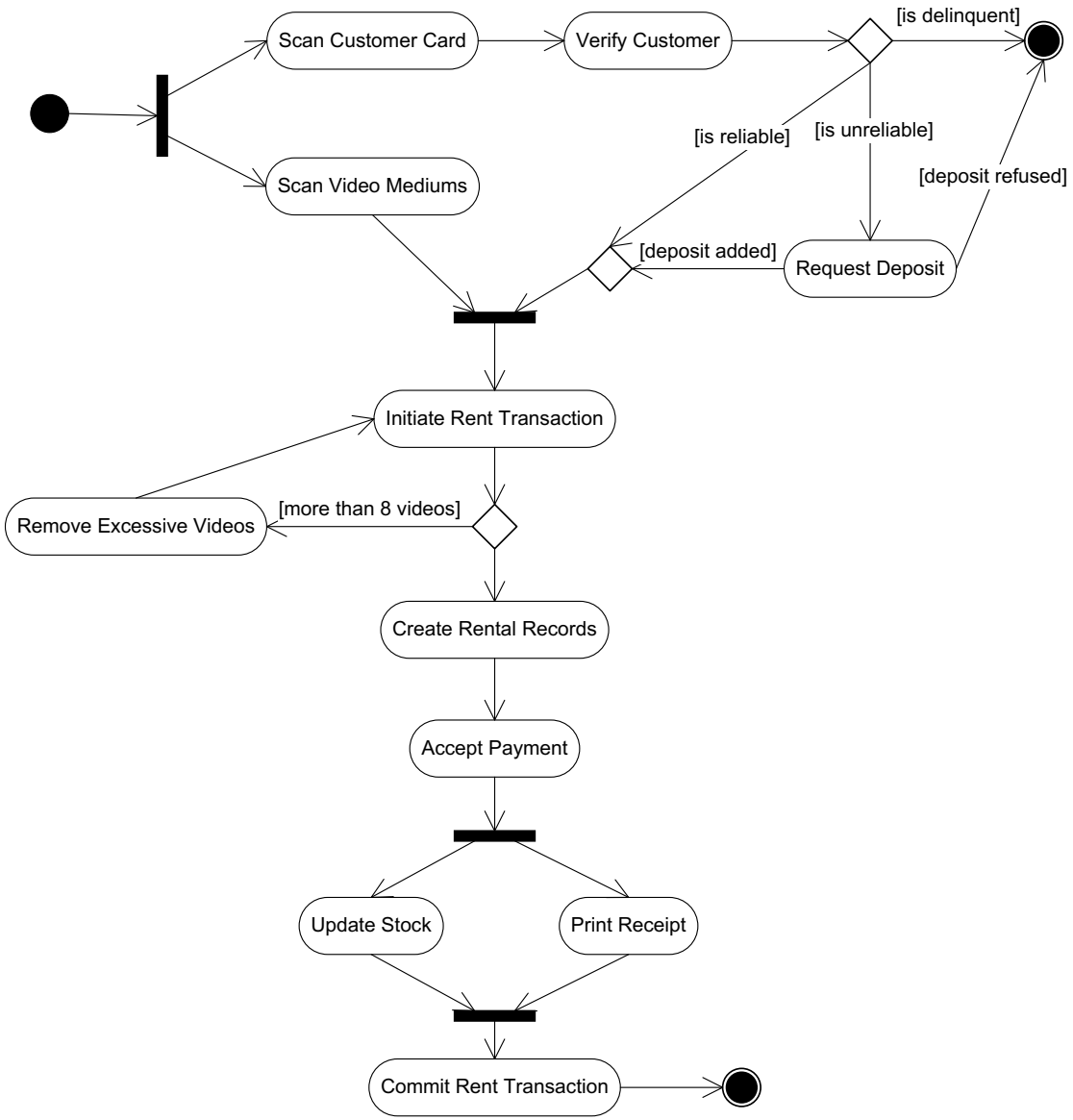
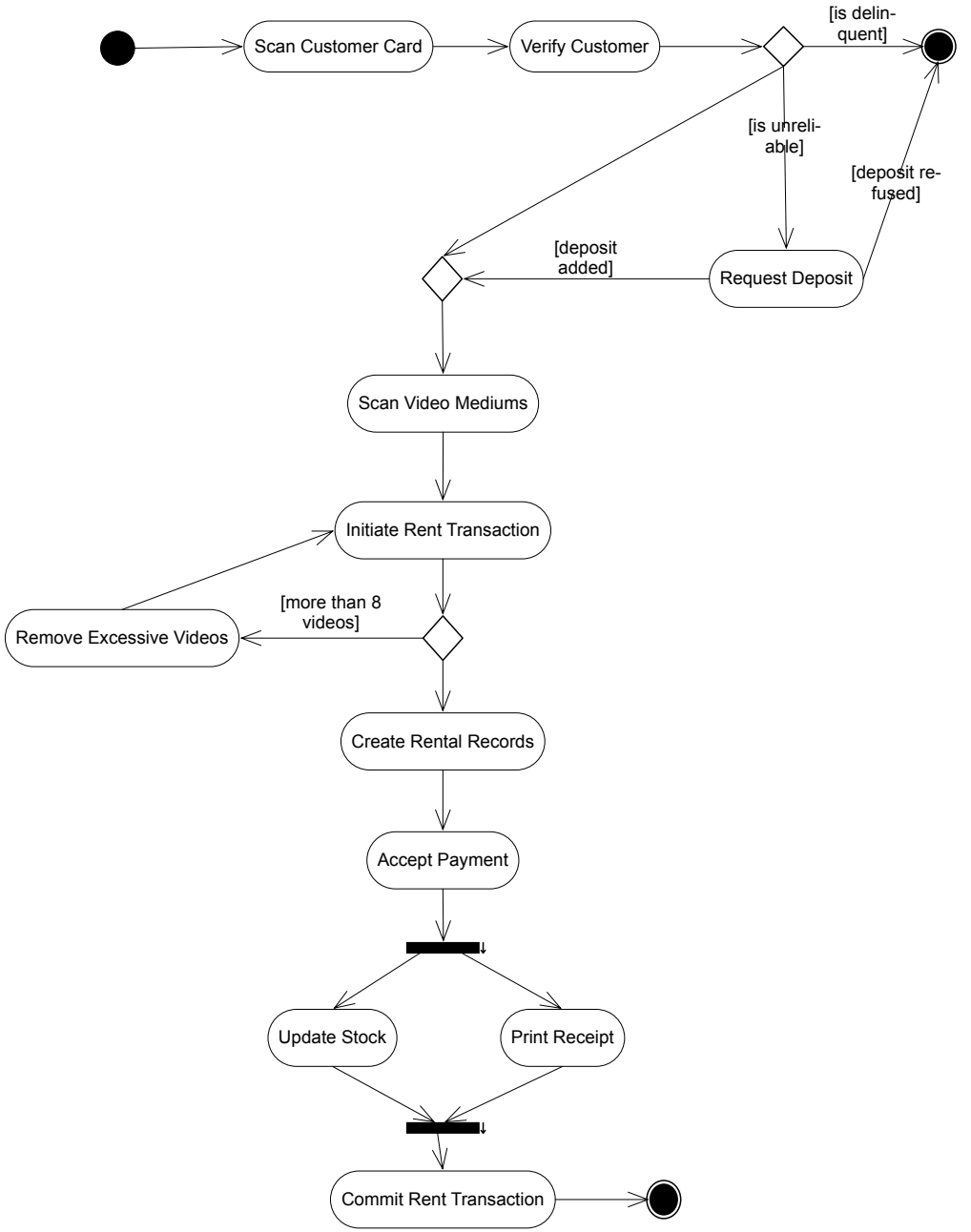
Example B.5 – Video Store (fixed)

Rent Video use case



Example B.5 – Video Store (improved)

Rent Video use case



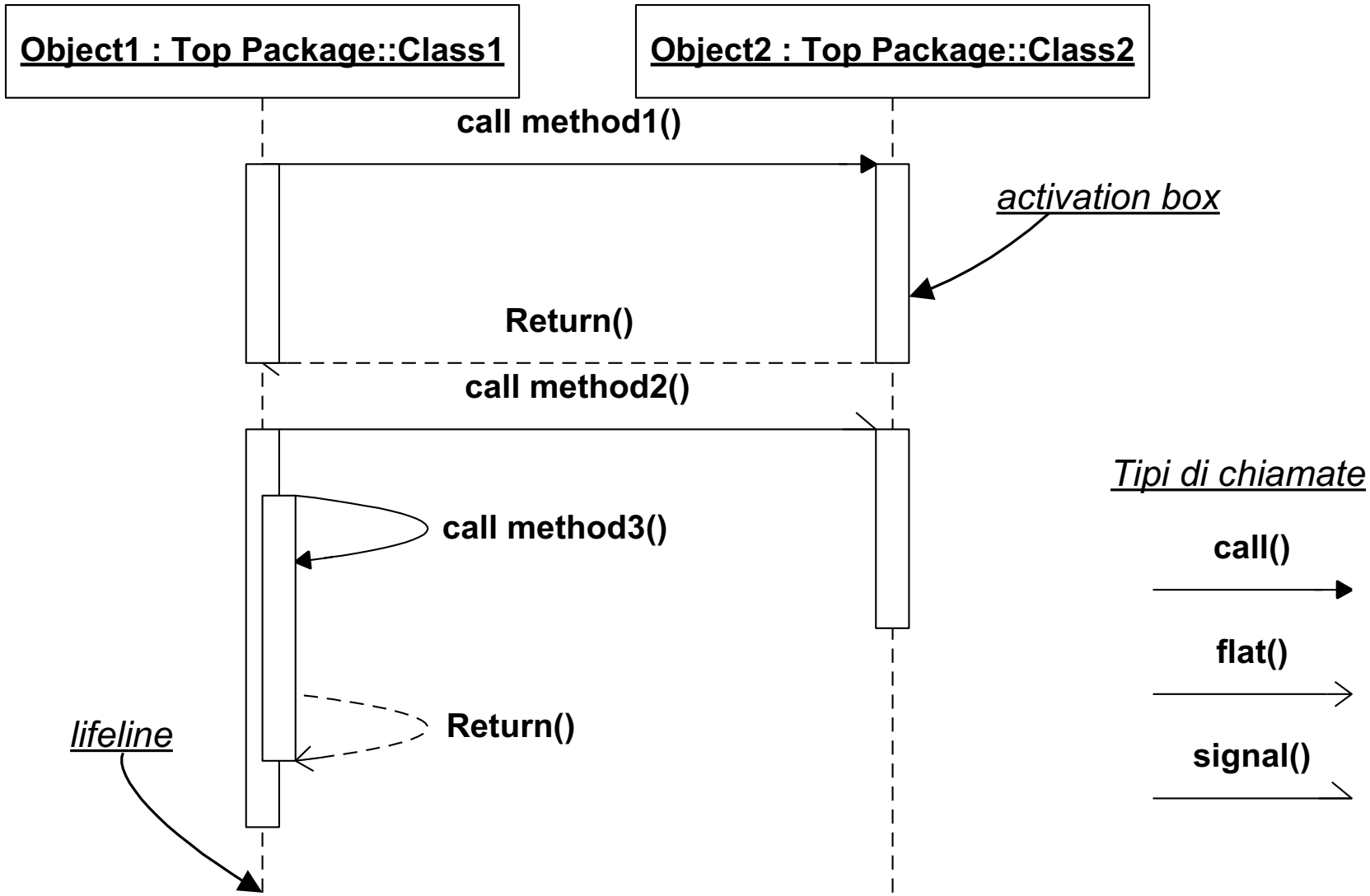
Diagrammi di interazione

- **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

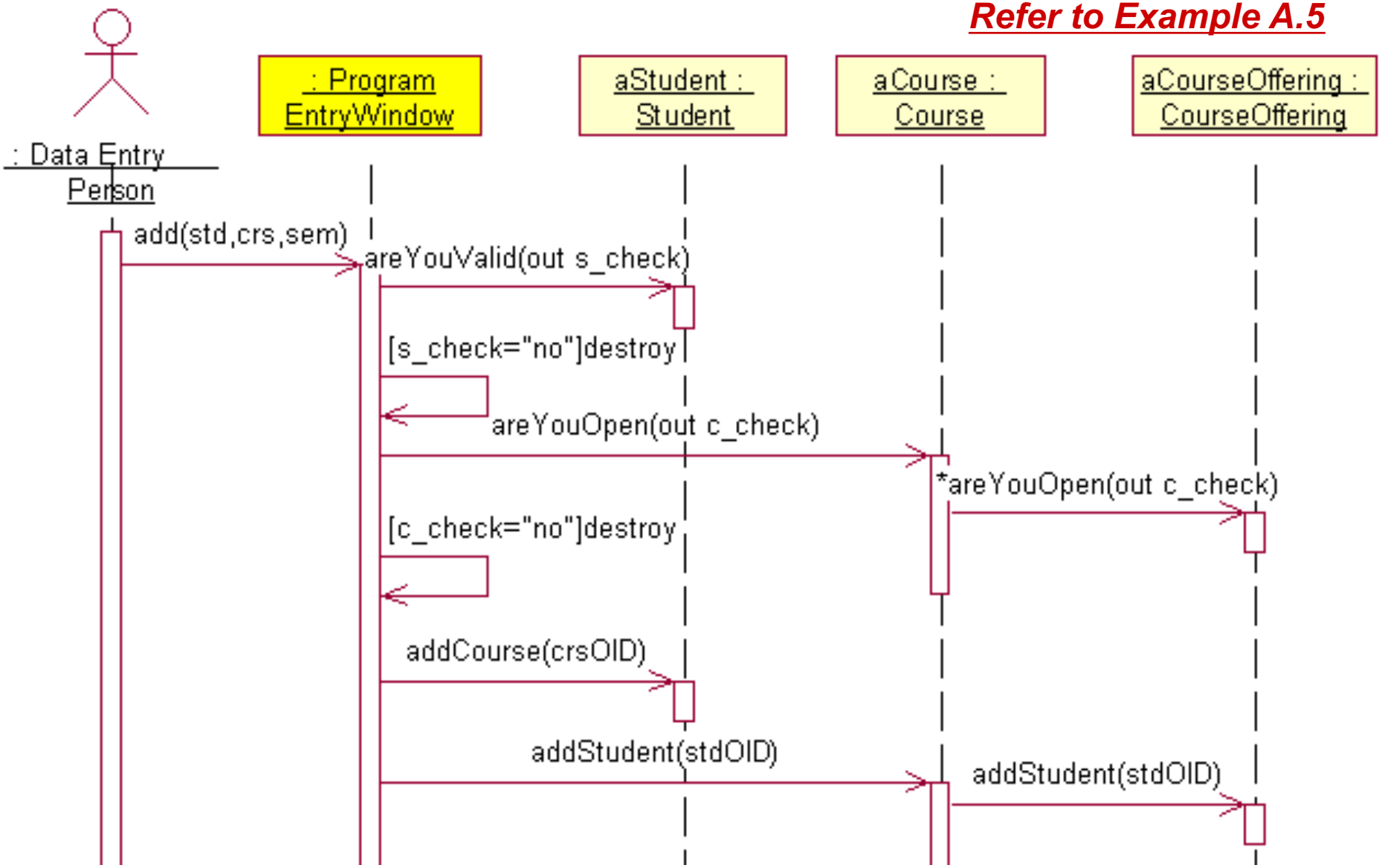
- Le attività dell'*activity diagram* vengono mappate come messaggi (di tipo "richiesta esecuzione attività") in un *sequence diagram*
- Un messaggio può rappresentare:
  - Un ***signal***
    - Denota una **chiamata di tipo asincrono**
    - L'oggetto mittente continua l'esecuzione dopo aver inviato 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

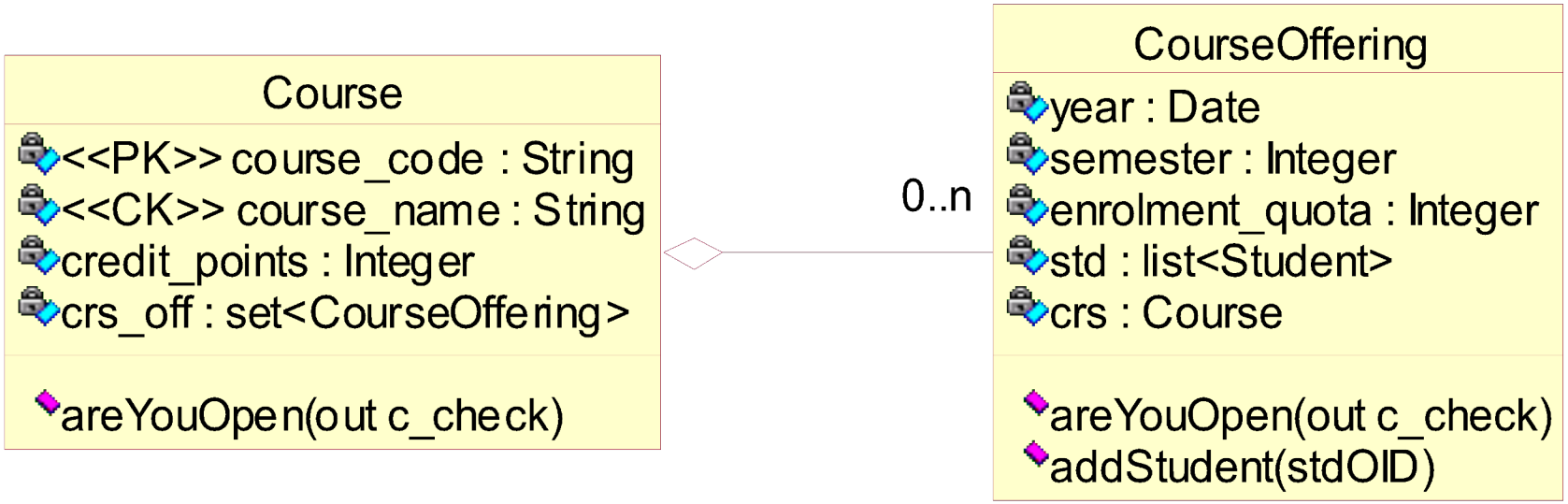
- Definisce l'insieme di operazioni che la classe mette a disposizione delle altre classi
- Durante la fase di OOA, si determina la *signature* dell'operazione, che consiste di:
  - Nome dell'operazione
  - Lista degli argomenti formali
  - Tipo di ritorno
- Durante la fase di OOD, si definisce l'algoritmo 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)

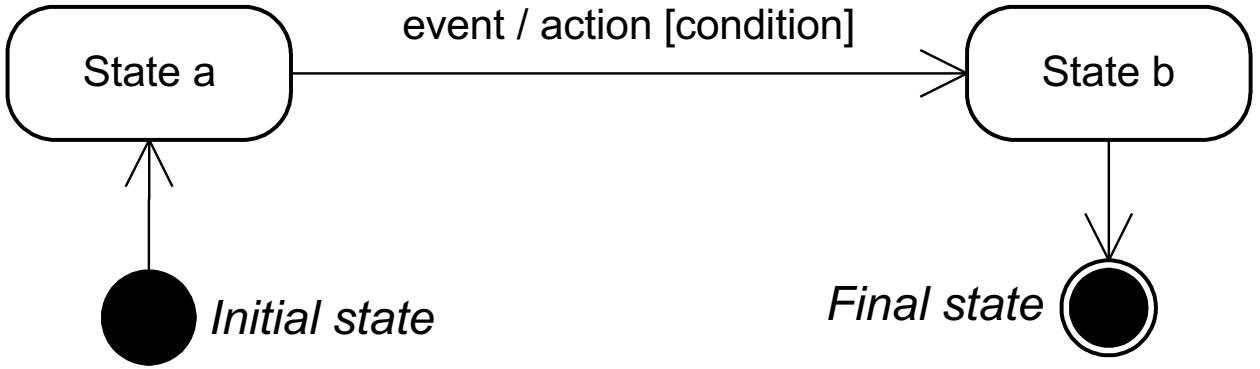
Example A.7 – University Enrolment

- 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`



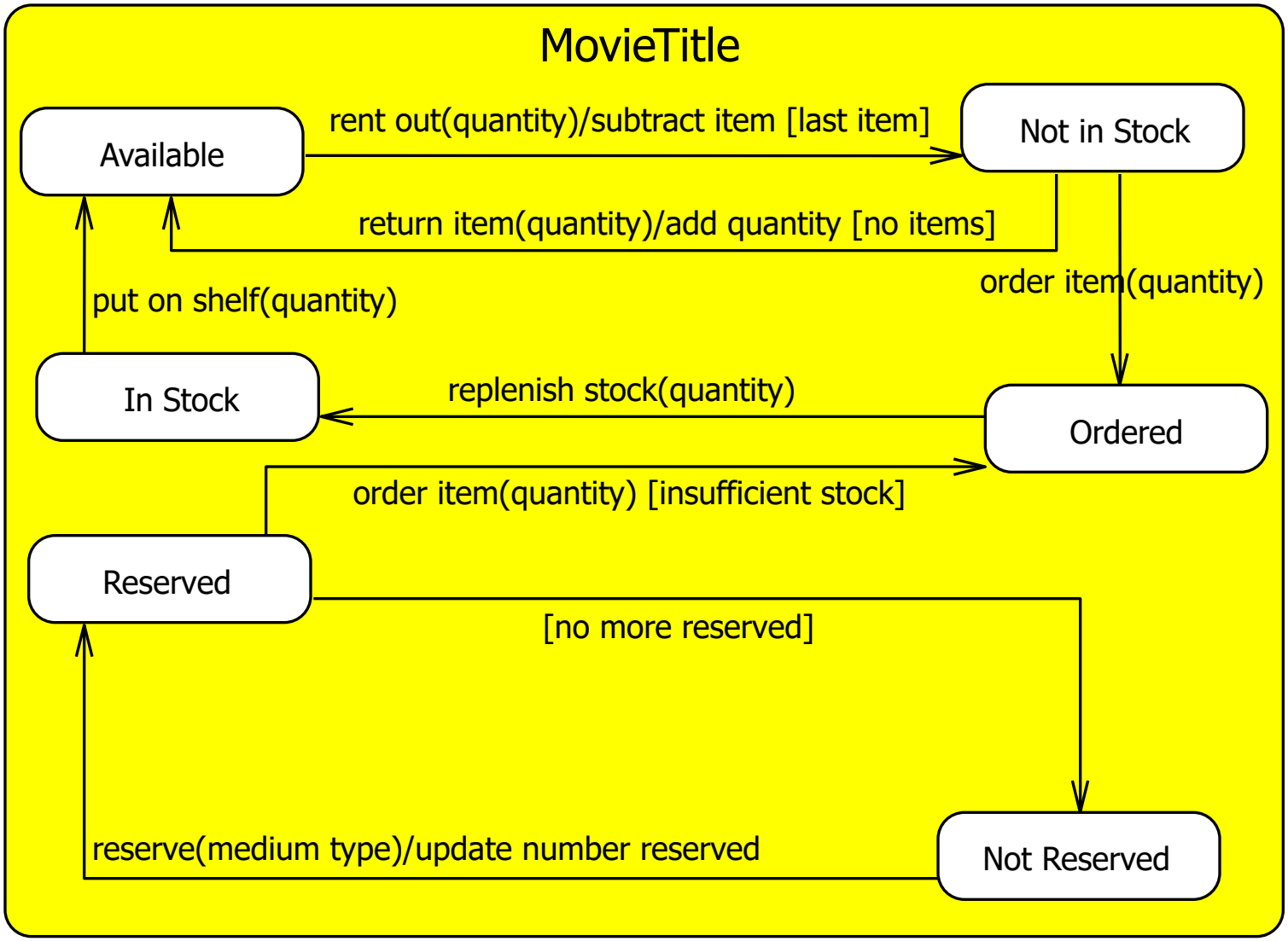
Modello dinamico

- 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

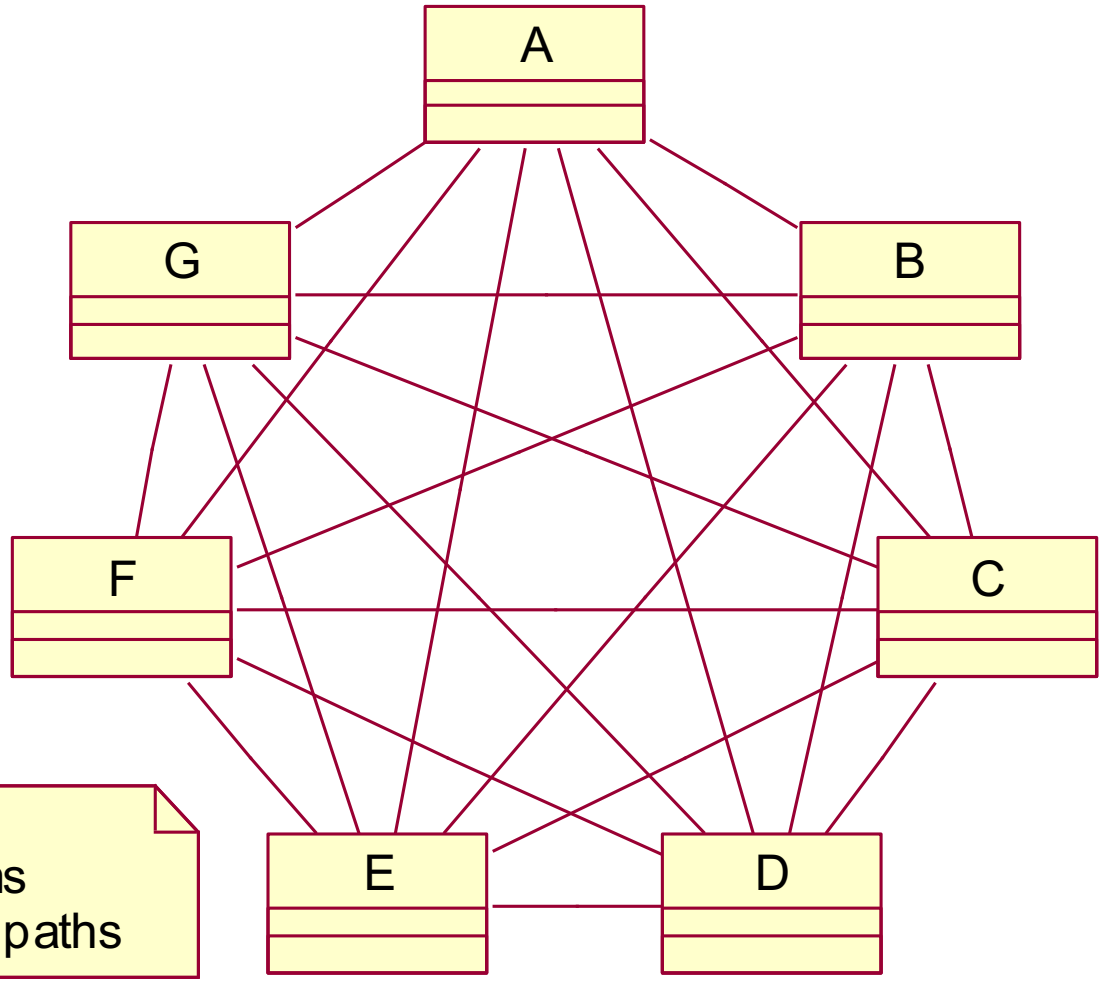


Gestione della complessità nei modelli di OOA

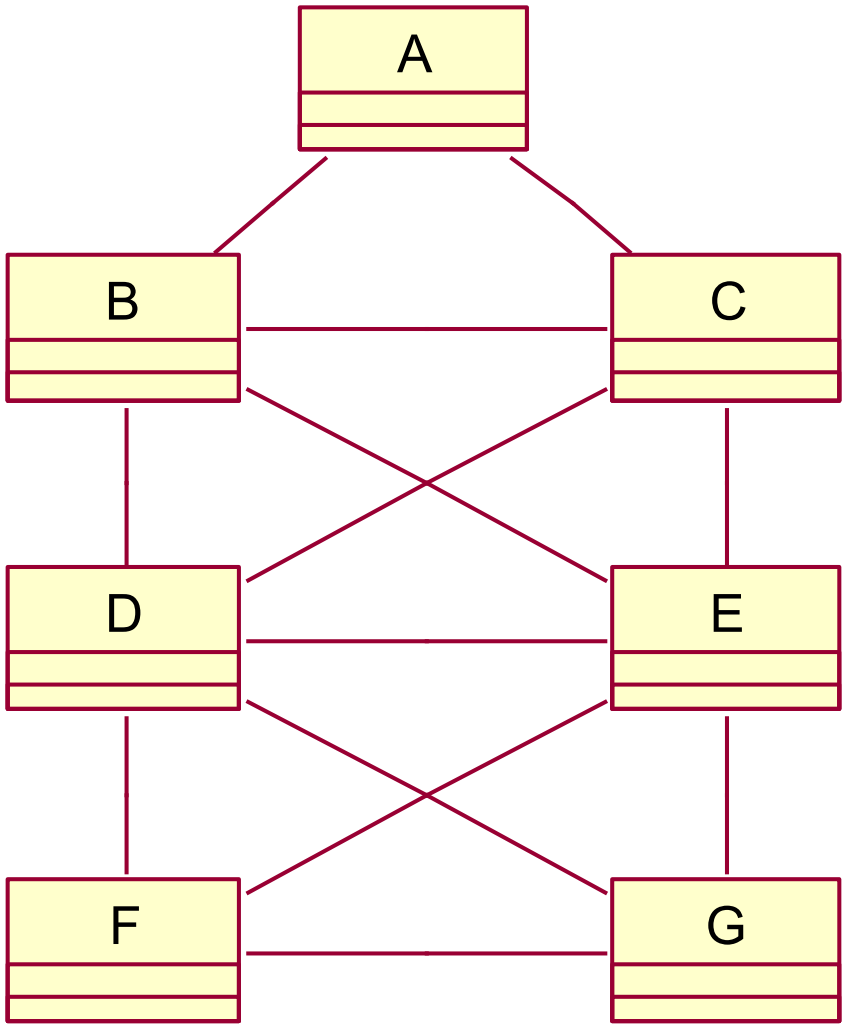
- Nella fase di OOA per sistemi software di grandi dimensioni occorre gestire in modo opportuno l'intrinseca complessità dei modelli
- Le associazioni tra classi nel modello dei dati formano complesse *reti di interconnessione*, in cui i cammini di comunicazione crescono in modo esponenziale con l'aggiunta 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

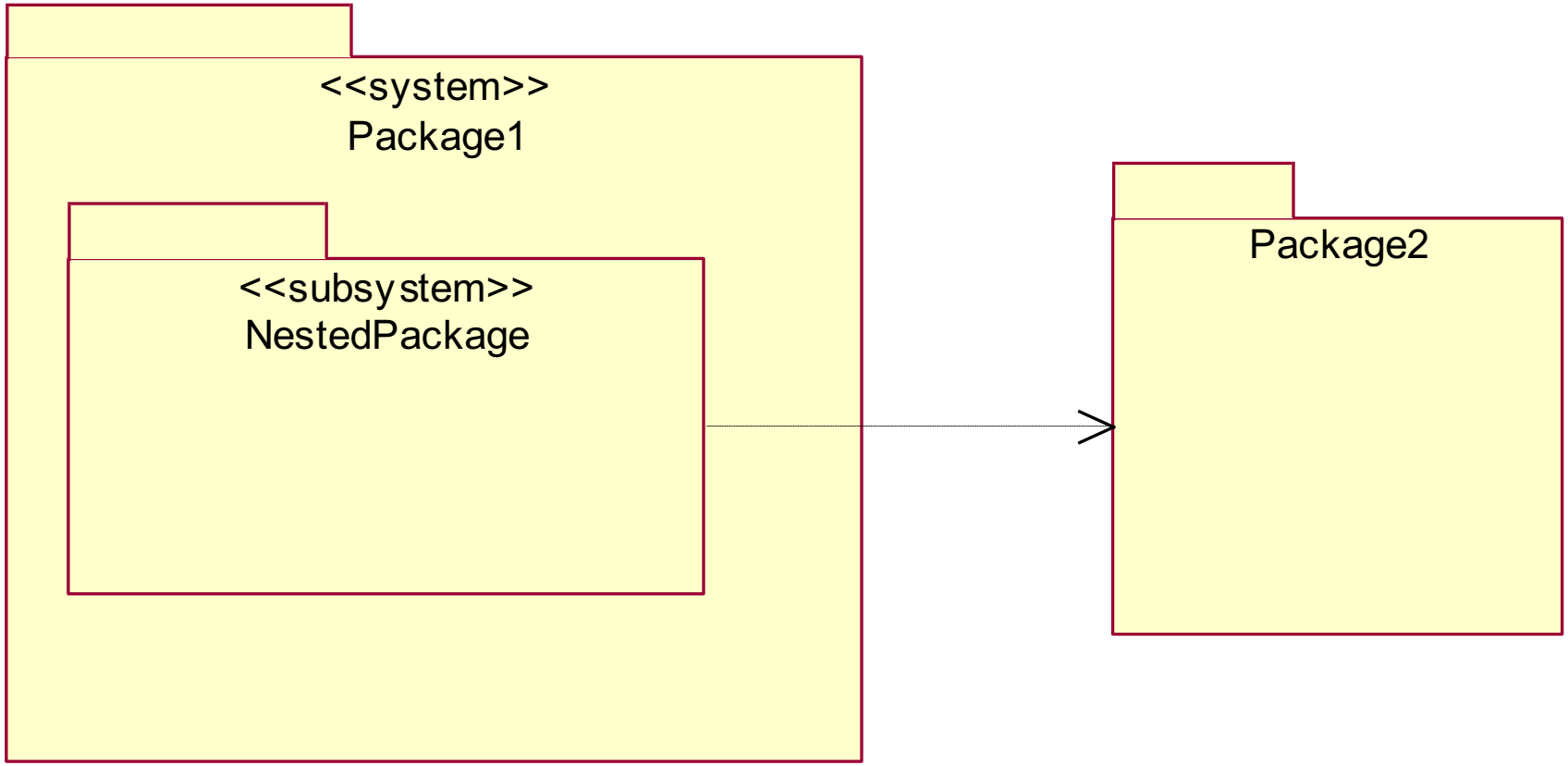


4 layers
13 connections
26 interaction paths

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*