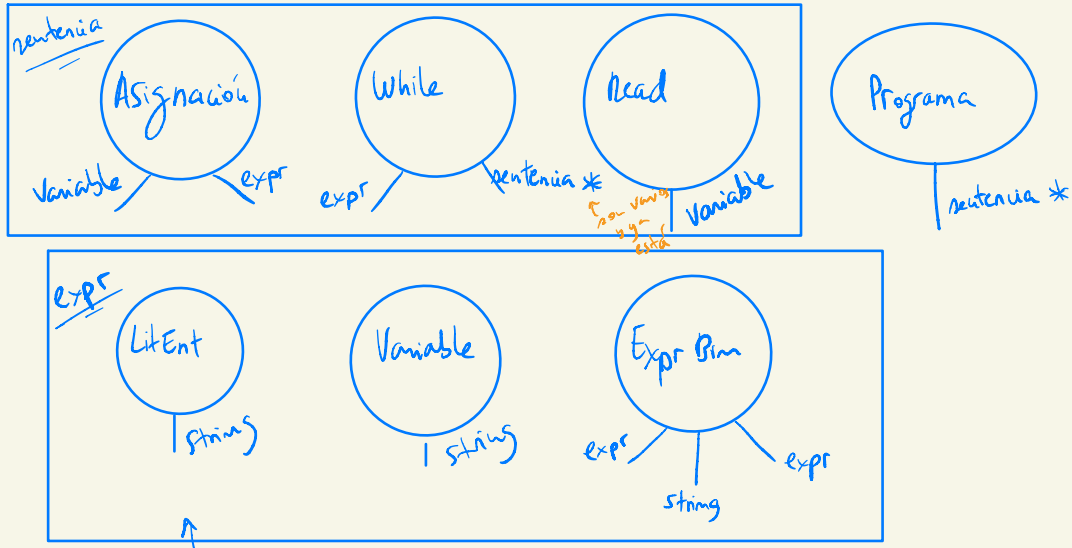
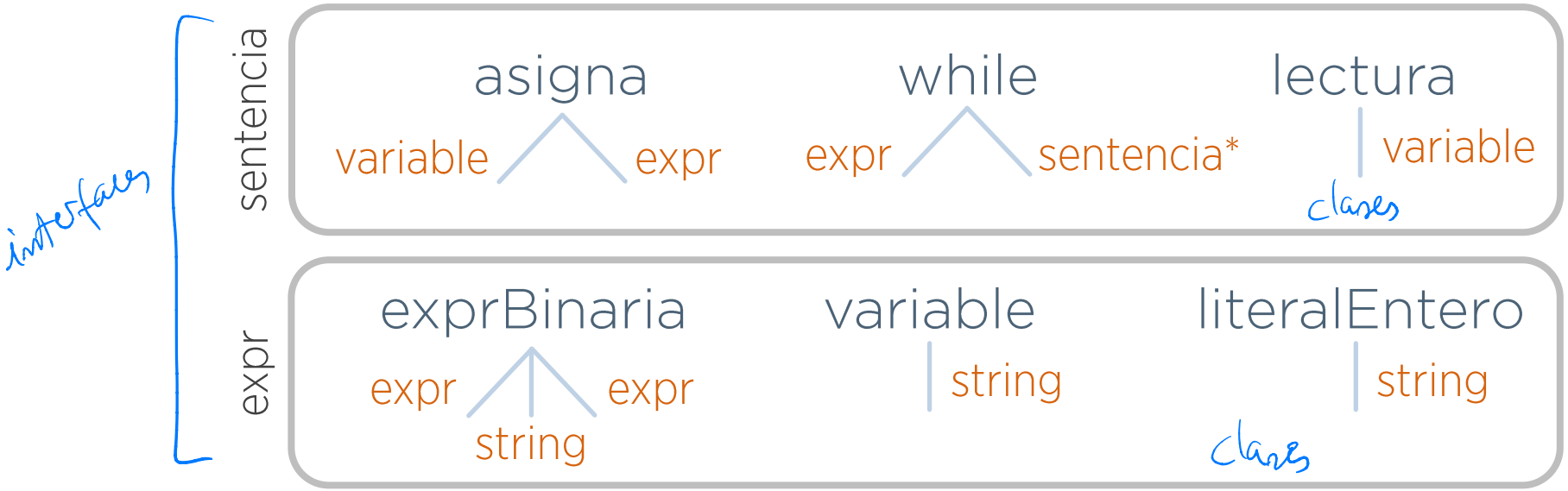
Ejercicio

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



mirar siguiente página



programa
sentencia*

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

Nodo
Categoría
Tipo Java

el tipo es obligatorio

el nombre es opcional

Describir los nodos que hay y qué hijos 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 → definicion * sentencia *

definicion → nombre : String tipo

if → sentencia → sentencia* sentencia* exp

print → sentencia → expr

variable → expr → valor : String

litReal → expr → valor : String

litEntero → expr → valor : String

no está bien, mirad apuntes

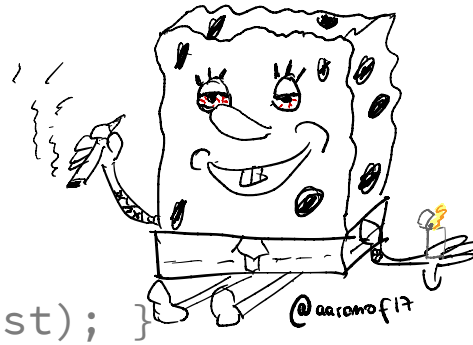
```
@parser::header {
    import ast.*;
}

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); }
    ;

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

para arreglar los errores e importar

objeto en el que se guarda

token

lexema



```
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 returns [List<Sentence> list = new ArrayList<Sentence>]
    : (sentence { $list.add($sentence.ast); } )*
    ;
sentence returns [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
    | '(' expr ')'                          { $ast = $expr.ast; }
    | IDENT                                 { $ast = new Variable($IDENT.text); }
    ;
```