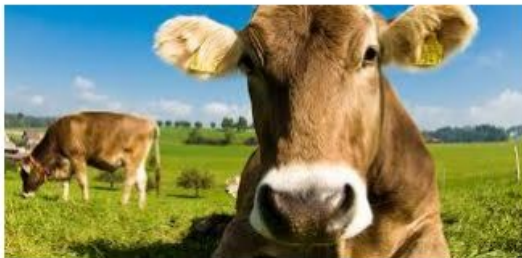


# Class Modeling in UML

# How many classes? and instances?



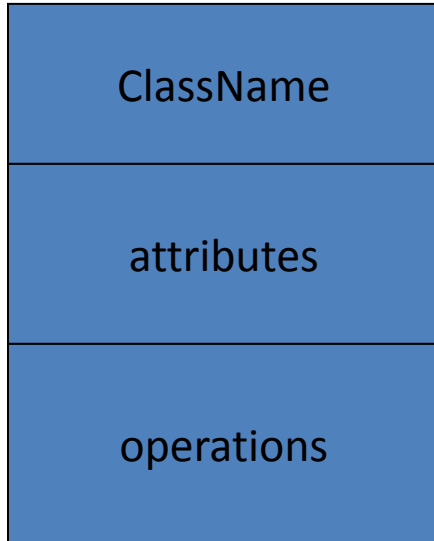
# What is modeling?

- Modeling consists of building an abstraction of reality.
- Abstractions are simplifications because:
  - They ignore irrelevant details and
  - They only represent the relevant details.
- What is *relevant* or *irrelevant* depends on the purpose of the model.
- UML (Unified Modeling Language)
  - An emerging standard for modeling object-oriented software.

# Class Diagram

- Describes the **static structure** of the system: Objects, Attributes, Associations.
- Used
  - during requirements analysis to model problem domain concepts
  - during system design to model subsystems and interfaces
  - during object design to model classes
- A class represent a concept
- A class encapsulates state (attributes) and behavior (operations)
- Each attribute has a type
- Each operation has a signature
- The class name is the only mandatory information

# Classes



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

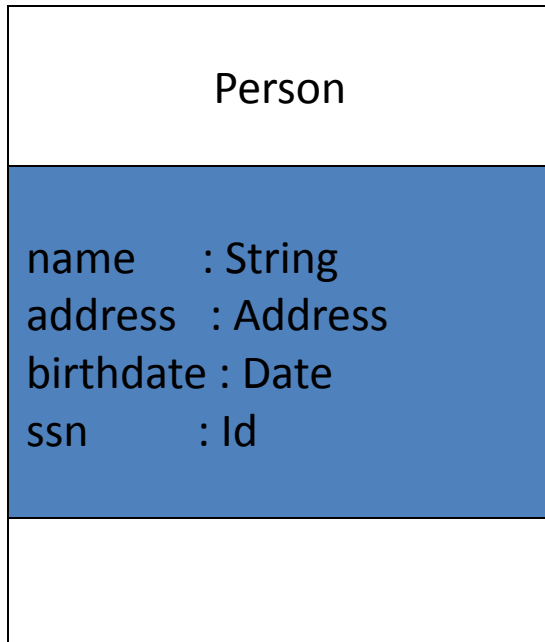
Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations in separate, designated compartments.

# Class Names

ClassName
attributes
operations

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.

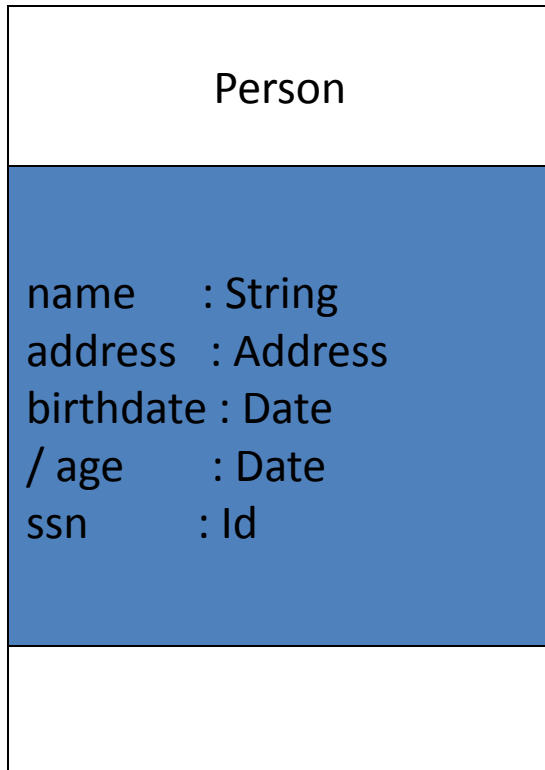
# Class Attributes (Cont'd)

Attributes are usually listed in the form:

attributeName : Type

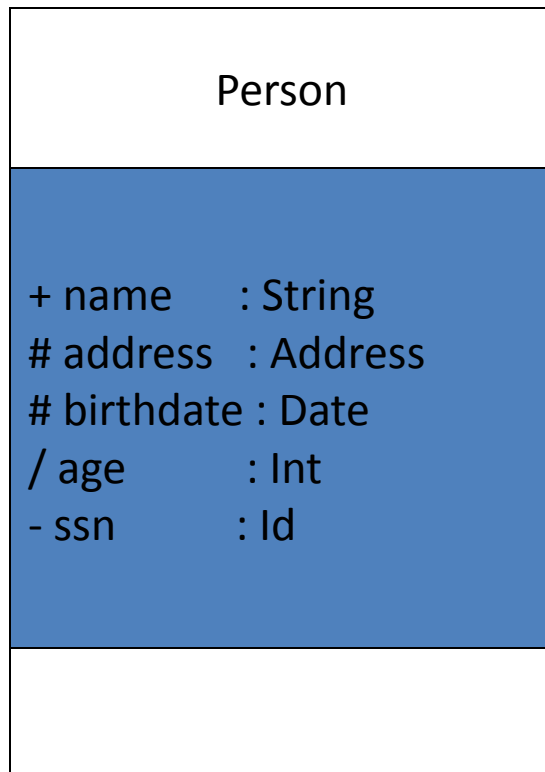
A *derived* attribute is one that can be computed from other attributes, but doesn't actually exist. For example, a Person's age can be computed from his birth date. A derived attribute is designated by a preceding '/' as in:

/ age : Date





# Class Attributes (Cont'd)



Attributes can be:

- + public

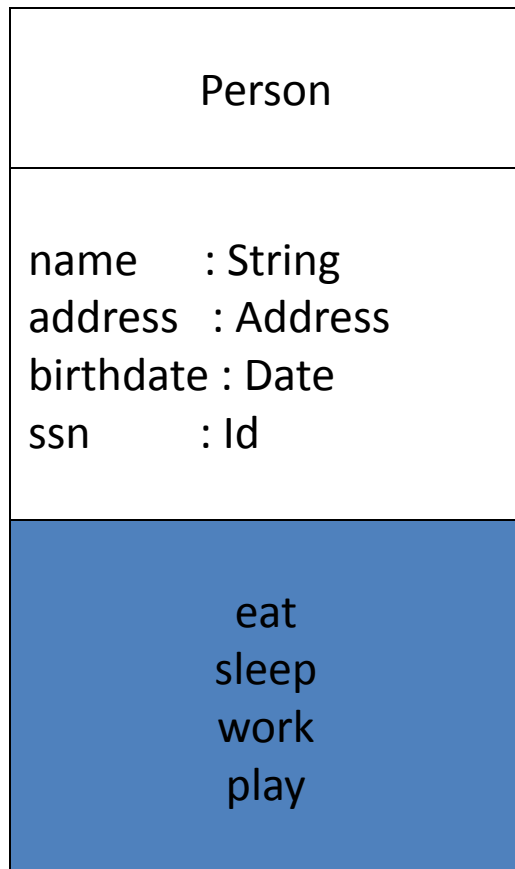
- # protected

- private

- / derived

underline static attributes

# Class Operations



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

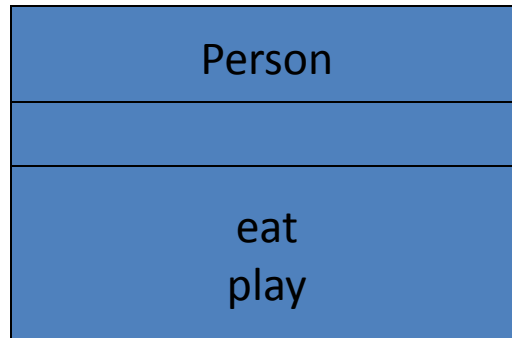
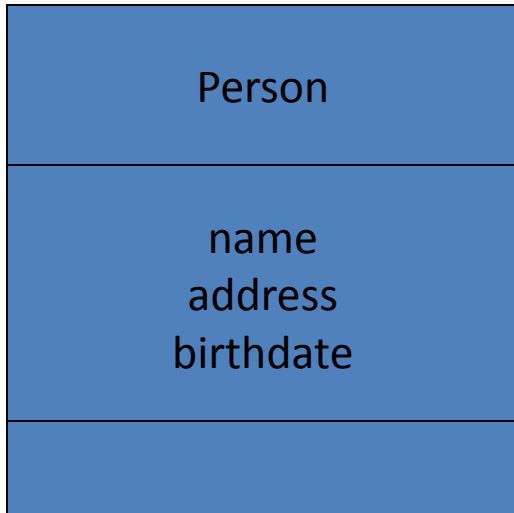
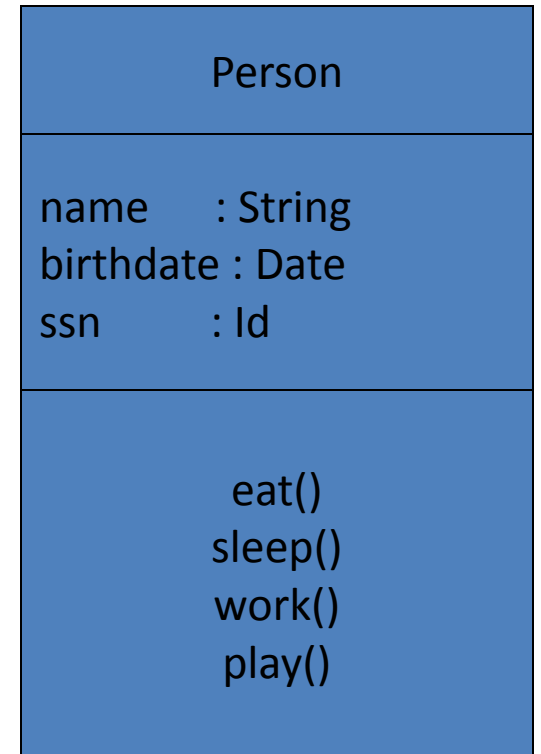
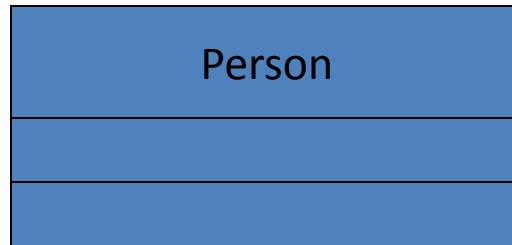
# Class Operations (Cont'd)

PhoneBook
<code>newEntry (n : Name, a : Address, p : PhoneNumber, d : Description)</code> <code>getPhone ( n : Name, a : Address) : PhoneNumber</code>

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

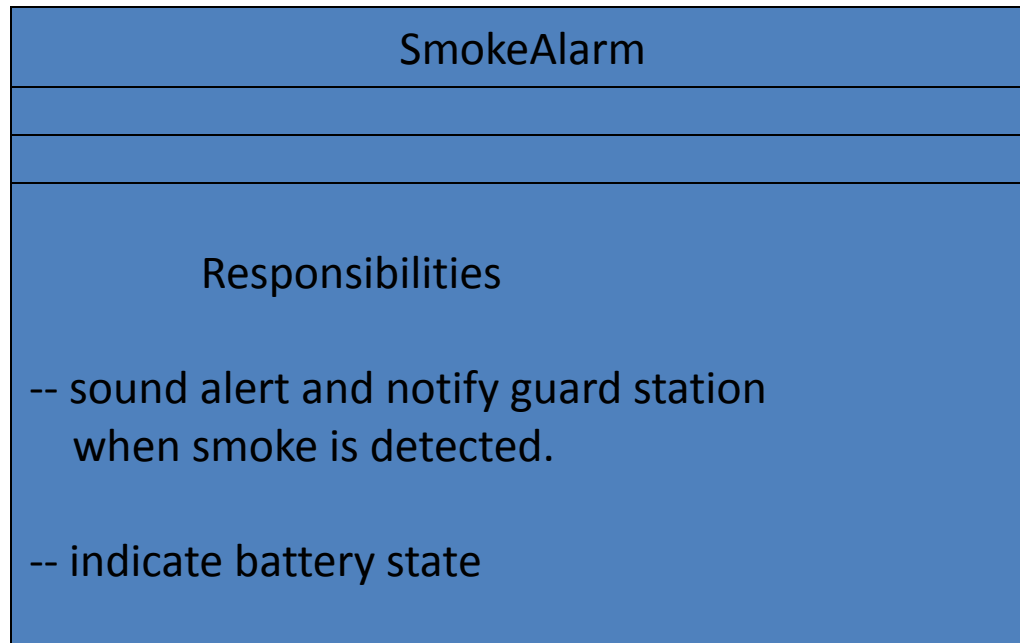
When drawing a class, you needn't show attributes and operation in every diagram.



# 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

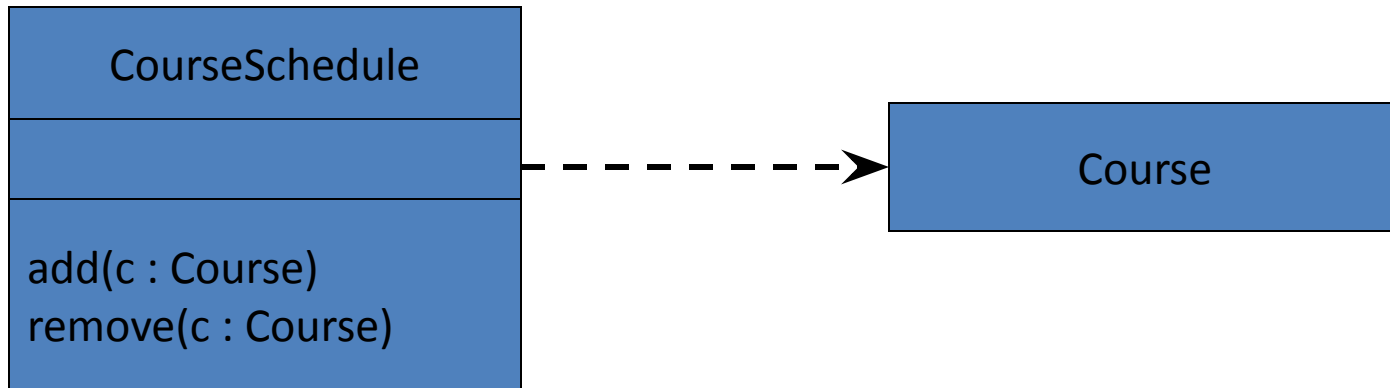
In UML, object interconnections (logical or physical), are modeled as relationships.

There are three kinds of relationships in UML:

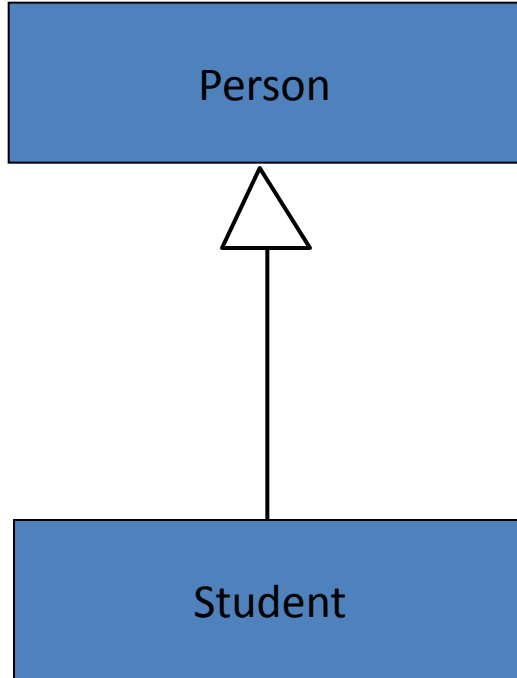
- dependencies
- generalizations
- associations

# 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

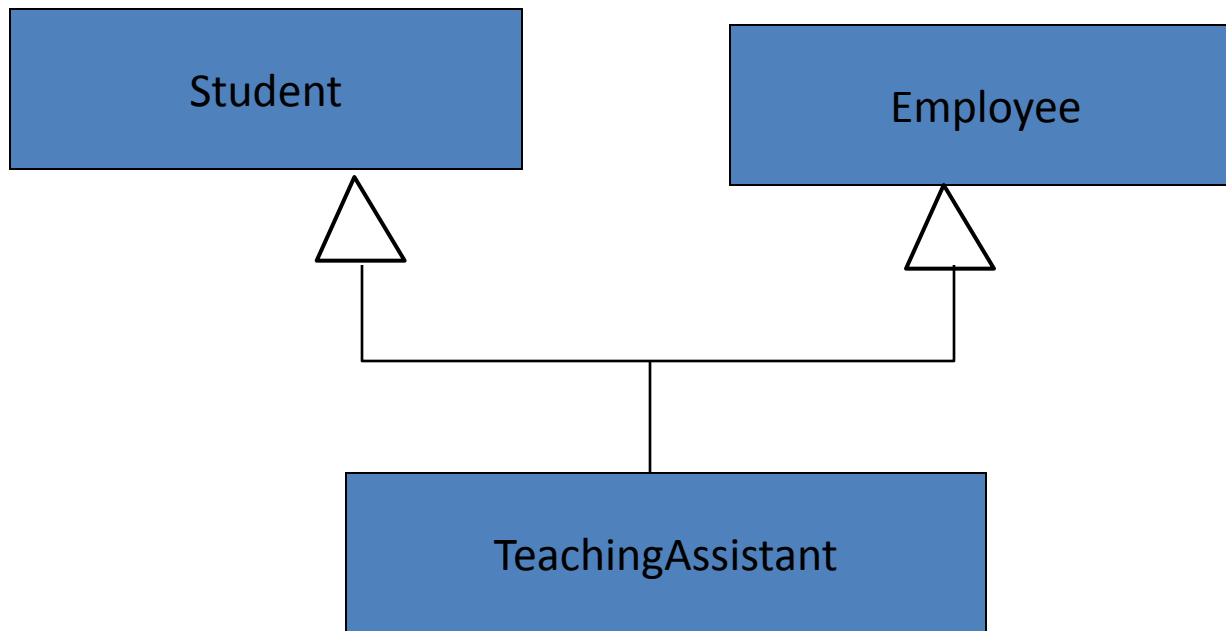


*A generalization* connects a subclass to its superclass. It denotes an inheritance of attributes and behavior from the superclass to the subclass and indicates a specialization in the subclass of the more general superclass.



# Generalization Relationships (Cont'd)

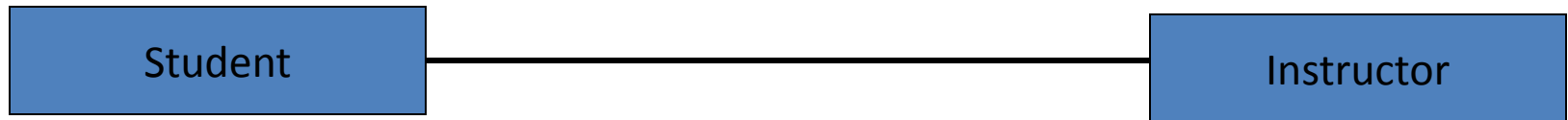
UML permits a class to inherit from multiple superclasses, although some programming languages (*e.g.*, Java) do not permit multiple inheritance.



# Association Relationships

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

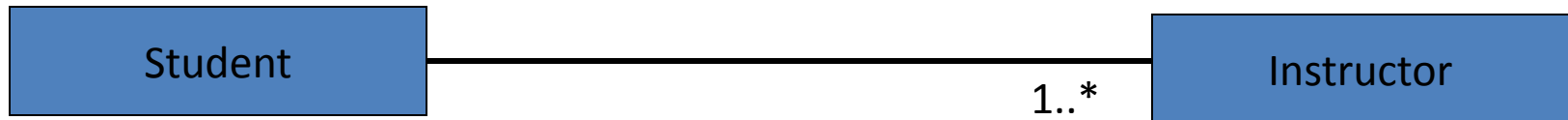
An *association* denotes that link.



# Association Relationships (Cont'd)

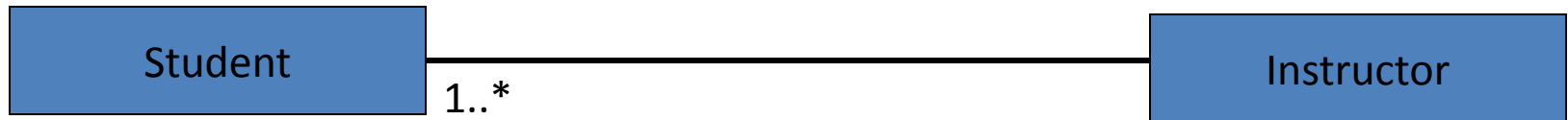
We can indicate the *multiplicity* of an association by adding *multiplicity adornments* to the line denoting the association.

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



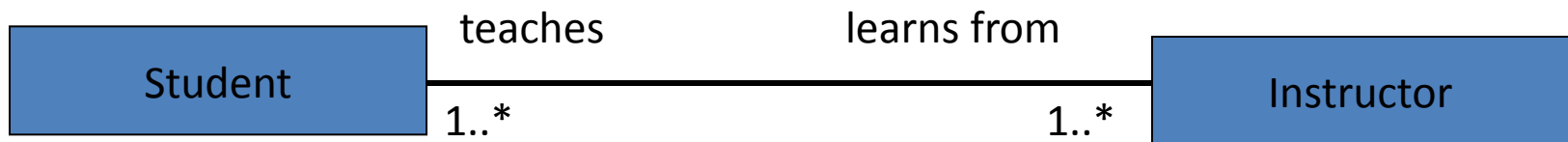
# Association Relationships (Cont'd)

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



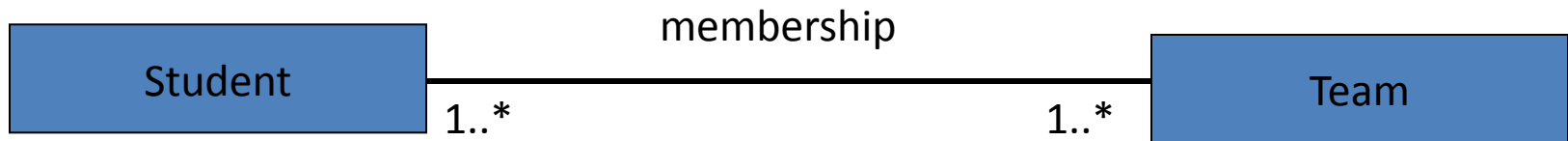
# Association Relationships (Cont'd)

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



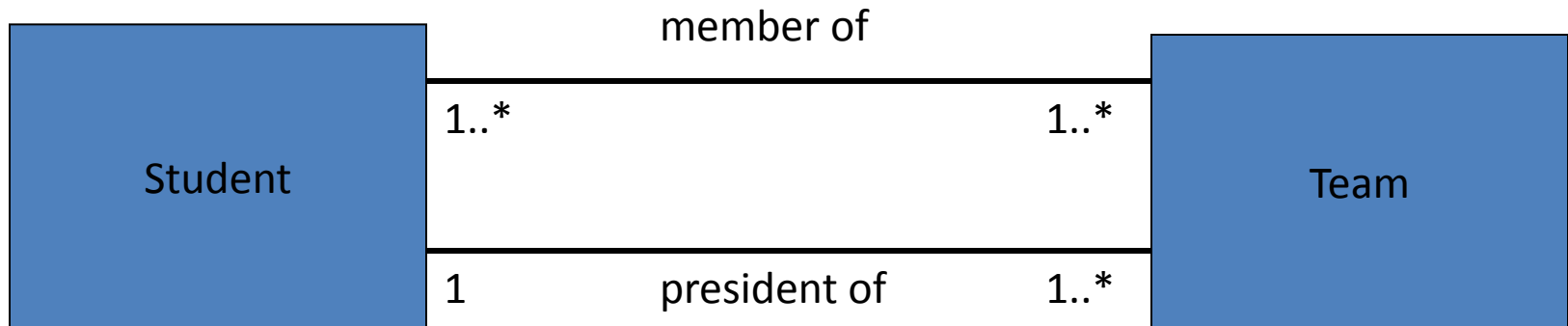
# Association Relationships (Cont'd)

We can also name the association.



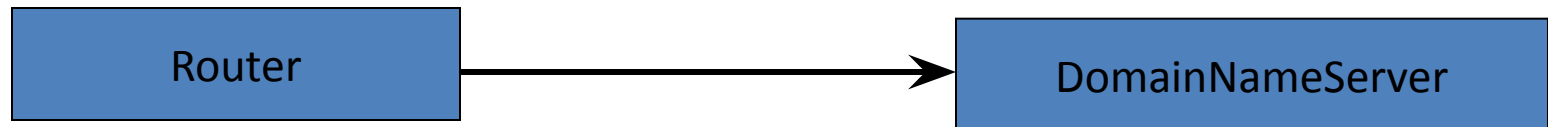
# Association Relationships (Cont'd)

We can specify dual associations.



# Association Relationships (Cont'd)

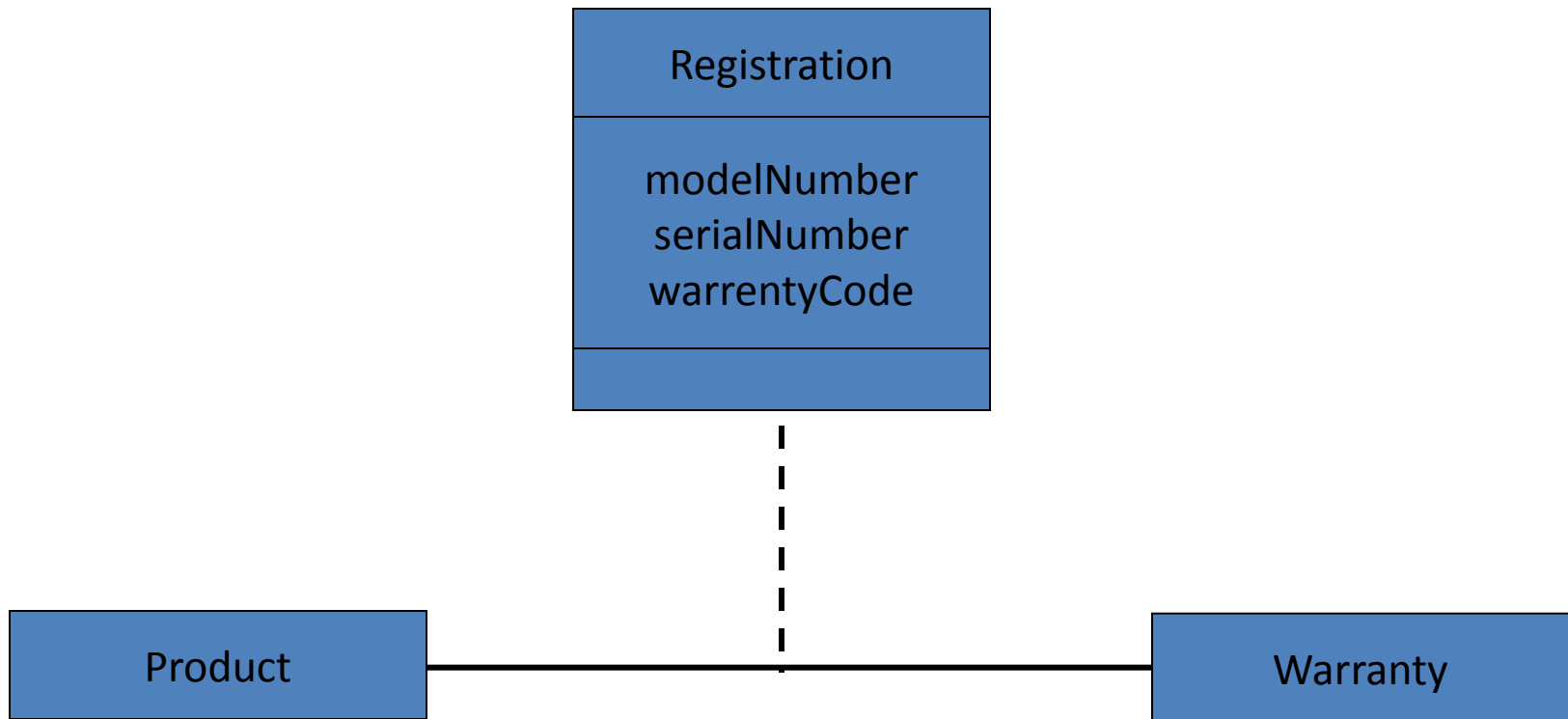
We can constrain the association relationship by defining the *navigability* of the association. Here, a *Router* object requests services from a *DNS* object by sending messages to (invoking the operations of) the server. The direction of the association indicates that the server has no knowledge of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





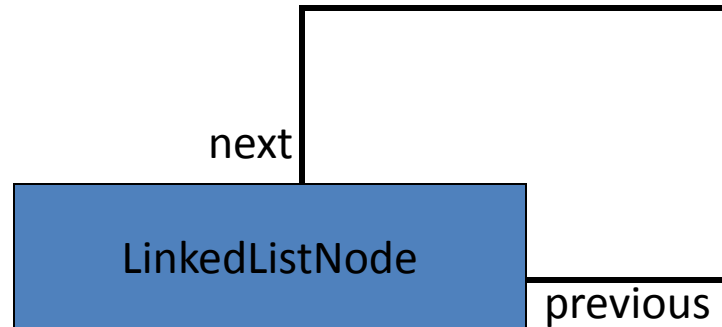
# Association Relationships (Cont'd)

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



# Association Relationships (Cont'd)

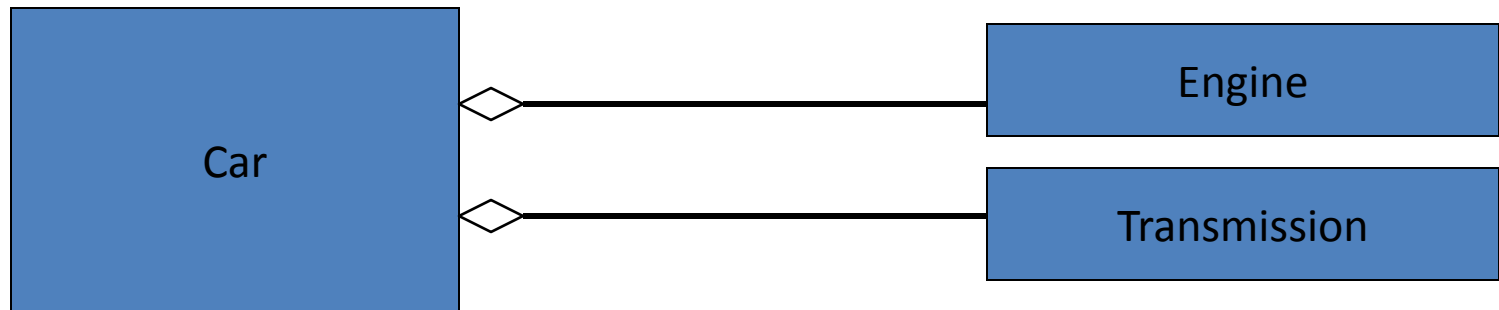
A class can have a *self association*.



# Association Relationships (Cont'd)

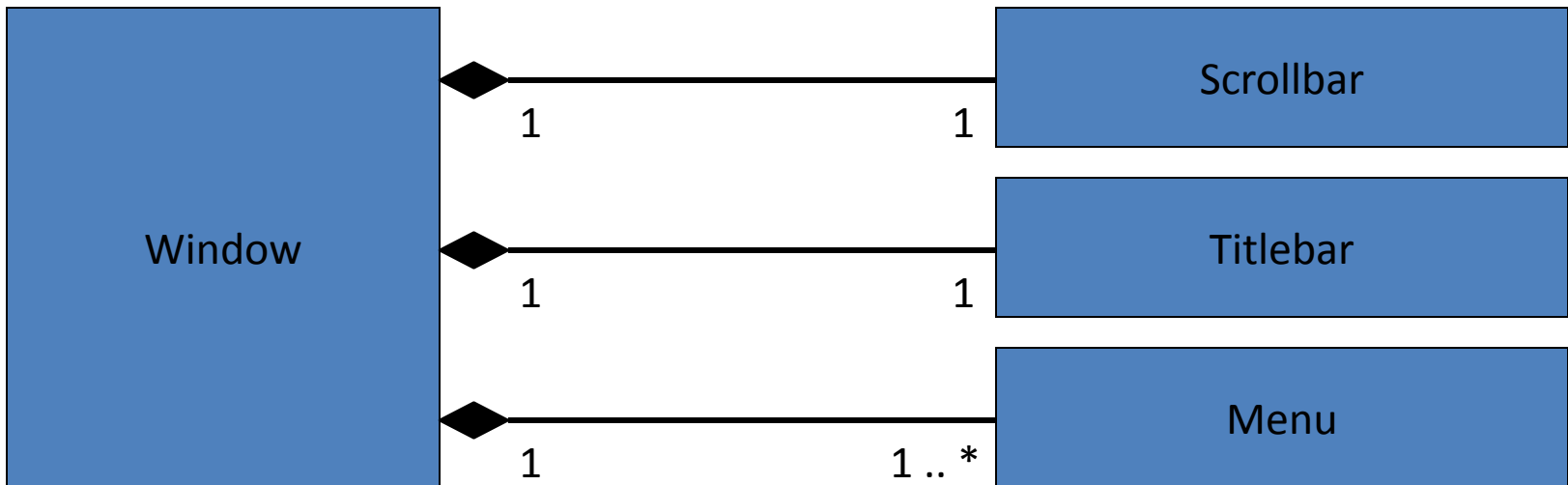
We can model objects that contain other objects by way of special associations called *aggregations* and *compositions*.

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.

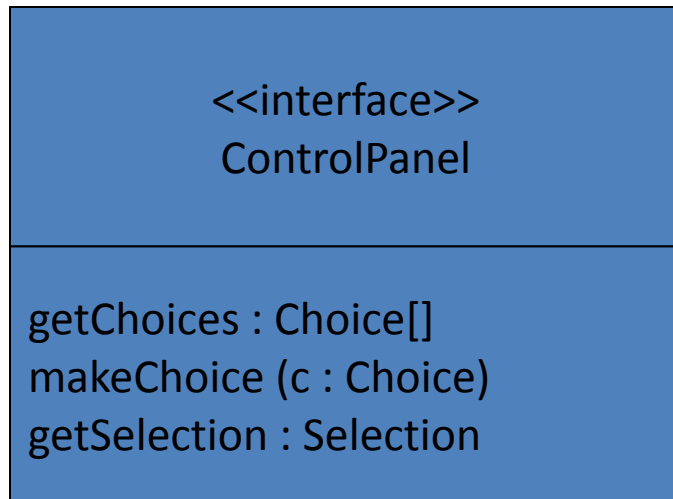


# Interfaces



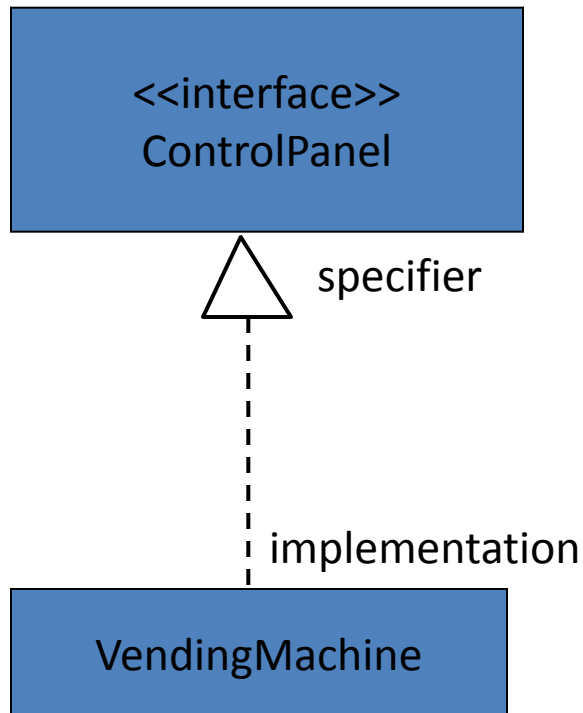
An *interface* is a named set of operations that specifies the behavior of objects without showing their inner structure. It can be rendered in the model by a one- or two-compartment rectangle, with the *stereotype* <<interface>> above the interface name.

# Interface Services



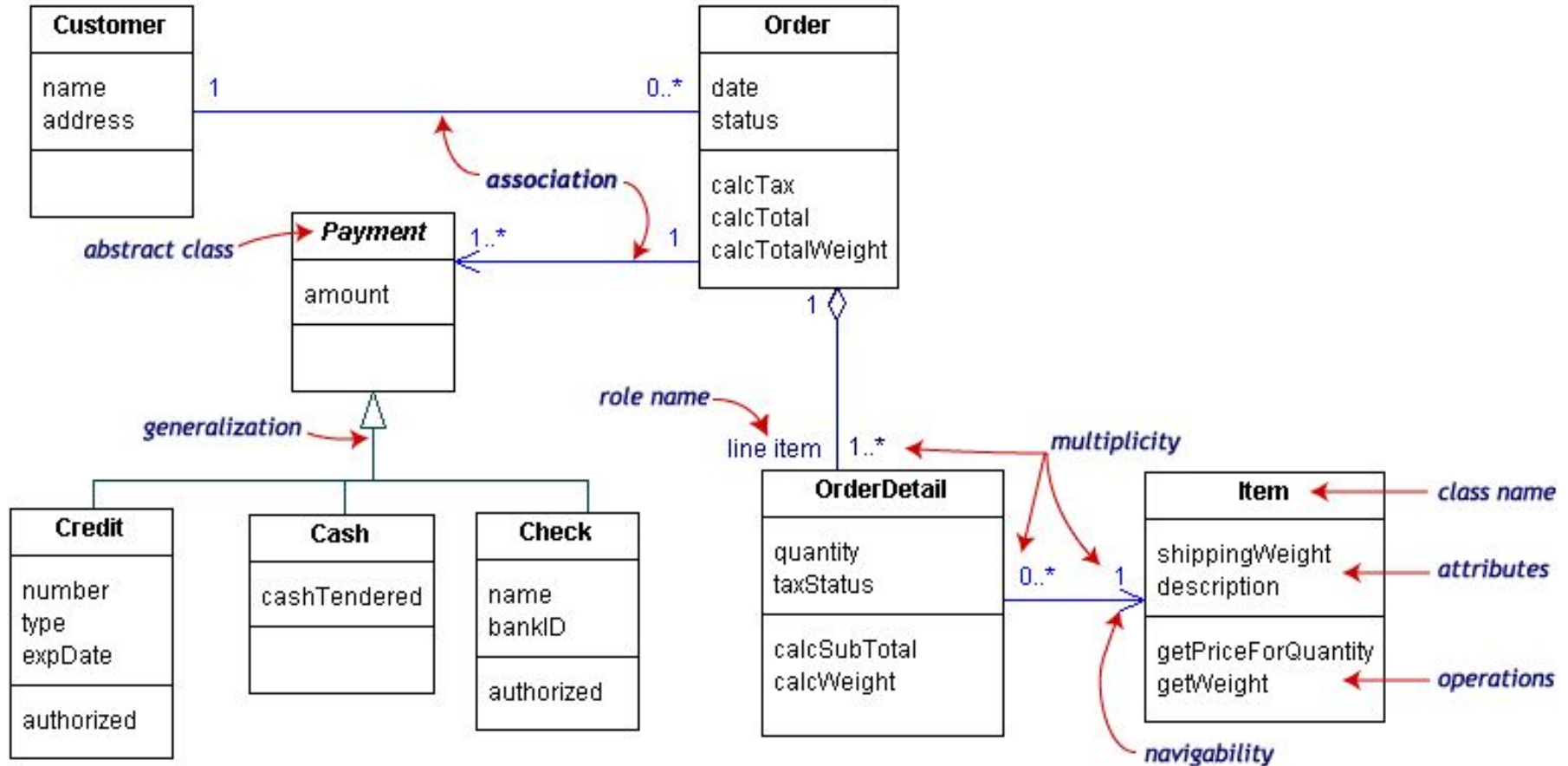
Interfaces do not get instantiated. They have no attributes or state. Rather, they specify the services offered by a related class.

# Interface Realization Relationship



A *realization* relationship connects a class with an interface that supplies its behavioral specification. It is rendered by a dashed line with a hollow triangle towards the specifier.

# UML Class Example





# Self-Check

1. *Shape, Rectangle, Cuboid, Point*
2. *Book, Page*
3. *Person, Employee, Manager, Engineer, Customer*
4. *Problem Statement: A stock exchange lists many companies.  
Each company is uniquely identified by a ticker symbol*
5. *Country, CapitalCity – Association?*

1. What is the relationship between Department and Employee?

```
class Department
```

```
{ ...
```

```
private:
```

```
string name;
```

```
Employee* receptionist; };
```

```
class Date{.. public: void  
printDate();};
```

```
class Student
```

```
{
```

```
private:
```

```
Date dob;
```

```
...
```

```
};
```

```
main()
```

```
{Student obj; ..}
```

[uml-diagrams.org](http://uml-diagrams.org)

Class diagram - reference