CLASS DIAGRAM AND OBJECT DIAGRAM

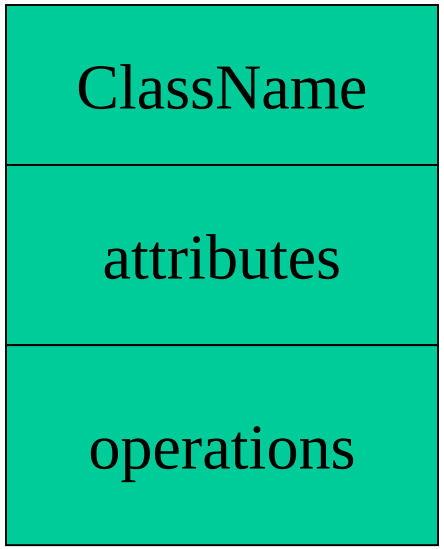
REFERENCES

- https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&sqi=2&ved=0ahUKEwjAsI689tLSAhVIL48KHTeTBAMQFggZMAA&url=https%3A%2F%2Fwww.cs.drexel.edu%2F~spiros%2Fteaching%2FCS575%2Fslides%2Fuml.ppt&usg=AFQjCNGh3szSqGV7GdUsivm6KWKr-ApLuw&bvm=bv.149397726,d.c2I

- https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=13&cad=rja&uact=8&ved=0ahUKEwi90N339tLSAhUIvo8KHauvAswQFghNMAw&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FClass_diagram&usg=AFQjCNEsis0dv8_-e6GVgu_EUwLtMK_SOw&bvm=bv.149397726,d.c2I

- https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=14&cad=rja&uact=8&ved=0ahUKEwi90N339tLSAhUIvo8KHauvAswQFghZMA0&url=https%3A%2F%2Fwww.tutorialspoint.com%2Fuml%2Fuml_class_diagram.htm&usg=AFQjCNEtis6n1gL114w9hECHYxJPNpPw4w&bvm=bv.149397726,d.c2I

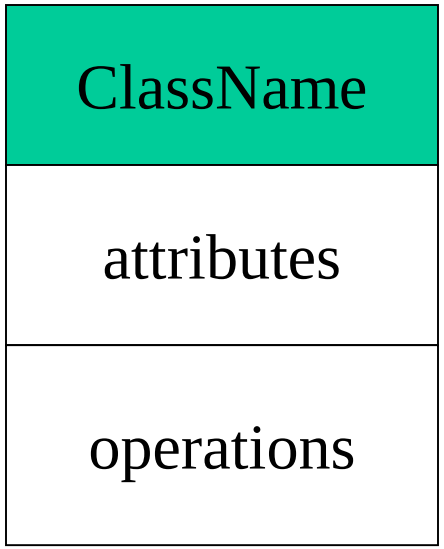
Classes



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

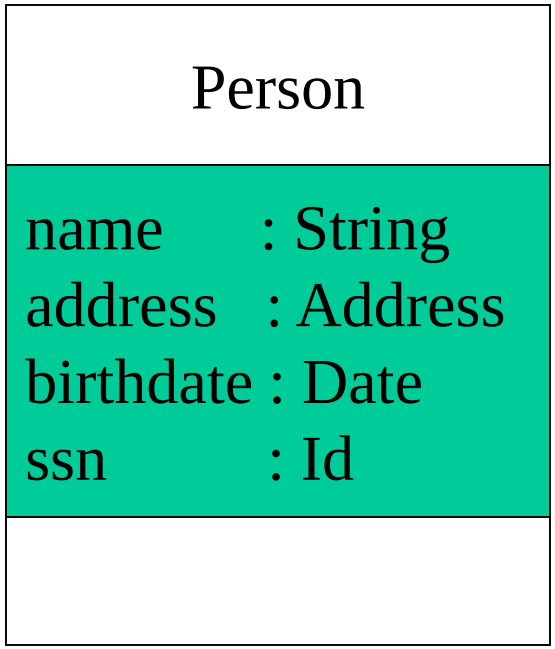
Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations in separate, designated compartments.

Class Names



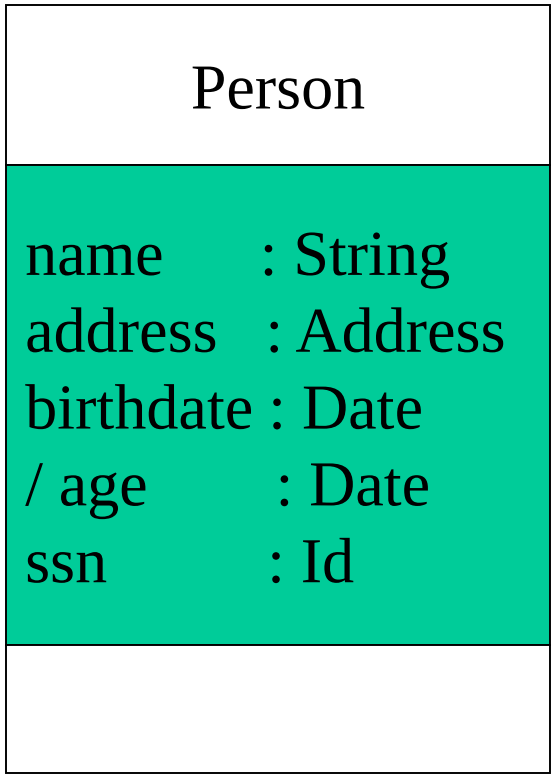
The name of the class is the only required tag in the graphical representation of a class. It always appears in the top-most compartment.

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)



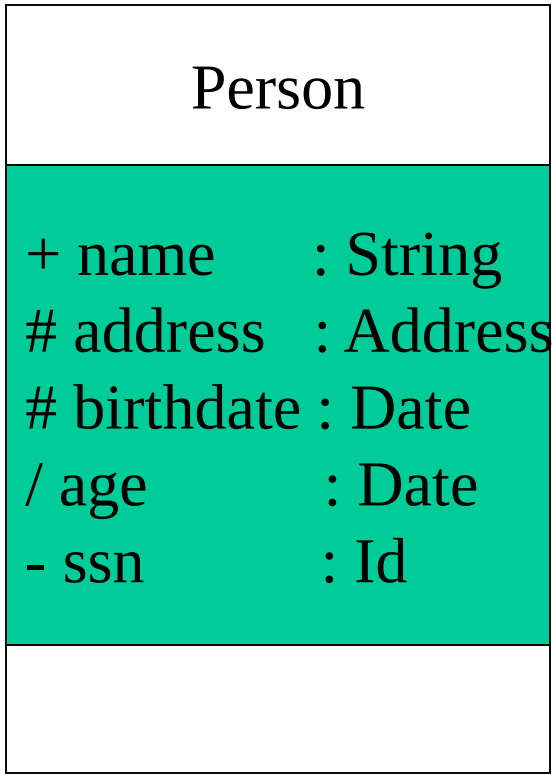
Attributes are usually listed in the form:

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

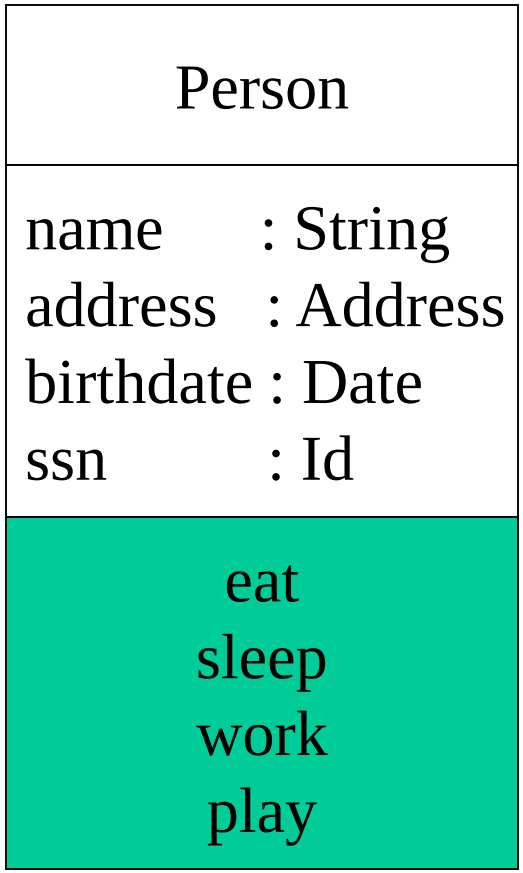
Class Attributes (Cont'd)



Attributes can be:

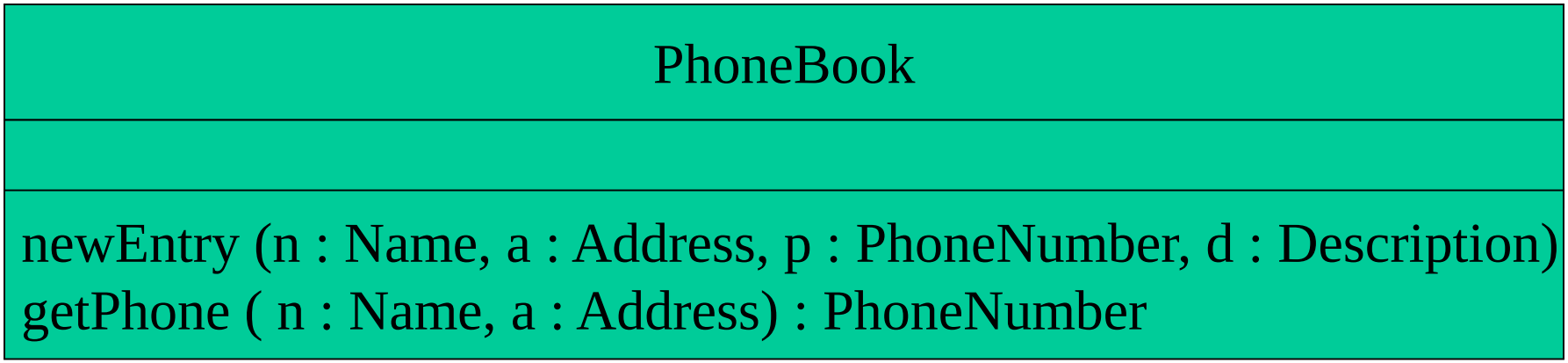
+ public
# protected
- private
/ derived

Class Operations



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

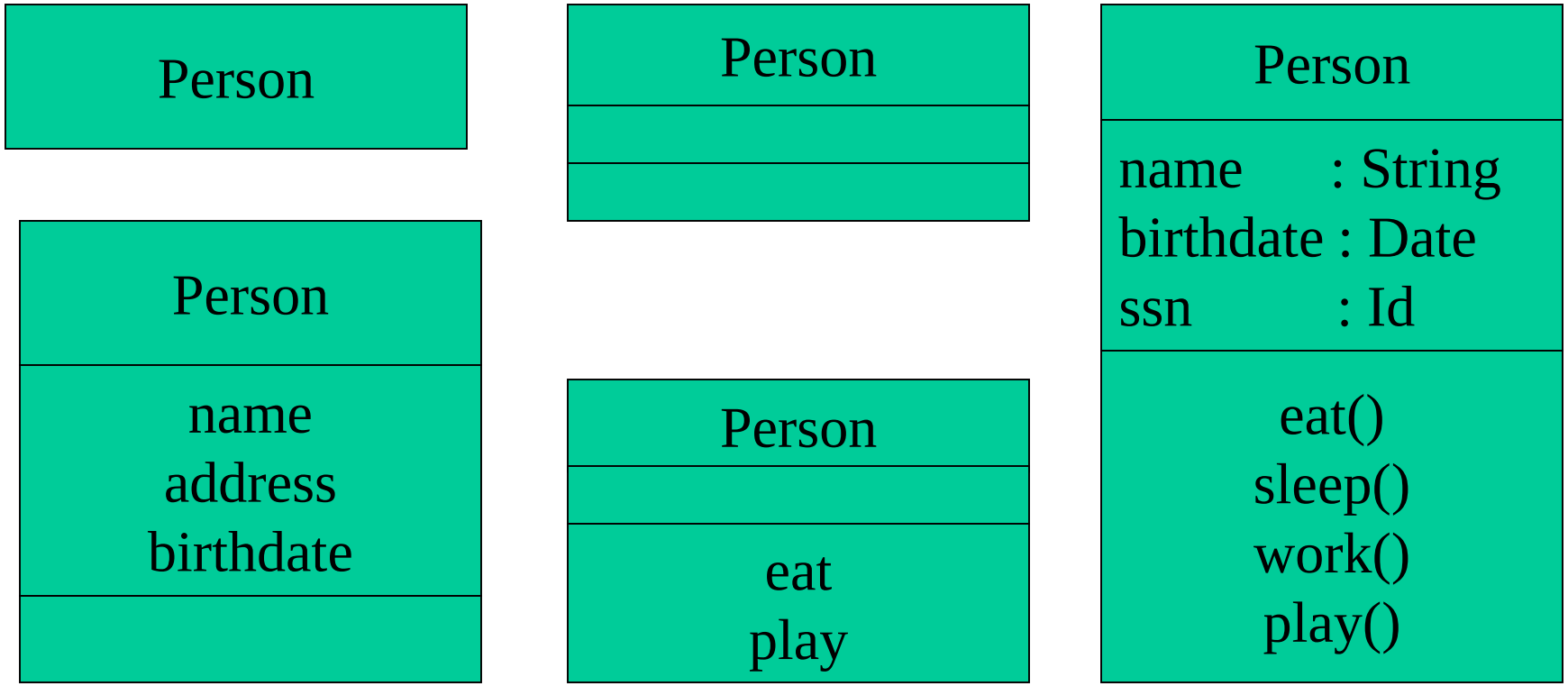
Class Operations (Cont'd)



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.



Relationships

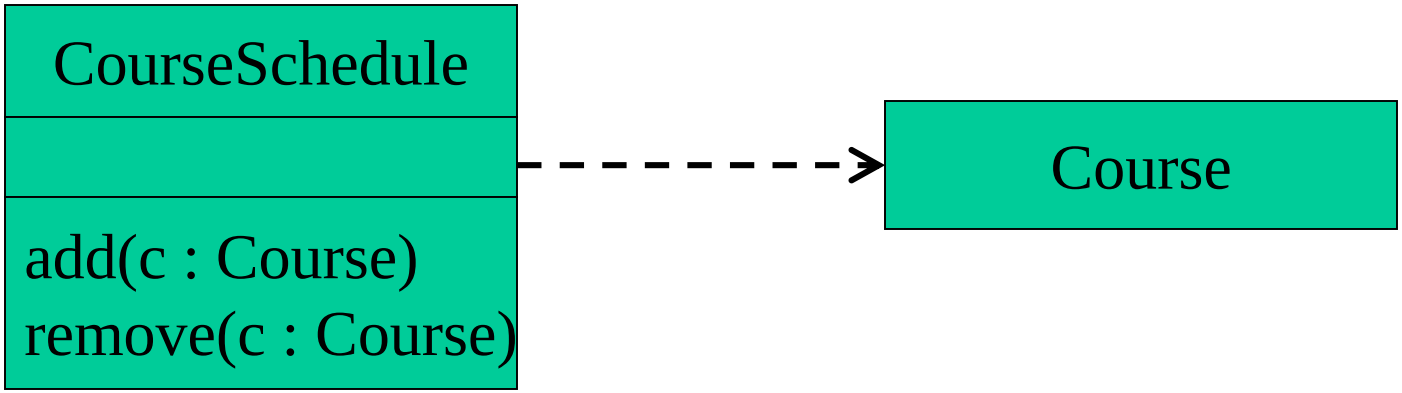
In UML, object interconnections (logical or physical), are modeled as relationships.

There are three kinds of relationships in UML:

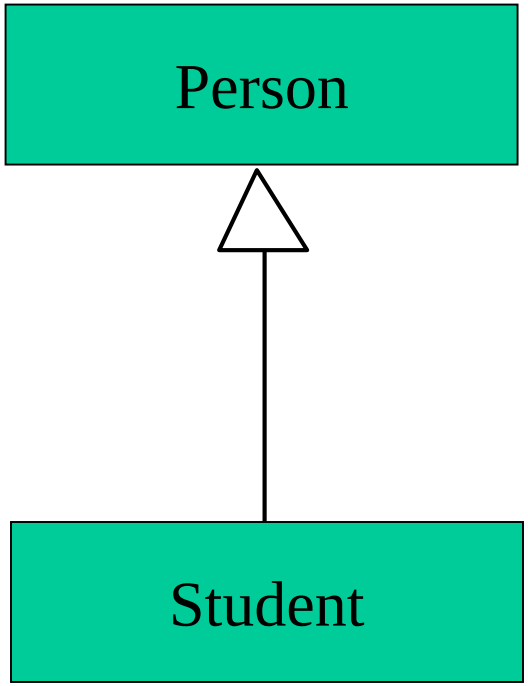
- dependencies
- generalizations
- associations

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*.



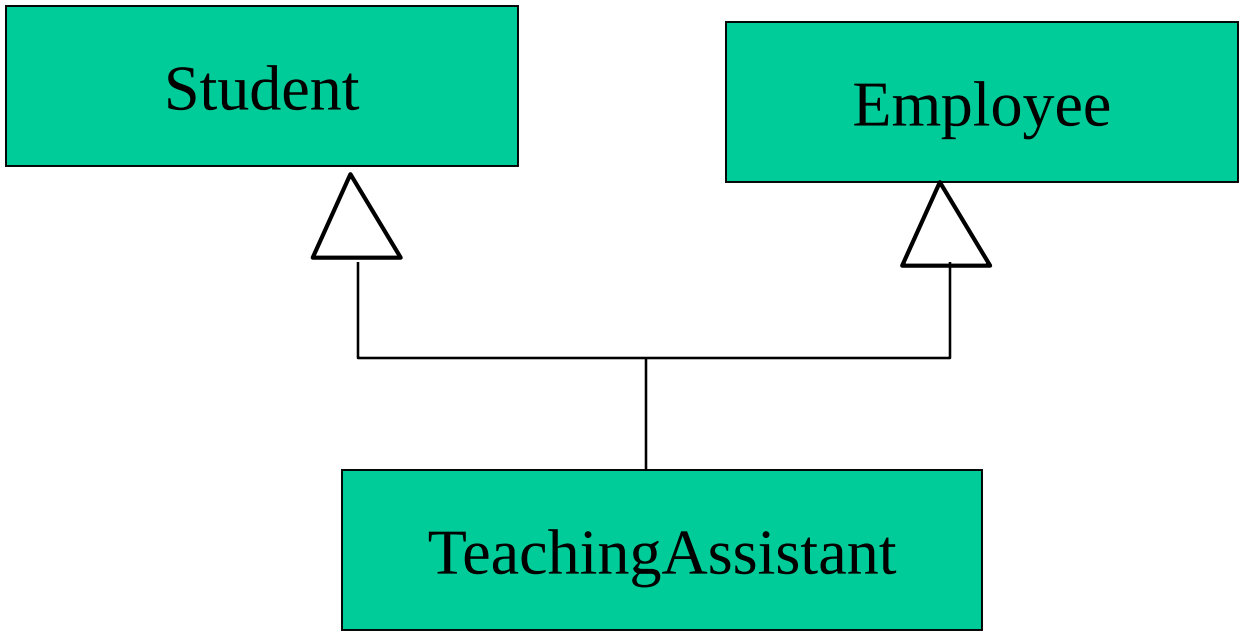
Generalization Relationships



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)

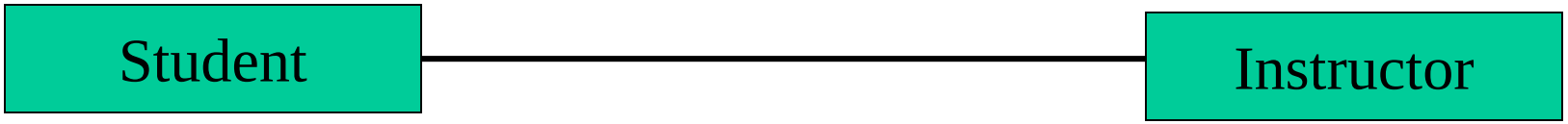
UML permits a class to inherit from multiple superclasses, although some programming languages (*e.g.*, Java) do not permit multiple inheritance.



Association Relationships

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

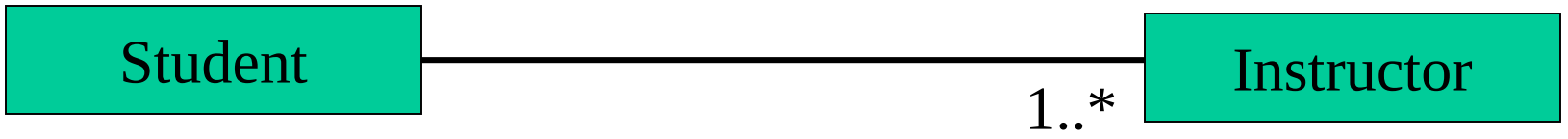
An *association* denotes that link.



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*:

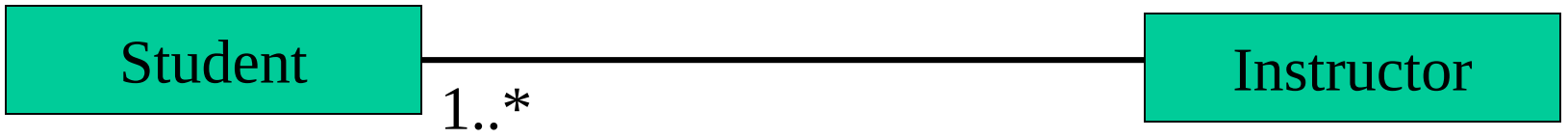


Multiplicity

0	No instances (rare)
0..1	No instances, or one instance
1	Exactly one instance
0..*	Zero or more instances
*****	Zero or more instances
1..*	One or more instances

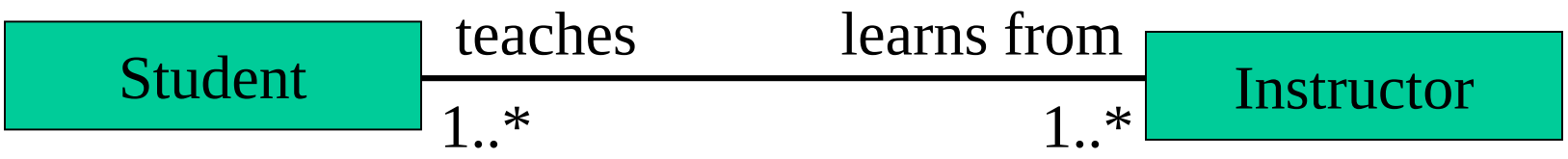
Association Relationships (Cont'd)

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



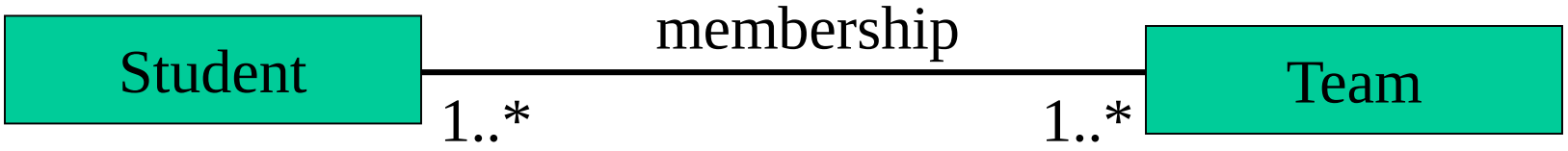
Association Relationships (Cont'd)

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



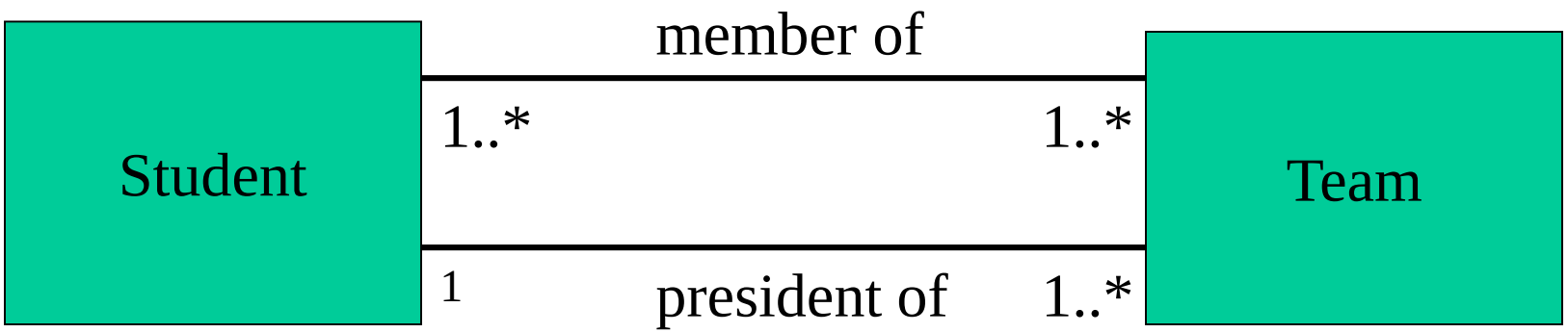
Association Relationships (Cont'd)

We can also name the association.



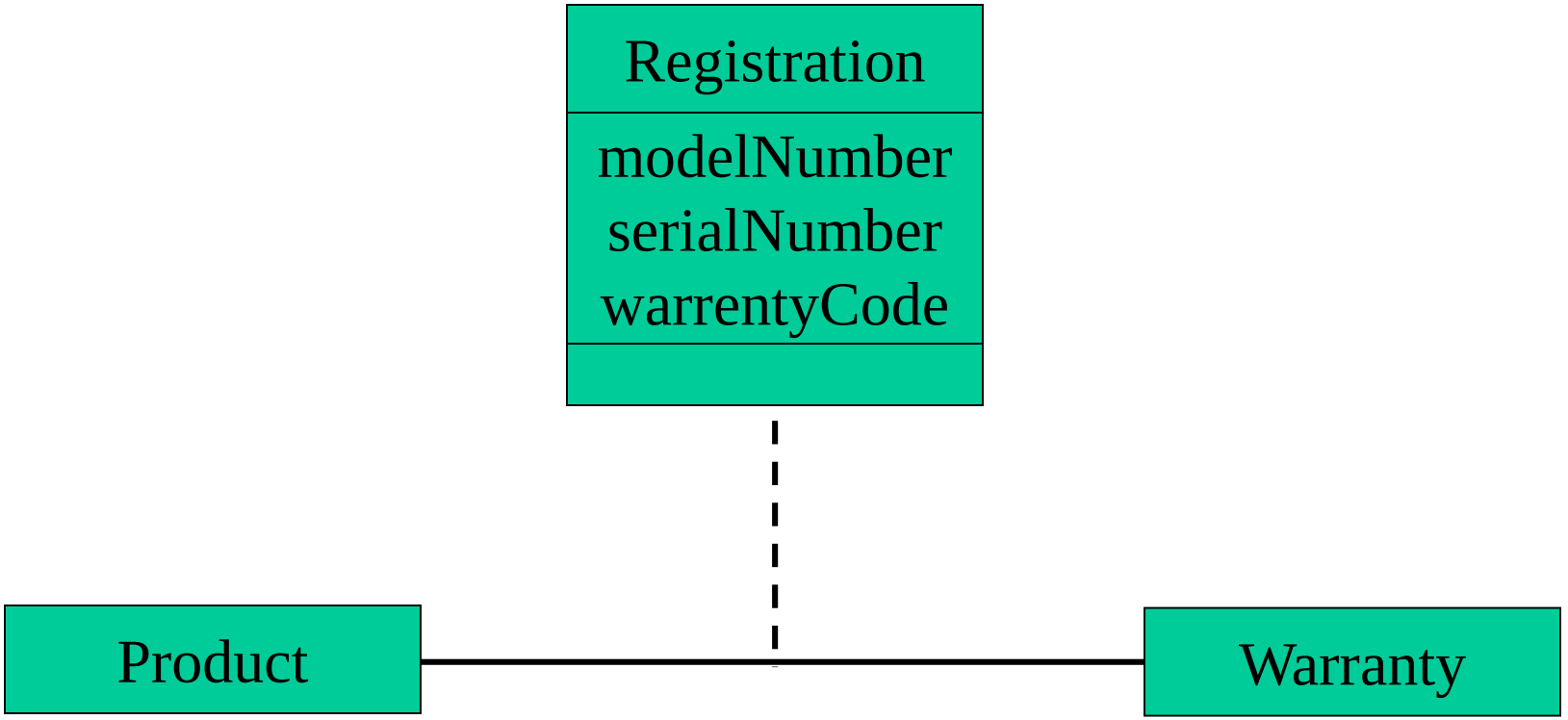
Association Relationships (Cont'd)

We can specify dual associations.



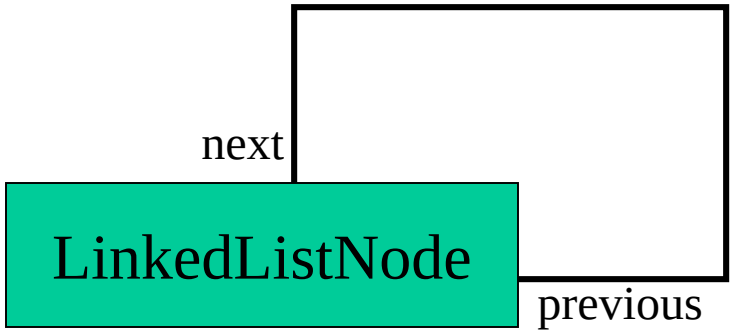
Association Relationships (Cont'd)

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



Association Relationships (Cont'd)

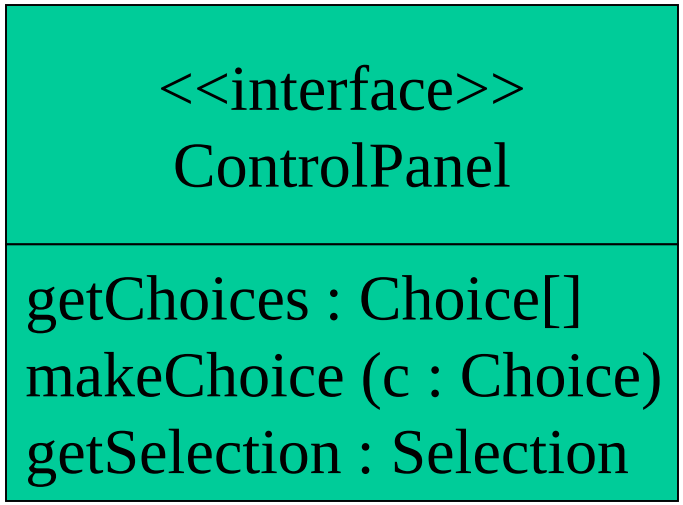
A class can have a *self association*.



Interfaces

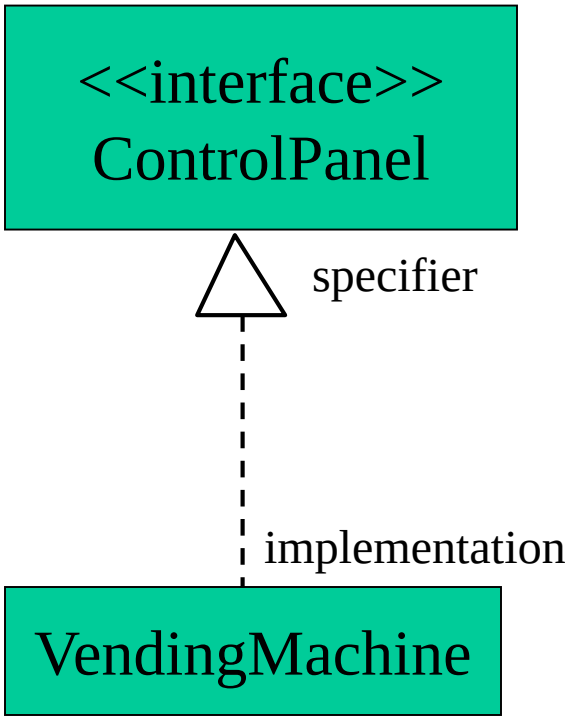
An *interface* is a named set of operations that specifies the behavior of objects without showing their inner structure. It can be rendered in the model by a one- or two-compartment rectangle, with the *stereotype* <<interface>> above the interface name.

Interface Services



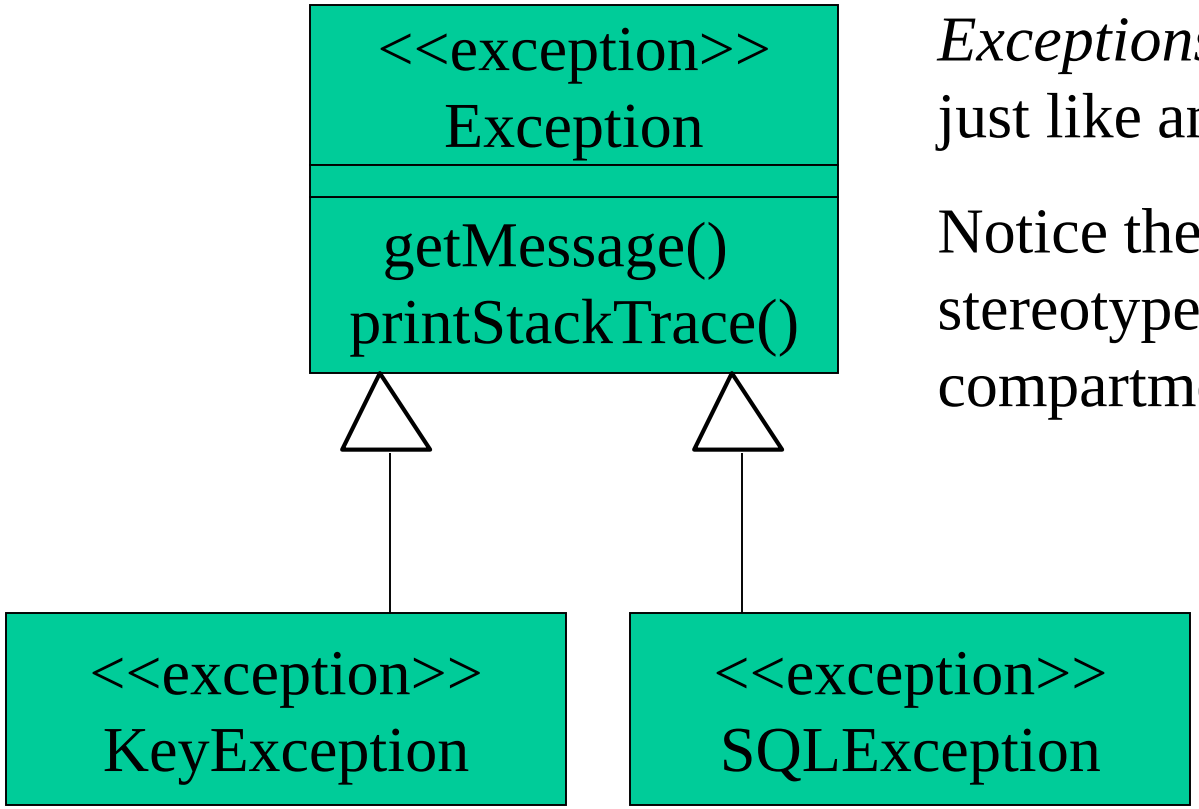
Interfaces do not get instantiated. They have no attributes or state. Rather, they specify the services offered by a related class.

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.

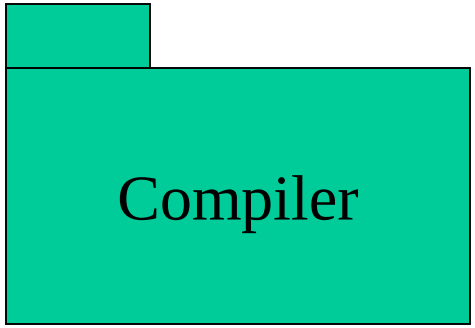
Exceptions



Exceptions can be modeled just like any other class.

Notice the <<exception>> stereotype in the name compartment.

Packages



A *package* is a container-like element for organizing other elements into groups.

A package can contain classes and other packages and diagrams.

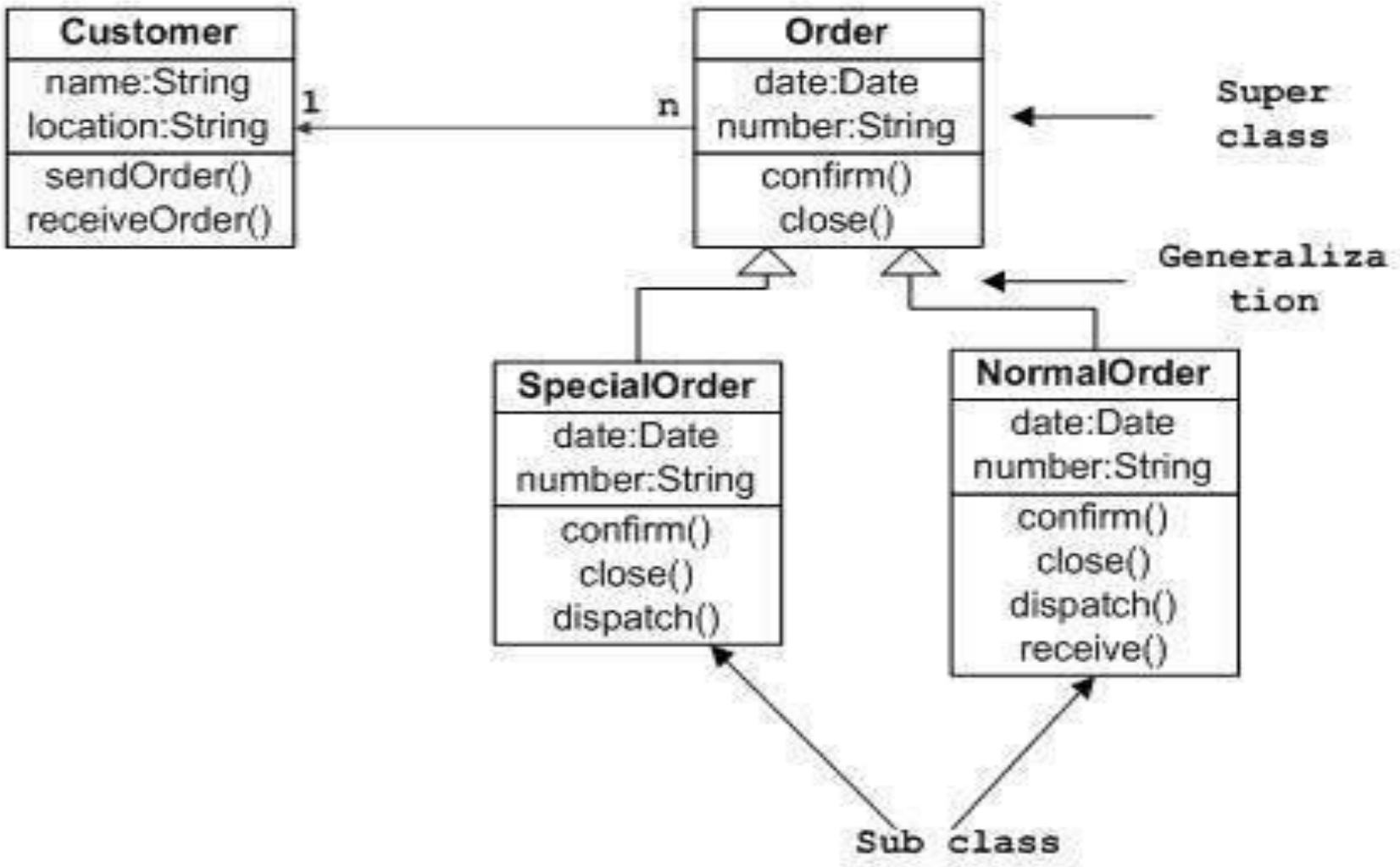
Packages can be used to provide controlled access between classes in different packages.

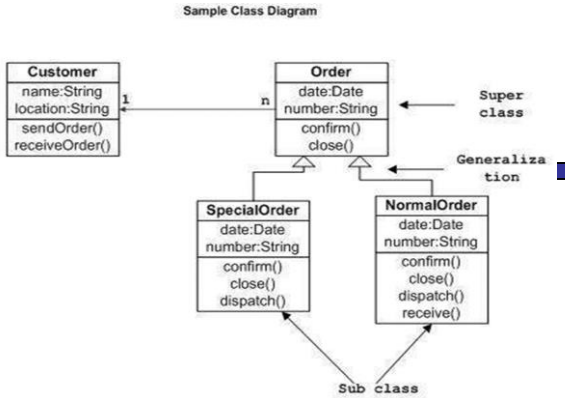
OBJECT DIAGRAM

Values of different elements need to be captured at a particular time state to include in the object diagram.

If a different time of purchase is considered then these values will change accordingly.

Sample Class Diagram





Object diagram of an order management system