CS 496: HW6 - SOOL+
Due: 30 April 2023, 11:59pm

1 Introducing SOOL+

This assignment asks you to extend SOOL with two simple (unrelated) features:

1. an `instanceOf?(e,id)` predicate; and
2. support for method overloading.

This extended language will be referred to as SOOL+.

2 `InstanceOf?` Predicate

A program in SOOL+ has the same concrete syntax as that of SOOL except that a new expression is supported (the last grammar production below):

⟨Expression⟩ ::= `new` ⟨Identifier⟩`(`⟨Expression⟩$^{*(,)}$`)`
⟨Expression⟩ ::= `self`
⟨Expression⟩ ::= `send` ⟨Expression⟩⟨Identifier⟩`(`⟨Expression⟩$^{*(,)}$`)`
⟨Expression⟩ ::= `super` ⟨Identifier⟩`(`⟨Expression⟩$^{*(,)}$`)`
⟨Expression⟩ ::= `instanceOf?(`⟨Expression⟩, ⟨Identifier⟩`)`

The expression `instanceOf?(e,id)` returns true if the object resulting from evaluated `e` is an instance of class `id` or one of its subclasses. Consider the following example (`ex5.sool`):

```
(* class declarations *)

class c extends object {
}

class d extends c {
}

class e extends c {
}

(* main expression *)
let o=new d()
```

```
in instanceof?(o,d)
```

Listing 1: Example program in SOOL+

When we evaluate it we obtain the following result:

```
# interpf "ex5.sool";;
- : exp_val Sool.ReM.result = Ok (BoolVal true)
```

utop

The result is true since `new d()` evaluates to the object `ObjectVal ("d", EmptyEnv)` whose class is indeed `d`. If we change line 14 in Listing 1 to `instanceof?(o,c)`, then we still get `Ok (BoolVal true)`. However, if we change it to `instanceof?(o,e)`, then we get `Ok (BoolVal false)`. Finally, if we change line 14 in Listing 1 to `instanceof?(o,a)`, where `a` is a class that does not exist, then we should get an error:

```
- : exp_val Sool.ReM.result = Error "is_subclass: class a not found"
```

utop

Extend the interpreter to deal with this new feature:

```
let rec eval_expr : expr -> exp_val ea_result =
  fun e ->
  match e with
  ...
  | IsInstanceOf(e,id) ->
    failwith "implement"
```

interp.ml

You will need to implement a helper function

```
is_subclass:string -> string -> class_env -> exp_val ea_result
```

such that `subclass id1 id2 class_env` (whose type is `exp_val ea_result`) given an environment, ignores it and returns true or false depending on whether `id1` is a subclass of `id2`. You may place the helper function in the file `interp.ml` itself. Here are some sample tests for `is_subclass`:

```
# interpf "ex5.sool";;
- : exp_val Sool.ReM.result = Ok (ObjectVal ("d", EmptyEnv))
utop # is_subclass "e" "c" !g_class_env EmptyEnv;;
- : exp_val Sool.ReM.result = Ok (BoolVal true)
utop # is_subclass "d" "e" !g_class_env EmptyEnv;;
- : exp_val Sool.ReM.result = Ok (BoolVal false)
```

utop

3 Adding Support for Overloading

Consider the expression presented in Listing 2. Evaluation of this program in SOOL will result in an error. The reason is that the `add` method in line 28 expects one argument but the method call in line 49, namely `send o add(2,3)`, is providing two. Line 49 is presumably referring to the method `add` declared in line 34 which does expect two arguments.

```
(* Example 4 *)

(* counterc *)
class counterc extends object {
  field c
  method initialize() { set c=7 }
  method add(i) { set c=c+i }
  method bump() { send self add(1) }
  method read() { c }
}

(* reset counter *)
class resetc extends counterc {
  field v
  method reset() { set c=v }
  method setReset(i) { set v=i }
}

(* backup counter *)
class bkpcc extends resetc {
  field b
  method initialize() {
    begin
      super initialize();
      set b=12
    end
  }
  method add(i) {
    begin
      send self backup();
      super add(i)
    end
  }
  method add(i,j) {
    begin
      send self backup();
      super add(i);
      super add(j)
    end
  }
  method backup() { set b=c }
  method restore() { set c=b}
 }

(* main expression *)
let o = new bkpcc  ()
in begin
     send o restore();
     send o add(2,3);
```

```
50      send o read()
    end
```

Listing 2: Example program in SOOL+

```
# interpf "ex4.sool";;
Sool.ReM.Error "add: args and params have different lengths"
```

utop

Your task is to fix this by providing support for overloading. One easy way to achieve this is to silently change the name of methods (both in their declarations and when they are called) so that the number of parameters is included as a prefix. This technique is called *name mangling*. The method declaration in line 28 of Listing 2 stores `1_add` as the name of the method rather than `add`. Likewise, the method declaration in line 34 of Listing 2 stores `2_add` as the name of the method rather than `add`. The method call in line 49 of Listing 2 is processed as a call to `2_add` rather than `add`.

Include support for overloading using name mangling. You must use the following helper function already included in the stub:

```
let name_mangle n es =
  n^"_"^string_of_int (List.length es)
```

The task is simple: just place a call to `name_mangle` in all the right places so that Listing 2 runs correctly and produces the following result:

```
# interpf "ex4.sool";;
- : exp_val Sool.ReM.result = Ok (NumVal 17)
```

utop

4 Submission Instructions

Submit a file `HW6.zip` with the entire stub with the missing parts completed.