A Two-Page "OO Green Card" for Students and Teachers

Leslie J. Waguespack, Jr., Ph.D.
LWaguespack@Bentley.edu
Computer Information Systems Department, Bentley College
Waltham, Massachusetts 02154-4705, USA

Abstract

Long before the current political turbulence surrounding immigration became so wide-spread almost everyone in the computing industry recognized the term "green card" as a pocket-sized reference document describing the most commonly required detail-knowledge about a computer's architecture (e.g. IBM 360 Green Card). It placed at ready-reach the details of formats, operations, resource locations and encodings that defined the immutable structures that a machine-level programmer would need to hold close during the programming and debugging of system software. The metaphor is used here to describe an attempt to provide the same ready-reach reference to the immutable details of the object-oriented paradigm by means of a highly distilled explanation of the terminology and operational relationships – language usually referred to as an "ontology." The object-oriented paradigm has been "mainstream" in IS education for ten years and for some twenty years it's been "mainstream" in IS development. Although familiar with the syntax of one or more OO programming languages, the underlying OO concepts remain a mystery to many IS students. And if the current crop of IS textbooks are any indication, they remain somewhat of a mystery to many IS educators. The "green card" described here attempts to address both concerns: offering a programming language-independent explanation of OO concepts and delivering it in a condensed format that can underpin pedagogy across implementations, languages and methodologies.

Keywords: object-oriented paradigm, object-oriented ontology, object-oriented pedagogy, object-orientation, object-orientation quick reference

1. INTRODUCTION

The object-oriented paradigm has been in the "mainstream" of information system development for the last two decades. It has been "mainstream" in IS education for at least the last decade. In many instances the only exposure that students and some faculty have to the object-oriented paradigm comes through tools and programming languages all of which represent not only an incomplete subset of object-oriented concepts but, they often also include interpretations, additions and omissions that serve their respective designers' opinions for efficiency and/or convenience. Needless to say, these designers' primary goals do not emphasize paradigm clarity.

The fact that equilibrium in the interpretation of the OO paradigm is yet unattained is evidenced in a recent journal article that attempted to define the fundamental aspects of the object-oriented paradigm through the "democratic" approach of counting the occurrence of OO terms used in academic publications. (Armstrong 2006) While this approach sheds light on the terminology that garners the most attention in academic discourse it's value as a paradigm definition is somewhat dubious. Other evidence that the OO paradigm remains somewhat of a mystery among IS educators are the numerous spurious explanations that are found in contemporary IS textbooks on analysis and design as reported in a survey on the textbook treatment of modeling. (Waguespack 2006) And at a recent international conference on IS education one enthusiastic presenter explained that employing the Unified Modeling Language (UML) for data modeling required no adaptation in pedagogy because there was no appreciable difference modeling using the object-oriented paradigm versus the entity-relationship model!

Although it may be true that the only valid definition of a programming language exists in the implementation of its compiler, that does not seem to be an appropriate means of defining the object-oriented paradigm nor establishing a pedagogy for expressing it. Therefore implementation is a banished element in this attempt at describing the object-oriented paradigm and the explanation found in the "two-page OO green card" relies only on the abstractions distilled from seminal expositions of the concepts as found in original descriptions. (Dahl 1966, Wegner 1990) The interested reader can find a more complete history of the object-oriented paradigm in (Capretz 2003).

2. THE OO PARADIGM WITHOUT LANGUAGE OR SYNTAX

Every *language* that is invented to express concepts carries with it the understanding and the biases of the inventor. Depending on his/her purpose(s) those biases simplify certain tasks performed with the language but, may obscure underlying concepts.

As a special case programming language design in addition must cope with the feasibility of automated translation and interoperability with other programming languages and operating systems. Designers must consider upward, downward, and cross-compatibility within versions of a programming language. Compromises and assumptions are chosen to make the resulting language efficient, effective and marketable.

The goal of this description of the object-oriented paradigm is to strip away the extraneous facets that programming language design must use to achieve their "practical" product requirements; and in so doing to succinctly make the underlying object-oriented paradigm concepts evident and understandable. This is an ambitious task to say the least! But, if it may be achieved, it provides a knowledge-base that the teacher and student can carry from one object-oriented programming language to another exposing how they treat an OO concept alike or how they treat it differently.

3. ONTOLOGY OF THE OBJECT-ORIENTED PARADIGM

The ontology presented here is consistent with the practice in computer science and information science categorizing a domain of concepts (i.e. individuals, attributes, relationships and classes). This ontology of the object-oriented paradigm attempts to eschew the vestiges of implementation languages and development methodologies in order to expose the core nature and value of object-oriented concepts. The object-oriented ontology is arranged as follows (and is depicted graphically in the map in Figure 1 below while an illustration of the two-page rendering of the "green card" is found in appendix A):

A. Individuals
B. Attributes
   - Data Attributes
   - Behavioral Attributes
C. Classes
D. Relationships
   - Structural Relationships
     - Inheritance
   - Behavioral Relationships
     - Association
     - Message Passing
     - Polymorphism

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 selec-

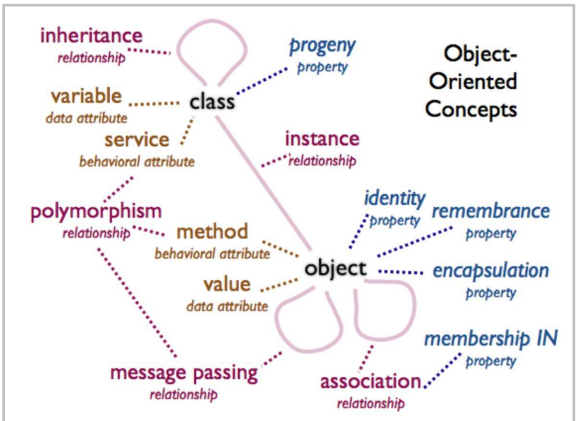


Figure 1 – Object-Oriented Concept Map

tion 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*).

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.

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*.

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.

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 also identical for each and every *object* generated of that *class*.

Relationships – Relationships in the *object* paradigm exist on two dimensions: structural and behavioral.

Structural Relationships – The structural relationship is based primarily on the properties of *identity*, *remembrance* and *progeny*.

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*.

Behavioral Relationships – The behavioral relationships are based primarily on the property of *membership IN*, and the capacity of *objects* to "act."

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 in*. *Membership in* 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* that 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*.

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.

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.

4. DISCUSSION

By design this ontology omits a variety of object-oriented language characteristics that are a matter of designer's choice rather than paradigm. There are myriad examples. Here are but a very few.

OO languages treat *encapsulation* in rich variety. Visibility and accessibility rules in C++ are governed by the arrangement of programming elements in the file structure of the source code text – the inclusion or repetition of "headers." (Stroustrup 1986) Java approaches the issue with a variety of visibility options: *private*, *protected* and *public*. (Schildt 2007) Languages such as Smalltalk adhere to the paradigm description above more strictly by preventing any access to object attributes except via the agency of an object's services. (Goldberg 1983)

Inheritance is likewise treated with variety. Some languages like Smalltalk allow only a single *parent class* for any *child class* while other languages like C++ permit multiple *parents*. This distinction leads to numerous issues that must be considered when the paradigm reaches the stage of methodology and implementation, but these issues do not involve the nature of the OO paradigm and eventually fall into the arena of style preferences. And as such they become the matter of quality assessment rather than paradigm definition.

Some OO programming languages treat the definition of structure that is the *class* as an *object* itself (i.e. "classes are 'first-class' objects"). In this interpretation, along with their definitional role providing the template

of structure for their progeny, each class is also an object – sometimes with its own data and behavioral attributes distinct from those designated for its "offspring."

In terms of paradigm comparison the property of *identity* defined in this ontology casts into clear distinction the notion of identity in the entity-relationship model that remains the predominant paradigm for data and database modeling. In the OO ontology *identity* is independent of its realization: "*Identity* enables the unambiguous designation or selection of every *object* (physical or abstract) within a domain of discourse." In most object-oriented implementations the *identity* of an object is realized by an "object identifier" in some form that sustains the object's unique *identity* regardless of any of its attributes. In the entity-relationship modeling paradigm however, the *identity* of an instance is determined by a unique (and therefore unambiguous) combination of attribute values collectively referred to as a candidate key or by designation the primary key. (Wegner 1990)

5. SUMMARY

In this very short presentation we propose a succinct, compact description of the object-oriented paradigm without the embellishments or compromises often necessary to support computer-based translation (as in a compiled language) or a graphically augmented representation such as UML. The ontology is derived from the very earliest of conceptions of the object-oriented paradigm at a time before there was competition for commercial-dominance, language or methodology standardization. The primary value of this approach to explaining the object-oriented paradigm is two-fold.

First, absent the accidents of implementation that accompany all programming languages both the student and teacher of object-orientation have a basis for discriminating between those features that are essential to the paradigm and those that are accidental to an implementation of it. (Brooks 1987) It also facilitates assessing OO's role in more advanced applications of the paradigm (e.g. in areas such as reuse and component-based systems engineering). (Waguespack and Schiano 2006)

Second, the individual characteristics depicted are primarily elemental. These characteristics may be readily distinguished from one another and identified in other paradigms of modeling and programming languages thus permitting pedagogies to emerge patterned after Ledgard's "ten mini-languages." (Ledgard 1971).

Object-orientation has been likened to a religion with its saints, zealots and heretics. For that reason and the fact that at its core it is a framework or pattern for creating abstractions, conceptions in the human mind, it may not be possible to find a unique depiction of the paradigm itself. As with all models, this explanatory model for the object-oriented paradigm cannot be judged as perfect, but perhaps it may be judged as useful.

6. ACKNOWLEDGEMENTS

Special thanks are due my colleagues in Computer Information Systems at Bentley College for their insightful discussions and comments on several earlier drafts of these ideas and to Bentley's administration for the continuing support of this exploration. Thanks also to the computing faculty at the University of South Alabama who provided an opportunity for and contributed to some the earliest discussions of this work. Thanks to the ISECON community (organizers, editors, reviewers and participants) for continuing to support a venue where ideas and discussions of information systems education can be aired and debated for the benefit of all our students and our disciplines.

7. References

Armstrong, D. J (2006) "The quarks of object-oriented development," *Communications of the ACM*, 49, 2 (February): pp. 123-128.

Brooks, Frederick P. (1987) "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer*, Vol. 20, No. 4 (April) pp. 10-19.

Capretz, L. F. (2003) "A brief history of the object-oriented approach." *SIGSOFT Software Engineering Notes* Vol. 28, No. 2 (March), p. 6.

Dahl, O. J. and Nygaard, K. (1966) "Simula: An Algol-Based Simulation Language," *Communications of the ACM*, Vol. 9, No. 9, (September), pp. 671-678.

Goldberg, A. and Robson, D. (1983) Smalltalk 80: The Language and its Implementation, Addison-Wesley, Reading, Massachusetts

Ledgard, H. F. (1971) "Ten Mini-Languages: A Study of Topical Issues in Programming Languages." *ACM Computing Surveys* Vol. 3, No. 3 (Sep.), pp. 115-146.

Schildt, Herbert (2007) Java™: The Complete Reference, Seventh Edition, McGraw-Hill Osborne Media, Two-Penn Plaza, New York, NY

Stroustrup, Bjarne (1986) The C++ Programming Language, Addison-Wesley, Reading, Massachusetts

Waguespack, L.J. (2006) "Metaphors, Polymorphism, Domain Analysis, and Reuse: Teaching Modeling in the Object-Oriented Paradigm." *Information Systems Education Journal*, 4 (81), (2006).. http://isedj.org/4/81/. ISSN: 1545-679X

Waguespack, L.J. and Schiano, W.T. (2006) A Reuse Reference Grid for Strategic Reuse Goals Assessment, Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)

Wegner, P. (1990) "Concepts and paradigms of object-oriented programming." *SIGPLAN OOPS Messenger,* Vol. 1, No 1. (August), pp. 7-8

Appendix A

Green Card Illustration

The OO Green Card may be effectively reproduced as the front and back of a single 8.5" x 11" sheet of paper. Terms used with special meaning are italicized. Those initially defined are also bolded.

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

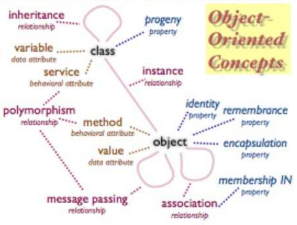
Data attributes serve to store information (data) within an *object* and implement the property of ***remembrance***. Data attributes are completely contained within an object protected by *encapsulation*. *Remembrance* is manifest statically as "what *can* be remembered," a ***data attribute variable***.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

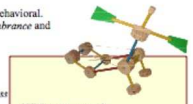
Variations on *membership* derive from the intent of the relationship and generally fall into the categories of *association* and *composition*. Any designated collection of objects defines a relationship between those *objects* called ***association***. By the simple fact that they are members in the 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.



Without syntax?

Every *language* that is invented to express concepts carries with it the understanding and the biases of the inventor. Depending on his/her purpose(s) those biases simplify certain tasks performed with the language but may obscure the underlying concepts.

Programming language design must deal with the feasibility of automated translation and interoperability with other programming languages and operating systems. Compromises and assumptions are chosen to make the resulting language efficient, effective and marketable.

The goal of this description of the object-oriented paradigm is to succinctly make the concepts understandable - an ambitious task to say the least!

- Professor Waguespack