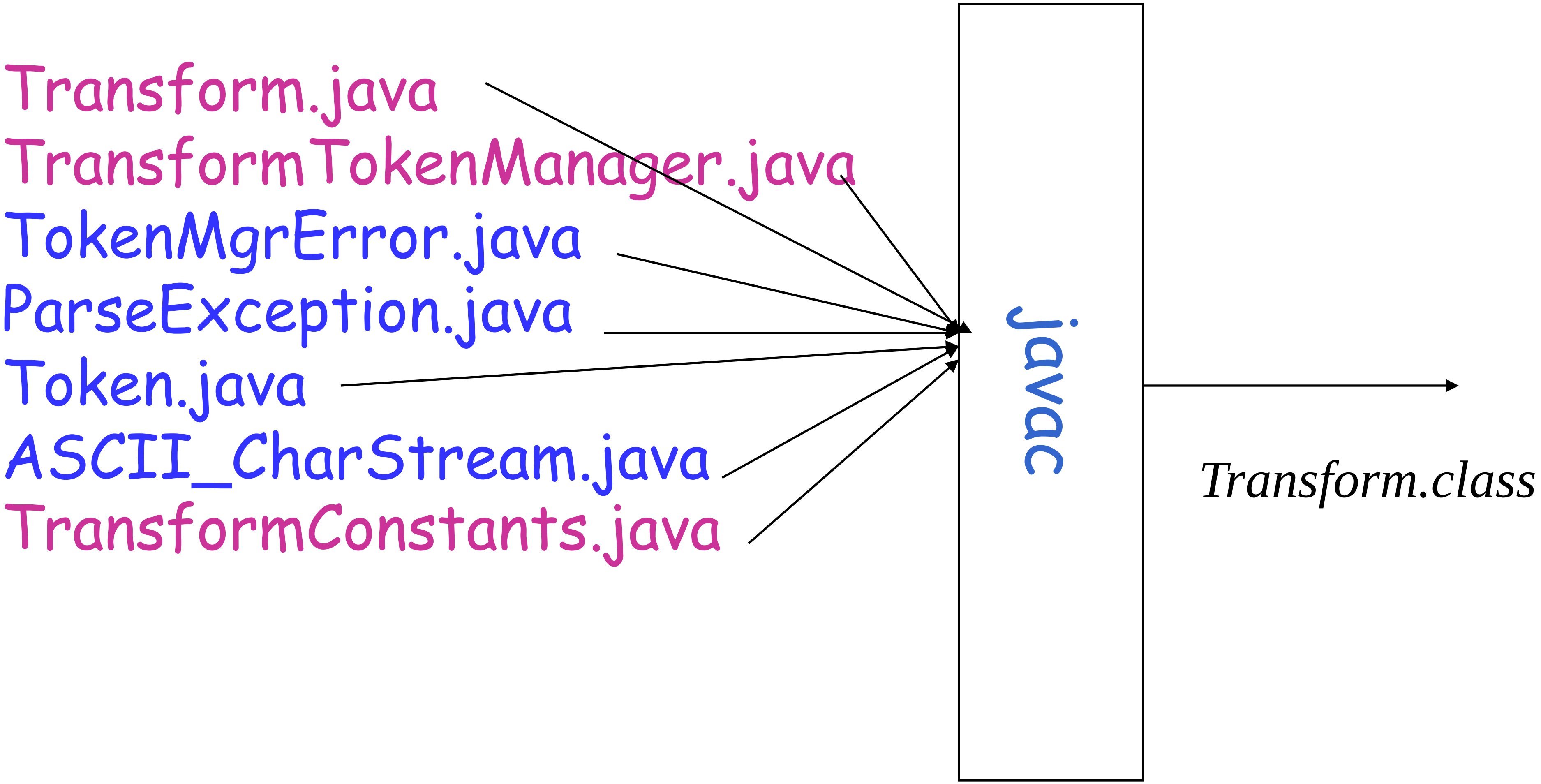
ParseException.java

Token.java

ASCII_CharStream.java

TransformConstants.java

JavaCC

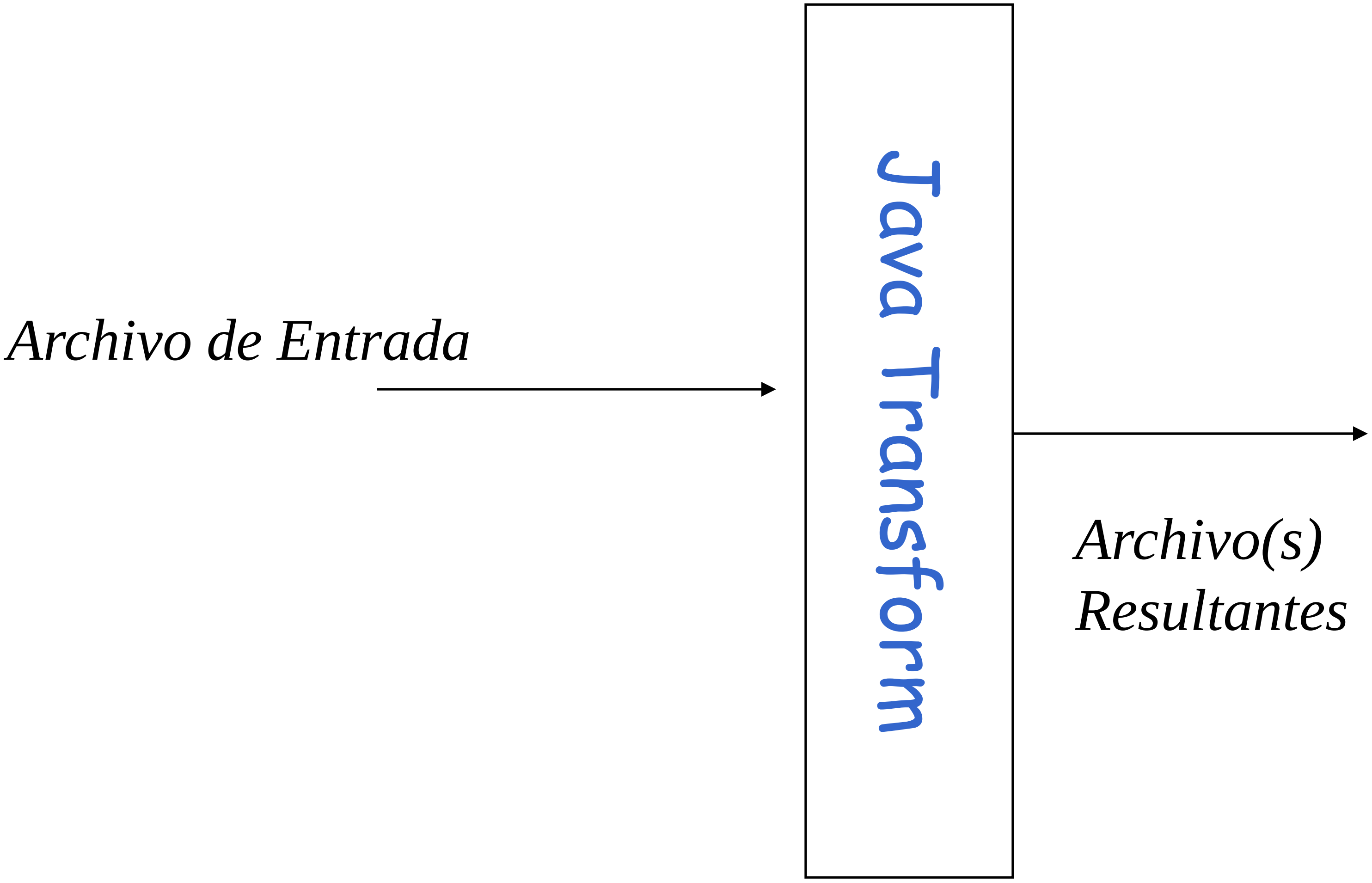


Transform.java
TransformTokenManager.java
TokenMgrError.java
ParseException.java
Token.java
ASCII_CharStream.java
TransformConstants.java

javac

Transform.class

JavaCC



Archivo de Entrada

Java Transform

Archivo(s)
Resultantes

Sintaxis de los archivos para JavaCC

```
javacc_input ::=
   [javacc_options]
   "PARSER_BEGIN" "(" <IDENTIFIER> ")"
   java_compilation_unit
   "PARSER_END" "(" <IDENTIFIER> ")"
   productions
  <EOF>
```

Sintaxis de los archivos para JavaCC

javacc_options ::=

“options" “{" (option_binding)* “}"]

option_binding ::=

“IGNORE_CASE" “=" *java_boolean_literal* “;"

| “OUTPUT_DIRECTORY" “=" *java_string_literal* “;"

Sintaxis de los archivos para JavaCC

```
productions ::=
     (production)* [tok_mng_decls]
   (production)*


production ::=
          javacode_production
        | regular_exp_production
        | bnf_production
```

Sintaxis de los archivos para JavaCC

```
Regular_expr_production ::=
  [ lexical_state_list ]
     regexp_kind [ “[“ “IGNORE_CASE” “]” ]
   “{“regexp_spec (“|” regexp_spec )*“}”
```

Sintaxis de los archivos para JavaCC

```
regexpr_kind   ::= "TOKEN"
                   | "SPECIAL_TOKEN"
                   | "SKIP"
                   | "MORE"

regexpr_spec ::=

  regular_expression [ java_block ] [ ":" java_identifier ]
```

Sintaxis de los archivos para JavaCC

```
regular_expression ::=
        java_string_literal
    | “<" [ [ "#" ] java_identifier ":" ]
                        complex_reg_exp_choices
        ">"
    | "<" java_identifier ">"
    |"<" "EOF" ">"
```

Sintaxis de los archivos para JavaCC

```
complex_reg_exp_choices ::=
    complex_regular_expression
   ( "|" complex_regular_expression )*

complex_regular_expression::=
    (complex_regular_expression_unit )*
```

Sintaxis de los archivos para JavaCC

```
complex_regular_express

        java_string_literal
    |   "<" java_identifier ">"
    |   character_list
    |   "(" complex_reg_exp_choices ")" [ "+" | "*" | "?" ]
```

Ojo: note que al agregar el símbolo +, * o ? . La expresión al la que se aplica debe estar entre paréntesis!

Sintaxis de los archivos para JavaCC

character_list ::=

["~"] "[" [char_descriptor ("," character_descriptor)*] “]"

char_descriptor ::=

java_string_literal ["-" *java_string_literal*]

Sintaxis de los archivos para JavaCC

bnf_production ::=

java_return_type

java_identifier "(" *java_parameter_list* ")" ":"

java_block

"{" expansion_choices "}“

expansion_choices ::= expansion ("|" expansion)*

expansion ::= (expansion_unit)*

Sintaxis de los archivos para JavaCC

```
expansion_unit ::=
     java_block
|    "(" expansion_choices ")" [ "+" |      |  ?  ]
|    "[" expansion_choices "]"
|     [ java_assignment_lhs "=" ] regular_expression
|     [java_assignment_lhs "=" ] java_method_invocation
```

Ojo: note que al agregar el símbolo +, * o ? . La expresión al la que se aplica debe estar entre paréntesis!