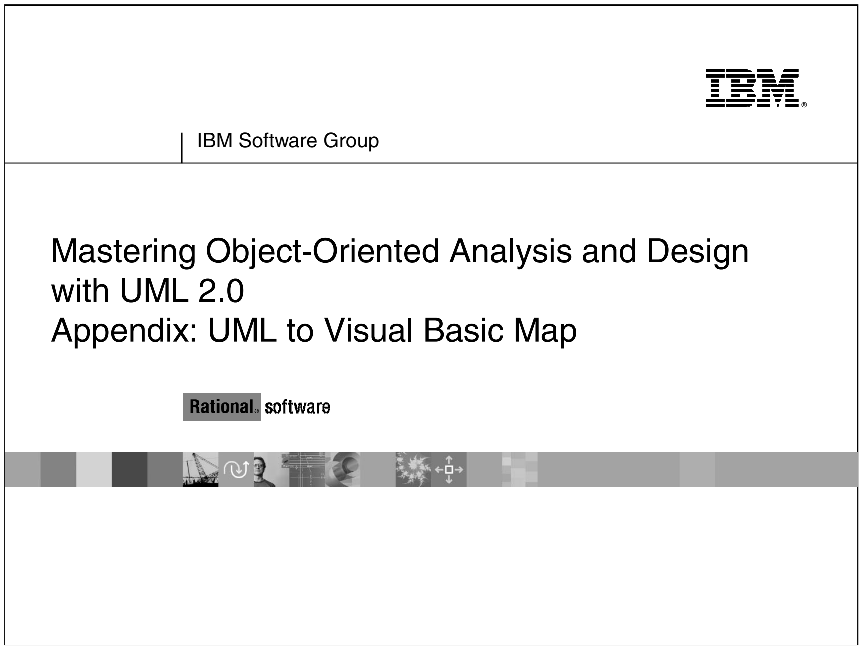
Instructor Notes:

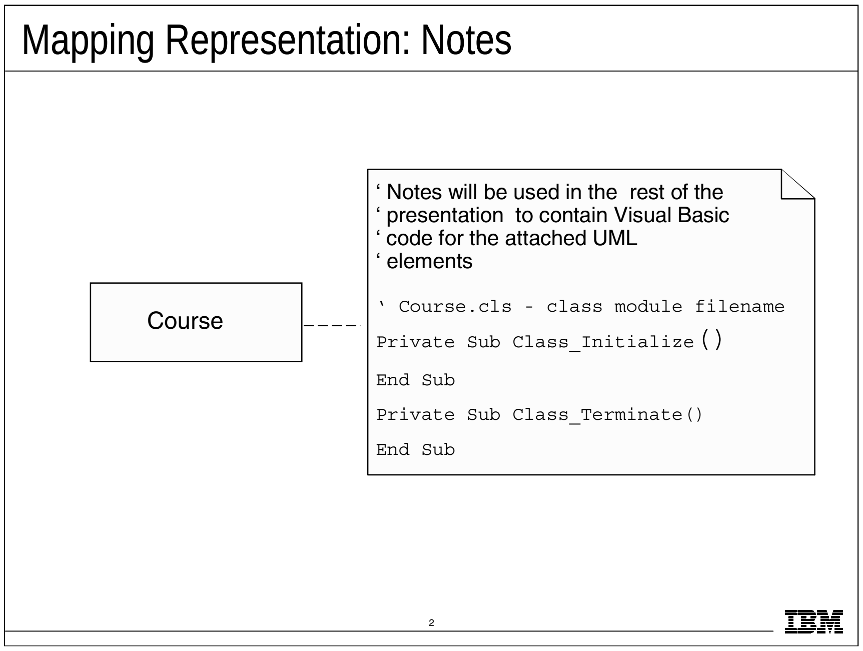


IBM Software Group

Mastering Object-Oriented Analysis and Design with UML 2.0
Appendix: UML to Visual Basic Map

Rational. software

Instructor Notes:



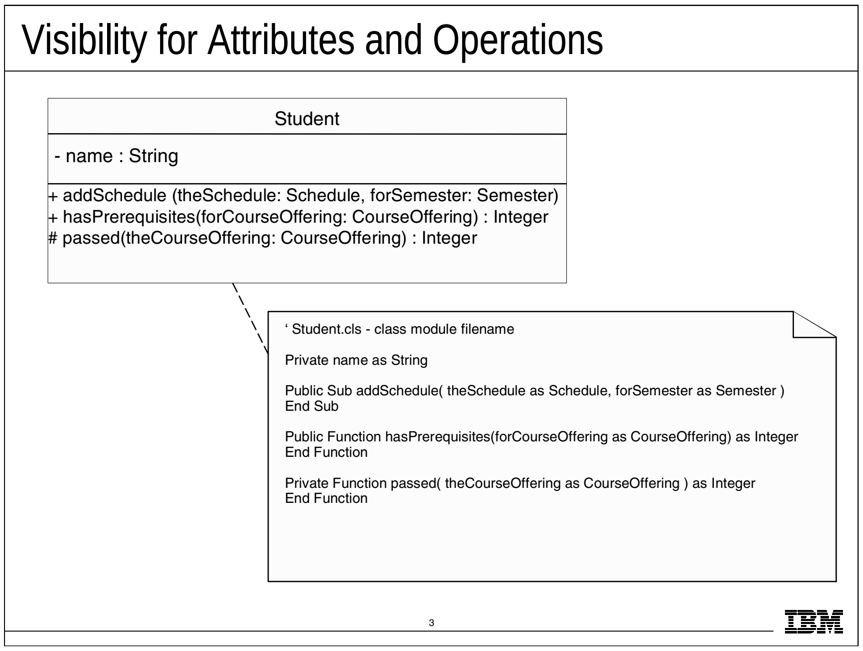
Mapping Representation: Notes

```
‘ Notes will be used in the rest of the
‘ presentation to contain Visual Basic
‘ code for the attached UML
‘ elements

‘ Course.cls - class module filename
Private Sub Class_Initialize()
End Sub
Private Sub Class_Terminate()
End Sub
```

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”. VB gives the code module a .CLS extension. The code module has a property that defines the name of the class in the class module.

Instructor Notes:



Visibility for Attributes and Operations

Student
- name : String
+ addSchedule (theSchedule: Schedule, forSemester: Semester)
+ hasPrerequisites(forCourseOffering: CourseOffering) : Integer
# passed(theCourseOffering: CourseOffering) : Integer

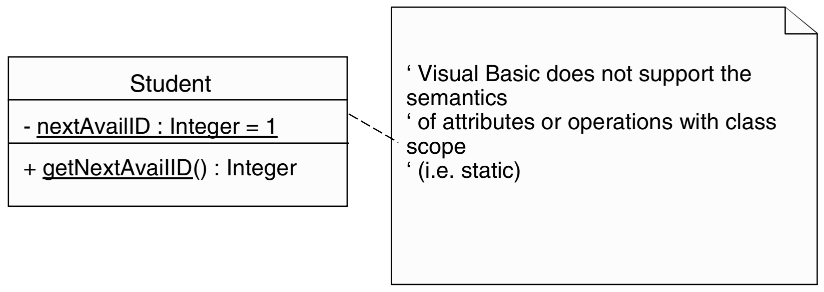
```
‘ Student.cls - class module filename

Private name as String

Public Sub addSchedule( theSchedule as Schedule, forSemester as Semester )
End Sub

Public Function hasPrerequisites(forCourseOffering as CourseOffering) as Integer
End Function

Private Function passed( theCourseOffering as CourseOffering ) as Integer
End Function
```

3

Visual Basic does not support protected access.

Instructor Notes:

Class Scope Attributes and Operations



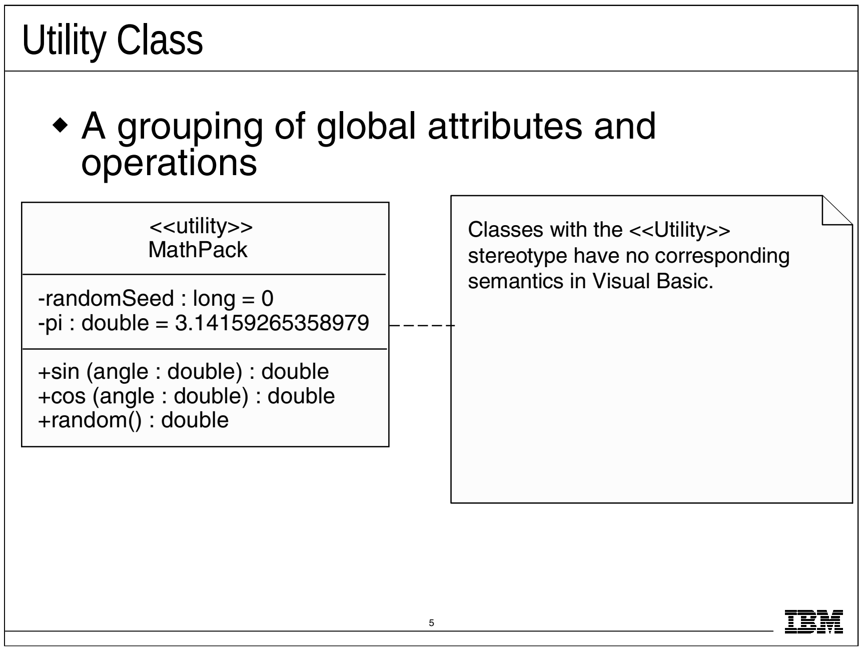
‘ Visual Basic does not support the semantics
‘ of attributes or operations with class scope
‘ (i.e. static)

4

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.

Instructor Notes:



Utility Class

- A grouping of global attributes and operations

<<utility>>
MathPack

-randomSeed : long = 0
-pi : double = 3.14159265358979

+sin (angle : double) : double
+cos (angle : double) : double
+random() : double

Classes with the <<Utility>> stereotype have no corresponding semantics in Visual Basic.

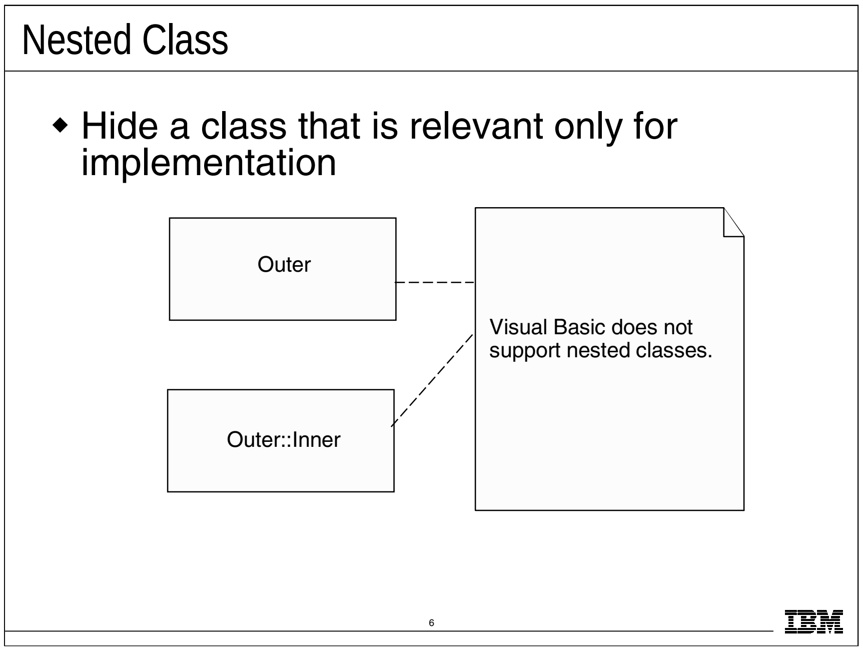
5

IBM

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.

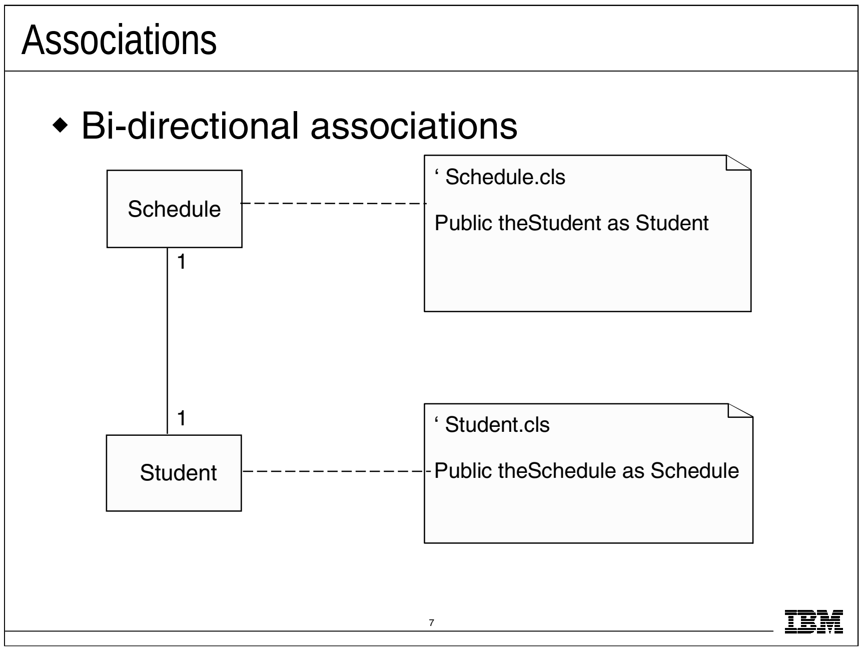
Instructor Notes:



Nested Class

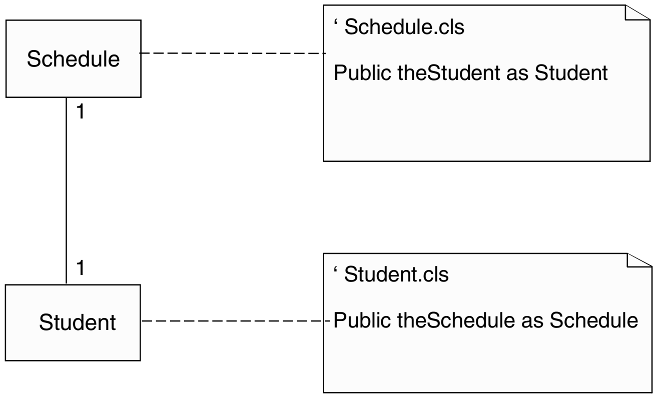
- Hide a class that is relevant only for implementation

Instructor Notes:



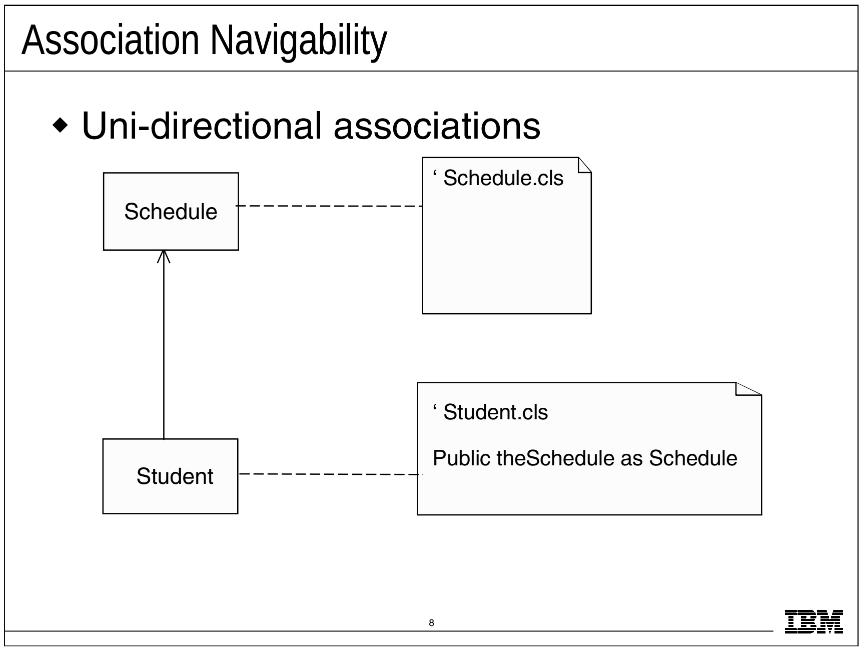
Associations

- Bi-directional associations



7

Instructor Notes:



Association Navigability

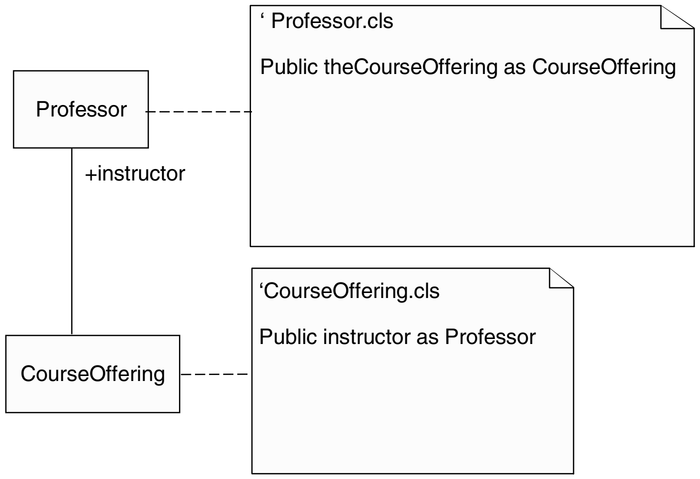
- Uni-directional associations

```
' Schedule.cls
```

```
' Student.cls

Public theSchedule as Schedule
```

Instructor Notes:

Association Roles

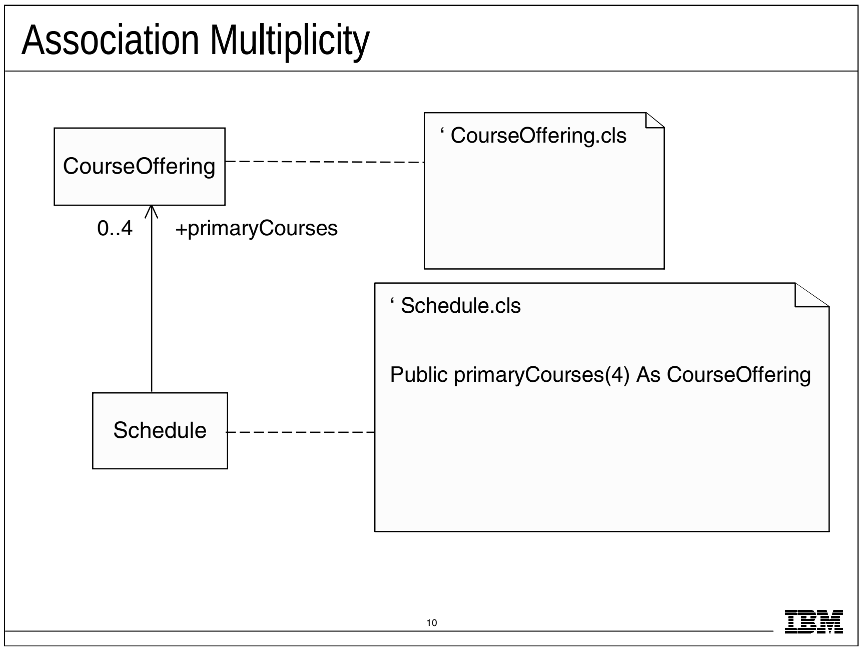


```
' Professor.cls

Public theCourseOffering as CourseOffering
```

```
'CourseOffering.cls

Public instructor as Professor
```

Roles on the end of the association can add clarity.

Instructor Notes:



Association Multiplicity

```
' Schedule.cls

Public primaryCourses(4) As CourseOffering
```

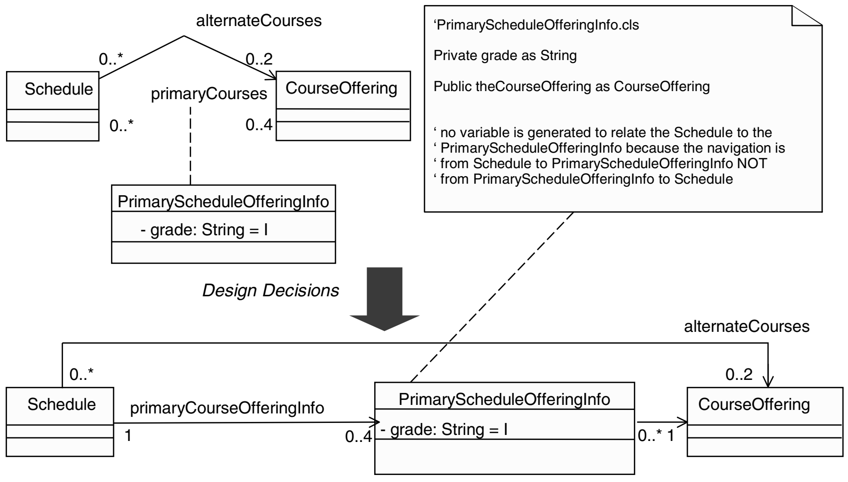
Multiplicity is the number of instances of one class in relation to the other.

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.

Instructor Notes:

Association Class



```
'PrimaryScheduleOfferingInfo.cls

Private grade as String

Public theCourseOffering as CourseOffering


' no variable is generated to relate the Schedule to the
' PrimaryScheduleOfferingInfo because the navigation is
' from Schedule to PrimaryScheduleOfferingInfo NOT
' from PrimaryScheduleOfferingInfo to Schedule
```

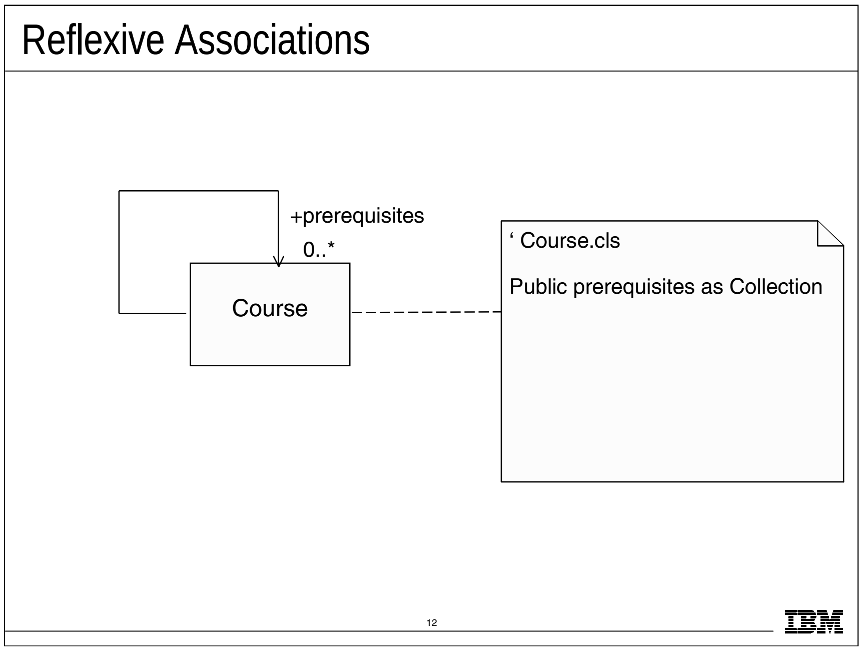
11

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).

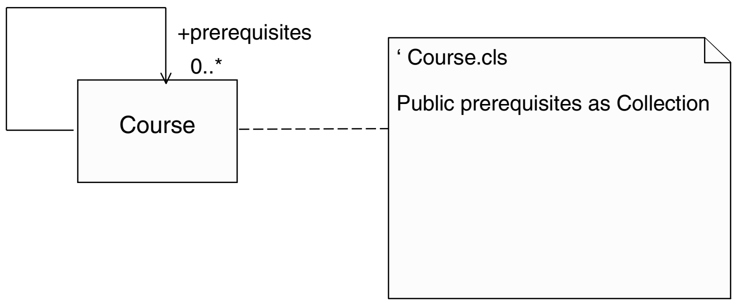
During design, some decisions are made regarding navigation between the involved classes.

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.

Instructor Notes:



Reflexive Associations



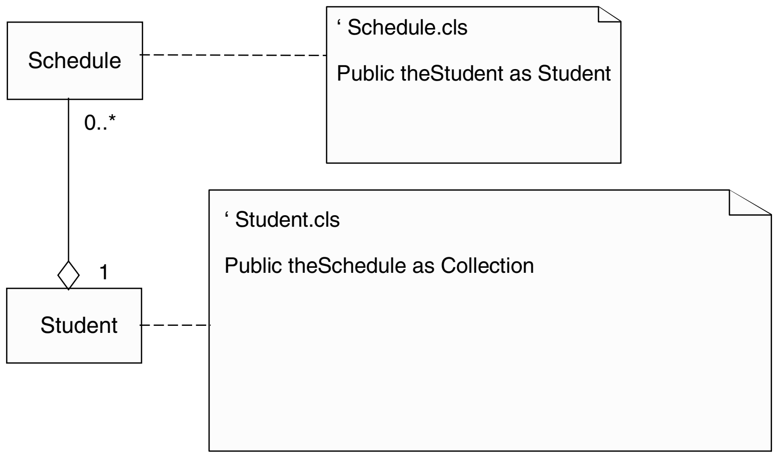
‘ Course.cls

Public prerequisites as Collection

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

Instructor Notes:

Aggregation



```
' Schedule.cls

Public theStudent as Student
```

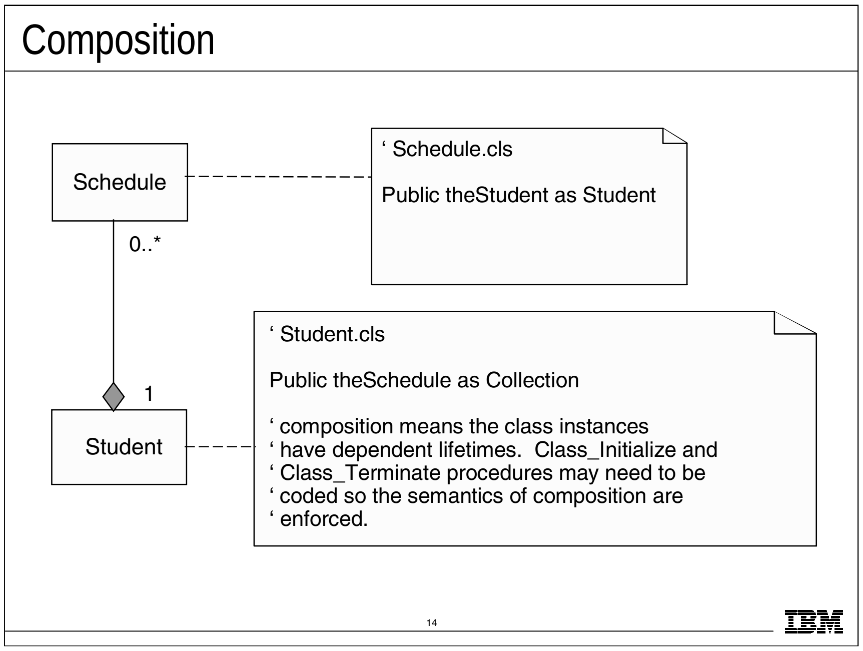
```
' Student.cls

Public theSchedule as Collection
```

13

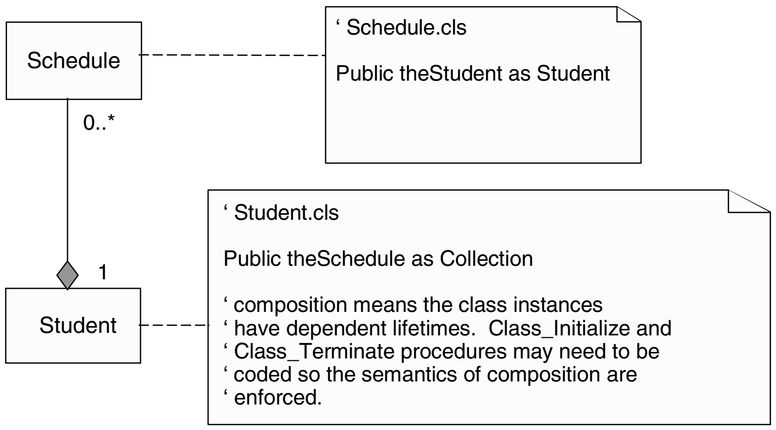
Collection is a reusable list class available in the VB programming environment. It could be replaced by any other available list class.

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

Instructor Notes:



Composition



‘ Schedule.cls

Public theStudent as Student

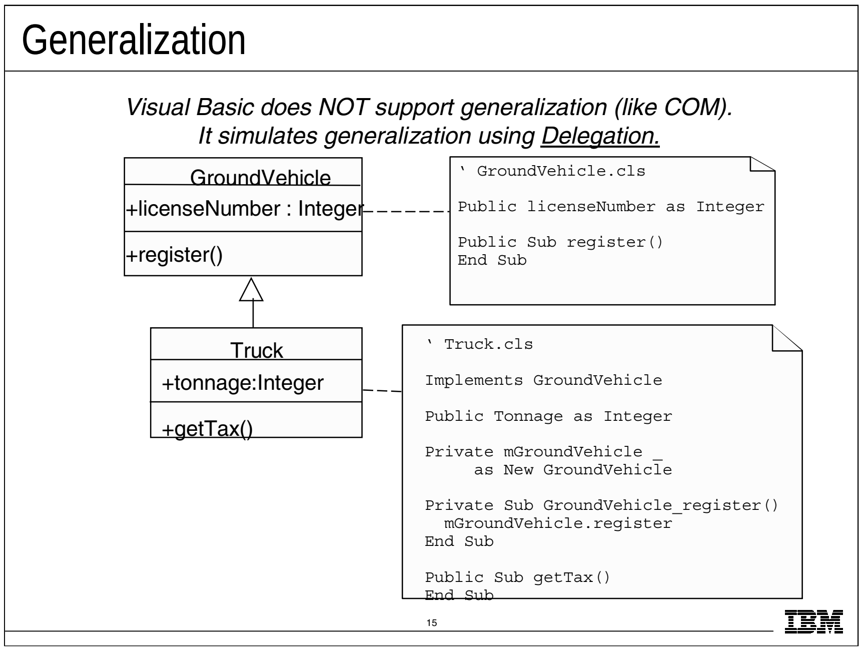
‘ Student.cls

Public theSchedule as Collection

‘ composition means the class instances
‘ have dependent lifetimes. Class_Initialize and
‘ Class_Terminate procedures may need to be
‘ coded so the semantics of composition are
‘ enforced.

14

Instructor Notes:



Generalization

Visual Basic does NOT support generalization (like COM).
It simulates generalization using Delegation.

```
' GroundVehicle.cls

Public licenseNumber as Integer

Public Sub register()
End Sub
```

```
' Truck.cls

Implements GroundVehicle

Public Tonnage as Integer

Private mGroundVehicle _
     as New GroundVehicle

Private Sub GroundVehicle_register()
  mGroundVehicle.register
End Sub

Public Sub getTax()
End Sub
```

15

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 VB classes are based on COM technology, there is no inheritance. VB 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 VB simulation of generalization.

If a subclass inherits from a super class:

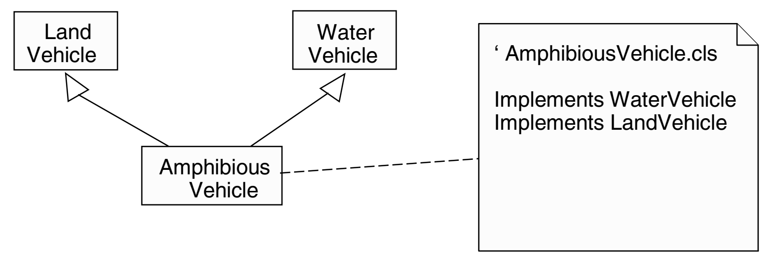
1. Subclass objects can be created

2. The subclass object has two interfaces - the one it "inherited" from the superclass and the one it defines itself.

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

Instructor Notes:

Multiple Inheritance



16

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

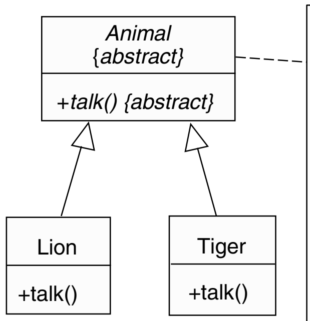
See the preceding slide for limitation on VB and generalization / inheritance.

Circular inheritance is supported

Inheritance depths of more than one level are not supported.

Instructor Notes:

Abstract Class



Visual Basic does not support abstract classes.

All classes can be instantiated.

Making a class Abstract may be useful for communicating a design decision that instances should not be created.

However, Visual Basic will not enforce it.

Visual Basic does not directly support abstract operations.

17

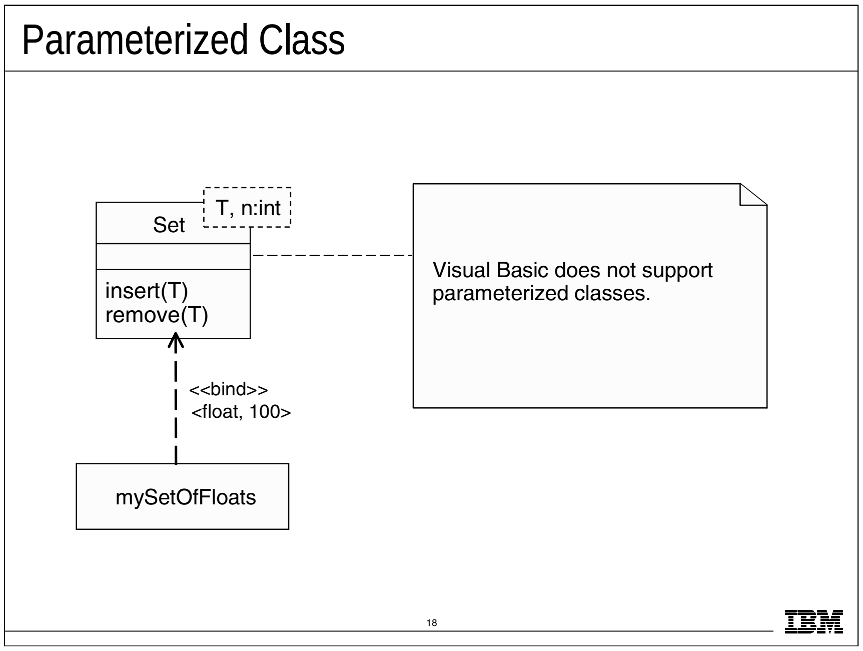
Remember, an abstract class is a class for which no instances are created.

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

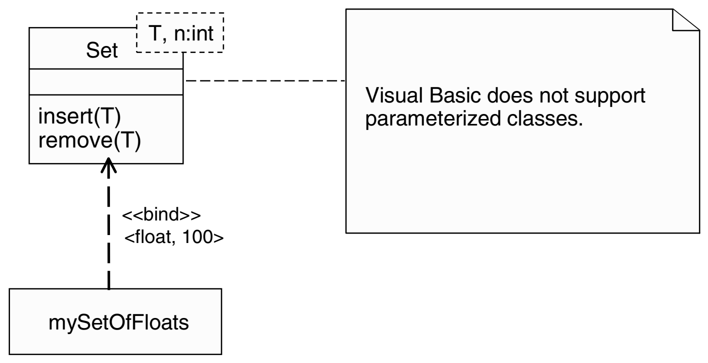
VB does not support abstract operations. All operations in VB code have an implementation - even if the implementation doesn't do anything. You may choose to use an Abstract operation to indicate that the implementation for that operation should be empty.

VB does not support the semantics of Abstract classes. In VB 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 VB code itself.

Instructor Notes:



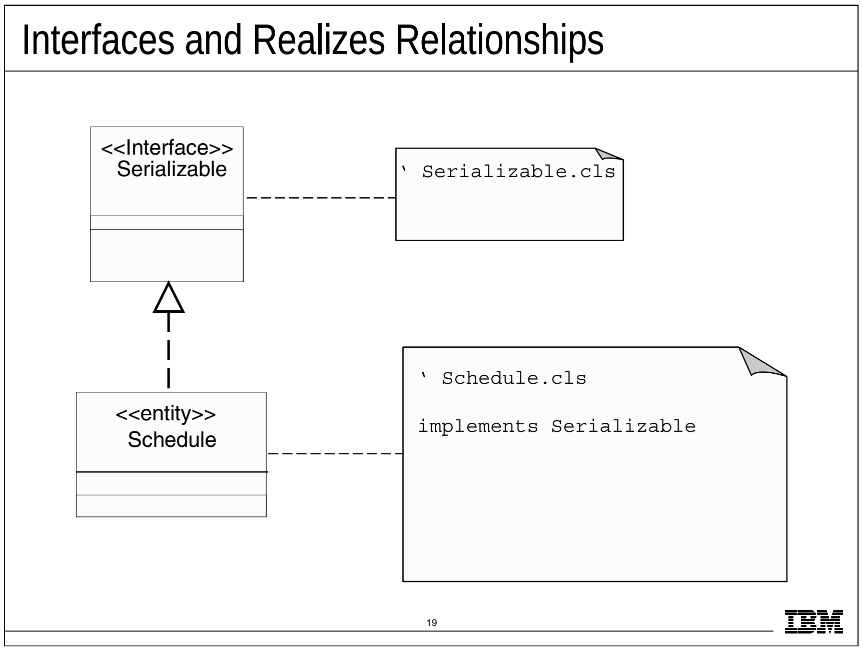
Parameterized Class



18

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

Instructor Notes:



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 (for example, VB 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 VB implements it's version of interface inheritance. In particular - things you might consider interfaces in VB are really class modules. Unlike UML Interfaces - VB interfaces:

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

b) Provide an implementation for operations

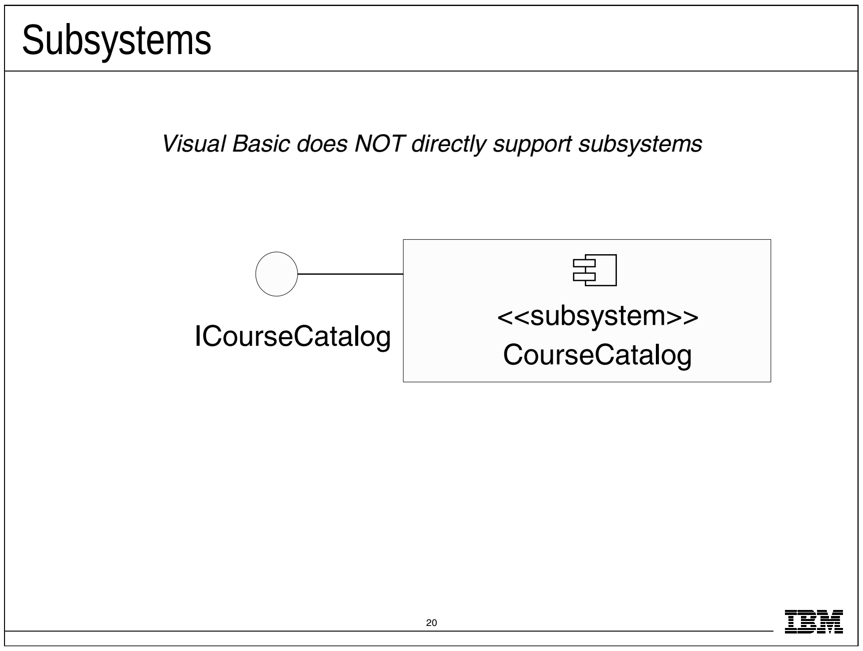
c) May contain attributes.

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".

Instructor Notes:

In Visual Basic, all classes are actually ActiveX automation servers. They are made available as ActiveX automation interfaces from either executables (<<EXE>>) or DLLs (<<DLL>>).



Subsystems

Visual Basic does NOT directly support subsystems

ICourseCatalog
<<subsystem>>
CourseCatalog
20

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.