# Analizador Sintactico
# Compiladores e Intérpretes
# 2012

Garay, Iñaki LU 67387

25 de septiembre de 2012

# Índice general

## 0.1. Uso

### 0.1.1. Invocación y uso

Utilizando el intérprete de Python para correr el script del módulo principal:

```
C:\> python.exe parser_main.py <archivo_entrada> [<archivo_salida>]
```

La especificación del archivo de salida es opcional. Si se omite, el analizador sintactico mostrará su salida por pantalla. Si no se especifica un archivo de entrada, o se pasan más argumentos que el archivo de entrada y el de salida, el analizador léxico mostrará un mensaje explicando el uso por linea de comando.

Los archivos de prueba se encuentran en el directorio `/tests/parser`. Para correrlos:

```
$ python2 parser_main.py tests/parser/001-cor-minimal.java
```

El analizador sintactico también emite un archivo `DEBUG.log`, el cual contiene una traza de los procedimientos llamados. Cada linea de `DEBUG.log` tiene una de los siguientes formatos:

- `>procedimiento` indica que se entro en el procedimiento, los cuales tienen los mismos nombres que los no terminales en la gramatica, por lo cual se puede relacionar el procedimiento con la regla de produccion de manera relativamente facil.
- `<procedimiento` indica que se salio del procedimiento.
- `p procedimiento` indica que se predice que se va a tomar una produccion del no terminal procedimiento, esto sucede siempre cuando hay varias alternativas de reglas a tomar, y el token actual pertenece a $FIRST(procedimiento)$.
- `m token` indica que se logro un match entre el token actual en el lookahead y el token esperado por la regla predicha por el analizador sintactico. La alternativa a que se muestre este mensaje es que no se produjo un match, se produceuna excepcion y se muestra el mensaje de error correspondiente.

A continuacion se muestra la salida del analizador sintactico para el caso de prueba correcto minimal:

```
$ python2 parser_main.py tests/parser/001-cor-minimal.java
1:0     -   <TK_CLASSDEF>          :: classDef
2:8     -   <TK_IDENTIFIER>        :: id
2:11    -   <TK_BRACE_OPEN>        :: {
2:13    -   <TK_IDENTIFIER>        :: id
3:6     -   <TK_PAREN_OPEN>        :: (
3:7     -   <TK_PAREN_CLOSE>       :: )
3:8     -   <TK_SEMICOLON>         :: ;
4:0     -   <TK_BRACE_CLOSE>       :: }
5:0     -   <TK_CLASS>             :: class
```

```
6:5     -   <TK_IDENTIFIER>         :: id
6:8     -   <TK_BRACE_OPEN>         :: {
7:0     -   <TK_IDENTIFIER>         :: id
7:6     -   <TK_PAREN_OPEN>         :: (
7:7     -   <TK_PAREN_CLOSE>        :: )
7:8     -   <TK_BRACE_OPEN>         :: {
7:10    -   <TK_BRACE_CLOSE>        :: }
8:0     -   <TK_BRACE_CLOSE>        :: }
9:0     -   <TK_EOF>                :: EOF
La sintaxis de tests/parser/001-cor-minimal.java es correcta.
```

A continuacion se muestra el contenido de `DEBUG.log` para el mismo caso de prueba:

```
$ cat DEBUG.log
> start
> classdef_start
> classdef
m TK_CLASSDEF
m TK_IDENTIFIER
> superr
< superr
> classdef_body
m TK_BRACE_OPEN
> classdef_body_rest
m TK_IDENTIFIER
> classdef_ctor_or_method
> classdef_ctor_rest
> formal_args
m TK_PAREN_OPEN
> formal_args_rest
m TK_PAREN_CLOSE
< formal_args_rest
< formal_args
m TK_SEMICOLON
> classdef_body_rest
< classdef_body_rest
< classdef_ctor_rest
< classdef_ctor_or_method
< classdef_body_rest
m TK_BRACE_CLOSE
< classdef_body
< classdef
> classdef_start_rest
< classdef_start_rest
< classdef_start
> class_start
> classs
m TK_CLASS
m TK_IDENTIFIER
```

```
> superr
< superr
> class_body
m TK_BRACE_OPEN
> class_body_rest
m TK_IDENTIFIER
> class_field_ctor_or_method
> class_ctor_rest
> formal_args
m TK_PAREN_OPEN
> formal_args_rest
m TK_PAREN_CLOSE
< formal_args_rest
< formal_args
> block
m TK_BRACE_OPEN
> statements
< statements
m TK_BRACE_CLOSE
< block
> class_ctor_or_method
< class_ctor_or_method
< class_ctor_rest
< class_field_ctor_or_method
< class_body_rest
m TK_BRACE_CLOSE
< class_body
< classs
> class_start_rest
< class_start_rest
< class_start
< start
SYNTAX OK
```

## 0.2. Gramática

La gramática en notacion BNF sin extensiones, con recursion a izquierda eliminada y factorizada se presenta a continuación:

```
start                       ::= classdef_start class_start

classdef_start              ::= classdef classdef_start_rest

classdef_start_rest         ::= classdef classdef_start_rest
classdef_start_rest         ::= LAMBDA

classdef                    ::= TK_CLASSDEF TK_IDENTIFIER
                                super classdef_body
```

```
classdef_body               ::= TK_BRACE_OPEN
                                classdef_body_rest
                                TK_BRACE_CLOSE

classdef_body_rest          ::= TK_IDENTIFIER classdef_ctor_or_method
classdef_body_rest          ::= primitive_type_void TK_IDENTIFIER
                                classdef_method_rest
classdef_body_rest          ::= LAMBDA

classdef_ctor_or_method     ::= TK_IDENTIFIER classdef_method_rest
classdef_ctor_or_method     ::= classdef_ctor_rest

classdef_ctor_rest          ::= formal_args TK_SEMICOLON
                                classdef_body_rest

classdef_method_rest        ::= formal_args TK_SEMICOLON
                                classdef_methods

classdef_methods            ::= classdef_method classdef_methods
classdef_methods            ::= LAMBDA

classdef_method             ::= method_type TK_IDENTIFIER
                                formal_args TK_SEMICOLON

class_start                 ::= class class_start_rest

class_start_rest            ::= class class_start_rest
class_start_rest            ::= LAMBDA

class                       ::= TK_CLASS TK_IDENTIFIER super class_body

class_body                  ::= TK_BRACE_OPEN class_body_rest TK_BRACE_CLOSE

class_body_rest             ::= TK_IDENTIFIER class_field_ctor_or_method
class_body_rest             ::= TK_VOID TK_IDENTIFIER class_method_rest
class_body_rest             ::= primitive_type TK_IDENTIFIER
                                class_field_or_method
class_body_rest             ::= LAMBDA

class_field_ctor_or_method  ::= TK_IDENTIFIER class_field_or_method
class_field_ctor_or_method  ::= class_ctor_rest

class_field_or_method       ::= class_var_declaration_list
class_field_or_method       ::= class_method_rest

class_var_declaration_list  ::= TK_COMMA TK_IDENTIFIER
                                class_var_declaration_list
class_var_declaration_list  ::= TK_SEMICOLON class_body_rest
```

```
class_method_rest           ::= formal_args block class_methods

class_methods               ::= class_method class_methods
class_methods               ::= LAMBDA

class_method                ::= method_type TK_IDENTIFIER
                                formal_args block

class_ctor_rest             ::= formal_args block class_ctor_or_method

class_ctor_or_method        ::= TK_IDENTIFIER class_ctor_or_method_rest
class_ctor_or_method        ::= primitive_type_void TK_IDENTIFIER
                                class_method_rest
class_ctor_or_method        ::= LAMBDA

class_ctor_or_method_rest   ::= TK_IDENTIFIER class_method_rest
class_ctor_or_method_rest   ::= class_ctor_rest

super                       ::= TK_EXTENDS TK_IDENTIFIER
super                       ::= LAMBDA

formal_args                 ::= TK_PAREN_OPEN formal_args_rest

formal_args_rest            ::= TK_PAREN_CLOSE
formal_args_rest            ::= formal_arg_list TK_PAREN_CLOSE

formal_arg_list             ::= formal_arg formal_arg_list_rest

formal_arg_list_rest        ::= TK_COMMA formal_arg formal_arg_list_rest
formal_arg_list_rest        ::= LAMBDA

formal_arg                  ::= type TK_IDENTIFIER

method_type                 ::= TK_VOID
method_type                 ::= type

type                        ::= TK_IDENTIFIER
type                        ::= primitive_type

primitive_type_void         ::= void
primitive_type_void         ::= primitive_type

primitive_type              ::= TK_BOOLEAN
primitive_type              ::= TK_CHAR
primitive_type              ::= TK_INT
primitive_type              ::= TK_STRING

block                       ::= TK_BRACE_OPEN statements TK_BRACE_CLOSE

statements                  ::= statement statements
```

```
statements                  ::= LAMBDA

statement                   ::= closed_statement
statement                   ::= open_statement

open_statement              ::= TK_IF TK_PAREN_OPEN expression
                                TK_PAREN_CLOSE statement

open_statement              ::= TK_IF TK_PAREN_OPEN expression
                                TK_PAREN_CLOSE closed_statement
                                TK_ELSE open_statement

open_statement              ::= TK_FOR
                                TK_PAREN_OPEN expression
                                TK_SEMICOLON expression
                                TK_SEMICOLON expression
                                TK_PAREN_CLOSE open_statement

open_statement              ::= TK_WHILE TK_PAREN_OPEN expression
                                TK_PAREN_CLOSE open_statement

closed_statement            ::= TK_IF
                                TK_PAREN_OPEN expression TK_PAREN_CLOSE
                                closed_statement
                                TK_ELSE closed_statement

closed_statement            ::= TK_FOR
                                TK_PAREN_OPEN expression
                                TK_SEMICOLON expression
                                TK_SEMICOLON expression
                                TK_PAREN_CLOSE closed_statement

closed_statement            ::= TK_WHILE
                                TK_PAREN_OPEN expression TK_PAREN_CLOSE
                                closed_statement

closed_statement            ::= simple_statement

simple_statement            ::= TK_SEMICOLON
simple_statement            ::= TK_RETURN statement_return_rest
simple_statement            ::= block
simple_statement            ::= expression

statement_return_rest       ::= TK_SEMICOLON
statement_return_rest       ::= expression TK_SEMICOLON

expression                  ::= assignment_expr

assignment_expr             ::= TK_ASSIGNMENT expression
assignment_expr             ::= LAMBDA
```

```
logical_expr                ::= logical_or_expr logical_expr_rest

logical_expr_rest           ::= logical_expr
logical_expr_rest           ::= LAMBDA

logical_or_expr             ::= logical_and_expr logical_or_expr_rest

logical_or_expr_rest        ::= TK_OR logical_or_expr
logical_or_expr_rest        ::= LAMBDA

logical_and_expr            ::= equality_expr logical_and_expr_rest

logical_and_expr_rest       ::= TK_AND logical_and_expr
logical_and_expr_rest       ::= LAMBDA

equality_expr               ::= relational_expr equality_expr_rest

equality_expr_rest          ::= TK_EQUALS    equality_expr
equality_expr_rest          ::= TK_NOTEQUALS equality_expr
equality_expr_rest          ::= LAMBDA

relational_expr             ::= term_expr relational_expr_rest

relational_expr_rest        ::= TK_LT   relational_expr
relational_expr_rest        ::= TK_GT   relational_expr
relational_expr_rest        ::= TK_LTEQ relational_expr
relational_expr_rest        ::= TK_GTEQ relational_expr
relational_expr_rest        ::= LAMBDA

term_expr                   ::= factor_expr term_expr_rest

term_expr_rest              ::= TK_ADD term_expr
term_expr_rest              ::= TK_SUB term_expr
term_expr_rest              ::= LAMBDA

factor_expr                 ::= unary_expr factor_expr_rest

factor_expr_rest            ::= TK_MUL factor_expr
factor_expr_rest            ::= TK_DIV factor_expr
factor_expr_rest            ::= TK_MOD factor_expr
factor_expr_rest            ::= LAMBDA

unary_expr                  ::= TK_ADD unary_expr
unary_expr                  ::= TK_SUB unary_expr
unary_expr                  ::= TK_NOT unary_expr
unary_expr                  ::= primary

primary                     ::= literal primary_rest
primary                     ::= TK_PAREN_OPEN expression TK_PAREN_CLOSE
```

```
                                primary_rest
primary                     ::= TK_NEW TK_IDENTIFIER actual_args
                                primary_rest
primary                     ::= TK_SUPER TK_PERIOD
                                TK_IDENTIFIER actual_args
                                primary_rest
primary                     ::= TK_THIS primary_rest_this
                                primary_rest
primary                     ::= TK_IDENTIFIER primary_rest_id
                                primary_rest

primary_rest                ::= TK_PERIOD TK_IDENTIFIER actual_args
                                primary_rest
primary_rest                ::= LAMBDA

primary_rest_this           ::= TK_PERIOD TK_IDENTIFIER
primary_rest_this           ::= LAMBDA

primary_rest_id             ::= actual_args
primary_rest_id             ::= LAMBDA

literal                     ::= TK_NULL
literal                     ::= TK_TRUE
literal                     ::= TK_FALSE
literal                     ::= TK_INT_LITERAL
literal                     ::= TK_CHAR_LITERAL
literal                     ::= TK_STRING_LITERAL

actual_args                 ::= TK_PAREN_OPEN actual_args_rest
j
actual_args_rest            ::= TK_PAREN_CLOSE
actual_args_rest            ::= expr_list TK_PAREN_CLOSE

expr_list                   ::= expression expr_list_rest

expr_list_rest              ::= TK_COMMA expr_list
expr_list_rest              ::= LAMBDA
```

### 0.2.1. Cambios realizados:

Primary original:

```
<primary>  ::= <identifier>
<primary>  ::= <new_expr>
<new_expr> ::= <literal>
<new_expr> ::= this
<new_expr> ::= this . <identifier>
<new_expr> ::= ( <expression> )
<new_expr> ::= new <identifier> <actual_args>
```

```
<new_expr> ::= <identifier> <actual_args>
<new_expr> ::= super . <identifier> <actual_args>
<new_expr> ::= <primary> . <identifier> <actual_args>
```

Reemplazo terminales por tokens:

```
<primary>  ::= IDENTIFIER
<primary>  ::= <new_expr>
<new_expr> ::= <literal>
<new_expr> ::= THIS
<new_expr> ::= THIS PERIOD IDENTIFIER
<new_expr> ::= PAREN_OPEN <expression> PAREN_CLOSE
<new_expr> ::= NEW IDENTIFIER <actual_args>
<new_expr> ::= IDENTIFIER <actual_args>
<new_expr> ::= SUPER PERIOD IDENTIFIER <actual_args>
<new_expr> ::= <primary> PERIOD IDENTIFIER <actual_args>
```

Lifteo new_expr:

```
<primary>  ::= IDENTIFIER
<primary>  ::= <literal>
<primary>  ::= THIS
<primary>  ::= THIS PERIOD IDENTIFIER
<primary>  ::= PAREN_OPEN <expression> PAREN_CLOSE
<primary>  ::= NEW IDENTIFIER <actual_args>
<primary>  ::= IDENTIFIER <actual_args>
<primary>  ::= SUPER PERIOD IDENTIFIER <actual_args>
<primary>  ::= <primary> PERIOD IDENTIFIER <actual_args>
```

Reordeno para ver prefijos comunes mejor:

```
<primary>  ::= <literal>
<primary>  ::= PAREN_OPEN <expression> PAREN_CLOSE
<primary>  ::= NEW IDENTIFIER <actual_args>
<primary>  ::= SUPER PERIOD IDENTIFIER <actual_args>
<primary>  ::= THIS
<primary>  ::= THIS PERIOD IDENTIFIER
<primary>  ::= IDENTIFIER
<primary>  ::= IDENTIFIER <actual_args>
<primary>  ::= <primary> PERIOD IDENTIFIER <actual_args>
```

Factorizo this:

```
<primary>  ::= <literal>
<primary>  ::= PAREN_OPEN <expression> PAREN_CLOSE
<primary>  ::= NEW IDENTIFIER <actual_args>
<primary>  ::= SUPER PERIOD IDENTIFIER <actual_args>

<primary>           ::= THIS <primary_rest_this>
<primary_rest_this> ::= LAMBDA
```

```
<primary_rest_this> ::= PERIOD IDENTIFIER

<primary>  ::= IDENTIFIER
<primary>  ::= IDENTIFIER <actual_args>
<primary>  ::= <primary> PERIOD IDENTIFIER <actual_args>
```

Factorizo identifier:

```
<primary> ::= <literal>
<primary> ::= PAREN_OPEN <expression> PAREN_CLOSE
<primary> ::= NEW IDENTIFIER <actual_args>
<primary> ::= SUPER PERIOD IDENTIFIER <actual_args>

<primary>           ::= THIS <primary_rest_this>
<primary_rest_this> ::= LAMBDA
<primary_rest_this> ::= PERIOD IDENTIFIER

<primary>           ::= IDENTIFIER <primary_rest_id>
<primary_rest_id>   ::= LAMBDA
<primary_rest_id>   ::= <actual_args>

<primary> ::= <primary> PERIOD IDENTIFIER <actual_args>
```

Elimino la recursion a izq de primary:

```
<primary> ::= <literal>                              <primary_rest>
<primary> ::= PAREN_OPEN <expression> PAREN_CLOSE    <primary_rest>
<primary> ::= NEW IDENTIFIER <actual_args>           <primary_rest>
<primary> ::= SUPER PERIOD IDENTIFIER <actual_args>  <primary_rest>
<primary> ::= THIS <primary_rest_this>               <primary_rest>
<primary> ::= IDENTIFIER <primary_rest_id>           <primary_rest>

<primary_rest> ::= LAMBDA
<primary_rest> ::= PERIOD IDENTIFIER <actual_args>

<primary_rest_this> ::= LAMBDA
<primary_rest_this> ::= PERIOD IDENTIFIER <primary_rest_id>

<primary_rest_id>   ::= LAMBDA
<primary_rest_id>   ::= <actual_args>
```

### 0.2.2. FIRST y FOLLOW

Se calcularon los conjuntos $FIRST()$ y $FOLLOW()$ para los no terminales pertinentes y se los codifico en el archivo `first_follow.py`.

El codigo es sumamente declarativo e indica que tokens pertenecen a cada conjunto. La funcion `set()` crea un conjunto y la expresion `s1 | s2` produce la union de los conjuntos s1 y s2, y es generalizable a mas de dos conjuntos.

Los conjuntos se guardan en variables con el nombres de la forma `first_ noterminal`y `follow_ noterminal`, donde `noterminal` corresponde con el nombre del simbolo no terminal asociado para el cual se calcula el conjunto $FIRST()$ y $FOLLOW()$ respectivamente..

El orden en que estan definidos los conjuntos lamentablemente no corresponde con el orden de los no terminales correspondientes en la gramatica, pero resulta inevitable por la necesidad de tener definidos previamente conjuntos que son incluidos en otros.

```
primitive_types                  = set([TK_BOOLEAN, TK_CHAR,
                                   TK_INT, TK_STRING])

void_type                        = set([TK_VOID])

identifiers                      = set([TK_IDENTIFIER])

method_types                     = primitive_types |
                                   void_type | identifiers

literals                         = set([TK_NULL, TK_TRUE, TK_FALSE,
                                   TK_INT_LITERAL, TK_CHAR_LITERAL,
                                   TK_STRING_LITERAL])

if_for_while                     = set([TK_IF, TK_FOR, TK_WHILE])

unary_operators                  = set([TK_ADD, TK_SUB, TK_NOT])

first_actual_args                = set([TK_PAREN_OPEN])

first_block                      = set([TK_BRACE_OPEN])

first_classdef_start             = set([TK_CLASSDEF])
first_classdef_ctor_rest         = set([TK_PAREN_OPEN])
first_classdef_method            = method_types

first_class                      = set([TK_CLASS])
first_class_ctor_rest            = set([TK_PAREN_OPEN])
first_class_method               = method_types
first_class_method_rest          = set([TK_PAREN_OPEN])
first_class_start                = set([TK_CLASS])
first_class_var_declaration_list = set([TK_COMMA, TK_SEMICOLON])

first_primitive_type             = primitive_types
first_primitive_type_void        = primitive_types | void_type

first_primary                    = identifiers | literals |
                                   set([TK_PAREN_OPEN, TK_NEW,
                                   TK_SUPER, TK_THIS])
```

```
first_expression                  = unary_operators | first_primary

first_expr_list                   = first_expression

first_formal_arg_list             = primitive_types | identifiers

first_literal                     = literals

first_logical_expr                = unary_operators | first_primary

first_simple_statement            = first_logical_expr |
                                    set([TK_SEMICOLON, TK_RETURN,
                                    TK_BRACE_OPEN])

first_closed_statement            = first_simple_statement | if_for_while

first_open_statement              = if_for_while

first_statement                   = first_closed_statement

follow_classdef_body_rest         = set([TK_BRACE_CLOSE])

follow_classdef_ctor_or_method    = follow_classdef_body_rest

follow_classdef_ctor_rest         = follow_classdef_ctor_or_method

follow_classdef_method_rest       = follow_classdef_body_rest |
                                    follow_classdef_ctor_or_method

follow_classdef_methods           = follow_classdef_method_rest

follow_class_start                = set([TK_EOF])

follow_class_start_rest           = follow_class_start

follow_class_body_rest            = set([TK_BRACE_CLOSE])

first_class_body_rest             = identifiers | void_type |
                                    primitive_types |
                                    set([TK_BRACE_CLOSE])

follow_class_ctor_or_method       = first_class_body_rest

follow_class_ctor_rest            = follow_class_ctor_or_method

follow_class_field_ctor_or_method = follow_class_body_rest

follow_class_field_or_method      = follow_class_field_ctor_or_method

follow_class_method_rest          = follow_class_body_rest |
```

```
                                  follow_class_field_or_method

follow_class_var_declaration_list = follow_class_field_or_method

follow_class_methods              = follow_class_method_rest

follow_super                      = set([TK_BRACE_OPEN])
first_super                       = set([TK_EXTENDS]) | follow_super

follow_formal_arg_list            = set([TK_PAREN_CLOSE])
follow_formal_arg_list_rest       = follow_formal_arg_list

follow_assignment_expr            = set([TK_SEMICOLON,
                                  TK_PAREN_CLOSE, TK_COMMA])

follow_logical_expr               = set([TK_SEMICOLON,
                                  TK_PAREN_CLOSE, TK_ASSIGNMENT,
                                  TK_COMMA])

follow_logical_expr_rest          = follow_logical_expr

follow_logical_or_expr_rest       = first_expression | follow_logical_expr

first_logical_or_expr_rest        = set([TK_OR]) |
                                  follow_logical_or_expr_rest

follow_logical_and_expr_rest      = first_logical_or_expr_rest

first_logical_and_expr_rest       = set([TK_AND]) |
                                  follow_logical_and_expr_rest

follow_equality_expr              = first_logical_and_expr_rest

follow_equality_expr_rest         = follow_equality_expr

first_equality_expr_rest          = set([TK_EQUALS, TK_NOTEQUALS]) |
                                  follow_equality_expr_rest

follow_relational_expr            = first_equality_expr_rest

follow_relational_expr_rest       = follow_relational_expr

first_relational_expr_rest        = set([TK_LT, TK_GT, TK_LTEQ, TK_GTEQ]) |
                                  follow_relational_expr

follow_term_expr                  = first_relational_expr_rest

follow_term_expr_rest             = follow_term_expr

first_term_expr_rest              = set([TK_ADD, TK_SUB]) |
```

```
                                  follow_term_expr_rest

follow_factor_expr                = first_term_expr_rest

follow_factor_expr_rest           = follow_factor_expr

first_factor_expr_rest            = set([TK_MUL, TK_DIV, TK_MOD]) |
                                  follow_factor_expr_rest

follow_unary_expr                 = first_factor_expr_rest

follow_primary                    = follow_unary_expr

follow_primary_rest               = follow_primary

follow_primary_rest_id            = first_primary

follow_primary_rest_this          = first_primary

follow_expr_list_rest             = set([TK_PAREN_CLOSE])

follow_statements                 = set([TK_BRACE_CLOSE])
```

## 0.3. Verificación

### 0.3.1. Errores detectados

El analizador léxico reconoce los siguientes tipos de errores:

- Comentario del tipo /* */ que no esté propiamente cerrado.
- Informa de tokens no reconocidos.
- Caracter no reconocido: si se intenta ingresar un caracter que no pertenece al alfabeto se producirá un error.
- Si el archivo de entrada especificado no existe, se producirá un error.
- Si se encuentra un caracter literal de mas de un caracter, e.g. 'hola', se producirá un error. En cambio, si es un caracter válido, e.g. 'n', se aceptará.

El analizador sintactico reconoce los siguientes tipos de errores:

### 0.3.2. Casos de prueba

Los dos casos de prueba iniciales, 001 y 002 tienen formas sentenciales validas. El primero es lo minimo que reconoce la gramatica, y el segundo es mas abarcativo e intenta activart todas las producciones.

El resto de los casos de prueba disparan el reconocimiento de algun error. Cada excepcion que puede levantar el codigo del analizador tiene un caso de prueba correspondiente.

Cada caso de prueba contiene un comentario indicando su proposito y el error cuya dateccion intenta producir en el analizador sintactico.

- 001-cor-minimal.java
- 002-cor-full.java
- 003-err-noclassdef.java
- 004-err-noclass.java
- 005-err-badclassid.java
- 006-err-noclassdefbodybraceopen.java
- 007-err-noclassdefbodybraceclose.java
- 008-err-classdefmethodbadidentifier.java
- 009-err-badmethodidenitifier.java
- 010-err-badconstructortermination.java
- 011-err-classdefmethoddeclnosemicolon.java