

# Modeling with objects. Part I: structural (static) models

Lecture 23

# The goal of system modeling

The goal is to render precise, concise “blueprint” of a software system

The model is a **tool of communication**:

- To the client
- To the software team
- To the testing team
- To the maintenance team

# Unified modeling language (UML)

- UML – a widely accepted universal notation for describing system models
- Created by James Rumbaugh, Grady Booch, and Ivar Jacobson (“Three amigos”) – in use since 1997
- UML is a part of RUP (Rational Unified Process) methodology for system modeling

# Computer-Aided Software Engineering (CASE) tools

Pro:

- Ease of use
- Automatic content generators
- Automatic code generators
- Project management tools

Contra:

- Inflexible
- May lack exporting into a vendor-neutral format
- Form over substance

# Main steps of system modeling

1. Narrative problem statement (system requirements)
2. Data modeling (static system modeling)
3. Modeling functional side (dynamic system modeling)
4. Testing the model: is it functionally correct

# 1. Requirements

- Describe **categories** of users
- Describe **situations** for each user
- Subdivided into:
  - *Functional requirements*: goal-oriented and look-and-feel. Goal-oriented – about **what** system does
  - *Technical requirements* – about **how** system does it

# Example: Functional requirements (p.1/3)

## **Automated Student Registration System (SRS)**

- **Overall system goal:** This system will enable students to register online for courses each semester, as well as track a student's progress toward completion of his or her degree.

# SRS Functional requirements (p.2/3)

- When a student first enrolls at the university, the student uses the SRS to set forth a plan of study as to which courses he or she plans on taking to satisfy a particular degree program, and chooses a faculty advisor.
- The SRS will verify whether or not the proposed plan of study satisfies the requirements of the degree that the student is seeking. Once a plan of study has been established, then, during the registration period preceding each semester, the student is able to view the schedule of classes online, and choose whichever classes he or she wishes to attend, indicating the preferred section (day of week and time of day) if the class is offered by more than one professor. The SRS will verify whether or not the student has satisfied the necessary prerequisites for each requested course by referring to the student's online transcript of courses completed and grades received (the student may review his or her transcript online at any time).



# SRS Functional requirements (p.3/3)

- Assuming that (a) the prerequisites for the requested course(s) are satisfied, (b) the course(s) meets one of the student's plan of study requirements, and (c) there is room available in each of the class(es), the student is enrolled in the class(es).
- If (a) and (b) are satisfied, but (c) is not, the student is placed on a first-come, first-served waiting list. If a class/section that the student was previously waitlisted for becomes available (either because some other student has dropped the class or because the seating capacity for the class has been increased), the student is automatically enrolled in the waitlisted class, and an email message to that effect is sent to the student. It is the student's responsibility to drop the class if it is no longer desired; otherwise, he or she will be billed for the course.
- Students may drop a class up to the end of the first week of the semester in which the class is being taught.

# Formalizing requirements

- Identify actors and roles
- Identify system scope
- Generate use cases

# Actors and roles

- Actor – anybody or anything which will interact with our system
- Human users
- Other programs

# SRS actors: humans

- Student
- Faculty
- Department chair
- Registrar's office
- Alumni
- Prospective students

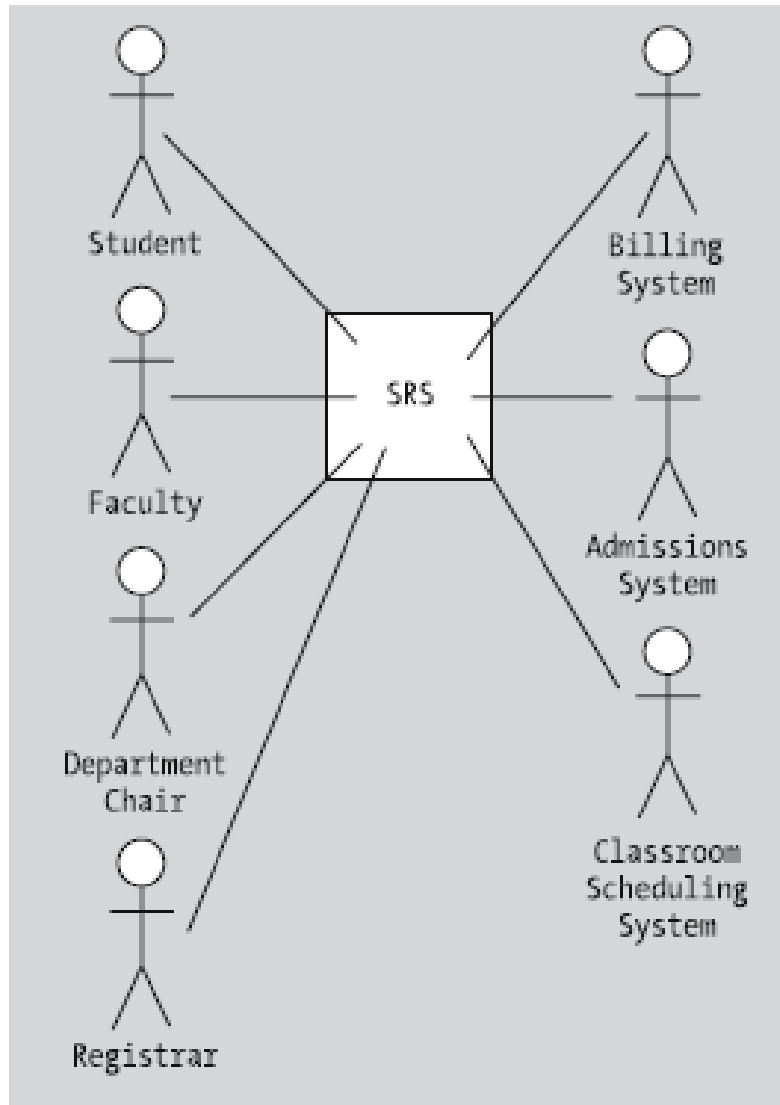
# SRS actors: computerized systems

- Billing system
- Classroom scheduling system
- Admissions system

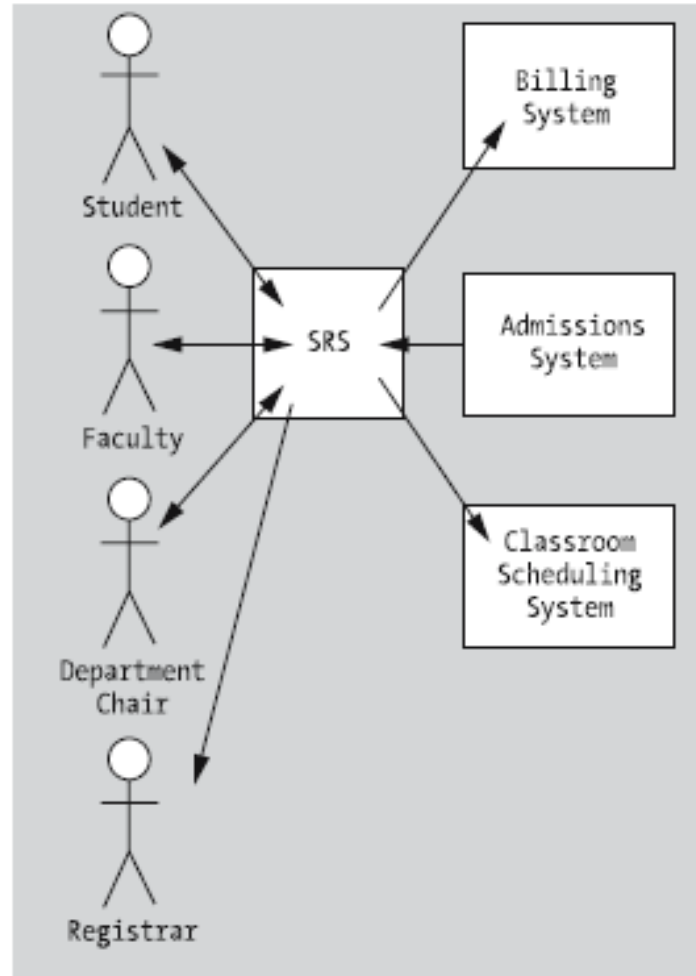
# System scope

- Remove actors which are outside the scope of a system
- Client decides

# UML general use case diagram



# A customized version: direction of information flow





# Use cases

- The logical thread of the system: a sequence of possible events
- A use case is always initiated by an actor
- High-level use cases can be decomposed into more detailed use cases

# List all high-level use cases

## Student:

- Register for a Course
- Drop a Course
- Determine a Student's Course Load
- Choose a Faculty Advisor
- Establish a Plan of Study
- View the Schedule of Classes

## Faculty:

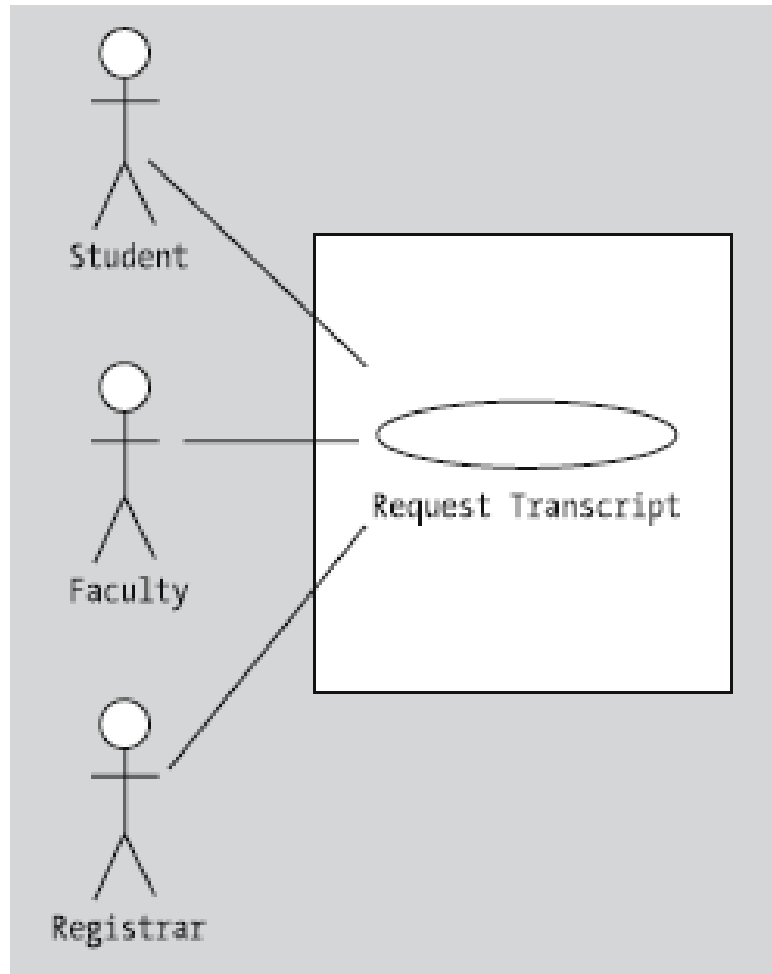
- Request a Student Roster for a Given Course
- Request a Transcript for a Given Student
- Post Final Semester Grades for a Given Course
  
- Maintain Course Information (e.g., change the course description, reflect a different instructor for the course, and so on)
- Determine a Student's Eligibility for Graduation

# Each use case can be decomposed

## **Register for a course:**

- 1.** Verify that a student has met the prerequisites.
- 2.** Check student's plan of study to ensure that this course is required.
- 3.** Check for availability of a seat in the course.
- 4.** (Optionally) Place student on a wait list.

# UML specific Use case diagram

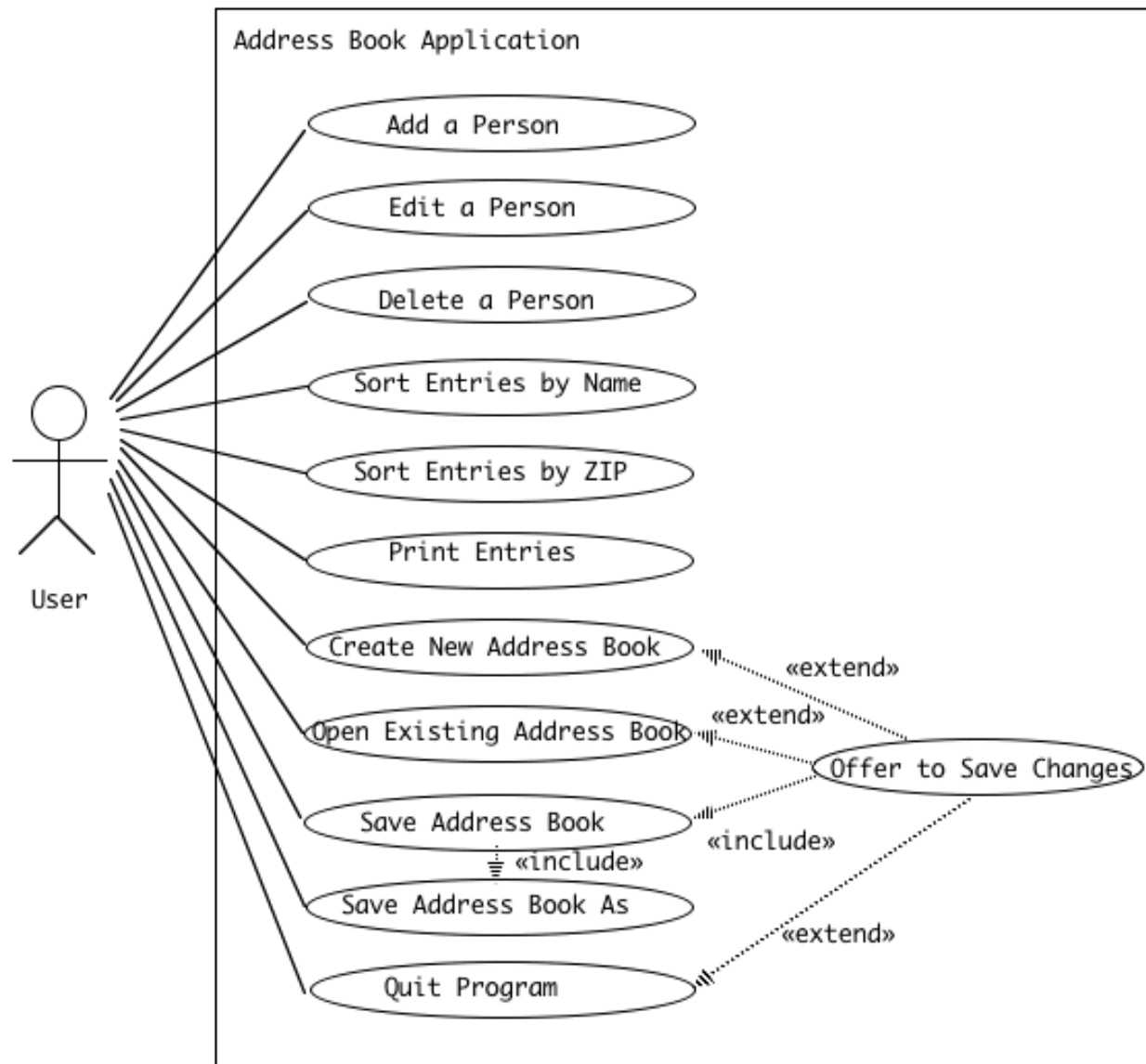


# Example 1: Address book

- The software to be designed can be used to maintain an address book. An address book holds a collection of entries, each recording a person's first and last names, address, city, state, zip, and phone number.
- It must be possible to add a new person to an address book, to edit existing information about a person (except the person's name), and to delete a person. It must be possible to sort the entries in the address book alphabetically by last name (with ties broken by first name if necessary), or by ZIP code (with ties broken by name if necessary). It must be possible to print out all the entries in the address book in "mailing label" format.
- It must be possible to create a new address book, to open a disk file containing an existing address book, to close an address book, and to save an address book to a disk file.

# Exercise 1

- Actors
- Scope
- Use case diagram



# Example 2: Tetris game

- *Tetris* is a puzzle game where a user manipulates pieces composed of square blocks, each made into a different geometric shape, that fall from the top of the game board to the bottom where the pieces accumulate.
- The game space is displayed as a framed screen. The shapes fall down from the top of the screen. A new piece appears after the current one reaches the bottom of the available game space.
- While a piece is falling, the user should be able to move it sideways and/or rotate it in order to fit it in with the accumulated pieces.
- The goal is to fill all spaces along a horizontal line. If that is accomplished, the line is erased, points are earned, and more of the game space is available for play.
- When a certain number of lines are cleared, the game enters a new level. As the game progresses, each level causes the blocks to fall faster.
- If the pieces accumulate and reach the top of the screen, then the game is over.



# Static system modeling: identifying objects

Reminder: Software objects – abstractions of real objects and concepts: physical and conceptual objects

# Executive summary of requirements

- For modeling purposes, the long requirements document should be transformed into an *executive summary*
- The executive summary should then be verified with the future users of the software product

# Noun phrase analysis to identify classes

List all nouns / noun phrases from requirements document

# Example: Functional requirements (p.1/3)

## **Automated Student Registration System (SRS)**

- **Overall system goal:** This system will enable students to register online for courses each semester, as well as track a student's progress toward completion of his or her degree.

# SRS Functional requirements (p.2/3)

- When a student first enrolls at the university, the student uses the SRS to set forth a plan of study as to which courses he or she plans on taking to satisfy a particular degree program, and chooses a faculty advisor.
- The SRS will verify whether or not the proposed plan of study satisfies the requirements of the degree that the student is seeking. Once a plan of study has been established, then, during the registration period preceding each semester, the student is able to view the schedule of classes online, and choose whichever classes he or she wishes to attend, indicating the preferred section (day of week and time of day) if the class is offered by more than one professor. The SRS will verify whether or not the student has satisfied the necessary prerequisites for each requested course by referring to the student's online transcript of courses completed and grades received (the student may review his or her transcript online at any time).




# SRS Functional requirements (p.3/3)

- Assuming that (a) the prerequisites for the requested course(s) are satisfied, (b) the course(s) meets one of the student's plan of study requirements, and (c) there is room available in each of the class(es), the student is enrolled in the class(es).
- If (a) and (b) are satisfied, but (c) is not, the student is placed on a first-come, first-served waiting list. If a class/section that the student was previously waitlisted for becomes available (either because some other student has dropped the class or because the seating capacity for the class has been increased), the student is automatically enrolled in the waitlisted class, and an email message to that effect is sent to the student. It is the student's responsibility to drop the class if it is no longer desired; otherwise, he or she will be billed for the course.
- Students may drop a class up to the end of the first week of the semester in which the class is being taught.

# Elimination

- Replace plural with singular
- Sort and remove duplicates
- Remove 'system', context (University)
- Remove processes
- Consolidate synonyms
- Remove roles

# Elimination examples

- ~~Class, course, section~~
- Transcript, ~~courses completed, grades received~~  Logical synonyms
- Professor, ~~faculty advisor~~  Roles
- ~~Waitlisted class, prerequisite, requested course~~ 



# Remaining list of potential classes

- Course
- Day of week\*
- Degree\*
- Email message+
- Plan of study
- Professor
- Room\*
- Schedule of classes+
- Seating capacity\*
- Section
- Semester\*
- Student
- Time of day\*
- Transcript
- (First-come, first-served) Wait list

# Leave independent classes only

Test:

- Does this class have attributes?
- Does it provide services?

If no services for other classes – make them attributes:

- Room
- Day of week
- Degree
- Seating capacity
- Semester
- Time of day

# Remove implementation classes: leave domain classes only

- E-mail messages
- Schedule of classes

# Survived classes

- Course
- PlanOfStudy
- Professor
- Section
- Student
- Transcript
- WaitList

# Add actor classes

**Rule: if any user associated with any actor type A is going to need to manipulate (access or modify) information concerning an actor type B when A is logged on to the SRS, then B needs to be included as a class in our model**

- Student
- Faculty
- Department chair
- Registrar's office
  
- Billing system
- Classroom scheduling system
- Admissions system

# Generalization with inheritance

- Course
- Section
- WaitList
  
- PlanOfStudy
- Transcript
  
- Person:
  - Professor
  - Student

# Model dictionary (1/2)

- **Course:** A semester-long series of lectures, assignments, exams, etc., that all relate to a particular subject area, and which are typically associated with a particular number of credit hours; a unit of study toward a degree. For example, Web programming is a required **course** for the Bachelor of Computing Science degree
- **PlanOfStudy:** A list of the **courses** that a student intends to take to fulfill the **course** requirements for a particular degree.
- **Professor:** A member of the faculty who teaches **sections** or advises **students**.

# Model dictionary (2/2)

- **Section:** The offering of a particular **course** during a particular semester on a particular day of the week and at a particular time of day.
- **Student:** A person who is currently enrolled at the university and who is eligible to register for one or more **sections**.
- **Transcript:** A record of all of the **courses** taken to date by a particular **student** at this university, including which semester each **course** was taken in, the grade received, and the credits granted for the **course**, as well as a reflection of an overall total number of credits earned and the **student's** grade point average (GPA).



# Java API

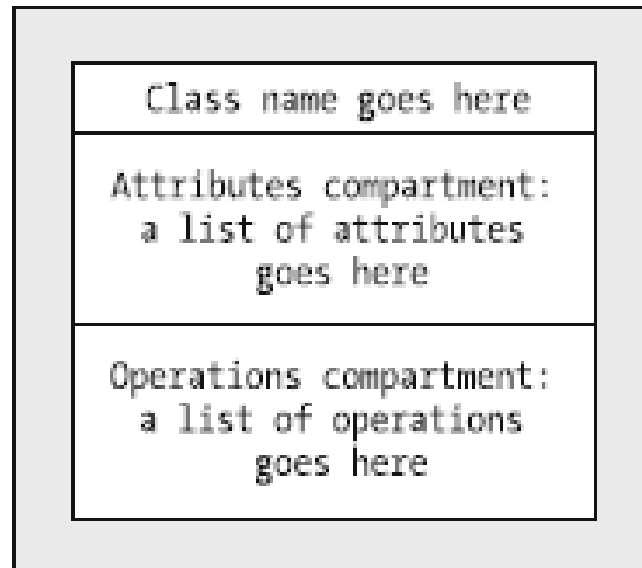
These class descriptions should be put in the header documentation of each future class:

```
/**
```

```
The class represents a person who is currently enrolled  
at the university and who is eligible to register for one  
or more sections.*/
```

```
public class Student{
```

# UML class diagram



# Example 1: Address book

- The software to be designed can be used to maintain an address book. An address book holds a collection of entries, each recording a person's first and last names, address, city, state, zip, and phone number.
- It must be possible to add a new person to an address book, to edit existing information about a person (except the person's name), and to delete a person. It must be possible to sort the entries in the address book alphabetically by last name (with ties broken by first name if necessary), or by ZIP code (with ties broken by name if necessary). It must be possible to print out all the entries in the address book in "mailing label" format.
- It must be possible to create a new address book, to open a disk file containing an existing address book, to close an address book, and to save an address book to a disk file.

# Example 2: Tetris game

- *Tetris* is a puzzle game where a user manipulates pieces composed of square blocks, each made into a different geometric shape, that fall from the top of the game board to the bottom where the pieces accumulate.
- The game space is displayed as a framed screen. The shapes fall down from the top of the screen. A new piece appears after the current one reaches the bottom of the available game space.
- While a piece is falling, the user should be able to move it sideways and/or rotate it in order to fit it in with the accumulated pieces.
- The goal is to fill all spaces along a horizontal line. If that is accomplished, the line is erased, points are earned, and more of the game space is available for play.
- When a certain number of lines are cleared, the game enters a new level. As the game progresses, each level causes the blocks to fall faster.
- If the pieces accumulate and reach the top of the screen, then the game is over.

# Determining structural relationships between classes

- Association
- Aggregation
- Composition
- Inheritance

# Verb phrases analysis method

- Find all verb phrases that suggest structural relationship between objects
- Ignore transient actions or behaviors

# Example: Functional requirements (p.1/3)

## Automated Student Registration System (SRS)

- Overall system goal: This system will **enable students to register online for courses** each semester, as well as **track a student's progress toward completion of his or her degree.**

# SRS Functional requirements (p.2/3)

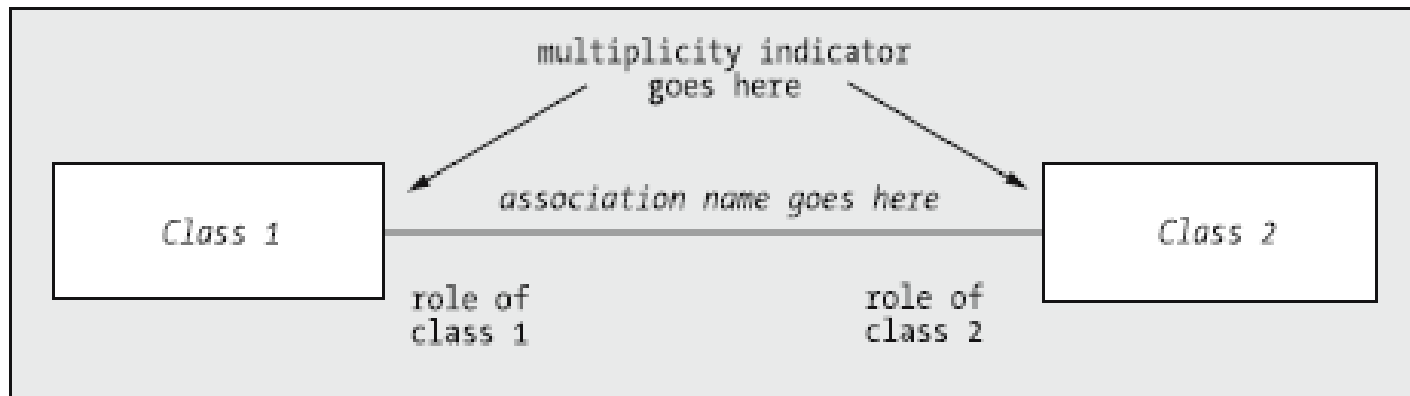
- When a student **first enrolls at the university**, the student uses the SRS to **set forth a plan of study** as to which courses he or she plans on taking to satisfy a particular degree program, and chooses a faculty advisor.
- The SRS will verify whether or not the proposed plan of study satisfies the requirements of the degree that the student is seeking. Once a plan of study has been established, then, during the registration period preceding each semester, the student is able to **view the schedule of classes online**, and choose whichever classes he or she wishes to attend, indicating the preferred section (day of week and time of day) if the class is offered by more than one professor. The SRS will verify whether or not the student has satisfied the necessary prerequisites for each requested course by referring to the student's online transcript of courses completed and grades received (the student may review his or her transcript online at any time).



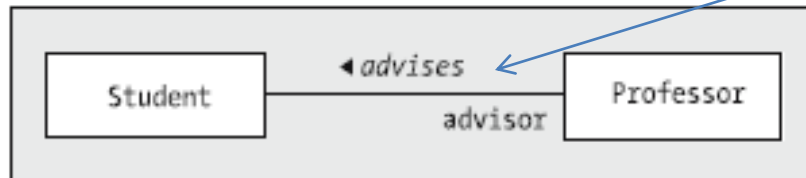
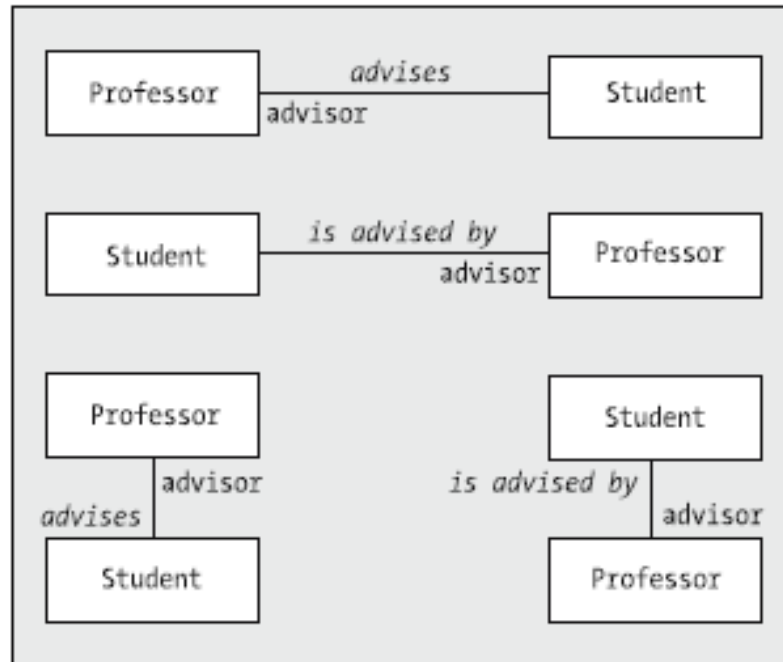
# Association matrix

	Section	Course	PlanOfStudy	Professor	Student	Transcript
Section	x	instance of	x	is taught by	is taught for	included in
Course	includes	prerequisite for	included into	x	v	x
PlanOfStudy	x	calls for	x	x	set up by	x
Professor	teaches	x	x	x	advises, teaches	x
Student	registered for	plans to take	observes	is advised by, studies under	x	owns
Transcript	includes	x	x	x	belongs to	x

# Representing structural relationships between classes

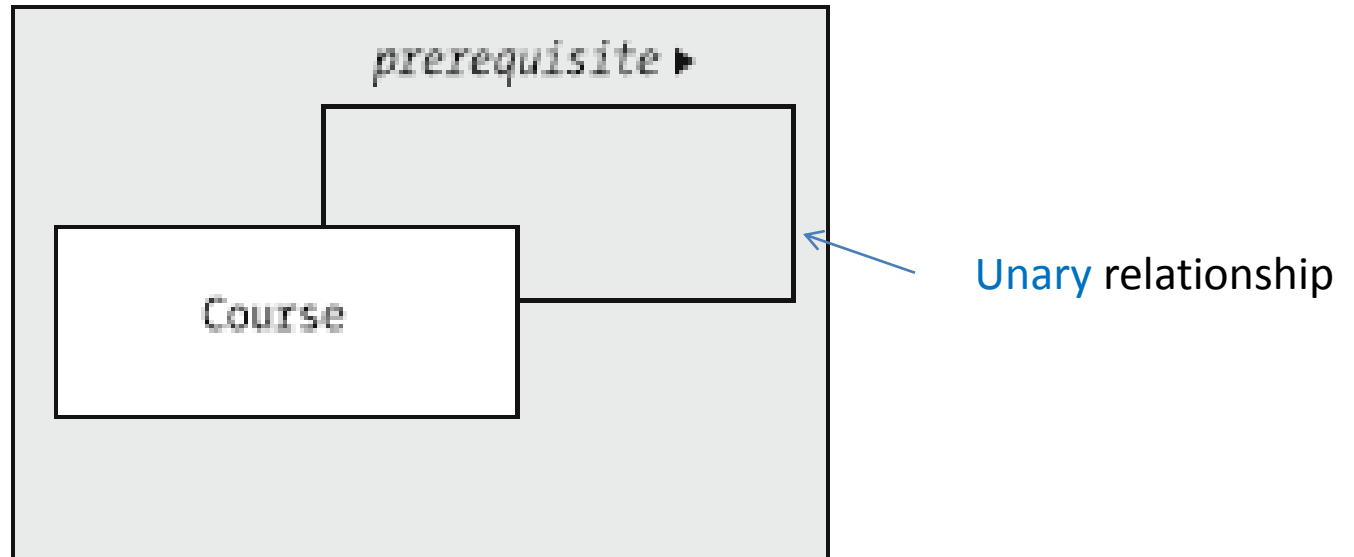


# Example: *advises* relationship

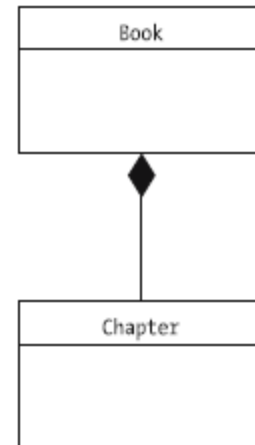
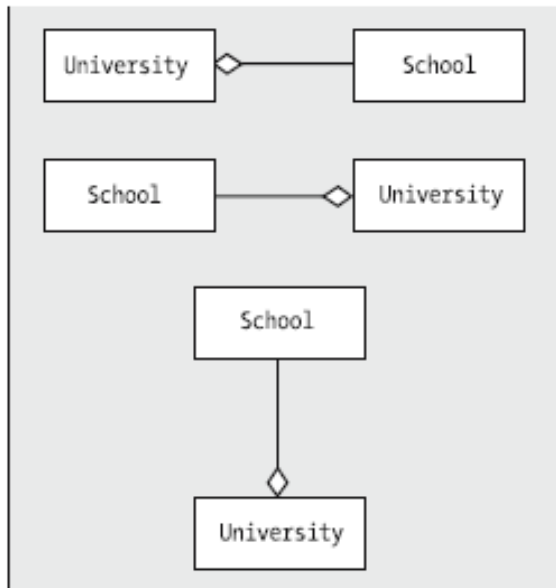


Binary  
relationship  
(between two  
different classes)

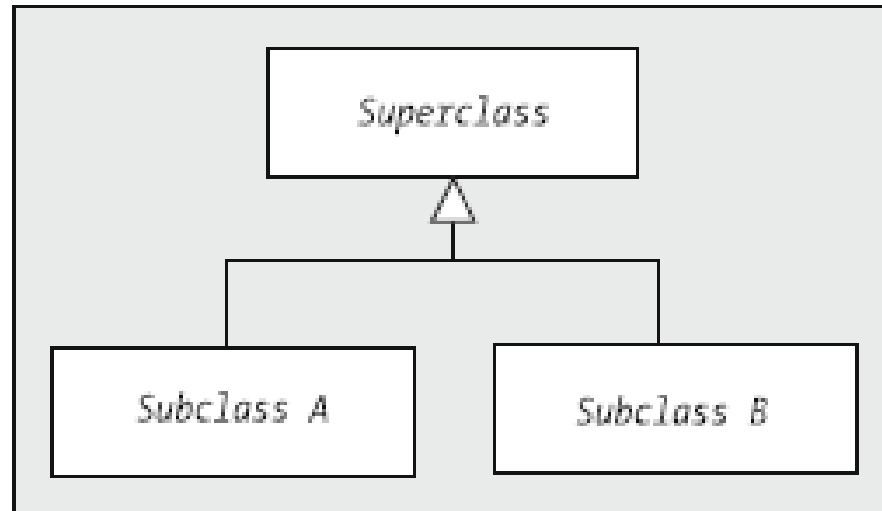
# Example: *prerequisite* relationship



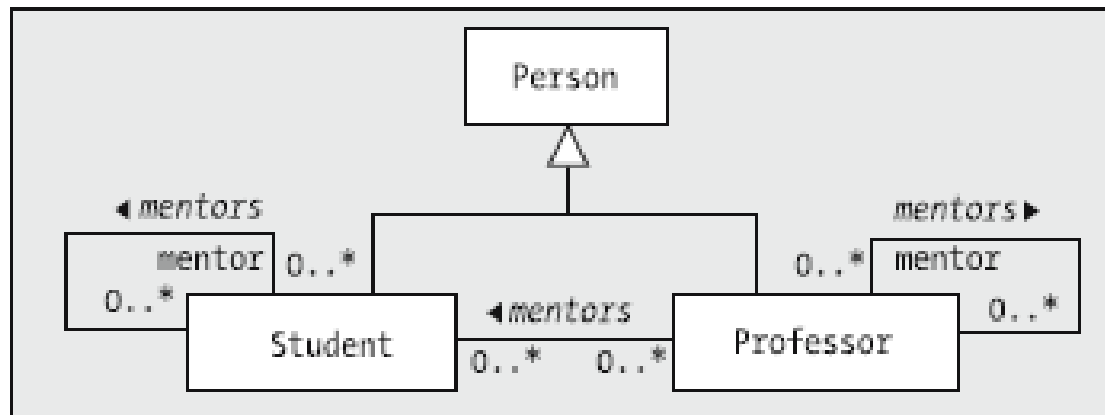
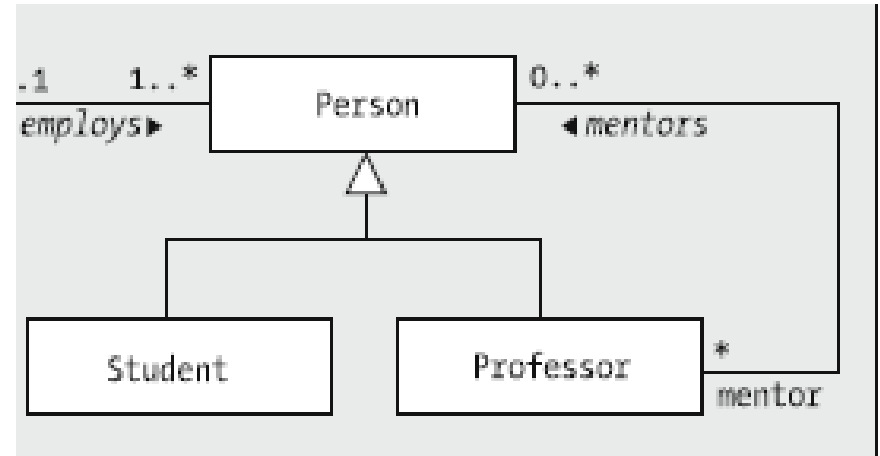
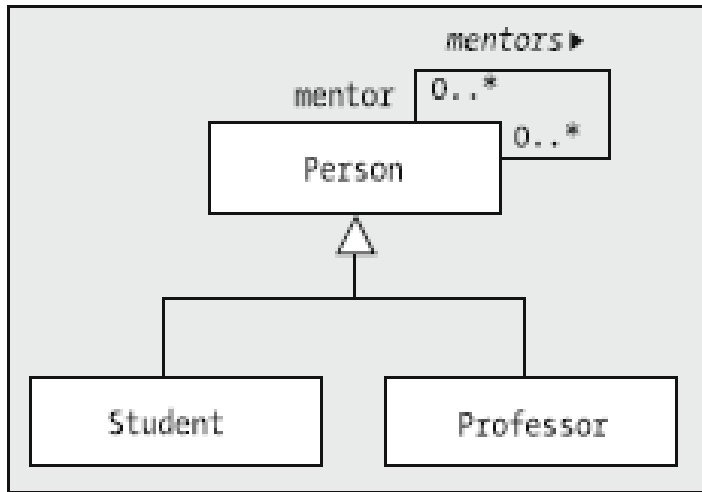
# Aggregation and composition



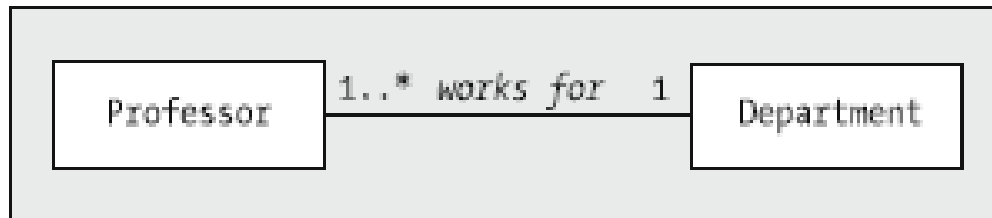
# Inheritance



# Associations are inherited

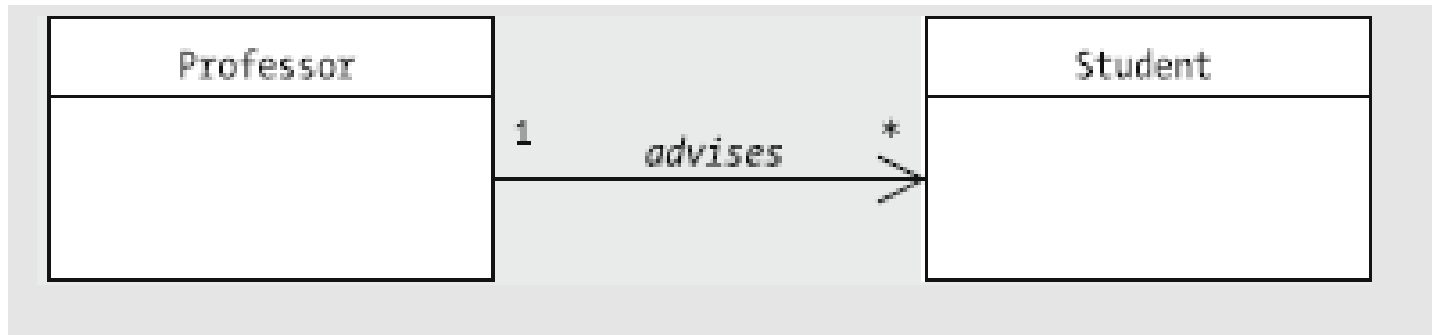


# Reflecting multiplicity

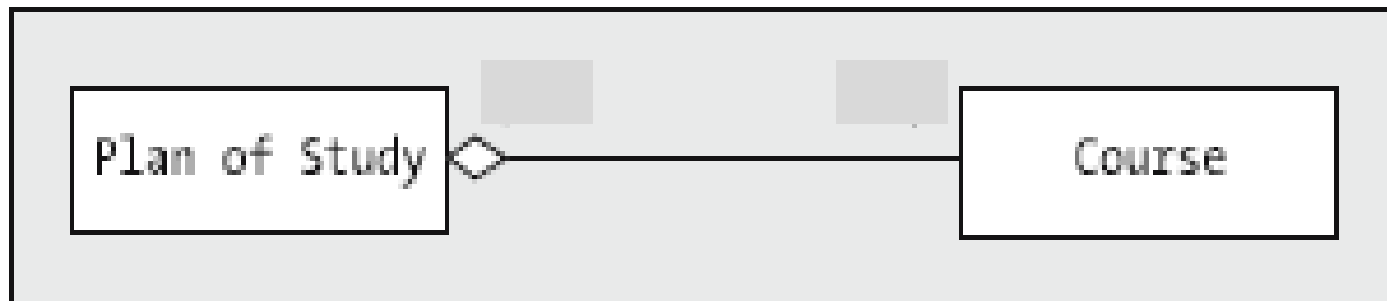
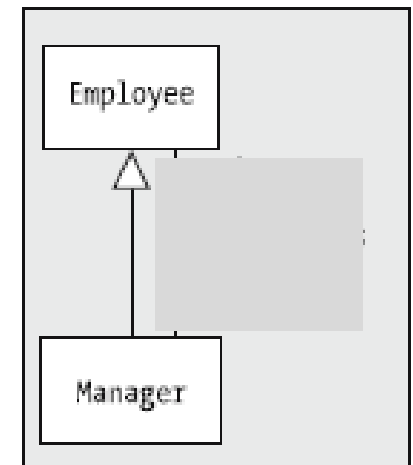
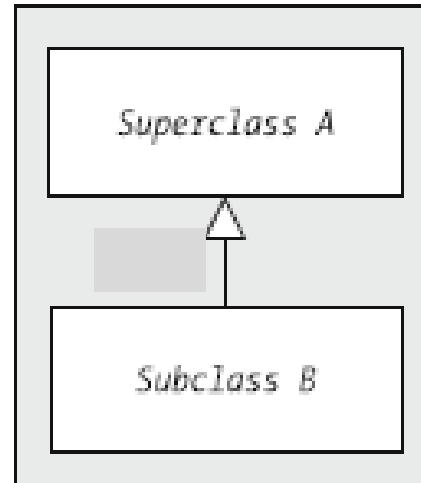
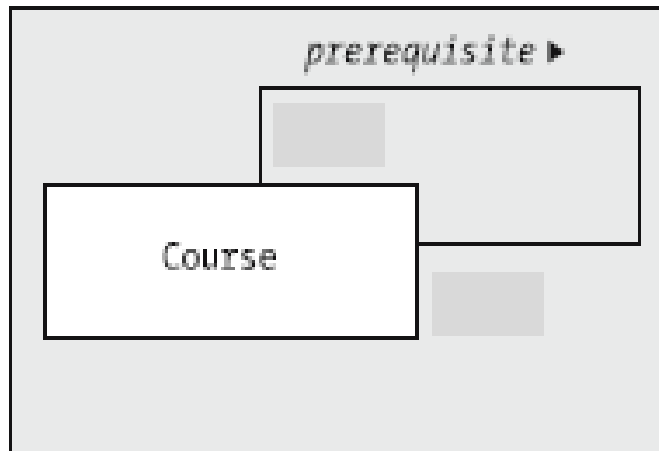




# Directions: who initiates conversation



# Exercise: multiplicity



# Instance variables depict associations

```
public class Course {  
    // Attributes.  
    // a collection of Section object handles  
    private Collection offeredAs;  
}
```

```
public class Section {  
    // Attributes.  
    // a handle on the related Course object  
    private Course representedCourse;  
}
```

1 to many

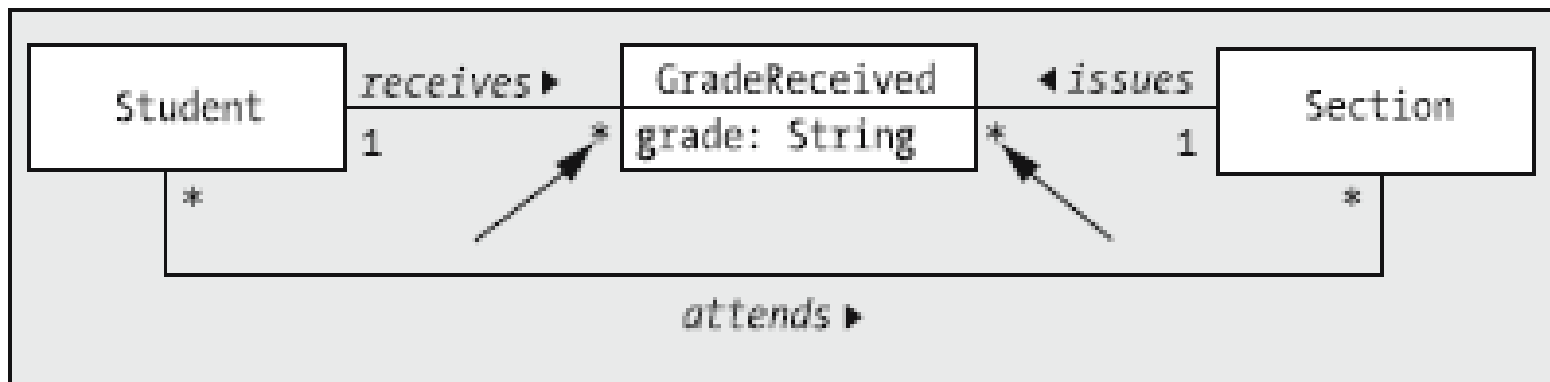


# Association classes

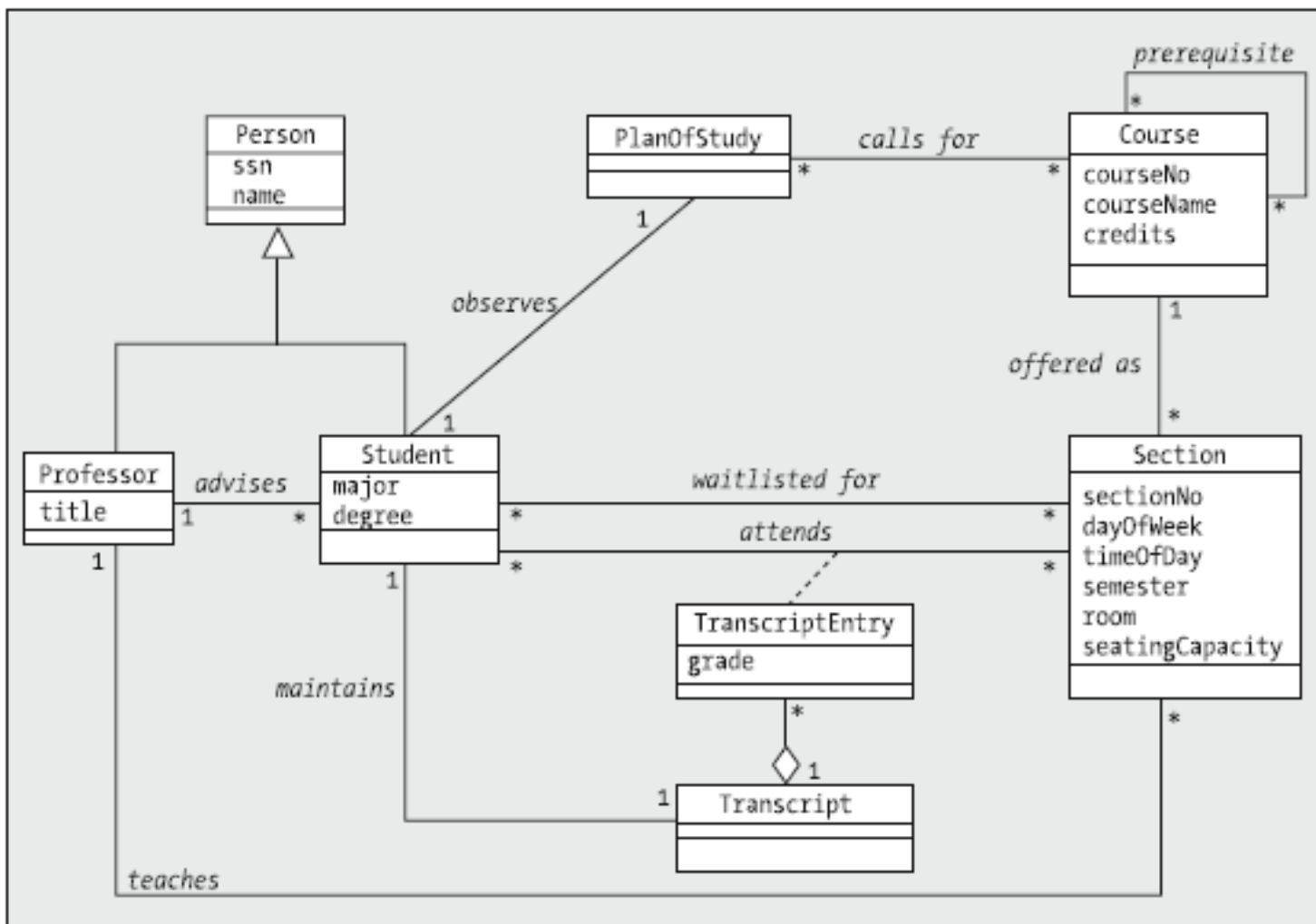


- Where do we put `gradeReceived`?

# Association classes



# Final static class diagram



# Object (instance) diagrams

