D

A glossary of object technology

This glossary provides brief definitions of the principal terms of object technology, discussed in detail in the rest of this book. *Italics font* in a definition marks a term or phrase, other than the ubiquitous "class" and "object", that is itself the subject of another definition.

Abstract class

See *deferred class*.

Abstract data type (ADT)

A set of mathematical elements specified by listing the functions applicable to all these elements and the formal properties of these functions.

Abstract object

An element of an *ADT*.

Ancestor (of a class)

The class itself, or one of its direct or indirect parents.

Assertion

A formal condition describing the semantic properties of software elements, especially routines and loops. Used in expressing *contracts*. Assertions include in particular *preconditions*, *postconditions*, *class invariants* and *loop invariants*.

Assignment attempt

An operation that conditionally attaches an object to a reference, only if the object's type *conforms* to the type declared for the corresponding *entity*.

Asynchronous call

A call which lets its caller proceed before it completes. Antonym: *synchronous call*.

Attribute

The description of a *field* present in all the instances of a class. Along with the *routine*, one of the two forms of *feature*.

Behavior class

A class, usually *deferred*, describing a set of adaptable behaviors through *effective routines* relying on some components (usually *deferred features*) that may be *redeclared* to capture specific variants of the general behaviors.

Class
A partially or totally implemented abstract data type. Serves both as a *module* and as a *type* (or type pattern if the class is *generic*.)

Class invariant
An *assertion* which must be satisfied on creation of every instance of a class, and preserved by every exported routine of the class, so that it will be satisfied by all instances of the class whenever they are externally observable.

Client
A class that uses the features of another, its *supplier*, on the basis of the supplier's interface specification (*contract*).

Cluster
A group of related classes or, recursively, of related clusters.

Component
See *reusable software component*.

Concurrent
Able to use two or more *processors*. Antonym: *sequential*.

Conformance
A relation between types. A type conforms to another if it is derived from it by inheritance.

Constrained genericity
A form of *genericity* where a formal generic parameter represents not an arbitrary type, but one that is required to *conform* to a certain type, known as the constraint. See *constrained genericity*.

Container data structure
An *object* whose primary use is to provide access to a number of other objects. Examples include lists, queues, stacks, arrays.

Contract
The set of precise conditions that govern the relations between a *supplier* class and its *clients*. The contract for a class includes individual contracts for the exported routines of the class, represented by preconditions and postconditions, and the global class properties, represented by the class invariant. See also *Design by Contract*.

Contravariance
The policy allowing a feature *redeclaration* to change the *signature* so that a new result type will *conform* to the original but the original argument types conform to the new. See also: *covariance*, *novariance*.

Covariance
The policy allowing a feature *redeclaration* to change the *signature* so that the new types of both arguments and result *conform* to the originals. See also: *contravariance*, *novariance*.

Current object (or: current instance)
During the execution of an object-oriented software system, the target of the most recently started routine call.

Defensive programming
A technique of fighting potential errors by making every module check for many possible consistency conditions, even if this causes redundancy of checks performed by *clients* and *suppliers*. Contradicts *Design by Contract*.

Deferred class
A class which has at least one *deferred feature*. Antonym: *effective class*.

Deferred feature
A feature which, in a certain class, has a specification but no implementation. May be declared as deferred in the class itself, or inherited as deferred and not *effected* in the class. Antonym: *effective feature*.

Descendant (of a class)
The class itself, or one of its direct or indirect heirs.

Design by Contract
A method of software construction that designs the components of a *system* so that they will cooperate on the basis of precisely defined *contracts*. See also: *defensive programming*.

Direct instance (of a class)
An object built according to the mold defined by the class.

Dynamic
Occurring during the execution of a *system*. See also *run time*. Antonym: *static*.

Dynamic binding
The guarantee that every execution of an operation will select the correct version of the operation, based on the type of the operation's target.

Dynamic typing
The policy whereby applicability of operations to their target objects is only checked at run time, prior to executing each operation.

Effect
A class effects a feature if it inherits it in *deferred* form and provides an *effecting* for that feature.

Effecting
A *redeclaration* which provides an implementation (as *attribute* or *routine*) of a feature inherited in *deferred* form.

Effective class
A class which only has *effective features* (that is to say, does not introduce any *deferred feature*, and, if it inherits any deferred feature, effects it). Antonym: *deferred class*.

Effective feature
A feature declared with an implementation — either as a routine which is not *deferred*, or as an *attribute*. Antonym: *deferred feature*.

Encapsulation
See *information hiding*.

Entity
A name in the software text that denotes a run-time value (*object* or *reference*).

Event-driven computation
A style of software construction where developers define the control structure by listing possible external events and the system's response to each of them, rather than by specifying a pre-ordained sequence of steps.

Exception
The inability of a routine to achieve its *contract* through one of its possible strategies. May result in particular from a *failure* of a routine called by the original routine. Will be treated as *resumption, organized panic* or *false alarm*.

Exporting a feature
Making the feature available to *clients*. Exports may be selective (to specified classes only) or general.

Extendibility
The ability of a software system to be changed easily in response to different choices of requirements, architecture, algorithms or data structures.

Failure
The inability of a routine's execution to fulfill the routine's *contract*. Must trigger an *exception*.

False alarm
Along with *resumption* and *organized panic*, one of the three possible responses to an *exception*; resumes the execution of the current strategy, possibly after taking some corrective action.

Feature renaming
The attribution, by a class, of a new name to an inherited feature, not changing any other property. See also *redeclaration*.

Field
One of the values making up an *object*.

Function
A routine which returns a result. (The other form of routine is the *procedure*.)

Garbage collection
A facility provided by the *runtime* to recycle the memory space used by objects that have become useless. Garbage collection is automatic, that is to say does not require any change to the text of the *systems* whose objects are being recycled.

Generalization
The process of turning specialized program elements into general-purpose, reusable software components.

Generating class
Same as *generator*.

Generator (of an object)
The class of which the object is a *direct instance*.

Generic class
A class having formal parameters representing types. Such a class will yield a type only through *generic derivation*.

Generic derivation
The process of providing a type for each formal generic parameter of a *generic class*, yielding a type as a result.

Genericity
The support, by a software notation, for type-parameterized modules; specifically, in an O-O notation, for *generic classes*. Can be *unconstrained* or *constrained*.

Heir (of a class)
A class that inherits from the given class. Antonym: *parent*.

Identity
See *object identity*.

Information hiding
The ability to prevent certain aspects of a class from being accessible to its clients, through an explicit *exporting* policy and through reliance on the *short form* as the primary vehicle for class documentation.

Inheritance
A mechanism whereby a class is defined in reference to others, adding all their features to its own.

Instance (of a class)
An object built according to the mold defined by the class or any one of its proper descendants. See also *direct instance, proper descendant*, *generator*.

Instance variable
Smalltalk term for attribute.

Interface (of a class)
See *contract, abstract data type*.

Invariant
See *class invariant, loop invariant*.

Iterator
A control structure describing preordained sequencing of some actions but not defining the actions themselves. Iterators often apply to data structures, such as an iterator describing the traversal of a list or a tree.

Loop invariant
An *assertion* which must be satisfied prior to the first execution of a loop, and preserved by every iteration, so that it will hold on loop termination.

Loop variant
An integer expression which must be non-negative prior to the first execution of a loop, and decreased by every iteration, so that it will garantee loop termination.

Message
Routine call.

Metaclass

A class whose instances are classes themselves.

Method

Smalltalk term for routine.

Module

A unit of software decomposition. In the object-oriented approach, classes provide the basic form of module.

Multiple inheritance

The unrestricted form of inheritance, whereby a class may have any number of parents. Antonym: *single inheritance*.

Non-separate

Antonym of *separate*.

Novariance

The policy allowing prohibiting any feature *redeclaration* from changing the *signature*. See also: *contravariance*, *covariance*.

Object

A run-time data structure made of zero or more values, called *fields*, and serving as the computer representation of an *abstract object*. Every object is an instance of some class.

Object identity

A property that uniquely identifies an object independently of its current contents (*fields*).

Object-oriented

Built from *classes*, *assertions*, *genericity*, *inheritance*, *polymorphism* and *dynamic binding*.

Object-oriented analysis

The application of *object-oriented* concepts to the modeling of problems and systems from both software and non-software domains.

Object-oriented database

A repository of *persistent objects*, permitting their storage and retrieval on the basis of *object-oriented* concepts, and supporting database properties such as concurrent access, locking and transactions.

Object-oriented design

The process of building the architecture of *systems* through *object-oriented* concepts.

Object-oriented implementation

The process of building executable software systems through *object-oriented* concepts. Differs from *object-oriented design* primarily by the level of abstraction.

Organized panic

Along with *resumption* and *false alarm*, one of the three possible responses to an *exception*; abandons the execution of the current strategy, triggering an exception in the caller, after restoring the *class invariant* for the *current object*.

Overloading

The ability to let a feature name denote two or more operations.

Package

A module of non-object-oriented languages such as Ada, providing encapsulation of a set of variables and routines.

Parallel

See *concurrent*.

Parameterized class

See *generic class*.

Parent (of a class)

A class from which the given class inherits. Antonym: *heir*.

Persistence

The ability of a software development environment or language to make objects *persistent* and support the retrieval of persistent objects for use by systems.

Persistent object

An object that (through storage in a file or database or transmission across a network) survives executions of systems that create or manipulate it. Antonym: *transient object*.

Polymorphic data structure

A *container data structure* hosting objects of two or more possible types.

Polymorphism

The ability for an element of the software text to denote, at run time, objects of two or more possible types.

Postcondition

An *assertion* attached to a routine, which must be guaranteed by the routine's body on return from any call to the routine if the *precondition* was satisfied on entry. Part of the *contract* governing the routine.

Precondition

An *assertion* attached to a routine, which must be guaranteed by every client prior to any call to the routine. Part of the *contract* governing the routine.

Predicate

See *assertion*.

Procedure

A routine which does not return a result. (The other form of routine is the *function*.)

Processor

A mechanism providing a single thread of computation. May be a physical device, such as the CPU of a computer, or a software device, such as a task or thread of an operating system.

Program

See *system*.

Proper ancestor (of a class)
A direct or indirect parent of the class.

Proper descendant (of a class)
A direct or indirect heir of the class.

Redeclaration
A feature declaration which, instead of introducing a new feature, adapts some properties (such as the *signature*, *precondition*, *postcondition*, implementation, *deferred/effective* status, but not the name) of a feature inherited from a *parent*. A redeclaration may be a *redefinition* or an *effecting*. See also *feature renaming*.

Redefinition
A *redeclaration* which is not an *effecting*, that is to say, changes some properties of a feature inherited as effective, or changes the specification of a feature inherited as *deferred* while leaving it deferred.

Reference
A run-time value that uniquely identifies an object.

Renaming
See *feature renaming*.

Retrying
Along with *organized panic* and *false alarm*, one of the three possible responses to an *exception*; tries a new strategy for achieving the routine's *contract*.

Reusability
The ability of a software development method to yield software elements that can be used in many different applications, and to support a software development process relying on pre-existing *reusable software components*.

Reusable software component
An element of software that can be used by many different applications.

Reversible development
A software development process that lets insights gained in later phases affect the results obtained in earlier phases. Normally part of a *seamless development* process.

Root class
The *generator* of a system's *root object*. Executing the system means creating an instance of the root class (the root object), and calling a creation procedure on that instance.

Root object
The first object created in the execution of a system.

Routine
A computation defined in a class, and applicable to the instances of that class. Along with the *attribute*, one of the two forms of *feature*.

Runtime (noun, one word)
Any set of facilities supporting the execution of systems. See also next entry.

Run time (noun, two words)
The time when a *system* is being executed. Also used as an adjective, with a hyphen, as in "the run-time value of an *entity*". See also *dynamic* and previous entry.

Schema evolution
Change to one or more classes of which some *persistent* instances exist.

Seamless development
A software development process which uses a uniform method and notation throughout all activities, such as problem modeling and analysis, design, implementation and maintenance. See also *reversible development*.

Selective export
See *exporting a feature*.

Separate
Handled by a different *processor*. Antonym: non-separate.

Sequential
Running on only one *processor*. Antonym: *concurrent*.

Short form (of a class)
A form of class documentation generated from the class text, showing only interface properties of the class. The short form documents the *contract* attached to the class and the underlying *abstract data type*.

Signature (of a feature)
The type part of the feature's specification. For an attribute or a function, includes the result type; for a routine, includes the number of arguments and the type of each.

Single inheritance
A restricted form of inheritance whereby each class may have at most one parent. Antonym: *multiple inheritance*.

Software component
See *reusable software component*.

Specification (of a class)
The *short form* of the class.

Specification (of a feature)
The properties of a feature that are relevant to a client. Includes the name, *signature*, header comment and *contract* of the feature.

Subcontract
The ability of a class to let some proper *descendant* handle some of its feature calls, thanks to *redeclaration* and *dynamic binding*.

Supplier
A class that provides another, its *client*, with features to be used through an interface specification (*contract*).

Static
Applying to the text of a *system*, not to a particular execution. Antonym: *dynamic*.

Static binding
The premature choice of operation variant, resulting in possibly wrong results and (in favorable cases) run-time system crash.

Static typing
The ability to check, on the basis of the software text alone, that no execution of a system will ever try to apply to an object an operation that is not applicable to that object.

Synchronous call
A call which forces the caller to wait until it completes. Antonym: *asynchronous call*.

System
A set of classes that can be assembled to produce an executable result.

Template
C++ term for *generic class* (for *unconstrained genericity* only).

Traitor
A reference to a *separate* object, associated in the software text with an *entity* that is declared as non-separate.

Transient object
An object that exists only during the execution of the system that creates it. Antonym: *persistent object*.

Type
The description of a set of objects equipped with certain operations. In the object-oriented approach every type is based on a class.

Type checking, typing
See *static typing*, *dynamic typing*.

Unconstrained genericity
A form of *genericity* where a formal generic parameter represents an arbitrary type. See *constrained genericity*.

Variant
See *loop variant*.