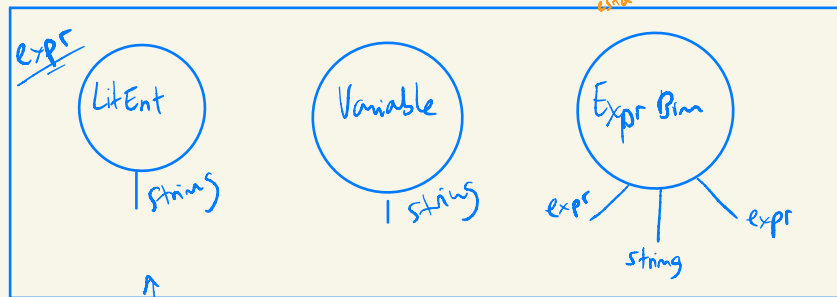
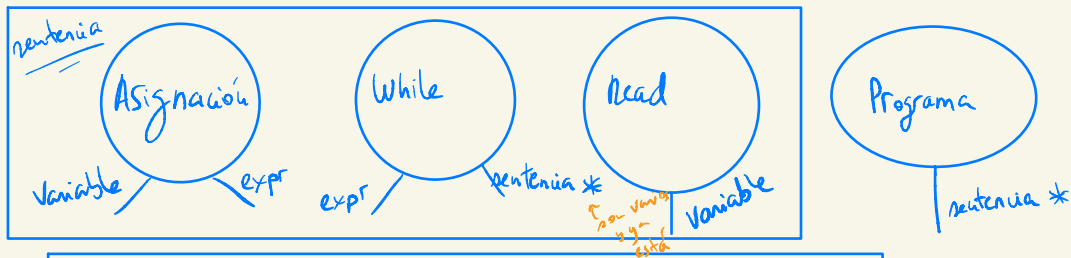


## Ejercicio

```
a = 2
while (a < 100) {
    read b;
    a = a + b;
}
```



CATEGORÍA SINTÁCTICA

mirar  
siguiente  
página

interfaces

sentencia

asigna  
variable / expr

while  
expr / sentencia\*

lectura  
variable  
clases

programa  
|  
sentencia\*

expr

exprBinaria  
expr / string / expr

variable  
|  
string

literalEntero  
|  
string  
clases

**programa** → sentencia\*

**asigna** : sentencia → variable expr

**while** : sentencia → expr sentencia\*

**lectura** : sentencia → variable

**exprBinaria** : expr → left: expr operador: string right: expr

**variable** : expr → nombre: string

**literalEntero** : expr → valor: string

el tipo es obligatorio

el nombre es opcional

Nodo  
Categoría  
Tipo Java

Describir los modos que hay y qué bujes tiene cada uno

- Tipos entero y double.
- Condiciones de tipo entero.
- Las definiciones antes de las sentencias.
- Puede haber varias sentencias en cada rama del if.

```
double a;
```

```
if (a > 2) then  
    print a;  
else  
    print g(5, a) + 8.3;  
endif
```

```
print f(a * 2);
```

programa  $\rightarrow$  definicion \* sentencia \*

definicion  $\rightarrow$  nombre : String tipo

if  $\rightarrow$  sentencia  $\rightarrow$  sentencia\* sentencia\* exp

print  $\rightarrow$  sentencia  $\rightarrow$  expr

variable  $\rightarrow$  expr  $\rightarrow$  valor : String

litreal  $\rightarrow$  expr  $\rightarrow$  valor : String

lit Entero  $\rightarrow$  expr  $\rightarrow$  valor : String

↑  
no está  
bien, mirar  
punteros

```
@parser::header {
  import ast.*;
}
```

← para arreglar  
los errores e importar

```
start returns[Programa ast]
  : definicion print EOF { $ast = new Programa($definicion.ast, $print.ast); }
  ;
```

```
definicion returns[Definicion ast]
  : tipo IDENT ';' { $ast = new Definicion($tipo.ast, $IDENT.text); }
  ;
```

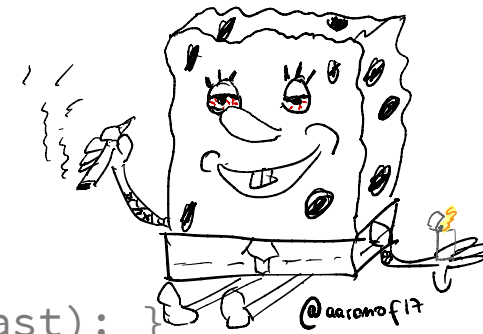
↑ objeto en  
el que se guarda

↑ token    ↑ lexema

```
tipo returns[Tipo ast]
  : 'int' { $ast = new TipoInt(); }
  | 'float' { $ast = new TipoReal(); }
  ;
```

```
print returns[Print ast]
  : 'print' expr ';' { $ast = new Print($expr.ast); }
  ;
```

```
expr returns[Expression ast]
  : IDENT { $ast = new Variable($IDENT.text); }
  | LITENT { $ast = new LiteralEntero($LITENT.text); }
  ;
```



**start**

: sentences EOF  
;

**sentences**

: (sentence \*)  
;

**sentence**

: 'print' expr ';' | left=expr '=' right=expr ';' ;

**expr**

: left=expr op=('\*' | '/') right=expr  
| left=expr op=('+' | '-') right=expr  
| '(' expr ')'  
| IDENT  
;

**program** → sentence\*

**print**: sentence → expr

**assignment**: sentence → left:expr  
right:expr

**arithmetic**: expr → left:expr  
operator:string  
right:expr

**variable**: expr → name:string

@parser :: header {  
import ast.\*

{

**start** returns (Program ast)

: sentences EOF { \$ast = new Program (\$sentences.list); }

;

**sentences** return [List<Sentence> list = new ArrayList<Sentence>]

: (sentence { \$list.add(\$sentence.ast); })\*

;

**sentence** return (Sentence ast)

: 'print' expr ';' { \$ast = new Print(\$expr.ast); }

| left=expr '=' right=expr ';' { \$ast = new Assignment(\$left.ast, \$right.ast); }

;

**expr**

: left=expr op=('\*' | '/') right=expr { \$ast = new Arithmetic(\$left.ast, \$op.text, \$right.ast); }

| left=expr op=('+' | '-') right=expr { \$ast = new Arithmetic(\$left.ast, \$op.text, \$right.ast); }

| '(' expr ')'

| IDENT { \$ast = \$expr.ast; }

;

{ \$ast = new Variable(\$IDENT.text); }