Object Oriented Programming (Tutorial).

Conference Paper *in* ACM SIGAPL APL Quote Quad · May 1990

DOI: 10.1145/97811.97817 · Source: DBLP

1 author:

Manuel Alfonseca
Universidad Autónoma de Madrid
184 PUBLICATIONS **1,254** CITATIONS

OBJECT-ORIENTED PROGRAMMING TUTORIAL

By Manuel Alfonseca
IBM Madrid Scientific Center
Paseo de la Castellana, 4
28046 Madrid (SPAIN)

ABSTRACT

A classical procedural program (written in COBOL, FORTRAN, BASIC, PASCAL, LISP or APL2) is made of sentences that execute sequentially in a predefined order, that depends only on the values of the data the program is working with. This order can usually be deduced by visual inspection of the program.

A non-procedural program (written in PROLOG, for instance) contains a certain number of instructions that will not be executed in a predefined order. They receive control from an *inference processor*, a procedural program that decides in every moment the order in which the sentences of the program should receive control (should be fired).

In both the procedural and the non-procedural cases, the basic unit of execution is the *program*. The data only provide values that will be used to perform computations or to decide the order of execution. A given application is a hierarchical set of programs (modules) each of which is capable of invoking other programs in the hierarchy. The data may be global (accessible from every program in the hierarchy) or local (accessible by the program where they belong and, sometimes, by those at a lower level in the hierarchy).

In Object-Oriented Programming (OOP in short), things are different. Here it is the *data* that are organized in a basic control hierarchy. One piece of data may be linked to another through a relation of descendancy, and this fact gives rise to a network (usually a tree) similar to the hierarchy of programs in procedural programming. There are also programs in OOP, but they are appendages to the data (in the same way as in classical programming data are appendages of programs). It is possible to build global programs (accessible to all the data in the hierarchy) and local programs (accessible from certain objects and their descendants).

In OOP, the execution of a program is fired by means of a *message* that somebody (the user, another program or an object) sends to a given object. The recipient of the message decides which program should be executed (it may be a local program, or a global program which must be located through the network that defines the structure of the objects).

There is a certain amount of confusion on what is Object-Oriented Programming and what is not. This has happened before, with other fields in Computer Science. There are still people, for instance, that call Artificial Intelligence to any program that is written in Prolog or Lisp. In the same way, there are those who maintain that any program written in Smalltalk, C++ or Objective C is OOP. As in the AI example, this is not always the case.

Another source of confusion comes from the fact that Object-Oriented Programming has been frequently used to build complicated user interfaces, with window systems, icons and so forth, and this has produced the unexpected result that many people believe that any program including these interfaces is OOP. Again, this is obviously wrong.

The basic elements of OOP are **Objects**, **Methods** and **Messages**.

An **Object** is a complex data element that possesses structure and is a part of an organization. It embodies three different concepts:

- A set of relations to other objects (usually represented by pointers).
- A set of properties (which have values).
- A set of methods (defined by means of executable code).

A **method** is a procedural program written in any language. What other programming systems call **functions**, **programs** or **procedures**, Object-Oriented Programming calls **methods**. There is no essential difference between programs in any language and methods, except for the fact that in pure OOP the CALL instruction is not allowed. The reason for this is obvious: the CALL instruction is

used in classical programming to build the hierarchy of programs, that OOP replaces by a hierarchy in the data. Therefore, OOP can be defined as *programming without CALL*, in the same way that Structured Programming was defined as *programming without GOTO.*

A message is a request to activate a method sent by an object to another object. The requested method can belong to the receiving object, or it may be inherited from one of its ancestors in the hierarchy.

The fundamental properties of object oriented programming can be summarized thus:

- **Encapsulation**: all the information related to a given object (including properties, methods and relations) is directly accessible from this object and form a part of its structure. The object can be transplanted to another data organization as a self-defined unit.
- **Inheritance**: objects may inherit properties and methods from other objects. In this way, it is not necessary to define many times a given property if it is shared by a certain number of objects. Inheritance may be simple (if the object has a single parent) or multiple (if there are several parents for a given object).
- **Polymorphism**: programs (methods) and properties may be made local to certain objects and their descendants. Thus, the same names can be used at different objects to perform related, but different procedures.

The main benefits of OOP can be summarized as follows:

- **Modeling Power**: OOP data structures and hierarchies are more similar to conceptual models than the hierarchical code structures of classical programming. Therefore, the conversion from the mental model to the programmed application is more natural and simple.
- **Modularity**: Objects are natural modules. Encapsulation isolates their data and programs from other objects. Information hiding makes it possible to work in a truly modular way. A large application can be separated into pieces, each of which can be independently programmed by different people with very few interface considerations.
- **Extensibility**: The hierarchical structure of OOP data, together with hiding and inheritance, make it very easy to add new instances, new classes, new properties or new behavior to a given application. Incremental programming is thus a typical property of OOP applications.
- **Elimination of redundancy**: Properties and methods common to many objects can be defined at a common ancestor and inherited from that ancestor. Therefore, a given property or method need be defined only once.
- **Reusability**: Classes in an OOP system (common ancestors to many objects) can be transported and reused in many applications with practically no change.

OOP languages can be classified in two groups: pure OOP languages (where the CALL instruction is strictly forbidden), such as SMALLTALK and Hypertalk, and hybrid OOP languages, that allow it in certain cases, such as C++, Objective C, Object Pascal, Eiffel and APL2 (see references 1 to 5). This tutorial will describe in detail how to do OOP in APL2.

1.0 References

1. Cox, B. *Object-Oriented Programming: an Evolutionary Approach.* Addison Wesley, 1986.
2. Meyer, B. *Object-Oriented Software Construction.* Prentice Hall, 1988.
3. Peterson, G. (editor) *Object-Oriented Computing.* Vol. 1 & 2. IEEE order no. 821, 822. 1987.
4. *Smalltalk-V User's Guide.* Digitalk Inc.
5. Alfonseca, M. *Frames, Semantic Networks and Object-Oriented Programming in APL2* IBM J. Res. Dev., 33:5, p. 502-510, Sep. 1989.