1. File hw8_1.java contains an interpreter for an object-oriented, untyped variant of the “toy language.” A program now consists of a set of class definitions followed by an expression to be evaluated (corresponding to the “main program.”) Class definitions consist of a class name, superclass name, member field and function definitions. Expressions are much as before, except that there is no letfun construct (the only functions are class member functions), the format of applications has changed to specify both an object and a member function name from that object to be applied, and there is a new constructor to create new objects of a specified class. Here are the most important syntax changes:

```
prog := classdefs exp
classdef := 'class' id '<' id members
classdefs := <empty>
    | classdef classdefs
member := 'field' id '=' exp
    | 'fun' id '(' ids ')' '=' exp
members := <empty>
    | member members
exp := ...
    | 'new' id
factor := ...
    | 'this'
    | factor '.' id '(' expsc ')'
```

The semantics of the language are similar to those of conventional object-oriented languages. Classes are arranged in a hierarchy, with an (empty) built-in class Object at the root. Objects are created (via new) as members of a named class. A newly created object has all the members of that class, plus the members of all its ancestors.

The initial value of each field is specified by an expression in the field definition. Fields can contain integers, booleans, or other objects. Fields of a object are only accessible for reading and writing within functions defined within the class of that object or its subclasses. (Actually, they are also accessible from its superclasses, but that is really a kind of bug in my implementation; if this were a typed language, such accesses could be prohibited statically.)

All function applications are “virtual” (in C++ terminology); i.e., when a function is applied, its name is searched for first in the receiving class, then in its superclass, and so on. Within any function body, the keyword this always refers to the receiving object of the function call.

As usual, further details of the semantics may be revealed by experimenting with the interpreter and reading its source code!
Your assignment is to extend the interpreter in two ways. Combine your modifications into a single file sol8_1.java and include it in your bundle.

(a) Add a new boolean-valued operator \texttt{instanceof} to determine if a given object is an instance of a class (or one of its subclasses). The grammatical extension is:

\begin{verbatim}
cexp := ...  
  | aexp 'instanceof' id
\end{verbatim}

The expression \( e \texttt{ instanceof } c \) should return true if and only if expression \( e \) evaluates to an object which was created by \texttt{new} \( c' \) where \( c' \) is \( c \) or a subclass of \( c \). (This is essentially the same definition as for the \texttt{instanceof} operator in Java.)

(b) Enhance the function application mechanism to permit optional specification of a “starting classname” to use when searching for the function in the class hierarchy. Here’s the syntactic change:

\begin{verbatim}
factor := ...  
  | factor '.' id '(' expsc ')'  
  | factor '.' '[' id ']' id '(' expsc ')'  
\end{verbatim}

The first form represents normal “virtual” function application, with search for the function beginning in the receiving object as usual. In the second form, the \[id\] following the dot must be the name of a superclass of the receiving object’s class (or the name of that class itself); search for the function starts at the named class rather than at the receiving class. Here’s an example:

\begin{verbatim}
class F < Object  
  fun h() = 0

class G < F  
  fun h() = 1

let g = new G  
in let a = g.h()  
in let b = g.[F]h()  
in a + b
\end{verbatim}

Here the first call to \texttt{h} will invoke the version of \texttt{h} in \texttt{G}, the class of the receiving object, so \texttt{a} will be bound to 1. The second call to \texttt{h} will invoke the version of \texttt{h} in the specified superclass \texttt{F}, so \texttt{b} will be bound to 0.

Hint: The easiest way to code this solution is to add an optional class name field to \texttt{AppExp}.

2. Do problem 10.11 from Scott. Type your solution into a file hw8_2.txt and include it in your bundle.