

MACIASZEK, L.A. (2005):  
*Requirements Analysis and System Design*, 2<sup>nd</sup> ed.  
Addison Wesley, Harlow England, 504p.  
ISBN 0 321 20464 6

---

Chapter 4  
*Requirements Specification*

© Pearson Education Limited 2005

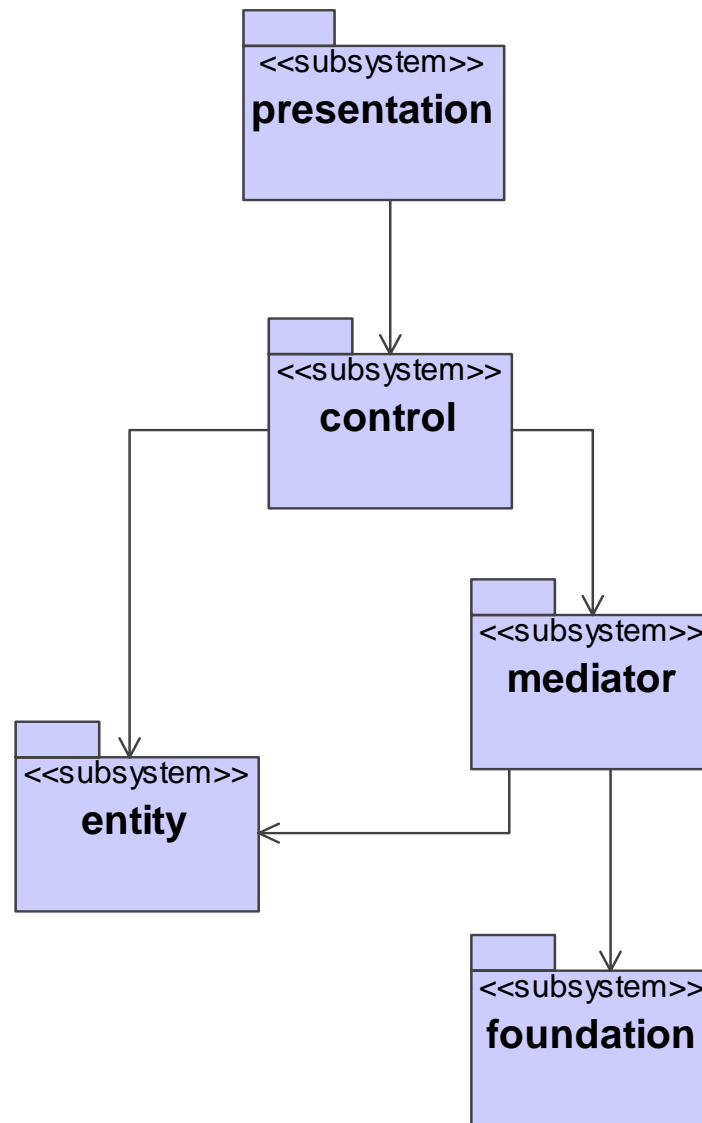
# *Topics*

- *Architectural prerogatives*
  - *BCED in RASD 1ed → PCMEF in RASD 2ed*
- *State specifications*
- *Behavior specifications*
- *State change specifications*

# *Architectural design*

- *Design*
  - *detailed*
  - *architectural*
- *Object dependencies → complexity and supportability*
- *Architectural model*
  - *hierarchical layers*
  - *restrictions on object inter-communications*
- *RASD 1/e → BCED (boundary, control, entity, database)*
- *RASD 2/e → PCMEF (presentation, control, mediator, entity, foundation)*

# PCMEF framework



# *PCMEF subsystems*

- *The presentation subsystem*
  - *classes that handle the graphical user interface (GUI) and assist in human-computer interactions.*
- *The control subsystem*
  - *classes capable to understand what program logic is*
    - *searching for information in entity objects*
    - *asking the mediator layer to bring entity objects to memory from the database.*
- *The entity subsystem*
  - *manages business objects currently in memory*
  - *container classes*
  - *containers are linked*
- *The mediator subsystem*
  - *mediates between entity and foundation subsystems to ensure that control gets access to business objects*
  - *manages the memory cache and synchronizes the states of business objects between memory and the database*
- *The foundation subsystem*
  - *classes that know how to talk to the database*
  - *produces SQL to read and modify the database*

# *Architectural principles*

- *DDP – downward dependency principle*
- *UNP – upward notification principle*
- *NCP – neighbor communication principle*
- *APP – acquaintance package principle*
- *EAP – explicit association principle*
- *CEP – cycle elimination principle*
- *CNP – class naming principle*

# *DDP, UNP, NCP*

## ■ *DDP*

- *higher PCMEF layers depend on lower layers*
- *lower layers should be designed to be more stable*

## ■ *UNP*

- *upward communication that minimizes object dependencies*
- *lower layers rely on interfaces and event processing (publisher/subscriber protocols) to communicate with objects in higher layers*

## ■ *NCP*

- *objects can communicate across layers only by using direct neighbors*
- *chains of message passing*

# *APP, EAP, CEP, CNP*

## ■ *APP*

- *separate layer of interfaces to support more complex object communication under strict supportability guidelines*
- *subsystem of interfaces only*
  - *other objects in the system can use these interfaces, and pass them in arguments to method calls, instead of concrete objects → classes in non-neighboring subsystems can communicate without knowing the concrete suppliers of services (and, therefore, without creating dependencies on concrete classes).*

## ■ *EAP*

- *legitimizes run-time object communication in compile-time data structures.*

## ■ *CEP*

- *cyclic dependencies, between classes and other structures (methods, packages, subsystems)*
- *unavoidable, but can be neutralized*
  - *extra classes to reduce a network of calls to a hierarchy*
  - *purposeful use of interfaces*

## ■ *CNP*

- *name of each class and each interface in the system should identify the subsystem/package layer to which it belongs*
- *ensuring that each class begins with a single letter identifying the PCMEF layer (i.e. P, C, etc.)*
  - *EVideo means that the class is in the entity subsystem*
  - *IMVideo means that the interface is in the mediator subsystem.*

# *State specifications*

- **Object state** is determined by the values of its attributes and associations
- **State specification:**
  - *Model of data structures*
  - *Static view on the system*
  - *Class operations left out in initial specs*
  - *Emphasis on entity classes (“business objects”)*

# *Modeling classes*

- *Cornerstone of OO development – a system is a set of collaborating (and classified) objects*
- *Iterative and incremental process*
- *CASE tool*
  - *For collaborative development*
  - *For personal productivity otherwise*

# *Discovering classes*

- *No two analysts will come up with the identical class models for the same application domain*
- *Discovering classes*
  - *Noun phrase*
  - *Common class patterns*
  - *Use case driven*
  - *CRC*
  - *Mixed*

# *Noun phrase approach*

- *Nouns considered candidate classes*
- *Three kinds of candidate classes*
  - *Irrelevant (can be skipped)*
  - *Relevant*
  - *Fuzzy*
- *Assumes that the Requirements Document is complete and correct*

# *Common class pattern approach*

- *Derives candidate classes from the classification theory of objects*
- *One possible classification pattern:*
  - *Concept (e.g. Reservation)*
  - *Event (e.g. Arrival)*
  - *Organization (e.g. Department)*
  - *People (e.g. Passenger)*
  - *Place (e.g. TravelOffice)*
- *Just a guidance*
- *Only loosely bound to user requirements*
- *Possible naming misinterpretations*

# *Use case driven approach*

- *Assumes that:*
  - *Use Case Diagrams (and possibly some high-level Sequence Diagrams) have been developed*
  - *Narrative descriptions for each use case exist*
- *Similar to the noun phrase approach*
- *Function-driven (problem-driven)*
- *Relies on the completeness of use case models*

# *CRC approach*

- *CRC – classes, responsibilities, collaborators*
- *More than a technique for class discovery*
- *Animated brainstorming sessions*
- *Identifies classes from the analysis of how objects collaborate to perform business functions (use cases)*
- *Suitable also for:*
  - *Verification of classes discovered with other methods*
  - *Determination of class properties*

# *Mixed approach*

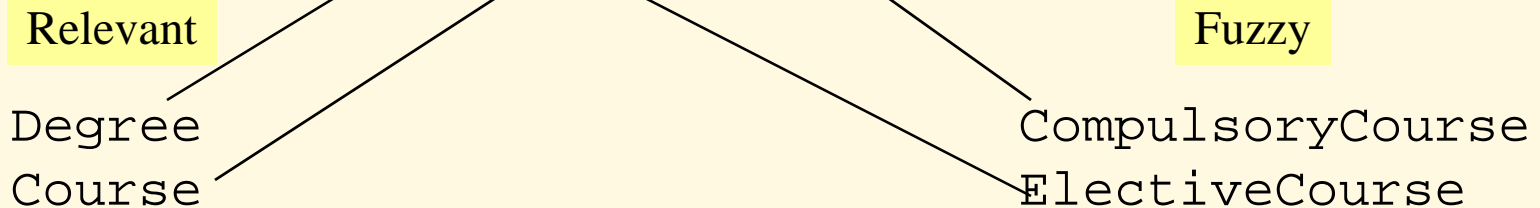
- *Perhaps with elements of all four previous approaches*
- *Middle-out rather than top-down or bottom-up*
- *One possible scenario:*
  - *Initial classes – domain knowledge*
  - *Common class patterns approach to guide*
  - *Noun phrase approach to add more classes*
  - *Use case approach to verify*
  - *CRC to brainstorm*

# *Guidelines for class discovery*

- *Statement of purpose*
- *Description for a set of objects*
  - *Singleton classes*
- *Houses a set of attributes*
  - *Identifying attributes - keys*
  - *OID*
- *Class or attribute?*
- *Houses a set of operations (what does the class do?)*

# Example 4.1 – University Enrolment

- Consider the following requirements for the University Enrolment system and identify the candidate classes:
  - Each university degree has a number of compulsory courses and a number of elective courses.



# *Example 4.1 – University Enrolment*

- *More requirements:*
  - *Each course is at a given level and has a credit-point value*
  - *A course can be part of any number of degrees*
  - *Each degree specifies minimum total credit points value required for degree completion*
  - *Students may combine course offerings into programs of study suited to their individual needs and leading to the degree in which enrolled*

## *Example 4.1– University Enrolment (solution)*

<i>Relevant classes</i>	<i>Fuzzy classes</i>
<i>Course</i>	<i>CompulsoryCourse</i>
<i>Degree</i>	<i>ElectiveCourse</i>
<i>Student</i>	<i>Sudyprogram</i>
<i>CourseOffering</i>	

## Example 4.2 – Video Store

- Consider the following requirements for the Video Store system and identify the candidate classes:
- The video store keeps in stock an extensive library of current and popular movie titles. A particular movie may be held on video tapes or disks.

Relevant

MovieTitle  
VideoTape  
VideoDisk

Irrelevant

VideoStore  
Stock  
Library

# *Example 4.2 – Video Store*

## ■ *More requirements:*

- *Video tapes are in either "Beta" or "VHS" format*
- *Video disks are in DVD format*
- *Each movie has a particular rental period (expressed in days), with a rental charge to that period*
- *The video store must be able to immediately answer any inquiries about a movie's stock availability and how many tapes and/or disks are available for rental*
- *The current condition of each tape and disk must be known and recorded*

## *Example 4.2 – Video Store (solution)*

<b><i>Relevant classes</i></b>	<b><i>Fuzzy classes</i></b>
<i>MovieTitle</i> <i>VideoMedium</i>	<i>RentalConditions</i>
<i>VideoTape</i>	
<i>VideoDisk</i> <i>(or DVDDisk)</i>	
<i>BetaTape</i>	
<i>VHSTape</i>	

# Example 4.3 – Contact Management

- Consider the following requirements for the Contact Management system and identify the candidate classes:
  - To "keep in touch" with current and prospective customer base
  - To win new contracts
  - To store the names, phone numbers, postal and courier addresses, etc. of organizations and contact persons in these organizations
  - To schedule tasks and events for the employees with regard to relevant contact persons
  - Employees can schedule tasks and events for other employees or for themselves
  - A task is a group of events that take place to achieve a result (e.g. to solve customer's problem)
  - Typical types of events are: phone call, visit, sending a fax, arranging for training, etc.

## *Example 4.3 – Contact Management (solution)*

<b><i>Relevant classes</i></b>	<b><i>Fuzzy classes</i></b>
<i>Organization</i>	<i>CurrentOrg</i>
<i>Contact</i>	<i>ProspectiveOrg</i>
<i>Employee</i>	<i>PostalAddress</i>
<i>Task</i>	<i>CourierAddress</i>
<i>Event</i>	

# Specifying classes

## ■ In Class Diagram

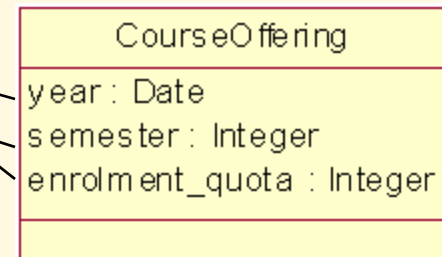
- Each class given a name (and possibly a code)
- Singular noun
  - Recommendation – multiple words joined; each word starting with a capital letter (e.g. `PostalAddress`)
- Meaningful
- Short (less than 30 characters)

## ■ Class properties to be defined

- Attributes (initially those that capture interesting object states)
  - Recommendations:
    - small letters; underscore to separate words (e.g. `street_name`)
    - start with a small letter; capitalize successive words (`streetName`)
- Operations (can be delayed till later analysis stages or even till design)

# Example 4.4 – University Enrolment

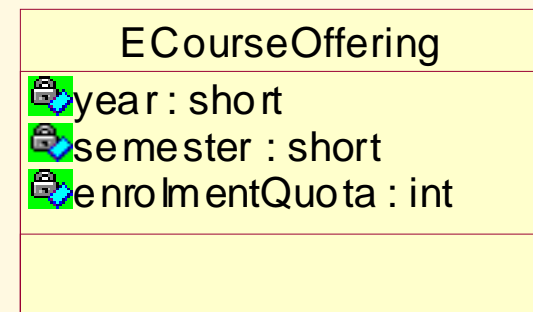
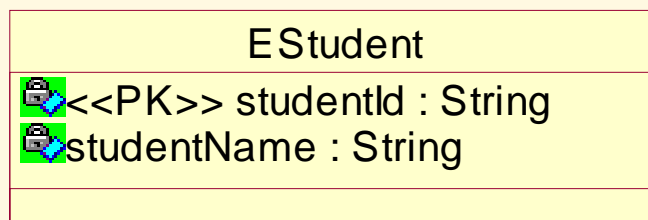
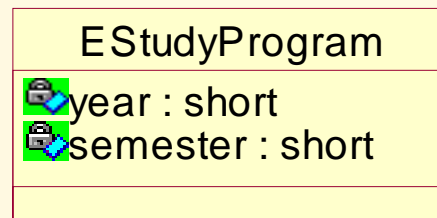
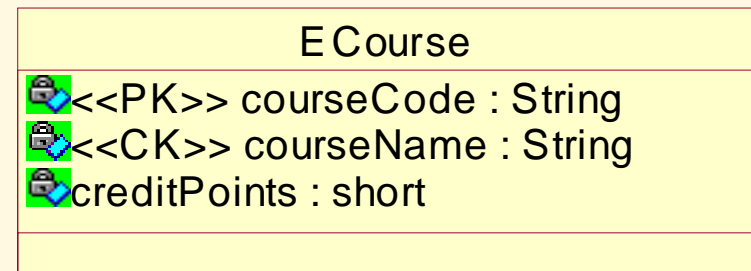
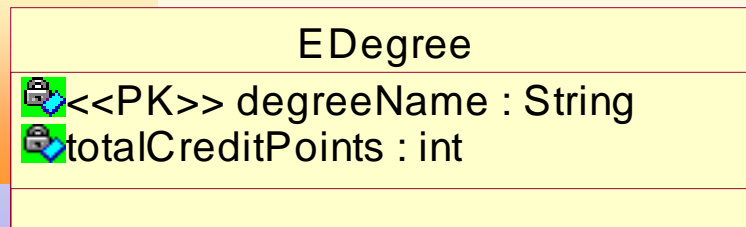
- Refer to Example 4.1
- Consider the following additional requirements from the Requirements Document:
  - A student's choice of courses may be restricted by timetable clashes and by limitations on the number of students who can be enrolled in the current course offering.



# *Example 4.4 – University Enrolment*

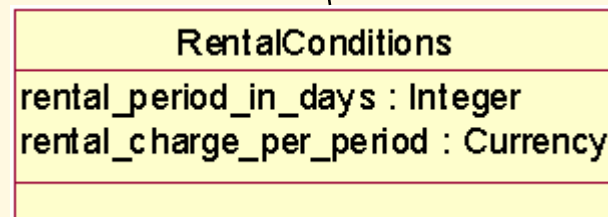
- *More requirements:*
  - *A student's proposed program of study is entered on on-line enrolment system T*
  - *The system checks the program's consistency and reports any problems*
  - *The problems need to be resolved with the help of an academic adviser*
  - *The final program of study is subject to academic approval by the delegate of the Head of Division and it is then forwarded to the Registrar*

## Example 4.4 – University Enrolment (solution)



## Example 4.5 – Video Store

- Refer to Example 4.2
- The additional requirements are:
  - The rental charge differs depending on video medium: tape or disk (but it is the same for the two categories of tapes: Beta and VHS).

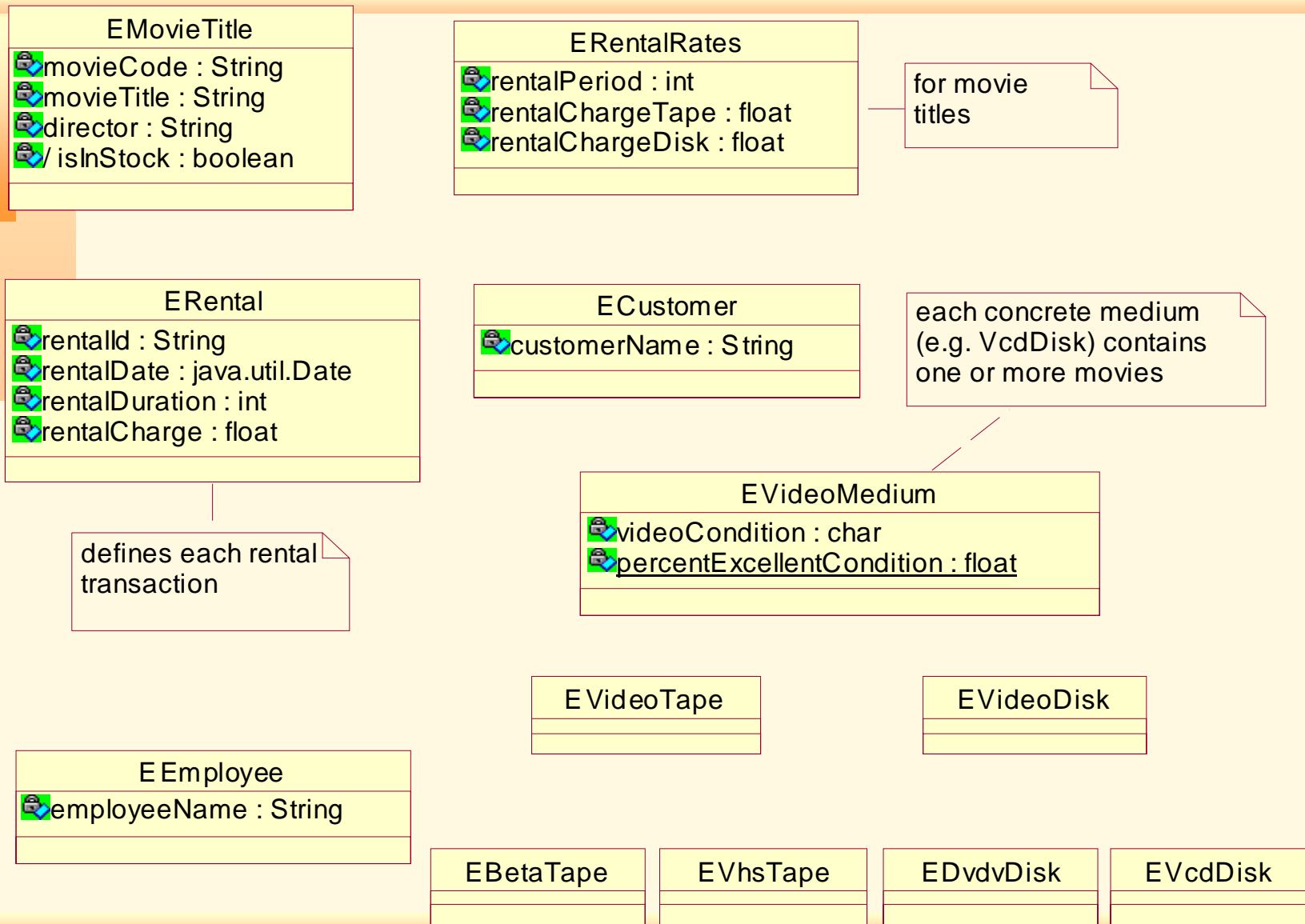


RentalConditions
rental_period_in_days : Integer
rental_charge_per_period : Currency

## *Example 4.5 – Video Store*

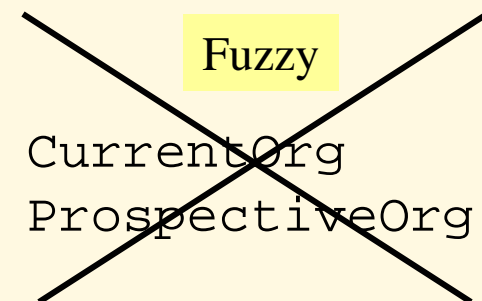
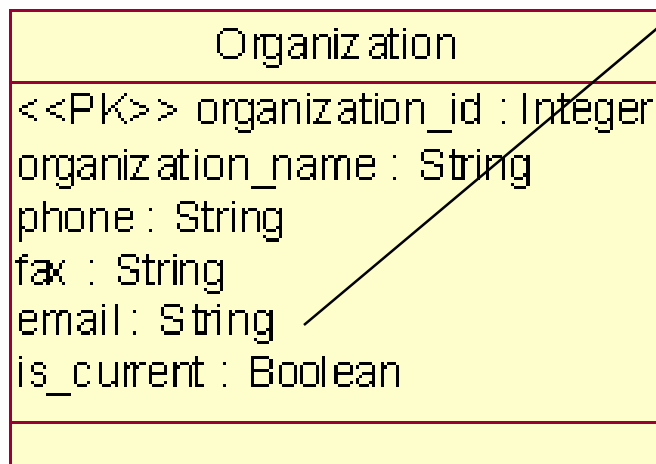
- *More requirements:*
  - *The system should accommodate future video storage formats in addition to VHS tapes, Beta tapes and DVD disks*
  - *The employees frequently use a movie code, instead of movie title, to identify the movie*
  - *The same movie title may have more than one release by different directors*

# Example 4.5 – Video Store (solution)



# Example 4.6 – Contact Management

- Refer to Example 4.3 and consider the following additional information
  - A customer is considered current if there exists a contract with that customer for delivery of our products or services. Contract management is, however, outside the scope of our system.

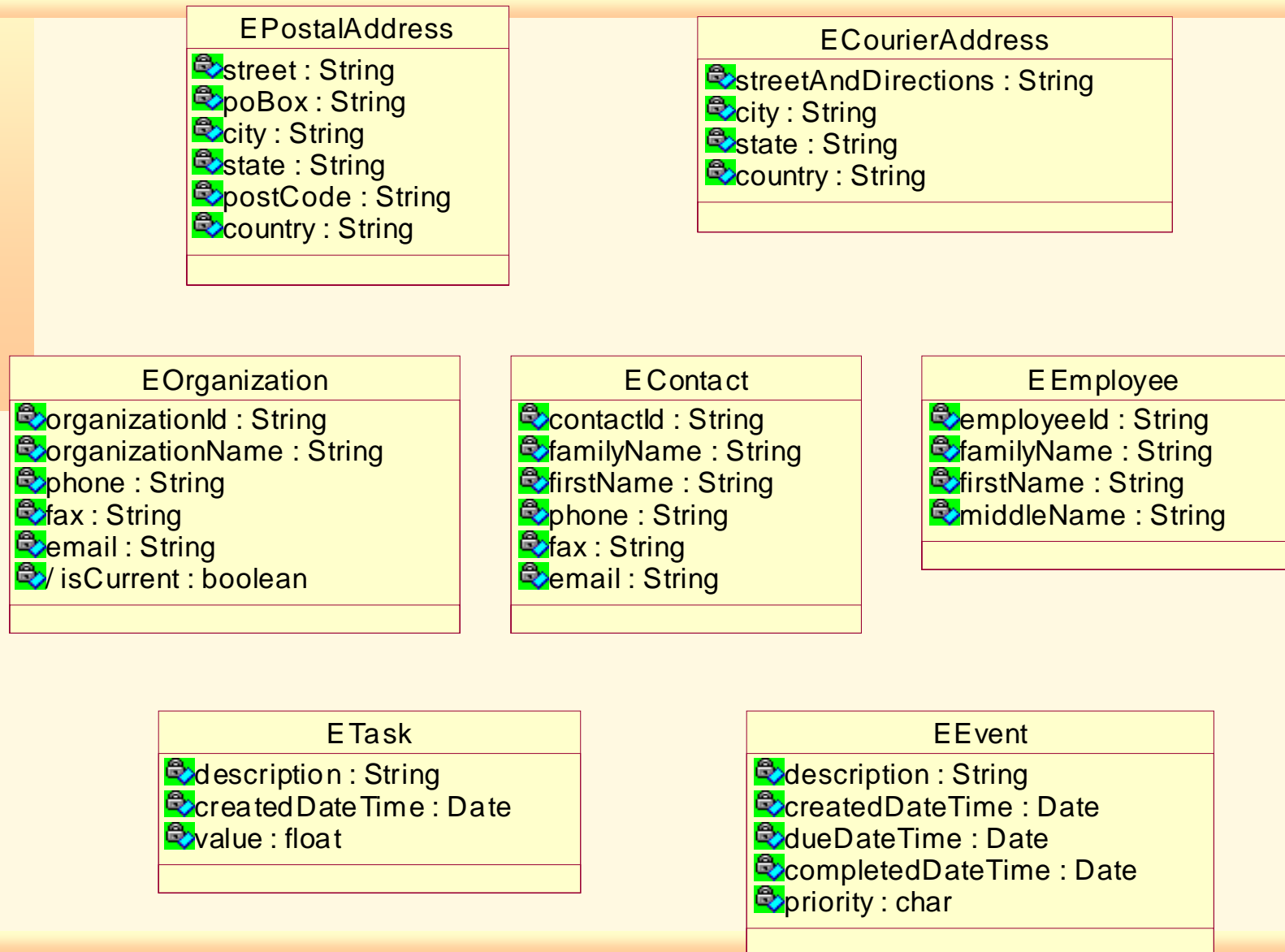


# Example 4.6 – Contact Management

## ■ More requirements:

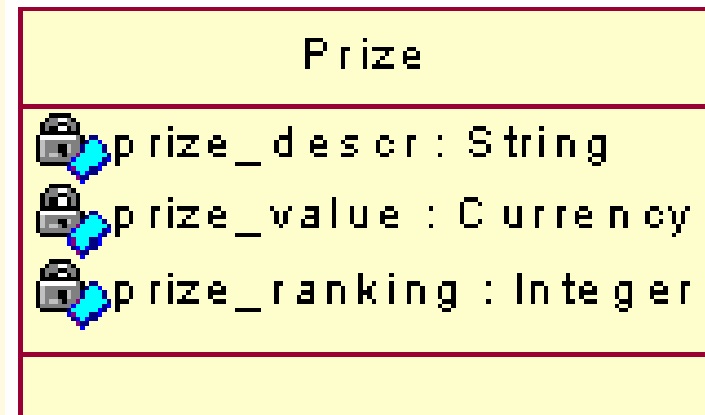
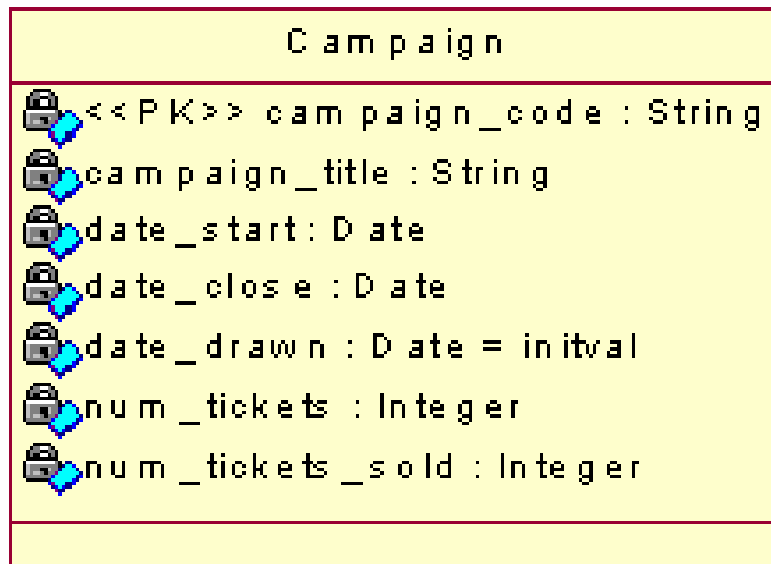
- *Reports on contacts based on postal and courier addresses (e.g. find all customers by post code)*
- *Date and time of the task creation are recorded*
- *The "money value" of a task can be stored*
- *Events for the employee are displayed on the employee's screen in the calendar-like pages (one day per page).*
  - *The priority of each event (low, medium or high) is visually distinguished on the screen*
- *Not all events have a "due time" - some are "untimed"*
- *Event creation time cannot be changed, but the due time can.*
- *Event completion date and time are recorded*
- *The system stores identifications of employees who created tasks and events, who are scheduled to do the event ("due employee"), and who completed the event*

# Example 4.6 – Contact Management (solution)



# Example 4.7 - Telemarketing

- Consider the following additional information
  - Each campaign
    - Has a title that is generally used for referring to it
    - Has also a unique code for internal reference
    - Runs over a fixed period of time
  - Soon after the campaign is closed, the prizes are drawn and the holders of winning tickets are advised



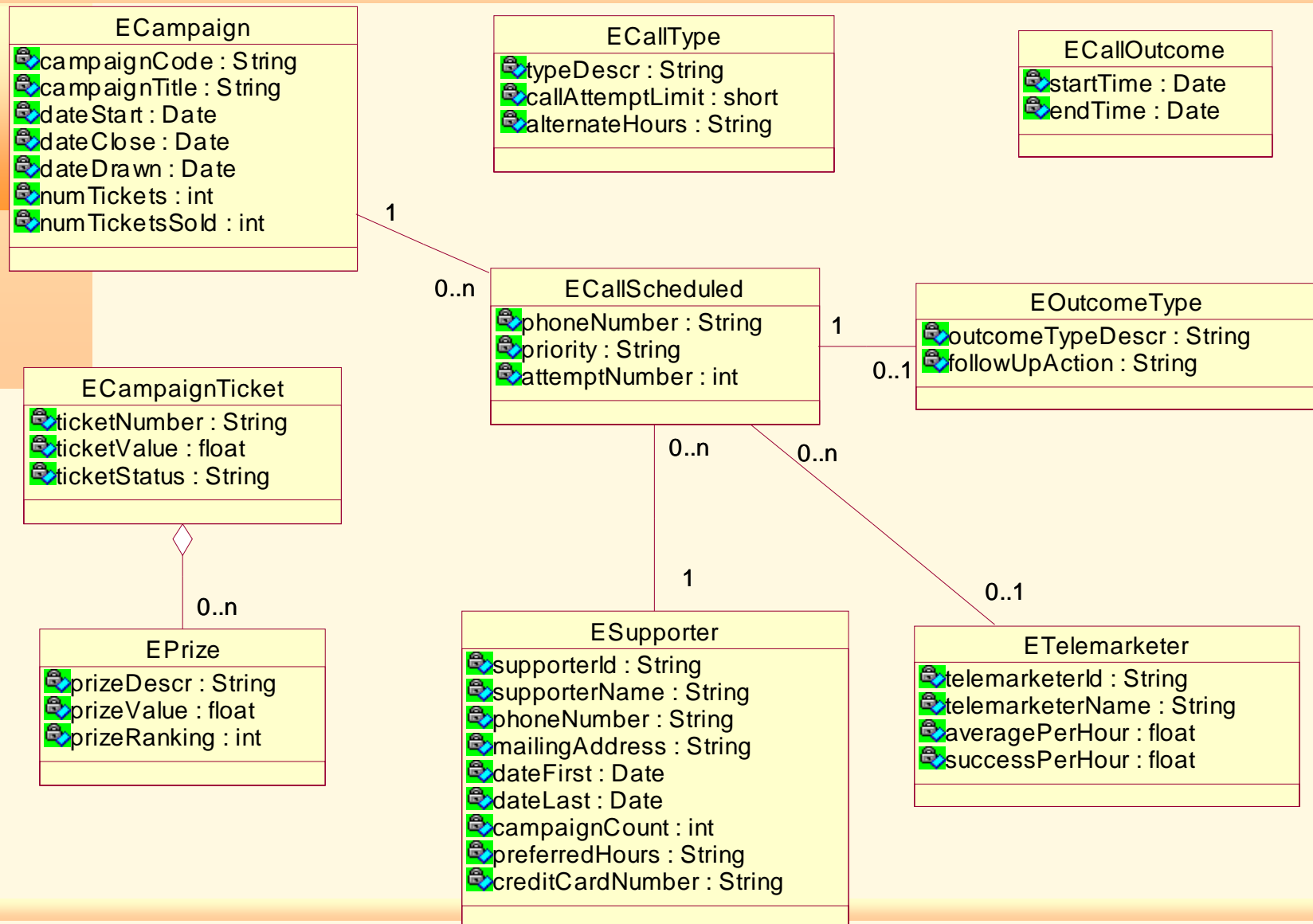
# Example 4.7 - Telemarketing

- *More requirements:*
  - *Tickets are uniquely numbered within each campaign*
  - *The total number of tickets in a campaign, number of tickets sold so far, and the current status of each ticket are known (e.g. available, ordered, paid for, prize winner)*
  - *To determine the performance of the society's telemarketers, the duration of calls and the successful call outcomes (i.e. resulting in ordered tickets) are recorded*
  - *Extensive information about supporters is maintained*
    - *Contact details (address, phone number, etc.)*
    - *Historical details such as the first and most recent dates when a supporter had participated in a campaign*
    - *Any known supporter's preferences and constraints (e.g. times not to call, usual credit card number)*

# Example 4.7 - Telemarketing

- *More requirements:*
  - *Telemarketing calls are made according to their priorities*
  - *Calls which are unanswered or where an answering machine was found, are rescheduled*
    - *Times of repeat calls are alternated*
    - *Number of repeat calls is limited*
      - *Limits may be different for different call types (e.g. a normal "solicitation" call may have different limit than a call to remind a supporter of an outstanding payment)*
  - *Call outcomes are categorized - success (i.e. tickets ordered), no success, call back later, no answer, engaged, answering machine, fax machine, wrong number, disconnected.*

# Example 4.7 – Telemarketing (solution)



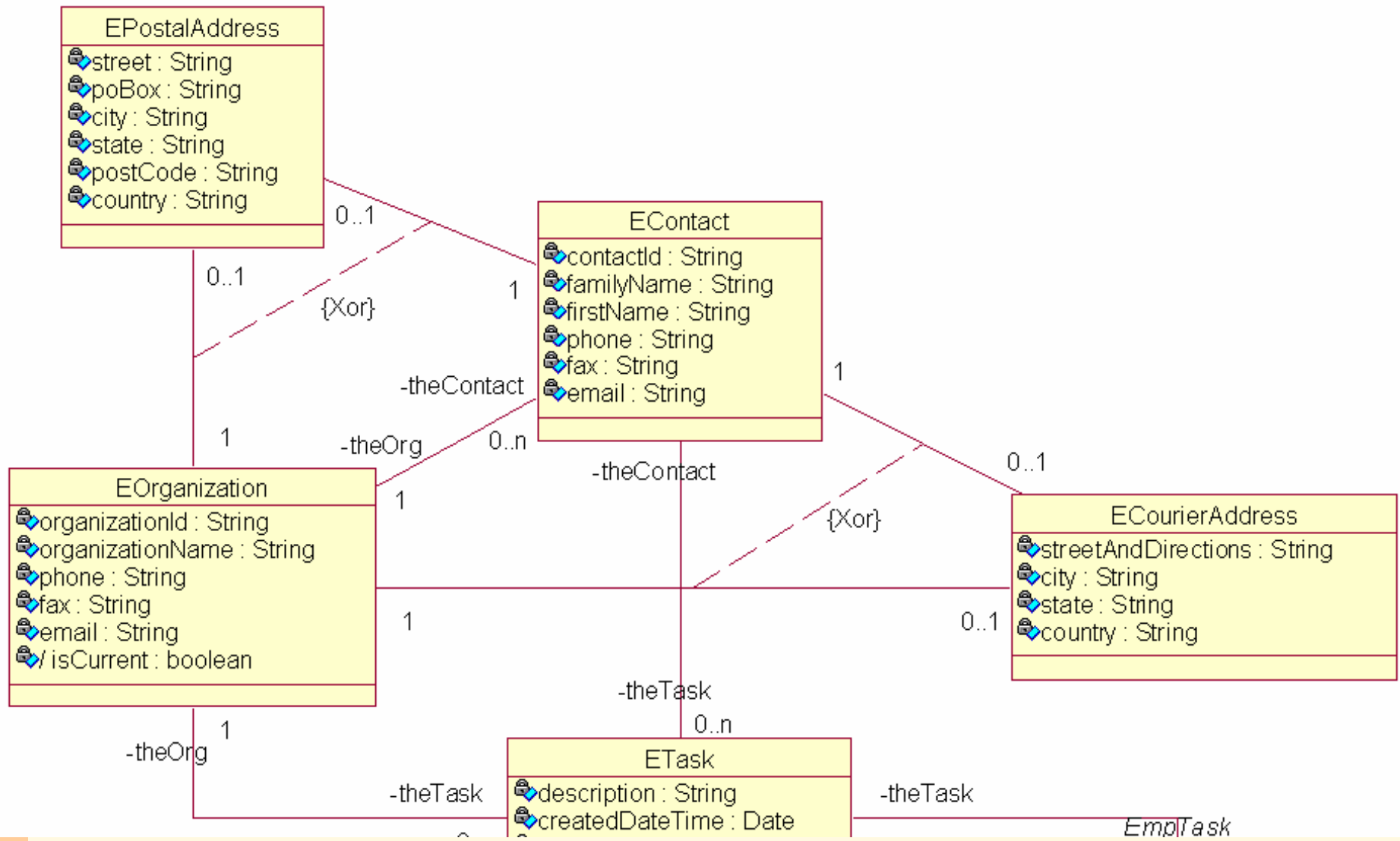
# *Discovering associations*

- *Side effect of discovering classes*
- *Some attributes are associations*
- *“Dry-run” of use cases to discover more associations*
- *Avoid ternary associations*
- *Cycles of associations that do not commute*

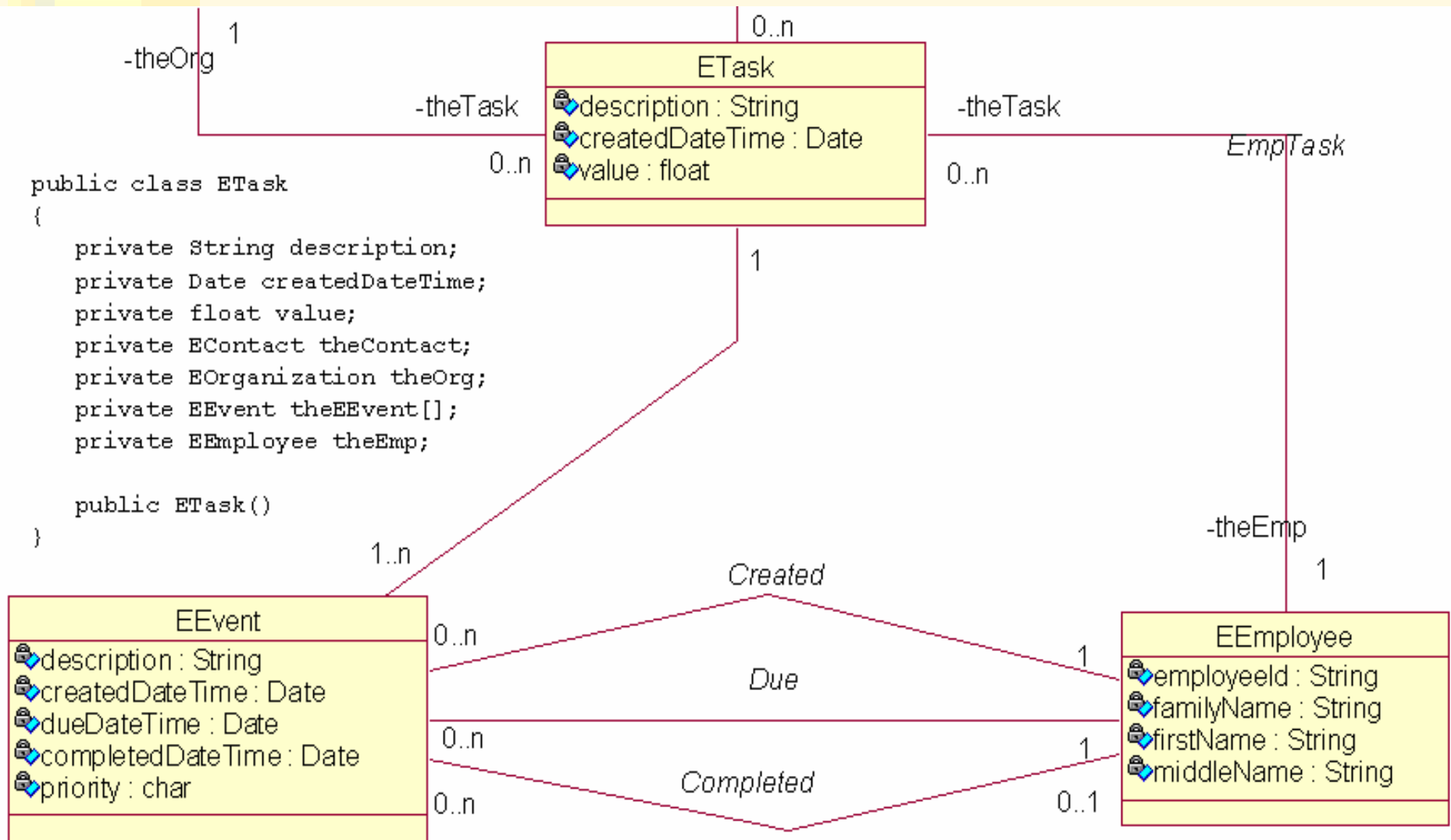
# *Specifying associations*

- *Naming associations*
  - *Recommendation – small letters; capitalizing the first letters of successive words (e.g. empTask)*
- *Naming association roles*
- *Determining multiplicity*
  - *Lower and/or upper multiplicity bounds can be omitted initially*
- *Rolenames for recursive associations*

# Example 4.8 – Contact Management (solution – 1)



# Example 4.8 – Contact Management (solution – 2)



# *Modeling aggregation and composition*

- *Four semantics for aggregation possible*
  - *ExclusiveOwns (e.g. Book has Chapter)*
    - *Existence-dependency*
    - *Transitivity*
    - *Asymmetry*
    - *Fixed property*
  - *Owns (e.g. Car has Tire)*
    - *No fixed property*
  - *Has (e.g. Division has Department)*
    - *No existence dependency*
    - *No fixed property*
  - *Member (e.g. Meeting has Chairperson)*
    - *No special properties except membership*

# *Discovering aggregation*

- *Discovered in parallel with discovery of associations*
- *The litmus test phrases*
  - *“has”*
  - *“is-part-of”*
- *Can relate more than two classes*

# *Specifying aggregation*

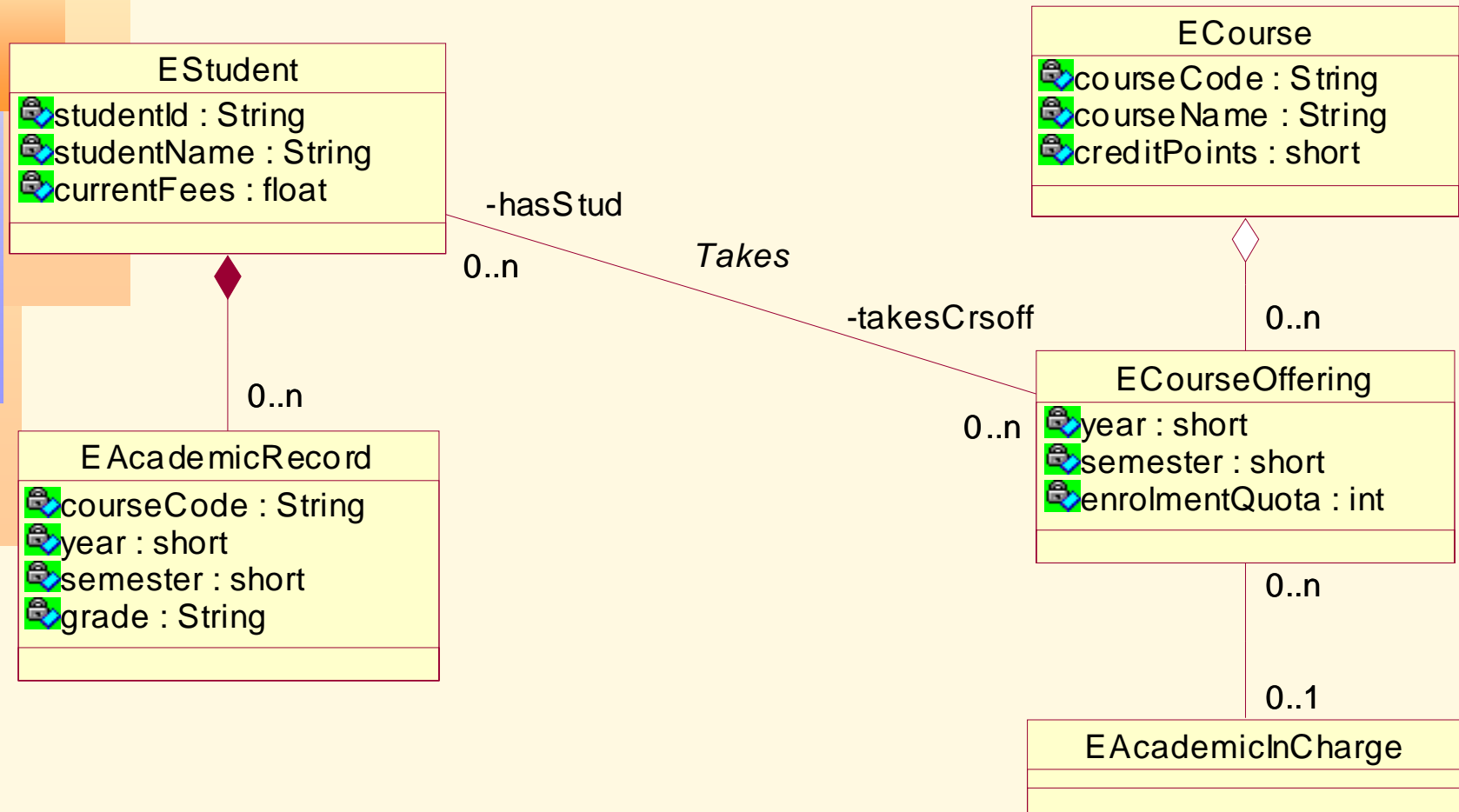
## ■ *UML supports*

- *Aggregation*
  - *By-reference semantics*
  - *Hollow diamond*
  - *Corresponds to Has and Member aggregations*
- *Composition*
  - *By-value semantics*
  - *Solid diamond*
  - *Corresponds to ExclusiveOwns and Owns aggregations*

# *Example 4.9 – University Enrolment*

- *Consider the following additional requirements:*
  - *The student's academic record to be available on demand*
  - *The record to include information about the student's grades in each course that the student enrolled in (and has not withdrawn without penalty)*
  - *Each course has one academic in charge of a course, but additional academics may also teach in it*
    - *There may be a different academic in charge of a course each semester*
    - *There may be different academics for each course each semester*

# Example 4.9 – University Enrolment (solution)



# *Modeling generalization*

- *Common features abstracted into a more generic class*
- *Subclasses **inherit** (reuse) superclass features*
- **Substitutability** – *subclass object is a legal value for a superclass variable*  
*(e.g. a variable holding `Fruit` objects can have an `Apple` object as its value)*
- **Polymorphism** – *the same operation can have different implementations in different classes*
- **Abstract operation** – *implementation provided in subclasses*
- **Abstract class** – *class with no direct instance objects*
  - *A class with an abstract operation is abstract*

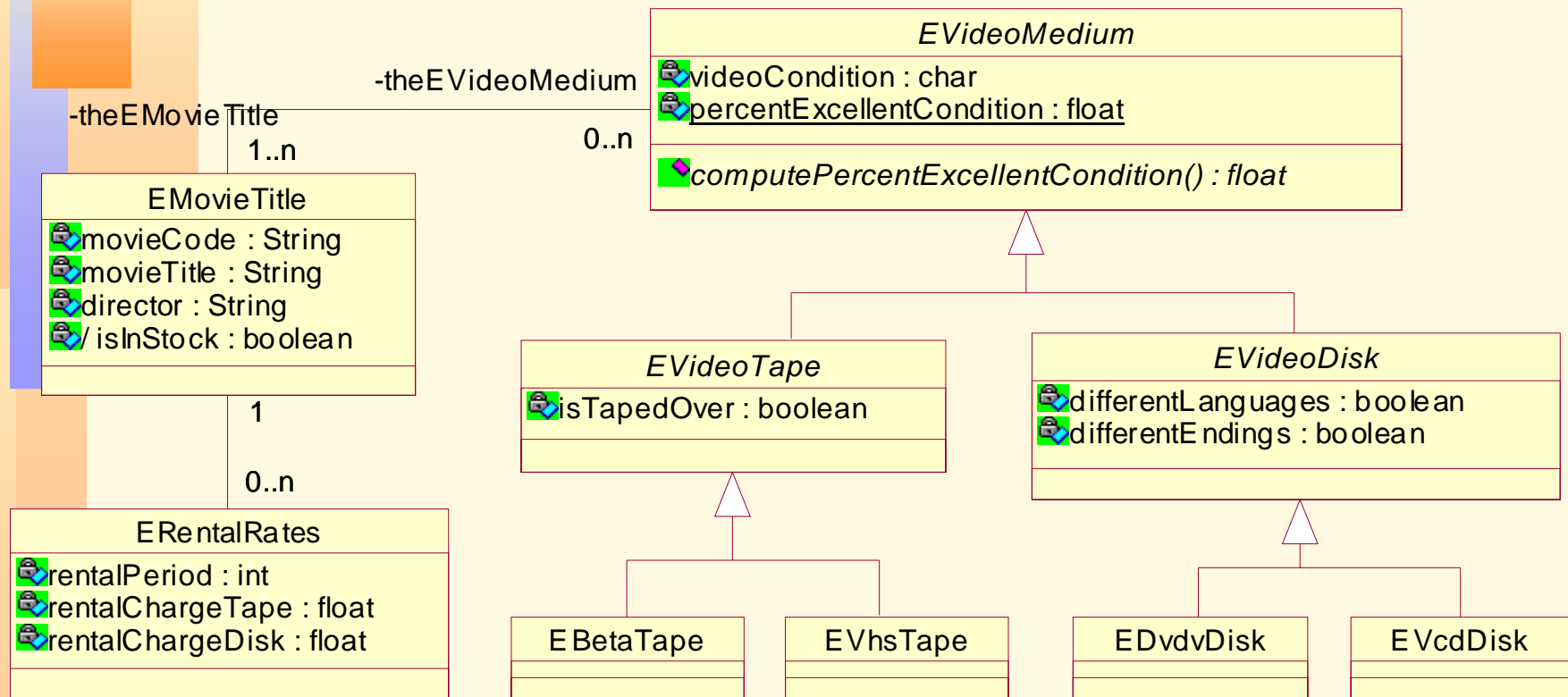
## *Discovering and specifying generalization*

- *Some discovered in parallel with discovery of associations*
- *The litmus test phrases*
  - *“can-be”*
  - *“is-a-kind-of”*
- *Multiple inheritance possible*
- *Solid line with an arrowhead pointing to the superclass*

## *Example 4.10 – Video Store*

- *The classes identified so far imply a generalization hierarchy rooted at the class `VideoMedium`*
- *Extend the model to include relationships between classes, and specify generalization relationships*
- *Assume that the Video Store needs to know if a `VideoTape` is a brand new tape or it was already taped over (this can be captured by an attribute `is_taped_over`)*
- *Assume also that the storage capacity of a `VideoDisk` allows holding multiple versions of the same movie, each in a different language or with different endings*

# Example 4.10 – Video Store (solution)



# *Modeling interfaces*

- *Interfaces*
  - *do not have attributes (except constants), associations or states*
  - *they only have operations, but all operations are implicitly public and abstract*
    - *operations are declared (i.e. turned into implemented methods) in classes which implement these interfaces.*
- *Interfaces do not have associations to classes but they may be targets of one-way associations from classes*
  - *this happens when an attribute that implements an association is typed with an interface, rather than with a class*
  - *the value of any such attribute will be a reference to some class that implements the interface*
- *An interface may have a generalization relationship to another interface*
  - *this means that an interface can extend another interface by inheriting its operations*

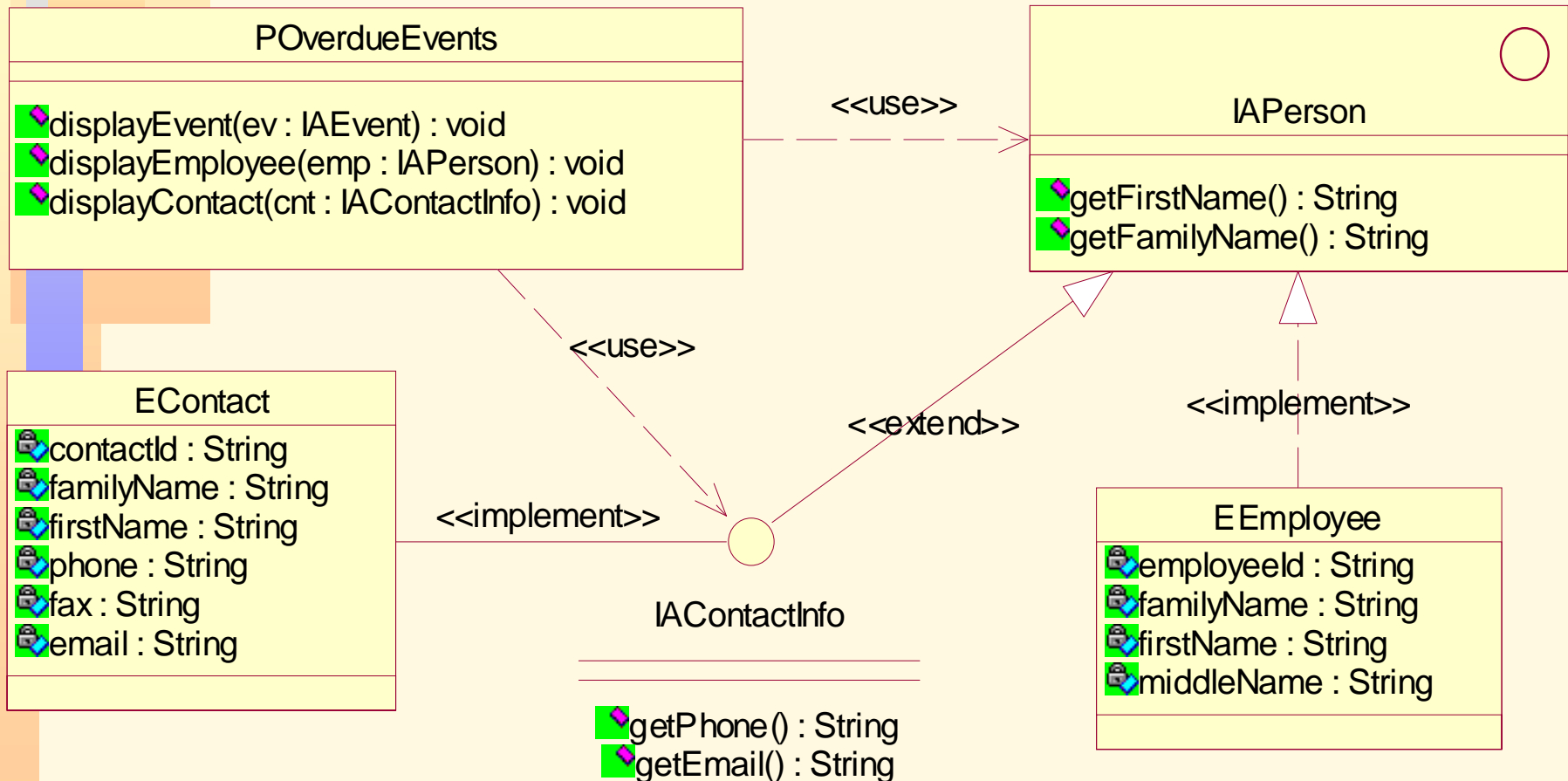
# *Discovering and specifying interfaces*

- *Interfaces are not discovered from the analysis of the application domain*
- *They are discovered based on design considerations*
  - *fundamental for enforcing architectural frameworks, such as the PCMEF framework*
  - *interface reveals only a limited portion of the behavior of an actual class*
- *Class that uses (requires) the interface can be indicated by a dashed arrow pointing to the interface*
  - *the arrow can be stereotyped with the keyword «use»*
- *Class that implements (realizes) the interface is indicated by a dashed lined with a triangular end*
  - *the line can be stereotyped with the keyword «implement»*

# Example 4-11 – Contact Management

- Consider classes *EContact* and *EEmployee*
  - some attributes in common (*firstName*, *familyName*)
  - operations that provide access to these attributes can be extracted into a single interface.
- There is a need to display information about overdue events to the screen
  - presentation-layer class has the responsibility to display a list of overdue events together with names of contacts and employees, as well as with the additional contact details (phone and email) to contacts
- Propose a model such that the presentation class uses one or more interfaces implemented by *EEmployee* and *EContact* to support part of the “display overdue events” functionality

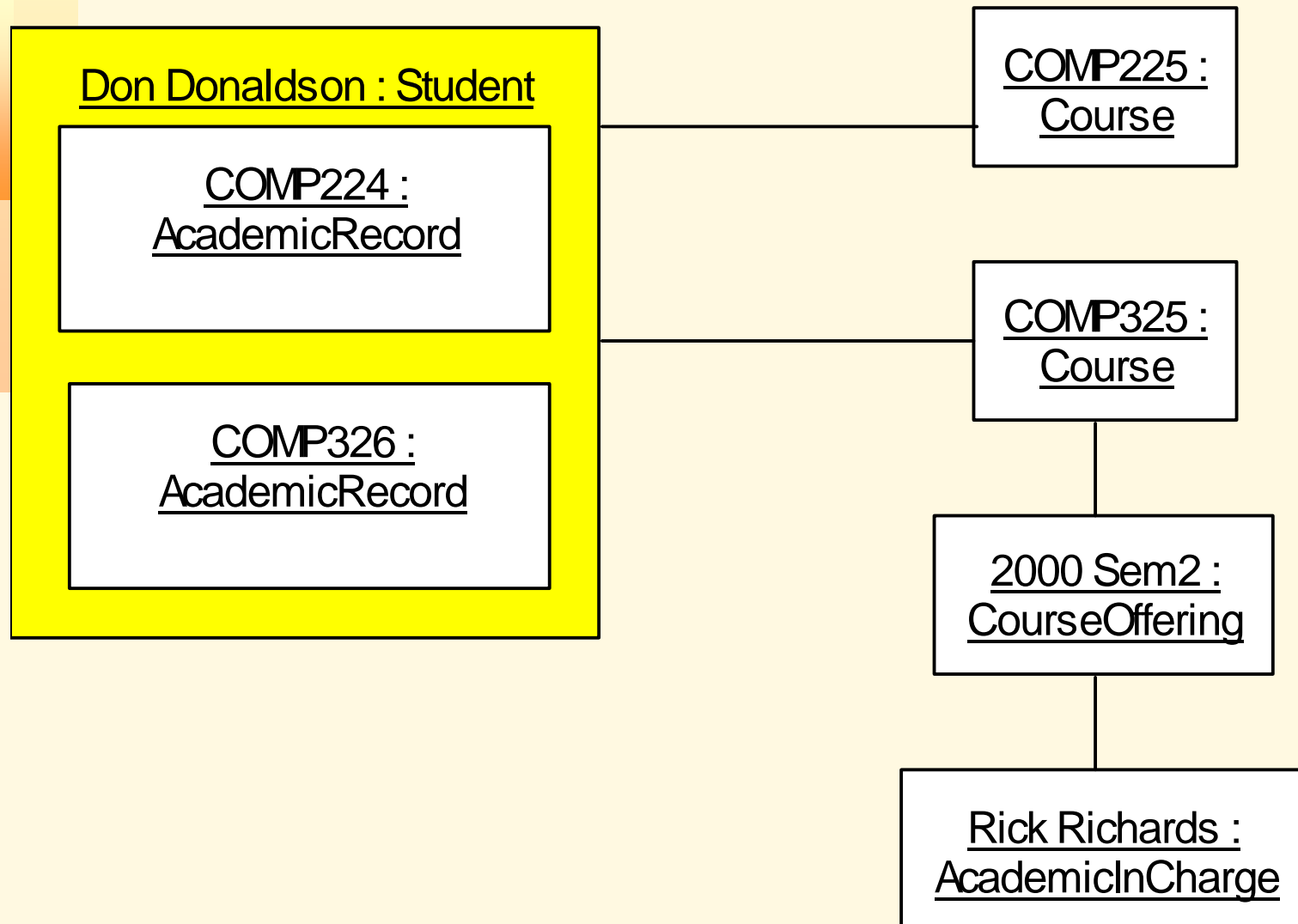
# Example 4-11 – Contact Management



# *Modeling and specifying objects*

- *Only to exemplify*
  - *To illustrate complex relationships between objects*
  - *To demonstrate changes to objects over time*
  - *To illustrate object collaboration*

# Example 4.12 – University Enrolment



# *Behavior specification*

- *Depicted in use cases*
- *Determines which classes are involved in execution of use cases*
  - *Main class operations identified*
  - *Message passing between objects captured*
  - *Control classes and boundary classes considered*
- *Computations modeled in Activity Diagrams*
- *Interactions modeled in Sequence Diagrams or Collaboration Diagrams*

# *Modeling use cases*

- *Complete piece of functionality*
  - *Main flow*
  - *Subflows*
  - *Alternate flows*
- *Piece of externally visible functionality*
- *Orthogonal piece of functionality*
- *Piece of functionality initiated by an actor*
- *Piece of functionality that delivers an identifiable value to an actor*

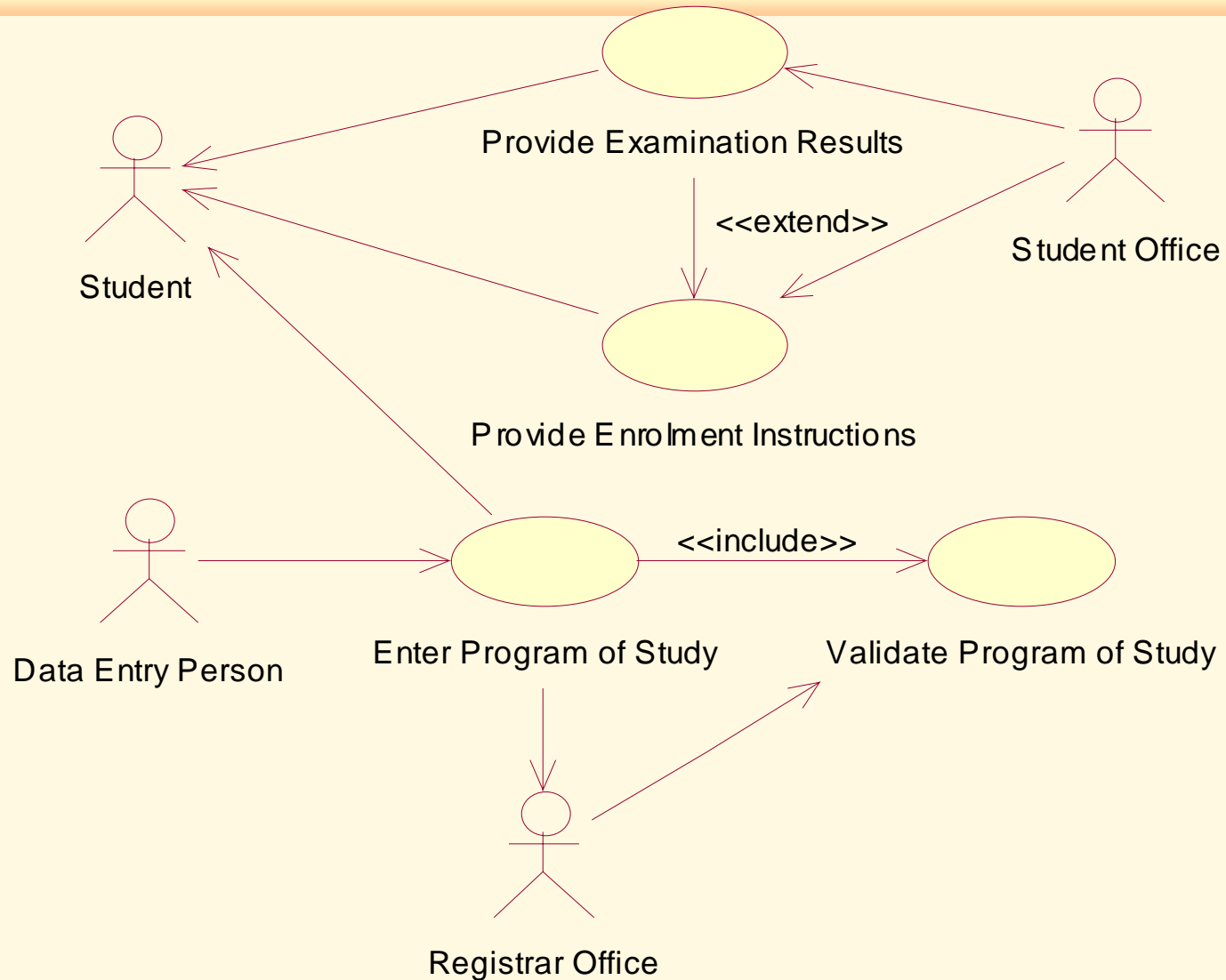
# *Discovering use cases*

- *Discovered from*
  - *Requirements identified in the Requirements Document*
  - *Actors and their purpose in the system*
- *Questions to ask*
  - *What are the main tasks performed by each actor?*
  - *Will an actor access or modify information in the system?*
  - *Will an actor inform the system about any changes in other systems?*
  - *Should an actor be informed about unexpected changes in the system?*

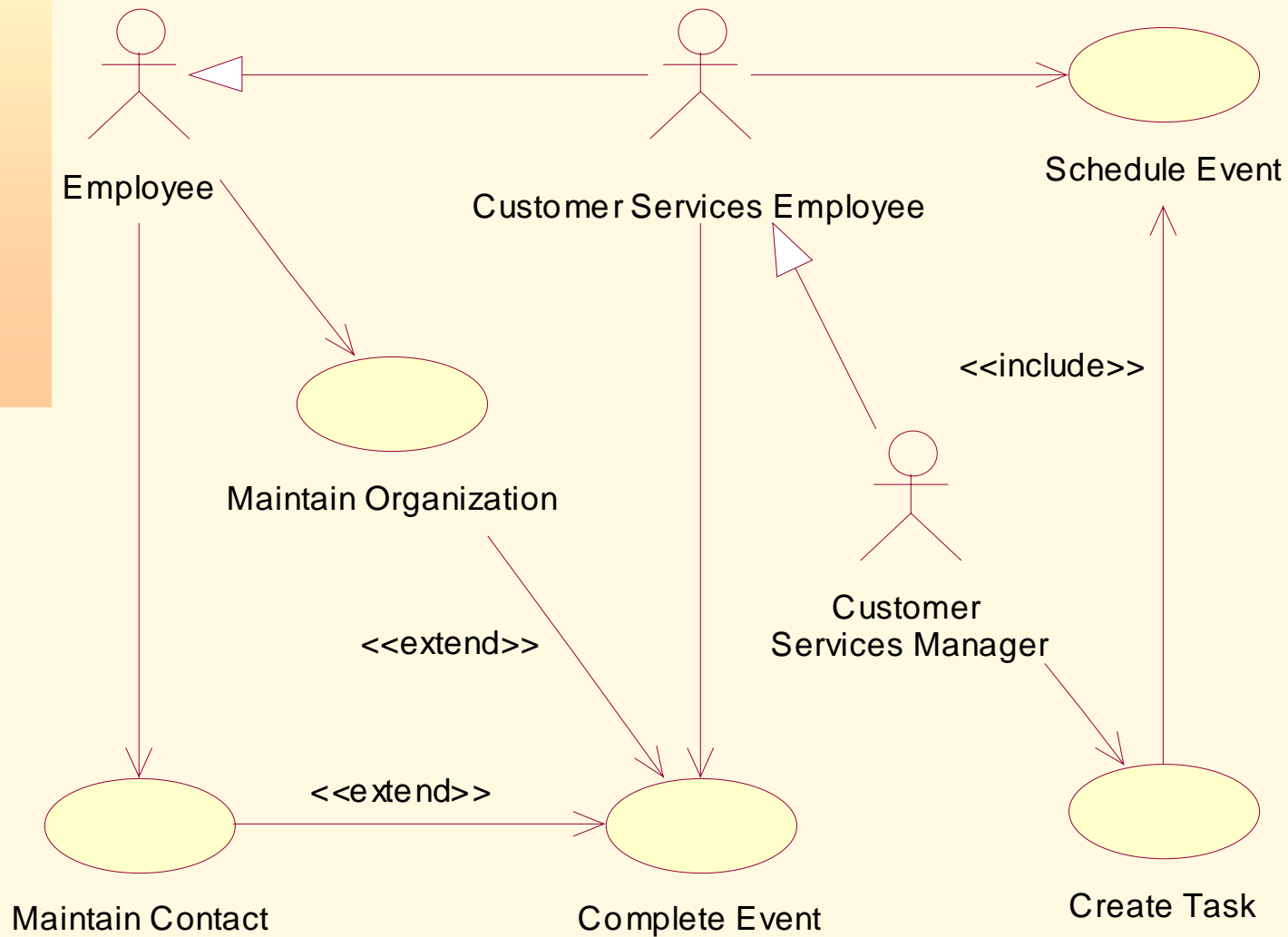
# *Specifying use cases*

- *Actors*
- *Use cases*
- *Four kinds of relationships*
  - *Association (between actor and use case)*
  - *Include (stereotyped with the word: «include»)*
    - *Included use case is always necessary for the completion of the activating use case*
  - *Extend (stereotyped with the word: «extend»)*
    - *Another use is activated occasionally at specific extension point*
  - *Generalization*
- *Relationships to be used with restraint*

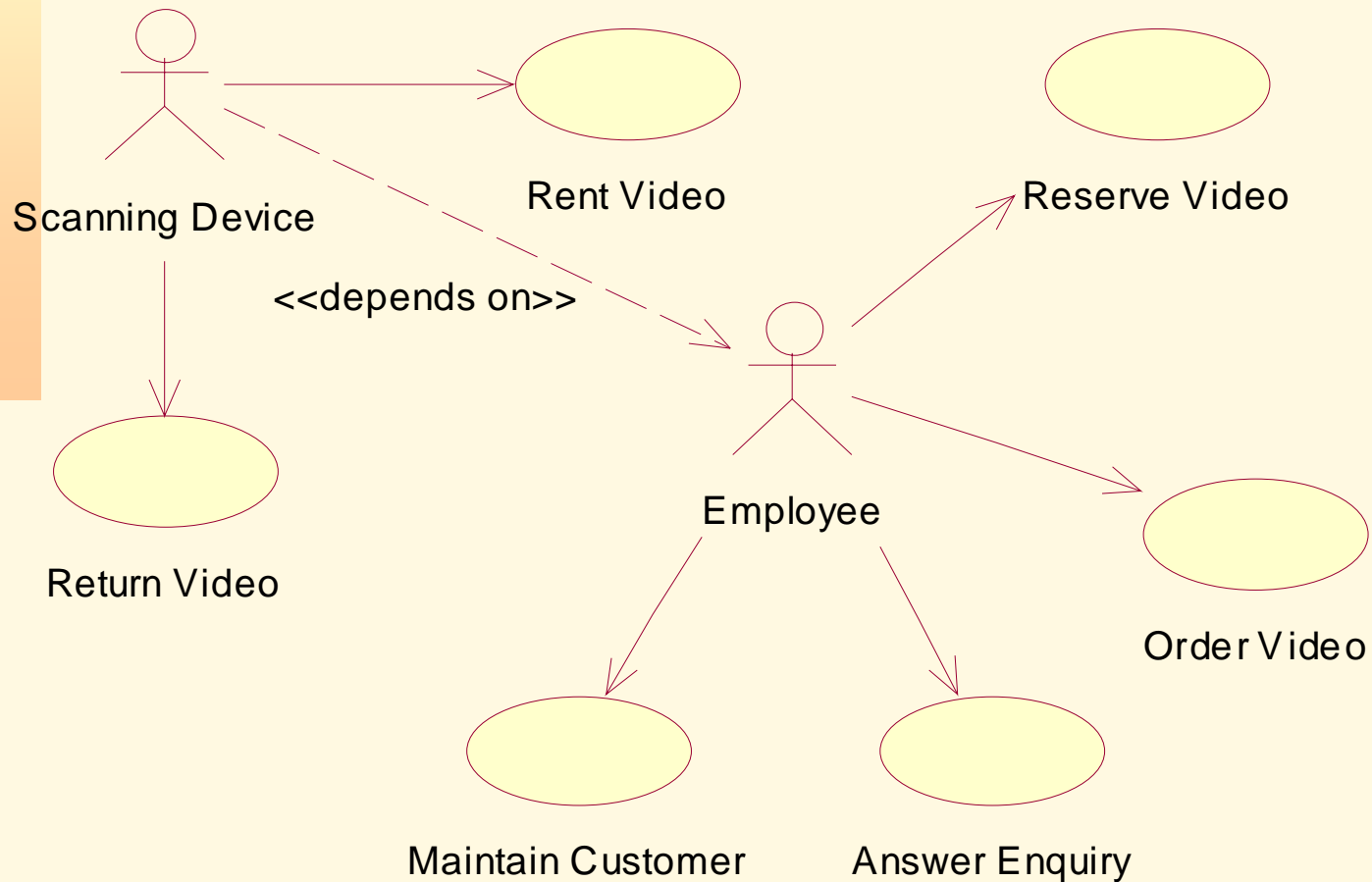
# Example 4.13 – University Enrolment



# Example 4.14 – Contact Management



# Example 4.15 – Video Store



## *Example 4.15 – Video Store (Rent Video)*

<i>Brief Description</i>	<p>A customer wishes to rent a video tape or disk that is picked from the store's shelves or that has been previously reserved by the customer.</p> <p>Provided the customer has a non-delinquent account, the tape is rented out once the payment has been received.</p> <p>If the tape is not returned in a timely fashion, an overdue notice is mailed to the customer.</p>
<i>Actors</i>	Employee, Scanning Device
<i>Preconditions</i>	<p>Video tape or disk is available to be hired.</p> <p>Customer has a membership card. Scanner devices work correctly.</p> <p>Employee at the front desk knows how to use the system.</p>

## *Example 4.15 – Video Store (Rent Video)*

### *Main Flow*

A customer may ask an employee about video availability (including a reserved video) or may pick a tape or disk from the shelves. The video and the membership card are scanned and any delinquent or overdue details are brought up for the employee to query the customer about. If the customer does not have a delinquent rating, then he/she can hire up to a maximum of eight videos. However, if the rating of the customer is 'unreliable' then a deposit of one rental period for each tape or disk is requested. Once the amount payable is received, the stock is updated and the tapes and disks are handed out to the customer together with the rental receipt. The customer pays by cash, credit card or electronic transfer. Each rental record stores (under the customer's account) the check-out and due-in dates together with the identification of the employee. A separate rental record is created for each video hired.

The use case will generate an overdue notice to the customer if the video has not been returned within two days of the due date, and a second notice after another two days (and at that time the customer is noted as 'delinquent').

## *Example 4.15– Video Store (Rent Video)*

<i>Alternative Flows</i>	<p>A customer does not have a membership card. In this case, the 'Maintain Customer' use case may be activated to issue a new card.</p> <p>An attempt to rent too many videos.</p> <p>No videos can be rented because of the customer's delinquent rating.</p> <p>The video medium or membership card cannot be scanned because of damage to them.</p> <p>The electronic transfer or credit card payment is refused.</p>
<i>Postconditions</i>	<p>Videos are rented out and the database is updated accordingly.</p>



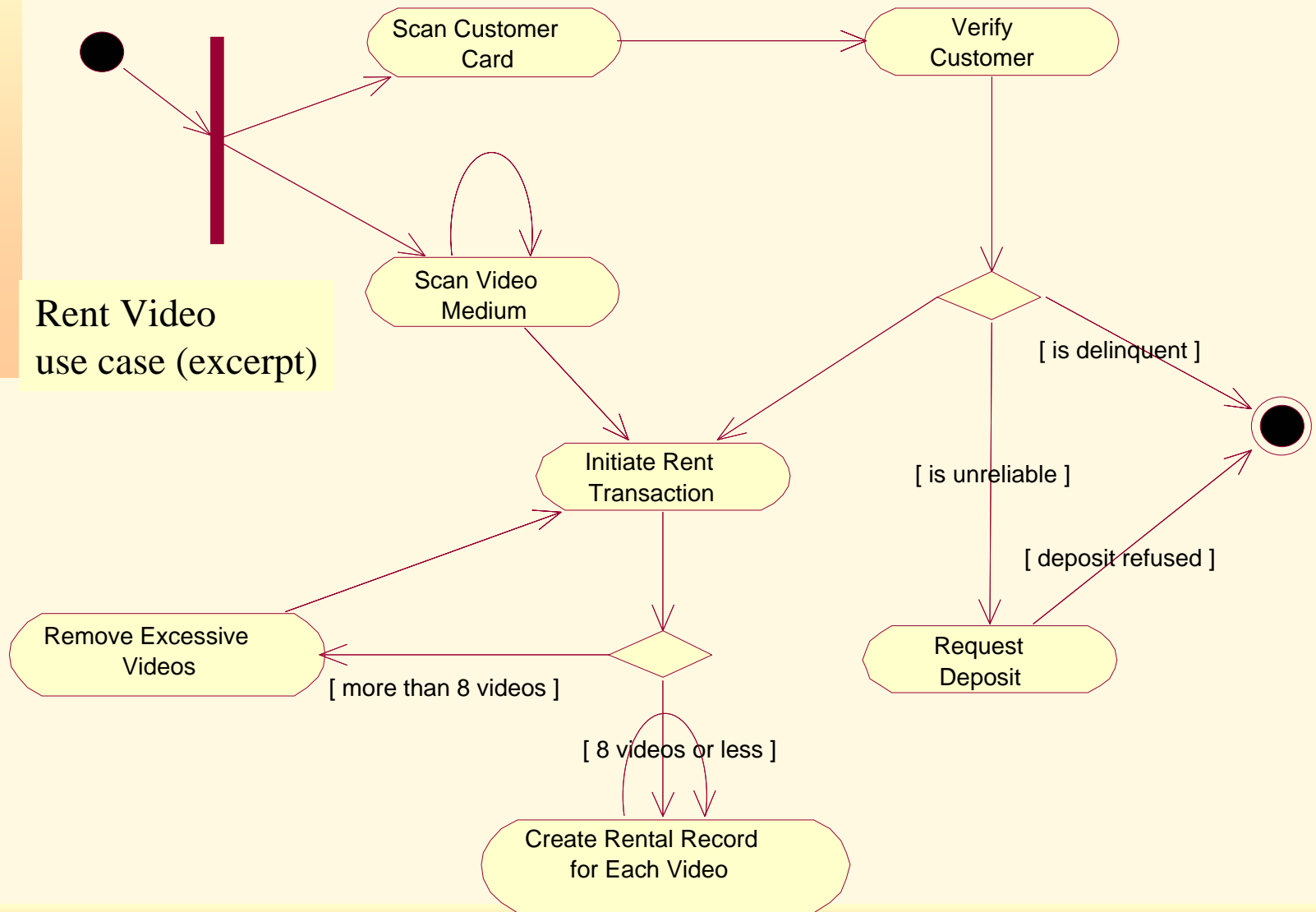
# *Modeling activities*

- *Activity Diagrams*
- *Flow of logic*
  - *Sequential control*
  - *Concurrent control*
- *Can be used at different levels of abstraction*
  - *To define execution of a use case*
  - *To define execution of an operation*

# *Discovering and specifying actions*

- *The execution proceeds from one **action state** to the next*
- *An **action state** completes when its computation is completed*
- ***Actions** can be discovered from the narrative specifications of **use cases***
- *Actions are connected by **transition lines***
- ***Synchronization** bars (fork and re-join)*
- ***Branch** diamonds (branch and merge)*
- ***External events** not normally modeled on activity graphs*

# Example 4.17 – Video Store (solution)



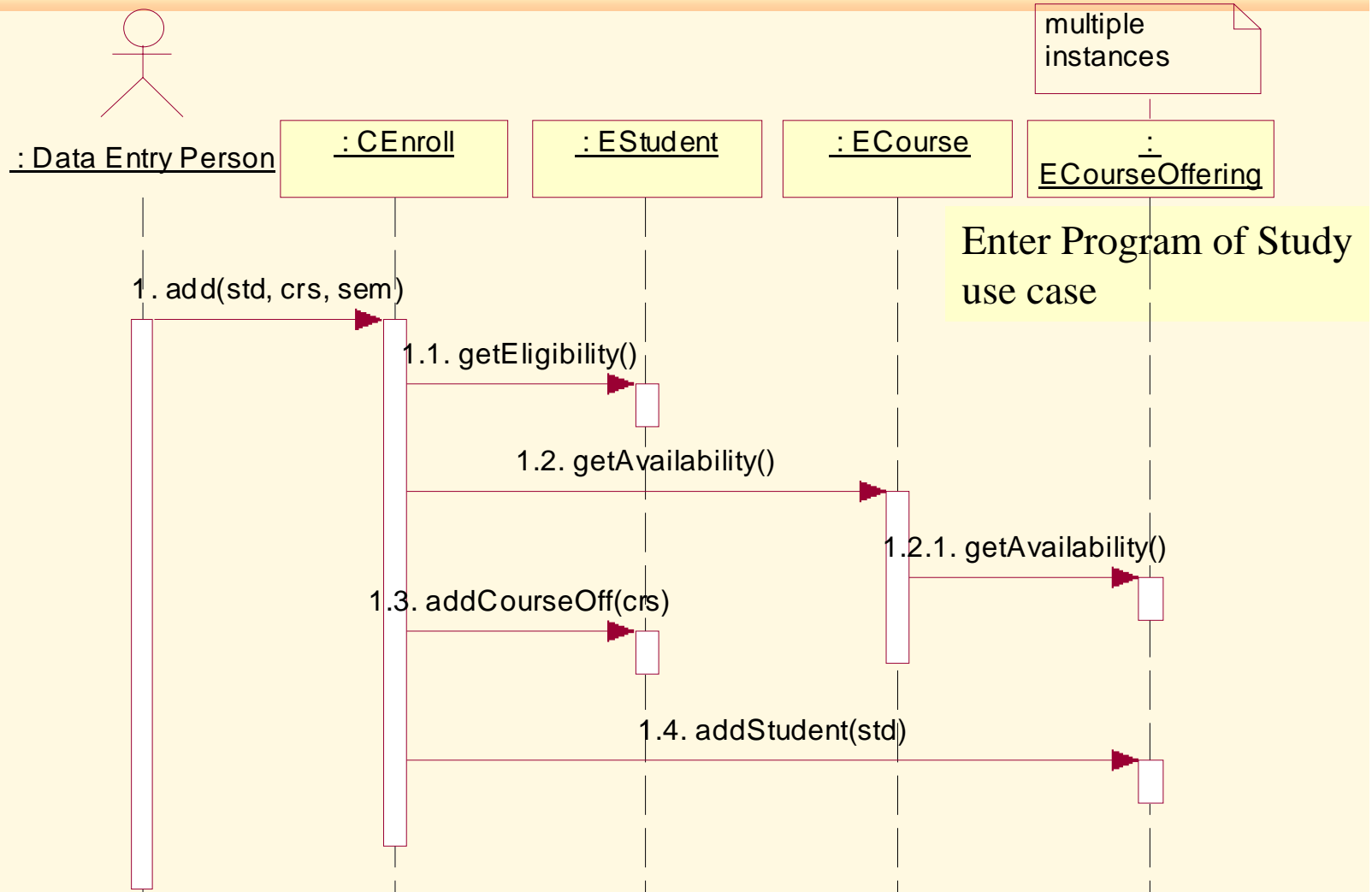
# *Modeling interactions*

- *Sequence Diagrams*
  - *Show an exchange of messages between objects arranged in a time sequence*
  - *More useful in analysis*
- *Collaboration Diagrams*
  - *Emphasize the relationships between objects along which the messages are exchanged*
  - *More useful in design*
- *Can be used to determine operations in classes*

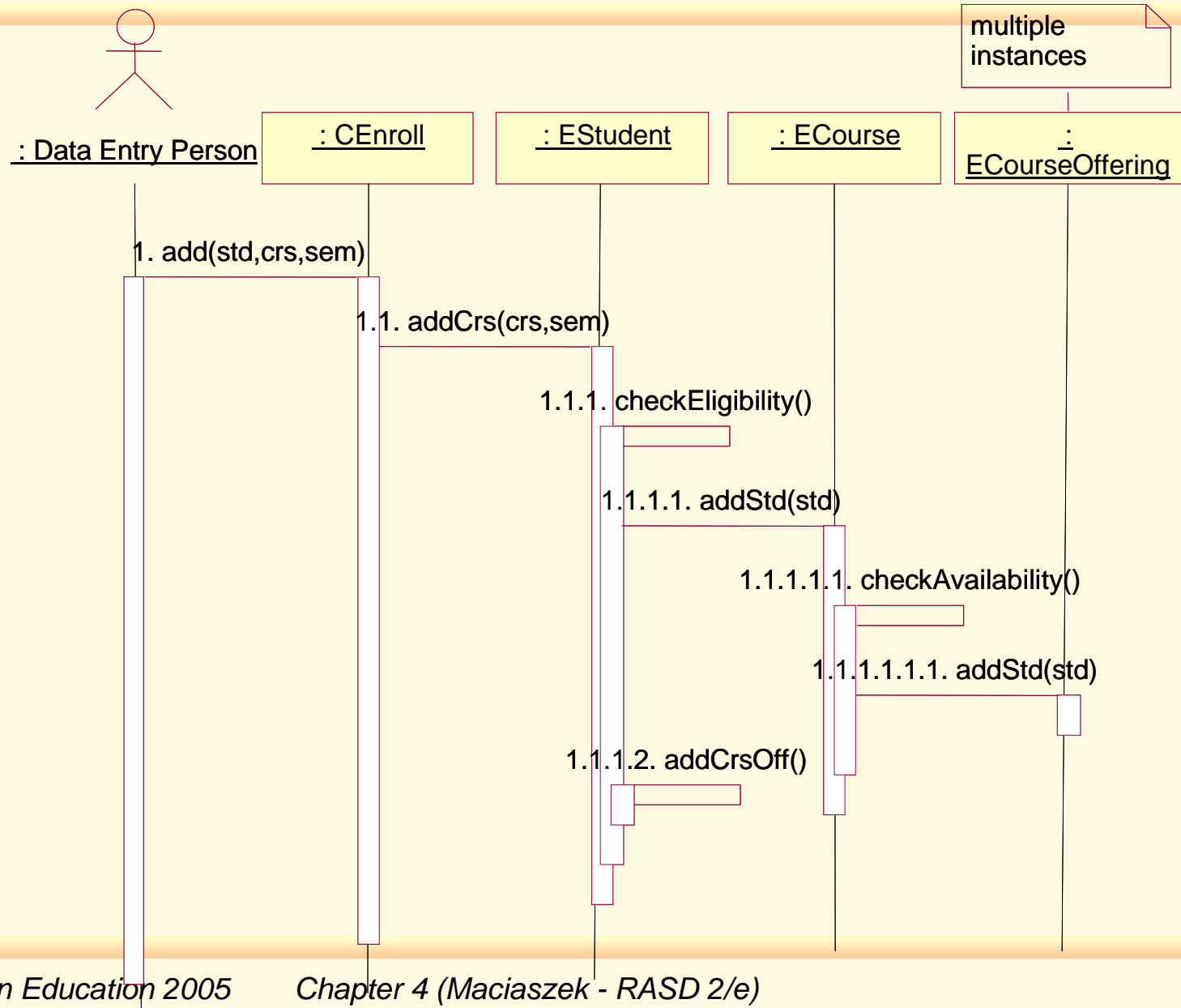
# *Message sequences*

- *Actions in Activity Diagrams are mapped to messages to Sequence Diagrams*
- *Message can be a:*
  - *Signal*
    - *Denotes asynchronous inter-object communication*
    - *The sender continues executing after sending the signal message*
  - *Call*
    - *Denotes synchronous invocation of an operation*
    - *The return message can return some values to the caller or it can just acknowledge that the operation completed*

# Example 4.18 – UE (centralized interaction)



# Example 4.18 – UE (distributed interaction)



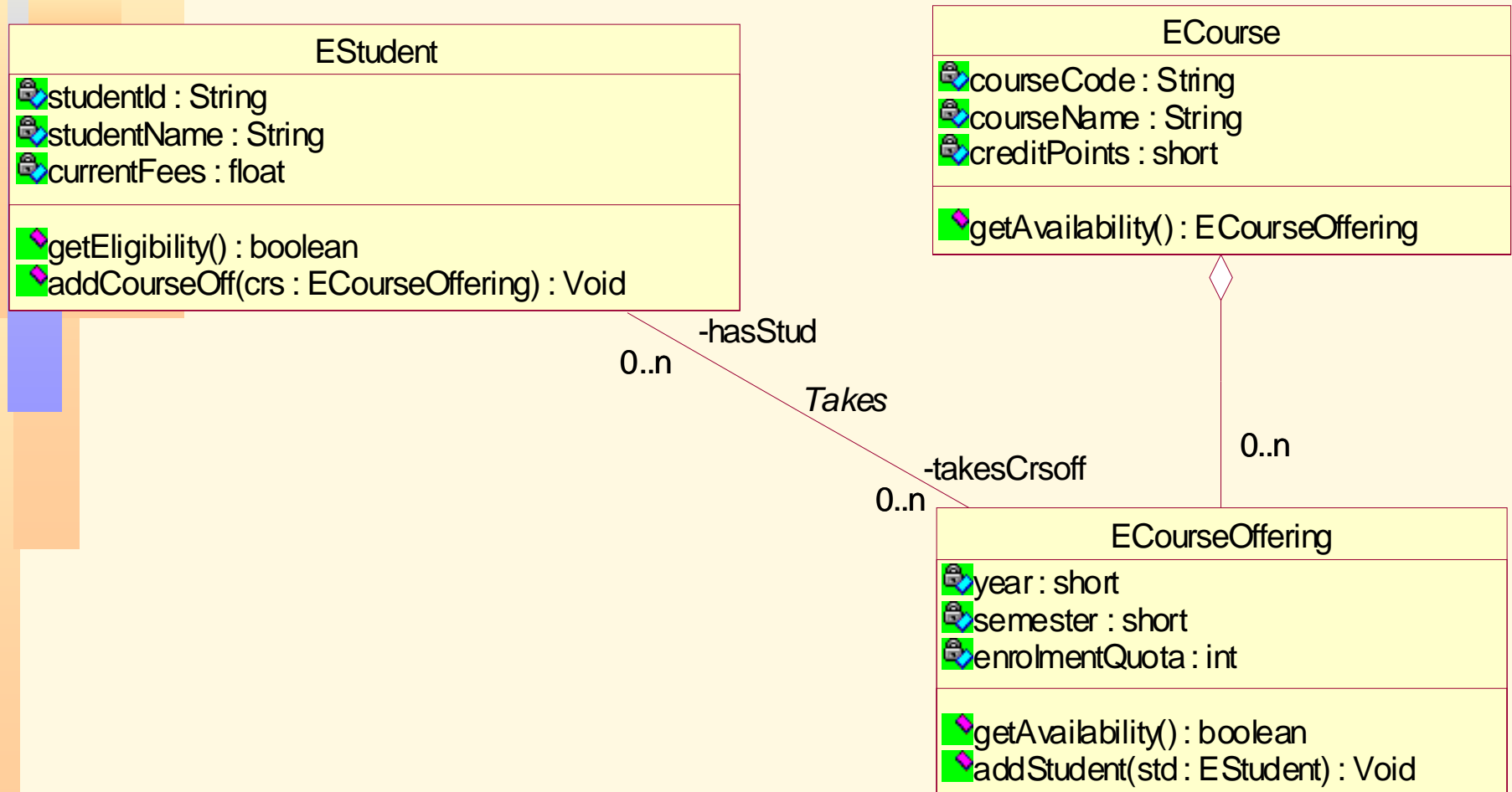
# *Modeling public interfaces*

- *Determined by the set of operations that the class offers as its service*
- *In analysis*
  - *Signature of each operation is defined*
    - *Operation name*
    - *List of formal arguments*
    - *Return type*
- *In design*
  - *Algorithm of a method that implements the operation is defined*
- *Operation can have*
  - *Instance scope*
  - *Class (static) scope (\$ in front of operation name)*

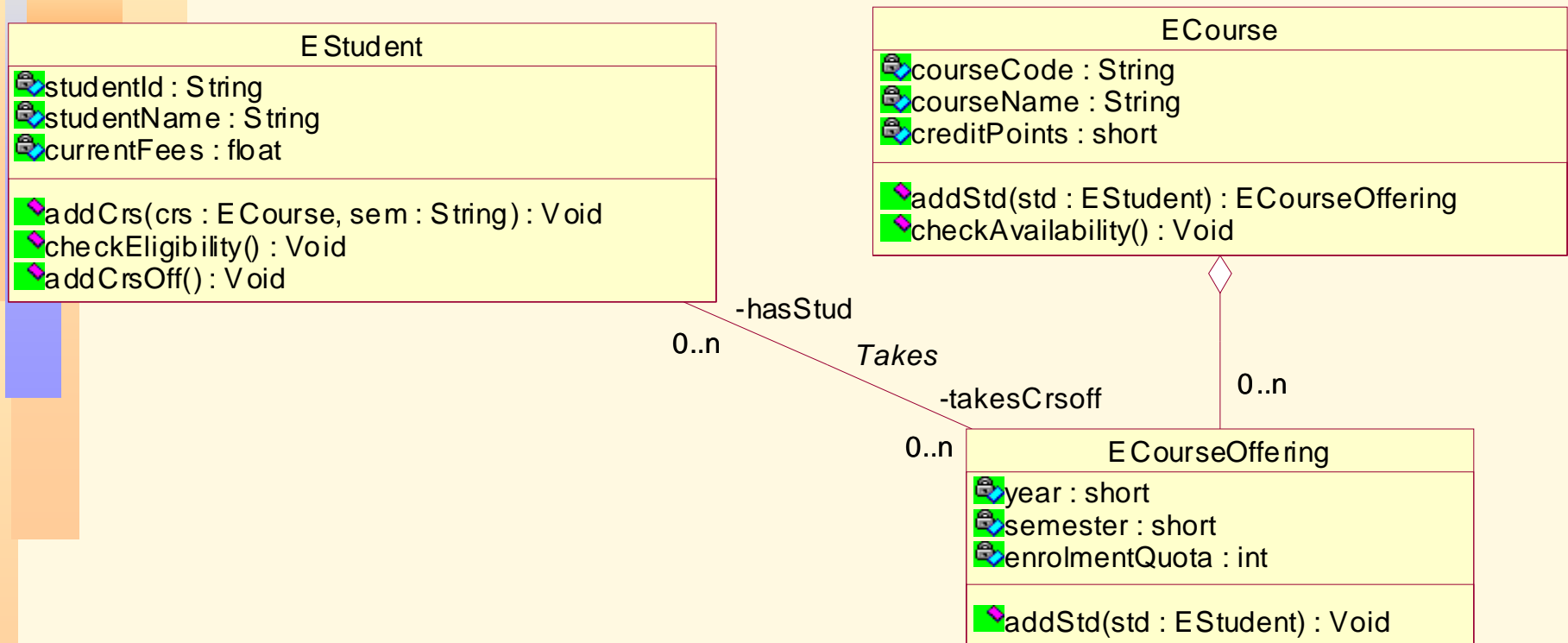
# *Discovering class operations*

- *From Sequence Diagrams*
  - *Message to an object must be serviced by an operation in that object*
- *From expected object responsibilities, including the CRUD operations*
  - *Create – a new object instance*
  - *Read – the state of an object*
  - *Update – the state of an object*
  - *Delete – i.e. destroy itself*

# Example 4.19 – UE (centralized interaction)



# Example 4.19 – UE (distributed interaction)



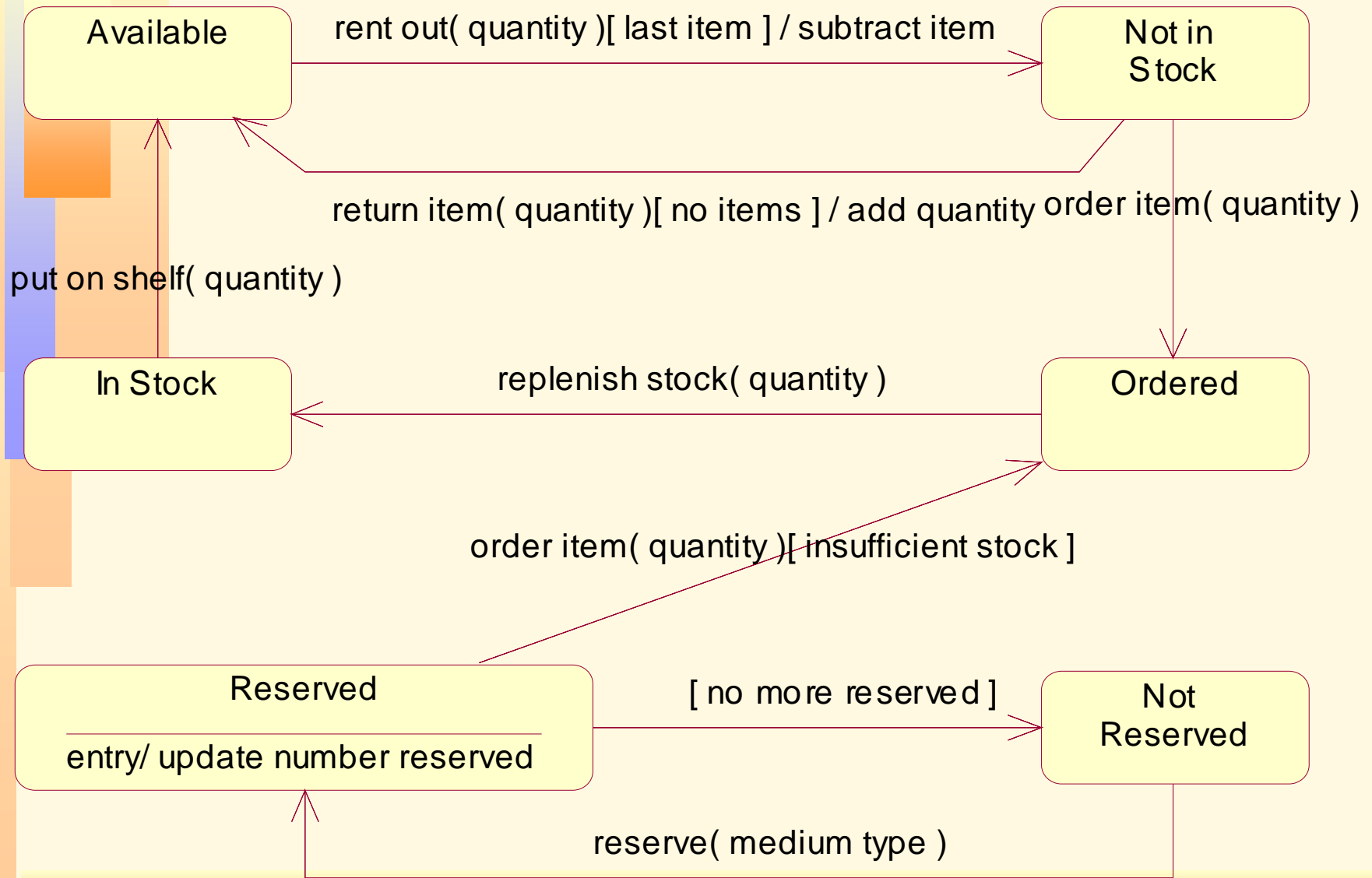
# *State change specifications*

- *Statechart Diagrams*
- *For each class that exhibits an interesting dynamic behavior*
- *Changes to some attributes signify state changes*

# *Specifying object states*

- *State transition fires when a certain event occurs or a certain condition is satisfied*
  - *transition line does not have to be labeled with an event name*
  - *condition itself (written in square brackets) can fire the transition*
- *Transition can be triggered by*
  - *Signal event*
  - *Call event*
  - *Change event*
  - *Time event*

# Example 4.19 – Video Store



# Summary

- *The critical importance of **architecture** in system development (BCED → PCMEF)*
- ***State specifications** describe the IS world from the static perspective of classes, their attribute content and their relationships*
  - *There are many methods of class discovery*
  - *Class diagrams visualize classes and relationships : associations, aggregations and generalizations*
- ***Behavioral specifications** describe the IS world from the operational (functional) perspective*
  - *Use case diagrams provide simple visualization – each use case is given narrative specification*
  - *Other behavioral diagrams include activity diagrams, interactions diagrams, and addition of operations to classes.*
- ***State change specifications** describe the IS world from the dynamic perspective*
  - *Statechart diagrams allow modeling of state changes*