UML Cheat-Sheet

The Unified Modeling Language (UML) is a standard tool for documenting, designing, and analyzing object-oriented (OO) systems. The language is primarily graphical, and is considered a higher-level abstraction than specific OO programming languages such as C++ or Java. This is a cheat-sheet for some of the conventions used on this wiki.

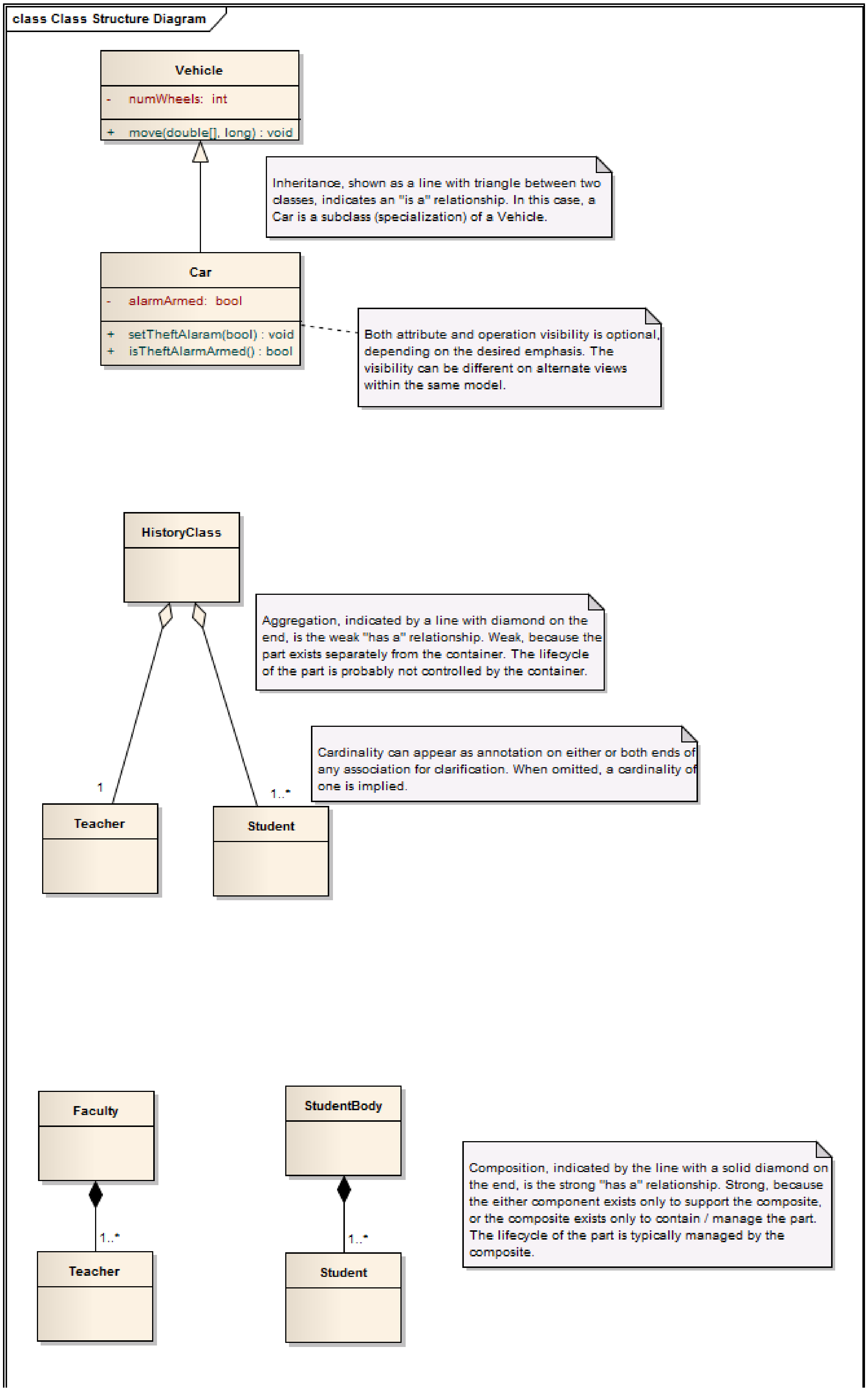
Note that we aren't using any type of code-generation or round-trip engineering that many UML tools support. For this reason we can be rather lax with our use of the language. (But also keep in mind that since we don't have automation in the process, the diagrams can quickly become out of date!)

A single UML model can consist of any number of diagrams, from among up to 14 different types. Each diagram is considered to belong to one of two views: static (also known as structural) or dynamic (aka. behavioral). Static views include the well-known class diagram, the component diagram, the deployment diagram, and others. The dynamic views include state machine diagram, activity diagram, sequence diagram, and others. A given UML model may have multiple diagrams of a given type, and may omit diagrams of some types.

We tend to use the Class Diagram and the Sequence Diagram the most, so we'll focus on those at this time. The Class Diagram shows inheritance, composition and other static relationships among classes and types. Sequence diagrams show the sequencing (cause and effect) behaviors among different objects, and become particularly useful in distributed systems to define component-level interactions.

Class Diagram

Here are some of the common symbols and meanings found on a **Class Diagram**.


class Class Structure Diagram
Vehicle
- numWheels: int
+ move(double[], long) : void
Car
- alarmArmed: bool
+ setTheftAlaram(bool) : void
+ isTheftAlarmArmed() : bool
HistoryClass
1
1..*
Teacher
Student
Faculty
StudentBody
1..*
1..*
Teacher
Student

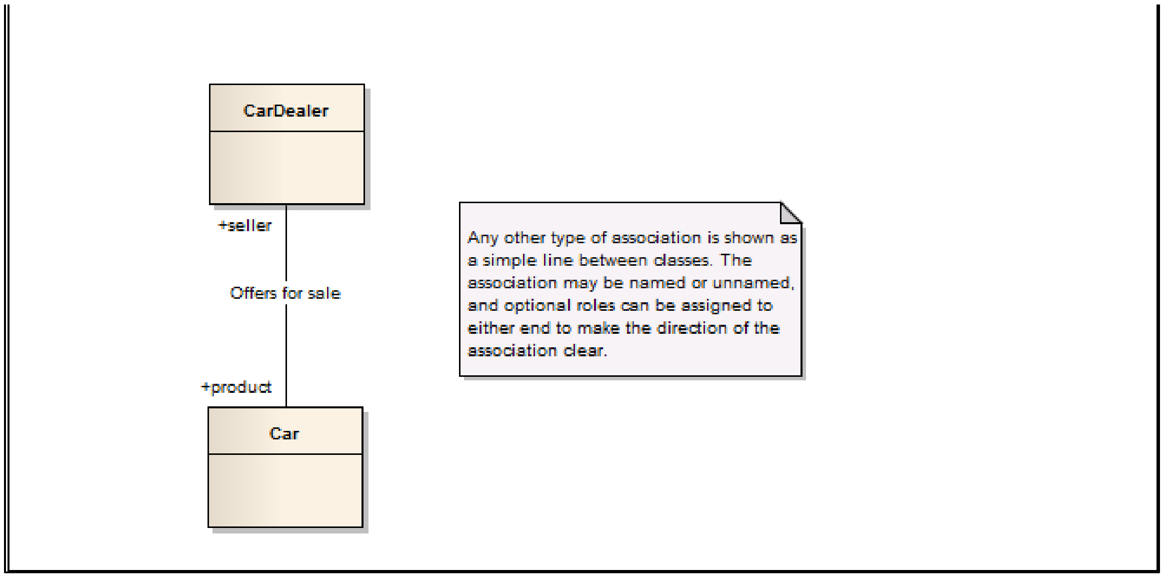
Inheritance, shown as a line with triangle between two classes, indicates an "is a" relationship. In this case, a Car is a subclass (specialization) of a Vehicle.

Both attribute and operation visibility is optional, depending on the desired emphasis. The visibility can be different on alternate views within the same model.

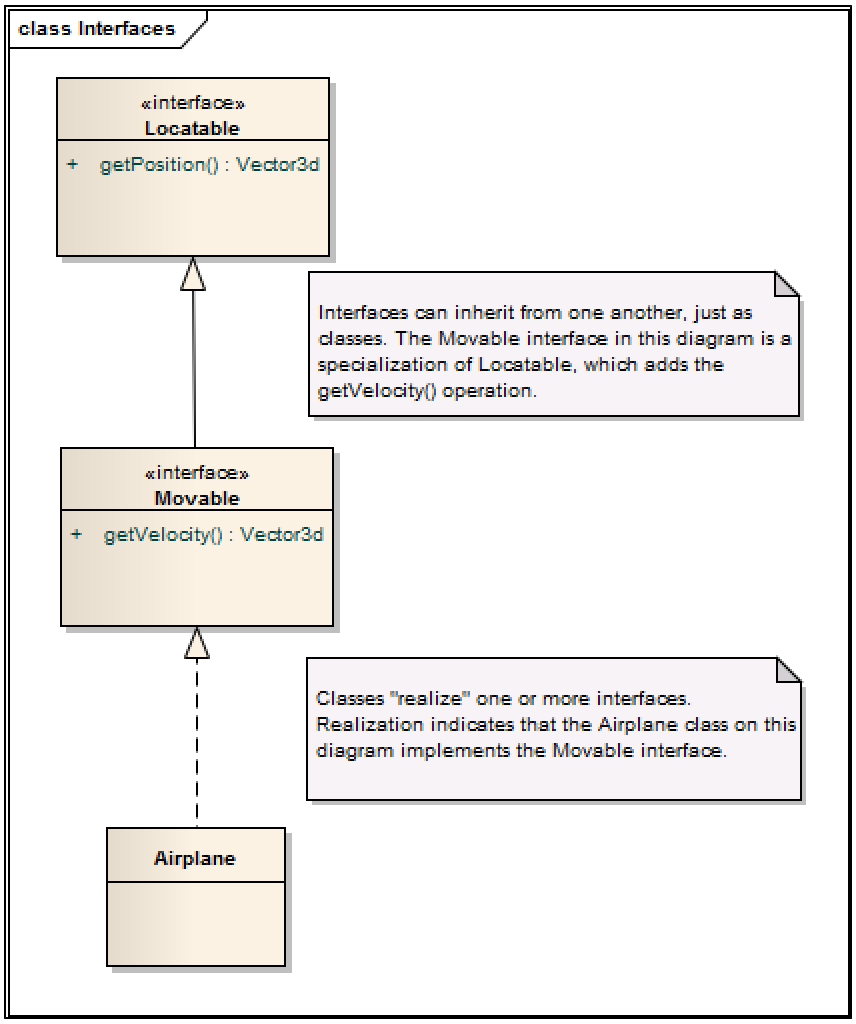
Aggregation, indicated by a line with diamond on the end, is the weak "has a" relationship. Weak, because the part exists separately from the container. The lifecycle of the part is probably not controlled by the container.

Cardinality can appear as annotation on either or both ends of any association for clarification. When omitted, a cardinality of one is implied.

Composition, indicated by the line with a solid diamond on the end, is the strong "has a" relationship. Strong, because the either component exists only to support the composite, or the composite exists only to contain / manage the part. The lifecycle of the part is typically managed by the composite.

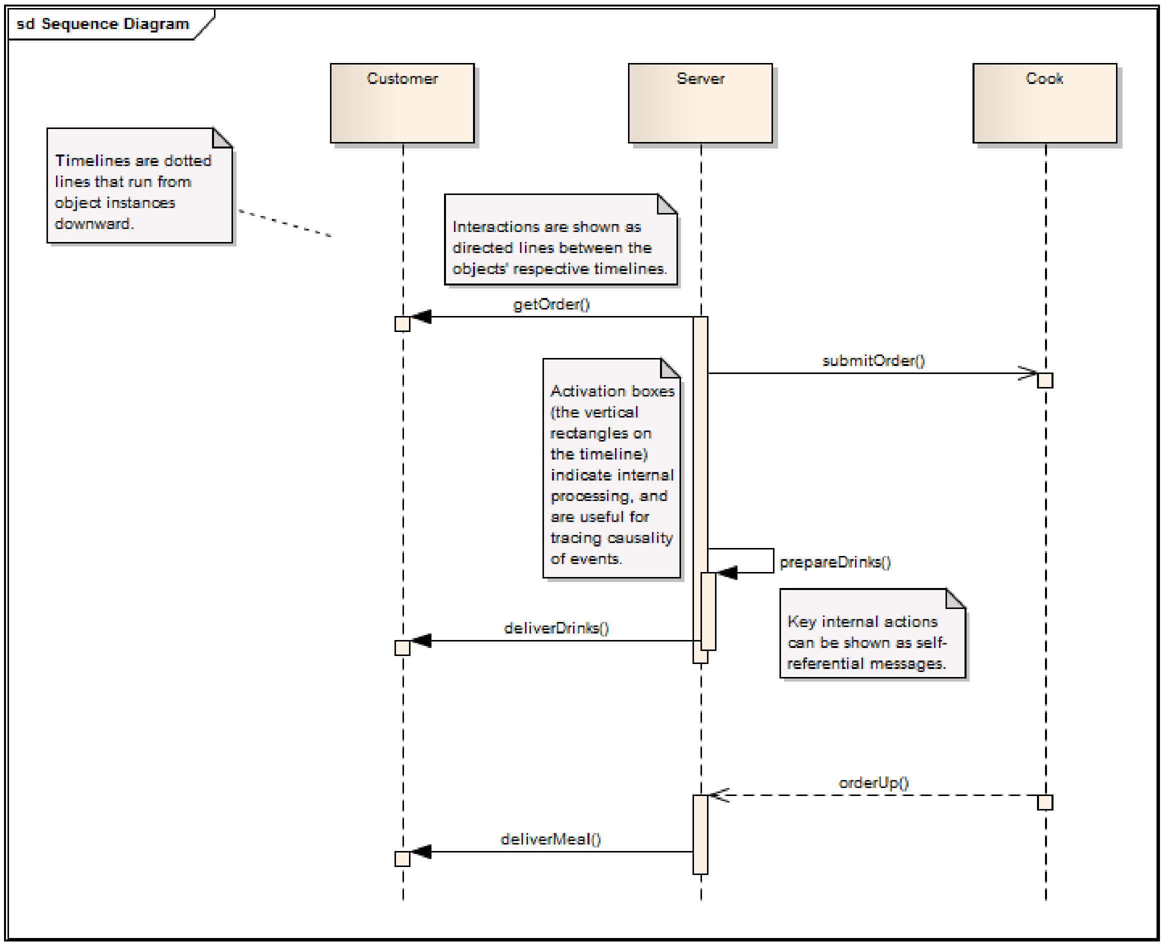

CarDealer
+seller
Offers for sale
+product
Car
Any other type of association is shown as a simple line between classes. The association may be named or unnamed, and optional roles can be assigned to either end to make the direction of the association clear.

The Class symbol (the boxes) can be annotated with a stereotype tag. Stereotypes can used for almost anything, but are most commonly used for denoting a well-know role of a common design pattern (ex: <<singleton>>), or for denoting an interface as opposed to a class. An interface defines a syntactic contract for communication with a component separate from the implementation of the component, a concept which maps to the Java and C# constructs of *interface*, or in C++ to a class that consists of only pure, virtual methods. Note that on the Class Diagram, interfaces may inherit from one another, but classes "realize" (i.e. implement) interfaces.


class Interfaces
«interface»
Locatable
+ getPosition() : Vector3d
Interfaces can inherit from one another, just as classes. The Movable interface in this diagram is a specialization of Locatable, which adds the getVelocity() operation.
«interface»
Movable
+ getVelocity() : Vector3d
Classes "realize" one or more interfaces. Realization indicates that the Airplane class on this diagram implements the Movable interface.
Airplane

Sequence Diagram

Sequence diagrams are one of the key types of behavioral diagrams in UML. They show interactions between objects, and can show activation periods in response to stimulus, and resultant interactions.


sd Sequence Diagram
Customer
Server
Cook
Timelines are dotted lines that run from object instances downward.
Interactions are shown as directed lines between the objects' respective timelines.
getOrder()
submitOrder()
Activation boxes (the vertical rectangles on the timeline) indicate internal processing, and are useful for tracing causality of events.
prepareDrinks()
Key internal actions can be shown as self-referential messages.
deliverDrinks()
orderUp()
deliverMeal()