Basic Concepts in Object Oriented Programming.

By Kristen Nygaard,

University of Oslo, Norway.

1. History.

The term **object-oriented programming** is derived from the **object** concept in the Simula 67 programming language. In that language an execution of a computer program is organized as the joint execution of a (possibly variable) collection of objects. The collection as a whole is represented by a **system object**, and objects sharing a common structure are said to constitute a **class**, described in the program by a common **class declaration**.

The concept may, however be traced further back, to the introduction of digital simulation as an important tool for analysis in the 1940s.

This development took place in many different countries. The author's involvement started in 1949, when an integral equation model used for calculating the diameter of the uranium rods of the first Norwegian nuclear reactor was substituted by a Monte Carlo simulation ("by hand"). In that model the physical paths and histories of a large number of neutrons were generated and a statistical analysis of their properties used to estimate the proper choice of rod diameter.

Later on, the same approach was used in operational research situations when no appropriate mathematical model could be generated: Instead a process portraying the situation to be analyzed was generated within a computer, with entities in the outside phenomenon being represented by corresponding entities appearing in the computer program execution.

A number of **simulation languages** started to appear, each representing a standardized view of the systems to be simulated. Simscript, GPSS, CSL, Sol and Simula I are early examples.

In Simula I a more general view was adopted. The purpose of Simula I was from the outset to provide, at the same time, both a **system description language** and a **simulation programming language.**

Simula should give its users a set of concepts in terms of which they could comprehend the system considered and a language for precise and complete description of its properties. It should thus be a tool both for the person writing the description and for people with whom he wanted to communicate about the system.

At the same time this system description should, with the necessary input/output and data analysis information added, be compilable into a computer simulation program, providing quantitative information about the system's behaviour.

In Simula I the later Simula 67 "classes" are named "activities" and "objects" named "processes".

Simula I, designed in 1962-64 and available in 1965, was immediately used outside simulation, as a general programming language: The computing process could by Simula be organized as system of interacting program execution components, and this approach proved to be fruitful in a wide range of application areas. This, and some new ideas (e.g. subclasses and virtuals) resulted in the current Simula language, Simula 67, which is a general programming language, also used as a simulation language.

The object oriented approach to programming proved its usefulness in a number of areas, exploited by many researchers, e.g. :

- Simulation.
- Organization of concurrency, through the monitor concept.
- Structured programming.
- Abstract data types. The class concept encapsulates both the substance, the representation of values and the operations associated with a data type in one declaration.
- VLSI design.

A major widening of the use of object oriented programming occurred as a result of the development of the Smalltalk language. In Smalltalk the incremental program execution of Lisp is combined with Simula's class/subclass and virtual concepts. The development was associated with the parallel development of personal work stations that were in need of languages of this kind. Object oriented constructs were combined with Lisp : through "Flavors" in Zeta-Lisp (with "multiple inheritance" added) and through the Inter-Lisp based language Loops.

Today a number of new object oriented languages are appearing. At he same time important work is being done to develop other paradigms, in particular function oriented programming and logic programming.

2. Informatics, Information Processes and Structure.

In the opinion of the author, the science of informatics is not an abstract science, akin to mathematics. It relates to the study of real-world phenomena:

Definition 1: **Informatics** is the science that has as its domain the information aspects of phenomena in nature and society.

The most important phenomena studied in informatics are information processes. Program executions in computers, information handling in offices, planning in corporations etc. are examples of such processes. The above definition of informatics does not imply that all phenomena to which we may associate information aspects "belong" to informatics. Informatics represent one way of looking at phenomena, a **perspective**. Data processing in a post office may be regarded as an information process, as an economic

process or as a social process, and thus "belong" to informatics, economics or sociology, depending upon the perspective chosen.

The perspective of informatics and then of object-oriented programming may be presented through the sequence of concepts defined in this and the next section.

Definition 2: A **process** is a development of a part of the world through transformations during a time interval. **Structure** of a process is limitations of its set of possible states, and thus of its set of possible sequences of states.

In most programming languages the transformations are prescribed by **imperatives** (and thought of as actions) in process structure descriptions called **programs**.

When we apply the perspective of informatics, processes will be regarded as information processes, implying that specific qualities of the process are considered:

Definition 3: The three qualities of an **information process** are:

- its **substance**, the physical matter of which it consists (objects, files, records, variables etc.).
- **measurable properties** of its substance (e.g., **values** of variables).
- **transformations** of its substance and thus its properties (the execution of imperatives).

Definition 4: A **state** of an information process is expressed by describing:

- the moment at which the state is recorded,
- its substance, measurements of its properties, and transformations going on, all at that moment.

Definition 5: An **attribute** is the association of some substance and:

- a name,
- a set of elements,
- at any given time one of the elements of that set.

Definition 6: A **reference** is an attribute for which the associated set has **substance** as elements.

Definition 7: A **quantity** is an attribute for which:

- a way of **measuring** a property of its substance is defined,
- there exists a **mapping** from the set of measurements of every possible state to the elements of the attribute's associated set, called the **value set**,
- the associated element, called the attribute's **value**, corresponds to the measurement of the property at the given moment.

Definition 8: A **pattern** attribute is an attribute prescribing **common structure** of a category of objects.

In the definition of informatics given above contains the term "information". It is the opinion of the author that this term only may be defined usefully in science as associated with a perspective, as e.g. "geographical information", "chemical information". One way of defining "information in informatics" is to state that within informatics:

Definition 9: Information is description of the structure and state of processes in terms of their reference, quantity and patterns attributes and their ongoing transformations.

This definition relates strictly to the basic information process level in the process/structure hierarchy, and not to the complex processes of e.g. system development.

In programming the task is to prescribe the structure of a program execution, that is, an information process. The program is both a description for people of the process and, at the same time, a prescription of structure for the computer. The tools for making these descriptions/prescriptions are the programming languages. A basic characteristics of a programming language thus is the way in which the language organizes the structure of the substance, the measurable properties and their values, and the transformations of state within the process.

3. Systems and Object Oriented Programming.

Within informatics the notion of a **system** is very often encountered. In the general Simula-related system description language DELTA the following system definition is introduced:

Definition 10: A **system** is a part of the world that a person (or group of persons) **chooses** to regard as a whole consisting of **components**, each component characterized by **properties** that are selected as being relevant and by **actions** related to these properties and those of other components.

According to this definition no part of the world "is a system" as an inherent property. It is a system if we choose a system perspective.

We are now able to state that:

In object oriented programming an information process is regarded as a system developing through transformations of its state. The substance of the process is organized as the system components, called objects. A measurable property of the substance is a property of an object. Transformations of state are regarded as actions by objects.

4. Abstraction Mechanisms and Hierarchies.

A programming language contains abstraction mechanisms - tools for describing the common structure of similar phenomena. Using the above definition of an information process, it is seen that:

- A **class declaration** abstracts substance.

- A **type declaration** abstracts values and measurable properties.

- A **procedure declaration** abstracts action.

In the BETA object oriented programming language these declarations are unified into one abstraction mechanism, the **pattern declaration.**

Another important aspect of programming languages are the tools available for organizing hierarchies. Using BETA (and Simula) as an example, we find that:

- Action hierarchies are organized through stacks.

- Value hierarchies are organized through patterns/subpatterns and by the not yet fully developed value specification constructs.

- Substance hierarchies are organized through nested objects ("objects within objects").

- Structure (concept) hierarchies are organized through patterns/subpatterns (classes/subclasses).

- Program execution (from hardware through operating systems layers to program execution) through the actor/role concept (in BETA).

5. Perspectives on Programming.

Within informatics there are many important but different perspectives that may or may not be mutually conflicting. In programming three important perspectives are:

- Function oriented programming: The computing process is viewed as a sequence of applications of functions to an input, producing an output, that in its turn may be the input to another functions etc.

- Object oriented programming: The computing process is viewed (as described above) as the development of a system, consisting of objects (components), through sequences of changing states.

- Constraint oriented programming: The computing process is viewed as a deduction process, developing from an initial state, the process being restricted by sets of constraints and inputs from the environment, information about the states being deduced by an inferencing algorithm. Logic programming using first order predicate logic is currently the most important example (Prolog).

It seems obvious to the author that all these three perspectives should be supported within any new general programming language in the future. No perspective will "win" as some seem to believe.