# Live Oak

## Grammar Conventions :

- Symbols with special meaning:
  - `*` : $\geq 0$ occurences
  - `+` : $> 0$ occurences
  - `?` : 1 or 0 occurence
  - `[ ]` : character class construction operator
  - `( )` : parentheses used for grouping
- Anything in red is terminal
- UPPE_CASE symbols are non-terminals.

## Live Oak - 0 : Expressions, Assignment

```
PROGAM  ⟶  BODY
BODY    ⟶  VAR_DECL*  BLOCK
VAR_DECL ⟶  TYPE  ID  ( ,  ID )*  ;
BLOCK   ⟶  { STMT+ }
STMT    ⟶  VAR = EXPR; | ;
EXPR    ⟶  ( EXPR ? EXPR : EXPR )
         | ( EXPR BINOP EXPR )
         | ( UNOP EXPR )
         | ( EXPR ) | VAR | LITERAL
BINOP   ⟶  [ + - * / % & | <> = ]
UNOP    ⟶  [ ~ ! ]
TYPE    ⟶  int | bool | String
VAR     ⟶  IDENTIFIER
LITERAL ⟶  NUM | true | false | STRING
NUM     ⟶  [ 0 - 9 ]+
STRING  ⟶  " [ ASCII character ]* "
IDENTIFIER ⟶ [a-zA-Z] ( [a-zA-Z0-9_] )*
```

## Live Oak - 1 : Imperitive Programming

STMT → if (EXPR) BLOCK else BLOCK
| while (EXPR) BLOCK
| break;
| previous clauses from Live Oak - 0

... same definitions as Live Oak - 0

## Live Oak - 2 : Procedual Programming

PROGRAM → METHOD_DECL* ~~| BODY~~
METHOD_DECL → TYPE METHOD ( FORMALS ? ) { BODY }
BODY → VAR_DECL* BLOCK
FORMALS → TYPE ID ( , TYPE ID )*
ACTUALS → EXPR ( , EXPR )*
VAR_DECL → TYPE ID ( , TYPE ID )* ;
BLOCK → { STMT + }
STMT → return EXPR ;
| previous clauses from Live Oak - 1
EXPR → METHOD ( ACTUALS ? )
| previous clauses from Live Oak - 1
METHOD → IDENTIFIER

... same definitions from Live Oak - 1

## Live Oak - 3 : Classes and Objects

PROGRAM → CLASS_DECL*
CLASS_DECL → class CLASS ( VAR_DECL* ) { METHOD_DECL* }
METHOD_DECL → TYPE METHOD ( FORMALS ? ) { BODY }
EXPR → this | null | new CLASS ( ACTUALS ? )
| CLASS . METHOD ( ACTUALS ? )
~~| METHOD ( ACTUALS ? )~~
| previous clauses from Live Oak - 2
TYPE → void | CLASS | previous clauses from Live Oak - 2
CLASS → IDENTIFIER

... same definitions from Live Oak - 2

## LiveOak: the final definitions

```
PROGRAM      → CLASS_DECL*
CLASS_DECL   → class CLASS ( VAR_DECL* ) { METHOD_DECL* }
METHOD_DECL  → TYPE METHOD ( FORMALS? ) { BODY }
METHOD       → IDENTIFIER
BODY         → VAR_DECL* BLOCK
FORMALS      → TYPE ID ( , TYPE ID )*
ACTUALS      → EXPR ( , EXPR )*
VAR_DECL     → TYPE ID ( , TYPE ID )* ;
BLOCK        → { STMT+ }
STMT         → return EXPR ;
             | if (EXPR) BLOCK else BLOCK
             | while (EXPR) BLOCK
             | break;
             | VAR = EXPR; | ;
EXPR         → this | null | new CLASS (ACTUALS?)
             | CLASS . METHOD (ACTUALS?)
             | ( EXPR ? EXPR : EXPR)
             | ( EXPR BINOP EXPR)
             | ( UNOP EXPR)
             | ( EXPR) | VAR | LITERAL
VAR          → IDENTIFIER
BINOP        → [ + - * / % & | <> = ]
UNOP         → [ ~ ! ]
TYPE         → void | CLASS | int | bool | String
CLASS        → IDENTIFIER
LITERAL      → NUM | true | false | STRING
NUM          → [ 0-9 ]+
STRING       → " [ ASCII character ]* "
IDENTIFIER   → [a-zA-Z] ( [a-zA-Z0-9_] )*
```