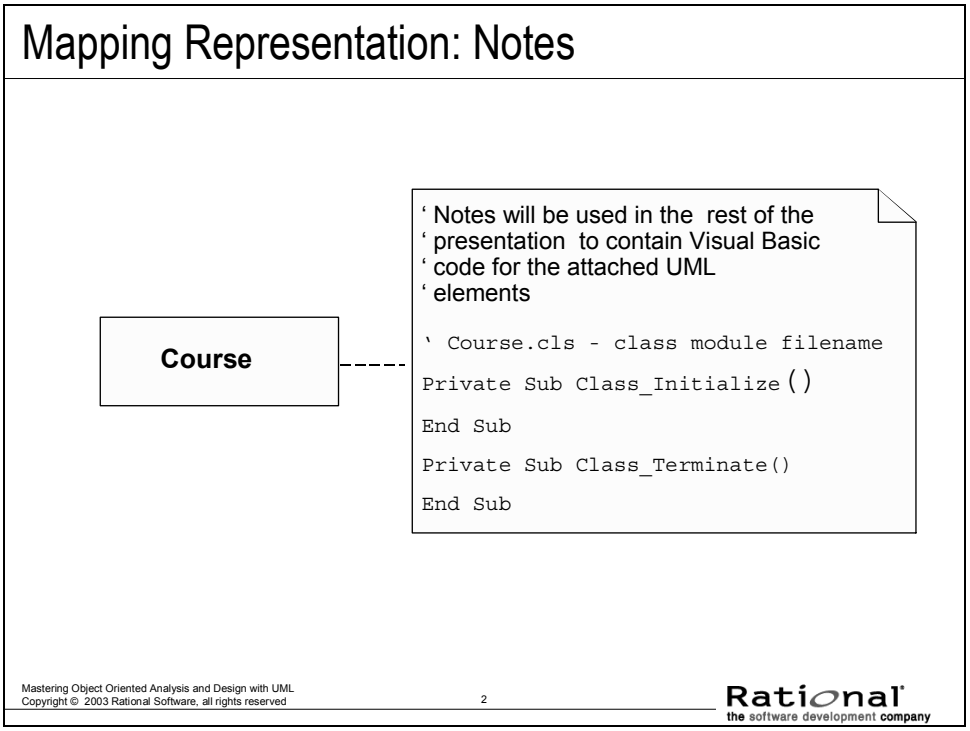
▸ ▸ ▸ Module 5
UML to Visual Basic Map

Mastering Object-Oriented Analysis and Design with UML

Appendix: UML to Visual Basic Map

Topics

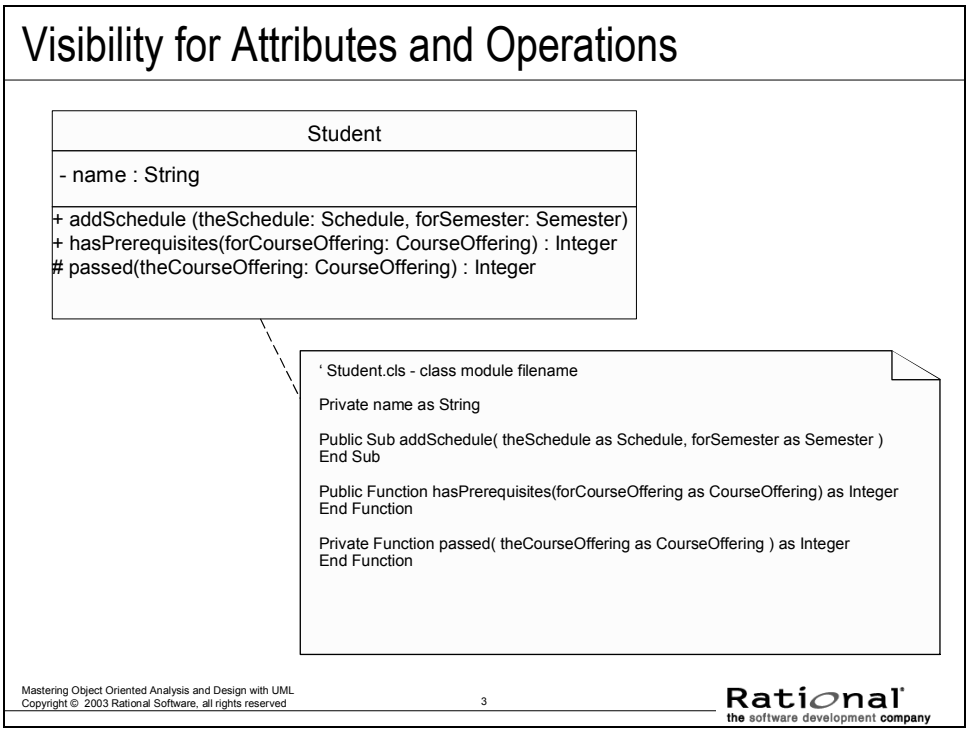
Mapping Representation: Notes



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

Visual Basic does not have a class declaration like other programming languages. In Visual Basic you create a special code module called a "class module". VISUAL BASIC gives the code module a .CLS extension. The code module has a property that defines the name of the class in the class module.

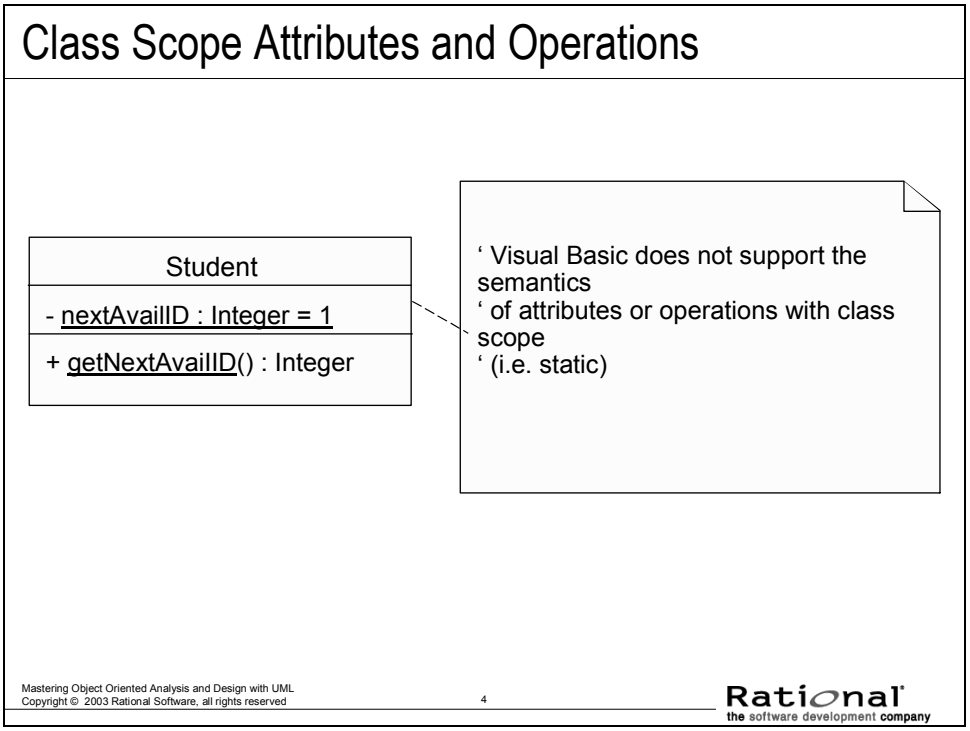
Visibility for Attributes and Operations



Visual Basic does not support protected access.

Rose98 maps protected access in the design model to private access in Visual Basic.

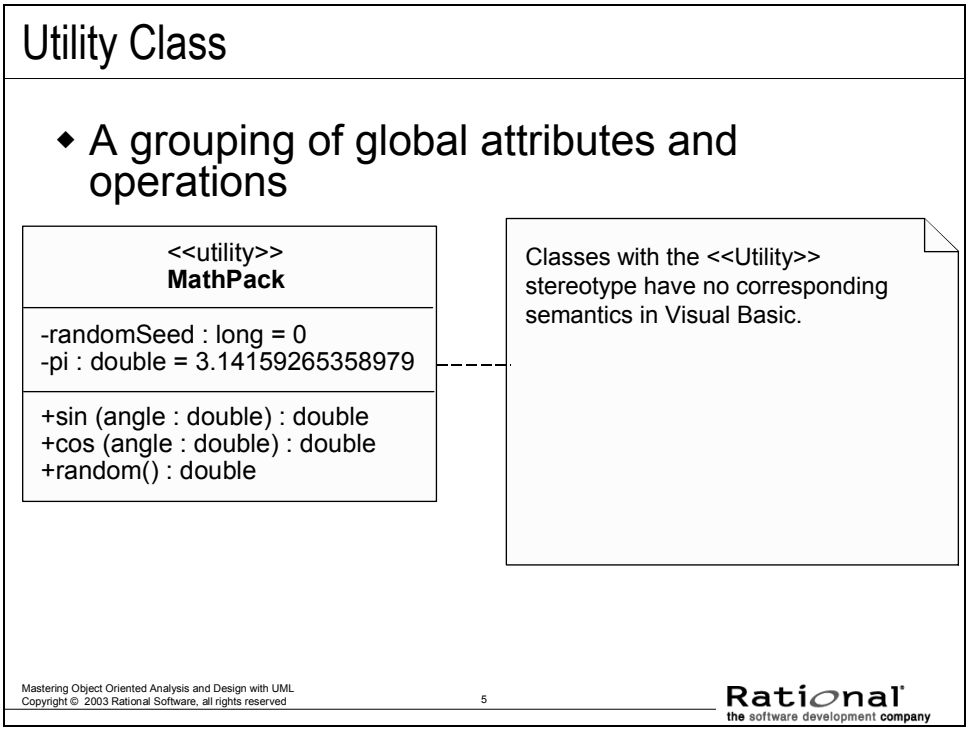
Class Scope Attributes and Operations



Local variables within Visual Basic procedures (subs or functions) can be declared static.

Procedures can also be declared static. If a procedure is declared to be static it means that ALL the local variables for the procedure will be static and retain their values between calls to the procedure.

Utility Class



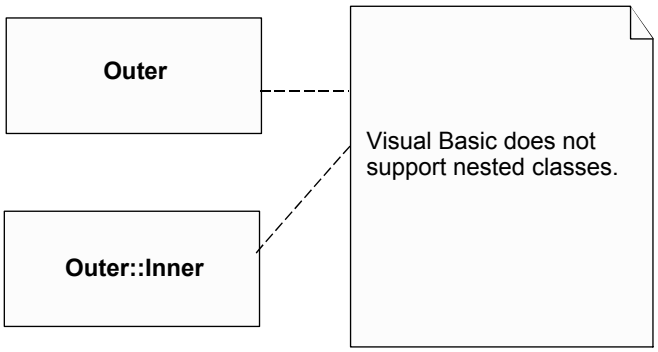
Because Visual Basic does not support classes with static attributes static operations, classes with the <<Utility>> stereotype have no semantic meaning that can be represented in Visual Basic.

The closest concept in Visual Basic is the module. A module is simply a source code file that is composed of data and free functions. Rose98 - when reverse engineering the files from Visual Basic create a ClassUtility instead of a Class and applies a <<Module>> stereotype.

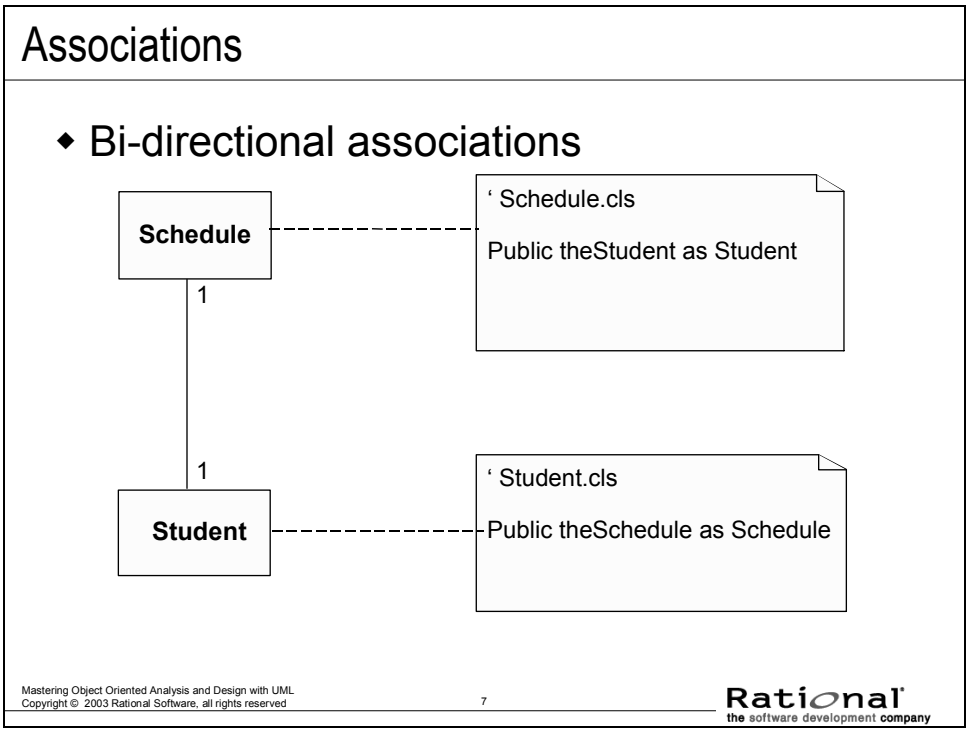
Nested Class

Nested Class

- Hide a class that is relevant only for implementation

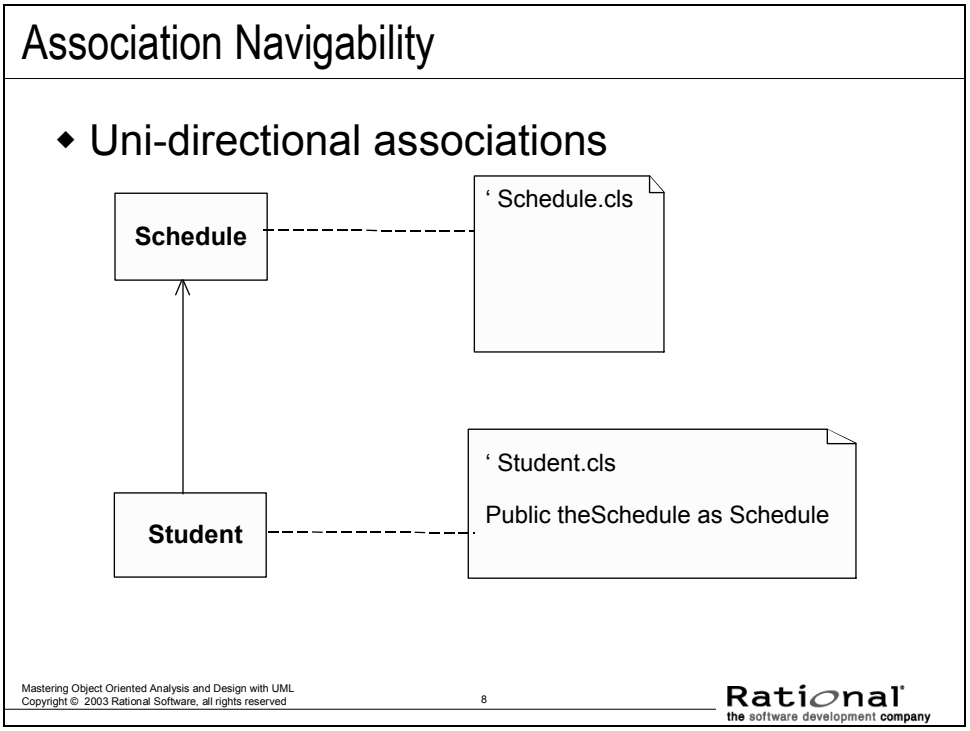


Associations



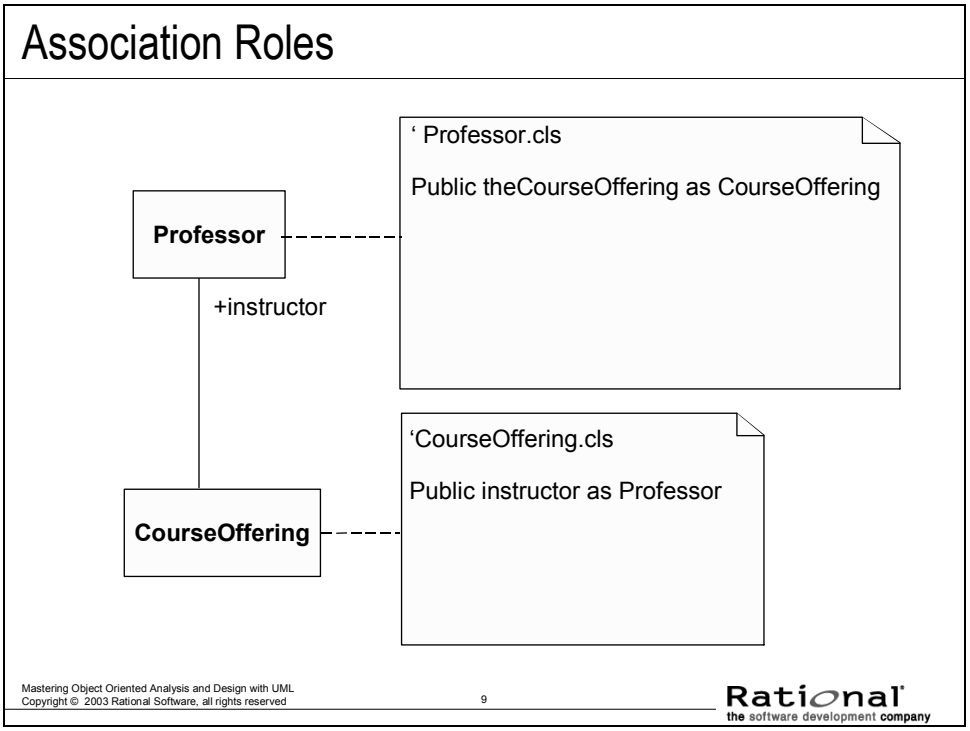
Rose defaults the access on roles, if not specified, to public

Association Navigability



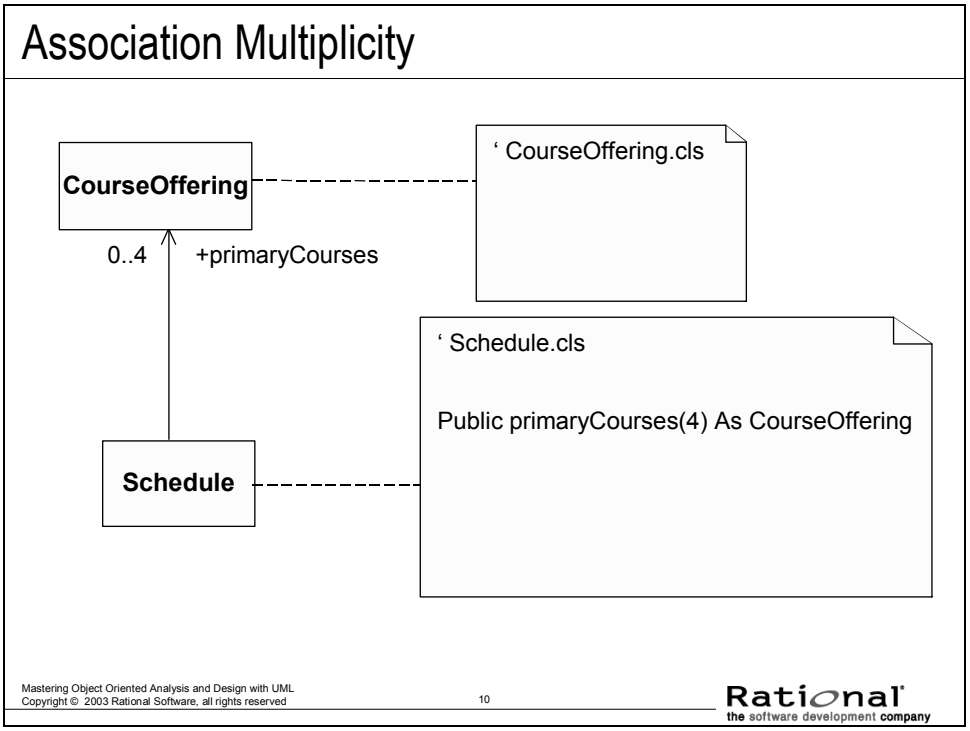
Rose defaults the access on roles, if not specified, to public

Association Roles



Roles on the end of the association can add clarity.

Association Multiplicity

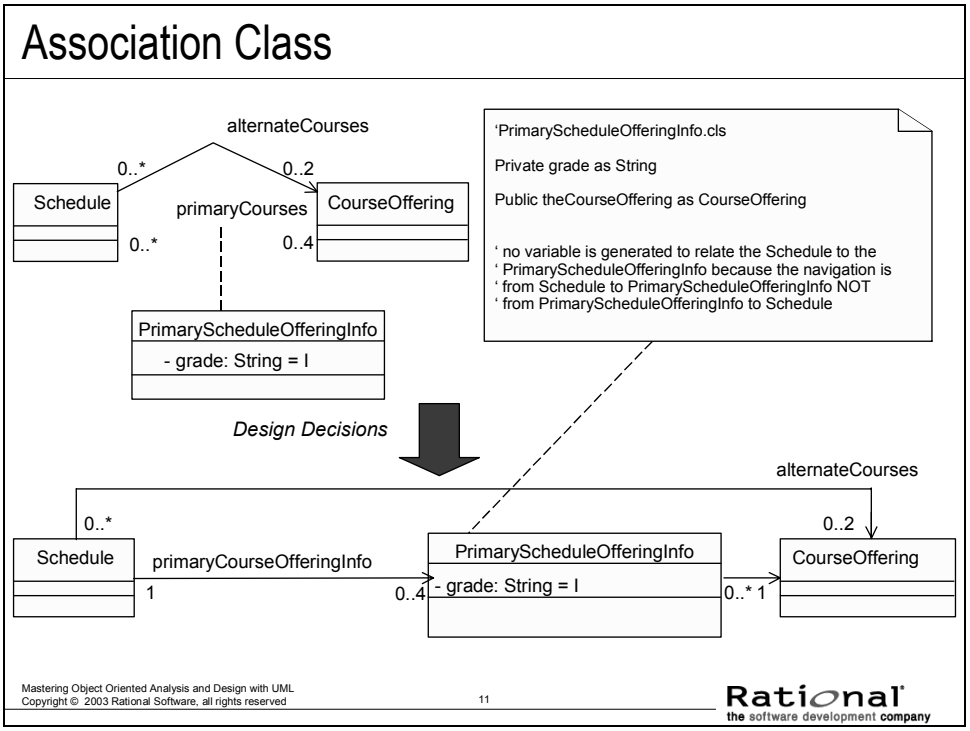


Multiplicity is the number of instances of one class in relation to the other.

The code that gets generated for multiplicity depends on what the values are. For multiplicity with * as the value, Visual Basic would typically use a Collection object because the limits are unbounded.

The example above has a bounded limit on the multiplicity (4), so Visual Basic can use and array.

Association Class

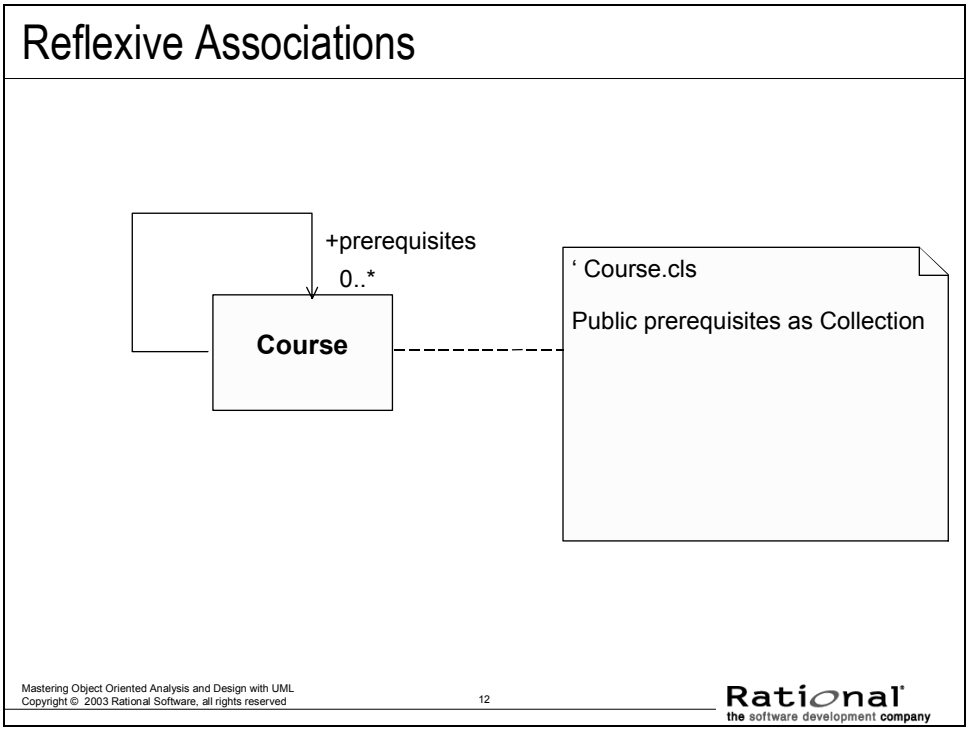


Remember, an association class is a class that is connected to an association. There is an instance of the association class for every instance of the relationship (e.g., for every link).

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

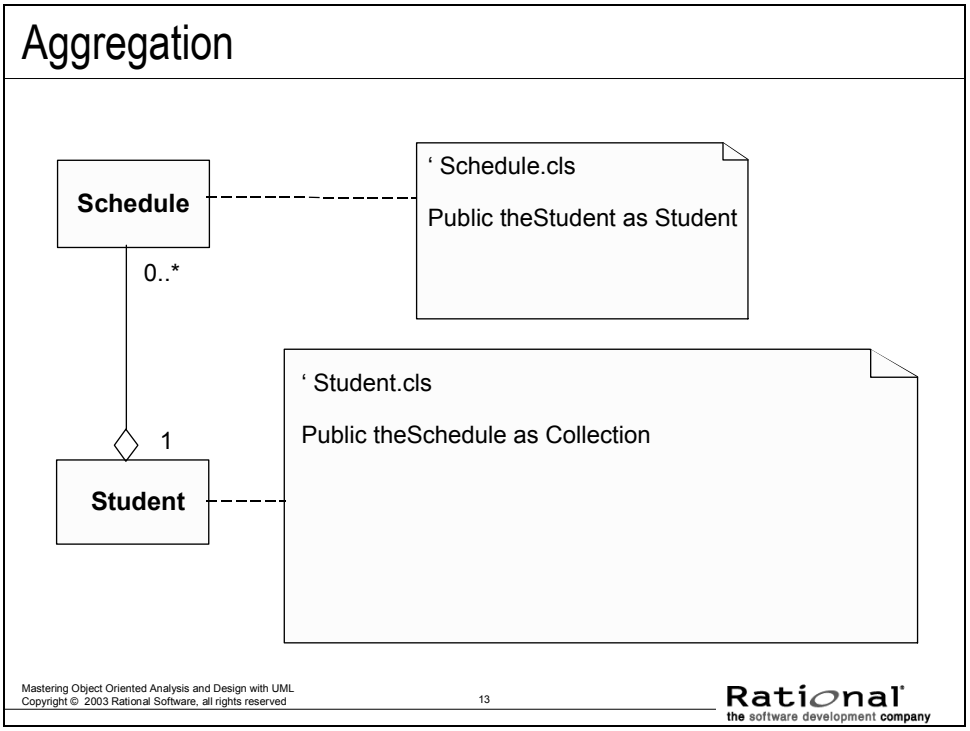
A subset of the class operations and attributes are shown above. For this example, we included a subset to demonstrate the UML construct we are emphasizing.

Reflexive Associations



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

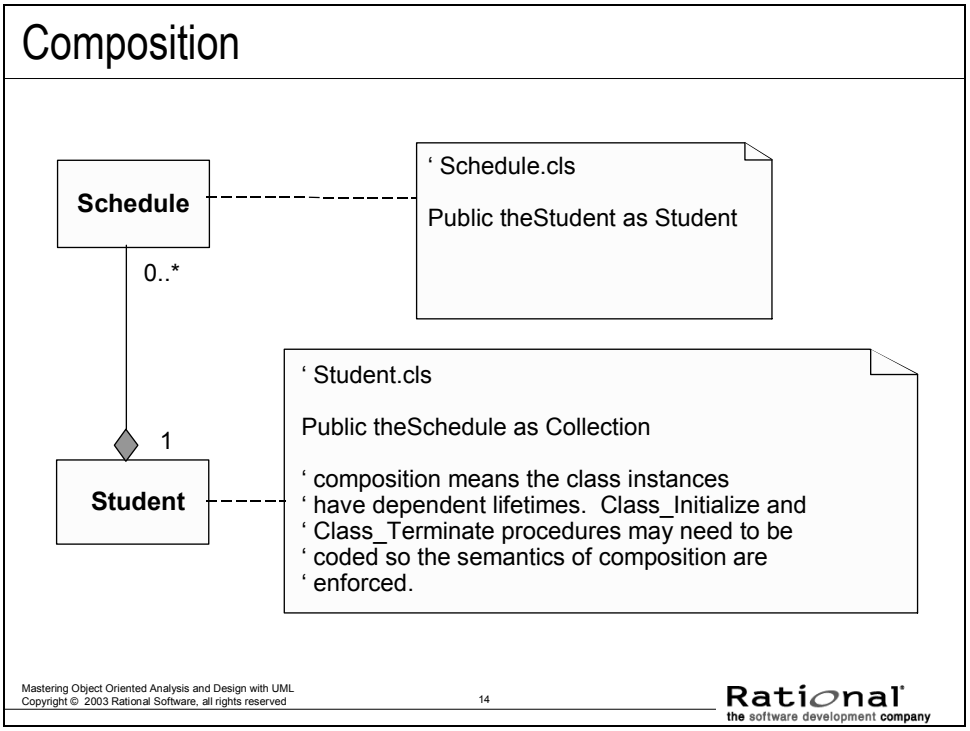
Aggregation



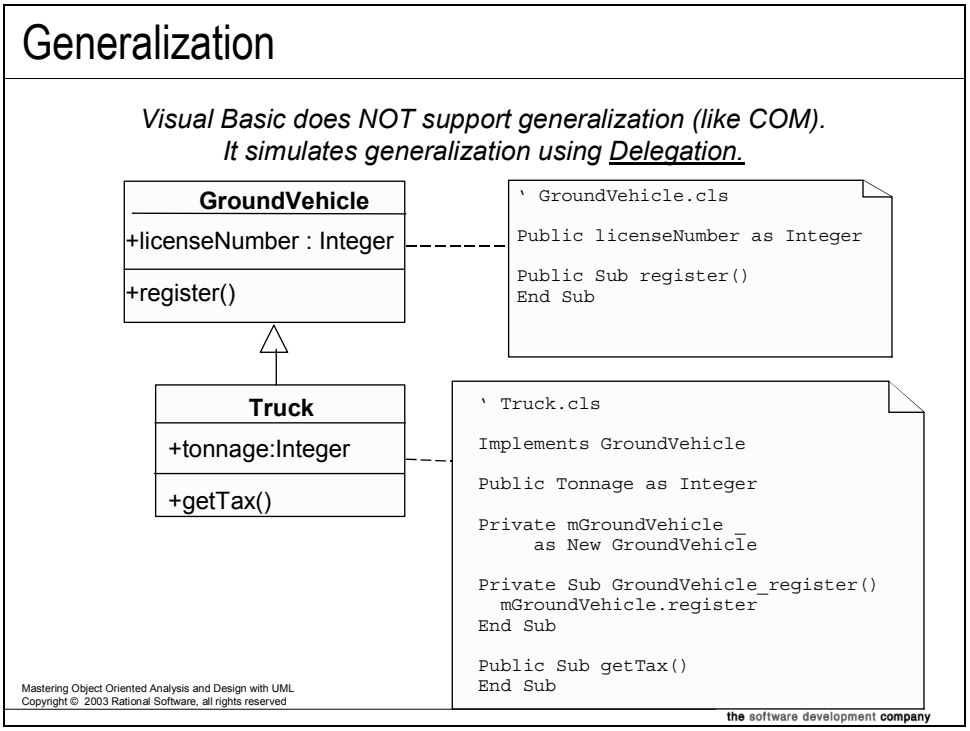
Collection is a reusable list class available in the VISUAL BASIC programming environment. It could be replaced by any other available list class.

VISUAL BASIC has no explicit construct for aggregation. In VISUAL BASIC, the code for aggregation looks the same as it does for “vanilla” association.

Composition



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.

Visual Basic has NO concept of generalization. Because VISUAL BASIC classes are based on COM technology, there is no inheritance. VISUAL BASIC can simulate something LIKE generalization via delegation. It works around the issue taking some concepts from Interfaces and other concepts from delegation, but all inferred by saying a class "implements" another class.

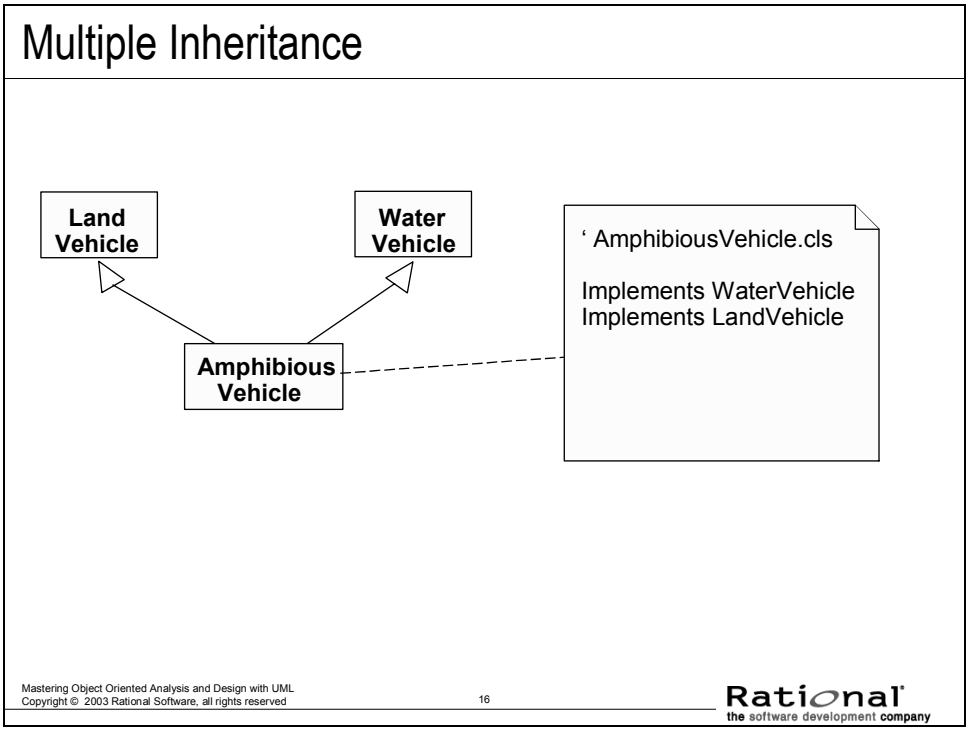
The code in the above example demonstrates VISUAL BASIC simulation of generalization.

If a subclass inherits from a super class:

1. Subclass objects can be created
2. The subclass object has two interfaces - the one it "inherited" from the superclass and the one it defines itself.

Generalization hierarchies in VISUAL BASIC can only be 2 levels deep -- a superclass and subclass. Subclasses cannot have more subclasses.

Multiple Inheritance



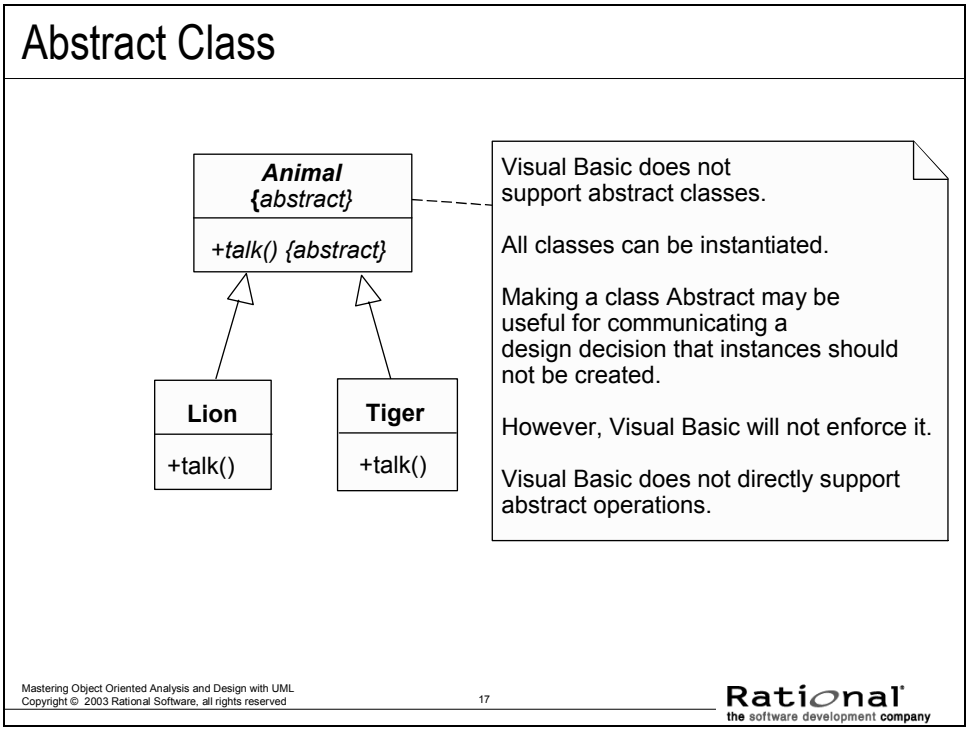
Visual Basic will support multiple inheritance. Semantically it means that the sub class implements the interfaces of the super classes.

See the preceding slide for limitation on VISUAL BASIC and generalization / inheritance.

Circular inheritance is supported

Inheritance depths of more than one level are not supported.

Abstract Class



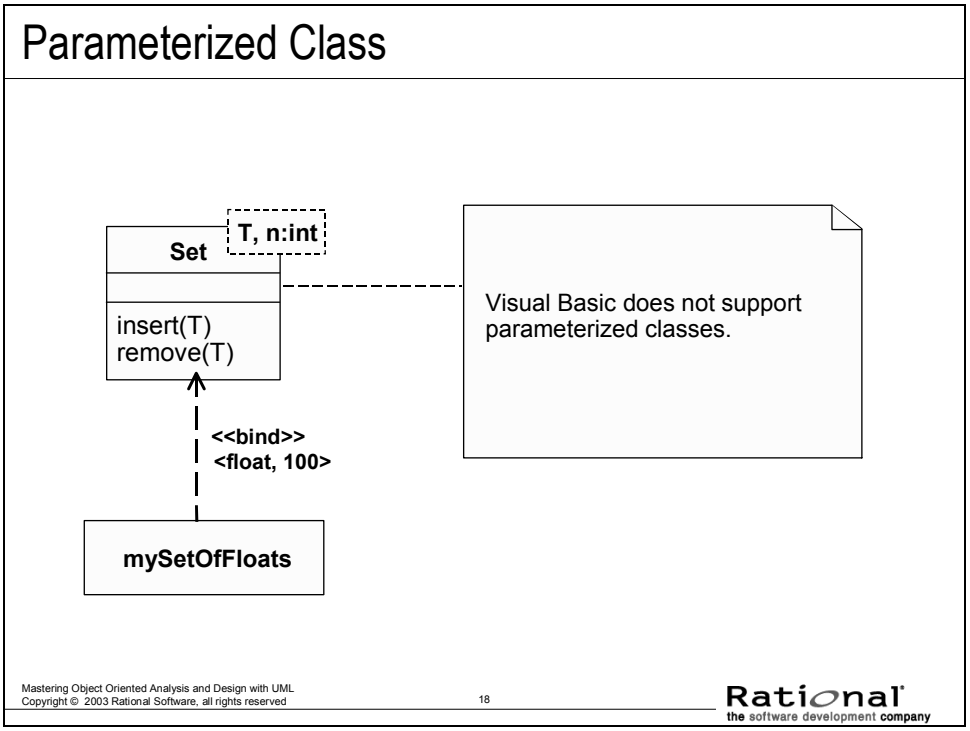
Remember, an abstract class is a class for which no instances are created.

All classes in VISUAL BASIC can be instantiated. Making a class abstract can be used to communicate intentions but there is not language enforcement for the intention.

VISUAL BASIC does not support abstract operations. All operations in VISUAL BASIC code have an implementation - even if the implementation doesn't do anything. You may choose to use an Abstract operation to indicate that the implementation for that operation should be empty.

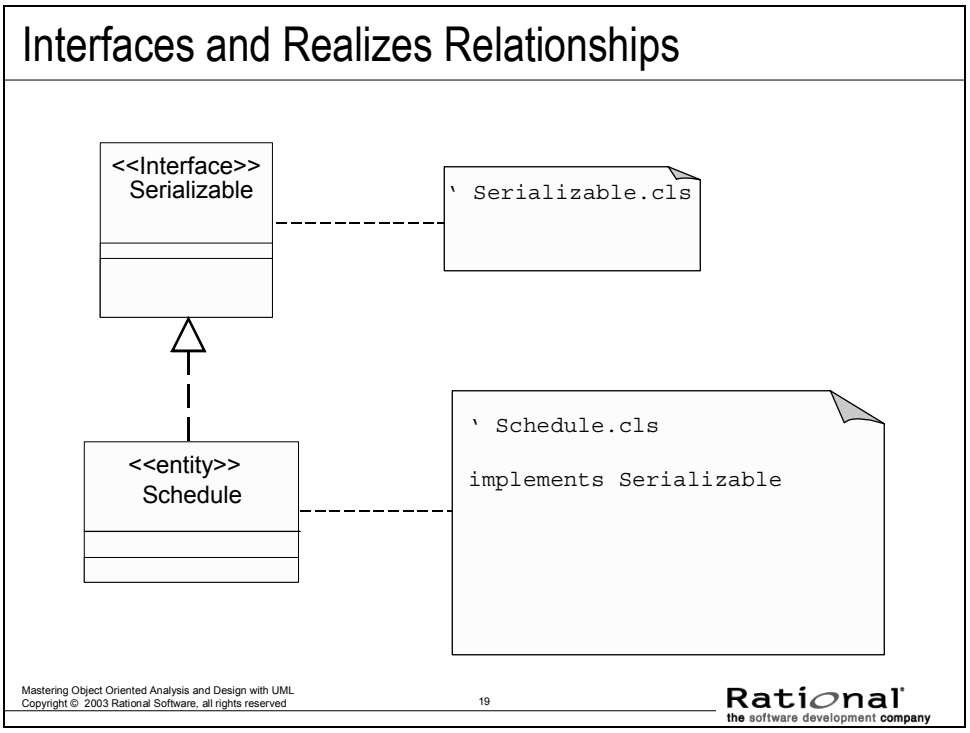
VISUAL BASIC does not support the semantics of Abstract classes. In VISUAL BASIC all classes are actually ActiveX automation servers and ActiveX may provide a mechanism that supports the "Abstract" class semantics. It would not be reflected however in the VISUAL BASIC code itself.

Parameterized Class



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

Interfaces and Realizes Relationships



Visual Basic mixes the UML concepts of Interfaces and Classes. If you have a Class, you also have an Interface with the same name.

Visual Basic does not directly support interfaces and the realizes relationship. If a realizes relationship is drawn between two Visual Basic classes (e.g. VISUAL BASIC Class Modules), it means that a given class will also support the same properties and methods defined in the class it is "realizing".

Visual Basic refers to this as a class module "implements" another class module. Implements is a Visual Basic keyword. For a code example of implements, see the previous Generalization slide.

<<Interface>> in the UML has very specific semantics. These semantics differ from those of how VISUAL BASIC implements it's version of interface inheritance. In particular - things you might consider interfaces in VISUAL BASIC are really class modules. Unlike UML Interfaces - VISUAL BASIC interfaces:

a) Can be instantiated (they are just class modules)

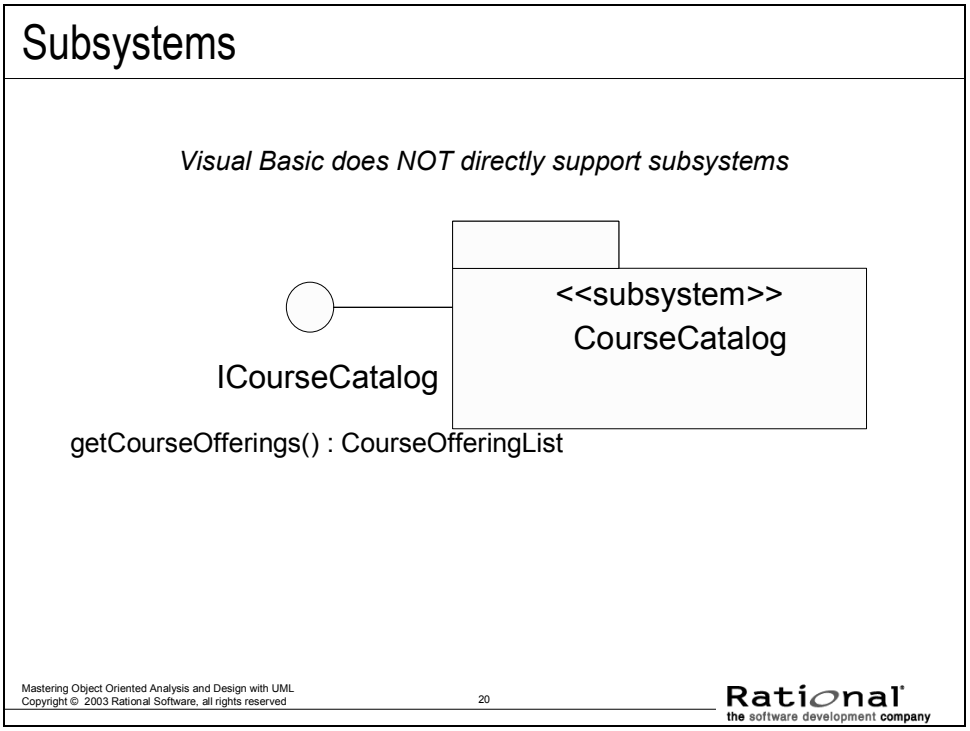
b) Provide an implementation for operations

c) May contain attributes.

This can be interpreted as a practical application of the UML for a particular language.

In the example above, the Schedule class supports two interfaces - "Schedule" and "Serializable".

Subsystems



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

Visual Basic does not directly support subsystems. A workaround is to map subsystems to Visual Basic Projects. The projects can be ActiveX DLLs or ActiveX EXEs. The Visual Basic class modules assigned to the subsystem are exposed for use by clients of the subsystem as COM interfaces.