UML 2.0-based Terms and Definitions

abstract class
A class that cannot be directly instantiated. Contrast: *concrete class*.
abstraction
The result of emphasizing certain features of a thing while de-emphasizing other features that are not relative. An abstraction is defined relative to the perspective of the viewer.
action
A fundamental unit of behavior specification that represents some transformation or processing in the modeled system, be it a computer system or a real-world system. Actions are contained in activities, which provide their
context. See: *activity*.
action sequence
An expression that resolves to a sequence of actions.
action state
A state that represents the execution of an atomic action, typically the invocation of an operation.
activation
The initiation of an action execution.
active class
A class whose instances are active objects. See: *active object.*
active object
An object that may execute its own behavior without requiring method invocation. This is sometimes referred to as "the object having its own thread of control." The points at which an active object responds to communications from other objects are determined solely by the behavior of the active object and not by the invoking object. This implies that an active object is both autonomous and interactive to some degree. See: *active class, thread.*
activity
A specification of parameterized behavior that is expressed as a flow of execution via a sequencing of subordinate units (whose primitive elements are individual actions). See *actions*.
activity diagram
A diagram that depicts behavior using a control and data-flow model.
actor
A construct that is employed in use cases that define a role that a user or any other system plays when interacting with the system under consideration. It is a type of entity that interacts, but which is itself external to the subject. Actors may represent human users, external hardware, or other subjects. An actor does not necessarily represent a specific physical entity. For instance, a single physical entity may play the role of several different actors and, conversely, a given actor may be played by multiple physical entities.
actual parameter
Synonym: *argument*.
aggregate
A class that represents the "whole" in an aggregation (whole-part) relationship. See: *aggregation*.
aggregation
A special form of association that specifies a whole-part relationship between the aggregate (whole) and a component part. See: *composition*.
analysis
The phase of the system development process whose primary purpose is to formulate a model of the problem domain that is independent of implementation considerations. Analysis focuses on what to do; design focuses on how to do it. Contrast: *design*.
analysis time
Refers to something that occurs during an analysis phase of the software development process. See: *design time, modeling time.*
architecture

A coherent organizational structure of a system. An architecture can be hierarchically decomposed into parts that interact via links that connect parts, and constraints for assembling parts.
argument
A binding for a parameter that is resolved later. An independent variable.
artifact
A physical piece of information that is used or produced by a development process. Examples of Artifacts include models, source files, scripts, and binary executable files. An artifact may constitute the implementation of a deployable component. Synonym: *product*. Contrast: *component*.
association
A relationship that may occur between instances of classifiers.
association class
A model element that has both association and class properties. An association class can be seen as an association that also has class properties, or as a class that also has association properties.
association end
The endpoint of an association, which connects the association to a classifier.
attribute
A structural feature of a classifier that characterizes instances of the classifier. An attribute relates an instance of a classifier to a value or values through a named relationship.
auxiliary class
A stereotyped class that supports another more central or fundamental class, typically by implementing secondary logic or control flow. Auxiliary classes are typically used together with focus classes, and are particularly useful for specifying the secondary business logic or control flow of components during design. See also: *focus*.
behavior
The observable effects of an operation or event, including its results. It specifies the computation that generates the effects of the behavioral feature. The description of a behavior can take a number of forms: interaction, state machine, activity, or procedure (a set of actions).
behavior diagram
A form of diagram that depict behavioral features.
behavioral feature
A dynamic feature of a model element, such as an operation or method.
behavioral model aspect
A model aspect that emphasizes the behavior of the instances in a system, including their methods, collaborations, and state histories.
binary association
An association between two classes. A special case of an n-ary association.
binding
The creation of a model element from a template by supplying arguments for the parameters of the template.
boolean
An enumeration whose values are true and false.
boolean expression
An expression that evaluates to a boolean value.
cardinality
The number of elements in a set. Contrast: *multiplicity*.
child
In a generalization relationship, the specialization of another element, the parent. See: *subclass*, *subtype*. Contrast: *parent*.
call
An action state that invokes an operation on a classifier.
class
A classifier that describes of a set of objects that share the same specifications of features, constraints, and semantics.

classifier
A collection of instances that have something in common. A classifier can have features that characterize its instances. Classifiers include interfaces, classes, datatypes, and components.
classification
The assignment of an instance to a classifier. See *dynamic classification*, *multiple classification* and *static classification*.
class diagram
A diagram that shows a collection of declarative (static) model elements, such as classes, types, and their contents and relationships.
client
A classifier that requests a service from another classifier. Contrast: *supplier.*
collaboration
The specification of how an operation or classifier, such as a use case, is realized by a set of classifiers and associations playing specific roles used in a specific way. The collaboration defines an interaction. See: *interaction*.
collaboration occurrence
A particular use of a collaboration to explain the relationships between the parts of a classifier or the properties of an operation. It may also be used to indicate how a collaboration represents a classifier or an operation. A collaboration occurrence indicates a set of roles and connectors that cooperate within the classifier or operation according to a given collaboration, indicated by the type of the collaboration occurrence. There may be multiple occurrences of a given collaboration within a classifier or operation, each involving a different set of roles and connectors. A given role or connector may be involved in multiple occurrences of the same or different collaborations. See: *collaboration*.
communication diagram
A diagram that focuses on the interaction between lifelines where the architecture of the internal structure and how this corresponds with the message passing is central. The sequencing of messages is given through a sequence numbering scheme. Sequence diagrams and communication diagrams express similar information, but show it in different ways. See: *sequence diagram*.
compile time
Refers to something that occurs during the compilation of a software module. See: *modeling time, run time*.
component
A modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. A component defines its behavior in terms of provided and required interfaces. As such, a component serves as a type, whose conformance is defined by these provided and required interfaces (encompassing both their static as well as dynamic semantics).
component diagram
A diagram that shows the organizations and dependencies among components.
composite
A class that is related to one or more classes by a composition relationship. See: *composition*.
composite aggregation
Synonym: *composition*.
composite state
A state that consists of either concurrent (orthogonal) substates or sequential (disjoint) substates. See: *substate*.
composite structure diagram
A diagram that depicts the internal structure of a classifier, including the interaction points of the classifier to other parts of the system. It shows the configuration of parts that jointly perform the behavior of the containing classifier. The architecture diagram specifies a set of instances playing parts (roles), as well as their required relationships given in a particular context.
composition
A form of aggregation which requires that a part instance be included in at most one composite at a time, and that the composite object is responsible for the creation and destruction of the parts. Composition may be recursive. Synonym: *composite aggregation*.

concrete class
A class that can be directly instantiated. Contrast: *abstract class.*
concurrency
The occurrence of two or more activities during the same time interval. Concurrency can be achieved by
interleaving or simultaneously executing two or more threads. See: *thread.*
concurrent substate
A substate that can be held simultaneously with other substates contained in the same composite state. See: *composite state.* Contrast: *disjoint substate.*
connectable element
An abstract metaclass representing model elements which may be linked via connector. See: *connector*.
connector
A link that enables communication between two or more instances. The link may be realized by something as simple as a pointer or by something as complex as a network connection.
constraint
A semantic condition or restriction. It can be expressed in natural language text, mathematically formal notation, or in a machine-readable language for the purpose of declaring some of the semantics of a model element.
container
1. An instance that exists to contain other instances, and that provides operations to access or iterate over its contents. (for example, arrays, lists, sets).
2. A component that exists to contain other components.
containment hierarchy
A namespace hierarchy consisting of model elements, and the containment relationships that exist between them. A containment hierarchy forms a graph.
context
A view of a set of related modeling elements for a particular purpose, such as specifying an operation.
data type
A type whose values have no identity (i.e., they are pure values). Data types include primitive built-in types (such as integer and string) as well as enumeration types.
delegation
The ability of an object to issue a message to another object in response to a message. Delegation can be used as an alternative to inheritance. Contrast: *inheritance*.
dependency
A relationship between two modeling elements, in which a change to one modeling element (the independent element) will affect the other modeling element (the dependent element).
deployment diagram
A diagram that depicts the execution architecture of systems. It represents system artifacts as nodes, which are connected through communication paths to create network systems of arbitrary complexity. Nodes are typically defined in a nested manner, and represent either hardware devices or software execution environments. See: *component diagrams*.
derived element
A model element that can be computed from another element, but that is shown for clarity or that is included for design purposes even though it adds no semantic information.
design
The phase of the system development process whose primary purpose is to decide how the system will be implemented. During design strategic and tactical decisions are made to meet the required functional and quality requirements of a system.
design time
Refers to something that occurs during a design phase of the system development process. See: *modeling time*. Contrast: *analysis time*.
development process
A set of partially ordered steps performed for a given purpose during system development, such as constructing models or implementing models.

diagram
A graphical presentation of a collection of model elements, most often rendered as a connected graph of arcs (relationships) and vertices (other model elements). UML supports the diagrams listed in Appendix A.
disjoint substate
A substate that cannot be held simultaneously with other substates contained in the same composite state. See: *composite state*. Contrast: *concurrent substate*.
distribution unit
A set of objects or components that are allocated to a process or a processor as a group. A distribution unit can be represented by a run-time composite or an aggregate.
domain
An area of knowledge or activity characterized by a set of concepts and terminology understood by practitioners in that area.
dynamic classification
The assignment of an instance from one classifier to another. Contrast: *multiple classification, static classification*.
element
A constituent of a model.
entry action
An action that a method executes when an object enters a state in a state machine regardless of the transition taken to reach that state.
enumeration
A data type whose instances a list of named values. For example, RGBColor = {red, green, blue}. Boolean is a predefined enumeration with values from the set {false, true}.
event
The specification of a significant occurrence that has a location in time and space and can cause the execution of an associated behavior. In the context of state diagrams, an event is an occurrence that can trigger a transition.
exception
A special kind of signal, typically used to signal fault situations. The sender of the exception aborts execution and execution resumes with the receiver of the exception, which may be the sender itself. The receiver of an exception is determined implicitly by the interaction sequence during execution; it is not explicitly specified.
execution occurrence
A unit of behavior within the lifeline as represented on an interaction diagram.
exit action
An action that a method executes when an object exits a state in a state machine regardless of the transition taken to exit that state.
export
In the context of packages, to make an element visible outside its enclosing namespace. See: *visibility.* Contrast: *export* [OMA], *import*.
expression
A string that evaluates to a value of a particular type. For example, the expression "(7 + 5 * 3)" evaluates to a value of type number.
extend
A relationship from an extension use case to a base use case, specifying how the behavior defined for the extension use case augments (subject to conditions specified in the extension) the behavior defined for the base use case. The behavior is inserted at the location defined by the extension point in the base use case. The base use case does not depend on performing the behavior of the extension use case. See *extension point*, *include*.
extension
An aggregation that is used to indicate that the properties of a metaclass are extended through a stereotype, and that gives the ability to flexibly add and remove stereotypes from classes.
facade
A stereotyped package containing only references to model elements owned by another package. It is used to provide a 'public view' of some of the contents of a package.

feature
A property, such as an operation or attribute, that characterizes the instances of a classifier.
final state
A special kind of state signifying that the enclosing composite state or the entire state machine is completed.
fire
To execute a state transition. See: *transition*.
focus class
A stereotyped class that defines the core logic or control flow for one or more auxiliary classes that support it. Focus classes are typically used together with one or more auxiliary classes, and are particularly useful for specifying the core business logic or control flow of components during design. See also: *auxiliary class*.
focus of control
A symbol on a sequence diagram that shows the period of time during which an object is performing an action, either directly or through a subordinate procedure.
formal parameter
Synonym: *parameter*.
framework
A stereotyped package that contains model elements which specify a reusable architecture for all or part of a system. Frameworks typically include classes, patterns or templates. When frameworks are specialized for an application domain, they are sometimes referred to as application frameworks. See: *pattern*.
generalizable element
A model element that may participate in a generalization relationship. See: *generalization*.
generalization
A taxonomic relationship between a more general classifier and a more specific classifier. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier indirectly has features of the more general classifier. See: *inheritance.*
guard condition
A condition that must be satisfied in order to enable an associated transition to fire.
implementation
A definition of how something is constructed or computed. For example, a class is an implementation of a type, a method is an implementation of an operation.
implementation class
A stereotyped class that specifies the implementation of a class in some programming language (e.g., C++, Smalltalk, Java) in which an instance may not have more than one class. An Implementation class is said to realize a type if it provides all of the operations defined for the type with the same behavior as specified for the type's
operations. See also: *type*.
implementation inheritance
The inheritance of the implementation of a more general element. Includes inheritance of the interface. Contrast: *interface inheritance*.
import
In the context of packages, a dependency that shows the packages whose classes may be referenced within a given package (including packages recursively embedded within it). Contrast: *export*.
include
A relationship from a base use case to an inclusion use case, specifying how the behavior for the base use case contains the behavior of the inclusion use case. The behavior is included at the location which is defined in the base use case. The base use case depends on performing the behavior of the inclusion use case, but not on its structure (i.e., attributes or operations). See *extend*.
inheritance
The mechanism by which more specific elements incorporate structure and behavior of more general elements. See *generalization*.
initial state

A special kind of state signifying the source for a single transition to the default state of the composite state.
instance
An entity that has unique identity, a set of operations that can be applied to it, and state that stores the effects of the operations. See: *object*.
interaction
A specification of how stimuli are sent between instances to perform a specific task. The interaction is defined in the context of a collaboration. See *collaboration.*
interaction diagram
A generic term that applies to several types of diagrams that emphasize object interactions. These include
communication diagrams, sequence diagrams, and the interaction overview diagram.
interaction overview diagram
A diagram that depicts interactions through a variant of activity diagrams in a way that promotes overview of the control flow. It focuses on the overview of the flow of control where each node can be an interaction diagram.
interface
A named set of operations that characterize the behavior of an element.
interface inheritance
The inheritance of the interface of a more general element. Does not include inheritance of the implementation. Contrast: *implementation inheritance*.
internal transition
A transition signifying a response to an event without changing the state of an object.
layer
The organization of classifiers or packages at the same level of abstraction. A layer may represent a horizontal slice through an architecture, whereas a partition represents a vertical slice. Contrast: *partition*.
lifeline
A modeling element that represents an individual participant in an interaction. A lifeline represents only one interacting entity.
link
A semantic connection among a tuple of objects. An instance of an association. See: *association*.
link end
An instance of an association end. See: *association end*.
message
A specification of the conveyance of information from one instance to another, with the expectation that activity will ensue. A message may specify the raising of a signal or the call of an operation.
metaclass
A class whose instances are classes. Metaclasses are typically used to construct metamodels.
meta-metamodel
A model that defines the language for expressing a metamodel. The relationship between a meta-metamodel and a metamodel is analogous to the relationship between a metamodel and a model.
metamodel
A model that defines the language for expressing a model.
metaobject
A generic term for all metaentities in a metamodeling language. For example, metatypes, metaclasses, metaattributes, and metaassociations.
method
The implementation of an operation. It specifies the algorithm or procedure associated with an operation.
model
A representation of a set of modules of a process, system, or subject area along with a representation of data, activities, relationships, and constraints, generally developed for understanding, analysis, improvement, and/or replacement of the process.
model aspect

A dimension of modeling that emphasizes particular qualities of the metamodel. For example, the structural model aspect emphasizes the structural qualities of the metamodel.
model elaboration
The process of generating a repository type from a published model. Includes the generation of interfaces and implementations which allows repositories to be instantiated and populated based on, and in compliance with, the model elaborated.
model element
An element that is an abstraction drawn from the system being modeled. Contrast: *view element*.
model library
A stereotyped package that contains model elements that are intended to be reused by other packages. A model library differs from a profile in that a model library does not extend the metamodel using stereotypes and tagged definitions. A model library is analogous to a class library in some programming languages.
modeling time
Refers to something that occurs during a modeling phase of the system development process. It includes analysis time and design time. Usage note: When discussing object systems, it is often important to distinguish between modeling-time and run-time concerns. See: *analysis time, design time*. Contrast: *run time*.
module
A separate and distinct unit of a process, system, or subject area that may be used as a conceptual component of a model or a part in a system.
multiple classification
The assignment of an instance directly to more than one classifier at the same time. See: *static classification*,
dynamic classification.
multiple inheritance
A semantic variation of generalization in which a type may have more than one supertype. Contrast: *single inheritance*.
multiplicity
A specification of the range of allowable cardinalities that a set may assume. Multiplicity specifications may be given for association ends, parts within composites, repetitions, and other purposes. Essentially a multiplicity is a (possibly infinite) subset of the non-negative integers. Contrast: *cardinality*.
n-ary association
An association among three or more classes. Each instance of the association is an n-tuple of values from the respective classes. Contrast: *binary association*.
name
A string used to identify a model element.
namespace
A part of the model in which the names may be defined and used. Within a namespace, each name has a unique meaning. See: *name*.
node
A classifier that represents a run-time computational resource, which generally has at least memory and often processing capability. Run-time objects and components may reside on nodes.
note
An annotation attached to an element or a collection of elements. A note has no semantics. Contrast: *constraint*.
object
An instance of a class. See: *class, instance*.
object diagram
A diagram that encompasses objects and their relationships at a point in time. An object diagram may be considered a special case of a class diagram or a communication diagram. See: *class diagram, communication diagram.*
object flow state
A state in an activity diagram that represents the passing of an object from the output of actions in one state to the input of actions in another state.

object lifeline
A line in a sequence diagram that represents the existence of an object over a period of time. See: *sequence diagram*.
operation
A feature which declares a service that can be performed by instances of the classifier of which they are instances.
package
A general purpose mechanism for organizing elements into groups. Packages may be nested within other packages.
package diagram
A diagram that depicts how model elements are organized into packages and the dependencies among them, including package imports and package extensions.
parameter
An argument of a behavioral feature. A parameter specifies arguments that are passed into or out of an invocation of a behavioral element like an operation. A parameter's type restricts what values can be passed. Synonyms: *formal parameter*. Contrast: *argument*.
parameterized element
The descriptor for a class with one or more unbound parameters. Synonym: *template*.
parent
In a generalization relationship, the generalization of another element, the child. See: *subclass*, *subtype*. Contrast: *child*.
part
An element representing a set of instances that are owned by a containing classifier instance or role of a classifier. (See *role.)* Parts may be joined by attached connectors and specify configurations of linked instances to be created within an instance of the containing classifier.
participate
The connection of a model element to a relationship or to a reified relationship. For example, a class participates in an association, an actor participates in a use case.
partition
A grouping of any set of model elements based on a set of criteria.
1. activity diagram: A grouping of activity nodes and edges. Partitions divide the nodes and edge to constrain and show a view of the contained nodes. Partitions can share contents. They often correspond to organizational units in a business model. They may be used to allocate characteristics or resources among the nodes of an activity.
2. architecture: A set of related classifiers or packages at the same level of abstraction or across layers in a layered architecture. A partition represents a vertical slice through an architecture, whereas a layer represents a horizontal slice. Contrast: *layer*.
pattern
A template collaboration that describes the structure of a design pattern. UML patterns are more limited than those used by the design pattern community. In general, design patterns involve many non-structural aspects, such as heuristics for their use and usage trade-offs.
persistent object
An object that exists after the process or thread that created it has ceased to exist.
pin
A model element that represents the data values passed into a behavior upon its invocation as well as the data values returned from a behavior upon completion of its execution.
port
A feature of a classifier that specifies a distinct interaction point between that classifier and its environment or between the (behavior of the) classifier and its internal parts. Ports are connected to other ports through connectors through which requests can be made to invoke the behavioral features of a classifier.
postcondition
A constraint expresses a condition that must be true at the completion of an operation.
powertype
A classifier whose instances are also subclasses of another classifier. Power types, then, are metaclasses with an extra twist: the instances are also subclasses.

precondition
A constraint expresses a condition that must be true when an operation is invoked.
primitive type
A pre-defined data type without any relevant substructure (i.e., is not decomposable) such as an integer or a string. It may have an algebra and operations defined outside of UML, for example, mathematically.
procedure
A set of actions that may be attached as a unit to other parts of a model, for example, as the body of a method. Conceptually a procedure, when executed, takes a set of values as arguments and produces a set of values as results, as specified by the parameters of the procedure.
process
1. A heavyweight unit of concurrency and execution in an operating system. Contrast: *thread*, which includes heavyweight and lightweight processes. If necessary, an implementation distinction can be made using stereotypes.
2. A software development process—the steps and guidelines by which to develop a system.
3. To execute an algorithm or otherwise handle something dynamically.
profile
A stereotyped package that contains model elements that have been customized for a specific domain or purpose using extension mechanisms, such as stereotypes, tagged definitions and constraints. A profile may also specify model libraries on which it depends and the metamodel subset that it extends.
projection
A mapping from a set to a subset of it.
property
A named value denoting a characteristic of an element. A property has semantic impact. Certain properties are predefined in the UML; others may be user defined. See: *tagged value*.
pseudo-state
A vertex in a state machine that has the form of a state, but doesn't behave as a state. Pseudo-states include initial and history vertices.
physical system
1. The subject of a model.
2. A collection of connected physical units, which can include software, hardware and people, that are organized to accomplish a specific purpose. A physical system can be described by one or more models, possibly from different viewpoints. Contrast: *system*.
qualifier
An association attribute or tuple of attributes whose values partition the set of objects related to an object across an association.
realization
A specialized abstraction relationship between two sets of model elements, one representing a specification (the supplier) and the other representing an implementation of the latter (the client). Realization can be used to model stepwise refinement, optimizations, transformations, templates, model synthesis, framework composition, etc.
receive [a message]
The handling of a stimulus passed from a sender instance. See: *sender, receiver*.
receiver
The object handling a stimulus passed from a sender object. Contrast: *sender*.
reception
A declaration that a classifier is prepared to react to the receipt of a signal.
reference
1. A denotation of a model element.
2. A named slot within a classifier that facilitates navigation to other classifiers. Synonym: *pointer*.
refinement
A relationship that represents a fuller specification of something that has already been specified at a certain level of detail. For example, a design class is a refinement of an analysis class.
relationship

An abstract concept that specifies some kind of connection between elements. Examples of relationships include associations and generalizations.
repository
A facility for storing object models, interfaces, and implementations.
requirement
A desired feature, property, or behavior of a system.
responsibility
A contract or obligation of a classifier.
reuse
The use of a pre-existing artifact.
role
The named set of features defined over a collection of entities participating in a particular context.
Collaboration: The named set of behaviors possessed by a class or part participating in a particular context.
Part: a subset of a particular class which exhibits a subset of features possessed by the class
Associations: A synonym for association end often referring to a subset of classifier instances that are participating in the association.
run time
The period of time during which a computer program or a systemexecutes. Contrast: *modeling time*.
scenario
A specific sequence of actions that illustrates behaviors. A scenario may be used to illustrate an interaction or the execution of a use case instance. See: *interaction.*
semantic variation point
A point of variation in the semantics of a metamodel. It provides an intentional degree of freedom for the interpretation of the metamodel semantics.
send [a message]
The passing of a stimulus from a sender instance to a receiver instance. See: *sender, receiver*.
sender
The object passing a stimulus to a receiver instance. Contrast: *receiver*.
sequence diagram
A diagram that depicts an interaction by focusing on the sequence of messages that are exchanged, along with their corresponding event occurrences on the lifelines. Unlike a communication diagram, a sequence diagram includes time sequences but does not include object relationships. A sequence diagram can exist in a generic form (describes all possible scenarios) and in an instance form (describes one actual scenario). Sequence diagrams and communication diagrams express similar information, but show it in different ways. See: *communication diagram*.
signal
The specification of an asynchronous stimulus that triggers a reaction in the receiver in an asynchronous way and without a reply. The receiving object handles the signal as specified by its receptions. The data carried by a send request and passed to it by the occurrence of the send invocation event that caused the request is represented as attributes of the signal instance. A signal is defined independently of the classifiers handling the signal.
signature
The name and parameters of a behavioral feature. A signature may include an optional returned parameter.
single inheritance
A semantic variation of generalization in which a type may have only one supertype. Synonym: *multiple inheritance* [OMA]. Contrast: *multiple inheritance*.
slot
A specification that an entity modeled by an instance specification has a value or values for a specific structural feature.
software module
A unit of software storage and manipulation. Software modules include source code modules, binary code modules, and executable code modules.

specification
A set of requirements for a system or other classifier. Contrast: *implementation*.
state
A condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event. Contrast: *state* [OMA].
state machine diagram
A diagram that depicts discrete behavior modeled through finite state-transition systems. In particular, it specifies the sequences of states that an object or an interaction goes through during its life in response to events, together with its responses and actions. See: *state machine*.
state machine
A behavior that specifies the sequences of states that an object or an interaction goes through during its life in response to events, together with its responses and actions.
static classification
The assignment of an instance to a classifier where the assignment may not change to any other classifier. Contrast: *dynamic classification.*
stereotype
A class that defines how an existing metaclass (or stereotype) may be extended, and enables the use of platform or domain specific terminology or notation in addition to the ones used for the extended metaclass. Certain stereotypes are predefined in the UML, others may be user defined. Stereotypes are one of the extensibility mechanisms in UML. See: *constraint, tagged value.*
stimulus
The passing of information from one instance to another, such as raising a signal or invoking an operation. The receipt of a signal is normally considered an event. See: *message*.
string
A sequence of text characters. The details of string representation depend on implementation, and may include character sets that support international characters and graphics.
structural feature
A static feature of a model element, such as an attribute.
structural model aspect
A model aspect that emphasizes the structure of the objects in a system, including their types, classes, relationships, attributes, and operations.
structure diagram
A form of diagram that depicts the elements in a specification that are irrespective of time. Class diagrams and component diagrams are examples of structure diagrams.
subactivity state
A state in an activity diagram that represents the execution of a non-atomic sequence of steps that has some duration.
subclass
In a generalization relationship, the specialization of another class, the superclass. See: *generalization*. Contrast: *superclass*.
submachine state
A state in a state machine that is equivalent to a composite state but whose contents are described by another state machine.
substate
A state that is part of a composite state. See: *concurrent state, disjoint state*.
subpackage
A package that is contained in another package.
subsystem
A unit of hierarchical decomposition for large systems. A subsystem is commonly instantiated indirectly. Definitions of subsystems vary widely among domains and methods, and it is expected that domain and method profiles will specialize this construct. A subsystem may be defined to have specification and realization elements.
subtype
In a generalization relationship, the specialization of another type, the supertype. See: *generalization*. Contrast: *supertype*.
superclass

In a generalization relationship, the generalization of another class, the subclass. See: *generalization.* Contrast: *subclass*.
supertype
In a generalization relationship, the generalization of another type, the subtype. See: *generalization*. Contrast: *subtype*.
supplier
A classifier that provides services that can be invoked by others. Contrast: *client*.
synch state
A vertex in a state machine used for synchronizing the concurrent regions of a state machine.
system
An organized array of elements functioning as a unit Also, a top-level subsystem in a model.
tagged value
The explicit definition of a property as a name-value pair. In a tagged value, the name is referred as the tag. Certain tags are predefined in the UML; others may be user defined. Tagged values are one of three extensibility mechanisms in UML. See: *constraint, stereotype.*
template
Synonym: *parameterized element*.
thread [of control]
A single path of execution through a program, a dynamic model, or some other representation of control flow. Also, a stereotype for the implementation of an active object as lightweight process. See *process*.
time event
An event that denotes the time elapsed since the current state was entered. See: *event*.
time expression
An expression that resolves to an absolute or relative value of time.
timing diagram
An interaction diagram that shows the change in state or condition of a lifeline (representing a Classifier Instance or Classifier Role) over linear time. The most common usage is to show the change in state of an object over time in response to accepted events or stimuli.
top level
A stereotype denoting the top-most package in a containment hierarchy. The topLevel stereotype defines the outer limit for looking up names, as namespaces "see" outwards. For example, opLevel subsystem represents the top of the subsystem containment hierarchy.
trace
A dependency that indicates a historical or process relationship between two elements that represent the same concept without specific rules for deriving one from the other.
transient object
An object that exists only during the execution of the process or thread that created it.
transition
A relationship between two states indicating that an object in the first state will perform certain specified actions and enter the second state when a specified event occurs and specified conditions are satisfied. On such a change of state, the transition is said to fire.
type
A stereotyped class that specifies a domain of objects together with the operations applicable to the objects, without defining the physical implementation of those objects. A type may not contain any methods, maintain its own thread of control, or be nested. However, it may have attributes and associations. Although an object may have at most one implementation class, it may conform to multiple different types. See also: *implementation class* Contrast: *interface*.
type expression
An expression that evaluates to a reference to one or more types.
uninterpreted
A placeholder for a type or types whose implementation is not specified by the UML. Every uninterpreted value has a corresponding string representation. See: *any* [CORBA].
usage
A dependency in which one element (the client) requires the presence of another element (the supplier) for its correct functioning or implementation.

use case
The specification of a sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system. See: *use case instances.*
use case diagram
A diagram that shows the relationships among actors and the subject (system), and use cases.
use case instance
The performance of a sequence of actions being specified in a use case. An instance of a use case. See: *use case class*.
use case model
A model that describes a system's functional requirements in terms of use cases.
utility
A stereotype that groups global variables and procedures in the form of a class declaration. The utility attributes and operations become global variables and global procedures, respectively. A utility is not a fundamental modeling construct, but a programming convenience.
value
An element of a type domain.
vertex
A source or a target for a transition in a state machine. A vertex can be either a state or a pseudo-state. See: *state, pseudo-state.*
view
A projection of a model that is seen from a given perspective or vantage point and omits entities that are not relevant to this perspective.
view element
A textual and/or graphical projection of a collection of model elements.
view projection
A projection of model elements onto view elements. A view projection provides a location and a style for each view
element.
visibility
An enumeration whose value (public, protected, or private) denotes how the model element to which it refers may be seen outside its enclosing namespace.