## APPENDIX: A Smalltalk Simulation of the Interpreter

```
Class new title: 'Context';
  fields: 'sender method pc tempframe stackptr mclass receiver';
  asFollows
```

*Contexts carry the dynamic state of Smalltalk processes. They are accessed in efficient ways by the microcode interpreter. At the same time, they are instances of a perfectly normal Smalltalk class. In this way, the full generality of Smalltalk can be applied to examining and tracing the progress of Smalltalk execution.*

*The code below differs from the actual code in Smalltalk-76 in that it corresponds to the slightly simplified categories of the text, and has not been carefully checked for off-by-1 errors.*

*Beyond the specifics in the text, the interested reader will want to know:*

- *"˙" is the subscript message, as in: tempframe˙lobits*
- *except for assignment, "←" is treated as an agglutinating message part, as in: t˙i ← self pop*
- *the ⇒ symbol indicates conditional execution; if the preceding value is true, then the following body of code is executed, and control exits the outer (!) brackets. This "if-only" form serves to build dispatch tables as in the message "next" below*
- *the default value returned from any message is "self", the receiver of the message. Other values may be returned with the "⇑" symbol.*

*The messages "instfield: n" and "instfield: n← val", which are used below to read and write the n-th field of an instance, clearly violate the principle of modularity. This reflects that the buck stops here, and these primitive messages appear nowhere else in the system.*

### Access to Fields

```
sender: sender method: method pc: pc tempframe: tempframe stackptr: stackptr
   mclass: mclass receiver: receiver    "initializes all fields"
```

### Simulation of the Interpreter

```
step | byte lobits                            "dispatch on next code syllable"
   [byte ← self nextbyte.
   lobits ← byte|16.
   byte/16=1⇒[self push: receiver instfield: lobits];        "load from instance"
         =2⇒[self push: tempframe˙lobits];                   "load from temps (and args)"
         =3⇒[self push: (method literals: lobits)];          "load from literals"
         =4⇒[self push: (method literals: lobits) value];    "load indirectly from literals"
         =5⇒[self push: self instfield: lobits];             "load from this Context"
         =6⇒[self push: ☞(˜1 0 1 2 10 true false nil)˙lobits]; "frequent constants"
         =7⇒[⇑self send: (method literal: lobits)];
         =8⇒[⇑self send: (SpecialMessages˙lobits)];          "frequent messages"
         =9⇒[lobits<8⇒[pc← pc+lobits]                        "short jump forward"
               self pop⇒[]  pc← pc+lobits-8];                "short branch if false and pop"
         =10⇒[lobits<8⇒[pc← lobits-3*256+self nextbyte+pc]"long jump forward and back"
               self pop⇒[pc← pc+1];                          "skip extension byte on true"
               pc← lobits-11*256+self nextbyte+pc];          "long bfp"
         =11⇒[lobits=0⇒[self pop];                           "pop stack"
               =1⇒[self store: self top into: self nextbyte]; "store"
               =2⇒[self store: self pop into: self nextbyte]; "store and pop"
               =3⇒[sender push: self top.  ⇑sender]]         "return value to sender"
   ]
store: val into: field | lobits               "same encoding as above"
   [lobits ← byte|16.
   field/16=1⇒[receiver instfield: lobits ← val];      "store into instance"
         =2⇒[tempframe˙lobits ← val];                  "store into temps (and args)"
         =3⇒[user notify: 'invalid store'];            "can't store into literals"
         =4⇒[(method literals: lobits) value ← val];"store indirectly through literals"
         =5⇒[self instfield: lobits ← val]             "store into this Context"
   ]
send: message | class meth callee t i      "send a message"
   [class ← self top class.
   until⁰ (meth← class lookup: message) do⁰ "look up the method"
```

```
        [class← class superclass.                 "follow the superclass chain if necess"
        class=nil⇒[user notify: 'Unrecognized message: '+message]]
    [meth primitive⇒                              "If flagged as primitive, then do it"
        [self doprimitive: meth⇒[⇑self]]]. "If it fails, proceed with send"
    callee← Context new                           "create new Context, and fill its fields"
        sender: self method: meth pc: meth startpc
        tempframe: (t← Vector new: meth tframesize) stackptr: meth startstack
        mclass: class receiver: self pop.
    for° i to: meth nargs do°                     "pass arguments"
        [t•i← self pop]
    ⇑callee]                                      "return new Context, so it becomes current"
nextbyte          "step pc and return next code syllable"
    [⇑method•(pc← pc+1)]
```

## Stack-related Messages

```
push: val         "push value onto top of stack"
    [tempframe•(stackptr← stackptr+1) ← val]
top    "return value on top of stack"
    [⇑tempframe•stackptr]
pop | t           "pop value off stack and return it"
    [t← tempframe•stackptr.
    stackptr← stackptr-1. ⇑t]
!
```