# BONUS! Simple Objects CS345H Due 10am Tuesday Dec 6, 2011

Extend your previous language to define a simple object model. The objects support dynamic dispatch, but not inheritance or private members.

Modify the parser and interpreter framework that was implemented in the previous assignments. Call the new language SOOL (simple object-oriented language).

- Modify your grammar to implement your parser for your language.
- *If possible, implement the new features with as few changes as possible to the code you turned in for your previous assignment.* This will highlight the potential extensibility of java libraries. If you must change existing code, then you must submit a list of changes and the reason for the change. Discuss whether the original code could have been written in a way that would allow the new features to be defined without changing existing code.

## New Features to Implement

* Assignment statements. The ability to change the value of a local variable. The binding of that variable is changed and the change is visible in all environments that refer to the binding.
* Object creation. Objects are created by capturing the current environment as a value. The environment is the collection of bindings active at the point where the object is created. There is no protection (public, private) for identifiers in the environment. All identifiers in the environment are accessible.
* Object access. The only way to use an object is to lookup identifiers within the object, as in "`obj.id`". The value accessed may be a function, so that the form "`obj.m(args)`" is legal. It calls the function named "m" that is defined in the environment value bound to "obj".
* Sequences of statements. The value of the sequence is the value of the last expression in the sequence.

## Syntax of SOOL

The syntax of the JCFWAER language with these additional features can be captured with the following EBNF. The terminal symbols are marked in quotes. The quotes are not part of the language.

```
E ::= num | "true"  | "false"
    | id
*   | "makeobj"                                  environment capture
*   | E "." id                                   field access
    | E "(" E "," … "," E ")"
    | E "*" E      | E "/" E
    | E "+" E      | E "-" E
    | E "=" E      | E "<" E
    | "!" E
    | E "&&" E
    | E "||" E
*   | id ":=" E                                  assignment
    | "if" E "then" E "else" E
    | "with" id "=" E ";" … ";" id "=" E "in" E
    | "fun" "(" id "," … "," id ")" E
*   | "{" E ";" … ";" E "}"                      sequence of expressions
    | "(" E ")"
```

Lines marked with * are new. All other parts of the grammar are unchanged. In this grammar, the ellipsis (...) appears between an optional separator and a nonterminal, means that the nonterminal can be repeated zero or more times with the separator between each occurrence. If there is no separator, then the nonterminal must occur at least once.

The productions in the grammar are written from *highest precedence* to *lowest.*

**Example of SOOL**

You must write test classes that implement object-oriented sets.

```
with
  Empty = with
    member = fun(n) false;
    empty = true
    in makeobj;
  Insert = fun(x, s) with
    member = fun(n) if  n=x then true else s.member(n);
    empty = false
    in makeobj
in
with s = Insert(3, Empty) in {
  s := Insert(6, s);
  s.member(3)
}
```

Note: If you score higher on this assignment than your lowest assignment grade, then this assignment grade will be used in place of your lowest grade.