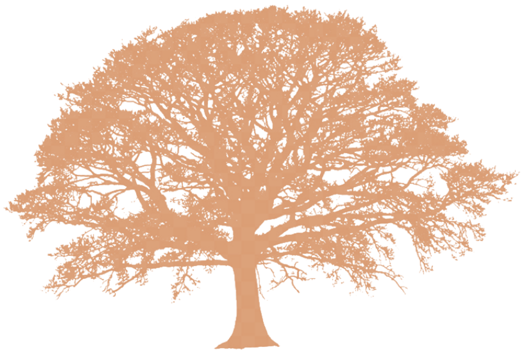
The Source Language



- LiveOak
    - A high-level language for pedagogical use.
    - Derived from Bali.
    - The name is a homage to Java® and to Texas.
- LiveOak is statically typed, strict, imperative.
- Layered into several tiers of functionality.
    - LiveOak-0: Expressions, assignment statements, sequential control flow.
    - LiveOak-1: Imperative programming with structured control flow,
    - LiveOak-2: Procedural programming.
    - LiveOak-3: Objects and classes, no inheritance.

Grammar Conventions

- Symbols in red are terminals.
  - Lower-case symbols denote literal values (e.g., `int`) that are reserved keywords and cannot be used as identifiers for variables, methods, or classes.
  - Non-alphanumeric characters (e.g., `{`) denote literals consisting of only the non-alphanumeric characters. These are typically operators or grouping constructs.

- UPPER-CASE symbols are non-terminals.

- Several grammar symbols (in blue) have special meaning:
  - `*` means zero or more occurrences.
  - `+` means one or more occurrences.
  - `?` means one or zero occurrences.
  - `[ ]` is the character class construction operator.
  - `( )` are parentheses used for grouping.

LiveOak-0: Expressions, Assignment

```
PROGRAM → BODY
BODY → VAR_DECL* BLOCK
VAR_DECL → TYPE ID (, ID)* ;
BLOCK → { STMT+ }
STMT → VAR = EXP ; | ;
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_])*
```

LiveOak-1: Imperative Programming

```
STMT → if ( EXPR ) BLOCK else BLOCK
     | while ( EXPR ) BLOCK
     | break ;
     | previous clauses
```

`PROGRAM`, `BODY`, `VAR_DECL`, `BLOCK`, `EXPR`, `BINOP`, `UNOP`, `TYPE`, `VAR`, `LITERAL`, `NUM`, `STRING`, `IDENTIFIER` remain unchanged from LiveOak-0.

LiveOak-2: Procedural 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 (, ID)* ;
BLOCK → { STMT+ }
STMT → return EXPR ;
     | previous clauses
EXPR → METHOD ( ACTUALS? )
     | previous clauses
METHOD → IDENTIFIER
```

BINOP, UNOP, TYPE, VAR, LITERAL, NUM, STRING, IDENTIFIER remain unchanged from LiveOak-1.

LiveOak-3: Classes and Objects

PROGRAM → CLASS_DECL* ~~| (METHOD_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*

TYPE → void | CLASS | *previous clauses*

CLASS → IDENTIFIER

BODY, FORMALS, ACTUALS, VAR_DECL, BLOCK, STMT, BINOP, UNOP, VAR, LITERAL, METHOD, NUM, STRING, IDENTIFIER remain unchanged from LiveOak-2.