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.

LiveOak-3 Program Example

```
Class treeCell
  (int v; treeCell left, right;)
{
  treeCell(int i, treeCell l, treeCell r) {v = i; left = l; right = r;}
  treeCell getLeft() {return left;}
  treeCell getRight() {return right;}
  int getObject() {return v;}
}
```

LiveOak-3 Program Example

```
Class treeCell
  (int v; treeCell left, right;)
{
  treeCell(int i, treeCell l, treeCell r) {v = i; left = l; right = r;}
  treeCell getLeft() {return left;}
  treeCell getRight() {return right;}
  int getObject() {return v;}
}
```

```
class Main ()
{
  int main() {
    treeCell t;
    t = new treeCell(13,
                     new treeCell(7, null, null),
                     new treeCell(9, null, null));
    return Walk(t,4000);
  }
  int Walk(treeCell t, int minValue) {
    int nodeValue;
    if ((t = null)) {return minValue;}
    else {
      nodeValue = t.getObject();
      minValue = Walk(t.getLeft(), minValue);
      if ((nodeValue < minValue))
        {return Walk(t.getRight(), nodeValue);}
      else {return Walk(t.getRight(), minValue);}
    }
  }
}
```

Language-Level Transformations

- Translate the definition of an instance method

```
retType f(type1 v1, ...)
```

in class C as though it was written as

```
retType C$f(C this, type1 v1, ...)
```

Language-Level Transformations

- Translate the definition of an instance method

```
retType f(type1 v1, ...)
```

in class C as though it was written as

```
retType C$f(C this, type1 v1, ...)
```

- If name `r` is of type `C`, translate the instance method invocation

```
r.m(e1, e2, ...)
```

as though it was written as

```
C$m(r, e1, e2, ...)
```

Language-Level Transformations

- Translate the definition of an instance method

```
retType f(type1 v1, ...)
```

  in class `C` as though it was written as

```
retType C$f(C this, type1 v1, ...)
```

- If name `r` is of type `C`, translate the instance method invocation

```
r.m(e1, e2, ...)
```

  as though it was written as

```
C$m(r, e1, e2, ...)
```

- Handle object constructors with a wrapper.
  - In the wrapper, allocate storage for the object record from the heap.
  - Pass a reference to this area as the first parameter of the constructor call (which is just a normal method call).
  - There is no `return` statement in the constructor, so the wrapper has to handle returning the object reference to the constructor's caller.