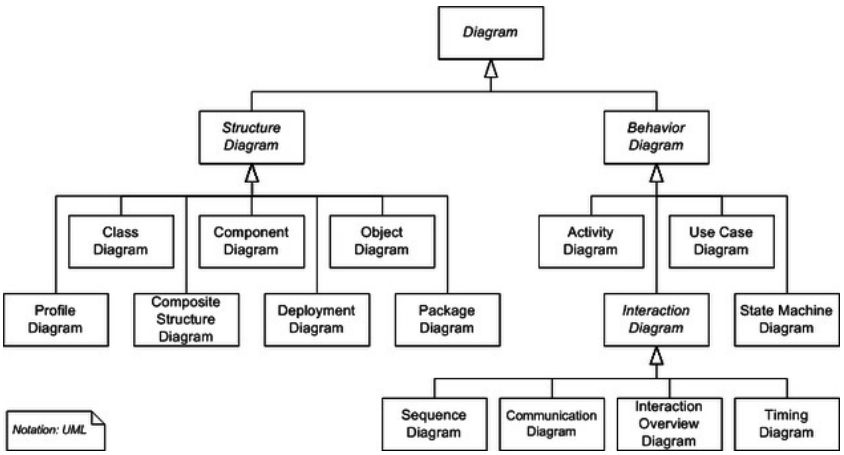
WIKIPEDIA

Class diagram

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.



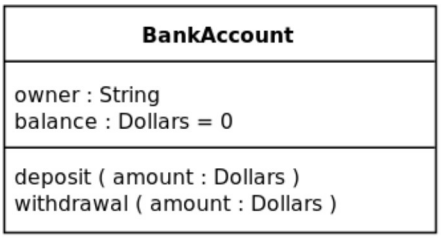
Hierarchy of UML 2.5 Diagrams, shown as a class diagram. The individual classes are represented just with one compartment, but they often contain up to three compartments.

The class diagram is the main building block of object-oriented modeling. It is used for general conceptual modeling of the structure of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling.[1] The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.

In the diagram, classes are represented with boxes that contain three compartments:

- The top compartment contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.
- The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase.
- The bottom compartment contains the operations the class can execute. They are also left-aligned and the first letter is lowercase.

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



A class with three compartments.

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

Contents

Members

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

Visibility

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

- `+` Public
- `-` Private
- `#` Protected
- `~` Package

Derived property is a property which value (or values) is produced or computed from other information, for example, by using values of other properties.

Derived property is shown with its name preceded by a forward slash '/'. [4]

Scope

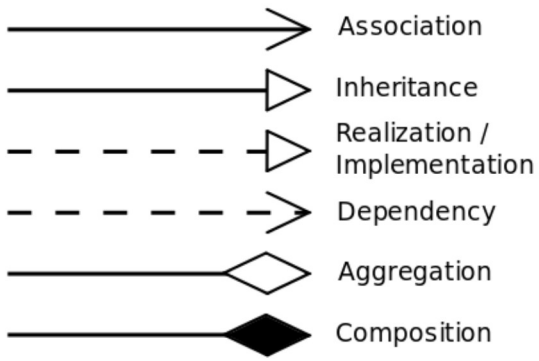
The UML specifies two types of scope for members: *instance* and *classifier*, and the latter is represented by underlined names.[5]

- **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 classifer’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 defines the following relationships:



UML relations notation

Instance-level relationships

Dependency

A *dependency* is a semantic connection between dependent and independent model elements.[6] It exists between two elements if changes to the definition of one element (the server or target) may cause changes to the other (the client or source). This association is uni-directional. A dependency is displayed as a dashed line with an open arrow that points from the client to the supplier.

Association

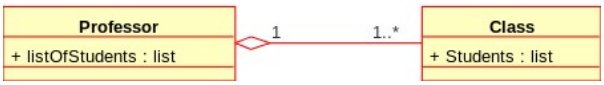
An *association* represents a family of links. A binary association (with two ends) is normally represented as a line. An association can link any number of classes. An association with three links is called a ternary association. An association can be named, and the ends of an association can be adorned with role names, ownership indicators, multiplicity, visibility, and other properties.

Class diagram example of association between two classes

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.

Aggregation



Class diagram showing Aggregation between two classes. Here, a Professor 'has a' class to teach.

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 shown in the image, a Professor 'has a' class to teach. 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; it must be a binary association. Furthermore, there is hardly a difference between aggregations and associations during implementation, and the diagram may skip aggregation relations altogether.[7]

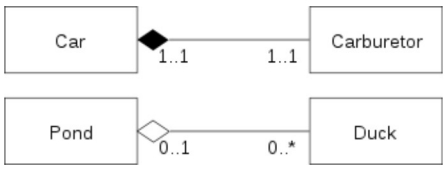
Aggregation can occur when a class is a collection or container of other classes, but the contained classes do not have a strong *lifecycle dependency* on the container. The contents of the container still exist when the container is destroyed.

In UML, it is graphically represented as a *hollow* diamond shape on the containing class with a single line that connects it to the contained 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.

Example: Library and Students. Here the student can exist without library, the relation between student and library is aggregation.

Composition

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



Two class diagrams. The diagram on top shows Composition between two classes: A Car has exactly one Carburetor, and a Carburetor is a part of one Car. Carburetors cannot exist as separate parts, detached from a specific car. The diagram on bottom shows Aggregation between two classes: A Pond has zero or more Ducks, and a Duck has at most one Pond (at a time). Duck can exist separately from a Pond, e.g. it can live near a lake. When we destroy a Pond we usually do not kill all the Ducks.

Differences between Composition and Aggregation

Composition relationship

1. When attempting to represent real-world whole-part relationships, e.g. an engine is a part of a car.
2. When the container is destroyed, the contents are also destroyed, e.g. a university and its departments.

Aggregation relationship

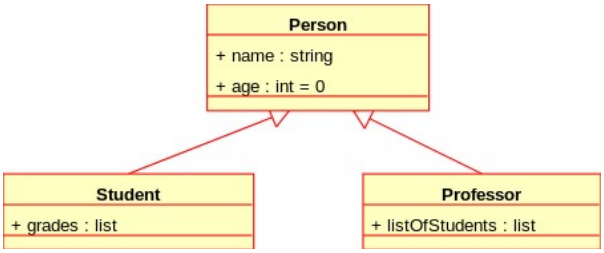
1. When representing a software or database relationship, e.g. car model engine ENG01 is part of a car model CM01, as the engine, ENG01, may be also part of a different car model.[8]
2. When the container is destroyed, the contents are usually not destroyed, e.g. a professor has students, when the professor dies the students don't die along with him or her.

Thus the aggregation relationship is often "catalog" containment to distinguish it from composition's "physical" containment.

Class-level relationships

Generalization/Inheritance

It indicates that one of the two related classes (the *subclass*) is considered to be a specialized form of the other (the *super type*) and the superclass is considered a Generalization of the 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: humans are a subclass of simian, which is 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).



Class diagram showing generalization between the superclass *Person* and the two subclasses *Student* and *Professor*

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.

> A is a type of B
> For example, "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/Implementation

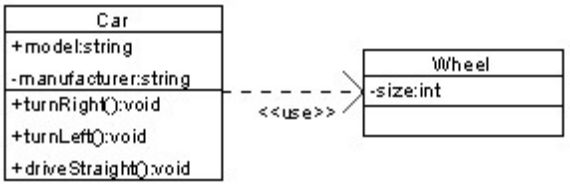
In UML modelling, a realization relationship is a relationship between two model elements, in which one model element (the client) realizes (implements or executes) the behavior that the other model element (the supplier) specifies.

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, whereas 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/components and interfaces shows that the class/component realizes the operations offered by the interface.

```
        symbolic of realization                 -------▻
```

General relationship

Dependency



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)

Dependency is a weaker form of bond that 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. Sometimes the relationship between two classes is very weak. They are not implemented with member variables at all. Rather they might be implemented as member function arguments.

Multiplicity

This association relationship indicates that (at least) one of the two related classes make reference to the other. This relationship is usually described as "A has a B" (a mother cat has kittens, kittens have a mother cat).

The UML representation of an association is a line connecting the two associated classes. At each end of the line there is optional notation. For example, we can indicate, using an arrowhead that the pointy end is visible from the arrow tail. We can indicate ownership by the placement of a ball, the role the elements of that end play by supplying a name for the role, and the *multiplicity* of instances of that entity (the range of number of objects that participate in the association from the perspective of the other end).

0	No instances (rare)
0..1	No instances, or one instance
1	Exactly one instance
1..1	Exactly one instance
0..*	Zero or more instances
*****	Zero or more instances
1..*	One or more instances

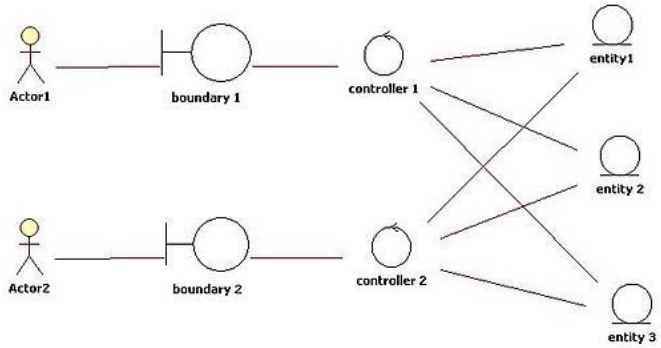
Analysis stereotypes

Entities

Entity classes model long-lived 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.



See also

- Executable UML
- List of UML tools
- Object-oriented modeling
- Dependency (UML)

Related diagrams

- Domain model
- Entity–relationship model
- Object diagram
- DSCD

References

1. Sparks, Geoffrey. "Database Modeling in UML" (http://www.methodsandtools.com/archive/archive.php?id=9). Retrieved 8 September 2011.
2. Scott W. Ambler (2009) UML 2 Class Diagrams (http://www.agilemodeling.com/artifacts/classDiagram.htm). Webdoc 2003-2009. Accessed Dec 2, 2009
3. *UML Reference Card, Version 2.1.2* (http://www.holub.com/goodies/uml/), Holub Associates, August 2007, retrieved 12 March 2011
4. "UML derived property is property which value is produced or computed from other information, for example, by using other properties" (https://www.uml-diagrams.org/derived-property.html). *www.uml-diagrams.org*. Retrieved 2019-01-24.
5. 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.
6. Fowler (2003) UML Distilled: A Brief Guide to the Standard Object Modeling Language
7. "UML Tutorial part 1: class diagrams" (https://web.archive.org/web/20070103141438/http://www.objectmentor.com/resources/articles/umlClassDiagrams.pdf) (PDF). Archived from the original (http://www.objectmentor.com/resources/articles/umlClassDiagrams.pdf) (PDF) on 2007-01-03. Retrieved 2015-07-18.
8. Goodwin, David. "Modelling and Simulation, p. 26" (http://www2.warwick.ac.uk/fac/sci/physics/research/condensedmatt/imr_cdt/students/david_goodwin/teaching/modelling/l3_objectorientation.pdf) (PDF). *The University of Warwick*. Retrieved 28 November 2015.

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)
