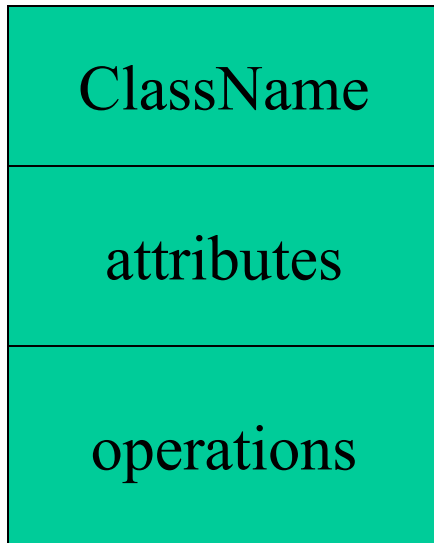


# ***Class Diagram***

Ex-4

Software Engineering lab

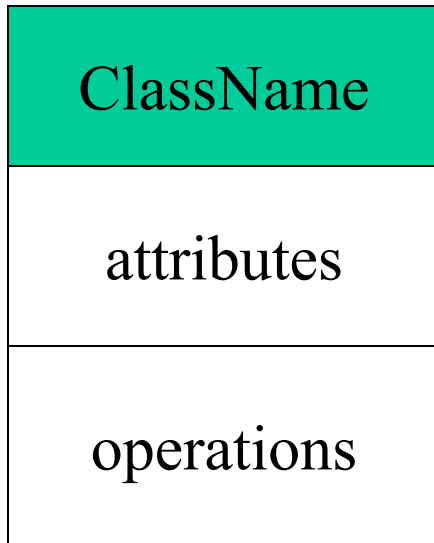
# *Classes*



A *class* is a description of a set of objects that share the same attributes, operations, relationships, and semantics.

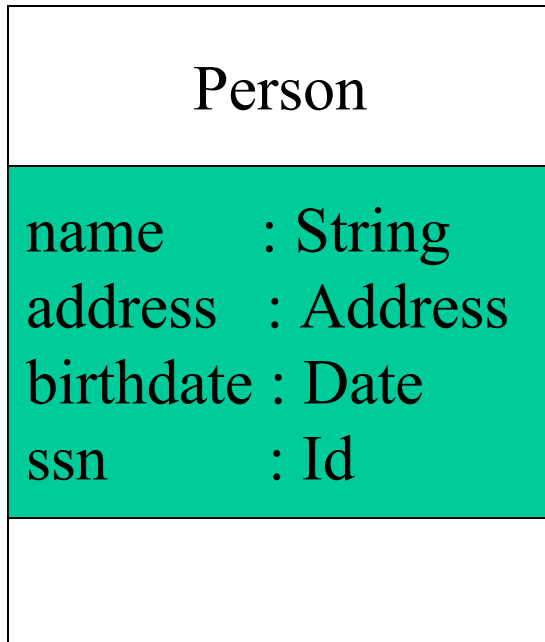
Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations in separate, designated compartments.

# *Class Names*



The name of the class is the only required tag in the graphical representation of a class. It always appears in the top-most compartment.

# *Class Attributes*

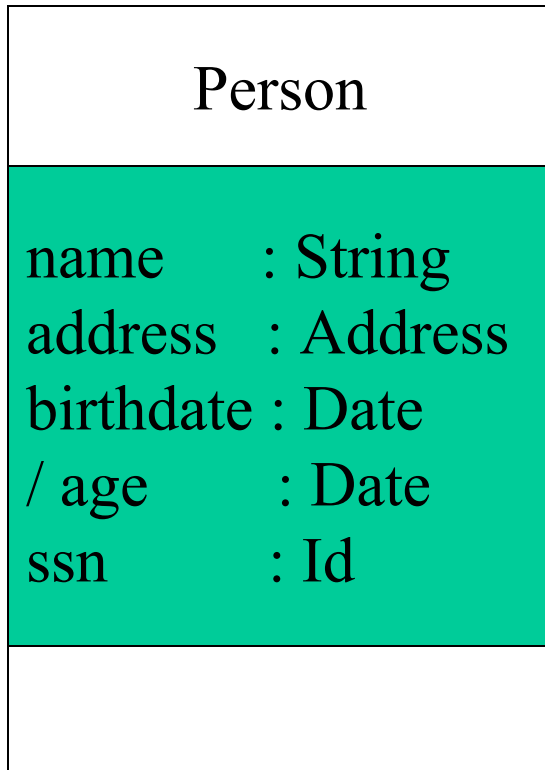


An *attribute* is a named property of a class that describes the object being modeled. In the class diagram, attributes appear in the second compartment just below the name-compartment.

# Class Attributes (Cont'd)

Attributes are usually listed in the form:

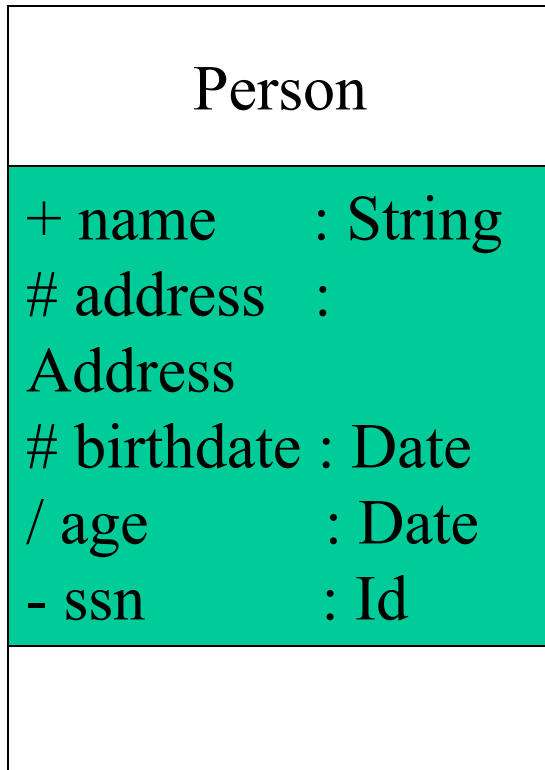
attributeName : Type



A *derived* attribute is one that can be computed from other attributes, but doesn't actually exist. For example, a Person's age can be computed from his birth date. A derived attribute is designated by a preceding '/' as in:

/ age : Date

# *Class Attributes (Cont'd)*



Attributes can be:

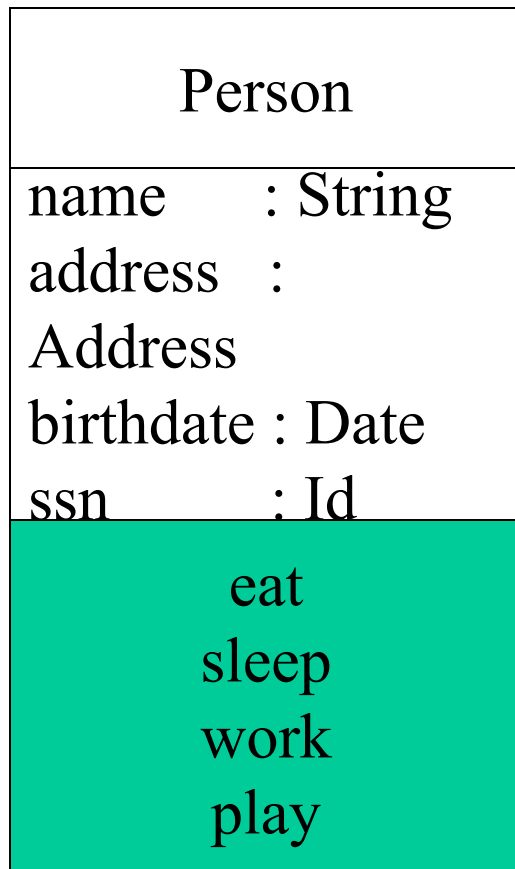
+ public

# protected

- private

/ derived

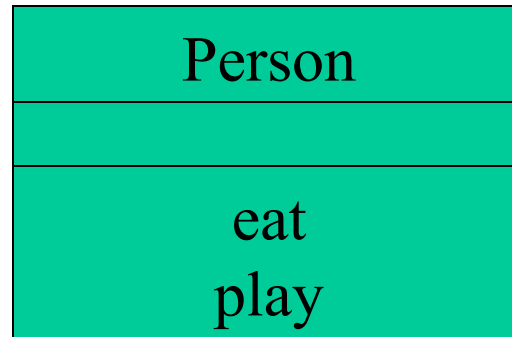
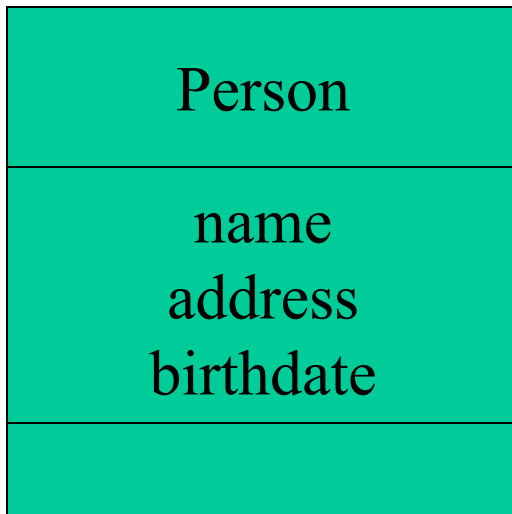
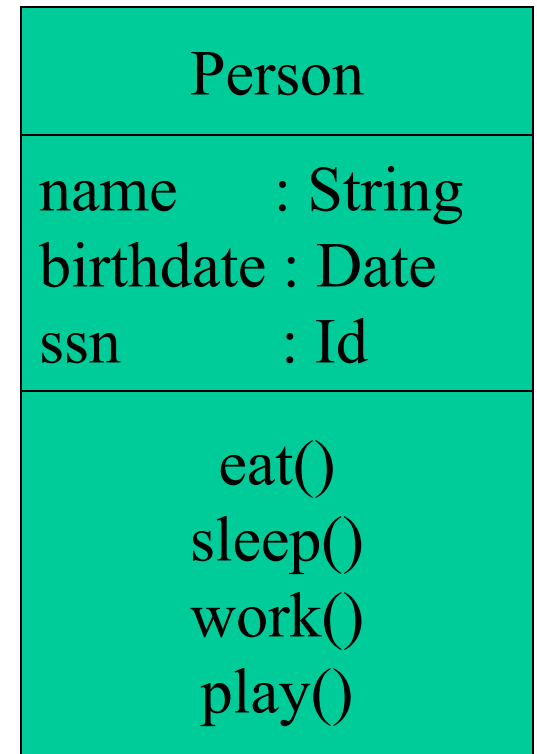
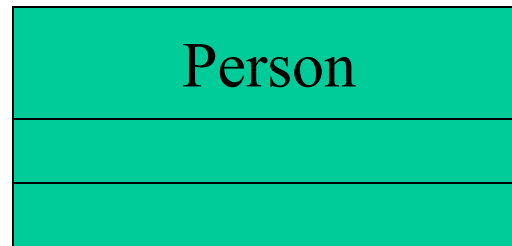
# *Class Operations*



*Operations* describe the class behavior and appear in the third compartment.

# *Depicting Classes*

When drawing a class, you needn't show attributes and operation in every diagram.





# *Relationships*

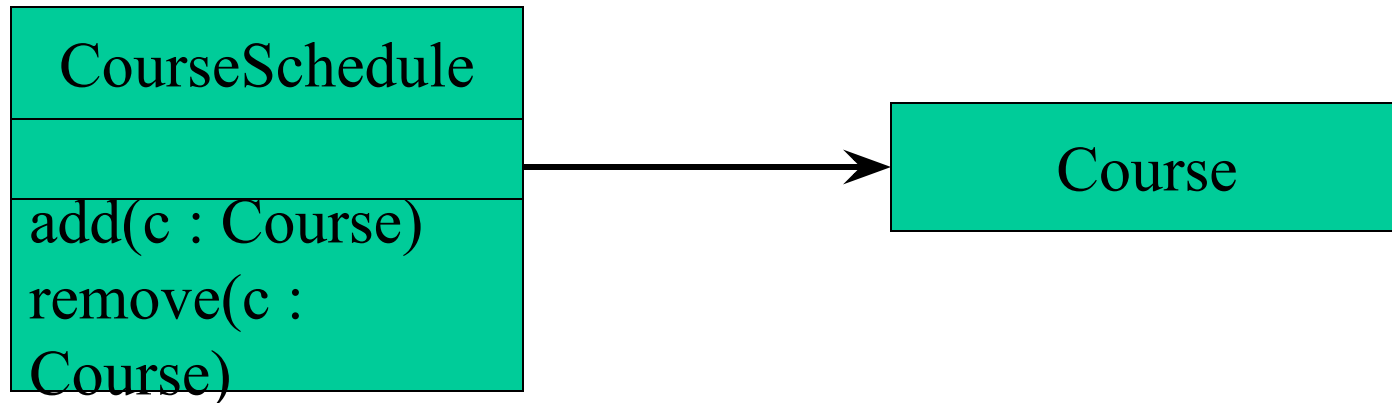
In UML, object interconnections (logical or physical), are modeled as relationships.

There are three kinds of relationships in UML:

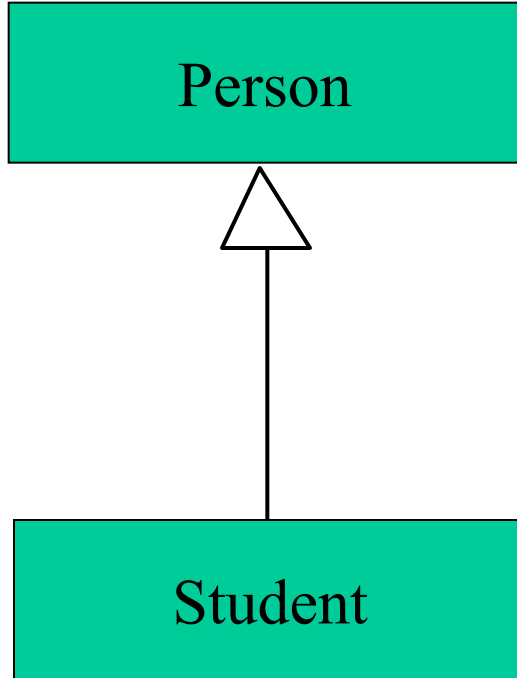
- dependencies
- generalizations
- associations

# *Dependency Relationships*

A *dependency* indicates a semantic relationship between two or more elements. The dependency from *CourseSchedule* to *Course* exists because *Course* is used in both the **add** and **remove** operations of *CourseSchedule*.



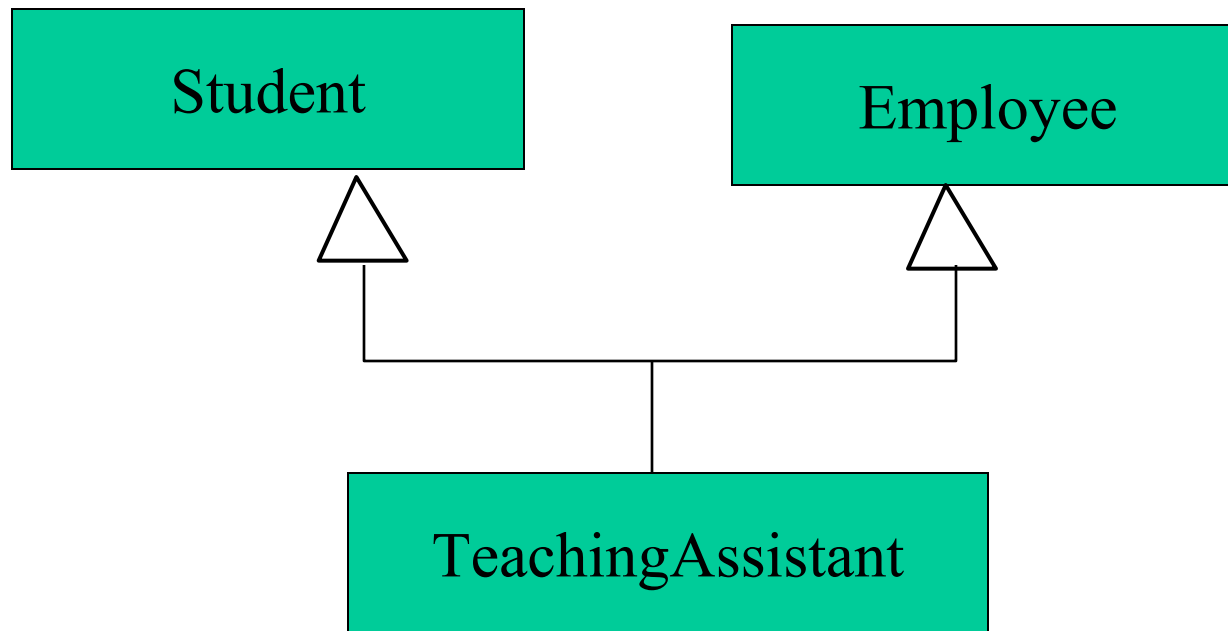
# *Generalization Relationships*



*A generalization* connects a subclass to its superclass. It denotes an inheritance of attributes and behavior from the superclass to the subclass and indicates a specialization in the subclass of the more general superclass.

# *Generalization Relationships (Cont'd)*

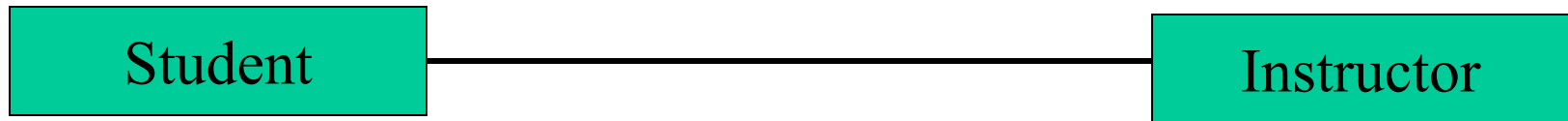
UML permits a class to inherit from multiple superclasses, although some programming languages (*e.g.*, Java) do not permit multiple inheritance.



# *Association Relationships*

If two classes in a model need to communicate with each other, there must be link between them.

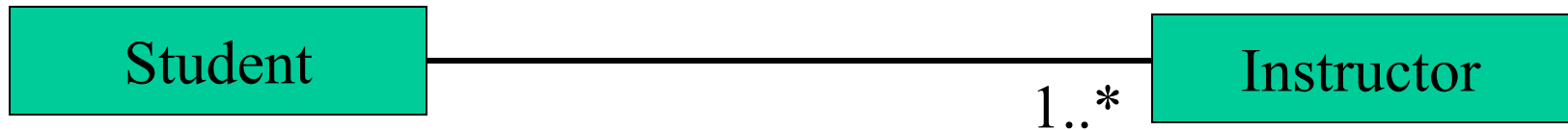
An *association* denotes that link.



# *Association Relationships (Cont'd)*

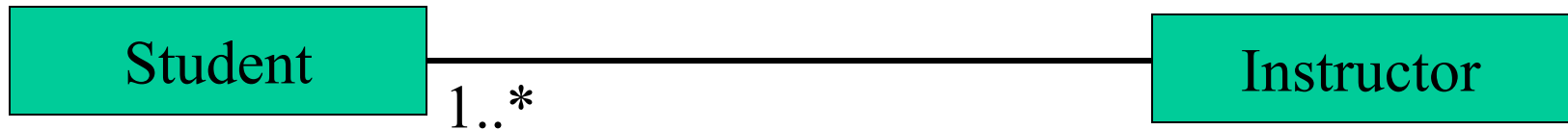
We can indicate the *multiplicity* of an association by adding *multiplicity adornments* to the line denoting the association.

The example indicates that a *Student* has one or more *Instructors*:



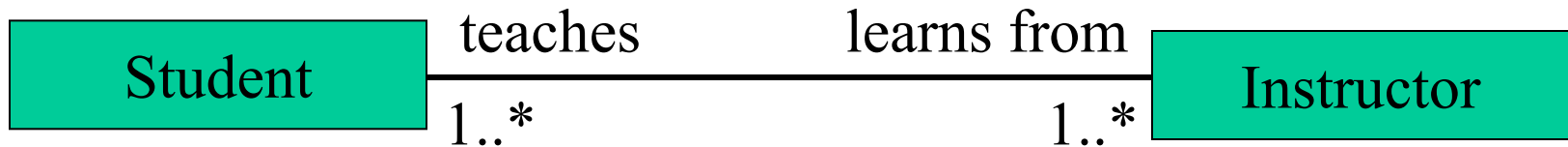
# *Association Relationships (Cont'd)*

The example indicates that every *Instructor* has one or more *Students*:



# Association Relationships (Cont'd)

We can also indicate the behavior of an object in an association (*i.e.*, the *role* of an object) using *rolenames*.





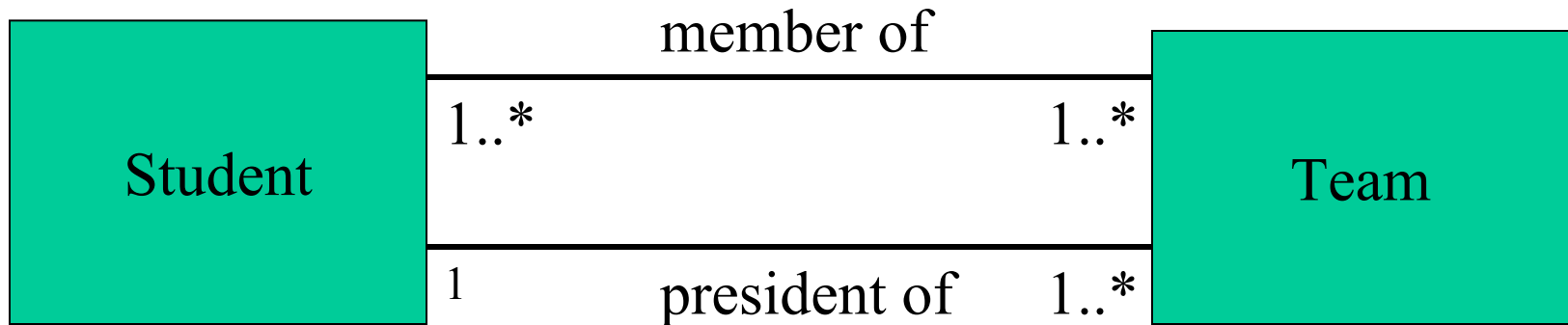
# *Association Relationships (Cont'd)*

We can also name the association.



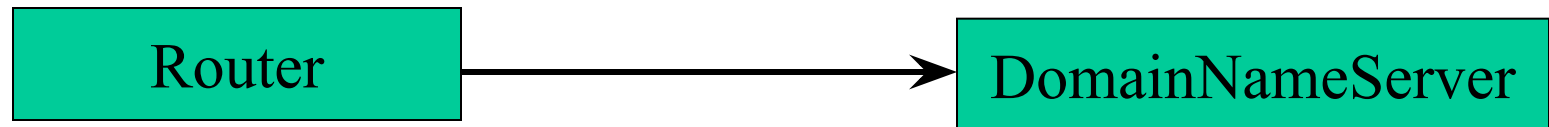
# *Association Relationships (Cont'd)*

We can specify dual associations.



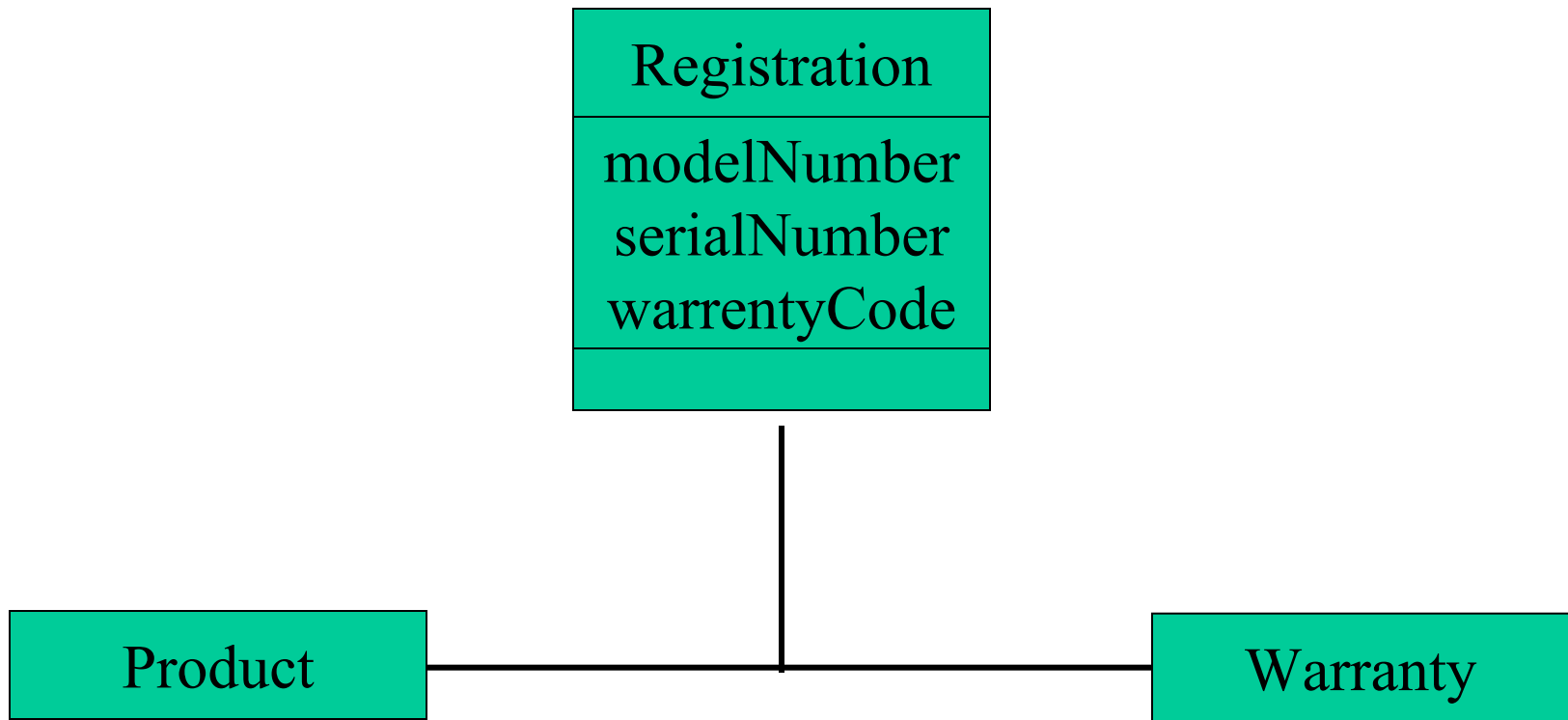
# *Association Relationships (Cont'd)*

We can constrain the association relationship by defining the *navigability* of the association. Here, a *Router* object requests services from a *DNS* object by sending messages to (invoking the operations of) the server. The direction of the association indicates that the server has no knowledge of the *Router*.



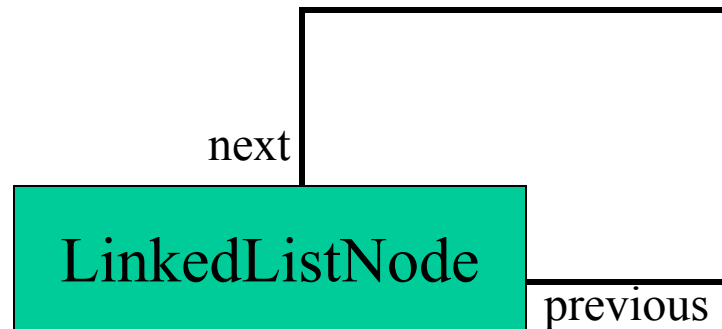
# Association Relationships (Cont'd)

Associations can also be objects themselves, called *link classes* or an *association classes*.



# *Association Relationships (Cont'd)*

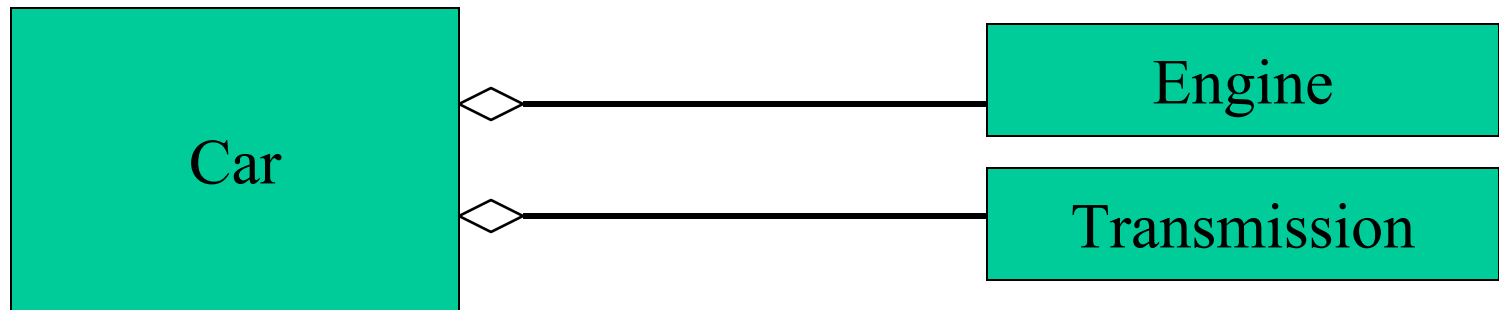
A class can have a *self association*.



# *Association Relationships (Cont'd)*

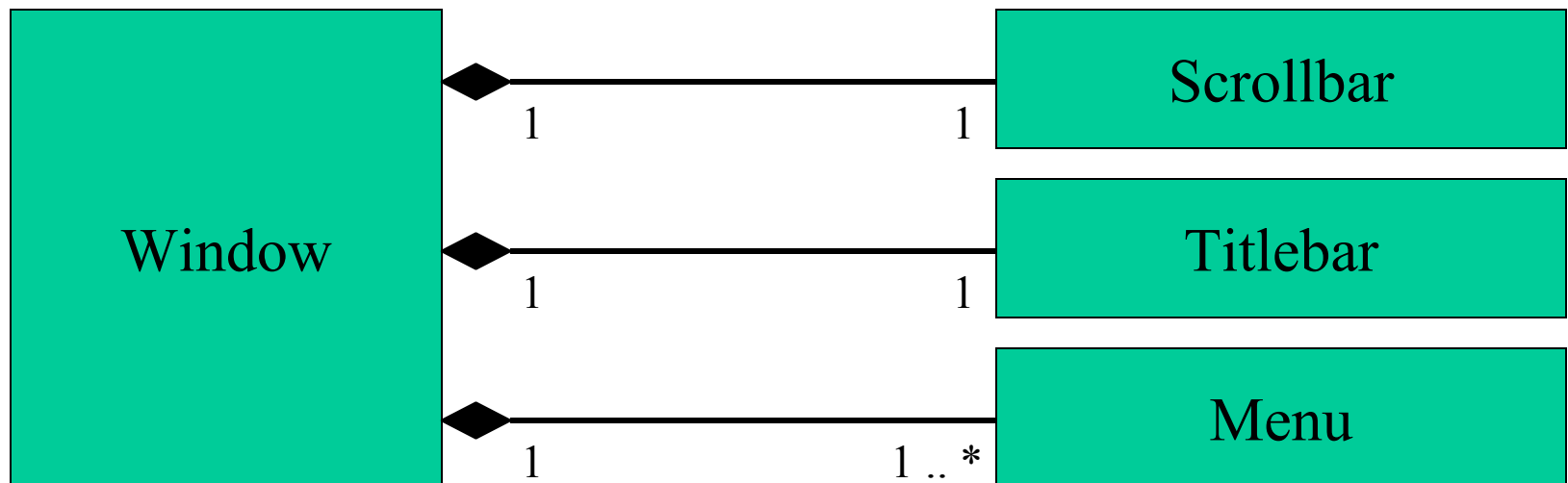
We can model objects that contain other objects by way of special associations called *aggregations* and *compositions*.

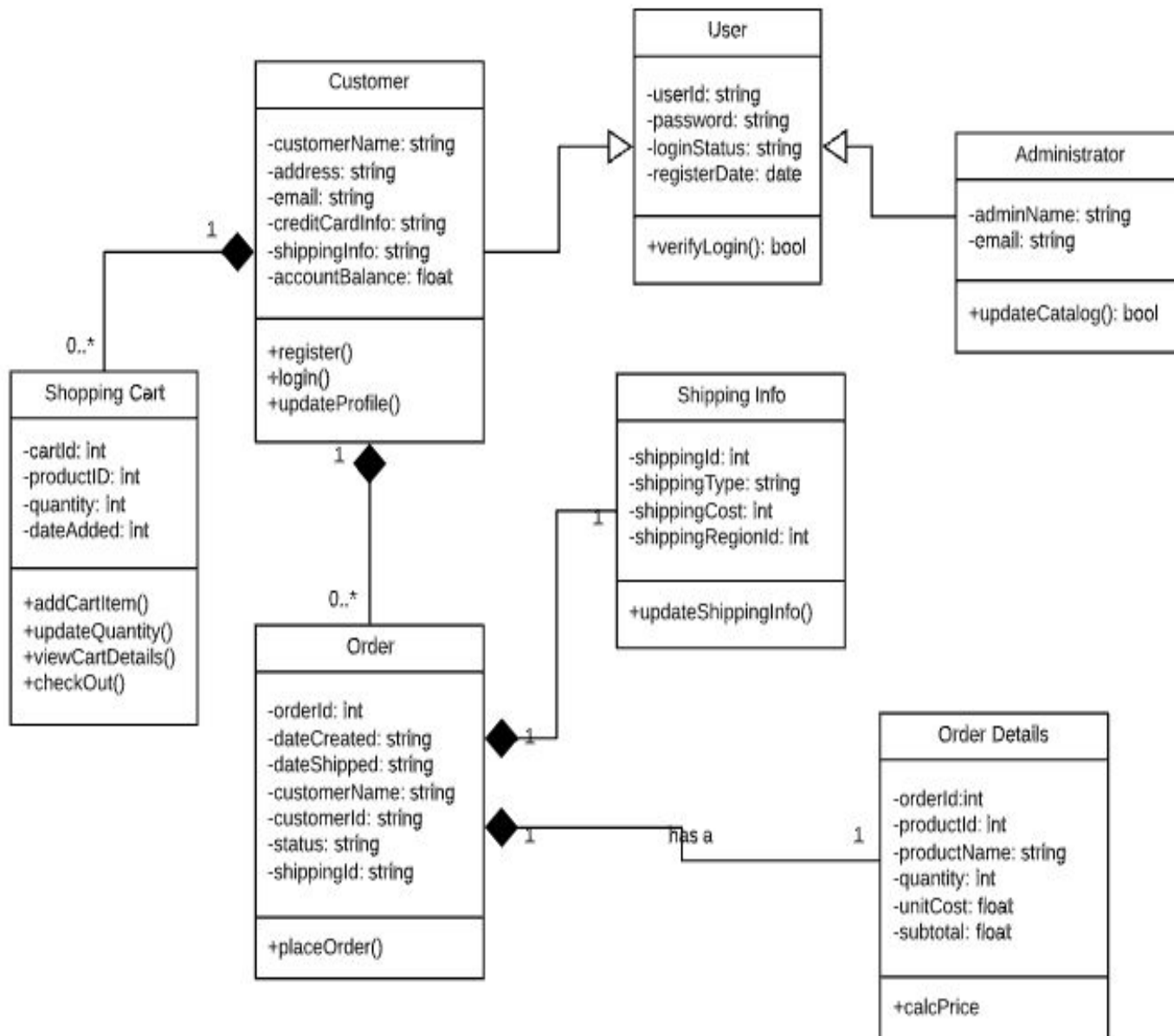
An *aggregation* specifies a whole-part relationship between an aggregate (a whole) and a constituent part, where the part can exist independently from the aggregate. Aggregations are denoted by a hollow-diamond adornment on the association.



# Association Relationships (Cont'd)

A *composition* indicates a strong ownership and coincident lifetime of parts by the whole (*i.e.*, they live and die as a whole). Compositions are denoted by a filled-diamond adornment on the association.





Sample class diagram