Object Oriented Analysis - OOA

- La **fase di OOA** definisce, secondo un approccio ad oggetti, COSA un prodotto software deve fare (mentre la fase di OOD definisce, sempre secondo un approccio ad oggetti, COME un prodotto software deve fare quanto specificato in fase di OOA)
- OOA e OOD devono fornire, ciascuno dal proprio punto di vista, una rappresentazione corretta, completa e consistente:
  - degli aspetti statici e strutturali relativi ai dati (*modello dei dati*)
  - degli aspetti funzionali del sistema (*modello comportamentale*)
  - degli aspetti di "controllo" e di come le funzioni del modello comportamentale modificano i dati introdotti nel modello dei dati (*modello dinamico*)

Metodi di OOA

- Un metodo di OOA definisce l'insieme di procedure, tecniche e strumenti per un approccio sistematico alla gestione e allo sviluppo della fase di OOA
- L'input di un metodo di OOA è costituito dall'*insieme dei requisiti utente* (contenuti nel documento di analisi dei requisiti)
- L'output di un metodo di OOA è costituito dall'*insieme dei modelli del sistema* che definiscono la specifica del prodotto software (e che sono anch'essi contenuti nel documento di analisi dei requisiti)
- I metodi di OOA fanno principalmente uso di notazioni visuali (diagrammi), ma possono essere affiancati da metodi tradizionali per la definizione di requisiti di sistema di tipo testuale (in linguaggio naturale strutturato)
- Lo sviluppo dei modelli di OOA non è un processo sequenziale (prima modello dei dati, poi modello comportamentale, infine modello dinamico)
- La costruzione dei modelli avviene in parallelo, e ciascun modello fornisce informazioni utili per gli altri modelli
- I metodi di OOA fanno uso di un approccio iterativo, con aggiunta di dettagli per raffinamenti successivi (iterazioni)

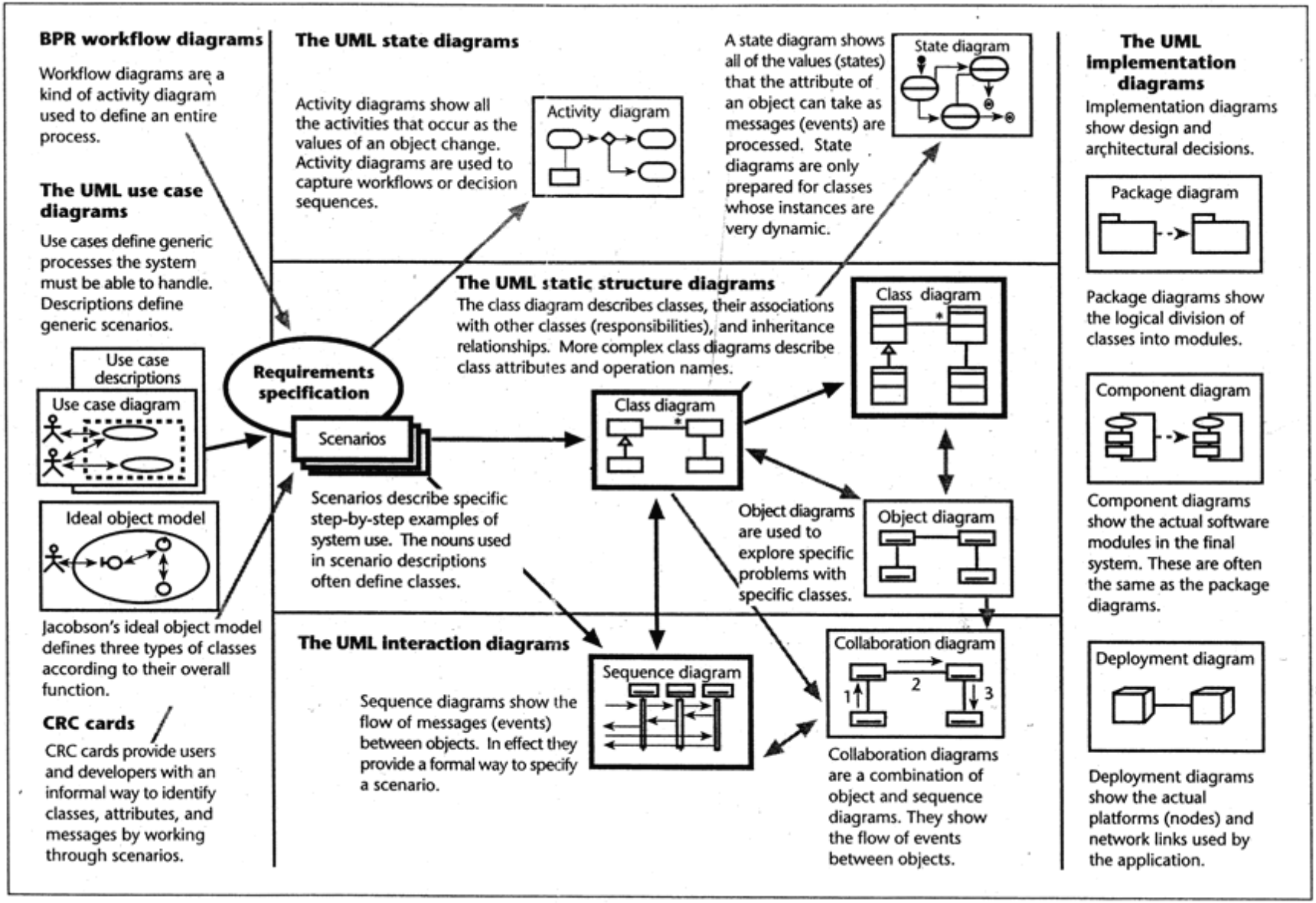
Alcuni metodi di OOA (e OOD)

- Catalysis: metodo OO particolarmente indicato per lo sviluppo di sistemi software a componenti distribuiti.
- Objectory: metodo ideato da I. Jacobson che fonda lo sviluppo di prodotti software ad oggetti sull'individuazione dei casi d'uso utente (*use case driven*).
- Shlaer/Mellor: metodo OO particolarmente indicato per lo sviluppo di sistemi software *real-time*.
- OMT (Object Modeling Technique): metodo sviluppato da J. Rumbaugh basato su tecniche di modellazione del software iterative. Pone in particolare risalto la *fase di OOA*.
- Booch: metodo basato su tecniche di modellazione del software iterative. Pone in particolare risalto la *fase di OOD*.
- Fusion: metodo sviluppato dalla HP a metà degli anni novanta. Rappresenta il primo tentativo di standardizzazione per lo sviluppo di software orientato agli oggetti. Si basa sulla fusione dei metodi OMT e Booch.

Notazioni per OOA (e OOD)

- Ciascun metodo di OOA (e OOD) fa uso di una propria notazione per la rappresentazione dei modelli del sistema
- Al fine di unificare le notazioni per i metodi di OOA e OOD è stato introdotto il linguaggio **UML** (**Unified Modeling Language**), adottato nel 1997 come standard OMG (Object Management Group)
- UML è un linguaggio standard per la descrizione di sistemi software (orientati agli oggetti). Si compone di *nove formalismi di base* (diagrammi con semantica e notazione data) e di un insieme di *estensioni*.
- UML è un linguaggio di descrizione, non è un metodo né definisce un processo
- *Unified Software Development Process* (in breve *Unified Process*) è un tentativo di standardizzazione di processo di sviluppo di sistemi orientati agli oggetti basato sull'uso di UML.

Diagrammi UML



Formalismi UML

I *nove formalismi di base* dello UML sono:

1. Use case diagram

   evidenziano la modalità (caso d'uso) con cui gli utenti (attori) utilizzano il sistema. Possono essere usati come supporto per la definizione dei requisiti utente.

2. Class diagram

   consentono di rappresentare le classi con le relative proprietà (attributi, operazioni) e le associazioni che le legano.

3. State diagram

   rappresentano il comportamento dinamico dei singoli oggetti di una classe in termini di stati possibili e transizioni di stato per effetto di eventi.

4. Activity diagram

   sono particolari state diagram, in cui gli stati rappresentati rappresentano azioni in corso di esecuzione. Sono particolarmente indicati per la produzione di modelli di work-flow.

Formalismi UML (2)

5. Sequence diagram

   evidenziano le interazioni (messaggi) che oggetti di classi diverse si scambiano nell'ambito di un determinato caso d'uso, ordinate in sequenza temporale. A differenza dei diagrammi di collaborazione, non evidenziano le relazioni tra oggetti.

6. Collaboration diagram

   descrivono le interazioni (messaggi) tra oggetti diversi, evidenziando le relazioni esistenti tra le singole istanze.

7. Object diagram

   permettono di rappresentare gli oggetti e le relazioni tra essi nell'ambito di un determinato caso d'uso.

8. Component diagram

   evidenziano la strutturazione e le dipendenze esistenti tra componenti software.

9. Deployment diagram

   evidenziano le configurazioni dei nodi elaborativi di un sistema real-time ed i componenti, processi ed oggetti assegnati a tali nodi.

Modello dei dati

- Rappresenta da un punto di vista statico e strutturale l'organizzazione logica dei dati da elaborare
- Le strutture dati sono definite mediante lo stato degli oggetti, che viene determinato dal valore assegnato ad attributi e associazioni
- Il modello dei dati viene specificato mediante il formalismo dei *class diagram* che permette di definire:
  - classi
  - attributi di ciascuna classe
  - operazioni di ciascuna classe
  - associazioni tra classi
- Il modello dei dati è di fondamentale importanza, visto che, secondo l'approccio ad oggetti, un sistema software è costituito da un insieme di oggetti (classificati) che collaborano

Modello dei dati (2)

- Il modello dei dati viene costruito in modo *iterativo* ed *incrementale*
- Si tratta di un processo creativo, in cui giocano un ruolo importante sia l'esperienza dell'analista che la comprensione del dominio applicativo
- Durante la fase iniziale di costruzione del modello dei dati occorre concentrarsi sulle cosiddette ***entity classes***, ovvero quelle classi che definiscono il dominio applicativo e che sono rilevanti per il sistema
- Le *control classes* (che gestiscono la "logica" del sistema) e *boundary classes* (che rappresentano l'interfaccia utente) vengono introdotte successivamente, usando le informazioni del *modello comportamentale*
- Le operazioni di ciascuna classe vengono identificate a partire dal *modello comportamentale*, per cui vengono inizialmente trascurate

Approcci per l'identificazione delle classi

- *Noun phrase*
- *Common class patterns*
- *Use case driven*
- *CRC*
- *Mixed*

Approccio *noun phrase*

- Una frase nominale (*noun phrase*) è una frase in cui il sostantivo ha una prevalenza sulla parte verbale (sono frasi di tipo *assertivo*)
- I sostantivi delle frasi nominali usate per la stesura dei requisiti utente sono considerati ***candidate classes***
- La lista delle *candidate classes* viene suddivisa in tre gruppi:
  - *Irrelevant* (non appartengono al dominio applicativo e quindi possono essere scartate)
  - *Relevant* (evidenziano caratteristiche di *entity classes*)
  - *Fuzzy* (non si hanno sufficienti informazioni per classificarle come *relevant* o *irrelevant*, vanno analizzate successivamente)
- Si assume che l'insieme dei requisiti utente sia completo e corretto

Approccio *common class patterns*

- Basato sulla *teoria della classificazione*
- Le *candidate classes* vengono identificate a partire da gruppi (*pattern*) di classi predefinite:
  - *Concept* (es. *Reservation*)
  - *Events* (es. *Arrival*)
  - *Organization* (es. *AirCompany*)
  - *People* (es. *Passenger*)
  - *Places* (es. *TravelOffice*)
- Non è un approccio sistematico, ma può rappresentare una utile guida
- A differenza dell'approccio *noun phrase*, non si concentra sul documento dei requisiti utente
- Può causare problemi di interpretazione dei nomi delle classi

Approccio *use case driven*

- Si assume che:
  - Siano già stati sviluppati gli *use case diagram* (e possibilmente anche i *sequence diagram* più significativi)
  - Per ogni *use case* sia fornita una descrizione testuale dello scenario di funzionamento
- Simile all'approccio *noun phrase* (si considera l'insieme degli *use case* come insieme dei requisiti utente)
- Si assume che l'insieme degli *use case* sia completo e corretto
- Approccio *function-driven* (o *problem-driven* secondo la terminologia object oriented)

Approccio *CRC*

- L'approccio CRC (Class - Responsibility – Collaborators) è basato su riunioni in cui si fa uso di apposite *card*
- Ciascuna *card* rappresenta una classe, e contiene tre compartimenti, che identificano:
  - Il nome della classe
  - Le responsabilità assegnate alla classe
  - Il nome di altre classi che collaborano con la classe
- Le classi vengono identificate analizzando come gli oggetti collaborano per svolgere le funzioni di sistema
- Approccio utile per
  - *Verifica di classi* identificate con altri metodi
  - *Identificazione di attributi e operazioni* di ciascuna classe

CLASS Elevator Controller
RESPONSIBILITY
1. Turn on elevator button
2. Turn off elevator button
3. Turn on fl oor button
4. Turn off fl oor button
5. Open elevator doors
6. Close elevator doors
7. Move elevator one fl oor up
8. Move elevator one fl oor down
COLLABORATION
1. Class **Elevator Button**
2. Class **Floor Button**
3. Class **Elevator**

Approccio *mixed*

- Basato su elementi presenti in ciascuno degli approcci precedenti
- Un possibile scenario potrebbe essere il seguente:
  1. L'insieme iniziale delle classi viene identificato in base all'esperienza dell'analista, facendosi eventualmente guidare dall'approccio *common class patterns*
  2. Altre classi possono essere aggiunte usando sia l'approccio *noun phrase* che l'approccio *use case driven* (se gli *use case diagram* sono disponibili)
  3. Infine l'approccio *CRC* può essere usato per verificare l'insieme delle classi identificate

Linee guida per l'identificazione delle *entity classes*

1. Ogni classe deve avere un ben preciso *statement of purpose*
2. Ogni classe deve prevedere un insieme di istanze (oggetti) – le cosiddette *singleton classes* (per la quali si prevede una singola istanza) non sono di norma classificabili come *entity classes*
3. Ogni classe deve prevedere un insieme di attributi (non un singolo attributo)
4. Distinguere tra elementi che possono essere modellati come classi o come attributi
5. Ogni classe deve prevedere un insieme di operazioni (anche se inizialmente le operazioni vengono trascurate, i servizi che la classe mette a disposizione sono implicitamente derivabili dallo *statement of purpose*)

Casi di studio (*)

A. *University Enrolment*

B. *Video Store*

C. *Contact Management*

D. *Telemarketing*

(*) MACIASZEK, L.A. (2001): *Requirements Analysis and System Design. Developing Information Systems with UML*, Addison Wesley

A. University Enrolment
Problem statement

- The **university** offers
  - Undergraduate and postgraduate degrees
  - To full-time and part-time students
- The **university structure**
  - Divisions containing departments
  - Single division administers each degree
  - Degree may include courses from other divisions
- **University enrolment** system
  - Individually tailored programs of study
  - Prerequisite courses
  - Compulsory courses
  - Restrictions
    - Timetable clashes
    - Maximum class sizes, etc.

A. University Enrolment
Problem statement (2)

- The system is required to
  - Assist in pre-enrolment activities
  - Handle the enrolment procedures
- **Pre-enrolment activities**
  - Mail-outs of
    - Last semester's examination grades to students
    - Enrolment instructions
- **During enrolment**
  - Accept students' proposed programs of study
  - Validate for prerequisites, timetable clashes, class sizes, special approvals, etc.
- Resolutions to some of the problems may require consultation with academic advisers or academics in charge of course offerings

B. Video Store
Problem statement

- The **video store**
  - Rentals of video tapes and disks to customers
  - All video tapes and disks bar-coded
  - Customer membership also be bar-coded.
- Existing customers can place reservations on videos to be collected at specific date
- Answering customer enquiries, including enquiries about movies that the video store does not stock (but may order on request)

C. Contact Management
Problem statement

- The **market research company** with established customer base of organizations that buy market analysis reports
- The company is constantly on the search for new customers
- **Contact management** system
  - Prospective customers
  - Actual customers
  - Past customers
- The new contact management system to be developed internally and be available to all employees in the company, but with varying levels of access
  - Employees of Customer Services Department will take the ownership of the system
- The system to permit flexible scheduling and re-scheduling of contact-related activities so that the employees can successfully collaborate to win new customers and foster existing relationships

D. Telemarketing
Problem statement

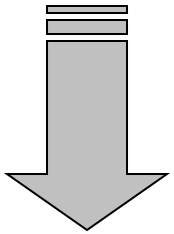
- The **charitable society** sells lottery tickets to raise funds
  - **Campaigns** to support currently important charitable causes
  - Past contributors (**supporters**) targeted through telemarketing and/or direct mail-outs
- Rewards (special bonus campaigns)
  - For bulk buying
  - For attracting new contributors
- The society does not randomly target potential supporters by using telephone directories or similar means

D. Telemarketing
Problem statement (2)

- **Telemarketing application**
  - To support up to fifty telemarketers working simultaneously
  - To schedule the phone calls according to pre-specified priorities and other known constraints
  - To dial up the scheduled phone calls
  - To re-schedule unsuccessful connections
  - To arrange other telephone callbacks to supporters
  - To records the conversation outcomes, including ticket orders and any changes to supporter records

Example A.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.



Relevant

```
Degree
Course
```

Fuzzy

```
CompulsoryCourse
ElectiveCourse
```

Example A.1 – University Enrolment

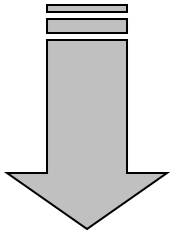
- More requirements:
  - Each course is at a given level and has a credit-point value
  - A course can be part of any number of degrees
  - Each degree specifies minimum total credit points value required for degree completion
  - Students may combine course offerings into programs of study suited to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

Example A.1– University Enrolment (solution)

Relevant classes	***Fuzzy classes***
Course	CompulsoryCourse
Degree	ElectiveCourse
Student	StudyProgram
CourseOffering	

Example B.1 – 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 B.1 – 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 B.1 – Video Store (solution)

Relevant classes	***Fuzzy classes***
MovieTitle	RentalConditions
VideoMedium	
VideoTape	
VideoDisk (or DVDDisk)	
BetaTape	
VHSTape	

Example C.1 – Contact Management

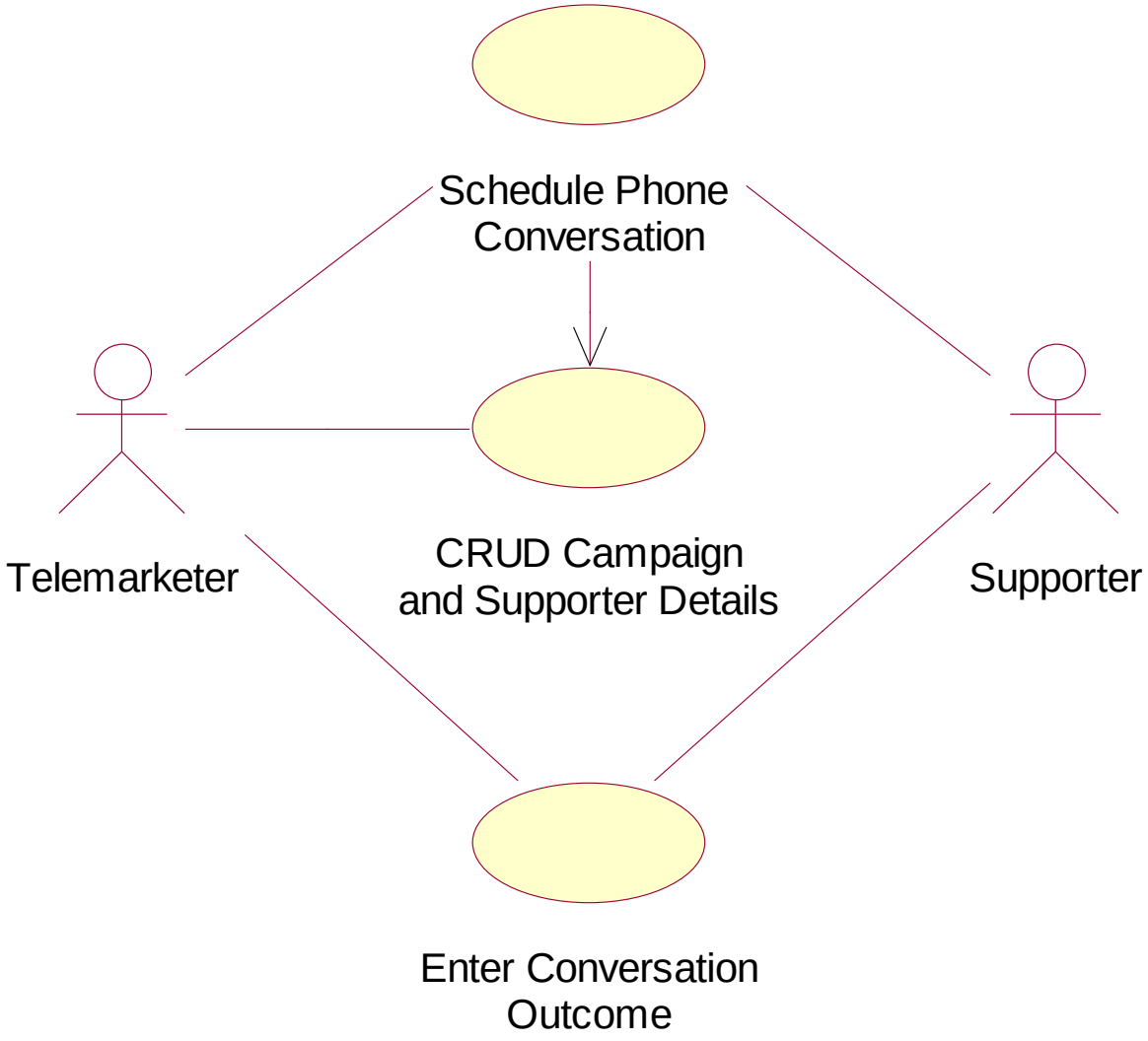
- Consider the following requirements for the Contact Management system and identify the candidate classes:
  - To "keep in touch" with current and prospective customer base
  - To store the names, phone numbers, postal and courier addresses, etc. of organizations and contact persons in these organizations
  - 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 C.1 – Contact Management (solution)

Relevant classes	***Fuzzy classes***
Organization	CurrentOrg
Contact	ProspectiveOrg
Employee	PostalAddress
Task	CourierAddress
Event	

Example D.1 – Telemarketing

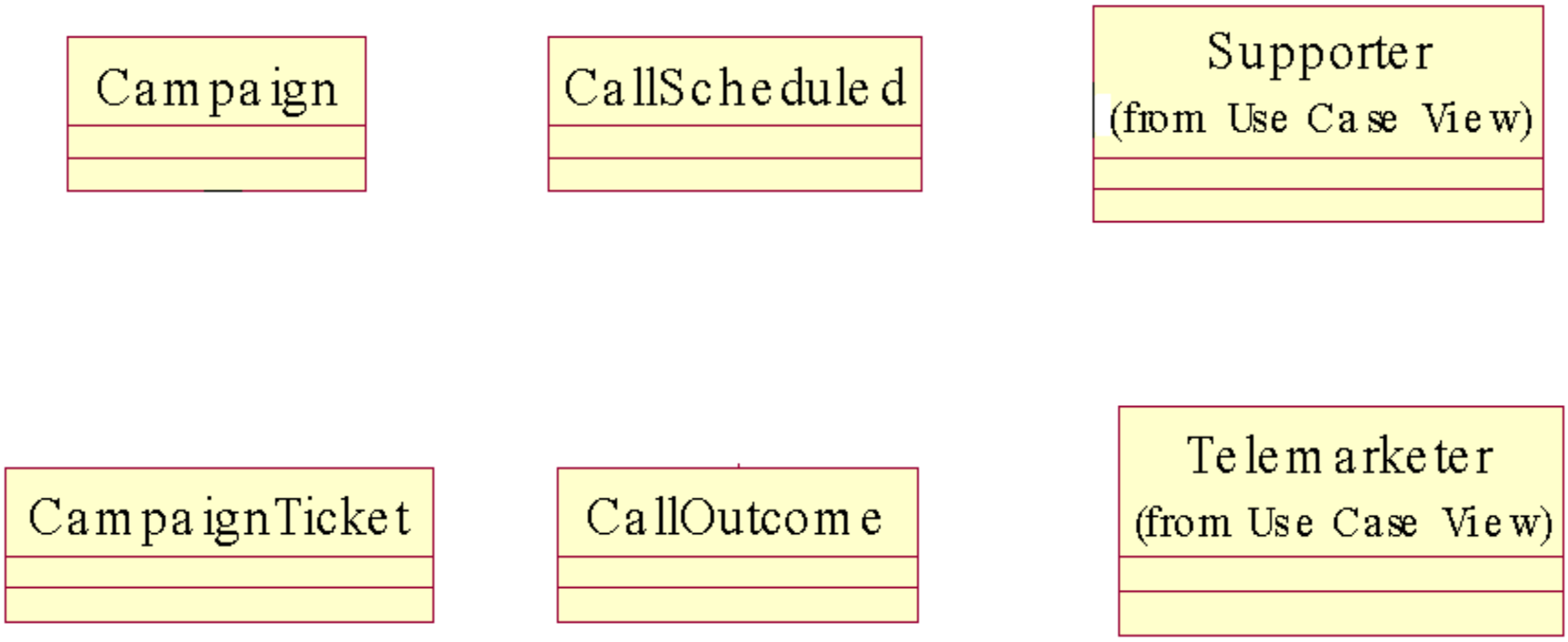
Business use case diagram



Example D.1 - Telemarketing

- Consider the following textual description for the Telemarketing system's use cases and identify the candidate classes:
  - The telemarketer requests the system that the phone call to a supporter be scheduled and dialed up
  - Upon successful connection, the telemarketer offers lottery tickets to the supporter. During a conversation, the telemarketer may need to access and modify both campaign and supporter details (*CRUD*, *create* – *read* – *update* – *delete*)
  - Finally, the telemarketer enters the conversation outcome, i.e. the successful or unsuccessful results of the telemarketing action

Example D.1 – Telemarketing (solution)



Linee guida per la specifica delle classi

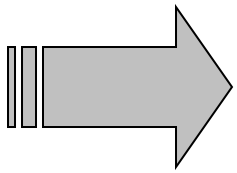
- Nomi di classe
  - Associare ad ogni classe un nome significativo nello specifico dominio applicativo
  - Adottare una convenzione standard per assegnare nomi alle classi, ad esempio:

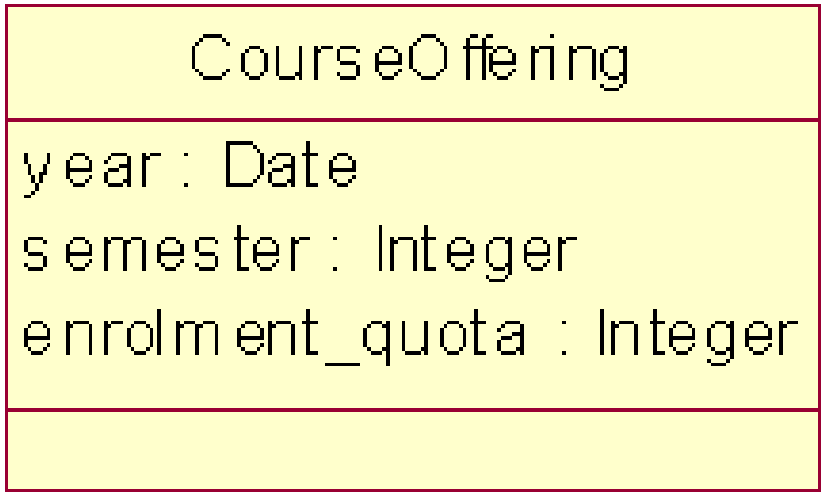
    nome singolare, parole multiple devono essere congiunte, con l'iniziale di ciascuna parola in carattere maiuscolo (es. `PostalAddress`)
  - Definire una lunghezza massima per i nomi delle classi (non più di 30 caratteri)
- Attributi e operazioni
  - Considerare inizialmente solo attributi che caratterizzano possibili stati di interesse per gli oggetti
  - Adottare una convenzione standard per assegnare nomi agli attributi, ad esempio:

    le parole devono essere scritte in carattere minuscolo, separate da un carattere di *underscore* (es. `street_name`)
  - Ritardare l'aggiunta di operazioni fino al momento in cui sia disponibile il modello comportamentale, da cui vanno derivate

Example A.2 – University Enrolment

- Refer to Example A.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.

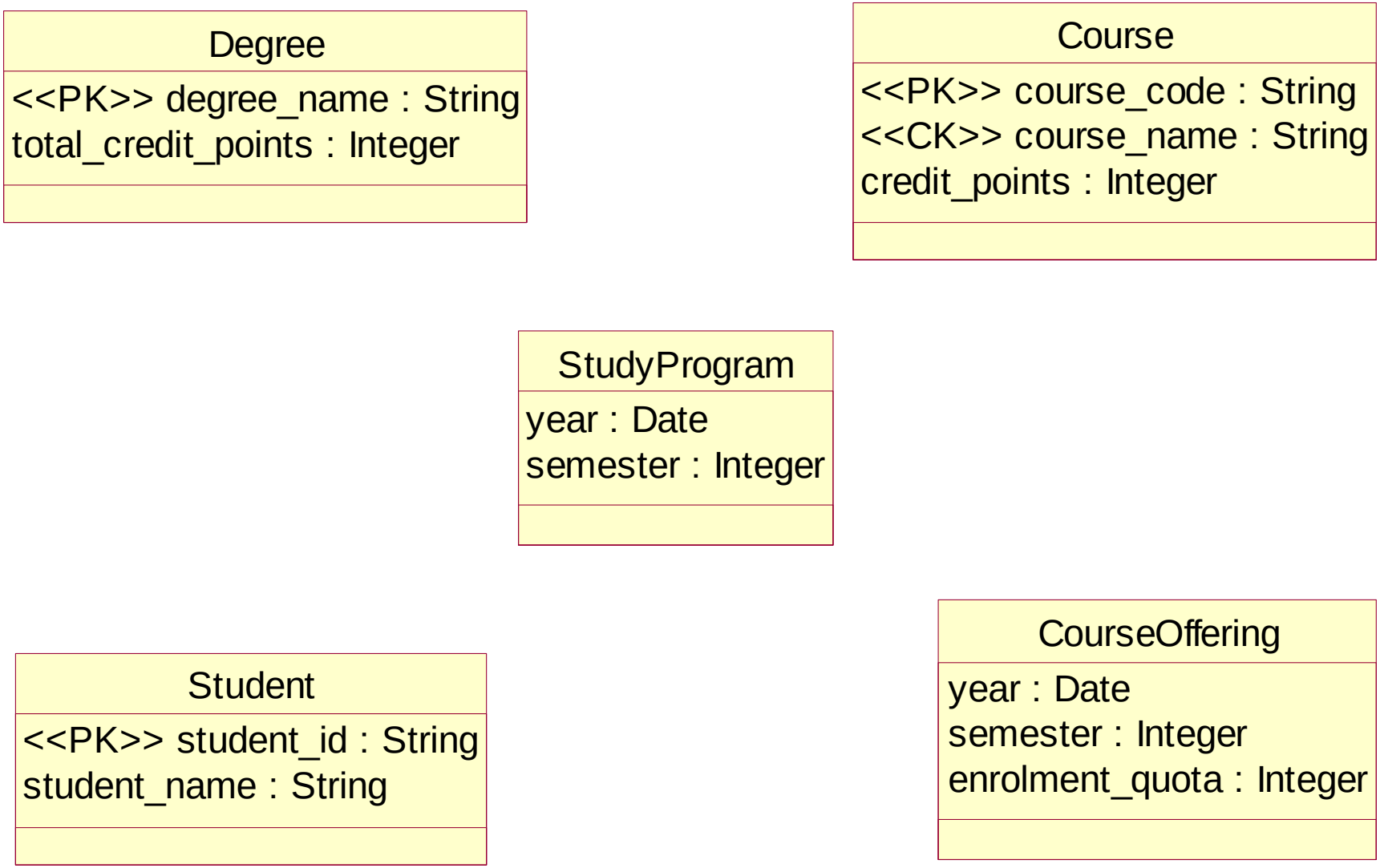




Example A.2 – University Enrolment

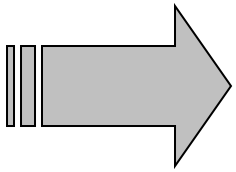
- More requirements:
  - A student's proposed program of study is entered in the on-line enrolment system
  - The system checks the program's consistency and reports any problems
  - The problems need to be resolved with the help of an academic adviser
  - The final program of study is subject to academic approval by the delegate of the Head of Division and it is then forwarded to the Registrar

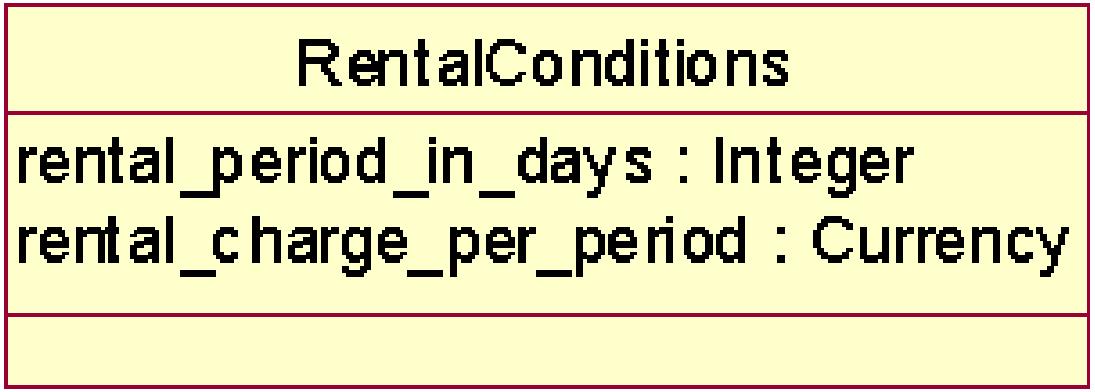
Example A.2 – University Enrolment (solution)



Example B.2 – Video Store

- Refer to Example B.1
- 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).

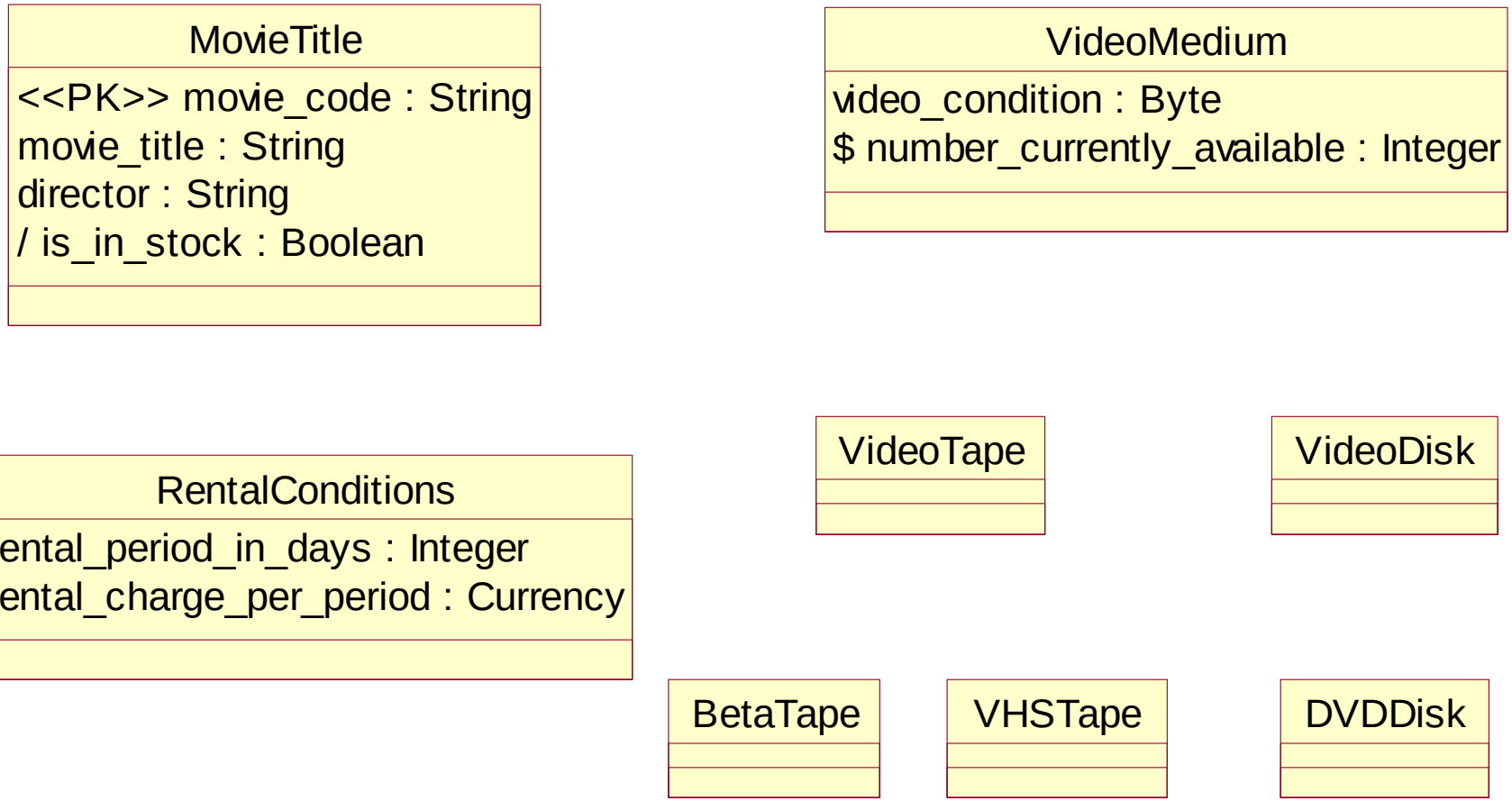




Example B.2 – Video Store

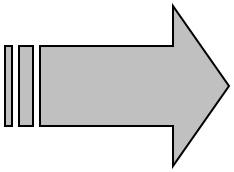
- More requirements:
  - The system should accommodate future video storage formats in addition to VHS tapes, Beta tapes and DVD disks
  - The employees frequently use a movie code, instead of movie title, to identify the movie
  - The same movie title may have more than one release by different directors

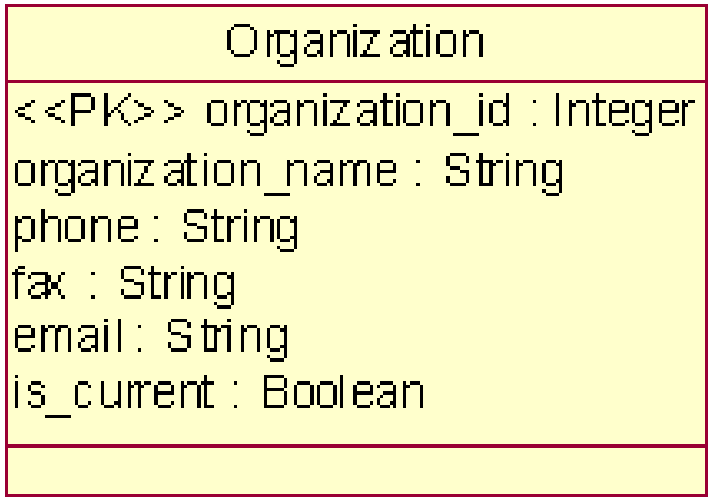
Example B.2 – Video Store (solution)

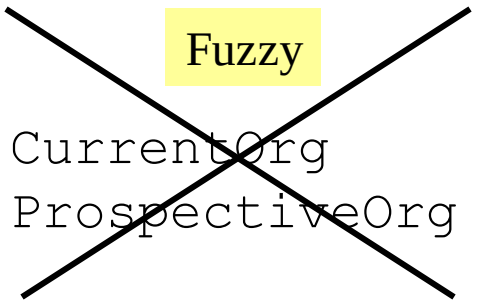


Example C.2 – Contact Management

- Refer to Example C.1 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.



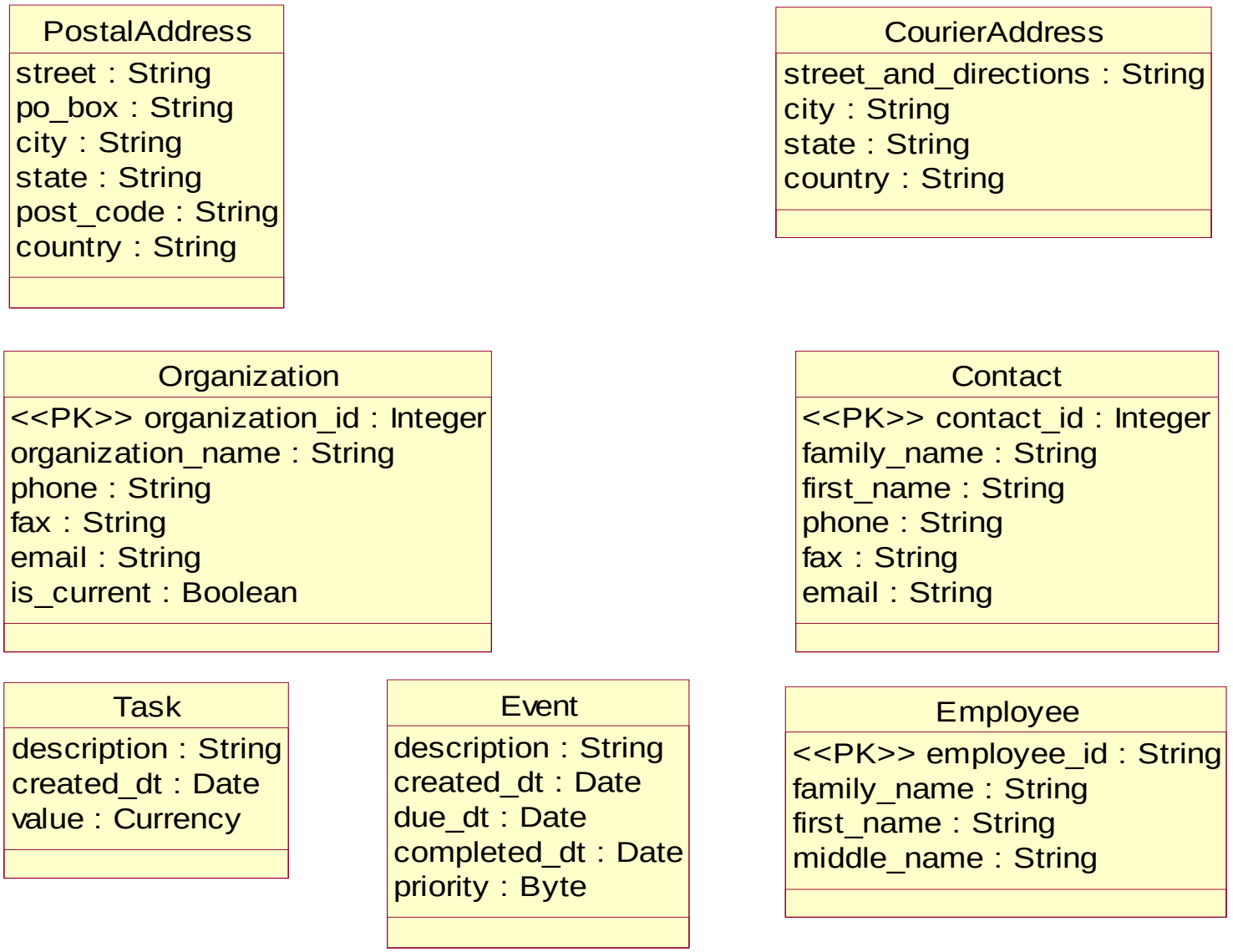




Example C.2 – Contact Management

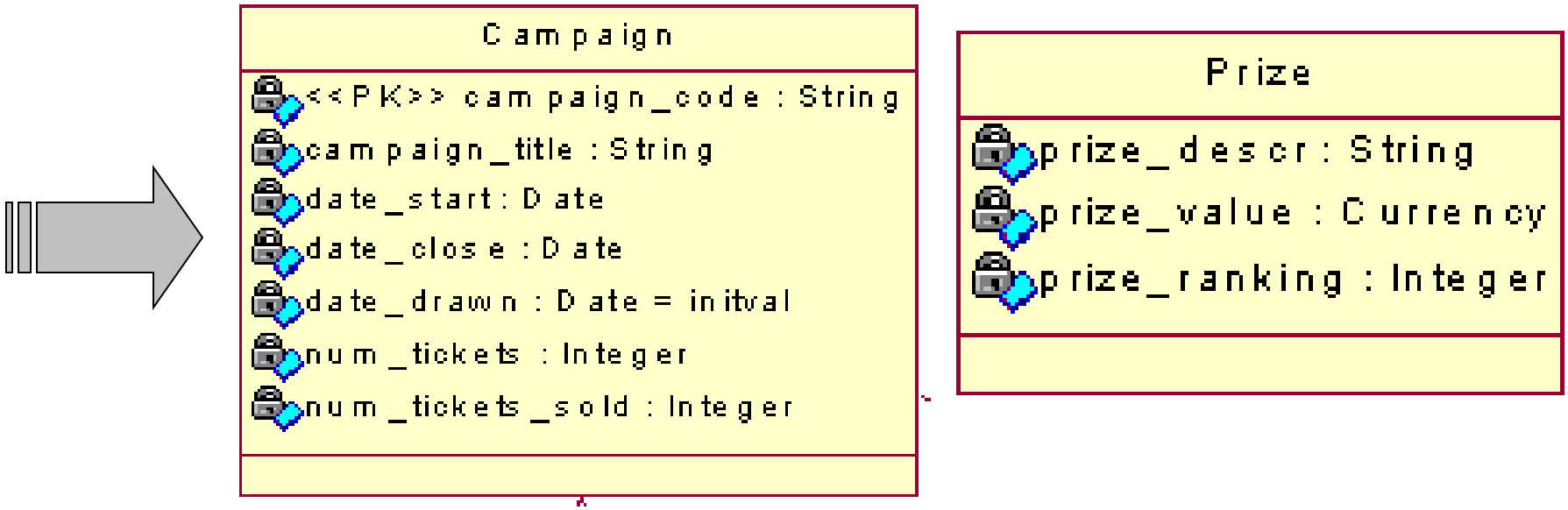
- More requirements:
  - Reports on contacts based on postal and courier addresses (e.g. find all customers by post code)
  - Date and time of the task creation are recorded
  - The "money value" of a task can be stored
  - Events for the employee are displayed on the employee's screen in the calendar-like 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 C.2 – Contact Management (solution)



Example D.2 - Telemarketing

- Refer to Example D.1
- 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



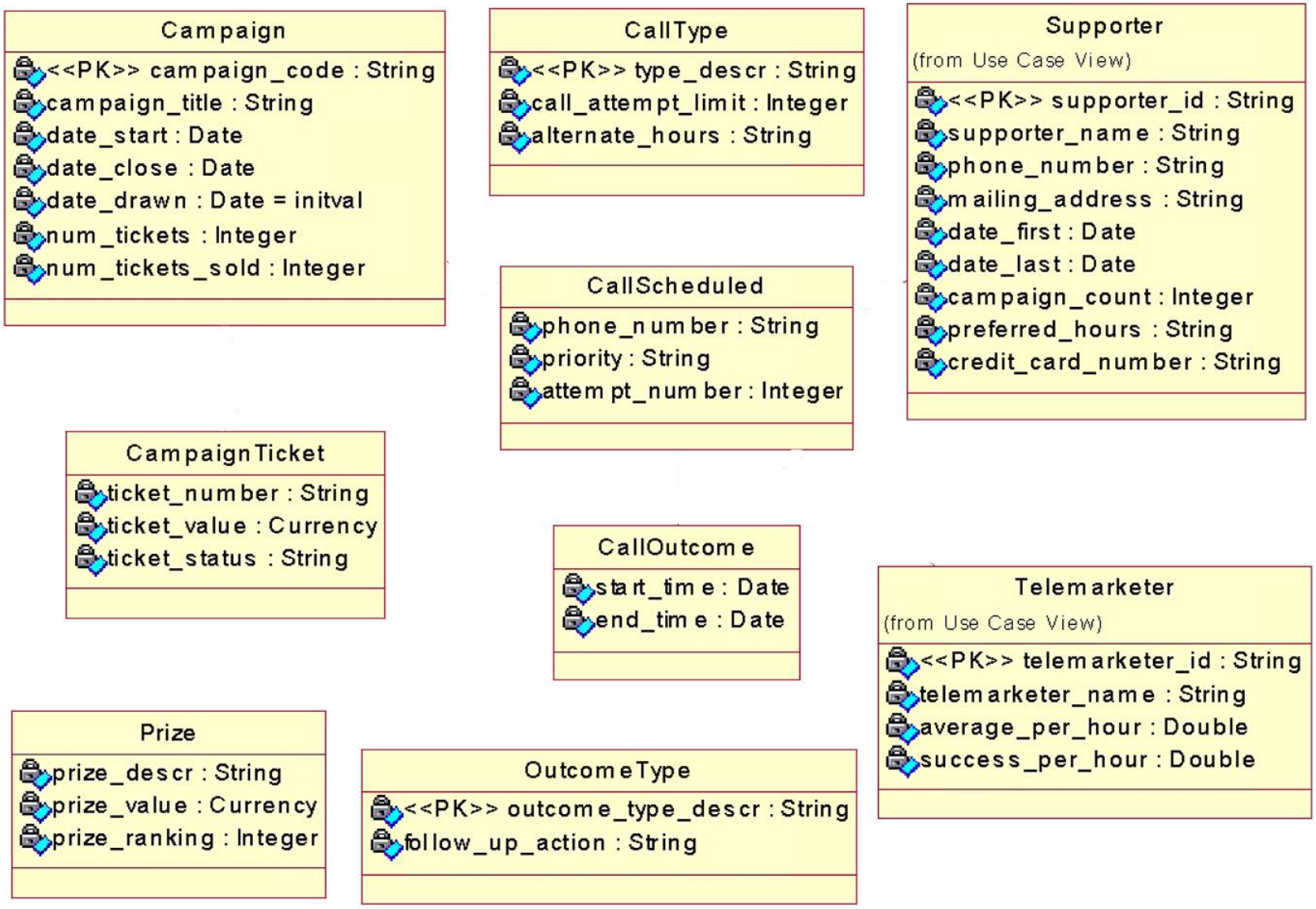
Example D.2 - Telemarketing

- 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 D.2 - 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 D.2 – Telemarketing (solution)



Identificazione delle associazioni

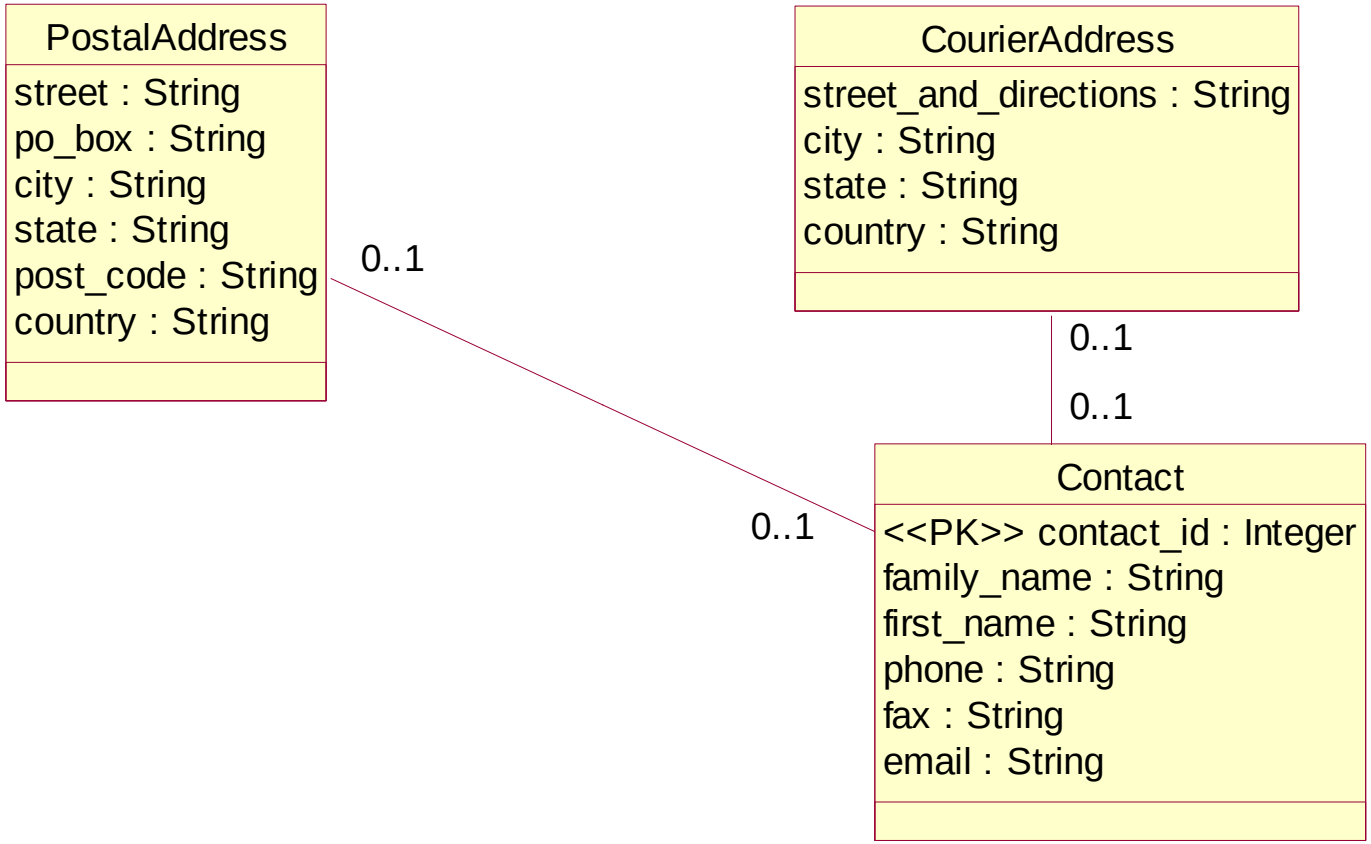
- Alcuni *attributi* identificati con le classi rappresentano associazioni (ogni attributo di tipo non primitivo dovrebbe essere modellato come un'associazione alla classe che rappresenta quel tipo di dato)
- Ogni *associazione ternaria* dovrebbe essere rimpiazzata con un *ciclo di associazioni binarie*, per evitare problemi di interpretazione
- Nei cicli di associazioni almeno un'associazione potrebbe essere eliminata e gestita come *associazione derivata*, anche se per problemi di efficienza spesso si introducono associazioni ridondanti

Specifica delle associazioni

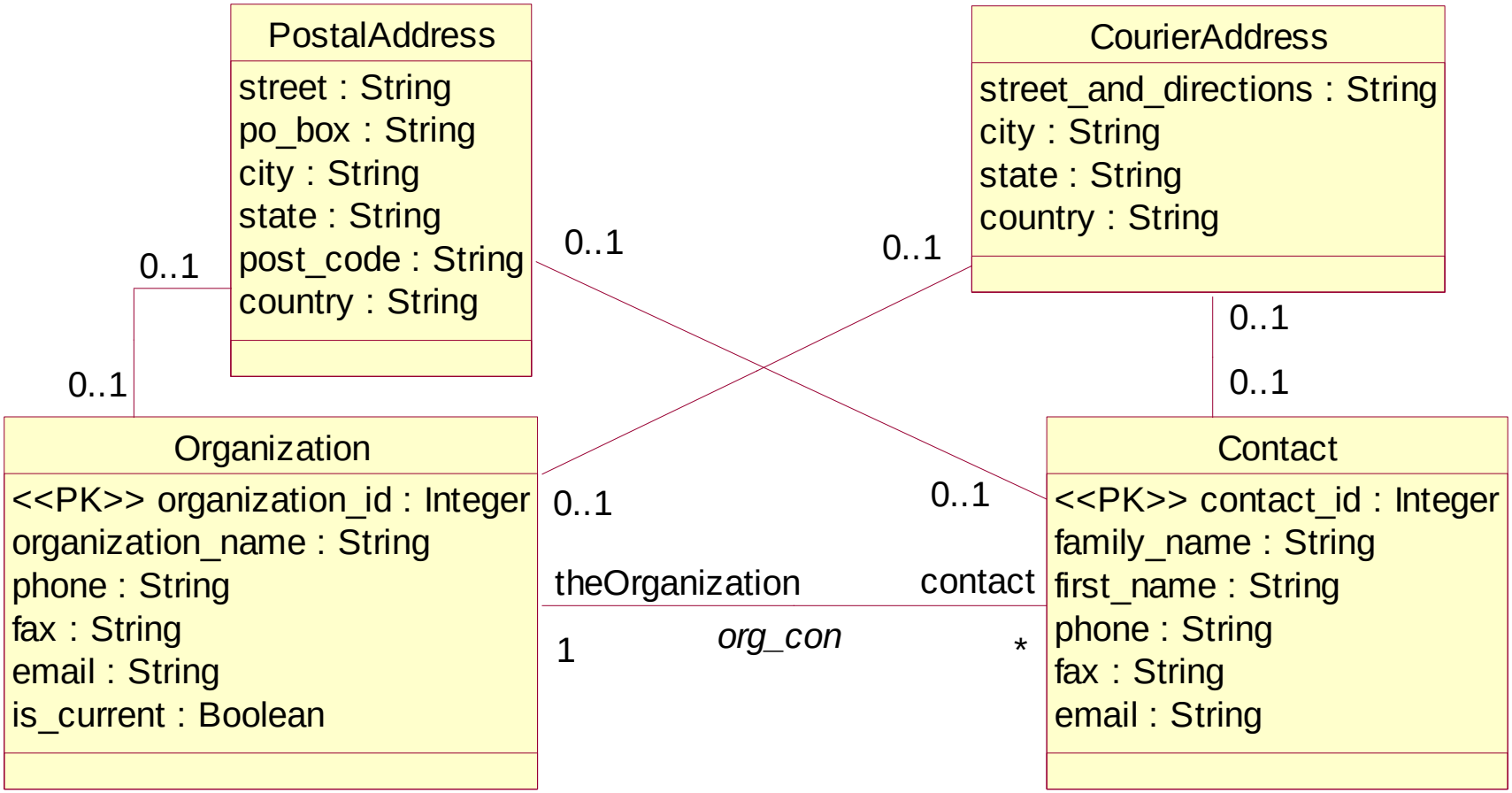
- Per *assegnare nomi alle associazioni* adottare la stessa convenzione usata per gli attributi (le parole devono essere scritte in carattere minuscolo, separate da un carattere di *underscore*)
- Assegnare nomi di ruolo (*rolename*) alle estremità dell'associazione (i *rolename* diventano i nomi degli attributi nella classe all'estremità opposta dell'associazione)
- Determinare la *molteplicità* delle associazioni (ad entrambe le estremità)

Example C.3 – Contact Management

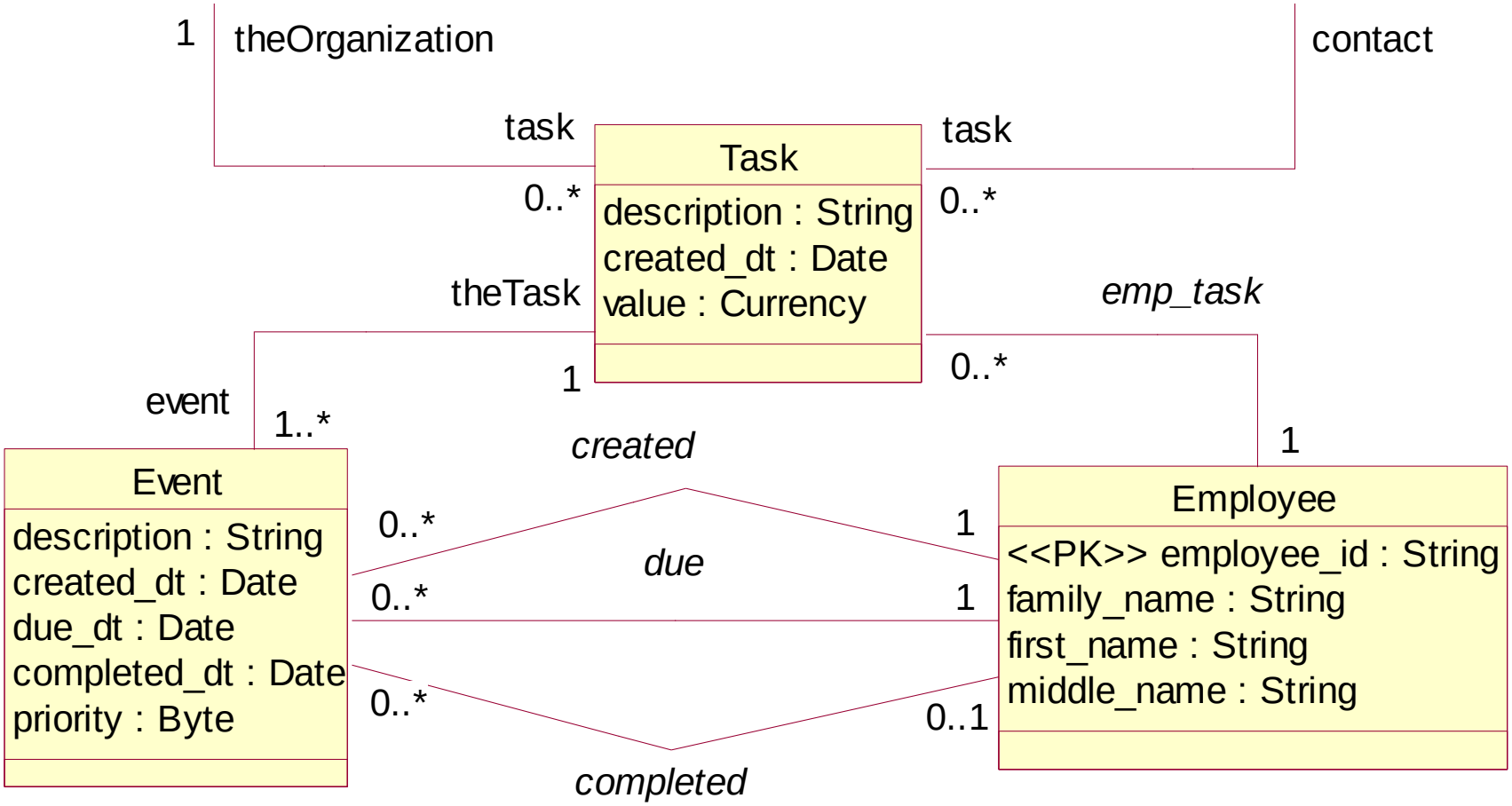
- Refer to Examples C.1 and C.2 - specify associations
- Consider, for example, the requirement:
  - The system allows producing various reports on our contacts based on postal and courier addresses



Example C.3 – Contact Management (solution – 1)



Example C.3 – Contact Management (solution – 2)



Aggregazione

- Rappresenta una relazione di tipo "*whole-part*" (contenimento) tra una classe composta (*superset class*) e l'insieme di una o più classi componenti (*subset classes*)
- Può assumere quattro differenti *significati*:
  - **ExclusiveOwns** (e.g. `Book` *`has`* `Chapter`, or `Chapter` *`is part of a`* `Book`)
    - *Existence-dependency*
    - *Transitivity*
    - *Asymmetricity*
    - *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

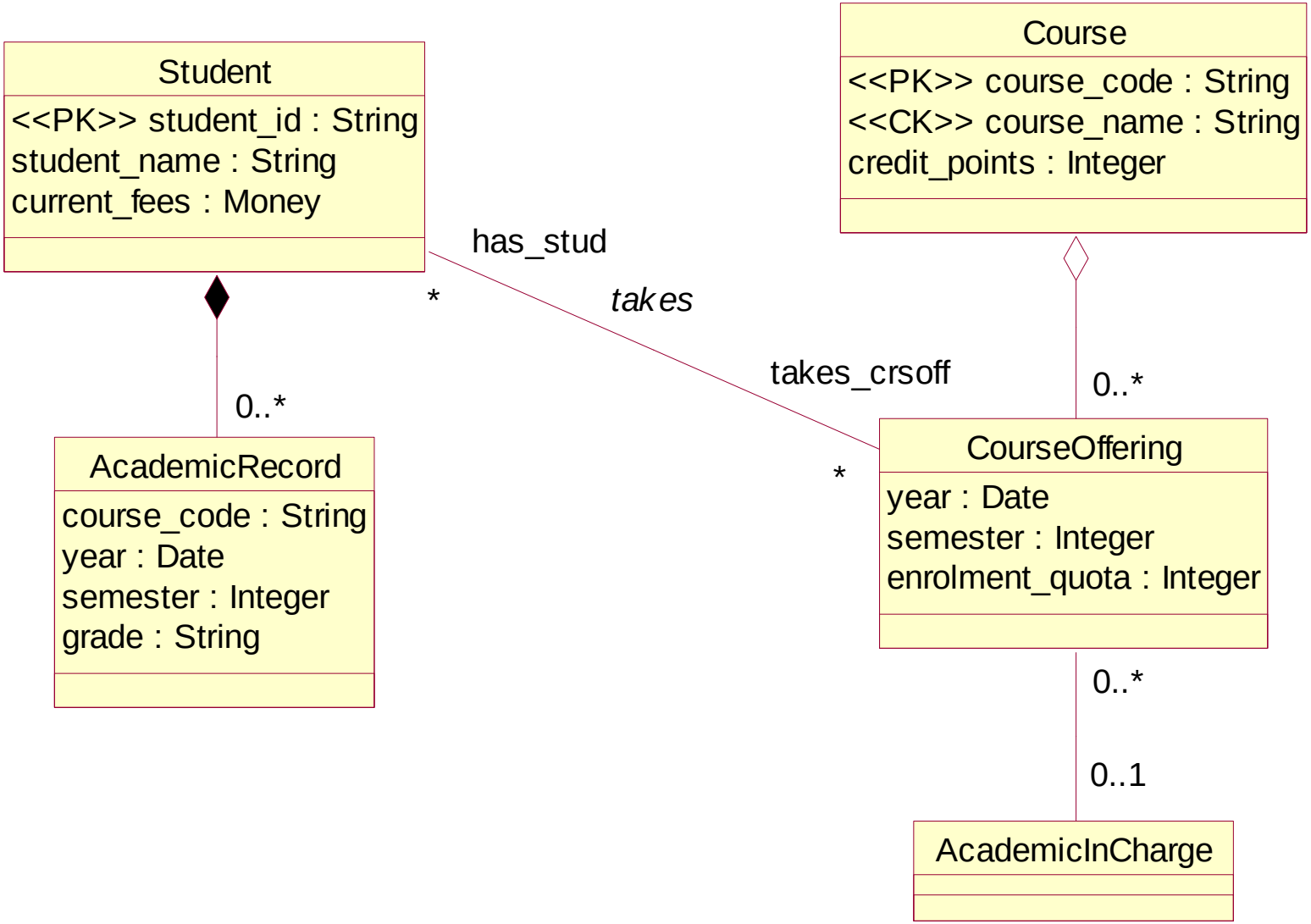
Specifica di aggregazione in UML

- **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 A.3 – University Enrolment

- Refer to Examples A.1 and A.2
- 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 A.3 – University Enrolment (solution)



Ereditarietà (generalizzazione)

- Usata per rappresentare la condivisione di attributi ed operazioni tra classi
- Le caratteristiche comuni sono modellate in una classe più *generica* (**superclasse**), che viene *specializzata* nell'insieme di **sottoclassi**
- Una sottoclasse *eredita* attributi ed operazioni della superclasse
- Caratteristiche:
  - Sostituibilità: un oggetto della sottoclasse è un valore legale per una variabile avente come tipo la superclasse (es. una variabile di tipo `Frutta` può avere un oggetto di tipo `Mela` come suo valore)
  - Polimorfismo: la stessa operazione può avere differenti implementazioni nelle sottoclassi

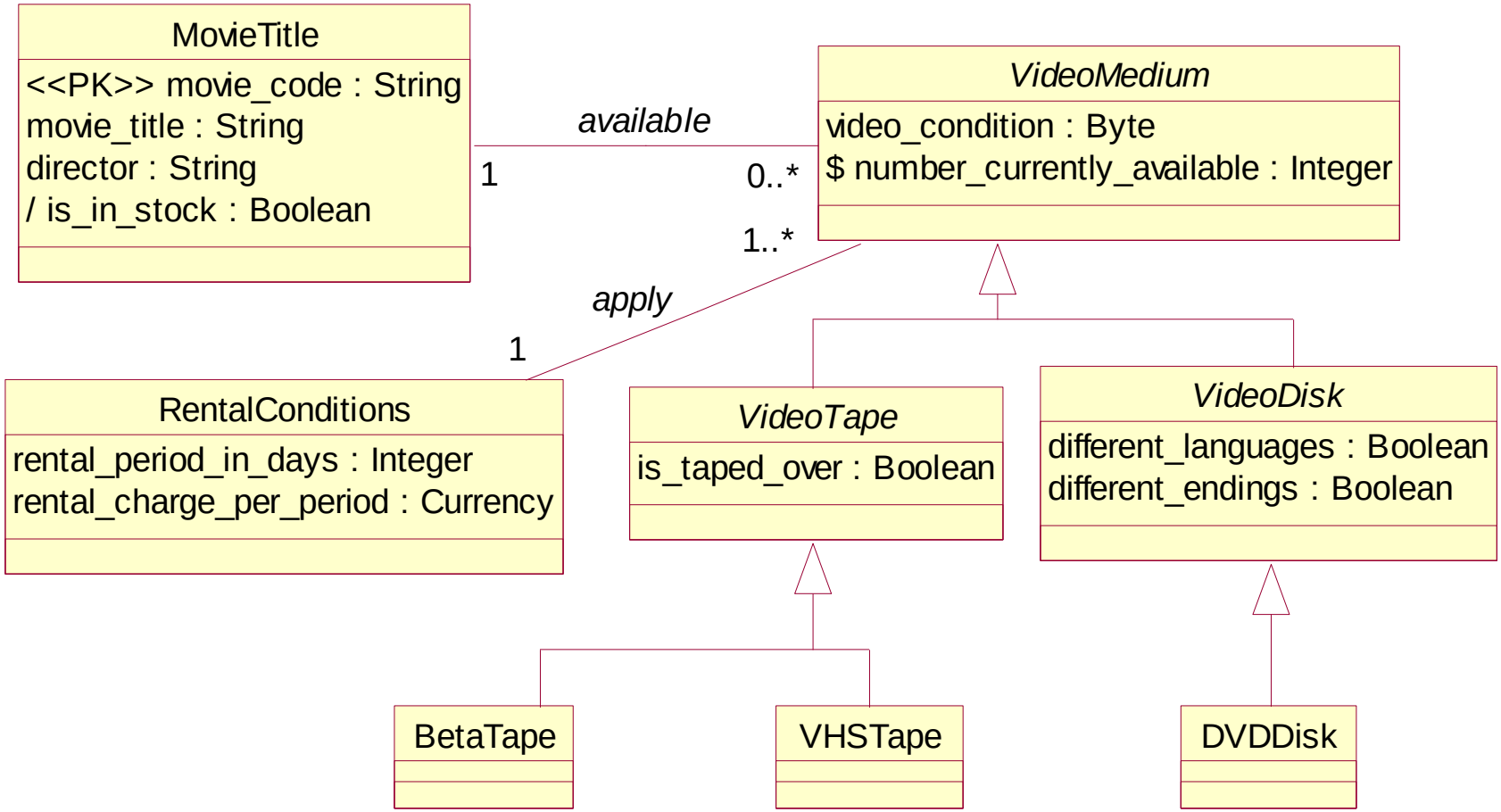
Specifica di ereditarietà in UML

- Rappresenta relazioni di tipo:
  - "can-be"
    - Es. `Student` can be a `TeachingAssistant`
  - "is-a-kind-of"
    - Es. `TeachingAssistant` is a kind of `Student`
- Supporto ad ereditarietà multipla
  - Es. `TeachingAssistant` is also a kind of `Teacher`
- Viene rappresentata in UML con una linea, che collega la sottoclasse con la superclasse, avente una freccia diretta verso la superclasse

Example B.3 – Video Store

- Refer to Examples B.1 and B.2
- The classes identified in Example B.2 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 B.3 – Video Store (solution)

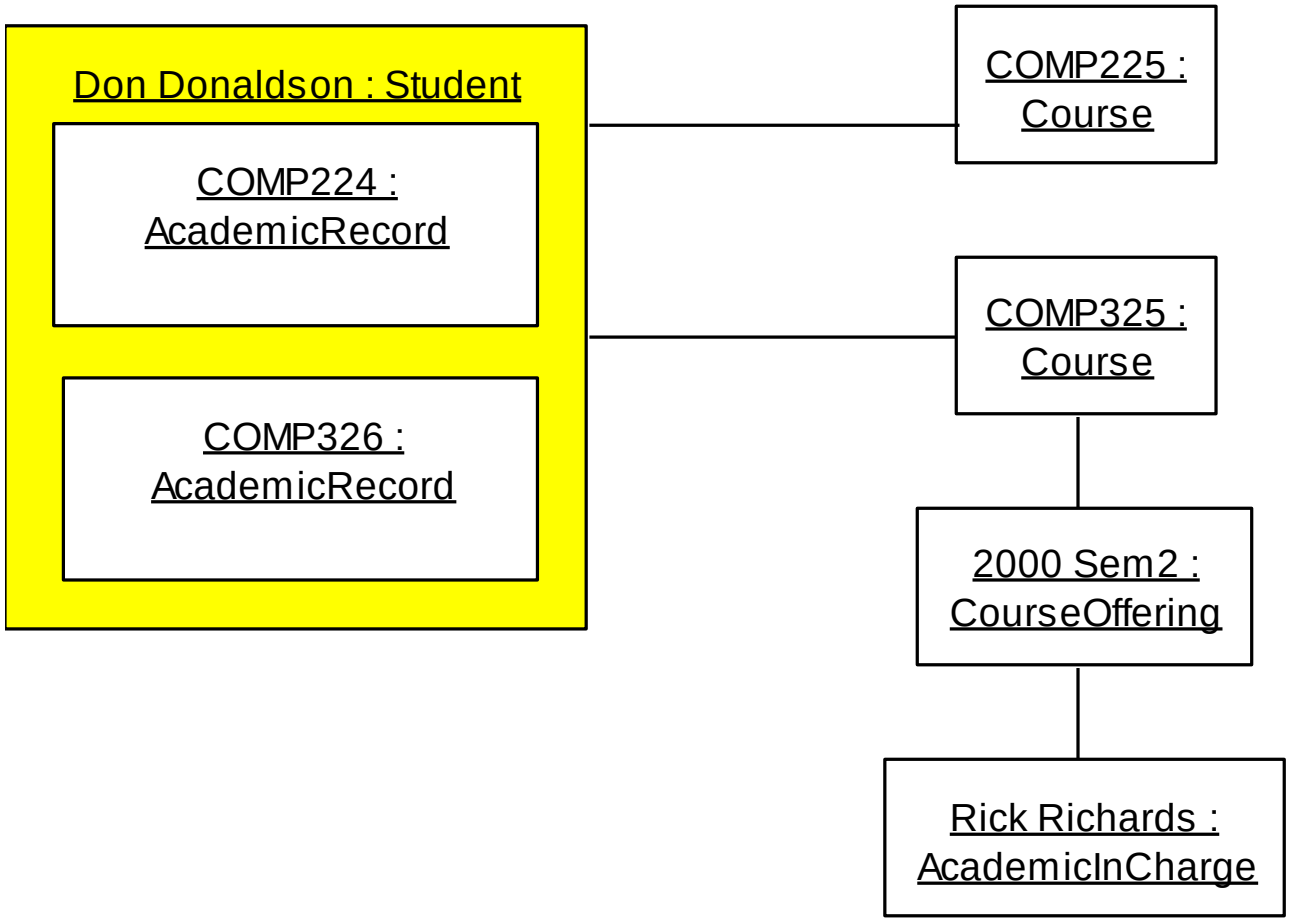


Object Diagram

- Rappresentazione grafica di istanze di classi
- Usati per
  - modellare relazioni complesse tra classi (a scopo esemplificativo)
  - illustrare le modifiche ai singoli oggetti durante l'evoluzione del sistema
  - Illustrare la collaborazione tra oggetti durante l'evoluzione del sistema

Example A.4 – University Enrolment

- Show an object diagram with few objects representing the classes in Example A.3



Modello comportamentale

- Rappresenta gli aspetti funzionali del sistema da un punto di vista operativo, evidenziando come gli oggetti collaborano ed interagiscono al fine di offrire i servizi che il sistema mette a disposizione
- Fa uso di vari formalismi:
  - Use case diagram (per descrivere scenari di funzionamento)
  - Activity diagram (per descrivere il flusso di elaborazione)
  - Sequence diagram (per descrivere l'interazione tra gli oggetti)
  - Collaboration diagram (per descrivere l'interazione tra gli oggetti)
- Viene costruito in modo iterativo ed incrementale, usando le informazioni del modello dei dati, che a sua volta fa uso del modello comportamentale per identificare *operazioni* e classi aggiuntive (*control classes* e *boundary classes*)

Use Case Diagram

- Può essere sviluppato a differenti livelli di astrazione (sia in fase di OOA che OOD)
- Durante la fase di OOA, si concentra su ***COSA*** il sistema deve fare (*scenari di funzionamento*)
- Un caso d'uso rappresenta:
  - una funzionalità completa (flusso principale, sottoflussi e alternative)
  - una funzionalità visibile dall'esterno
  - un comportamento ortogonale (ogni use case viene eseguito in modo indipendente dagli altri)
  - una funzionalità originata da un attore del sistema (una volta originato, il caso d'uso può interagire con altri attori)
  - una funzionalità che produce un risultato significativo per un attore

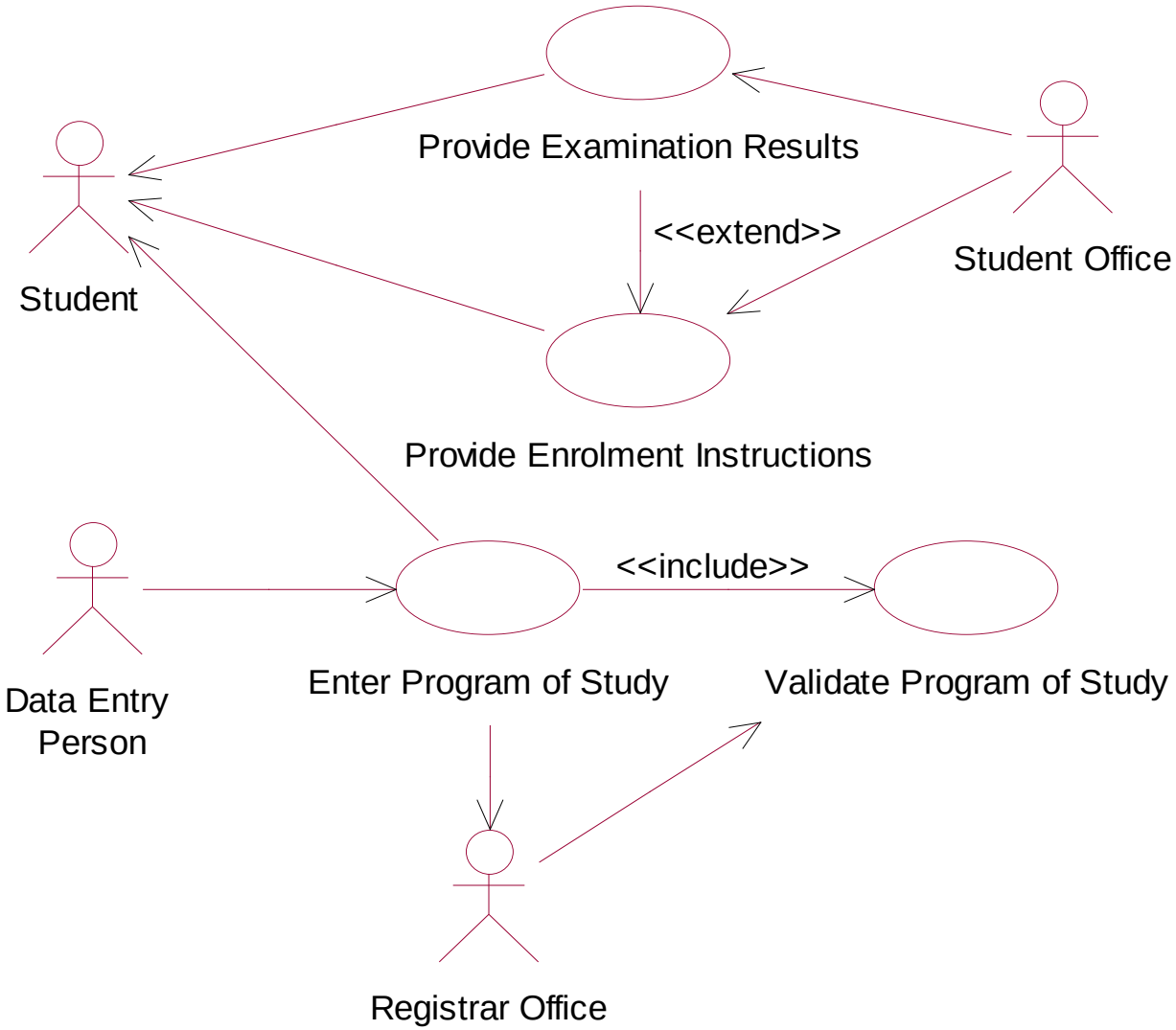
Identificazione dei casi d'uso

- A partire da:
  - Insieme dei requisiti utente
  - Attori del sistema (insieme ai relativi obiettivi)
- L'identificazione può essere facilitata facendosi guidare dalle seguenti domande:
  - Quali sono i compiti principali svolti da ciascun attore?
  - Un attore accede o modifica l'informazione nel sistema?
  - L'attore rappresenta il tramite mediante cui il sistema viene informato di modifiche apportate in altri sistemi?
  - L'attore deve essere informato di eventuali cambiamenti avvenuti nel sistema?
- Durante la fase di OOA, i casi d'uso identificano le necessità degli attori del sistema

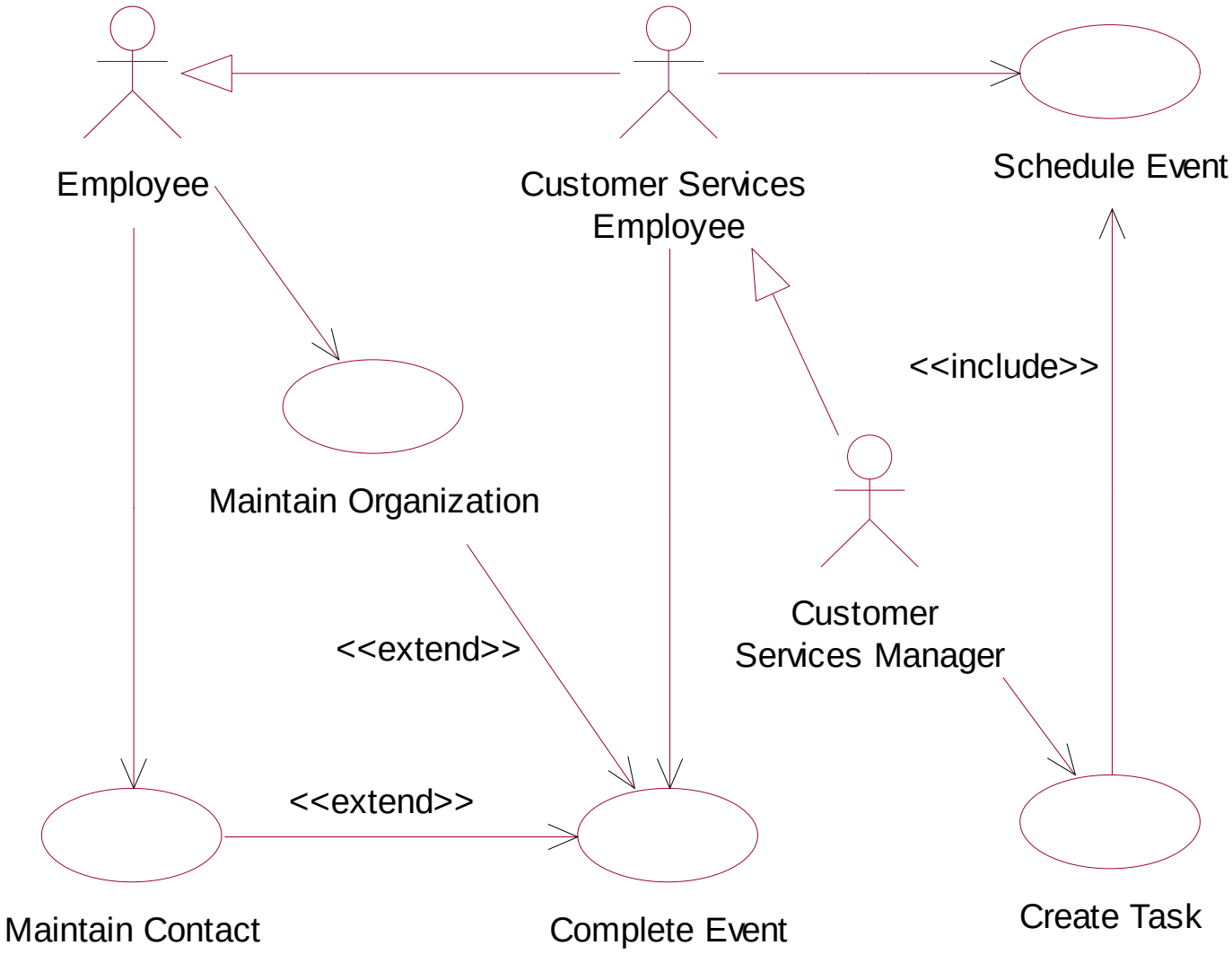
Specifica di *use case diagram*

- Rappresentazione grafica di attori, casi d'uso e relazioni
- Si possono rappresentare quattro tipi di relazioni:
  - Associazione (tra attore e caso d'uso)
  - Include (identificato dallo *stereotype*: «include»)
    - Un caso d'uso *included* è sempre necessario per completare il caso d'uso con il quale è messo in relazione
  - Extend (identificato dallo *stereotype* : «extend»)
    - Un caso d'uso *extended* può attivare il caso d'uso dal quale viene esteso (ma tale attivazione non è necessaria per completare il caso d'uso *extended*)
  - Generalizzazione

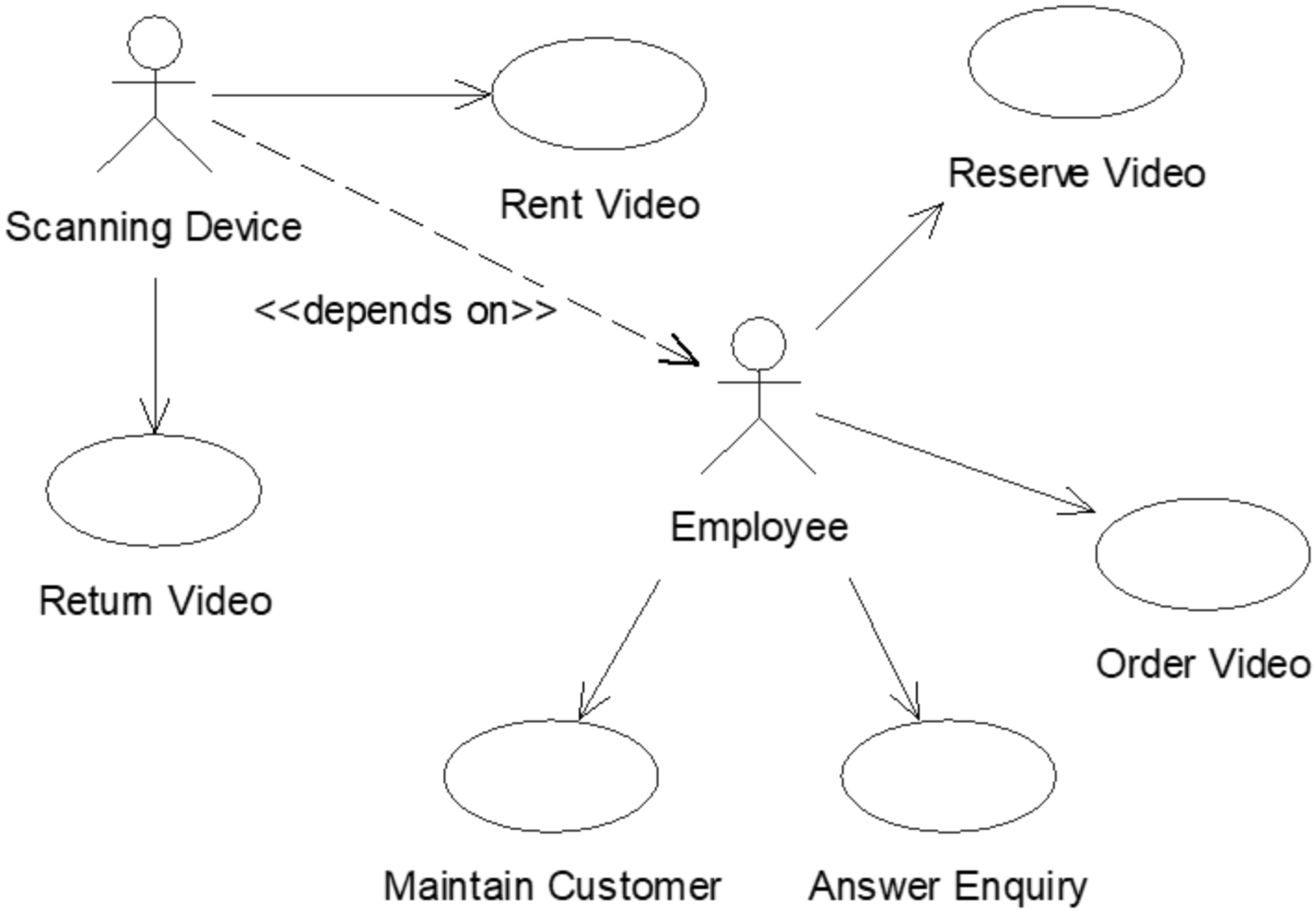
Example A.5 – University Enrolment



Example C.3 – Contact Management



Example B.4 – Video Store



Example B.4 – Video Store (Rent Video - 1)

Brief Description	*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.* *Provided the customer has a non-delinquent account, the tape is rented out once the payment has been received.* *If the tape is not returned in a timely fashion, an overdue notice is mailed to the customer.*
Actors	*Employee, Scanning Device*
Preconditions	*Video tape or disk is available to be hired. Customer has a membership card. Scanner devices work correctly.* *Employee at the front desk knows how to use the system.*

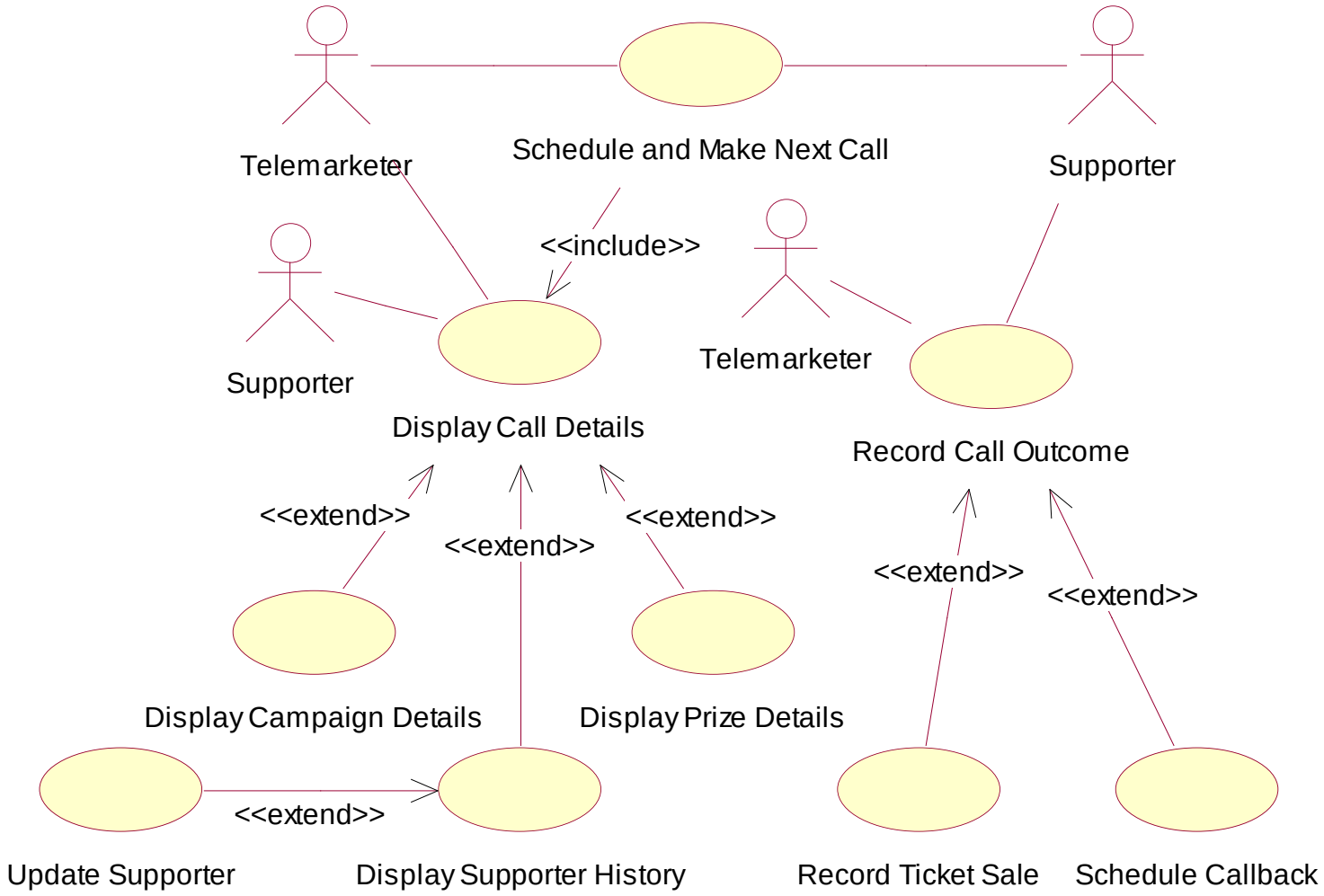
Example B.4 – Video Store (Rent Video - 2)

Main Flow	*A customer may inquire an employee about video availability (including a reserved video) or may pick a tape or disk from the shelves. The video and membership card are scanned and any delinquent or overdue details are brought up for the employee attention. 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 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 a 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 B.4 – Video Store (Rent Video - 3)

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

Example D.3 – Telemarketing



Activity Diagram

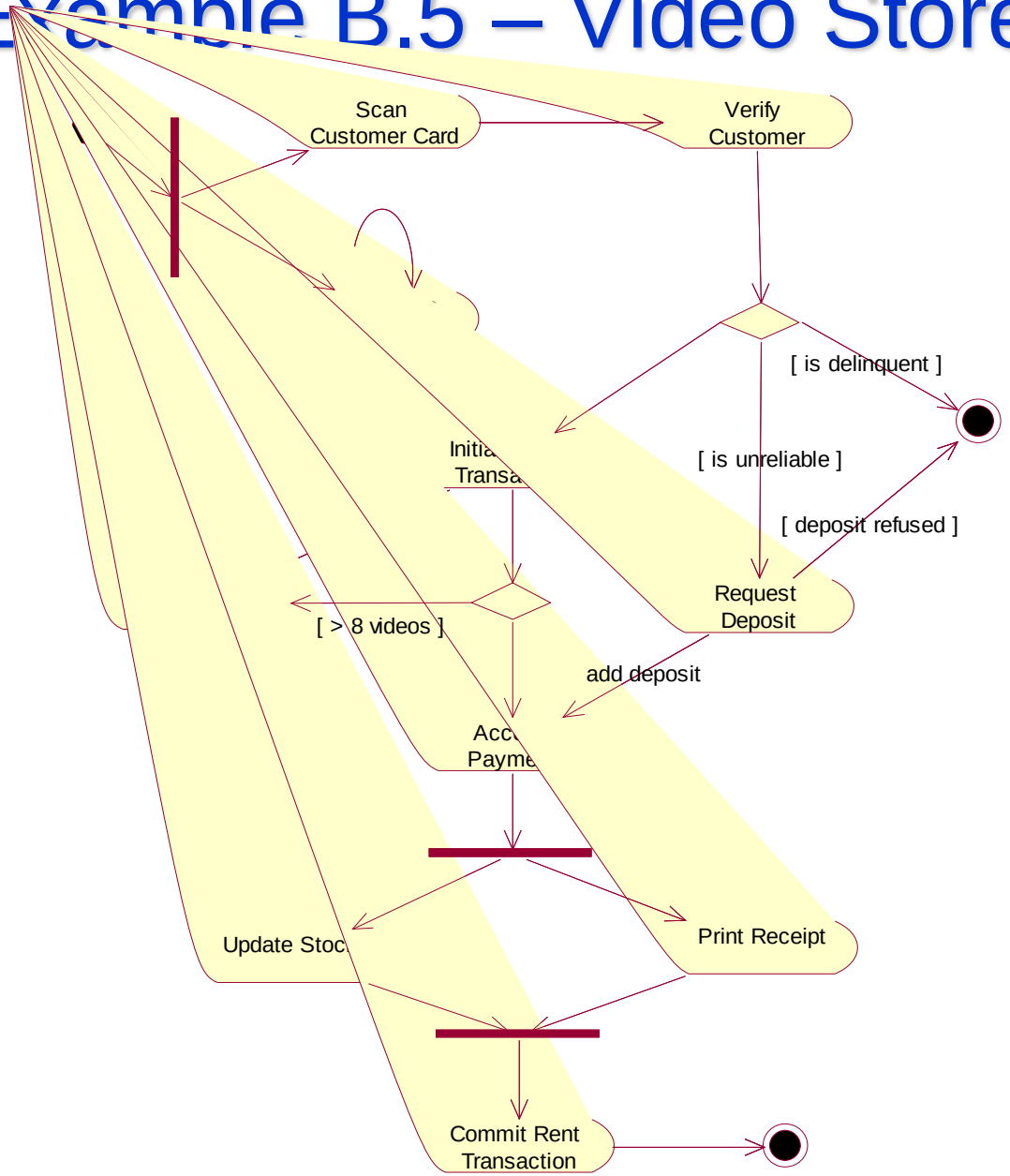
- Rappresenta a vari livelli di astrazione il flusso di esecuzione, sia *sequenziale* che *concorrente*, in una applicazione object-oriented
- E' una variante degli *state diagram*, in cui gli *stati* rappresentano l'esecuzione di *azioni* e le *transizioni* sono attivate dal completamento di tali azioni
- Usato principalmente in fase di OOD per rappresentare il flusso di esecuzione delle operazioni definite nel *class diagram*
- In fase di OOA, viene usato per rappresentare il flusso delle attività nella esecuzione di un caso d'uso (un caso d'uso può essere associato ad uno o più *activity diagram*)
- Poichè non vengono mostrati gli oggetti che eseguono le attività, può essere costruito anche in assenza del *class diagram*
- In presenza del class diagram, ogni attività può essere associata ad una o più operazioni appartenenti ad una o più classi

Specifica di activity diagram

- Un evento (*esterno*) che origina un caso d'uso viene modellato come un evento che causa l'esecuzione di un *activity diagram*
- Gli *action state* vengono identificati a partire dalla descrizione testuale dello scenario di funzionamento di un caso d'uso
- Gli *action state* vengono quindi associati mediante *transition lines* (che possono essere controllate da *guard conditions*)
- Le transizioni in uscita da un *action state* vengono percorse quando l' *action state* viene completato (l'esecuzione procede da un *action state* al successivo)
- Un *action state* viene completato quando la sua elaborazione termina
- Flussi concorrenti vengono modellati con barre di sincronizzazione (barre *fork-join*)
- Flussi alternativi vengono modellati con nodi decisionali (*branch/merge diamonds*)
- Eventi esterni non sono generalmente modellati

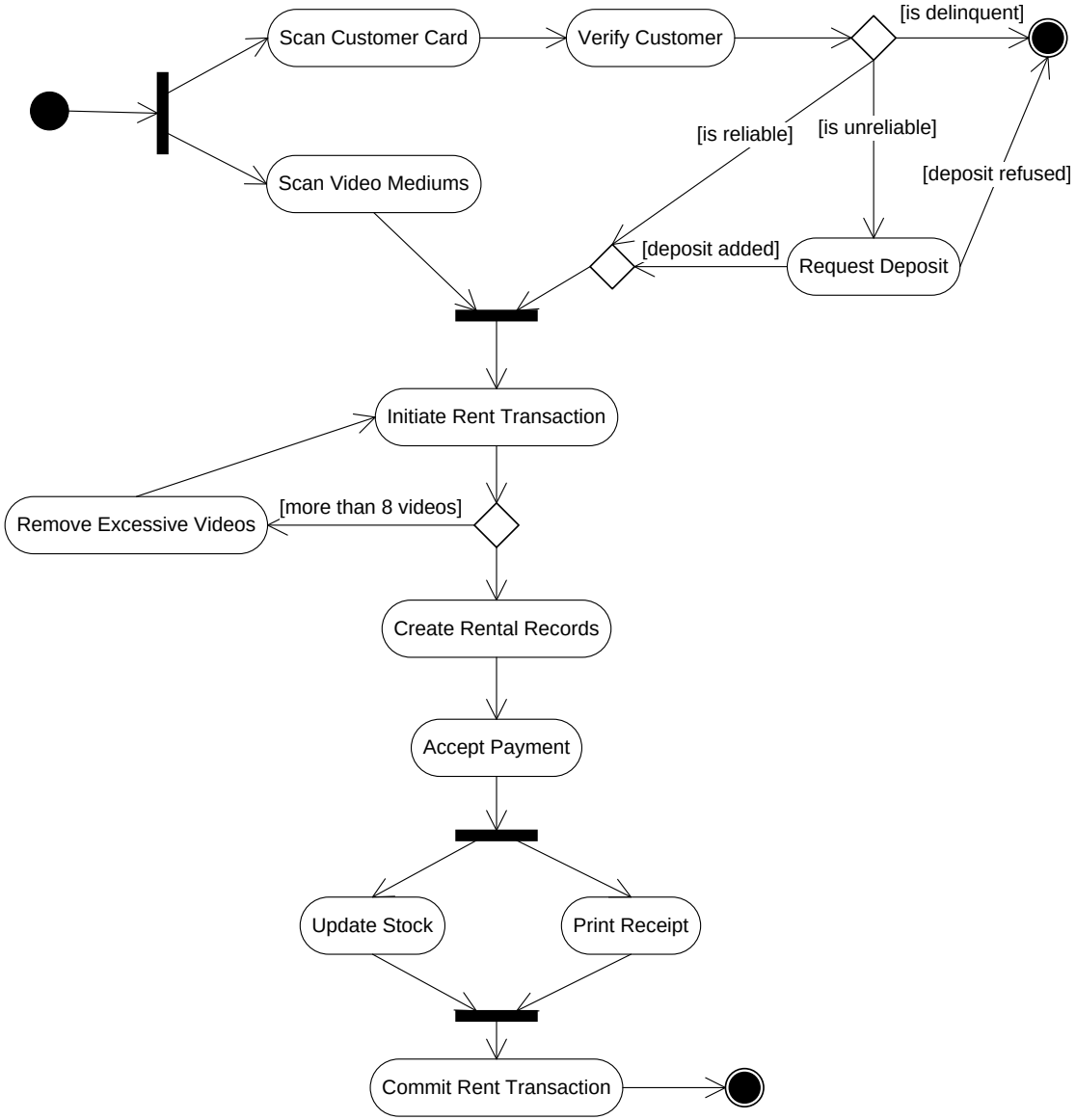
Example B.5 – Video Store

Rent Video use case



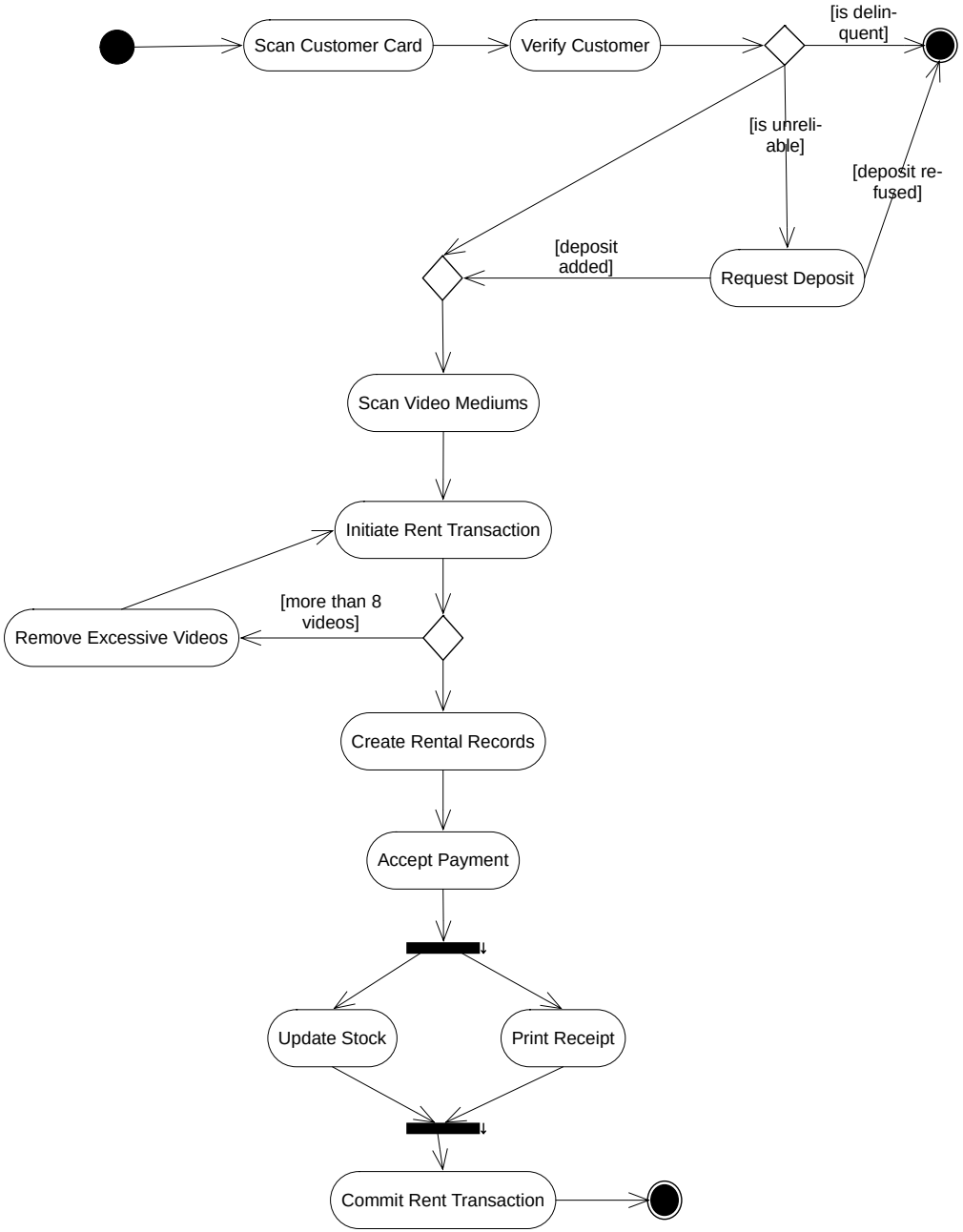
Example B.5 – Video Store (fixed)

Rent Video use case



Example B.5 – Video Store (improved)

Rent Video use case



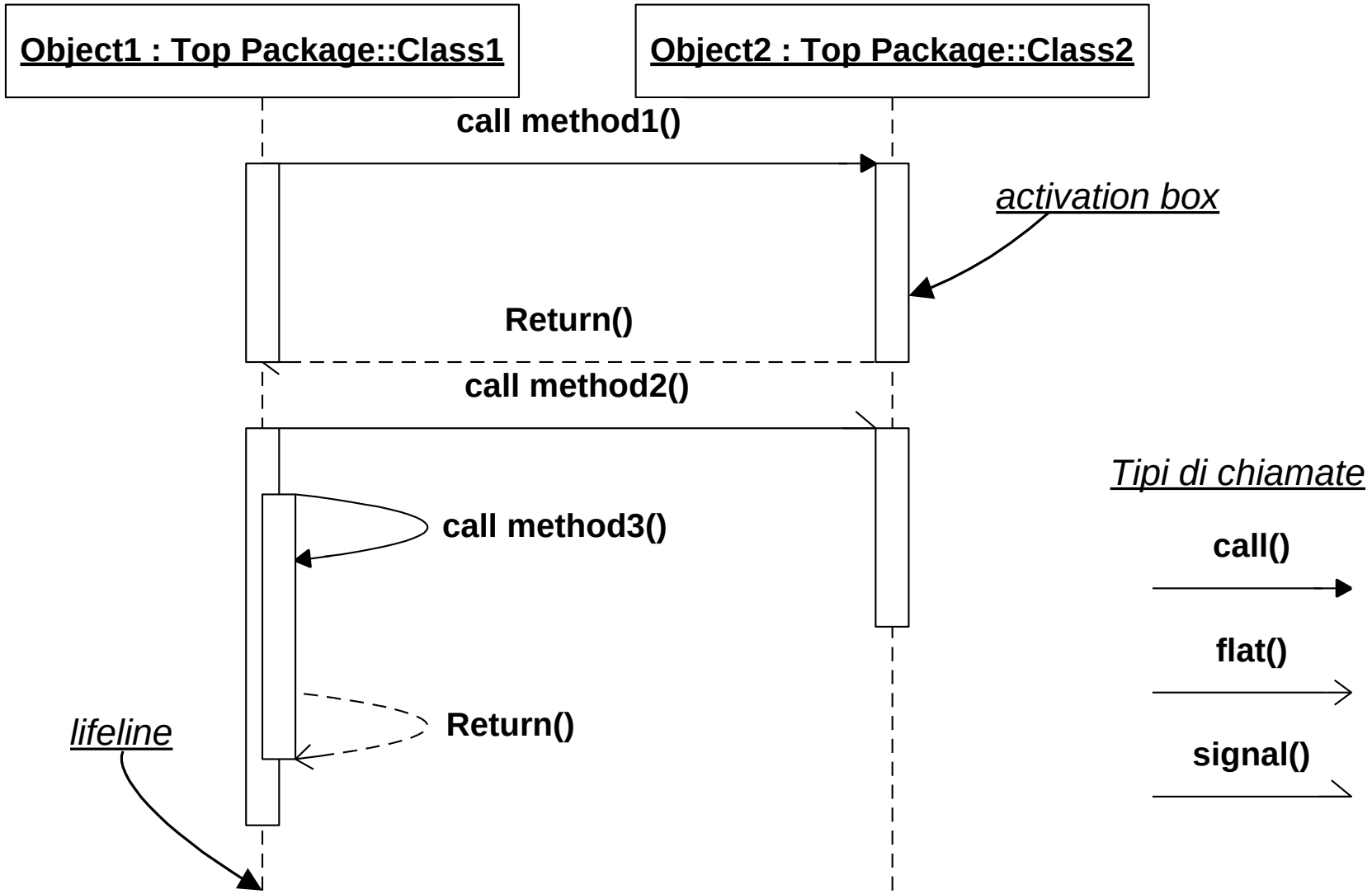
Diagrammi di interazione

- **Sequence diagram**
  - Descrive lo scambio di messaggi tra oggetti in ordine temporale
  - Usato principalmente in fase di OOA
- **Collaboration diagram**
  - Descrive lo scambio di messaggi tra oggetti mediante relazioni tra gli oggetti stessi
  - Usato principalmente in fase di OOD
- *Sequence diagram* e *collaboration diagram* permettono di identificare le operazioni delle classi nel *class diagram*
- *Sequence diagram* e *collaboration diagram* sono rappresentazioni equivalenti e possono essere generati in modo automatico l'uno dall'altro

Specifica di sequence diagram

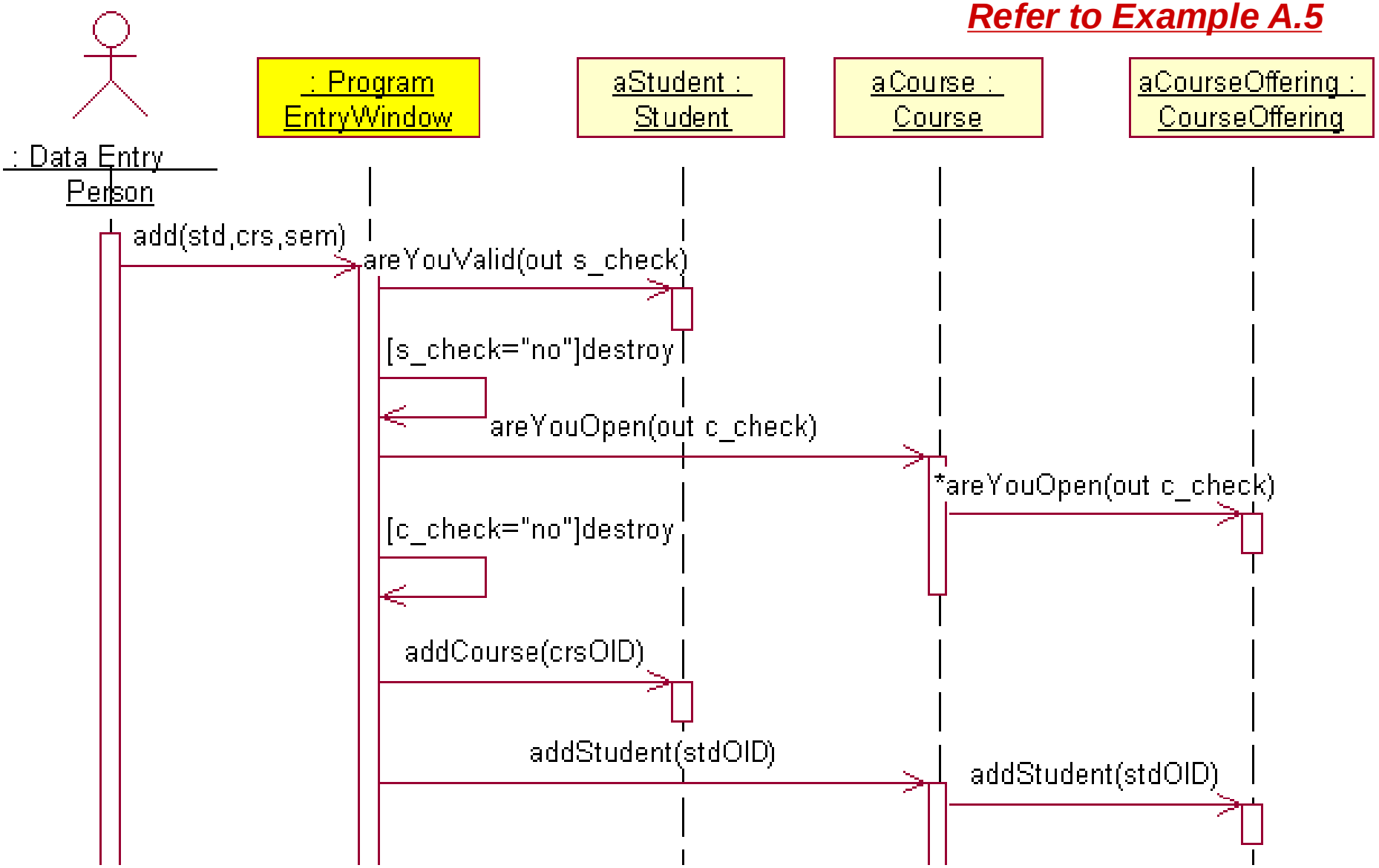
- Le attività dell'*activity diagram* vengono mappate come messaggi (di tipo "richiesta esecuzione attività") in un *sequence diagram*
- Un messaggio può rappresentare:
  - Un ***signal***
    - Denota una **chiamata di tipo asincrono**
    - L'oggetto mittente continua l'esecuzione dopo aver inviato il messaggio asincrono
  - Una ***call***
    - Denota una **chiamata di tipo sincrono**
    - L'oggetto mittente blocca l'esecuzione dopo aver inviato il messaggio sincrono, in attesa della risposta da parte dell'oggetto destinatario (che può o meno contenere valori di ritorno)

Notazione per *sequence diagram*



Example A.6 – University Enrolment

Refer to Example A.5



Interfaccia pubblica di classe

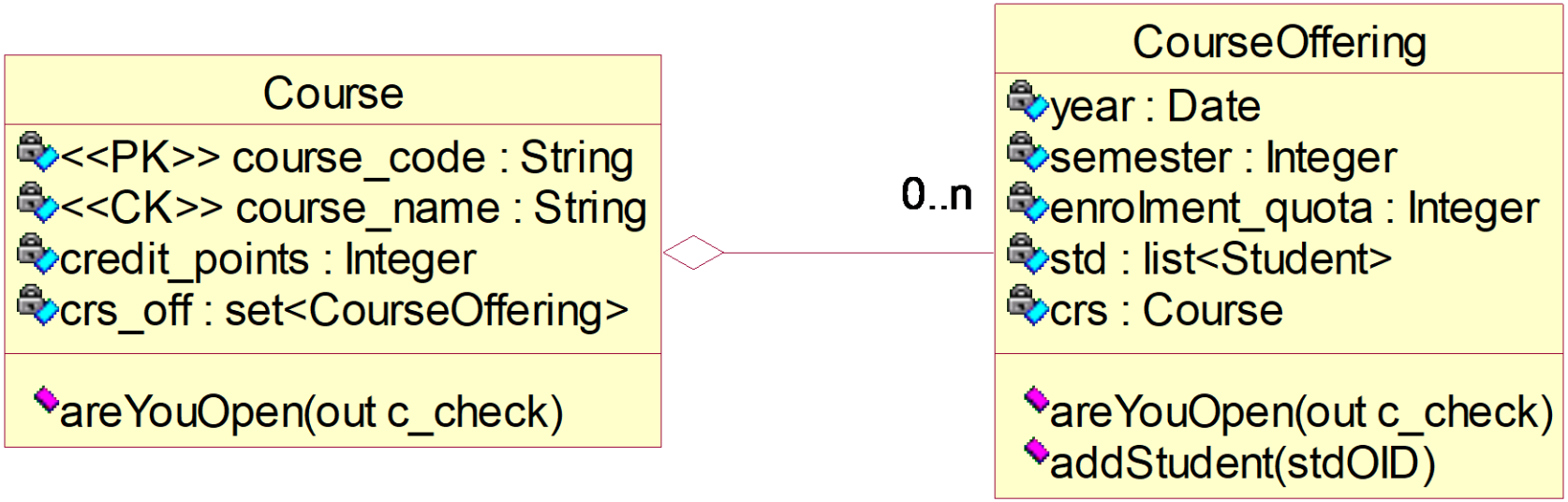
- Definisce l'insieme di operazioni che la classe mette a disposizione delle altre classi
- Durante la fase di OOA, si determina la *signature* dell'operazione, che consiste di:
  - Nome dell'operazione
  - Lista degli argomenti formali
  - Tipo di ritorno
- Durante la fase di OOD, si definisce l'algoritmo che implementa l'operazione
- Una operazione può avere:
  - ***Instance scope***
  - ***Class (static) scope***
    - rappresentata con un carattere `$` che precede il nome dell'operazione
    - agisce su *class object* (classi con attributi *static*)

Identificazione delle operazioni

- Dai sequence diagram
  - Ogni messaggio inviato ad un oggetto identifica un metodo della classe a cui appartiene tale oggetto
- Usando criteri aggiuntivi, come ad esempio:
  - il criterio CRUD, secondo cui ogni oggetto deve supportare le seguenti operazioni primitive (CRUD operations):
    - Create (una nuova istanza)
    - Read (lo stato di un oggetto)
    - Update (lo stato di un oggetto)
    - Delete (l'oggetto stesso)

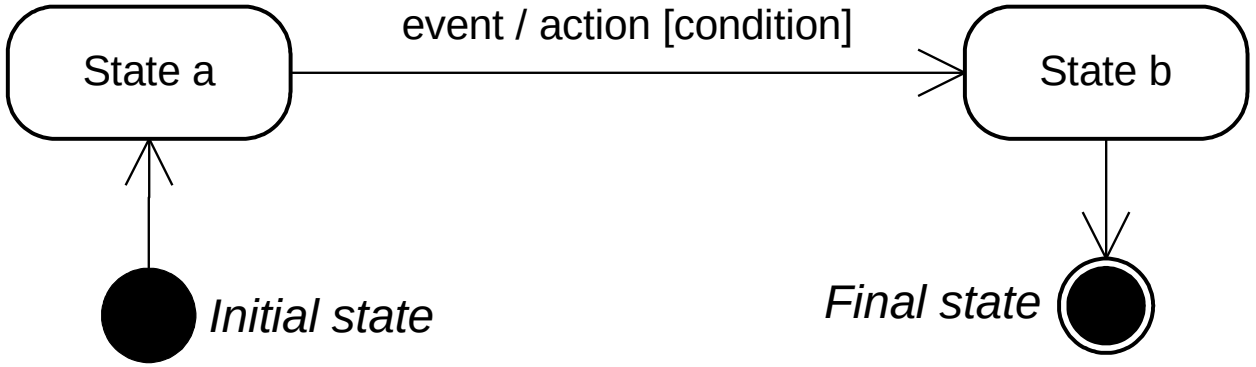
Example A.7 – University Enrolment

- Refer to Examples A.3 and A.6 and to the classes `Course` and `CourseOffering`
- Derive operations from the Sequence Diagram and add them to the classes `Course` and `CourseOffering`



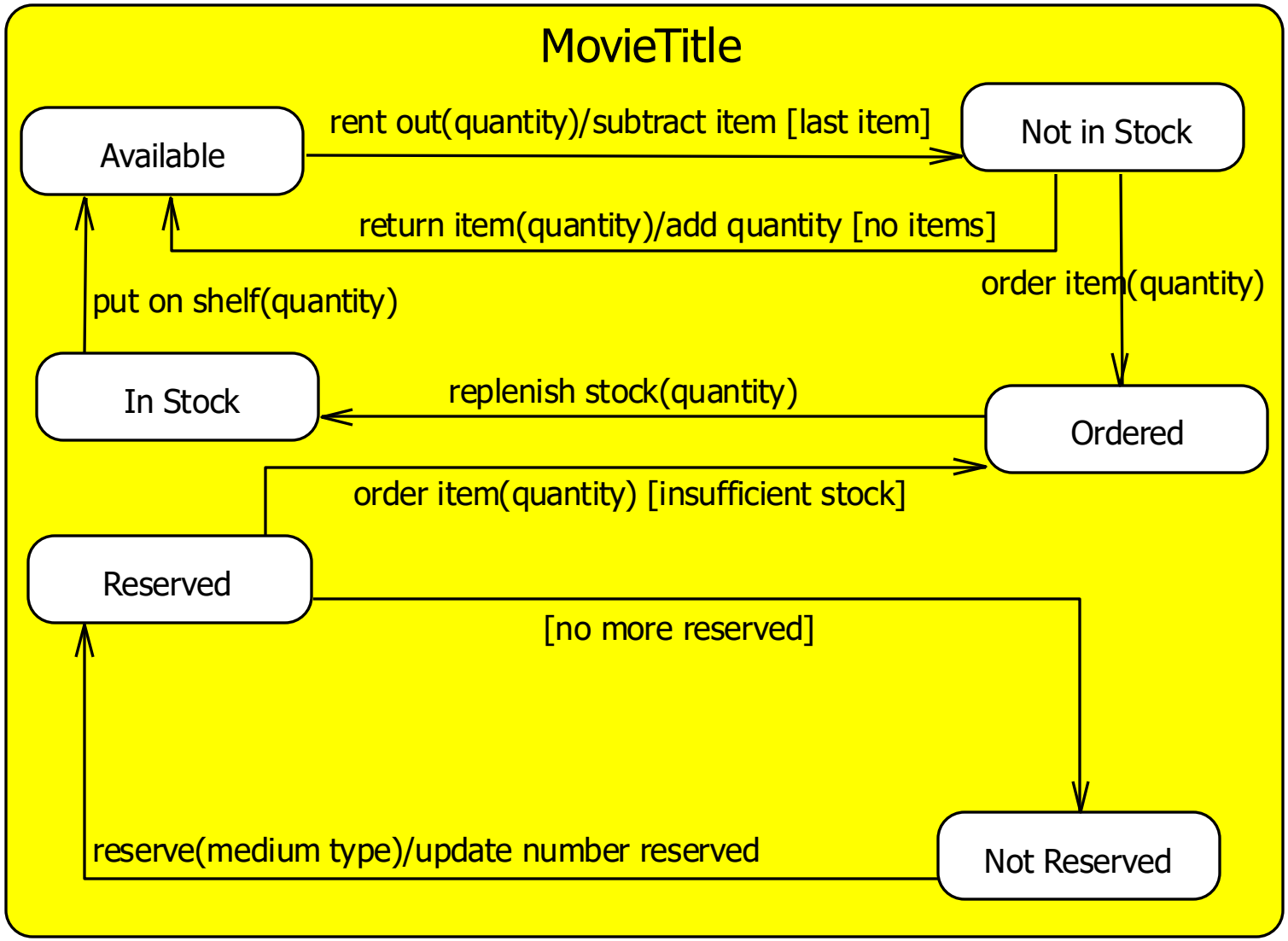
Modello dinamico

- Rappresenta il comportamento dinamico degli oggetti di una singola classe, in termini di stati possibili ed eventi e condizioni che originano transizioni di stato (assieme alle eventuali azioni da svolgere a seguito dell'evento verificatosi)
- Fa uso del formalismo *State Diagrams*



- Viene costruito per ogni classe di controllo (per le quali è interessante descrivere il comportamento dinamico)
- Usato principalmente per applicazioni scientifiche e *real-time* (meno frequentemente nello sviluppo di applicazioni gestionali)

Example B.5 – Video Store

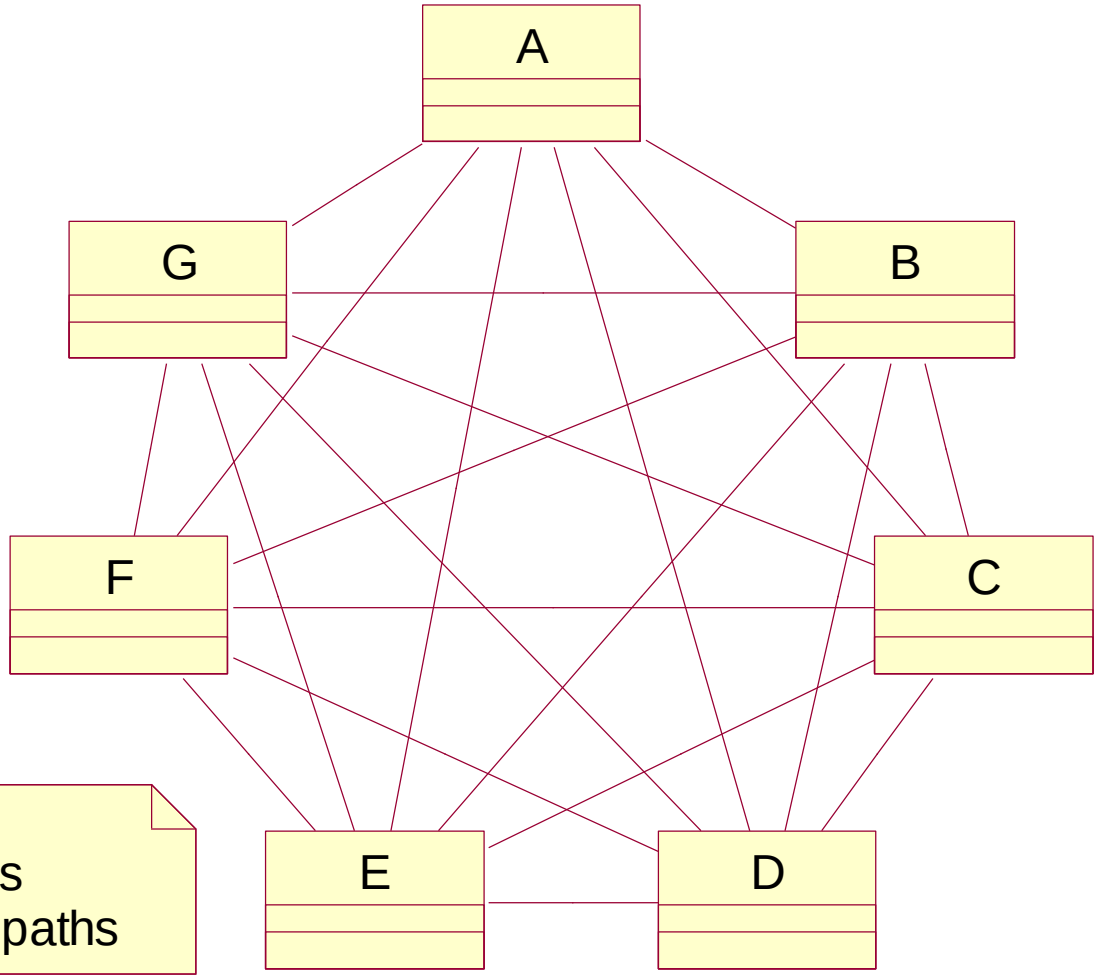


Gestione della complessità nei modelli di OOA

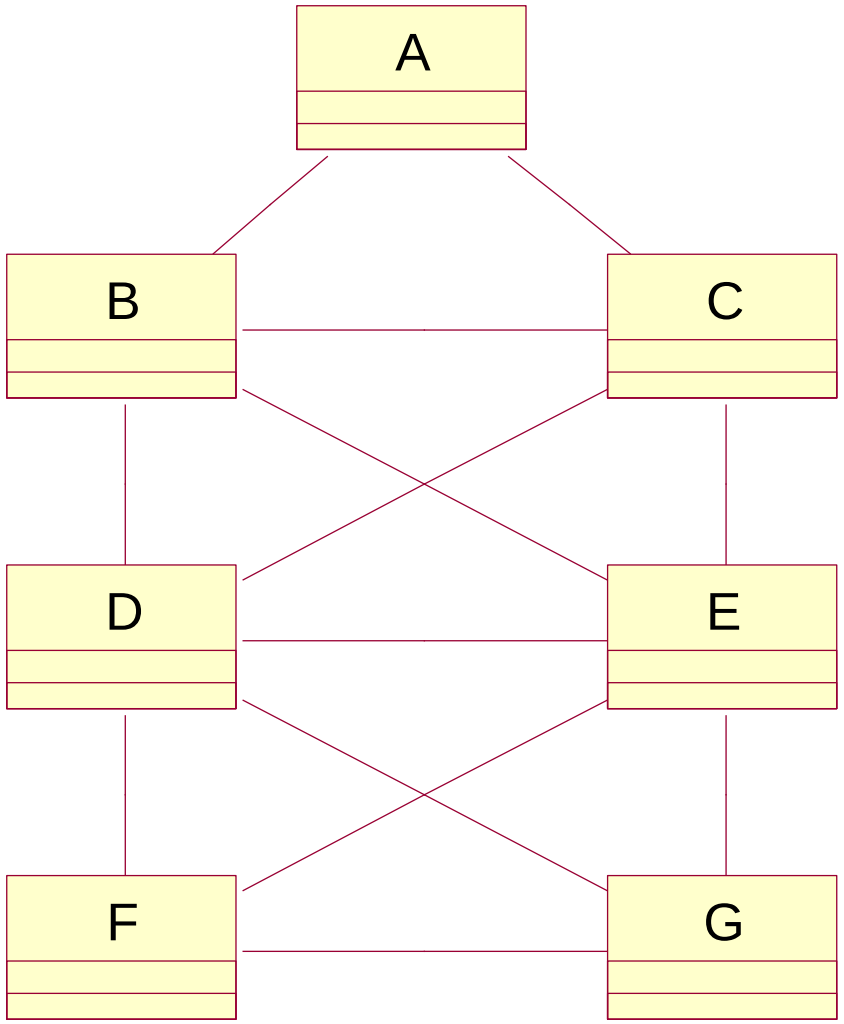
- Nella fase di OOA per sistemi software di grandi dimensioni occorre gestire in modo opportuno l'intrinseca complessità dei modelli
- Le associazioni tra classi nel modello dei dati formano complesse *reti di interconnessione*, in cui i cammini di comunicazione crescono in modo esponenziale con l'aggiunta di nuove classi
- L'introduzione di **gerarchie di classi** permette di ridurre tale complessità da *esponenziale* a *polinomiale*, grazie all'introduzione di opportuni *strati di classi* che vincolano la comunicazione tra classi appartenenti allo stesso strato o a strati adiacenti

Class diagram non stratificato

n(n-1)/2
possibili
connessioni tra
n classi



Class diagram stratificato

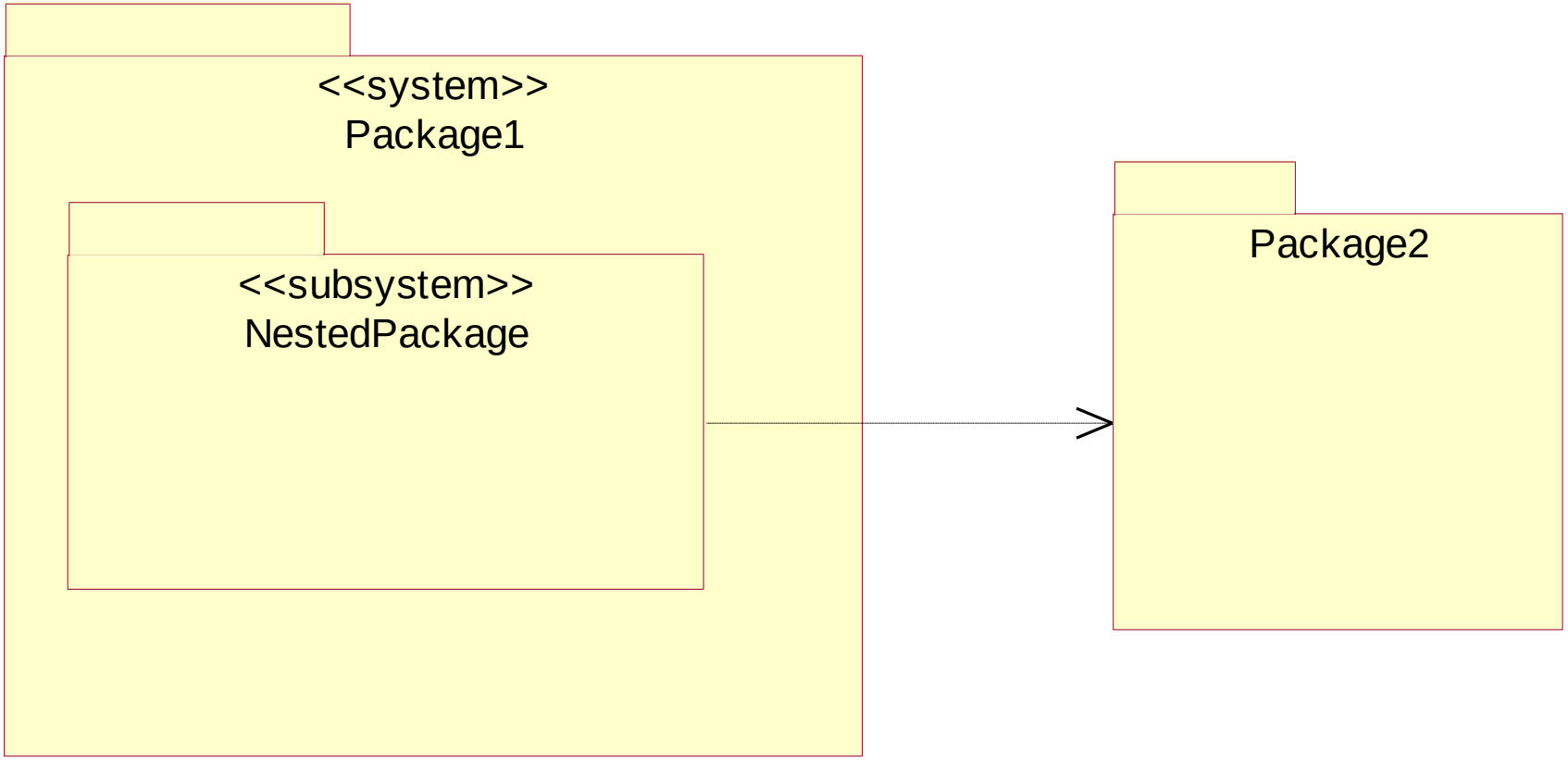


4 layers
13 connections
26 interaction paths

UML Package

- L'UML fornisce la nozione di **package** per rappresentare un gruppo di classi o di altri elementi (ad esempio casi d'uso)
- I *package* possono essere *annidati* (il *package* esterno ha accesso ad ogni classe contenuta all'interno dei *package* in esso annidati)
- Una classe può appartenere ad un solo *package*, ma può comunicare con classi appartenenti ad altri *package*
- La comunicazione tra classi appartenenti a *package* differenti viene controllata mediante dichiarazione di visibilità (*private*, *protected*, o *public*) delle classi all'interno dei *package*

Dipendenza tra *package*



la relazione di dipendenza non è specificata, ma sta ad indicare che eventuali modifiche a *Package2* potrebbero richiedere modifiche di *NestedPackage*

Package diagram

- In UML non esiste il concetto di *package diagram*
- I *package* possono essere creati all'interno di:
  - class diagram
  - use case diagram
- Si possono specificare due tipi di relazioni tra package:
  - **Generalization**

    implica anche *dependency*
  - **Dependency**

    usage dependency, *access* dependency, *visibility* dependency

Approccio BCE

- **Boundary package (BCE)**
  - Descrive classi i cui oggetti gestiscono l'interfaccia tra un attore ed il sistema
  - Le classi catturano una porzione dello stato del sistema e la presentano all'utente in forma *visuale*
- **Control package (BCE)**
  - Descrive classi i cui oggetti intercettano l'input dell'utente e controllano l'esecuzione di uno scenario di funzionamento del sistema
  - Le classi rappresentano azioni ed attività di uno *use case*
- **Entity package (BCE)**
  - Descrive classi i cui oggetti gestiscono l'accesso alle entità fondamentali del sistema
  - Le classi corrispondono alle *strutture dati* gestite dal sistema

Gerarchia di *package BCE*