Class Diagrams

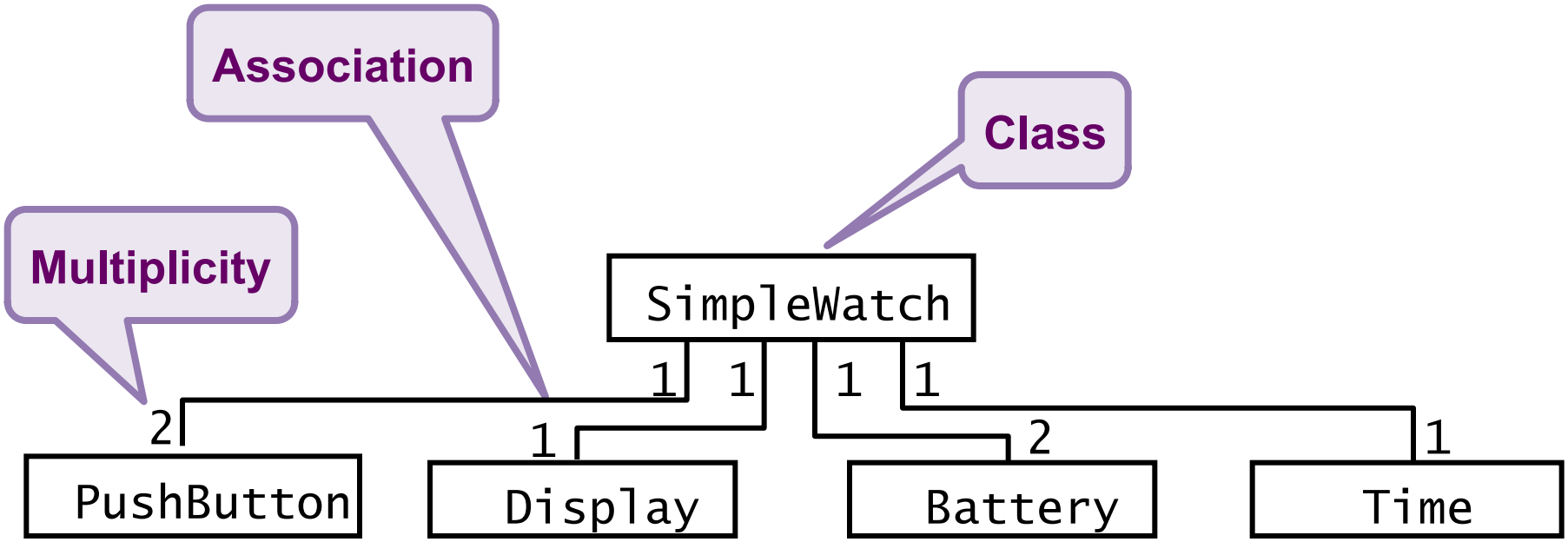
CSCI-4448 - Boese

University of Colorado **Boulder**

Slides from https://www.cs.drexel.edu/~spiros/teaching/CS575/slides/uml.ppt

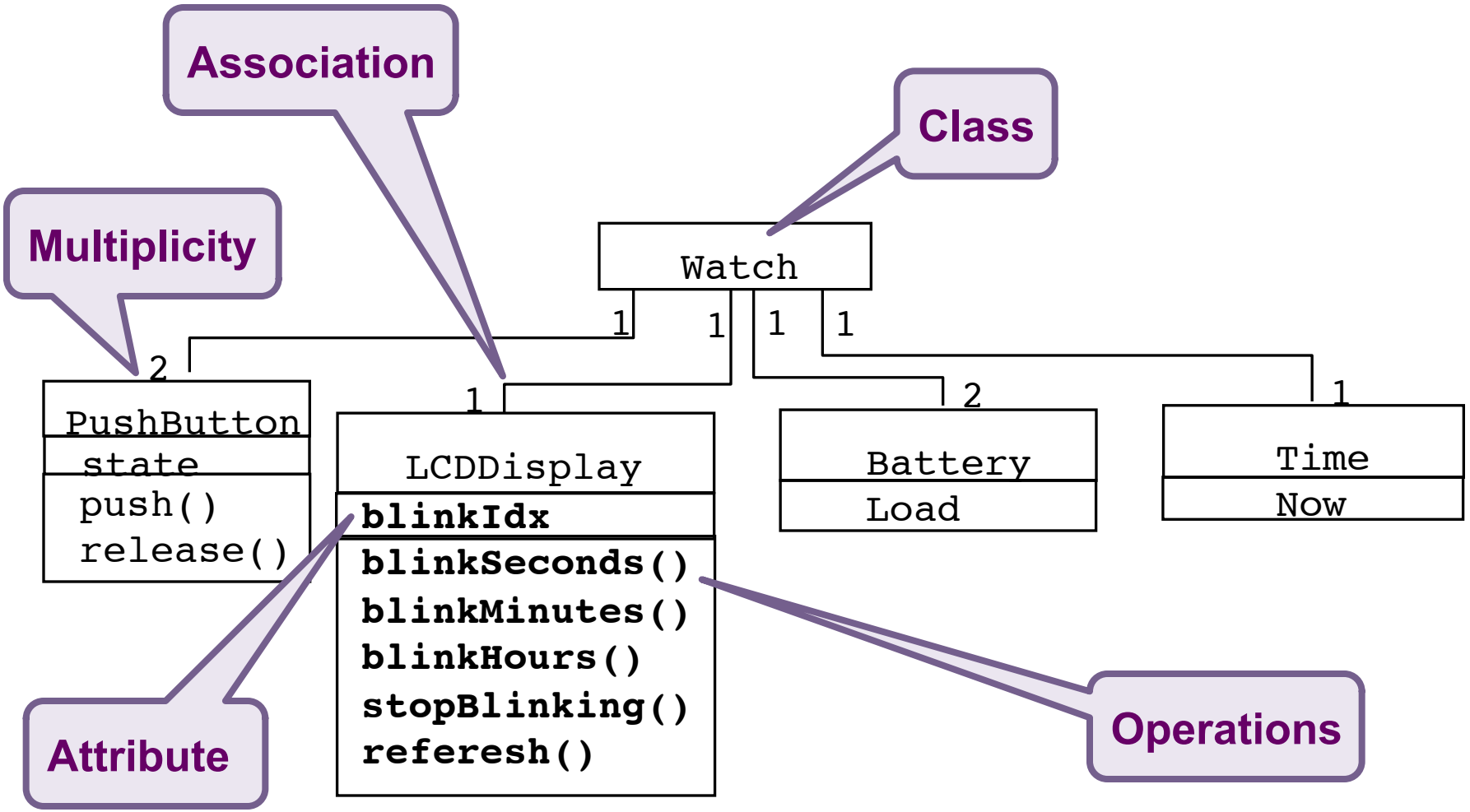
Overview

Class diagrams

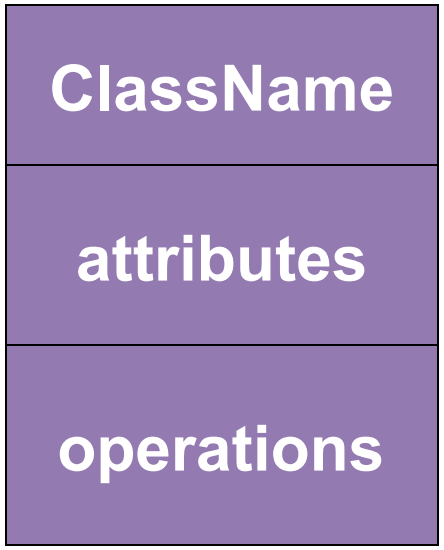


Class diagrams represent the **structure of the system**

Class diagrams



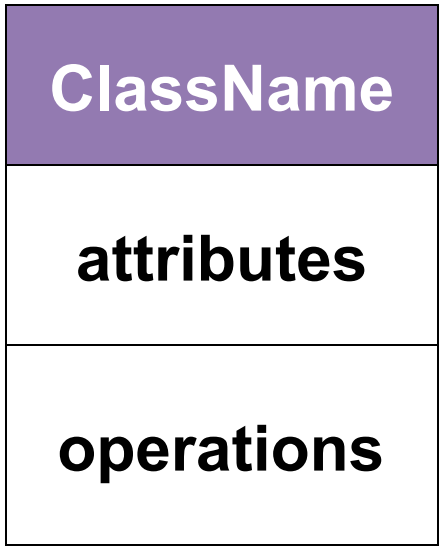
Classes



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

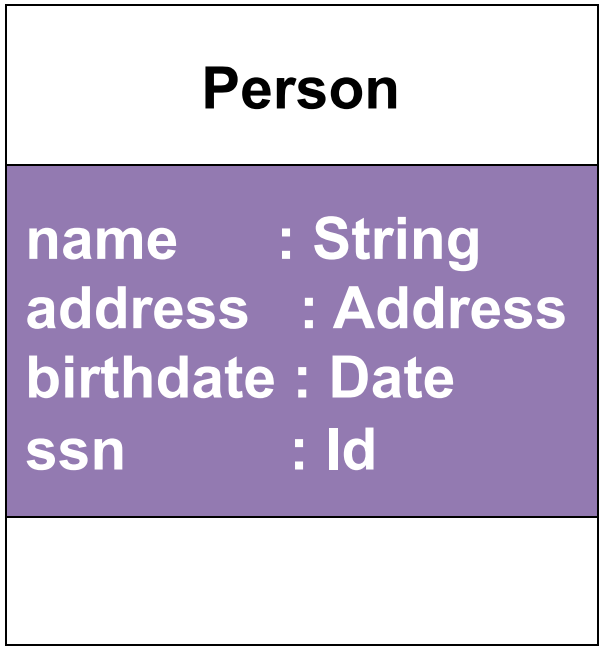
Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations in separate, designated compartments.

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 box.

Class Attributes



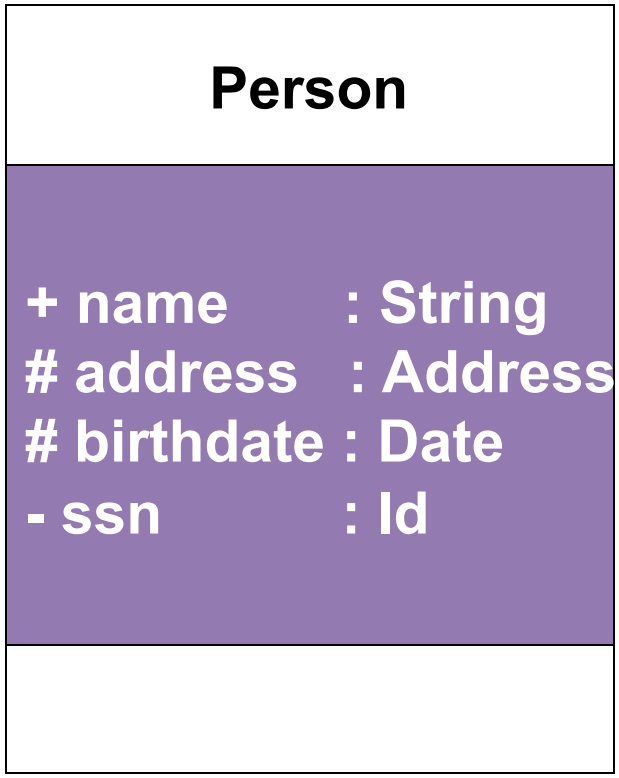
An ***attribute*** is a named property of a class that describes the object being modeled.

In the class diagram, attributes appear in the second box.

Attributes are listed in the form:

attributeName : Data Type

Class Attributes



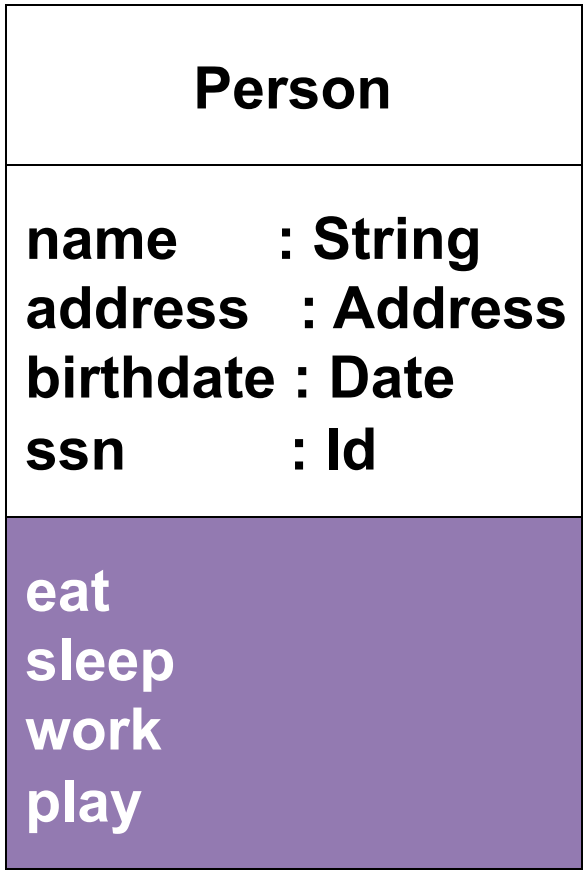
Attributes can be:

+ public

protected

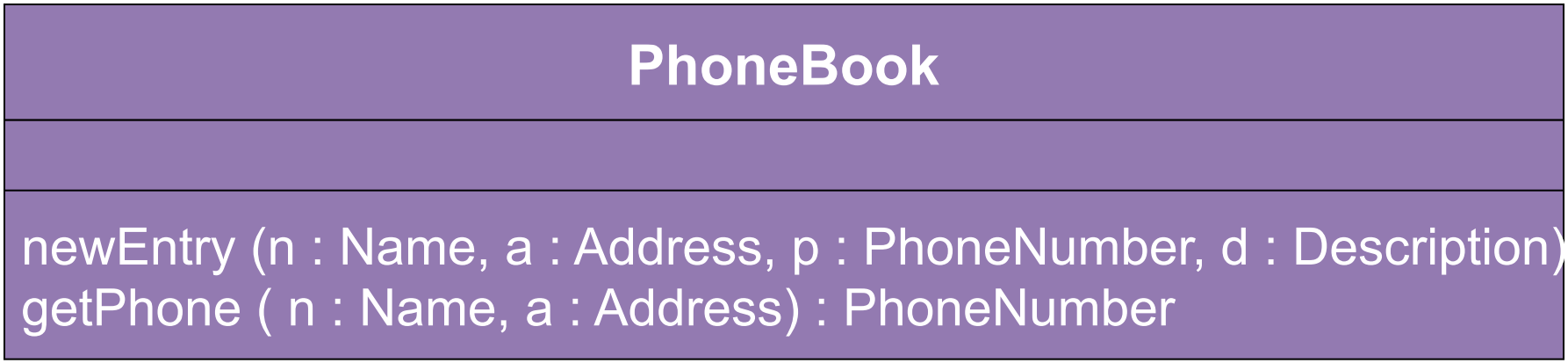
- private

Class Operations



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

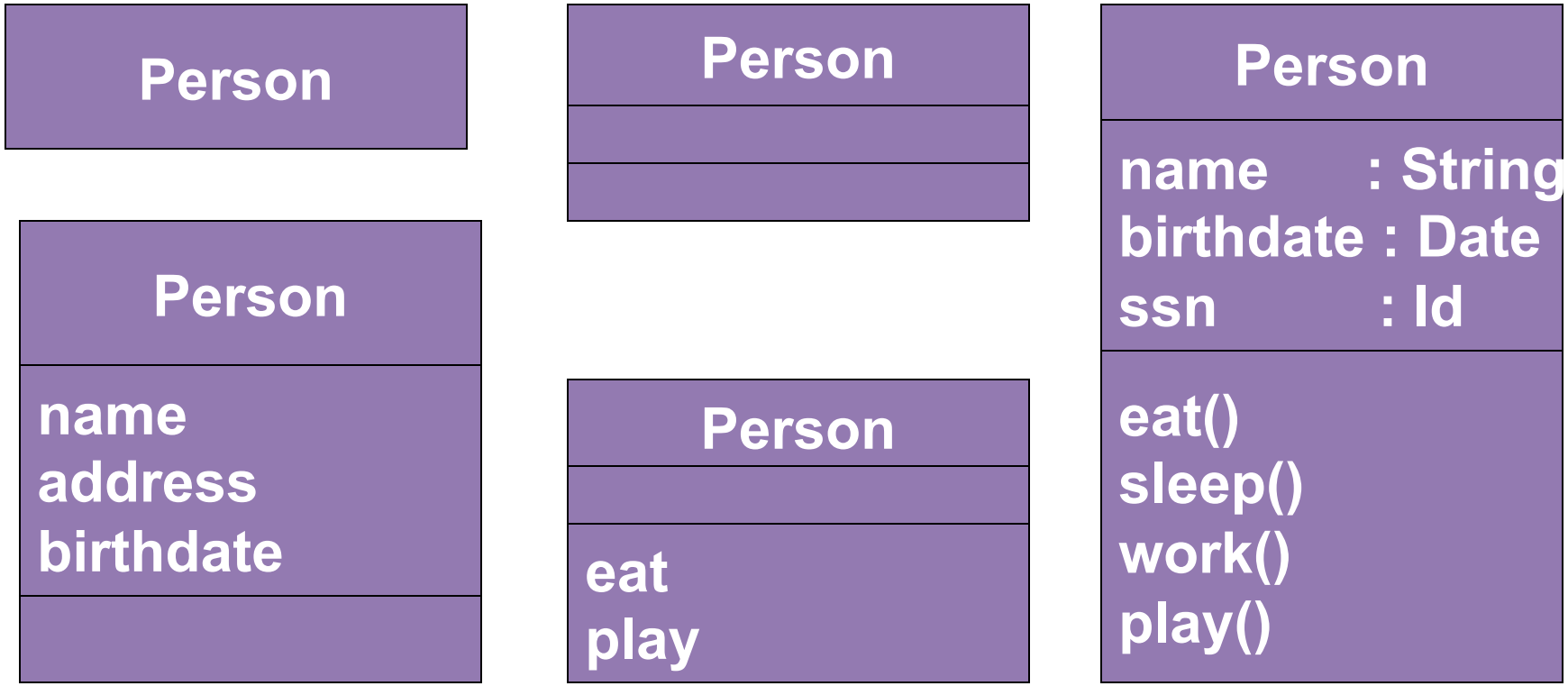
Class Operations



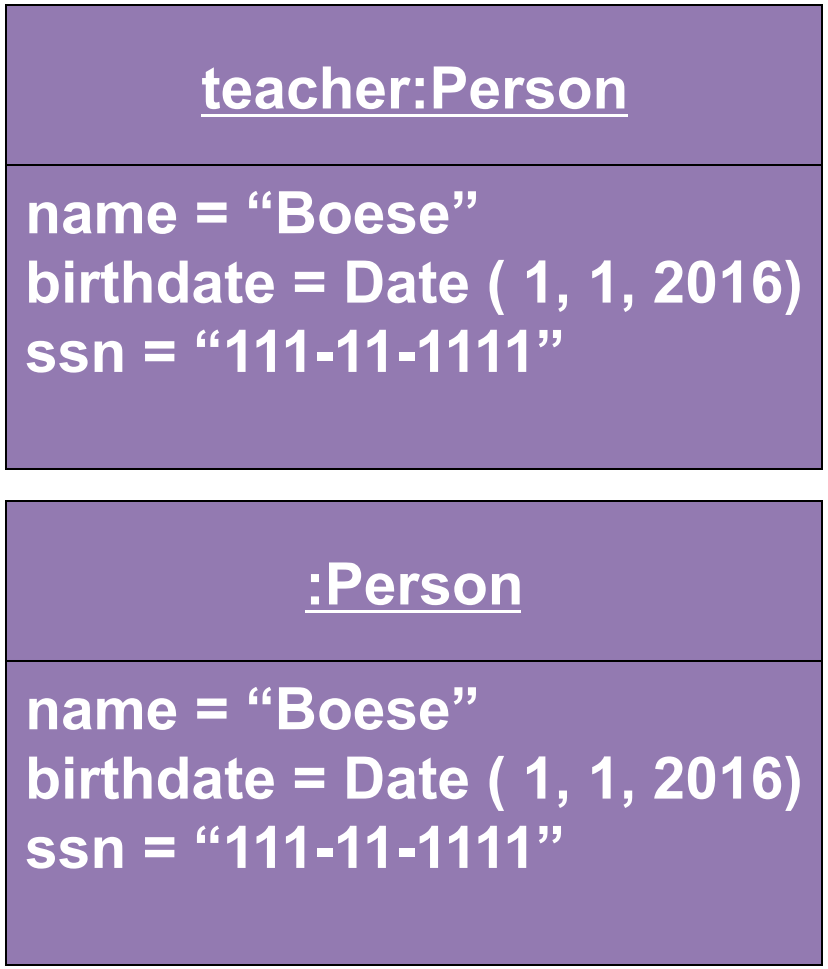
You can specify an operation by stating its **signature**: listing the name, type, and default value of all parameters, and, in the case of functions, a return type.

Depicting Classes

When drawing a class, you needn't show attributes and operation in *every* diagram. *(Class name is the only mandatory info)*



Instance of a class



An **instance** of a class shows the attributes with their values.

The name of an instance is underlined.

The name can contain only the class name as an anonymous instance, e.g.

:Person

Relationships

Relationships

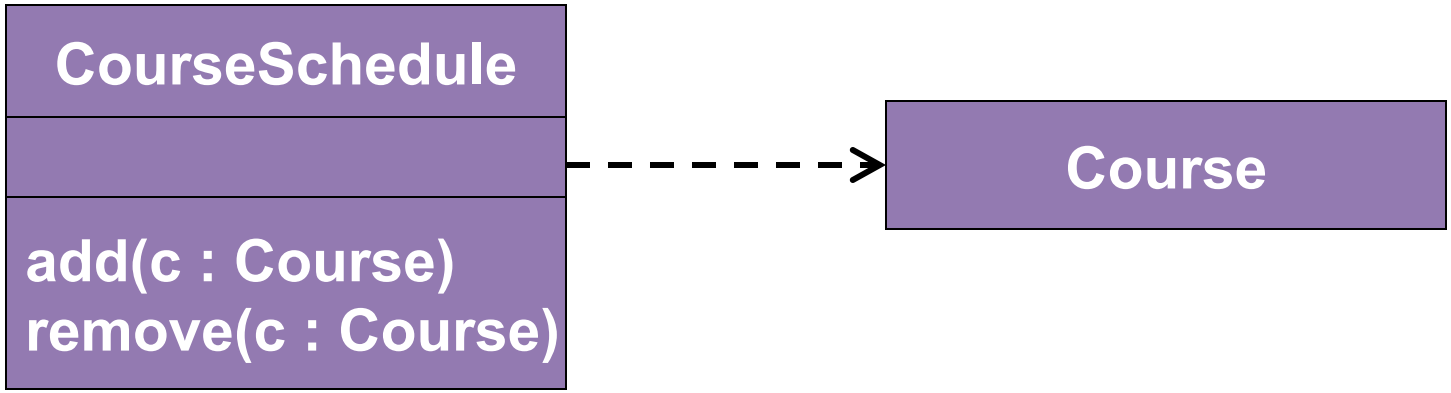
In UML, object interconnections (logical or physical), are modeled as relationships.

There are three kinds of relationships in UML:

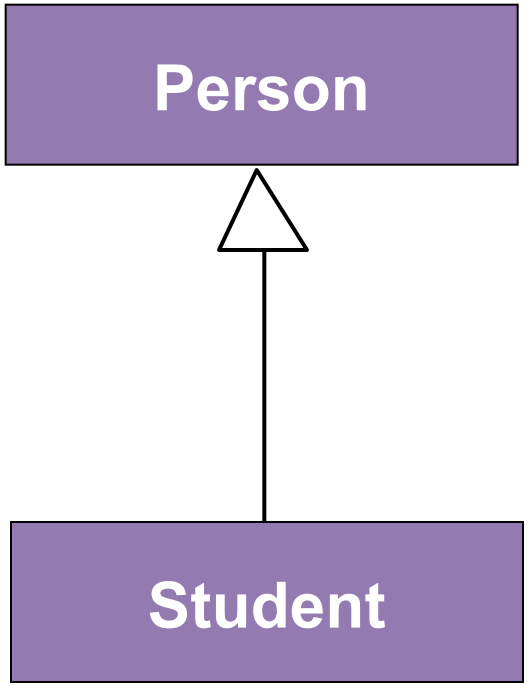
- Dependencies
- Generalizations
- Associations

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

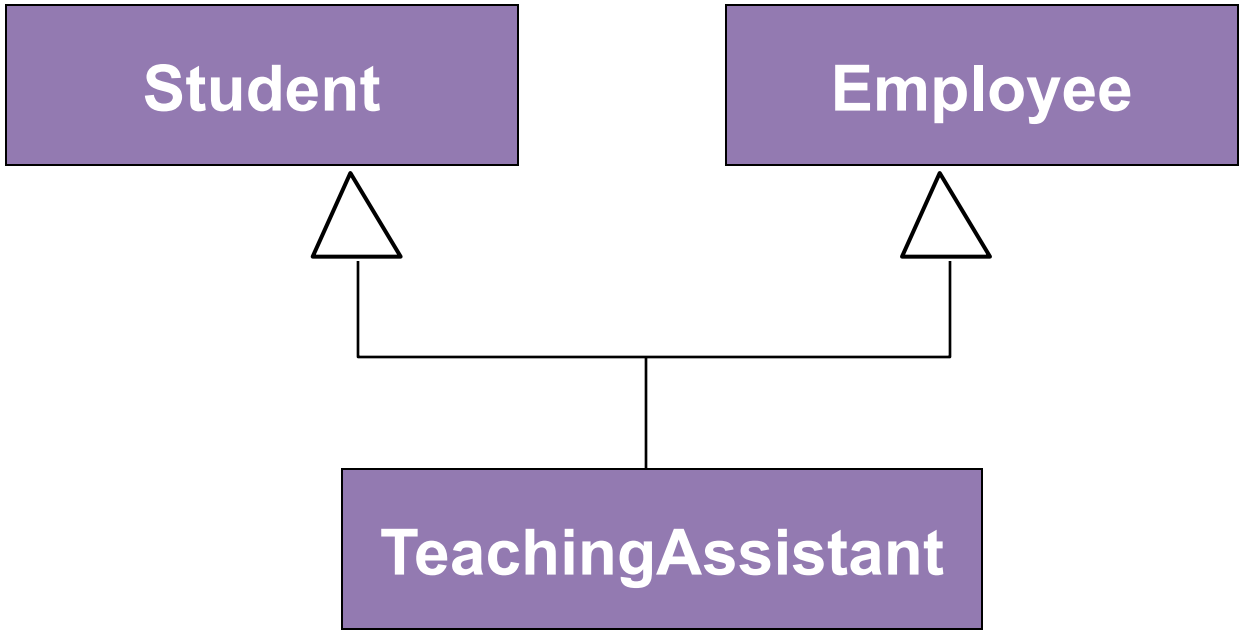


A ***generalization*** connects a subclass to its superclass.

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

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



Association Relationships

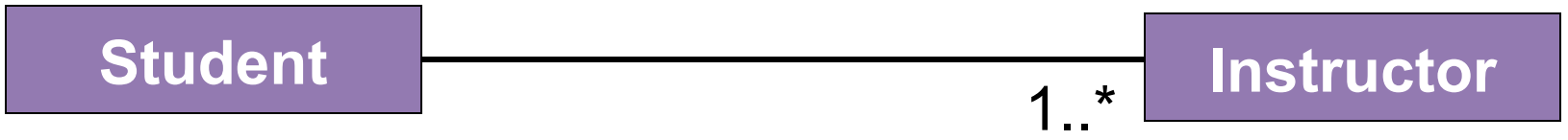
If two classes in a model need to communicate with each other, there must be link between them.

An ***association*** denotes that link.

Association Relationships

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

Table 3: Multiplicity values and their indicators

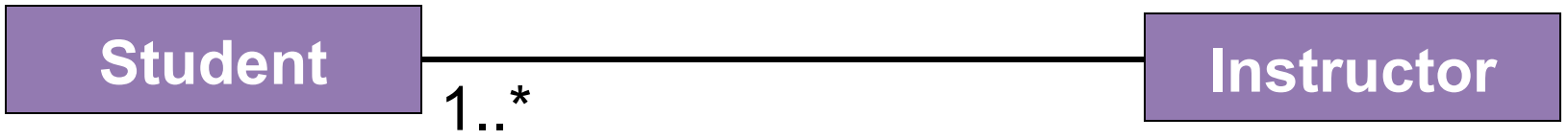
Potential Multiplicity Values

Indicator	Meaning
0..1	Zero or one
1	One only
0..*	Zero or more
*	Zero or more
1..*	One or more
3	Three only
0..5	Zero to Five
5..15	Five to Fifteen

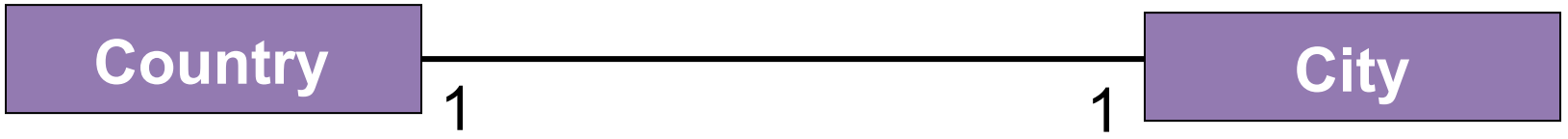
http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/

Association Relationships

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

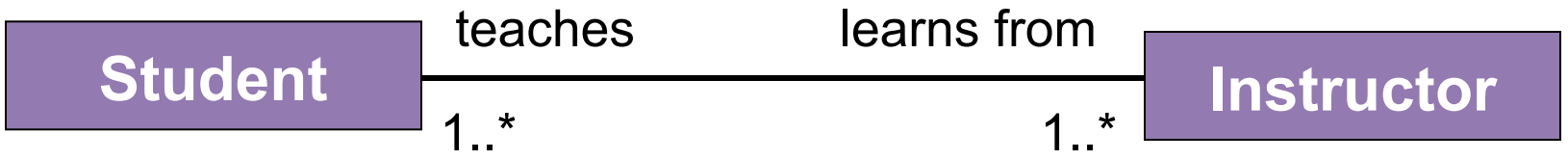


The example indicates that every *Country* has one and only one *City*, and every *City* has one and only one *Country.*



Association Relationships

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

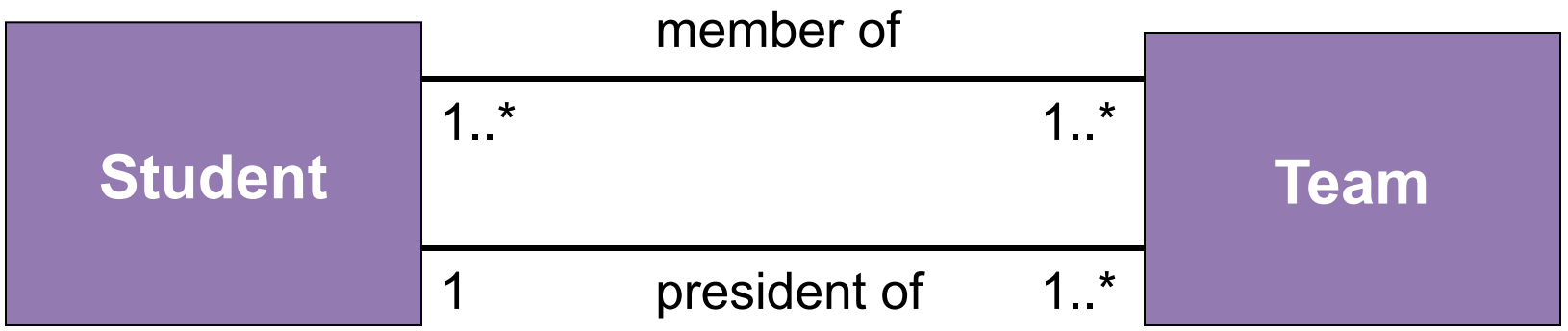


Association Relationships

We can also name the association.

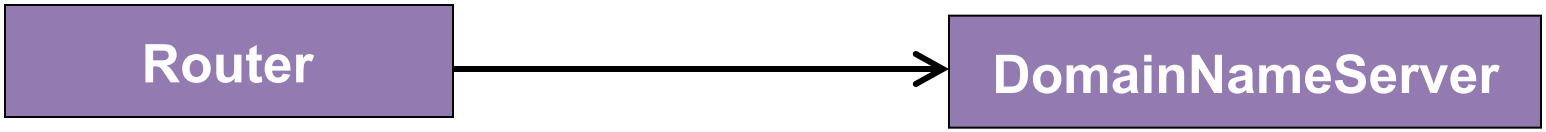
Association Relationships

We can specify dual associations.



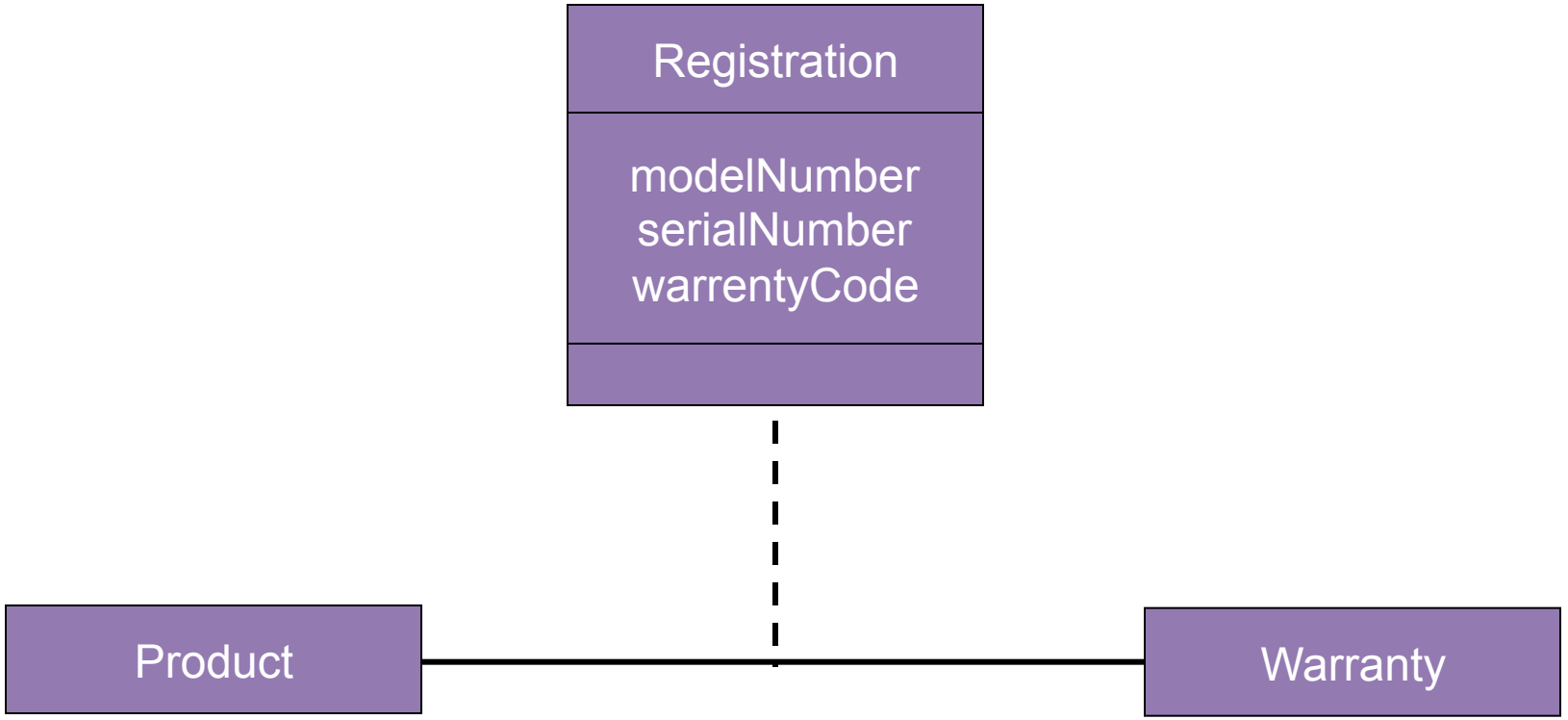
Association Relationships

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*.



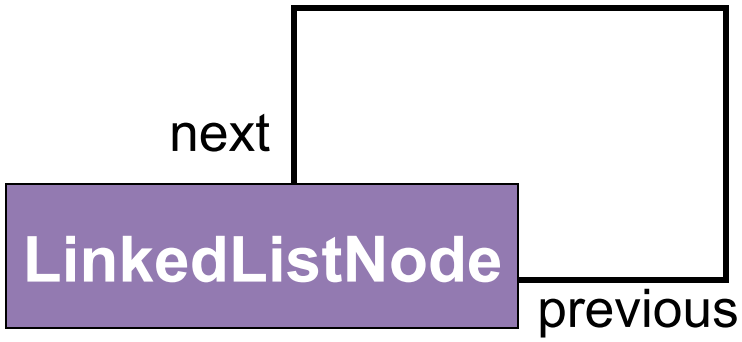
Association Relationships (Cont'd)

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



Association Relationships

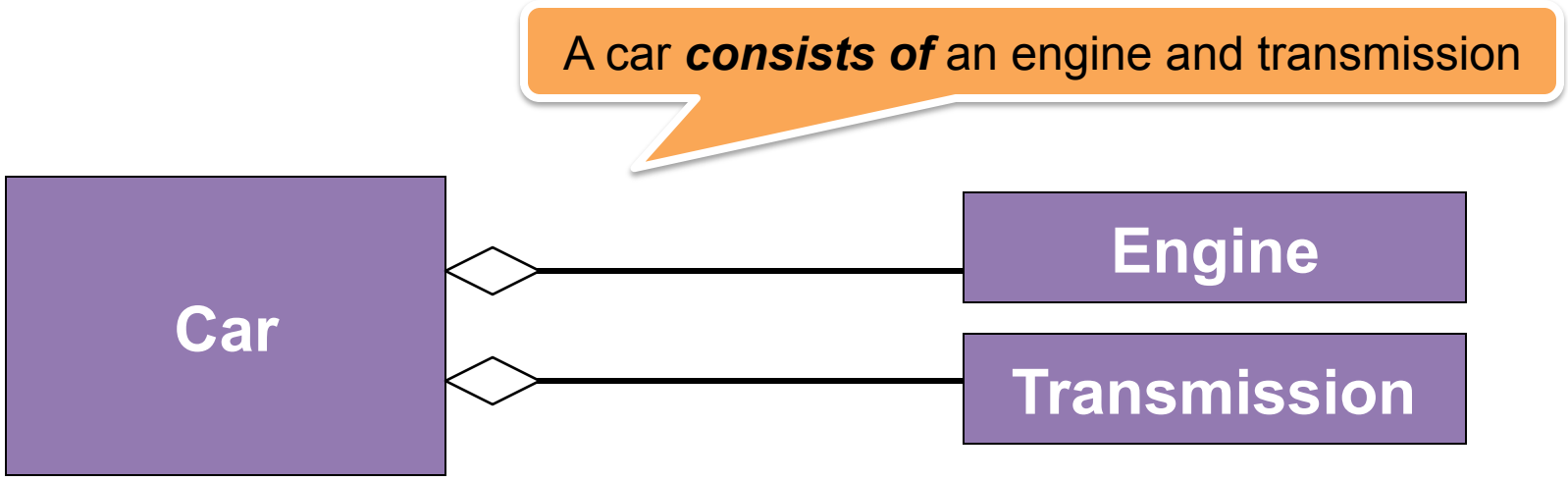
A class can have a ***self association***.



Association Relationships

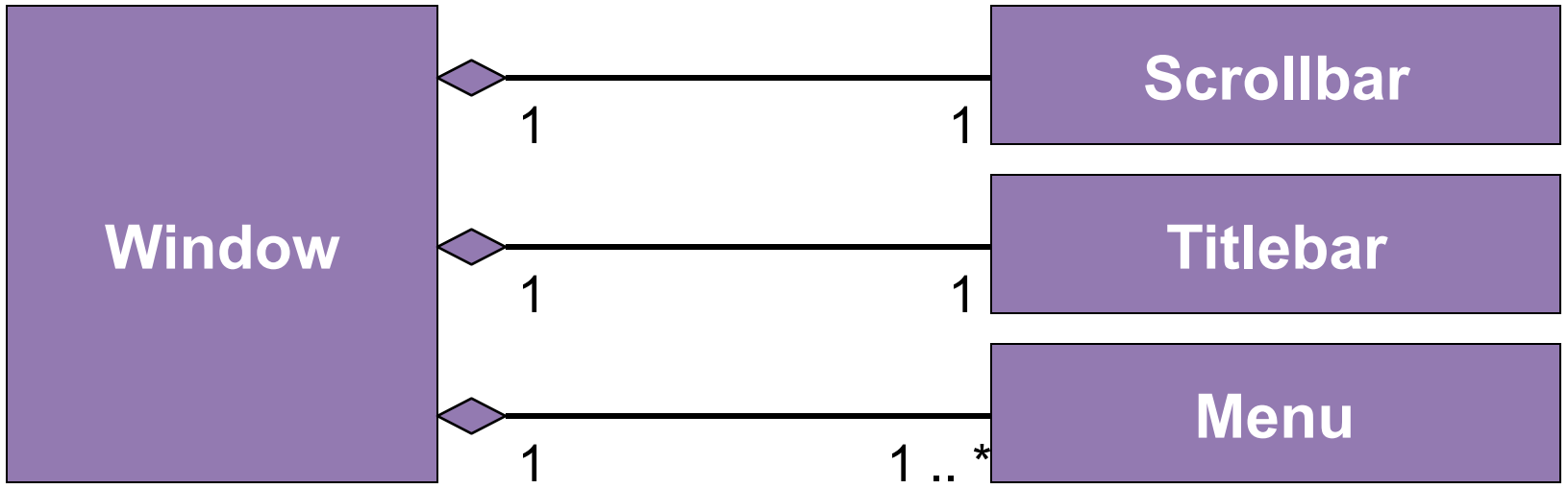
We can model objects that contain other objects by way of special associations called ***aggregations*** and ***compositions***.

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.



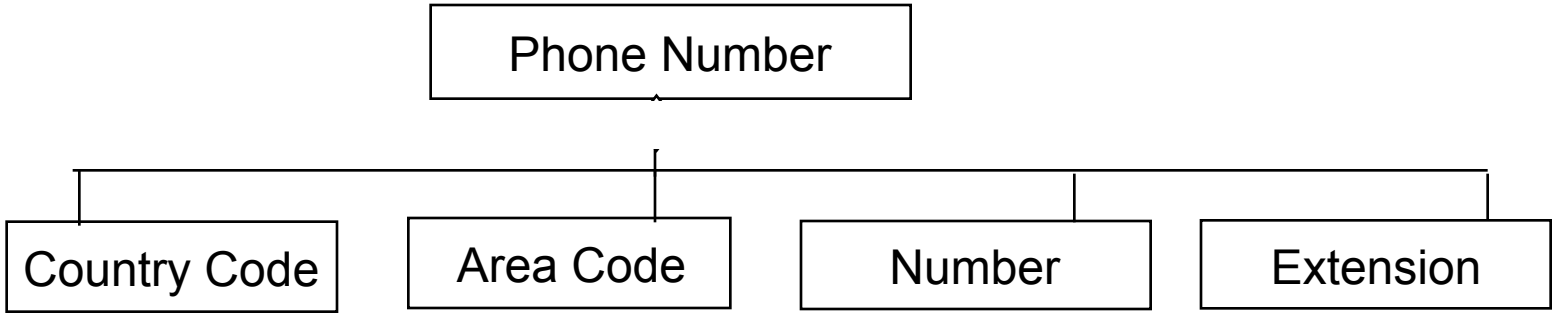
Association Relationships

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 on the association.

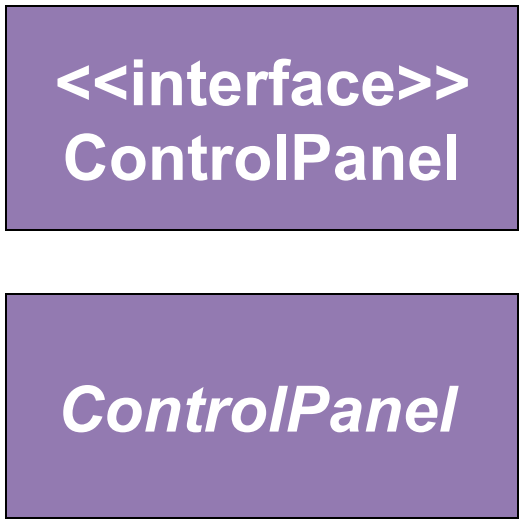


Composition or Aggregation?

- Which would you use: composition or aggregation?
  - Phone Number
    - Country code
    - Area code
    - Number
    - Extension



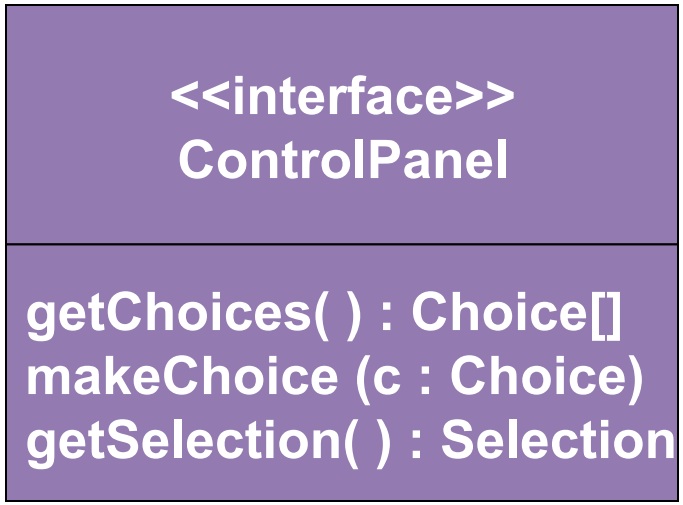
Interfaces



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 (name and optional methods), with the *stereotype* <<interface>> above the interface name (or italicized).

Interface Services

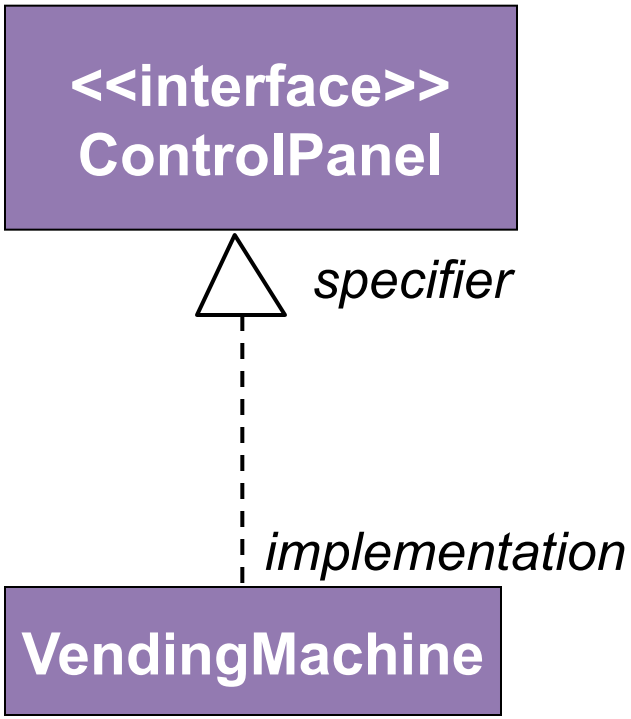


Interfaces do not get instantiated.

They have no attributes or state.

Rather, they specify the services offered by a related class.

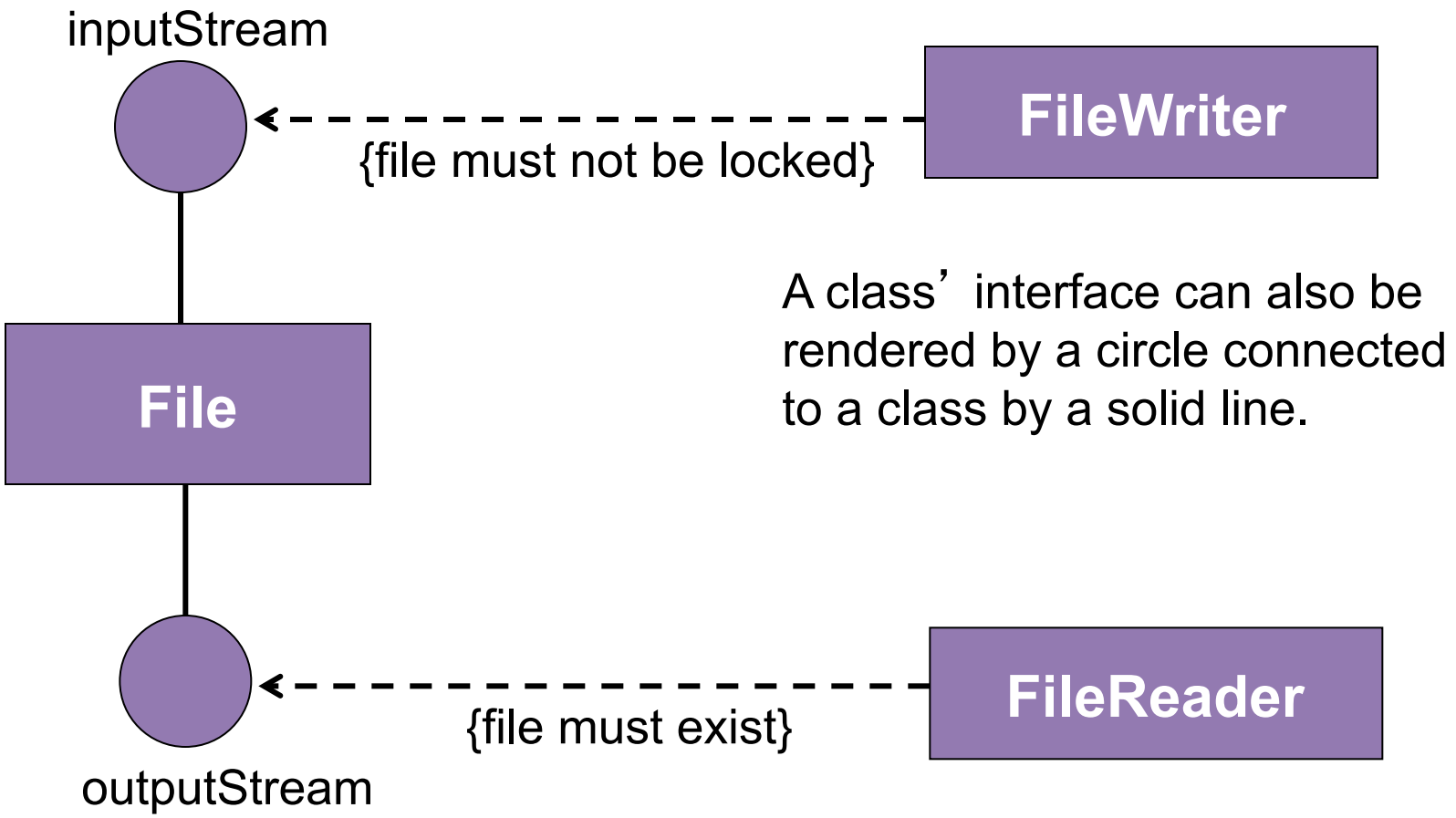
Interface Realization Relationship



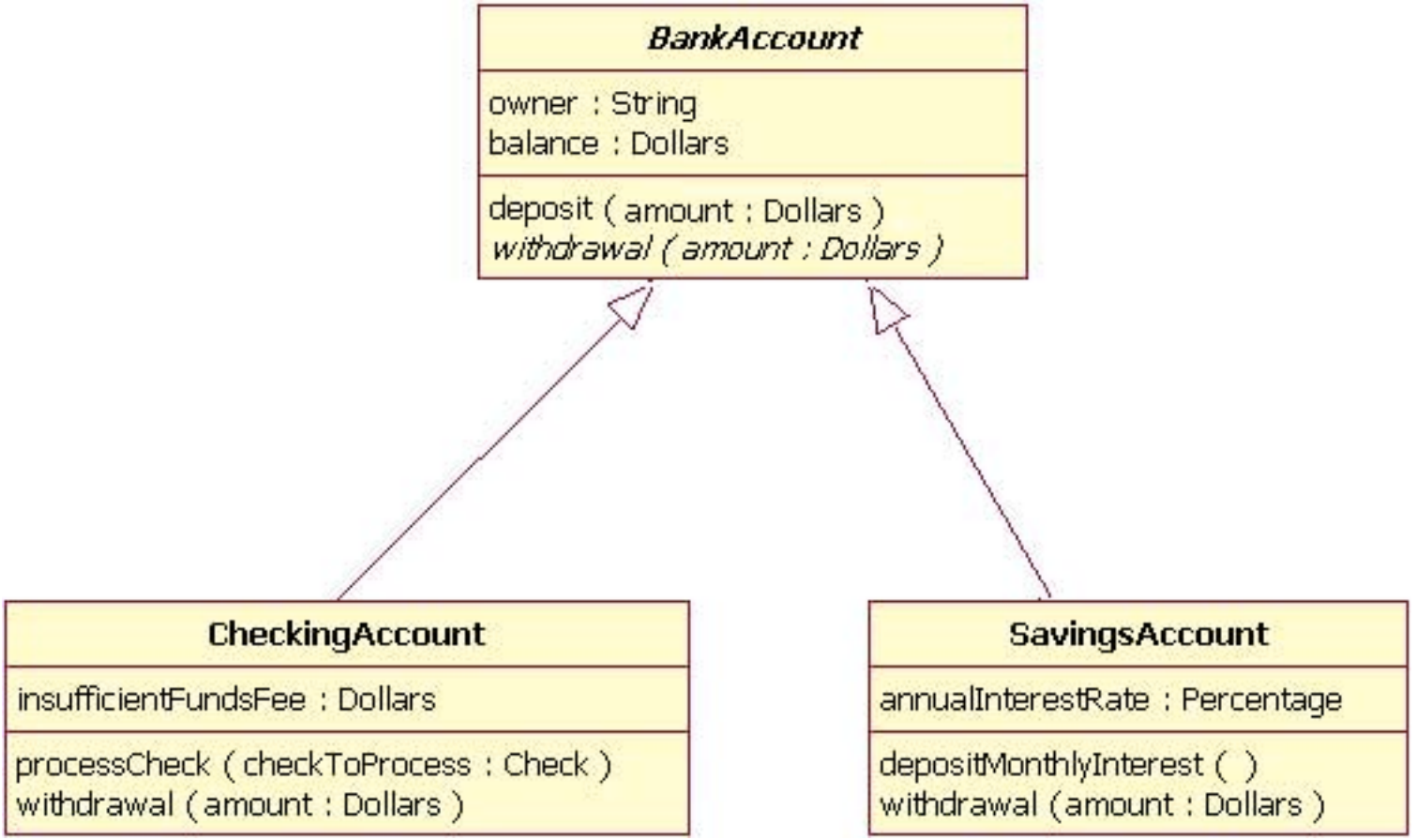
A ***realization*** relationship connects a class with an interface that supplies its behavioral specification.

It is rendered by a dashed line with a hollow triangle towards the specifier.

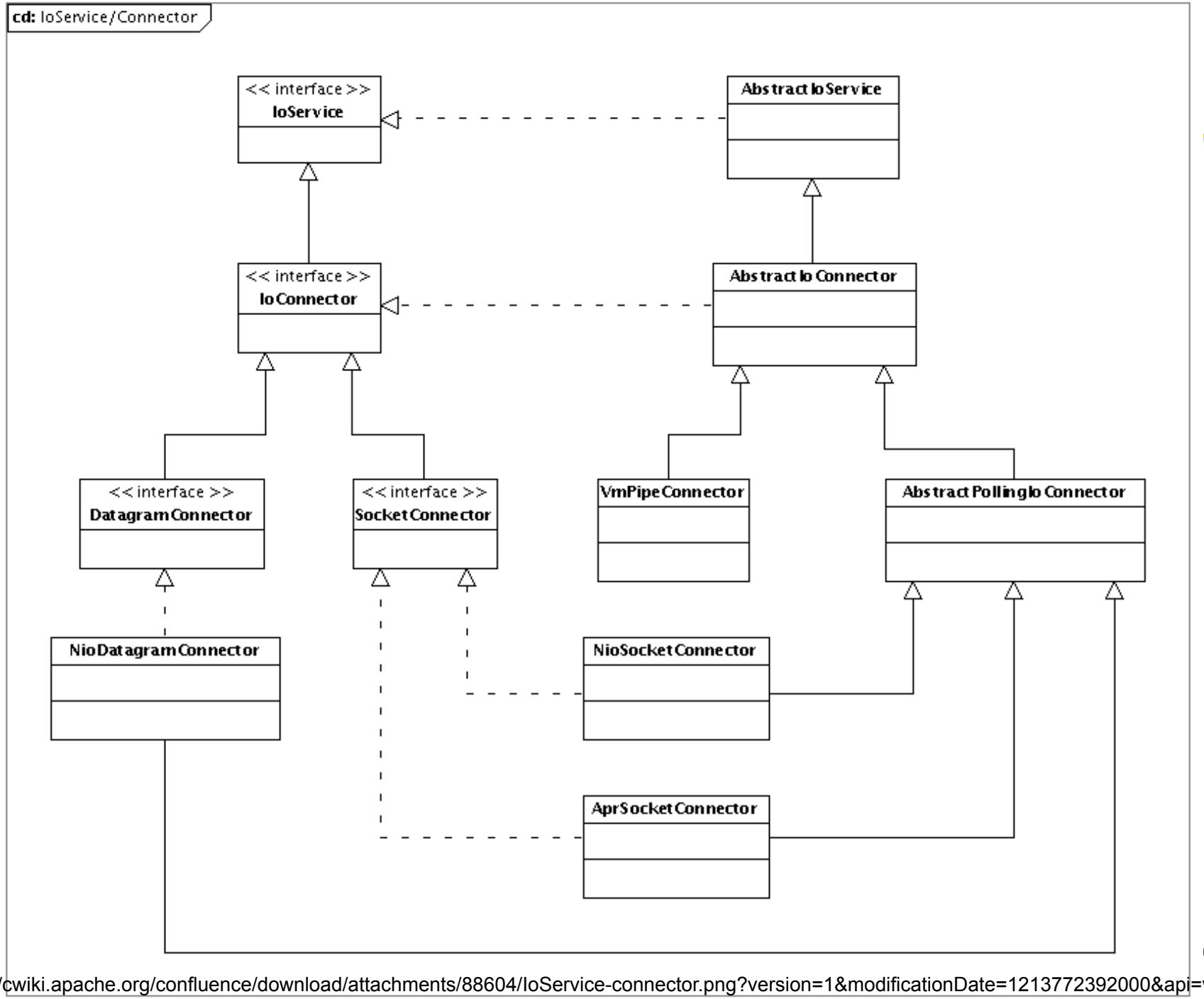
Interfaces


inputStream
FileWriter
{file must not be locked}
File
A class' interface can also be rendered by a circle connected to a class by a solid line.
FileReader
{file must exist}
outputStream

Examples

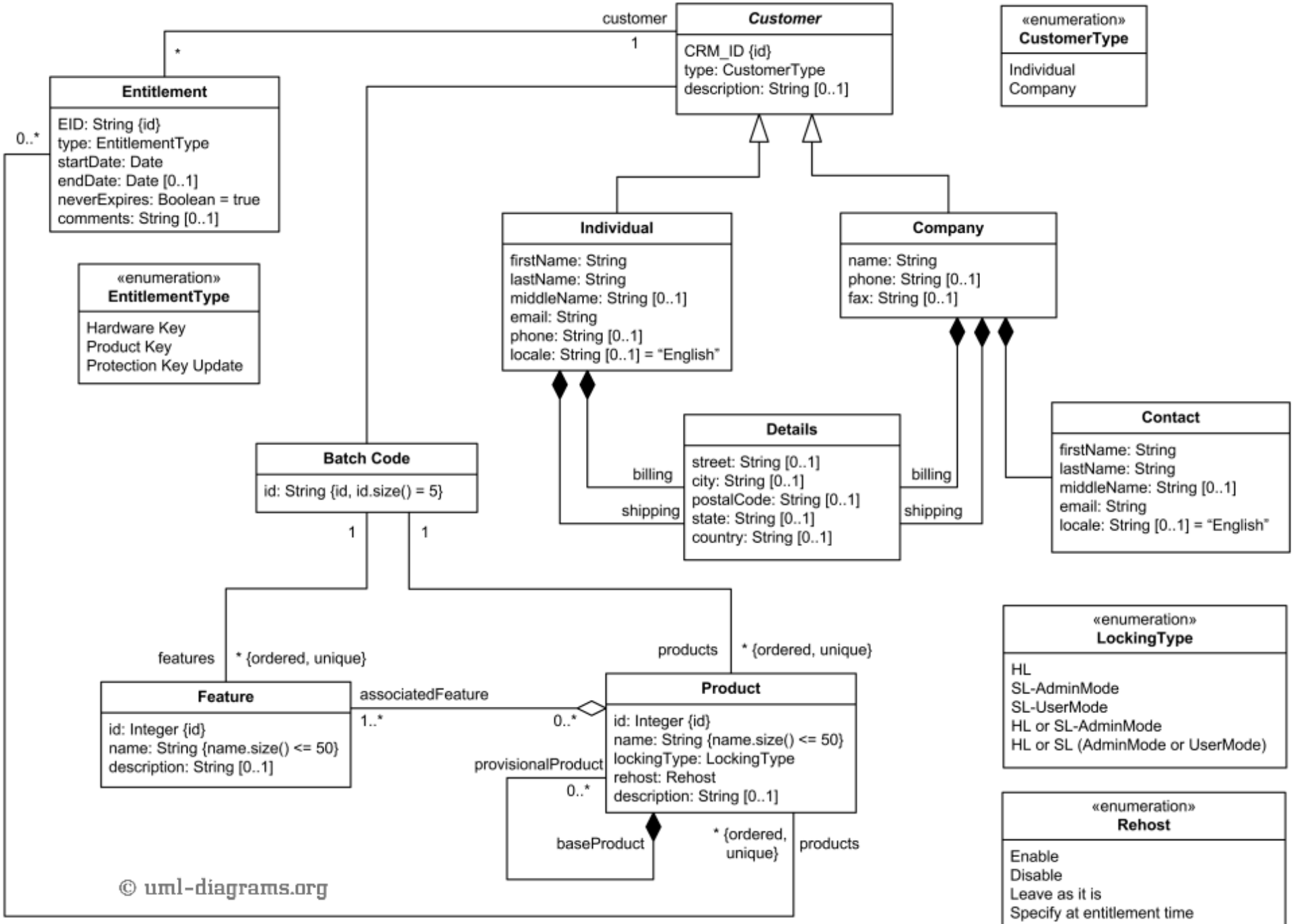


http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/bell_fig4.jpg

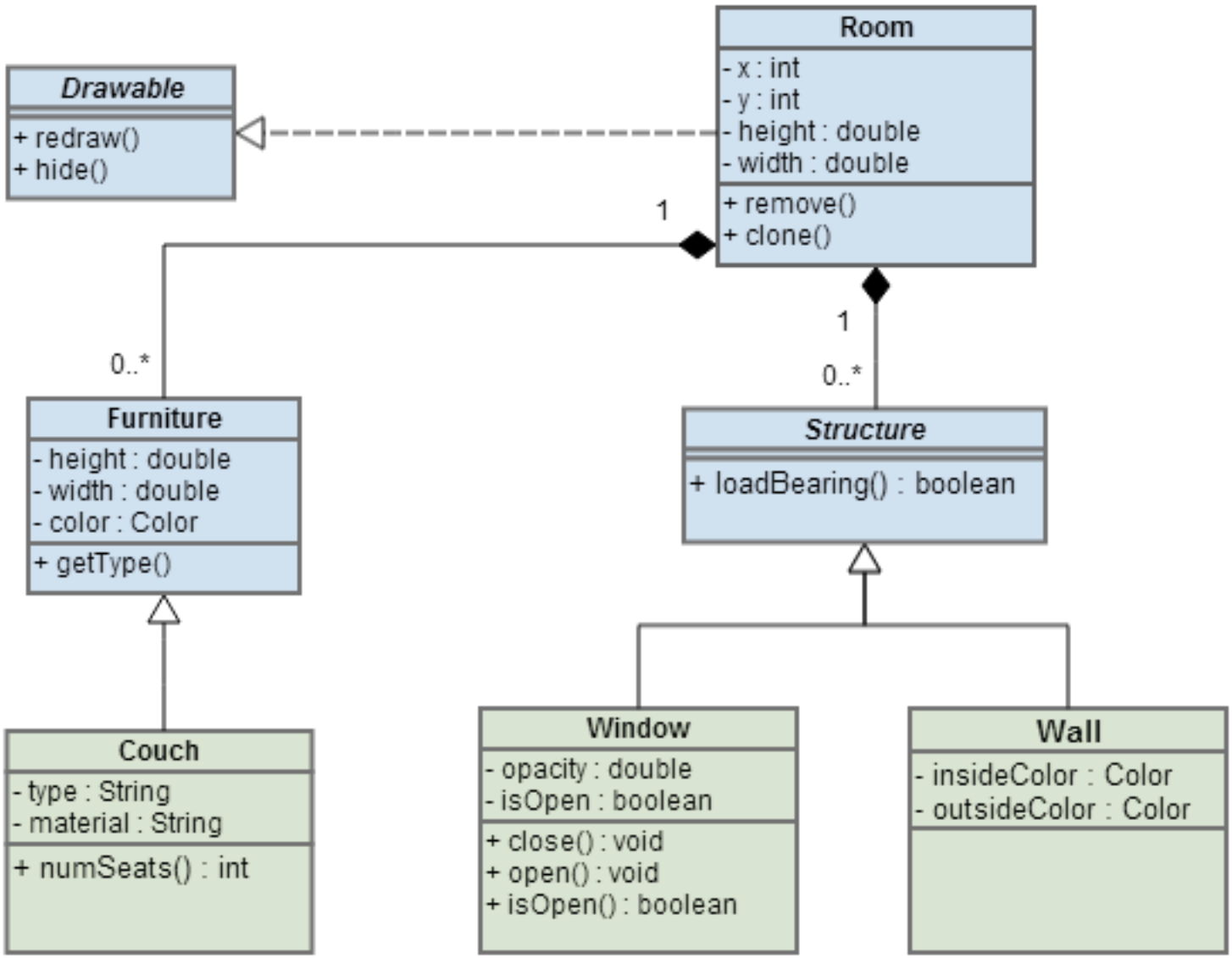


ese

https://cwiki.apache.org/confluence/download/attachments/88604/IoService-connector.png?version=1&modificationDate=1213772392000&api=v2



http://www.uml-diagrams.org/examples/class-diagram-example-hasp-licensing-domain.png



http://www.gliffy.com/go/publish/image/4218693/L.png