The OO Paradigm

Without a Language or Syntax!
What is the object world all about?

The Object-Oriented System Ontology

This ontology is consistent with the practice in computer science and information science categorizing a domain of concepts (i.e. individuals, attributes, relationships and classes). In this ontology of the object-oriented paradigm I attempt to minimize the vestiges of implementation languages and development methodologies in order to expose the core nature and value of object-oriented concepts.

1. Individuals

The most concrete concept in the object-oriented paradigm is the object. It derives from the living physical experience of humans seeing and touching things. In that experience objects are separable – distinguishable from other objects by nature of their physical presence and location regardless of any other discernible characteristics they may possess. This characteristic of “individual-ness” leads to the property of identity. Identity enables the unambiguous designation or selection of every object physical or abstract within a domain of discourse.

Objects have an “inside,” an “outside,” and a “surface” that separates the inside from the outside. An object contains anything that exists on the “inside” of the object. Since the surface of most physical objects is opaque, usually the contents are invisible and untouchable by anyone on the outside. This property renders the object’s contents impervious to meddling and is called encapsulation (or information hiding).

2. Attributes

Attributes are those characteristics that are inherent to an object. In the object paradigm attributes define either data or behavioral characteristics - each of which has a static and dynamic form. Attributes in static form combine to define what is called the structure of an object. From inception to extinction the structure of an object is immutable.

2.1. Data Attributes

Data attributes serve to store information (data) within an object and implement the property of remembrance. Data attributes are completely contained within an object protected by encapsulation. Remembrance is manifest statically as “what can be remembered,” a data attribute variable. It is manifest dynamically as a definition of “what is remembered,” a particular data attribute value.

2.2. Behavioral Attributes

Behavioral attributes serve to define the animate nature of an object. In its static form each behavioral attribute defines “what an object can do,” usually called a service. In its corresponding dynamic form this behavioral attribute defines “how a service is accomplished,” usually called a method (or operation). Methods define “activity” performed in an object model. A method may simply be access to remembrance inside an object or it may be complex sometimes employing the involvement of other services of the same or other objects to accomplish its responsibility. Methods reside within the object subject to encapsulation while services are visible at the surface of the object available for collaboration.

3. Classes

The class concept combines both a definition of structure and the generation of object(s) based on that structure. Every object is an instance of a specific class and shares the same static structure defined by that class with every other object of that class. The responsibility of generating instances that share the same structure is the property of progeny. The class concept thereby fuses the existence of the objects to that of their class; objects cannot exist independent of their defining class. Objects are said to be members of their class.
Along with the static behavioral structure of *service* defined in the *class*, the dynamic behavioral attribute, *method*, may also be defined. Defined in the *class* this dynamic behavioral attribute, “*how* a service is accomplished,” is identical for each and every *object* generated of that *class*.

4. Relationships

Relationships in the object paradigm exist on two dimensions: structural and behavioral. The structural relationships are based primarily on the properties of *identity, remembrance* and *progeny*.

4.1. Structural Relationships

4.1.1. Inheritance

Inheritance is a relationship between *classes*. The structure defined in one *class* is used as the foundation of *structure* in another. By foundation it is meant that all the *structure* of the first is replicated in the second and additional *structure* in terms of *data attributes* or *services* may be added or *methods* for replicated services may be altered (overridden). The replicated *structure* defines how the two *classes* are alike. The additions or alterations define how they are different. The *class* defining all the *structure* shared between them is called the *parent class* (*super class, generalization*) while the other is called the *child class* (*sub class, specialization*). It is said that the *child class* proceeds from or is derived from the *parent class*. Successive application of *inheritance* defining related *classes* results in a *class hierarchy*.

4.2. Behavioral Relationships

The behavioral relationships are based primarily on the property of *membership* IN, and the capacity of *objects* to “act.”

4.2.1. Association

An *association* is a relationship between *objects*. *Objects* are intrinsically separable by way of the *identity* property. At the same time, humans are compelled to categorize their experience of things in the physical world. Humans superimpose groupings that collect *objects* into *sets* (a foundation of mathematics based on human experience). *Objects* become members in a group only by designation. This property is called *membership*. Membership is independent of *identity* or *attribute*. This property also permits humans to identify an *object* that is not in a set (i.e. discrimination). (Membership in a group is discretionary and is distinct from membership of a *class* which is intrinsic by way of progeny.)

Variations on *membership* derive from the intent of the relationship and generally fall into the categories of *association* and *composition*. Any designated collection of objects defines a relationship between those *objects* called *association*. By the simple fact that they are members in the same relationship that membership defines how they relate. When the existence of the *objects* themselves is coupled with their membership; that is to say, if one (or the other or both) would not exist if it were not related to the other then the relationship is called a *composition*.

4.2.2. Message Passing

*Message passing* is a relationship between *objects*. *Message passing* relies on the *identity* property and *services*. A *message* is a communication between a *sender object* and *receiver object* where the *sender* requests that the *receiver* render one of its *services*. The *sender* and *receiver* may be one in the same *object*. The *message* designates the *receiver’s identity*, the *receiver’s service* to be performed along with any parameters that the *service’s protocol* may require. Since the *message* is a request there are no implicit timing constraints determining when the *service* is accomplished. Unless explicitly designated a *message* results in an asynchronous activity on the part of the *receiver* without acknowledgment or returned information.

4.2.3. Polymorphism

*Polymorphism* results from the interplay of *message passing*, *behavioral attributes* and *classes*. A *sender* directs a *message* to a *receiver* designating a *service* of that *receiver*. A *message* does not designate a *method*. The regime that determines which *method* satisfies a service request is called *binding*. If the *method* (corresponding to the *service*) is defined in the *class* of the *receiver object*, that *method* is invoked. If the *service* of the *receiver’s class* is inherited (and not overridden), the corresponding *method* defined in the nearest progenitor (*parent class*) of the receiving *object’s class* is invoked.

©2007, Les Waguespack, Ph.D.