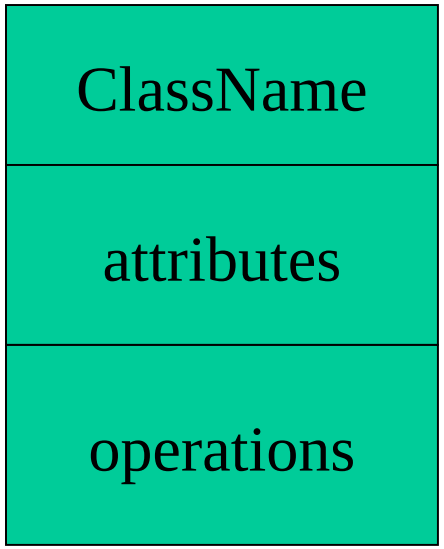
Software Design

Class Diagram

Material based on
[Booch99, Rambaugh99, Jacobson99, Fowler97, Brown99]

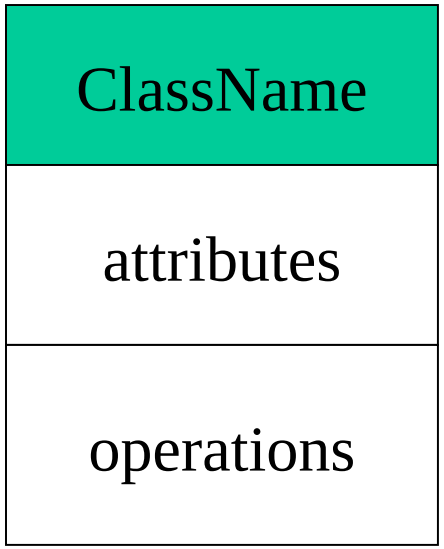
Classes



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

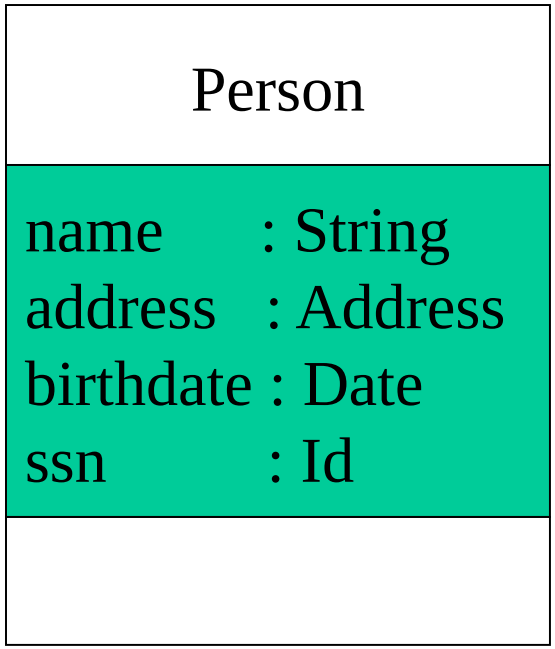
Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations in separate, designated compartments.

Class Names



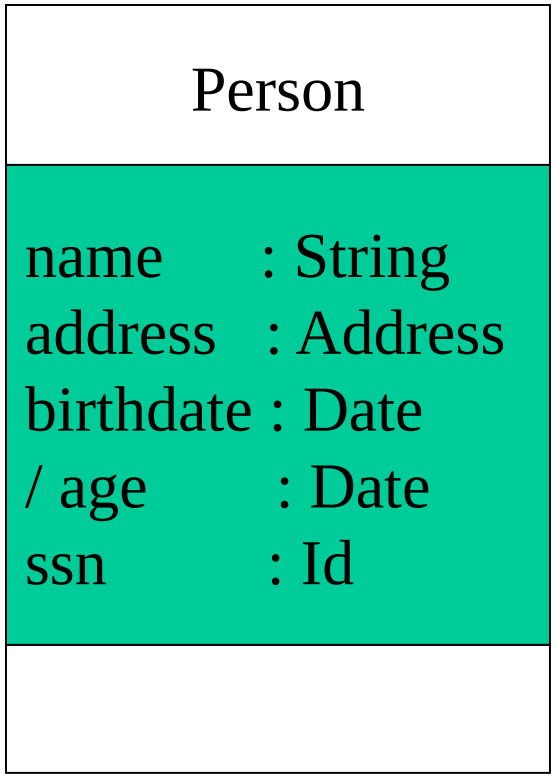
The name of the class is the only required tag in the graphical representation of a class. It always appears in the top-most compartment.

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)



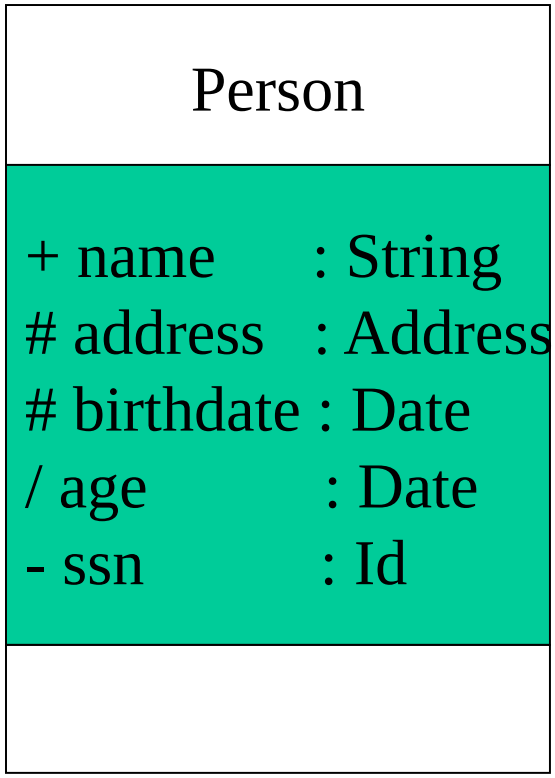
Attributes are usually listed in the form:

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

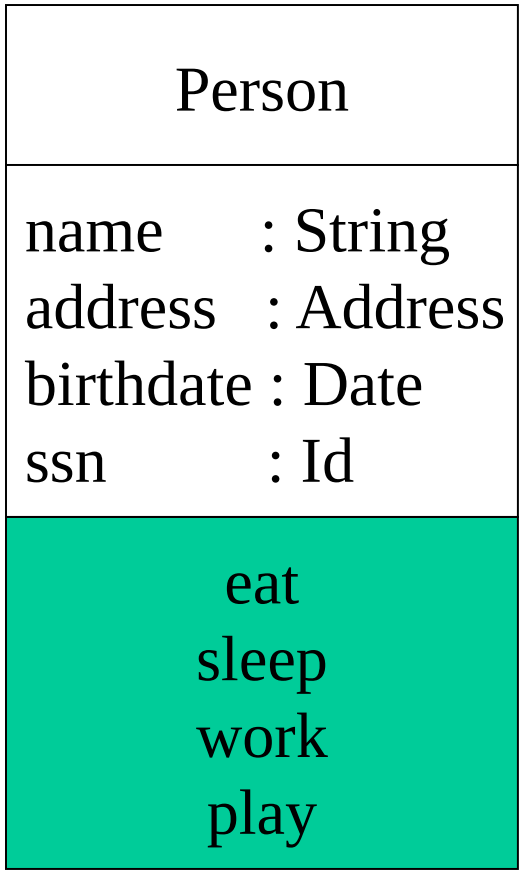
Class Attributes (Cont'd)



Attributes can be:

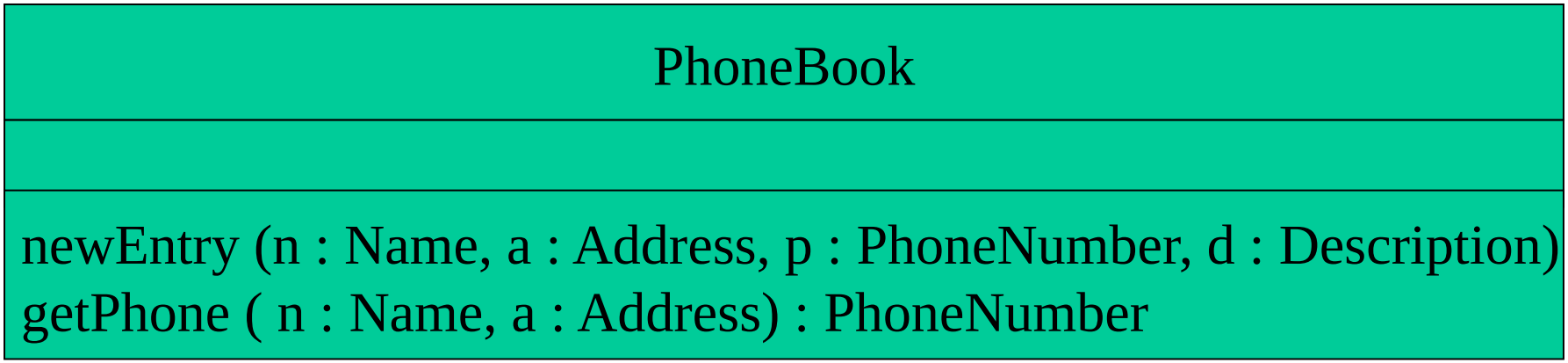
+ public
# protected
- private
/ derived

Class Operations



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

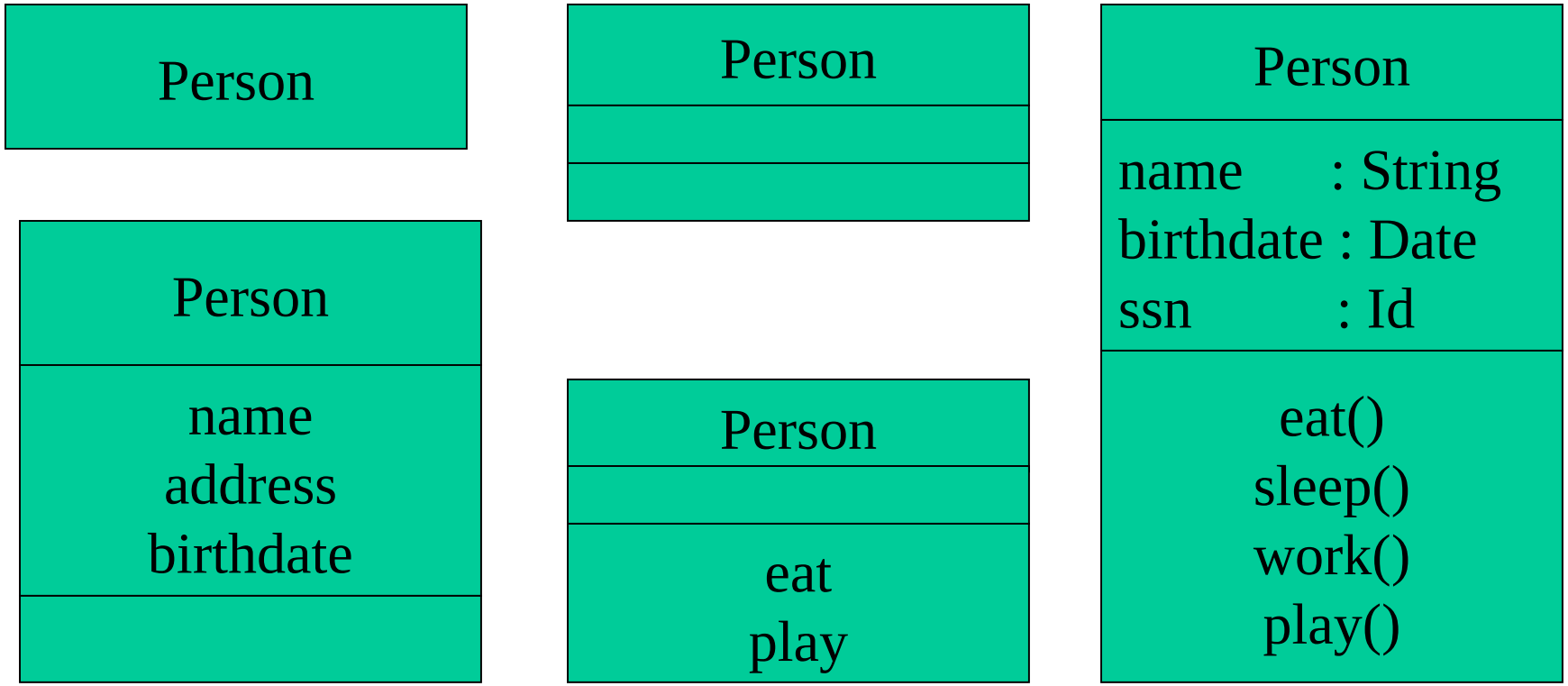
Class Operations (Cont'd)



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.

Depicting Classes

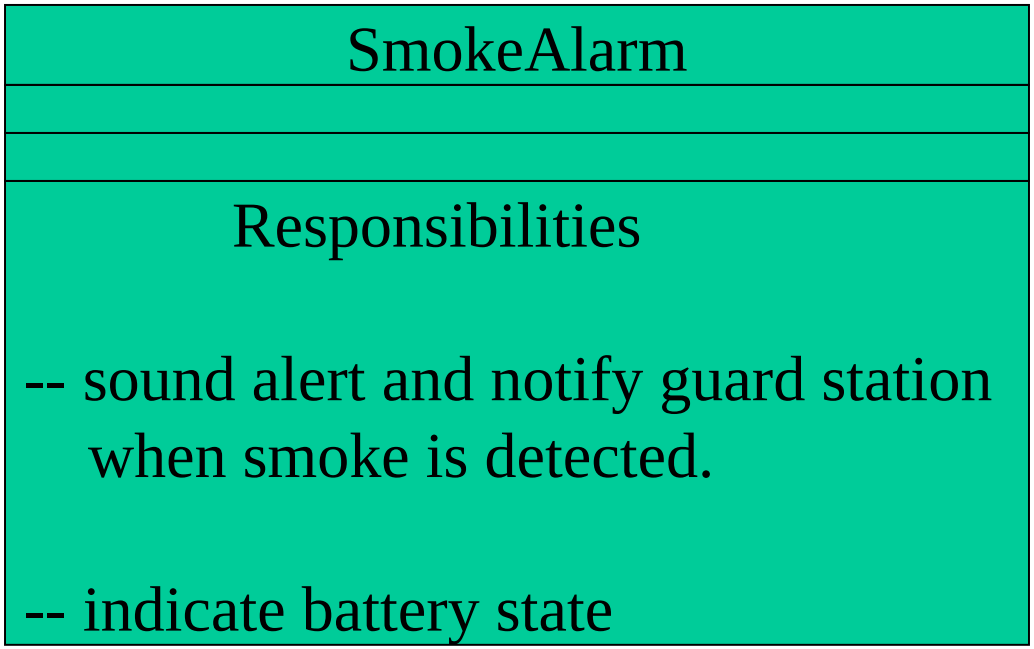
When drawing a class, you needn't show attributes and operation in every diagram.



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.



Relationships

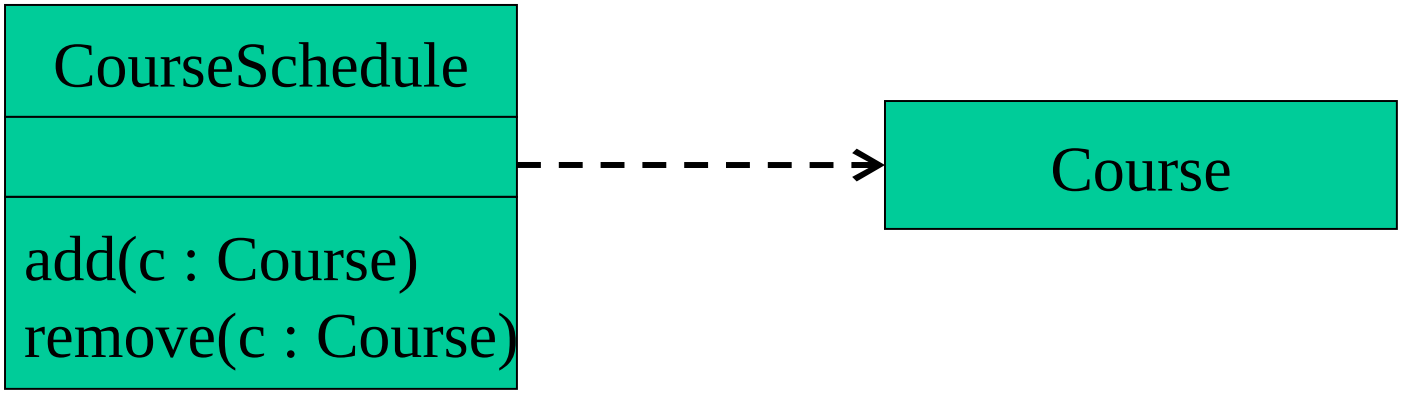
In UML, object interconnections (logical or physical), are modeled as relationships.

There are three kinds of relationships in UML:

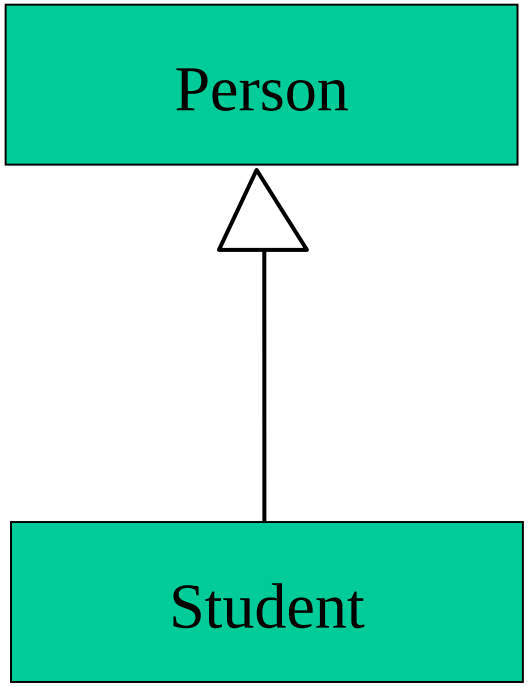
- dependencies
- generalizations
- associations

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*.



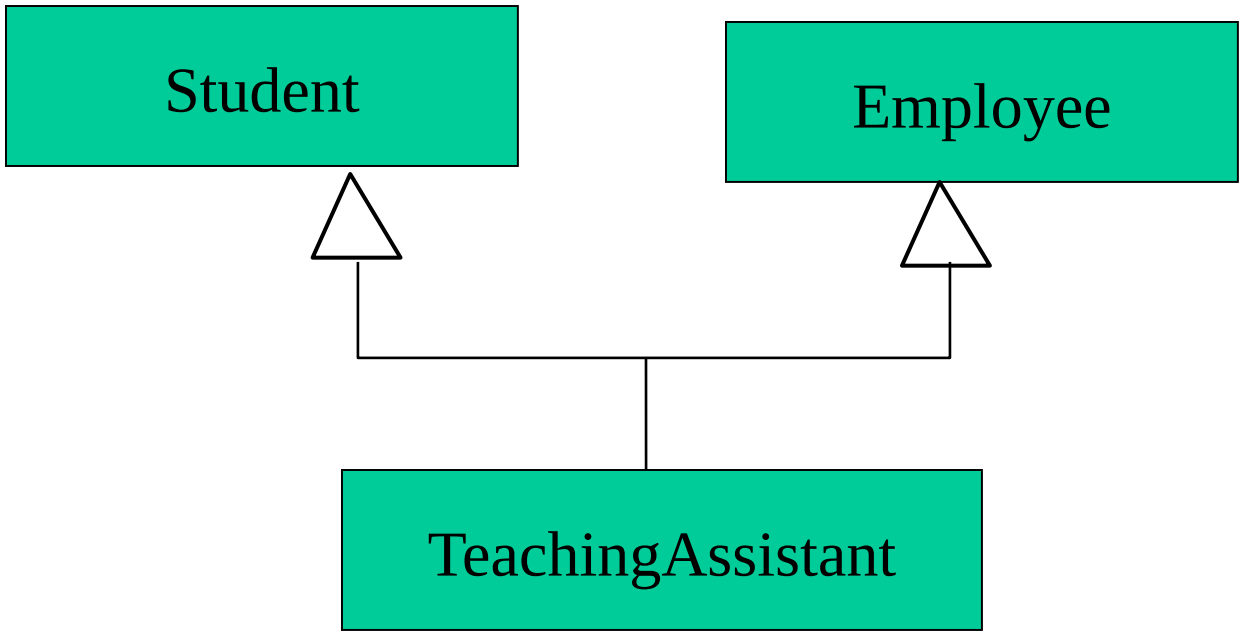
Generalization Relationships



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

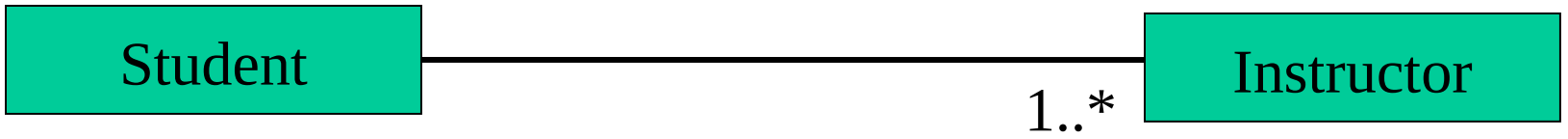
If two classes in a model need to communicate with each other, there must be link between them.

An *association* denotes that link.

Association Relationships (Cont'd)

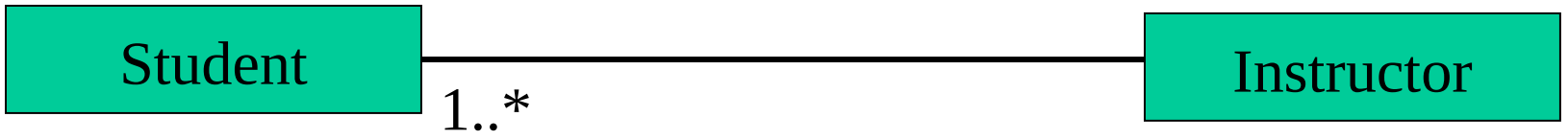
We can indicate the *multiplicity* of an association by adding *multiplicity adornments* to the line denoting the association.

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



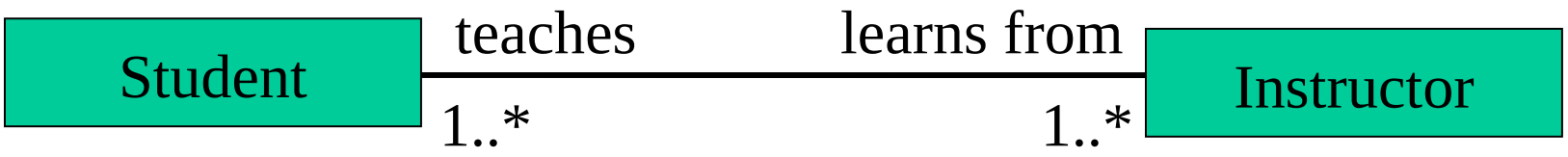
Association Relationships (Cont'd)

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



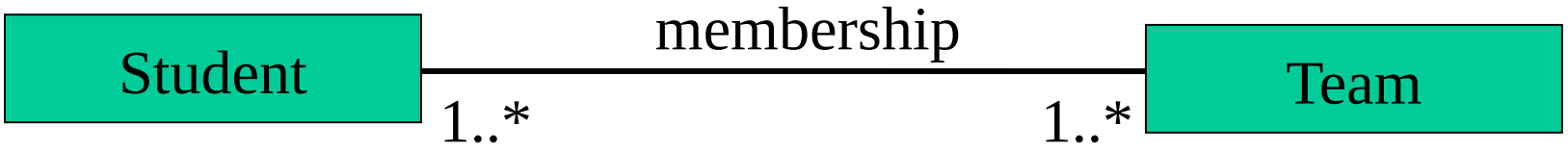
Association Relationships (Cont'd)

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



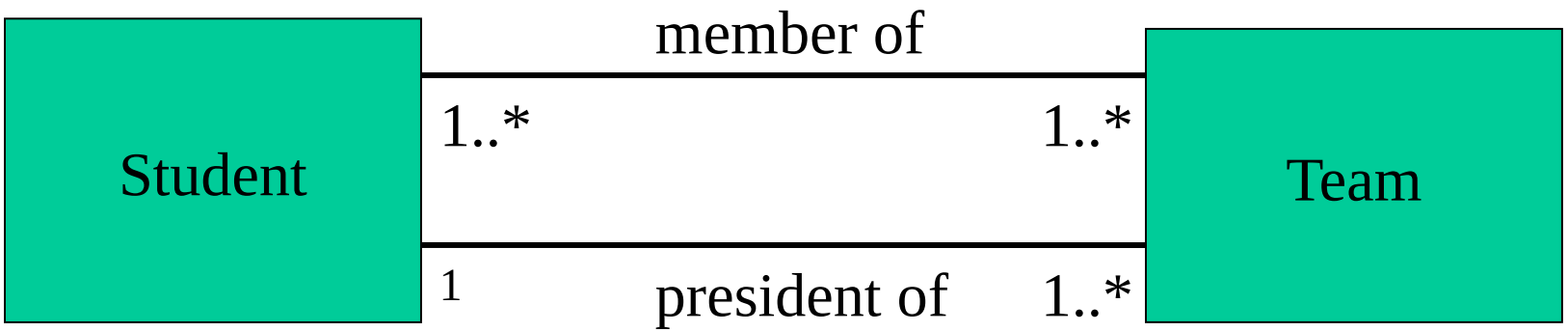
Association Relationships (Cont'd)

We can also name the association.



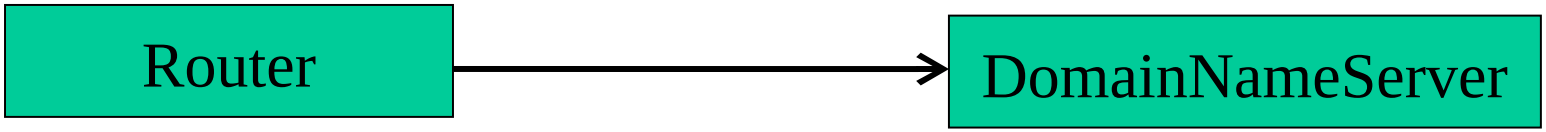
Association Relationships (Cont'd)

We can specify dual associations.



Association Relationships (Cont'd)

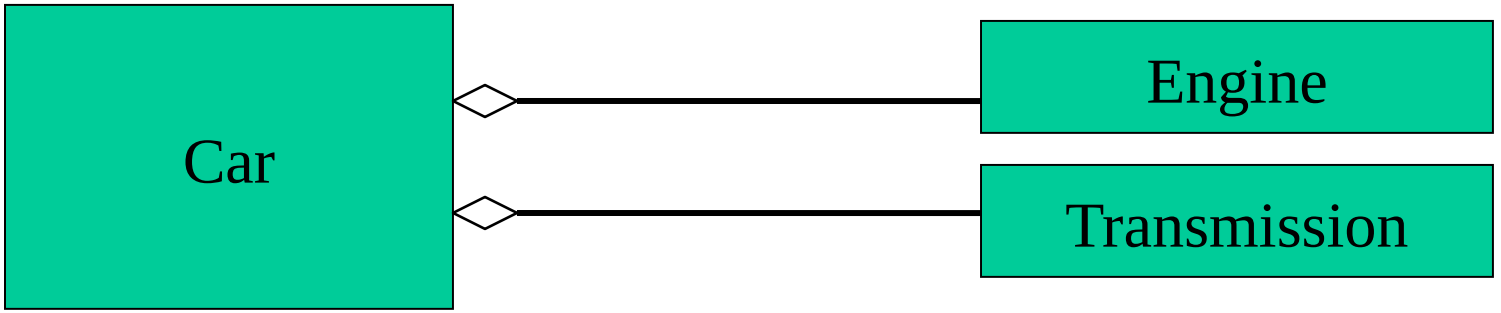
We can constrain the association relationship by defining the *navigability* of the association. Here, a *Router* object requests services from a *DNS* object by sending messages to (invoking the operations of) the server. The direction of the association indicates that the server has no knowledge of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



Association Relationships (Cont'd)

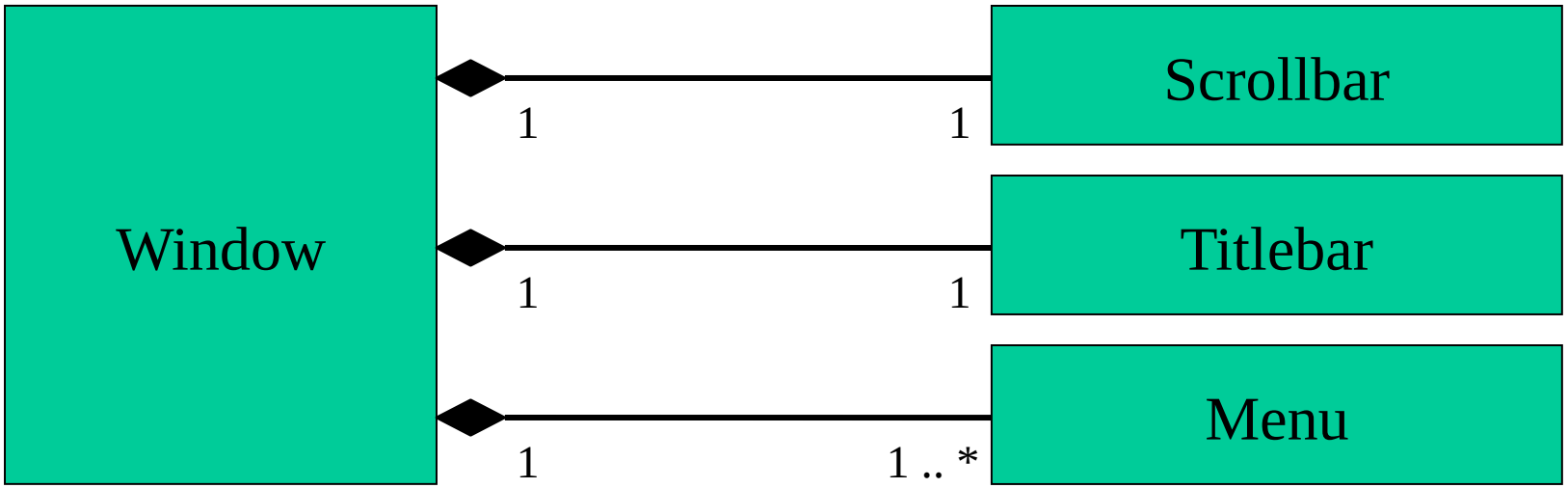
We can model objects that contain other objects by way of special associations called *aggregations* and *compositions*.

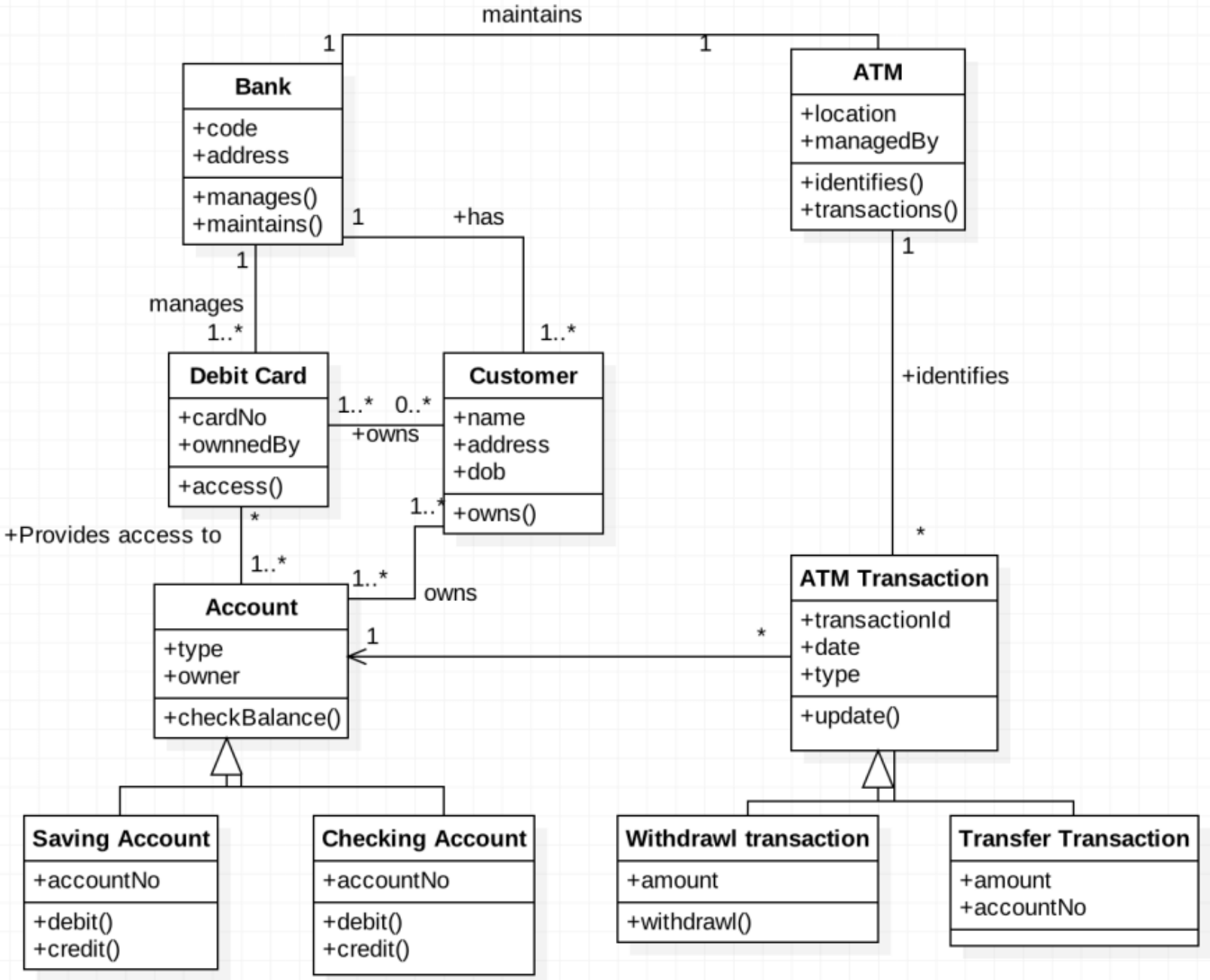
An *aggregation* specifies a whole-part relationship between an aggregate (a whole) and a constituent part, where the part can exist independently from the aggregate. Aggregations are denoted by a hollow-diamond adornment on the association.



Association Relationships (Cont'd)

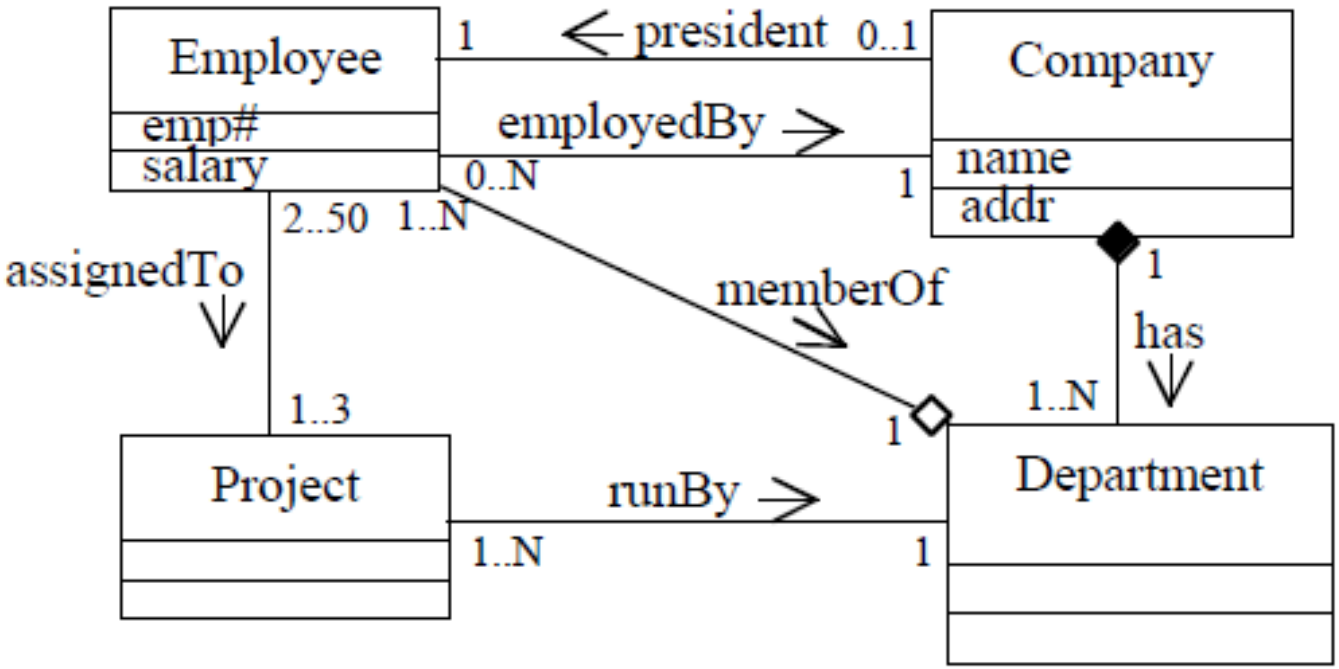
A *composition* indicates a strong ownership and coincident lifetime of parts by the whole (*i.e.*, they live and die as a whole). Compositions are denoted by a filled-diamond adornment on the association.




maintains
1
1
Bank
+code
+address
+manages()
+maintains()
ATM
+location
+managedBy
+identifies()
+transactions()
1
+has
1
1
manages
1..*
1..*
1
Debit Card
+cardNo
+ownnedBy
+access()
1..*
0..*
+owns
Customer
+name
+address
+dob
+owns()
+identifies
1..*
*
+Provides access to
1..*
1..*
owns
Account
+type
+owner
+checkBalance()
1
*
ATM Transaction
+transactionId
+date
+type
+update()
Saving Account
+accountNo
+debit()
+credit()
Checking Account
+accountNo
+debit()
+credit()
Withdrawl transaction
+amount
+withdrawl()
Transfer Transaction
+amount
+accountNo

Class Diagram

- Consider the example of companies:
- Companies employ employees(who can only work for one company), and consist of one or more departments
- Each company has a single president, who is an employee
- Departments have employees as members and run projects(one or more)
- Employees can work in 1 to 3 projects, while a project can have 2 to 50 assigned employees
- You may assume that companies have a name and address, while employees have a empId and salary
- How will you design a Class Diagram


Employee
emp#
salary
1
president
0..1
Company
name
addr
employedBy
0..N
1
2..50
1..N
assignedTo
memberOf
1
has
1..N
1..3
1
Project
runBy
1..N
1
Department