Forth SOOP

WIL BADEN

```
CLASS classname

    VARIABLE membername

    n  CHARS BUFFER: membername

    n  CELLS BUFFER: membername

    n  CONSTANT membername

    : membername   ... ;

END

class  BUILD objectname

object member

class  SUBCLASS classname

    SUPER member

    COMMON forthword
```

For a classic look at simple object oriented programming let's start with Class — an extension of an existing Forth word.

A Class is a defining word for a collection of future definitions. As such, it is an extension of CREATE–DOES>, except that CREATE–DOES> is limited to one data area given as an address on the stack, and one function operating on that address.

A Class may have many named data areas and many named functions working with each other. A Class in Forth SOOP constructs its future definitions with **:** for a function, **CONSTANT** for a simple form of function, **BUFFER:** for a data area, and **VARIABLE** for a simple form of data area. A Class ends its future definitions with **END**.

A silly example shows this.

```
CLASS GREETING
VARIABLE COUNT
: HI  ." Hello "      1 COUNT +! ;
: BYE ." Good bye "  -1 COUNT +! ;
END
```

These definitions are incomplete. Before you can use them you must build an instance of them into an Object. This is called instantiation.

```
GREETING BUILD JOHN
```

Building an Object of a Class gives separate data areas for each Object, but will use the same definitions for functions. The data areas are initialized to 0

— you can use that in programming. Any initialization of data areas must be done after being built. The data areas are the Instance Variables of the Object.

Now you can use the operations.

```
JOHN HI   JOHN COUNT ?   JOHN BYE
```

You can build other objects with their own data areas.

```
GREETING BUILD JACK

JACK HI   JACK BYE
```

You can extend and specialize your future definitions.

```
GREETING SUBCLASS SPANISH
: HI  ." Buenos dias "  1 COUNT +! ;
: BYE ." Adios "  SUPER BYE ;
VARIABLE BASE
: STATE  COUNT @ BASE !  0 COUNT ! ;
END

SPANISH BUILD JUAN
```

SUPER BYE uses the definition of **BYE** in the superclass. **SUPER** isn't needed for **COUNT** because it's not defined in the subclass.

You can compile a usual Forth word with **COMMON forthword**.

An important part of object orientation is to hide implementation. Forth SOOP helps you do this by being case–sensitive to the names of definitions inside the Class, and treating references as upper case outside the Class. Thus in a Class all upper case words are Public and all lower case and mixed case words are Private. Outside a Class the case is irrelevant and Public words can be spelled with lower or mixed case.

The representation of Classes and Objects is simple.

```
CLASS
    The head of the list of members
    Size of data space needed
    Pointer to superclass

Member -- a VARIABLE, BUFFER:, CONSTANT, or :
    Name
    Execution-token

OBJECT
    Pointer to its class
    Assigned data space
```

SOOP and other approaches to object oriented Forth unify and supersede application vocabularies, **CREATE DOES>** defining words, and struct definitions. It can also encompass multi–tasking. Local variables become instance variables. The direct act of sorting is mostly eliminated.

In constructing a class the simple names of members make it easier to code, and the clearer code is more maintainable. Intelligent objects take the place of navigating the structure. Familiar names are used to construct the members of classes.

In a class, the data layout and operations are collected in one place. Operations that don't apply to an object can be kept away from it. Many run–time errors or crashes will be caught at compile–time.

```
 1     ( Class for Files )
 2     CLASS FILES

 4         VARIABLE File

 6         : REWIND  File @ COMMON REWIND ;

 8         : CLOSE        ( -- )
 9             File @ ?DUP IF  CLOSE-FILE ABORT" Can't CLOSE-FILE "
10                             0 File !
11             THEN
12         ;

14     END

16     ( Class for Input Files )
17     FILES SUBCLASS INPUTFILES

19         : OPEN              ( str len -- )
20              R/O COMMON OPEN-FILE ABORT" Can't OPEN-FILE " File !
21         ;

23         128 CONSTANT MAXLINE
24         MAXLINE 2 + CHARS BUFFER: Line

26         : READ-LINE    ( -- line length more )
27             Line DUP MAXLINE File @ COMMON READ-LINE
28                  ABORT" Can't READ-LINE "
29         ;

31         : READ         ( -- false | line length true )
32             READ-LINE ( line length more) DUP 0=
33                  IF  NIP NIP ( false) REWIND  THEN
34         ;

36         : LIST  BEGIN READ WHILE  TYPE CR  STOP? UNTIL THEN ;

38     END

40     ( Build object for the principal input file. )
41     INPUTFILES BUILD INPUT

43     ( Class for Output Files )
44     FILES SUBCLASS OUTPUTFILES

46         : OPEN                          ( str len -- )
47             2DUP DELETE-FILE DROP
```

```
48            W/O CREATE-FILE ABORT" Can't CREATE-FILE "
49            File !
50        ;

52        : WRITE                          ( str len -- )
53            File @ WRITE-LINE ABORT" Can't WRITE-LINE "
54        ;

56    END
```

CLASS
CLASS ***class***
Construct a class.
A class is a defining word for a collection of future definitions.

VARIABLE WITHIN A CLASS
VARIABLE ***membername***
Define a variable member of a class.

BUFFER: WITHIN A CLASS
n **CHARS BUFFER:** ***membername***
n **CELLS BUFFER:** ***membername***
Define a data area member of a class.

CONSTANT WITHIN A CLASS
n **CONSTANT** ***membername***
Define a constant member of a class.

: WITHIN A CLASS
: ***membername*** **... ;**
Define a function member of a class.

; WITHIN A CLASS
: ***membername*** **... ;**
Terminate the definition of a function member of a class.

END WITHIN A CLASS
END
Terminate the construction of a class.

BUILD
class **BUILD** ***objectname***
Build object *objectname* as an instance of *class* .
Objects are used ***object member*** .

COMMON WITHIN A CLASS
COMMON ***forthword***
Compile forthword from standard dictionary.

SUPER WITHIN A CLASS
SUPER ***member***
Compile a member, beginning lookup for it in the superclass.
This disambiguates members with the same name.

SUBCLASS
class **SUBCLASS** ***classname***
Construct an extension or specialization of a class.