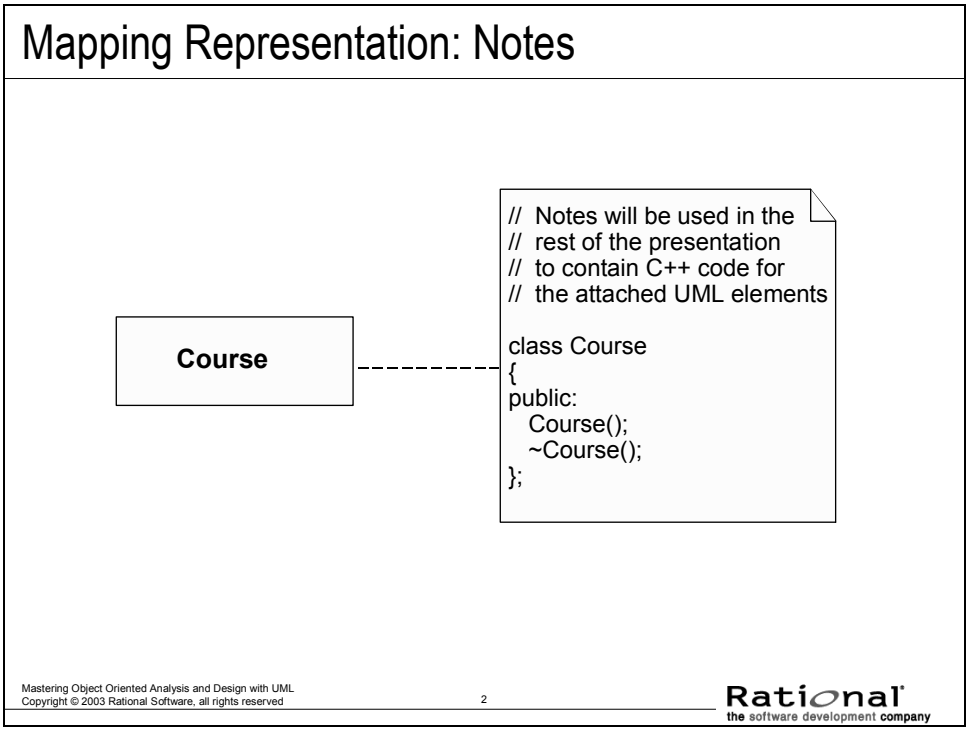
▸ ▸ ▸ Module 3
UML to C++ Mapping

Mastering Object-Oriented Analysis and Design with UML

Appendix: UML to C++ Mapping

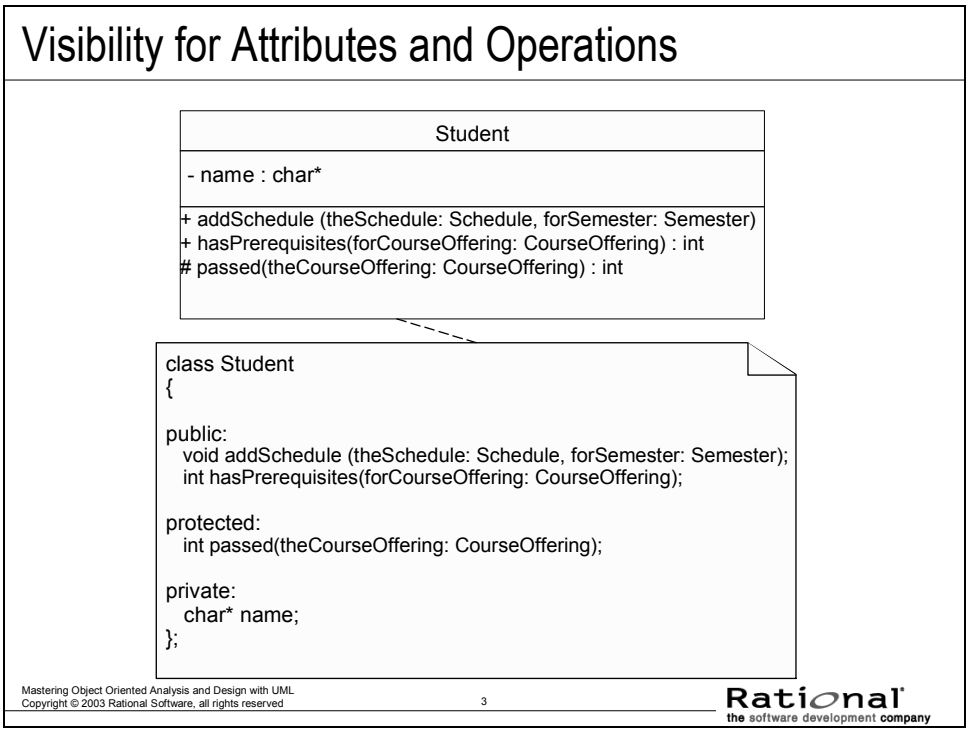
Topics

Mapping Representation: Notes



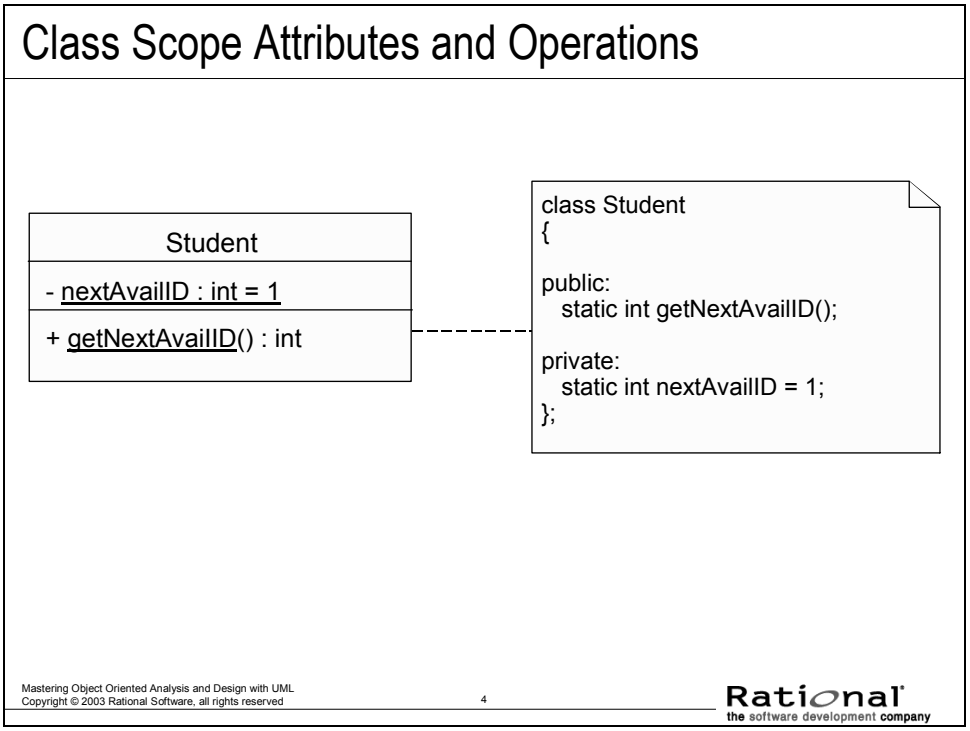
If you remember from earlier in the course, a note can be added to any UML element. It is represented as a 'dog eared' rectangle. The note may be anchored to a specific element(s) with a dashed line

Visibility for Attributes and Operations



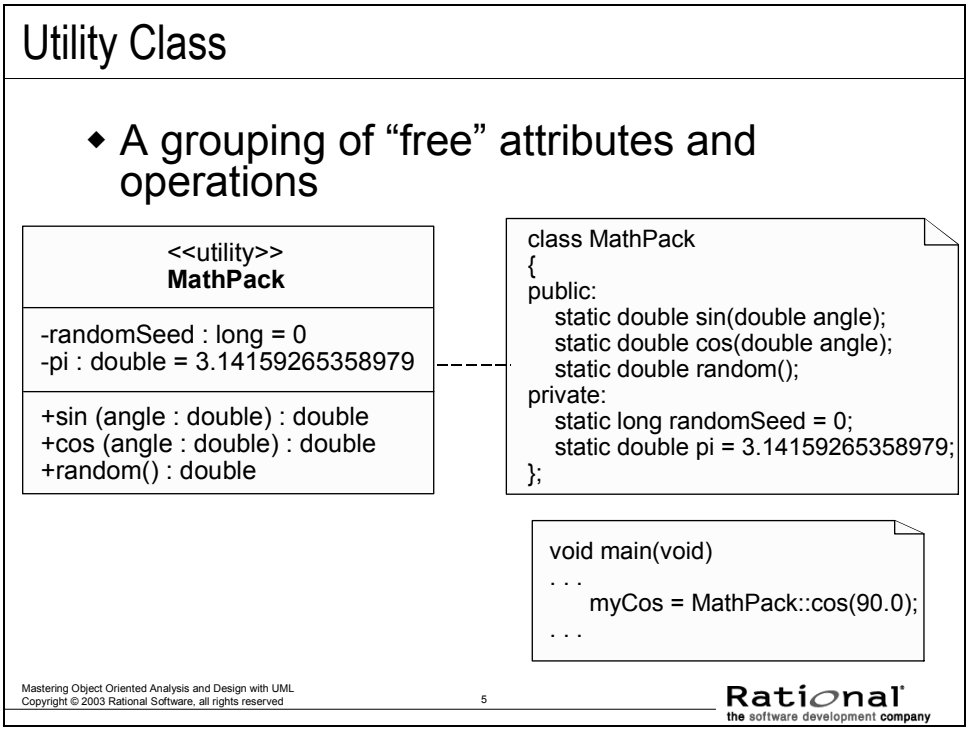
A subset of the Student class operations and attributes are shown above. For this example, we included a subset to demonstrate the UML construct we are emphasizing.

Class Scope Attributes and Operations



A subset of the Student class operations and attributes are shown above. For this example, we included a subset to demonstrate the UML construct we are emphasizing.

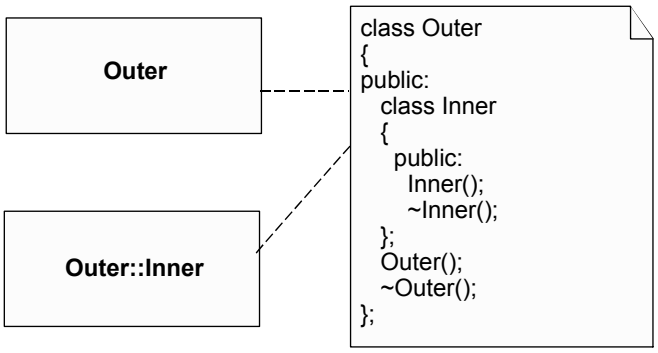
Utility Class



Nested Class

Nested Class

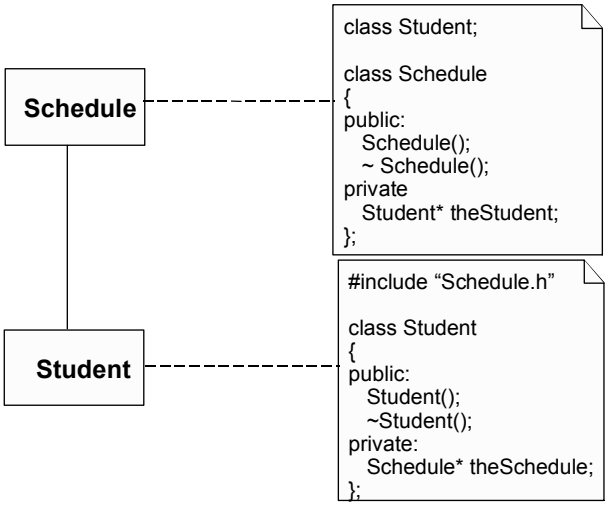
- Hide a class that is relevant only for implementation



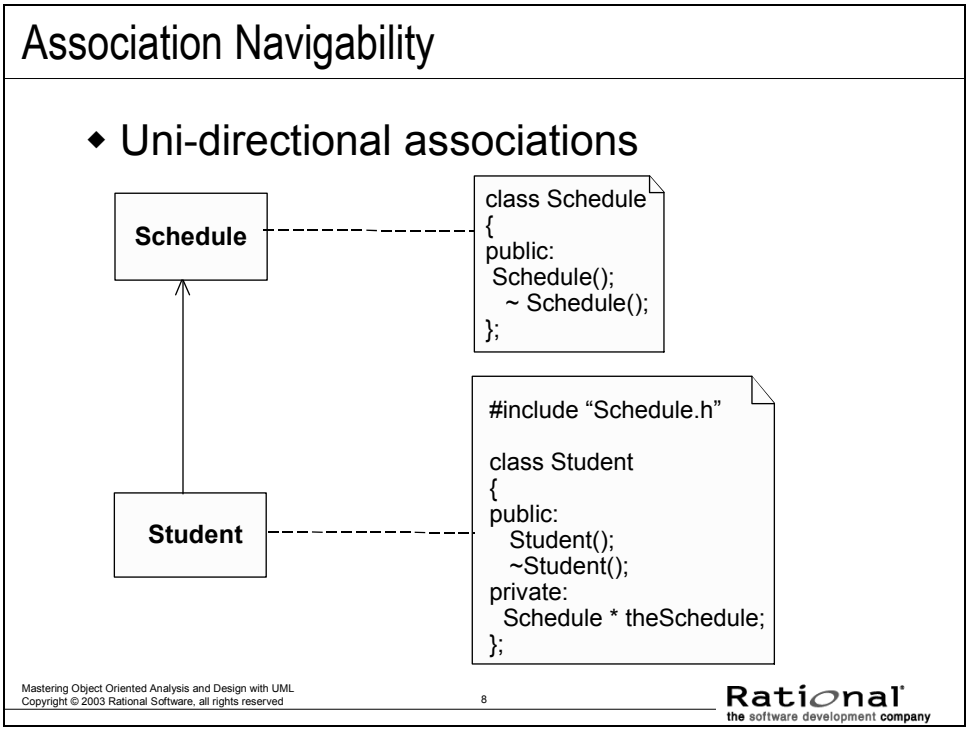
Associations

Associations

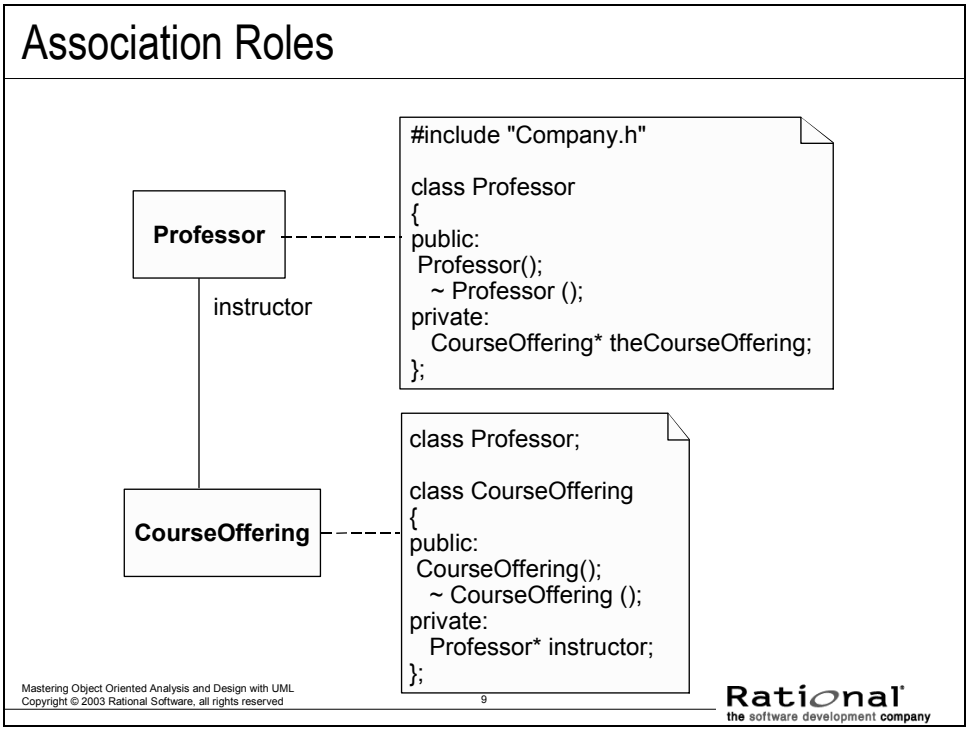
- Bi-directional associations



Association Navigability

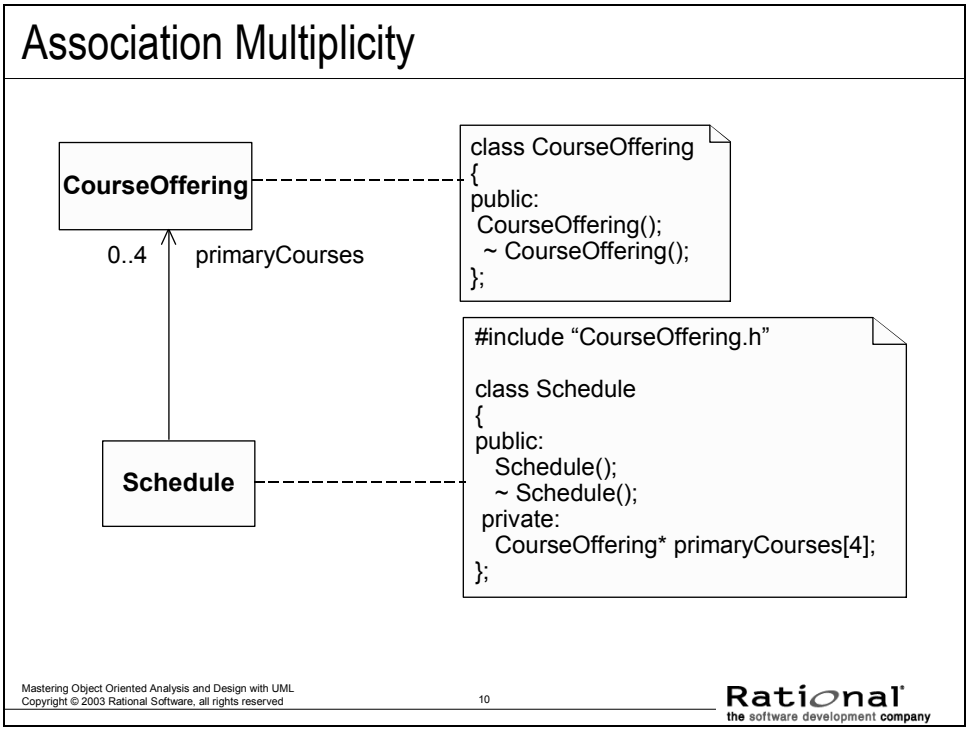


Association Roles



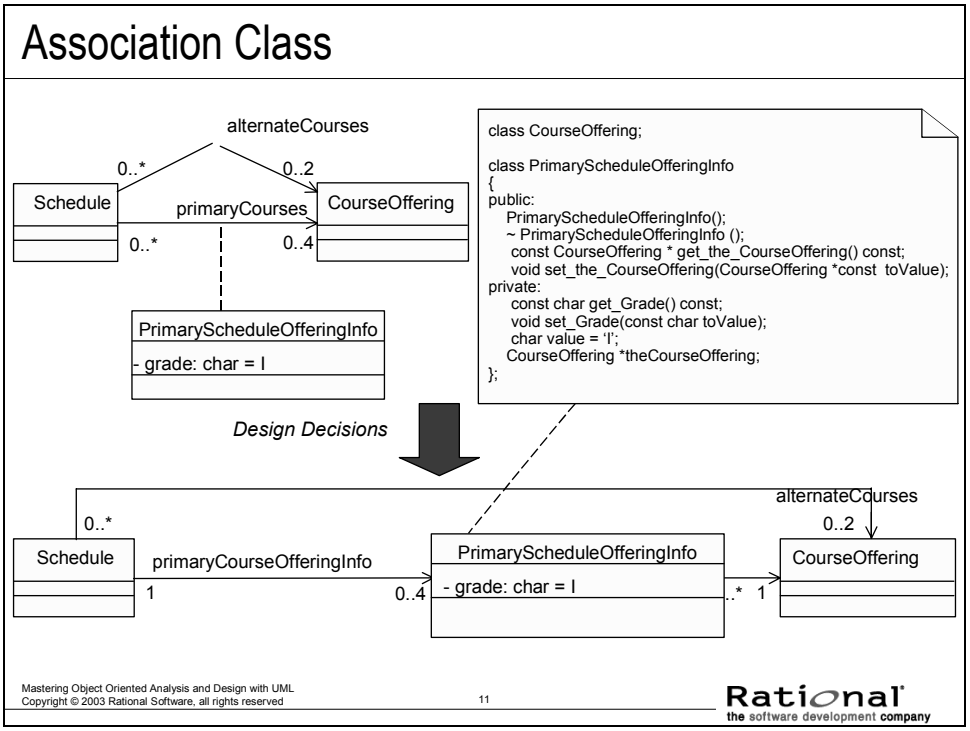
Roles on the end of the association can add clarity.

Association Multiplicity



Multiplicity is the number of instances of one class in relation to the other.

Association Class

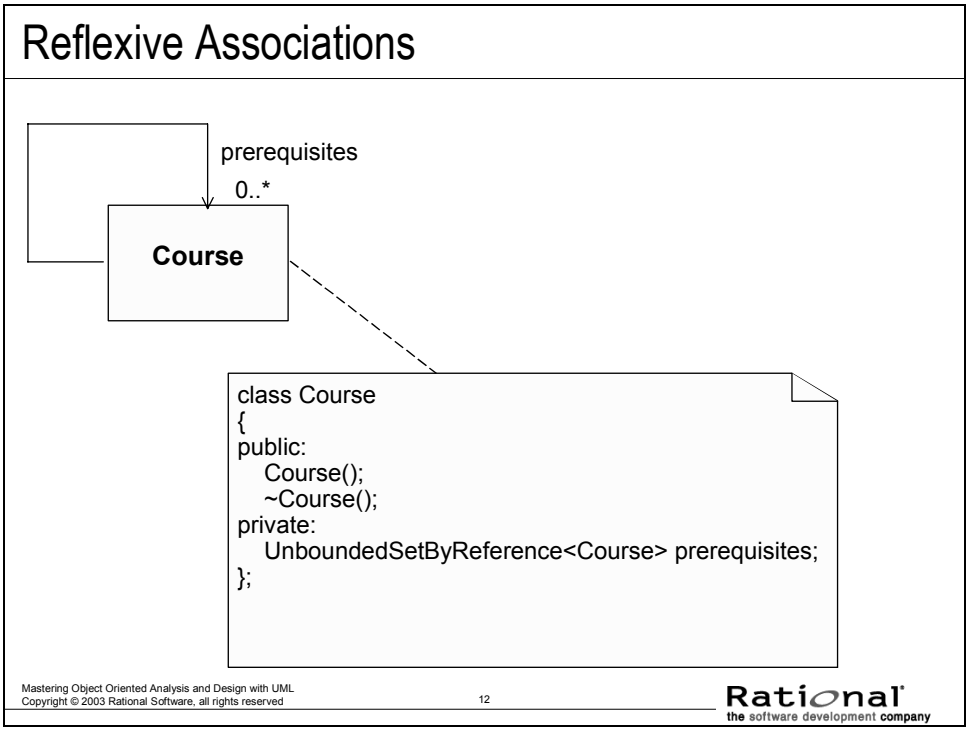


Remember, an association class is a class that is connected to an association. There is an instance of the association class for every instance of the relationship (e.g., for every link).

During design, some decisions are made regarding navigation between the involved classes.

A subset of the class operations and attributes are shown above. For this example, we included a subset to demonstrate the UML construct we are emphasizing.

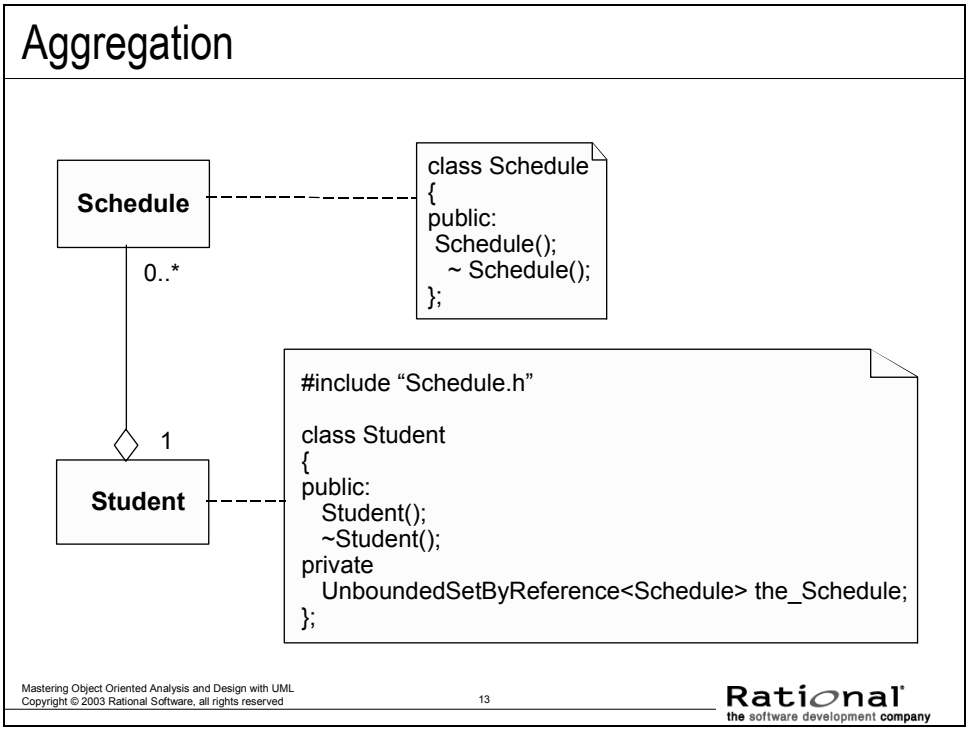
Reflexive Associations



A class may have an association with objects of the same type.

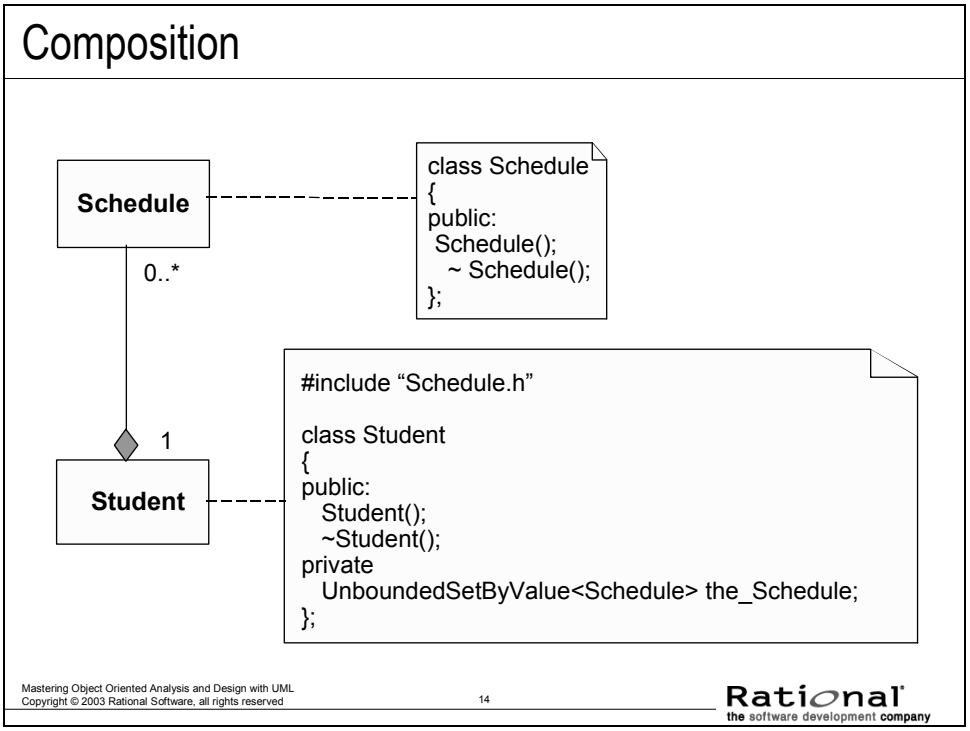
In the above example, unboundedSetByReference is just an example of a template (that supports and unbounded list of courses).

Aggregation



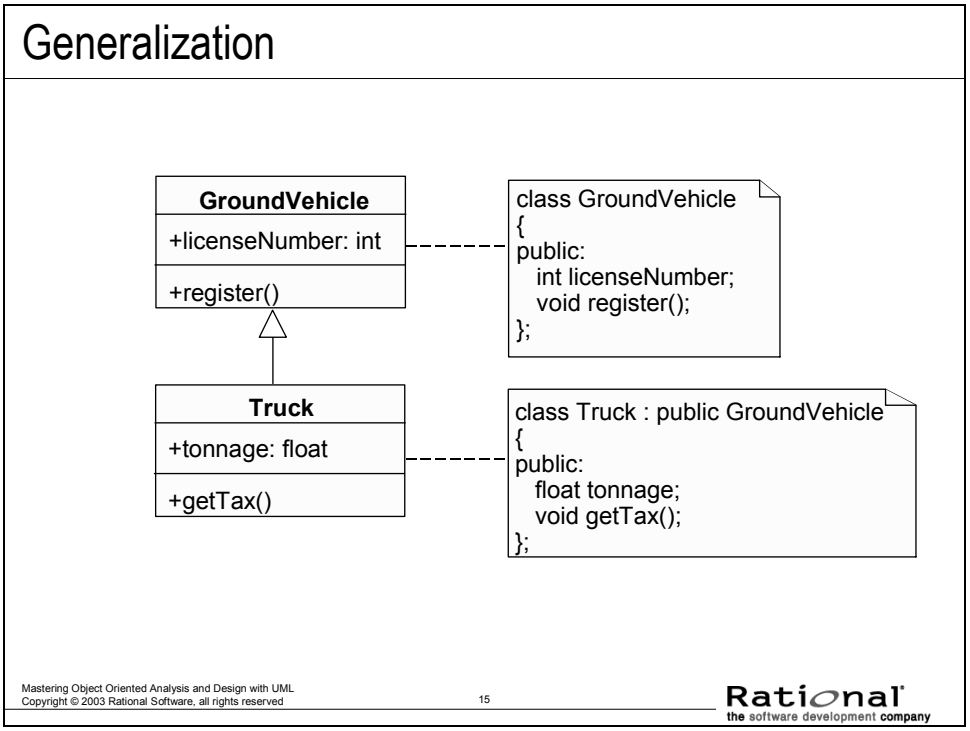
Note: UnboundedSetByReference represents a parameterized class that has been instantiated with “Schedule”. It could be replaced by any reusable list class available in the programming environment.

Composition



Note: UnboundedSetByReference represents a parameterized class that has been instantiated with “Schedule”. It could be replaced by any reusable list class available in the programming environment.

Generalization

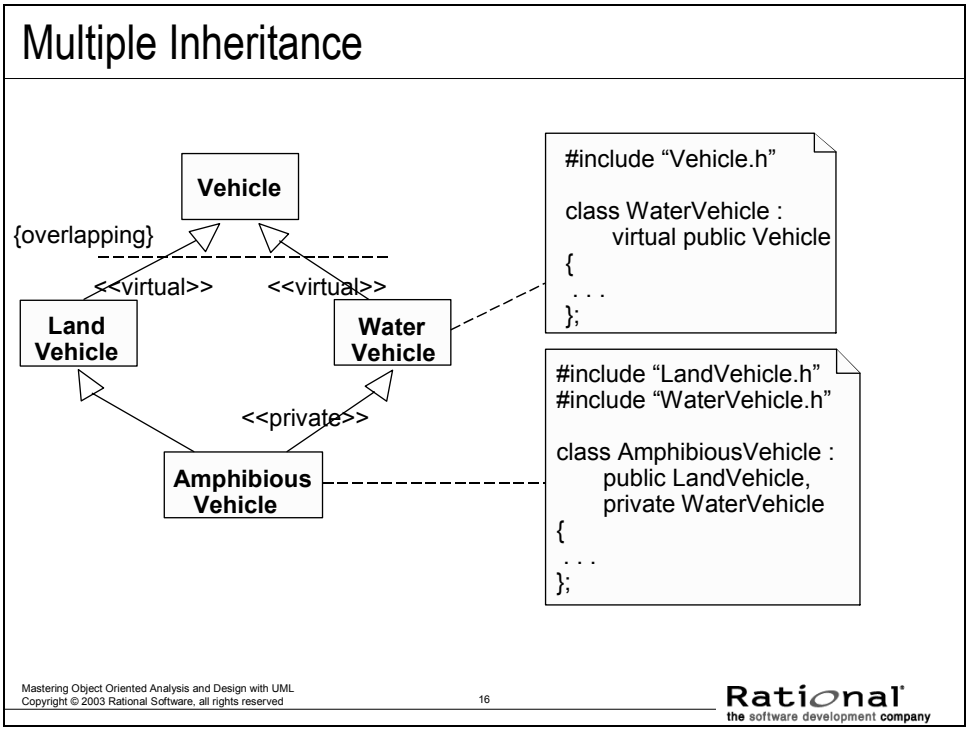


Remember generalization is a "is-a" or "kind-of" association. It is used to represent generalization / specialization.

Generalization is modeled as an open triangular arrowhead pointing to the base on the base class end.

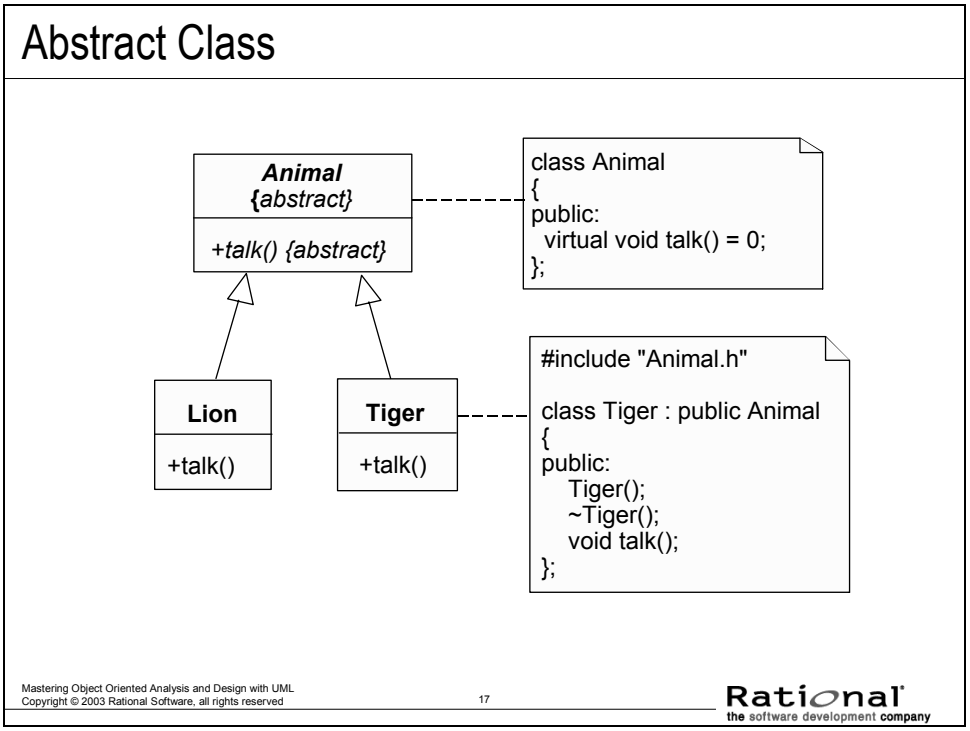
In C++, inheritance is a language implementation mechanism for specialization. The use of inheritance does not guarantee substitutability (i.e., is-a programming), so the distinction is important. Delegation is another way to implement specialization.)

Multiple Inheritance



Remember, subclasses that are not mutually exclusive can be annotated with the UML {overlapping} constraint. This supports multiple inheritance.

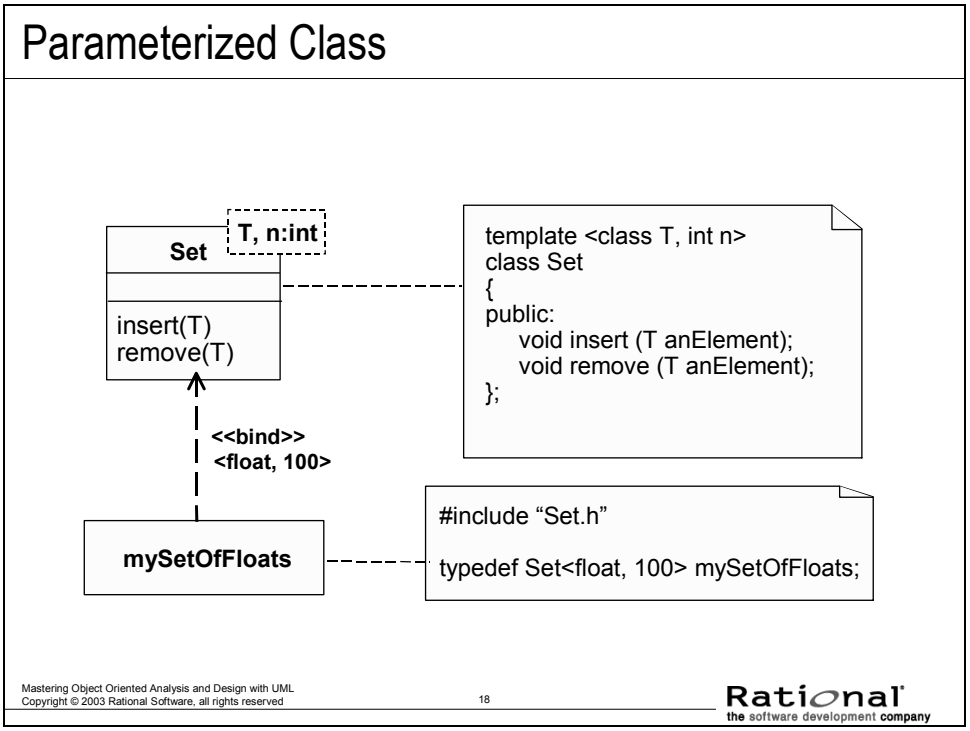
Abstract Class



Remember, an abstract class is a class for which no instances are created.

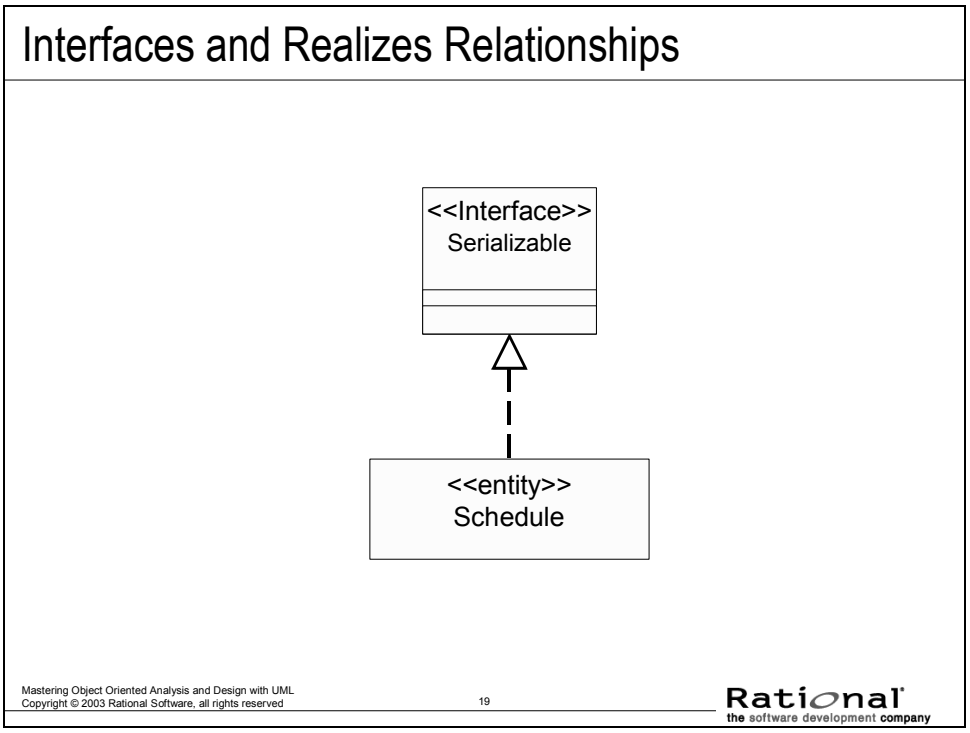
In C++, an abstract class contains at least one pure virtual function.

Parameterized Class



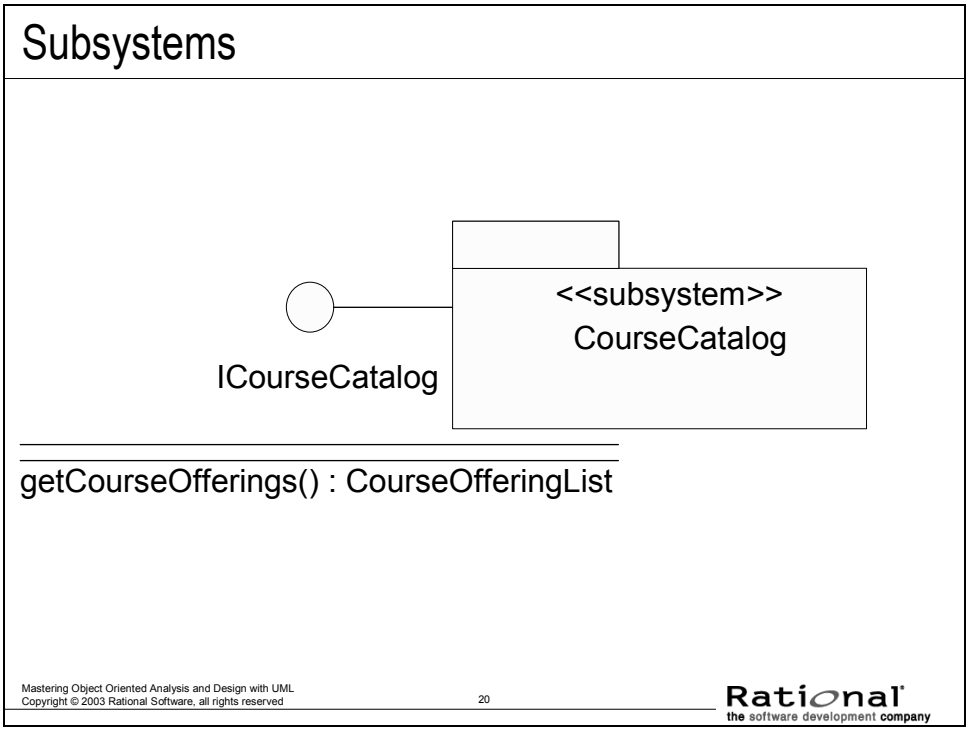
Remember, a parameterized class is a class which defines other classes. They are often used for container classes.

Interfaces and Realizes Relationships



C++ does not directly support interfaces and the realizes relationship. Generalization and abstract base classes are used to simulate realizes and interfaces. Abstract base classes are used for the interfaces. To "realize" an interface, another class inherits from the abstract base class.

Subsystems



As discussed within the OOAD course, subsystems are the Design Model representation for components.

Subsystems do not map to a specific C++ language construct.