Object Oriented Analysis

Chapter 05

Introduction

- ✓ Requirement Gathering and analysis are highly interrelated and iterative
- ✓ Essential models evolve into corresponding analysis artifacts
- ✓ Use Case Models – how your users work with your system
- ✓ Sequence Diagrams & activity diagrams – to flesh out and verify the logic contained in your use cases

System Use Case Modeling

- Difference from essential use case modeling
  - High level implementation decisions
- Composed of use case diagrams and documentation describing the use cases, actors and associations
- Describes a sequence of actions that provide a measurable value to an actor and is drawn as a horizontal ellipse
- An actor is a person, organization or external system that plays a role in one or more interactions with your systems
  - Drawn as stick figures
- Associations between actors and classes are indicated in use case diagrams, a relationship exists whenever an actor is involved with an interaction described by a use case
  - Exist between use cases
  - Depicted as a line (arrows are optional)
- System Boundary Box – Rectangle around the use cases

Writing System Use Cases

- Begin with your essential use cases and modify them to reflect the information captured within your UML sequence diagrams, activity diagrams and user interface diagrams.
- **Rework** your use cases to reflect opportunities for reuse applying the UML stereotypes of <<extend>> and <<include>>
- Two common styles exist for writing use cases
  - **Narrative style**
    - Basic and alternate courses of actions are writing one at a time
  - **Action Response Style**
    - Advantage: it is easier to see how actors interact with the system and how the system responds
    - Disadvantage: it is harder to understand the flow of logic of the use case

Action Response Style

STUDENT

1. The student wants to enroll in a seminar
2. The student inputs his name and student number into the system via some rule
5. The student indicates the seminar in which e wants to enroll

SYSTEM

3. The system verifies the student is eligible to enroll in seminars at the university, according to business rule
4. the system displays seminar selection screen" which indicates the list of available seminars
6. The system validates the student is eligible to enroll in the seminar
7. The system validates the seminar fits into the existing schedule of the student
8. The system calculates the fees based on the fee published in the course catalog applicable student fees and applicable taxes

Reuse in Use Case Models

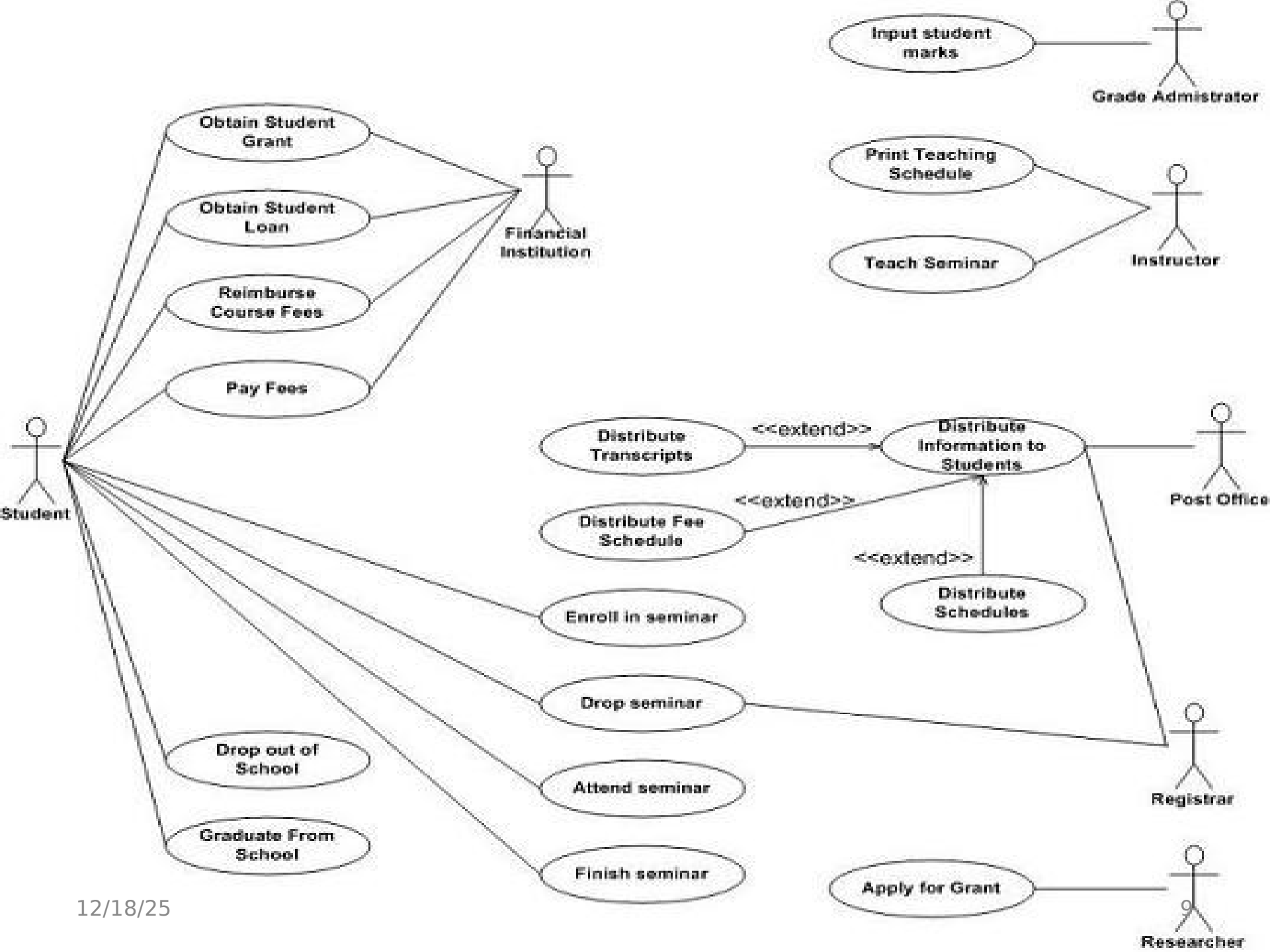
<<extend>>, <<include>>
and inheritance

Extend Associations between use cases

- The <<extend>> stereotype is used to indicate an extend association
- It is a generalization relationship where an extending use case continues the behavior of a base use case
- Accomplishes this by conceptually inserting additional action sequences into the base use case sequence
- When an extending use case activity sequence is completed the base use case continues
- Alternate course of the base use case
- Introduce an extending use case when
  - The logic for the alternate course of action is at a complexity level similar to that of the basic course of action
  - There is a need for an alternative course for an alternative course of action
- Extension points are placed in based use cases to indicate where the logic of the extending use case replaces that of the base use case
- One use case extends the bhr of another use case at a specific point. Example: make repeated payment and make one payment extend make payments use case.

Include Associations between Use Cases

- Generalization relationship denoting the inclusion of the behavior described by another user case
- Invocation of a use case by another one
- Denoted by the <<include>> stereotype
- One use case includes the behavior of another use case at specific point.
- Example: make payment includes check funds availability.


Input student marks
Grade Administrator
Obtain Student Grant
Obtain Student Loan
Reimburse Course Fees
Pay Fees
Financial Institution
Print Teaching Schedule
Teach Seminar
Instructor
Student
Distribute Transcripts
<<extend>>
Distribute Information to Students
Post Office
Distribute Fee Schedule
<<extend>>
<<extend>>
Distribute Schedules
Enroll in seminar
Drop seminar
Drop out of School
Attend seminar
Graduate From School
Finish seminar
Registrar
Apply for Grant
Researcher

Use Case Modeling Tips and Techniques

- Write from the point of view of the actor in the active voice
- Write scenario text, not functional requirements
- Use case is neither a class specification nor a data specification
- Don’t forget the user interface
- Create a use case template
- Organize you use case diagrams consistently
- Don’t forget the system response to the actions of actors
- Alternate courses of action are important
- Don’t get hung up on <<include>> and <<extend>> associations
- Use cases drive user documentation
- Use cases drive presentations

Sequence Diagrams

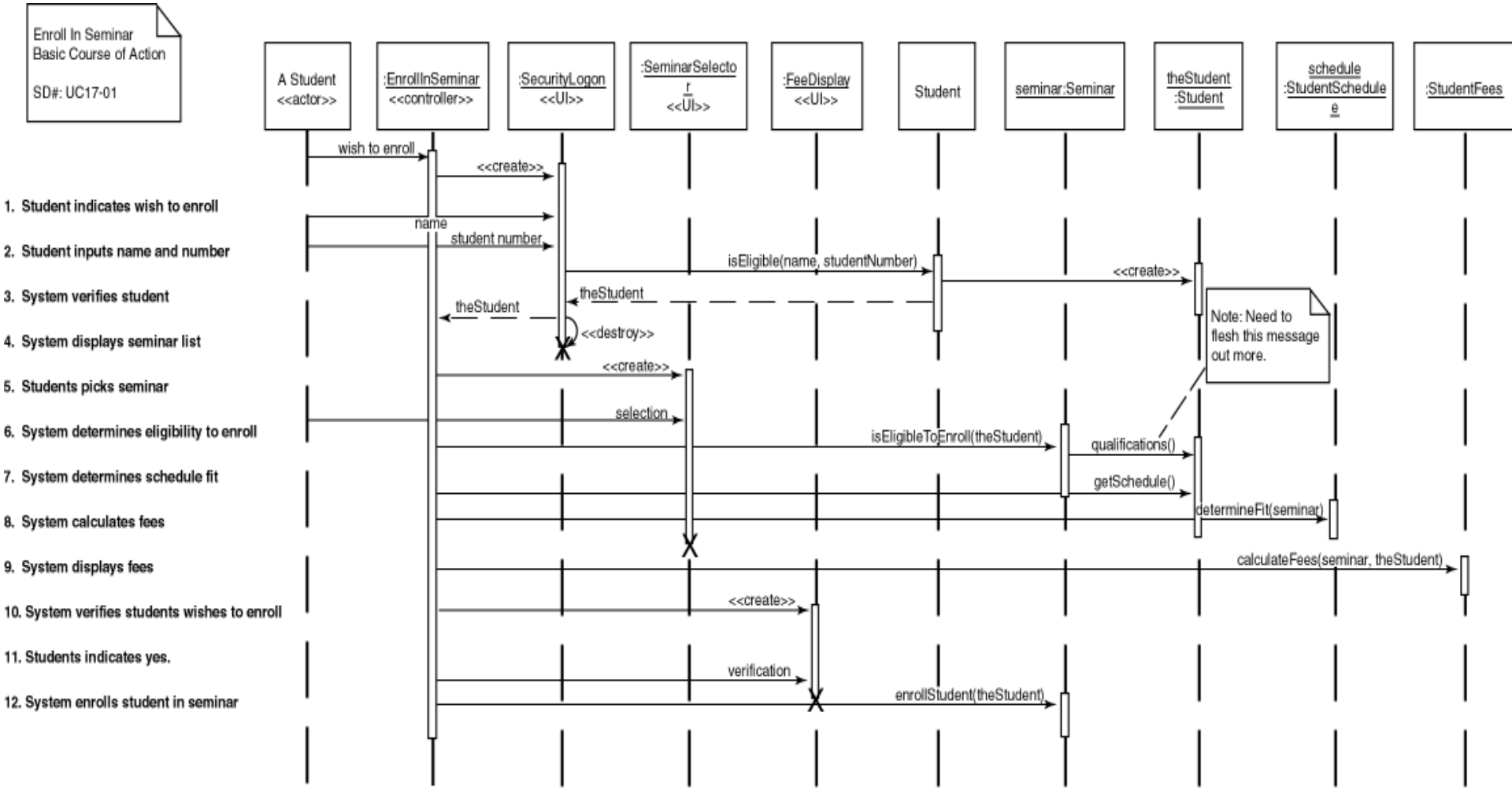
- Used to model the logic of usage **scenarios**
- A description of a potential way your system is used
- One entire path through a use case
- Model the flow of logic within your system in a visual manner
- The boxes across the top of the diagram represent classifiers or their instances, typically uses cases, objects, classes or actors
- Because you can send messages to both objects and classes, objects respond to messages through the invocation of an operation, and classes do so through the invocation of static operations, it makes sense to include both on sequence diagrams
- Because actors initiate and take an active part in usage scenarios, they are also included in sequence diagrams

Notations in sequence diagrams

- **Objects** – format – “name: Classname”
- **Classes** – format – “ClassName”
- **Actors** – format – “ActorName”, <<actor>>
- **User Interface elements** - <<UI>>
- **Dashed lines** – object lifelines
- **Long thin boxes** – method invocation boxes indicating that processing is being performed by the target object/class to fulfill a message
- **X at the bottom of a method invocation box** – object have been removed
- **Messages** – labeled arrows
- When the source and target of a message is an object or class the label is the signature of the method invoked in response to the message
- If either the source or target is a human actor, then the message is labeled with brief text describing the information being communicated

How to draw sequence diagrams

- Identify the scope of the sequence diagram
- List the use case steps down the left hand side
- Introduce boxes for each actor
- Introduce controller classes
- Introduce a box for each major UI element
- Introduce a box for each included use case
- Identify appropriate messages for each use case step
- Add a method – invocation box for each invocation of a method
- Add destruction messages where appropriate
- Add your business classes and objects
- Update your class model
- Update your user interface model
- Update your use case model


Enroll In Seminar
Basic Course of Action
SD#: UC17-01
A Student
<<actor>>
:EnrollInSeminar
<<controller>>
:SecurityLogon
<<UI>>
:SeminarSelector
<<UI>>
:FeeDisplay
<<UI>>
Student
seminar:Seminar
theStudent
:Student
schedule
:StudentSchedule
:StudentFees
1. Student indicates wish to enroll
2. Student inputs name and number
3. System verifies student
4. System displays seminar list
5. Students picks seminar
6. System determines eligibility to enroll
7. System determines schedule fit
8. System calculates fees
9. System displays fees
10. System verifies students wishes to enroll
11. Students indicates yes.
12. System enrolls student in seminar
wish to enroll
<<create>>
name
student number
isEligible(name, studentNumber)
<<create>>
theStudent
theStudent
<<destroy>>
Note: Need to flesh this message out more.
<<create>>
selection
isEligibleToEnroll(theStudent)
qualifications()
getSchedule()
determineFit(seminar)
calculateFees(seminar, theStudent)
<<create>>
verification
enrollStudent(theStudent)

Why and When should you draw sequence diagrams

- Sequence diagrams are a great ways to validate and flesh out your logic
- To document your design, at least from the point of view of use cases
- Great mechanism for detecting bottlenecks in your design.
- Gives a feel for which classes in your application are going to be complex

Conceptual Modeling: Class Diagrams

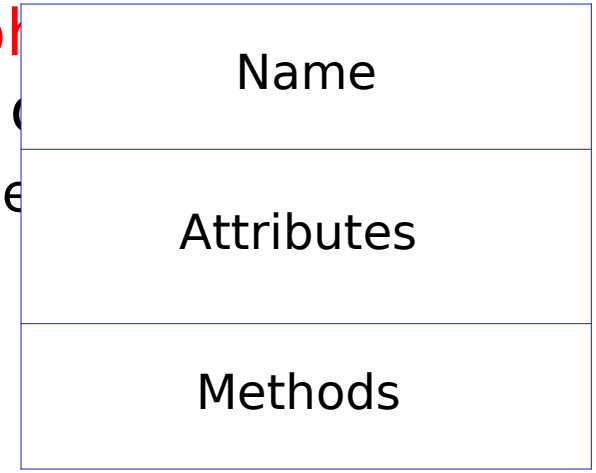
- The mainstay of OOSAD
- Uses many modeling concepts discussed in the previous chapter
- Shows the class, interrelationship (including inheritance, aggregation and association), and the operations and attributes of the classes
- Start by putting the Domain Model (CRC) as a base
- While CRC modeling provides an excellent overview of a system, it doesn't provide the details needed to actually build it

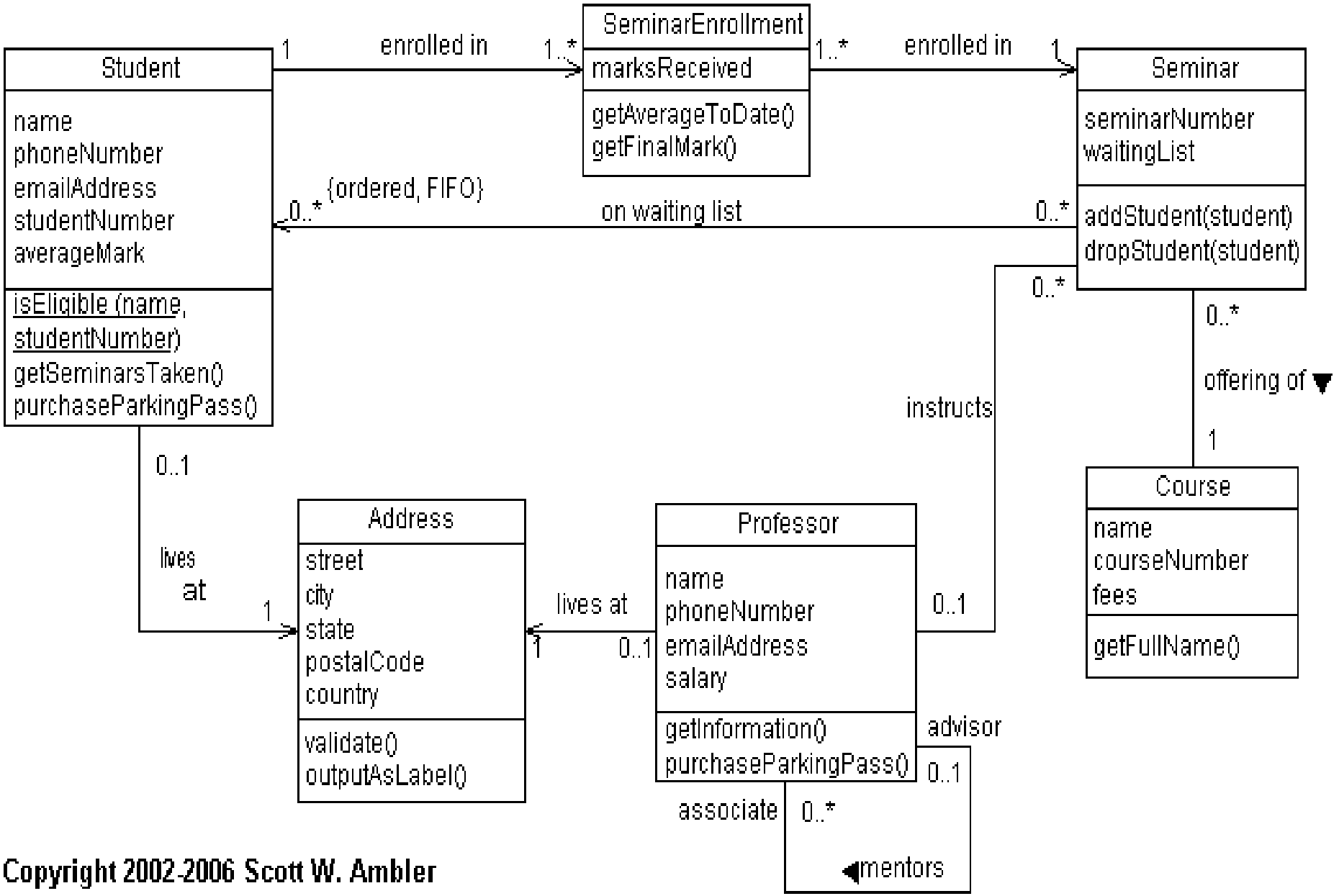
Conceptual Modeling

- For each CRC model create a concrete class in the class diagram, with the exception of cards that represent actors
- Collaborations from a user interface class implies a dependency, whereas collaborations from business domain classes imply either association or aggregation between the classes
- Responsibilities are modeled as attributes and methods
- Modeling user interface classes on class diagrams often adds a lot of clutter without adding much useful information

Modeling Classes, Methods and Attributes

- Representation of an object, template of which objects are created
- Modeled as rectangle with three sections
- Model your classes to an **appropriate level of detail**
  - Eg. Take out address as a class
- **Class normalization** – refactoring the behavior of classes to increase their coh
  reduce the coupling between c
  - Semester and Semester enrollme





Copyright 2002-2006 Scott W. Ambler

Modeling Associations

- Objects are often associated with other objects
- Modeled as lines connecting the two classes whose instances are involved in the relationship
- Identifying the multiplicities of an associations is an important part of modeling it
  - Cardinality and optionality
- Indicate the direction in which the label should be read
  - When it is not clear which way a label should be read
- Roles may also be indicated only when the information adds value
  - It is not clear from the association label what the roles are
  - If there is recursive association
  - if there are several associations between two classes

Modeling Dependencies

- Are used to model transitory associations
  - When one of the two classes is not persistent
- UI classes
- Dashed arrow
- Transitory classes are not three sectioned boxes
- Only indicate name of the class



Documenting Class Models

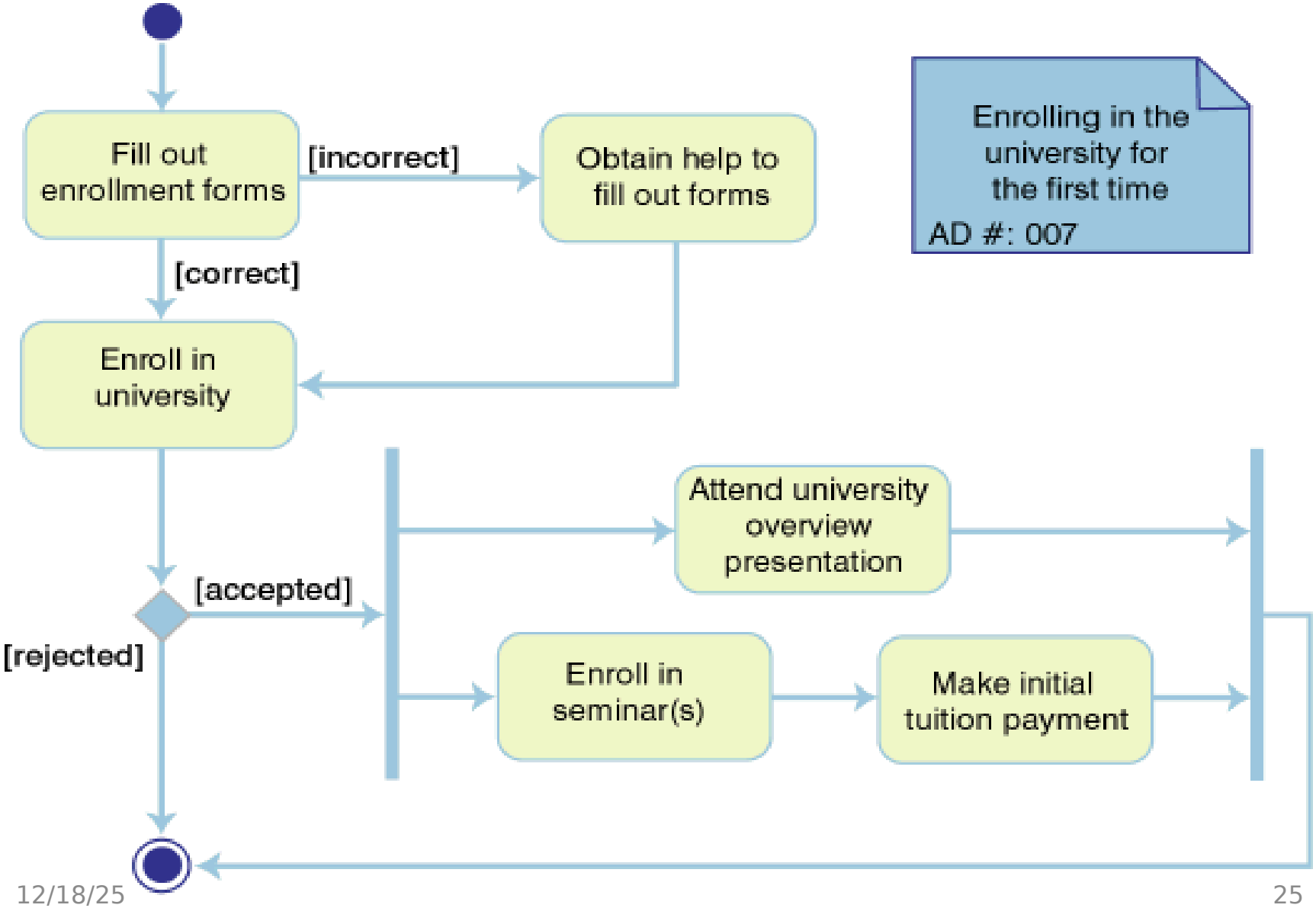
- Classes – purpose, persistent/transitory, aliases, references
- Attributes – type, example, range, references
- Methods – pseudo codes, parameters, return values, preconditions, post conditions, reference
- Inheritance – why?
- Association – label, multiplicities, roles, reference
- Aggregation and composition – same as association

Conceptual Class Modeling Tips

- You don’t have to get it perfect at the start
- Start at your domain model
- Evolve your class diagram via sequence diagrams
- Focus on the problem space
- Focus on fulfilling the requirements first
- Use meaningful names
- Perform object oriented analysis
- Understand and effectively apply analysis patterns
- Class model in parallel with user interface prototyping

Activity Diagramming

- Used to model the logic of a
  - single operation/method
  - Single use case
  - Flow of logic of a business process
- Equivalent to DFDs in structured development
- Starting point = filled circle
- Ending point = filled circle with a border
- Processes = Rounded rectangle
- Decision point = Diamond
- Transitions between activities = Arrows
- Start and End of Parallel processes = Thick bars


Fill out enrollment forms
[incorrect]
Obtain help to fill out forms
[correct]
Enroll in university
[accepted]
[rejected]
Attend university overview presentation
Enroll in seminar(s)
Make initial tuition payment
Enrolling in the university for the first time
AD #: 007

How to draw activity diagrams

- Identify the scope of the activity diagrams
- Add start and end points
- Add activities
- Add transitions from the activities
- Add decision points
- Identify opportunities for parallel activities
- Document activity diagrams

User Interface Prototyping

- Iterative analysis technique in which users are actively involved in the making-up of the UI for a system
- 2 purposes
  - Explore the problem space your system addresses
  - Explore the solution space of your system
- 4 high level steps in UI prototyping
  - Determine Needs
  - Build Prototype
  - Evaluate Prototype
  - Determine if you are finished

Determine the needs of Users

- Evolve all or part of your essential user interface prototype
- Convert your hand drawings, flip chart paper and sticky notes into something a little more substantial
- Begin this process by making platform decisions
  - Browser or GUI, Java, mainframe based
- Update other models as you proceed with UI prototyping

Building the Prototype

- Use prototyping tool or high level language
  - Screen pages
  - Reports
- Don’t invest a lot of time in making the code “good” because chances are high you will scrap large portions of your prototype code when portions or all of your prototype fail the evaluation

Evaluating the prototype

- Evaluation is done by your SMEs to verify that it meets their needs
- Three basic questions
  - What is good about the UI prototype?
  - What is bad about the UI prototype?
  - What is missing from the UI prototype?
- 4th step
  - Scrap parts of it, modify parts, and even add brand new parts
  - Stop this process when it is no longer generating new ideas

Prototyping Tips and Techniques

- Work with the real users
- Use Prototyping tool
- Get your SMEs to work with the prototype
- Understand the underlying business
- Don’t spend a lot of time making the code good
- Only prototype features that you can actually build
- Get an interface expert to help you design it
- Explain what a prototype is
- Avoid implementation decisions as long as possible

End of this chapter