University of Science, VNU-HCM
Faculty of Information Technology

Review: Object-Oriented Programming

Assoc. Prof. TRAN Minh Triet
Department of Software Engineering

TRUONG DAI HOC KHOA HOC TU NHIEN
KHTN
TP. HO CHI MINH

Software Analysis and Design

References

Object-Oriented Analysis and Design with Applications
by Grady Booch et.al., Addison-Wesley, 2007
(chapter 3)

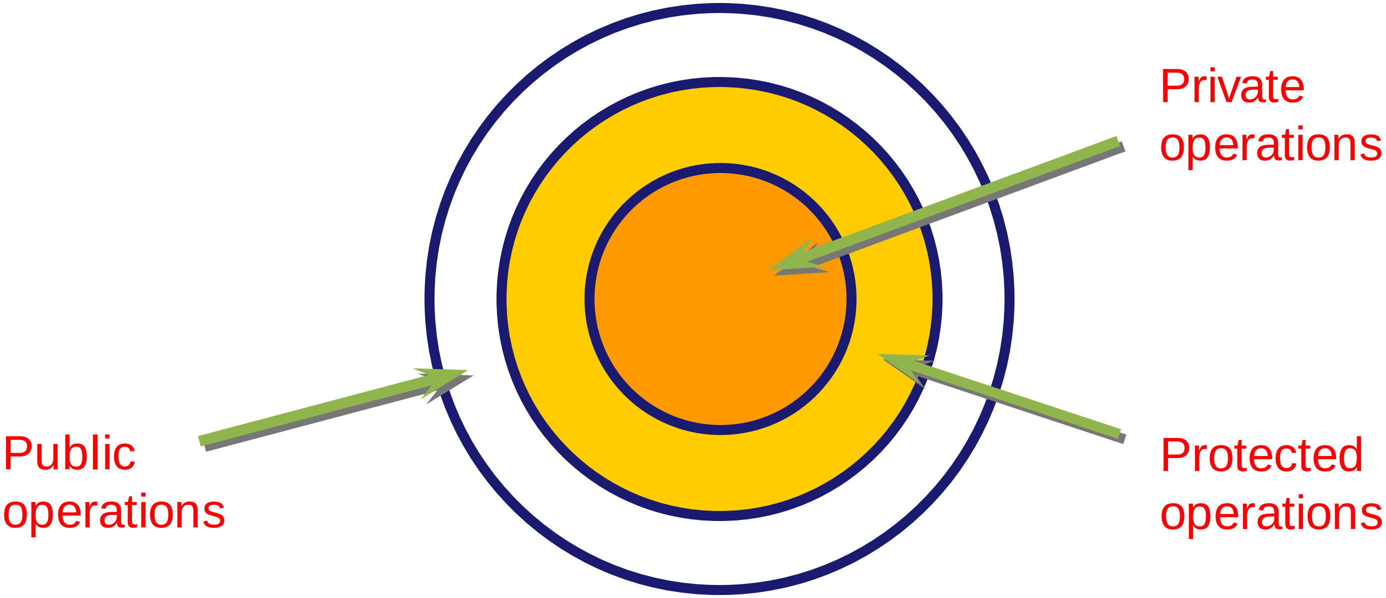
Object-Oriented Analysis and Design
Understanding System Development with UML 2.0
By Mike O'Docherty, John Wiley & Sons, 2005
(chapter 2, chapter 7 – section 7.4)

Course "Mastering Object-Oriented Analysis and Design with UML 2.0"
by IBM Software Group

Course "Object-Oriented Analysis & Design with UML"
by Tran Hanh Nhi, Tran Minh Triet, and Nguyen Van Khiet
Faculty of Information Technology, University of Science, 2008

Operation Visibility

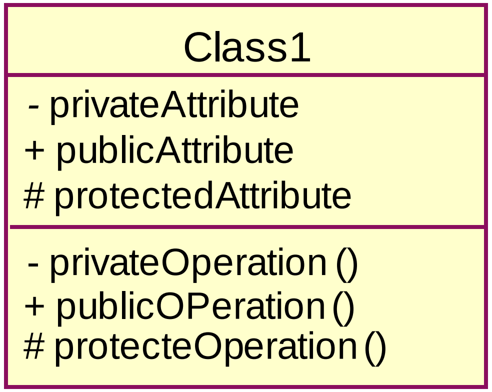
- Visibility is used to enforce encapsulation
- May be public, protected, or private



How Is Visibility Noted?

- The following symbols are used to specify export control:

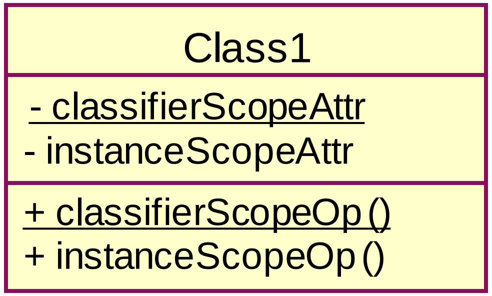
+	Public access
#	Protected access
-	Private access



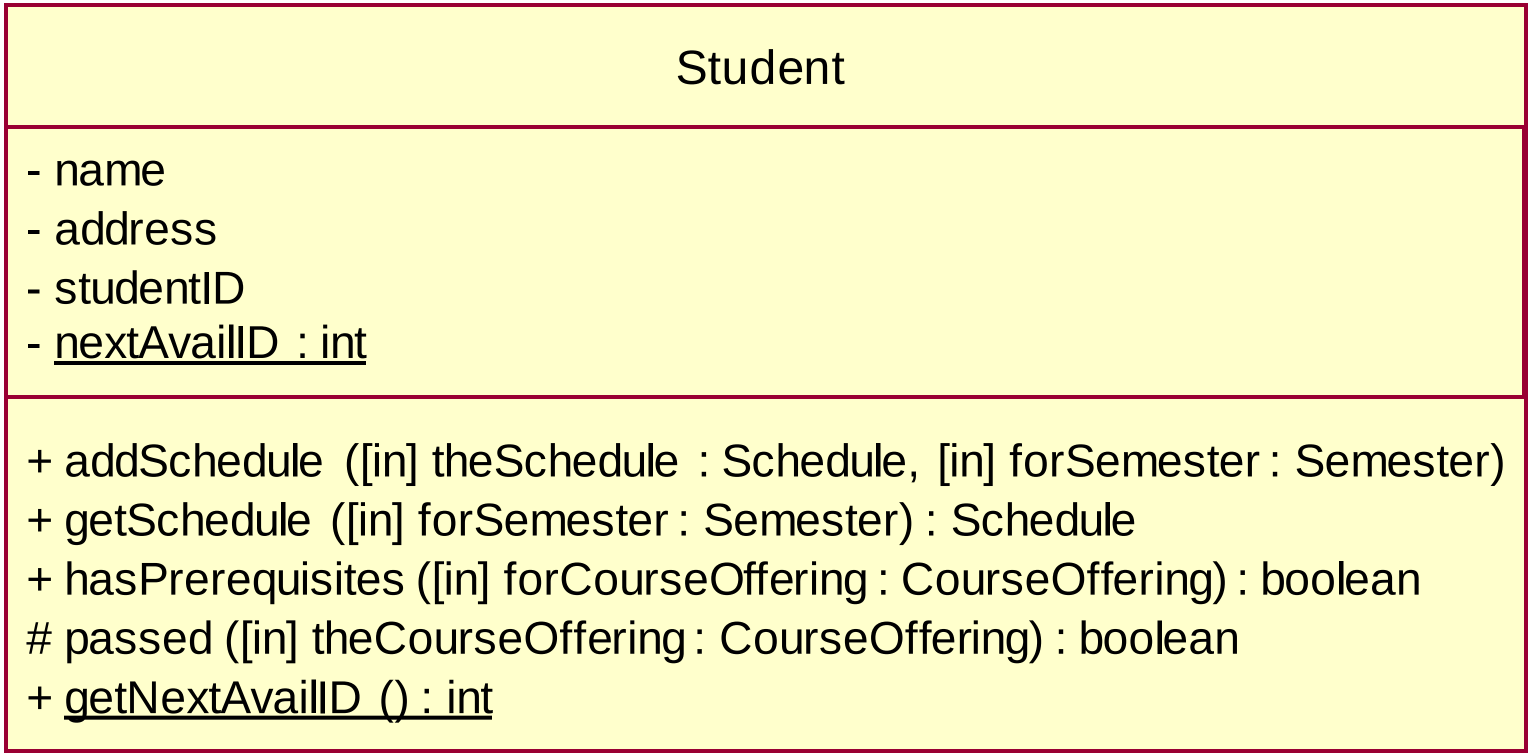
```
Class1
- privateAttribute
+ publicAttribute
# protectedAttribute
- privateOperation ()
+ publicOPeration ()
# protecteOperation ()
```

Scope

- Determines number of instances of the attribute/operation
  - Instance: one instance for each class instance
  - Classifier: one instance for all class instances
- Classifier scope is denoted by underlining the attribute/operation name



Example: Scope

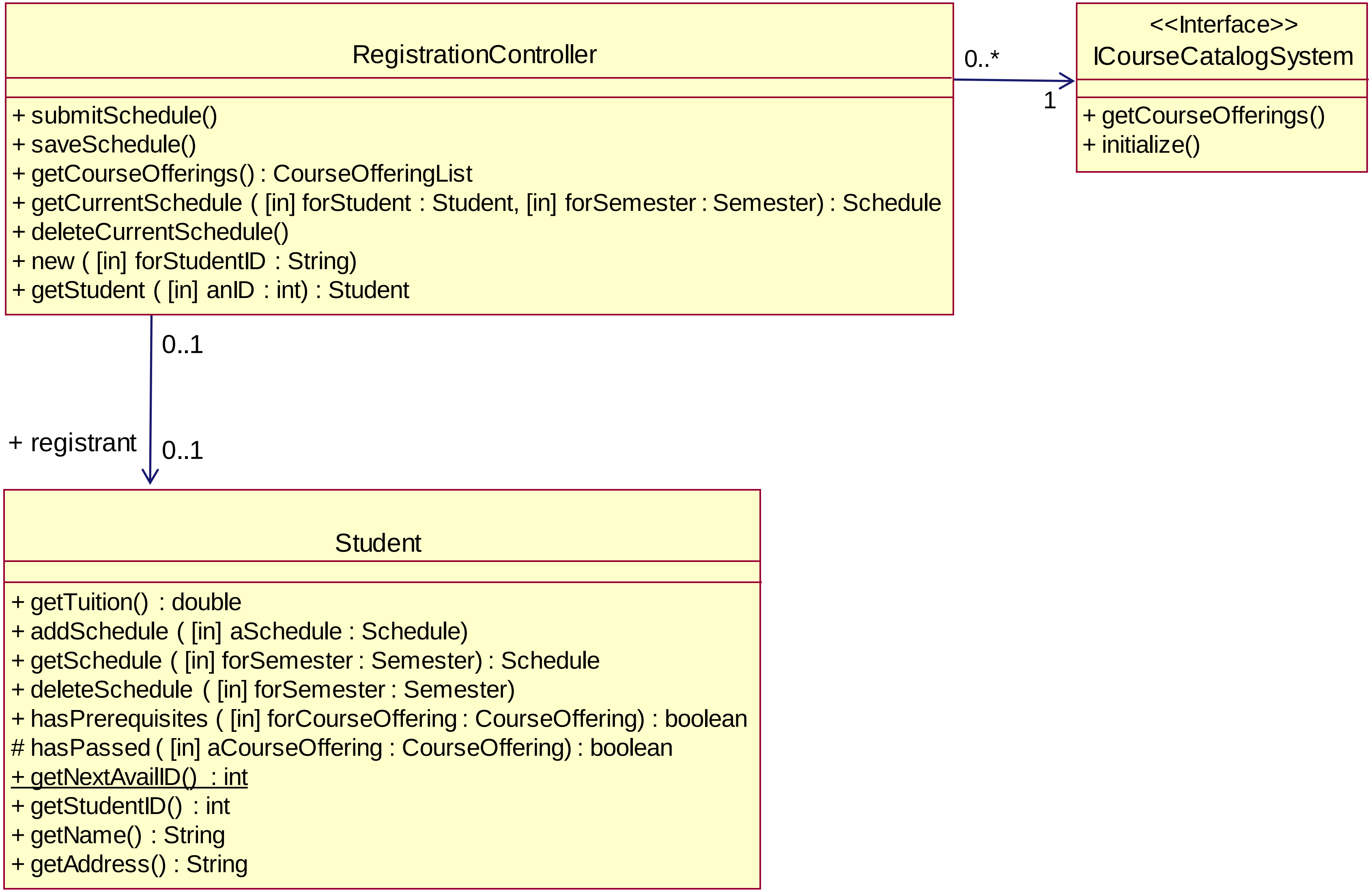


Student

- name
- address
- studentID
- nextAvailID : int

+ addSchedule ([in] theSchedule : Schedule, [in] forSemester : Semester)
+ getSchedule ([in] forSemester : Semester) : Schedule
+ hasPrerequisites ([in] forCourseOffering : CourseOffering) : boolean
# passed ([in] theCourseOffering : CourseOffering) : boolean
+ getNextAvailID () : int

Example: Define Operations



RegistrationController

+ submitSchedule()
+ saveSchedule()
+ getCourseOfferings() : CourseOfferingList
+ getCurrentSchedule ([in] forStudent : Student, [in] forSemester : Semester) : Schedule
+ deleteCurrentSchedule()
+ new ([in] forStudentID : String)
+ getStudent ([in] anID : int) : Student

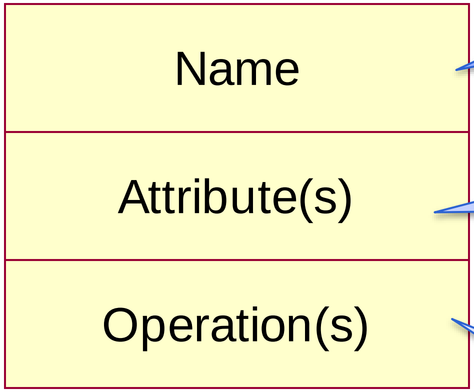
<<Interface>>
ICourseCatalogSystem

+ getCourseOfferings()
+ initialize()

Student

+ getTuition() : double
+ addSchedule ([in] aSchedule : Schedule)
+ getSchedule ([in] forSemester : Semester) : Schedule
+ deleteSchedule ([in] forSemester : Semester)
+ hasPrerequisites ([in] forCourseOffering : CourseOffering) : boolean
# hasPassed ([in] aCourseOffering : CourseOffering) : boolean
+ getNextAvailID() : int
+ getStudentID() : int
+ getName() : String
+ getAddress() : String

More Notations…



Name

Attribute(s)

Operation(s)

Regular: a regular class
Italic: an abstract class/an interface
Underlined: an object (not a class)

Regular: a regular attribute
Italic: N/A
Underlined: a static attribute

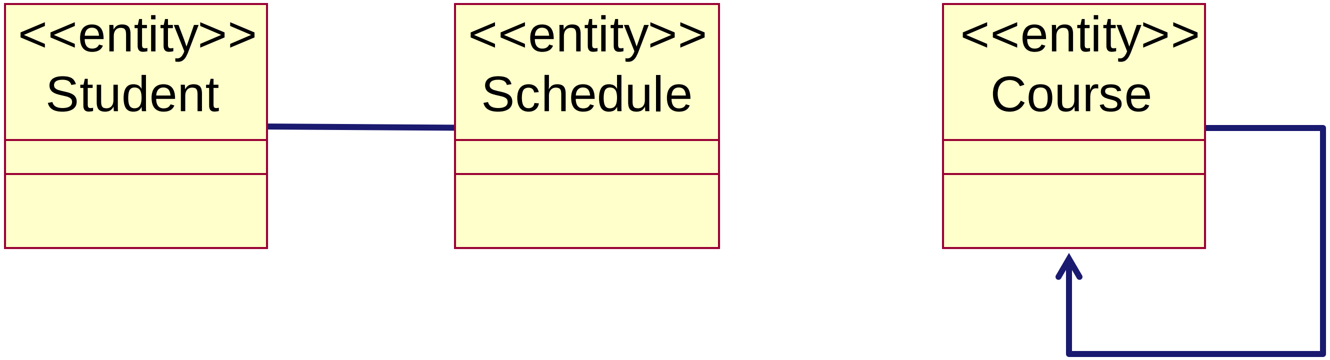
Regular: a regular operation
Italic: a virtual/override operation
Underlined: a static operation

Define Associations

- Purpose
  - Refine remaining associations
- Things to look for :
  - Association vs. Aggregation
  - Aggregation vs. Composition
  - Attribute vs. Association
  - Navigability
  - Association class design
  - Multiplicity design

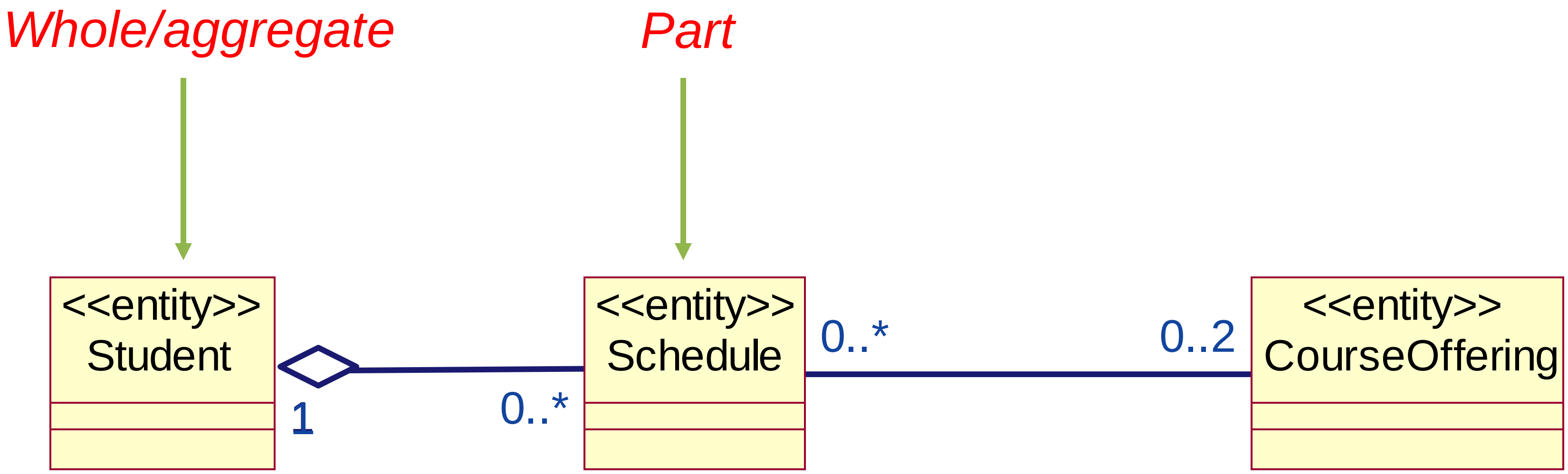
What Is an Association?

- The semantic relationship between two or more classifiers that specifies connections among their instances
- A structural relationship, specifying that objects of one thing are connected to objects of another



What Is Aggregation?

- ❖ A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts



Whole/aggregate

Part

<<entity>>
Student

1

0..*

<<entity>>
Schedule

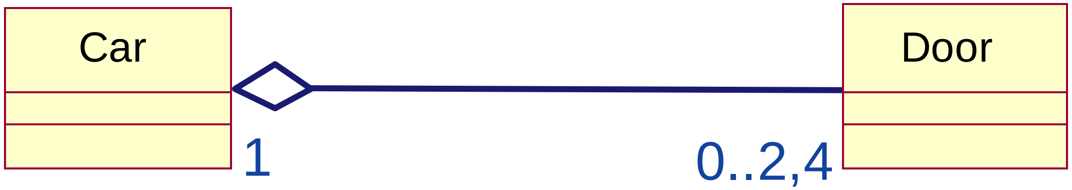
0..*

0..2

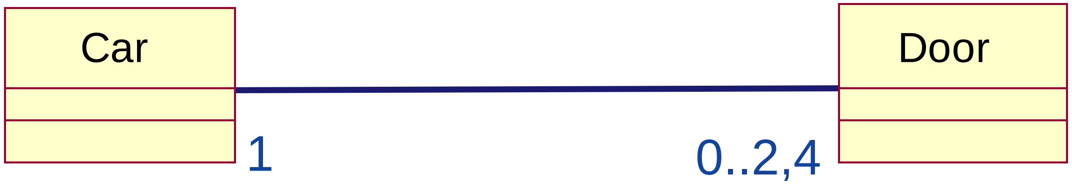
<<entity>>
CourseOffering

Association or Aggregation?

- If two objects are tightly bound by a whole-part relationship
  - The relationship is an aggregation.



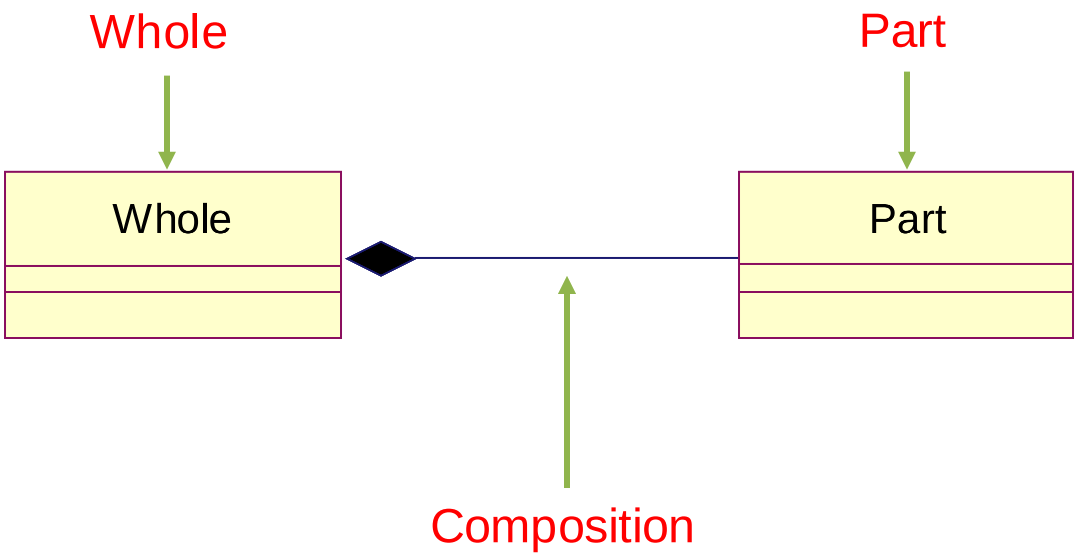
- If two objects are usually considered as independent, although they are often linked
  - The relationship is an association.



When in doubt, use association.

What Is Composition?

- A form of aggregation with strong ownership and coincident lifetimes
  - The parts cannot survive the whole/aggregate



Whole

Part

Whole

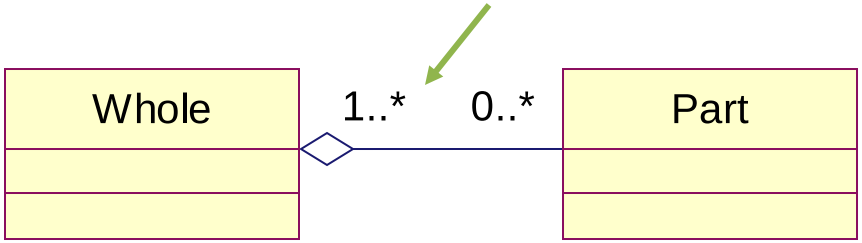
Part

Composition

Aggregation: Shared vs. Non-shared

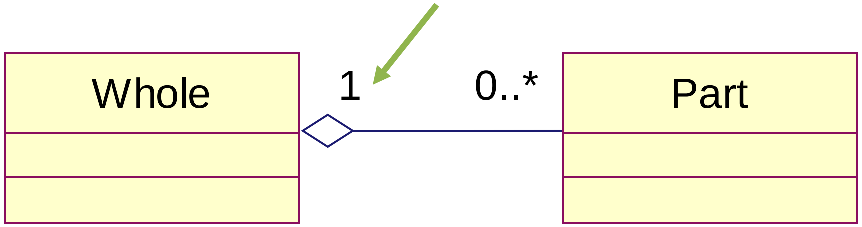
- Shared Aggregation

Multiplicity > 1

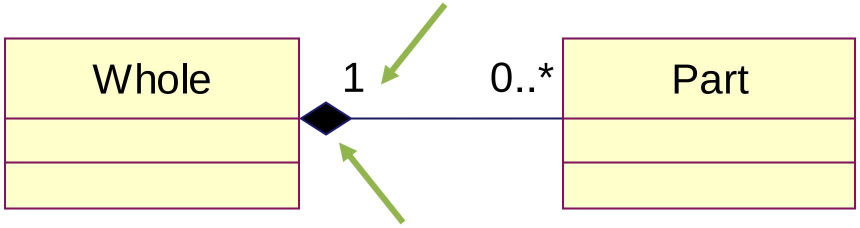


- Non-shared Aggregation

Multiplicity = 1



Multiplicity = 1

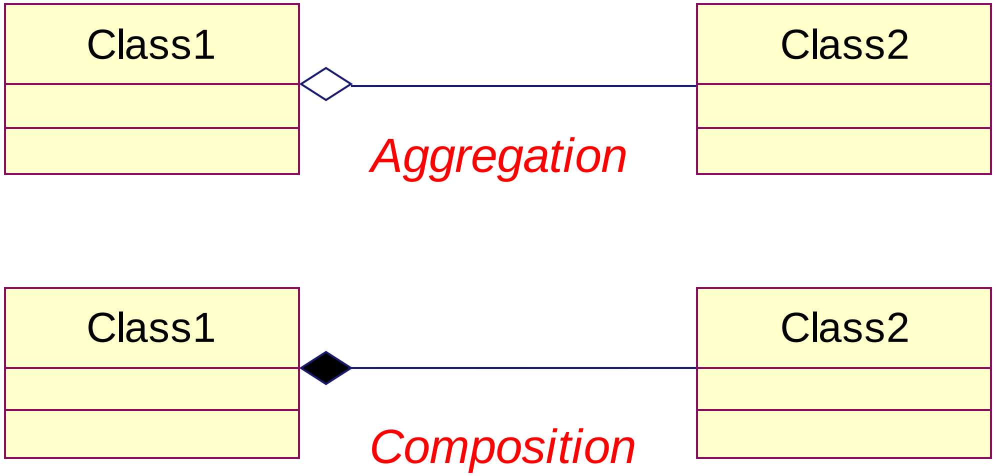


Composition

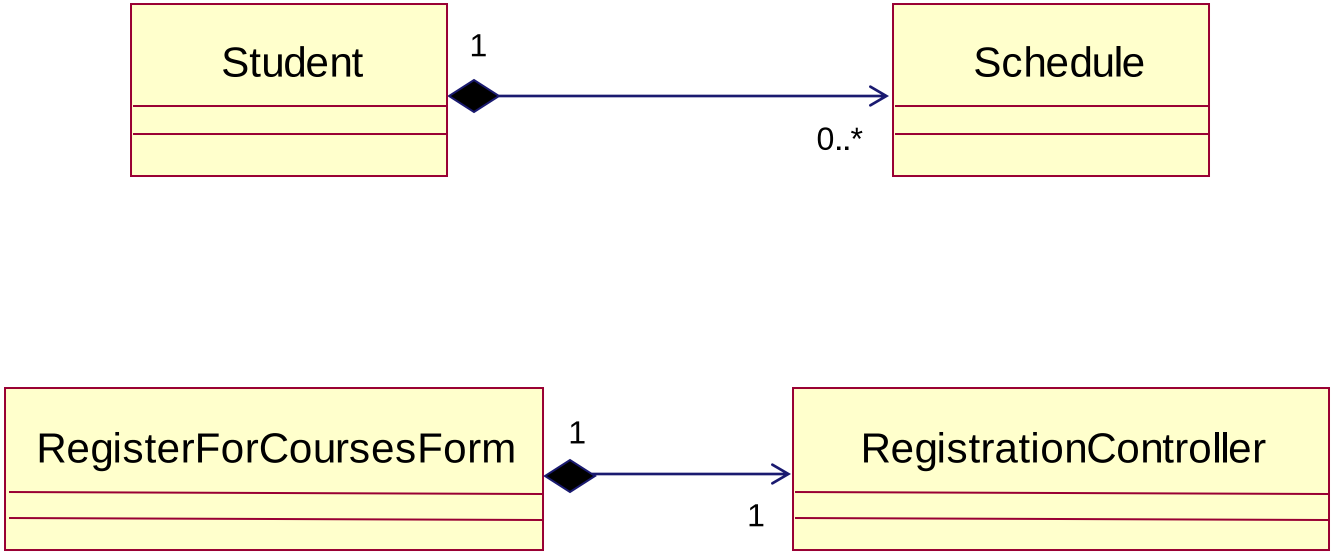
By definition, composition is non-shared aggregation.

Aggregation or Composition?

- Consideration
  - Lifetimes of Class1 and Class2



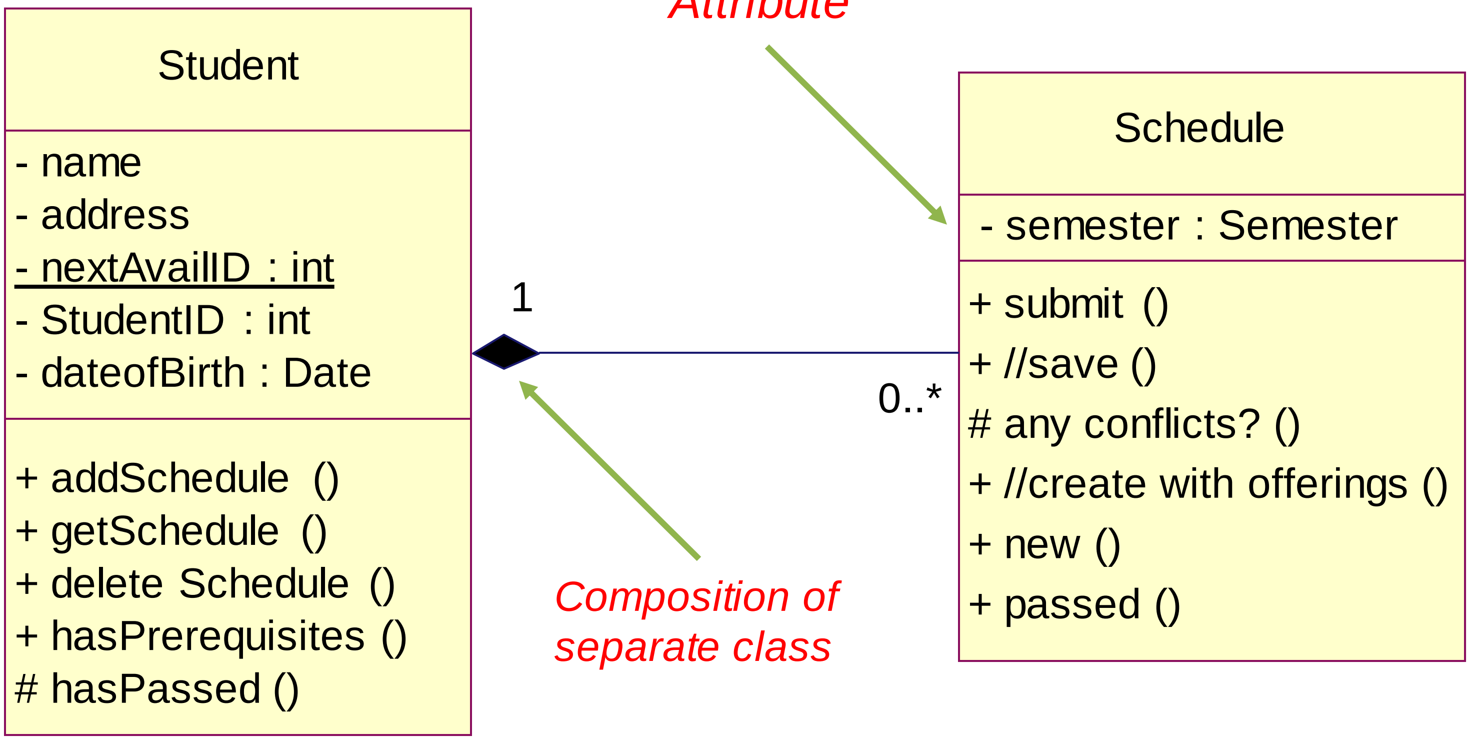
Example: Composition



Attributes vs. Composition

- Use composition when
  - Properties need independent identities
  - Multiple classes have the same properties
  - Properties have a complex structure and properties of their own
  - Properties have complex behavior of their own
  - Properties have relationships of their own
- Otherwise use attributes

Example: Attributes vs. Composition



Student

- name
- address
- nextAvailID : int
- StudentID : int
- dateofBirth : Date

+ addSchedule ()
+ getSchedule ()
+ delete Schedule ()
+ hasPrerequisites ()
# hasPassed ()

Schedule

- semester : Semester

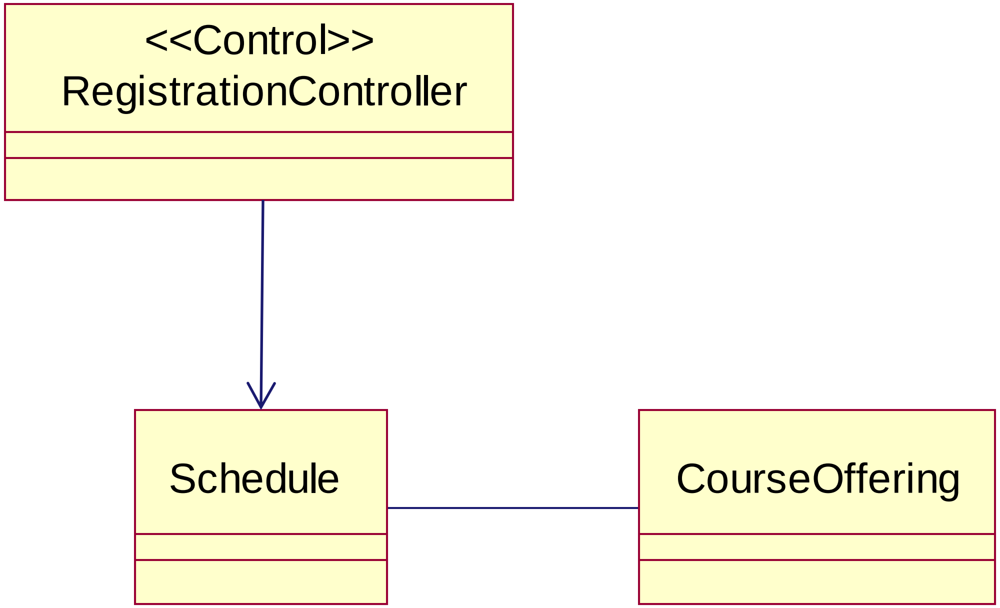
+ submit ()
+ //save ()
# any conflicts? ()
+ //create with offerings ()
+ new ()
+ passed ()

Attribute

Composition of separate class

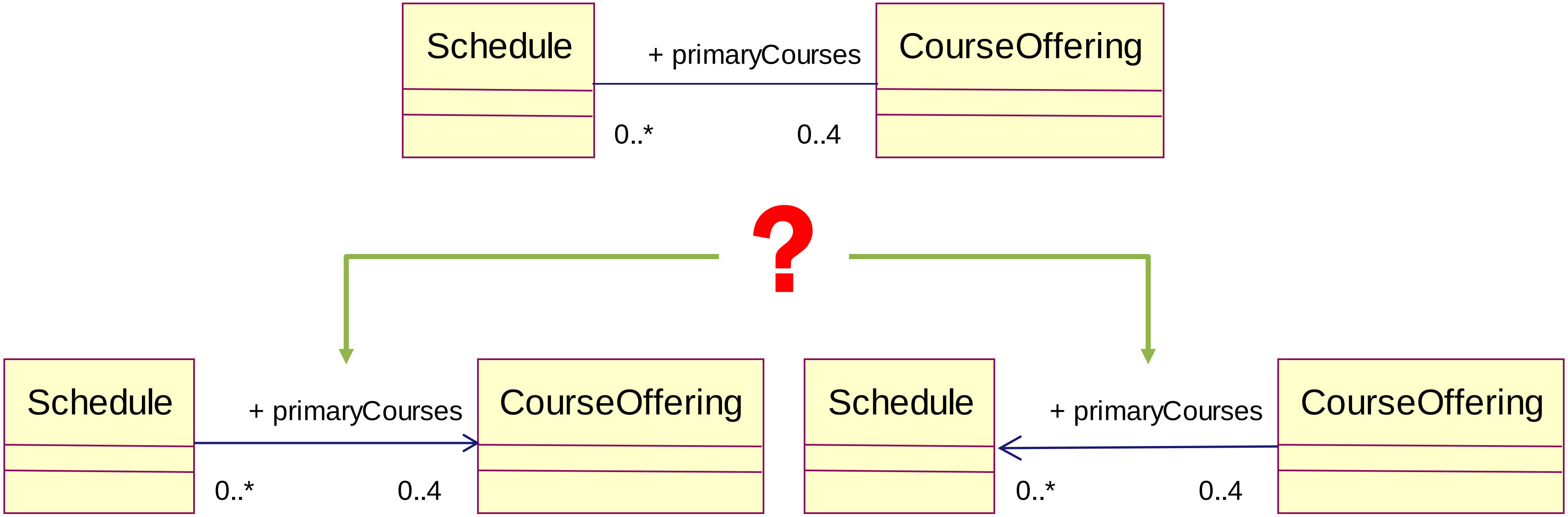
Review: What Is Navigability?

- Indicates that it is possible to navigate from an associating class to the target class using the association



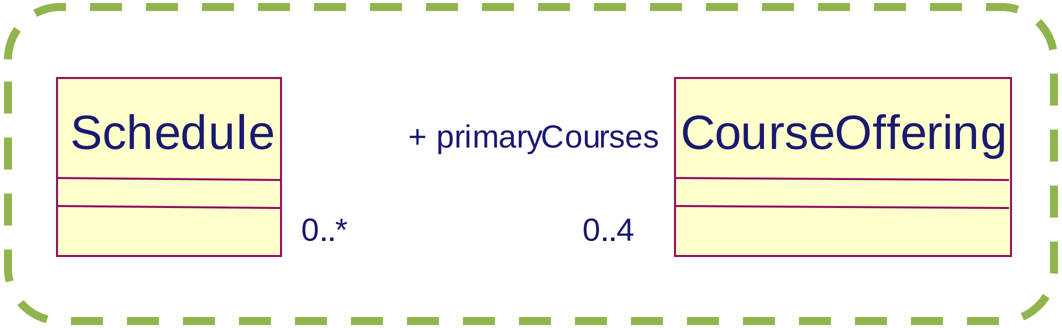
Navigability: Which Directions Are Needed?

- Explore interaction diagrams
- Even when both directions seem required, one may work
  - Navigability in one direction is infrequent
  - Number of instances of one class is small

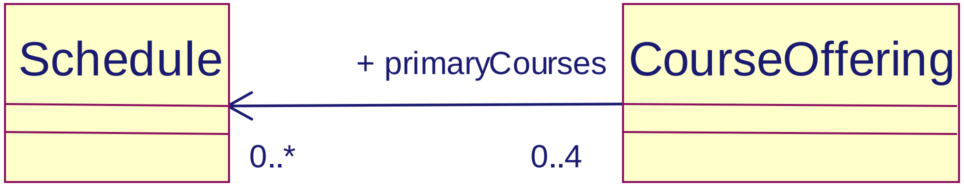


Example: Navigability Refinement

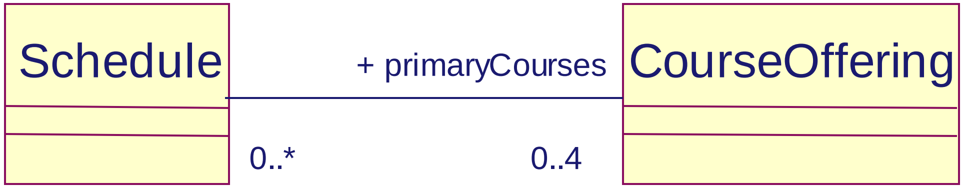
- Total number of Schedules is small, or
- Never need a list of the Schedules on which the CourseOffering appears



- Total number of CourseOfferings is small, or
- Never need a list of CourseOfferings on a Schedule

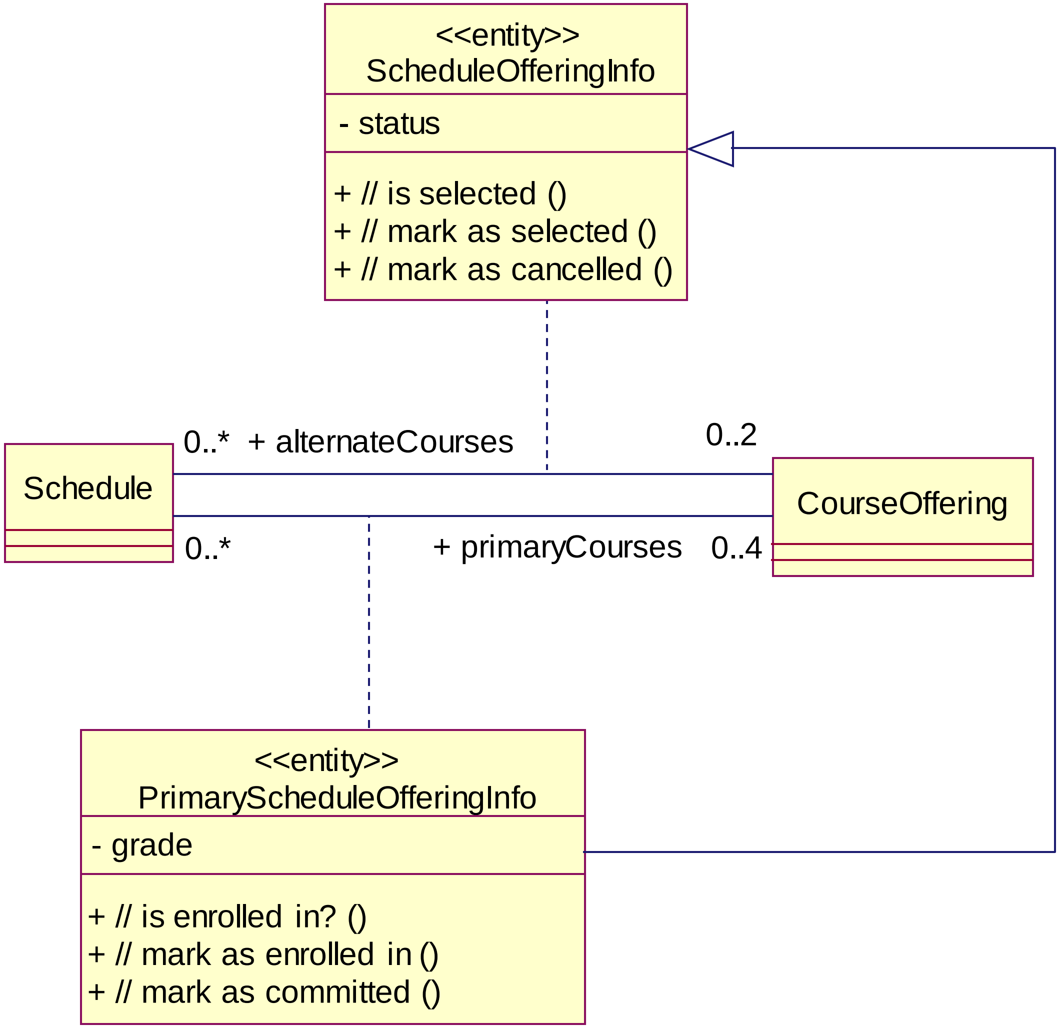


- Total number of CourseOfferings and Schedules are not small
- Must be able to navigate in both directions



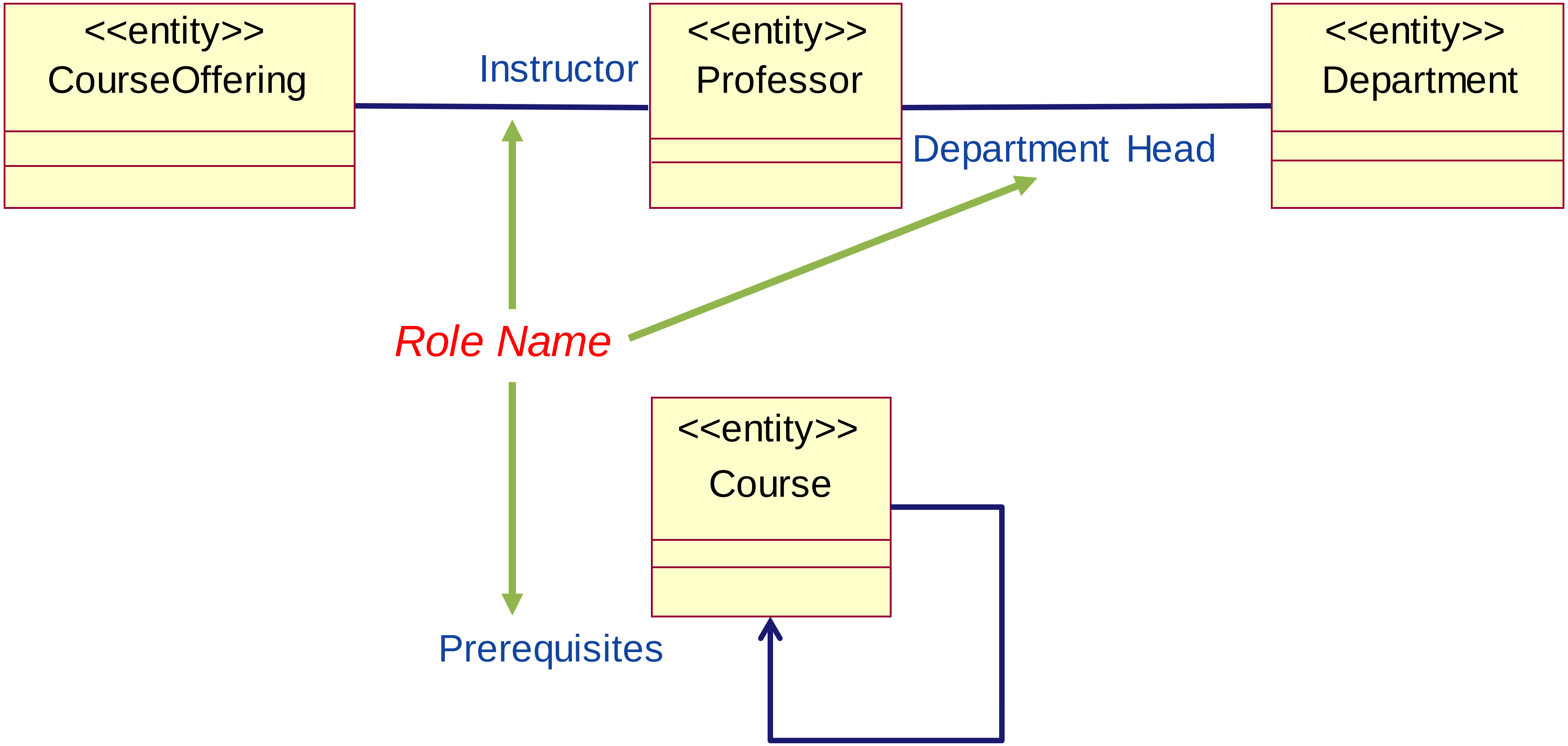
Association Class

- A class is “attached” to an association
- Contains properties of a relationship
- Has one instance per link

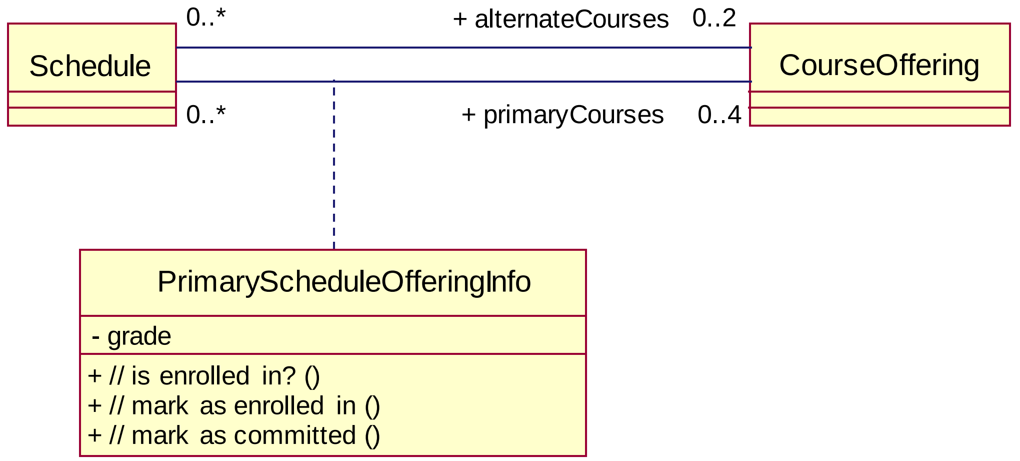


What Are Roles?

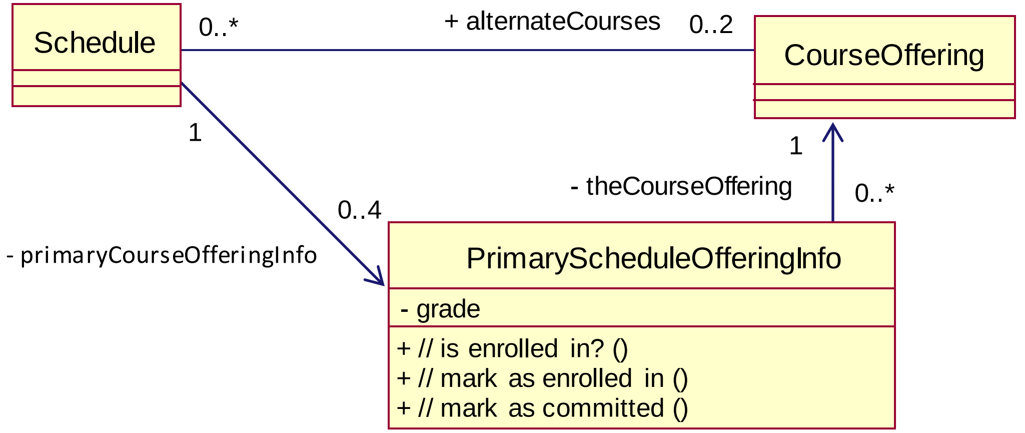
- The “face” that a class plays in the association



Example: Association Class Design

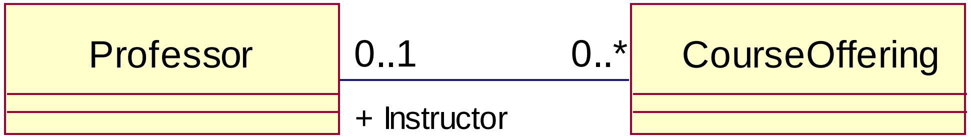


Design Decisions



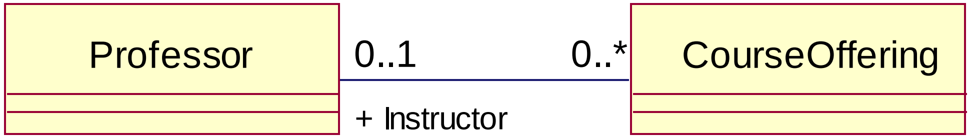
Multiplicity Design

- Multiplicity = 1, or Multiplicity = 0..1
  - May be implemented directly as a simple value or pointer
  - No further "design" is required

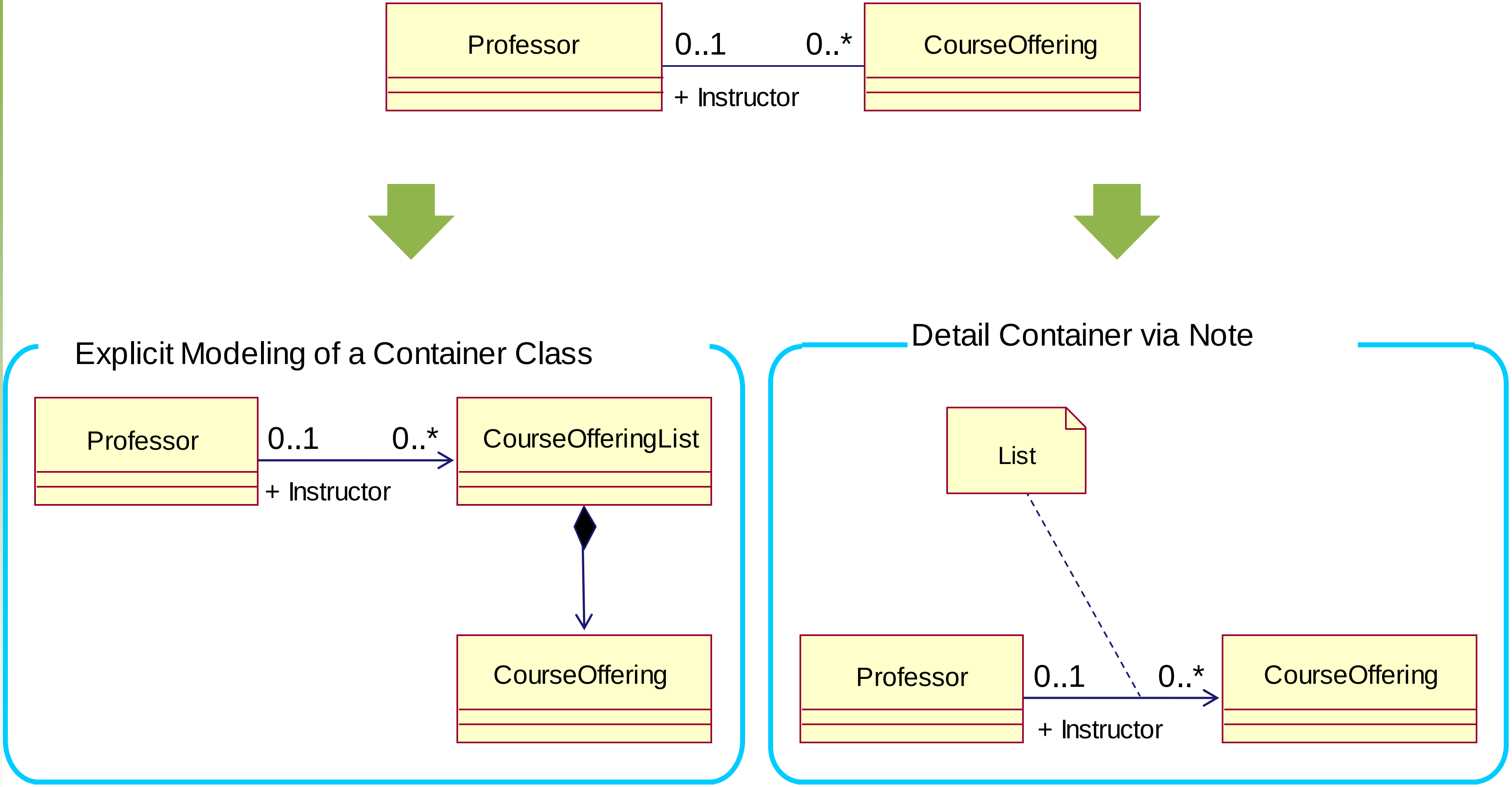


- Multiplicity > 1
  - Cannot use a simple value or pointer
  - Further "design" may be required

Needs a container for CourseOfferings



Multiplicity Design Options

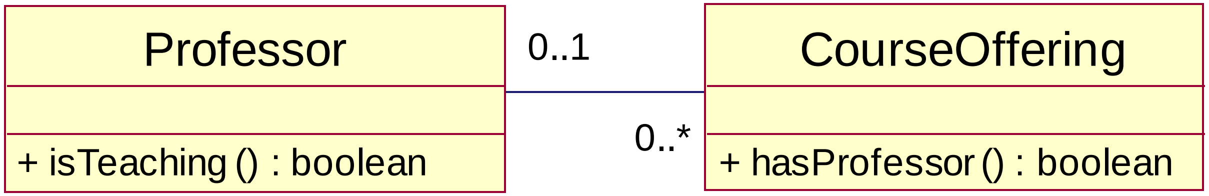


Explicit Modeling of a Container Class

Detail Container via Note

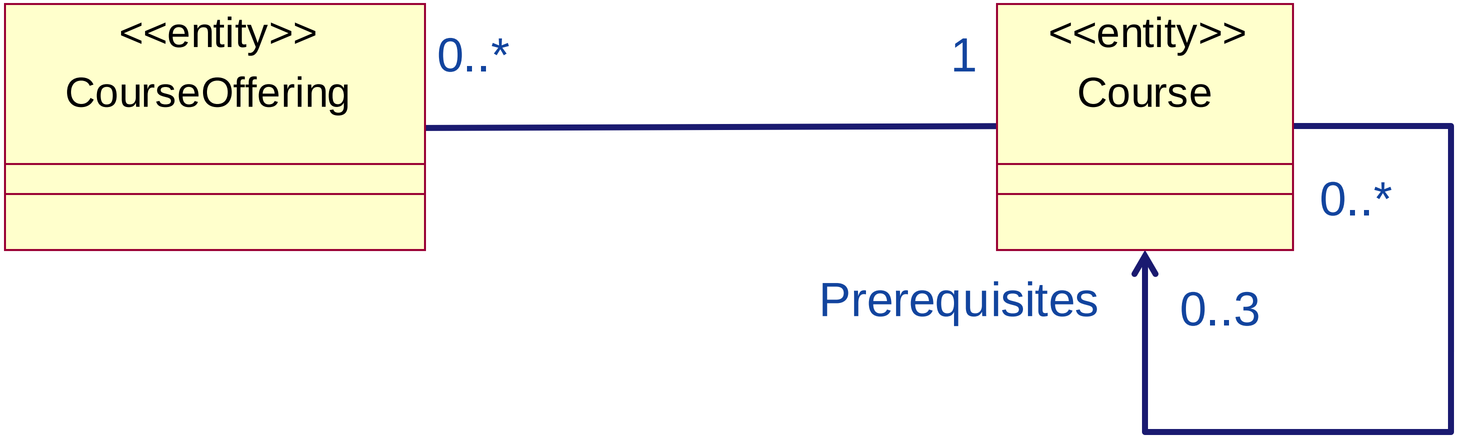
Multiplicity Design: Optionality

- If a link is optional, make sure to include an operation to test for the existence of the link



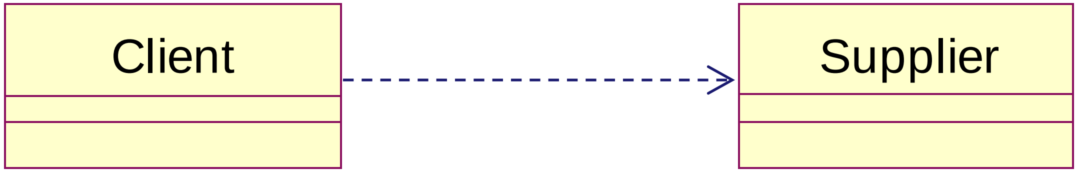
What Does Multiplicity Mean?

- Multiplicity answers two questions:
  - Is the association mandatory or optional?
  - What is the minimum and maximum number of instances that can be linked to one instance?



Define Dependency

- What Is a Dependency?
  - A relationship between two objects

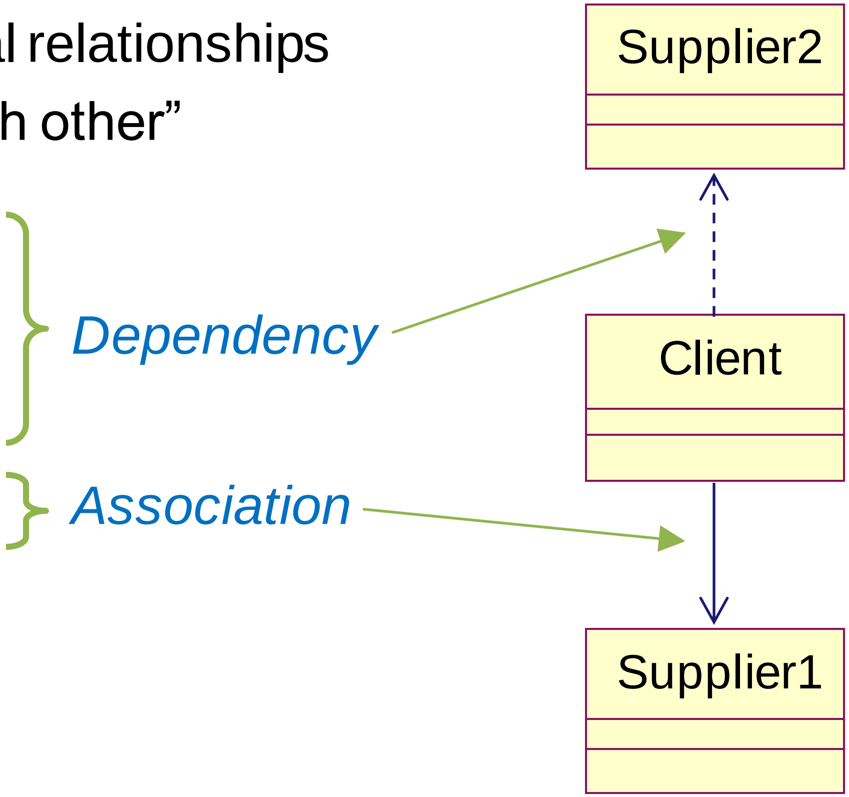


- Purpose
  - Determine where structural relationships are NOT required

- Things to look for :
  - What causes the supplier to be visible to the client

Dependencies vs. Associations

- Associations are structural relationships
- Dependencies are non-structural relationships
- In order for objects to “know each other” they must be visible
  - Local variable reference
  - Parameter reference
  - Global reference
  - Field reference



Dependency

Association

Supplier2

Client

Supplier1

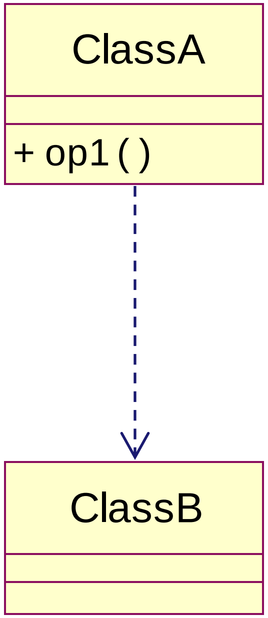
Associations vs. Dependencies in Collaborations

- An instance of an association is a link
  - All links become associations unless they have global, local, or parameter visibility
  - Relationships are context-dependent
- Dependencies are transient links with:
  - A limited duration
  - A context-independent relationship
  - A summary relationship

A dependency is a secondary type of relationship in that it doesn't tell you much about the relationship. For details you need to consult the collaborations.

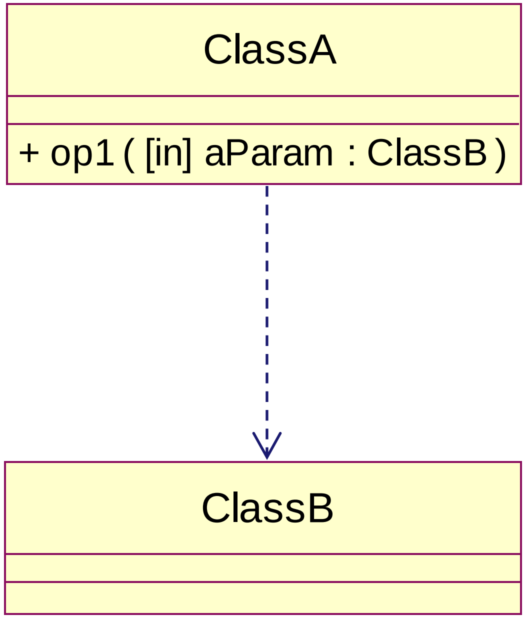
Local Variable Visibility

- The op1() operation contains a local variable of type ClassB



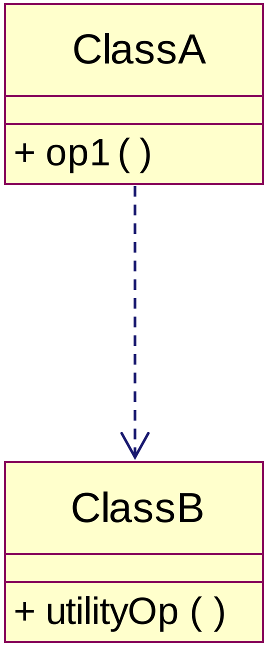
Parameter Visibility

- The ClassB instance is passed to the ClassA instance



Global Visibility

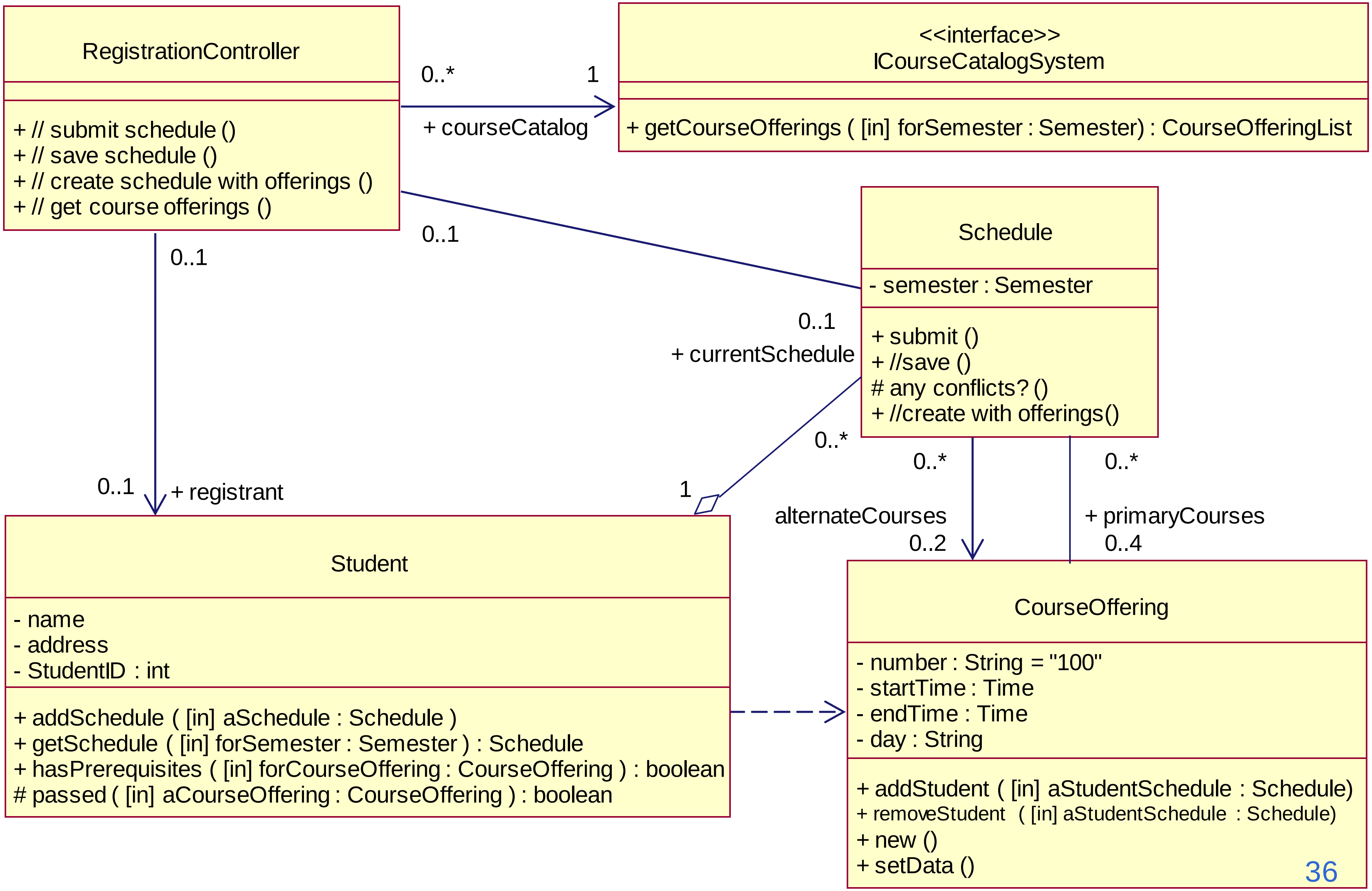
- The ClassUtility instance is visible because it is global



Identifying Dependencies: Considerations

- Permanent relationships — Association (field visibility)
- Transient relationships — Dependency
  - Multiple objects share the same instance
    - Pass instance as a parameter (parameter visibility)
    - Make instance a managed global (global visibility)
  - Multiple objects don’t share the same instance (local visibility)
- How long does it take to create/destroy?
  - Expensive? Use field, parameter, or global visibility
  - Strive for the lightest relationships possible

Example: Define Dependencies (before)



RegistrationController

+ // submit schedule ()
+ // save schedule ()
+ // create schedule with offerings ()
+ // get course offerings ()

<<interface>>
ICourseCatalogSystem

+ getCourseOfferings ([in] forSemester : Semester) : CourseOfferingList

Schedule

- semester : Semester

+ submit ()
+ //save ()
# any conflicts? ()
+ //create with offerings()

Student

- name
- address
- StudentID : int

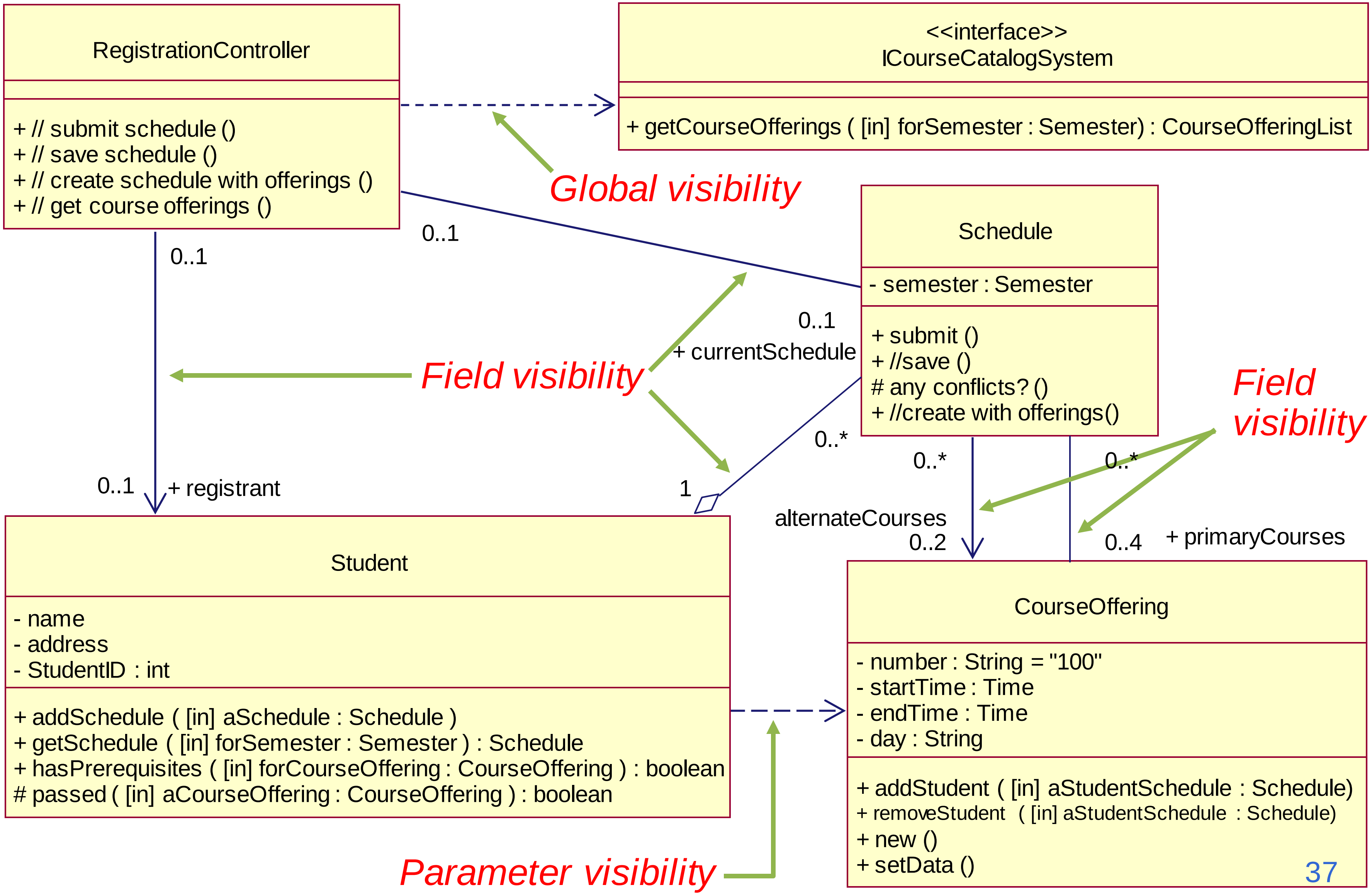
+ addSchedule ([in] aSchedule : Schedule)
+ getSchedule ([in] forSemester : Semester) : Schedule
+ hasPrerequisites ([in] forCourseOffering : CourseOffering) : boolean
# passed ([in] aCourseOffering : CourseOffering) : boolean

CourseOffering

- number : String = "100"
- startTime : Time
- endTime : Time
- day : String

+ addStudent ([in] aStudentSchedule : Schedule)
+ removeStudent ([in] aStudentSchedule : Schedule)
+ new ()
+ setData ()

Example: Define Dependencies (after)



RegistrationController

+ // submit schedule ()
+ // save schedule ()
+ // create schedule with offerings ()
+ // get course offerings ()

<<interface>>
ICourseCatalogSystem

+ getCourseOfferings ([in] forSemester : Semester) : CourseOfferingList

Schedule

- semester : Semester

+ submit ()
+ //save ()
# any conflicts? ()
+ //create with offerings()

Student

- name
- address
- StudentID : int

+ addSchedule ([in] aSchedule : Schedule)
+ getSchedule ([in] forSemester : Semester) : Schedule
+ hasPrerequisites ([in] forCourseOffering : CourseOffering) : boolean
# passed ([in] aCourseOffering : CourseOffering) : boolean

CourseOffering

- number : String = "100"
- startTime : Time
- endTime : Time
- day : String

+ addStudent ([in] aStudentSchedule : Schedule)
+ removeStudent ([in] aStudentSchedule : Schedule)
+ new ()
+ setData ()

Global visibility

Field visibility

Field visibility

Parameter visibility