TinyTalk, a Subset of Smalltalk-76 for 64KB Microcomputers

by Kim McCall and Larry Tesler

Xerox Palo Alto Research Center

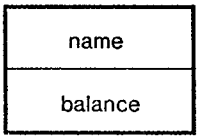
Palo Alto, CA 94304

SMALLTALK-76

Smalltalk-76 is an interactive, object-oriented programming language.

All data structures are represented as *objects*. For example, a simple bank account might be represented as an object with two data fields, *name* and *balance*, along with a set of *methods* (procedures) to operate upon those fields.

A bank account



Each object is an *instance* of some *class*. The class defines the internal structure and the behavior of all its instances. The object shown above might be an instance of class BankAccount.

Processing is invoked by sending *messages* to objects. Sending a message to an object is very much like calling a procedure in a procedural language. Among the messages to which a bank account could respond might be:

balance

deposit: amount

withdraw: amount

A message consists of an identifying key called a *selector* and zero or more *arguments*. The selectors of the messages shown above are *balance*, *deposit:*, and *withdraw:*. Two of those messages have an argument named *amount*.

The response to a message is implemented by a *method* (procedure) which can evaluate expressions, assign to variables, and/or send further messages in order to achieve the desired result. A bank account might respond to the message *deposit: amount* by invoking the following method:

```
balance ← balance + amount
```

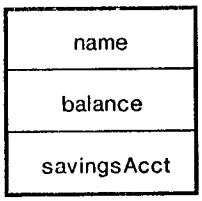
Each class has a *message dictionary* which associates a set of selectors with a corresponding set of methods. Part of the message dictionary of class BankAccount might be:

Selector	*Method*
balance	balance
deposit:	balance ← balance + amount
withdraw:	IF balance < amount THEN false ELSE balance ← balance - amount

When a message is sent to an object, the selector is looked up in the message dictionary of the object's class in order to find the appropriate method to execute. Thus, there is an important difference between sending a message in Smalltalk and calling a procedure in a procedural language. The decision of exactly which response is appropriate is left to the receiver instead of being decided ahead of time by the sender. The exclusive use of such "generic" procedures enhances modularization of programs, because the sender of a message doesn't have to be concerned at all about the internal structure, or even the class, of the receiver.

Further factorization of the representation of data and methods in the language is facilitated by *subclassing*. Any class may be declared to be a subclass of another, thus inheriting its code and data structure. The subclass can add further specificity of its own. For example, class OverdraftBankAccount might be a subclass of BankAccount that allows overdraft. Each instance could have an extra field, savingsAcct, to reference another account that could be tapped in case of an overdraft.

An overdraft bank account



Within class OverdraftBankAccount, the method **withdraw: amount** might be defined as:

```
IF balance < amount THEN
    IF savingsAcct withdraw: amount - balance THEN
        balance ← 0
    ELSE false
ELSE balance ← balance - amount.
```

The above summary is intended only to sketch the barest essentials of Smalltalk-76. A more complete discussion of the language can be found in [Ingalls].

Several versions of Smalltalk-76 have been implemented on minicomputers with 128K bytes or more of RAM, a swapping disk, a high resolution bit-mapped display, and a pointing device. The programming environment includes over a hundred classes, thousands of methods, and thousands of other objects. Applications programs typically define a dozen or more classes and a hundred or more procedures, and they may create thousands of additional objects.

TINYTALK

The programming language TinyTalk provides a subset of Smalltalk-76 which is suitable for implementation on a 64K byte microcomputer. The TinyTalk compiler accepts essentially the full Smalltalk-76 syntax. Its main restriction is that space limitations permit only a few thousand objects. TinyTalk does not support multiple processes. It does not require or support a bit-mapped display, or a pointing device, but it can be used with any standard ASCII keyboard and terminal. While Smalltalk-76 uses reference counting to determine when objects can be deallocated, TinyTalk employs a compacting garbage collector to guarantee reclamation of the limited space available. Finally, some of the performance optimizations incorporated into Smalltalk-76 have been left out of TinyTalk for the sake of simplicity and space.

We have implemented TinyTalk on a microcomputer containg an Intel 8086 and 64K bytes of RAM. Most memory space is occupied by the interpreter, storage manager, and standard class library (including an incremental compiler and an interactive source language debugger), leaving approximately 8K bytes of free space for user programs -- about two or three pages of source code.

Because TinyTalk's space and I/O restrictions are fairly severe, it should be clear that the system is intended for students who wish to gain practice in object-oriented programming rather than for supporting large-scale applications programs.

IMPLEMENTATION

All objects other than integers are referenced within the system by an Ordinary Object Pointer (*OOP*). The OOP is an index into an indirect pointer table. Each entry in the table gives the location in the heap at which the fields of the corresponding object are stored. The indirect table makes relocation of the object during compaction very cheap and easy.

When a new object is created, space is allocated on the top of the heap. The heap is allowed to grow until it reaches a certain limit, at which time the compacting garbage collector is invoked.

Object code in TinyTalk is made fairly compact by a *resident incremental compiler* which translates source code programs into *byte-code strings* whose syllables represent the main atomic operations performed by the system. The byte-code strings are executed by the TinyTalk *byte-code interpreter.* The interpreter uses a conventional push-down stack for keeping track of its state. See [Ingalls] for a more detailed discussion of the byte-code interpreter.

SAMPLE SESSION

A sample dialogue in TinyTalk might proceed as follows. The user's input is prompted by a '!'. The system's reponse is shown on the next line.

```
! acct ← BankAccount new name: 'John A. Doe'
    John A. Doe
! acct deposit: 100
    100
! acct deposit: 250
    350
! acct withdraw: 375
    false
! acct withdraw: 325
    25
```

CONCLUSIONS

TinyTalk is a simple object-oriented language that has been implemented on a 64KB microcomputer. It is suitable for reimplementation and/or use by students who wish to gain experience with object-oriented programming languages. We plan to publish the details of the implementation in the near future as part of a larger work about Smalltalk.

REFERENCES

[Ingalls, Daniel H.] "The Smalltalk-76 Programming System: Design and Implementation" *Fifth Annual ACM Symposium on Principles of Programming Languages*, Tucson, Arizona, January 23-25, 1978.