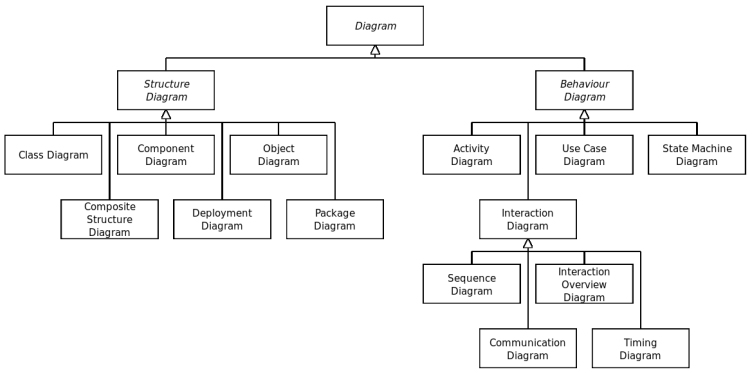
Class diagram



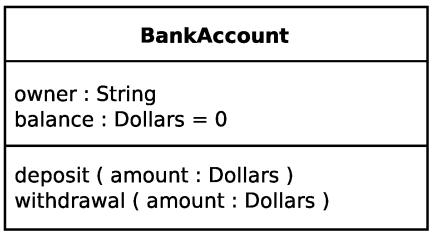
Hierarchy of UML 2.0 Diagrams, shown as a class diagram. The individual classes are represented just with one compartment, but they often contain up to three compartments.

UML diagrams
Structural UML diagrams
• Class diagram • Component diagram • Composite structure diagram • Deployment diagram • Object diagram • Package diagram • Profile diagram
Behavioral UML diagrams
• Activity diagram • Communication diagram • Interaction overview diagram • Sequence diagram • State diagram • Timing diagram • Use case diagram
• v • t • e [1]

In software engineering, a **class diagram** in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

Introduction

The class diagram is the main building block of object oriented modelling. It is used both for general conceptual modelling of the systematics of the application, and for detailed modelling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed.



A class with three sections.

In the diagram, classes are represented with boxes which contain three parts:

- The top part contains the name of the class
- The middle part contains the attributes of the class
- The bottom part gives the methods or operations the class can take or undertake

In the design of a system, a number of classes are identified and grouped together in a class diagram which helps to determine the static relations between those objects. With detailed modelling, the classes of the conceptual design are often split into a number of subclasses.

In order to further describe the behaviour of systems, these class diagrams can be complemented by state diagram or UML state machine.[2]

Members

UML provides mechanisms to represent class members, such as attributes and methods, and additional information about them.

Visibility

To specify the visibility of a class member (i.e., any attribute or method) these are the following notations that must be placed before the member's name:[3][*citation needed*]

```
"+"       Public
"-"       Private
"#"       Protected
"/"       Derived (can be combined with one of the others)
"_"       Static
"~"       Package
```

Scope

The UML specifies two types of scope for members: *instance* and *classifier*.[4]

- **Classifier members** are commonly recognized as “static” in many programming languages. The scope is the class itself.
  - Attribute values are equal for all instances
  - Method invocation does not affect the instance’s state
- **Instance members** are scoped to a specific instance.
  - Attribute values may vary between instances
  - Method invocation may affect the instance’s state (i.e., change instance’s attributes)

To indicate a classifier scope for a member, its name must be underlined. Otherwise, instance scope is assumed by default.

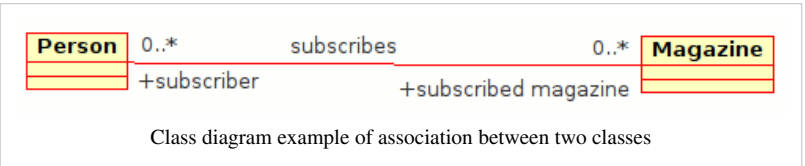
Relationships

A relationship is a general term covering the specific types of logical connections found on class and object diagrams. UML shows the following relationships:

Links

A *Link* is the basic relationship among objects.

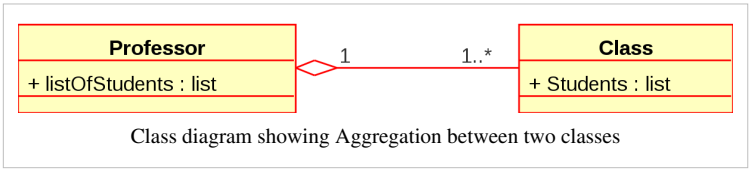
Association



Class diagram example of association between two classes

An *association* represents a family of links. Binary associations (with two ends) are normally represented as a line. An association can be named, and the ends of an association can be adorned with role names, ownership indicators, multiplicity, visibility, and other properties.
There are four different types of association: bi-directional, uni-directional, Aggregation (includes Composition aggregation) and Reflexive. Bi-directional and uni-directional associations are the most common ones.
For instance, a flight class is associated with a plane class bi-directionally. Association represents the static relationship shared among the objects of two classes. Example: "department offers courses", is an association relation.

Aggregation



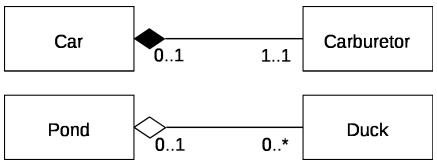
Class diagram showing Aggregation between two classes

Aggregation is a variant of the "has a" association relationship; aggregation is more specific than association. It is an association that represents a part-whole or part-of relationship. As a type of association, an aggregation can be named and have the same adornments that an association can. However, an aggregation may not involve more than two classes.

Aggregation can occur when a class is a collection or container of other classes, but where the contained classes do not have a strong *life cycle dependency* on the container—essentially, if the container is destroyed, its contents are not.

In UML, it is graphically represented as a *hollow* diamond shape on the containing class end of the tree with a single line that connects the contained class to the containing class. The aggregate is semantically an extended object that is treated as a unit in many operations, although physically it is made of several lesser objects.

Composition



Class diagram showing Composition between two classes at top and Aggregation between two classes at bottom

Composition is a stronger variant of the "owns a" association relationship; composition is more specific than aggregation.

Composition usually has a strong *life cycle dependency* between instances of the container class and instances of the contained class(es): If the container is destroyed, normally every instance that it contains is destroyed as well. (Note that, where allowed, a part can be removed from a composite before the composite is deleted, and thus not be deleted as part of the composite.)

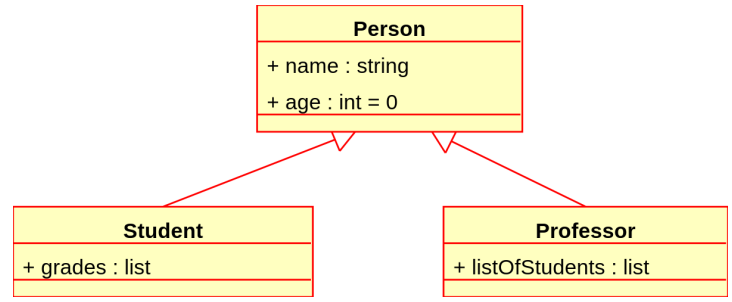
The UML graphical representation of a composition relationship is a *filled* diamond shape on the containing class end of the tree of lines that connect contained class(es) to the containing class.

Differences between composition and aggregation

When attempting to represent real-world whole-part relationships, e.g., an engine is a part of a car, the composition relationship is most appropriate. However, when representing a software or database relationship, e.g., car model engine ENG01 is part of a car model CM01, an aggregation relationship is best, as the engine, ENG01 may be also part of a different car model. Thus the aggregation relationship is often called "catalog" containment to distinguish it from composition's "physical" containment.

Class level relationships

Generalization



Class diagram showing generalization between one superclass and two subclasses

The Generalization relationship ("is a") indicates that one of the two related classes (the *subclass*) is considered to be a specialized form of the other (the *super type*) and superclass is considered as ***'Generalization***' of subclass. In practice, this means that any instance of the subtype is also an instance of the superclass. An exemplary tree of generalizations of this form is found in biological classification: human beings are a subclass of simian, which are a subclass of mammal, and so on. The relationship is most easily understood by the phrase 'an A is a B' (a human is a mammal, a **mammal** is an animal).

The UML graphical representation of a Generalization is a hollow triangle shape on the superclass end of the line (or tree of lines) that connects it to one or more subtypes.

The generalization relationship is also known as the *inheritance* or *"is a"* relationship.

The *superclass* (base class) in the generalization relationship is also known as the *"parent"*, *superclass*, *base class*, or *base type*.

The *subtype* in the specialization relationship is also known as the *"child"*, *subclass*, *derived class*, *derived type*, *inheriting class*, or *inheriting type*.

Note that this relationship bears no resemblance to the biological parent/child relationship: the use of these terms is extremely common, but can be misleading.

- Generalization-Specialization relationship

  A is a type of B

  E. g. "an oak is a type of tree", "an automobile is a type of vehicle"

Generalization can only be shown on class diagrams and on Use case diagrams.

Realization

In UML modelling, a realization relationship is a relationship between two model elements, in which one model element (the client) realizes (implements or executes) the behavior that the other model element (the supplier) specifies.

The UML graphical representation of a Realization is a hollow triangle shape on the interface end of the *dashed* line (or tree of lines) that connects it to one or more implementers. A plain arrow head is used on the interface end of the dashed line that connects it to its users. In component diagrams, the ball-and-socket graphic convention is used (implementors expose a ball or lollipop, while users show a socket).

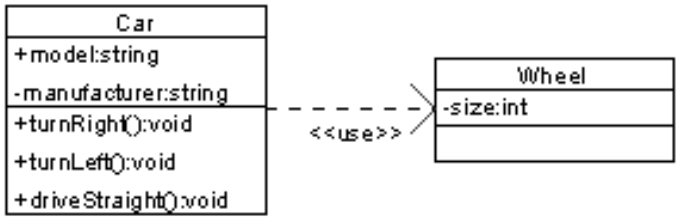
Realizations can only be shown on class or component diagrams.

A realization is a relationship between classes, interfaces, components, and packages that connects a client element with a supplier element. A realization relationship between classes and interfaces and between components and interfaces shows that the class realizes the operations offered by the interface.

General relationship

Dependency

Dependency is a weaker form of bond which indicates that one class depends on another because it uses it at some point in time. One class depends on another if the independent class is a parameter variable or local variable of a method of the dependent class. This is different from an association, where an attribute of the dependent class is an instance of the independent class.



Class diagram showing dependency between "Car" class and "Wheel" class (An even clearer example would be "Car depends on Wheel", because Car already *aggregates* (and not just *uses*) Wheel)

Multiplicity

The association relationship indicates that (at least) one of the two related classes makes reference to the other. In contrast with the generalization relationship, this is most easily understood through the phrase 'A has a B' (a mother cat has kittens, kittens have a mother cat).

The UML representation of an association is a line with an optional arrowhead indicating the *role* of the object(s) in the relationship, and an optional notation at each end indicating the *multiplicity* of instances of that entity (the number of objects that participate in the association).

0..1	No instances, or one instance (optional, mayWikipedia:Please clarify)
1	Exactly one instance
0..* or *****	Zero or more instances
1..*	One or more instances (at least one)

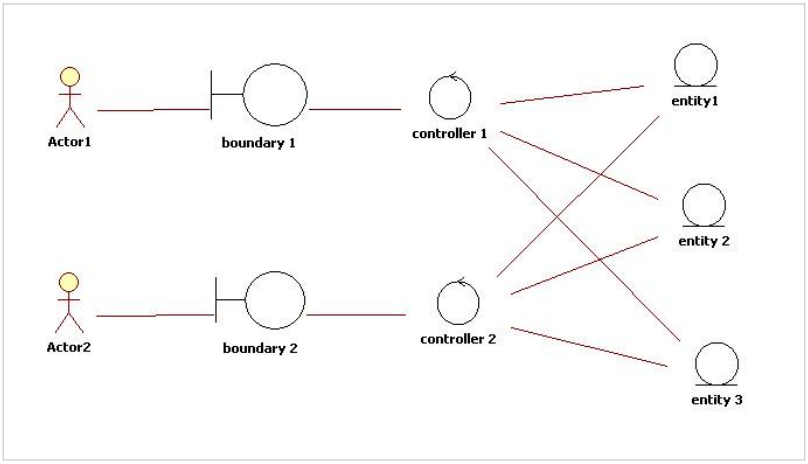
Analysis stereotypes

In the early stages of a project's technical analysis, class diagrams can be used to produce early conceptual models of the system. Classes at this stage often take the form of boundaries, controls and entities and rarely survive into the design without heavy changes.

Entities

Entity classes model the information handled by the system, and sometimes the behavior associated with the information. They should not be identified as database tables or other data-stores.

They are drawn as circles with a short line attached to the bottom of the circle. Alternatively, they can be drawn as normal classes with the «entity» stereotype notation above the class name.



References

[1] http://en.wikipedia.org/w/index.php?title=Template:UML_diagram_types&action=edit
[2] Scott W. Ambler (2009) UML 2 Class Diagrams (http://www.agilemodeling.com/artifacts/classDiagram.htm). Webdoc 2003-2009. Accessed Dec 2, 2009
[3] Holub Associates: UML Reference Card (http://www.holub.com/goodies/uml/), Version 2.1.2: August 2007. Retrieved 12 March 2011.
[4] OMG Unified Modeling Language (OMG UML) Superstructure (http://www.omg.org/spec/UML/2.3/Superstructure/PDF/), Version 2.3: May 2010. Retrieved 23 September 2010.

External links

- Introduction to UML 2 Class Diagrams (http://www.agilemodeling.com/artifacts/classDiagram.htm)
- UML 2 Class Diagram Guidelines (http://www.agilemodeling.com/style/classDiagram.htm)
- IBM Class diagram Introduction (http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/)
- OMG UML 2.2 specification documents (http://www.omg.org/spec/UML/2.2/)
- UML 2 Class Diagrams (http://www.uml-diagrams.org/class-diagrams.html)

Article Sources and Contributors

Class diagram *Source*: http://en.wikipedia.org/w/index.php?oldid=585491342 *Contributors*: 16@r, 9258fahsflkh917fas, Active Banana, Alexf, Anandmr07, Andonic, Astralblue, Autumnalmonk, Avuthu, Barts1a, Bennetto, Bergsten, Betacommand, Bhanks, Bigtophat, BrainMarble, Brick Thrower, Calm, Can't sleep, clown will eat me, Carmichael, Certes, Chico75, Chuunen Baka, Closedmouth, Codeshepherd, Crazycomputers, Ctbolt, DHorse1, Dancsi5000, Danim, Daveryan, DePiep, Dfoxvog, Disavian, Dmccreary, Doug Bell, Dougher, Dr. Zombieman, Dreadstar, EagerToddler39, Esap, Finem, Flewis, Gang65, Gerardohc, Gilliam, Gimboid13, Giridharhb, Gregly, HamMElion, Harendra1987, Heiko, Ian.desouza, Infrangible, Isarra (HG), J.delanoy, JaGa, JakobVoss, JamesBWatson, Jdiemer, Jeff G., Jeltz, Jessedoherty, Jmcw37, Joe Lennon, Josephmarty, Jozrael, Juandev, Justin W Smith, Kanenas, Karl80, Ketsuekigata, Khalid hassani, Kif2, Kishorekumar 62, Klemen Kocjancic, Korg, LambdaB, LaurentSmith, Leirbag.arc, LeonardoGregianin, Ligulem, Lloydsmart, Luna Santin, MC10, Makslane, Mark Renier, Materialscientist, Mathiasl26, Mdd, Medgno, MirianBruckschen, Mjchonoles, MrOllie, Muu-karhu, Myhlow, Mykolas OK, Niharcatseye2k4, Nltheshadow, Nonmal, Noodlez84, Oberiko, Ohnoitsjamie, Oktal, Oni Lukos, P pateriya, Pccestudent, Penumbra2000, Pinethicket, PsyberS, Quod erat demonstrandum 3.14159, Raghul.jayagopal, Rcronk, Rodasmith, Roma emu, Ronbarak, SAE1962, Sae1962, Samirsyed, Seanhalle, Shadowjams, Shenme, Shilpa More, Shnako, Shoessss, Sinisa.rudan, Sminthopsis84, Snigbrook, Snooper77, SpeedyGonsales, Spoxox, Sundar22in, TXAggie, Taemyr, Taeshadow, Tbhotch, Techtonik, The Rambling Man, The Utahraptor, The prophet wizard of the crayon cake, Ticaro, Tide rolls, Timothyclethbridge, Titodutta, Tobias Bergemann, Touriste (usurped), TyrantX, Ubiquity, Umawera, Unschool, Urhixidur, VictoriaUni111, Vipb, Wavelength, West.andrew.g, Widr, Wikidudeman, Wikipelli, Winchelsea, Windchaser, Writ Keeper, Yunshui, 𝕏, 582 anonymous edits

Image Sources, Licenses and Contributors

Image:Uml diagram.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Uml_diagram.svg *License*: Creative Commons Attribution-Sharealike 2.5 *Contributors*: Dave A Ryan

File:BankAccount1.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:BankAccount1.svg *License*: GNU Free Documentation License *Contributors*: Donald Bell

Image:UML role example.gif *Source*: http://en.wikipedia.org/w/index.php?title=File:UML_role_example.gif *License*: GNU Free Documentation License *Contributors*: User:SreeBot

Image:KP-UML-Aggregation-20060420.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:KP-UML-Aggregation-20060420.svg *License*: Public Domain *Contributors*: Ma-Lik, Snooper77, 2 anonymous edits

Image:AggregationAndComposition.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:AggregationAndComposition.svg *License*: Public Domain *Contributors*: ACiD2, Mikm, Pne, Mjchonoles, Rjgodoy, NevilleDNZ, Tobias Bergemann

Image:KP-UML-Generalization-20060325.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:KP-UML-Generalization-20060325.svg *License*: Public Domain *Contributors*: Bayo, Noodlez84, 2 anonymous edits

Image:Class Dependency.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Class_Dependency.png *License*: Creative Commons Attribution 3.0 *Contributors*: Samirsyed

Image:EntityControlBoundary Pattern.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:EntityControlBoundary_Pattern.jpg *License*: Public Domain *Contributors*: http://www.cs.sjsu.edu/~pearce/modules/lectures/ooa/analysis/ecb.htm

License