


Modeling with objects. Part II: dynamic models

Lecture 24

To complete UML class diagram:

- Classes
- Structural relationships between classes: association, aggregation, composition
- Generalization/Specialization hierarchies
- Attributes
-  Methods

Interacting objects

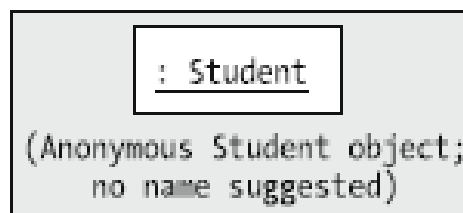
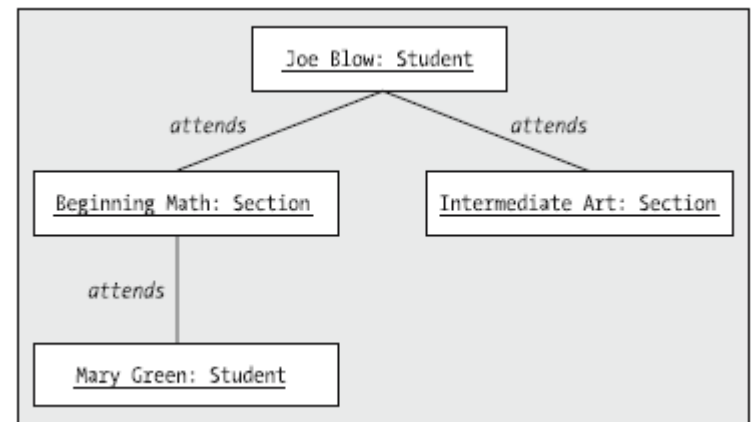
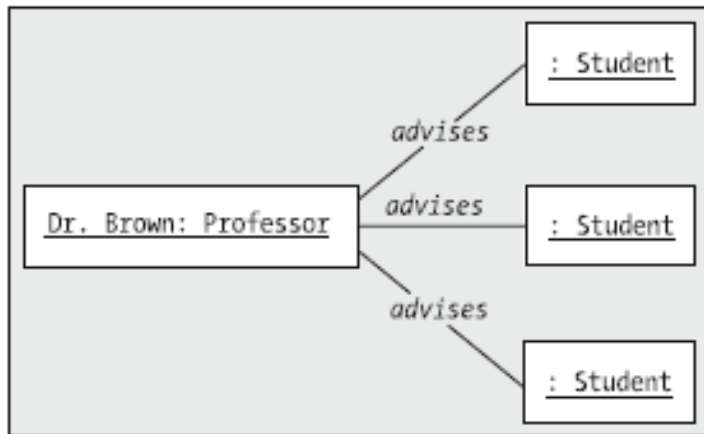
- We can determine the required methods of a given object only by looking what services are expected from this object by other participating objects
- In order to find all the methods, we need to go through sequences of possible events

Object State

- The collective set of all object attribute values at a given point of time
- This includes 'simple' attributes and attributes which represent references to other objects

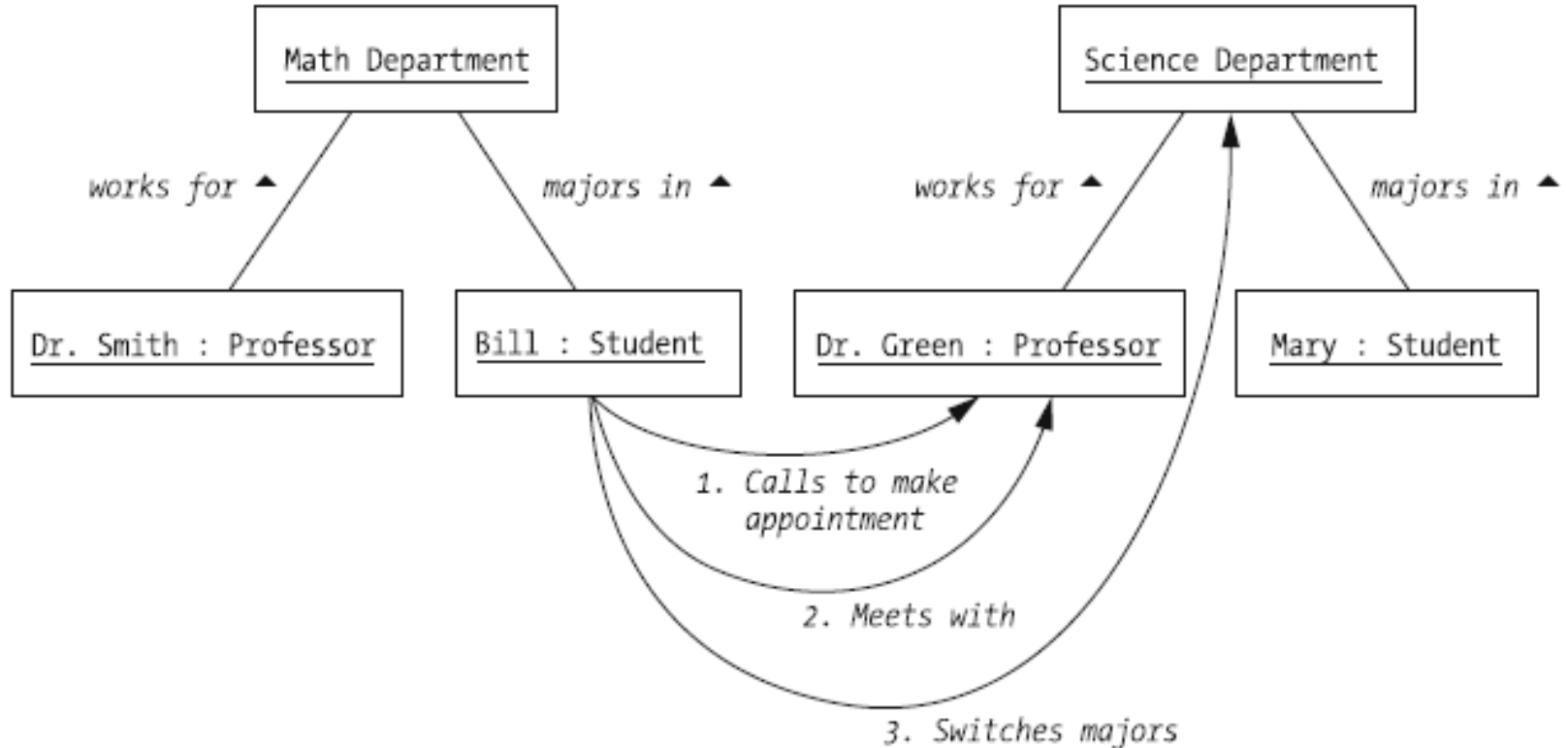
Object (**instance**) diagrams

- Reflect a sample state of a given object

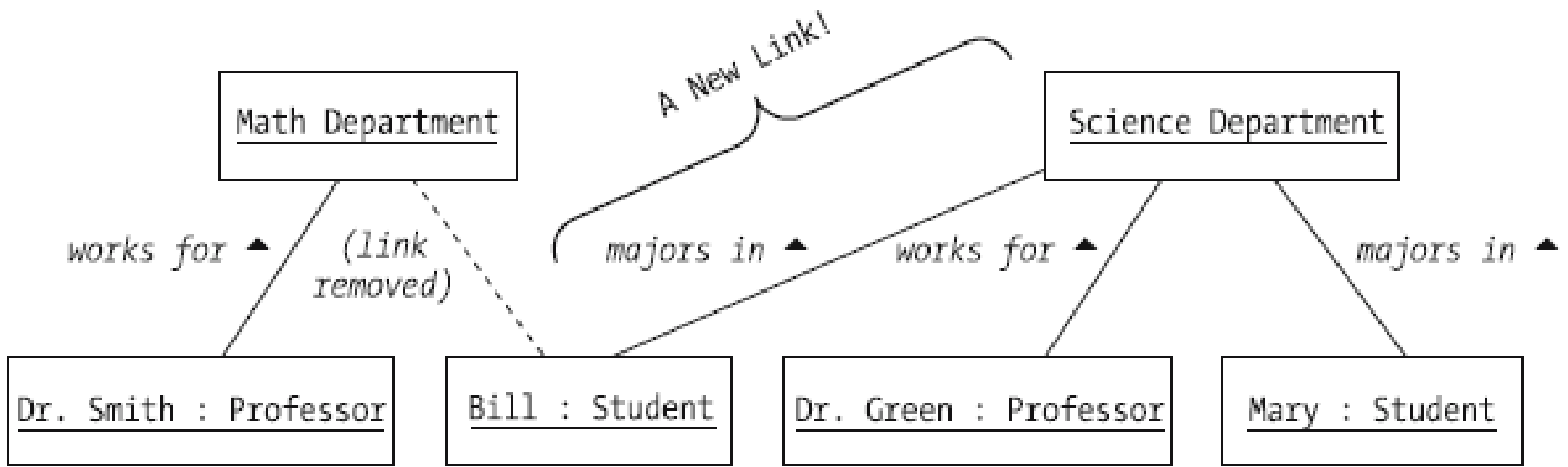


Object interactions affect their state

Example: switching majors



Only some objects change state as a result of interaction



Object interactions are triggered by *events*

- Event->message=method call
- An event can be initiated by:
 - user
 - another automatic system
 - another object



Internal event

Response to a message by an object

- Change its state

```
Professor p = new Professor();  
Student s = new Student();  
p.addAdvisee(s);
```

- Delegate message to another object

```
Section x = new Section();  
Student s = new Student();  
x.register(s);
```

//Inside Section class:

```
public boolean register(Student s) {  
    boolean completed = s.successfullyCompleted(some prerequisite);  
    if (completed) { //register the student and return true  
        else { return false;}  
    }  
}
```

- Return a value
- Ignore

```
// Professor p "ignores" this message, if a student is already in his advisee list  
p.addAdvisee(s);
```

Scenarios

Scenario: A sequence of events esp. when imagined; esp. : an account or synopsis of a possible course of action or events.

Dictionary definition

- A scenario depicts a sequence of internal events
- Each scenario is a hypothetical instance of how a particular use case might play out

Scenario = *instance of use case*

Best case scenario #1:

Instance of use case “Registration for a course”

1. Fred, a student, logs on to the SRS.
2. He views the schedule of classes for the current semester to determine which section(s) he wishes to register for.
3. Fred requests a seat in a particular section of a course entitled “Object-oriented programming,” course number CSCI331, section 1.
4. Fred’s plan of study is checked to ensure that the requested course is appropriate for his overall degree goals. (Assuming students are not permitted to take courses outside of their plans of study.)
5. His transcript is checked to ensure that he has satisfied all of the prerequisites for the requested course, if there are any.
6. Seating availability in the section is confirmed.
7. The section is added to Fred’s current course load.

Scenario #2:

Instance of use case “Registration for a course”

1. -||-
2. -||-
3. -||-
4. -||-
5. His transcript is checked to ensure that he has satisfied all of the prerequisites for the requested course, if there are any.
6. ***Seating availability in the section is checked, but the section is found to be full.***
7. ***Fred is asked if he wishes to be put on a first come, first served wait list.***
8. ***Fred chooses to be placed on the wait list.***

Sequence diagrams

- Used to formalize scenarios
- Then used to deduce required object methods

Steps to build UML sequence diagrams for a given scenario

1. Determine participating objects and actors
2. Place **objects** horizontally and draw *life lines* for each object
3. Indicate sending of each message by a solid line between a pair of objects
4. Indicate returning message by a dashed line
5. Arrange sending and receiving of messages chronologically from top to bottom according to the given scenario

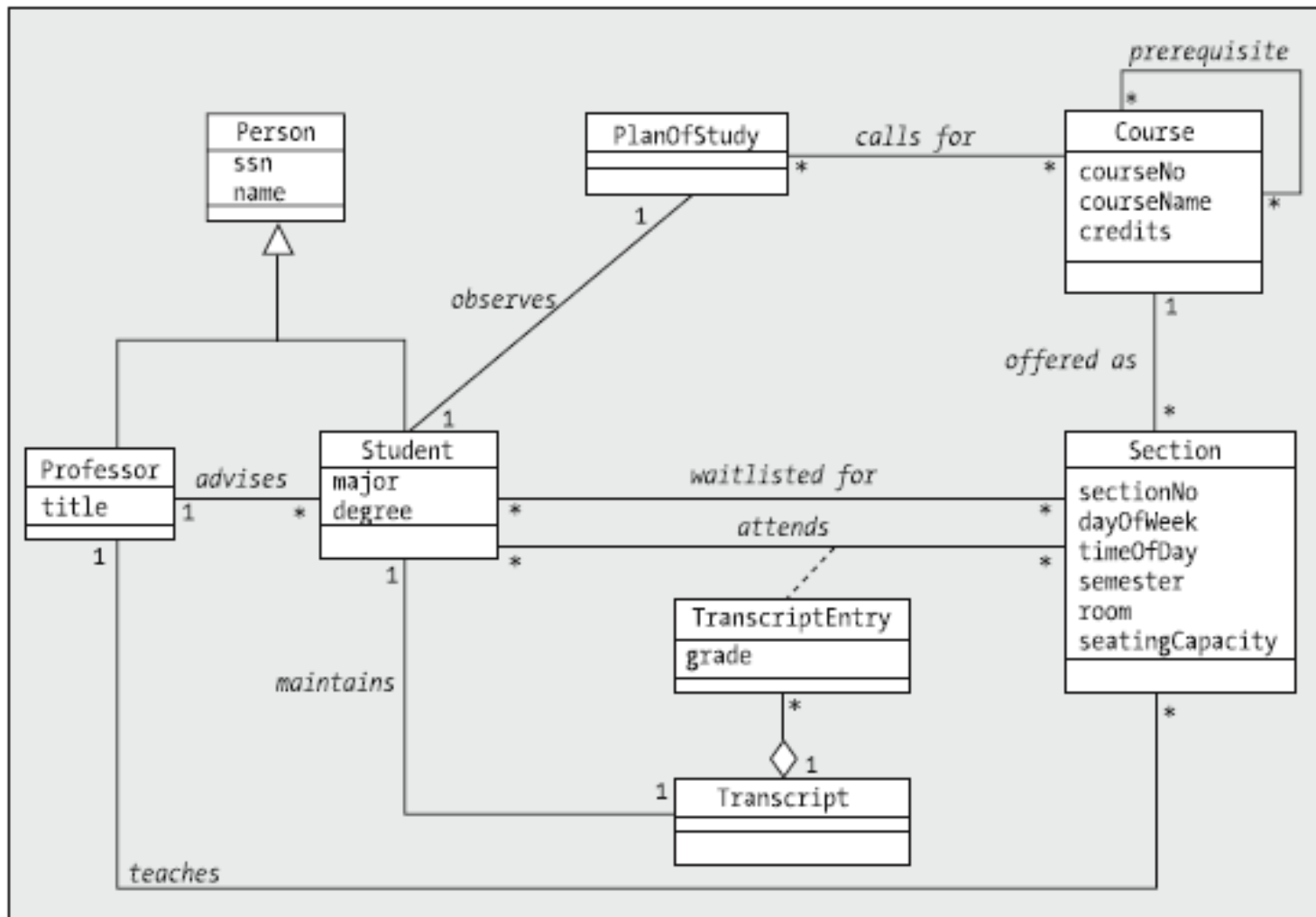
Scenario #1:

1. Fred, a student, logs on to the SRS.
2. He views the schedule of classes for the current semester to determine which section(s) he wishes to register for.
3. Fred requests a seat in a particular section of a course entitled “Object-oriented programming,” course number CSCI331, section 1.
4. Fred’s plan of study is checked to ensure that the requested course is appropriate for his overall degree goals. (Assuming students are not permitted to take courses outside of their plans of study.)
5. His transcript is checked to ensure that he has satisfied all of the prerequisites for the requested course, if there are any.
6. Seating availability in the section is confirmed.
7. The section is added to Fred’s current course load.

Sequence diagram Step 1. List of participants

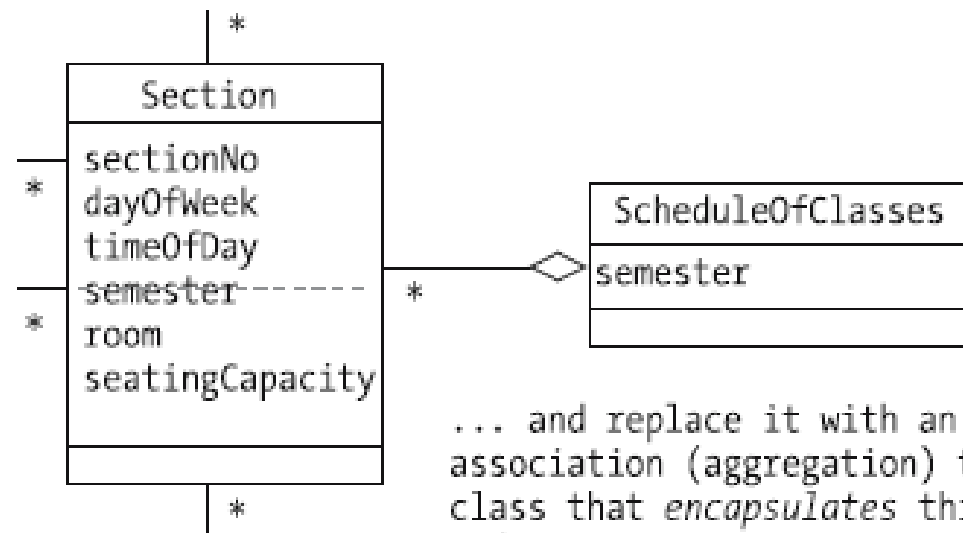
- One **Student object** (representing Fred)
- One **Section object** (representing the course entitled “Object-oriented programming”)
- One **PlanOfStudy object**, belonging to Fred
- One **Transcript object**, also belonging to Fred
- One **ScheduleOfClasses object**
- One **Student actor** (Fred again!)

We don't have ScheduleOfClasses in our structural class diagram



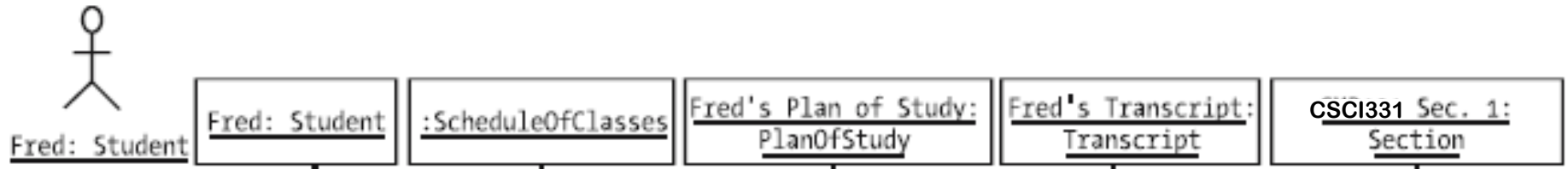
Update UML class diagram

Remove
an attribute ...

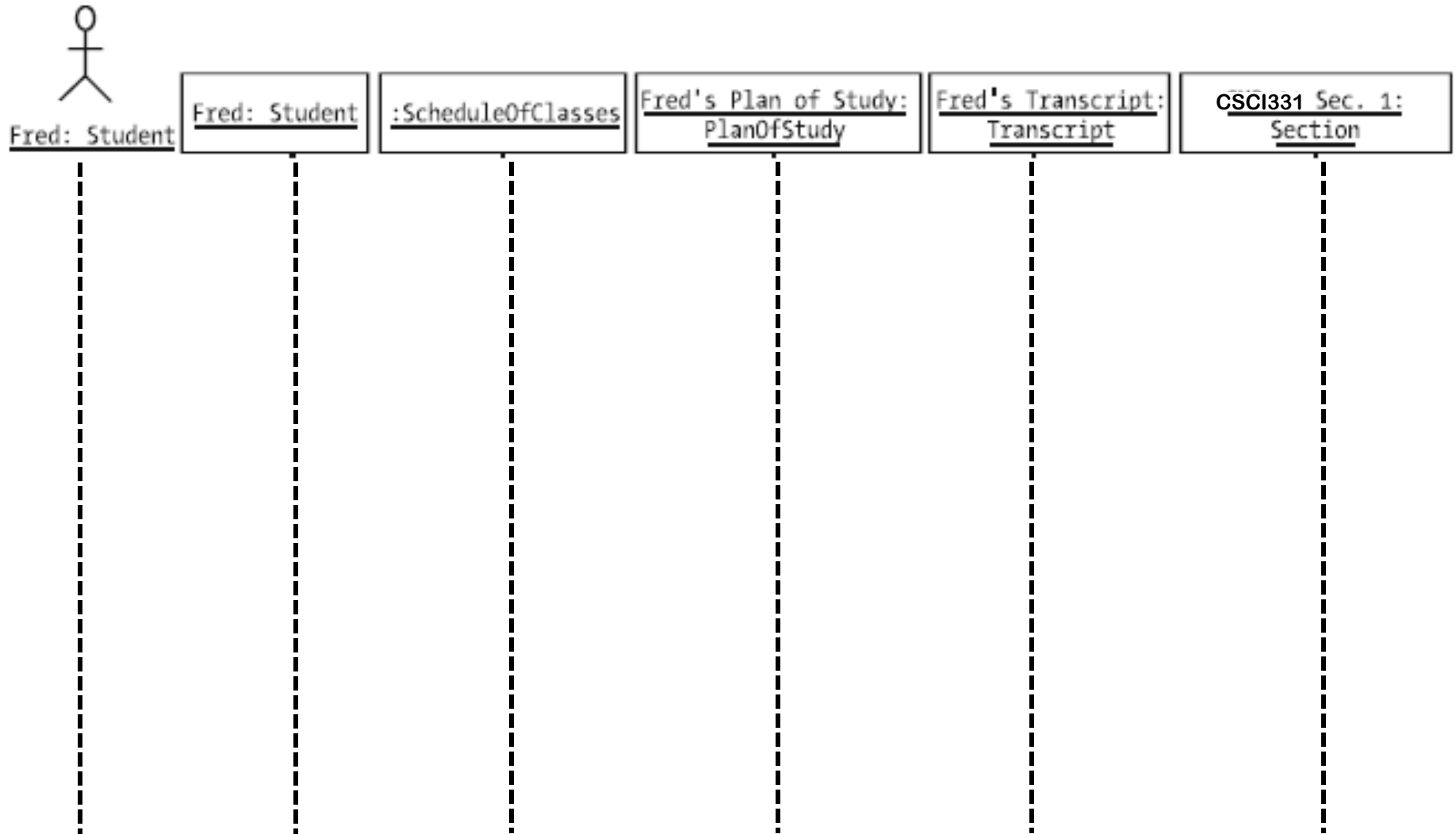


... and replace it with an
association (aggregation) to a
class that *encapsulates* this
information.

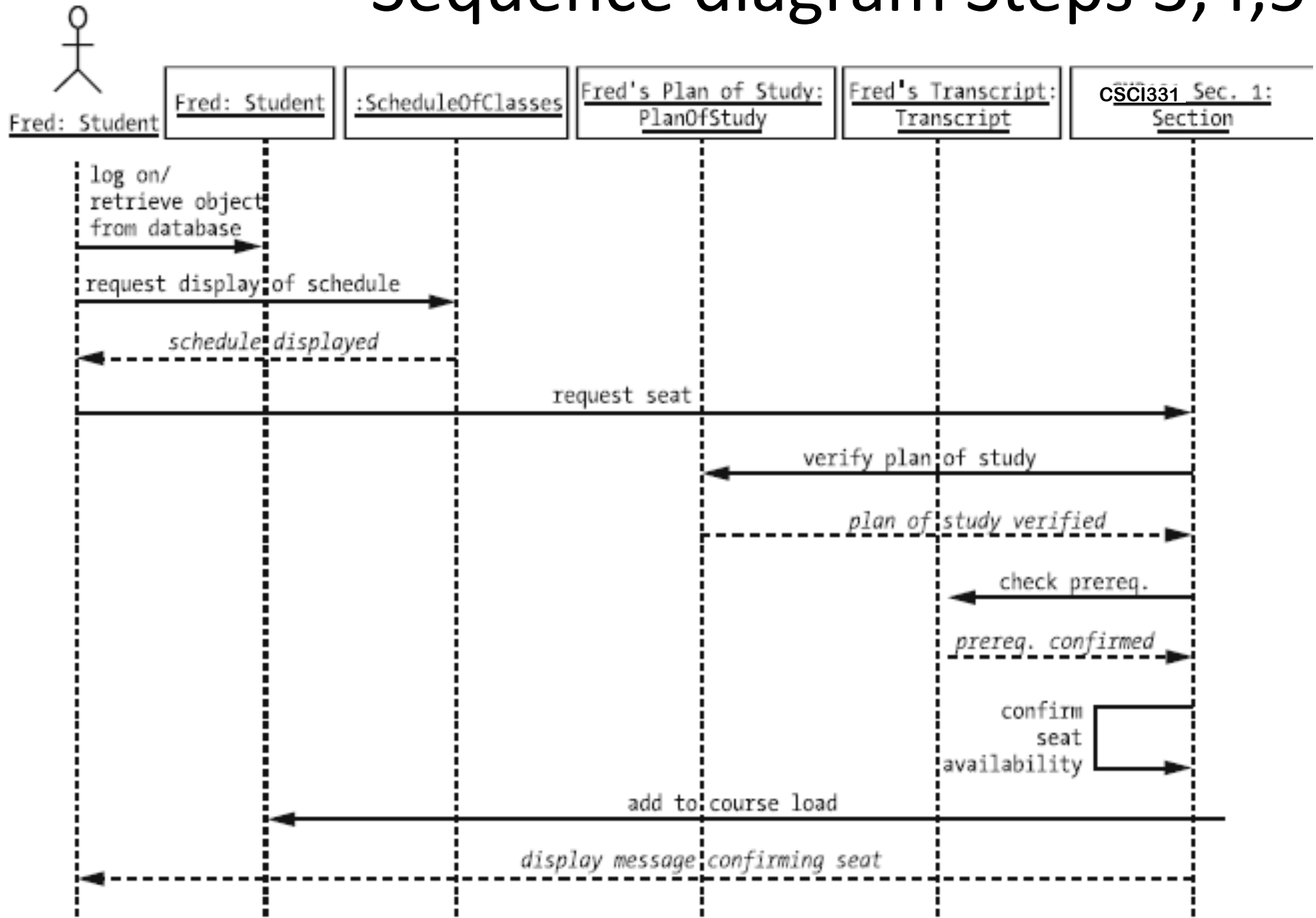
Sequence diagram Step 2. Arrange participating **objects** in a header row



Sequence diagram Step 2. Draw life lines



Sequence diagram Steps 3,4,5



Deduce object methods from sequence diagrams

We step through the diagram, one lifeline at a time, and study all incoming and outgoing lines:

- Arrows representing a new request being made of an object—**solid-line incoming arrows**—signal **methods** that the receiving object must be able to perform
- Arrows representing responses from an operation that some other object has performed—**dashed-line arrows**—hint at the **return type** of the method from which this response is being issued

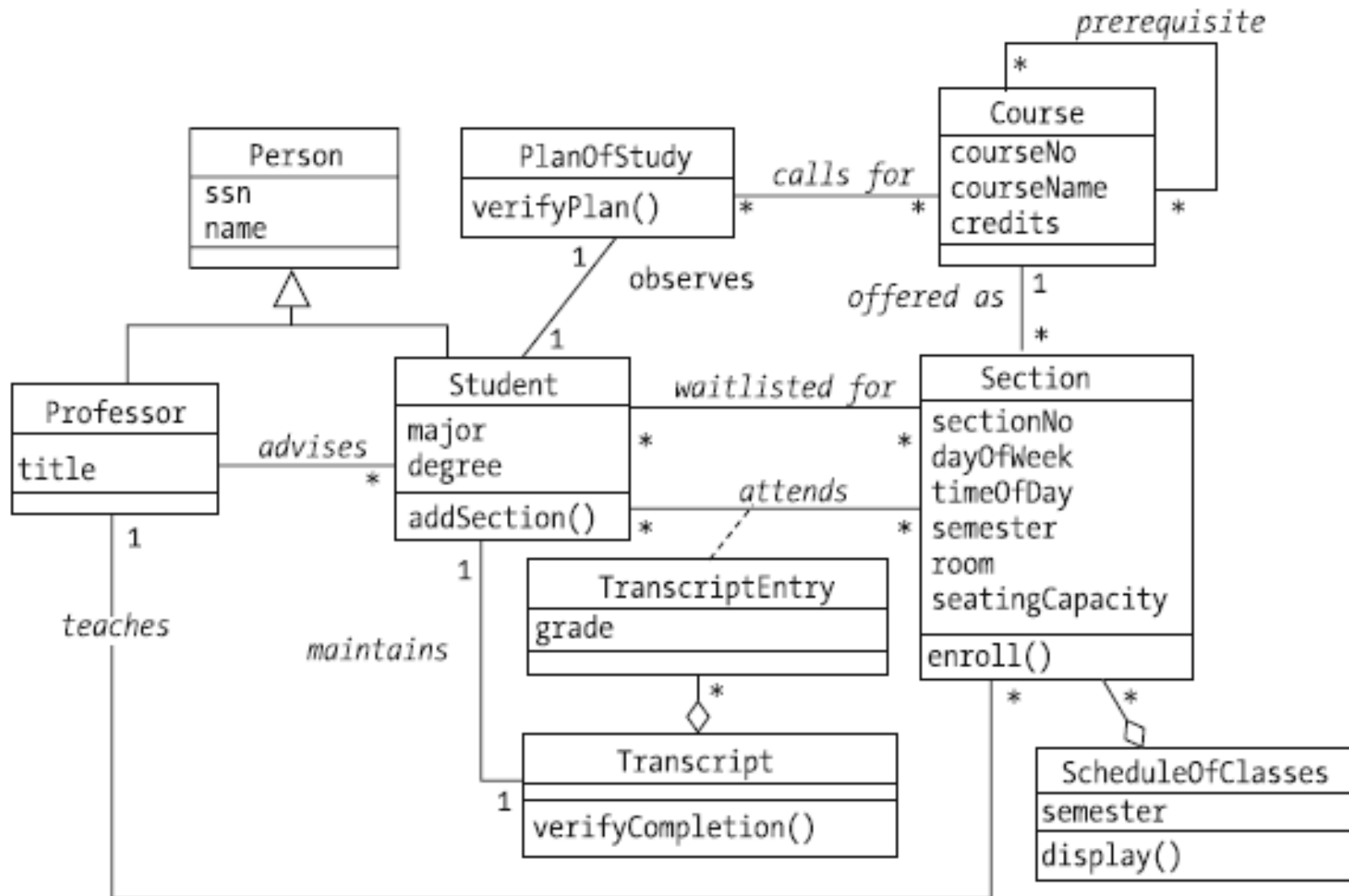
Methods collected from a sequence diagram for Scenario #1

Arrow label	IN / OUT	Points to class X	Method to be added to class X
Log on	IN	Student	Constructor , loading data from storage
Display Schedule of classes	IN	ScheduleOfClasses	Collection getAllScheduledClasses()
Schedule displayed	OUT	Student (actor)	GUI (not model) method
Request seat	IN	Section	boolean enroll(Student s)
Verify plan of study	IN	PlanOfStudy	boolean verifyPlan(Course c)
Plan of study verified	IN	Section	-
Check prerequisite (s)	IN	Transcript	boolean verifyCompletion(Course c)
Prerequisite confirmed	OUT	Section	-
Check seat availability	IN	Section	private confirmSeatAvailability()
Add to course load	IN	Student	addSection(Section s)
Display confirmation message	OUT	Student (actor)	GUI (not model) method

Now you can assign class implementation to team members

Class	Required public methods
Student	Constructor , loading data from storage addSection (Section s)
ScheduleOfClasses	Collection getAllScheduledClasses()
Section	boolean enroll(Student s) Consists of: <ul style="list-style-type: none">• s.planOfStudy.verifyPlan (this.course)• s.transcript.verifyCompletion(this.course.prerequisites)
PlanOfStudy	boolean verifyPlan(Course c)
Transcript	boolean verifyCompletion(Course c)
Section	private confirmSeatAvailability()

Final UML class diagram of a SRS system



System model

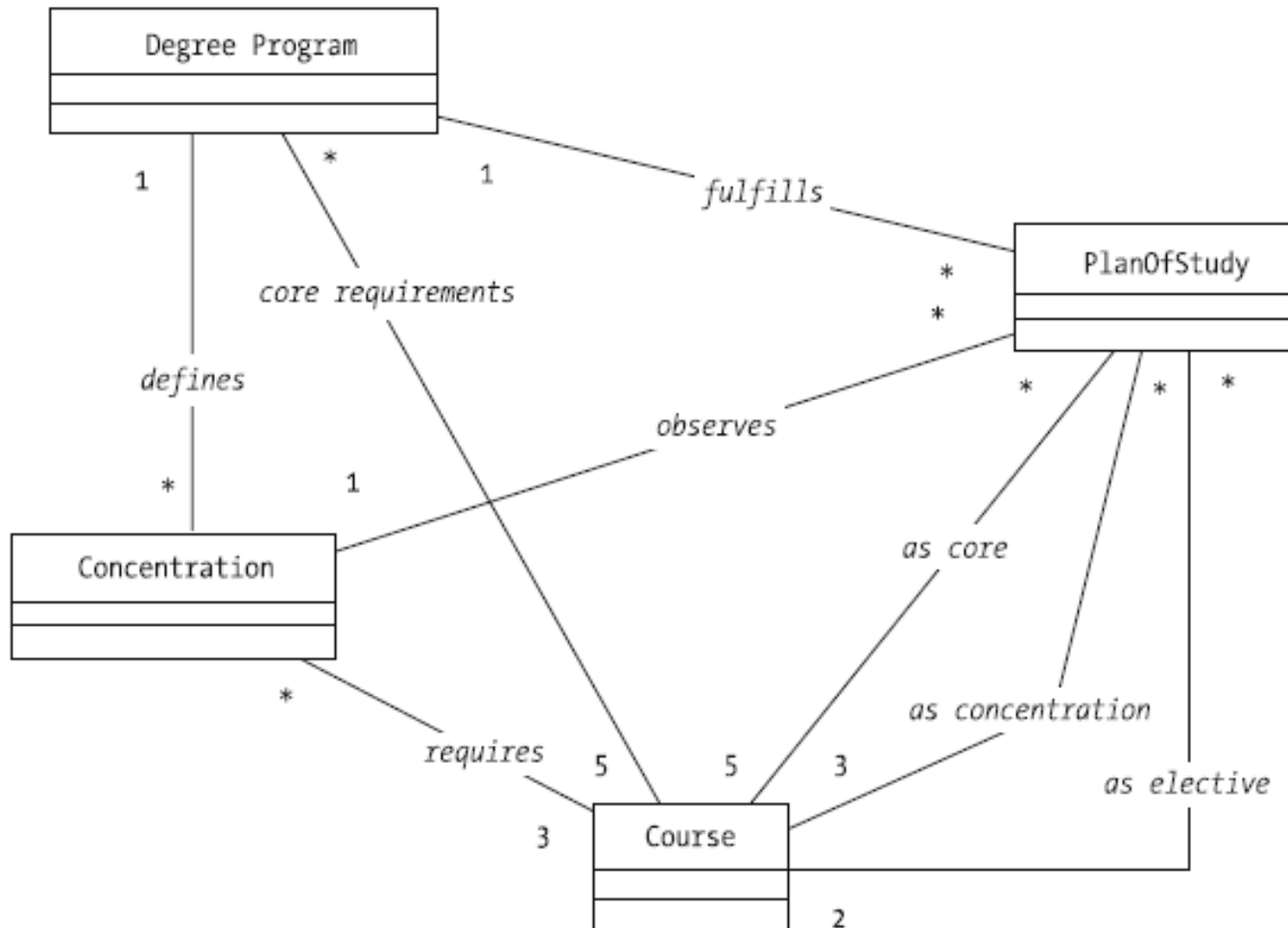
- Problem statement (requirement document)
- Supporting Use-cases: general Use case diagram with actors, high-level use cases broken down to detailed use cases
- Sequence diagrams for all important scenarios
- UML class diagram

Testing the model

- Revisit if requirements match the final model: to see if something is missing
- At least two formal walk-throughs with software development team and with clients

We overlooked one sentence

The SRS will verify whether or not the proposed plan of study satisfies the requirements of the degree that the student is seeking.



New project

We are asked to design system for Online travel reservations for small travel agency.

They decide to enable their customers to make travel reservations online via the Web (most of their competitors take such requests over the phone).

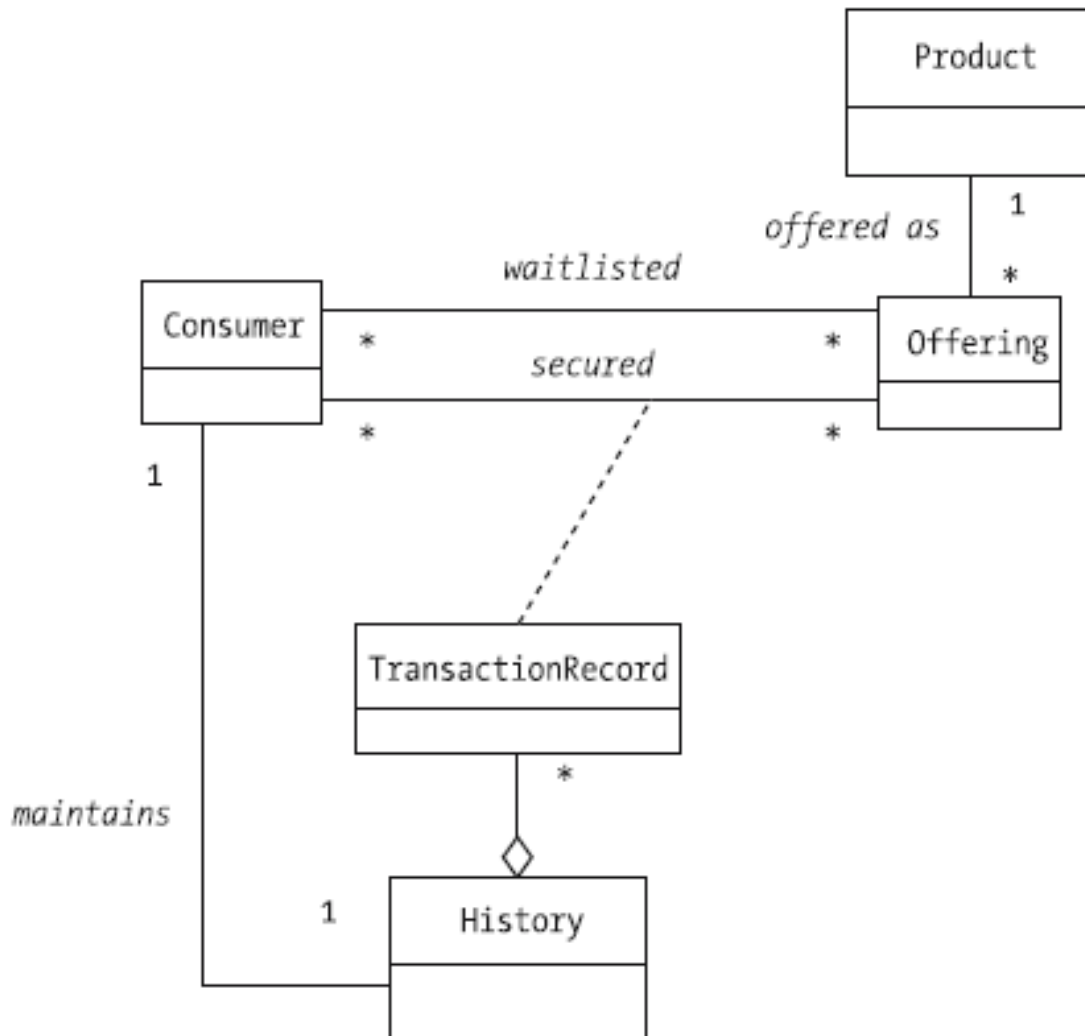
Travel agency system requirements

- For any given travel package—let's say a ten-day trip to Ireland—WBY offers numerous trips throughout the year. Each trip has a maximum client capacity, so if a client can't get a confirmed seat for one of the trips, he or she may request a position on a first-come, first-served wait list.
- In order to keep track of each client's overall experience with WBY, the travel agency plans on following up with each client after a trip to conduct a satisfaction survey, and will ask the client to rate his or her experience for that trip on a scale of 1 to 10, with 10 being outstanding.
- By doing so, WBY can determine which trips are the most successful, so as to offer them more frequently in the future, as well as perhaps eliminating those that are less popular. WBY will also be able to make more informed recommendations for future trips that a given client is likely to enjoy by studying that client's travel satisfaction history.

This looks very familiar if we

- Substitute TravelPackage for Course
- Substitute Trip for Section
- Substitute Client for Student
- Substitute TripRecord for TranscriptEntry
- Substitute TravelHistory for Transcript

A general-purpose class diagram for reservation system



Successful designs should be reused,
and not built from scratch