What is UML

- Unified Modeling Language
- A language for modelling software systems from requirements to specification
- The goal is to become a common language for creating models of object oriented computer software

Benefits of UML

- You know exactly what you are getting
- You will have lower development costs
- Your software will behave as you expect it to. Fewer surprises
- The right decisions are made before you are given poorly written code. Less overall costs
- We can develop more memory and processor efficient systems
- System maintenance costs will be lower. Less relearning takes place
- Working with a new developer will be easier.
- Communication with programmers and outside contractors will be more efficient

Types of UML Diagrams

- **Use Case Diagram**
  - Description of a system's behavior from a user's point of view
- **Class Diagram**
  - Models class structure and contents using design elements such as classes, packages, and objects
  - Displays relationships such as containment and inheritance

Types of UML Diagrams

- **Sequence Diagram**
  - Shows the time-based dynamics of the interaction between objects
  - Two dimensions; time and different objects
- **Collaboration Diagram**
  - Displays the interaction organized around the objects and their links to one another
  - Numbers are used to show the sequence of messages

Types of UML Diagrams

- **State Diagram**
  - Displays the sequences of states that an object of an interaction goes through during its life in response to received stimuli
- **Activity Diagram**
  - Displays a special state diagram where most of the states are action states and most of the transitions are triggered by completion of the actions in the source states
  - Like a flowchart

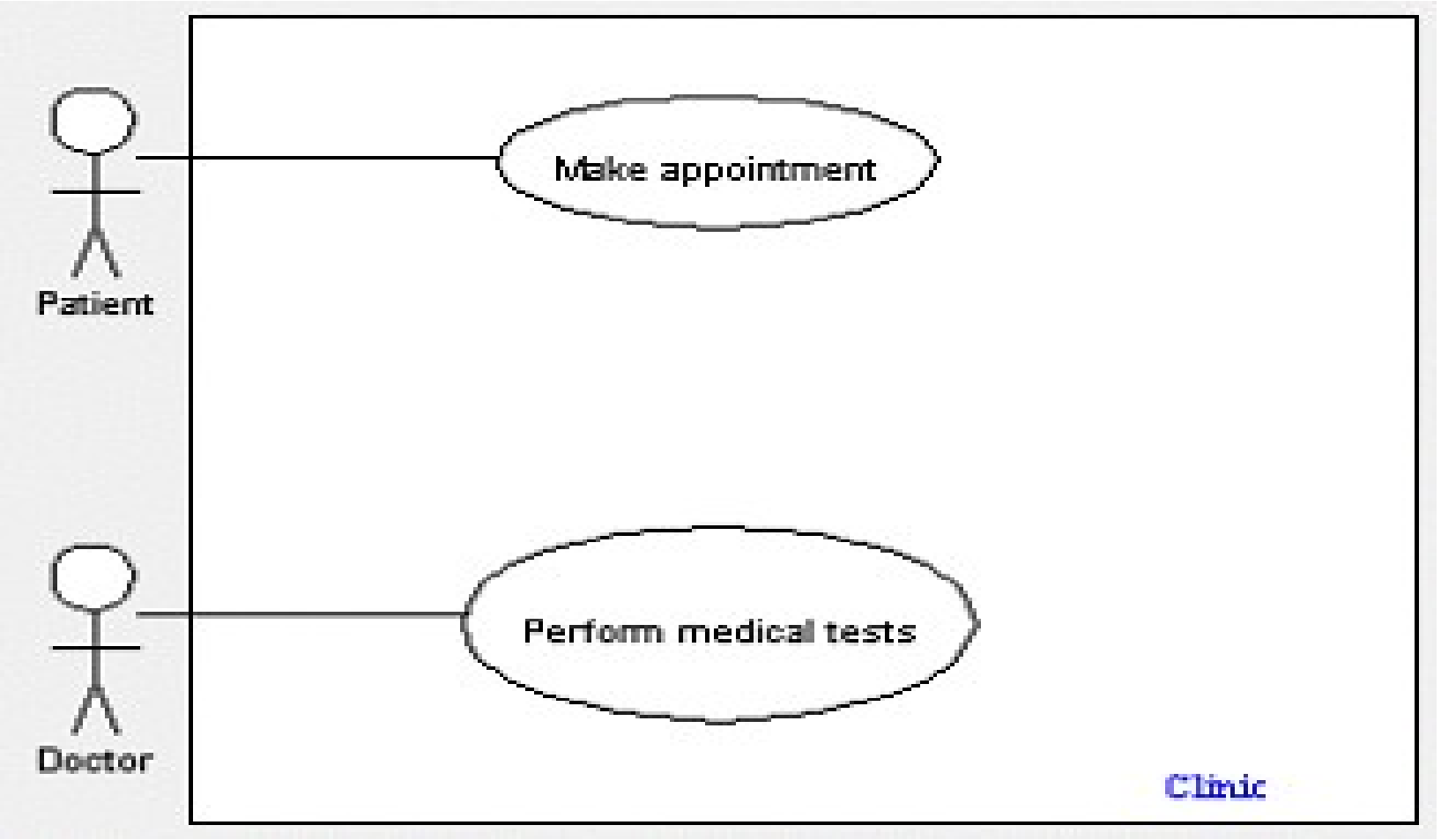
Types of UML Diagrams

- **Component Diagram**
  - Displays the high level packaged structure of the code itself
  - Dependencies among components are shown, including source code, binary code, and executable components
- **Deployment Diagram**
  - Displays the configuration of run-time processing elements and the software components, processes, and objects that live on them

Use Case Diagrams

- Use case diagrams show how a system's users interact with it
  - i.e. the system's requirements
- Use case diagrams represent:
  - **Actors:** things (often people) outside the system that interact with it
  - **Use cases:** tasks the system supports
  - **Associations** between the two

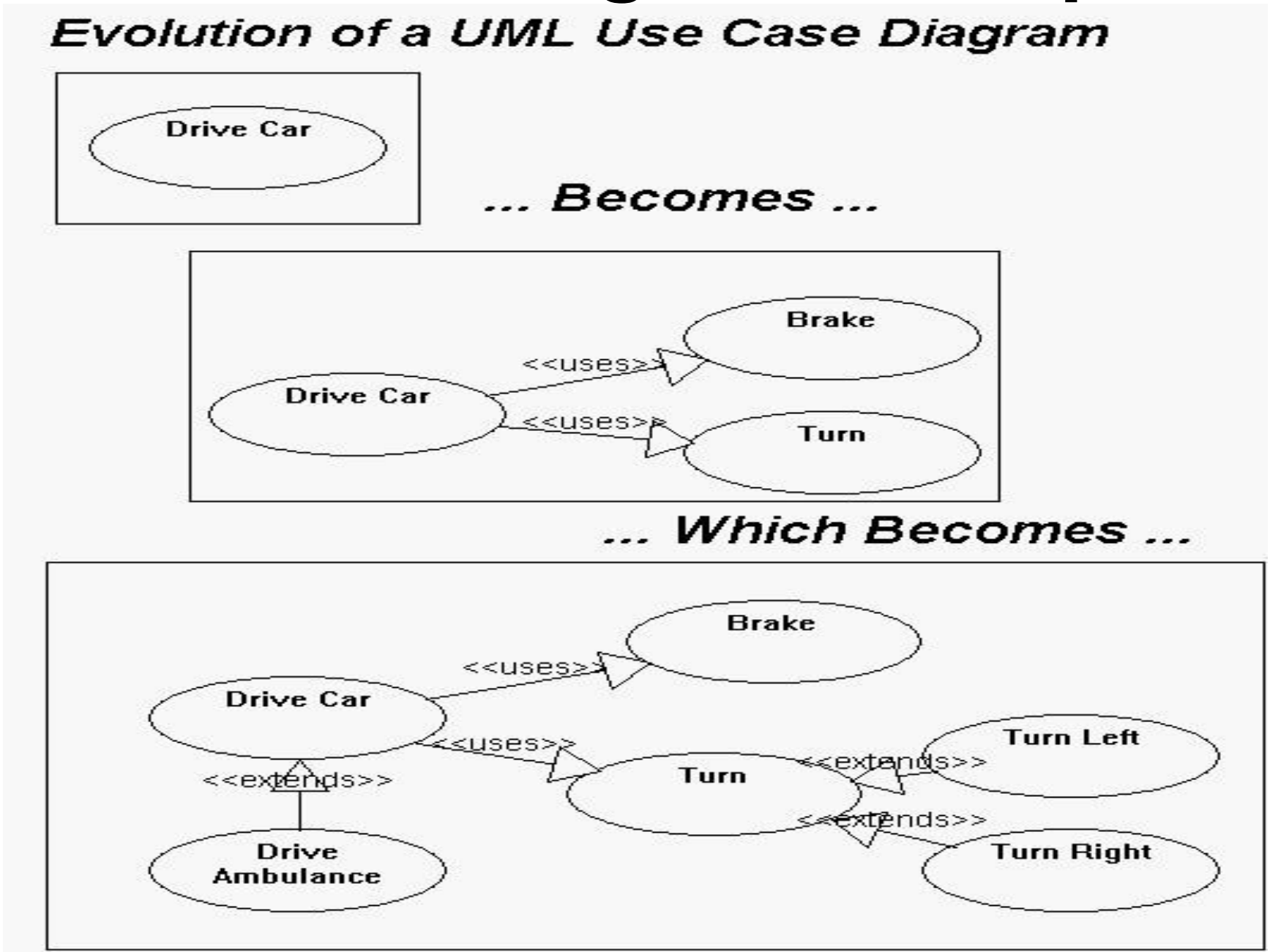
Use Case Diagram Example

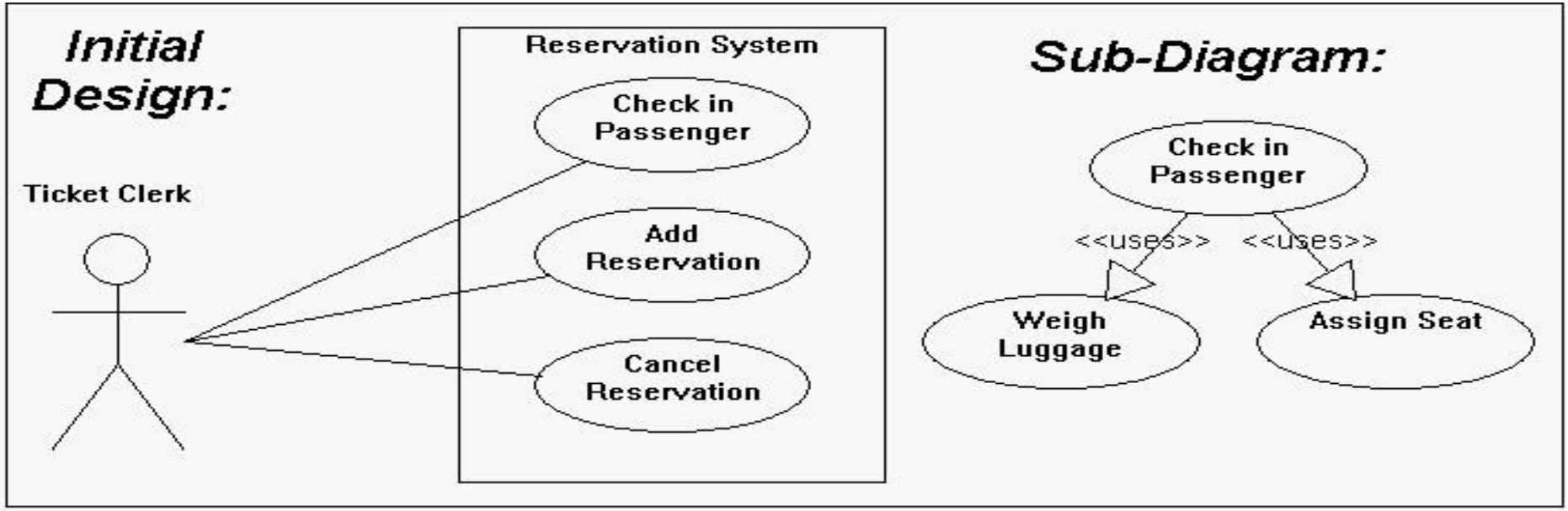


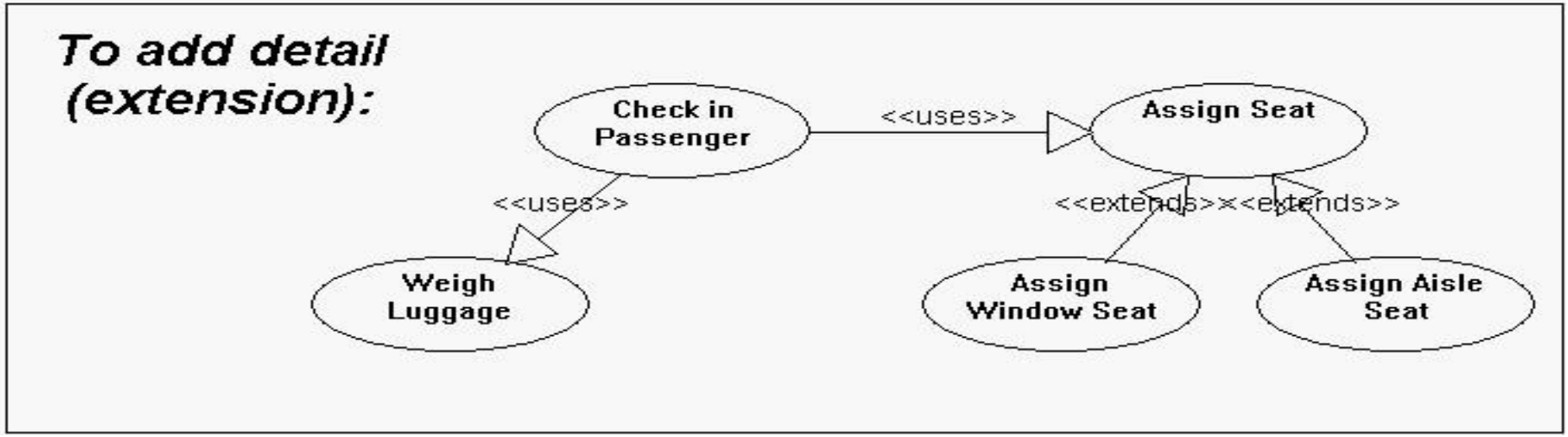
Use Case Diagrams

- Used in almost every project
- Helpful in exposing requirements analysis and planning the project
- During the initial stage of a project most use cases should be defined, but as the project continues more might become visible

Use Case Diagram Example




Initial Design:
Reservation System
Check in Passenger
Add Reservation
Cancel Reservation
Ticket Clerk
Sub-Diagram:
Check in Passenger
<<uses>>
<<uses>>
Weigh Luggage
Assign Seat

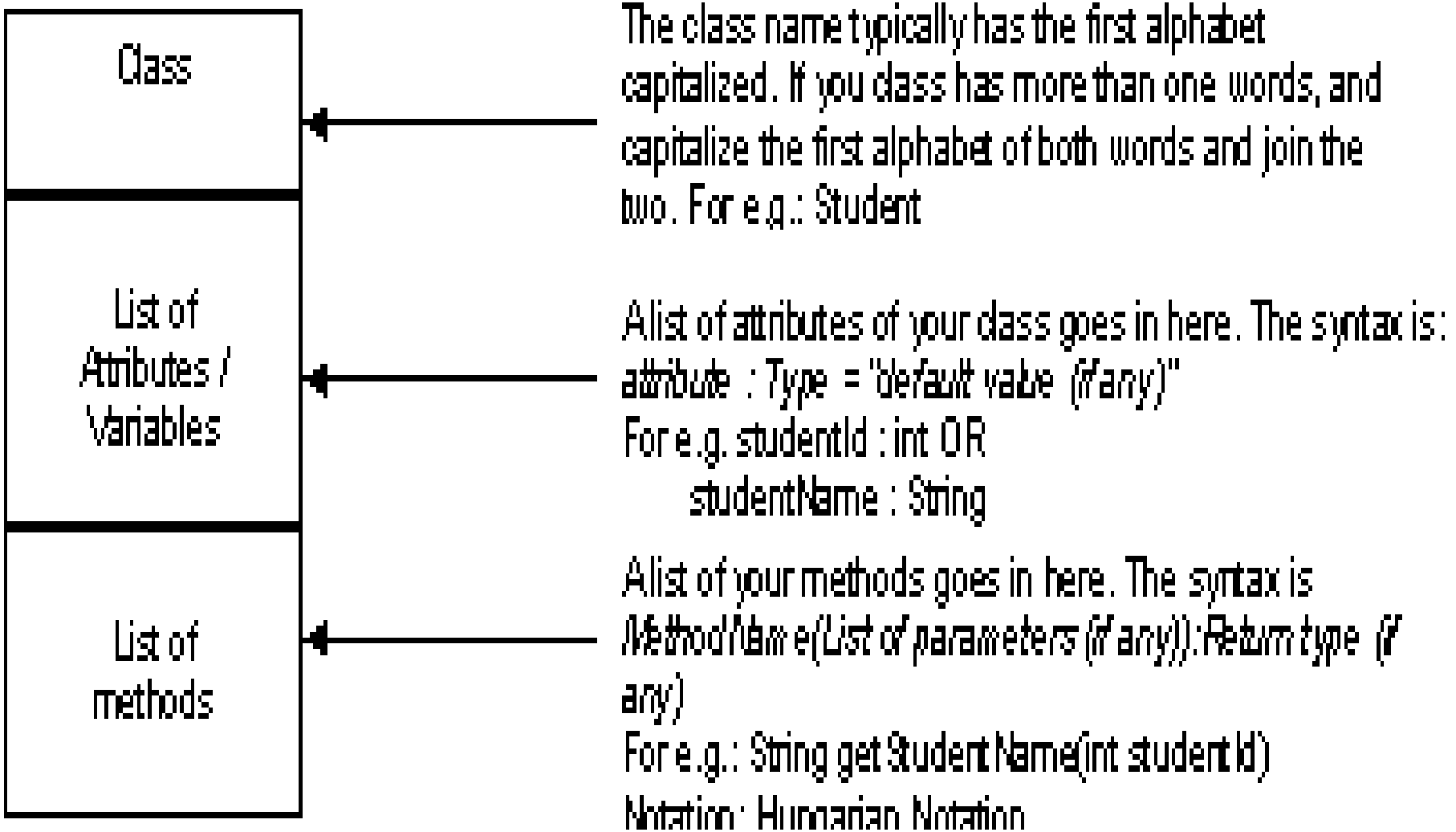

To add detail (extension):
Check in Passenger
<<uses>>
Assign Seat
<<uses>>
<<extends>>
<<extends>>
Weigh Luggage
Assign Window Seat
Assign Aisle Seat

Class Diagrams

- Widely used to describe the types of objects in a system and their relationships
- Model class structure and contents using design elements such as classes, packages and objects
- Describe three different perspectives when designing a system; conceptual, specification, and implementation

Class Diagrams

- Classes are composed of three components:



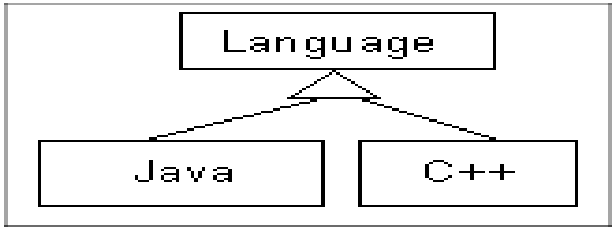
Class Diagrams

- Used in nearly all Object Oriented software designs

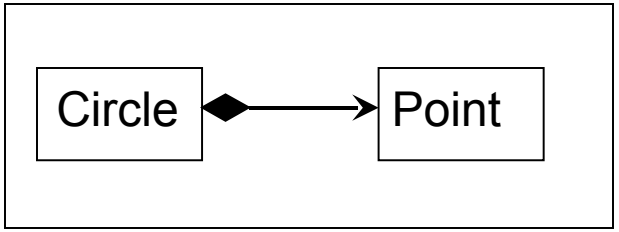
- Used to descibe the classes of the system and their relationships to each other

Class Diagrams

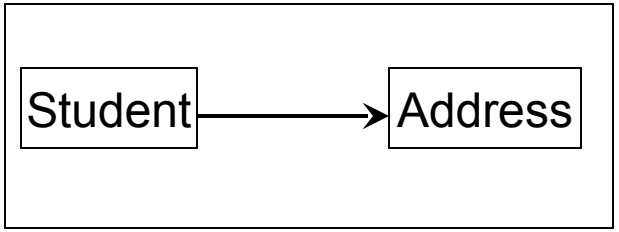
- **Relationships Between Classes**
  - Inheritance



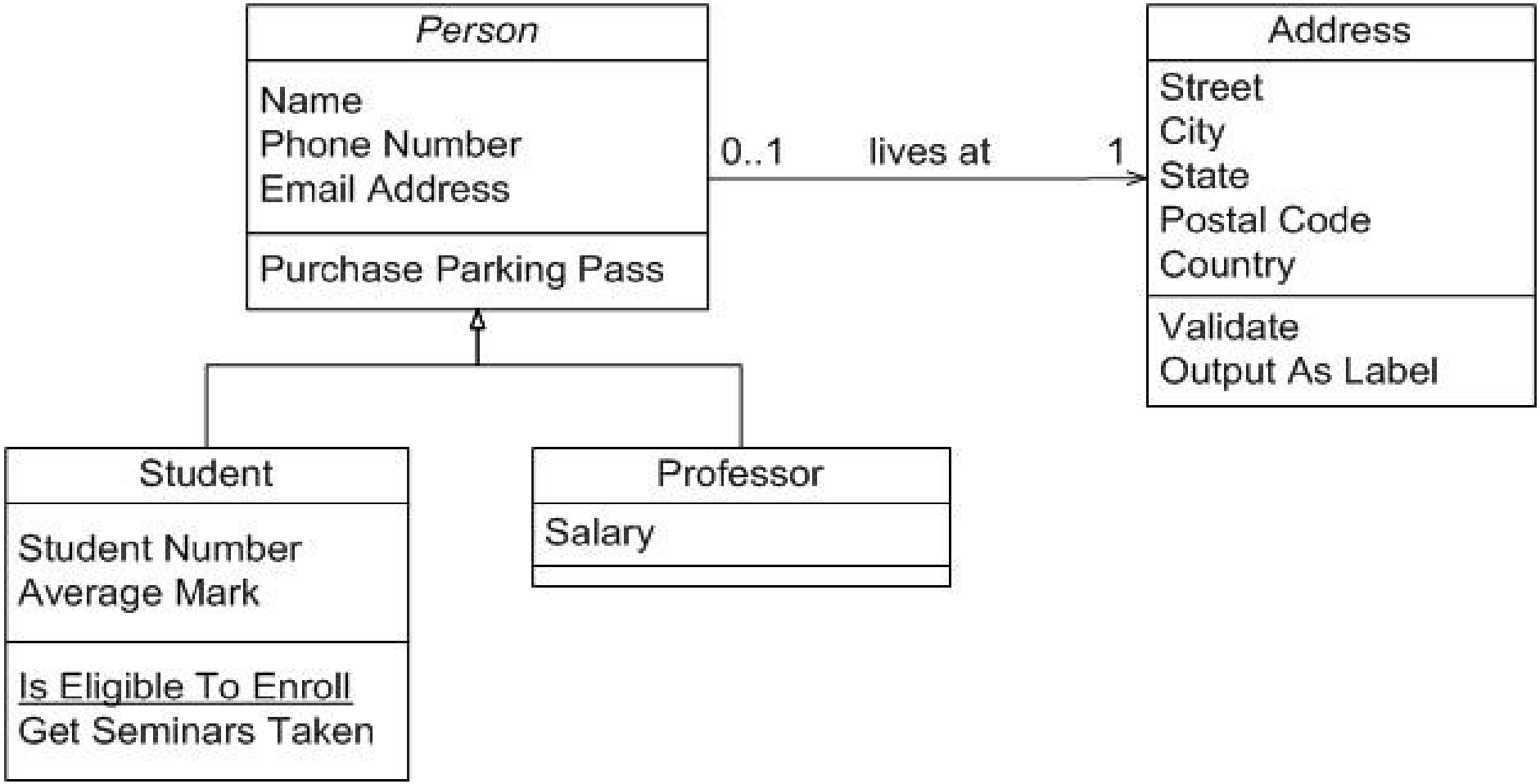
  - Composition



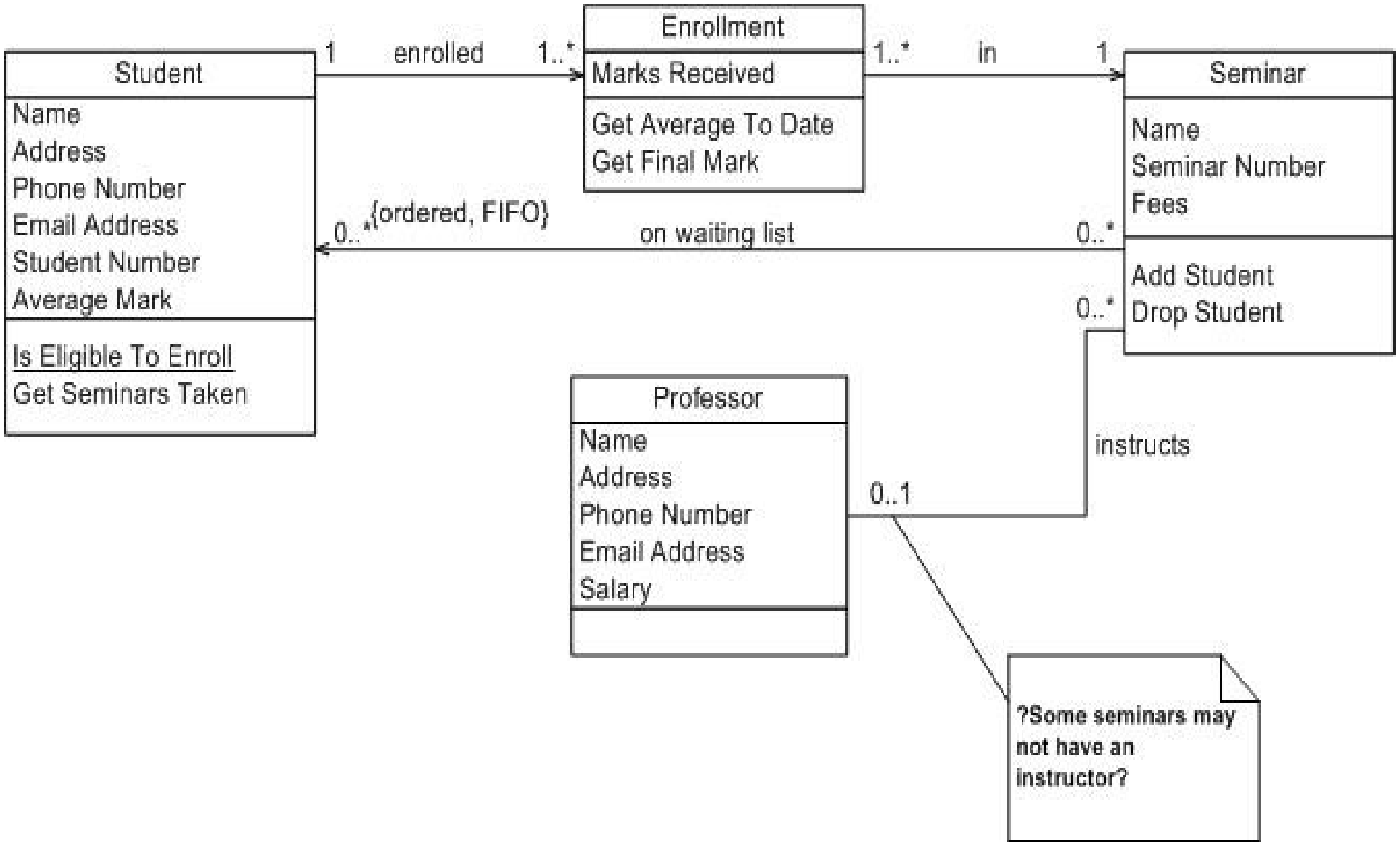
  - Associaton



Class Diagram Example

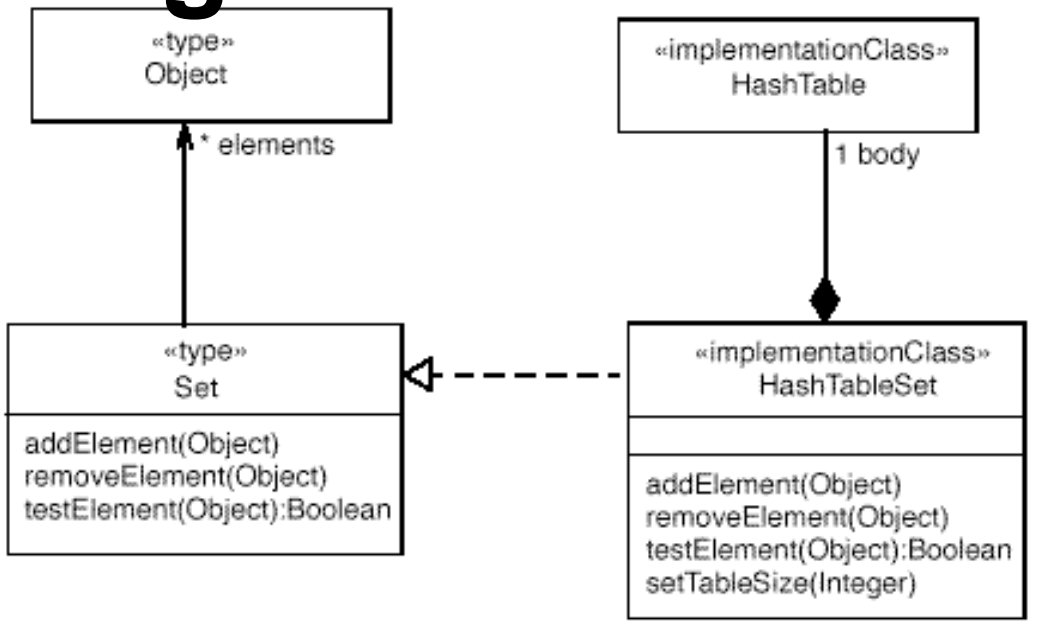


Class Diagram Example

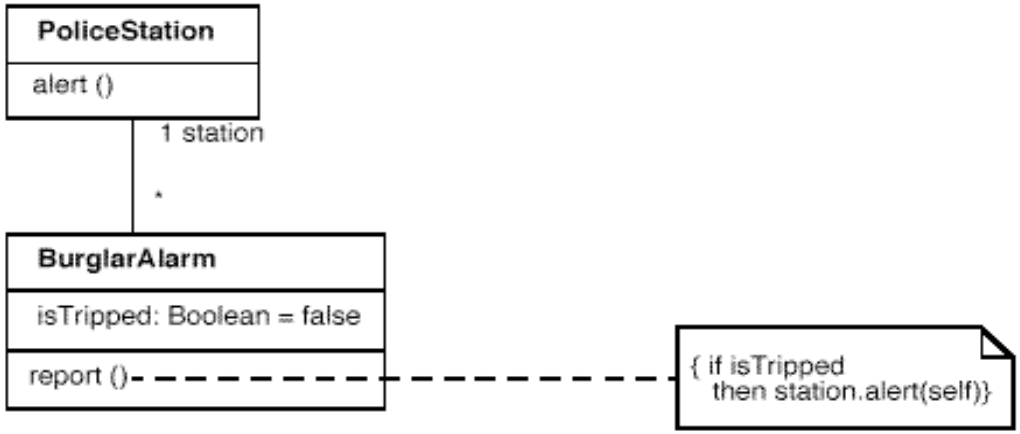


Class Diagrams

- Use of templates, interfaces, and types



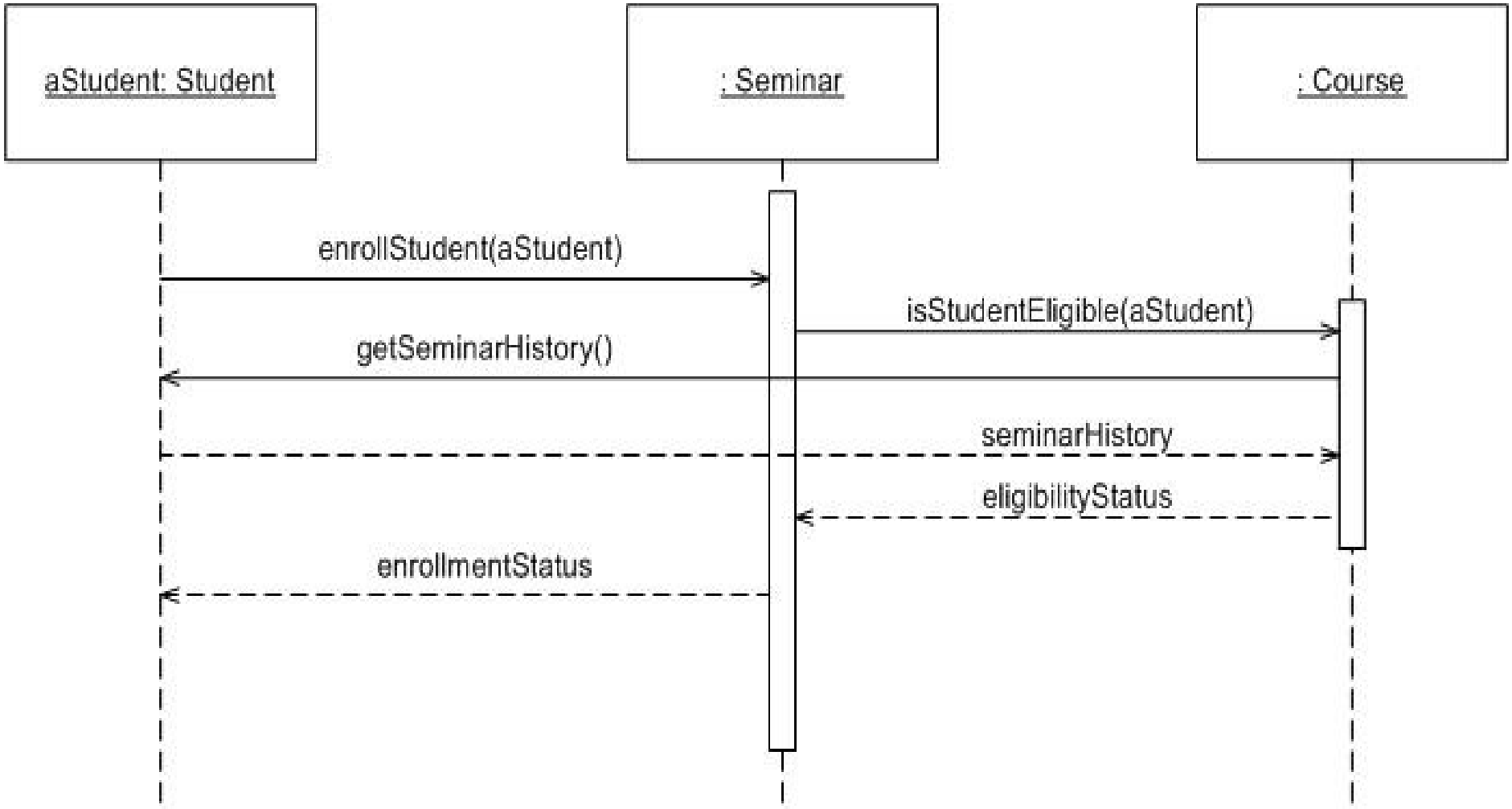
- Can even specify body of methods



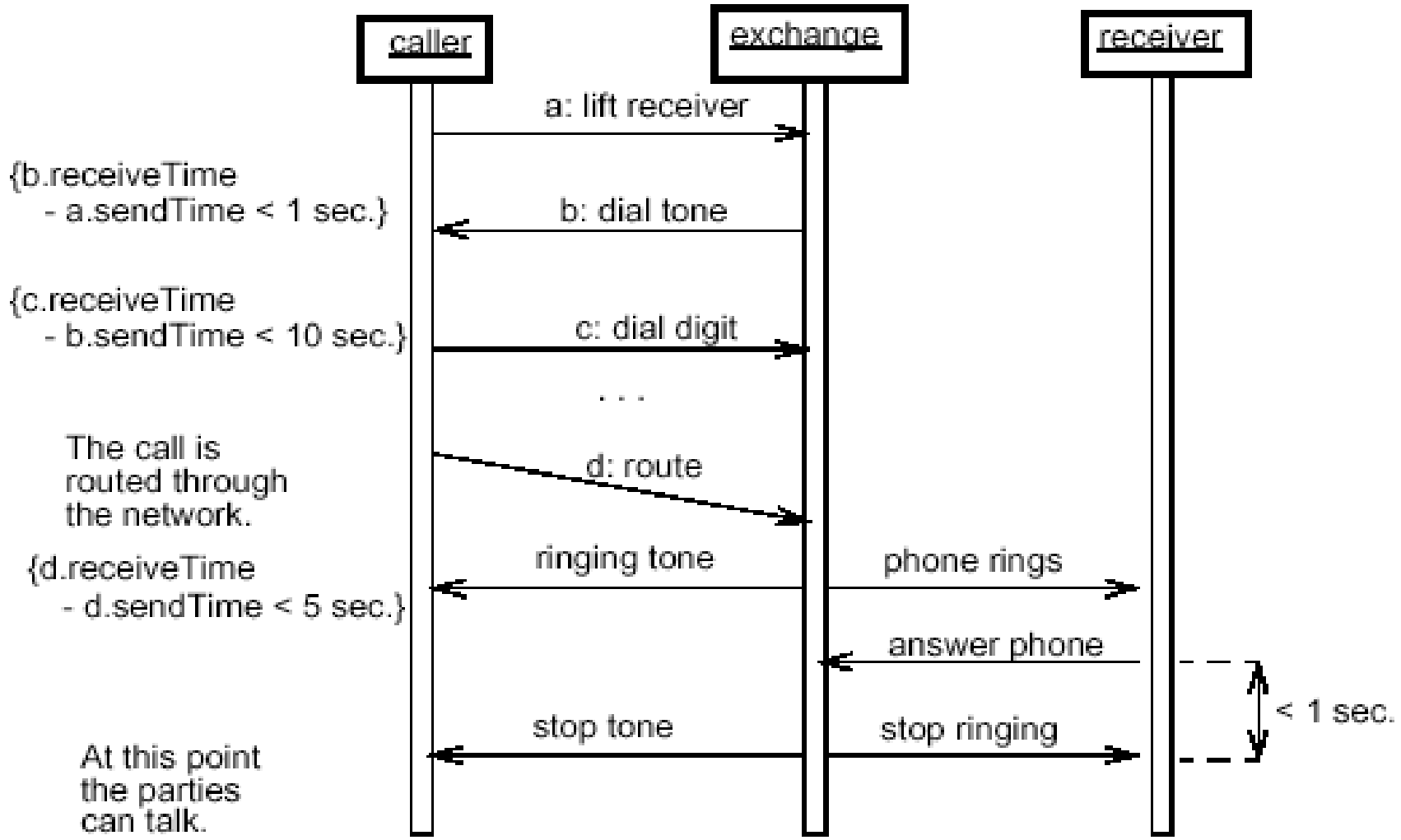
Sequence Diagrams

- Dynamic model view
- Details how operation are carried out, what messages are sent and when
- Two dimensions:
  - Time
  - Objects

Sequence Diagram Example



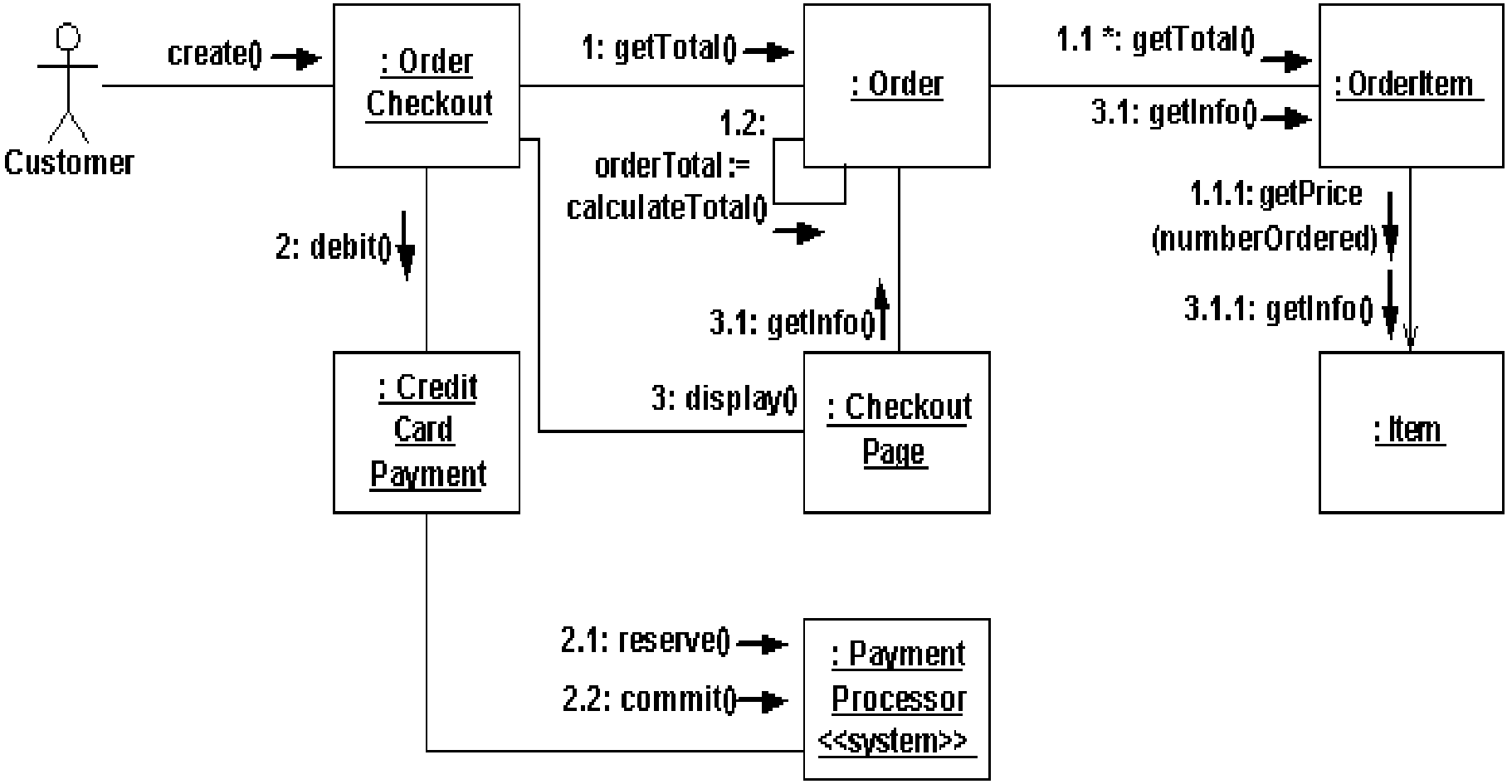
Example Sequence Diagram



Collaboration Diagrams

- Give the same information as sequence diagrams but they focus on object roles instead of the times that messages are sent
- Object roles are vertices and messages are the connecting links
- Each message has a sequence number

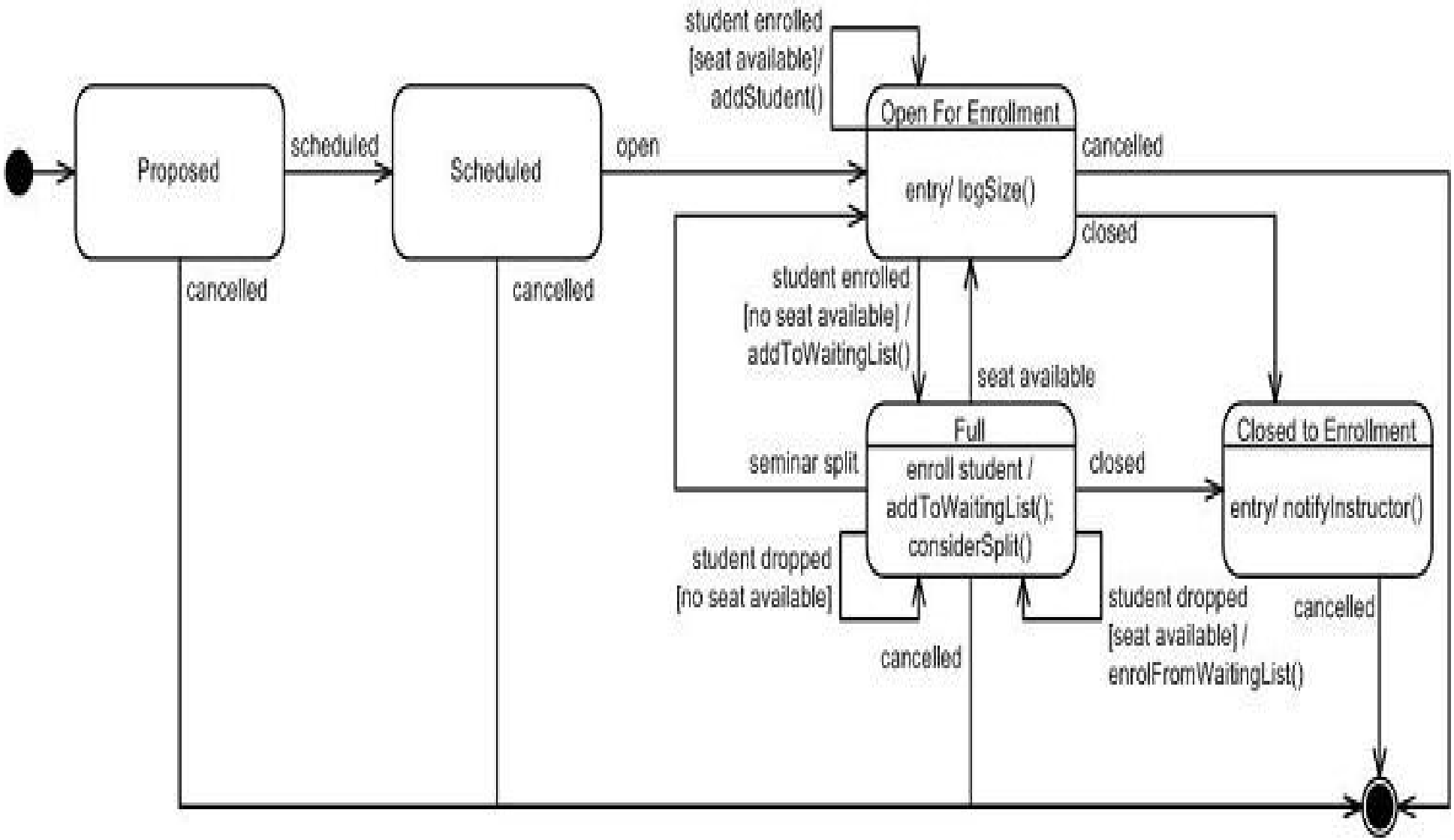
Collaboration Diagram Example



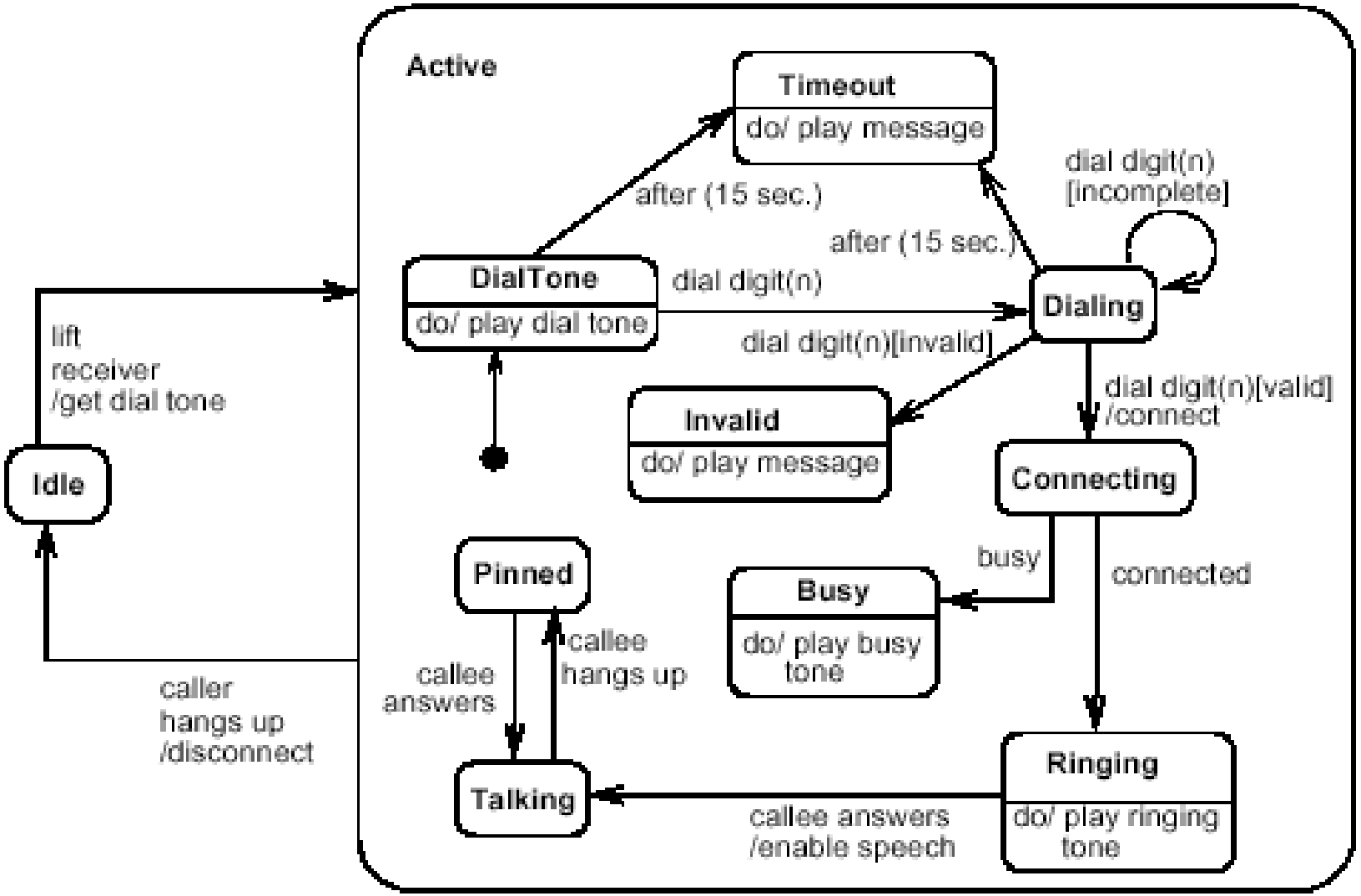
State Diagrams

- Shows the possible states of the object and the transitions that cause a change in state
- Initial state is a dummy to start the action
- Final states are also dummy states that terminate the action

State Diagram Example



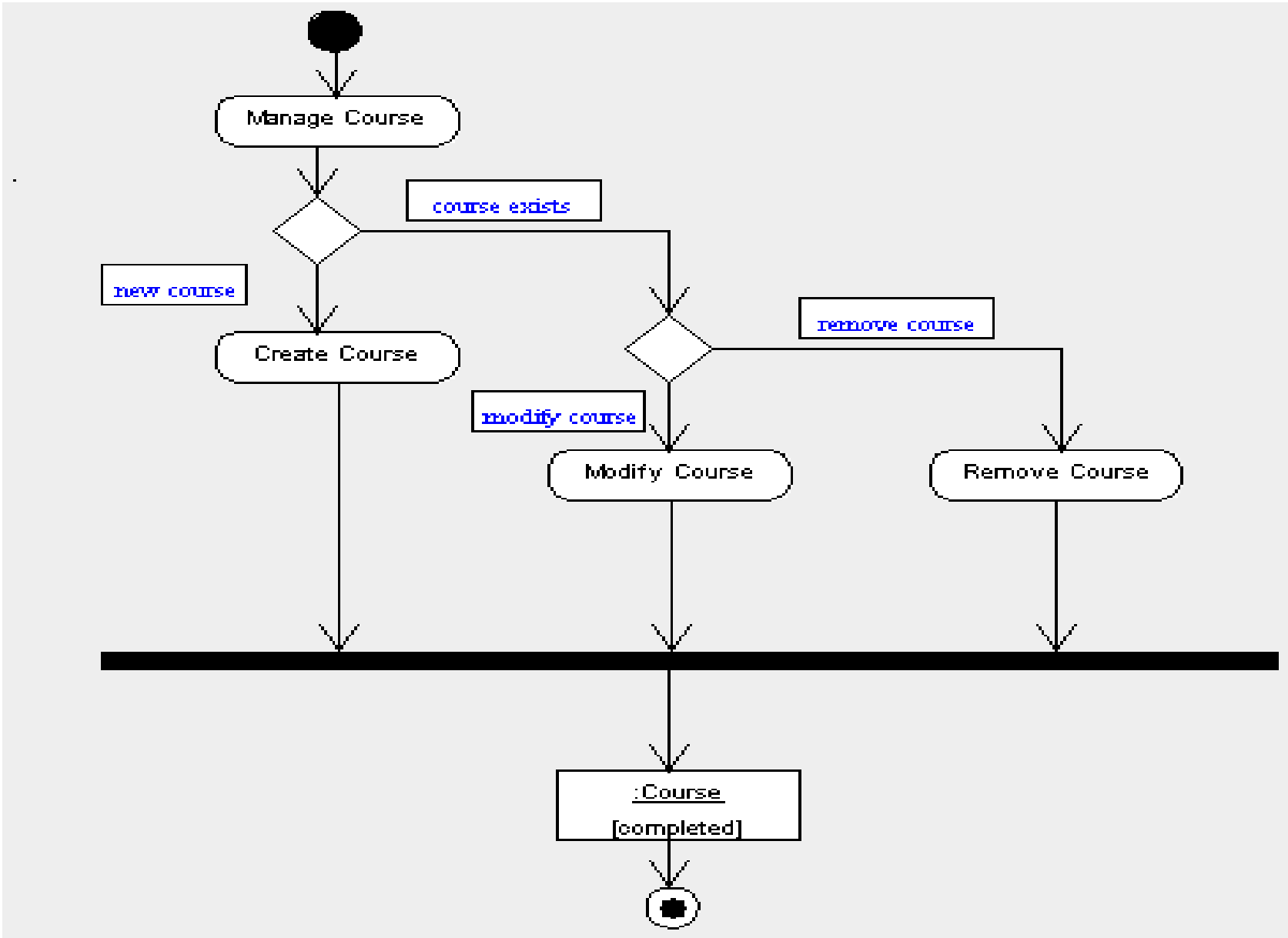
Example State Diagram



Activity Diagrams

- Similar to state diagrams
  - Activity diagram focuses on the flow of activities involved in a single process
  - State diagram focuses attention on an object undergoing a process
- Is essentialy a fancy flowchart

Activity Diagram Example



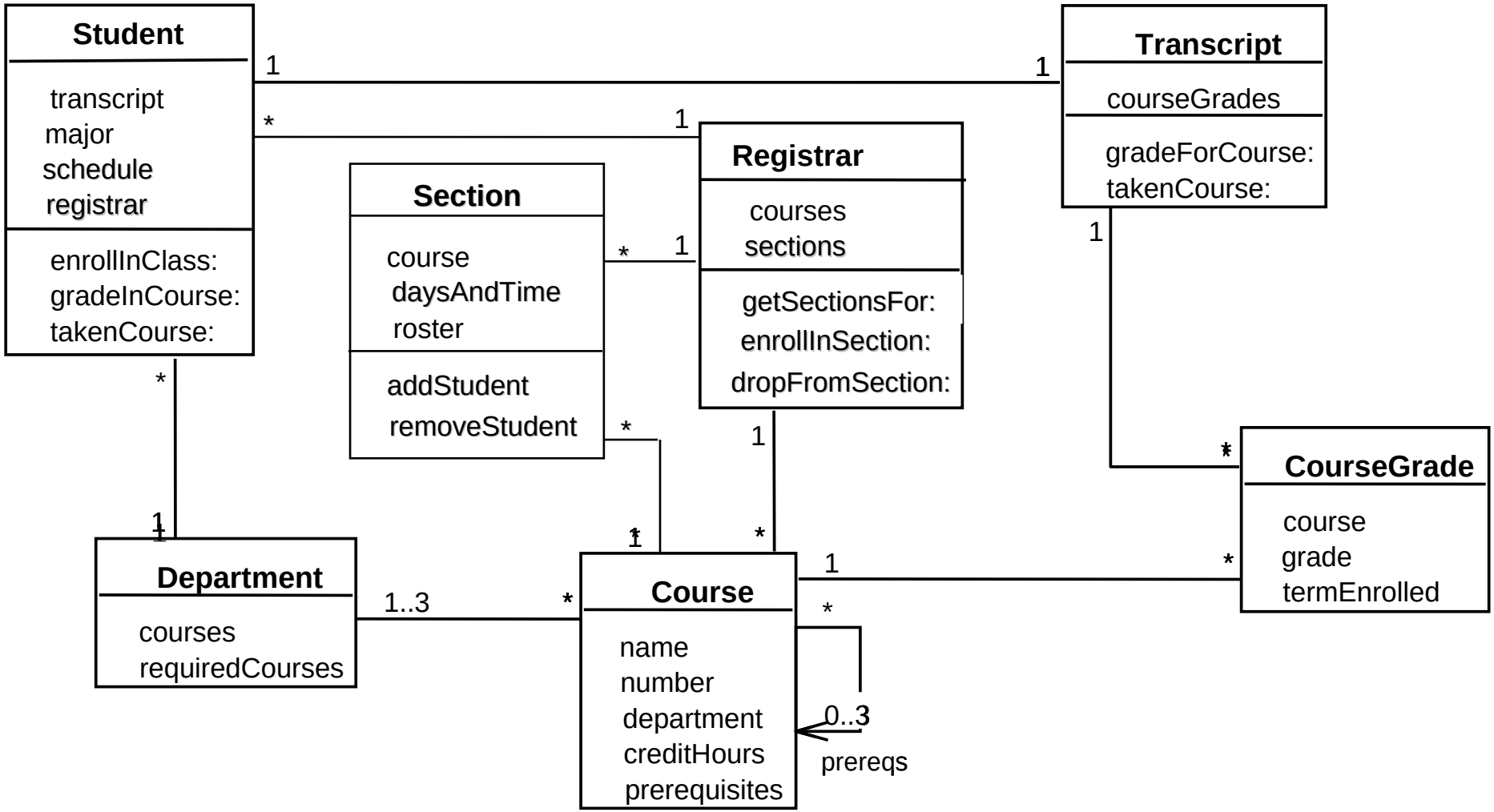
UML in Real Practice

- You don't typically use *all* the diagrams
  - You'll choose between them based on preference and particular situation
- You typically use *many* diagrams
  - A single use case may not capture all scenarios
  - If you are going to use statecharts, there are probably *lots* of objects with states
  - Each sequence/collaboration diagram only shows *one* interaction

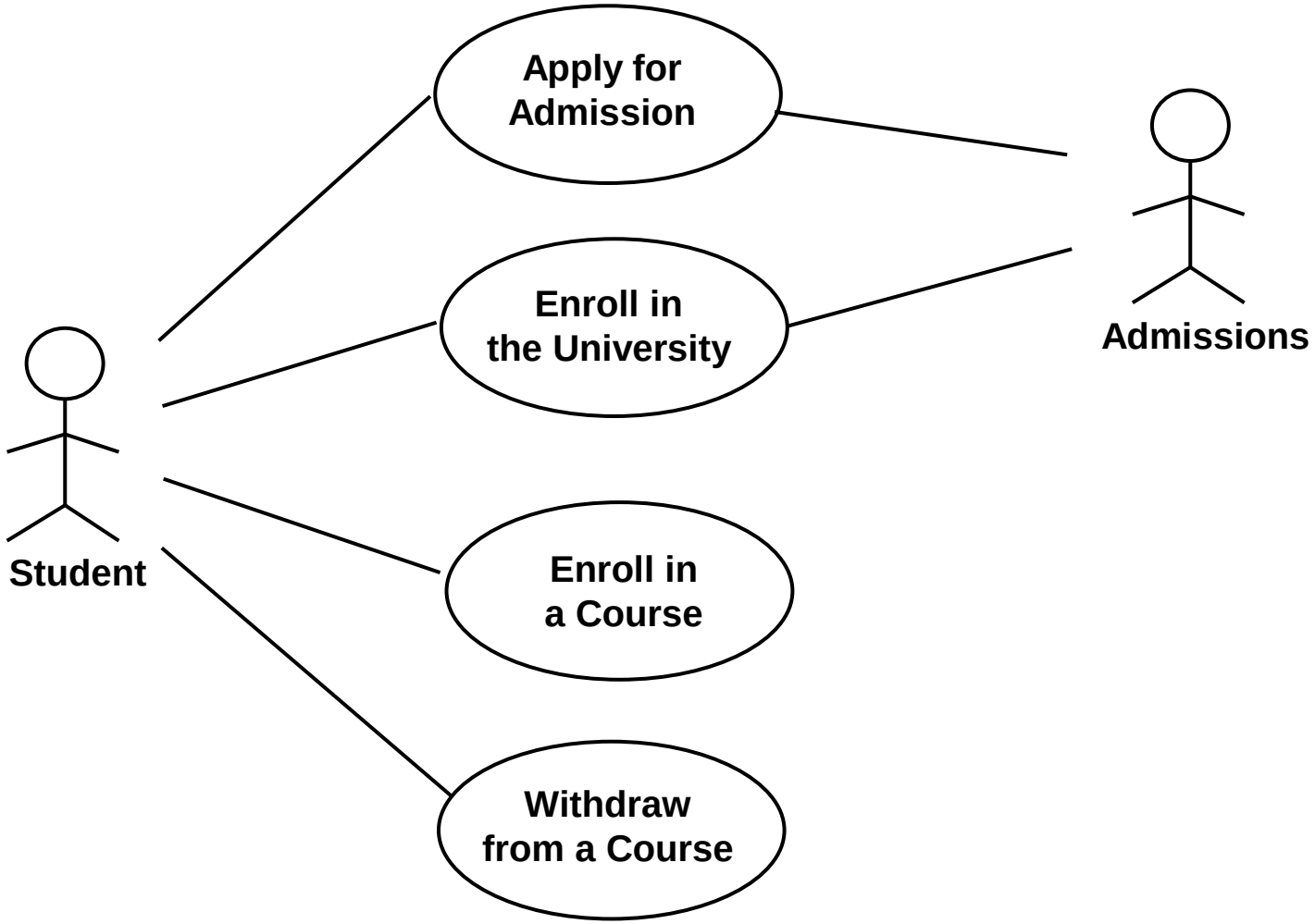
Example: Student Registration System

- Not going to do *all* the diagrams
  - Not all types, not even all that completely specify the system
- But this is an application you know, so the examples may help make sense

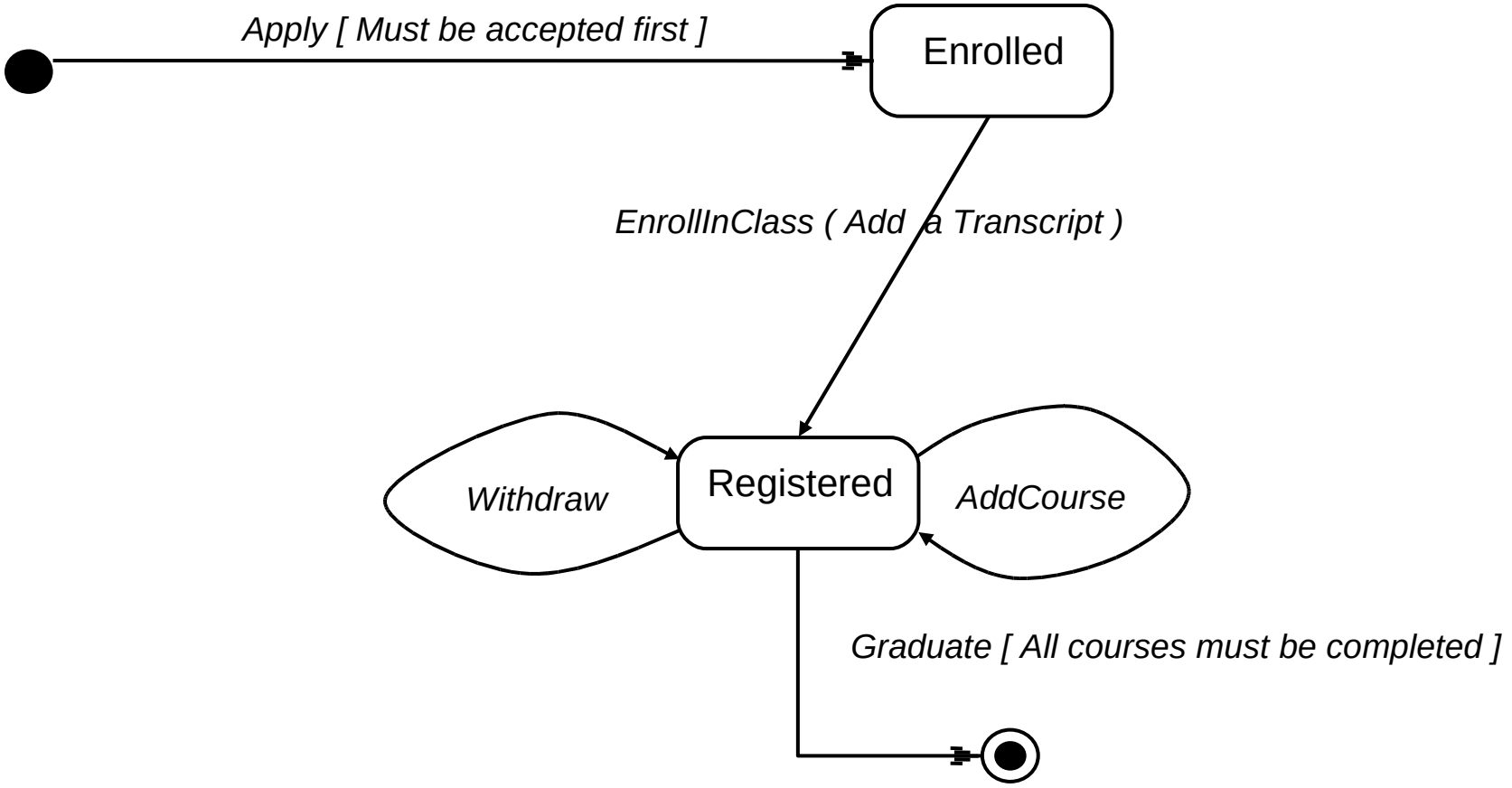
Student Registration Class Diagram



Partial Use Case Diagram



States of a Student



Sequence Diagram: Registering for Course