



COMPUTING INTEGRITY

INCORPORATED

60 Belvedere Avenue
Point Richmond, CA 94801-4023
510.233.5400 Sales
510-233.5444 Support
510.233.5446 Facsimile



Object-Oriented Vocabulary: An Introduction¹ 16 January 2010 Thomas Mercer-Hursh, Ph.D.

In reading about Object-Oriented Programming, one is likely to encounter many words with special meaning in an OO context. Sometimes, the meaning of these words may seem apparent from the context, since they are mostly special uses of existing English words, but at times their meaning may be unclear or the true meaning may be different from the meaning one might intuit in that particular context. In order to provide those starting out with OOABL² a reference by which they can check the meanings of such words, I have compiled the following list. This is not to be considered a substitute for systematic reading and study, but might help the newcomer to OO thinking to get started in their reading.

Abstract Method: A method which is defined only by a Signature, i.e., its parameters and return type, but no code to implement the method. A Class containing an Abstract Method cannot itself be instantiated³. Instead, a Class which Inherits from the Class with the Abstract method must provide a Concrete Implementation if it is to be Instantiated. Any Class which is to be Instantiated directly must have resolved all Abstract Methods in its Class Hierarchy.

Abstraction: Knowledge⁴, behavior⁵, or relationships not associated with a specific instance or implementation.

Aggregation: A relationship between one Object and one or more other Objects in which one object “contains” the other objects or which one would describe by the phrase “part of”, i.e., some group of teachers is “part of” a school. Aggregation differs from Composition in that in Aggregation, the contained Object has the potential to stand alone, i.e., if the school is disbanded, the teachers still exist.

Association: A structural relationship between two objects indicating that they are or can be used together.

Cast: A programming language method for converting data types, e.g., casting an object passed as a Superclass or Interface into the actual specific Class of the original object.

¹ First in a series of short whitepapers on OO analysis, design, and programming.

² Statements about what is and is not in ABL are based on version 10.2B.

³ This is true in OOABL, but may not be true of all OO languages.

⁴ In this vocabulary, the term “knowledge” is used in preference to “data” because of a possible tendency for people to equate data with something stored in a database. In a Responsibility, an Object is expected to “know” something, but that may not equate to a data element. E.g., it might be a computed value.

⁵ In modeling terminology, “behaviors” are represented by “operations”, but behavior is preferred in this vocabulary since it is more general in meaning and less tied to a specific type of implementation.

Class: The definition of a type, including state and behavior of that type. Knowledge includes data members and properties of the Class. Behavior includes methods and events. In modeling terms, one can think of a Class as a set of Objects, i.e., the common knowledge elements, behavior, and relationships of those Objects independent of any specific instance data or state.

Class Hierarchy: All classes in a particular Generalization taken as a whole. This term is also used to refer to all Classes in a particular Generalization from which one specific Leaf Class inherits, i.e., all those which contribute to the Responsibilities of that Class, i.e any Classes from which the Class inherits, including all parents of those parents until there are no further parents⁶, and all Interfaces which the Class Implements.

Client: Any entity that invokes the Responsibilities of another entity. The entities may be Objects, Packages, or Subsystems.

Cohesion: The degree to which the Responsibilities or aspects of a module or component form a meaningful unit. High cohesion implies a high level of interconnectedness around a single common purpose. Low cohesion implies loose connection and the likelihood of multiple purposes or Responsibilities.

Composition: A relationship between one Object and one or more other Objects in which one object “contains” the other objects or which one would describe by the phrase “part of”, i.e., some departments are “part of” a university. Composition differs from Aggregation in that in Composition, destroying the container object also destroys the contained Objects, if a university is disbanded, the departments no longer exist.

Concrete: A specific Instance or Implementation, i.e., not Abstract.

Coupling: The degree of mutual interdependence between entities. Generally, one wishes the Coupling between entities to be as small as possible given the Collaboration of the two entities.

Collaboration: Two or more entities which cooperate to solve a problem.

Contract: The specification of the Responsibility of a Class or subsystem. This specification manifests itself as the externally visible knowledge and behaviors which a Class makes available for the use of other Classes (see Signature). Like the conditions and obligations of a business contract, a Class is free to alter its internal knowledge and behavior freely as long as it does not disturb the specification of the way in which it interacts with other Classes. Thus, as long as the Contract remains unchanged, no Client Class needs to be modified. See Design By Contract.

Delegation: One object relying on another to implement a part of its overall functionality. A Delegate is created as a separate object to further separation of concerns, particularly when the main object is very complex and the Delegate has variations appropriate for Specialization. Once separated, the Delegate and the original Object are peers, each with their own separate sphere of Responsibility. It is important that this separation exist in the problem space, i.e., the Delegate should be an intrinsic decomposition recognizable in the problem space, not merely an arbitrary

⁶ In OOABL all classes ultimately inherit implicitly from `Progress.Lang.Object`.

cluster of knowledge and behaviors. The container object, i.e., the one that has Delegated some of its behavior, may expose Methods by which that Delegated behavior can be accessed so that Clients of that object need not be aware of the Delegate.

Dependency: A relationship between Objects in which one Object requires another in order to function.

Design By Contract®⁷ (DbC): Design by defining formal, precise and verifiable interface specifications for software components, which extend the ordinary definition of abstract knowledge types with preconditions, postconditions and invariants. These specifications are referred to as Contracts.

Derived Class: See Subclass.

Encapsulation: Localizing very closely related Responsibilities in a Class and hiding the implementation of those Responsibilities behind the Contract for that Class. The Class should include all of the knowledge and behavior needed to implement its Responsibilities, but should expose only such of that as is necessary for other objects to interact with it. See Separation of Concerns, Single Responsibility Principle, and Interface Segregation Principle in Object-Oriented Design Principles⁸.

Event: A message which announces a business-relevant change in state. In OOABL these are Strongly Typed.

Extension: Modifying the Contract of a Class by adding additional knowledge or behavior to the existing Contract, leaving the existing Contract unmodified. This is commonly accomplished by Subclassing the Class and putting new knowledge or behavior in the Subclass.

Generalization: Recognizing common knowledge and behavior in a group of classes and extracting that knowledge and behavior into a Superclass which is a parent to each existing class. Thus, each Subclass retains only its unique knowledge and behavior. When an object from a generalization is instantiated it is always a member of a leaf Subclass and possesses all the Responsibilities of every Superclass in a direct line of ascent. See Specialization and Inheritance.

Implement: Providing concrete behavior for a Method whose Signature has been defined in another Class in the Class Hierarchy. Any one Class can implement multiple Interfaces.

Inheritance: The mechanism to implement Generalization and Specialization. When one class (the Subclass) Inherits from another class (the Superclass) it acquires all of the knowledge and behavior of the Superclass, although it can elect to selectively override that behavior. Any specific Object in a Generalization hierarchy will have all of the knowledge and behavior of its own Class and all of the Classes in a direct line of ascent unless a method overrides a method higher in the hierarchy.

⁷ Because *Design by Contract* is a registered trademark of Eiffel Software in the United States, many developers refer to it as Programming by Contract, Contract Programming, or Contract-First development.

⁸ Mercer-Hursh, Thomas. *Object-Oriented Design Principles*.

Instance: An Object with a unique identity. Only one instance with a given identity can exist at any given time.

Instantiation: The process of creating a specific instance of an object.

Interface: A Class which defines a Contract in abstract terms so that this Contract can be Implemented by multiple Classes. This provides a uniform component to the Contract of all Classes which Implement the Interface. Any Class Implementing an Interface can be substituted for any other Class Implementing that Interface wherever a parameter is defined as being the type of the Interface rather than some concrete Class.

Invariants: A fundamental characteristic that is unlikely to change in the context. “Context” can refer to the problem space, e.g., normal requirements changes, or a broad class of similar subject matters, e.g., all GUI displays all use the same basic mechanisms regardless of the semantics of what is displayed. Extracting invariants is just another form of abstraction; one strives to find a level of abstraction where the details no longer matter.⁹

Leaf Class: In a Generalization hierarchy, one of the terminal nodes, i.e., the Subclass which will actually be Instantiated in the running system. It is generally considered poor design to instantiate a Superclass, although the type of the Superclass may be specified as the type of a parameter when any of the Subclasses can be supplied.

Member: An element of a Class definition which defines a knowledge or behavior Responsibility. Knowledge Members are variables and properties. Behavior Members are Methods and Events.

Message: In the context of modeling, a communication from a Class that identifies some state or event. In code, this modeling message manifests itself as a Member of the Class, i.e. Methods, Properties, and Events. I.e., in coding, Messages are essentially the reverse of their intent in modeling. In modeling, the object is communicating something about itself to whomever might be interested, but in code many Messages become instructions to do something. When speaking of Objects, a Message is often more loosely interpreted to be a communication of knowledge or instructions between two Objects, which is closer to the modeling intent.

Method: The implementation of a single unit of behavior Responsibility in a Class. A Method is similar to an Internal Procedure in non-OO programming except that it may have a return value, has more features, and its signature is checked against calls at compile time.

Multiplicity: In a Relationship between two Classes, the count of the number of each type of Class which may be at the end of the Relationship. While any number is possible, most multiplicities are stated in terms of 0 (no instances), 1 (exactly one instance), and * (many instances). Often, a range is specified using two dots, e.g., 0..1 (no instances or one instance, an optional Relationship), 1..* (one or more instances). Both ends of a Relationship have their own Multiplicity. E.g., the Relationship between Car and Owner would be 0..* on the Car end since one Owner can own no cars or many cars and either 1..* or 0..* on the Owner end depending on whether the context allowed for a Car to have no Owners.

⁹ Definition adapted from H.S. Lahman, personal communication.

Object: In modeling terms, an object is a abstraction of an entity in the problem space, i.e., one object corresponds to one entity. Objects with the same Responsibilities that Abstract similar entities get collected as a Class or in Class sets such as a Class Hierarchy. In common usage, one often hears an Object considered as a specific runtime instance of a Class, but one should remember that the purpose of the Object is to correspond to a specific entity in the problem space and that the Class is our effort to create one definition which covers multiple such specific entities.

Overloading: A polymorphism in which the same name is used for different, but analogous Abstractions. In practice, this means defining multiple instances of a Method with the same name, but with different signatures, i.e., different numbers or types of arguments. In use, the correct Method definition is chosen according to the match between the signature of the call and the signature of the available Overloaded Methods with that name.

Override: When a Subclass provides a matching definition for a method in its Superclass, the method in the Subclass overrides the definition in the Superclass. This Override may either totally replace the definition in the Superclass or the method of the Subclass may invoke the logic of the Superclass as part of its operation.

Package: A collection of closely related and cooperating Classes.

Polymorphism: Can mean several different, but related things. Inclusion or Subtype Polymorphism is defining a Method in a Superclass and then providing different implementations of that Method in various Subclasses. Referencing the Method in the Superclass will call the implementation of the current Subclass instance, thus providing different behaviors for the same Method depending on which Subclass is currently in existence without having to recognize the Type of the Subclass. Overload Polymorphism is provided by Overloading a Method. Operator Overloading, which does not exist in OOABL, is providing alternate, type-specific behaviors for primitive operators such as + and -.

Programming by Contract: See Design by Contract.

Property: An attribute or data Member of a Class. In OOABL these are characterized by having a usage syntax which gives the appearance of public access, but includes internal features to control read-only or write-only access and to provide any desired code to map between the external appearance and the internal implementation.

Realization: A component providing a Concrete Implementation for an Interface.

Responsibility: An obligation to know or do something. A coherent function that a Class performs; sometimes referred to in terms of a “reason to change”. For example, if a given display needs to change, it is likely that it needs to change in terms of format or it needs to change in terms of content, but not both. Thus, format and content are separate Responsibilities.

Signature: The visible characteristics of a Method, i.e., its returned Type and parameters.

Specialization: When one recognizes that an existing class has some different knowledge or behavior in specific rôles, contexts, or states, one can then create a relationship in which the existing class is a parent or Superclass to a set of classes (the Subclasses), each of which implements the

knowledge and behavior appropriate to a particular rôle, context, or state. See Generalization and Inheritance. Many authors do not use the word Specialization, but refer to all instances as Generalization.

Static: A Static data or behavior member is defined as being scoped to the class rather than to the individual Instance, i.e., Object. Thus, all Instances of that Class have access to the same member, as do all Clients of those Objects. The Class containing a Static member does not have to be Instantiated prior to the use of the Static member and, once a Static member has been used, it remains available during the balance of the session.

Static Class: A Class containing all Static Members¹⁰.

Strongly Typed: Types defined by the developer are enforced by the compiler for conformance to a specific Class definition. Among other benefits, this allows checking many possible errors at compile time instead of discovering them at run time.

Subclass: A Specialization of a Superclass, i.e., a Class that inherits from a parent Superclass, typically containing unique knowledge or behavior separate from that in the Superclass and different from that contained in other Subclasses of the same Superclass.

Subsystem: A Package or collection of Packages which together represent the total set of Responsibilities for some area. Subsystems can be at many levels or sizes and can be either vertical, i.e., all Responsibilities from database interface to UI for a particular area of functionality, or horizontal, i.e., all Responsibilities related to some layer, e.g., data access.

Superclass: A Generalization of the common knowledge and behavior in some number of Subclasses, i.e., a Class into which the common knowledge and behaviors has been placed so that a single Class can be Inherited by the Subclasses and serve as a container for that common knowledge and behavior.

Type: The structure and semantics of an Object independent of its Implementation. A Class defines the Type of the Objects created from it.

This basic vocabulary is, of course, the smallest of introductions to the concepts of Object-Oriented Programming. The reader is encouraged in additional reading to provide depth and context to these concepts.

¹⁰ In OOABL, one cannot define the class itself as Static, but in practice a class with all Static Members functions as a Static Class.