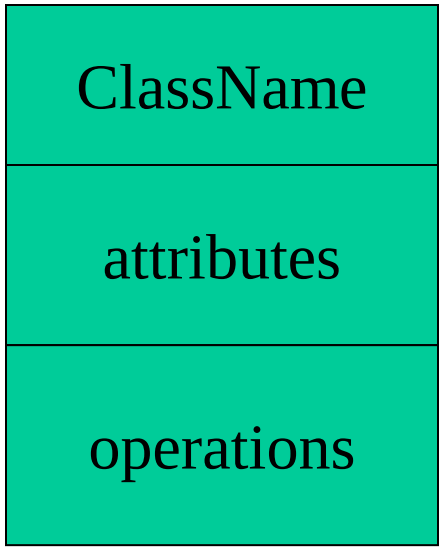
Class Diagram and Relationship

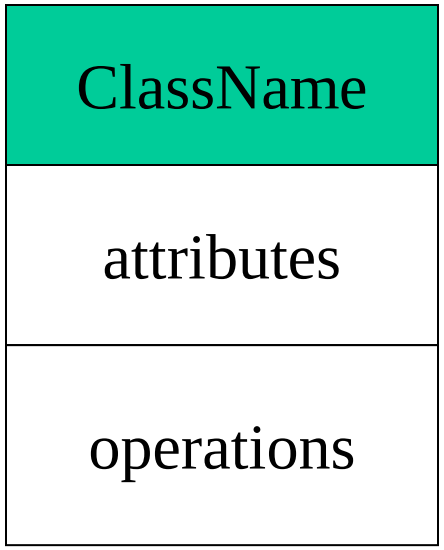
Classes



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

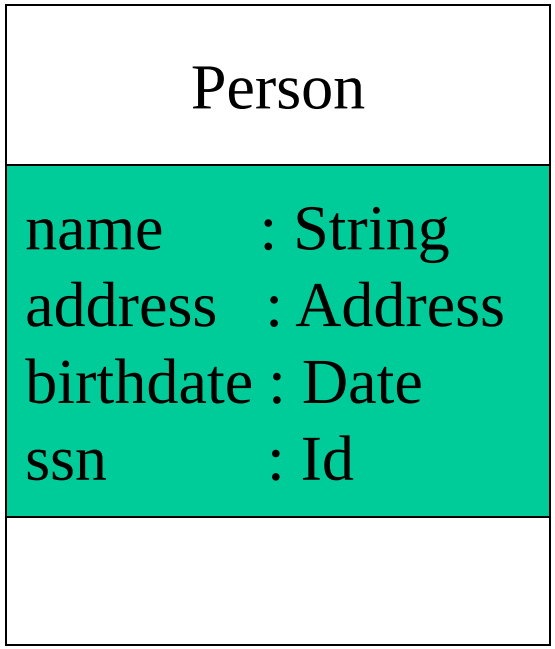
Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations in separate, designated compartments.

Class Names



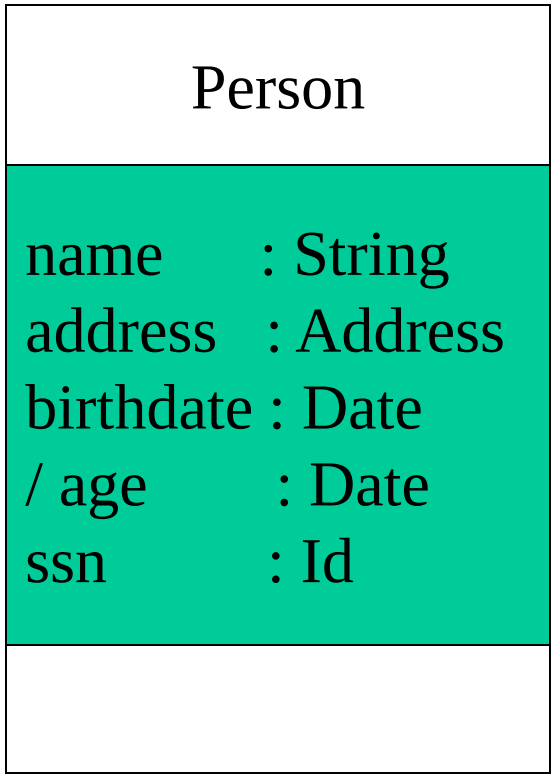
The name of the class is the only required tag in the graphical representation of a class. It always appears in the top-most compartment.

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)



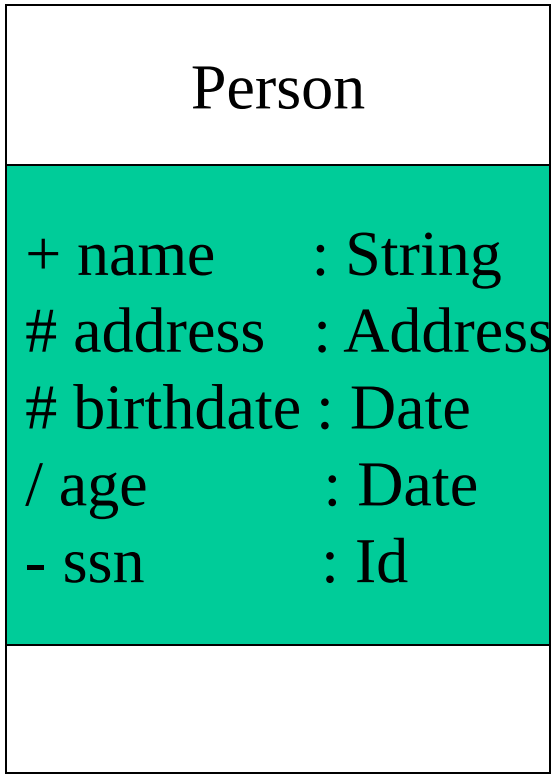
Attributes are usually listed in the form:

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)

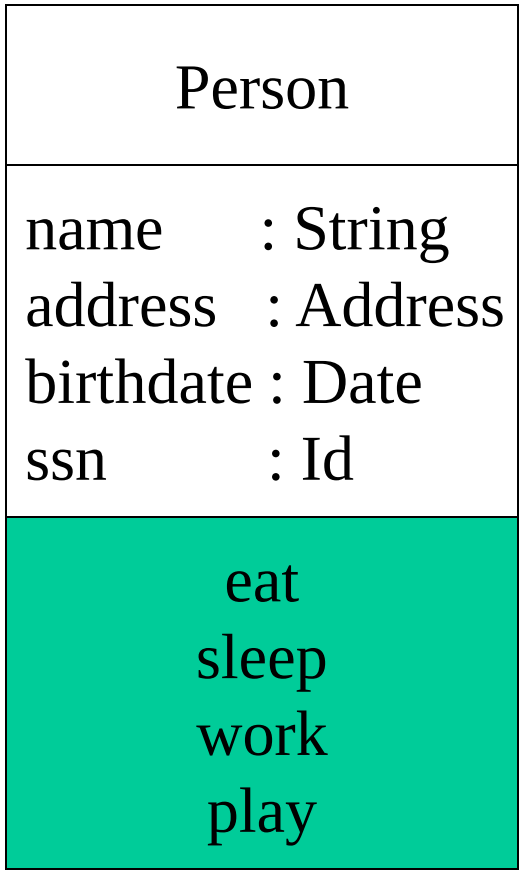


Attributes can be:

+ public
# protected
- private
/ derived

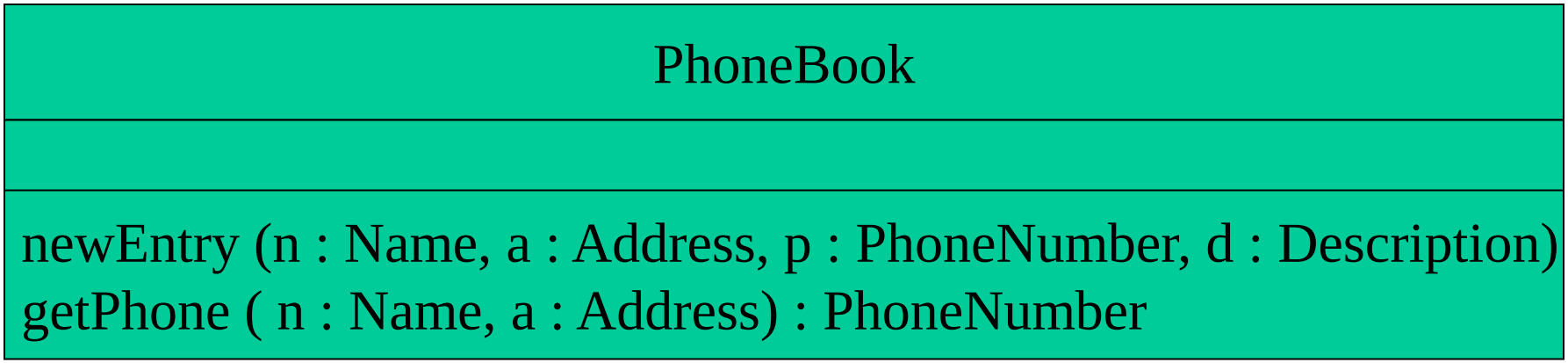
Static (underlined)

Class Operations



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

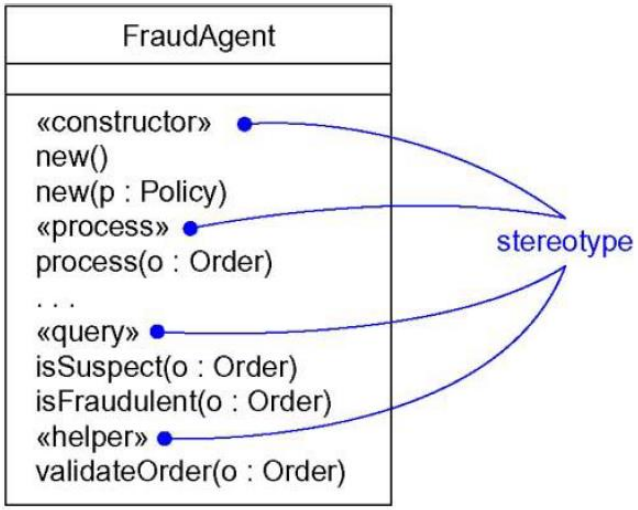
Class Operations (Cont'd)



You can specify an operation by stating its signature: listing the name, type, and default value of all parameters, and, in the case of functions, a return type.

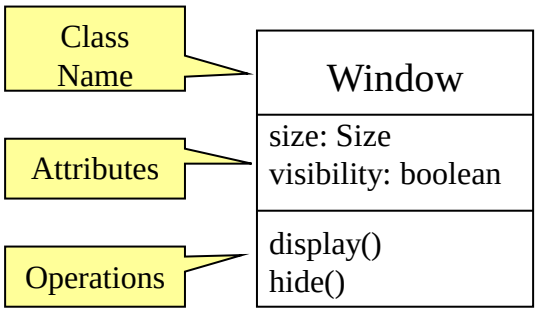
Organizing Attributes and Operations

- Don’t have to show every attribute and operation at once
- End each list with ellipsis (…)
- Can also prefix each group with a descriptive category using stereotypes



Class

- Attributes and operations may
  - have their visibility marked:
  - "+" for *public*
  - "#" for *protected*
  - "–" for *private*
  - "~" for *package*

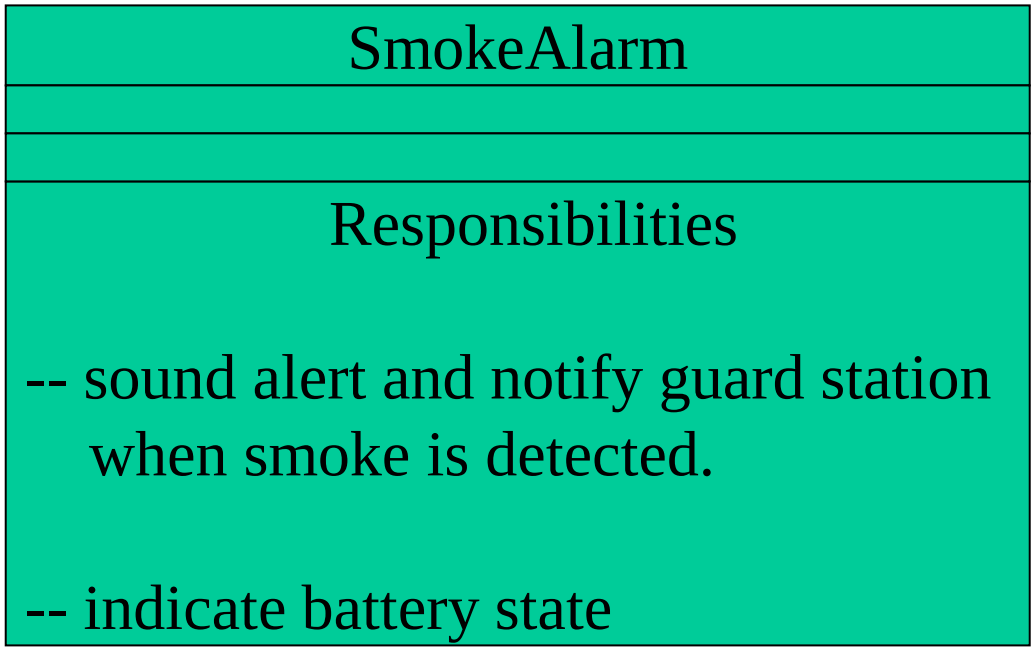


Class Names

- Name must be unique within its enclosing package
- Simple name and Path name
- Class names are noun phrases from the vocabulary of the system

Class Responsibilities

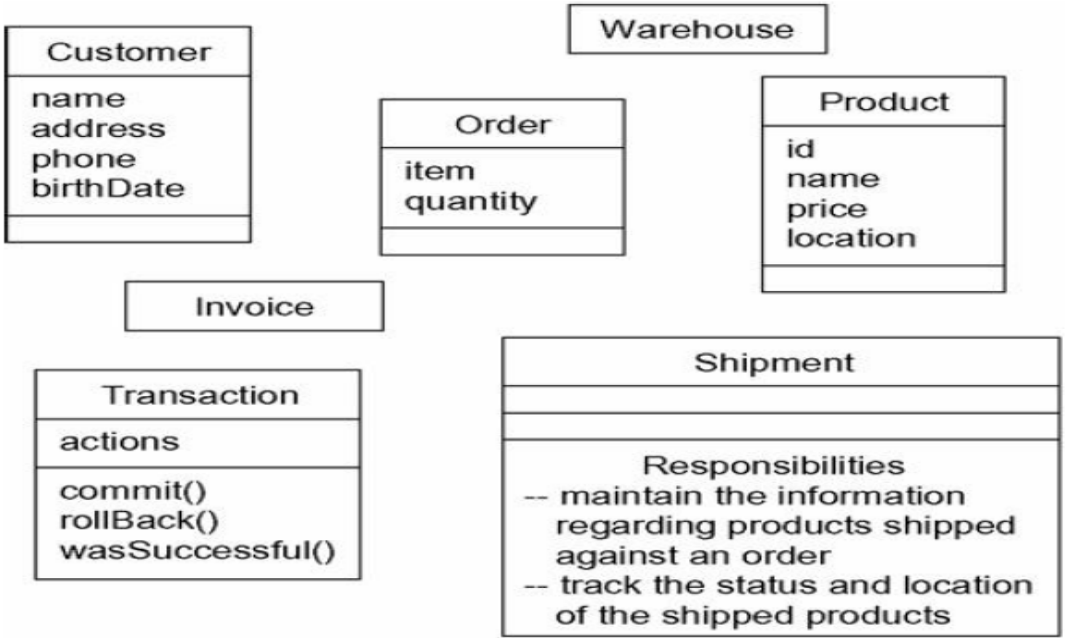
- A class may also include its responsibilities in a class diagram.
- A responsibility is a contract or obligation of a class to perform a particular service.
- Attributes and operations are the features that carry out the class's responsibility



Techniques used - CRC cards and use case based-analysis

Modeling the Vocabulary of a System

- Identify the things that describe the problem
- For each abstraction, identify a set of responsibilities
- Provide the attributes and operations needed to carry out those responsibilities

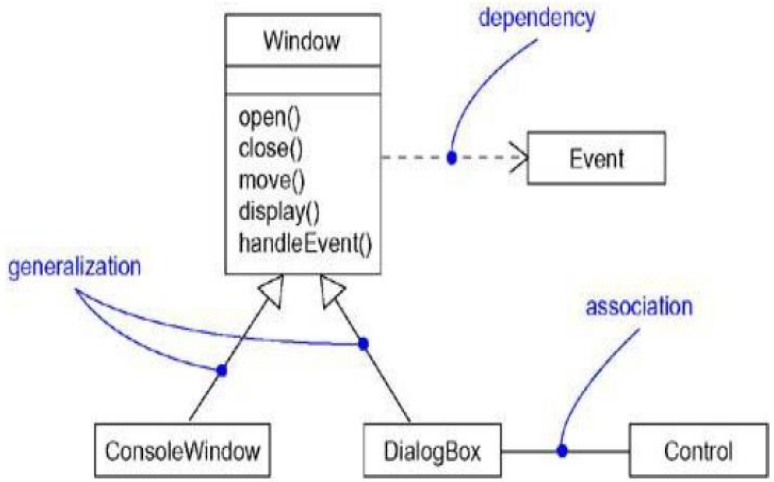


Relationships

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

There are three kinds of relationships in UML:

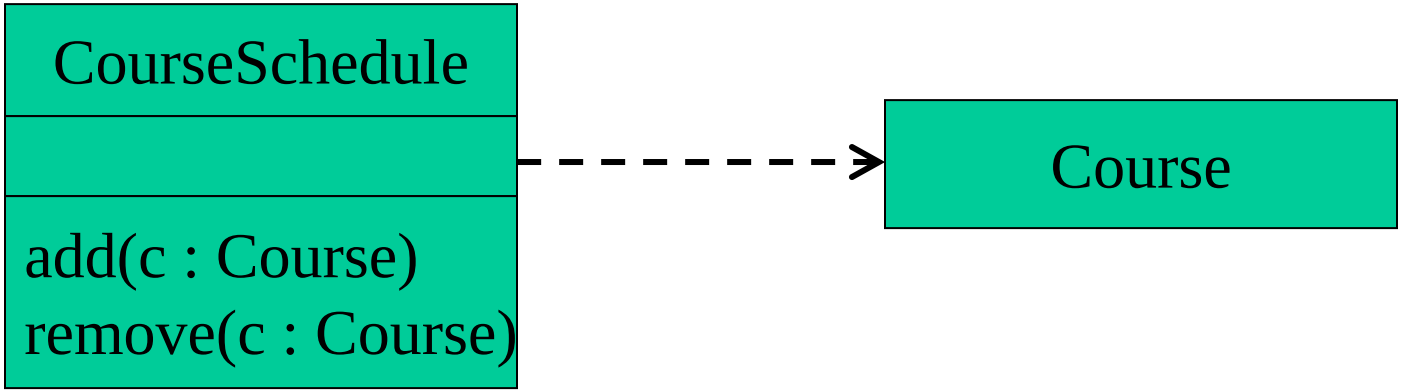
- dependencies
- generalizations
- associations



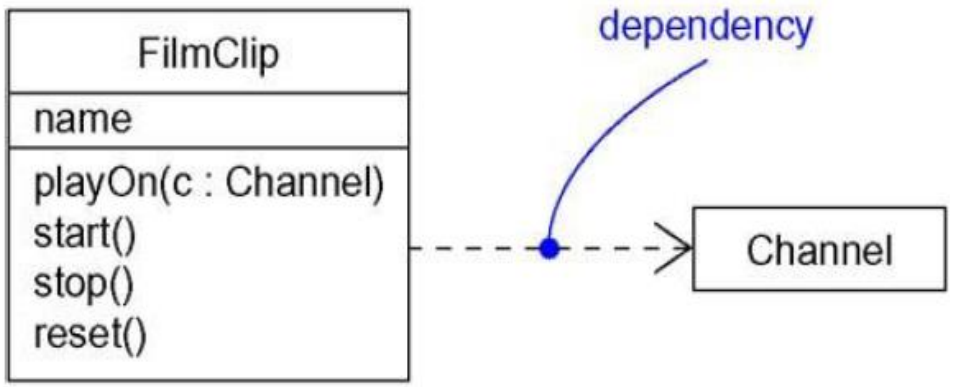
Dependency Relationships

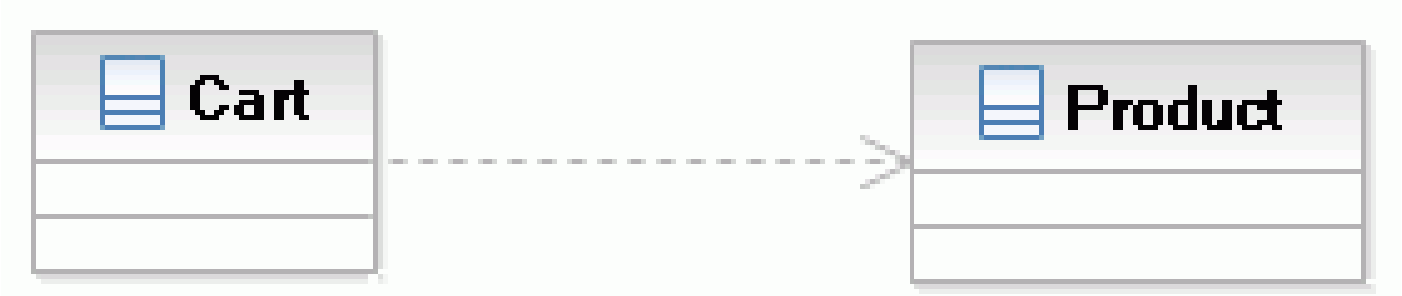
A *dependency* indicates a semantic relationship between two or more elements.

It is a special type of association (using) exists between two classes, if changes to the definition of one may cause changes to the other (but not the other way around).
The dependency from *CourseSchedule* to *Course* exists because *Course* is used in both the **add** and **remove** operations of *CourseSchedule*.

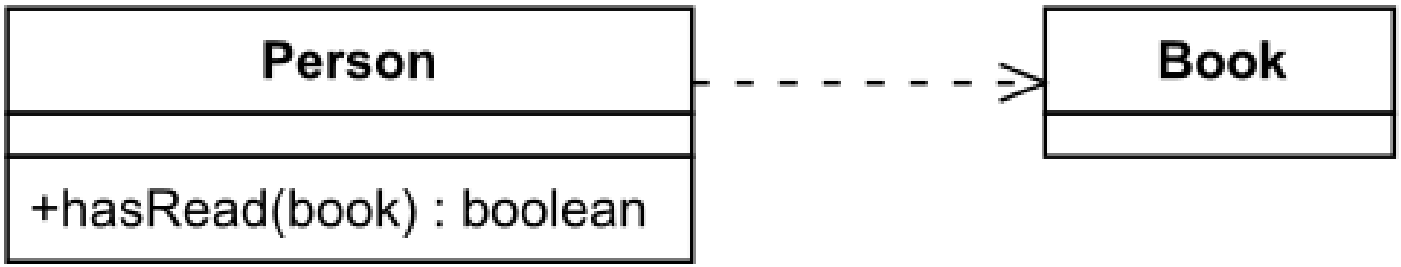


Dependency Relationships



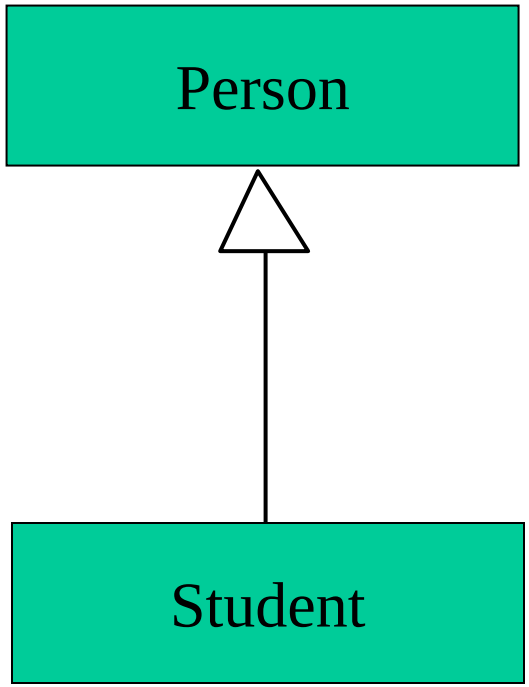


Dependency Relationships



Dependency is a weaker form of relationship which indicates that one class depends on another because it uses it at some point in time.

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.

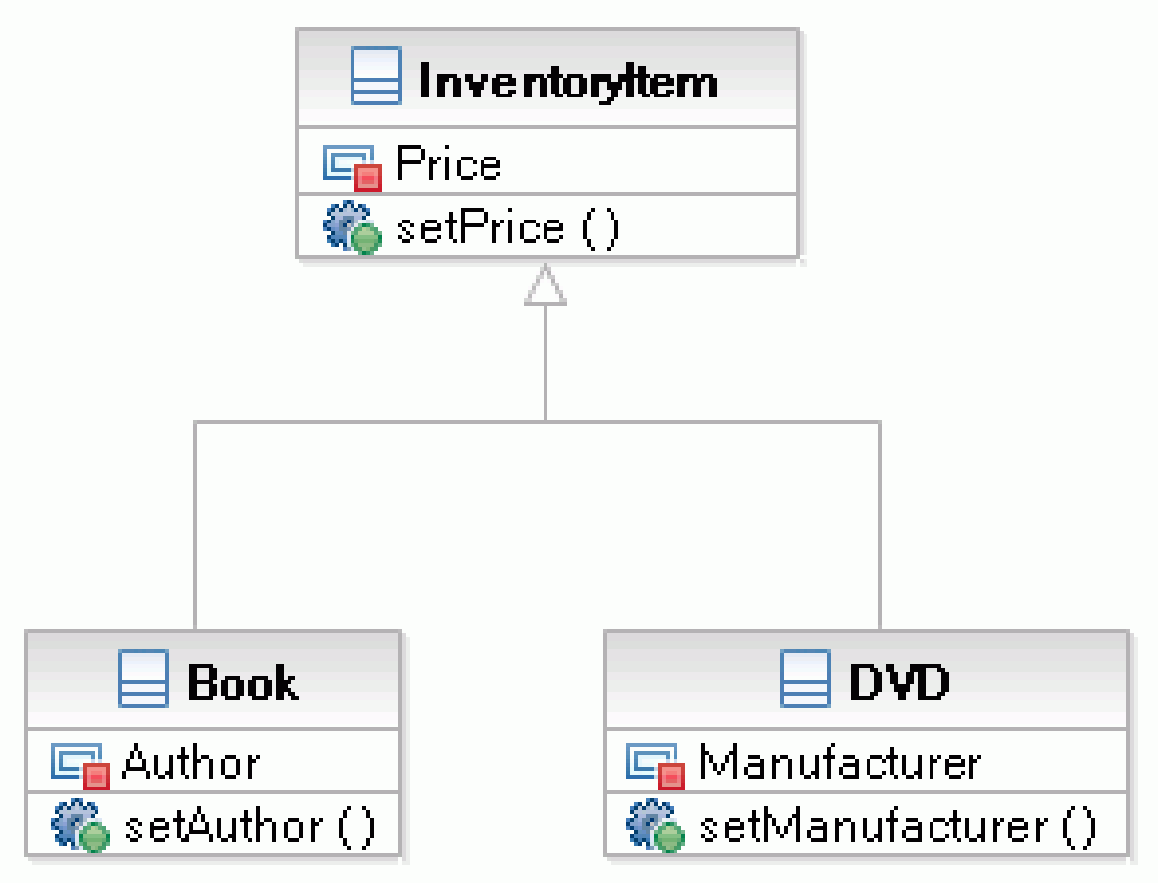
The specific classifier inherits the features of the more general classifier.

Is-a-kind-of" relationship

Polymorphism:

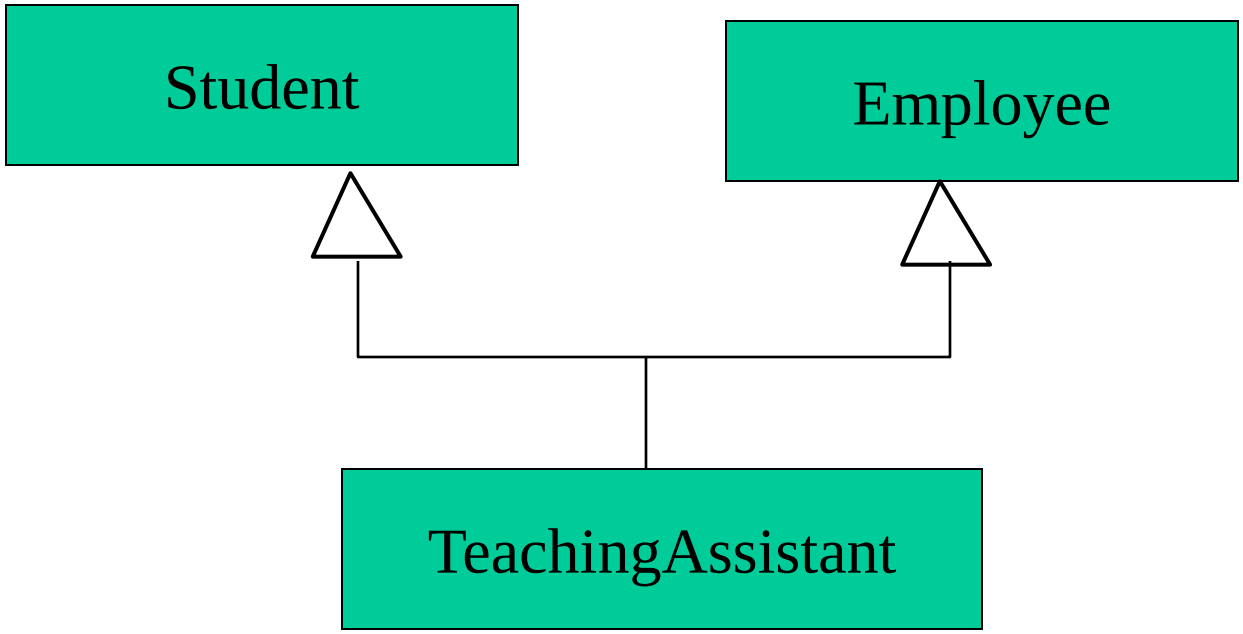
Operation of child has the same signature as an operation in the parent but overrides it

Generalization Relationships



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.



Generalization

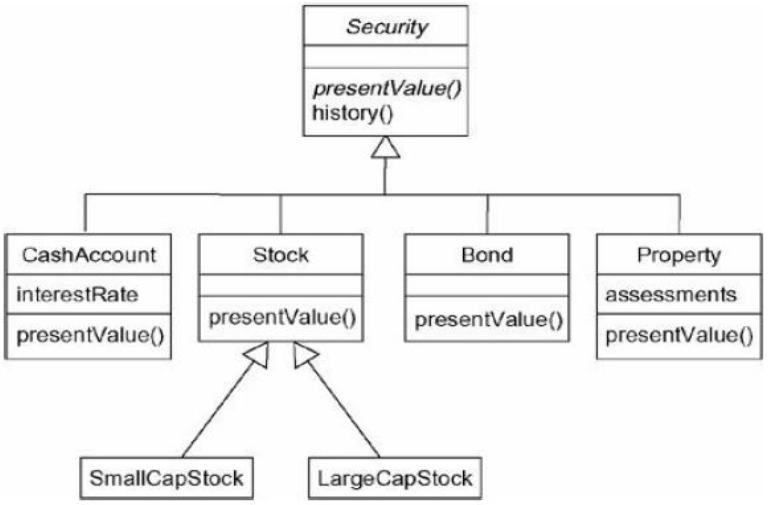
- A sub-class inherits from its super-class
  - Attributes
  - Operations
  - Relationships

- A sub-class may
  - Add attributes and operations
  - Add relationships
  - Refine (override) inherited operations

- A generalization relationship **may not** be used to model interface implementation.

Modeling Single Inheritance

- Find classes that are structurally or behaviorally similar while modeling the vocabulary
- To model inheritance
  - Look for common responsibilities
  - Elevate these to a more general class
  - Specify that the more specific class inherits from the more general class



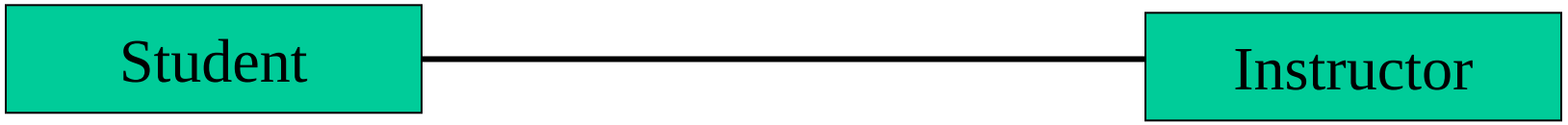
Association Relationships

Associations are relationships between classes in a UML Class Diagram. They are represented by a solid line (Bi-directional) between classes.

An *association* denotes that link to each other

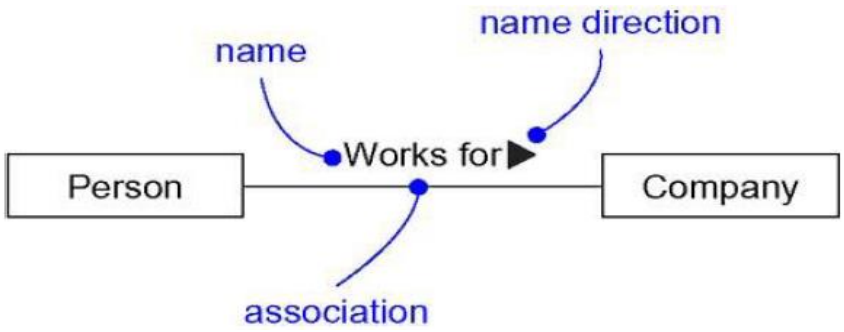
Specifies that objects of one thing are connected to objects of another

Type: Bi-directional, Uni-directional, Self Association



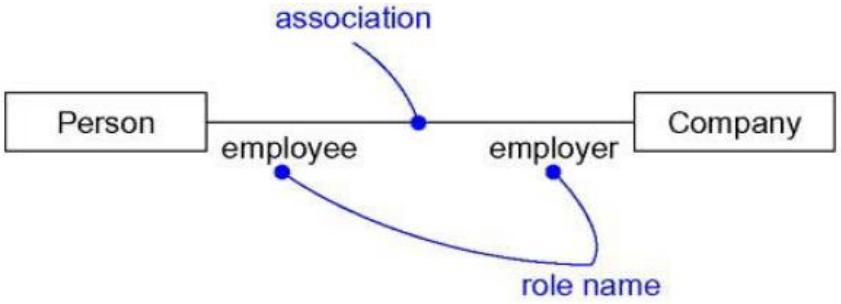
Association : Adornments

- Name
  - Use a name to describe the nature of the relationship
  - Can give a direction to the name



Association : Adornments

- Role
  - The face that the class at the far end of the association presents to the class at the near end
  - Explicitly name the role a class plays (*End name* or *Role name*)
  - Same class can play the same or different roles in other associations



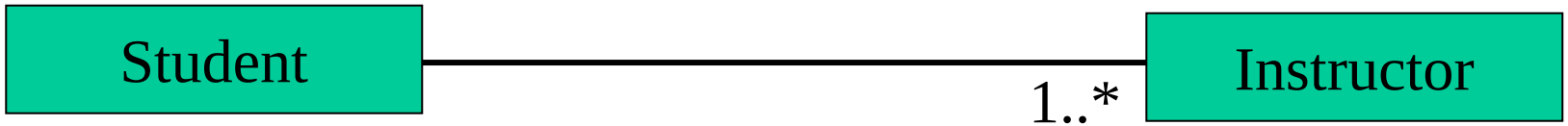
Association: Adornments

- Multipicity

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

It States how many objects may be connected across an instance of an association

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



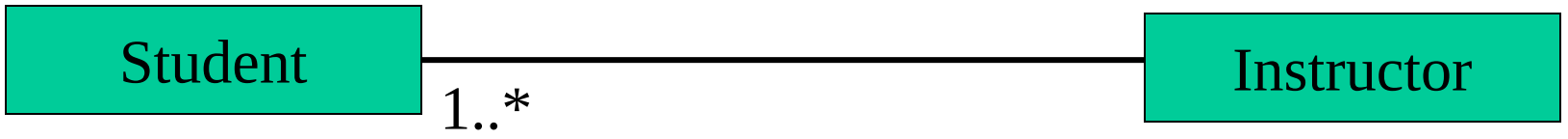
Some typical examples of multiplicity:

An expression that evaluates to a range of values or an explicit value

Multiplicity	Cardinality
0..0	Collection must be empty
0..1	No instances or one instance
1..1 or 1	Exactly one instance
0..*	Zero or more instances
1..*	At least one instance
5..5 or 5	Exactly 5 instances
m..n	At least m but no more than n instances

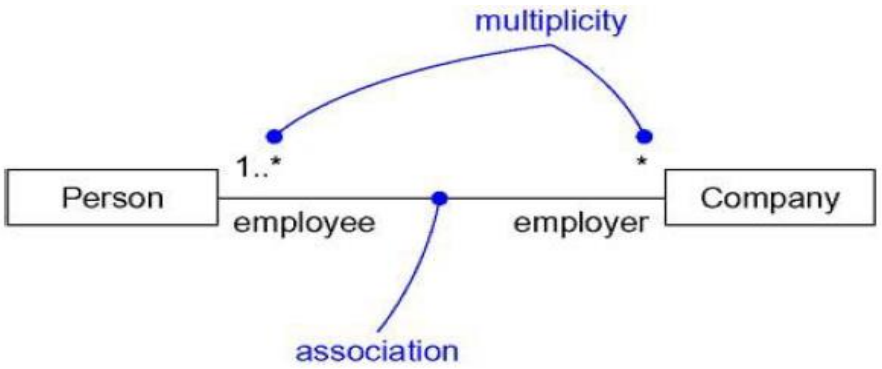
Association Relationships: Adornments

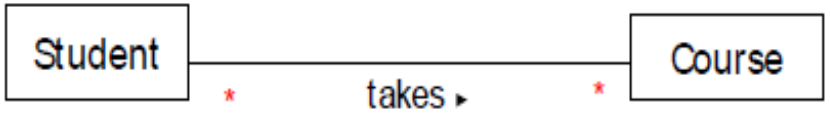
The example indicates that every *Instructor* has one or more *Students*:



Multiplicity

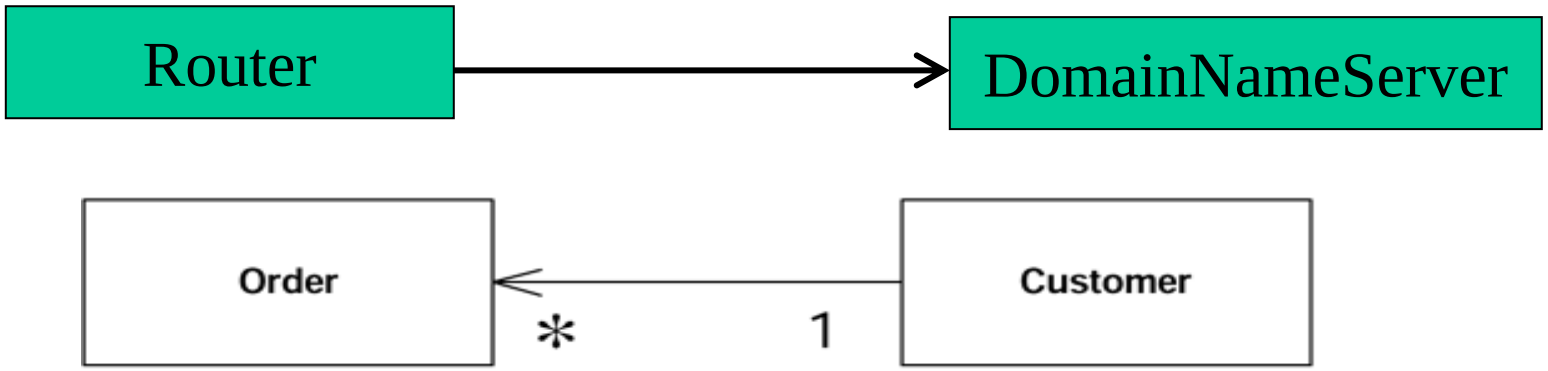
– Specify complex multiplicities by using a list, 0..1, 3..4, 6..* (any number of objects other than 2 or 5)





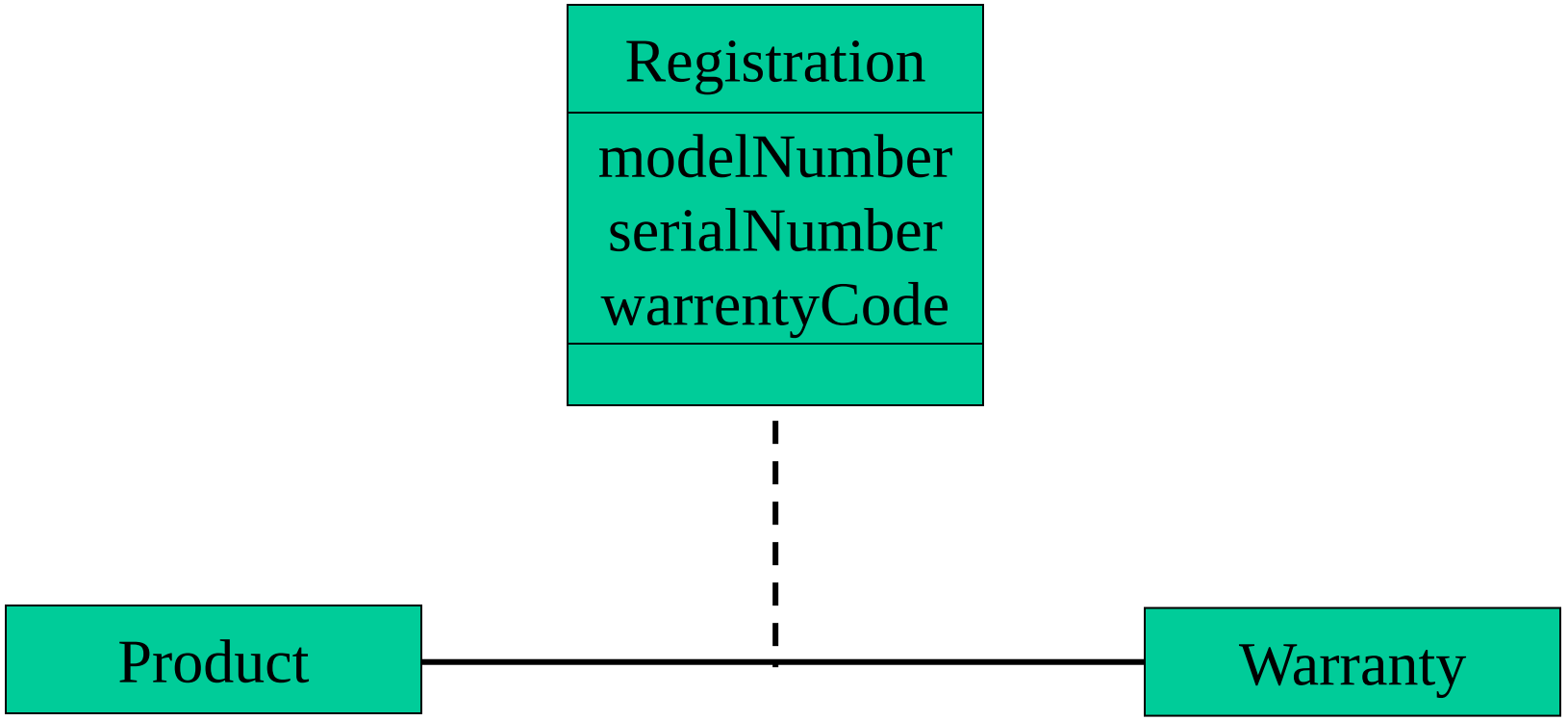
Association Relationships (Unidirectional)

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*.

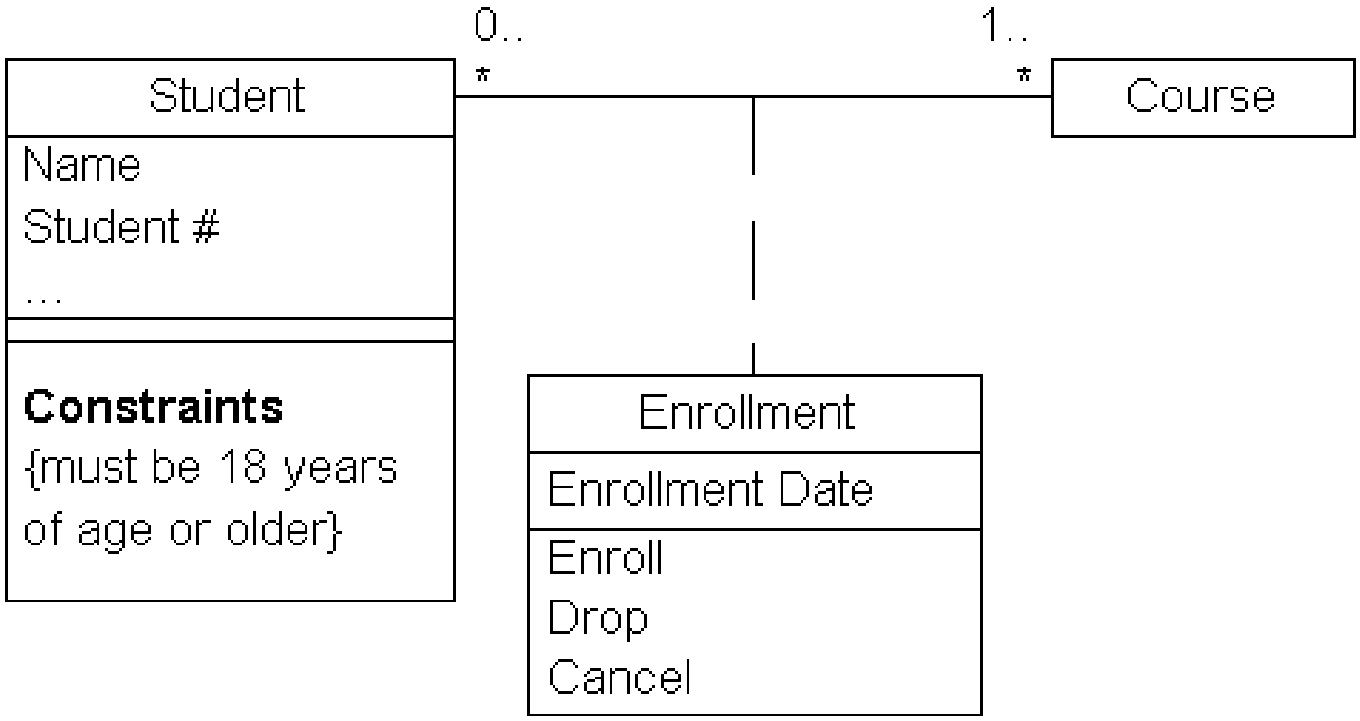


Association Relationships (Cont'd)

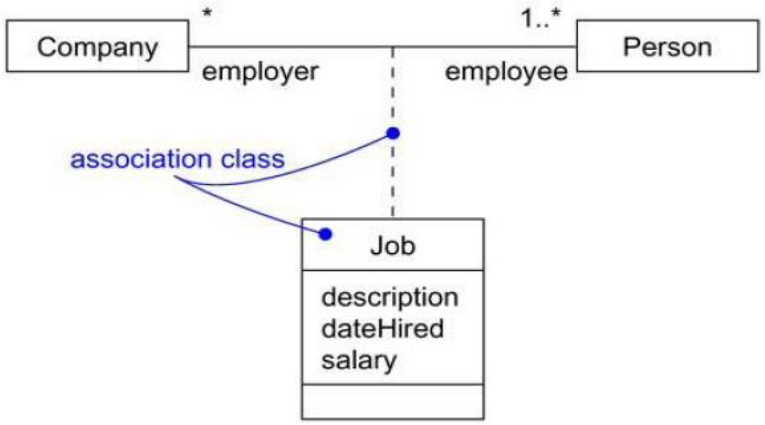
Associations can also be objects themselves, called *link classes* or an *association classes*.



Association Relationships (Cont'd)



Association Relationships (Cont'd)

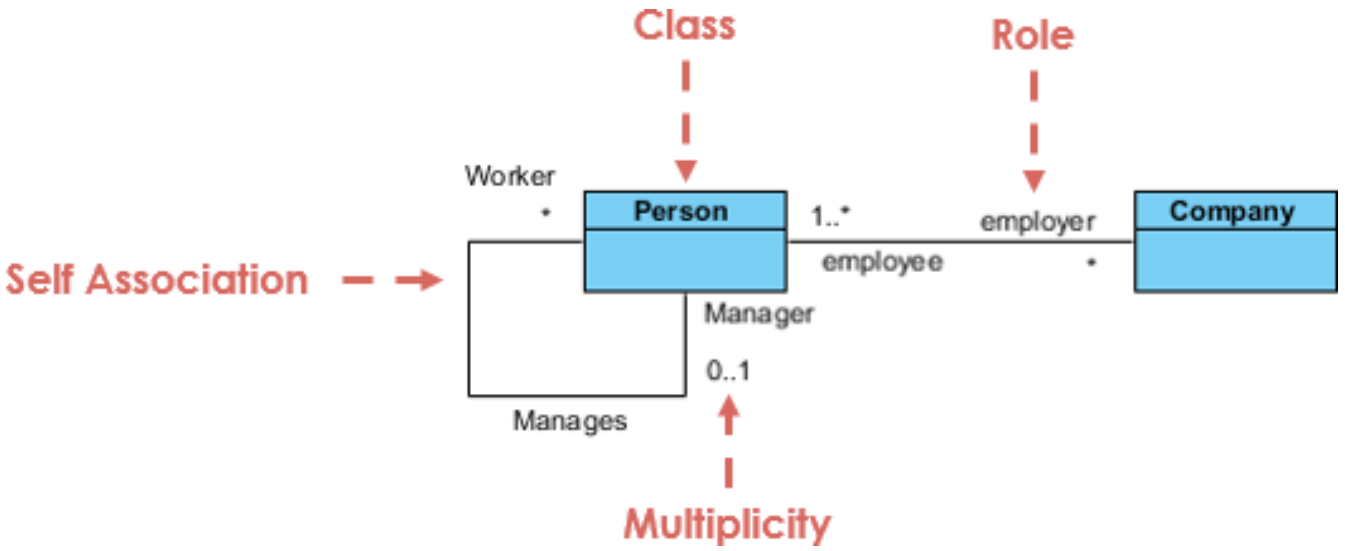


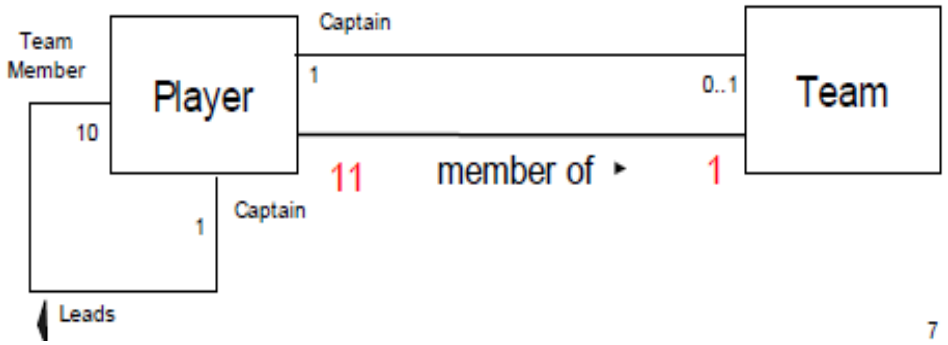
Ex: In the relationship between a Company and a Person, there is Job that represent the properties of that relationship

Represents exactly one pairing of Person and Company

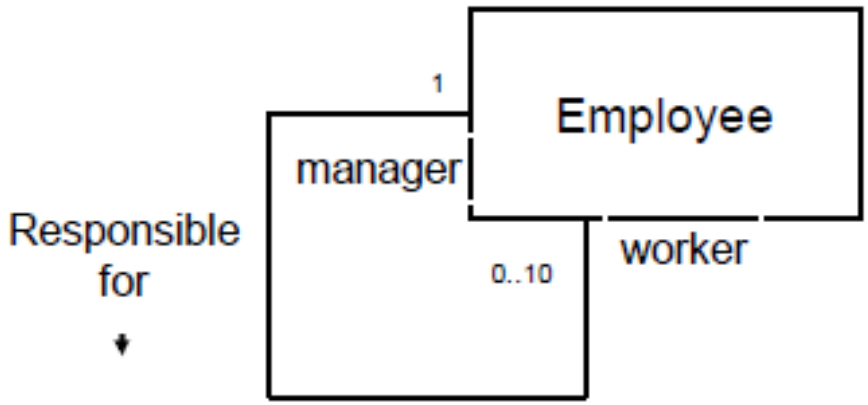
Association Relationships (Cont'd)

A class can have a ***self association***.



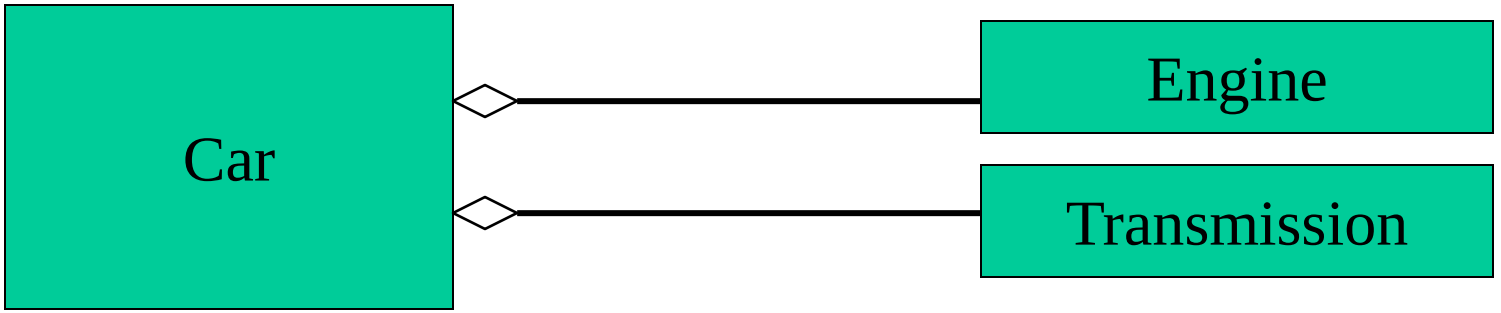


self association

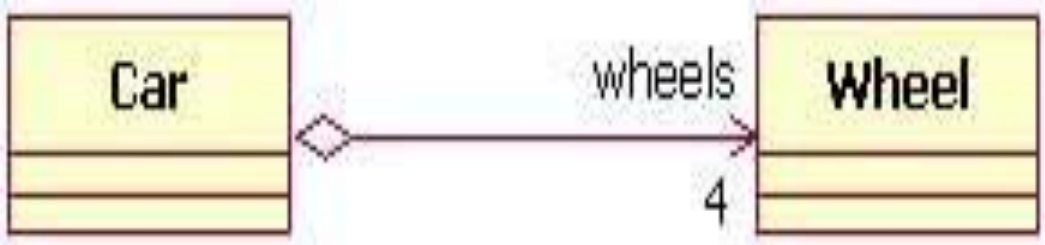


Association Relationships (Cont'd)

- 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.
- Represents a **"has-a"** relationship



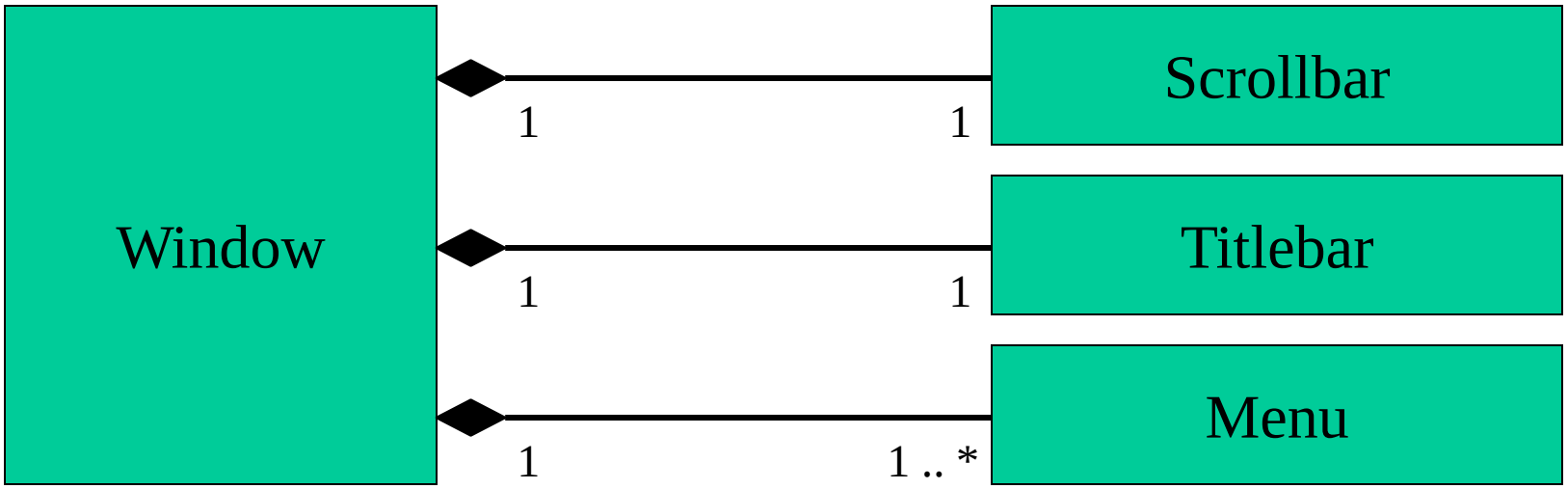
Aggregation

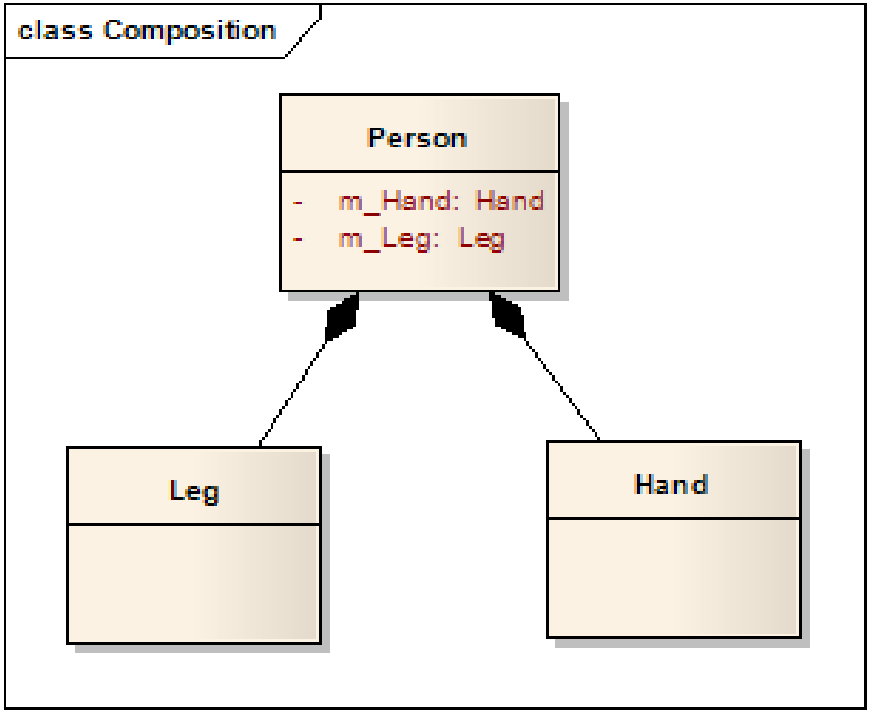


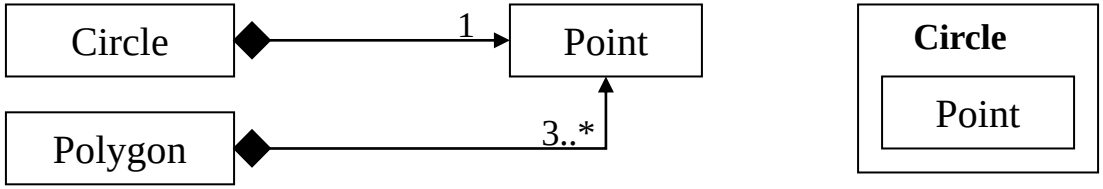
A door **is** part of a car. A car **is not** part of a door.

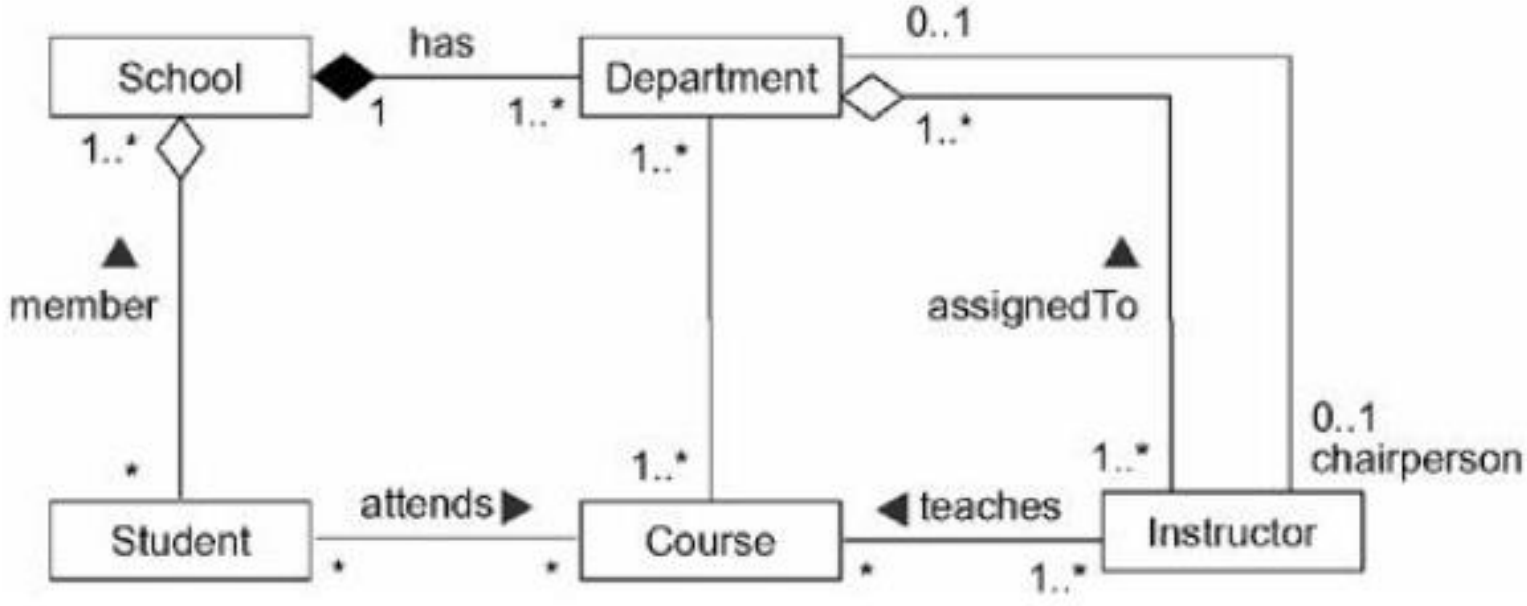
Association Relationships (Cont'd)

A ***composition*** **(A strong form of aggregation)** 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.




class Composition
Person
- m_Hand: Hand
- m_Leg: Leg
Leg
Hand


Circle
1
Point
Polygon
3..*
Circle
Point


School
has
Department
0..1
1
1..*
1..*
1..*
1..*
member
assignedTo
*
1..*
1..*
0..1
chairperson
Student
attends
Course
teaches
Instructor
*
*
*
1..*

Aggregation and Composition: Key Differences

1.Existence Dependency:

1. In aggregation, the "part" class can exist independently of the "whole" class.
2. In composition, the "part" class typically cannot exist independently; it's dependent on the "whole" class.

2.Strength of Relationship:

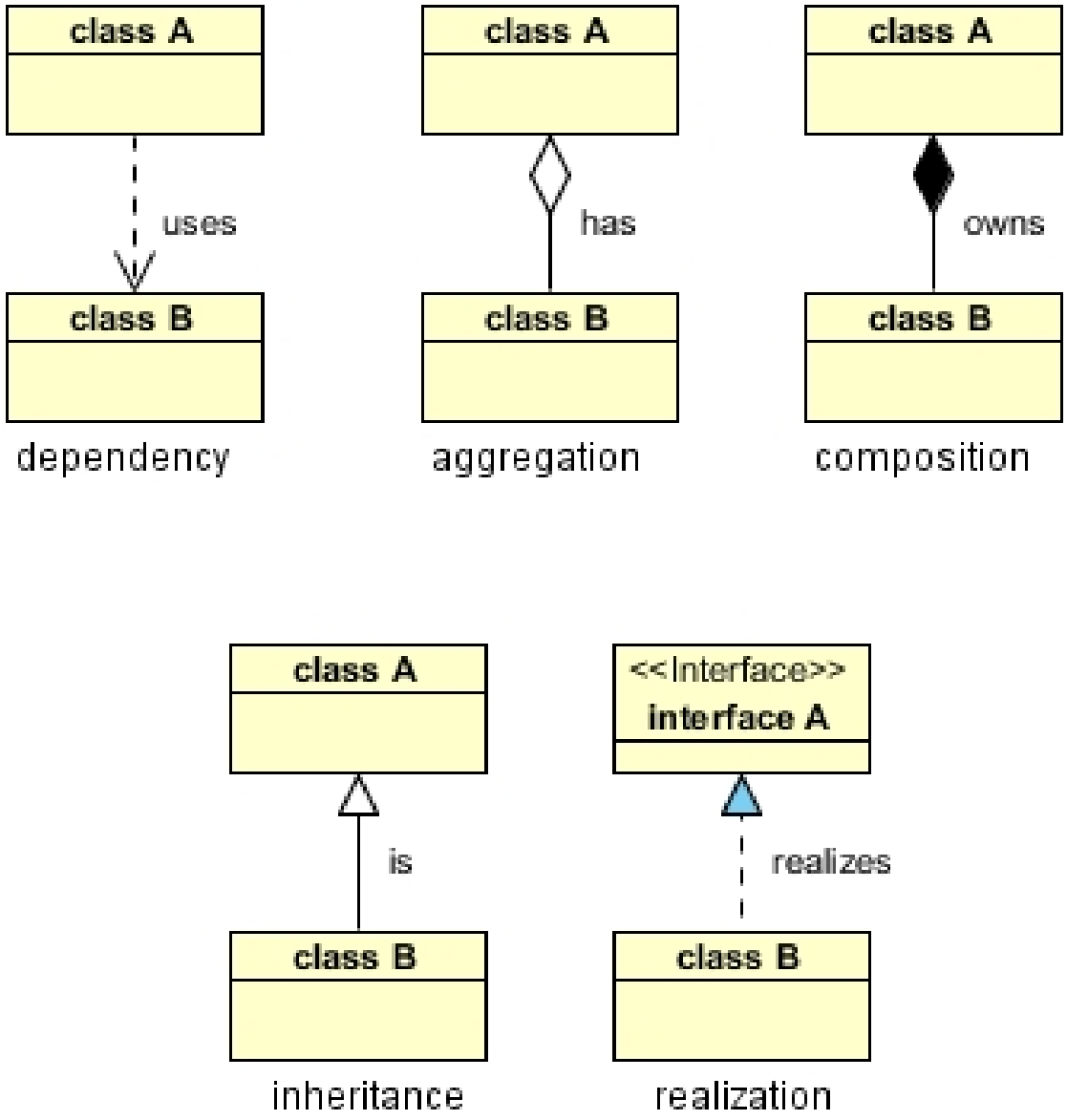
1. Aggregation represents a weaker form of association, where the "part" can be associated with multiple "wholes."
2. Composition represents a stronger form of association, indicating a strict ownership relationship where a "part" belongs exclusively to a single "whole."

3.Notation:

1. Aggregation is denoted by an unfilled diamond shape.
2. Composition is denoted by a filled diamond shape.

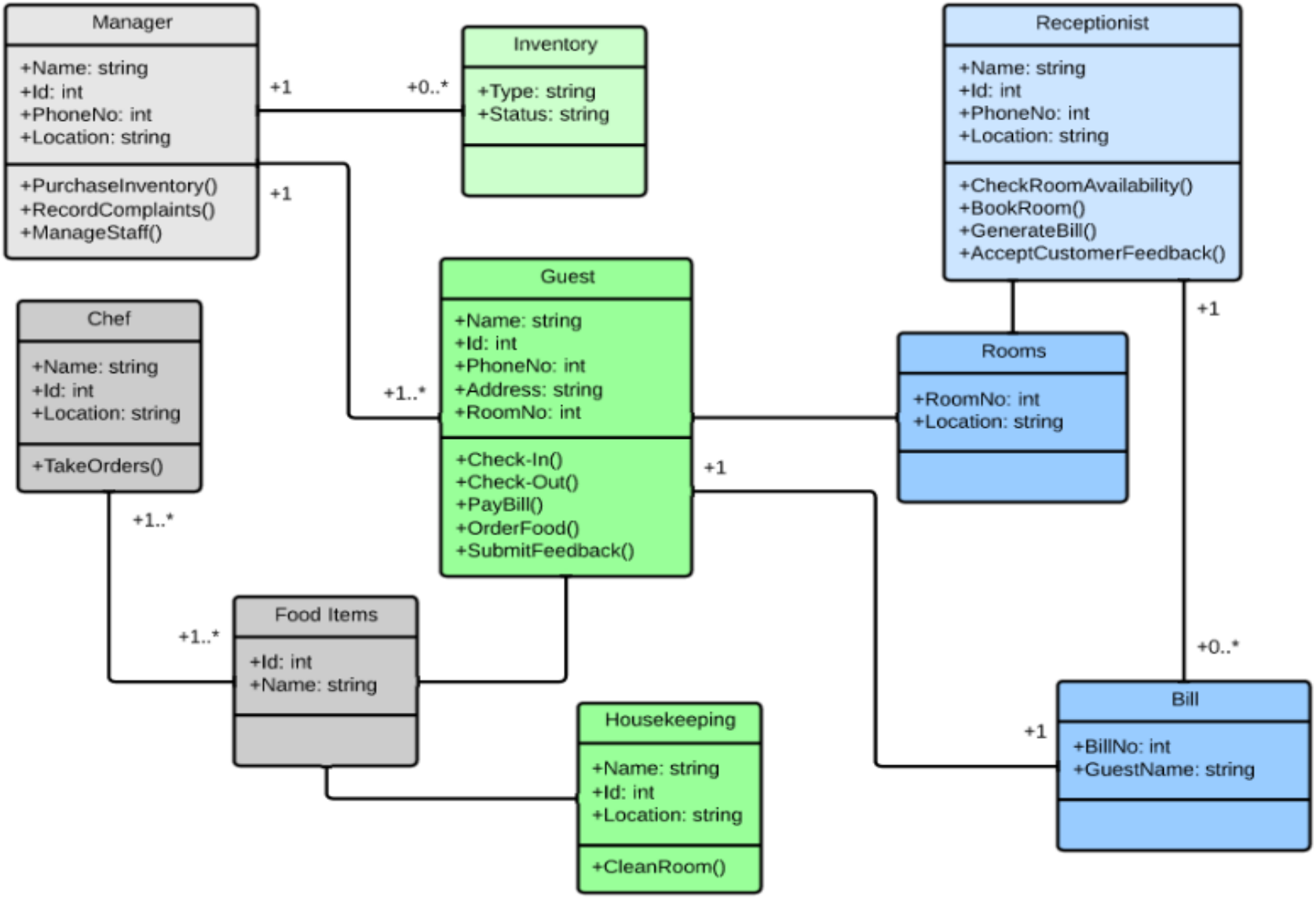
Realization

- A realization relationship indicates that one class implements a behavior specified by another class (an interface or protocol).
- An interface can be realized by many classes.
- A class may realize many interfaces.


class A
uses
class B
dependency
class A
has
class B
aggregation
class A
owns
class B
composition
class A
is
class B
inheritance
<<Interface>>
interface A
realizes
class B
realization

Example Class Diagram –

hotel management system



Example Class Diagram - ATM system