

# SIDL

---

## Simple Drawing Language

Enrico Zamagni

Attività progettuale per il corso di  
Linguaggi e Modelli Computazionali M  
Prof. Enrico Denti

# Obiettivo

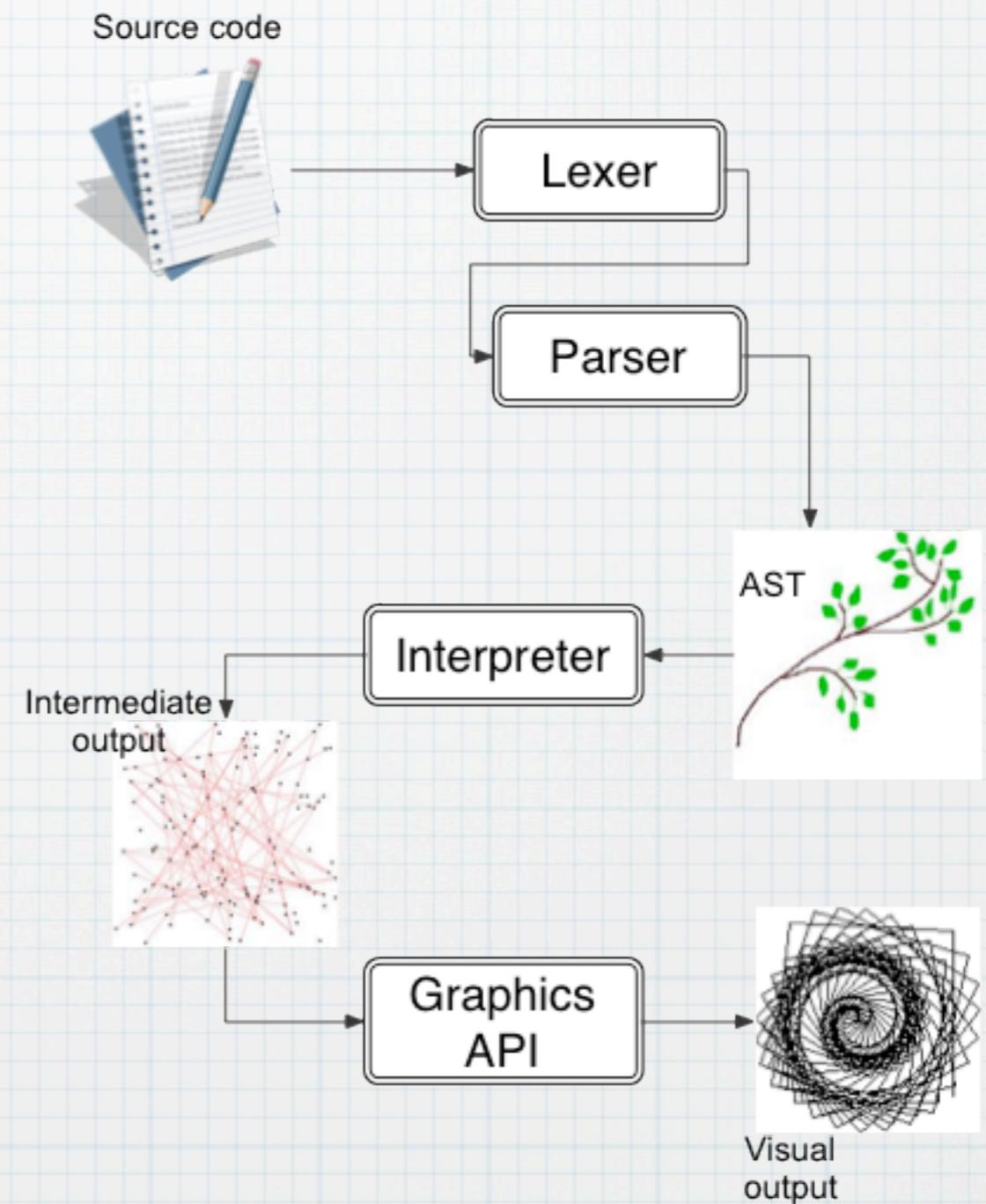
- \* Un linguaggio per la generazione di “turtle graphics”
- \* Ispirato al linguaggio Logo
- \* Le istruzioni comandano un cursore virtuale su un piano cartesiano bidimensionale
- \* Tre operazioni fondamentali
  - \* Spostamento (4 direzioni)
  - \* Orientamento (orario o antiorario)
  - \* Controllo penna (alzata o abbassata)

# Caratteristiche

- \* Sintassi semplice e naturale
- \* Solo funzionalità essenziali
  - \* Procedure
  - \* Funzioni
  - \* Variabili (unico tipo: numeri a precisione singola)
  - \* Istruzioni di controllo
  - \* Espressioni numeriche e logiche
- \* Ambiente di esecuzione minimale
  - \* Single-threaded
  - \* Scoping lessicale e locale

# Architettura

- \* Lexer e Parser generati automaticamente
- \* Interprete realizzato con pattern Visitor
  - \* possibilità di più interpreti per output differenziati



# Scelte implementative

- \* ANTLR v3.4
  - \* Sviluppato attivamente e ben documentato
  - \* IDE versatile e funzionale (ANTLRWorks)
  - \* Supporto nativo per AST
  - \* Buona rilevazione di errori
- \* Java 6
- \* GUI minimale

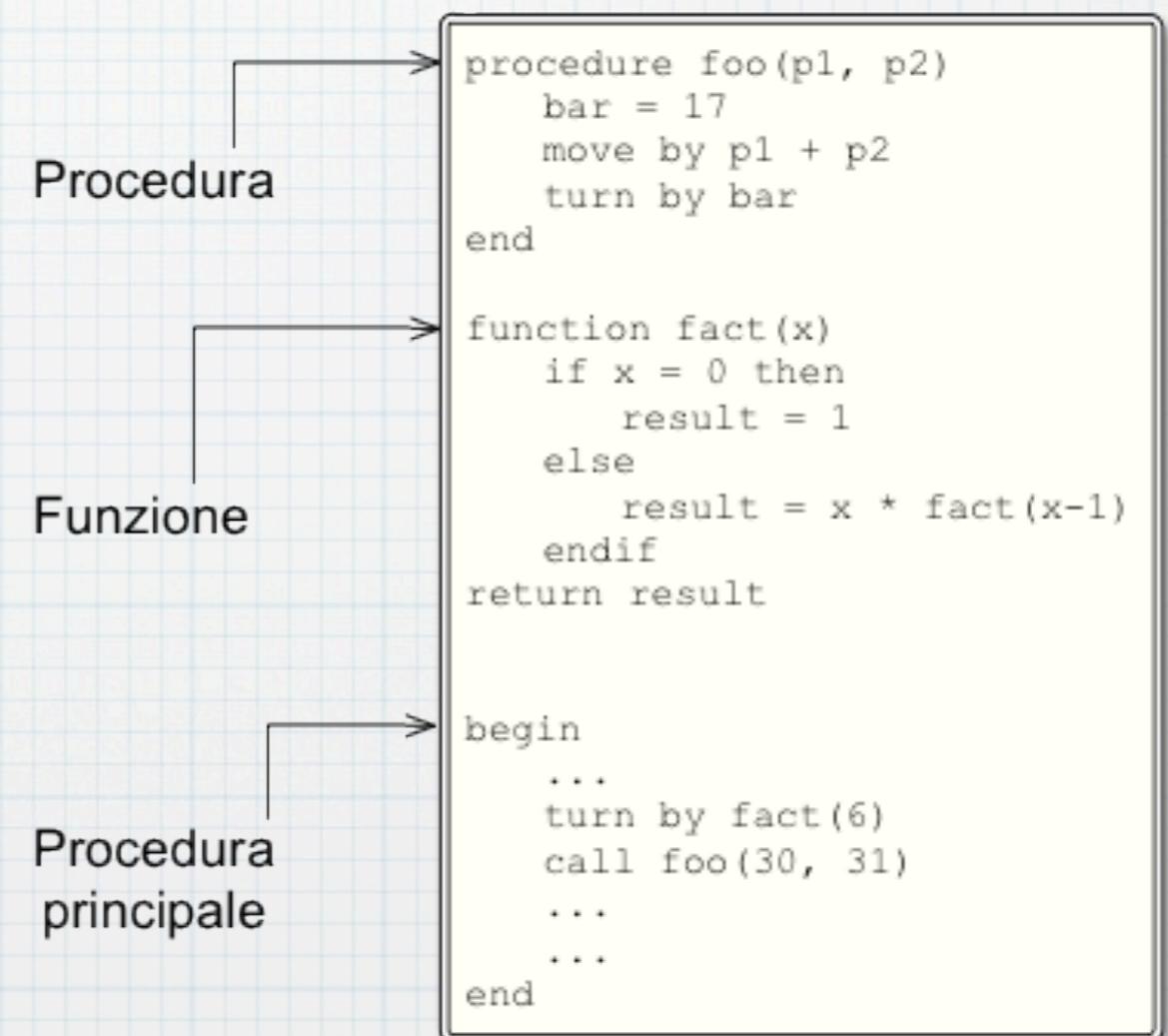
Grammatica

# Struttura generale

**program**

: (procedure | function | NL) \* mainproc ;

- \* Programma come collezione di funzioni e procedure seguite dalla procedura principale
- \* Funzioni e procedure sono entità diverse
- \* La procedura principale è una speciale procedura senza nome e parametri
- \* Scope sempre locale a funzioni e procedure
- \* Grammatica organizzata su due livelli
  - \* Sintassi del linguaggio
  - \* Sintassi espressioni (logiche e numeriche)



# Procedure

```
procedure
:   'procedure' ID ('(' idlist? ')')? NL block 'end' NL ;

mainproc
:   'begin' NL block 'end' ;

idlist  :   ID (',' ID)* ;

block
:   (statement | NL)+ ;

procedure_call
: 'call' ID ('(' num_expr_list? ')')? ;
```

- \* Finalizzate al controllo dello spostamento del cursore
- \* Possono essere richiamate solo da altre procedure
- \* Condivisione dati solo via parametri (passati per valore)

```
ID
:   (CHAR|'_')
    (CHAR | DIGIT | '_')* ;

fragment CHAR
: 'a'...'z' | 'A'...'Z';
```

# Istruzioni

```
block
:   (statement | NL) + ;

statement
:   (assignment | action | procedure_call | control) NL ;

NL   :   '\r'? '\n' ;
```

- \* Istruzioni sempre terminate da token NL
- \* Quattro tipi di istruzioni
  - \* assegnamenti
  - \* azioni
  - \* chiamate a procedura
  - \* istruzioni di controllo

# Azioni

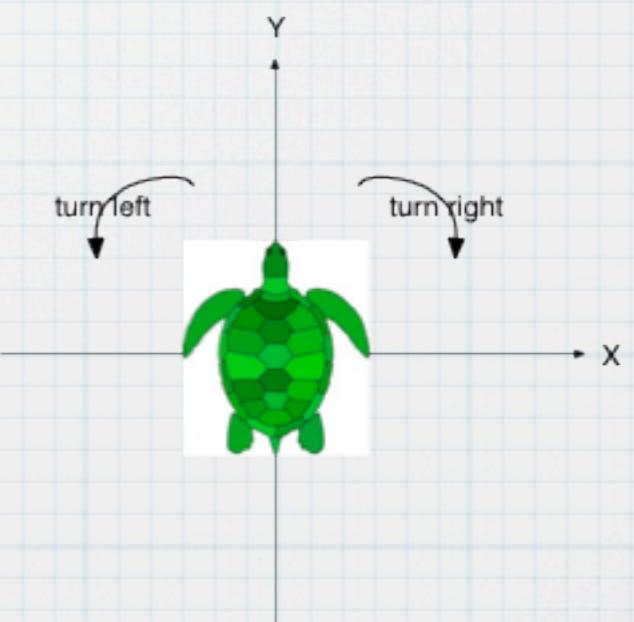
```
action
:   action_pen | action_move | action_turn ;

action_pen
:   ('drop' | 'pull') 'pen' ;

action_move
:   'move' ('forward' | 'backward' | 'left' | 'right')? 'by' num_expr ;

action_turn
:   'turn' ('left' | 'right')? 'by' num_expr ;
```

- \* Controllano posizione e orientamento del cursore e suo stato
- \* Direzione omissibile per forme più compatte
  - \* Move: assume 'forward' come default
  - \* Turn: assume 'right' (clockwise) come default
- \* Stato iniziale del cursore:
  - \* posizione (0, 0) del sistema
  - \* penna abbassata
  - \* orientamento 0° (asse Y positivo)
- \* Angoli espressi in gradi sessagesimali



# Espressioni numeriche

- \* “Solita” grammatica con aggiunta di riferimenti
  - \* variabili
  - \* valori di funzione
  - \* necessario un aumento del lookahead per distinguerli
- \* Costituiscono un livello inferiore rispetto alla grammatica del linguaggio
  - \* AST dedicato
  - \* Visitor dedicato e autonomo
  - \* restituiscono sempre un valore numerico al livello superiore

```
num_expr
:   term (( '+' | '-' ) term)* ;
term
:   factor (( '*' | '/' ) factor)* ;
factor
:   NUMBER | '(' num_expr ')' | '-'? reference ;

reference
options{k=2;}
: ID | function_call ;

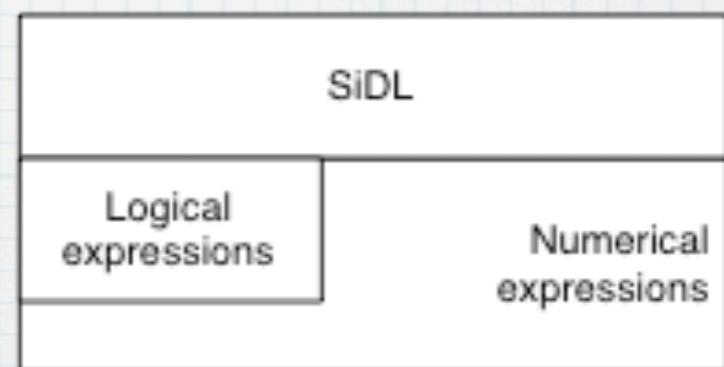
NUMBER : '-'? INT ('.' DIGIT+)?;
fragment INT      : '0' | NZ_DIGIT DIGIT* ;
fragment DIGIT   : '0' | NZ_DIGIT;
fragment NZ_DIGIT : '1'...'9';
```

# Espressioni logiche

```
logic_expr
:   relation ( ('and' | 'or' | 'xor') relation)*
|   'not' relation;

relation
:   num_expr ('<' | '<=' | '=' | '!='
|   '>=' | '>') num_expr
|   '[' logic_expr ']';
```

- \* Applicano i classici operatori logici a relazioni tra espressioni numeriche
- \* Costituiscono anch'esse un livello inferiore rispetto alla grammatica del linguaggio
  - \* Visitor dedicato e autonomo
  - \* Restituiscono un valore booleano
    - \* non possono rappresentare un valore numerico in un assegnamento
    - \* utilizzate unicamente nelle istruzioni di controllo
  - \* Parentesi quadre per esprimere priorità tra relazioni



# Funzioni

```
function
:  'function' ID '(' idlist? ')' block? 'return' num_expr NL ;
function_call
:  ID '(' num_expr_list? ')' ;
num_expr_list
:  num_expr (',' num_expr)* ;
```

- \* Restituiscono sempre e solo un valore numerico
- \* Non alterano lo stato del cursore
  - \* nel corpo di una funzione sono ammessi solo assegnamenti e istruzioni di controllo
  - \* controllo semantico a livello di interprete
- \* Sempre richiamabili (anche ricorsivamente)
- \* Condivisione dati solo via parametri (passati per valore)
- \* Classico modello applicativo

```
function sqr(x) return x*x

function fact(x)
  if x = 0 then
    result = 1
  else
    result = x * fact(x-1)
  endif
  return result

function pi() return 3,1415926
```

# Assegnamenti

```
assignment
  : ID '=' assignment_target ;
assignment_target
  options{k=2;}
  : num_expr | ID '=' assignment_target ;
```

- \* Alterano l'ambiente di esecuzione aggiungendo o modificando variabili
- \* Possibilità di eseguire assegnamenti multipli
- \* Nessun bisogno di dichiarare le variabili prima di utilizzarle
- \* Ovviamente, nessun type check e nessun modificatore

# Istruzioni di controllo

```
control
:   control_if | control_loop | control_fromto ;

control_if
:   'if' logic_expr 'then' block ('elseif' logic_expr 'then' block)*
    ('else' block)? 'endif' ;
control_loop
:   'do' ( condition block 'loop' | block 'loop' condition ) ;
condition
:   ('while' | 'until') logic_expr ;
control_fromto
:   ('with' ID)? 'from' num_expr 'to' num_expr block 'loop' ;
```

- \* Sempre utilizzabili, anche annidate
- \* Tre tipologie
  - \* if tradizionale con eventuali clausole elseif e/o else
  - \* cicli do..loop utilizzabili in quattro modalità differenti
  - \* ciclo from..to per iterazioni su un intervallo di valori
- \* Ciclo from..to particolarmente flessibile
  - \* Può usare o meno una variabile come indice
    - \* la variabile indice resta visibile fuori dal corpo
    - \* indice non modificabile all'interno del corpo
  - \* Stepping automatico
  - \* Valori di intervallo convertiti automaticamente a interi

# Istruzioni di controllo

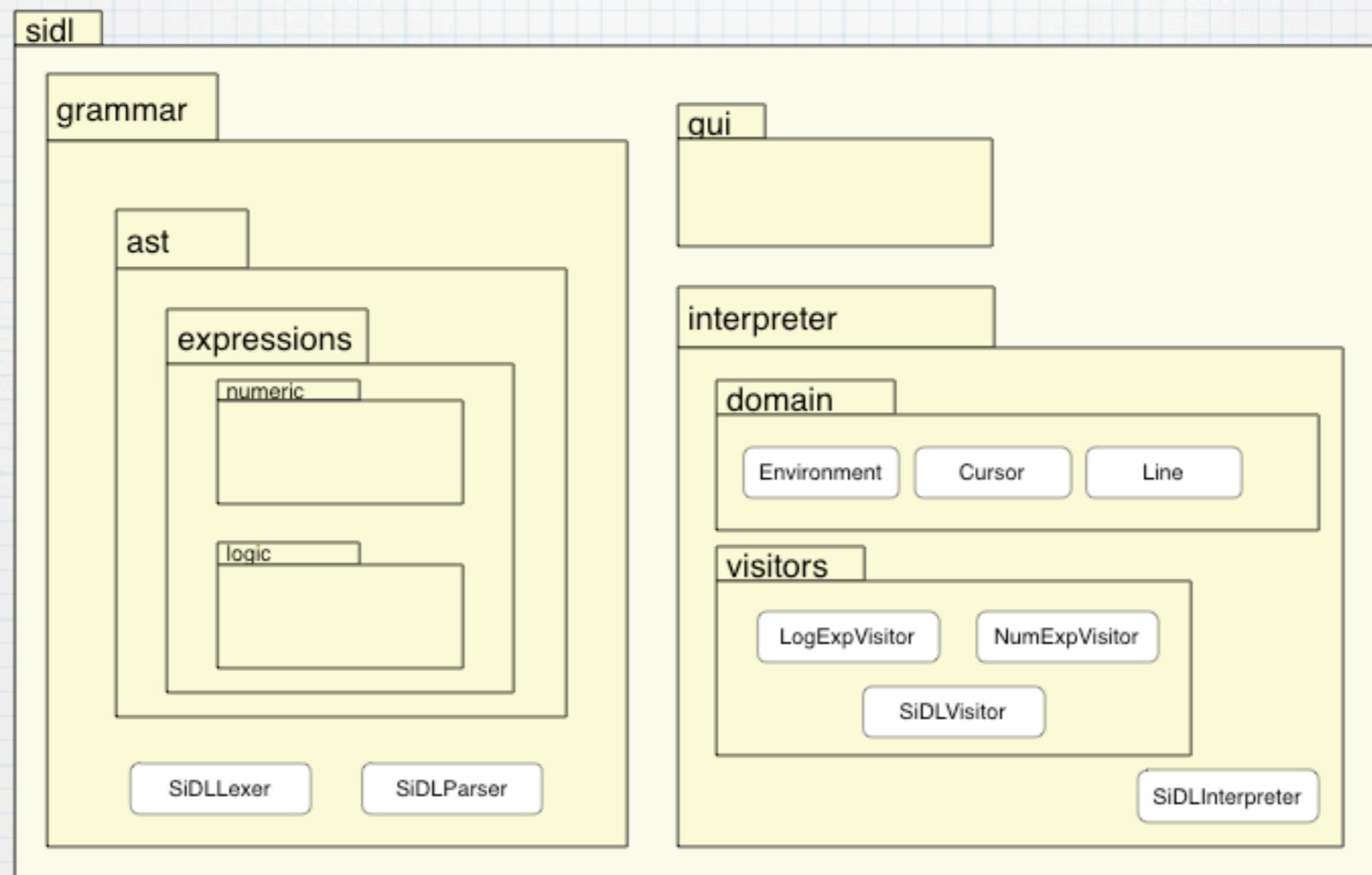
```
control
: control_if | control_loop | control_fromto ;

control_if
: 'if' logic_expr 'then' block ('elseif' logic_expr 'then' block)*
  ('else' block)? 'endif' ;
control_loop
: 'do' ( condition block 'loop' | block 'loop' condition ) ;
condition
: ('while' | 'until') logic_expr ;
control_fromto
: ('with' ID)? 'from' num_expr 'to' num_expr block 'loop' ;
```

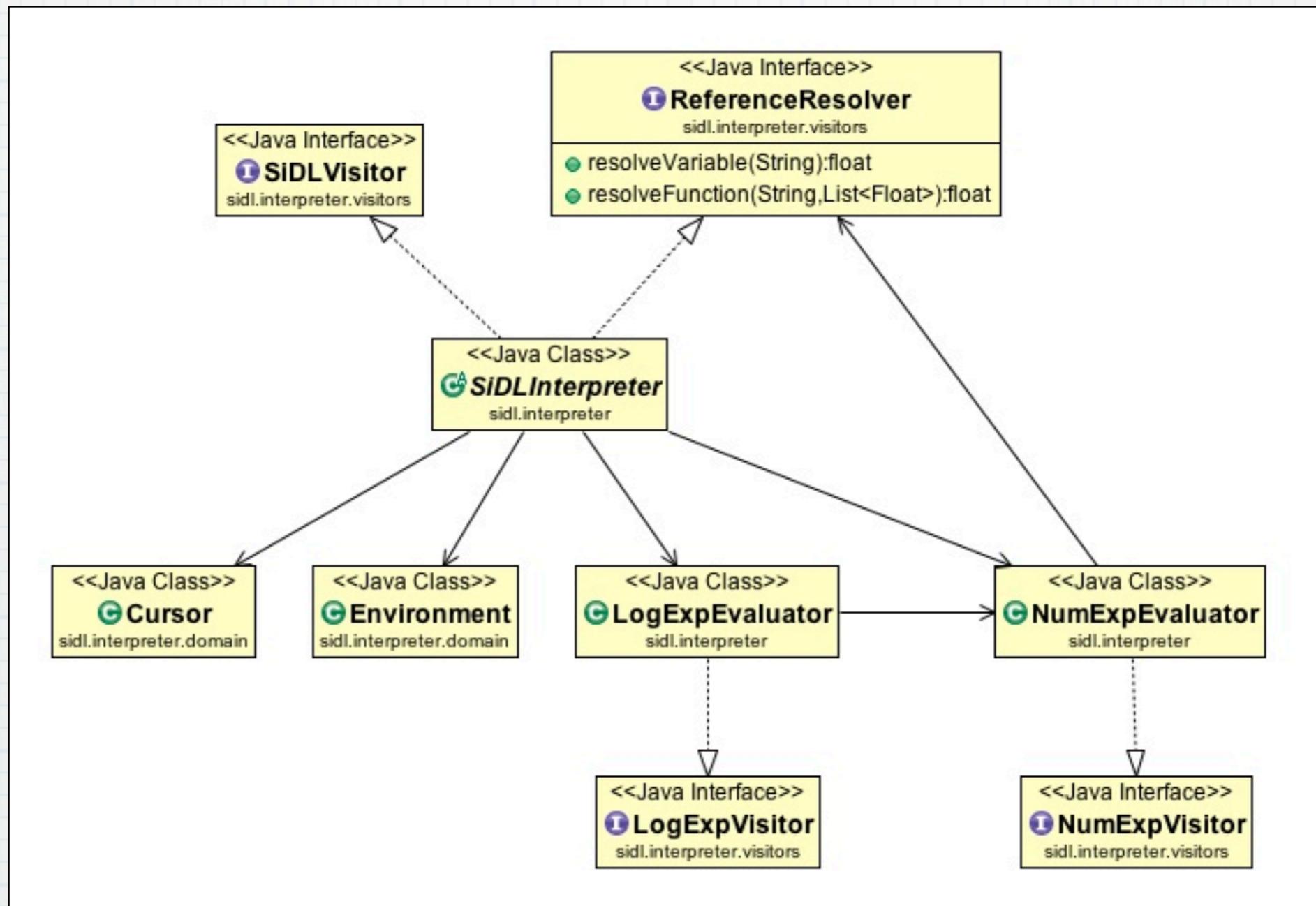
if val < 100 then	do while size > 50	from 1 to 360
call doSomething	move by size	move by 1
elseif val < 50 then	turn by 45	turn by 1
call	size = size-1	loop
doSomethingMore	loop	...
else		with i from 100 to 1
val = 100	do	call drawCircle
endif	turn left by 90	loop
	move by size	
	size = size+10	
	until size >= 150	

# Implementazione

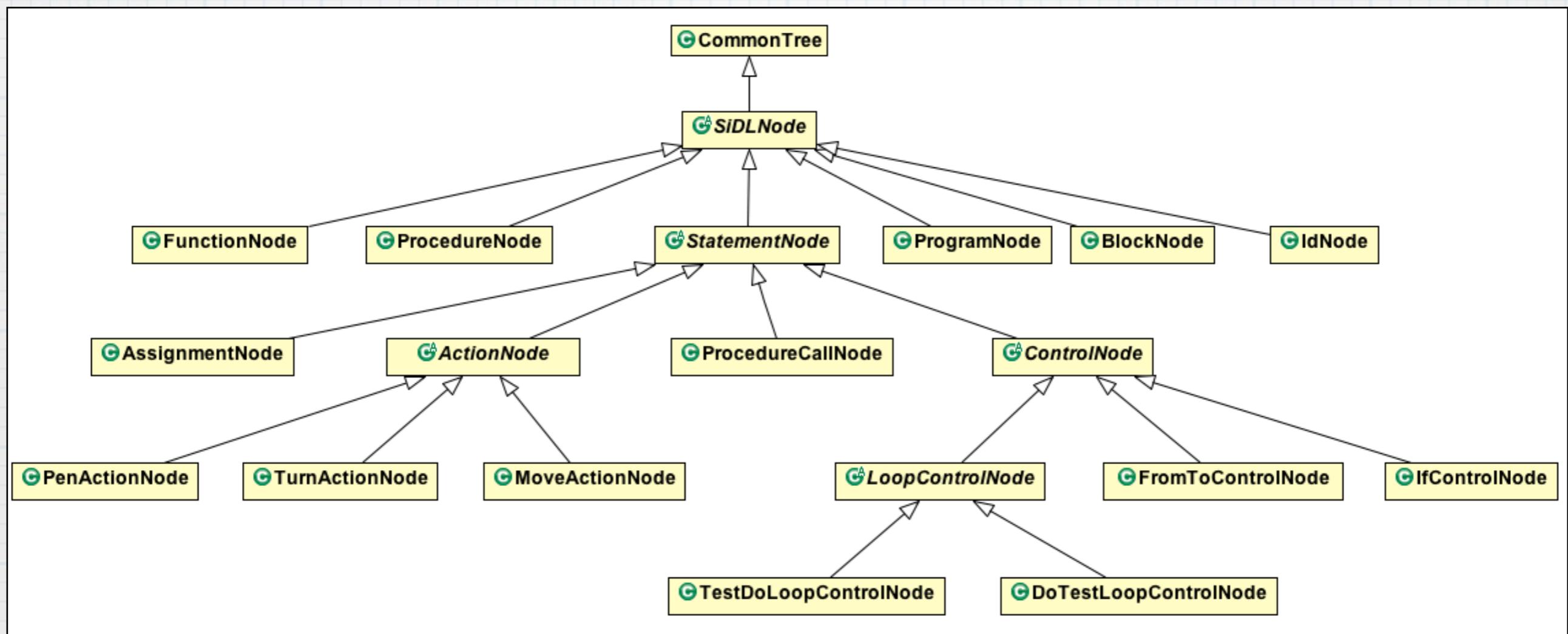
# Struttura complessiva



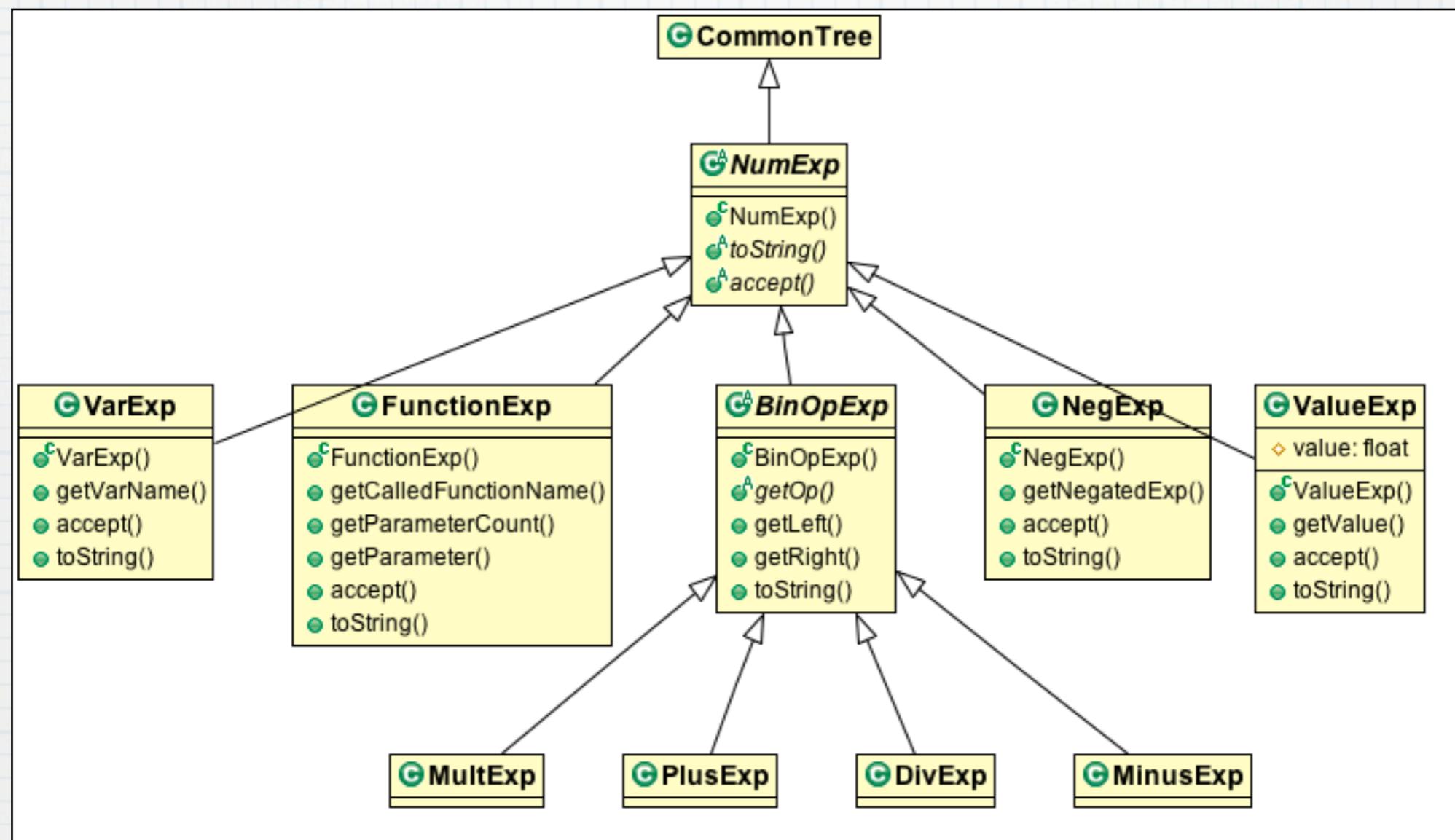
# Interprete



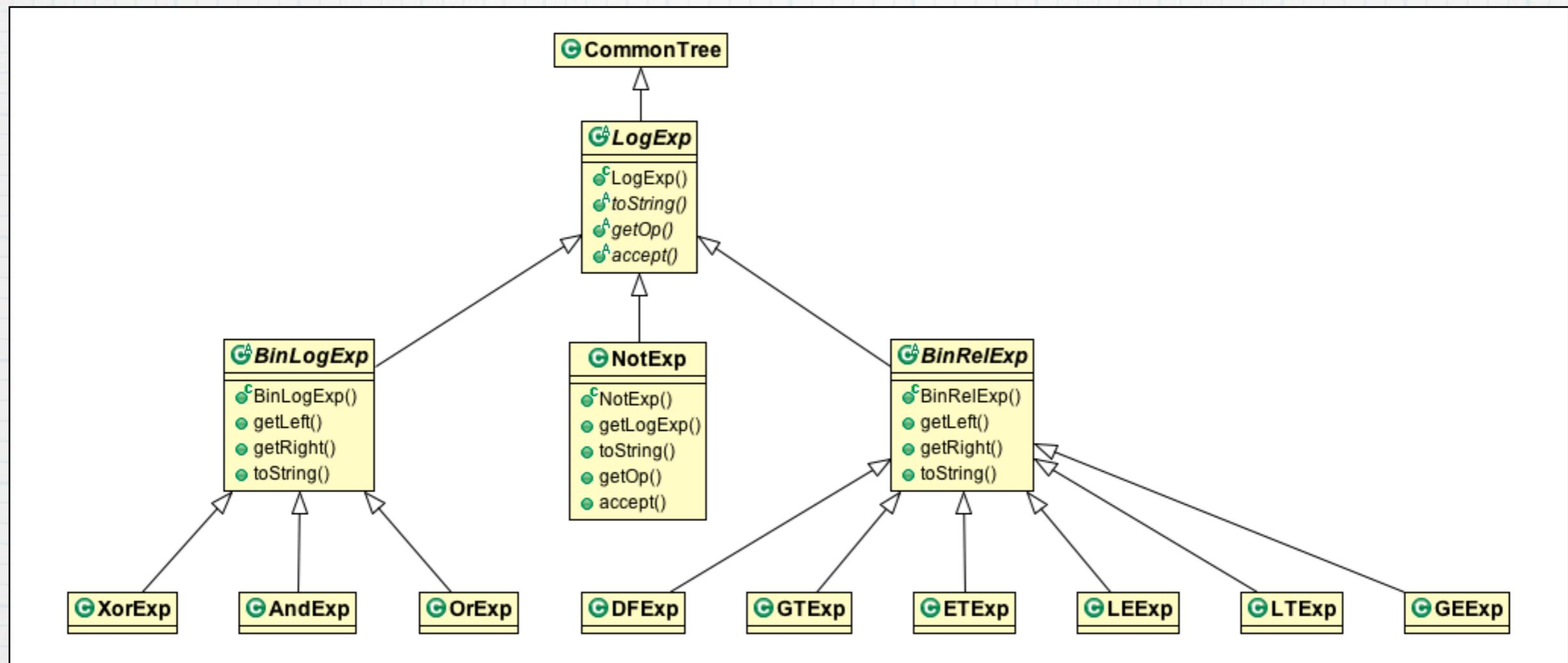
# AST



# AST

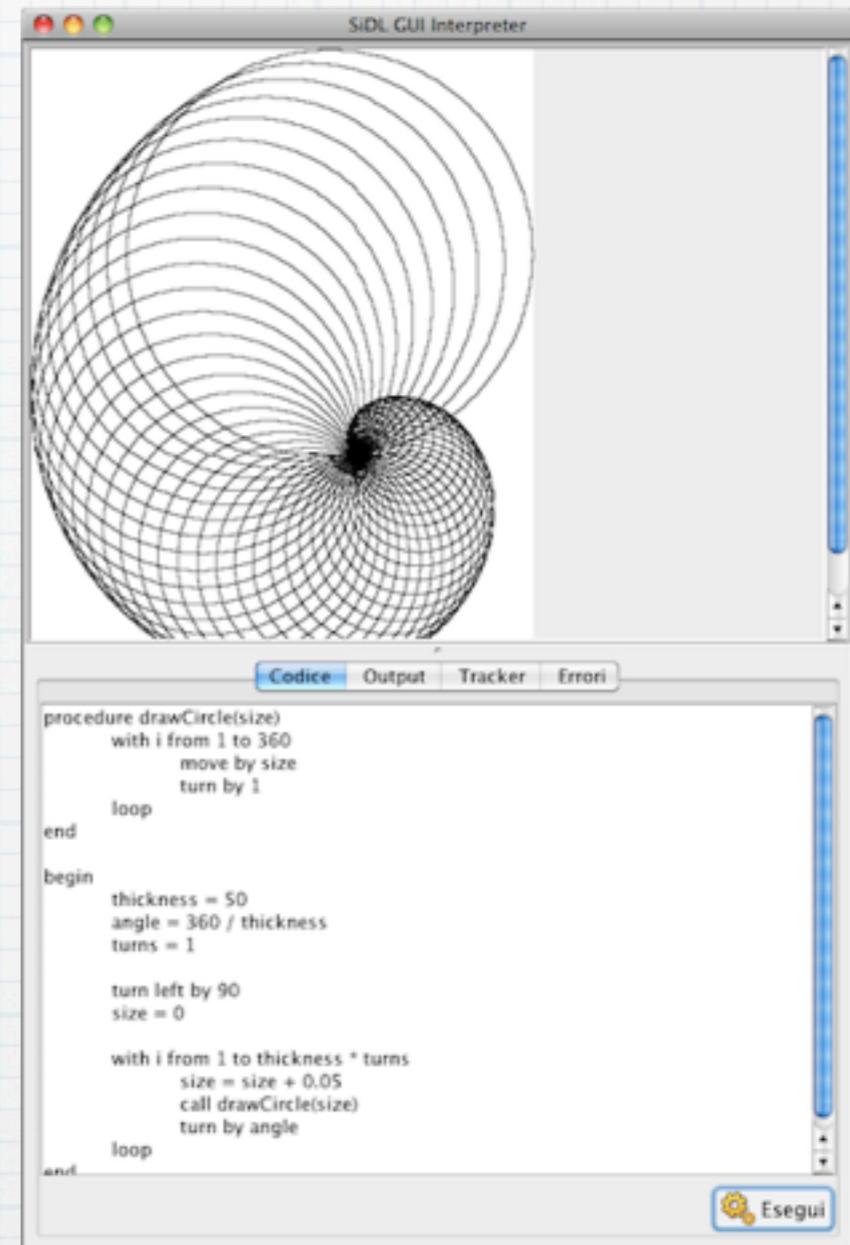


# AST



# Demo

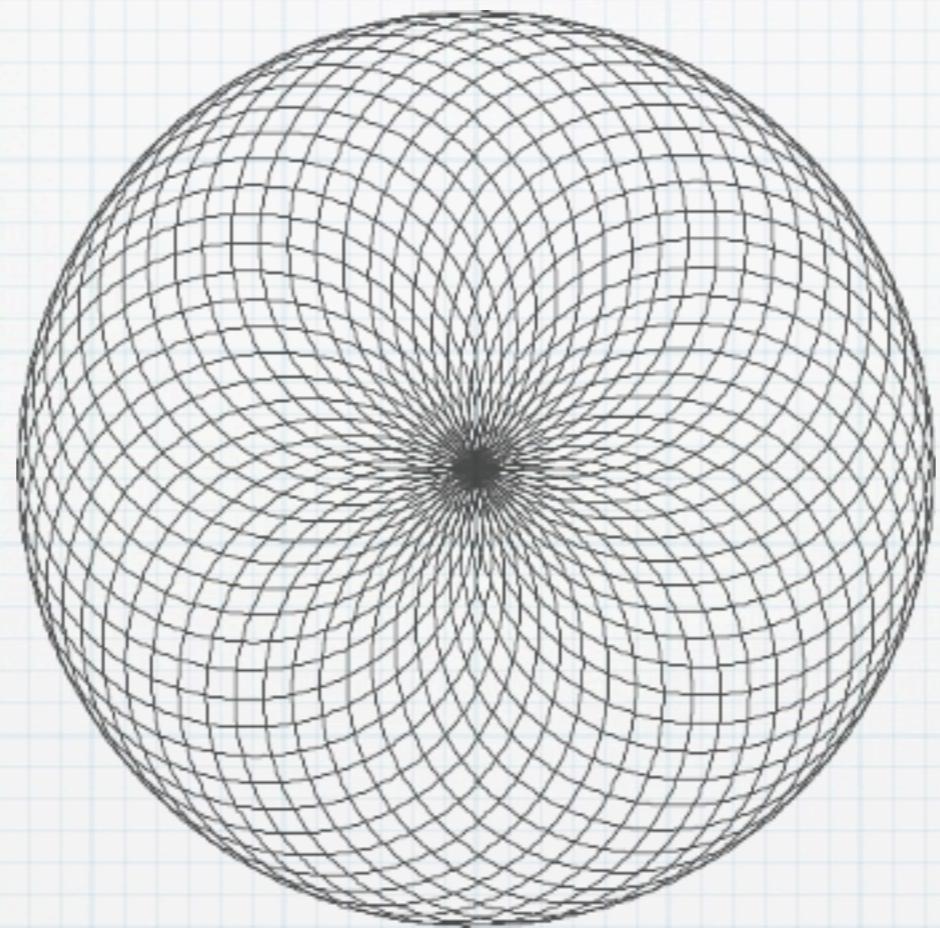
Vengono forniti alcuni sorgenti di esempio nel package `sidl.examples`



```
java -cp SIDL.jar sidl.MainController
```

# Possibili sviluppi

- \* Maggiore flessibilità
  - \* Variabili globali
  - \* Maggiore controllo sul flusso di esecuzione
    - \* break, continue, return, end ...
  - \* Funzione principale parametrizzabile
- \* Funzionalità aggiuntive
  - \* Controllo errori più approfondito
  - \* Overloading di procedure e funzioni
- \* Introduzione di alcuni tipi primitivi
  - \* int, real, cursor, boolean ...
- \* Arrays
- \* Possibilità di esecuzione parallela
  - \* Più cursori disegnano uno stesso output
  - \* SIMD-like oppure fork-like
- \* Aggiunta della terza dimensione
- \* Ambiente di sviluppo più completo



*Alma Mater Studiorum  
Università di Bologna  
A.A. 2011/12*